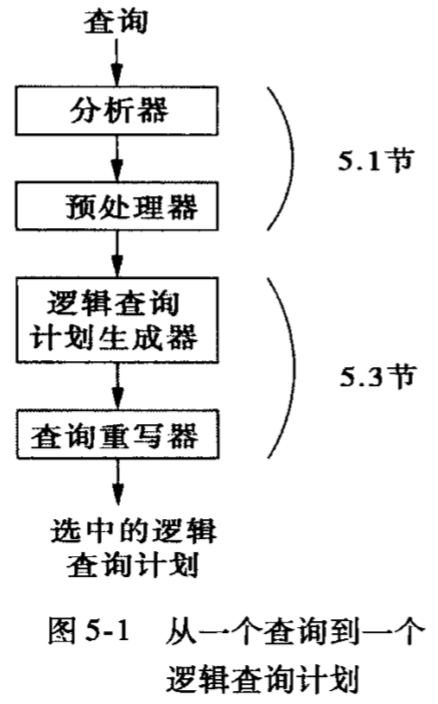
## Chapter 5.1 Grammatical Analysis and Pre - Processing

In this Chapter, we will discuss the Grammatical Analysis of SQL, and give the basic Grammatical Necessitates of this language. Of course, we will discuss Query that includes one virtual view and some other Pre - Processing steps.



### Chapter 5.1.1 Grammatical Analysis and Grammatical Analysis Tree

The task of Grammatical Analysis is receive the text wrote by SQL and converted to the Grammatical Analysis Tree, the nodes of this tree belongs to two of below:

1. Atom: They belongs to Lexical Type.

* Example: Key Word(Select), Relation Name, Attribute Name, Constant, Brackets, Operators, and other Module Elements.

1. Grammar Type: Query Sub - Composition that makes sense in one Query. Here we use Angle Brackets to represent the Grammar Type by enclosing those descriptive names.

* Example: <Query> is used to present Query with form of select - from - where. <Condition> is used to present any expressions that belongs to conditions, for example, those expressions following after WHERE in SQL.

If the current point is the type of Atom, then the Atom has no Sub - Atom. But if the Atom is the type of Grammar Type, then its Sub - Atom uses the Grammar Rule to describe it.

### Chapter 5.1.2 One Simple Collection Grammar in SQL

In this Chapter, we present some rules of one Simple Collection Grammar in SQL.

***Principle:***

*Query:*

Grammatical Type <Query> is used to present some rules of SQL Query Statements. Here is the one rule:

*<Query> ::= SELECT <SelList> From <FromList> WHERE <Condition>*

*‘::=’* signal is used to present the meaning of “could be present as”. The Grammar Type *<SelList>* and *<FromList>* can be used to present the list after SELECT and FROM. Grammar Type *<Condition>* is used to present the SQL condition.

*Attention:*

This kind of rule does not accept the multi - selection sub - statements, such as *GROUP BY*, *HAVING* or *ORDER BY* sub - statement, also it does not accept *DISTINCT*, *UNION*, *JOIN* or other binary operator.

*Selection List:*

*<SelList> ::= <Attribute>, <SelList>*

*<SelList> ::= <Attribute>*

These two rules show that selection can be used to presented by any Attribute List divided by comma: single attribute or one attribute, one comma and one random list with one or multi - attributes.

*Attention:*

In the whole SQL grammar rule, we can accept the rules that the expression and aggregation function and alias of attribute and expression.

*From List:*

*<FromList> ::= <Relation>, <FromList>*

*<FromList> ::= <Relation>*

The <FromList> consists of random relation list divided by comma. Of course, we neglect the possibility that the element in from list can be expression, such as join or sub - query.

*Condition:*

*<Condition> ::= <Condition> AND <Condition>*

*<Condition> ::= <Attribute> IN <Query>*

*<Condition> ::= <Attribute> = <Attribute>*

*<Condition> ::= <Attribute> LIKE <Pattern>*

Here, we neglect the rules below: OR, NOT, EXISTS, and other comparison operator, constant operation value, and many other structure, these all are needed in the integral SQL Grammar.

