# 窗口函数

考查知识点：

• 如何用窗口函数解决排名问题、Top N问题、前百分之N问题、累计问题、每组内比较问题、连续问题。

## 什么是窗口函数

窗口函数也叫作OLAP（Online Analytical Processing，联机分析处理）函数，可以对数据库中的数据进行复杂分析。

窗口函数的通用语法如下：

|  |
| --- |
| <窗口函数> over(partition by <用于分组的列名>  order by <用于排序的列名>) |

我们看一下这个语法里每部分表示什么。

（1）<窗口函数>的位置可以放两种函数：一种是专用窗口函数，比如用于排名的函数，比如rank()、dense\_rank()、row\_number()；另一种是汇总函数，比如sum()、avg()、count()、max()、min()。

（2）<窗口函数>后面的over关键字括号里的内容有两部分：一个是partition by，表示按某列分组；另一个是order by，表示对分组后的结果按某列排序。

（3）因为窗口函数通常是对where或者group by子句处理后的结果进行操作的，所以窗口函数原则上只能写在select子句中。

SQL的运行顺序:

from 1

where 2

group by 3

having 4

select 5

基本查询

开窗查询

order by 6

limit 7

窗口函数可以解决这几类经典问题：排名问题、Top N问题、前百分之N问题、累计问题、每组内比较问题、连续问题。

这些问题在工作中你会经常遇到，

比如，排名问题，对用户搜索关键字按搜索次数排名、对商品按销售量排名。

再如，领导想让你找出每个部门业绩排名前10的员工进行奖励，这其实就是Top N问题。

再如，要分析复购用户有多少，这类问题属于前百分之N的问题。

再如，公司对各月发放的工资累计求和，医院要经常统计累计患者数，这类问题就是累计问题。

下面我们通过面试题来介绍如何使用窗口函数解决实际问题。

## 排名问题

### 题1: 学生成绩排名

#### 抛真题

现有“成绩表”，需要我们取得每名学生不同课程的成绩排名.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **已知条件 分数表** | | |  | **结果** | | | |
| **student\_name** | **course\_name** | **score** |  | **student\_name** | **course\_name** | **score** | **rn** |
| 小明 | 数学 | 85 |  | 小明 | 物理 | 92 | 1 |
| 小明 | 英语 | 78 |  | 小明 | 数学 | 85 | 2 |
| 小明 | 物理 | 92 |  | 小明 | 英语 | 78 | 3 |
| 小红 | 数学 | 90 |  | 小李 | 数学 | 90 | 1 |
| 小红 | 英语 | 80 |  | 小李 | 英语 | 85 | 2 |
| 小李 | 数学 | 90 |  | 小李 | 物理 | 85 | 3 |
| 小李 | 数学 | 60 |  | 小李 | 数学 | 60 | 4 |
| 小李 | 英语 | 85 |  | 小红 | 数学 | 90 | 1 |
| 小李 | 物理 | 85 |  | 小红 | 英语 | 80 | 2 |

|  |
| --- |
| drop database if exists db\_1; create database db\_1; use db\_1;  CREATE TABLE tb\_score (  student\_name VARCHAR(50),  course\_name VARCHAR(50),  score INT );  INSERT INTO tb\_score (student\_name, course\_name, score) VALUES  ('小明', '数学', 85),  ('小明', '英语', 78),  ('小明', '物理', 92),  ('小红', '数学', 90),  ('小红', '英语', 80),  ('小李', '数学', 90),  ('小李', '数学', 60),  ('小李', '英语', 85),  ('小李', '物理', 85) ;  select *\** from tb\_score; |

#### 分析

#### 实现

|  |
| --- |
| drop database if exists db\_1; create database db\_1; use db\_1;  CREATE TABLE tb\_score (  student\_name VARCHAR(50),  course\_name VARCHAR(50),  score INT );  INSERT INTO tb\_score (student\_name, course\_name, score) VALUES  ('小明', '数学', 85),  ('小明', '英语', 78),  ('小明', '物理', 92),  ('小红', '数学', 90),  ('小红', '英语', 80),  ('小李', '数学', 90),  ('小李', '数学', 60),  ('小李', '英语', 85),  ('小李', '物理', 85) ;  select \* from tb\_score;  # todo 题目要求: 现有“成绩表”，需要我们取得每名学生不同课程的成绩排名. -- 1 每名学生 --> 分组 --> 1 group by 2 开窗排名分组 partition by select  student\_name, course\_name, score,  -- 2 成绩排名  row\_number() over (partition by student\_name order by score desc) rn, # row\_number 排名函数, 不会考虑并列问题  rank() over (partition by student\_name order by score desc) rk, # rank 排名函数 会考虑并列问题, 排名不是连续的  dense\_rank() over (partition by student\_name order by score desc) d\_rk # dense\_rank 排名函数, 会考虑并列问题, 排名是连续的 from tb\_score ;  梅西 大罗 小罗 内马尔 51 1 1 C罗 哈兰德 姆巴佩 50球 4 2 |

