ORIE 3120

Solution to Prelim 1

Given on March 5, 2019

Problem 1 [9 pts]

We have a database with two tables, T1 and T2. T1 has three columns (A, B, and C), and T2 has two columns (D and E).

T1

A	B	C
1	2	5
2	3	7
3	3	2
4	3	1
5	2	6

T2

D	E
1	5
2	2
3	7

For each valid foreign key relationship that may be created using this database, list in the space below the foreign key's table and field, and the referenced table and field. "Valid" means that SQLite would not generate an error if you specified this relationship. Do not include foreign key relationships from a field to itself. You do not need to fill in all of the rows in the space below.

Foreign Key Table	Foreign Key Field	Referenced Table	Referenced Field
T1	B	T2	D
T1	B	T1	A
T2	D	T1	A
T2	E	T1	C

Problem 2 [15 points]

We have a database with two tables, T1 and T2. T1 has two columns (A and B), and T2 has three columns (A, B, and C).

T1

A	B
1	apple
2	orange
3	pear
4	grape

T2

A	B	C
3	1	1
2	2	2
2	1	1
1	1	3
1	2	1
1	1	1

For each of the following queries, write the query result. Write your query result as a table, with the column names written as they would be by SQLite. For example, for the query SELECT * FROM T1, the answer would be:

A	B
1	apple
2	orange
3	pear
4	grape

Any NULLs in the response may be left as blanks.

(a) SELECT A, B, C FROM T2 WHERE A <> 2 OR B>1

A	B	C
3	1	1
2	2	2
1	1	3
1	2	1
1	1	1

(b) SELECT T1.A AS T1_A, T1.B AS T1_B, T2.A AS T2_A
FROM T1 LEFT JOIN T2 ON T1.A=T2.C
ORDER BY T1.A, T2.A

T1_A	T1_B	T2_A
1	apple	1
1	apple	1
1	apple	2
1	apple	3
2	orange	2
3	pear	1
4	grape	*NULL*

(c) SELECT A, B, SUM(C) FROM T2 GROUP BY 1, 2

A	B	SUM(C)
1	1	4
1	2	1
2	1	1
2	2	2
3	1	1

Problem 3 [20 pts]

Consider a database scheme with 4 tables: PurchaseOrders, Products, ProductCategory, and DistributionCenters. This database is used by a company to track how long it takes products to be processed (sorted) when they are shipped to one of three different distribution centers. You may assume that there are no NULLs in the database.

The Products table tells us the category ID for each of 10 different products. Its primary key is ProductID. It has a foreign key whose child is Products.ProductCategoryID and parent is ProductCategory.ID.

Products

ProductID	ProductCategoryID
1	A
2	A
3	A
4	A
5	A
6	B
7	B
8	B
9	B
10	B

The ProductCategory table tells us the English name of each of the ProductCategoryIDs from the Products table. Its primary key is ID.

ProductCategory

ID	Name
A	Furniture
B	Appliances

The DistributionCenters table tells us the English name of the city in which each of the three different distribution centers is located. Its primary key is DC.

DistributionCenters

DC	Name
1	Chicago
2	Los Angeles
3	New York

The PurchaseOrders table has a record for each time a particular product was shipped to a particular distribution center. One such record is called a “purchase order”, or “PO”. It gives a unique id (the PO_Num), the ProductID of the product being shipped (where ProductID appears in the Products table), the time in days that it took to process the shipment (SortTime), and the Distribution Center (DC) to which the product was shipped. Its primary key is PO_Num. It has a foreign key whose child is PurchaseOrders.ProductID and whose parent is Products.ProductID. It has another foreign key whose child is PurchaseOrders.DC and whose parent is DistributionCenters.DC.

