第08章_聚合函数

我们上一章讲到了 SQL 单行函数。实际上 SQL 函数还有一类

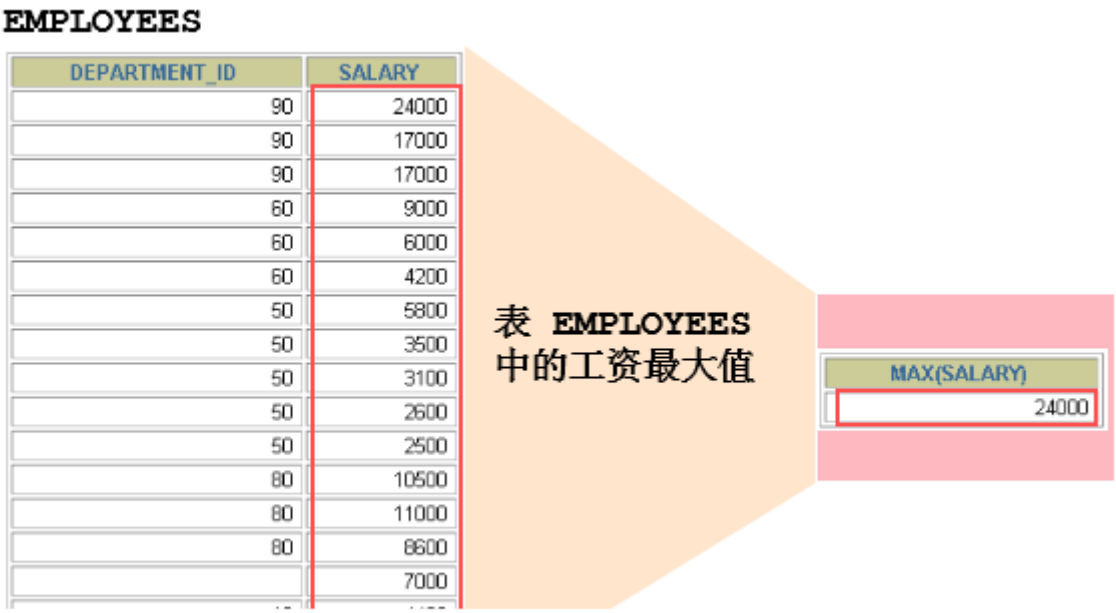
叫做聚合（或聚集、分组）函数，它是对一组数据进行汇总的函数，输入的是一组数据的集合

输出的是单个值

1. 聚合函数介绍

- **什么是聚合函数**

聚合函数作用于一组数据，并对一组数据返回一个值。



常用的聚合函数：

- **聚合函数类型**
  - **AVG()**
  - **SUM()**
  - **MAX()**
  - **MIN()**
  - **COUNT()**
- 聚合函数语法

```
SELECT        [column,] group function(column), ...
FROM          table
[WHERE        condition]
[GROUP BY     column]
[ORDER BY     column];
```

- 聚合函数不能嵌套调用。比如不能出现类似“AVG(SUM(字段名称))”形式的调用。

1.1 AVG和SUM函数

可以对**数值型数据**使用AVG 和 SUM 函数

注意：对字符串、日期都不适用

```
SELECT AVG(salary), MAX(salary),MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';

+-------------+-------------+-------------+-------------+
| AVG(salary) | MAX(salary) | MIN(salary) | SUM(salary) |
+-------------+-------------+-------------+-------------+
| 8272.727273 |    11500.00 |     6000.00 |   273000.00 |
+-------------+-------------+-------------+-------------+
1 row in set (0.00 sec)
```

1.2 MIN和MAX函数

可以对**任意数据类型**的数据使用 MIN 和 MAX 函数

既可以对字符串、日期进行比较

```
SELECT MIN(hire_date), MAX(hire_date)
FROM      employees;

+----------------+----------------+
| MIN(hire_date) | MAX(hire_date) |
+----------------+----------------+
| 1987-06-17     | 2000-04-21     |
+----------------+----------------+
1 row in set (0.00 sec)
```

1.3 COUNT函数

- COUNT(*)返回表中记录总数，适用于**任意数据类型。**

```
SELECT COUNT(*)
FROM employees

+----------+
| COUNT(*) |
+----------+
|      107 |
+----------+
1 row in set (0.00 sec)
```

- COUNT(expr)

