



EDB Postgres™ Advanced Server

リリース **13**

Database Compatibility for Oracle® Developers SQL Guide

2020 年 11 月 29 日

目次

第 1 章	はじめに	1
第 2 章	ALTER DIRECTORY	3
第 3 章	ALTER INDEX	5
第 4 章	ALTER PROCEDURE	7
第 5 章	ALTER PROFILE	9
第 6 章	ALTER QUEUE	13
第 7 章	ALTER QUEUE TABLE	17
第 8 章	ALTER ROLE... IDENTIFIED BY	19
第 9 章	ALTER ROLE - Managing Database Link and DBMS_RLS Privileges	21
第 10 章	ALTER SEQUENCE	25
第 11 章	ALTER SESSION	27
第 12 章	ALTER TABLE	29
第 13 章	ALTER TRIGGER	35
第 14 章	ALTER TABLESPACE	39
第 15 章	ALTER USER... IDENTIFIED BY	41
第 16 章	ALTER USER ROLE... PROFILE MANAGEMENT CLAUSES	43
第 17 章	CALL	47
第 18 章	COMMENT	49
第 19 章	COMMIT	51
第 20 章	CREATE DATABASE	53

第 21 章	CREATE PUBLIC DATABASE LINK	55
第 22 章	CREATE DIRECTORY	69
第 23 章	CREATE FUNCTION	71
第 24 章	CREATE INDEX	79
第 25 章	CREATE MATERIALIZED VIEW	83
第 26 章	CREATE PACKAGE	85
第 27 章	CREATE PACKAGE BODY	89
第 28 章	CREATE PROCEDURE	97
第 29 章	CREATE PROFILE	105
第 30 章	CREATE QUEUE	109
第 31 章	CREATE QUEUE TABLE	111
第 32 章	CREATE ROLE	115
第 33 章	CREATE SCHEMA	117
第 34 章	CREATE SEQUENCE	119
第 35 章	CREATE SYNONYM	123
第 36 章	CREATE TABLE	125
第 37 章	CREATE TABLE AS	135
第 38 章	CREATE TRIGGER	137
第 39 章	CREATE TYPE	149
第 40 章	CREATE TYPE BODY	157
第 41 章	CREATE USER	161
第 42 章	CREATE USER ROLE... PROFILE MANAGEMENT CLAUSES	163
第 43 章	CREATE VIEW	167
第 44 章	DELETE	169
第 45 章	DROP DATABASE LINK	173

第 46 章	DROP DIRECTORY	175
第 47 章	DROP FUNCTION	177
第 48 章	DROP INDEX	179
第 49 章	DROP PACKAGE	181
第 50 章	DROP PROCEDURE	183
第 51 章	DROP PROFILE	185
第 52 章	DROP QUEUE	187
第 53 章	DROP QUEUE TABLE	189
第 54 章	DROP SYNONYM	191
第 55 章	DROP ROLE	193
第 56 章	DROP SEQUENCE	195
第 57 章	DROP TABLE	197
第 58 章	DROP TABLESPACE	199
第 59 章	DROP TRIGGER	201
第 60 章	DROP TYPE	203
第 61 章	DROP USER	205
第 62 章	DROP VIEW	207
第 63 章	EXEC	209
第 64 章	GRANT	211
64.1	データベースオブジェクトの付与	213
64.2	ロールの付与	215
64.3	システム権限の付与	217
第 65 章	INSERT	219
第 66 章	LOCK	223
第 67 章	REVOKE	227
第 68 章	ROLLBACK	231

第 69 章	ROLLBACK TO SAVEPOINT	233
第 70 章	SAVEPOINT	235
第 71 章	SELECT	237
71.1	FROM 句	239
71.2	WHERE 句	241
71.3	GROUPBY 句	241
71.4	HAVING 句	242
71.5	選択リスト	243
71.6	UNION 句	243
71.7	INTERSECT 句	244
71.8	マイナス句	244
71.9	CONNECTBY 句	245
71.10	ORDERBY 句	245
71.11	DISTINCT一意句	247
71.12	FORUPDATE 句	248
第 72 章	SET CONSTRAINTS	249
第 73 章	SET ROLE	251
第 74 章	SET TRANSACTION	253
第 75 章	TRUNCATE	255
第 76 章	UPDATE	257
第 77 章	Conclusion	261
索引		263

第 1 章

はじめに

このガイドでは、AdvancedServer でサポートされている Oracle データベースと互換性のある SQL コマンドの概要を説明します。このセクションの SQL コマンドは、Oracle データベースと AdvancedServer データベースの両方で機能します。

次の点に注意してください。

- AdvancedServer は、ここにリストされていない他のコマンドをサポートしています。これらのコマンドには、Oracle に相当するものがない場合や、OracleSQL コマンドと同様または同じ機能を提供しますが、構文は異なります。
- このセクションの SQL コマンドは、各コマンドで使用可能な完全な構文、オプション、および機能を必ずしも表しているわけではありません。ほとんどの場合、Oracle データベースと互換性のない構文、オプション、および機能は、コマンドの説明と構文から省略されています。
- AdvancedServer のドキュメントセットには、Oracle データベースと互換性のないコマンド機能が記載されています。

第 2 章

ALTER DIRECTORY

Name

ALTER DIRECTORY -- は、*CREATEDIRECTORY* コマンドを使用して作成されたディレクトリの所有者を変更します。

Synopsis

```
ALTER DIRECTORY <name> OWNER TO <rolename>
```

Description

ALTER DIRECTORY ...OWNER TO コマンドは、ディレクトリの所有者を変更します。このコマンドを実行するには、スーパーユーザー権限が必要です。ディレクトリの新しい所有者にもスーパーユーザー権限が必要です。

Parameters

name

変更するディレクトリの名前。

rolename

ディレクトリを所有する所有者の名前。

Examples

次の例は、所有権の変更を示しています。bob と carol はスーパーユーザーです。bob はディレクトリ EMPDIR の現在の所有者です。

```
SELECT * FROM all_directories where directory_name = 'EMPDIR' order
by 1,2,3;
  owner | directory_name | directory_path
-----+-----+-----
  bob   | EMPDIR         | /path
(1 row)
```

ディレクトリ EMPDIR の所有権を carol に変更するには:

```
ALTER DIRECTORY EMPDIR OWNER TO carol;
ALTER DIRECTORY

SELECT * FROM all_directories where directory_name = 'EMPDIR' order by
1,2,3;
  owner | directory_name | directory_path
-----+-----+-----
  carol | EMPDIR         | /path
(1 row)
```

ディレクトリの所有権が変更され、carol に付与されます。

See Also

[ディレクトリの作成、ディレクトリの削除](#)

第 3 章

ALTER INDEX

Name

ALTER INDEX -既存のインデックスを変更します。

Synopsis

AdvancedServer は、Oracle データベースと互換性のある ALTER INDEX コマンドの 3 つのバリエーションをサポートしています。最初のバリエーションを使用して、インデックスの名前を変更します。

```
ALTER INDEX <name> RENAME TO <new_name>
```

ALTER INDEX コマンドの 2 番目のバリエーションを使用して、インデックスを再構築します。

```
ALTER INDEX <name> REBUILD
```

ALTER INDEX コマンドの 3 番目のバリエーションを使用して、PARALLEL または NOPARALLEL 句を設定します。

```
ALTER INDEX <name> { NOPARALLEL | PARALLEL [ <integer> ] }
```

Description

ALTER INDEX は既存のインデックスの定義を変更します。RENAME 句は、インデックスの名前を変更します。REBUILD 句はインデックスを再構築し、インデックスの古いコピーをインデックスのテーブルに基づいて更新されたバージョンに置き換えます。

REBUILD 句は、PostgreSQL の REINDEX コマンドを呼び出します。REBUILD 句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-reindex.html>

PARALLEL 句は、インデックスの再構築を並列化するために使用できるインデックスの並列度を設定します。

NOPARALLEL 句は、デフォルト値を使用するように並列処理をリセットします。reloptions は parallel_workers パラメータを 0 として表示します。

ALTER INDEX は保存データには影響しません。

Parameters

name

既存のインデックスの名前（スキーマで修飾されている可能性があります）。

new_name

インデックスの新しい名前。

PARALLEL

並列度を指定するには、PARALLEL 句を含めます。parallel_workers パラメータを、インデックスの再構築の並列度に設定します。PARALLEL を指定しても、ある程度の並列処理が提供されない場合、サーバーはデフォルトの並列処理を実行します。

NOPARALLEL

NOPARALLEL を指定すると、並列処理がデフォルト値にリセットされます。

integer

integer は、並列度（インデックスの再構築時に使用される parallel_workers の数）を示します。

Examples

インデックスの名前を name_idx から empname_idx に変更するには:

```
ALTER INDEX name_idx RENAME TO empname_idx;
```

empname_idx という名前のインデックスを再構築するには:

```
ALTER INDEX empname_idx REBUILD;
```

次の例では、empname_idx インデックスの並列度を 7 に設定しています。

```
ALTER INDEX empname_idx PARALLEL 7;
```

See Also

CREATEINDEX、*DROPINDEX*

第 4 章

ALTER PROCEDURE

Name

ALTER PROCEDURE

Synopsis

```
ALTER PROCEDURE <procedure_name options> [RESTRICT]
```

Description

ALTER PROCEDURE ステートメントを使用して、プロシージャが SECURITY INVOKER または SECURITY DEFINER であることを指定します。

Parameters

procedure_name

procedure_name は、ストアドプロシージャの（おそらくスキーマ修飾された）名前を指定します。

options は:

[EXTERNAL] SECURITY DEFINER

プロシージャを作成したユーザーの権限でプロシージャを実行するようにサーバーに指示するには、SECURITY DEFINER を指定します。EXTERNAL キーワードは互換性のために受け入れられますが、無視されます。

[EXTERNAL] SECURITY INVOKER

SECURITY INVOKER を指定すると、サーバーは、プロシージャを起動しているユーザーの権限でプロシージャを実行します。EXTERNAL キーワードは互換性のために受け入れられますが、無視されます。

RESTRICT キーワードは互換性のために受け入れられますが、無視されます。

Examples

次のコマンドは、update_balance プロシージャが、プロシージャを呼び出すユーザーの権限で実行されることを指定します。

```
ALTER PROCEDURE update_balance SECURITY INVOKER;
```

See Also

CREATEPROCEDURE、*DROPPROCEDURE*

第 5 章

ALTER PROFILE

Name

ALTER PROFILE –既存のプロファイルを変更する

Synopsis

```
ALTER PROFILE <profile_name> RENAME TO <new_name>;

ALTER PROFILE <profile_name>
    LIMIT {<parameter value>}[...];
```

Description

ALTER PROFILE コマンドを使用して、ユーザー定義のプロファイルを変更します。AdvancedServer は、コマンドの 2 つの形式をサポートしています。

- ALTER PROFILE…RENAME TO を使用して、プロファイルの名前を変更します。
- ALTER PROFILE…LIMIT を使用して、プロファイルに関連付けられた制限を変更します。

LIMIT 句と 1 つ以上のスペース区切りの parameter/value ペアを含めて AdvancedServer によって適用されるルールを指定するか、ALTER PROFILE…RENAME TO を使用してプロファイルの名前を変更します。

Parameters

profile_name

プロファイルの名前。

new_name

new_name は、プロファイルの新しい名前を指定します。

parameter

parameter は、プロファイルによって制限される属性を指定します。

value

value はパラメータの制限を指定します。

AdvancedServer は、parameter ごとに以下に示す value をサポートしています。

FAILED_LOGIN_ATTEMPTS は、PASSWORD_LOCK_TIME で指定された期間、サーバーがユーザーをアカウントからロックアウトする前にユーザーが試行できるログイン試行の失敗回数を指定します。サポートされる値は次のとおりです。

- 0 より大きい INTEGER 値。
- DEFAULT - DEFAULT プロファイルで指定された FAILED_LOGIN_ATTEMPTS の値。
- DEFAULT - DEFAULT プロファイルで指定された FAILED_LOGIN_ATTEMPTS の値。
- UNLIMITED - 接続ユーザーは、ログイン試行の失敗回数に制限はありません。

PASSWORD_LOCK_TIME は、FAILED_LOGIN_ATTEMPTS のためにロックされたアカウントをサーバーがロック解除するまでに経過する必要がある時間の長さを指定します。サポートされる値は次のとおりです。

- 0 以上の NUMERIC 値。日の小数部分を指定するには、小数値を指定します。たとえば、4.5 という値を使用して 4 日、12 時間を指定します。
- DEFAULT - DEFAULT プロファイルで指定された PASSWORD_LOCK_TIME の値。
- UNLIMITED - データベースのスーパーユーザーによって手動でロックが解除されるまで、アカウントはロックされます。

PASSWORD_LIFE_TIME は、ユーザーに新しいパスワードの入力を求めるまでに現在のパスワードを使用できる日数を指定します。PASSWORD_LIFE_TIME 句を使用してパスワードによる有効期限が切れてからロールによる接続が拒否されるまでの日数を指定する場合は、PASSWORD_GRACE_TIME 句を含めます。PASSWORD_GRACE_TIME が指定されていない場合、パスワードは PASSWORD_GRACE_TIME のデフォルト値で指定された日に失効し、ユーザーは新しいパスワードが提供されるまでコマンドの実行を許可されません。サポートされる値は次のとおりです。

- 0 以上の NUMERIC 値。日の小数部分を指定するには、小数値を指定します。たとえば、4.5 という値を使用して 4 日、12 時間を指定します。
- DEFAULT - DEFAULT プロファイルで指定された PASSWORD_LIFE_TIME の値。
- UNLIMITED - パスワードには有効期限がありません。

PASSWORD_GRACE_TIME は、パスワードの有効期限が切れてからユーザーがパスワードの変更を強制されるまでの猶予期間の長さを指定します。猶予期間が終了すると、ユーザーは接続を許可されますが、期限切れのパスワードを更新するまでコマンドの実行は許可されません。サポートされる値は次のとおりです。

- 0 以上の NUMERIC 値。日の小数部分を指定するには、小数値を指定します。たとえば、4.5 という値を使用して 4 日、12 時間を指定します。
- DEFAULT - DEFAULT プロファイルで指定された PASSWORD_GRACE_TIME の値。
- UNLIMITED - 猶予期間は無限です。

PASSWORD_REUSE_TIME は、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日数を指定します。PASSWORD_REUSE_TIME および PASSWORD_REUSE_MAX パラメータは、一緒に使用することを目的としています。これらのパラメーターの一方に有限値を指定し、他方が UNLIMITED の場合、古いパスワードは再利用できません。両方のパラメーターが UNLIMITED に設定されている場合、パスワードの再利用に関する制限はありません。サポートされる値は次のとおりです。

- 0 以上の NUMERIC 値。日の小数部分を指定するには、小数値を指定します。たとえば、4.5 という値を使用して 4 日、12 時間を指定します。
- DEFAULT - DEFAULT プロファイルで指定された PASSWORD_REUSE_TIME の値。
- UNLIMITED - パスワードは制限なしで再利用できます。

PASSWORD_REUSE_MAX は、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を指定します。PASSWORD_REUSE_TIME および PASSWORD_REUSE_MAX パラメータは、一緒に使用することを目的としています。これらのパラメーターの一方に有限値を指定し、他方が UNLIMITED の場合、古いパスワードは再利用できません。両方のパラメーターが UNLIMITED に設定されている場合、パスワードの再利用に関する制限はありません。サポートされる値は次のとおりです。

- 0 以上の INTEGER 値。
- DEFAULT - DEFAULT プロファイルで指定された PASSWORD_REUSE_MAX の値。
- UNLIMITED - パスワードは制限なしで再利用できます。

PASSWORD_VERIFY_FUNCTION はパスワードの複雑さを指定します。サポートされる値は次のとおりです。

- PL/SQL 関数の名前。
- DEFAULT - DEFAULT プロファイルで指定された PASSWORD_VERIFY_FUNCTION の値。
- NULL

PASSWORD_ALLOW_HASHED は、暗号化されたパスワードの使用を許可するかどうかを指定します。値を TRUE に指定すると、システムは、クライアント側でハッシュ計算された暗号化パスワードを指定することにより、ユーザーがパスワードを変更できるようにします。ただし、値を FALSE に指定した場合、パスワードを効果的に検証するにはプレーンテキスト形式で指定する必要があります。そうしないと、サーバーが暗号化されたパスワードを受信した場合にエラーがスローされます。サポートされる値は次のとおりです。

- BOOLEAN 値 TRUE/ON/YES/1 または FALSE/OFF/NO/0。

- DEFAULT – DEFAULT プロファイルで指定された PASSWORD_ALLOW_HASHED の値。

注釈: PASSWORD_ALLOW_HASHED は Oracle 互換のパラメーターではありません。

Examples

次の例では、acctg_profile という名前のプロファイルを変更します。

```
ALTER PROFILE acctg_profile
    LIMIT FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 1;
```

acctg_profile は、ログインロールがサーバーへの接続を試行するときに失敗した接続試行をカウントします。このプロファイルは、ユーザーが3回の試行で正しいパスワードで認証されなかった場合、アカウントが1日間ロックされることを指定します。

次の例では、acctg_profile の名前を payables_profile に変更します。

```
ALTER PROFILE acctg_profile RENAME TO payables_profile;
```

See Also

[プロファイルの作成](#)、[プロファイルの削除](#)

第 6 章

ALTER QUEUE

AdvancedServer には、ALTER QUEUE SQL コマンドに追加の構文（Oracle によって提供されていない）が含まれています。この構文は DBMS_AQADM パッケージに関連して使用できます。

Name

ALTER QUEUE -- を使用すると、スーパーユーザーまたは aq_administrator_role 特権を持つユーザーがキューの属性を変更できます。

Synopsis

このコマンドには 4 つの形式があります。このコマンドの最初の形式は、キューの名前を変更します。

```
ALTER QUEUE <queue_name> RENAME TO <new_name>
```

Parameters

queue_name

既存のキューの名前（オプションでスキーマ修飾）。

RENAME TO

キューの名前を変更するには、RENAME TO 句とキューの新しい名前を含めます。

new_name

キューの新しい名前。

ALTER QUEUE コマンドの 2 番目の形式は、キューの属性を変更します。

```
ALTER QUEUE <queue_name> SET [ ( { <option_name option_value> } [,SET  
<option_name>
```

Parameters

queue_name

既存のキューの名前（オプションでスキーマ修飾）。

SET 句と option_name/option_value ペアを含めて、キューの属性を変更します。

option_name option_value

新しいキューに関連付けられるオプションの名前とオプションの対応する値。重複するオプション名を指定すると、サーバーはエラーを返します。

- option_name が `retries` の場合、デキューを試行できる回数を表す整数を指定します。
- option_name が `retrydelay` の場合、遅延を秒単位で表す倍精度値を指定します。
- option_name が `retention` の場合、保持時間を秒単位で表す倍精度値を指定します。

ALTER QUEUE コマンドの 3 番目の形式を使用して、特定のキューでのエンキューまたはデキューを有効または無効にします。

```
ALTER QUEUE <queue_name> ACCESS { START | STOP } [ FOR { enqueue | dequeue }  
] [ NOWAIT ]
```

Parameters

queue_name

既存のキューの名前（オプションでスキーマ修飾）。

ACCESS

ACCESS キーワードを含めて、特定のキューでのエンキューおよびデキュー、あるいはその両方を有効または無効にします。

START | STOP

START および STOP キーワードを使用して、キューの目的の状態を示します。

FOR enqueue|dequeue

FOR 句を使用して、指定したキューのアクティビティをエンキューまたはデキューの状態を指定しているかどうかを示します。

NOWAIT

NOWAIT キーワードを含めて、キューの状態を変更する前にサーバーが未処理のトランザクションの完了を待たないように指定します。NOWAIT キーワードは、STOP の ACCESS 値を指定する場合にのみ使用できます。NOWAIT が START の ACCESS 値で指定されている場合、サーバーはエラーを返します。

4 番目のフォームを使用して、特定のキューの ADD または DROP コールバックの詳細を指定します。

```
ALTER QUEUE <queue_name> { ADD | DROP } CALL TO <location_name> [ WITH
<callback_option> ]
```

Parameters

queue_name

既存のキューの名前（オプションでスキーマ修飾）。

ADD | DROP

ADD または DROP キーワードを含めて、キューのコールバックの詳細を追加または削除できます。

location_name

location_name は、コールバックプロシージャの名前を指定します。

callback_option

callback_option は context にすることができます。この句を含める場合は RAW 値を指定します。

Examples

次の例では、キューの名前を work_queue_east から work_order に変更します。

```
ALTER QUEUE work_queue_east RENAME TO work_order;
```

次の例では、work_order という名前のキューを変更し、再試行回数を 100 に、再試行間の遅延を 2 秒に、キューがデキューされたメッセージを保持する時間の長さを 10 秒に設定します。

```
ALTER QUEUE work_order SET (retries 100, retrydelay 2, retention 10);
```

次のコマンドは、work_order という名前のキューでのエンキューおよびデキューを有効にします。

```
ALTER QUEUE work_order ACCESS START;
ALTER QUEUE work_order ACCESS START FOR enqueue;
ALTER QUEUE work_order ACCESS START FOR dequeue;
```

次のコマンドは、work_order という名前のキューでのエンキューおよびデキューを無効にします。

```
ALTER QUEUE work_order ACCESS STOP NOWAIT;
ALTER QUEUE work_order ACCESS STOP FOR enqueue;
ALTER QUEUE work_order ACCESS STOP FOR dequeue;
```

See Also

CREATEQUEUE、*DROPQUEUE*

第 7 章

ALTER QUEUE TABLE

AdvancedServer には、ALTER QUEUE SQL コマンドに追加の構文（Oracle によって提供されていない）が含まれています。この構文は DBMS_AQADM パッケージに関連して使用できます。

Name

ALTER QUEUE TABLE-- は既存のキューテーブルを変更します。

Synopsis

ALTER QUEUE TABLE を使用して、既存のキューテーブルの名前を変更します。

```
ALTER QUEUE TABLE <name> RENAME TO <new_name>
```

Description

ALTER QUEUE TABLE を使用すると、スーパーユーザーまたは aq_administrator_role privilege を持つユーザーが既存のキュー表の名前を変更できます。

Parameters

name

既存のキュー表の名前（オプションでスキーマ修飾）。

new_name

キューテーブルの新しい名前。

Examples

キューテーブルの名前を wo_table_east から work_order_table に変更するには:

```
ALTER QUEUE TABLE wo_queue_east RENAME TO work_order_table;
```

See Also

CREATEQUEUETABLE、DROPQUEUETABLE

第 8 章

ALTER ROLE... IDENTIFIED BY

Name

ALTER ROLE -データベースロールに関連付けられたパスワードを変更する

Synopsis

```
ALTER ROLE <role_name> IDENTIFIED BY <password>
        [REPLACE <prev_password>]
```

Description

CREATEROLE 特権のないロールは、このコマンドを使用して自分のパスワードを変更できます。特権のないロールには、プロファイルで PASSWORD_VERIFY_FUNCTION が NULL でない場合、REPLACE 句と以前のパスワードを含める必要があります。REPLACE 句が非スーパーユーザーによって使用されると、サーバーは提供されたパスワードを既存のパスワードと比較し、パスワードが一致しない場合はエラーを発生させます。

データベースのスーパーユーザーは、このコマンドを使用して、任意のロールに関連付けられたパスワードを変更できます。スーパーユーザーに REPLACE 句が含まれている場合、その句は無視されます。前のパスワードと一致しない値はエラーをスローしません。

パスワードを変更する役割に SUPERUSER 属性がある場合、スーパーユーザーはこのコマンドを発行する必要があります。CREATEROLE 属性を持つロールは、このコマンドを使用して、スーパーユーザーではないロールに関連付けられたパスワードを変更できます。

Parameters

role_name

パスワードを変更するロールの名前。

password

ロールの新しいパスワード。

prev_password

ロールの以前のパスワード。

Examples

役割のパスワードを変更するには:

```
ALTER ROLE john IDENTIFIED BY xyRP35z REPLACE 23PJ74a;
```

第 9 章

ALTER ROLE - Managing Database Link and DBMS_RLS Privileges

AdvancedServer には、ALTER ROLE コマンド用の追加の構文（Oracle では提供されていません）が含まれています。この構文は、Oracle データベースと互換性のあるデータベースリンクの作成と削除、およびきめ細かいアクセス制御（DBMS_RLS を使用）に関連する特権を割り当てるときに役立ちます。

CREATE DATABASE LINK

CREATE DATABASE LINK 特権を持つユーザーは、プライベートデータベースリンクを作成できます。次の ALTER ROLE コマンドは、指定されたロールがプライベートデータベースリンクを作成できるようにする AdvancedServer ロールに特権を付与します。

```
ALTER ROLE role_name  
    WITH [CREATEDBLINK | CREATE DATABASE LINK]
```

このコマンドは機能的に同等です:

```
GRANT CREATE DATABASE LINK to role_name
```

次のコマンドを使用して、特権を取り消します。

```
ALTER ROLE role_name  
    WITH [NOCREATEDBLINK | NO CREATE DATABASE LINK]
```

注釈: CREATEDBLINK および NOCREATEDBLINK キーワードは非推奨の構文と見なされる必要があります。CREATE DATABASE LINK と NO CREATE DATABASE LINK の構文オプションを使用することをお勧めします。

CREATE PUBLIC DATABASE LINK

CREATE PUBLIC DATABASE LINK 特権を持つユーザーは、公開データベースリンクを作成できます。次の ALTER ROLE コマンドは、指定されたロールがパブリックデータベースリンクを作成できるようにする AdvancedServer ロールに特権を付与します。

```
ALTER ROLE role_name
    WITH [CREATEPUBLICDBLINK | CREATE PUBLIC DATABASE LINK]
```

このコマンドは機能的に同等です:

```
GRANT CREATE PUBLIC DATABASE LINK to role_name
```

次のコマンドを使用して、特権を取り消します。

```
ALTER ROLE role_name
    WITH [NOCREATEPUBLICDBLINK | NO CREATE PUBLIC DATABASE LINK]
```

注釈: CREATEPUBLICDBLINK および NOCREATEPUBLICDBLINK キーワードは非推奨の構文と見なされる必要があります。CREATE PUBLIC DATABASE LINK と NO CREATE PUBLIC DATABASE LINK の構文オプションを使用することをお勧めします。

DROP PUBLIC DATABASE LINK

DROP PUBLIC DATABASE LINK 特権を持つユーザーは、パブリックデータベースリンクを削除できます。次の ALTER ROLE コマンドは、指定されたロールがパブリックデータベースリンクを削除できるようにする AdvancedServer ロールに特権を付与します。

```
ALTER ROLE role_name
    WITH [DROPPUBLICDBLINK | DROP PUBLIC DATABASE LINK]
```

このコマンドは機能的に同等です:

```
GRANT DROP PUBLIC DATABASE LINK to role_name
```

次のコマンドを使用して、特権を取り消します。

```
ALTER ROLE role_name
    WITH [NODROPPUBLICDBLINK | NO DROP PUBLIC DATABASE LINK]
```

注釈: DROPPUBLICDBLINK および NODROPPUBLICDBLINK キーワードは非推奨の構文と見なされる必要があります。DROP PUBLIC DATABASE LINK と NO DROP PUBLIC DATABASE LINK の構文オプションを使用することをお勧めします。

EXEMPT ACCESS POLICY

EXEMPT ACCESS POLICY 特権を持つユーザーは、きめ細かいアクセス制御（DBMS_RLS）ポリシーから免除されます。これらの特権を持つユーザーは、DBMS_RLS ポリシーによって制約されているテーブル内の行を表示または変更できます。次の ALTER ROLE コマンドは、定義された DBMS_RLS ポリシーから指定されたロールを免除する AdvancedServer ロールに特権を付与します。

```
ALTER ROLE role_name
    WITH [POLICYEXEMPT | EXEMPT ACCESS POLICY]
```

このコマンドは機能的に同等です:

```
GRANT EXEMPT ACCESS POLICY TO role_name
```

次のコマンドを使用して、特権を取り消します。

```
ALTER ROLE role_name
    WITH [NOPOLICYEXEMPT | NO EXEMPT ACCESS POLICY]
```

注釈: POLICYEXEMPT および NOPOLICYEXEMPT キーワードは非推奨の構文と見なされる必要があります。EXEMPT ACCESS POLICY と NO EXEMPT ACCESS POLICY の構文オプションを使用することをお勧めします。

See Also

CREATEROLE、*DROPROLE*、*GRANT*、*REVOKE*、*SETROLE*

第 10 章

ALTER SEQUENCE

Name

ALTER SEQUENCE -- シーケンスジェネレーターの定義を変更します

Synopsis

```
ALTER SEQUENCE <name> [ INCREMENT BY <increment> ]  
  [ MINVALUE <minvalue> ] [ MAXVALUE <maxvalue> ]  
  [ CACHE <cache> | NOCACHE ] [ CYCLE ]
```

Description

ALTER SEQUENCE は、既存のシーケンスジェネレーターのパラメーターを変更します。ALTER SEQUENCE コマンドで特に設定されていないパラメーターは、以前の設定を保持します。

Parameters

name

変更するシーケンスの名前（オプションでスキーマ修飾）。

increment

INCREMENT BY increment 句はオプションです。正の値は昇順、負の値は降順になります。指定しない場合、古い増分値が維持されます。

minvalue

オプションの句 MINVALUE minvalue は、シーケンスが生成できる最小値を決定します。指定しない場合、現在の最小値が維持されます。キーワード NO MINVALUE を使用して、この動作をデフォルトの 1 および $-2^{63}-1$ に戻すことができます。ただし、昇順と降順はそれぞれ異なりますが、この用語は Oracle データベースと互換性があります。

maxvalue

オプションの句 `MAXVALUE maxvalue` は、シーケンスの最大値を決定します。指定しない場合、現在の最大値が維持されます。キーワード `NO MAXVALUE` を使用して、この動作を 2/のデフォルトに戻すことができます:^{sup:63-1} および `-1` は、それぞれ昇順と降順のシーケンスですが、この用語は Oracle データベースと互換性があります。

`cache`

オプションの句 `CACHE cache` は、アクセスを高速化するためにメモリに事前に割り当てて保存するシーケンス番号の数を指定します。最小値は 1 です（一度に生成できる値は 1 つのみ、つまり `NOCACHE`）。指定しない場合、古いキャッシュ値が維持されます。

`CYCLE`

`CYCLE` オプションを使用すると、`maxvalue` または `minvalue` にそれぞれ昇順または降順で到達したときに、シーケンスが折り返されます。制限に達すると、次に生成される数値はそれぞれ `minvalue` または `maxvalue` になります。指定しない場合、古いサイクルの動作が維持されます。キーワード `NO CYCLE` を使用してシーケンスを変更し、リサイクルしないように注意してください。ただし、この用語は Oracle データベースと互換性がありません。

Notes

同じシーケンスから番号を取得する同時トランザクションのブロックを回避するために、`ALTER SEQUENCE` はロールバックされません。変更はすぐに有効になり、元に戻すことはできません。

`ALTER SEQUENCE` が `NEXTVAL` の結果にすぐに影響することはありません。現在のバックエンド以外のバックエンドには、事前に割り当てられた（キャッシュされた）シーケンス値があります。変更されたシーケンスパラメータに気付く前に、すべてのキャッシュされた値を使い果たします。現在のバックエンドはすぐに影響を受けます。

Examples

シーケンスの増分値とキャッシュ値 `serial` を変更します。

```
ALTER SEQUENCE serial INCREMENT BY 2 CACHE 5;
```

See Also

[シーケンスの作成](#)、[シーケンスの削除](#)

第 11 章

ALTER SESSION

Name

ALTER SESSION -- はランタイムパラメータを変更します

Synopsis

```
ALTER SESSION SET <name> = <value>
```

Description

ALTER SESSION コマンドは、ランタイム構成パラメータを変更します。ALTER SESSION は、現在のセッションで使用される値にのみ影響します。これらのパラメーターの一部は、Oracle 構文との互換性のためにのみ提供されており、AdvancedServer のランタイム動作にはまったく影響しません。その他は、対応する AdvancedServer データベースサーバーランタイム構成パラメータを変更します。

Parameters

name

設定可能なランタイムパラメータの名前。使用可能なパラメーターは以下のとおりです。

value

パラメータの新しい値。

Configuration Parameters

次の構成パラメーターは、ALTER SESSION コマンドを使用して変更できます。

NLS_DATE_FORMAT (string)

日付と時刻の値の表示形式と、あいまいな日付入力値の解釈規則を設定します。AdvancedServer datestyle ランタイム構成パラメーターを設定するのと同じ効果があります。

NLS_LANGUAGE (string)

メッセージが表示される言語を設定します。AdvancedServer lc_messages ランタイム構成パラメーターを設定するのと同じ効果があります。

NLS_LENGTH_SEMANTICS (string)

有効な値は BYTE および CHAR です。デフォルトは BYTE です。このパラメーターは構文の互換性のためにのみ提供されており、AdvancedServer には影響しません。

OPTIMIZER_MODE (string)

クエリのデフォルトの最適化モードを設定します。有効な値は ALL_ROWS、CHOOSE、FIRST_ROWS、FIRST_ROWS_10、FIRST_ROWS_100、および FIRST_ROWS_1000 です。デフォルトは CHOOSE です。このパラメーターは AdvancedServer に実装されています。

QUERY_REWRITE_ENABLED (string)

有効な値は TRUE、FALSE、および FORCE です。デフォルトは FALSE です。このパラメーターは構文の互換性のためにのみ提供されており、AdvancedServer では効果がありません。

QUERY_REWRITE_INTEGRITY (string)

有効な値は ENFORCED、TRUSTED、および STALE_TOLERATED です。デフォルトは ENFORCED です。このパラメーターは構文の互換性のためにのみ提供されており、AdvancedServer では効果がありません。

Examples

言語を UTF-8 エンコードで米国英語に設定します。この例では、値 en_US.UTF-8 は AdvancedServer に指定する必要がある形式であることに注意してください。このフォームは、Oracle データベースと互換性がありません。

```
ALTER SESSION SET NLS_LANGUAGE = 'en_US.UTF-8';
```

日付の表示形式を設定します。

```
ALTER SESSION SET NLS_DATE_FORMAT = 'dd/mm/yyyy';
```

第 12 章

ALTER TABLE

Name

ALTER TABLE -- はテーブルの定義を変更します

Synopsis

```
ALTER TABLE <name>
    action [, ...]
ALTER TABLE <name>
    RENAME COLUMN <column> TO <new_column>
ALTER TABLE <name>
    RENAME TO <new_name>
ALTER TABLE <name>
    { NOPARALLEL | PARALLEL [ <integer> ] }
```

ここで、action は次のいずれかです。

```
ADD <column type> [ column_constraint [ ... ] ]
DROP COLUMN <column>
ADD <table_constraint>
DROP CONSTRAINT <constraint_name> [ CASCADE ]
```

Description

ALTER TABLE は既存のテーブルの定義を変更します。いくつかのサブフォームがあります。

ADD column type

このフォームは、CREATE TABLE と同じ構文を使用してテーブルに新しい列を追加します。

DROP COLUMN

このフォームは、テーブルから列を削除します。列に関連するインデックスとテーブル制約も自動的に削除されます。

`ADD table_constraint`

このフォームは、テーブルに新しい制約を追加します。詳細については、[CREATETABLE](#) を参照してください。

`DROP CONSTRAINT`

このフォームは、テーブルの制約を削除します。現在、テーブルの制約には一意の名前を付ける必要はないため、指定した名前に一致する制約が複数存在する場合があります。一致する制約はすべて削除されます。

`RENAME`

RENAME フォームは、テーブル（またはインデックス、シーケンス、ビュー）の名前またはテーブル内の個々の列の名前を変更します。保存されたデータには影響はありません。

PARALLEL 句は、テーブルの並列度を設定します。NOPARALLEL 句は値をデフォルトにリセットします。reloptions は parallel_workers パラメーターを 0 として表示します。

スーパーユーザーには、任意のユーザーのテーブルでトリガーを作成する権限がありますが、ユーザーは自分が所有するテーブルでのみトリガーを作成できます。ただし、テーブルの所有権が変更されると、トリガーを所有するテーブル所有者と一致したときに、トリガーの暗黙オブジェクトの所有権が更新されます。

ALTER TRIGGER ...ON AUTHORIZATION コマンドを使用して、トリガーの暗黙的なオブジェクト所有者を変更できます。詳細については、[ALTERTRIGGER](#) を参照してください。

ALTER TABLE を使用するには、テーブルを所有する必要があります。

Parameters

name

変更する既存のテーブルの名前（スキーマ修飾されている可能性があります）。

column

新規または既存の列の名前。

new_column

既存の列の新しい名前。

new_name

テーブルの新しい名前。

type

新しい列のデータ型。

table_constraint

テーブルの新しいテーブル制約。

constraint_name

削除する既存の制約の名前。

CASCADE

ドロップされた制約に依存するオブジェクトを自動的にドロップします。

PARALLEL

PARALLEL を指定して、並列度を選択します。また、テーブルで並列スキャンを実行するときに parallel_workers パラメータを設定することにより、並列度を指定することもできます。並列度を含めずに PARALLEL を指定すると、インデックスはデフォルトの並列度を使用します。

NOPARALLEL

NOPARALLEL を指定すると、並列処理がデフォルト値にリセットされます。

integer

integer は並列度を示します。これは、テーブルで並列スキャンを実行するために並列操作で使用される parallel_workers の数です。

Notes

ADD COLUMN を呼び出すと、テーブル内のすべての既存の行が列のデフォルト値で初期化されます（DEFAULT 句が指定されていない場合は null）。NULL 以外のデフォルトの列を追加するには、テーブル全体を書き換える必要があります。大きなテーブルの場合、これにはかなりの時間がかかる場合があります。一時的に 2 倍のディスクスペースが必要になります。CHECK または NOT NULL 制約を追加するには、テーブルをスキャンして既存の行が制約を満たすことを確認する必要があります。

DROP COLUMN フォームは列を物理的に削除するのではなく、単に SQL 操作から見えなくします。テーブル内の後続の挿入および更新操作では、列の NULL 値が格納されます。したがって、列の削除は迅速ですが、削除された列が占有していたスペースは再利用されないため、テーブルのディスク上のサイズはすぐには縮小されません。既存の行が更新されると、時間の経過とともにスペースが回収されます。

システムカタログ表の一部を変更することは許可されていません。有効なパラメーターの詳細については、[CREATETABLE](#) を参照してください。

Examples

タイプ VARCHAR2 の列をテーブルに追加するには:

```
ALTER TABLE emp ADD address VARCHAR2(30);
```

テーブルから列を削除するには:

```
ALTER TABLE emp DROP COLUMN address;
```

既存の列の名前を変更するには:

```
ALTER TABLE emp RENAME COLUMN address TO city;
```

既存のテーブルの名前を変更するには:

```
ALTER TABLE emp RENAME TO employee;
```

チェック制約をテーブルに追加するには:

```
ALTER TABLE emp ADD CONSTRAINT sal_chk CHECK (sal > 500);
```

テーブルからチェック制約を削除するには:

```
ALTER TABLE emp DROP CONSTRAINT sal_chk;
```

emp テーブルで並列度を 0 にリセットするには:

```
ALTER TABLE emp NOPARALLEL;
```

次の例では、dept という名前のテーブルを作成し、dept テーブルを変更して dname 列の一意のキーを定義して有効にします。制約 dept_dname_uq は、dname 列を一意のキーとして識別します。USING_INDEX 句は、一意の制約を有効にするために指定されたインデックスステートメントを使用して、テーブル dept にインデックスを作成します。

```
CREATE TABLE dept (  
    deptno          NUMBER(2),  
    dname           VARCHAR2(14),  
    loc             VARCHAR2(13)  
);
```

```
ALTER TABLE dept  
    ADD CONSTRAINT dept_dname_uq UNIQUE(dname)  
        USING INDEX (CREATE UNIQUE INDEX idx_dept_dname_uq ON dept (dname));
```

