



下载APP



特别放送（三） | MySQL 8 都有哪些新特征？

2021-05-13 朱晓峰

MySQL 必知必会

[进入课程 >](#)**讲述：朱晓峰**

时长 09:26 大小 8.65M



你好，我是朱晓峰。今天，我来和你聊一聊 MySQL 8 的新特征。

作为应用最广泛的三大关系型数据库之一，MySQL 的背后有一个强大的开发团队，使 MySQL 能够持续迭代和创新，满足不断变化的用户需求。在 MySQL 8 中，就有很多新特征。

今天，我就给你介绍两个重要的新特征：窗口函数和公用表表达式（Common Table Expressions，简称 CTE）。它们可以帮助我们使用相对简单的查询语句，实现更加强大的查询功能。



什么是窗口函数？

窗口函数的作用类似于在查询中对数据进行分组，不同的是，分组操作会把分组的结果聚合成一条记录，而窗口函数是将结果置于每一条数据记录中。一会儿我会借助一个例子来对比下，在此之前，你要先掌握窗口函数的语法结构。

窗口函数的语法结构是：

```
1 函数 OVER ([PARTITION BY 字段])
```

[复制代码](#)

或者是：

```
1 函数 OVER 窗口名 ... WINDOW 窗口名 AS ([PARTITION BY 字段名])
```

[复制代码](#)

现在，我借助一个小例子来解释一下窗口函数的用法。

假设我现在有这样一个数据表，它显示了某购物网站在每个城市每个区的销售额：


```
1 mysql> SELECT * FROM demo.test1;
2 +-----+-----+-----+-----+
3 | id | city | county | salesvalue |
4 +-----+-----+-----+-----+
5 | 1 | 北京 | 海淀 | 10.00 |
6 | 2 | 北京 | 朝阳 | 20.00 |
7 | 3 | 上海 | 黄埔 | 30.00 |
8 | 4 | 上海 | 长宁 | 10.00 |
9 +-----+-----+-----+-----+
10 4 rows in set (0.00 sec)
```

[复制代码](#)

现在我想计算一下，这个网站在每个城市的销售总额、在全国的销售总额、每个区的销售额占所在城市销售额中的比率，以及占总销售额中的比率。


如果用分组和聚合函数，就需要分好几步来计算。

第一步，计算总销售金额，并存入临时表 demo.a：

 复制代码

```
1 mysql> CREATE TEMPORARY TABLE demo.a      -- 创建临时表
2     -> SELECT SUM(salesvalue) AS salesvalue -- 计算总计金额
3     -> FROM demo.test1;
4 Query OK, 1 row affected (0.02 sec)
5 Records: 1  Duplicates: 0  Warnings: 0
```


我们查看一下临时表 demo.a 的内容，来验证一下计算结果：

 复制代码

```
1 mysql> SELECT * FROM demo.a;
2 +-----+
3 | salesvalue |
4 +-----+
5 |      70.00 |
6 +-----+
7 1 row in set (0.00 sec)
```

结果显示，总计金额已经存入临时表 demo.a 中了。

第二步，计算每个城市的销售总额并存入临时表 demo.b：

 复制代码

```
1 mysql> CREATE TEMPORARY TABLE demo.b      -- 创建临时表
2     -> SELECT city,SUM(salesvalue) AS salesvalue -- 计算城市销售合计
3     -> FROM demo.test1
4     -> GROUP BY city;
5 Query OK, 2 rows affected (0.01 sec)
6 Records: 2  Duplicates: 0  Warnings: 0
```

我们查看一下临时表 demo.b 的内容，验证一下计算的结果：

 复制代码

```
1 mysql> SELECT * FROM demo.b;
2 +-----+-----+
3 | city | salesvalue |
4 +-----+-----+
5 | 北京 |      30.00 |
6 | 上海 |      40.00 |
7 +-----+-----+
```

```
8 2 rows in set (0.00 sec)
```

