# 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。

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。

1

2

3

4

... ...

## 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 |

|  |
| --- |
| -- 删除表格  DROP TABLE t\_score;  -- 创建表格  CREATE TABLE t\_score (  student\_name VARCHAR2(50),  course\_name VARCHAR2(50),  score NUMBER  );  -- 插入数据  INSERT INTO t\_score (student\_name, course\_name, score) VALUES ('小明', '数学', 85);  INSERT INTO t\_score (student\_name, course\_name, score) VALUES ('小明', '英语', 78);  INSERT INTO t\_score (student\_name, course\_name, score) VALUES ('小明', '物理', 92);  INSERT INTO t\_score (student\_name, course\_name, score) VALUES ('小红', '数学', 90);  INSERT INTO t\_score (student\_name, course\_name, score) VALUES ('小红', '英语', 80);  INSERT INTO t\_score (student\_name, course\_name, score) VALUES ('小李', '数学', 90);  INSERT INTO t\_score (student\_name, course\_name, score) VALUES ('小李', '数学', 60);  INSERT INTO t\_score (student\_name, course\_name, score) VALUES ('小李', '英语', 85);  INSERT INTO t\_score (student\_name, course\_name, score) VALUES ('小李', '物理', 85);  -- 查询数据  SELECT \* FROM t\_score; |

#### 分析

#### 实现

|  |
| --- |
| -- 需求： 现有“成绩表”，需要我们取得每名学生不同课程的成绩排名.  -- 每名 -- 分组 --> 1 group by 2 partition by  -- 排名 -- 1 row\_number rank dense\_rank  select  t.\*,  row\_number() over(partition by student\_name order by score desc) as 排名  from t\_score t  ; |

#### 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 |  |  |  |

|  |
| --- |
| -- 创建表格  CREATE TABLE t\_salary\_table (  employee\_id NUMBER,  department\_id NUMBER,  salary NUMBER  );  -- 插入数据  -- 插入数据  INSERT INTO t\_salary\_table (employee\_id, department\_id, salary) VALUES (1, 1, 50000);  INSERT INTO t\_salary\_table (employee\_id, department\_id, salary) VALUES (2, 1, 52000);  INSERT INTO t\_salary\_table (employee\_id, department\_id, salary) VALUES (3, 1, 48000);  INSERT INTO t\_salary\_table (employee\_id, department\_id, salary) VALUES (4, 1, 51000);  INSERT INTO t\_salary\_table (employee\_id, department\_id, salary) VALUES (5, 1, 49000);  INSERT INTO t\_salary\_table (employee\_id, department\_id, salary) VALUES (6, 2, 60000);  INSERT INTO t\_salary\_table (employee\_id, department\_id, salary) VALUES (7, 2, 58000);  INSERT INTO t\_salary\_table (employee\_id, department\_id, salary) VALUES (8, 2, 62000);  INSERT INTO t\_salary\_table (employee\_id, department\_id, salary) VALUES (9, 2, 59000);  INSERT INTO t\_salary\_table (employee\_id, department\_id, salary) VALUES (10, 2, 61000);  -- 查询数据  SELECT \* FROM t\_salary\_table;  -- 查询每个部门去除最高、最低薪水后的平均薪水 |

#### 分析

#### 实现

|  |
| --- |
| -- 查询每个部门去除最高、最低薪水后的平均薪水  select  t.\*,  row\_number() over(partition by department\_id order by salary asc) rn1,  row\_number() over(partition by department\_id order by salary desc) rn2  from t\_salary\_table t  ;  with t1 as (  select  t.\*,  row\_number() over(partition by department\_id order by salary asc) rn1,  row\_number() over(partition by department\_id order by salary desc) rn2  from t\_salary\_table t  )  select  department\_id,  avg(salary) avg\_salary  from t1  where rn1>1 and rn2>1  group by department\_id  ; |

#### 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 |  |  |

|  |
| --- |
| -- 创建表格  CREATE TABLE t\_score (  student\_id NUMBER,  score NUMBER  );  -- 插入数据  INSERT INTO t\_score (student\_id, score) VALUES (1, 85);  INSERT INTO t\_score (student\_id, score) VALUES (2, 78);  INSERT INTO t\_score (student\_id, score) VALUES (3, 92);  INSERT INTO t\_score (student\_id, score) VALUES (4, 90);  INSERT INTO t\_score (student\_id, score) VALUES (5, 80);  INSERT INTO t\_score (student\_id, score) VALUES (6, 88);  -- 查询数据  SELECT \* FROM t\_score;  -- 查询去除最高分、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 |  |  |  |  |  |

|  |
| --- |
| -- 删除表格  DROP TABLE t\_score;  -- 创建表格  CREATE TABLE t\_score (  course\_id NUMBER,  student\_id NUMBER,  score NUMBER  );  -- 插入数据  INSERT INTO t\_score VALUES (1, 1, 85);  INSERT INTO t\_score VALUES (1, 2, 78);  INSERT INTO t\_score VALUES (1, 3, 92);  INSERT INTO t\_score VALUES (1, 4, 90);  INSERT INTO t\_score VALUES (1, 5, 80);  INSERT INTO t\_score VALUES (1, 6, 92);  INSERT INTO t\_score VALUES (1, 7, 78);  INSERT INTO t\_score VALUES (1, 8, 92);  INSERT INTO t\_score VALUES (1, 9, 85);  INSERT INTO t\_score VALUES (2, 1, 88);  INSERT INTO t\_score VALUES (2, 2, 82);  INSERT INTO t\_score VALUES (2, 3, 90);  INSERT INTO t\_score VALUES (2, 4, 85);  INSERT INTO t\_score VALUES (2, 5, 78);  INSERT INTO t\_score VALUES (2, 6, 88);  INSERT INTO t\_score VALUES (2, 7, 82);  INSERT INTO t\_score VALUES (2, 8, 90);  INSERT INTO t\_score VALUES (2, 9, 82);  COMMIT;  -- 查询数据  SELECT \* FROM t\_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

|  |
| --- |
| -- 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。  -- 每门课程 -- 分组  select  t.\*,  dense\_rank() over(partition by course\_id order by score desc) as rn  from t\_score t  ;  with t1 as (  select  t.\*,  dense\_rank() over(partition by course\_id order by score desc) as rn  from t\_score t  )  select \*  from t1  where rn<=3  ; |

