

SQL 移行調査編

製作者
担当企業名 SRA OSS, Inc. 日本支社

改訂履歴

版	改訂日	変更内容
1.0	2013/04/22	初版
1.1	2017/06/26	・「2.3.1. NULL と空文字列」で使用している関数名を修正 (ISNULL → NULLIF)。
1.2	2018/04/20	以下は修正 ・「2.2.1 MERGE 文」に INSERT ~ ON CONFLICT を追加 ・「2.2.2 ビューに対する更新」の記載を変更 以下は節を追加 ・「2.2.5 UPDATE 文の別名」 ・「2.3.4 制約チェックのタイミング」 ・「2.3.5 除算を含む計算」 ・「2.4.4. トランザクション分離レベル:リードコミテッドでの実行差異」

ライセンス



本作品は CC-BY ライセンスによって許諾されています。

ライセンスの内容を知りたい方は <http://creativecommons.org/licenses/by/2.1/jp/> でご確認ください。

文書の内容、表記に関する誤り、ご要望、感想等につきましては、PGECcons のサイトを通じてお寄せいただきますようお願いいたします。

サイト URL <https://www.pgecons.org/contact/>

Microsoft、Windows Server、SQL Server、米国 Microsoft Corporation の米国及びその他の国における登録商標または商標です。
 Oracle は、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。
 PostgreSQL は、PostgreSQL Community Association of Canada のカナダにおける登録商標およびその他の国における商標です。
 その他、本資料に記載されている社名及び商品名はそれぞれ各社が商標または登録商標として使用している場合があります。

はじめに

■本資料の目的

本資料は、異種 DBMS から PostgreSQL へ SQL を移行する作業の難易度およびボリュームの事前判断と、実際に書き換えを行う際の参考資料として利用されることを想定しています。

■本資料で記載する範囲

本資料では、移行元の異種 DBMS として Oracle Database および Microsoft SQL Server を想定し、これらの DBMS から PostgreSQL へ SQL を移行する際に書き換えが必要である箇所とその書き換え方針について、DML とトランザクション処理を中心に記載します。スキーマ、ストアドプロシージャ、組み込み関数に関する SQL については本資料では取り扱っていません。これらに関しては、それぞれ「スキーマ移行調査編」、「ストアドプロシージャ移行調査編」、「組み込み関数移行調査編」を参照してください。

■本資料で扱う用語の定義

資料で記述する用語について以下に定義します。

表 1: 用語定義

No.	用語	意味
1	DBMS	データベース管理システムを指します。ここでは、PostgreSQL および異種 DBMS の総称として利用します。
2	異種 DBMS	PostgreSQL ではない、データベース管理システムを指します。本資料では、Oracle Database、Microsoft SQL Server が該当します。
3	Oracle	データベース管理システムの Oracle Database を指します。
4	SQL Server	データベース管理システムの Microsoft SQL Server を指します。

■本資料で扱う DBMS およびツール

本書では以下の DBMS を前提にした調査結果を記載します。

表 2: 本書で扱う DBMS

DBMS 名称	バージョン
PostgreSQL	9.6.5
Oracle Database	11gR2 11.2.0.2.0
Microsoft SQL Server	2008 R2

目次

1. SQL 移行調査の概要.....	5
1.1. 対象とする SQL について.....	5
1.2. SQL の差異と書き換え方針について.....	5
1.3. 標準 SQL について.....	5
2. Oracle から PostgreSQL への移行.....	6
2.1. SELECT 文.....	6
2.2. 更新系.....	9
2.3. その他の書き換え.....	10
2.4. トランザクション.....	12
3. SQL Server から PostgreSQL への移行.....	14
3.1. SELECT 文.....	14
3.2. 更新系.....	14
3.3. その他の書き換え.....	16
3.4. トランザクション.....	17
4. 別紙一覧.....	19

1. SQL 移行調査の概要

本資料では、Oracle Database (以下、Oracle と呼称) および Microsoft SQL Server (以下、SQL Server と呼称) から PostgreSQL へアプリケーションを移行する際に問題となる SQL の差異とその書き換え方針について記載します。本章では調査内容の概要について説明します。

1.1. 対象とする SQL について

本資料では、アプリケーションプログラムの中で使用される頻度が高いと思われる DML (SELECT, INSERT, UPDATE, DELETE など)、およびトランザクション処理関連 SQL を対象としています。その他のスキーマ、ストアードプロシージャに関する SQL と組み込み関数の差異および移行方針については本資料では取り扱っていません。これらについては、それぞれ「スキーマ移行調査編」、「ストアードプロシージャ移行調査編」、「組み込み関数移行調査編」を参照してください。

1.2. SQL の差異と書き換え方針について

異種 DBMS と PostgreSQL では SQL の構文や仕様に違いがあるため、移行元の SQL 文が移行先の PostgreSQL では動作しない場合があります。本資料では、そのような SQL を書き換え、移行先の PostgreSQL にて同等の機能で動作させる際の参考となる方針を記載しています。ただし、これは書き換える前後で完全に同じ動作を保証するものではありません。また、本資料は異種 DBMS と PostgreSQL の間に生じる全ての SQL の差異および書き換え方針を網羅したのではなく、本資料の内容以外の書き換えが必要になるケースも存在します。さらに、ケースによっては SQL 文の書き換えのみではなく SQL 文を発行するアプリケーション側での対応が適切な場合もあるかもしれません。実際の移行の際はこの可能性も含めて検討する必要があります。

