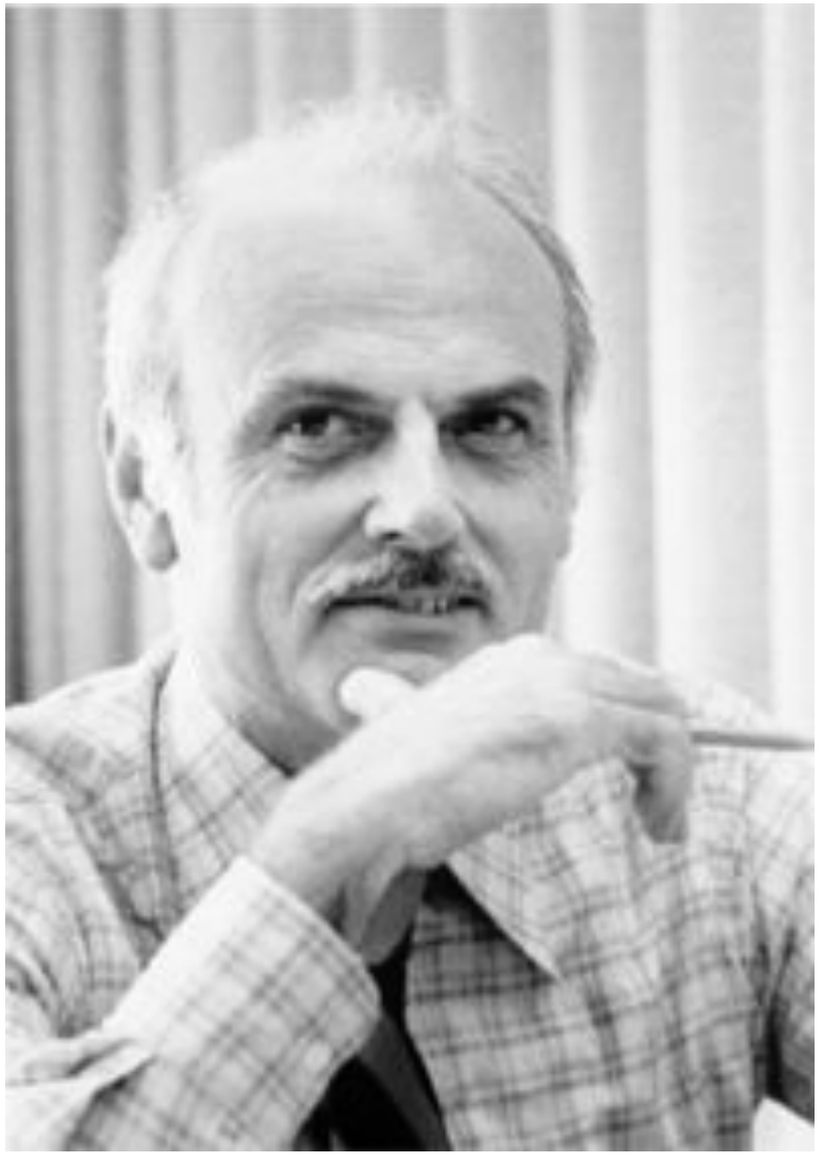
datalab

data is everywhere, value is hidden

Relational Databases

Lecturer: Азат Якупов (Azat Yakupov)
https://datalaboratory.one



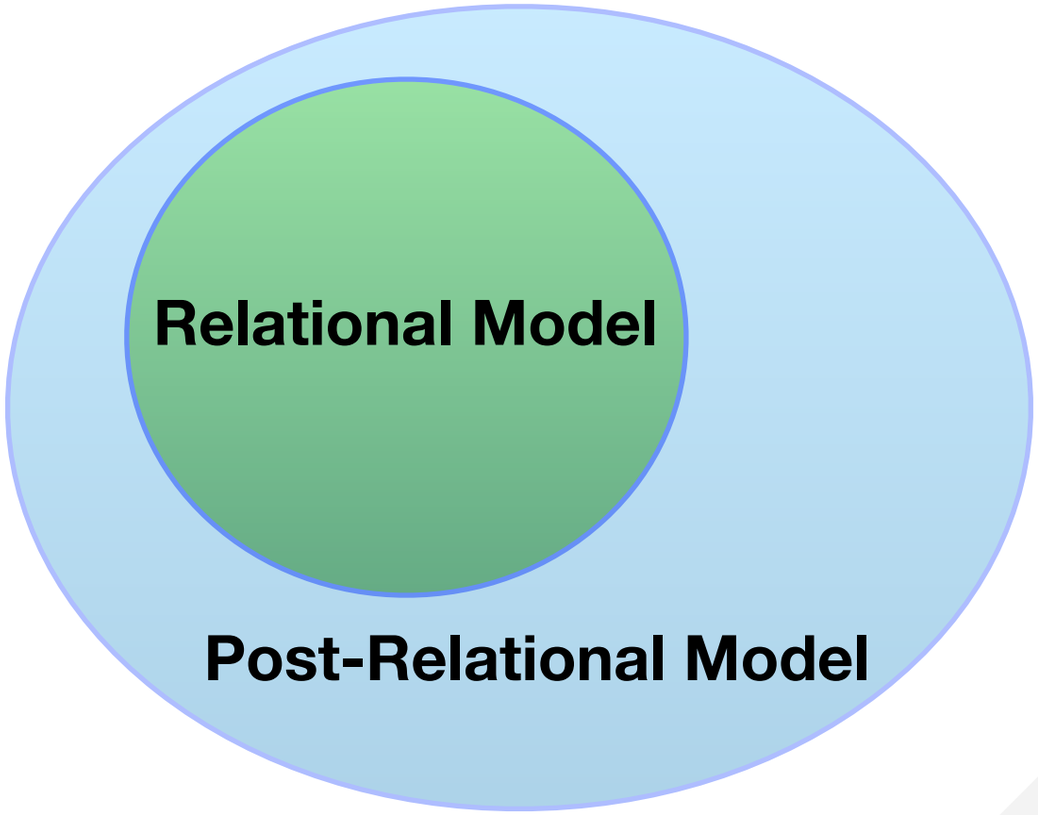
“All information in the database must be cast explicitly in terms of values in relations and in no other way”