#### 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 |  |  |  |  |  |  |

|  |
| --- |
| -- 删除表格  DROP TABLE t\_employee;  -- 创建表格  CREATE TABLE t\_employee (  emp\_id NUMBER,  emp\_name VARCHAR2(50),  salary NUMBER,  department\_id NUMBER  );  -- 插入数据  INSERT INTO t\_employee (emp\_id, emp\_name, salary, department\_id)  SELECT 1, '小明', 50000, 1 FROM dual UNION ALL  SELECT 2, '小红', 52000, 1 FROM dual UNION ALL  SELECT 3, '小李', 48000, 1 FROM dual UNION ALL  SELECT 4, '小张', 60000, 1 FROM dual UNION ALL  SELECT 5, '小王', 58000, 1 FROM dual UNION ALL  SELECT 6, '小刚', 62000, 1 FROM dual UNION ALL  SELECT 7, '小丽', 45000, 2 FROM dual UNION ALL  SELECT 8, '小芳', 47000, 2 FROM dual UNION ALL  SELECT 9, '小晓', 49000, 2 FROM dual UNION ALL  SELECT 10, '小华', 52000, 2 FROM dual UNION ALL  SELECT 11, '小雷', 52000, 2 FROM dual;  COMMIT;  -- 查询数据  SELECT \* FROM t\_employee; |

#### 分析

#### 实现

#### 小结

## 累计问题

累计问题在日常工作里经常会遇到，图6.10所示的新冠疫情实时大数据报告，包括累计确诊、累计治愈等的人数信息，这些累计数据是怎么分析出来的呢？



其实，使用汇总函数作为窗口函数就可以实现累计分析。比如，汇总函数sum()用在窗口函数中，表示对数据进行累计求和。下面通过面试题来介绍。

### 力扣: 579. 查询员工的累计薪水

#### 抛真题

表：Employee

+-------------+------+

| Column Name | Type |

+-------------+------+

| id | int |

| month | int |

| salary | int |

+-------------+------+

(id, month) 是该表的主键(具有唯一值的列的组合)。

表中的每一行表示 2020 年期间员工一个月的工资。

编写一个解决方案，在一个统一的表中计算出每个员工的 累计工资汇总 。

员工的 累计工资汇总 可以计算如下:

对于该员工工作的每个月，将 该月 和 前两个月 的工资 加 起来。这是他们当月的 3 个月总工资和 。如果员工在前几个月没有为公司工作，那么他们在前几个月的有效工资为 0 。

不要 在摘要中包括员工 最近一个月 的 3 个月总工资和。

不要 包括雇员 没有工作 的任何一个月的 3 个月总工资和。

返回按 id 升序排序 的结果表。如果 id 相等，请按 month 降序排序。

结果格式如下所示。

示例 1

输入：

Employee table:

+----+-------+--------+

| id | month | salary |

+----+-------+--------+

| 1 | 1 | 20 |

| 2 | 1 | 20 |

| 1 | 2 | 30 |

| 2 | 2 | 30 |

| 3 | 2 | 40 |

| 1 | 3 | 40 |

| 3 | 3 | 60 |

| 1 | 4 | 60 |

| 3 | 4 | 70 |

| 1 | 7 | 90 |

| 1 | 8 | 90 |

+----+-------+--------+

输出：

+----+-------+--------+

| id | month | Salary |

+----+-------+--------+

| 1 | 7 | 90 |

| 1 | 4 | 130 |

| 1 | 3 | 90 |

| 1 | 2 | 50 |

| 1 | 1 | 20 |

| 2 | 1 | 20 |

| 3 | 3 | 100 |

| 3 | 2 | 40 |

+----+-------+--------+

解释：

员工 “1” 有 5 条工资记录，不包括最近一个月的 “8”:

- 第 '7' 个月为 90。

- 第 '4' 个月为 60。

- 第 '3' 个月是 40。

- 第 '2' 个月为 30。

- 第 '1' 个月为 20。

因此，该员工的累计工资汇总为:

+----+-------+--------+

| id | month | salary |

+----+-------+--------+

| 1 | 7 | 90 | (90 + 0 + 0)

| 1 | 4 | 130 | (60 + 40 + 30)

| 1 | 3 | 90 | (40 + 30 + 20)

| 1 | 2 | 50 | (30 + 20 + 0)

| 1 | 1 | 20 | (20 + 0 + 0)

+----+-------+--------+

请注意，'7' 月的 3 个月的总和是 90，因为他们没有在 '6' 月或 '5' 月工作。

员工 '2' 只有一个工资记录('1' 月)，不包括最近的 '2' 月。

+----+-------+--------+

| id | month | salary |

+----+-------+--------+

| 2 | 1 | 20 | (20 + 0 + 0)

+----+-------+--------+

员工 '3' 有两个工资记录，不包括最近一个月的 '4' 月:

- 第 '3' 个月为 60 。

- 第 '2' 个月是 40。

因此，该员工的累计工资汇总为:

+----+-------+--------+

| id | month | salary |

+----+-------+--------+

| 3 | 3 | 100 | (60 + 40 + 0)

| 3 | 2 | 40 | (40 + 0 + 0)

