



Database Compatibility for Oracle Developer's Tools and Utilities Guide

リリース **13**

EnterpriseDB

2020 年 12 月 30 日

目次

第 1 章	Introduction	1
第 2 章	EDB*Loader	3
2.1	Data Loading Methods	5
2.2	General Usage	6
2.3	Building the EDB*Loader Control File	7
2.3.1	EDB Loader Control File Examples	19
2.4	Invoking EDB*Loader	29
2.4.1	Exit Codes	36
2.5	Direct Path Load	37
2.6	Parallel Direct Path Load	38
2.7	Remote Loading	42
2.8	Updating a Table with a Conventional Path Load	43
第 3 章	EDB*Wrap	45
3.1	Using EDB*Wrap to Obfuscate Source Code	46
第 4 章	動的ランタイム計装ツールアーキテクチャ (DRITA)	51
4.1	Configuring and Using DRITA	51
4.2	DRITA Functions	53
4.2.1	get_snaps()	53
4.2.2	sys_rpt()	53
4.2.3	sess_rpt()	54
4.2.4	sessid_rpt()	56
4.2.5	sesshist_rpt()	57
4.2.6	purgesnap()	61
4.2.7	truncsnap()	62
4.3	Simulating Statspack AWR Reports	63
4.3.1	edbreport()	63
4.3.2	stat_db_rpt()	73
4.3.3	stat_tables_rpt()	74
4.3.4	statio_tables_rpt()	77
4.3.5	stat_indexes_rpt()	79

4.3.6	statio_indexes_rpt()	81
4.4	Performance Tuning Recommendations	84
4.5	Event Descriptions	85
第 5 章	Conclusion	89
索引		91

第 1 章

Introduction

このガイドに記載されているツールとユーティリティを使用すると、Oracle ユーティリティを操作することに慣れている開発者は、使い慣れた環境で AdvancedServer を操作できます。

このガイドのセクションでは、AdvancedServer でサポートされている互換性のあるツールとユーティリティについて説明します。これらには以下が含まれます：

- EDB*Loader
- EDB*Wrap
- 動的ランタイム計測

EDB*Plus コマンドラインクライアントは、SQL*Plus コマンドをサポートする AdvancedServer へのユーザーインターフェイスを提供します。EDB* Plus でできること：

- データベースオブジェクトのクエリ
- ストアドプロシージャを実行する
- SQL コマンドからの出力のフォーマット
- バッチスクリプトを実行する
- OS コマンドを実行する
- レコード出力

EDB*Plus の詳細なインストールおよび使用情報については、次の EDBWeb サイトから入手できる*EDB*Plus ユーザーガイド* を参照してください。

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

AdvancedServer がサポートする機能の詳細については、次の Web サイトで入手可能な AdvancedServer ガイドの完全なライブラリを参照してください。

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

第 2 章

EDB*Loader

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

EDB*Loader の機能は次のとおりです。

- OracleSQL*Loader データのロード方法のサポート-従来のパス・ロード、ダイレクト・パス・ロードおよびパラレル・ダイレクト・パス・ロード
- OracleSQL*Loader と互換性のある制御ファイルのディレクティブの構文
- 区切り文字で区切られたフィールドまたは固定幅のフィールドを持つ入力データ
- 拒否されたレコードを収集するための不良ファイル
- 複数のターゲットテーブルの読み込み
- 対象テーブルの選択基準を満たさないレコードを収集するためのファイルを破棄します
- EDB*Loader セッションとエラーメッセージを記録するためのログファイル
- 標準入力からのデータ読み込みとリモート読み込み、特にリモートホスト上の大規模なデータソースに役立ちます

これらの機能については、次のセクションで詳しく説明します。

注釈: 以下は、EDB*Loader クライアントとデータベースサーバー間の重要なバージョン互換性の制限です。

- EDB*Loader プログラム（edblldr と呼ばれる）を呼び出すと、パラメーターとディレクティブ情報をデータベースサーバーに渡します。edblldr
- PgPool-II や PgBouncer などの接続プーラーと組み合わせた EDB*Loader の使用はサポートされていません。EDB*Loader は AdvancedServer バージョン 13 に直接接続する必要があります。また、次のコマンド

は、接続プールを通じてデータをロードするために使用できるオプションの一部です。

- `psql /copy`
- `jdbc copyIn`
- `psycopg2 copy_from`
- バージョン 9.2 以前の AdvancedServer では、バージョン 13、12、11、10、9.6、または 9.5EDB*Loader クライアントの使用はサポートされていません。

2.1 Data Loading Methods

OracleSQL*Loader と同様に、EDB* Loader は 3 つのデータロード方法をサポートしています。

- 従来のパス負荷
- ダイレクトパスロード
- パラレルダイレクトパスロード

従来のパスロードは、EDB*Loader で使用されるデフォルトの方法です。基本的な挿入処理は、テーブルに行を追加するために使用されます。

他のメソッドに対する従来のパスロードの利点は、テーブル制約と、null 制約、チェック制約、一意のインデックス、外部キー制約、トリガーなどのテーブルで定義されたデータベースオブジェクトが従来のパスロード中に実施されることです。

1 つの例外は、テーブルで定義された AdvancedServer の*ルール*が適用されないことです。EDB*Loader は、ルールが定義されているテーブルをロードできますが、ルールは実行されません。結果として、ルールを使用して実装されたパーティションテーブルは、EDB* Loader を使用してロードできません。

注釈: AdvancedServer ルールは、CREATE RULE コマンドによって作成されます。AdvancedServer ルールは、Oracle で使用されるルールおよびルールセットと同じデータベースオブジェクトではありません。

EDB*Loader は、ダイレクトパスロードもサポートしています。直接パスロードは、従来のパスロードよりも高速ですが、ほとんどのタイプの制約とトリガーをテーブルから削除する必要があります。詳細については、[ダイレクトパスロード](#) を参照してください。

最後に、EDB*Loader は、パラレルダイレクトパスロードをサポートしています。パラレルダイレクトパスロードでは、複数の EDB* Loader セッションを同時に実行して単一のテーブルをロードできるため、パフォーマンスがさらに向上します。詳細については、[ParallelDirectPathLoad](#) を参照してください。

2.2 General Usage

EDB*Loader は、シングルスバイトまたはマルチバイトの文字セットで、区切り文字で区切られたフィールドまたは固定幅のフィールドを持つデータファイルをロードできます。区切り文字には、1 つ以上のシングルスバイト文字またはマルチバイト文字で構成される文字列を指定できます。データファイルのエンコードとデータベースのエンコードは異なる場合があります。データファイルのデータベースエンコーディングへの文字セット変換がサポートされています。

各 EDB*Loader セッションは、単一の独立したトランザクションとして実行されます。トランザクションを中止する EDB*Loader セッション中にエラーが発生した場合、セッション中に行われたすべての変更はロールバックされます。

一般に、データファイルのフォーマットエラーは、トランザクションの中断にはつながりません。代わりに、不正な形式のレコードは*badfile*と呼ばれるテキストファイルに書き込まれます。エラーの理由は、ログファイルに記録されます。

データベースの整合性エラーの原因となるレコードは、トランザクションの中止とロールバックをもたらします。フォーマットエラーと同様に、エラーの原因となったレコードは不良ファイルに書き込まれ、その理由はログファイルに記録されます。

注釈: EDB*Loader は、データベースの整合性エラーにより EDB*Loader でロールバックが発生するという点で OracleSQL*Loader と異なります。OracleSQL*Loader では、エラーの原因となったレコードのみが拒否されます。以前にテーブルに挿入されたレコードは保持され、拒否されたレコードの後にロードが続行されます。

以下は、トランザクションを中断させないフォーマットエラーのタイプの例です。

- 数値列に非数値をロードしようとした
- 数値が数値列に対して大きすぎます
- 文字列の最大長に対して文字値が長すぎます
- 不適切な形式の日付値を日付列にロードしようとした

以下は、トランザクションを中止し、EDB*Loader セッションで行われたすべての変更をロールバックするデータベースエラーのタイプの例です。

- 主キーや一意のインデックスなどの一意の制約違反
- 参照整合性制約の違反
- チェック制約の違反
- 行の挿入の結果として起動されたトリガーによってスローされたエラー

2.3 Building the EDB*Loader Control File

EDB*Loader を呼び出すとき、提供される引数のリストには制御ファイルの名前が含まれている必要があります。制御ファイルには、EDB* Loader が入力データファイルからテーブルをロードするために使用する命令が含まれています。制御ファイルには、次のような情報が含まれます。

- ロードするデータを含む入力データファイルの名前。
- データファイルからロードするテーブルの名前。
- 1 つまたは複数のテーブル内の列の名前と、データファイル内の対応するフィールドの配置。
- データファイルがフィールドを区切るために区切り文字列を使用するかどうか、またはフィールドが固定列位置を占有するかどうかの指定。
- データファイルからどのレコードを特定のテーブルにロードするかを選択するオプションの選択基準。
- 不正にフォーマットされたレコードを収集するファイルの名前。
- テーブルの選択基準を満たさないレコードを収集する廃棄ファイルの名前。

EDB*Loader 制御ファイルの構文は次のとおりです。

```
[ OPTIONS (<param=value> [, <param=value> ] ...) ]
LOAD DATA
  [ CHARACTERSET <charset> ]
  [ INFILE '{ <data_file> | <stdin> }' ]
  [ BADFILE '<bad_file>' ]
  [ DISCARDFILE '<discard_file>' ]
  [ { DISCARDMAX | DISCARDS } <max_discard_recs> ]
[ INSERT | APPEND | REPLACE | TRUNCATE ]
[ PRESERVE BLANKS ]
{ INTO TABLE <target_table>
  [ WHEN <field_condition> [ AND <field_condition> ] ... ]
  [ FIELDS TERMINATED BY '<termstring>'
    [ OPTIONALLY ENCLOSED BY '<enclstring>' ] ]
[ RECORDS DELIMITED BY '<delimstring>' ]
[ TRAILING NULLCOLS ]
  (<field_def> [, <field_def> ] ...)
} ...
```

field_def は、target_table の column_name に挿入されるデータの場所、データ形式、または値を記述する指定された data_file のフィールドを定義します。field_def の構文は次のとおりです。

```
<column_name> {
  CONSTANT <val> |
  FILLER [ POSITION (<start:end>) ] [ <fieldtype> ] |
  BOUNDFILLER [ POSITION (<start:end>) ] [ <fieldtype> ] |
```

(次のページに続く)

(前のページからの続き)

```
[ POSITION (<start:end>) ] [ <fieldtype> ]
[ NULLIF <field_condition> [ AND <field_condition> ] ...]
[ PRESERVE BLANKS ] [ "<expr>" ]
}
```

ここで、fieldtype は次のいずれかです。

```
CHAR [(<length>)] | DATE [(<length>)] [ "<datemask>" ] |
INTEGER EXTERNAL [(<length>)] |
FLOAT EXTERNAL [(<length>)] | DECIMAL EXTERNAL [(length)] |
ZONED EXTERNAL [(<length>)] | ZONED [(<precision> [,<scale>])]
```

Description

data_file、bad_file、および discard_file の指定には、完全なディレクトリパスまたはファイル名への相対ディレクトリパスが含まれる場合があります。ファイル名が単独で、または相対ディレクトリパスで指定されている場合、ファイルは存在するものと見なされるか (data_file の場合)、または作成されます (bad_file または discard_file の場合) edbldr が呼び出される現在の作業ディレクトリ。

ディレクトリパスやファイル名を参照するときに、EDB*Loader 制御ファイル内に環境変数への参照を含めることができます。環境変数の参照のフォーマットは、Windows システムと Linux システムで異なります。

- Linux では、形式は \$ENV_VARIABLE または \${ENV_VARIABLE} です
- Windows では、形式は %ENV_VARIABLE% です

ENV_VARIABLE は、ディレクトリパスやファイル名に設定された環境変数です。

EDBLDR_ENV_STYLE 環境変数は、EDB*Loader が存在するオペレーティングシステムに関係なく、環境変数参照を Windows スタイルの参照または Linux スタイルの参照として解釈するように AdvancedServer に指示します。この環境変数を使用して、EDB* Loader 用の移植可能な制御ファイルを作成できます。

- Windows システムでは、EDBLDR_ENV_STYLE を linux または unix に設定して、AdvancedServer に制御ファイル内の Linux スタイルの参照を認識させるようにします。
- Linux システムでは、Windows に EDBLDR_ENV_STYLE を設定して、AdvancedServer が制御ファイル内の Windows スタイルの参照を認識するように指示します。

オペレーティングシステムアカウント enterprisedb には、data_file で指定されたディレクトリとファイルの読み取り権限が必要です。

オペレーティングシステムアカウント enterprisedb には、bad_file および discard_file が書き込まれるディレクトリに対する書き込み権限が必要です。

注釈: data_file、bad_file、discard_file のファイル名には、それぞれ .dat、.bad、.dsc の拡張子を含める必要があります。指定されたファイル名に拡張子が含まれていない場合、EDB*Loader は、実際の

ファイル名に前述の適切な拡張子が含まれていると想定します。

EDB*Loader セッションでデータ形式エラーが発生し、BADFILE 句が指定されておらず、edbldr が呼び出されたときにコマンドラインで BAD パラメーターが指定されていない場合、control_file_base.bad という名前の不正なファイルが作成されます edbldr が呼び出される現在の作業ディレクトリ。control_file_base は、edbldr セッションで使用される制御ファイルのベース名（つまり、拡張子なしのファイル名）です。

次のすべての条件に該当する場合、EDB*Loader セッションでレコードが破棄されても、破棄ファイルは作成されません。

- 廃棄ファイルを指定するための DISCARDFILE 句は、制御ファイルに含まれていません。
- 廃棄ファイルを指定するための DISCARD パラメータは、コマンドラインに含まれていません。
- 破棄されるレコードの最大数を指定するための DISCARDMAX 句は、制御ファイルには含まれません。
- 破棄されるレコードの最大数を指定するための DISCARDS 句は、制御ファイルには含まれません。
- 破棄されたレコードの最大数を指定するための DISCARDMAX パラメーターは、コマンドラインに含まれていません。

破棄ファイル名を明示的に指定するための DISCARDFILE 節も DISCARD パラメーターも指定されておらず、DISCARDMAX または DISCARDS が指定されている場合、EDB*Loader セッションは、データファイル名を使用して破棄ファイルを作成します .dsc の拡張子。

注釈: キーワード DISCARD と DISCARDS には区別があります。DISCARD は、廃棄ファイル名を指定するために使用される EDB*Loader コマンドラインパラメーターです（参照: 一般的な使用法<general_usage>。キーワード ``DISCARDS`` と DISCARDMAX は、EDB*Loader セッションを終了する前に許可される破棄されたレコードの最大数を指定する同じ機能を提供します。」または DISCARDMAX の設定はデータベースに保持され、ロールバックされません。

INSERT、APPEND、REPLACE、または TRUNCATE のいずれかを指定すると、ターゲットテーブルに行を追加する方法のデフォルトアクションが確立されます。省略した場合、デフォルトのアクションは INSERT が指定された場合と同じです。

FIELDS TERMINATED BY 節が指定されている場合、field_def に対して POSITION (start:end) 節は指定できません。あるいは、FIELDS TERMINATED BY 句が指定されていない場合、すべての field_def には POSITION (start:end) 句、fieldtype(length) 句、または CONSTANT 句のいずれかが含まれている必要があります。

Parameters

OPTIONS param=value

OPTIONS 句を使用して、EDB*Loader ディレクティブを表す param=value ペアを指定します。edbldr が呼び出されたときに OPTIONS 節とコマンドラインの両方でパラメーターが指定されている場合、コマンドライン設定が使用されます。

以下のパラメーター/値ペアの 1 つ以上を指定します。

DIRECT= { FALSE | TRUE }

DIRECT が TRUE に設定されている場合、EDB*Loader は従来のパスワードの代わりに直接パスワードを実行します。DIRECT のデフォルト値は FALSE です。

レプリケートされたテーブルにデータをロードするときに DIRECT=true を設定しないでください。EDB*Loader を使用してレプリケートされたテーブルにデータをロードし、DIRECT=true を設定すると、インデックスはテーブル内の行を省略したり、削除された行への参照を含む可能性があります。EnterpriseDB は、複製されたテーブルにデータをロードするための直接挿入をサポートしていません。

ダイレクトパスワードの詳細については、[ダイレクトパスワード](#) を参照してください。

ERRORS=error_count

error_count は、EDB*Loader セッションを中止する前に許可されるエラーの数を指定します。デフォルトは 50 です。

FREEZE= { FALSE | TRUE }

FREEZE を TRUE に設定して、データを frozen 行でコピーすることを示します。現在および将来のすべてのトランザクションから見えることが保証されているタプルは、トランザクション ID のラップアラウンドを防ぐために凍結としてマークされます。凍結タプルの詳細については、次の PostgreSQL コードキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/routine-vacuuming.html>

FREEZE オプションを使用する場合、制御ファイルで TRUNCATE のデータ読み込みタイプを指定する必要があります。FREEZE は直接読み込みではサポートされていません。

デフォルトでは、FREEZE is FALSE です。

PARALLEL= { FALSE | TRUE }

PARALLEL を TRUE に設定して、この EDB*Loader セッションが、パラレルダイレクトパスワードに参加している多数の同時 EDB* Loader セッションの 1 つであることを示します。PARALLEL のデフォルト値は FALSE です。

PARALLEL が TRUE の場合、DIRECT パラメーターも TRUE に設定する必要があります。パラレルダイレクトパスワードの詳細については、[パラレルダイレクトパスワード](#) を参照してください。

ROWS=n

n は、EDB*Loader が次の n 行のセットをロードする前にコミットする行数を指定します。

ロード中に (ROWS パラメーターが指定されている) EDB*Loader が無効な行を検出した場合、エラーが発生する前にコミットされた行は宛先テーブルに残ります。

SKIP=skip_count

skip_count は、ロードを開始する前にスキップする必要がある入力データファイルの先頭のレコード数を指定します。デフォルトは 0 です。

SKIP_INDEX_MAINTENANCE={ FALSE | TRUE }

SKIP_INDEX_MAINTENANCE が TRUE の場合、インデックスのメンテナンスはダイレクトパスロードの一部として実行されず、ロードされたテーブルのインデックスは無効としてマークされます。SKIP_INDEX_MAINTENANCE のデフォルト値は FALSE です。

注: パラレルダイレクトパスのロード中、ターゲットテーブルのインデックスは更新されず、ロードの完了後に無効としてマークされます。

REINDEX コマンドを使用して、インデックスを再構築できます。REINDEX コマンドの詳細については、次の PostgreSQL コードキュメントを参照してください。