结果显示，每个城市的销售总计金额已经计算成功了。

第三步，计算各区销售占所在城市的总计金额的比例，和占全部销售总计金额的比例。我们可以通过下面的连接查询获得需要的结果：

[复制代码](#)

```
1 mysql> SELECT a.city AS 城市,a.county AS 区,a.salesvalue AS 区销售额,
2     -> b.salesvalue AS 市销售额,a.salesvalue/b.salesvalue AS 市比率,
3     -> c.salesvalue AS 总销售额,a.salesvalue/c.salesvalue AS 总比率
4     -> FROM demo.test1 AS a
5     -> JOIN demo.b AS b ON (a.city=b.city) -- 连接市统计结果临时表
6     -> JOIN demo.a AS c          -- 连接总计金额临时表
7     -> ORDER BY a.city,a.county;
8 +-----+-----+-----+-----+-----+-----+-----+
9 | 城市 | 区   | 区销售额 | 市销售额 | 市比率   | 总销售额 | 总比率   |
10 +-----+-----+-----+-----+-----+-----+-----+
11 | 上海 | 长宁 | 10.00 | 40.00 | 0.250000 | 70.00 | 0.142857 |
12 | 上海 | 黄埔 | 30.00 | 40.00 | 0.750000 | 70.00 | 0.428571 |
13 | 北京 | 朝阳 | 20.00 | 30.00 | 0.666667 | 70.00 | 0.285714 |
14 | 北京 | 海淀 | 10.00 | 30.00 | 0.333333 | 70.00 | 0.142857 |
15 +-----+-----+-----+-----+-----+-----+-----+
16 4 rows in set (0.01 sec)
```

结果显示：市销售金额、市销售占比、总销售金额、总销售占比都计算出来了。

同样的查询，如果用窗口函数，就简单多了。我们可以用下面的代码来实现：

[复制代码](#)

```
1 mysql> SELECT city AS 城市,county AS 区,salesvalue AS 区销售额,
2     -> SUM(salesvalue) OVER(PARTITION BY city) AS 市销售额, -- 计算市销售额
3     -> salesvalue/SUM(salesvalue) OVER(PARTITION BY city) AS 市比率,
4     -> SUM(salesvalue) OVER() AS 总销售额, -- 计算总销售额
5     -> salesvalue/SUM(salesvalue) OVER() AS 总比率
6     -> FROM demo.test1
7     -> ORDER BY city,county;
8 +-----+-----+-----+-----+-----+-----+-----+
9 | 城市 | 区   | 区销售额 | 市销售额 | 市比率   | 总销售额 | 总比率   |
10 +-----+-----+-----+-----+-----+-----+-----+
11 | 上海 | 长宁 | 10.00 | 40.00 | 0.250000 | 70.00 | 0.142857 |
12 | 上海 | 黄埔 | 30.00 | 40.00 | 0.750000 | 70.00 | 0.428571 |
13 | 北京 | 朝阳 | 20.00 | 30.00 | 0.666667 | 70.00 | 0.285714 |
```

```

14 | 北京 | 海淀 | 10.00 | 30.00 | 0.333333 | 70.00 | 0.142857 |
15 +-----+-----+-----+-----+-----+-----+-----+
16 4 rows in set (0.00 sec)

```

结果显示，我们得到了与上面那种查询同样的结果。

你看，使用窗口函数，我们只用了一步就完成了查询，过程简单多了。而且，由于没有用到临时表，执行的效率也更高了。很显然，**在这种需要用到分组统计的结果对每一条记录进行计算的场景下，使用窗口函数更好。**

除了可以进行分组统计，窗口函数还有一些自己独有的函数，可以对分组内的数据进行处理，比较常用的就是排序函数 RANK()、DENSE_RANK() 和 ROW_NUMBER()。


为了帮助你理解这几个函数的作用，我举个小例子。

假设我们有这样一张学生成绩表：

```

1 mysql> SELECT * FROM demo.test2;
2 +-----+-----+-----+
3 | id | student | points |
4 +-----+-----+-----+
5 | 1 | 张三 | 89 |
6 | 2 | 李四 | 77 |
7 | 3 | 王五 | 88 |
8 | 4 | 赵六 | 90 |
9 | 5 | 孙七 | 90 |
10 | 6 | 周八 | 88 |
11 +-----+-----+-----+
12 6 rows in set (0.00 sec)

```

 复制代码

如果我们需要对表中的数据进行排序，就可以使用排序函数，代码如下所示：

```

1 mysql> SELECT student,points,
2     -> RANK() OVER w AS 排序1,
3     -> DENSE_RANK() OVER w AS 排序2,
4     -> ROW_NUMBER() OVER w AS 排序3
5     -> FROM demo.test2
6     -> WINDOW w AS (ORDER BY points DESC);

```

 复制代码

```

7  +-----+-----+-----+-----+
8  | student | points | 排序1 | 排序2 | 排序3 |
9  +-----+-----+-----+-----+
10 | 赵六    |      90 |      1 |      1 |      1 |
11 | 孙七    |      90 |      1 |      1 |      2 |
12 | 张三    |      89 |      3 |      2 |      3 |
13 | 王五    |      88 |      4 |      3 |      4 |
14 | 周八    |      88 |      4 |      3 |      5 |
15 | 李四    |      77 |      6 |      4 |      6 |
16 +-----+-----+-----+-----+
17 6 rows in set (0.01 sec)

```

结果显示：

RANK() 函数把并列计算在内，并且并列影响排位；

DENSE_RANK() 函数也计算并列，但是并列不影响排位；

ROW_NUMBER() 函数不计算并列，只是简单排序。

因此，我们就可以根据这些函数的特点，计算分组中的排位信息。如果不计算并列，就用 ROW_NUMBER() 函数；计算并列但不占用位次，就用 DENSE_RANK() 函数；计算并列且占用位次，就用 RANK() 函数。

