第 5 章

CHAPTER 5

SQL 数字的查询与处理

5.1 MySQL 8.0 的数字

每种语言与数据库都含有针对处理数字的数据类型。

在 Java 中含有 short、int、long、float、double 等数据类型，用于处理与数字有关的逻辑与业务。

在 MySQL 中含有 int、bigint、float、double 等数据类型，用于处理与数字有关的逻辑与业务。

5.1.1 MySQL 8.0 中的数字类型

MySQL 8.0 中支持的数字类型如表 5-1 所示。

表 5-1 MySQL 8.0 中支持的数字类型

数字类型	长度	释义
tinyint[(m)]	有符号整数：－128～127 无符号整数：0～255	需要 1 字节存储空间。m 是可选的显示宽度，不影响值的范围
smallint[(m)]	有符号整数：－32 768～32 767 无符号整数：0～65 535	需要 2 字节存储空间。m 是可选的显示宽度，不影响值的范围
mediumint[(m)]	有符号整数：－8 388 608～8 388 607 无符号整数：0～16 777 215	需要 3 字节存储空间。m 是可选的显示宽度，不影响值的范围
int[(m)]	有符号整数：－2 147 483 648～2 147 483 647 无符号整数：0～4 294 967 295	需要 4 字节存储空间。m 是可选的显示宽度，不影响值的范围
bigint[(m)]	有符号整数：－9 223 372 036 854 775 808～9 223 372 036 854 775 807 无符号整数：0～18 446 744 073 709 551 615	需要 8 字节存储空间。m 是可选的显示宽度，不影响值的范围，但是要注意对较大的值进行代数运算可能会出现错误

续表

数字类型	长　度	释　义
float[(m,d)]	精确到小数点后 7 位左右	单精度的浮点值，m 为最大位数，并可包含小数点后的 d 位小数。注意许多十进制的数字遇浮点类型会丧失精度，若需要确切的值，则应更改为 decimal 类型
double[(m,d)]	精确到小数点后 15 位左右	m 为最大位数(最大位数包含整数＋小数)，并可包含小数点后的 d 位小数。注意许多十进制的数字遇浮点类型会丧失精度，若需要确切的值，则应更改为 decimal 类型
double precision[(m,d)]	double 的别名	与 double 一致
real[(m,d)]	默认为 double 别名	若设置了 real_as_float，则为 float 别名
decimal[(m[,d])]	m(精确度)：高达 65 位 d(小数)：0～30 位	非浮点的确切十进制类型，其中 m 为最大位数，并可包含小数点后的 d 位小数。如有必要，则类型将四舍五入
bit[(m)]	m 默认值为 1，范围为 1～64	位字段类型，其中 m 用于指定位数。若输入值位数小于 m，则对齐到最后一位。 若要单独命名每位，则可参考 set
serial	范围：0～18 446 744 073 709 551 615	serial 是 bigint unsigned not null auto_increment unique 的别名。 unsigned 指无序号 not null 指不为空 auto_increment 指自增 unique 指唯一约束
bool		bool 是 tinyint(1)的别名
boolean		boolean 是 tinyint(1)的别名
dec[(m[,d])]		dec 是 decimal 的别名，除了 dec 之外 decimal 的别名还有 fixed 和 numeric

5.1.2　tinyint 类型、bool 类型、boolean 类型

tinyint 类型是 MySQL 中最小的数字类型，仅需 1 字节进行存储，可存储有符号整数－128～127，或无符号整数 0～255。

创建表时将字段设置为 tinyint 类型的 SQL 语句如下：

```
create table test_tinyint(t1 tinyint(1));
```

向 test_tinyint 表中添加测试数据，SQL 语句如下：

```
insert into test_tinyint values(10);
insert into test_tinyint values(127);
insert into test_tinyint values( - 128);
```

查询语句如下：

```
select * from test_tinyint;
```

运行后，结果集如图 5-1 所示。

在 MySQL 中，bool 与 boolean 本质上就是 tinyint 字段，可创建字段为 bool 类型和 boolean 类型的 SQL 语句如下：

```
create table test_bool(t1 bool,t2 boolean);
```

向 test_bool 表中添加测试数据，SQL 语句如下：

```
insert into test_bool values(0,0);
insert into test_bool values(1,1);
insert into test_bool values(true,true);
insert into test_bool values(false,false);
```

查询语句如下：

```
select * from test_bool;
```

运行后，结果集如图 5-2 所示。

```
+------+
| t1   |
+------+
|   10 |
|  127 |
| -128 |
+------+
3 rows in set (0.00 sec)
```

图 5-1 查询 tinyint 字段结果

```
+------+------+
| t1   | t2   |
+------+------+
|    0 |    0 |
|    1 |    1 |
|    1 |    1 |
|    0 |    0 |
+------+------+
4 rows in set (0.00 sec)
```

图 5-2 查看 boolean 字段结果

5.1.3 无符号整数类型

在 MySQL 8.0 中创建数字类型在默认情况下是“含有符号的数字类型”，例如创建含有 int 数字类型的测试表，SQL 语句如下：

```
create table test_int1 (t1 int);
```

增加测试表之后，可插入列值 2 147 483 648，SQL 示例语句与报错如下：

```
insert into test_int1 value(2147483648);
-- 报错 Out of range value for column 't1' at row 1
```

报错中说明已经超过 t1 列值的承受能力了。MySQL 8.0 中提供了有符号整数和无符号整数两种写法，默认创建的数字类型皆为有符号整数，创建无符号整数时需要增加关键字 unsigned，创建无符号整数的 SQL 语句如下：

```
-- 创建无符号整数类型的表 test_int2
create table test_int2 (t1 int unsigned);
-- 添加数据
insert into test_int2 value(2147483648);
```

