TECHNICAL NOTE

ORACLE DATE TO TIMESTAMP CONVERSION DURING PEOPLETOOLS UPGRADE

Prepared By David Kurtz, Go-Faster Consultancy Ltd.
Technical Note
Version 1.00
Friday 3 January 2014
(E-mail: david.kurtz@go-faster.co.uk, telephone +44-7771-760660)
File: date_to_timestamp_conversion.docx, 3 January 2014

Contents

Timestamps in PeopleSoft

I am working on a PeopleSoft upgrade project. We are going from PeopleTools 8.49 to 8.53. One of the things that happen is that some date columns in the Oracle database become timestamps.

Timestamps were introduced by Oracle in version 10g of the database, and provide the ability to store times accurate to the nanosecond (although the default is microsecond). Dates are accurate to the whole second.

There are 3 types of temporal column in PeopleSoft as defined on PSDBFIELD. Prior to PeopleTools 8.50 they all become Oracle data columns in the database. However, from PeopleTools 8.50; Time and DateTime fields are built as Timestamp columns.if bit 5 of PSSTATUS.DATABASE_OPTIONS (value 32) is set.

PeopleTools Field Type		Database Column Type	
		PeopleTools <= 8.49	**PeopleTools >= 8.50**
4	**Date**	DATE	DATE
5	**Time**	DATE	**TIMESTAMP**
6	**DateTime**	DATE	**TIMESTAMP**

Timestamps must be handled differently to dates in SQL. Some date arithmetic must be done differently; in particular the difference between two timestamps is a timestamp rather than a number of days. Therefore this setting also controls how PeopleCode date macros expand on Oracle.

During the upgrade, PeopleSoft Change Assistant simply alters all the Time and DateTime columns from dates to timestamps. This generally works well. The data value doesn't appear to get longer, so the block doesn't run out of room leading to row migration, and so it isn't necessary to rebuild every table that is affected.

However, there are some limitations. If the column being converted to a timestamp falls into one of the following categories you will get an error.

- The column is a key column in a function-based index.
- The table or index is partitioned by the column.

The functional index key issue has not affected PeopleSoft customers because the delivered upgrade template drops all the indexes before altering the tables and rebuilds them again afterwards.

However, dropping and recreating all these indexes can be very time consuming and increases the duration of the outage required to perform the upgrade. This has been my incentive to find a better way.

Function-Based Indexes

PeopleSoft Application Desginer defines some key and search fields as descending. The rows in components and the results of search dialogue are sorted on the key fields in the order specified. Application Designer then indexes these columns in descending order (prior to PeopleTools 8 and since PeopleTools 8.47). If any column in an Oracle index is in decending order the index is created as a function-based index. Consequently, there can be a lot of descending indexes in a PeopleSoft system! HCM and Campus Solutions products are particularly affected because many tables are effective-dated, and the field EFFDT is usually a descending key field.

It is not possible to alter a column to a timestamp if it appears anywhere in a function-based index. You get the following error:

```
ORA-30556: functional index is defined on the column to be modified
```

Although rebuilding all the indexes does have some advantages, there are a lot of indexes and it can be a time-consuming task when only a small proportion of indexes are actually affected, typically less than 5%.

Partitioning

Partitioning is not something that you encounter in a vanilla PeopleSoft system, but it can be added by customisation. You must generate the necessary DDL yourself if you want to use it. However, from PeopleTools 8.51 Application Designer will preserve existing partitioning.

In the system on which I am working, a number of audit tables a partitioned by AUDIT_STAMP which is a DateTime field.

```
ORA-14060: data type or length of an[1] table partitioning column may not
be changed
```

We have no alternative but to rebuild these tables and repopulate the data. This has also dealt with all locally partitioned indexes.

We have also found that we have one global index partitioned on a timestamp.

```
ORA-14061: data type or length of an index partitioning column may not
be changed
```

We also have had to drop this index in order to alter the table.

[1] This grammatically error is in the message!

My Approach

We have had no alternative but to rebuild and repopulate our partitioned audit tables which are partitioned by a DateTime field. However, that is what we did when we first partitioned them. The scripts are very similar to those generated by Application Designer. The table is renamed, a new one is built, and the data is copied. In our case these scripts are built with a PL/SQL utility[2]. This also addressed the need to rebuild the locally partitioned indexes.

