Intersects bet for Records in Common

join ( ) key ( )

Join : ① Inner join (USING)

② Case when then

③ Self join

④ Left join . (most common) <> Right join

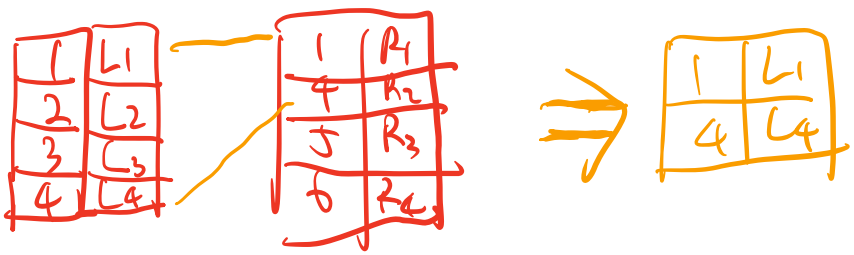
⑤ Full join

⑥ CROSS join

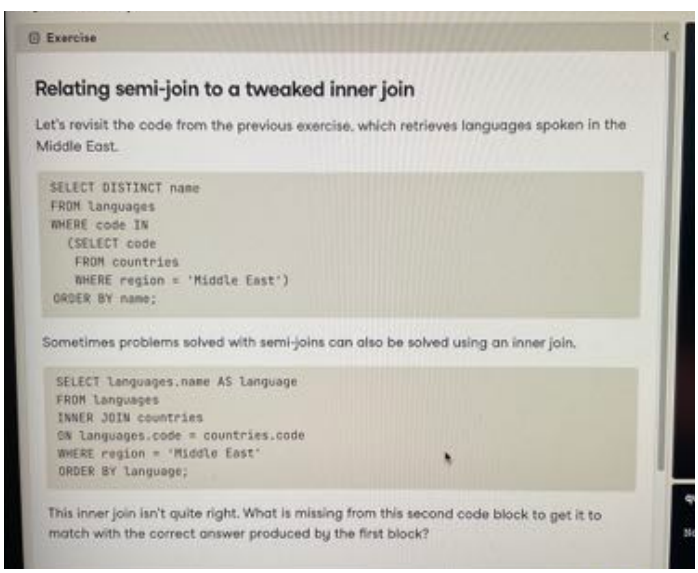
Union (all)

Intersect

⑦ semi join



⑧ anti join

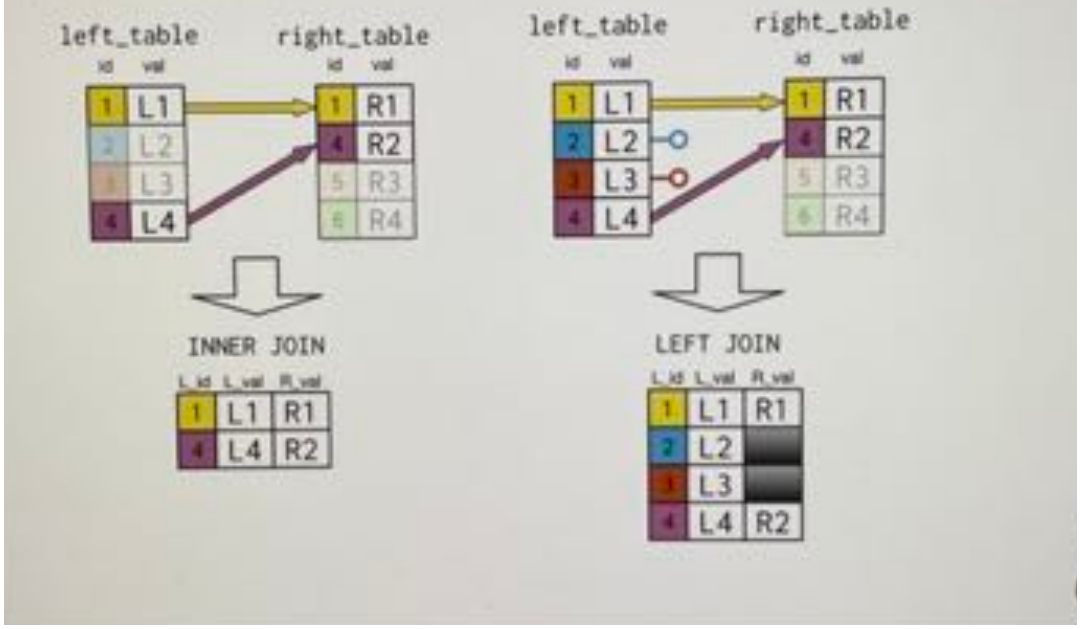


Subquery made WHERE clause.

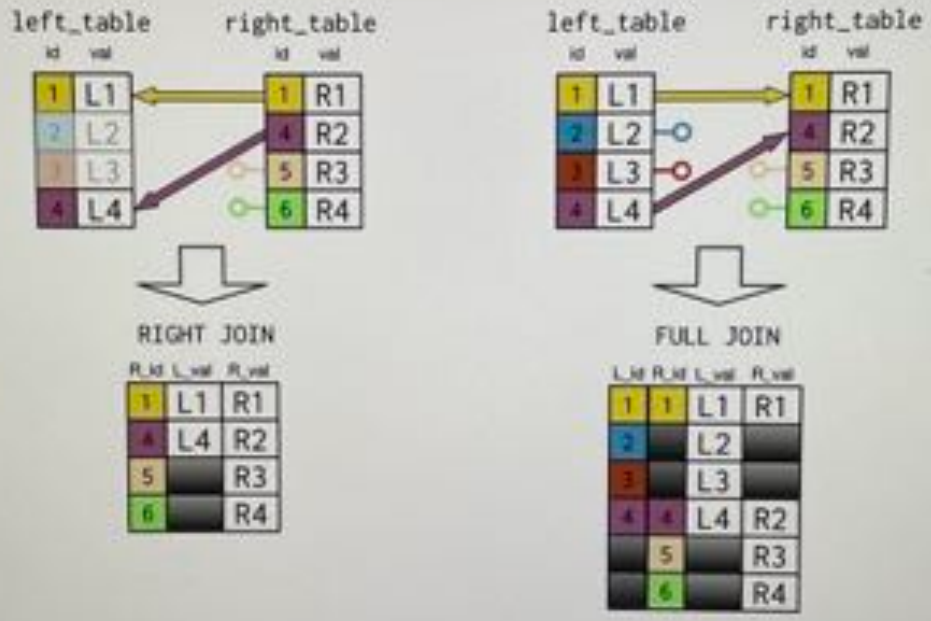
Types of joins

- INNER JOIN
  - Self-joins
- OUTER JOIN
  - LEFT JOIN
  - RIGHT JOIN
  - FULL JOIN
- CROSS JOIN
- Semi-join / Anti-join

