## Chapter 5.4 The Estimation of Calculation Cost

***Definition:***

After we analyze the Grammar of one Query and convert to one Logic Query Plan, the next step is that we must convert the Logic Query Plan to the Physical Plan.

Normally, when we check multiple Physical Plan and evaluate each Physical Plan, or just evaluate the cost of this conversion. After through this kind of evaluation, normally is called *Enumeration based on Cost*, however we choose the Physical Query Plan with the least estimation cost.

For each Physical Plan, we choose:

* For each Law that satisfied the *Combination Law* and *Distribution Law*, for example decide the sequence of Join, Union, Intersection and Grouping Operator.
* For Algorithm of each Operator in the Logical Plan, for example, decide whether to use *Nested Loop Join* or *Hash Join*.
* For other Operators, such as Scan, Sort and so on, they are required in the Physical Plan but *not visible* in Logical Plan.
* The method that pass one parameter to the next parameter, such as, through using *the intermediate result* or through using *iterator*.Then pass one tuple of one parameter each time or one main memory buffer area.

In order to make each selection, we need to know the cost of each Physical Plan. Also if there has no execution plan, we can not know it’s cost. However, executing one Query Project is much more than select a Project. So, in order to not execute multi projects for one Query, this forced us to estimate the cost of this plan.

Of course, before we discuss the Physical Plan Enumeration, we must consider how could we estimate the cost of these Plan.

### Chapter 5.4.1 The Estimation of Intermediate Relation Size

***Definition:***

Here we choose from the Physical Plan is to minimize the estimation cost when executing Query. Here, no matter we use which method to execute the Query Plan, and no matter how the cost of Query Plan is estimate, then the size of intermediate Relation has big influence on the cost.

Normally, we can use the rule to estimate the tuple numbers in the Intermediate Relation, make these rule:

1. Give the Exact Estimation.
2. Easy to Calculation.
3. Logical Consistent, find out one way which is to say the estimation of one intermediate Relation Size do not depend on this Relation. *(For example, the estimation size of multi Relation Join should not depend on the sequence when we calculate.)*

***Supplement:***

We need to mention that, there do not exist one method which can satisfy all these three conditions. Here we give one simple rule that satisfies most situation.

*Luckily, here we just want to help find one Physical Query Plan but not estimate the accurate size.* Although this inaccurate estimation do has some error with the actual estimation, but as long as it help decide the final best Physical Query Plan, then we can let this tiny error go.

### Chapter 5.4.2 The Estimation of Projection Operator

***Instruction:***

Here we makes the traditional De - Duplication Projection as the Package Projection followed with the De - Duplication Operator. Normally, the Projection Operator does not the same as other Operators, since the size of result can be calculated precisely. If the size of Projection decreased, then just because of some of the fields are eliminated. However, enlarge Projection Operator will cause generate the new fields, and they are the combinations that had before, under this circumstance, Projection Operator actually adds the size of Relation.

***Example1:***

Assume that Relation R(a, b, c) is one relation, among which a, b are the integer that occupies 4 bytes length, c is the string with 100 bytes length. Also assume that the tuple head needs 12 bytes. Then each tuple of Relation R will occupies 12 + 4 + 4 + 100 = 120 bytes.

Assume that the block is 1024 bytes length, and among this the block head occupies 24 bytes. ( 1024 - 24 ) / 120 = 8, then each block can be used to store 8 tuples. By the way, assume that T(R) = 10 000, which is to say, there has 10 000 tuples in Relation R, then B(R) = 10 000 / 8 = 1250.

***Example2:***

Consider the S = Projection a + b -> x, c (R), then we use the sum of a and b to substitute the a and b. The tuple of Relation S occupies 4 + 100 + 12 = 116 bytes. Although the total bytes of tuple of Relation S is much smaller, but we can still store 8 tuples in one block. Therefore, T(S) = 10 000 and B(S) = 1250.

***Example3:***

Now let’s consider U = Projection a, b (R), then in this Relation, the tuple occupies 4 + 4 + 12 = 20 bytes. Also T(U) = 10 000, then (1024 - 24 ) / 20 = 50 can be stored in one block. B(U) = 10 000 / 50 = 200.

### Chapter 5.4.3 The Estimation of Selection Operator

***Background:***

When we execute Selection Operator, then normally, the number of tuples are decreased, although the size of tuples stay original.

The simplest Selection Situation is when one Attribute equals to one *Common Variable*, then there has one simpler way to estimate the size of Result.

However the premise is that we know or we estimate that the attribute can get different value.

