## Chapter 5.7 Finishing Selection of Physics Query Plan

We already analyzed Query and converted it into Initialized Logical Query Plan, Through enlargement, we can use the same technology to Sort for Multi - Union, Multi - Intersection or Sort for random Combination or Exchange or Sort.

We need several steps to change Logic Plan into the whole Physical Query Plan. Here we still need to include several principal points:

1. In some early steps such as *Dynamic Programming Join Sequence*, and we has not selected the Algorithm, so we need to choose the *Execution Query Algorithm*.
2. For about when the Intermediate Result would be ***Materialized*** *( Materialized means to stored and saved in the Disk. )* and when it would be ***Pipeline Operation*** *( It only created in Main Memory and no need to be saved completely. )*
3. The ***comment*** *for Physical Query Plan Operator*, it needs to include Access Detail to the Stored Relation and the related Algebra Operator Execution Detail.

In this chapter, we would not include all Operator Selection but only include two most important Operators: *Select and Join*. Whether to *Materialize* them or *Pipeline Operators*. Also, we gives *Comments of Physical Query Plan*.

### Chapter 5.7.1 Select one Selection Method

The Most Important Step of Choosing one Physical Query Plan is to choose the Algorithm for each Selection Operator.

***Review:***

Before, we mentioned *Selection(c)( R ), which means to Select the Condition C and Check which tuple would satisfies the condition.* And after that we consider that Condition C would be Attribute equals to Constant and for this property we had the possibility to have one index. If so, we can find all tuples that satisfies Condition C, but no need to check all Relation R.

***Introduction:***

*Enlarge the question, and assume that we have one Selection Condition, it has several Conditions AND. Assume that at least One Condition is the format of A Theta c, among which A is the Attribute With Index, c is Constant and one comparable Operator =, <, >.*

1. *Each Physical Plan would use some attributes, which:*
2. *Has one Index.*
3. *Compare with one Constant.*

*After we use these indexes to identify the Tuple Collection which satisfies every condition. (We have discussed before that read the tuple from disk, how could we find all tuples that satisfy all conditions.)*

For convenience, we consider the Algorithm limited to conditions below:

1. Consider Index Scan Physical Operation, for those all tuples that satisfies the condition a) to query and get all tuples.
2. Consider each index in condition one and check whether it has satisfied the remained condition. *This kind of physical Operation is called Filter.*

*Except this kind of Physical Plan, we still need to consider do not utilize any indexes, but it does read all Relations and pass every tuple to the Filter to check whether the condition is satisfied the plan or not.*

What we need to do is to estimate the cost of each possible selection, and decide which possible Physical Plans can be used.

***Selection Principal:***

For convenience, here we only consider the cost to access the data block but not the index block. Here is the outline of the estimated cost for different estimation. *Assume that Operation is Selection (c) (R), and the condition C is one or more AND.*

1. *Table Scan Algorithm combine with one Filter*, and the cost equals to:
2. If Relation R is aggregated, then it is B(R).
3. If Relation R is not aggregated, then it is T(R).
4. Select the Equal Value Option, *just as a = 10, exists the index about attribute equals to 10*, and use the index scan to find out all matched tuples, and then tuples are going to be filtered to check whether they are satisfied with all condition C, so the cost equals to:
5. If Index is aggregated, then the cost equals to B(R)/V(R, a).
6. If Index is not aggregated, then the cost equals to T(R)/V(R, a).
7. Select *Non-Equal Value Option, just as b < 20, then exists index about attribute b*, and use Index - Scan to search and match tuples, and then filter the scanned tuples to check whether they are all satisfied Condition C, the cost of the plan are:
8. If Index is aggregated, then the cost equals to B(R)/3.
9. If Index is not aggregated, then the cost equals to T(R)/3.

***Example:***

Consider the Selection x=1 AND y = 2 AND z < 5 ( R ), among which Relation R(x, y, z) has parameters below: *T(R) = 5000, B(R) = 200, V(R, x) = 100, and V(R, y) = 500. Besides, assume that Relation R is aggregated, and all x, y and z have indexes, only index z is aggregated.* Below are all options:

1. Scan table and filter the table. Its cost equals to B(R), since Relation   
   R is aggregated.
2. Index by using x = 1 and find all tuples x = 1, after that using filter to check y = 2 and z < 5. The cost would be T(R) / V(R, x) = 5000/100 = 50.
3. Index by using y = 2 and find all tuples y = 2, after that using filter to check x = 1 and z < 5. The cost would be T(R)/V(R, y) = 5000/500 = 10.
4. Index by using z < 5 and find all tuples z < 5, after that using filter to check x = 1 and y = 2. The cost would be B(R)/3 = 200/3 = 67.

The least cost is the third one, of which the cost equals to 10 times disk I/O. The Best Physical Plan is to search all tuples that satisfy y = 2 and then filter other two conditions.