本資料で対象とする SQL の PostgreSQL, Oracle, SQL Server の3つの DBMS における差異は別紙「SQL 差異表」にまとめましたので参照してください。各 SQL 機能への対応を○で、非対応を×で表し、備考欄にその他の参考となる情報を記載してあります。

1.3. 標準 SQL について

標準 SQL とは、SQL の国際標準規格であり、公式な名称は、ISO/IEC 9075 “Database Language SQL” です。2013 年 3 月現在、2011 年に改定されたものが最新版であり、これは SQL:2011 と呼ばれています。

PostgreSQL の開発では最新の標準 SQL に準拠しようとしており、実際に PostgreSQL 9.2 は SQL:2011 の主な機能のほとんどをサポートしています。SQL:2011 の機能の内、何がサポートされており、何がサポートされていないのかについては、PostgreSQL のドキュメントから知ることができます。また標準 SQL に対して PostgreSQL が独自の拡張を加えた機能についてもドキュメントの SQL リファレンスに記載されています。

以後、本資料で「標準 SQL」といった場合には SQL:2011 のことを指します。

別紙「SQL 差異表」には、各 SQL 機能が標準 SQL に準拠しているかの情報を付与しています。本資料および SQL 差異表の作成にあたり、各機能が標準 SQL に準拠しているかどうかの判断は PostgreSQL のドキュメントを参考としました。

2. Oracle から PostgreSQL への移行

本章では Oracle から PostgreSQL へ移行の際に生じる SQL の書き換え方針を紹介します。

2.1. SELECT 文

2.1.1. Oracle 独自形式の外部結合

Oracle には独自の外部結合演算子「(+)」が存在します。この演算子を使った結合は、標準 SQL 準拠の LEFT (RIGHT) OUTER JOIN 構文を用いて以下のように書き換えます。

【テーブル foo, bar の右外部結合】

Oracle	PostgreSQL
SELECT * FROM foo, bar WHERE foo.id = bar.id (+)	SELECT * FROM foo LEFT OUTER JOIN bar ON foo.id = bar.id

【テーブル foo, bar の左外部結合】

Oracle	PostgreSQL
SELECT * FROM foo, bar WHERE foo.id (+) = bar.id	SELECT * FROM foo RIGHT OUTER JOIN bar ON foo.id = bar.id

また、以下のように、外部結合演算子 (+) と UNION 句を併用して完全外部結合を行っている場合があります。このようなクエリは FULL OUTER JOIN を用いて書き換えます。

【テーブル foo, bar の完全外部結合】

Oracle	PostgreSQL
SELECT * FROM foo, bar WHERE foo.id = bar.id (+) UNION SELECT * FROM foo, bar WHERE foo.id (+) = bar.id	SELECT * FROM foo FULL OUTER JOIN bar ON foo.id = bar.id

2.1.2. DUAL 表

Oracle では SELECT 文の FROM 句を省略できないため、表を必要としない処理の場合には DUAL 表が使われます。PostgreSQL では FROM 句が省略可能であるので DUAL 表は存在しません。SELECT 文より「FROM DUAL」を取り除く必要があります。

【現在日時を表示】

Oracle	PostgreSQL
SELECT current_timestamp FROM DUAL	SELECT current_timestamp

あるいは、該当する全ての SELECT 文を書き換える代わりに、以下のような1件のレコードのみを持つ dual テーブルを PostgreSQL 側で定義しておく方法もあります。

【dual テーブルの定義】

<pre>CREATE TABLE dual (dummy VARCHAR(1)); INSERT INTO dual VALUES ('X'); SELECT current_timestamp FROM DUAL;</pre>
--

2.1.3. ROWNUM 擬似列

Oracle では検索結果の行番号を取得するのに ROWNUM 擬似列を用いることができます。Oracle で検索結果の表示件数を制限する場合には、これを用いるのが一般的です。しかし、ROWNUM 擬似列は Oracle 固有の機能であり、PostgreSQL には存在しません。

PostgreSQL では、標準 SQL の Window 関数の1つである row_number()により行番号の取得が可能です。これを用いると、ROWNUM 擬似列を用いた表示件数制限は以下のように書き換えることができます。

【tbl テーブルのデータを id で昇順ソートし、最初から 10 行のレコードを取得する】

Oracle	PostgreSQL
<pre>SELECT * FROM (SELECT * FROM tbl ORDER BY id) WHERE ROWNUM <= 10</pre>	<pre>SELECT * FROM (SELECT row_number() OVER (ORDER BY id) AS rownum, id FROM tbl) AS t WHERE rownum <= 10</pre>

PostgreSQL 独自の機能である LIMIT、OFFSET を用いると、より簡素な SQL 文で表示件数を制御することも可能です。

【tbl テーブルのデータを id で昇順ソートし、11 番目から 15 番目までの 5 行のレコードを取得する (LIMIT)】

<pre>SELECT * FROM tbl ORDER BY id LIMIT 5 OFFSET 10</pre>
--

また、これと同じ処理は、標準 SQL 準拠である FETCH 句を用いて書き換えることもできます。標準 SQL への準拠を重視する場合は FETCH 句を用いるのがよいでしょう。