PurchaseOrders

PO_Num	ProductID	SortTime	DC
392	1	4	1
393	5	15	2
394	9	6	2
395	2	9	3
396	2	12	1
397	2	5	1
398	5	5	1
399	4	4	1
400	6	11	3
401	2	2	1
402	9	4	1
403	8	3	1

We want to predict, for each distribution center and product category, of what the sort time will be. To do this, we write a sequence of queries. The results can come back in any order.

a) (5 points) Write a query or queries that will calculate, for each distribution center, the average sort time for all purchase orders arriving to that distribution center. For the mock data given, the result should look like the following. Please use the same field names in your query. Below, you may refer to these query results as a view with the name Q01.

DC	AvgSortTime
1	4.875
2	10.5
3	10.0

SELECT DC , AVG (SortTime) AS AvgSortTime FROM PurchaseOrders , Products WHERE PurchaseOrders . ProductID = Products . ProductID GROUP BY DC ORDER BY DC
WHERE can be substituted with properly executed JOIN

or

SELECT DC , AVG (SortTime) AS AvgSortTime FROM PurchaseOrders GROUP BY DC ORDER BY DC

(the ORDER BY DC is optional)

b) (5 points) Write a query or queries that will calculate, for each distribution center and product category, the average sort time for all purchase orders from that product category arriving to that distribution center. It should also calculate the number of purchase orders in that product category arriving to that distribution center. For the mock data given, the result should look like the following. Please use the same field names in your query. Below, you may refer to these query results as a view with the name Q02.

DC	ProductCategoryID	AvgSortTime	CountSortTime
1	A	5.333333333333333	6
1	B	3.5	2
2	A	15.0	1
2	B	6.0	1
3	A	9.0	1
3	B	11.0	1

SELECT DC, ProductCategoryID, AVG(SortTime) AS AvgSortTime, Count(SortTime) AS CountSortTime
FROM PurchaseOrders, Products
WHERE PurchaseOrders.ProductID = Products.ProductID
GROUP BY DC, ProductCategoryID
ORDER BY DC, ProductCategoryID

(the ORDER BY is optional)
WHERE can be substituted with properly executed JOIN

3cd Rubric:

-½ for ambiguous table names (i.e. not specifying which table a specific field is being pulled from)

-1 for doing an improper join (or missing a WHERE condition if pulling from 2 tables without a join).

c) (5 points) Join the queries you created in parts (a) and (b) to calculate, for every pair of DC and ProductCategoryID that appear in Q02, a calculated column called PredictedSortTime. If the number of purchases orders for a pair of DC and ProductCategoryID is strictly greater than 3, then PredictedSortTime is the AvgSortTime from Q02. If not, it is AvgSortTime from Q01. For the mock data given, the result should look like the following. Please use the same field names in your query. Below, you may refer to these query results as a view with the name Q03.

DC	ProductCategoryID	PredictedSortTime
1	A	5.333333333333333
1	B	4.875
2	A	10.5
2	B	10.5
3	A	10.0
3	B	10.0

```
SELECT Q02.DC AS DC, Q02.ProductCategoryID AS ProductCategoryID,
CASE WHEN Q02.CountSortTime > 3 THEN  Q02.AvgSortTime ELSE Q01.AvgSortTime END
AS PredictedSortTime
FROM Q01, Q02
WHERE Q01.DC = Q02.DC
ORDER BY DC, ProductCategoryID
```

(The ORDER BY is optional)

d) (5 points) Join Q03 with DistributionCenters and ProductCategory to report the same results as Q03, but with DC replaced by the English name of the distribution center, and the ProductCategoryID replaced by the English name of the product category. For the mock data given, the result should look like the following. Please use the same field names in your query.

```
SELECT
DistributionCenters.Name AS DistributionCenterName,
ProductCategory.Name AS ProductCategoryName,
Q03.PredictedSortTime AS PredictedSortTime
FROM Q03, DistributionCenters, ProductCategory
WHERE Q03.DC = DistributionCenters.DC AND Q03.ProductCaetgoryID = ProductCategory.ID
```

Problem 4 [20 pts]

An Ithaca-area girl scout troop is selling boxes of cookies over a weekend in March. The cookie supplier charges them $4 per box. The troop sells each box for $5. Boxes of cookies that are not sold by the end of the weekend are returned to the supplier who reimburses $3.50 for each. The troop leader estimates that demand for cookies will be normally distributed with a mean of 100 boxes and a standard deviation of 10 boxes.

a) (5 points) Find the optimal number of boxes of cookies to order. Round to the nearest integer.

