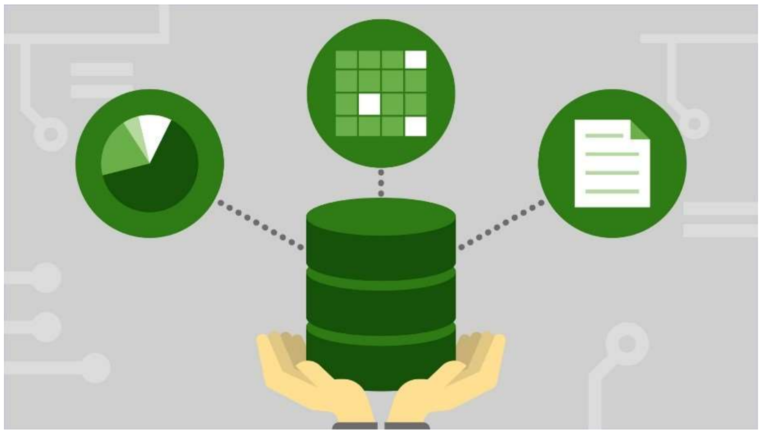
به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه پایگاه داده

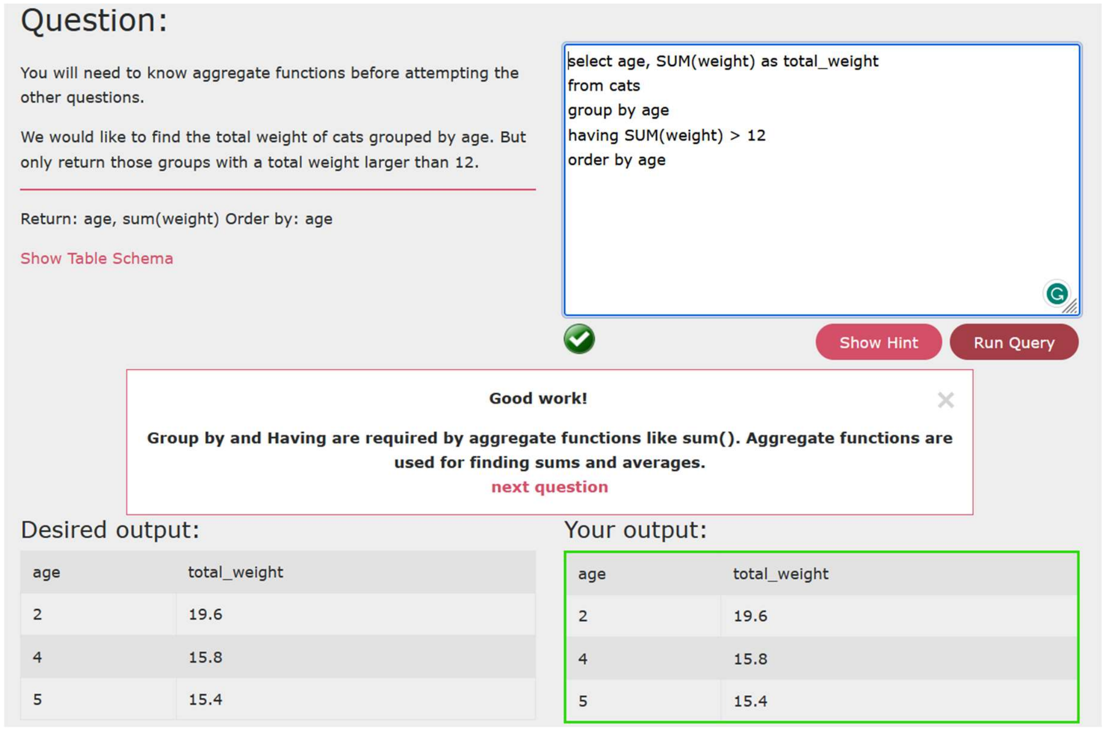
پیش‌گزارش شماره ۷

پرنیان فاضل
۸۱۰۱۹۸۵۱۶

بهار ۱۴۰۲

Refresher on Aggregates

به ازای هر دستورکار، توضیحات و پاسخ‌های خود را در این قسمت به صورت مختصر وارد کنید. با توجه به اینکه تمامی دستورکارهای آزمایشگاه، شامل تولید خروجی هستند، کافی است عکسی از خروجی هر مرحله از کار به همراه توضیح مختصر آنرا در این قسمت



Question:

You will need to know aggregate functions before attempting the other questions.

We would like to find the total weight of cats grouped by age. But only return those groups with a total weight larger than 12.

Return: age, sum(weight) Order by: age

Show Table Schema

```
select age, SUM(weight) as total_weight
from cats
group by age
having SUM(weight) > 12
order by age
```

Good work!

Group by and Having are required by aggregate functions like sum(). Aggregate functions are used for finding sums and averages.

next question

Desired output:

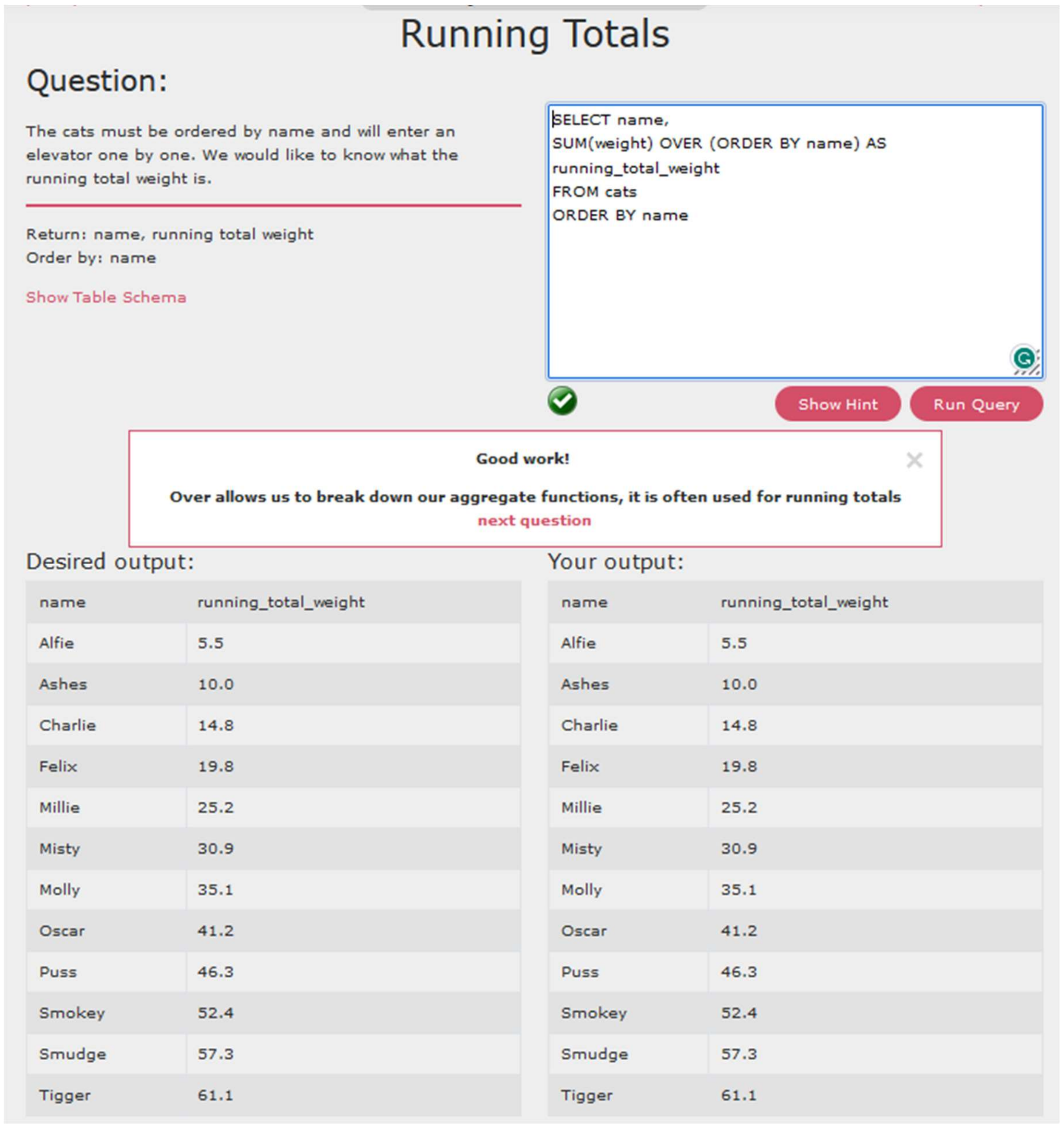
age	total_weight
2	19.6
4	15.8
5	15.4

Your output:

age	total_weight
2	19.6
4	15.8
5	15.4

This SQL query selects the age and the total weight of cats in the cats table and groups them by age. The "sum" function is used to calculate the total weight for each age group. The having clause is then applied to filter out only those age groups with a total weight greater than 12. Finally, the result is ordered by age in ascending order.

Running Totals


Running Totals

Question:

The cats must be ordered by name and will enter an elevator one by one. We would like to know what the running total weight is.

Return: name, running total weight
Order by: name

```
SELECT name,
SUM(weight) OVER (ORDER BY name) AS
running_total_weight
FROM cats
ORDER BY name
```

Good work!

Over allows us to break down our aggregate functions, it is often used for running totals

Desired output:

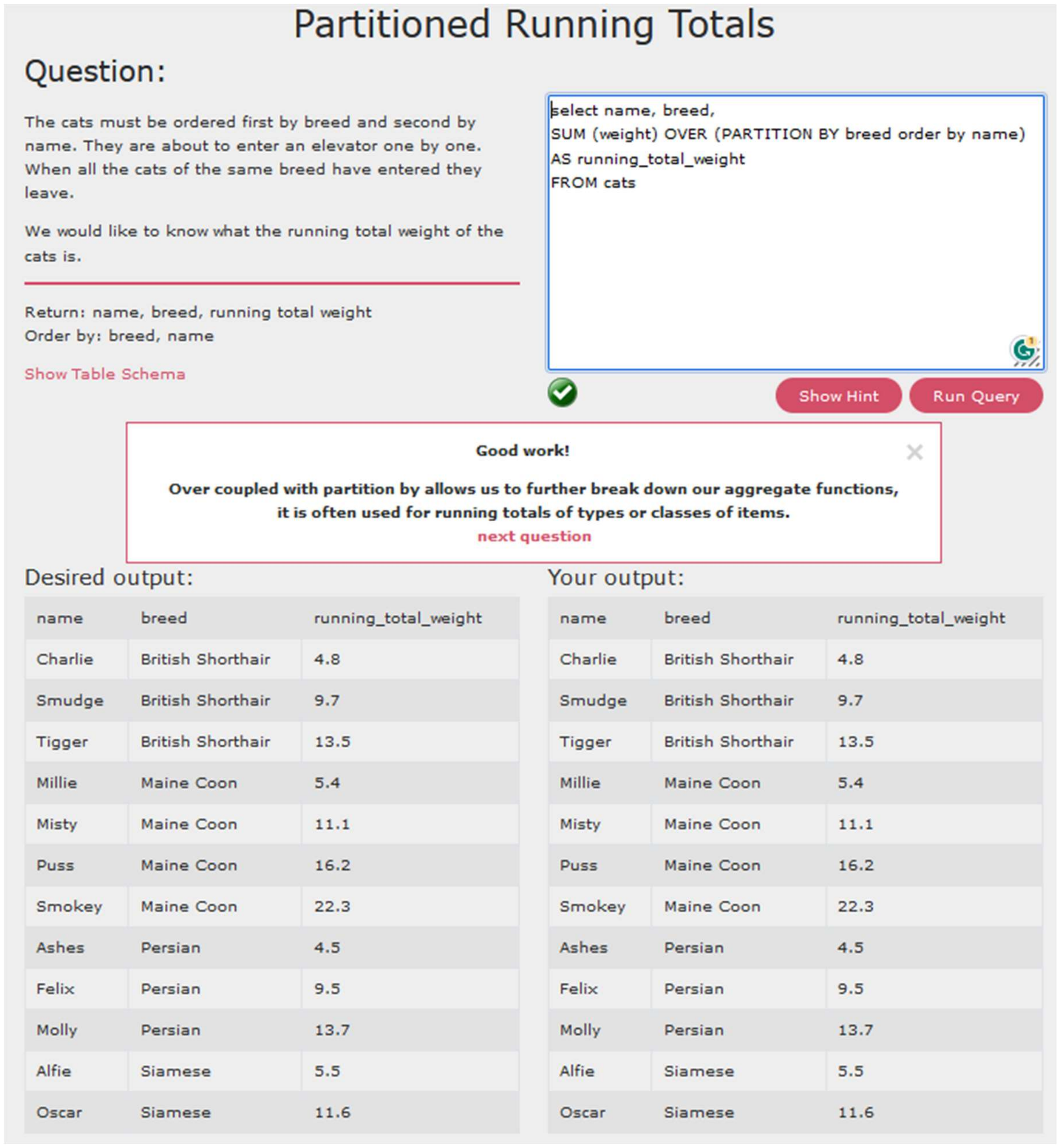
name	running_total_weight
Alfie	5.5
Ashes	10.0
Charlie	14.8
Felix	19.8
Millie	25.2
Misty	30.9
Molly	35.1
Oscar	41.2
Puss	46.3
Smokey	52.4
Smudge	57.3
Tigger	61.1

Your output:

name	running_total_weight
Alfie	5.5
Ashes	10.0
Charlie	14.8
Felix	19.8
Millie	25.2
Misty	30.9
Molly	35.1
Oscar	41.2
Puss	46.3
Smokey	52.4
Smudge	57.3
Tigger	61.1

The first query uses an ORDER BY clause inside the OVER() function to order the result set by the name column, and calculates a running total of the weight for each row based on that ordering. This means that the running_total_weight column will show the cumulative weight of all rows with names that come before or equal to the current row, sorted by name.

