

# Digital Career Institute

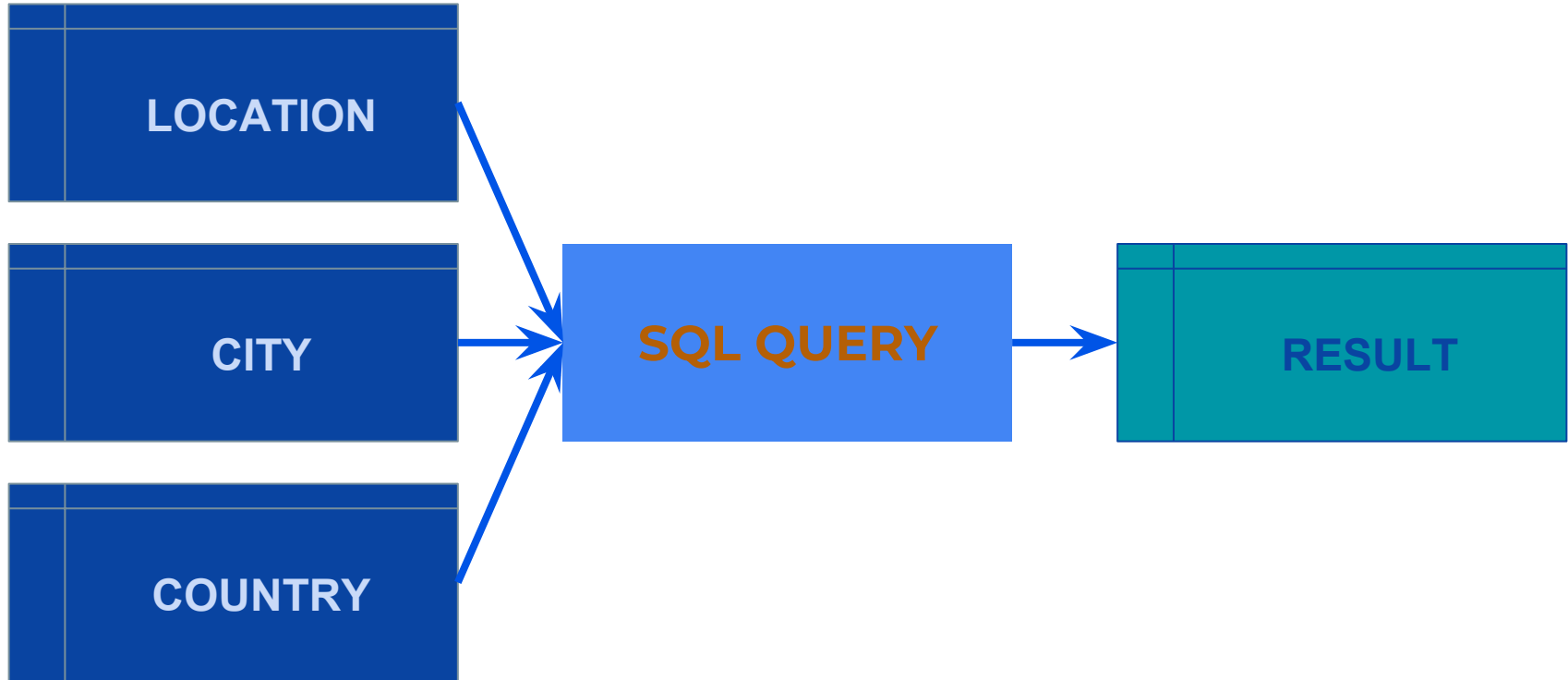
## Python Course - Advanced SQL



# Subqueries

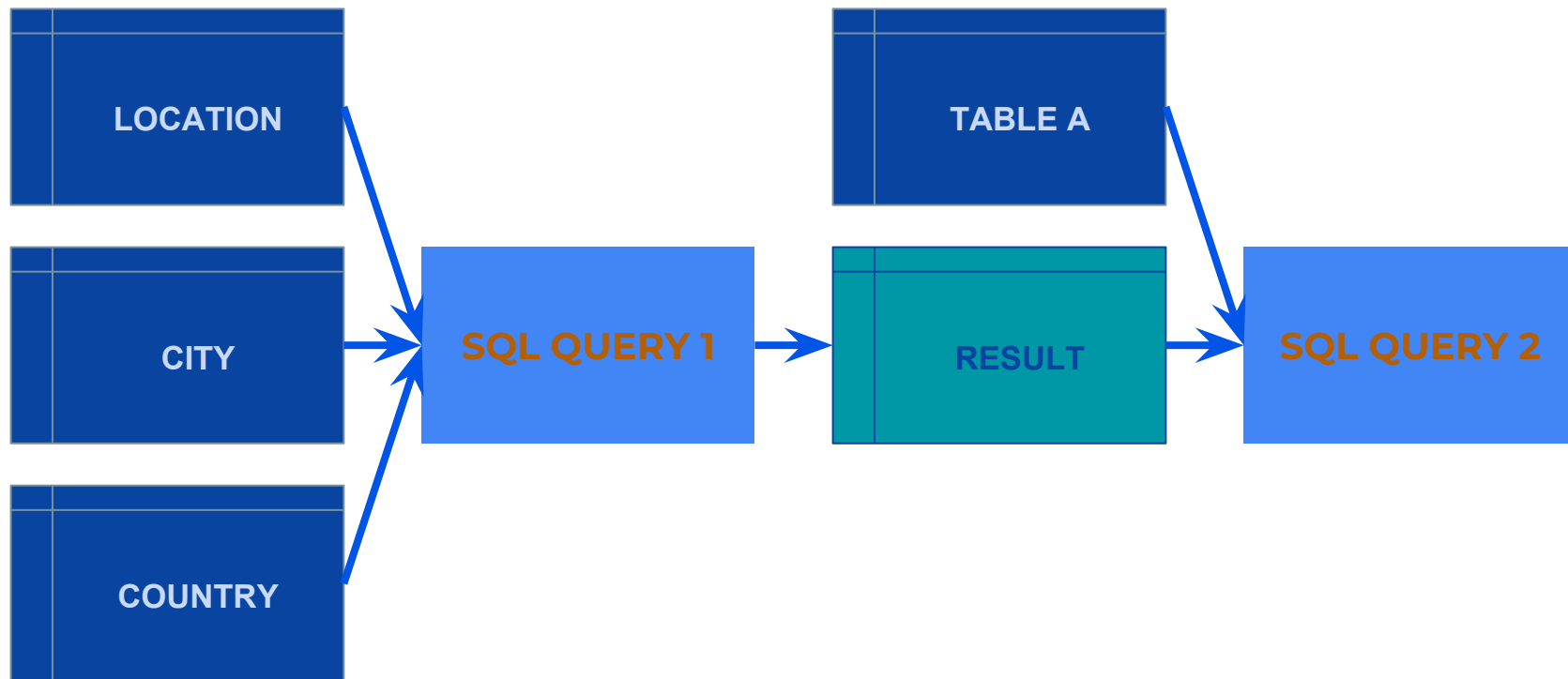
# SQL as a Function

DLI



# Using Output as Input

DLI



# Using Output as Input

DLI

```
SELECT inner_query."Location name" FROM (  
    SELECT  
        name AS "Location name",  
        type AS "Location Type",  
        quarterly_earnings AS "Earnings"  
    FROM Location  
    ) AS inner_query;
```

We can reuse the output of an SQL query as an **input table**.

All subqueries used as tables must have an alias.

```
SELECT * FROM Location  
JOIN (SELECT * FROM Location) AS duplicated  
ON duplicated.id=Location.id;
```

# Using Output as Input

We can reuse the output of an SQL query as an **input value**.

```
SELECT * FROM Location  
WHERE city_id NOT IN (SELECT id FROM City);
```

The subquery cannot return a bi-dimensional table.

It can return a list of values if the outer SQL expects a list.

```
SELECT name, (  
    SELECT name FROM City  
    WHERE City.id = Location.city_id  
) AS "City name"  
FROM Location;
```

If used in the **SELECT** section, the subquery must return a single value.