INNER JOIN vs LEFT JOIN

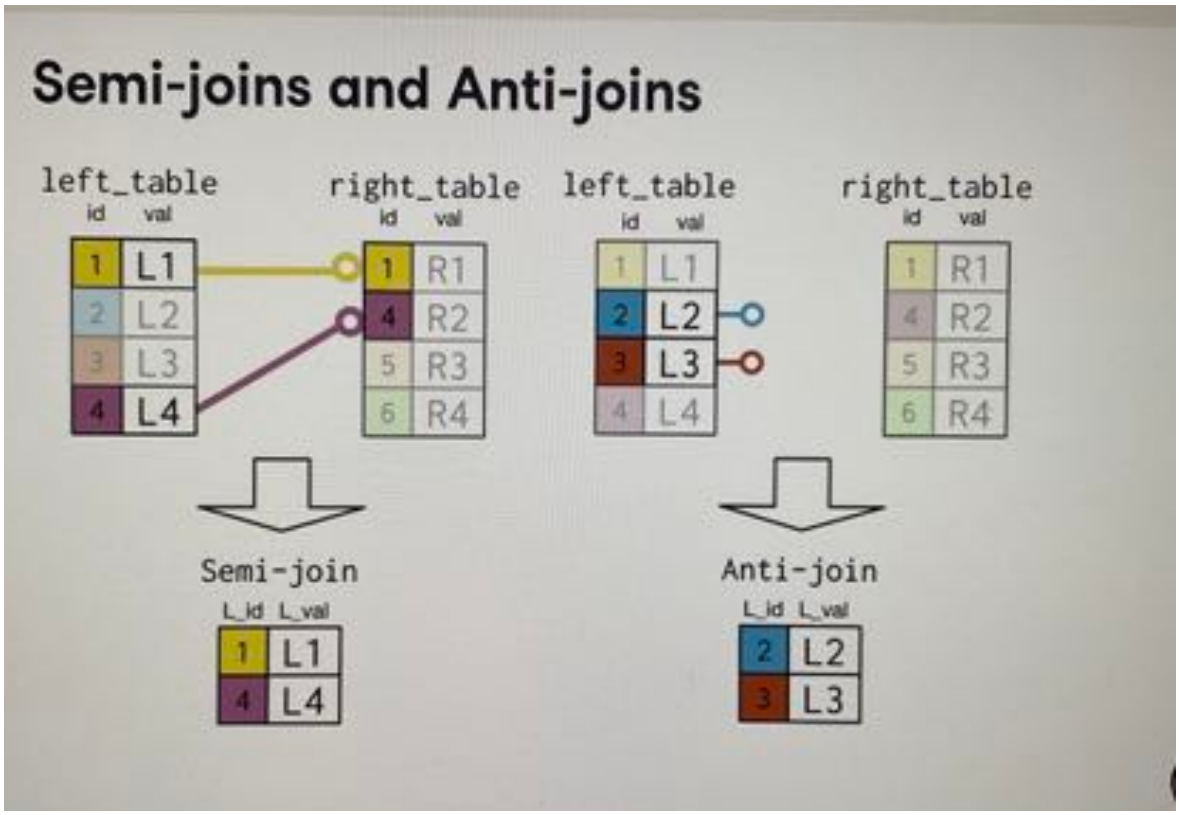


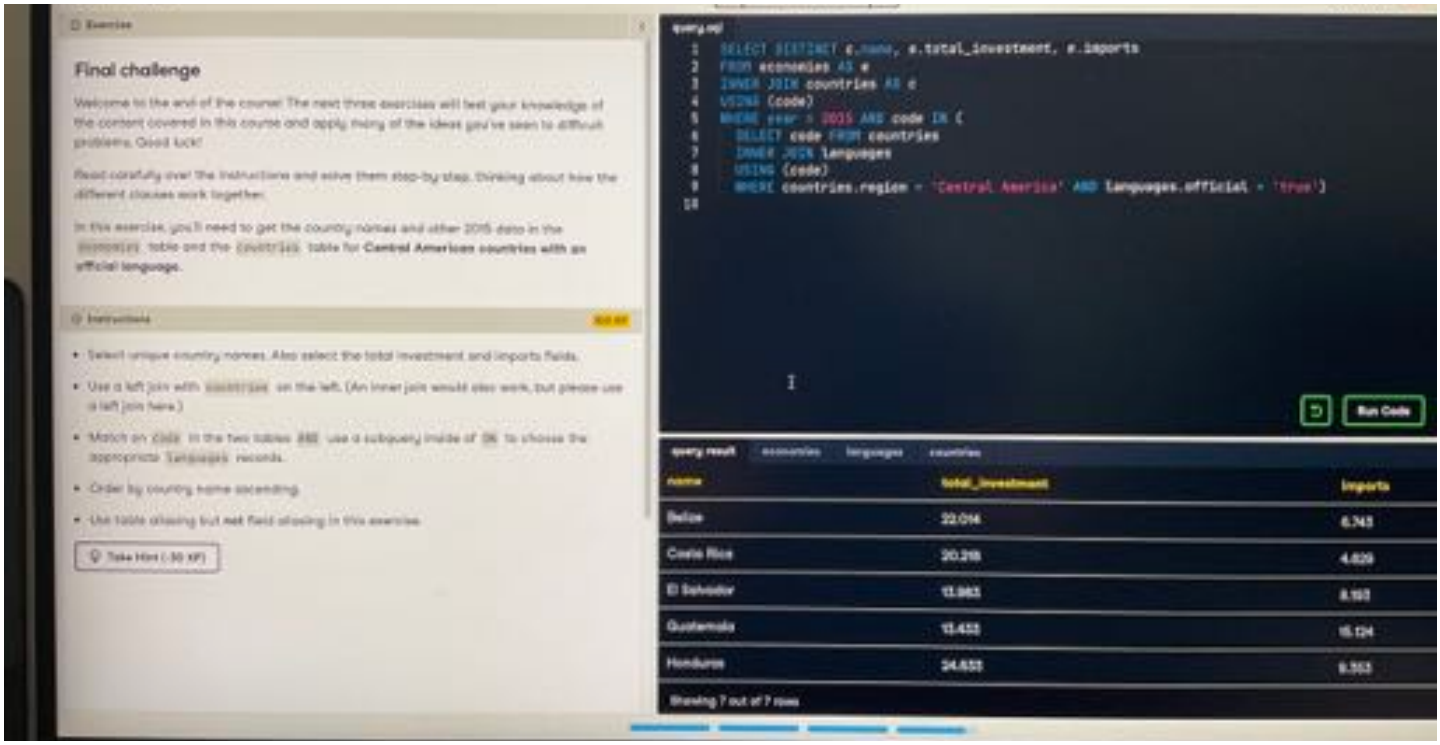
RIGHT JOIN vs FULL JOIN



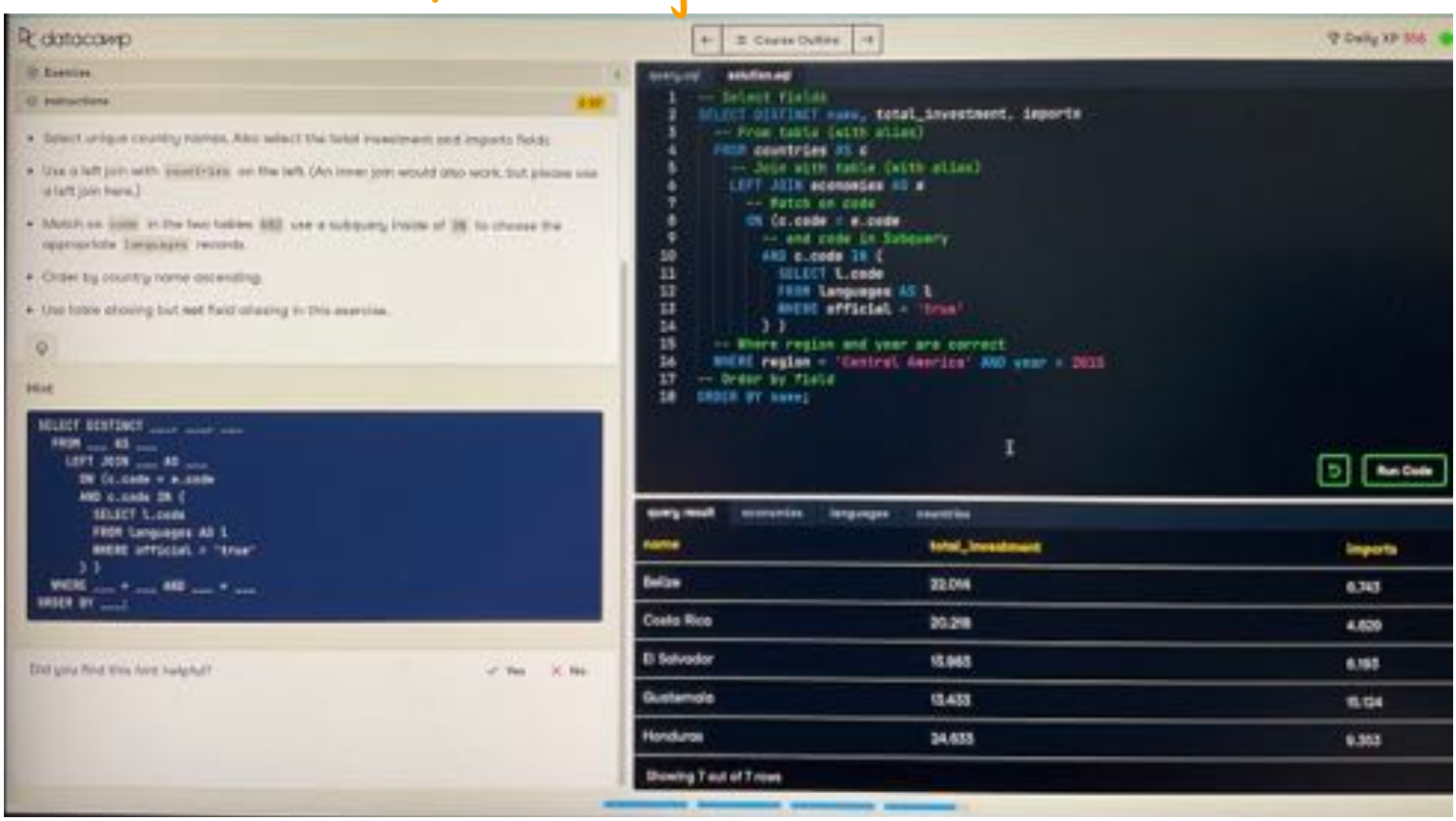
CROSS JOIN with code

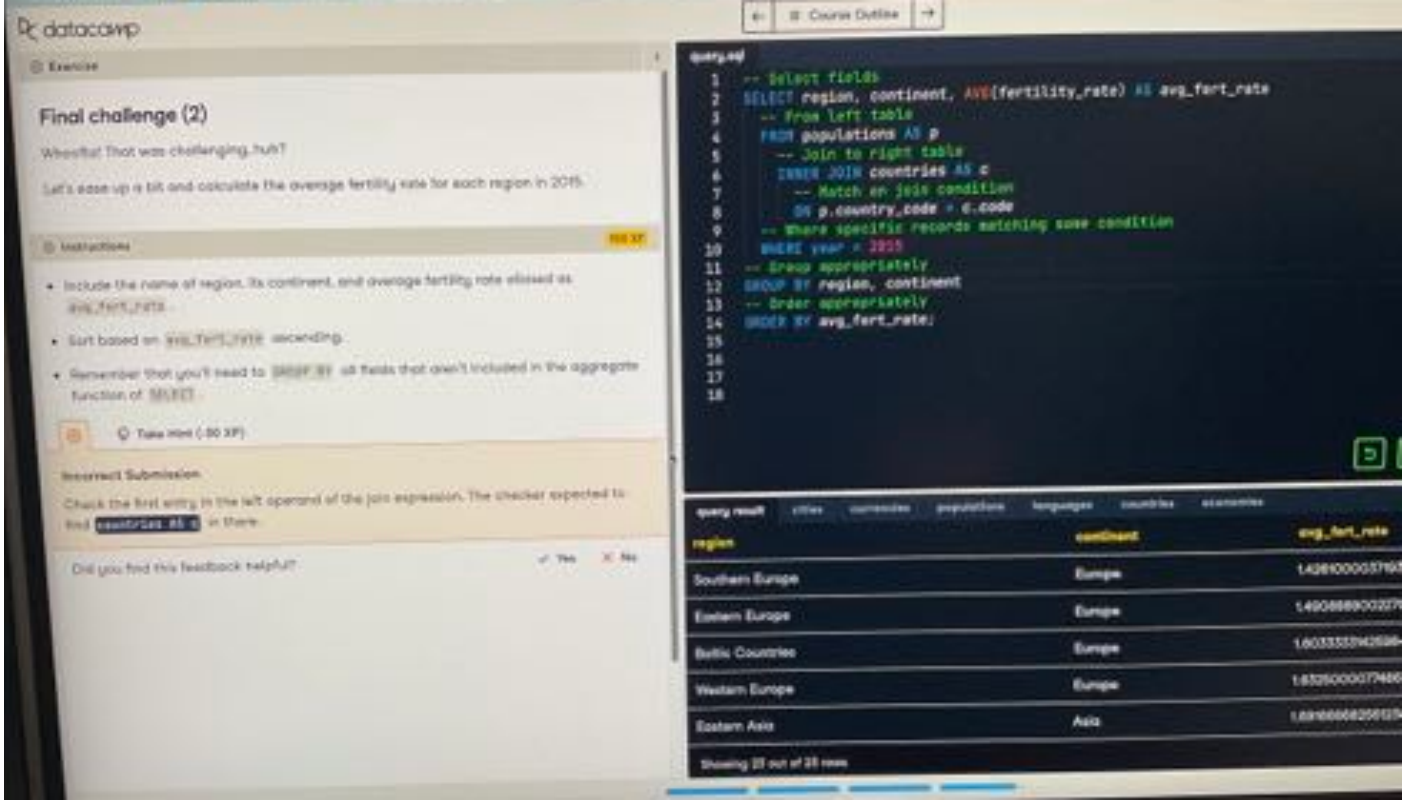

```sql
SELECT table1.id AS id1,
       table2.id AS id2
FROM table1