次の例では、emp という名前のテーブルを作成し、emp テーブルを変更して ename 列の主キーを定義して有効にします。emp_ename_pk 制約は、ename 列を emp テーブルの主キーとして識別します。USING_INDEX 句は、プライマリ制約を有効にするために指定されたインデックスステートメントを使用して、テーブル emp にインデックスを作成します。

```
CREATE TABLE emp (  
    empno          NUMBER(4) NOT NULL,  
    ename          VARCHAR2(10),  
    job            VARCHAR2(9),  
    sal            NUMBER(7,2),
```

(次のページに続く)

(前のページからの続き)

```
deptno      NUMBER(2)
);
```

```
ALTER TABLE emp
  ADD CONSTRAINT emp_ename_pk PRIMARY KEY (ename)
  USING INDEX (CREATE INDEX idx_emp_ename_pk ON emp (ename));
```

See Also

CREATETABLE、*DROPTABLE*

第 13 章

ALTER TRIGGER

Name

ALTER TRIGGER -- はトリガーの定義を変更します

Synopsis

AdvancedServer は、ALTER TRIGGER コマンドの 3 つのバリエーションをサポートしています。最初のバリエーションを使用して、トリガー定義を変更せずに特定のトリガーの名前を変更します。

```
ALTER TRIGGER <name> ON <table_name> RENAME TO <new_name>
```

トリガーが拡張機能に依存している場合は、ALTER TRIGGER コマンドの 2 番目のバリエーションを使用します。拡張機能が削除されると、トリガーも自動的に削除されます。

```
ALTER TRIGGER <name> ON <table_name> DEPENDS ON EXTENSION <extension_name>
```

ALTER TRIGGER コマンドの 3 番目のバリエーションを使用して、トリガーのオブジェクトの所有権を変更します。

```
ALTER TRIGGER <name> ON <table_name> AUTHORIZATION <rolespec>
```

AdvancedServer でサポートされている ALTER TRIGGER コマンドの互換性のない実装の使用については、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/sql-altertrigger.html>

Description

ALTER TRIGGER は、既存のトリガーのプロパティを変更します。トリガーがそのプロパティを変更できるように動作するテーブルを所有する必要があります。

トリガーの暗黙オブジェクトの所有者を変更するには、ALTER TRIGGER ...ON AUTHORIZATION コマンドを使用できます。ALTER TRIGGER ...ON AUTHORIZATION コマンドを実行して、トリガーの暗黙的なオブ

ジェクト所有権を承認後にユーザーに割り当てる権限が必要です。

Parameters

name

変更するトリガーの名前。

table_name

トリガーが動作するテーブルの名前。

rolespec

rolespec は、トリガーオブジェクトの所有者を決定します。

Examples

次の例には、ユーザー bob と carol がスーパーユーザーとして含まれています。ユーザー bob はテーブル emp を所有し、ユーザー carol は emp_sal_trig という名前のトリガーを所有しています。これはテーブル emp で作成されます。

```
SELECT relname, relowner::regrole FROM pg_class WHERE relname = 'emp';
 relname | relowner
-----+-----
 emp      | bob
(1 row)
```

```
SELECT pronomame, proowner::regrole FROM pg_proc WHERE oid = (SELECT tgfooid
FROM pg_trigger WHERE tgname = 'emp_sal_trig') ORDER BY oid;
 pronomame | proowner
-----+-----
 emp_sal_trig_emp | carol
(1 row)
```

テーブル emp の所有権をユーザー bob から新しい所有者 edb に変更するには:

```
ALTER TABLE emp OWNER TO edb;
ALTER TABLE

SELECT relname, relowner::regrole FROM pg_class WHERE relname = 'emp';
 relname | relowner
-----+-----
 emp      | edb
(1 row)
```

テーブルの所有権はユーザー bob から所有者 edb に変更されますが、emp_sal_trig のトリガー所有権は変更されず、ユーザー carol によって所有されます。次に、テーブル emp のトリガー emp_sal_trig を変更し、所有者 edb に許可を付与します。

```
ALTER TRIGGER emp_sal_trig ON emp AUTHORIZATION edb;
ALTER TRIGGER

SELECT pronomame, proowner::regrole FROM pg_proc WHERE oid = (SELECT tgfoaid
FROM pg_trigger WHERE tgname = 'emp_sal_trig') ORDER BY oid;
  pronomame          | proowner
-----+-----
  emp_sal_trig_emp | edb
(1 row)
```

テーブル emp のトリガー所有権 emp_sal_trig が変更され、所有者 edb に付与されます。

See Also

CREATETRIGGER、*DROPTRIGGER*

第 14 章

ALTER TABLESPACE

Name

ALTER TABLESPACE -- は表領域の定義を変更します

Synopsis

```
ALTER TABLESPACE <name> RENAME TO <newname>
```

Description

ALTER TABLESPACE は表領域の定義を変更します。

Parameters

name

既存のテーブルスペースの名前。

newname

テーブルスペースの新しい名前。新しい名前は pg_ で始めることはできません。そのような名前はシステムテーブルスペース用に予約されているためです。

Examples

テーブルスペースの名前を empspace から employee_space に変更します。

```
ALTER TABLESPACE empspace RENAME TO employee_space;
```

See Also

DROPTABLESPACE

第 15 章

ALTER USER... IDENTIFIED BY

Name

ALTER USER -- はデータベースユーザーアカウントを変更します

Synopsis

```
ALTER USER <role_name> IDENTIFIED BY <password> REPLACE <prev_password>
```

Description

CREATEROLE 特権のないロールは、このコマンドを使用して自分のパスワードを変更できます。プロファイルで PASSWORD_VERIFY_FUNCTION が NULL でない場合、非特権ロールには REPLACE 句と以前のパスワードを含める必要があります。REPLACE 句が非スーパーユーザーによって使用されると、サーバーは提供されたパスワードを既存のパスワードと比較し、パスワードが一致しない場合はエラーを発生させます。

データベースのスーパーユーザーは、このコマンドを使用して、任意のロールに関連付けられたパスワードを変更できます。スーパーユーザーに REPLACE 句が含まれている場合、その句は無視されます。前のパスワードと一致しない値はエラーをスローしません。

パスワードを変更する役割に SUPERUSER 属性がある場合、スーパーユーザーはこのコマンドを発行する必要があります。CREATEROLE 属性を持つロールは、このコマンドを使用して、スーパーユーザーではないロールに関連付けられたパスワードを変更できます。

Parameters

role_name

パスワードを変更するロールの名前。

password

ロールの新しいパスワード。

prev_password

ロールの以前のパスワード。

Examples

ユーザーパスワードを変更します。

```
ALTER USER john IDENTIFIED BY xyRP35z REPLACE 23PJ74a;
```

See Also

[ユーザーの作成](#)、[ユーザーの削除](#)

第 16 章

ALTER USER|ROLE... PROFILE MANAGEMENT CLAUSES

Name

ALTER USER|ROLE

Synopsis

```
ALTER USER|ROLE <name> [[WITH] option[...]]
```

option は、次の互換性のある句です。

```
PROFILE <profile_name>  
| ACCOUNT {LOCK|UNLOCK}  
| PASSWORD EXPIRE [AT '<timestamp>']
```

または option は、次の互換性のない句にすることができます。

```
| PASSWORD SET AT '<timestamp>'  
| LOCK TIME '<timestamp>'  
| STORE PRIOR PASSWORD {'<password>' '<timestamp>'} [, ...]
```

AdvancedServer でサポートされている ALTER USER または ALTER ROLE コマンドの管理句については、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-commands.html>

データベース管理者のみが、プロファイル管理を実施する ALTER USER|ROLE 句を使用できます。句は、次の動作を強制します。

PROFILE 句と profile_name を含めて、定義済みプロファイルをロールに関連付けるか、ユーザーに関連付けられている定義済みプロファイルを変更します。

ACCOUNT 句と LOCK または UNLOCK キーワードを含めて、ユーザーアカウントをロックまたはロック解除状態にすることを指定します。

LOCK TIME 'timestamp' 句と日付/時刻値を含めて、指定された時間にロールをロックし、このロールに割り当てられたプロファイルの PASSWORD_LOCK_TIME パラメーターで示される時間にロールをロック解除します。ACCOUNT LOCK 句で LOCK TIME を使用する場合、ACCOUNT UNLOCK 句を使用してデータベースのスーパーユーザーのみがロールをロック解除できます。

AT 'timestamp' キーワードとともに PASSWORD EXPIRE 句を含めて、ロールに関連付けられたパスワードの有効期限が切れる日時を指定します。AT 'timestamp' キーワードを省略すると、パスワードはすぐに期限切れになります。

PASSWORD SET AT 'timestamp' キーワードを含めて、パスワード変更日を指定された時間に設定します。

STORE PRIOR PASSWORD {'password' 'timestamp'} [, ...] 句を含めてパスワード履歴を変更し、新しいパスワードとパスワードが設定された時刻を追加します。

各ログインロールに設定できるプロファイルは 1 つだけです。現在ログインロールに関連付けられているプロファイルを検出するには、DBA_USERS ビューの profile 列を照会します。

Parameters

name

指定されたプロファイルが関連付けられるロールの名前。

password

ロールに関連付けられたパスワード。

profile_name

ロールに関連付けられるプロファイルの名前。

timestamp

条項が施行される日時。timestamp の値を指定する場合は、値を一重引用符で囲みます。

Notes

ALTER USER または ALTER ROLE コマンドの Postgres 互換句については、次の場所にある 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/sql-alterrole.html>

Examples

次のコマンドは、ALTER USER... PROFILE コマンドを使用して acctg という名前のプロファイルを john という名前のユーザーに関連付けます。

```
ALTER USER john PROFILE acctg_profile;
```

次のコマンドは、ALTER ROLE... PROFILE コマンドを使用して acctg という名前のプロファイルを john という名前のユーザーに関連付けます。

```
ALTER ROLE john PROFILE acctg_profile;
```

See Also

CREATEUSER\ROLE...PROFILEMANAGEMENTCLAUSES

第 17 章

CALL

Name

CALL

Synopsis

```
CALL <procedure_name> '(' [<argument_list>] ') '
```

Description

CALL ステートメントを使用して、プロシージャを呼び出します。CALL ステートメントを使用するには、CALL ステートメントが呼び出すプロシージャに対する EXECUTE 特権が必要です。

Parameters

`procedure_name`

`procedure_name` は（オプションでスキーマ修飾された）プロシージャ名です。

`argument_list`

`argument_list` は、プロシージャに必要な引数のコンマ区切りリストを指定します。`argument_list` の各メンバーは、プロシージャで予期される正式な引数に対応することに注意してください。各引数は、IN パラメーター、OUT パラメーター、または INOUT パラメーターです。

注釈: パッケージ関数を呼び出すときは、CALL ステートメントで OUT パラメーターを指定する必要があります。OUT パラメーターは、パッケージのオーバーロード時に INOUT パラメーターとして機能します。

Examples

CALL ステートメントは、プロシージャに必要な引数に応じて、いくつかの形式のいずれかを取ります。

```
CALL update_balance();  
CALL update_balance(1,2,3);
```

第 18 章

COMMENT

Name

COMMENT -- はオブジェクトのコメントを定義または変更します

Synopsis

```
COMMENT ON
{
  TABLE <table_name> |
  COLUMN <table_name.column_name>
} IS '<text>'
```

Description

COMMENT には、データベースオブジェクトに関するコメントが格納されます。コメントを変更するには、同じオブジェクトに対して新しい COMMENT コマンドを発行します。オブジェクトごとに 1 つのコメント文字列のみが保存されます。コメントを削除するには、text に空の文字列（スペースを入れずに 2 つの連続した単一引用符）を指定します。オブジェクトがドロップされると、コメントは自動的にドロップされます。

Parameters

table_name

コメントするテーブルの名前。テーブル名はスキーマ修飾することができます。

table_name.column_name

コメントする table_name 内の列の名前。テーブル名はスキーマ修飾することができます。

text

新しいコメント。

Notes

現在、コメントに対するセキュリティメカニズムはありません。データベースに接続しているユーザーは、そのデータベース内のオブジェクトのすべてのコメントを見ることができます（ただし、所有していないオブジェクトのコメントを変更できるのはスーパーユーザーのみです）。Do not put security-critical information in a comment。

Examples

テーブルにコメントを添付します emp：

```
COMMENT ON TABLE emp IS 'Current employee information';
```

emp テーブルの empno 列にコメントを添付します。

```
COMMENT ON COLUMN emp.empno IS 'Employee identification number';
```

これらのコメントを削除します。

```
COMMENT ON TABLE emp IS '';  
COMMENT ON COLUMN emp.empno IS '';
```

第 19 章

COMMIT

Name

COMMIT -- 現在のトランザクションをコミットします

Synopsis

```
COMMIT [ WORK ]
```

Description

COMMIT は現在のトランザクションをコミットします。トランザクションによって行われたすべての変更は他のユーザーに見えるようになり、クラッシュが発生した場合に永続性が保証されます。

Parameters

WORK

オプションのキーワード-効果はありません。

Notes

トランザクションを中止するには ROLLBACK を使用します。トランザクション内にないときに COMMIT を発行しても害はありません。

注釈: ランタイムスタックに Oracle スタイルの SPL プロシージャがある場合、plpgsql プロシージャで COMMIT を実行するとエラーがスローされます。

Examples

現在のトランザクションをコミットし、すべての変更を永続的にするには:

```
COMMIT;
```

See Also

ROLLBACK、*ROLLBACKTOSAVEPOINT*

第 20 章

CREATE DATABASE

Name

CREATE DATABASE -- 新しいデータベースを作成する

Synopsis

```
CREATE DATABASE <name>
```

Description

CREATE DATABASE は新しいデータベースを作成します。

データベースを作成するには、スーパーユーザーであるか、特別な CREATEDB 権限が必要です。通常、作成者は新しいデータベースの所有者になります。CREATEDB 特権を持つ非スーパーユーザーは、自分が所有するデータベースのみを作成できます。

新しいデータベースは、標準システムデータベース template1 を複製することで作成されます。

Parameters

name

作成するデータベースの名前。

Notes

CREATE DATABASE はトランザクションブロック内では実行できません。

「データベースディレクトリを初期化できませんでした」という行に沿ったエラーは、ほとんどの場合、データディレクトリのアクセス権の不足、ディスク全体、またはその他のファイルシステムの問題に関連しています。

Examples

新しいデータベースを作成するには:

```
CREATE DATABASE employees;
```

第 21 章

CREATE PUBLIC DATABASE LINK

Name

CREATE [PUBLIC] DATABASE LINK -- は新しいデータベースリンクを作成します。

Synopsis

```
CREATE [ PUBLIC ] DATABASE LINK <name>
CONNECT TO { CURRENT_USER |
             <username> IDENTIFIED BY '<password>' }
USING { postgres_fdw '<fdw_connection_string>' |
       [ oci ] '<oracle_connection_string>' }
```

Description

CREATE DATABASE LINK は新しいデータベースリンクを作成します。データベースリンクは、DELETE、INSERT、SELECT または UPDATE コマンド内でリモートデータベース内のテーブルまたはビューへの参照を許可するオブジェクトです。データベースリンクを参照するには、SQL コマンドで参照されるテーブル名またはビュー名に @dblink を追加します。dblink はデータベースリンクの名前です。

データベースリンクはパブリックまたはプライベートにできます。public database link は、すべてのユーザーが使用できるものです。private database link は、データベースリンクの所有者のみが使用できます。PUBLIC オプションを指定すると、パブリックデータベースリンクが作成されます。省略すると、プライベートデータベースリンクが作成されます。

CREATE DATABASE LINK コマンドを指定すると、データベースリンク名と指定した接続属性が pg_catalog.edb_dblink という名前の AdvancedServer システムテーブルに保存されます。特定のデータベースリンクを使用する場合、このデータベースリンクを定義する edb_dblink エントリを含むデータベースは local database と呼ばれます。edb_dblink エントリ内で接続属性が定義されているサーバーとデータベースは remote database と呼ばれます。

ローカルデータベースへの接続中に、データベースリンクへの参照を含む SQL コマンドを発行する必要があります。SQL コマンドが実行されると、適切な認証と接続がリモートデータベースに行われ、@dblink 参照が追加

されたテーブルまたはビューにアクセスします。

Note: データベースリンクを使用して、スタンバイデータベースサーバー内のリモートデータベースにアクセスすることはできません。スタンバイデータベースサーバーは、高可用性、負荷分散、およびレプリケーションに使用されます。

Postgres データベースサーバーの高可用性、負荷分散、およびレプリケーションの詳細については、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/static/high-availability.html>

Note: AdvancedServer12 の場合、CREATE DATABASE LINK コマンドは、Oracle バージョン 10g リリース 2 (10.2)、Oracle バージョン 11g リリース 2 (11.2)、および Oracle バージョン 12c リリース 1 (12.1) に対してテストされ、使用が認定されています。

Note: edb_dblink_oci.rescans GUC は、postgresql.conf ファイルのサーバーレベルで SCROLL または SERIALIZABLE に設定できます。SET コマンドを使用してセッションレベルで設定することもできますが、dblink 接続のキャッシュのため、この設定は既存の dblink 接続には適用されません。

edb_dblink_oci は、SCROLL と SERIALIZABLE の両方のタイプの再スキャンをサポートします。デフォルトでは、SERIALIZABLE に設定されています。SERIALIZABLE に設定すると、edb_dblink_oci は PostgreSQL の REPEATABLE READ に対応する Oracle 側の SERIALIZABLE トランザクション分離レベルを使用します。

- これは、単一の PostgreSQL ステートメントが複数の Oracle クエリにつながる可能性があり、それにより、シリアル化可能な分離レベルを使用して一貫した結果を提供するために必要です。
- 長時間実行される DML トランザクションと同時のテーブル変更（たとえば、ADD、UPDATE、または DELETE ステートメント）により、シリアル化エラーが発生する場合があります。このような障害が発生した場合、OCI は ORA-08177:can't serialize access for this transaction をレポートし、アプリケーションはトランザクションを再試行する必要があります。
- SCROLL の再スキャンは高速ですが、反復するたびに現在の行の位置が 1 にリセットされます。SERIALIZABLE の再スキャンには、SCROLL の再スキャンよりもパフォーマンス上の利点があります。

Parameters

PUBLIC

すべてのユーザーが使用できるパブリックデータベースリンクを作成します。省略すると、データベースリンクはプライベートになり、データベースリンクの所有者のみが使用できます。

name

データベースリンクの名前。

username

リモートデータベースへの接続に使用されるユーザー名。

CURRENT_USER

CURRENT_USER を含めて、AdvancedServer がリモートサーバーへの接続を確立するときにリンクを使用しているロールに関連付けられたユーザーマッピングを使用することを指定します。

password

username のパスワード。

postgres_fdw

外部データラッパー postgres_fdw をリモート AdvancedServer データベースへの接続として指定します。postgres_fdw がデータベースにインストールされていない場合は、CREATE EXTENSION コマンドを使用して postgres_fdw をインストールします。詳細については、<https://www.postgresql.org/docs/current/static/sql-createextension.html> にある PostgreSQLCore ドキュメントの CREATE EXTENSION コマンドを参照してください。

fdw_connection_string

postgres_fdw 外部データラッパーの接続情報を指定します。

oci

リモート Oracle データベースへの接続を指定します。これは AdvancedServer のデフォルトの動作です。

oracle_connection_string

oci 接続の接続情報を指定します。

Notes

非公開データベースリンクを作成するには、CREATE DATABASE LINK 権限が必要です。パブリックデータベースリンクを作成するには、CREATE PUBLIC DATABASE LINK 権限が必要です。

Setting up an Oracle Instant Client for oci-dblink

oci-dblink を使用するには、Oracle インスタントクライアントをダウンロードし、データベースリンクが作成される AdvancedServer データベースを実行しているホストにインストールする必要があります。

インスタントクライアントは、次のサイトからダウンロードできます。

<http://www.oracle.com/technetwork/database/features/instant-client/index-097480.html>

Oracle Instant Client for Linux

次の手順は、AdvancedServer を実行している Linux ホストに適用されます。

AdvancedServer を実行している Linux ホストに libaio ライブラリ（Linux ネイティブの非同期 I/O 機能）が既にインストールされていることを確認してください。

libaio ライブラリは、次のコマンドでインストールできます。

```
yum install libaio
```

ダウンロードした Oracle インスタントクライアントに、バージョン番号のサフィックスのない libclntsh.so という名前のファイルが含まれていない場合、ダウンロードしたバージョンのライブラリファイルを指す libclntsh.so という名前のシンボリックリンクを作成する必要があります。インスタントクライアントディレクトリに移動し、次のコマンドを実行します。

```
ln -s libclntsh.so.version libclntsh.so
```

version は libclntsh.so ライブラリのバージョン番号です。例:

```
ln -s libclntsh.so.12.1 libclntsh.so
```

リモート Oracle データベースへのデータベースリンクを参照する SQL コマンドを実行する場合、AdvancedServer は、Oracle インスタントクライアントライブラリが AdvancedServer ホスト上のどこにあるかを知る必要があります。

LD_LIBRARY_PATH 環境変数には、libclntsh.so ファイルを含む Oracle クライアントインストールディレクトリへのパスを含める必要があります。たとえば、libclntsh.so を含むインストールディレクトリが /tmp/instantclient であると仮定します。

```
export LD_LIBRARY_PATH=/tmp/instantclient:$LD_LIBRARY_PATH
```

注釈: AdvancedServer を起動または再起動するために pg_ctl ユーティリティを実行する場合、LD_LIBRARY_PATH 環境変数設定が有効になっている必要があります。

現在のセッションを pg_ctl を直接呼び出して AdvancedServer を起動または再起動するユーザーアカウント (enterprisedb など) として実行している場合は、pg_ctl を呼び出す前に LD_LIBRARY_PATH を設定してください。

enterprisedb ユーザーアカウントのホームディレクトリの下にある .bash_profile ファイル内に LD_LIBRARY_PATH を設定できます (つまり、~enterprisedb/.bash_profile ファイル内に LD_LIBRARY_PATH を設定します)。これにより、enterprisedb としてログインしたときに LD_LIBRARY_PATH が設定されます。

AdvancedServer を起動または再起動するために systemctl または service コマンドで Linux サービススクリプトを使用している場合、スクリプトが pg_ctl utility を呼び出すときに有効になるように LD_LIBRARY_PATH を設定する必要があります。

たとえば、AdvancedServer の環境変数を設定するには、/etc/systemd/system/edb-as-13.service という名前のファイルを作成できます。ファイル内に /lib/systemd/system/edb-as-13.service を含めます。

LD_LIBRARY_PATH=/tmp/instantclient と仮定すると、次を指定して環境変数を含めることができます。

```
[Service]
Environment=LD_LIBRARY_PATH=/tmp/instantclient:$LD_LIBRARY_PATH
Environment=ORACLE_HOME=/tmp/instantclient
```

次に、次のコマンドを使用して systemd をリロードします。

```
systemctl daemon-reload
```

次に、次のコマンドを使用して AdvancedServer サービスを再起動します。

```
systemctl restart edb-as-13
```

LD_LIBRARY_PATH 設定を含めるために変更する必要がある特定のスクリプトファイルは、AdvancedServer バージョン、インストール先の Linux システム、およびグラフィカルインストーラーまたは RPM パッケージでインストールされたかどうかによって異なります。

EDB Postgres Advanced Server Installation Guide の適切なバージョンを参照して、起動環境に影響を与えるサービススクリプトを特定します。インストールガイドは、次の場所にあります。

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

Oracle Instant Client for Windows

次の手順は、AdvancedServer を実行している Windows ホストに適用されます。

リモート Oracle データベースへのデータベースリンクを参照する SQL コマンドを実行する場合、AdvancedServer は、Oracle インスタントクライアントライブラリが AdvancedServer ホスト上のどこにあるかを知る必要があります。

Windows の PATH システム環境変数を設定して、oci.dll ファイルを含む Oracle クライアントインストールディレクトリを含めます。

別の方法として、postgresql.conf ファイルの oracle_home 構成パラメーターの値を設定できます。oracle_home 構成パラメーターで指定された値は、Windows の PATH 環境変数をオーバーライドします。

postgresql.conf ファイルに oracle_home 構成パラメーターを設定するには、次の行を追加してファイルを編集します。

```
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。例:

```
oracle_home = 'C:/tmp/instantclient_10_2'
```

PATH 環境変数または `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。

注釈: `tnsnames.ora` がフェールオーバーモードで設定されている場合、`client:server` の障害が発生すると、クライアント接続はセカンダリサーバー（通常はバックアップサーバー）と確立されます。その後、プライマリサーバーが再開すると、クライアントは新しいセッションが確立されるまでセカンダリサーバーへの接続を保持します。新しいクライアント接続は、プライマリサーバーと自動的に確立されます。プライマリサーバーとセカンダリサーバーが同期していない場合、セカンダリサーバーへの接続を確立したクライアントと、後でプライマリサーバーに接続したクライアントが異なるデータベースビューを表示する可能性があります。

Examples

Creating an oci-dblink Database Link

次の例は、`CREATE DATABASE LINK` コマンドを使用して、`oci-dblink` 接続を介して `AdvancedServer` のインスタンスを Oracle サーバーに接続するデータベースリンク（`chicago` という名前）を作成する方法を示しています。接続情報は、パスワードが `mypassword` のユーザー `admin` として Oracle にログインするように `AdvancedServer` に指示します。`oci` オプションを含めると、これが `oci-dblink` 接続であることを `AdvancedServer` に伝えます。接続文字列 `'//127.0.0.1/acctg'` は、サーバーのアドレスとデータベースの名前を指定します。

```
CREATE DATABASE LINK chicago
CONNECT TO admin IDENTIFIED BY 'mypassword'
USING oci '//127.0.0.1/acctg';
```

注釈: 接続文字列で（IP アドレスの代わりに）ホスト名を指定できます。

Creating a postgres_fdw Database Link

次の例は、`CREATE DATABASE LINK` コマンドを使用して、`postgres_fdw` 外部データラッパー接続を介して `AdvancedServer` のインスタンスを別の `AdvancedServer` インスタンスに接続するデータベースリンク（`bedford` という名前）を作成する方法を示しています。接続情報は、パスワードが `mypassword` のユーザー `admin` としてログインするように `AdvancedServer` に指示します。`postgres_fdw` オプションを含めると、これが `postgres_fdw` 接続であることを `AdvancedServer` に伝えます。接続文字列 `'host=127.0.0.1 port=5444 dbname=marketing'` は、サーバーのアドレスとデータベースの名前を指定します。

```
CREATE DATABASE LINK bedford
CONNECT TO admin IDENTIFIED BY 'mypassword'
USING postgres_fdw 'host=127.0.0.1 port=5444 dbname=marketing';
```

注釈: 接続文字列で (IP アドレスの代わりに) ホスト名を指定できます。

Using a Database Link

次の例は、AdvancedServer でデータベースリンクを使用して Oracle データベースに接続する方法を示しています。この例では、AdvancedServer サンプルアプリケーションの emp テーブルのコピーが Oracle データベースに作成され、サンプルアプリケーションを含む 2 番目の AdvancedServer データベースクラスターがポート 5443 で接続を受け入れることを前提としています。

ポート 1521 の 127.0.0.1 にある xe という名前の Oracle データベースへの oralink という名前のパブリックデータベースリンクを作成します。ユーザー名 edb とパスワード password で Oracle データベースに接続します。

```
CREATE PUBLIC DATABASE LINK oralink CONNECT TO edb IDENTIFIED BY 'password'
USING '//127.0.0.1:1521/xe';
```

データベースリンク oralink を使用して、Oracle データベースの emp テーブルで SELECT コマンドを発行します。

```
SELECT * FROM emp@oralink;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	17-DEC-80 00:00:00	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81 00:00:00	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81 00:00:00	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81 00:00:00	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81 00:00:00	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81 00:00:00	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81 00:00:00	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87 00:00:00	3000		20
7839	KING	PRESIDENT		17-NOV-81 00:00:00	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81 00:00:00	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87 00:00:00	1100		20
7900	JAMES	CLERK	7698	03-DEC-81 00:00:00	950		30
7902	FORD	ANALYST	7566	03-DEC-81 00:00:00	3000		20
7934	MILLER	CLERK	7782	23-JAN-82 00:00:00	1300		10

(14 rows)

fdwlink という名前のプライベートデータベースリンクを、ポート 5444 で実行されているホスト 192.168.2.22 にある edb という名前の AdvancedServer データベースに作成します。ユーザー名 enterprisedb とパスワード password で AdvancedServer データベースに接続します。

```
CREATE DATABASE LINK fdwlink CONNECT TO enterprisedb IDENTIFIED BY
'password' USING postgres_fdw 'host=192.168.2.22 port=5444 dbname=edb';
```

ローカルの edb_dblink システムテーブルのデータベースリンク oralink および fdwlink の属性を表示します。

```
SELECT lnkname, lnkuser, lnkconnstr FROM pg_catalog.edb_dblink;
```

lnkname	lnkuser	lnkconnstr
oralink	edb	//127.0.0.1:1521/xe
fdwlink	enterprisedb	

(2 rows)

Oracle データベースの emp テーブルと AdvancedServer データベースの dept テーブルを結合します。

```
SELECT d.deptno, d.dname, e.empno, e.ename, e.job, e.sal, e.comm FROM
emp@oralink e, dept@fdwlink d WHERE e.deptno = d.deptno ORDER BY 1, 3;
```

deptno	dname	empno	ename	job	sal	comm
10	ACCOUNTING	7782	CLARK	MANAGER	2450	
10	ACCOUNTING	7839	KING	PRESIDENT	5000	
10	ACCOUNTING	7934	MILLER	CLERK	1300	
20	RESEARCH	7369	SMITH	CLERK	800	
20	RESEARCH	7566	JONES	MANAGER	2975	
20	RESEARCH	7788	SCOTT	ANALYST	3000	
20	RESEARCH	7876	ADAMS	CLERK	1100	
20	RESEARCH	7902	FORD	ANALYST	3000	
30	SALES	7499	ALLEN	SALESMAN	1600	300
30	SALES	7521	WARD	SALESMAN	1250	500
30	SALES	7654	MARTIN	SALESMAN	1250	1400
30	SALES	7698	BLAKE	MANAGER	2850	
30	SALES	7844	TURNER	SALESMAN	1500	0
30	SALES	7900	JAMES	CLERK	950	

(14 rows)

Pushdown for an oci Database Link

oci-dblink を使用してリモート Oracle データベースで SQL ステートメントを実行する場合、外部サーバーで処理のプッシュダウンが発生する特定の状況があります。

Pushdown は、SQL ステートメントが発行されたローカルクライアントではなく、外部（つまり、リモート）サーバーでの処理の発生を示します。データはローカルクライアントに返される前にリモートサーバーで処理されるため、プッシュダウンによりパフォーマンスが向上する可能性があります。

プッシュダウンは、標準の SQL 結合操作（内部結合、左外部結合、右外部結合、および完全外部結合）を持つステートメントに適用されます。結果のデータセットで並べ替えが指定されている場合でも、プッシュダウンは発生

します。

プッシュダウンが発生するためには、特定の基本条件が満たされる必要があります。結合操作に関するテーブルは、同じ外部サーバーに属し、外部サーバーと同じ接続情報（つまり、CREATE DATABASE LINK コマンドで定義された同じデータベースリンク）を使用する必要があります。

SQL ステートメントにプッシュダウンを使用するかどうかを判断するには、EXPLAIN コマンドを使用して実行プランを表示します。

EXPLAIN コマンドについては、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/sql-explain.html>

次の例では、次のように作成されたデータベースリンクを使用しています。

```
CREATE PUBLIC DATABASE LINK oralink CONNECT TO edb IDENTIFIED BY 'password'
USING '///192.168.2.23:1521/xe';
```

次の例は、内部結合の実行計画を示しています。

```
EXPLAIN (verbose, costs off) SELECT d.deptno, d.dname, e.empno, e.ename FROM
dept@oralink d, emp@oralink e WHERE d.deptno = e.deptno ORDER BY 1, 3;
```

QUERY PLAN

```
-----
Foreign Scan
  Output: d.deptno, d.dname, e.empno, e.ename
  Relations: (_dblink_dept_1 d) INNER JOIN (_dblink_emp_2 e)
  Remote Query: SELECT r1.deptno, r1.dname, r2.empno, r2.ename FROM (dept
r1 INNER JOIN emp r2 ON ((r1.deptno = r2.deptno))) ORDER BY r1.deptno
ASC NULLS LAST, r2.empno ASC NULLS LAST
(4 rows)
```

INNER JOIN 操作は外部スキャンセクションで発生することに注意してください。この結合の出力は次のとおりです。

deptno	dname	empno	ename
10	ACCOUNTING	7782	CLARK
10	ACCOUNTING	7839	KING
10	ACCOUNTING	7934	MILLER
20	RESEARCH	7369	SMITH
20	RESEARCH	7566	JONES
20	RESEARCH	7788	SCOTT
20	RESEARCH	7876	ADAMS
20	RESEARCH	7902	FORD
30	SALES	7499	ALLEN

(次のページに続く)

(前のページからの続き)

30		SALES		7521		WARD
30		SALES		7654		MARTIN
30		SALES		7698		BLAKE
30		SALES		7844		TURNER
30		SALES		7900		JAMES

(14 rows)

以下は、左外部結合の実行計画を示しています。

```
EXPLAIN (verbose, costs off) SELECT d.deptno, d.dname, e.empno, e.ename FROM
dept@oralink d LEFT OUTER JOIN emp@oralink e ON d.deptno = e.deptno ORDER
BY 1, 3;
```

QUERY PLAN

```
-----
Foreign Scan
  Output: d.deptno, d.dname, e.empno, e.ename
  Relations: (_dblink_dept_1 d) LEFT JOIN (_dblink_emp_2 e)
  Remote Query: SELECT r1.deptno, r1.dname, r2.empno, r2.ename FROM (dept
r1 LEFT JOIN emp r2 ON ((r1.deptno = r2.deptno))) ORDER BY r1.deptno ASC
NULLS LAST, r2.empno ASC NULLS LAST
(4 rows)
```

この結合の出力は次のとおりです。

deptno		dname		empno		ename
-----+-----+-----+-----						
10		ACCOUNTING		7782		CLARK
10		ACCOUNTING		7839		KING
10		ACCOUNTING		7934		MILLER
20		RESEARCH		7369		SMITH
20		RESEARCH		7566		JONES
20		RESEARCH		7788		SCOTT
20		RESEARCH		7876		ADAMS
20		RESEARCH		7902		FORD
30		SALES		7499		ALLEN
30		SALES		7521		WARD
30		SALES		7654		MARTIN
30		SALES		7698		BLAKE
30		SALES		7844		TURNER
30		SALES		7900		JAMES
40		OPERATIONS				

(15 rows)

次の例は、emp 結合テーブルが同じ外部サーバー上ではなくローカルに存在するため、処理全体がプッシュダウンされない場合を示しています。

```
EXPLAIN (verbose, costs off) SELECT d.deptno, d.dname, e.empno, e.ename FROM
dept@oralink d LEFT OUTER JOIN emp e ON d.deptno = e.deptno ORDER BY 1, 3;
```

QUERY PLAN

```
-----
Sort
  Output: d.deptno, d.dname, e.empno, e.ename
  Sort Key: d.deptno, e.empno
  -> Hash Left Join
    Output: d.deptno, d.dname, e.empno, e.ename
    Hash Cond: (d.deptno = e.deptno)
    -> Foreign Scan on _dblink_dept_1 d
      Output: d.deptno, d.dname, d.loc
      Remote Query: SELECT deptno, dname, NULL FROM dept
    -> Hash
      Output: e.empno, e.ename, e.deptno
      -> Seq Scan on public.emp e
        Output: e.empno, e.ename, e.deptno
(13 rows)
```

この結合の出力は、前の左外部結合の例と同じです。

Creating a Foreign Table from a Database Link

注釈: このセクションで説明する手順は、Oracle データベースと互換性がありません。

データベースリンクを作成したら、このデータベースリンクに基づいて外部テーブルを作成できます。次に、外部テーブルを使用して、データベースリンク構文を使用する代わりに、外部テーブルを参照するリモートテーブルにアクセスできます。データベースリンクを使用するには、SQL コマンドで参照されるテーブル名またはビュー名に @dblink を追加する必要があります。dblink はデータベースリンクの名前です。

この手法は、リモート Oracle アクセスの oci-dblink 接続、またはリモート Postgres アクセスの postgres_fdw 接続に使用できます。

次の例は、リモート Oracle テーブルにアクセスするための外部テーブルの作成を示しています。

まず、前述のようにデータベースリンクを作成します。以下は、Oracle データベースに接続するための oralink という名前のデータベースリンクの作成です。

```
CREATE PUBLIC DATABASE LINK oralink CONNECT TO edb IDENTIFIED BY 'password'
USING '//127.0.0.1:1521/xe';
```

次のクエリは、データベースリンクを示しています。

```
SELECT lnkname, lnkuser, lnkconnstr FROM pg_catalog.edb_dblink;
```

```
lnkname | lnkuser | lnkconnstr
-----+-----+-----
oralink | edb      | //127.0.0.1:1521/xe
(1 row)
```

データベースリンクを作成すると、AdvancedServer は対応する外部サーバーを作成します。次のクエリは、外部サーバーを表示します。

```
SELECT srvname, srvowner, srvfdw, srvtype, srvoptions FROM
pg_foreign_server;
```

```
srvname | srvowner | srvfdw | srvtype | srvoptions
-----+-----+-----+-----+-----
oralink | 10       | 14005 |          | {connstr="//127.0.0.1:1521/xe}
(1 row)
```

外部サーバーの詳細については、次の PostgreSQLCore ドキュメントの CREATE SERVER コマンドを参照してください。

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

次のように外部テーブルを作成します。

```
CREATE FOREIGN TABLE emp_ora (
    empno          NUMERIC(4),
    ename          VARCHAR(10),
    job            VARCHAR(9),
    mgr            NUMERIC(4),
    hiredate       TIMESTAMP WITHOUT TIME ZONE,
    sal            NUMERIC(7,2),
    comm           NUMERIC(7,2),
    deptno         NUMERIC(2)
)
SERVER oralink
OPTIONS (table_name 'emp', schema_name 'edb'
);
```

CREATE FOREIGN TABLE コマンドの次の点に注意してください。

- CREATE FOREIGN TABLE コマンドの最後にある SERVER 句で指定された名前は、外部サーバーの名前です。この例では、pg_foreign_server のクエリの srvname 列に表示される oralink です。
- テーブル名とスキーマ名は、table および schema オプションによって OPTIONS 句で指定されます。
- CREATE FOREIGN TABLE コマンドで指定された列名は、リモートテーブルの列名と一致する必要があります。

- 一般に、CONSTRAINT 句はリモートテーブルで定義されていると見なされるため、外部テーブルでは受け入れられないか強制されない場合があります。

CREATE FOREIGN TABLE コマンドの詳細については、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/sql-createforeigntable.html>

以下は、外部テーブルのクエリです。

```
SELECT * FROM emp_ora;
```

empno	ename	job	mgr	hiredate	sal	comm
deptno						
-----+-----+-----+-----+-----+-----+-----						
+-----+						
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						
7782	CLARK	MANAGER	7839	09-JUN-81 00:00:00	2450.00	
10						
7788	SCOTT	ANALYST	7566	19-APR-87 00:00:00	3000.00	
20						
7839	KING	PRESIDENT		17-NOV-81 00:00:00	5000.00	
10						
7844	TURNER	SALESMAN	7698	08-SEP-81 00:00:00	1500.00	0.00
30						
7876	ADAMS	CLERK	7788	23-MAY-87 00:00:00	1100.00	
20						
7900	JAMES	CLERK	7698	03-DEC-81 00:00:00	950.00	
30						
7902	FORD	ANALYST	7566	03-DEC-81 00:00:00	3000.00	
20						
7934	MILLER	CLERK	7782	23-JAN-82 00:00:00	1300.00	
10						
(14 rows)						

