3rd ASSIGNMENT – Physical Design on Oracle® DB

The DBA is in charge, among other responsibilities, of monitoring and tuning the system. In this assignment, we are going to simulate this situation, although we will significantly simplify the processes. Specifically, we will focus on the design of the physical layer of the system.

We will start off the database that we have designed and used in the other two assignments. Specifically, we'll take the proposed solution for the first assignment (which was also the start point for second assignment). The first step is to set up the testing environment. To do this, everything from the second assignment that may hinder the evaluation (views, indexes, triggers, auxiliary tables... and testing data) will be eliminated. Thus, we will run the scripts that create and populate the DB ("createBD_2023.sql" and "insert2023 - errors_fixed.sql") and that were provided for the second assignment. Also, the definition of the "tracks" table will be modified, adding two new columns (*searchk* and *lyrics*). The PL/SQL code that creates these columns and provides the values to one of them must be executed once (after running the two mentioned scripts). This code is provided in the first attachment of this document (and it is also incorporated into the P3 script).

The first step that we should take is to analyze what operations occur in the database, focusing on the most relevant ones. Besides, not all of them have the same frequency and this has to be taken into account. Such analysis has already been carried out, concluding that the most important are:

- Process P_1: updating one row (selection by the attribute labelled *searchk*)
- Process P_2: 1st query (from the 2nd assignment).
- Process P_3: 2nd query (from the 2nd assignment).

The relative frequencies of these three processes are {0.98,0.01,0.01} (ie, for every time both queries are run, 98 update operations are executed). The PL/SQL code that implements these processes is provided in the annex, at the end of this document. With that code, a workload has been created that includes the operations to be run and in the proper proportion and order. This is especially relevant since the update processes affect the performance of the rest of the processes. Therefore, in this practical scenario, it is not allowed to alter those operations running order. This methodology (evaluation of a workload on a controlled environment) is not the working method for this purpose (tunning physical design), but it is instructional and adequate in this assignment.

Along with this statement, a script (script_statistics_2023.sql) is attached. When executed, it creates a package (PKG_COSTES) in the database. Within this package, there are three procedures that implement the mentioned workload, and make testing possible: the PR_PREPARE procedure initializes the variable that decides which update operations are going to be performed; the PR_WORKLOAD procedure that contains the workload itself; and the PR_RESET procedure that is responsible for returning the database to the initial state (since the execution of the workload alters the state of the database). It also contains an auxiliary procedure for time control, and a procedure (RUN_TEST) that is responsible for repeating the workload one or more times (indicated by the *ite* parameter) and then displays the average cost (in both accesses and response time).

This procedure (RUN_TEST) is the only one that the students have to call to carry out the measurements (it calculates and shows the average time and the number of accesses of the n repetitions of the workload). Optionally, students can modify the script with one of these aims:

- Modify the PR_WORKLOAD procedure to add execution guidelines (hints)
- Modify PR_RESET procedure to initialize the state of any structure of your physical design that could be affected by the WL update operations (that is, that affects the table *tracks*).
- Modify the experiment, eliminating the initialization of the structures to study another scenario.

Therefore, the next step is to study the performance of the standard workload to see the initial costs. To do this, run the mentioned package creation (this is required only once), and call the run_test procedure. Keep in mind that the number repetitions of the workload should not be less than 5 to have a minimally reliable estimation (although between 10 and 20 iterations are recommended, no more than 10 iterations are necessary).

Once we know the efficiency of the initial design, the next step is to analyze current state and its possibilities. Specifically, we will analyze (a) the initial physical design; (b) the nature of the operations to be executed; (c) the execution plan of each operation (algorithms used in its resolution), providing details of its execution; and (d) the average costs calculated over several runs.

Based on this study, we will develop our physical design proposal aimed at reducing the number of accesses to secondary storage, and the global running time. Physical design proposals may entail changes that require rebuilding part of the system. With already existing systems, structural changes are difficult to implement, because they usually involve altering structures in production that already contain data. Conversely, newly created systems and testing environments ease the application of structural changes. In this assignment, almost any structure studied in the subject is allowed (if it is within the possibilities of the DBMS, of course). You can change physical parameters of the objects (pctfree, tablespace, etc.), create structures (single-table or multi-table clusters, indexes, etc.), add execution directives (hints), ... However, materialized views won't be allowed. It is mandatory to consider (at least) the inclusion of one index and one cluster. Each element's design, implementation and evaluation have to be included in the assignment report; nevertheless, if when analyzing its performance it is concluded that it is not advantageous, you can remove it from the final design (keeping its design and performance measurements in the report).

The physical design should be implemented along with the original structures (that is, modifying the original *createBD_2023.sql* script). When running it, everything will be destroyed and created again with the new physical design, although empty of course. After executing it, the database must be populated again by running "insert2023 - errors_fixed.sql". It is also necessary to create the two new columns into the "tracks" table (this code can be added to the end of the createBD script for convenience). In case the WL is modified, it will also be necessary to create the package PKG_COSTES again, by running the (modified) script.