  返回**expr不为空**的记录总数。

```
SELECT COUNT(commission_pct)
FROM   employees

+-----------------------+
| COUNT(commission_pct) |
+-----------------------+
|                    35 |
+-----------------------+
1 row in set (0.00 sec)
```

```
#方式1：COUNT(*)
#方式2：COUNT(1)
#方式3：COUNT(具体字段) ： 不一定对！【因为null不算】

#② 注意：计算指定字段出现的个数时，是不计算NULL值的。
SELECT COUNT(commission_pct)
FROM employees;
+-----------------------+
| COUNT(commission_pct) |
+-----------------------+
|                    35 |
+-----------------------+
-- 表中数据一共有107 rows in set (0.00 sec)
```

```
-- 验证
SELECT commission_pct
FROM employees
WHERE commission_pct IS NOT NULL;
35 rows in set (0.00 sec)
```

- **问题：用count(*)，count(1)，count(列名)谁好呢?**

  其实，对于MyISAM引擎的表是没有区别的。这种引擎内部有一计数器在维护着行数。

  Innodb引擎的表用count(*),count(1)直接读行数，复杂度是O(n)，因为innodb真的要去数一遍。但好于具体的count(列名)。

- **问题：能不能使用count(列名)替换count(*)?**

  不要使用 count(列名)来替代 `count(*)`，`count(*)` 是 SQL92 定义的标准统计行数的语法，跟数据库无关，跟 NULL 和非 NULL 无关。

  说明：count(*)会统计值为 NULL 的行，而 count(列名)不会统计此列为 NULL 值的行。

- **公式：AVG = SUM / COUNT**

```
-- SUN COUNT AVG 有NULL不算 所以 AVG(commission_pct) =
SUM(commission_pct)/COUNT(commission_pct)
SELECT AVG(salary),SUM(salary)/COUNT(salary),
AVG(commission_pct),SUM(commission_pct)/COUNT(commission_pct),
SUM(commission_pct) / 107
FROM employees;
+-------------+--------------------------+--------------------+-------------
------------------------------+-------------------
|AVG(salary)|SUM(salary)/COUNT(salary)|AVG(commission_pct)|SUM(commission_pct)
/COUNT(commission_pct) | SUM(commission_pct) / 107 |
+-------------+--------------------------+--------------------+-------------
------------------------------+-------------------
| 6461.682243 |           6461.682243 |          0.222857 |
              0.222857 |                  0.072897 |
+-------------+--------------------------+--------------------+-------------
------------------------------+-------------------
```

- **需求：查询公司中平均奖金率**

```
#错误的!
-- 因为是平均 所以是个人就要算 聚合函数AVG是不会算NULL
SELECT AVG(commission_pct)
FROM employees;
```

```
#正确的:
SELECT SUM(commission_pct) / COUNT(IFNULL(commission_pct,0)),
AVG(IFNULL(commission_pct,0))
FROM employees;
```

总结：

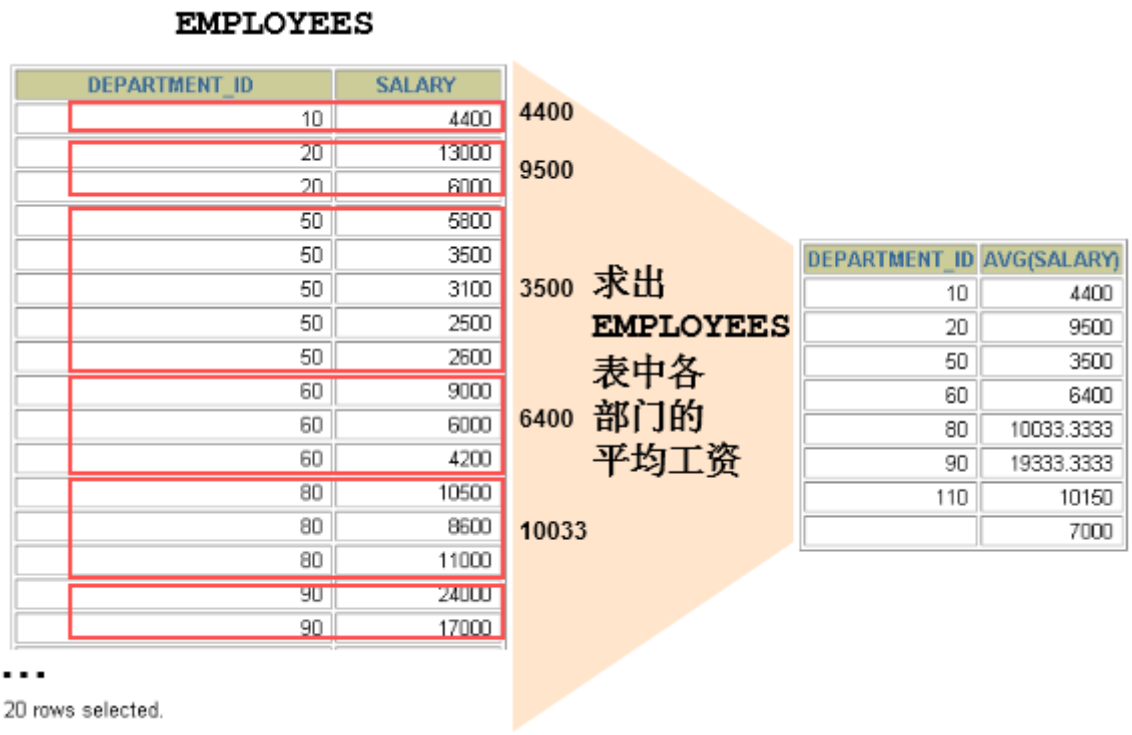
1、如何需要统计表中的记录数，使用COUNT(*)、COUNT(1)、COUNT(具体字段) 哪个效率更高呢？