Information Principle

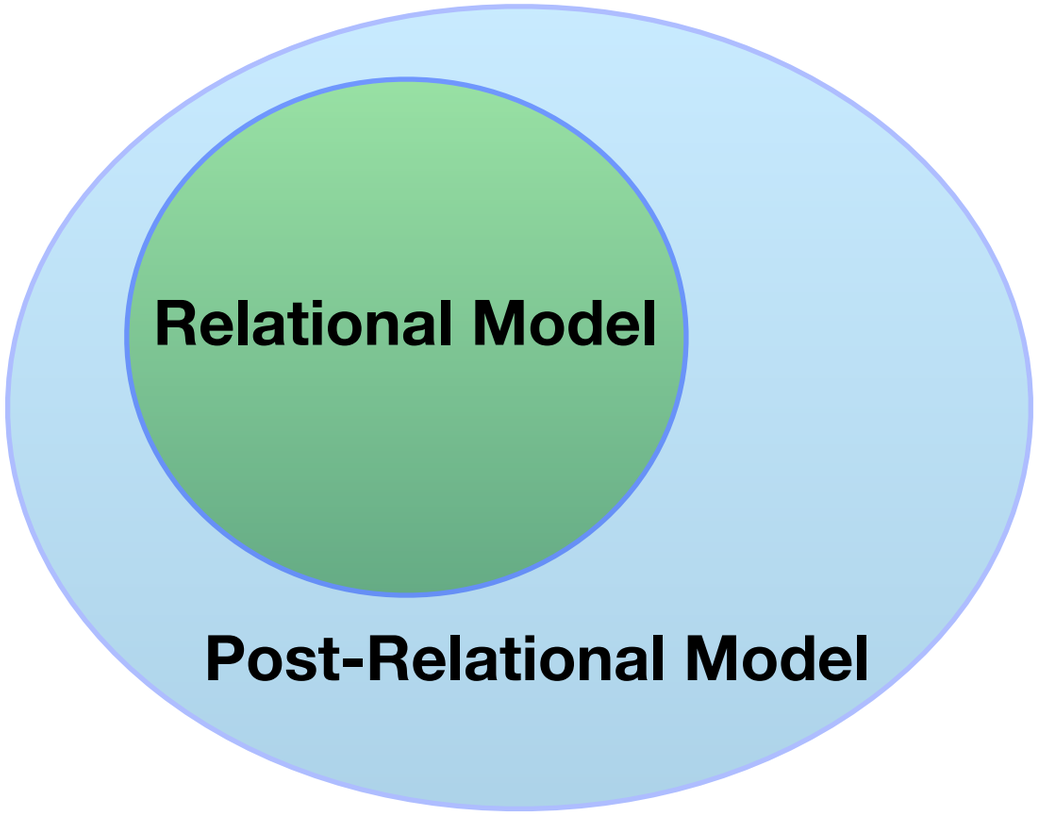
- Data Definition Language (~ DDL)
- Data Integrity (~ database constraints)
- Data Manipulation Language (~ DML)

