MySQL学习笔记（Day013：作业讲解一/Rank/视图/UNION/触发器）

MySQL学习

一. 作业讲解

- **查询employees表中非基层用户的最近详细信息**

> 老师的讲解的版本中存在问题，重新作为作业

- **统计dbt3库下orders每周每个客户的订单数量**

1. 思路
   - 找到订单中最小周(week)之前的一周的周一，这里进行了简化，使用了 `1970-01-05` 作为周一标记，作为起始(`start`)

```
[root@MyServer ~]> cal 1 1970
    January 1970
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10  # 1970-01-05 刚好是周一，用1月12号，19号，26号等也是可以的
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

   - 在起始条件周一（ `start`)的基础上 `增加6天` 时间，就是周日，即为一周的结束标记(`end`)
   - 通过对 `start` （周一）、 `end` （周日）以及 `o_custkey` 进行 `分组` ，使用 `count(o_orderkey)` 得到对应的数量

2. SQL语句

```
-- 最终结果
select o_custkey,
ADDDATE('1970-01-05', INTERVAL FLOOR(DATEDIFF(o_orderdate, '1970-01-05')/7)*7 Day) as start,
ADDDATE('1970-01-05', INTERVAL FLOOR(DATEDIFF(o_orderdate, '1970-01-05')/7)*7 + 6 Day) as end,
count(o_orderkey) as total
from dbt3.orders group by o_custkey, start, end;

-- DATEDIFF(o_orderdate, '1970-01-05')
mysql> select datediff('1971-01-01', '1970-01-05');  -- 随意取一个1971-01-01，作为演示
+---------------------------------------+
| datediff('1971-01-01', '1970-01-05') |
+---------------------------------------+
|                                   361 |  -- 1971-01-01 减去 1970-01-5 为361天
+---------------------------------------+
1 row in set (0.00 sec)

mysql> select datediff('1971-01-01', '1970-01-05') / 7;  -- 求周数
+-------------------------------------------+
| datediff('1971-01-01', '1970-01-05') / 7 |
+-------------------------------------------+
|                                   51.5714 |  -- 相隔51.5714周
+-------------------------------------------+
1 row in set (0.00 sec)

-- FLOOR 和 ROUND函数
-- FLOOR(arg)是返回一个不大于arg的最大整数，其实就是取整
mysql> select floor(5.4);
+------------+
| floor(5.4) |
+------------+
|          5 |
+------------+
1 row in set (0.00 sec)

mysql> select floor(5.5);
+------------+
| floor(5.5) |
+------------+
|          5 |
+------------+
1 row in set (0.00 sec)

mysql> select floor(5.6);
+------------+
| floor(5.6) |
+------------+
|          5 |
+------------+
1 row in set (0.00 sec)

-- ROUND(X, D) 返回值是对数字X保留到小数点后D位，D默认为0，结果符合四舍五入原则：如果D为负数，则保留小数点左边（整数）的位数
mysql> select round(5.4);  -- 默认D为0，四舍五入
+------------+
| round(5.4) |
+------------+
|          5 |
+------------+
1 row in set (0.00 sec)

mysql> select round(5.5);  -- 默认D为0，四舍五入
+------------+
| round(5.5) |
+------------+
|          6 |
+------------+
1 row in set (0.00 sec)

mysql> select round(5.123, 1); -- 设D为1，保留小数点右边1为，四舍五入，不进位
+-----------------+
| round(5.123, 1) |
+-----------------+
|             5.1 |
+-----------------+
1 row in set (0.00 sec)

mysql> select round(5.163, 1); -- 设D为1，保留小数点右边1为，四舍五入，进位
+-----------------+
| round(5.163, 1) |
+-----------------+
|             5.2 |
+-----------------+
1 row in set (0.00 sec)

mysql> select round(524.163, -1);  -- 保留小数点左边1位，不进位
+--------------------+
| round(524.163, -1) |
+--------------------+
|                520 |
+--------------------+
1 row in set (0.00 sec)

mysql> select round(524.163, -2);  -- 保留小数点左边2位，不进位
+--------------------+
| round(524.163, -2) |
+--------------------+
|                500 |
+--------------------+
1 row in set (0.00 sec)

mysql> select round(554.163, -2);  -- 保留小数点左边2位，进位
+--------------------+
| round(554.163, -2) |
+--------------------+
|                600 |
+--------------------+
1 row in set (0.00 sec)