+----+-------+--------+

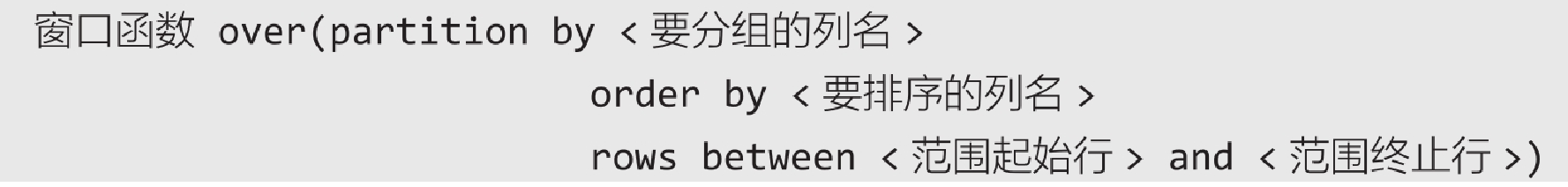
|  |
| --- |
| -- 579. 查询员工的累计薪水  drop table t\_employee;  create table t\_employee (id int, month int, salary int);  truncate table t\_employee;  insert into t\_employee values ('1', '1', '20');  insert into t\_employee values ('2', '1', '20');  insert into t\_employee values ('1', '2', '30');  insert into t\_employee values ('2', '2', '30');  insert into t\_employee values ('3', '2', '40');  insert into t\_employee values ('1', '3', '40');  insert into t\_employee values ('3', '3', '60');  insert into t\_employee values ('1', '4', '60');  insert into t\_employee values ('3', '4', '70');  insert into t\_employee values ('1', '7', '90');  insert into t\_employee values ('1', '8', '90');  commit;  select \* from t\_employee order by id, month desc;  -- 编写一个解决方案，在一个统一的表中计算出每个员工的 累计工资汇总 。  --  -- 员工的 累计工资汇总 可以计算如下:  -- 对于该员工工作的每个月，将 该月 和 前两个月 的工资 加 起来。这是他们当月的 3 个月总工资和 。如果员工在前几个月没有为公司工作，那么他们在前几个月的有效工资为 0 。  -- 不要 在摘要中包括员工 最近一个月 的 3 个月总工资和。  -- 不要 包括雇员 没有工作 的任何一个月的 3 个月总工资和。  -- 返回按 id 升序排序 的结果表。如果 id 相等，请按 month 降序排序。 |

#### 分析

【移动窗口】

移动窗口，顾名思义，“窗口”（也就是操作数据的范围）不是固定的，而是随着设定条件逐行移动的。

在over后面的子句中，使用rows加“范围关键字”可以设置移动窗口，语法如下：



“rows between <范围起始行> and <范围终止行>”用于指定移动窗口的范围，范围包含起始行和终止行。

其中，“范围起始行”和“范围终止行”使用特定关键字表示，常用的特定关键字如下。

• n preceding：当前行的前n行。

• n following：当前行的后n行。

• current row：当前行。

• unbounded preceding：第1行。

• unbounded following：最后1行。

例如：

现有2022年11月前7天的某地区新冠病毒感染患者确诊数据，共7行数据，如表6.28所示。

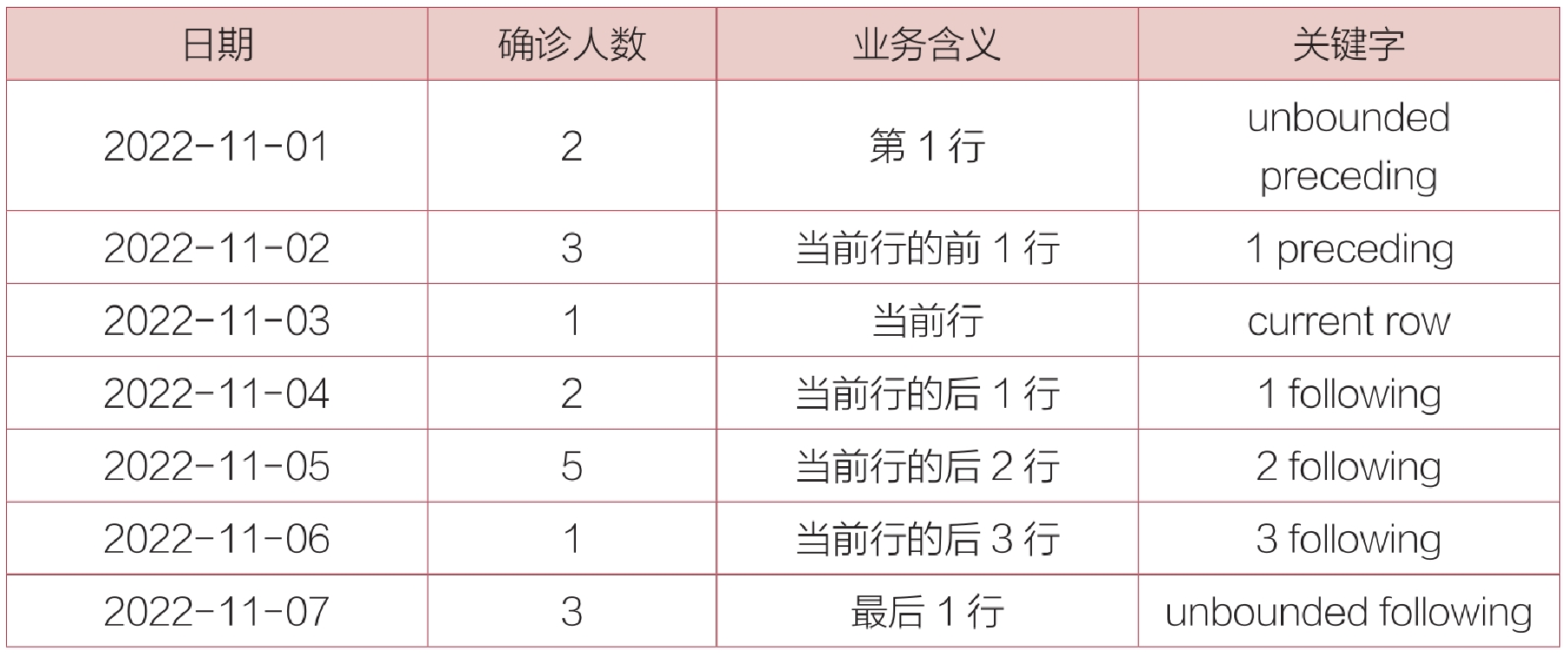
在按“日期”列正向排序的前提下，在以“日期”为“2022-11-03”的这行数据为“当前行”时：

