**しばちょう先生の試して納得！DBAへの道**

しばちょう先生の試して納得！DBAへの道 indexページ▶▶

# しばちょう先生の試して納得！DBAへの道 第53回 SQLパフォーマンスの高速化の限界を目指せ！(1)



みなさん、こんにちは。"しばちょう"こと柴田長（しばた つかさ）です。

今回から複数回にわたり、先日開催された[Oracle Code Tokyo 2017]においてLive Demoでチャレンジさせて頂いた、「SQLパフォーマンスの高速化の限界を目指せ！」の内容をお届けしたいと思います。

初回の今回は、デモで用いたスキーマを作成するところから、問題となっているSELECT文を実行してリアルタイムSQL監視でボトルネックを特定し、パーティション化によって高速化させる部分まで解説できればと考えています。これまでの連載でも取り上げてきた機能ではありますが、実戦でどのように活用するのかの参考になれば幸いです。

以下の演習をOracle Database 12c Release 12.2.0.1 Enterprise Editionのデータベースで試してみてください。Oracle Database 12c Release 12.1.0.2環境でも動作させることは可能ですが、是非、Oracle Database Cloud Service（以降、DBCS）のトライアルでのご活用もお勧めします。その際には、Database In-Memory機能を使用可能なExtreme Performance Packageをご利用ください。DBCS上にデータベースを作成する手順は、「第50回 [Oracle Database 12c Release 2] Oracle Database Cloud Service上にデータベースを作成」を参考にしてみてくださいね。

また、ご紹介する機能の効果を体感頂くためには、使用するH/Wスペックが大切な要素となってきます。推奨環境としては、CPUコアが8つ以上、物理メモリが16GB以上（SGA+PGAで8GBは欲しい）、データファイルを配置するデバイスはHDDです。これらの理由は演習を通して解説させていただきますね。もちろん動作確認だけが目的であれば、この通りではありません。

ちなみに、Oracle Code Tokyo 2017の公開資料は、「こちら」からダウンロードすることが可能ですので、そちらも参考にしてみてくださいね。

**1. GitHub web site: https://github.com/oracle/db-sample-schemas/releases/latest からOracle Database 12.2.0.1 Sample Schemasを作成するスクリプトをダウンロードしてください。**

Oracle Databaseをインストールした直後のバイナリ上（$ORACLE_HOME/demo/schema）にはHuman Resourcesスキーマを作成するスクリプトが配置されていますが、今回使用したいのはSales Historyスキーマであるため、GitHubからダウンロードする必要があります。そのような情報は何処に記載されているのか？と言うと、マニュアル「Database Sample Schemas」の「Installing Sample Schemas from GitHub」の章です。

ダウンロードして、以下のようにデータベース・サーバー上で圧縮ファイルを展開して実行すれば・・・と思いがちですが、もう少しオマジナイが必要です。

```
$ unzip db-sample-schemas-12.2.0.1.zip
Archive:  db-sample-schemas-12.2.0.1.zip
99049719d53c2e0810b7a8462965636b98161131
   creating: db-sample-schemas-12.2.0.1/
  inflating: db-sample-schemas-12.2.0.1/CONTRIBUTING.md
  ...

$ cd db-sample-schemas-12.2.0.1; ls
bus_intelligence  human_resources  mk_dir.sql    mkunplug.sql  product_media  sales_history
CONTRIBUTING.md   info_exchange    mkplug.sql    mkverify.sql  README.md      shipping
drop_sch.sql      LICENSE.md       mksample.sql  order_entry   README.txt

$ perl -p -i.bak -e 's#__SUB__CWD__#'$(pwd)'#g' *.sql */*.sql */*.dat
```

展開された各スクリプト内には、「__SUB__CWD__」という文字列が埋め込まれているので、これを展開したディレクトリ名に置換してあげなければ実行できない仕組みになっています。これはREADME.mdファイルの中身を読むことで、その必要性を理解できるのですが、ちょっと若干不親切ですね。。。

**2. サンプルスキーマ用の表領域の作成をしてください。**

```
$ sqlplus / as sysdba
SQL> show PDBs

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         4 POCO                           READ WRITE NO

alter session set container=POCO ;
create bigfile tablespace TBS1 datafile '+DATA(DATAFILE)' size 40g ;
```

上記の実行例では、Pluggable DatabaseであるPOCOに接続して40GBの表領域TBS1を作成しています。かなり容量が大きめですが、この後の性能改善効果をご確認したいのであれば、是非用意して頂きたいです。その際は、外付けのUSB HDD上にOracle Virtual Boxの仮想ディスクを作成して頂くことも一つの案ですね。