To minimize the number of indexes which must be rebuilt I have written a PL/SQL script that:

- identifies the indexes that need to be dropped,
- captures the DDL to recreate the indexes using DBMS_METADATA and stores it in a table,
- drops the indexes,
- alters the columns that cannot be altered with the index in place,
- recreates the index.

The script successfully handles partitioned function-based indexes.

On this particular HR system we only rebuilt about 400 indexes instead of over 10000. Now the standard PeopleSoft upgrade template can be run without dropping or recreating any further indexes.

[2] See http://www.go-faster.co.uk/gpdoc.htm#Managing_Oracle_Table_Partitioning

APPENDIX

PL/SQL Script

gfc_desc_timestamp_index.sql

This PL/SQL script can be downloaded from http://www.go-faster.co.uk/scripts/gfc_desc_timestamp_index.sql. It is designed to be run in SQL*Plus logged in as SYSADM (the owner of the PeopleSoft database).

The script contains two anonymous PL/SQL blocks. The first identifies the indexes that need to be dropped and the columns to be altered, and the second does the work.

In this document, I have commented on the script using footnotes.

```
REM gfc_desc_timestamp_index.sql
REM (c)Go-Faster Consultancy 2013
REM look for indexes with descending columns that need to converted from dates to timestamp
REM could have several columns in one index or several indexes with the same column

set echo on pause off timi on trimspool on pages 99 lines 131 long 32767 serveroutput on
column index_ddl format a131
column column_expression format a20
column column_position heading 'Col|Pos' format 999
column data_type format a10
spool gfc_desc_timestamp_index0

lock table gfc_desc_timestamp_index in exclusive mode
/
pause

drop table gfc_desc_timestamp_index purge
/

CREATE TABLE gfc_desc_timestamp_index[3]
(recname VARCHAR2(15) NOT NULL
,table_name VARCHAR2(18) NOT NULL
,index_name VARCHAR2(18) NOT NULL
,column_position INTEGER NOT NULL
,column_name VARCHAR2(18)
,column_expression CLOB
,index_ddl CLOB
,insert_timestamp TIMESTAMP DEFAULT SYSTIMESTAMP NOT NULL
,CONSTRAINT gfc_desc_timestamp_index PRIMARY KEY (table_name, index_name, column_position)
)
/

TRUNCATE TABLE gfc_desc_timestamp_index
/
```

[3] This table holds the list of columns and indexes to be processed by the script.

```
--this procedure finds the columns
DECLARE

PROCEDURE alterit[4]
(p_recname           VARCHAR2
,p_table_name        VARCHAR2
,p_index_name        VARCHAR2
,p_column_position   INTEGER
,p_column_name       VARCHAR2
,p_data_type         VARCHAR2
,p_column_expression LONG DEFAULT NULL
) IS
  l_column_name VARCHAR2(18);
  l_ddl CLOB;
  l_stattype_locked VARCHAR2(5);
BEGIN
  dbms_output.put_line(TO_CHAR(SYSDATE,'hh24:mi:ss')||'>'||p_recname||'/'||p_table_name||'/'||p_index_name
                       ||'/'||p_column_position||'/'||p_column_name||'/'||p_column_expression);
  BEGIN
    IF p_column_expression IS NULL THEN --check the column in peoplesoft
      dbms_output.put_line(TO_CHAR(SYSDATE,'hh24:mi:ss')||'>Testing column '||p_column_name);
      SELECT f.fieldname[5]
      INTO   l_column_name
      FROM   psrecfielddb r
      ,      psdbfield f
      WHERE  f.fieldname = r.fieldname
      AND    f.fieldtype IN(5,6) --time/datetime column in peoplesoft
      AND    r.recname = p_recname
      AND    r.fieldname = p_column_name
      AND    p_data_type = 'DATE'; --date column in the database
    ELSE  --interpret column expression to get column
      dbms_output.put_line(TO_CHAR(SYSDATE,'hh24:mi:ss')||'>Testing expression '||p_column_expression);
      SELECT f.fieldname[6]
      INTO   l_column_name
      FROM   psrecfielddb r
      ,      psdbfield f
      ,      user_tab_columns c
      WHERE  f.fieldname = r.fieldname
      AND    f.fieldtype IN(5,6) --time/datetime column in peoplesoft
      AND    r.recname = p_recname
      AND    INSTR(UPPER(p_column_expression),UPPER('"'||r.fieldname||'"'))>0
```

[4] This procedure is used to test each index/column combination.

