WINDOW FUNCTIONS

By Willem Booysen

Version 1.5

Willem

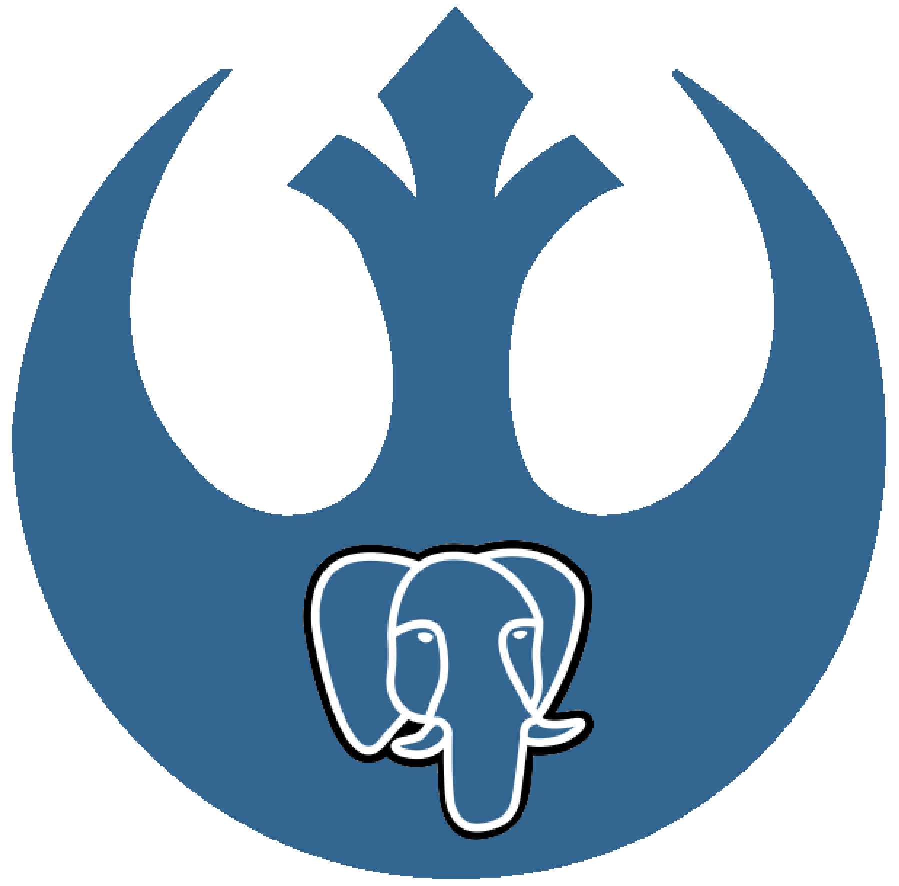
About me

Accountant turned Accidental DBA

God loving, happily married man with 2 wonderful kids.

I'm an accountant by trade and an Accidental DBA by luck.
Spend most of my Postgres time in SQL scripting.

Let's get on with it!



Content Overview

What you can expect in this presentation

STEP 1
WHY WINDOW FUNCTIONS?

What are Window Functions and why use them?

STEP 2
GETTING TO KNOW OUR DATA

Create the database and tables we'll use in this Presentation and getting to know our data

STEP 3

WINDOWS VS PARTITIONS

Understand how your base result set, windows and partitions interact.

STEP 4

BASIC SYNTAX

OVER()

PARTITION BY

STEP 5

ROWS AND RANKS

Because rows should know their place

STEP 6

LAG and LEAD

This has nothing to do with gaming…

STEP 7

FIRST & LAST

It's not as simple as it sounds…

STEP 8

LESS BASIC SYNTAX

Rows, Ranges, Unbounded, following and preceding… Your head **will** hurt here.

STEP 9

RUNNING TOTALS

Because I'm an accountant…

STEP 10

WATCH OUT!

Things you should be aware of

FINISH

What is a Window Function?

"A WINDOW FUNCTION performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row — <u>the rows retain their separate identities</u>."

—PostgreSQL Manual

“What?”