**3. Oracle Sample SchemaのSales Historyスキーマをインストールしてください。**

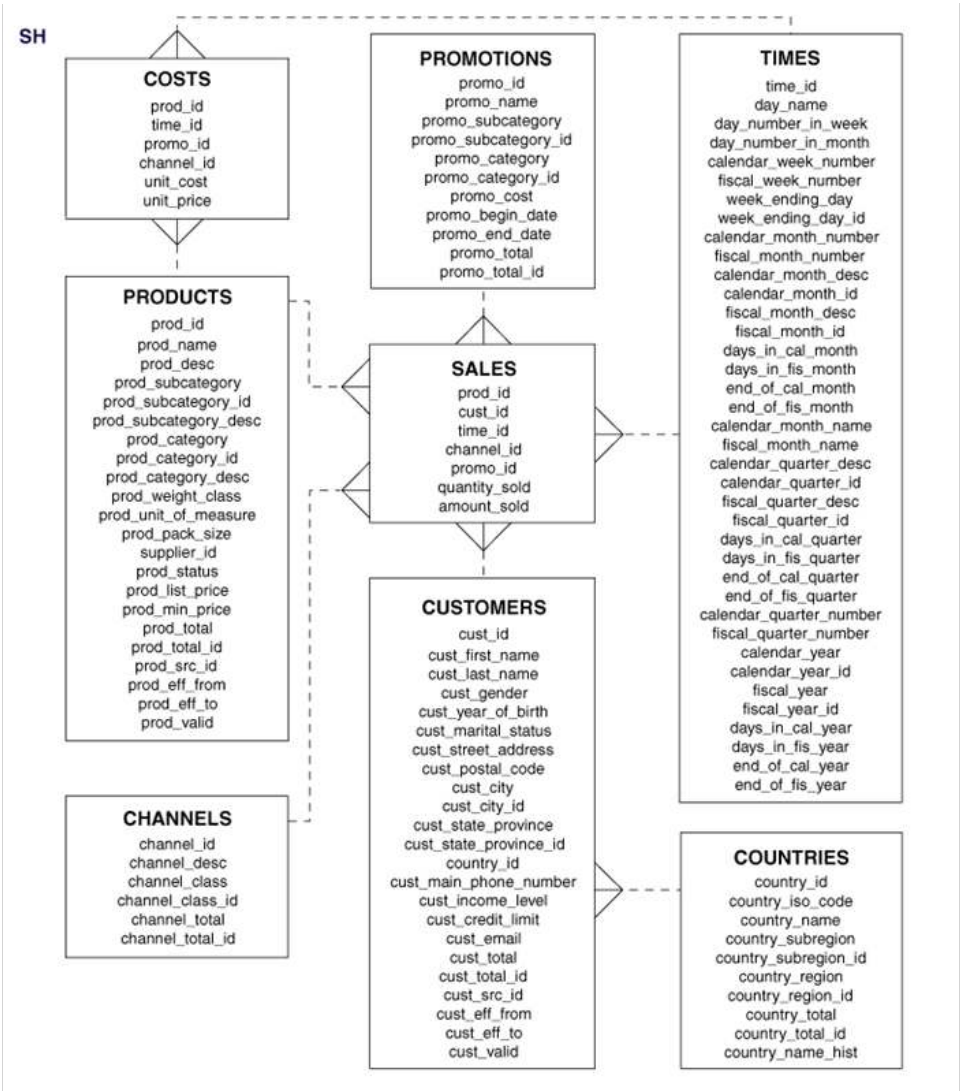
```
$ sqlplus /nolog
SQL> @mksample.sql   oracle12345 oracle12345 oracle12345 oracle12345
oracle12345 oracle12345 TBS1 TEMP /tmp/ localhost:1521/poco
```

README.txtファイルの中に記載があるのですが、今回使用したいSHスキーマはOEスキーマが存在している必要があります。そのOEスキーマはHRスキーマが必要である等、依存関係が少し激しいです。そのため、サンプルスキーマを一旦すべて作成してしまった方が早いと思うので、上記の実行例ではmksample.sqlを実行しています。このmksample.sqlには引数を指定することになるのですが、これもまた、以下の通りREADME.txt内に記載がありますので、これに従って実行してみてくださいね。

```
2.4     Invoke the Sample Schema creation script. Using the values listed
        in Sections 2.1, 2.2, and 2.3 the command would look like this:

        SQL> @?/demo/schema/mksample

                  EXAMPLE TEMP
                $ORACLE_HOME/demo/schema/log/ localhost:1521/pdb

        The mksample script lists all the objects created in the data dictionary.
```

ちなみに、最後の引数はスキーマを作成するデータベースのサービスへ接続する接続子であり、tnsnames.oraに定義しているのであればそちらを指定することも可能です。今回は、Localhost上でポート番号1521でリッスンしているListenerプロセスに対して、サービス名pocoへの接続を要求していますので、皆さんの環境に合わせて書き直してみてください。

実際に実行してみると、（色々とエラーが出力されているように見えますが）何とか目的のSHスキーマが完成します。




**4. 次のSQL文を実行して、今回のパフォーマンス・チューニングの効果が見え易いようにSHスキーマのSALES表のカスタマイズを実施します。**

```
$ sqlplus sh/oracle12345@localhost:1521/poco
SQL>
-- 元のSALES表を別名に変更
rename SALES to SALES_ORG;

-- ダミー列を2つ追加した新たなSALES表（非パーティション表）を作成
CREATE TABLE "SH"."SALES"
   (    "PROD_ID" NUMBER NOT NULL ENABLE,
        "CUST_ID" NUMBER NOT NULL ENABLE,
```

```
        "TIME_ID" DATE NOT NULL ENABLE,
        "CHANNEL_ID" NUMBER NOT NULL ENABLE,
        "PROMO_ID" NUMBER NOT NULL ENABLE,
        "QUANTITY_SOLD" NUMBER(10,2) NOT NULL ENABLE,
        "AMOUNT_SOLD" NUMBER(10,2) NOT NULL ENABLE,
        "DUMMY1" CHAR(100),
        "DUMMY2" CHAR(110)
   );

-- 元のSALES表（SALES_ORG）から新規作成したSALES表へレコードをコピー
insert /*+append */ into SH.SALES
  nologging
    select PROD_ID,
           CUST_ID,
           TIME_ID+15*365+3, -- 年月日のデータを15年間進める
           CHANNEL_ID,
           PROMO_ID,
           QUANTITY_SOLD,
           AMOUNT_SOLD,
           rpad(to_char(mod(CUST_ID,30)),100,'dummy1'),
           rpad(to_char(mod(CUST_ID,30)),110,'dummy2')
      from SH.SALES_ORG;
commit;

-- データ容量（レコード数）を256MBから約32GBまで増加
alter session force parallel dml parallel 8 ;
alter session force parallel query parallel 8 ;
insert /*+append */ into SH.SALES nologging select * from SH.SALES;
commit;
insert /*+append */ into SH.SALES nologging select * from SH.SALES;
commit;
insert /*+append */ into SH.SALES nologging select * from SH.SALES;
commit;
insert /*+append */ into SH.SALES nologging select * from SH.SALES;
commit;
insert /*+append */ into SH.SALES nologging select * from SH.SALES;
commit;
insert /*+append */ into SH.SALES nologging select * from SH.SALES;
commit;
insert /*+append */ into SH.SALES nologging select * from SH.SALES;
commit;

col segment_name for a32
select SEGMENT_NAME, BYTES/1024/1024 from USER_SEGMENTS where SEGMENT_NAME='SALES' ;

SEGMENT_NAME                     BYTES/1024/1024
-------------------------------- ---------------
SALES                                  31739.125
```