#### 小结

### 题2: 去除最大值、最小值后求平均值

#### **抛真题**

“薪水表”中记录了雇员编号、部门编号和薪水。要求查询出每个部门去除最高、最低薪水后的平均薪水。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **已知条件 薪资表** | | |  | **结果** | |
| **employee\_id** | **department\_id** | **salary** |  | **department\_id** | **avg\_salary** |
| 1 | 1 | 50000 |  | 1 | 50000 |
| 2 | 1 | 52000 |  | 2 | 60000 |
| 3 | 1 | 48000 |  |  |  |
| 4 | 1 | 51000 |  |  |  |
| 5 | 1 | 49000 |  |  |  |
| 6 | 2 | 60000 |  |  |  |
| 7 | 2 | 58000 |  |  |  |
| 8 | 2 | 62000 |  |  |  |
| 9 | 2 | 59000 |  |  |  |
| 10 | 2 | 61000 |  |  |  |

|  |
| --- |
| drop database if exists db\_1; create database db\_1; use db\_1;  CREATE TABLE tb\_salary\_table (  employee\_id INT,  department\_id INT,  salary INT );   INSERT INTO tb\_salary\_table (employee\_id, department\_id, salary) VALUES  (1, 1, 50000),  (2, 1, 52000),  (3, 1, 48000),  (4, 1, 51000),  (5, 1, 49000),  (6, 2, 60000),  (7, 2, 58000),  (8, 2, 62000),  (9, 2, 59000),  (10, 2, 61000);  select \* from tb\_salary\_table;  # todo 题目要求: “薪水表”中记录了雇员编号、部门编号和薪水。要求查询出每个部门去除最高、最低薪水后的平均薪水。 |

#### 分析

#### 实现

#### 小结

### **题3: 去除最大值、最小值后求平均值**

#### 抛真题

“成绩表”记录了学号和成绩，计算该6名同学的成绩中去除最高分、最低分后的平均分数。

|  |  |  |  |
| --- | --- | --- | --- |
| 分数表 | |  | 结果 |
| **student\_id** | **score** |  | **avg\_score** |
| 1 | 85 |  | 85.75 |
| 2 | 78 |  |  |
| 3 | 92 |  |  |
| 4 | 90 |  |  |
| 5 | 80 |  |  |
| 6 | 88 |  |  |

|  |
| --- |
| drop database if exists db\_1;  create database db\_1;  use db\_1;  CREATE TABLE tb\_score (  student\_id INT,  score INT  );  INSERT INTO tb\_score (student\_id, score)  VALUES  (1, 85),  (2, 78),  (3, 92),  (4, 90),  (5, 80),  (6, 88);  select \* from tb\_score;  # todo 题目要求: “成绩表”记录了学号和成绩，计算该6名同学的成绩中去除最高分、最低分后的平均分数。 |

