2014 年 7 月 16 日 作成
2015 年 3 月 16 日 改定

PostgreSQL Internals (2)

日本ヒューレット・パッカード株式会社
篠田典良

謝辞

本資料の作成と公開にあたり、永安悟史様（アップタイム・テクノロジーズ合同会社)、渡部亮太様（株式会社コーソル）にレビューいただきました。アドバイスありがとうございました。

日本ヒューレット・パッカード株式会社の社内では高橋智雄さん、北山貴広さん、竹島彰子さん（いずれもテクノロジー事業統括 サービス統括本部）にレビューいただきました。ありがとうございます。

更新版の作成にあたり、永安悟史様、渡部亮太様、高橋智雄さん、北山貴広さん、竹島彰子さんにあらためてご意見をいただきました。ありがとうございます。

オープンソース製品を開発するすべてのエンジニアに感謝します。本文書が少しでもPostgreSQL を利用する皆様の役に立ちますように。

2014 年 7 月 16 日 作成
2015 年 3 月 4 日 改定
篠田典良

目次

用語集

表 1　略語／用語

略語/用語	説明
contrib モジュール	PostgreSQL の拡張モジュールを指す。標準で使用できる contrib モジュールの一覧はマニュアル「Appendix F. Additional Supplied Modules[1]」に掲載されている。
GUC	PostgreSQL のパラメータが保存されるメモリー領域（Global Unified Configuration）
JDBC	Java アプリケーションから RDBMS を利用するための標準 API とクラス
LSN	Log Serial Number を指す。仮想的なトランザクション・ログの書き込み位置を示す。
OID（オブジェクト ID）	データベース内部で作成されるオブジェクトを識別する ID で、符号なし 32 ビット値を持つ。
PL/pgSQL	PostgreSQL のストアド・プロシージャ記述言語のひとつ Oracle Database の PL/SQL とある程度互換性がある。
PostgreSQL	オープンソースデータベース製品
psql	PostgreSQL に付属する SQL 文を実行するためのユーティリティ
TID (Tuple ID)	テーブル内のレコードを一意に示す ID。レコードの物理位置を示す。
WAL	PostgreSQL のトランザクション・ログ（Write Ahead Logging）ファイル
XID（トランザクション ID）	トランザクションを一意に識別する ID、レコードの新旧を識別する符号なし 32 ビット値
アーカイブログ	リカバリに使用される WAL のコピー
システム・カタログ	PostgreSQL データベース全体のメタ情報を格納している領域
タプル	テーブル内のレコードを示す

[1] http://www.postgresql.org/docs/9.4/static/contrib.html

表 1（続）　略語／用語

略語/用語	説明
データベース・クラスタ	PostgreSQL データベース全体の管理情報が格納されているディレクトリ
リレーション	テーブルをリレーションと呼ぶ場合がある
テーブル空間	オブジェクトが格納されるファイルシステム上のディレクトリ。表領域と呼ばれる場合もある。

1. 本文書について

1.1 本文書の目的

本文書は PostgreSQL を利用するエンジニア向けに、PostgreSQL 周辺ソフトウェアの内部構造やマニュアルに記載されていない動作に関する知識を提供することを目的としています。

1.2 本文書の対象読者

本文書は、既にある程度 PostgreSQL に関する知識を持っているエンジニア向けに記述しています。インストール、基本的な管理等は実施できることを前提としています。

1.3 本文書の範囲

本文書の記述範囲は PostgreSQL にアクセスする JDBC Driver、contrib モジュール、および頻繁に使用される PostgreSQL 追加モジュールの一部を範囲としています。PostgreSQL 本体の内部構造については本文書の姉妹編である「PostgreSQL Internals (1)」を参照してください。内容については作成者が独習用に調査した結果をまとめた資料であるため、技術レベルや網羅性にばらつきがあります。

1.4 本文書の対応バージョン

本文書は原則として以下のバージョンを対象としています。

表 2 対象バージョン

種別	バージョン	備考
データベース	PostgreSQL 9.4	9.4.0
オペレーティング・システム	Red Hat Enterprise Linux 6 Update 5 (x86-64)	2.6.32-431
Java Runtime	OpenJDK Runtime Environment 1.7.0_45	
JDBC Driver	PostgreSQL JDBC Driver 9.4	9.41200 jdbc41
pg_hint_plan	1.1.3	
pg_rman	1.2.11	

1.5 本文書の更新

本文書は要望があれば更新する予定ですが、時期や更新内容は決定していません。

1.6 本文書に対する質問・意見および責任

本文書の内容は日本ヒューレット・パッカード株式会社の公式見解ではありません。また内容の間違いにより生じた問題について作成者および所属会社は責任を負いません。本文書に対するご意見、ご感想等については日本ヒューレット・パッカード株式会社 テクノロジーコンサルティング事業統括 篠田典良（noriyoshi.shinoda@hp.com）までお知らせください。

1.7 表記

1.7.1 表記の変換

中括弧（{}）で囲まれた部分は、何等かの文字列に変換されることを示しています。以下の表記を使用します。

表 3 表記

表記	説明	例
{HOSTNAME}	ホスト名	dbhost1
{OID}	任意の OID 番号	12993
{INSTALL}	PostgreSQL インストール・ディレクトリ	/opt/PostgreSQL/9.4
{PGDATA}	データベース・クラスタ用ディレクトリ	/opt/PostgreSQL/9.4/data
{PGDATABASE}	接続先ディレクトリ名	postgres
{PORT}	接続待ちポート番号	5432
{SPACE}	空白（0x20）の入力	
{SQL}	任意の SQL 文	SELECT * FROM table1
{TAB}	タブ（0x09）の入力	
{PASSWORD}	表示されないパスワードの入力	secret
${文字列}	環境変数が展開されることを示す	${PGDATA}

1.7.2 例の表記

本文書内にはコマンドや SQL 文の実行例が含まれます。例は以下の表記で記載しています。

表 4 例の表記

表記	説明	備考
#	Linux root ユーザーのプロンプト	
$	Linux 一般ユーザーのプロンプト	
太字	ユーザーが入力する文字列	
postgres=#	PostgreSQL 管理者が利用する psql プロンプト	
postgres=>	PostgreSQL 一般ユーザーが利用する psql プロンプト	

2. JDBC Driver

Java アプリケーションから PostgreSQL データベースに対してアクセスする標準として JDBC Driver を使う方法があります。ここでは JDBC Driver を使用する上で注意が必要な点を記述しています。

2.1 Connection オブジェクトの使用方法

PostgreSQL インスタンスに対して Connection オブジェクトを使って接続を行う場合、いくつかの接続プロパティを指定することができます。

2.1.1 接続プロパティの設定方法

java.sql.DriverManager クラスの getConnection メソッドには、データベース接続のために必要な情報として url（String）パラメータを指定します。url パラメータは以下の形式の文字列です。ホスト名を省略した場合はローカルホストを、ポート番号を省略した場合は 5432 が使用されます。JDBC Driver 9.4 では複数ホスト名を記述できるようになりました。検証結果は後述します。

例 1 URL 指定形式

```
jdbc:postgresql:{PGDATABASE}
jdbc:postgresql://{HOSTNAME}/{PGDATABASE}
jdbc:postgresql://{HOSTNAME}:{PORT}/{PGDATABASE}
```

□ url パラメータによるプロパティ設定

上記 url パラメータには追加情報としてプロパティを指定することができます。データベース名の後にクエスチョン・マーク（?）を指定し、「プロパティ名=値」の形式で記述されるセットをアンパーサンド（&）で結合します。下記は user, password, ssl プロパティを指定した例です。

例 2 プロパティ指定

```
jdbc:postgresql:appldb1?user=postgres&password=secret&ssl=true
```

プロパティ名と値のセットを java.util.Properties オブジェクトに格納して接続時に指定することもできます。間違ったプロパティ名やプロパティ値を指定しても接続エラーは発生しません。プロパティ値が認識できない場合はクライアント側のログに警告が出力されます。また url に指定する要素はすべて大文字／小文字を区別します。

□　ファイルによるプロパティ設定

プロパティは特定のファイル内に記述することもできます。環境変数 CLASSPATH に含まれるディレクトリ以下に「org/postgresql/driverconfig.properties」ファイルを配置し、ファイル内の各行に「プロパティ名=値」の形式でプロパティを記述します。ただしこのファイルはインスタンスに対する接続後に参照されるため、ssl プロパティのような接続時に使用されるプロパティは記述できません。

2.1.2 接続プロパティの一覧

URL には以下のプロパティを指定することができます。

表 5　プロパティの一覧（オンライン・マニュアルに記載されている）

プロパティ	説明	デフォルト値
user	接続ユーザー名	OS ユーザー名[2]
password	接続パスワード	-
ssl	SSL 接続を使用するか	false
sslfactory	SSL 接続情報	-
sslfactoryarg	SSL 接続パラメータ	-
compatible	互換性情報	9.4
sendBufferSize	送信バッファ・サイズ	-1
receiveBufferSize	受信バッファ・サイズ	-1
protocolVersion	接続プロトコルのバージョン	0
loglevel	ログ出力レベル	0
charSet	文字コード	UTF-8
allowEncodingChanges	エンコードの変更を許可	false
logUnclosedConnections	閉じられていない Connection をログ	false
prepareThreshold	Server Prepare Statement を使用する閾値	5
loginTimeout	ログイン・タイムアウト	0
socketTimeout	ソケット読み込みタイムアウト	0
tcpKeepAlive	Keep Alive の使用	false
unknownLength	不定サイズ列の長さ指定	2147483647
stringtype	文字列のデータ型指定	varchar
kerberosServerName	GSSAPI 認証サーバー名	-
jaasApplicationName	JAAS システム名	-
currentSchema[3]	デフォルト・スキーマ名	null
targetServerType	接続先インスタンスのタイプ	any
hostRecheckSeconds	ホストの再チェック間隔	10
loadBalanceHosts	ロードバランスを行うか	false

[2] System.getProperty("user.name")の値が使用されます。

[3] currentSchema, targetServerType, loadBalanceHosts, hostRecheckSeconds プロパティは PostgreSQL JDBC Driver 9.4 から追加されました。

表 5（続）プロパティの一覧（オンライン・マニュアルに記載されている）

プロパティ	説明	デフォルト値
gsslib	Java GSS ライブラリ関連	auto
sspiServiceClass	SSPI 認証関連？	POSTGRES
ApplicationName	アプリケーション名	-
binaryTransferEnable	バイナリ転送有効データ型	
binaryTransferDisable	バイナリ転送無効データ型	-
connectTimeout	接続タイムアウト	0
useSpnego	SPNEGO を使用	false
readOnly	読み取り専用	false
disableColumnSanitiser	列のサニタイザーを無効にする	false
assumeMinServerVersion	接続可能な最低バージョン	-

以下のプロパティはオンライン・マニュアルには記載されていませんが、ソースコード上には実装されています。一部を除いて動作の検証は行っていません。

表 6　プロパティの一覧（オンライン・マニュアルに記載されていない）

プロパティ	説明	デフォルト値
binaryTransfer	バイナリ転送の有効化	true
sslmode	SSL モード	-
sslcert	SSL 証明	-
sslkey	SSL キー	-
sslpasswordcallback	SSL パスワード・コールバック	-
sslpassword	SSL パスワード	-
sslrootcert	SSL ルート証明	-
sslhostnameverifier	SSL ホスト名	-

2.1.3 prepareThreshold プロパティ

prepareThreshold プロパティは、同一 SQL 文を複数回実行した場合に自動的に Server Prepare Statement（PREPARE 文）を使用する閾値を指定します。デフォルト値は 5 です。このため 5 回目の SQL 文実行から Server Prepare Statement が使用されます。
設定は以下の方法で変更できます。

- 接続 URL に指定する
- Connection オブジェクトから PGConnection オブジェクトを取得し、setPrepareThreshold メソッドで指定する。
- Statement オブジェクトから PGStatement オブジェクトを取得し、setPrepareThreshold メソッドで指定する。

prepareThreshold プロパティで指定した SQL 文の実行回数は Statement オブジェクト単位で計算されます。このため同一の SQL 文を異なる Statement オブジェクトで実行しても Server Prepare Statement は使用されません。

以下の例は、単一の PreparedStatement オブジェクトのバインド変数を変更しながら 5 回実行しています。prepareThreshold プロパティを 3 に指定し、PostgreSQL インスタンスの log_statement パラメータを all に設定した場合のログです。

例 3 prepareThreshold 実行時のサーバー・ログ

```
LOG:  execute <unnamed>: SELECT COUNT(*) FROM bind1 WHERE c2=$1
DETAIL:  parameters: $1 = 'maj'
LOG:  execute <unnamed>: SELECT COUNT(*) FROM bind1 WHERE c2=$1
DETAIL:  parameters: $1 = 'min'
LOG:  execute S_1: SELECT COUNT(*) FROM bind1 WHERE c2=$1
DETAIL:  parameters: $1 = 'maj'
LOG:  execute S_1: SELECT COUNT(*) FROM bind1 WHERE c2=$1
DETAIL:  parameters: $1 = 'min'
LOG:  execute S_1: SELECT COUNT(*) FROM bind1 WHERE c2=$1
DETAIL:  parameters: $1 = 'maj'
```

3 回目の LOG:から「<unnamed>」部分が「S_1」に変化して、Server Prepared Statement が利用されていることがわかります。

2.1.4 loglevel プロパティ

loglevel プロパティはクライアント側のログ出力のレベルを指定します。デフォルト値は 0 で、ログ出力を行いません。有効な値は 1（=org.postgresql.Driver.INFO）、2（=org.postgresql.Driver.DEBUG）です。2 以上の整数を指定してもエラーは発生せず、2 と見なされます。整数値以外の値[4]を指定すると org.postgresql.util.PSQLException 例外が発生します。

loglevel=2 を指定すると JDBC Driver を使った各種メソッドの実行単位で詳細なログが

[4] Integer#parseInt メソッドが例外を発生させる場合。

出力されます。以下の例は特定のメソッドを実行した場合に出力されるログです。

例 4　DriverManager#getConnection メソッド実行時

```
07:35:33.535 (1) PostgreSQL 9.4 JDBC4.1 (build 1200)
07:35:33.544 (1) Trying to establish a protocol version 3 connection to localhost:5432
07:35:33.561 (1) Receive Buffer Size is 43690
07:35:33.561 (1) Send Buffer Size is 84580
07:35:33.566 (1) FE=> StartupPacket(user=demo, database=demodb, client_encoding=UTF8, DateStyle=ISO, TimeZone=Asia/Tokyo, extra_float_digits=2)
07:35:33.570 (1) <=BE AuthenticationOk
07:35:33.581 (1) <=BE ParameterStatus(application_name = )
07:35:33.581 (1) <=BE ParameterStatus(client_encoding = UTF8)
<<途中省略>>
07:35:33.594 (1) <=BE ReadyForQuery(I)
07:35:33.595 (1) compatible = 90400
07:35:33.595 (1) loglevel = 2
07:35:33.595 (1) prepare threshold = 5
07:35:33.599 (1) types using binary send = INT8_ARRAY,TIMESTAMPTZ,FLOAT4_ARRAY,FLOAT8_ARRAY,UUID,TEXT_ARRAY,VARCHAR_ARRAY,BYTEA,TIME,FLOAT4,FLOAT8,INT2_ARRAY,TIMETZ,INT2,INT8,INT4,INT4_ARRAY,TIMESTAMP,POINT,BOX
07:35:33.600 (1) types using binary receive = INT8_ARRAY,TIMESTAMPTZ,FLOAT4_ARRAY,FLOAT8_ARRAY,UUID,TEXT_ARRAY,VARCHAR_ARRAY,BYTEA,TIME,DATE,FLOAT4,FLOAT8,INT2_ARRAY,TIMETZ,INT2,INT8,INT4,INT4_ARRAY,TIMESTAMP,POINT,BOX
07:35:33.601 (1) integer date/time = true
getConnection returning org.postgresql.Driver
```

