SCRUTINIZING COLUMN-ORIENTED DATABASES

OBJECTIVES

This is time to get familiar with the internals of column-oriented databases.

A column-oriented database uses vertical fragmentation to store data. Thus, opposite to row-oriented databases (i.e., traditional relational databases) that store data by rows, they store it by columns. As consequence, in front of random queries (e.g., those of a Data Warehouse) it tends to yield a much better effective read ratio.

In this lab we will scrutinize the differences between row-oriented and column-oriented databases. Although row-oriented database vendors claim they can achieve similar performance with the correct tuning, this turns not to be true.

In this lab you are aimed at implementing the different ways a row-oriented relational database can be tuned for read-only queries. Therefore, you will be asked to tune Oracle 11g in three different ways: using index-only query answering, materialized views and nested tables (to simulates vertical fragmentation) and discuss the query execution carried out by Oracle in each case. You are then asked to compare that execution plan with the internal structures and query processing a column-oriented database does and realize there are some optimizations that a row-oriented will never be able to do.

The list of objectives for this session is as follows:

- Given a read-only workload, decide what of the three main tunings for relational databases give a better result:
    - Index-only query answering techniques (with both B+ and bitmaps),
    - Materialized views,
    - Using UDTs and nested tables to simulate vertical fragmentation.
- Understand the execution plan proposed by Oracle for each case.
- Then, realise about the main drawbacks in these execution plans. Considering how column-oriented databases store and process data, discuss each execution plan with regard to what a column-oriented database would do and identify the differences.

REQUIRED KNOWLEDGE

A script to delete any kind of object, the description of the operations in Oracle's explanation plan and some hints are also provided as additional material.
Your lab mates for this session will be that of the team creation 3 event.

TOOLS

These exercises will be done in Oracle 11g and Learn-SQL (a document to upload explaining your remote quiz solution).

DELIVERABLES

The main task for this lab is to solve the Learn-SQL *3rd Lab: Scrutinizing Column-Oriented Databases* (remote quiz). To do so, you must work on your Oracle 11g account, create your solution and then provide your login and password in the Learn-SQL remote quiz to enable the automatic assessment.

Then, afterwards, you must upload a document to Learn-SQL to the *3rd Lab Document Submission* (assignment) answering the questions you will find in the *questionnaire* document.

Be aware that this lab main objective is to think about how a column-oriented database would outperform Oracle (i.e., a row-oriented database). Thus, pay attention to deliver a proper document answering the questionnaire and showing understanding in the matter.

OPTIONAL ACTIVITIES

Repeat these exercises in a column-oriented database:

- Go to http://www.monetdb.org/Home and download MonetDB, an open-source column-oriented database.
- Implement the schema proposed in the training and launch the queries.
- Ask MonetDB to let you know the execution plan. This is a real column-oriented database! You will see the differences with Oracle.

APPENDIX A

A detailed step by step guide to run your solution to the remote quiz in Oracle. This appendix explains how to create your solution when only creating views and indexes (i.e., exercise 1 of the Learn-SQL remote quiz)

Each exercise in the Learn-SQL remote quiz has an attachment. These attachments follow the same structure here explained. Be sure you understand what each part does to smoothly progress in this lab. Importantly, **you MUST run these parts in order** (as explained below and written in the attachment) to allow Learn-SQL properly assess your solution. Be sure to follow these steps each time you create your solution in Oracle:

- Database schema (of the exercise): It contains a set of CREATE tables. Remember tables must be created with PCTFREE 0 and ENABLE ROW MOVEMENT as explained later in this appendix.

- Inserts: Data you must insert in the tables. Just execute this script. Note that, at the end, you must execute "`ALTER TABLE` *`table_name`* `SHRINK SPACE`". This guarantees the table is compacted.

- Next, you are supposed to insert your solution (i.e., indexes and / or materialized views). **See the next section *creating indexes and views* for more details**.

- The next code, when executed, updates the catalog statistics (needed to keep the database catalog up to date).

- Finally, execute the last part, creating the measure table, to know the average cost for your solution. Note that each query is properly weighted according to the weights given in the statement. Some insights about this code:

    - The query cost considered to assess your solution is that provided by Oracle in the execution (measured in terms of gets from the cache) for each query (weighted by the query weight).

    - The used space value you must honor, if there is a space constraint, is the one corresponding to USER_TS_QUOTAS. After creating your structures and inserting the needed data, trigger this query to check its value:

      ```
      SELECT SUM(BLOCKS) FROM USER_TS_QUOTAS
      ```

CREATING INDEXES AND VIEWS

These constraints are needed to let Learn-SQL properly assess your solution. These are rules about how to create the structures, **which you must create in the following way and no other**:

- Table without clustered index, nor clustered structure, nor hash:

  ```
  CREATE TABLE name (attributes) PCTFREE 0 ENABLE ROW MOVEMENT;

  Insertions

  ALTER TABLE name SHRINK SPACE; // Compress the table
  ```

- B+ tree (after the insertions):

  ```
  CREATE TABLE name (attributes) PCTFREE 0 ENABLE ROW MOVEMENT;

  Insertions

  ALTER TABLE name SHRINK SPACE;  // Compress the table
  CREATE INDEX name ON table (attributes) PCTFREE 33;
  ```

  - We create the index after the insertions to assure that it holds the desired load factor (normally, 2/3).

