O'REILLY®

Advanced SQL Queries in 90 Minutes

Alice Zhao

Alice Zhao

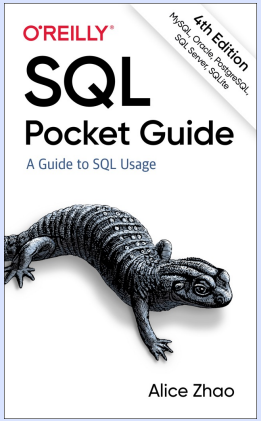
A Dash of Data

Current Role

Data Science & Analytics Instructor

Past Roles

- Data Scientist
- Analyst
- Consultant
- Engineer
- Author



Setting Expectations

Advanced SQL Queries in 90 Minutes

- The focus of this course is on advanced **queries**
- Ideal audience:
  - Taken a beginner SQL course
  - Want to learn concepts beyond the basics

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

SQL Querying Basics

A Quick Refresher: The SELECT Statement

Columns to display

Table(s) to pull from

Filter rows

Split rows into groups

Sort rows

Return limited rows

```
SELECT   name,
         SUM(num_babies) AS total
FROM     baby_names
WHERE    year BETWEEN 1990 AND 1999
GROUP BY name
ORDER BY total DESC
LIMIT 5;
```

SQL Querying Basics

A Quick Refresher: JOIN Types

Name	Fruit
Abby	Apples
Bobby	Bananas
Coco	Coconuts
Dan	Dates

Name	Dessert
Coco	Cookies
Dan	Donuts
Evie	Eclairs

```
SELECT *
FROM   fruits f INNER JOIN
       dessert d
       ON f.Name = d.Name
```

Name	Fruit	Name	Dessert
Coco	Coconuts	Coco	Cookies
Dan	Dates	Dan	Donuts

SQL Querying Basics

A Quick Refresher: JOIN Types

Name	Fruit
Abby	Apples
Bobby	Bananas
Coco	Coconuts
Dan	Dates

Name	Dessert
Coco	Cookies
Dan	Donuts
Evie	Eclairs

```
SELECT *
FROM   fruits f INNER JOIN
       dessert d
       ON f.Name = d.Name
```

Name	Fruit	Name	Dessert
Coco	Coconuts	Coco	Cookies
Dan	Dates	Dan	Donuts

```
SELECT *
FROM   fruits f OUTER JOIN
       dessert d
       ON f.Name = d.Name
```

Name	Fruit	Name	Dessert
Abby	Apples	NULL	NULL
Bobby	Bananas	NULL	NULL
Coco	Coconuts	Coco	Cookies
Dan	Dates	Dan	Donuts
NULL	NULL	Evie	Eclairs

SQL Querying Basics

A Quick Refresher: JOIN Types

Name	Fruit
Abby	Apples
Bobby	Bananas
Coco	Coconuts
Dan	Dates

Name	Dessert
Coco	Cookies
Dan	Donuts
Evie	Eclairs

```
SELECT *
FROM   fruits f LEFT JOIN
       dessert d
       ON f.Name = d.Name
```

Name	Fruit	Name	Dessert
Abby	Apples	NULL	NULL
Bobby	Bananas	NULL	NULL
Coco	Coconuts	Coco	Cookies
Dan	Dates	Dan	Donuts

```
SELECT *
FROM   fruits f RIGHT JOIN
       dessert d
       ON f.Name = d.Name
```

Name	Fruit	Name	Dessert
Coco	Coconuts	Coco	Cookies
Dan	Dates	Dan	Donuts
NULL	NULL	Evie	Eclairs

Poll

(Can choose multiple answers)

What are the advanced querying concepts that you are most curious about?

- Window Functions
- Subqueries and Common Table Expressions
- Case Statements (with GROUP BY, COUNT, SUM and Pivoting)
- Numeric, DateTime and String Functions (Regular Expressions, COALESCE, etc.)
- Other (please share in the chat)

Window Functions

Advanced SQL Queries

Window Functions

Aggregate Functions vs Window Functions

sales_table

	Name	Month	Sales
group window	Ava	May	2
	Ava	July	11
group window	Zed	April	3
	Zed	May	14
	Zed	June	7
	Zed	July	1

```
-- Aggregate function
SELECT   Name,
         SUM(Sales) AS Total_Sales
FROM     sales_table
GROUP BY name;
```

Name	Total_Sales
Ava	13
David	25

```
-- Window function
SELECT   Name,
         ROW_NUMBER() OVER
         (PARTITION BY NAME
         ORDER BY SALES) AS Row_Number
FROM     sales_table;
```

Name	Row_Number
Ava	1
Ava	2
Zed	1
Zed	2
Zed	3
Zed	4

Aggregate functions collapse rows, whereas window functions leave them as is.

Window Functions

Breaking Down the Window Function

States that this is a window function

(required)

How each window should be sorted before the function is applied

(optional)

```
ROW_NUMBER() OVER ( PARTITION BY NAME ORDER BY SALES )
```

The function you want to apply to each window:

- `ROW_NUMBER`
- `RANK`
- `FIRST_VALUE`
- `LAG`

(required)

How you are splitting your data into windows, on:

- One column
- Multiple columns
- The entire table (if left blank)

(optional)

Window Functions

ROW_NUMBER() OVER ()

sales_table

Name	Month	Sales
Ava	May	2
Ava	July	11
Zed	April	3
Zed	May	14
Zed	June	7
Zed	July	1

window

```
-- Return all row numbers
SELECT   Name, Month, Sales,
         ROW_NUMBER() OVER () AS Row_Number
FROM     sales_table;
```

Name	Month	Sales	Row_Number
Ava	May	2	1
Ava	July	11	2
Zed	April	3	3
Zed	May	14	4
Zed	June	7	5
Zed	July	1	6

```
ROW_NUMBER() OVER ( PARTITION BY NAME ORDER BY SALES )
```

Window Functions

ROW_NUMBER() OVER (PARTITION BY NAME)

sales_table

	Name	Month	Sales
window	Ava	May	2
	Ava	July	11
window	Zed	April	3
	Zed	May	14
	Zed	June	7
	Zed	July	1

```
-- Return all row numbers within each window
SELECT   Name, Month, Sales,
         ROW_NUMBER() OVER (PARTITION BY Name)
         AS Row_Number
FROM     sales_table;
```