例 5　PreparedStatement#executeQuery メソッド実行時

```
07:35:33.623                (1)                simple                execute,
handler=org.postgresql.jdbc2.AbstractJdbc2Statement$StatementResultHandler@566
fbd76, maxRows=0, fetchSize=0, flags=17
07:35:33.623 (1)  FE=> Parse(stmt=null,query="SELECT COUNT(*) FROM bind1 WHERE
c2=$1",oids={1043})
07:35:33.623 (1)  FE=> Bind(stmt=null,portal=null,$1=<'maj'>)
07:35:33.624 (1)  FE=> Describe(portal=null)
07:35:33.624 (1)  FE=> Execute(portal=null,limit=0)
07:35:33.624 (1)  FE=> Sync
07:35:33.645 (1)  <=BE ParseComplete [null]
07:35:33.645 (1)  <=BE BindComplete [null]
07:35:33.646 (1)  <=BE RowDescription(1)
07:35:33.646 (1)        Field(,INT8,8,T)
07:35:33.646 (1)  <=BE DataRow(len=5)
07:35:33.646 (1)  <=BE CommandStatus(SELECT 1)
07:35:33.656 (1)  <=BE ReadyForQuery(I)
```

2.1.5 readOnly プロパティ

readOnly プロパティは JDBC Driver 9.3 から利用できましたが、ドキュメントには記述がありませんでした。JDBC Driver 9.4 からマニュアルに記載されました。boolean 値を取り、デフォルト値は false です。true を指定すると更新系の SQL 文を実行すると以下の例外が発生します。

例 6　readOnly=true 指定時の例外（Exception#printStackTrace メソッドで出力）

```
Exception in thread "main" org.postgresql.util.PSQLException: ERROR: cannot execute INSERT in a read-only
transaction
        at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2270)
        at org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:1998)
        at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:255)
        at org.postgresql.jdbc2.AbstractJdbc2Statement.execute(AbstractJdbc2Statement.java:570)
        at
org.postgresql.jdbc2.AbstractJdbc2Statement.executeWithFlags(AbstractJdbc2Statement.java:406)
        at org.postgresql.jdbc2.AbstractJdbc2Statement.execute(AbstractJdbc2Statement.java:398)
        at bind1a.main(bind1a.java:39)
```

2.1.6 sendBufferSize, receiveBufferSize プロパティ

ソケットの送信バッファと受信バッファのサイズを指定します。デフォルト値は-1 で、システム・デフォルトを使用します。最大値は以下のカーネル・パラメータに依存します。カーネル・パラメータの値よりも大きい値を指定した場合でもエラーは発生せず、カーネル・パラメータの値が使用されます。

表 7　sendBufferSize, receiveBufferSize プロパティの制限

プロパティ	上限を決めるカーネル・パラメータ	上限のデフォルト値
sendBufferSize	net.core.wmem_max	124,928
receiveBufferSize	net.core.rmem_max	124,928

公式なマニュアル（http://jdbc.postgresql.org/documentation/94/connect.html）には一部プロパティ名として recvBufferSize と示されていますが、実装は receiveBufferSize です。

Red Hat Enterprise Linux 6.5 / OpenJDK 1.7.0 環境ではプロパティのデフォルト値の検証結果は以下の通りです。

表 8　sendBufferSize, receiveBufferSize プロパティのデフォルト

プロパティ	デフォルト値	備考
sendBufferSize	84,580	
receiveBufferSize	43,690	

□　パラメータ値の確認

設定値を確認するためには、loglevel プロパティを 1 または 2 に設定します。以下のログが出力されます。

例 7　sendBufferSize / receiveBufferSize プロパティの確認

```
06:52:25.814 (1) PostgreSQL 9.4 JDBC4.1 (build 1200)
06:52:25.823 (1) Trying to establish a protocol version 3 connection to localhost:5432
06:52:25.840 (1) Receive Buffer Size is 43690
06:52:25.840 (1) Send Buffer Size is 84580
```

2.1.7 ApplicationName プロパティ

ApplicationName プロパティは JDBC Driver 9.4 からマニュアルには記述が追加されました。デフォルト値は空文字です。プロパティに文字列を指定すると、pg_stat_activity カタログの application_name 列に指定値が表示されるようになります。

2.1.8 ssl プロパティ

ssl プロパティを true に指定するとクライアントとサーバー間の通信に SSL が使用されるようになります。PostgreSQL インスタンスが SSL に対応していない場合、接続は失敗します。

例 8　サーバーが SSL に対応していないエラー（Exception）

```
Exception in thread "main" org.postgresql.util.PSQLException: The server does not
support SSL.
        at
org.postgresql.core.v3.ConnectionFactoryImpl.enableSSL(ConnectionFactoryImpl.j
ava:322)
<<以下省略>>
```

2.1.9 stringtype プロパティ

PrepareStatement#setString メソッドのデータ型を指定します。指定できる値は varchar（デフォルト値）または unspecified です。パラメータ値は大文字／小文字を意識しません。varchar または unspecified 以外の値を指定すると、以下の例外が発生します。

例 9　stringtype プロパティ値のエラー（Exception）

```
org.postgresql.util.PSQLException: Unsupported value for stringtype parameter:
BADVALUE
```

2.1.10 logUnclosedConnections プロパティ

デフォルト値は false です。true に設定すると Java VM の GC により finalize メソッドがコールされた場合、close メソッドが呼ばれていない Connection メソッドの情報をログに出力します。以下のログが出力されますが、動作は未確認です。

例 10　logUnclosedConnections プロパティ関連ログ

```
Finalizing a Connection that was never closed:
```

2.1.11 disableColumnSanitiser プロパティ

このプロパティは JDBC Driver 9.4 からオンライン・ドキュメントに記述が追加されました。boolean 値を取り、デフォルト値は false です。

ResultSet#get*(String) メソッドは、内部的に列名から列番号へ変換するハッシュ・テーブルを作成します。標準では列名を小文字に変換してハッシュ・テーブルを作成しています。disableColumnSanitiser プロパティを true に設定すると、列名の小文字変換の処理をスキップします。テーブルに、大文字／小文字のみ異なる列名が存在する場合に影響を受ける場合があります。

2.1.12 unknownLength プロパティ

text 型等、サイズが決まっていない列のデータ型に対して ResultSetMetaData#getColumnDisplaySize メソッドや、ResultSetMetaData#getPrecision メソッド実行時に返る値を返します。デフォルト値は Integer.MAX_VALUE（2,147,483,647）です。

数値は整数であればマイナスの値でも受け付けますが、整数以外を指定すると以下のエラーが発生します。

例 11　unknownLength プロパティ値のエラー（Exception）

```
org.postgresql.util.PSQLException: unknownLength parameter value must be an integer
```

2.1.13 currentSchema プロパティ

currentSchema プロパティはデフォルトのスキーマ名を設定します。PostgreSQL JDBC Driver 9.4 で追加されました。このプロパティを指定すると、パラメータ search_path は無効になります。このため search_path に含まれるスキーマ内のテーブルにアクセスする SQL には、スキーマ名の修飾が必要になります。

2.2 DataSource オブジェクトの使用方法

JDBC Driver を使って PostgreSQL インスタンスに接続する方法として DriverManager オブジェクトを使用する方法と、DataSource オブジェクトを使用する方法があります。ここでは DataSource オブジェクトを使用する方法について説明しています。

2.2.1 DataSource オブジェクトの実装

PostgreSQL JDBC Driver には複数の DataSource オブジェクトが提供されています。接続プールの機能によって以下の 2 オブジェクトを使うことができます。

表 9　DataSource オブジェクトの実装

クラス名	接続プール機能	備考
org.postgresql.ds.PGSimpleDataSource	なし	
org.postgresql.ds.PGPoolingDataSource	あり	

2.2.2 接続プロパティ

オンライン・マニュアルには一部のプロパティのみが記載されています。DataSource オブジェクトに対しても DriverManager オブジェクトと同様のプロパティを指定することができます。プロパティ名を羅列した URL は setURL メソッドで指定することができます。基本的に DriverManager オブジェクトで指定する URL と、DataSource オブジェクトで指定する URL は同じですが、いくつかのプロパティが異なります。

> setURL メソッドは PostgreSQL 9.3 用の JDBC Driver である postgresql-9.3-1103.jdbc41.jar ファイルではエラーにならなかった構文が postgresql-9.4-1200.jdbc41.jar ファイルでは java.lang.NullPointerException 例外を発生してエラーになる場合があります。このバグは認識されており、更新版がリリースされる予定です。

2.2.3 接続プール

接続プール機能を持った PGPoolingDataSource オブジェクトには、DataSource オブジェクト名を指定する setDataSourceName メソッド、初期接続数を指定する setInitialConnections メソッドおよび最大接続数を指定する setMaxConnections メソッドが提供されています。

□　プロパティ初期値

initialConnections プロパティ、maxConnections プロパティの初期値は 0 です。実際には初期セッションは 1 になり、上限は無制限（インスタンスの max_connections パラメータに依存）となります。

どちらのプロパティも最初の getConnection メソッドが実行され、PostgreSQL インスタンスとの接続が発生した後では変更できません。変更しようとすると以下の例外が発生します。

例 12　DataSource 使用後にプロパティ変更時に発生する例外

```
Exception in thread "main" java.lang.IllegalStateException: Cannot set Data Source
properties after DataSource has been used
        at
org.postgresql.ds.jdbc23.AbstractJdbc23PoolingDataSource.setMaxConnections(Abs
tractJdbc23PoolingDataSource.java:222)
```

□　初期接続の実際の開始契機

実際に PostgreSQL インスタンスとの接続が開始されるのは、最初に DataSource#getConnection メソッドが発生した時点です。最初の接続時に initialConnections プロパティで指定された数のセッションが作成されます。

□　最大接続以上に Connection オブジェクトを要求した場合

maxConnections プロパティで指定した数まで getConnection メソッドを実行すると、次回の getConnection メソッドは、使用中の Connection オブジェクトがクローズされるまで待機状態になります。

□　最大接続以上の初期接続

maxConnections プロパティの値以上の initialConnections を指定した場合、エラーは発生せず、initialConnections の値が有効になります。

□ セッションをクローズした場合

取得した Connection オブジェクトをクローズ（close メソッド）しても、物理セッションはクローズされません。

2.2.4 ログの出力

PGPoolingDataSource クラス、PGSimpleDataSource クラス共に loglevel プロパティが用意されており、Connection クラスと同様のログ設定を行うことができます。ログの出力先は DataSource#setLogWriter メソッドに PrintWriter オブジェクトを指定します。

出力されるログは Connection オブジェクトのログと同等です。以下は loglevel=2 を指定した場合のログ出力例です。

例 13　DataSource#getConnection メソッド実行時のログ

```
08:36:43.232 (1) PostgreSQL 9.4 JDBC4.1 (build 1200)
08:36:43.241 (1) Trying to establish a protocol version 3 connection to
localhost:5432
08:36:43.258 (1) Receive Buffer Size is 43690
08:36:43.258 (1) Send Buffer Size is 84580
08:36:43.260 (1) FE=> StartupPacket(user=demo, database=demodb,
client_encoding=UTF8, DateStyle=ISO, TimeZone=Asia/Tokyo, extra_float_digits=2)
08:36:43.264 (1) <=BE AuthenticationOk
08:36:43.274 (1) <=BE ParameterStatus(application_name = )
08:36:43.274 (1) <=BE ParameterStatus(client_encoding = UTF8)
<<途中省略>>
08:36:43.281 (1) <=BE ReadyForQuery(I)
08:36:43.282 (1)     compatible = 90400
08:36:43.282 (1)     loglevel = 2
08:36:43.282 (1)     prepare threshold = 5
08:36:43.286 (1)                    types using binary send =
INT8_ARRAY,TIMESTAMPTZ,FLOAT4_ARRAY,FLOAT8_ARRAY,UUID,TEXT_ARRAY,VARCHAR_ARRAY
,BYTEA,TIME,FLOAT4,FLOAT8,INT2_ARRAY,TIMETZ,INT2,INT8,INT4,INT4_ARRAY,TIMESTAM
P,POINT,BOX
08:36:43.288 (1)                    types using binary receive =
INT8_ARRAY,TIMESTAMPTZ,FLOAT4_ARRAY,FLOAT8_ARRAY,UUID,TEXT_ARRAY,VARCHAR_ARRAY
,BYTEA,TIME,DATE,FLOAT4,FLOAT8,INT2_ARRAY,TIMETZ,INT2,INT8,INT4,INT4_ARRAY,TIM
ESTAMP,POINT,BOX
08:36:43.288 (1)     integer date/time = true
getConnection returning org.postgresql.Driver
```

例 14　Connection#executeQuery メソッドに SELECT 文指定時のログ

```
08:36:43.313                    (1)                    simple                    execute,
handler=org.postgresql.jdbc2.AbstractJdbc2Statement$StatementResultHandler@145
1e300, maxRows=0, fetchSize=0, flags=17
08:36:43.313 (1)  FE=> Parse(stmt=null,query="SELECT COUNT(*) FROM bind1 WHERE
c2=$1",oids={1043})
08:36:43.313 (1)  FE=> Bind(stmt=null,portal=null,$1=<'maj'>)
08:36:43.314 (1)  FE=> Describe(portal=null)
08:36:43.314 (1)  FE=> Execute(portal=null,limit=0)
08:36:43.314 (1)  FE=> Sync
08:36:43.332 (1)  <=BE ParseComplete [null]
08:36:43.332 (1)  <=BE BindComplete [null]
08:36:43.333 (1)  <=BE RowDescription(1)
08:36:43.333 (1)         Field(,INT8,8,T)
08:36:43.334 (1)  <=BE DataRow(len=5)
08:36:43.334 (1)  <=BE CommandStatus(SELECT 1)
08:36:43.344 (1)  <=BE ReadyForQuery(I)
```

2.3 ロードバランス機能

PostgreSQL JDBC Driver 9.4 には接続時に複数のインスタンスを選択するロードバランス機能が提供されました。以下のプロパティを使用できます。

