## Chapter 4.1 Introduction of Physics Query Plan Operator

***Principle:***

The Physics Query Plan consists of operator, and each operator realizes one step of the plan. The Physics Operator is usually a realization of Relation Algebra Operator. But, the Physics Operator is also needed to do some other tasks which has nothing to do with Relation Algebra Operator.

***Example:***

*Scan* a table is calling each tuple of some Relation of Operation Object from Relation Algebra Expression into main memory. Of course, this relation is a Typical Operation Data of some other operations.

***Supplement:***

The concept *iterator* also needs to be introduced. It is an important method which is included in Operators which is used to transmit the request for tuples and results.

### Chapter 4.1.1 Scan Table

***Principle:***

The basic thing that we do for the Physics Query Plan is to read the whole contents of the relation R. One variant of Operator is to include a simple *Predication(谓词)*, and we only need to read the tuple that satisfied with Predication.

***Methods:*** (Locate tuples in Relation R)

1. In many situation, relation R is stored in some area of the secondary storage and the tuples are arranged in the blocks. The system knows the blocks that including the tuples of Relation R, it can get these blocks one by one. The Operation is called *Table - Scan*.
2. If there has any index on a random property of Relation R, then we can use the index to retrieve all tuples of Relation R. For instance, the sparse index can be used to guide us to all blocks that including Relation R. The Operation is called *Index – Scan*.

***Attention:***

We can get all tuples from index of all Relations, and can also get some tuples that having the specific values (or values in specific range).

### Chapter 4.1.2 Sorting when Scan Table

***Principle: (Make us Sort Relation)***

1. Query the Relation which may includes *ORDER BY*, and sort the Relation.
2. Multiple algorithms of Relation Algebra require Sorted Operation Object.

***Operation:***

*Sort – Scan* accepts Relation R and illustrations of other properties, and generates the finial sorted R.

***Methods: (To realize the Sort - Scan)***

1. Scan the property A in Relation R, (such as B – Tree index), get sorted R.
2. Relation R is small, and even can be stored into main memory, then scan table or index to get required tuples. At last, using *main memory sort algorithm*.
3. If R is too big to store into the main memory, just using *Merge Sort Algorithm*.

### Chapter 4.1.3 Calculation Module of Physics Operator

***Principle:***

* A *Query* normally includes several *Relation Algebra Operations*, and a *Physics Query Plan* includes several *Physics Operators*.
* The number of Disk I/O is used to weigh the cost of Operation. This standard is the same as one point: *The time getting data from disk is longer than from main memory.*

*Assume:*

Any operation objects of each operator is located in disk but the result of operator is in main memory.

***Principle:***

* If Final Query Result from Operator needs to be wrote back to disk, then the cost is dependent on the size of result but not how it calculated. Simply add the Cost to Overall Cost of Query. Any written back cost of the result will not influence which algorithm we choose.
* The Query Result of operator is normally not written back to Disk.
* For iterator, if the Result of Operator O1 is constructed in the main memory, and we do not need to write back to the disk, but pass to the another Operator O2, which saves us time.

### Chapter 4.1.4 Parameters to Measure Cost

***Introduction:***

1. One Parameter to present the cost of operator. If Optimizer decides to get the fastest Query Plan, then it is necessary to estimate the cost.
2. A parameter is needed to present the memory size that the operator is using and some other parameters are also needed to measure the size of operations.

***Assumption:***

1. The main memory is being divided into buffer pool, and the size of buffer pool is the same size as disk block. M presents the number of buffer pool when the specific operator executes.

*(M can be seen as whole main memory or much smaller than whole memory. Actually, M is an estimation value of buffer pool that a specific Operation can get.)*

1. The next step is to consider the parameters that the cost needs when visiting the related relation. These relations that calculate the data and distribution are always calculated, so that it can help choose the Physics Operators.

***Principles:***

Assume that each time, data in one block can be visited. Then there exist three types of Parameters B, T and V:

1. When describing the size of Relation R, then we only care about the number of blocks that include all tuples of Relation R. The number of block can be described as *B(R)* and R is Relation while B is Block. *(R is stored in block B or nearly B block.)*
2. We also need to know the number of tuples of Relation R, then we describe as *T(R)*. *(If we need the number of tuples that Relation R contains, then using T/B.)*
3. At last, we hope to take reference from the number of different values in one column. If R is one Relation, and the property is a, then *V (R, a)* is the number of different value of the whole column a in the Relation R.

