## Chapter 5.2 Algebra Law used for Query Project Improvement

This chapter will show some Algebra Laws and these laws are used to convert one Expression Tree to another equal Expression Tree, and the latter may has the more effective Physics Query Plan.

*The result of applying these Algebra Expressions is the Logical Query Plan, it is the output of Query Rewrite Phase.*

### Chapter 5.2.1 Commutation Law and Association Law

***Definition:***

1. Commutation Law: The results are always the same even there have some sequential changes of the parameters.
2. Association Law: The calculation can start from the left, also it can start from the right.

***Laws:***

Multi - Operators of Relation Algebra satisfy the Commutation Law and Association Law.

* R \* S = S \* R; (R \* S) \* T = R \* (S \* T)
* R join S = S join R; (R join S) join T = R join (S join T)
* R union S = S union R; (R union S) union T = R union (S union T)
* R intersection S = S intersection R; (R intersection S) intersection T = R intersection (S intersection T)

*(Attention: These laws are established for Set and Package.)*

***Example:***

Verify the Commutation Law: R join S = S join R:

* Assume that the tuple t exists in the result of R join S, which is to say tuple t exists in the left expression. Then tuple r must exists in Relation R, and the tuple s exists in Relation S, they must be have the same value on the common property t. Therefore when we calculate the right expression S join R, then tuple s and r will combine as the tuple t.
* Because our Relation Algebra is a package, but not set, so we must verify that if tuple t appears in the left for n times, then t should also appears in the right for n times.
* Assume that tuple t appears in the left for n times ,then tuple r in Relation R must appears for nr times, while tuple s in Relation S must appears for ns times, nr \* ns = n.
* When we calculate the right expression S join R, tupe s should appear ns times, tuple r should appear nr times, then we can get nr\*ns times t copies, then n tuple t.

***Supplement:***

Theta Join is changeable. R join c S = S join c R, as long as the condition is meaningful, then Theta Join also satisfies the Association Law.

***Example:***

Assume that we have three Relation R(a, b), S(b, c), T(c, d), the expression:

[ R join (R.b > S.b) S ] join (a < d) T

Here we can not calculate Relation S join T first, since attribute a and d do not belong to Relation S and Relation T. So when we use the Theta Join, we need to pay attention to it.

### Chapter 5.2.2 Law Related with Selection

***Principle:***

Since the Selection Operation can be used to decrease the size of Relation, so the most important rule to process the effective query is that as long as we do not change the result of expression, then we can move the Selection Operation down as lower as we can.

*(Push Down Selection is the main method to operate Query Optimizer.)*

***Law:***

The first two laws that relates with Selection Operator is the Decomposition Operation.

* *Selection (c1 and c2) (R) = Selection c1 (Selection c2 (R))*
* *Selection (c1 or c2) (R) = (Selection c1 (R)) union (Selection c2 (R)) (R is the package, since if R is set, then the duplicates will not be removed.)*
* *Selection c1 (Selection c2 (R)) = Selection c2 (Selection c1 (R)) (The Sequence of c1 and c2 is flexible, normally we can exchange the Sequence of c1 and c2.)*

***Example:***

R(a, b, c) is a Relation. Then

* *Selection (a = 1 OR a = 3) AND (b < c) (R) =>*
* *Selection (a = 1 OR a = 3) [ Selection (b < c) (R) ] =>*
* *Selection (a = 1) [ Selection (b < c) (R) ] Union Selection (a = 3) [ Selection (b < c) ]*

*(For the division of OR operator, it requires that the parameter is set and use Union.)*

*The Other Way Around is:*

*Selection (b < c) [ Selection (a = 1 OR a = 3) (R) ] =>*

*Selection (b < c) { [ Selection (a = 1) (R) ] Union [ Selection (a = 3) (R) ] }*

***Instruction:***

Other law related with Selection permits us to proceed Push Down Selection for Unary Operator: Product, Union, Intersection, Difference, Join. There have three types law, which can be decided by whether each parameter can be chosen or a must:

1. Union, Selection Operator must be pushed down to two parameters.
2. Difference, Selection Operator must be pushed down to the first parameter, and for the second parameter can be chosen.
3. For other Operator, only require that Selection Operator be pushed down to one of the parameters.
4. For Join and Product, there is no meaning to pushed down Selection Operator to two parameters, since the parameter can have or not have the selected attribute.