2.3.1 複数ホストの指定

従来のバージョンでは接続先インスタンスは url パラメータにホスト名とポート番号を一つだけ指定していました。PostgreSQL JDBC Driver 9.4 では複数のインスタンスを指定することができるようになりました。従来と同じ部分に複数のホストとポート番号をカンマ（,）で区切って指定します。

例 15 複数インスタンスの URL 指定形式

```
jdbc:postgresql://host1:5432, host2:5432/demodb
```

DriverManager#connect メソッドによる接続を行うと、標準状態では、先頭に指定されたインスタンスから順番に接続を試行します。すべてのホストの接続が失敗した場合には

例外が発生します。

2.3.2 loadBalanceHosts プロパティ

インスタンスに接続する際には、インスタンスを url パラメータに記述した順で接続します。loadBalanceHosts パラメータを true に設定すると、接続先インスタンスをランダムに選択することができます（デフォルト値は false）。true または false 以外の値を指定してもエラーにはなりません。

従来、このような機能を実現するには pgpool-II が必要でしたが、SELECT 文のみの場合には JDBC Driver の機能だけで負荷分散を実現することができるようになります。ただし更新 SQL の場合に自動的にマスター・インスタンスを使用することは行わないため、更新処理を含むアプリケーションでは pgpool-II の方が高機能になります。

2.3.3 targetServerType プロパティ

接続先インスタンスの種類を指定するパラメータが targetServerType です。以下の値を使用することができます。下記の表以外の値を指定すると、org.postgresql.util.PSQLException 例外が発生します。

接続先インスタンスの種類の選択は、接続先インスタンスで実行される SHOW TRANSACTION_READ_ONLY 文から判断しています。実行結果が ON (TRUE) の場合、スレーブ・インスタンスとみなします。

表 10 targetServerType プロパティの選択肢

設定値	説明	備考
any	すべての種類のインスタンスに接続	デフォルト値
master	レプリケーション環境のマスター・インスタンスにのみ接続します。マスター・インスタンスが存在しない場合は例外が発生します。	
slave	レプリケーション環境のスレーブ・インスタンスにのみ接続します。スレーブ・インスタンスが存在しない場合には例外が発生します。	
preferSlave	レプリケーション環境のスレーブ・インスタンスに接続します。スレーブ・インスタンスが存在しない場合にはマスター・インスタンスに接続します。	

2.3.4 hostRecheckSeconds プロパティ

インスタンスの稼働状態をキャッシュする秒数を指定します。デフォルト値は 10 秒です。プロパティに数値以外の値を指定した場合、org.postgresql.util.PSQLException 例外が発生します。

2.4 ResultSet の列名

ResultSet オブジェクトからデータを取得する「get<データ型>」メソッドには、列番号を指定するメソッドと列名を文字列として指定するメソッドが定義されています。以下の例では getString メソッドに列名を指定しています。

例 16　ResultSet オブジェクトから列名を指定したデータ取得

```
String sql = "SELECT column1, column2 FROM table1" ;
ResultSet rs = statement.executeQuery(sql) ;
String result = rs.getString("column1") ;
```

PostgreSQL の JDBC Driver は関数を指定した場合や、列の別名を指定した場合、SELECT で指定した列名と、「get<データ型>」で指定する列名が異なる場合があります。

下記のように、JDBC メソッドで指定する列名は、SELECT 文で指定された別名または列名を小文字化した値です。ダブル・クオーテーション（""）で囲んだ場合はそのまま使用できます。また列に対して関数を使った場合、メソッドに指定する列名は関数名のみになります。

表 11 ResultSet オブジェクトの列名

SELECT 文の列名	JDBC メソッド列名	変更される点
COLUMN1	column1	小文字化
Column2	column2	小文字化
SUM(*)	sum	小文字化／パラメータ削除
trim(column4) COL4	col4	別名を小文字化
trim(column5) AS COL5	col5	別名を小文字化
COUNT(*) AS col6	col6	別名を小文字化
column7 AS “COL7”	COL7	別名をそのまま

複数列に同じ関数名を指定した場合、最初の関数名が指定した列の値が帰ります。下記の例では、result 変数には「TRIM(column1)」列の値が返ります。

例 17　関数を指定した列からのデータ取得

```
String sql = "SELECT TRIM(column1), TRIM(column2) FROM table1" ;
ResultSet rs = statement.executeQuery(sql) ;
String result = rs.getString("trim") ;
```

2.5 Fetch Size の設定

Statement オブジェクトの setFetchSize メソッドは SELECT 文を実行した場合のキャッシュ・レコード数を取得します。

2.5.1 メモリー使用

Statement オブジェクトの setFetchSize メソッドは SELECT 文を実行した場合のキャッシュ・レコード数を設定します。この設定は自動コミット設定がオフの場合にのみ有効になります（Connection#setAutoCommit(false)を実行)。デフォルト値は 0 で、全レコードがキャッシュ対象になります。このため、デフォルト状態では検索結果の全レコードを Java VM ヒープ・メモリーに展開してから ResultSet#next メソッドが実行できるようになります。大規模なデータを検索する環境では検索のみの場合でも自動コミットをオフに設定してメモリー領域の肥大化を防ぐことを検討します。

2.5.2 実行計画の変化

Fetch Size を指定して SQL 文を実行すると、実行計画はパラメータ cursor_tuple_fraction で指定した割合に応じて実行計画を変更します（デフォルト値 0.1)。パラメータ値を小さくすると先頭レコードの取得が高速になる実行計画が選択されます（Nested Loop 結合等)。値を大きくするとレコード全体の取得時間を短くする実行計画が選択されます（Hash Join 等)。

パラメータを上限である 1.0 に指定すると、全レコードを意味します。下限の 0 に指定すると実際には 0.0000000001（1-e10）が使用されます。

2.6 未実装の機能

以下のメソッドは実装されていません。該当メソッドを実行すると、org.postgresql.Driver.notImplemented 例外が発生します。下記表の「パラメータ列」はメソッドのパラメータ型をカンマ（,）で区切っています。スラッシュ（/）はオーバーライドされた同名のメソッドのパラメータを示します。

表 12　未実装メソッド（Class: org.postgresql.Driver）

メソッド名	パラメータ	備考
getParentLogger	-	

表 13　未実装メソッド（Class: org.postgresql.core.v2.QueryExecutorImpl.java）

メソッド名	パラメータ	備考
fetch	ResultCursor, ResultHandler, int	

表 14　未実装メソッド（Class: org.postgresql.ds.jdbc4.AbstractJdbc4ConnectionPoolDataSource）

メソッド名	パラメータ	備考
getParentLogger	-	

表 15　未実装メソッド（Class: org.postgresql.ds.jdbc4.AbstractJdbc4PooledConnection）

メソッド名	パラメータ	備考
getParentLogger	-	

表 16　未実装メソッド（Class: org.postgresql.ds.jdbc4.AbstractJdbc4PoolingDataSource）

メソッド名	パラメータ	備考
getParentLogger	-	

表 17　未実装メソッド（Class: org.postgresql.ds.jdbc4.AbstractJdbc4SimpleDataSource）

メソッド名	パラメータ	備考
getParentLogger	-	

表 18　未実装メソッド（Class: org.postgresql.jdbc2.AbstractJdbc2Array）

メソッド名	パラメータ	備考
getArrayImpl	long, int, Map	条件あり

表 19　未実装メソッド（Class: org.postgresql.jdbc2.AbstractJdbc2Clob）

メソッド名	パラメータ	備考
position	String, long / Clob, long	

表 20　未実装メソッド（Class: org.postgresql.jdbc2.AbstractJdbc2ResultSet）

メソッド名	パラメータ	備考
getURL	int / String	
getObjectImpl	int, java.util.Map	条件あり
getRef	int	

表 21　未実装メソッド（Class: org.postgresql.jdbc2.AbstractJdbc2Statement）

メソッド名	パラメータ	備考
setRef	int, Ref	
getBlob	int	
getClob	int	
getObjectImpl	int, java.util.Map	条件あり
getRef	int	
registerOutParameter	int, int, String	

表 22　未実装メソッド（Class: org.postgresql.jdbc3.AbstractJdbc3Clob）

メソッド名	パラメータ	備考
setString	long, String / long, String, int, int	
setAsciiStream	long	
setCharacterStream	long	

表 23　未実装メソッド（Class: org.postgresql.jdbc3. AbstractJdbc3DatabaseMetaData）

メソッド名	パラメータ	備考
getSuperTypes	String, String, String	
getSuperTables	String, String, String	
getAttributes	String, String, String, String	

表 24　未実装メソッド（Class: org.postgresql.jdbc3.AbstractJdbc3ResultSet）

メソッド名	パラメータ	備考
getURL	int / String	
updateRef	int, java.sql.Ref / String, java.sql.Ref	
updateBlob	int, java.sql.Blob / String, java.sql.Blob	
updateClob	int, java.sql.Clob / String, java.sql.Clob	

表 25　未実装メソッド（Class: org.postgresql.jdbc3. AbstractJdbc3Statement）

メソッド名	パラメータ	備考
setURL	int, java.net.URL / String, java.net.URL	
registerOutParameter	String, int / String, int, int / String, int, String	
getURL	int	
setNull	String, int / String, int, String	
setBoolean	String, boolean	
setByte	String, byte	
setShort	String, short	
setInt	String, int	
setLong	String, long	
setFloat	String, float	
setDouble	String, double	
setBigDecimal	String, BigDecimal	
setString	String, String	
setBytes	String, byte[]	
setDate	String, java.sql.Date / String, java.sql.Date, Calendar	
setTime	String, java.sql.Time / String, java.sql.Time, Calendar	
setTimestamp	String, java.sql.Timestamp / String, java.sql.Timestamp, Calendar	
setAsciiStream	String, java.io.InputStream, int	
setBinaryStream	String, java.io.InputStream, int	
setObject	String, Object, int, int / String, Object, int / String, Object	
setCharacterStream	String, java.io.Reader, int	
getString	String	
getBoolean	String	
getByte	String	

表 25（続）未実装メソッド（Class: org.postgresql.jdbc3. AbstractJdbc3Statement）

メソッド名	パラメータ	備考
getShort	String	
getInt	String	
getLong	String	
getFloat	String	
getDouble	String	
getBytes	String	
getDate	String / String, Calendar	
getTime	String / String, Calendar	
getTimestamp	String / String, Calendar	
getObject	String	
getBigDecimal	String	
getObjectImpl	String, java.util.Map	
getRef	String	
getBlob	String	
getClob	String	
getArray	String	
getURL	String	

表 26　未実装メソッド（Class: org.postgresql.jdbc4.AbstractJdbc4Blob）

メソッド名	パラメータ	備考
getBinaryStream	long, long	

表 27　未実装メソッド（Class: org.postgresql.jdbc4.AbstractJdbc4Clob）

メソッド名	パラメータ	備考
getCharacterStream	long, long	

表 28　未実装メソッド（Class: org.postgresql.jdbc4.AbstractJdbc4Connection）

メソッド名	パラメータ	備考
createClob	-	
createBlob	-	
createNClob	-	
createStruct	String, Object[]	
createQueryObject	Class<T>	
setNetworkTimeout	Executor, int	
getNetworkTimeout	-	

表 29　未実装メソッド（Class: org.postgresql.jdbc4.AbstractJdbc4DatabaseMetaData）

メソッド名	パラメータ	備考
getRowIdLifetime	-	
getFunctions	String, String, String	
getFunctionColumns	String, String, String, String	
getPseudoColumns	String, String, String, String	

表 30　未実装メソッド（Class: org.postgresql.jdbc4.AbstractJdbc4ResultSet）

メソッド名	パラメータ	備考
getRowId	int	
updateRowId	int, RowId	
getHoldability	-	
updateNString	int, String	
updateNClob	int, NClob / int, Reader / int, Reader, long / String, NClob	
getNClob	int / String	
updateBlob	int, InputStream, long / String, InputStream, long / int, InputStream / String, InputStream	
updateClob	int, Reader, long / int, Reader / String, Reader, long / String, Reader	
getNString	int / String	
getNCharacterStream	int / String	
updateNCharacterStream	int, Reader, int / int, Reader / int, Reader, long / String, Reader, int / String, Reader / String, Reader, long	
updateCharacterStream	int, Reader, long/ int, Reader / String, Reader, long / String, Reader	
updateBinaryStream	int, InputStream, long / int, InputStream / String, InputStream, long / String, InputStream	
updateAsciiStream	int, InputStream, long / int, InputStream / String, InputStream, long / String, InputStream	
getObject	int, Class<T>	

表 31　未実装メソッド（Class: org.postgresql.jdbc4.AbstractJdbc4Statement）

メソッド名	パラメータ	備考
setRowId	int, RowId / String, RowId	
setNString	int, String / String, String	
setNCharacterStream	int, Reader, long / int, Reader / String, Reader, long / String, Reader	
setCharacterStream	int, Reader, long / int, Reader / String, Reader, long / String, Reader	
setAsciiStream	int, InputStream, long / int, InputStream / String, InputStream, long / String, InputStream	
setNClob	int, NCLob / int, Reader, long / int, Reader / String, NClob	
setClob	int, Reader, long / int, Reader / String, Reader, long / String, Reader / String, Clob	
setBlob	int, InputStream, long / int, InputStream / String, InputStream, long / String, InputStream / String, Blob	
setNClob	int, NClob, int, Reader, long / int, Reader / String, Reader, long / String, Reader	
getRowId	int / String	
setBinaryStream	int, InputStream / String, InputStream long / String, InputStream	
getNClob	int / String	
setSQLXML	String, SQLXML	
getSQLXML	String	
getNString	int / String	
getNCharacterStream	int / String	
getCharacterStream	int / String	
getParentLogger	-	
getObject	int, Class<T> / String, Class<T>	

表 32　未実装メソッド（Class: org.postgresql.xa.jdbc4.AbstractJdbc4XADataSource）

メソッド名	パラメータ	備考
getParentLogger	-	

2.7 ドライバー実装クラス

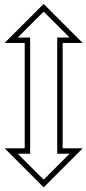
Java 7 / Java 8 環境では JDBC Driver として postgresql-9.4-1200.jdbc41.jar ファイルを使います。この JDBC ドライバーで定義されている JDBC インターフェースは実際には以下のクラスを使用します。

表 33　主なインターフェースとクラス

インターフェース	クラス	備考
Blob	org.postgresql.jdbc4.Jdbc4Blob	
Clob	org.postgresql.jdbc4.Jdbc4Clob	
Connection	org.postgresql.jdbc4.Jdbc4Connection	
DatabaseMetaData	org.postgresql.jdbc4.Jdbc4DatabaseMetaData	
PreparedStatement	org.postgresql.jdbc4.Jdbc4PreparedStatement	
ResultSet	org.postgresql.jdbc4.Jdbc4ResultSet	
ResultSetMetaData	org.postgresql.jdbc4.Jdbc4ResultSetMetaData	
Statement	org.postgresql.jdbc4.Jdbc4Statement	
SQLXML	org.postgresql.jdbc4.Jdbc4SQLXML	