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

charset

CHARACTERSET 句を使用して、data_file の文字セットエンコーディングを識別します。charset は文字セット名です。データファイルのエンコーディングが制御ファイルのエンコーディングと異なる場合、この句が必要です。(制御ファイルのエンコーディングは、edbldr が呼び出されるクライアントのエンコーディングに常に含まれている必要があります。)

charset 設定の例は、UTF8、SQL_ASCII、および SJIS です。

クライアントからデータベースへの文字セット変換の詳細については、次の URL で入手可能な 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/multibyte.html>

data_file

target_table にロードされるデータを含むファイル。データファイルの各レコードは、target_table に挿入される行に対応しています。

ファイル名に拡張子が指定されていない場合、EDB*Loader は、ファイルの拡張子が .dat、たとえば mydatafile.dat であると想定します。

注釈: edbldr が呼び出されたときにコマンドラインで DATA パラメーターが指定されている場合、コマ

ンドライン DATA パラメーターで指定されたファイルが代わりに使用されます。

INFILE 節とコマンドラインの DATA パラメーターが省略されている場合、データファイル名は制御ファイル名と同一であると見なされますが、拡張子は .dat です。

stdin

標準入力を使用して、データを直接 EDB*Loader にロードする場合は、stdin（すべて小文字）を指定します。これは、ロードする大量のレコードを生成するデータソースに役立ちます。

bad_file

エラーのためにロードできない data_file レコードを受け取るファイル。不良ファイルは、拒否または不良レコードを収集するために生成されます。

AdvancedServer バージョン 12 以降では、不良または拒否されたレコードがある場合にのみ不良ファイルが生成されます。ただし、名前と場所が同じ既存の不良ファイルがあり、edblldr の新しいバージョンを呼び出した後に不良レコードが生成されない場合、既存の不良ファイルは変更されません。

ファイル名に拡張子が指定されていない場合、EDB*Loader は、ファイルの拡張子が .bad、たとえば mybadfile.bad であると想定します。

注釈: edblldr が呼び出されたときにコマンドラインで BAD パラメーターが指定されている場合、コマンドライン BAD パラメーターで指定されたファイルが代わりに使用されます。

discard_file

WHEN 句のあるテーブルでは選択基準が満たされず、WHEN 句のないテーブルがないため、どのテーブルにもロードされない入力データレコードを受信するファイル。（すべてのレコードは、WHEN 句のないテーブルの選択基準を満たします。）

ファイル名に拡張子が指定されていない場合、EDB*Loader は、ファイルの拡張子が .dsc、たとえば mydiscardfile.dsc であると想定します。

注釈: edblldr が呼び出されたときにコマンドラインで DISCARD パラメーターが指定されている場合、コマンドライン DISCARD パラメーターで指定されたファイルが代わりに使用されます。

```
{ DISCARDMAX | DISCARDS } max_discard_recs
```

EDB*Loader セッションを終了する前に入力データレコードから検出される可能性がある破棄されたレコードの最大数。（破棄されたレコードは、前述の discard_file パラメーターの説明で説明されています。）max_discard_recs で指定された整数値の前に、キーワード DISCARDMAX または DISCARDS を使用できます。

たとえば、`max_discard_recs` が 0 の場合、最初に破棄されたレコードが検出されると、EDB*Loader セッションは終了します。`max_discard_recs` が 1 の場合、2 番目の破棄されたレコードが検出された場合、EDB* Loader セッションは終了します。

`max_discard_recs` を超えて EDB*Loader セッションが終了すると、データベースにロードされた以前の入力データレコードが保持されます。それらはロールバックされません。

INSERT | APPEND | REPLACE | TRUNCATE

データをターゲット表にロードする方法を指定します。INSERT、APPEND、REPLACE、または TRUNCATE のいずれかが指定されている場合、すべてのテーブルのデフォルトアクションが確立され、デフォルトの INSERT が上書きされます。

INSERT

データは空のテーブルにロードされます。テーブルが最初に空でない場合、EDB*Loader は例外をスローし、データをロードしません。

注釈: テーブルに行が含まれる場合、EDB*Loader を呼び出す前に TRUNCATE コマンドを使用してテーブルを空にする必要があります。TRUNCATE コマンドの代わりに DELETE コマンドを使用してテーブルを空にすると、EDB*Loader は例外をスローします。OracleSQL* Loader では、DELETE または TRUNCATE コマンドを使用して表を空にできます。

APPEND

データは、テーブル内の既存の行に追加されます。テーブルも最初は空の場合があります。

REPLACE

REPLACE キーワードと TRUNCATE キーワードは機能的に同じです。テーブルは、新しいデータをロードする前に EDB*Loader によって切り捨てられます。

注釈: REPLACE 操作の結果として、テーブルの削除トリガーは起動されません。

TRUNCATE

テーブルは、新しいデータをロードする前に EDB*Loader によって切り捨てられます。TRUNCATE 操作の結果として、テーブルの削除トリガーは起動されません。

PRESERVE BLANKS

すべてのターゲットテーブルについて、オプションのエンクロージャー区切り文字が存在しない場合は先頭の空白を保持し、フィールドが所定のサイズで指定されている場合は末尾の空白をそのまま残します。省略した場合、デフォルトの動作では、先頭と末尾の空白が削除されます。

target_table

データがロードされるテーブルの名前。テーブル名はスキーマ修飾されている場合があります（たとえば、enterprisedb.emp）。指定されたターゲットはビューであってはなりません。

field_condition

次の形式をとる条件句:

```
[ ( [ ] { (start:end) | column_name } { = | != | <> } 'val' [ ] ) ]
```

この条件節は、INTO TABLE target_table 節の一部である WHEN 節、および構文図で field_def として示されるフィールド定義の一部である NULLIF 節に使用されます。

start と end は、data_file の列位置を指定する正の整数で、定数 val と比較されるフィールドの開始と終了を示します。各レコードの最初の文字は、start 値の 1 で始まります。

column_name は、構文図の field_def で定義されているデータファイルのフィールド定義に割り当てられた名前を指定します。

(start:end) または column_name のいずれかを使用すると、data_file のレコードの一部が「val」で指定された値と比較され、true または false として評価されます。

field_condition テキスト（特に val 文字列）で使用されるすべての文字は、データベースエンコーディングで有効でなければなりません。（データ変換を実行するために、EDB*Loader は最初に val 文字列の文字をデータベースエンコーディングに変換し、次にデータファイルエンコーディングに変換します。）

WHEN field_condition [AND field_condition] 句では、特定のレコードについてそのような条件がすべて TRUE と評価されると、EDB*Loader はそのレコードを target_table に挿入しようとします。挿入操作が失敗すると、レコードは bad_file に書き込まれます。

特定のレコードについて、すべての INTO TABLE 句に対して WHEN 句が TRUE に評価されない場合、EDB*Loader セッションに廃棄ファイルが指定されていると、レコードは discard_file に書き込まれます。

この句に対する field_condition の効果については、このパラメータリストの NULLIF 句の説明を参照してください。

termstring

data_file の各フィールドを区切る 1 つ以上の文字列。データベースのエンコーディングで有効な文字であれば、シングルスバイトでもマルチバイトでもかまいません。介在する文字がない termstring が 2 回連続して出現すると、対応する列が null に設定されます。

enclstring

フィールド値を data_file で囲むために使用される 1 つ以上の文字の文字列。データベースのエンコーディングで有効な文字であれば、シングルスバイトでもマルチバイトでもかまいません。データの一部として termstring が表示されるフィールドでは enclstring を使用します。

delimstring

data_file の各レコードを区切る 1 つ以上の文字の文字列。データベースのエンコーディングで有効な文字であれば、シングルスバイトでもマルチバイトでもかまいません。介在する文字がない delimstring が 2 回連続して出現すると、対応する行がテーブルにロードされません。最後のレコード（つまり、データファイルの最後）も delimstring 文字で終了する必要があります。そうしないと、最終レコードがテーブルにロードされません。

注釈: RECORDS DELIMITED BY 'delimstring' 句は、Oracle データベースと互換性がありません。

TRAILING NULLCOLS

TRAILING NULLCOLS が指定されている場合、特定のレコードの data_file にデータがない列リストの列は、行が挿入されるときに null に設定されます。これは、列リストの最後にある 1 つ以上の連続した列にのみ適用されます。

レコードの最後でフィールドが省略され、TRAILING NULLCOLS が指定されていない場合、EDB*Loader はレコードにフォーマットエラーが含まれていると想定し、それを不良ファイルに書き込みます。

column_name

field_def で定義されたフィールド値が挿入される target_table の列の名前。フィールド定義に FILLER または BOUNDFILLER 句が含まれている場合、column_name はテーブルの列の名前である必要はありません。FILLER および BOUNDFILLER 句によりフィールドデータをテーブル列にロードできないため、任意の識別子名を使用できます。

CONSTANT val

フィールド定義で割り当てられている列データ型と型互換性のある定数を指定します。一重引用符または二重引用符で val を囲むことができます。val に空白が含まれる場合は、引用符を囲む必要があります。

CONSTANT 句を使用すると、挿入された各行の列に割り当てられる値が完全に決まります。同じフィールド定義に他の句を含めることはできません。

TERMINATED BY 句を使用して data_file のフィールドを区切る場合、CONSTANT 句のフィールド定義に対応する data_file に区切られたフィールドがあってはなりません。言い換えると、EDB*Loader は、CONSTANT 句を含むフィールド定義の data_file にフィールドがないことを前提としています。

FILLER

フィールド定義の識別子がテーブル内の実際の列名である場合、フィールド定義によって定義されたフィールドのデータを関連する列にロードしないことを指定します。そのような場合、列はヌルに設定されます。FILLER または BOUNDFILLER 句の使用は、フィールド定義を実際の列名で識別する必要がある唯一の状況です。

BOUNDFILLER 句とは異なり、FILLER 句で定義された識別子は SQL 式で参照しないでください。expr パラメータの説明を参照してください。

BOUNDFILLER

フィールド定義の識別子がテーブル内の実際の列名である場合、フィールド定義によって定義されたフィールドのデータを関連する列にロードしないことを指定します。そのような場合、列はヌルに設定されます。FILLER または BOUNDFILLER 句の使用は、フィールド定義を実際の列名で識別する必要がある唯一の状況です。

FILLER 句とは異なり、BOUNDFILLER 句で定義された識別子は、SQL 式で参照できます。expr パラメータの説明を参照してください。

POSITION (start:end)

固定幅フィールドデータファイルのレコード内のフィールドの場所を定義します。start と end は正の整数です。レコードの最初の文字の開始値は 1 です。

CHAR [(length)] | DATE [(length)] [/ "<datemask>"] |

INTEGER EXTERNAL [(length)] |

FLOAT EXTERNAL [(length)] | DECIMAL EXTERNAL [(length)] |

ZONED EXTERNAL [(length)] | ZONED [(precision>[,<scale>])]

data_file のデータフィールドの形式を記述するフィールドタイプ。

注釈: フィールドタイプの指定はオプションで（説明目的のみ）、EDB*Loader がフィールドのデータをテーブル列に正常に挿入するかどうかには影響しません。正常なロードは、列のデータ型とフィールド値の互換性に依存します。たとえば、データ型が NUMBER(7,2) の列は 2600 を含むフィールドを正常に受け入れますが、フィールドに 26XX などの値が含まれる場合、挿入は失敗し、レコードは bad_file に書き込まれます。

ZONED データは人間が読み取れないことに注意してください。ZONED データは、各桁が個別のニブル/ニブル/4 ビットフィールドにエンコードされる内部形式で保存されます。各 ZONED 値の最後のバイトには、1 桁（上位 4 ビット）と符号（下位 4 ビット）が含まれます。

length

関連付けられた列にロードされる値の長さを指定します。

fieldtype(length) 節とともに「11」（「12」:「13」）節が指定されている場合、フィールドの終了位置は指定された「14」の値で上書きされます。つまり、列にロードされる値の長さは、「11」（「12」:「13」の「13」位置ではなく、「12」位置から始まる「14」値によって決まります。）条項。したがって、列にロード

される値は、「11」（「12」：「13」）で定義されたフィールドよりも短い場合があります。または、指定された「14」サイズによって*end* 位置を超える場合があります。

FIELDS TERMINATED BY 'termstring' 節が INTO TABLE 節の一部として指定され、フィールド定義に fieldtype(length) 節が含まれている場合、指定された length 値が以上である限り、レコードは受け入れられますレコードのすべてのそのようなフィールドを囲む termstring 文字によって決定されるフィールド長まで。指定された length 値が、そのようなフィールドの termstring 文字を囲むことによって決定されるフィールド長よりも短い場合、レコードは拒否されます。

FIELDS TERMINATED BY 'termstring' 句が指定されておらず、POSITION（「10」：「11」）句が「12」句を含むフィールドに含まれていない場合、このフィールドの開始位置は前のフィールドの終了位置に続く次の文字。前のフィールドの終了位置は、前のフィールドに「12」節が含まれている場合は length 値の終わり、フィールドに POSITION（「10」：「11」）が含まれている場合は「11」パラメーター）「12」句のない句。

precision

precision を使用して、ZONED 値の長さを指定します。

ZONED に指定された precision 値が POSITION 句で提供された情報に基づいてサーバーによって計算された長さと競合する場合、EDB*Loader は precision に指定された値を使用します。

scale

scale は、ZONED 値の小数点の右側の桁数を指定します。

datemask

日付フィールドの日、月、および年コンポーネントの順序と略語を指定します。

注釈: DATE フィールドタイプが列の SQL 式とともに指定されている場合、DATE の後、SQL 式の前に datemask を指定する必要があります。expr パラメータの次の説明を参照してください。

NULLIF field_condition [AND field_condition] ...

field_condition の構文については、このパラメーターセクションで前述した field_condition の説明を参照してください。

すべてのフィールド条件が TRUE と評価される場合、フィールド定義で column_name で識別される列はヌルに設定されます。フィールド条件が FALSE と評価されると、フィールド定義に従って通常発生する適切な値に列が設定されます。

PRESERVE BLANKS

このオプションが表示される列では、オプションのエンクロージャー区切り文字が存在しない場合は先頭の空白を保持し、フィールドが所定のサイズで指定されている場合は末尾の空白をそのまま残します。省略し

た場合、デフォルトの動作では、先頭と末尾の空白が削除されます。

`expr`

フィールド定義で割り当てられている列データ型と型互換性のあるスカラー値を返す **SQL** 式。二重引用符で `expr` を囲む必要があります。 `expr` には、列名の前にコロン文字 (:) を付けることにより、フィールドリスト内の任意の列への参照を含めることができます (`FILLER` 句を持つフィールドを除く)。

`expr` は、**SQL** の `SELECT` ステートメントで構成することもできます。 `SELECT` ステートメントを使用する場合、次の規則を適用する必要があります。

- `SELECT` ステートメントは、括弧 (`SELECT ...`) で囲む必要があります。
- 選択リストは、 `SELECT` キーワードに続く正確に 1 つの式で構成する必要があります。
- 結果セットは複数の行を返してはなりません。行が返されない場合、結果の式の戻り値は `null` です。

以下は、 `SELECT` ステートメントを使用するための構文です。

```
/"(SELECT expr [ FROM table_list [ WHERE condition ] ])/"
```

注釈: `FROM table_list` 句を省略すると、Oracle データベースとの互換性がなくなります。テーブルを指定する必要がない場合、`FROM DUAL` 句の使用は Oracle データベースと互換性があります。

2.3.1 EDB Loader Control File Examples

以下は、制御ファイルとそれに対応するデータファイルの例です。

Delimiter-Separated Field Data File

次の制御ファイルは、emp テーブルに行を追加する区切り文字で区切られたデータファイルを使用します。

```
LOAD DATA
  INFILE 'emp.dat'
  BADFILE 'emp.bad'
  APPEND
  INTO TABLE emp
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  TRAILING NULLCOLS
(
  empno,
  ename,
  job,
  mgr,
  hiredate,
  sal,
  deptno,
  comm
)
```

上記の制御ファイルでは、APPEND 句を使用して、emp テーブルに追加の行を挿入できます。

以下は、対応する区切り文字で区切られたデータファイルです。

```
9101,ROGERS,CLERK,7902,17-DEC-10,1980.00,20
9102,PETERSON,SALESMAN,7698,20-DEC-10,2600.00,30,2300.00
9103,WARREN,SALESMAN,7698,22-DEC-10,5250.00,30,2500.00
9104,"JONES, JR.",MANAGER,7839,02-APR-09,7975.00,20
```

TRAILING NULLCOLS 句を使用すると、comm 列を提供する最後のフィールドを最初と最後のレコードから省略できます。これらのレコードから挿入された行の comm 列は null に設定されます。

コンマ区切り文字はフィールド値の一部であるため、二重引用符の囲み文字は最後のレコードの値 JONES, JR. を囲みます。

次のクエリは、EDB*Loader セッションの後にテーブルに追加された行を表示します。

```
SELECT * FROM emp WHERE empno > 9100;

 empno| ename      | job      | mgr|   hiredate      | sal  | comm  | deptno
-----+-----+-----+---+-----+-----+-----+-----
  9101| ROGERS     | CLERK    | 7902| 17-DEC-10 00:00:00|1980.00|      |    20
```

(次のページに続く)

(前のページからの続き)

9102	PETERSON		SALESMAN	7698	20-DEC-10	00:00:00	2600.00	2300.00	30
9103	WARREN		SALESMAN	7698	22-DEC-10	00:00:00	5250.00	2500.00	30
9104	JONES, JR.		MANAGER	7839	02-APR-09	00:00:00	7975.00		20

(4 rows)

Fixed-Width Field Data File

次の例は、emp テーブルに同じ行をロードするが、固定幅フィールドを含むデータファイルを使用する制御ファイルです。

```
LOAD DATA
  INFILE 'emp_fixed.dat'
  BADFILE 'emp_fixed.bad'
  APPEND
  INTO TABLE emp
  TRAILING NULLCOLS
  (
    empno POSITION (1:4),
    ename POSITION (5:14),
    job POSITION (15:23),
    mgr POSITION (24:27),
    hiredate POSITION (28:38),
    sal POSITION (39:46),
    deptno POSITION (47:48),
    comm POSITION (49:56)
  )
```

上記の制御ファイルでは、FIELDS TERMINATED BY および OPTIONALLY ENCLOSED BY 句はありません。代わりに、各フィールドに POSITION 句が含まれるようになりました。

以下は、固定幅フィールドを含む対応するデータファイルです。

9101	ROGERS	CLERK	7902	17-DEC-10	1980.0020
9102	PETERSON	SALESMAN	7698	20-DEC-10	2600.0030 2300.00
9103	WARREN	SALESMAN	7698	22-DEC-10	5250.0030 2500.00
9104	JONES, JR.	MANAGER	7839	02-APR-09	7975.0020