The underage cost is c_u = $5-$4=$1.
The overage cost is c_o = $4-$3.50 = $0.50.
The newsvendor quantile is q = c_u / (c_o + c_u) = 1 / (1+.5) = 0.6666
Looking in the normal cdf table, the value of z with F(z) = q is 0.43
The optimal order quantity is 100+10*0.43 = 104.3.
Rounding to the nearest integer gives 104.

C_o = 1 point
C_u = 1 point
F = 1 point
Z score = 1 point
Q = 1 point

b) (5 points) Find the sum of the expected overage and underage costs. Round to the nearest cent.

E[G(104,D)] = 0.5 * ((104-100)*F(0.43) + 10*f(0.43)) + 1 * ((100-104)*(1-F(0.43)) + 10*f(0.43))
= $5.45

(One can also compute z here using the rounded Q, which gives z = (104-100)/10 = 0.4 instead of 0.43. This changes the answer to
E[G(104,D)] = 0.5 * ((104-100)*F(0.40) + 10*f(0.40)) + 1 * ((100-104)*(1-F(0.40)) + 10*f(0.40))
= 0.5 * ((104-100)*0.65542 + 10*0.36827) + 1 * ((100-104)*(1-0.65542) + 10*0.36827)
= $5.46. It is only different by 1 cent.)

Equation = 2 points
F value = 1 point
f value = 1 point
Total Value = 1 point

c) (5 points) Find the expected profit. Round to the nearest cent.

Overage/underage cost is the loss in profit compared with setting Q=D.
Indeed, this is how we calculated overage/underage cost:

- When Q < D (underage), our cost is $5-$4, which is the loss in profit relative to setting Q=D
- When Q > D (overage), our cost is $4-$3.50, which is the loss in profit relative to setting Q=D

If Q=D, then our profit is 100*($5-$4) = $100.
Thus, our expected profit is $100 - expected overage / underage cost = $100 - $5.45 = $94.55

-1 point = using 104 instead of 100
-1 point = using underage/overage cost in the wrong way
-1 point = forgetting order cost
-1 point = setup is correct, but value is wrong

d) (3 points) What is the probability that the troop will earn at least $100 in profit if it orders 100 boxes?
0.5

The answer is 0.5 because $100 in profit happens if (and only if) demand D is >= 100 boxes. (Assuming Q=100.)

P(D >= 100) is the probability that a normal random variable is greater than its mean, which is 0.5. To compute P(D >= 100) mechanically, you can do:

- D = 100 + 10 * Z where Z is Normal(0,1)
- P(D >= 100) = P(100 + 10*Z >= 100) = P(10*Z >= 0) = P(Z >= 0) = 1-F(0) = 1-0.5 = 0.5

Z=0 = 1 point
P(Z>=0) = 2 points

e) (2 points) How many boxes should the troop order to maximize the probability that they earn at least $90 in profit?
90

For there to be $90 in profit, we must sell at least 90 boxes. (This is because we can earn at most $1 of profit for each box of demand.)

Thus, for any Q, P($90 in profit) <= P(D >= 90).

If we stock 90 boxes, then we will get $90 in profit whenever D >= 90.

So, when Q=90, P($90 in profit) = P(D >= 90).

That means that Q=90 maximizes the probability that the troop earns at least $90 in profit.