最新の JDBC インターフェースを実装した各クラスは、古い Java バージョンのクラスを継承することで実装されています。例えば java.sql.Connection インターフェースを実装した org.postgresql.jdbc4. Jdbc4Connection クラスの継承関係は以下のようになっています。

表 34　Connection インターフェースの基底クラスと継承クラス

継承関係	クラス	備考
基底	org.postgresql.PGConnection	
	org.postgresql.core.BaseConnection	implements
	org.postgresql.jdbc2.AbstractJdbc2Connection	
	org.postgresql.jdbc3g.AbstractJdbc3gConnection	
	org.postgresql.jdbc4.AbstractJdbc4Connection	
継承	org.postgresql.jdbc4.Jdbc4Connection	

3. 追加モジュール

3.1 postgres_fdw

3.1.1 postgres_fdw とは

postgres_fdw は外部システムに対して SQL 文を投入する FOREIGN DATA WRAPPER に対応する contrib モジュール[5]で、リモート・ホストの PostgreSQL インスタンスに SQL 文を投入することができます。従来から PostgreSQL には、ネットワーク上に稼働する PostgreSQL インスタンスに SQL 文を投入する機能として dblink 関数（contrib モジュール）が提供されてきました。しかしこの関数は扱いにくいため、より簡単で洗練された実装として postgres_fdw モジュールが開発されました。PostgreSQL 9.3 からは FOREIGN DATA WRAPPER に対する書込みを行うことが可能になりました。

3.1.2 postgres_fdw の使用方法

postgres_fdw を使用するためには以下の手順で実行します。

- CREATE EXTENSION 文で postgres_fdw をロード
- CREATE SERVER 文でリモート・インスタンスを特定
- CREATE USER MAPPING 文でリモート・アクセスするユーザーを特定
- CREATE FOREIGN TABLE 文でリモート・テーブルを特定
- ローカル・テーブルと同様に DML を実行

例 18 postgres_fdw の使用方法

```
postgres=# CREATE EXTENSION postgres_fdw ;
CREATE EXTENSION
postgres=# CREATE SERVER remsvr FOREIGN DATA WRAPPER postgres_fdw
        OPTIONS (host 'remhost1', dbname 'dbappl1', port '5432') ;
CREATE SERVER
postgres=# CREATE USER MAPPING FOR public SERVER remsvr
        OPTIONS (user 'postgres', password 'secret');
CREATE USER MAPPING
postgres=# CREATE FOREIGN TABLE data1 (c1 NUMERIC, c2 VARCHAR(10))
        SERVER remsvr;
postgres=# SELECT * FROM data1 ;
```

[5] マニュアルは http://www.postgresql.org/docs/9.4/static/postgres-fdw.html

3.1.3 実行される SQL

リモート・インスタンスで SQL 文が投入されると、postgres_fdw 経由でリモート・インスタンスに SQL 文が送信され、結果が返されます。ここでは postgres_fdw が送信する SQL 文について検証しています。ユーザーが投入した SQL 文は以下のように変更されてリモート・インスタンスに送信されます。

- 列名の「*」は、列名リストに変換
- コメントは削除
- 予約語は大文字に変換
- 集計（SUM, COUNT 等）／グルーピング（GROUP BY）／順番（ORDER BY）／レコード制限（LIMIT）は削除
- 結合は２つの SELECT 文に分解

以下の例は、リモート・インスタンスに対して「SELECT * FROM data1」文を投入した場合に実行される SQL 文です。下記のように複数の SQL 文に分解されて実行されます。

- SART TRANSACTION ISOLATION LEVEL REPEATABLE READ
- DECLARE CURSOR
- FETCH
- CLOSE
- COMMIT TRANSACTION

例 19 実行される SQL 文の例

```
START TRANSACTION ISOLATION LEVEL REPEATABLE READ
DECLARE c1 CURSOR FOR
        SELECT c1, c2 FROM public.data1
FETCH 100 FROM c1
FETCH 100 FROM c1
..
CLOSE c1
COMMIT TRANSACTION
```

次の表は、投入した SQL 文と実行された SQL 文の対比です。START TRANSACTION 文、FETCH 文、CLOSE 文、COMMIT 文は省略しています。

表 35 ローカル投入 SQL とリモート実行 SQL（検索）

ローカル投入 SQL	リモート実行 SQL
select * from data1	DECLARE c1 CURSOR FOR SELECT c1, c2 FROM public.data1
select c1 from data1	DECLARE c1 CURSOR FOR SELECT c1 FROM public.data1
select /* comment */ * from data1	DECLARE c1 CURSOR FOR SELECT c1, c2 FROM public.data1
select c1 from data1 order by 1	DECLARE c1 CURSOR FOR SELECT c1 FROM public.data1
select count(*) from data1	DECLARE c1 CURSOR FOR SELECT NULL FROM public.data1
select count(c1) from data1	DECLARE c1 CURSOR FOR SELECT c1 FROM public.data1
select c1 from data1 group by c1	DECLARE c1 CURSOR FOR SELECT c1 FROM public.data1
select * from data1 where c1 = 10	DECLARE c1 CURSOR FOR SELECT c1, c2 FROM public.data1 WHERE ((c1 < 10::numeric))
select * from data1 where octet_length(c2) = 2	DECLARE c1 CURSOR FOR SELECT c1, c2 FROM public.data1 WHERE ((octet_length(c2) = 2))
select d1.c1, d2.c2 from data1 d1, data2 d2 where d1.c1 = d2.c1	DECLARE c1 CURSOR FOR SELECT c1 FROM public.data1 WHERE ((c1 < 100::numeric)) DECLARE c2 CURSOR FOR SELECT c1, c2 FROM public.data2 WHERE ((c1 < 10::numeric))

SELECT 文を投入すると、カーソルを作成し、100 レコードずつフェッチを行っています。フェッチ数の 100 はソースコード（contrib/postgres_fdw/postgres_fdw.c）にハードコードされているため現状では変更できません。次の例は更新文で実行される SQL 文です。UPDATE 文、DELETE 文ではカーソルを定義して、1 レコードずつ UPDATE / DELETE 文を実行していることがわかります。

表 36 ローカル投入 SQL とリモート実行 SQL（更新）

ローカル投入 SQL	リモート実行 SQL
update data1 set c1 = c1 + 1 where c1 < 10	DECLARE c1 CURSOR FOR SELECT c1, c2, ctid FROM public.data1 WHERE ((c1 < 10::numeric)) FOR UPDATE FETCH 100 FROM c1 UPDATE public.data1 SET c1 = $2 WHERE ctid = $1 … UPDATE public.data1 SET c1 = $2 WHERE ctid = $1 DEALLOCATE pgsql_fdw_prep_2
insert into data1 values (20000, 'after')	INSERT INTO public.data1(c1, c2) VALUES ($1, $2) DEALLOCATE pgsql_fdw_prep_3
delete from data1 where c1 = 20000	DECLARE c1 CURSOR FOR SELECT ctid FROM public.data1 WHERE ((c1 = 20000::numeric)) FOR UPDATE FETCH 100 FROM c1 … DELETE FROM public.data1 WHERE ctid = $1 DEALLOCATE pgsql_fdw_prep_4

3.1.4 接続時に使用される情報

postgres_fdw によるリモート・インスタンスへの接続は libpq ライブラリの PQconnectdbParams 関数を使用します。接続を行う際には、CREATE SERVER 文と CREATE USER MAPPING 文で指定されたリモート・インスタンス情報に加えて以下の情報を追加して接続されます。

表 37 追加情報

属性	設定値
fallback_application_name	postgres_fdw
client_encoding	ローカル・データベースのエンコーディング名

3.2 pg_stat_statements

3.2.1 pg_stat_statements の概要

pg_stat_statements モジュールは PostgreSQL インスタンス上で実行された SQL 文および実行統計情報を蓄積し、データベース管理者に公開する contrib モジュールです。パフォーマンス・チューニング対象となる SQL 文を特定したい場合に有効です。マニュアルは以下の URL を参照してください。

http://www.postgresql.org/docs/9.4/static/pgstatstatements.html

□　インストール方法

パラメータ shared_preload_libraries に'pg_stat_statements'を登録し、監視対象データベースに対して CREATE EXTENSION pg_stat_statements 文を実行します。

□　独自パラメータ

pg_stat_statements モジュールは以下の独自パラメータを持ちます。パラメータに使用できる値はマニュアルの通りです。下記表のコメント欄は pg_settings カタログの short_desc 列の内容です。

表 38　独自パラメータの説明

パラメータ	説明
pg_stat_statements.max	Sets the maximum number of statements tracked by pg_stat_statements.
pg_stat_statements.track	Selects which statements are tracked by pg_stat_statements.
pg_stat_statements.track_utility	Selects whether utility commands are tracked by pg_stat_statements.
pg_stat_statements.save	Save pg_stat_statements statistics across server shutdowns.

□　作成されるオブジェクトとメモリー

CREATE EXTENSION 文を実行すると以下のオブジェクトが作成されます。

表 39　作成されるオブジェクト

種類	オブジェクト名	説明
FUNCTION	pg_stat_statements	統計データの参照
FUNCTION	pg_stat_statements_reset	統計データのリセット
VIEW	pg_stat_statements	pg_stat_statements(true)関数を実行するビュー

インストールが完了すると共有メモリー上に小さな領域（struct pgssSharedState のサイズ）を追加確保します。

3.2.2 pg_stat_statements の仕様

pg_stat_statements モジュールが使用するデータと制限について調査しました。

□ 入出力ファイル

pg_stat_statements モジュールにより収集されたデータは pg_stat_statements ビューで参照できますが、このビューの実体は{PGDATA}/pg_stat_tmp/pgss_query_texts.stat ファイルです。このファイルが保存されるディレクトリは、パラメータ stats_temp_directory には依存しません。

パラメータ pg_stat_statements.save パラメータが on の場合、pgss_query_texts.stat ファイルは、インスタンス停止時に{PGDATA}/pg_stat/pg_stat_statements.stat ファイルに内容がコピーされます。コピーが完了すると元ファイル pgss_query_texts.stat ファイルは削除されます。インスタンス起動時に pg_stat_statements.stat ファイルが存在するとファイルを読み込み、{PGDATA}/pg_stat_tmp/pgss_query_texts.stat ファイルに展開されます。元のファイルは削除されます。

□　同一と見なされる SQL

実行された SQL 文には ID が付与されます（queryid 列）。この ID が同一の SQL は同一と見なされて pg_stat_statements ビュー内では単一レコードになります。下記のように一部のみ異なる SQL は単一 SQL と見なされます。

- リテラルのみ異なる SQL。リテラル部分はクエスチョン・マーク（?）に変換されて query 列に格納されます。
- SQL 文の大文字／小文字が異なる SQL（SELECT と select 等）
- 単語間のスペースの個数が異なる SQL
- コメント部分が異なる SQL
- 文内に改行コードが含まれる SQL

微妙に構文が異なるが同一と見なされた SQL 文の query 列には最初に実行された SQL

文が格納されます。

□　制限

永続データである pg_stat_statements.stat ファイルに対する書込みはインスタンス停止時（smart モード、fast モード、immediate モード共通）に行われます。このため OS のクラッシュや postmaster プロセスの異常終了等、インスタンス停止処理が正常に行えない場合、SQL 実行統計情報は失われます。

3.2.3 pg_stat_statements ビュー

pg_stat_statements ビューは以下の列から構成されます。

表 40　pg_stat_statements ビューの列情報

列名	データ型	説明	備考
userid	oid	SQL 文を実行したユーザーの OID	
dbid	oid	SQL 文が実行されたデータベースの OID	
queryid	bigint	SQL 文の内部ハッシュ番号	9.4～
query	text	SQL 文の文字列	先頭部分
calls	bigint	SQL 文実行回数	
total_time	double	処理時間の累計（ミリ秒）	
rows	bigint	取得または更新レコード数累計	
shared_blks_hit	bigint	共有バッファ・キャッシュにヒットしたブロック数	
shared_blks_read	bigint	共有バッファに読み込まれたブロック数	
shared_blks_dirtied	bigint	共有バッファでダーティ状態になったブロック数	
shared_blks_written	bigint	共有バッファから書き込まれたブロック数	
local_blks_hit	bigint	ローカル・キャッシュ・ヒットしたブロック数	
local_blks_read	bigint	読み込まれたローカル・ブロック数	
local_blks_dirtied	bigint	ダーティ状態になったブロック数	
local_blks_written	bigint	書き込まれたローカル・ブロック数	
temp_blks_read	bigint	読み込まれた一時ブロック数	
temp_blks_written	bigint	書き込まれた一時ブロック数	
blk_read_time	double	ブロック読み取り時間（ミリ秒）	
blk_write_time	double	ブロック書き込み時間（ミリ秒）	

□ 検索の例

以下の例では、pg_stat_statements ビューを検索しています。実際には WHERE 句にはリテラル値を指定して実行していますが、query 列にはバインド変数化された SQL 文が格納されていることがわかります。

例 20　EXECUTE 文実行時のログ

```
postgres=> SELECT query, calls, total_time, rows FROM pg_stat_statements WHERE query
LIKE '%bind1%' ;
                        query                         | calls | total_time | rows
------------------------------------------------------+-------+------------+------
 select count(*) from bind1 where c2=?;               |     2 |    305.308 |    2
 select count(*) from bind1 where c1 between ? and ?; |     1 |     27.778 |    1
(2 rows)
```

3.3 auto_explain

3.3.1 auto_explain の概要

auto_explain モジュールは PostgreSQL インスタンス上で実行された SQL 文の実行計画を自動的にログに出力する contrib モジュールです。パフォーマンスが悪い SQL 文の特定と実行計画の確認の両方を一度に実施できるモジュールになります。マニュアルは以下の URL を参照してください。

http://www.postgresql.org/docs/9.4/static/auto-explain.html

□　インストール方法

パラメータ shared_preload_libraries に'auto_explain'を登録します。

□　独自パラメータ

auto_explain モジュールは以下の独自パラメータを持ちます。パラメータに使用できる値はマニュアルの通りです。下記表のコメント欄は pg_settings カタログの short_desc 列の内容です。

表 41　独自パラメータの説明

パラメータ	説明（extra_desc 列）
auto_explain.log_min_duration	Sets the minimum execution time above which plans will be logged.
auto_explain.log_analyze	Use EXPLAIN ANALYZE for plan logging.
auto_explain.log_verbose	Use EXPLAIN VERBOSE for plan logging.
auto_explain.log_buffers	Log buffers usage.
auto_explain.log_triggers	Include trigger statistics in plans.(This has no effect unless log_analyze is also set.)
auto_explain.log_format	EXPLAIN format to be used for plan logging.
auto_explain.log_nested_statements	Log nested statements.
auto_explain.log_timing	Collect timing data, not just row counts.

