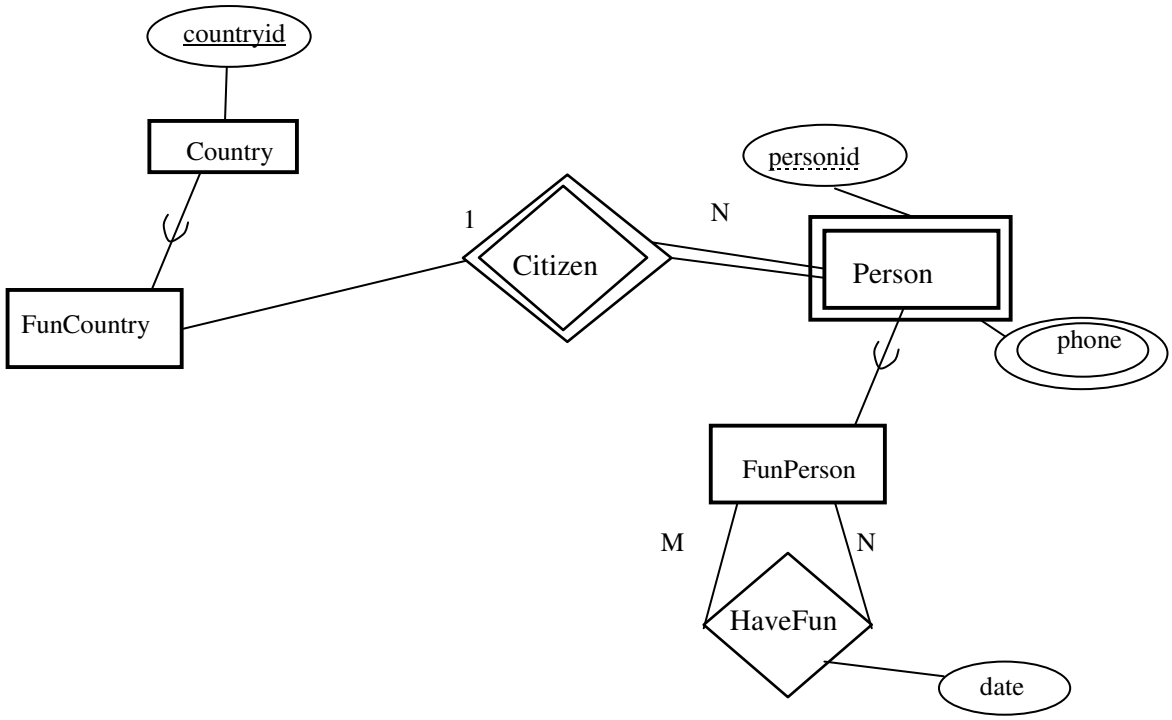
TDDD37/TDDB77 October 2010

Question 3. Translation of EER to relational schema (4 p):



Translate the EER diagram above into a relational model (you have to follow the algorithm seen in the course). Mark the primary keys with solid underlining and the foreign keys with dotted underlining and an arrow from the foreign key to the attribute(s) pointed by the foreign key.

A solution:

Country(countryid)

FunCountry(countryid) **0.5 p**

Person(personid,countryid) **1 p**

Phone(number,personid,countryid) **1 p**

FunPerson(personid,countryid) **0.5 p**

HaveFun(date,personid1,countryid1,personid2,countryid2) **1 p**

Question 6. Transactions and concurrency control (1 + 1 + 1 = 3 p):

a. Is the following transaction schedule serializable? Motivate your answer.

T1	**T2**	**T3**
	read(y)	
	y:=y+1	
	write(y)	
read(x)		
x:=x+1		
write(x)		
		read(y)
		y:=y+1
		write(y)
	read(x)	
	x:=x+1	
	write(x)	

b. Apply the two-phase locking protocol to the transactions above.

c. Complete the following sentence with one of the four options given. If we use the two-phase locking protocol there can be a) starvation, b) deadlocks, c) both starvation and deadlocks, d) neither starvation nor deadlocks.

Solution:

a. The existing conflicts are between T1's read(x) and T2's write(x), between T1's write(x) and T2's read(x), between T1's write(x) and T2's write(x), between T2's read(y) and T3's write(y), between T2's write(y) and T3's read(y), between T2's write(y) and T3's write(y). These conflicts give rise to the following conflict graph:

T1 ⟶ T2 ⟶ T3

Since the conflict graph above has no directed cycle, the schedule is serializable.

b.

T1

lock write(x)
read(x)
x:=x+1
write(x)
unlock(x)

T2

lock write(x)

lock write(y)
read(y)
y:=y+1
write(y)
read(x)
x:=x+1
write(x)
unlock(x)
unlock(y)

T3

lock write(y)
read(y)
y:=y+1
write(y)
unlock(y)

c. With the two-phase locking protocol, there can be both starvation and deadlocks. To see the deadlocks part, consider a transaction T1 that issues lock write(x), followed by another transaction T2 issuing lock write(y), followed by T1 issuing lock write(y), followed by T2 issuing lock write(x). To see the starvation part, note that whether starvation happens or not has nothing to do with the two-phase locking protocol but with the strategy used to re-start the transactions aborted. So, this strategy can be so bad as to create starvation.

Question 7. Database recovery (3 p):

Apply the three recovery methods seen in the course to the system log below. Show all operations that are performed during the recovery. In the correct order!

Part of system log:

Start-transaction T1
Write-item T1, A, 1, 2
Start-transaction T2
Write-item T1, A, 2, 4
Write-item T2, B, 5, 6
Commit T1
Start-transaction T3
Start-transaction T4
Write-item T3, C, 7, 8
Write-item T3, C, 8, 10
Write-item T2, B, 6, 12
Checkpoint
Commit T2
→system crash

Solution:

Note that T2 is the only transaction that has committed after the checkpoint, and that T3 and T4 are still active when the system crashes.

Deferred update (NO-UNDO/REDO): Redo T2's operations in the order they appear in the system log.

Write-item T2, B, 5, 6
Write-item T2, B, 6, 12

Immediate update I (UNDO/NO-REDO): Undo T3's and T4's operations in the reverse order they appear in the system log.

UNDO Write-item T3, C, 8, 10 = Write-item T3, C, ?, 8
UNDO Write-item T3, C, 7, 8 = Write-item T3, C, ?, 7

Immediate update II (UNDO/REDO): First, undo T3's and T4's operations in the reverse order they appear in the system log and, then, redo T2's operations in the order they appear in the system log.

UNDO Write-item T3, C, 8, 10 = Write-item T3, C, ?, 8
UNDO Write-item T3, C, 7, 8 = Write-item T3, C, ?, 7
Write-item T2, B, 5, 6
Write-item T2, B, 6, 12

Question 8. Optimization (1 + 1 + 1 = 3p)

a. Let A, B, C and D be four tables with 10 attributes each. Each of the attributes has the UNIQUE constraint. Optimize the following MySQL query:

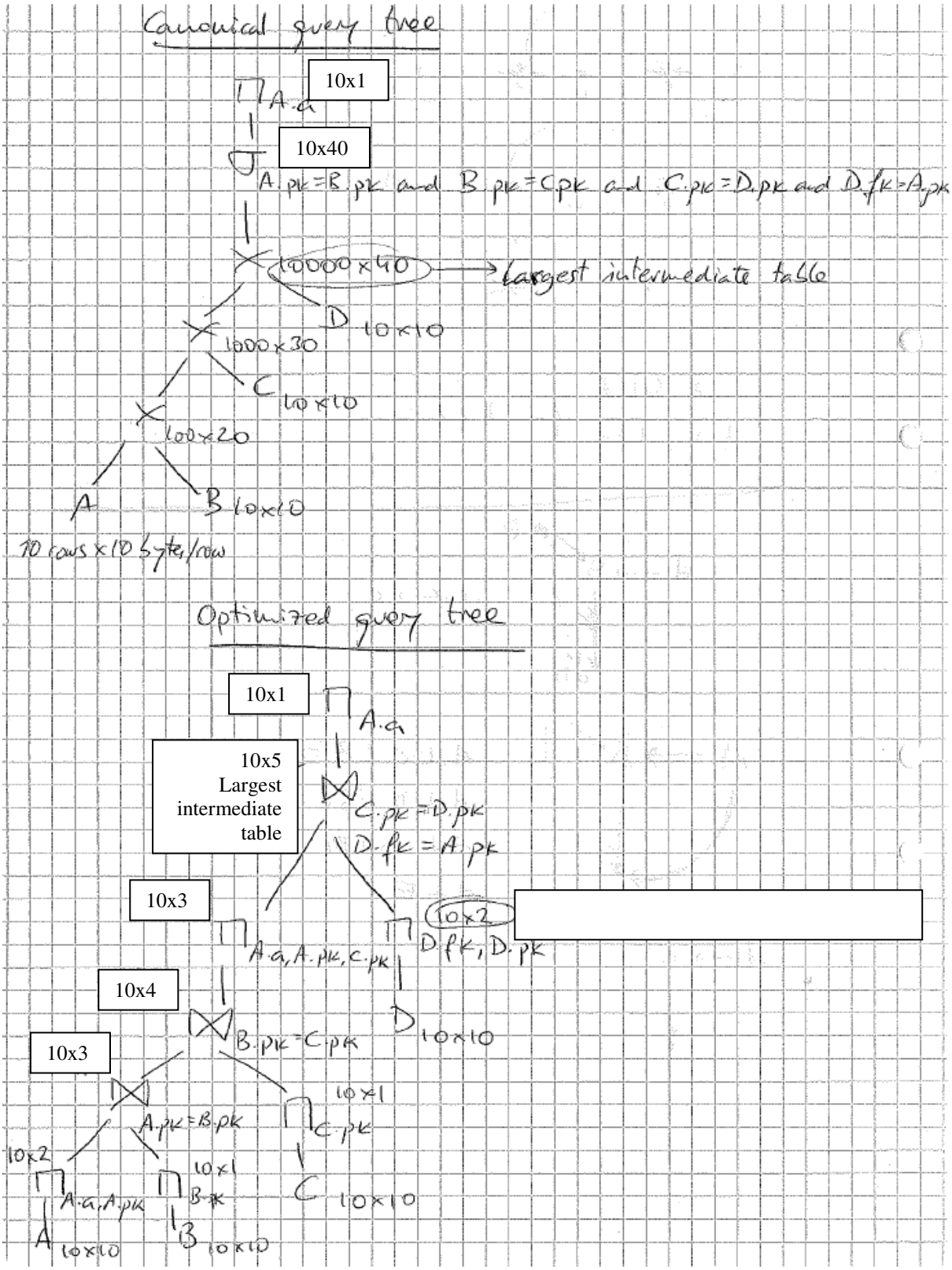
```
SELECT A.a
FROM A, B, C, D
WHERE A.pk=B.pk AND B.pk=C.pk AND C.pk=D.pk AND D.funnykey=A.pk;
```

b. Assume that the tables do not contain any NULL value. Assume also that each table contains 10 tuples and that each attribute is of size 1 byte. Show that the optimized query tree is more efficient than the canonical query tree.

c. Why does query optimization replace a selection followed by a Cartesian product with a join operation ?

Solution:

a and b.

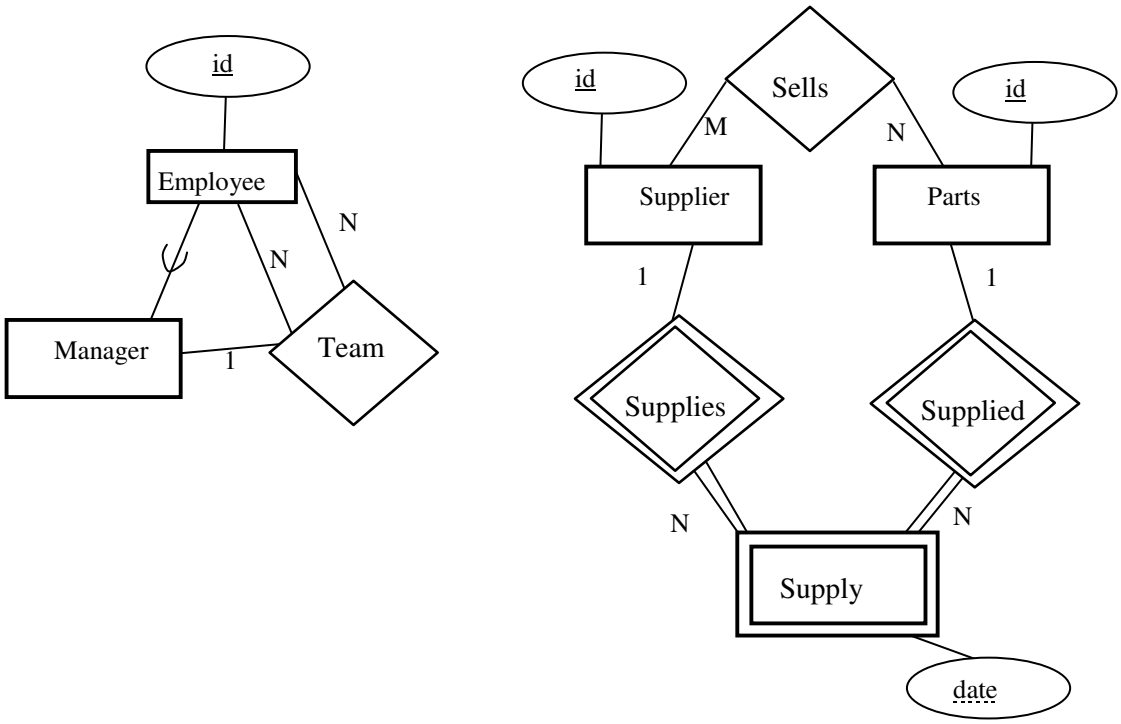


c.

A join implies less space for the intermediate tables since it can be implemented efficiently, i.e. without creating the Cartesian product result first.

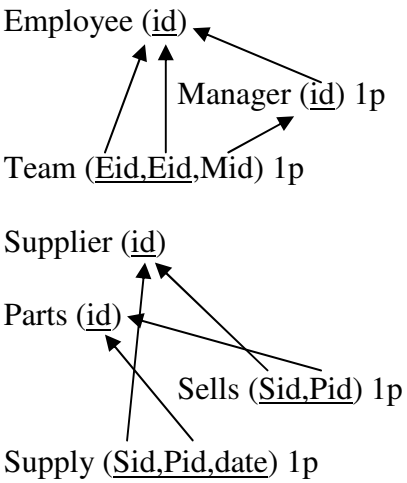
TDDD37/TDDB77 January 2011

Question 3. Translation of EER to relational schema (4 p):



Translate the EER diagram above into a relational model (you have to follow the algorithm seen in the course). Mark the primary keys with solid underlining and the foreign keys with dotted underlining and an arrow from the foreign key to the attribute(s) pointed by the foreign key.

A solution:



Question 6. Transactions and concurrency control (2+1+2=5p):

a. Use the two transactions below to give an example of a transaction schedule that is serializable but not serial. Explain why the schedule is serializable.

T1	**T2**
read(x)	read(x)
read(y)	read(y)
x:=x+y	y:=y+x
write(x)	write(y)

b. Apply the two-phase locking protocol to each of the transactions above.

c. Use the two transactions below to give an example of a transaction schedule that is not serializable. Explain why the schedule is not serializable.

A solution:

a.

T1	**T2**
read(x)	
	read(x)
	read(y)
	y:=y+x
	write(y)
read(y)	
x:=x+y	
write(x)	

The only conflicts in the schedule above are between T2's r(x) and T1's w(x), and between T2's w(y) and T1's r(y). Then, the conflict graph looks like: T2 $\longrightarrow$ T1. This graph does not have any directed cycle and, thus, the schedule above is serializable. The schedule above is not serial because it is not equal to T1 followed by T2 nor to T2 followed by T1. It is (conflict) EQUIVALENT to the latter though. This is what serializable means.

b.

T1	**T2**
lock write(x)	lock read(x)
lock read(y)	lock write(y)
read(x)	read(x)
read(y)	read(y)
x:=x+y	y:=y+x
write(x)	write(y)
unlock(x)	unlock(x)
unlock(y)	unlock(y)

c.