*Basic Grammar Type:*

Grammar Types <Attribute>, <Relation>, <Pattern> are the special types, because they are not defined by Grammar Rule, but they are defined by the Atomic Rule.

* <Relation> could be substituted by any meaningful string for the current Relation.
* <Pattern> could be substituted by any string enclosed by any quotation, this string is the legal SQL model matching.

***Example:***

StarsIn(movieTitle, movieYear, starName)

MovieStar(name, address, gender, birthdate)

*SQL:*

*SELECT movieTitle*

*FROM StarsIn*

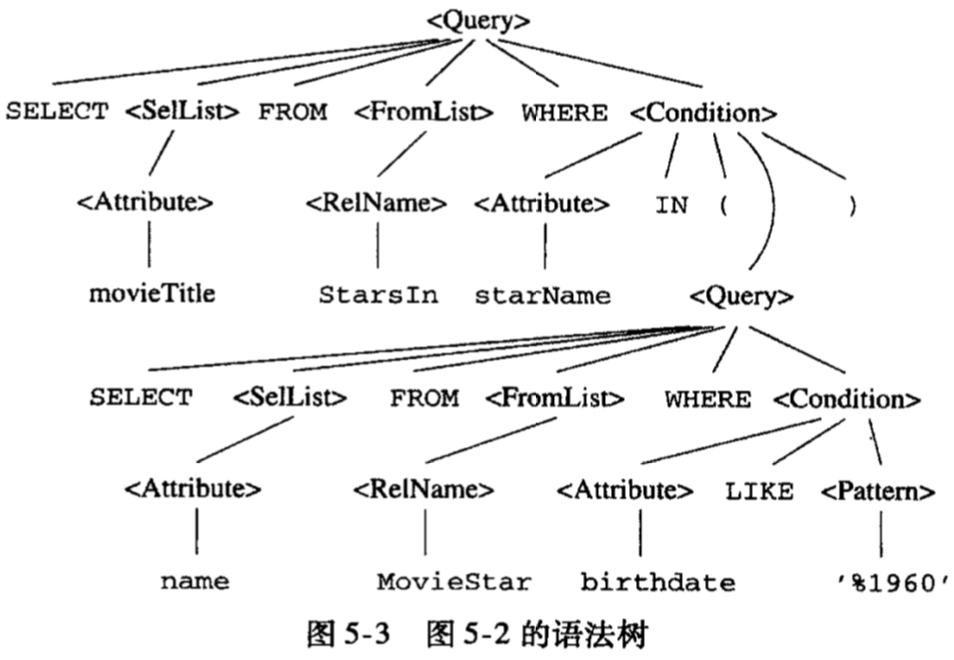
*WHERE starName IN (*

*SELECT name*

*FROM MovieStar*

*WHERE birthdate LIKE ‘%1960’*

*)*



The condition in ‘WHERE’ sub - statement is much more complex. It belongs to the mode of ‘attribute - IN - Query with enclosure’. This kind of sub - query has its own select list and from list, and LIK operator simple condition.