【tbl テーブルのデータを id で昇順ソートし、11 番目から 15 番目までの 5 行のレコードを取得する (FETCH)】

<pre>SELECT * FROM tbl ORDER BY id OFFSET 10 ROWS FETCH FIRST 5 ROWS ONLY</pre>

2.1.4. 階層型問い合わせ

Oracle 独自の機能である階層型問い合わせを用いると、階層構造となっているデータから階層順にデータを取り出すことができます。階層構造データとは、例えば表 2.1 に示した staff テーブルのような上司と部下の関係です。この構造をツリーで表すと図 2.1 のようになります。

表 2.1: staff テーブル

ID	NAME	MANAGER_ID
1	John	
2	Paul	1
3	Anna	1
4	Peter	2
5	Steve	4
6	Ken	3
7	Bob	3

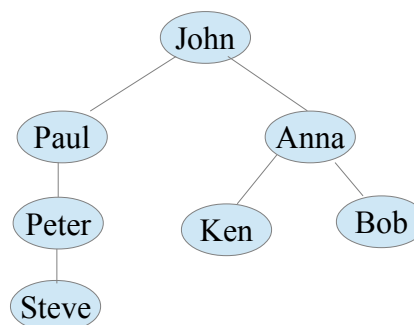
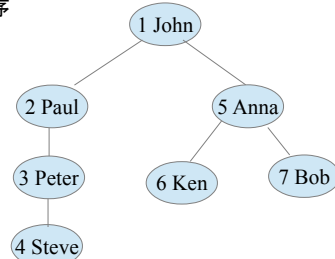
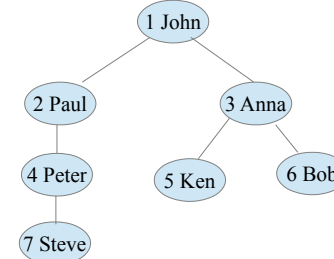


図 2.1: staff テーブルの階層構造

Oracle の階層型問い合わせでは START BY 句でルートとなるデータを、CONNECT BY 句でノード間の親子関係を指定すると、ルートから辿れるデータを「深さ優先探索」の順に取り出すことができます。PostgreSQL でこれと同様の機能は、標準 SQL の WITH 句を使用した再帰的問い合わせを用いることで実現可能です。ただし、再帰型問い合わせを用いた場合には、データ取り出し順序が Oracle とは異なって「幅優先探索」となることに注意してください。

【staff テーブルに対する階層問い合わせ】

Oracle	PostgreSQL
<pre>SELECT name FROM staff START WITH name = 'John' CONNECT BY manager_id = PRIOR id</pre>	<pre>WITH RECURSIVE rec (id, name, manager_id) AS (SELECT id, name, manager_id FROM staff WHERE name = 'John' UNION ALL SELECT staff.id, staff.name, staff.manager_id FROM staff, rec AS prior WHERE staff.manager_id = prior.id) SELECT name FROM rec</pre>
結果: 深さ優先順序 1. John 2. Paul 3. Peter 4. Steve 5. Anna 6. Ken 7. Bob 	結果: 幅優先順序 1. John 2. Paul 3. Anna 4. Peter 5. Ken 6. Bob 7. Steve 

また、標準 SQL 準拠ではありませんが、connectby 関数を用いると Oracle の CONNECT BY と同様に深さ優先順序の階層問い合わせが可能です。connectby 関数を使用するには PostgreSQL の追加モジュールの contrib/tablefunc をデータベースにインストールする必要があります。

【staff テーブルに対する階層問い合わせ (connectby 関数)】

<pre>SELECT name FROM connectby('staff', 'id', 'manager_id', 'id', '1', 0) AS t (id int, manager_id int, level int, pos int) JOIN staff ON staff.id=t.id ORDER BY pos</pre>

2.1.5. UNIQUE による重複行の除去

Oracle では検索結果から重複行を取り除く際に UNIQUE を使うことができます。これは標準 SQL の DISTINCT と同じ働きをします。PostgreSQL ではこのような UNIQUE の使い方はできないので、DISTINCT に書き直します。

【重複する行の除去】

Oracle	PostgreSQL
SELECT UNIQUE * FROM tbl	SELECT DISTINCT * FROM tbl

2.1.6. MINUS 演算子による差集合計算

Oracle で検索結果の差集合を求める場合に MINUS 演算子を使いますが、この演算子は Oracle 独自のもので PostgreSQL には存在しません。PostgreSQL は MINUS 演算子と同等の機能を持つ EXCEPT 演算子に対応しています。なお、EXCEPT 演算子は SQL 標準に従ったものです。

2.1.7. FROM 句中のサブクエリの別名

Oracle ではサブクエリには別名は必須ではありませんが、PostgreSQL では FROM 句の中のサブクエリには別名を付ける必要があります。

【FROM 句の中のサブクエリの別名は必須】

SELECT * FROM (SELECT * FROM tbl) AS sub
--

2.2. 更新系

2.2.1. MERGE 文

MERGE 文はテーブルに既存の行がある場合には更新を、ない場合には新規に挿入を行う SQL 文です。標準 SQL に従ったものですが PostgreSQL はこれに対応していません。PostgreSQL では INSERT~ON CONFLICT (9.5 以降) または WITH 句の中で UPDATE 文を用いることにより、これと同等の機能を実現することができます。なお、INSERT~ON CONFLICT、更新を含む WITH 句は PostgreSQL 独自の拡張です。