5.1.4　数字类型的精度

创建 double 类型，将精度设置为 2 的 SQL 语句如下：

```
-- 创建 float 和 decimal 类型表
create table test_float(t1 float(10,2), t2 decimal(10,2));
```

其中 float 的最大位数为 10，代表着整数最多为 8 位，小数为 2 位。添加数据的示例 SQL 语句如下：

```
insert into test_float values(12345678.21, 12345678.12);
```

若输入以下 SQL 语句，MySQL 则会报错。

```
insert into test_float values(1234567890, 1234567890);
-- 报错为 Out of range value for column 't1' at row 1
```

在没有输入任何小数时会默认将小数填充为 0。添加没有任何小数的 SQL 语句如下：

```
insert into test_float values(12345678, 12345678);
```

通过 SQL 语句查询 test_float 的结果集如图 5-3 所示。

```
+-------------+-------------+
| t1          | t2          |
+-------------+-------------+
| 12345678.00 | 12345678.00 |
| 12345678.00 | 12345678.12 |
+-------------+-------------+
2 rows in set (0.00 sec)
```

图 5-3　查看浮点字段结果

5.2　数字常用函数与运算符

MySQL 中一共含有 37 种不同的数字常用函数(Numeric Functions)与运算符，可通过如下 SQL 语句在 MySQL 数据库中查询数字常用函数。

```
//代码位置：全书代码/5.2 数字常用函数与运算符.sql

select
       *
from
       mysql.help_topic ht
where
       ht.help_category_id = (
             select
```

```
            hc.help_category_id
        from
            mysql.help_category hc
        where
            hc.name = 'Numeric Functions'              #数字处理相关函数
    );
```

在实际工作中对于 MySQL 数字处理大多数基于加减乘除和百分数，其余函数使用较少，例如正弦函数 sin()、余弦函数 cos()、正弧线函数 asin()、正切函数 atan()、余切函数 cot()，此处仅列举部分 MySQL 处理数字的相关常用函数。

5.2.1 div()函数

div()将对数字进行整数除法，为数字处理类函数。从除法结果中丢弃任何小数部分，SQL 语句如下：

```
select 5 div 2, -5 div 2, 5 div -2, -5 div -2;
-- 返回结果 2, -2, -2, 2
```

5.2.2 abs()函数

abs()函数将返回绝对值，为数字处理类函数。若输入内容为 null 关键字，则会返回 null，SQL 语句如下：

```
select abs(2);                                          #返回 2
select abs(-32);                                        #返回 32
```

5.2.3 ceiling()函数

ceiling(X)函数将返回不小于 X 的最小整数值，为数字处理类函数，SQL 语句如下：

```
select ceiling(1.23);                                   #返回 2
select ceiling(-1.23);                                  #返回 -1
```

5.2.4 floor()函数

floor(X)函数将返回不大于 X 的最大整数值，为数字处理类函数，SQL 语句如下：

```
select floor(1.23), floor(-1.23);                       #返回 1, -2
```

5.2.5 pow()函数和 power()函数

pow(X,Y)函数将返回 X 的 Y 次幂，为数字处理类函数，SQL 语句如下：

```
select pow(2,2);                                    #返回 4
select pow(2, -2);                                  #返回 0.25
```

power(X,Y)函数是 pow(X,Y)函数的别名,效果完全相同。

5.2.6　rand()函数

rand()返回范围为 0≤v<1.0 的随机浮点值,v 为返回的值,为数字处理类函数。如需返回整数,则需使用以下的公式:

```
FLOOR(i + RAND() * (j - i))
```

例如要获得随机整数在范围 7≤R<12 内,SQL 语句如下:

```
select floor(7 + (rand() * 5));
```

5.2.7　truncate()函数

truncate(X,D)函数返回数字 X,截断为 D 位小数。为数字处理类函数。若 D 为 0,则结果并没有小数点或小数部分。D 可以是负数。使值 X 的小数点左边的 D 位变为 0,SQL 语句如下:

```
//代码位置: 全书代码/5.2.7 truncate()函数.sql

select truncate(1.223,1);                           #返回 1.2
select truncate(1.999,1);                           #返回 1.9
select truncate(1.999,0);                           #返回 1
select truncate( -1.999,1);                         #返回 -1.9
select truncate(122, -2);                           #返回 100
select truncate(10.28 * 100,0);                     #返回 1028
```

5.3　聚合函数

MySQL 还经常运用聚合函数和修饰符(Aggregate Functions and Modifiers)类型的函数。在 MySQL 数据库中查询聚合函数和修饰符的 SQL 语句如下。

```
//5.3 聚合函数.sql

select
       *
from
      mysql.help_topic ht
where
      ht.help_category_id = (
            select
                  hc.help_category_id
```

```
        from
              mysql.help_category hc
        where
              hc.name = 'Aggregate Functions and Modifiers'
                    #聚合函数和修饰符
    );
```

聚合函数和修饰符是 MySQL 中比较特殊的函数类型,聚合函数的特点是可以根据 group by 关键字进行分组聚合。

group by 分组必须跟随在聚合函数之后,但聚合函数的使用不一定非得含有 group by 分组。

5.3.1 count(distinct)函数

count(distinct)函数可进行去重统计总行数,SQL 语句如下:

```
select count(distinct sal) from emp;                    #返回 11
select count(sal) from emp;                             #返回 14
```