• “日期”为“2022-11-01”的这行数据是“当前行的前2行”，同时它一直是该数据集中的“第1行”。

• “日期”为“2022-11-02”的这行数据是“当前行的前1行”。

• “日期”为“2022-11-04”的这行数据是“当前行的后1行”。

• “日期”为“2022-11-07”的这行数据是“当前行的后4行”，同时它一直是该数据集中的“最后1行”。



当范围设定为“unbounded preceding”至“current row”时，统计的就是“2022-11-01”至“2022-11-03”的累计确诊情况。

回到本面试题，这是计算从第1行起截至当前行的累计求和问题。

同时，是对成绩累计求和，所以，语法中的<窗口函数>写成sum（成绩）.

#### 实现

|  |
| --- |
| select  t.\*,  row\_number() over(partition by id order by month) as rn,  sum(salary) over(partition by id order by month  range between 2 preceding and current row) as sum\_salary  from t\_employee t  ; |

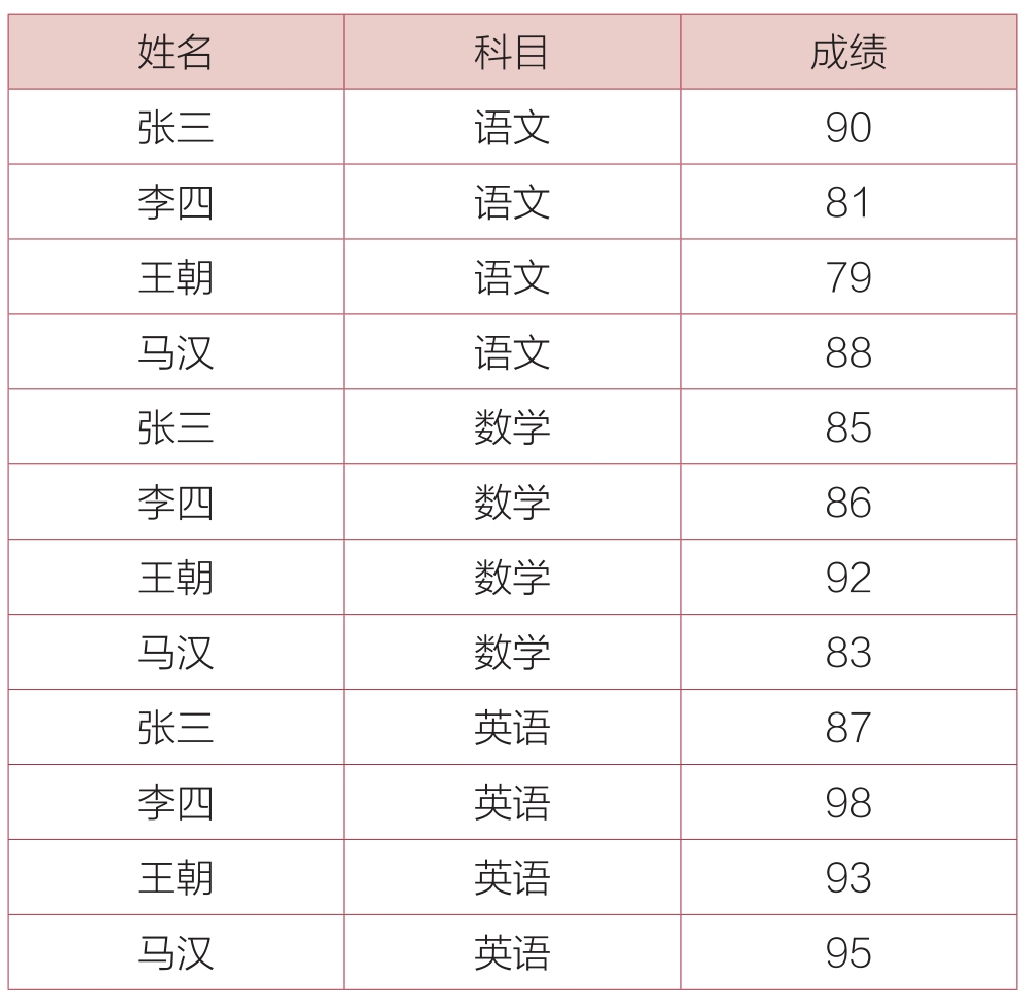
#### 小结

## 每组内比较问题

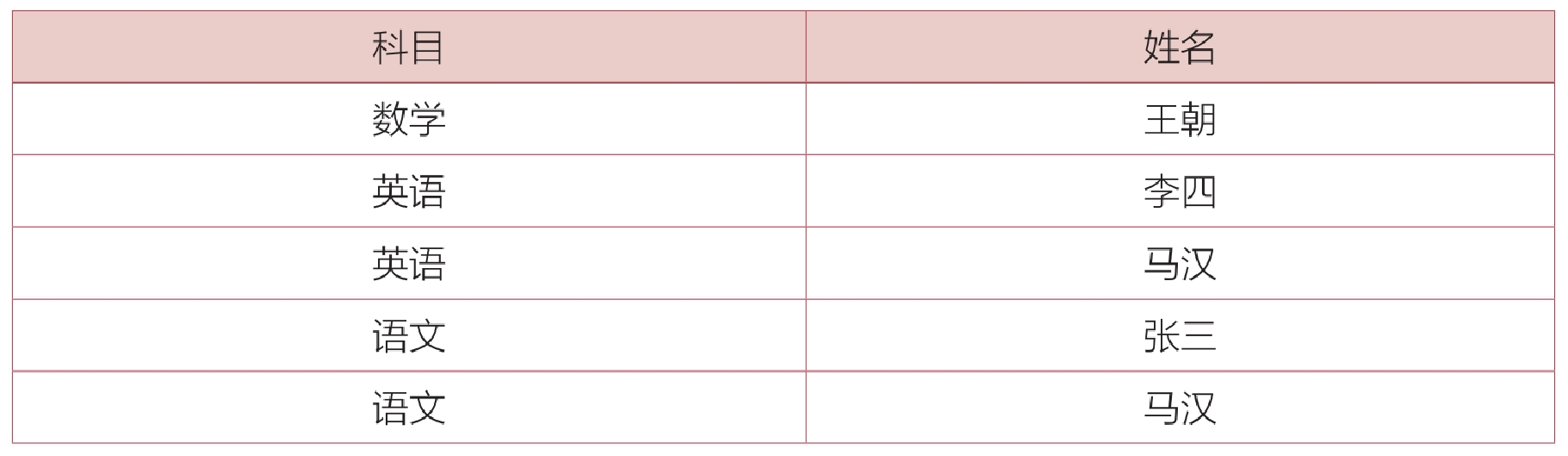
### 题1: 每组大于平均值

#### 抛真题

“成绩表”，记录了每个学生各科的成绩。现在要查找单科成绩高于该科目平均成绩的学生名单。



以下是结果:



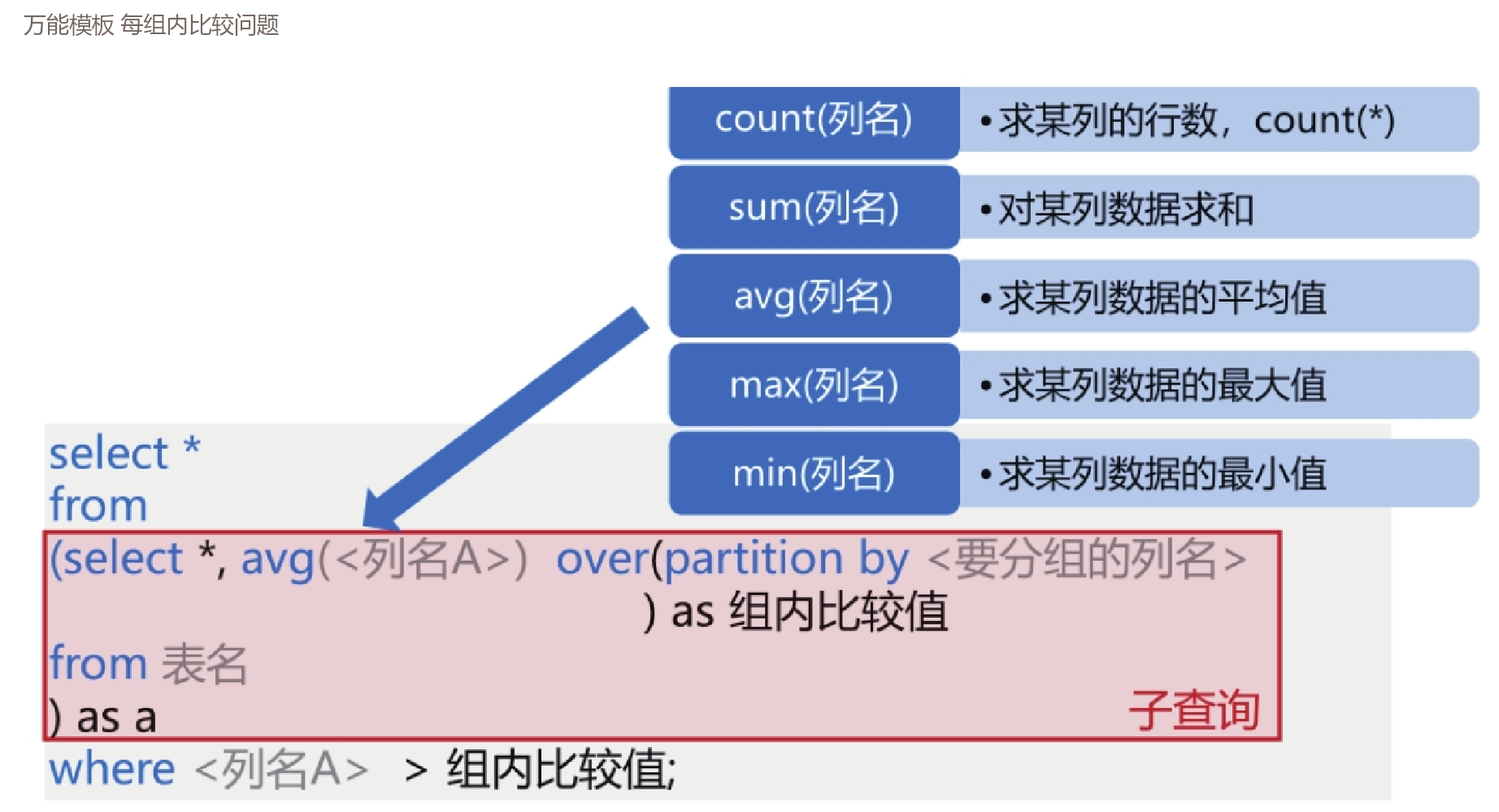
准备工作

|  |
| --- |
| -- 创建表格  CREATE TABLE t\_score (  student\_name VARCHAR2(20),  course\_name VARCHAR2(20),  score NUMBER(3)  );  -- 插入数据  INSERT INTO t\_score VALUES ('张三', '语文', 90);  INSERT INTO t\_score VALUES ('李四', '语文', 81);  INSERT INTO t\_score VALUES ('王朝', '语文', 79);  INSERT INTO t\_score VALUES ('马汉', '语文', 88);  INSERT INTO t\_score VALUES ('张三', '数学', 85);  INSERT INTO t\_score VALUES ('李四', '数学', 86);  INSERT INTO t\_score VALUES ('王朝', '数学', 92);  INSERT INTO t\_score VALUES ('马汉', '数学', 83);  INSERT INTO t\_score VALUES ('张三', '英语', 87);  INSERT INTO t\_score VALUES ('李四', '英语', 98);  INSERT INTO t\_score VALUES ('王朝', '英语', 93);  INSERT INTO t\_score VALUES ('马汉', '英语', 95);  COMMIT;  -- 查询数据  SELECT \* FROM t\_score;  *# todo “成绩表”，记录了每个学生各科的成绩。现在要查找单科成绩高于该科目平均成绩的学生名单。* |