【diff テーブルの値を master テーブルにマージする。

(master テーブルに ID が一致する行があったら、diff.val を master.val に足し加える。

ID が一致する行がない場合には、diff の内容を master に新規登録する。)]

Oracle	PostgreSQL
<pre>MERGE INTO master USING diff ON master.id = diff.id WHEN MATCHED THEN UPDATE SET master.val = master.val + diff.val WHEN NOT MATCHED THEN INSERT VALUES (diff.id, diff.val)</pre>	<p>① INSERT~ON CONFLICT の場合</p> <pre>INSERT INTO master VALUES (diff.id, diff.val) ON CONFLICT (id) DO UPDATE SET master.val = master.val + diff.val</pre> <p>②更新を含む WITH 句</p> <pre>WITH inpt AS (SELECT * FROM diff), updt AS (UPDATE master.val = master.val + inpt.val FROM inpt WHERE master.id = inpt.id RETURNING master.id) INSERT INTO master (SELECT * FROM inpt WHERE id NOT IN (SELECT id FROM updt))</pre>

2.2.2. ビューに対する更新

PostgreSQL9.3 以降ではビューに対する更新が可能です。ただし、Oracle 同様に全てのビューに対して更新可能なわけではなく、集約関数、集合を使用していない、複数テーブルをマージしていないなどの条件があり、満たせない場合は読み取り専用のビューとなります。作成したビューが更新可能かどうかは、情報スキーマの views ビューを参照することで確認できます。

【更新可能ビューの確認】

```
select table_name, is_updatable, is_insertable_into from information_schema.views where table_name = 'v_emp';
```

table_name	is_updatable	is_insertable_into
v_emp (1 行)	YES	YES

2.2.3. マルチテーブル INSERT

Oracle の INSERT 文では複数のテーブルに対してデータを挿入することが可能です。しかし PostgreSQL の INSERT 文にはそのような機能はなく、個々のテーブルに毎に INSERT 文を実行する必要があります。

ただし、Oracle のマルチテーブル INSERT を利用すると1つのテーブルに複数行を挿入することが可能であり、そのようなクエリは標準 SQL に従い以下のように書き換え可能です。

【テーブル tbl にレコードを 3 行挿入する】

Oracle	PostgreSQL
<pre>INSERT ALL INTO tbl VALUES (1, 'one') INTO tbl VALUES (2, 'two') INTO tbl VALUES (3, 'three') SELECT * FROM DUAL</pre>	<pre>INSERT INTO tbl VALUES (1, 'one'), (2, 'two'), (3, 'three')</pre>

2.2.4. DELETE 文の FROM

Oracle の DELETE 文では FROM キーワードが省略可能ですが、PostgreSQL では省略することはできません。もし FROM が省略されている場合には書き足す必要があります。

【DELETE 文の FROM は省略できない】

Oracle	PostgreSQL
DELETE tbl WHERE id = 2	DELETE FROM tbl WHERE id = 2

2.2.5. UPDATE 文の別名

PostgreSQL の UPDATE 文で SET 句で更新対象列に別名を付けるとエラーとなります。別名が記載されている場合は取り除きます。

【UPDATE 文の別名】

Oracle	PostgreSQL
UPDATE emp AS e SET e.ename = 'XXXXXXXX'	UPDATE emp AS e SET ename = 'XXXXXXXX' 別名 e を取り除く

2.3. その他の書き換え

2.3.1. NULL と空文字列

Oracle では空の文字列は NULL と同値として扱われますが、これは標準 SQL に準拠したものではありません。PostgreSQL においてはこれらは区別されます。文字列の結合や検索をふくむクエリの実行結果が移行の前後で異なる場合があります。また、Oracle ではテーブルに空文字列を挿入すると NULL に自動変換されますが、PostgreSQL では変換されず空文字列のまま格納されます。

PostgreSQL で Oracle と同じように空文字列を NULL とみなさせるには、NULLIF 関数を使って変換する方法があります。

【空文字列を NULL とみなす検索】

SELECT * FROM staff WHERE NULLIF(name, '') IS NOT NULL
--

2.3.2. REGEXP_LIKE 条件による正規表現マッチング

Oracle は REGEXP_LIKE 条件を使って正規表現マッチングを行います。この条件は、POSIX 正規表現規格に準拠しています。PostgreSQL で POSIX 正規表現のマッチングを行う場合には正規表現マッチ演算子(表 2.2)を用います。

表 2.2: 正規表現マッチ演算子

演算子	説明	例 (結果はすべて真)
~	正規表現に一致、大文字小文字の区別あり	'thomas' ~ '.*thomas.*'
~*	正規表現に一致、大文字小文字の区別なし	'thomas' ~* '.*Thomas.*'
!~	正規表現に一致しない、大文字小文字の区別あり	'thomas' !~ '.*Thomas.*'
!~*	正規表現に一致しない、大文字小文字の区別なし	'thomas' !~* '.*vadim.*'

【POSIX 正規表現を使って p で始まるか e が2回現れる名前を検索】

Oracle	PostgreSQL
SELECT * FROM staff WHERE REGEXP_LIKE(lower(name), '^p(e.*){2}')	SELECT * FROM staff WHERE name ~* '^p(e.*){2}'