Name	Month	Sales	Row_Number
Ava	May	2	1
Ava	July	11	2
Zed	April	3	1
Zed	May	14	2
Zed	June	7	3
Zed	July	1	4

```
ROW_NUMBER() OVER ( PARTITION BY NAME ORDER BY SALES )
```

Window Functions

ROW_NUMBER() OVER (PARTITION BY NAME ORDER BY SALE)

sales_table

	Name	Month	Sales
window	Ava	May	2
	Ava	July	11
window	Zed	April	3
	Zed	May	14
	Zed	June	7
	Zed	July	1

```
-- Return all row numbers within each window
-- where the rows are ordered by sales
SELECT   Name, Month, Sales,
         ROW_NUMBER() OVER (PARTITION BY Name
         ORDER BY Sales) AS Sales_Rank
FROM     sales_table;
```

Name	Month	Sales	Sales_Rank
Ava	May	2	1
Ava	July	11	2
Zed	July	1	1
Zed	April	3	2
Zed	June	7	3
Zed	May	14	4

```
ROW_NUMBER() OVER ( PARTITION BY NAME ORDER BY SALES )
```

Window Functions

ROW_NUMBER() OVER (PARTITION BY NAME ORDER BY SALE DESC)

sales_table

	Name	Month	Sales
window	Ava	May	2
	Ava	July	11
window	Zed	April	3
	Zed	May	14
	Zed	June	7
	Zed	July	1

```
-- Return all row numbers within each window
-- where the rows are ordered by sales
SELECT   Name, Month, Sales,
         ROW_NUMBER() OVER (PARTITION BY NAME
         ORDER BY Sales DESC) AS Sales_Rank
FROM     sales_table;
```

Name	Month	Sales	Sales_Rank
Ava	July	11	1
Ava	May	2	2
Zed	May	14	1
Zed	June	7	2
Zed	April	3	3
Zed	July	1	4

```
ROW_NUMBER() OVER ( PARTITION BY NAME ORDER BY SALES )
```

Window Functions

Breaking Down the Window Function

States that this is a window function

(required)

How each window should be sorted before the function is applied

(optional)

```
ROW_NUMBER() OVER ( PARTITION BY NAME ORDER BY SALES )
```

The function you want to apply to each window:

- `ROW_NUMBER`
- `RANK`
- `FIRST_VALUE`
- `LAG`

(required)

How you are splitting your data into windows, on:

- One column
- Multiple columns
- The entire table (if left blank)

(optional)

Window Functions

Function Comparison

```
SELECT   Name, Babies,
         ROW_NUMBER() OVER () AS ROW_NUMBER,
         RANK() OVER () AS RANK,
         DENSE_RANK() OVER () AS DENSE_RANK,
FROM     baby_names;
```

baby_names

Name	Babies
Olivia	99
Emma	80
Charlotte	80
Amelia	75
Sophia	72
Isabella	70
Ava	70
Mia	64

Name	Babies	ROW_NUMBER	RANK	DENSE_RANK
Olivia	99	1	1	1
Emma	80	2	2	2
Charlotte	80	3	2	2
Amelia	75	4	4	3
Sophia	72	5	5	4
Isabella	70	6	6	5
Ava	70	7	6	5
Mia	64	8	8	6

Window Functions

Function Comparison

```
SELECT   Name, Babies,
         ROW_NUMBER() OVER () AS ROW_NUMBER,
         RANK() OVER () AS RANK,
         DENSE_RANK() OVER () AS DENSE_RANK,
FROM     baby_names;
```

baby_names

Name	Babies
Olivia	99
Emma	80
Charlotte	80
Amelia	75
Sophia	72
Isabella	70
Ava	70
Mia	64

Name	Babies	ROW_NUMBER	RANK	DENSE_RANK
Olivia	99	1	1	1
Emma	80	2	2	2
Charlotte	80	3	2	2
Amelia	75	4	4	3
Sophia	72	5	5	4
Isabella	70	6	6	5
Ava	70	7	6	5
Mia	64	8	8	6

ROW_NUMBER
breaks all ties

Window Functions

Function Comparison

```
SELECT   Name, Babies,
         ROW_NUMBER() OVER () AS ROW_NUMBER,
         RANK() OVER () AS RANK,
         DENSE_RANK() OVER () AS DENSE_RANK,
FROM     baby_names;
```

baby_names

Name	Babies
Olivia	99
Emma	80
Charlotte	80
Amelia	75
Sophia	72
Isabella	70
Ava	70
Mia	64

Name	Babies	ROW_NUMBER	RANK	DENSE_RANK
Olivia	99	1	1	1
Emma	80	2	2	2
Charlotte	80	3	2	2
Amelia	75	4	4	3
Sophia	72	5	5	4
Isabella	70	6	6	5
Ava	70	7	6	5
Mia	64	8	8	6

RANK keeps the tie

Window Functions

Function Comparison

```
SELECT   Name, Babies,
         ROW_NUMBER() OVER () AS ROW_NUMBER,
         RANK() OVER () AS RANK,
         DENSE_RANK() OVER () AS DENSE_RANK,
FROM     baby_names;
```

baby_names

Name	Babies
Olivia	99
Emma	80
Charlotte	80
Amelia	75
Sophia	72
Isabella	70
Ava	70
Mia	64

Name	Babies	ROW_NUMBER	RANK	DENSE_RANK
Olivia	99	1	1	1
Emma	80	2	2	2
Charlotte	80	3	2	2
Amelia	75	4	4	3
Sophia	72	5	5	4
Isabella	70	6	6	5
Ava	70	7	6	5
Mia	64	8	8	6

DENSE_RANK keeps the tie and doesn't skip numbers

Window Functions

Example Use Cases

1. Return the first value in each group
2. Return the second value in each group
3. **Interactive Exercise:** Return the first 5 values in each group
4. Return the prior row value
5. Calculate the moving average

Window Functions

Return the First Value in Each Group

```
-- Display the most popular name for each gender
SELECT   Gender, Name, Babies,
                          OVER (PARTITION BY Gender
         ORDER BY Babies DESC) AS Top_Name
FROM     baby_names;
```

baby_names

Gender	Name	Babies
Female	Charlotte	80
Female	Emma	82
Female	Olivia	99
Male	James	85
Male	Liam	110
Male	Noah	95

window

window

Gender	Name	Babies
Female	Olivia	99
Female	Emma	82
Female	Charlotte	80
Male	Liam	110
Male	Noah	95
Male	Oliver	85

