UiT THE ARCTIC UNIVERSITY OF NORWAY

ASSESSMENT GUIDELINE

For exam in: INF-2700 Database Systems
Date: Thursday 03.12.2015

The assessment guideline contains 9 pages, including this cover page

Contact person: Weihai Yu.
Phone: 41429077

UNIVERSITY OF TROMSØ

Question 1 (40%)

Below are some database tables with example data for a shopping application.

- Products

pid	pname	price
p01	cup	49
p02	hat	99
p03	pen	19

- Customers

cid	cname	city
c01	ida	oslo
c02	ida	alta
c03	ole	alta
c04	tom	oslo

- Orders

oid	cid	pid	quantity
101	c01	p01	3
102	c01	p02	5
103	c01	p02	1
111	c02	p01	2
112	c02	p02	2
113	c02	p03	2
121	c03	p01	1
122	c03	p01	9
131	c04	p01	1

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

Foreign keys in Orders:

- cid: references cid of Customers
- pid: references pid of Products

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*.

Note: In the result tables of your SQL queries, there should be *no* identical (duplicate) rows.

Relational algebra *and* SQL (1–5):

1. Names of all customers.

   The result for the example database is:

cname
ida
ole
tom

$\Pi_{cname} Customers$

```
SELECT DISTINCT cname
FROM   customers;
```

2. Products with price lower than 50 Kr.

   The result for the example database is:

pid	pname	price
p01	cup	49
p03	pen	19

$\sigma_{price<50} Products$

```
SELECT *
FROM   products
WHERE  price < 50;
```

3. Orders from Ida in Alta.

   The result for the example database is:

oid	pname	quantity
111	cup	2
112	hat	2
113	pen	2

$\Pi_{oid,pname,quantity}(\sigma_{cname='ida' \land city='alta'} Customers \bowtie Products)$

```
SELECT oid, pname, quantity
FROM   orders natural join customers natural join products
WHERE  cname = 'ida' and city = 'alta';
```

4. Cids of customers who ordered both p01 and p02.

   The result for the example database is:

cid
c01
c02

$\Pi_{cid}(\sigma_{pid='p01}Products) \cap \Pi_{cid}(\sigma_{pid='p02}Products)$

```
SELECT DISTINCT cid
FROM   orders
WHERE  pid = 'p01'
INTERSECT
SELECT cid
FROM   orders
WHERE  pid = 'p02';
```

or:

$\Pi_{cid}(\rho_{o1}(Orders) \bowtie_{o1.cid=o2.cid \wedge o1.pid='p01' \wedge o2.pid='p02'} \rho_{o2}(Orders))$

```
SELECT DISTINCT o1.cid
FROM   orders o1, orders o2
WHERE  o1.cid = o2.cid and o1.pid = 'p01' and o2.pid = 'p02';
```

5. Cids of customers who only ordered one kind of product.

   The result for the example database is:

cid
c03
c04

$\Pi_{cid}Orders - \Pi_{cid}(\rho_{o1}(Orders) \bowtie_{o1.cid=o2.cid \wedge o1.pid \neq o2.pid} \rho_{o2}(Orders))$

```
SELECT DISTINCT cid
FROM   orders
EXCEPT
SELECT o1.cid
FROM   orders o1, orders o2
WHERE  o1.cid = o2.cid and o1.pid <> o2.pid;
```

or:

$\sigma_{count(pid)=1}({}_{cid}\mathcal{G}_{count(pid)}(Orders))$

```
SELECT DISTINCT cid
FROM   orders
GROUP BY cid
HAVING COUNT(DISTINCT pid) = 1;
```

SQL *only* (6–10):

6. Number of different kinds of products.

   The result for the example database is:

numberOfProducts
3

```
SELECT COUNT(*) AS 'numberOfProducts'
FROM   products;
```

7. Total price of orders from customers in Oslo.

   The result for the example database is:

totalPriceFromOslo
790

```
SELECT SUM(price * quantity) AS 'totalPriceFromOlso'
FROM   orders natural join customers natural join products
WHERE  city = 'oslo';
```

8. Cids of customers who ordered more than two kinds of products.

   The result for the example database is:

cid	numberOfOrderedProducts
c02	3

```
SELECT cid, COUNT(DISTINCT pid) AS 'numberOfProducts'
FROM   orders
GROUP BY cid
HAVING numberOfProducts  > 2;
```

