七 窗口函数避坑指南

- 知道哪些情况下不能使用窗口函数
- 掌握窗口函数与分组聚合一起使用时的技巧

0 数据介绍

- auction表：该表存储了某在线拍卖网站的部分历史网拍信息。 包含以下几列：
  - `id` - 拍卖的唯一ID
  - `category_id` - 类别的ID，例如：家具，卫生用品等
  - `asking_price` - 卖方设定的初始价格
  - `final_price` - 最高竞价，支付给卖家的最终价格
  - `views` - 拍卖的点击次数
  - `participants` - 参与拍卖的用户数量
  - `country` - 拍卖的国家
  - `ended` - 拍卖截止日期

id	category_id	asking_price	final_price	views	participants	country	ended
1	1	190.07	219.66	93	16	Spain	2017/1/5
2	4	34.7	54.7	187	31	Spain	2017/1/5
3	5	124.85	155.95	237	59	Spain	2017/1/5
4	3	141.42	205.73	289	58	Spain	2017/1/6
5	2	31.11	66.45	165	83	Spain	2017/1/9
6	3	106.18	125.07	27	3	Spain	2017/1/6
7	2	124.83	150.93	266	53	Spain	2017/1/6
8	1	151.71	164.54	158	53	Spain	2017/1/8
9	4	51.44	87.02	235	59	France	2017/1/5
10	2	118.97	178.21	191	38	France	2017/1/5
11	5	38.5	69.61	44	7	France	2017/1/6
12	4	20.87	35.57	298	37	France	2017/1/8
13	2	56.45	112.42	267	45	Germany	2017/1/5
14	3	189.2	242.16	234	33	Germany	2017/1/6
15	2	43.15	88.01	92	12	Germany	2017/1/6
16	5	158.92	179.18	17	2	Germany	2017/1/6
17	1	64.55	129.46	155	78	UK	2017/1/5
18	4	196.07	237.86	63	21	UK	2017/1/5
19	2	171.26	190.57	194	39	UK	2017/1/6
20	3	157.81	206.63	218	31	Italy	2017/1/5
21	2	135.16	197.43	47	12	Italy	2017/1/7
22	4	172.98	197.07	297	42	Italy	2017/1/6
23	5	163.89	218.99	90	18	Italy	2017/1/9
24	3	115.76	137.49	136	19	Italy	2017/1/6
25	3	149.89	208.09	25	3	Italy	2017/1/7

1 不能使用窗口函数的情况

1.1 不能在Where子句中使用窗口函数

- 我们在第一小节介绍过，Where 条件中不能使用窗口函数，原因是SQL的执行顺序决定了窗口函数实在Where子句之后执行的，具体执行顺序如下：
  - FROM
  - WHERE
  - GROUP BY
  - 聚合函数
  - HAVING
  - **窗口函数**
  - SELECT
  - DISTINCT
  - UNION
  - ORDER BY
  - OFFSET
  - LIMIT
- 上面的顺序意味着，在FROM，WHERE，GROUP BY或HAVING子句中的写窗口函数会报错，所以下面的写法是错误的

```
SELECT
  id,
  final_price
FROM auction
WHERE final_price > AVG(final_price) OVER()
```

- 上面的SQL中我们想查询出所有拍卖中，最终成交价格高于平均成交价格的拍卖，我们可以使用子查询来实现该需求

```
SELECT
  id,
  final_price
FROM (
  SELECT
    id,
    final_price,
    AVG(final_price) OVER() AS avg_final_price
  FROM auction) c
WHERE final_price > avg_final_price
```

- 我们在FROM子句中使用了子查询，在子查询中我们先计算了平均成交价格，子查询先执行，在外部查询中再去使用

练习78

- 需求：找到浏览次数低于平均浏览次数的拍卖，返回拍卖ID `id`，国家 `country`，浏览次数 `views`

```
SELECT
  id,
  country,
  views
FROM (
  SELECT
    id,
    country,
    views,
    AVG(views) OVER() AS avg_views
  FROM auction) c
WHERE views < avg_views;
```