ID	Person
1	Ivan Ivanov

ID	Hobby	PersonId
1	music	1
2	blog	1

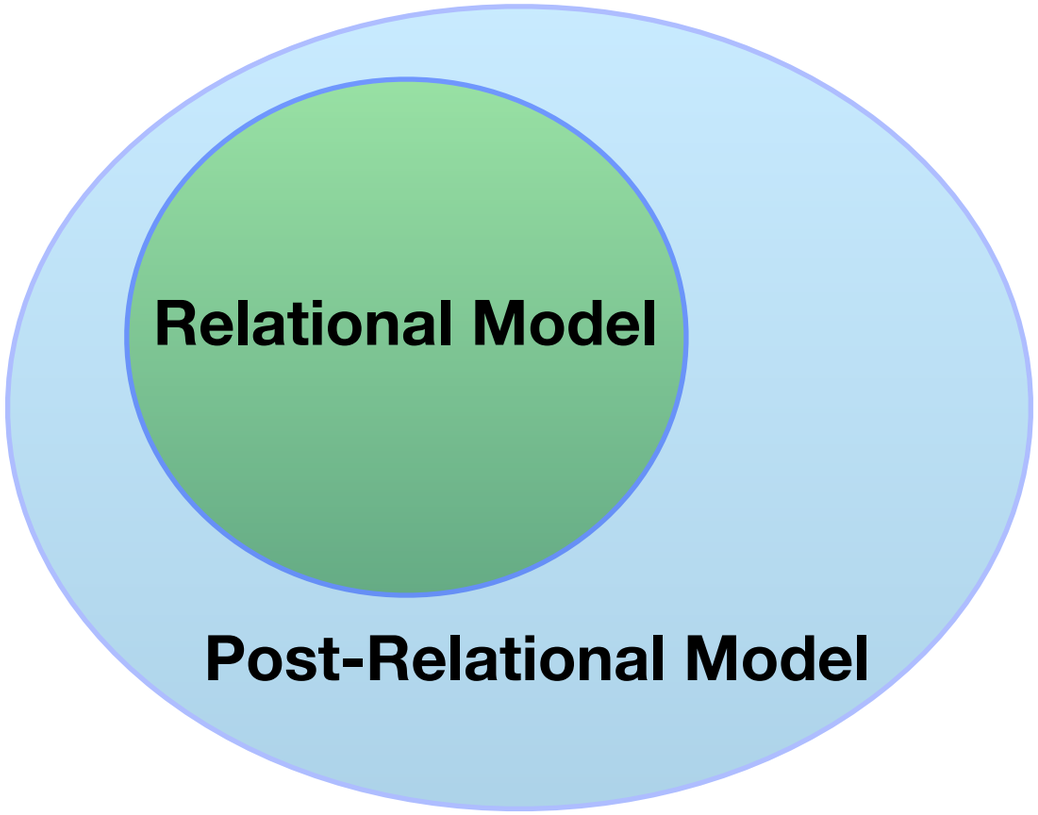


ID	Person	Hobbies
1	Ivan Ivanov	<hobbies> <item value =“music” /> <item value =“blog” /> </hobbies>