Partitioned Running Totals



Partitioned Running Totals

Question:

The cats must be ordered first by breed and second by name. They are about to enter an elevator one by one. When all the cats of the same breed have entered they leave.

We would like to know what the running total weight of the cats is.

Return: name, breed, running total weight
Order by: breed, name

Show Table Schema

```
select name, breed,
SUM (weight) OVER (PARTITION BY breed order by name)
AS running_total_weight
FROM cats
```

Good work!

Over coupled with partition by allows us to further break down our aggregate functions, it is often used for running totals of types or classes of items.

next question

Desired output:

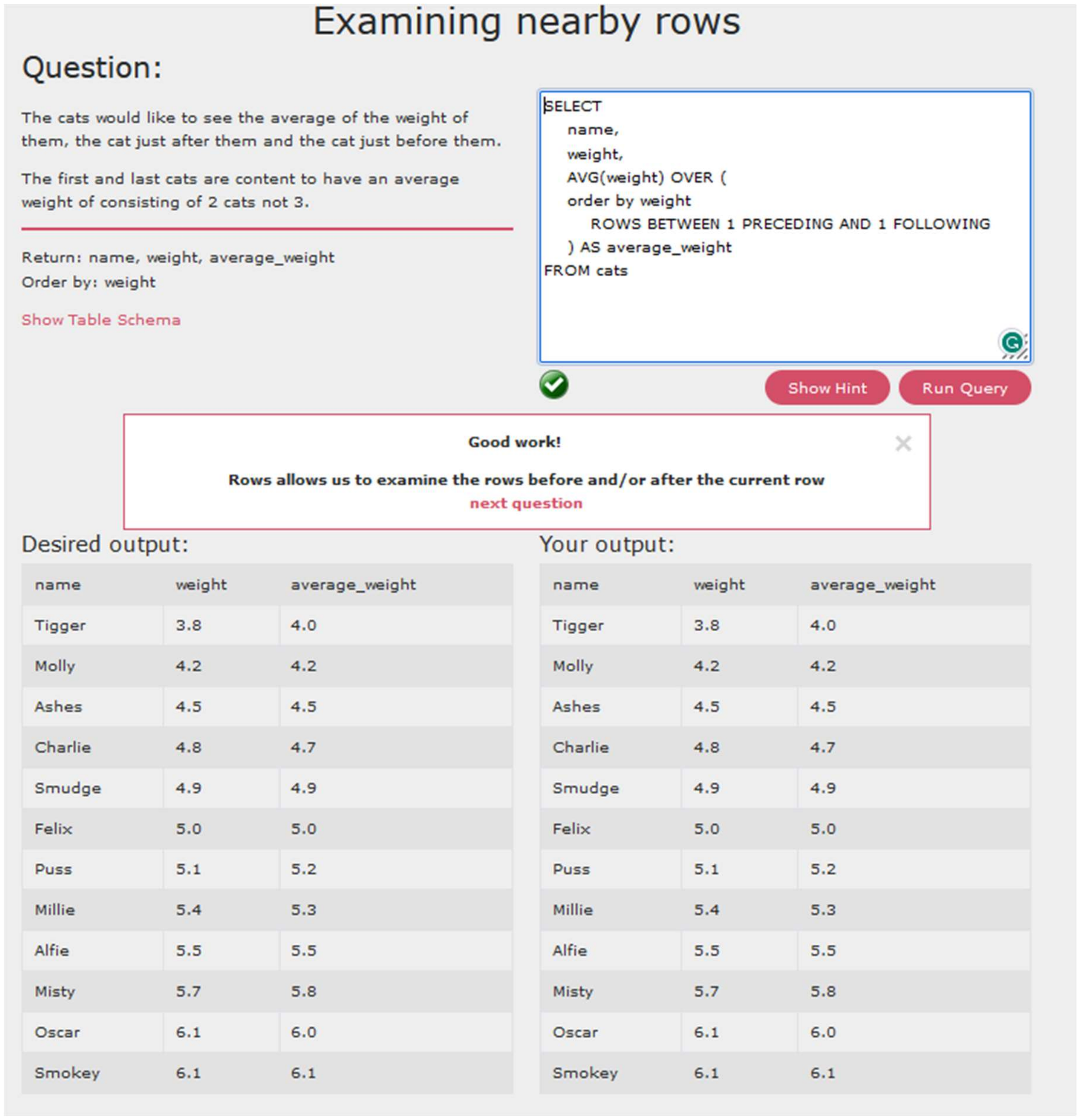
name	breed	running_total_weight
Charlie	British Shorthair	4.8
Smudge	British Shorthair	9.7
Tigger	British Shorthair	13.5
Millie	Maine Coon	5.4
Misty	Maine Coon	11.1
Puss	Maine Coon	16.2
Smokey	Maine Coon	22.3
Ashes	Persian	4.5
Felix	Persian	9.5
Molly	Persian	13.7
Alfie	Siamese	5.5
Oscar	Siamese	11.6

Your output:

name	breed	running_total_weight
Charlie	British Shorthair	4.8
Smudge	British Shorthair	9.7
Tigger	British Shorthair	13.5
Millie	Maine Coon	5.4
Misty	Maine Coon	11.1
Puss	Maine Coon	16.2
Smokey	Maine Coon	22.3
Ashes	Persian	4.5
Felix	Persian	9.5
Molly	Persian	13.7
Alfie	Siamese	5.5
Oscar	Siamese	11.6

The "running_total_weight" column shows the cumulative weight of all cats within the same breed that come before or equal to the current row, sorted by name. The PARTITION BY clause is used to group the data by breed, and the ORDER BY clause is used to order the rows within each breed by name.

Examining nearby rows



Examining nearby rows

Question:

The cats would like to see the average of the weight of them, the cat just after them and the cat just before them.

The first and last cats are content to have an average weight of consisting of 2 cats not 3.

Return: name, weight, average_weight
Order by: weight

Show Table Schema

```
SELECT
    name,
    weight,
    AVG(weight) OVER (
    order by weight
       ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
    ) AS average_weight
FROM cats
```

Good work!

Rows allows us to examine the rows before and/or after the current row

next question

Desired output:

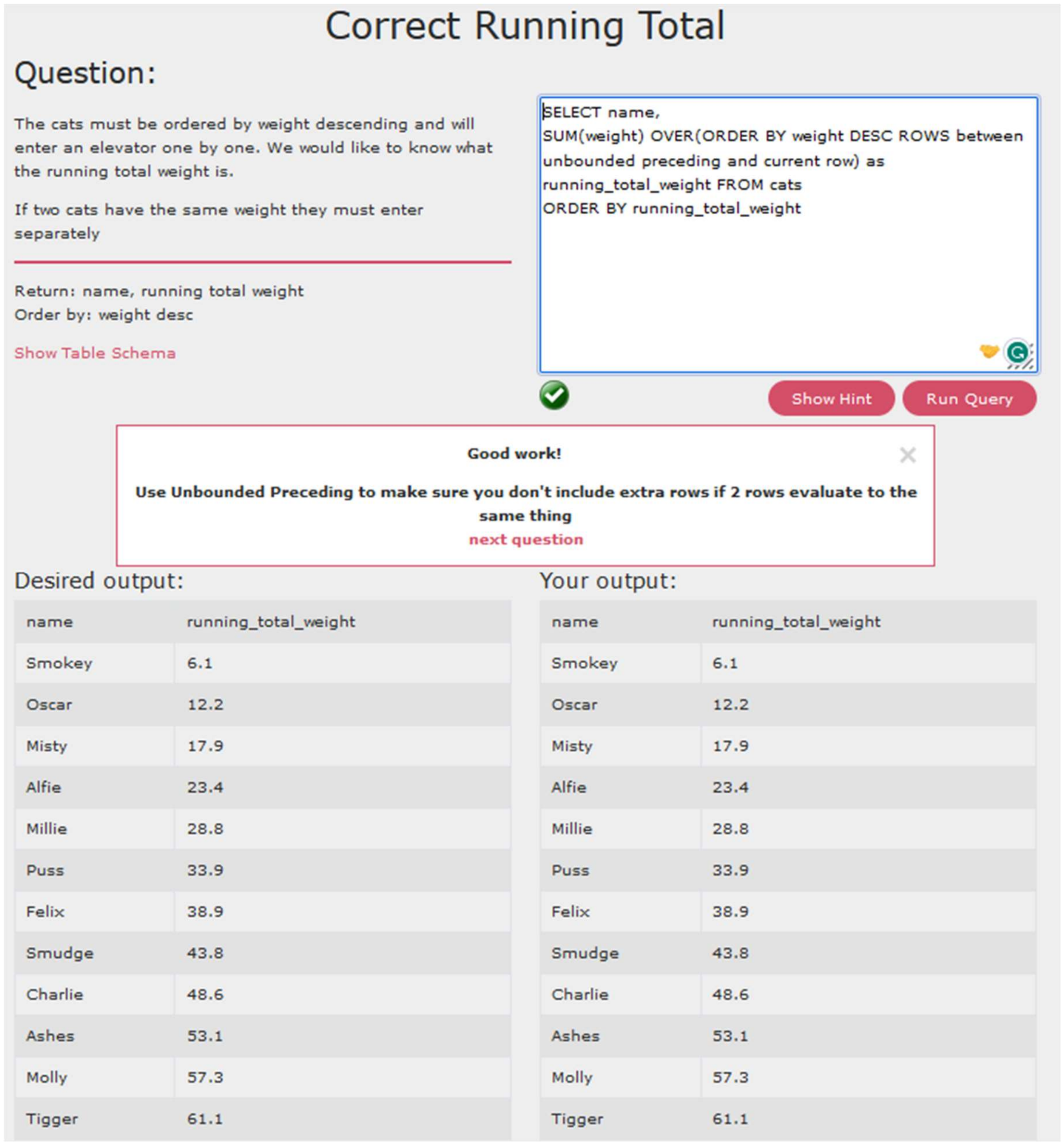
name	weight	average_weight
Tigger	3.8	4.0
Molly	4.2	4.2
Ashes	4.5	4.5
Charlie	4.8	4.7
Smudge	4.9	4.9
Felix	5.0	5.0
Puss	5.1	5.2
Millie	5.4	5.3
Alfie	5.5	5.5
Misty	5.7	5.8
Oscar	6.1	6.0
Smokey	6.1	6.1

Your output:

name	weight	average_weight
Tigger	3.8	4.0
Molly	4.2	4.2
Ashes	4.5	4.5
Charlie	4.8	4.7
Smudge	4.9	4.9
Felix	5.0	5.0
Puss	5.1	5.2
Millie	5.4	5.3
Alfie	5.5	5.5
Misty	5.7	5.8
Oscar	6.1	6.0
Smokey	6.1	6.1

This SQL query selects data from the "cats" table and calculates the average weight of each cat and the cats immediately before and after it, sorted by weight. The query returns the cat name, weight, and average weight columns.

Correct Running Total


Correct Running Total

Question:

The cats must be ordered by weight descending and will enter an elevator one by one. We would like to know what the running total weight is.

If two cats have the same weight they must enter separately

Return: name, running total weight
Order by: weight desc

Show Table Schema

```
SELECT name,
SUM(weight) OVER(ORDER BY weight DESC ROWS between
unbounded preceding and current row) as
running_total_weight FROM cats
ORDER BY running_total_weight
```

Good work!