查询结果

id	country	views
1	Spain	93
6	Spain	27
8	Spain	158
11	France	44
15	Germany	92
16	Germany	17
17	UK	155
18	UK	63
21	Italy	47
23	Italy	90
24	Italy	136
25	Italy	25

1.2 不能在HAVING子句中使用窗口函数

- 使用HAVING的时候也会遇到相同的问题，看下面的SQL，我们的需求是查询出国内平均成交价格高于所有拍卖平均成交价格的国家

```
SELECT
  country,
  AVG(final_price)
FROM auction
GROUP BY country
HAVING AVG(final_price) > AVG(final_price) OVER ();
```

- 上面的SQL中，我们在HAVING中计算了按国家分组的成交平均价格，通过窗口函数计算了所有国家拍卖平均成交价格，但是上面的写法是错误的 运行会报错 3593 - You cannot use the window function 'avg' in this context.', Time: 0.000000s

练习 79

- 修改上面的SQL，使用子查询完成需求

```
SELECT
  country,
  AVG(final_price)
FROM auction
GROUP BY country
HAVING AVG(final_price) > (SELECT AVG(final_price) FROM
auction);
```

查询结果

country	AVG(final_price)
Germany	155.4425
UK	185.963333
Italy	194.283333

1.3 不能在GROUP BY子句中使用窗口函数

- 窗口函数也不能在GROUP BY子句中使用，看下面的SQL

```
SELECT
  NTILE(4) OVER(ORDER BY views DESC) AS quartile,
  MIN(views),
  MAX(views)
FROM auction
GROUP BY NTILE(4) OVER(ORDER BY views DESC);
```

- 上面的SQL中，我们想将所有的拍卖信息按照浏览次数排序，并均匀分成4组，然后计算每组的最小和最大浏览量
  - 运行之后会报错 3593 - You cannot use the window function 'ntile' in this context.', Time: 0.000000s
- 我们还是可以用子查询修改上面的SQL，实现需求

```
SELECT
  quartile,
  MIN(views),
  MAX(views)
FROM
  (SELECT
    views,
    ntile(4) OVER(ORDER BY views DESC) AS quartile
  FROM auction) c
GROUP BY quartile;
```

查询结果

quartile	MIN(views)	MAX(views)
1	235	298
2	165	234
3	90	158
4	17	63

练习 80

- 需求：将所有的拍卖数据按照底价从高到低排序，平均分成6组，将数据按照组别分组，查询每组的最大最小和平均底价，并将所有数据按照组别排序

```
SELECT
  group_no,
  MIN(asking_price),
  AVG(asking_price),
  MAX(asking_price)
FROM (
  SELECT
    asking_price,
    NTILE(6) OVER(ORDER BY asking_price) AS group_no
  FROM auction) c
GROUP BY group_no
ORDER BY group_no;
```

查询结果

group_no	min	avg	max
1	20.87	33.6660	43.15
2	51.44	69.6550	106.18
3	115.76	121.1025	124.85
4	135.16	144.5450	151.71
5	157.81	162.9700	171.26
6	172.98	187.0800	196.07

1.4 在ORDER BY中使用窗口函数

- 通过上面的例子我们知道，只能在SELECT 和 ORDER BY子句中使用窗口函数，WHERE HAVING GROUP BY中只能使用子查询
- 接下来我们来看一下是否能在ORDER BY子句中使用窗口函数
- 需求：将所有的拍卖按照浏览量降序排列，并均分成4组，按照每组编号升序排列
    - `id`，`views` 和分组情况（`quartile`）

```
SELECT
  id,
  views,
  NTILE(4) OVER(ORDER BY views DESC) AS quartile
FROM auction
ORDER BY NTILE(4) OVER(ORDER BY views DESC);
```