Window Functions

Return the First Value in Each Group

```
-- Display the most popular name for each gender
SELECT   Gender, Name, Babies,
         FIRST_VALUE(Name) OVER (PARTITION BY Gender
         ORDER BY Babies DESC) AS Top_Name
FROM     baby_names;
```

baby_names

	Gender	Name	Babies
window	Female	Charlotte	80
	Female	Emma	82
	Female	Olivia	99
window	Male	James	85
	Male	Liam	110
	Male	Noah	95

Gender	Name	Babies	Top_Name
Female	Olivia	99	Olivia
Female	Emma	82	Olivia
Female	Charlotte	80	Olivia
Male	Liam	110	Liam
Male	Noah	95	Liam
Male	Oliver	85	Liam

Window Functions

Return the First Value in Each Group

```
-- Display the most popular name
-- for each gender (part 2)

SELECT Gender, Name, Babies
FROM
(SELECT   Gender, Name, Babies,
          FIRST_VALUE(Name) OVER
          (PARTITION BY Gender
          ORDER BY Babies DESC)
          AS Top_Name
FROM      baby_names) AS tn
WHERE Name = Top_Name;
```

Gender	Name	Babies
Female	Olivia	99
Male	Liam	110

```
-- Display the most popular name for each gender
SELECT    Gender, Name, Babies,
          FIRST_VALUE(Name) OVER (PARTITION BY Gender
          ORDER BY Babies DESC) AS Top_Name
FROM      baby_names;
```

Gender	Name	Babies	Top_Name
Female	Olivia	99	Olivia
Female	Emma	82	Olivia
Female	Charlotte	80	Olivia
Male	Liam	110	Liam
Male	Noah	95	Liam
Male	Oliver	85	Liam

There is also a LAST_VALUE function

Window Functions

Return the Second Value in Each Group

```
-- Display the 2nd most popular name for each gender
SELECT   Gender, Name, Babies,
                              OVER (PARTITION BY Gender
         ORDER BY Babies DESC) AS Second_Name
FROM     baby_names;
```

baby_names

Gender	Name	Babies
Female	Charlotte	80
Female	Emma	82
Female	Olivia	99
Male	James	85
Male	Liam	110
Male	Noah	95

window

window

Gender	Name	Babies
Female	Olivia	99
Female	Emma	82
Female	Charlotte	80
Male	Liam	110
Male	Noah	95
Male	Oliver	85

Window Functions

Return the Second Value in Each Group

```
-- Display the 2nd most popular name for each gender
SELECT   Gender, Name, Babies,
         NTH_VALUE(Name, 2) OVER (PARTITION BY Gender
         ORDER BY Babies DESC) AS Second_Name
FROM     baby_names;
```

baby_names

Gender	Name	Babies
Female	Charlotte	80
Female	Emma	82
Female	Olivia	99
Male	James	85
Male	Liam	110
Male	Noah	95

window

window

Gender	Name	Babies	Second_Name
Female	Olivia	99	NULL
Female	Emma	82	Emma
Female	Charlotte	80	Emma
Male	Liam	110	NULL
Male	Noah	95	Noah
Male	Oliver	85	Noah

Window Functions

Return the Second Value in Each Group

```
-- Display the 2nd most popular
-- name for each gender (part 2)

SELECT Gender, Name, Babies
FROM
(SELECT   Gender, Name, Babies,
          NTH_VALUE(Name, 2) OVER
          (PARTITION BY Gender
          ORDER BY Babies DESC)
          AS Second_Name
FROM      baby_names) AS sn
WHERE Name = Second_Name;
```

Gender	Name	Babies
Female	Emma	82
Male	Noah	95

```
-- Display the 2nd most popular name for each gender
SELECT    Gender, Name, Babies,
          NTH_VALUE(Name, 2) OVER (PARTITION BY Gender
          ORDER BY Babies DESC) AS Second_Name
FROM      baby_names;
```

Gender	Name	Babies	Second_Name
Female	Olivia	99	NULL
Female	Emma	82	Emma
Female	Charlotte	80	Emma
Male	Liam	110	NULL
Male	Noah	95	Noah
Male	Oliver	85	Noah

Interactive Exercise:

Return the First 5 Values of Each Group

Window Functions

Return the Prior Row Value

```
-- Return the prior sales row
SELECT   Name, Month, Sales,
                    OVER (PARTITION BY Name
         ORDER BY Month) AS Prior_Sales
FROM     sales_table;
```

sales_table

	Name	Month	Sales
window	Ava	5	2
	Ava	6	11
	Ava	7	5
window	Zed	4	3
	Zed	5	14
	Zed	6	7
	Zed	7	1

Name	Month	Sales
Ava	5	2
Ava	6	11
Ava	7	5
Zed	4	3
Zed	5	14
Zed	6	7
Zed	7	1

There is also a LEAD function

Window Functions

Return the Prior Row Value

```
-- Return the prior sales row
SELECT   Name, Month, Sales,
         LAG(Sales) OVER (PARTITION BY Name
         ORDER BY Month) AS Prior_Sales
FROM     sales_table;
```

sales_table

	Name	Month	Sales
window	Ava	5	2
	Ava	6	11
	Ava	7	5
window	Zed	4	3
	Zed	5	14
	Zed	6	7
	Zed	7	1

Name	Month	Sales	Prior_Sales
Ava	5	2	NULL
Ava	6	11	2
Ava	7	5	11
Zed	4	3	NULL
Zed	5	14	3
Zed	6	7	14
Zed	7	1	7

There is also a LEAD function

Window Functions

Calculate the Moving Average

```
-- Return the three month moving average
SELECT   Name, Month, Sales,
                  OVER (PARTITION BY Name
         ORDER BY Month

FROM     sales_table;
```

sales_table

	Name	Month	Sales
window	Ava	5	2
	Ava	6	11
	Ava	7	5
window	Zed	4	3
	Zed	5	14
	Zed	6	7
	Zed	7	1