2.3.3. 比較演算子 ^=

Oracle では不等を表す演算子に ^= が使用できますが、この演算子は PostgreSQL には存在しません。◇ 演算子で置き換えてください。

2.3.4. 制約チェックのタイミング

Oracle と PostgreSQL では制約のチェックのタイミングが異なります。Oracle では DML 文の変更実行後に制約のチェックを行います。PostgreSQL では行単位にチェックが行われるため、行の物理的な格納順に依存して SQL が結果が異なる場合があります。

【制約チェックの例】

```
SELECT * FROM test_tbl;
pkey | col
-----+-----
  1 | a
  2 | aa
  3 | ABC
```

pkey はプライマリキー

・NG となるケース (Oracle ではエラーになりません)

```
UPDATE test_tbl SET pkey = pkey + 1;
ERROR: duplicate key value violates unique constraint "pkey"
DETAIL: Key (pkey)=(2) already exists
```

・OK となるケース

```
UPDATE test_tbl SET pkey = pkey - 1;
UPDATE 3
```

2.3.5. 除算を含む計算

PostgreSQL では乗除算は前方より順に行われます。このため割り切れない除算は数値型の精度に依存し丸め処理が行われた後に後方の計算が行われるため、Oracle と異なる結果になります。PostgreSQL 内では除算を含む計算をさせず、結果のみを格納するような変更が必要となります。

【計算結果の差 (例: $1 \div 3 \times 3$, $1/3 + 1/3 + 1/3$)】

Oracle	PostgreSQL
SELECT 1/3*3 結果 FROM dual 結果 ----- 1	SELECT 1/3*3; ?column? ----- 0 select 1.0/3.0*3.0; ?column? ----- 0.99999999999999999999
SELECT (1/3 + 1/3 + 1/3) 結果 FROM dual 結果 ----- 1	SELECT (1/3 + 1/3 + 1/3) ?column? ----- 0 select (1.0/3.0 + 1.0/3.0 + 1.0/3.0); ?column? ----- 0.99999999999999999999

2.4. トランザクション

Oracle と PostgreSQL におけるトランザクション処理に違いがあり、移行の際には注意が必要です。本節ではその違いについて概説します。

2.4.1. トランザクションの開始と自動コミット

Oracle ではトランザクションは SQL の実行によって暗黙的に開始されます。一方 PostgreSQL では標準 SQL 準拠の `START TRANSACTION` 文か、あるいは PostgreSQL 独自の `BEGIN` 文を実行して、明示的にトランザクションを開始する必要があります。PostgreSQL ではトランザクションを開始せずに発行されたコマンドは暗黙的にコミットされます。これは「自動コミット」と呼ばれています。

2.4.2. DDL の暗黙コミットとロールバック

Oracle では `CREATE TABLE` などの DDL 実行の前後で自動的に `COMMIT` が発行されます。DDL が実行された時点で、DDL 実行前に成功したコマンド全てと DDL の結果がコミットされます。そのため、DDL をロールバックすることはできません。一方、PostgreSQL では DDL が自動的にコミットを発行することではなく、トランザクション中で発行された DDL はロールバックすることが可能です。

2.4.3. トランザクションの終了とトランザクション中のエラー

`COMMIT` 文、あるいは `ROLLBACK` 文の実行でトランザクションが終了します。また、Oracle では前節で説明した理由により DDL の実行によってもトランザクションが終了します。なお PostgreSQL には標準 SQL 準拠の `COMMIT` 文、`ROLLBACK` 文の他、それぞれ同じ意味をもつ `END` 文、`ABORT` 文が存在します。

Oracle では `COMMIT` 文が実行されると、トランザクション内で成功したコマンドの結果のみをコミットし、失敗したコマンドは単に無視されます。一方 PostgreSQL では、トランザクション内でエラーが発生した場合はそのトランザクション全体が失敗とみなされます。エラーが発生したトランザクションではそれ以降のコマンド実行ができず、`COMMIT` 文を実行した場合には自動的に `ROLLBACK` が発行されます。

2.4.4. トランザクション分離レベル:リードコミテッドでの実行差異

Oracle も PostgreSQL もトランザクション分離レベルのデフォルトはリードコミテッドになります。トランザクションが正常に終了した場合であっても以下に示すように実行結果に差異がある場合もあります。

【トランザクションの実行差異の例】

時系列	Oracle		PostgreSQL	
	トラン1	トラン2	トラン1	トラン2
1	SQL> select * from test_trn; <pre> COL1 COL2 ----- 1 1 2 1 3 2 4 2 5 3 </pre> 5行が選択されました。	SQL> select * from test_trn; <pre> COL1 COL2 ----- 1 1 2 1 3 2 4 2 5 3 </pre> 5行が選択されました。	test01=# select * from test_trn; col1 col2 ----+---- 1 1 2 1 3 2 4 2 5 3 (5行)	test01=# select * from test_trn; col1 col2 ----+---- 1 1 2 1 3 2 4 2 5 3 (5行)
2	SQL> update test_trn set col2 = col2 + 1; 5行が更新されました。		test01=# begin; BEGIN test01=# update test_trn set col2 = col2 + 1; UPDATE 5	
3		SQL> delete from test_trn where col2 = 3;		test01=# begin; BEGIN test01=# delete from test_trn where col2 = 3;
4	SQL> commit; コミットが完了しました。		test01=# commit; COMMIT	
5		2行が削除されました。 SQL> commit; コミットが完了しました。		DELETE 0 test01=# commit; COMMIT
6	SQL> select * from test_trn; <pre> COL1 COL2 ----- 1 2 2 2 5 4 </pre> 3行が選択されました。	SQL> select * from test_trn; <pre> COL1 COL2 ----- 1 2 2 2 5 4 </pre> 3行が選択されました。	test01=# select * from test_trn; col1 col2 ----+---- 1 2 2 2 3 3 4 3 5 4 (5行)	test01=# select * from test_trn; col1 col2 ----+---- 1 2 2 2 3 3 4 3 5 4 (5行)