5.3.2 查询每个部门的平均薪资

例如公司表(emp 表)中含有薪资列(sal 列),但是当直接使用 avg()函数对 emp 表的 sal 列进行聚合计算平均值时会算出所有人员的薪资平均值。若想按照公司的不同部门进行不同平均值计算,则需使用 group by 对 emp 表的部门进行分组。

因为聚合函数将会运行在 group by 关键字之后,所以可以进行分组聚合,SQL 语句如下:

```
//5.3 聚合函数.sql

---查询每个部门的平均薪资
select
   deptno,
   avg(sal)
from
   emp
group by
   deptno;
```

聚合函数中包括的返回列最大值 max()函数、返回列最小值 min()函数、返回列平均值 avg()函数、返回列总和 sum()函数、返回列平均值 avg()函数、返回列总行数 count()函数、返回列总行数且去重 count(distinct)函数等。

聚合函数只需在函数中输入列名便可以使用,针对列名进行聚合计算。

所有聚合函数皆可使用 group by 进行分组。

5.3.3　查询每个部门的薪资最高与最低的人(携带提成)

在 emp 表中含有 comm 字段代表提成,正常情况下只需使用 sal+comm 便可以获得携带提成的薪资,但此张表含有问题,即 comm 字段可能为 null。若使用了 1+null 的方式,则会返回 null,SQL 语句如下:

```
select 1 + null;                          #返回 null
```

因为提成中含有 null 值,所以此刻不能直接使用 sal+comm 的方式获取携带提成的薪资。

因为此时需要排除 null 字段,但若使用 where 语句排除,则可能会把不含提成的人连月薪都不进行统计了,所以此处需要使用前文提过的 coalesce()函数将 null 值转换为 0,SQL 语句如下:

```
select sal + coalesce(comm,0) from emp;
```

运行后,结果集如图 5-4 所示。

```
+----------------------+
| sal+coalesce(comm,0) |
+----------------------+
|                  800 |
|                 1900 |
|                 1750 |
|                 3000 |
|                 1250 |
|                 4250 |
|                 2450 |
|                 3000 |
|                 5000 |
|                 1500 |
|                 1100 |
|                  950 |
|                 3000 |
|                 1300 |
+----------------------+
14 rows in set (0.02 sec)
```

图 5-4　coalesce 转换后的结果

此时就可把工资与提成加到一起了,并且没有遗漏任何人,增加 group by 针对部门进行分组即可实现“每个部门的薪资最高与最低的人(携带提成)”的需求,SQL 语句如下:

```
//5.3 聚合函数.sql

select
   deptno as '部门编号',
   max(sal + coalesce(comm, 0)) as '部门最高薪资(携带提成)',
   min(sal + coalesce(comm, 0)) as '部门最低薪资(携带提成)'
from
   emp
group by
   deptno;
```

运行后,结果集如图 5-5 所示。

```
+--------------+-----------------------------------+-----------------------------------+
| 部门编号     | 部门最高薪资(携带提成)            | 部门最低薪资(携带提成)            |
+--------------+-----------------------------------+-----------------------------------+
|           20 |                              3000 |                               800 |
|           30 |                              4250 |                               950 |
|           10 |                              5000 |                              1300 |
+--------------+-----------------------------------+-----------------------------------+
3 rows in set (0.01 sec)
```

图 5-5　最终结果

5.3.4 查询每个部门的薪资总额

MySQL 使用 sum()函数记录相加后的总和，并且可通过 group by 针对 emp 表进行分组，这样便可获得每个部门的薪资总额，SQL 语句如下：

```
//5.3 聚合函数.sql

select
   deptno as '部门编号',
   sum(sal) as '部门的薪资总额'
from
   emp
group by
   deptno;
```

运行后，结果集如图 5-6 所示。

```
+--------------+--------------------+
| 部门编号     | 部门的薪资总额     |
+--------------+--------------------+
|           20 |              10900 |
|           30 |               9400 |
|           10 |               8750 |
+--------------+--------------------+
3 rows in set (0.01 sec)
```

图 5-6 查询每个部门的薪资总额结果集

5.3.5 查询每个部门有多少人

MySQL 使用 count()记录总行数，通过 group by 针对 emp 表进行分组，即可获得每个部门有多少人，SQL 语句如下：

```
//5.3 聚合函数.sql

select
   deptno as '部门编号',
   count(sal) as '总人数'
from
   emp
group by
   deptno;
```

运行后，结果集如图 5-7 所示。

```
+--------------+----------+
| 部门编号     | 总人数   |
+--------------+----------+
|           20 |        5 |
|           30 |        6 |
|           10 |        3 |
+--------------+----------+
3 rows in set (0.01 sec)
```

图 5-7 查询每个部门有多少人的结果集

5.3.6　查询每个部门有多少人没有提成

首先需要判断没有提成的人，即 emp 表中 comm 字段为空，然后因为需求中提到了需要根据每个部门进行分组，所以需要使用 group by，最后需要使用 count()函数统计人数，SQL 语句如下：

```
//5.3 聚合函数.sql

select
   deptno as '部门名称',
   count(ename) as '人数'
from
   emp e
where
   e.comm is null
or
   e.comm = 0
group by
   deptno;
```

运行后，结果集如图 5-8 所示。

```
+--------------+--------+
| 部门名称     | 人数   |
+--------------+--------+
|           20 |      5 |
|           30 |      3 |
|           10 |      3 |
+--------------+--------+
3 rows in set (0.00 sec)
```

