INF-2700
Assignment 3
Magnus Dahl-Hansen
November 2022

Task 1. Implementing Natural Join

Two different algorithms have been implemented for the natural join operation: *Nested Loop Join* and *Block Nested Loop Join*. Both algorithms have in common that they will create and return a table as a result of the values that the joined tables (table 1 and table 2) have in common. The algorithms uses the pre-implemented methods to iterate over the pages and extract values by using a position variable. Each time a value is retrieved from a page the position variable is incremented so that the next record in a page can be fetched. When the position variable in each page is updated a check is made to see if the position has become greater than the size of the page. If this is the case, the current page will be unpinned and the next page (if it exists) will be retrieved, and the process is repeated.

Nested Loop Join is an algorithm that retrieves the first page and the first value in the page from the outer table and compares it with all possible pages and values from the inner table. After the first value have been compared the algorithm repeats the process with the remaining pages and values from the outer table until all the values have been compared with all values from the inner table. This algorithm have been implemented by using a for-loop which sequentially retrieves the pages that the outer table consists of. For every page that is retrieved a while-loop will execute which retrieves all the records that the page contains. Then in turn for each and every record that is extracted, a new for-loop is executed which will sequentially extract pages from the inner table and a new while-loop will extract all records from each of these pages. When a record value from both the outer and inner table have been extracted, they are compared to see if they are equal. If the two integer values are identical they will be added to the result table as a new record but will also be appended to an array of integers. The array is used to keep track of which integers already exists in the result table, so that no duplicate numbers are added. In a scenario which the array already contains a given value or that the values from the tables are not identical, they are discarded.

Block Nested Loop Join is an algorithm which compares all records from the first page of the outer table with all values from the inner table before getting the next page of the outer table. This way the number of reads is reduced because all the pages from the inner table does not need to be read for each and every record from the outer table. The pages from the inner table only needs to be reads for every page in the outer table.
This algorithm is implemented by using for-loops that sequentially extracts pages from the outer table and the inner table and then using while-loops that sequentially extracts and compared records from the different pages. Since the for-loop for the inner table is placed inside the for-loop of the outer table, all pages from the inner table will be extracted for every page in the outer table. In the same way as *Nested Loop Join* is implemented, a check is made to see if a record from the outer

page is identical to a record from the inner page. If the record values are identical and they do not exists in the array that looks for duplicate values, they are added as a new record to the result table and the integer value is added to the array.

The code has been tested by implementing two functions which creates test tables and performing natural join on them. The function which creates a table will fill it with two fields and a given number of records. The record values which are appended can either be only even numbers or both odd and even numbers ranging from 0 to the record size of the table. By joining one table containing only even numbers with a table of equal size containing both even and odd numbers, one can expect the result of the join to be half the size of the original table sizes, and therefore create an easy way to check if the join operation is successful.
The function which performs the natural join sends two given tables through either the *Nested Loop Join* or the *Block Nested Loop Join* algorithm. While natural join is performed the number of disk seeks, reads, writes and IO's is counted and displayed at the end of the join operation.

Compiling and running the program:

The program is compiled by navigating to the folder "mda105/assignment-3/db2700" in the terminal and run the "make" command.
This will generate two executable files. The test file can be run by typing "./run_test" which will perform either a "nested loop join" or a "block nested loop join" on two test tables.
The base program can be run by typing "./run_front". While using the base program the user can type the "help" command to understand how the program works.

To use the "nested loop join" algorithm or the "block nested loop join" algorithm in the test or the base program the user has to manually change the implementation. This can be done by commenting in or out the function lines in the file: *schema.c* on line number 1041 and 1042.

Task 2. Performance

Four tests have been performed to test the performance of the different algorithms. The tests have been carried out with tables that contains two integer fields and 366, 1000, 10.000 and 50.000 records.