查询结果

id	views	quartile
12	298	1
22	297	1
4	289	1
13	267	1
7	266	1
3	237	1
9	235	1
14	234	2
20	218	2
19	194	2
10	191	2
2	187	2
5	165	2
8	158	3
17	155	3
24	136	3
1	93	3
15	92	3
23	90	3
18	63	4

2 窗口函数与GROUP BY一起使用

- 之前我们介绍了，窗口函数在GROUP BY 子句之后执行，那么当我们在SQL中使用了GROUP BY 或者 HAVING对数据进行聚合之后，**窗口函数能处理的数据是聚合之后的数据而不是原始数据**，看下面的例子

```
SELECT
  category_id,
  final_price,
  AVG(final_price) OVER()
FROM auction;
```

- 上面的SQL是一句非常简单的查询，查询了类别ID，成交价格，和所有拍卖的平均成交价格
- 接下来我们对上面的SQL做一个简单的调整，添加一个 `GROUP BY` 子句

```
SELECT
  category_id,
  MAX(final_price),
  AVG(final_price) OVER()
FROM auction
GROUP BY category_id;
```

上面的SQL是错误的，因为经过 `GROUP BY` 分组之后结果只有一列 `category_id` ， 此时再运行窗口函数，数据中并不包含 `final_price`

- 我们再对上述窗口函数进行调整，看下这次能否正确执行

```
SELECT
  category_id,
  MAX(final_price) AS max_final,
  AVG(MAX(final_price)) OVER() AS `avg`
FROM auction
GROUP BY category_id;
```

查询结果

category_id	max_final	avg
3	242.16	223.220000
5	218.99	223.220000
4	237.86	223.220000
2	197.43	223.220000
1	219.66	223.220000

- 可以看到查询成功了，因为我们使用了聚合函数MAX(final_price)，分组之后可以执行,执行之后可以再执行窗口函数
    - 注意，聚合函数嵌套使用这是唯一场景

练习81

- 需求：将拍卖数据按国家分组，查询如下字段：
    - 国家 `country`
    - 每组最少参与人数 `min`
    - 所有组最少参与人数的平均值 `avg`

```
SELECT
  country,
  MIN(participants) AS `min`,
  AVG(MIN(participants)) OVER() AS `avg`
FROM auction
GROUP BY country;
```

查询结果

country	min	avg
Spain	3	7.2
France	7	7.2
Germany	2	7.2
UK	21	7.2
Italy	3	7.2

练习 82

- 需求：按照商品类别 `category_id` 分组，计算每种商品起拍价的最高价格 `max` ，平均最高价格 `avg`

```
SELECT
  category_id,
  MAX(asking_price) AS `max`,
  AVG(MAX(asking_price)) OVER() AS `avg`
FROM auction
GROUP BY category_id;
```

查询结果

category_id	max	avg
1	190.07	182.098
4	196.07	182.098
5	163.89	182.098
3	189.2	182.098
2	171.26	182.098

3 Rank时使用聚合函数

- 我们可以在聚合函数的结果上使用RANK函数，看下面的例子

```
SELECT
  country,
  COUNT(id),
  RANK() OVER(ORDER BY COUNT(id) DESC) AS `rank`
FROM auction
GROUP BY country;
```

查询结果

country	COUNT(id)	rank
Spain	8	1
Italy	6	2
France	4	3
Germany	4	3
UK	3	5

- 我们按国家进行分组，计算了每个国家的拍卖次数，再根据拍卖次数对国家进行排名

练习 83

- 需求： 按商品分类 `category_id` 分组，对成交价格 `final_price` 求和 `sum` ，对所有类别按成交价格的总金额排序，返回序号 `rank`
  - 返回字段 `category_id` ， `sum` ， `rank`

```
SELECT
  category_id,
  SUM(final_price) AS `sum`,
  RANK() OVER(ORDER BY SUM(final_price) DESC) AS `rank`
FROM auction
GROUP BY category_id;
```

