Re-Exam in INF-2700 – Solutions

Database Systems

2012-02-20

dummy page

1 (40%)

Below are some database tables for a task management application. The example data show that task `wash dishess` is of category `home` and is assigned to `Ole` and `Anna` as two sub-tasks, and that `Anna` finished her part of the `wash dishes` task on `2012-02-20`.

- `People`

Pid	Name
2	Karin
6	Tom
7	Ole
9	Anna

- `Tasks`

Tid	Title	Category	Size
101	change light bulb	home	1
102	wash dishes	home	8
103	clean desk	job	3
105	bake cake	home	10
108	dance	fun	12
110	walk dog	home	5

- `DoTask`

Pid	**Tid**	SubSize	Status	UpdatedOn
6	101	1	done	2012-02-18
7	102	3	todo	2012-02-19
9	102	5	done	2012-02-20
7	103	3	todo	2012-02-20
9	105	8	done	2012-02-20
2	108	12	done	2012-02-20

The *primary keys* of the tables are in **bold** font.

Foreign keys:

- `DoTask`
  - `Pid`: references `Pid` of `People`
  - `Tid`: references `Tid` of `Tasks`

  The person with `pid` has the responsibility for (maybe part of) the task with `tid`.

Write queries to find the required information.
Queries 1–5 must be formulated in *both relational algebra and SQL.*
Queries 6–10 need only be formulated in *SQL.*

Suggested solution

$\rightsquigarrow$ *SQL scripts tested with SQLite are listed at the end of this document.*

1. Names of all people.

   The result for the example database is:

Name
Karin
Tom
Ole
Anna

$\Pi_{Name}(People)$

2. Tasks (titles and sub-sizes) assigned to `Ole`.

   The result for the example database is:

Title	SubSize
wash dishes	3
clean desk	3

$\Pi_{Title,SubSize}(\sigma_{Name='Ole'}People \bowtie DoTask \bowtie Tasks)$

3. Names of people who share the `wash dishes` task.

   The result for the example database is:

Name
Ole
Anna

$\Pi_{Name}(People \bowtie DoTask \bowtie \sigma_{Title='wash\ dishes'}Tasks)$

4. Titles of tasks that have not been assigned to anybody.

   The result for the example database is:

Title
walk dog

$\Pi_{Title}(Tasks \bowtie (\Pi_{Tid}Tasks - \Pi_{Tid}DoTask))$

5. Names of people who do not share a category of tasks with any other people.

   The result for the example database is:

Name
Karin

   $pc_1 \leftarrow \Pi_{Pid,Category}(People \bowtie DoTask \bowtie Tasks)$
   $pc_2 \leftarrow \Pi_{Pid,Category}(People \bowtie DoTask \bowtie Tasks)$
   $p \leftarrow \Pi_{pc_1.Pid}(pc_1 \bowtie_{pc_1.Pid \neq pc_2.Pid \wedge pc_1.Category = pc_2.Category} pc_2)$
   $\Pi_{Name}(Peopel \bowtie (\Pi_{Pid}People - p))$

6. List of Ole's `todo` tasks (together with sub-sizes and update date) in ascending order of updated date.

   The result for the example database is:

Title	SubSize	UpdatedOn
wash dishes	3	2012-02-12
clean desk	3	2012-02-20

7. Total sub-sizes finished on `2012-02-20`.

   The result for the example database is:

FinishedSize
25

8. Tasks not completely done (sum of finished sub-sizes, if any, is less than the size of the task).

   The result for the example database is:

Title
wash dishes
clean desk
bake cake
walk dog

9. Tasks assigned to multiple people.

   The result for the example database is:

Title
wash dishes

10. Date(s) with the most amount of finished work.

    The result for the example database is:

date	FinishedSize
2012-02-20	25

2 (20%)

We now consider the physical design of the table files. Assume that the files are organized as the following:

- `People` file uses *hash file organization* with hashing on attribute `Pid`.
- `Tasks` file uses *hash file organization* with hashing on attribute `Tid`.
- `DoTask` file uses *hash file organization* with hashing on attribute `Pid` *and* has a *hash index* on attribute `Tid`.

