## 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

### Chapter 5.7.4 Unary Pipeline Operation

### Chapter 5.7.5 Pipeline Operation of Binary Operators

### Chapter 5.7.6 Sign used for Physics Query Plan

### Chapter 5.7.7 Sort of Physics Operator