Second Set of Mandatory Exercises in INF3100 V2008

This set of exercises is primarily to be solved by groups of two and two students. We also accept individual assignments, but encourage you to work in groups. We do not permit groups of more than two persons. We will perform spot tests where we pick out single persons that must go through their solutions with us.

Preparation and submission of the assignment must be done in accordance with the directions in force at the Department of Informatics, see
`http://www.ifi.uio.no/studinf/skjemaer/erklaring.pdf` (Norwegian)
`http://www.ifi.uio.no/studinf/skjemaer/declaration.pdf` (English)
Submitting an assignment serves as an acknowledgement on the directions having been read and understood.

Deadline: **Friday 9. May, 12:00 o'clock.**
The deadline is final; no delays will be granted. All exercises must be answered in order to get the assignment approved.

Write complete names and user names on the assignment. The assignment is to be sent by email as a PDF file to the group teacher. The email shall have the following subject:
`Subject: Oblig 2 inf3100 (<username studt1>, <username studt2>)`

Students that have an approved mandatory exercise and still want to withdraw from the exam, should provide the group teacher with a paper copy and obtain an endorsement. This applies only to students that withdraw from the exam before the 14-day withdrawal limit.

Consider a simple relational database for a tools wholesaler. The customers are retailers (shops or enterprises, or even individuals, that sell the tools to private persons and professionals). The retailers can order products from the wholesaler using an internet application. The wholesaler has several warehouses in different locations; when a customer orders products, the items are picked from the warehouse(s) that are closest to the customer's address.

The database contains a relation *Employee* where every employee is assigned a unique employee number and where the name and address of each employee are found. The database also contains information about the products and warehouses. The relation *Product* contains for each product a corresponding unique code, a product name, and unit price. The relation *Stock* contains information about how many units of each product are stored in each warehouse.

The products are classified in product groups, this classification is found in *ProductGroup*. A product can belong to several product groups. Some employees have special knowledge about some product groups. The relation *AreaOfResponsibility* contains information about this.

In the *Customer* relation each customer is assigned a unique customer number, the customer's name, if it is a one-person enterprise, address, and the enterprise's official registration number. To have a bookkeeping of the orders the database has two more relations *OrderLine* and *Order*. Each order has a corresponding order number *orderID*. The relation *OrderLine* contains for each product in an order how many units the order comprises. *Order* contains information about the identity of the customer of each order and the ordering date.

In the database schema primary keys are underlined twice. Other candidate keys have one underline. There are foreign keys from *Stock* and *ProductGroup* to *Product*, from *AreaOfResponsibility* to *Employee*, from *OrderLine* to *Product* and *Order*, and from *Order* to *Customer*. Attributes contained in foreign keys are named after the attributes they refer to.

$$
\begin{aligned}
&\mathit{Employee}\,(\underline{\underline{\mathit{empID}}}, \mathit{name}, \mathit{address})\\
&\mathit{Customer}\,(\underline{\underline{\mathit{custID}}}, \mathit{name}, \mathit{ftype}, \mathit{address}, \underline{\mathit{enterprno}})\\
&\mathit{Product}\,(\underline{\underline{\mathit{prodID}}}, \mathit{prodname}, \mathit{unitpr})\\
&\mathit{Stock}\,(\underline{\underline{\mathit{prodID}, \mathit{whname}}}, \mathit{nmb})\\
&\mathit{ProductGroup}\,(\underline{\underline{\mathit{prodID}, \mathit{prodgr}}})\\
&\mathit{AreaOfResponsibility}\,(\underline{\underline{\mathit{prodgr}, \mathit{empID}}})\\
&\mathit{OrderLine}\,(\underline{\underline{\mathit{orderID}, \mathit{prodID}}}, \mathit{nmbord})\\
&\mathit{Order}\,(\underline{\underline{\mathit{orderID}}}, \mathit{custID}, \mathit{date})
\end{aligned}
\tag{1}
$$

The attribute *date* in *Order* is assumed to be a standard SQL DATE, i.e. text on e.g. the format '2007-12-31'. *Customer* can be an individual or a company, this can be seen from the value of *ftype* in *Customer*, which for individuals

is 'E', while companies have a 'B'. If the customer is not a company, then *enterprno* is the social security number instead.