-- 所以这里使用floor函数取整， 超过一周且不满一周的则落在start 和 end 中间

mysql> select floor(datediff('1971-01-01', '1970-01-05') / 7);
+-------------------------------------------------+
| floor(datediff('1971-01-01', '1970-01-05') / 7) |
+-------------------------------------------------+
|                                              51 |  -- 取整为51周
+-------------------------------------------------+
1 row in set (0.00 sec)

mysql> select floor(datediff('1971-01-01', '1970-01-05') / 7) * 7;
+-----------------------------------------------------+
| floor(datediff('1971-01-01', '1970-01-05') / 7) * 7 |
+-----------------------------------------------------+
|                                                 357 |  -- 乘以7，得出357天差值
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> select adddate('1970-01-05', interval floor(datediff('1971-01-01', '1970-01-05')/7)*7 day) as Monday_start;
-- 使用adddate函数，在1970-01-05(周一)的基础上，增加357天（51周），得到的值1970-12-28也是周一，标记为start
+--------------+
| Monday_start |
+--------------+
| 1970-12-28   |  -- 1970-12-28（周一），而传入的值1971-01-01是该周的周五
+--------------+
1 row in set (0.00 sec)

-- 以同样的方法得到周日
mysql> select adddate('1970-01-05', interval floor(datediff('1971-01-01', '1970-01-05')/7)*7 + 6 day) as Sunday_end;
+------------+
| Sunday_end |
+------------+
| 1971-01-03 |
+------------+
1 row in set (0.00 sec)

-- 通过以上的周一和周日的计算，即可求出1971-01-01这天所在的一周，得出start（周一）和end（周日）
-- 然后对start和end以及o_custkey做分组操作，并通过count(o_orderkey)得到分组的定单量
```

- **使用子查询实现RowNumber**

1. 思路
   - 假设当前在第N行记录，通过主键emp_no遍历有多少行的记录 `小于等于` 当前行,即为当前行的行数

2. SQL语句

```
SELECT
(SELECT COUNT(1) FROM employees b WHERE b.emp_no <= a.emp_no ) AS row_number,
emp_no,CONCAT(last_name," ",first_name) name,gender,hire_date
FROM employees a ORDER BY emp_no LIMIT 10;

-- 假设当前在第5行
mysql> select  b.emp_no  from employees.employees as b order by b.emp_no limit 5;
+--------+
| emp_no |
+--------+
|  10001 |
|  10002 |
|  10003 |
|  10004 |
|  10005 |  -- 第5行的emp_no是10005
+--------+
5 rows in set (0.00 sec)

mysql> select  count(*)  from employees.employees as b where b.emp_no <= 10005 order by b.emp_no;
查找小于等于5的行数有几行
+----------+
| count(*) |
+----------+
|        5 |  -- 小于等于10005的记录有5行，则5就是10005该行记录的行号
+----------+
1 row in set (0.00 sec)

-- 将该子查询的结果即可作为RowNumber
-- 但是该子查询的效率较低。不推荐使用。

-- 推荐Day012中提及的自定义变量方式
SELECT @a:=@a+1 AS row_number,
    emp_no,CONCAT(last_name," ",first_name) name,gender,hire_date
    FROM employees,(SELECT @a:=0) AS a LIMIT 10;
```

二. Rank

> 给出不同的用户的分数，然后根据分数计算排名