### Chapter 5.7.2 Select Join Method

### ***Background:***

### In Chapter 4 has introduced the Cost related with all kinds of Join Algorithm. Assume that we know the Buffer Volume that are available for Join Operation, then we need to figure out to choose which formulate that relates with *Sort Join*, *Hash Join* and *Index Join*.

If we can not make sure or figure out the available Buffer Area for Query Operation, or we do not have important size parameters, such as V(R, a), there still have some principle to let us choose one Join method. The same thinking adapts to other Binary Operators, such as Union, Completion Relation, Unary Operator, Grouping and De-Duplication.

Method:

* One method is to call *One - Trip Join*, hope the Buffer Area Management can give enough Buffer Areas for Join, or close the Buffer Area Management and Jolt is not the key Cost. Another Substitution is to select one *Nested Loop Cycle Join* and hope if can not ensure to assign enough Buffer Area for Left Parameter, and store into main memory, then the parameter would not be divided into too many slices, and Join result is still reasonable and effective.
* Choose *Sort Join* when random one of these two points would be satisfied:

1. One or two Parameters have already been sorted based on Join Attributes.
2. For the same Attributes that have two or multiple Join, such as

*( R(a, b) Join S(a, c) ) Join T(a, b)*

Here, Sort Attribute a for Relation R and S would cause Attribute a being sorted in final Result R Join S, and in the second Sort Sequence, it would be used directly.

* When one Join, *R(a, b) Join S(b, c)*, among which Relation R is relatively small, and there has one index in Join Attribute S.b, then we should choose *Index Join*.
* If we have no chance to use Sorted Relation or Index, and needs multiple Join, then *Hash Join* maybe the Best Option, since *scan times depend on the size of smaller parameter but not two parameters.*

### Chapter 5.7.3 Pipeline Operation and Materialization

### ***Definition:***

* *Materialization* - The Original Method of Query Plan is to sort Operation and *save intermediate result of every result on the Disk until it is required by another Operator*. This strategy is called Materialized, since each Intermediate Relationship would be Materialized in the Disk.
* *Pipeline* - The more effective and elegant method is to *run several Operators simultaneously*. *The tuple that calculated by one Operator would passed over to the next Operator directly, and no need to save the intermediate tuples in the Disk.* This method is called Pipeline Operation. Generally, it is executed by one Iterator Network, and the method of Iterator Network would call each other at the appropriate time.

* *Pipeline Advantage and Defects* - Pipeline saves Disk I/O, but it also has defects. For any time, several operators need to *share memory*, then it is possible to *select the specific algorithm with higher Disk I/O requirement*, otherwise *‘Jolt’* would happens, and further it would *waste all Disk I/O saved by Pipeline Operation*, and even more.

***Materialization in Main Memory***

Imaging there has a method between *Pipeline Operation* and *Materialization*, and *before the whole result in this method that is passed to the consumption is stored in the Main Memory*. This kind of Operation is called *Pipeline Operation*. The first thing that consumption needs to do is to organize the most part of the whole Relations.

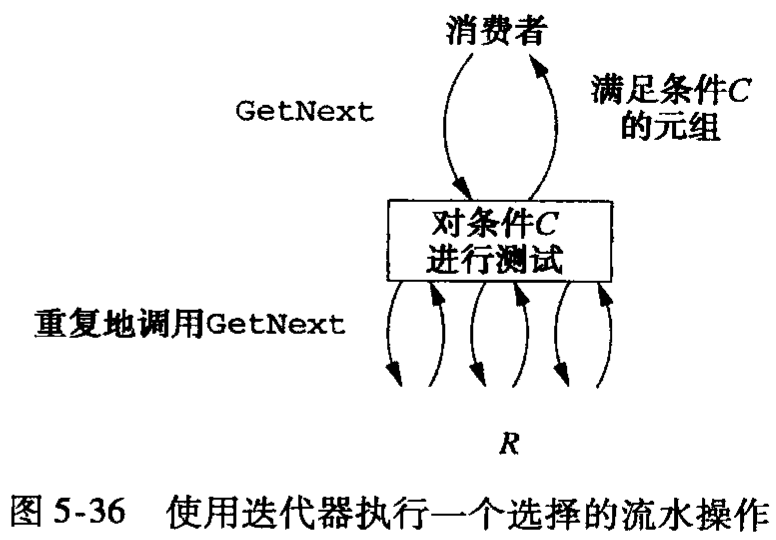
### Chapter 5.7.4 Unary Pipeline Operation

***Description:***

*Unary Operations - Selection and Projection - are both great Selection Operators in Pipeline Operation.* Since these Operators are all Unary Operators, we do not need any Input and Output of Multi - Blocks.