9. Cids of customers who ordered the most expensive product.

   The result for the example database is:

cid
c01
c02

```
SELECT DISTINCT cid
FROM   orders natural join products
WHERE  price = (SELECT MAX(price)
                FROM   products);
```

10. Cids of customers who ordered all products that c01 ordered.

    The result for the example database is:

cid
c02

```
SELECT DISTINCT cid
FROM    orders o
WHERE   cid != 'c01' and
        NOT EXISTS
         (SELECT pid
          FROM    orders
          WHERE   cid ='c01'
        EXCEPT
        SELECT pid
          FROM    orders
          WHERE   cid = o.cid);
```

Question 2 (20%)

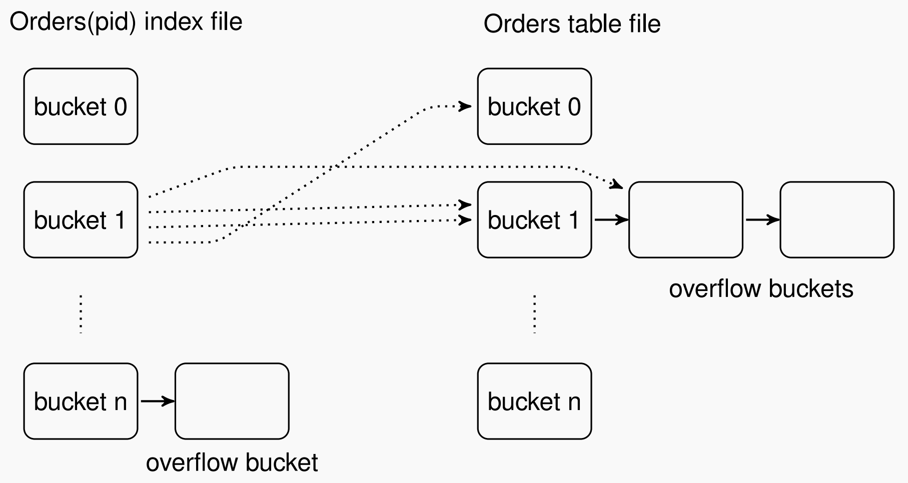
Now consider the physical data processing for the database in Question 1.

We decide to organize the database data as below:

- Table Products is organized with hash on pid.
- Table Customers is organized with hash on cid.
- Table Orders is organized with hash on cid. In addition, there is a hash index on pid.

Answer the following questions.

1. Sketch how Orders data are organized with a figure and some brief description.



Orders table file:

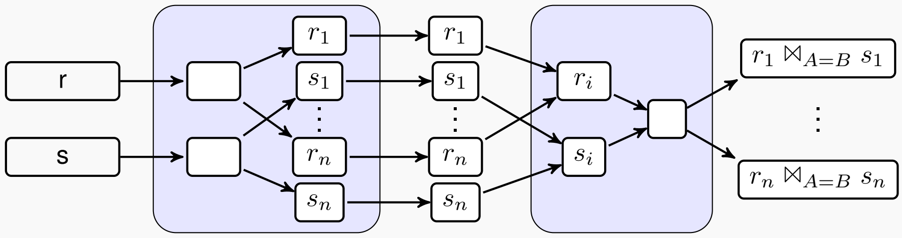
- A *hash-organized file* consists of buckets.
- A *bucket* is one or more disk blocks containing records.
- *hash function* $b = h(cid)$ calculates the address b of bucket from the search key value cid.
- All records in the same bucket have the same hash value on search key.
- When a bucket is full, *overflow buckets* are created.

Orders(pid) index file:

- The organization of the index file is similar to the hash-organized file.
- There is a pid index entry for every record in the Orders table.

2. We are going to make a natural join of the tables Customers and Orders $Customers \bowtie Orders$, with the *hash join* algorithm.

   Describe how the algorithm works.



With hash join, $r \bowtie s$ consists of two steps:

- partition r and s using a hash function,
- join the partitions with nested-loop join.

In our case, because both Orders and Customers are organized with hash on cid, we can skip the first step.

3. What is the primary performance overhead of database systems in general?

   Disk IOs: for hard disk, number of seeks and number of block transfers.

   What is the performance overhead of $Customers \bowtie Orders$ with hash join? (You should make reasonable assumptions of data sizes.)

