Department of Computer Science

1 Castle Point Terrace
Hoboken, NJ 07030
stevens.edu

SQL Homework #2 – Main Ideas

Sample queries to show how to compute/pull data for BEFORE and AFTER (the current months, quarters, etc.). Try running the queries below with the simple data provided (i.e., "months" table).

```
/*---
create table months
(
   month integer
);

insert into months values (1);
insert into months values (2);
insert into months values (3);
insert into months values (4);
insert into months values (5);
insert into months values (6);
insert into months values (7);
insert into months values (8);
insert into months values (9);
insert into months values (10);
insert into months values (11);
insert into months values (12);

select * from months;
---*/

WITH before as
(
   select m1.month, m2.month before_mo
     from months m1 left join months m2
       on m1.month - 1 = m2.month
),
after as
(
   select m1.month, m2.month after_mo
     from months m1 left join months m2
       on m1.month + 1 = m2.month
)
select c.month, b.before_mo, a.after_mo
  from before b, months c, after a
 where b.month = c.month
   and c.month = a.month
```

Department of Computer Science
1 Castle Point Terrace
Hoboken, NJ 07030
stevens.edu

Query #1:

For each *customer, product* and *month, count* the number of sales transactions that were between the *previous* and the *following* month's average sales quantities. For January and December, display <NULL> or 0.

1. Create a "reference" table (e.g., using WITH) for AVG (quant) for (current month – 1) and AVG (quant) for (current month + 1)), e.g.,

CUST	PROD	MONTH	PREV_AVG	NEXT_AVG
Sam	Banana	1	<NULL>	20
Sam	Banana	2	10	30
Sam	Banana	3	20	40
. . .	. . .	. . .	. . .	. . .

There are many ways to generate the "reference" table above, here's one version that will show you in a step-by-step fashion:

a. (q1)-compute avg() for (cust, prod, month)
b. (q2)-using q1, compute the avg() for (cust, prod, month-1), i.e., month before
c. (q3)-using q1, compute the avg() for (cust, prod, month+1), i.e., month after
d. (q4)-join q1 & q2 to bring current month's avg and previous month's avg
e. (q5)-join q3 & q4 to bring avg's for before month, current month, and after month
f. (q6)-(to make things simple), you can "covert" NULLs to 0's for month before
g. (q7) convert NULLs to 0's for month after
h. (q8) join q6 & q7 to have exactly the same content as q5, except all NULLs are now converted to 0's

⇨ At this point, you have a complete picture of the "reference" table – looks somewhat cumbersome, but I'm trying to show you how to create such a table, step-by-step.

2. Join the "reference" table (q8 from above) with the "sales" table to compute the final result (COUNT(Quant))

```
SELECT . . .
  FROM sales s, reference r
 WHERE s.month = r.month
   AND (s.quant between r.prev_avg and r.next_avg)
    OR (s.quant between r.next_avg and r.prev_avg)
GROUP BY . . .
```

Query #2:

For *customer* and *product*, show the average sales *before*, *during* and *after* each *month* (e.g., for February, show average sales of January and March. For "before" January and "after" December, display <NULL>. The "YEAR" attribute is not considered for this query – for example, both January of 2017 and January of 2018 are considered January regardless of the year.

- This query is essentially the same the sub query of Query #1 that produces the "reference" table.

Department of Computer Science
1 Castle Point Terrace
Hoboken, NJ 07030
stevens.edu

Query #3:

For each *customer*, *product* and *state* combination, compute (1) the product's average sale of this customer for the state (i.e., the simple AVG for the group-by attributes – this is the easy part), (2) the average sale of the product and the state but for *all of the other customers*, (3) the customer's average sale for the given state, but for *all of the other products*, and (4) the customer's average sale for the given product, but for *all of the other states*.

1. Create a "base table" by computing CURRENT_AVG for (cust, prod, state)

CUST	PROD	STATE	CURRENT_AVG
Dan	Apple	NJ	20
Dan	Banana	NY	30
Dan	Cherry	CT	40
. . .	. . .	. . .	. . .

2. Join output of step 1 with "sales" to compute OTHER_CUST_AVG

```
SELECT . . . AVG(s.quant)
  FROM q1, sales s
 WHERE q1.prod = s.prod AND q1.state = s.state AND q1.cust != s.cust
GROUP BY q1.cust, q1.prod, q1.state
```

3. Similarly, join output of step 1 with "sales" to compute OTHER_PROD_AVG
4. Similarly, join output of step 1 with "sales" to compute OTHER_STATE_AVG
5. Join output of steps 1, 2, 3 & 4.

