Database Systems WS 2023/24
Exercise 9: Distributed 08.01.2024, Due 15.01.2024 12:00 MEZ
Submitted by Erik Schwede, Suma Keerthi

1 Skyline Queries (1 P.)

Given the TPC-H[1] table **"part"**.

We want to compute the skyline over the following four dimensions:

- *p_size:* larger is better,
- *p_retailprice:* less is better,
- *p_container:* Use the function `getContainerSize` from below: less is better.
- *p_brand:* Cannot be compared.

In OLAT you will find a file `skylineFunc.sql`, which implements the `getContainerSize` function. You can either copy the create-statement from the file and execute it in your DBMS or you can directly execute the file with the following command (you may have to change the database name to the one you used to create it).

```
psql -d tpch -f skylineFunc.sql
```

1. Create a SQL query to calculate the number of elements in the skyline defined above (without using the `SKYLINE`-operator). Submit the query and the result.

   Solution:

   ```
   SELECT COUNT(*)
   FROM part p
   WHERE NOT EXISTS(
   SELECT *
   FROM part p1
   WHERE p1.p_size >= p.p_size
   AND p1.p_retailprice <= p.p_retailprice
   AND getContainerSize(p1.p_container) <= getContainerSize(p.
       p_container)
   AND p1.p_brand = p.p_brand
   AND (p1.p_size > p.p_size OR p1.p_retailprice < p.p_retailprice OR
       getContainerSize(p1.p_container) < getContainerSize(p.p_container
       )))
   ```

   The output of the query is 250.

2. Write the same query using the `SKYLINE`-operator. (You do not have to install the plugin and execute the query.)

   Solution:

   ```
   SELECT COUNT(*)
   FROM part p
   SKYLINE OF p.p_size MAX, p.p_retailprice MIN,
     getContainerSize(p.p_container) MIN, p.p_brand DIFF;
   ```

[1] `http://dbis.informatik.uni-kl.de/files/teaching/ws1819/dbs/protected/tpch.dmp.gz`

2 Skyline NN (1 P.)

Implement the nearest neighbor (NN) method to compute skylines in a language of your choice. A basic Java template is available in OLAT. This template provides a (fake) R-Tree implementation which can be used to query for the nearest neighbor of a point, all points within a rectangle, and the nearest neighbor within a rectangle. If you are using another language, you are allowed to also implement a fake R-Tree with the methods described above (or use a real implementation without a included skyline function).

The program should output the calculated skyline (the template already takes care of this).

Submit your code and the output for the following points (If you use the template, the points are already included):

$$(10, 20), (12, 10), (8, 11), (16, 19), (6, 4), (5, 6), (14, 12), (2, 5), (3, 10), (13, 19), (17, 5), (9, 3), (20, 8), (8, 10)$$

Solution:

```
=====================================
Starting Test 1
=====================================
Calculated skyline: [(2,5), (9,3), (6,4)]
Expected skyline: [(2,5), (9,3), (6,4)]
Test one successful!
=====================================
Starting Test 2
=====================================
Calculated skyline: [(404,54), (173,146), (126,292), (7,7709), (458,20), (5,8080), (1030,2), (12,377)]
Expected skyline: [(404,54), (173,146), (126,292), (7,7709), (458,20), (5,8080), (1030,2), (12,377)]
Test two successful!
=====================================
Result of solution is correct.
=====================================

Process finished with exit code 0
```

Submitted by Erik Schwede, Suma Keerthi

3 Transactions (1 P.)

1. **A database has the consistency condition $0 \leq A \leq B$. Describe, for the following transactions, if the condition is fulfilled. Explain your answer.**

   T_1 : `B:= 2*A; A:= 2*B;`

   T_2 : `B:= A+1; A:= B+1;`

   T_3 : `A:= A+2*B; B:= 2*A+B;`

   Solution: The consistency condition of database is met **if** $0 \leq A \leq B$.