Single Physical Record Data File – RECORDS DELIMITED BY Clause

次の例は、同じ行を emp テーブルにロードするが、1つの物理レコードを持つデータファイルを使用する制御ファイルです。表の行としてロードされる個々のレコードは、RECORDS DELIMITED BY 句で指定されたセミコロ文字 (;) で終了します。

```
LOAD DATA
  INFILE 'emp_recdelim.dat'
  BADFILE 'emp_recdelim.bad'
```

(次のページに続く)

(前のページからの続き)

```

APPEND
INTO TABLE emp
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  RECORDS DELIMITED BY ';'
  TRAILING NULLCOLS
(
  empno,
  ename,
  job,
  mgr,
  hiredate,
  sal,
  deptno,
  comm
)

```

以下は、対応するデータファイルです。コンテンツは、データファイル内の単一の物理レコードです。レコード区切り文字は、最後のレコードの後に（つまり、ファイルの最後に）含まれます。

```

9101,ROGERS,CLERK,7902,17-DEC-10,1980.00,20,;9102,PETERSON,SALESMAN,7698,20-
DEC-10,2600.00,30,2300.00;9103,WARREN,SALESMAN,7698,22-DEC-
10,5250.00,30,2500.00;9104,"JONES, JR.",MANAGER,7839,02-APR-09,7975.00,20,;

```

FILLER Clause

次の制御ファイルは、sal 列と comm 列のデータフィールドでの FILLER 句の使用を示しています。EDB*Loader はこれらのフィールドの値を無視し、対応する列を null に設定します。

```

LOAD DATA
  INFILE      'emp_fixed.dat'
  BADFILE     'emp_fixed.bad'
APPEND
INTO TABLE emp
  TRAILING NULLCOLS
(
  empno      POSITION (1:4),
  ename      POSITION (5:14),
  job        POSITION (15:23),
  mgr        POSITION (24:27),
  hiredate   POSITION (28:38),
  sal        FILLER POSITION (39:46),
  deptno     POSITION (47:48),
  comm       FILLER POSITION (49:56)
)

```

前の固定幅フィールドの例と同じ固定幅データファイルを使用すると、表の結果の行は次のようになります。

```
SELECT * FROM emp WHERE empno > 9100;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno
9101	ROGERS	CLERK	7902	17-DEC-10 00:00:00			20
9102	PETERSON	SALESMAN	7698	20-DEC-10 00:00:00			30
9103	WARREN	SALESMAN	7698	22-DEC-10 00:00:00			30
9104	JONES, JR.	MANAGER	7839	02-APR-09 00:00:00			20

(4 rows)

BOUNDFILLER Clause

次の制御ファイルは、job 列と mgr 列のデータフィールドで BOUNDFILLER 句を使用する方法を示しています。EDB*Loader はこれらのフィールドの値を無視し、FILLER 句と同じ方法で対応する列を null に設定します。ただし、FILLER 句の列とは異なり、BOUNDFILLER 句の列は、列 jobdesc に示すように式で使用できます。

```
LOAD DATA
  INFILE      'emp.dat'
  BADFILE     'emp.bad'
  APPEND
  INTO TABLE empjob
    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
    TRAILING NULLCOLS
  (
    empno,
    ename,
    job      BOUNDFILLER,
    mgr      BOUNDFILLER,
    hiredate FILLER,
    sal      FILLER,
    deptno   FILLER,
    comm     FILLER,
    jobdesc  ":job || ' for manager ' || :mgr"
  )
```

以下は、この例で使用される区切り文字で区切られたデータファイルです。

```
9101,ROGERS,CLERK,7902,17-DEC-10,1980.00,20
9102,PETERSON,SALESMAN,7698,20-DEC-10,2600.00,30,2300.00
9103,WARREN,SALESMAN,7698,22-DEC-10,5250.00,30,2500.00
9104,"JONES, JR.",MANAGER,7839,02-APR-09,7975.00,20
```

次の表は、前述の制御ファイルとデータファイルを使用してロードされます。

```
CREATE TABLE empjob (
  empno      NUMBER(4) NOT NULL CONSTRAINT empjob_pk PRIMARY KEY,
  ename      VARCHAR2(10),
```

(次のページに続く)

(前のページからの続き)

```

    job          VARCHAR2(9),
    mgr          NUMBER(4),
    jobdesc      VARCHAR2(25)
);

```

表の結果の行は次のように表示されます。

```
SELECT * FROM empjob;
```

empno	ename	job	mgr	jobdesc
9101	ROGERS			CLERK for manager 7902
9102	PETERSON			SALESMAN for manager 7698
9103	WARREN			SALESMAN for manager 7698
9104	JONES, JR.			MANAGER for manager 7839

(4 rows)

Field Types with Length Specification

次の例は、長さを指定したフィールドタイプ句を含む制御ファイルです。

```

LOAD DATA
  INFILE 'emp_fixed.dat'
  BADFILE 'emp_fixed.bad'
  APPEND
  INTO TABLE emp
  TRAILING NULLCOLS
  (
    empno      CHAR(4),
    ename      CHAR(10),
    job        POSITION (15:23) CHAR(9),
    mgr        INTEGER EXTERNAL(4),
    hiredate   DATE(11) "DD-MON-YY",
    sal        DECIMAL EXTERNAL(8),
    deptno     POSITION (47:48),
    comm       POSITION (49:56) DECIMAL EXTERNAL(8)
  )

```

注釈: POSITION 句と fieldtype(length) 句は、各フィールド定義に 2 つの句の少なくとも 1 つが含まれている限り、個別に、または組み合わせて使用できます。

以下は、固定幅フィールドを含む対応するデータファイルです。

```
9101ROGERS      CLERK      790217-DEC-10  1980.0020
```

(次のページに続く)

(前のページからの続き)

9102	PETERSON	SALESMAN	769820-DEC-10	2600.0030	2300.00
9103	WARREN	SALESMAN	769822-DEC-10	5250.0030	2500.00
9104	JONES, JR.	MANAGER	783902-APR-09	7975.0020	

表の結果の行は次のように表示されます。

```
SELECT * FROM emp WHERE empno > 9100;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno
9101	ROGERS	CLERK	7902	17-DEC-10 00:00:00	1980.00		20
9102	PETERSON	SALESMAN	7698	20-DEC-10 00:00:00	2600.00	2300.00	30
9103	WARREN	SALESMAN	7698	22-DEC-10 00:00:00	5250.00	2500.00	30
9104	JONES, JR.	MANAGER	7839	02-APR-09 00:00:00	7975.00		20

(4 rows)

NULLIF Clause

次の例では、sal カラムの NULLIF 句を使用してジョブ MANAGER の従業員に対して null を設定し、comm カラムで従業員が SALESMAN でなく部門 30。つまり、従業員が SALESMAN または部門 30 のメンバーである場合、通信値が受け入れられます。

制御ファイルは次のとおりです。

```
LOAD DATA
  INFILE      'emp_fixed_2.dat'
  BADFILE     'emp_fixed_2.bad'
  APPEND
  INTO TABLE emp
  TRAILING NULLCOLS
  (
    empno      POSITION (1:4),
    ename      POSITION (5:14),
    job        POSITION (15:23),
    mgr        POSITION (24:27),
    hiredate   POSITION (28:38),
    sal        POSITION (39:46) NULLIF job = 'MANAGER',
    deptno     POSITION (47:48),
    comm       POSITION (49:56) NULLIF job <> 'SALESMAN' AND deptno <> '30'
  )
```

以下は、対応するデータファイルです。

9101	ROGERS	CLERK	790217-DEC-10	1980.0020	
9102	PETERSON	SALESMAN	769820-DEC-10	2600.0030	2300.00
9103	WARREN	SALESMAN	769822-DEC-10	5250.0030	2500.00
9104	JONES, JR.	MANAGER	783902-APR-09	7975.0020	

(次のページに続く)

(前のページからの続き)

9105	ARNOLDS	CLERK	778213-SEP-10	3750.0030	800.00
9106	JACKSON	ANALYST	756603-JAN-11	4500.0040	2000.00
9107	MAXWELL	SALESMAN	769820-DEC-10	2600.0010	1600.00

表の結果の行は次のように表示されます。

```
SELECT empno, ename, job, NVL(TO_CHAR(sal), '--null--') "sal",
       NVL(TO_CHAR(comm), '--null--') "comm", deptno FROM emp WHERE empno > 9100;
```

empno	ename	job	sal	comm	deptno
9101	ROGERS	CLERK	1980.00	--null--	20
9102	PETERSON	SALESMAN	2600.00	2300.00	30
9103	WARREN	SALESMAN	5250.00	2500.00	30
9104	JONES, JR.	MANAGER	--null--	--null--	20
9105	ARNOLDS	CLERK	3750.00	800.00	30
9106	JACKSON	ANALYST	4500.00	--null--	40
9107	MAXWELL	SALESMAN	2600.00	1600.00	10

(7 rows)

注釈: ジョブが MANAGER であるため、従業員 JONES, JR. の sal 列はヌルです。

従業員のデータファイルの comm 値 PETERSON、WARREN、ARNOLDS、MAXWELL はすべて、emp テーブルの comm 列に読み込まれます。これらの従業員は SALESMAN または部門 30 のメンバー。

従業員 JACKSON のデータファイル内の 2000.00 の comm 値は無視され、この従業員は SALESMAN でも部門のメンバーでもないため、emp テーブルの comm 列はヌルに設定されます 30。

SELECT Statement in a Field Expression

次の例では、フィールド定義の式で SELECT ステートメントを使用して、列にロードする値を返します。

```
LOAD DATA
  INFILE      'emp_fixed.dat'
  BADFILE     'emp_fixed.bad'
  APPEND
  INTO TABLE emp
  TRAILING NULLCOLS
  (
    empno      POSITION (1:4),
    ename      POSITION (5:14),
    job        POSITION (15:23) "(SELECT dname FROM dept WHERE deptno =
    :deptno)",
    mgr        POSITION (24:27),
    hiredate   POSITION (28:38),
```

(次のページに続く)

(前のページからの続き)

```

    sal      POSITION (39:46),
    deptno   POSITION (47:48),
    comm     POSITION (49:56)
)

```

SELECT ステートメントで使用される dept テーブルの内容は次のとおりです。

```

SELECT * FROM dept;

deptno |      dname      |      loc
-----+-----+-----
    10 | ACCOUNTING     | NEW YORK
    20 | RESEARCH       | DALLAS
    30 | SALES          | CHICAGO
    40 | OPERATIONS     | BOSTON
(4 rows)

```

以下は、対応するデータファイルです。

```

9101ROGERS      CLERK      790217-DEC-10    1980.0020
9102PETERSON    SALESMAN    769820-DEC-10    2600.0030    2300.00
9103WARREN      SALESMAN    769822-DEC-10    5250.0030    2500.00
9104JONES, JR.  MANAGER    783902-APR-09    7975.0020

```

表の結果の行は次のように表示されます。

```

SELECT * FROM emp WHERE empno > 9100;

empno| ename      | job      | mgr|   hiredate      | sal   | comm   |deptno
-----+-----+-----+---+-----+-----+-----+-----
  9101| ROGERS     | RESEARCH|7902| 17-DEC-10 00:00:00|1980.00|        |   20
  9102| PETERSON   | SALES   |7698| 20-DEC-10 00:00:00|2600.00| 2300.00|   30
  9103| WARREN     | SALES   |7698| 22-DEC-10 00:00:00|5250.00| 2500.00|   30
  9104| JONES, JR. | RESEARCH|7839| 02-APR-09 00:00:00|7975.00|        |   20
(4 rows)

```

注釈: job 列には、データファイルのジョブ名の代わりに SELECT ステートメントによって返される dept テーブルの dname 列の値が含まれます。

Multiple INTO TABLE Clauses

次の例は、複数の INTO TABLE 句の使用を示しています。この例では、emp テーブルと同じデータ定義で2つの空のテーブルが作成されます。次の CREATE TABLE コマンドは、これらの2つの空のテーブルを作成し、元の emp テーブルから行を挿入しません。

```
CREATE TABLE emp_research AS SELECT * FROM emp WHERE deptno = 99;
CREATE TABLE emp_sales AS SELECT * FROM emp WHERE deptno = 99;
```

次の制御ファイルには、2つの INTO TABLE 句が含まれています。また、APPEND 句がないため、INSERT のデフォルト操作が使用されます。これには、テーブル emp_research と emp_sales が空である必要があります。

```
LOAD DATA
  INFILE      'emp_multitbl.dat'
  BADFILE     'emp_multitbl.bad'
  DISCARDFILE 'emp_multitbl.dsc'
  INTO TABLE emp_research
    WHEN (47:48) = '20'
    TRAILING NULLCOLS
  (
    empno      POSITION (1:4),
    ename      POSITION (5:14),
    job        POSITION (15:23),
    mgr        POSITION (24:27),
    hiredate   POSITION (28:38),
    sal        POSITION (39:46),
    deptno     CONSTANT '20',
    comm       POSITION (49:56)
  )
  INTO TABLE emp_sales
    WHEN (47:48) = '30'
    TRAILING NULLCOLS
  (
    empno      POSITION (1:4),
    ename      POSITION (5:14),
    job        POSITION (15:23),
    mgr        POSITION (24:27),
    hiredate   POSITION (28:38),
    sal        POSITION (39:46),
    deptno     CONSTANT '30',
    comm       POSITION (49:56) "ROUND (:comm + (:sal * .25), 0)"
  )
```

WHEN 句は、列 47 から 48 で指定されたフィールドに 20 が含まれる場合、レコードが emp_research テーブルに挿入され、同じフィールドに 30 が含まれる場合、レコードが emp_sales に挿入されることを指定します。どちらの条件も当てはまらない場合、レコードは emp_multitbl.dsc という名前の廃棄ファイルに書き込まれます。

deptno 列には CONSTANT 句が指定されているため、指定した定数値が各レコードの deptno に挿入されます。CONSTANT 句を使用する場合、定数値が割り当てられる列名以外のフィールド定義内の唯一の句でなければなりません。

最後に、emp_sales テーブルの comm 列に SQL 式が割り当てられます。列名の先頭にコロン文字 (:) を付け

ることにより、式で列名を参照できます。

以下は、対応するデータファイルです。

9101	ROGERS	CLERK	790217-DEC-10	1980.0020	
9102	PETERSON	SALESMAN	769820-DEC-10	2600.0030	2300.00
9103	WARREN	SALESMAN	769822-DEC-10	5250.0030	2500.00
9104	JONES, JR.	MANAGER	783902-APR-09	7975.0020	
9105	ARNOLDS	CLERK	778213-SEP-10	3750.0010	
9106	JACKSON	ANALYST	756603-JAN-11	4500.0040	

従業員の ARNOLDS と JACKSON のレコードは、47 から 48 の列に 10 と 40 を含み、WHEN 句のいずれも満たさないため、EDB*Loader はこれら 2 つのレコードを廃棄ファイルに書き込みます。emp_multitbl.dsc。その内容は次のとおりです。

9105	ARNOLDS	CLERK	778213-SEP-10	3750.0010	
9106	JACKSON	ANALYST	756603-JAN-11	4500.0040	

以下は、emp_research および emp_sales テーブルにロードされる行です。

```
SELECT * FROM emp_research;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno
9101	ROGERS	CLERK	7902	17-DEC-10 00:00:00	1980.00		20.00
9104	JONES, JR.	MANAGER	7839	02-APR-09 00:00:00	7975.00		20.00

(2 rows)

```
SELECT * FROM emp_sales;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno
9102	PETERSON	SALESMAN	7698	20-DEC-10 00:00:00	2600.00	2950.00	30.00
9103	WARREN	SALESMAN	7698	22-DEC-10 00:00:00	5250.00	3813.00	30.00

(2 rows)

2.4 Invoking EDB*Loader

EDB*Loader を実行するには、スーパーユーザー権限が必要です。次のコマンドを使用して、コマンドラインから EDB* Loader を呼び出します。

```
edblldr [ -d <dbname> ] [ -p <port> ] [ -h <host> ]
[ USERID={ <username/password> | <username>/ | <username> | / } ]
[ { -c | connstr= } <CONNECTION_STRING> ]
  CONTROL=<control_file>
[ DATA=<data_file> ]
[ BAD=<bad_file>]
[ DISCARD=<discard_file> ]
[ DISCARDMAX=<max_discard_recs> ]
[ HANDLE_CONFLICTS={ FALSE | TRUE } ]
[ LOG=<log_file> ]
[ PARFILE=<param_file> ]
[ DIRECT={ FALSE | TRUE } ]
[ FREEZE={ FALSE | TRUE } ]
[ ERRORS=<error_count> ]
[ PARALLEL={ FALSE | TRUE } ]
[ ROWS=<n> ]
[ SKIP=<skip_count> ]
[ SKIP_INDEX_MAINTENANCE={ FALSE | TRUE } ]
[ edb_resource_group=<group_name> ]
```

Description

-d オプション、-p オプション、または -h オプションを省略すると、データベース、ポート、およびホストのデフォルトは、edb-psql などの他の AdvancedServer ユーティリティプログラムと同じルールに従って決定されます、たとえば。

-d オプション、-p オプション、-h オプション、および PARFILE パラメーターを除く、前述の構文図にリストされているパラメーターは、パラメーターファイル*で指定できます。パラメータファイルは、`PARFILE=param_file` を使用して `edblldr` を呼び出すときにコマンドラインで指定します。一部のパラメーターは、制御ファイルの `OPTIONS` 句で指定できます。制御ファイルの詳細については、[:ref:Building the EDB Loader Control File<building_the_edb_loader_control_file>](#) を参照してください。

control_file、data_file、bad_file、「10」、「11」、および param_file の仕様には、完全なディレクトリパスまたはファイル名への相対ディレクトリパスが含まれる場合があります。ファイル名が単独で、または相対ディレクトリパスとともに指定されている場合、ファイルは存在する (control_file、data_file、または param_file の場合)、または作成される (bad_file の場合)、「10」、または「11」。「12」の呼び出し元の現在の作業ディレクトリに関連しています。

注釈: 制御ファイルは、edblldr が呼び出されるクライアントの文字セットエンコーディングに存在する必要があります。クライアントがデータベースエンコーディングとは異なるエンコーディングである場合、edblldr を

呼び出す前に PGCLIENTENCODING 環境変数をクライアントのエンコーディングに設定する必要があります。これは、クライアントとデータベースサーバー間で文字セット変換が適切に行われるようにするために行う必要があります。

edblldr の呼び出しに使用されるオペレーティングシステムアカウントには、control_file、data_file、および param_file で指定されたディレクトリとファイルに対する読み取り権限が必要です。

オペレーティングシステムアカウント enterprisedb には、bad_file、discard_file、log_file を書き込むディレクトリへの書き込み権限が必要です。

注釈: control_file、data_file、bad_file、discard_file、log_file のファイル名には、それぞれ .ctl、.dat、.bad、.dsc、.log の拡張子を含める必要があります。指定されたファイル名に拡張子が含まれていない場合、EDB*Loader は、実際のファイル名に前述の適切な拡張子が含まれていると想定します。

