PostgreSQL でストアドプロシージャを使用する

技術を知る

FUJITSU

- [ ] 導入／環境設定
- [x] 移行
- [ ] 性能
- [ ] チューニング
- [ ] バックアップ／リカバリー
- [ ] 冗長化／負荷分散
- [ ] 監視
- [ ] データ連携
- [ ] 災害対策
- [ ] 豆知識

PostgreSQL 11 では、ストアドプロシージャが機能追加され、ストアドプロシージャがサポートされている Oracle など他社データベースからの移行性が向上しました。ここでは、PostgreSQL 11 のストアドプロシージャについて解説します。

1. ストアドプロシージャとは

ストアドプロシージャは、あらかじめデータベースを操作する一連の処理をデータベースサーバーに格納したものです。さまざまなクライアントやアプリケーションからストアドプロシージャを呼び出すだけで一連の処理が実行できます。異なるアプリケーションからデータベースに対して、汎用的な処理を実行するような場合（1 日の売り上げを集計するなど）に、データベースサーバー上のストアドプロシージャを呼び出すだけで処理することができます。
ストアドプロシージャは、データベースシステムごとに記述できる言語が異なります。PostgreSQL では、下記の言語で作成可能です。

- 手続き言語（PL/pgSQL, PL/Python, PL/Tcl, PL/Perl）
- SQL
- C 言語

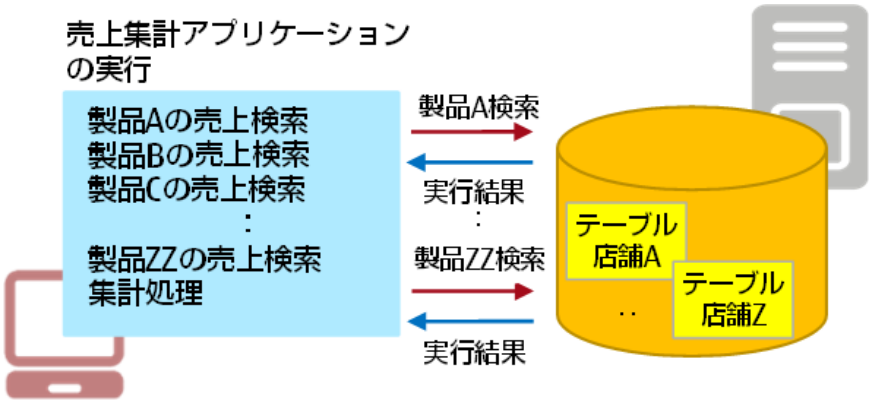
1.1 ストアドプロシージャのメリット

ストアドプロシージャを利用するメリットを説明します。

処理の高速化

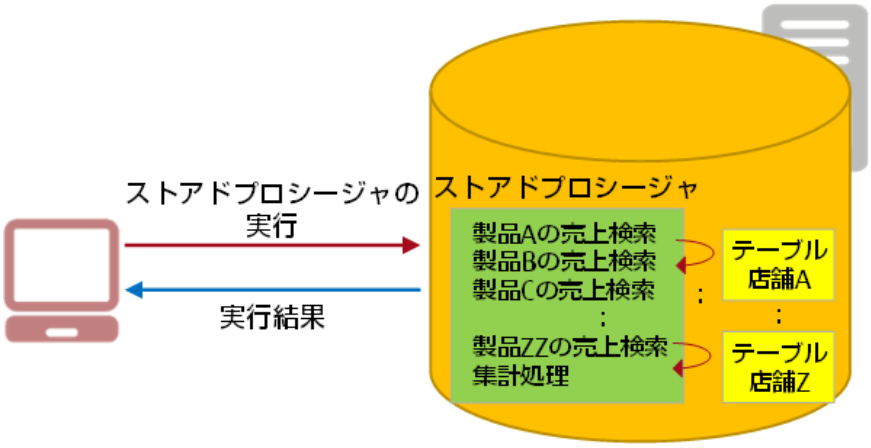
ストアドプロシージャでは、1 回の要求で複数の SQL 文が実行できます。
クライアントからアプリケーションを実行する場合、通常は、クライアントからサーバーに 1 つの SQL を送信し結果を受信、また次の SQL を送信というように、クライアントとサーバー間で SQL の送信と受信が繰り返されるため、処理に時間がかかります。このような場合に、ストアドプロシージャを利用すると、クライアントから 1 回呼び出すだけで複数の SQL 処理が可能となり、クライアントには、最終的な実行結果だけが送付されるため、ネットワーク負荷がかからず処理時間も短縮できます。