—Me

Basically, Window Functions were created to stop people from using Self Joins and generally reduce the complexity of queries around analytics, aggregate data
and extensive use of cursors.

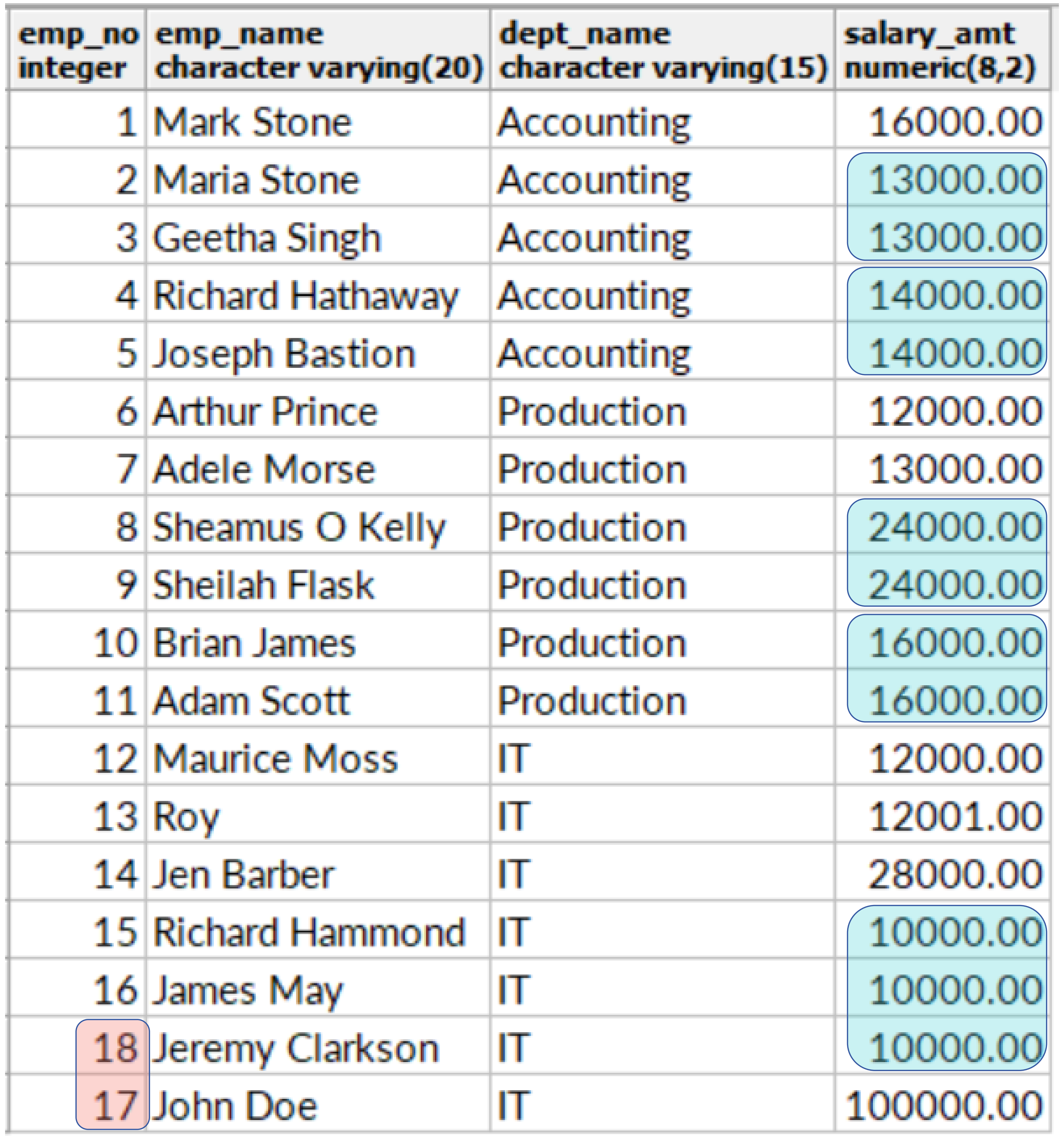
(Purely my opinion based on my Google searches and reading Stackoverflow comments)