T1	**T2**
read(x)	
read(y)	
	read(x)
	read(y)
	y:=y+x
	write(y)
x:=x+y	
write(x)	

The only conflicts in the schedule above are between T2's r(x) and T1's w(x), and between T2's w(y) and T1's r(y). Then, the conflict graph looks like: T2 ⇄ T1. This graph has a directed cycle and, thus, the schedule above is not serializable.

Question 7. Database recovery (3p):

Apply the three recovery methods seen in the course to the system log below. Show all operations that are performed during the recovery. In the correct order!

Part of system log:

Start-transaction T1
Write-item T1, A, 1, 2
Start-transaction T2
Write-item T1, A, 2, 4
Write-item T2, B, 5, 6
Start-transaction T3
Start-transaction T4
Write-item T4, C, 6, 7
Write-item T3, C, 7, 8
Write-item T3, C, 8, 10
Checkpoint
Write-item T2, B, 6, 12
Checkpoint
Commit T1
Commit T2
→system crash

Solution:

Note that T1 and T2 are the only transaction that have committed after the last checkpoint, and that T3 and T4 are still active when the system crashes.

Deferred update (NO-UNDO/REDO): Redo T1 and T2's operations in the order they appear in the system log.

Write-item T1, A, 1, 2
Write-item T1, A, 2, 4
Write-item T2, B, 5, 6
Write-item T2, B, 6, 12

Immediate update I (UNDO/NO-REDO): Undo T3's and T4's operations in the reverse order they appear in the system log.

UNDO Write-item T3, C, 8, 10 = Write-item T3, C, ?, 8
UNDO Write-item T3, C, 7, 8 = Write-item T3, C, ?, 7
UNDO Write-item T4, C, 6, 7 = Write-item T4, C, ?, 6

Immediate update II (UNDO/REDO): First, undo T3's and T4's operations in the reverse order they appear in the system log and, then, redo T1 and T2's operations in the order they appear in the system log.

UNDO Write-item T3, C, 8, 10 = Write-item T3, C, ?, 8
UNDO Write-item T3, C, 7, 8 = Write-item T3, C, ?, 7
UNDO Write-item T4, C, 6, 7 = Write-item T4, C, ?, 6
Write-item T1, A, 1, 2
Write-item T1, A, 2, 4
Write-item T2, B, 5, 6
Write-item T2, B, 6, 12

Question 8. Optimization (1+1+1=3p)

It was the same optimization question as in TDDD37/TDDB77 October 2010.

TDD12 May 2011

Uppgift 5. Datastrukturer (2 + 1 +1 = 4 p):

Vi har en tabell med 32999 poster. Varje post är 3 bytes lång. Posterna har två nyckel fält (eng. key attributes) X och Y. Filen är sorterad enligt fält X. Databasen använder blockstorleken B = 100 bytes och posterna lagras obrutna (eng. unspanning).

a) Hur många block måste access:as för att hitta en post med ett givet värde för fältet X
   I. när man inte använder något index alls ?
   II. när man använder ett index ? Varje indexpost är 2 bytes lång.
b) Hur många block måste access:as för att hitta en post med ett givet värde för fältet Y
   I. när man inte använder något index alls ?
   II. när man använder ett index ? Varje indexpost är 2 bytes lång.
c) I vilket av fallen ger indexet bäst förbättring ? Varför ?

Obs. $\log_2 2^x = x$.

Solution:

a.I) X is a key ordering field. Then, we can run a binary search to find a given value of X. In the worst case, this takes ceiling(log_2 b) where b is the number of blocks to store the data file. To compute b, compute first the blocking factor (i.e. number of data records per block):

bf = floor(100/3) = 33 data records/block

Note that we used the floor function because of unspanned allocation (i.e. records must be store completely in the block). Then,

b = ceiling(32999/33) = 1000 data blocks

Then, the number of data blocks to access is at most ceiling(log_2 1000) = 10.

a.II) Since X is the key ordering field, the index is a primary index. In a primary index, we have as many index records as data blocks we have, i.e. 1000. Let us first compute how many blocks we need to store the index. The blocking factor for the index is

bf_i = floor(100/2) = 50 index records/block

Then, we need

b_i = ceiling(1000/50) = 20 blocks to store the index

Since the index file is ordered by definition, we can run a binary search to find the data block where the value of X is. Then, we need at most ceiling(log_2 20) + 1 block accesses to get to the data, i.e. 6 accesses. Note that the "+1" comes from the access to read the data whereas the other five come from the search for the pointer to the data block in the index file.

b.I) Since Y is not an ordering field, then we have to run a linear search to find a given value of Y. Therefore, the average number of data blocks to access in order to find the given value is 1000/2.

b.II) Since Y is not an ordering field, the index is a secondary index. Note that Y is a key field. Then, the index has as many entries as data records, i.e. 32999. Since the blocking factor for the index is 50, the number of blocks to store the index is

b_i = ceiling(32999/50) = 660 blocks

Since the index file is ordered by definition, we can run a binary search to find the data block where the value of Y is. Then, we need at most ceiling(log_2 660) + 1 block accesses to get to the data, i.e. 11 accesses. Note that the "+1" comes from the access to read the data whereas the other 10 come from the search for the pointer to the data block in the index file.

c) The improvement is greater in the case b) because the index allows us to reduce the number of accesses from linear in the number of data blocks to log_2. In the case a) the complexity is log_2 in both cases because X is the ordering field. Of course, this does not mean that the index in case a) is useless, because it still provides us with some gain, but not as much as in case b).

Uppgift 6. Transaktioner och samtidighet (2 + 1 = 3 p):

a) Ge ett transaktionsschema som är serialiserbart men ej seriellt. Bevisa att schemat är serialiserbart. Schemat måste innehålla minst två transaktioner och varje transaktion måste innehålla minst en read och en write instruktion.
b) Använd tvåfaslåsning protokollet (eng. two phase locking protocol) på transaktionen nedan.

```
Read(a);
a:=a+1;
Write(a);
Read(b);
b:=a+b;
Write(b);
```

Solution:

a)

T1	**T2**
read(x)	
	read(x)
	read(y)
	y:=y+x
	write(y)
read(y)	
x:=x+y	
write(x)	

The only conflicts in the schedule above are between T2's r(x) and T1's w(x), and between T2's w(y) and T1's r(y). Then, the conflict graph looks like: T2 ⟶ T1. This graph does not have any directed cycle and, thus, the schedule above is serializable. The schedule above is not serial because it is not equal to T1 followed by T2 nor to T2 followed by T1. It is (conflict) EQUIVALENT to the latter though. This is what serializable means.

b)

```
LockWrite(a);
LockWrite(b);
Read(a);
a:=a+1;
Write(a);
Read(b);
b:=a+b;
Write(b);
Unlock(a);
Unlock(b);
```

Uppgift 7. Databasåterställning (3 + 1 = 4p):

a) Använd de tre återställningsmetoder vi har sett i kursen på systemloggen nedan. Visa alla operationer som görs vid återställningen av databasen. I rätt ordning!
b) Behöver man lagra all information som lagrats i systemloggen nedan för återställning med omedelbar uppdatering version 1 (eng. immediate update version 1, i.e. no-redo/undo) ? Förklara korfattat ditt svar.

Part of system log:

```
Start-transaction T1
Write-item T1, A, 10, 20
Start-transaction T2
Write-item T1, B, 10, 20
Write-item T2, C, 10, 20
Commit T1
Start-transaction T3
Write-item T3, D, 20, 30
Checkpoint
Write-item T2, C, 20, 40
Commit T2
   →system crash
```

Solution:

a) Note that T2 is the only transaction that has committed after the last checkpoint, and that T3 is still active when the system crashes.

Deferred update (NO-UNDO/REDO): Redo T2's operations in the order they appear in the system log.

```
Write-item T2, C, 10, 20
Write-item T2, C, 20, 40
```

Immediate update I (UNDO/NO-REDO): Undo T3's operations in the reverse order

they appear in the system log.

UNDO Write-item T3, D, 20, 30 = Write-item T3, C, ?, 20

Immediate update II (UNDO/REDO): First, undo T3's operations in the reverse order they appear in the system log and, then, redo T2's operations in the order they appear in the system log.

UNDO Write-item T3, D, 20, 30 = Write-item T3, C, ?, 20
Write-item T2, C, 10, 20
Write-item T2, C, 20, 40

b) We do not need to store the new value of the write operations, because we will never redo them. We do not need to store the checkpoints either, because we know that all the changes are in disk whenever we see a commit instruction. So, the checkpoints do not really save anything to disk. For the same reason, we could also remove all the committed transactions from the log file.

TDDD37/TDDB77 August 2011

Exercise 5. Data structures (2 + 1 + 1 = 4 p):

We have a file with 30000 records. Each record is 5 bytes long. The records have two key attributes X and Y. The file is not sorted ordered. The database uses a block size of B=150 bytes and unspanning allocation.

1. How many blocks do you have to access to find a record with a given value for Y
    a. when you do not use any index ?
    b. when you use a secondary index ? Each index record is 3 bytes long.
    c. when you use a multilevel index ? Each index record is 3 bytes long.
2. Explain why the multilevel index is the fastest.

Recall that $\log_2 2^x = x$.

Solution:

1.a) Since Y is not an ordering field, we have to run a linear search to find a given value of Y. Therefore, the average number of data blocks to access in order to find the given value is ceiling(b/2), where b is the number of blocks to store the data file. To compute b, compute first the blocking factor (i.e. number of data records per block):

bf = floor(150/5) = 30 data records/block

Note that we used the floor function because of unspanned allocation (i.e. records must be store completely in the block). Then,

b = ceiling(30000/30) = 1000 data blocks

Then, the number of data blocks to access is on average ceiling(b/2) = 500, and 1000 in the worst case.

1.b) Since Y is not an ordering field, the index is a secondary index. Note that Y is a key field. Then, the index has as many entries as data records, i.e. 30000. Since the blocking factor for the index is

bf = floor(150/3) = 50 index records/block

the number of blocks to store the index is

b_i = ceiling(30000/50) = 600 blocks

Since the index file is ordered by definition, we can run a binary search to find the appropriate index block. Then, we need at most ceiling(log_2 600) + 1 block accesses to get to the data, i.e. 11 accesses. Note that the "+1" comes from the access to read the data whereas the other 10 come from the search for the pointer to the data block in the index file.

1.c) To construct a multilevel index, we construct an index of the index file constructed in 1.b. Since this index file is ordered according to a key value, the index we are going to construct is a primary index and, thus, it has as many entries as blocks in the index in 1.b, i.e. 600. Since

the blocking factor for the index is 50, we need 12 blocks to store the new index. This new index is called a level 2 index, whereas the one built in 1.b is a level 1 index. Now, let us build a level 3 index, i.e. a index of the level 2 index file. Again, the index we are going to construct is a primary index and, thus, it has as many entries as blocks in the level 2 index, i.e. 12. Since the blocking factor for the index is 50, we need only one block to store the new index. And we are done. If the level 3 index would have taken more than one block, then we would have constructed a level 4 index, level 5, etc. until we reach a level with a single block.

To access a data entry, we need to read the block of the level 3 index, find the pointer to the appropriate block of the level 2 index and read this block, find the pointer to the appropriate block of the level 1 index and read this block, find the pointer to the appropriate data block and read this block. So, we need 4 block accesses.

2) The multilevel index is the fastest because there is no search (finding the pointer to the next block to read only involves the block that we have read before to main memory and, thus, it does not imply accessing new blocks from disk).

Exercise 6. Transactions and concurrency control (2 + 1 + 1 = 4 p):

1. Give a transaction schedule that is NOT serializable. Show that the schedule is not serializable.

2. Use the two phase locking protocol in the transactions in your schedule. What for do we use the two phase locking protocol ?

3. Show that there can be deadlocks with the two phase locking protocol. You do not need to use the same transactions as in your schedule above, i.e. you can make up new ones.

Solution:

1)

T1	**T2**
read(x)	
	read(x)
	x:=x+1
	write(x)
x:=x+1	
write(x)	

The only conflicts in the schedule above are between T1's r(x) and T2's w(x), between T1's w(x) and T2's w(x), and between T1's w(x) and T2's r(x). Then, the conflict graph has a directed cycle and, thus, the schedule above is not serializable. This means that the schedule above is not (conflict) equivalent to T1 followed by T2 or to T2 followed by T1.

2)

T1	**T2**
write lock(x)	write lock(x)
read(x)	read(x)

x:=x+1	x:=x+1
write(x)	write(x)
unlock(x)	unlock(x)

The two phase locking protocol guarantees that any schedule involving these transactions is serializable.

3)

T1	**T2**
write lock(x)	write lock(y)
write lock(y)	write lock(x)
…	…

A deadlock will occur if T1 takes the lock on x but T2 takes the lock on y before T1. Then, each transaction will wait for the other transaction to release the corresponding lock. This waiting will go on forever.

Exercise 7. Database recovery (3 p):

Use the three recovery methods we have seen in the course in the system log below. Show all the operations (in the right order) that must be done to recover from the crash.

Part of system log:

Start-transaction T1
Write-item T1, A, 10, 20
Start-transaction T2
Write-item T1, B, 10, 20
Write-item T2, C, 10, 20
Write-item T2, C, 20, 40
Commit T1
Commit T2
Checkpoint
Start-transaction T3
Write-item T3, D, 20, 30
Commit T3
→system crash

Solution:

Note that T3 is the only transaction that has committed after the last checkpoint, and that there is no transaction that is active when the system crashes.

Deferred update (NO-UNDO/REDO): Redo T3's operations in the order they appear in the system log.

Write-item T3, D, 20, 30

Immediate update I (UNDO/NO-REDO): Do nothing.

Immediate update II (UNDO/REDO): Redo T3's operations in the order they appear in the system log.

Write-item T3, D, 20, 30

TDDD37/TDDB77 October 2011

Practical part (15 p)

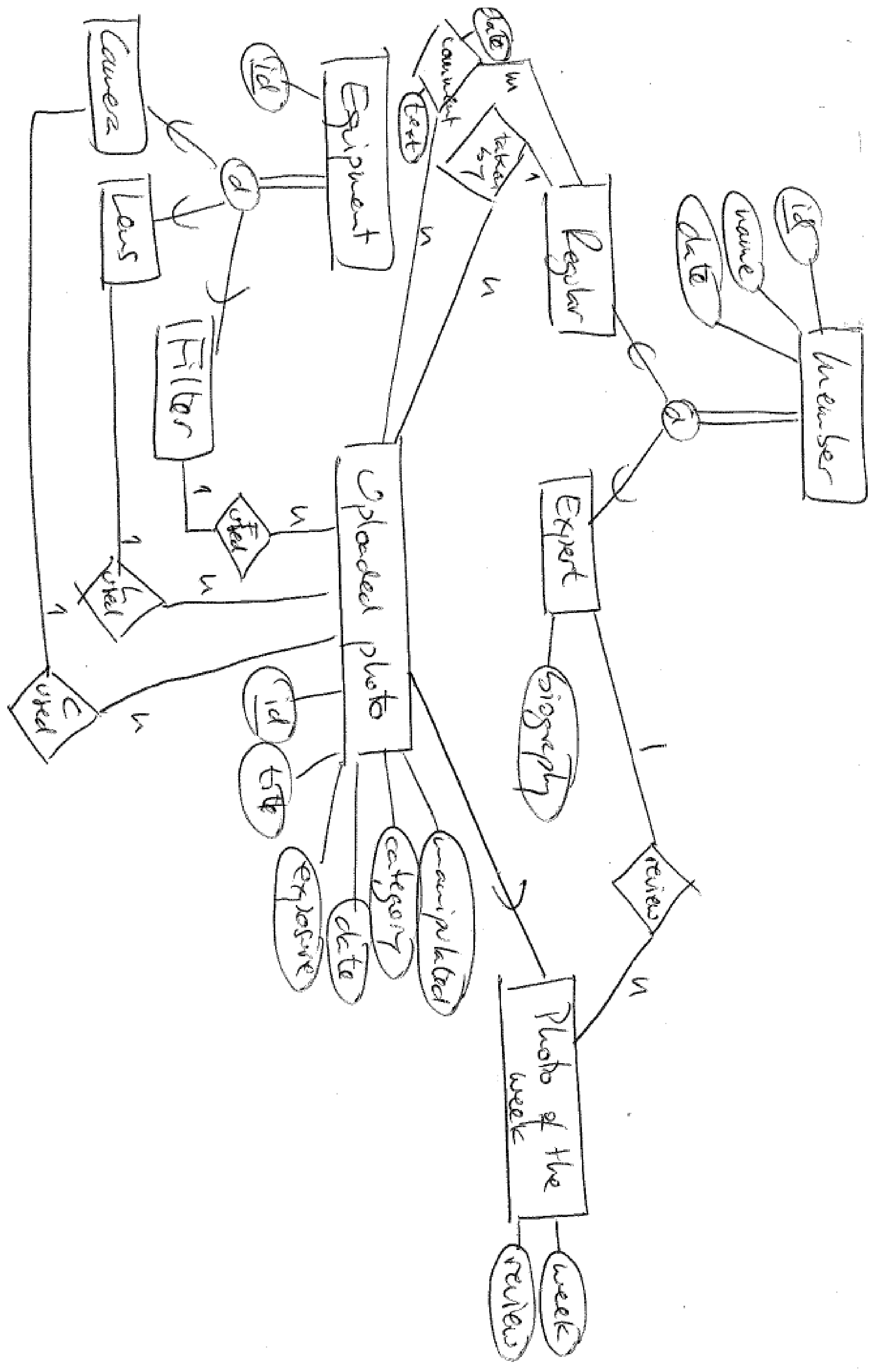
Question 1. Data modeling with EER diagram (5 p):

Read the whole exercise before starting.

