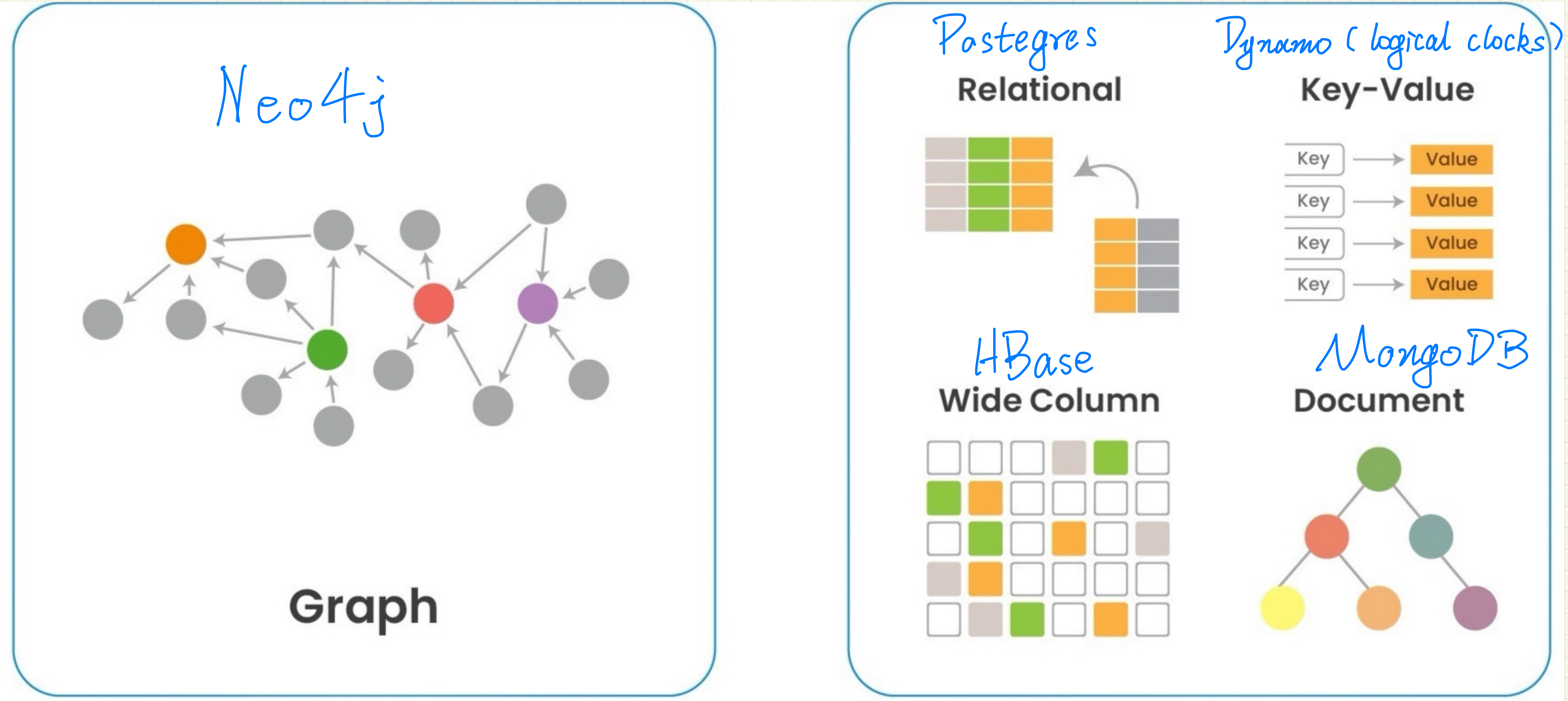
Exercise Session 12 : Graph Databases & SQL Grouping

① Graph Databases (Neo4j)
- Data model (vs. RDBMS)
- System design
- Querying (Cypher)
- Indexing (vs. RDBMS)
- Grouping

② RDF

③ Grouping Options in SQL
- Aggregation leveling
- Grouping sets
- Cube
- Roll up



Neo4j

Postegres

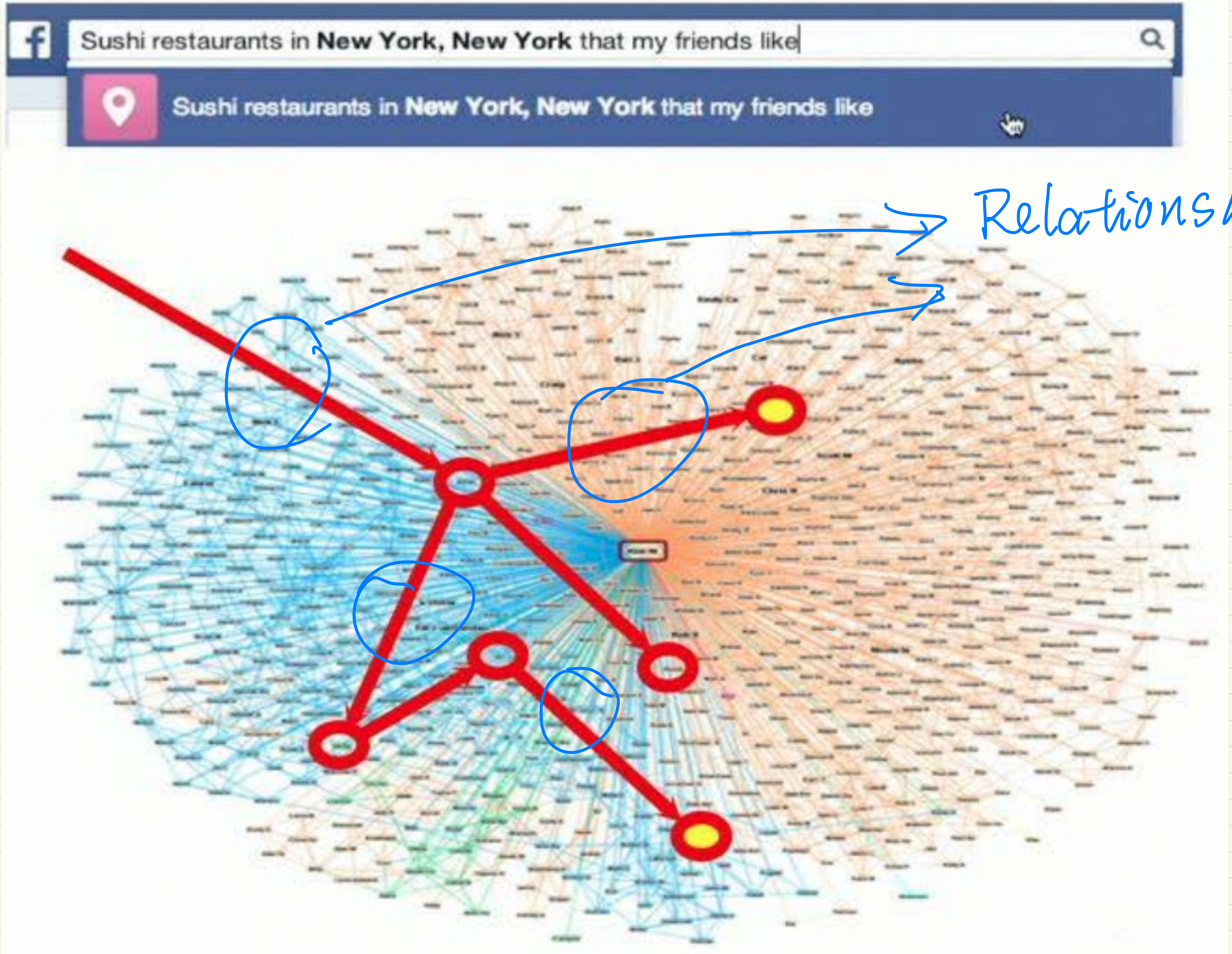
Dynamo (logical clocks)

HBase

MongoDB

Graph DB Use Cases

* "Graphy" workloads ⇒ Simplify modeling and quering



Sushi restaurants in **New York, New York** that my friends like

Sushi restaurants in **New York, New York** that my friends like

Relationships ⇒ Joins

(RDBMS)