[5] This query is used when the index expression is null because the index column is not descending or part of a function. It checks that the PeopleSoft field that matches the column is either a DateTime field (type 5) or a Time field (type 6).

[6] This query is used when the index expression is not null. It checks that either a DateTime field (type 5) or a Time field (type 6) field in PeopleSoft is referenced in the index expression and that it that matches a PeopleSoft DateTime field (type 5) or a Time field (type 6).

```
  AND    c.table_name = p_table_name
  AND    c.column_name = r.fieldname
  AND    c.data_type = 'DATE'; --date column in the database
END IF;
dbms_output.put_line('Column '||p_column_name||' is date and should be a timestamp');[7]

l_ddl := dbms_metadata.get_ddl('INDEX',p_index_name); --extract create index DDL[8]

BEGIN --check if table stats are locked[9]
  SELECT stattype_locked
  INTO l_stattype_locked
  FROM   user_tab_statistics s
  WHERE  s.table_name = p_table_name
  AND    s.partition_name IS NULL
  AND    s.subpartition_name IS NULL;

  IF l_stattype_locked = 'ALL' THEN --remove compute stats on index DDL if stats locked on table
    l_ddl := REPLACE(l_ddl,'COMPUTE STATISTICS','');
  END IF;

EXCEPTION WHEN no_data_found THEN --table stats not locked
  l_stattype_locked := '';
END;


IF INSTR(UPPER(l_ddl),'NOLOGGING') > 0 THEN[10]
  NULL; --already contains NOLOGGING
ELSIF INSTR(UPPER(l_ddl),' LOGGING') > 0 THEN
  l_ddl := REPLACE(l_ddl,' LOGGING',' NOLOGGING');
ELSE
  l_ddl := l_ddl||' NOLOGGING';
END IF;

IF INSTR(UPPER(l_ddl),'NOPARALLEL') > 0 THEN[11]
  l_ddl := REPLACE(l_ddl,' NOPARALLEL',' PARALLEL');
ELSIF INSTR(UPPER(l_ddl),' PARALLEL') > 0 THEN
  NULL; --already contains PARALLEL
```

[7] If either of the preceding queries returns a row then the index will prevent the date column being converted to a timestamp.

[8] The DDL to recreate the index is extracted using the Oracle supplied DBMS_METADATA package.

[9] If the table statistics are locked then the COMPUTE STATISTICS keyword will be removed from the create index command.

[10] The NOLOGGING keyword is added if it is not already there. If the LOGGING keyword is present it will be replaced with NOLOGGING.

[11] The PARALLEL keyword is added if it is not already there. If the NOPARALLEL keyword is present it will be replaced with PARALLEL. Note that if there is a non-default degree of parallelism specified that will be preserved.

```
    ELSE
      l_ddl := l_ddl||' PARALLEL';
    END IF;

    --so we only insert the row if we can find a row in preceding query
    INSERT INTO gfc_desc_timestamp_index[12]
    (recname, table_name, index_name, column_position, column_name, column_expression, index_ddl)
    VALUES
    (p_recname, p_table_name, p_index_name, p_column_position, l_column_name, p_column_expression, l_ddl);
    COMMIT;

  EXCEPTION
    WHEN no_data_found THEN NULL; --not an error but jump insert if column lookup fails
  END;

END alterit;

BEGIN
  FOR i IN ([13] --first query looks for function based indexes with an columns DATEs that should become
TIMESTAMPs
    WITH n AS ([14]
           SELECT LTRIM(TO_CHAR(rownum,'99')) n
           FROM dual
           CONNECT BY LEVEL <= 99
    ), r AS ([15]
           SELECT r.recname
           ,      DECODE(r.sqltablename,' ','PS_'||r.recname,r.sqltablename) table_name
           FROM   sysadm.psrecdefn r
           WHERE  r.rectype IN(0,7) --records or temporary records
           UNION ALL
           SELECT r.recname[16]
           ,      DECODE(r.sqltablename,' ','PS_'||r.recname,r.sqltablename)||n.n[17]
           FROM   sysadm.psrecdefn r, n
           WHERE  r.rectype IN(7) --records or temporary records
    )
    SELECT /*+LEADING(r n s i ic tc ie)*/
           r.recname, r.table_name, i.index_name, ic.column_position, ic.column_name, tc.data_type
    ,      ie.column_expression
    FROM   r
```

[12] The DDL to rebuild the index is inserted into the working storage table.