2、如果使用的是MyISAM 存储引擎，则三者效率相同，都是O(1)，因为他会维护一个字段存储总记录数

3、如果使用的是InnoDB 存储引擎，则三者效率：COUNT(*) = COUNT(1)> COUNT(字段)

2. GROUP BY

2.1 基本使用



- **查询各个部门的平均工资，最高工资**

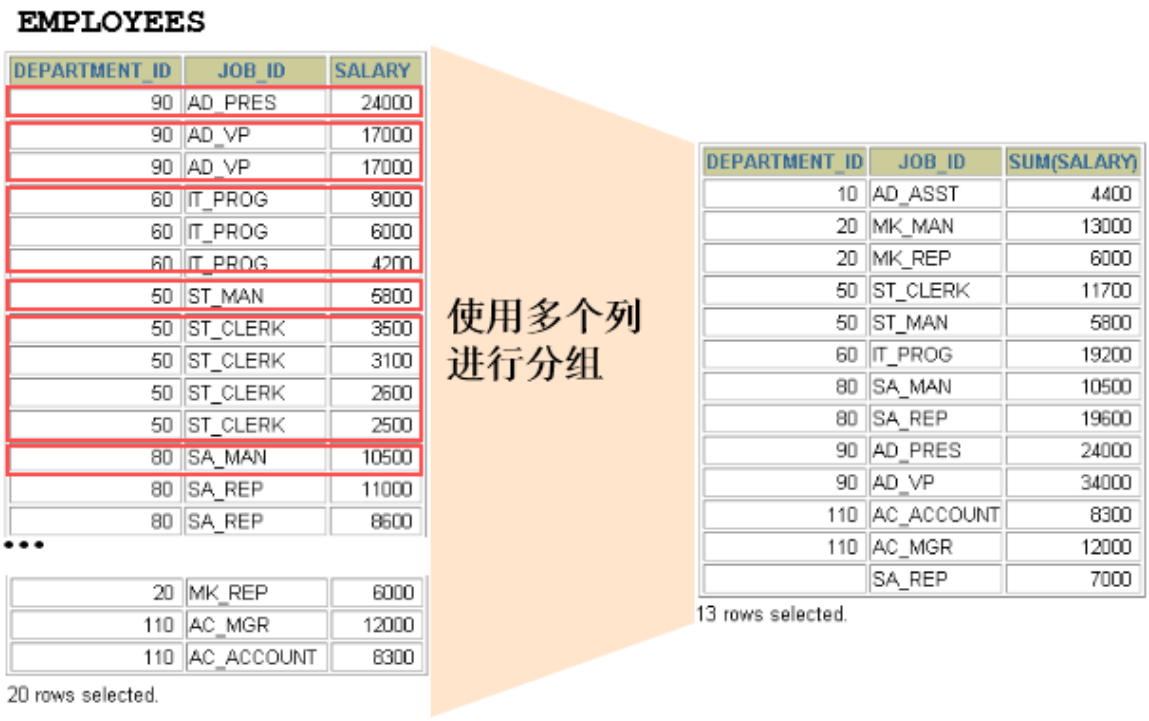
```
SELECT e.`department_id`,AVG(e.`salary`) "平均工资",
MAX(e.`salary`) "最高工资"
FROM employees e
GROUP BY e.`department_id`;
+---------------+--------------+--------------+
| department_id | 平均工资      | 最高工资      |
+---------------+--------------+--------------+
|          NULL |  7000.000000 |      7000.00 |
|            10 |  4400.000000 |      4400.00 |
|            20 |  9500.000000 |     13000.00 |
|            30 |  4150.000000 |     11000.00 |
|            40 |  6500.000000 |      6500.00 |
```

```
|            50 |  3475.555556 |      8200.00 |
|            60 |  5760.000000 |      9000.00 |
|            70 | 10000.000000 |     10000.00 |
|            80 |  8955.882353 |     14000.00 |
|            90 | 19333.333333 |     24000.00 |
|           100 |  8600.000000 |     12000.00 |
|           110 | 10150.000000 |     12000.00 |
+---------------+--------------+--------------+
```