An online photo forum has much information which it must keep track of. Each of its members has a unique id, user name, registration date and the photography equipments he/she is using. Photography equipments include different cameras, lens and filter. Members can upload their own photos for sharing. Each photo has its unique id, title, exposure date, post date, category, and whether it is manipulated. The forum keeps track of the member who uploads a photo, and the photography equipments that were used for shooting the photo. Every member can comment on photos. Each comment has its post date. The forum invites professional photographers, such as journalists, to become their expert members. Each expert member has a biography. Every week one photo is selected as "*the photo of week*". A photo can be "*the photo of week*" only once. The forum invites one expert member to write a short review on "*the photo of week*". For "*the photo of week*" the forum stores the week number, the review and the reviewer.

Draw an EER diagram for the photo forum for the data described above.

A solution:


Camera
Lens
Filter
Equipment
id
Uploaded photo
Regular
Member
id
name
date
Expert
biography
review
Photo of the week
week
review
comment
text
taken by
used
exposure
date
category
manipulated

Question 2. SQL (1 + 1.5 + 1.5 + 2 = 6 p):

Study the following relations describing suppliers of chips, customers and orders:

Supplier:

Id	**Name**
1	Estrella
2	OLW
3	Eldorado
…	…

Customer:

Id	**Name**
1	ICA
2	Hemköp
3	Willys
…	…

Order:

Supplier	**Customer**	**Date**	**Amount (kkr)**
1	1	16/10/2011	100
1	2	08/10/2011	95
2	1	08/10/2011	105
3	2	16/10/2011	100
3	3	08/10/2011	80
…	...	…	…

Supplier is a foreign key referring to Supplier (id)
Customer is a foreign key referring to Custom (id)

Write SQL queries for the following:

1. List the names of Suppliers that have at least one order with ICA.
2. List the names of Suppliers that do not have any order with ICA.
3. List the names of Suppliers that have more than 10 orders with ICA.
4. List the pairs of Suppliers that do not have orders with the same customer. For the data above we expect the following answer:

Supplier1	**Supplier2**
Eldorado	OLW
OLW	Eldorado

A solution:

1)

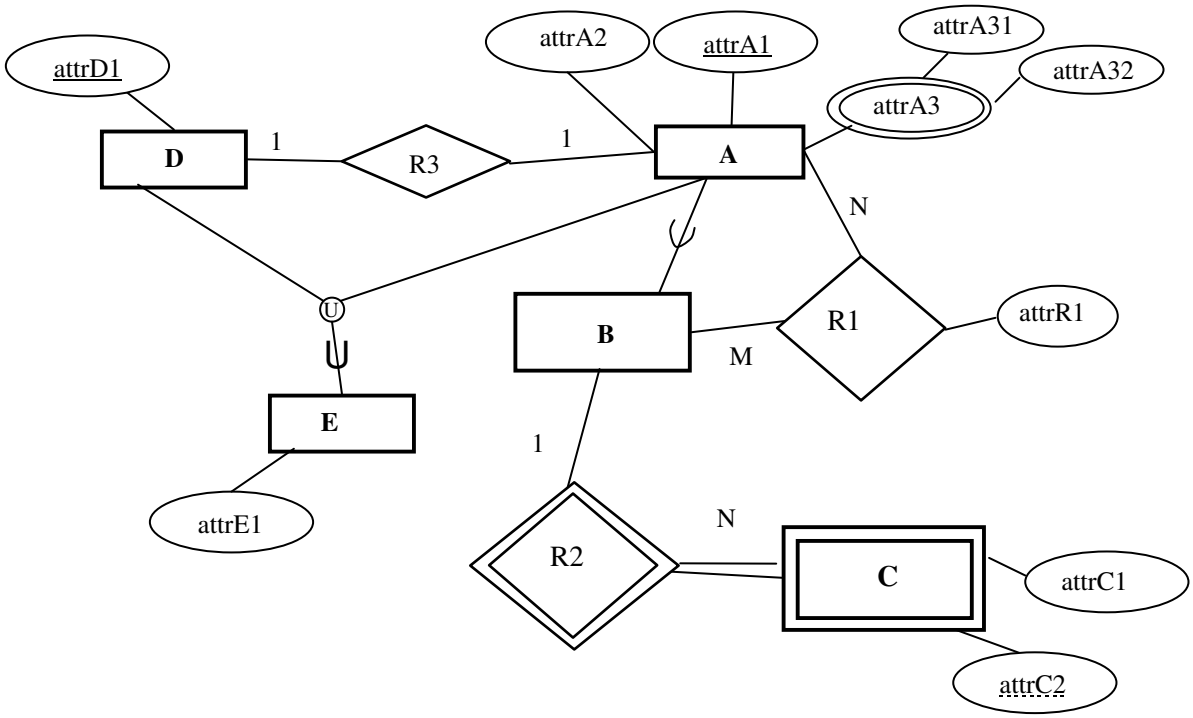
```
select name
from supplier
where id in (select order.supplier
             from order, customer
             where order.customer=customer.id and customer.name='ICA');
```

2)
```
select name
from supplier
where id not in (select order.supplier
                 from order, customer
                 where order.customer=customer.id and customer.name='ICA');
```

3)
```
select supplier.name,count(supplier.name)
from supplier,order,customer
where supplier.id=order.supplier and order.customer=customer.id and customer.name='ICA'
group by supplier.name
having count(supplier.name)>10;
```

4)
```
select s1.name,s2.name
from supplier s1, supplier s2
where (s1.id,s2.id) not in (select o1.supplier,o2.supplier
                            from order o1, order o2
                            where o1.customer=o2.customer);
```

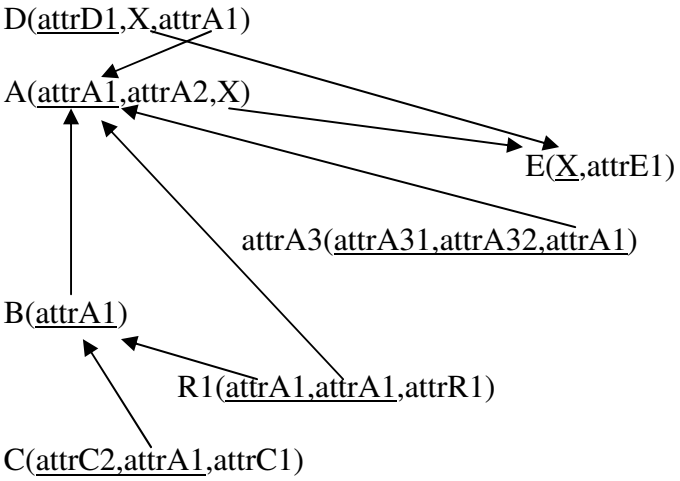
Question 3. Translation of EER to relational schema (4 p):



Translate the EER diagram above into a relational model (you have to follow the algorithm seen in the course). Mark the primary keys with solid underlining and the foreign keys with

dotted underlining and an arrow from the foreign key to the attribute(s) pointed by the foreign key.

Solution:



Theoretical part (18 points)

Question 4. Normalization (1 + 2 + 1 = 4 p):

Given the relation R(A, B, C, D, E, F, G, H) with functional dependencies {AB→CDEFGH, CD→B, D→EFGH, E→FGH, FG→E, G→H},

1. Find all the candidate keys of R. Use the inference rules in the course to reach your conclusion. Do not use more than one rule in each derivation step.
2. Normalize R to 2NF. Explain the process step by step.
3. Why do we normalize relations ?

Solution:

1) The functional dependency AB→CDEFGH implies that AB is a candidate key. We now show that ACD is also a candidate key.

AB→CDEFGH implies AB→EFGH by decomposition
AB→EFGH and CD→B imply ACD→EFGH by pseudotransitive
CD→B implies ACD→AB by augmentation and, thus, ACD→B by decomposition
ACD→B and ACD→EFGH imply ACD→BEFGH by union

2) The solution to 1) implies that A, B, C and D are prime and E, F, G and H non-prime. Since D→EFGH violates the definition of 2NF, we have to split the original table into

R(A,B,C,D) with AB and ACD as candidate keys and functional dependencies {AB→CD, CD→B}

R2(D,E,F,G,H) with D as candidate key and functional dependencies {D→EFGH, E→FGH, FG→E, G→H}

Now, R and R2 satisfy the definition of 2NF.

3) We normalize to avoid repetition, which causes waste of space and update anomalies.

Question 5. Data structures (2 + 2 + 1 = 5 p):

We have a file with 30000 records. Each record is 5 bytes long. The records have two key attributes X and Y. The file is ordered on X. The database uses a block size of B=100 bytes and unspanning allocation. Each index record is 4 bytes long.

1. Our goal is to perform at most 11 block accesses to find a record with a given value for X. Do we need to create a primary index or a static multilevel index to reach our goal ?
2. Our goal is to perform at most 11 block accesses to find a record with a given value for Y. Do we need to create a secondary index or a static multilevel index to reach our goal ?
3. What are B-trees and B+-trees ?

Recall that $\log_2 2^x = x$. That is, $\log_2 1 = 0$, $\log_2 2 = 1$, $\log_2 4 = 2$, $\log_2 8 = 3$, $\log_2 16 = 4$, $\log_2 32 = 5$, $\log_2 64 = 6$, $\log_2 128 = 7$, $\log_2 256 = 8$, $\log_2 512 = 9$, $\log_2 1024 = 10$, $\log_2 2048 = 11$, etc.

Solution:

1) Since X is an ordering field, we can run a binary search to find a given value of X. Therefore, the average number of data blocks to access in order to find the given value is ceiling(log_2 b), where b is the number of blocks to store the data file. To compute b, compute first the blocking factor (i.e. number of data records per block):

bf = floor(10/5) = 20 data records/block

Note that we used the floor function because of unspanned allocation (i.e. records must be store completely in the block). Then,

b = ceiling(30000/20) = 1500 data blocks

Then, the number of data blocks to access is on average ceiling(log_2 b) = 11. Then, we do not need any index to reach our goal, it suffices with the primary access method.

2) Since Y is not an ordering field, we have to run a linear search to find a given value of Y. Therefore, the average number of data blocks to access in order to find the given value is ceiling(b/2) where, as seen before, b=1500. Since this is not good enough, we need an index. Since Y is not an ordering field, the index is a secondary index. Note that Y is a key field. Then, the index has as many entries as data records, i.e. 30000. Since the blocking factor for the index is

bf = floor(100/4) = 25 index records/block

the number of blocks to store the index is

b_i = ceiling(30000/25) = 1200 blocks

Since the index file is ordered by definition, we can run a binary search to find the appropriate index block. Then, we need at most ceiling(log_2 1200) + 1 block accesses to get to the data, i.e. 12 accesses. Note that the "+1" comes from the access to read the data whereas the other 10 come from the search for the pointer to the data block in the index file. Since this is not good enough, we need a static multilevel index.

To construct a static multilevel index, we construct an index of the secondary index file constructed above. Since this index file is ordered according to a key value, the index we are going to construct is a primary index and, thus, it has as many entries as blocks in the index above, i.e. 1200. Since the blocking factor for the index is 25, we need 48 blocks to store the new index. This new index is called a level 2 index, whereas the one built above is a level 1 index. Now, let us build a level 3 index, i.e. a index of the level 2 index file. Again, the index we are going to construct is a primary index and, thus, it has as many entries as blocks in the level 2 index, i.e. 48. Since the blocking factor for the index is 25, we need only two blocks to store the new index. Now, let us build a level 4 index, i.e. a index of the level 3 index file. Again, the index we are going to construct is a primary index and, thus, it has as many entries as blocks in the level 3 index, i.e. 2. Since the blocking factor for the index is 25, we need only one block to store the new index. And we are done.

To access a data entry, we need to read the block of the level 4 index, find the pointer to the appropriate block of the level 3 index and read this block, find the pointer to the appropriate block of the level 2 index and read this block, find the pointer to the appropriate block of the level 1 index and read this block, find the pointer to the appropriate data block and read this block. So, we need 5 block accesses.

A somehow unorthodox but valid solution is to stop the multilevel construction after level 2, since this implies ceiling(log_2 48) + 2 = 8 block access, i.e. ceiling(log_2 48) to find the appropriate pointer in the index (which is an ordered file and, thus, admits a binary search) plus one access to read the appropriate block of the level 1 index plus another block access to read the appropriate data block.

3) B-trees and B+-trees are data structures that can be used to store dynamic multilevel indexes. In such indexes, the reorganization of the index is minimal (since the nodes are typically filled at 70 %) when the data file changes due to the addition, removal or modification of some records.

Question 6. Transactions and concurrency control (2 + 1 = 3 p):

1. Is the following transaction schedule serializable? Motivate your answer.

T1	T2	T3
	read(x)	
	x:=x+1	

```
                write(x)


        read(x)
        x:=x+1
        write(x)


                read(y)
                y:=y+1
                write(y)


                                read(y)
                                y:=y+1
                                write(y)

        read(y)
        y:=y+1
        write(y)
```

2. Does this schedule permit the two-phase locking protocol, i.e. can you apply the protocol so that the transactions interleave as in the schedule above ? Justify your answer.

Solution:

1) The conflicts in the schedule above are between T1's r(x) and T2's w(x), between T1's w(x) and T2's w(x), and between T1's w(x) and T2's r(x), between T1's r(y) and T2's w(y), between T1's w(y) and T2's w(y), and between T1's w(y) and T2's r(y), between T1's r(y) and T3's w(y), between T1's w(y) and T3's w(y), and between T1's w(y) and T3's r(y), between T2's r(y) and T3's w(y), between T2's w(y) and T3's w(y), and between T2's w(y) and T3's r(y). Then, the conflict graph looks like T2->T1, T3->T1, and T2->T3 and, thus, it has no directed cycle, which means that the schedule above is serializable, which means that the schedule above is (conflict) equivalent to T2 followed by T3 followed by T1.

2)

```
T1          T2              T3

        lock-write(x)
        lock-write(y)
        read(x)
        x:=x+1
        write(x)
        unlock(x)
```

```
lock-write(x)
read(x)
x:=x+1
write(x)

        read(y)
        y:=y+1
        write(y)
        unlock(y)

                        lock-write(y)
                        read(y)
                        y:=y+1
                        write(y)
                        unlock(y)
lock-write(y)
read(y)
y:=y+1
write(y)
unlock(x)
unlock(y)
```

Question 7. Database recovery (3 p):

Apply the three recovery methods seen in the course to the system log below. Show all operations that are performed during the recovery. In the correct order!