***Example:***

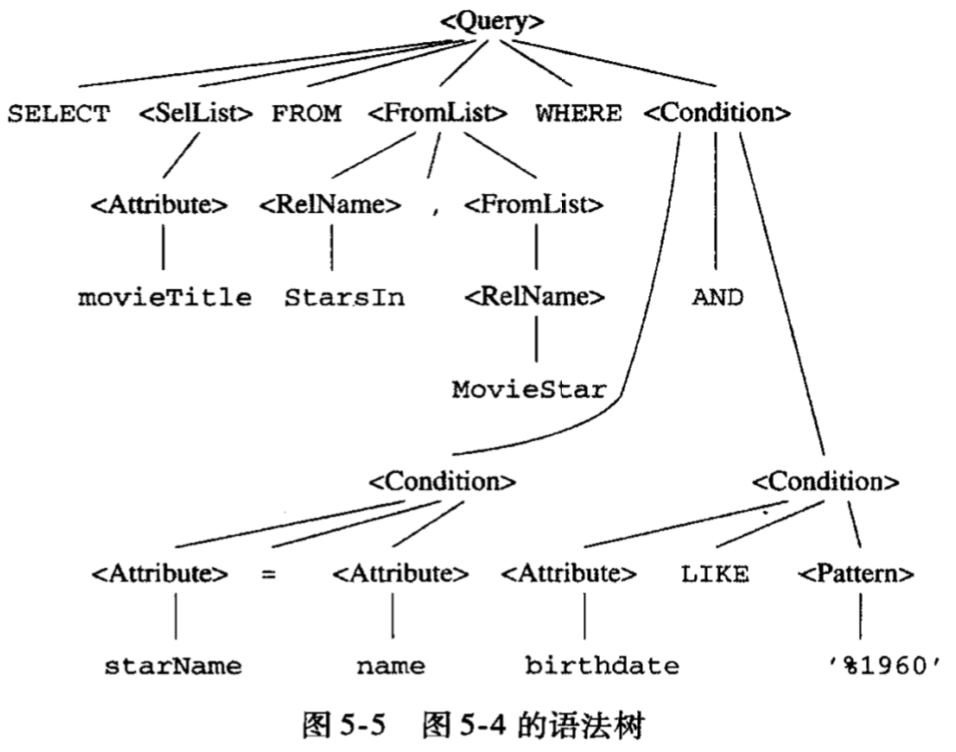
*SQL:*

*SELECT movieTitle*

*FROM StarIN, MovieStar*

*WHERE starName = name AND*

*birthday LIKE ‘%1960’*



### Chapter 5.1.3 Pre - Processing

***Guideline:***

Pre - Processor has many important functions. If the Relation in the Query Statement is actually a Virtual View, then we need to substitute the Grammar Tree in the from list when using this Relation. This Grammar Tree is defined by the View, but it is a Query Statement in natural.

Pre - Processor is also responsible for checking the semantic. Even this Query Statement makes sense in Grammar, but it actually may violate one or multi - semantic rules when using the name. For example:

1. *The Usage of Checking Relation:* The Relation exists in the Relation must be the Relation or View in the current Schema.
2. *The Usage of Checking and Parsing Attribute:* Each Attribute exist in the SELECT statement or WHERE statement must be the attribute in the current range of Relations.

* If the attribute movieTitle in the select list should be in the range of the only Relation StarsIn, then the Processor should check the usage of the attribute movieTitle.
* If the Query Statement did not attach the Quoted Relation to the attribute, then Processor will add the Quoted Relation to resolve each attribute.
* Processor also checks the ambiguity. If some attributes belong to two or more Relation Range, then it reports error.

1. *The Usage of Checking Type:* The types of all attributes should be adapted to its usage. The Processor needs to check the operator and to ensure that they works on the proper and compatibility value.

### Chapter 5.1.4 Query that Pre - Processing involves View

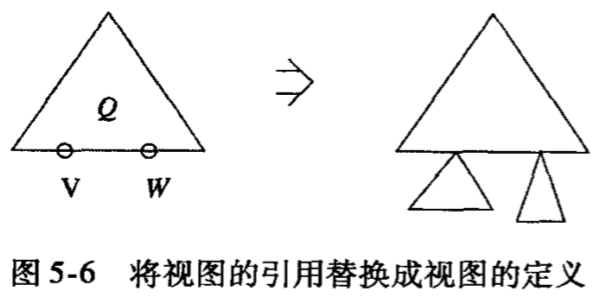
***Principle:***

If one of the Operation Object in the current Query is the Virtual View, then the Processor need to substitute this object with the Grammar Tree.

***Example:***

The Query Q is present by the Tree in Relation Algebra. Of course, this tree may has some leaf nodes as View. We assumes that there exist two views - View V and View W. In order to explain Q according to the basic table, then we need the definition of View V and W. These definitions are also related with Query, so they can be represent by Relation Algebra or Grammar Tree.

Here we use the copy of tree root to substitute the leaf node of the View in Q Query. Therefore we substitute the leaf V and W for the View definition. The resulting tree is the equal query on the original table and the equally original query about View.