CROSS JOIN table2;
```

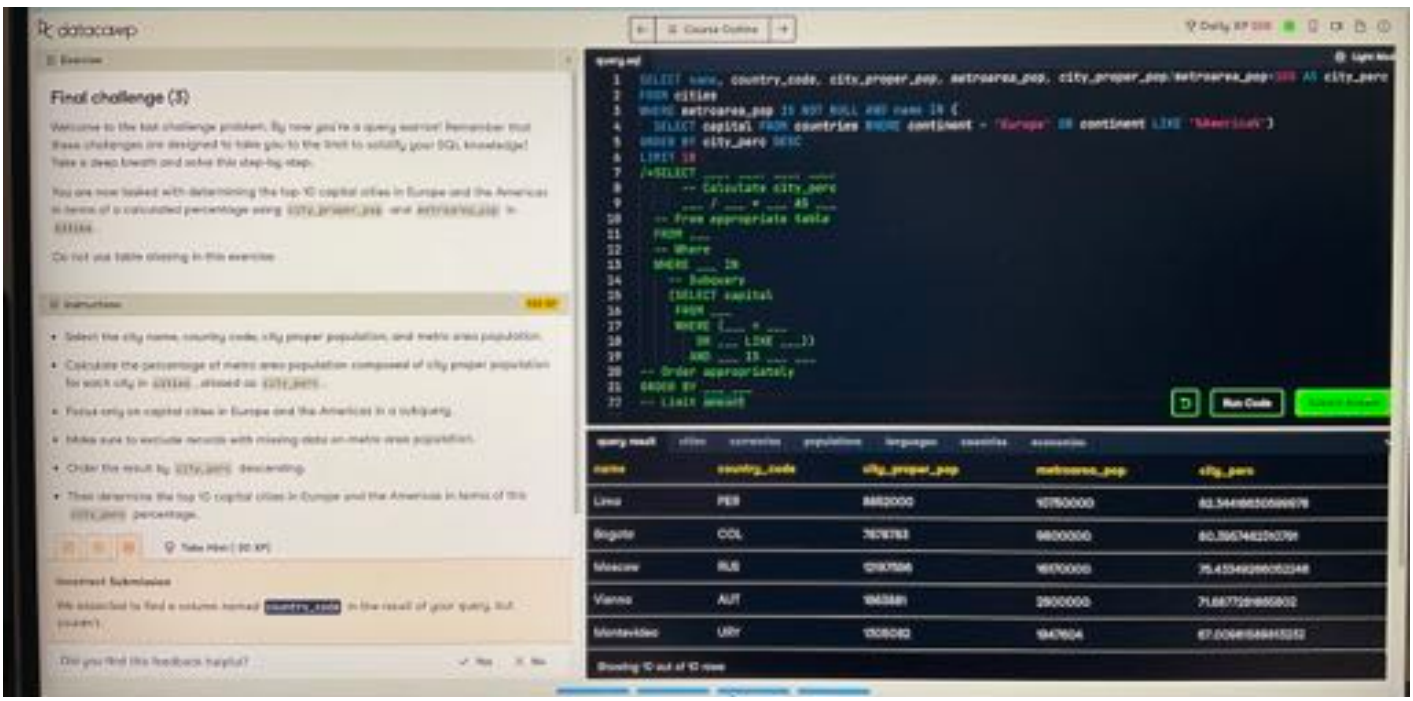

Semi-joins and Anti-joins
left_table
id val
1 L1
2 L2
3 L3
4 L4
right_table
id val
1 R1
4 R2
5 R3
6 R4
Semi-join
L_id L_val
1 L1
4 L4
left_table
id val
1 L1
2 L2
3 L3
4 L4
right_table
id val
1 R1
4 R2
5 R3
6 R4
Anti-join
L_id L_val
2 L2
3 L3


