## Chapter 5.6 Selection of Connection Sequence

In this Chapter, we take key problem into consideration: Choose Sequence for Join which relates to more than three Relations.

### Chapter 5.6.1 The Meaning of Left and Right Connection Parameters

***Introduction:***

When choose the sequence for Join, then we need to remember that Join Operator is mostly not symmetrical, therefore, seen from this kind of meaning, two represented Relations are totally different, the Join cost depends on which Relation represents which meaning.

***Example:***

For one trip Join, it read the smaller Relation into Main Memory, and form one type of structure, it called Hash Table, therefore it can be used conveniently to match tuples from other Relation. Then read other Relation, one block for each time, and start Join Operator for tuples in the Block and in the Main Memory.

***Definition:***

*Prerequisite:*

When try to choose one Physical Plan, we decide to use *One-Trip Join*.

* Choose the Smaller Relation and save it into Main Memory, then this kind of Relation is called *Construction Relation*.
* For Right Join Parameter, it is called *Query Relation*, and we try to match tuple in the Block with the already Saved Tuple in Main Memory.

***Here other Join Algorithm that are divided by parameters, included:***

1. Inner Loop Join, in that kind of Join, the left Parameter is the outer Loop Relation.
2. Index Join, in that kind of Join, there should have index in the Right Parameter.

### Chapter 5.6.2 Connection Tree

***Situation:***

When we try to Join two Relations, then we need to sort all Parameters. According to the normal custom, we need to choose the smaller Relation as the left parameter. The size of each parameter is important and has different meaning. This is normal, since the Query that relates to Join needs to choose at least one Attribute, and the Selection Operator makes estimation decreasing.

***Example:***

*For Query:*

SELECT movieTitle

FROM StarsIn, MovieStar

WHERE starName = name AND

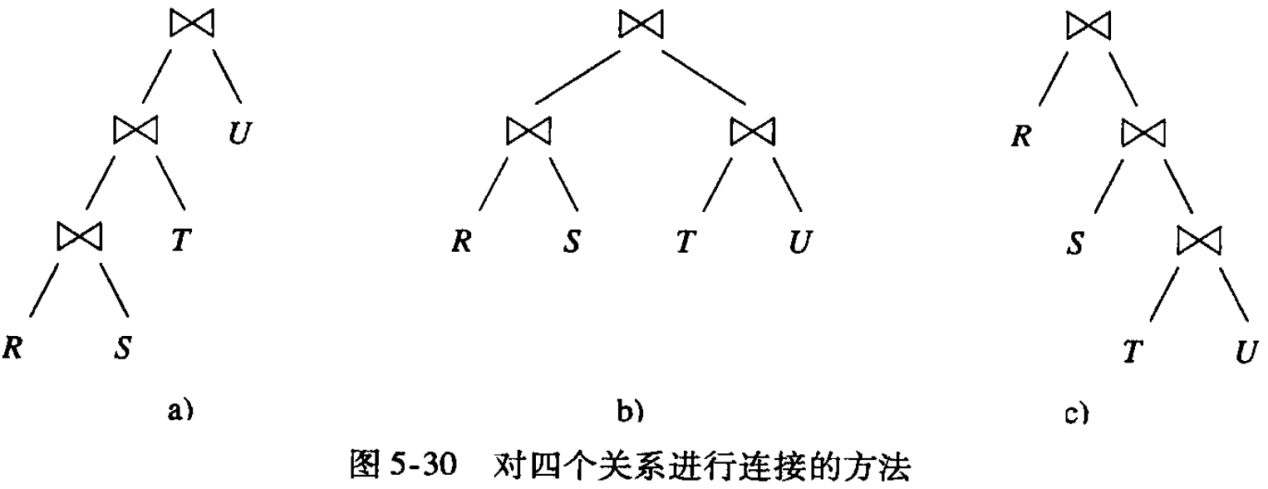
Birthdate LIKE ‘%1960’;

This is an example that we try to Join Relations include StatsIn and MovieStar, but there have not include any estimation size of Relations. We can think that one Selection will get almost 1/50 tuples among Relation MovieStar.