I’ll illustrate this soon, but first you need to understand the underlying data used in the coming examples

OUR DATA

3 Departments

- Accounting (5)
- Production (6)
- IT (7)



emp_no integer	emp_name character varying(20)	dept_name character varying(15)	salary_amt numeric(8,2)
1	Mark Stone	Accounting	16000.00
2	Maria Stone	Accounting	13000.00
3	Geetha Singh	Accounting	13000.00
4	Richard Hathaway	Accounting	14000.00
5	Joseph Bastion	Accounting	14000.00
6	Arthur Prince	Production	12000.00
7	Adele Morse	Production	13000.00
8	Sheamus O Kelly	Production	24000.00
9	Sheilah Flask	Production	24000.00
10	Brian James	Production	16000.00
11	Adam Scott	Production	16000.00
12	Maurice Moss	IT	12000.00
13	Roy	IT	12001.00
14	Jen Barber	IT	28000.00
15	Richard Hammond	IT	10000.00
16	James May	IT	10000.00
18	Jeremy Clarkson	IT	10000.00
17	John Doe	IT	100000.00

Duplicate Salaries

Messed up index

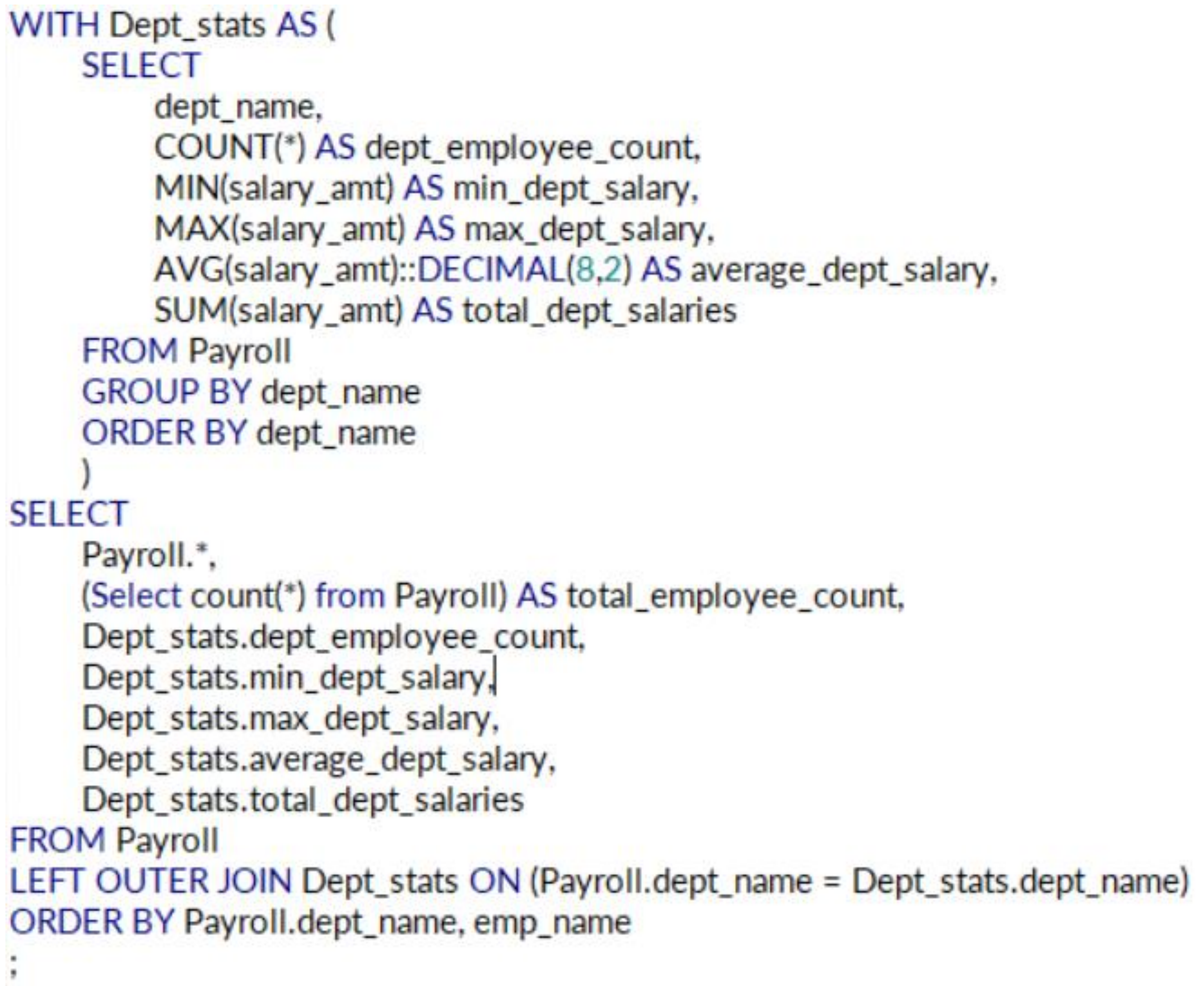
DEMO

TIME FOR SOME FUN

*

Demo Recap

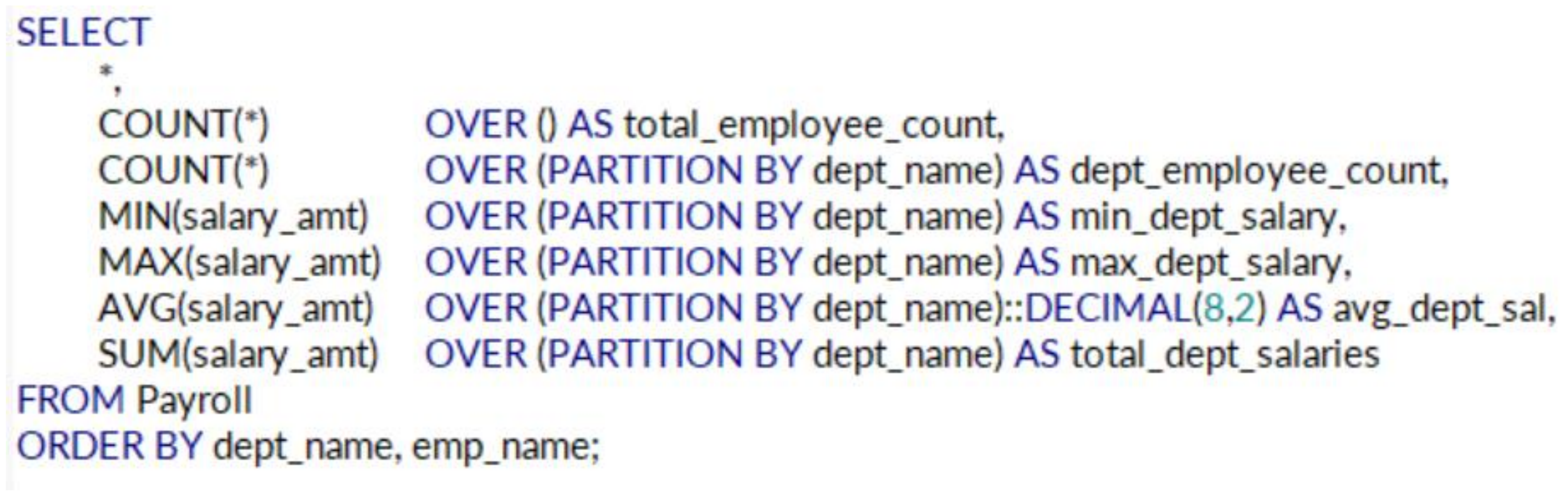
Traditional Method



```
WITH Dept_stats AS (
	SELECT
		dept_name,
		COUNT(*) AS dept_employee_count,
		MIN(salary_amt) AS min_dept_salary,
		MAX(salary_amt) AS max_dept_salary,
		AVG(salary_amt)::DECIMAL(8,2) AS average_dept_salary,
		SUM(salary_amt) AS total_dept_salaries
	FROM Payroll
	GROUP BY dept_name
	ORDER BY dept_name
	)
SELECT
	Payroll.*,
	(Select count(*) from Payroll) AS total_employee_count,
	Dept_stats.dept_employee_count,
	Dept_stats.min_dept_salary,
	Dept_stats.max_dept_salary,
	Dept_stats.average_dept_salary,
	Dept_stats.total_dept_salaries
FROM Payroll
LEFT OUTER JOIN Dept_stats ON (Payroll.dept_name = Dept_stats.dept_name)
ORDER BY Payroll.dept_name, emp_name
;
```