#### 分析

#### 实现

#### 小结

## Top N问题

工作中会经常遇到这样的业务问题：

• 如何找到每个类别下用户最喜欢的商品？

• 如何找到每个类别下用户点击最多的5个商品？

这类问题其实就是非常经典的Top N问题，也就是在对数据分组后，取每组里的最大值、最小值，或者每组里最大的N行（Top N）数据。

下面以面试题为例，我们来看如何解决Top N问题，并总结出这类问题的万能模板。

### 题1: 查询前三名的成绩

#### **抛真题**

题目要求: “成绩表”中记录了学生选修的课程号、学生的学号，以及对应课程的成绩。为了对学生成绩进行考核，现需要查询每门课程前三名学生的成绩。

注意：如果出现同样的成绩，则视为同一个名次。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **输入** | | |  | **输出** | | | |
| **course\_id** | **student\_id** | **score** |  | **course\_id** | **student\_id** | **score** | **rn** |
| 1 | 1 | 85 |  | 1 | 3 | 92 | 1 |
| 1 | 2 | 78 |  | 1 | 6 | 92 | 1 |
| 1 | 3 | 92 |  | 1 | 8 | 92 | 1 |
| 1 | 4 | 90 |  | 1 | 4 | 90 | 2 |
| 1 | 5 | 80 |  | 1 | 1 | 85 | 3 |
| 1 | 6 | 92 |  | 1 | 9 | 85 | 3 |
| 1 | 7 | 78 |  | 2 | 3 | 90 | 1 |
| 1 | 8 | 92 |  | 2 | 8 | 90 | 1 |
| 1 | 9 | 85 |  | 2 | 1 | 88 | 2 |
| 2 | 1 | 88 |  | 2 | 6 | 88 | 2 |
| 2 | 2 | 82 |  | 2 | 4 | 85 | 3 |
| 2 | 3 | 90 |  |  |  |  |  |
| 2 | 4 | 85 |  |  |  |  |  |
| 2 | 5 | 78 |  |  |  |  |  |
| 2 | 6 | 88 |  |  |  |  |  |
| 2 | 7 | 82 |  |  |  |  |  |
| 2 | 8 | 90 |  |  |  |  |  |
| 2 | 9 | 82 |  |  |  |  |  |

|  |
| --- |
| drop database if exists db\_1;  create database db\_1;  use db\_1;  CREATE TABLE tb\_score (  course\_id INT,  student\_id INT,  score int  );  INSERT INTO tb\_score (course\_id, student\_id, score)  VALUES  (1, 1, 85),  (1, 2, 78),  (1, 3, 92),  (1, 4, 90),  (1, 5, 80),  (1, 6, 92),  (1, 7, 78),  (1, 8, 92),  (1, 9, 85),  (2, 1, 88),  (2, 2, 82),  (2, 3, 90),  (2, 4, 85),  (2, 5, 78),  (2, 6, 88),  (2, 7, 82),  (2, 8, 90),  (2, 9, 82);  select \* from tb\_score;  # todo 题目要求: “成绩表”中记录了学生选修的课程号、学生的学号，以及对应课程的成绩。为了对学生成绩进行考核，现需要查询每门课程前三名学生的成绩。  # todo 注意：如果出现同样的成绩，则视为同一个名次。 |

#### 分析

#### 实现

|  |
| --- |
|  |

#### 小结

### 题2: 查询排在前两名的工资

#### 抛真题

“雇员表”中是公司雇员的信息，每个雇员有其对应的工号、姓名、工资和部门编号。

现在要查找每个部门工资排在前两名的雇员信息，若雇员工资一样，则并列获取。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **已知条件 雇员表** | | | |  | **结果表** | | | | |
| **emp\_id** | **emp\_name** | **salary** | **department\_id** |  | **emp\_id** | **emp\_name** | **salary** | **department\_id** | **rn** |
| 1 | 小明 | 50000 | 1 |  | 6 | 小刚 | 62000 | 1 | 1 |
| 2 | 小红 | 52000 | 1 |  | 4 | 小张 | 60000 | 1 | 2 |
| 3 | 小李 | 48000 | 1 |  | 10 | 小华 | 52000 | 2 | 1 |
| 4 | 小张 | 60000 | 1 |  | 11 | 小雷 | 52000 | 2 | 1 |
| 5 | 小王 | 58000 | 1 |  | 9 | 小晓 | 49000 | 2 | 2 |
| 6 | 小刚 | 62000 | 1 |  |  |  |  |  |  |
| 7 | 小丽 | 45000 | 2 |  |  |  |  |  |  |
| 8 | 小芳 | 47000 | 2 |  |  |  |  |  |  |
| 9 | 小晓 | 49000 | 2 |  |  |  |  |  |  |
| 10 | 小华 | 52000 | 2 |  |  |  |  |  |  |
| 11 | 小雷 | 52000 | 2 |  |  |  |  |  |  |