***Law:***

* *Selection c (R Union S) = Selection c (R) Union Selection c (S) => Union*
* *Selection c (R - S) = Selection c (R) - (S) => Difference*
* *Selection c (R - S) = Selection c (R) - Selection c (S) => Difference*

***Law:***

The laws below permits to push down Selection to one or two parameters. For Selection c, we just push down the Selection to the Relation includes all attributes. Assume that the Relation R includes all attributes mentioned in Condition c, then:

* *Selection c (R \* S) = Selection c (R) \* S => Product*
* *Selection c (R Join S) = Selection c (R) Join S => Union*
* *Selection c (R Join [d] S) = Selection c (R) Join [d] S => Union*
* *Selection c (R Intersection S) = Selection c (R) Intersection S => Intersection*

If Condition c only relates with the attribute in S, then:

* *Selection c (R \* S) = R \* Selection c (S) => Product*

***Law:***

If Relation R and S includes the attribute in Condition c, then we can use the law below:

* *Selection c (R Join S) = Selection c (R) Join Selection c (S)*

*Attention:*

If the Operator is Product or Join (D), we can not use this kind of Law since Relation R and S has no common attributes. But for Intersection, this kind of law is always useful, since the mode in Relation R and S are the same.

***Example:***

Consider the Relation R(a, b), S(b, c) and expression

*Selection (a = 1 OR a = 3) AND (b < c) (R Join S)*, here condition b < c can only be used on Relation S, but the condition a = 1 and a = 3 can only be used on Relation R. Therefore:

*Selection (a = 1 OR a = 3) [ Selection (b < c) (R Join S) ]*

After that, we can push OR condition downwards, and we get:

*Selection (a = 1 OR a = 3) [R Join Selection (b < c) (S)]*

At last, we get:

*Selection (a = 1 OR a = 3) (R) Join Selection (b < c) (S)*

***Normal Law:***

1. Selection Operation for random Empty Relation is empty.
2. For the condition C always equals to true, then Selection c (R) = R.
3. If Relation R is empty, then R Union S = S.

### Chapter 5.2.3 Push Down Selection

***Principle:***

The Push Down Selection in Expression, is the most powerful tool in the Query Optimizer - using the right expression to substitute with the left expression.

***Example:***

Assume that we have Relation:

*StarsIn(title, year, starName)*

*Movies(title, year, length, genre, studioName, producer)*

The Definition of View MoviesOf1996:

*CREATE VIEW MoviesOf1996 AS:*

*SELECT \**

*FROM Movies*

*WHERE year = 1996;*

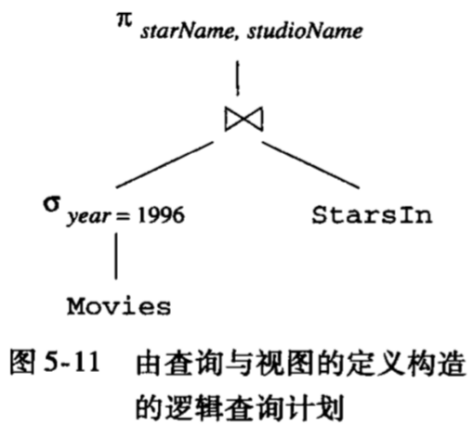
Using the SQL Query:

*SELECT starName, studioName*

*FROM MoviesOf1996 NATERAL JOIN StarsIn;*

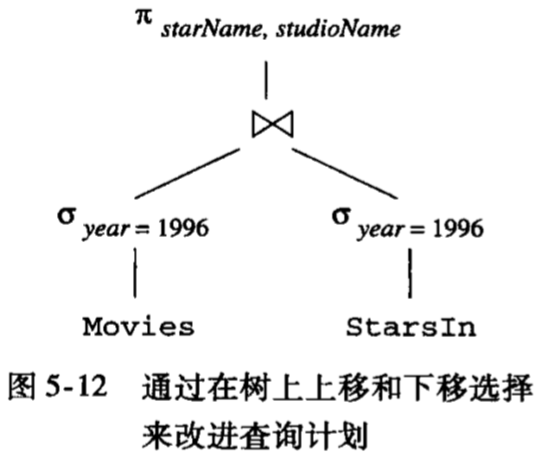
Here, we can define the View MoviesOf1996 by using the expression

*Selection (year = 1996) (Movies)*, so the SQL Query is using the Natural Join with StarsIn and project on the attributes starName and studioName. Therefore, we can conclude the process as the *Logic Query Plan* as below:



Since the Selection Operation has been used in the lowest level, therefore, we can no longer push Selection Operation down. But we can reverse the Expression here:

*Selection c (R Join S) = ( Selection c R ) Join ( Selection c S )*, we can push down *Selection (year = 1996)* down to two sub - nodes. The Logic Query Plan has been improved as the picture below:



This Logic Query Plan may be improved, since after using *Selection (year = 1996)* on the Relation StarsIn, then the volume of Relation StarsIn must be decreased.