***Example:***

Here consideration the definition of the view ParamountMovies:

*SQL:*

CREATE VIEW ParamountMovies AS

SELECT title, year

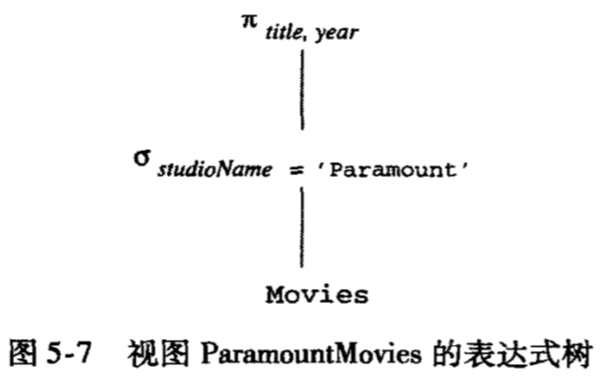
FROM Movies

WHERE studioName = ‘Paramount’;

*Analysis Table:*

|  |  |
| --- | --- |
| Selection (title, year) | Operation |
| Projection (studioName = ‘Paramount’) | Condition |
| Movies | Relation |

*Expression Tree:*



*SQL:*

SELECT title

FROM ParamountMovies

WHERE year = 1979

*Analysis Table:*

|  |  |
| --- | --- |
| Selection (title) | Operation |
| Projection (year = 1979) | Condition |
| ParamountMovies | Virtual View |

*Expression Tree:*

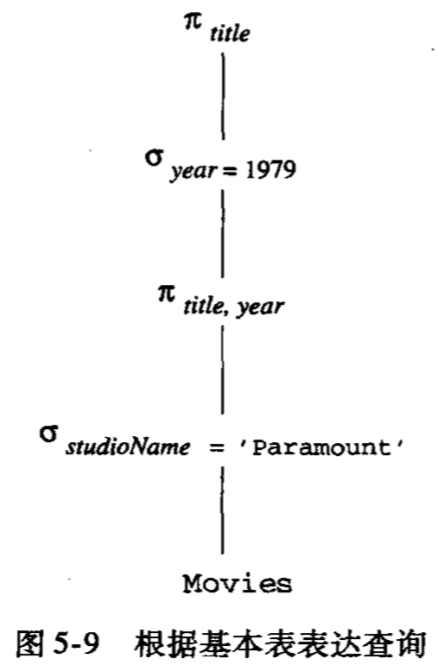


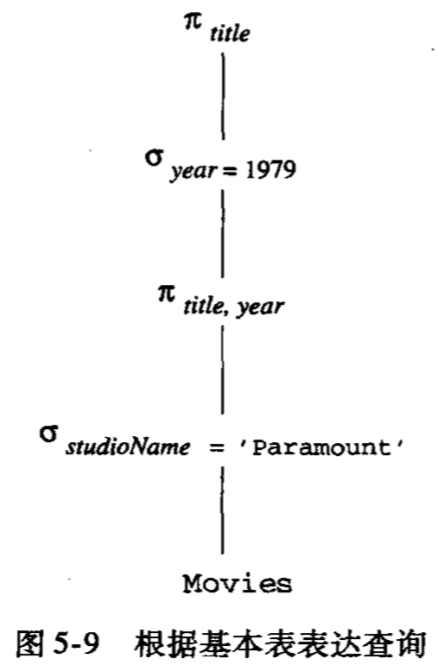
Below, we use the first Virtual View to substitute for the Virtual View in the second table.

*Analysis Table:*

|  |  |
| --- | --- |
| Selection (title) | Operation |
| Projection (year = 1979) | Condition |
| Selection (title, year) | Operation |
| Projection (studioName = ‘Paramount’) | Condition |
| Movies | Relation |

*Expression Tree:*





To be more specific, here we move Selection and Projection operation from the lower level to the upper level and merge them together.

*Analysis Table:*

|  |  |
| --- | --- |
| Selection (title) | Operation |
| Projection (year = 1979 AND studioName = ‘Paramount’) | Condition |
| Movies | Relation |

*Expression Tree:*