2 points for correct answer

Problem 5 [10 pts]

We have a table Strings containing one field, A, with the following data:

A
aaaaaa
bbbbbb
cccccc
dddddd
abadab
acacac
adadad
bcbcbc
bdbdbd
cdcdcd
abcabc
aabbaa
aaccaa
aaddaa
bbaabb
bbccbb
bbddbb
ccaacc
ccbbcc
ccddcc

Write a query that produces the following query result. You do not need to reproduce the same order, but you do need to reproduce the field names and the field values.

A	F1	F2	F3	F4	F5	F6	F7
aaaaaa	a	aaaa	aaaaaa			AAAAAA	aaa
bbbbbb	b	bbbb	bbbbbb		bbbbbb	BBBBBB	bbb
cccccc	c	cccc	cccccc	cccccc	cccccc	CCCCCC	ccc
dddddd	d	dddd	dddddd	dddddd	dddddd	DDDDDD	ddd
ababab	a	baba	zzz		babab	ABABAB	aba
acacac	a	caca	acacac	acacac	cacac	ACACAC	aca
adadad	a	dada	adadad	adadad	dadad	ADADAD	ada
bcbcbc	b	cbcb	bcbcbc	bcbcbc	bcbcbc	BCBCBC	bcb
bdbdbd	b	dbdb	bdbdbd	bdbdbd	bdbdbd	BDBDBD	bdb
cdcdcd	c	dcdc	cdcdcd	cdcdcd	cdcdcd	CDCDCD	cdc
abcabc	a	bcab	zczc	abcabc	bcabc	ABCABC	cab
aabbaa	a	abba	azbaa		bbaa	AABBAA	bba
aaccaa	a	acca	aaccaa	aacc	ccaa	AACCAA	cca
aaddaa	a	adda	aaddaa	aadd	ddaa	AADDAA	dda
bbaabb	b	baab	bbazb		bbaabb	BBAABB	aab
bbccbb	b	bccb	bbccbb	bbcc	bbccbb	BBCCBB	ccb
bbddbb	b	bddb	bbddbb	bbdd	bbddbb	BBDDBB	ddb
ccaacc	c	caac	ccaacc	ccaacc	ccaacc	CCAACC	aac
ccbbcc	c	cbbc	ccbbcc	ccbbcc	ccbbcc	CCBBCC	bbc
ccddcc	c	cddc	ccddcc	ccddcc	ccddcc	CCDDCC	ddc

Write your query here:

```
SELECT A,
SUBSTR(A, 1, 1) AS F1,
SUBSTR(A, 2, 4) AS F2,
REPLACE(A, 'ab', 'z') AS F3,
RTRIM(A, 'ab') AS F4,
LTRIM(A, 'a') AS F5,
UPPER(A) AS F6,
SUBSTR(A, -4, 3) AS F7
FROM Strings
```

-1 for missing/incorrect column (-0.5 if student was close to the answer)

-Note: A maximum of one point was taken off on the whole question for off by one errors

-0.5 for incorrect/missing Select or From statements

-0.5 for incorrect column names

Problem 6 [15 pts]

Kaiser Permanente (a large health care provider) operates a warehouse for storing and distributing medical supplies in the San Francisco Bay Area. They are evaluating how frequently they buy sterile bandages for use in medical procedures. Their supplier charges them $10 per box of bandages ordered, as well as an additional $100 per order for shipping and processing. The lead time between placing an order and receiving it is 7 days. Demand for bandages from individual health care providers serviced from the warehouse is constant over time, and is 60 boxes per day. The holding cost for a box of bandages is $10 per year. Assume there are 365 days per year.

a) Find the optimal number of boxes of bandages in each order. Round to the nearest integer.

We use the EOQ formula:
Q* = sqrt(2*K*lambda / h) = sqrt((2*$100*60 boxes / day) / *($10 / year * 1 year / 365 days))
= sqrt(2*100*60*365 / 10))= sqrt(438000) = 661.815684311

661 or 662 boxes (662 boxes is preferred)

5 points correct answer
-1 not rounded

If incorrect answer,
+2 correct Q* equation
+2 variables plugged in correctly

b) Find the inventory level that would trigger a replenishment order.