Use Unbounded Preceding to make sure you don't include extra rows if 2 rows evaluate to the same thing

next question

Desired output:

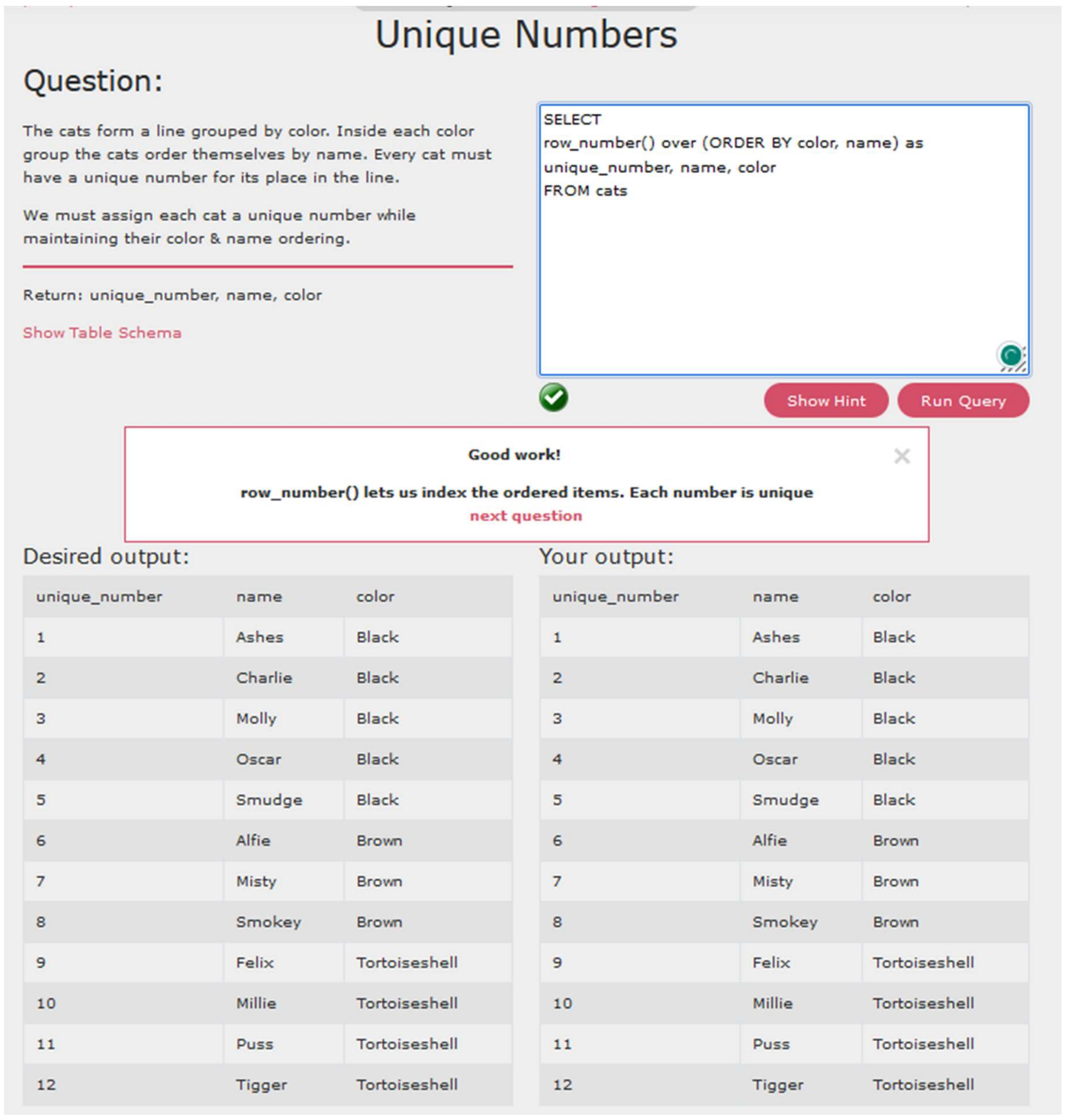
name	running_total_weight
Smokey	6.1
Oscar	12.2
Misty	17.9
Alfie	23.4
Millie	28.8
Puss	33.9
Felix	38.9
Smudge	43.8
Charlie	48.6
Ashes	53.1
Molly	57.3
Tigger	61.1

Your output:

name	running_total_weight
Smokey	6.1
Oscar	12.2
Misty	17.9
Alfie	23.4
Millie	28.8
Puss	33.9
Felix	38.9
Smudge	43.8
Charlie	48.6
Ashes	53.1
Molly	57.3
Tigger	61.1

The SUM() function is used with the OVER() clause to calculate the cumulative weight of all cats that come before or equal to the current row, sorted by the weight in descending order. The ROWS BETWEEN clause specifies that we want to include all rows from the beginning of the partition (unbounded preceding) to the current row in the calculation of the running total. Finally, the query sorts the results by the running total weight.

Unique Numbers



Unique Numbers

Question:

The cats form a line grouped by color. Inside each color group the cats order themselves by name. Every cat must have a unique number for its place in the line.

We must assign each cat a unique number while maintaining their color & name ordering.

Return: unique_number, name, color

```
SELECT
row_number() over (ORDER BY color, name) as
unique_number, name, color
FROM cats
```

Good work!

row_number() lets us index the ordered items. Each number is unique

next question

Desired output:

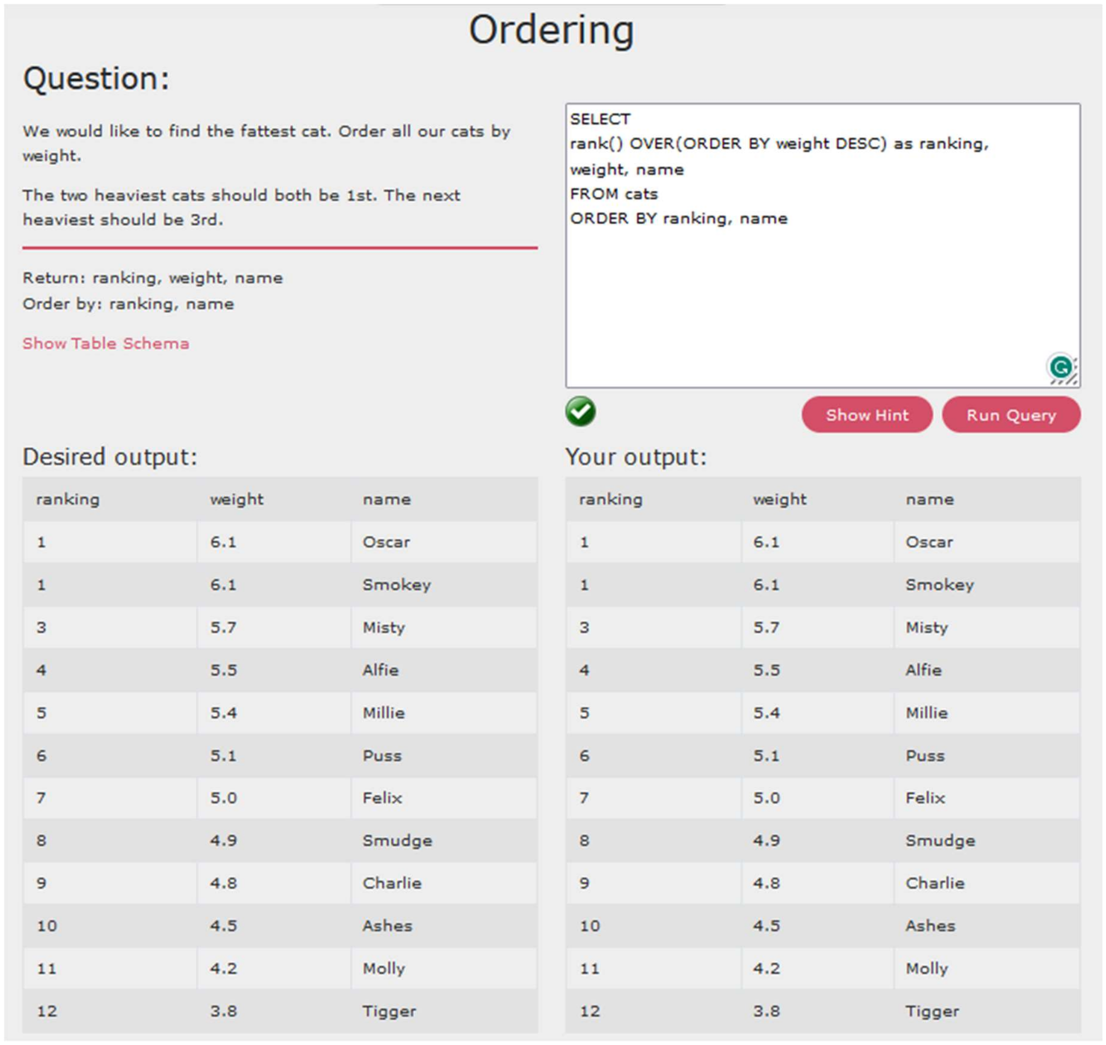
unique_number	name	color
1	Ashes	Black
2	Charlie	Black
3	Molly	Black
4	Oscar	Black
5	Smudge	Black
6	Alfie	Brown
7	Misty	Brown
8	Smokey	Brown
9	Felix	Tortoiseshell
10	Millie	Tortoiseshell
11	Puss	Tortoiseshell
12	Tigger	Tortoiseshell

Your output:

unique_number	name	color
1	Ashes	Black
2	Charlie	Black
3	Molly	Black
4	Oscar	Black
5	Smudge	Black
6	Alfie	Brown
7	Misty	Brown
8	Smokey	Brown
9	Felix	Tortoiseshell
10	Millie	Tortoiseshell
11	Puss	Tortoiseshell
12	Tigger	Tortoiseshell

This SQL query assigns a unique number to each row in the "cats" table, sorted by the cat's color and name. The "unique_number" column shows the unique number assigned to each row, the "name" column shows the name of each cat, and the "color" column shows the color of each cat. The row_number() function is used with the OVER() clause to assign a unique number to each row based on the sorting criteria specified in the ORDER BY clause.

Ordering



Ordering

Question:

We would like to find the fattest cat. Order all our cats by weight.

The two heaviest cats should both be 1st. The next heaviest should be 3rd.

Return: ranking, weight, name
Order by: ranking, name

Show Table Schema

```
SELECT
rank() OVER(ORDER BY weight DESC) as ranking,
weight, name
FROM cats
ORDER BY ranking, name
```

Desired output:

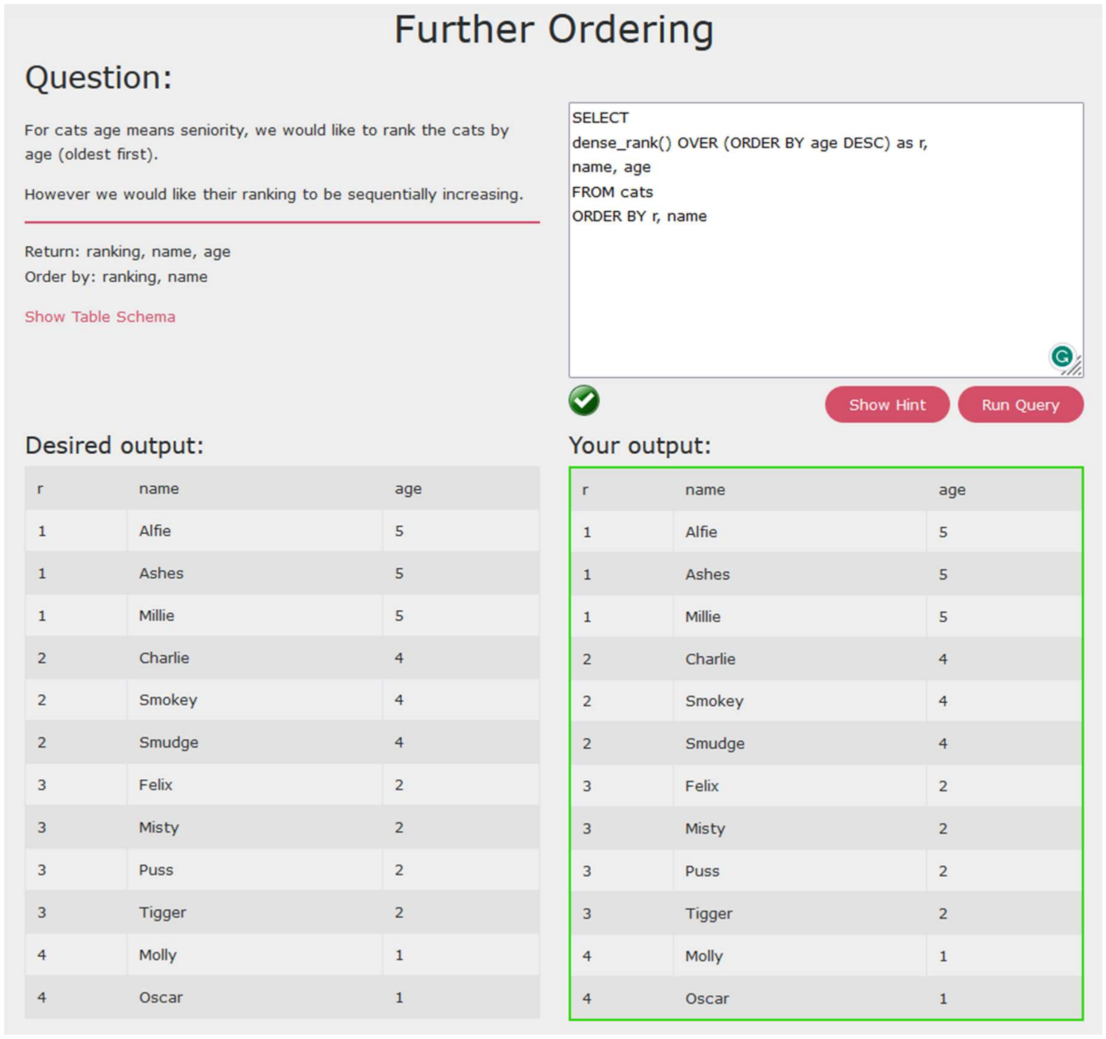
ranking	weight	name
1	6.1	Oscar
1	6.1	Smokey
3	5.7	Misty
4	5.5	Alfie
5	5.4	Millie
6	5.1	Puss
7	5.0	Felix
8	4.9	Smudge
9	4.8	Charlie
10	4.5	Ashes
11	4.2	Molly
12	3.8	Tigger