Name	Month	Sales
Ava	5	2
Ava	6	11
Ava	7	5
Zed	4	3
Zed	5	14
Zed	6	7
Zed	7	1

Window Functions

Calculate the Moving Average

```
-- Return the three month moving average
SELECT   Name, Month, Sales,
         AVG(Sales) OVER (PARTITION BY Name
         ORDER BY Month ROWS BETWEEN 2 PRECEDING
         AND CURRENT ROW) AS Three_Month_MA
FROM     sales_table;
```

sales_table

Name	Month	Sales
Ava	5	2
Ava	6	11
Ava	7	5
Zed	4	3
Zed	5	14
Zed	6	7
Zed	7	1

Name	Month	Sales	Three_Month_MA
Ava	5	2	NULL
Ava	6	11	
Ava	7	5	
Zed	4	3	
Zed	5	14	
Zed	6	7	
Zed	7	1	

Window Functions

Calculate the Moving Average

```
-- Return the three month moving average
SELECT   Name, Month, Sales,
         AVG(Sales) OVER (PARTITION BY Name
         ORDER BY Month ROWS BETWEEN 2 PRECEDING
         AND CURRENT ROW) AS Three_Month_MA
FROM     sales_table;
```

sales_table

Name	Month	Sales
Ava	5	2
Ava	6	11
Ava	7	5
Zed	4	3
Zed	5	14
Zed	6	7
Zed	7	1

Name	Month	Sales	Three_Month_MA
Ava	5	2	NULL
Ava	6	11	NULL
Ava	7	5	
Zed	4	3	
Zed	5	14	
Zed	6	7	
Zed	7	1	

Window Functions

Calculate the Moving Average

```
-- Return the three month moving average
SELECT   Name, Month, Sales,
         AVG(Sales) OVER (PARTITION BY Name
         ORDER BY Month ROWS BETWEEN 2 PRECEDING
         AND CURRENT ROW) AS Three_Month_MA
FROM     sales_table;
```

sales_table

Name	Month	Sales
Ava	5	2
Ava	6	11
Ava	7	5
Zed	4	3
Zed	5	14
Zed	6	7
Zed	7	1

Name	Month	Sales	Three_Month_MA
Ava	5	2	NULL
Ava	6	11	NULL
Ava	7	5	6
Zed	4	3	
Zed	5	14	
Zed	6	7	
Zed	7	1	

Window Functions

Calculate the Moving Average

```
-- Return the three month moving average
SELECT   Name, Month, Sales,
         AVG(Sales) OVER (PARTITION BY Name
         ORDER BY Month ROWS BETWEEN 2 PRECEDING
         AND CURRENT ROW) AS Three_Month_MA
FROM     sales_table;
```

sales_table

Name	Month	Sales
Ava	5	2
Ava	6	11
Ava	7	5
Zed	4	3
Zed	5	14
Zed	6	7
Zed	7	1

Name	Month	Sales	Three_Month_MA
Ava	5	2	NULL
Ava	6	11	NULL
Ava	7	5	6
Zed	4	3	NULL
Zed	5	14	
Zed	6	7	
Zed	7	1	

Window Functions

Calculate the Moving Average

```
-- Return the three month moving average
SELECT   Name, Month, Sales,
         AVG(Sales) OVER (PARTITION BY Name
         ORDER BY Month ROWS BETWEEN 2 PRECEDING
         AND CURRENT ROW) AS Three_Month_MA
FROM     sales_table;
```

sales_table

Name	Month	Sales
Ava	5	2
Ava	6	11
Ava	7	5
Zed	4	3
Zed	5	14
Zed	6	7
Zed	7	1

Name	Month	Sales	Three_Month_MA
Ava	5	2	NULL
Ava	6	11	NULL
Ava	7	5	6
Zed	4	3	NULL
Zed	5	14	NULL
Zed	6	7	
Zed	7	1	

Window Functions

Calculate the Moving Average

```
-- Return the three month moving average
SELECT   Name, Month, Sales,
         AVG(Sales) OVER (PARTITION BY Name
         ORDER BY Month ROWS BETWEEN 2 PRECEDING
         AND CURRENT ROW) AS Three_Month_MA
FROM     sales_table;
```

sales_table

Name	Month	Sales
Ava	5	2
Ava	6	11
Ava	7	5
Zed	4	3
Zed	5	14
Zed	6	7
Zed	7	1

Name	Month	Sales	Three_Month_MA
Ava	5	2	NULL
Ava	6	11	NULL
Ava	7	5	6
Zed	4	3	NULL
Zed	5	14	NULL
Zed	6	7	8
Zed	7	1	

Window Functions

Calculate the Moving Average

```
-- Return the three month moving average
SELECT   Name, Month, Sales,
         AVG(Sales) OVER (PARTITION BY Name
         ORDER BY Month ROWS BETWEEN 2 PRECEDING
         AND CURRENT ROW) AS Three_Month_MA
FROM     sales_table;
```

sales_table

Name	Month	Sales
Ava	5	2
Ava	6	11
Ava	7	5
Zed	4	3
Zed	5	14
Zed	6	7
Zed	7	1

Name	Month	Sales	Three_Month_MA
Ava	5	2	NULL
Ava	6	11	NULL
Ava	7	5	6
Zed	4	3	NULL
Zed	5	14	NULL
Zed	6	7	8
Zed	7	1	7.3333

Window Functions

Calculate the Moving Average

```
-- Return the three month moving average
SELECT   Name, Month, Sales,
         AVG(Sales) OVER (PARTITION BY Name
         ORDER BY Month ROWS BETWEEN 2 PRECEDING
         AND CURRENT ROW) AS Three_Month_MA
FROM     sales_table;
```

sales_table

Name	Month	Sales
Ava	5	2
Ava	6	11
Ava	7	5
Zed	4	3
Zed	5	14
Zed	6	7
Zed	7	1

Name	Month	Sales	Three_Month_MA
Ava	5	2	NULL
Ava	6	11	NULL
Ava	7	5	6
Zed	4	3	NULL
Zed	5	14	NULL
Zed	6	7	8
Zed	7	1	7.3333

Window Functions

Summary

- `ROW_NUMBER() OVER (PARTITION BY NAME ORDER BY SALES DESC)`
  - Alternative rankings: `RANK, DENSE_RANK`
- Example use cases
  - Return the first or last value in each group: `FIRST_VALUE, LAST_VALUE`
  - Return the second or nth value in each group: `NTH_VALUE`
  - Return the prior or next row value: `LAG, LEAD`
  - Calculate the moving average: `ROWS BETWEEN 2 PRECEDING AND CURRENT_ROW`

Q&A

Window Functions

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

Subqueries

A Query Within Another Query

```
-- Display the most popular name
-- for each gender

SELECT Gender, Name, Babies
FROM
```