60 boxes per day, over 7 days, means that we need a replenishment order at 420 boxes

5 points correct answer
-1 not rounded

If incorrect, partial credit awarded for correct R equation (+2) and plugging in variables correctly (+2)

c) Find the average order cost per year under the optimal order quantity, including both the cost of the bandages themselves and the shipping/processing costs. Round to the nearest cent.

G = K*lambda / Q + c*lambda + hQ/2
= ($100) * (60 boxes / day) * (365 days / year) / (662 boxes)

+ ($10 / box) * (60 boxes / day) * (365 days / year)
+ ($10 / year) * (662 boxes) /2
= $3308.157099698 + $219000 + $3310 = $225618.16

Not including hQ/2 also accepted.

5 points for correct answer (differences in rounding fine)

If incorrect,
5 points..
-2 points for each term forgotten in G equation
-1-2 for incorrect units (most commonly, forgetting to multiple one or multiple terms by 365)
-1 not rounding to nearest cent
-1 answer not given as numerical value

*carry over errors should not affect more than -1 point

Problem 7 [10 pts]

The table below describes requests for service from riders on a ridesharing platform on a single day in a single city. Each record in the table contains the time that the request was made in seconds since the start of the day, a unique id associated with that request (request_ID), and the latitude and longitude of the rider's origin. The table is called "trips".

Trips

request_ID	origin_lat	origin_lng	request_time
643219101	38.889931	-77.009003	40982.112
643219102	38.889939	-77.009104	40982.578
643219103	38.88996	-77.008901	40985.022
643219104	38.888721	-77.009523	41010.781
643219105	38.888531	-77.008923	41018.315

We refer to the number of seconds between a request and the next one in time as the *interarrival time*.

(a)[Write a query (or sequence of queries) that calculates the interarrival time for each request, except the last one in the dataset. You may assume that the request_time for each request is unique. Although the order of request_ID is the same as the order of request_time in the above dataset, do not assume this in your query. You may refer to your query below as a view called Q01. Here is what your query's result would look like on the data above:

[5 points]
+1 selecting correct 2 cols, have 4 rows, attempted the question
-1 if doesn't have min or subtracted the request times in wrong order
+1 if group by correct and where statement correct
-1 if don't get the min difference

request_ID	interarrival_time
643219101	0.466
643219102	2.444
643219103	25.759
643219104	7.534

```
SELECT T1.request_ID,
MIN(T2.request_time - T1.request_time) AS interarrival_time
FROM
trips as T1, trips as T2
where T2.request_time > T1.request_time
GROUP BY 1
```

(The actual query run to produce the screenshot above rounded the interarrival times, and also allowed request times to be non-unique, as shown below)

```
SELECT T1.request_ID,
ROUND(MIN(T2.request_time - T1.request_time),3) AS interarrival_time
FROM
trips as T1,
trips as T2
where T2.request_time >= T1.request_time
```

```
AND T2.request_ID <> T1.request_ID
GROUP BY 1
```

(b) Write a query or sequence of queries that calculates the average interarrival time in seconds, and the fraction of requests for which the interarrival time is strictly above 1 second. Your query should work regardless of the number of records in trips. Your query should produce the following result on the data above:

Avg_Interarrival_Time	Fraction_Above_1
9.05075	0.75

[5 points]
+1 for aggregation function/counting when >=1 or >1
+ 1 for right columns
+1 for dividing by right denominator
-1 if counted different columns

```
SELECT
AVG(interarrival_time) AS Avg_Interarrival_Time,
SUM(CASE WHEN interarrival_time >= 1 THEN 1 ELSE 0 END)*1.0 / COUNT(*) AS
Fraction_Above_1
FROM Q7
```