   T_1 : `B:= 2*A; A:= 2*B;`

   Here, B is updated before A. Hence, the value of B upon solving would become

   $$B := 2 * A; \textbf{B:= 2A}$$

   When A gets updated, it would fetch the value of B from the previous output to process the transaction

   $$A := 2 * B; A := 2 * 2A; \textbf{A:=4A}$$

   The value of A is greater than the value of B. Hence, the **database consistency condition is not met** in this case.

   Let us consider the values of A = 1 and B = 1 for the given transaction, then the value of B would be

   $$B := 2A; \textbf{B:= 2}$$

   The value of A would be

   $$A := 4A; \textbf{A:= 4}$$

   . Even in this case, **the database consistency condition is not met.**

   Let us consider the values of A = 0 and B = 0 for the given transaction, then the value of B would be

   $$B := 2A; \textbf{B:= 0}$$

   The value of A would be

   $$A := 4A; \textbf{A:= 0}$$

   In this case, **the database consistency condition is met when A:= 0 and B:= 0**;

   T_2 : `B:= A+1; A:= B+1;`

   Here also the database consistency condition is not met because B is updated before A and later it is being using while processing. The value of **B:= A+1;**

   The value of A would be $A := B + 1; A := A + 1 + 1;$ (Substituted the value of B from previous output) **A:= A+2;**

   Considering the values of A=1 and B=1 and substituting in the above equations we get,

   B:=2 ; A:=3 -(i)

   Even if we consider the values of A=0 and B=0, we get **B:= 1 ; A:=2** -(ii)

   In both the cases (i) and (ii), A is greater than B, **the database consistency condition is not met.**

   T_3 : `A:= A+2*B; B:= 2*A+B;`

   Here, A got updated before B and later the value of A was used in B, there might be chances that the database consistency condition is met.

   Solving the values of A and B, we get

A:= A+2*B and
B:= 2*A+B; B:= 2*(A+2*B)+B (Substituting the value of A from above)
B:= 2A+5B

Let us consider the values of A = 1 and B = 1 for the given transaction. Substituting the value of A and B in A:= A+2*B; we get **A:=3** -(iii)

Substituting the value of A (by considering the previous output of A) and B in B:= 2A+5B; we get **B:=11** -(iv)

Let us consider the values of A = 0 and B = 0 for the given transaction. Substituting the value of A and B in A:= A+2*B; we get **A:=0** -(v)

Substituting the value of A (by considering the previous output of A) and B in B:= 2A+5B; we get **B:=5** -(vi)

Hence from (iii), (iv), (v) and (vi), we can say that, $A \leq B$ and the **database consistency condition is met.**

2. **For each of the previous transactions, provide the input, read, and write operations. Assume immediate write-back of the changed values. Describe the effect of these operations on the main memory and the hard disk. The initial values are $A = 5$ and $B = 10$.**

Solution: Input loads the data from Disk to Main memory, Read and Write operations are performed in transactions local address space, Output loads the data from Main memory back to Disk.

T_1 : `B:= 2*A; A:= 2*B;`

INPUT(A) : Loads A from Disk to Main memory (A=5)
READ(A,t): Reads the value of A from Main memory to the local transaction variable
t:= 2 * A: Computes the value of A (t:= 2*5= 10)
WRITE(B,t): Immediate write-back of changed values. Writes the value of B from local transaction variable to Main memory **(B:= 10)**
READ(B,t): Reads the value of B from the Main memory
t:= 2 * B: Computes the value of B (t:=2 * 10 = 20)
WRITE(A,t): Writes the value of A from local transaction variable to Main memory **(A:= 20)**
As per database consistency condition, $A \leq B$ but T_1, $A > B$ which does not meet the condition.

