

# 日々是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2020年4月14日 火曜日

## APEX\_STRING.SPLIT関数呼び出し時の性能について

Twitterをみていたら、Martin D'Souzaによる"[Optimizing MEMBER OF with APEX\\_STRING.SPLIT](#)"という記事が目にとまりました。APEX\_STRING.SPLITの性能の話は興味があるので内容を確認してみました。以下、その確認できたことを紹介します。

まず、APEX\_STRING.SPLITという関数ですが、日本語マニュアルの説明は[こちら](#)になります。例えば、こういった文字列"A:B:C"を、apex\_t\_varchar2という配列型の要素 A,B,C に分割します。色々な用途で使えます。APEXではページ・アイテムがチェックボックスの場合、選択した値は':'で区切られた複数の値として設定されます。例えば、選択肢がA, B, Cとあって、AとBがチェックされるとページ・アイテムとしての値は'A:B'になります。それを分割するために、APEX\_STRING.SPLITを使うことができます。

もう少し具体的な例を挙げてみましょう。従業員のサンプル・データを使った対話グリッドに、選んだ職種のみ表示/編集の対象とするチェックボックスを付けました。



The screenshot shows an Oracle APEX interactive grid with columns: EmpID, EmpName, Job, Age, Hiredate, Sal, Comm, and Deptno. There are checkboxes in the first column for each row. The first row is highlighted in blue. The grid is titled 'EMPLOYEE' and has a search bar at the top.

EmpID	EmpName	Job	Age	Hiredate	Sal	Comm	Deptno
7800	クラーク	アナリスト	29	21-12-1982	5000.0000		20
7801	キング	アナリスト	35	21-05-1981	5000.0000		20
7802	クラーク	社長	30	21-05-1981	5000.0000		10
7803	クラーク	セールス	29	21-05-1981	5000.0000	200	30
7804	クラーク	セールス	29	21-05-1981	5000.0000	200	30
7805	クラーク	セールス	29	21-05-1981	5000.0000	200	30
7806	クラーク	セールス	29	21-05-1981	5000.0000	200	30
7807	クラーク	セールス	29	21-05-1981	5000.0000	200	30
7808	クラーク	セールス	29	21-05-1981	5000.0000	200	30
7809	クラーク	セールス	29	21-05-1981	5000.0000	200	30

チェックボックスとなっているページ・アイテムの名前は、P8\_JOB\_SELECTIONSとしています。LOVの定義として、以下のSQL問い合わせを設定しています。

```
select job as disp_value, job as ret_value from emp group by job
```

EMP表に存在するすべての職種が選択対象となります。スクリーンショット上で選択しているのは、アナリスト、社長、セールスなので、P8\_JOB\_SELECTIONSの値は"アナリスト:社長:セールス"になります。

このチェックボックスの選択結果を対話グリッドに反映させるために、対話グリッドのソースを以下のように変更します。

WHERE句に以下を追加しています。

```
job in (select column_value from apex_string.split(:P8_JOB_SELECTIONS, ':'))
```

そして、送信するページ・アイテムとしてP8\_JOB\_SELECTIONSを指定します。後は、ページ・アイテムP8\_JOB\_SELECTIONSの値が変更されたときに対話グリッドをリフレッシュする動的アクションを設定すると、チェックボックスの選択に対話グリッドの検索/編集対象を制限する機能の実装が完了です。

APEX\_STRING.SPLITはPL/SQLで実装されているため、SQLから呼び出されるときはSQLからPL/SQLへのコンテキスト・スイッチ（結構重い）が発生します。WHERE句が評価される度にAPEX\_STRING.SPLITが呼び出されるようだと、すごく遅くなります。では、それを避けるためには何を注意する必要があるのでしょうか。

先ほどの対話グリッドで実際に実行されているSQLは以下になります。

```
select * from emp
where job in (select column_value from apex_string.split('アナリスト:社長:セールス',':'));
```

