Databases II
2019-11-21

Execution plans, hints
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)

The tables have indexes too.

Exercise 1
Query:
Give the sum amount of products where prod_id=2 and supl_id=2.

Give hints in order to use the following execution plans:
a) No index

```
SELECT /*+ no_index(s)*/ sum(amount)
FROM supply s
WHERE prod_id=2 and supl_id=2;
```

b) Two indexes and the intersection of ROWID-s (AND-EQUAL in plan).

```
SELECT /*+ index(s) and_equal(s supply_prod_idx supply_supplier_idx) */ sum(amount)
FROM supply s
WHERE prod_id=2 and supl_id=2;
```

Exercise 2
Give a SELECT statement which has the following execution plan.

```
PLAN (OPERATION + OPTIONS + OBJECT_NAME)
---------------------------------------
SELECT STATEMENT +  +
  SORT + AGGREGATE +
    TABLE ACCESS + FULL + PRODUCT
```

```
select /*+ full(p) */ sum(weight)
from nikovits.product p where color='piros';
```

```
SELECT STATEMENT +  +
  SORT + AGGREGATE +
    TABLE ACCESS + BY INDEX ROWID + PRODUCT
      INDEX + UNIQUE SCAN + PROD_ID_IDX
```

```
select /*+ index(p) */ sum(weight)
from nikovits.product p where prod_id=1;
```

```
SELECT STATEMENT +  +
  SORT + AGGREGATE +
    HASH JOIN +  +
      TABLE ACCESS + FULL + PROJECT
      TABLE ACCESS + FULL + SUPPLY
```

```
select /*+ full(p) */ sum(amount)
from nikovits.supply s natural join nikovits.project p
where address='Szeged';
```

```
SELECT STATEMENT +  +
  HASH + GROUP BY +
    HASH JOIN +  +
      TABLE ACCESS + FULL + PROJECT
      TABLE ACCESS + FULL + SUPPLY
```

```
select /*+ full(p) */ sum(amount)
from nikovits.supply s natural join nikovits.project p
where address='Szeged' group by prod_id;
```

```
SELECT STATEMENT +  +
  SORT + AGGREGATE +
    MERGE JOIN +  +
      SORT + JOIN +
        TABLE ACCESS + BY INDEX ROWID BATCHED + PRODUCT
          INDEX + RANGE SCAN + PROD_COLOR_IDX
      SORT + JOIN +
        TABLE ACCESS + FULL + SUPPLY
```

```
select /*+ use_merge(s p) index(p) */ sum(amount)
from nikovits.supply s natural join nikovits.product p
where color='piros';


SELECT STATEMENT +  +
  FILTER +  +
    HASH + GROUP BY +
      HASH JOIN +  +
        TABLE ACCESS + FULL + PROJECT
        HASH JOIN +  +
          TABLE ACCESS + FULL + SUPPLIER
          TABLE ACCESS + FULL + SUPPLY

select /*+ no_index(s) leading(sr) */ sum(amount)
from nikovits.supply s, nikovits.supplier sr, nikovits.project p
where s.supl_id=sr.supl_id and s.proj_id=p.proj_id
and sr.address='Pecs' and p.address='Szeged'
group by prod_id having prod_id > 100;
```

Logging, recovery

Basic operations:
Input (x): system reads block containing x into memory
Output (x): system writes block containing x to disk
Read (x,t): read x into transaction's local variable t (input(x) if necessary)
Write (x,t): write value of t into x in memory (input(x) if necessary)
t:= ... give new value to local variable t

Rules of UNDO log:
1. write log entries to disk (Write Ahead Log) [<T, ...> ... + FLUSH LOG]
2. write modified data elements to disk [output(X)] (-> problem: too frequent output)
3. write COMMIT log entry to log file on disk [<T, commit> + FLUSH LOG]

Exercise 3
The following is a sequence of undo-log records written by two transactions T and U:
<start T>
<T, A, 10>
<start U>
<U, B, 20>
<T, C, 30>
<U, D, 40>
<T, A, 11>
<U, B, 21>
<COMMIT U>
<T, E, 50>
<COMMIT T>
Describe the action of the recovery manager, including changes to both disk and the log, if there is a crash and the last log record to appear on disk is:
(a) <START U>
<ABORT,U>, WRITE(A,10), OUTPUT(A), <ABORT,T>, FLUSH LOG
(b) <C0MMIT U>
WRITE(A,11), OUTPUT(A), WRITE(C,30), OUTPUT(C), WRITE(A,10) OUTPUT(A), <ABORT,T>, FLUSH LOG
(c) <T, E, 50>
WRITE(E,50), OUTPUT(E), WRITE(A,11), OUTPUT(A), WRITE(C,30), OUTPUT(C), WRITE(A,10) OUTPUT(A), <ABORT,T>, FLUSH LOG
(d) <C0MMIT T>
Do nothing

Rules of REDO log:
1. write log entries to disk (Write Ahead Log) [<T, ...> ... + FLUSH LOG]
2. write COMMIT log entry to log file on disk [<T, commit> + FLUSH LOG]
3. write modified data elements to disk [output(X)] (-> problem: too late output)
4. write END log entry to log file on disk [<T, end> + FLUSH LOG]

Exercise 4
Repeat Exercise 3 with redo logging.
<start T>
<T, A, 10>
<start U>
<U, B, 20>
<T, C, 30>
<U, D, 40>
<T, A, 11>
<U, B, 21>
<COMMIT U>
<T, E, 50>
<COMMIT T>
Describe the action of the recovery manager, including changes to both disk and the log, if there is a crash and the last log record to appear on disk is:
(a) <START U>
Do nothing
(b) <C0MMIT U>
WRITE(B,20), OUTPUT(B), WRITE(D,40) OUTPUT(D), WRITE(B,21) OUTPUT(B), <END,U>, FLUSH LOG
(c) <T, E, 50>
same as in b)
(d) <C0MMIT T>
WRITE(B,20), OUTPUT(B), WRITE(D,40) OUTPUT(D), WRITE(B,21) OUTPUT(B), <END,U>, FLUSH LOG, WRITE(A,10), OUTPUT(A), WRITE(C,30) OUTPUT(C), WRITE(A,11), OUTPUT(A), WRITE(E,50), OUTPUT(E), <END,T>, FLUSH LOG

Rules of UNDO/REDO log:
1. write log entries to disk (Write Ahead Log)
   <T, COMMIT> can be written before OUTPUT or after OUTPUT

Exercise 5

The following is a sequence of undo/redo-log records written by two transactions T and U:

<START T>;
<T, A, 10, 11>;
<START U>;
<U, B, 20, 21 >;
<T, C, 30, 31>;
<U, D, 40, 41>;
<C0MMIT U>;
<T, E, 50, 51>;
<C0MMIT T>.

Describe the action of the recovery manager, including changes to both disk and the log, if there is a crash and the last log record to appear on disk is:

(a) <START U>

undo steps for T and U

(b) <C0MMIT U>

undo steps for T and redo steps for U

(c) <T, E, 50, 51>

undo steps for T and redo steps for U

(d) <C0MMIT T >

redo steps for T and U