VS

Window Functions

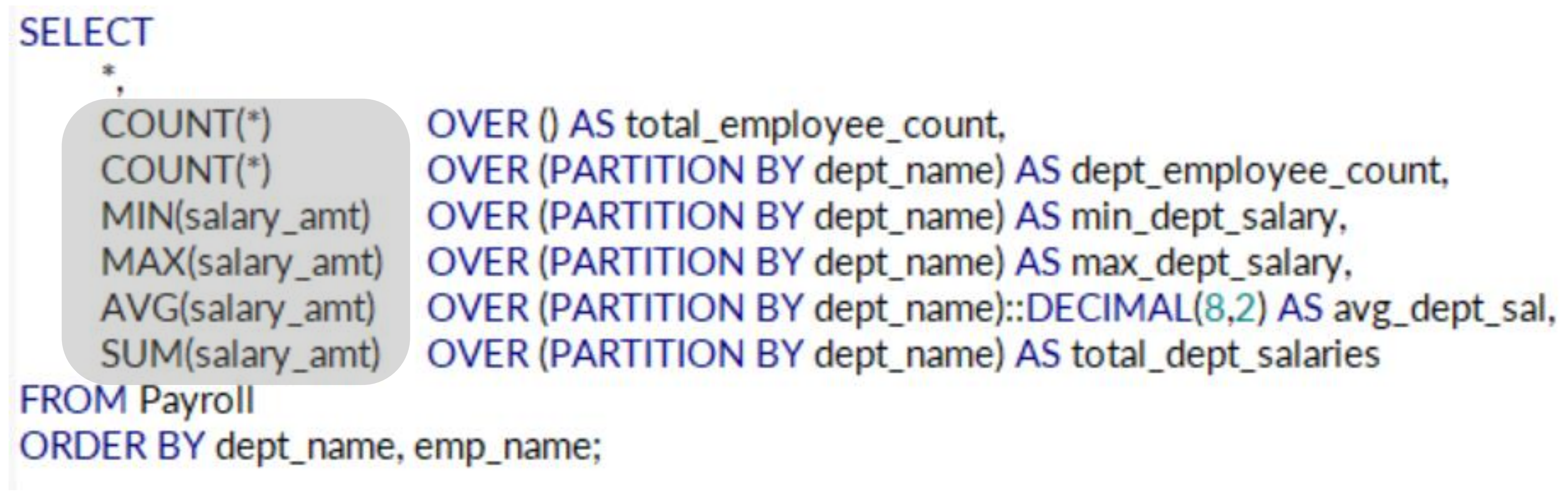


```
SELECT
	*,
	COUNT(*)        OVER () AS total_employee_count,
	COUNT(*)        OVER (PARTITION BY dept_name) AS dept_employee_count,
	MIN(salary_amt) OVER (PARTITION BY dept_name) AS min_dept_salary,
	MAX(salary_amt) OVER (PARTITION BY dept_name) AS max_dept_salary,
	AVG(salary_amt) OVER (PARTITION BY dept_name)::DECIMAL(8,2) AS avg_dept_sal,
	SUM(salary_amt) OVER (PARTITION BY dept_name) AS total_dept_salaries
FROM Payroll
ORDER BY dept_name, emp_name;
```