#### 分析

#### 实现

#### 小结

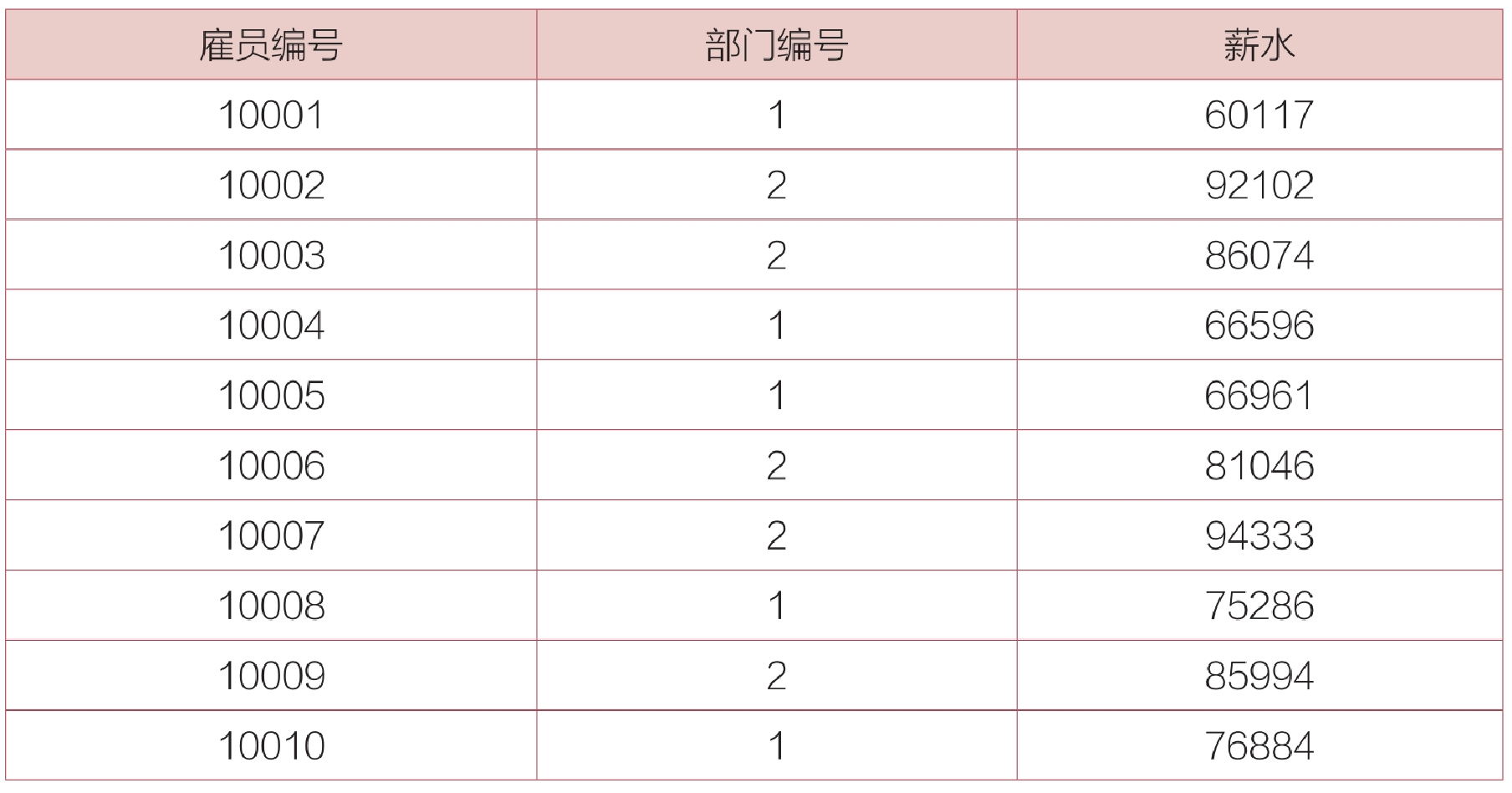


### 题2: 低于平均薪水的雇员

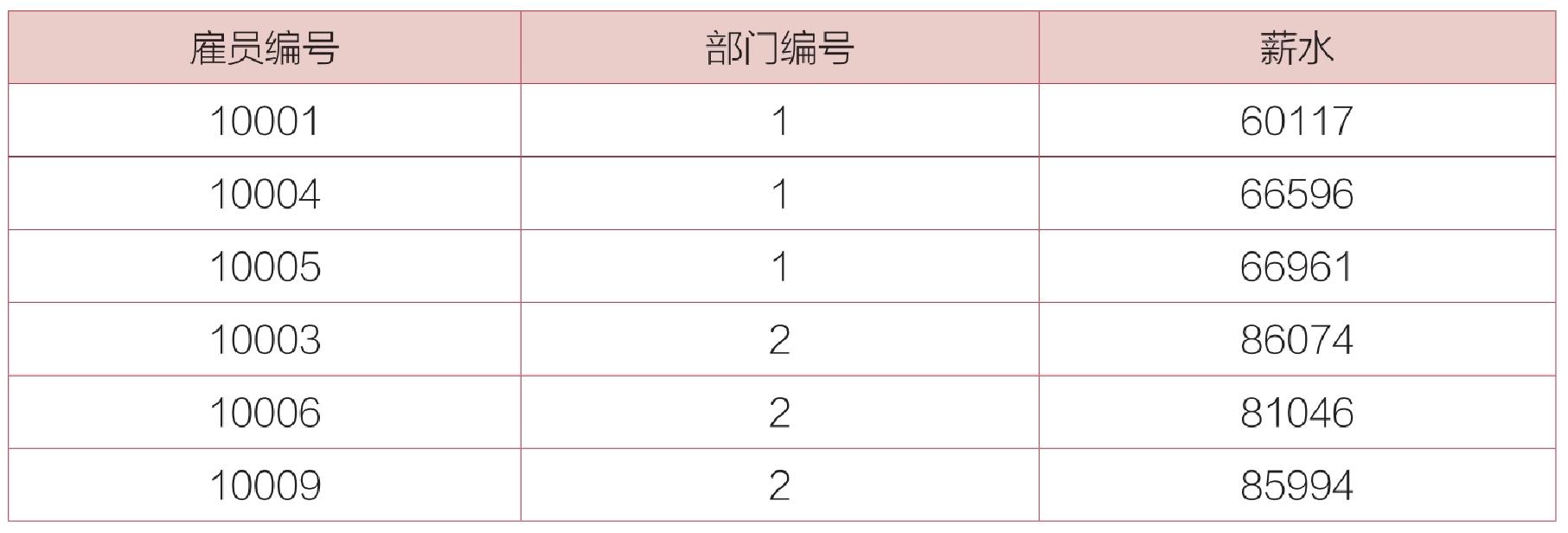
#### 抛真题

“薪水表”，包含雇员编号、部门编号和薪水的信息。

现在公司要找出每个部门低于平均薪水的雇员，然后进行培训来提高雇员工作效率，从而提高雇员薪水。



结果如下:



|  |
| --- |
| -- 创建员工表  CREATE TABLE t\_employee (  employeeID INT,  departmentID INT,  salary INT  );  -- 插入数据  INSERT INTO t\_employee (employeeID, departmentID, salary) VALUES (10001, 1, 60117);  INSERT INTO t\_employee (employeeID, departmentID, salary) VALUES (10002, 2, 92102);  INSERT INTO t\_employee (employeeID, departmentID, salary) VALUES (10003, 2, 86074);  INSERT INTO t\_employee (employeeID, departmentID, salary) VALUES (10004, 1, 66596);  INSERT INTO t\_employee (employeeID, departmentID, salary) VALUES (10005, 1, 66961);  INSERT INTO t\_employee (employeeID, departmentID, salary) VALUES (10006, 2, 81046);  INSERT INTO t\_employee (employeeID, departmentID, salary) VALUES (10007, 2, 94333);  INSERT INTO t\_employee (employeeID, departmentID, salary) VALUES (10008, 1, 75286);  INSERT INTO t\_employee (employeeID, departmentID, salary) VALUES (10009, 2, 85994);  INSERT INTO t\_employee (employeeID, departmentID, salary) VALUES (10010, 1, 76884);  # 现在公司要找出每个部门低于平均薪水的雇员，然后进行培训来提高雇员工作效率，从而提高雇员薪水。 |

#### 分析

#### 实现

#### 小结

## 连续问题

连续问题用偏移窗口函数lead()、lag()来解决。我们通过面试题来看一下如何解决连续问题。

### 题1: 拼多多数据分析面试题：连续3次为球队得分的球员名单

#### 抛真题

两支篮球队进行了激烈的比赛，比分交替上升。比赛结束后，你有一个两队分数的明细表（名称为“分数表”）。表中记录了球队、球员号码、球员姓名、得分分数及得分时间。现在球队要对比赛中表现突出的球员进行奖励。