Subquery

parenthesis →

```
(SELECT   Gender, Name, Babies,
          FIRST_VALUE(Name) OVER
          (PARTITION BY Gender
          ORDER BY Babies DESC)
          AS Top_Name
FROM      baby_names) AS tn
```

column alias ←

subquery alias ←

```
WHERE Name = Top_Name;
```

When to use a subquery

1. It takes multiple steps to get the results that you are looking for
2. You can break a bigger problem down into smaller problems

Subqueries

Multiple Steps to Get to Results

sales_table

Name	Month	Sales
Ava	May	2
Ava	June	11
Ava	July	5
Zed	April	3
Zed	May	14
Zed	June	7
Zed	July	1

AVG = 6.14

Goal: Return the rows where the number of sales is greater than the average number of sales

```
-- Step 1: Find the average number of sales

SELECT AVG(Sales) FROM sales_table;


-- Step 2: Find the rows where the number of sales
-- is greater than the average number of sales

SELECT *
FROM   sales_table
WHERE  Sales > (SELECT AVG(Sales) FROM sales_table);
```

Subqueries

Multiple Steps to Get to Results

Results

Name	Month	Sales
Ava	June	11
Zed	May	14
Zed	June	7

AVG = 6.14

Goal: Return the rows where the number of sales is greater than the average number of sales

```
-- Step 1: Find the average number of sales

SELECT AVG(Sales) FROM sales_table;


-- Step 2: Find the rows where the number of sales
-- is greater than the average number of sales

SELECT *
FROM   sales_table
WHERE  Sales > (SELECT AVG(Sales) FROM sales_table);
```

Subqueries

Break a Big Problem Into Smaller Problems

```
-- Return popular names from the 90s

SELECT     c.city, b.year, b.name,
           SUM(b.babies) AS total

FROM       baby_names b
           LEFT JOIN city_details c
           ON b.city_id = c.city_id

WHERE      year BETWEEN 1990 AND 1999

GROUP BY   c.city, b.year, b.name
ORDER BY   total DESC;
```

1. Join two large tables
2. Filter and aggregate the results

```
-- Return popular names from the 90s

SELECT     c.city, b.year, b.name, total

FROM       (SELECT    city_id, year, name,
                      SUM(babies) AS total
            FROM      baby_names
            WHERE     year BETWEEN 1990 AND 1999
            GROUP BY  city, year, name) b

           LEFT JOIN city_details c
           ON b.city_id = c.city_id

ORDER BY   total DESC;
```

Subquery

1. Filter and aggregate a table
2. Join a smaller table with a large table

This query will take less time to run

Subquery Alternatives

Other Ways to Work with Multiple Queries

1. JOINs
2. Common Table Expressions (CTEs)

JOINs

Use JOINs Instead of Correlated Subqueries

```
-- Return the number of baby names rows
-- with a city in the city table

SELECT COUNT(*)

FROM   baby_names b

WHERE  EXISTS (SELECT *
               FROM city_details c
               WHERE b.city_id = c.city_id);
```

correlated subquery

```
-- Return the number of baby names rows
-- with a city in the city table

SELECT COUNT(*)

FROM   baby_names b
       INNER JOIN city_details c
       ON b.city_id = c.city_id;
```

Subqueries should be stand alone. In the case of a correlated subquery, use a JOIN instead for a faster run time.

Common Table Expressions (CTEs)

Use CTEs For Cleaner Code

```
-- Display the most popular name
-- for each gender

SELECT Gender, Name, Babies
FROM
```

Subquery

```
(SELECT   Gender, Name, Babies,
          FIRST_VALUE(Name) OVER
          (PARTITION BY Gender
          ORDER BY Babies DESC)
          AS Top_Name
FROM      baby_names) AS tn

WHERE Name = Top_Name;
```

```
-- Display the most popular name
-- for each gender
```

CTE

```
WITH tn AS (SELECT Gender, Name, Babies,
            FIRST_VALUE(Name) OVER
            (PARTITION BY Gender
            ORDER BY Babies DESC)
            AS Top_Name

SELECT Gender, Name, Babies
FROM   tn
WHERE Name = Top_Name;
```

Advantages of CTEs

- Cleaner code – can state multiple CTEs upfront
- Can reference multiple times within the query

Common Table Expressions (CTEs)

Recursive CTE Example (A Query That References Itself)

stock_prices

Date	Price
2023-03-01	668.27
2023-03-03	678.83
2023-03-04	635.40
2023-03-06	591.01

```
-- Step 1: Use a recursive CTE to generate dates (MySQL)
WITH RECURSIVE my_dates(dt) AS (SELECT '2023-03-01'
                                UNION ALL
                                SELECT dt + INTERVAL 1 DAY
                                FROM my_dates
                                WHERE dt < '2023-03-06')

SELECT * FROM my_dates;
```

Recursive CTE

Date
2023-03-01
2023-03-02
2023-03-03
2023-03-04
2023-03-05
2023-03-06

Common Table Expressions (CTEs)

Recursive CTE Example (A Query That References Itself)

stock_prices

Date	Price
2023-03-01	668.27
2023-03-03	678.83
2023-03-04	635.40
2023-03-06	591.01

```
-- Step 2: Join it back with the original table
WITH RECURSIVE my_dates(dt) AS (SELECT '2023-03-01'
                                UNION ALL
                                SELECT dt + INTERVAL 1 DAY
                                FROM my_dates
                                WHERE dt < '2023-03-06')

SELECT d.dt, s.price
FROM   my_dates d
       LEFT JOIN stock_prices s
       ON d.dt = s.date;
```

Recursive CTE

Date	Price
2023-03-01	668.27
2023-03-02	NULL
2023-03-03	678.83
2023-03-04	635.40
2023-03-05	NULL
2023-03-06	591.01

Subqueries

Summary

- When to use subqueries
  - Multiple steps to get to results
  - Breaking down a big problem into smaller problems
- Subquery alternatives
  - Use JOINs instead of correlated subqueries
  - Use Common Table Expressions (CTEs) for cleaner code
  - Shoutout: Use temp tables and views to save outputs

Q&A

Subqueries & Common Table Expressions

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

CASE Statements

Overview

- A simple CASE statement can be used for IF-ELSE logic
- CASE statements can be used along with other keywords for more advanced data manipulations
    - CASE Statements with GROUP BY
    - CASE Statements with COUNT / SUM
    - **Interactive Exercise**: CASE and SUM Practice