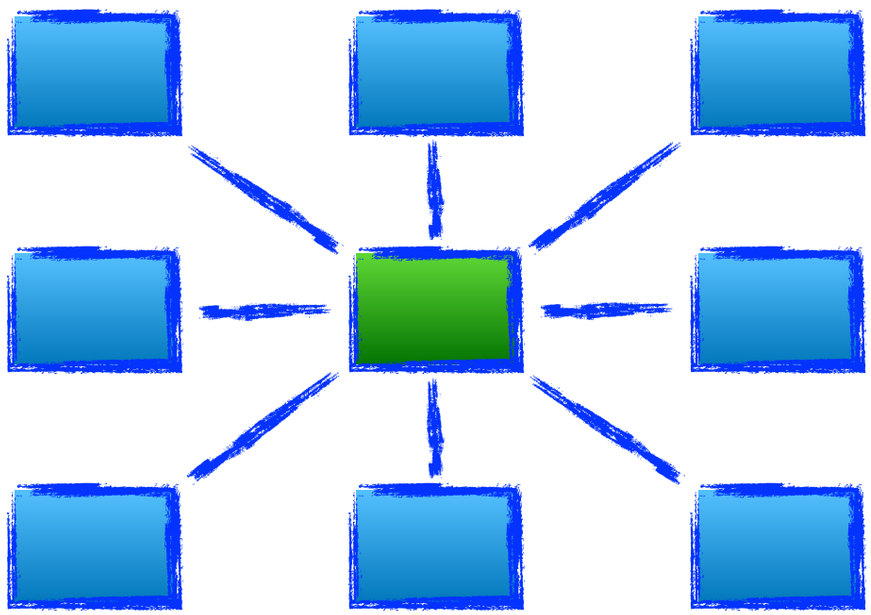


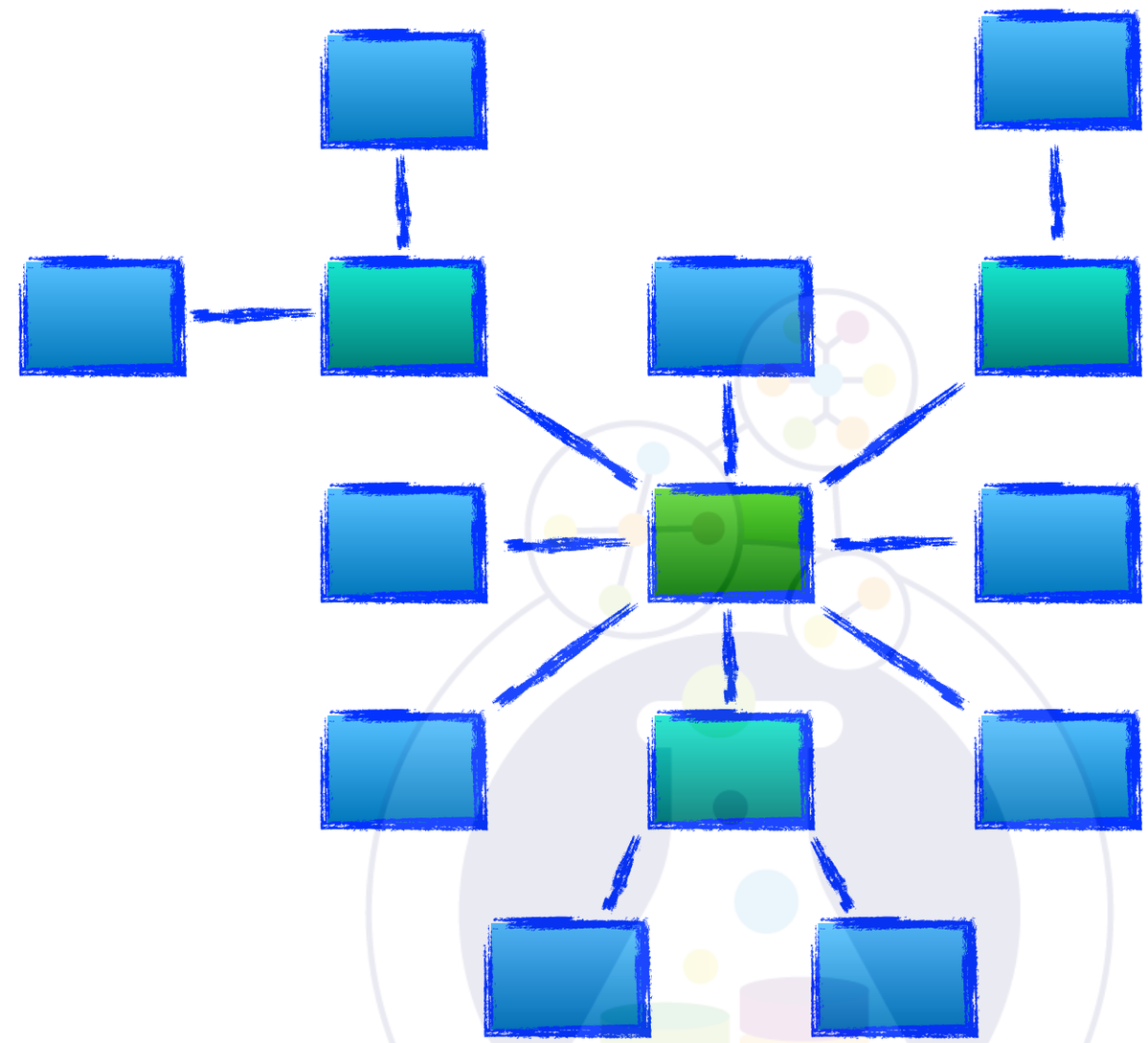
ID	Person	Hobbies
1	Ivan Ivanov	

ID	Hobby
1	music
2	blog



ID	Person	Hobbies
1	Ivan Ivanov	{ "hobbies": [{"value": "music"}, {"value": "blog"}] }





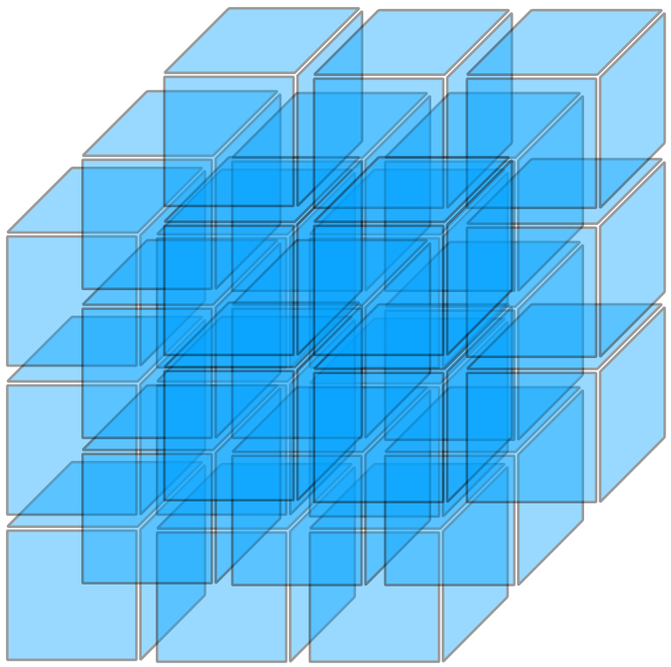
Dimensional Table	
column1	*type1*
column2	*type2*