Your output:

ranking	weight	name
1	6.1	Oscar
1	6.1	Smokey
3	5.7	Misty
4	5.5	Alfie
5	5.4	Millie
6	5.1	Puss
7	5.0	Felix
8	4.9	Smudge
9	4.8	Charlie
10	4.5	Ashes
11	4.2	Molly
12	3.8	Tigger

This SQL query assigns a ranking to each cat based on their weight, sorted by the weight in descending order. "ranking" shows the rank of each cat based on their weight, "weight" shows the weight of each cat, and "name" shows the name of each cat.

Further Ordering



Further Ordering

Question:

For cats age means seniority, we would like to rank the cats by age (oldest first).

However we would like their ranking to be sequentially increasing.

Return: ranking, name, age
Order by: ranking, name

```
SELECT
dense_rank() OVER (ORDER BY age DESC) as r,
name, age
FROM cats
ORDER BY r, name
```

Desired output:

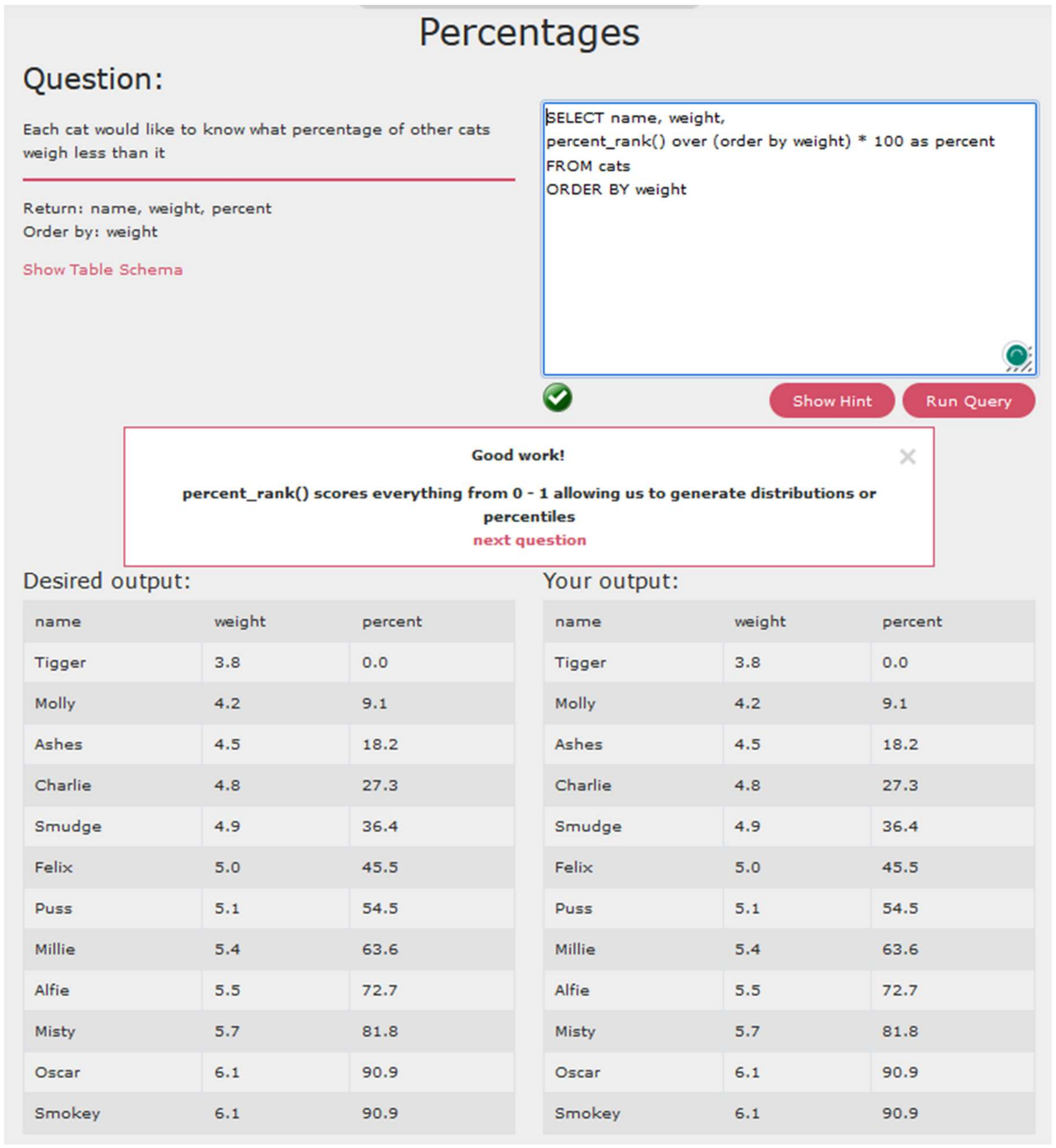
r	name	age
1	Alfie	5
1	Ashes	5
1	Millie	5
2	Charlie	4
2	Smokey	4
2	Smudge	4
3	Felix	2
3	Misty	2
3	Puss	2
3	Tigger	2
4	Molly	1
4	Oscar	1

Your output:

r	name	age
1	Alfie	5
1	Ashes	5
1	Millie	5
2	Charlie	4
2	Smokey	4
2	Smudge	4
3	Felix	2
3	Misty	2
3	Puss	2
3	Tigger	2
4	Molly	1
4	Oscar	1

This query assigns a dense rank to each cat based on their age in descending order, using the dense_rank() function with the OVER() clause. The query then selects the cat's name and age columns, along with the dense rank column aliased as "r". The results are then ordered by the dense rank column and then by the cat's name in ascending order.

Percentages


Percentages

Question:

Each cat would like to know what percentage of other cats weigh less than it

Return: name, weight, percent
Order by: weight

Show Table Schema

```
SELECT name, weight,
percent_rank() over (order by weight) * 100 as percent
FROM cats
ORDER BY weight
```

Good work!

percent_rank() scores everything from 0 - 1 allowing us to generate distributions or percentiles

next question

Desired output:

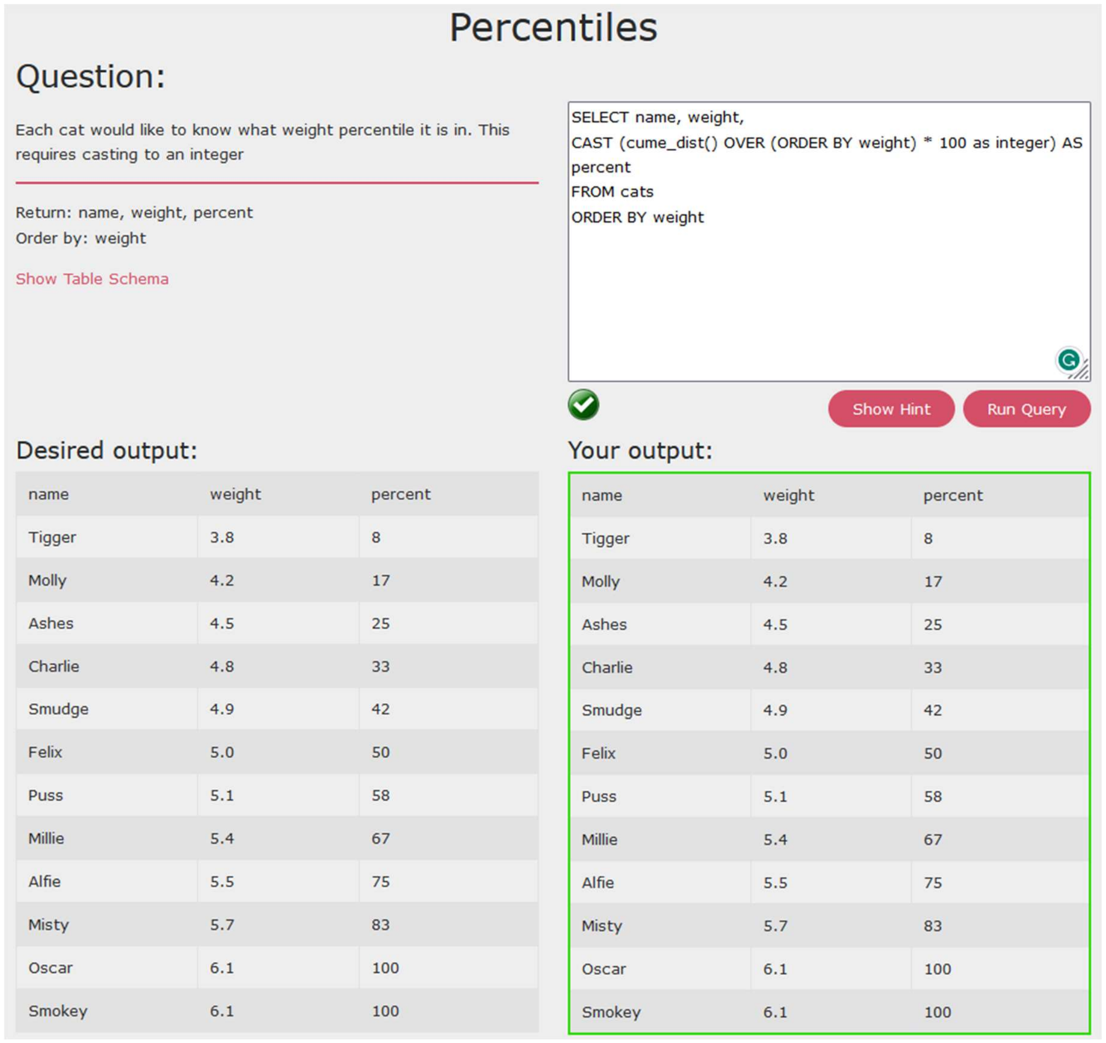
name	weight	percent
Tigger	3.8	0.0
Molly	4.2	9.1
Ashes	4.5	18.2
Charlie	4.8	27.3
Smudge	4.9	36.4
Felix	5.0	45.5
Puss	5.1	54.5
Millie	5.4	63.6
Alfie	5.5	72.7
Misty	5.7	81.8
Oscar	6.1	90.9
Smokey	6.1	90.9

Your output:

name	weight	percent
Tigger	3.8	0.0
Molly	4.2	9.1
Ashes	4.5	18.2
Charlie	4.8	27.3
Smudge	4.9	36.4
Felix	5.0	45.5
Puss	5.1	54.5
Millie	5.4	63.6
Alfie	5.5	72.7
Misty	5.7	81.8
Oscar	6.1	90.9
Smokey	6.1	90.9

This SQL query selects the name and weight of all cats in the "cats" table, and calculates the percentile rank of each cat's weight using the percent_rank() window function. The resulting percent rank is then multiplied by 100 to convert it to a percentage. The query then returns the name, weight, and percent rank of each cat, and orders the results by weight in ascending order.