- **查询各个job_id的平均工资**

```
SELECT job_id,AVG(salary)
FROM employees
GROUP BY job_id;
+------------+--------------+
| job_id     | AVG(salary)  |
+------------+--------------+
| AC_ACCOUNT |  8300.000000 |
| AC_MGR     | 12000.000000 |
| AD_ASST    |  4400.000000 |
| AD_PRES    | 24000.000000 |
| AD_VP      | 17000.000000 |
| FI_ACCOUNT |  7920.000000 |
| FI_MGR     | 12000.000000 |
| SH_CLERK   |  3215.000000 |
| ST_CLERK   |  2785.000000 |
| ST_MAN     |  7280.000000 |
+------------+--------------+
```

2.2 使用多个列分组



用两个字段进行分组

可以使用GROUP BY子句将表中的数据分成若干组

```
SELECT column, group_function(column)
FROM table
[WHERE  condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

> **明确：WHERE一定放在FROM后面**

- **查询各个department_id,job_id的平均工资**

```
#方式1：
SELECT department_id,job_id,AVG(salary)
FROM employees
-- 先按部门分组，再按职位分组
GROUP BY  department_id,job_id;
+---------------+------------+--------------+
| department_id | job_id     | AVG(salary)  |
+---------------+------------+--------------+
|          NULL | SA_REP     |  7000.000000 |
|            10 | AD_ASST    |  4400.000000 |
|            20 | MK_MAN     | 13000.000000 |
|            20 | MK_REP     |  6000.000000 |
|           100 | FI_MGR     | 12000.000000 |
|           110 | AC_ACCOUNT |  8300.000000 |
|           110 | AC_MGR     | 12000.000000 |
+---------------+------------+--------------+
```

```
#方式2：
SELECT job_id,department_id,AVG(salary)
FROM employees
-- 先按职位分组，再按部门分组
GROUP BY job_id,department_id;
+------------+---------------+--------------+
| job_id     | department_id | AVG(salary)  |
+------------+---------------+--------------+
| AC_ACCOUNT |           110 |  8300.000000 |
| AC_MGR     |           110 | 12000.000000 |
| AD_ASST    |            10 |  4400.000000 |
+------------+---------------+--------------+
```

注意：

在SELECT列表中所有未包含在组函数中的列都应该包含在 GROUP BY子句中

```
SELECT   department_id,job_id, AVG(salary)
FROM     employees
GROUP BY department_id ;

+---------------+---------+--------------+
| department_id | job_id  | AVG(salary)  |
+---------------+---------+--------------+
|          NULL | SA_REP  |  7000.000000 |
|            10 | AD_ASST |  4400.000000 |
|            20 | MK_MAN  |  9500.000000 |
|            30 | PU_MAN  |  4150.000000 |
|            40 | HR_REP  |  6500.000000 |
|            50 | ST_MAN  |  3475.555556 |
|            60 | IT_PROG |  5760.000000 |
|            70 | PR_REP  | 10000.000000 |
|            80 | SA_MAN  |  8955.882353 |
|            90 | AD_PRES | 19333.333333 |
|           100 | FI_MGR  |  8600.000000 |
|           110 | AC_MGR  | 10150.000000 |
+---------------+---------+--------------+
12 rows in set (0.00 sec)
```

注意：

上面虽然有结果，但是结果是错误的；

上面语句是按部门分组，所以一个部门对应一个平均工资；

但是一个部门对应有多个工种，这里只是随便选择出一个

包含在 GROUP BY 子句中的列不必包含在SELECT 列表中

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id;

+--------------+
| AVG(salary)  |
+--------------+
|  7000.000000 |
|  4400.000000 |
|  9500.000000 |
|  4150.000000 |
|  6500.000000 |
|  3475.555556 |
|  5760.000000 |
| 10000.000000 |
|  8955.882353 |
| 19333.333333 |
|  8600.000000 |
| 10150.000000 |
+--------------+
12 rows in set (0.00 sec)
```