Part of system log:

```
Start-transaction T1
Write-item T1, A, 5, 6
Start-transaction T2
Write-item T2, B, 2, 4
Write-item T2, B, 4, 7
Commit T1
Start-transaction T3
Write-item T3, A, 6, 8
Write-item T3, A, 8, 10
Write-item T2, B, 7, 2
Checkpoint
Start-transaction T4
Commit T2
Write-item T4, C, 1, 2
```

→system crash

Solution:

Note that T2 is the only transaction that has committed after the last checkpoint, and that T3 and T4 are still active when the system crashes.

Deferred update (NO-UNDO/REDO): Redo T2's operations in the order they appear in the system log.

Write-item T2, B, ?, 4
Write-item T2, B, ?, 7
Write-item T2, B, ?, 2

Immediate update I (UNDO/NO-REDO): Undo T3's and T4's operations in the reverse order they appear in the system log.

Undo Write-item T4, C, 1, 2 = Write-item T4, C, ?, 1
Undo Write-item T3, A, 8, 10 = Write-item T3, A, ?, 8
Undo Write-item T3, A, 6, 8 = Write-item T3, A, ?, 6

Immediate update II (UNDO/REDO): First, undo T3's and T4's operations in the reverse order they appear in the system log and then, redo T2's operations in the order they appear in the system log.

Undo Write-item T4, C, 1, 2 = Write-item T4, C, ?, 1
Undo Write-item T3, A, 8, 10 = Write-item T3, A, ?, 8
Undo Write-item T3, A, 6, 8 = Write-item T3, A, ?, 6
Write-item T2, B, ?, 4
Write-item T2, B, ?, 7
Write-item T2, B, ?, 2

Question 8. Optimization (1 + 1 + 1 = 3 p):

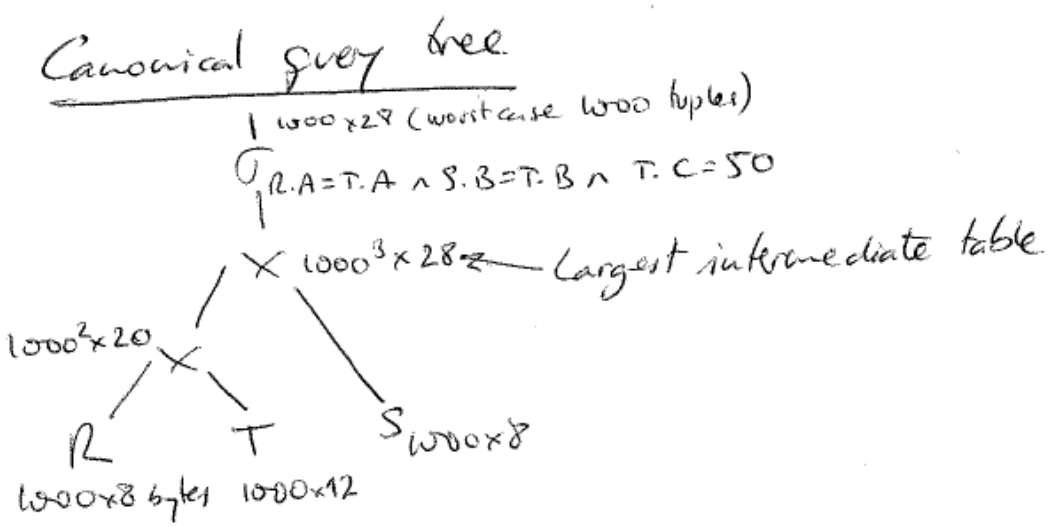
1. Let R(A,X), S(B, Y), and T(A, B, C) be three tables with the underlined attributes as keys. Optimize the following MySQL query:

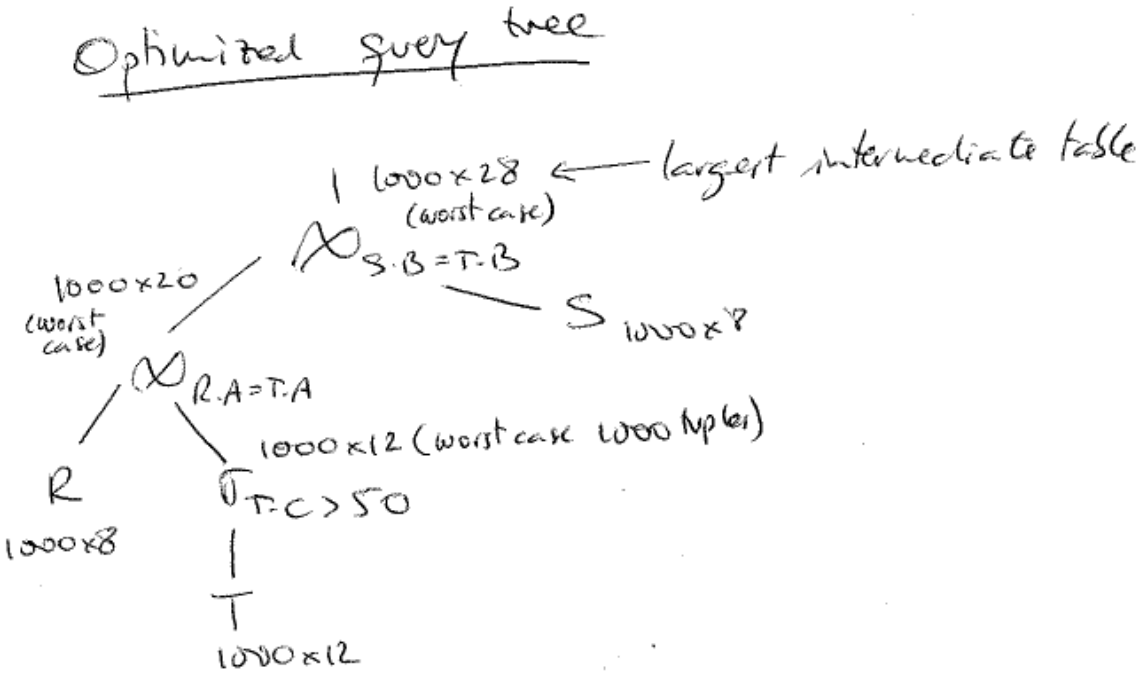
   ```
   SELECT *
   FROM R, S, T
   WHERE R.A = T.A AND S.B = T.B AND T.C > 50;
   ```

2. Assume that the tables do not contain any NULL value. Assume also that each table contains 1000 tuples and that each attribute is of size 4 byte. Show that the optimized query tree is more efficient than the canonical query tree.
3. Why does query optimization replace a selection followed by a Cartesian product with a join operation ?

Solution:

1 and 2)





3) A join implies less space for the intermediate tables since it can be implemented efficiently, i.e. without creating the Cartesian product result first.

TDDD37/TDDB77/TDDD12 January 2012

Practical part (14 points)

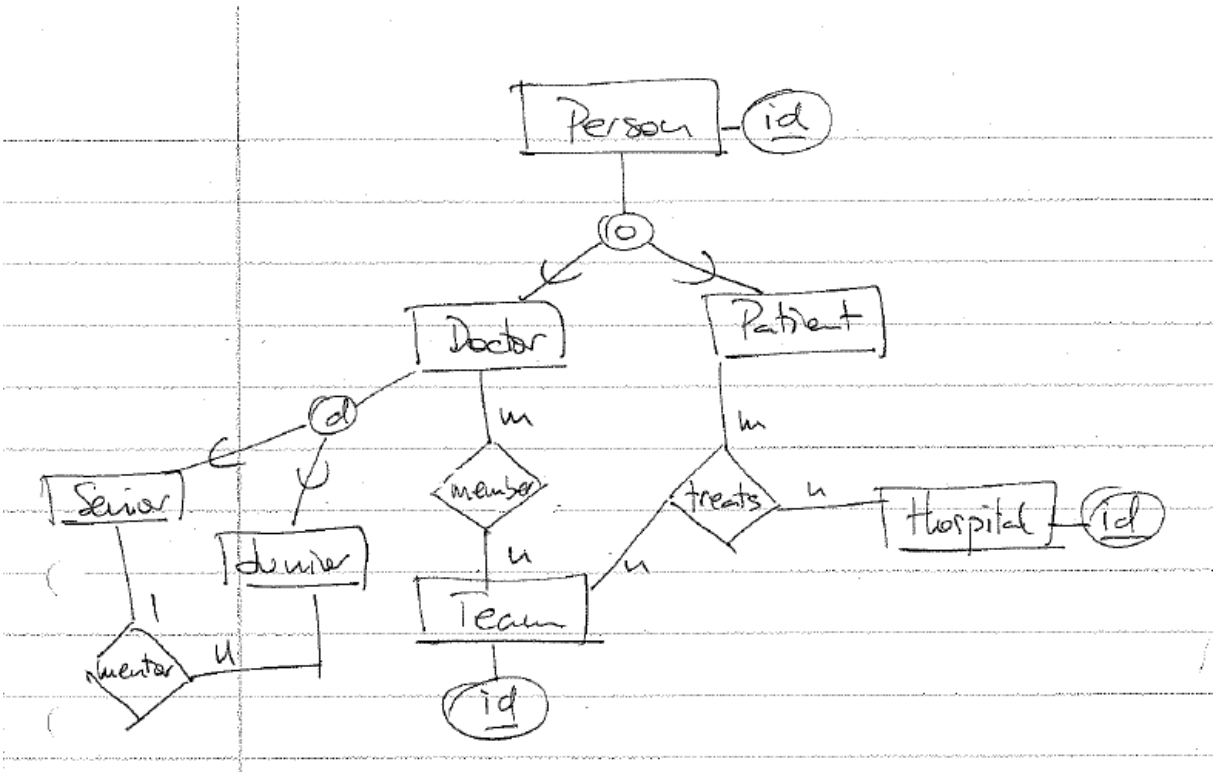
Question 1. Data modeling with EER diagram (5 p):

Read the whole exercise before you start.

We want to create a database to store information about the Swedish National Health System. Specifically, we want to store information about doctors, hospitals and patients. We also want to store information about which doctor or team of doctors treated which patient in which hospital. Notice that not only single doctors but also teams of doctors can treat patients. Furthermore, we assume doctors can work in several hospitals and that patients can be treated in several hospitals. Notice that doctors can also be patients. We also want to distinguish between senior and junior doctors. Every junior doctor has a senior doctor as mentor. We want to store who is the mentor of whom.

Your task is to build an EER model that they can use for creating the database. Clearly write down your choices and assumptions in case you find that something in the information above is not clear.

A solution:



Question 2. SQL (1 + 2 + 2 = 5 p):

Team

id	name	Arena	founded

Player

id	name	position	age

Playing

id	team	player	year	points

team is a foreign key reference to *id* in the table Team.
player is a foreign key reference to *id* in the table Player.

points is the total number of points a player scored for a team in a year.
Note that a player can play for more than one team in the same year.

1. List the names of the teams founded before 1980.

2. List the name of each player that has played for more than one team during the year 2011.

3. For each player, show the first year she played and the total number of points she scored in that year.

A solution:

```
1. select name
from team
where founded < 1980;

2. select name
from player
where 1 < (select count(*)
                from playing
                where playing.player=player.id and year=2011);
```

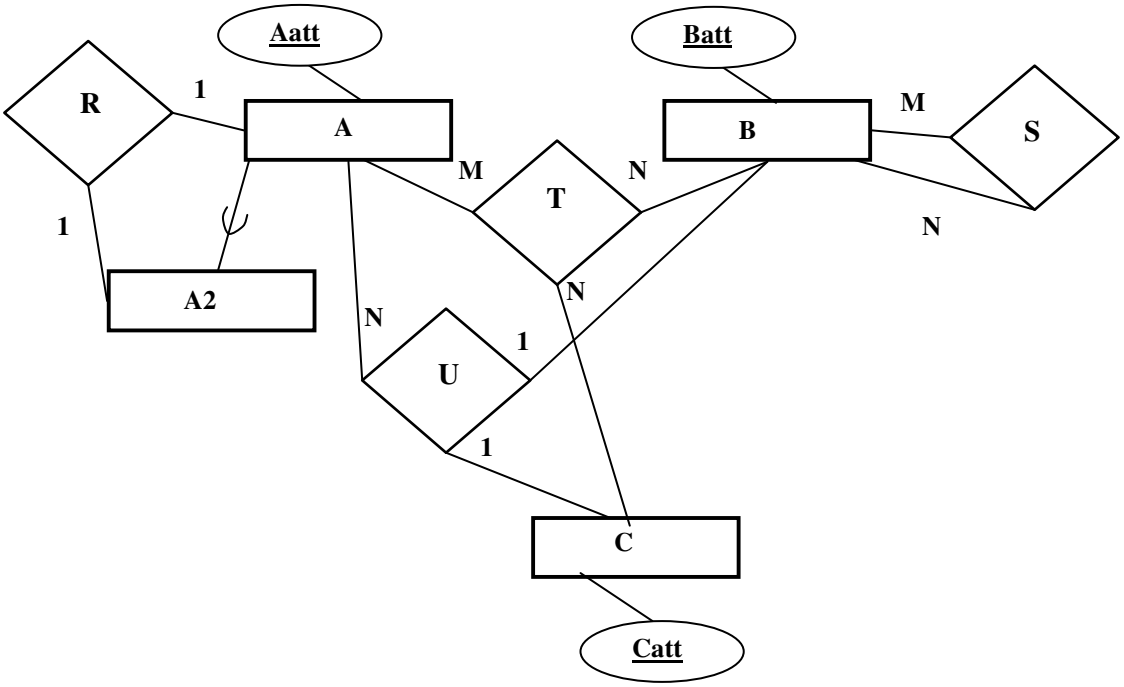
or

```
select name, count(*)
from player, playing
where player.id=playing.player and year=2011
group by name
having count(*)>1;
```

or

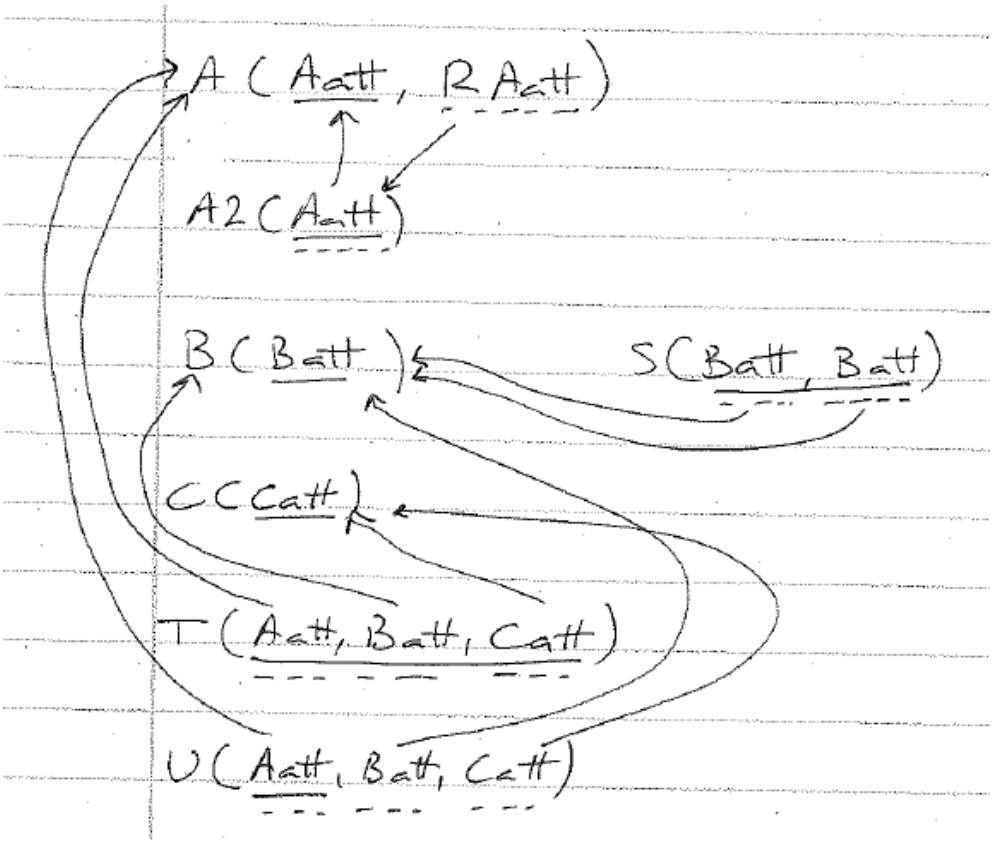
```
select distinct name
from player, playing p1, playing p2
where player.id=p1.player and player.id=p2.player and p1.year=2011 and p2.year=2011 and
p1.team!=p2.team;

3.   select name, year, sum(points)
from player, playing
where player.id=playing.player and year in (select min(year)
                                                    from playing
                                                    where playing.player=player.id)
group by name, year;
```

Question 3. Translation EER to relational schema (4 p):



Translate the EER diagram to a relational schema (use the algorithm seen in the course).