はい、コチラは特に解説は不要かと思います。元のSALES表はパーティション表なのですが、この後の演習でパーティション化による効果（パーティション・プルーニング）を確認したいために非パーティション表として新たなSALES表を作成し直しています。また、元のレコード数（データ量）では小さ過ぎてチューニングの効果が見えづらいために、CHAR型の列データをダミーで追加した上でレコード数を増幅しています。

ちなみに、一つの検証する上でのテクニックなのですが、ダミー列に対してRPAD関数等を用いて変換したデータ「rpad(to_char(mod(CUST_ID,30)),100,'dummy1')」を挿入していますが、圧縮率を調整するために「mod関数（第一引数を第二引数で割り算した際の余りを出力する関数）」を活用しています。この例だと「30」でCUST_ID列の値を割り算しているので、あまりの値のパターンとしては0~29の30パターンです。つまり、このダミー列の値の種類が30種類という事になります。この種類を減らせば、より圧縮率が高くなりますし、逆に増やせば圧縮率が低くなります。是非、お試しくださいね。あ、圧縮の効果については次回の連載で扱う予定ですよ～

**5. 次のSQL文を実行して、演習4でカスタマイズしたSALES表の日付データに対応するように、TIMES表のカスタマイズを実施します。**

```
$ sqlplus sh/oracle12345@localhost:1521/poco
SQL>
rename TIMES to TIMES_ORG;
CREATE TABLE "SH"."TIMES"
( "TIME_ID" DATE NOT NULL ,
  "DAY_NAME" VARCHAR2(9) NOT NULL ,
  "DAY_NUMBER_IN_WEEK" NUMBER(1) NOT NULL ,
  "DAY_NUMBER_IN_MONTH" NUMBER(2) NOT NULL ,
  "CALENDAR_WEEK_NUMBER" NUMBER(2) NOT NULL ,
  "FISCAL_WEEK_NUMBER" NUMBER(2) NOT NULL ,
  "WEEK_ENDING_DAY" DATE NOT NULL ,
  "WEEK_ENDING_DAY_ID" NUMBER NOT NULL ,
  "CALENDAR_MONTH_NUMBER" NUMBER(2) NOT NULL ,
  "FISCAL_MONTH_NUMBER" NUMBER(2) NOT NULL ,
  "CALENDAR_MONTH_DESC" VARCHAR2(8) NOT NULL ,
  "CALENDAR_MONTH_ID" NUMBER NOT NULL ,
  "FISCAL_MONTH_DESC" VARCHAR2(8) NOT NULL ,
  "FISCAL_MONTH_ID" NUMBER NOT NULL ,
  "DAYS_IN_CAL_MONTH" NUMBER NOT NULL ,
  "DAYS_IN_FIS_MONTH" NUMBER NOT NULL ,
  "END_OF_CAL_MONTH" DATE NOT NULL ,
  "END_OF_FIS_MONTH" DATE NOT NULL ,
  "CALENDAR_MONTH_NAME" VARCHAR2(9) NOT NULL ,
  "FISCAL_MONTH_NAME" VARCHAR2(9) NOT NULL ,
  "CALENDAR_QUARTER_DESC" CHAR(7) NOT NULL ,
  "CALENDAR_QUARTER_ID" NUMBER NOT NULL ,
  "FISCAL_QUARTER_DESC" CHAR(7) NOT NULL ,
```

```
  "FISCAL_QUARTER_ID" NUMBER NOT NULL ,
  "DAYS_IN_CAL_QUARTER" NUMBER NOT NULL ,
  "DAYS_IN_FIS_QUARTER" NUMBER NOT NULL ,
  "END_OF_CAL_QUARTER" DATE NOT NULL ,
  "END_OF_FIS_QUARTER" DATE NOT NULL ,
  "CALENDAR_QUARTER_NUMBER" NUMBER(1) NOT NULL ,
  "FISCAL_QUARTER_NUMBER" NUMBER(1) NOT NULL ,
  "CALENDAR_YEAR" NUMBER(4) NOT NULL ,
  "CALENDAR_YEAR_ID" NUMBER NOT NULL ,
  "FISCAL_YEAR" NUMBER(4) NOT NULL ,
  "FISCAL_YEAR_ID" NUMBER NOT NULL ,
  "DAYS_IN_CAL_YEAR" NUMBER NOT NULL ,
  "DAYS_IN_FIS_YEAR" NUMBER NOT NULL ,
  "END_OF_CAL_YEAR" DATE NOT NULL ,
  "END_OF_FIS_YEAR" DATE NOT NULL )
PARTITION BY RANGE(TIME_ID)
(
partition p2013q1 values less than (to_date('2013/04/01','YYYY/MM/DD')),
partition p2013q2 values less than (to_date('2013/07/01','YYYY/MM/DD')),
partition p2013q3 values less than (to_date('2013/10/01','YYYY/MM/DD')),
partition p2013q4 values less than (to_date('2014/01/01','YYYY/MM/DD')),
partition p2014q1 values less than (to_date('2014/04/01','YYYY/MM/DD')),
partition p2014q2 values less than (to_date('2014/07/01','YYYY/MM/DD')),