Fact Table	
cell1	*type1*
cell2	*type2*
cell3	*type1*
cell4	*type3*

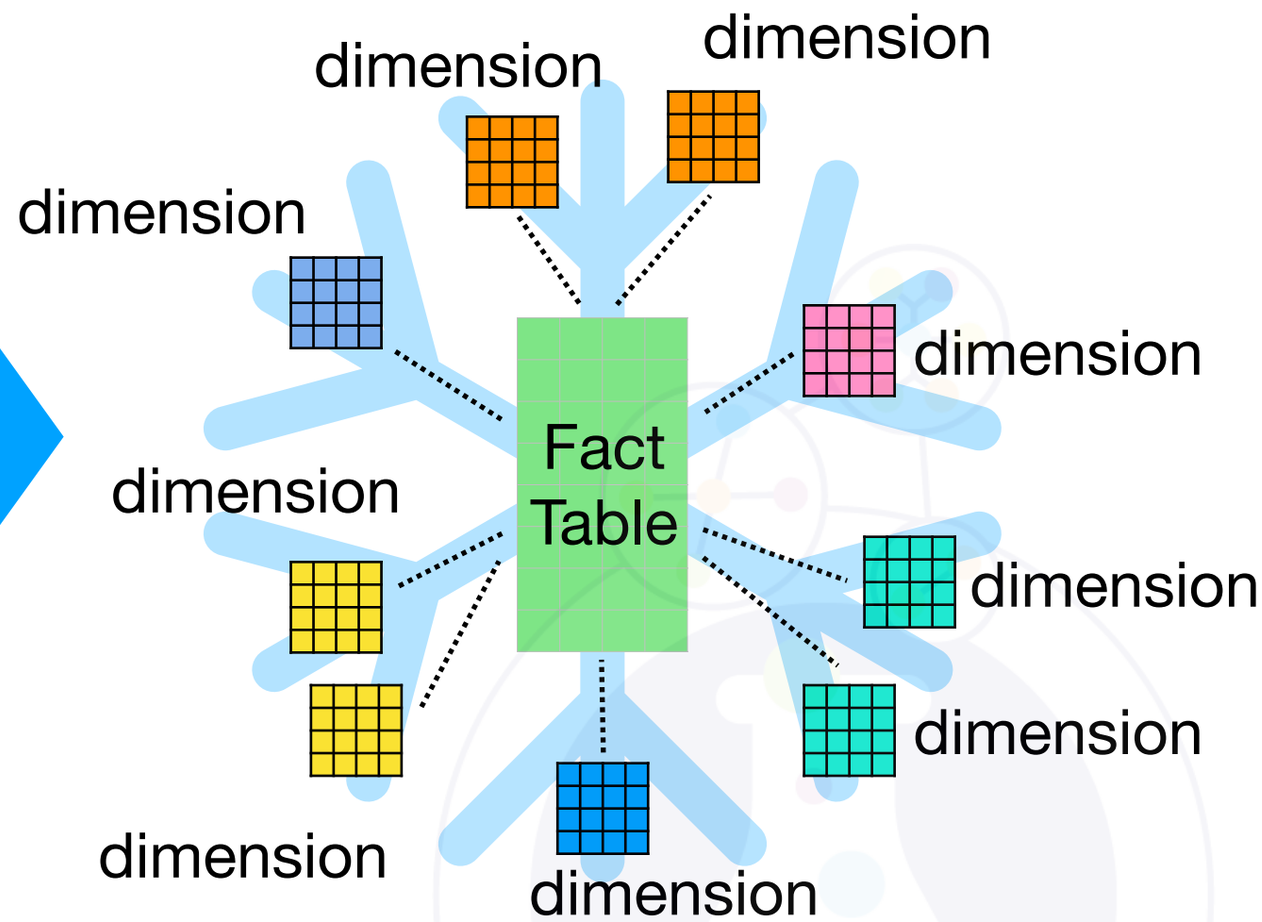
Dimensional Table	
column1	*type1*
column2	*type2*

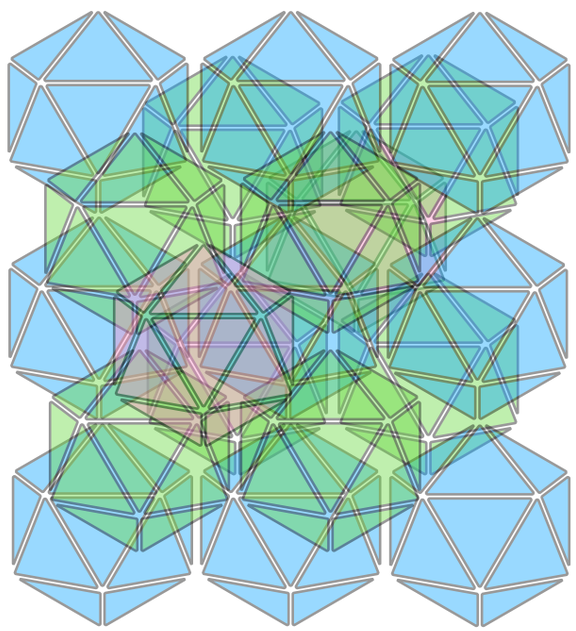
Dimensional Table	
column1	*type1*
column2	*type2*