[13] This cursor looks for function-based indexes. It returns ascending date columns and all expressions. It does not filter the expressions because they are held in a long column.

[14] This in-line query simply returns 99 rows with the numbers 1 to 99 in a string without any leading characters. This will be used to produce the list of non-shared temporary records.

[15] This in-line query returns the list of all tables to be checked

[16] There can be up to 99 non-shared instances of temporary records.

[17] Note that the instance number is simply appended to the table name

```
    ,      user_indexes i
    ,      user_ind_columns ic
           LEFT OUTER JOIN user_tab_columns tc
              ON  tc.table_name = ic.table_name
              AND tc.column_name = ic.column_name
              AND tc.data_type = 'DATE'
           LEFT OUTER JOIN user_ind_expressions ie
              ON  ie.table_name = ic.table_name
              AND ie.index_name = ic.index_name
              AND ie.column_position = ic.column_position
    WHERE  i.table_name = r.table_name
    AND    i.index_type = 'FUNCTION-BASED NORMAL'[18]
    AND    ic.table_name = r.table_name
    AND    ic.table_name = i.table_name
    AND    ic.index_name = i.index_name
    AND   (tc.data_type = 'DATE' OR ie.column_expression IS NOT NULL)[19]
--  AND    r.recname = 'FO_TL_RPDTM_TA3'
--  AND    r.recname = 'TL_EXCEPTION'
--  AND    ROWNUM <= 50
  ) LOOP
    alterit(i.recname
           ,i.table_name
           ,i.index_name
           ,i.column_position
           ,i.column_name
           ,i.data_type
           ,i.column_expression
           )[20];
  END LOOP;

  FOR i IN ([21]--2nd query looks for NORMAL (ie not function based) GLOBAL partitioned indexes partitioned
on DATE columns that should become TIMESTAMPs
    WITH r AS (
           SELECT r.recname
           ,      DECODE(r.sqltablename,' ','PS_'||r.recname,r.sqltablename) table_name
           FROM   sysadm.psrecdefn r
           WHERE  r.rectype=0 --regular records only[22]
    )
    SELECT /*+LEADING(r pi kc)*/ r.recname, r.table_name, i.index_name, ic.column_position, ic.column_name,
tc.data_type
```

[18] For each table we are looking for any function based indexes

[19] Either the indexed column is a date column (rather than a timestamp or it is a function because there is an entry in *user_ind_expressions* which has been outer-joined in this query.

[20] Each indexed column produced by the query will be tested by the *alterit()* function

[21] This query is similar to the first, but we are only looking for regular table records with globally partitioned indexes partitioned on DATE columns.

[22] We don't need to inspect temporary records because they would never been partitioned. It simply wouldn't make sense.

```
    FROM   r
    ,      user_indexes i
    ,      user_part_indexes pi
    ,      user_part_key_columns kc
    ,      user_ind_columns ic
    ,      user_tab_columns tc
    WHERE  i.table_name = r.table_name
    AND    i.index_type = 'NORMAL'
    AND    pi.table_name = r.table_name
    AND    pi.index_name = i.index_name
    AND    pi.locality = 'GLOBAL' --global partitioned index
    AND    ic.table_name = r.table_name
    AND    ic.index_name = i.index_name
    AND    tc.table_name = ic.table_name
    AND    tc.column_name = ic.column_name
    AND    tc.data_type = 'DATE'
    AND    kc.object_type = 'INDEX'
    AND    kc.name = i.index_name
    AND    kc.name = pi.index_name
    AND    kc.column_name = ic.column_name[23]
--  AND    r.recname = 'TL_EXCEPTION'
  ) LOOP
    alterit(i.recname
           ,i.table_name
           ,i.index_name
           ,i.column_position
           ,i.column_name
           ,i.data_type
           );
  END LOOP;
END;
/

pause
spool gfc_desc_timestamp_index1
set lines 80
ttitle 'Number of Tables / Indexes / Columns to convert to Oracle Timestamp'
select count(distinct table_name) tables
,      count(distinct index_name) indexes
,      count(*) columns
from gfc_desc_timestamp_index x
/

ttitle 'Columns to be converted to Oracle Timestamp that appear in more than one index'
select table_name, column_name, count(*) indexes
from gfc_desc_timestamp_index x
group by table_name, column_name
having count(*)>1
/
```

