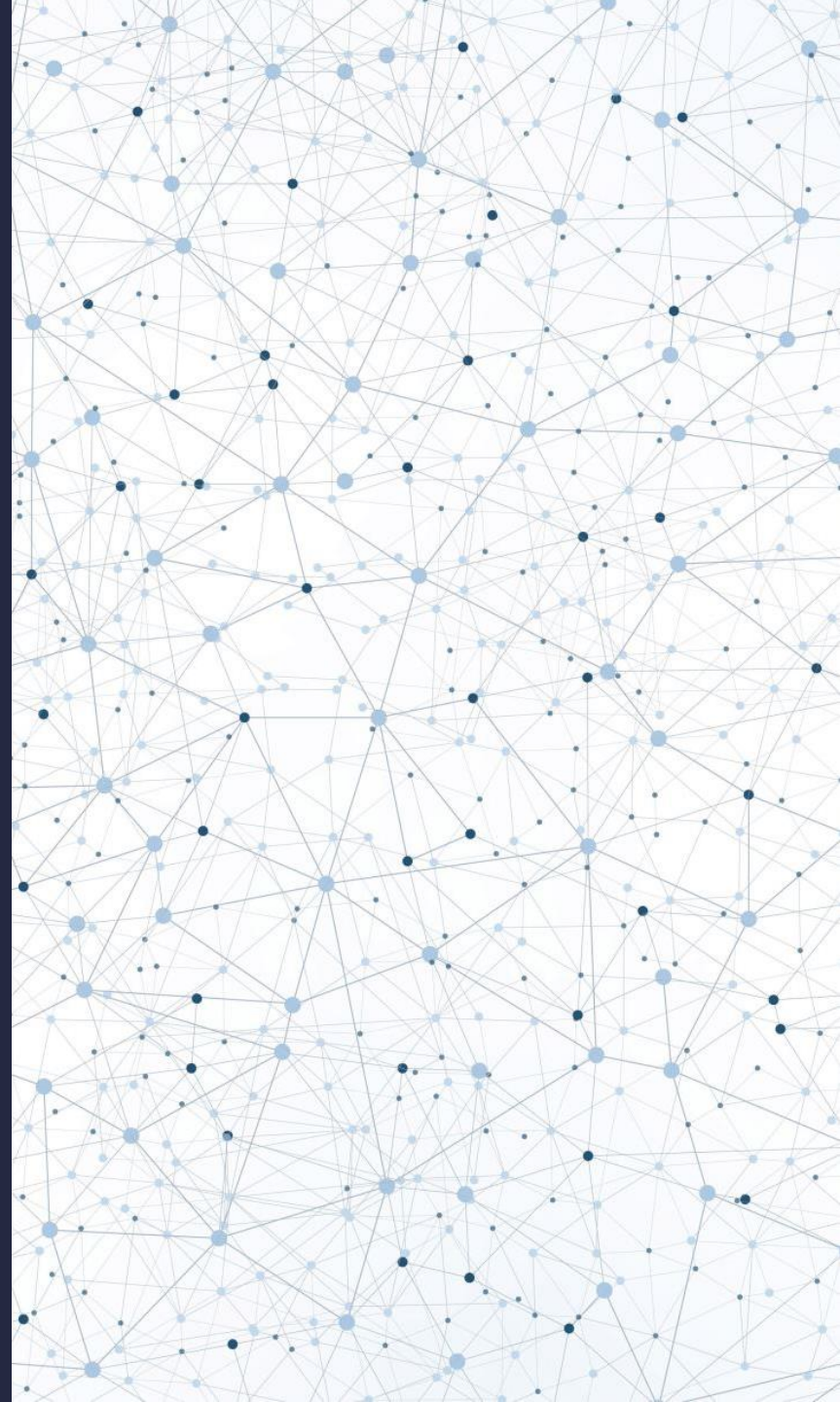
WEEK 2: SQL

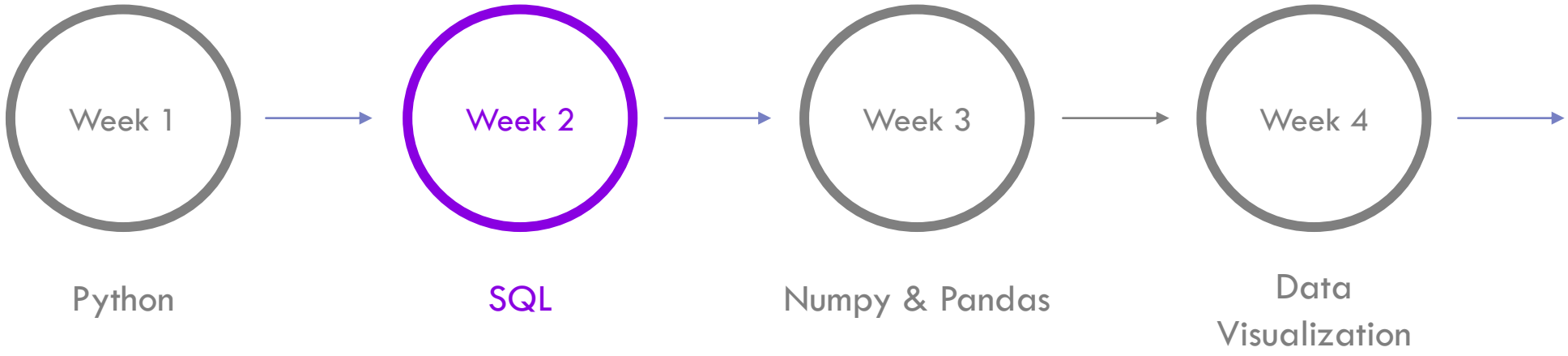
DATA SCIENCE BOOTCAMP
SPRING' 23

Instructor: Kartik Jindgar



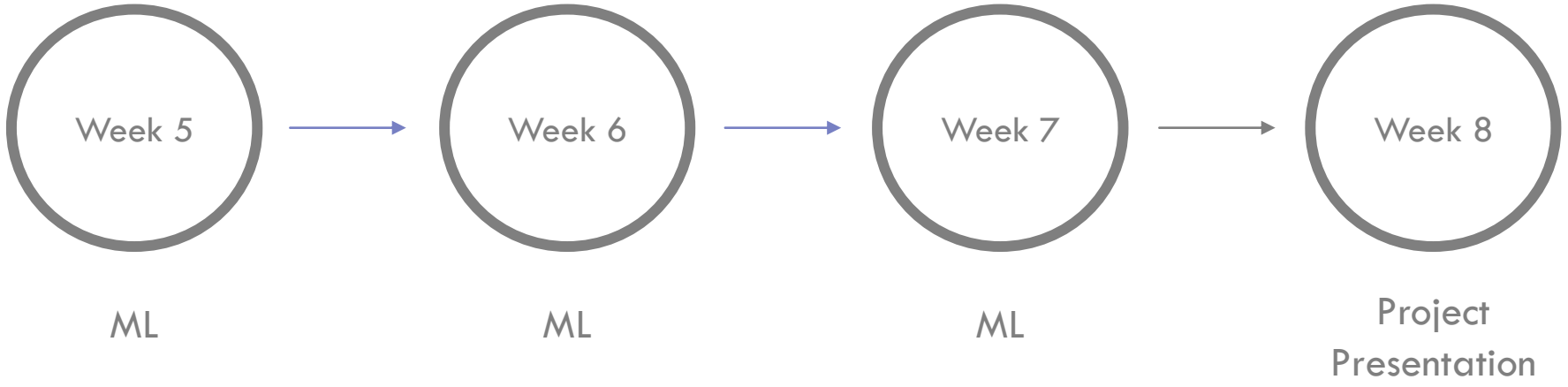
ABOUT THE BOOTCAMP

- We have an action-packed agenda!



ABOUT THE BOOTCAMP

- We have an action-packed agenda!



RESOURCES

- Review the syllabus!
- Join the SLACK community – Perfect place to ask your questions and discuss interesting problems!
- Visit the GitHub page to access ppts and jupyter notebooks discussed in each section
- You can email us at datasciencebootcamp@nyu.edu

REVIEW

- Why Python ?
- Datatypes
- Control Statements
- Loops
- Functions
- Object Oriented Programming

AGENDA

- **Fundamentals of SQL** for Data Science technical interviews

SET UP

We will be working with the following tables today

SALES
Date
Order_id
Item_id
Customer_id
Quantity
Revenue

ITEMS
Item_id
Item_name
Price
department

CUSTOMERS
Customer_id
First_name
Last_name
Address

BASIC CLAUSES

Syntax:

SELECT columns

FROM table

We can use wildcard character * to select all columns.
We can also use Limit at the end of the query to limit the number of records fetched

Eg:

```
SELECT *
FROM sales
LIMIT 10
```

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

Syntax:

SELECT columns

FROM table

WHERE condition

We can use Where clause to filter the rows that are fetched by our query
Some of the operators that we can use are – LIKE, =, >,<, IS NULL, IN, BETWEEN, etc.