- Bitmap[1]:

  ```
  CREATE TABLE name (attributes) PCTFREE 0 ENABLE ROW MOVEMENT;

  Insertions

  ALTER TABLE name SHRINK SPACE; //Compress the table

  ALTER TABLE table MINIMIZE RECORDS_PER_BLOCK; //Compress the bitmap

  CREATE BITMAP INDEX name ON table(attributes) PCTFREE 0;
  ```

Bitmaps should only be considered for attributes with values repeated –over 100 times. To know how many times is repeated each value of a bitmap compute it as follows: #rows in the table / # distinct keys in the index (check the catalog, USER_TABLES and USER_INDEXES to find these values).

- Materialized views:

  ```
  CREATE MATERIALIZED VIEW view_name ORGANIZATION HEAP PCTFREE 0
  BUILD IMMEDIATE REFRESH COMPLETE ON DEMAND ENABLE QUERY REWRITE AS
  ( query );
  ```

  - **Do not consider FAST ON COMMIT views.**

[1] Due to their compression techniques, bitmaps generate little unforeseen variations in query costs (usually one or two blocks). This potential deviation will never result in more than one point in the exercise, and will be manually (off-line) corrected by the proctor.

General rules if you want to go for any other solution (although the maximum mark can be obtained just considering the above elements):

- “ROW MOVEMENT” must be enabled for any table not in a cluster. It will allow you to “SHRINK SPACE” just after the insertions.
- “MINIMIZE RECORDS_PER_BLOCK” must be executed after the insertions of the table and before the creation of any bitmap index on it (you only need to execute it once, even if you create many bitmap indexes over the same table).

ADDITIONAL INFORMATION

- You might be interested in the following catalog tables. They will let you know what is happening and give the actual values for space and cost (inside parenthesis you have some attributes that may be interesting):
    - `USER_TABLES (TABLE_NAME, CLUSTER_NAME, IOT_TYPE, IOT_NAME, PCT_FREE, BLOCKS, NUM_ROWS, AVG_ROW_LEN, LAST_ANALYZED)`
    - `USER_TAB_COLS (TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGTH, AVG_COL_LEN, NULLABLE, LAST_ANALYZED)`
    - `USER_INDEXES (INDEX_NAME, TABLE_NAME, INDEX_TYPE, UNIQUENESS, PCT_FREE, BLEVEL, LEAF_BLOCKS, DISTINCT_KEYS, LAST_ANALYZED, JOIN_INDEX)`
    - `USER_SEGMENTS (SEGMENT_NAME, SEGMENT_TYPE, BYTES, BLOCKS)`
    - `USER_TS_QUOTAS (TABLESPACE_NAME, BYTES, BLOCKS)`
    - `RECYCLEBIN (ORIGINAL_NAME)`

APPENDIX B

A detailed step by step guide to run your solution to the remote quiz in Oracle. This appendix explains how to create vertical fragmentations in Oracle via nested tables (i.e., exercise 2 of the Learn-SQL remote quiz)

Vertical fragmentation is not directly supported by Oracle. Instead, it can be simulated using object-relational features, which changes a bit how to proceed. To avoid getting lost, follow these steps to modify the attached file and build your solution:

- Decide how to vertically fragment the database. You may want to use the affinity matrix method we saw in the lectures. For each vertical fragment you must create a nested table. Here, you need to assess whether a pure vertical partitioning (i.e., each column is stored separately in a nested table) or a hybrid approach (i.e., group some columns in each nested table).

- If you used the affinity matrix method, be careful; **Oracle's best solution could not be the one theoretically identified**. Can you understand why if so? *Hint: Oracle nested tables are, in the end, row-oriented structures and therefore, they simulate vertical fragmentation but are not real vertical fragments. Thus, there is extra costs hidden when using them (check Oracle's execution plan to realise) that may spoil the result.*

- Replace the create table with the corresponding CREATEs for the collection types and the table containing the nested tables. As before, any table must be created with PCTFREE 0 and ENABLE ROW MOVEMENT (this does not apply to nested tables).

- Modify the INSERT script to adapt it to your new object-relational schema (basically, you need to insert data into nested tables now and thus you may want to refresh the previous session on object-relational).

- Finally, to adapt the last part, you need to create three views view1, view2 and view3. Each view must adapt each of the queries in the exercise statement to your new object-relational schema. For example, consider Q1:

  ```
  SELECT cand AS a, AVG(val) AS b FROM poll_answers
  GROUP BY cand);
  ```

  You must create a view view1 adapting this query to your nested tables and retrieving the same data. For example, assume *cand* and *val* are in two different nested tables p1 and p2 then (how to query nested tables is, again, material from the previous lab):

  ```
  CREATE VIEW view1 AS
  SELECT p1.cand AS a, AVG(p2.val) AS b
  FROM poll_answer pa, TABLE(pa.part1) p1, TABLE(pa.part2) p2
  GROUP BY p1.cand
  ```

IMPORTANT: Each attribute in the view select must have an alias, and these aliases must coincide with the alphabet sorted lexicographically. In other words, the first attribute must have a as alias (see the example), the second one b, the third c, and so on. Failing to do so, your exercise will not be correctly assessed by Learn-SQL.

All the rest remain the same (e.g., you may want to create indexes in the nested tables as shown in the previous appendix) and you can still ask Oracle for the execution plan of a query. Remember to update the statistics before sending your answer.