T_2 : `B:= A+1; A:= B+1;`

INPUT(A) : If A is not present in Main memory, it gets loaded from Disk (A=5)
READ(A,t): Reads the value of A from Main memory to the local transaction variable
t:= A + 1: Computes the value of A (t:= 5 + 1 = 6)
WRITE(B,t): Immediate write-back of changed values. Writes the value of B from local transaction variable to Main memory **(B:= 6)**
READ(B,t): Reads the value of B from the Main memory
t:= B + 1: Computes the value of B (t:=6 + 1 = 7)
WRITE(A,t): Immediate write-back of changed values. Writes the value of A from local transaction variable to Main memory **(A:= 7)**
As per database consistency condition, $A \leq B$ but in transaction T_2, $A > B$ which does not meet the condition.

T_3 : `A:= A+2*B; B:= 2*A+B;`

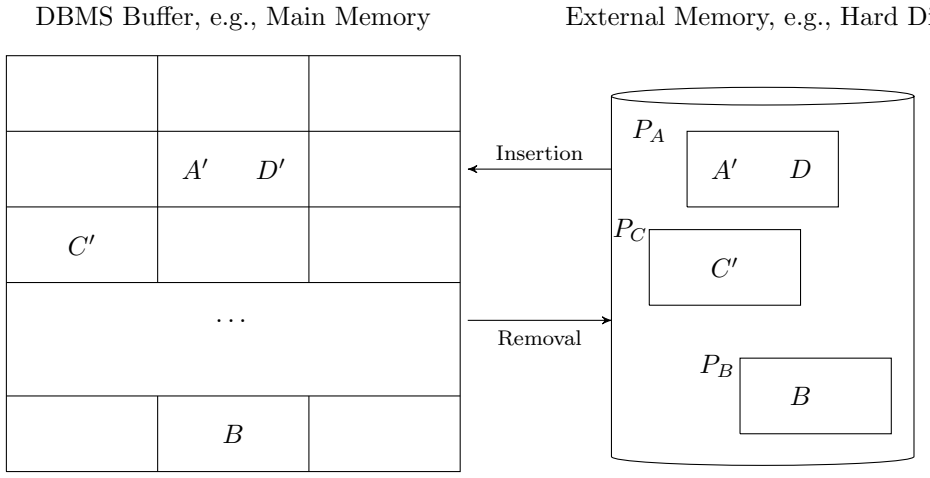
INPUT(A) : If A is not present in Main memory, it gets loaded from Disk (A=5)
READ(A,t): Reads the value of A from Main memory to the local transaction variable
INPUT(B) : Loads B from Disk to Main memory (B=10)

READ(B,t): Reads the value of B from Main memory to local transaction variable
t:= A + 2 * B: Computes the value of A (t:= 5 + 2 * 10 = 25)
WRITE(A,t): Immediate write-back of changed values. Writes the value of A from local transaction variable to Main memory **(A:= 25)**
READ(A,t): Reads the value of A from Main memory to the local transaction variable
READ(B,t): Reads the value of B from Main memory to local transaction variable
t:= 2 * A + B: Computes the value of B (t:=2 * 25 + 10= 60)
WRITE(B,t): Immediate write-back of changed values. Writes the value of B from local transaction variable to Main memory **(B:= 60)**

This transaction T_3 satisfies the database consistency condition of $A \leq B$

OUTPUT writes the data from Main Memory to Disk upon mentioning after each transaction. Until then, the Main memory would be used. Now that, we did not have to OUTPUT the transactions, the Disk is unaffected.

3. **Given two transactions T_1 and T_2 and the following situation (' marks changed entries):**



The following table describes the operations of T_1 and T_2 at different times.

Time	T_1	T_2
0	READ(A,a)	READ(C,c)
10	a:=a+10	c:=c*2
20		READ(B,b)
30	WRITE(A,a)	
40	READ(D,d)	b:=b+c/4
50	d:=17*d+42	
60	OUTPUT(A)	WRITE(C,c)
70	WRITE(D,d)	OUTPUT(C)
80	OUTPUT(D)	WRITE(B,b)
90	COMMIT	
100		COMMIT

Discuss if the illustration matches the entries of the table. Does it only represent the table at a specific time (if yes, when and why) or does it not represent it at all?

