INF-2700 DATABASE SYSTEMS

NESTED JOIN & BLOCK NESTED JOIN

ALEXANDER EINSHØJ

11.11.16

Introduction

In this paper, the implementation of the nested loop join and block nested loop join algorithms is described.

Technical Background

Nested loop

The nested loop join includes an algorithm which takes as input two different tables in a database. The output consists of all distinct fields in both of the tables, with some precautions. Let's call the first table the 'left table' and the table to be joined, the 'right table'.

All of the columns, or fields in the left table are immediately added to the resulting table. When looping through the fields in the right table, all fields are added to the resulting table, except in the case when an existing field in the resulting table is also found in the right table. This is the common value of the tables. In this paper, this is the value on which the table will be joined.

Thus, the output of a nested loop join will consist of all the values in the left table, and all the values in the right table, except the field name which is shared or equal for both tables.

The algorithm works as follows:
Assume you've got two tables, A and B, and those two tables to be joined.

```
For each tuple a in A do
        For each tuple b in B do
                If a and b satisfy the join condition
                        Then output the tuple <a, b>
```

[1] Fig 1

Block nested loop

The block nested loop join is a variation of the nested loop join. This poses as a more efficient solution to the basic nested loop join. The idea behind this algorithm is to save disk operations by a considerable amount compared to what the basic nested loop join offers.

```
for Lblock in Ltbl
  for Rblock in Rtbl
    for Lrec in Ltbl
      for Rrec in Rtbl
        if (lval == rval)
          Output tuple <lval, rval>
```

Fig2

Figure 2 shows us the algorithm in pseudocode. The main difference between the two algorithms lies with block nested loop only scanning Lblock once for every Rblock, thus matching an entire block from Lblock against an entire block from Rblock. By saving Lblock in memory, the amount of reads needed to complete the algorithm is lowered by a significant amount. There are variants of block nested loop, in which one might vary how much from Ltable is stored into memory.

Design

Fig 1 is a pseudocode version of a general nested loop join. In this assignment, however, assumptions has been made. There is an assumption that the common field between the tables is of integer type. In the current implementation, the given API has been used as much as possible, while trying to satisfy the implementation of a general nested loop join.

The algorithm iterates through each record present in table A, within the common field, and compares it to every record present in table B within the same field. The resulting table should only consist of the records in which the common value is the same for both tables. This is the essence of the join operation.

A downside to this method is that it features a lot of reads from disk, because the entire table B needs to be read for each record in table A. This is where the block nested loop join algorithm presents a more efficient solution.

Implementation

The program is implemented in C. The database creation application is implemented in Python.

Discussion

Although no tests could be done to compare the two algorithms due to the block nested loop implementation is not satisfactory yet, seen from the nature of the algorithms and how they work, it is apparent that the block nested loop join will use a significant less amount of disk reads to complete an operation. From this follows that the amount of time will also be lowered by an enormous amount.

Conclusion

The block nested loop join has been attempted to be implemented, though not successfully. The nested loop join should work as is.

References

[1] https://en.wikipedia.org/wiki/Nested_loop_join