*The alias, in this case, is the name of the field in the outer output.*

# Using Output as Input

```
SELECT name, city_id, EXISTS(  
    SELECT * FROM City WHERE id = Location.city_id  
) FROM Location;
```

The **EXISTS** operator returns **True** if the subquery returns any rows and **False** otherwise.

# Aggregate Functions



Some statistics may be extracted from a dataset using **aggregate functions**.

# Aggregate Functions

name	employees	type
Location 2	4	Local office
Location 4	2	Local office
Location 5	7	Local office
Location 6		Local office
Location 3	2	Selling point
Headquarters	24	Headquarters

```
SELECT
    COUNT(employees) ,
    SUM(employees) ,
    AVG(employees) ,
    MIN(employees) ,
    MAX(employees)
FROM Location;
```

COUNT	SUM	AVG	MIN	MAX
5	39	7.8	2	24

# Special Case: COUNT

```
SELECT
    COUNT(*) ,
    COUNT(employees) ,
    COUNT(name) ,
    COUNT(DISTINCT type)
FROM Location;
```

COUNT	COUNT	COUNT	COUNT
6	5	6	3

`count(*)` counts all records.

`count(field_name)` counts only the records that are not null on the *field\_name*.

`count(DISTINCT field_name)` counts the unique values in *field\_name*.

