

# Building complex calculations

REPORTING IN SQL



**Tyler Pernes**

Learning & Development Consultant

# Approaches

1. Window functions
2. Layered calculations

# Window functions

- References **other rows** in the table.

# Window functions

- References **other rows** in the table.

## No partition


group	value
A	5
A	4
B	6
B	3

# Window functions

- References **other rows** in the table.


## No partition

group	value
A	5
A	4
B	6
B	3



## Partitioned

group	value
A	5
A	4
B	6
B	3



# Window function syntax

```
SUM(value) OVER (PARTITION BY field ORDER BY field)
```

# Window function syntax

```
SUM(value) OVER (PARTITION BY field ORDER BY field)
```



# Window function syntax

```
SUM(value) OVER (PARTITION BY field ORDER BY field)
```

*optional*

- **PARTITION BY** = range of calculation
- **ORDER BY** = order of rows when running calculation



# Window function examples

## Total bronze medals

```
SELECT
  country_id,
  athlete_id,
  SUM(bronze) OVER () AS total_bronze
FROM summer_games;
```

```
+-----+-----+-----+
| country_id | athlete_id | total_bronze |
+-----+-----+-----+
| 11         | 77505      | 141          |
| 11         | 11673      | 141          |
| 14         | 85554      | 141          |
| 14         | 76433      | 141          |
+-----+-----+-----+
```

# Window function examples

## Country bronze medals

```
SELECT
  country_id,
  athlete_id,
  SUM(bronze) OVER (PARTITION BY country_id) AS total_bronze
FROM summer_games
```

```
+-----+-----+-----+
| country_id | athlete_id | total_bronze |
|-----|-----|-----|
| 11         | 77505      | 12           |
| 11         | 11673      | 12           |
| 14         | 85554      | 5            |
| 14         | 76433      | 5            |
+-----+-----+-----+
```

# Types of window functions

- `SUM()`
- `AVG()`
- `MIN()`
- `MAX()`

# Types of window functions

- `LAG()` and `LEAD()`

row	value
1	5
2	4
3	6
4	3
5	3

# Types of window functions

- `LAG()` and `LEAD()`

row	value
1	5
2	4
3	6
4	3
5	3

**LAG()**

**LEAD()**

# Types of window functions

- `ROW_NUMBER()` and `RANK()`

row	value	rank
1	5	2
2	4	3
3	6	1
4	3	4
5	3	4

# Window function on an aggregation

original\_table

team_id	player_id	points
1	4123	3
1	5231	6
2	8271	5

desired\_report

team_id	team_points	league_points
1	9	43
2	12	43
3	22	43

# Window function on an aggregation

## Final query

```
SELECT
  team_id,
  SUM(points) AS team_points,
  SUM(SUM(points)) OVER () AS league_points
FROM original_table
GROUP BY team_id;
```



# Window function on an aggregation

```
SELECT
  team_id,
  SUM(points) AS team_points,
  SUM(points) OVER () AS league_points
FROM original_table
GROUP BY team_id;
```

ERROR: points must be an aggregation or appear in a GROUP BY statement.

# Layered calculations

- Aggregate an existing aggregation
- Leverages a subquery

# Layered calculations example

## Step 1: Total bronze medals per country

```
SELECT country_id, SUM(bronze) as bronze_medals
FROM summer_games
GROUP BY country_id;
```

## Step 2: Convert to subquery and take the max

```
SELECT MAX(bronze_medals)
FROM
  (SELECT country_id, SUM(bronze) as bronze_medals
   FROM summer_games
   GROUP BY country_id) AS subquery;
```

# Planning out complex calculations

**Initial table**

row	value
1	5
2	4
3	6
4	3
5	3

**Final report**

row	value	rank
1	5	2
2	4	3
3	6	1
4	3	4
5	3	4

# Planning out complex calculations

**Initial table**

row	value
1	5
2	4
3	6
4	3
5	3



**Intermediate table**



**Final report**

row	value	rank
1	5	2
2	4	3
3	6	1
4	3	4
5	3	4

- Ordering for window function?
- Two aggregations with a layered calculation?