対照的に、次は同じリモートテーブルに対するクエリですが、外部テーブルの代わりにデータベースリンクを使用しています。

```
SELECT * FROM emp@oralink;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	17-DEC-80 00:00:00	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81 00:00:00	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81 00:00:00	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81 00:00:00	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81 00:00:00	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81 00:00:00	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81 00:00:00	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87 00:00:00	3000		20
7839	KING	PRESIDENT		17-NOV-81 00:00:00	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81 00:00:00	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87 00:00:00	1100		20
7900	JAMES	CLERK	7698	03-DEC-81 00:00:00	950		30
7902	FORD	ANALYST	7566	03-DEC-81 00:00:00	3000		20
7934	MILLER	CLERK	7782	23-JAN-82 00:00:00	1300		10

(14 rows)

注釈: 後方互換性の理由から、`USING postgres_fdw`ではなく`USING libpq`を書き込むことも可能です。ただし、`libpq`コネクタには、`postgres_fdw`コネクタに存在する多くの重要な最適化が欠けています。したがって、可能な限り`postgres_fdw`コネクタを使用する必要があります。`libpq`オプションは非推奨であり、将来のAdvancedServerリリースで完全に削除される可能性があります。

See Also

[*DROPDATABASELINK*](#)

第 22 章

CREATE DIRECTORY

Name

CREATE DIRECTORY -- はファイルシステムのディレクトリパスのエイリアスを作成します

Synopsis

```
CREATE DIRECTORY <name> AS '<pathname>'
```

Description

CREATE DIRECTORY コマンドは、ファイルシステムディレクトリのパス名のエイリアスを作成します。このコマンドを使用するには、データベースのスーパーユーザーである必要があります。

エイリアスが UTL_FILE パッケージのプログラムの適切なパラメータとして指定されると、オペレーティングシステムファイルが作成されるか、指定されたエイリアスに対応するディレクトリからアクセスされます。

Parameters

name

ディレクトリのエイリアス名。

pathname

エイリアス名で表される完全修飾ディレクトリパス。CREATE DIRECTORY コマンドは、オペレーティングシステムディレクトリを作成しません。適切なオペレーティングシステムコマンドを使用して、物理ディレクトリを個別に作成する必要があります。

Notes

UTL_FILE パッケージを使用してディレクトリを使用してファイルを作成および/または読み取る場合、オペレーティングシステムのユーザー ID enterprisedb は、ディレクトリに対する適切な読み取りおよび/または書き込み権限を持っている必要があります。

ディレクトリエイリアスは `pg_catalog.edb_dir` システムカタログテーブルに保存されます。 `edb_dir` は Oracle データベースと互換性のあるテーブルではないことに注意してください。

ディレクトリエイリアスは、Oracle データベースと互換性のある Oracle カタログビュー `SYS.ALL_DIRECTORIES` および `SYS.DBA_DIRECTORIES` から表示できます。

`DROP DIRECTORY` コマンドを使用して、ディレクトリエイリアスを削除します。ディレクトリエイリアスが削除されても、対応する物理ファイルシステムディレクトリは影響を受けません。適切なオペレーティングシステムコマンドを使用して、ファイルシステムディレクトリを削除する必要があります。

Linux システムでは、ディレクトリ名の区切り文字はスラッシュ (/) です。

Windows システムでは、ディレクトリ名の区切り文字は、スラッシュ (/) または 2 つの連続したバックスラッシュ (//) として指定できます。

Examples

Linux のディレクトリ `/tmp/empdir` に `empdir` というエイリアスを作成します。

```
CREATE DIRECTORY empdir AS '/tmp/empdir';
```

Windows のディレクトリ `C:/TEMP/EMPDIR` に `empdir` というエイリアスを作成します。

```
CREATE DIRECTORY empdir AS 'C:/TEMP/EMPDIR';
```

すべてのディレクトリエイリアスを表示します。

```
SELECT * FROM pg_catalog.edb_dir;
```

dirname	diowner	dirpath	diracl
empdir	10	C:/TEMP/EMPDIR	

(1 row)

Oracle データベースと互換性のあるビューを使用して、ディレクトリエイリアスを表示します。

```
SELECT * FROM SYS.ALL_DIRECTORIES;
```

owner	directory_name	directory_path
ENTERPRISEDB	EMPDIR	C:/TEMP/EMPDIR

(1 row)

See Also

[*ALTERDIRECTORY*](#)、[*DROPDIRECTORY*](#)

第 23 章

CREATE FUNCTION

Name

CREATE FUNCTION -- 新しい関数を定義します

Synopsis

```
CREATE [ OR REPLACE ] FUNCTION <name> [ (<parameters>) ]
    RETURN <data_type>
    [
        IMMUTABLE
      | STABLE
      | VOLATILE
      | DETERMINISTIC
      | [ NOT ] LEAKPROOF
      | CALLED ON NULL INPUT
      | RETURNS NULL ON NULL INPUT
      | STRICT
      | [ EXTERNAL ] SECURITY INVOKER
      | [ EXTERNAL ] SECURITY DEFINER
      | AUTHID DEFINER
      | AUTHID CURRENT_USER
      | PARALLEL { UNSAFE | RESTRICTED | SAFE }
      | COST <execution_cost>
      | ROWS <result_rows>
      | SET configuration_parameter
        { TO <value> | = <value> | FROM CURRENT }
    ...]
{ IS | AS }
    [ PRAGMA AUTONOMOUS_TRANSACTION; ]
    [ <declarations> ]
BEGIN
    <statements>
END [ <name> ];
```

Description

CREATE FUNCTION は新しい関数を定義します。CREATE OR REPLACE FUNCTION は、新しい関数を作成するか、既存の定義を置き換えます。

スキーマ名が含まれている場合、関数は指定されたスキーマに作成されます。それ以外の場合は、現在のスキーマで作成されます。新しい関数の名前は、同じスキーマ内の同じ入力引数タイプを持つ既存の関数と一致してはなりません。ただし、異なる入力引数タイプの関数は名前を共有する場合があります（これは *overloading* と呼ばれます）。（関数のオーバーロードは *AdvancedServer* の機能です-格納されたスタンドアロン関数のオーバーロードは *Oracle* データベースと互換性がありません。）

既存の関数の定義を更新するには、CREATE OR REPLACE FUNCTION を使用します。この方法で関数の名前または引数の型を変更することはできません（試してみると、実際には新しい別個の関数を作成することになります）。また、CREATE OR REPLACE FUNCTION では、既存の関数の戻り値の型を変更できません。そのためには、関数をドロップして再作成する必要があります。また、OUT パラメータを使用する場合、関数をドロップしない限り、OUT パラメータのタイプを変更することはできません。

関数を作成したユーザーが関数の所有者になります。

Parameters

name

name は関数の識別子です。

parameters

parameters は仮パラメータのリストです。

data_type

data_type は、関数の RETURN ステートメントによって返される値のデータ型です。

declarations

declarations は、変数、カーソル、タイプ、またはサブプログラムの宣言です。サブプログラム宣言が含まれる場合、他のすべての変数、カーソル、および型宣言の後に宣言する必要があります。

statements

statements は *SPL* プログラムステートメントです（BEGIN - END ブロックには EXCEPTION セクションが含まれる場合があります）。

IMMUTABLE

STABLE

VOLATILE

これらの属性は、クエリオプティマイザーに関数の動作を通知します。指定できる選択肢は 1 つだけです。VOLATILE がデフォルトの動作です。

IMMUTABLE は、関数がデータベースを変更できず、同じ引数値が指定されたときに常に同じ結果に達することを示します。データベースの検索は行わず、引数リストに直接存在しない情報を使用しません。この句を含めると、すべて定数の引数を使用した関数の呼び出しを、関数の値にすぐに置き換えることができます。

STABLE は、関数がデータベースを変更できないこと、および単一のテーブルスキャン内で、同じ引数値に対して一貫して同じ結果を返しますが、その結果は SQL ステートメント間で変わる可能性があることを示します。これは、データベースルックアップ、パラメーター変数（現在のタイムゾーンなど）などに依存する機能に適した選択です。

VOLATILE は、1 回のテーブルスキャン内でも関数値が変更される可能性があるため、最適化できないことを示します。副作用のある関数は、結果がかなり予測可能である場合でも、呼び出しが最適化されてしまうのを防ぐために、揮発性に分類する必要があることに注意してください。

DETERMINISTIC

DETERMINISTIC は IMMUTABLE の同義語です。DETERMINISTIC 関数はデータベースを変更できず、同じ引数値が与えられたときに常に同じ結果に到達します。データベースの検索は行わず、引数リストに直接存在しない情報を使用しません。この句を含めると、すべて定数の引数を使用した関数の呼び出しを、関数の値にすぐに置き換えることができます。

[NOT] LEAKPROOF

LEAKPROOF 関数には副作用はなく、関数の呼び出しに使用される値に関する情報は表示されません。

CALLED ON NULL INPUT

RETURNS NULL ON NULL INPUT

STRICT

CALLED ON NULL INPUT（デフォルト）は、引数の一部が NULL の場合にプロシージャが正常に呼び出されることを示します。必要に応じて NULL の値を確認し、適切に対応するのは作成者の責任です。

RETURNS NULL ON NULL INPUT または STRICT は、その引数のいずれかが NULL である場合、常に NULL を返すことを示します。これらの句が指定されている場合、NULL 個の引数がある場合、プロシージャは実行されません。代わりに、NULL の結果が自動的に想定されます。

[EXTERNAL] SECURITY DEFINER

SECURITY DEFINER は、関数を作成したユーザーの権限で実行することを指定します。これがデフォルトです。キーワード EXTERNAL は SQL 準拠に使用できますが、オプションです。

[EXTERNAL] SECURITY INVOKER

SECURITY INVOKER 句は、関数がそれを呼び出すユーザーの権限で実行されることを示します。キーワード EXTERNAL は SQL 準拠に使用できますが、オプションです。

AUTHID DEFINER

AUTHID CURRENT_USER

AUTHID DEFINER 節は [EXTERNAL] SECURITY DEFINER の同義語です。AUTHID 句が省略されている場合、または AUTHID DEFINER が指定されている場合、関数所有者の権限を使用して、データベースオブジェクトへのアクセス権限が決定されます。

AUTHID CURRENT_USER 節は [EXTERNAL] SECURITY INVOKER の同義語です。AUTHID CURRENT_USER が指定されている場合、機能を実行している現在のユーザーの権限を使用してアクセス権限が決定されます。

PARALLEL { UNSAFE | RESTRICTED | SAFE }

PARALLEL 句は、並列順次スキャン（並列モード）の使用を有効にします。パラレルシーケンシャルスキャンは、シリアルシーケンシャルスキャンとは対照的に、クエリ中に複数のワーカーを使用してリレーションを並行してスキャンします。

UNSAFE に設定すると、関数をパラレルモードで実行できません。SQL ステートメントにこのような関数が存在すると、シリアル実行プランが強制されます。これは、PARALLEL 句が省略された場合のデフォルト設定です。

RESTRICTED に設定すると、関数はパラレルモードで実行できますが、実行はパラレルグループリーダーに制限されます。特定のリレーションの資格に並列制限のあるものがある場合、そのリレーションは並列処理のために選択されません。

SAFE に設定すると、関数は制限なしで並列モードで実行できます。

COST execution_cost

execution_cost は、cpu_operator_cost を単位とする、関数の推定実行コストを示す正の数です。関数がセットを返す場合、これは返される行ごとのコストです。値が大きいと、プランナーは必要以上に関数を評価することを避けようとしています。

ROWS result_rows

result_rows は正の数であり、プランナーが関数が返すと予想される行の推定数を示します。これは、関数がセットを返すように宣言されている場合にのみ許可されます。デフォルトの仮定は 1000 行です。

SET configuration_parameter { TO value | = value | FROM CURRENT }

SET 句を使用すると、指定された構成パラメーターが、関数に入るときに指定された値に設定され、関数が終了するときに以前の値に復元されます。SET FROM CURRENT は、パラメータのセッションの現在の値を、関数が入力されたときに適用される値として保存します。

SET 句が関数に付加されている場合、同じ変数に対して関数内で実行される SET LOCAL コマンドの効果はその関数に制限されます。構成パラメーターの以前の値は、関数の終了時に復元されます。通常の SET コマンド（LOCAL なし）は、以前の SET LOCAL コマンドの場合と同様に SET 句をオーバーライドし、現在のトランザクションがロールされない限り、プロシージャ終了後もそのようなコマンドの効果が持続します。

PRAGMA AUTONOMOUS_TRANSACTION

PRAGMA AUTONOMOUS_TRANSACTION は、機能を自律型トランザクションとして設定するディレクティブです。

注釈: STRICT、LEAKPROOF、PARALLEL、COST、ROWS、および SET キーワードは、AdvancedServer の拡張機能を提供するものであり、Oracle ではサポートされていません。

Notes

AdvancedServer は機能のオーバーロードを許可します。つまり、異なる入力（IN、IN OUT）引数データ型がある限り、複数の異なる関数に同じ名前を使用できます。

Examples

関数 emp_comp は 2 つの数値を入力として受け取り、計算された値を返します。SELECT コマンドは、関数の使用法を示しています。

```
CREATE OR REPLACE FUNCTION emp_comp (
    p_sal          NUMBER,
    p_comm         NUMBER
) RETURN NUMBER
IS
BEGIN
    RETURN (p_sal + NVL(p_comm, 0)) * 24;
END;
```

```
SELECT ename "Name", sal "Salary", comm "Commission", emp_comp(sal, comm)
       "Total Compensation" FROM emp;
```

Name	Salary	Commission	Total Compensation
SMITH	800.00		19200.00
ALLEN	1600.00	300.00	45600.00
WARD	1250.00	500.00	42000.00
JONES	2975.00		71400.00
MARTIN	1250.00	1400.00	63600.00
BLAKE	2850.00		68400.00
CLARK	2450.00		58800.00
SCOTT	3000.00		72000.00
KING	5000.00		120000.00
TURNER	1500.00	0.00	36000.00
ADAMS	1100.00		26400.00
JAMES	950.00		22800.00
FORD	3000.00		72000.00
MILLER	1300.00		31200.00

(次のページに続く)

(前のページからの続き)

(14 rows)

関数 `sal_range` は、給与が指定範囲内にある従業員の数のカウントを返します。次の匿名ブロックは、最初の 2 回の呼び出しで引数のデフォルト値を使用して、関数を何度も呼び出します。

```
CREATE OR REPLACE FUNCTION sal_range (
    p_sal_min      NUMBER DEFAULT 0,
    p_sal_max      NUMBER DEFAULT 10000
) RETURN INTEGER
IS
    v_count        INTEGER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM emp
        WHERE sal BETWEEN p_sal_min AND p_sal_max;
    RETURN v_count;
END;

BEGIN
    DBMS_OUTPUT.PUT_LINE('Number of employees with a salary: ' ||
        sal_range);
    DBMS_OUTPUT.PUT_LINE('Number of employees with a salary of at least '
        || '$2000.00: ' || sal_range(2000.00));
    DBMS_OUTPUT.PUT_LINE('Number of employees with a salary between '
        || '$2000.00 and $3000.00: ' || sal_range(2000.00, 3000.00));
END;

Number of employees with a salary: 14
Number of employees with a salary of at least $2000.00: 6
Number of employees with a salary between $2000.00 and $3000.00: 5
```

次の例は、関数宣言で `AUTHID CURRENT_USER` 句と `STRICT` キーワードを使用する方法を示しています。

```
CREATE OR REPLACE FUNCTION dept_salaries(dept_id int) RETURN NUMBER
    STRICT
    AUTHID CURRENT_USER
BEGIN
    RETURN QUERY (SELECT sum(salary) FROM emp WHERE deptno = id);
END;
```

渡された入力パラメーターが `NULL` の場合に `NULL` を返すようにサーバーに指示するには、`STRICT` キーワードを含めます。`NULL` 値が渡された場合、関数は実行されません。

`dept_salaries` 関数は、関数を呼び出しているロールの特権で実行されます。現在のユーザーに「従業員の給与を表示するために」`emp` テーブルを照会する `SELECT` ステートメントを実行するための十分な権限がない場合、関数はエラーを報告します。関数を定義したロールに関連付けられた特権を使用するようにサーバーに指示するには、`AUTHID CURRENT_USER` 句を `AUTHID DEFINER` 句に置き換えます。

Other Pragmas (declared within a package specification)

PRAGMA RESTRICT_REFERENCES

AdvancedServer は、PRAGMA RESTRICT_REFERENCES を参照する構文を受け入れますが無視します。

See Also

[ドロップ関数](#)

第 24 章

CREATE INDEX

Name

CREATE INDEX -- は新しいインデックスを定義します

Synopsis

```
CREATE [ UNIQUE ] INDEX <name> ON <table>
    ( { <column> | ( <expression> ) | <constant> } )
    [ TABLESPACE <tablespace> ]
    ( { NOPARALLEL | PARALLEL [ <integer> ] } )
```

Description

CREATE INDEX は指定されたテーブルにインデックス name を作成します。インデックスは主にデータベースのパフォーマンスを向上させるために使用されます（ただし、不適切な使用はパフォーマンスを低下させます）。

インデックスのキーフィールドは、列名、定数、または括弧で囲まれた式として指定されます。複数列のインデックスを作成するために複数のフィールドを指定できます。

インデックスフィールドは、テーブル行の 1 つ以上の列の値から計算された式です。この機能を使用して、基本データの何らかの変換に基づいてデータに高速でアクセスできます。たとえば、UPPER(col) で計算されたインデックスでは、WHERE UPPER(col) = 'JIM' 節でインデックスを使用できます。

AdvancedServer は、B ツリーインデックス方式を提供します。B ツリーインデックスメソッドは、Lehman-Yao の高同時実行 B ツリーの実装です。

デフォルトでは、インデックスは IS NULL 句には使用されません。

インデックス定義で使用するすべての関数と演算子は/"不変/"でなければなりません。つまり、結果は引数のみに依存し、外部の影響（別のテーブルの内容や現在の時間など）に依存してはなりません。この制限により、インデックスの動作が明確に定義されます。インデックス式でユーザー定義関数を使用するには、作成時に関数を不変としてマークすることを忘れないでください。

パーティションテーブルにインデックスを作成する場合、`CREATE INDEX` コマンドはインデックスをテーブルのサブパーティションに伝達します。

`PARALLEL` 句は、インデックスの作成中に使用される並列度を指定します。`NOPARALLEL` 句は、並列度をデフォルト値にリセットします。`reloptions` は `parallel_workers` パラメーターを 0 として表示します。

注釈: `CREATE INDEX... PARALLEL` コマンドを使用して、定義に `PARALLEL` 句が含まれるテーブルにインデックスを作成する場合（作成時）、サーバーは、並列インデックスを作成するときに `CREATE INDEX` コマンドで提供される `PARALLEL` 句を使用します。

Parameters

UNIQUE

システムは、インデックスの作成時（データが既に存在する場合）およびデータが追加されるたびに、テーブル内の重複値をチェックします。エントリが重複するデータを挿入または更新しようとする、エラーが生成されます。

name

作成するインデックスの名前。ここにスキーマ名を含めることはできません。インデックスは常にその親テーブルと同じスキーマで作成されます。

table

インデックスを作成するテーブルの名前（スキーマで修飾されている可能性があります）。

column

テーブル内の列の名前。

expression

テーブルの 1 つ以上の列に基づく式。式は通常、構文に示されているように、括弧で囲む必要があります。ただし、式が関数呼び出しの形式である場合、括弧は省略できます。

constant

インデックスキーとして使用できる定数値。

通常、インデックス付けされたすべての列が `NULL` である行は、インデックスに含まれません。これは、オプティマイザーが特定のクエリにそのインデックスを使用できないことを意味します。この制限を克服するために、インデックスに定数を追加して、すべてのインデックス列が `NULL` である行がインデックスに含まれないようにすることができます。

tablespace

インデックスを作成するテーブルスペース。指定しない場合は `default_tablespace` が使用され、`default_tablespace` が空の文字列の場合はデータベースのデフォルトのテーブルスペースが使用されます。

PARALLEL

PARALLEL を指定して、並列度を選択します。parallel_workers パラメータを並列度に等しく設定して、並列化されたインデックスを作成します。あるいは、PARALLEL を指定しても並列度がリストされていない場合、インデックスはデフォルトの並列度を受け入れます。

NOPARALLEL

デフォルトの実行には NOPARALLEL を指定します。

integer

integer は並列度を示します。これは、インデックスの並列スキャンを実行するための並列操作で使われる parallel_workers の数です。

Notes

複数列のインデックスには、最大 32 個のフィールドを指定できます。

Examples

テーブル emp の列 ename に B ツリーインデックスを作成するには:

```
CREATE INDEX name_idx ON emp (ename);
```

上記と同じインデックスを作成し、index_tblspc テーブルスペースに配置するには:

```
CREATE INDEX name_idx ON emp (ename) TABLESPACE index_tblspc;
```

インデックスの定義に定数値 (1) を含めて、すべてのインデックス付き列が NULL である行を決して含まないインデックスを作成できます。

```
CREATE INDEX emp_dob_idx on emp (emp_dob, 1);
```

並列度を 7 に設定して、テーブル emp の name_idx にインデックスを作成するには:

```
CREATE UNIQUE INDEX name_idx ON emp (ename) PARALLEL 7;
```

See Also

[ALTERINDEX](#)、[DROPINDEX](#)

第 25 章

CREATE MATERIALIZED VIEW

Name

CREATE MATERIALIZED VIEW -- 新しいマテリアライズドビューを定義する

Synopsis

```
CREATE MATERIALIZED VIEW <name>
    [<build_clause>][<create_mv_refresh>] AS subquery

    Where <build_clause> is:

    BUILD {IMMEDIATE | DEFERRED}

    Where <create_mv_refresh> is:

    REFRESH [COMPLETE] [ON DEMAND]``
```

Description

CREATE MATERIALIZED VIEW は、クエリでビューが参照されるたびに更新されないクエリのビューを定義します。デフォルトでは、ビューの作成時にビューにデータが入力されます。BUILD DEFERRED キーワードを含めて、ビューの作成を遅らせることができます。

マテリアライズドビューは、スキーマで修飾できます。CREATE MATERIALIZED VIEW コマンドを呼び出すときにスキーマ名を指定すると、指定したスキーマにビューが作成されます。ビュー名は、同じスキーマ内の他のビュー、テーブル、シーケンス、またはインデックスの名前とは異なる必要があります。

Parameters

name

作成するビューの名前（オプションでスキーマ修飾）。

subquery

ビューの内容を指定する SELECT ステートメント。有効なクエリの詳細については、SELECT を参照してください。

build_clause

ビューにデータを入力するタイミングを指定するには、build_clause を含めます。BUILD IMMEDIATE または BUILD DEFERRED を指定します。

- BUILD IMMEDIATE は、すぐにビューを表示するようサーバーに指示します。これがデフォルトの動作です。
- BUILD DEFERRED は、後で（更新操作中に）ビューにデータを取り込むようにサーバーに指示します。

create_mv_refresh

マテリアライズドビューのコンテンツを更新するタイミングを指定するには、create_mv_refresh 句を含めます。句には、REFRESH キーワードの後に COMPLETE および/または ON DEMAND が続きます。ここで、

- COMPLETE は、マテリアライズドビューが更新されたときにビューの定義クエリを実行して、現在のコンテンツを破棄し、マテリアライズドビューをリロードするようサーバーに指示します。
- ON DEMAND は、DBMS_MVIEW パッケージを呼び出さか、Postgres の REFRESH MATERIALIZED VIEW ステートメントを呼び出すことにより、要求に応じてマテリアライズドビューを更新するようサーバーに指示します。これがデフォルトの動作です。

Notes

マテリアライズドビューは読み取り専用です。サーバーは、ビューで INSERT、UPDATE、または DELETE を許可しません。

ビューで参照されるテーブルへのアクセスは、ビューの所有者の許可によって決定されます。ビューのユーザーには、ビューで使用されるすべての関数を呼び出すためのアクセス許可が必要です。

Postgres の REFRESH MATERIALIZED VIEW コマンドの詳細については、次の場所にある PostgreSQLCore-Documentation を参照してください。

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

Examples

次の文は、dept_30 という名前のマテリアライズドビューを作成します。

```
CREATE MATERIALIZED VIEW dept_30 BUILD IMMEDIATE AS SELECT * FROM emp
WHERE deptno = 30;
```

このビューには、部門 30 で働く従業員について emp テーブルから取得した情報が含まれています。

第 26 章

CREATE PACKAGE

Name

CREATE PACKAGE -- 新しいパッケージ仕様を定義します

Synopsis

```
CREATE [ OR REPLACE ] PACKAGE <name>
[ AUTHID { DEFINER | CURRENT_USER } ]
{ IS | AS }
[ <declaration>; ] [, ...]
[ { PROCEDURE <proc_name>
    [ (<argname> [ IN | IN OUT | OUT ] <argtype> [ DEFAULT value ]
      [, ...]) ];
  [ PRAGMA RESTRICT_REFERENCES(<name>,
    { RNDS | RNPS | TRUST | WNDS | WNPS } [, ... ] ); ]
  |
  FUNCTION <func_name>
    [ (<argname> [ IN | IN OUT | OUT ] <argtype> [ DEFAULT value ]
      [, ...]) ]
    RETURN <rettype> [ DETERMINISTIC ];
  [ PRAGMA RESTRICT_REFERENCES(<name>,
    { RNDS | RNPS | TRUST | WNDS | WNPS } [, ... ] ); ]
  }
] [, ...]
END [ <name> ]
```

Description

CREATE PACKAGE は新しいパッケージ仕様を定義します。CREATE OR REPLACE PACKAGE は、新しいパッケージ仕様を作成するか、既存の仕様を置き換えます。

スキーマ名が含まれている場合、パッケージは指定されたスキーマに作成されます。それ以外の場合は、現在のスキーマで作成されます。新しいパッケージの名前は、既存のパッケージの定義を更新する場合を除き、同じスキーマ内の既存のパッケージと一致してはいけません。その場合は CREATE OR REPLACE PACKAGE を使用し

ます。

プロシージャを作成したユーザーがパッケージの所有者になります。

Parameters

name

作成するパッケージの名前（オプションでスキーマ修飾）。

DEFINER | CURRENT_USER

パッケージで参照されるデータベースオブジェクトへのアクセスを許可するかどうかを決定するために、パッケージ所有者の特権（DEFINER）またはパッケージ内のプログラムを実行する現在のユーザーの特権（CURRENT_USER）のどちらを使用するかを指定します。DEFINER がデフォルトです。

declaration

パブリック変数、タイプ、カーソル、または REF CURSOR 宣言。

proc_name

パブリックプロシージャの名前。

argname

引数の名前。

IN | IN OUT | OUT

引数モード。

argtype

プログラムの引数のデータ型。

DEFAULT value

入力引数のデフォルト値。

func_name

パブリック関数の名前。

rettype

戻りデータ型。

DETERMINISTIC

DETERMINISTIC は IMMUTABLE の同義語です。DETERMINISTIC プロシージャはデータベースを変更できず、同じ引数値を指定すると常に同じ結果になります。データベースの検索は行わず、引数リストに直

接存在しない情報を使用しません。この句を含めると、すべて定数の引数を使用したプロシージャの呼び出しをすぐにプロシージャの値に置き換えることができます。

RNDS | RNPS | TRUST | WNDS | WNPS

キーワードは互換性のために受け入れられ、無視されます。

Examples

パッケージ仕様 `empinfo` には、パブリック変数、パブリックプロシージャ、パブリック関数の3つのパブリックコンポーネントが含まれています。

```
CREATE OR REPLACE PACKAGE empinfo
IS
    emp_name          VARCHAR2(10);
    PROCEDURE get_name (
        p_empno       NUMBER
    );
    FUNCTION display_counter
    RETURN INTEGER;
END;
```

See Also

[ドロップパッケージ](#)

第 27 章

CREATE PACKAGE BODY

Name

CREATE PACKAGE BODY -- は新しいパッケージ本体を定義します

Synopsis

```
CREATE [ OR REPLACE ] PACKAGE BODY <name>
{ IS | AS }
  [ declaration; ] | [ forward_declaration ] [, ...]
  [ { PROCEDURE <proc_name>
    [ (<argname> [ IN | IN OUT | OUT ] <argtype> [ DEFAULT <value> ] [,
      ...]) ]
    [ STRICT ]
    [ LEAKPROOF ]
    [ PARALLEL { UNSAFE | RESTRICTED | SAFE } ]
    [ COST <execution_cost> ]
    [ ROWS <result_rows> ]
    [ SET <config_param> { TO <value> | = <value> | FROM CURRENT } ]
  { IS | AS }
    <program_body>
  END [ <proc_name> ];
  |
  FUNCTION <func_name>
  [ (<argname> [ IN | IN OUT | OUT ] <argtype> [ DEFAULT <value> ] [,
    ...]) ]
  RETURN <rettype> [ DETERMINISTIC ]
  [ STRICT ]
  [ LEAKPROOF ]
  [ PARALLEL { UNSAFE | RESTRICTED | SAFE } ]
  [ COST <execution_cost> ]
  [ ROWS <result_rows> ]
  [ SET <config_param> { TO <value> | = <value> | FROM CURRENT } ]
{ IS | AS }
  <program_body>
```

(次のページに続く)

(前のページからの続き)

```

        END [ <func_name> ];
    }
] [, ...]
[ BEGIN
    <statement>; [, ...] ]
END [ <name> ]

```

Where forward_declaration:=

```

[ { PROCEDURE <proc_name>
    [ (<argname> [ IN | IN OUT | OUT ] <argtype> [ DEFAULT <value> ] [, ...])
    ] ; ]
|
[ { FUNCTION <func_name>
    [ (<argname> [ IN | IN OUT | OUT ] <argtype> [ DEFAULT <value> ] [, ...])
    ]
    RETURN <rettype> [ DETERMINISTIC ]; ]

```

Description

CREATE PACKAGE BODY は新しいパッケージ本体を定義します。CREATE OR REPLACE PACKAGE BODY は、新しいパッケージボディを作成するか、既存のボディを置き換えます。

スキーマ名が含まれている場合、パッケージ本体は指定されたスキーマに作成されます。それ以外の場合は、現在のスキーマで作成されます。新しいパッケージ本体の名前は、同じスキーマ内の既存のパッケージ仕様と一致する必要があります。新しいパッケージボディ名は、既存のパッケージボディの定義を更新する場合を除き、同じスキーマ内の既存のパッケージボディと一致してはなりません。その場合は CREATE OR REPLACE PACKAGE BODY を使用します。

Parameters

name

作成するパッケージ本体の名前（オプションでスキーマ修飾）。

declaration

プライベート変数、タイプ、カーソル、または REF CURSOR 宣言。

forward_declaration

プロシージャまたは関数の前方宣言は、パッケージ本体内に現れ、実際の本体定義の前に宣言されます。ブロックでは、複数のサブプログラムを作成できます。それらがお互いを呼び出す場合、それぞれが前方宣言を必要とします。サブプログラムは、呼び出す前に宣言する必要があります。前方宣言を使用して、サブプログラムを定義せずに宣言できます。前方宣言とそれに対応する定義は、同じブロックに存在する必要があります。

proc_name

パブリックプロシージャまたはプライベートプロシージャの名前。proc_name が同一の署名を持つパッケージ仕様に存在する場合、それはパブリックであり、そうでない場合はプライベートです。

argname

引数の名前。

IN | IN OUT | OUT

引数モード。

argtype

プログラムの引数のデータ型。

DEFAULT value

入力引数のデフォルト値。

STRICT

STRICT キーワードは、NULL 引数を指定して呼び出された場合、関数が実行されないことを指定します。代わりに、関数は NULL を返します。

LEAKPROOF

LEAKPROOF キーワードは、関数が戻り値以外の引数に関する情報を明らかにしないことを指定します。

PARALLEL { UNSAFE | RESTRICTED | SAFE }

PARALLEL 句は、並列順次スキャン（並列モード）の使用を有効にします。パラレルシーケンシャルスキャンは、シリアルシーケンシャルスキャンとは対照的に、クエリ中に複数のワーカーを使用してリレーションを並行してスキャンします。

UNSAFE に設定すると、プロシージャまたは関数をパラレルモードで実行できません。このようなプロシージャまたは関数が存在すると、シリアル実行プランが強制されます。これは、PARALLEL 句が省略された場合のデフォルト設定です。

RESTRICTED に設定すると、プロシージャまたは関数を並列モードで実行できますが、実行は並列グループリーダーに制限されます。特定のリレーションの資格に並列制限のあるものがある場合、そのリレーションは並列処理のために選択されません。

SAFE に設定すると、プロシージャまたは関数を制限なしでパラレルモードで実行できます。

execution_cost

execution_cost は、cpu_operator_cost の単位で、関数の推定実行コストを示す正の数を指定します。関数がセットを返す場合、これは返される行ごとのコストです。デフォルトは 0.0025 です。

result_rows

`result_rows` は、クエリプランナーが関数が返すと予想される行の推定数です。デフォルトは 1000 です。

SET

SET 句を使用して、関数の継続時間のパラメーター値を指定します。

`config_param` はパラメータ名を指定します。

`value` はパラメータ値を指定します。

FROM CURRENT は、関数の終了時にパラメーター値が復元されることを保証します。

program_body

関数またはプロシージャーの本体を構成するプラグマ、宣言、および SPL ステートメント。

プラグマを PRAGMA AUTONOMOUS_TRANSACTION にして、関数またはプロシージャーを自律型トランザクションとして設定できます。

宣言には、変数、型、REF CURSOR、またはサブプログラムの宣言が含まれる場合があります。サブプログラム宣言が含まれる場合、他のすべての変数、型、および REF CURSOR 宣言の後に宣言する必要があります。

func_name

パブリック関数またはプライベート関数の名前。func_name が同一の署名を持つパッケージ仕様に存在する場合、それはパブリックであり、そうでない場合はプライベートです。

rettype

戻りデータ型。

DETERMINISTIC

DETERMINISTIC を含めて、同じ引数値が指定されたときに関数が常に同じ結果を返すように指定します。DETERMINISTIC 関数はデータベースを変更してはなりません。

注釈: DETERMINISTIC キーワードは、PostgreSQL の IMMUTABLE オプションと同等です。パッケージ本体のパブリック関数に DETERMINISTIC が指定されている場合、パッケージ仕様の関数宣言にも DETERMINISTIC を指定する必要があります。プライベート関数の場合、パッケージ仕様に関数宣言はありません。

statement

SPL プログラムステートメント。パッケージ初期化セクションのステートメントは、パッケージが最初に参照されるときにセッションごとに 1 回実行されます。

注釈: STRICT、LEAKPROOF、PARALLEL、COST、ROWS、および SET キーワードは、AdvancedServer の拡張機能を提供するものであり、Oracle ではサポートされていません。

Examples

以下は empinfo パッケージのパッケージ本体です。

```
CREATE OR REPLACE PACKAGE BODY empinfo
IS
    v_counter          INTEGER;
    PROCEDURE get_name (
        p_empno        NUMBER
    )
    IS
    BEGIN
        SELECT ename INTO emp_name FROM emp WHERE empno = p_empno;
        v_counter := v_counter + 1;
    END;
    FUNCTION display_counter
    RETURN INTEGER
    IS
    BEGIN
        RETURN v_counter;
    END;
BEGIN
    v_counter := 0;
    DBMS_OUTPUT.PUT_LINE('Initialized counter');
END;
```

次の 2 つの匿名ブロックは、empinfo パッケージのプロシージャと関数を実行し、パブリック変数を表示します。

```
BEGIN
    empinfo.get_name(7369);
    DBMS_OUTPUT.PUT_LINE('Employee Name      : ' || empinfo.emp_name);
    DBMS_OUTPUT.PUT_LINE('Number of queries: ' || empinfo.display_counter);
END;

Initialized counter
Employee Name      : SMITH
Number of queries: 1

BEGIN
    empinfo.get_name(7900);
    DBMS_OUTPUT.PUT_LINE('Employee Name      : ' || empinfo.emp_name);
    DBMS_OUTPUT.PUT_LINE('Number of queries: ' || empinfo.display_counter);
END;
```

(次のページに続く)

(前のページからの続き)

```
Employee Name      : JAMES
Number of queries: 2
```

次の例は、パッケージ本体内で前方宣言を使用する方法を示しています。この例では、給与が指定範囲内にある従業員の名前と人数を表示します。

```
CREATE OR REPLACE PACKAGE empinfo IS
    FUNCTION emp_comp (p_sal_range INTEGER) RETURN INTEGER;
END empinfo;

CREATE OR REPLACE PACKAGE BODY empinfo IS
    FUNCTION sal_range (p_sal_range INTEGER) RETURN INTEGER;
    PROCEDURE list_emp (p_sal_range INTEGER);

    FUNCTION emp_comp (p_sal_range INTEGER) RETURN INTEGER IS
    BEGIN
        dbms_output.put_line ('Employee details');
        return sal_range(p_sal_range);
    END;

    FUNCTION sal_range (p_sal_range INTEGER) RETURN INTEGER IS
        emp_cnt INTEGER;
    BEGIN
        select count(*) into emp_cnt from emp where sal <= p_sal_range;
        dbms_output.put_line('Number of employees in the salary range ' ||
p_sal_range|| ' is :'|| emp_cnt);
        list_emp(p_sal_range);
        return emp_cnt;
    END;

    PROCEDURE list_emp (p_sal_range IN INTEGER) IS
    BEGIN
        FOR i IN select ename from emp where sal <= p_sal_range
        LOOP
            dbms_output.put_line (i);
        END LOOP;
    END;

END empinfo;

SELECT empinfo.emp_comp(1500);
Employee details
Number of employees in the salary range 1500 is :7
(SMITH)
(WARD)
```

(次のページに続く)

(前のページからの続き)

```
(MARTIN)
(TURNER)
(ADAMS)
(JAMES)
(MILLER)
emp_comp
-----
          7
(1 row)
```

See Also*CREATEPACKAGE*、*DROPPACKAGE*

第 28 章

CREATE PROCEDURE

Name

CREATE PROCEDURE -- 新しいストアドプロシージャを定義します

Synopsis