2.3 GROUP BY中使用WITH ROLLUP

使用`WITH ROLLUP`关键字之后，在所有查询出的分组记录之后增加一条记录，该记录计算查询出的所有记录的总和，即统计记录数量。

```
SELECT department_id,AVG(salary)
FROM employees
WHERE department_id > 80
GROUP BY department_id WITH ROLLUP;

+---------------+--------------+
| department_id | AVG(salary)  |
+---------------+--------------+
|            90 | 19333.333333 |
|           100 |  8600.000000 |
|           110 | 10150.000000 |
|          NULL | 11809.090909 |
+---------------+--------------+
```

> 注意：
>
> 当使用ROLLUP时，不能同时使用ORDER BY子句进行结果排序，即ROLLUP和ORDER BY是互相排斥的

```
#说明：当使用ROLLUP时，不能同时使用ORDER BY子句进行结果排序，即ROLLUP和ORDER BY是互相排斥的。
#错误的：
SELECT department_id,AVG(salary) avg_sal
FROM employees
GROUP BY department_id WITH ROLLUP
ORDER BY avg_sal ASC;
-- ERROR 1221 (HY000): Incorrect usage of CUBE/ROLLUP and ORDER BY
```

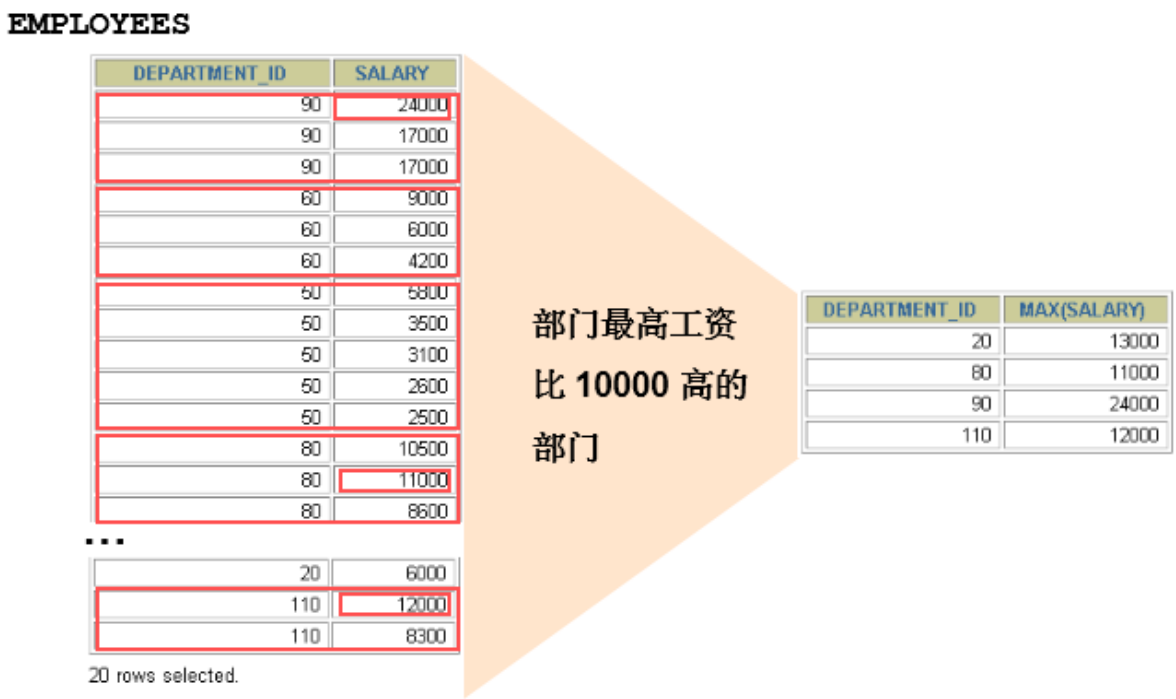
首先注意一个执行顺序：1、FROM 2、WHERE 3、SELECT 4、GROUP BY 5、ORDER BY

所以，分组后会出现一个记录所有记录总和的数据，该数据要参与ORDER BY 显然是不合理的

3. HAVING

3.1 基本使用

用于过滤数据



非法使用聚合函数： 不能在 WHERE 子句中使用聚合函数。如下：

```
#练习：查询各个部门中最高工资比10000高的部门信息
#错误的写法：
SELECT department_id,MAX(salary)
FROM employees
WHERE MAX(salary) > 10000
GROUP BY department_id;
-- ERROR 1111 (HY000): Invalid use of group function
```

```
#错误的写法：
SELECT department_id,MAX(salary)
FROM employees
HAVING MAX(salary) > 10000
GROUP BY department_id;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the right syntax to use near
'GROUP BY department_id' at line 4
```

```
#要求1：如果过滤条件中使用了聚合函数，则必须使用HAVING来替换WHERE。否则，报错。
#要求2：HAVING 必须声明在 GROUP BY 的后面。
#要求3：开发中，我们使用HAVING的前提是SQL中使用了GROUP BY。

#正确的写法：
SELECT department_id,MAX(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary) > 10000;
```

