Day03-MySQL基础

今日课程学习目标

```
常握 SQL查询时使用AS关键字起别名
常握 SQL子查询的使用
掌握 窗口函数的基础用法
掌握 PARTITION BY分区操作
掌握 排序函数的使用(产生排名)
掌握 自定义window frame操作
```

今日课程内容大纲

```
# 1. 子查询【重点】
    AS 起别名
    子查询操作
# 2. 窗口函数【重点】
    窗口函数简介
    窗口函数基础用法：OVER关键字
    PARTITION BY分区
    排序函数：产生排名
    自定义window frame

完整语法：
<window function> OVER(
    PARTITION BY 字段, ... # 分区
    ORDER BY 字段, ... # 对每个分区内的数据进行排序
    ROWS|RANGE BETWEEN 上限 AND 下限 # 设置每行关联的分区window frame范围
)

聚合函数：SUM、MAX、MIN、AVG、COUNT
排序函数：RANK、DENSE_RANK、ROW_NUMBER
```

知识点1：SQL查询时使用AS关键字起别名【掌握】

在SQL查询时，可以使用 AS 给表或者字段起别名

1）给查询字段起别名

```sql
-- 示例1：查询每个分类的商品数量
SELECT
    category_id,
    -- COUNT(*) AS product_count
    COUNT(*) product_count -- 起别名时，AS关键字可以省略
FROM products
GROUP BY category_id;

SELECT
    category_id,
    COUNT(*) `desc` -- 注意：别名为关键字时，别名两边要加``，否则报错
FROM products
GROUP BY category_id;
```

2）给表起别名

```sql
-- 示例2：查询每个分类名称所对应的商品数量(没有商品的分类的也要显示)
SELECT
    c.cname,
    COUNT(*) product_count
FROM category c -- category表的别名叫c
LEFT JOIN products p -- products表的别名叫p
ON c.cid = p.category_id
GROUP BY c.cid, c.cname;
```

知识点2：SQL子查询操作【掌握】

> 在一个 SELECT 语句中，嵌入了另外一个 SELECT 语句，那么被嵌入的 SELECT 语句称之为子查询语句，外部那个SELECT 语句则称为主查询。

主查询和子查询的关系：

1）子查询是嵌入到主查询中

2）**子查询是辅助主查询的，要么充当条件，要么充当数据源，要么充当查询字段**

3）子查询是可以独立存在的语句，是一条完整的 SELECT 语句

```sql
-- 示例1：查询当前商品大于平均价格的商品
-- ① 查询商品的平均价格
SELECT AVG(price) FROM products;

-- ② 查询所有商品
SELECT * FROM products;

-- ③ 将第①步的结果作为第②步的查询条件
SELECT
    *
FROM products
WHERE price > (SELECT AVG(price) FROM products);

-- 示例4：查询不同类型商品的平均价格
-- 查询结果字段：
--  category_id(分类id)、cname(分类名称)、avg(分类商品平均价格)

SELECT
    category_id,
    cname,
```

```
    avg
FROM (
    SELECT
        category_id,
        AVG(price) AS `avg`
    FROM products
    GROUP BY category_id
) a -- 子查询作用数据源时，必须起别名
JOIN category b
ON a.category_id = b.cid;

-- 示例5：针对 students 表的数据，计算每个同学的Score分数和整体平均分数的差值
SELECT
    *,
    (SELECT AVG(Score) FROM students) AS `avg`,
    Score - (SELECT AVG(Score) FROM students) AS `difference`
FROM students;
```

知识点3：SQL进阶-窗口函数简介【了解】

窗口函数是 MySQL8.0 以后加入的功能，之前需要通过定义临时变量和大量的子查询才能完成的工作，使用窗口函数实现起来更加简洁高效。**同时窗口函数也是面试是的高频考点。**

```
-- 示例1：针对 students 表的数据，计算每个同学的Score分数和整体平均分数的差值
SELECT
    *,
    AVG(Score) OVER() AS `avg`,
    Score - AVG(Score) OVER() AS `difference`
FROM students;
```

窗口函数的优点：

1）**简单**

- 窗口函数更易于使用。在上面的示例中，与使用聚合函数然后合并结果相比，使用窗口函数的 SQL 语句更加简单。

2）**快速**

- 这一点与上一点相关，使用窗口函数比使用替代方法要快得多。当你处理成百上千个千兆字节的数据时，这非常有用。

3）**多功能性**

- 最重要的是，窗口函数具有多种功能，比如：添加移动平均线、添加行号和滞后数据等等。

知识点4：窗口函数-基本用法【掌握】

OVER()关键字：

```
# 基础语法
<window function> OVER(...)
```

- `<window function>` 表示使用的窗口函数，窗口函数可以使用之前已经学过的聚合函数，比如 `COUNT()`、`SUM()`、`AVG()` 等，也可以是其他函数，比如 ranking 排序函数等，后面的课程中会介绍
- `OVER(...)` 的作用就是设置每行数据关联的窗口数据范围，`OVER()` 时，每行关联的数据范围都是整张表的数据。

