Name: Eva Lin

1.)

Using the **pir** database, generate a result set that contains the biggest winner (i.e. most total winnings) on each day of the show. Your result set should contain three columns: the date of the show, the biggest winner’s name on that date, and the biggest winner’s total winnings on that date (call this column “tot\_winnings”). Order your output by the date of the show in ascending order.

*Query*

with wins as

(

select date,name, SUM(Price)OVER(PARTITION BY date, name) as tot\_winnings

from five\_ep

where win = 1.0),

t2 AS

(

select \*,

MAX(tot\_winnings)OVER(PARTITION BY date) AS max\_winning

from wins

group by date,name)

select date,

name,

tot\_winnings

from t2

where tot\_winnings = max\_winning

order by date;

*Output*

date, name, tot\_winnings

2022-05-02, Michael, 19964

2022-05-03, Renee, 29327

2022-05-04, Caleigh, 60454

2022-05-05, Phillip, 36819

2022-05-06, Lisa, 49455

2.) Redo the following problem from HW#3 *without using any JOINs*:

Again using the **pir** database, focus on the “Big Wheel.” In particular, determine the highest first spin amount where the contestant decided to take a second spin and their total for both spins was still less than or equal to $1.00. Your result set should be a single value. (Feel free to ignore the one instance where a contestant took a third “bonus” spin. Put differently, focus only on a contestant’s first two spins.)

*Query*

with t1 as

(

select date, name, eventAmount, eventOrder, spinNum,

sum(eventAmount) over (partition by date, name) as total\_amount,

sum(eventAmount) over (partition by date, name order by date, eventOrder rows between unbounded preceding and current row) as seperate\_total\_amount

from five\_ep

where eventType = 'Big Wheel'

)

select MAX(seperate\_total\_amount)

from t1

where total\_amount<=1 and total\_amount!=seperate\_total\_amount;

*Output*

# MAX(seperate\_total\_amount)

0.50

3.)

Again using the **pir** database, focus on “Bidders Row.” Depending on the previous three contestants’ bids, it can sometimes be a good strategy for the fourth (i.e. last) bidder to bid exactly $1 more than the maximum bid up to that point. (In doing this, as long as the fourth bidder’s bid is less than the prize value, they will automatically win.) So, first identify all of the instances on “Bidders Row” where the last bidder bids exactly $1 more than maximum of the first three bids. What percentage of the time does the 4th bidder win when using this strategy? Your result set should have two rows and three columns. The first column (call this column “outcome\_group”) should have two categories: “bidder4\_wins” and “bidder4\_loses”. The other two columns should contain the appropriate counts (call this column “n\_times”) and percentages (call this column “percent\_total”) for each category.

*Query*

with t1 as

(

SELECT date, name, eventTypeCounter, eventOrder, win, eventAmount,

max(eventAmount) over (PARTITION BY date, eventTypeCounter order by date, eventTypeCounter, eventOrder rows between unbounded preceding and 1 preceding ) as max

FROM five\_ep

WHERE eventType= 'Bidders Row'

ORDER BY date, eventTypeCounter, eventOrder

),

t2 AS(

select \*, CASE WHEN win=1 THEN 'bidder4\_wins' ELSE 'bidder4\_loses' END AS outcome\_group

from t1

where eventOrder = 4 AND eventAmount - 1= max

)

SELECT outcome\_group, COUNT(\*) as n\_times, COUNT(\*)/(SELECT COUNT(\*) FROM t2) as percent\_total

FROM t2

GROUP BY outcome\_group;

*Output*

outcome\_group, n\_times, percent\_total

bidder4\_loses, 4, 0.5000

bidder4\_wins, 4, 0.5000

4.) How often do bidders make less-accurate bids on “Bidders Row” as the show progresses? Specifically, focus on all contestants that make at least two bids on “Bidders Row” during a show. You can define a bid’s “accuracy” as the absolute value of the difference between the bid and the actual value of prize. We want to find all contestants who are making incrementally “less accurate” bids for each bid they make on “Bidders Row.” If a contestant is “equally accurate” (i.e. the absolute value of the difference between the bid and the prize price is the same on consecutive bids) they should still be included here. Provide a count of the number of contestants who are incrementally “less accurate” for ALL of their bids, the average number of bids that they make, and the percentage of contestants in this set who actually win on their last bid. Just to be clear, your output should be a single row with three columns (one for each of the values we want to calculate).