**Let's practice!**  
REPORTING IN SQL

# Comparing groups

## REPORTING IN SQL



**Tyler Pernes**

Learning & Development Consultant

# Types of metrics

- **Volume** metrics
- **Efficiency** metrics



# Volume metrics

- Scale with **size**

# Volume metrics

- Scale with **size**

## Revenue



# Volume metrics

- Scale with **size**

## Revenue



## Population



# Percent of total calculation

basketball\_points table

team_id	player_id	points
1	482	92
1	165	47
2	222	64

```
SELECT team_id, SUM(points) AS points
FROM basketball_points
GROUP BY team_id;
```

# Percent of total calculation

basketball\_points table

team_id	player_id	points
1	482	92
1	165	47
2	222	64

team_id	points
1	782
2	625
3	487
4	398

# Percent of total calculation

## Step 1: Calculate total

```
SELECT
  team_id,
  SUM(points) AS points
  SUM(points) OVER () AS total_points
FROM basketball_points
GROUP BY team_id;
```

## Step 2: Calculate percent of total

```
SELECT
  team_id,
  SUM(points) AS points
  SUM(points) / SUM(points) OVER () AS perc_of_total
FROM basketball_points
GROUP BY team_id;
```

# Percent of total calculation

Results:

```
%20-----%20-----%20-----%20
| team_id | points | perc_of_total |
|-----|-----|-----|
| 1       | 782    | .34           |
| 2       | 625    | .27           |
| 3       | 487    | .21           |
| 4       | 398    | .17           |
%20-----%20-----%20-----%20
```

# Percent of total calculation

Percent of points scored **per player** for **each team**:

```
SELECT
  player_id,
  team_id,
  SUM(points) AS points
  SUM(points) / (SUM(points) OVER (PARTITION BY team_id)) AS perc_of_team
FROM basketball_points
GROUP BY player_id, team_id;
```



# Percent of total calculation

Results:

```
%20-----%20-----%20-----%20-----%20
| player_id | team_id | points | perc_of_team |
|-----|-----|-----|-----|
| 482       | 1       | 92     | .12          |
| 165       | 1       | 47     | .06          |
| 222       | 2       | 64     | .10          |
%20-----%20-----%20-----%20-----%20
```

# Efficiency metrics

- Does not scale with **size**
- Typically a **ratio**

# Efficiency metrics

- Does not scale with **size**
- Typically a **ratio**

## Profit Margin



# Efficiency metrics

- Does not scale with **size**
- Typically a **ratio**

## Profit Margin



## Revenue per customer



# Performance index

- Compares performance to a **benchmark**
- Benchmark typically an **average** or **median**

# Performance index

basketball\_summary table

team_id	games	points
1	24	782
2	20	625
3	12	487

- **Points per game** performance?

# Performance index

## Step 1: points per game for each team

```
SELECT
  team_id,
  points/games AS team_ppg
FROM basketball_summary;
```

## Step 2: points per game for entire league

```
SELECT
  team_id,
  points/games AS team_ppg,
  SUM(points) OVER () / SUM(games) OVER () AS league_ppg
FROM basketball_summary;
```

# Performance index

## Step 3: performance index

```
SELECT
  team_id,
  points/games AS team_ppg,
  SUM(points) OVER () / SUM(games) OVER () AS league_ppg,
  (points/games)
  /
  (SUM(points) OVER () / SUM(games) OVER ()) AS perf_index
FROM basketball_summary;
```



# Performance index

## Step 3: performance index

```
+-----+-----+-----+-----+
| team_id | team_ppg | league_ppg | perf_index |
|-----|-----|-----|-----|
| 1       | 32.6     | 33.8       | 0.96       |
| 2       | 31.3     | 33.8       | 0.92       |
| 3       | 40.6     | 33.8       | 1.20       |
+-----+-----+-----+-----+
```

<sup>1</sup> The results clearly state that team three scores 20% more points than the league average.

# Query time!

REPORTING IN SQL

# Comparing dates

REPORTING IN SQL



**Tyler Pernes**

Learning & Development Consultant

# Questions to answer


1. Last month vs previous month?
2. Rolling 7 days?

```
original_table
+-----+-----+
| date      | revenue |
+-----+-----+
| 2018-01-01 | 400     |
| 2018-01-02 | 380     |
| 2018-01-03 | 625     |
+-----+-----+
```

# Month-over-month comparison

- `LAG(value, offset)` outputs a value from an offset number of rows **previous to** the current row.
- `LEAD(value, offset)` outputs a value from an offset number of rows **after** the current row.

row	value
1	5
2	4
3	6
4	3
5	3



The diagram shows a table with 5 rows. Row 3 is highlighted in orange. A purple arrow labeled `LAG()` points from row 3 to row 1, indicating the value from 2 rows previous. A yellow arrow labeled `LEAD()` points from row 3 to row 5, indicating the value from 2 rows after.

