**SQL Bug Fixing: Fix the QUERY - Totaling**

Oh no! Timmys been moved into the database divison of his software company but as we know Timmy loves making mistakes. Help Timmy keep his job by fixing his query...

Timmy works for a statistical analysis company and has been given a task of totaling the number of sales on a given day grouped by each department name and then each day.

Resultant table:

day (type: date) {group by} [order by asc]

department (type: text) {group by} [In a real world situation it is bad practice to name a column after a table]

sale\_count (type: int)

Tables and relationship below:

A diagram of a data flow

Description automatically generated with medium confidence

**Solution:**

**SELECT DISTINCT DATE (s.transaction\_date) AS day, d.name AS department, COUNT(s.id) AS sale\_count**

**FROM department d**

**INNER JOIN sale s ON d.id = s.department\_id**

**GROUP BY d.name, day**

**ORDER BY day ASC**

**Calculating Batting Average**

In baseball, the batting average is a simple and most common way to measure a hitter's performace. Batting average is calculated by taking all the players hits and dividing it by their number of at\_bats, and it is usually displayed as a 3 digit decimal (i.e. 0.300).

Given a yankees table with the following schema,

-player\_id STRING

-player\_name STRING

-primary\_position STRING

-games INTEGER

-at\_bats INTEGER

-hits INTEGER

return a table with player\_name, games, and batting\_average.

We want batting\_average to be rounded to the nearest thousandth, since that is how baseball fans are used to seeing it. Format it as text and make sure it has 3 digits to the right of the decimal (pad with zeroes if neccesary).

Next, order our resulting table by batting\_average, with the highest average in the first row.

Finally, since batting\_average is a rate statistic, a small number of at\_bats can change the average dramatically. To correct for this, exclude any player who doesn't have at least 100 at bats.

Expected Output Table

-player\_name STRING

-games INTEGER

-batting\_average STRING

**Solution:**

**SELECT player\_name, games, CAST(ROUND(hits::NUMERIC / at\_bats, 3) AS DECIMAL(10,3))::TEXT AS batting\_average**

**FROM yankees**

**WHERE at\_bats >= 100**

**ORDER BY batting\_average DESC**

**SQL: Disorder**

You are given a table numbers with just one column, number. It holds some numbers that are already ordered.

You need to write a query that makes them un-ordered, as in, every possible ordering should appear equally often.

**Solution:**

**SELECT \***

**FROM numbers**

**ORDER BY RANDOM()**

**SQL with Pokemon: Damage Multipliers**

You have arrived at the Celadon Gym to battle Erika for the Rainbow Badge.

She will be using Grass-type Pokemon. Any fire pokemon you have will be strong against grass, but your water types will be weakened. The multipliers table within your Pokedex will take care of that.

Using the following tables, return the pokemon\_name, modifiedStrength and element of the Pokemon whose strength, after taking these changes into account, is greater than or equal to 40, ordered from strongest to weakest.

pokemon schema

id

pokemon\_name

element\_id

str

multipliers schema

id

element

multiplier

**Solution:**

**SELECT p.pokemon\_name, (p.str \* m.multiplier) AS modifiedStrength, m.element**

**FROM pokemon p**

**LEFT JOIN multipliers m ON p.element\_id = m.id**

**WHERE modifiedStrength >= 40**

**ORDER BY modifiedStrength DESC**

**SQL Basics: Simple table totaling**

For this challenge you need to create a simple query to display each unique clan with their total points and ranked by their total points.

people table schema

name

points

clan

You should then return a table that resembles below

select on

rank

clan

total\_points

total\_people

The query must rank each clan by their total\_points, you must return each unqiue clan and if there is no clan name (i.e. it's an empty string) you must replace it with [no clan specified], you must sum the total\_points for each clan and the total\_people within that clan.

##Note The data is loaded from the live leaderboard, this means values will change but also could cause the kata to time out retreiving the information.

**Solution:**

**SELECT DISTINCT ROW\_NUMBER() OVER (ORDER BY SUM(points) DESC) AS rank,**

**COALESCE(NULLIF(clan,''), '[no clan specified]') AS clan,**

**SUM(points) AS total\_points,**

**COUNT(name) AS total\_people**

**FROM people**

**GROUP BY clan**

**ORDER BY SUM(points) DESC**

**SQL Basics - Monsters using CASE**

You have access to two tables named top\_half and bottom\_half, as follows:

top\_half schema

id

heads

arms

bottom\_half schema

id

legs

tails

You must return a table with the format as follows:

output schema

id

heads

legs

arms

tails

species

The IDs on the tables match to make a full monster. For heads, arms, legs and tails you need to draw in the data from each table.

