Exercises for *Transaction Systems*, summer term 2016
Linnea Passing (linnea.passing@tum.de)
http://www-db.in.tum.de/teaching/ss16/transactions/

Sheet No. 8 – Sample Solution

Exercise 1 (5 points) Prove: In the “no blind writes” model, where each data item written by a transaction must have been read before in the same transaction, MVSR = MCSR.

Solution No solution. We overlooked an erratum to the book\(^1\), which rendered this exercise impossible.

Exercise 2 (5 points) Prove: In the “action” model, where each step is a combination of a read operation immediately followed by a write operation on the same data, MVSR = VSR.

Solution Let \( m \) be the MVSR schedule in the action model. Let \( m' \) be a serial mono-version schedule such that \( m \) and \( m' \) are view-equivalent. The action model and the MVSR property ensure that no transaction in \( m \) may overtake another on the same data item (i.e., \( r_1(x) \ r_2(x) \ w_2(x) \ w_1(x) \) is not allowed, because in all serial schedule one transaction would have to read the value written by the other transaction). Thus, an order is defined between all pairs of transactions that operate on the same data item. When such an order is established between two transactions because of *multiple* data items, the order has to be the same for all data items. When transforming \( m \) to \( m' \), the order of write operations cannot change due to the afore-mentioned order. Because of the action model, the order of read operations also cannot change. Thus, all write-read pairs (that form the RF relation) do not change. To put it together: The read-write pairs do not change, and they have to read the latest written version. As a result, \( m' \) is not only MVSR-equivalent to \( m \), but also VSR-equivalent.

Exercise 3 (10 points) Consider a database with a person table (unique name, city). Two operations:

- select(c): select * from person where city = c;
- update(n,c): update person set city = c where name = n;

A B+-tree for both attributes exists. It has height 2 (i.e., root and leaves). The operations are: lookup (search(key)), record fetch (fetch(rid)), record modification (modify(rid)), index maintenance (insert(key,rid) and delete(key,rid)). All operations are transformed into page reads and writes.

We consider two transactions:

- \( T_1 \) finds all persons from Munich and Garching.
- \( T_2 \) moves a couple (John and Jane Doe) from Munich to Garching.

Model them as 3-level transactions. Give a non-serial example for a 3-level schedule that is

\(^1\)Theorem 5.6: http://dbis-group.uni-muenster.de/dbms/media/books/transactionalInformationSystems/tis-errata.pdf
(a) tree-reducible
(b) *not* tree-reducible

**Solution** Assumption: No split/merge in the B+-tree occurs when John and Jane Doe are updated. Transactions 1 and 2:

```
T1
  select(MUC)  select(GAR)
       |         |
  search(MUC) fetch(rid1) . . .  search(GAR) fetch(rid2) . . .
         |        |
   r(root) r(leaf1) r(rid1)       r(root) r(leaf2) r(rid2)
```

```
T2
  update(John, GAR) . . .
        |    |
  search(John) fetch(ridOld) mod(John) del(John,ridOld) ins(John,ridNew)
                   |    |
     r(root) r(leaf1) r(ridOld) w(ridNew)r(root) w(leaf1) r(root) w(leaf1)
```

```
     del(Munich,ridOld) ins(Garching,ridNew)
                   |     |
         r(root2) w(leaf2) r(root2) w(leaf3)
```

Tree-reducible non-serial schedule (page level missing for lack of space):
search\(_1\)(MUC) search\(_1\)(GAR) fetch\(_1\)(rid\_1) fetch\(_1\)(\ldots)
search\(_2\)(John) search\(_2\)(Jane) fetch\(_1\)(rid\_2) fetch\(_1\)(\ldots) c_1
fetch\(_2\)(rid\_OldJohn) modify\(_2\)(John) delete\(_2\)(John, rid\_OldJohn) insert\(_2\)(John, rid\_NewJohn)
delete\(_2\)(Munich, rid\_OldJohn) insert\(_2\)(Garching, rid\_NewJohn) [same for Jane] c_2

Not tree-reducible non-serial schedule (only first level given):

select\(_1\)(MUC) update\(_2\)(John, GAR) select\(_1\)(GAR) c_1 update\(_2\)(Jane, GAR) c_2