# Month-over-month comparison

## Step 1: show revenue by month

```
SELECT
  DATE_PART('month',date) AS month,
  SUM(revenue) as current_rev
FROM original_table
GROUP BY month;
```

## Step 2: previous month's revenue

```
SELECT
  DATE_PART('month',date) AS month,
  SUM(revenue) as current_rev,
  LAG(SUM(revenue)) OVER (ORDER BY DATE_PART('month',date)) AS prev_rev
FROM original_table
GROUP BY month;
```

# Month-over-month comparison

## Step 3: percent change calculation

```
SELECT
  DATE_PART('month',date) AS month,
  SUM(revenue) as current_rev,
  LAG(SUM(revenue)) OVER (ORDER BY DATE_PART('month',date)) AS prev_rev,
  SUM(revenue)
  /
  LAG(SUM(revenue)) OVER (ORDER BY DATE_PART('month',date))-1 AS perc_change
FROM original_table
GROUP BY month;
```

# Month-over-month comparison

## Step 3: percent change calculation

```
+-----+-----+-----+-----+
| month | current_rev | prev_rev | perc_change |
+-----+-----+-----+-----+
| 01    | 15000       | null     | null        |
| 02    | 14000       | 15000    | -.06        |
| 03    | 21000       | 14000    | .50         |
+-----+-----+-----+-----+
```



# Rolling calculations

- Only take into account **7 rows**

New clause: **ROWS BETWEEN**

```
SUM(value) OVER (ORDER BY value ROWS BETWEEN N PRECEDING AND CURRENT ROW)
```

# Rolling calculations

## Rolling sum query

```
SELECT
    date,
    SUM(SUM(revenue)) OVER
        (ORDER BY date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS weekly_revenue
FROM original_table
GROUP BY date;
```

# New table: web\_data

web\_data

date	country_id	view
2018-01-01	1	24313
2018-01-01	2	3768
2018-01-01	3	26817

**Let's practice!**  
REPORTING IN SQL

# Course summary

REPORTING IN SQL



**Tyler Pernes**

Learning & Development Consultant

# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Understanding new datasets

# Course summary

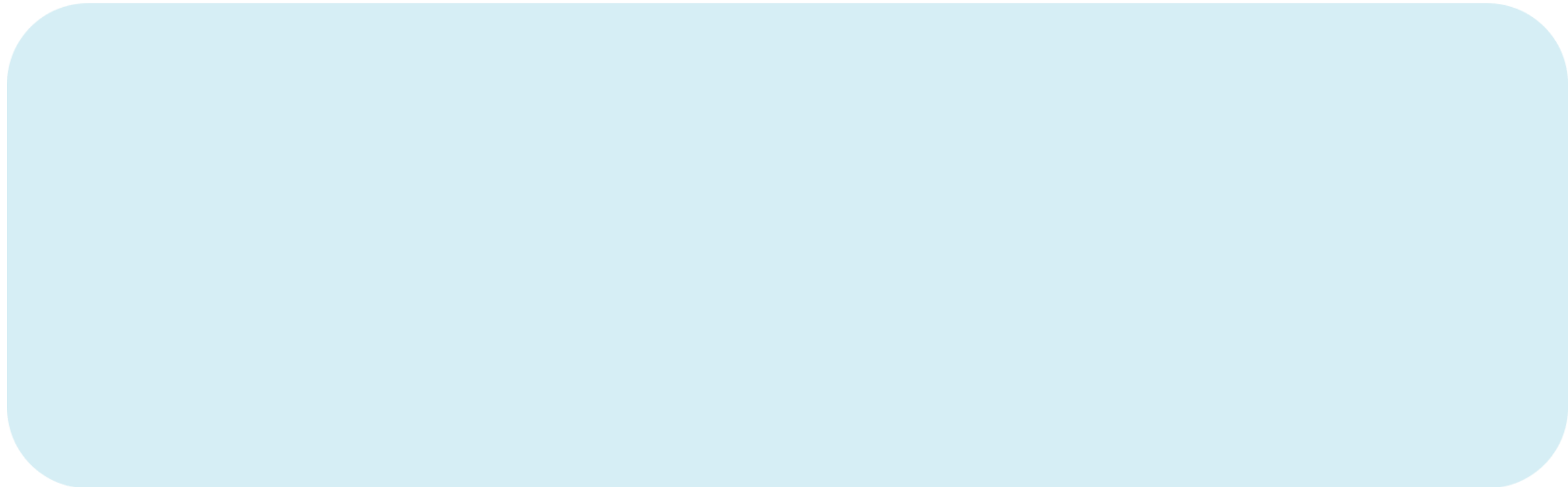
Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Understanding new datasets



# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Understanding new datasets

- E:R diagrams



# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Understanding new datasets

- E:R diagrams
- Data exploration

# Course summary

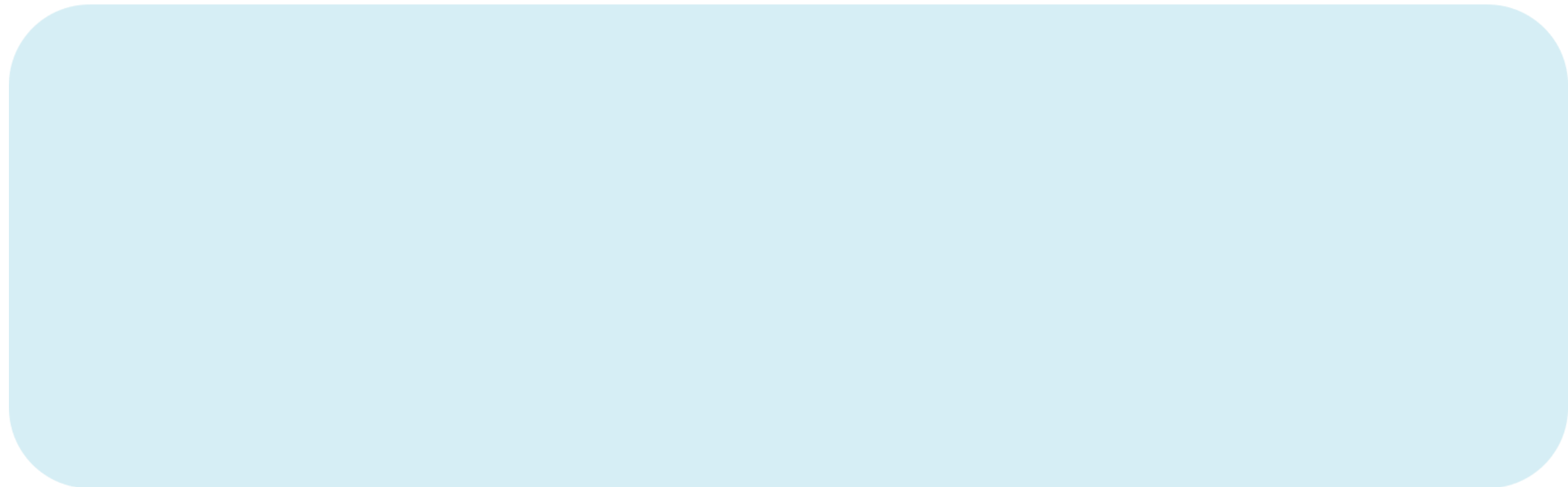
Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Building large queries



# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Building large queries

- Planning

# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Building large queries

- Planning
- Combining tables

# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Building large queries

- Planning
- Combining tables
- Creating custom fields

# Course summary

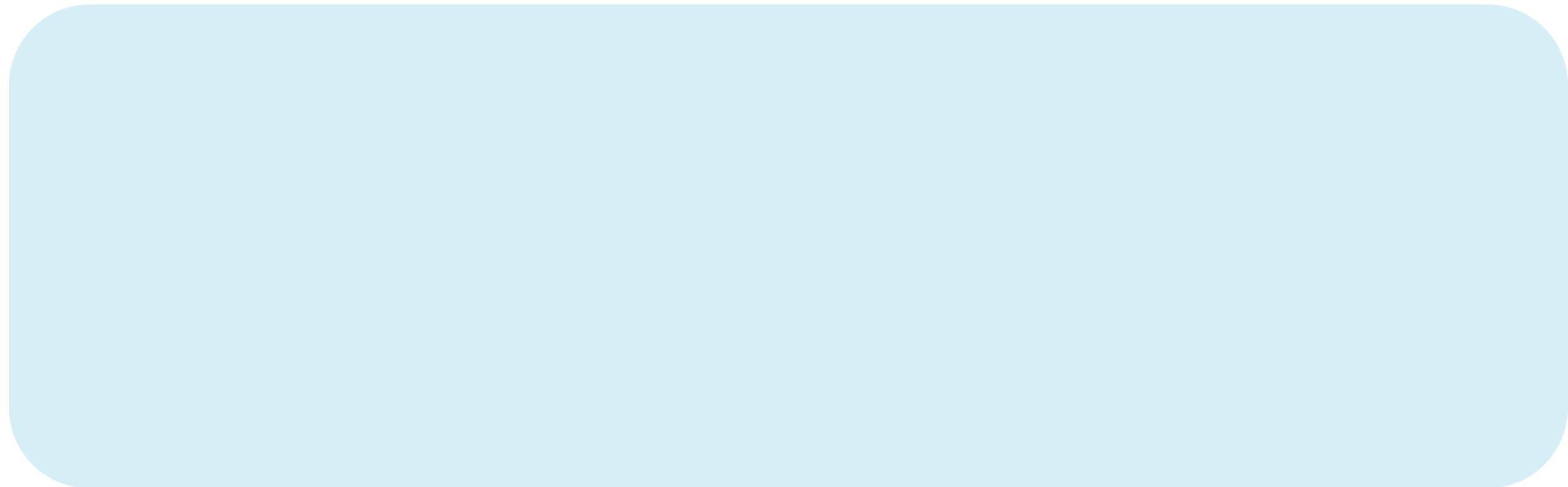
Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Cleaning data



# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Cleaning data

- Fixing data types – **CAST()**

# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Cleaning data

- Fixing data types – **CAST()**
- Parsing strings



# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Cleaning data

- Fixing data types – **CAST()**
- Parsing strings
- Handling nulls – **COALESCE()**

# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Cleaning data

- Fixing data types – **CAST()**
- Parsing strings
- Handling nulls – **COALESCE()**
- Dealing with duplication

# Course summary

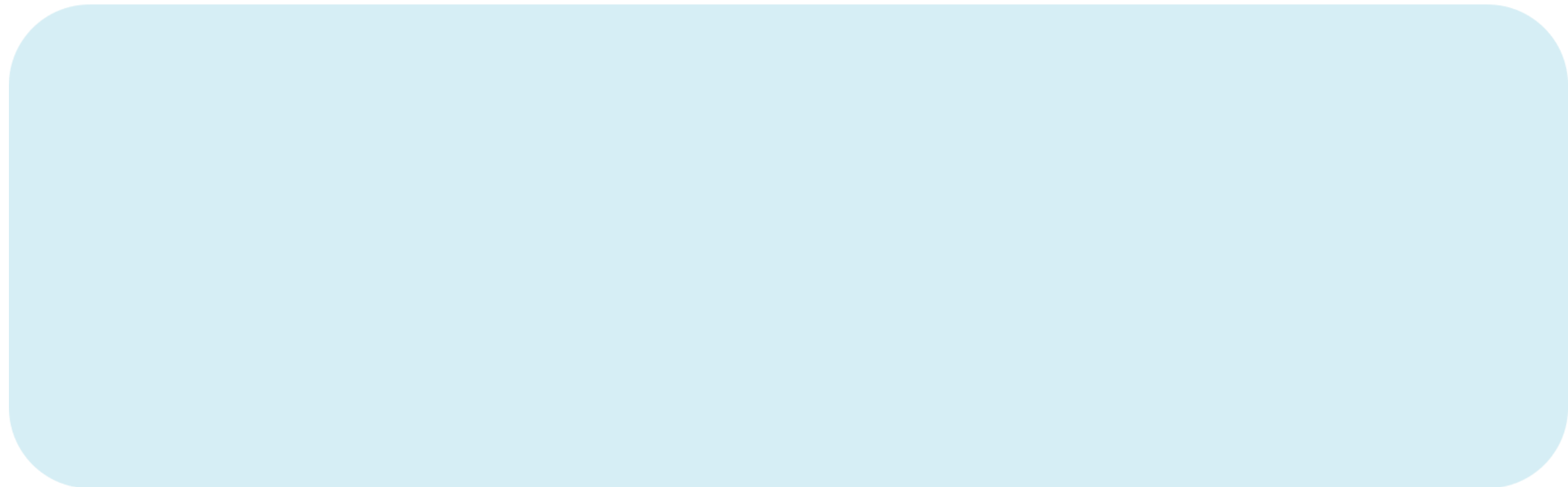
Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Complex calculations



# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Complex calculations

- Window functions

# Course summary

Chapter 1

Chapter 2

Chapter 3

Chapter 4

## Complex calculations

- Window functions
- Layered calculations

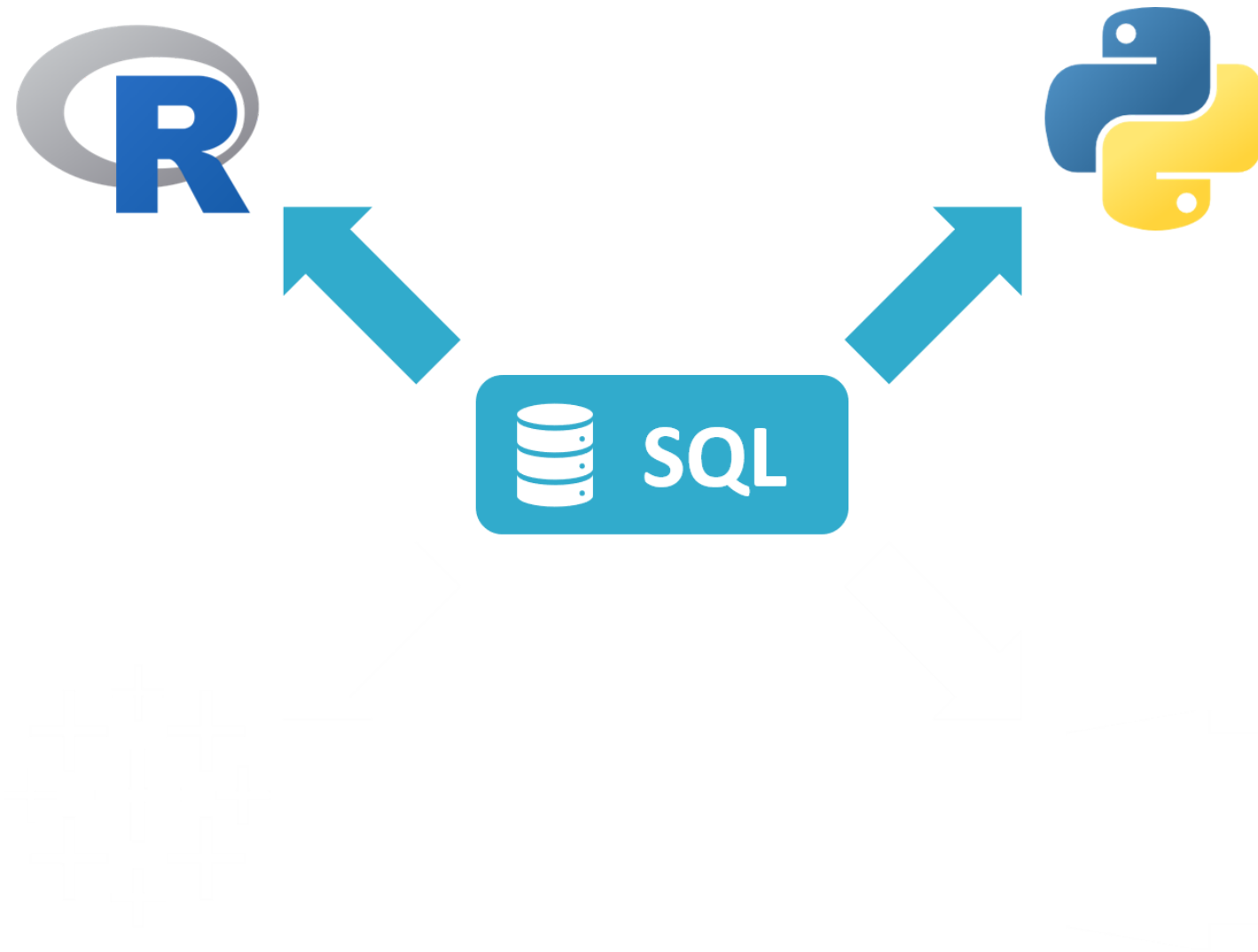
# Now what?