Parameters

dbname

ロードするテーブルを含むデータベースの名前。

port

データベースサーバーが接続を受け入れているポート番号。

host

データベースサーバーが実行されているホストの IP アドレス。

USERID={ username/password | username/ | username | / }

EDB*Loader は username でデータベースに接続します。username はスーパーユーザーでなければなりません。password は username のパスワードです。

USERID パラメーターを省略すると、EDB*Loader は username と「10」のプロンプトを表示します。USERID=username/ が指定されている場合、EDB*Loader1) PGPASSFILE が設定されている場合は環境変数 PGPASSFILE で指定されたパスワードファイルを使用し、2) PGPASSFILE が設定されていません。USERID=username が指定されている場合、EDB*Loader は「10」を要求します。「11」を指定すると、ユーザー名としてオペレーティングシステムアカウントを使用して接続が試行されます。

注釈: AdvancedServer 接続環境変数 PGUSER および PGPASSWORD は、EDB*Loader によって無視されます。PGPASSFILE 環境変数とパスワードファイルの詳細については、PostgreSQL のコアドキュメントを参照してください。

`-c CONNECTION_STRING`

`connstr=CONNECTION_STRING`

`-c` または `connstr=` オプションを使用すると、`libpq` でサポートされるすべての接続パラメーターを指定できます。このオプションを使用すると、SSL 接続パラメーターまたは `libpq` でサポートされている他の接続パラメーターも指定できます。`-d`、`-h`、`-p` または `userid=dbuser/dbpass` などの接続オプションが個別に提供されている場合、`-c` または `connstr=` オプションで提供された値をオーバーライドする場合があります。

`CONTROL=control_file`

`control_file` は、`EDB*Loader` ディレクティブを含む制御ファイルの名前を指定します。ファイル拡張子が指定されていない場合、`.ctl` の拡張子が想定されます。

制御ファイルの詳細については、[Building the EDB*Loader Control File](#) を参照してください。

`DATA=data_file`

`data_file` は、ターゲットテーブルにロードするデータを含むファイルの名前を指定します。ファイル拡張子が指定されていない場合、`.dat` の拡張子が想定されます。コマンドラインで `data_file` を指定すると、制御ファイルで指定された `INFILE` 句が上書きされます。

`data_file` の詳細については、[EDB*Loader 制御ファイルの構築](#) を参照してください。

`BAD=bad_file`

`bad_file` は、エラーのためにロードできない入力データレコードを受信するファイルの名前を指定します。コマンドラインで `bad_file` を指定すると、制御ファイルで指定された `BADFILE` 句が上書きされます。

`bad_file` の詳細については、[EDB*Loader 制御ファイルの構築](#) を参照してください。

`DISCARD=discard_file`

`discard_file` は、テーブルの選択基準を満たさない入力データレコードを受け取るファイルの名前です。コマンドラインで `discard_file` を指定すると、制御ファイルの `DISCARDFILE` 句が上書きされます。

`discard_file` の詳細については、[EDB*Loader 制御ファイルの構築](#) を参照してください。

`DISCARDMAX=max_discard_recs`

`max_discard_recs` は、`EDB*Loader` セッションを終了する前に入力データレコードから検出される可能性がある破棄されたレコードの最大数です。コマンドラインで `max_discard_recs` を指定すると、制御ファイルの `DISCARDMAX` または `DISCARDS` 句が上書きされます。

`max_discard_recs` の詳細については、[EDB*Loader 制御ファイルの構築](#) を参照してください。

`HANDLE_CONFLICTS={ FALSE | TRUE }`

一意制約違反のためにレコードの挿入が失敗した場合、EDB*Loader は操作全体を中止します。代わりに BAD ファイルに複製レコードを移動し、HANDLE_CONFLICTS を TRUE に設定して処理を続行するように EDB* Loader に指示できます。この動作は、インデックスが存在する場合にのみ適用されます。デフォルトでは、HANDLE_CONFLICTS は FALSE に設定されています。

TRUE に設定された HANDLE_CONFLICTS は、ダイレクトパスロードではサポートされていません。直接パスロード時に TRUE に設定すると、EDB*Loader はエラーをスローします。

LOG=log_file

log_file は、EDB*Loader が EDB* Loader セッションの結果を記録するファイルの名前を指定します。

LOG パラメータを省略すると、EDB*Loader は edbldr が呼び出されるディレクトリに control_file_base.log という名前のログファイルを作成します。control_file_base は、EDB* Loader セッションで使用される制御ファイルのベース名です。オペレーティングシステムアカウント enterprisedb には、ログファイルが書き込まれるディレクトリに対する書き込み権限が必要です。

PARFILE=param_file

param_file は、EDB*Loader セッションのコマンドラインパラメーターを含むファイルの名前を指定します。-d、-p、-h オプション、および PARFILE パラメータ自体を除く、このセクションにリストされているコマンドラインパラメータは、コマンドラインではなく param_file で指定できます。

param_file で指定されたパラメーターは、PARFILE オプションの前にコマンドラインで指定された同じパラメーターをオーバーライドします。PARFILE オプションの後に表示されるコマンドラインで指定されたパラメーターは、param_file で指定された同じパラメーターをオーバーライドします。

注釈: 他の EDB*Loader ファイルとは異なり、param_file に想定されるデフォルトのファイル名や拡張子はありませんが、OracleSQL* Loader の規則では、通常 .par が拡張子として使用されますが、必須ではありません。

DIRECT= { FALSE | TRUE }

DIRECT が TRUE に設定されている場合、EDB*Loader は従来のパスロードの代わりに直接パスロードを実行します。DIRECT のデフォルト値は FALSE です。

レプリケートされたテーブルにデータをロードするときに DIRECT=true を設定しないでください。EDB*Loader を使用してレプリケートされたテーブルにデータをロードし、DIRECT=true を設定すると、インデックスはテーブル内の行を省略したり、削除された行への参照を含む可能性があります。EnterpriseDB は、複製されたテーブルにデータをロードするための直接挿入をサポートしていません。

ダイレクトパスロードの詳細については、[ダイレクトパスロード](#) を参照してください。

FREEZE= { FALSE | TRUE }

FREEZE を TRUE に設定して、データを frozen 行でコピーすることを示します。現在および将来のすべてのトランザクションから見えることが保証されているタプルは、トランザクション ID のラップアラウンドを防ぐために凍結としてマークされます。凍結タプルの詳細については、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/routine-vacuuming.html>

FREEZE オプションを使用する場合、制御ファイルで TRUNCATE のデータ読み込みタイプを指定する必要があります。FREEZE は直接読み込みではサポートされていません。

デフォルトでは、FREEZE は FALSE です。

ERRORS=error_count

error_count は、EDB*Loader セッションを中止する前に許可されるエラーの数を指定します。デフォルトは 50 です。

PARALLEL= { FALSE | TRUE }

PARALLEL を TRUE に設定して、この EDB*Loader セッションが、パラレルダイレクトパスロードに参加している多数の同時 EDB* Loader セッションの 1 つであることを示します。PARALLEL のデフォルト値は FALSE です。

PARALLEL が TRUE の場合、DIRECT パラメーターも TRUE に設定する必要があります。

パラレルダイレクトパスロードの詳細については、[パラレルダイレクトパスロード](#) を参照してください。

ROWS=n

n は、EDB*Loader が次の n 行のセットをロードする前にコミットする行数を指定します。

SKIP=skip_count

ロードを開始する前にスキップする必要がある入力データファイルの先頭のレコード数。デフォルトは 0 です。

SKIP_INDEX_MAINTENANCE= { FALSE | TRUE }

TRUE に設定すると、インデックスのメンテナンスはダイレクトパスロードの一部として実行されず、ロードされたテーブルのインデックスは無効としてマークされます。SKIP_INDEX_MAINTENANCE のデフォルト値は FALSE です。

パラレルダイレクトパスのロード中、ターゲットテーブルのインデックスは更新されず、ロードの完了後に無効としてマークされます。

REINDEX コマンドを使用して、インデックスを再構築できます。REINDEX コマンドの詳細については、次の場所にある PostgreSQL コアドキュメントを参照してください。

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

edb_resource_group=group_name

group_name は、EDB*Loader セッションが割り当てられる EDBResourceManager リソースグループの名前を指定します。

セッションに割り当てられたデフォルトのリソースグループ（たとえば、ALTER ROLE ... SET edb_resource_group コマンドでデフォルトのリソースグループを割り当てられた EDB*Loader セッションを実行しているデータベースユーザー）は、指定されたリソースグループによってオーバーライドされます。edbldr コマンドラインで指定された edb_resource_group パラメータによって。

Examples

次の例では、現在の作業ディレクトリにある emp.ctl という名前の制御ファイルを使用して EDB*Loader を呼び出し、データベース edb にテーブルをロードします。

```
$ /usr/edb/as13/bin/edbldr -d edb USERID=enterprisedb/password
CONTROL=emp.ctl
EDB*Loader: Copyright (c) 2007-2020, EnterpriseDB Corporation.

Successfully loaded (4) records
```

次の例では、EDB*Loader はコマンドラインから省略されているため、ユーザー名とパスワードの入力を求めます。さらに、不良ファイルおよびログファイルのファイルは、BAD および LOG コマンドラインパラメーターで指定されます。

```
$ /usr/edb/as13/bin/edbldr -d edb CONTROL=emp.ctl BAD=/tmp/emp.bad
LOG=/tmp/emp.log
Enter the user name : enterprisedb
Enter the password :
EDB*Loader: Copyright (c) 2007-2020, EnterpriseDB Corporation.

Successfully loaded (4) records
```

次の例では、前述の例と同じパラメーターを使用して EDB*Loader を実行しますが、現在の作業ディレクトリにあるパラメーターファイルを使用します。SKIP および ERRORS パラメータも、パラメーターファイルのデフォルトから変更されています。パラメータファイル emp.par には、次のものが含まれています。

```
CONTROL=emp.ctl
BAD=/tmp/emp.bad
LOG=/tmp/emp.log
SKIP=1
ERRORS=10
```

EDB*Loader は、次のようにパラメーターファイルで呼び出されます。

```
$ /usr/edb/as13/bin/edbldr -d edb PARFILE=emp.par
Enter the user name : enterprisedb
```

(次のページに続く)

(前のページからの続き)

```
Enter the password :  
EDB*Loader: Copyright (c) 2007-2020, EnterpriseDB Corporation.  
  
Successfully loaded (3) records
```

次の例では、現在の作業ディレクトリにある `emp.ctl` 制御ファイルを使用して `connstr=` オプションを使用して **EDB*Loader** を呼び出し、`edb` という名前のデータベースにテーブルをロードします。

```
$ /usr/edb/as13/bin/edbldr connstr=\"sslmode=verify-ca sslcompression=0  
host=127.0.0.1 dbname=edb port=5444 user=enterprisedb\" CONTROL=emp.ctl  
EDB*Loader: Copyright (c) 2007-2020, EnterpriseDB Corporation.  
  
Successfully loaded (4) records
```

2.4.1 Exit Codes

EDB*Loader が終了すると、次のコードのいずれかが返されます。

終了コード	説明
0	すべての行が正常にロードされたことを示します。
1	EDB*Loader がコマンドラインまたは構文エラーを検出したか、回復不能なエラーのためにロード操作を中止したことを示します。
2	ロードは完了したが、一部（またはすべて）の行が拒否または破棄されたことを示します。
3	EDB*Loader で致命的なエラー（OS エラーなど）が発生したことを示します。このクラスのエラーは、PostgreSQL エラーの FATAL または PANIC 重大度レベルと同等です。

2.5 Direct Path Load

直接パスロード中に、EDB*Loader はデータをデータベースページに直接書き込み、データベースページはディスクに同期されます。従来のパスロードに関連付けられた挿入処理はバイパスされるため、パフォーマンスが向上します。

挿入処理をバイパスすると、ターゲット表に存在する可能性のある制約の種類が減ります。ダイレクトパスロードのターゲットテーブルでは、次のタイプの制約が許可されています。

- 主キー
- 非ヌル制約
- インデックス（一意または非一意）

ダイレクトパスロードのターゲットテーブルの制限は次のとおりです。

- トリガーは許可されていません
- チェック制約は許可されていません
- 別のテーブルを参照するターゲットテーブルの外部キー制約は許可されていません
- ターゲット表を参照する他の表に対する外部キー制約は許可されていません
- テーブルはパーティション化しないでください
- ルールはターゲットテーブルに存在する場合がありますが、実行されません

注釈: 現在、EDB*Loader での直接パスロードは、OracleSQL*Loader よりも制限されています。前述の制限は、ほとんどの場合、OracleSQL* Loader には適用されません。ダイレクトパスロードで使用する制御ファイルには、次の制限が適用されます。

- 複数のテーブルの読み込みはサポートされていません。つまり、制御ファイルで指定できる INTO TABLE 句は 1 つだけです。
- SQL 式は、INTO TABLE 句のデータフィールド定義では使用できません。
- FREEZE オプションは、ダイレクトパスロードではサポートされていません。

ダイレクトパスロードを実行するには、次の例に示すように DIRECT=TRUE オプションを追加します。

```
$ /usr/edb/as13/bin/edbldr -d edb USERID=enterprisedb/password
CONTROL=emp.ctl DIRECT=TRUE
EDB*Loader: Copyright (c) 2007-2020, EnterpriseDB Corporation.

Successfully loaded (4) records
```

2.6 Parallel Direct Path Load

同時に実行されている 2 つ以上のセッションにロードプロセスを分散することにより、ダイレクトパスロードのパフォーマンスをさらに向上させることができます。各セッションは、同じテーブルへの直接パスロードを実行します。

同じテーブルが複数のセッションからロードされるため、各 EDB*Loader セッションが独自のデータファイルを使用し、同じレコードが 2 回以上ロードされないように、テーブルにロードする入力レコードを複数のデータファイルに分割する必要があります。

パラレルダイレクトパスロードのターゲットテーブルには、単一セッションで実行されるダイレクトパスロードと同じ制限があります。

ダイレクトパスロードのターゲットテーブルの制限は次のとおりです。

- トリガーは許可されていません
- チェック制約は許可されていません
- 別のテーブルを参照するターゲットテーブルの外部キー制約は許可されていません
- ターゲット表を参照する他の表に対する外部キー制約は許可されていません
- テーブルはパーティション化しないでください
- ルールはターゲットテーブルに存在する場合がありますが、実行されません

さらに、各 EDB*Loader セッションで使用される制御ファイルに APPEND 句を指定する必要があります。

パラレルダイレクトパスロードを実行するには、パラレルダイレクトパスロードの参加者ごとに個別のセッションで EDB*Loader を実行します。このような各 EDB*Loader セッションの呼び出しには、DIRECT=TRUE および PARALLEL=TRUE パラメーターが含まれている必要があります。

各 EDB*Loader セッションは独立したトランザクションとして実行されるため、並列セッションの 1 つが中断して変更をロールバックしても、他の並列セッションによるロードは影響を受けません。

Note: パラレルダイレクトパスロードでは、各 EDB*Loader セッションはラウンドロビン方式でターゲットテーブルに固定数のブロックを予約します。最後に割り当てられたチャンク内のブロックの一部は使用されない場合があります。それらのブロックは初期化されません。その後、ターゲットテーブルで VACUUM コマンドを使用すると、次のようなこれらの初期化されていないブロックに関する警告が表示される場合があります。

```
WARNING: relation "emp" page 98264 is uninitialized --- fixing
WARNING: relation "emp" page 98265 is uninitialized --- fixing
WARNING: relation "emp" page 98266 is uninitialized --- fixing
```

これは予想される動作であり、データの破損を示すものではありません。

ターゲットテーブルのインデックスは、パラレルダイレクトパスのロード中は更新されないため、ロードの完了後に無効としてマークされます。インデックスを再構築するには、REINDEX コマンドを使用する必要があります。

次の例は、emp テーブルでの並列ダイレクトパスロードの使用を示しています。

注釈: AdvancedServer に付属のサンプル emp テーブルでパラレルダイレクトパスロードを試みる場合は、まず emp テーブルを参照するトリガーと制約を削除する必要があります。さらに、この例では主キー列 empno が NUMBER(4) から NUMBER に拡張され、より多くの行を挿入できるようになりました。

以下は、最初のセッションで使用される制御ファイルです。

```
LOAD DATA
  INFILE      '/home/user/loader/emp_parallel_1.dat'
  APPEND
  INTO TABLE emp
    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
    TRAILING NULLCOLS
  (
    empno,
    ename,
    job,
    mgr,
    hiredate,
    sal,
    deptno,
    comm
  )
```

パラレルダイレクトパスロードの場合、制御ファイルに APPEND 句を指定する必要があります。

以下は、最初のセッションでの EDB*Loader の呼び出しを示しています。DIRECT=TRUE および PARALLEL=TRUE パラメーターを指定する必要があります。

```
$ /usr/edb/as13/bin/edbldr -d edb USERID=enterprisedb/password
CONTROL=emp_parallel_1.ctl DIRECT=TRUE PARALLEL=TRUE
WARNING: index maintenance will be skipped with PARALLEL load
EDB*Loader: Copyright (c) 2007-2020, EnterpriseDB Corporation.
```

2 番目のセッションに使用される制御ファイルは次のように表示されます。最初のセッションで使ったものと同じですが、異なるデータファイルを使用することに注意してください。

```
LOAD DATA
  INFILE      '/home/user/loader/emp_parallel_2.dat'
  APPEND
  INTO TABLE emp
```

(次のページに続く)

(前のページからの続き)

```

FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(
  empno,
  ename,
  job,
  mgr,
  hiredate,
  sal,
  deptno,
  comm
)

```

上記の制御ファイルは、次のように 2 番目のセッションで使用されます。

```

$ /usr/edb/as13/bin/edbldr -d edb USERID=enterprisedb/password
CONTROL=emp_parallel_2.ctl DIRECT=TRUE PARALLEL=TRUE
WARNING: index maintenance will be skipped with PARALLEL load
EDB*Loader: Copyright (c) 2007-2020, EnterpriseDB Corporation.