|  |
| --- |
| drop database if exists db\_1;  create database db\_1;  use db\_1;  CREATE TABLE tb\_employee (  emp\_id INT,  emp\_name VARCHAR(50),  salary INT,  department\_id INT  );  INSERT INTO tb\_employee (emp\_id, emp\_name, salary, department\_id)  VALUES  (1, '小明', 50000, 1),  (2, '小红', 52000, 1),  (3, '小李', 48000, 1),  (4, '小张', 60000, 1),  (5, '小王', 58000, 1),  (6, '小刚', 62000, 1),  (7, '小丽', 45000, 2),  (8, '小芳', 47000, 2),  (9, '小晓', 49000, 2),  (10, '小华', 52000, 2),  (11, '小雷', 52000, 2);  select \* from tb\_employee;  # todo “雇员表”中是公司雇员的信息，每个雇员有其对应的工号、姓名、工资和部门编号。  # todo 现在要查找每个部门工资排在前两名的雇员信息，若雇员工资一样，则并列获取。 |

#### 分析

#### 实现

#### 小结

## 前百分之N问题

### 题1: 成绩排在前40%的学生信息

#### 抛真题

“成绩表”记录了学号、班级和成绩，现在查询每个班级成绩排在前40%的学生信息。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **已知条件 分数表** | | |  | **结果** | | | |
| **student\_id** | **class\_id** | **score** |  | **student\_id** | **class\_id** | **score** | **p\_rn** |
| 1 | 1 | 85 |  | 3 | 1 | 92 | 0 |
| 2 | 1 | 78 |  | 4 | 1 | 90 | 0.25 |
| 3 | 1 | 92 |  | 8 | 2 | 90 | 0 |
| 4 | 1 | 90 |  | 9 | 2 | 90 | 0 |
| 5 | 1 | 80 |  |  |  |  |  |
| 6 | 2 | 88 |  |  |  |  |  |
| 7 | 2 | 82 |  |  |  |  |  |
| 8 | 2 | 90 |  |  |  |  |  |
| 9 | 2 | 90 |  |  |  |  |  |

|  |
| --- |
| drop database if exists db\_1;  create database db\_1;  use db\_1;  CREATE TABLE tb\_grade (  student\_id INT,  class\_id INT,  score int  );  INSERT INTO tb\_grade (student\_id, class\_id, score)  VALUES  (1, 1, 85),  (2, 1, 78),  (3, 1, 92),  (4, 1, 90),  (5, 1, 80),  (6, 2, 88),  (7, 2, 82),  (8, 2, 90),  (9, 2, 90);  select \* from tb\_grade;  # todo 题目要求: “成绩表”记录了学号、班级和成绩，现在查询每个班级成绩排在前40%的学生信息。 |

#### 分析

#### 实现

#### 小结

### 题2: 用户访问次数

#### 抛真题

“用户访问次数表”含有 用户编号、用户类型、访问次数。

要求在剔除访问次数前20%的用户后得到每类用户的平均访问次数。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 已知条件 用户访问次数表 | | |  | 结果 | |
| **user\_id** | **user\_type** | **access\_count** |  | **user\_type** | **avg\_count** |
| 1 | 普通用户 | 50 |  | 管理员 | 120 |
| 2 | 管理员 | 120 |  | VIP用户 | 90 |
| 3 | VIP用户 | 80 |  | 普通用户 | 55 |
| 4 | 普通用户 | 60 |  |  |  |
| 5 | 管理员 | 150 |  |  |  |
| 6 | VIP用户 | 100 |  |  |  |
| 7 | 普通用户 | 70 |  |  |  |
| 8 | 管理员 | 130 |  |  |  |
| 9 | VIP用户 | 90 |  |  |  |
| 10 | 普通用户 | 40 |  |  |  |

|  |
| --- |
| drop database if exists db\_1;  create database db\_1;  use db\_1;  CREATE TABLE user\_access\_table (  user\_id INT,  user\_type VARCHAR(50),  access\_count INT  );  INSERT INTO user\_access\_table (user\_id, user\_type, access\_count)  VALUES  (1, '普通用户', 50),  (2, '管理员', 120),  (3, 'VIP用户', 80),  (4, '普通用户', 60),  (5, '管理员', 150),  (6, 'VIP用户', 100),  (7, '普通用户', 70),  (8, '管理员', 130),  (9, 'VIP用户', 90),  (10, '普通用户', 40);  select \* from user\_access\_table;  # todo “用户访问次数表”含有 用户编号、用户类型、访问次数。  # todo 要求在剔除访问次数前20%的用户后得到每类用户的平均访问次数。 |

#### 分析

#### 实现

#### 小结