INF-2700: Database Systems

Assignment 3

Magnus Lyngra

UiT id: mly004@uit.no

Git user: Opusmag

Exam id: i

November 7, 2023

1 Introduction

In this assignment we were tasked with implementing natural join using two different algorithm. The first was a nested loop join, the second was a block nested loop join. Below the design and implementation of these two algorithms for implementing the natural join operation will be explained.

In addition, the performance of each algorithm measured up against each other will be discussed in the experiments and results section and at the end there will be a section discussing if there is a different algorithm that would be better suited if the database was stored in a B+ tree organized file and how that algorithm would compare to the block nested loop join.

In order to run the program you must navigate to the correct directory (where you have downloaded the repository), run the "make" command, run ./run_front, then type the commands "test", "test2" and "select * from test_table natural join test_table2" and you will perform the natural join operation. By default the nested loop join is run. If you wish you test the block nested loop join, you must comment out line 1000 to 1025, and then uncomment line 1028 to line 1064. You must then save, run make cleanall and repeat the previous stated line of commands.

2 Design

[1, 2] The design of the natural join was largely influenced by the precode and the implementation of the previous assignments. In order to join to tables, these tables had to exist, so they were created and the tables needed schemas and records, so all this needed to be created, in addition to result records, schema and table that could hold the results. Since this had also been done in the previous assignments, the same recipe could be followed for this assignment. Finding the common field between the two was the first challenge for this implementation. Luckily it was specified that there only needed to be one common field, so the functions that created the test tables could be amended to have a common int field and a unique string field and in the function for the natural join. It would then be possible to loop through the fields in the tables to find the common one, then create a join table and add the fields from the other tables to that table.

In the book [1] page 704-707 details how to implement a nested-loop join and a block nested-loop join. For a nested-loop join, one simply uses a for loop to iterate over all the records in the left table, then another for loop to iterate over all the records in the right table and then check if the int field record value of the left table record is equal to the int field record value of the right table, if it is, the record can be added to the result table.

For the block nested-loop join, the design was intended to follow the book where now, instead of two four loops, the book describes four for loops. The first iterates over all the blocks in the left table, then the next

for loop iterates over all the blocks in the right table, then the third for loop iterates over all the records in the blocks in the left table and the fourth for loop iterates over all the records in the right table.

3 Implementation

[1, 2] The table_natural_join function takes two table pointers as arguments. It performs a natural join operation between the two tables. The function first finds a common integer field between the two schemas. If no common integer field is found, an error message is printed. The function then creates a result schema that is a copy of the left schema and adds fields from the right schema that are not already in the result schema.

The function then calculates the number of records per block in each table by subtracting the page header size from the block size and dividing the result by the length of the schema of each table. It sets the position of each table to the beginning using the set_tbl_position function.

The nested-loop join enters a nested loop structure to iterate over each record in each block in each table. For each record in the left table, it gets the corresponding page and sets the current position in the page. If the end of the page is reached, it breaks out of the loop. It then gets the record from the page using the get_page_record function.

For each record in the right table, it performs similar operations. If the integer field of the current record in the left table matches the integer field of the current record in the right table, it creates a new record in the result schema and assigns the integer field and the string fields from the left and right records to the new record. It then appends the new record to the result schema. This is hardcoded for simplicity which works in this case because there's only one int_field and that is hardcoded so it works, but it would not be advisable in a professional setting where the code would be used in production.

Finally, it unpins the current page in each table after all records in the page have been processed and returns the result table. For some reason the records are not ordered when using nested-loop join, but they are with block nested-loop join. Considering the code loops through all records in order, one would think they would be ordered in both cases, but they are not.

The block nested-loop join begins by declaring several integer variables to keep track of the number of blocks and records in the left and right tables, the current block and record being processed in each table, and the number of records per block in each table. It also declares two pointers of type page_p to hold the current page being processed in each table.

The block nested-loop join calculates the number of records per block in each table by subtracting the page header size from the block size and dividing the result by the length of the schema of each table. It sets the position of each table to the beginning using the set_tbl_position function.

The function then enters a nested loop structure to iterate over each record in each block in each table. For each record in the left table, it gets the corresponding page and sets the current position in the page. If the end of the page is reached, it breaks out of the loop. It then gets the record from the page using the get_page_record function.

For each record in the right table, it performs similar operations. If the integer field of the current record in the left table matches the integer field of the current record in the right table, it creates a new record in the result schema and assigns the integer field and the string fields from the left and right records to the new record. It then appends the new record to the result schema. This is hardcoded for simplicity which works in this case because there's only one int_field and that is hardcoded so it works, but it would not be advisable in a professional setting where the code would be used in production.

Finally, it unpins the current page in each table after all records in the page have been processed and returns the result table.

4 Experiments and Results

[1] In order to test the performance of the two algorithms against each other, the number of records in each of the two tables were set to 100 000. The nested loop join had the following number of disk seeks/reads/writes/IOs: 502387/285645540/105621/285751161. In comparison, the block nested loop join had the following number of disk seeks/reads/writes/IOs: 11432/8171020/10715/8181735. The block nested loop join has significantly fewer disk seeks/reads/writes/IOs than the nested loop join. This indicates that the block nested loop join is more efficient in terms of I/O operations and will likely have better performance than the nested loop join.

Fewer I/O operations mean that less time is spent reading from or writing to disk, which can significantly speed up queries and other operations. That is why database systems often use techniques like indexing, partitioning, and in-memory processing to minimize I/O operations.

5 B+ Tree organized database

[1, 3] A possible algorithm that could be used is a sort-merge join, which is a general technique for joining two sorted lists of data. One way to use a sort-merge join could be by doing the following. First, you would sort both tables by the join column using the B+ tree index. This would be done by traversing the leaf nodes of the B+ tree in order. Then, merge the two sorted tables by comparing the values in the join column and outputting the matching rows. This would be done by using two pointers that scan the tables from left to right and advance according to the comparison result.

Now as for comparing this to a block nested-loop join, there are advantages and disadvantages for both, depending on the size, order and index of the tables. A sort-merge join algorithm is more efficient for larger tables, especially if they are already sorted or have an index on the join column. A block nested-loop join algorithm is more suitable for small tables or when one table is much smaller than the other. A sort-merge join algorithm can produce an ordered output, while block nested-loop join algorithm might not give an ordered output. Sort-merge also requires more memory than a block nested-loop, which could be an issue depending on system specs.

6 Sources

References

[1] silberschatz Silberschatz, A., Korth, H. F.,, Sudarshan, S. (2010). Database system concepts. New York: McGraw-Hill.

[2] nested loop join Nested-loop join algorithm. Retrieved 16:45, October 29, 2023, from `https://dev.mysql.com/doc/refman/8.0/en/nested-loop-joins.html`

[3] sort-merge join Sort-merge join algorithm. Retrieved 18:30, November 5th, 2023, from `https://en.wikipedia.org/wiki/Sort-merge_join`