Final challenge
Welcome to the end of the course! The next three exercises will test your knowledge of the content covered in this course and apply many of the ideas you've seen to difficult problems. Good luck!
Read carefully over the instructions and solve them step-by-step, thinking about how the different clauses work together.
In this exercise, you'll need to get the country names and other 2015 data in the economies table and the countries table for Central American countries with an official language.
Instructions
Select unique country names. Also select the total investment and imports fields.
Use a left join with countries on the left. (An inner join would also work, but please use a left join here.)
Match on code. In the two tables AND use a subquery inside of ON to choose the appropriate languages records.
Order by country name ascending.
Use table aliasing but not field aliasing in this exercise.
Take Hint (-30 XP)
query.sql
SELECT DISTINCT c.name, e.total_investment, e.imports
FROM economies AS e
INNER JOIN countries AS c
USING (code)
WHERE year = 2015 AND code IN (
SELECT code FROM countries
INNER JOIN languages
USING (code)
WHERE countries.region = 'Central America' AND languages.official = 'true')
Run Code
query result
economies
languages
countries
name
total_investment
imports
Belize
22.014
0.743
Costa Rica
20.218
4.829
El Salvador
13.983
6.193
Guatemala
13.433
15.124
Honduras
24.633
9.353
Showing 7 out of 7 rows

两个结果一样


datacamp
Course Outline
Daily XP 306
Exercises
Instructions
Select unique country names. Also select the total investment and imports fields.
Use a left join with countries on the left. (An inner join would also work, but please use a left join here.)
Match on code. In the two tables AND use a subquery inside of ON to choose the appropriate languages records.
Order by country name ascending.
Use table aliasing but not field aliasing in this exercise.
Hint
SELECT DISTINCT ___, ___, ___
FROM ___ AS ___
LEFT JOIN ___ AS ___
ON c.code = e.code
AND c.code IN (
SELECT l.code
FROM languages AS l
WHERE official = 'true'
))
WHERE ___ < ___ AND ___ < ___
ORDER BY ___;
Did you find this hint helpful?
Yes
No
solution.sql
-- Select fields
SELECT DISTINCT name, total_investment, imports
-- From table (with alias)
FROM countries AS c
-- Join with table (with alias)
LEFT JOIN economies AS e
-- Match on code
ON (c.code = e.code
-- and code in Subquery
AND c.code IN (
SELECT l.code
FROM languages AS l
WHERE official = 'true'
))
-- Where region and year are correct
WHERE region = 'Central America' AND year = 2015
-- Order by field
ORDER BY name;
Run Code
query result
economies
languages
countries
name
total_investment
imports
Belize
22.014
0.743
Costa Rica
20.218
4.829
El Salvador
13.983
6.193
Guatemala
13.433
15.124
Honduras
24.633
9.353
Showing 7 out of 7 rows


datacamp
Final challenge (2)
Whoa!ba! That was challenging, huh?
Let's ease up a bit and calculate the average fertility rate for each region in 2015.
Instructions
Include the name of region, its continent, and average fertility rate aliased as avg_fert_rate.
Sort based on avg_fert_rate descending.
Remember that you'll need to GROUP BY all fields that aren't included in the aggregate function of SELECT.
Take Hint (-30 XP)
Incorrect Submission
Check the first entry in the left operand of the join expression. The checker expected to find countries AS c in there.
Did you find this feedback helpful?
Yes
No
query.sql
-- Select fields
SELECT region, continent, AVG(fertility_rate) AS avg_fert_rate
-- From left table
FROM populations AS p
-- Join to right table
INNER JOIN countries AS c
-- Match on join condition
ON p.country_code = c.code
-- Where specific records matching some condition
WHERE year = 2015
-- Group appropriately
GROUP BY region, continent
-- Order appropriately
ORDER BY avg_fert_rate;
query result
region
continent
avg_fert_rate
Southern Europe
Europe
Eastern Europe
Europe
Baltic Countries
Europe
Western Europe
Europe
Eastern Asia
Asia


datacamp
Final challenge (3)
Instructions
Select the city name, country code, city proper population, and metro area population.
Calculate the percentage of metro area population composed of city proper population for each city in cities, aliased as city_perc.
Focus only on capital cities in Europe and the Americas in a subquery.
Make sure to exclude records with missing data on metro area population.
Order the result by city_perc descending.
Then determine the top 10 capital cities in Europe and the Americas in terms of this city_perc percentage.
Take Hint (-30 XP)
Incorrect Submission
We expected to find a column named country_code in the result of your query, but couldn't.
Did you find this feedback helpful?
Yes
No
query.sql
SELECT name, country_code, city_proper_pop, metroarea_pop, city_proper_pop/metroarea_pop*100 AS city_perc
FROM cities
WHERE metroarea_pop IS NOT NULL AND name IN (
SELECT capital FROM countries WHERE continent = 'Europe' OR continent LIKE 'America%')
ORDER BY city_perc DESC
LIMIT 10
/*SELECT ___, ___, ___, ___
-- Calculate city_perc
___ / ___ * ___ AS ___
-- From appropriate table
FROM ___
-- Where
WHERE ___ IN
-- Subquery
(SELECT capital
FROM ___
WHERE (___ = ___
OR ___ LIKE ___))
AND ___ IS ___ ___
-- Order appropriately
ORDER BY ___ ___
-- Limit amount
Run Code
query result
name
country_code
city_proper_pop
metroarea_pop
city_perc
Lima
PER
8852000
10750000
Bogota
COL
7878783
9800000
Moscow
RUS
12197596
16170000
Vienna
AUT
1863881
2600000
Montevideo
URY
1305082
1947604

Simple vs. correlated subqueries

Simple Subquery

- Can be run *independently* from the main query
- Evaluated once in the whole query

Correlated Subquery

- *Dependent* on the main query to execute
- Evaluated in loops
  - **Significantly slows down query runtime**

case ~~when~~ CTE

Common Table Expressions

Common Table Expressions (CTEs)

- Table *declared* before the main query
- *Named* and *referenced* later in `FROM` statement

Setting up CTEs

```
WITH cte AS (
    SELECT col1, col2
    FROM table)
SELECT
    AVG(col1) AS avg_col
FROM cte;
```

Why use CTEs?