↓

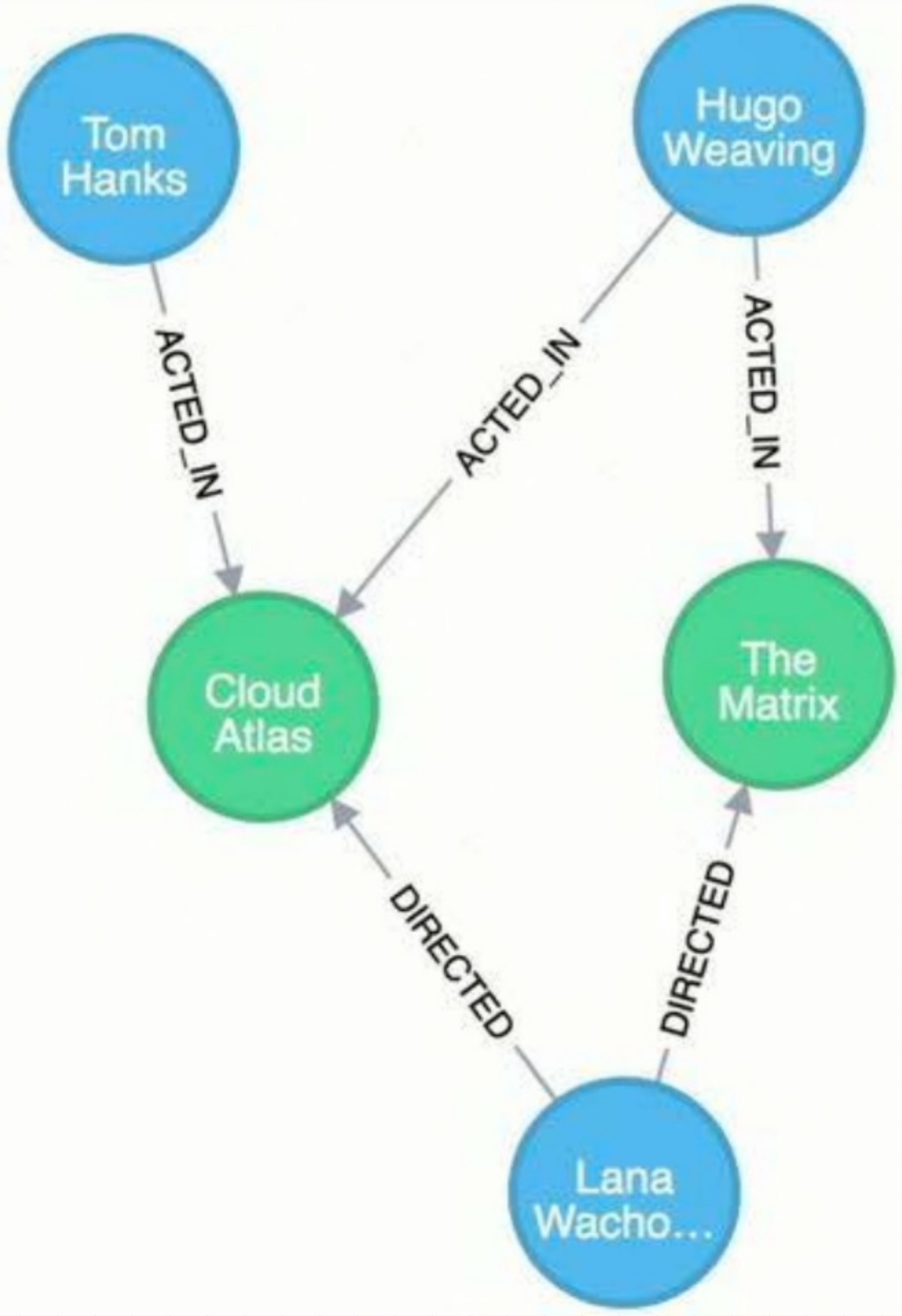
At runtime

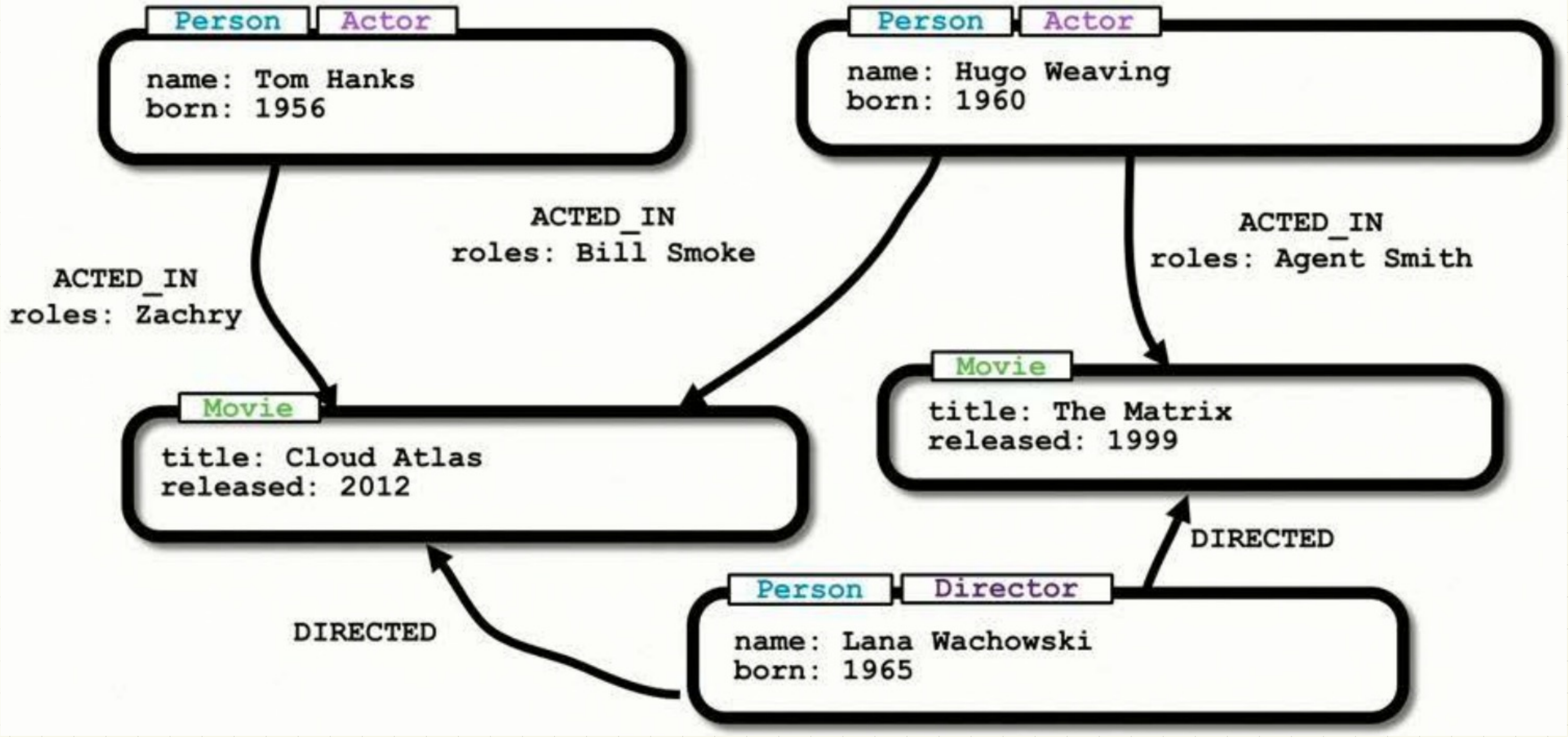
↓

Expensive

source: M. D. Allen

Graph Data Model : " Whiteboard-friendly "



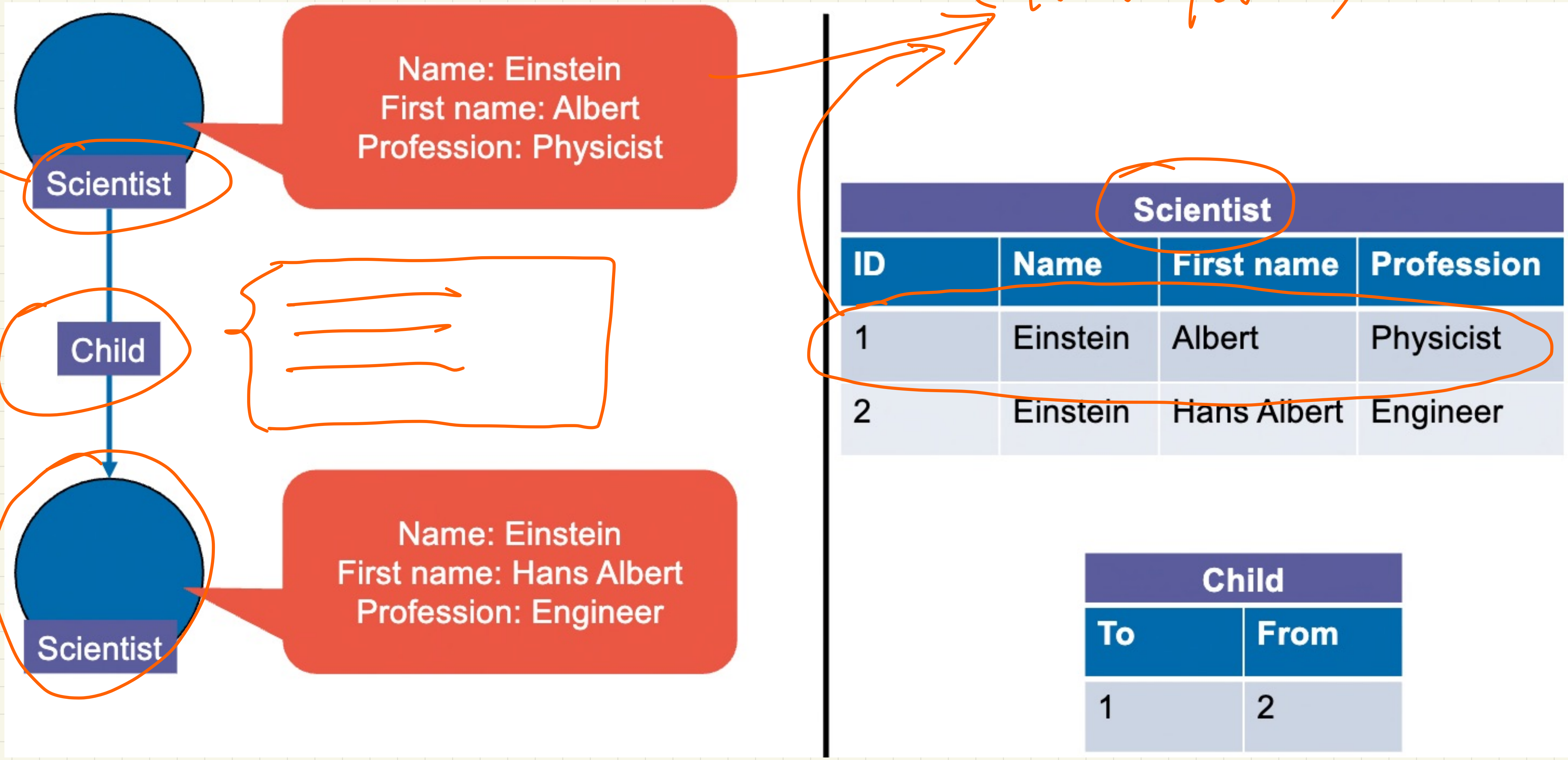


source: M. D. Allen

Mapping Terminology

Lable ⇔ Table : Grouping nodes ⇔ Tuples (K-V pairs)

Edge (relationship) ⇕ Joins



Name: Einstein
First name: Albert
Profession: Physicist

Name: Einstein
First name: Hans Albert
Profession: Engineer

Scientist			
ID	Name	First name	Profession
1	Einstein	Albert	Physicist
2	Einstein	Hans Albert	Engineer