partition p2014q3 values less than (to_date('2014/10/01','YYYY/MM/DD')),
partition p2014q4 values less than (to_date('2015/01/01','YYYY/MM/DD')),
partition p2015q1 values less than (to_date('2015/04/01','YYYY/MM/DD')),
partition p2015q2 values less than (to_date('2015/07/01','YYYY/MM/DD')),
partition p2015q3 values less than (to_date('2015/10/01','YYYY/MM/DD')),
partition p2015q4 values less than (to_date('2016/01/01','YYYY/MM/DD')),
partition p2016q1 values less than (to_date('2016/04/01','YYYY/MM/DD')),
partition p2016q2 values less than (to_date('2016/07/01','YYYY/MM/DD')),
partition p2016q3 values less than (to_date('2016/10/01','YYYY/MM/DD')),
partition p2016q4 values less than (to_date('2017/01/01','YYYY/MM/DD')),
partition p2017q1 values less than (to_date('2017/04/01','YYYY/MM/DD')),
partition p2017q2 values less than (to_date('2017/07/01','YYYY/MM/DD')),
partition p2017q3 values less than (to_date('2017/10/01','YYYY/MM/DD')),
partition p2017q4 values less than (to_date('2018/01/01','YYYY/MM/DD'))
)
TABLESPACE "TBS1" PCTFREE 10 INITRANS 1 MAXTRANS 255
LOGGING NOCOMPRESS
;
```

```
insert /* +append */ into times
select
TIME_ID+15*365+4,
DAY_NAME,
DAY_NUMBER_IN_WEEK,
DAY_NUMBER_IN_MONTH,
CALENDAR_WEEK_NUMBER,
FISCAL_WEEK_NUMBER,
WEEK_ENDING_DAY,
WEEK_ENDING_DAY_ID,
CALENDAR_MONTH_NUMBER,
FISCAL_MONTH_NUMBER,
CALENDAR_MONTH_DESC,
CALENDAR_MONTH_ID,
FISCAL_MONTH_DESC,
FISCAL_MONTH_ID,
DAYS_IN_CAL_MONTH,
DAYS_IN_FIS_MONTH,
END_OF_CAL_MONTH,
END_OF_FIS_MONTH,
CALENDAR_MONTH_NAME,
FISCAL_MONTH_NAME,
to_number(to_char(TIME_ID,'YYYY')+15)||'-'||lpad(CALENDAR_QUARTER_NUMBER,2,'0'),
CALENDAR_QUARTER_ID,
FISCAL_QUARTER_DESC,
FISCAL_QUARTER_ID,
DAYS_IN_CAL_QUARTER,
DAYS_IN_FIS_QUARTER,
END_OF_CAL_QUARTER,
END_OF_FIS_QUARTER,
CALENDAR_QUARTER_NUMBER,
FISCAL_QUARTER_NUMBER,
to_number(to_char(TIME_ID,'YYYY')+15),
CALENDAR_YEAR_ID,
FISCAL_YEAR,
FISCAL_YEAR_ID,
DAYS_IN_CAL_YEAR,
DAYS_IN_FIS_YEAR,
END_OF_CAL_YEAR,
END_OF_FIS_YEAR
from times_org;
commit;

-- カスタマイズしたTIMES表に主キーを作成
create unique index TIMES_NEW_PK on TIMES(TIME_ID) local;
alter table TIMES add primary key (TIME_ID) using index;

-- SHスキーマのオプティマイザ統計情報を収集
```

```
exec dbms_stats.gather_schema_stats(ownname=>'SH',degree=>8);
```

はい、こちらは演習4でSALES表のTIME_ID列の値を15年分進化させた為に生じてしまった作業ですね。これまで私の連載で体験してきて頂いたことが殆どですから、特に問題ないかと思います。敢えて重要なポイントをリマインドさせて頂くと、最後のオプティマイザ統計情報収集の引数ですね。高速化の目的で、「estimate_percent」で10%程度のサンプルデータで統計情報！を実行している方はいらっしゃいませんか？ごめんなさい。それはちょっと古い知識です。最近のOracle Databaseでは何も指定しない方が高速だったりしますので、「第8回 オプティマイザ統計情報の管理～統計収集の高速化を体験してみる～」を復習しておいてくださいね。

**6. FILESYSTEMIO_OPTIONS初期化パラメータを「SETALL」に設定して、ファイルシステムに対する非同期I/Oとダイレクト I/Oを有効化してください。**

```
$ sqlplus / as sysdba
SQL>
alter system set FILESYSTEMIO_OPTIONS='SETALL' scope=spfile ;

shutdown immediate
startup
alter pluggable database POCO open ;
```