[23] We are looking for columns in a globally partitioned index (listed on *user_part_indexes*) that are also partitioning columns (listed on *user_part_key_columns*)

```
ttitle 'Indexes that have more than one column to convert to Oracle Timestamp'
select table_name, index_name, count(*) columns
from gfc_desc_timestamp_index x
group by table_name, index_name
having count(*)>1
/

set lines 131
ttitle 'Summary of Index Rebuilds'
select *
from gfc_desc_timestamp_index
order by table_name, index_name
/
ttitle off

pause
set serveroutput on
spool gfc_desc_timestamp_index2

ALTER TRIGGER psft_ddl_lock ENABLE;
EXECUTE psft_ddl_lock.ddl_permitted(TRUE);

DECLARE
  l_sql CLOB;

  PROCEDURE exec_sql(p_sql CLOB) IS[24]
    l_cursor_name    INTEGER;
    l_rows_processed INTEGER;

    l_sql           dbms_sql.varchar2s;
    l_chunk_size     INTEGER := 256; --default chunk size
    l_counter        INTEGER := 0; --chunk counter
    l_offset         INTEGER := 1; --starting position of current chunk
    l_sql_length     INTEGER := 0; --overall SQL length
  BEGIN
    l_sql_length := LENGTH(p_sql);
    WHILE l_offset < l_sql_length LOOP
      l_counter := l_counter + 1;
      l_sql(l_counter) := SUBSTR(p_sql,l_offset,l_chunk_size);
      dbms_output.put_line('['||l_counter||']'||l_sql(l_counter));
      l_offset := l_offset + l_chunk_size;
    END LOOP;
    dbms_output.put_line('['||l_sql_length||','||l_counter||']');
    --EXECUTE IMMEDIATE ''||p_sql; --cannot pass clob to EXECUTE IMMEDIATE in 10g

    l_cursor_name := dbms_sql.open_cursor;
    dbms_sql.parse(c => l_cursor_name
                  , statement => l_sql
```

[24] This procedure executed the DDL in the CLOB passed to it. It uses the DBMS_SQL package because in Oracle 10g you can only pass DDL to EXECUTE IMMEDIATE in a VARCHAR. It is only possible to use a CLOB from Oracle 11g. The procedure breaks the SQL into 256 character chunks.

```
                  , lb => 1
                  , ub => l_counter
                  , lfflg => false
                  , language_flag => dbms_sql.native);
    l_rows_processed := dbms_sql.execute(l_cursor_name);
    dbms_sql.close_cursor(l_cursor_name);
  END exec_sql;

BEGIN
  FOR i IN([25]
    SELECT DISTINCT table_name
    FROM   gfc_desc_timestamp_index
--  WHERE  table_name = 'PS_ORG_COMM' --just for testing
    ORDER BY 1
  ) LOOP

    FOR j IN ([26]
      SELECT DISTINCT i.index_name
      FROM   gfc_desc_timestamp_index x
      ,      user_indexes i
      WHERE  x.table_name = i.table_name
      AND    x.table_name = i.table_name
      AND    x.index_name = i.index_name
      ORDER BY 1
    ) LOOP
      l_sql := 'DROP INDEX '||j.index_name;
      exec_sql(l_sql);
    END LOOP;

    FOR j IN ([27]
      SELECT DISTINCT column_name
      FROM   gfc_desc_timestamp_index
      WHERE  table_name = i.table_name
      ORDER BY 1
    ) LOOP
      l_sql := 'ALTER TABLE '||i.table_name||' MODIFY ('||j.column_name||' TIMESTAMP)';
      exec_sql(l_sql);
    END LOOP;
```

[25] This PL/SQL simply works though the contents of the table populated by the previous PL/SQL block for each DISTINCT table.

[26] For each table, each index referenced in the working storage table is dropped.

[27] For each table, each affected date column can now be altered to a timestamp.

```
    FOR j IN (28
      SELECT row_number() over (partition by index_name order by column_name) seq29
      ,      index_name
      ,      index_ddl
      FROM   gfc_desc_timestamp_index
      WHERE  table_name = i.table_name
      ORDER BY 1
    ) LOOP
      IF j.seq = 1 THEN --cannot do distinct on a clob
        exec_sql(j.index_ddl);
        exec_sql('ALTER INDEX '||j.index_name||' NOPARALLEL LOGGING');30
      END IF;
    END LOOP;

  END LOOP;
END;
/

spool off
```

[28] Now we can start to put the indexes back. For each table, each index is rebuilt.