*Query*

with t1 as

(

SELECT date, name, win, eventAmount, price, eventTypeCounter,

COUNT(\*) OVER(PARTITION BY name, date) AS bid\_times, ABS(eventAmount - price) AS delta

FROM five\_ep

WHERE eventType = 'Bidders Row'

ORDER BY date, name, eventTypeCounter

),

t2 as(

select date, name, win, eventAmount, price, eventTypeCounter, bid\_times, delta

from t1

where bid\_times >1

),

t3 as

(

select date, name, win, eventAmount, price, eventTypeCounter, bid\_times, delta,

avg(delta) over (partition by date,name order by eventTypeCounter rows between 1 preceding and current row ) as A

from t2

)

,t4 as (

select date, name,win, eventAmount, price, eventTypeCounter, bid\_times, delta,A,

case when delta>A then 'Yes'

else 'No' end as YN

from t3

),

t5 as

(

select date, name, win, eventAmount, price, eventTypeCounter, bid\_times, delta,A,YN, sum(YN='Yes') OVER (PARTITION BY date, name) as Final

from t4

ORDER BY date, name, eventTypeCounter

),

t6 as

(

select date, name, win, eventAmount, price, eventTypeCounter, bid\_times, delta,A,YN, Final, RANK() OVER(PARTITION BY date, name ORDER BY eventTypeCounter) as rank\_bid

from t5

where bid\_times-1 = Final

)

select COUNT(\*), avg(bid\_times), SUM(win=1)/COUNT(\*)

FROM t6

WHERE bid\_times = rank\_bid;

*Output*

COUNT(\*), avg(bid\_times), SUM(win=1)/COUNT(\*)

11, 2.2727, 0.4545

5.)

For each carrier, determine the average distance that each of its planes flies each day. Only include planes that have a tail number beginning with 'N'. IMPORTANT: only consider plane-days where the plane flies for exactly one carrier in that day. (And do not exclude cases by "hard-coding" those instances.) Exclude all cancelled and diverted flight records. Your output should include the airline carrier code, the total number of records for each carrier code, and the average distance that each of its planes flies each day.

**HINT—you will need to do some investigating on the internet here. There is no built-in “count distinct window function” but there is a way to implement that logic.**

*Query*

with t1 as

(

SELECT UniqueCarrier, TailNum, DATE(CONCAT(Year,'-',Month,'-',DayofMonth)) AS Date\_Field,

SUM(Distance) AS tot\_dist,

COUNT(Distance) OVER(PARTITION BY UniqueCarrier, TailNum, Date\_Field) AS CarrNum

FROM ontime

WHERE TailNum LIKE 'N%' AND

Cancelled = 0 AND

Diverted = 0

GROUP BY UniqueCarrier, TailNum, Date\_Field

)

SELECT UniqueCarrier,

AVG(tot\_dist),

COUNT(\*)

FROM t1

WHERE CarrNum = 1

GROUP BY UniqueCarrier;

*Output*

# UniqueCarrier, AVG(tot\_dist), COUNT(\*)

AA, 3500.6994, 187255

AQ, 3316.8476, 5931

AS, 3802.9433, 37063

B6, 4722.7156, 44343

CO, 3493.3098, 104591

DL, 3554.1577, 124297

EV, 2462.7026, 53478

F9, 4259.4716, 20431

FL, 3755.2460, 48041

HA, 3725.7369, 9473

MQ, 2447.0637, 87584

NW, 3071.3040, 101901

OH, 2329.0149, 46362

OO, 2676.2769, 93112

UA, 3959.0233, 130763

US, 3503.0464, 113695

WN, 4189.6043, 173853

XE, 2592.5843, 91070

YV, 2511.4389, 44176

6.) Provide a “Top-2” ranking for each month in terms of greatest average departure delay by airport. Focus only on non-cancelled, non-diverted flights. Only consider airports that have at least 1000 departures in every month (i.e. if an airport has at least one month with less than 1000 departures then it should be excluded from the rankings in all months). Use month names (i.e. January, February, etc.) and airport names instead of the “Origin” code. Your output should include month names, airport names, and rankings. Order your output from January to December then by ranking in ascending order.