上記表では時系列:5の結果で Oracle と PostgreSQL に仕様差があり、同じテーブルに同じ SQL を実行しても結果が異なる現象を発生させます。

なお、時系列:3の PostgreSQL のトラン2で BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ を指定すると、時系列:5はエラー (ERROR: could not serialize access due to concurrent update) となります。

3. SQL Server から PostgreSQL への移行

本章では SQL Server から PostgreSQL へ移行の際に生じる SQL の書き換え方針を紹介します。

3.1. SELECT 文

3.1.1. TOP 句

SQL Server ではクエリ結果の件数を制限するために独自の拡張である TOP 句を用います。PostgreSQL の SELECT 文には TOP 句はありません。Window 関数、LIMIT 句、FETCH 句のいずれかを用いて書き換える必要があります。詳しくは本資料の 2.1.3 節を参照して下さい。以下は標準 SQL 準拠である FETCH 句を用いた書き換え例です。

【tbl テーブルのデータを id で昇順ソートし、最初から 10 行のレコードを取得する】

SQL Server	PostgreSQL
SELECT TOP 10 * FROM tbl ORDER BY id	SELECT * FROM tbl ORDER BY id FETCH FIRST 10 ROWS ONLY

3.1.2. 列の別名

SQL Server は独自の構文により = を使った列の別名を指定することができます。PostgreSQL はこの構文に対応していません。標準 SQL の AS (省略可)を用いて以下のように書き換える必要があります。

【列に別名を付ける(SELECT 文)】

SQL Server	PostgreSQL
SELECT STAFF = s.name, MANAGER = m.name FROM staff AS s LEFT OUTER JOIN staff AS m ON s.manager_id = m.id	SELECT s.name AS STAFF, m.name AS MANAGER FROM staff AS s LEFT OUTER JOIN staff AS m ON s.manager_id = m.id

3.2. 更新系

3.2.1. TOP 句

SQL Server の TOP 句は更新系のクエリにも用いることができます。SELECT の結果を挿入する INSERT 文で TOP 句を使用すると、SELECT された結果の上位から指定された行数だけ取得して挿入します。PostgreSQL で同等の動作は SELECT 部分を LIMIT 句または FETCH 句を用いて書き換えることで実現可能です。

【tbl テーブルから偶数の id を昇順に 10 個取得して nums テーブルに挿入】

SQL Server	PostgreSQL
INSERT TOP (10) INTO nums SELECT * FROM tbl WHERE id % 2 = 0 ORDER BY id	INSERT INTO nums SELECT * FROM tbl WHERE id % 2 = 0 ORDER BY id LIMIT 10

UPDATE 文で TOP 句を用いると、指定された行数のレコードがランダムに選ばれ更新されます。PostgreSQL では random 関数と UPDATE 文の FROM 句を用いて同様の機能を実現できます。なお、UPDATE 文の FROM 句は PostgreSQL 独自の拡張機能です。

【nums テーブルのレコードをランダムに 10 つ選んで更新】

SQL Server	PostgreSQL
UPDATE TOP (10) nums SET value = value * 10	UPDATE nums AS t1 SET value = value * 10 FROM (SELECT value FROM nums ORDER BY random() LIMIT 10) AS t2 WHERE t1.value = t2.value

DELETE 文で TOP 句を用いると、指定された行数のレコードがランダムに選ばれ削除されます。PostgreSQL では random 関数と DELETE 文の USING 句を用いて同様の機能を実現できます。なお、DELETE 文の USING 句は PostgreSQL 独自の拡張機能です。

【nums テーブルのレコードをランダムに 10 つ選んで削除】

SQL Server	PostgreSQL
DELETE TOP (3) FROM nums	DELETE FROM nums AS t1 USING (SELECT value FROM nums ORDER BY random() LIMIT 10) AS t2 WHERE t1.value = t2.value

3.2.2. INSERT 文の INTO

SQL Server の INSERT 文では INTO キーワードが省略可能ですが、PostgreSQL では省略することはできません。もし INTO が省略されている場合には書き足す必要があります。

【INSERT 文の INTO は省略できない】

SQL Server	PostgreSQL
INSERT tbl VALUES (10, 'ten')	INSERT INTO tbl VALUES (10, 'ten')

3.2.3. DELETE 文の FROM

SQL Server の DELETE 文では FROM キーワードが省略可能ですが、PostgreSQL では省略することはできません。もし FROM が省略されている場合には書き足す必要があります。