- **ストアドプロシージャ未使用**



 © 2019 Fujitsu Limited

- **ストアドプロシージャ使用**

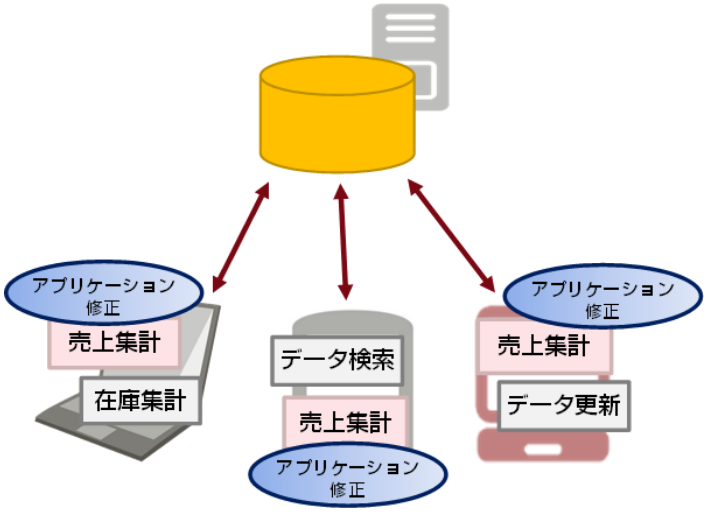


ただし、複数のクライアントから一斉にストアドプロシージャが呼び出される場合や比較的大規模なストアドプロシージャを利用する場合など、データベースサーバーの負荷が大きくなってしまうときがあります。性能面を考慮し、どの処理をストアドプロシージャにすると効果的なのか見極めが必要です。

汎用的な処理の共通化

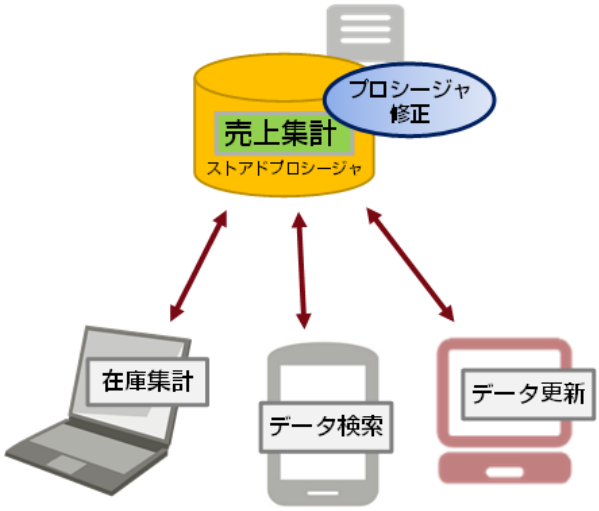
各アプリケーションでは、データベースに対し同様の処理を実行している場合があります。このような同様の処理を分離し、データベースにストアドプロシージャとして格納することで、処理を共通化することができます。処理が共通化されると、アプリケーション開発で新たな作りこみをする必要がないので、開発の効率化が期待できます。また、処理に変更が発生した場合、共通化しているストアドプロシージャの修正のみで良いため、保守性も向上します。
「現時点での在庫状況を集計する」や「1 日の売り上げを集計する」などは、汎用的に実行される処理なのでストアドプロシージャが向いています。

- **ストアドプロシージャ未使用**



 © 2019 Fujitsu Limited

- **ストアドプロシージャ使用**



1.2 ストアドプロシージャの特徴

下表に、ストアドプロシージャの使い方に関する主な特徴を示します。

パラメーター	IN パラメーター、INOUT パラメーター、VARIADIC パラメーターが指定可能 OUT パラメーターは利用不可
RETURNS 句	なし（INOUT パラメーターで値の返却が可能）
呼び出し方法	CALL 文で呼び出し
トランザクション制御	COMMIT、ROLLBACK 使用可