Solution: The illustration does not completely match with the entries of the table because in the Main memory, the changed entries of A' and C', are written to the Hard Disk. However, the changed entry of D' has still not been written to disk. It might be because of the occurance of crash.
Yes, the illustration represents a specific time between 70 and 80. These might be the timeperiods where the crash might have occured because, the WRITE(D) operation writes the changes from transaction local variable to the Main memory. The OUTPUT(D) operation writes the changes from Main Memory to the Hard disk and the image does not show illustration of D' reflected on the Hard disk.

If the system crashes during the execution, what would have to be done to guarantee ACID? Which parts of ACID are affected? How does the situation change at timestamp 101?
Solution:
For Transaction T1: From the table, we can state that the system might have crashed in Main Memory between the time of 70 and 80 because the data in the disk remained. In the transaction T1, until the time of 60, the OUTPUT(A) have been successfully writted to the disk. Hence, it is guarenteed that the value of A would be in disk.

However, the OUTPUT(D) has not been successfully written to disk due to crash. We have COMMIT operation at the end of transaction T1, which confirms the user that the operation performed have been successfully completed. However, we have observed from the given table and also the illustration that the transaction has failed and hence the commit will never be executed.

Hence, we perform UNDO on A (from A' to A) because even though A has been written to Hard Disk, the OUTPUT operation on D have failed and the commit did not execute affecting **CONSISTENCY** due to partial transactions. By performing UNDO on A, the states of A and D in P_A, Main Memory and also the Hard Disk would become same which guarentees the property of **ATOMICITY** which means all or none transactions.

For Transaction T2:
In transaction T2, we see that the changed entry of C' has been written to Hard Disk.
Even though there were transactions performed on B, the changed entry of B has not been written to the Main Memory because of crash and the OUTPUT(B) has not been perfomed before crash. Hence, the state of B is same in Main Memory and also the Hard Disk.

The situation change at timestamp 101:
Let us assume that the crash has recovered from the previous steps using log until timestamp 100. At 101, we see that both the transactions have commited. After the recovery, WRITE(B) would have been successfully performed however, the OUTPUT(B) operation has not been perfomed. This affects **CONSISTENCY** because, in Main Memory we get to see B' and in Hard disk we get to see B. Hence, we do **REDO** operation on B, which changes the state of B from B' in Hard Disk which guarentees the property of **DURABILITY**. If the user does not wish to perform OUTPUT(B), then the **UNDO** operation should be perfomed to revert the change from Main memory B' to B which guarentees **ATOMICITY.**

4 Recovery (1 P.)

In a DBMS three transactions T_1, T_2, and T_3 are executed concurrently. The data accessed performed by the transactions are stored in the log (Table 1).

1. The system crashes after step 19. None of the changes were written to the database, but the log is complete. Using this log, perform the three stages of recovery and explain what happens. Explain your steps in-detail.
2. Is the log changed after completing the recovery process? If yes, explain what changed.

	T_1	T_2	T_3	Log
				[LSN, TA, PageID, Redo, Undo, PrevLSN]
1.	BOT			$[\#01, T_1, -, BOT, -, 0]$
2.	$r(B, B_1)$			
3.	$B_1 = B_1 * 8$			
4.		BOT		$[\#02, T_2, -, BOT, -, 0]$
5.	$w(B, B_1)$			$[\#03, T_1, P_B, B = B * 8, B = B/8, \#01]$
6.		$r(A, A_1)$		
7.			BOT	$[\#04, T_3, -, BOT, -, 0]$
8.		$A_1 = A_1 + 9$		
9.		$w(A, A_1)$		$[\#05, T_2, P_A, A = A + 9, A = A - 9, \#02]$
10.	$abort$			$[\#06, T_1, -, abort, -, \#03]$
11.		$r(C, C_1)$		
12.			$r(A, A_2)$	
13.			$A_2 = A_2 + 3$	
14.		$C_1 = C_1/1$		
15.		$w(C, C_1)$		$[\#07, T_2, P_C, C = C/1, C = C * 1, \#05]$
16.		$commit$		$[\#08, T_2, -, commit, -, \#07]$
17.			$w(A, A_2)$	$[\#09, T_3, P_A, A = A + 3, A = A - 3, \#04]$
18.			$r(B, B_2)$	
19.			$B_2 = B_2/6$	
				Crash