图 5-8　查询每个部门有多少人没提成的结果集

5.3.7　查询某个部门薪资占全公司的百分比

查询部门编号为 10 的部门，其薪资总额占全公司薪资总额的百分比，SQL 语句如下：

```
//5.3 聚合函数.sql

select
   ( select sum(e.sal) from emp e where e.deptno = 10 )
   /
   ( select sum(e.sal) from emp e )
   *
   100
   as result;
```

运行后，结果集如图 5-9 所示。

```
+---------+
| result  |
+---------+
| 30.1205 |
+---------+
1 row in set (0.00 sec)
```

图 5-9　查询编号为 10 的部门薪资占全公司薪资百分比的结果集

当然此需求还有其他写法，例如使用后文将会提到的 case when 方式可以使用一条 SQL 语句就查询出来结果，SQL 语句如下：

```
//5.3 聚合函数.sql

select
   (
      sum(case when deptno = 10 then sal end)
      /
      sum(sal)
   ) * 100 as pct
from
   emp;
```

5.4 窗口函数

在 MySQL 8.0 中推出了新的函数类型，即窗口类函数（Window Functions），窗口函数可对结果集中的每行进行计算，或使用与该行相关的行进行计算。

5.4.1 窗口函数的语法

窗口函数的基本语法如下：

```
//5.4 窗口函数.sql

<窗口函数> over (
      partition by <用于分区的列名>
      order by <用于排序的列名>
      frame_clause <窗口大小>
)
```

如下 SQL 语句可在 MySQL 数据库中查询非聚合类窗口函数。之所以称 Window Functions 为非聚合类窗口函数，是因为在 MySQL 中聚合函数也可以作为窗口函数使用。

```
//5.4 窗口函数.sql

select
      *
from
      mysql.help_topic ht
where
      ht.help_category_id = (
            select
                  hc.help_category_id
            from
                  mysql.help_category hc
            where
                  hc.name = 'Window Functions'                    #窗口类型函数
      );
```

MySQL 中的非聚合类窗口函数统计如表 5-2 所示。

表 5-2　MySQL 非聚合类窗口函数统计

函 数 名 称	功能简易释义
cume_dist()	cume_dist()函数返回值的累计分布值。该值小于或等于行的值除以总行数
dense_rank()	分区内当前行的排名,无间隔
first_value()	允许使用 window frame 获取分区或结果集第 1 行的值
lag()	返回当前行之前的值
last_value()	选择数据中的最后一行
lead()	向前看多行并从当前行访问行的数据
nth_value()	从有序行集中的第 n 行获取值
ntile()	排序分区中的行并划分为特定数量的组。每个组分配一个分组号
percent_rank()	百分比排名值
rank()	分区内当前行的排名,有间隔
row_number()	分区内的当前行数

5.4.2　初步使用窗口函数

row_number()函数是最常用的窗口函数,row_number()函数可以给任意结果集增加序号和排序。查询 emp 全表的 SQL 语句如下:

```
//5.4 窗口函数.sql

select
   e.ename,
   e.job,
   e.deptno,
   e.sal
from
   emp e;
```

运行后,结果集如图 5-10 所示。

```
+--------+----------+--------+------+
| ename  | job      | deptno | sal  |
+--------+----------+--------+------+
| 张三   | 店员     |     20 |  800 |
| 李四   | 售货员   |     30 | 1600 |
| 王五   | 售货员   |     30 | 1250 |
| 赵六   | 经理     |     20 | 3000 |
| 薛七   | 售货员   |     30 | 1250 |
| 陈八   | 经理     |     30 | 2850 |
| 吴九   | 经理     |     10 | 2450 |
| 寅十一 | 文员     |     20 | 3000 |
| 王十二 | 总经理   |     10 | 5000 |
| 黄十三 | 售货员   |     30 | 1500 |
| 毛十四 | 店员     |     20 | 1100 |
| 陈十五 | 店员     |     30 |  950 |
| 张十六 | 文员     |     20 | 3000 |
| 刘十七 | 店员     |     10 | 1300 |
+--------+----------+--------+------+
14 rows in set (0.00 sec)
```

图 5-10　初步使用窗口函数的结果集

此时可以通过 row_number()函数增加行号列,SQL 语句如下:

```
//5.4 窗口函数.sql

select
   row_number() over() as row_num,
   e.ename,
   e.job,
   e.deptno,
   e.sal
from
   emp e;
```

运行后,结果集如图 5-11 所示。在该结果集中增加了 row_num 行号列。

```
+---------+--------+-----------+--------+------+
| row_num | ename  | job       | deptno | sal  |
+---------+--------+-----------+--------+------+
|       1 | 张三   | 店员      |     20 |  800 |
|       2 | 李四   | 售货员    |     30 | 1600 |
|       3 | 王五   | 售货员    |     30 | 1250 |
|       4 | 赵六   | 经理      |     20 | 3000 |
|       5 | 薛七   | 售货员    |     30 | 1250 |
|       6 | 陈八   | 经理      |     30 | 2850 |
|       7 | 吴九   | 经理      |     10 | 2450 |
|       8 | 寅十一 | 文员      |     20 | 3000 |
|       9 | 王十二 | 总经理    |     10 | 5000 |
|      10 | 黄十三 | 售货员    |     30 | 1500 |
|      11 | 毛十四 | 店员      |     20 | 1100 |
|      12 | 陈十五 | 店员      |     30 |  950 |
|      13 | 张十六 | 文员      |     20 | 3000 |
|      14 | 刘十七 | 店员      |     10 | 1300 |
+---------+--------+-----------+--------+------+
14 rows in set (0.00 sec)
```