SQL举例：

```sql
SELECT
    ID,
    Name,
    Gender,
    Score,
    -- OVER()：表示每行关联的窗口数据范围都是整张表的数据
    -- AVG(Score)：表示处理每行数据时，应用 AVG 对每行关联的窗口数据中的 Score 求平均
    AVG(Score) OVER() AS `AVG_Score`
FROM students;
```

典型应用场景：

1）场景1：**计算每个值和整体平均值的差值**

```sql
-- 需求：计算每个学生的 Score 分数和所有学生整体平均分的差值。
SELECT
    ID,
    Name,
    Gender,
    Score,
    AVG(Score) OVER() AS `AVG_Score`,
    Score - AVG(Score) OVER() AS `difference`
FROM students;
```

2）场景2：**计算每个值占整体之和的占比**

```sql
-- 需求：计算每个学生的Score分数占所有学生分数之和的百分比
SELECT
    ID,
    Name,
    Gender,
    Score,
    SUM(Score) OVER() AS `sum`,
    -- 计算百分比：要把 Score / SUM(Score) OVER() 的结果再乘 100
    Score / SUM(Score) OVER() * 100 AS `ratio`
FROM students;
```

知识点5：窗口函数-PARTITION BY分区【掌握】

思考题：

```
1. 如何计算每个学生的 Score 分数和同性别学生平均分的差值？
```

ID	Name	Gender	Score	Avg	difference
2	linda	Female	81.00	82.000000	-1.000000
3	lucy	Female	83.00	82.000000	1.000000
1	smart	Male	90.00	92.000000	-2.000000
4	david	Male	94.00	92.000000	2.000000

PARTITION BY分区：

```sql
# 基础语法
<window function> OVER(PARTITION BY 列名, ...)
```

- `PARTITION BY 列名, ...`的作用是按照指定的列对整张表的数据进行分区
- 分区之后，在处理每行数据时，`<window function>`是作用在该行数据关联的分区上，不再是整张表上

SQL举例：

```sql
SELECT
    ID,
    Name,
    Gender,
    Score,
    -- PARTITION BY Gender：按照性别对整张表的数据进行分区，此处会分成2个区
    -- AVG(Score)：处理每行数据时，应用 AVG 对该行关联分区数据中的 Score 求平均
    AVG(Score) OVER(PARTITION BY Gender) AS `Avg`
FROM students;
```

应用示例：

```sql
-- 需求：计算每个学生的 Score 分数和同性别学生平均分的差值
SELECT
    ID,
    Name,
    Gender,
    Score,
    -- PARTITION BY Gender：按照性别对整张表的数据进行分区，此处会分成2个区
    -- AVG(Score)：处理每行数据时，应用 AVG 对该行关联分区数据中的 Score 求平均
    AVG(Score) OVER(PARTITION BY Gender) AS `Avg`,
    Score - AVG(Score) OVER(PARTITION BY Gender) AS `difference`
FROM students;


-- 需求：计算每人各科分数与对应科目最高分的占比
SELECT
    name,
    course,
    score,
    -- 处理每行数据时，计算相同科目成绩的最高分
    MAX(score) OVER(PARTITION BY course) AS `max`,
    score / MAX(score) OVER(PARTITION BY course) AS `ratio`
FROM tb_score;
```

知识点6：窗口函数-排序函数使用【掌握】

思考题：

```
1. 如何将 tb_score 中的数据，按照分数从高到低产生一列排名序号？
```

name	course	score	`rank`
王五	数学	100.00	1
李四	数学	90.00	2
张三	语文	81.00	3
王五	语文	81.00	3
李四	语文	76.00	5
张三	数学	75.00	6

排序函数：

```sql
# 基础语法
<ranking function> OVER (ORDER BY 列名, ...)
```

- OVER() 中可以指定 ORDER BY 按照指定列对每一行关联的分区数据进行排序，然后使用排序函数对分区内的每行数据产生一个排名序号

SQL举例：

```sql
SELECT
    name,
    course,
    score,
    -- 此处 OVER() 中没有 PARTITION BY，所以整张表就是一个分区
    -- ORDER BY score DESC：按照 score 对每个分区内的数据降序排序
    -- RANK() 窗口函数的作用是对每个分区内的每一行产生一个排名序号
    RANK() OVER(ORDER BY score DESC) as `rank`
FROM tb_score;
```