The last step would be to verify that the improvements have had the desired effect. The results of the execution of the workload on the initial physical design must be compared with the results obtained on the complete physical design proposed and implemented. Optionally, the analysis can be extended by performing a new physical design and evaluating it: usually, the process of monitoring and tuning the system is a process of continuous refinement, although in this practice this is not necessary (it's only intended to improve the starting off situation in some way, and to provide a good analysis). You can (optionally) modify the experiment to evaluate the performance of successive executions without the restart operation (of the DB state) being performed.

Summary of steps to take:

- **Starting situation**: restore the original state of the DB
- **Measure** the performance of the standard workload (both time and number of accesses) executing several times.
- **Analyze** the execution plan for each operation, extracting the processes (typology) of which it is composed. Classify and sort all the processes in the physical structure, their frequency and an approximation of the cost.
- Based on that analysis, develop a proposal to reduce the number of accesses to secondary memory.
- Describe the **complete physical design**, starting from the basic organizations involved in it, as well as the auxiliary structures used to improve the performance.
- **Implement** the proposed complete physical design, and measure its performance over the proposed standard workload.
- **Compare** the performance of the original physical design versus the proposed one and interpret the results. Optionally, you can refine the proposal and implement improvements to the physical design (whose performance you should analyze and compare with the previous ones).
- **Document** all the work done in this assignment, paying special attention to physical design, performance measurements, execution plans, improvement proposals, and comparison of results.

Documentation to deliver:

A report must be submitted containing all the steps described above. In addition, it will contain the code for implementing the complete physical design that you have made (not all the code; only the modifications included into the DB creation script, or into the package).

Supporting Materials:

- Slides corresponding to the lab session.
- SQL Scripts (Oracle PL/SQL syntax): standard workload and procedures related to performance measurements based on querying the statistics collected and stored by Oracle.
- Oracle DBMS 12c user account.

ANNEX: Supporting Code

Structural Modifications: (creation of two columns into 'tracks', and valuation of one of them).

```
ALTER TABLE tracks add (searchk varchar2(20), lyrics VARCHAR2(4000));
UPDATE tracks set searchk=pair||'//'||sequ;
COMMIT;
```

Code for preparing the WL:

```
select searchk bulk collect into changes
   from (select searchk from tracks order by dbms_random.value)
   where rownum<=98;
```

Workload – Updating one row:

```
UPDATE tracks
   set lyrics = dbms_random.string('a',dbms_random.value(900,1200))
   where searchk=changes(i);
```

Workload – Query 1:

```
FOR fila in (
WITH authors as (select title,writer, writer musician from songs
                 UNION select title,writer,cowriter musician from songs),
     authorship as (select distinct band performer, title, writer, 1 flag
                       FROM involvement join authors using(musician) ),
     recordings as (select performer,tracks.title,writer
                       from albums join tracks using(pair)),
     recs_match as (select performer,
                           round(sum(flag)*100/count('c'),2) pct_recs
                       from recordings left join authorship
                            using(performer,title,writer)
                       group by performer),
     pers_match as (select performer,
                            round(sum(flag)*100/count('c'),2) pct_pers
                       from (select performer, songtitle title,
                                    songwriter writer
                             from performances) P
                                  left join authorship
                                  using(performer,title,writer)
                       group by performer)
SELECT performer, pct_recs, pct_pers
   from recs_match full join pers_match using(performer)
) LOOP null; END LOOP;
```

Workload – Query 2:

```
FOR fila in (
WITH recordings as (select performer,tracks.title, writer,
                           min(rec_date) rec, 1 token
                    from albums join tracks using(pair)
                    group by performer,tracks.title,writer),
     playbacks as (select P.performer,
                       sum(token)*100/count('x') percentage,
                       avg(nvl2(rec,when-rec,rec)) as age
                    FROM performances P left join recordings R
                         on(P.performer=R.performer
                            AND R.title=P.songtitle
                            AND R.writer=P.songwriter
                            AND P.when>R.rec)
                    GROUP BY P.performer
                    ORDER BY percentage desc)
SELECT performer, percentage, floor(age/365.2422) years,
       floor(mod(age,365.2422)/30.43685) months,
       floor(mod(age,365.2422)-
(floor(mod(age,365.2422)/30.43685)*30.43685)) days
   FROM playbacks WHERE rownum<=10
) LOOP null; END LOOP;
```

Code for resetting the DB initial state:

```
-- DML operations for resetting initial state
-- physically remove the table's contents, and inserts again

execute immediate 'TRUNCATE TABLE tracks';

INSERT INTO TRACKS (PAIR, sequ, title, writer, duration,
                    rec_date, studio, engineer, searchk)
  (SELECT DISTINCT album_pair, tracknum, min(song), min(writer),
                   min(duration), min(rec_date), min(studio),
                   min(engineer), album_pair||'//'||tracknum
     FROM fsdb.recordings
     WHERE album_pair is not null and tracknum is not null
           and song is not null and writer is not null
           and duration is not null and rec_date is not null
           and engineer is not null
           and not (song='Something jazzes' and writer='GB>>0785936179')
     GROUP BY album_pair,tracknum having count('s')=1
    );
COMMIT;
```