But since each Movie will include several Movie Stars, therefore we can assume that the size of StarsIn is far bigger than the size of MovieStar, and the second parameter of Join Operation *Selection ( birthdate LIKE ‘%1960’ ) MovieStar* is far less than the first Relation StarsIn.

As the result, we choose the MovieStar as the left Parameter which is different from the connection tree in 5 - 24.

When there have two Relations, and we can only choose one Relation as the left Parameter. When the choice relates to more than two Relations, and the number of Connection Tree will be increased quickly. For four Relations R, S, T and U, we try to connect these four trees and the possible tree structure as below. Normally, we will get 24 different trees.



### Chapter 5.6.3 Left Deep Connection Tree

### ***Definition:***

If Every Right Child in the Binary Tree, then it is Left Deep Connection Tree. Also, If Every Left Child in the Binary Tree, then it is Right Deep Connection Tree. There also has the situation that all trees are not the Left Deep Tree and not the Right Deep Tree.

***Advantage:***

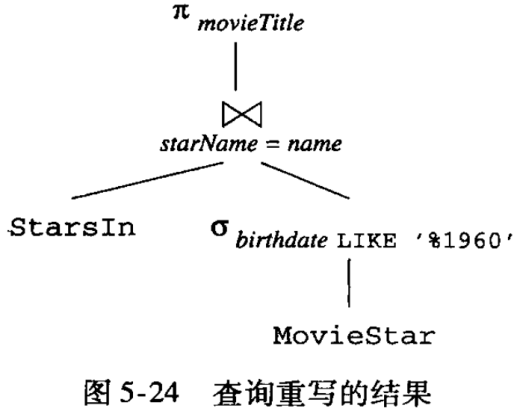
We will discuss advantages for Left Deep Connection Tree:

* For all possible Left Deep Connection Trees with given number of Tree Leaves, it maybe very big but it not such big like all other trees. So if we limit Query into Left Deep Connection Tree, then Query Plan can be used for bigger Query.
* Left Deep Connection Tree can be used well to interact with Normal Join Algorithm, especially Inner Loop Join and One - Trip Join. The Query Plan based on Left Deep Connection Tree would be more effective than non - Left Deep Connection Tree.

Actually, one tree node in Left Deep Connection Tree and Right Deep Connection Tree can be all Inner Tree Nodes with all other Operators except Join Operator.

***Example:***

Take the 5-24 as an example, this is actually Left Connection Tree, although the Selection Operator has been applied on the Right - Hand.



Here, for the Multi - Join with given numbers Relations, the number of Left Deep Connection Tree will not increase as fast as all other trees.

***Calculation:***

* For n Relations by using the Left Deep Connection Tree, we can have only one Type, and we can use n! to assign all Relations.
* For n Relations by using the Right Deep Connection Tree, the number will be the same as the Left Deep Connection Tree.
* For n Relations, the number of Tree Shape with n Relations can be decided by using Recursion below:

*T ( 1 ) = 1*

*T ( n ) = T ( i ) \* T ( n - i ) ( i ranges from 1 ~ n - 1. )*

***Explanation for Calculation Expression above:***

* For the second Equation, we can choose i from 1 to n - 1 randomly as the tree leave numbers and these leaves can be arranged by using the random arrangement method with i tree leaves, actually this can be selected from T ( i ) arrangement.
* Similarly, for the left n - i tree leaves, we can use one of T ( n - i ) method to decide it’s arrangement.

***Value of T ( n ):***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | 5 | 6 |
| T ( n ) | 1 | 1 | 2 | 5 | 14 | 42 |

Here, we use T ( n ) \* n ! to decide all Tree numbers by using Tree Leaves. Therefore, 6 tree leaves and Tree Number with all tree leaves are signified is 42 \* n !, which is to say, 30 240, among which 6 ! is the number of Left Deep Tree, also 6 ! is the number of Right Deep Tree.

### Chapter 5.6.4 Choose Connection Sequence and Grouping by Dynamic Programming

### Chapter 5.6.5 Dynamic Programming with Specific Cost Function

### Chapter 5.6.6 Choose Connection Sequence by Greedy Algorithm