3.3.2 auto_explain の仕様

auto_explain モジュールの仕様について調査しました。

□ パラメータの初期値

auto_explain.log_min_duration パラメータには実行計画をログに出力する SQL 文の最少時間を指定します。デフォルト値は-1 です。マニュアルには-1 を指定すると無効とされていますが。実際には 0 未満の値はすべて無効と見なされます。

□　ログ出力される SQL 文

実行計画に関係が無いトランザクション制御文（BEGIN、COMMIT、ROLLBACK）や、PREPARE 文はログに出力されません。また構文に問題が無く、実行計画が作成できた SQL でも、主キー違反等の理由によりエラーになった SQL 文の実行計画も出力されません。

PREPARE 文で作成された SQL 文を EXECUTE 文で実行する場合には、ログファイルには PREPARE 文が出力されます。下記の例は EXECUTE pg1(10)文を実行した場合のログです。

例 21　EXECUTE 文実行時のログ

```
LOG:  duration: 0.021 ms  plan:
        Query Text: PREPARE pg1(integer) AS SELECT COUNT(*) FROM pgbench_accounts
where aid=$1;
        Aggregate  (cost=8.31..8.32 rows=1 width=0)
          ->  Index Only Scan using pgbench_accounts_pkey on pgbench_accounts
(cost=0.29..8.31 rows=1 width=0)
                Index Cond: (aid = 10)
```

□　PL/pgSQL 内の SQL 文

PL/pgSQL によるファンクション内から発行された SQL 文の実行計画は、標準では出力されません。ファンクション内から実行される SQL 文の実行計画を出力するには、パラメータ auto_explain.log_nested_statement を true に設定する必要があります（デフォルト値は false）。以下の例は関数 func1 内で実行された SELECT 文の実行計画を出力したログ出力です。

例 22　PL/pgSQL 内の SQL 文ログ

```
LOG:  duration: 27.732 ms  plan:
        Query Text: SELECT COUNT(*) FROM bind1 WHERE c2=val
        Aggregate  (cost=2043.30..2043.31 rows=1 width=0)
          ->  Seq Scan on bind1  (cost=0.00..1793.25 rows=100020 width=0)
                Filter: ((c2)::text = 'maj'::text)
CONTEXT:  SQL statement "SELECT COUNT(*) FROM bind1 WHERE c2=val"
        PL/pgSQL function func1(character varying) line 5 at SQL statement
LOG:  duration: 36.791 ms  plan:
        Query Text: select func1('maj');
        Result  (cost=0.00..0.26 rows=1 width=0)
```

3.4 pg_hint_plan

3.4.1 pg_hint_plan の概要

pg_hint_plan モジュールは PostgreSQL インスタンス上で実行される SQL 文の実行計画を固定するためのヒント機能を実現するモジュールです。通常は統計情報から自動的に決定される実行計画に影響を与えて、想定通りの実行計画に変更することができます。
マニュアルは以下の URL を参照してください（2015 年 1 月の最新版 1.1.3 を使用）。

http://pghintplan.sourceforge.jp/pg_hint_plan-ja.html

□　インストール方法

ソースからビルドする場合には、上記 URL から PostgreSQL のバージョンに応じたソースコードをダウンロードします。

例 23　pg_hint_plan のインストール

```
$ make
gcc -O2 -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement
-Wendif-labels -Wmissing-format-attribute -Wformat-security -fno-strict-aliasing
-fwrapv -fpic -I. -I./ -I/usr/local/pgsql/include/server
-I/usr/local/pgsql/include/internal -D_GNU_SOURCE -c -o pg_hint_plan.o
pg_hint_plan.c
gcc -O2 -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement
-Wendif-labels -Wmissing-format-attribute -Wformat-security -fno-strict-aliasing
-fwrapv -fpic -L/usr/local/pgsql/lib -Wl,--as-needed
-Wl,-rpath,'/usr/local/pgsql/lib',--enable-new-dtags -shared -o
pg_hint_plan.so pg_hint_plan.o
$ su
Password: {PASSWORD}
# make install
/bin/mkdir -p '/usr/local/pgsql/share/extension'
/bin/mkdir -p '/usr/local/pgsql/share/extension'
/bin/mkdir -p '/usr/local/pgsql/lib'
/usr/bin/install -c -m 644 pg_hint_plan.control
'/usr/local/pgsql/share/extension/'
/usr/bin/install -c -m 644 pg_hint_plan--1.1.3.sql pg_hint_plan--1.1.2--1.1.3.sql
'/usr/local/pgsql/share/extension/'
/usr/bin/install -c -m 755  pg_hint_plan.so '/usr/local/pgsql/lib/'
```

インストール先ディレクトリ等は pg_config コマンドから自動的に判断されます。

パラメータ shared_preload_libraries に'pg_hint_plan'を登録し、監視対象データベースに対して CREATE EXTENSION pg_hint_plan 文を実行します。

□　作成されるオブジェクトとメモリー

CREATE EXTENSION 文を実行すると以下のオブジェクトが作成されます。

表 42　作成されるオブジェクト

種類	オブジェクト名	説明
SCHEMA	pg_hint_plan	オブジェクト格納用スキーマ
TABLE	pg_hint_plan.hints	ヒント保存テーブル
UNIQUE INDEX	hints_norm_and_app	hints テーブルに対する一意索引

□　独自パラメータ

pg_hint_plan モジュールは以下の独自パラメータを持ちます。パラメータに使用できる値はマニュアルの通りです。下記表のコメント欄は pg_settings カタログの short_desc 列の内容です。

表 43　独自パラメータの説明

パラメータ	説明
pg_hint_plan.enable_hint	Force planner to use plans specified in the hint comment preceding to the query.
pg_hint_plan.debug_print	Logs results of hint parsing.
pg_hint_plan.parse_messages	Message level of parse errors.
pg_hint_plan.message_level	Message level of debug messages.
pg_hint_plan.enable_hint_table	Force planner to not get hint by using table lookups.

□　使用できるヒント

以下のヒントが使用できます。構文はオンライン・マニュアル（http://pghintplan.sourceforge.jp/hint_list-ja.html）を参照してください。

表 44　使用できるヒント

ヒント	説明
SeqScan / NoSeqScan	シーケンシャル・スキャンを実施する / しない
IndexScan / NoIndexScan	インデックス・スキャンを実施する / しない
IndexScanRegexp	正規表現を使ったインデックス・スキャンを実施する
BitmapScan / NoBitmapScan	ビットマップ・スキャンを実施する
BitmapScanRegexp	正規表現を使ったビットマップ・スキャンを実施する
TidScan / NoTidScan	TID スキャンを実施する / しない
IndexOnlyScan / NoIndexOnlyScan	インデックス・オンリー・スキャンを実施する / しない
IndexOnlyScanRegexp	正規表現を使ったインデックス・オンリー・スキャンを実施する
NestLoop / NoNestLoop	ネステッド・ループ結合を実施する / しない
MergeJoin / NoMergeJoin	マージ結合を実施する / しない
HashJoin / NoHashJoin	ハッシュ結合を実施する / しない
Leading	テーブルの結合順序を指定する
Set	パラメータの設定
Rows	処理レコード数の設定

3.4.2 ヒント記述時の注意点

ここではマニュアルに記載が無い動作について説明しています。

□ ヒント構文のスペースと括弧

ヒント名と括弧の間には半角スペースを含めることができます({SPACE}, {TAB}はそれぞれスペース、タブの入力を示します)。全角スペースはヒントの構文エラーになります。

例 24　ヒント名と括弧の間のスペース

```
/*+ SeqScan{SPACE}(demo1) */
/*+ IndexScan{TAB}(demo1 idx1_demo1) */
```

□ ヒントの大文字／小文字の区別

ヒント名は大文字／小文字を区別しません。下記のヒントは同一と見なされます。

例 25　ヒント名の大文字／小文字

```
/*+ SeqScan(demo1) */
/*+ seqscan(demo1) */
```

□ オブジェクト名の大文字／小文字の区別

オブジェクト名は大文字／小文字を区別します。以下のヒント記述は無効です。

例 26　オブジェクト名の大文字／小文字区別

```
/*+ SeqScan(DEMO1) */ SELECT * FROM demo1 WHERE c1 = 1000
```

□ テーブルに別名がある場合、実体名を使えるか

テーブルに別名を指定した場合、ヒントにも別名を指定する必要があります。以下のヒントは無効です。SeqScan(d1) と指定すると有効になります。

例 27　別名とヒントの指定

```
/*+ SeqScan(demo1) */ SELECT * FROM demo1 d1 WHERE c1 = 1000
```

□ オブジェクト名にスキーマ名を修飾する必要があるか

SQL 文内のテーブル名にスキーマ名を修飾している場合でもヒントにはスキーマ名を指定しません。異なるスキーマ内の同一テーブルを結合する場合には別名を付けて区別します。下記のヒントは無効になります。SeqScan(demo1)と指定すると有効になります。

例 28　スキーマとヒントの指定

```
/*+ SeqScan(schema1.demo1) */ SELECT * FROM schema1.demo1 WHERE c1 = 1000
```

□　矛盾するヒントを記述した場合

SeqScan と NoSeqScan 等、矛盾するヒントを記述した場合、後から記述したヒントが有効になります。ログには使用されなかったヒントは duplicate hint 項目に出力されます。

□　コメントの記述

ヒントとして有効なコメント形式は「/*+ コメント */」のみです。「--+ コメント」形式では使用できません。

□ マニュアルに記載がある注意点

以下の点についてはオンライン・マニュアルに記載があります。

- ヒント以前に記述できる文字
- 複数コメントに記述するヒント
- PL/pgSQL ブロック内のヒント
- オブジェクト名に対する二重引用符の使用
- 別名の使用
- FROM 句に VALUE コマンドの使用時の注意
- ルール使用時の注意
- ビューに対するヒント
- 継承テーブルに対するヒント
- 文の途中に記述するヒント
- マルチステートメント使用時の注意
- 副問い合わせ結果に対するヒント
- 構文エラー発生時の動作
- 重複ヒントの扱い
- ネストされたコメントの扱い
- 機能制限

3.4.3 IndexScan の注意点

IndexScan ヒントを使用する上で注意する点を記述します。

□　存在しないインデックス名ヒント

テーブル名の記述が正しく、インデックス名に存在しないインデックスを指定すると、Index Scan 実行計画自体が無効になるため、他の有効なインデックスが存在しても使用されません。このため Seq Scan 実行計画が選択される可能性が高くなります。

例 29　存在しないインデックス指定時の実行計画

```
postgres=> EXPLAIN /*+ IndexScan(demo1 idx1_demo1) */ SELECT * FROM demo1 WHERE
c1=1000 ;
                                QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using idx1_demo1 on demo1  (cost=0.42..8.44 rows=1 width=11)
   Index Cond: (c1 = 1000::numeric)
(2 rows)

postgres=> EXPLAIN /*+ IndexScan(demo1 idxX_demo1) */ SELECT * FROM demo1 WHERE
c1=1000 ;
                                QUERY PLAN
-------------------------------------------------------------------------
 Seq Scan on demo1  (cost=10000000000.00..10000017906.00 rows=1 width=11)
   Filter: (c1 = 1000::numeric)
(2 rows)
```

上記例の SELECT 文は、ヒントを指定しない場合、idx2_demo1 インデックスが使用されます。１回目の SELECT 文はヒントの記述が正しいため、idx1_demo1 インデックスが使用されています。しかし２回目の SELECT 文では、存在しないインデックス名 idxX_demo1 が指定されています。この場合、実行計画が Seq Scan に変更されています。

□ NoIndexScan ヒント

NoIndexScan ヒントにテーブル名とインデックスを指定した場合、対象のインデックスの Index Scan が使用されません。テーブル名のみ指定した場合は、テーブルに構成された全インデックスの Index Scan が使用されません。ただし Bitmap Index Scan が使用される可能性があります。

3.4.4 結合ヒントの注意点

テーブル間の結合を行う NestLoop ヒント、MergeJoin ヒント、HashJoin ヒントはテーブル名を複数指定します。テーブルの記述順序とは無関係に外部テーブル、内部テーブルが決定されます。下記は NestLoop ヒントにテーブルを２つ指定しています。ヒントの記述を変更しても実行計画が変わっていないことがわかります。

例 30　結合ヒントの実行

```
postgres=> EXPLAIN /*+ NestLoop(d2 d1) */ SELECT d1.*, d2.* FROM demo1 d1, demo2
d2 WHERE d1.c1 = d2.c1 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Nested Loop  (cost=0.42..503010.00 rows=1000000 width=22)
   ->  Seq Scan on demo1 d1  (cost=0.00..15406.00 rows=1000000 width=11)
   ->  Index Scan using idx1_demo2 on demo2 d2  (cost=0.42..0.48 rows=1 width=11)
         Index Cond: (c1 = d1.c1)
(4 rows)

postgres=> EXPLAIN /*+ NestLoop(d1 d2) */ SELECT d1.*, d2.* FROM demo1 d1, demo2
d2 WHERE d1.c1 = d2.c1 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Nested Loop  (cost=0.42..503010.00 rows=1000000 width=22)
   ->  Seq Scan on demo1 d1  (cost=0.00..15406.00 rows=1000000 width=11)
   ->  Index Scan using idx1_demo2 on demo2 d2  (cost=0.42..0.48 rows=1 width=11)
         Index Cond: (c1 = d1.c1)
(4 rows)
```

3.4.5 Leading ヒントと Rows ヒント

Leading ヒントは結合順序を指定し、Rows ヒントは結合レコード数の見積もりを上書きします。しかしこれらのヒントは EXPLAIN 文による実行計画では動作を確認できませんでした。以下の例はテーブル構造、インデックス構造が完全に同じであるテーブル demo1、demo2 を結合しています。Leading ヒントで結合順序を変更していますが実行計画上は変化が見られません。

例 31　Leading ヒント

```
postgres=> EXPLAIN /*+ NestLoop(d1 d2) Leading(d2 d1) */ SELECT d1.*, d2.* FROM
demo1 d1, demo2 d2 WHERE d1.c1 = d2.c1 ;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Nested Loop  (cost=0.42..503010.00 rows=1000000 width=22)
   ->  Seq Scan on demo1 d1  (cost=0.00..15406.00 rows=1000000 width=11)
   ->  Index Scan using idx1_demo2 on demo2 d2  (cost=0.42..0.48 rows=1 width=11)
         Index Cond: (c1 = d1.c1)
(4 rows)

postgres=> EXPLAIN /*+ NestLoop(d1 d2) Leading(d1 d2) */ SELECT d1.*, d2.* FROM
demo1 d1, demo2 d2 WHERE d1.c1 = d2.c1 ;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Nested Loop  (cost=0.42..503010.00 rows=1000000 width=22)
   ->  Seq Scan on demo1 d1  (cost=0.00..15406.00 rows=1000000 width=11)
   ->  Index Scan using idx1_demo2 on demo2 d2  (cost=0.42..0.48 rows=1 width=11)
         Index Cond: (c1 = d1.c1)
(4 rows)
```

3.4.6 ログ

pg_hint_plan モジュールには、主にアプリケーション開発時に利用できるログ出力機能を持っています。

□　デバッグ・ログ

pg_hint_plan.debug_print パラメータを on に設定すると、LOG レベルで pg_hint_plan の動作状況がログ出力されます（デフォルト値は off）。以下のフォーマットでログが出力されます。