Query #4:

For each *customer,* find the top 3 highest quantities purchased in New Jersey (NJ). Show the customer's name, the quantity and product purchased, and the date they purchased it. If there are ties, show all – refer to the sample output below.

This is probably the easiest query of HW #2 – the idea is to look for "top 3" maximum quantities for the given customer:

- Part 1 – the first max is the most straightforward (i.e., it's a simple MAX(QUANT) for customers).
- Part 2 – to find the 2nd highest quantities, you must find the maximum from the quantities that are less than THE maximum quantities (from Part 1) – this can be done by joining 'sales' table and the output of Part 1.

```
SELECT s.cust, MAX(s.quant) AS second_max
  FROM sales AS s, part1 AS p1
 WHERE p1.cust = s.cust
   AND s.quant < p1.first_max
   AND state = 'NJ'
GROUP BY s.cust
```

- Part 3 – for the 3rd highest, you must find the maximum from the quantities that are less than both THE maximum and 2nd highest quantities – similar to part 2, this can be done by joining 'sales' and the output of part 2.
- … then UNION the 3 parts together.

This would be one way to do it, and there are many other ways to get the same answer.

Department of Computer Science
1 Castle Point Terrace
Hoboken, NJ 07030
stevens.edu

Query #5:

For each product, find the median sales quantity (assume an odd number of sales for simplicity of presentation). (NOTE – "***median***" is defined as "*denoting or relating to a value or quantity lying at the midpoint of a frequency distribution of observed values or quantities, such that there is an equal probability of falling above or below it.*" E.g., Median value of the list {13, 23, 12, 16, 15, 9, 29} is 15.

1. For each (prod, quant), compute the "position" ➔ count (quant), where quant <= current quant. Call this 'pos' (temp table).

PROD	QUANT	POS
Apple	10	1
Apple	20	4
Apple	20	4
Apple	20	4
Apple	50	5
. . .	. . .	. . .

NOTE: For the purpose of 'pos', you should create and use a "base" table of all combinations of (prod, quant) first, and join with the 'sales' table (vs. doing a self-join between 2 copies of 'sales').

```
with base as
(
      select distinct prod, quant
        from sales
      order by 1, 2
),
...
```

2. With the output above, find the "median position" with min (pos's that are >= ceiling (count (quant) / 2). The QUANT associated with the "median position" is the MEDIAN QUANT. Call this 'med_pos' (temp table).

PROD	MEDIAN_POS
Apple	3
Banana	5
Cherry	14
. . .	. . .

3. Join the results of 1 ('pos') & 2 ('med_pos') based on the 'pos.pos' and 'med_pos.median_pos' column, and the goal is the find the (prod, quant) pair of the 'pos' table. Because the list of QUANT's can contain duplicates (such as the example above), we cannot simple use the condition, "pos.pos = med_pos.median_pos'; instead, we need to find 'min (pos's that are >= median_pos)".

With the 2 temp tables, 'pos' (from step 1) and 'med_pos' (from step 2) . . .

```
WITH t1 (
   SELECT p.prod, p.quant, p.pos
     FROM pos p, med_pos mp
    WHERE p.prod = mp.prod and p.pos >= mp.median_pos
)
```

This will produce a temp table, t1 as follows:

PROD	QUANT	POS
Apple	20	4
Apple	20	4
Apple	20	4
Apple	50	5
. . .	. . .	. . .

The final step is to find the QUANT corresponding to the min(QUANT) – in the case above, this will find (Apple, 20)

```
WITH t2 (
   SELECT prod, quant, p.prod, min(p.quant) median_quant
     FROM t1
   GROUP BY prod
)
```

Below, I'm providing further details of the idea behind the query:

a. (base)-create all distinct combinations of (prod, quant), mainly as a base table to collect 'relative positions' for each quant of each prod.
b. (pos)-compute the relative position for each combo of (prod, quant) by joining 'base' and 'sales'.
c. (med_pos)-compute the 'median position' for each quant of each prod.
d. (meds)-in case there are multiple median (prod, quant)'s, we need to first collect ALL (prod, quant) pairs whose 'quant' values are >= median quant.
e. (FINAL query)-from 'meds', find the min(quant) which is the median quant.

```
with base as
(
      select distinct prod, quant
        from sales
      order by 1, 2
),
pos as
(
      select b.prod, b.quant, count(s.quant) pos
        from base b, sales s
       where ...
),
```

```
med_pos as
(
      ...
),
meds as
(
      ...
),
FINAL QUERY to find min() of meds
```