3.2 postgresql.conf ファイル

PostgreSQL全体の動作を制御する設定ファイルで、設定項目は大きく分けて、**表3-2**のカテゴリに分類されます。

各カテゴリには多くの設定項目がありますが、運用する際に検討や設定が必要なパラメータは限定されます(運用時に必要なパラメータや設定方針については第7章以降で説明します)。

3.2.1：設定項目の書式

設定項目は、項目名と項目値を「=」で繋いだ書式で記述します。「#」(ハッシュ記号)はコメントする際に利用し、#以降の行末までの記述は無視されます(**リスト3-1**)。

設定する値は、論理型、浮動小数点型、整数型、文字型、列挙型の5種類があります(**表3-3**)。メモリサイズや時間を指定するパラメータの場合、

表3-2　postgresql.confの主な設定カテゴリ

カテゴリ名	説明
接続と認証	接続、セキュリティ、認証の設定
資源の消費	共有メモリ、ディスク、ライタプロセスの設定
ログ先行書き込み	WAL 、チェックポイント、アーカイブの設定
レプリケーション	レプリケーションの設定
問い合わせ計画	問い合わせに対する実行計画の設定
エラー報告とログ取得	サーバログ出力に関する設定
実行時統計情報	統計情報の収集に関する設定
自動バキューム作業	自動バキュームに関する設定
クライアント接続デフォルト	接続したクライアントの挙動やロケールに関する設定
ロック管理	ロックやデッドロック検知の設定
バージョンとプラットフォーム	旧バージョンやプラットフォーム間の互換性に関する設定
エラー処理	エラーや障害発生時の設定

リスト3-1　設定項目の記述例

```
max_connections = 100      # 接続最大数を100に設定します
```


数字の後に単位を示す文字を続けて記述することで、簡易かつ読みやすい値を設定することができます(**表3-4**)。shared_buffersなど大きな値を設定する場合、単位を付与して設定の誤りを防ぎやすくします(**リスト3-2**)。また、小文字の「k」(キロ)は1000ではなく1024を示します。同様に、「M」(メガ)は1024の2乗、「G」(ギガ)は1024の3乗となります。なお、同じ設定項目を複数記述した場合、起動時にはエラーや警告は出力されず、後ろに記述されているほうが有効とみなされます(**コマンド3-1**)。

表3-3　設定できる型

型名	説明
論理型（boolean）	真偽値（on/off、true/false、yes/no、1/0）。大文字と小文字は区別しない
浮動小数点型（floating point）	小数点を含む数値。指数記号（e）を含む書式で記述することも可能
整数型（integer）	小数点を含まない数値
文字型（string）	任意の文字列。空白を含む場合は単一引用符で囲む必要がある。値に単一引用符を含む場合は、二重引用符（もしくは逆引用符）で囲む。空白を含まない文字列は単一引用符で囲む必要はない
列挙型（enum）	限定された値の集合。値の集合は項目ごとに異なる

表3-4　単位を指定する文字

指定対象	指定する文字	意味
メモリ	kB	キロバイト（1024）
	MB	メガバイト（1024^2）
	GB	ギガバイト（1024^3）
	TB	テラバイト（1024^4）
時間	ms	ミリ秒
	s	秒
	min	分
	h	時
	d	日

リスト3-2　値に単位を付与する例

```
shared_buffers = 128MB
```

5.2.2：一意性制約とNOT NULL制約

一意性制約とNOT NULL制約により、それぞれ列値に重複がないこと、NULLを含まないことを保証できます。なお、暗黙のB-treeインデックスは主キーだけでなく、一意性制約が定義された列にも作成されます(**コマンド5-2**)。

5.2.3：外部キー制約

複数のテーブル間でデータの整合性をとる(参照整合性と呼びます)場合、外部キーを使用します。外部キーをREFERENCESで指定することで、指定した先のテーブルに存在しない値を該当の列値として使用することができなくなります。

また、RESTRICT指定で参照先の行の削除を抑止したり、CASCADE指定で他のテーブルに依存している行を同時に削除することもできます。更新についても同様の指定が可能です(**コマンド5-3**〜**5-6**)。

外部キー制約の注意点

外部キー制約は複数のテーブル間でデータの整合をとるために非常に有効ですが、PostgreSQLで使う場合にはいくつか注意点があります。

コマンド5-2　暗黙的なインデックスの設定

```
=# CREATE TABLE foo (id1 int unique not null, id2 int unique, data1 int, ➥
data text); ↵
CREATE TABLE
=# ¥d foo ↵
                Table "public.foo"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id1    | integer |           | not null |
 id2    | integer |           |          |
 data1  | integer |           |          |
 data   | text    |           |          |
Indexes:
    "foo_id1_key" UNIQUE CONSTRAINT, btree (id1)
    "foo_id2_key" UNIQUE CONSTRAINT, btree (id2)
```