Percentiles



Percentiles

Question:

Each cat would like to know what weight percentile it is in. This requires casting to an integer

Return: name, weight, percent
Order by: weight

Show Table Schema

```
SELECT name, weight,
CAST (cume_dist() OVER (ORDER BY weight) * 100 as integer) AS
percent
FROM cats
ORDER BY weight
```

Desired output:

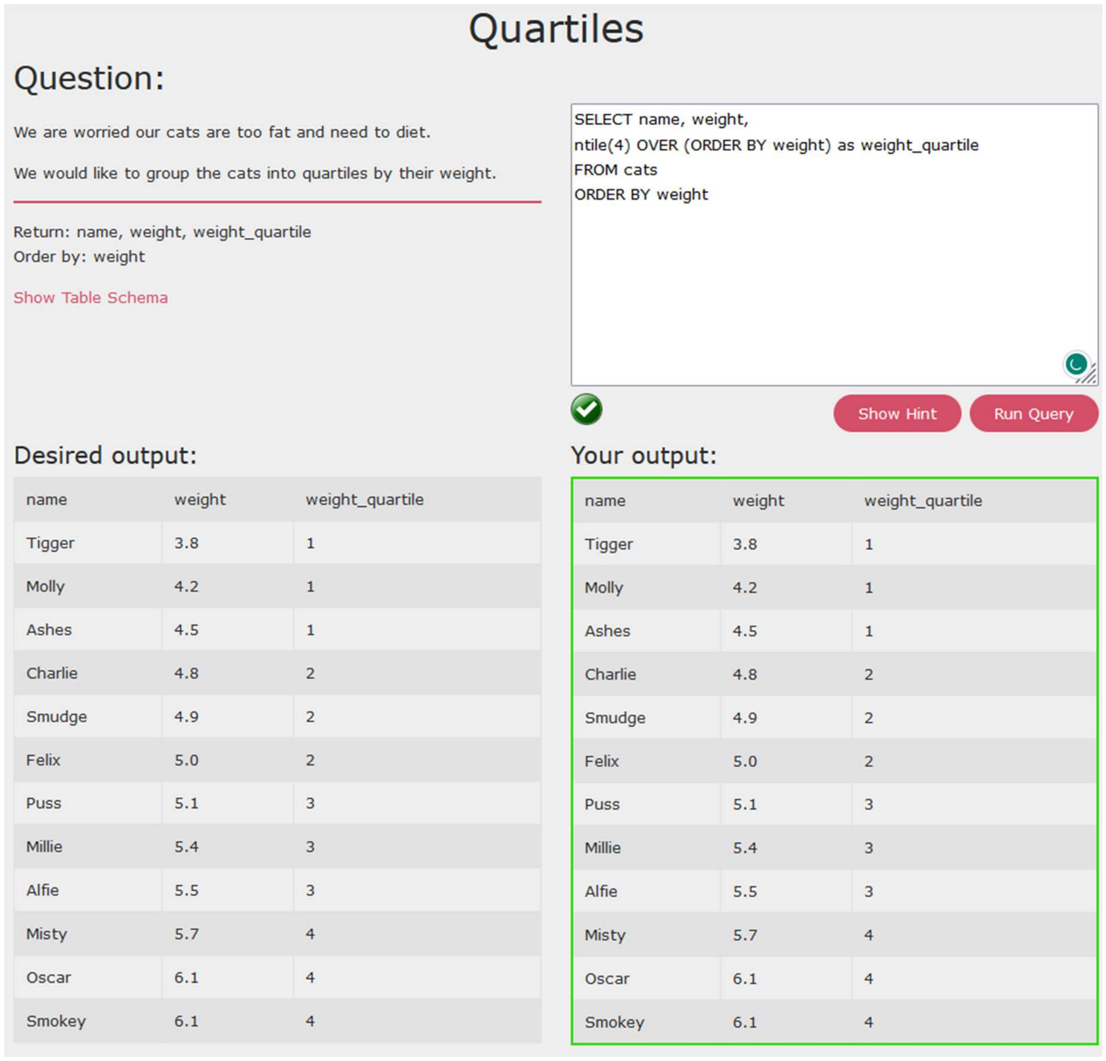
name	weight	percent
Tigger	3.8	8
Molly	4.2	17
Ashes	4.5	25
Charlie	4.8	33
Smudge	4.9	42
Felix	5.0	50
Puss	5.1	58
Millie	5.4	67
Alfie	5.5	75
Misty	5.7	83
Oscar	6.1	100
Smokey	6.1	100

Your output:

name	weight	percent
Tigger	3.8	8
Molly	4.2	17
Ashes	4.5	25
Charlie	4.8	33
Smudge	4.9	42
Felix	5.0	50
Puss	5.1	58
Millie	5.4	67
Alfie	5.5	75
Misty	5.7	83
Oscar	6.1	100
Smokey	6.1	100

This query selects the cat's name, weight, and the cumulative distribution of each cat's weight relative to the weights of all cats, using the cume_dist() function with the OVER() clause. The cumulative distribution is then multiplied by 100, cast to an integer, and aliased as "percent". The results are then ordered by the weight column in ascending order.

Quartiles



Quartiles

Question:

We are worried our cats are too fat and need to diet.

We would like to group the cats into quartiles by their weight.

Return: name, weight, weight_quartile
Order by: weight

Show Table Schema

```
SELECT name, weight,
ntile(4) OVER (ORDER BY weight) as weight_quartile
FROM cats
ORDER BY weight
```

Desired output:

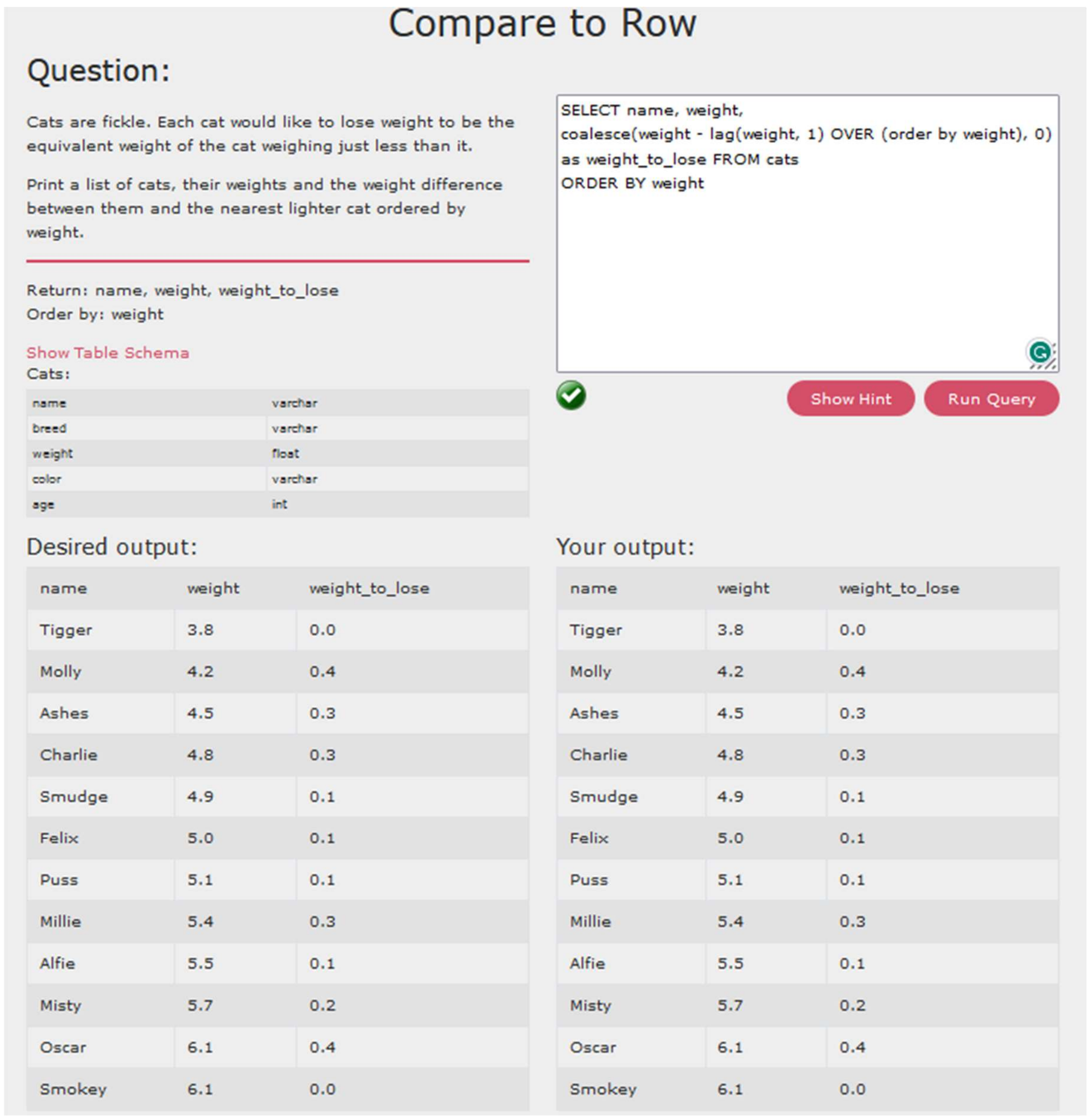
name	weight	weight_quartile
Tigger	3.8	1
Molly	4.2	1
Ashes	4.5	1
Charlie	4.8	2
Smudge	4.9	2
Felix	5.0	2
Puss	5.1	3
Millie	5.4	3
Alfie	5.5	3
Misty	5.7	4
Oscar	6.1	4
Smokey	6.1	4

Your output:

name	weight	weight_quartile
Tigger	3.8	1
Molly	4.2	1
Ashes	4.5	1
Charlie	4.8	2
Smudge	4.9	2
Felix	5.0	2
Puss	5.1	3
Millie	5.4	3
Alfie	5.5	3
Misty	5.7	4
Oscar	6.1	4
Smokey	6.1	4

The query selects the name and weight columns from the "cats" table and assigns each row a tile number based on the weight column divided into four equal groups (quartiles). This is done using the ntile window function with an argument of 4 and ordered by the weight column. The result set is then ordered by the weight column.

Compare to Row



Compare to Row

Question:

Cats are fickle. Each cat would like to lose weight to be the equivalent weight of the cat weighing just less than it.

Print a list of cats, their weights and the weight difference between them and the nearest lighter cat ordered by weight.

Return: name, weight, weight_to_lose
Order by: weight

Show Table Schema
Cats:

name	varchar
breed	varchar
weight	float
color	varchar
age	int

```
SELECT name, weight,
coalesce(weight - lag(weight, 1) OVER (order by weight), 0)
as weight_to_lose FROM cats
ORDER BY weight
```

Desired output:

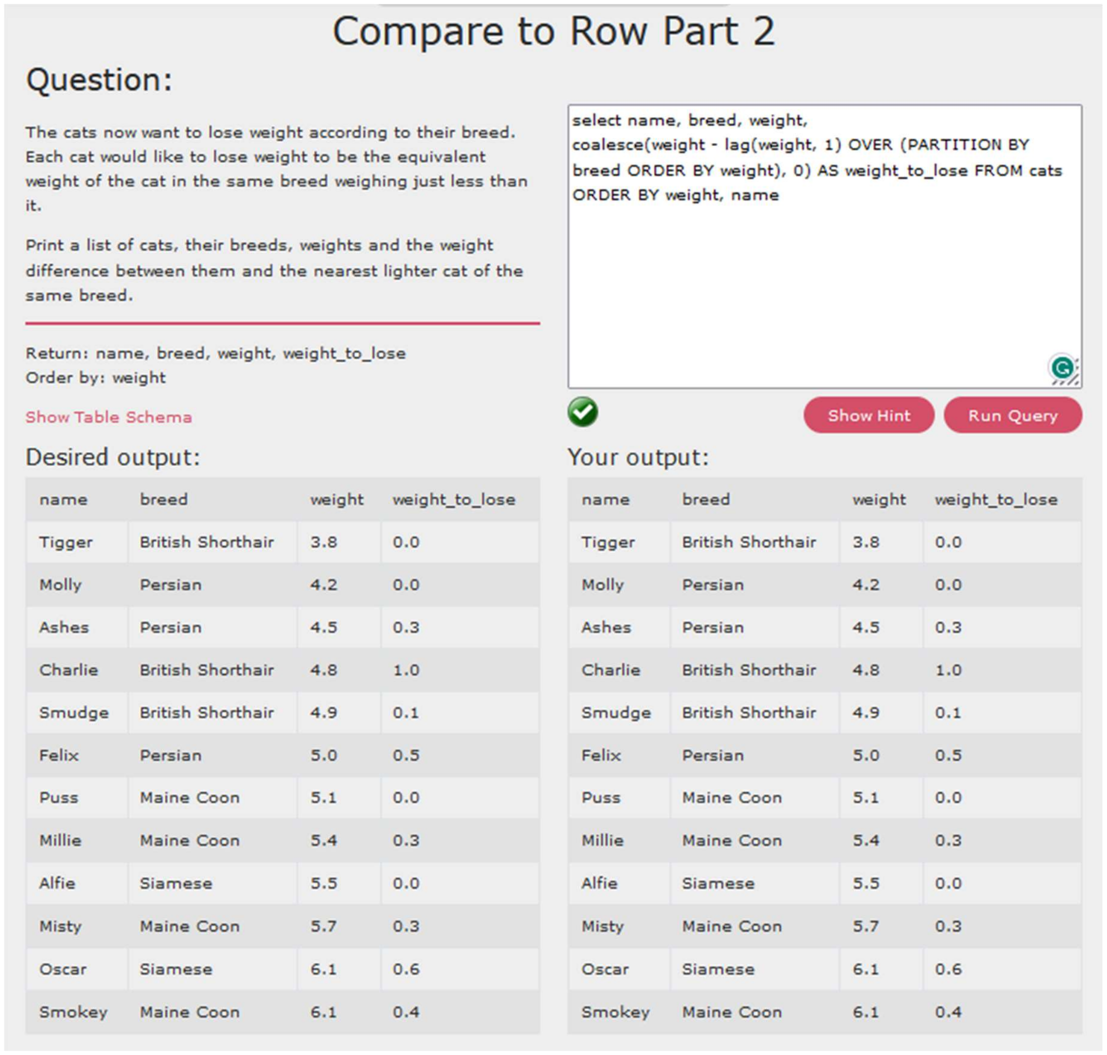
name	weight	weight_to_lose
Tigger	3.8	0.0
Molly	4.2	0.4
Ashes	4.5	0.3
Charlie	4.8	0.3
Smudge	4.9	0.1
Felix	5.0	0.1
Puss	5.1	0.1
Millie	5.4	0.3
Alfie	5.5	0.1
Misty	5.7	0.2
Oscar	6.1	0.4
Smokey	6.1	0.0