Solution:



Theoretical part (16 points)

Question 4. Normalization (2 p):

Normalize (1NF→2NF→3NF→BCNF) the relation R(A, B, C, D, E, F, G, H) with functional dependencies F={ABC→DEFGH, D→CEF, EF→GH}. *Explain your solution step by step.*

Solution:

The functional dependency ABC→DEFGH implies that ABC is a candidate key. We now show that ABD is also a candidate key.

ABC→DEFGH implies ABC→EFGH by decomposition
D→CEF implies D→C by decomposition
ABC→EFGH and D→C imply ABD→EFGH by pseudotransitive
D→C implies ABD→C by augmentation
ABD→C and ABD→EFGH imply ABD→CEFGH by union

The candidate keys above imply that A, B, C and D are prime and E, F, G and H non-prime. Since D→EFGH violates the definition of 2NF, we have to split the original table into

R1(A,B,C,D) with ABC and ABD as candidate keys and functional dependencies {ABC→D, D→C}

R2(D,E,F,G,H) with D as candidate key and functional dependencies {D→EFGH, EF→GH}.

Now, R1 and R2 satisfy the definition of 2NF. However, R2 does not satisfy the definition of 3NF due to EF→GH. Then, we have to split R2 into

R21(D,E,F) with D as candidate key and functional dependencies {D→EF}

R22(E,F,G,H) with EF as candidate key and functional dependencies {EF→GH}.

Now, R1, R21 and R22 satisfy the definition of 3NF. However, R1 does not satisfy the definition of BCNF due to D→C. Then, we have to split R1 into

R11(A,B,D) with candidate key A,B,D.

R12(D,C) with candidate key D.

Question 5. Data structures (2 + 2 + 1 = 5 p):

We have a file with 30000 records. Each record is 5 bytes long. The records have two key attributes X and Y. The file is ordered on X. The database uses a block size of B=100 bytes and unspanning allocation. Each index record is 4 bytes long.

1. Calculate the average number of block access needed to find a record with a given value for X when using the primary access method and when using a single level index.

2. Calculate the average number of block access needed to find a record with a given value for Y when using the primary access method and when using a single level index.
3. Explain why you obtain different results in the question 2 depending on whether you use an index or not.

Recall that $\log_2 2^x = x$. That is, $\log_2 1 = 0$, $\log_2 2 = 1$, $\log_2 4 = 2$, $\log_2 8 = 3$, $\log_2 16 = 4$, $\log_2 32 = 5$, $\log_2 64 = 6$, $\log_2 128 = 7$, $\log_2 256 = 8$, $\log_2 512 = 9$, $\log_2 1024 = 10$, $\log_2 2048 = 11$, etc.

Solution:

1) Since X is an ordering field, we can run a binary search to find a given value of X. Therefore, the average number of data blocks to access in order to find the given value is ceiling(log_2 b), where b is the number of blocks to store the data file. To compute b, compute first the blocking factor (i.e. number of data records per block):

bf = floor(10/5) = 20 data records/block

Note that we used the floor function because of unspanned allocation (i.e. records must be store completely in the block). Then,

b = ceiling(30000/20) = 1500 data blocks

Then, the number of data blocks to access is on average ceiling(log_2 b) = 11.

Since X is an ordering key field, the index is a primary index. Then, the index has as many entries as data blocks, i.e. 1500. Since the blocking factor for the index is

bf = floor(100/4) = 25 index records/block

the number of blocks to store the index is

b_i = ceiling(1500/25) = 60 blocks

Since the index file is ordered by definition, we can run a binary search to find the appropriate index block. Then, we need at most ceiling(log_2 60) + 1 block accesses to get to the data, i.e. 7 accesses. Note that the "+1" comes from the access to read the data whereas the other 7 come from the search for the pointer to the data block in the index file.

2) Since Y is not an ordering field, we have to run a linear search to find a given value of Y. Therefore, the average number of data blocks to access in order to find the given value is ceiling(b/2) where, as seen before, b=1500.

Since Y is not an ordering field, the index is a secondary index. Note that Y is a key field. Then, the index has as many entries as data records, i.e. 30000. Since the blocking factor for the index is

bf = floor(100/4) = 25 index records/block

the number of blocks to store the index is

b_i = ceiling(30000/25) = 1200 blocks

Since the index file is ordered by definition, we can run a binary search to find the appropriate index block. Then, we need at most ceiling(log_2 1200) + 1 block accesses to get to the data, i.e. 12 accesses. Note that the "+1" comes from the access to read the data whereas the other 10 come from the search for the pointer to the data block in the index file.

3) Y is not an ordering field, which means that its primary access method implies a linear search. If a secondary index is used, then we can use a binary search (which is faster than a linear search). Moreover, the linear search is run on the data file whereas the binary search is run on the index file, which takes fewer blocks than the data file (because the index records are smaller than the data records).

Question 6. Transactions and concurrency control (2 + 1 = 3 p):

1. Is the following transaction schedule serializable? Motivate your answer.

```
T1          T2          T3
read(x)
x:=x+1
write(x)
                        read(x)
                        x:=x+1
                        write(x)
            read(x)
            x:=x+1
            write(x)
read(y)
y:=y+1
write(y)
            read(y)
            y:=y+1
            write(y)
```

2. Does this schedule permit the two-phase locking protocol, i.e. can you apply the protocol so that the transactions interleave as in the schedule above ? Justify your answer.

Solution:

1) The conflicts in the schedule above are between T1's r(x) and T2's w(x), between T1's w(x) and T2's w(x), and between T1's w(x) and T2's r(x), between T1's r(y) and T2's w(y), between T1's w(y) and T2's w(y), and between T1's w(y) and T2's r(y), between T1's r(x) and T3's w(x), between T1's w(x) and T3's w(x), and between T1's w(x) and T3's r(x), between T2's r(x) and T3's w(x), between T2's w(x) and T3's w(x), and between T2's w(x) and T3's r(x). Then, the conflict graph looks like T1->T2, T1->T3, and T3->T2 and, thus, it

has no directed cycle, which means that the schedule above is serializable, which means that the schedule above is (conflict) equivalent to T1 followed by T3 followed by T2.

2)

```
T1          T2          T3

lock-write(x)
lock-write(y)
read(x)
x:=x+1
write(x)
unlock(x)
                        lock-write(x)
                        read(x)
                         x:=x+1
                         write(x)
                         unlock(x)
            lock-write(x)
            read(x)
             x:=x+1
             write(x)
read(y)
y:=y+1
write(y)
unlock(y)
             lock-write(y)
             read(y)
             y:=y+1
             write(y)
             unlock(x)
             unlock(y)
```

Question 7. Database recovery (3 p):

Apply the three recovery methods seen in the course to the system log below. Show all operations that are performed during the recovery. In the correct order!

Part of system log:
Start-transaction T2
Write-item T2, B, 3, 4
Start-transaction T3
Write-item T3, A, 7, 8
Write-item T3, A, 8, 1
Commit T2
Start-transaction T4
Write-item T4, B, 4, 5
Write-item T4, B, 5, 10
Write-item T3, A, 1, 5
Checkpoint

Start-transaction T1
Commit T3
Write-item T1, C, 8, 9
→system crash

Solution:

Note that T3 is the only transaction that has committed after the last checkpoint, and that T1 and T4 are still active when the system crashes.

Deferred update (NO-UNDO/REDO): Redo T3's operations in the order they appear in the system log.

Write-item T3, A, ?, 8
Write-item T3, A, ?, 1
Write-item T3, A, ?, 8

Immediate update I (UNDO/NO-REDO): Undo T1's and T4's operations in the reverse order they appear in the system log.

Undo Write-item T1, C, 8, 9 = Write-item T1, C, ?, 8
Undo Write-item T4, B, 5, 10 = Write-item T4, B, ?, 5
Undo Write-item T4, B, 4, 5 = Write-item T4, B, ?, 4

Immediate update II (UNDO/REDO): First, undo T1's and T4's operations in the reverse order they appear in the system log and then, redo T3's operations in the order they appear in the system log.

Undo Write-item T1, C, 8, 9 = Write-item T1, C, ?, 8
Undo Write-item T4, B, 5, 10 = Write-item T4, B, ?, 5
Undo Write-item T4, B, 4, 5 = Write-item T4, B, ?, 4
Write-item T3, A, ?, 8
Write-item T3, A, ?, 1
Write-item T3, A, ?, 8

Question 8. Optimization (1 + 1 + 1 = 3 p):

1. Let R(A,B), S(B, C), T(C,D), P(D,A) be four tables with the underlined attributes as keys. Optimize the following MySQL query:

```
SELECT *
FROM R, S, T, P
WHERE R.B = S.B AND S.C = T.C AND T.D = P.D AND P.A = R.A;
```

2. Assume that the tables do not contain any NULL value. Assume also that each table contains 1000 tuples and that each attribute is of size 4 byte. Show that the optimized query tree is more efficient than the canonical query tree.

3. Why does query optimization replace a selection followed by a Cartesian product with a join operation ?

Solution: Similar to previous years' solutions.

TDDD12/TDDD46/TDDB77 May 2012

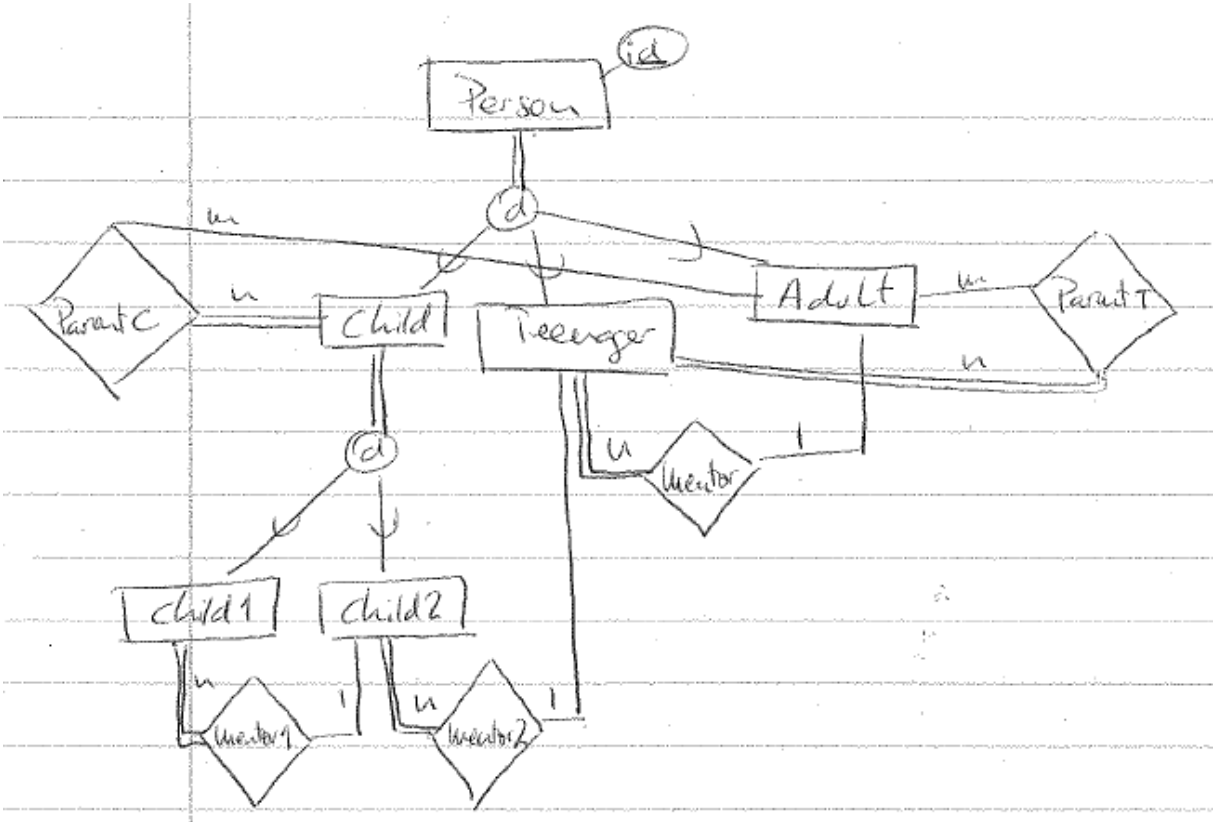
Question 1. Data modeling with EER diagram (5 p):

We want to create a database to store information about a new mentorship program. The program states that every child and teenager will have a mentor according to the following rules:

- Every teenager has a single mentor.
- The mentor of a teenager is always an adult.
- Every child has a single mentor.
- The mentor of a child is either a teenager or a child whose mentor is a teenager.

For every child and teenager, we want to store who his/her mentor is. We also want to store who his/her parents are. We assume that his/her parents are always adults, although the parents do not need to be his/her mentor. Your task is to build an EER model that we can use for creating the database. **You should not use more than one regular entity type, otherwise we will withdraw one point (however, you can use as many subclasses as you want)**. Clearly write down your choices and assumptions in case you find that something in the information above is not clear.

A solution:



Question 2. SQL (1 + 2 + 2 = 5 p):

Consider the following database schema

Country(Name, Code, Capital, Area, Population)
Organization(Name, Abbreviation, Established)
IsMember(Organization, Country, Type)

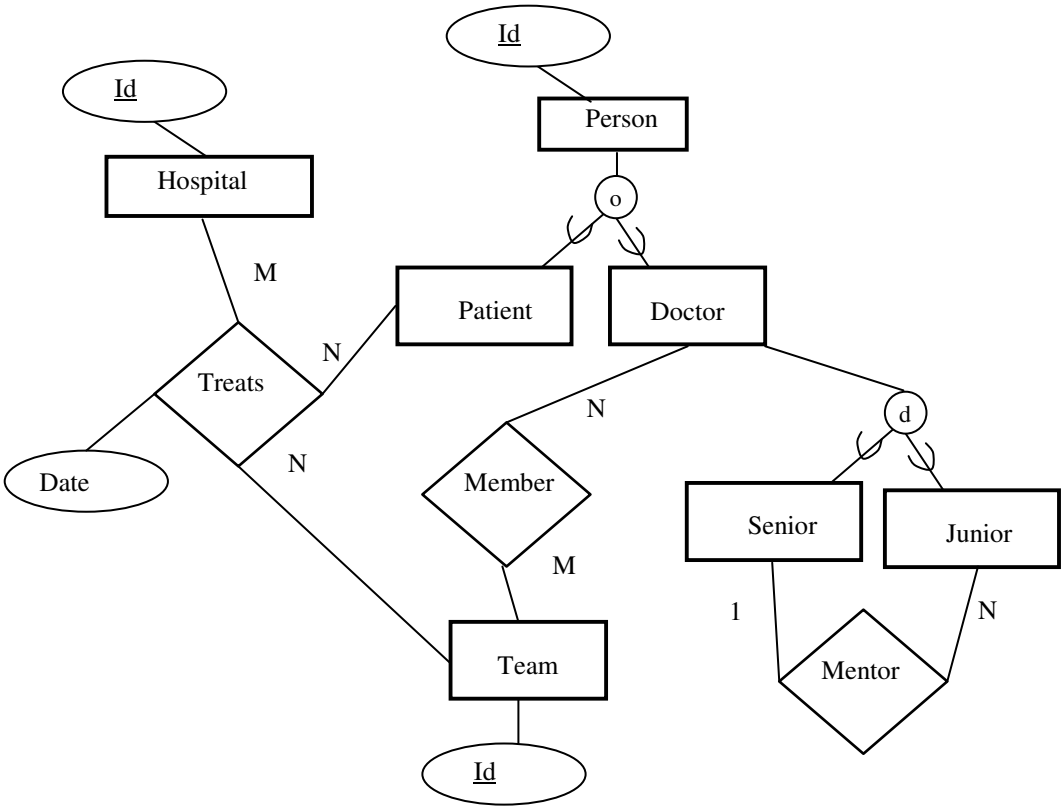
The attribute Organization in the table IsMember is a foreign key reference to the attribute Abbrevation in the table Organization.
The attribute Country in the table IsMember is a foreign key reference to the attribute Code in the table Country.