> 注意：RANK()产生的排名序号可能会不连续(**当有并列情况时**)

不同的排序函数：

- `RANK()`：产生的排名序号，有并列的情况出现时序号不连续
- `DENSE_RANK()`：产生的排序序号是连续的，有并列的情况出现时序号会重复
- `ROW_NUMBER()`：返回连续唯一的行号，排名序号不会重复

SQL举例：

```sql
SELECT
    name,
    course,
    score,
    -- 可能重复不连续
    RANK() OVER(ORDER BY score DESC) as `rank`,
    -- 一定连续，可能重复
    DENSE_RANK() OVER(ORDER BY score DESC) as `dense_rank`,
    -- 一定连续，且不重复
    ROW_NUMBER() OVER(ORDER BY score DESC) as `row_number`
FROM tb_score;
```

name	course	score	`rank`	`dense_rank`	`row_number`
王五	数学	100.00	1	1	1
李四	数学	90.00	2	2	2
张三	语文	81.00	3	3	3
王五	语文	81.00	3	3	4
李四	语文	76.00	5	4	5
张三	数学	75.00	6	5	6

PARTITION BY和排序函数配合使用：

```sql
-- 需求：按照不同科目，对学生的分数从高到低进行排名(要求：连续可重复)
SELECT
    name,
    course,
    score,
    -- 对每个分区内的每一行产生排名序号
    DENSE_RANK() OVER(
        -- 将整张表的数据按照科目进行分区
        PARTITION BY course
        -- 对每个分区内的数据按照score降序排列
        ORDER BY score DESC
    ) as `dense_rank`
FROM tb_score;
```

name	course	score	`dense_rank`
王五	数学	100.00	1
李四	数学	90.00	2
张三	数学	75.00	3
张三	语文	81.00	1
王五	语文	81.00	1
李四	语文	76.00	2

排序函数典型应用：

1）场景：**获取指定排名的数据**

```sql
-- 需求：获取每个科目，排名第二的学生信息
SELECT
    name,
    course,
    score
FROM (
    SELECT
        name,
        course,
        score,
        DENSE_RANK() OVER(
            PARTITION BY course
            ORDER BY score DESC
        ) as `dense_rank`
    FROM tb_score
) s
WHERE `dense_rank` = 2;
```

CTE公用表表达式：

CTE(公用表表达式)：Common Table Expresssion，类似于子查询，相当于一张临时表，可以在 CTE 结果的基础上，进行进一步的查询操作。

```sql
WITH some_name AS (
  --- your CTE ---
)
SELECT
  ...
FROM some_name
```

- 需要给CTE起一个名字（上面的例子中使用了`some_name`），具体的查询语句写在括号中

- 在括号后面，就可以通过 `SELECT` 将CTE的结果当作一张表来使用
- 将CTE称为“内部查询”，其后的部分称为“外部查询”
- 需要先定义CTE，即在外部查询的 `SELECT` 之前定义CTE

```sql
-- 需求：获取每个科目，排名第二的学生信息
WITH ranking AS (
    SELECT
        name,
        course,
        score,
        DENSE_RANK() OVER(
            PARTITION BY course
            ORDER BY score DESC
        ) as `dense_rank`
    FROM tb_score
)
SELECT
    name,
    course,
    score
FROM ranking
WHERE `dense_rank` = 2;
```

知识点7：窗口函数-自定义 window frame【掌握】

思考题：

现有一张某年度的月销量信息表 tb_sales，数据如下：

month	sales
1	10
2	23
3	14
4	5
5	32
6	22
7	52
8	12
9	19
10	36
11	33
12	69

如何计算截止到每个月的累计销量？1月：1月销量，2月：1月销量+2月销量，3月：1月销量+2月销量+3月销量，依次类推

分区数据范围和window frame范围：

在使用窗口函数处理表中的每行数据时，每行数据关联的数据有两种：

1）**每行数据关联的分区数据**

- OVER()中什么都不写时，整张表默认是一个分区
- OVER(PARTITION BY 列名, ...)：整张表按照指定的列被进行了分区

2）**每行数据关联的分区中的window frame数据**

- 每行关联的分区window frame数据范围 <= 每行关联的分区数据范围

目前我们所学的窗口函数中，有些窗口函数作用在分区上，有些函数作用在window frame上：

- **聚合函数(SUM、AVG、COUNT、MAX、MIN)作用于每行关联的分区window frame数据上**
- **排序函数(RANK、DENSE_RANK、ROW_NUMBER)作用于每行关联的分区数据上**