コマンド5-3　外部キー制約の例

```
=# ¥d ↵
           List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+----------
 public | order_items | table | postgres
 public | orders      | table | postgres
 public | products    | table | postgres
(3 rows)

=# ¥d products ↵
               Table "public.products"
   Column   |  Type   | Collation | Nullable | Default
------------+---------+-----------+----------+---------
 product_no | integer |           | not null |
 name       | text    |           |          |
 price      | integer |           |          |
Indexes:
    "products_pkey" PRIMARY KEY, btree (product_no)
Referenced by:
    TABLE "order_items" CONSTRAINT "order_items_product_no_fkey" FOREIGN ➥
KEY (product_no) REFERENCES products(product_no) ON DELETE RESTRICT

=# ¥d orders ↵
                  Table "public.orders"
     Column      |  Type   | Collation | Nullable | Default
-----------------+---------+-----------+----------+---------
 order_id        | integer |           | not null |
 shipping_address | text    |           |          |
Indexes:
    "orders_pkey" PRIMARY KEY, btree (order_id)
Referenced by:
    TABLE "order_items" CONSTRAINT "order_items_order_id_fkey" FOREIGN KEY ➥
(order_id) REFERENCES orders(order_id) ON DELETE CASCADE

=# ¥d order_items ↵
              Table "public.order_items"
   Column   |  Type   | Collation | Nullable | Default
------------+---------+-----------+----------+---------
 product_no | integer |           | not null |
 order_id   | integer |           | not null |
 quantity   | integer |           |          |
Indexes:
    "order_items_pkey" PRIMARY KEY, btree (product_no, order_id)
```

（次ページへ続く）

10.4.1：ホットスタンバイで実行可能なクエリ

ホットスタンバイでは**表10-8**の参照クエリのみが実行可能です。トランザクションIDの払い出しはされず、またWALに書き出されないため、更新処理は実行できません。これは、スタンバイでのMVCC（MultiVersion Concurrency Control；多版型同時実行制御）を保証できなくなるためです。

ベースバックアップの取得

スタンバイからpg_basebackupコマンドでベースバックアップを取得できます（PostgreSQL 9.2以降）。通常、ベースバックアップはpg_start_backupやpg_stop_backup関数で取得します。ホットスタンバイでは、これらの関数を直接実行できませんが、pg_basebackupコマンドは内部的には同等の処理でベースバックアップを転送します。このため、ホットスタンバイでもpg_basebackupコマンドでベースバックアップが取得できるようになっています。

スタンバイからpg_basebackupコマンドで取得したバックアップをリカバリするとき、hot_standbyを「on」にして任意の時刻やトランザクションIDを指定してリカバリすると、リカバリが完了した時点で一時停止します。これは、recovery.confファイルのrecovery_target_actionが「pause」（デフォルト値）の際の挙動であり、適切な位置にリカバリができたかを確認するための重要な機能ですが、一刻も早くリカバリをしたい場合には一時停止せずに進めたい場面が多々あります。一時停止しないようにするには、recovery_target_actionを「promote」にしてリカバリを実行します（PostgreSQL 9.4以前ではpause_at_recovery_targetを「off」にしてリカバリを実行します）。

なお、一時停止したリカバリを再開するにはpg_wal_replay_resume関数を実行します。そのほかにも、**表10-9**のような関数が用意されています。

表10-8　ホットスタンバイで実行可能なクエリ

コマンド	説明
SELECT、COPY TO	読み取りクエリ
DECLARE、FETCH、CLOSE	カーソル操作クエリ
SHOW、SET、RESET	パラメータ操作クエリ
BEGIN、COMMIT	DCL※コマンド
ACCESS SHARE、ROW SHARE、ROW EXCLUSIVE	いずれかを指定したLOCK TABLE文
PREPARE、EXECUTE、DEALLOCATE、DISCARD	準備済みステートメントを操作するクエリ
LOAD	ライブラリ読み込み操作

※Data Control Language（データ制御言語）

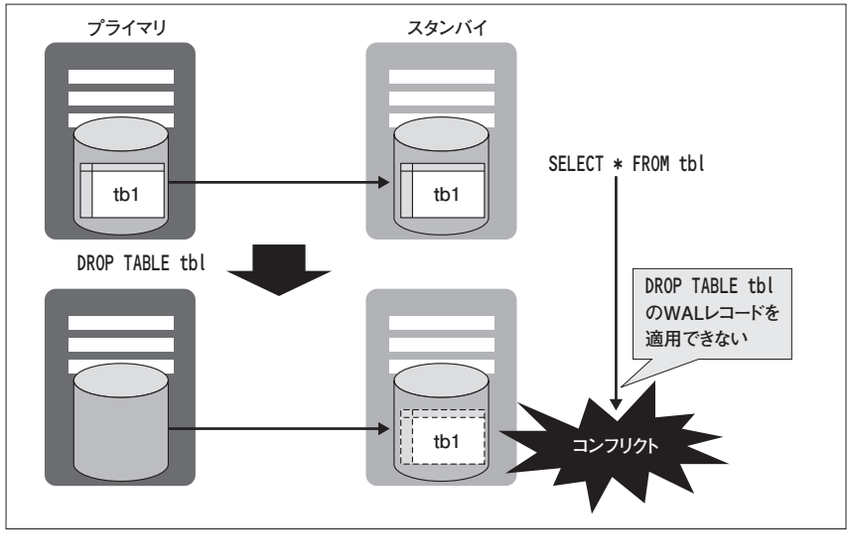
10.4.2：ホットスタンバイの弱点

ホットスタンバイの弱点は「コンフリクト」です。一貫性が確認されればすべての参照クエリがホットスタンバイで実行可能になるとはかぎりません。

表10-9　リカバリの停止／再開の関数

関数	説明
pg_is_wal_replay_paused()	リカバリが停止中であれば真を返す
pg_wal_replay_pause()	即座にリカバリを停止する
pg_wal_replay_resume()	リカバリ停止中であれば再開する