1. List all the organization names which Sweden ('SWE') is a member of.
2. For each organization, compute the sum of the population of its member countries. List the organization in descending order of this sum.
3. List the country names which are members of at least one organization, which Sweden is also a member of.

A solution:

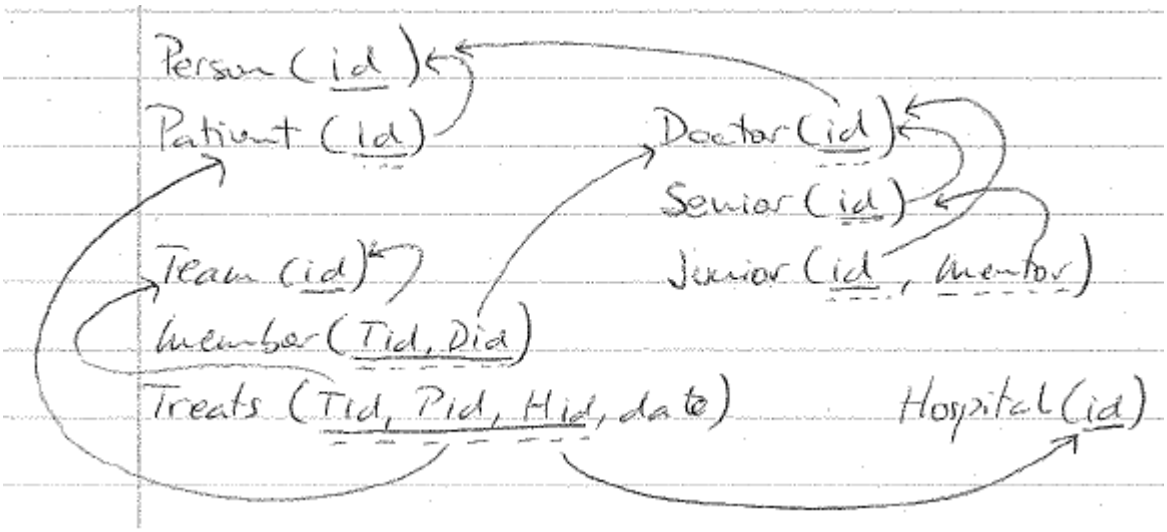
```
1.  SELECT Organization.Name
FROM Organization, IsMember
WHERE IsMmeber.Country = 'SWE' and Organization.Abbreviation =
IsMember.Organization;

2.  SELECT IsMember.Organization, sum(Population) SP
FROM Country, IsMember
WHERE Country.Code = IsMember.Country
GROUP BY IsMember.Organization
ORDER BY SP DESC;

3.  SELECT DISTINCT Country.Name
FROM IsMember, Country
WHERE IsMember.Country = Country.Code and IsMember.Organization IN
            (SELECT IM.Organization
            FROM IsMember IM
            WHERE IM.Country = 'SWE'
            );
```

Question 3. Translation of EER diagram into relational schema (5 p):



Translate the EER diagram to a relational schema (use the algorithm seen in the course).

A solution:



Question 4. Normalization (3 p):

Normalize (1NF→2NF→3NF→BCNF) the relation R(A, B, C, D, E, F, G) with functional dependencies F={ABC→DEFG, A→FG, B→D, DE→BC, F→G}. *Explain your solution step by step.*

Solution:

The functional dependency ABC→DEFG implies that ABC is a candidate key. We now show that ADE is also a candidate key.

ABC→DEFG implies ABC→FG by decomposition
ABC→FG and DE→BC imply ADE→FG by pseudotransitive
DE→BC implies ADE→BC by augmentation+decomposition
ADE→FG and ADE→BC imply ADE→BCFG by union

We now show that ABE is also a candidate key.

ADE→BCFG implies ADE→CFG by decomposition
ADE→CFG and B→D imply ABE→CFG by pseudotransitive
B→D implies ABE→D by augmentation+decomposition
ABE→CFG and ABE→D imply ABE→CDFG by union

The candidate keys above imply that A, B, C, D and E are prime and F and G non-prime. Since A→FG violates the definition of 2NF, we have to split the original table into

R1(A,B,C,D,E) with ABC and ADE as candidate keys and functional dependencies {ABC→DE, B→D, DE→BC}

R2(A,F,G) with A as candidate key and functional dependencies {F→G}.

Now, R1 and R2 satisfy the definition of 2NF. However, R2 does not satisfy the definition of 3NF due to F→G. Then, we have to split R2 into

R21(A,F) with A as candidate key and functional dependencies {A→F}

R22(F,G) with F as candidate key and functional dependencies {F→G}.

Now, R1, R21 and R22 satisfy the definition of 3NF. However, R1 does not satisfy the definition of BCNF due to B→D. Then, we have to split R1 into

R11(A,B,C,E) with candidate keys ABC and ABE and functional dependencies {ABC→E, BE→C}.

R12(B,D) with candidate key B and functional dependencies {B→D}.

R11 does not yet satisfy BCNF due to BE→C. Then, we have to split R11 into

R111(A,B,E) with candidate key ABE.

R112(B,E,C) with candidate key BE and functional dependency {BE→C}.

Question 5. Data structures (2 + 3 = 5 p):

We have a file with 1000000 records. Each record is 10 bytes long. The records have two key

attributes X and Y. The file is ordered on X. The database uses a block size of B=1000 bytes and unspanning allocation. Each index record is 4 bytes long.

1. Calculate the average number of block access needed to find a record with a given value for X when using (1) the primary access method and (2) a single level index.
2. Calculate the average number of block access needed to find a record with a given value for Y when using (1) the primary access method, (2) a single level index and (3) static multi-level index.

Recall that $\log_2 2^x = x$. That is, $\log_2 1 = 0$, $\log_2 2 = 1$, $\log_2 4 = 2$, $\log_2 8 = 3$, $\log_2 16 = 4$, $\log_2 32 = 5$, $\log_2 64 = 6$, $\log_2 128 = 7$, $\log_2 256 = 8$, $\log_2 512 = 9$, $\log_2 1024 = 10$, $\log_2 2048 = 11$, etc.

Solution:

1. Since X is an ordering field, we can run a binary search to find a given value of X. Therefore, the average number of data blocks to access in order to find the given value is ceiling(log_2 b), where b is the number of blocks to store the data file. To compute b, compute first the blocking factor (i.e. number of data records per block):

bf = floor(1000/10) = 100 data records/block

Note that we used the floor function because of unspanned allocation (i.e. records must be store completely in the block). Then,

b = ceiling(1000000/100) = 10000 data blocks

Then, the number of data blocks to access is on average ceiling(log_2 b) = 14.

Since X is an ordering key field, the index is a primary index. Then, the index has as many entries as data blocks, i.e. 10000. Since the blocking factor for the index is

bf = floor(1000/4) = 250 index records/block

the number of blocks to store the index is

b_i = ceiling(10000/250) = 40 blocks

Since the index file is ordered by definition, we can run a binary search to find the appropriate index block. Then, we need at most ceiling(log_2 40) + 1 block accesses to get to the data, i.e. 7 accesses. Note that the +1 comes from the access to read the data whereas the other 6 come from the search for the pointer to the data block in the index file.

2. Since Y is an not the ordering field, we have to run linear search to find a given value of Y. Therefore, the average number of data blocks to access in order to find the given value is ceiling(b/2), where b is computed above, i.e. 10000. Thus the number of block access is 5000.

Since Y is not an ordering field, the index is a secondary index. Note that Y is a key field.

Then, the index has as many entries as data records, i.e. 1000000. Since the blocking factor for the index is 250 as computed above, the number of blocks to store the index is

b_i = ceiling(1000000/250) = 4000 blocks

Since the index file is ordered by definition, we can run a binary search to find the appropriate index block. Then, we need at most ceiling(log_2 4000) + 1 block accesses to get to the data, i.e. 13 accesses. Note that the +1 comes from the access to read the data whereas the other 12 come from the search for the pointer to the data block in the index file.

To construct a static multilevel index, we construct an index of the secondary index file constructed above. Since this index file is ordered according to a key value, the index we are going to construct is a primary index and, thus, it has as many entries as blocks in the index above, i.e. 4000. Since the blocking factor for the index is 250, we need 16 blocks to store the new index. This new index is called a level 2 index, whereas the one built above is a level 1 index. Now, let us build a level 3 index, i.e. a index of the level 2 index file.
Again, the index we are going to construct is a primary index and, thus, it has as many entries as blocks in the level 2 index, i.e. 16. Since the blocking factor for the index is 250, we need only one block to store the new index. And we are done.

To access a data entry, we need to read the block of the level 3 index, find the pointer to the appropriate block of the level 2 index and read this block, find the pointer to the appropriate block of the level 1 index and read this block, find the pointer to the appropriate data block and read this block. So, we need 4 block accesses.

Question 6. Transactions and concurrency control (1 + 1 + 1 = 3 p):

Consider the following schedule:

T1	T2	T3
write(x)		
		write(x)
	write(x)	
	write(y)	
write(z)		
		write(y)

1. The schedule is not serializable. Justify this claim.
2. Is it possible to obtain a serializable schedule by deleting only one operation from the above schedule? How many possibilities are there?
3. If the answer from 2 is yes, consider all the possible new serializable schedules. Which one(s) from them permit(s) the two-phase locking protocol, i.e. can you apply the protocol so that the transactions interleave as in the schedule above ? Justify your answer.

Solution:

1. The schedule is not serializable because there is cycle in the conflict graph between T2 and T3, where T3 is ahead of T2 because of writing x and T2 is ahead of T3 because of writing y.

2. There are four possibilities:

(1) delete write(x) from T2

(2) delete write(y) from T2

(3) delete write(x) from T3

(4) delete write(y) from T3

All of the four result in a serializable schedule.

3. Now consider whether these schedules permit 2PL.

(1) The new schedule is as follows:

T1 T2 T3

write(x)

write(x)

write(y)

write(z)

write(y)

The following assignments of locks permit 2PL, namely in each transaction, all the locks are before the first unlock, and by issuing a lock to an object, this object is available, i.e. it is not locked by any other transaction.

T1 T2 T3

lock(x)

lock(z)

write(x)

unlock(x)

lock(x)

write(x)

lock(y)

write(y)

unlock(y)

write(z)

unlock(z)

lock(y)

write(y)

unlock(x)

unlock(y)

(2) The new schedule also permits 2PL.

T1 T2 T3

lock(x)

lock(z)

write(x)

unlock(x)

lock(x)

write(x)

lock(y)

unlock(x)

lock(x)

write(x)

unlock(x)

write(z)

unlock(z)

write(y)

unlock(y)

(3) The new schedule also permits 2PL.

T1 T2 T3

lock(x)

lock(z)

write(x)

unlock(x)

lock(x)

lock(y)

write(x)

write(y)

unlock(x)

unlock(y)

write(z)

unlock(z)

lock(y)

write(y)

unlock(y)

(4) The new schedule also permits 2PL.

T1 T2 T3

lock(x)

lock(z)

write(x)

unlock(x)

lock(x)

write(x)

unlock(x)

lock(x)

lock(y)

write(x)

write(y)

unlock(x)

unlock(y)

write(z)

unlock(z)

To sum up, all the four reduced schedule permit 2PL.

Question 7. Database recovery (2 + 1 + 1 = 4 p):

1. Apply the two immediate update recovery methods seen in the course to the system log below. Show all operations that are performed during the recovery. In the correct order!

   Part of system log:
   Start-transaction T2
   Write-item T2, B, 3, 4
   Start-transaction T3
   Write-item T3, A, 7, 8
   Checkpoint
   Write-item T3, A, 8, 1
   Commit T2
   Checkpoint
   Write-item T3, A, 1, 5
   Start-transaction T4
   Write-item T4, B, 4, 5
   Write-item T4, B, 5, 10
   Commit T3
   Checkpoint
   Start-transaction T1
   Write-item T1, C, 8, 9
   Commit T4
   →system crash

2. The cache can buffer up to three disk blocks. A transaction modifies on average four disk blocks. Which database recovery method do you recommend to use ?
3. Which database recovery strategy does not need that the checkpoints are stored in the system log ?

Solution:

1. Note that T4 is the only transaction that has committed after the last checkpoint, and that T1 is still active when the system crashes.

Immediate update I (UNDO/NO-REDO): Undo T1's operations in the reverse order they appear in the system log.

Undo Write-item T1, C, 8, 9 = Write-item T1, C, ?, 8

Immediate update II (UNDO/REDO): First, undo T1's operations in the reverse order they appear in the system log and then, redo T4's operations in the order they appear in the system log.

Undo Write-item T1, C, 8, 9 = Write-item T1, C, ?, 8
Write-item T4, B, ?, 5
Write-item T4, B, ?, 10

2. Any immediate update version. The reason is that, when running immediate update, the system can write to disk at any point. When running deferred update, the system cannot write to disk before the transaction commits. This is a problem because we do not have enough cache memory to store all the modifications performed by a transaction (i.e. 3 blocks in cache versus 4 blocks modified).

3. Immediate update version 1(NOREDO/UNDO), since any committed transactions' changes are in disk for sure after the transaction has committed.

TDDD12/TDDD46/TDDB77 May 2013

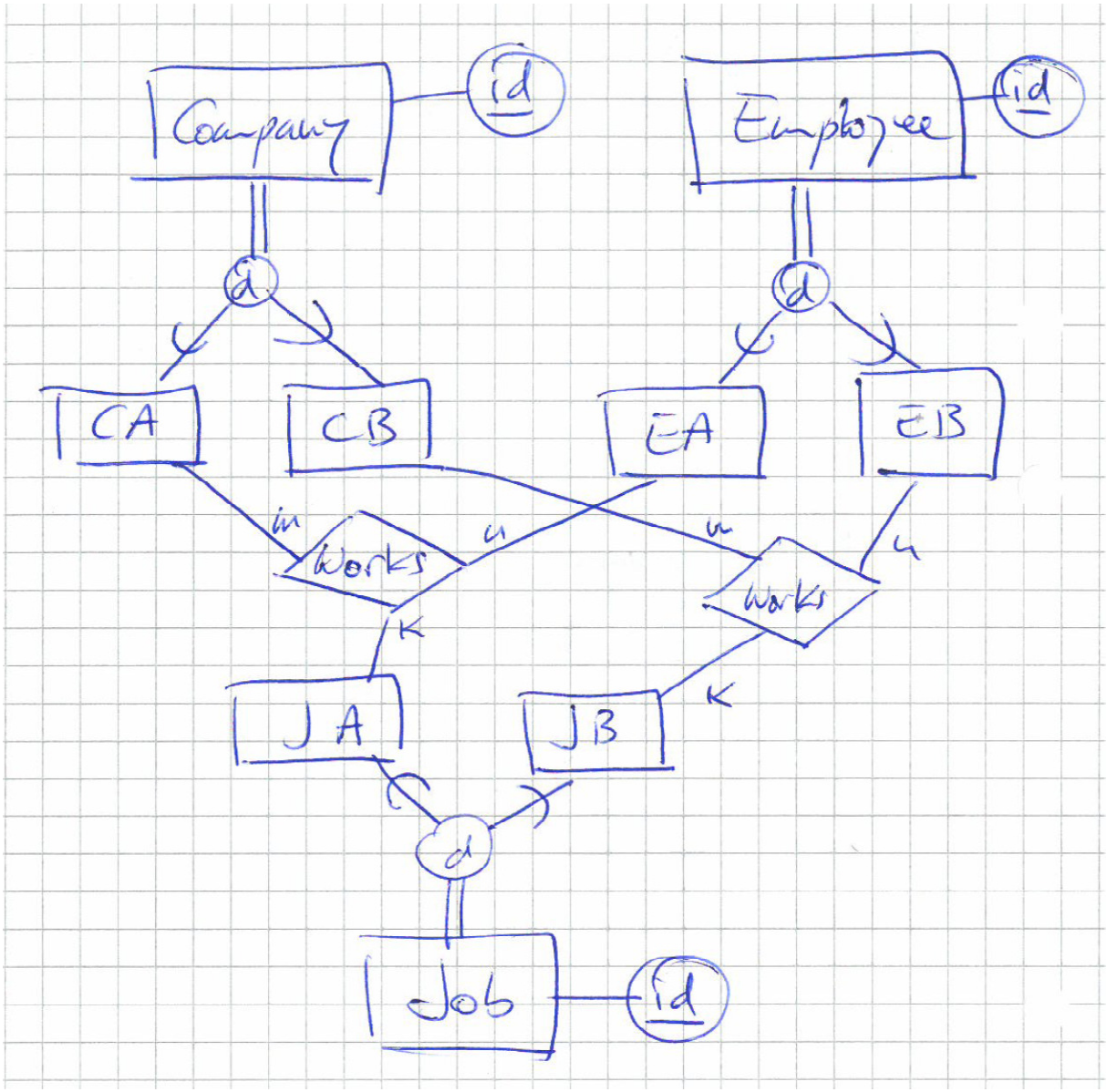
Question 1. Data modeling with EER diagram (4 + 1 = 5 p):

1. We want to create a database to store information about some companies, their employees and their jobs. Each company, employee and job can be classified as of type A or B. Companies of type A only have employees of type A, who only work in jobs of type A. Likewise, companies of type B only have employees of types B, who only work in jobs of type B. We want to create a database to store who works for which company in which job. Draw an EER diagram for such a database. Clearly write down your choices and assumptions in case you find that something in the information above is not clear.