Your output:

name	weight	weight_to_lose
Tigger	3.8	0.0
Molly	4.2	0.4
Ashes	4.5	0.3
Charlie	4.8	0.3
Smudge	4.9	0.1
Felix	5.0	0.1
Puss	5.1	0.1
Millie	5.4	0.3
Alfie	5.5	0.1
Misty	5.7	0.2
Oscar	6.1	0.4
Smokey	6.1	0.0

The query selects the name and weight columns from the "cats" table, and calculates the weight difference between the current row and the previous row using the lag window function ordered by the weight column. The coalesce function is used to handle the first row where there is no previous row to calculate the difference with, in which case it returns 0. LAG(weight, 1) OVER (ORDER BY weight) retrieves the weight of the previous row, ordered by the weight column. The COALESCE function is used to return the weight difference between the current row and the previous row. If the previous row does not exist (i.e., for the first row), the LAG function returns NULL, so the COALESCE function returns 0 instead of NULL. Therefore, coalesce(weight - lag(weight, 1) OVER (order by weight), 0) calculates the weight difference between the current and previous row's weight, and returns 0 for the first row.

Compare to Row Part 2



Compare to Row Part 2

Question:

The cats now want to lose weight according to their breed. Each cat would like to lose weight to be the equivalent weight of the cat in the same breed weighing just less than it.

Print a list of cats, their breeds, weights and the weight difference between them and the nearest lighter cat of the same breed.

Return: name, breed, weight, weight_to_lose
Order by: weight

```
select name, breed, weight,
coalesce(weight - lag(weight, 1) OVER (PARTITION BY
breed ORDER BY weight), 0) AS weight_to_lose FROM cats
ORDER BY weight, name
```

Desired output:

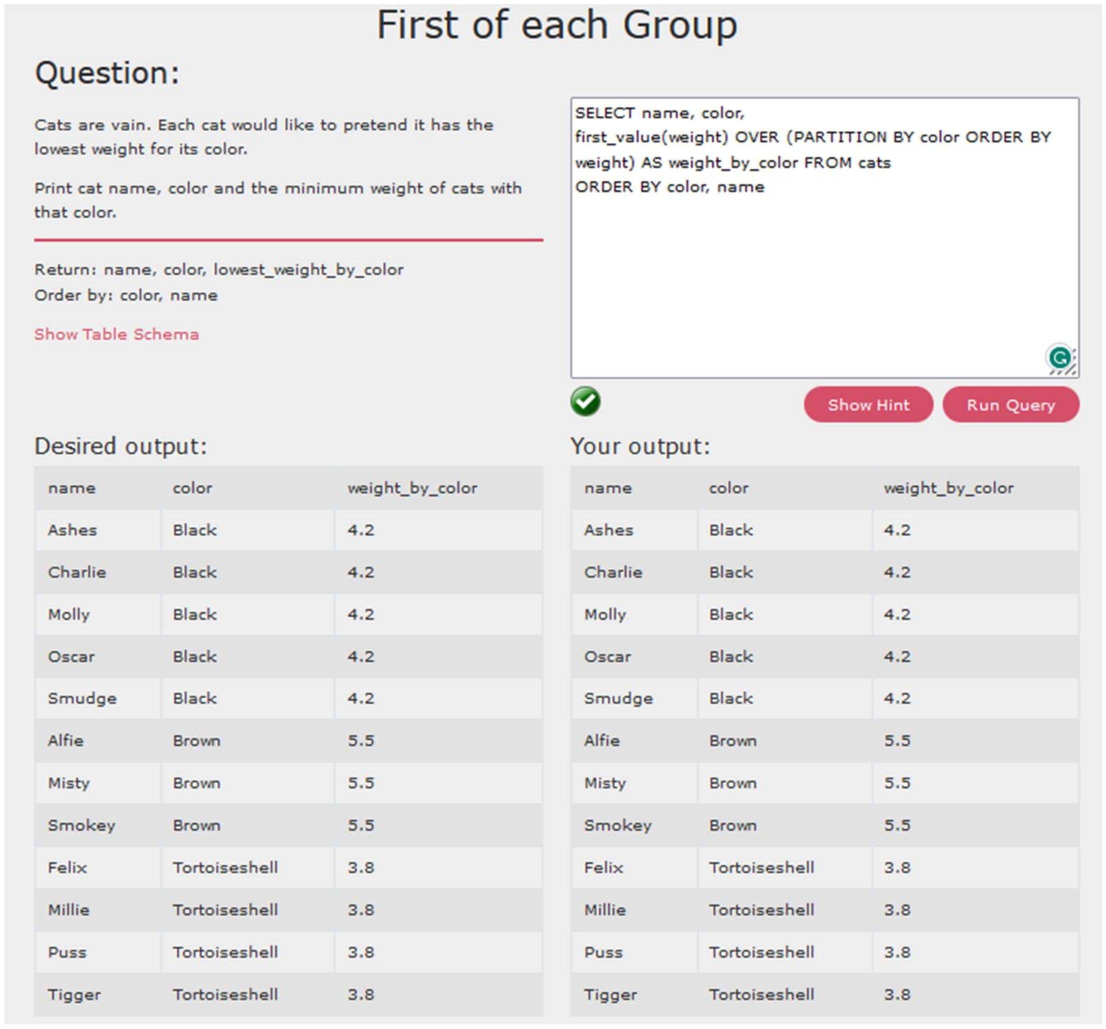
name	breed	weight	weight_to_lose
Tigger	British Shorthair	3.8	0.0
Molly	Persian	4.2	0.0
Ashes	Persian	4.5	0.3
Charlie	British Shorthair	4.8	1.0
Smudge	British Shorthair	4.9	0.1
Felix	Persian	5.0	0.5
Puss	Maine Coon	5.1	0.0
Millie	Maine Coon	5.4	0.3
Alfie	Siamese	5.5	0.0
Misty	Maine Coon	5.7	0.3
Oscar	Siamese	6.1	0.6
Smokey	Maine Coon	6.1	0.4

Your output:

name	breed	weight	weight_to_lose
Tigger	British Shorthair	3.8	0.0
Molly	Persian	4.2	0.0
Ashes	Persian	4.5	0.3
Charlie	British Shorthair	4.8	1.0
Smudge	British Shorthair	4.9	0.1
Felix	Persian	5.0	0.5
Puss	Maine Coon	5.1	0.0
Millie	Maine Coon	5.4	0.3
Alfie	Siamese	5.5	0.0
Misty	Maine Coon	5.7	0.3
Oscar	Siamese	6.1	0.6
Smokey	Maine Coon	6.1	0.4

The weight_to_lose column is calculated using the coalesce function and the lag function, which looks up the previous row's weight in the same breed group and subtracts it from the current weight. If there is no previous row, the coalesce function replaces the null value with 0. The resulting data is sorted by weight and name in ascending order.

First of each Group



First of each Group

Question:

Cats are vain. Each cat would like to pretend it has the lowest weight for its color.

Print cat name, color and the minimum weight of cats with that color.

Return: name, color, lowest_weight_by_color
Order by: color, name

Show Table Schema

```
SELECT name, color,
first_value(weight) OVER (PARTITION BY color ORDER BY
weight) AS weight_by_color FROM cats
ORDER BY color, name
```

Desired output:

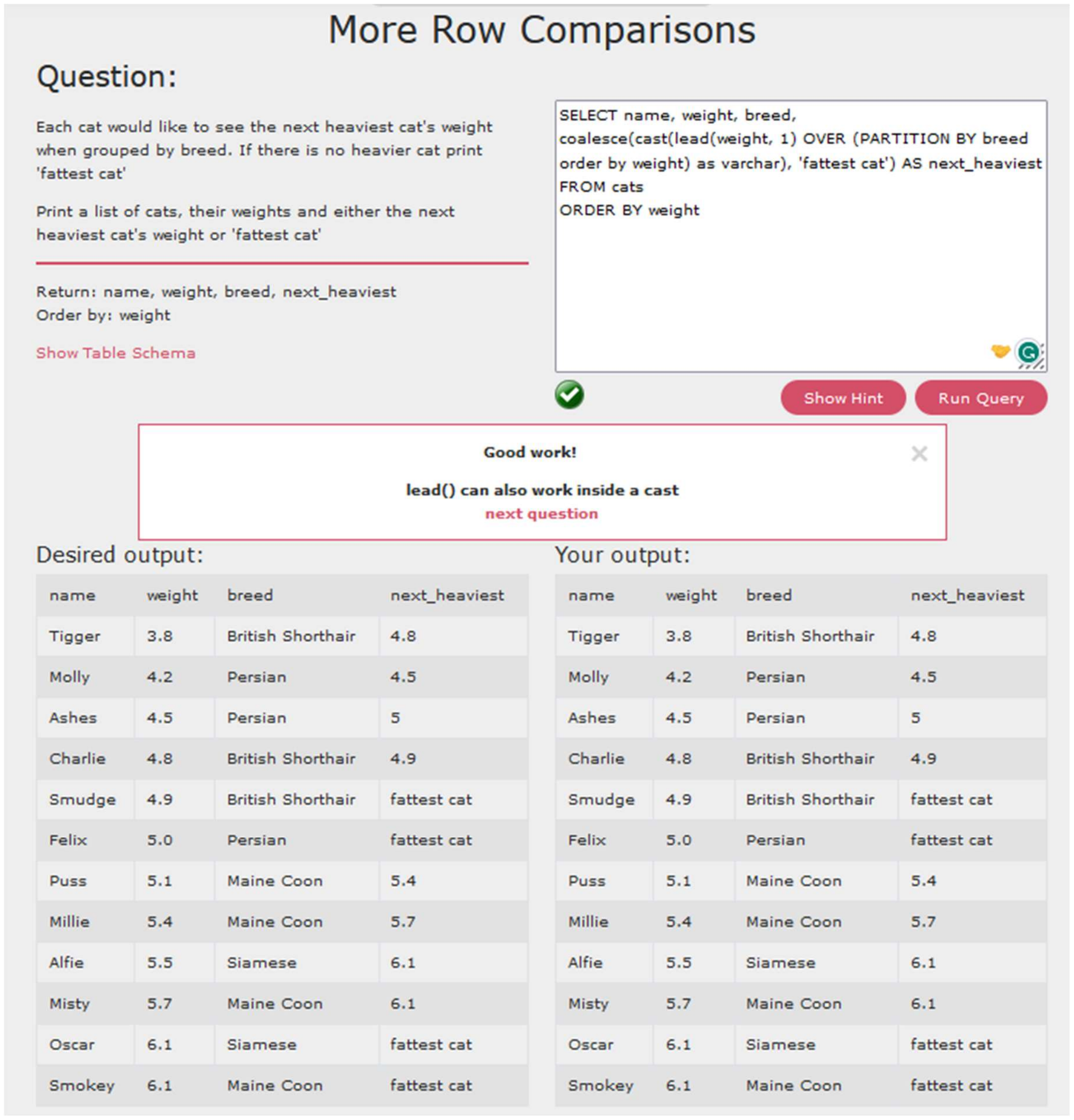
name	color	weight_by_color
Ashes	Black	4.2
Charlie	Black	4.2
Molly	Black	4.2
Oscar	Black	4.2
Smudge	Black	4.2
Alfie	Brown	5.5
Misty	Brown	5.5
Smokey	Brown	5.5
Felix	Tortoiseshell	3.8
Millie	Tortoiseshell	3.8
Puss	Tortoiseshell	3.8
Tigger	Tortoiseshell	3.8

Your output:

name	color	weight_by_color
Ashes	Black	4.2
Charlie	Black	4.2
Molly	Black	4.2
Oscar	Black	4.2
Smudge	Black	4.2
Alfie	Brown	5.5
Misty	Brown	5.5
Smokey	Brown	5.5
Felix	Tortoiseshell	3.8
Millie	Tortoiseshell	3.8
Puss	Tortoiseshell	3.8
Tigger	Tortoiseshell	3.8

It uses the "first_value" function to retrieve the first value of weight for each color partition, and "partition by" to specify that it should be computed over each color group. The result set is sorted first by color and then by name.