Eg: Pull sample of 20 sales from 05 January 2023

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

Syntax:

SELECT columns

FROM table

WHERE condition

We can use Where clause to filter the rows that are fetched by our query
Some of the operators that we can use are – LIKE, =, >,<, IS NULL, IN, BETWEEN, etc.

Eg: Pull sample of 20 sales from 05 January 2023

```
SELECT *
FROM sales
WHERE data = "01-05-2023"
LIMIT 20
```

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

COMMON AGGREGATE FUNCTIONS

- COUNT(column)

  count all non null values in the column
  count(*) will count all rows in the table

- COUNT(DISTINCT column)

  count all distinct values in the column

- SUM(column) and AVG(column)

  calculates the sum and average of a column

- MIN(column) and MAX(column)

  computes the max and min value in a column

Syntax:

```
SELECT columns,
        aggregate_fun (column)
FROM table
WHERE condition
GROUP BY columns
```

We **must** group by all non aggregate columns

Eg: For each day in January 2023 how much revenue did we generate and how many sales did we have?

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

Syntax:

```
SELECT columns,
        aggregate_fun (column)
FROM table
WHERE condition
GROUP BY columns
```

We **must** group by all non aggregate columns

Eg: For each day in January 2023 how much revenue did we generate and how many sales did we have?

```
SELECT date,
SUM(revenue) as rev,
COUNT(distinct order_id) as sales
FROM sales
WHERE date between
"01-01-2023" and "01-31-2023"
GROUP BY date
```

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

Syntax:

```
SELECT columns
        aggregate_fun (column)
FROM table
WHERE condition
GROUP BY columns
ORDER BY columns ASC/DESC
```

We can sort the rows based on certain columns

Eg: How many items do we have in each department. Sort the departments in descending order

Items – item_id, item_name, price, department

Syntax:

```
SELECT columns
        aggregate_fun (column)
FROM table
WHERE condition
GROUP BY columns
ORDER BY columns ASC/DESC
```

We can sort the rows based on certain columns

Eg: How many items do we have in each department. Sort the departments in descending order

```
SELECT department,
COUNT(*) as items,
FROM items
GROUP BY department [or 1]
ORDER BY items [or 2]
```

Items – item_id, item_name, price, department

Syntax:

```
SELECT columns
        aggregate_fun (column)
FROM table
WHERE condition
GROUP BY columns
HAVING condition
```

The ‘Having’ clause acts like a ‘Where’ condition for your aggregate columns

Eg: Pull any order that cost at least $1000 sorted by order revenue descending.

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

Syntax:

```
SELECT columns
        aggregate_fun (column)
FROM table
WHERE condition
GROUP BY columns
HAVING condition
```

The ‘Having’ clause acts like a ‘Where’ condition for your aggregate columns

Eg: Pull any order that cost at least $1000 sorted by order revenue descending.

```
SELECT order_id,
    SUM(revenue) as rev,
FROM SALES
GROUP BY order_id [or 1]
HAVING rev >=1000
ORDER BY  rev [or 2] desc
```

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

SQL COLUMN FUNCTIONS

SQL COLUMN FUNCTIONS

- **CASE WHEN * THEN ***

  [WHEN * THEN * ELSE *] END

  An IF/THEN statement for SQL

- **CAST(column AS dtype)**

  Changes a column's datatype (int64, string, float64 are the most common dtypes)

- **UPPER() and LOWER()**

  Adjusts the case of a string field for easier string matching

- **LIKE '%string%'**

  To match on 'string' with % acting as a wildcard (this is actually a conditional, not a function)

SQL COLUMN FUNCTIONS

- **CASE WHEN * THEN ***

  [WHEN * THEN * ELSE *] END

  An IF/THEN statement for SQL

- **CAST(column AS dtype)**

  Changes a column's datatype (int64, string, float64 are the most common dtypes)