注意：HAVING 一般配合 GROUP BY使用

因为HAVING是用来过滤 每一组是否符合某个要求

当没有使用GROUP BY时，说明只有一组；

只能说，一般不会对单独一组进行过滤

- **查询部门id为10,20,30,40这4个部门中最高工资比10000高的部门信息**

```
#方式1：推荐，执行效率高于方式2.
SELECT department_id,MAX(salary)
FROM employees
WHERE department_id IN (10,20,30,40)
GROUP BY department_id
HAVING MAX(salary) > 10000;
+---------------+-------------+
| department_id | MAX(salary) |
+---------------+-------------+
|            20 |    13000.00 |
|            30 |    11000.00 |
+---------------+-------------+

#方式2：
SELECT department_id,MAX(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary) > 10000 AND department_id IN (10,20,30,40);
+---------------+-------------+
| department_id | MAX(salary) |
+---------------+-------------+
|            20 |    13000.00 |
|            30 |    11000.00 |
```

```
+---------------+-------------+
```

过滤分组：HAVING子句

1. 行已经被分组，HAVING 对组进行过滤
2. 使用了聚合函数
3. 满足HAVING 子句中条件的分组将被显示
4. HAVING 不能单独使用，必须要跟 GROUP BY 一起使用

```
SELECT        column, group_function
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[HAVING       group_condition]
[ORDER BY     column];
```

小结：

1、当过滤条件中有聚合函数时，则此过滤条件**必须**声明在HAVING中
2、当**过滤条件中没有聚合函数**时，则此过滤条件声明在WHERE中或HAVING中都可以。但是，**建议大家声明在WHERE**中，效率高

3.2 WHERE和HAVING的对比

区别1：WHERE 可以直接使用表中的字段作为筛选条件，但不能使用分组中的计算函数作为筛选条件；HAVING 必须要与 GROUP BY 配合使用，可以把分组计算的函数和分组字段作为筛选条件。

这决定了，在需要对数据进行分组统计的时候，HAVING 可以完成 WHERE 不能完成的任务

这是因为，在查询语法结构中，WHERE 在 GROUP BY 之前，所以无法对分组结果进行筛选。

HAVING 在 GROUP BY 之后，可以使用分组字段和分组中的计算函数，对分组的结果集进行筛选，这个功能是 WHERE 无法完成的。

另外，WHERE排除的记录不再包括在分组中

区别2：如果需要通过连接从关联表中获取需要的数据，WHERE 是先筛选后连接，而 HAVING 是先连接后筛选

这一点，就决定了在关联查询中，WHERE 比 HAVING 更高效

因为 WHERE 可以先筛选，用一个筛选后的较小数据集和关联表进行连接，这样占用的资源比较少，执行效率也比较高

HAVING 则需要先把结果集准备好，也就是用未被筛选的数据集进行关联，然后对这个大的数据集进行筛选，这样占用的资源就比较多，执行效率也较低。

小结如下：

	优点	缺点
WHERE	先筛选数据再关联，执行效率高	不能使用分组中的计算函数进行筛选
HAVING	可以使用分组中的计算函数	在最后的结果集中进行筛选，执行效率较低

开发中的选择：

WHERE 和 HAVING 也不是互相排斥的，我们可以在一个查询里面同时使用 WHERE 和 HAVING。包含分组统计函数的条件用 HAVING，普通条件用 WHERE。这样，我们就既利用了 WHERE 条件的高效快速，又发挥了 HAVING 可以使用包含分组统计函数的查询条件的优点。当数据量特别大的时候，运行效率会有很大的差别

4. SELECT的执行过程

4.1 查询的结构

```
#方式1SQL92语法：
SELECT 字段1，字段2，字段3，聚合函数(字段4)，聚合函数(字段5)，聚合函数(字段6)
FROM 表1  ，表2  ，表3.......
WHERE 多表的连接条件
AND 不包含组函数的过滤条件
GROUP BY 字段1，字段2，字段3
HAVING 包含组函数的过滤条件
ORDER BY  ASC、DESC
LIMIT 偏移量，个数

#方式2SQL99语法：
SELECT 字段1，字段2，字段3，聚合函数(字段4)，聚合函数(字段5)，聚合函数(字段6)
FROM 表1  （LEFT、RIGHT）JOIN 表2
ON 多表的连接条件
（LEFT、RIGHT）JOIN 表3
ON 多表的连接条件
WHERE 不包含组函数的过滤条件
AND/OR 不包含组函数的过滤条件
GROUP BY 字段1，字段2，字段3
HAVING 包含组函数的过滤条件
ORDER BY  ASC、DESC
LIMIT  偏移量，个数

#其中：
```

```
#（1）from：从哪些表中筛选
#（2）on：关联多表查询时，去除笛卡尔积
#（3）where：从表中筛选的条件
#（4）group by：分组依据
#（5）having：在统计结果中再次筛选
#（6）order by：排序
#（7）limit：分页
```