   Assume that the main memory is large enough for each partition in the outer loop (the probing partition), each block from Orders and Customers is read into the memory only once.
   Let b_c and b_o be the number of blocks of tables Customers and Orders. Suppose $b_c < b_o$ and Customers is used in the outer loop. There are $2b_c$ seeks and $b_c + b_o$ block reads.
   We do not consider the overhead of writing the join result, because it is basically the same for all join algorithms (and depending on the use case, the result is not necessarily written back to disk).

Question 3 (20%)

Answer the following questions. Please explain the relevant concepts while answering the questions.

1. What is *functional dependency* $X \rightarrow Y$ of a relation instance r?

   Y's value is determined by X's value in r. More formally, for any pair of tuples in r, if they have the same value in X, they also have the same value in Y.

   For the example instance of table Customers in Question 1, check if the following functional dependencies are satisfied.

   a) $cid \rightarrow cname$

      Yes.

   b) $cname \rightarrow city$

      No.

   c) $\{cname, city\} \rightarrow cname$

      Yes.

   d) $\{cname, city\} \rightarrow cid$

      Yes.

2. What is *third normal form* (3NF)?

   For schema R with set of functional dependencies F. R is in 3NF if for any $\alpha \rightarrow \beta \in F$, one of the following is true:

   A. $\beta \in \alpha$ ($\alpha \rightarrow \beta$ is trivial),

   B. α is a superkey for R ($\alpha \rightarrow R$),

   C. each attribute in $\beta - \alpha$ is part of a candidate key for R.

3. Given the relation schema $R(A, B, C), F = \{A \rightarrow C, B \rightarrow C\}$. Explain why R is not in 3NF.

   AB is the only candidate key. None of A or B is a superkey. C is not part of the candidate key. So neither $A \rightarrow C$ nor $B \rightarrow C$ satisfies any of the above conditions.

4. Can you decompose schema R into 3NF with the *3NF synthesis algorithm*?

   i. Find a canonical cover F_c of F.

      F is already a canonical cover.

   ii. Make schemas from F_c.

      $R_1 = AC$, $R_2 = BC$.

   iii. The candidate key is not part of R_1 or R_2. Make a new schema out of the key: $R_3 = AB$.

   iv. Remove redundant schemas. There is none.

   The final result is $R_1 = AC$, $R_2 = BC$ and $R_3 = AB$.

5. Why do we need 3NF?

   Ideally, we need BCNF that allows functional dependencies of types A and B only. Then there is no redundancy caused by functional dependencies.
   Unfortunately, not every schema has a BCNF decomposition that is both lossless and preserves functional dependencies.
   For a relation schema, there is always a 3NF decomposition that is lossless (step iii) and preserves functional dependencies (due to condition C and step ii).

6. What problem may 3NF have?

   There may still be redundancies, because of condition C.

Question 4 (20%)

1. What is an *ACID transaction*?

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

   Atomicity The final effect on the data is all or nothing.

   Consistency Database is kept consistent (static) and individual transactions are consistent (dynamic).

   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 undesirable events.

2. What is a *log* for transaction processing?

   A log is a sequence of log records. It is used for transaction rollback and database recovery.

   How is a log organized?

   It typically has two parts: a stable part as a file on disk and a buffer part (tail) in main memory that will be flushed to disk.

   What are the operations on a log?

   A log is only updated by appending.

   append (unforced) appends a log record at the end of the log buffer.

   buffer flush the whole buffer part is flushed to disk (appended to the end of the log file).

   A forced append consists of an unforced append and a flush.
   A log is read upon rollback of transactions and recovery at DBMS restart after a system crash. It is typically read from the end and sequentially backward.

   What are the performance costs of these operations?

   An unforced append is a memory operation and the overhead can be ignored.
   A flush needs a seek and a number of disk block transfers.
   A read/scan needs a seek and a number of disk block transfers.
   If a hard drive is dedicated to the log, there is normally no need of seeks.

3. What kinds of log records do you need to rollback individual transactions when the database system is up and running?

   What information should the record contain?

   We need the following log records:

   updated (or undo) transaction id, before image (old value before update)

   To be able to undo an update

   start transaction id

   To know when a rollback is done

4. What if you also want the database to recover from a system crash?

   In addition, we need the following log records:

   commit/**abort** transaction id

   To know which transactions were active at the time of crash

   checkpoint transaction ids (active at the moment)

   To know when the recovery process can finish.

–END–