```

EDB*Loader は、それぞれのロード操作が完了すると、各セッションで次のメッセージを表示します。

```
Successfully loaded (10000) records
```

次のクエリは、**emp** テーブルのインデックスが INVALID としてマークされていることを示しています。

```

SELECT index_name, status FROM user_indexes WHERE table_name = 'EMP';

  index_name | status
-----+-----
    EMP_PK   | INVALID
(1 row)

```

注釈: `user_indexes` は、現在のユーザーが所有する Oracle データベースと互換性のあるインデックスのビューです。

emp テーブルのクエリは、次のように `REINDEX` コマンドを使用して再構築しない限り、インデックスを使用しません。

```
REINDEX INDEX emp_pk;
```

`user_indexes` に対する後続のクエリは、インデックスが VALID としてマークされていることを示しています。

```
SELECT index_name, status FROM user_indexes WHERE table_name = 'EMP';
```

index_name	status
EMP_PK	VALID

(1 row)

2.7 Remote Loading

EDB*Loader は、リモート読み込み*と呼ばれる機能をサポートしています。リモートロードでは、ロードするテーブルを含むデータベースは、入力データソースで EDB Loader が呼び出されるホストとは異なるホスト上のデータベースサーバーで実行されます。

この機能は、ロードするデータが大量にあり、データベースサーバーを実行しているホスト上に大きなデータファイルを作成したくない場合に役立ちます。

さらに、標準入力機能を使用して、別のプログラムやスクリプトなどのデータソースから EDB*Loader に直接データをパイプし、リモートデータベースにテーブルをロードできます。これにより、EDB*Loader 用のディスク上にデータファイルを作成するプロセスがバイパスされます。

標準入力を使用してリモートローディングを実行するには、以下が必要です。

- edbldr プログラムは、EDB*Loader セッションのデータソースで呼び出されるクライアントホストにインストールする必要があります。
- データを EDB*Loader の標準入力に直接パイプできるように、制御ファイルには INFILE 'stdin' 句が含まれている必要があります。詳細については、INFILE 句と EDB*Loader 制御ファイルの詳細について、[Building the EDB*Loader Control File](#) を参照してください。
- 制御ファイル、不良ファイル、破棄ファイル、ログファイルなど、EDB*Loader で使用されるすべてのファイルは、edbldr が呼び出されるクライアントホスト上に存在するか、作成される必要があります。
- EDB*Loader を呼び出す場合、-h オプションを使用して、リモートデータベースサーバーの IP アドレスを指定します。詳細については、EDB*Loader の呼び出しについては、[Invoking EDB*Loader](#) を参照してください。
- オペレーティングシステムのパイプ演算子 (|) または入力リダイレクト演算子 (<) を使用して、入力データを EDB*Loader に提供します。

次の例では、datasource という名前のソースからパイプされたデータを使用して、192.168.1.14 のデータベースサーバーで実行されているデータベースをロードします。

```
datasource | ./edbldr -d edb -h 192.168.1.14 USERID=enterprisedb/password  
CONTROL=remote.ctl
```

以下は、標準入力の使用方法の別の例です。

```
./edbldr -d edb -h 192.168.1.14 USERID=enterprisedb/password  
CONTROL=remote.ctl < datasource
```

2.8 Updating a Table with a Conventional Path Load

EDB*Loader を従来のパスロードとともに使用して、テーブル内の行を更新し、新しいデータを既存のデータとマージできます。EDB* Loader を呼び出して更新を実行すると、サーバーは、一致する主キーを持つ既存の行をテーブルで検索します。

- サーバーは、一致するキーを持つ行を見つけると、既存の行を新しい行に置き換えます。
- サーバーは、一致するキーを持つ行を見つけられない場合、新しい行をテーブルに追加します。

EDB*Loader を使用してテーブルを更新するには、テーブルに主キーが必要です。EDB* Loader を使用してパーティションテーブルを UPDATE にすることはできません。

UPDATE を実行するには、従来のパスロードを実行するときと同じ手順を使用します。

1. UPDATE または INSERT にする行を含むデータファイルを作成します。
2. INFILE キーワードを使用してデータファイルの名前を指定する制御ファイルを定義します。EDB*Loader 制御ファイルの構築については、[Building the EDB* Loader Control File](#) を参照してください。
3. データベース名、接続情報、および制御ファイルの名前を指定して、EDB*Loader を呼び出します。EDB*Loader の呼び出しについては、[Invoking EDB* Loader](#) を参照してください。

次の例では、AdvancedServer サンプルデータとともに配布される emp テーブルを使用しています。デフォルトでは、テーブルには次のものが含まれます。

```
edb=# select * from emp;
```

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)

次の制御ファイル（emp_update.ctl）は、テーブル内のフィールドをコンマ区切りリストで指定します。制御ファイルは、emp テーブルで UPDATE を実行します。

```

LOAD DATA
  INFILE 'emp_update.dat'
  BADFILE 'emp_update.bad'
  DISCARDFILE 'emp_update.dsc'
UPDATE INTO TABLE emp
FIELDS TERMINATED BY ","
(empno, ename, job, mgr, hiredate, sal, comm, deptno)

```

更新または挿入されるデータは emp_update.dat ファイルに保存されます。emp_update.dat に含まれるもの:

```

7521,WARD,MANAGER,7839,22-FEB-81 00:00:00,3000.00,0.00,30
7566,JONES,MANAGER,7839,02-APR-81 00:00:00,3500.00,0.00,20
7903,BAKER,SALESMAN,7521,10-JUN-13 00:00:00,1800.00,500.00,20
7904,MILLS,SALESMAN,7839,13-JUN-13 00:00:00,1800.00,500.00,20
7654,MARTIN,SALESMAN,7698,28-SEP-81 00:00:00,1500.00,400.00,30

```

データベースの名前 (edb)、データベースのスーパーユーザーの名前 (および関連するパスワード)、および制御ファイルの名前 (emp_update.ctl) を指定して、EDB*Loader を呼び出します。

```
edblldr -d edb userid=user_name/password control=emp_update.ctl
```

更新の実行後、emp テーブルには次が含まれます。

```

edb=# select * from emp;
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  |MANAGER  | 7839 | 22-FEB-81 00:00:00 | 3000.00 | 0.00     | 30
7566 |JONES |MANAGER  | 7839 | 02-APR-81 00:00:00 | 3500.00 | 0.00     | 20
7654 |MARTIN|SALESMAN | 7698 | 28-SEP-81 00:00:00 | 1500.00 | 400.00   | 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
7903 |BAKER |SALESMAN | 7521 | 10-JUN-13 00:00:00 | 1800.00 | 500.00   | 20
7904 |MILLS |SALESMAN | 7839 | 13-JUN-13 00:00:00 | 1800.00 | 500.00   | 20
7934 |MILLER|CLERK    | 7782 | 23-JAN-82 00:00:00 | 1300.00 |          | 10
(16 rows)

```

現在 emp テーブルにある 3 人の従業員の情報を含む行が更新され、新しい従業員の行が追加されます (BAKER および MILLS)

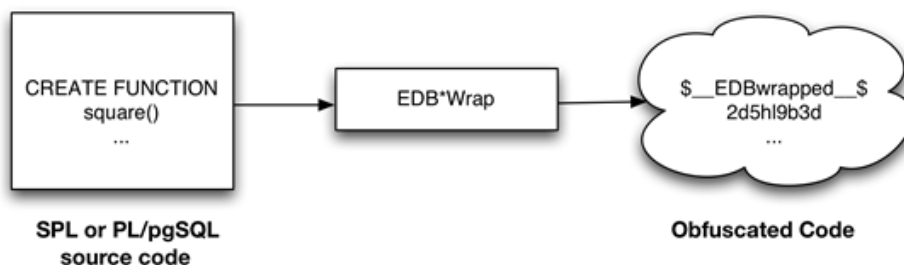
第 3 章

EDB*Wrap

EDB*Wrap ユーティリティは、独自のソースコードとプログラム（関数、ストアドプロシージャ、トリガー、およびパッケージ）を不正な検査から保護します。EDB*Wrap プログラムは、SPL または PL/pgSQL ソースコード（プレーンテキスト）を含むファイルを、読み取りがほとんど不可能な形式の同じコードを含むファイルに変換します。コードの難読化された形式を取得したら、そのコードを PostgreSQL サーバーに送信すると、サーバーはそれらのプログラムを難読化された形式で保存します。EDB*Wrap はコードを不明瞭にしますが、テーブル定義はまだ公開されています。

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

CREATE PACKAGE ステートメントをラップすると、パッケージ API が他の開発者から隠されます。パッケージ本体ではなくパッケージヘッダーをラップして、ユーザーがパッケージプロトタイプで定義されているパッケージプロトタイプおよびその他のパブリック変数を表示できるようにすることができます。ユーザーがパッケージに含まれるプロトタイプを確認できるようにするには、EDBWrap を使用して、edbwrap 入力ファイルの CREATE PACKAGE BODY ステートメントのみを難読化し、CREATE PACKAGE ステートメントを省略します。パッケージヘッダーソースはプレーンテキストで保存され、パッケージ本体のソースとパッケージ関数およびプロシージャは難読化されて保存されます。



ラップすると、ソースコードとプログラムをラップ解除またはデバッグすることはできません。リバースエンジニアリングは可能ですが、非常に困難です。

ソースファイル全体が 1 つのユニットにラップされます。ラップされたファイルに含まれる `psql` メタコマンドは、ファイルの実行時に認識されません。`psql` メタコマンドを含む難読化されたファイルを実行すると、構文エラーが発生します。`edbwrap` は SQL ソースコードを検証しません-平文形式に構文エラーが含まれている場合、`edbwrap` は文句を言いません。代わりに、難読化されたフォームを実行しようとする、サーバーはエラーを報告し、ファイル全体を中止します。

3.1 Using EDB*Wrap to Obfuscate Source Code

EDB*Wrap はコマンドラインユーティリティです。単一の入力ソースファイルを受け入れ、内容を難読化し、単一の出力ファイルを返します。`edbwrap` ユティリティを起動する場合、難読化するソースコードを含むファイルの名前を指定する必要があります。`edbwrap` が難読化された形式のコードを書き込むファイルの名前を指定することもできます。`edbwrap` には、3 つの異なるコマンドラインスタイルがあります。最初のスタイルは、Oracle の `wrap` ユティリティと互換性があります。

```
edbwrap iname=<input_file> [oname= <output_file>]
```

`iname=input_file` 引数は、入力ファイルの名前を指定します。`input_file` に拡張子が含まれていない場合、`edbwrap` は `input_file.sql` という名前のファイルを検索します

`oname=output_file` 引数（オプション）は、出力ファイルの名前を指定します。`output_file` に拡張子が含まれていない場合、`edbwrap` は名前に `.plb` を追加します。

出力ファイル名を指定しない場合、`edbwrap` は入力ファイル名から派生した名前のファイルに書き込みます。`edbwrap` は入力ファイル名からサフィックス（通常 `.sql`）を取り除き、`.plb` を追加します。

`edbwrap` は、他の 2 つのコマンドラインスタイルを提供します。

```
edbwrap --iname <input_file> [--oname <output_file>]
edbwrap -i <input_file> [-o <output_file>]
```

コマンドラインスタイルを混在させることができます。入力ファイル名と出力ファイル名を導出するための規則は、使用するスタイルに関係なく同一です。

`edbwrap` が難読化されたコードを含むファイルを生成したら、通常、`edb-psql` などのクライアントアプリケーションを使用してそのファイルを PostgreSQL サーバーにフィードします。サーバーは難読化されたコードを 1 行ずつ実行し、SPL および PL/pgSQL プログラムのソースコードをラップ形式で保存します。

要約すると、EDB*Wrap でコードを難読化するには、次のようにします。

1. ソースコードファイルを作成します。
2. EDB*Wrap を呼び出して、コードを難読化します。
3. ファイルをプレーンテキスト形式でインポートします。

次のシーケンスは edbwrap 機能を示しています。

まず、list_emp プロシージャのソースコードを（プレーンテキスト形式で）作成します。

```
[bash] ^c2^a0cat listemp.sql
CREATE OR REPLACE PROCEDURE list_emp
IS
^c2^a0 ^c2^a0 v_empno ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 NUMBER(4);
^c2^a0 ^c2^a0 v_ename ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 VARCHAR2(10);
^c2^a0 ^c2^a0 CURSOR emp_cur IS
^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 SELECT empno, ename FROM emp ORDER BY empno;
BEGIN
^c2^a0 ^c2^a0 OPEN emp_cur;
^c2^a0 ^c2^a0 DBMS_OUTPUT.PUT_LINE('EMPNO ^c2^a0 ^c2^a0ENAME');
^c2^a0 ^c2^a0 DBMS_OUTPUT.PUT_LINE('----- ^c2^a0 ^c2^a0-----');
^c2^a0 ^c2^a0 LOOP
^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 FETCH emp_cur INTO v_empno, v_ename;
^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 EXIT WHEN emp_cur%NOTFOUND;
^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 DBMS_OUTPUT.PUT_LINE(v_empno ||\| ' ^c2^a0_
^c2^a0 ' ||\| v_ename);
^c2^a0 ^c2^a0 END LOOP;
^c2^a0 ^c2^a0 CLOSE emp_cur;
END;
/
```

edb-psql などのクライアントアプリケーションで list_emp プロシージャをインポートできます。

```
[bash] edb-psql edb
Welcome to edb-psql 8.4.3.2, the EnterpriseDB interactive terminal.
Type: ^c2^a0\copyright for distribution terms
^c2^a0 ^c2^a0 ^c2^a0 \h for help with SQL commands
^c2^a0 ^c2^a0 ^c2^a0 \? for help with edb-psql commands
^c2^a0 ^c2^a0 ^c2^a0 \g or terminate with semicolon to execute query
^c2^a0 ^c2^a0 ^c2^a0 \q to quit
edb=# \i listemp.sql
CREATE PROCEDURE
```

pg_proc システムテーブルを調べると、サーバーに保存されているプレーンテキストのソースコードを表示できます。

```
edb=# SELECT prosrc FROM pg_proc WHERE proname = 'list_emp';
^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0_
^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 prosrc
-----

        v_empno ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 NUMBER(4);
^c2^a0 ^c2^a0 v_ename ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 VARCHAR2(10);
^c2^a0 ^c2^a0 CURSOR emp_cur IS
```

(次のページに続く)

(前のページからの続き)

```

^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 SELECT empno, ename FROM emp ORDER BY empno;
BEGIN
^c2^a0 ^c2^a0 OPEN emp_cur;
^c2^a0 ^c2^a0 DBMS_OUTPUT.PUT_LINE('EMPNO ^c2^a0 ^c2^a0ENAME');
^c2^a0 ^c2^a0 DBMS_OUTPUT.PUT_LINE('----- ^c2^a0 ^c2^a0-----');
^c2^a0 ^c2^a0 LOOP
^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 FETCH emp_cur INTO v_empno, v_ename;
^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 EXIT WHEN emp_cur%NOTFOUND;
^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 DBMS_OUTPUT.PUT_LINE(v_empno ||'|'| '^c2^a0
↪^c2^a0 '|\\|\\| v_ename);
^c2^a0 ^c2^a0 END LOOP;
^c2^a0 ^c2^a0 CLOSE emp_cur;
END
(1 row)

edb=# quit

```

次に、EDB*Wrap を使用してプレーンテキストファイルを難読化します。

```

[bash] edbwrap -i listemp.sql
EDB*Wrap Utility: Release 8.4.3.2

Copyright (c) 2004-2020 EnterpriseDB Corporation. ^c2^a0All Rights Reserved.

Using encoding UTF8 for input
Processing listemp.sql to listemp.plb

Examining the contents of the output file (listemp.plb) file reveals
that the code is obfuscated:

[bash] cat listemp.plb
$__EDBwrapped__$
UTF8
d+6DL30RVaGjYMIzkuoSZAQgtBw7MhYFuAFkBsfYfhdJ0rjwBv+bHr1FCyH6j9SgH
movU+bYI+jR+hR2jbzq3sovHKEyZIp9y3/GckbQgualRhIlGpyWfE0dltDUpkYRLN
/OUXmk0/P4H6EI98sAHevGDhOWI+58DjJ44qhZ+l5NNEVxbWDztpb/s5sdX4660qQ
Ozx3/gh8VvkqS2JbcxYmpjmrwVr6fAXfb68Ml9mW2Hl7fNtxcb5kjSzXvfWR2XYzJf
KFNrEhbLlDlTVlSEC5wE6lGlwhYvXOf22m1R2IFns0MtF9fwnBWAs1YqjR00j6+fc
er/f/efAFh4=
$__EDBwrapped__$

```

ラップされたファイルの 2 行目にエンコード名が含まれていることがわかります（この場合、エンコードは UTF8 です）。ファイルを難読化すると、edbwrap はロケールを調べて入力ファイルのエンコーディングを推測します。たとえば、ロケールが en_US.utf8 に設定されているときに edbwrap を実行している場合、edbwrap は入力ファイルが UTF8 でエンコードされていると想定します。edbwrap を実行した後、出力ファイルを必ず確認してください。ラップされたファイルに含まれるロケールが入力ファイルのエンコーディングと一致しない場合、ロケールを変更して入力ファイルを再ラップする必要があります。

プレーンテキストコードで動作する同じツールを使用して、難読化されたコードを PostgreSQL サーバーにインポートできます。

```
[bash] edb-psql edb
Welcome to edb-psql 8.4.3.2, the EnterpriseDB interactive terminal.
Type: ^c2^a0\copyright for distribution terms
^c2^a0^c2^a0 ^c2^a0 ^c2^a0 \h for help with SQL commands
^c2^a0^c2^a0 ^c2^a0 ^c2^a0 \? for help with edb-psql commands
^c2^a0^c2^a0 ^c2^a0 ^c2^a0 \g or terminate with semicolon to execute query
^c2^a0^c2^a0 ^c2^a0 ^c2^a0 \q to quit

edb=# \i listemp.plb
CREATE PROCEDURE

Now, the pg_proc system table contains the obfuscated code:

edb=# SELECT prosrc FROM pg_proc WHERE proname = 'list_emp';
^c2^a0^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0
↪^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0 ^c2^a0
↪^c2^a0 prosrc
-----
^c2^a0 $__EDBwrapped__$
^c2^a0 UTF8
^c2^a0 dw4B9Tz69J3W0sy0GgYJQa+G2sLZ3IOyxS8pDyuOTFuiYe/EXiEatwwG3h3tdJk
^c2^a0 ea+AIp35dS/4idbN8wpegM3s994dQ3R97NgNHfvTQnO2vtd4wQtsQ/Zc4v4Lhfj
^c2^a0 nlV+A4UpHI5oQEnXeAch2LcRD87hkU0uo1ESeQV8IrXaj9BsZr+ueROnwhGs/Ec
^c2^a0 pva/tRV4m9RusFn0wyr38u4Z8w4dfnPW184Y3o6It4b3aH07WxTkWrMLmOZW1jJ
^c2^a0 Nu6u4o+ezO64G9QKPazgehs1v4JB9NQnuocActfDSPMY7R7anmgw
^c2^a0 $__EDBwrapped__$
(1 row)
```

