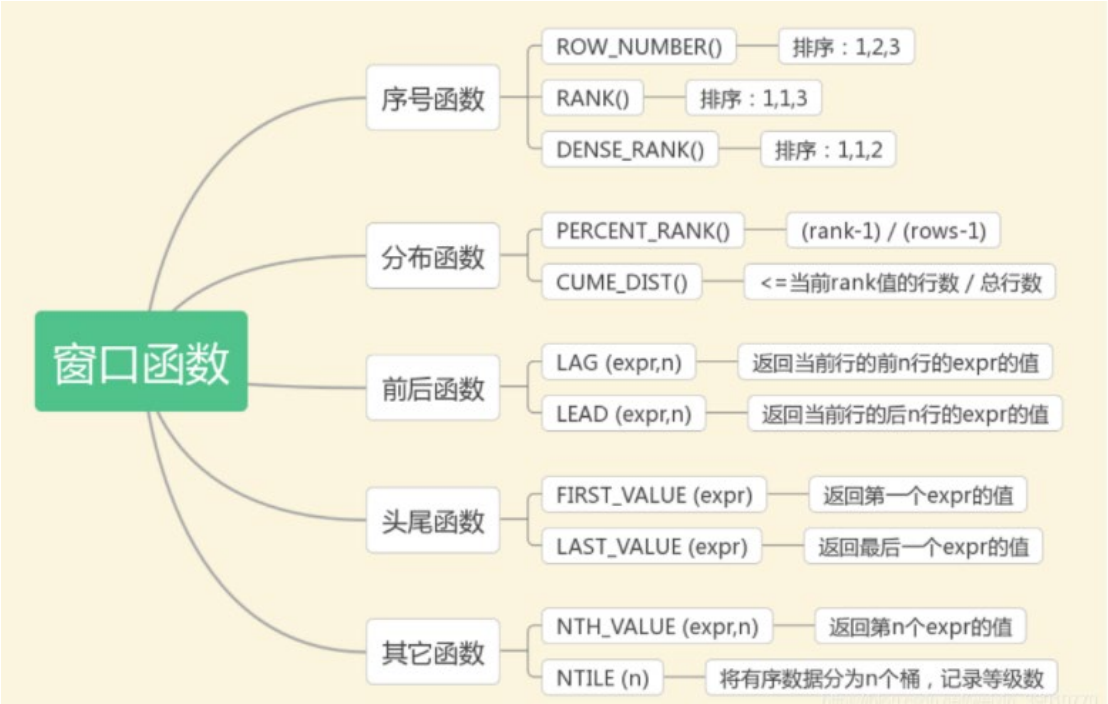
修改支持窗口函数

目录

常见窗口函数 ：



在 RAW_MAP 中添加的函数如下:

```
1.  //窗口函数
2.          SQL_FUNCTION_MAP.put("rank", "");//得到数据项在分组中的排名，排名相等的时候会留下空位
3.          SQL_FUNCTION_MAP.put("dense_rank", ""); //得到数据项在分组中的排名，排名相等的时候不会留下空位
4.          SQL_FUNCTION_MAP.put("row_number", "");//按照分组中的顺序生成序列，不存在重复的序列
5.          SQL_FUNCTION_MAP.put("ntile", "");//用于将分组数据按照顺序切分成 N 片，返回当前切片值，不支持
    ROWS_BETWEE
6.          SQL_FUNCTION_MAP.put("first_value", "");//取分组排序后，截止到当前行，分组内第一个值
7.          SQL_FUNCTION_MAP.put("last_value", "");//取分组排序后，截止到当前行，分组内的最后一个值
8.          SQL_FUNCTION_MAP.put("lag", "");//统计窗口内往上第 n 行值。第一个参数为列名，第二个参数为往上第 n 行（可
    选，默认为 1），第三个参数为默认值（当往上第 n 行为 NULL 时候，取默认值，如不指定，则为 NULL）
```

```
9.          SQL_FUNCTION_MAP.put("lead", "");//统计窗口内往下第 n 行值。第一个参数为列名，第二个参数为往下第 n 行（可选，默认为 1），第三个参数为默认值（当往下第 n 行为 NULL 时候，取默认值，如不指定，则为 NULL）
10.         SQL_FUNCTION_MAP.put("cume_dist", "");//)返回（小于等于当前行值的行数）/（当前分组内的总行数）
11.         SQL_FUNCTION_MAP.put("percent_rank", "");//返回（组内当前行的 rank 值-1）/（分组内做总行数-1）
```