```
mysql> create table test_rank(id int, score int);
Query OK, 0 rows affected (0.16 sec)

mysql> insert into test_rank values(1, 10), (2, 20), (3, 30), (4, 30), (5, 40), (6, 40);
Query OK, 6 rows affected (0.05 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> select * from test_rank;
+------+-------+
| id   | score |
+------+-------+
|    1 |    10 |
|    2 |    20 |
|    3 |    30 |
|    4 |    30 |
|    5 |    40 |
|    6 |    40 |
+------+-------+
6 rows in set (0.00 sec)

mysql> set @prev_value := NULL;  -- 假设比较到第N行，设置一个变量prev_value用于存放第N-1行score的分数
                                  -- 用于比较第N行的score和第N-1行的score
                                  -- prev_value可以理解为 是临时保存第N-1行的score的变量
Query OK, 0 rows affected (0.00 sec)

mysql> set @rank_count := 0;      -- 用于存放当前的排名
Query OK, 0 rows affected (0.00 sec)

mysql> select  id, score,
    -> case
    ->     when @prev_value = score then @rank_count
           -- 相等则prev_value不变，  并返回rank_count（第一次为NULL，不会相等，所以跳转到下一个when语句）
    ->     when @prev_value := score then @rank_count := @rank_count + 1
           -- 不等，则第N行的score赋值(:=)给prev_value。且rank_count增加1
    -> end as rank_column   -- case 开始的，end结尾
    -> from test_rank
    -> order by score desc;
+------+-------+-------------+
| id   | score | rank_column |
+------+-------+-------------+
|    5 |    40 |           1 |
|    6 |    40 |           1 |
|    3 |    30 |           2 |
|    4 |    30 |           2 |
|    2 |    20 |           3 |
|    1 |    10 |           4 |
+------+-------+-------------+
6 rows in set (0.00 sec)

-- case
--   when [condition_1] then [do_something_1]
--   when [condition_2] then [do_something_2]
--   end
-- 语法：如果 condition_1条件满足，则执行 do_something_1 然后就跳出,不会执行condition_2;
--       如果 condition_1条件不满足，则继续执行到 condition_2。以此类推。
--
-- 作业：使用一条语句写出Rank
--
```

三. 视图

> 官方文档

- **创建视图**

```
--
-- 创建视图
--
mysql> create view view_rank as select * from test_rank;  -- 针对上面的test_rank创建一个视图
Query OK, 0 rows affected (0.03 sec)

-- 也可以对select结果增加条件进行过滤后，再创建视图


mysql> show create table test_rank\G
*************************** 1. row ***************************
       Table: test_rank
Create Table: CREATE TABLE `test_rank` (    -- 得到的是表结构
  `id` int(11) DEFAULT NULL,
  `score` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)


mysql> show create table view_rank\G  -- 他是以一张表的形式存在的，可通过show tables看到
*************************** 1. row ***************************
                View: view_rank
         Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `view_rank` AS select `test_rank`.`id` AS `id`,`test_rank`.`score` AS `score` from `test_rank`
                      -- 和真正的表不同的是，这里show出来的是视图的定义
character_set_client: utf8
collation_connection: utf8_general_ci
1 row in set (0.00 sec)

mysql> select * from view_rank;  -- 可以直接查询该视图得结果
+------+-------+
| id   | score |
+------+-------+
|    1 |    10 |
|    2 |    20 |
|    3 |    30 |
|    4 |    30 |
|    5 |    40 |
|    6 |    40 |
+------+-------+
6 rows in set (0.00 sec)

-- 视图的作用是，可以对开发人员是透明的，可以隐藏部分关键的列
-- 视图在mysql是虚拟表。根据视图的定义，还是取执行定义中的select语句。


-- 只开放部分列
mysql> create view view_rank_1 as select id from test_rank; -- 只开放id列
Query OK, 0 rows affected (0.04 sec)

mysql> select * from view_rank_1;  -- 即使 select * ，也只能看到id列，具有隐藏原来表中部分列的功能
+------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
+------+
6 rows in set (0.00 sec)