- Executed once
  - CTE is then stored in memory
  - Improves query performance
- Improving organization of queries
- Referencing other CTEs
- Referencing itself (`SELF JOIN`)

Differentiating Techniques

Joins

- Combine 2+ tables
  - Simple operations/aggregations

Correlated Subqueries

- Match subqueries & tables
  - Avoid limits of joins
  - **High processing time**

Multiple/Nested Subqueries

- Multi-step transformations
  - Improve accuracy and reproducibility

Common Table Expressions

- Organize subqueries sequentially
- Can reference other CTEs

Different use cases

Joins

- 2+ tables (*What is the total sales per employee?*)

Correlated Subqueries

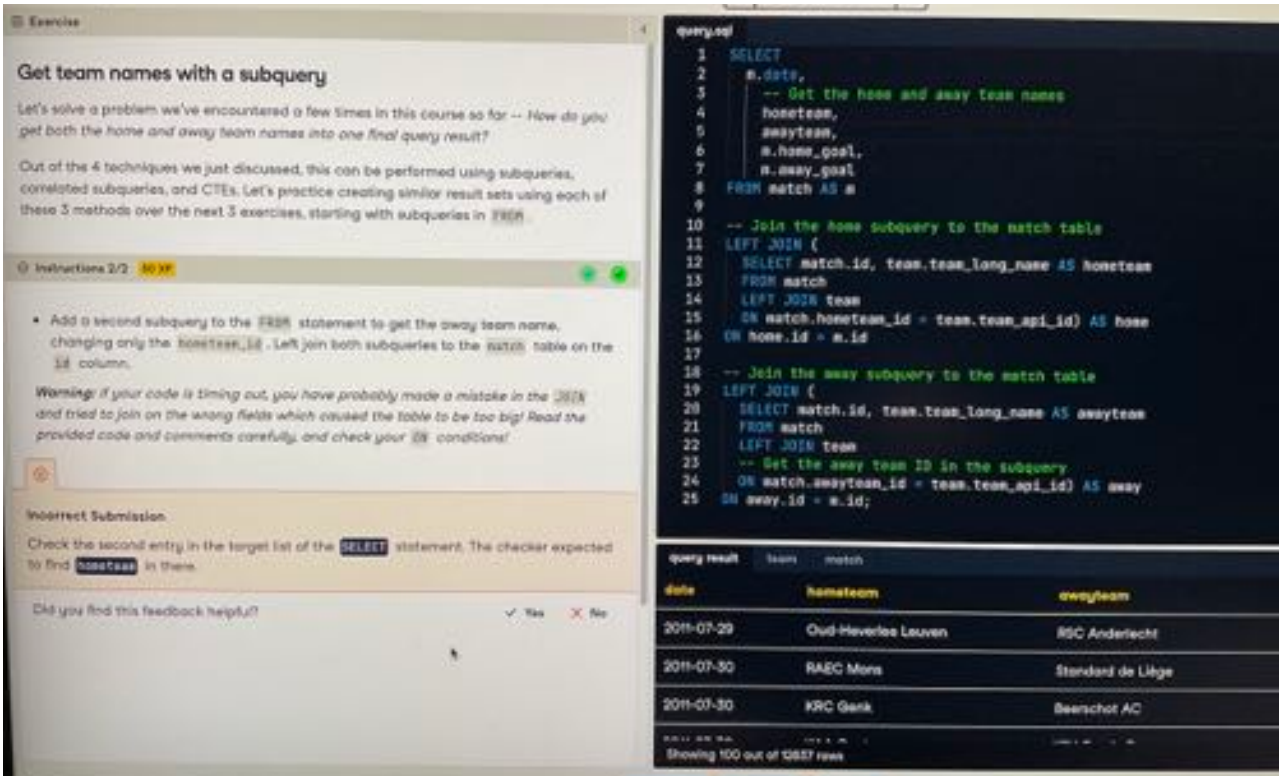
- *Who does each employee report to in a company?*

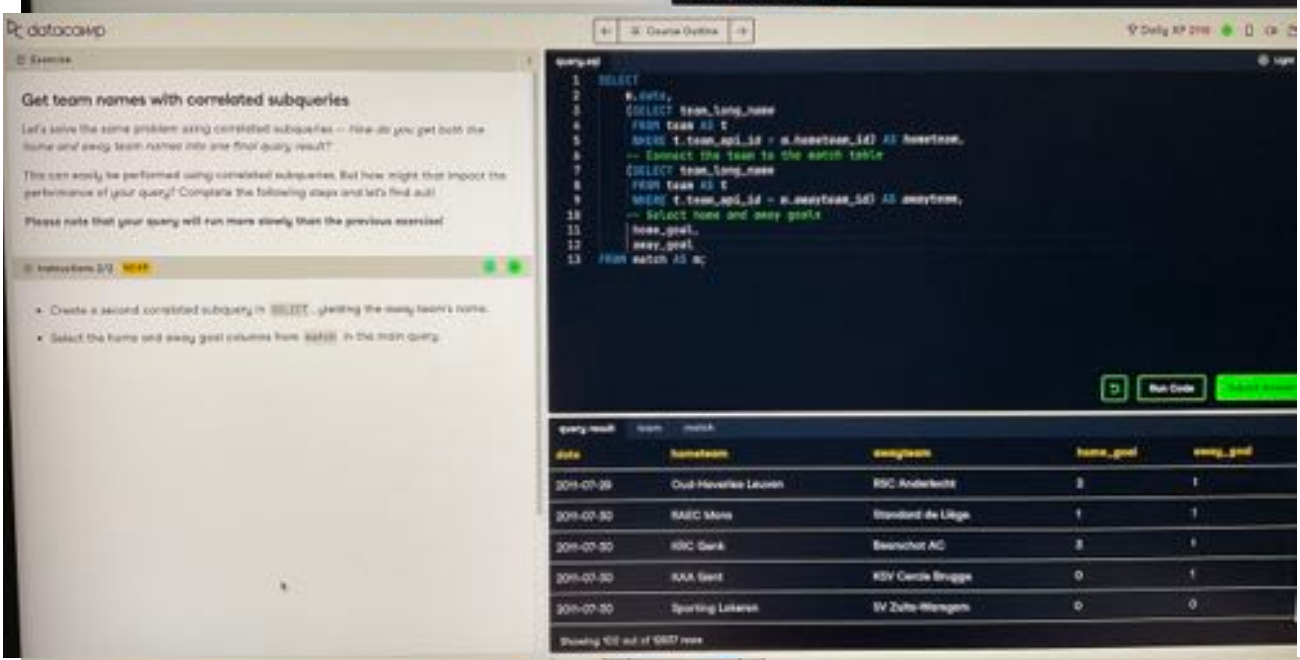
Multiple/Nested Subqueries

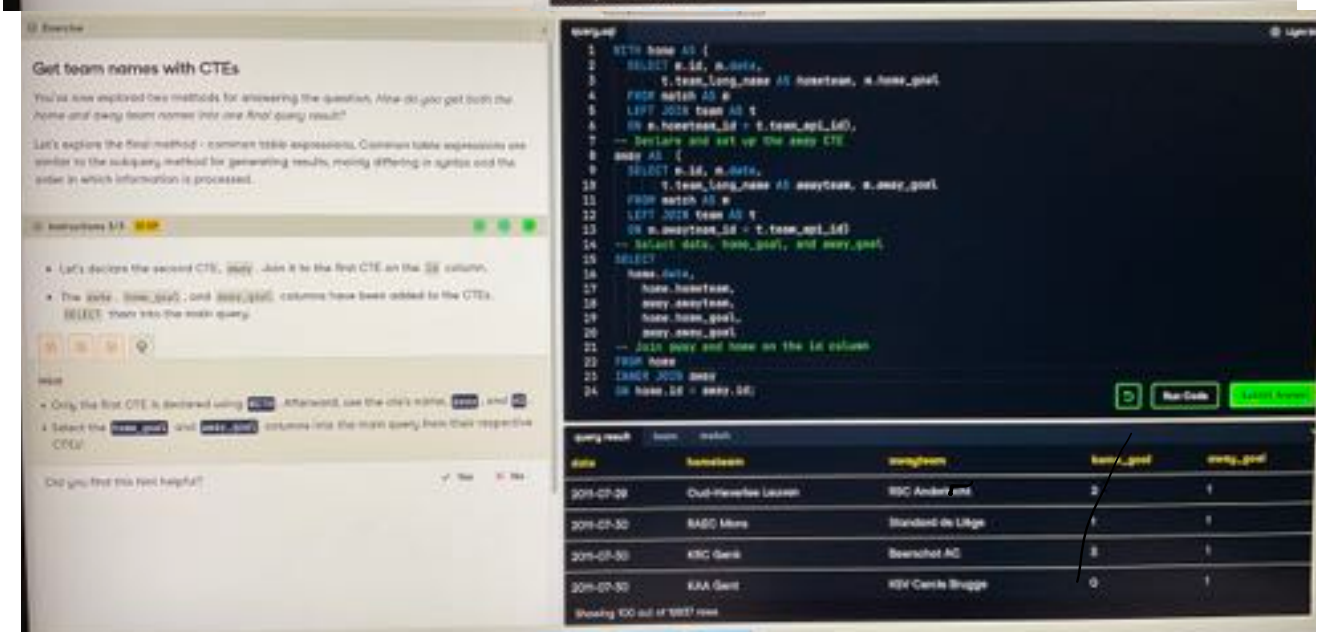
- *What is the average deal size closed by each sales representative in the quarter?*