プレーンテキスト形式を呼び出すのと同じ方法で、難読化されたコードを呼び出します。

```
edb=# exec list_emp;

EMPNO ^c2^a0 ^c2^a0ENAME
----- ^c2^a0 ^c2^a0-----
7369 ^c2^a0 ^c2^a0 SMITH
7499 ^c2^a0 ^c2^a0 ALLEN
7521 ^c2^a0 ^c2^a0 WARD
7566 ^c2^a0 ^c2^a0 JONES
7654 ^c2^a0 ^c2^a0 MARTIN
7698 ^c2^a0 ^c2^a0 BLAKE
7782 ^c2^a0 ^c2^a0 CLARK
7788 ^c2^a0 ^c2^a0 SCOTT
7839 ^c2^a0 ^c2^a0 KING
7844 ^c2^a0 ^c2^a0 TURNER
7876 ^c2^a0 ^c2^a0 ADAMS
```

(次のページに続く)

(前のページからの続き)

```
7900  ^^c2^^a0  ^^c2^^a0  JAMES
7902  ^^c2^^a0  ^^c2^^a0  FORD
7934  ^^c2^^a0  ^^c2^^a0  MILLER
```

```
EDB-SPL Procedure successfully completed
edb=# quit
```

pg_dump を使用してデータベースをバックアップすると、ラップされたプログラムはアーカイブファイルで難読化されたままになります。

Postgres サーバーによって生成される監査ログには、ラップされたプログラムがプレーンテキスト形式で表示されることに注意してください。ソースコードは、プログラムの実行中に生成される SQL エラーメッセージにプレーンテキストで表示されます。

注釈: 現時点では、次のステートメントで作成されたオブジェクトの本文は難読化された形式で保存されません。

```
CREATE [OR REPLACE] TYPE type_name AS OBJECT
CREATE [OR REPLACE] TYPE type_name UNDER type_name
CREATE [OR REPLACE] TYPE BODY type_name
```

第 4 章

動的ランタイム計装ツールアーキテクチャ (DRITA)

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

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

4.1 Configuring and Using DRITA

AdvancedServer の postgresql.conf ファイルには、タイミングデータの収集を制御する timed_statistics という名前の構成パラメーターが含まれています。有効なパラメーター値は TRUE または FALSE です。デフォルト値は FALSE です。

これは、postgresql.conf ファイルで、またはセッションの進行中に変更できる動的パラメーターです。DRITA を有効にするには、次のいずれかを行う必要があります。

postgresql.conf ファイルを変更し、timed_statistics パラメーターを TRUE に設定します。

または

EDB-PSQL クライアントを使用してサーバーに接続し、次のコマンドを呼び出します。

```
SET timed_statistics = TRUE
```

timed_statistics パラメータを変更した後、開始スナップショットを作成します。スナップショットは、各

タイマーとイベントカウンターの現在の状態をキャプチャします。サーバーは、開始スナップショットを後のスナップショットと比較して、システムパフォーマンスを測定します。

edbsnap() 関数を使用して、開始スナップショットを取得します。

```
edb=# SELECT * FROM edbsnap();
      edbsnap
-----
Statement processed.
(1 row)
```

次に、評価するワークロードを実行します。ワークロードが完了したら（またはワークロード中の戦略的なポイントで）、別のスナップショットを作成します。

```
edb=# SELECT * FROM edbsnap();
      edbsnap
-----
Statement processed.
(1 row)
```

セッション中に複数のスナップショットをキャプチャできます。次に、DRITA 機能とレポートを使用してスナップショットを管理および比較し、パフォーマンス情報を評価します。

4.2 DRITA Functions

DRITA 関数を使用して、待機情報を収集し、スナップショットを管理できます。DRITA 機能は、インストールが Oracle データベースと互換性があるか、PostgreSQL 互換モードであるかに関係なく、AdvancedServer13 によって完全にサポートされます。

4.2.1 get_snaps()

get_snaps() 関数は、現在のスナップショットのリストを返します。署名は次のとおりです。

```
get_snaps()
```

次の例は、get_snaps() 関数を使用してスナップショットのリストを表示する方法を示しています。

```
SELECT * FROM get_snaps();
      get_snaps
-----
 1 25-JUL-18 09:49:04.224597
 2 25-JUL-18 09:49:09.310395
 3 25-JUL-18 09:49:14.378728
 4 25-JUL-18 09:49:19.448875
 5 25-JUL-18 09:49:24.52103
 6 25-JUL-18 09:49:29.586889
 7 25-JUL-18 09:49:34.65529
 8 25-JUL-18 09:49:39.723095
 9 25-JUL-18 09:49:44.788392
10 25-JUL-18 09:49:49.855821
11 25-JUL-18 09:49:54.919954
12 25-JUL-18 09:49:59.987707
(12 rows)
```

結果リストの最初の列には、スナップショット識別子が表示されます。2 番目の列には、スナップショットがキャプチャされた日時が表示されます。

4.2.2 sys_rpt()

sys_rpt() 関数はシステム待機情報を返します。署名は次のとおりです。

```
sys_rpt(<beginning_id>, <ending_id>, <top_n>)
```

Parameters

beginning_id

beginning_id は、開始セッション ID を表す整数値です。

ending_id

ending_id は、終了セッション識別子を表す整数値です。

top_n

top_n は返す行数を表します

この例は、sys_rpt() 関数の呼び出しを示しています。

```
SELECT * FROM sys_rpt(9, 10, 10);
```

sys_rpt

WAIT NAME	COUNT	WAIT TIME	% WAIT
wal flush	8359	1.357593	30.62
wal write	8358	1.349153	30.43
wal file sync	8358	1.286437	29.02
query plan	33439	0.439324	9.91
db file extend	54	0.000585	0.01
db file read	31	0.000307	0.01
other lwlock acquire	0	0.000000	0.00
ProcArrayLock	0	0.000000	0.00
CLogControlLock	0	0.000000	0.00
(11 rows)			

結果セットに表示される情報には次のものが含まれます。

列名	説明
WAIT NAME	待機の名前。
COUNT	待機イベントが発生した回数。
WAIT TIME	待機イベントの時間（秒単位）。
% WAIT	このセッションのこの待機で使用する合計待機時間の割合。

4.2.3 sess_rpt()

sess_rpt() 関数は、セッション待機情報を返します。署名は次のとおりです。

```
sess_rpt(<beginning_id>, <ending_id>, <top_n>)
```

Parameters

beginning_id

beginning_id は、開始セッション ID を表す整数値です。

ending_id

ending_id は、終了セッション識別子を表す整数値です。

top_n

top_n は返す行数を表します

次の例は、sess_rpt() 関数の呼び出しを示しています。

```
SELECT * FROM sess_rpt(8, 9, 10);
```

sess_rpt					
ID	USER	WAIT NAME	COUNT	TIME	% WAIT SES %
WAIT ALL					
3501	enterprise	wal flush	8354	1.354958	30.61
30.61					
3501	enterprise	wal write	8354	1.348192	30.46
30.46					
3501	enterprise	wal file sync	8354	1.285607	29.04
29.04					
3501	enterprise	query plan	33413	0.436901	9.87
9.87					
3501	enterprise	db file extend	54	0.000578	0.01
0.01					
3501	enterprise	db file read	56	0.000541	0.01
0.01					
3501	enterprise	ProcArrayLock	0	0.000000	0.00
0.00					
3501	enterprise	CLogControlLock	0	0.000000	0.00
0.00					

(10 rows)

結果セットに表示される情報には次のものが含まれます。

列名	説明
ID	セッションのプロセス ID。
USER	待機中のユーザーの名前。
WAIT NAME	待機イベントの名前。
COUNT	待機イベントが発生した回数。
TIME	秒単位の待機イベントの長さ。
% WAIT SES	このセッションのこの待機で使用する合計待機時間の割合。
% WAIT ALL	この待機（すべてのセッション）で使用する合計待機時間の割合。

4.2.4 sessid_rpt()

sessid_rpt() 関数は、指定されたバックエンドのセッション ID 情報を返します。署名は次のとおりです。

```
sessid_rpt(<beginning_id>, <ending_id>, <backend_id>)
```

Parameters

beginning_id

beginning_id は、開始セッション ID を表す整数値です。

ending_id

ending_id は、終了セッション識別子を表す整数値です。

backend_id

backend_id は、バックエンド識別子を表す整数値です。

次のコードサンプルは、sessid_rpt() の呼び出しを示しています。

```
SELECT * FROM sessid_rpt(8, 9, 3501);
```

sessid_rpt						
ID	USER	WAIT NAME	COUNT	TIME	% WAIT	SES %
3501	enterprise	CLogControlLock	0	0.000000	0.00	
0.00						
3501	enterprise	ProcArrayLock	0	0.000000	0.00	
0.00						
3501	enterprise	db file read	56	0.000541	0.01	
0.01						
3501	enterprise	db file extend	54	0.000578	0.01	
0.01						
3501	enterprise	query plan	33413	0.436901	9.87	
9.87						
3501	enterprise	wal file sync	8354	1.285607	29.04	
29.04						
3501	enterprise	wal write	8354	1.348192	30.46	
30.46						
3501	enterprise	wal flush	8354	1.354958	30.61	
30.61						

(10 rows)

結果セットに表示される情報には次のものが含まれます。

列名	説明
ID	待機のプロセス ID。
USER	セッションを所有するユーザーの名前。
WAIT NAME	待機イベントの名前。
COUNT	待機イベントが発生した回数。
TIME	秒単位の待機の長さ。
% WAIT SES	このセッションのこの待機で使われる合計待機時間の割合。
% WAIT ALL	この待機（すべてのセッション）で使われる合計待機時間の割合。

4.2.5 sesshist_rpt()

sesshist_rpt () 関数は、指定されたバックエンドのセッション待機情報を返します。署名は次のとおりです。

```
sesshist_rpt (<snapshot_id>, <session_id>)
```

Parameters

snapshot_id

snapshot_id は、スナップショットを識別する整数値です。

session_id

session_id はセッションを表す整数値です。

次の例は、sesshist_rpt () 関数の呼び出しを示しています。

注釈: 次の例は短縮されています。実際には 1300 行以上が生成されました。

```
SELECT * FROM sesshist_rpt (9, 3501);
```

sesshist_rpt							
ID	USER	SEQ	WAIT NAME	ELAPSED	File	Name	#
of Blk	Sum of Blks						
3501	enterprise 1	query plan	13	0	N/A		
0	0						
3501	enterprise 1	query plan	13	0	edb_password_history		
0	0						
3501	enterprise 1	query plan	13	0	edb_password_history		

(次のページに続く)

(前のページからの続き)

0	0				
3501	enterprise 1	query plan	13	0	edb_password_history
0	0				
3501	enterprise 1	query plan	13	0	edb_profile
0	0				
3501	enterprise 1	query plan	13	0	edb_profile_name_ind
0	0				
3501	enterprise 1	query plan	13	0	edb_profile_oid_inde
0	0				
3501	enterprise 1	query plan	13	0	edb_profile_password
0	0				
3501	enterprise 1	query plan	13	0	edb_resource_group
0	0				
3501	enterprise 1	query plan	13	0	edb_resource_group_n
0	0				
3501	enterprise 1	query plan	13	0	edb_resource_group_o
0	0				
3501	enterprise 1	query plan	13	0	pg_attribute
0	0				
3501	enterprise 1	query plan	13	0	pg_attribute_relid_a
0	0				
3501	enterprise 1	query plan	13	0	pg_attribute_relid_a
0	0				
3501	enterprise 1	query plan	13	0	pg_auth_members
0	0				
3501	enterprise 1	query plan	13	0	pg_auth_members_memb
0	0				
3501	enterprise 1	query plan	13	0	pg_auth_members_role
0	0				
			.		
			.		
			.		
3501	enterprise 2	wal flush	149	0	N/A
0	0				
3501	enterprise 2	wal flush	149	0	edb_password_history
0	0				
3501	enterprise 2	wal flush	149	0	edb_password_history
0	0				
3501	enterprise 2	wal flush	149	0	edb_password_history
0	0				
3501	enterprise 2	wal flush	149	0	edb_profile
0	0				
3501	enterprise 2	wal flush	149	0	edb_profile_name_ind
0	0				
3501	enterprise 2	wal flush	149	0	edb_profile_oid_inde
0	0				
3501	enterprise 2	wal flush	149	0	edb_profile_password
0	0				

(次のページに続く)

(前のページからの続き)

3501	enterprise 2	wal flush	149	0	edb_resource_group
0	0				
3501	enterprise 2	wal flush	149	0	edb_resource_group_n
0	0				
3501	enterprise 2	wal flush	149	0	edb_resource_group_o
0	0				
3501	enterprise 2	wal flush	149	0	pg_attribute
0	0				
3501	enterprise 2	wal flush	149	0	pg_attribute_relid_a
0	0				
3501	enterprise 2	wal flush	149	0	pg_attribute_relid_a
0	0				
3501	enterprise 2	wal flush	149	0	pg_auth_members
0	0				
3501	enterprise 2	wal flush	149	0	pg_auth_members_memb
0	0				
3501	enterprise 2	wal flush	149	0	pg_auth_members_role
0	0				
		.			
		.			
		.			
3501	enterprise 3	wal write	148	0	N/A
0	0				
3501	enterprise 3	wal write	148	0	edb_password_history
0	0				
3501	enterprise 3	wal write	148	0	edb_password_history
0	0				
3501	enterprise 3	wal write	148	0	edb_password_history
0	0				
3501	enterprise 3	wal write	148	0	edb_profile
0	0				
3501	enterprise 3	wal write	148	0	edb_profile_name_ind
0	0				
3501	enterprise 3	wal write	148	0	edb_profile_oid_inde
0	0				
3501	enterprise 3	wal write	148	0	edb_profile_password
0	0				
3501	enterprise 3	wal write	148	0	edb_resource_group
0	0				
3501	enterprise 3	wal write	148	0	edb_resource_group_n
0	0				
3501	enterprise 3	wal write	148	0	edb_resource_group_o
0	0				
3501	enterprise 3	wal write	148	0	pg_attribute
0	0				
3501	enterprise 3	wal write	148	0	pg_attribute_relid_a
0	0				
3501	enterprise 3	wal write	148	0	pg_attribute_relid_a

(次のページに続く)

(前のページからの続き)

0	0				
3501	enterprise 3	wal write	148	0	pg_auth_members
0	0				
3501	enterprise 3	wal write	148	0	pg_auth_members_memb
0	0				
3501	enterprise 3	wal write	148	0	pg_auth_members_role
0	0				
		.			
		.			
		.			
3501	enterprise 24	wal write	130	0	pg_toast_1255
0	0				
3501	enterprise 24	wal write	130	0	pg_toast_1255_index
0	0				
3501	enterprise 24	wal write	130	0	pg_toast_2396
0	0				
3501	enterprise 24	wal write	130	0	pg_toast_2396_index
0	0				
3501	enterprise 24	wal write	130	0	pg_toast_2964
0	0				
3501	enterprise 24	wal write	130	0	pg_toast_2964_index
0	0				
3501	enterprise 24	wal write	130	0	pg_toast_3592
0	0				
3501	enterprise 24	wal write	130	0	pg_toast_3592_index
0	0				
3501	enterprise 24	wal write	130	0	pg_type
0	0				
3501	enterprise 24	wal write	130	0	pg_type_oid_index
0	0				
3501	enterprise 24	wal write	130	0	pg_type_typname_nsp_
0	0				
(1304 rows)					

結果セットに表示される情報には次のものが含まれます。

列名	説明
ID	システムが割り当てた待機の識別子。
USER	待機が発生したユーザーの名前。
SEQ	待機イベントのシーケンス番号。
WAIT NAME	待機イベントの名前。
ELAPSED	マイクロ秒単位の待機イベントの長さ。
File	ファイルの relfilenode 番号。
Name	使用可能な場合、待機イベントに関連するファイル名の名前。
# of Blk	イベントの特定のインスタンスに対して読み取りまたは書き込みされたブロック番号。
Sum of Blks	読み取られたブロックの数。

4.2.6 purgesnap()

purgesnap() 機能は、スナップショットテーブルからスナップショットの範囲を削除します。署名は次のとおりです。

```
purgesnap(<beginning_id>, <ending_id>)
```

Parameters

beginning_id

beginning_id は、開始セッション ID を表す整数値です。

ending_id

ending_id は、終了セッション識別子を表す整数値です。

purgesnap() は beginning_id から ending_id までのすべてのスナップショットを削除します (包括的) :

```
SELECT * FROM purgesnap(6, 9);

          purgesnap
-----
Snapshots in range 6 to 9 deleted.
(1 row)
```

例の実行後の get_snaps() 関数の呼び出しは、スナップショット 6 から 9 がスナップショットテーブルから削除されたことを示しています。

```
SELECT * FROM get_snaps();

          get_snaps
-----
```

(次のページに続く)

(前のページからの続き)

```

1 25-JUL-18 09:49:04.224597
2 25-JUL-18 09:49:09.310395
3 25-JUL-18 09:49:14.378728
4 25-JUL-18 09:49:19.448875
5 25-JUL-18 09:49:24.52103
10 25-JUL-18 09:49:49.855821
11 25-JUL-18 09:49:54.919954
12 25-JUL-18 09:49:59.987707
(8 rows)

```

4.2.7 truncsnap()

`truncsnap()` 関数を使用して、スナップショットテーブルからすべてのレコードを削除します。署名は次のとおりです。

```
truncsnap()
```

例:

```

SELECT * FROM truncsnap();

      truncsnap
-----
Snapshots truncated.
(1 row)

```

`truncsnap()` 関数を呼び出した後に `get_snaps()` 関数を呼び出すと、スナップショットテーブルからすべてのレコードが削除されたことがわかります。

```

SELECT * FROM get_snaps();

get_snaps
-----
(0 rows)

```

4.3 Simulating Statspack AWR Reports

このセクションで説明する関数は、OracleStatspack/AWR（自動ワークロードリポジトリ）レポートに含まれる情報に相当する情報を返します。スナップショットを作成するとき、システムカタログテーブルのパフォーマンスデータは履歴テーブルに保存されます。以下にリストされているレポート機能は、指定された 2 つのスナップショットの違いについてレポートします。

- `stat_db_rpt()`
- `stat_tables_rpt()`
- `statio_tables_rpt()`
- `stat_indexes_rpt()`
- `statio_indexes_rpt()`

レポート機能は個別に実行するか、`edbreport()` 機能呼び出して 5 つすべての機能を実行できます。

4.3.1 edbreport()

`edbreport()` 機能には、他のレポート機能からのデータに加えて、追加のシステム情報が含まれます。署名は次のとおりです。

```
edbreport(<beginning_id>, <ending_id>)
```

Parameters

`beginning_id`

`beginning_id` は、開始セッション ID を表す整数値です。

`ending_id`

`ending_id` は、終了セッション識別子を表す整数値です。

`edbreport()` 関数の呼び出しは、システム情報と他の statspack 関数によって返されるレポートを含む複合レポートを返します。

```
SELECT * FROM edbreport(9, 10);

                                edbreport
-----
EnterpriseDB Report for database acctg                25-JUL-18
Version: PostgreSQL 13.0 (EnterpriseDB Advanced Server 13.0.0) on x86_64-pc-
linux-gnu,
compiled by gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-18), 64-bit
```