2. ストアドプロシージャの定義例と実行例

ストアドプロシージャの定義例と実行例を説明します。以降の定義例では、手続き言語として、PL/pgSQL を使用しています。

定義例

ストアドプロシージャは、CREATE PROCEDURE 文で定義します。下記の例では、パラメーターを INOUT に指定することで、値を返しています。

```
CREATE PROCEDURE proc1(INOUT p1 TEXT)  -- INOUT パラメーターを指定します
AS $$
BEGIN
    p1 := '!! ' ||  p1 ||' !!';
    RAISE NOTICE 'Procedure Parameter: %', p1 ;
END;
$$
LANGUAGE plpgsql ;
```

 © 2019 Fujitsu Limited

実行例

ストアドプロシージャは、CALL 文で実行します。

```
mydb=# CALL proc1 ('Stored Procedure supported in PostgreSQL 11');
NOTICE:  Procedure Parameter: !! Stored Procedure supported in PostgreSQL 11 !!
                      p1
-------------------------------------------------------
!! Stored Procedure supported in PostgreSQL 11 !!
(1 行)
```

2.1 トランザクション制御を実装したストアドプロシージャの例

ストアドプロシージャでは、COMMIT / ROLLBACK によるトランザクション制御を実装できます。
COMMIT / ROLLBACK を記述した以下のようなストアドプロシージャを作成し、COMMIT / ROLLBACK の動作を確認してみます。以下の例では、整数 FOR ループ内で、i は 1 から 100 の値を取り、10 の倍数の場合には処理を COMMIT、10 の倍数でない場合には処理を ROLLBACK します。

```
CREATE PROCEDURE proc2()
LANGUAGE plpgsql
AS $$
BEGIN
    FOR i IN 1..100 LOOP
        INSERT INTO samp(number) VALUES(i);
        IF i % 10 = 0 THEN  -- i/10 の余りがゼロか否かの判定
            COMMIT;         -- 10 の倍数の場合：COMMIT する
        ELSE
            ROLLBACK;       -- 10 の倍数でない場合：ROLLBACK する
        END IF;
    END LOOP;
END
$$;
```

実際にストアドプロシージャを実行し、その実行結果を見てみましょう。10 の倍数だけが samp テーブルに挿入されており、COMMIT / ROLLBACK が機能していることがわかります。

```
mydb=# CREATE TABLE samp(number integer);
CREATE TABLE



mydb=# CALL proc2();
CALL
mydb=# SELECT * FROM samp;
number
--------
     10
     20
     30
     40
     50
```

 © 2019 Fujitsu Limited

```
     60
     70
     80
     90
    100
(10 行)
```

また、ストアドプロシージャ内から別のストアドプロシージャの呼び出し（入れ子）も可能です。

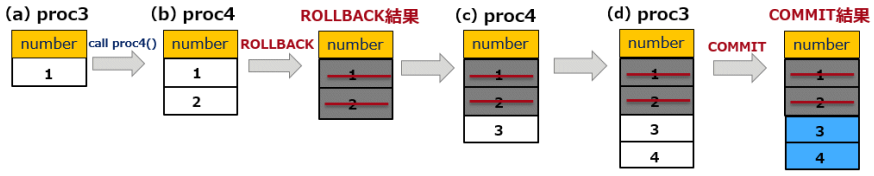
```
CREATE PROCEDURE proc3() LANGUAGE plpgsql AS $$
BEGIN
 INSERT INTO samp VALUES(1);   --  (a)
 CALL proc4();
 INSERT INTO samp VALUES(4);   --  (d)
 COMMIT;
END;
$$;

CREATE PROCEDURE proc4() LANGUAGE plpgsql AS $$
BEGIN
 INSERT INTO samp VALUES(2);   --  (b)
 ROLLBACK;
 INSERT INTO samp VALUES(3);   --  (c)
END;
$$;
```

ストアドプロシージャを実行し、実行結果を見てみます。上記の例では、proc3 から proc4 を呼び出していますが、proc4 はサブトランザクションとならず、呼び出し元の処理（a）が proc4 で ROLLBACK され、無効になっていることがわかります。

```
mydb=# DELETE FROM samp;
DELETE 10
mydb=# CALL proc3();
CALL
mydb=# SELECT * FROM samp;    -- proc4 の ROLLBACK により、(a) および (b) の行が挿入されていません。
number
-------
    3
    4
(2 行)
```