- **UPPER() and LOWER()**

  Adjusts the case of a string field for easier string matching

- **LIKE '%string%'**

  To match on 'string' with % acting as a wildcard (this is actually a conditional, not a function)

Eg: What was the average order value in 2022

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

SQL COLUMN FUNCTIONS

- **CASE WHEN * THEN *** **[WHEN * THEN * ELSE *] END**

  An IF/THEN statement for SQL

- **CAST(column AS dtype)**

  Changes a column's datatype (int64, string, float64 are the most common dtypes)

- **UPPER() and LOWER()**

  Adjusts the case of a string field for easier string matching

- **LIKE '%string%'**

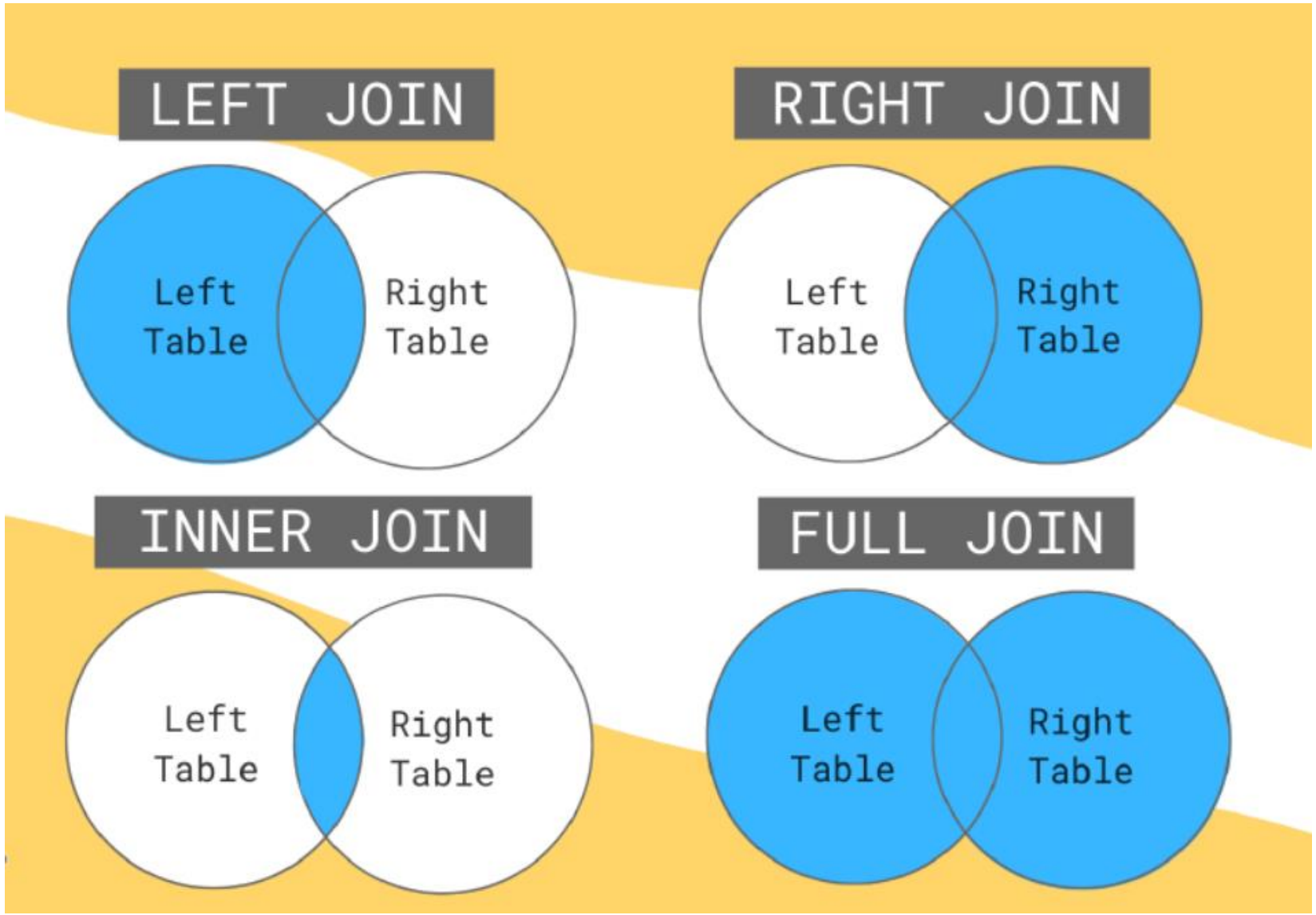
  To match on 'string' with % acting as a wildcard (this is actually a conditional, not a function)