CASE Statements

Basic CASE Statement

pizza_table

category	name	price
Chicken	California Chicken	20.75
Chicken	Chicken Pesto	20.75
Classic	Greek	20.5
Classic	Hawaiian	16.5
Classic	Pepperoni	15.25
Supreme	Spicy Italian	20.75
Veggie	Five Cheese	18.5
Veggie	Cheese	14.95

```
-- Label rows as premium, discount or standard pizza
SELECT category, name, price
       (CASE WHEN price > 20 THEN 'Premium'
             WHEN price < 16 THEN 'Discount'
             ELSE 'Standard' END) AS Pizza_Type
FROM pizza_table;
```

category	name	price	Pizza Type
Chicken	California Chicken	20.75	Premium
Chicken	Chicken Pesto	20.75	Premium
Classic	Greek	20.5	Premium
Classic	Hawaiian	16.5	Standard
Classic	Pepperoni	15.25	Discount
Supreme	Spicy Italian	20.75	Premium
Veggie	Five Cheese	18.5	Standard
Veggie	Cheese	14.95	Discount

CASE Statements

CASE Statement with a GROUP BY

category	name	price	Pizza Type
Chicken	California Chicken	20.75	Premium
Chicken	Chicken Pesto	20.75	Premium
Classic	Greek	20.5	Premium
Classic	Hawaiian	16.5	Standard
Classic	Pepperoni	15.25	Discount
Supreme	Spicy Italian	20.75	Premium
Veggie	Five Cheese	18.5	Standard
Veggie	Cheese	14.95	Discount

```
-- Find the number of pizzas of each Pizza Type
SELECT (CASE WHEN price > 20 THEN 'Premium'
             WHEN price < 16 THEN 'Discount'
             ELSE 'Standard' END) AS Pizza_Type,
       COUNT(*) AS Num_Pizzas
FROM pizza_table
GROUP BY Pizza_Type;
```

Pizza Type	Num_Pizzas
Discount	2
Premium	4
Standard	2

CASE Statements

CASE Statement with SUMs (Pivoting)

category	name	price	Pizza Type
Chicken	California Chicken	20.75	Premium
Chicken	Chicken Pesto	20.75	Premium
Classic	Greek	20.5	Premium
Classic	Hawaiian	16.5	Standard
Classic	Pepperoni	15.25	Discount
Supreme	Spicy Italian	20.75	Premium
Veggie	Five Cheese	18.5	Standard
Veggie	Cheese	14.95	Discount

```
-- Create a summary table of categories & pizza types
SELECT category,
       SUM(CASE WHEN price > 20 THEN 1 ELSE 0 END)
         AS Premium,
       SUM(CASE WHEN price < 16 THEN 1 ELSE 0 END)
         AS Discount,
       SUM(CASE WHEN price BETWEEN 16 AND 20 THEN 1
         ELSE 0 END) AS Standard
FROM pizza_table
GROUP BY category;
```

category	Premium	Discount	Standard
Chicken	2	0	0
Classic	1	1	1
Supreme	1	0	0
Veggie	0	1	1

CASE Statements

CASE Statement with SUMs (Pivoting)

category	name	price	Pizza Type
Chicken	California Chicken	20.75	Premium
Chicken	Chicken Pesto	20.75	Premium
Classic	Greek	20.5	Premium
Classic	Hawaiian	16.5	Standard
Classic	Pepperoni	15.25	Discount
Supreme	Spicy Italian	20.75	Premium
Veggie	Five Cheese	18.5	Standard
Veggie	Cheese	14.95	Discount

```
-- Create a summary table of categories & pizza types
SELECT category,
       SUM(CASE WHEN price > 20 THEN 1 ELSE 0 END)
         AS Premium,
       SUM(CASE WHEN price < 16 THEN 1 ELSE 0 END)
         AS Discount,
       SUM(CASE WHEN price BETWEEN 16 AND 20 THEN 1
         ELSE 0 END) AS Standard
FROM pizza_table
GROUP BY category;
```

category	Premium	Discount	Standard
Chicken	2	0	0
Classic	1	1	1
Supreme	1	0	0
Veggie	0	1	1

CASE Statements

CASE Statement with SUMs (Pivoting)

category	name	price	Pizza Type
Chicken	California Chicken	20.75	Premium
Chicken	Chicken Pesto	20.75	Premium
Classic	Greek	20.5	Premium
Classic	Hawaiian	16.5	Standard
Classic	Pepperoni	15.25	Discount
Supreme	Spicy Italian	20.75	Premium
Veggie	Five Cheese	18.5	Standard
Veggie	Cheese	14.95	Discount

```
-- Create a summary table of categories & pizza types
SELECT category,
       SUM(CASE WHEN price > 20 THEN 1 ELSE 0 END)
         AS Premium,
       SUM(CASE WHEN price < 16 THEN 1 ELSE 0 END)
         AS Discount,
       SUM(CASE WHEN price BETWEEN 16 AND 20 THEN 1
         ELSE 0 END) AS Standard
FROM pizza_table
GROUP BY category;
```

category	Premium	Discount	Standard
Chicken	2	0	0
Classic	1	1	1
Supreme	1	0	0
Veggie	0	1	1

CASE Statements

CASE Statement with SUMs (Pivoting)

category	name	price	Pizza Type
Chicken	California Chicken	20.75	Premium
Chicken	Chicken Pesto	20.75	Premium
Classic	Greek	20.5	Premium
Classic	Hawaiian	16.5	Standard
Classic	Pepperoni	15.25	Discount
Supreme	Spicy Italian	20.75	Premium
Veggie	Five Cheese	18.5	Standard
Veggie	Cheese	14.95	Discount

```
-- Create a summary table of categories & pizza types
SELECT category,
       SUM(CASE WHEN price > 20 THEN 1 ELSE 0 END)
         AS Premium,
       SUM(CASE WHEN price < 16 THEN 1 ELSE 0 END)
         AS Discount,
       SUM(CASE WHEN price BETWEEN 16 AND 20 THEN 1
         ELSE 0 END) AS Standard
FROM pizza_table
GROUP BY category;
```

category	Premium	Discount	Standard
Chicken	2	0	0
Classic	1	1	1
Supreme	1	0	0
Veggie	0	1	1