-- 不要取用select * from 去创建视图，因为mysql会把*逐个解析成列。
-- 当原来的表结构发生变化时，视图的表结构是不会发生变化的，视图在创建的瞬间，便确定了结构。
-- 比如，当你alter原来的表 增加列(add columns)时,再去查询该视图,新增加的列是不存在的。

mysql> alter table test_rank add column  c int default 0; -- 增加一列名字为c，默认值为0
Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from test_rank;  -- 查询原表，新的列c出现了
+------+-------+------+
| id   | score | c    |
+------+-------+------+
|    1 |    10 |    0 |
|    2 |    20 |    0 |
|    3 |    30 |    0 |
|    4 |    30 |    0 |
|    5 |    40 |    0 |
|    6 |    40 |    0 |
+------+-------+------+
6 rows in set (0.00 sec)

mysql> select * from view_rank; -- 尽管view_rank用select * 创建，但当时没有列c，所以无法得到c列的值
+------+-------+
| id   | score |
+------+-------+
|    1 |    10 |
|    2 |    20 |
|    3 |    30 |
|    4 |    30 |
|    5 |    40 |
|    6 |    40 |
+------+-------+
6 rows in set (0.00 sec)

-- 注意：mysql中的视图都是虚拟表。不像Oracle可以物化成真实存在的表。
--       每次查询视图，实际上还是去查询的原来的表，只是查询的规则是在视图创建时经过定义的。
```

- **视图的算法**
  视图的算法(`ALGORITHM`)有三种方式：
  - `UNDEFINED` 默认方式，由MySQL来判断使用下面的哪种算法
  - `MERGE` ： 每次 通过 物理表 查询得到结果，把结果merge(合并)起来返回
  - `TEMPTABLE` ：产生一张 临时表 ，把数据放入临时表后，客户端再去临时表取数据（ 不会缓存 ）

> `TEMPTABLE` 特点 ：即使访问条件一样，第二次查询还是会去读取物理表中的内容，并重新生成一张临时表,并不会取缓存之前的表。*（ 临时表是Memory存储引擎，默认放内存，超过配置大小放磁盘 ）*
>
> 当查询有一个较大的结果集时，使用 `TEMPTABLE` 可以快速的结束对该物理表的访问，从而可以快速释放这张物理表上占用的资源。然后客户端可以对临时表上的数据做一些耗时的操作，而不影响原来的物理表。

所以一般我们使用 `UNDEFINED` ，由MySQL自己去判断

四. UNION

1. `UNION` 的作用是将两个查询的结果集进行合并。
2. UNION必须由 两条或两条以上 的SELECT语句组成，语句之间用关键字 `UNION` 分隔。
3. UNION中的每个查询必须包含相同的列（ 类型相同或可以隐式转换 ）、表达式或聚集函数。

```
mysql> create table test_union_1(a int, b int);
Query OK, 0 rows affected (0.18 sec)

mysql> create table test_union_2(a int, c int);
Query OK, 0 rows affected (0.15 sec)

mysql> insert into test_union_1 values(1, 2), (3, 4), (5, 6), (10, 20);
Query OK, 4 rows affected (0.06 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> insert into test_union_2 values(10, 20), (30, 40), (50, 60);
Query OK, 3 rows affected (0.03 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from test_union_1;
+------+------+
| a    | b    |
+------+------+
|    1 |    2 |
|    3 |    4 |
|    5 |    6 |
|   10 |   20 |  -- test_union_1 中的10, 20
+------+------+
4 rows in set (0.00 sec)

mysql> select * from test_union_2;
+------+------+
| a    | c    |
+------+------+
|   10 |   20 | -- test_union_2 中的10, 20
|   30 |   40 |
|   50 |   60 |
+------+------+
3 rows in set (0.00 sec)

mysql> select a, b as t from test_union_1
    -> union
    -> select * from test_union_2;
+------+------+
| a    | t    |
+------+------+
|    1 |    2 |
|    3 |    4 |
|    5 |    6 |
|   10 |   20 | -- 只出现了一次 10, 20，union会去重
|   30 |   40 |
|   50 |   60 |
+------+------+
6 rows in set (0.00 sec)

