Sheet 9

Problem 1.a

Solution

```
SELECT Count(*) FROM part part1
WHERE NOT EXISTS(
SELECT * FROM part part2
WHERE part2.p_size >= part1.p_size
AND part2.p_retailprice <= part1.p_retailprice
AND getContainerSize(part2.p_container) <= getContainerSize(part1.p_container)
AND part2.p_brand = part1.p_brand
AND (
part2.p_size > part1.p_size
OR part2.p_retailprice < part1.p_retailprice
OR getContainerSize(part2.p_container) < getContainerSize(part1.p_container)
)
)
```

Problem 1.b

Solution

```
SELECT Count(*) FROM part
WHERE SKYLINE OF p_size MAX, p_retailprice MIN, getContainerSize(p_container) MIN,
p_brand DIFF;
```

2.

Solution provided in .java file.

Output:

```
PS C:\Users\Nabid.Imteaj\Downloads\study\ex 9\exec> javac .\Skyline.java
PS C:\Users\Nabid.Imteaj\Downloads\study\ex 9\exec> java Skyline
=======================================
Starting Test 1
=======================================
Closest: (2,5)
Closest: (6,4)
Closest: (9,3)
Calculated skyline: [(2,5), (9,3), (6,4)]
Expected skyline: [(2,5), (9,3), (6,4)]
Test one successful!
=======================================
Starting Test 2
=======================================
Closest: (173,146)
Closest: (404,54)
Closest: (458,20)
Closest: (1030,2)
Closest: (126,292)
Closest: (12,377)
Closest: (7,7709)
Closest: (5,8080)
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.
=======================================
```

Sheet-9

Problem 31

(a) Consistency condition $0 \leq A \leq B$

T_1: $B := A + B$; $A := A + B$

If we consider $A = B = 0$, then we can find $A = B = 0$, which satisfies the condition. However, for $A = 3$, $B = 4$, the results will be $B = 7$ and $A = 10$, where $A > B$, so condition will be failed.

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

First statement makes A larger than B which fails the condition. As both the statements to be executed to complete the transaction, we need to wait till execution of 2nd statement. After execution of 2nd statement, we've made B larger than A. So T_2 satisfies the condition.

T_3: $A := A + B;$ $B := A + B;$

The operation is exactly same as T_2, that means, after execution of 2nd statement, T_3 will satisfy the condition.

(b) For T_1 ! Consistency condition $0 \le A \le$

Following READ/WRITE operations will be executed:-

⇒ INPUT(A) := Read A in buffer

⇒ INPUT(B) := Read B in buffer

⇒ READ (A, a) := $a = 5$

⇒ READ (B, b) := $b = 10$

⇒ WRITE(B,a+b):= B=15

⇒ READ(A,a):= a=5

⇒ READ(B,b):= b=15

⇒ WRITE(A,a+b):= A=20

Here we can see A>20, so transaction will be stopped according to consistency condition.

For T_2!

⇒ INPUT(B):= Read B to buffer

⇒ READ(B,b)!= b=10

⇒ WRITE(A,b+1):= A=11

⇒ READ(A,a)!= a=11

⇒ WRITE(B,a+1)!= B=12

Here B>A, so, in order to guarantee consistency, B should be written first to disk before A

For T_3:

⇒ INPUT(A) := Read A to buffer

⇒ INPUT(B) := Read B to buffer

⇒ READ(A, a) := $a = 5$

⇒ READ(B, b) := $b = 10$

⇒ WRITE(A, a+b) := $A = 15$

⇒ READ(A, a) := $a = 15$

⇒ READ(B, b) := $b = 10$

⇒ WRITE(B, a+b) := $B = 25$

Here $B > A$, so consistency can be guaranteed after crash if B is written first to disk than A.

(C)

Whether illustration matches entries or not: From the illustration, we can see, changes to A has already been written to external disk, but changes to C (which is C'), has not yet been written in external disk. Time 60 denotes the situation, where changes of A has been written to disk and changes to C are awaiting to be written to external disk. Executions after 60 are not illustrated in the figure at all.

What have to be done to guarantee ACID? Which parts of ACID are affected?

In time of system crash, we need to revert back transaction and erase contents from external disk, which will ensure Atomicity, that means all or nothing.

Situation at timestamp 91

From the table, we can see that, at timestamp 90, both T_1 and T_2 has been committed. So changes should be durably persisted, that means Durability should be assured.

4. a)

There are three phases of Recovery: Analysis, Redo and Undo.

Analysis: In this phase, the log file is inspected from the beginning to the end to identify winner and loser transections.	From the Log Table, we can see that the transaction **T3** was committed only, thus **T3** is the only winner transaction. Transection **T1** was still running when the crash happened and **T2** was aborted before the crash. As a result, both **T1** and **T2** are marked as loser transaction.
Redo: Here, all logged changes including the changes made by loser transactions, which are not yet written to disk, are written to disk *in the order of execution time*.	As the system crashed after step 18 and none of the changes were written to the disk, then all the log changes will be written to the disk as they appear.
Undo: In the last stage of recovery, changes made by the loser transactions are made undone, in reverse order of their original execution.	As **T1** was winner transaction, so the changes made by LSN 06 will not be undone. But changes made by LSN 03, 05, 07 will be made undone in reverse order as **T1** and **T2** were loser transection.

4. b)

After completing the recovery process, compensation log record (CLR) will be added to the log file. For each undo operation, one CLR will be created. As a result, CLR will not be created for T3, but will be created for T1 and T2.

Log entries after completed restart:

- [#01, T1, -, BOT, -, 0]
- [#02, T2, -, BOT, -, 0]
- [#03, T1, PA, A = A − 2.3, A = A + 2.3, #01]
- [#04, T3, -, BOT, -, 0]
- [#05, T2, Pb, B = B − 10, B = B + 10, #02]
- [#06, T3, Pb, B = B 2 , B = B ∗ 2, #04]
- [#07, T2, Pc, C = C + 5, C = C − 5, #05]
- [#08, T2, -, abort, -, #07]
- [#09, T3, -, commit, -, #06]
- <#08’, T2, -, -, #08, #07>
- <#07’, T2, Pc, C = C − 5, #08’, #05>
- <#05‘, T2, Pb, B = B + 10, #07’, #02>
- <#03’, T1, PA, A = A + 2.3, #03, #01>
- <#02’, T2, -, -, #05’, 0>
- <#01’, T1, -, -, #03’, 0>