For the species, if the monster has more heads than arms, more tails than legs, or both, it is a 'BEAST' else it is a 'WEIRDO'. This needs to be captured in the species column.

All rows should be returned (10).

Tests require the use of CASE. Order by species.

**Solution:**

**SELECT t.id, t.heads, t.arms, b.legs, b.tails,**

**(CASE**

**WHEN t.heads > t.arms OR b.tails > b.legs THEN 'BEAST'**

**ELSE 'WEIRDO'**

**END) AS species**

**FROM top\_half t JOIN bottom\_half b ON t.id = b.id**

**ORDER BY species**

**SQL Bug Fixing: Fix the JOIN**

Oh no! Timmys been moved into the database divison of his software company but as we know Timmy loves making mistakes. Help Timmy keep his job by fixing his query...

Timmy works for a statistical analysis company and has been given a task of calculating the highest average salary for a given job, the sample is compiled of 100 applicants each with a job and a salary. Timmy must display each unique job, the total average salary, the total people and the total salary and order by highest average salary. Timmy has some bugs in his query, help Timmy fix his query so he can keep his job!

people table schema

id

name

job table schema

id

people\_id

job\_title

salary

resultant table schema

job\_title (unique)

average\_salary (float, 2 dp)

total\_people (int)

total\_salary (float, 2 dp)

**Solution:**

**SELECT DISTINCT j.job\_title**

**,CAST(ROUND((SUM(j.salary) / COUNT(p)), 2) AS FLOAT) AS average\_salary**

**,COUNT(p.id) AS total\_people**

**,CAST(ROUND(SUM(j.salary), 2) AS FLOAT) AS total\_salary**

**FROM people p**

**INNER JOIN job j ON p.id = j.people\_id**

**GROUP BY j.job\_title**

**ORDER BY average\_salary DESC**

**SQL Basics: Simple PIVOTING data WITHOUT CROSSTAB**

This kata is inspired by SQL Basics: Simple PIVOTING data by matt c.

You need to build a pivot table WITHOUT using CROSSTAB function. Having two tables products and details you need to select a pivot table of products with counts of details occurrences (possible details values are ['good', 'ok', 'bad'].

Results should be ordered by product's name.

Model schema for the kata is:

your query should return table with next columns

name

good

ok

bad

Compare your table to the expected table to view the expected results.

**Solution:**

**SELECT p.name,**

**(SELECT COUNT(d.detail) FROM details d WHERE detail = 'good' AND p.id = d.product\_id ) as good,**

**(SELECT COUNT(d.detail) FROM details d WHERE detail = 'ok' AND p.id = d.product\_id) as ok,**

**(SELECT COUNT(d.detail) FROM details d WHERE detail = 'bad' AND p.id = d.product\_id) as bad**

**FROM products p INNER JOIN details d ON p.id = d.product\_id**

**GROUP BY p.name, p.id**

**ORDER BY p.name**

**SQL Basics: Simple Hierarchical structure**

NOTE: Most difficult query at the moment, required further investigation.

For this challenge you need to create a RECURSIVE Hierarchical query. You have a table employees of employees, you must order each employee by level. You must use a WITH statement and name it employee\_levels after that has been defined you must select from it.

A Level is in correlation what manager managers the employee. e.g. an employee with a manager\_id of NULL is at level 1 and then direct employees with the employee at level 1 will be level 2.

employees table schema

id

first\_name

last\_name

manager\_id (can be NULL)

resultant schema

level

id

first\_name

last\_name

manager\_id (can be NULL)

**Solution:**

**WITH RECURSIVE employee\_levels AS (**

**SELECT 1 AS level, id, first\_name, last\_name, manager\_id**

**FROM employees**

**WHERE manager\_id IS NULL**

**UNION ALL**

**SELECT (el.level + 1) AS level, e.id, e.first\_name, e.last\_name, e.manager\_id**

**FROM employees e**

**INNER JOIN employee\_levels el ON e.manager\_id = el.id**

**)**

**SELECT \***

**FROM employee\_levels**

**ORDER BY level, id;**