Child	
To	From
1	2

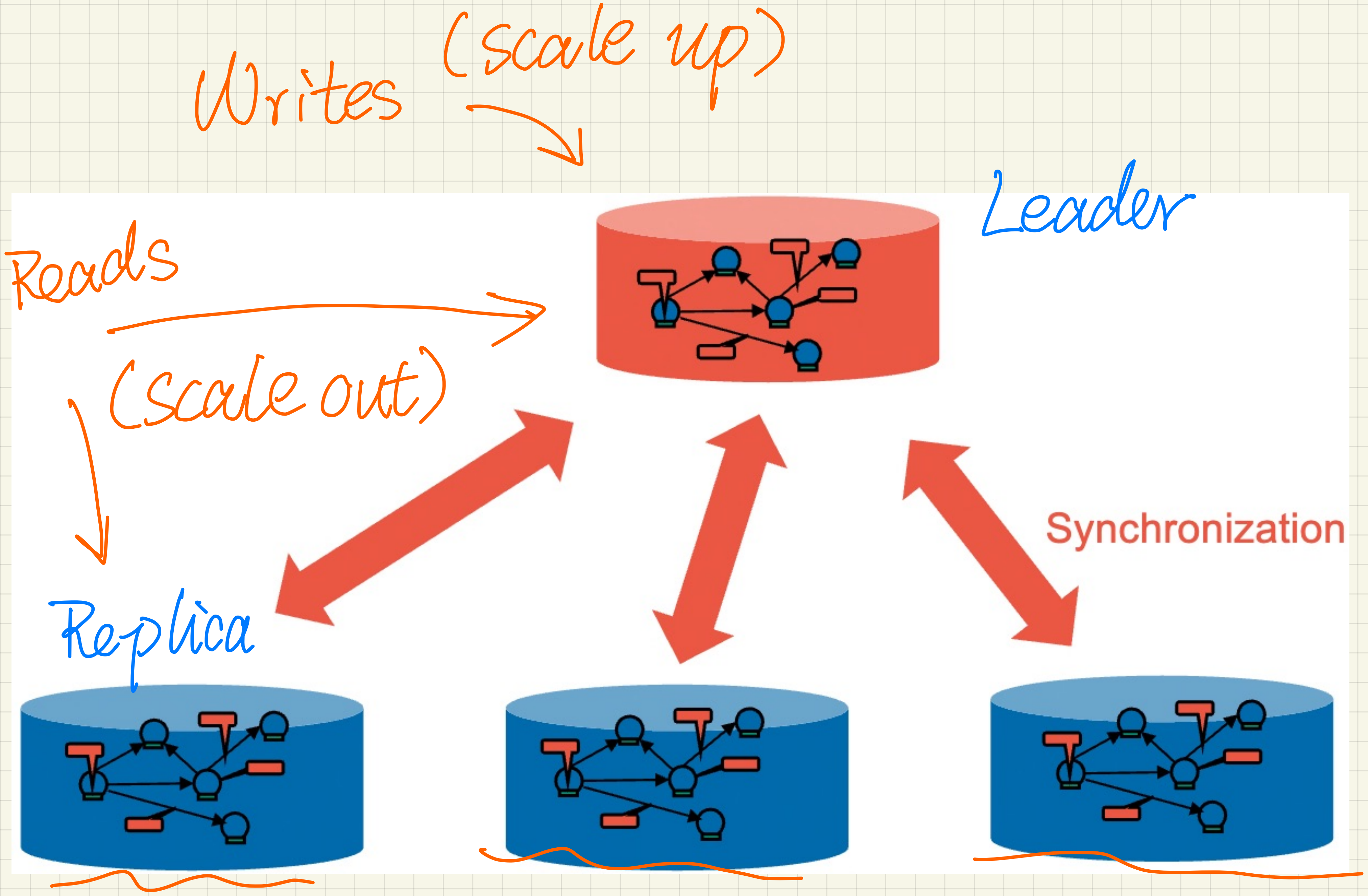
Neo4j System Design

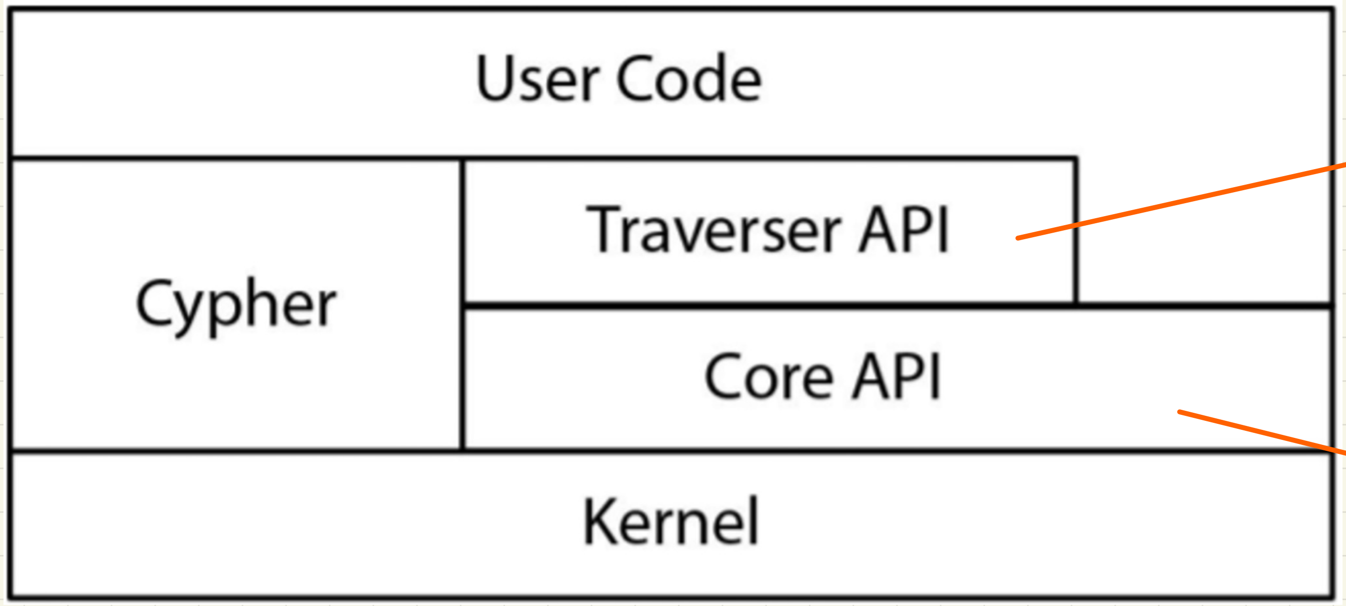
Single-Leader replication

(≃ MongoDB)

↳ Eventually consistent

⇒ Strong consistency (ACID)*





User Code

Cypher

Traverser API

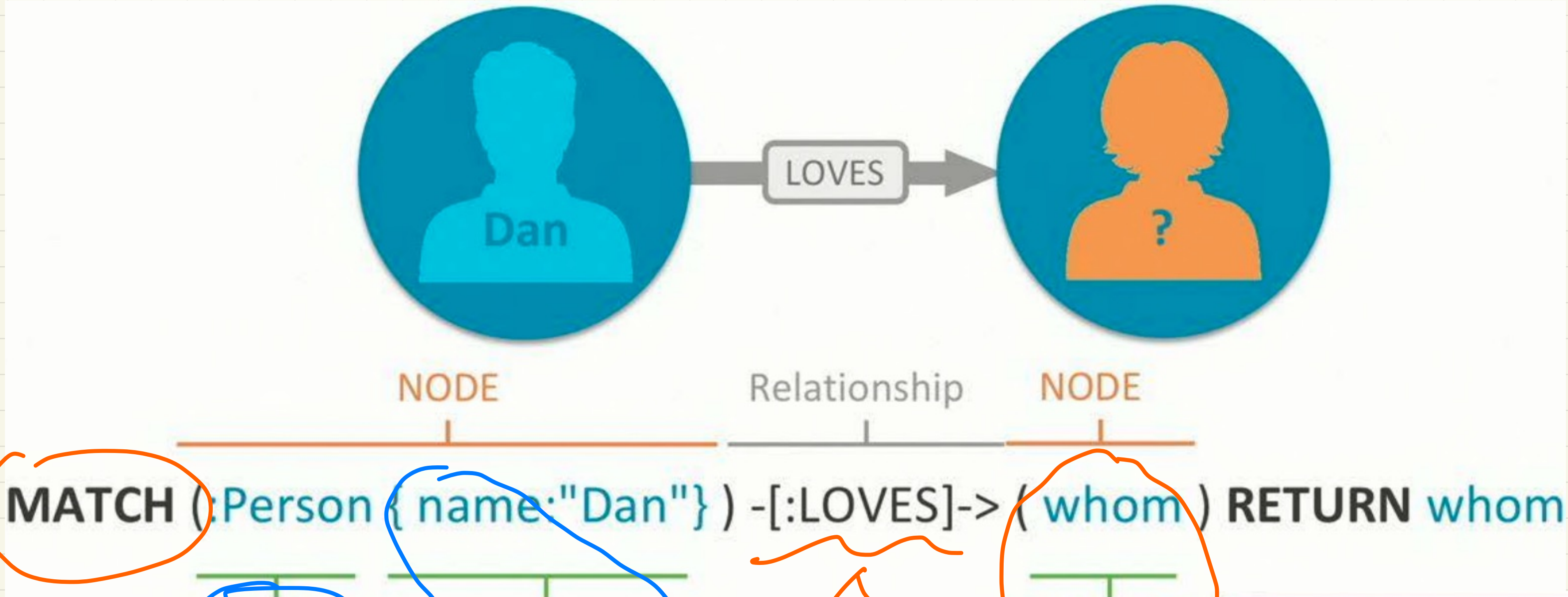
Core API

Kernel

→ High-level, declarative (DFS/BFS, pattern matching ...)

→ Low-leve, imperative (CRUD ops, fine-tune graphs ...)

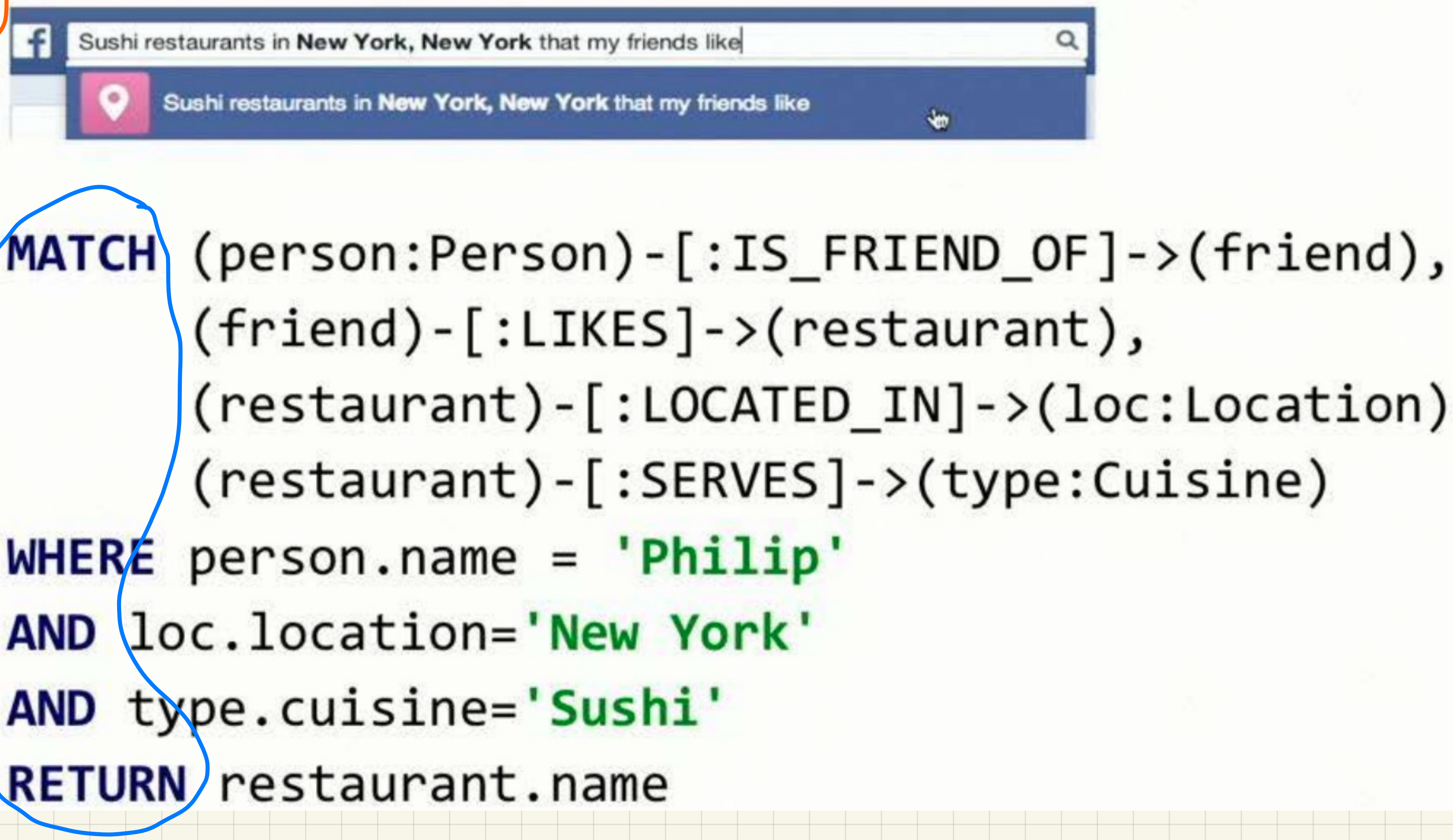
Querying in Neo4j : Cypher



```
MATCH (:Person {name:"Dan"}) -[:LOVES]-> (whom) RETURN whom
```