图 5-11　增加 row_num 列的结果集

5.4.3　partition by 关键字

partition by 用于将数据行拆分成多个分区,窗口函数基于每行数据所在的区进行计算并返回结果,partition by 的作用类似于 group by 分组。若省略了 partition by 关键字,则所有的数据作为一个组进行计算。

在增加行号的例子中,此时可以增加分区关键字,让行号根据部门编号(deptno)进行分区,SQL 语句如下:

```
//5.4.3 partition by关键字.sql

select
   row_number() over(
    partition by e.deptno
   ) as row_num,
   e.ename,
   e.job,
   e.deptno,
   e.sal
```

```
from
   emp e;
```

运行后，结果集如图 5-12 所示。可以看到此时结果集根据部门编号进行了分区，以部门编号为 10、20、30 分为不同的区。每个区含有自身不同的行号。

```
+---------+---------+-----------+--------+------+
| row_num | ename   | job       | deptno | sal  |
+---------+---------+-----------+--------+------+
|       1 | 吴九    | 经理      |     10 | 2450 |
|       2 | 王十二  | 总经理    |     10 | 5000 |
|       3 | 刘十七  | 店员      |     10 | 1300 |
|       1 | 张三    | 店员      |     20 |  800 |
|       2 | 赵六    | 经理      |     20 | 3000 |
|       3 | 寅十一  | 文员      |     20 | 3000 |
|       4 | 毛十四  | 店员      |     20 | 1100 |
|       5 | 张十六  | 文员      |     20 | 3000 |
|       1 | 李四    | 售货员    |     30 | 1600 |
|       2 | 王五    | 售货员    |     30 | 1250 |
|       3 | 薛七    | 售货员    |     30 | 1250 |
|       4 | 陈八    | 经理      |     30 | 2850 |
|       5 | 黄十三  | 售货员    |     30 | 1500 |
|       6 | 陈十五  | 店员      |     30 |  950 |
+---------+---------+-----------+--------+------+
14 rows in set (0.04 sec)
```

图 5-12　根据部门编号进行分区的结果集

5.4.4　order by 关键字

order by 排序用于指定分区内的排序方式，与 select 中的 order by 子句的作用类似，通常用于数据的排名分析。

在以上例子中，虽然使用了 partition by 关键字作为分区处理，但是因为分区之后每个部门并不是根据薪资(sal)高低进行排序的，所以此刻可以增加 order by 关键字，针对部门分区后进行排序，SQL 语句如下：

```
//5.4.4 order by 关键字.sql

select
   row_number() over(
    partition by e.deptno
    order by e.sal desc
   ) as row_num,
   e.ename,
   e.job,
   e.deptno,
   e.sal
from
   emp e;
```

运行后，结果集如图 5-13 所示。

```
+---------+-----------+-----------+--------+------+
| row_num | ename     | job       | deptno | sal  |
+---------+-----------+-----------+--------+------+
|       1 | 王十二    | 总经理    |     10 | 5000 |
|       2 | 吴九      | 经理      |     10 | 2450 |
|       3 | 刘十七    | 店员      |     10 | 1300 |
|       1 | 赵六      | 经理      |     20 | 3000 |
|       2 | 寅十一    | 文员      |     20 | 3000 |
|       3 | 张十六    | 文员      |     20 | 3000 |
|       4 | 毛十四    | 店员      |     20 | 1100 |
|       5 | 张三      | 店员      |     20 |  800 |
|       1 | 陈八      | 经理      |     30 | 2850 |
|       2 | 李四      | 售货员    |     30 | 1600 |
|       3 | 黄十三    | 售货员    |     30 | 1500 |
|       4 | 王五      | 售货员    |     30 | 1250 |
|       5 | 薛七      | 售货员    |     30 | 1250 |
|       6 | 陈十五    | 店员      |     30 |  950 |
+---------+-----------+-----------+--------+------+
14 rows in set (0.00 sec)
```

图 5-13　增加 order by 关键字的结果集

5.4.5　rank()函数

rank()是一个窗口函数，rank()函数为分区或结果集中的每行分配排名，而排名值含有间隔。

此时可查询 emp 表中每名员工的薪资，SQL 语句如下：

```
//5.4.5 rank()函数.sql

select
      e.deptno,
      e.ename,
      e.sal
from
      emp e
```

运行后，结果集如图 5-14 所示。

```
+--------+-----------+------+
| deptno | ename     | sal  |
+--------+-----------+------+
|     20 | 张三      |  800 |
|     30 | 李四      | 1600 |
|     30 | 王五      | 1250 |
|     20 | 赵六      | 3000 |
|     30 | 薛七      | 1250 |
|     30 | 陈八      | 2850 |
|     10 | 吴九      | 2450 |
|     20 | 寅十一    | 3000 |
|     10 | 王十二    | 5000 |
|     30 | 黄十三    | 1500 |
|     20 | 毛十四    | 1100 |
|     30 | 陈十五    |  950 |
|     20 | 张十六    | 3000 |
|     10 | 刘十七    | 1300 |
+--------+-----------+------+
14 rows in set (0.00 sec)
```

图 5-14　查询 emp 表中每名员工的薪资的结果集

可通过 rank()函数查询每个部门薪资的排序，SQL 语句如下：

```
//5.4.5 rank()函数.sql

select
   rank()over(
      partition by e.deptno
      order by sal desc
   ) as rank_num,
   e.deptno,
   e.ename,
   e.sal
from
   emp e
```