パフォーマンス測定を実施する上で必ず設定しておいてほしい初期化パラメータがこちらですね。「第5回 SQLの実行計画からパフォーマンスの違いを読み解く」を参考にしてみてください。

という事で、ようやくDemoを行う環境準備が完了しましたね。いざ、対象のクエリ（SELECT文）を実行してみましょう！

**7. 次のI/Oバウンドなクエリ（SELECT文）を実行し、その実行時間、傾向やボトルネックをリアルタイムSQL監視レポートで確認してください。**

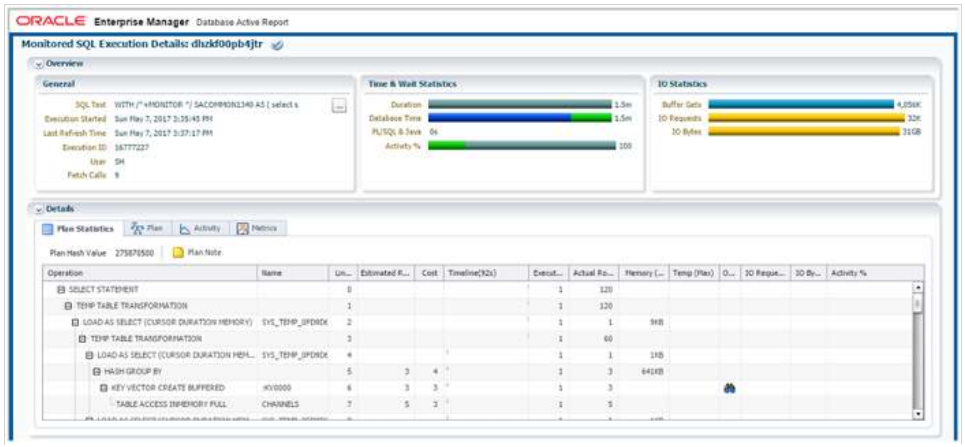
```
$ view query_IO.sql

WITH /*+MONITOR */
SACOMMON1340 AS
(  select sum(T220.AMOUNT_SOLD) as c1, sum(T220.QUANTITY_SOLD) as c2,
          T147.CHANNEL_CLASS as c3, T228.CALENDAR_QUARTER_DESC as c4,
          T228.CALENDAR_YEAR as c5, T185.PROD_CATEGORY as c6
    from CHANNELS T147, PRODUCTS T185,
          SALES T220, TIMES T228
   where ( T220.TIME_ID < to_date('2014/01/01','YYYY/MM/DD')
     and T228.TIME_ID = T220.TIME_ID
     and T147.CHANNEL_ID = T220.CHANNEL_ID
     and T185.PROD_ID = T220.PROD_ID)
   group by T147.CHANNEL_CLASS,
            T185.PROD_CATEGORY,
            T228.CALENDAR_QUARTER_DESC,
            T228.CALENDAR_YEAR),
SAWITH0 AS
(  select distinct 0 as c1, D1.c3 as c2, D1.c4 as c3, D1.c5 as c4,
                   D1.c6 as c5, D1.c2 as c6, D1.c1 as c7, cast(NULL as DOUBLE PRECISION ) as c8
  from SACOMMON1340 D1),
SAWITH1 AS
(  select D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4,
          D1.c5 as c5, D1.c6 as c6, D1.c7 as c7, D1.c8 as c8, sum(D1.c7) as c9
  from SAWITH0 D1
  group by D1.c1, D1.c2, D1.c3, D1.c4, D1.c5, D1.c6, D1.c7, D1.c8),
SAWITH2 AS
(  select distinct 1 as c1, D1.c3 as c2, D1.c4 as c3, D1.c5 as c4,
          D1.c6 as c5, D1.c2 as c6, D1.c1 as c7
  from SACOMMON1340 D1),
SAWITH3 AS
(  select D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4,
          D1.c5 as c5, D1.c6 as c6, D1.c7 as c7, sum(D1.c6) as c8, sum(D1.c7) as c9
  from SAWITH2 D1
  group by D1.c1, D1.c2, D1.c3, D1.c4, D1.c5, D1.c6, D1.c7),
SAWITH4 AS
((    select D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4, D1.c5 as c5,
           D1.c6 as c6, D1.c7 as c7, D1.c8 as c8,
           sum(D1.c9) over (partition by D1.c3, D1.c4, D1.c5) as c9
      from SAWITH1 D1
    union all
    select D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4, D1.c5 as c5,
           D1.c6 as c6, D1.c7 as c7,
           sum(D1.c8) over (partition by D1.c3, D1.c4, D1.c5) as c8,
           sum(D1.c9) over (partition by D1.c3, D1.c4, D1.c5) as c9
      from SAWITH3 D1   ))
select D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4, D1.c5 as c5,
       D1.c6 as c6, D1.c7 as c7, D1.c8 as c8, D1.c9 as c9
from SAWITH4 D1 order by c1, c3, c5, c4;
```

```
$ sqlplus sh/oracle12345@localhost:1521/poco
SQL>
set time on timing on pages 0 lines 200
@query_IO.sql

...(省略)...

経過: 00:01:29.52
```