1 FDs and MVDs

Consider the following alternative to the relational database schema (1):

$$
\begin{aligned}
&\mathit{Employee}\,(\mathit{empID}, \mathit{name}, \mathit{address})\\
&\mathit{Product}\,(\mathit{prodID}, \mathit{prodname}, \mathit{unitpr})\\
&\mathit{Stock}\,(\mathit{prodID}, \mathit{whname}, \mathit{nmb})\\
&\mathit{ProductGroupInfo}\,(\mathit{prodID}, \mathit{prodgr}, \mathit{empID})\\
&\mathit{OrderLine}\,(\mathit{orderID}, \mathit{prodID}, \mathit{nmbord})\\
&\mathit{OrderInfo}\,(\mathit{orderID}, \mathit{date}, \mathit{custID}, \mathit{name}, \mathit{ftype}, \mathit{address}, \mathit{enterprno})
\end{aligned}
\tag{2}
$$

(i) Legal instances of *OrderInfo* in schema (2) shall reflect exactly the legal instances of the relations *Customer* and *Order* in schema (1). Which FDs must in that case hold in *OrderInfo*?

(ii) Describe which normal forms are violated in *OrderInfo*. For each violation, describe the FDs that violate the normal form and why it is violated.

(iii) Legal instances of *ProductGroupInfo* in schema (2) shall reflect exactly the legal instances of the relations *ProductGroup* and *AreaOfResponsibility* in schema (1). Which FDs and MVDs must in that case hold in *ProductGroupInfo*?

(iv) Describe which normal forms are violated by the MVDs in (iii) and why they are violated.

2 SQL

Consider the schema in (1) and answer the following questions using SQL.

(i) Find name and address of all customers that have ordered products from the product group 'POWERAGGREGATES'.

(ii) Make a list containing name, address and the total amount each customer has made orders for in May 2007. Customers that have not ordered any products this month, shall not be included in the list.

(iii) Make a list of all products in the product group 'POWERAGGREGATES'. Each line in the list shall contain the product's name, unit price, total number of units ordered in year 2007, and the number of different customers that ordered the product in 2007. Sort the list according to total number of ordered items, and such that the product with the highest number of orders is listed first, and products that haven't been ordered by anybody, are listed last.

3 OQL

In appendix 1 is a description of an object-oriented version of the tools wholesaler database. Use this description as a basis, and answer the questions (i)-(iii) in exercise 2 using OQL.

4 Implementation

In this problem you will demonstrate your understanding of how a database system is implemented. We will take as our starting point the relational database described in schema (1).

Consider the following SQL-query that finds the names and addresses of one-person enterprises who have ordered products from the product group FORKLIFTS in 2007:

```
select   Customer.name, Customer.address
from     Customer, Order, OrderLine, ProductGroup
where    Customer.custID = Order.custID                  and
         Order.orderID = OrderLine.orderID               and
         OrderLine.prodID = ProductGroup.prodID          and
         Customer.ftype = 'E'                            and
         ProductGroup.prodgr = 'FORKLIFTS'               and
         Order.date like '2007%'
```

The database has clustered indexes on the primary keys. In addition there are indexes on the attribute *date* in *Order* and the attribute *enterprno* in *Customer*.

4.1 Parsing

(i) Use the simple grammar in appendix 2 for making a parse tree of the query above.

(ii) What are the main tasks of the preprocessor?

4.2 Logical query plan

Convert the parse tree in exercise 4.1(i) to a logical query plan in relational algebra (draw the expression tree). NB! The solution shall not be optimized.

4.3 Optimization

Optimize the logical query plan in exercise 4.2 (draw the new expression tree).

4.4 Execution

If the volume of the data is so large that we cannot keep the complete data set in memory, we base execution of the (full relation) operators on two principles.

(i) Name the principles and the idea behind each. (Give a brief description.)

(ii) In which cases will you use the different methods? Motivate your answer.

4.5 Data storage

In this exercise we shall focus on the following subset of schema (1):

Customer (*custID*, *name*, *ftype*, *address*, *enterprno*)

Product (*prodID*, *prodname*, *unitpr*)

OrderLine (*orderID*, *prodID*, *nmbord*)

Order (*orderID*, *custID*, *date*)

Assume that we have a disk with the following specifications for storing our data:

Disk platters:	10 (with 2 surfaces each)
Tracks:	10.000 per surface
Number of sectors per track:	1000 (a non-zoned disk)
Byte per sector:	512
Byte per gap:	64
Average search time:	5 ms.
Track-track search:	0,5 ms
Rotation speed:	15.000 RPM

Also assume that each relation is stored clustered on the disk. The database contains the following number of tuples:

Number of *Customer*-tuples:	100.000
Number of *Product*-tuples:	10.000
Number of *OrderLine*-tuples:	10.000.000
Number of *Order*-tuples:	1.000.000

In addition the following information is at hand:

- Every block has a header of 20 bytes.
- Every record has a head of 10 bytes.
- Every attribute has the following size in bytes:

Customer		*Product*		*OrderLine*		*Order*	
custID:	16	*prodID*:	16	*orderID*:	16	*orderID*:	16
name:	40	*prodname*:	40	*prodID*:	16	*custID*:	16
ftype:	1	*unitpr*:	10	*nmbord*:	10	*date*:	10
address:	40						
enterprno:	11						

(i) What is the utilizable capacity of the disk? (Remember that each disk has two surfaces.)