查询结果

category_id	sum	rank
3	1125.17	1
2	984.02	2
5	623.73	3
4	612.22	4
1	513.66	5

练习 84

- 需求：按拍卖结束日期 `ended` 分组，计算每组平均浏览量（`views`），并按平均浏览量对所有组排名返回序号 `rank`
  - 返回字段：拍卖结束日期 `ended`，平均浏览量 `avg`，每组排名 `rank`

```
SELECT
  ended,
  AVG(views) AS `avg`,
  RANK() OVER(ORDER BY AVG(views) DESC) AS `rank`
FROM auction
GROUP BY ended;
```

查询结果

ended	avg	rank
2017-01-08	228.0000	1
2017-01-05	182.8889	2
2017-01-06	159.6000	3
2017-01-09	127.5000	4
2017-01-07	36.0000	5

4 利用GROUP BY计算环比

- 我们可以利用GROUP BY 分组之后，结合**leads**, **lags** 计算环比（相邻两天的差值），看下面的SQL

```
SELECT
  ended,
  SUM(final_price) AS sum_price,
  LAG(SUM(final_price)) OVER(ORDER BY ended)
FROM auction
GROUP BY ended
ORDER BY ended
```

- 我们利用GROUP BY按结束拍卖日期对所有拍卖进行分组，对每天结束的所有拍卖的最终成交价格求和，又利用LAG函数计算了前一天的全天最终成交价格

查询结果

ended	sum_price	lag
2017/1/5	1381.91	NULL
2017/1/6	1585.82	1381.91
2017/1/7	405.52	1585.82
2017/1/8	200.11	405.52
2017/1/9	285.44	200.11

练习 85

- 需求：按拍卖结束日期 `ended` 分组分析所有拍卖的浏览数据 `views`，返回如下字段：
  - 每组的拍卖结束日期 `ended`
  - 每组的总浏览量 `sum`
  - 每组的前一组总浏览量 `previous_day`
  - 比较结束日期相邻两天浏览量的差值 `delta`

```
SELECT
  ended,
  SUM(views) AS `sum`,
  LAG(SUM(views)) OVER(ORDER BY ended) AS previous_day,
  SUM(views) - LAG(SUM(views)) OVER(ORDER BY ended) AS delta
FROM auction
GROUP BY ended
ORDER BY ended;
```

查询结果

ended	sum	previous_day	delta
2017-01-05	1646	null	null
2017-01-06	1596	1646	-50
2017-01-07	72	1596	-1524
2017-01-08	456	72	384
2017-01-09	255	456	-201

5 对GROUP BY分组后的数据使用 PARTITION BY

- 我们可以对GROUPY BY分组后的数据进一步分组（`PARTITION BY`），再次强调，使用`GROUP BY`之后使用窗口函数，只能处理分组之后的数据，而不是处理原始数据，看下面的例子

```
SELECT
  country,
  ended,
  SUM(views) AS views_on_day,
  SUM(SUM(views)) OVER(PARTITION BY country)
    AS views_country
FROM auction
GROUP BY country, ended
ORDER BY country, ended
```

- 上面的SQL中：
  - 我们先将所有的数据按照国家`country`和拍卖结束时间`ended`分组，然后显示了国家名字，和拍卖结束日期
  - 接下来的两句，`SUM(views) AS views_on_day`根据`GROUP BY`分组结果（先国家，后日期），对每组的浏览量求和
  - `SUM(SUM(views)) OVER(PARTITION BY country) AS views_country`这是一个窗口函数，只对国家进行分组，计算每个国家拍卖的总浏览量

查询结果

country	ended	views_on_day	views_country
France	2017/1/5	426	768
France	2017/1/6	44	768
France	2017/1/8	298	768
Germany	2017/1/5	267	610
Germany	2017/1/6	343	610
Italy	2017/1/5	218	813
Italy	2017/1/6	433	813
Italy	2017/1/7	72	813
Italy	2017/1/9	90	813
Spain	2017/1/5	517	1422
Spain	2017/1/6	582	1422
Spain	2017/1/8	158	1422
Spain	2017/1/9	165	1422
UK	2017/1/5	218	412
UK	2017/1/6	194	412