Common Table Expressions

- *How did the marketing, sales, growth, & engineering teams perform on key metrics?*


Exercise
Get team names with a subquery
Let's solve a problem we've encountered a few times in this course so far -- How do you get both the home and away team names into one final query result?
Out of the 4 techniques we just discussed, this can be performed using subqueries, correlated subqueries, and CTEs. Let's practice creating similar result sets using each of these 3 methods over the next 3 exercises, starting with subqueries in FROM.
Instructions 2/2 50 XP
Add a second subquery to the FROM statement to get the away team name, changing only the hometeam_id. Left join both subqueries to the match table on the id column.
Warning: If your code is timing out, you have probably made a mistake in the JOIN and tried to join on the wrong fields which caused the table to be too big! Read the provided code and comments carefully, and check your ON conditions!
Incorrect Submission
Check the second entry in the target list of the SELECT statement. The checker expected to find hometeam in there.
Did you find this feedback helpful?
Yes
No
query.sql
SELECT
m.date,
-- Get the home and away team names
hometeam,
awayteam,
m.home_goal,
m.away_goal
FROM match AS m
-- Join the home subquery to the match table
LEFT JOIN (
SELECT match.id, team.team_long_name AS hometeam
FROM match
LEFT JOIN team
ON match.hometeam_id = team.team_api_id) AS home
ON home.id = m.id
-- Join the away subquery to the match table
LEFT JOIN (
SELECT match.id, team.team_long_name AS awayteam
FROM match
LEFT JOIN team
-- Get the away team ID in the subquery
ON match.awayteam_id = team.team_api_id) AS away
ON away.id = m.id;
query result
team
match
date
hometeam
awayteam
2011-07-29
Oud-Heverlee Leuven
RSC Anderlecht
2011-07-30
RAEC Mons
Standard de Liège
2011-07-30
KRC Genk
Beerschot AC
Showing 100 out of 12837 rows


datacamp
Exercise
Get team names with correlated subqueries
Let's solve the same problem using correlated subqueries -- How do you get both the home and away team names into one final query result?
This can easily be performed using correlated subqueries. But how might that impact the performance of your query? Complete the following steps and let's find out!
Please note that your query will run more slowly than the previous exercises!
Instructions 2/2 50 XP
Create a second correlated subquery in SELECT, yielding the away team's name.
Select the home and away goal columns from match in the main query.
query.sql
SELECT
m.date,
(SELECT team_long_name
FROM team AS t
WHERE t.team_api_id = m.hometeam_id) AS hometeam,
-- Connect the team to the match table
(SELECT team_long_name
FROM team AS t
WHERE t.team_api_id = m.awayteam_id) AS awayteam,
-- Select home and away goals
home_goal,
away_goal
FROM match AS m;
Run Code
Submit Answer
query result
team
match
date
hometeam
awayteam
home_goal
away_goal
2011-07-29
Oud-Heverlee Leuven
RSC Anderlecht
3
1
2011-07-30
RAEC Mons
Standard de Liège
1
1
2011-07-30
KRC Genk
Beerschot AC
3
1
2011-07-30
KAA Gent
KSV Cercle Brugge
0
1
2011-07-30
Sporting Lokeren
SV Zulte-Waregem
0
0
Showing 100 out of 12837 rows


Exercise
Get team names with CTEs
You've now explored two methods for answering the question, How do you get both the home and away team names into one final query result?
Let's explore the final method - common table expressions. Common table expressions are similar to the subquery method for generating results, mainly differing in syntax and the order in which information is processed.
Instructions 1/3 60 XP
Let's declare the second CTE, away. Join it to the first CTE on the id column.
The date, home_goal, and away_goal columns have been added to the CTEs. SELECT them into the main query.
Hint
Only the first CTE is declared using WITH. Afterward, use the cte's name, home, and AS.
Select the home_goal and away_goal columns into the main query from their respective CTEs!
Did you find this hint helpful?
Yes
No
query.sql
WITH home AS (
SELECT m.id, m.date,
t.team_long_name AS hometeam, m.home_goal
FROM match AS m
LEFT JOIN team AS t
ON m.hometeam_id = t.team_api_id),
-- Declare and set up the away CTE
away AS (
SELECT m.id, m.date,
t.team_long_name AS awayteam, m.away_goal
FROM match AS m
LEFT JOIN team AS t
ON m.awayteam_id = t.team_api_id)
-- Select date, home_goal, and away_goal
SELECT
home.date,
home.hometeam,
away.awayteam,
home.home_goal,
away.away_goal
-- Join away and home on the id column
FROM home
INNER JOIN away
ON home.id = away.id;
Run Code
Submit Answer
query result
team
match
date
hometeam
awayteam
home_goal
away_goal
2011-07-29
Oud-Heverlee Leuven
RSC Anderlecht
3
1
2011-07-30
RAEC Mons
Standard de Liège
1
1
2011-07-30
KRC Genk
Beerschot AC
3
1
2011-07-30
KAA Gent
KSV Cercle Brugge
0
1
Showing 100 out of 12837 rows

Windows Functions

What you've learned so far

- `CASE` statements
- Simple subqueries
- Nested and correlated subqueries
- Common table expressions

① ROW_NUMBERS() OVER()

② LAG(column, n) OVER(...) returns column's value at row n before current row

Current and last champions

Query

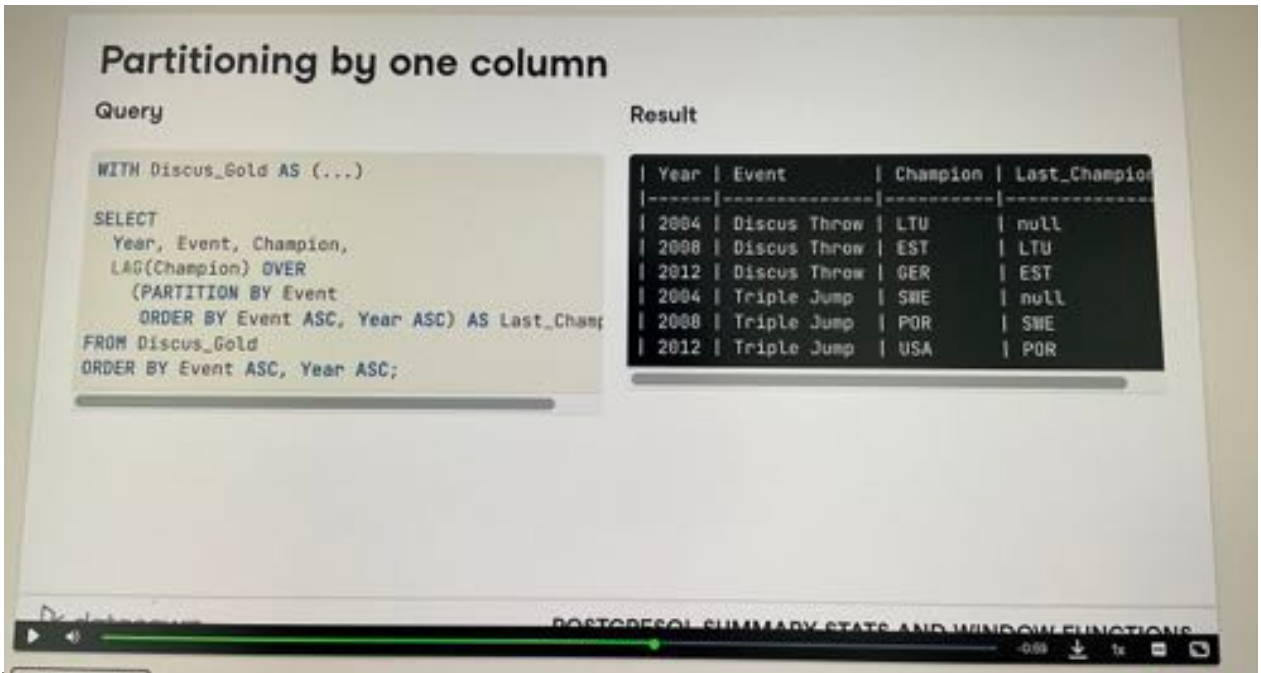
```
WITH Discus_Gold AS (
  SELECT
    Year, Country AS Champion
  FROM Summer_Medals
  WHERE
    Year IN (1996, 2000, 2004, 2008, 2012)
    AND Gender = 'Men' AND Medal = 'Gold'
    AND Event = 'Discus Throw')

SELECT
  Year, Champion,
  LAG(Champion, 1) OVER
    (ORDER BY Year ASC) AS Last_Champion
FROM Discus_Gold
ORDER BY Year ASC;
```