Eg: What was the average order value in 2022

```
SELECT SUM(revenue) /
       SUM (distinct order_id) as
       revenue_per_order
FROM SALES
WHERE CAST(date AS string)
like "%-%-2022"
```

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

JOINS


LEFT JOIN
Left Table
Right Table
RIGHT JOIN
Left Table
Right Table
INNER JOIN
Left Table
Right Table
FULL JOIN
Left Table
Right Table

Syntax:

```
SELECT columns
FROM table1 as A
JOIN* table2 as B
On A.column = B.column
```

- The join key should be specified using its column name in each table.
- You can join on several keys by using AND A.key2=B.key2, etc.

Eg: How much revenue has every item we sell generated?

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue
ITEMS – Item_id, Item_name, price, department

Syntax:

```
SELECT columns
FROM table1 as A
JOIN* table2 as B
On A.column = B.column
```

- The join key should be specified using its column name in each table.
- You can join on several keys by using AND A.key2=B.key2, etc.

Eg: How much revenue has every item we sell generated?

```
SELECT i.item_id
    SUM(s.revenue) as rev,
FROM Items as i
Left Join sales as s
On i.item_id = s.sales_id
GROUP BY 1
```

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue
ITEMS – Item_id, Item_name, price, department

Syntax:

```
SELECT columns
FROM table1 as A
JOIN* table2 as B
On A.column = B.column
```

- The join key should be specified using its column name in each table.
- You can join on several keys by using AND A.key2=B.key2, etc.

Eg: How much revenue has every item we sell generated?

```
SELECT i.item_id
    COALESCE(SUM(s.revenue),0) as rev,
FROM Items as i
Left Join sales as s
On i.item_id = s.sales_id
GROUP BY 1
```

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue
ITEMS – Item_id, Item_name, price, department

SUBQUERIES

Syntax:

SELECT columns
FROM table
WHERE column_val [**<,>,IN, etc.**]
(SELECT …)

- Subqueries are SQL queries that are nested inside a larger query. They can be used in the SELECT, FROM, WHERE and/or HAVING statements.
- Typically, when using a subquery in the SELECT, WHERE or HAVING statements, the subquery must only return one value.

Eg: Pull the sales that generated more revenue than order '2567'.

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

Syntax:

```
SELECT columns
FROM table
WHERE column_val [<,>,IN, etc.]
      (SELECT ...)
```

- Subqueries are SQL queries that are nested inside a larger query. They can be used in the SELECT, FROM, WHERE and/or HAVING statements.
- Typically, when using a subquery in the SELECT, WHERE or HAVING statements, the subquery must only return one value.

Eg: Pull the sales that generated more revenue than order '2567'.

```
SELECT order_id,
       SUM(revenue) as rev
FROM sales
GROUP BY order_id
HAVING rev > (
       SELECT SUM(revenue)
       FROM sales
       WHERE order_id = '2567')
```

SALES – Date, Order_id, Item_id, Customer_id, Quantity, Revenue

THINGS TO KEEP IN MIND

TIPS

- Pay attention to the **order of tables** when you are joining them

TIPS

- Remember to **group by** every column you aren't aggregating

TIPS

- You are not required to **COALESCE** over all null values.

  We typically only coalesce numerical values when we want to capture all entries including those with no ‘value’.

TIPS

- When using a conditional for null values, you cannot use '=' and MUST use '**IS NULL**'

TIPS

- Use distinct when values might be duplicated across multiple rows.
- Don’t use distinct on a table’s key- it isn’t necessary to dedupe a key that is unique.

TIPS

- Make sure you are talking about your code and thought process while you write it!
- Interviewers want to know that you understand what you are doing!

PRACTICE QUESTIONS

WRITE A SQL QUERY TO -

- Pull total number of orders that were completed on 18th March 2023
- Pull total number of orders that were completed on 18th March 2023 with the first name ‘John’ and last name ‘Doe’
- Pull total number of customers that purchased in January 2023 and the average amount spend per customer
- Pull the departments that generated less than $600 in 2022
- What is the most and least revenue we have generated by an order
- What were the orders that were purchased in our most lucrative order

QnA