# Now what?

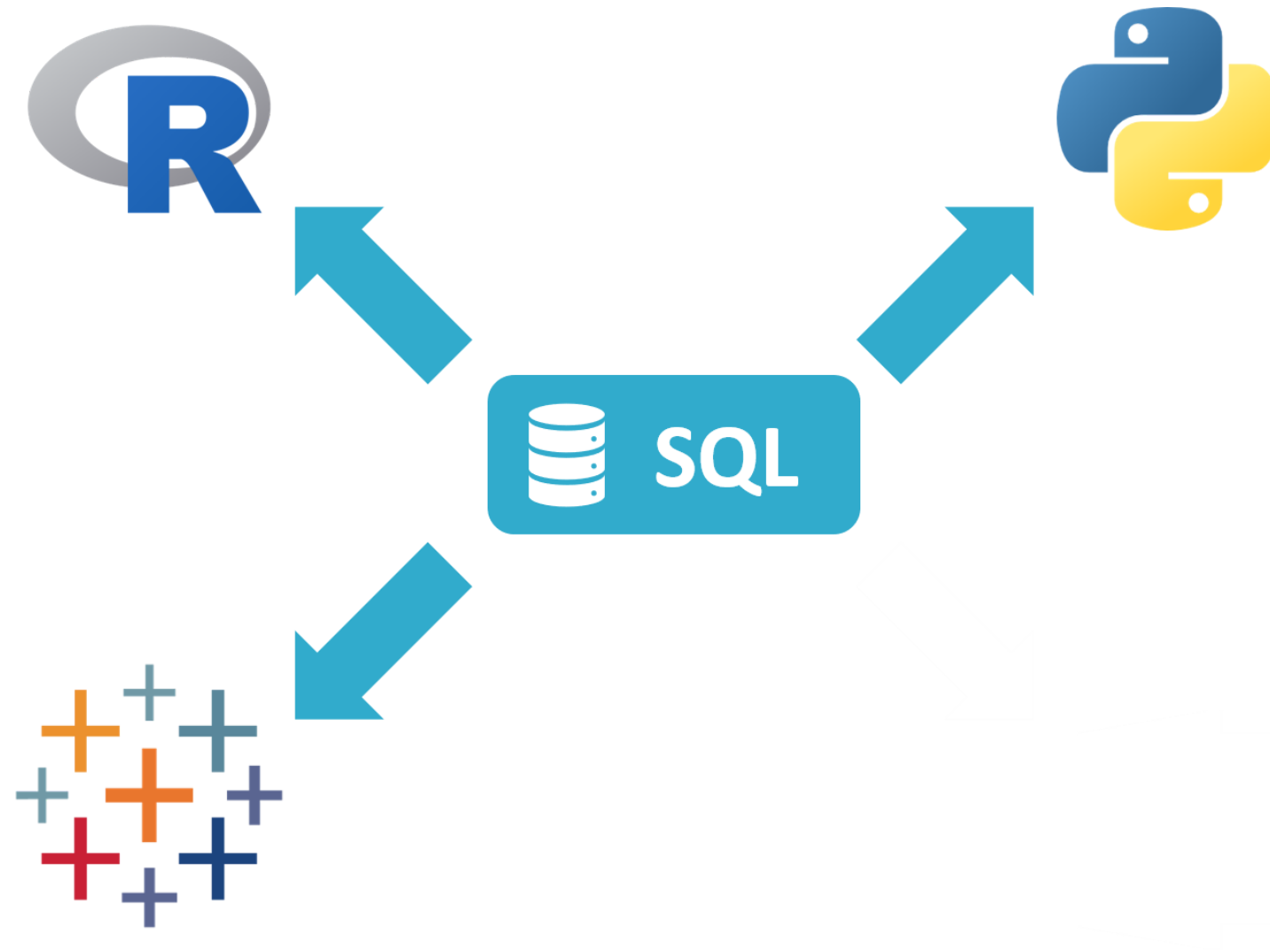


# Now what?