2. What is the difference between an entity and an entity type ? And between a relationship and a relationship type ?

Solution:

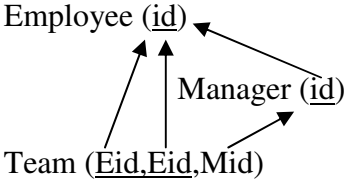
1.

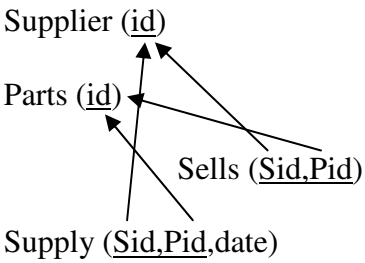


2. An entity represents an object of the miniworld. An entity type represents all the objects of the miniworld sharing certain characteristics. For instance, my car is an entity whereas the entity type Car represents all the cars in the miniworld, including mine. Likewise, a relationship relates two or more entities whereas a relationship type relates two or more entity types.

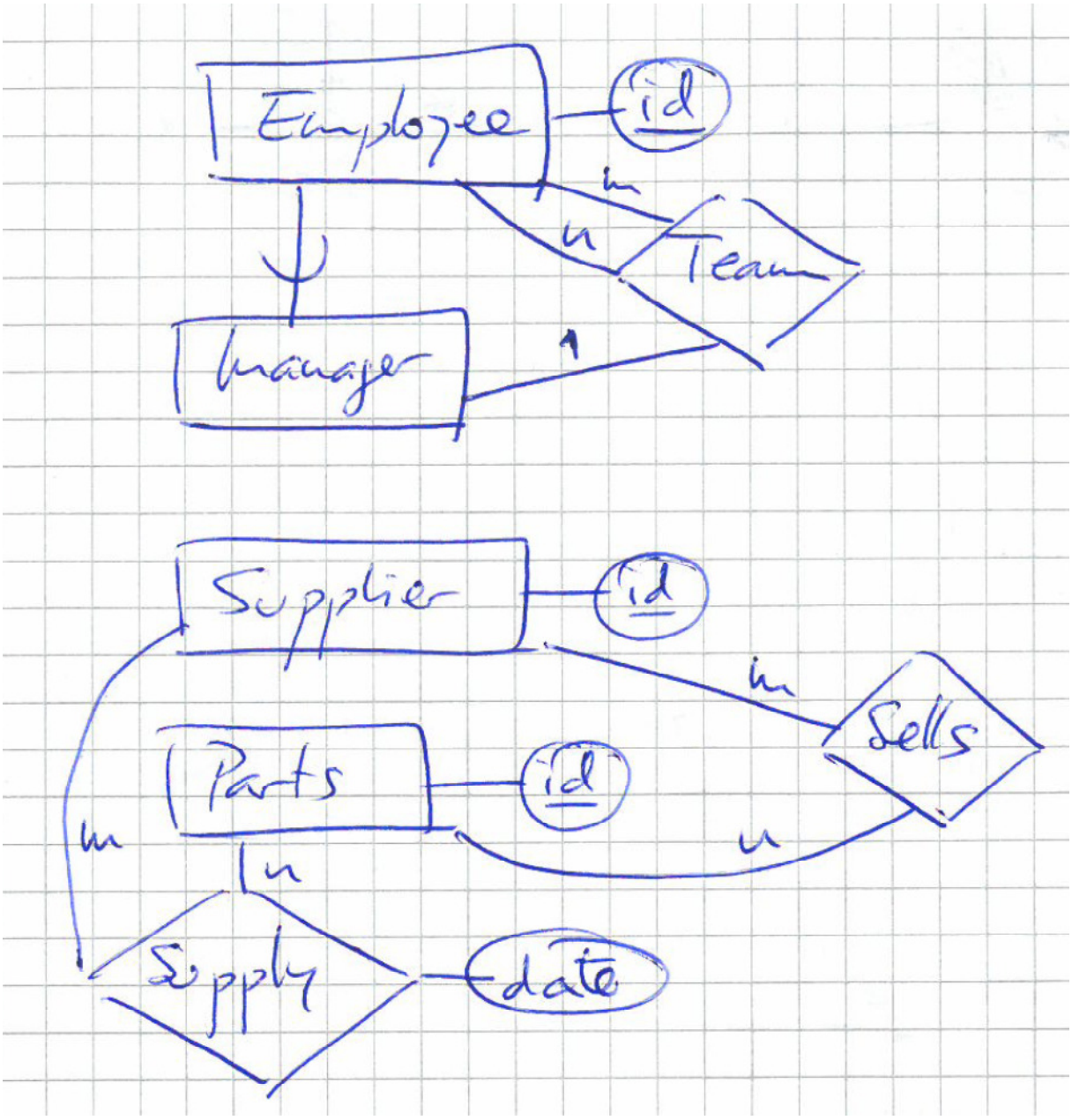
Question 3. EER diagram and relational schema (5 p):

Draw an EER diagram that, when translated using the algorithm seen in the course, may result in the following relational model.





Solution:



Question 4. Normalization (2 + 1 = 3 p):

1. Normalize (1NF→2NF→3NF→BCNF) the relation R(A, B, C, D) with functional dependencies {AB→CD, C→B, D→C}. Explain your solution step by step. Bear in mind that a relation can have several candidate keys.

2. Give an example of a relation that is in BCNF but not in 3NF. If this is not possible, explain why.

Solution:

1. The functional dependency AB→CD implies that AB is a candidate key. We now show that AC and AD are also candidate keys.

   AB→CD and C→B imply AC→CD by pseudotransitive and, thus, AC→D by decomposition
   C→B implies AC→AB by augmentation and, thus, AC→B by decomposition
   AC→B and AC→D imply AC→BD by union.

   The proof for AD is similar.

   The candidate keys above imply that A, B, C and D are prime attributes. Then, the relation is in 3NF. The relation is not in BCNF because C→B violates the definition. Then, we have to split the original table into

   R1(A,C,D) with AC and AD as candidate keys and functional dependencies {AC→D, D→C}, and

   R2(C,B) with C as candidate key and functional dependencies {C→B}.

   Now, R2 satisfies the definition of BCNF. However, R1 does not satisfy it due to D→C. Then, we have to split R1 into

   R11(A,D) with AD as candidate key and no functional dependencies, and

   R12(D,C) with D as candidate key and functional dependencies {D→C}.

   Now, R11, R12 and R2 satisfy the definition of BCNF.

2. It is not possible because, by definition, if a relation is in BCNF then it is also in 3NF.

Question 7. Database recovery (3 + 1 = 4 p):

1. Apply the recovery method for the three update methods seen in the course to the system log below. Show all operations (in the correct order) that are performed during the recovery.

   Part of system log:
   Start-transaction T1
   Start-transaction T2
   Start-transaction T3
   Start-transaction T4
   Start-transaction T5
   Write-item T1, B, 3, 4
   Commit T1
   Checkpoint

Write-item T2, B, 3, 4
Commit T2
Checkpoint
Write-item T3, B, 3, 4
Commit T3
Write-item T4, B, 3, 4
→system crash

2. Assume that system crashes are rare and, thus, their influence on performance can be ignored. Assume that all the transactions always write on the same data item. Then, which update method is to be preferred and why ?

Solution:

1. Note that T3 is the only transaction that has committed after the last checkpoint, and that T4 and T5 are still active when the system crashes.

   Deferred update (NO-UNDO/REDO): Redo T3's operations in the order they appear in the system log.

   Write-item T3, B, ?, 4

   Immediate update I (UNDO/NO-REDO): Undo T4's and T5's operations in the reverse order they appear in the system log.

   Undo Write-item T4, B, 3, 4 = Write-item T4, B, ?, 3

   Immediate update II (UNDO/REDO): First, undo T4's and T5's operations in the reverse order they appear in the system log and then, redo T3's operations in the order they appear in the system log.

   Undo Write-item T4, B, 3, 4 = Write-item T4, B, ?, 3
   Write-item T3, B, ?, 4

2. Since all transactions write on the same data item, the cache is not going to be flushed. This implies that (i) in deferred update and immediate update version 2, the only time when blocks are written to disk is when a checkpoint is executed, and (ii) in immediate update version 1, the only time when blocks are written to disk is when a checkpoint or a commit is executed. Then, we prefer deferred update or immediate update version 2, because they minimize the number of block accesses.

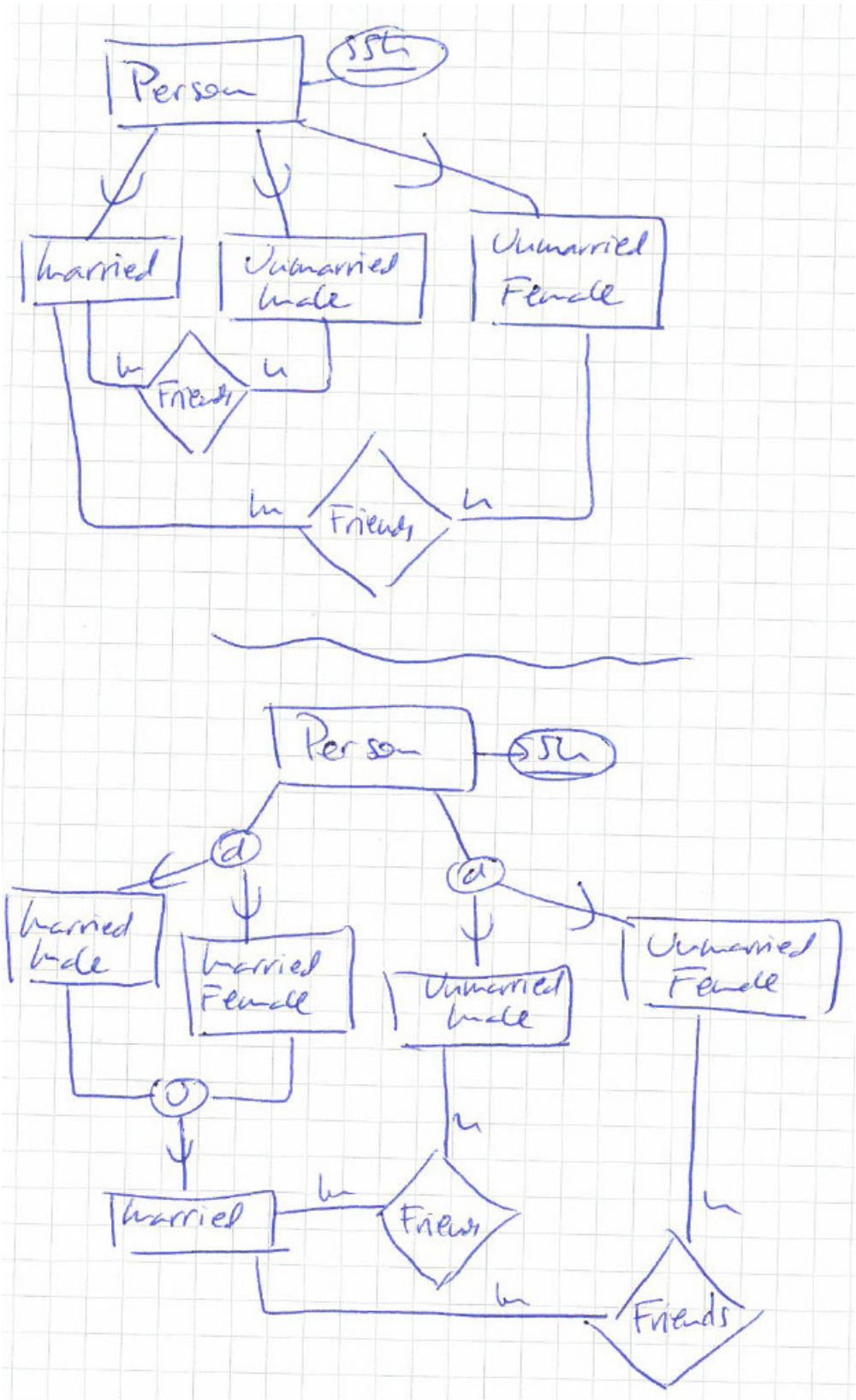
TDDD12/TDDD46/TDDB77 August 2013

Question 1. Data modeling with EER diagram (3 + 2 = 5 p):

We want to create a database to store information about the relationships of a group of people. Specifically, we want to store who is married and who is not. For each married person, we also want to store his/her unmarried male friends and his/her unmarried female friends.

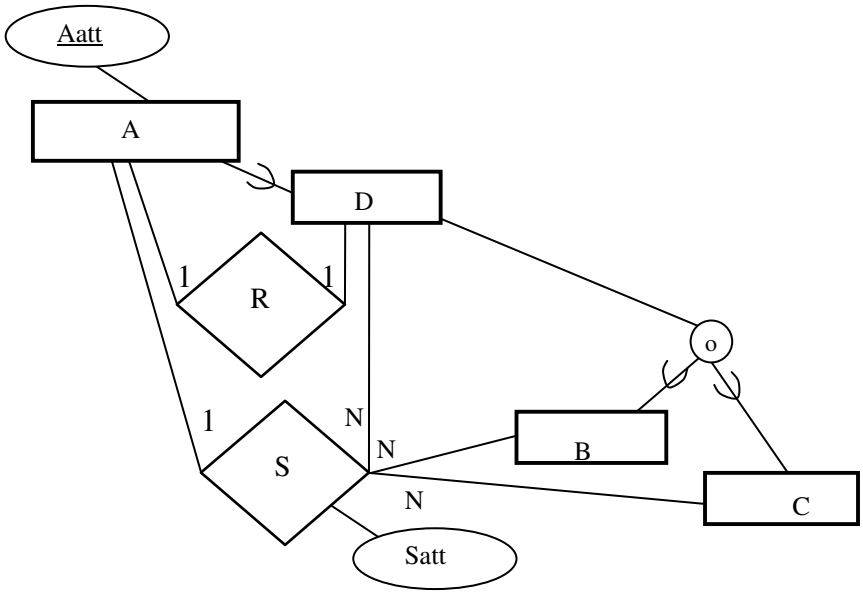
Draw two different EER diagrams for the description above. You are only allowed to use the strong entity type Person, whose entities are characterized by a unique social security number (SSN). You can use as many subclasses as you want. Clearly write down your choices and assumptions in case you find that something in the information above is not clear.