Tabelle 1: Log

4.1 Solution: The Redo operation is catgorized into two types.
i)**Selective Redo** - The winner transactions are included in Redo
ii) **Complete Redo** - The winner as well as loser transactions are included in the Redo. Now that we have concurrent transactions, we need to consider **Complete Redo** for this question.

The three stages of Recovery are
1)**Analysis** (Identification of the current state of Database)
2)**Redo** (Complete replay of history)
3)**Undo** (Removal of changes made by Losers)

In **Analysis** stage, the winner and loser transactions are identified. From the log file , we can clearly see that T_1 and T_2 are the **Winner transactions** because they have completed successfully before the crash

occured. T_3 has failed during the crash and hence this is the **Loser transaction.** All the changes made by this transaction to the disk needs to be undone.

In the **Redo** stage, we perform Complete Redo of all the transactions (Winners and Losers). Hence, we redo all the below steps of the transactions. **The redo steps are performed on the log file from beginning till the end.**

T_1 LSN#03 B = B * 8
T_2 LSN#05 A = A + 9
T_1 LSN#06 Abort , undo is performed here. This goes back to PrevLSN mentioned in LSN#06 which is LSN#03 and performs undo by B = B / 8, then it refers the PrevLSN of LSN#03 which is LSN#01.
In LSN#01, as it is the Beginning of the Transaction, the PrevLSN for #1 is 0. Hence, the undo operation has been successfully completed.
T_2 LSN#07 C = C / 1
T_3 LSN#09 A = A + 3

In the **Undo**, we perform Undo of the loser transactions. **The undo steps are performed on the log file from the starting from the end of the log file until the beginning of the log file.** As per the table, T_3 is the loser transaction. Hence, we check the perform Undo on T_3 as follows

T_3 LSN#09 A = A - 3

After the above step, we check the PrevLSN of LSN#09 which is #04. The LSN#04 has PrevLSN as 0 and hence the Undo operation has been successfully completed. After the recovery, this transaction of T_3 would not be visible on the Disk.

4.2 Solution:

Yes, the log file changes after the recovery process. **The Compensation Log Records (CLR) are created for every undo operation.**
The CLR consists of LSN with other details and also the undo information from actual operation.
The CLR is checked from the beginning of the file until the end of the file.
For this table, the CLR is created for Loser transaction which is T_3
<#09', T_3, P_A, A = A - 3, #09,#04 >
#09' refers to the CLR.

The undo operation of LSN is the redo operation of CLR. where #09 refers to the PrevLSN of Undo operation of the transaction and #04 refers to the UndoNxtLSN.

<#04', T_3, _, _, #09',0>
Here, the PrevLSN is 0 and hence it means that the undo operation ends and hence we add PrevCLR to the output of the log. If the recovery process starts again, we see the CLRs and observe that the undo has already been done when it checks the PrevLSN of the CLR as it points to already visited transaction which does not result in multiple executions of Redo in CLR.

The CLR is also created for aborted transactions because, on aborting a transaction, the operation that undergoes is also an Undo operation.
Hence, the aborted transaction in the table was LSN#06, and we create CLR for PrevLSN of LSN#06 which is #03.

<#03', T_1, P_A, B = B / 8, #03,#01 >
<#01', T_1, _, _, #03',#0 >