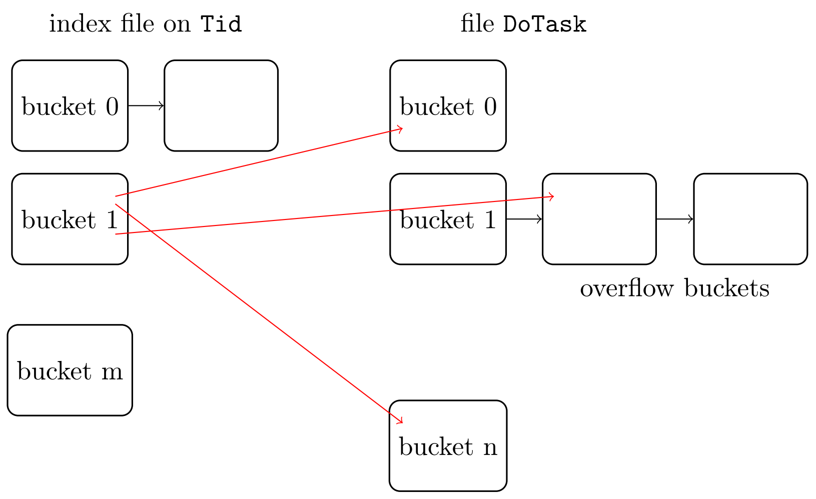
Assume the following about the data sizes:

- The sizes of both disk blocks and buffer pages are 4 kilo bytes.
- There are 100 buffer pages allocated to your algorithms.
- The sizes of data records of all three tables are 100 bytes.
- There 10000 people, each being assigned 400 sub-tasks.
- Every task is assigned to 4 people as 4 sub-tasks.
- You may make further assumptions.

Answer the following questions. To keep your answers focused, consider only static hashing.

Suggested solution

1. How are DoTask data organized on disk? Please draw a figure to illustrate the data organization.

   The DoTask file consists of a number of *buckets*. In the same bucket, all records have the same hash value of the Pid field.

   A bucket is typically a disk block, but can also be smaller or larger than a block

   If a bucket is not large enough for all records with the same hash value on the Pid field, there is a chain of *overflow buckets* for these records.

   The index on Tid is a secondary index and the index entries are stored in a hash organized file, which consists of buckets similar to the organization of the DoTask file. The records of the file are index entries (for all DoTask records) of the form $<$ Tid value, address $>$ where address is the address of the disk block containing the record as well as the offset of the record in the block.

2. Sketch an algorithm to make a natural join of the People and DoTask tables.

   The join is a loop that terminates when there is no more bucket from the files and no more record in input buffer pages:

   - read a number of People and DoTask buckets with the same Pid hash values in input buffer pages
   - compare the records from the buckets of the two files and write the join records into the output buffet pages
   - when an output page is full, write to disk

3. What is the performance overhead of your algorithm for the above question?

   The performance overhead is measured as the *number of disk seeks* and the *number of disk block transfers*.

   The calculations are only used to verify that students understand what performance means and how their algorithms work. They are not meant to be precise.

   Assume a bucket consists of one disk block, all buckets are half full and there is no overflow bucket. `People` file takes 500 buckets. `DoTask` file takes 200,000 buckets. The join result takes $(500/2) \times (200000/2) = 25,000,000$ (full) disk blocks.

   The files are read in once and the resulte written out once. Totally there are **25,200,500** disk block transfers.

   Assume we allocate 33 buffer pages for each input file and the output, there are around 25,200,500 / 33 = **760,000** disk seeks.

4. Sketch an algorithm to make a natural join of the `Tasks` and `DoTask` tables.

   - read the `Tasks` records in input buffer pages
   - for each `Tasks` record, use the hash index to read the `DoTask` records with the same `Tid` value
   - write the join records into the output buffet pages
   - when an output page is full, write to disk

5. What is the performance overhead of your algorithm for the above question?

   Assume we allocate 50 buffer pages for reading the `Tasks` buckets and 50 pages for the join results.

   There are 1,000,000 `Tasks` records in 50,000 buckets. That is, 50,000 block transfers and 1000 seeks.