Demo Recap

It all starts with an Aggregate Function

Window Functions



```
SELECT
    *,
    COUNT(*)          OVER () AS total_employee_count,
    COUNT(*)          OVER (PARTITION BY dept_name) AS dept_employee_count,
    MIN(salary_amt)   OVER (PARTITION BY dept_name) AS min_dept_salary,
    MAX(salary_amt)   OVER (PARTITION BY dept_name) AS max_dept_salary,
    AVG(salary_amt)   OVER (PARTITION BY dept_name)::DECIMAL(8,2) AS avg_dept_sal,
    SUM(salary_amt)   OVER (PARTITION BY dept_name) AS total_dept_salaries
FROM Payroll
ORDER BY dept_name, emp_name;
```

A visual guide to Windows and Partitions

Knowing WHERE it's at is half the battle

04

OVER (PARTITION BY…)
Split the Base Data Set into PARTITIONs and open a Window OVER each of them

03

OVER ()
Open a Window OVER the entire Base Data Set.

A window's beauty is limited to the landscape beyond – the Base Data Set

02

SELECT … WHERE …
Our Base Data Set is the result of limiting expressions, like WHERE

01

YOUR TABLE
All the data within your table, before any queries against it

Basic Syntax

OVER (PARTITION BY...)

OVER ()

OVER () =

OVER (PARTITION BY...)

OVER ()

OVER (**PARTITION BY…**)

OVER ()

OVER (PARTITION BY... ORDER BY ...)

OVER ()

OVER (PARTITION BY… ORDER BY …)

You can also control the order in which rows are processed by window functions using the ORDER BY clause.

The window ORDER BY does not have to match the order in which the rows are output (the order of the Base Data Set)

DEMO

TIME FOR SOME FUN

Ranking

It’s not as simple as first, second and third…

One cannot assign a rank without ORDER

Ranking

It's not as simple as first, second and third…

1
2
3
4

Row_Numbers

Allocates row numbers based on the ORDER BY specified within the Window.

1
2
2
4

Rank

Duplicate values are assigned the same rank, SKIPPING the next number in line.

1
2
2
3

Dense_Rank

Duplicate values are assigned the same rank, no values are skipped.

DEMO

ORDER IN THE COURT!

Advanced Ranking

(For Data Scientists and Statisticians)

percent_rank()
Relative rank of the current row… (rank -1) / (total rows – 1)

cume_dist()
Relative rank of the current row… (no or rows preceding or peer with current row) / (total rows)

ntile(num_buckets integer)
Returns integer ranging from 1 to the argument value, dividing the partition as equally as possible

DEMO… AGAIN

GOOD LUCK WITH THIS ONE…

LAG and LEAD

Offset from the current row

Syntax

LEAD/LAG (column, offset, default_value) OVER (...)

FIRST and LAST

Offset relative to beginning/end of the window frame

Syntax

```
FIRST_VALUE (column) OVER (...)
LAST_VALUE (column) OVER (...)
```

window frame

Window Frames increase with each row inside your partition, from row 1. Think of it as analytics

step
        by
            step,

row
      by
            row

(based on your partition order)

DEMO

LAG & LEAD

with some

FIRST & LAST

Catchy Phrases

Mostly hidden by default

ROWS BETWEEN

2 PRECEDING

AND

3 FOLLOWING

Row	Function / Position
1	First_Value / Min
2	…
3	…
4	…
5	Lag
6	Current Row
7	Lead
8	…
9	…
10	…
11	Last_Value / Max

ROWS BETWEEN

UNBOUNDED PRECEDING

AND

UNBOUNDED FOLLOWING

Unless you have an ORDER BY…
Then the default becomes:

RANGE BETWEEN

2 PRECEDING

AND

3 FOLLOWING

Row	Function / Position
1	First_Value / Min
2	…
3	…
4	…
5	Lag
6	Current Row
7	Lead
8	…
9	…
10	…
11	Last_Value / Max