# Special Case: AVG

DLI

```
SELECT
    AVG(employees),
    SUM(employees)::numeric/COUNT(*)
FROM Location;
```

AVG	?column?
7.8	6.5

Similarly, `avg(field_name)` produces the average considering only those records that are not null on the *field\_name*.

# Aggregating & Grouping

DLI

```
SELECT  
    COUNT(employees)  
FROM Location;
```


When we use an aggregate function like this, the engine automatically groups all records in the table together to extract the summary value.

Grouping records can also be tailored using the **GROUP BY** clause.

# Grouping Records

```
SELECT city_id  
FROM Location GROUP BY city_id;
```

name	city_id	type
Location 2	1	Local office
Location 4	3	Local office
Location 5	4	Local office
Location 6	21	Local office
Location 3	2	Selling point
Headquarters	2	Headquarters




city_id
1
3
4
21
2

# Grouping Records

```
SELECT type  
FROM Location GROUP BY type;
```

name	city_id	type
Location 2	1	Local office
Location 4	3	Local office
Location 5	4	Local office
Location 6	21	Local office
Location 3	2	Selling point
Headquarters	2	Headquarters



type
Headquarters
Local office
Selling point

# Grouping Records

```
SELECT name, city_id
FROM Location GROUP BY city_id;
ERROR: column "Location.name" must appear in the GROUP
BY clause or be used in an aggregate function
```

When using the **GROUP BY** clause, only the fields in this clause can be selected.

**Aggregate functions** can be used to extract calculations from fields not in the **GROUP BY** clause.



# Group & Aggregate

```
SELECT city_id, COUNT(*)  
FROM Location GROUP BY city_id;
```

name	city_id	type
Location 2	1	Local office
Location 4	3	Local office
Location 5	4	Local office
Location 6	21	Local office
Location 3	2	Selling point
Headquarters	2	Headquarters



city_id	count
1	1
3	1
4	1
21	1
2	2

# Group & Aggregate

```
SELECT
    type,
    SUM(employees),
    AVG(employees),
    MIN(employees),
    MAX(employees)
FROM Location GROUP BY type;
```

type	SUM	AVG	MIN	MAX
Headquarters	24	24	24	24
Selling point	2	2	2	2
Local office	15	4.33	2	7

# Join, Group & Aggregate

```
SELECT
    Country.name AS "Country",
    COUNT(*) AS "Num. locations",
    AVG(employees) AS "Avg. employees per location"
FROM Location
LEFT JOIN City on City.id = Location.city_id
LEFT JOIN Country on City.country_id = Country.id
GROUP BY Country.name
```

Country	Num. locations	Avg. employees per location
	2	7
Germany	1	2
France	1	4
USA	2	13

# Join, Group, Aggregate & Pivot

```
SELECT
    Country.name AS "Country",
    COUNT(*) AS "Num. locations",
    AVG(employees) AS "Avg. employees per location"
FROM Location
LEFT JOIN City on City.id = Location.city_id
LEFT JOIN Country on City.country_id = Country.id
WHERE Location.type = "Local office"
GROUP BY Country.name
```

Country	Num. locations	Avg. employees per location
	2	7
Germany	1	2
France	1	4

# Join, Group, Aggregate & Filter

DLI

```
SELECT
    Country.name AS "Country",
    COUNT(*) AS "Num. locations",
    AVG(employees) AS "Avg. employees per location"
FROM Location
LEFT JOIN City on City.id = Location.city_id
LEFT JOIN Country on City.country_id = Country.id
GROUP BY Country.name
HAVING COUNT(*) > 1
```

Country	Num. locations	Avg. employees per location
	2	7
USA	2	13

# Join, Group, Aggregate, Filter & Order

DLI

```
SELECT
    Country.name AS "Country",
    COUNT(*) AS "Num. locations",
    AVG(employees) AS "Avg. employees per location"
FROM Location
LEFT JOIN City on City.id = Location.city_id
LEFT JOIN Country on City.country_id = Country.id
GROUP BY Country.name
HAVING COUNT(*) > 1
ORDER BY "Avg. employees per location" DESC
```

Country	Num. locations	Avg. employees per location
USA	2	13
	2	7

# We learned ...

- That any SQL output can be used as input for another SQL.
- That we can use these subqueries both as input tables and input values.
- That we can extract some statistics with aggregate functions.
- That we can also group records having a same value in a field and we can combine this with aggregate functions to extract statistics for each group.
- That we can also use all this with join and filtering operations to perform complex analytical SQL queries.

A large group of people, mostly young adults, are posing for a group photo in a room with a projector screen in the background. They are arranged in several rows, with some people sitting on the floor in the front. Many of them are making peace signs or other celebratory gestures. The text "THANK YOU" is overlaid in large white letters in the center of the image.

# THANK YOU

Contact Details  
DCI Digital Career Institute gGmbH