1 **Introduction**

In this paper, we describe an implementation of the join operation, natural join using nested-loop join and the block nested-loop join.

2 **Background**

In this assignment the used programming language is c. The code is written on the same files as the previous assignment.

3 **Design**

In this section a brief overview of the design is described. The goal is to make two different algorithms for the join operation and benchmarking them to see which is the most efficient.

In this assignment we implement two different algorithms to perform our natural join operation. The first one uses the nested-loop join. In this algorithm, for every field in the outer relation, we iterate through the entire table for the inner relation. We do this until we find a field that has the same name and the same type in both tables. If we do not find any in common, we cant continue with the join operation. In this implementation we assume that there's only one field in common in the two tables.

The block nested-loop join is a bit different in its two outer loops while the two inner ones are very similar to the nested-loop join, hence the name. The difference is in the two outer loops where we work with blocks. For every block that contains data for the outer relation, we iterate through the inner relation's blocks, then for every field in the outer block we iterate through every field in the contained by the inner block. A block is simply a block of data we load in to memory using pages.

To copy records over from both tables into the new join table, we loop through every record and add it's values to the new record before adding the record to the new table.

For two tables to be eligble for a natural join operation there must be at least one common field, which means that two fields, one from each table, has to have the same name and of the same data type. The resulting jointable will only have 1 instance of that field. The records copied into the jointable will only be the ones where both tables have both values in the common field.

4 **Implementation**

The nested-loop join algorithm is done by using a nested loop. We can either use two for loops or two while loops. In this implementation the duo while loops are used. We simply start with a nested while loop on the field descriptors from the two join tables. We make a copy of the schema for the left table, so that we only need to copy over the fields from the right table. In this implementation we assume that there is only one field that is in common from both tables. So we check the left table's field type and name with the right table's fields. If there is a match, we know that the two tables can be joined together, if not we simply delete the copied schema and return null.

In this implementation I did not get to implement the block nested-loop join algorithm as I unfortunatly ran out of time. The implementation would rely on having 4 loops nested, the first two outer loops would read a block on both tables, while the inner two loops would pretty much to the same thing as the nested-loop join in adding the fields etc. The blocks we read into memory would be the blocks containing data for both tables IE the records. We would then compare the two to look for any common fields, as mentioned in this assignment we assume there is only one common field. And then we would know that these two tables could be joined together.

This is how I did it to get the records copied from both tables into the new join table, this function would be called after any of the two join algorithms above finished the joining and reported that the join is possible.

To copy the records from each table we use another set of nested while loops. Here we essentially do the same as the nested-loop described above, but we loop on the get_record function which fetches the next record from a schema and places it in an input pointer. Doing this we can compare a record from the outer loop with every record in the inner loop. We assume that the common field is always in the first index in the record, so at index 0. We compare the data from left and right on index 0, if its a match we have found a record with a common value in both tables, so we copy the data over to the join table. Then we move to the next record in the outer loop, and reset the inner table's position so that we can loop through the entire table again and compare all records to find more common entries.

5 **Discussion**

Currently in this implementation there is a bug that sometimes adds multiple duplicates of a record in the join table, I have not spent time researching this issue but it happens on spesific types of tables where there are different number of fields on both tables to join.

The benchmark is done by the same number of records in both tables and the same data. This means that the system has to og through each and every one.

Records	Time
1000	0.06 s
5000	1.4 s
10000	5.6 s

As we can see when doubling the amount of records from 5000 to 10.000 we have an increase in execution time by 4 times. The nested-loop join is fastest when both relations can be fit fully in memory, which means each block only needs to be written once and the number of block transfers from disk to memory is only the number of blocks in left table + number of blocks in right table and only 2 seeks. The worst case is when we can only hold one block for each relation, increasing the number of block reads by alot.

I did not get to benchmark the block nested for loop because I could not get the function done in time. However we would assume to see a better time than the nested-loop because for each block in the inner relation is read only once for each block in the outer relation, instead of once per field in the outer relation. This is the worst case. This means that we in total have fewer block reads than the nested-loop join, as we only read each block once in the inner relation. Both algorithms have the same best case scenarios. As we can see the block nested-loop join's worst case scenario is better than the nested-loop join's worst case scenario, while the best case is the same for both.

If both tables in the join are stored in B+-tree organized files where the search keys are the common attribute(field) of the two tables, we have the advantage of the records being already sorted by the file organization. The lower values on the left side and the bigger ones on the right side. Another advantage is that its very quick to find the common field. We simply search for the current field we want to find a common field for, since the search keys are the common attribute, we will find out if we have a field in common or not quickly. As mentioned the record values would then be sorted due to the file organization, this also makes it quick and easy for us to find a record in our common field with the same value in the common field as we are searching. If we follow the link from a common field to it's data at the bottom, we can instantly see if the value we want to find is in it since it's sorted in increasing order. For this reason I would suggest using a merge join algorithm as it requires both inputs to be sorted, which they allready are in the file organization.

Comparing the merge join algorithm to the block nested-loop I can assume that the merge join algorithm would be quicker due to its sorted data requirements. The block nested-loop does not care about the data being sorted or not(altough it would still increase its execution time though), so it has no requirements regarding sorting. It reads each table's blocks and iterates through them, searching for the records and compares them. This algorithm would get slower as the file size grows, while the merge join would receive a minor increase as the file increases, due to bigger height in the b+ tree due to insertions.

6 **Conclusion**

I'm not very happy about the implementation as I did not get to finish it, namely the algorithm for block nested-loop join. Because of this I also did not get to benchmark it and compare it with the nested-loop join by numbers. But I did discuss the theoretical side of both were we concluded that the block nested-loop join would indeed be faster because it would only need to read each block once for each inner block as its worst case,while the best case scenario for both the algorithms remain the same.

Sources

[1] Database system concepts (2015), Abraham Silberschatz, Henry F. Korth, S. Sudarshan