*{ If [a1, a2, a3, ..., an] is one property list, then V(R, [a1, a2, a3, ..., an]) is the different number of values in the property column a1, a2, a3, ..., an. }*

### Chapter 4.1.5 I/O Cost of Scan Operator

***Principle:***

1. The Number of Disk I/O of Table – Scan Operator

* The Relation R is Cluster, then the number of Disk I/O of Table – Scan Operator is nearly to be B. In the same way, if R can be stored into main memory, then we can read R into main memory and sort, then realize Table – Scan operation, so the disk I/O is B.
* The Relation R is not Cluster, then the number of Disk I/O of Table – Scan Operator is much higher. If the relation R is distributed in other relation, then the block number when proceeding Table – Scan is the same as the number of tuple in R, which equals to T. (If sort is needed, then the number of disk I/O equals to T.)

1. The Number of Disk I/O of Index – Scan Operator

* The block number that one index needs of Relation R is much less than B(R). So scan the whole R needs more I/O operation than scan the whole index.

***Attention:***

Using B or T as the cost estimation when visiting whole Cluster or non – Cluster Relation by visiting index.

### Chapter 4.1.6 The Iterator that Realizes Physics Operator

***Principles:***

Many Physics Operators can be realized as Iterator. The Iterator is the collection of three methods which can enable the user to get the tuple once a time. The three methods that form one Operation Iterator are:

1. *Open()*
2. Start the process to get tuple, but not really get it.
3. Initializes all data structures which are needed in the initialization process.

***Attention:***

The method Open() is called for each Operation Object.

1. *GetNext()*
2. Return the next tuple as result.
3. Adjust data structure in order to get the follow - up tuple.
4. If no more tuple can be returned, then return the special value *NotFound*.

***Attention:***

When getting the next tuple from the result, then GetNext() method is called on the Operation Object for one or more times.

1. *Close()*
2. End after all tuples are acquired.

***Attention:***

The method Close() is called for each Operation Object.

*(When describing Iterator and their functions, we assume that each kind of Iterator has a class, this so called ‘Class’ will define methods Open(), GetNext() and Close().)*

***Example 1:***

*Description:*

Take the Table - Scan Operator as an example. In the Query Plan, we use Instance Object of TableScan Class which is parameterized by the Relation R.

*Assumption:*

1. R is the Relation which is clustered in one block and can be visited conveniently. *(Getting next block from relation R can be realized by Storage System.)*
2. There exists one *Tuple Catalog* which enables us to visit the next tuple in the block easily or estimate whether we has reached the last tuple.

*Explanation:*

*Assumption:*

1. Block pointer b and tuple pointer t that points to the pointer b.
2. These two pointers b and t can exceed the last block or the last tuple.

*Pseudo - Code:*

Open() {

b := the first block of relation R;

t := the first tuple in the first block;

}

GetNext() {

IF ( t has exceeded the last tuple in the block b ) {

Put forward of the block b;

IF ( no next block ) {

RETURN NotFound;

}

t := the first tuple of the block b;

}

oldt := t; // Keep the old tuple t;

t := the next tuple of block b;

RETURN oldt;

}

Close() {

}

*Attention:*

1. Pseudo - Code describes three basic methods of Iterator.
2. Method Close() does nothing but clear the inner structure of DBMS.
3. Method Close() notifies Buffer Manager that some buffers are no longer needed any more or notifies Concurrency Manager that the Read Operation has been finished.

***Example 2:***

*Description:*

Take the Sort - Scan as an example. When we read the tuple of Relation R and return them according to the sorted sequence. Before we check each tuple, even one tuple can not be returned.

*Assumption:*

1. The Relation R is small and can be stored into the main memory directly.

*Explanation:*

1. Method Open() puts the whole Relation R into main memory and sorts the tuple of Relation R. After that GetNext() returns each tuple sequentially.
2. Method Open() will not sort the whole Relation R, but the method GetNext() choose the first tuple to return which means it executes Selection Sort Algoritm.

***Example 3:***

*Description:*

Considering how one Iterator can call other Iterators. R UNION S, here we generate all Relation R, then generate all Relation S, here we do not consider whether there is any duplication. Using R and S to present the Iterator of Relation R and S. Here, R.Open(), R.GetNext() and R.Close(), are the same in the Iterator S, S.Open(), S.GetNext() and S.Close().

*Pseudo - Code:*

Open() {

R.Open();

CurRel := R;

}

*( Recursive Call )*

GetNext() {

IF (CurRel = R) {

t := R.GetNext();

IF (t <> NotFound) {

RETURN t;

}

ELSE {

S.Open();

CurRel := S;

}

}

Return S.GetNext();

}

Close() {

R.Close();

S.Close();

}