### Chapter 5.2.4 Law Related with Projection

***Principle:***

Projection can be pushed down to multi other Operators just like Selection. Push down Projection is different with push down Selection, when push down Projection, then it’s common to leave Projection to the original place. While ‘Push Down Projection’ does related with introducing a new Projection under one existing Projection.

***Difference:(Selection & Projection)***

*Selection* can be useful when *decreasing the size of Relation* with a big factor while *Projection* can be useful only when *decreasing the length of tuple without changing the number of tuple*.

***Terminology:***

* Considering the terms in the Projection List E->x, E is one Attribute or Expression that contains the attribute or constant.
* All attributes in Projection List E are called Input Property and attribute x is called Output Property.
* If one term is Single Property, then it is not only Input Property but also Output Property.
* If the attributes in a Projection List do not contain renaming attributes and single attribute expression, then this Projection is called *Simple Projection*.

***Example 5.9:***

*Projection a, b, c (R)* is simple; *a, b, c* are Input Property and Output Property, but *Projection a + b -> x, c (R)* is not Simple Projection, and Ouput Property is x and c.

***Law:***

We can introduce Projection Operator in any place in the Expression Tree as long as the attributes that it eliminate will not be used any longer, and will not exist in the result of the whole Expression.

* *Projection L( R Join S ) = Projection L ( Projection M ( R ) Join Projection N ( S ) )* (Here M and N are Join Property, and they are all involved in the Input Property of L in Relation R and Relation S.)
* *Projection L ( R Join c S ) = Projection L ( Projection M ( R ) Join c Projection N( S ) )* (Here M and N are Join Property, just mentioned in the condition c, and they are all involved in the Input Property of L in Relation R and Relation S.)
* *Projection L ( R Product S ) = Projection L (Projection M ( R ) Product Projection N ( S ))* (Here M and N are all properties that involved in Relation R and Relation S.)

***Example 5.10:***

R(a, b, c) and S(c, d, e) are two Relations. Considering the expression *Projection a + e -> x, b -> y ( R Join S )*. Here Input Property in Projection are a, b, c, and c is the only join attribute. Then,

*Projection a + e -> x, b -> y ( Projection a, b, c ( R ) Join Projection b, c ( S ) )*

Here, *Projection a, b, c ( R )* is one Simple Projection, and it is all Projection Properties of Relation R. We try to eliminates this projection, and we get the third equal expression.

*Projection a + e -> x, b -> y ( R Join Projection c, e ( S ) )* that is to say, compared with the original expression, here we just remove the Attribute d from Relation S.

***Law:***

We can use Projection before Join, which is:

*Projection L ( R JoinB S ) = Projection L ( R ) JoinB Projection L ( S )*

*(Conversely, we can not push Projection down to the Set Union or Package, Set Intersection or Difference.)*

***Example 5.11:***

R(a, b) consists of one tuple { (1, 2) }, S(a, b) consists of one tuple { (1, 3) }. Although *Projection a ( R intersection S ) = Projection a (Empty) = Empty*.

But *Projection a ( R ) Intersection ( S ) = { ( 1 ) } Intersection { ( 1 ) } = { ( 1 ) }*

***Example 5.12:***

R(a, b, c) and S(c, d, e) are Relations, consider Join and Projection *Projection a + b -> x, d + e -> y (R Join S)*. Also we can rename a + b to x and move to the Relation R, and with the same operation with S, move d + e to the Relation S. Then,

*Projection x, y ( Projection a + b -> x, c ( R ) Join Projection d + e -> y, c ( S ) )*

Also, here we need to pay attention that when x or y is c, then we can not rename to c, since one Relation can not have two same name attributes of c, therefore we rename it. For example:

Turn *Projection a + b -> c, d + e -> y (R Join S)* to

*Projection z -> c, y ( Projection a + b -> z, c ( R ) Join Projection d + e -> y, c ( S ) )*

***Law:***

Here we can also push Projection down to Selection:

* *Projection L ( Selection C ( R ) ) = Projection L ( Selection C ( Projection M (R) ) )*  ( Here M is the Input Property of L or Attribute List that mentioned in Condition C. )

### Chapter 5.2.5 Join and Product Law

***Law:***

* *R Join c S = Selection c ( R Product S )*
* *R Join S = Projection L (Selection c ( R Product S ) )* (Condition C means that compare the same properties and check whether they have the same values. Here L contains one property in each equal pair and other property list. )

Normally, we always consider this rule from right to left. *Since we know that Join is much more faster than Product following a Selection.*

### Chapter 5.2.6 Eliminate Duplication Law

### Chapter 5.2.7 Grouping and Aggregation Law