Dimensional Table	
column1	*type1*
column2	*type2*

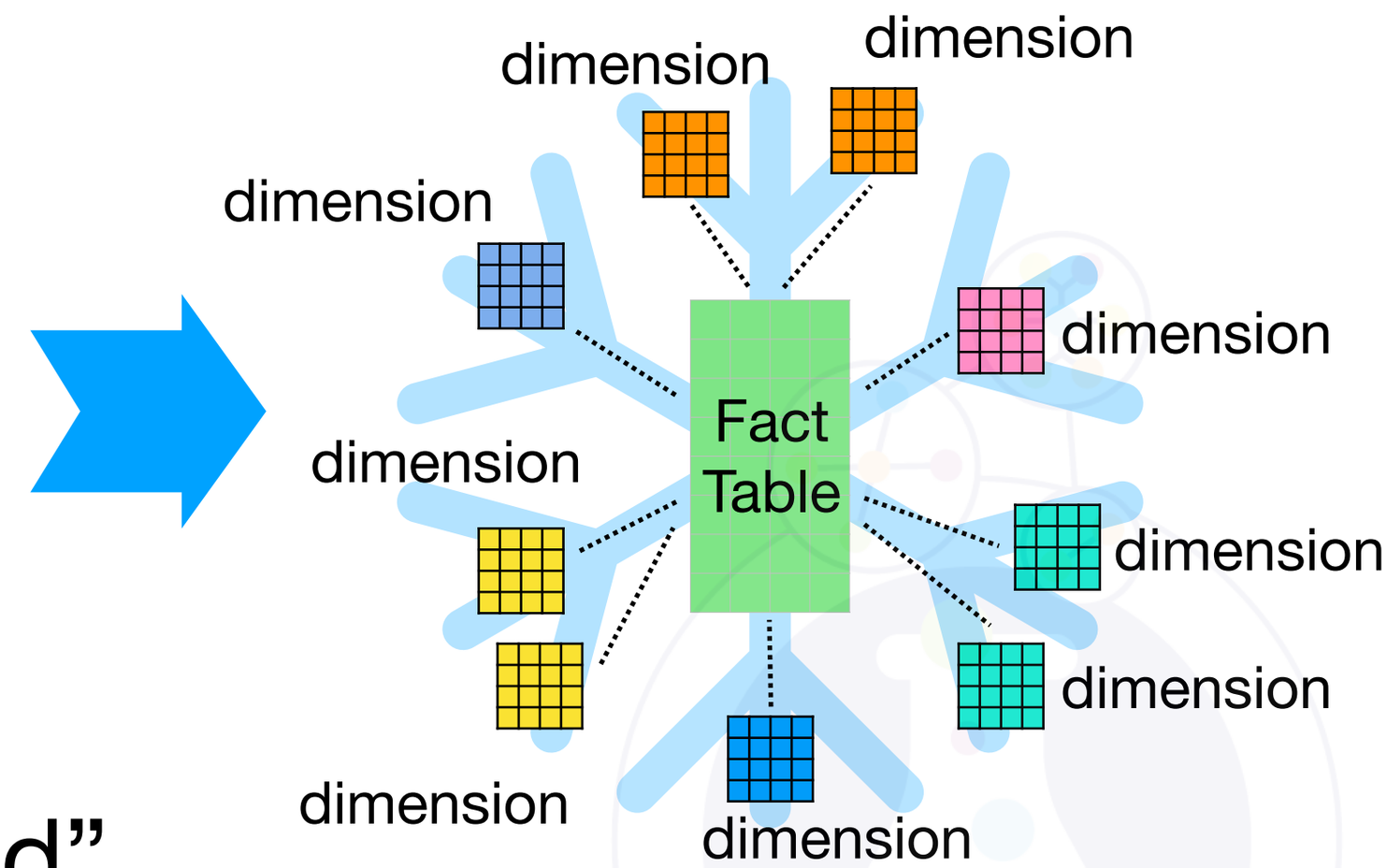