***Estimation:***

Make S = Selection (A = C) ( R ), among this A is the attribute of Relation R, also c is one common variable.   
***Equivalence Situation:***

* T(S) = T(R) / V(R, A)

*(Here, V(R, A) is the tuple number of Relation R when the tuple equals to A.)*

***Non - Equivalence Situation:***

Make S = Selection (A < variable) (R), then somebody may estimates that T(R)/2 is the size estimation of T(S). But, normally, we suggest that *one - third number of tuples* will be returned.

* T(S) = T(R) / 3

***Non - Equal Situation:***

Make S = Selection (A != variable) (R), then we suggest that all tuples are satisfied the situation, which is to say, T(S) = T(R) as the estimation.

* T(S) = T(R) (V(R, a) - 1) / V(R, a)

*(Here, since we admit that there has 1 / V(R, a) tuple of Relation R doesn’t staisfy the situation. So here is the explanation.)*

***Multi - Condition Situation:***

When the Selection Conditions consists of multi - AND with Equivalence or Non - Equivalence Situation, then we can just use the simple Selection Cascade, among it just check one of the condition each time.

*(Attention that the result equals to the factor multiple the original size of Relation which is just as the described before.)*

***Example1:***

R(a, b, c) is one Relation, S = Selection a = 10 AND b < 20 (R), and T(R) = 10 000, V(R, a) = 50. Therefore the estimation of T(S) = 10 000 / ( 50 \* 3 ) = 67.

***Contradiction In Example1:***

Consider that S = Selection a = 10 AND a > 20 (R), after that, according to the rule, then T(S) = 10 000 / ( 3 \* 50 ) = 67. But here, we can analyze that there has no tuples can satisfy both conditions, a = 10 and a > 20.

*(Under this situation, the optimizer can be used to find the Selection Condition that satisfies the logical value equals to FALSE and use the Empty Set to substitute S.)*

***OR Situation1:***

When the situation relates with OR condition, such as *S = Selection c1 Or c2 (R)*, then the size of result will be harder to decide.

One simple assumption is that there have no tuples that satisfy both condition, therefore the result will be the sum of two conditions that satisfy each condition.

Of course, this will cause *Overestimation*, but the truth sometimes will cause the wrong result: The tuples in Relation S will be much more than in Relation R.

***OR Situation2:***

One more complex but much more precise estimation is that, assume that c1 and c2 are all independent:

S = Selection c1 OR c2 (R), then here, if there has n tuples in Relation R and among which m1 satisfies the c1 and m2 satisfies the c2, then we estimate that the tuple number of Relation S will be: *n \* ( 1 - (1 - m1/n)(1 - m2/n) )*

***Here is the explanation:***

* (1 - m1/n) is the partition that does not satisfy Condition c1.
* (1 - m2/n) is the partition that does not satisfy Condition c2.
* The product of these two is the number of tuple that does not satisfy the condition in Relation S. 1 minus this product is the number that belongs to the Relation S.

***Example2:***

Assume that in Relation R(a, b), T(R) = 10 000 tuples, also

S = Selection (a =10 OR b < 20) (R), here V(R, a) = 50, then the tuples that satisfies a = 10, we estimates that will be 200, which is to say T(R) / V(R, a). Then the tuples that satisfied b < 2, we estimates it will be equals to T(R) / 3 = 10 000 / 3 = 3333.

Here we simply estimates of the size of S will be the sum of these two value = 3333 + 200 = 3533. Based on the independent condition of a = 20 and b < 20, the much more complex estimation is 10 000 \* (1 - (200/10 000) (1 - 3333/10 000) )= 3466. In this example, there has no much difference between two estimation.

***Attention:***

It is much likely that the last operator in the Selection Operator may be *NOT*. The estimation number of tuple in Relation R that satisfies *NOT C* will equals to *T(R) - Number of Tuples (That satisfies condition C.)*

### Chapter 5.4.4 The Estimation of Join Operator

***Perquisite:***

Before we do any investigation, we need to assume that Natural Join within two Relations just involve Equivalence Comparison of two attributes, which is to say, we do some research on Join R(X, Y) Natural Join S(Y, Z), but normally, we assume that *the attribute Y is the single attribute and X, Z can represent any attributes*.

***Assumption:***

*(Here we do not know How Y value in Relation R connects with Y value in Relation S.)*

* Two Relations maybe has the non - intersection Y - value collection, under this situation, Join is the empty Set and T(R Natural Join S) = 0.
* Y maybe the Main Key of Relation S and the Foreign Key of Relation R, therefore, each tuple in Relation R will connect to one tuple in Relation S, and T(R Natural Join S) = T(R).
* Almost all tuples in Relation S and R have the same Y - value, under this situation, T(R Natural Join S) = T(R) \* T(S).

### Chapter 5.4.5 Natural Join of Multi - Join - Property

### Chapter 5.4.6 Join of Multi - Relation

Chapter 5.4.7 The Estimation of Other Operators