RANGE BETWEEN

UNBOUNDED PRECEDING

AND

UNBOUNDED FOLLOWING

DEMO

TIME FOR SOME FUN

Catchy Phrases

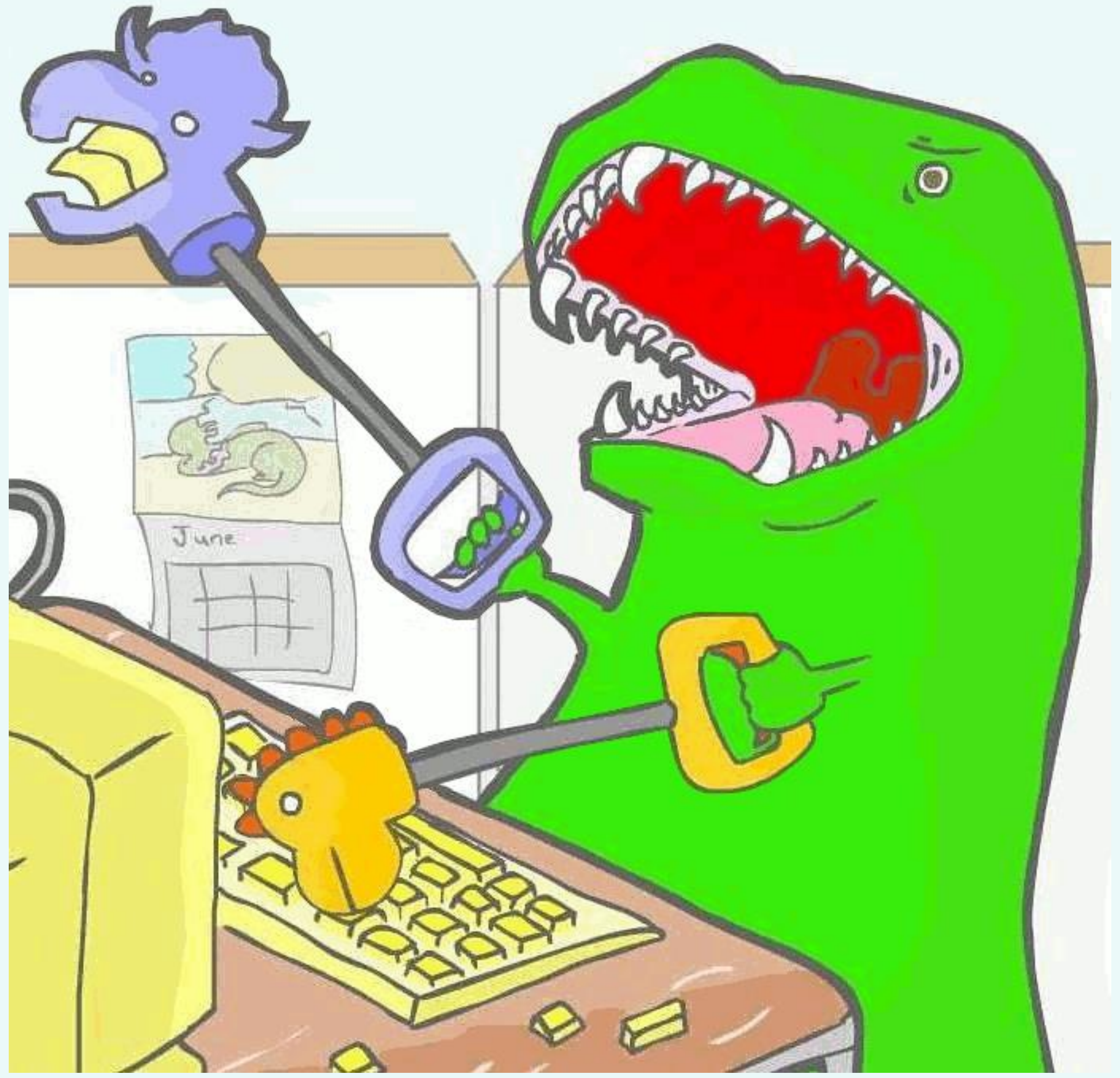
Mostly hidden by default

What is the difference between ROWS between and RANGE between?

- "ROWS" is over "PARTITION BY“
- "RANGE" is over "ORDER BY" (within the Partition of course)

Running Totals

Because I'm an accountant



DEMO

Run Mr Totals. Run!

WARNING

1. Issues with Distinct()
2. You cannot use Window Functions in your WHERE clauses
3. Window Frames effect functions, e.g. MIN/MAX/FIRST/LAST

DEMO

Crash and burn

The End