例 32　pg_hint_plan.debug_print

```
LOG:  available indexes for IndexScan(demo1): idx1_demo1
STATEMENT:  /*+ IndexScan(demo1 idx1_demo1) */ SELECT * FROM demo1 WHERE c1=1000;
LOG:  pg_hint_plan:
        used hint:
        IndexScan(demo1 idx1_demo1)
        not used hint:
        duplication hint:
        error hint:

STATEMENT:  /*+ IndexScan(demo1 idx1_demo1) */ SELECT * FROM demo1 WHERE c1=1000;
```

上記は、SQL「/*+ IndexScan(demo1 idx1_demo1) */ SELECT * FROM demo1 WHERE c1=1000」に対するログ出力です。先頭のログは IndexScan ヒントにより使用できるインデックス情報が出力されています。次の LOG には、ヒントの仕様状況が出力されます。

表 45　ログの出力項目

項目	説明
used hint:	使用されたヒント
not used hint:	使用されなかったヒント
duplication hint:	重複して記述されたヒント
error hint:	記述が間違っているヒント

ヒントが実際には使用されなくても、テーブル名が正しい場合は used hint: に出力される場合があります。

□　パース・エラー

pg_hint_plan.parse_messages パラメータは構文エラー時のログ出力レベルを指定します。デフォルト値は info です。ヒント構文のパースに失敗すると以下のログが出力されます。ヒントのパース・エラーが発生しても SQL 文自体は実行されます。

例 33　ヒントのパース・エラー

```
postgres=> /*+ SeqScan() */ SELECT * FROM demo1 WHERE c1 = 100 ;
INFO:  pg_hint_plan: hint syntax error at or near " "
DETAIL:  SeqScan hint requires a relation.
```

上記例では、SeqScan ヒントにテーブル名の指定がありません。パース・エラーのログは構文エラーの場合にのみ出力されます。具体的には以下のメッセージが出力されます。

表 46　ログの出力項目

エラー	メッセージ
(が無い	Opening parenthesis is necessary.
) が無い	Closing parenthesis is necessary.
引用符が閉じていない	Unterminated quoted string.
文字列長が 0	Zero-length delimited string.
解析できないヒント名	Unrecognized hint keyword
コメントが閉じていない	Unterminated block comment.
ネスト・コメントのエラー	Nested block comments are not supported.
ヒントのコンフリクト	Conflict %s hint.
単一テーブル・ヒント	%s hint accepts only one relation.
テーブル名が必要	%s hint requires a relation.
２つ以上のテーブル・ヒント	%s hint requires at least two relations.
ヒント構文エラー	%s hint requires two sets of relations when parentheses nests.
GUC 設定エラー	%s hint requires name and value of GUC parameter.
レコード数の値エラー	Unrecognized rows value type notation.
レコード数の値エラー	%s hint requires valid number as rows estimation.
テーブル名が不定	Relation name ¥"%s¥" is ambiguous.
テーブル名が重複	Relation name ¥"%s¥" is duplicated.

3.4.7 テーブル・ヒント

パラメータ pg_hint_plan.enable_hint_table を on に設定すると、テーブル・ヒントが利用できます。テーブル hint_plan.hints に実行される SQL 文とヒントを格納することで、自動的にヒントを付加した実行計画が作成されます。

ただしインスタンス全体で pg_hint_plan.enable_hint_table パラメータを on に設定すると、CREATE EXTENSION 文を実行していないデータベースでも本機能が動作するため、SQL 文の実行がエラーになります。下記は CREATE EXTENSION 文を実行していない postgres データベースに対して SELECT 文を実行した場合のエラーです。

例 34　SQL 文実行エラー

```
postgres=# SELECT * FROM pg_settings ;
ERROR:  relation "hint_plan.hints" does not exist
LINE 1: SELECT hints   FROM hint_plan.hints  WHERE norm_query_string...
                             ^
QUERY:  SELECT hints   FROM hint_plan.hints  WHERE norm_query_string = $1    AND
( application_name = $2     OR application_name = '' )  ORDER BY application_name
DESC
postgres=#
```

3.5 pg_rman

3.5.1 pg_rman の概要

pg_rman モジュールは PostgreSQL インスタンスのデータを安全にバックアップするためのコマンドを提供します。2015 年 1 月現在の最新バージョンは 1.2.11 です。詳細は以下の URL で確認してください。

http://sourceforge.net/projects/pg-rman/

□　インストール方法

ソースコードを展開し、make および make install コマンドによりインストールが行われます。インストール後は{INSTALL}/bin/pg_rman コマンドが作成されます。

3.5.2 pg_rman の使用

pg_rman を使用するための簡単な説明です。

□　バックアップ・カタログの作成

バックアップ・データの保存や、環境を管理するためにバックアップ・カタログを作成する必要があります。バックアップ・カタログは空ディレクトリを指定します。環境変数 BACKUP_PATH を指定することで、pg_rman コマンド実行時の指定を省略することができます。ディレクトリは絶対パスで指定する必要があります。

例 35　バックアップ・カタログの作成

```
$ pg_rman init -B pg_rman_catalog -D ${PGDATA}
ERROR: -B, --backup-path must be an absolute path
$ pg_rman init -B /pg_rman_catalog -D ${PGDATA}
INFO: ARCLOG_PATH is set to '/arch'
INFO: SRVLOG_PATH is set to '/data/pg_log'
$ export BACKUP_PATH=/pg_rman_catalog
```

ディレクトリ内にはいくつかのファイルとサブ・ディレクトリが作成されます。

表 47　バックアップ・カタログ内のディレクトリ／ファイル

ディレクトリ／ファイル		説明
{YYYYMMDD}		バックアップ先ディレクトリ（バックアップ日）
{HHMISS}		バックアップ先ディレクトリ（バックアップ時刻）
	archlog	アーカイブログのバックアップ・ディレクトリ
	backup.ini	バックアップ時のパラメータ・ファイル
	database	データベース・クラスタのバックアップ・ディレクトリ
	mkdirs.sh	データベース・クラスタのディレクトリ作成スクリプト
	srvlog	サーバーログ・ディレクトリ
	file_archlog.txt	アーカイブログ・ファイル一覧
	file_database.txt	データベース・クラスタ・ファイル一覧
	file_srvlog.txt	ログファイル一覧
backup		不明
pg_rman.ini		pg_rman パラメータのデフォルト値
timeline_history		不明

□　バックアップの作成

バックアップを行うにはサブ・コマンドとして backup を指定します。更にいくつかのパラメータを指定することができます。接続先を指定しない場合、ローカル・インスタンスに管理者権限で接続します。下記の例はフル・バックアップを実行しています。

例 36　フル・バックアップの実行

```
$ pg_rman backup --backup-mode=full --with-serverlog
INFO: database backup start
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
```

--backup-mode=full パラメータを指定すると、データベース・クラスタおよび全アーカイブログがバックアップされます。

フル・バックアップを実行する際には以下の SQL 文が実行されます。

- SELECT current_setting('block_size')
- SELECT current_setting('wal_block_size')
- SELECT * from pg_xlogfile_name_offset(pg_start_backup('YYYY-MM-DD HH:MI:SS with pg_rman', 't'))
- SELECT * FROM pg_xlogfile_name_offset(pg_stop_backup())
- SELECT txid_current();

バックアップ・モードの指定によりバックアップされるデータは以下の通りです。archive を指定した場合でも、ログスイッチが発生し、最新のアーカイブログまでバックアップされます。

表 48　--backup-mode パラメータ指定

モード	セグメント	WAL ファイル	アーカイブログ	設定ファイル
full	○	×	○	○
incremental	○（差分のみ）	×	○	×
archive	×	×	○	×

標準以外のテーブル空間を作成していた場合は、バックアップ・カタログ内の pg_tblspc ディレクトリ内にシンボリック・リンクではなく実ファイルとしてバックアップされます。

□ 検証

validate サブ・コマンドを指定すると、バックアップの整合性を検証することができます。

例 37　バックアップの検証

```
$ pg_rman validate
INFO: validate: 2015-01-14 11:29:24 backup and archive log files by CRC
```

□ 確認

バックアップの一覧を確認するにはサブ・コマンドとして show を指定します。

例 38　バックアップの一覧

```
$ pg_rman show
==========================================================================
Start               Time   Total    Data     WAL     Log  Backup   Status
==========================================================================
2015-01-14 11:47:33    0m   295MB    ----    33MB   1066B   326MB   DONE
2015-01-14 11:29:24    0m   295MB    ----   100MB   153kB   393MB   OK
2015-01-14 11:19:56    8m      0B    ----    ----    ----      0B   ERROR
```

show コマンドに、Start 列の日次を指定すると、バックアップの詳細を表示することができます。Status 列にはバックアップの状態が出力されます。以下のステータスが表示されます。

表 49　Status 列の表示

Status 列の表示	説明	備考
OK	バックアップ、検証完了	
RUNNING	バックアップ実行中	
ERROR	バックアップ障害	バックアップ途中で終了等
DELETING	バックアップ削除中	
DONE	バックアップ完了、未検証	validate により OK に変更
CORRUPT	バックアップ不正	
UNKNOWN	不明なステータス	
DELETED	バックアップ削除済	--show-all 指定時のみ表示

タイムラインを表示するには show timeline を指定します。

例 39　タイムラインの表示

```
$ pg_rman show timeline
========================================================
Start               Mode  Current TLI  Parent TLI  Status
========================================================
2015-01-14 11:47:33  FULL            1           0  OK
2015-01-14 11:29:24  FULL            1           0  OK
2015-01-14 11:19:56  FULL            1           0  ERROR
```

Mode 列には操作が出力されます。以下の情報が表示されます。

表 50　Mode 列の表示

Mode 列の表示	説明
FULL	フル・バックアップ
INCR	インクリメンタル・バックアップ
ARCH	アーカイブログのバックアップ

□ リストア

サブ・コマンド restore を実行するとリストアを行うことができます。リストアしたデータベース・クラスタには recovery.conf ファイルが作成されます。元のデータベース・クラスタが削除されている場合は以下の警告メッセージが出力されます。データベース・クラスタの保護モードは変更されません。

例 40　リストア時の警告

```
$ pg_rman restore
WARNING: can't open pg_controldata file "/data/global/pg_control": No such file
or directory
INFO: validate: 2015-01-15 06:39:29 backup and archive log files by SIZE
INFO: restore complete. Recovery starts automatically when the PostgreSQL server
is started.
$ cat /data/recovery.conf
# recovery.conf generated by pg_rman 1.2.11
restore_command = 'cp /arch/%f %p'
recovery_target_timeline = '1'
```

またアーカイブログ用ディレクトリには、バックアップ・カタログ内に保存されたアーカイブログ・ファイルへのシンボリック・リンクが作成されます。

3.5.3 差分バックアップ検証

pg_rman は差分バックアップの機能を提供しています。差分バックアップは、前回取得した LSN 以降に更新したブロックを差分としてバックアップします。読み込み単位はブロック（=ページ）のサイズである 8 KB です。ここでは差分バックアップについて検証しました。

□ 差分バックアップ要件

差分バックアップを行うためには、事前にフル・バックアップが行われ、かつ検証（validate）済である必要があります。フル・バックアップが検証を行っていない場合、以下のメッセージが出力されてバックアップは行われません。

例 41　差分バックアップの実行

```
$ pg_rman backup --backup-mode=incremental
INFO: database backup start
ERROR: There is no validated full backup with current timeline.Please take a full
backup and validate it before doing an incremental backup.
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
```

□ 差分として出力されるデータ

テーブルを一部更新し、差分バックアップを実行することで、バックアップされたセグメント・ファイルのサイズを確認します。

例 42　差分バックアップの検証

```
-- フル・バックアップの取得
$ pg_rman backup --backup-mode=full
INFO: database backup start
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
-- 一部データの更新
postgres=> INSERT INTO large1(c0) VALUES (200000) ;
INSERT 0 1
-- ファイルの確認
postgres=> SELECT pg_relation_filepath('large1') ;
 pg_relation_filepath
----------------------
 base/16385/16446
(1 row)
postgres=> ¥! ls -l data/base/16385/16446*
-rw-------. 1 postgres postgres 1073741824 Jan 15 06:37 data/base/16385/16446
-rw-------. 1 postgres postgres  892362752 Jan 19 06:59 data/base/16385/16446.1
-rw-------. 1 postgres postgres     499712 Jan 15 06:35 data/base/16385/16446_fsm
-- 差分バックアップ
$ pg_rman backup --backup-mode=incremental
INFO: database backup start
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
-- バックアップされたファイルの確認
$ find ${BACKUP_PATH} -name '16446*' | xargs ls -l
-rw-------.  1  postgres  postgres  959890816  Jan  15  06:39
pg_rman_catalog/20150115/063929/database/base/16385/16446
-rw-------.  1  postgres  postgres  873389600  Jan  15  06:40
pg_rman_catalog/20150115/063929/database/base/16385/16446.1
-rw-------.  1  postgres  postgres       5756  Jan  19  07:04
pg_rman_catalog/20150119/070428/database/base/16385/16446.1
```

バックアップ・カタログからテーブル large1 を構成するファイル 16446 を検索すると、差分バックアップ時更新されたファイルのみがバックアップされていることがわかります。またサイズから、更新された一部分のみバックアップされています。

□　バックアップ・ファイルの一覧

ファイルが更新されたかを比較するために、バックアップ・カタログ内には以下の２つのファイルが作成されます。

file_database.txt

file_arclog.txt

これらのファイルはそれぞれ、データベース・クラスタ内のファイル名とアーカイブログ・ファイル名が格納されています。ファイルごとに１行のデータが以下のフォーマットで書き込まれます。

データベース・クラスタからの相対パス 種別 サイズ CRC モード 更新日時

例 43　file_database.txt

```
$ cd ${BACKUP_PATH}
$ cd 20150119/063411
$ cat file_database.txt
PG_VERSION f 4 1719332343 0600 2015-01-14 04:40:58
backup_label f 222 1389378647 0600 2015-01-19 06:34:11
base d 0 0 0700 2015-01-14 05:24:38
base/1 d 0 0 0700 2015-01-14 04:41:20
base/1/12735 f 131072 2694225761 0600 2015-01-14 04:41:03
base/1/12735_fsm f 24576 3874894805 0600 2015-01-14 04:41:03
base/1/12735_vm F 32 2136789639 0600 2015-01-14 04:41:03
base/1/12737 F 13016 4053936 0600 2015-01-14 04:41:03
<< 以下省略 >>
```

3.6 passwordcheck

3.6.1 passwordcheck の概要

passwordcheck モジュールは PostgreSQL ユーザーのパスワードを強化する contrib モジュールです。ユーザー作成時やパスワード変更時により複雑なパスワードを指定することを強制できます。利用するためには、パラメータ shared_preload_libraries に passwordcheck を指定します。

