Databases II
2019-11-14

Exercise 1

Create PLAN_TABLE, generate execution plans into this table for different SELECT statements and present the tree structure of the execution plans (see expl.txt).

Exercise 2

Create your own copy from the following tables and answer the following query.
QUERY:
Give the name of the departments which have an employee with salary category 1.

NIKOVITS.EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)
NIKOVITS.DEPT(deptno, dname, loc)
NIKOVITS.SAL_CAT(category, lowest_sal, highest_sal)

The query could be:

```
SELECT DISTINCT dname
FROM emp e JOIN sal_cat sc ON sal BETWEEN lowest_sal AND highest_sal
NATURAL JOIN dept d
WHERE category = 1;
```

See the execution plan of the previous query, then create an index for any of the tables that can be used by the query.
Check the new execution plan, if the index is really used in it!

An index could be:

```
CREATE INDEX emp_sal_idx ON emp(sal);
```

Exercise 3

Compare the two similar queries in runtime_example.txt, compare the execution plans and tell what the difference is. Why one of them is much faster? See COST and CARDINALITY for the nodes.

The difference: The table in question is partitioned. Only the partition containing the relevant rows would be read by the DBMS. In the slower query there is an operation performed on the attribute to be checked therefore it was not obvious for the DBMS which partition to check. Therefore it checked all. Hence the added computational time.

Exercise 4

The owner of the following tables is NIKOVITS.

PRODUCT(prod_id, name, color, weight)
SUPPLIER(supl_id, name, status, address)
PROJECT(proj_id, name, address)
SUPPLY(supl_id, prod_id, proj_id, amount, sDate)

Suppliers transport different products to the projects.
You can see in table SUPPLY the supplier (supl_id), the product (prod_id), the project (proj_id), the date of the transport and the amount of the products transported on the specific date.

The tables were created with the following statements:

```
CREATE TABLE product(prod_id, name, color, weight) AS SELECT * FROM nikovits.cikk;
CREATE TABLE supplier(supl_id, name, status, address) AS SELECT * FROM nikovits.szallito;
CREATE TABLE project(proj_id, name, address) AS SELECT * FROM nikovits.projekt;
CREATE TABLE supply(supl_id, prod_id, proj_id, amount, sDate) AS SELECT * FROM
nikovits.szallit;
GRANT select on product to public;
GRANT select on supplier to public;
GRANT select on project to public;
GRANT select on supply to public;

-- The tables have indexes too.
CREATE INDEX prod_color_idx ON product(color);
CREATE UNIQUE INDEX prod_id_idx ON product(prod_id);
CREATE UNIQUE INDEX proj_id_idx ON PROJECT(proj_id);
CREATE UNIQUE INDEX supplier_id_idx ON supplier(supl_id);
CREATE INDEX supply_supplier_idx ON supply(supl_id);
CREATE INDEX supply_proj_idx ON supply(proj_id);
CREATE INDEX supply_prod_idx ON supply(prod_id);
```

QUERY

Give the sum amount of products where color = 'piros' ('piros' in Hungarian means 'red').

```
SELECT SUM(amount)
FROM supply NATURAL JOIN product
WHERE color='piros';
```

Give hints in order to use the following execution plans (see hints.txt):

a) no index at all

```
SELECT /*+ no_index(s) no_index(p)  */ SUM(amount)
FROM supply s NATURAL JOIN product p
WHERE color='piros';
```

b) one index

```
SELECT /*+ index(s) */ SUM(amount)
FROM supply s NATURAL JOIN product p
WHERE color='piros';
```

c) index for both tables

```
SELECT /*+ index(s) index (p)*/ SUM(amount)
FROM supply s NATURAL JOIN product p
WHERE color='piros';
```

d) SORT-MERGE join

```
SELECT /*+ use_merge(s p) */ SUM(amount)
FROM supply s NATURAL JOIN product p
WHERE color='piros';
```

e) NESTED-LOOPS join

```
SELECT /*+ use_nl(s p) */ SUM(amount)
FROM supply s NATURAL JOIN product p
WHERE color='piros';
```

f) NESTED-LOOPS join and no index

```
SELECT /*+ use_nl(s p) no_index(s) no_index(p) */ SUM(amount)
FROM supply s NATURAL JOIN product p
WHERE color='piros';
```