[29] It is not possible to do a DISTINCT on a CLOB or LONG. It is possible to get multiple rows in the working storage table for the same index if there are multiple date columns to be converted. So instead I have used an analytic function to count the number of rows for each index and we process the first one.

[30] Each index is altered to be LOGGING and NOPARALLEL. Note, this will overwrite any custom parallelism on the index.

Sample Spool Files

A number of spool files are created by the script.

gfc_desc_timestamp_index0.lst

This file simply reports on the set up and execution of the first PL/SQL block.

```
SYSADM-MHRSYT2A.1.475:28265.MHRSYT2A.oracle.thrvm>
SYSADM-MHRSYT2A.1.475:28265.MHRSYT2A.oracle.thrvm>drop table gfc_desc_timestamp_index
  2  /

Table dropped.

Elapsed: 00:00:01.07
SYSADM-MHRSYT2A.1.475:28265.MHRSYT2A.oracle.thrvm>
SYSADM-MHRSYT2A.1.475:28265.MHRSYT2A.oracle.thrvm>CREATE TABLE gfc_desc_timestamp_index
  2  (recname VARCHAR2(15) NOT NULL
  3  ,table_name VARCHAR2(18) NOT NULL
  4  ,index_name VARCHAR2(18) NOT NULL
  5  ,column_position INTEGER NOT NULL
  6  ,column_name VARCHAR2(18)
  7  ,column_expression CLOB
  8  ,index_ddl CLOB
  9  ,CONSTRAINT gfc_desc_timestamp_index PRIMARY KEY (table_name, index_name, column_position)
 10  )
 11  /

Table created.

Elapsed: 00:00:00.62
SYSADM-MHRSYT2A.1.475:28265.MHRSYT2A.oracle.thrvm>
SYSADM-MHRSYT2A.1.475:28265.MHRSYT2A.oracle.thrvm>--this procedure finds the columns
SYSADM-MHRSYT2A.1.475:28265.MHRSYT2A.oracle.thrvm>DECLARE
  2    l_column_name VARCHAR2(18);
  3  BEGIN
…
 59  END;
 60  /

PL/SQL procedure successfully completed.
```

gfc_desc_timestamp_index1.lst

```
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>set lines 80
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>ttitle 'Number of Tables / Indexes / Columns to convert
to Oracle Timestamp'
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>select count(distinct table_name) tables
  2  ,       count(distinct index_name) indexes
  3  ,       count(*) columns
  4  from gfc_desc_timestamp_index x
  5  /

Mon Dec 09                                                              page    1[31]
      Number of Tables / Indexes / Columns to convert to Oracle Timestamp

    TABLES    INDEXES    COLUMNS
---------- ---------- ----------
       122        138        138

1 row selected.

Elapsed: 00:00:00.51
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>ttitle 'Columns to be converted to Oracle Timestamp that
appear in more than one index'
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>select table_name, column_name, count(*) indexes
  2  from gfc_desc_timestamp_index x
  3  group by table_name, column_name
  4  having count(*)>1
  5  /

Mon Dec 09                                                              page    1[32]
 Columns to be converted to Oracle Timestamp that appear in more than one index

TABLE_NAME         COLUMN_NAME           INDEXES
------------------ ------------------ ----------
PS_ITEM_LINE_CHGDT DTTM_STAMP                  3
PS_HR_NP_NOTE      HR_NP_CREATE_DTTM           2
PS_R_FANLTR_STDNT  DTTM_STAMP                  6
PS_ORG_COMM        COMM_DTTM                   5
PS_QUICK_POST_TBL  GRP_TIMESTAMP               3
PS_ORG_COMMENT     COMMENT_DTTM                3

6 rows selected.

Elapsed: 00:00:00.59
```

[31] You can see how few table and indexes are affected by this problem, so there is a significant saving in time with this approach.