```
CREATE [OR REPLACE] PROCEDURE <name> [ (<parameters>) ]
[
    IMMUTABLE
  | STABLE
  | VOLATILE
  | DETERMINISTIC
  | [ NOT ] LEAKPROOF
  | CALLED ON NULL INPUT
  | RETURNS NULL ON NULL INPUT
  | STRICT
  | [ EXTERNAL ] SECURITY INVOKER
  | [ EXTERNAL ] SECURITY DEFINER
  | AUTHID DEFINER
  | AUTHID CURRENT_USER
  | PARALLEL { UNSAFE | RESTRICTED | SAFE }
  | COST <execution_cost>
  | ROWS <result_rows>
  | SET <configuration_parameter>
    { TO <value> | = <value> | FROM CURRENT }
... ]
{ IS | AS }
[ PRAGMA AUTONOMOUS_TRANSACTION; ]
[ <declarations> ]
BEGIN
    <statements>
END [ <name> ];
```

Description

CREATE PROCEDURE は新しいストアドプロシージャを定義します。CREATE OR REPLACE PROCEDURE は、新しいプロシージャを作成するか、既存の定義を置き換えます。

スキーマ名が含まれている場合、プロシージャは指定されたスキーマに作成されます。それ以外の場合は、現在のスキーマで作成されます。新しいプロシージャの名前は、同じスキーマ内の同じ入力引数タイプを持つ既存のプロシージャと一致してはなりません。ただし、異なる入力引数タイプのプロシージャは名前を共有する場合があります（これは *overloading* と呼ばれます）。（プロシージャのオーバーロードは *AdvancedServer* の機能です。ストアドスタンドアロンプロシージャのオーバーロードは *Oracle* データベースと互換性がありません。）

既存のプロシージャの定義を更新するには、CREATE OR REPLACE PROCEDURE を使用します。この方法でプロシージャの名前または引数の型を変更することはできません（実際に試してみると、実際には新しい別個のプロシージャを作成することになります）。OUT パラメータを使用する場合、OUT パラメータのタイプを変更するには、プロシージャをドロップする必要があります。

Parameters

name

name はプロシージャの識別子です。

parameters

parameters は仮パラメータのリストです。

declarations

declarations は、変数、カーソル、タイプ、またはサブプログラムの宣言です。サブプログラム宣言が含まれる場合、他のすべての変数、カーソル、および型宣言の後に宣言する必要があります。

statements

statements は *SPL* プログラムステートメントです（BEGIN - END ブロックには *EXCEPTION* セクションが含まれる場合があります）。

IMMUTABLE

STABLE

VOLATILE

これらの属性は、クエリオプティマイザーにプロシージャの動作を通知します。指定できる選択肢は 1 つだけです。VOLATILE がデフォルトの動作です。

IMMUTABLE は、プロシージャがデータベースを変更できず、同じ引数値が指定されたときに常に同じ結果に達することを示します。データベースの検索は行わず、引数リストに直接存在しない情報を使用しません。この句を含めると、すべて定数の引数を使用したプロシージャの呼び出しをすぐにプロシージャの値に置き換えることができます。

STABLE は、プロシージャがデータベースを変更できないこと、および単一のテーブルスキャン内で同じ引数値に対して一貫して同じ結果を返しますが、その結果は SQL ステートメント間で変わる可能性があることを示します。これは、データベースルックアップ、パラメータ変数（現在のタイムゾーンなど）などに依存する手順に適した選択です。

VOLATILE は、1 回のテーブルスキャン内でもプロシージャ値が変更される可能性があるため、最適化できないことを示します。副作用のある関数は、結果がかなり予測可能である場合でも、呼び出しが最適化されてしまうのを防ぐために、揮発性に分類する必要があることに注意してください。

DETERMINISTIC

DETERMINISTIC は IMMUTABLE の同義語です。DETERMINISTIC プロシージャはデータベースを変更できず、同じ引数値を指定すると常に同じ結果になります。データベースの検索は行わず、引数リストに直接存在しない情報を使用しません。この句を含めると、すべて定数の引数を使用したプロシージャの呼び出しをすぐにプロシージャの値に置き換えることができます。

[NOT] LEAKPROOF

LEAKPROOF プロシージャには副作用がなく、プロシージャの呼び出しに使用される値に関する情報は表示されません。

CALLED ON NULL INPUT

RETURNS NULL ON NULL INPUT

STRICT

CALLED ON NULL INPUT（デフォルト）は、引数の一部が NULL の場合にプロシージャが正常に呼び出されることを示します。必要に応じて NULL の値を確認し、適切に対応するのは作成者の責任です。

RETURNS NULL ON NULL INPUT または STRICT は、その引数のいずれかが NULL である場合、常に NULL を返すことを示します。これらの句が指定されている場合、NULL 個の引数がある場合、プロシージャは実行されません。代わりに、NULL の結果が自動的に想定されます。

[EXTERNAL] SECURITY DEFINER

SECURITY DEFINER は、プロシージャを作成したユーザーの権限で実行することを指定します。これがデフォルトです。キーワード EXTERNAL は SQL 準拠に使用できますが、オプションです。

[EXTERNAL] SECURITY INVOKER

SECURITY INVOKER 句は、プロシージャを呼び出すユーザーの権限でプロシージャが実行されることを示します。キーワード EXTERNAL は SQL 準拠に使用できますが、オプションです。

AUTHID DEFINER

AUTHID CURRENT_USER

AUTHID DEFINER 節は [EXTERNAL] SECURITY DEFINER の同義語です。AUTHID 句が省略されている場合、または AUTHID DEFINER が指定されている場合、プロシージャ所有者の権限を使用して、データベースオブジェクトへのアクセス権限が決定されます。

AUTHID CURRENT_USER 節は [EXTERNAL] SECURITY INVOKER の同義語です。AUTHID CURRENT_USER を指定すると、プロシージャを実行している現在のユーザーの権限を使用して、アクセス権限が決定されます。

PARALLEL { UNSAFE | RESTRICTED | SAFE }

PARALLEL 句は、並列順次スキャン（並列モード）の使用を有効にします。パラレルシーケンシャルスキャンは、シリアルシーケンシャルスキャンとは対照的に、クエリ中に複数のワーカーを使用してリレーションを並行してスキャンします。

UNSAFE に設定すると、プロシージャをパラレルモードで実行できません。このようなプロシージャが存在すると、シリアル実行プランが強制されます。これは、PARALLEL 句が省略された場合のデフォルト設定です。

RESTRICTED に設定すると、プロシージャはパラレルモードで実行できますが、実行はパラレルグループリーダーに制限されます。特定のリレーションの資格に並列制限のあるものがある場合、そのリレーションは並列処理のために選択されません。

SAFE に設定すると、プロシージャは制限なしでパラレルモードで実行できます。

COST execution_cost

execution_cost は、cpu_operator_cost を単位とする、プロシージャの推定実行コストを示す正数です。プロシージャがセットを返す場合、これは返される行ごとのコストです。値が大きいと、プランナーは必要以上に関数を評価することを避けようとしています。

ROWS result_rows

result_rows は正の数で、プランナーがプロシージャが返すと予想される行の推定数を示します。これは、プロシージャがセットを返すように宣言されている場合にのみ許可されます。デフォルトの想定は 1000 行です。

SET configuration_parameter { TO value | = value | FROM CURRENT }

SET 句により、指定された構成パラメータは、プロシージャに入るときに指定された値に設定され、プロシージャが終了するときに以前の値に復元されます。SET FROM CURRENT は、パラメータのセッションの現在の値を、プロシージャに入るときに適用される値として保存します。

SET 句がプロシージャに付加されている場合、同じ変数に対してプロシージャ内で実行される SET LOCAL コマンドの効果はプロシージャに制限されます。構成パラメータの以前の値は、プロシージャの終了時に復元されます。通常の SET コマンド（LOCAL なし）は、以前の SET LOCAL コマンドの場合と同様に SET 句をオーバーライドし、現在のトランザクションがロールされない限り、プロシージャ終了後もそのようなコマンドの効果が持続します戻る。

PRAGMA AUTONOMOUS_TRANSACTION

PRAGMA AUTONOMOUS_TRANSACTION は、プロシージャを自律型トランザクションとして設定するディレクティブです。

注釈:

- STRICT、LEAKPROOF、PARALLEL、COST、ROWS、および SET キーワードは、AdvancedServer の拡張機能を提供するものであり、Oracle ではサポートされていません。
- IMMUTABLE、STABLE、STRICT、LEAKPROOF、COST、ROWS、PARALLEL { UNSAFE | RESTRICTED | SAFE } 属性は、EDBSPL プロシージャでのみサポートされています。
- デフォルトでは、ストアードプロシージャは SECURITY DEFINERS として作成されます。plpgsql で定義されたストアードプロシージャは SECURITY INVOKERS として作成されます。

Examples

次の手順では、emp テーブルの従業員をリストします。

```
CREATE OR REPLACE PROCEDURE list_emp
IS
    v_empno          NUMBER(4);
    v_ename          VARCHAR2(10);
    CURSOR emp_cur IS
        SELECT empno, ename FROM emp ORDER BY empno;
BEGIN
    OPEN emp_cur;
    DBMS_OUTPUT.PUT_LINE('EMPNO      ENAME');
    DBMS_OUTPUT.PUT_LINE('-----      -----');
    LOOP
        FETCH emp_cur INTO v_empno, v_ename;
        EXIT WHEN emp_cur%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_empno || '      ' || v_ename);
    END LOOP;
    CLOSE emp_cur;
END;

EXEC list_emp;

EMPNO      ENAME
-----      -----
7369       SMITH
7499       ALLEN
7521       WARD
7566       JONES
7654       MARTIN
```

(次のページに続く)

(前のページからの続き)

7698	BLAKE
7782	CLARK
7788	SCOTT
7839	KING
7844	TURNER
7876	ADAMS
7900	JAMES
7902	FORD
7934	MILLER

次の手順では、IN OUT および OUT 引数を使用して、最初に指定された従業員番号を使用し、それが見つからない場合は指定された名前を使用した検索に基づいて従業員の番号、名前、およびジョブを返します匿名ブロックがプロシージャを呼び出します。

```
CREATE OR REPLACE PROCEDURE emp_job (
    p_empno      IN OUT emp.empno%TYPE,
    p_ename      IN OUT emp.ename%TYPE,
    p_job        OUT      emp.job%TYPE
)
IS
    v_empno      emp.empno%TYPE;
    v_ename      emp.ename%TYPE;
    v_job        emp.job%TYPE;
BEGIN
    SELECT ename, job INTO v_ename, v_job FROM emp WHERE empno = p_empno;
    p_ename := v_ename;
    p_job   := v_job;
    DBMS_OUTPUT.PUT_LINE('Found employee # ' || p_empno);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        BEGIN
            SELECT empno, job INTO v_empno, v_job FROM emp
                WHERE ename = p_ename;
            p_empno := v_empno;
            p_job   := v_job;
            DBMS_OUTPUT.PUT_LINE('Found employee ' || p_ename);
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('Could not find an employee with ' ||
                    'number, ' || p_empno || ' nor name, ' || p_ename);
                p_empno := NULL;
                p_ename := NULL;
                p_job   := NULL;
        END;
END;

DECLARE
```

(次のページに続く)

(前のページからの続き)

```

v_empno      emp.empno%TYPE;
v_ename      emp.ename%TYPE;
v_job        emp.job%TYPE;
BEGIN
    v_empno := 0;
    v_ename := 'CLARK';
    emp_job(v_empno, v_ename, v_job);
    DBMS_OUTPUT.PUT_LINE('Employee No: ' || v_empno);
    DBMS_OUTPUT.PUT_LINE('Name       : ' || v_ename);
    DBMS_OUTPUT.PUT_LINE('Job       : ' || v_job);
END;

Found employee CLARK
Employee No: 7782
Name       : CLARK
Job       : MANAGER

```

次の例は、プロシージャ宣言で AUTHID DEFINER および SET 句を使用する方法を示しています。update_salary プロシージャは、プロシージャを定義しているロールの特権を、プロシージャを呼び出しているロールに（プロシージャの実行中に）伝えます。

```

CREATE OR REPLACE PROCEDURE update_salary(id INT, new_salary NUMBER)
    SET SEARCH_PATH = 'public' SET WORK_MEM = '1MB'
    AUTHID DEFINER IS
BEGIN
    UPDATE emp SET salary = new_salary WHERE emp_id = id;
END;

```

SET 句を含めて、プロシージャの検索パスを public に、ワークメモリを 1MB に設定します。他の手順、関数、およびオブジェクトは、これらの設定の影響を受けません。

この例では、AUTHID DEFINER 句は、それ以外の場合はプロシージャ内のステートメントの実行を許可されないロールに一時的に特権を付与します。プロシージャを呼び出すロールに関連付けられた特権を使用するようにサーバーに指示するには、AUTHID DEFINER 句を AUTHID CURRENT_USER 句に置き換えます。

See Also

ドロップ手順、[ALTERPROCEDURE](#)

第 29 章

CREATE PROFILE

Name

CREATE PROFILE - は新しいプロファイルを作成します

Synopsis

```
CREATE PROFILE <profile_name>
    [LIMIT {<parameter value>} ... ];
```

Description

CREATE PROFILE は新しいプロファイルを作成します。LIMIT 句と 1 つ以上のスペース区切り parameter/value ペアを含めて、AdvancedServer によって適用されるルールを指定します。

AdvancedServer は、DEFAULT という名前のデフォルトプロファイルを作成します。CREATE ROLE コマンドを使用して新しいロールを作成すると、新しいロールは DEFAULT プロファイルに自動的に関連付けられます。AdvancedServer の以前のバージョンから AdvancedServer10 にアップグレードする場合、アップグレードプロセスにより、アップグレードされたバージョンのロールが DEFAULT プロファイルに自動的に作成されます。

CREATE PROFILE を使用するには、スーパーユーザーである必要があります。

AdvancedServer によって適用されるルールを指定するには、LIMIT 句と 1 つ以上のスペース区切り parameter/value ペアを含めます。

Parameters

profile_name

プロファイルの名前。

parameter

ルールによって監視されるパスワード属性。

value

サーバーがアクションを実行する前に `parameter` が到達する必要がある値。

AdvancedServer は、`parameter` ごとに以下に示す `value` をサポートしています。

`FAILED_LOGIN_ATTEMPTS` は、`PASSWORD_LOCK_TIME` で指定された期間、サーバーがユーザーをアカウントからロックアウトする前にユーザーが試行できるログイン試行の失敗回数を指定します。サポートされる値は次のとおりです。

- 0 より大きい INTEGER 値。
- DEFAULT - DEFAULT プロファイルで指定された `FAILED_LOGIN_ATTEMPTS` の値。
- UNLIMITED - 接続ユーザーは、ログイン試行の失敗回数に制限はありません。

`PASSWORD_LOCK_TIME` は、`FAILED_LOGIN_ATTEMPTS` のためにロックされたアカウントをサーバーがロック解除するまでに経過する必要がある時間の長さを指定します。サポートされる値は次のとおりです。

- 0 以上の NUMERIC 値。日の小数部分を指定するには、小数値を指定します。たとえば、4.5 という値を使用して 4 日、12 時間を指定します。
- DEFAULT - DEFAULT プロファイルで指定された `PASSWORD_LOCK_TIME` の値。
- UNLIMITED - データベースのスーパーユーザーによって手動でロックが解除されるまで、アカウントはロックされます。

`PASSWORD_LIFE_TIME` は、ユーザーに新しいパスワードの入力を求めるまでに現在のパスワードを使用できる日数を指定します。`PASSWORD_LIFE_TIME` 句を使用してパスワードによる有効期限が切れてからロールによる接続が拒否されるまでの日数を指定する場合は、`PASSWORD_GRACE_TIME` 句を含めます。`PASSWORD_GRACE_TIME` が指定されていない場合、パスワードは `PASSWORD_GRACE_TIME` のデフォルト値で指定された日に失効し、ユーザーは新しいパスワードが提供されるまでコマンドの実行を許可されません。サポートされる値は次のとおりです。

- 0 以上の NUMERIC 値。日の小数部分を指定するには、小数値を指定します。たとえば、4.5 という値を使用して 4 日、12 時間を指定します。
- DEFAULT - DEFAULT プロファイルで指定された `PASSWORD_LIFE_TIME` の値。
- UNLIMITED - パスワードには有効期限がありません。

`PASSWORD_GRACE_TIME` は、パスワードの有効期限が切れてからユーザーがパスワードの変更を強制されるまでの猶予期間の長さを指定します。猶予期間が終了すると、ユーザーは接続を許可されますが、期限切れのパスワードを更新するまでコマンドの実行は許可されません。サポートされる値は次のとおりです。

- 0 以上の NUMERIC 値。1 日の小数部分を指定するには、10 進数値を指定します。たとえば、値 4.5 を使用して 4 日、12 時間を指定します。
- DEFAULT - DEFAULT プロファイルで指定された `PASSWORD_GRACE_TIME` の値。

- UNLIMITED –猶予期間は無限です。

PASSWORD_REUSE_TIME は、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日数を指定します。PASSWORD_REUSE_TIME および PASSWORD_REUSE_MAX パラメータは、一緒に使用することを目的としています。これらのパラメーターの一方に有限値を指定し、他方が UNLIMITED の場合、古いパスワードは再利用できません。両方のパラメーターが UNLIMITED に設定されている場合、パスワードの再利用に関する制限はありません。サポートされる値は次のとおりです。

- 0 以上の NUMERIC 値。日の小数部分を指定するには、小数値を指定します。たとえば、4.5 という値を使用して 4 日、12 時間を指定します。
- DEFAULT - DEFAULT プロファイルで指定された PASSWORD_REUSE_TIME の値。
- UNLIMITED –パスワードは制限なしで再利用できます。

PASSWORD_REUSE_MAX は、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を指定します。PASSWORD_REUSE_TIME および PASSWORD_REUSE_MAX パラメータは、一緒に使用することを目的としています。これらのパラメーターの一方に有限値を指定し、他方が UNLIMITED の場合、古いパスワードは再利用できません。両方のパラメーターが UNLIMITED に設定されている場合、パスワードの再利用に関する制限はありません。サポートされる値は次のとおりです。

- 0 以上の INTEGER 値。
- DEFAULT - DEFAULT プロファイルで指定された PASSWORD_REUSE_MAX の値。
- UNLIMITED –パスワードは制限なしで再利用できます。

PASSWORD_VERIFY_FUNCTION はパスワードの複雑さを指定します。サポートされる値は次のとおりです。

- PL/SQL 関数の名前。
- DEFAULT - DEFAULT プロファイルで指定された PASSWORD_VERIFY_FUNCTION の値。
- NULL

PASSWORD_ALLOW_HASHED は、暗号化されたパスワードの使用を許可するかどうかを指定します。値を TRUE に指定すると、システムは、クライアント側でハッシュ計算された暗号化パスワードを指定することにより、ユーザーがパスワードを変更できるようにします。ただし、値を FALSE に指定した場合、パスワードを効果的に検証するにはプレーンテキスト形式で指定する必要があります。そうしないと、サーバーが暗号化されたパスワードを受信した場合にエラーがスローされます。サポートされる値は次のとおりです。

- BOOLEAN 値 TRUE/ON/YES/1 または FALSE/OFF/NO/0。
- DEFAULT – DEFAULT プロファイルで指定された PASSWORD_ALLOW_HASHED の値。

注釈: PASSWORD_ALLOW_HASHED は Oracle 互換のパラメーターではありません。

Notes

DROP PROFILE コマンドを使用して、プロファイルを削除します。

Examples

次のコマンドは、acctg という名前のプロファイルを作成します。プロファイルは、ユーザーが 5 回の試行で正しいパスワードで認証されなかった場合、アカウントが 1 日間ロックされることを指定します。

```
CREATE PROFILE acctg LIMIT
    FAILED_LOGIN_ATTEMPTS 5
    PASSWORD_LOCK_TIME 1;
```

次のコマンドは、sales という名前のプロファイルを作成します。このプロファイルは、ユーザーが 90 日ごとにパスワードを変更する必要があることを指定しています。

```
CREATE PROFILE sales LIMIT
    PASSWORD_LIFE_TIME 90
    PASSWORD_GRACE_TIME 3;
```

プロファイルで指定された 90 日が経過する前にユーザーがパスワードを変更していない場合、ログイン時に警告が発行されます。3 日間の猶予期間が過ぎると、アカウントはパスワードを変更するまでコマンドの呼び出しを許可されなくなります。

次のコマンドは、accts という名前のプロファイルを作成します。プロファイルは、ユーザーがパスワードを最後に使用してから 180 日以内にパスワードを再使用できないことを指定し、パスワードを再使用する前に少なくとも 5 回パスワードを変更する必要があります。

```
CREATE PROFILE accts LIMIT
    PASSWORD_REUSE_TIME 180
    PASSWORD_REUSE_MAX 5;
```

次のコマンドは、resources という名前のプロファイルを作成します。プロファイルは password_rules という名前のユーザー定義関数を呼び出し、指定されたパスワードが複雑さの基準を満たしていることを確認します。

```
CREATE PROFILE resources LIMIT
    PASSWORD_VERIFY_FUNCTION password_rules;
```

See Also

ALTERPROFILE、*DROPPROFILE*

第 30 章

CREATE QUEUE

AdvancedServer には、CREATE QUEUE SQL コマンドに追加の構文（Oracle によって提供されていない）が含まれています。この構文は、DBMS_AQADM と関連付けて使用できます。

Name

CREATE QUEUE - はキューを作成します。

Synopsis

CREATE QUEUE を使用して新しいキューを定義します。

```
CREATE QUEUE <name> QUEUE TABLE <queue_table_name> [ ( { <option_name>
option_value> } [, ... ] ) ]
```

option_name および対応する option_value は次のとおりです。

```
TYPE [normal_queue | exception_queue]
RETRIES [INTEGER]
RETRYDELAY [DOUBLE PRECISION]
RETENTION [DOUBLE PRECISION]
```

Description

CREATE QUEUE 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またはシステム定義の aq_administrator_role 権限を持つ任意のユーザーが現在のデータベースに新しいキューを作成できます。

キューの名前がスキーマ修飾されている場合、キューは指定されたスキーマに作成されます。スキーマが CREATE QUEUE コマンドに含まれていない場合、キューは現在のスキーマに作成されます。キューは、キューテーブルが存在するスキーマでのみ作成できます。キューの名前は、同じスキーマ内の他のキューの名前と一意である必要があります。

DROP QUEUE を使用してキューを削除します。

Parameters

name

作成するキューの名前（オプションでスキーマ修飾）。

queue_table_name

このキューが関連付けられているキューテーブルの名前。

option_name option_value

新しいキューに関連付けられるオプションの名前、およびオプションに対応する値。CREATE QUEUE の呼び出しに重複するオプション名が含まれている場合、サーバーはエラーを返します。次の値がサポートされています。

TYPE	キューが通常のキューであることを示すには normal_queue を指定し、キューが例外キューであることを示すには exception_queue を指定します。例外キューは、デキュー操作のみを受け入れます。
RETRIES	キューからメッセージを削除する最大試行回数を指定する INTEGER 値。
RETRYDELAY	サーバーがメッセージを再試行するまで待機する ROLLBACK の後の秒数を指定する DOUBLE PRECISION 値。
RETENTION	デキュー後にメッセージがキューテーブルに保存される秒数を指定する DOUBLE PRECISION 値。

Examples

次のコマンドは、work_order_table という名前のキューテーブルに関連付けられた work_order という名前のキューを作成します。

```
CREATE QUEUE work_order QUEUE TABLE work_order_table (RETRIES 5, RETRYDELAY
2);
```

サーバーは、キューからメッセージを削除するために 5 回の試行を許可し、試行間の 2 秒の再試行遅延を強制します。

See Also

ALTERQUEUE、*DROPQUEUE*

第 31 章

CREATE QUEUE TABLE

AdvancedServer には、CREATE QUEUE TABLE SQL コマンドに追加の構文（Oracle によって提供されていない）が含まれています。この構文は、DBMS_AQADM と関連付けて使用できます。

Name

CREATE QUEUE TABLE-- は新しいキューテーブルを作成します。

Synopsis

CREATE QUEUE TABLE を使用して、新しいキューテーブルを定義します。

```
CREATE QUEUE TABLE <name> OF <type_name> [ ( { <option_name option_value> }  
[, ... ] ) ]
```

option_name および対応する option_value は次のとおりです。

option_name	option_value
SORT_LIST	priority, enq_time
MULTIPLE_CONSUMERS	FALSE, TRUE
MESSAGE_GROUPING	NONE, TRANSACTIONAL

次のページに続く

表 1 – 前のページからの続き

option_name	option_value
STORAGE_CLAUSE	TABLESPACE tablespace_name, PCTFREE integer, PCTUSED integer, INITTRANS integer, MAXTRANS integer, STORAGE storage_option storage_option は次の 1 つ以上です。 MINEXTENTS integer, MAXEXTENTS integer, PCTINCREASE integer, INITIAL size_clause, NEXT, FREELISTS integer, OPTIMAL size_clause, BUFFER_POOL {KEEP RECYCLE DEFAULT}. TABLESPACE オプションのみが適用されることに注意してください。他のすべては互換性のために受け入れられ、無視されます。TABLESPACE 句を使用して、テーブルが作成されるテーブルスペースの名前を指定します。

Description

CREATE QUEUE TABLE は、スーパーユーザーまたは aq_administrator_role 特権を持つユーザーが新しいキューテーブルを作成できるようにします。

CREATE QUEUE TABLE への呼び出しにスキーマ名が含まれている場合、キューテーブルは指定されたスキーマに作成されます。スキーマ名が指定されていない場合、現在のスキーマに新しいキューテーブルが作成されます。

キュー表の名前は、同じスキーマ内の他のキュー表の名前と一意である必要があります。

Parameters

name

新しいキューテーブルの名前（オプションでスキーマ修飾）。

type_name

キューテーブル内の各エントリのペイロードを記述する既存のタイプの名前。タイプの定義については、CREATE TYPE を参照してください。

option_name option_value

新しいキューテーブルに関連付けられるオプションの名前、およびオプションに対応する値。CREATE QUEUE TABLE の呼び出しに重複するオプション名が含まれている場合、サーバーはエラーを返します。次の値が受け入れられます。

SORT_LIST	SORT_LIST オプションを使用して、キューのデキュー順序を制御します。キューのソートに使用される列の名前を（昇順で）指定します。現在受け入れられている値は、<code>enq_time</code> と <code>priority</code> の次の組み合わせです。 <pre> enq_time. priority priority. enq_time priority enq_time </pre> その他の値はエラーを返します。
MULTIPLE_CONSUMERS	メッセージに複数のコンシューマー（TRUE）を含めることができるか、メッセージごとに1つのコンシューマー（FALSE）に制限するかを示す BOOLEAN 値。
MESSAGE_GROUPING	各メッセージを個別にデキューすることを示すには <code>none</code> を指定し、1つのトランザクションの結果としてキューに追加されるメッセージをグループとしてデキューすることを示すには <code>transactional</code> を指定します。
STORAGE_CLAUSE	STORAGE_CLAUSE を使用して、テーブル属性を指定します。STORAGE_CLAUSE は <code>TABLESPACE tablespace_name, PCTFREE integer, PCTUSED integer, INITTRANS integer, MAXTRANS integer, STORAGE storage_option</code> かもしれません <code>storage_option</code> は次の1つ以上です。 <pre> MINEXTENTS integer, MAXEXTENTS integer, PCTINCREASE integer, INITIAL size_clause, NEXT, FREELISTS integer, OPTIMAL size_clause, BUFFER_POOL {KEEP RECYCLE DEFAULT}. </pre> TABLESPACE オプションのみが適用されることに注意してください。他のすべては互換性のために受け入れられ、無視されます。TABLESPACE 句を使用して、テーブルが作成されるテーブルスペースの名前を指定します。

Examples

キュー表を作成する前に、ユーザー定義型を作成する必要があります。型は、テーブル内の列とデータ型を記述します。次のコマンドは、`work_order` という名前のタイプを作成します。

```
CREATE TYPE work_order AS (name VARCHAR2, project TEXT, completed BOOLEAN);
```

次のコマンドは、`work_order` タイプを使用して `work_order_table` という名前のキューテーブルを作成します。

```
CREATE QUEUE TABLE work_order_table OF work_order (sort_list (enq_time,
priority));
```

See Also

ALTERQUEUETABLE、DROPQUEUETABLE

第 32 章

CREATE ROLE

Name

CREATE ROLE -- は新しいデータベースロールを定義します

Synopsis

```
CREATE ROLE <name> [IDENTIFIED BY <password> [REPLACE old_password]]
```

Description

CREATE ROLE は、AdvancedServer データベースクラスターに新しいロールを追加します。ロールは、データベースオブジェクトを所有し、データベース権限を持つことができるエンティティです。ロールは、その使用方法に応じて、「ユーザー」、「グループ」、またはその両方と見なすことができます。新しく作成されたロールには LOGIN 属性がないため、セッションの開始には使用できません。ALTER ROLE コマンドを使用して、ロールに LOGIN 権限を付与します。CREATE ROLE コマンドを使用するには、CREATEROLE 特権を持っているか、データベースのスーパーユーザーである必要があります。

IDENTIFIED BY 句が指定されている場合、CREATE ROLE コマンドは、新しく作成されたロールと同じ名前のスキーマを作成します。

注釈: ロールはデータベースクラスターレベルで定義されるため、クラスター内のすべてのデータベースで有効です。

Parameters

name

新しい役割の名前。

IDENTIFIED BY password

ロールのパスワードを設定します。(パスワードは、LOGIN 属性を持つロールでのみ使用されますが、それなしでロールに定義できます。) パスワード認証を使用する予定がない場合は、このオプションを省略できます。

Notes

ALTER ROLE を使用してロールの属性を変更し、DROP ROLE を使用してロールを削除します。CREATE ROLE で指定された属性は、後の ALTER ROLE コマンドで変更できます。

GRANT と REVOKE を使用して、グループとして使用されているロールのメンバーを追加および削除します。

ロール名とパスワードの最大長の制限は 63 文字です。

Examples

パスワードを使用して、admins という名前のロール（およびスキーマ）を作成します。

```
CREATE ROLE admins IDENTIFIED BY Rt498zb;
```

See Also

ALTERROLE、*DROPROLE*、*GRANT*、*REVOKE*、*SETROLE*

第 33 章

CREATE SCHEMA

Name

CREATE SCHEMA -- は新しいスキーマを定義します

Synopsis

```
CREATE SCHEMA AUTHORIZATION <username> <schema_element> [ ... ]
```

Description

CREATE SCHEMA コマンドのこのバリエーションは、username が所有し、1 つ以上のオブジェクトが入力された新しいスキーマを作成します。スキーマとオブジェクトの作成は単一のトランザクション内で行われるため、すべてのオブジェクトが作成されるか、スキーマを含むオブジェクトは作成されません。(注意:Oracle データベースを使用している場合、新しいスキーマは作成されません。username、つまりスキーマが存在している必要があります。)

スキーマは基本的に名前空間です。名前付きオブジェクト（テーブル、ビューなど）が含まれ、その名前は他のスキーマに存在する他のオブジェクトの名前と重複する場合があります。名前付きオブジェクトにアクセスするには、スキーマ名をプレフィックスとして使用して名前を「修飾」するか、目的のスキーマを含む検索パスを設定します。修飾されていないオブジェクトは、現在のスキーマ（検索パスの先頭にあり、関数 CURRENT_SCHEMA で決定できます）に作成されます。（検索パスの概念と CURRENT_SCHEMA 関数は、Oracle データベースと互換性はありません。）

CREATE SCHEMA には、スキーマ内にオブジェクトを作成するサブコマンドが含まれます。サブコマンドは、スキーマの作成後に発行される個別のコマンドと本質的に同じように扱われます。作成されたオブジェクトはすべて、指定されたユーザーが所有します。

Parameters

username

新しいスキーマを所有するユーザーの名前。スキーマには username と同じ名前が付けられます。スーパーユーザーのみが、自分以外のユーザーが所有するスキーマを作成できます。(注意:AdvancedServer で

は、ロール `username` がすでに存在している必要がありますが、スキーマは存在してはなりません。Oracle では、ユーザー（つまりスキーマ）が存在する必要があります。

`schema_element`

スキーマ内で作成されるオブジェクトを定義する SQL ステートメント。CREATE TABLE、CREATE VIEW、GRANT は CREATE SCHEMA 内の句として受け入れられます。他の種類のオブジェクトは、スキーマが作成された後、個別のコマンドで作成される場合があります。

Notes

スキーマを作成するには、呼び出し元のユーザーが現在のデータベースに対する CREATE 権限を持っている必要があります。（もちろん、スーパーユーザーはこのチェックをバイパスします。）

AdvancedServer には、Oracle データベースと互換性のない CREATE SCHEMA コマンドの他の形式があります。

Examples

```
CREATE SCHEMA AUTHORIZATION enterprisedb
  CREATE TABLE empjobs (ename VARCHAR2(10), job VARCHAR2(9))
  CREATE VIEW managers AS SELECT ename FROM empjobs WHERE job = 'MANAGER'
  GRANT SELECT ON managers TO PUBLIC;
```

第 34 章

CREATE SEQUENCE

Name

CREATE SEQUENCE -- 新しいシーケンスジェネレーターを定義する

Synopsis

```
CREATE SEQUENCE <name> [ INCREMENT BY <increment> ]  
[ { NOMINVALUE | MINVALUE <minvalue> } ]  
[ { NOMAXVALUE | MAXVALUE <maxvalue> } ]  
[ START WITH <start> ] [ CACHE <cache> | NOCACHE ] [ CYCLE ]
```

Description

CREATE SEQUENCE は、新しいシーケンス番号ジェネレーターを作成します。これには、name という名前の新しい特別な単一行テーブルの作成と初期化が含まれます。ジェネレーターは、コマンドを発行したユーザーが所有します。

スキーマ名が指定されている場合、シーケンスは指定されたスキーマに作成されます。指定されていない場合は、現在のスキーマに作成されます。シーケンス名は、同じスキーマの他のシーケンス、テーブル、インデックス、またはビューの名前とは異なる必要があります。

シーケンスが作成されたら、関数 NEXTVAL および CURRVAL を使用してシーケンスを操作します。これらの関数は、『Oracle 開発者向けデータベース互換性のシーケンス操作関数リファレンスガイド』に記載されています。

Parameters

name

作成されるシーケンスの名前（オプションでスキーマ修飾）。

increment

オプションの句 INCREMENT BY increment は、現在のシーケンス値に追加して新しい値を作成する値を指定します。正の値は昇順、負の値は降順になります。デフォルト値は 1 です。

NOMINVALUE | MINVALUE minvalue

オプションの句 MINVALUE minvalue は、シーケンスが生成できる最小値を決定します。この句が指定されていない場合、デフォルトが使用されます。デフォルトは、それぞれ昇順および降順の場合の 1 および $-2^{63}-1$ です。キーワード NOMINVALUE を使用して、この動作をデフォルトに設定できることに注意してください。

NOMAXVALUE | MAXVALUE maxvalue

オプションの句 MAXVALUE maxvalue は、シーケンスの最大値を決定します。この句が指定されていない場合、デフォルト値が使用されます。デフォルトは、昇順と降順の場合、それぞれ $2^{63}-1$ と -1 です。キーワード NOMAXVALUE を使用して、この動作をデフォルトに設定できることに注意してください。

start

オプションの句 START WITH start を使用すると、シーケンスをどこからでも開始できます。デフォルトの開始値は、昇順の場合は minvalue、降順の場合は maxvalue です。

cache

オプションの句 CACHE cache は、アクセスを高速化するためにメモリに事前に割り当てて保存するシーケンス番号の数を指定します。最小値は 1（一度に生成できる値は 1 つのみ、つまり NOCACHE）であり、これもデフォルトです。

CYCLE

CYCLE オプションを使用すると、maxvalue または minvalue にそれぞれ昇順または降順で到達したときに、シーケンスが折り返されます。制限に達すると、次に生成される数値はそれぞれ minvalue または maxvalue になります。

CYCLE が省略された場合（デフォルト）、シーケンスが最大値に達した後に NEXTVAL を呼び出すとエラーが返されます。キーワード NO CYCLE を使用してデフォルトの動作を取得できますが、この用語は Oracle データベースと互換性がないことに注意してください。

Notes

シーケンスはビッグ整数演算に基づいているため、範囲は 8 バイト整数の範囲（-9223372036854775808～9223372036854775807）を超えることはできません。一部の古いプラットフォームでは、8 バイト整数をコンパイラがサポートしていない場合があります。この場合、シーケンスは通常の INTEGER 演算（-2147483648～+2147483647 の範囲）を使用します。

複数のセッションで同時に使用されるシーケンスオブジェクトに 1 より大きい cache 設定を使用すると、予想しない結果が得られる場合があります。各セッションは、シーケンスオブジェクトへの 1 回のアクセス中に連続するシーケンス値を割り当ててキャッシュし、それに応じてシーケンスオブジェクトの最後の値を増やします。次に、そのセッション内で次に cache-1 が NEXTVAL を使用すると、シーケンスオブジェクトに触れることなく、事前に割り当てられた値が返されます。そのため、セッション内で割り当てられているが使用されていない番号は、そのセッションが終了すると失われ、シーケンスに「穴」が生じます。

さらに、複数のセッションが個別のシーケンス値を割り当てることが保証されていますが、すべてのセッションが考慮されると、シーケンス外の値が生成される場合があります。たとえば、`cache` 設定が 10 の場合、セッション A は値 1..10 を予約して `NEXTVAL=1` を返し、セッション B は値 11..20 を予約して `NEXTVAL=11` を返してからセッション A が `NEXTVAL=2` を生成します。したがって、`cache` 設定が 1 の場合、`NEXTVAL` 値が連続して生成されると想定しても安全です。`cache` の設定が 1 よりも大きい場合、`NEXTVAL` の値はすべて別個のものであり、純粋に順番に生成されるものではないと想定する必要があります。また、最後の値は、`NEXTVAL` によってまだ返されているかどうかに関係なく、セッションによって予約された最新の値を反映します。

Examples

101 から始まる `serial` という昇順シーケンスを作成します。

```
CREATE SEQUENCE serial START WITH 101;
```

このシーケンスから次の番号を選択します。

```
SELECT serial.NEXTVAL FROM DUAL;

nextval
-----
      101
(1 row)
```

`NOCACHE` オプションで `supplier_seq` というシーケンスを作成します。

```
CREATE SEQUENCE supplier_seq
    MINVALUE 1
    START WITH 1
    INCREMENT BY 1
    NOCACHE;
```

このシーケンスから次の番号を選択します。

```
SELECT supplier_seq.NEXTVAL FROM DUAL;

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

See Also

[ALTERSEQUENCE](#)、[ドロップシーケンス](#)

第 35 章

CREATE SYNONYM

Name

CREATE SYNONYM -- は新しい同義語を定義します

Synopsis

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM [<schema>.]<syn_name>  
      FOR <object_schema>.<object_name>[@<dblink_name>];
```

Description

CREATE SYNONYM は、特定のタイプのデータベースオブジェクトの同義語を定義します。AdvancedServer は、次の同義語をサポートしています。

- テーブル
- 景色
- マテリアライズドビュー
- シーケンス
- ストアドプロシージャ
- ストアドファンクション
- 種類
- データベースリンクを介してアクセス可能なオブジェクト
- 他の同義語

Parameters

syn_name

`syn_name` は同義語の名前です。シノニム名はスキーマ内で一意である必要があります。

`schema`

`schema` は、シノニムが存在するスキーマの名前を指定します。スキーマ名を指定しない場合、検索パスの最初の既存のスキーマにシノニムが作成されます。

`object_name`

`object_name` はオブジェクトの名前を指定します。

`object_schema`

`object_schema` は、参照されるオブジェクトが存在するスキーマの名前を指定します。

`dblink_name`

`dblink_name` は、オブジェクトへのアクセスに使用するデータベースリンクの名前を指定します。

REPLACE 句を含めて、既存の同義語定義を新しい同義語定義に置き換えます。

PUBLIC 句を含めて、`public` スキーマに同義語を作成します。Oracle データベースと互換性のある CREATE PUBLIC SYNONYM コマンドは、`public` スキーマに存在する同義語を作成します。

```
CREATE [OR REPLACE] PUBLIC SYNONYM <syn_name> FOR
<object_schema>.<object_name>;
```

これは書くための簡単な方法です:

```
CREATE [OR REPLACE] SYNONYM public.<syn_name> FOR
<object_schema>.<object_name>;
```

Notes

シノニムによって参照されるオブジェクトへのアクセスは、シノニムの現在のユーザーの権限によって決定されます。シノニムユーザーは、基になるデータベースオブジェクトに対する適切なアクセス許可を持っている必要があります。

Examples

`enterprisedb` という名前のスキーマで `emp` テーブルのシノニムを作成します。

```
CREATE SYNONYM personnel FOR enterprisedb.emp;
```

See Also

[*DROPSYNONYM*](#)

第 36 章

CREATE TABLE

Name

CREATE TABLE -- 新しいテーブルを定義する

Synopsis