自定义 window frame 范围：

> 自定义 window frames 有两种方式： `ROWS` 和 `RANGE`

```
# 基本语法
<window function> OVER (
  PARTITION BY 列名, ...
  ORDER BY 列名, ...
  [ROWS|RANGE] BETWEEN 上限 AND 下限
)
```

- `PARTITION BY 列名, ...`：按照指定的列，对整张表的数据进行分区
- `ORDER BY 列名, ...`：按照指定的列，对每个分区内的数据进行排序
- `[ROWS|RANGE] BETWEEN 上限 AND 下限`：在分区数据排序的基础上，设置每行关联的分区window frame范围

上限和下限的设置：

- `UNBOUNDED PRECEDING`：对上限无限制
- `PRECEDING`：当前行之前的 n 行 （n 表示具体数字如：`5 PRECEDING`）
- `CURRENT ROW`：仅当前行
- `FOLLOWING`：当前行之后的 n 行 （n 表示具体数字如：`5 FOLLOWING`）
- `UNBOUNDED FOLLOWING`：对下限无限制
- 注意：**上限需要在下限之前**，比如：`ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING` 是错误的

```
-- 需求：计算截止到每个月的累计销量。1月：1月销量，2月：1月销量+2月销量，3月：1月销量+2月销量+3月销量，依次类推
SELECT
    month,
    sales,
    SUM(sales) OVER(
        # 按照 month 对每个分区(注：此处就一个分区->整张表)数据进行排序
        ORDER BY month
        # 指定每行关联分区的 window frame 范围
        # UNBOUNDED PRECEDING：上限不限制
        # CURRENT ROW：当前行
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS `running_total`
FROM tb_sales;
```

window frame定义的简略写法：

自定义 window frame 的边界时，如果使用了 `CURRENT ROW` 作为上边界或者下边界，可以使用如下简略写法：

- `ROWS UNBOUNDED PRECEDING` 等价于 `BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`
- `ROWS n PRECEDING` 等价于 `BETWEEN n PRECEDING AND CURRENT ROW`

- `ROWS CURRENT ROW` 等价于 `BETWEEN CURRENT ROW AND CURRENT ROW`
- 注意，简略写法不适合 `FOLLOWING` 的情况

ROWS和RANGE的区别：

ROWS和RANGE关键字，都可以用来自定义 windowframe 范围：

```
ROWS BETWEEN 上限 AND 下限
RANGE BETWEEN 上限 AND 下限
```

但两者区别如下：

- ROWS是根据分区数据排序之后，每一行的 row_number 确定每行关联的 window frame 范围的

```
CURRENT ROW：仅代表当前行

# 假设某一行数据的 row_number 为5，ROWS自定义window frame如下：
ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING
# 则这一行关联的window frame是：5-2 <= row_number <= 5+2 的数据
```

- RANGE是根据分区数据排序之后，每一行的 dense_rank 值确实每行关联的 window frame 范围的

```
CURRENT ROW：代表和当前行排序列的值相同的所有行

# 假设某一行排序列的值为5，RNAGE自定义window frame如下：
RANGE BETWEEN 2 PRECEDING AND 2 FOLLOWING
# 则这一行关联的window frame是：5-2 <= 排序列的值 <= 5+2 的数据
```

默认的window frame：

在 OVER 中只要添加了 ORDER BY，在没有写ROWS或RANGE的情况下，会有一个默认的 window frame范围：

- RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

```
-- 需求：计算截止到每个月的累计销量。1月：1月销量，2月：1月销量+2月销量，3月：1月销量+2月销量+3月销量，依次类推
SELECT
    month,
    sales,
    DENSE_RANK() OVER(
        ORDER BY month
    ) AS `dense_rank`,
    SUM(sales) OVER(
        # 按照 month 对每个分区(注：此处就一个分区->整张表)数据进行排序
        ORDER BY month
        # OVER 中添加了 ORDER BY 之后，默认的 window frame 范围
        # RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS `running_total`
FROM tb_sales;
```

PARTITION BY和自定义window frame：

```
-- 需求：利用tb_revenue表计算每个商店截止到每个月的累计销售额。1月：1月销量，2月：1月销量+2月销量，3月：1月销量+2月销量+3月销量，依次类推

SELECT
    store_id,
    month,
    revenue,
```

```
    SUM(revenue) OVER(
        # 按照 store_id 对整张表的数据进行分区
        PARTITION BY store_id
        # 按照 month 对每个分区内的数据排序
        ORDER BY month
        # 设置 每行关联的分区 window frame 范围
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS `sum`
FROM tb_revenue;
```