运行后，结果集如图 5-15 所示。

```
+----------+--------+-----------+------+
| rank_num | deptno | ename     | sal  |
+----------+--------+-----------+------+
|        1 |     10 | 王十二    | 5000 |
|        2 |     10 | 吴九      | 2450 |
|        3 |     10 | 刘十七    | 1300 |
|        1 |     20 | 赵六      | 3000 |
|        1 |     20 | 寅十一    | 3000 |
|        1 |     20 | 张十六    | 3000 |
|        4 |     20 | 毛十四    | 1100 |
|        5 |     20 | 张三      |  800 |
|        1 |     30 | 陈八      | 2850 |
|        2 |     30 | 李四      | 1600 |
|        3 |     30 | 黄十三    | 1500 |
|        4 |     30 | 王五      | 1250 |
|        4 |     30 | 薛七      | 1250 |
|        6 |     30 | 陈十五    |  950 |
+----------+--------+-----------+------+
14 rows in set (0.01 sec)
```

图 5-15　通过 rank()函数查询每个部门薪资排序的结果集

此时可看出每个部门含有最高薪资的人，例如部门编号为 10 的薪资最高的人为王十二，部门编号为 20 的共有 3 人薪资最高，其薪资均为 3000 元。部门编号为 20 的薪资排行，没有第 2 名和第 3 名，只有第 1 名和第 4 名以后。此种排名方式被称为“含有间隔”排名。

rank()函数只能作为排序所存在，并不能直接获取 rank_num 列等于 1 的表达式，如想只获取 rank_num 薪资最高者，则需要在外再嵌套一层 select 查询语句，SQL 语句如下：

```
//5.4.5 rank()函数.sql

select
    *
from
   (
      select
         rank()over(
            partition by e.deptno
            order by
               sal desc
         ) as rank_num,
         e.deptno,
         e.ename,
         e.sal
      from
         emp e
   ) la
where
    la.rank_num = 1;
```

运行后，结果集如图 5-16 所示。

```
+----------+--------+-----------+------+
| rank_num | deptno | ename     | sal  |
+----------+--------+-----------+------+
|        1 |     10 | 王十二    | 5000 |
|        1 |     20 | 赵六      | 3000 |
|        1 |     20 | 寅十一    | 3000 |
|        1 |     20 | 张十六    | 3000 |
|        1 |     30 | 陈八      | 2850 |
+----------+--------+-----------+------+
5 rows in set (0.03 sec)
```

图 5-16　获取每个部门薪资最高者的结果集

5.4.6　dense_rank()函数

dense_rank()是一个窗口函数，dense_rank()函数为分区或结果集中的每行分配排名，而排名值没有间隔。

可以使用 dense_rank()函数对部门进行分区以获取数据，查询每个部门的薪资排名，SQL 语句如下：

```
//5.4.6 dense_rank()函数.sql

select
   e.deptno,
   e.sal,
   dense_rank()over(
      partition by e.deptno
      order by
         e.sal desc
   ) as rnk
from
   emp e;
```

运行后，结果集如图 5-17 所示。部门编号为 20 的组员薪资并列第 1 名的，rnk 值皆为 1，其低于第 1 名的为第 2 名。该排名方式被称作“没有间隔”。

```
+--------+------+-----+
| deptno | sal  | rnk |
+--------+------+-----+
|     10 | 5000 |   1 |
|     10 | 2450 |   2 |
|     10 | 1300 |   3 |
|     20 | 3000 |   1 |
|     20 | 3000 |   1 |
|     20 | 3000 |   1 |
|     20 | 1100 |   2 |
|     20 |  800 |   3 |
|     30 | 2850 |   1 |
|     30 | 1600 |   2 |
|     30 | 1500 |   3 |
|     30 | 1250 |   4 |
|     30 | 1250 |   4 |
|     30 |  950 |   5 |
+--------+------+-----+
14 rows in set (0.41 sec)
```

图 5-17　体验“没有间隔”特性的结果集

之前的 rank()函数，部门编号为 20 的组员工资并列第 1 名的，其 rnk 值皆为 1，此时低于第 1 名的为第 4 名。该排名方式被称作“含有间隔”。

5.4.7　percent_rank()函数

percent_rank()函数将会计算分区或结果集中行的百分位排名。例如此时可以使用 percent_rank()函数查询每名员工占全公司薪资的百分位排名，SQL 语句如下：

```
//5.4.7 percent_rank()函数.sql

select
   e.ename,
   e.sal,
   percent_rank() over (
      order by
         e.sal asc
   ) as '百分位排名'
from
   emp e;
```

运行后，结果集如图 5-18 所示。

如果使用 asc 排序方式，则会让薪资最高的人排在最后，若使用 desc 排序方式，则会让薪资最低的人排在最后。

因为张十六的薪资在公司约 77%的人之上，全公司共 14 人，所以张十六的薪资处于全公司约 10 名以上（14 人×77%）。

由于赵六、寅十一、张十六的薪资并列，所以此 3 人的百分位均约为 77%。

若嫌弃百分位小数过多，则可使用 round()函数精确小数。本示例将百分位精确至 2 位小数即可，SQL 语句如下：

ename	sal	百分位排名
张三	800	0
陈十五	950	0.07692307692307693
毛十四	1100	0.15384615384615385
王五	1250	0.23076923076923078
薛七	1250	0.23076923076923078
刘十七	1300	0.38461538461538464
黄十三	1500	0.46153846153846156
李四	1600	0.5384615384615384
吴九	2450	0.6153846153846154
陈八	2850	0.6923076923076923
赵六	3000	0.7692307692307693
寅十一	3000	0.7692307692307693
张十六	3000	0.7692307692307693
王十二	5000	1