filter

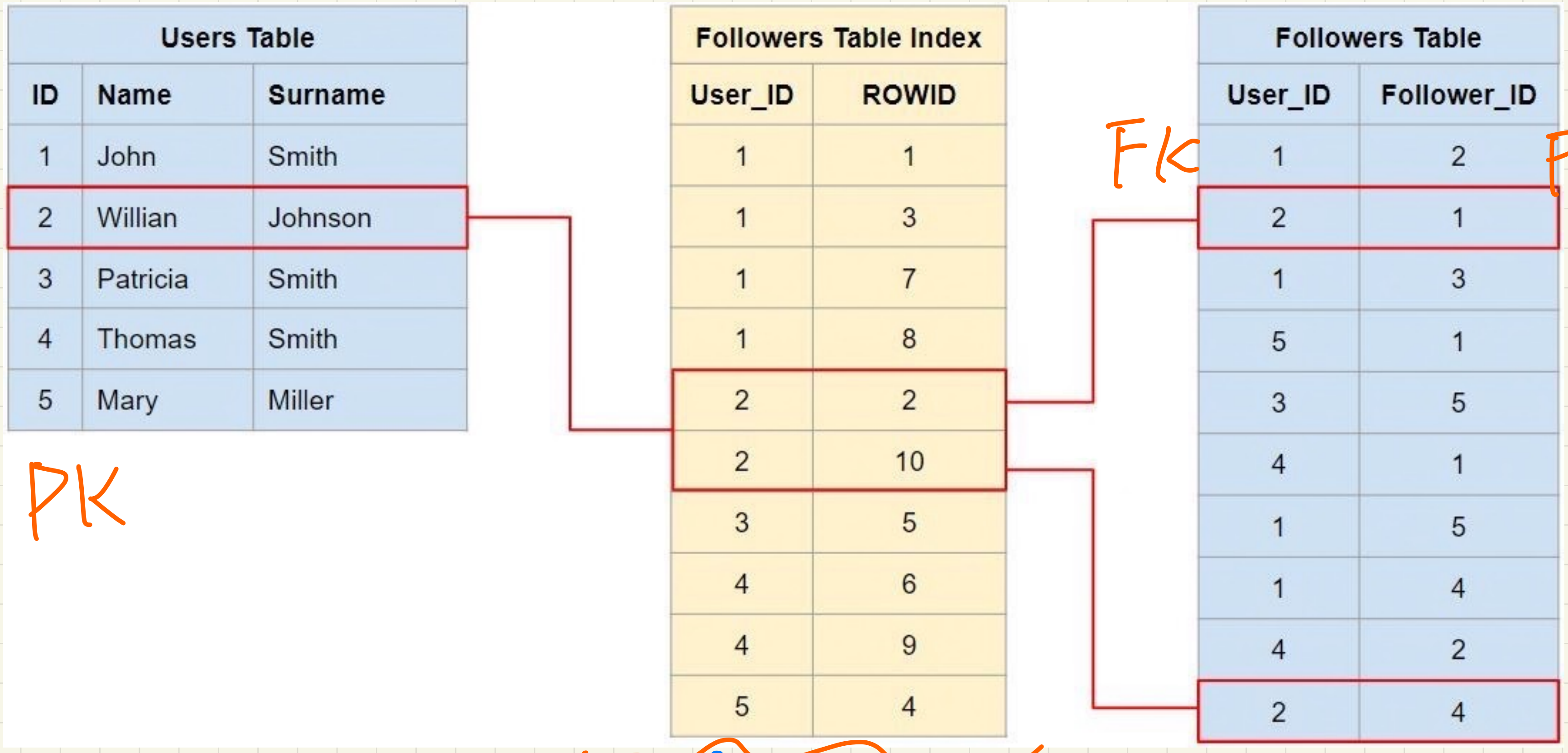
Case sensitive


Sushi restaurants in New York, New York that my friends like
Sushi restaurants in New York, New York that my friends like

```
MATCH (person:Person)-[:IS_FRIEND_OF]->(friend),
      (friend)-[:LIKES]->(restaurant),
      (restaurant)-[:LOCATED_IN]->(loc:Location)
      (restaurant)-[:SERVES]->(type:Cuisine)
WHERE person.name = 'Philip'
AND loc.location='New York'
AND type.cuisine='Sushi'
RETURN restaurant.name
```

Case insensitive

Indexing in Neo4j



Users Table

ID	Name	Surname
1	John	Smith
2	Willian	Johnson
3	Patricia	Smith
4	Thomas	Smith
5	Mary	Miller

Followers Table Index

User_ID	ROWID
1	1
1	3
1	7
1	8
2	2
2	10
3	5
4	6
4	9
5	4

Followers Table

User_ID	Follower_ID
1	2
2	1
1	3
5	1
3	5
4	1
1	5
1	4
4	2
2	4

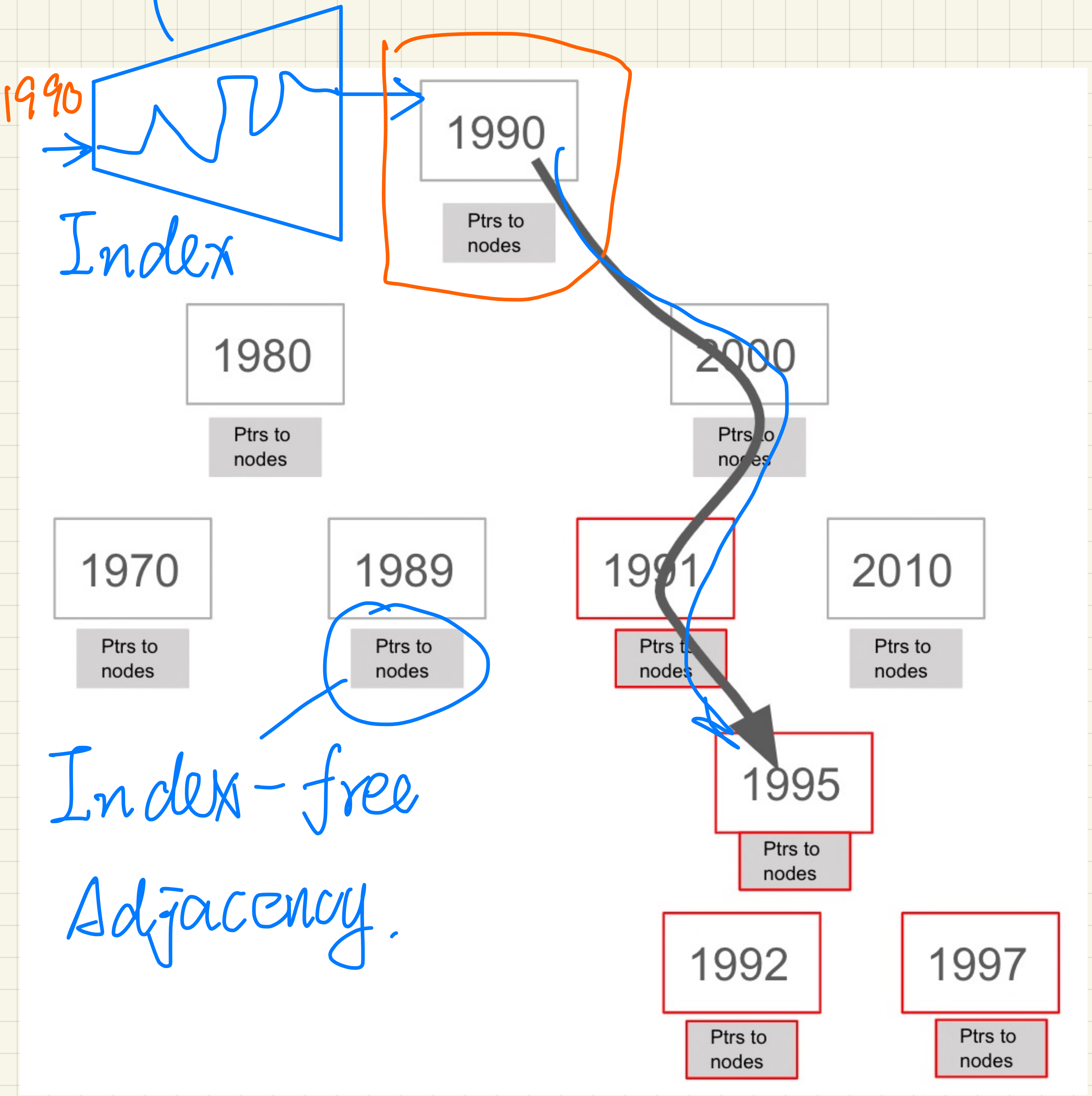
FK FK

PK

RDBMS Index for Joins

Neo4j index

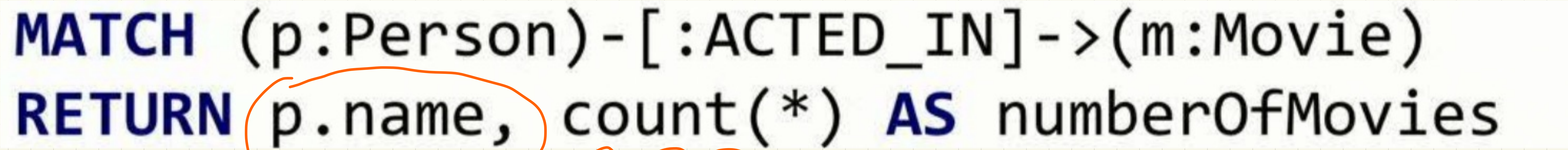
find the starting point



Index-free Adjacency.