(ii) What factors apply when accessing a disk block, and what is average access time for an arbitrary 4 Kbyte block?

(iii) How much space do the relations need on disk in the case of "unspanned" storage (i.e. if no single record needs more than one block)?

(iv) How long time does it take to read all of the relation *OrderLine* uninterrupted if we assume arbitrary placement of data in disk blocks on the disk?

(v) How can one optimize such a read of a relation by changing block placement, and what will the new time for reading all of *OrderLine* be? NB: Write down any assumptions that you make!

5 Transactions

The customers order products over the Internet using a "shopping cart" application. Typically the customers will first go through a phase where they examine what products are available, and then a phase where they choose which products, and how many units, to order. At completion they ask the shopping cart application to carry out and confirm the order.

We are going to concentrate on what happens in the relation *Stock*. In order to complete an order on behalf of a customer, the application executes a transaction that reads the tuples of the products that are to be ordered, checks that there are sufficient units in stock of each product, and for those where there are enough units available, determines from which warehouses to collect the items, and finally counts down *nmb*. If for a product there is not a sufficient number of units available, no order is made of that product.

If we let $r_i(A)$ and $w_i(A)$ denote that a transaction T_i respectively reads and writes a tuple A in *Stock*, two possible transactions in a given database state can be as follows:

$$\begin{aligned} T_1 :&\quad r_1(A); r_1(B); w_1(A); w_1(B) \\ T_2 :&\quad r_2(A); r_2(B); w_2(B) \end{aligned}$$

(T_1 orders units of two products, represented by A and B respectively; T_2 only orders units of B because it turned out not to be sufficient units of A in stock.)

5.1 Serializability

Consider the following schedule S_1 of T_1 and T_2:

$$S_1 : r_1(A); r_2(A); r_1(B); w_1(A); w_1(B); r_2(B); w_2(B)$$

(i) Show that S_1 is not conflict serializable.

(ii) Find an assumption about A and B that makes S_1 serializable. Is it reasonable to assume that S_1 is serializable in the shopping cart application? State the reasons for your answer.

Consider a slightly improved application where the wholesaler is interested in advertising on his homepage how well visited the page is. Every transaction therefore concludes by writing a time stamp on the home page, this time stamp illustrates when the homepage was last visited. Writing a time stamp is done by the operation $w_i(Z)$. Consider the following execution plan S_2 consisting of three transactions T_3, T_4 and T_5:

$$S_2 : \quad r_3(A); w_3(A); r_4(A); r_4(B); r_5(C); w_4(B); \\ w_4(Z); w_3(Z); w_5(C); w_5(Z)$$

(iii) Draw the precedence graph of S_2.

(iv) Is S_2 conflict serializable? Give reasons for your answer.

(v) Draw the polygraph of S_2.

(vi) Is S_2 view serializable? State the reasons for your answer.

5.2 Concurrency control

5.2.1 Pessimistic protocol

Assume that we have exclusive locks on each tuple in *Stock*. The locks are to be used as usual, in that each read or write action is to be preceded by a locking and succeeded by an unlocking operation. In addition every transaction is to use two phase locking (2PL).

Let the action $l_i(Y)$ denote that T_i takes the lock on Y, and $u_i(Y)$ that T_i unlocks Y.

(i) Add actions of the form $l_i(Y)$ and $u_i(Y)$ to each of T_1 and T_2 such that they fulfil the locking rules under 2PL, and also release locks as soon as feasible.

(ii) Describe what happens if we try to execute the actions of the resulting transactions such that the read and write actions are executed as much as possible in the order given by S_1.

Now assume that we for each tuple in *Stock* have two locks – a shared one (S-lock) and an exclusive one (X-lock), where an S-lock can be upgraded to (traded for) an X-lock at need. In other respects the locks are to be used as usual for S/X-locks and in accordance with 2PL.