14 rows in set (0.01 sec)

图 5-18　每名员工占全公司薪资的百分位排名的结果集

```
//5.4.7 percent_rank()函数.sql

select
   e.ename,
   e.sal,
   round(
      percent_rank()over (
         order by
            e.sal asc
```

```
        ),
        2
    ) as '百分位排名'
from
    emp e;
```

运行后，结果集如图 5-19 所示。

```
+-----------+------+-----------------+
| ename     | sal  | 百分位排名      |
+-----------+------+-----------------+
| 张三      |  800 |               0 |
| 陈十五    |  950 |            0.08 |
| 毛十四    | 1100 |            0.15 |
| 王五      | 1250 |            0.23 |
| 薛七      | 1250 |            0.23 |
| 刘十七    | 1300 |            0.38 |
| 黄十三    | 1500 |            0.46 |
| 李四      | 1600 |            0.54 |
| 吴九      | 2450 |            0.62 |
| 陈八      | 2850 |            0.69 |
| 赵六      | 3000 |            0.77 |
| 寅十一    | 3000 |            0.77 |
| 张十六    | 3000 |            0.77 |
| 王十二    | 5000 |               1 |
+-----------+------+-----------------+
14 rows in set (0.02 sec)
```

图 5-19　使用 round()函数精确小数的结果集

5.4.8　ntile()函数

ntile()函数可以辅助用户在分组后给每个分组增加一个编号，方便后续获取。ntile()函数的括号中需要输入一个正整数值 n，在分组后，序号将会将 n 设置为最大值。

若使用了最大值的 n 之后还有分组没被增加编号的情况，则将会从 1 重新计数，即假设含有 3 个分组，但 n 被设置成了 2，则含有分组编号为分组 1、分组 2、分组 1。

若使用了一部分 n 之后没有分组需要使用编号了，则与正常展示无异。

例如，此时使用 ntile()函数针对每个部门薪资总额分组后进行编号，SQL 语句如下：

```
//5.4.8 ntile()函数.sql

select
    e.deptno,
    sum(e.sal) sum_num,
    ntile(3) over (
        order by e.deptno
    ) group_no
from
    emp e
group by
     e.deptno
```

运行后，结果集如图 5-20 所示。

```
+--------+---------+----------+
| deptno | sum_num | group_no |
+--------+---------+----------+
|     10 |    8750 |        1 |
|     20 |   10900 |        2 |
|     30 |    9400 |        3 |
+--------+---------+----------+
3 rows in set (0.01 sec)
```

图 5-20　给分组增加编号的结果集

5.5　聚合函数窗口化

所有的聚合函数均支持窗口化的使用方式，其中包括前文提到过的 max()、min()、sum()、group_concat()等函数。只需在常规聚合函数之后增加 over()子句。

窗口函数是 MySQL 中最特殊的函数形式，其特殊性在于窗口操作不会将查询行组折叠成单个输出行，相反窗口函数会为每行产生结果。

该特性在 5.4 节有所体现，此处可以使用两种 sum()函数分别进行演示，分别为使用聚合函数和聚合函数窗口化的方式。

使用聚合函数查询全公司总月薪的 SQL 语句如下：

```
select sum(sal) from emp;
```

运行后，结果集如图 5-21 所示。

使用聚合函数窗口化的方式查询全公司总月薪的 SQL 语句如下：

```
select sum(sal) over() from emp;
```

运行后，结果集如图 5-22 所示。因为含有 14 个员工，所以其数字不会折叠，会展示出 14 行。

```
+----------+
| sum(sal) |
+----------+
|    29050 |
+----------+
1 row in set (0.01 sec)
```

图 5-21　使用聚合函数查询全公司总月薪的结果集

```
+-----------------+
| sum(sal) over() |
+-----------------+
|           29050 |
|           29050 |
|           29050 |
|           29050 |
|           29050 |
|           29050 |
|           29050 |
|           29050 |
|           29050 |
|           29050 |
|           29050 |
|           29050 |
|           29050 |
|           29050 |
+-----------------+
14 rows in set (0.01 sec)
```

图 5-22　聚合函数窗口化使数字不折叠的结果集

使用聚合函数查询公司内每个部门的平均薪资，SQL 语句如下：

```
select
   deptno,
   avg(sal)
from
   emp
group by
   deptno;
```

运行后，结果集如图 5-23 所示。

使用聚合函数窗口化的方式查询公司内每个部门的平均薪资，SQL 语句如下：

```
select
   deptno,
   avg(sal) over(partition by deptno) as 'avg(sal)'
from
   emp;
```

运行后，结果集如图 5-24 所示。

```
+--------+-----------+
| deptno | avg(sal)  |
+--------+-----------+
|     20 | 2180.0000 |
|     30 | 1566.6667 |
|     10 | 2916.6667 |
+--------+-----------+
3 rows in set (0.00 sec)
```

图 5-23 使用聚合函数查询公司内每个部门的平均薪资的结果集

```
+--------+-----------+
| deptno | avg(sal)  |
+--------+-----------+
|     10 | 2916.6667 |
|     10 | 2916.6667 |
|     10 | 2916.6667 |
|     20 | 2180.0000 |
|     20 | 2180.0000 |
|     20 | 2180.0000 |
|     20 | 2180.0000 |
|     20 | 2180.0000 |
|     30 | 1566.6667 |
|     30 | 1566.6667 |
|     30 | 1566.6667 |
|     30 | 1566.6667 |
|     30 | 1566.6667 |
|     30 | 1566.6667 |
+--------+-----------+
14 rows in set (0.34 sec)
```