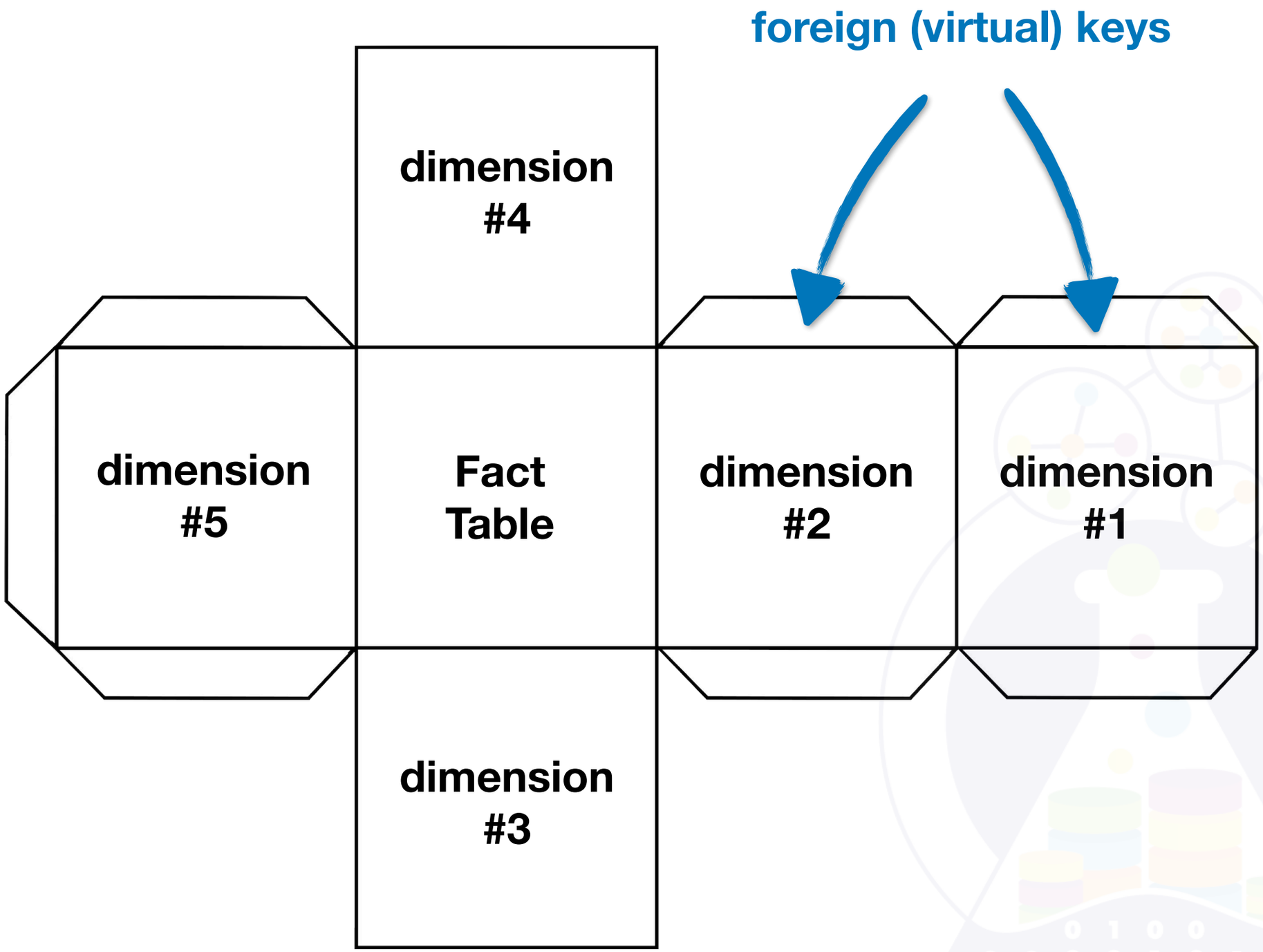
MOLAP Cube

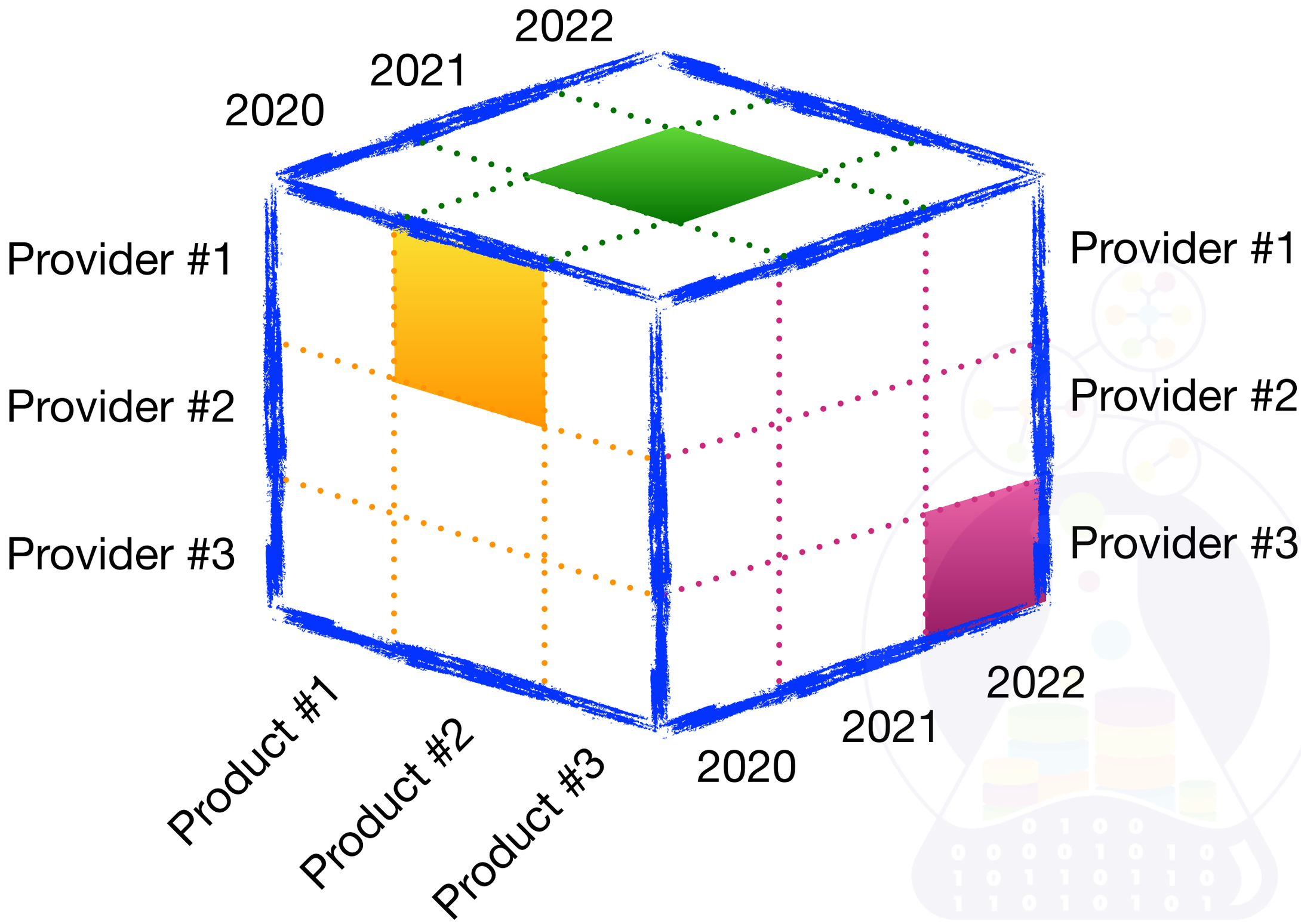