描述 SQL 语句内 OVER 子句划定的内容，这个内容就是窗口函数的作用域。而在 OVER 子句中，定义了窗口所覆盖的与当前行相关的数据行集、行的排序及其他的相关元素。

窗口函数语法

window_function_name(window_name/expression)
OVER (
[partition_defintion]
[order_definition]
[frame_definition])

窗口数据集由"[partition_defintion]"，"[order_definition]"，"[frame_definition]"确定。

可以看出窗口函数的特点　是 语句 + OVER + 语句　，而且语句中的参数是由括号分隔的 。 需要设计这样的语法解析 。那么为了支持解析这样的函数，在原来代码的基础之上进行了修改添加　。添加的核心代码部分如下，有注释 。

```
1.  //是窗口函数   fun(arg0,agr1) OVER (agr0 agr1 ...)
2.                  int overindex = expression.indexOf("OVER"); // OVER 的位置
3.                  String s1 = expression.substring(0, overindex); // OVER 前半部分
4.                  String s2 = expression.substring(overindex); // OVER 后半部分
5.
6.                  int index1 = s1.indexOf("("); //  函数 "(" 的起始位置
7.                  String fun = s1.substring(0, index1); // 函数名称
8.                  int end = s2.lastIndexOf(")"); // 后半部分 “)” 的位置
9.
10.                 if (index1 >= end) {
11.                     throw new IllegalArgumentException("字符 " + expression + " 不合法！"
12.                             + "@column:value 中 value 里的 SQL 函数必须为 function(arg0,arg1,...) 这种格式！");
13.                 }
14.                 if (fun.isEmpty() == false) {
15.                     if (FunctionsAndRaws.SQL_FUNCTION_MAP == null || FunctionsAndRaws.SQL_FUNCTION_MAP.isEmpty()) {
16.                         if (StringUtil.isName(fun) == false) {
17.                             throw new IllegalArgumentException("字符 " + fun + " 不合法！"
```

```
                                    + "预编译模式
下 @column:\"column0,column1:alias;function0(arg0,arg1,...);function1(...):alias...\""
                                    + " 中 function 必须符合小写英文单词的 SQL 函数名格式! ");
                        }
                    } else if (FunctionsAndRaws.SQL_FUNCTION_MAP.containsKey(fun) == false) {
                        throw new IllegalArgumentException("字符 " + fun + " 不合法! "
                                + "预编译模式
下 @column:\"column0,column1:alias;function0(arg0,arg1,...);function1(...):alias...\""
                                + " 中 function 必须符合小写英文单词的 SQL 函数名格式! 且必须是后端允许调用
的 SQL 函数!");
                    }
                }

                // 获取前半部分函数的参数解析   fun(arg0,agr1)
                String agrsString1[] = parseArgsSplitWithComma(s1.substring(index1 + 1, s1.lastIndexOf(")")
), false);


                int index2 = s2.indexOf("("); // 后半部分 “(”的起始位置
                String argString2 = s2.substring(index2 + 1, end); // 后半部分的参数
                // 别名
                String alias = s2.lastIndexOf(":") < 0 ? null : s2.substring(s2.lastIndexOf(":") + 1);
                // 获取后半部分的参数解析 (agr0 agr1 ...)
                String argsString2[] = parseArgsSplitWithComma(argString2,false);
                expression = fun + "(" + StringUtil.getString(agrsString1) + ")" + " OVER " + "(" + StringU
til.getString(argsString2) + ")" + (StringUtil.isEmpty(alias, true) ? "" : " AS " + quote + alias + quote)
;
```