以下は、上記を実行したときの samp テーブルの状態遷移です。



© 2019 Fujitsu Limited

3. Oracle から PostgreSQL へのストアドプロシージャの移行

Oracle のストアドプロシージャを PostgreSQL に移行する場合、PostgreSQL 10 まではファンクションに移行する必要がありましたが、PostgreSQL 11 ではストアドプロシージャが利用可能となり、移行がしやすくなりました。
Oracle のストアドプロシージャを PostgreSQL に移行する例を示します。下記の例では、赤字部分の修正のみで移行できます。

Oracle

```
CREATE OR REPLACE PROCEDURE proc5(id IN INTEGER) IS
  count INTEGER;
BEGIN
  LOCK TABLE samp2 IN EXCLUSIVE MODE;
  SELECT count(*) INTO count FROM samp2;
  IF count >= 10 THEN
    COMMIT;
    raise_application_error(-20000,
      'Unable to create a new data.');
  END IF;
  BEGIN
    INSERT INTO samp2 VALUES (id);
  EXCEPTION
    WHEN dup_val_on_index THEN NULL;
  END;
  COMMIT;
END;
/
```

PostgreSQL

```
CREATE OR REPLACE PROCEDURE proc5(id IN integer) AS $$
DECLARE
  count integer;
BEGIN
  LOCK TABLE samp2 IN EXCLUSIVE MODE;
  SELECT count(*) INTO count FROM samp2;
  IF count >= 10 THEN
    COMMIT;
    RAISE EXCEPTION 'Unable to create a new data.';
  END IF;
  BEGIN
    INSERT INTO samp2 VALUES (id);
  EXCEPTION
    WHEN unique_violation THEN      -- 何もしない
  END;
  COMMIT;
END;
$$ LANGUAGE plpgsql;
```

- （備考）　上記の例では、samp2 テーブルのデータ件数を事前に判定し、すでに 10 件以上登録済みの場合は、メッセージを出力します。10 件に満たない場合は、パラメーターで渡されたデータを samp2 テーブルに挿入します。

参考

Fujitsu Enterprise Postgres には、他社データベースからの移行支援サービスがあります。移行支援サービスでは、ストアドプロシージャをはじめとするアプリケーションの移行アセスメントから資産移行、導入までのトータルなサポートを提供しています。

3.1 トランザクション制御に関する制約

PostgreSQL と Oracle では、トランザクション制御の仕様が異なります。PostgreSQL では、トランザクション制御に関して以下の制約がありますので、Oracle からの移行時に注意が必要です。

ファンクション内や明示的なトランザクション内からストアドプロシージャを呼び出した場合、そのストアドプロシージャ内で COMMIT / ROLLBACK が実行されるとエラーになります。

```
mydb=# BEGIN;
BEGIN
mydb=# CALL p_samp3();
ERROR:不正なトランザクション終了
```

明示的なトランザクション開始

p_samp3でCOMMIT/ROLLBACKが実行されると、エラーとなる

BEGIN と EXCEPTION の間に、COMMIT / ROLLBACK は記述できません。この場合、EXCEPTION が必要な処理を BEGIN から END でサブブロック化し、COMMIT / ROLLBACK はサブブロックの外に記述します。

© 2019 Fujitsu Limited

■誤った記述例

```
CREATE PROCEDURE p_samp4()
  LANGUAGE plpgsql AS $$
    BEGIN
      LOOP
        IF ~ THEN
          処理1
          COMMIT;
        END IF;
      END  LOOP;

    EXCEPTION
      処理A
      COMMIT;

  END;
$$;
```

記述できない

■正しい記述例

```
CREATE PROCEDURE p_samp4()
  LANGUAGE plpgsql AS $$
    BEGIN
      LOOP
        IF ~ THEN
          BEGIN
            処理1
          EXCEPTION
            処理A
            COMMIT;
            RETURN;
          END;
          COMMIT;
        END IF;
      END  LOOP;

  END;
$$;
```

サブブロック化

ここでは、PostgreSQL のストアドプロシージャを解説しました。アプリケーション開発の効率化と処理の高速化を実現するため、ストアドプロシージャの使用を検討してください。

2019 年 5 月 31 日

© 2019 Fujitsu Limited