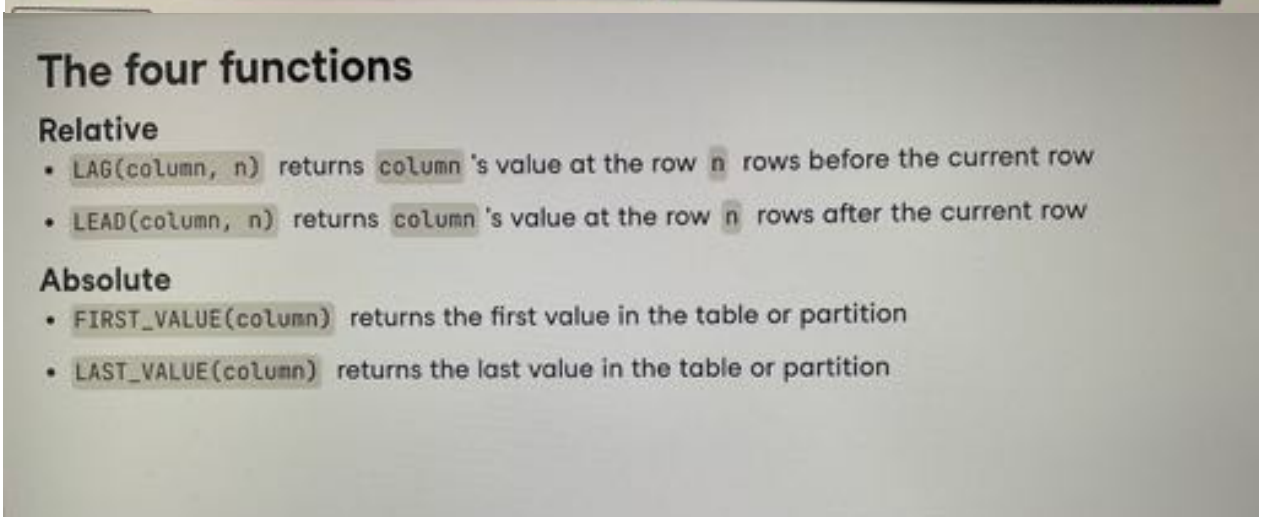
Result

Year	Champion	Last_Champion
1996	GER	null
2000	LTU	GER
2004	LTU	LTU
2008	EST	LTU
2012	GER	EST

③ PARTITION_BY (similar like group-by but the results aren't rolled into one column)

(ROW_NUMBER and LAG will Reset by partition).





Partitioning with LEAD

- `LEAD(Champion, 1)` without `PARTITION BY`

Year	Event	Champion	Next_Champion
2004	Discus Throw	LTU	EST
2008	Discus Throw	EST	GER
2012	Discus Throw	GER	SWE
2004	Triple Jump	SWE	POR
2008	Triple Jump	POR	USA
2012	Triple Jump	USA	null

- `LEAD(Champion, 1)` with `PARTITION BY Event`

Year	Event	Champion	Next_Champion
2004	Discus Throw	LTU	EST
2008	Discus Throw	EST	GER
2012	Discus Throw	GER	null
2004	Triple Jump	SWE	POR
2008	Triple Jump	POR	USA
2012	Triple Jump	USA	null

So far, you've seen the functions used over the entire table, but what happens if you want to partition?

The ranking functions

- `ROW_NUMBER()` always assigns unique numbers, even if two rows' values are the same
- `RANK()` assigns the same number to rows with identical values, skipping over the next numbers in such cases
- `DENSE_RANK()` also assigns the same number to rows with identical values, but doesn't skip over the next numbers

Query

```
WITH Country_Medals AS (
  SELECT
    Country, COUNT(*) AS Medals
  FROM Summer_Medals
  GROUP BY Country),

SELECT
  Country, Medals,
  NTILE(3) OVER (ORDER BY Medals DESC) AS Third
FROM Country_Medals;
```

Result

Country	Medals	Third
USA	4585	1
URS	2049	1
GBR	1720	1
...	...	...
CZE	56	2
LTU	55	2
...	...	...
DOM	6	3
BWI	5	3
...	...	...

MAX Window function

Query

```
WITH Brazil_Medals AS (...)

SELECT
  Year, Medals,
  MAX(Medals)
    OVER (ORDER BY Year ASC) AS Max_Medals
FROM Brazil_Medals;
```

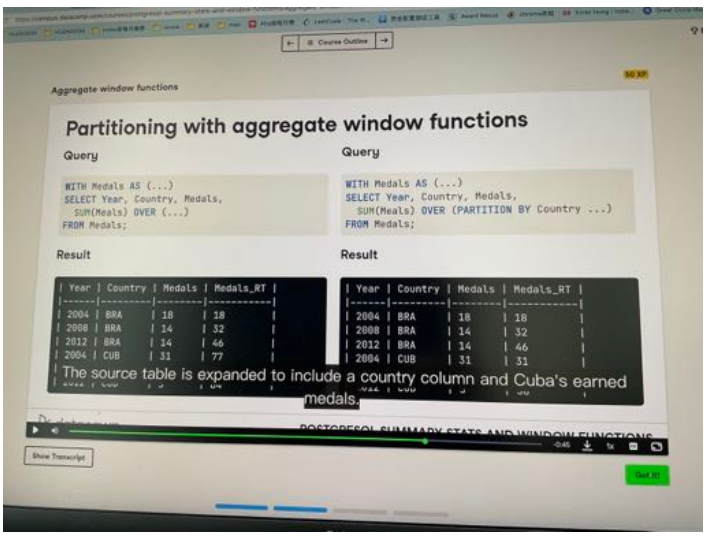
Result

Year	Medals	Max_Medals
1992	13	13
1996	5	13
2004	18	18
2008	14	18
2012	14	18

注意这里没有用range from function,所以是根据已知的找最大

So far

SUM( )



MIN( )

Frame :

- Frame: `RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`
- Without the frame, `LAST_VALUE` would return the row's value in the City column
- By default, a frame starts at the beginning of a table or partition and ends at the current

ROWS BETWEEN

- `ROWS BETWEEN [START] AND [FINISH]`
  - `n PRECEDING` : `n` rows before the current row
  - `CURRENT ROW` : the current row
  - `n FOLLOWING` : `n` rows after the current row

Examples

- `ROWS BETWEEN 3 PRECEDING AND CURRENT ROW` — 4 rows
- `ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING` — 3 rows
- `ROWS BETWEEN 5 PRECEDING AND 1 PRECEDING` — 5 rows

Overview

- Moving average (MA): Average of last `n` periods
  - **Example:** 10-day MA of units sold in sales is the average of the last 10 days' sold units
  - Used to indicate momentum/trends
  - Also useful in eliminating seasonality
- Moving total: Sum of last `n` periods
  - **Example:** Sum of the last 3 Olympic games' medals
  - Used to indicate performance; if the sum is going down, overall performance is going down