More Row Comparisons



More Row Comparisons

Question:

Each cat would like to see the next heaviest cat's weight when grouped by breed. If there is no heavier cat print 'fattest cat'

Print a list of cats, their weights and either the next heaviest cat's weight or 'fattest cat'

Return: name, weight, breed, next_heaviest
Order by: weight

```
SELECT name, weight, breed,
coalesce(cast(lead(weight, 1) OVER (PARTITION BY breed
order by weight) as varchar), 'fattest cat') AS next_heaviest
FROM cats
ORDER BY weight
```

Good work!

lead() can also work inside a cast

Desired output:

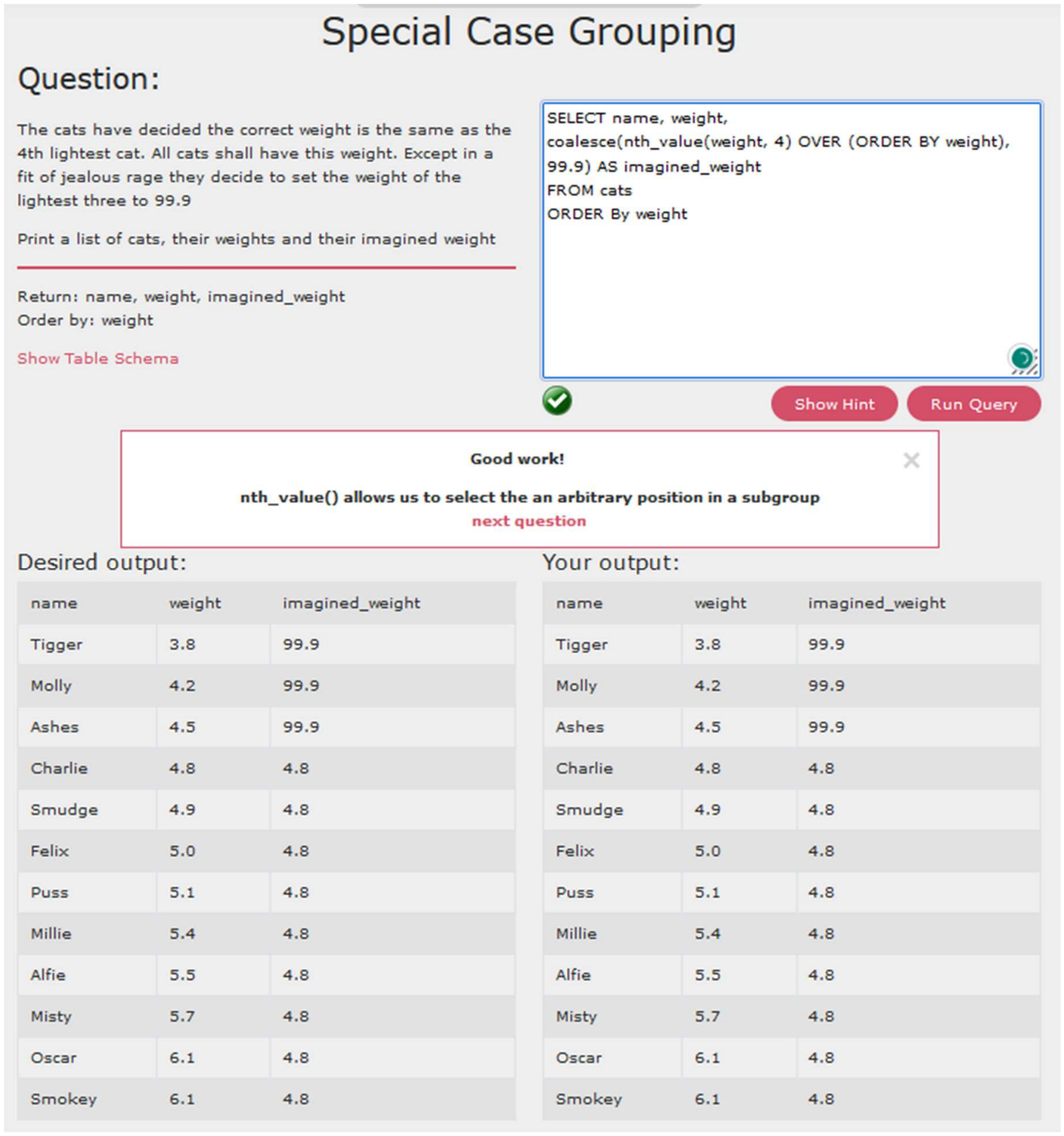
name	weight	breed	next_heaviest
Tigger	3.8	British Shorthair	4.8
Molly	4.2	Persian	4.5
Ashes	4.5	Persian	5
Charlie	4.8	British Shorthair	4.9
Smudge	4.9	British Shorthair	fattest cat
Felix	5.0	Persian	fattest cat
Puss	5.1	Maine Coon	5.4
Millie	5.4	Maine Coon	5.7
Alfie	5.5	Siamese	6.1
Misty	5.7	Maine Coon	6.1
Oscar	6.1	Siamese	fattest cat
Smokey	6.1	Maine Coon	fattest cat

Your output:

name	weight	breed	next_heaviest
Tigger	3.8	British Shorthair	4.8
Molly	4.2	Persian	4.5
Ashes	4.5	Persian	5
Charlie	4.8	British Shorthair	4.9
Smudge	4.9	British Shorthair	fattest cat
Felix	5.0	Persian	fattest cat
Puss	5.1	Maine Coon	5.4
Millie	5.4	Maine Coon	5.7
Alfie	5.5	Siamese	6.1
Misty	5.7	Maine Coon	6.1
Oscar	6.1	Siamese	fattest cat
Smokey	6.1	Maine Coon	fattest cat

This SQL query selects the name, weight, and breed of cats, and uses the lead function to get the weight of the next heaviest cat within the same breed. The coalesce function is used to handle the case where there is no next heaviest cat within the breed, in which case the string 'fattest cat' is used instead. LEAD() function, which returns the value of a specified column from the next row within the same partition based on the specified order. In this case, the LEAD() function is applied to the weight column, and it looks for the weight of the next heaviest cat within the same breed partition.

Special Case Grouping



Special Case Grouping

Question:

The cats have decided the correct weight is the same as the 4th lightest cat. All cats shall have this weight. Except in a fit of jealous rage they decide to set the weight of the lightest three to 99.9

Print a list of cats, their weights and their imagined weight

Return: name, weight, imagined_weight
Order by: weight

Show Table Schema

```
SELECT name, weight,
coalesce(nth_value(weight, 4) OVER (ORDER BY weight),
99.9) AS imagined_weight
FROM cats
ORDER By weight
```

Good work!

nth_value() allows us to select the an arbitrary position in a subgroup

next question

Desired output:

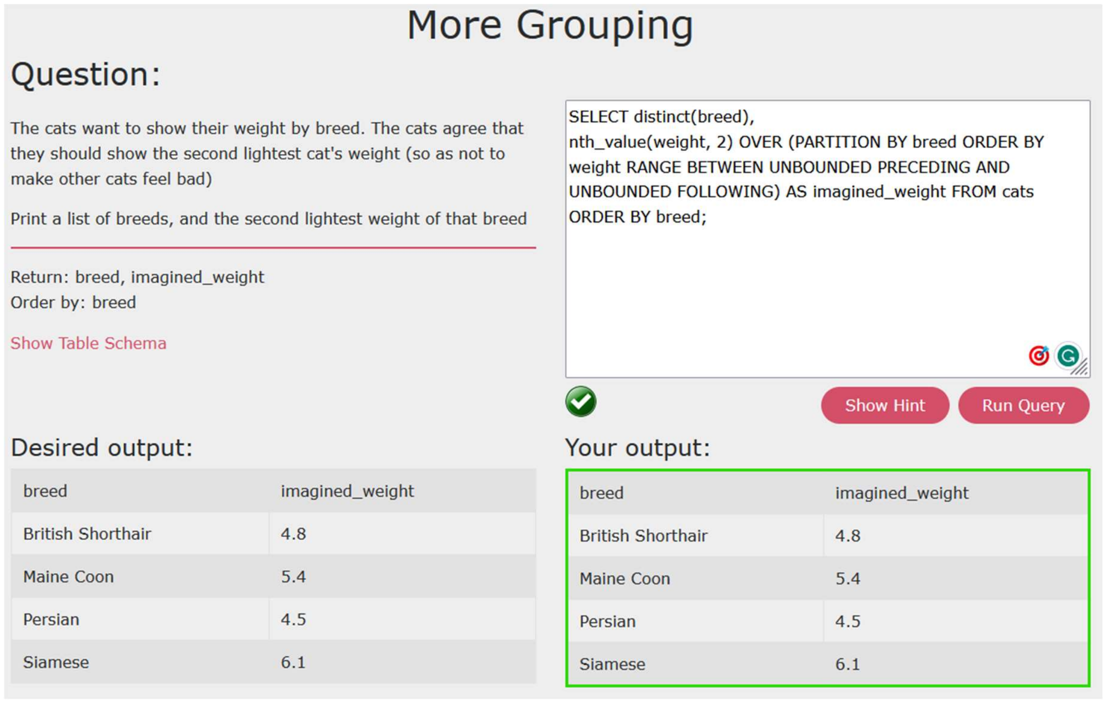
name	weight	imagined_weight
Tigger	3.8	99.9
Molly	4.2	99.9
Ashes	4.5	99.9
Charlie	4.8	4.8
Smudge	4.9	4.8
Felix	5.0	4.8
Puss	5.1	4.8
Millie	5.4	4.8
Alfie	5.5	4.8
Misty	5.7	4.8
Oscar	6.1	4.8
Smokey	6.1	4.8

Your output:

name	weight	imagined_weight
Tigger	3.8	99.9
Molly	4.2	99.9
Ashes	4.5	99.9
Charlie	4.8	4.8
Smudge	4.9	4.8
Felix	5.0	4.8
Puss	5.1	4.8
Millie	5.4	4.8
Alfie	5.5	4.8
Misty	5.7	4.8
Oscar	6.1	4.8
Smokey	6.1	4.8

This query uses the "nth_value" function to calculate the fourth highest weight of all cats and assigns it to a new column called "imagined_weight". If there are fewer than 4 cats in the table, it assigns the value 99.9 to "imagined_weight" using the "coalesce" function.