*Query*

use airline\_ontime;

with t1 as

(

select Origin,

Name, avg(DepDelay) as avgdelay, MONTH(CONCAT(Year,'-',Month,'-',DayofMonth)) AS time,

count(\*) AS tot

from ontime

inner join airports on ontime.Origin = airports.IATA

where Cancelled = 0 and Diverted = 0

group by Origin, time),

t2 as

(select \*,

case when tot >= 1000 then 1

else 0 end as tot101

from t1),

t3 as

(select \*, sum(tot101) over(partition by Origin) as tot101sum

from t2),

t4 as

(

select \*,

rank() over(partition by time order by avgdelay desc) as avgdelayrank

from t3

where tot101sum = 12)

select MONTHNAME(STR\_TO\_DATE (time,'%m')) as monthy,

name, avgdelay, avgdelayrank

from t4

where avgdelayrank <= 2;

*Output*

# monthy, name, avgdelay, avgdelayrank

January, Chicago O'Hare International Airport, 20.3296, 1

January, Dallas Fort Worth International Airport, 15.4303, 2

February, Chicago O'Hare International Airport, 27.8510, 1

February, John F Kennedy International Airport, 24.9096, 2

March, Newark Liberty International Airport, 23.3893, 1

March, Chicago O'Hare International Airport, 22.7968, 2

April, Newark Liberty International Airport, 22.1133, 1

April, Philadelphia International Airport, 19.3934, 2

May, Dallas Love Field, 15.2588, 1

May, Newark Liberty International Airport, 15.0690, 2

June, Dallas Fort Worth International Airport, 28.9849, 1

June, Newark Liberty International Airport, 26.9831, 2

July, John F Kennedy International Airport, 24.6021, 1

July, Newark Liberty International Airport, 24.0898, 2

August, John F Kennedy International Airport, 24.1843, 1

August, Newark Liberty International Airport, 23.0243, 2

September, Hartsfield Jackson Atlanta International Airport, 13.0577, 1

September, Ted Stevens Anchorage International Airport, 10.6383, 2

October, Newark Liberty International Airport, 16.0838, 1

October, Philadelphia International Airport, 14.8218, 2

November, Newark Liberty International Airport, 17.9794, 1

November, La Guardia Airport, 13.7912, 2

December, Des Moines International Airport, 37.3682, 1

December, Chicago O'Hare International Airport, 32.6914, 2

7.) Consider all of the users in our “yelp” database that have made at least 10 reviews. (You want to verify the number of reviews that a user has made using the data in the “review” table NOT the “review\_count” column in the “user” table.) For all of these users, what is the average number of time (in days) between their 1st and 10th reviews?

*Query*

use yelp;

with t1 as(

select user\_id, date, row\_number() over(partition by user\_id order by Date) as A, count(\*) over(partition by user\_id) AS NEW

FROM review

) ,

T2 as

(

select user\_id, date as D1, lag(DATE,9) over (partition by user\_id order by A)as D2, A

from T1

where NEW >= 10

)

select avg(DATEDIFF(D1,D2))

from T2

where A=10;

*Output*

# avg(DATEDIFF(D1,D2))

723.0240

8.) Yikes! That’s a pretty long time! Let’s dig a little deeper. Consider the same set of users from the previous question. But now, split them into two different groups: those that gave a “high” rating (i.e. 4 or 5 stars) on their first review versus those that gave a “low” rating on their first review (i.e. less than 4 stars). For each of these groups, calculate the average amount of time (in days) *between each consecutive review in the first 10 reviews*. So, you are calculating the average amount of time between the 1st and 2nd reviews, the 2nd and 3rd reviews, the 3rd and 4th reviews, etc. Your output should have 9 rows and 3 columns and should look like the template I have included below…

*Query*

with t1 as

(SELECT user\_id, COUNT(\*)

FROM review

GROUP BY user\_id

HAVING COUNT(\*) >= 10),

t2 as

(SELECT review.\*,row\_number()over(partition by user\_id order by date) as Aorder

from t1

inner join review

on review.user\_id = t1.user\_id),

t3 as

(

select user\_id,stars,date, Aorder,

case when stars >= 4 and Aorder = 1 then 'HIGH'

when stars < 4 and Aorder = 1 then 'LOW'

else NULL end as scale\_rate

from t2),

t4 as

(select \*,

max(scale\_rate)over(partition by user\_id) as idname,

lead(date)over(PARTITION BY user\_id ORDER BY Aorder) as next\_date

from t3)