[32] Some columns appear in more than one index. This is why I drop all the affected indexes for a table before altering the columns.

```
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>ttitle 'Indexes that have more than one column to convert
to Oracle Timestamp'[33]
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>select table_name, index_name, count(*) columns
  2  from gfc_desc_timestamp_index x
  3  group by table_name, index_name
  4  having count(*)>1
  5  /

no rows selected

Elapsed: 00:00:00.48
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>set lines 131
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>ttitle 'Summary of Index Rebuilds'
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>select *
  2  from gfc_desc_timestamp_index
  3  order by table_name, index_name
  4  /

Mon Dec 09                                                                                                    page    1[34]
                                            Summary of Index Rebuilds

                                              Col
RECNAME         TABLE_NAME         INDEX_NAME          Pos COLUMN_NAME        COLUMN_EXPRESSION
--------------- ------------------ ------------------ ---- ------------------ --------------------
INDEX_DDL
--------------------------------------------------------------------------------------------------------------------------------
…
HR_NP_NOTE      PS_HR_NP_NOTE      PS0HR_NP_NOTE         8 HR_NP_CREATE_DTTM  "HR_NP_CREATE_DTTM"

  CREATE INDEX "SYSADM"."PS0HR_NP_NOTE" ON "SYSADM"."PS_HR_NP_NOTE" ("HR_NP_SUBJECT", "OBJECTOWNERID", "HR_NP_SUB_ID", "HR_NP_NOTE_
KEY1", "HR_NP_NOTE_KEY2", "HR_NP_NOTE_KEY3", "HR_NP_NOTE_KEY4", "HR_NP_CREATE_DTTM" DESC)
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "PSINDEX"
  NOLOGGING PARALLEL

HR_NP_NOTE      PS_HR_NP_NOTE      PS_HR_NP_NOTE         7 HR_NP_CREATE_DTTM  "HR_NP_CREATE_DTTM"

  CREATE UNIQUE INDEX "SYSADM"."PS_HR_NP_NOTE" ON "SYSADM"."PS_HR_NP_NOTE" ("OBJECTOWNERID", "HR_NP_SUB_ID", "HR_NP_NOTE_KEY1", "HR
_NP_NOTE_KEY2", "HR_NP_NOTE_KEY3", "HR_NP_NOTE_KEY4", "HR_NP_CREATE_DTTM" DESC)
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "PSINDEX"
  NOLOGGING PARALLEL
```

[33] It is possible for an index to contain more than one date column to be converted to a timestamp, but I haven't encountered that.

[34] Finally, this report simply lists each of the indexes to be rebuilt.

```
……
138 rows selected.

Elapsed: 00:01:44.40
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>ttitle off
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>set serveroutput on
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>spool gfc_desc_timestamp_index2
```

gfc_desc_timestamp_index2.lst

This file reports on the execution of the second PL/SQL block that does all the work. All the DDL statements are listed.

```
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>DECLARE
  2    l_sql CLOB;
…
 53  END;
 54  /


…
[24[35]]DROP INDEX PS0HR_NP_NOTE
[24]DROP INDEX PS_HR_NP_NOTE
[62]ALTER TABLE PS_HR_NP_NOTE MODIFY (HR_NP_CREATE_DTTM TIMESTAMP)
[463]
  CREATE INDEX "SYSADM"."PS0HR_NP_NOTE" ON "SYSADM"."PS_HR_NP_NOTE" ("HR_NP_SUBJECT", "OBJECTOWNERID", "HR_NP_SUB_ID",
"HR_NP_NOTE_KEY1", "HR_NP_NOTE_KEY2", "HR_NP_NOTE_KEY3", "HR_NP_NOTE_KEY4", "HR_NP_CREATE_DTTM" DESC)
  PCTFREE 10 INITRANS 2
MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS
1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "PSINDEX"
  NOLOGGING PARALLEL
[44]ALTER INDEX PS0HR_NP_NOTE NOPARALLEL LOGGING
[453]
  CREATE UNIQUE INDEX "SYSADM"."PS_HR_NP_NOTE" ON "SYSADM"."PS_HR_NP_NOTE" ("OBJECTOWNERID", "HR_NP_SUB_ID",
"HR_NP_NOTE_KEY1", "HR_NP_NOTE_KEY2", "HR_NP_NOTE_KEY3", "HR_NP_NOTE_KEY4", "HR_NP_CREATE_DTTM" DESC)
  PCTFREE 10 INITRANS 2
MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS
1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "PSINDEX"
  NOLOGGING PARALLEL
[44]ALTER INDEX PS_HR_NP_NOTE NOPARALLEL LOGGING
…

PL/SQL procedure successfully completed.

Elapsed: 00:00:53.46
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>
SYSADM-MHRPRE1A.1.312:23947.MHRPRE1A.oracle.phpor>spool off
```

[35] The numbers inside the square brackets […] indicate the length of the DDL command in characters. HR_NP_NOTE is an example of where two indexes have to be dropped because they both reference HR_NP_CREATE_DTTM which is converted to a timestamp.