***Process:***

We execute Unary Pipeline Operation by Iterators. Each time when we need to use other tuple, the consumer only needs to call *GetNext() operator*.



* Under such situation of Projection, we need to call *GetNext()* on all tuples, and also operate Projection Operator to that tuple, return the result to consumer.
* For Selection Operator *Selection (c)*, we need to call *GetNext()* for several times, until find out a tuple that satisfies *condition C*.

### Chapter 5.7.5 Pipeline Operation of Binary Operators

### ***Definition:***

*Binary Operator can be the Pipeline Operation.* We use one Buffer Area to pass the result to Consumer, and one time one block. However, the number of Calculation Result and Consumption Result are totally different, they depends on the size of result and the size of parameters.

***Example:***

Here, we use Physics Query Plan below:

*R(w, x) Join S(x, y) Join U(y, z)*

Assume we have below conditions:

1. Relation R occupies 5000 blocks; Relation S and U occupies 10, 000 blocks.
2. For some k, the intermediate result R Join S would occupy k blocks.
3. Execute two Join into Hash Join, or just One Trip or Two Trip, these all depends on variable k.
4. There have 101 available Buffer areas.

***Procedure:***

At first, consider Relation R Join S. There have not got any Relations in main memory, so we need to take one Two Trip Hash Join. If the smaller Relation is divided into 100 blocks in first Trip at most, each block occupies 50 blocks. If there has 50 blocks in Bucket of Relation R, in the second Trip Relation R Hash Join S would use 51 Buffer Areas, among which 50 buffer areas are used for Join the result of R Join S and Relation U.

***Situation One:***

Assume that k <= 49, which is to say the result of Relation R Join S would occupy 49 blocks at most. So we can Pipeline the result of R Join S into 49 Buffer Areas, and organize it to check as Hash Table. Also, we need to use one Buffer Area to read each block of Relation U. We can also execute the second Trip as One Trip Join.

*The total number of Disk I/O is:*

1. *45,000, execute the Two Trip Hash Join for Relation R and Relation S.*
2. *10,000, execute One Trip Hash Join for Relation ( R Join S ) Join U.*

*The total Disk I/O equals to 55,000.*

***Situation Two:***

Assume that k > 49, but k <= 5000. Then we can still Pipeline Operation on Relation R Join S, but here we need another strategy, in this strategy, this Relation would Hash 50 Buckets Join with Relation U.

1. Before we start Relation R Join with S, we need to put Relation U by 50 Buckets 200 Blocks each.
2. Next, we need to use 51 Buckets to Hash Join for Relation R and S, but when each tuple generated by Relation R Join Relation S, then we put it into Left Bucket remained. When the Buffer Area has been filled with Buckets, then we write them back to Disk.
3. At last, we need to Join Relation R, S and U bucket by bucket. Here, the Bucket R Join S is used as Construction Usage while Relation U is used as the Query Usage.

*The total Disk I/O for Pipeline Operation is:*

* *20,000 is used to read and write Relation U back to the Bucket.*
* *45,000 is used to execute Two Trip Hash Join.*
* *k is used to write out the Bucket of Relation R Join S.*
* *k + 10,000 is used to read and write the final bucket of Relation R Join S Join U.*

The total Cost equals to 75,000 + 2 \* k. Attention when k turns from 49 to 50, then One Trip Join would turn to Two Trip Join.

***Solution Three:***

Assume that k > 5000. Then we would do below:

1. Hash Join Two Trip Hash Join and calculate Relation R Join S, and all result would be stored into Disk.
2. Join Relation R with S and U, and we still use the Two Trip Hash Join. Attention that B(U) = 10, 000, then we can use 100 buckets to execute the Two Trip Join, but do not consider the size of value k.

*The total Disk I/O:*

* *45000, used to Two Trip Join Relation R and S.*
* *k is used to save the result of R Join S on Disk.*
* *30000 + 3k, which is used to on Relation U Two Trip Hash Join with S.*

The total Cost equals to 75,000 + 4k. The table below gives the integrity Join Plan.

|  |  |  |  |
| --- | --- | --- | --- |
| *Range k* | *Pipeline or Materialization* | *Last Join Algorithm* | *Total Disk I/O* |
| *k <= 49* | *Pipeline Operation* | *One Trip* | *55 000* |
| *50 <= k <= 5000* | *Pipeline Operation* | *50 Buckets, Two Trips* | *75 000 + 2k* |
| *k > 5000* | *Materialization* | *100 Buckets, Two Trips* | *75 000 + 4k* |

### Chapter 5.7.6 Sign used for Physics Query Plan

### Explanation:

### Leaf Operator:

Selection Physics Operator:

Physics Sort Operator:

Other Relational Algebra Operator:

Example:

Example:

### Chapter 5.7.7 Sort of Physics Operator