A solution:

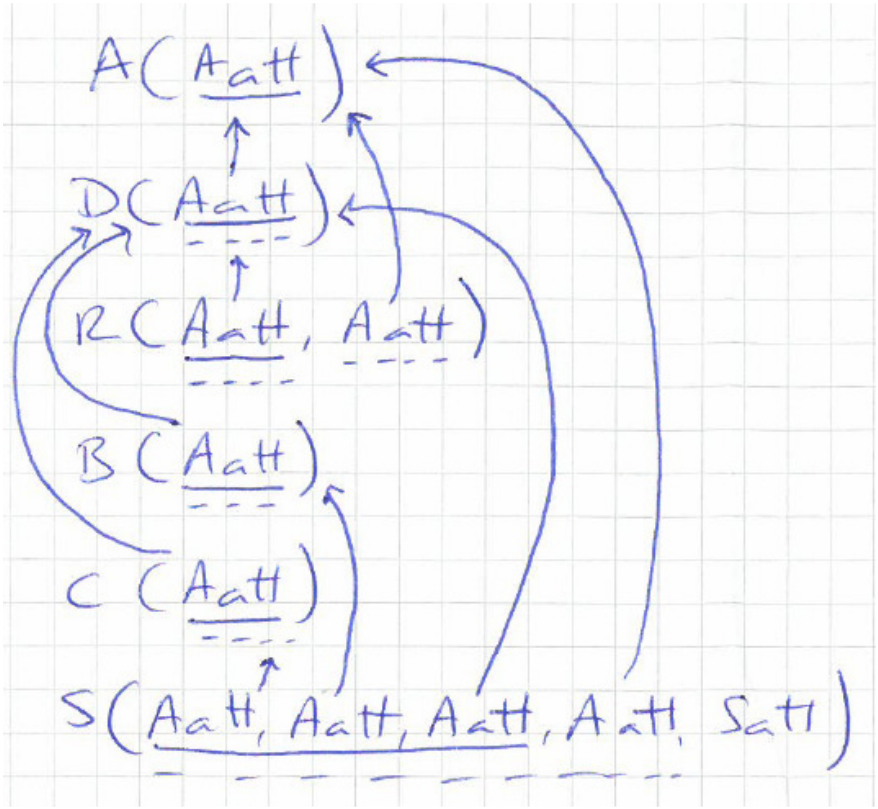


Question 3. EER diagram and relational schema (5 p):

Translate the EER diagram below into a relational schema. Use the algorithm seen in the course.



A solution:



Question 4. Normalization (2 + 1 = 3 p):

1. Normalize (1NF→2NF→3NF→BCNF) the relation R(A, B, C, D, E, F, G, H, I) with functional dependencies {CDE→F, DE→G, E→H, I→E}. Explain your solution step by step. Bear in mind that a relation can have several candidate keys.
2. Do we always have to normalize every relation ? Explain why your answer is yes or no.

Solution:

1. E→H (given) implies DE→H by augmentation and decomposition, which together with DE→G (given) implies DE→GH by union, which implies CDE→GH by augmentation and decomposition, which together with CDE→F (given) implies CDE→GHF by union, which together with I→E (given) implies CDI→GHF by pseudotransitive. Moreover, I→E implies CDI→E by augmentation and decomposition. Thus, CDI→GHFE by union. Then, ABCDI→GHFE by augmentation and decomposition. This is the only candidate key in the relation as none of the attributes in ABCDI appears in the right-hand side of any functional dependency given.

   The candidate key above implies that A, B, C, D and I are prime attributes. Then, the relation is in 1NF. The relation is not in 2NF because CDI→FGHE violates the definition. Then, we have to split the original table into

   R1(A,B,C,D,I) with ABCDI as candidate key and no functional dependencies, and

   R2(C,D,I,F,G,E,H) with CDI as candidate key and functional dependencies {CDE→F, DE→G, E→H, I→E}.

   Now, R1 satisfies the definition of 2NF, 3NF and BCNF. However, R2 does not satisfy yet the definition of 2NF due to DI→GHE. Then, we have to split R2 into

   R21(C,D,I,F) with CDI as candidate key and functional dependencies {CDI→F}, and

   R22(D,I,G,H,E) with DI as candidate key and functional dependencies {DE→G, E→H, I→E}.

   Now, R21 satisfies the definition of 2NF, 3NF and BCNF. However, R22 does not satisfy yet the definition of 2NF due to I→HE. Then, we have to split R22 into

   R221(D,I,G) with DI as candidate key and functional dependencies {DI→G}, and

   R222(I,H,E) with I as candidate key and functional dependencies {E→H, I→E}.

   Now, R221 satisfies the definition of 2NF, 3NF and BCNF. However, R222 satisfies the definition of 2NF but not that of 3NF due to E→H. Then, we have to split R222 into

   R2221(I,E) with I as candidate key and functional dependencies {I→E}, and

   R2222(E,H) with E as candidate key and functional dependencies {E→H}.

Now, all the tables are in BCNF.

2. No, we do not need to normalize every relation to BCNF. Normalization reduces redundancy and the updating anomalies at the cost of having more but smaller tables, which may imply a greater cost in time to answer some queries as more join operations may be needed.

Question 7. Database recovery (3 + 1 = 4 p):

1. Apply the three recovery methods seen in the course to the system log below. Show all operations (in the correct order) that are performed during the recovery.

   Part of system log:
   Start-transaction T2
   Write-item T2, B, 3, 4
   Start-transaction T3
   Write-item T3, A, 7, 8
   Checkpoint
   Write-item T3, A, 8, 1
   Commit T2
   Checkpoint
   Write-item T3, A, 1, 5
   Start-transaction T4
   Write-item T4, B, 4, 5
   Write-item T4, B, 5, 10
   Commit T3
   Start-transaction T1
   Write-item T1, C, 8, 9
   Commit T4
   →system crash

2. Consider the following statement: The deferred update recovery method always produces a serializable schedule. Is the statement true or false ?

Solution:

1. Note that T3 and T4 are the only transactions that have committed after the last checkpoint, and that T1 is still active when the system crashes.

   Deferred update (NO-UNDO/REDO): Redo T3's and T4's operations in the order they appear in the system log.

   Write-item T3, A, 7, 8
   Write-item T3, A, 8, 1
   Write-item T3, A, 1, 5
   Write-item T4, B, 4, 5
   Write-item T4, B, 5, 10

Immediate update I (UNDO/NO-REDO): Undo T1's operations in the reverse order they appear in the system log.

Undo Write-item T1, C, 8, 9 = Write-item T1, C, ?, 8

Immediate update II (UNDO/REDO): First, undo T1's operations in the reverse order they appear in the system log and then, redo T3's and T4's operations in the order they appear in the system log.

Undo Write-item T1, C, 8, 9 = Write-item T1, C, ?, 8
Write-item T3, A, 7, 8
Write-item T3, A, 8, 1
Write-item T3, A, 1, 5
Write-item T4, B, 4, 5
Write-item T4, B, 5, 10

2. False. Recovery and concurrency control are two independent questions.

TDDD12/TDDD81/TDDD74 May 2014

Question 2. SQL (1 + 2 + 2 = 5 p):

Consider the following database schema

Country(Name, Code, Capital, Area, Population)
Organization(Name, Abbreviation, Established)
IsMember(Organization, Country, Joined)

The attribute *Organization* in the table IsMember is a foreign key reference to *Abbreviation* in the table Organization.
The attribute *Country* in table IsMember is a foreign key reference to *Code* in the table Country.

Examples of the tuples from the above relational schema are as follows:

Country(Sweden, SWE, Stockholm, 449964, 9514000)
Organization(European Union, EU, 1952)
IsMember(EU, SWE, 1995-01-01)

1. List the name of the organization that was first established.
2. List the name of all the organizations with less than five members.
3. List the name of all the countries that joined some organization after Sweden did it, i.e. ignore those organizations Sweden is not member of.

Solution

1

```
Select Name
From Organization
Where Established in (select min(Established) from Organization);
```

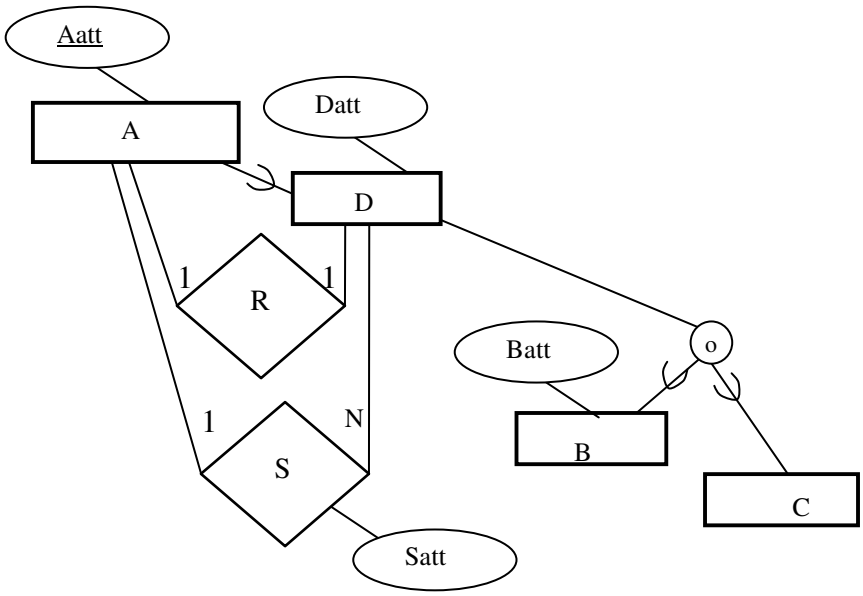
2

```
Select Name, count(*)
From Organization, IsMember
Where Organization.Abbreviation = IsMember.Organization
Group by Name
Having count(*) < 5;
```

3

```
Select distinct Name
From Country,IsMember A, IsMemberB
Where Country.Code = A.Country and A.Organization = B.Organization and A.Joined >
B.Joined and B.Country like 'SWE';
```

Question 3. EER diagram and relational schema (2 + 2 + 1 = 5 p):



1. Translate the EER diagram above into a relational schema with more than one relation (i.e. table). Use the algorithm seen in the course.
2. Translate the EER diagram above into a relational schema with only one relation (i.e. table). Use the algorithm seen in the course. If you think that it is impossible, then explain why.

3. Discuss briefly advantages and disadvantages of having one versus several relations (i.e. tables) in a relational schema.

Solution

1

A(Aatt,isD,Datt,RAatt,SAatt,Satt) where the foreign keys are to A itself.
B(Aatt,Batt) where Aatt is also a foreign key is to A.
C(Aatt) where Aatt is also a foreign key is to A.

2

A(Aatt,isD,Datt,isB,Batt,isC,RAatt,SAatt,Satt) where the foreign keys are to A itself.

3

Advantage of having one table: Fast query answering as no join is needed.
Disadvantage of having one table: Waste of space as many attributes may take null value, plus redundancy and risk of updating anomalies.

Question 4. Normalization (1 + 1 + 1 = 3 p):

1. Give an example of a relation that is in 3NF but not in BCNF. If you think that it is impossible, then explain why.
2. Give an example of a relation that is in BCNF but not in 3NF. If you think that it is impossible, then explain why.
3. What is a candidate key?

Solution

1

R(A,B,C) with functional dependencies {AB->C, C->B}.

2

Impossible because BCNF implies 3NF by definition.

3

A candidate key is a minimal set of attributes whose values uniquely identify the tuples in the relation.

Question 5. Data structures (2 + 2 + 1 = 5 p):

We have a file with 2000000 records. Each record is 20 bytes long. The records have two key attributes X and Y. The file is ordered on X. The database uses a block size of B=4000

bytes and unspanning allocation. Each index record is 4 bytes long.

1. Calculate the average number of block access needed to find a record with a given value for X when using (a) the primary access method and (b) a single level index.
2. Calculate the average number of block access needed to find a record with a given value for Y when using (a) the primary access method and (b) a single level index.
3. In which of the two cases above does the index provide a greater gain over the primary access method ? Explain why the gains in the two cases above are different.

Recall that $\log_2 2^x = x$. That is, $\log_2 1 = 0$, $\log_2 2 = 1$, $\log_2 4 = 2$, $\log_2 8 = 3$, $\log_2 16 = 4$, $\log_2 32 = 5$, $\log_2 64 = 6$, $\log_2 128 = 7$, $\log_2 256 = 8$, $\log_2 512 = 9$, $\log_2 1024 = 10$, $\log_2 2048 = 11$, $\log_2 4096 = 12$, $\log_2 8192 = 13$, $\log_2 16384 = 14$ etc.

Solution

1

The primary access method is a binary search. Since the we can have 4000/20=200 records per block, we need 10000 blocks to store the whole file and, thus, ceiling(log_2 10000) block access at most to find a given record.

We have 10000 data blocks and, thus, we need 10000 index entries. In a block, we can have 4000/4=1000 index entries. Thus, we need 10 blocks to store the index. This implies ceiling(log_2 10)+1 block access to find a given record.

2

The primary access method is a linear search. Since we can have 10000 data blocks, we need 10000 block access in the worst case and 5000 on average to find a given record.

We have 2000000 records and, thus, we need 2000000 index entries. In a block, we can have 4000/4=1000 index entries. Thus, we need 2000 blocks to store the index. This implies ceiling(log_2 2000)+1 block access to find a given record.

3

The gain is clearly greater in the second case, since the index allows us to move from a linear search (primary access method) to a binary search.

Question 6. Transactions and concurrency control (1 + 1 + 1 = 3 p):

Consider the following schedule:

T1	T2	T3
read(x)		
read(z)		
z=z+x		

```
write(z)
                read(x)
                read(y)
                y=y+x
                write(y)
                                read(y)
                                y=y+1
                                write(y)
read(y)
y=y+1
write(y)
```

1. Is the schedule serializable? Justify your answer as shown in the classroom.

2. Apply the two-phase locking protocol to the schedule above.

3. Does the two-phase locking protocol prevent starvation ? If yes, explain how. If not, show a counter-example.

Solution

1

The conflicts are between the read(y) and write(y) instructions of T1 and T2, of T1 and T3, and of T2 and T3. Then, the conflict graph looks like T1<-T3<-T2->T1 and, thus, the schedule is serializable.

2

T1: read_lock(x), write_lock(z), write_lock(y) and then the 7 original instructions and then unlock(x), unlock(y), unlock(z)
T2: read_lock(x), write_lock(y) and then the 4 original instructions and then unlock(x), unlock(y)
T3: write_lock(y) and then the 3 original instructions and then unlock(y)

3

The two-phase locking protocol has nothing to do with starvation. The protocol can produce deadlocks and these have to be resolved in some way. If this is not done in a fair way, i.e. the same transaction gets always aborted, then starvation may occur.

Question 7. Database recovery (2 + 1 + 1 = 4 p):

1. Apply the three recovery methods seen in the course to the system log below. Show all operations (in the correct order) that are performed during the recovery.

   Part of system log:
   Start-transaction T2
   Write-item T2, B, 3, 4
   Start-transaction T3

Write-item T3, A, 7, 8
Write-item T3, A, 8, 1
Write-item T3, A, 1, 5
Start-transaction T4
Write-item T4, B, 4, 5
Write-item T4, B, 5, 10
Start-transaction T1
Write-item T1, C, 8, 9
Write-item T1, C, 9, 10
Commit T1
Commit T2
Commit T3
Commit T4
Checkpoint
→system crash

2. What is a checkpoint ?

3. Besides at checkpoints, when does a transaction write to disk ?

Solution

1

The lists of active and committed transactions since the last checkpoint are empty. Then, no operation is needed to recover from the crash.

2

A checkpoint is a system instruction that writes to disk every buffer whose dirty bit is 1 and pin bit is 0.

3

Besides at checkpoints, changes may be written to disk when the cache is flushed and when a transaction commits (under immediate update version 1).