练习86

- 需求：将所有数据按照类别 `category_id` 和拍卖结束日期 `ended` 分组，返回
  - 类别ID， `category_id`
  - 结束日期 `ended`
  - 当前类别，当日结束的拍卖的平均成交价格 `daily_avg_final_price`
  - 所有类别日平均成交价格最大值 `daily_max_avg`

```sql
SELECT
  category_id,
  ended,
  AVG(final_price) AS daily_avg_final_price,
  MAX(AVG(final_price)) OVER(PARTITION BY category_id) AS
daily_max_avg
FROM auction
GROUP BY category_id, ended
ORDER BY category_id, ended;
```

查询结果

category_id	ended	daily_avg_final_price	daily_max_avg
1	2017-01-05	174.5600	174.5600
1	2017-01-08	164.5400	174.5600
2	2017-01-05	145.3150	197.4300
2	2017-01-06	143.1700	197.4300
2	2017-01-07	197.4300	197.4300
2	2017-01-09	66.4500	197.4300
3	2017-01-05	206.6300	208.0900
3	2017-01-06	177.6125	208.0900
3	2017-01-07	208.0900	208.0900
4	2017-01-05	126.5267	197.0700
4	2017-01-06	197.0700	197.0700
4	2017-01-08	35.5700	197.0700
5	2017-01-05	155.9500	218.9900
5	2017-01-06	124.3950	218.9900
5	2017-01-09	218.9900	218.9900

小结

- 窗口函数只能出现在SELECT和ORDER BY子句中
- 如果查询的其他部分（WHERE，GROUP BY，HAVING）需要窗口函数，请使用子查询，在子查询中使用窗口函数
- 如果查询使用聚合或GROUP BY，请记住窗口函数只能处理分组后的结果，而不是原始的表数据

练习数据介绍

- book表，每一本书都有：
  - 唯一的ID `id`，作者ID `author`，出版日期 `publish_year` ，售价 `price` 以及评分 `rating` (0~10分)

id	author_id	publish_year	price	rating
1	1	2016	40	9
2	1	2016	33	5
3	2	2016	28	7
4	2	2015	31	9
5	3	2015	37	5
6	3	2014	31	3
7	1	2016	35	7
8	2	2016	28	5
9	1	2015	32	7
10	3	2016	36	3

练习 87

- 需求：把所有的书按照评分从低分到高分平均分成4组，给每组分配一个编号（ `bucket` ），分别取出每组最低 `min` 和最高 `max` 的得分
  - 返回字段： `bucket` ， `min` ， `max`

```sql
SELECT
  bucket,
  MIN(rating) AS `min`,
  MAX(rating) AS `max`
FROM (
  SELECT
    rating,
    NTILE(4) OVER(ORDER BY rating) AS bucket
  FROM book) c
GROUP BY bucket;
```

查询结果

bucket	min	max
3	7	7
4	9	9
2	5	7
1	3	5

练习 88

- 需求：统计作者的出版数量，对每一个作者显示:
  - 作者id `author_id`
  - 这个作者出版了多少本书（`number_of_books`）
  - 按照出版书籍多少降序排列的排名 `rank`

```sql
SELECT
  author_id,
  COUNT(id) AS number_of_books,
  RANK() OVER(ORDER BY COUNT(id) DESC) AS `rank`
FROM book
GROUP BY author_id;
```

author_id	number_of_books	rank
1	4	1
3	3	2
2	3	2

练习 89

- 需求：分析每年书籍出版情况
  - 发行年 `publish_year`
  - 当年发行了多少本书 `count`
  - 前一年发行了多少本书 `lag`

```
SELECT
  publish_year,
  COUNT(id) AS `count`,
  LAG(COUNT(id)) OVER(ORDER BY publish_year) AS `lag`
FROM book
GROUP BY publish_year
ORDER BY publish_year;
```

查询结果

publish_year	count	lag
2014	1	null
2015	3	1
2016	6	3