(次のページに続く)

(前のページからの続き)

```

Begin snapshot: 9 at 25-JUL-18 09:49:44.788392

End snapshot: 10 at 25-JUL-18 09:49:49.855821

Size of database acctg is 173 MB
  Tablespace: pg_default Size: 231 MB Owner: enterprisedb
  Tablespace: pg_global Size: 719 kB Owner: enterprisedb

Schema: pg_toast_temp_1 Size: 0 bytes Owner: enterprisedb
Schema: public Size: 158 MB Owner: enterprisedb

```

レポートの概要に表示される情報には、データベースの名前とバージョン、現在の日付、スナップショットの開始日時と終了日時、データベースとテーブルスペースの詳細、スキーマ情報が含まれます。

Top 10 Relations by pages	
TABLE	RELPAGES
-----	-----
pgbench_accounts	16394
pgbench_history	391
pg_proc	145
pg_attribute	92
pg_depend	81
pg_collation	60
edb\$stat_all_indexes	46
edb\$statio_all_indexes	46
pg_description	44
edb\$stat_all_tables	29

Top 10 Relations by pages セクションに表示される情報は次のとおりです。

列名	説明
TABLE	テーブルの名前。
RELPAGES	テーブル内のページ数。

Top 10 Indexes by pages	
INDEX	RELPAGES
-----	-----
pgbench_accounts_pkey	2745
pg_depend_reference_index	68
pg_depend_depender_index	63
pg_proc_proname_args_nsp_index	53
pg_attribute_relid_attnam_index	25

(次のページに続く)

(前のページからの続き)

```

pg_description_o_c_o_index          24
pg_attribute_relid_attnum_index     17
pg_proc_oid_index                   14
pg_collation_name_enc_nsp_index     12
edb$stat_idx_pk
10

```

Top 10 Indexes by pages セクションに表示される情報は次のとおりです。

列名	説明
INDEX	インデックスの名前。
RELPAGES	インデックス内のページ数。

Top 10 Relations by DML				
SCHEMA	RELATION	UPDATES	DELETES	INSERTS
public	pgbench_accounts	117209	0	1000000
public	pgbench_tellers	117209	0	100
public	pgbench_branches	117209	0	10
public	pgbench_history	0	0	117209

Top 10 Relations by DML セクションに表示される情報は次のとおりです。

列名	説明
SCHEMA	テーブルが存在するスキーマの名前。
RELATION	テーブルの名前。
UPDATES	テーブルで実行された更新の数。
DELETES	テーブルで実行された DELETE の数。
INSERTS	テーブルで実行された INSERTS の数。

DATA from pg_stat_database					
DATABASE	NUMBACKENDS	XACT COMMIT	XACT ROLLBACK	BLKS READ	BLKS HIT
HIT RATIO					

acctg	0	8261	0	117	127985
99.91					

レポートの pg_stat_database セクションのデータに表示される情報には、次のものが含まれます。

列名	説明
DATABASE	データベースの名前。
NUMBACKENDS	このデータベースに現在接続されているバックエンドの数。これは、現在の状態を反映する値を返すこのビューの唯一の列です。他のすべての列は、最後のリセット以降の累積値を返します。
XACT COMMIT	コミットされたこのデータベース内のトランザクションの数。
XACT ROLLBACK	ロールバックされたこのデータベース内のトランザクションの数。
BLKS READ	このデータベースで読み取られたディスクブロックの数。
BLKS HIT	バッファキャッシュで既にディスクブロックが見つかった回数（読み取りが不要な場合）。
HIT RATIO	共有バッファキャッシュでブロックが見つかった回数の割合。

DATA from pg_buffercache	
RELATION	BUFFERS
pgbench_accounts	16665
pgbench_accounts_pkey	2745
pgbench_history	751
edb\$statio_all_indexes	94
edb\$stat_all_indexes	94
edb\$stat_all_tables	60
edb\$statio_all_tables	56
edb\$session_wait_history	34
edb\$statio_idx_pk	17
pg_depend	17

レポートの DATA from pg_buffercache セクションに表示される情報は次のとおりです。

列名	説明
RELATION	テーブルの名前。
BUFFERS	リレーションで使用される共有バッファの数。

注釈: DATA from pg_buffercache のレポートを取得するには、pg_buffercache モジュールがデータベースにインストールされている必要があります。CREATE EXTENSION コマンドでインストールを実行します。

CREATE EXTENSION 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-createextension.html>

DATA from pg_stat_all_tables ordered by seq scan							
SCHEMA		RELATION			SEQ SCAN	REL TUP READ	
IDX	SCAN						
IDX	TUP	READ	INS	UPD	DEL		

public					pgbench_branches	8258	82580
0							
0			0	8258	0		
public					pgbench_tellers	8258	825800
0							
0			0	8258	0		
pg_catalog					pg_class	7	3969
92							
80			0	0	0		
pg_catalog					pg_index	5	950
31							
38			0	0	0		
pg_catalog					pg_namespace	4	144
5							
4			0	0	0		
pg_catalog					pg_database	2	12
7							
7			0	0	0		
pg_catalog					pg_am	1	1
0							
0			0	0	0		
pg_catalog					pg_authid	1	10
2							
2			0	0	0		
sys					callback_queue_table	0	0
0							
0			0	0	0		
sys					edb\$session_wait_history	0	0
0							
0			125	0	0		

DATA from pg_stat_all_tables ordered by seq scan セクションに表示される情報は次のとおりです。

列名	説明
SCHEMA	テーブルが存在するスキーマの名前。
RELATION	テーブルの名前。
SEQ SCAN	このテーブルで開始された順次スキャンの数。
REL TUP READ	テーブルで読み取られたタプルの数。
IDX SCAN	テーブルで開始されたインデックススキャンの数。
IDX TUP READ	読み取られたインデックスタプルの数。
INS	挿入された行の数。
UPD	更新された行の数。
DEL	削除された行の数。

DATA from pg_stat_all_tables ordered by rel tup read						
SCHEMA	RELATION	SEQ SCAN	REL TUP READ	IDX SCAN	INS	UPD
DEL						

public	pgbench_tellers	8258	825800	0		
0					0	8258
public	pgbench_branches	8258	82580	0		
0					0	8258
pg_catalog	pg_class	7	3969	92		
80					0	0
pg_catalog	pg_index	5	950	31		
38					0	0
pg_catalog	pg_namespace	4	144	5		
4					0	0
pg_catalog	pg_database	2	12	7		
7					0	0
pg_catalog	pg_authid	1	10	2		
2					0	0
pg_catalog	pg_am	1	1	0		
0					0	0
sys	callback_queue_table	0	0	0		
0					0	0
sys	edb\$session_wait_history	0	0	0		
0					125	0

DATA from pg_stat_all_tables ordered by rel tup read セクションに表示される情報は次のとおりです。

列名	説明
SCHEMA	テーブルが存在するスキーマの名前。
RELATION	テーブルの名前。
SEQ SCAN	テーブルで実行された順次スキャンの数。
REL TUP READ	テーブルから読み取られたタプルの数。
IDX SCAN	テーブルで実行されたインデックススキャンの数。
IDX TUP READ	読み取られたインデックスタプルの数。
INS	挿入された行の数。
UPD	更新された行の数。
DEL	削除された行の数。

DATA from pg_statio_all_tables							
SCHEMA	RELATION		HEAP	HEAP	IDX	IDX	
TOAST	TOAST	TIDX	TIDX	READ	HIT	READ	HIT
READ	HIT	READ	HIT				

public		pgbench_accounts	32	25016	0	49913	
0	0	0	0				
public		pgbench_tellers	0	24774	0	0	
0	0	0	0				
public		pgbench_branches	0	16516	0	0	
0	0	0	0				
public		pgbench_history	53	8364	0	0	
0	0	0	0				
pg_catalog		pg_class	0	199	0	187	
0	0	0	0				
pg_catalog		pg_attribute	0	198	0	395	
0	0	0	0				
pg_catalog		pg_proc	0	75	0	153	
0	0	0	0				
pg_catalog		pg_index	0	56	0	33	
0	0	0	0				
pg_catalog		pg_amop	0	48	0	56	
0	0	0	0				
pg_catalog		pg_namespace	0	28	0	7	
0	0	0	0				

DATA from pg_statio_all_tables セクションに表示される情報は次のとおりです。

列名	説明
SCHEMA	テーブルが存在するスキーマの名前。
RELATION	テーブルの名前。
HEAP READ	読み込まれたヒープブロックの数。
HEAP HIT	ヒットしたヒープブロックの数。
IDX READ	読み取られたインデックスブロックの数。
IDX HIT	ヒットしたインデックスブロックの数。
TOAST READ	読み取られたトーストブロックの数。
TOAST HIT	ヒットしたトーストブロックの数。
TIDX READ	読み取られたトーストインデックスブロックの数。
TIDX HIT	ヒットしたトーストインデックスブロックの数。

DATA from pg_stat_all_indexes				
SCHEMA		RELATION		INDEX
IDX	SCAN	IDX TUP	READ	IDX TUP FETCH

-				

public			pgbench_accounts	pgbench_accounts_pkey
16516		16679	16516	
pg_catalog			pg_attribute	
pg_attribute_relid_attnum_index			196	402 402
pg_catalog			pg_proc	pg_proc_oid_index
70		70	70	
pg_catalog			pg_class	pg_class_oid_index
61		61	61	
pg_catalog			pg_class	
pg_class_relname_nsp_index				
31	19	19		
pg_catalog			pg_type	pg_type_oid_index
22		22	22	
pg_catalog			edb_policy	
edb_policy_object_name_index				
21		0	0	
pg_catalog			pg_amop	pg_amop_fam_strat_index
16		16	16	
pg_catalog			pg_index	pg_index_indexrelid_index
16		16	16	
pg_catalog			pg_index	pg_index_indrelid_index
15		22	22	

DATA from pg_stat_all_indexes セクションに表示される情報は次のとおりです。

列名	説明
SCHEMA	インデックスが存在するスキーマの名前。
RELATION	インデックスが定義されているテーブルの名前。
INDEX	インデックスの名前。
IDX SCAN	このインデックスで開始されたインデックススキャンの数。
IDX TUP READ	このインデックスのスキャンによって返されたインデックスエントリの数
IDX TUP FETCH	このインデックスを使用した単純なインデックススキャンによってフェッチされたライブテーブル行の数。

```

DATA from pg_statio_all_indexes

SCHEMA          RELATION          INDEX
IDX BLKS READ  IDX BLKS HIT
-----
public          pgbench_accounts      pgbench_accounts_pkey
0                49913
pg_catalog       pg_attribute
pg_attribute_relid_attnum_index    0      395
sys              edb$stat_all_indexes    edb$stat_idx_pk
1                382
sys              edb$statio_all_indexes    edb$statio_idx_pk
1                382
sys              edb$statio_all_tables      edb$statio_tab_pk
2                262
sys              edb$stat_all_tables        edb$stat_tab_pk
0                259
sys              edb$session_wait_history    session_waits_hist_pk
0                251
pg_catalog       pg_proc                pg_proc_oid_index
0                142
pg_catalog       pg_class                pg_class_oid_index
0                123
pg_catalog       pg_class                pg_class_relname_nsp_index
0                63

```

DATA from pg_statio_all_indexes セクションに表示される情報は次のとおりです。

列名	説明
SCHEMA	インデックスが存在するスキーマの名前。
RELATION	インデックスが定義されているテーブルの名前。
INDEX	インデックスの名前。
IDX BLKS READ	読み取られたインデックスブロックの数。
IDX BLKS HIT	ヒットしたインデックスブロックの数。

System Wait Information

WAIT NAME	COUNT	WAIT TIME	% WAIT
-----	-----	-----	-----
wal flush	8359	1.357593	30.62
wal write	8358	1.349153	30.43
wal file sync	8358	1.286437	29.02
query plan	33439	0.439324	9.91
db file extend	54	0.000585	0.01
db file read	31	0.000307	0.01
other lwlock acquire	0	0.000000	0.00
ProcArrayLock	0	0.000000	0.00
CLogControlLock	0	0.000000	0.00

System Wait Information セクションに表示される情報は次のとおりです。

列名	説明
WAIT NAME	待機の名前。
COUNT	待機イベントが発生した回数。
WAIT TIME	秒単位の待機時間の長さ。
% WAIT	このセッションのこの待機で使用する合計待機時間の割合。

Database Parameters from postgresql.conf

PARAMETER	SETTING	
CONTEXT	MINVAL	MAXVAL
-----	-----	-----
allow_system_table_mods		off
postmaster		
application_name		psql.bin
user		
archive_command		(disabled)
sighup		
archive_mode		off
postmaster		
archive_timeout		0
sighup	0	1073741823
array_nulls		on
user		
authentication_timeout		60
sighup	1	600
autovacuum		on
sighup		
autovacuum_analyze_scale_factor		0.1
sighup	0	100

(次のページに続く)

(前のページからの続き)

```

autovacuum_analyze_threshold      50
sighup          0                2147483647
autovacuum_freeze_max_age        200000000
postmaster    100000            2000000000
autovacuum_max_workers            3
postmaster     1                262143
autovacuum_multixact_freeze_max_age 400000000
postmaster     10000            2000000000
autovacuum_naptime                60
sighup          1                2147483
autovacuum_vacuum_cost_delay      20
sighup        -1                 100
.
.
.

```

Database Parameters from postgresql.conf セクションに表示される情報は次のとおりです。

列名	説明
PARAMETER	パラメーターの名前。
SETTING	パラメータに割り当てられた現在の値。
CONTEXT	パラメーター値の設定に必要なコンテキスト。
MINVAL	パラメーターに許可される最小値。
MAXVAL	パラメーターに許可される最大値。

4.3.2 stat_db_rpt()

署名は次のとおりです。

```
stat_db_rpt(<beginning_id>, <ending_id>)
```

Parameters

beginning_id

beginning_id は、開始セッション ID を表す整数値です。

ending_id

ending_id は、終了セッション識別子を表す整数値です。

次の例は、stat_db_rpt() 機能を示しています。

```
SELECT * FROM stat_db_rpt(9, 10);
```

```
stat_db_rpt
```

```
DATA from pg_stat_database
```

```
DATABASE  NUMBACKENDS    XACT COMMIT  XACT ROLLBACK  BLKS READ  BLKS HIT
HIT RATIO
```

```
-----
acctg      0              8261          0              117         127985
99.91
(5 rows)
```

レポートの pg_stat_database セクションのデータに表示される情報には、次のものが含まれます。

列名	説明
DATABASE	データベースの名前。
NUMBACKENDS	このデータベースに現在接続されているバックエンドの数。これは、現在の状態を反映する値を返すこのビューの唯一の列です。他のすべての列は、最後のリセット以降の累積値を返します。
XACT COMMIT	コミットされたこのデータベース内のトランザクションの数。
XACT ROLLBACK	このデータベースのロールバックされたトランザクションの数。
BLKS READ	読み取られたブロックの数。
BLKS HIT	ヒットしたブロックの数。
HIT RATIO	共有バッファキャッシュでブロックが見つかった回数の割合。

4.3.3 stat_tables_rpt()

署名は次のとおりです。

```
function_name(<beginning_id>, <ending_id>, <top_n>, <scope>)
```

Parameters

beginning_id

beginning_id は、開始セッション ID を表す整数値です。

ending_id

ending_id は、終了セッション識別子を表す整数値です。

top_n

top_n は返す行数を表します

scope

スコープは、関数が統計情報を返すテーブルを決定します。SYS、USER または ALL を指定します。

- SYS は、関数がシステム定義テーブルに関する情報を返すことを示します。テーブルは、pg_catalog、information_schema、または sys のスキーマのいずれかに格納されている場合、システムテーブルと見なされます。
- USER は、関数がユーザー定義テーブルに関する情報を返すことを示します。
- ALL は、関数がすべてのテーブルに関する情報を返すことを指定します。

stat_tables_rpt() 関数は 2 つの部分からなるレポートを返します。レポートの最初の部分には次が含まれます。

```
SELECT * FROM stat_tables_rpt(8, 9, 10, 'ALL');
```

stat_tables_rpt									

DATA from pg_stat_all_tables ordered by seq scan									
SCHEMA		RELATION			SEQ	SCAN	REL	TUP	READ
ID	SCAN								
ID	TUP	READ	INS	UPD	DEL				

public					pgbench_branches	8249		82490	
0									
0		0	8249	0					
public					pgbench_tellers	8249		824900	
0									
0		0	8249	0					
pg_catalog					pg_class	7		3969	
92									
80		0	0	0					
pg_catalog					pg_index	5		950	
31									
38		0	0	0					
pg_catalog					pg_namespace	4		144	
5									
4		0	0	0					
pg_catalog					pg_am	1		1	
0									
0		0	0	0					

(次のページに続く)

(前のページからの続き)

pg_catalog			pg_authid	1	10
2					
2	0	0	0		
pg_catalog			pg_database	1	6
3					
3	0	0	0		
sys			callback_queue_table	0	0
0					
0	0	0	0		
sys			edb\$session_wait_history	0	0
0					
0	125	0	0		

DATA from pg_stat_all_tables ordered by seq scan セクションに表示される情報は次のとおりです。

列名	説明
SCHEMA	テーブルが存在するスキーマの名前。
RELATION	テーブルの名前。
SEQ SCAN	テーブルでの順次スキャンの数。
REL TUP READ	テーブルから読み取られたタブルの数。
IDX SCAN	テーブルで実行されたインデックススキャンの数。
IDX TUP READ	テーブルから読み取られたインデックスタブルの数。
INS	挿入された行の数。
UPD	更新された行の数。
DEL	削除された行の数。

レポートの 2 番目の部分には次のものが含まれます。

DATA from pg_stat_all_tables ordered by rel tup read						
SCHEMA	RELATION			SEQ SCAN REL	TUP READ	IDX
SCAN						
IDX TUP READ	INS	UPD	DEL			

public			pgbench_tellers	8249	824900	0
0	0	8249	0			
public			pgbench_branches	8249	82490	0
0	0	8249	0			
pg_catalog			pg_class	7	3969	92
80	0	0	0			
pg_catalog			pg_index	5	950	31
38	0	0	0			

(次のページに続く)

(前のページからの続き)

pg_catalog		pg_namespace	4	144	5
4	0	0	0		
pg_catalog		pg_authid	1	10	2
2	0	0	0		
pg_catalog		pg_database	1	6	3
3	0	0	0		
pg_catalog		pg_am	1	1	0
0	0	0	0		
sys		callback_queue_table	0	0	0
0	0	0	0		
sys		edb\$session_wait_history	0	0	0
0	125	0	0		
(29 rows)					