3.6.2 チェックされる内容

passwordcheck モジュールは CREATE USER 文、ALTER USER 文実行時に指定されたパスワードをチェックし、要件に合致するパスワードのみ実行を許可します。以下の制限が有効になります。

表 51　チェックされる内容（平文パスワード）

チェック項目	制限	備考
パスワード文字数	8 文字以上	
ユーザー名との関係	ユーザー名を含まない	
アルファベット	アルファベットを含む	
アルファベット以外	アルファベット以外を含む	
平易なパスワード	CrackLib によるチェック	CrackLib 統合時のみ

表 52　チェックされる内容（MD5 パスワード）

チェック項目	制限	備考
ユーザー名との関係	ユーザー名と異なる	

□ エラー・メッセージ

以下のメッセージが表示される場合があります。

表 53　メッセージ

チェック	メッセージ	備考
ユーザー名を含むパスワード	password must not contain user name	
パスワード長が短い	password is too short	
アルファベットとそれ以外を含む	password must contain both letters and nonletters	
CrackLib に含まれる単語	password is easily cracked	
平文と MD5 以外を使用	unrecognized password type: %d	
CrackLib に違反する単語を使用	password is easily cracked	

例 44　file_database.txt

```
postgres=# CREATE USER demo1 PASSWORD 'demo1' ;
ERROR:  password is too short
postgres=# CREATE USER demo2 PASSWORD 'strdemo2str' ;
ERROR:  password must not contain user name
postgres=# ALTER USER demo3 PASSWORD 'abcdefghi' ;
ERROR:  password must contain both letters and nonletters
postgres=# CREATE USER demo4 PASSWORD 'abcdefg1' ;
ERROR:  password is easily cracked
```

3.6.3 CrackLib との統合

passwordcheck モジュールは CrackLib ライブラリ（http://sourceforge.net/projects/cracklib/）が提供する辞書を使用してパスワードを制限することができますが、標準では無効になっています。辞書ファイル/usr/lib/cracklib_dict が標準で使用されます。変更する場合は configure コマンドのオプション--with-default-dict で指定します。passwordcheck モジュールの Makefile には CrackLib を使用するマクロ（PG_CPPFLAGS と SHLIB_LINK）が無効になっているため、コメントをはずしてリビルドを行います。

□ Red Hat Enterprise Linux 6.5 環境におけるインストール

Red Hat Enterprise Linux 6.5 には CrackLib ライブラリが提供されていますが、いくつか注意点があります。

- crack.h ファイルの準備

Red Hat Enterprise Linux が提供する rpm パッケージには、passwordcheck モジュール

に必要な crack.h が含まれません。CrackLib 2.8.16 のソースコードまたは cracklib-devel パッケージから crack.h ファイルを passwordcheck モジュールのディレクトリにコピーします。

- libcrack.so ファイルの準備

/usr/lib64 ディレクトリには libcrack.so.2 ファイルのみ存在するため、シンボリック・リンクを作成しあます。

例 45　シンボリック・リンクの作成

```
# cd /usr/lib64
# ln -s libcrack.so.2 libcrack.so
# ls -l libcrack.so
lrwxrwxrwx. 1 root root 13 Jan 21 04:37 libcrack.so -> libcrack.so.2
```

- ディクショナリの設定

Red Hat Enterprise Linux の CrackLib ライブラリには辞書として「/usr/share/cracklib/cracklib-small*」が提供されています。Makefile の CRACKLIB_DICTPATH に/usr/share/cracklib/cracklib-small を指定します。

例 46　Makefile の修正

```
# contrib/passwordcheck/Makefile

MODULE_big = passwordcheck
OBJS = passwordcheck.o

# uncomment the following two lines to enable cracklib support
PG_CPPFLAGS = -DUSE_CRACKLIB
        '-DCRACKLIB_DICTPATH="/usr/share/cracklib/cracklib-small "'
SHLIB_LINK = -lcrack
<<以下省略>>
```

- ビルドとインストール

make コマンドでビルドとインストールを行います。

例 47　ビルドとインストール

```
$ make clean
rm -f passwordcheck.so    libpasswordcheck.a   libpasswordcheck.pc
rm -f passwordcheck.o
$ make
gcc -O2 -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement
-Wendif-labels -Wmissing-format-attribute -Wformat-security -fno-strict-aliasing
-fwrapv                              -fpic                                -DUSE_CRACKLIB
'-DCRACKLIB_DICTPATH="/usr/share/cracklib/cracklib-small"'          -I.          -I.
-I../../src/include -D_GNU_SOURCE    -c -o passwordcheck.o passwordcheck.c
gcc -O2 -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement
-Wendif-labels -Wmissing-format-attribute -Wformat-security -fno-strict-aliasing
-fwrapv  -fpic  -shared  -o  passwordcheck.so  passwordcheck.o  -L../../src/port
-L../../src/common                                                  -Wl,--as-needed
-Wl,-rpath,'/usr/local/pgsql/lib',--enable-new-dtags   -lcrack
# make install
/bin/mkdir -p '/usr/local/pgsql/lib'
/usr/bin/install         -c         -m         755                 passwordcheck.so
'/usr/local/pgsql/lib/passwordcheck.so'
```

3.7 pg_upgrade

3.7.1 pg_upgrade の概要

pg_upgrade はデータベース・クラスタのバージョンアップを、安全かつ高速に実施するための contrib モジュールです。pg_upgrade コマンドを実行することで、古いバージョンのデータベース・クラスタに含まれるデータを最新バージョンにアップグレードします。pg_upgrade の実行には以下の 3 つのモードがあります。

□ 確認モード

アップグレードが実現できるかをチェックするモードです。実際の移行モードでも確認は行われます。

□ リンクによる移行（リンクモード）

旧バージョンのデータベース・クラスタ内のセグメント・ファイルを直接利用してアップグレードを行います。ファイルのコピーを伴わないため、高速にアップグレードできますが、旧バージョンのデータベース・クラスタは使用できなくなります。

□ コピーによる移行（コピーモード）

標準の移行方法です。新バージョンのデータベース・クラスタに旧バージョンのファイルをコピーしてアップグレード版のデータベース・クラスタを作成します。リンクモードよりも時間がかかりますが、旧バージョンと併用が可能です。

表 54　pg_upgrade のモード

処理内容	チェック	リンク	コピー
アップグレードの可否チェック	実施	実施	実施
必要容量	—	クラスタのみ	クラスタとデータ
アップグレード時間	—	短時間	長時間
コマンド・パラメータ	--check	--link	なし
インスタンスの停止	必要	必要	必要

3.7.2 pg_upgrade の実行手順

pg_upgrade コマンド実行方法について簡単に説明します。

□　事前準備

pg_upgrade コマンドを実行する前に、旧バージョンの PostgreSQL が稼働するサーバー

で以下の準備を行います。

- 新バージョンのバイナリ・インストール
  旧バージョンと同一の configure オプションを使用します。
- 新バージョンのデータベース・クラスタ作成
  旧バージョンと互換性があるデータベース・クラスタを作成します。
- インスタンスの停止
  新バージョン、旧バージョン共にインスタンスを停止します。

□　pg_upgrade コマンドの実行

旧バージョンのバイナリ、データベース・クラスタ、新バージョンのバイナリ、データベース・クラスタを指定して新バージョンに含まれる pg_upgrade コマンドを実行します。

例 48　実行例

```
$ pg_upgrade -d /usr/local/pg916/data -D /usr/local/pg940/data -b
/usr/local/pg916/bin -B /usr/local/pg940/bin
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok
Checking database user is a superuser                       ok
Checking for prepared transactions                          ok
Checking for reg* system OID user data types                ok
<<以下省略>>
```

3.7.3 互換性の検証

pg_upgrade コマンドによるアップグレードを成功させるには、新旧バージョンの PostgreSQL バイナリおよびデータベース・クラスタ設定（initdb 設定）が互換性を持っている必要があります。ここでは initdb のコマンド・オプションの差により、pg_upgrade が成功するかを検証しました。ここでは以下のバージョンを使って検証を行います。

表 55　移行のテスト環境

比較	旧バージョン	新バージョン	備考
バージョン	PostgreSQL 9.1.6	PostgreSQL 9.4.0	
configure オプション	デフォルト	デフォルト	
バイナリ・ディレクトリ	/usr/local/pg916/bin	/usr/local/pg940/bin	
データベース・クラスタ	/usr/local/pg916/data	/usr/local/pg940/data	

□　ページ・チェックサム機能が異なる

データベース・クラスタ間でページ・チェックサム設定が異なる場合はアップグレードできません。下記の例では新バージョンのデータベース・クラスタでチェックサム機能が有効になっています。

例 49　チェックサム機能が異なる

```
$ pg_upgrade -d /usr/local/pg916/data -D /usr/local/pg940/data -b
/usr/local/pg916/bin -B /usr/local/pg940/bin

Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok

old and new pg_controldata checksum versions are invalid or do not match
Failure, exiting
```

□　ロケールが異なる場合

新旧バージョンのデータベース・クラスタ間でロケール設定が異なる場合はアップグレードできません。下記の例では旧バージョンのデータベース・クラスタが ja_JP ロケール、新バージョンのデータベース・クラスタが「ロケールなし」を指定しています。

例 50　ロケールが異なる

```
$ pg_upgrade -d /usr/local/pg916/data -D /usr/local/pg940/data -b
/usr/local/pg916/bin -B /usr/local/pg940/bin
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok
Checking database user is a superuser                       ok
Checking for prepared transactions                          ok
Checking for reg* system OID user data types                ok
Checking for contrib/isn with bigint-passing mismatch       ok
Checking for invalid "line" user columns                    ok

lc_collate cluster values do not match:  old "ja_JP.eucJP", new "C"
Failure, exiting
```

□　エンコーディングが異なる場合

ロケールは共通で、エンコーディングのみ異なる場合でもアップグレードできません。以下の例では、旧バージョンのデータベース・クラスタはエンコーディングが日本語 EUC、新バージョンのデータベース・クラスタは UTF-8 が指定されています。

例 51　エンコーディングが異なる

```
$ pg_upgrade -d /usr/local/pg916/data -D /usr/local/pg916/data -b /usr/local/
pg916/bin -B /usr/local/pg940/bin
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok
Checking database user is a superuser                       ok
Checking for prepared transactions                          ok
Checking for reg* system OID user data types                ok
Checking for contrib/isn with bigint-passing mismatch       ok
Checking for invalid "line" user columns                    ok

encoding cluster values do not match:  old "EUC_JP", new "UTF8"
Failure, exiting
```

□　その他のチェック

pg_upgrade コマンドは上記で説明した以外に、以下の項目をチェックします。これらの比較はすべて pg_control ファイルの内容から取得しています。下記の表以外にもバージョン等、多数のチェック項目が定義されています。

表 56 主なチェック項目（pg_control ファイルから）

チェック項目	エラー・メッセージ
アライメントの比較（32 ビット版と 64 ビット版）	old and new pg_controldata alignments are invalid or do not match Likely one cluster is a 32-bit install, the other 64-bit
ブロック・サイズ	old and new pg_controldata block sizes are invalid or do not match
最大セグメント・サイズ	old and new pg_controldata maximum relation segement sizes are invalid or do not match
WAL ブロック・サイズ	old and new pg_controldata WAL block sizes are invalid or do not match
WAL セグメント・サイズ	old and new pg_controldata WAL segment sizes are invalid or do not match
最大オブジェクト名長	old and new pg_controldata maximum identifier lengths are invalid or do not match
最大 TOAST チャンク・サイズ	old and new pg_controldata maximum TOAST chunk sizes are invalid or do not match
date/time 型のタイプ	Old and new pg_controldata date/time storage types do not match. You will need to rebuild the new server with configure option --disable-integer-datetimes or get server binaries built with those options.

3.7.4 アップグレードの実行

pg_upgrade コマンドが成功すると、Upgrade Complete メッセージが表示されます。コマンド実行中には、一時ファイルをカレント・ディレクトリに作成するため、書き込み可能なディレクトリで pg_upgrade コマンドを実行する必要があります。

pg_upgrade コマンドを実行したカレント・ディレクトリに以下のシェル・スクリプトが生成されます。

- analyze_new_cluster.sh
  「vacuumdb --all --analyze-in-stages」コマンドを実行します。
- delete_old_cluster.sh
  「rm -rf 旧バージョンのデータベース・クラスタ」コマンドを実行します。

例 52　pg_upgrade の成功（コピーモード）

```
$ pg_upgrade -d /usr/local/pg916/data -D /usr/local/pg916/data -b /usr/local/
pg916/bin -B /usr/local/pg940/bin
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok
Checking database user is a superuser                       ok
<<途中省略>>
Checking for prepared transactions                          ok

If pg_upgrade fails after this point, you must re-initdb the
new cluster before continuing.

Performing Upgrade
------------------
Analyzing all rows in the new cluster                       ok
Freezing all rows on the new cluster                        ok
Deleting files from new pg_clog                             ok
<<途中省略>>

Upgrade Complete
----------------
Optimizer statistics are not transferred by pg_upgrade so,
once you start the new server, consider running:
    analyze_new_cluster.sh
Running this script will delete the old cluster's data files:
    delete_old_cluster.sh
Running this script will delete the old cluster's data files:
    delete_old_cluster.sh
```

pg_upgrade コマンドで移行される情報はデータベース、テーブルやビュー等のオブジェクト、ユーザーの情報のみで、postgresql.conf ファイル、pg_hba.conf ファイル等の設定ファイルの内容は引き継がれません。

□　pg_upgrade コマンドによるインスタンス起動

pg_upgrade コマンド実行前は、新旧バージョンのインスタンスは停止しています。pg_upgrade コマンド実行中に、旧バージョンのインスタンスは１回、新バージョンのインスタンスは２回起動されます。pg_upgrade コマンド終了時には新旧共にインスタンスは停止しています。

3.7.5 内部構造

pg_upgrade コマンドの内部構造について調査した内容を記載しています。

□　一時ファイル

pg_upgrade コマンドは、カレント・ディレクトリに以下のファイルを作成します。--retain パラメータを指定すると一時ファイルは削除されずに残されます。pg_upgrade_dump_{OID}.custom ファイルの{OID}は pg_database カタログの oid 列の値です。

表 57　一時ファイル

ファイル名	説明	備考
pg_upgrade_dump_{OID}.custom	pg_dump コマンド出力ファイル	
pg_upgrade_dump_{OID}.log	pg_dump コマンド出力ログ	
pg_upgrade_dump_globals.sql	pg_dumpall コマンドによるグローバル・オブジェクトの出力	
pg_upgrade_internal.log	pg_upgrade コマンド・ログ	
pg_upgrade_server.log	インスタンス起動停止ログ	
pg_upgrade_utility.log	各種コマンド実行ログ	

□　一時オブジェクト

pg_upgrade コマンド実行中に新バージョン、旧バージョンのデータベース・クラスタには一時的にオブジェクトが作成されます。このため読み込み専用のデータベース（レプリケーションのスタンバイ等）は移行に失敗します。これらの一時作成されるオブジェクトの実体は、${INSTALL}/lib/pg_upgrade_support.so ライブラリで定義されています。pg_upgrade_support.so ライブラリのソースコードは contrib/pg_upgrade_support ディレクトリに格納されています。