4.2 SELECT执行顺序

你需要记住 SELECT 查询时的两个顺序：

1. 关键字的顺序是不能颠倒的：

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ... LIMIT...
```

2.SELECT 语句的执行顺序（在 MySQL 和 Oracle 中，SELECT 执行顺序基本相同）：

```
FROM -> WHERE、ON -> GROUP BY -> HAVING -> SELECT 的字段 -> DISTINCT -> ORDER BY
-> LIMIT
```

1、FROM 两个表进行CROSS JOIN，获取笛卡尔积；

2、通过WHERE或是ON给出多表间的连接条件进行记录过滤

3、判断是不是外连接，如是，则补充主表有记录而不满足多表间连接条件的记录

4、WHERE过滤数据

5、GROUP BY 进行数据分组

6、HAVING进行数据过滤

7、SELECT 选择需要的字段

8、DISTINCT去重

9、ORDER BY 排序

10、LIMIT 分页

比如你写了一个 SQL 语句，那么它的关键字顺序和执行顺序是下面这样的：

```
SELECT DISTINCT player_id, player_name, count(*) as num # 顺序 5
FROM player JOIN team ON player.team_id = team.team_id # 顺序 1
WHERE height > 1.80 # 顺序 2
GROUP BY player.team_id # 顺序 3
HAVING num > 2 # 顺序 4
ORDER BY num DESC # 顺序 6
LIMIT 2 # 顺序 7
```

在 SELECT 语句执行这些步骤的时候，每个步骤都会产生一个`虚拟表`，然后将这个虚拟表传入下一个步骤中作为输入。需要注意的是，这些步骤隐含在 SQL 的执行过程中，对于我们来说是不可见的

4.3 SQL 的执行原理

SELECT 是先执行 FROM 这一步的。在这个阶段，如果是多张表联查，还会经历下面的几个步骤：

1. 首先先通过 CROSS JOIN 求笛卡尔积，相当于得到虚拟表 vt（virtual table）1-1；
2. 通过 ON 进行筛选，在虚拟表 vt1-1 的基础上进行筛选，得到虚拟表 vt1-2；
3. 添加外部行。如果我们使用的是左连接、右链接或者全连接，就会涉及到外部行，也就是在虚拟表 vt1-2 的基础上增加外部行，得到虚拟表 vt1-3

当然如果我们操作的是两张以上的表，还会重复上面的步骤，直到所有表都被处理完为止。这个过程得到是我们的原始数据

当我们拿到了查询数据表的原始数据，也就是最终的虚拟表`vt1`，就可以在此基础上再进行`WHERE 阶段`。在这个阶段中，会根据 vt1 表的结果进行筛选过滤，得到虚拟表`vt2`。

然后进入第三步和第四步，也就是`GROUP 和 HAVING 阶段`。在这个阶段中，实际上是在虚拟表 vt2 的基础上进行分组和分组过滤，得到中间的虚拟表`vt3`和`vt4`。

当我们完成了条件筛选部分之后，就可以筛选表中提取的字段，也就是进入到`SELECT 和 DISTINCT 阶段`。

首先在 SELECT 阶段会提取想要的字段，然后在 DISTINCT 阶段过滤掉重复的行，分别得到中间的虚拟表`vt5-1`和`vt5-2`。

当我们提取了想要的字段数据之后，就可以按照指定的字段进行排序，也就是`ORDER BY 阶段`，得到虚拟表`vt6`。

最后在 vt6 的基础上，取出指定行的记录，也就是`LIMIT 阶段`，得到最终的结果，对应的是虚拟表`vt7`。

当然我们在写 SELECT 语句的时候，不一定存在所有的关键字，相应的阶段就会省略。

同时因为 SQL 是一门类似英语的结构化查询语言，所以我们在写 SELECT 语句的时候，还要注意相应的关键字顺序

所谓底层运行的原理，就是我们刚才讲到的执行顺序

合函数的课后练习

查询公司员工工资的最大值，最小值，平均值，总和

