## Chapter 4.3 Nested Loop Join

We need to pay attention to *Join Operator Algorithm* which is named as *Nested Loop Join*, the cost of Nested Loop Join is *one pass with half*. Among all kinds of algorithms, tuples of one Relation will be read for once while tuples of another Relation will be read repeatly.

Nested Loop Join can be used on Relation with any size; There is no need to store the whole Relation into the main memory.

### Chapter 4.3.1 Nested Loop Join based on Tuple

***Introduction:***

The simplest loop is aim to each tuple of Relation. For *Nested Loop Join based on Tuple* algorithm, we connect *R(X, Y) AND S(Y, Z)*:

*For each tuple s in Relation S DO:*

*For each tuple r in Relation R DO:*

*IF r connects with s and form the tuple t then:*

*output tuple t;*

*End;*

*End;*

*End;*

***Cost:***

If we do not pay attention on the block buffer method in Relation R and Relation S, then the cost of I/O will reach up to *T(R) \* T(S)*.

***Improvement:***

1. Using the index of join property to find the specific r tuple for s tuple, do not need to read the whole Relation R.
2. Pay more attention on How tuple r and s are distributed in the main memory. Then when we execute the inner loop, then use the main memory as much as we can to make the disk I/O less and less.

### Chapter 4.3.2 Nested Loop Join Iterator based on Tuple

***Principle:***

Nested Loop Join fits for Iterator. The Iterator of R Join S can be easily constructed by Iterators of Relation R and S, such as R.Open(). Assume that the Relation R and S are both empty.

***Pseudo – Code:***

*Open ( ) {*

*R.Open( );*

*S.Open( );*

*s := S.GetNext( );*

*}*

*GetNext ( ) {*

*REPEAT {*

*r := R.GetNext();*

*If ( r = NotFound ) {*

*R.Close ( );*

*s := S.GetNext ( ); // Start the next Loop.*

*If ( s = NotFound ) RETURN; // If there is no more loop, just return.*

*R.Open ( ); // Re-open the Relation R and get the next tuple r.*

*r := R.GetNext();*

*}*

*} UNTIL ( tuple r can join with s);*

*Return tuple that r joins with s;*

*}*

*Close ( ) {*

*R.Close();*

*S.Close();*

*}*

### Chapter 4.3.3 Nested Loop Join Algorithm based on Block

***Improvement:***

Calculate R join S by using Nested Loop Join Algorithm based on Block:

1. Organize the Operation Objects according to the Block of two Relation R and S. *(In order to reduce the disk I/O to read Relation R.)*
2. Using the main memory to store the tuple of Relation S, S is the Relation in outer cycle. *(Connect the tuples of Relation R with tuple of Relation S as much as possible.)*

***Assumption:***

Assume that *B(S) <= B(R)*, and assume that B(S) > M which means Relation R and Relation S can not be stored into the main memory in one time.

***Steps:***

1. Read M - 1 blocks of Relation S into main memory.
2. Construct a search data structure for the tuple in main memory. *(The key of search data structure is the Common Property of Relation R and S.)*
3. Go through all blocks of Relation R, and read each block into Mth main memory.
4. Compare all tuples of the current block of Relation R and S in main memory.
5. Output the connected tuple.

***Pseudo - Code:***

*For each chunk that sized up as M - 1 blocks in Relation S*

*Read the chunk of Relation S into main memory;*

*Organize the Common Property of Relation R and S as Search Structure;*

*For each block b in Relation R:*

*Read the block b into main memory;*

*For each tuple t in Block b in Relation R DO BEGIN:*

*Find the tuples in b that can connect with tuples in S;*

*Output the join tuple connected by two tuples in Relation R and S;*

*END;*

*END;*

*END;*

***Example:***

1. B(R) = 1000 and B(S) = 500, and M = 101. We use 100 main memory to buffer 100 chunks for Relation S, so 5 times is needed in the outer cycle. For each cycle, 100 disk I/O is needed to read the chunk of Relation S, and in the inner cycle, 1000 disk I/O is needed to read Relation R.

*(The total disk I/O equals to 5500. { 500/100 \* (100 + 1000)= 5 \* 1100 = 5500 } )*

1. *If we change the reading sequence of Relation R and S, then the total disk I/O equals to { 1000/100 \* (100 + 500) = 10 \* 600 = 6000 }*.

### Chapter 4.3.4 Nested Loop Join Analysis

***Analysis:***

1. Assume that B(R), B(S) and M, and Relation S is the smaller relation, then the outer cycle times equals to B(S) / (M - 1). Then in each inner cycle, read M - 1 blocks and B(R) block.

*(The the total disk I/O equals to B(S) / (M - 1) \* (B(R) + (M - 1) ) or B(S) \* (B(R) + (M - 1) ) /(M - 1).)*

1. If M, B(R) and B(S) are large while M is the smaller one, then the nearest total cost equals to *B(S) \* B(S) / M.*
2. Although Nested Loop Join is not the most effective algorithm, we still need to pay extra attention in early Relation DBMS, because it is the only algorithm that available.

### Chapter 4.3.5 Algorithm Conclusion So Far

|  |  |  |  |
| --- | --- | --- | --- |
| *Operator* | *M* | *Disk I/O* | *Chapter* |
| *Selection* | *1* | *B* | *4.2.1* |
| *Projection* |
| *Grouping* | *B* | *B* | *4.2.2* |
| *Deduplication* |
| *Union* | *min( B(R), B(S) )* | *B(R) + B(S)* | *4.2.3* |
| *Intersection* |
| *Difference* |
| *Product* |
| *Join = Natural Join* |
| *Join = Nested Loop Join* | *M >= 2* | *B(S) \* B(S) / M* | *4.3.3* |