mysql> select a, b as t from test_union_1
    -> union all   -- 使用 union all 可以不去重
    -> select * from test_union_2;
+------+------+
| a    | t    |
+------+------+
|    1 |    2 |
|    3 |    4 |
|    5 |    6 |
|   10 |   20 | -- test_union_1 中的10, 20
|   10 |   20 | -- test_union_2 中的10, 20
|   30 |   40 |
|   50 |   60 |
+------+------+
7 rows in set (0.00 sec)

mysql> select a, b as t from test_union_1 where a > 2
    -> union
    -> select * from test_union_2 where c > 50;  -- 使用where过滤也可以
+------+------+
| a    | t    |
+------+------+
|    3 |    4 |
|    5 |    6 |
|   10 |   20 |
|   50 |   60 |
+------+------+
4 rows in set (0.00 sec)
```

> 如果知道数据本身具有唯一性，没有重复，则建议使用 `union all` ，因为 `union` 会做 去重操作 ，性能会比 `union all` 要低

五. 触发器

> 官方文档

- **定义**
  - 触发器的对象是 表 ，当表上出现 特定的事件 时 触发 该程序的执行
- **触发器的类型**
  - **`UPDATE`**
    - update 操作
  - **`DELETE`**
    - delete 操作
    - replace 操作
      - **注意：**drop，truncate等DDL操作 不会触发 DELETE
  - **`INSERT`**
    - insert 操作
    - load data 操作
    - replace 操作

> 注意， `replace` 操作会 触发两次 ，一次是 `UPDATE` 类型的触发器，一次是 `INSERT` 类型的触发器
> MySQL 5.6版本同一个类型的触发器只能有一个*(针对一个表)*
> MySQL 5.7允许多个同一类型的触发器
>
> **注意：触发器只触发DML(Data Manipulation Language)操作，不会触发DDL(Data Definition Language)操作*（create,drop等操作）***

- **创建触发器**

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name  -- 触发器名字
    trigger_time trigger_event  -- 触发时间和事件
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }   -- 事件之前还是之后触发

trigger_event: { INSERT | UPDATE | DELETE }  -- 三个类型

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name


mysql> create table test_trigger_1 (
    ->     name varchar(10),
    ->     score int(10),
    -> primary key (name));
Query OK, 0 rows affected (0.14 sec)

    mysql> delimiter //  -- 将语句分隔符定义成 // （原来是';'）
    mysql> create trigger  trg_upd_score  -- 定义触发器名字
        -> before update on test_trigger_1  -- 作用在test_trigger_1 更新(update)之前(before)
        -> for each row  -- 每行
        -> begin  -- 开始定义
        -> if new.score < 0 then   -- 如果新值小于0
        ->     set new.score=0;        -- 则设置成0
        -> elseif new.score > 100 then  -- 如果新值大于100
        ->     set new.score = 100;  -- 则设置成100
        -> end if;  -- begin对应的 结束
        -> end;//   -- 结束，使用新定义的 '//' 结尾
    Query OK, 0 rows affected (0.03 sec)

mysql> delimiter ;  -- 恢复 ';' 结束符

-- new.col : 表示更新以后的值
-- old.col : 表示更新以前的值(只读)

mysql> insert into test_trigger_1 values ("tom", 200);  -- 插入新值
Query OK, 1 row affected (0.04 sec)

mysql> select * from test_trigger_1;
+------+-------+
| name | score |
+------+-------+
| tom  |   200 |  -- 没改成100，因为定义的是update，而执行的是insert
+------+-------+
1 row in set (0.00 sec)

mysql> update test_trigger_1
    -> set score=300 where name='tom'; -- 改成300
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> select * from test_trigger_1;
+------+-------+
| name | score |
+------+-------+
| tom  |   100 | -- 通过触发器的设置，大于100的值被修改成100
+------+-------+
1 row in set (0.00 sec)
```