```
SELECT MAX(salary) max_sal ,MIN(salary) mim_sal,AVG(salary)
avg_sal,SUM(salary) sum_sal
FROM employees;
+----------+---------+-------------+-----------+
| max_sal  | mim_sal | avg_sal     | sum_sal   |
+----------+---------+-------------+-----------+
| 24000.00 | 2100.00 | 6461.682243 | 691400.00 |
+----------+---------+-------------+-----------+
-- 没有分组
```

选择具有各个job_id的员工人数

```
SELECT job_id,COUNT(*) number
FROM employees
GROUP BY job_id;

+------------+--------+
| job_id     | number |
+------------+--------+
| AC_ACCOUNT |      1 |
| AC_MGR     |      1 |
| AD_ASST    |      1 |
| SA_REP     |     30 |
| SH_CLERK   |     20 |
| ST_CLERK   |     20 |
| ST_MAN     |      5 |
+------------+--------+
19 rows in set (0.00 sec)
```

查询员工最高工资和最低工资的差距（DIFFERENCE）

```
SELECT MAX(salary) - MIN(salary) "DIFFERENCE"
FROM employees;

+------------+
| DIFFERENCE |
+------------+
|   21900.00 |
+------------+
```

查询各个管理者手下员工的最低工资，去掉最低工资低于6000的管理者，没有管理者的员工不计算在内

```
SELECT manager_id,MIN(salary)
FROM employees
WHERE manager_id IS NOT NULL
GROUP BY manager_id
HAVING MIN(salary) >= 6000;
+------------+-------------+
| manager_id | MIN(salary) |
+------------+-------------+
|        102 |     9000.00 |
|        108 |     6900.00 |
|        145 |     7000.00 |
|        146 |     7000.00 |
|        147 |     6200.00 |
|        148 |     6100.00 |
|        149 |     6200.00 |
|        201 |     6000.00 |
|        205 |     8300.00 |
+------------+-------------+
9 rows in set (0.00 sec)
```

查询各个管理者手下员工的最低工资，其中最低工资不得低于6000，没有管理者的员工不计算在内

```
SELECT manager_id,MIN(salary)
FROM employees
WHERE manager_id IS NOT NULL
AND salary>=6000
GROUP BY manager_id;

+------------+-------------+
| manager_id | MIN(salary) |
+------------+-------------+
|        100 |     6500.00 |
|        101 |     6500.00 |
|        102 |     9000.00 |
|        103 |     6000.00 |
|        108 |     6900.00 |
|        145 |     7000.00 |
|        146 |     7000.00 |
|        147 |     6200.00 |
|        148 |     6100.00 |
|        149 |     6200.00 |
|        201 |     6000.00 |
|        205 |     8300.00 |
+------------+-------------+
12 rows in set (0.00 sec)
```

查询所有部门的名字，location_id，员工数量和平均工资，并按平均工资降序

```
-- 这里不能用count(*)因为只要有记录就会当作有一条记录；而此处用了外连接，这就会造成某些部门没有人，但count出一条记录
-- 所以这里采用count（具体字段）
SELECT d.department_name,d.location_id,COUNT(employee_id),AVG(salary)
FROM departments d LEFT JOIN employees e
ON d.`department_id` = e.`department_id`
GROUP BY department_name,location_id
ORDER BY AVG(salary);
```

```
+----------------------+-------------+--------------------+--------------+
| department_name      | location_id | COUNT(employee_id) | AVG(salary)  |
+----------------------+-------------+--------------------+--------------+
| Treasury             |        1700 |                  0 |         NULL |
| Shareholder Services |        1700 |                  0 |         NULL |
| Payroll              |        1700 |                  0 |         NULL |
| Shipping             |        1500 |                 45 |  3475.555556 |
| Executive            |        1700 |                  3 | 19333.333333 |
+----------------------+-------------+--------------------+--------------+
27 rows in set (0.00 sec)
```

查询每个工种、每个部门的部门名、工种名和最低工资

```
SELECT d.department_name,e.job_id,MIN(salary)
FROM departments d LEFT JOIN employees e
ON d.`department_id` = e.`department_id`
GROUP BY department_name,job_id;
```

```
+----------------------+------------+-------------+
| department_name      | job_id     | MIN(salary) |
+----------------------+------------+-------------+
| Accounting           | AC_ACCOUNT |     8300.00 |
| Accounting           | AC_MGR     |    12000.00 |
| Administration       | AD_ASST    |     4400.00 |
| Benefits             | NULL       |        NULL |
| Construction         | NULL       |        NULL |
| Contracting          | NULL       |        NULL |
+----------------------+------------+-------------+
```