Implicit Grouping

⇒ Group by any non-aggregate fields in RETURN

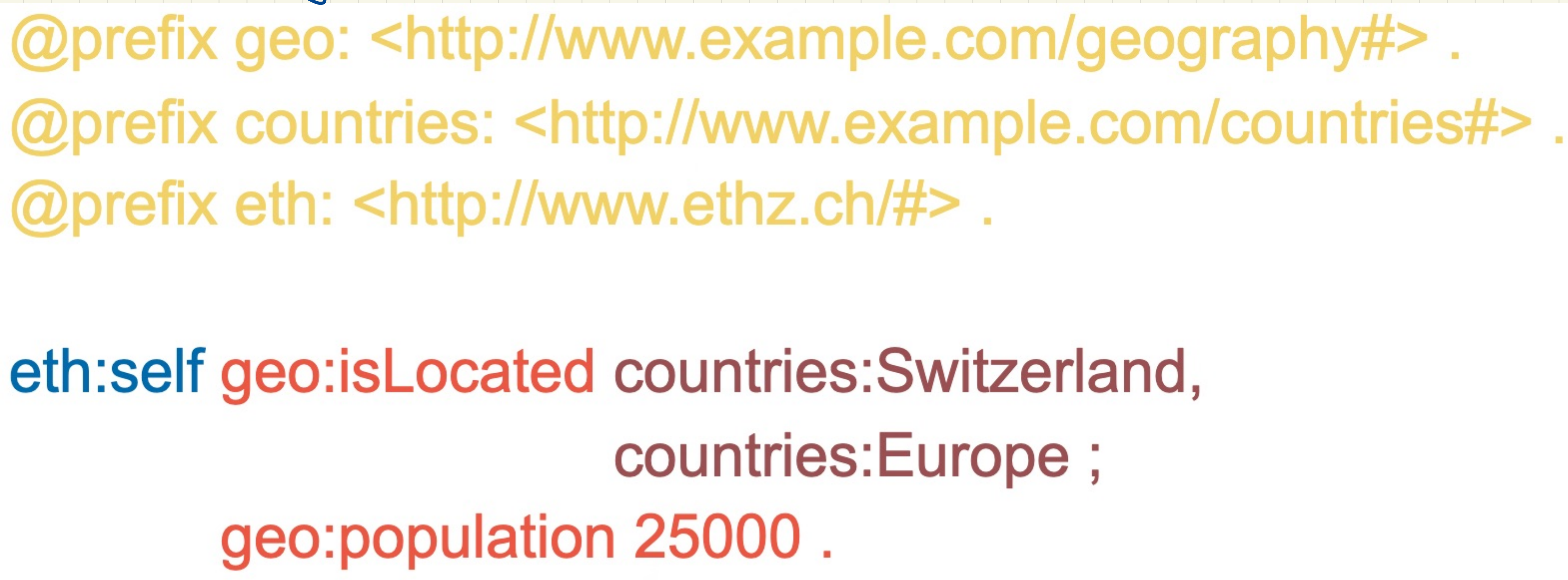


```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
RETURN p.name, count(*) AS numberOfMovies
```

agg.

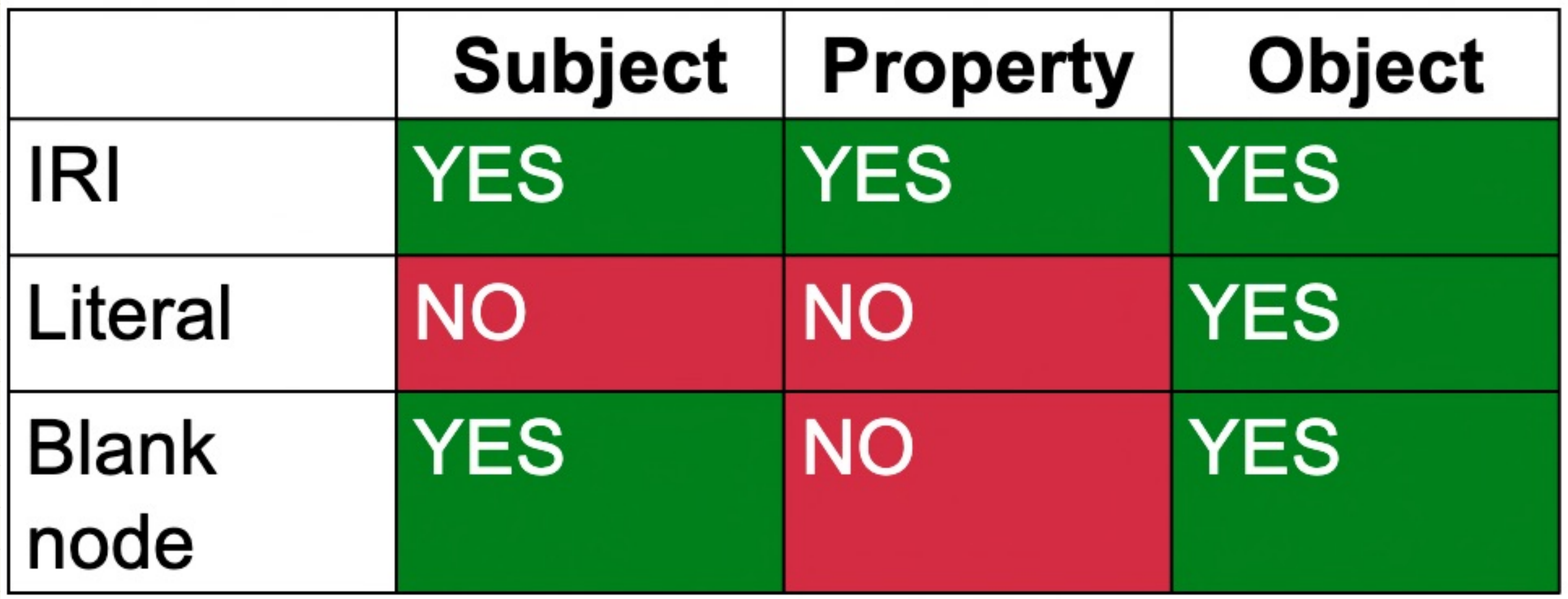
RDF

① Turtle syntax



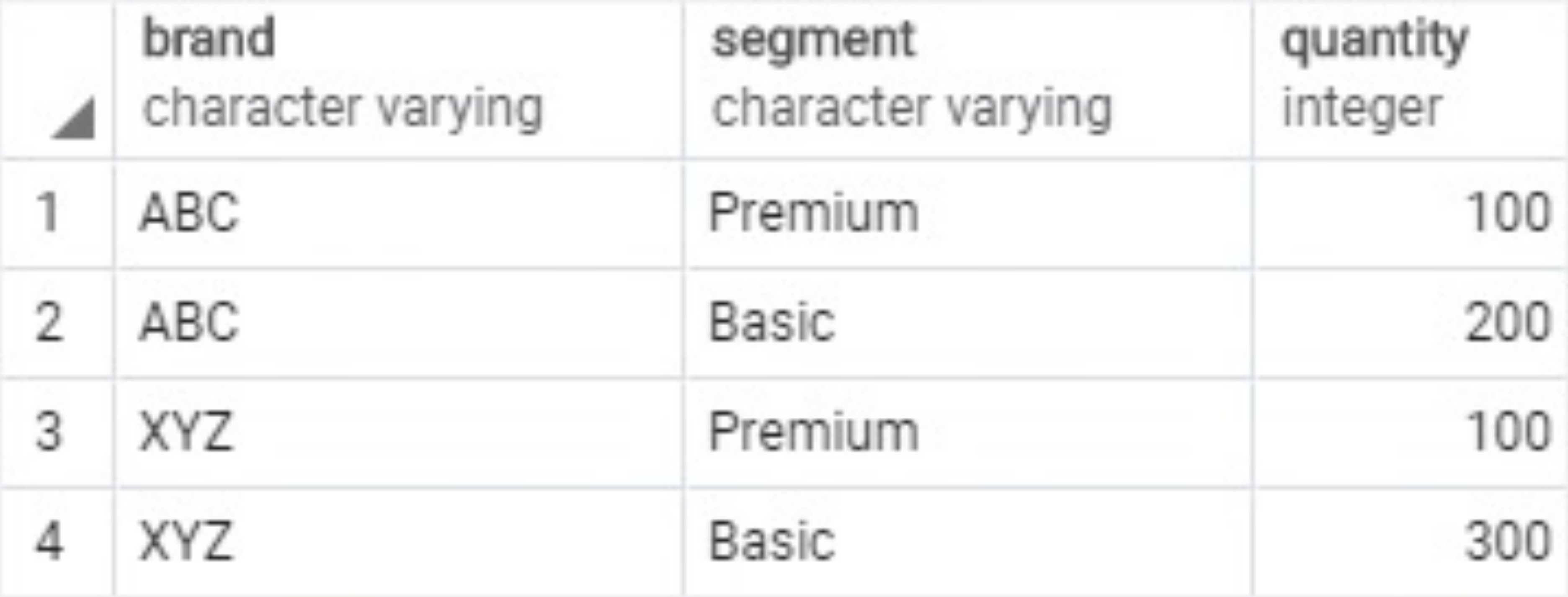
```
@prefix geo: <http://www.example.com/geography#> .
@prefix countries: <http://www.example.com/countries#> .
@prefix eth: <http://www.ethz.ch/#> .

eth:self geo:isLocated countries:Switzerland,
                       countries:Europe ;
    geo:population 25000 .
```