Let the actions $ls_i(Y)$ and $lx_i(Y)$ denote that T_i takes the S- and X-lock respectively on Y, and $u_i(Y)$ that T_i unlocks all locks on Y.

(iii) Add actions of the form $ls_i(Y)$, $lx_i(Y)$, and $u_i(Y)$ to each of T_1 and T_2 such that they obey the rules of S/X-locks under 2PL, and such that the transactions do not use X-locks more than strictly required (i.e., they use upgrading whenever possible). Locks are to be unlocked as soon as feasible.

(iv) Describe what happens if we try to execute the actions of the resulting transactions such that the read and write actions are executed as much as possible in the order given by S_1.

5.2.2 Optimistic protocol

We will now look at what happens if we use a time stamp protocol. Assume that T_1 gets the time stamp t_1 and T_2 the time stamp t_2, where $t_1 < t_2$.

(i) Descibe what happens with T_1 and T_2 if we try to execute the actions in the order given by S_1.

Assume further that T_3, T_4 and T_5 get the time stamps t_3, t_4 and t_5 respectively, where $t_3 < t_4 < t_5$. Also assume that T_4 aborts after all its actions have been done.

(ii) Describe what happens with T_3 and T_5 if we try to execute the actions in the order given by S_2.

6 Logging

(i) Describe briefly the principle of undo logging. What kind of log-records do you need? When shall the different log-records be written to disk?

(ii) Describe briefly the difference between undo and redo logging.

7 RAID 6

A common type of disk cabinet (in 2008) contains 14 physical disks numbered from 1 to 14. Assume that we organize these as RAID 6 with disks 1, 2, 4, and 8 as Hamming-coded parity disks. The remainding 10 disks are ordinary data disks.

Explain how the system can reconstruct disks 3 and 11 if these crash at the same time and need replacement.

Appendix 1 – Object-oriented version of the wholesaler database

```
class Employee (extent employees key empID)
{
  attribute string empID;
  attribute string name;
  attribute string address;
  relationship Set<ProductGroup> areasofresponsibility
    inverse ProductGroup::resposibles;
}

class Customer (extent customers key custID)
{
  attribute string custID;
  attribute string name;
  attribute string address;
  relationship EnterpriseNumber enterprnumber
    inverse EnterpriseNumber::enterprinfo;
  relationship Set<Order> orders
    inverse Order::customer;
}

class EnterpriseNumber (extent enterprisenumbers key enterprno)
{
  attribute integer enterprno;
  attribute character ftype;
  relationship Customer enterprinfo
    inverse Customer::enterprnumber;
}
```

```
class Product (extent products key prodID)
{
  attribute string prodID;
  attribute string prodname;
  attribute integer unitpr;
  relationship Set<OrderLine> orders
    inverse orderline::product;
  relationship Set<Stock> stock
    inverse Stock::product;
  relationship Set<ProductGroupRel> productgroups
    inverse ProductGroupRel::products;
}

class Stock (extent stocks)
{
  attribute integer nmb;
  relationship Warehouse warehouse
    inverse Warehouse::stock;
  relationship Product product
    inverse Product::stock;
}

class Warehouse (extent warehouses)
{
  attribute string whname;
  relationship Set<Stock> stock
    inverse Stock::warehouse;
}

class ProductGroupRel (extent productgrouprels)
{
  attribute string prodgr;
  relationship Set<Employee> responsibles
    inverse Employee areasofresponsibility;
  relationship Set<Product> products
    inverse Product::productgroups;
}
```

```
class Order (extent orders key orderID)
{
  attribute string orderID;
  attribute date date;
  relationship Customer customer
    inverse Customer::orders;
  relationship Set<OrderLine> orderlines
    inverse OrderLine::order;
}

class OrderLine (extent orderlines)
{
  attribute integer nmb;
  relationship Order order
    inverse Order::orderlines;
  relationship Product product
    inverse Product::products;
}
```

Appendix 2 – Grammar for query parsing

```
<query> ::= <SFW>

<SFW> ::= SELECT <selList> FROM <fromList> WHERE <condition>

<selList> ::= <attribute>
<selList> ::= <attribute>, <selList>

<fromList> ::= <relation>
<fromList> ::= <relation, <fromList>

<condition> ::= <condition> AND <condition>
<condition> ::= <attribute> = <attribute>
<condition> ::= <attribute> = <pattern>
<condition> ::= <attribute> LIKE <pattern>
```

Elementary syntactical categories like `<attribute>`, `<relation>`, and `<pattern>` have no rules, but are translated to the name of the attribute, the name of the relation, and a string with quotes respectively.

End of mandatory exercise 2