SELECT avg(case when idname = 'HIGH' then DATEDIFF(next\_date, date)

else NULL end) AS avghigh,

avg(case when idname = 'LOW' then DATEDIFF(next\_date, date)

else NULL end) as avglow

FROM t4

WHERE Aorder <= 9

GROUP BY Aorder

*Output*

avghigh, avglow

109.9347, 130.6891

80.8857, 91.3200

72.3944, 79.8019

68.4676, 74.1342

65.8324, 72.3623

68.8059, 73.4430

70.7861, 75.0033

76.2304, 79.2411

86.4745, 90.2166

9.) Homer Simpson loves many things in life, including beer (his favorite brand is “Duff”). In what season/episode does Homer say “beer” or “Duff” for the 50th, 150th, and 250th times? Be sure to only include instances of “beer” or “Duff” linked to Homer’s character\_id=2. Be aware that a single record in the “script\_lines” table can contain multiple instances of “beer” and/or “Duff” and I want you to count all of them. For instance, if Homer says “I love Duff beer. Duff beer is the greatest!” in one “script\_lines” record then that counts as 4, not 1. Specifically, count all instances of the strings “beer” and “duff” in the “normalized\_text” field. (You don’t need to worry about any oddities here. For instance, if Homer says “beeeeeer” out of excitement, something like that can be ignored. You should just want to identify all records containing “beer” and/or “duff” and count the number of instances of these two strings in those records.) Your output should include the season, episode number within the season, episode title, location, raw\_text, and ‘beer/duff counter’ for each of the 50th, 150th, and 250th occurrences (so your output should only have 3 rows). (It’s okay to make the font for your output very small and wrap text for the title and raw\_text columns.)

*Query*

use simpsons;

with t1 as(

select \*,

(CHAR\_LENGTH(lower(normalized\_text))-CHAR\_LENGTH(REPLACE(REPLACE(LOWER(normalized\_text),'beer',''), 'duff','')))/4 as totnum

from script\_lines

where character\_id=2 and

(lower(normalized\_text) like '%beer%' or lower(normalized\_text) like '%duff%')),

t2 as(

select \*,

sum(totnum) OVER(ORDER BY id ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS sumtotnum

FROM t1)

select episodes.season,

episodes.number\_in\_season,

episodes.title,lower(raw\_location\_text) as loc, raw\_text, round(totnum) as 'beer/duff counter'

from t2

left join episodes

on t2.episode\_id = episodes.id

where sumtotnum =50 or sumtotnum =150 or sumtotnum =250;

*Output*

season, number\_in\_season, title, loc, raw\_text, beer/duff counter

4, 13, Selma's Choice, simpson home, Homer Simpson: Hi, kids. How was Duff Gardens?, 1

11, 5, E-I-E-I-(Annoyed Grunt), moe's tavern, Homer Simpson: (TO SELF, IN AWE) Wow, a free beer! And I owe it all to a little glove slap. (SINGING TO TUNE OF "LOVE SHACK") GLOVE SLAP..., 1

26, 5, Opposites A-Frack, skinner home, Homer Simpson: And all those new workers are gonna want to spend their earnings buying what you're selling: beer., 1

10.) (This is not a trivial query. Or, at least it wasn’t for me. Do not attempt to solve the problem all at once—start simple and build your way up to the solution, checking your output along the way.) Barney Gumble (character\_id=18) is a funny character on the show who is likely Homer Simpson’s best friend. Some have argued that Barney “always seems to be stuck in Homer’s shadow” and should be given more “airtime” independent of Homer. We will investigate this claim. Specifically, we want to count the number of Barney’s “appearances” with and without Homer (character\_id=2). However, a “Barney appearance” will not simply be defined as a record in “script\_lines” with character\_id=18—if he has multiple lines in an episode that are relatively close to one another we want to consider those lines as being part of the same “appearance.” Specifically, order your script\_lines data by “episode\_id” and then by “number” and consider each episode separately in your calculations. First, find all records credited to Barney (character\_id=18), including speaking and non-speaking lines (in what is shown below I denote this with “barney\_ind”). Then, consider a “Barney appearance” to consist of the three records before the first instance of barney\_ind=1, the three records after the last instance of barney\_ind=1, and all records in-between as long as there is no gap of 6 consecutive records or more that are not credited to Barney. So, let’s run through a couple of examples. First, suppose I have the following…