表 58　一時オブジェクト（旧バージョン・データベース）

オブジェクト名	種類	備考
info_rels	TEMPORARY TABLE	

表 59　一時オブジェクト（新バージョン・データベース）

オブジェクト名	種類	備考
info_rels	TEMPORARY TABLE	
binary_upgrade	SCHEMA	
binary_upgrade.set_next_pg_type_oid	FUNCTION	
binary_upgrade.set_next_array_pg_type_oid	FUNCTION	
binary_upgrade.set_next_toast_pg_type_oid	FUNCTION	
binary_upgrade.set_next_heap_pg_class_oid	FUNCTION	
binary_upgrade.set_next_index_pg_class_oid	FUNCTION	
binary_upgrade.set_next_pg_enum_oid	FUNCTION	
binary_upgrade.set_next_pg_authid_oid	FUNCTION	
binary_upgrade.create_empty_extension	FUNCTION	

□　実行されるコマンド

以下のコマンドが実行されます。主にログファイル pg_upgrade_utility.log から抜粋しています。下記以外に pg_ctl コマンドによるインスタンスの起動／停止が発生します。

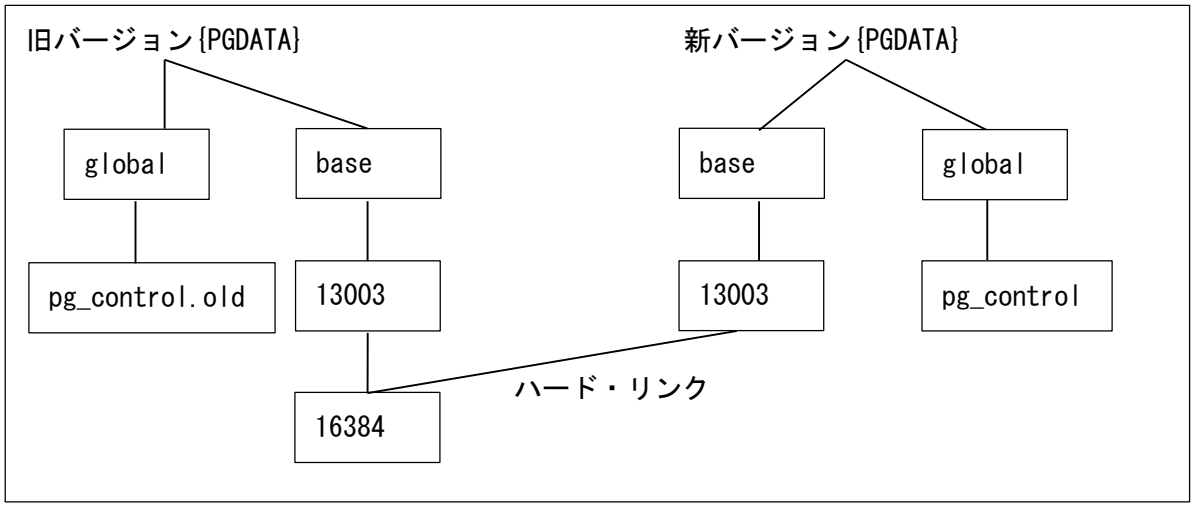
表 60　実行される主なコマンド（セグメント・ファイルのコピーを除く）

コマンド	主なオプション	インスタンス
pg_dumpall	--globals-only --quote-all-identifiers --binary-upgrade	旧バージョン
pg_dump	--schema-only --quote-all-identifiers --binary-upgrade --format=custom	旧バージョン
vacuumdb	--all --analyze	新バージョン
vacuumdb	--all --freeze	新バージョン
cp	pg_clog のコピー pg_multixact のコピー	旧バージョン→新バージョン
pg_restore	-exit-on-error --verbose --dbname	新バージョン
pg_resetxlog	-f -x	新バージョン
pg_resetxlog	-f -e	新バージョン
pg_resetxlog	-m	新バージョン
pg_resetxlog	-l	新バージョン
psql	-f "pg_upgrade_dump_globals.sql"	新バージョン
pg_resetxlog	-o	新バージョン
initdb	--sync-only	新バージョン

3.7.6 リンクモードの動作

リンクモードはデータベース・クラスタ内のセグメント・ファイルを新旧バージョンで共有することで移行時間を短縮する機能です。ユーザーが作成したオブジェクトのセグメント・ファイルはハード・リンクにより新旧バージョンのデータベース・クラスタで共有されます。旧バージョンのデータベース・クラスタ内に保存された pg_control ファイルはインスタンス起動によるデータベース破壊を防ぐために、ファイル名が pg_control.old に変更されます。

図 1　リンクモードによる移行



セグメント・ファイルはハード・リンクにより共有されるため、新旧バージョンのデータベース・クラスタは同一のファイルシステム内に配置する必要があります。新旧バージョンのデータベース・クラスタが異なるファイルシステムに配置されている場合、以下のメッセージが表示されて移行は失敗します。

例 53　異なるファイルシステム間のリンクモード

```
Could not create hard link between old and new data directories: Invalid cross-device
link
In link mode the old and new data directories must be on the same file system volume.
Failure, exiting
```

3.7.7 テーブル空間の移行

旧バージョンでデータベース・クラスタの外部に作成されたテーブル空間が存在する場合でも pg_upgrade コマンドは正常に終了します。ただしテーブル空間のディレクトリは新旧バージョンのデータベース・クラスタで共有されます。それぞれの{PGDATA}/pg_tblspc ディレクトリ内のファイルは同一ディレクトリを指すシンボリック・リンクが作成されます。テーブル空間用のディレクトリ内には、新バージョンのフォーマットを示すディレクトリが作成され、セグメント・ファイル等が作成されます。

例 54　テーブル空間の移行

```
$ ls -l /usr/local/pg940/data/pg_tblspc/
lrwxrwxrwx. 1 postgres postgres 26 Jan 22 07:41 16414 -> /usr/local/pg916/ts1
$ ls -l /usr/local/pg916/data/pg_tblspc/
lrwxrwxrwx. 1 postgres postgres 26 Jan 22 07:39 16385 -> /usr/local/pg916/ts1
$ ls -l /usr/local/pg916/ts1
total 8
drwx------. 3 postgres postgres 4096 Jan 22 08:20 PG_9.1_201105231 ←旧
drwx------. 3 postgres postgres 4096 Jan 22 08:22 PG_9.4_201409291 ←新
```

3.8 pgstattuple

3.8.1 pgstattuple の概要

pgstattuple はテーブルやインデックスに関する物理的な統計情報を取得する関数を提供する contrib モジュールです。

□ pgstattuple モジュールの利用方法

利用するためには、パラメータ shared_preload_libraries に pgstattuple を指定します。モジュールを利用するデータベースに管理者権限で接続し、CREATE EXTENSION pgstattuple 文を実行します。

□ 利用可能な機能

pgstattuple モジュールをインストールすると、以下の関数が利用できるようになります。これらの関数は superuser 権限を持つユーザーのみ利用できます。下記の関数対象となるオブジェクトに対して読み取りロック（AccessShareLock）を取得します。

表 61 提供される関数

関数名	機能	備考
pgstattuple	オブジェクトの物理統計情報を返す。	レコード形式
pgstatindex	B-tree インデックスの統計情報を返す。	レコード形式
pgstatginindex	GIN インデックスの統計情報を返す。	レコード形式
pg_relpages	テーブル内のページ数を返す。	

例 55 一般ユーザーによる実行エラー

```
postgres=> SELECT * FROM pg_relpages('sample1') ;
ERROR:  must be superuser to use pgstattuple functions
```

3.8.2 pgstattuple 関数

pgstattuple 関数はオブジェクトの物理統計情報を出力する関数です。

□ pgstattuple 関数の使用方法

pgstattuple 関数の戻り値はレコード形式のため SELECT 文の FROM 句に指定することができます。

pgstattuple 関数は、パラメータとして pg_class カタログの relname 列で示されるオブジェクト名または OID を指定します。オブジェクト名を指定する場合は、スキーマ名を修

飾することができます。スキーマ名を省略すると、パラメータ search_path で指定されたスキーマ名が検索されます。

例 56　pgstattuple 関数の実行

```
postgres=# ¥x
Expanded display is on.
postgres=# SELECT * FROM pgstattuple('public.stat1');
-[ RECORD 1 ]------+--------
table_len          | 4440064
tuple_count        | 100100
tuple_len          | 3483282
tuple_percent      | 78.45
dead_tuple_count   | 0
dead_tuple_len     | 0
dead_tuple_percent | 0
free_space         | 20480
free_percent       | 0.46
```

□　pgstattuple 関数が示す情報とサポートされるオブジェクト

pgstattuple 関数は以下の情報を出力します。

表 62　pgstattuple 関数が返すレコード

列名	説明	備考
table_len	物理データ・サイズ	バイト
tuple_count	有効なタプル数	
tuple_len	有効なタプルの物理サイズ	バイト
tuple_percent	有効なタプルの割合	パーセント
dead_tuple_count	無効なタプル数	
dead_tuple_len	無効なタプルの物理サイズ	バイト
dead_tuple_percent	無効なタプルの割合	パーセント
free_space	総空き領域サイズ	バイト
free_percent	総空き領域割合	パーセント

pgstattuple 関数は以下のオブジェクトの統計情報を表示できます。

- TABLE

- UNLOGGED TABLE
- TEMP TABLE
- SEQUENCE
- MATERIALIZED VIEW
- INDEX (B-tree, Hash, GiST)
- TOAST TABLE
- TOAST INDEX

サポートされないオブジェクト（VIEW, GIN INDEX, SP-GiST INDEX 等）が指定された場合、以下のエラー・メッセージが表示されます。

例 57　サポートされないオブジェクトの指定

```
postgres=> SELECT * FROM pgstattuple('public.view1') ;
ERROR:  "view1" (view) is not supported
```

3.8.3 pgstatindex 関数 / pgstatginindex 関数

pgstatindex 関数および pgstatginindex 関数はインデックスの物理情報を出力する関数です。

□ pgstatindex 関数

pgstatindex 関数は B-tree インデックスの物理情報を出力する関数です。以下の情報を出力します。

表 63　pgstatindex 関数が返すレコード

列名	機能	備考
version	インデックスのバージョン	
tree_level	ツリーのレベル	
index_size	インデックスの総ページ数	root + leaf + internal + deleted + empty
root_block_no	ルートブロック番号	
internal_pages	内部ページ数	
leaf_pages	リーフページ数	
empty_pages	空ページ数	
deleted_pages	削除されたページ数	
avg_leaf_density	リーフページの平均密度	free / max_avail
leaf_fragmentation	リーフページの断片化割合	fragments / leaf_pages

以下は pgstatindex 関数の実行例です。

例 58　pgstatindex 関数の実行

```
postgres=# ¥x
Expanded display is on.
postgres=# SELECT * FROM pgstatindex('idx1_data2') ;
-[ RECORD 1 ]------+------
version            | 2
tree_level         | 0
index_size         | 8192
root_block_no      | 1
internal_pages     | 0
leaf_pages         | 1
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 24.83
leaf_fragmentation | 0
```

B-tree インデックス以外のインデックスを指定すると以下のエラーが出力されます。

例 59　サポートされないオブジェクトの指定

```
postgres=> SELECT * FROM pgstatindex('idx1_gin1') ;
ERROR:  relation "idx1_gin1" is not a btree index
```

□　pgstatginindex 関数

pgstatginindex 関数は GIN インデックスの物理情報を出力する関数です。以下の情報を出力します。

表 64　pgstatginindex 関数が返すレコード

列名	機能	備考
version	GIN バージョン番号	
pending_pages	待機中リスト内のページ数	
pending_tuples	待機中リスト内のレコード数	

以下は pgstatginindex 関数の実行例です。

例 60　pgstatginindex 関数の実行

```
postgres=# ¥x
Expanded display is on.
postgres=# SELECT * FROM pgstatginindex('idx1_gin1') ;
-[ RECORD 1 ]--+-----
version        | 2
pending_pages  | 3
pending_tuples | 1000
```

3.8.4 pg_relpages 関数

pg_relpages 関数はテーブルの物理ページ数（＝ブロック数）を返します。内部的には RelationGetNumberOfBlocksInfork 関数（src/backend/storage/buffer/bufmgr.c）をコールしています。

例 61　pg_relpages 関数の実行

```
postgres=# SELECT pg_relpages('data1') ;
 pg_relpages
-------------
         542
(1 row)
```

付録. 参考文献

付録 1. 書籍

PostgreSQL について参考になる書籍の情報です。

表 65 書籍の情報

書籍名	著者（敬称略）	出版社	備考
PostgreSQL 徹底入門 第 3 版	笠原 辰仁 北川 俊広 坂井 潔 坂本 昌彦 佐藤 友章 石井 達夫（監修）	翔泳社	
PostgreSQL 全機能バイブル	鈴木 啓修	技術評論社	
内部構造から学ぶ PostgreSQL 設計・運用計画の鉄則	勝俣 智成 佐伯 昌樹 原田 登志	技術評論社	
PostgreSQL 9.0 High Performance	Gregory Smith	PACKT	
PostgreSQL Replication	Zoltan Boszormenyi Hans-Jurgen Schonig	PACKT	
PostgreSQL 9 Admin Cookbook	Simon Riggs Hannu Krosing	PACKT	

付録 2. URL

PostgreSQL について参考になる URL の情報です。

表 66 参考になる URL

サイト	URL
PostgreSQL 日本語版ドキュメント	http://www.postgresql.jp/document/
PostgreSQL 公式ドキュメント	http://www.postgresql.org/docs/
PostgreSQL JDBC Driver	http://jdbc.postgresql.org/
PostgreSQL Internals	http://www.postgresqlinternals.org/index.php
PostgreSQL Deep Dive	http://pgsqldeepdive.blogspot.jp/
オープンソースデータベース標準教科書	http://www.oss-db.jp/ossdbtext/text.shtml
日本 PostgreSQL ユーザー会	https://www.postgresql.jp/
Let's Postgres	http://lets.postgresql.jp/
PostgreSQL エンタープライズ・コンソーシアム	https://www.pgecons.org/
EnterpriseDB	http://www.enterprisedb.com/
Michael Paquier - Open source developer based in Japan	http://michael.otacoo.com/
PostgreSQL は、SELECT もロックを獲得する	http://d.hatena.ne.jp/chiheisen/20100310/1268238033
PostgreSQL SQL チューニング入門 入門編	http://www.slideshare.net/MikiShimogai/sql-42453213
PostgreSQL SQL チューニング入門 実践編	http://www.slideshare.net/satoshiyamada71697/postgresql-sql
より深くオプティマイザとそのチューニング	http://www.slideshare.net/hayamiz/ss-42415350

変更履歴

変更履歴

版	日付	作成者	説明
1.0	16-Mar-2015	篠田典良@日本 HP	公開版を作成

以上