接下来，我们再来学习 MySQL 8 的另一个重要新特征：公用表表达式。

什么是公用表表达式？

公用表表达式是一个命名的临时结果集。它存在于单个查询语句中，主要作用就是可以代替子查询，并且可以被后面的查询多次引用。

依据语法结构和执行方式的不同，公用表表达式分为普通公用表表达式和递归公用表表达式 2 种。

什么是普通公用表表达式？

普通公用表表达式的语法结构是：

```

1 WITH
2 CTE名称 AS (子查询)


```

 复制代码

3 `SELECT|DELETE|UPDATE` 语句；

普通公用表表达式类似于子查询，不过，跟子查询不同的是，它可以被多次引用，而且可以被其他的普通公用表表达式所引用。

举个小例子，假设我们有一个商品信息表（demo.goodsmaster），它保存的是商品信息，还有一个每日销售统计表（demo.dailystatistics），保存的是每日的销售统计信息。现在超市经营者想要查出都卖了什么商品，我们就可以先通过子查询查出所有销售过的商品的编号，然后查出这些商品的商品信息，代码如下所示：

 复制代码

```
1 mysql> SELECT * FROM demo.goodsmaster
2 -> WHERE itemnumber IN
3 -> (SELECT DISTINCT itemnumber      -- 子查询，查出所有销售过的商品的编号
4 -> FROM demo.dailystatistics);
5 +-----+-----+-----+-----+-----+-----+
6 | itemnumber | barcode | goodsname | specification | unit | salesprice |
7 +-----+-----+-----+-----+-----+-----+
8 | 1 | 0001 | 书 | 16开 | 本 | 89.00 |
9 | 2 | 0002 | 笔 | 黑色 | 支 | 3.00 |
10 | 3 | 0003 | 胶水 | 无机 | 瓶 | 15.00 |
11 +-----+-----+-----+-----+-----+-----+
12 3 rows in set (0.01 sec)
```

这个查询也可以用普通公用表表达式的方式完成：

 复制代码

```
1 mysql> WITH
2 -> cte AS (SELECT DISTINCT itemnumber FROM demo.dailystatistics)
3 -> SELECT *
4 -> FROM demo.goodsmaster a JOIN cte
5 -> ON (a.itemnumber = cte.itemnumber);
6 +-----+-----+-----+-----+-----+-----+-----+
7 | itemnumber | barcode | goodsname | specification | unit | salesprice | itemn
8 +-----+-----+-----+-----+-----+-----+-----+
9 | 1 | 0001 | 书 | 16开 | 本 | 89.00 | 1 |
10 | 2 | 0002 | 笔 | 黑色 | 支 | 3.00 | 2 |
11 | 3 | 0003 | 胶水 | 无机 | 瓶 | 15.00 | 3 |
12 +-----+-----+-----+-----+-----+-----+-----+
13 3 rows in set (0.00 sec)
```



可以看到，普通公用表表达式代替了第一种查询方式中的子查询，并且得到了同样的结果。

这个例子说明，公用表表达式可以起到子查询的作用。以后如果遇到需要使用子查询的场景，你可以在查询之前，先定义公用表表达式，然后在查询中用它来代替子查询。而且，跟子查询相比，公用表表达式有一个优点，就是定义过公用表表达式之后的查询，可以像一个表一样多次引用公用表表达式，而子查询则不能。

好了，我们再来学习下递归公用表表达式。

什么是递归公用表表达式？

递归公用表表达式也是一种公用表表达式，只不过，除了普通公用表表达式的特点以外，它还有自己的特点，就是**可以调用自己**。它的语法结构是：

 复制代码

```
1 WITH RECURSIVE
2 CTE名称 AS (子查询)
3 SELECT|DELETE|UPDATE 语句;
```

递归公用表表达式由 2 部分组成，分别是种子查询和递归查询，中间通过关键字 UNION [ALL] 进行连接。这里的**种子查询，意思就是获得递归的初始值**。这个查询只会运行一次，以创建初始数据集，之后递归查询会一直执行，直到没有任何新的查询数据产生，递归返回。

同样，为了帮助你理解递归公用表表达式的工作原理，我来举个小例子。

假设我们有这样一张人员信息表（demo.teach），里面包含人员编号、名称和老师编号。

id (人员编号)	fname (名称)	teacherid (老师编号)
101	张三	NULL
102	李四	101
103	王五	102
104	赵六	101
105	孙七	104
106	周八	105