```
CREATE [ GLOBAL TEMPORARY ] TABLE <table_name> (
  { <column_name> <data_type> [ DEFAULT <default_expr> ]
    [ <column_constraint> [ ... ] ] | <table_constraint> } [, ...]
)
[ WITH ( ROWIDS [= <value> ] ) ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]
[ TABLESPACE <tablespace> ]
{ NOPARALLEL | PARALLEL [ <integer> ] }
```

ここで、column_constraint は次のとおりです。

```
[ CONSTRAINT <constraint_name> ]
{ NOT NULL |
  NULL |
  UNIQUE [ USING INDEX TABLESPACE <tablespace> ] |
  PRIMARY KEY [ USING INDEX TABLESPACE <tablespace> ] |
  CHECK (<expression>) |
  REFERENCES <reftable> [ ( <refcolumn> ) ]
    [ ON DELETE <action> ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED |
  INITIALLY IMMEDIATE ]
```

そして table_constraint は:

```
[ CONSTRAINT <constraint_name> ]
{ UNIQUE ( <column_name> [, ...] )
  [ USING INDEX [ <create_index_statement> ] TABLESPACE <tablespace> ] |
```

(次のページに続く)

(前のページからの続き)

```

PRIMARY KEY ( <column_name> [, ...] )
    [ USING INDEX [ <create_index_statement> ] TABLESPACE <tablespace> ] |
CHECK ( <expression> ) |
FOREIGN KEY ( <column_name> [, ...] )
    REFERENCES <reftable> [ ( <refcolumn> [, ...] ) ]
    [ ON DELETE <action> ] }
[ DEFERRABLE | NOT DEFERRABLE ]
[ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]

```

Description

CREATE TABLE は、現在のデータベースに新しい空のテーブルを作成します。テーブルは、コマンドを発行するユーザーが所有します。

スキーマ名が指定されている場合 (CREATE TABLE myschema.mytable ... など)、指定されたスキーマにテーブルが作成されます。それ以外の場合は、現在のスキーマで作成されます。一時テーブルは特別なスキーマに存在するため、一時テーブルの作成時にスキーマ名が与えられない場合があります。テーブル名は、同じスキーマ内の他のテーブル、シーケンス、インデックス、またはビューの名前とは異なる必要があります。

CREATE TABLE は、テーブルの 1 行に対応する複合型を表すデータ型も自動的に作成します。したがって、同じスキーマの既存のデータ型と同じ名前をテーブルに付けることはできません。

PARALLEL 句は、テーブルの並列度を設定します。PARALLEL 句を指定しない場合、サーバーは関係サイズに基づいて値を決定します。

NOPARALLEL 句はデフォルト実行の並列性をリセットし、reloptions は parallel_workers パラメーターを 0 として表示します。

テーブルには 1600 列を超えることはできません。(実際には、タプル長の制約により、有効な制限は低くなります)。

オプションの制約句は、挿入または更新操作が成功するために新規または更新された行が満たす必要がある制約 (またはテスト) を指定します。制約は、さまざまな方法でテーブル内の有効な値のセットを定義するのに役立つ SQL オブジェクトです。

制約を定義するには、テーブル制約と列制約の 2 つの方法があります。列制約は、列定義の一部として定義されます。テーブル制約定義は特定の列に関連付けられておらず、複数の列を含むことができます。すべての列制約は、表制約として記述することもできます。制約が 1 つの列のみに影響する場合、列制約は表記上の便宜にすぎません。

Parameters

GLOBAL TEMPORARY

指定した場合、テーブルは一時テーブルとして作成されます。一時テーブルは、セッションの終了時に、またはオプションで現在のトランザクションの終了時に自動的に削除されます (以下の ON COMMIT を参

照)。同じ名前の既存の永続テーブルは、スキーマ修飾名で参照されない限り、一時テーブルが存在する間は現在のセッションからは見えません。また、一時テーブルは、作成されたセッションの外部からは見えません。（グローバル一時テーブルのこの側面は、Oracle データベースと互換性がありません。）一時テーブルで作成されたインデックスも自動的に一時的になります。

`table_name`

作成するテーブルの名前（オプションでスキーマ修飾）。

`column_name`

新しいテーブルに作成される列の名前。

`data_type`

列のデータ型。これには、配列指定子が含まれる場合があります。AdvancedServer に含まれるデータ型の詳細は、「OracleDevelopers リファレンス・ガイドのデータベース互換性*」のデータ型を参照してください。

`DEFAULT default_expr`

DEFAULT 句は、列定義が表示される列にデフォルトのデータ値を割り当てます。値は、変数のない式です（現在のテーブルの他の列へのサブクエリおよび相互参照は許可されていません）。デフォルト式のデータ型は、列のデータ型と一致する必要があります。

デフォルトの式は、列の値を指定しない挿入操作で使用されます。列にデフォルトがない場合、デフォルトは null です。

`CONSTRAINT constraint_name`

列またはテーブルの制約のオプション名。指定しない場合、システムは名前を生成します。

`NOT NULL`

列に NULL 値を含めることはできません。

`NULL`

列には NULL 値を含めることができます。これがデフォルトです。

この句は、非標準の SQL データベースとの互換性のためにのみ使用できます。新しいアプリケーションでは使用しないでください。

`UNIQUE` -列の制約

`UNIQUE (column_name [, ...] )` -テーブル制約

UNIQUE 制約は、テーブルの 1 つ以上の異なる列のグループに一意の値のみが含まれることを指定します。一意のテーブル制約の動作は列制約の動作と同じですが、複数の列にまたがる追加機能があります。

ユニーク制約の目的上、ヌル値は等しいとは見なされません。

各一意のテーブル制約は、テーブルに定義された他の一意または主キー制約によって指定された列のセットとは異なる列のセットに名前を付ける必要があります。（それ以外の場合は、2 回リストされた同じ制約になります。）

PRIMARY KEY -列の制約

PRIMARY KEY (column_name [, ...]) -テーブル制約

主キー制約は、1 つまたは複数のテーブルの列に一意の（重複しない）非 NULL 値のみが含まれることを指定します。技術的には、PRIMARY KEY は単に UNIQUE と NOT NULL の組み合わせですが、主キーは他のテーブルがこのセットに依存している可能性があるため、主キーとして列のセットを識別すると、スキーマの設計に関するメタデータも提供されます行の一意の識別子としての列の。

列制約としてもテーブル制約としても、テーブルには 1 つの主キーのみを指定できます。

主キー制約は、同じテーブルに定義された一意の制約によって名前が付けられた他の列セットとは異なる列セットに名前を付ける必要があります。

CHECK (expression)

CHECK 句は、挿入または更新操作が成功するために新規または更新された行が満たさなければならないブール結果を生成する式を指定します。TRUE または「不明」と評価される式は成功します。挿入または更新操作のいずれかの行が FALSE の結果を生成した場合、エラー例外が発生し、挿入または更新はデータベースを変更しません。列制約として指定されたチェック制約はその列の値のみを参照する必要がありますが、表制約に表示される式は複数の列を参照する場合があります。

現在、CHECK 式にはサブクエリを含めることも、現在の行の列以外の変数を参照することもできません。

REFERENCES reftable[(refcolumn `` ) ][ ``ON DELETE action]-列の制約

FOREIGN KEY (column [, ...]) REFERENCES reftable [(refcolumn [, ...])][ON DELETE action]-テーブル制約

これらの句は外部キー制約を指定します。これにより、新しいテーブルの 1 つ以上の列のグループには、参照されるテーブルのある行の参照される列の値と一致する値のみが含まれます。refcolumn を省略すると、reftable の主キーが使用されます。参照される列は、参照されるテーブルの一意または主キー制約の列である必要があります。

また、参照された列のデータが変更されると、この表の列のデータに対して特定のアクションが実行されます。ON DELETE 句は、参照先テーブルの参照先行行が削除されているときに実行するアクションを指定します。制約が遅延可能であっても、参照アクションを遅延させることはできません。各句の次の可能なアクションは次のとおりです。

CASCADE

削除された行を参照する行を削除するか、参照する列の値を参照される列の新しい値にそれぞれ更新します。

SET NULL

参照列を NULL に設定します。

参照される列が頻繁に変更される場合、外部キー列に関連付けられた参照アクションをより効率的に実行できるように、外部キー列にインデックスを追加するのが賢明です。

DEFERRABLE

NOT DEFERRABLE

これは、制約を延期できるかどうかを制御します。遅延可能ではない制約は、すべてのコマンドの直後にチェックされます。遅延可能な制約のチェックは、トランザクションの終了まで延期される場合があります (SET CONSTRAINTS コマンドを使用)。NOT DEFERRABLE がデフォルトです。現在、この句を受け入れるのは外部キー制約のみです。他のすべての制約タイプは遅延可能ではありません。

INITIALLY IMMEDIATE

INITIALLY DEFERRED

制約が遅延可能な場合、この句は制約をチェックするデフォルトの時間を指定します。制約が INITIALLY IMMEDIATE の場合、各ステートメントの後にチェックされます。これがデフォルトです。制約が INITIALLY DEFERRED の場合、トランザクションの最後でのみチェックされます。制約チェック時間は SET CONSTRAINTS コマンドで変更できます。

WITH (ROWIDS [= value])

テーブルの ROWIDS オプションには、TRUE/ON/1 または FALSE/OFF/0 に等しい value が含まれます。TRUE/ON/1 に設定すると、ROWID 列が新しいテーブルに作成されます。ROWID は、1 で始まりテーブルの各行に割り当てられるシーケンスに基づく自動インクリメント値です。値が指定されていない場合、デフォルト値は常に TRUE です。

デフォルトでは、ROWID 列に一意のインデックスが作成されます。ALTER および DROP 操作は ROWID 列で制限されています。

ON COMMIT

トランザクションブロックの終了時の一時テーブルの動作は、ON COMMIT を使用して制御できます。2 つのオプションは次のとおりです。

PRESERVE ROWS

トランザクションの終了時に特別なアクションは行われません。これがデフォルトの動作です。(この側面は Oracle データベースと互換性がないことに注意してください。Oracle のデフォルトは DELETE ROWS です。)

DELETE ROWS

一時テーブルのすべての行は、各トランザクションブロックの最後に削除されます。基本的に、自動 TRUNCATE は各コミットで行われます。

TABLESPACE tablespace

tablespace は、新しいテーブルが作成されるテーブルスペースの名前です。指定しない場合は default tablespace が使用され、default_tablespace が空の文字列の場合はデータベースのデフォルトのテーブルスペースが使用されます。

USING INDEX [create_index_statement] TABLESPACE tablespace

この句を使用すると、UNIQUE または PRIMARY KEY 制約に関連付けられたインデックスが作成されるテーブルスペースを選択できます。指定しない場合は default tablespace が使用され、default_tablespace が空の文字列の場合はデータベースのデフォルトのテーブルスペースが使用されます。

create_index_statement オプションを指定すると、データベースサーバーは、一意または主キー制約を有効にするインデックスを作成します。制約で指定された列とインデックスの列は同じである必要がありますが、それらの表示順序は異なる場合があります。

PARALLEL

PARALLEL 句を含めて、テーブルの並列度を指定します。parallel_workers パラメータを並列度に等しく設定して、テーブルの並列スキャンを実行します。または、PARALLEL を指定し、ある程度の並列処理を含めない場合、インデックスはデフォルトの並列処理を使用します。

NOPARALLEL

デフォルトの実行には NOPARALLEL を指定します。

integer

integer は並列度を示します。これは、テーブルで並列スキャンを実行する並列操作で使用する parallel_workers の数です。

Notes

AdvancedServer は、一意性を強制するために、一意性制約と主キー制約ごとにインデックスを自動的に作成します。したがって、主キー列の明示的なインデックスを作成する必要はありません。(詳細については、CREATE INDEX を参照してください。)

Examples

テーブル dept とテーブル emp を作成します。

```
CREATE TABLE dept (
    deptno      NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
    dname       VARCHAR2(14),
    loc         VARCHAR2(13)
);
CREATE TABLE emp (
    empno       NUMBER(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
    ename       VARCHAR2(10),
    job         VARCHAR2(9),
    mgr         NUMBER(4),
    hiredate    DATE,
    sal         NUMBER(7,2),
    comm        NUMBER(7,2),
    deptno      NUMBER(2) CONSTRAINT emp_ref_dept_fk
                REFERENCES dept(deptno)
);
```

テーブル dept に一意のテーブル制約を定義します。一意のテーブル制約は、テーブルの 1 つ以上の列で定義できます。

```
CREATE TABLE dept (
    deptno      NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
    dname       VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
    loc         VARCHAR2(13)
);
```

チェック列制約を定義します。

```
CREATE TABLE emp (
    empno       NUMBER(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
    ename       VARCHAR2(10),
    job         VARCHAR2(9),
    mgr         NUMBER(4),
    hiredate    DATE,
    sal         NUMBER(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
    comm        NUMBER(7,2),
    deptno      NUMBER(2) CONSTRAINT emp_ref_dept_fk
                REFERENCES dept(deptno)
);
```

チェックテーブル制約を定義します。

```
CREATE TABLE emp (
    empno       NUMBER(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
    ename       VARCHAR2(10),
    job         VARCHAR2(9),
    mgr         NUMBER(4),
```

(次のページに続く)

(前のページからの続き)

```

hiredate      DATE,
sal           NUMBER(7,2),
comm         NUMBER(7,2),
deptno       NUMBER(2) CONSTRAINT emp_ref_dept_fk
              REFERENCES dept(deptno),
CONSTRAINT new_emp_ck CHECK (ename IS NOT NULL AND empno > 7000)
);

```

テーブル jobhist の主キーテーブル制約を定義します。主キーテーブルの制約は、テーブルの 1 つ以上の列で定義できます。

```

CREATE TABLE jobhist (
  empno          NUMBER(4) NOT NULL,
  startdate      DATE NOT NULL,
  enddate        DATE,
  job            VARCHAR2(9),
  sal            NUMBER(7,2),
  comm          NUMBER(7,2),
  deptno         NUMBER(2),
  chgdesc        VARCHAR2(80),
  CONSTRAINT jobhist_pk PRIMARY KEY (empno, startdate)
);

```

これにより、列にリテラル定数のデフォルト値 job が割り当てられ、hiredate のデフォルト値が行が挿入される日付になります。

```

CREATE TABLE emp (
  empno          NUMBER(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
  ename          VARCHAR2(10),
  job            VARCHAR2(9) DEFAULT 'SALESMAN',
  mgr            NUMBER(4),
  hiredate       DATE DEFAULT SYSDATE,
  sal            NUMBER(7,2),
  comm          NUMBER(7,2),
  deptno         NUMBER(2) CONSTRAINT emp_ref_dept_fk
              REFERENCES dept(deptno)
);

```

テーブルスペース diskvol1 にテーブル dept を作成します。

```

CREATE TABLE dept (
  deptno         NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
  dname          VARCHAR2(14),
  loc            VARCHAR2(13)
) TABLESPACE diskvol1;

```

次の PARALLEL の例は、テーブル sales を作成し、並列度を 6 に設定します。

```
CREATE TABLE sales (deptno number) PARALLEL 6 WITH (FILLFACTOR=66);
```

次の NOPARALLEL の例は、テーブル sales_order を作成し、並列度を 0 に設定します。

```
CREATE TABLE sales_order (deptno number) NOPARALLEL WITH (FILLFACTOR=66);
```

次の例では、dept という名前のテーブルを作成します。定義により、dname 列に一意のキーが作成されます。制約 dept_dname_uq は dname 列を一意のキーとして識別します。上記のステートメントには、USING_INDEX 句が含まれています。これは、一意の制約を有効にするために指定されたインデックスステートメントを使用して、テーブル dept にインデックスを明示的に作成します。

```
CREATE TABLE dept (
    deptno          NUMBER(2) NOT NULL,
    dname           VARCHAR2(14),
    loc            VARCHAR2(13),
    CONSTRAINT dept_dname_uq UNIQUE(dname)
        USING INDEX (CREATE UNIQUE INDEX idx_dept_dname_uq ON dept(dname))
);
```

次の例では、emp という名前のテーブルを作成します。定義により、ename 列に主キーが作成されます。emp_ename_pk 制約は、ename 列を emp テーブルの主キーとして識別します。上記のステートメントには USING_INDEX 句が含まれています。これは、プライマリ制約を有効にするために指定されたインデックスステートメントを使用して、テーブル emp に明示的にインデックスを作成します。

```
CREATE TABLE emp (
    empno          NUMBER(4) NOT NULL,
    ename          VARCHAR2(10),
    job            VARCHAR2(9),
    sal            NUMBER(7,2),
    deptno         NUMBER(2),
    CONSTRAINT emp_ename_pk PRIMARY KEY (ename)
        USING INDEX (CREATE INDEX idx_emp_ename_pk ON emp(ename))
);
```

See Also

[ALTERTABLE](#)、[DROPTABLE](#)

第 37 章

CREATE TABLE AS

Name

CREATE TABLE AS -- はクエリの結果から新しいテーブルを定義します

Synopsis

```
CREATE [ GLOBAL TEMPORARY ] TABLE <table_name>
  [ ( <column_name> [, ...] ) ]
  [ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]
  [ TABLESPACE tablespace ]
  AS <query>
```

Description

CREATE TABLE AS はテーブルを作成し、SELECT コマンドで計算されたデータをそのテーブルに入力します。テーブルの列には、SELECT の出力列に関連付けられた名前とデータ型があります（ただし、新しい列名の明示的なリストを指定することで列名をオーバーライドできます）。

CREATE TABLE AS はビューの作成と似ていますが、実際にはまったく異なります。新しいテーブルを作成し、クエリを一度だけ評価して新しいテーブルに最初に入力します。新しいテーブルは、クエリのソーステーブルに対するその後の変更を追跡しません。対照的に、ビューは、クエリが実行されるたびに、定義する SELECT ステートメントを再評価します。

Parameters

GLOBAL TEMPORARY

指定した場合、テーブルは一時テーブルとして作成されます。詳細については、CREATE TABLE を参照してください。

table_name

作成するテーブルの名前（オプションでスキーマ修飾）。

`column_name`

新しいテーブルの列の名前。列名が指定されていない場合、クエリの出力列名から取得されます。

`query`

クエリステートメント（`SELECT` コマンド）。許可される構文の説明については、`SELECT` を参照してください。

第 38 章

CREATE TRIGGER

Name

CREATE TRIGGER -- は単純なトリガーを定義します

Synopsis

```
CREATE [ OR REPLACE ] TRIGGER <name>
  { BEFORE | AFTER | INSTEAD OF }
  { INSERT | UPDATE | DELETE | TRUNCATE }
    [ OR { INSERT | UPDATE | DELETE | TRUNCATE } ] [, ...]
  ON <table>
  [ REFERENCING { OLD AS <old> | NEW AS <new> } ...]
  [ FOR EACH ROW ]
  [ WHEN <condition> ]
  [ DECLARE
    [ PRAGMA AUTONOMOUS_TRANSACTION; ]
    <declaration>; [, ...] ]
  BEGIN
    <statement>; [, ...]
  [ EXCEPTION
    { WHEN <exception> [ OR <exception> ] [...] THEN
      <statement>; [, ...] } [, ...]
  ]
  END
```

Name

CREATE TRIGGER -- は複合トリガーを定義します

Synopsis

```
CREATE [ OR REPLACE ] TRIGGER <name>
  FOR { INSERT | UPDATE | DELETE | TRUNCATE }
    [ OR { INSERT | UPDATE | DELETE | TRUNCATE } ] [, ...]
```

(次のページに続く)

(前のページからの続き)

```

    ON <table>
    [ REFERENCING { OLD AS <old> | NEW AS <new> } ... ]
    [ WHEN <condition> ]
    COMPOUND TRIGGER
    [ <private_declaration>; ] ...
    [ <procedure_or_function_definition> ] ...
    <compound_trigger_definition>
    END

```

private_declaration は、任意のプロシージャまたは関数からアクセスできるプライベート変数の識別子です。0、1、またはそれ以上のプライベート変数があります。private_declaration は次のいずれかになります。

- 可変宣言
- 記録宣言
- コレクション宣言
- REF CURSOR とカーソル変数宣言
- TYPE のレコード、コレクション、および REF CURSORs の定義
- 例外
- オブジェクト変数宣言

ここで、procedure_or_function_definition :=

```
procedure_definition | function_definition
```

ここで、procedure_definition :=

```

PROCEDURE proc_name[ argument_list ]
    [ options_list ]
    { IS | AS }
    procedure_body
    END [ proc_name ];

```

ここで、procedure_body :=

```

[ declaration; ] [, ...]
BEGIN
    statement; [...]
[ EXCEPTION
    { WHEN exception [OR exception] [...] ] THEN statement; }
    [...]
]

```

ここで、function_definition:=

```
FUNCTION func_name [ argument_list ]
  RETURN rettype [ DETERMINISTIC ]
  [ options_list ]
  { IS | AS }
  function_body
END [ func_name ] ;
```

ここで、function_body:=

```
[ declaration; ] [, ...]
BEGIN
  statement; [...]
[ EXCEPTION
  { WHEN exception [ OR exception ] [...] THEN statement; }
  [...]
]
```

ここで、compound_trigger_definition:=

```
{ compound_trigger_event } { IS | AS }
  compound_trigger_body
END [ compound_trigger_event ] [ ... ]
```

ここで、compound_trigger_event:=

```
[ BEFORE STATEMENT | BEFORE EACH ROW | AFTER EACH ROW |
  AFTER STATEMENT | INSTEAD OF EACH ROW ]
```

ここで、compound_trigger_body:=

```
[ declaration; ] [, ...]
BEGIN
  statement; [...]
[ EXCEPTION
  { WHEN exception [OR exception] [...] THEN statement; }
  [...]
]
```

Description

CREATE TRIGGER は新しいトリガーを定義します。CREATE OR REPLACE TRIGGER は、新しいトリガーを作成するか、既存の定義を置き換えます。

CREATE TRIGGER キーワードを使用して新しいトリガーを作成する場合、新しいトリガーの名前は、同じテーブルで定義されている既存のトリガーと一致してはなりません。新しいトリガーは、トリガーイベントが定義されているテーブルと同じスキーマに作成されます。

既存のトリガーの定義を更新する場合は、`CREATE OR REPLACE TRIGGER` キーワードを使用します。

Oracle と互換性のある構文を使用してトリガーを作成すると、トリガーは `SECURITY DEFINER` 関数として実行されます。

Parameters

`name`

作成するトリガーの名前。

`BEFORE` | `AFTER`

トリガーイベントの前または後にトリガーを起動するかどうかを決定します。

`INSTEAD OF`

`INSTEAD OF` トリガーは更新可能なビューを変更します。トリガーが実行され、基になるテーブルが適切に更新されます。 `INSTEAD OF` トリガーは、更新または変更されたビューの行ごとに実行されます。

`INSERT` | `UPDATE` | `DELETE` | `TRUNCATE`

トリガーイベントを定義します。

`table`

トリガーイベントが発生するテーブルまたはビューの名前。

`condition`

`condition` は、トリガーが実際に実行されるかどうかを決定するブール式です。 `condition` が `TRUE` と評価されると、トリガーが起動します。

単純なトリガー定義に `FOR EACH ROW` キーワードが含まれる場合、`WHEN` 句は `OLD.column_name` または `NEW.column_name` をそれぞれ書き込むことにより、古い行値または新しい行値、あるいはその両方の列を参照できます。 `INSERT` トリガーは `OLD` を参照できず、`DELETE` トリガーは `NEW` を参照できません。

複合トリガー定義に `WHEN` 句を持つ文レベルのトリガーが含まれている場合、`WHEN` 句の式を評価せずにトリガーが実行されます。同様に、複合トリガー定義に `WHEN` 句を持つ行レベルのトリガーが含まれている場合、式が `TRUE` と評価されるとトリガーが実行されます。

トリガーに `INSTEAD OF` キーワードが含まれる場合、`WHEN` 句が含まれない場合があります。 `WHEN` 句にサブクエリを含めることはできません。

`REFERENCING { OLD AS old | NEW AS new } ...`

古い行と新しい行を参照するための `REFERENCING` 句。ただし、`old` は `old` またはすべて小文字で保存される同等の名前の識別子（たとえば、`REFERENCING OLD AS old`、`REFERENCING OLD AS OLD`、または `REFERENCING OLD AS /"old/"`）。また、`new` は、`new` という名前の識別子またはすべて

小文字で保存される同等の識別子（REFERENCING NEW AS new、REFERENCING NEW AS NEW、REFERENCING NEW AS /"new/" など）でのみ置き換えることができます。

OLD AS old と NEW AS new のいずれか、または両方のフレーズを REFERENCING 句で指定できます（REFERENCING NEW AS New OLD AS Old など）。

この句は、old または new 以外の識別子を使用できないため、Oracle データベースと互換性がありません。

FOR EACH ROW

トリガーイベントによって影響を受けるすべての行に対してトリガーを 1 回起動するか、SQL ステートメントごとに 1 回だけ起動するかを決定します。指定した場合、トリガーは影響を受ける行ごとに 1 回起動されます（行レベルトリガー）。それ以外の場合、トリガーはステートメントレベルのトリガーです。

PRAGMA AUTONOMOUS_TRANSACTION

PRAGMA AUTONOMOUS_TRANSACTION は、トリガーを自律型トランザクションとして設定するディレクティブです。

declaration

変数、タイプ、REF CURSOR、またはサブプログラム宣言。サブプログラム宣言が含まれる場合、他のすべての変数、型、および REF CURSOR 宣言の後に宣言する必要があります。

statement

SPL プログラムステートメント。DECLARE - BEGIN - END ブロックは、それ自体が SPL ステートメントと見なされることに注意してください。したがって、トリガー本体にはネストされたブロックが含まれる場合があります。

exception

NO_DATA_FOUND、OTHERS などの例外条件名

Examples

以下は、トリガーステートメント（テーブル emp での挿入、更新、または削除）が実行された後に起動するステートメントレベルのトリガーです。

```
CREATE OR REPLACE TRIGGER user_audit_trig
  AFTER INSERT OR UPDATE OR DELETE ON emp
DECLARE
  v_action          VARCHAR2(24);
BEGIN
  IF INSERTING THEN
    v_action := ' added employee(s) on ';
  ELSIF UPDATING THEN
    v_action := ' updated employee(s) on ';
  ELSIF DELETING THEN
```

(次のページに続く)

(前のページからの続き)

```

        v_action := ' deleted employee(s) on ';
    END IF;
    DBMS_OUTPUT.PUT_LINE('User ' || USER || v_action ||
        TO_CHAR(SYSDATE, 'YYYY-MM-DD'));
END;
```

以下は、テーブル emp で各行が挿入、更新、または削除される前に起動する行レベルトリガーです。

```

CREATE OR REPLACE TRIGGER emp_sal_trig
    BEFORE DELETE OR INSERT OR UPDATE ON emp
    FOR EACH ROW
DECLARE
    sal_diff      NUMBER;
BEGIN
    IF INSERTING THEN
        DBMS_OUTPUT.PUT_LINE('Inserting employee ' || :NEW.empno);
        DBMS_OUTPUT.PUT_LINE('..New salary: ' || :NEW.sal);
    END IF;
    IF UPDATING THEN
        sal_diff := :NEW.sal - :OLD.sal;
        DBMS_OUTPUT.PUT_LINE('Updating employee ' || :OLD.empno);
        DBMS_OUTPUT.PUT_LINE('..Old salary: ' || :OLD.sal);
        DBMS_OUTPUT.PUT_LINE('..New salary: ' || :NEW.sal);
        DBMS_OUTPUT.PUT_LINE('..Raise      : ' || sal_diff);
    END IF;
    IF DELETING THEN
        DBMS_OUTPUT.PUT_LINE('Deleting employee ' || :OLD.empno);
        DBMS_OUTPUT.PUT_LINE('..Old salary: ' || :OLD.sal);
    END IF;
END;
```

以下は、表 emp に複合トリガー hr_trigger を定義することにより、従業員の給与の変更を記録する複合トリガーの例です。

まず、emp という名前のテーブルを作成します。

```

CREATE TABLE emp(EMPNO INT, ENAME TEXT, SAL INT, DEPTNO INT);
CREATE TABLE
```

次に、hr_trigger という名前の複合トリガーを作成します。トリガーは、4つのタイミングポイントのそれぞれを使用して、INSERT、UPDATE、またはDELETE ステートメントで給与を変更します。グローバル宣言セクションでは、初期給与は 10,000 と宣言されています。

```

CREATE OR REPLACE TRIGGER hr_trigger
    FOR INSERT OR UPDATE OR DELETE ON emp
    COMPOUND TRIGGER
```

(次のページに続く)

(前のページからの続き)

```

-- Global declaration.
var_sal NUMBER := 10000;

BEFORE STATEMENT IS
BEGIN
    var_sal := var_sal + 1000;
    DBMS_OUTPUT.PUT_LINE('Before Statement: ' || var_sal);
END BEFORE STATEMENT;

BEFORE EACH ROW IS
BEGIN
    var_sal := var_sal + 1000;
    DBMS_OUTPUT.PUT_LINE('Before Each Row: ' || var_sal);
END BEFORE EACH ROW;

AFTER EACH ROW IS
BEGIN
    var_sal := var_sal + 1000;
    DBMS_OUTPUT.PUT_LINE('After Each Row: ' || var_sal);
END AFTER EACH ROW;

AFTER STATEMENT IS
BEGIN
    var_sal := var_sal + 1000;
    DBMS_OUTPUT.PUT_LINE('After Statement: ' || var_sal);
END AFTER STATEMENT;

END hr_trigger;

Output: Trigger created.

```

INSERT レコードをテーブル emp に入れます。

```
INSERT INTO emp (EMPNO, ENAME, SAL, DEPTNO) VALUES (1111, 'SMITH', 10000, 20);
```

INSERT ステートメントは、次の出力を生成します。

```

Before Statement: 11000
Before each row: 12000
After each row: 13000
After statement: 14000
INSERT 0 1

```

UPDATE ステートメントは従業員の給与レコードを更新し、特定の従業員番号の給与を 15000 に設定します。

```
UPDATE emp SET SAL = 15000 where EMPNO = 1111;
```

UPDATE ステートメントは、次の出力を生成します。

```
Before Statement: 11000
Before each row: 12000
After each row: 13000
After statement: 14000
UPDATE 1

SELECT * FROM emp;
  EMPNO | ENAME |  SAL  | DEPTNO
-----+-----+-----+-----
    1111 | SMITH | 15000 |      20
(1 row)
```

DELETE ステートメントは、従業員の給与レコードを削除します。

```
DELETE from emp where EMPNO = 1111;
```

DELETE ステートメントは、次の出力を生成します。

```
Before Statement: 11000
Before each row: 12000
After each row: 13000
After statement: 14000
DELETE 1

SELECT * FROM emp;
  EMPNO | ENAME |  SAL  | DEPTNO
-----+-----+-----+-----
(0 rows)
```

TRUNCATE ステートメントは、emp テーブルからすべてのレコードを削除します。

```
CREATE OR REPLACE TRIGGER hr_trigger
FOR TRUNCATE ON emp
  COMPOUND TRIGGER
-- Global declaration.
var_sal NUMBER := 10000;
BEFORE STATEMENT IS
BEGIN
  var_sal := var_sal + 1000;
  DBMS_OUTPUT.PUT_LINE('Before Statement: ' || var_sal);
END BEFORE STATEMENT;

AFTER STATEMENT IS
BEGIN
  var_sal := var_sal + 1000;
  DBMS_OUTPUT.PUT_LINE('After Statement: ' || var_sal);
```

(次のページに続く)

(前のページからの続き)

```
END AFTER STATEMENT;

END hr_trigger;

Output: Trigger created.
```

TRUNCATE ステートメントは、次の出力を生成します。

```
TRUNCATE emp;
Before Statement: 11000
After statement: 12000
TRUNCATE TABLE
```

Note: TRUNCATE ステートメントは、BEFORE STATEMENT または AFTER STATEMENT のタイミングポイントでのみ使用できます。

次の例では、emp テーブルに hr_trigger という名前の複合トリガーを作成し、INSERT、UPDATE、または DELETE ステートメントが emp テーブルに影響を与えるたびに従業員の給与をチェックおよび印刷する「10」条件を設定します。データベースは、行レベルトリガーの「10」条件を評価し、「10」条件が TRUE と評価された場合、トリガーは行ごとに 1 回実行されます。ステートメントレベルのトリガーは、「10」条件に関係なく実行されます。

```
CREATE OR REPLACE TRIGGER hr_trigger
FOR INSERT OR UPDATE OR DELETE ON emp
REFERENCING NEW AS new OLD AS old
WHEN (old.sal > 5000 OR new.sal < 8000)
COMPOUND TRIGGER

BEFORE STATEMENT IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Before Statement');
END BEFORE STATEMENT;

BEFORE EACH ROW IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Before Each Row: ' || :OLD.sal || ' ' || :NEW.sal);
END BEFORE EACH ROW;

AFTER EACH ROW IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('After Each Row: ' || :OLD.sal || ' ' || :NEW.sal);
END AFTER EACH ROW;

AFTER STATEMENT IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('After Statement');
```

(次のページに続く)

(前のページからの続き)

```
END AFTER STATEMENT;

END hr_trigger;
```

レコードをテーブル emp に挿入します。

```
INSERT INTO emp(EMPNO, ENAME, SAL, DEPTNO) VALUES(1111, 'SMITH', 1600, 20);
```

INSERT ステートメントは、次の出力を生成します。

```
Before Statement
Before Each Row: 1600
After Each Row: 1600
After Statement
INSERT 0 1
```

UPDATE ステートメントは、従業員の給与レコードを更新し、給与を 7500 に設定します。

```
UPDATE emp SET SAL = 7500 where EMPNO = 1111;
```

UPDATE ステートメントは、次の出力を生成します。

```
Before Statement
Before Each Row: 1600 7500
After Each Row: 1600 7500
After Statement
UPDATE 1

SELECT * from emp;
 empno | ename | sal  | deptno
-----+-----+-----+-----
  1111 | SMITH | 7500 |      20
(1 row)
```

DELETE ステートメントは、従業員の給与レコードを削除します。

```
DELETE from emp where EMPNO = 1111;
```

DELETE ステートメントは、次の出力を生成します。

```
Before Statement
Before Each Row: 7500
After Each Row: 7500
After Statement
DELETE 1
```

(次のページに続く)

(前のページからの続き)

```
SELECT * from emp;  
 empno |  ename  |  sal  | deptno  
-----+-----+-----+-----  
(0 rows)
```

See Also*ALTERTRIGGER*、*DROPTRIGGER*

第 39 章

CREATE TYPE

Name

CREATE TYPE -- は、新しいユーザー定義型を定義します。これは、オブジェクト型、コレクション型（ネストされたテーブル型または VARRAY 型）、または複合型にすることができます。

Synopsis

Object Type

```
CREATE [ OR REPLACE ] TYPE <name>
  [ AUTHID { DEFINER | CURRENT_USER } ]
  { IS | AS } OBJECT
  ( { <attribute> { <datatype> | <objtype> | <collecttype> } }
    [, ...]
    [ <method_spec> ] [, ...]
  ) [ [ NOT ] { FINAL | INSTANTIABLE } ] ...
```

ここで、method_spec は次のとおりです。

```
[ [ NOT ] { FINAL | INSTANTIABLE } ] ...
[ OVERRIDING ]
  <subprogram_spec>
```

そして subprogram_spec は:

```
{ MEMBER | STATIC }
{ PROCEDURE <proc_name>
  [ ( [ SELF [ IN | IN OUT ] <name> ]
    [, <argname> [ IN | IN OUT | OUT ] <argtype>
      [ DEFAULT <value> ]
    ] ...)
  ]
|
```

(次のページに続く)

(前のページからの続き)

```

FUNCTION <func_name>
  [ ( [ SELF [ IN | IN OUT ] <name> ]
    [ , <argname> [ IN | IN OUT | OUT ] <argtype>
      [ DEFAULT <value> ]
    ] ... )
  ]
RETURN <rettype>
}

```

Nested Table Type

```

CREATE [ OR REPLACE ] TYPE <name> { IS | AS } TABLE OF
  { <datatype> | <objtype> | <collecttype> }

```

Varray Type

```

CREATE [ OR REPLACE ] TYPE <name> { IS | AS }
  { VARRAY | VARYING ARRAY } (<maxsize>) OF { <datatype> | <objtype> }

```

Composite Type

```

CREATE [ OR REPLACE ] TYPE <name> { IS | AS }
( [ attribute <datatype> ] [ , ... ]
)

```

Description

CREATE TYPE は、新しいユーザー定義のデータ型を定義します。作成できる型は、オブジェクト型、ネストした表型、VARRAY 型または複合型です。ネストした表および VARRAY 型は、collections として知られる型のカテゴリに属します。

Composite types は Oracle データベースと互換性がありません。ただし、このセクションで説明する他のタイプと同様に、SPL プログラムから複合タイプにアクセスできます。

注釈:

- パッケージの場合のみ、複合型は、パッケージ仕様またはパッケージ本体内の TYPE IS RECORD ステートメントで宣言されたユーザー定義のレコード型に含めることができます。このようなネストされた構造は、関数、プロシージャ、トリガーなどの他の SPL プログラムでは許可されていません。
- CREATE TYPE コマンドで、スキーマ名が含まれている場合、指定されたスキーマでタイプが作成されます。それ以外の場合、現在のスキーマで作成されます。新しい型の名前は、既存の型の定義を更新することを意図している場合を除き、同じスキーマ内の既存の型と一致してはなりません。その場合、CREATE OR REPLACE TYPE を使用します。

- 現在、OR REPLACE オプションを使用して、既存のオブジェクトタイプの属性を追加、削除、または変更することはできません。DROP TYPE コマンドを使用して、最初に既存のオブジェクトタイプを削除します。OR REPLACE オプションを使用して、既存のオブジェクトタイプのメソッドを追加、削除、または変更できます。
- ALTER TYPE ALTER ATTRIBUTE コマンドの PostgreSQL 形式を使用して、既存のオブジェクトタイプの属性のデータタイプを変更できます。ただし、ALTER TYPE コマンドはオブジェクトタイプの属性を追加または削除できません。

タイプを作成したユーザーがタイプの所有者になります。

Parameters

name

作成するタイプの名前（オプションでスキーマ修飾）。

DEFINER | CURRENT_USER

オブジェクトタイプの所有者の特権（DEFINER）またはオブジェクトタイプのメソッドを実行している現在のユーザーの特権（CURRENT_USER）を使用して、参照されるデータベースオブジェクトへのアクセスを許可するかどうかを決定します。オブジェクトタイプ。DEFINER がデフォルトです。

attribute

オブジェクト型または複合型の属性の名前。

datatype

オブジェクト型または複合型の属性、または作成されるコレクション型の要素を定義するデータ型。

objtype

オブジェクトタイプの属性または作成されるコレクションタイプの要素を定義するオブジェクトタイプの名前。

collecttype

オブジェクトタイプの属性または作成されるコレクションタイプの要素を定義するコレクションタイプの名前。

FINAL

NOT FINAL

オブジェクトタイプの場合、サブタイプをオブジェクトタイプから派生できるかどうかを指定します。FINAL（サブタイプはオブジェクトタイプから派生できません）がデフォルトです。

method_spec の場合、メソッドをサブタイプでオーバーライドできるかどうかを指定します。NOT FINAL（メソッドはサブタイプでオーバーライドされる場合があります）がデフォルトです。

INSTANTIABLE

NOT INSTANTIABLE

オブジェクトタイプの場合、このオブジェクトタイプのオブジェクトインスタンスを作成できるかどうかを指定します。INSTANTIABLE（このオブジェクトタイプのインスタンスを作成できます）がデフォルトです。NOT INSTANTIABLE を指定する場合、NOT FINAL も指定する必要があります。オブジェクトタイプのいずれかのメソッドの method_spec に NOT INSTANTIABLE 修飾子が含まれる場合、オブジェクトタイプ自体は、オブジェクトタイプ仕様の右括弧に続く NOT INSTANTIABLE および NOT FINAL で定義する必要があります。

method_spec の場合、オブジェクトタイプ定義がメソッドの実装を提供するかどうかを指定します。INSTANTIABLE（オブジェクトタイプの CREATE TYPE BODY コマンドがメソッドの実装を提供する）がデフォルトです。NOT INSTANTIABLE が指定されている場合、オブジェクトタイプの CREATE TYPE BODY コマンドにメソッドの実装を含めることはできません。

OVERRIDING

OVERRIDING が指定されている場合、method_spec は、同じデータ型、同じ順序、同じ戻り型（メソッドが関数の場合）で定義された同じ名前の同じメソッド引数の数を持つ同じ名前のメソッドをオーバーライドしますスーパータイプで。

MEMBER

STATIC

サブプログラムがオブジェクトインスタンスで動作する場合は、MEMBER を指定します。サブプログラムが特定のオブジェクトインスタンスとは独立して動作する場合は、STATIC を指定します。

proc_name

作成するプロシージャの名前。

SELF [IN | IN OUT] name

メンバーメソッドの場合、データ型が定義されているオブジェクト型のデータ型である SELF という名前の暗黙の組み込みパラメーターがあります。SELF は、現在メソッドを呼び出しているオブジェクトインスタンスを指します。SELF は、パラメータリストで IN または IN OUT パラメータとして明示的に宣言できます。明示的に宣言する場合、SELF をパラメータリストの最初のパラメーターにする必要があります。SELF が明示的に宣言されていない場合、そのパラメータモードのデフォルトは、メンバープロシージャでは IN OUT、メンバー関数では IN です。

argname

引数の名前。引数は、メソッド本体内でこの名前によって参照されます。

argtype

メソッドの引数のデータ型。引数の型は、基本データ型でも、ネストした表やオブジェクト型などのユーザー定義型でもかまいません。ベースタイプには長さを指定しないでください。たとえば、`VARCHAR2(10)`ではなく`VARCHAR2`を指定します。

DEFAULT value

メソッド呼び出しで入力引数が指定されていない場合、入力引数のデフォルト値を指定します。モード `IN OUT` または `OUT` の引数には `DEFAULT` を指定できません。

func_name

作成する関数の名前。

rettype

戻りデータ型。`argtype` にリストされている型のいずれかです。`argtype` については、`rettype` の長さを指定しないでください。

maxsize

`VARRAY` で許可される要素の最大数。

Examples

Creating an Object Type

オブジェクトタイプ `addr_obj_typ` を作成します。