ROWS vs RANGE

- `RANGE BETWEEN [START] AND [FINISH]`
  - Functions much the same as `ROWS BETWEEN`
  - `RANGE` treats duplicates in `OVER`'s `ORDER BY` subclause as a single entity

Table

Year	Medals	Rows_RT	Range_RT
1992	10	10	10
1996	50	60	110
2000	50	110	110
2004	60	170	230
2008	60	230	230
2012	70	300	300

- `ROWS BETWEEN` is almost always used over `RANGE BETWEEN`

Pivoting

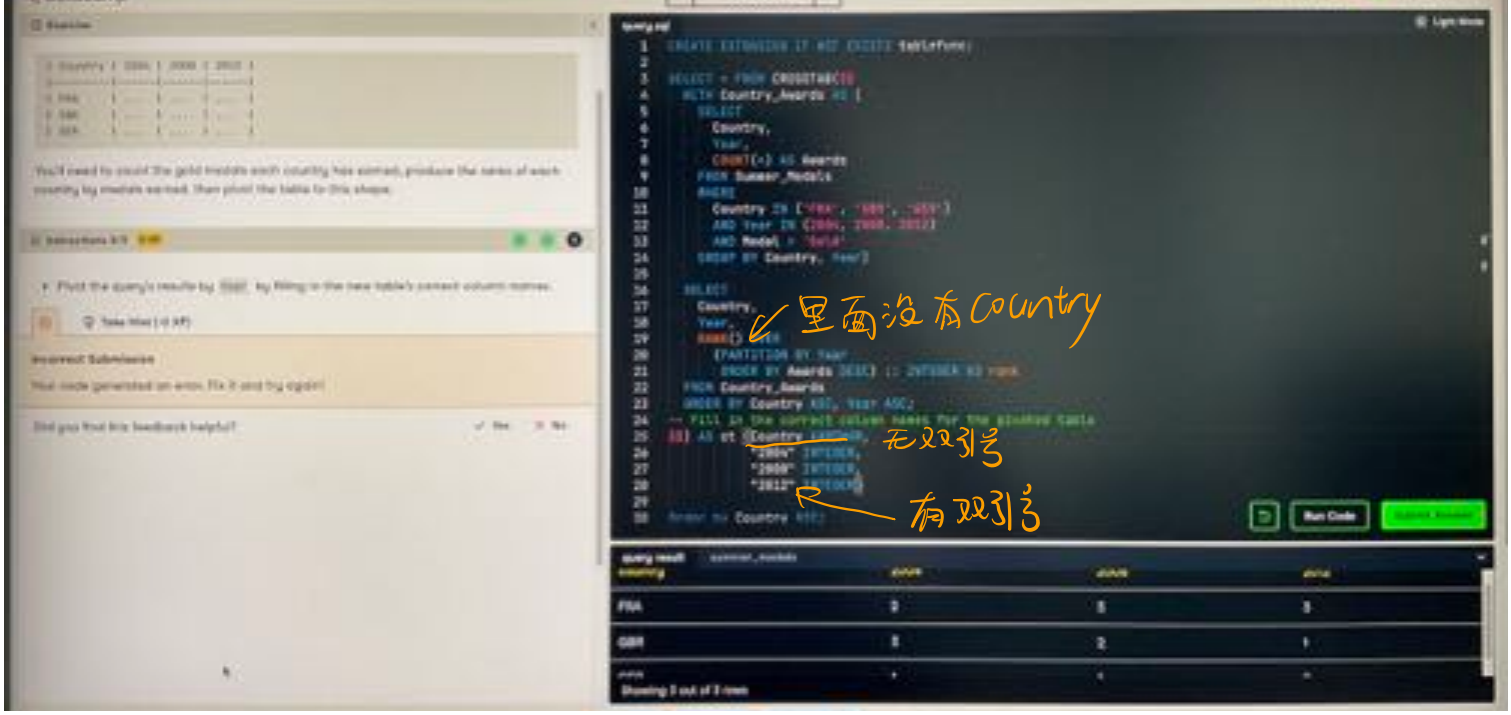
CROSSTAB

Enter CROSSTAB

```
CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT * FROM CROSSTAB($$
  source_sql TEXT
$$) AS ct (column_1 DATA_TYPE_1,
           column_2 DATA_TYPE_2,
           ...,
           column_n DATA_TYPE_N);
```

原sql语句

双引号，不能单引号



ROLLUP and CUBE

The old way

```
SELECT
  Country, Medal, COUNT(*) AS Awards
FROM Summer_Medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY Country, Medal
ORDER BY Country ASC, Medal ASC
UNION ALL

SELECT
  Country, 'Total', COUNT(*) AS Awards
FROM Summer_Medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY Country, 2
ORDER BY Country ASC;
```

更简洁.

Enter ROLLUP

```
SELECT
  Country, Medal, COUNT(*) AS Awards
FROM Summer_Medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY Country, ROLLUP(Medal)
ORDER BY Country ASC, Medal ASC;
```

- `ROLLUP` is a `GROUP BY` subclause that includes extra rows for group-level aggregations
- `GROUP BY Country, ROLLUP(Medal)` will count all `Country` - and `Medal` -level totals, then count only `Country` -level totals and fill in `Medal` with `null` s for these rows

ROLLUP - Query

```
SELECT
  Country, Medal, COUNT(*) AS Awards
FROM summer_medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY ROLLUP(Country, Medal)
ORDER BY Country ASC, Medal ASC;
```

- `ROLLUP` is hierarchical, de-aggregating from the leftmost provided column to the right-most
  - `ROLLUP(Country, Medal)` includes `Country` -level totals
  - `ROLLUP(Medal, Country)` includes `Medal` -level totals

Enter CUBE

```
SELECT
  Country, Medal, COUNT(*) AS Awards
FROM summer_medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY CUBE(Country, Medal)
ORDER BY Country ASC, Medal ASC;
```

- `CUBE` is a non-hierarchical `ROLLUP`
- It generates all possible group-level aggregations
  - `CUBE(Country, Medal)` counts `Country` -level, `Medal` -level, and grand totals

CUBE - Result

Country	Medal	Awards
CHN	Bronze	57
CHN	Gold	74
CHN	Silver	53
CHN	null	184
RUS	Bronze	56
RUS	Gold	43
RUS	Silver	44
RUS	null	143
null	Bronze	113
null	Gold	117
null	Silver	97
null	null	327

- Notice that `Medal` -level totals are included

ROLLUP vs CUBE

Source

Year	Quarter	Sales
2008	Q1	12
2008	Q2	15
2009	Q1	21
2009	Q2	27

`ROLLUP(Year, Quarter)`

Year	Quarter	Sales
2008	null	27
2009	null	48
null	null	75

`CUBE(Year, Quarter)`

Above rows + the following

Year	Quarter	Sales
null	Q1	33
null	Q2	42

这时CUBE没用
因为多出来的
2行没有用

Enter COALESCE

- `COALESCE()` takes a list of values and returns the first non-`null` value, going from left to right
- `COALESCE(null, null, 1, null, 2) ? 1`
- Useful when using SQL operations that return `null` s
  - `ROLLUP` and `CUBE`
  - Pivoting
  - `LAG` and `LEAD`

Annihilating nulls

Query

```
SELECT
  COALESCE(Country, 'Both countries') AS Country,
  COALESCE(Medal, 'All medals') AS Medal,
  COUNT(*) AS Awards
FROM summer_medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY ROLLUP(Country, Medal)
ORDER BY Country ASC, Medal ASC;
```

Result

Country	Medal	Awards
Both countries	All medals	327
CHN	All medals	184
CHN	Bronze	57
CHN	Gold	74
CHN	Silver	53
RUS	All medals	143
RUS	Bronze	56
RUS	Gold	43
RUS	Silver	44

Enter STRING_AGG

- `STRING_AGG(column, separator)` takes all the values of a column and concatenates them, with `separator` in between each value

`STRING_AGG(Letter, ', ')` transforms this...

```
| Letter |
|--------|
| A      |
| B      |
| C      |
```

...into this

```
A, B, C
```

Sakila database