如果甲是乙的老师，那么，我们可以把乙叫做甲的徒子，如果同时乙又是丙的老师，那么丙就是乙的徒子，是甲的徒孙。

下面我们尝试用查询语句列出所有具有徒孙身份的人员信息。

如果用我们之前学过的知识来解决，会比较复杂，至少要进行 4 次查询才能搞定：

第一步，先找出初代老师，就是不以任何人为老师的人，把结果存入临时表；

第二步，找出所有以初代老师为师的人，得到一个徒子集，把结果存入临时表；

第三步，找出所有以徒子为师的人，得到一个徒孙集，把结果存入临时表。

第四步，找出所有以徒孙为师的人，得到一个结果集。

如果第四步的结果集为空，则计算结束，第三步的结果集就是我们需要的徒孙集了，否则就必须继续进行第四步，一直到结果集为空为止。比如上面的这个数据表，就需要到第五步，才能得到空结果集。而且，最后还要进行第六步：把第三步和第四步的结果集合并，这样才能最终获得我们需要的结果集。

如果用递归公用表表达式，就非常简单了。我介绍下具体的思路。

用递归公用表表达式中的种子查询，找出初代老师。字段 `n` 表示代次，初始值为 1，表示是第一代老师。

用递归公用表表达式中的递归查询，查出以这个递归公用表表达式中的人为老师的人，并且代次的值加 1。直到没有人以这个递归公用表表达式中的人为老师了，递归返回。

在最后的查询中，选出所有代次大于等于 3 的人，他们肯定是第三代及以上代次的学生了，也就是徒孙了。这样就得到了我们需要的结果集。

这里看似也是 3 步，实际上是一个查询的 3 个部分，只需要执行一次就可以了。而且也不需要用临时表保存中间结果，比刚刚的方法简单多了。

下面是具体的代码：

[复制代码](#)

```
1 mysql> WITH RECURSIVE
2 -> cte AS (
3 -> SELECT id,fname,teacherid,1 AS n FROM demo.teach WHERE id = 101 -- 种子查询,
4 -> UNION ALL
5 -> SELECT a.id,a.fname,a.teacherid,n+1 FROM demo.teach AS a JOIN cte
6 -> ON (a.teacherid = cte.id) -- 递归查询, 找出以递归公用表表达式的人为老师的人
7 -> )
8 -> SELECT id,fname FROM cte WHERE n>=3; -- 从结果集中筛选代次大于等于3的, 得到所有徒
9 +-----+-----+
10 | id | fname |
11 +-----+-----+
12 | 103 | 王五 |
13 | 105 | 孙七 |
14 | 106 | 周八 |
15 +-----+-----+
16 3 rows in set (0.00 sec)
```

结果显示，王五、孙七和周八是徒孙。结果显然是正确的。

总之，递归公用表表达式对于查询一个有共同的根节点的树形结构数据，非常有用。它可以不受层级的限制，轻松查出所有节点的数据。如果用其他的查询方式，就比较复杂了。

总结

这节课，我们学习了 MySQL 8 的 2 个重要新功能：窗口函数和公用表表达式。

窗口函数的特点是可以分组，而且可以在分组内排序。另外，窗口函数不会因为分组而减少原表中的行数，这对我们在原表数据的基础上进行统计和排序非常有用。

公用表表达式的作用是可以替代子查询，而且可以被多次引用。递归公用表表达式对查询有一个共同根节点的树形结构数据非常高效，可以轻松搞定其他查询方式难以处理的查询。

当然，除了今天学习的窗口函数和公用表表达式，MySQL 8 还有许多其他的新特征，比如，完善了对空间位置信息的处理；支持对表的 DDL 操作（创建、修改和删除表）的原子性，使得 `CREATE TABLE ... SELECT` 语句能够成为一个原子操作，提高了数据安全性，等等。

如果你想要从旧版本切换到 MySQL 8，课下可以点击这个 [🔗 链接](#) 进一步了解一下。

思考题

假设我有一个会员销售统计表（demo.memtrans），其中包括会员名称、商品名称和销售金额，具体数据如下：

membername (会员名称)	goodsname (商品名称)	actualvalue (销售金额)
张三	书	890
李四	笔	30
王五	书	89

请使用窗口函数查询会员名称、商品名称、销售金额、总计金额和销售占比。

欢迎在留言区写下你的思考和答案，我们一起交流讨论。如果你觉得今天的内容对你有所帮助，也欢迎你把它分享给你的朋友或同事，我们下节课见。

提建议

更多学习推荐

 极客大学

175 道 Go 工程师 大厂常考面试题

限量免费领取 

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 特别放送（二）| 经典面试题讲解第二弹

下一篇 特别放送（四）| 位置信息：如何进行空间定位？

精选留言 (1)

 写留言freshswq 

2021-05-13

1.建表语句：

DROP TABLE

IF

EXISTS memtrans;

...

展开 

作者回复: 请参考思考题答案



1