APEX\_STRING.SPLITが呼び出されたときにメッセージを表させるために、APEX\_STRING.SPLITのラッパーとなるファンクションを定義します。

```
create or replace function wrapper_split(
  p_str in varchar2,
  p_sep in varchar2)
  return apex_t_varchar2
as
begin
  dbms_output.put_line('APEX_STRING.SPLIT CALLED at ' || systimestamp);
  return apex_string.split(
    p_str => p_str,
    p_sep => p_sep
  );
end wrapper_split;
```

定義したラッパー・ファンクションの動作を確認します。DBMS\_OUTPUTの出力結果とSELECT文の実行結果が混らないように、双方ともDBMS\_OUTPUTの出力結果として表示させます。

```

begin
  for r in
  (
    select * from wrapper_split('アナリスト:社長:セールス',':')
  )
  loop
    dbms_output.put_line(r.column_value);
  end loop;
end;

```

実行結果は以下になります。

APEX\_STRING.SPLIT CALLED at 20-04-14 06:21:31.423595000 +00:00

アナリスト  
社長  
セールス

SELECT文 1 回の実行で、APEX\_STRING.SPLITが一度だけ呼ばれていることがわかります。

次に対話グリッドで使われているSQLの結果を確認します。EMP表は全部で 1 4 行です。

```

begin
  for r in
  (
    select * from emp
    where job in (select column_value from wrapper_split('アナリスト:社長:セールス',':'))
  )
  loop
    dbms_output.put_line(r.job);
  end loop;
end;

```

実行結果は以下になります。選択された行は 7 行です。APEX\_STRING.SPLITの呼び出しは一度だけで、 1 4 回ではありません。

APEX\_STRING.SPLIT CALLED at 20-04-14 06:26:32.582635000 +00:00

アナリスト  
アナリスト  
社長  
セールス  
セールス  
セールス  
セールス

例えばこれが、以下のようなPL/SQLで記述されたファンクションの場合を考えてみます。UPPER()へのラッパー・ファンクションです。

```

create function my_upper(
  p_value varchar2
)
return varchar2
is
begin
  dbms_output.put_line('UPPER CALLED AT ' || systimestamp);
  return upper(p_value);
end;

```

このMY\_UPPER()を検索条件に与えたSQLを実行します。

```

begin
  for r in
  (

```

```

select * from emp
where my_upper(job) = 'セールス'
)
loop
  dbms_output.put_line(r.job);
end loop;
end;

```

実行結果は以下になります。

```

UPPER CALLED AT 20-04-14 06:35:06.321017000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321131000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321142000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321150000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321157000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321164000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321171000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321178000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321187000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321194000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321201000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321208000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321214000 +00:00
UPPER CALLED AT 20-04-14 06:35:06.321221000 +00:00

```

```

セールス
セールス
セールス
セールス

```

MY\_UPPER()は14回呼び出されています。検索結果の行数（この結果では4行）ではなく、EMP表の行数だけ呼び出されます。その度にコンテキスト・スイッチが発生するため、パフォーマンス面での大きなデメリットになります。

さて、対話グリッドのSELECT文の検索条件にAPEX\_STRING.SPLITを使った際に、呼び出し回数は1回でした。理由を理解するために、このSQLの実行計画を確認します。

EMP表とIN句に与えているSELECT文の結果で(HASH)JOINしているのがわかります。つまり、実際には、元ブログにあるような以下のSQLとして実行されています。

```

select e.* from emp e
join (select column_value from apex_string.split('アナリスト:社長:セールス',';')) j
on e.job = j.column_value;

```

結果として、APEX\_STRING.SPLITの呼び出しが1回になっています。

さて、member ofを使うと、もっとSQLがきれいに書ける、という意見がありました。次のようなSQLになります。

```

select * from emp
where job member of apex_string.split('アナリスト:社長:セールス',';');

