Lab 3 - SimpleDB Report

Exercise1:

I used a basic loop for the join operation. It iterates over each tuple in child 1 operator, and for each of these, iterate through the child 2 operator's tuples. If a tuple pair meets the predicate, we merge and return them. If not, the iteration continues. After checking all the tuples of child 2 against the current tuple of child 1, we rewind child 2 and move on to the next tuple of child 1.

Initially, my implementation had a flaw, which cost me a lot of time to debug. When a matching tuple from the child 2 satisfied the predicate with a tuple from the child 1, I returned the match but didn't finish checking the remaining tuples in the child 2 against that same tuple of the child 1 in the next iteration, i was directly moving on to the next tuple of child 1. To correct this, I implemented a global variable to remember the current tuple from the first child. This ensures that all potential matches in the second child are fully evaluated against the current first child's tuple before moving to the next one.

Exercise2:

In the integer aggregator part for the AVG operator, I used an additional data structure named 'count' to track the number of elements for calculating the average. We sum the elements and count them separately. When returning the result, we divide the sum by the count. This 'count' data structure was intended only for the AVG operator. However, I used it by mistake with the COUNT operator, which led to problems. It took significant time to debug and realize that I was storing COUNT results in the 'count' structure meant for AVG calculations only and i was returning another empty structures.

Exercise3:

In the insert tuple method of the heapfile, I initially thought it wasn't responsible for creating a new page when all existing pages were full, and that it should throw an exception instead. I believed the correct approach was to evict a page first, then create a new one. However, I couldn't pass the test until I added functionality to create a new page when all others were full. At this point, I also had to implement the writePage function to save the newly created pages to disk. I made an error in the writePage function when calculating the offset, whichwasn't caught in the current tests but caused major issues in the next exercise and took a long time to debug.

Exercise4:

I encountered an issue here due to an mistake in the write method of the bufferpool (calculation of the offset), which the insert method was using. Identifying the bug was challenging as there were no errors, but the program failed due to an assertion mismatch in the system test. To troubleshoot, I had to investigate the test code, adding debugging statements to observe the actual tuples written to disk. I noticed issues like duplicate tuples and tuples in the wrong order. It took significant time to determine that the root cause of these problems was the write function of bufferpool.

Exercise5:

For the eviction policy, I implemented LRU-K, where K is configurable in the code. Each page is associated with a linked list representing a FIFO queue that stores access times. When eviction is needed, we evict the page with the oldest Kth access time by removing it from the bufferPool's 'PageToId' structure along with its access history. I also updated the 'getPage' method to refresh a page's access history each time it's accessed. Most pages in test case are accessed only 1 or 2 times. Although I set K to 5 in LRU-K, it defaults to using the oldest access time in the history if a page hasn't been accessed K times.

Time spent on the Lab

This lab took me more time and effort than the last one. I worked around 35 to 40 hours during the Christmas vacation and the following weeks. Working alone was difficult but very rewarding as this was an opportnity for me to learn in full the concepts see in the lectures and put them to practice. This was also an opportnity for me to hone my debugging skills.

Querry tests

Table:

```
1,10 || 1,20 || 1,30 || 2,40 || 2,50 || 2,60 || 3,70 || 3,80 || 3,90 ||
4,100 || 4,110 ||4,120
```

First test:

```
SimpleDB> SELECT d.f1, MAX(d.f2) FROM data d WHERE d.f1 > 2 GROUP BY d.f1;

d.f1    max (d.f2)
------------------------
3       90
4       120

 2 rows.
Transaction 3 committed.
----------------
0.03 seconds
```

Second Test

```
SimpleDB> SELECT d.f1, COUNT(d.f1) FROM data d WHERE d.f1 > 2 GROUP BY
d.f1;

d.f1    count (d.f1)
--------------------------
3       3
4       3
```