【DELETE 文の FROM は省略できない】

SQL Server	PostgreSQL
DELETE tbl WHERE id = 10	DELETE FROM tbl WHERE id = 10

3.2.4. OUTPUT 句

SQL Server では更新された行の結果を返すのに OUTPUT 句を使用することができます。PostgreSQL には OUTPUT 句は存在しません。代わりに PostgreSQL 独自の拡張である RETURNING 句を用います。

なお SQL Server では UPDATE 文で OUTPUT 句を用いて「更新される前の値」を返すこともできますが、PostgreSQL にはそのような機能はありません。更新前の値が必要な場合には事前に回避しておく必要があります。

【INSERT された行を返す】

SQL Server	PostgreSQL
INSERT INTO tbl OUTPUT INSERTED.* VALUES (11, 'eleven')	INSERT INTO tbl VALUES (11, 'eleven') RETURNING *

【UPDATE された行の結果を返す】

SQL Server	PostgreSQL
UPDATE tbl SET value = 'ELEVEN' OUTPUT INSERTED.* WHERE id = 11	UPDATE tbl SET value='ELEVEN' WHERE id = 11 RETURNING *

【DELETE された行を返す】

SQL Server	PostgreSQL
DELETE FROM tbl OUTPUT DELETED.* WHERE id = 11	DELETE FROM tbl WHERE id = 11 RETURNING *

3.2.5. MERGE

MERGE 文はテーブルに既存の行がある場合には更新を、ない場合には新規に挿入を行う SQL 文です。標準 SQL に従ったものですが PostgreSQL はこれに対応していません。PostgreSQL では WITH 句の中で UPDATE 文を用いることにより、これと同等の機能を実現することができます。なお、更新を含む WITH 句は PostgreSQL 独自の拡張です。

【diff テーブルの値を master テーブルにマージする。

(master テーブルに ID が一致する行があったら、diff.val を master.val に足し加える。

ID が一致する行がない場合には、diff の内容を master に新規登録する。)]

SQL Server	PostgreSQL
<pre> MERGE INTO master USING diff ON master.id = diff.id WHEN MATCHED THEN UPDATE SET master.val = master.val + diff.val WHEN NOT MATCHED THEN INSERT VALUES (diff.id, diff.val) </pre>	<pre> WITH inpt AS (SELECT * FROM diff), updt AS (UPDATE master.val = master.val + inpt.val FROM inpt WHERE master.id = inpt.id RETURNING master.id) INSERT INTO master (SELECT * FROM inpt WHERE id NOT IN (SELECT id FROM updt)) </pre>

3.2.6. ビューに対する更新

SQL Server ではビューに対する更新が可能ですが PostgreSQL ではビューに対して更新することはできません。ただし、RULE もしくはトリガーと組み合わせることで、更新可能なビューと同等な機能を実現することが可能です。その方法は「スキーマ移行調査編」の第5章で述べられていますので、そちらを参照してください。

3.3. その他の書き換え

3.3.1. 文字列リテラルの区切り文字

SQL Server では SET QUOTED_IDENTIFIER が OFF の場合には、文字列リテラルを表すのに二重引用符("...")を使用可能です。しかし、PostgreSQL では二重引用符をこの用途では使用することはできません。代わりに引用符('...')で書き換える必要があります。

【文字列リテラルの区切り文字に二重引用符は使えない】

SQL Server	PostgreSQL
SELECT * FROM tbl WHERE value= "nine"	SELECT * FROM tbl WHERE value= 'nine'

3.3.2. 識別子の区切り文字

SQL Server では識別名を表すのに角括弧([...])を使用可能ですが、PostgreSQL では角括弧をこの用途で使うことはできません。代わりに二重引用符("...")で書き換える必要があります。

【識別子の区切り文字に引用符は使えない】

SQL Server	PostgreSQL
SELECT * FROM [long name table]	SELECT * FROM "long name table"

3.3.3. 文字列連結演算子

SQL Server では文字列の連結に + 演算子を用います。PostgreSQL では + 演算子はこの用途に使用することができません。|| 演算子で置き換えてください。

【文字列の連結】

SQL Server	PostgreSQL
SELECT 'Elephants' + ' never ' + 'forget.'	SELECT 'Elephants' ' never ' 'forget.'

3.3.4. LIKE 演算子

SQL Server では LIKE 演算子の中で文字クラスを使うことができますが、PostgreSQL の LIKE 演算子は文字クラスに対応していません。かわりに SQL 正規表現を扱える SIMILAR TO 演算子で置き換えます。

【A~G 以外の文字から始まり A~G で終わる名前を検索】

SQL Server	PostgreSQL
SELECT name FROM staff WHERE name LIKE '[^a-g][a-g]'	SELECT name FROM staff WHERE lower(name) SIMILAR TO '[^a-g][a-g]'

3.3.5. 大文字/小文字、全角/半角、平仮名/片仮名の区別

SQL Server では照合順序の設定によっては、文字列比較の際に大文字/小文字、全角/半角、平仮名/片仮名が区別されません。PostgreSQL ではこれらは区別されますので移行の際には注意が必要です。アプリケーション側を変更することで対応する他に、SQL の書き換えでもある程度の対応は可能です。