```

確かにSQLは読みやすくなります。では、呼び出し回数を確認してみましょう。

```

begin
  for r in
  (
    select * from emp

```

```

where job member of wrapper_split('アナリスト:社長:セールス',':')
)
loop
  dbms_output.put_line(r.job);
end loop;
end;
```

実行結果は以下になります。

```

APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.414940000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415131000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415164000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415197000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415229000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415245000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415260000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415275000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415291000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415306000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415321000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415336000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415351000 +00:00
APEX_STRING.SPLIT CALLED at 20-04-14 06:54:17.415366000 +00:00
```

社長  
 アナリスト  
 アナリスト  
 セールス  
 セールス  
 セールス  
 セールス

EMP表の行数だけAPEX\_STRING.SPLITが呼ばれています。これは嬉しくありません。この検索条件をスカラ・サブクエリ・キャッシングを働かせるSQLに書き換えることにより、呼び出し回数を1回にすることができます。SQLは次のようになります。

```

select * from emp
where job member of (select apex_string.split('アナリスト:社長:セールス',':') from dual)
```

確認結果は省略しますが、APEX\_STRING.SPLITの呼び出し回数は1回になっています。実行計画は以下です。最初のSQLとは異なる実行計画で、さて、どちらが速いでしょうか。前者は社長、アナリスト、セールスを別々の値として扱った上で、それぞれのハッシュ値をとってジョインすることで検索対象を絞り込んでいます。後者は、社長、アナリスト、セールスをひとつの値としてキャッシュし、EMP表の行すべてで、JOBがこの3つの値に含まれるかどうか判断しています。確認は省きますが、ハッシュ値で絞り込む前者の方が高速だろうと思われます。

ステップ	オペレーション	コスト	実行時間	メモリ	実行回数	SQL文
0	SELECT STATEMENT	1	0	0	1	select * from emp where job member of (select apex_string.split('アナリスト:社長:セールス',':') from dual)
1	EMP TABLE ACCESS	1	0	0	1	EMP
2	TABLE ACCESS	1	0	0	1	DUAL

スカラ・サブクエリとは、1つの行から1つの列値のみを戻す副問合せです。オラクルのマニュアルでは、[スカラ副問い合わせ式](#)として説明がされています。スカラ・サブクエリ・キャッシングはスカラ・サブクエリの結果、つまり1つの値ですが、これを与えられているバインド変数の値をキーとして(メモリ上に作成されたSQLが実行されている間だけ有効な)ハッシュ・テーブルに登録します。バインド変数として与えられる値をキーにして、メモリ上のハッシュ・テーブルから結果を取得することで、検索処理の実行を省きます。今回の例はSELECT文にバインド変数を含まないため、結果は1つだけであり、必ずキャッシュにヒットします。そのため、APEX\_STRING.SPLITの呼び出しは1度だけです。かなり以前からある機能ですが、この仕組みを説明した文書は少なく、[これくらい](#)でしょうか。とはいえ、仕組みは難しいものではありません。

APEXが提供しているPL/SQL APIを効果的に使用するために、知識の引き出しにいておきたい機能のひとつです。

## 追記

その1：元ブログのラッパー・ファンクションは内部で、[Logger](#)を使用しています。元ブログの記事を書いたMartin D'SouzaがLoggerの作者です。

その2：最近知ったのですが、私はAPEX\_STRING.SPLITを使用する際に以下のようにtable()で囲っていたのですが、18cからはtable()は省略できるようになっています。

```
job in (select column_value from table(apex_string.split(:P8_JOB_SELECTIONS, ':')))
```

完

Yuji N. 時刻: 18:02

共有

---

◀

ホーム

▶

[ウェブ バージョンを表示](#)

### 自己紹介

[Yuji N.](#)

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。  
こちらの記事につきましては、免責事項の参照をお願いいたします。

[詳細プロフィールを表示](#)

Powered by Blogger.

---