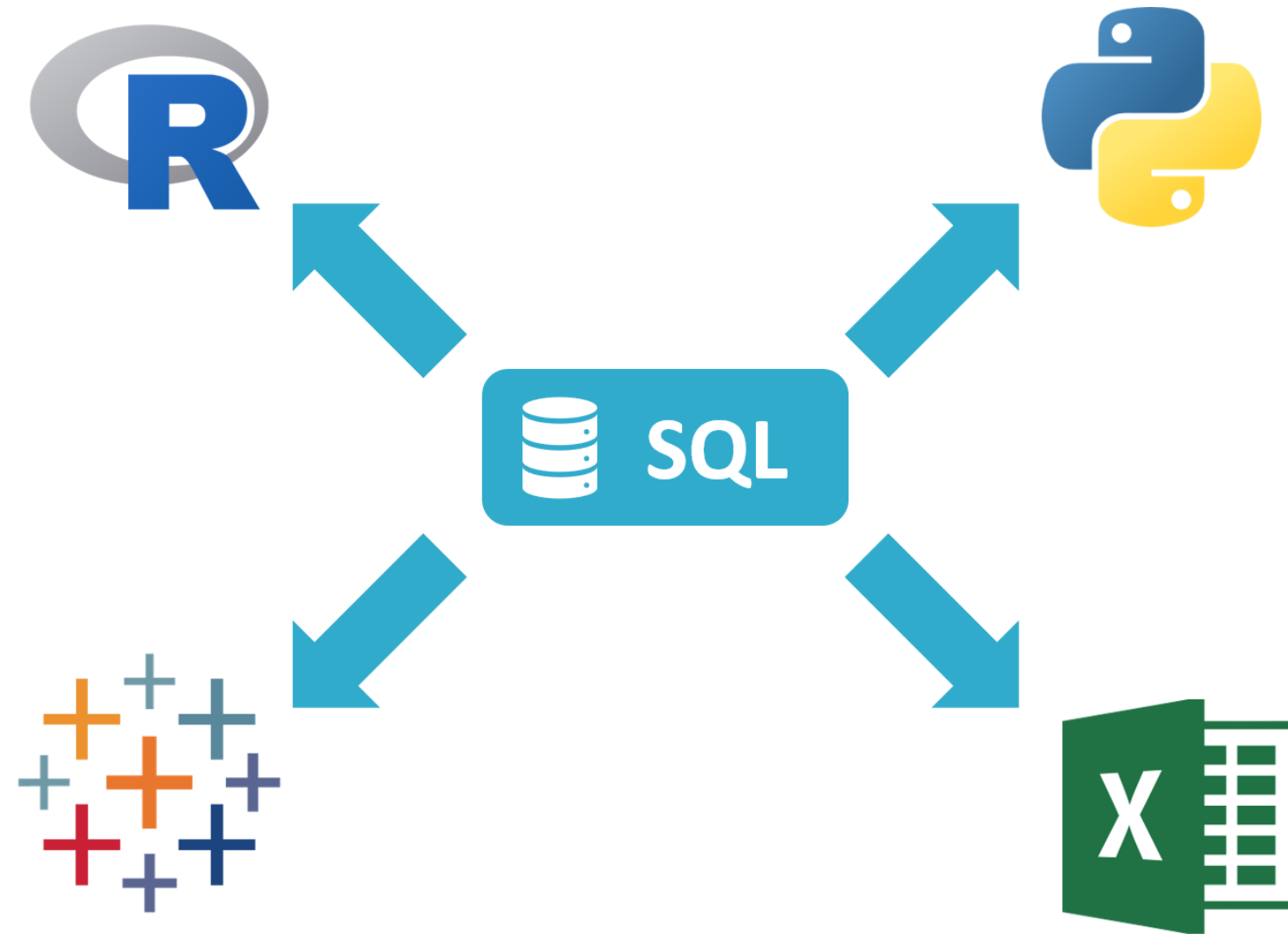




# Now what?



# Now what?



**Thank you!**  
REPORTING IN SQL