大文字/小文字を区別しない文字列比較のためには PostgreSQL 独自の ILIKE 演算子や ~* 演算子が用意され

ています。(～* 演算子については 2.3.2 節を参照してください)。その他に、upper 関数または lower 関数を使って、文字列を予め大文字か小文字のどちらかに変換してから比較を行う方法があります。lower 関数を使った例は前節の LIKE 演算子の書き換え例を参照してください。

全角/半角の変換は以下のように translate 関数を用いることで可能です。同様の処理を行うユーザ関数を作成しておくことで lower 関数と同じ要領で使うことができます。同様の方法で他の記号の全角/半角変換や平仮名/片仮名の変換も可能です。

【全角文字を含むデータからの検索】

SQL Server	PostgreSQL
SELECT name FROM products WHERE code LIKE '%PGQL-5432-xxx%'	SELECT * FROM products WHERE translate(upper(code), '-0123456789ABCDEFGHIJKLMN O P Q R S T U V W X Y Z', '-0123456789ABCDEFGHIJKLMN O P Q R S T U V W X Y Z') LIKE '%PGQL-5432-xxx%'

3.3.6. 比較演算子 !<、!>

SQL Server では「小さくない」、「大きくない」を表す演算子 !<、!> が使用できますが、この演算子は PostgreSQL には存在しません。「以上」、「以下」を表す演算子 >=、<= で置き換えてください。

3.3.7. 排他的論理和演算子

SQL Server でビット演算の「排他的論理和(XOR)」を表す演算子は ^ ですが、PostgreSQL ではこの演算子は数の累乗を表します。PostgreSQL の排他的論理和演算子 # で置き換えてください。

3.4. トランザクション

本節では SQL Server と PostgreSQL のトランザクションに関する SQL の差異について概説します。

3.4.1. BEGIN TRANSACTION

SQL Server ではトランザクションの開始に BEGIN TRANSACTION 文を用います。これは標準 SQL に準拠したものではありませんが PostgreSQL でも同じ名前の文が存在します。

SQL Server ではこの文を略して”BEGIN TRAN”と書くことが可能ですが、この構文は PostgreSQL に存在しません。略さない形式か、単に”BEGIN”と書き直す必要があります。

【トランザクションの開始】

SQL Server	PostgreSQL
BEGIN TRAN	BEGIN

また、SQL Server では BEGIN TRANSACTION 文のオプションとしてトランザクションに名前を付けることができますが、PostgreSQL の BEGIN 文にはそのような機能はありません。

3.4.2. COMMIT TRANSACTION

トランザクションのコミットは COMMIT TRANSACTION 文で行います。COMMIT 文は標準 SQL に準拠していますが、標準 SQL が規定しているのは”COMMIT”と”COMMIT WORK”の2種類の構文のみです。しかしながら、PostgreSQL は”COMMIT TRANSACTION”という構文にも対応しています。

SQL Server ではこの文を略して”COMMIT TRAN”と書くことが可能ですが、この構文は PostgreSQL に対応していません。略さない形式か、単に”COMMIT”と書き直してください。

【トランザクションのコミット】

SQL Server	PostgreSQL
COMMIT TRAN	COMMIT

また、SQL Serve の COMMIT TRANSACTION ではトランザクションの名前を指定することができますが、PostgreSQL の COMMIT 文にはそのような機能はありません。

3.4.3. SAVE TRANSACTION

SQL Server ではセーブポイントの保存に SAVE TRANSACTION 文を用います。これは標準 SQL には存在しない文です。PostgreSQL では標準 SQL に準拠した SAVEPOINT 文を用います。



【セーブポイントの設定】

SQL Server	PostgreSQL
SAVE TRANSACTION savepoint_name	SAVEPOINT savepoint_name

3.4.4. ROLLBACK TRANSACTION

トランザクションのロールバックは ROLLBACK TRANSACTION 文で行います。ROLLBACK 文は標準 SQL に準拠していますが、標準 SQL が規定しているのは”ROLLBACK”と”ROLLBACK WORK”の2種類の構文のみです。しかしながら、PostgreSQL は”ROLLBACK TRANSACTION”という構文にも対応しています。

SQL Server ではこの文を略して”ROLLBACK TRAN”と書くことが可能ですが、この構文は PostgreSQL に対応していません。略さない形式か、単に”ROLLBACK”と書き直してください

【トランザクションのロールバック】

SQL Server	PostgreSQL
ROLLBACK TRAN	ROLLBACK

また、SQL Server では ROLLBACK TRANSACTION 文にトランザクションの名前を指定することができますが、PostgreSQL の ROLLBACK 文にはそのような機能はありません。

SQL Server はセーブポイントへの復帰にも ROLLBACK TRANSACTION 文を用います。この場合にはトランザクション名のかわりにセーブポイント名を指定します。PostgreSQL では標準 SQL 準拠の ROLLBACK TO SAVEPOINT 文を用います。

【セーブポイントへの復帰】

SQL Server	PostgreSQL
ROLLBACK TRAN savepoint_name	ROLLBACK TO SAVEPOINT savepoint_name

4. 別紙一覧

- 別紙:SQL 差異表

著者

版	所属企業・団体名	部署名	氏名
SQL 移行調査編 第 1.0 版 (2012 年度 WG2)	SRA OSS, Inc. 日本支社	技術開発部	長田 悠吾
SQL 移行調査編 第 1.2 版 (2018 年度 WG2)	富士通エフ・アイ・ピー株式会社	システム技術推進部	多田 明弘