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| episode\_id | number | character\_id | barney\_ind |  |
| 114 | 0 | ***NULL*** | 0 |  |
| 114 | 1 | 165 | 0 | 3 records before first indicator |
| 114 | 2 | 17 | 0 |
| 114 | 3 | 17 | 0 |
| 114 | 4 | 18 | 1 | No gap of 6 or more consecutive zeros in-between |
| 114 | 5 | 17 | 0 |
| 114 | 6 | 17 | 0 |
| 114 | 7 | 17 | 0 |
| 114 | 8 | 2 | 0 |
| 114 | 9 | 17 | 0 |
| 114 | 10 | 18 | 1 |
| 114 | 11 | 17 | 0 |
| 114 | 12 | 165 | 0 |
| 114 | 13 | 18 | 1 |
| 114 | 14 | 17 | 0 | 3 records after last indicator |
| 114 | 15 | 17 | 0 |
| 114 | 16 | 2 | 0 |
| 114 | 17 | 17 | 0 |  |
| 114 | 18 | 17 | 0 |  |
| 114 | 19 | 2 | 0 |  |
| 114 | 20 | 17 | 0 |  |
| 114 | 21 | 14 | 0 |  |
| 114 | 22 | 2 | 0 |  |

The above yellow block of records would define one “Barney appearance.” It consists of the three records before the first “barney\_ind,” the three records after the last “barney\_ind,” and there is no gap of six or more consecutive records with “barney\_ind”=0 in between. Also note that since there is at least one instance of character\_id=2 (Homer) in this “Barney appearance,” this appearance would be counted as one “with Homer.” However, the situation below is a little different…

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| episode\_id | number | character\_id | barney\_ind |  |  |
| 1 | 267 | 31 | 0 |  |  |
| 1 | 268 | 22 | 0 |  |  |
| 1 | 269 | 10 | 0 |  |  |
| 1 | 270 | 22 | 0 | 3 records before first indicator |  |
| 1 | 271 | 1 | 0 |  |
| 1 | 272 | ***NULL*** | 0 |  |
| 1 | 273 | 18 | 1 |  |  |
| 1 | 274 | 2 | 0 | 3 records after last indicator | Gap of 6 consecutive zeros in-between |
| 1 | 275 | ***NULL*** | 0 |
| 1 | 276 | 24 | 0 |
| 1 | 277 | 32 | 0 | 3 records before first indicator |
| 1 | 278 | ***NULL*** | 0 |
| 1 | 279 | 2 | 0 |
| 1 | 280 | 18 | 1 |  |  |
| 1 | 281 | 2 | 0 | 3 records after last indicator |  |
| 1 | 282 | 8 | 0 |  |
| 1 | 283 | 2 | 0 |  |
| 1 | 284 | 33 | 0 |  |  |
| 1 | 285 | 2 | 0 |  |  |
| 1 | 286 | 8 | 0 |  |  |

Here, we have a gap of 6 records between two records with “barney\_ind”=1. So, the blue block of records would correspond with one “Barney appearance” and the yellow block of records would correspond with a second “Barney appearance.” Again, also note that Homer appears in both of these “Barney appearances.”

For simplicity, do not worry about checking the “location\_id” variable to potentially make sure that all of the dialogue is happening at the same place. Your final output should include two columns and two rows showing the count of “Barney appearances” with and without Homer.

*Query*

with t1 as

(select number, episode\_id,character\_id,

character\_id = 18 as 18id

FROM script\_lines

),

t2 as

(SELECT \*, row\_number()over(PARTITION BY episode\_id ORDER BY number) AS rownum

FROM t1

WHERE 18id = 1),

t3 as(

SELECT t1.\*,t2.rownum,

MAX(t2.rownum)OVER(PARTITION BY t1.episode\_id) AS maxrm

FROM t1

LEFT JOIN t2

ON t1.episode\_id = t2.episode\_id and t1.number = t2.number

ORDER BY episode\_id, number)

SELECT t1.\*,

SUM(rownum IS NULL) OVER(PARTITION BY episode\_id ORDER BY number rows between 1 FOLLOWING AND 3 FOLLOWING)

FROM t3

*Output*