图 5-24 以聚合函数窗口化的方式查询公司内每个部门的平均薪资的结果集

5.6 MySQL 8.0 处理数字相关的复杂查询

以下为有关 MySQL 处理数字相关的复杂查询与详解。

5.6.1 计算众数

1. 问题

众数的数学定义是“在给定数据集中出现频率最高的元素”，例如需要找出 20 号部门的员工薪水众数，SQL 语句如下：

```
select sal from emp where deptno = 20 order by sal;
```

运行后，计算众数之前的结果集如图 5-25 所示。计算众数之后的结果集如图 5-26 所示。

```
+------+
| sal  |
+------+
|  800 |
| 1100 |
| 3000 |
| 3000 |
| 3000 |
+------+
5 rows in set (0.00 sec)
```

图 5-25　计算众数之前

```
+------+
| sal  |
+------+
| 3000 |
+------+
1 row in set (0.01 sec)
```

图 5-26　计算众数之后

2. SQL 演化

首先求出 20 号部门的各个阶层薪资及各阶层薪资的人数，SQL 语句如下：

```
//5.6.1 计算众数.sql

select
   sal,
   count( * ) as cnt
from
   emp
where
   deptno = 20
group by
   sal
```

运行后，结果集如图 5-27 所示。

```
+------+-----+
| sal  | cnt |
+------+-----+
|  800 |   1 |
| 3000 |   3 |
| 1100 |   1 |
+------+-----+
3 rows in set (0.01 sec)
```

图 5-27　20 号部门的各个阶层薪资及各阶层薪资的人数的结果集

可以看出，月薪一共含有 3 个等级，分别是 800、3000、1100，其中 3000 占的人数最多，含有 3 人，而后可根据各阶层薪资的人数 cnt 进行排序，SQL 语句如下：

```
//5.6.1 计算众数.sql

select
   sal,
   dense_rank() over(
      order by
         cnt desc
   ) as rnk
from
```

```
    (
       select
          sal,
          count( * ) as cnt
       from
          emp
       where
          deptno = 20
       group by
          sal
    ) x;
```

运行后,结果集如图 5-28 所示,即薪资为 3000 等级的人数排第 1 位,800 与 1100 等级的并列第 2 位。

```
+------+-----+
| sal  | rnk |
+------+-----+
| 3000 |   1 |
|  800 |   2 |
| 1100 |   2 |
+------+-----+
3 rows in set (0.01 sec)
```

图 5-28　根据分级薪资人数排序的结果集

最终获取排序 rnk 为 1 的数据即可,这就是部门编号为 20 的薪资里哪种薪资类型为最高,也就是找出了 20 号部门的员工薪资众数,SQL 语句如下:

```
//5.6.1 计算众数.sql

select
   sal
from
   (
      select
         sal,
         dense_rank() over(
            order by
               cnt desc
         ) as rnk
      from
         (
            select
               sal,
               count( * ) as cnt
            from
               emp
            where
               deptno = 20
            group by
               sal
         ) x
```

```
    ) y
where
    rnk = 1;
```

5.6.2　计算中值

1. 问题

中值的数学定义是："中值(又称中位数)是指将统计总体中的各个变量值按大小顺序排列起来,形成一个数列,处于变量数列中间位置的变量值就称为中位数",此时需要查出部门编号为 20 的员工薪资的中值是多少。

为计算中值,按照数学定义将部门编号为 20 的员工薪资的变量值按大小顺序排列起来,形成一个数列,SQL 语句如下:

```
//5.6.2 计算中值.sql

select
    row_number() over (
        order by e.sal
    ) row_num,
    e.sal
from
    emp e
where
    e.deptno = '20'
```

运行后,计算中值之前的结果集如图 5-29 所示。计算中值之后的结果集如图 5-30 所示。

```
+---------+------+
| row_num | sal  |
+---------+------+
|       1 |  800 |
|       2 | 1100 |
|       3 | 3000 |
|       4 | 3000 |
|       5 | 3000 |
+---------+------+
5 rows in set (0.06 sec)
```

图 5-29　计算中值之前的结果集

```
+---------+------+
| row_num | sal  |
+---------+------+
|       3 | 3000 |
+---------+------+
1 row in set (0.02 sec)
```

图 5-30　计算中值之后的结果集

2. SQL 演化

获取部门编号为 20 的员工总数,并除以 2 获取中值的位置,SQL 语句如下:

```
//5.6.2 计算中值.sql

select
    floor(count(e.sal) / 2) + 1
from
    emp e
where
    e.deptno = '20'
```

运行后，结果集如图 5-31 所示。

```
+----------------------------+
| floor(count(e.sal) / 2) + 1 |
+----------------------------+
|                          3 |
+----------------------------+
1 row in set (0.01 sec)
```

图 5-31　获取中值位置的结果集

拼接两条 SQL 语句。根据排列后的数列进行获取，只需获取 row_num 为 3 的薪资数目便可以查出部门编号为 20 的员工薪资的中值是多少，SQL 语句如下：

```
//5.6.2 计算中值.sql

select
    *
from
   (
      select
         row_number() over (
            order by e.sal
         ) row_num,
         e.sal
      from
         emp e
      where
         e.deptno = '20'
   ) as rum
where
   rum.row_num = (
      select
         floor(count(e.sal) / 2) + 1
      from
         emp e
      where
         e.deptno = '20'
   );
```