DATA from pg_stat_all_tables ordered by rel tup read セクションに表示される情報は次のとおりです。

列名	説明
SCHEMA	テーブルが存在するスキーマの名前。
RELATION	テーブルの名前。
SEQ SCAN	テーブルで実行された順次スキャンの数。
REL TUP READ	テーブルから読み取られたタプルの数。
IDX SCAN	テーブルで実行されたインデックススキャンの数。
IDX TUP READ	インデックススキャンによってフェッチされたライブ行の数。
INS	挿入された行の数。
UPD	更新された行の数。
DEL	削除された行の数。

4.3.4 statio_tables_rpt()

署名は次のとおりです。

```
statio_tables_rpt(<beginning_id>, <ending_id>, <top_n>, <scope>)
```

Parameters

beginning_id

beginning_id は、開始セッション ID を表す整数値です。

ending_id

ending_id は、終了セッション識別子を表す整数値です。

top_n

top_n は返す行数を表します

scope

スコープは、関数が統計情報を返すテーブルを決定します。SYS、USER または ALL を指定します。

- SYS は、関数がシステム定義テーブルに関する情報を返すことを示します。テーブルは、pg_catalog、information_schema、または sys のスキーマのいずれかに格納されている場合、システムテーブルと見なされます。
- USER は、関数がユーザー定義テーブルに関する情報を返すことを示します。
- ALL は、関数がすべてのテーブルに関する情報を返すことを指定します。

statio_tables_rpt() 関数は、次を含むレポートを返します。

```
SELECT * FROM statio_tables_rpt(9, 10, 10, 'SYS');
```

statio_tables_rpt						

DATA from pg_statio_all_tables						
SCHEMA	RELATION		HEAP	HEAP	IDX	IDX
TOAST	TIDX	TIDX	READ	HIT	READ	HIT
HIT	READ	HIT	READ			

sys		edb\$stat_all_indexes	8	18	1	382
0						
0	0	0				
sys		edb\$statio_all_index	8	18	1	382
0						
0	0	0				
sys		edb\$statio_all_table	5	12	2	262
0						
0	0	0				
sys		edb\$stat_all_tables	4	10	0	259
0						
0	0	0				
sys		edb\$session_wait_his	2	6	0	251
0						
0	0	0				

(次のページに続く)

(前のページからの続き)

sys			edb\$session_waits	1	4	0	12
0							
0	0	0					
sys			callback_queue_table	0	0	0	0
0							
0	0	0					
sys			dual	0	0	0	0
0							
0	0	0					
sys			edb\$snap	0	1	0	2
0							
0	0	0					
sys			edb\$stat_database	0	2	0	7
0							
0	0	0					
(15 rows)							

DATA from pg_statio_all_tables セクションに表示される情報は次のとおりです。

列名	説明
SCHEMA	リレーションが存在するスキーマの名前。
RELATION	リレーションの名前。
HEAP READ	読み込まれたヒープブロックの数。
HEAP HIT	ヒットしたヒープブロックの数。
IDX READ	読み取られたインデックスブロックの数。
IDX HIT	ヒットしたインデックスブロックの数。
TOAST READ	読み取られたトーストブロックの数。
TOAST HIT	ヒットしたトーストブロックの数。
TIDX READ	読み取られたトーストインデックスブロックの数。
TIDX HIT	ヒットしたトーストインデックスブロックの数。

4.3.5 stat_indexes_rpt()

署名は次のとおりです。

```
stat_indexes_rpt(<beginning_id>, <ending_id>, <top_n>, <scope>)
```

Parameters

beginning_id

beginning_id は、開始セッション ID を表す整数値です。

ending_id

ending_id は、終了セッション識別子を表す整数値です。

top_n

top_n は返す行数を表します

scope

スコープは、関数が統計情報を返すテーブルを決定します。SYS、USER または ALL を指定します。

- SYS は、関数がシステム定義テーブルに関する情報を返すことを示します。テーブルは、pg_catalog, information_schema または sys のスキーマのいずれかに格納されている場合、システムテーブルと見なされます。
- USER は、関数がユーザー定義テーブルに関する情報を返すことを示します。
- ALL は、関数がすべてのテーブルに関する情報を返すことを指定します。

stat_indexes_rpt() 関数は、次を含むレポートを返します。

```
edb=# SELECT * FROM stat_indexes_rpt(9, 10, 10, 'ALL');

                                stat_indexes_rpt
-----
--
-----
      DATA from pg_stat_all_indexes

SCHEMA              RELATION              INDEX
IDX SCAN            IDX TUP READ IDX TUP FETCH
-----
--
-----
public              pgbench_accounts      pgbench_accounts_pkey
16516                16679                16516
pg_catalog           pg_attribute
pg_attribute_relid_attnum_index      196      402      402
pg_catalog           pg_proc              pg_proc_oid_index
70                   70                   70
pg_catalog           pg_class              pg_class_oid_index
61                   61                   61
pg_catalog           pg_class              pg_class_relname_nsp_index
31                   19                   19
pg_catalog           pg_type              pg_type_oid_index
22                   22                   22
pg_catalog           edb_policy
edb_policy_object_name_index
21                   0                   0
```

(次のページに続く)

(前のページからの続き)

pg_catalog		pg_amop	pg_amop_fam_strat_index
16	16	16	
pg_catalog		pg_index	pg_index_indexrelid_index
16	16	16	
pg_catalog		pg_index	pg_index_indrelid_index
15	22	22	
(14 rows)			

DATA from pg_stat_all_indexes セクションに表示される情報は次のとおりです。

列名	説明
SCHEMA	リレーションが存在するスキーマの名前。
RELATION	リレーションの名前。
INDEX	インデックスの名前。
IDX SCAN	スキャンされたインデックスの数。
IDX TUP READ	読み取られたインデックスタブルの数。
IDX TUP FETCH	フェッチされたインデックスタブルの数。

4.3.6 statio_indexes_rpt()

署名は次のとおりです。

```
statio_indexes_rpt(<beginning_id>, <ending_id>, <top_n>, <scope>)
```

Parameters

beginning_id

beginning_id は、開始セッション ID を表す整数値です。

ending_id

ending_id は、終了セッション識別子を表す整数値です。

top_n

top_n は返す行数を表します

scope

scope は、関数が統計情報を返すテーブルを決定します。SYS、USER または ALL を指定します。

- SYS は、関数がシステム定義テーブルに関する情報を返すことを示します。テーブルは、pg_catalog、information_schema、または sys のスキーマのいずれかに格納されている場合、システムテーブルと見なされます。

- USER は、関数がユーザー定義テーブルに関する情報を返すことを示します。
- ALL は、関数がすべてのテーブルに関する情報を返すことを指定します。

statio_indexes_rpt() 関数は、次を含むレポートを返します。

```
edb=# SELECT * FROM statio_indexes_rpt(9, 10, 10, 'SYS');
```

statio_indexes_rpt			

DATA from pg_statio_all_indexes			
SCHEMA		RELATION	INDEX
IDX	BLKS READ	IDX BLKS HIT	

		pg_catalog	pg_attribute
		pg_attribute_relid_attnum_index	0 395
1	382	sys	edb\$stat_all_indexes edb\$stat_idx_pk
1	382	sys	edb\$statio_all_indexes edb\$statio_idx_pk
2	262	sys	edb\$statio_all_tables edb\$statio_tab_pk
0	259	sys	edb\$stat_all_tables edb\$stat_tab_pk
0	251	sys	edb\$session_wait_history session_waits_hist_pk
0	142	pg_catalog	pg_proc pg_proc_oid_index
0	123	pg_catalog	pg_class pg_class_oid_index
0	63	pg_catalog	pg_class pg_class
0	45	pg_catalog	pg_class_relname_nsp_index pg_type pg_type_oid_index
(14 rows)			

DATA from pg_statio_all_indexes レポートに表示される情報には次のものが含まれます。

列名	説明
SCHEMA	リレーションが存在するスキーマの名前。
RELATION	インデックスが定義されているテーブルの名前。
INDEX	インデックスの名前。
IDX BLKS READ	読み取られたインデックスブロックの数。
IDX BLKS HIT	ヒットしたインデックスブロックの数。

4.4 Performance Tuning Recommendations

パフォーマンスチューニングに DRITA レポートを使用するには、特定のレポートの上位 5 つのイベントを確認し、不均衡に大きな割合のリソースを使用するイベントを探します。合理化されたシステムでは、おそらくユーザー I/O が最大の待機時間を構成します。待機は、CPU 使用率と合計時間のコンテキストで評価する必要があります。残りの時間が CPU 時間によって消費されている場合、2 時間の合計測定間隔のうち 2 分かかる場合、イベントは重要ではない可能性があります。全体の時間の中で最も高い割合を消費する応答時間のコンポーネント (CPU/"work/"時間または他の/"wait/"時間) を評価する必要があります。

イベントを評価するときは、次のことに注意してください。

イベントの種類	説明
チェックポイント待機	チェックポイント待機は、チェックポイントパラメータの調整が必要であることを示す場合があります (<code>checkpoint_segments</code> および <code>checkpoint_timeout</code>)。
WAL 関連の待機	WAL 関連の待機は、 <code>wal_buffers</code> が小さすぎることを示す場合があります。
SQL 解析の待機	待機数が多い場合は、準備済みステートメントを使用してみてください。
db ファイルのランダム読み取り	高い場合は、適切なインデックスと統計が存在することを確認してください。
db ファイルのランダム書き込み	高い場合、 <code>bgwriter_delay</code> を減らす必要があります。
btree ランダムロックの取得	インデックスが再構築されていることを示す場合があります。活動時間の少ない時間にインデックス作成をスケジュールします。

パフォーマンスのレビューには、ハードウェア、オペレーティングシステム、ネットワーク、およびアプリケーションの SQL ステートメントの綿密な調査も含める必要があります。

4.5 Event Descriptions

次の表に、DRITA によって表示される基本的な待機イベントを示します。

イベント名	説明
add in shmem lock acquire	廃止/未使用
bgwriter communication lock acquire	bgwriter (バックグラウンドライタ) プロセスは、bgwriter とバックエンドプロセスの間でメッセージを同期する短期ロックを待機しました。
btree vacuum lock acquire	サーバーは、次に利用可能なバキュームサイクル ID へのアクセスを同期する短期ロックを待機しました。
buffer free list lock acquire	サーバーは、(共有メモリ内の) 空きバッファのリストへのアクセスを同期する短期間のロックを待機しています。
checkpoint lock acquire	サーバープロセスは、同時チェックポイントを防止する短期間のロックを待機しています。
checkpoint start lock acquire	サーバーは、bgwriter チェックポイントスケジュールへのアクセスを同期する短期ロックを待機しています。
clog control lock acquire	サーバーは、コミットログへのアクセスを同期する短期間のロックを待機しています。
control file lock acquire	サーバーは、制御ファイルへの書き込みアクセスを同期する短期的なロックを待機しました (これは通常、小さい数値である必要があります)。
db file extend	サーバープロセスは、ファイルの末尾に新しいページを追加している間にオペレーティングシステムを待機しました。
db file read	サーバープロセスが (ディスクからの) 読み取りの完了を待機しました。
db file write	サーバープロセスが (ディスクへの) 書き込みの完了を待機しました。
db file sync	サーバープロセスは、オペレーティングシステムがすべての変更をディスクにフラッシュするのを待っています。
first buf mapping lock acquire	サーバーは、共有バッファマッピングテーブルへのアクセスを同期する短期間のロックを待機しています。
freespace lock acquire	サーバーは、空き領域マップへのアクセスを同期する短期ロックを待機しています。
lwlock acquire	サーバーは、このセクションの他の場所で説明されていない短期ロックを待機しています。
multi xact gen lock acquire	サーバーは、次の使用可能なマルチトランザクション ID へのアクセスを同期する短期的なロックを待機しています (SELECT...FORSHARE ステートメントが実行されるとき)。

次のページに続く

表 1 – 前のページからの続き

イベント名	説明
multi xact member lock acquire	サーバーは、マルチトランザクションメンバーファイルへのアクセスを同期する短期的なロックを待機しています (SELECT...FORSHARE ステートメントが実行されるとき)。
multi xact offset lock acquire	サーバーは、マルチトランザクションオフセットファイルへのアクセスを同期する短期的なロックを待機しています (SELECT...FORSHARE ステートメントの実行時)。
oid gen lock acquire	サーバーは、次に利用可能な OID (オブジェクト ID) へのアクセスを同期する短期ロックを待機しています。
query plan	サーバーは、SQL ステートメントの実行計画を計算しました。
rel cache init lock acquire	サーバーは、リレーションキャッシュの同時ロード/アンロードを防ぐ短期ロックを待機しています。
shmem index lock acquire	サーバーは、共有メモリマップへのアクセスを同期する短期的なロックを待機しています。
sinval lock acquire	サーバーは、キャッシュの無効化状態へのアクセスを同期する短期ロックを待機しています。
sql parse	サーバーは SQL ステートメントを解析しました。
subtrans control lock acquire	サーバーは、サブトランザクションログへのアクセスを同期する短期間のロックを待機しています。
tablespace create lock acquire	サーバーは、 CREATETABLESPACE または DROPTABLESPACE コマンドが同時に実行されるのを防ぐ短期ロックを待機しています。
two phase state lock acquire	サーバーは、準備されたトランザクションのリストへのアクセスを同期する短期ロックを待機しました。
wal insert lock acquire	サーバーは、先書きログへの書き込みアクセスを同期する短期ロックを待機しています。数値が大きい場合、WAL バッファのサイズが小さすぎることを示している可能性があります。
wal write lock acquire	サーバーは、先行書き込みログのフラッシュを同期する短期ロックを待機しています。
wal file sync	サーバーは、先行書き込みログがディスクに同期するのを待機しました (デフォルトでは「fsync」である wal_sync_method パラメーターに関連しています。このパラメーターを open_sync に変更すると、パフォーマンスが向上します)。
wal flush	サーバーは、先行書き込みログがディスクにフラッシュされるのを待っています。
wal write	サーバーは、先行書き込みログバッファへの書き込みを待機しました (この値が高いと予想されます)。
xid gen lock acquire	サーバーは、次の使用可能なトランザクション ID へのアクセスを同期する短期ロックを待機しています。

軽量ロックで待機イベントが発生すると、DRITA によっても表示されます。軽量ロック は、共有メモリ内の特定のデータ構造を保護するために使用されます。

特定の待機イベントは、関連する軽量ロックのグループの 1 つを待機しているサーバープロセスが原因である可能性があり、これは*軽量ロックトランシェ*と呼ばれます。個々の軽量ロックトランシェは DRITA では表示されませんが、それらの合計は other lwlock acquire という名前の単一のイベントで表示されます。

DRITA によって表示される軽量ロックのリストと説明については、次の場所にある PostgreSQL コアドキュメントのセクション 28.2、The Statistics Collector を参照してください。

<https://www.postgresql.org/docs/current/static/monitoring-stats.html>

セクション 28.2.2 の下。Viewing Statistics、軽量ロックは表 28-4 にリストされています wait_event Wait Event Type 列が LWLock を示す説明。

次の例では、軽量ロック ProcArrayLock、CLogControlLock、WALBufMappingLock、XidGenLock を表示しています。

```
postgres=# select * from sys_rpt(40,70,20);
               sys_rpt
```

WAIT NAME	COUNT	WAIT TIME	% WAIT
wal flush	56107	44.456494	47.65
db file read	66123	19.543968	20.95
wal write	32886	12.780866	13.70
wal file sync	32933	11.792972	12.64
query plan	223576	4.539186	4.87
db file extend	2339	0.087038	0.09
other lwlock acquire	402	0.066591	0.07
ProcArrayLock	135	0.012942	0.01
CLogControlLock	212	0.010333	0.01
WALBufMappingLock	47	0.006068	0.01
XidGenLock	53	0.005296	0.01
(13 rows)			

DRITA は、特定の AdvancedServer 製品機能に関連して発生する待機イベントも表示します。

これらの AdvancedServer 機能固有の待機イベントと other lwlock acquire イベントを次の表にリストします。

イベント名	説明
BulkLoadLock	サーバーは、EDB*Loader に関連するアクセスを待機しています。
EDBResoureManagerLock	サーバーは、EDBResourceManager に関連するアクセスを待機しています。
other lwlock acquire	軽量ロックトランシェの待機の合計。

第 5 章

Conclusion

EDBPostgresAdvancedServerDatabaseCompatibilityforOracleDevelopersToolsandUtilitiesGuide

著作権 ©2007-2020EnterpriseDBCorporation。

無断複写・転載を禁じます。

EnterpriseDB®Corporation

34CrosbyDrive、Suite201、Bedford、MA01730、アメリカ

T+17813573390F+19784671307E

info@enterprisedb.com

www.enterprisedb.com

- EnterpriseDB および PostgresEnterpriseManager は、EnterpriseDBCorporation の登録商標です。EDB および EDBPostgres は、EnterpriseDBCorporation の商標です。Oracle は、Oracle、Inc の登録商標です。その他の商標は、それぞれの所有者の商標です。
- EDB は、存在する場合、EDBPostgres 製品のログオン UI の設計、コーディングのベストプラクティスの確立、レビュー、および入力検証の検証を行います。EDB は追加の入力コンポーネントに対して同じアプローチに従いますが、製品の性質上、限定された検証が可能な信頼できるユーザーによって入力および送信されるフリーフォーム SQL、WMI またはその他の文字列を受け入れる必要がある場合があります。このような場合、ユーザーが誤った入力や危険な入力を禁止することはできません。
- EDB は、将来、認証された信頼できるユーザーからのフリーフォーム SQL、WMI またはその他の潜在的に危険な入力を受け入れる製品に機能を追加する権利を留保しますが、そのような機能はすべて、可能な限り最小のリスクを提供するように設計およびテストされます可能であれば、スーパーユーザーまたは同等の特権が必要です。
- EDB は、すべての潜在的な脅威を予測できることを保証しないため、すべての潜在的な脆弱性が対処または検討されたことを完全に保証することはできません。

索引

Building the EDB*Loader Control File, 7

Conclusion, 89

Configuring and Using DRITA, 51

Data Loading Methods, 5

Direct Path Load, 37

DRITA Functions, 53

EDB Loader Control File Examples, 19

EDB*Loader, 3

edbreport(), 63

Event Descriptions, 85

Exit Codes, 36

General Usage, 6

get_snaps(), 53

Introduction, 1

Invoking EDB*Loader, 29

Parallel Direct Path Load, 38

Performance Tuning Recommendations, 84

purgesnap(), 61

Remote Loading, 42

sess_rpt(), 54

sesshist_rpt(), 57

sessid_rpt(), 56

Simulating Statspack AWR Reports, 63

stat_db_rpt(), 73

stat_indexes_rpt(), 79

stat_tables_rpt(), 74

statio_indexes_rpt(), 81

statio_tables_rpt(), 77

sys_rpt(), 53

truncsnap(), 62

Updating a Table with a Conventional Path Load, 43

Using EDB*Wrap to Obfuscate Source Code, 46