②

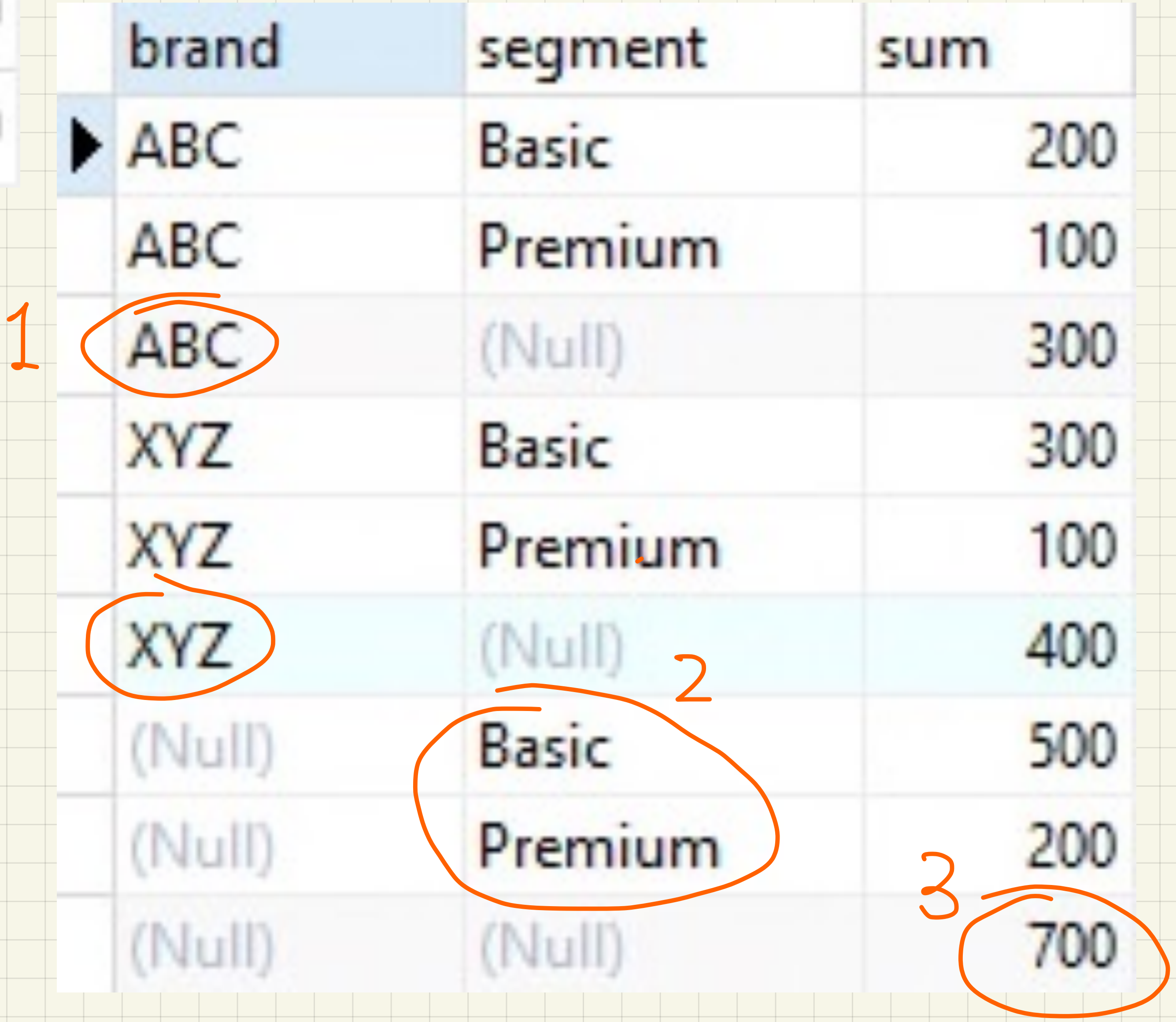


	Subject	Property	Object
IRI	YES	YES	YES
Literal	NO	NO	YES
Blank node	YES	NO	YES

Aggregation Leveling

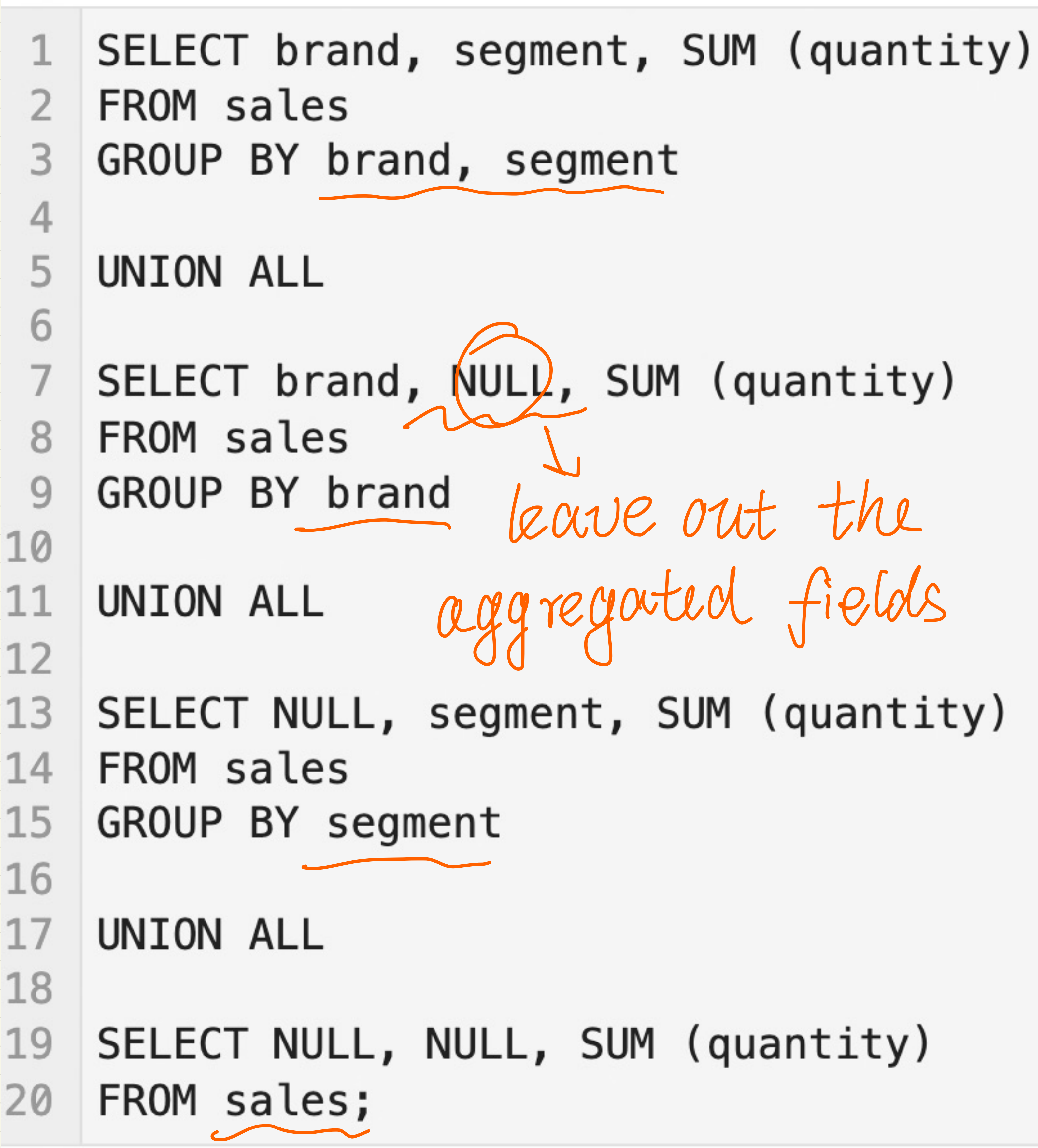


	brand character varying	segment character varying	quantity integer
1	ABC	Premium	100
2	ABC	Basic	200
3	XYZ	Premium	100
4	XYZ	Basic	300



	brand	segment	sum
▶	ABC	Basic	200
	ABC	Premium	100
1	ABC	(Null)	300
	XYZ	Basic	300
	XYZ	Premium	100
	XYZ	(Null)	400
2	(Null)	Basic	500
	(Null)	Premium	200
3	(Null)	(Null)	700

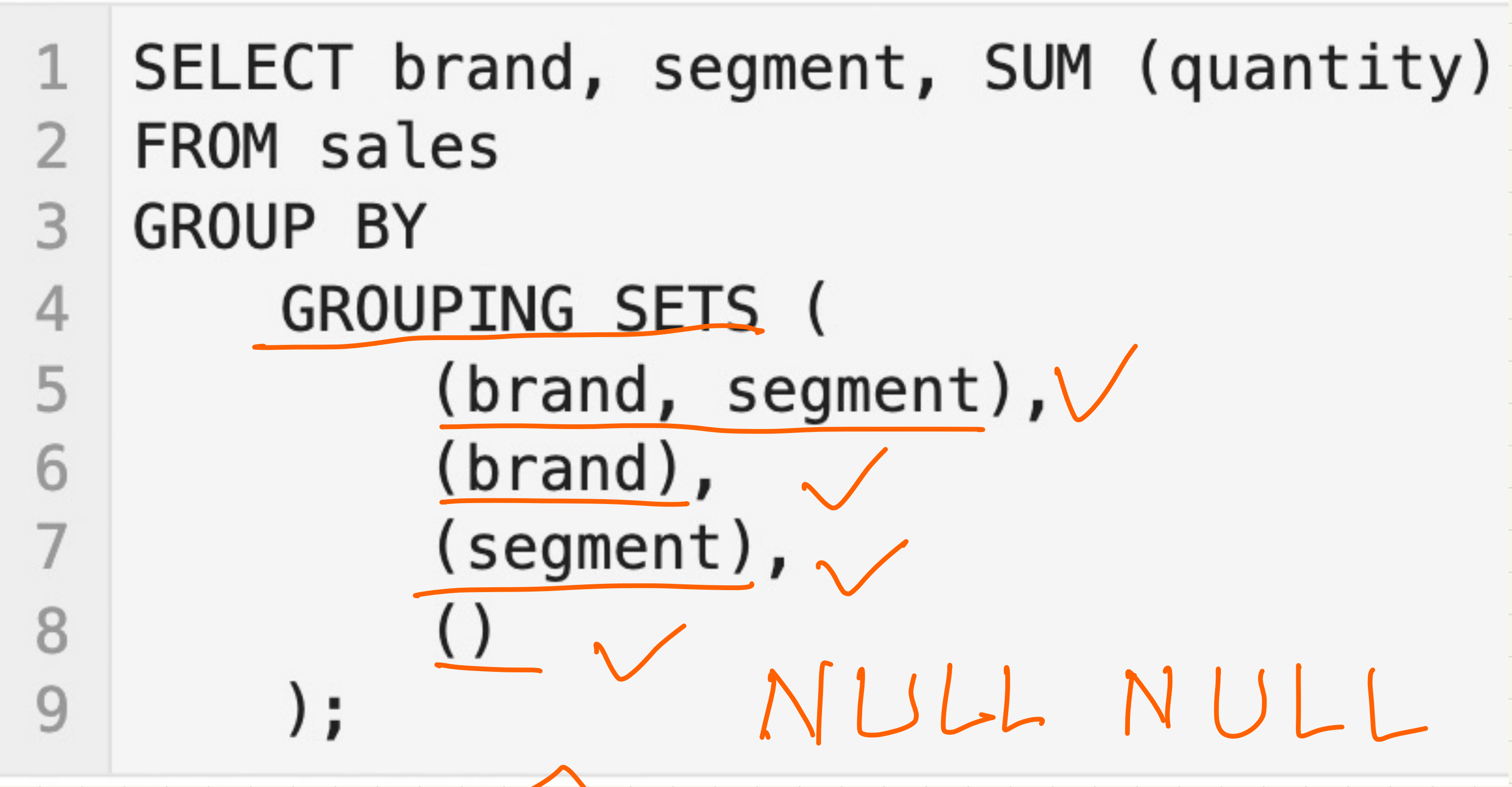
① Manual grouping



```
SELECT brand, segment, SUM (quantity)
FROM sales
GROUP BY brand, segment

UNION ALL

SELECT brand, NULL, SUM (quantity)
FROM sales
GROUP BY brand

UNION ALL

SELECT NULL, segment, SUM (quantity)
FROM sales
GROUP BY segment

UNION ALL

SELECT NULL, NULL, SUM (quantity)
FROM sales;
```