   Assume every `DoTask` is obtained with 2 disk reads (one for the index entry and one for the record). There are 4,000,000 records. Therefore 8,000,000 block transfers and seeks.

   The join result takes 2,500,000,000 (full) disk blocks. That is, 2,500,000,000 block transfers and 50,000,000 seeks.

   Summing these up: **2,508,050,000** block transfers and **58,001,000** seeks.

3 (20%)

Answer the following questions. Try to give formal definitions of the corresponding concepts.

1. What is *functional dependency* `P` $\rightarrow$ `Q` of a relation instance r?

A	B	C
1	x	t
1	y	t
2	z	u

   Given the above relation instance, check if the following functional dependencies are satisfied:

   - `A` $\rightarrow$ `B`
   - `B` $\rightarrow$ `C`
   - `A` $\rightarrow$ `C`
   - `AC` $\rightarrow$ `B`
   - `AB` $\rightarrow$ `C`

   Suggested solution

   Functional dependency For any pair of tuples in r, if they have the same value in `P`, they also have the same value in `Q`.

   NO `A` $\rightarrow$ `B`, `AC` $\rightarrow$ `B`,

   YES `B` $\rightarrow$ `C`, `A` $\rightarrow$ `C`, `AB` $\rightarrow$ `C`

2. What is a *schema decomposition*? What is the purpose of making a schema decomposition?

   Suggested solution

   For relation schema (R, F), the set of schemas $(R_1, F_1)(R_2, F_2), \ldots, (R_n, F_n)$ is a decomposition if, $R = R_1 \cup R_2 \cup \ldots \cup R_n$ (no new attributes) and F logically implies F_i (no new dependencies).

   Redundancies can be removed when a schema is decomposed into some normal form.

3. What is a *lossless* schema decomposition?

   Suggested solution

   A decomposition of R into $R_1, R_2, \ldots, R_n$ is *lossless* if for every instance r, $r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \ldots \bowtie \Pi_{R_n}(r)$

4. What is a schema decomposition that *preserves functional dependencies*?

   Suggested solution

   A decomposition is *dependency preserving* if $F_1 \cup F_2 \cup \ldots \cup F_n$ also logically implies F.

4 (20%)

1. What is an *ACID transaction*?

   Suggested solution

   A transaction is a group of operations on shared (database) data.

   Atomicity all or nothing

   Consistency database is kept consistent and individual transactions are consistent

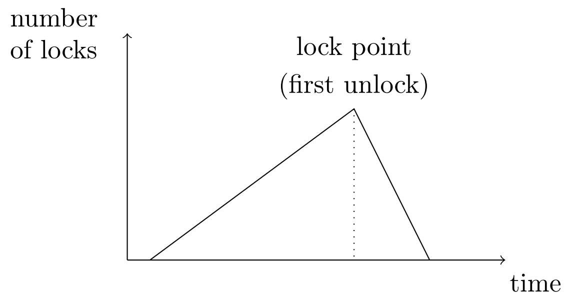
   Isolation interleaved executions of concurrent transactions have the same effect as isolated (serial) executions.

   Durability If a transaction commits, the result is not affected by possible subsequent events.

2. Describe the *two-phase locking* protocol (*2PL*), *strict two-phase locking* protocol (*S2PL*), and *rigorous two-phase locking* protocol (*R2PL*).

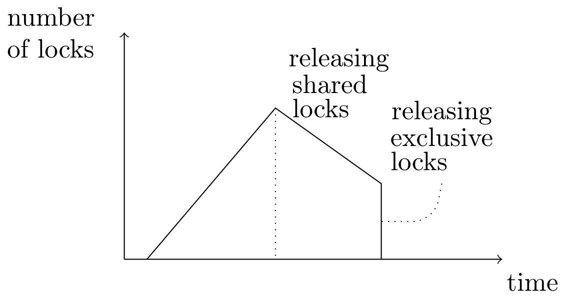
   Suggested solution

   - 2PL
     - a transaction has a *lock* phase (*growing* phase) followed by an *unlock* phase (*shrinking* phase)
     - guarantees serializability

