SESSION ON ACCESS STRUCTURES

OBJECTIVES

- Decide for each structure (i.e., B+, Hash, Clustered index and Clustered Structure) and operation (i.e., table scan, select one tuple, select several tuples and join), whether it is generally worth to use that structure to perform the operation and how it would be used.
- Decide the indexes to be defined according to the expected operations.
- Explain the difference between the theoretical structures explained and the Oracle implementation.
- Create structures in Oracle, use the catalog and see the execution plan.

ACTIVITIES

Given a simple situation (one table and one query, or two tables and a join) optionally constrained with respect to space:

- Propose suitable configurations (structure type and involved attributes),
- Estimate space and cost corresponding to several configurations, and
- Compare estimations and real values taken from Oracle and explain observed differences.

DELIVERABLES

- LearnSQL:
  - Students must create the structure that gives the lowest cost as reported by the Oracle Explain Plan utility conforming to the space constraint, if any.
    - In the solution you must provide in here **just introduce your user name and password to access your Oracle account**.
- Paper:
  - Students will be asked about relevant differences between theoretical results and Oracle results and about the causes of such differences.

CONSTRAINTS

- The used space value that is taken into account if there is a space constraint is the one corresponding to USER_TS_QUOTAS.
- Index-only query answering is not considered.

- Data structures can only be created in the following way:
  - Table without clustered index, nor clustered structure, nor hash: CREATE TABLE name (attributes) PCTFREE 0;

- Table with clustered index:
  CREATE TABLE name (attributes, PRIMARY KEY(…))
  ORGANIZATION INDEX PCTFREE 33;
  Insertions
  ALTER TABLE name MOVE;
  This way, we reconstruct the index and we assure that it holds the desired load factor (2/3).
  As it can be deduced, **we can only define a clustered index over the primary key of the table.**
- B+ tree (after the insertions):
  CREATE TABLE name (attributes) PCTFREE 0;
  Insertions
  CREATE INDEX name ON table (attributes) PCTFREE 33;
  We create the index after the insertions to assure that it holds the desired load factor (2/3).
- Table with hash:
  CREATE CLUSTER name (attributes type) SINGLE TABLE HASHKEYS 1'25*B PCTFREE 0;
  CREATE TABLE name (attributes) CLUSTER name(attributes);
- Clustered structure:
  CREATE CLUSTER cluster_name (a1 type) PCTFREE 33;
  CREATE INDEX index_name ON CLUSTER cluster_name PCTFREE 33;
  CREATE TABLE table_name1 (...,a2 type,...) CLUSTER cluster_name(a2);
  CREATE TABLE table_name2 (...,a3 type,...) CLUSTER cluster_name(a3);
  Insertions
  ALTER INDEX index_name REBUILD;
  This way, we reconstruct the index and we assure that it holds the desired load factor (2/3).

REQUIRED KNOWLEDGE

- Taught material: access structures (see slides section in Moodle).
- Additional material: correspondence theory-practice (see this session's section in Moodle).

ADDITIONAL INFORMATION

- The following catalog tables 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, 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_CLUSTERS (CLUSTER_NAME, PCT_FREE, CLUSTER_TYPE, HASHKEYS, SINGLE_TABLE)
    - USER_SEGMENTS (SEGMENT_NAME, SEGMENT_TYPE, BYTES, BLOCKS)
    - USER_TS_QUOTAS (TABLESPACE_NAME, BYTES, BLOCKS)
    - RECYCLEBIN (ORIGINAL_NAME)

- Some interesting hints to know or remember:
    - Blocks are 8 Kbytes (either table blocks and B+ nodes)
    - Addresses use 32 bits
    - Variable *u* is the number of entries in a B+ leaf.
    - Variable *d* is half the number of entries that fit in a B+ leaf with a load factor of 1
    - Variable *h* is the B+ height after insertions (BLEVEL attribute in the catalog)
    - Variable *B* is the number of blocks of a non-indexed table