leave out the aggregated fields

② Grouping Sets

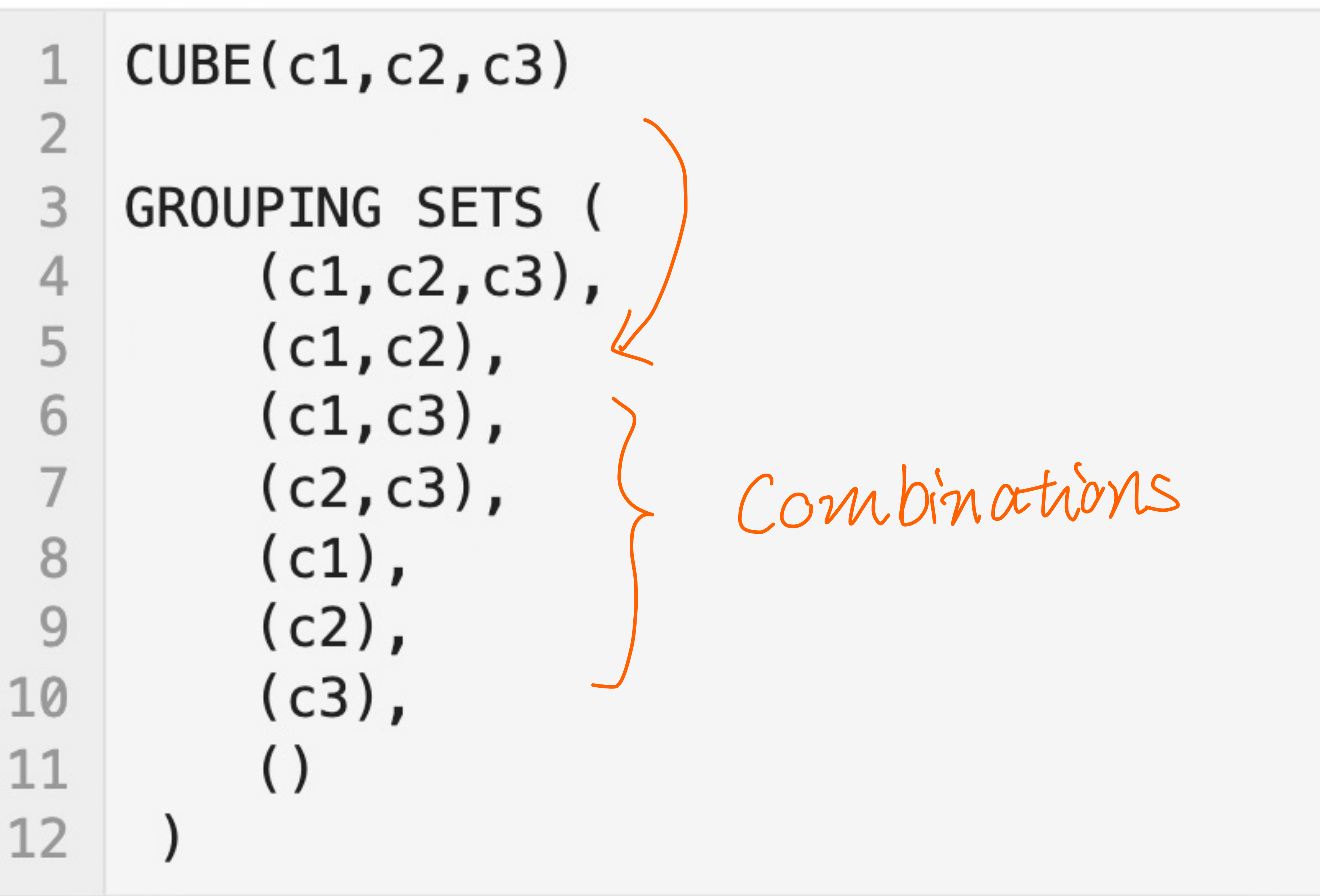


```
SELECT brand, segment, SUM (quantity)
FROM sales
GROUP BY
    GROUPING SETS (
        (brand, segment), ✓
        (brand),  ✓
        (segment), ✓
        ()  ✓
    );
```

NULL NULL

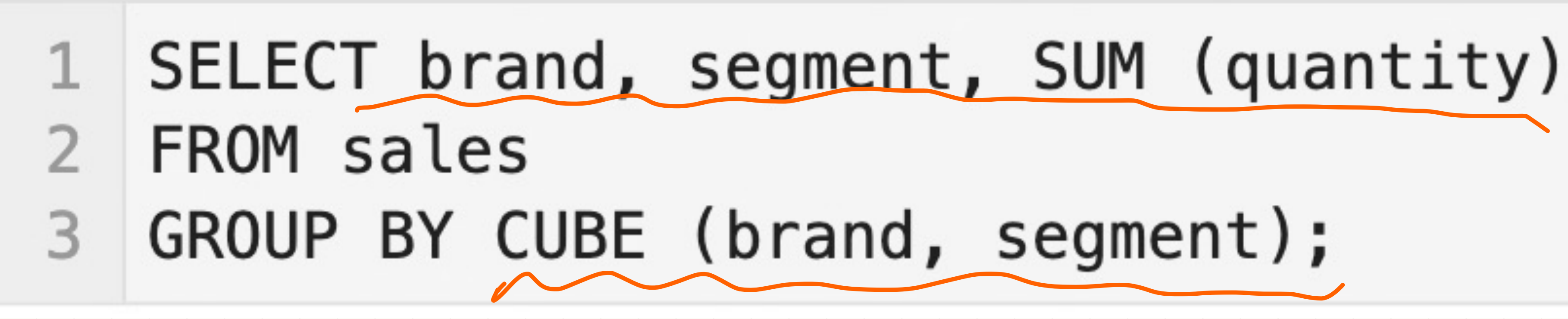
translate

③ Grouping with Cube

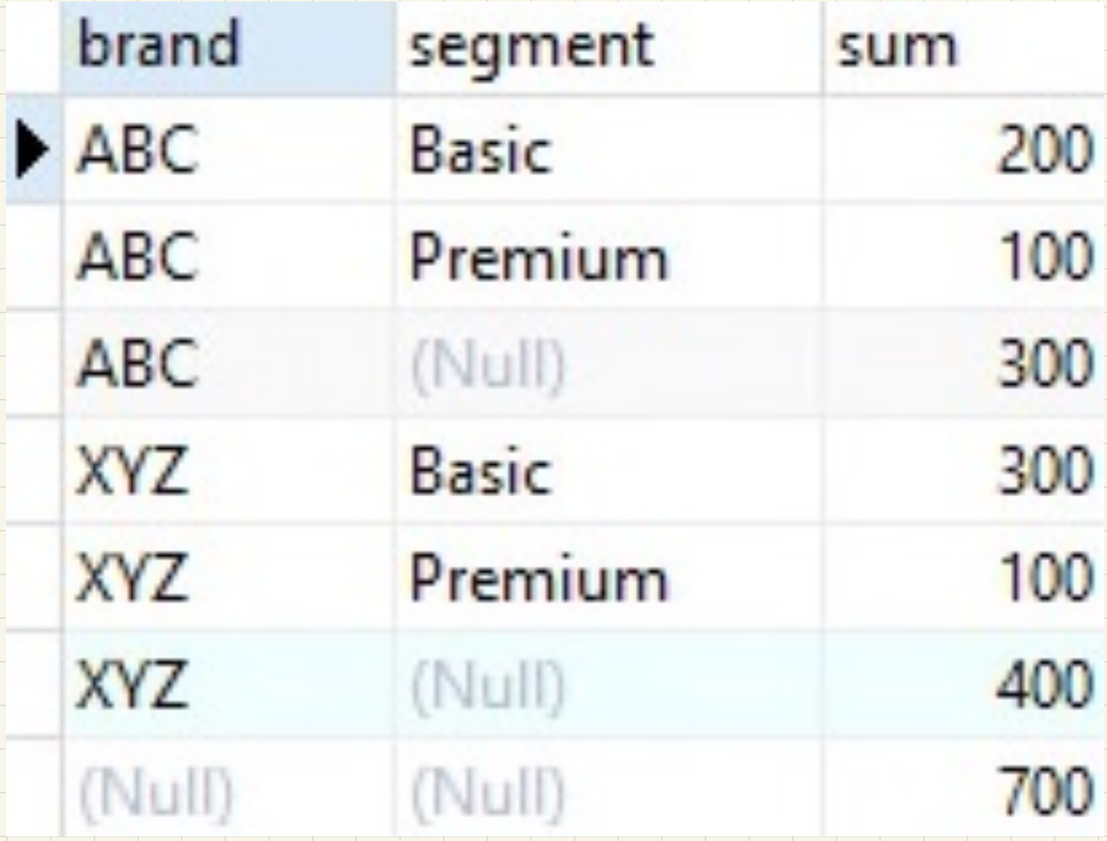


```
CUBE(c1,c2,c3)

GROUPING SETS (
    (c1,c2,c3),
    (c1,c2),
    (c1,c3),
    (c2,c3),
    (c1),
    (c2),
    (c3),
    ()
 )
```

Combinations

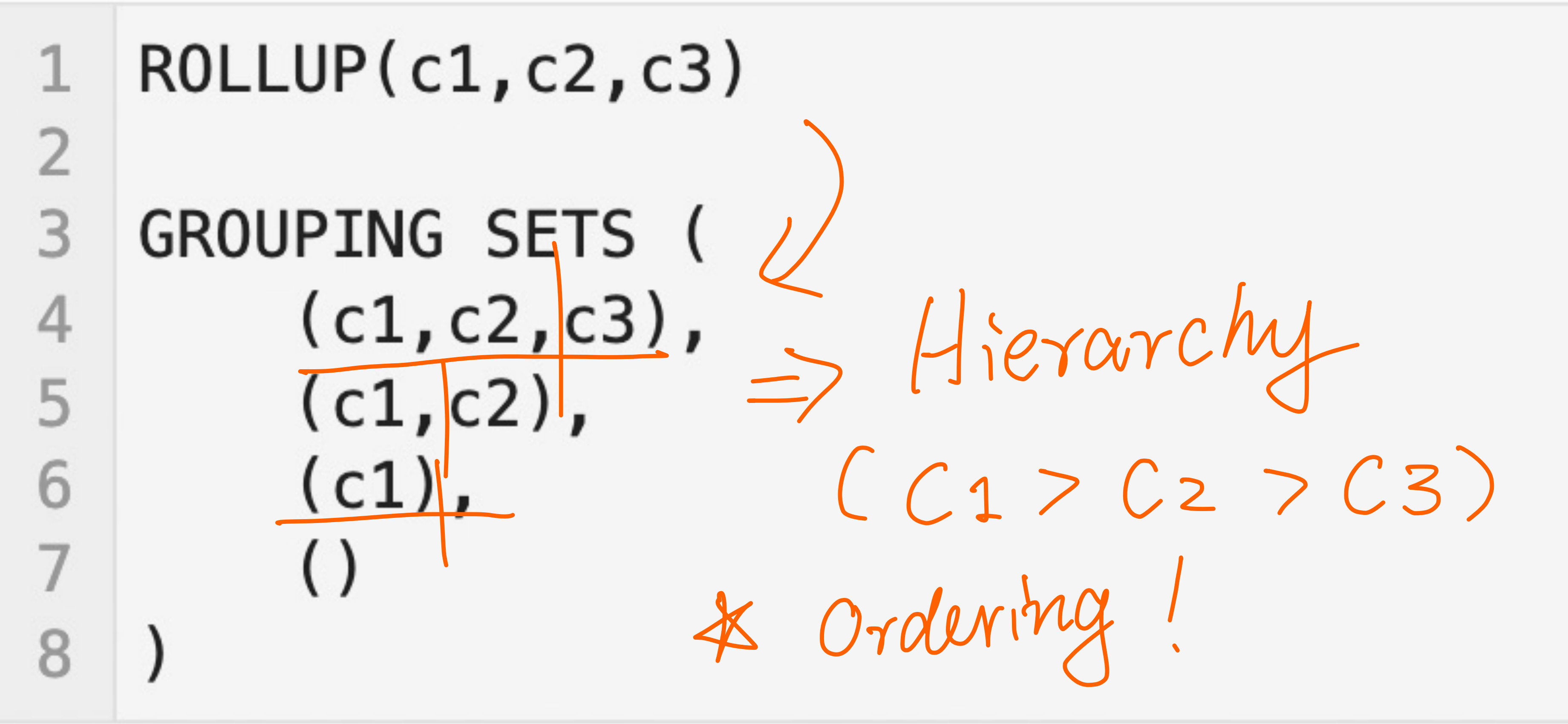


```
SELECT brand, segment, SUM (quantity)
FROM sales
GROUP BY CUBE (brand, segment);
```



brand	segment	sum
ABC	Basic	200
ABC	Premium	100
ABC	(Null)	300
XYZ	Basic	300
XYZ	Premium	100
XYZ	(Null)	400
(Null)	(Null)	700