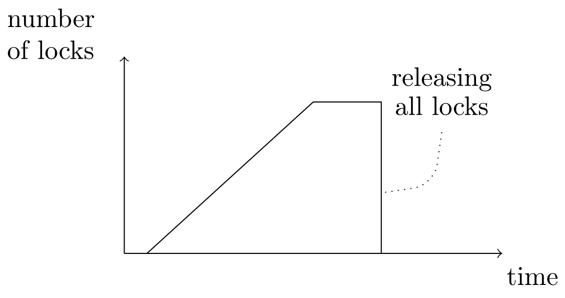


   - S2PL
     - exclusive locks are held until transaction commits

  - guarantees serializability and *strict schedules* (no dirty reads and writes)



- R2PL
  - all locks are held until transaction commits
  - guarantees serializability and *strict schedules*
  - easier to implement



3. Discuss the advantages and shortcomings of these protocols.

   Suggested solution

   See above.

–END–

5 SQLite scripts

```
-- create_tasks_db.sql
-- run sqlite3 tasks.sqlite3 < create_tasks_db.sql
-- to create the database file

DROP TABLE People;
DROP TABLE Tasks;
DROP TABLE DoTask;


CREATE TABLE People (
  pid INTEGER PRIMARY KEY,
  name TEXT NOT NULL
);

CREATE TABLE Tasks (
  tid INTEGER PRIMARY KEY,
  title TEXT NOT NULL,
  category TEXT NOT NULL,
  size INTEGER NOT NULL
);

CREATE TABLE DoTask (
  pid INTEGER NOT NULL,
  tid INTEGER NOT NULL,
  subsize INTEGER NOT NULL,
  status TEXT NOT NULL,
  updatedon TEXT NOT NULL,
  PRIMARY KEY (pid, tid)
);

.separator |
.import ./datafiles/People.txt People
.import ./datafiles/Tasks.txt Tasks
.import ./datafiles/DoTask.txt DoTask
```

```
-- queries_tasks.sql
-- run sqlite3 tasks.sqlite3 < queries_tasks.sql

.mode column
.headers on

select '- 1: names of all people';
SELECT name
FROM   people;
```

```
select '- 2: tasks titles and sub-sizes allocated to Ole';
SELECT title, subsize
FROM   people p, tasks t, dotask d
WHERE  p.name = "Ole" and p.pid = d.pid and t.tid = d.tid;

SELECT '- 3: people sharing dishes task';
SELECT name
FROM   people NATURAL JOIN dotask NATURAL JOIN tasks
WHERE  title = 'dishes';

select '- 4: tasks not allocated';
SELECT title
FROM   tasks
WHERE  tid NOT IN (SELECT tid FROM dotask);

select '- 5: people not sharing category with other';
SELECT p.name
FROM people p
WHERE  NOT EXISTS
       (SELECT category
        FROM tasks t, dotask d
        WHERE p.pid = d.pid AND t.tid = d.tid
       INTERSECT
        SELECT category
         FROM   people p2, tasks t2, dotask d2
         WHERE p2.pid <> p.pid AND p2.pid = d2.pid AND t2.tid = d2.tid);

select '- 6: Oles todo task in updated-on order';
SELECT title, subsize, updatedon
FROM   people natural join dotask natural join tasks
WHERE  name = "Ole" AND status = 'todo'
ORDER BY updatedon;

select '- 7: total sub-sizes finished on 2012-02-20';
SELECT sum(subsize) as finished_size
FROM   dotask
WHERE  updatedon = '2012-02-20' AND status = 'done';

select '- 8: tasks not completely done';
SELECT t.title
FROM   tasks t
WHERE  t.tid NOT IN (SELECT tid FROM dotask WHERE status = 'done')
       OR
       t.size >
       (SELECT sum(subsize)
        FROM tasks natural join dotask
        WHERE tid = t.tid AND status = 'done');

SELECT '- 9: tasks with shared duties';
SELECT title
FROM   dotask NATURAL JOIN tasks
GROUP BY tid
HAVING COUNT(*) > 1;

select '- 10: most productive date';
SELECT date, max(finished_size) as finished_size
FROM (SELECT updatedon as date, sum(subsize) as finished_size
      FROM   dotask
      WHERE  status = 'done'
      GROUP BY updatedon);
```