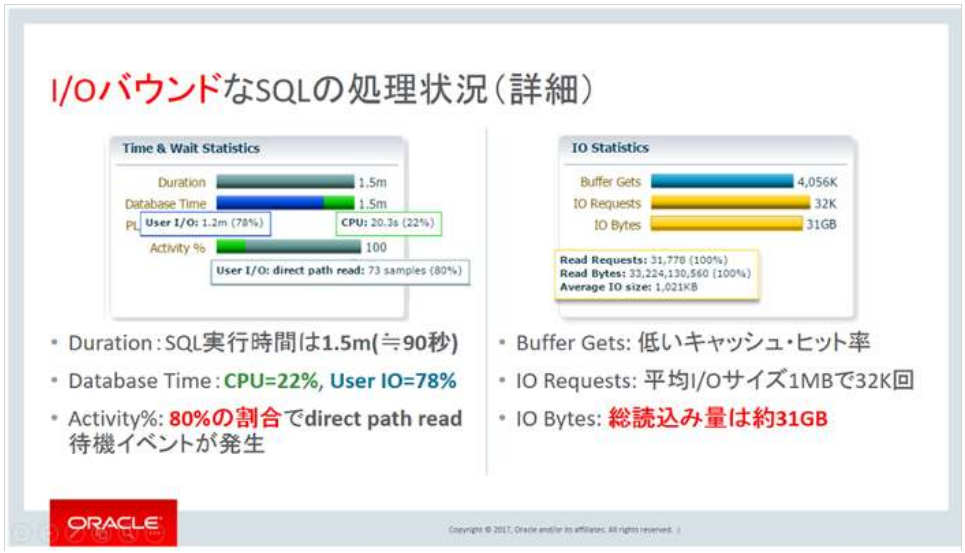
チャット

私のDemo環境では約90秒程度で実行が完了しました。この実行時間は主にディスク装置の性能に依存します。どこかのデータ分析ツールから出力されたSELECT文を私が少し手直ししてクエリですが、with句が連続して並んでいて正直目が回ります。このSQLを書き直してチューニングすることなんて誰もしたくないですよね。なので、今回は、このSELECT文は最後までこのままの状態で実行していきますよ。

リアルタイムSQL監視レポートに確実に出力せるために、「MONITOR」ヒント句を埋め込んでいることが特徴です。Enterprise ManagerでSQL監視の画面に移動して頂ければ、次のようなレポートを確認することが可能です。もし、Enterprise Manager環境が無い場合には、「第6回 続・SQLの実行計画からパフォーマンスの違いを読み解く」でご紹介している、DBMS_SQLTUNEパッケージのREPORT_SQL_MONITORファンクションを使用して、Activeレポートを取得してみてください。




こちらのレポートの分析結果は、セミナーのスライドを流用させてください。（新幹線での移動中に、こちらの連載を執筆していて、締切まであと1時間しかないのです。お許しを〜）




**8. 次のCPUバウンドなクエリ（SELECT文）を実行し、その実行時間、傾向やボトルネックをリアルタイムSQL監視レポートで確認してください。**

```
$ view query_CPU.sql

WITH /*+MONITOR */
DUMMY_SALES AS
( select * from (select 0 from CHANNELS ) D1, sales D2),
SACOMMON1340 AS
(  select sum(T220.AMOUNT_SOLD) as c1, sum(T220.QUANTITY_SOLD) as c2,
          T147.CHANNEL_CLASS as c3, T228.CALENDAR_QUARTER_DESC as c4,
          T228.CALENDAR_YEAR as c5, T185.PROD_CATEGORY as c6
     from CHANNELS T147, PRODUCTS T185,
          DUMMY_SALES T220, TIMES T228
    where ( T220.TIME_ID < to_date('2014/01/01','YYYY/MM/DD')
      and T228.TIME_ID = T220.TIME_ID
      and T147.CHANNEL_ID = T220.CHANNEL_ID
      and T185.PROD_ID = T220.PROD_ID)
    group by T147.CHANNEL_CLASS,
             T185.PROD_CATEGORY,
             T228.CALENDAR_QUARTER_DESC,
             T228.CALENDAR_YEAR),
SAWITH0 AS
(  select distinct 0 as c1, D1.c3 as c2, D1.c4 as c3, D1.c5 as c4,
                   D1.c6 as c5, D1.c2 as c6, D1.c1 as c7, cast(NULL as DOUBLE PRECISION ) as c8
   from SACOMMON1340 D1),
SAWITH1 AS
(  select D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4,
          D1.c5 as c5, D1.c6 as c6, D1.c7 as c7, D1.c8 as c8, sum(D1.c7) as c9
   from SAWITH0 D1
   group by D1.c1, D1.c2, D1.c3, D1.c4, D1.c5, D1.c6, D1.c7, D1.c8),
```