図10-5　コンフリクトの発生



文のオプションと同じ効果があります。

15.3 実行計画の構造

ここからは実行計画の中身について見ていきます。一般的に、ある問い合わせには同じ結果を導き出すための方法が、複数存在しています。プランナは統計情報や経験的な計算式に基づいてコストを計算し、可能なかぎり多くの組み合わせの中から、もっともコストの小さい実行計画を選択します。

EXPLAIN文は、プランナが導き出した最適な実行計画を出力します(**コマンド15-12**)。

実行計画では処理する単位を「ノード」と呼び、ノードは階層構造を持ちます。テキスト形式で表示される実行計画は1行目に最上位のノードが現れ、順に「**->**」の記号とインデントによって複数のノードが表れます。同じ階層のノードは同じインデントに揃えられています。基本的に、問い合わせの実行は実行計画の表示とは逆にもっとも深い階層のノードから順番に実行され、最上位ノードは一番最後に実行されます(**図15-1**)。一見複雑な構造

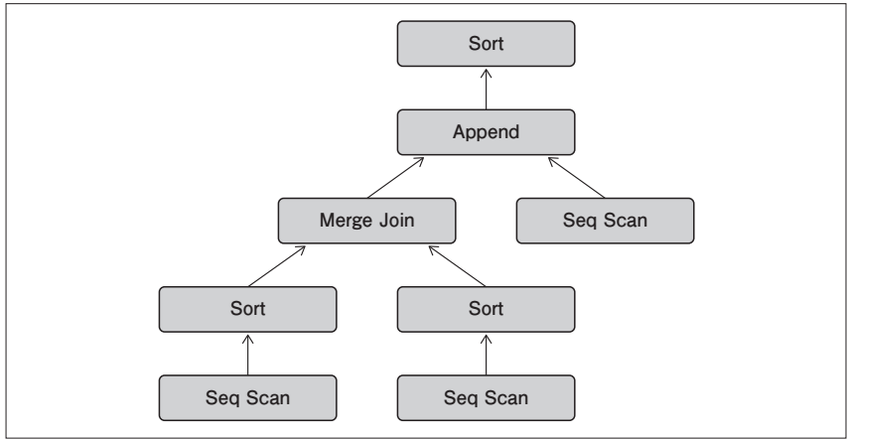
コマンド15-12　EXPLAIN文の出力例

```
=# EXPLAIN SELECT eid FROM dept JOIN emp USING ( did ) UNION ALL SELECT ア
eid FROM emp WHERE eid > 1000 ORDER BY eid; ↵
                              QUERY PLAN
-----------------------------------------------------------------------
 Sort  (cost=3.37..3.38 rows=5 width=4)
   Sort Key: emp.eid
   -> Append  (cost=2.13..3.31 rows=5 width=4)
        -> Merge Join  (cost=2.13..2.21 rows=4 width=4)
             Merge Cond: (dept.did = emp.did)
             -> Sort  (cost=1.05..1.06 rows=3 width=4)
                  Sort Key: dept.did
                  -> Seq Scan on dept  (cost=0.00..1.03 rows=3 width=4)
             -> Sort  (cost=1.08..1.09 rows=4 width=8)
                  Sort Key: emp.did
                  -> Seq Scan on emp  (cost=0.00..1.04 rows=4 width=8)
        -> Seq Scan on emp emp_1  (cost=0.00..1.05 rows=1 width=4)
             Filter: (eid > 1000)
(13 rows)
```


をしていても、階層構造を辿ることで実行計画の内容が理解できます。

続いて、ノードを「スキャン系」「複数のデータを結合」「データを加工」の3つに分類して、それぞれ代表的なものを説明します。

図15-1　実行計画の木構造と実行順序



15.3.1：スキャン系ノード

データを取り出す役割を担うノードには「シーケンシャルスキャン」や「インデックススキャン」があります。データを取り出すノードは通常、実行計画のもっとも深い階層に現れ、一番最初に実行されるノードです。実行計画におけるスキャン系ノードの代表的なものは**表15-3**のとおりです。スキャン系ノードでは、どのテーブルに対してスキャンしたかと、データを選別するフィルタ条件が表示されるので[注1]、どのようにデータを取り出すかを読み取ることができます。

シーケンシャルスキャン（Seq Scan）

もっともシンプルなシーケンシャルスキャン(Seq Scan)の実行計画は**コマ**

注1　Function Scanの場合は、テーブルの代わりに関数名を表示します。EXPLAIN文にVERBOSEオプションを付けると関数の入力値などの詳細な情報も表示します。