More Grouping


More Grouping

Question:

The cats want to show their weight by breed. The cats agree that they should show the second lightest cat's weight (so as not to make other cats feel bad)

Print a list of breeds, and the second lightest weight of that breed

Return: breed, imagined_weight
Order by: breed

Show Table Schema

```
SELECT distinct(breed),
nth_value(weight, 2) OVER (PARTITION BY breed ORDER BY
weight RANGE BETWEEN UNBOUNDED PRECEDING AND
UNBOUNDED FOLLOWING) AS imagined_weight FROM cats
ORDER BY breed;
```

Desired output:

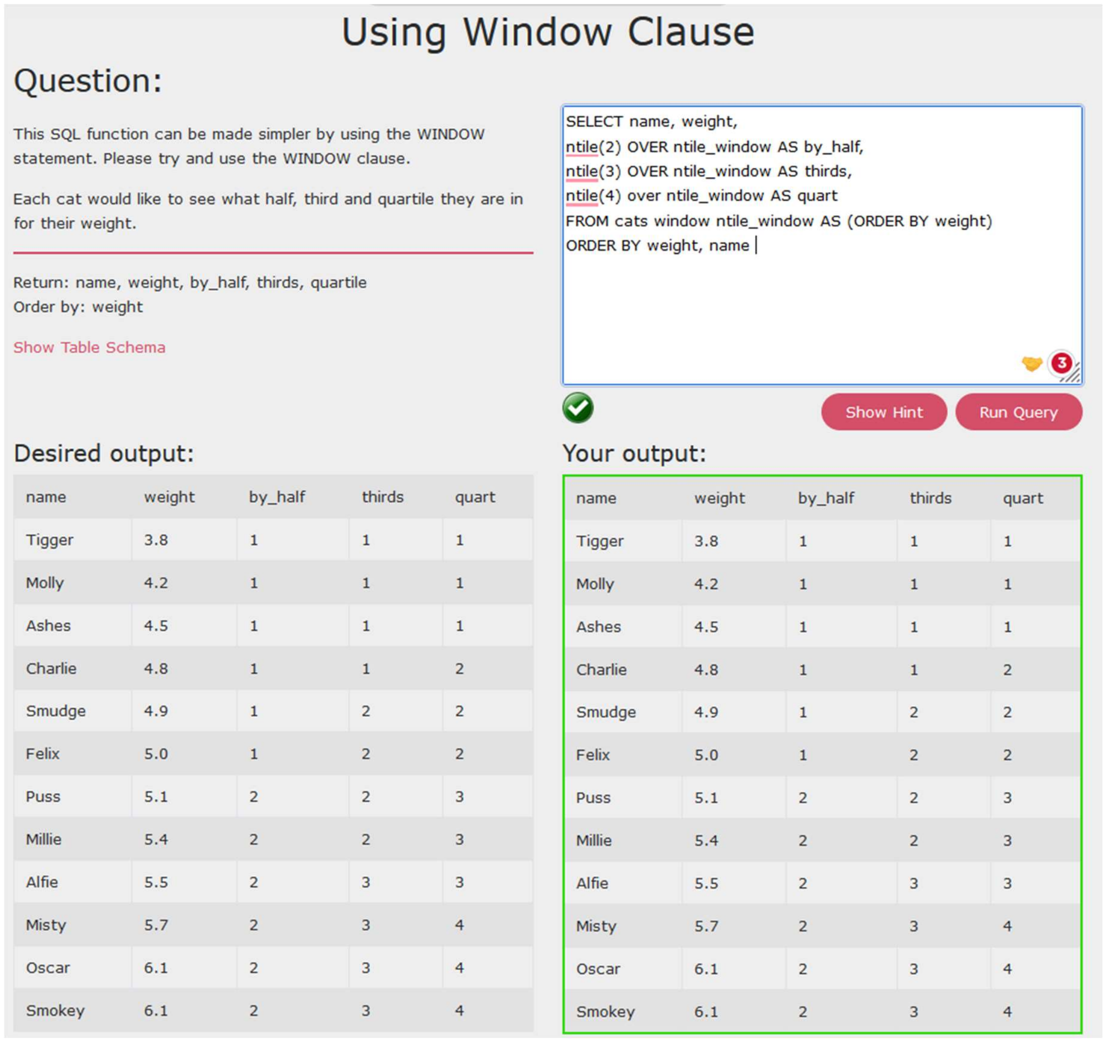
breed	imagined_weight
British Shorthair	4.8
Maine Coon	5.4
Persian	4.5
Siamese	6.1

Your output:

breed	imagined_weight
British Shorthair	4.8
Maine Coon	5.4
Persian	4.5
Siamese	6.1

This SQL query retrieves the distinct breeds of cats from the cats table, then calculates the second heaviest weight for each breed using the nth_value window function. The nth_value function returns the nth value within a window, where n is specified as a parameter. The window is defined by the "PARTITION BY" clause, which partitions the data by breed, and the "ORDER BY" clause, which sorts the data by weight. The "RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING" clause specifies the range of the window as all rows within the partition.

Using Window Clause



Using Window Clause

Question:

This SQL function can be made simpler by using the WINDOW statement. Please try and use the WINDOW clause.

Each cat would like to see what half, third and quartile they are in for their weight.

Return: name, weight, by_half, thirds, quartile
Order by: weight

Show Table Schema

```
SELECT name, weight,
ntile(2) OVER ntile_window AS by_half,
ntile(3) OVER ntile_window AS thirds,
ntile(4) over ntile_window AS quart
FROM cats window ntile_window AS (ORDER BY weight)
ORDER BY weight, name
```

Desired output:

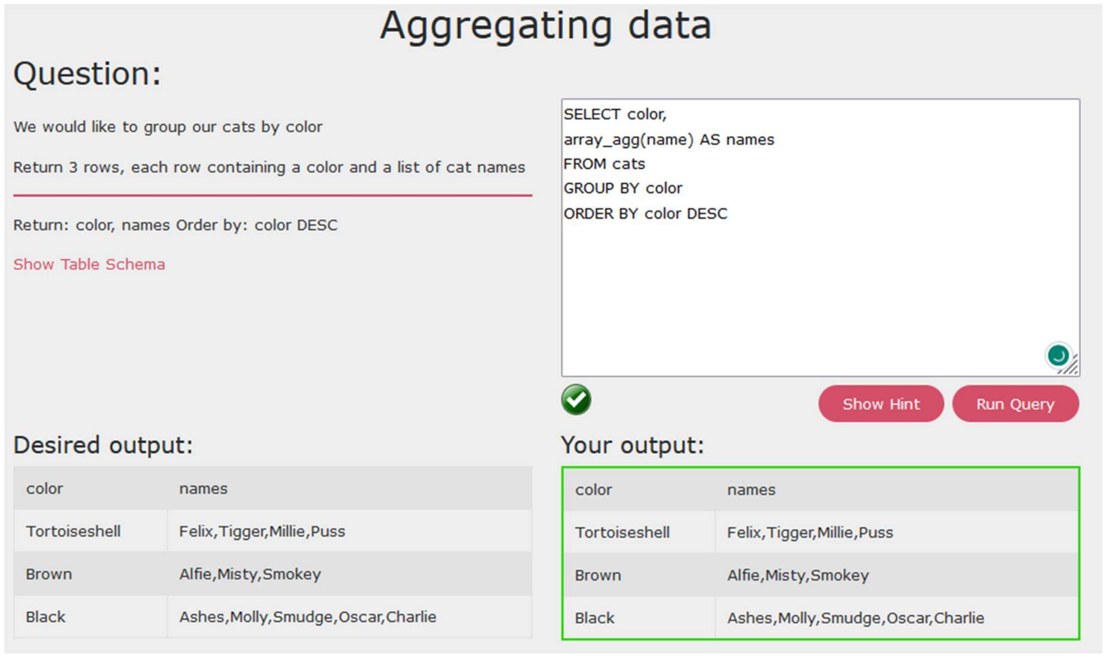
name	weight	by_half	thirds	quart
Tigger	3.8	1	1	1
Molly	4.2	1	1	1
Ashes	4.5	1	1	1
Charlie	4.8	1	1	2
Smudge	4.9	1	2	2
Felix	5.0	1	2	2
Puss	5.1	2	2	3
Millie	5.4	2	2	3
Alfie	5.5	2	3	3
Misty	5.7	2	3	4
Oscar	6.1	2	3	4
Smokey	6.1	2	3	4

Your output:

name	weight	by_half	thirds	quart
Tigger	3.8	1	1	1
Molly	4.2	1	1	1
Ashes	4.5	1	1	1
Charlie	4.8	1	1	2
Smudge	4.9	1	2	2
Felix	5.0	1	2	2
Puss	5.1	2	2	3
Millie	5.4	2	2	3
Alfie	5.5	2	3	3
Misty	5.7	2	3	4
Oscar	6.1	2	3	4
Smokey	6.1	2	3	4

This SQL query selects the name and weight of each cat in the cats table. It also calculates the position of each cat within the following groups: "by_half" (divided into two equal-weight groups), "thirds" (divided into three equal-weight groups), and "quart" (divided into four equal-weight groups). This is done using the "ntile" function, which returns an integer value indicating which bucket the current row belongs to, based on the specified number of groups.

Aggregating data



Aggregating data

Question:

We would like to group our cats by color

Return 3 rows, each row containing a color and a list of cat names

Return: color, names Order by: color DESC

Show Table Schema

```
SELECT color,
array_agg(name) AS names
FROM cats
GROUP BY color
ORDER BY color DESC
```

Desired output:

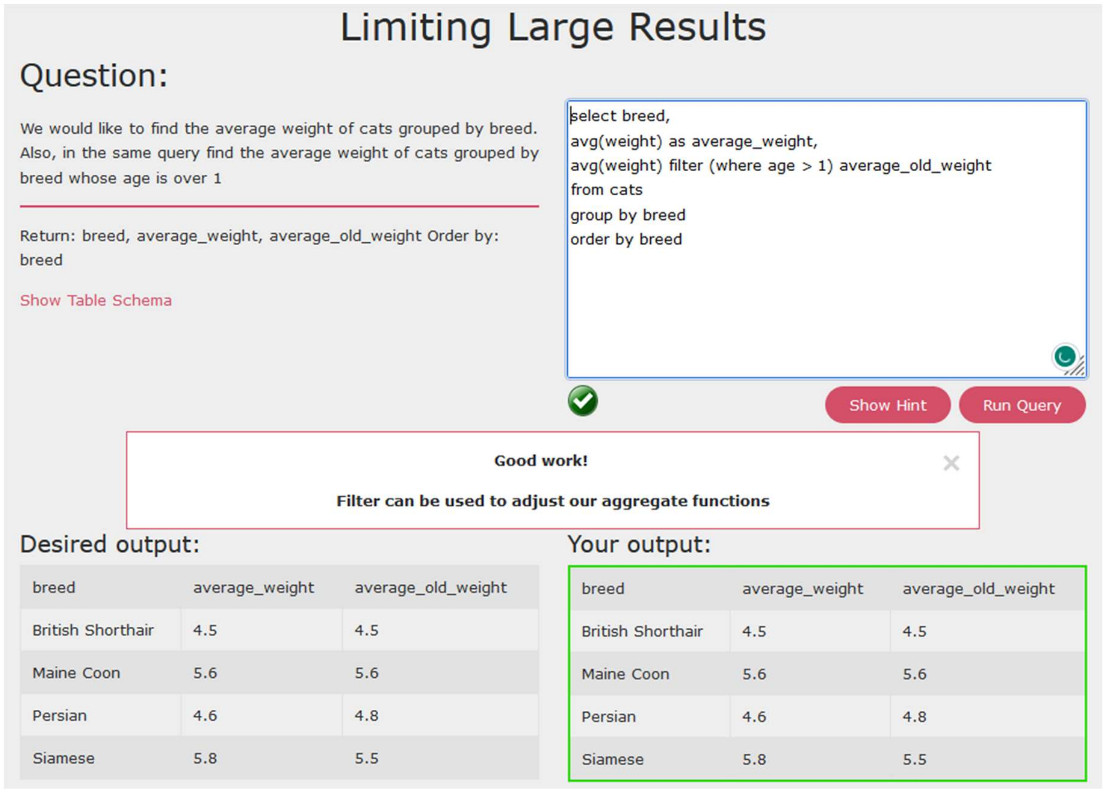
color	names
Tortoiseshell	Felix,Tigger,Millie,Puss
Brown	Alfie,Misty,Smokey
Black	Ashes,Molly,Smudge,Oscar,Charlie

Your output:

color	names
Tortoiseshell	Felix,Tigger,Millie,Puss
Brown	Alfie,Misty,Smokey
Black	Ashes,Molly,Smudge,Oscar,Charlie

This SQL query groups all cats by their color using the GROUP BY clause, and then applies the aggregate function array_agg to the name column. This function aggregates all the names of the cats with the same color into an array. The result is a list of colors and their corresponding arrays of cat names. The ORDER BY clause is used to sort the results in descending order based on the color column.

Limiting Large Results


Limiting Large Results

Question:

We would like to find the average weight of cats grouped by breed. Also, in the same query find the average weight of cats grouped by breed whose age is over 1

Return: breed, average_weight, average_old_weight Order by: breed

```
select breed,
avg(weight) as average_weight,
avg(weight) filter (where age > 1) average_old_weight
from cats
group by breed
order by breed
```

Good work!

Filter can be used to adjust our aggregate functions

Desired output:

breed	average_weight	average_old_weight
British Shorthair	4.5	4.5
Maine Coon	5.6	5.6
Persian	4.6	4.8
Siamese	5.8	5.5

Your output:

breed	average_weight	average_old_weight
British Shorthair	4.5	4.5
Maine Coon	5.6	5.6
Persian	4.6	4.8
Siamese	5.8	5.5

This SQL query selects the breed, average weight, and average weight for cats over the age of one from the cats table. The query uses the avg function to calculate the average weight of all cats, and the filter clause is used to calculate the average weight of cats over the age of one. The results are grouped by breed and ordered by breed in ascending order.

مشکلات و توضیحات تکمیلی

مشکلی نبود.

آنچه آموختم / پیشنهادات

با توابع پنجره‌ای و کارکرد آن آشنا شدم.