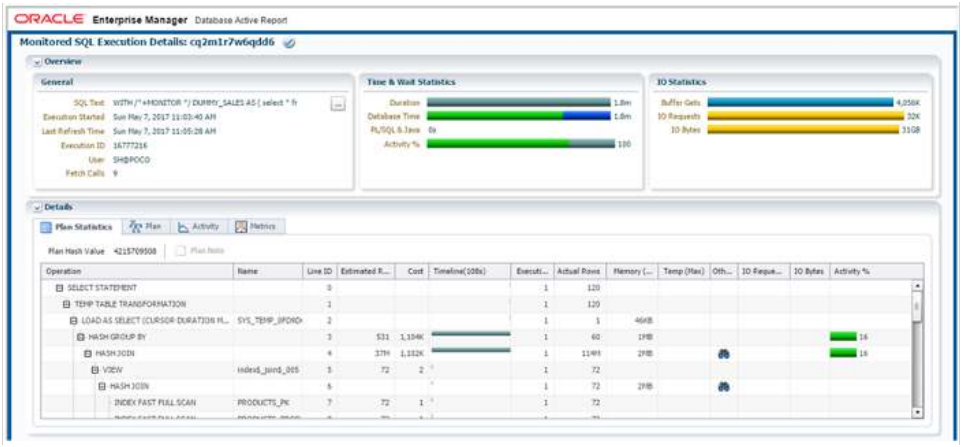
チャット

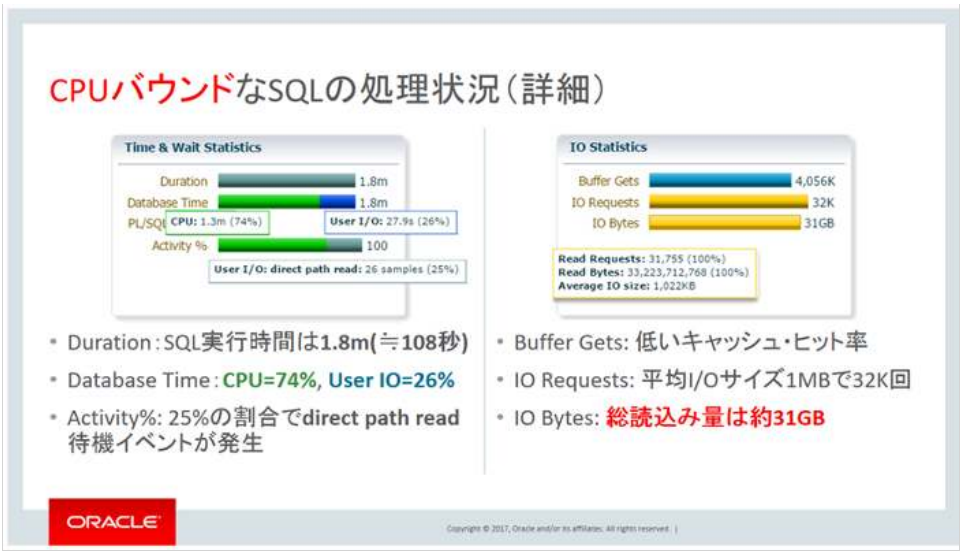
```
SAWITH2 AS
(  select distinct 1 as c1, D1.c3 as c2, D1.c4 as c3, D1.c5 as c4,
          D1.c6 as c5, D1.c2 as c6, D1.c1 as c7
  from SACOMMON1340 D1),
SAWITH3 AS
(  select D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4,
          D1.c5 as c5, D1.c6 as c6, D1.c7 as c7, sum(D1.c6) as c8, sum(D1.c7) as c9
  from SAWITH2 D1
  group by D1.c1, D1.c2, D1.c3, D1.c4, D1.c5, D1.c6, D1.c7),
SAWITH4 AS
((    select D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4, D1.c5 as c5,
           D1.c6 as c6, D1.c7 as c7, D1.c8 as c8,
           sum(D1.c9) over (partition by D1.c3, D1.c4, D1.c5) as c9
      from SAWITH1 D1
    union all
    select D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4, D1.c5 as c5,
           D1.c6 as c6, D1.c7 as c7,
           sum(D1.c8) over (partition by D1.c3, D1.c4, D1.c5) as c8,
           sum(D1.c9) over (partition by D1.c3, D1.c4, D1.c5) as c9
      from SAWITH3 D1   ))
select D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4, D1.c5 as c5,
       D1.c6 as c6, D1.c7 as c7, D1.c8 as c8, D1.c9 as c9
from SAWITH4 D1 order by c1, c3, c5, c4;
```

```
$ sqlplus sh/oracle12345@localhost:1521/poco
SQL>
set time on timing on pages 0 lines 200
@query_CPU.sql

...(省略)...

経過: 00:01:46.81
```

先ほどのクエリは、I/Oバウンド（I/Oを良く使用する）なSQL文でしたが、こちらのもう一つのクエリは、CPUバウンド（CPUを良く使用する）SQL文となります。ベースは、I/OバウンドなSQL文であり、先頭の方にワザと直積するサブ・クエリを追加していることで、メモリ上でレコード数を5倍に増幅することで、I/O量を変えずにCPU処理量だけを増やす検証ならではのテクニックを使っていますので、参考にしてみてください。

つまり、こちらの実行時間は、I/OバウンドなSQL文に対して、CPU処理時間が上乗せされるわけですから、I/OバウンドなSQL文よりも実行時間が長くなるのは私としては当たり前だったりします。