```
CREATE OR REPLACE TYPE addr_obj_typ AS OBJECT (
    street      VARCHAR2(30),
    city        VARCHAR2(20),
    state       CHAR(2),
    zip         NUMBER(5)
);
```

メンバーメソッド `display_emp` を含むオブジェクトタイプ `emp_obj_typ` を作成します。

```
CREATE OR REPLACE TYPE emp_obj_typ AS OBJECT (
    empno       NUMBER(4),
    ename       VARCHAR2(20),
    addr        ADDR_OBJ_TYP,
    MEMBER PROCEDURE display_emp (SELF IN OUT emp_obj_typ)
);
```

静的メソッド `get_dname` を含むオブジェクトタイプ `dept_obj_typ` を作成します。

```
CREATE OR REPLACE TYPE dept_obj_typ AS OBJECT (
    deptno      NUMBER(2),
    STATIC FUNCTION get_dname (p_deptno IN NUMBER) RETURN VARCHAR2,
```

(次のページに続く)

(前のページからの続き)

```
MEMBER PROCEDURE display_dept
);
```

Creating a Collection Type

データ型 NUMBER(8,2) のネストされたテーブル型 budget_ttbl_typ を作成します。

```
CREATE OR REPLACE TYPE budget_ttbl_typ IS TABLE OF NUMBER(8,2);
```

Creating and Using a Composite Type

次の例は、匿名ブロックからアクセスされる複合型の使用法を示しています。

複合タイプは、次によって作成されます。

```
CREATE OR REPLACE TYPE emphist_typ AS (
    empno          NUMBER(4),
    ename          VARCHAR2(10),
    hiredate       DATE,
    job            VARCHAR2(9),
    sal            NUMBER(7,2)
);
```

以下は、複合型にアクセスする匿名ブロックです。

```
DECLARE
    v_emphist      EMPHIST_TYP;
BEGIN
    v_emphist.empno    := 9001;
    v_emphist.ename    := 'SMITH';
    v_emphist.hiredate := '01-AUG-17';
    v_emphist.job      := 'SALESMAN';
    v_emphist.sal      := 8000.00;
    DBMS_OUTPUT.PUT_LINE('  EMPNO: ' || v_emphist.empno);
    DBMS_OUTPUT.PUT_LINE('  ENAME: ' || v_emphist.ename);
    DBMS_OUTPUT.PUT_LINE('HIREDATE: ' || v_emphist.hiredate);
    DBMS_OUTPUT.PUT_LINE('    JOB: ' || v_emphist.job);
    DBMS_OUTPUT.PUT_LINE('    SAL: ' || v_emphist.sal);
END;

EMPNO: 9001
ENAME: SMITH
HIREDATE: 01-AUG-17 00:00:00
    JOB: SALESMAN
    SAL: 8000.00
```

次の例は、パッケージ本体内で宣言されたユーザー定義のレコード型からアクセスされる複合型の使用法を示しています。

複合タイプは、次によって作成されます。

```
CREATE OR REPLACE TYPE salhist_typ AS (
    startdate      DATE,
    job            VARCHAR2(9),
    sal            NUMBER(7,2)
);
```

パッケージ仕様は、次によって定義されます。

```
CREATE OR REPLACE PACKAGE emp_salhist
IS
    PROCEDURE fetch_emp (
        p_empno      IN NUMBER
    );
END;
```

パッケージ本体は次によって定義されます。

```
CREATE OR REPLACE PACKAGE BODY emp_salhist
IS
    TYPE emprec_typ IS RECORD (
        empno      NUMBER(4),
        ename      VARCHAR(10),
        salhist    SALHIST_TYP
    );
    TYPE emp_arr_typ IS TABLE OF emprec_typ INDEX BY BINARY_INTEGER;
    emp_arr      emp_arr_typ;

    PROCEDURE fetch_emp (
        p_empno      IN NUMBER
    )
    IS
        CURSOR emp_cur IS SELECT e.empno, e.ename, h.startdate, h.job, h.sal
            FROM emp e, jobhist h
            WHERE e.empno = p_empno
            AND e.empno = h.empno;

        i          INTEGER := 0;
    BEGIN
        DBMS_OUTPUT.PUT_LINE('EMPNO  ENAME      STARTDATE  JOB          ' ||
            'SAL          ');
        DBMS_OUTPUT.PUT_LINE('-----  -----  -----  -----  ' ||
            '-----');

        FOR r_emp IN emp_cur LOOP
            i := i + 1;
            emp_arr(i) := (r_emp.empno, r_emp.ename,
```

(次のページに続く)

(前のページからの続き)

```

        (r_emp.startdate, r_emp.job, r_emp.sal));
    END LOOP;

    FOR i IN 1 .. emp_arr.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE(emp_arr(i).empno || ' ' ||
            RPAD(emp_arr(i).ename,8) || ' ' ||
            TO_CHAR(emp_arr(i).salhist.startdate,'DD-MON-YY') || ' ' ||
            RPAD(emp_arr(i).salhist.job,10) || ' ' ||
            TO_CHAR(emp_arr(i).salhist.sal,'99,999.99'));
    END LOOP;
END;
END;
```

パッケージ本体の TYPE emprec_typ IS RECORD データ構造の宣言では、salhist フィールドは CREATE TYPE salhist_typ ステートメントで作成された SALHIST_TYP 複合タイプで定義されていることに注意してください。

連想配列定義 TYPE emp_arr_typ IS TABLE OF emprec_typ は、SALHIST_TYP 複合型で定義されたフィールド salhist を含むレコード型データ構造 emprec_typ を参照します。

emp テーブルと jobhist テーブルの結合から配列をロードし、配列の内容を表示するパッケージプロシージャの呼び出しを以下に示します。

```
EXEC emp_salhist.fetch_emp(7788);
```

EMPNO	ENAME	STARTDATE	JOB	SAL
7788	SCOTT	19-APR-87	CLERK	1,000.00
7788	SCOTT	13-APR-88	CLERK	1,040.00
7788	SCOTT	05-MAY-90	ANALYST	3,000.00

```
EDB-SPL Procedure successfully completed
```

See Also

CREATETYPEBODY、*DROPTYPE*

第 40 章

CREATE TYPE BODY

Name

CREATE TYPE BODY -- 新しいオブジェクトタイプボディを定義します

Synopsis

```
CREATE [ OR REPLACE ] TYPE BODY <name>
  { IS | AS }
  <method_spec> [...]
END
```

ここで、method_spec は次のとおりです。

```
subprogram_spec
```

そして subprogram_spec は:

```
{ MEMBER | STATIC }
{ PROCEDURE <proc_name>
  [ ( [ SELF [ IN | IN OUT ] <name> ]
    [, <argname> [ IN | IN OUT | OUT ] <argtype>
      [ DEFAULT <value> ]
    ] ...)
  ]
}
{ IS | AS }
  <program_body>
END;
|
FUNCTION <func_name>
  [ ( [ SELF [ IN | IN OUT ] <name> ]
    [, <argname> [ IN | IN OUT | OUT ] <argtype>
      [ DEFAULT <value> ]
    ] ...)
  ]
```

(次のページに続く)

(前のページからの続き)

```
    ]
    RETURN <rettype>
{ IS | AS }
    <program_body>
END;
}
```

Description

CREATE TYPE BODY は、新しいオブジェクトタイプボディを定義します。CREATE OR REPLACE TYPE BODY は、新しいオブジェクトタイプのボディを作成するか、既存のボディを置き換えます。

スキーマ名が含まれている場合、オブジェクトタイプの本体は指定されたスキーマに作成されます。それ以外の場合は、現在のスキーマで作成されます。新しいオブジェクトタイプボディの名前は、同じスキーマ内の既存のオブジェクトタイプ仕様と一致する必要があります。新しいオブジェクトタイプボディ名は、既存のオブジェクトタイプボディの定義を更新する場合を除き、同じスキーマ内の既存のオブジェクトタイプボディと一致してはなりません。その場合は CREATE OR REPLACE TYPE BODY を使用します。

Parameters

name

ボディを作成するオブジェクトタイプの名前（オプションでスキーマ修飾）。

MEMBER

STATIC

サブプログラムがオブジェクトインスタンスで動作する場合は、MEMBER を指定します。サブプログラムが特定のオブジェクトインスタンスとは独立して動作する場合は、STATIC を指定します。

proc_name

作成するプロシージャの名前。

SELF [IN | IN OUT] name

メンバーメソッドの場合、データ型が定義されているオブジェクト型のデータ型である SELF という名前の暗黙の組み込みパラメーターがあります。SELF は、現在メソッドを呼び出しているオブジェクトインスタンスを指します。SELF は、パラメータリストで IN または IN OUT パラメータとして明示的に宣言できます。明示的に宣言する場合、SELF をパラメータリストの最初のパラメーターにする必要があります。SELF が明示的に宣言されていない場合、そのパラメータモードのデフォルトは、メンバープロシージャでは IN OUT、メンバー関数では IN です。

argname

引数の名前。引数は、メソッド本体内でこの名前によって参照されます。

argtype

メソッドの引数のデータ型。引数の型は、基本データ型でも、ネストした表やオブジェクト型などのユーザー定義型でもかまいません。ベースタイプには長さを指定しないでください。たとえば、`VARCHAR2(10)`ではなく `VARCHAR2` を指定します。

DEFAULT value

メソッド呼び出しで入力引数が指定されていない場合、入力引数のデフォルト値を指定します。モード `IN OUT` または `OUT` の引数には `DEFAULT` を指定できません。

program_body

関数またはプロシージャの本体を構成するプラグマ、宣言、および `SPL` ステートメント。プラグマを `PRAGMA AUTONOMOUS_TRANSACTION` にして、関数またはプロシージャを自律型トランザクションとして設定できます。

func_name

作成する関数の名前。

rettype

戻りデータ型。 `argtype` にリストされている型のいずれかです。 `argtype` については、 `rettype` の長さを指定しないでください。

Examples

`CREATE TYPE` コマンドの例で指定されたオブジェクトタイプ `emp_obj_typ` のオブジェクトタイプボディを作成します。

```
CREATE OR REPLACE TYPE BODY emp_obj_typ AS
  MEMBER PROCEDURE display_emp (SELF IN OUT emp_obj_typ)
  IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Employee No   : ' || empno);
    DBMS_OUTPUT.PUT_LINE('Name          : ' || ename);
    DBMS_OUTPUT.PUT_LINE('Street         : ' || addr.street);
    DBMS_OUTPUT.PUT_LINE('City/State/Zip: ' || addr.city || ', ' ||
      addr.state || ' ' || LPAD(addr.zip,5,'0'));
  END;
END;
```

`CREATE TYPE` コマンドの例で指定されたオブジェクトタイプ `dept_obj_typ` のオブジェクトタイプボディを作成します。

```
CREATE OR REPLACE TYPE BODY dept_obj_typ AS
  STATIC FUNCTION get_dname (p_deptno IN NUMBER) RETURN VARCHAR2
```

(次のページに続く)

(前のページからの続き)

```
IS
    v_dname      VARCHAR2(14);
BEGIN
    CASE p_deptno
        WHEN 10 THEN v_dname := 'ACCOUNTING';
        WHEN 20 THEN v_dname := 'RESEARCH';
        WHEN 30 THEN v_dname := 'SALES';
        WHEN 40 THEN v_dname := 'OPERATIONS';
        ELSE v_dname := 'UNKNOWN';
    END CASE;
    RETURN v_dname;
END;
MEMBER PROCEDURE display_dept
IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Dept No      : ' || SELF.deptno);
    DBMS_OUTPUT.PUT_LINE('Dept Name   : ' ||
        dept_obj_typ.get_dname(SELF.deptno));
END;
END;
```

See Also*CREATETYPE*、*DROPTYPE*

第 41 章

CREATE USER

Name

CREATE USER -- 新しいデータベースユーザーアカウントを定義する

Synopsis

```
CREATE USER <name> [IDENTIFIED BY <password>]
```

Description

CREATE USER は、AdvancedServer データベースクラスターに新しいユーザーを追加します。このコマンドを使用するには、データベースのスーパーユーザーである必要があります。

CREATE USER コマンドを指定すると、新しいユーザーと同じ名前でスキーマが作成され、新しいユーザーが所有します。このユーザーによって作成された非修飾名を持つオブジェクトは、このスキーマに作成されます。

Parameters

name

ユーザーの名前。

password

ユーザーのパスワード。パスワードは、ALTER USER を使用して後で変更できます。

Notes

ユーザー名とパスワードに許可される最大長は 63 文字です。

Examples

john という名前のユーザーを作成します。

```
CREATE USER john IDENTIFIED BY abc;
```

See Also

[ユーザーの削除](#)

第 42 章

CREATE USER|ROLE... PROFILE MANAGEMENT CLAUSES

Name

CREATE USER|ROLE

Synopsis

```
CREATE USER|ROLE <name> [[WITH] option [...]]
```

option は、次の互換性のある句です。

```
PROFILE <profile_name>
| ACCOUNT {LOCK|UNLOCK}
| PASSWORD EXPIRE [AT '<timestamp>']
```

または option は、次の互換性のない句にすることができます。

```
| LOCK TIME '<timestamp>'
```

AdvancedServer でサポートされている CREATE USER または CREATE ROLE コマンドの管理句については、次の場所にある 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/sql-commands.html>

Description

CREATE ROLE|USER... PROFILE は、AdvancedServer データベースクラスターに関連付けられたプロファイルを持つ新しいロールを追加します。

CREATE USER コマンドで作成されたロールは、(デフォルトで) ログインロールです。CREATE ROLE コマンドで作成されたロールは (デフォルトでは) ログインロールではありません。CREATE ROLE コマンドでログインアカウントを作成するには、LOGIN キーワードを含める必要があります。

データベース管理者のみが、プロファイル管理を実施する `CREATE USER|ROLE` 句を使用できます。これらの句は、次の動作を強制します。

`PROFILE` 句と `profile_name` を含めて、定義済みプロファイルをロールに関連付けるか、ユーザーに関連付けられている定義済みプロファイルを変更します。

`ACCOUNT` 句と `LOCK` または `UNLOCK` キーワードを含めて、ユーザーアカウントをロックまたはロック解除状態にすることを指定します。

`LOCK TIME 'timestamp'` 句と日付/時刻値を含めて、指定された時間にロールをロックし、このロールに割り当てられたプロファイルの `PASSWORD_LOCK_TIME` パラメーターで示された時間にロールをロック解除します。`ACCOUNT LOCK` 句で `LOCK TIME` を使用する場合、`ACCOUNT UNLOCK` 句を使用してデータベースのスーパーユーザーのみがロールをロック解除できます。

オプションの `AT 'timestamp'` キーワードとともに `PASSWORD EXPIRE` 句を含めて、ロールに関連付けられたパスワードの有効期限が切れる日時を指定します。`AT 'timestamp'` キーワードを省略すると、パスワードはすぐに期限切れになります。

各ログインロールに設定できるプロファイルは1つだけです。現在ログインロールに関連付けられているプロファイルを検出するには、`DBA_USERS` ビューの `profile` 列を照会します。

Parameters

`name`

ロールの名前。

`profile_name`

ロールに関連付けられているプロファイルの名前。

`timestamp`

条項が施行される日時。 `timestamp` の値を指定する場合は、値を一重引用符で囲みます。

Examples

次の例では、`CREATE USER` を使用して、`acctg_profile` プロファイルに関連付けられた `john` という名前のログインロールを作成します。

```
CREATE USER john PROFILE acctg_profile IDENTIFIED BY "1safepwd" ;
```

`john` は、パスワード `1safepwd` を使用してサーバーにログインできます。

次の例では、`CREATE ROLE` を使用して、`acctg_profile` プロファイルに関連付けられた `john` という名前のログインロールを作成します。

```
CREATE ROLE john PROFILE acctg_profile LOGIN PASSWORD "1safepwd" ;
```

john は、パスワード 1safepwd を使用してサーバーにログインできます。

See Also

ALTERUSER|ROLE...PROFILEMANAGEMENTCLAUSES

第 43 章

CREATE VIEW

Name

CREATE VIEW -- は新しいビューを定義します

Synopsis

```
CREATE [ OR REPLACE ] VIEW <name> [ ( <column_name> [, ...] ) ]  
AS <query>
```

Description

CREATE VIEW はクエリのビューを定義します。ビューは物理的に実体化されていません。代わりに、クエリがクエリで参照されるたびにクエリが実行されます。

CREATE OR REPLACE VIEW は似ていますが、同じ名前のビューが既に存在する場合は置き換えられます。

スキーマ名が指定されている場合（たとえば、CREATE VIEW myschema.myview ...）、指定されたスキーマにビューが作成されます。それ以外の場合は、現在のスキーマで作成されます。ビュー名は、同じスキーマ内の他のビュー、テーブル、シーケンス、またはインデックスの名前とは異なる必要があります。

Parameters

name

作成するビューの名前（オプションでスキーマ修飾）。

column_name

ビューの列に使用されるオプションの名前リスト。指定しない場合、列名はクエリから推定されます。

query

ビューの列と行を提供するクエリ（つまり、SELECT ステートメント）。

有効なクエリの詳細については、SELECT を参照してください。

Notes

現在、ビューは読み取り専用です-システムは、ビューでの挿入、更新、または削除を許可しません。ビューの挿入などを他のテーブルの適切なアクションに書き換えるルールを作成することにより、更新可能なビューの効果を得ることができます。

ビューで参照されるテーブルへのアクセスは、ビューの所有者の権限によって決まります。ただし、ビューで呼び出される関数は、ビューを使用するクエリから直接呼び出された場合と同じように扱われます。したがって、ビューのユーザーには、ビューで使用されるすべての関数を呼び出すためのアクセス許可が必要です。

Examples

部門 30 のすべての従業員で構成されるビューを作成します。

```
CREATE VIEW dept_30 AS SELECT * FROM emp WHERE deptno = 30;
```

See Also

[ドロップビュー](#)

第 44 章

DELETE

Name

DELETE -- はテーブルの行を削除します

Synopsis

```
DELETE [ <optimizer_hint> ] FROM <table>[@<dblink> ]
  [ WHERE <condition> ]
  [ RETURNING <return_expression> [, ...]
    { INTO { <record> | <variable> [, ...] }
      | BULK COLLECT INTO <collection> [, ...] } ]
```

Description

DELETE は、WHERE 句を満たす行を指定されたテーブルから削除します。WHERE 句がない場合、テーブル内のすべての行が削除されます。結果は有効ですが、空のテーブルです。

注釈: TRUNCATE コマンドは、テーブルからすべての行を削除するためのより高速なメカニズムを提供します。

RETURNING INTO { record | variable [, ...] } 節は、DELETE コマンドが SPL プログラム内で使用される場合にのみ指定できます。さらに、DELETE コマンドの結果セットに複数の行を含めることはできません。そうしないと、例外がスローされます。結果セットが空の場合、ターゲットレコードまたは変数の内容は null に設定されます。

RETURNING BULK COLLECT INTO collection [, ...] 節は、DELETE コマンドが SPL プログラム内で使用される場合にのみ指定できます。BULK COLLECT INTO 節のターゲットとして複数の「10」が指定されている場合、各「10」は単一のスカラーフィールドで構成されている必要があります。つまり、「10」はレコードであってはなりません。DELETE コマンドの結果セットには、0 行、1 行、または複数行が含まれる場合があります。結果セットの各行で評価された return_expression は、最初の要素から始まる「10」の要素になります。「10」の既存の行はすべて削除されます。結果セットが空の場合、「10」は空になります。

削除するテーブルの DELETE 特権と、条件で値が読み取られるテーブルの SELECT 特権が必要です。

Parameters

`optimizer_hint`

実行計画を選択するためのオプティマイザーへのコメントが埋め込まれたヒント。

`table`

既存のテーブルの名前（オプションでスキーマ修飾）。

`dblink`

リモートデータベースを識別するデータベースリンク名。データベースリンクの詳細については、CREATE DATABASE LINK コマンドを参照してください。

`condition`

削除する行を決定する BOOLEAN 型の値を返す値式。

`return_expression`

`table` の 1 つ以上の列を含む可能性のある式。 `return_expression` に `table` の列名が指定されている場合、 `return_expression` が評価されるときに列に代入される値は、削除された行の値です。

`record`

評価された `return_expression` が割り当てられるフィールドを持つレコード。最初の `return_expression` は `record` の最初のフィールドに割り当てられ、2 番目の `return_expression` は `record` の 2 番目のフィールドに割り当てられます。 `record` のフィールドの数は式の数と正確に一致する必要がありますフィールドは、割り当てられた式と型互換性がなければなりません。

`variable`

評価された `return_expression` が割り当てられる変数。 `return_expression` と `variable` が複数指定されている場合、 `first return_expression` は最初の `variable` に割り当てられ、2 番目の `return_expression` は 2 番目の `variable` に割り当てられます。 INTO キーワードは RETURNING キーワードに続く式の数と正確に一致する必要があります、変数は割り当てられた式と型互換性がなければなりません。

`collection`

評価された `return_expression` から要素が作成されるコレクション。単一フィールドのコレクションまたはレコードタイプのコレクションである単一コレクション、または複数のコレクションがあり、各コレクションが単一フィールドで構成されている必要があります。戻り式の数は数が一致し、指定されたすべてのコレクション内のフィールドの数を並べる必要があります。対応する各 `return_expression` および `collection` フィールドは、タイプ互換である必要があります。

Examples

jobhist テーブルから従業員 7900 のすべての行を削除します。

```
DELETE FROM jobhist WHERE empno = 7900;
```

テーブル jobhist をクリアします。

```
DELETE FROM jobhist;
```

See Also

TRUNCATE

第 45 章

DROP DATABASE LINK

Name

DROP DATABASE LINK -- はデータベースリンクを削除します

Synopsis

```
DROP [ PUBLIC ] DATABASE LINK name
```

Description

DROP DATABASE LINK は既存のデータベースリンクを削除します。このコマンドを実行するには、スーパーユーザーまたはデータベースリンクの所有者である必要があります。

Parameters

name

削除するデータベースリンクの名前。

PUBLIC

name がパブリックデータベースリンクであることを示します。

Examples

oralink という名前のパブリックデータベースリンクを削除します。

```
DROP PUBLIC DATABASE LINK oralink;
```

edblink という名前のプライベートデータベースリンクを削除します。

```
DROP DATABASE LINK edblink;
```

See Also

パブリックデータベースリンクの作成

第 46 章

DROP DIRECTORY

Name

DROP DIRECTORY -- はファイルシステムのディレクトリパスのディレクトリエイリアスを削除します

Synopsis

```
DROP DIRECTORY <name>
```

Description

DROP DIRECTORY は、CREATE DIRECTORY コマンドで作成されたファイルシステムディレクトリパスの既存のエイリアスを削除します。このコマンドを実行するには、スーパーユーザーである必要があります。

ディレクトリエイリアスが削除されても、対応する物理ファイルシステムディレクトリは影響を受けません。適切なオペレーティングシステムコマンドを使用して、ファイルシステムディレクトリを削除する必要があります。

Parameters

name

削除するディレクトリエイリアスの名前。

Examples

empdir という名前のディレクトリエイリアスを削除します。

```
DROP DIRECTORY empdir;
```

See Also

CREATEDIRECTORY、*ALTERDIRECTORY*

第 47 章

DROP FUNCTION

Name

DROP FUNCTION -- は機能を削除します

Synopsis

```
DROP FUNCTION [ IF EXISTS ] <name>
  [ ([ [ <argmode> ] [ <argname> ] <argtype> ] [, ...]) ]
  [ CASCADE | RESTRICT ]
```

Description

DROP FUNCTION は、既存の関数の定義を削除します。このコマンドを実行するには、スーパーユーザーまたは関数の所有者である必要があります。これがオーバーロードされた関数である場合、関数へのすべての入力（IN、IN OUT）引数データ型を指定する必要があります。（この要件は Oracle データベースと互換性がありません。Oracle では、関数名のみが指定されます。AdvancedServer は関数名のオーバーロードを許可するため、入力引数のデータ型によって指定される関数シグネチャオーバーロードされた関数。）

IF EXISTS、CASCADE、または RESTRICT の使用は Oracle データベースと互換性がなく、AdvancedServer のみが使用します。

Parameters

IF EXISTS

関数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

既存の関数の名前（オプションでスキーマ修飾）。

argmode

引数のモード:IN、IN OUT、または OUT。省略した場合、デフォルトは IN です。DROP FUNCTION は OUT 引数に実際には注意を払っていないことに注意してください。関数のアイデンティティを決定する

ために必要なのは入力引数のみです。したがって、IN および IN OUT 引数のみをリストするだけで十分です。(argmode の仕様は Oracle データベースと互換性がなく、AdvancedServer にのみ適用されます。)

argname

引数の名前。DROP FUNCTION は実際には引数名に注意を払っていないことに注意してください。これは、関数の ID を判断するために必要なのは引数のデータ型だけだからです。(argname の仕様は Oracle データベースと互換性がなく、AdvancedServer にのみ適用されます。)

argtype

関数の引数のデータ型。(argtype の仕様は Oracle データベースと互換性がなく、AdvancedServer にのみ適用されます。)

CASCADE

関数に依存するオブジェクト（演算子やトリガーなど）、およびそれらのオブジェクトに依存するすべてのオブジェクトを自動的に削除します。

RESTRICT

依存するオブジェクトがある場合、関数の削除を拒否します。これがデフォルトです。

Examples

次のコマンドは、emp_comp 機能を削除します。

```
DROP FUNCTION emp_comp (NUMBER, NUMBER);
```

See Also

[*CREATEFUNCTION*](#)

第 48 章

DROP INDEX

Name

`DROP INDEX --` はインデックスを削除します

Synopsis

```
DROP INDEX <name>
```

Description

`DROP INDEX` は、データベースシステムから既存のインデックスを削除します。このコマンドを実行するには、スーパーユーザーまたはインデックスの所有者である必要があります。インデックスに依存するオブジェクトがある場合、エラーが発生し、インデックスは削除されません。

Parameters

`name`

削除するインデックスの名前（オプションでスキーマ修飾）。

Examples

このコマンドは、インデックス `name_idx` を削除します。

```
DROP INDEX name_idx;
```

See Also

[*CREATEINDEX*](#)、[*ALTERINDEX*](#)

第 49 章

DROP PACKAGE

Name

DROP PACKAGE -- はパッケージを削除します

Synopsis

```
DROP PACKAGE [ BODY ] <name>
```

Description

DROP PACKAGE は既存のパッケージを削除します。このコマンドを実行するには、スーパーユーザーまたはパッケージの所有者である必要があります。BODY を指定すると、パッケージ本体のみが削除され、パッケージの仕様は削除されません。BODY を省略すると、パッケージの仕様と本文の両方が削除されます。

Parameters

name

削除するパッケージの名前（オプションでスキーマ修飾）。

Examples

このコマンドは emp_admin パッケージを削除します:

```
DROP PACKAGE emp_admin;
```

See Also

CREATEPACKAGE、*CREATEPACKAGEBODY*

第 50 章

DROP PROCEDURE

Name

DROP PROCEDURE -- は手順を削除します

Synopsis

```
DROP PROCEDURE [ IF EXISTS ] <name>
  [ ([ [ <argmode> ] [ <argname> ] <argtype> ] [, ...]) ]
  [ CASCADE | RESTRICT ]
```

Description

DROP PROCEDURE は、既存のプロシージャの定義を削除します。このコマンドを実行するには、スーパーユーザーまたはプロシージャの所有者である必要があります。これがオーバーロードされたプロシージャである場合、プロシージャへのすべての入力（IN、IN OUT）引数データ型を指定する必要があります。（この要件は、Oracle データベースと互換性がありません。Oracle では、プロシージャ名のみが指定されます。AdvancedServer では、プロシージャ名のオーバーロードが許可されます。オーバーロードされたプロシージャ。）

IF EXISTS、CASCADE、または RESTRICT の使用は Oracle データベースと互換性がなく、AdvancedServer のみが使用します。

Parameters

IF EXISTS

プロシージャ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

既存のプロシージャの名前（オプションでスキーマ修飾）。

argmode

引数のモード:IN、IN OUT、または OUT。省略した場合、デフォルトは IN です。DROP PROCEDURE は実際には OUT 引数に注意を払っていないことに注意してください。これは、プロシージャの ID を決定

するために入力引数のみが必要なためです。したがって、IN および IN OUT 引数のみをリストするだけで十分です。(`argmode` の仕様は Oracle データベースと互換性がなく、AdvancedServer にのみ適用されます。)

`argname`

引数の名前。DROP PROCEDURE は実際には引数名に注意を払っていないことに注意してください。これは、プロシージャの ID を決定するために必要なのは引数のデータ型だけだからです。(`argname` の仕様は Oracle データベースと互換性がなく、AdvancedServer にのみ適用されます。)

`argtype`

プロシージャの引数のデータ型。(`argtype` の仕様は Oracle データベースと互換性がなく、AdvancedServer にのみ適用されます。)

CASCADE

プロシージャに依存するオブジェクト、およびそれらのオブジェクトに依存するすべてのオブジェクトを自動的に削除します。

RESTRICT

依存するオブジェクトがある場合、プロシージャの削除を拒否します。これがデフォルトです。

Examples

次のコマンドは、`select_emp` プロシージャを削除します。

```
DROP PROCEDURE select_emp;
```

See Also

CREATEPROCEDURE、*ALTERPROCEDURE*

第 51 章

DROP PROFILE

Name

DROP PROFILE - はユーザー定義のプロファイルを削除します

Synopsis

```
DROP PROFILE [IF EXISTS] <profile_name> [CASCADE | RESTRICT];
```

Description

指定したプロファイルが存在しない場合にエラーをスローしないようにサーバーに指示するには、IF EXISTS 句を含めます。プロファイルが存在しない場合、サーバーは通知を発行します。

オプションの CASCADE 句を含めて、現在プロファイルに関連付けられているユーザーを default プロファイルに再割り当てしてから、プロファイルをドロップします。オプションの RESTRICT 句を含めて、ロールに関連付けられているプロファイルを削除しないようにサーバーに指示します。これがデフォルトの動作です。

Parameters

profile_name

削除されるプロファイルの名前。

Examples

次の例では、acctg_profile という名前のプロファイルを削除します。

```
DROP PROFILE acctg_profile CASCADE;
```

このコマンドは、最初に acctg_profile プロファイルに関連付けられているロールを default プロファイルに再度関連付けてから、acctg_profile プロファイルをドロップします。

次の例では、acctg_profile という名前のプロファイルを削除します。

```
DROP PROFILE acctg_profile RESTRICT;
```

コマンドの RESTRICT 句は、プロファイルに関連付けられたロールがある場合に acctg_profile をドロップしないようにサーバーに指示します。

See Also

[プロファイルの作成、プロファイルの変更](#)

第 52 章

DROP QUEUE

AdvancedServer には、DROP QUEUE SQL コマンドに追加の構文（Oracle によって提供されていない）が含まれています。この構文は、DBMS_AQADM と関連付けて使用できます。

Name

DROP QUEUE -- は既存のキューを削除します。

Synopsis

DROP QUEUE を使用して既存のキューを削除します。

```
DROP QUEUE [IF EXISTS] <name>
```

Description

DROP QUEUE は、スーパーユーザーまたは aq_administrator_role 特権を持つユーザーが既存のキューを削除できるようにします。

Parameters

name

削除されるキューの名前（スキーマで修飾されている可能性があります）。

IF EXISTS

キューが存在しない場合にエラーを返さないようにサーバーに指示するには、IF EXISTS 句を含めます。サーバーは通知を発行します。

Examples

次の例では、work_order という名前のキューを削除します。

```
DROP QUEUE work_order;
```

See Also

CREATEQUEUE、*ALTERQUEUE*

第 53 章

DROP QUEUE TABLE

AdvancedServer には、DROP QUEUE TABLE SQL コマンドに追加の構文（Oracle によって提供されていない）が含まれています。この構文は、DBMS_AQADM と関連付けて使用できます。

Name

DROP QUEUE TABLE-- はキューテーブルを削除します。

Synopsis

DROP QUEUE TABLE を使用してキューテーブルを削除します。

```
DROP QUEUE TABLE [ IF EXISTS ] <name> [, ...]
[CASCADE | RESTRICT]
```

Description

DROP QUEUE TABLE を指定すると、スーパーユーザーまたは aq_administrator_role 権限を持つユーザーがキューテーブルを削除できます。

Parameters

name

削除されるキューテーブルの名前（スキーマで修飾されている可能性があります）。

IF EXISTS

キューテーブルが存在しない場合にエラーを返さないようにサーバーに指示するには、IF EXISTS 句を含めます。サーバーは通知を発行します。

CASCADE

キューテーブルに依存するオブジェクトを自動的に削除するには、CASCADE キーワードを含めます。

RESTRICT

RESTRICT キーワードを含めて、キューテーブルに依存するオブジェクトがある場合、キューテーブルの削除を拒否するようサーバーに指示します。これがデフォルトです。

Examples

次の例では、work_order_table という名前のキューテーブルとそれに依存するオブジェクトを削除します。

```
DROP QUEUE TABLE work_order_table CASCADE;
```

See Also

CREATEQUEUETABLE、*ALTERQUEUETABLE*

第 54 章

DROP SYNONYM

Name

DROP SYNONYM -- 同義語を削除する

Synopsis

```
DROP [PUBLIC] SYNONYM [<schema>.]<syn_name>
```

Description

DROP SYNONYM は既存の同義語を削除します。このコマンドを実行するには、スーパーユーザーまたはシノニムの所有者であり、シノニムが存在するスキーマに対する USAGE 権限が必要です。

Parameters

`syn_name`

`syn_name` は同義語の名前です。シノニム名はスキーマ内で一意である必要があります。

`schema`

`schema` は、シノニムが存在するスキーマの名前を指定します。

スキーマで修飾できる他のオブジェクトと同様に、検索パスに同じ名前の 2 つの同義語がある場合があります。削除するシノニムの名前を明確にするには、スキーマ名を含めます。シノニムが DROP SYNONYM コマンドでスキーマ修飾されていない限り、AdvancedServer は検索パスで見つかったシノニムの最初のインスタンスを削除します。

オプションで PUBLIC 句を含めて、public スキーマにある同義語を削除できます。Oracle データベースと互換性のある DROP PUBLIC SYNONYM コマンドは、public スキーマにある同義語を削除します。

```
DROP PUBLIC SYNONYM syn_name;
```

次の例では、シノニム `personnel` を削除します。

```
DROP SYNONYM personnel;
```

See Also

[シノニムの作成](#)

第 55 章

DROP ROLE

Name

DROP ROLE -- データベースロールを削除する

Synopsis

```
DROP ROLE <name> [ CASCADE ]
```

Description

DROP ROLE は指定されたロールを削除します。スーパーユーザーの役割を削除するには、自分でスーパーユーザーになる必要があります。スーパーユーザー以外のロールを削除するには、CREATEROLE 権限が必要です。

クラスタの任意のデータベースでまだ参照されているロールは削除できません。その場合、エラーが発生します。ロールを削除する前に、所有するすべてのオブジェクトを削除（または所有権を再割り当て）し、ロールに付与された特権をすべて取り消す必要があります。

ロールに関係するロールメンバーシップを削除する必要はありません。DROP ROLE は、他のロールのターゲットロールと、ターゲットロールの他のロールのメンバーシップを自動的に取り消します。他の役割は削除されず、影響もありません。

または、ロールが所有するオブジェクトのみが、ロールが所有するスキーマ内に属し、ロールと同じ名前を持つ場合、CASCADE オプションを指定できます。この場合、DROP ROLE name CASCADE コマンドの発行者はスーパーユーザーでなければならない、名前付きロール、スキーマ、およびスキーマ内のすべてのオブジェクトが削除されます。

Parameters

name

削除するロールの名前。

CASCADE

指定された場合、ロールまたはスキーマに他の依存関係が存在しない限り、ロール（およびスキーマに属するロールが所有するすべてのオブジェクト）が所有するスキーマも削除します。

Examples

ロールを削除するには:

```
DROP ROLE admins;
```

See Also

CREATEROLE、*SETROLE*、*GRANT*、*REVOKE*

第 56 章

DROP SEQUENCE

Name

DROP SEQUENCE -- シーケンスを削除

Synopsis

```
DROP SEQUENCE <name> [, ...]
```

Description

DROP SEQUENCE はシーケンス番号ジェネレーターを削除します。このコマンドを実行するには、スーパーユーザーまたはシーケンスの所有者である必要があります。

Parameters

name

シーケンスの名前（オプションでスキーマ修飾）。

Examples

シーケンスを削除するには、serial:

```
DROP SEQUENCE serial;
```

See Also

ALTERSEQUENCE、*CREATESEQUENCE*

第 57 章

DROP TABLE

Name

DROP TABLE -- テーブルを削除

Synopsis

DROP TABLE <name> [CASCADE RESTRICT CASCADE CONSTRAINTS]
--

Description

DROP TABLE はデータベースからテーブルを削除します。テーブルを破壊できるのは所有者のみです。テーブルを破壊せずに行のテーブルを空にするには、DELETE を使用します。DROP TABLE は、ターゲットテーブルに存在するインデックス、ルール、トリガー、および制約を常に削除します。

Parameters

name

削除するテーブルの名前（オプションでスキーマ修飾）。

RESTRICT キーワードを含めて、依存するオブジェクトがある場合にサーバーがテーブルの削除を拒否するように指定します。これはデフォルトの動作です。テーブルに依存するオブジェクトがある場合、DROP TABLE コマンドはエラーを報告します。

テーブルに依存するオブジェクトを削除するには、CASCADE 句を含めます。

CASCADE CONSTRAINTS 句を含めて、AdvancedServer が指定されたテーブルの依存制約（他のオブジェクトタイプを除く）を削除するように指定します。

Examples

次のコマンドは、依存関係のない emp という名前のテーブルを削除します。