问题：请你写一个SQL语句，统计出连续3次为球队得分的球员名单。

准备工作

|  |
| --- |
| -- 创建分数表，并为列名增加注释  drop table t\_score;  CREATE TABLE t\_score (  team\_name VARCHAR2(50),  player\_id INT,  player\_name VARCHAR2(50),  score INT,  score\_time TIMESTAMP  );  COMMENT ON COLUMN t\_score.team\_name IS '球队名称';  COMMENT ON COLUMN t\_score.player\_id IS '球员ID';  COMMENT ON COLUMN t\_score.player\_name IS '球员姓名';  COMMENT ON COLUMN t\_score.score IS '得分';  COMMENT ON COLUMN t\_score.score\_time IS '得分时间';  INSERT INTO t\_score VALUES ('洛杉矶湖人队', 23, '勒布朗·詹姆斯', 3, TO\_TIMESTAMP('2023-12-25 10:00:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('洛杉矶湖人队', 23, '勒布朗·詹姆斯', 3, TO\_TIMESTAMP('2023-12-25 10:15:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('洛杉矶湖人队', 23, '勒布朗·詹姆斯', 1, TO\_TIMESTAMP('2023-12-25 10:30:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('洛杉矶湖人队', 3, '安东尼·戴维斯', 2, TO\_TIMESTAMP('2023-12-25 10:32:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('洛杉矶湖人队', 23, '勒布朗·詹姆斯', 3, TO\_TIMESTAMP('2023-12-25 10:45:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('洛杉矶湖人队', 23, '勒布朗·詹姆斯', 3, TO\_TIMESTAMP('2023-12-25 11:00:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('洛杉矶湖人队', 23, '勒布朗·詹姆斯', 2, TO\_TIMESTAMP('2023-12-25 11:15:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('洛杉矶湖人队', 23, '勒布朗·詹姆斯', 2, TO\_TIMESTAMP('2023-12-25 11:30:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('金州勇士队', 30, '斯蒂芬·库里', 1, TO\_TIMESTAMP('2023-12-25 10:10:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('金州勇士队', 30, '斯蒂芬·库里', 1, TO\_TIMESTAMP('2023-12-25 10:25:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('金州勇士队', 30, '斯蒂芬·库里', 1, TO\_TIMESTAMP('2023-12-25 10:40:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('金州勇士队', 11, '克莱·汤普森', 2, TO\_TIMESTAMP('2023-12-25 10:45:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('金州勇士队', 30, '斯蒂芬·库里', 2, TO\_TIMESTAMP('2023-12-25 10:55:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('金州勇士队', 30, '斯蒂芬·库里', 2, TO\_TIMESTAMP('2023-12-25 11:10:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('金州勇士队', 30, '斯蒂芬·库里', 3, TO\_TIMESTAMP('2023-12-25 11:25:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('金州勇士队', 30, '斯蒂芬·库里', 3, TO\_TIMESTAMP('2023-12-25 11:40:00', 'YYYY-MM-DD HH24:MI:SS'));  INSERT INTO t\_score VALUES ('金州勇士队', 30, '斯蒂芬·库里', 3, TO\_TIMESTAMP('2023-12-25 11:55:00', 'YYYY-MM-DD HH24:MI:SS'));  -- 查询数据  SELECT \* FROM t\_score; |

#### 分析

#### 实现

#### 小结

### 题2: 力扣603. 连续空余座位抛真题

#### 抛真题

表: Cinema

+-------------+------+

| Column Name | Type |

+-------------+------+

| seat\_id | int |

| free | bool |

+-------------+------+

Seat\_id 是该表的自动递增主键列。

在 PostgreSQL 中，free 存储为整数。请使用 ::boolean 将其转换为布尔格式。

该表的每一行表示第 i 个座位是否空闲。1 表示空闲，0 表示被占用。

查找电影院所有连续可用的座位。

返回按 seat\_id 升序排序 的结果表。

测试用例的生成使得两个以上的座位连续可用。

结果表格式如下所示。

示例 1:

输入:

Cinema 表:

+---------+------+

| seat\_id | free |

+---------+------+

| 1 | 1 |

| 2 | 0 |

| 3 | 1 |

| 4 | 1 |

| 5 | 1 |

+---------+------+

输出:

+---------+

| seat\_id |

+---------+

| 3 |

| 4 |

| 5 |

+---------+

|  |
| --- |
| Create table Cinema (seat\_id number primary key, free varchar(2));  insert into Cinema (seat\_id, free) values ('1', '1');  insert into Cinema (seat\_id, free) values ('2', '0');  insert into Cinema (seat\_id, free) values ('3', '1');  insert into Cinema (seat\_id, free) values ('4', '1');  insert into Cinema (seat\_id, free) values ('5', '1');  select \* from cinema;  -- todo 查找电影院所有连续可用的座位。  -- todo 返回按 seat\_id 升序排序 的结果表。  -- todo 测试用例的生成使得两个以上的座位连续可用。 |

#### 分析

#### 实现

#### 小结

### 题3: 力扣之 180. 连续出现的数字

#### 抛真题

表：Logs

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| id | int |

| num | varchar |

+-------------+---------+

在 SQL 中，id 是该表的主键。

id 是一个自增列。

找出所有至少连续出现三次的数字。

返回的结果表中的数据可以按 任意顺序 排列。

结果格式如下面的例子所示：

示例 1:

输入：

Logs 表：

+----+-----+

| id | num |

+----+-----+

| 1 | 1 |

| 2 | 1 |

| 3 | 1 |

| 4 | 2 |

| 5 | 1 |

| 6 | 2 |

| 7 | 2 |

+----+-----+

输出：

Result 表：

+-----------------+

| ConsecutiveNums |

+-----------------+

| 1 |

+-----------------+

解释：1 是唯一连续出现至少三次的数字。

|  |
| --- |
| Create table Logs (id int, num int); Truncate table Logs; insert into Logs (id, num) values ('1', '1'); insert into Logs (id, num) values ('2', '1'); insert into Logs (id, num) values ('3', '1'); insert into Logs (id, num) values ('4', '2'); insert into Logs (id, num) values ('5', '1'); insert into Logs (id, num) values ('6', '2'); insert into Logs (id, num) values ('7', '2');  select *\** from logs;  *# todo 需求: 找出所有至少连续出现三次的数字。 # todo 返回的结果表中的数据可以按 任意顺序 排列。* |

#### 分析

#### 实现

#### 小结