CASE Statements

CASE Statement with SUMs (Pivoting)

category	name	price	Pizza Type
Chicken	California Chicken	20.75	Premium
Chicken	Chicken Pesto	20.75	Premium
Classic	Greek	20.5	Premium
Classic	Hawaiian	16.5	Standard
Classic	Pepperoni	15.25	Discount
Supreme	Spicy Italian	20.75	Premium
Veggie	Five Cheese	18.5	Standard
Veggie	Cheese	14.95	Discount

```
-- Create a summary table of categories & pizza types
SELECT category,
       SUM(CASE WHEN price > 20 THEN 1 ELSE 0 END)
         AS Premium,
       SUM(CASE WHEN price < 16 THEN 1 ELSE 0 END)
         AS Discount,
       SUM(CASE WHEN price BETWEEN 16 AND 20 THEN 1
         ELSE 0 END) AS Standard
FROM pizza_table
GROUP BY category;
```

category	Premium	Discount	Standard
Chicken	2	0	0
Classic	1	1	1
Supreme	1	0	0
Veggie	0	1	1

CASE Statements

CASE Statement with SUMs (Pivoting)

category	name	price	Pizza Type
Chicken	California Chicken	20.75	Premium
Chicken	Chicken Pesto	20.75	Premium
Classic	Greek	20.5	Premium
Classic	Hawaiian	16.5	Standard
Classic	Pepperoni	15.25	Discount
Supreme	Spicy Italian	20.75	Premium
Veggie	Five Cheese	18.5	Standard
Veggie	Cheese	14.95	Discount

```
-- Create a summary table of categories & pizza types
SELECT category,
       SUM(CASE WHEN price > 20 THEN 1 ELSE 0 END)
         AS Premium,
       SUM(CASE WHEN price < 16 THEN 1 ELSE 0 END)
         AS Discount,
       SUM(CASE WHEN price BETWEEN 16 AND 20 THEN 1
         ELSE 0 END) AS Standard
FROM pizza_table
GROUP BY category;
```

category	Premium	Discount	Standard
Chicken	2	0	0
Classic	1	1	1
Supreme	1	0	0
Veggie	0	1	1

Interactive Exercise:

CASE and SUM Practice

CASE Statements

Summary

- Basic syntax: **`CASE WHEN … THEN … WHEN … THEN … ELSE … END`**
- To aggregate by group: use with GROUP BY
- To pivot: use with COUNT or SUM

Q&A

CASE Statements

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

Common SQL Functions

Functions apply a calculation or transformation, and output a value

Aggregate Functions	Numeric Functions	Datetime Functions	String Functions	Data Type Functions
AVG()	CEIL()	CURRENT_ DATE	CONCAT()	CAST()
COUNT()	FLOOR()	CURRENT_ TIME	LENGTH()	CONVERT()
MAX()	LOG()	DATEDIFF()	REGEXP()	Null Functions
MIN()	ROUND()	DATE_TRUNC()	SUBSTR()	
SUM()	SQRT()	EXTRACT()	TRIM()	COALESCE()

Numeric Function Example

ROUND() is one of many numeric functions that you can apply to numeric columns

pizza_table

Name	Price
Cheese	$15.25
Pepperoni	$17.95
Veggie	$19.45
Supreme	$20.75

```
SELECT    ROUND(Price * 1.08, 2)
               AS Price_with_Tax
FROM      pizza_table;
```

Name	Price_with_Tax
Cheese	$16.47
Pepperoni	$19.39
Veggie	$21.01
Supreme	$22.41

DateTime Function Example

EXTRACT() allows you to extract a component of a datetime column

pizza_orders

Order_ID	Order_Date
1011	2023-06-28
1012	2023-06-30
1013	2023-07-01
1014	2023-07-02

```
-- MySQL, Oracle, and PostgreSQL
SELECT EXTRACT(month FROM order_date) AS order_month
FROM pizza_orders;

-- SQL Server
SELECT DATEPART('month', order_date) AS order_month
FROM pizza_orders;

-- SQLite
SELECT strftime('%m', order_date) AS order_month
FROM pizza_orders;
```

Order_Month
06
06
07
07

String Function Example

Regular expressions are useful for finding patterns in text data

movies

Title
10 Things I Hate About You
22 Jump Street
The Blues Brothers
Ferris Bueller's Day Off

```
-- MySQL
SELECT *
FROM movies
WHERE title REGEXP '\\d';

-- Oracle
WHERE REGEXP_LIKE(title, '\d');

-- PostgreSQL
WHERE title ~ '\d';

-- SQL Server
WHERE title LIKE '%[0-9]%';
```

Title
10 Things I Hate About You
22 Jump Street

String Function Example

Regular expressions are useful for finding patterns in text data

baby_names

Name
Edward
Elsa
Estella
Evan
Ezra

```
-- Find names that start with an E
-- and end with an A (PostgreSQL)

SELECT name
FROM baby_names
WHERE name ~ 'E.*a$';
```

Name
Elsa
Estella
Ezra

CAST Function Example

CAST() temporarily converts a column to a different data type (numeric, string, datetime)

my_table

id	str_column
1	1.33
2	5.5
3	7.8

```
SELECT *
FROM my_table
WHERE CAST (str_column AS DECIMAL) > 3;
```

id	str_column
2	5.5
3	7.8

COALESCE Function Example

COALESCE() replaces NULL values with another value

stock_prices

Date	Price
2023-03-01	668.27
2023-03-02	NULL
2023-03-03	678.83
2023-03-04	635.40
2023-03-05	NULL
2023-03-06	591.01

```
SELECT date,
       COALESCE(price, 600) AS price
FROM   stock_prices;
```

Date	Price
2023-03-01	668.27
2023-03-02	600
2023-03-03	678.83
2023-03-04	635.40
2023-03-05	600
2023-03-06	591.01

COALESCE Function Example

COALESCE() replaces NULL values with another value

stock_prices

Date	Price
2023-03-01	668.27
2023-03-02	NULL
2023-03-03	678.83
2023-03-04	635.40
2023-03-05	NULL
2023-03-06	591.01

```
SELECT date,
       COALESCE(price,
       LAG(price) OVER
       (ORDER BY date)) AS price
FROM   stock_prices;
```

Date	Price
2023-03-01	668.27
2023-03-02	668.27
2023-03-03	678.83
2023-03-04	635.40
2023-03-05	635.40
2023-03-06	591.01