Test 1: Natural join on two tables with 366 records:

```
    NESTED_LOOP_JOIN()
Nested_loop_join found: 183 matches
INFO:  Number of disk seeks/reads/writes/IOs: 5/12/1/13
INFO:  test_tbl_natural_join() done.
```

Nested Loop Join:

```
    BLOCK_NESTED_LOOP_JOIN
Block_nested_loop_join found: 183 matches
INFO:  Number of disk seeks/reads/writes/IOs: 5/12/1/13
INFO:  test_tbl_natural_join() done.
```

Block Nested Loop Join:

Test 2: Natural join on two tables with 1000 records:

```
   NESTED_LOOP_JOIN()
Nested_loop_join found: 500 matches
INFO:  Number of disk seeks/reads/writes/IOs: 1024/17017/4/17021
INFO:  test_tbl_natural_join() done.
```

Nested Loop Join:

```
    BLOCK_NESTED_LOOP_JOIN
Block_nested_loop_join found: 500 matches
INFO:  Number of disk seeks/reads/writes/IOs: 41/306/4/310
INFO:  test_tbl_natural_join() done.
```

Block Nested Loop Join:

Test 3: Natural join on two tables with 10.000 records:

```
   NESTED_LOOP_JOIN()
Nested_loop_join found: 5000 matches
INFO:  Number of disk seeks/reads/writes/IOs: 10244/1640164/40/1640204
INFO:  test_tbl_natural_join() done.
```

Nested Loop Join:

```
    BLOCK_NESTED_LOOP_JOIN
Block_nested_loop_join found: 5000 matches
INFO:  Number of disk seeks/reads/writes/IOs: 408/27060/40/27100
INFO:  test_tbl_natural_join() done.
```

Block Nested Loop Join:

Test 4: Natural join on two tables with 50.000 records:

```
   NESTED_LOOP_JOIN()
Nested_loop_join found: 25000 matches
INFO:  Number of disk seeks/reads/writes/IOs: 51226/41000820/203/41001023
INFO:  test_tbl_natural_join() done.
```

Nested Loop Join:

```
    BLOCK_NESTED_LOOP_JOIN
Block_nested_loop_join found: 25000 matches
INFO:  Number of disk seeks/reads/writes/IOs: 2046/673220/203/673423
INFO:  test_tbl_natural_join() done.
```

Block Nested Loop Join:

By analyzing the test results I can conclude that the implemented algorithms work correctly. The algorithms counts the expected number of matches and displays the correct values when running the test program in the terminal.

The tests also shows that:

- The *Nested Loop Join* uses on average 25 times more disk seeks than *Block Nested Loop Join.*
- *The Nested Loop Join* uses on average 55 times more disk reads than *Block Nested Loop Join*.
- Both algorithms uses the same number of disk writes.
- *The Nested Loop Join* uses on average 55 times more disk IO's than *Block Nested Loop Join.*

These results clearly shows that the *Block Nested Loop Join* algorithm is more efficient than the *Nested Loop Join* algorithm. However, this is only true if the memory size is limited and the database does not fit inside the available buffer size. This is proven by the first test which shows that the two algorithms has the exact same result when the record sizes range from 0 - 366. When the database is small enough to fit inside memory then the *Nested Loop Join* is just as a good

algorithm to use as the *Block Nested Loop Join* algorithm. If only one or none of the databases fits inside memory, then the *Block Nested Loop Join* algorithm is the most efficient one to use.

Task 3. Think out of the box

The block nested algorithm compares each block from the outer table (B_r) with each block in the inner table (B_s). In a case where the database does not fit entirely into memory the cost of this algorithm is: (B_r *B_s) + B_r. In the best case scenario where the database fits entirely into memory, the cost of this algorithm will be only B_r+B_s. However, in most scenarios this is not the case...

A good alternative to the *Block Nested Loop Join* algorithm is the *Merge Join* algorithm. *Merge Join* works by having pointers to point to the values of each table. It starts by comparing the first value of each table, then proceeds to increment the pointer of the inner table to compare the values from the inner table as long as these values match the outer table's values. When the matches are no longer identical, the algorithm will increment the pointer in the table that has the lowest value. This process will continue until the end of each table.

The reason why *Merge Join* is such a good alternative to *Block Nested Loop Join* is because it only needs to iterate over each value in each table once, instead of going back to the beginning and iterate over the entire table again as one would do in *Block Nested Loop Join*. In the best case scenario *Merge Join* only uses B_r + B_s number of block reads, compared to *Block Nested Loop Join's* (B_r * B_s) + B_r number of block reads.
Another positive case with *Merge Join* is that is will equally efficient regardless of the size of the database. In contrast the tables have to be sorted for the *Merge Join* to work. So in a case where the tables are not sorted, the sorting cost of the tables must be included in the calculation.