④ Grouping with Rollup

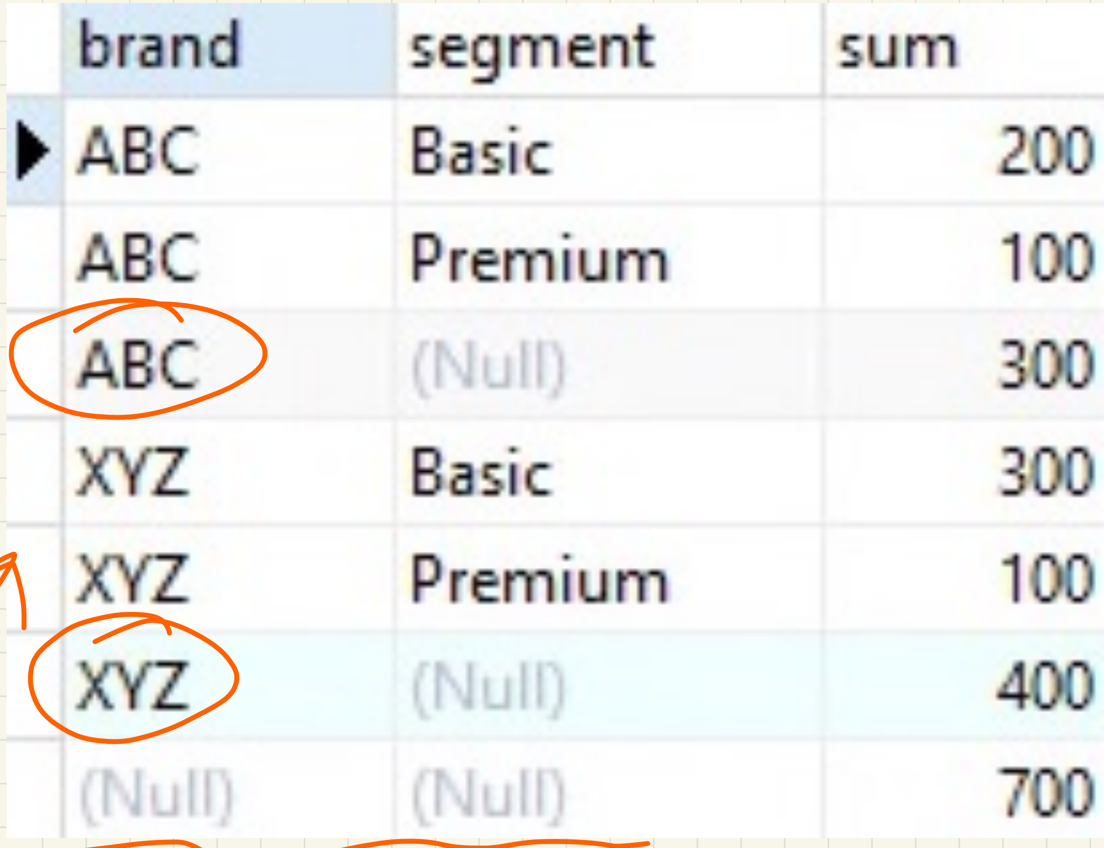


```
ROLLUP(c1,c2,c3)

GROUPING SETS (
      (c1,c2,c3),
      (c1,c2),
      (c1),
      ()
)
```

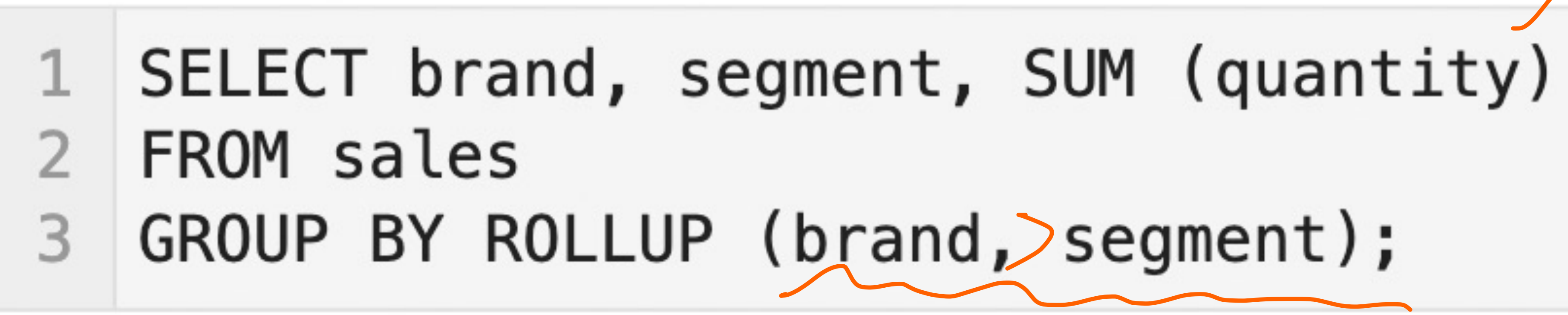
⇒ Hierarchy

(C1 > C2 > C3)

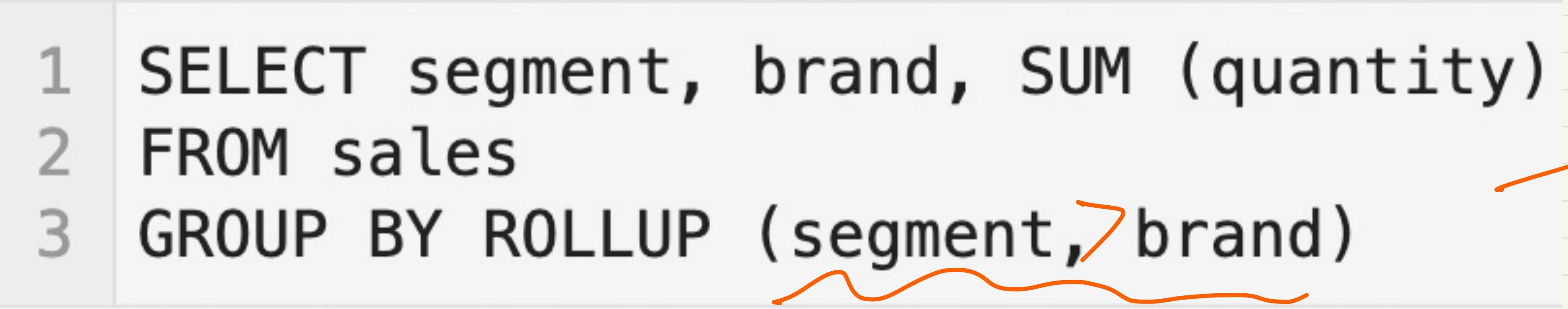
* Ordering !



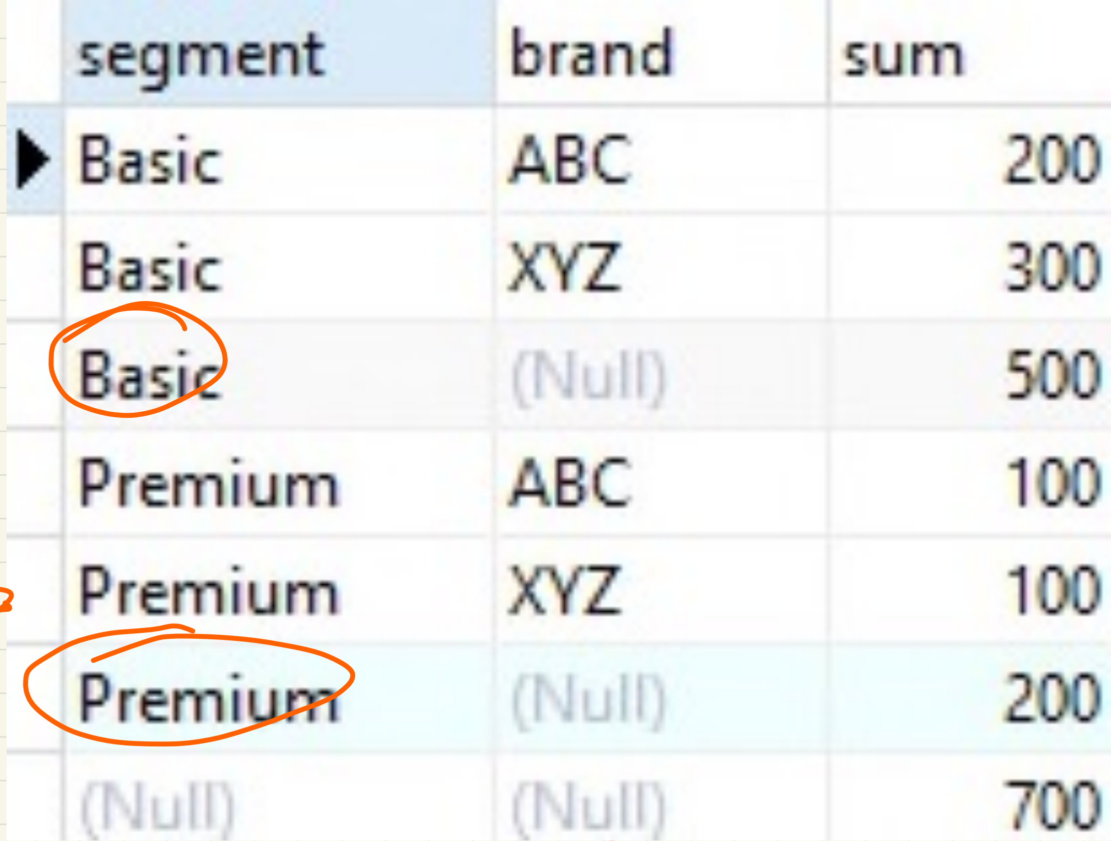
	brand	segment	sum
▶	ABC	Basic	200
	ABC	Premium	100
	ABC	(Null)	300
	XYZ	Basic	300
	XYZ	Premium	100
	XYZ	(Null)	400
	(Null)	(Null)	700



```
SELECT brand, segment, SUM (quantity)
FROM sales
GROUP BY ROLLUP (brand, segment);
```



```
SELECT segment, brand, SUM (quantity)
FROM sales
GROUP BY ROLLUP (segment, brand)
```



	segment	brand	sum
▶	Basic	ABC	200
	Basic	XYZ	300
	Basic	(Null)	500
	Premium	ABC	100
	Premium	XYZ	100
	Premium	(Null)	200
	(Null)	(Null)	700