```
DROP TABLE emp;
```

DROP TABLE コマンドの結果は、テーブルに依存関係があるかどうかによって異なります。drop behavior を指定すると、結果を制御できます。たとえば、orders と items の 2 つのテーブルを作成すると、items テーブルは orders テーブルに依存します。

```
CREATE TABLE orders
  (order_id int PRIMARY KEY, order_date date, ...);
CREATE TABLE items
  (order_id REFERENCES orders, quantity int, ...);
```

AdvancedServer は、指定したドロップ動作に応じて、orders テーブルをドロップするときに次のいずれかのアクションを実行します。

- DROP TABLE orders RESTRICT を指定すると、AdvancedServer はエラーを報告します。
- DROP TABLE orders CASCADE を指定すると、AdvancedServer は orders テーブルと items テーブルをドロップします。
- DROP TABLE orders CASCADE CONSTRAINTS を指定すると、AdvancedServer は orders テーブルを削除し、items テーブルから外部キーの指定を削除しますが、items テーブルは削除しません。

See Also

[*CREATETABLE*](#)、[*ALTERTABLE*](#)

第 58 章

DROP TABLESPACE

Name

DROP TABLESPACE -- 表領域を削除する

Synopsis

```
DROP TABLESPACE <tablespacename>
```

Description

DROP TABLESPACE はシステムからテーブルスペースを削除します。

表領域は、所有者またはスーパーユーザーのみが削除できます。テーブルスペースは、削除する前にすべてのデータベースオブジェクトを空にしておく必要があります。現在のデータベースのオブジェクトが表領域を使用していない場合でも、他のデータベースのオブジェクトが表領域に残っている可能性があります。

Parameters

tablespacename

表スペースの名前。

Examples

システムから表領域 employee_space を削除するには:

```
DROP TABLESPACE employee_space;
```

See Also

[*ALTERTABLESPACE*](#)

第 59 章

DROP TRIGGER

Name

DROP TRIGGER -- トリガーを削除

Synopsis

```
DROP TRIGGER <name>
```

Description

DROP TRIGGER は、関連するテーブルからトリガーを削除します。このコマンドは、トリガーが定義されているテーブルのスーパーユーザーまたは所有者が実行する必要があります。

Parameters

name

削除するトリガーの名前。

Examples

emp_salary_trig という名前のトリガーを削除します。

```
DROP TRIGGER emp_salary_trig;
```

See Also

CREATETRIGGER、*ALTERTRIGGER*

第 60 章

DROP TYPE

Name

DROP TYPE -- は型定義を削除します

Synopsis

```
DROP TYPE [ BODY ] <name>
```

Description

DROP TYPE はタイプ定義を削除します。このコマンドを実行するには、スーパーユーザーまたはそのタイプの所有者でなければなりません。

オプションの BODY 修飾子は、オブジェクトタイプの定義にのみ適用され、コレクションタイプや複合タイプには適用されません。BODY を指定すると、オブジェクトタイプの本体のみが削除され、オブジェクトタイプの指定は削除されません。BODY を省略すると、オブジェクトタイプの指定と本文の両方が削除されます。

名前付きタイプに依存する他のデータベースオブジェクトがある場合、タイプは削除されません。

Parameters

name

削除する型定義の名前。

Examples

addr_obj_typ という名前のオブジェクトタイプをドロップします。

```
DROP TYPE addr_obj_typ;
```

budget_tbl_typ という名前のネストしたテーブルタイプを削除します。

```
DROP TYPE budget_tbl_typ;
```

See Also

CREATETYPE、*CREATETYPEBODY*

第 61 章

DROP USER

Name

DROP USER -- データベースユーザーアカウントを削除する

Synopsis

```
DROP USER <name> [ CASCADE ]
```

Description

DROP USER は指定されたユーザーを削除します。スーパーユーザーを削除するには、自分でスーパーユーザーになる必要があります。非スーパーユーザーを削除するには、CREATEROLE 権限が必要です。

クラスタのデータベースでまだ参照されているユーザーは削除できません。その場合、エラーが発生します。ユーザーを削除する前に、ユーザーが所有するすべてのオブジェクトを削除（または所有権を再割り当て）し、ユーザーに付与された権限を取り消す必要があります。

ただし、ユーザーに関係するロールメンバーシップを削除する必要はありません。DROP USER は、他のロールのターゲットユーザー、およびターゲットユーザーの他のロールのメンバーシップを自動的に取り消します。他の役割は削除されず、影響もありません。

または、ユーザーが所有するオブジェクトのみが、ユーザーが所有するスキーマ内に属し、ユーザーと同じ名前を持つ場合、CASCADE オプションを指定できます。この場合、DROP USER name CASCADE コマンドの発行者はスーパーユーザーである必要があります、指定されたユーザー、スキーマ、およびスキーマ内のすべてのオブジェクトが削除されます。

Parameters

name

削除するユーザーの名前。

CASCADE

指定した場合、ユーザーまたはスキーマに対する他の依存関係が存在しない限り、ユーザーが所有し、ユーザーと同じ名前のスキーマ（およびスキーマに属するユーザーが所有するすべてのオブジェクト）も削除します。

Examples

オブジェクトを所有しておらず、他のオブジェクトに対する権限も付与されていない john という名前のユーザーアカウントを削除するには:

```
DROP USER john;
```

オブジェクトに対する特権が付与されておらず、ユーザー john が所有する john という名前のスキーマ外のオブジェクトを所有していないユーザーアカウント john を削除するには:

```
DROP USER john CASCADE;
```

See Also

CREATEUSER、*ALTERUSER*

第 62 章

DROP VIEW

Name

DROP VIEW -- はビューを削除します

Synopsis

```
DROP VIEW <name>
```

Description

DROP VIEW は既存のビューを削除します。このコマンドを実行するには、データベースのスーパーユーザーまたはビューの所有者である必要があります。他のオブジェクトがこのビューに依存している場合（ビューのビューなど）、名前の付いたビューは削除されません。

Oracle と互換性のある DROP VIEW コマンドの形式は CASCADE 節をサポートしていません。ビューとその依存関係を削除するには、PostgreSQL 互換形式の DROP VIEW コマンドを使用します。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-dropview.html>

Parameters

name

削除するビューの名前（オプションでスキーマ修飾）。

Examples

このコマンドは、dept_30 という名前のビューを削除します。

```
DROP VIEW dept_30;
```

See Also

[CREATEVIEW](#)

第 63 章

EXEC

Name

EXEC

Synopsis

```
EXEC function_name ['(' [<argument_list>] ')']
```

Description

EXECUTE

Parameters

procedure_name

procedure_name は、（オプションでスキーマ修飾された）関数名です。

argument_list

argument_list は、関数に必要な引数のコンマ区切りリストを指定します。 argument_list の各メンバーは、関数が予期する正式な引数に対応することに注意してください。各仮引数は、IN パラメーター、OUT パラメーター、または INOUT パラメーターです。

Examples

EXEC ステートメントは、関数に必要な引数に応じて、いくつかの形式のいずれかを取ります。

```
EXEC update_balance;  
EXEC update_balance();  
EXEC update_balance(1,2,3);
```


第 64 章

GRANT

Name

GRANT -- はアクセス権を定義します

Synopsis

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | REFERENCES }
        [, ...] | ALL [ PRIVILEGES ] }
      ON tablename
      TO { username | groupname | PUBLIC } [, ...]
      [ WITH GRANT OPTION ]

GRANT { { INSERT | UPDATE | REFERENCES } (column [, ...]) }
      [, ...]
      ON tablename
      TO { username | groupname | PUBLIC } [, ...]
      [ WITH GRANT OPTION ]

GRANT { SELECT | ALL [ PRIVILEGES ] }
      ON sequencename
      TO { username | groupname | PUBLIC } [, ...]
      [ WITH GRANT OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
      ON FUNCTION progname
      ( [ [ argmode ] [ argname ] argtype ] [, ...] )
      TO { username | groupname | PUBLIC } [, ...]
      [ WITH GRANT OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
      ON PROCEDURE progname
      [ ( [ [ argmode ] [ argname ] argtype ] [, ...] ) ]
      TO { username | groupname | PUBLIC } [, ...]
      [ WITH GRANT OPTION ]
```

(次のページに続く)

(前のページからの続き)

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }  
  ON PACKAGE packagename  
  TO { username | groupname | PUBLIC } [, ...]  
  [ WITH GRANT OPTION ]  
  
GRANT role [, ...]  
  TO { username | groupname | PUBLIC } [, ...]  
  [ WITH ADMIN OPTION ]  
  
GRANT { CONNECT | RESOURCE | DBA } [, ...]  
  TO { username | groupname } [, ...]  
  [ WITH ADMIN OPTION ]  
  
GRANT CREATE [ PUBLIC ] DATABASE LINK  
  TO { username | groupname }  
  
GRANT DROP PUBLIC DATABASE LINK  
  TO { username | groupname }  
  
GRANT EXEMPT ACCESS POLICY  
  TO { username | groupname }
```

Description

GRANT コマンドには、データベースオブジェクト（テーブル、ビュー、シーケンス、またはプログラム）に対する権限を付与するもの、ロールのメンバーシップを付与するもの、システム権限を付与するものの3つの基本的な変形があります。これらのバリエーションは多くの点で似ていますが、別々に説明するのに十分なほど異なっています。

AdvancedServer では、ユーザーとグループの概念が `role` と呼ばれる単一のタイプのエンティティに統合されました。このコンテキストでは、`user` は LOGIN 属性を持つロールです。このロールを使用して、セッションを作成し、アプリケーションに接続できます。`group` は LOGIN 属性を持たないロールです。このロールは、セッションの作成やアプリケーションへの接続には使用できません。

ロールは1つ以上の他のロールのメンバーである場合があるため、グループに属するユーザーの従来の概念は引き続き有効です。ただし、ユーザーとグループの一般化により、ユーザーはユーザーに「所属」、グループはグループに「所属」、グループはユーザーに「所属」し、ロールの一般的なマルチレベル階層を形成できます。ユーザー名とグループ名は同じ名前空間を共有するため、GRANT コマンドで被付与者がユーザーであるかグループであるかを区別する必要はありません。

64.1 データベースオブジェクトの付与

GRANT コマンドのこのバリエーションは、データベースオブジェクトに対する特定の特権をロールに付与します。これらの特権は、すでに付与されているものがあれば、それらに追加されます。

キーワード PUBLIC は、後で作成されるロールを含むすべてのロールに特権が付与されることを示します。PUBLIC は、すべてのロールを常に含む暗黙的に定義されたグループと考えることができます。特定のロールには、直接付与された特権、現在メンバーであるロールに付与された特権、および PUBLIC に付与された特権の合計があります。

WITH GRANT OPTION が指定されている場合、特権の受信者はそれを他の人に順番に付与できます。許可オプションがないと、受信者はそれを行うことができません。付与オプションは PUBLIC に付与できません。

所有者はデフォルトですべての特権を持っているため、オブジェクトの所有者（通常はオブジェクトを作成したユーザー）に特権を付与する必要はありません。（ただし、所有者は、安全のために自分の特権の一部を取り消すこともできます。）オブジェクトをドロップする権利または定義を変更する権利は、付与可能な特権では説明されません。それは所有者に固有のものであり、付与または取り消すことはできません。所有者には、オブジェクトに対するすべての付与オプションも暗黙的に含まれています。

オブジェクトのタイプに応じて、初期のデフォルト権限には PUBLIC への一部の権限の付与が含まれる場合があります。デフォルトは、テーブルへのパブリックアクセスなし、関数、プロシージャ、およびパッケージに対する EXECUTE 特権です。オブジェクトの所有者は、もちろんこれらの特権を取り消すことができます。（最大限のセキュリティを確保するには、オブジェクトを作成する同じトランザクションで REVOKE を発行します。そうすると、別のユーザーがオブジェクトを使用できるウィンドウがなくなります。）

可能な特権は次のとおりです。

SELECT

指定したテーブル、ビュー、またはシーケンスの任意の列から SELECT を許可します。シーケンスの場合、この特権は currval 機能の使用も許可します。

INSERT

指定したテーブルに新しい行の INSERT を許可します。

UPDATE

指定されたテーブルの列の UPDATE を許可します。SELECT ... FOR UPDATE にはこの特権も必要です（SELECT 特権に加えて）。

DELETE

指定されたテーブルの行の DELETE を許可します。

REFERENCES

外部キー制約を作成するには、参照元テーブルと参照先テーブルの両方でこの権限が必要です。

EXECUTE

指定されたパッケージ、プロシージャ、または関数の使用を許可します。パッケージに適用すると、パッケージのすべてのパブリックプロシージャ、パブリック関数、パブリック変数、レコード、カーソル、その他のパブリックオブジェクトおよびオブジェクトタイプを使用できます。これは、関数、プロシージャ、およびパッケージに適用できる唯一の種類の特権です。

EXECUTE 特権を付与する **AdvancedServer** 構文は、**Oracle** データベースと完全に互換性がありません。**AdvancedServer** では、キーワード **FUNCTION**、**PROCEDURE**、または **PACKAGE** のいずれかでプログラム名を修飾する必要がありますが、これらのキーワードは **Oracle** では省略しなければなりません。関数の場合、**AdvancedServer** では、関数名の後にすべての入力（**IN**、**IN OUT**）引数データ型が必要です（関数引数がない場合は空の括弧を含む）。プロシージャの場合、プロシージャに 1 つ以上の入力引数がある場合、すべての入力引数データ型を指定する必要があります。**Oracle** では、関数とプロシージャの署名を省略する必要があります。これは、すべてのプログラムが **Oracle** で同じ名前空間を共有するのに対し、関数、プロシージャ、およびパッケージは **AdvancedServer** で独自の名前空間を持ち、プログラム名をある程度オーバーロードできるためです。

ALL PRIVILEGES

使用可能なすべての特権を一度に付与します。

他のコマンドに必要な特権は、それぞれのコマンドのリファレンスページにリストされています。

64.2 ロールの付与

GRANT コマンドのこのバリエーションは、ロールのメンバーシップを 1 つ以上の他のロールに付与します。ロールのメンバーシップは、ロールに付与された特権を各メンバーに伝えるため、重要です。

WITH ADMIN OPTION が指定されている場合、メンバーはそのロールのメンバーシップを他のユーザーに付与し、そのロールのメンバーシップも取り消すことができます。admin オプションがなければ、普通のユーザーはそれできません。

データベースのスーパーユーザーは、あらゆるロールのメンバーシップを誰にでも付与または取り消すことができます。CREATEROLE 特権を持つロールは、スーパーユーザー以外のロールのメンバーシップを付与または取り消すことができます。

次の意味を持つ 3 つの定義済みロールがあります。

CONNECT

CONNECT ロールを付与することは、被付与者に LOGIN 特権を付与することと同等です。付与者には CREATEROLE 権限が必要です。

RESOURCE

RESOURCE ロールを付与することは、被付与者と同じ名前のスキーマに CREATE および USAGE 特権を付与することと同等です。このスキーマは、許可が付与される前に存在している必要があります。付与者は、このスキーマの CREATE または USAGE 特権を被付与者に付与する特権を持っている必要があります。

DBA

DBA ロールを付与することは、被付与者をスーパーユーザーにすることと同等です。付与者はスーパーユーザーでなければなりません。

Notes

REVOKE コマンドは、アクセス権限を取り消すために使用されます。

オブジェクトの非所有者がオブジェクトに対する GRANT 特権を試みると、ユーザーがオブジェクトに対する特権を持っていない場合、コマンドは完全に失敗します。特権が利用可能である限り、コマンドは続行しますが、ユーザーが許可オプションを持っている特権のみを許可します。GRANT ALL PRIVILEGES フォームは、付与オプションが保持されていない場合に警告メッセージを発行し、他のフォームは、コマンドで明示的に指定された特権のいずれかの付与オプションが保持されていない場合に警告を発行します。（原則として、これらのステートメントはオブジェクト所有者にも適用されますが、所有者は常にすべての付与オプションを保持しているものとして扱われるため、ケースは発生しません。）

データベースのスーパーユーザーは、オブジェクト権限の設定に関係なく、すべてのオブジェクトにアクセスすることに注意してください。これは、Unix システムの root の権利に匹敵します。root と同様に、絶対に必要な場合を除き、スーパーユーザーとして操作することは賢明ではありません。

スーパーユーザーが GRANT または REVOKE コマンドを発行することを選択した場合、コマンドは、影響を受けるオブジェクトの所有者によって発行されたかのように実行されます。特に、このようなコマンドを介して付与された特権は、オブジェクトの所有者によって付与されたように見えます。(ロールメンバースhipの場合、メンバースhipは含まれるロール自体によって付与されたようです。)

GRANT および REVOKE は、影響を受けるオブジェクトの所有者ではないが、オブジェクトを所有するロールのメンバーであるロール、または特権 WITH GRANT OPTION を保持するロールのメンバーでも実行できますオブジェクト。この場合、特権は、オブジェクトを実際に所有するロールまたは特権 WITH GRANT OPTION を保持するロールによって付与されたものとして記録されます。

たとえば、テーブル `t1` がロール `g1` によって所有され、ロール `u1` がメンバーである場合、`u1` は `t1` に対する特権を `u2` に付与できますが、それらの特権は `g1` によって直接付与されます。役割 `g1` の他のメンバーは、後でそれらを取り消すことができます。

GRANT を実行するロールが、複数のロールメンバースhipパスを介して間接的に必要な特権を保持している場合、どのロールが許可を行ったとして記録されるかは不明です。そのような場合は、`SET ROLE` を使用して GRANT を実行する特定の役割になることがベストプラクティスです。

現在、AdvancedServer は、テーブルの個々の列に対する特権の付与または取り消しをサポートしていません。考えられる回避策の 1 つは、目的の列のみを持つビューを作成し、そのビューに特権を付与することです。

Examples

テーブル `emp` のすべてのユーザーに挿入特権を付与します。

```
GRANT INSERT ON emp TO PUBLIC;
```

ビュー `salesemp` のユーザー `mary` に使用可能なすべての特権を付与します。

```
GRANT ALL PRIVILEGES ON salesemp TO mary;
```

上記は実際にスーパーユーザーまたは `emp` の所有者によって実行された場合、すべての特権を付与しますが、他の誰かによって実行された場合、他の誰かが付与オプションを持っているパーミッションのみを付与します。

ユーザー `joe` にロール `admins` のメンバースhipを付与します。

```
GRANT admins TO joe;
```

ユーザー `joe` に `CONNECT` 権限を付与します。

```
GRANT CONNECT TO joe;
```

See Also

[REVOKE](#)、[SETROLE](#)

64.3 システム権限の付与

GRANT コマンドのこのバリエーションは、データベース内で特定の system 操作を実行する機能をロールに付与します。システム特権は、必ずしも 1 つのスキーマの範囲内でない特定のデータベースオブジェクトを作成または削除する機能に関連しています。データベースのスーパーユーザーのみがシステム権限を付与できます。

CREATE [PUBLIC] DATABASE LINK

CREATE [PUBLIC] DATABASE LINK 特権により、指定されたロールがデータベースリンクを作成できます。PUBLIC キーワードを含めて、ロールがパブリックデータベースリンクを作成できるようにします。PUBLIC キーワードを省略して、指定されたロールがプライベートデータベースリンクを作成できるようにします。

DROP PUBLIC DATABASE LINK

DROP PUBLIC DATABASE LINK 特権により、ロールはパブリックデータベースリンクを削除できます。プライベートデータベースリンクを削除するためにシステム権限は必要ありません。プライベートデータベースリンクは、リンク所有者またはデータベーススーパーユーザーによって削除される場合があります。

EXEMPT ACCESS POLICY

EXEMPT ACCESS POLICY 特権により、ロールは、ターゲットデータベースオブジェクトに関連付けられているポリシー関数を呼び出さずに SQL コマンドを実行できます。ロールは、データベース内のすべてのセキュリティポリシーから免除されます。

EXEMPT ACCESS POLICY 特権は、EXEMPT ACCESS POLICY 特権を持つロールへのメンバーシップによって継承できません。たとえば、次の GRANT コマンドのシーケンスでは、joe に EXEMPT ACCESS POLICY 特権が付与されている enterprisedb ロールのメンバーシップが joe に付与されていても、ユーザー joe は EXEMPT ACCESS POLICY 特権を取得しません：

```
GRANT EXEMPT ACCESS POLICY TO enterprisedb;
GRANT enterprisedb TO joe;
```

ロールに EXEMPT ACCESS POLICY 権限がある場合、システムカタログ表 pg_authid の rolpolicyexempt 列は true に設定されます。

Examples

ユーザー joe に CREATE PUBLIC DATABASE LINK 権限を付与します。

```
GRANT CREATE PUBLIC DATABASE LINK TO joe;
```

ユーザー joe に DROP PUBLIC DATABASE LINK 権限を付与します。

```
GRANT DROP PUBLIC DATABASE LINK TO joe;
```

ユーザー joe に EXEMPT ACCESS POLICY 権限を付与します。

```
GRANT EXEMPT ACCESS POLICY TO joe;
```

Using the ALTER ROLE Command to Assign System Privileges

AdvancedServer の ALTER ROLE コマンドは、次の割り当てに使用できる構文もサポートしています。

- パブリックまたはプライベートのデータベースリンクを作成するために必要な特権。
- パブリックデータベースリンクを削除するために必要な特権。
- EXEMPT ACCESS POLICY 特権。

ALTER ROLE 構文は、Oracle データベースと互換性のある各コマンドと機能的に同等です。

See Also

REVOKE、*ALTERROLE*

第 65 章

INSERT

Name

INSERT -- テーブルに新しい行を作成

Synopsis

```
INSERT INTO <table>[@<dblink> ] [ ( <column> [, ...] ) ]
    { VALUES ( { <expression> | DEFAULT } [, ...] )
      [ RETURNING <return_expression> [, ...]
        { INTO { <record> | <variable> [, ...] }
          | BULK COLLECT INTO <collection> [, ...] } } ]
    | <query> }
```

Description

INSERT を使用すると、テーブルに新しい行を挿入できます。一度に 1 つの行を挿入することも、クエリの結果として複数の行を挿入することもできます。

ターゲットリストの列は、任意の順序でリストできます。ターゲットリストにない各列は、宣言されたデフォルト値または **null** のデフォルト値を使用して挿入されます。

各列の式が正しいデータ型でない場合、自動型変換が試行されます。

RETURNING INTO { record | variable [, ...] } 句は、INSERT コマンドが **SPL** プログラム内で使用され、VALUES 句が使用されている場合にのみ指定できます。

RETURNING BULK COLLECT INTO collection [, ...] 節は、INSERT コマンドが **SPL** プログラム内で使用される場合にのみ指定できます。BULK COLLECT INTO 句のターゲットとして複数の collection が指定されている場合、各 collection は単一のスカラーフィールドで構成されている必要があります。つまり、collection はレコードであってはなりません。挿入された行ごとに return_expression が評価され、最初の要素から collection の要素になります。collection の既存の行はすべて削除されます。結果セットが空の場合、collection は空になります。

テーブルに挿入するには、テーブルに対する INSERT 権限が必要です。query 句を使用してクエリから行を挿入する場合、クエリで使用されるテーブルに対する SELECT 権限も必要です。

Parameters

table

既存のテーブルの名前（オプションでスキーマ修飾）。

dblink

リモートデータベースを識別するデータベースリンク名。データベースリンクの詳細については、CREATE DATABASE LINK コマンドを参照してください。

column

table の列の名前。

expression

column に割り当てる式または値。

DEFAULT

この列にはデフォルト値が入力されます。

query

挿入する行を提供するクエリ（SELECT ステートメント）。構文の説明については、SELECT コマンドを参照してください。

return_expression

table の 1 つ以上の列を含む可能性のある式。return_expression に table の列名が指定されている場合、return_expression の評価時に列に代入される値は次のように決定されます。

return_expression で指定された列に INSERT コマンドで値が割り当てられている場合、割り当てられた値は return_expression の評価に使用されます。

return_expression で指定された列に INSERT コマンドで値が割り当てられておらず、テーブルの列定義に列のデフォルト値がない場合、return_expression の評価では null が使用されます。

return_expression で指定された列に INSERT コマンドで値が割り当てられておらず、テーブルの列定義に列のデフォルト値がある場合、return_expression の評価ではデフォルト値が使用されます。

record

評価された return_expression が割り当てられるフィールドを持つレコード。最初の return_expression は record の最初のフィールドに割り当てられ、2 番目の return_expression

は record の 2 番目のフィールドに割り当てられます。record のフィールドの数は式の数と正確に一致する必要がありますフィールドは、割り当てられた式と型互換性がなければなりません。

variable

評価された return_expression が割り当てられる変数。複数の return_expression と variable が指定されている場合、最初の return_expression が最初の variable に割り当てられ、2 番目の return_expression が 2 番目の variable に割り当てられます。INTO キーワードは RETURNING キーワードに続く式の数と正確に一致する必要があります、変数は割り当てられた式と型互換性がなければなりません。

collection

評価された return_expression から要素が作成されるコレクション。単一フィールドのコレクションまたはレコードタイプのコレクションである単一コレクション、または複数のコレクションがあり、各コレクションが単一フィールドで構成されている必要があります。戻り式の数は一貫性が保たれ、指定されたすべてのコレクション内のフィールドの数を並べる必要があります。対応する各 return_expression および collection フィールドは、タイプ互換である必要があります。

Examples

テーブル emp に単一の行を挿入します。

```
INSERT INTO emp VALUES (8021, 'JOHN', 'SALESMAN', 7698, '22-FEB-07', 1250, 500, 30);
```

この 2 番目の例では、列 comm が省略されているため、デフォルト値は null になります。

```
INSERT INTO emp (empno, ename, job, mgr, hiredate, sal, deptno)
VALUES (8022, 'PETERS', 'CLERK', 7698, '03-DEC-06', 950, 30);
```

3 番目の例では、値を指定するのではなく、hiredate 列と comm 列に DEFAULT 句を使用しています。

```
INSERT INTO emp VALUES (8023, 'FORD', 'ANALYST', 7566, NULL, 3000, NULL, 20);
```

この例では、部門名のテーブルを作成し、dept テーブルの dname 列から選択してテーブルに挿入します。

```
CREATE TABLE deptnames (
    deptname          VARCHAR2(14)
);
INSERT INTO deptnames SELECT dname FROM dept;
```


第 66 章

LOCK

Name

LOCK -- はテーブルをロックします

Synopsis

```
LOCK TABLE <name> [, ...] IN <lockmode> MODE [ NOWAIT ]
```

ここで、lockmode は次のいずれかです。

ROW SHARE | ROW EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE

Description

LOCK TABLE はテーブルレベルのロックを取得し、必要に応じて競合するロックが解放されるのを待ちます。NOWAIT を指定すると、LOCK TABLE は目的のロックの取得を待機しません。すぐに取得できない場合、コマンドは中止され、エラーが発生します。ロックが取得されると、現在のトランザクションの残りの間ロックが保持されます。（UNLOCK TABLE コマンドはありません。ロックは常にトランザクション終了時に解放されます。）

テーブルを参照するコマンドのロックを自動的に取得する場合、AdvancedServer は常に可能な限り最小のロックモードを使用します。LOCK TABLE は、より制限的なロックが必要な場合に備えています。たとえば、アプリケーションがコミットされた分離レベルでトランザクションを実行し、テーブルのデータがトランザクションの間安定していることを確認する必要があるとします。これを実現するには、クエリを実行する前にテーブルで SHARE ロックモードを取得します。これにより、同時データ変更が防止され、テーブルの後続の読み取りでコミット済みデータの安定したビューが表示されます。これは、SHARE ロックモードがライターによって取得された ROW EXCLUSIVE ロックと競合し、LOCK TABLE 名 IN SHARE MODE ステートメントが待機するためです。ROW EXCLUSIVE モードの同時保有者がコミットまたはロールバックするまで。したがって、ロックを取得すると、未コミットの未解決の書き込みはありません。さらに、ロックを解除するまで何も開始できません。

シリアライズ可能な分離レベルでトランザクションを実行するときに同様の効果を得るには、データ変更ステートメントを実行する前に LOCK TABLE ステートメントを実行する必要があります。シリアル化可能なトランザクションのデータのビューは、最初のデータ変更ステートメントが始まると凍結されます。後の LOCK TABLE

は、同時書き込みを防ぎますが、トランザクションが読み取るものが最新のコミット値に対応することを保証しません。

この種のトランザクションがテーブル内のデータを変更する場合、SHARE モードではなく SHARE ROW EXCLUSIVE ロックモードを使用する必要があります。

これにより、このタイプのトランザクションは一度に 1 つだけ実行されます。これがないと、デッドロックが発生する可能性があります。2 つのトランザクションが両方とも SHARE モードを取得し、ROW EXCLUSIVE モードも取得して実際に更新を実行できない場合があります。(トランザクションのロックは競合しないため、トランザクションは SHARE モードを保持している場合は ROW EXCLUSIVE モードを取得できますが、他の誰かが SHARE モードを保持している場合は取得できません。) デッドロックを回避するには、すべてのトランザクションが同じオブジェクトを同じ順序で使用し、1 つのオブジェクトに複数のロックモードが関係している場合、トランザクションは常に最も制限の厳しいモードを最初に取得する必要があります。

Parameters

name

ロックする既存のテーブルの名前 (オプションでスキーマ修飾)。

コマンド `LOCK TABLE a, b;` は `LOCK TABLE a;` `LOCK TABLE b` と同等です。テーブルは、`LOCK TABLE` コマンドで指定された順序で 1 つずつロックされます。

lockmode

ロックモードは、このロックが競合するロックを指定します。

ロックモードが指定されていない場合、サーバーは最も制限の厳しいモード `ACCESS EXCLUSIVE` を使用します。(`ACCESS EXCLUSIVE` は Oracle データベースと互換性がありません。AdvancedServer では、この構成モードにより、他のトランザクションがロックされたテーブルにいかなる方法でもアクセスできなくなります。)

NOWAIT

`LOCK TABLE` は競合するロックが解放されるまで待機しないことを指定します。指定したロックを待機せずにすぐに取得できない場合、トランザクションは中止されます。

Notes

`LOCK` のすべての形式には、`UPDATE` または `DELETE` 特権が必要です。

`LOCK TABLE` は、トランザクションが終了するとすぐにロックが解除されるため、トランザクションブロック内でのみ役立ちます。トランザクションブロックの外側に表示される `LOCK TABLE` コマンドは自己完結型トランザクションを形成するため、ロックは取得されるとすぐに削除されます。

`LOCK TABLE` はテーブルレベルのロックのみを扱うため、`ROW` を含むモード名はすべて誤った呼び名です。通常、これらのモード名は、ロックされたテーブル内で行レベルのロックを取得するユーザーの意図を示すものとして読む必要があります。また、`ROW EXCLUSIVE` モードは共有可能なテーブルロックです。 `LOCK TABLE` に関

する限り、すべてのロックモードは同じセマンティクスを持ち、どのモードがどのモードと競合するかについてのルールのみが異なることに注意してください。

第 67 章

REVOKE

Name

REVOKE -- はアクセス権を削除します

Synopsis

```
REVOKE { { SELECT | INSERT | UPDATE | DELETE | REFERENCES }
  [,...] | ALL [ PRIVILEGES ] }
  ON tablename
  FROM { username | groupname | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ]

REVOKE { SELECT | ALL [ PRIVILEGES ] }
  ON sequencename
  FROM { username | groupname | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ]

REVOKE { EXECUTE | ALL [ PRIVILEGES ] }
  ON FUNCTION progname
  ( [ [ argmode ] [ argname ] argtype ] [, ...] )
  FROM { username | groupname | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ]

REVOKE { EXECUTE | ALL [ PRIVILEGES ] }
  ON PROCEDURE progname
  [ ( [ [ argmode ] [ argname ] argtype ] [, ...] ) ]
  FROM { username | groupname | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ]

REVOKE { EXECUTE | ALL [ PRIVILEGES ] }
  ON PACKAGE packagename
  FROM { username | groupname | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ]
```

(次のページに続く)

(前のページからの続き)

```
REVOKE role [, ...] FROM { username | groupname | PUBLIC }
    [, ...]
    [ CASCADE | RESTRICT ]

REVOKE { CONNECT | RESOURCE | DBA } [, ...]
    FROM { username | groupname } [, ...]

REVOKE CREATE [ PUBLIC ] DATABASE LINK
    FROM { username | groupname }

REVOKE DROP PUBLIC DATABASE LINK
    FROM { username | groupname }

REVOKE EXEMPT ACCESS POLICY
    FROM { username | groupname }
```

Description

REVOKE コマンドは、1 つ以上のロールから以前に付与された特権を取り消します。キーワード PUBLIC は、すべての役割の暗黙的に定義されたグループを指します。

特権タイプの意味については、GRANT コマンドの説明を参照してください。

特定のロールには、直接付与された特権、現在のメンバーであるロールに付与された特権、および PUBLIC に付与された特権の合計があることに注意してください。したがって、たとえば、PUBLIC から SELECT 特権を取り消すことは、必ずしもすべての役割がオブジェクトの SELECT 特権を失ったことを意味するものではありません。

権限が付与オプションで付与されていた場合、その権限の付与オプションとその権限自体が取り消されます。

ユーザーが付与オプションで特権を保持し、他のユーザーにそれを付与した場合、それらの他のユーザーが保持する特権は依存特権と呼ばれます。最初のユーザーが保持している特権または付与オプションが取り消され、依存する特権が存在する場合、CASCADE を指定するとそれらの依存する特権も取り消されます。そうでない場合、取り消しアクションは失敗します。この再帰的取り消しは、この REVOKE コマンドの対象であるユーザーを追跡可能な一連のユーザーを通じて付与された特権にのみ影響します。したがって、影響を受けるユーザーは、他のユーザーからも付与された特権を効果的に保持できます。

注釈: CASCADE は、Oracle データベースと互換性のあるオプションではありません。デフォルトでは、Oracle は常に依存特権をカスケードしますが、AdvancedServer では CASCADE キーワードを明示的に指定する必要があります。そうしないと、REVOKE コマンドが失敗します。

ロールのメンバーシップを取り消す場合、代わりに GRANT OPTION は ADMIN OPTION と呼ばれますが、動作は似ています。

Notes

ユーザーは、そのユーザーによって直接付与された特権のみを取り消すことができます。たとえば、ユーザー「13」が付与オプション付きの特権をユーザー「14」に付与し、ユーザー「14」がそれをユーザー「15」に付与した場合、ユーザー「13」は「」から直接特権を取り消すことはできません。代わりに、ユーザー「13」はユーザー「14」から付与オプションを取り消し、CASCADE オプションを使用して、ユーザー「15」から順に特権を取り消すことができます。別の例として、「13」と「14」の両方が「15」に同じ特権を付与した場合、「13」は自身の付与を取り消すことができますが、「14」の付与は取り消せないため、「15」は実質的に特権を持ちます。

オブジェクトの非所有者がオブジェクトに対する REVOKE 特権を試みると、ユーザーがオブジェクトに対する特権を持っていない場合、コマンドは完全に失敗します。何らかの特権が利用可能である限り、コマンドは続行しますが、ユーザーが許可オプションを持っている特権のみを取り消します。REVOKE ALL PRIVILEGES フォームは、付与オプションが保持されていない場合に警告メッセージを発行し、他のフォームは、コマンドで明示的に指定された特権のいずれかの付与オプションが保持されていない場合に警告を発行します。（原則として、これらのステートメントはオブジェクト所有者にも適用されますが、所有者は常にすべての付与オプションを保持しているものとして扱われるため、ケースは発生しません。）

スーパーユーザーが GRANT または REVOKE コマンドを発行することを選択した場合、コマンドは、影響を受けるオブジェクトの所有者によって発行されたかのように実行されます。すべての特権は最終的にオブジェクト所有者から取得されるため（許可オプションのチェーンを介して間接的に）、スーパーユーザーがすべての特権を取り消すことは可能ですが、これには上記の CASCADE の使用が必要になる場合があります。

REVOKE は、影響を受けるオブジェクトの所有者ではないが、オブジェクトを所有するロールのメンバーであるロール、またはオブジェクトに対する WITH GRANT OPTION の特権を保持するロールのメンバーでもあるロールによって実行できます。この場合、コマンドは、実際にオブジェクトを所有するか、特権 WITH GRANT OPTION を保持するロールによって発行されたかのように実行されます。たとえば、テーブル `t1` がロール「10」によって所有され、そのロール `u1` がメンバーである場合、`u1` は「10」によって付与されたとして記録された `t1` に対する特権を取り消すことができます。これには、`u1` とロール「10」の他のメンバーによる助成金が含まれます。

REVOKE を実行するロールが複数のロールメンバーシップパスを介して間接的に特権を保持している場合、コマンドを実行するために使用するロールが指定されていません。そのような場合は、SET ROLE を使用して REVOKE を実行する特定の役割になることがベストプラクティスです。そうしないと、意図した権限以外の権限が取り消されるか、まったく取り消されない場合があります。

注釈: AdvancedServer の ALTER ROLE コマンドは、パブリックデータベースリンクまたはプライベートデータベースリンクの作成に必要なシステム権限を取り消したり、きめ細かいアクセス制御ポリシー（DBMS_RLS）を免除する構文もサポートしています。ALTER ROLE 構文は、Oracle データベースと互換性のある REVOKE コマンドと機能的に同等です。

Examples

テーブル `emp` のパブリックの挿入特権を取り消します。

```
REVOKE INSERT ON emp FROM PUBLIC;
```

ビュー `salesemp` のユーザー `mary` からすべての特権を取り消します。

```
REVOKE ALL PRIVILEGES ON salesemp FROM mary;
```

これは実際には「私が付与したすべての特権を取り消す」ことを意味することに注意してください。

ユーザー `joe` からロール `admins` のメンバーシップを取り消します。

```
REVOKE admins FROM joe;
```

ユーザー `joe` から `CONNECT` 特権を取り消します。

```
REVOKE CONNECT FROM joe;
```

ユーザー `joe` から `CREATE DATABASE LINK` 特権を取り消します。

```
REVOKE CREATE DATABASE LINK FROM joe;
```

ユーザー `joe` から `EXEMPT ACCESS POLICY` 特権を取り消します。

```
REVOKE EXEMPT ACCESS POLICY FROM joe;
```

See Also

[*GRANT*](#)、[*SETROLE*](#)

第 68 章

ROLLBACK

Name

ROLLBACK -- 現在のトランザクションを中止します

Synopsis

ROLLBACK [WORK]

Description

ROLLBACK は現在のトランザクションをロールバックし、トランザクションによって行われたすべての更新を破棄します。

Parameters

WORK

オプションのキーワード-効果はありません。

Notes

COMMIT を使用して、トランザクションを正常に終了します。

トランザクション内にないときに ROLLBACK を発行しても害はありません。

注釈: ランタイムスタックに Oracle スタイルの SPL プロシージャがある場合、plpgsql プロシージャで ROLLBACK を実行するとエラーがスローされます。

Examples

すべての変更を中止するには:

```
ROLLBACK;
```

See Also

COMMIT、セーブポイントへのロールバック、SAVEPOINT

第 69 章

ROLLBACK TO SAVEPOINT

Name

ROLLBACK TO SAVEPOINT -- はセーブポイントにロールバックします

Synopsis