Q&A

Numeric, DateTime & String Functions

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

Agenda

Advanced SQL Queries in 90 Minutes

- Introduction (5 minutes)
- Window Functions (30 minutes)
- Subqueries and Common Table Expressions (20 minutes)
- Break (5 minutes)
- Case Statements (10 minutes)
- Numeric, DateTime and String Functions (10 minutes)
- Wrap Up + Q&A (10 minutes)

Wrap Up

Topics Covered

- Window Functions
- Subqueries and Common Table Expressions
- CASE Statements
- Numeric, DateTime and String Functions

Wrap Up

Window Functions

States that this is a window function

(required)

How each window should be sorted before the function is applied

(optional)

```
ROW_NUMBER() OVER ( PARTITION BY NAME ORDER BY SALES )
```

The function you want to apply to each window:

- `ROW_NUMBER`
- `RANK`
- `FIRST_VALUE`
- `LAG`

(required)

How you are splitting your data into windows, on:

- One column
- Multiple columns
- The entire table (if left blank)

(optional)

Wrap Up

Subqueries and Common Table Expressions

```
-- Display the most popular name
-- for each gender

SELECT Gender, Name, Babies
FROM
```

Subquery

```
(SELECT   Gender, Name, Babies,
          FIRST_VALUE(Name) OVER
          (PARTITION BY Gender
          ORDER BY Babies DESC)
          AS Top_Name
FROM      baby_names) AS tn
```

```
WHERE Name = Top_Name;
```

```
-- Display the most popular name
-- for each gender
```

CTE

```
WITH tn AS (SELECT Gender, Name, Babies,
            FIRST_VALUE(Name) OVER
            (PARTITION BY Gender
            ORDER BY Babies DESC)
            AS Top_Name
```

```
SELECT Gender, Name, Babies
FROM   tn
WHERE Name = Top_Name;
```

Wrap Up

CASE Statements

category	name	price	Pizza Type
Chicken	California Chicken	20.75	Premium
Chicken	Chicken Pesto	20.75	Premium
Classic	Greek	20.5	Premium
Classic	Hawaiian	16.5	Standard
Classic	Pepperoni	15.25	Discount
Supreme	Spicy Italian	20.75	Premium
Veggie	Five Cheese	18.5	Standard
Veggie	Cheese	14.95	Discount

```
-- Pivot the Pizza Type column
SELECT category,
       SUM(CASE WHEN price > 20 THEN 1 ELSE 0 END)
         AS Premium,
       SUM(CASE WHEN price < 16 THEN 1 ELSE 0 END)
         AS Discount,
       SUM(CASE WHEN price BETWEEN 16 AND 20 THEN 1
         ELSE 0 END) AS Standard
FROM pizza_table
GROUP BY category;
```

category	Premium	Discount	Standard
Chicken	2	0	0
Classic	1	1	1
Supreme	1	0	0
Veggie	0	1	1

Wrap Up

Numeric, DateTime and String Functions

Aggregate Functions	Numeric Functions	Datetime Functions	String Functions	Data Type Functions
AVG()	CEIL()	CURRENT_ DATE	CONCAT()	CAST()
COUNT()	FLOOR()	CURRENT_ TIME	LENGTH()	CONVERT()
MAX()	LOG()	DATEDIFF()	REGEXP()	Null Functions
MIN()	ROUND()	DATE_TRUNC()	SUBSTR()	
SUM()	SQRT()	EXTRACT()	TRIM()	COALESCE()

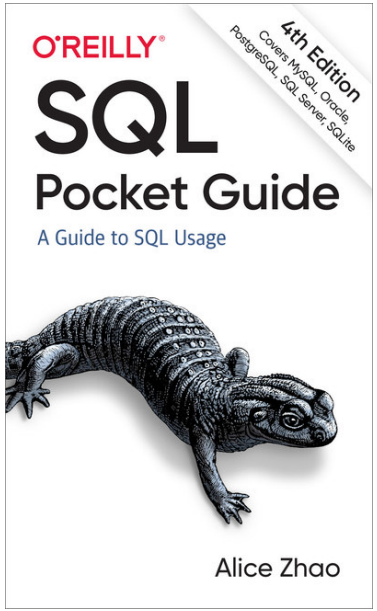
Wrap Up

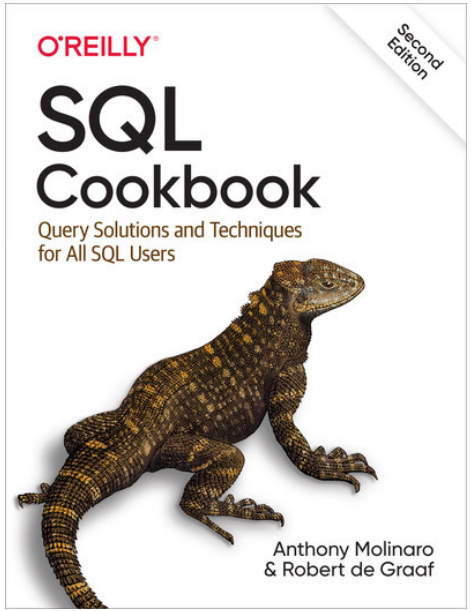
Group Discussion

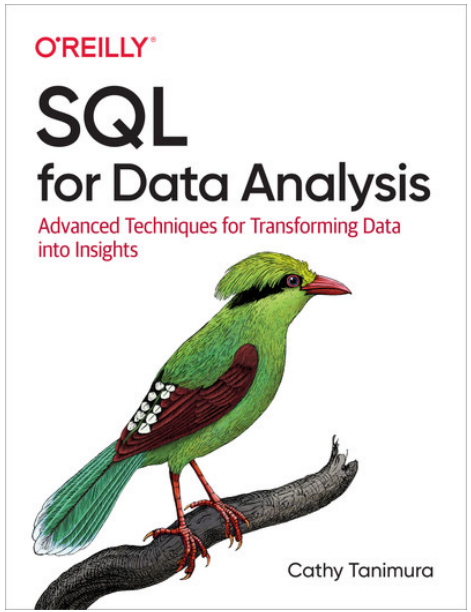
- What concepts from today will you keep in mind going forward?
- What are some other concepts you're curious about that weren't covered today?

Additional Resources

O'Reilly Learning







Live Online Course: Writing Better SQL in 90 Minutes with Alice Zhao

O'REILLY®

Advanced SQL Queries in 90 Minutes

Alice Zhao, A Dash of Data


O'REILLY®