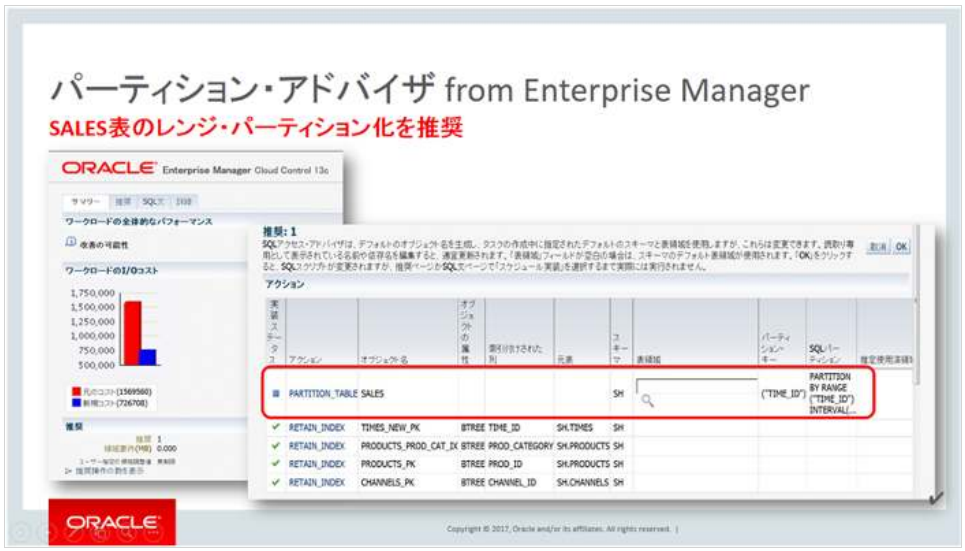




チャット

では、この2つのクエリの実行時間を、SQL文を書き換えずに高速化していく！というのが、今回の私のDemoの主旨でした。という事で、一つ目

のチューニング方法としてパーティション化を体験してみましょう。

**9. Enterprise ManagerからSQLアクセス・アドバイザのパーティション・アドバイザを実行し、直前で実行した2つのクエリに適したパーティション構成の推奨を得てください。**


Enterprise Managerで実行すると、SALES表をTIME_ID列をパーティション・キーとしたレンジ・パーティション化せよと、アドバイスされれば成功です。直前で実行していたSQL文（ワークロード）を参考としたアドバイスですから、間違いはありませんよね。もちろん、この推奨（パーティション化）によって性能劣化を引き起こすSQL文が存在すると推測される場合には、その情報も結果として出力されますので、とても便利な機能ですよね。ぜひ色々と試してみてください。

**10. Oracle Database 12c Releaes 2から実装された、オンラインで非パーティション表からパーティション表へ変換する機能を使用して、演習8でアドバイスされたSALES表をパーティション化（TIME_ID列をパーティション・キーとしたレンジ・パーティション）を実行してください。**

```
$ sqlplus sh/oracle12345@localhost:1521/poco
SQL>
alter table SALES modify partition by range(TIME_ID)
interval(numtoyminterval(1, 'MONTH'))
(partition values less than (to_date('2013-01-01 00:00:00','YYYY-MM-DD HH24:MI:SS'))) online
-- update indexes
 ;
```

はい、こちらは、Oracle Database 12c Release 2でヒッソリと新機能一覧に記載されていた機能です。私はこういった細かい新機能が大好きで、新機能ガイドを上から下までじっくり眺めていたりします。古くからの機能である、表のオンライン再定義でも同じことを行うことが可能ですが、ここまでシンプルになるとこちらを使いたくなりますよね。まだまだ私も使い切れてはいませんので、次回以降の連載でも詳しく取り上げていきたいと思います。皆さんも試してみてくださいね。（原稿の締め切りが・・・）

**11. 演習7で実行したI/Oバウンドなクエリ（query_IO.sql）を実行し、パーティション化による高速化の効果と傾向を確認してください。**

```
$ sqlplus sh/oracle12345@localhost:1521/poco
SQL>
set time on timing on pages 0 lines 200
@query_IO.sql

...(省略)...

経過: 00:00:19.43
```

じゃーん！！ですね。非パーティション表の場合に約90秒間要していたクエリが、4倍以上も高速化しているではありませんか！とちょっと驚いてください。今回のSALES表は、2013年～2016年の4年間分のデータが格納されており、SQL文のWhere句（where ( T220.TIME_ID < to_date('2014/01/01','YYYY/MM/DD')）を確認してみると、2013年の1年間分のみが集計対象となることがわかります。つまり、非パーティション表では4年間分の全レコードをDisk上から読み込んでいたのに対して、パーティション化することで一年分のレコードだけDisk上から読めば良くなった。つまりDisk I/O量が1/4まで減少していることがI/O時間の短縮につながり、結果、SQL文の実行時間の高速化が実現しているのです。これが、復習になりますが、パーティション・プルーニングの効果でしたよね。「第25回 パーティション化による問い合わせのパフォーマンスの向上」を参考にしてみてください。


**12. 演習8で実行したCPUバウンドなクエリ（query_CPU.sql）を実行し、パーティション化による高速化の効果と傾向を確認してください。**

```
$ sqlplus sh/oracle12345@localhost:1521/poco
SQL>
set time on timing on pages 0 lines 200
@query_CPU.sql

...(省略)...

経過: 00:01:11.89
```

同様に、CPUバウンドなクエリもパーティション化したSALES表に対して実行してみると、あれ？高速化はしていますが、IOバウンドなクエリほどは改善していませんね。それはCPUバウンドだからです。実行時間の内訳としてCPU処理時間の方が大きいので、パーティション・プルーニングの効果も大きくはないのですね。


さてさて、いかがでしたでしょうか？今回の演習は、基本的には過去の連載で取り上げてきた機能を使っているだけですが、より実践に近い形で活用する例として皆様の参考になれば幸いでございます。まだ前半だけしか解説できていませんので、次回は後半として、パラレル実行、圧縮、Database In-Memoryの効果を一挙にご紹介していきたいと思います。

原稿の締め切りまで残り10分。。。駆け足で説明不足な点は申し訳ございません。時間を見つけて適宜Updateをかけていきたいと考えております。という事で、今回も最後までお付き合い頂きまして、ありがとうございました。次回もご愛読のほど、よろしくお願いします！

しばちょう先生の試して納得！DBAへの道　indexページ▶▶