```
ROLLBACK [ WORK ] TO [ SAVEPOINT ] <savepoint_name>
```

Description

セーブポイントが確立された後に実行されたすべてのコマンドをロールバックします。セーブポイントは有効なままであり、必要に応じて再びロールバックできます。

ROLLBACK TO SAVEPOINT は、指定されたセーブポイントの後に確立されたすべてのセーブポイントを破棄します。

Parameters

savepoint_name

ロールバックするセーブポイント。

Notes

確立されていないセーブポイント名を指定するとエラーになります。

ROLLBACK TO SAVEPOINT は SPL プログラム内ではサポートされていません。

Examples

実行されたコマンドの効果を取消すために、セーブポイント `depts` が確立されました:

```
\set AUTOCOMMIT off
INSERT INTO dept VALUES (50, 'HR', 'NEW YORK');
SAVEPOINT depts;
```

(次のページに続く)

(前のページからの続き)

```
INSERT INTO emp (empno, ename, deptno) VALUES (9001, 'JONES', 50);  
INSERT INTO emp (empno, ename, deptno) VALUES (9002, 'ALICE', 50);  
ROLLBACK TO SAVEPOINT depts;
```

See Also

COMMIT、*ROLLBACK*、*SAVEPOINT*

第 70 章

SAVEPOINT

Name

SAVEPOINT -- は、現在のトランザクション内で新しいセーブポイントを定義します

Synopsis

```
SAVEPOINT <savepoint_name>
```

Description

SAVEPOINT は、現在のトランザクション内に新しいセーブポイントを確立します。

セーブポイントは、トランザクション内の特別なマークであり、確立後に実行されたすべてのコマンドをロールバックし、トランザクションの状態をセーブポイントの時点の状態に復元できます。

Parameters

savepoint_name

セーブポイントに付けられる名前。

Notes

ROLLBACK TO SAVEPOINT を使用して、セーブポイントにロールバックします。

セーブポイントは、トランザクションブロック内でのみ確立できます。トランザクション内で複数のセーブポイントを定義できます。

前のセーブポイントと同じ名前で別のセーブポイントが確立されると、古いセーブポイントが保持されますが、ロールバック時には最新のセーブポイントのみが使用されます。

SAVEPOINT は SPL プログラム内ではサポートされていません。

Examples

セーブポイントを確立し、その後、確立後に実行されたすべてのコマンドの効果を元に戻すには:

```
\set AUTOCOMMIT off
INSERT INTO dept VALUES (50, 'HR', 'NEW YORK');
SAVEPOINT depts;
INSERT INTO emp (empno, ename, deptno) VALUES (9001, 'JONES', 50);
INSERT INTO emp (empno, ename, deptno) VALUES (9002, 'ALICE', 50);
SAVEPOINT emps;
INSERT INTO jobhist VALUES (9001, '17-SEP-07', NULL, 'CLERK', 800, NULL, 50, 'New
Hire');
INSERT INTO jobhist VALUES (9002, '20-SEP-07', NULL, 'CLERK', 700, NULL, 50, 'New
Hire');
ROLLBACK TO depts;
COMMIT;
```

上記のトランザクションは dept テーブルに行をコミットしますが、 emp および jobhist テーブルへの挿入はロールバックされます。

See Also

COMMIT、*ROLLBACK*、*ROLLBACKTOSAVEPOINT*

第 71 章

SELECT

Name

SELECT -- はテーブルまたはビューから行を取得します

Synopsis

```
SELECT [ optimizer_hint ] [ ALL | DISTINCT | UNIQUE ]
  * | expression [ AS output_name ] [, ...]
FROM from_item [, ...]
[ WHERE condition ]
[ [ START WITH start_expression ]
  CONNECT BY { PRIOR parent_expr = child_expr |
              child_expr = PRIOR parent_expr }
  [ ORDER SIBLINGS BY expression [ ASC | DESC ] [, ...] ] ]
[ GROUP BY { expression | ROLLUP ( expr_list ) |
            CUBE ( expr_list ) | GROUPING SETS ( expr_list ) } [, ...]
  [ LEVEL ] ]
[ HAVING condition [, ...] ]
[ { UNION [ ALL ] | INTERSECT | MINUS } select ]
[ ORDER BY expression [ ASC | DESC ] [, ...] ]
[ FOR UPDATE [WAIT n|NOWAIT|SKIP LOCKED]]
```

ここで、from_item は次のいずれかです。

```
table_name[@dblink ] [ alias ]
( select ) alias
from_item [ NATURAL ] join_type from_item
  [ ON join_condition | USING ( join_column [, ...] ) ]
```

Description

SELECT は、1 つ以上のテーブルから行を取得します。 SELECT の一般的な処理は次のとおりです。

1. FROM リストのすべての要素が計算されます。(FROM リストの各要素は実テーブルまたは仮想テーブルで

す。) FROM リストで複数の要素が指定されている場合、それらは相互に結合されます。(以下の FROM 節を参照してください。)

2. WHERE 句を指定すると、条件を満たさないすべての行が出力から削除されます。(以下の WHERE 節を参照してください。)
3. GROUP BY 句を指定すると、出力は 1 つ以上の値で一致する行のグループに分割されます。HAVING 句が存在する場合、指定された条件を満たさないグループが削除されます。(以下の GROUP BY 節および HAVING 節を参照してください。)
4. 演算子 UNION、「10」、「11」を使用すると、複数の SELECT ステートメントの出力を組み合わせ、単一の結果セットを形成できます。UNION 演算子は、結果セットの一方または両方にあるすべての行を返します。「10」演算子は、厳密に両方の結果セットにあるすべての行を返します。「11」演算子は、最初の結果セットにあるが 2 番目の結果セットにはない行を返します。3 つの場合すべてで、重複する行は削除されます。UNION 演算子の場合、ALL を指定すると、重複は削除されません。(以下の UNION 節、「10」節、および「11」節を参照してください。)
5. 実際の出力行は、選択した各行の SELECT 出力式を使用して計算されます。(以下の SELECT リストを参照してください。)
6. CONNECT BY 句は、階層関係を持つデータを選択するために使用されます。このようなデータには、行間に親子関係があります。(CONNECT BY 節を参照してください。)
7. ORDER BY 句が指定されている場合、返される行は指定された順序でソートされます。ORDER BY が指定されていない場合、システムが生成するのに最も速い順序で行が返されます。(以下の ORDER BY clause を参照してください。)
8. DISTINCT | UNIQUE は、結果から重複行を削除します。ALL (デフォルト) は、重複を含むすべての候補行を返します。(以下の DISTINCT | UNIQUE 節を参照してください。)
9. FOR UPDATE 句により、SELECT ステートメントが選択された行を同時更新に対してロックします。(以下の FOR UPDATE 節を参照してください。)

テーブルの値を読み取るには、テーブルに対する SELECT 権限が必要です。FOR UPDATE を使用するには UPDATE 特権も必要です。

Parameters

optimizer_hint

実行計画を選択するためのオブティマイザーへのコメントが埋め込まれたヒント。オブティマイザーのヒントについては、『Oracle 開発者向けデータベース互換性リファレンスガイド』の関数と演算子を参照してください。

71.1 FROM 句

FROM 句は、SELECT ステートメントの 1 つ以上のソーステーブルを指定します。構文は次のとおりです。

```
FROM source [, ...]
```

source は次の要素のいずれかです。

table_name[@dblink]

既存のテーブルまたはビューの名前（オプションでスキーマ修飾）。dblink は、リモートデータベースを識別するデータベースリンク名です。データベースリンクの詳細については、CREATE DATABASE LINK コマンドを参照してください。

alias

エイリアスを含む FROM アイテムの代替名。エイリアスは、簡潔にするため、または自己結合のあいまいさをなくすために使用されます（同じテーブルが複数回スキャンされる場合）。エイリアスが提供されると、テーブルまたは関数の実際の名前が完全に隠されます。たとえば、FROM foo AS f の場合、SELECT の残りの部分では、この FROM 項目を foo ではなく f として参照する必要があります。

select

サブ SELECT は FROM 句に表示できます。これは、この単一の SELECT コマンドの間、出力が一時テーブルとして作成されたかのように機能します。サブ SELECT は括弧で囲む必要があり、別名を指定する必要がありますことに注意してください。

join_type

次のいずれか:

[INNER] JOIN

LEFT [OUTER] JOIN

RIGHT [OUTER] JOIN

FULL [OUTER] JOIN

CROSS JOIN

INNER および OUTER 結合タイプの場合、結合条件、つまり NATURAL、ON join_condition、または USING (join_column [, ...]) のいずれか 1 つだけを指定する必要があります。意味については以下を参照してください。CROSS JOIN の場合、これらの条項は表示されません。

JOIN 句は、2 つの FROM 項目を組み合わせます。必要に応じて括弧を使用して、ネストの順序を決定します。括弧がない場合、JOINS は左から右にネストします。いずれの場合も、JOIN は FROM 項目を区切るコンマよりも強くバインドします。

CROSS JOIN と INNER JOIN は単純なデカルト積を生成します。これは、FROM の最上位にある 2 つのアイテムをリストした場合と同じ結果ですが、結合条件（存在する場合）によって制限されます。CROSS JOIN は INNER JOIN ON (TRUE) と同等です。つまり、修飾によって行は削除されません。これらの結合タイプは、プレーンな FROM と WHERE ではできないことは何もしないので、単なる表記上の便宜です。

LEFT OUTER JOIN は、修飾されたデカルト積のすべての行（つまり、結合条件に合格したすべての結合行）と、結合条件に合格した右側の行がなかった左側のテーブルの各行のコピーを 1 つ返します。この左側の行は、右側の列にヌル値を挿入することにより、結合されたテーブルの全幅に拡張されます。どの行が一致するかを決定する際に、JOIN 句の条件のみが考慮されることに注意してください。その後、外部条件が適用されます。

逆に、RIGHT OUTER JOIN は、結合されたすべての行に加えて、一致しない右側の行ごとに 1 行を返します（左側にヌルが拡張されます）。左右の入力を切り替えることで LEFT OUTER JOIN に変換できるため、これは単なる表記上の利便性です。

FULL OUTER JOIN は、結合されたすべての行に加えて、一致しない左側の行ごとに 1 行（右側に null が拡張）、および一致しない右側の行ごとに 1 行（左側に null が拡張）を返します。

ON join_condition

join_condition は、結合内のどの行が一致すると見なされるかを指定する BOOLEAN 型の値（WHERE 句に類似）をもたらす式です。

USING (join_column [, ...])

USING (a, b, ...) という形式の句は ON left_table.a = right_table.a AND left_table.b = right_table.b の省略形です。また、USING は、同等の列の各ペアの一方のみが結合出力に含まれ、両方は含まれないことを意味します。

NATURAL

NATURAL は、同じ名前を持つ 2 つのテーブルのすべての列に言及する USING リストの省略形です。

複数のソースが指定されている場合、結果はすべてのソースのデカルト積（クロス結合）です。通常、返される行をデカルト積の小さなサブセットに制限するために、修飾条件が追加されます。

Example

次の例は、dept テーブルからすべてのエントリを選択します。

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

71.2 WHERE 句

オプションの WHERE 句の形式は次のとおりです。

```
WHERE condition
```

ここで、condition は、タイプ BOOLEAN の結果に評価される任意の式です。この条件を満たさない行は出力から削除されます。実際の行の値が変数参照に置き換えられたときに TRUE を返す場合、行は条件を満たします。

Example

次の例では、emp テーブルと dept テーブルの内容を結合します。WHERE emp テーブルの deptno 列の値は deptno の deptno 列の値と等しくなります表:

```
SELECT d.deptno, d.dname, e.empno, e.ename, e.mgr, e.hiredate
FROM emp e, dept d
WHERE d.deptno = e.deptno;
```

deptno	dname	empno	ename	mgr	hiredate
10	ACCOUNTING	7934	MILLER	7782	23-JAN-82 00:00:00
10	ACCOUNTING	7782	CLARK	7839	09-JUN-81 00:00:00
10	ACCOUNTING	7839	KING		17-NOV-81 00:00:00
20	RESEARCH	7788	SCOTT	7566	19-APR-87 00:00:00
20	RESEARCH	7566	JONES	7839	02-APR-81 00:00:00
20	RESEARCH	7369	SMITH	7902	17-DEC-80 00:00:00
20	RESEARCH	7876	ADAMS	7788	23-MAY-87 00:00:00
20	RESEARCH	7902	FORD	7566	03-DEC-81 00:00:00
30	SALES	7521	WARD	7698	22-FEB-81 00:00:00
30	SALES	7844	TURNER	7698	08-SEP-81 00:00:00
30	SALES	7499	ALLEN	7698	20-FEB-81 00:00:00
30	SALES	7698	BLAKE	7839	01-MAY-81 00:00:00
30	SALES	7654	MARTIN	7698	28-SEP-81 00:00:00
30	SALES	7900	JAMES	7698	03-DEC-81 00:00:00

(14 rows)

71.3 GROUPBY 句

オプションの GROUP BY 句の形式は次のとおりです。

```
GROUP BY { expression | ROLLUP ( expr_list ) |
          CUBE ( expr_list ) | GROUPING SETS ( expr_list ) } [, ...]
```

GROUP BY は、グループ化された式と同じ値を共有するすべての選択された行を単一の行に凝縮します。expression は、入力列名、出力列の名前または序数 (SELECT リスト項目)、または入力列の値から形成される任意の式です。あいまいな場合、GROUP BY の名前は出力列名ではなく入力列名として解釈されます。

ROLLUP、CUBE、および GROUPING SETS は、多次元分析をサポートするための GROUP BY 節の拡張です。

集計関数を使用されている場合、各グループを構成するすべての行で集計関数が計算され、各グループに個別の値が生成されます（GROUP BY がない場合、集計は選択されたすべての行で計算された単一の値を生成します）。GROUP BY が存在する場合、SELECT リスト式が集計関数内を除き、グループ化されていない列を参照することは無効です。グループ化されていない列に対して返される値は複数あるためです。

Example

次の例では、emp テーブルの sal 列の合計を計算し、結果を部門番号でグループ化します。

```
SELECT deptno, SUM(sal) AS total
  FROM emp
 GROUP BY deptno;
```

deptno	total
10	8750.00
20	10875.00
30	9400.00

(3 rows)

71.4 HAVING 句

オプションの HAVING 句の形式は次のとおりです。

```
HAVING condition
```

ここで、condition は WHERE 句で指定したものと同じです。

HAVING は、指定された条件を満たさないグループ行を削除します。HAVING は WHERE; WHERE とは異なり、GROUP BY を適用する前に個々の行をフィルタリングしますが、HAVING は GROUP BY によって作成されたグループ行をフィルタリングします。condition で参照される各列は、集約関数内に参照が表示されない限り、明確にグループ化列を参照する必要があります。

Example

すべての従業員の sal 列を合計するには、部門番号で結果をグループ化し、10000 未満のグループ合計を表示します。

```
SELECT deptno, SUM(sal) AS total
  FROM emp
 GROUP BY deptno
 HAVING SUM(sal) < 10000;
```

deptno	total
--------	-------

(次のページに続く)

(前のページからの続き)

```

-----+-----
      10 | 8750.00
      30 | 9400.00
(2 rows)

```

71.5 選択リスト

SELECT リスト（キーワード SELECT と FROM の間）は、SELECT ステートメントの出力行を形成する式を指定します。式は、FROM 句で計算された列を参照できます（通常は参照します）。AS output_name 句を使用すると、出力列に別の名前を指定できます。この名前は主に、表示する列にラベルを付けるために使用されます。また、ORDER BY および GROUP BY 句で列の値を参照するために使用できますが、WHERE または HAVING 句では使用できません。そこで、代わりに式を書き出す必要があります。

式の代わりに、選択した行のすべての列の省略形として*を出力リストに書き込むことができます。

Example

次の例の SELECT リストは、結果セットに empno 列、ename 列、mgr 列、および hiredate 列を含めることを指定しています。

```

SELECT empno, ename, mgr, hiredate FROM emp;

 empno | ename  | mgr  |      hiredate
-----+-----+-----+-----
  7934 | MILLER | 7782 | 23-JAN-82 00:00:00
  7782 | CLARK  | 7839 | 09-JUN-81 00:00:00
  7839 | KING   |      | 17-NOV-81 00:00:00
  7788 | SCOTT  | 7566 | 19-APR-87 00:00:00
  7566 | JONES  | 7839 | 02-APR-81 00:00:00
  7369 | SMITH  | 7902 | 17-DEC-80 00:00:00
  7876 | ADAMS  | 7788 | 23-MAY-87 00:00:00
  7902 | FORD   | 7566 | 03-DEC-81 00:00:00
  7521 | WARD   | 7698 | 22-FEB-81 00:00:00
  7844 | TURNER | 7698 | 08-SEP-81 00:00:00
  7499 | ALLEN  | 7698 | 20-FEB-81 00:00:00
  7698 | BLAKE  | 7839 | 01-MAY-81 00:00:00
  7654 | MARTIN | 7698 | 28-SEP-81 00:00:00
  7900 | JAMES  | 7698 | 03-DEC-81 00:00:00
(14 rows)

```

71.6 UNION 句

UNION 句の形式は次のとおりです。

```
select_statement UNION [ ALL ] select_statement
```

`select_statement` は、ORDER BY または FOR UPDATE 節のない SELECT ステートメントです。（ORDER BY は、括弧で囲まれている場合、部分式に付加できます。括弧がない場合、これらの句は、右側の入力式ではなく、UNION の結果に適用されます。）

UNION 演算子は、関連する SELECT ステートメントによって返される行の集合を計算します。行は、結果セットの少なくとも 1 つに表示される場合、2 つの結果セットの和集合にあります。UNION の直接オペランドを表す 2 つの SELECT ステートメントは、同じ数の列を生成する必要があり、対応する列は互換性のあるデータ型である必要があります。

ALL オプションを指定しない限り、UNION の結果には重複行が含まれません。ALL は重複の排除を防ぎます。

同じ SELECT ステートメント内の複数の UNION 演算子は、括弧で特に指定されていない限り、左から右に評価されます。

現在、FOR UPDATE は UNION の結果にも UNION の入力にも指定できません。

71.7 INTERSECT 句

INTERSECT 句の形式は次のとおりです。

```
select_statement INTERSECT select_statement
```

`select_statement` は、ORDER BY または FOR UPDATE 句のない SELECT ステートメントです。

INTERSECT 演算子は、関連する SELECT ステートメントによって返される行の集合交差を計算します。行は、両方の結果セットに表示される場合、2 つの結果セットの共通部分にあります。

INTERSECT の結果には、重複する行は含まれません。

同じ SELECT ステートメント内の複数の INTERSECT 演算子は、括弧で別段の指示がない限り、左から右に評価されます。INTERSECT は UNION よりも強く結合します。つまり、`A UNION B INTERSECT C` は `A UNION (B INTERSECT C)` と読み替えられます。

71.8 マイナス句

MINUS 句の一般的な形式は次のとおりです。

```
select_statement MINUS select_statement
```

`select_statement` は、ORDER BY または FOR UPDATE 句のない SELECT ステートメントです。

MINUS 演算子は、左の SELECT ステートメントの結果に含まれる行のセットを計算しますが、右の SELECT ステートメントの結果には含まれません。

MINUS の結果には、重複する行は含まれません。

同じ SELECT ステートメント内の複数の MINUS 演算子は、括弧で別段の指示がない限り、左から右に評価されます。MINUS は UNION と同じレベルでバインドします。

71.9 CONNECTBY 句

CONNECT BY 句は、階層クエリを実行するときに行の親子関係を決定します。一般的な形式は次のとおりです。

```
CONNECT BY { PRIOR parent_expr = child_expr |
            child_expr = PRIOR parent_expr }
```

parent_expr は、候補の親行で評価されます。FROM 句によって返された行に対して parent_expr = child_expr の結果 TRUE になる場合、この行は親の子と見なされます。

次のオプションの句は、CONNECT BY 句と組み合わせて指定できます。

START WITH start_expression

start_expression が TRUE と評価される FROM 句によって返される行は、階層のルートノードになります。

ORDER SIBLINGS BY expression [ASC | DESC] [, ...]

階層の兄弟行は、結果セットの expression で順序付けられます。

注釈: AdvancedServer は、CONNECT BY 句での AND（または他の演算子）の使用をサポートしていません。

71.10 ORDERBY 句

オプションの ORDER BY 句の形式は次のとおりです。

```
ORDER BY expression [ ASC | DESC ] [, ...]
```

expression は、出力列の名前または序数（SELECT リスト項目）であるか、入力列の値から形成される任意の式です。

ORDER BY 句を使用すると、指定した式に従って結果行がソートされます。左端の式に従って 2 つの行が等しい場合、次の式に従って比較されます。指定されたすべての式に従って等しい場合、実装依存の順序で返されます。

序数は、結果列の序数（左から右）の位置を指します。この機能により、一意の名前を持たない列に基づいて順序を定義できます。AS 句を使用して結果列に名前を割り当てることが常に可能であるため、これは絶対に必要ではありません。

SELECT 結果リストに表示されない列を含む ORDER BY 句で任意の式を使用することもできます。したがって、次のステートメントは有効です。

```
SELECT ename FROM emp ORDER BY empno;
```

この機能の制限は、UNION、INTERSECT、または MINUS 句の結果に適用される ORDER BY 句は、式ではなく出力列名または番号のみを指定できることです。

ORDER BY 式が結果列名と入力列名の両方に一致する単純な名前である場合、ORDER BY はそれを結果列名として解釈します。これは、GROUP BY が同じ状況で行う選択の反対です。この不整合は、SQL 標準との互換性を保つために作成されています。

オプションで、ORDER BY 句の式の後にキーワード ASC（昇順）または DESC（降順）を追加できます。指定しない場合、デフォルトで ASC が想定されます。

null 値は他のどの値よりも高くソートされます。つまり、昇順のソート順では null 値が最後にソートされ、降順のソート順では null 値が先頭にソートされます。

文字列データは、データベースクラスターの初期化時に確立されたロケール固有の照合順序に従って並べ替えられます。

注釈: SELECT DISTINCT が指定されている場合、または SELECT ステートメントに SELECT DISTINCT ...ORDER BY 句が含まれている場合、ORDER BY のすべての式が SELECT DISTINCT クエリの選択リストに存在する必要があります。

Examples

次の 2 つの例は、2 番目の列（dname）の内容に従って個々の結果をソートする同一の方法です。

```
SELECT * FROM dept ORDER BY dname;
```

deptno	dname	loc
10	ACCOUNTING	NEW YORK
40	OPERATIONS	BOSTON
20	RESEARCH	DALLAS
30	SALES	CHICAGO

(4 rows)

```
SELECT * FROM dept ORDER BY 2;
```

(次のページに続く)

(前のページからの続き)

```

deptno |      dname      |      loc
-----+-----+-----
      10 | ACCOUNTING     | NEW YORK
      40 | OPERATIONS     | BOSTON
      20 | RESEARCH       | DALLAS
      30 | SALES          | CHICAGO
(4 rows)

```

次の例では、SELECT DISTINCT ...ORDER BY 句を使用して、テーブル emp から job と deptno をフェッチします。

```

CREATE TABLE EMP (EMPNO NUMBER(4) NOT NULL,
ENAME VARCHAR2(10),
JOB VARCHAR2(9),
DEPTNO NUMBER(2));

```

```

INSERT INTO EMP VALUES (7369, 'SMITH', 'CLERK', 20);
INSERT 0 1
INSERT INTO EMP VALUES (7499, 'ALLEN', 'SALESMAN', 30);
INSERT 0 1
INSERT INTO EMP VALUES (7521, 'WARD', 'SALESMAN', 30);
INSERT 0 1
INSERT INTO EMP VALUES (7566, 'JONES', 'MANAGER', 20);
INSERT 0 1

```

```

SELECT DISTINCT e.job, e.deptno FROM emp e ORDER BY e.job, e.deptno;
 job      | deptno
-----+-----
CLERK     |      20
MANAGER   |      20
SALESMAN  |      30
(3 rows)

```

71.11 DISTINCT|一意句

SELECT ステートメントが DISTINCT または UNIQUE を指定する場合、すべての重複行が結果セットから削除されます（重複の各グループから 1 行が保持されます）。DISTINCT または UNIQUE 節は、SELECT ステートメントと一緒に使用すると同義です。ALL キーワードは反対を指定します。すべての行が保持されます。それがデフォルトです。

DISTINCT または UNIQUE キーワードを含む SELECT ステートメントの不適切な使用に起因するエラーメッセージには、以下に示すように DISTINCT | UNIQUE キーワードの両方が含まれます。

```
psql: ERROR: FOR UPDATE is not allowed with DISTINCT/UNIQUE clause
```

71.12 FORUPDATE 句

FOR UPDATE 句の形式は次のとおりです。

```
FOR UPDATE [WAIT n|NOWAIT|SKIP LOCKED]
```

FOR UPDATE を指定すると、SELECT ステートメントによって取得された行が更新のためにロックされます。これにより、現在のトランザクションが終了するまで、行が他のトランザクションによって変更または削除されるのを防ぎます。選択した行を UPDATE、DELETE、または SELECT FOR UPDATE にしようとするトランザクションは、現在のトランザクションが終了するまでブロックされます。別のトランザクションの UPDATE、DELETE、または SELECT FOR UPDATE がすでに選択した行をロックしている場合、SELECT FOR UPDATE は最初のトランザクションが完了するのを待ってから、更新された行をロックして返します（または行（行が削除された場合））。

FOR UPDATE は、返された行を個々のテーブル行で明確に識別できないコンテキストでは使用できません（たとえば、集計で）。

FOR UPDATE オプションを使用してロック設定を指定します。

- WAIT n キーワードを含めて、SELECT ステートメントが別のセッションによってロックされた行を待機する秒数（または小数秒）を指定します。小数秒を指定するには、小数形式を使用します。たとえば、WAIT 1.5 は 1 秒半待機するようサーバーに指示します。小数点の右側に 4 桁まで指定します。
- 現在のセッションで行をロックできない場合にすぐにエラーを報告するには、NOWAIT キーワードを含めます。
- SKIP LOCKED を含めて、可能であればサーバーに行をロックするように指示し、別のセッションによって既にロックされている行をスキップします。

第 72 章

SET CONSTRAINTS

Name

SET CONSTRAINTS -- は、現在のトランザクションの制約チェックモードを設定します

Synopsis

```
SET CONSTRAINTS { ALL | name [, ...] } { DEFERRED | IMMEDIATE }
```

Description

SET CONSTRAINTS は、現在のトランザクション内の制約チェックの動作を設定します。IMMEDIATE 制約は、各ステートメントの最後にチェックされます。DEFERRED 制約は、トランザクションがコミットされるまでチェックされません。各制約には、独自の IMMEDIATE または DEFERRED モードがあります。

作成時に、制約には DEFERRABLE INITIALLY DEFERRED、DEFERRABLE INITIALLY IMMEDIATE、または NOT DEFERRABLE の 3 つの特性のいずれかが与えられます。3 番目のクラスは常に IMMEDIATE であり、SET CONSTRAINTS コマンドの影響を受けません。最初の 2 つのクラスは指定されたモードですべてのトランザクションを開始しますが、それらの動作は SET CONSTRAINTS によってトランザクション内で変更できます。

制約名のリストを持つ SET CONSTRAINTS は、それらの制約だけのモードを変更します（すべて遅延可能でなければなりません）。特定の名前に一致する複数の制約がある場合、すべてが影響を受けます。SET CONSTRAINTS ALL は、すべての遅延可能な制約のモードを変更します。

SET CONSTRAINTS が制約のモードを DEFERRED から IMMEDIATE に変更すると、新しいモードがさかのぼって有効になります。トランザクションの終了時にチェックされていた未処理のデータ変更は、代わりに SET CONSTRAINTS コマンド。そのような制約に違反すると、SET CONSTRAINTS は失敗します（制約モードは変更されません）。したがって、SET CONSTRAINTS を使用して、トランザクションの特定のポイントで制約のチェックを強制することができます。

現在、外部キー制約のみがこの設定の影響を受けます。チェック制約および一意制約は、常に事実上遅延可能ではありません。

Notes

このコマンドは、現在のトランザクション内の制約の動作のみを変更します。したがって、トランザクションブロックの外部でこのコマンドを実行すると、効果がないように見えます。

第 73 章

SET ROLE

Name

SET ROLE -- は、現在のセッションの現在のユーザー識別子を設定します

Synopsis

```
SET ROLE { rolename | NONE }
```

Description

このコマンドは、現在の SQL セッションコンテキストの現在のユーザー識別子を rolename に設定します。SET ROLE の後、指定されたロールが最初にログインしたロールであるかのように、SQL コマンドの許可チェックが実行されます。

指定された rolename は、現在のセッションユーザーがメンバーになっているロールでなければなりません。セッションユーザーがスーパーユーザーの場合、任意のロールを選択できます。

NONE は、現在のユーザー ID を現在のセッションユーザー ID にリセットします。これらのフォームは、どのユーザーでも実行できます。

Notes

このコマンドを使用して、特権を追加したり、特権を制限したりできます。セッションユーザーロールに INHERITS 属性がある場合、SET ROLE にできるすべてのロールのすべての特権が自動的にあります。この場合、SET ROLE は、セッションユーザーとそれがメンバーである他のロールに直接割り当てられたすべての特権を事実上削除し、指定されたロールで使用可能な特権のみを残します。セッションユーザーロールに NOINHERITS 属性がある場合、SET ROLE はセッションユーザーに直接割り当てられた特権をドロップし、代わりに指定されたロールで使用可能な特権を取得します。スーパーユーザーが非スーパーユーザーの役割に対して SET ROLE を選択すると、スーパーユーザーの特権が失われます。

Examples

ユーザー mary はロール admins の ID を引き継ぎます:

```
SET ROLE admins;
```

ユーザー mary は自分の ID に戻ります。

```
SET ROLE NONE;
```

第 74 章

SET TRANSACTION

Name

SET TRANSACTION -- は現在のトランザクションの特性を設定します

Synopsis

```
SET TRANSACTION transaction_mode
```

ここで、transaction_mode は次のいずれかです。

```
ISOLATION LEVEL { SERIALIZABLE | READ COMMITTED }
```

```
READ WRITE | READ ONLY
```

Description

SET TRANSACTION コマンドは、現在のトランザクションの特性を設定します。後続のトランザクションには影響しません。使用可能なトランザクション特性は、トランザクション分離レベルとトランザクションアクセスモード（読み取り/書き込みまたは読み取り専用）です。トランザクションの分離レベルは、他のトランザクションが同時に実行されているときにトランザクションが表示できるデータを決定します。

READ COMMITTED

ステートメントは、開始前にコミットされた行のみを表示できます。これがデフォルトです。

SERIALIZABLE

現在のトランザクションのすべてのステートメントは、このトランザクションで最初のクエリまたはデータ変更ステートメントが実行される前にコミットされた行のみを見ることができます。

トランザクションの最初のクエリまたはデータ変更ステートメント（SELECT、INSERT、DELETE、UPDATE、または FETCH）が実行された後は、トランザクション分離レベルを変更できません。トランザクションアクセスモードは、トランザクションが読み取り/書き込みまたは読み取り専用かどうかを決定します。読み取り/書き込みがデフォルトです。

トランザクションが読み取り専用の場合、次の SQL コマンドは許可されません。書き込み先のテーブルが一時テーブルでない場合、INSERT、UPDATE、DELETE。すべての CREATE、ALTER、および DROP コマンド。COMMENT、GRANT、REVOKE、TRUNCATE；実行するコマンドがリストにあるものの場合は「10」。これは、ディスクへのすべての書き込みを妨げない読み取り専用の高レベルの概念です。

第 75 章

TRUNCATE

Name

TRUNCATE -- はテーブルを空にします

Synopsis

```
TRUNCATE TABLE <name> [DROP STORAGE]
```

Description

TRUNCATE は、テーブルからすべての行をすばやく削除します。非修飾の DELETE と同じ効果がありますが、実際にはテーブルをスキャンしないため、高速です。これは、大きなテーブルで最も役立ちます。

DROP STORAGE 句は互換性のために受け入れられますが、無視されます。

Parameters

name

切り捨てられるテーブルの名前（オプションでスキーマ修飾）。

Notes

他のテーブルからテーブルへの外部キー参照がある場合、TRUNCATE は使用できません。このような場合に有効性をチェックするには、テーブルスキャンが必要になりますが、全体のポイントはそれを行うことではありません。

TRUNCATE は、テーブルに存在する可能性のあるユーザー定義の ON DELETE トリガーを実行しません。

Examples

次のコマンドは、accounts という名前のテーブルを切り捨てます。

```
TRUNCATE TABLE accounts;
```

See Also

ドロップビュー、DELETE

第 76 章

UPDATE

Name

UPDATE -- はテーブルの行を更新します

Synopsis

```
UPDATE [ <optimizer_hint> ] <table>[@<dblink> ]
    SET <column> = { <expression> | DEFAULT } [, ...]
[ WHERE <condition> ]
[ RETURNING <return_expression> [, ...]
    { INTO { <record> | <variable> [, ...] }
    | BULK COLLECT INTO <collection> [, ...] } ]
```

Description

UPDATE は、条件を満たすすべての行の指定された列の値を変更します。変更する列のみを SET 句で指定する必要があります。明示的に変更されていない列は、以前の値を保持します。

RETURNING INTO { record | variable [, ...] } 句は、SPL プログラム内でのみ指定できます。さらに、UPDATE コマンドの結果セットは複数の行を返してはなりません。そうしないと、例外がスローされます。結果セットが空の場合、ターゲットレコードまたは変数の内容は null に設定されます。

RETURNING BULK COLLECT INTO collection [, ...] 節は、UPDATE コマンドが SPL プログラム内で使用される場合にのみ指定できます。BULK COLLECT INTO 節のターゲットとして複数の「10」が指定されている場合、各「10」は単一のスカラーフィールドで構成されている必要があります。つまり、「10」はレコードであってはなりません。UPDATE コマンドの結果セットには、0 行、1 行、または複数行が含まれる場合があります。結果セットの各行で評価された return_expression は、最初の要素から始まる「10」の要素になります。「10」の既存の行はすべて削除されます。結果セットが空の場合、「10」は空になります。

テーブルを更新するには、そのテーブルに対する UPDATE 特権と、expression または condition で値が読み取られるテーブルに対する SELECT 特権が必要です。

Parameters

`optimizer_hint`

実行計画を選択するためのオプティマイザーへのコメントが埋め込まれたヒント。

`table`

更新するテーブルの名前（オプションでスキーマ修飾）。

`dblink`

リモートデータベースを識別するデータベースリンク名。データベースリンクの詳細については、`CREATE DATABASE LINK` コマンドを参照してください。

`column`

テーブル内の列の名前。

`expression`

列に割り当てる式。式では、この列とテーブル内の他の列の古い値を使用できます。

`DEFAULT`

列をデフォルト値に設定します（特定のデフォルト式が割り当てられていない場合、`null` になります）。

`condition`

タイプ `BOOLEAN` の値を返す式。この式が `true` を返す行のみが更新されます。

`return_expression`

テーブルの 1 つ以上の列を含む可能性のある式。テーブルの列名が `return_expression` に指定されている場合、`return_expression` の評価時に列に代入される値は次のように決定されます。

`return_expression` で指定された列に `UPDATE` コマンドで値が割り当てられている場合、割り当てられた値は `return_expression` の評価に使用されます。

`return_expression` で指定された列に `UPDATE` コマンドで値が割り当てられていない場合、影響を受ける行の列の現在の値が `return_expression` の評価に使用されます。

`record`

評価された `return_expression` が割り当てられるフィールドを持つレコード。最初の `return_expression` は `record` の最初のフィールドに割り当てられ、2 番目の `return_expression` は `record` の 2 番目のフィールドに割り当てられます。 `record` のフィールドの数は式の数と正確に一致する必要がありますフィールドは、割り当てられた式と型互換性がなければなりません。

`variable`

評価された `return_expression` が割り当てられる変数。複数の `return_expression` と `variable` が指定されている場合、最初の `return_expression` が最初の `variable` に割り当て

られ、2 番目の `return_expression` が 2 番目の `variable` に割り当てられます。INTO キーワードは RETURNING キーワードに続く式の数と正確に一致する必要があり、変数は割り当てられた式と型互換性がなければなりません。

`collection`

評価された `return_expression` から要素が作成されるコレクション。単一フィールドのコレクションまたはレコードタイプのコレクションである単一コレクション、または複数のコレクションがあり、各コレクションが単一フィールドで構成されている必要があります。戻り式の数は数が一致し、指定されたすべてのコレクション内のフィールドの数を並べる必要があります。対応する各 `return_expression` および `collection` フィールドは、タイプ互換である必要があります。

Examples

dept テーブルの部門 20 の場所を AUSTIN に変更します。

```
UPDATE dept SET loc = 'AUSTIN' WHERE deptno = 20;
```

emp テーブルに `job = SALESMAN` があるすべての従業員について、10% で給与を更新し、500 で手数料を増やします。

```
UPDATE emp SET sal = sal * 1.1, comm = comm + 500 WHERE job = 'SALESMAN';
```


第 77 章

Conclusion

Oracle®DevelopersSQL ガイドのための EDBPostgres™AdvancedServer データベース互換性

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、EDB 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

ALTER DIRECTORY, 3
 ALTER INDEX, 5
 ALTER PROCEDURE, 7
 ALTER PROFILE, 9
 ALTER QUEUE, 13
 ALTER QUEUE TABLE, 17
 ALTER ROLE – Managing Database Link and
 DBMS_RLS Privileges, 21
 ALTER ROLE... IDENTIFIED BY, 19
 ALTER SEQUENCE, 25
 ALTER SESSION, 27
 ALTER TABLE, 29
 ALTER TABLESPACE, 39
 ALTER TRIGGER, 35
 ALTER USER|ROLE... PROFILE MANAGEMENT CLAUSES, 43
 ALTER USER... IDENTIFIED BY, 41

CALL, 47
 COMMENT, 49
 COMMIT, 51
 Conclusion, 261
 CREATE DATABASE, 53
 CREATE DIRECTORY, 69
 CREATE FUNCTION, 71
 CREATE INDEX, 79
 CREATE MATERIALIZED VIEW, 83
 CREATE PACKAGE, 85
 CREATE PACKAGE BODY, 89
 CREATE PROCEDURE, 97
 CREATE PROFILE, 105
 CREATE PUBLIC DATABASE LINK, 55
 CREATE QUEUE, 109
 CREATE QUEUE TABLE, 111
 CREATE ROLE, 115
 CREATE SCHEMA, 117
 CREATE SEQUENCE, 119
 CREATE SYNONYM, 123
 CREATE TABLE, 125
 CREATE TABLE AS, 135
 CREATE TRIGGER, 137
 CREATE TYPE, 149
 CREATE TYPE BODY, 157
 CREATE USER, 161
 CREATE USER|ROLE... PROFILE MANAGEMENT CLAUSES,
 163
 CREATE VIEW, 167

DELETE, 169
 DROP DATABASE LINK, 173
 DROP DIRECTORY, 175
 DROP FUNCTION, 177
 DROP INDEX, 179
 DROP PACKAGE, 181
 DROP PROCEDURE, 183
 DROP PROFILE, 185
 DROP QUEUE, 187
 DROP QUEUE TABLE, 189

DROP ROLE, 193
 DROP SEQUENCE, 195
 DROP SYNONYM, 191
 DROP TABLE, 197
 DROP TABLESPACE, 199
 DROP TRIGGER, 201
 DROP TYPE, 203
 DROP USER, 205
 DROP VIEW, 207

EXEC, 209

GRANT, 211

INSERT, 219

LOCK, 223

REVOKE, 227
 ROLLBACK, 231
 ROLLBACK TO SAVEPOINT, 233

SAVEPOINT, 235
 SELECT, 237
 SET CONSTRAINTS, 249
 SET ROLE, 251
 SET TRANSACTION, 253

TRUNCATE, 255

UPDATE, 257