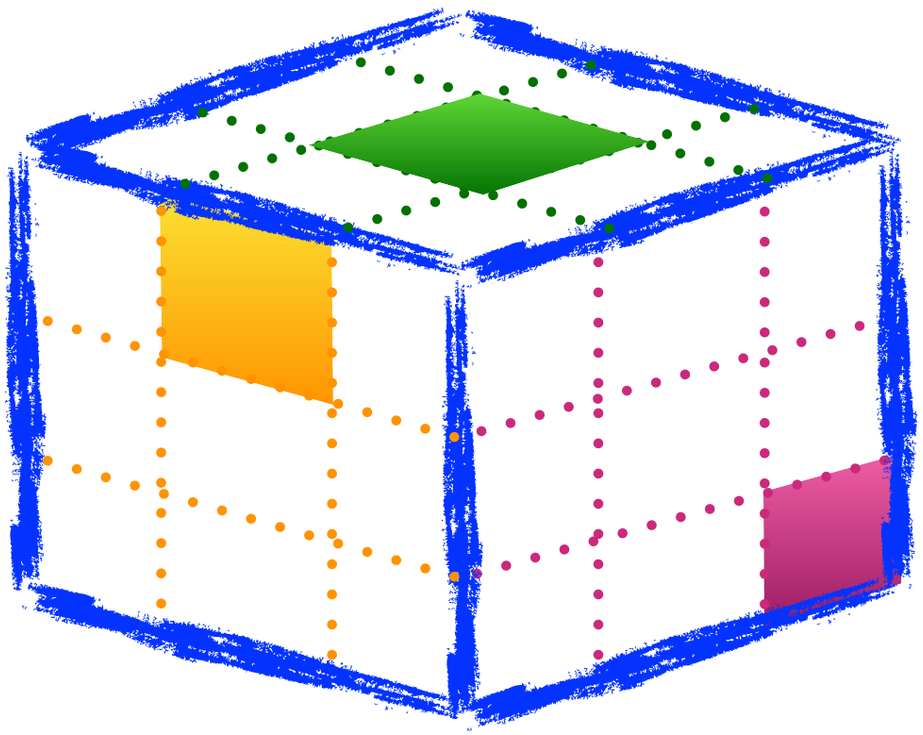


MOLAP “manifold”



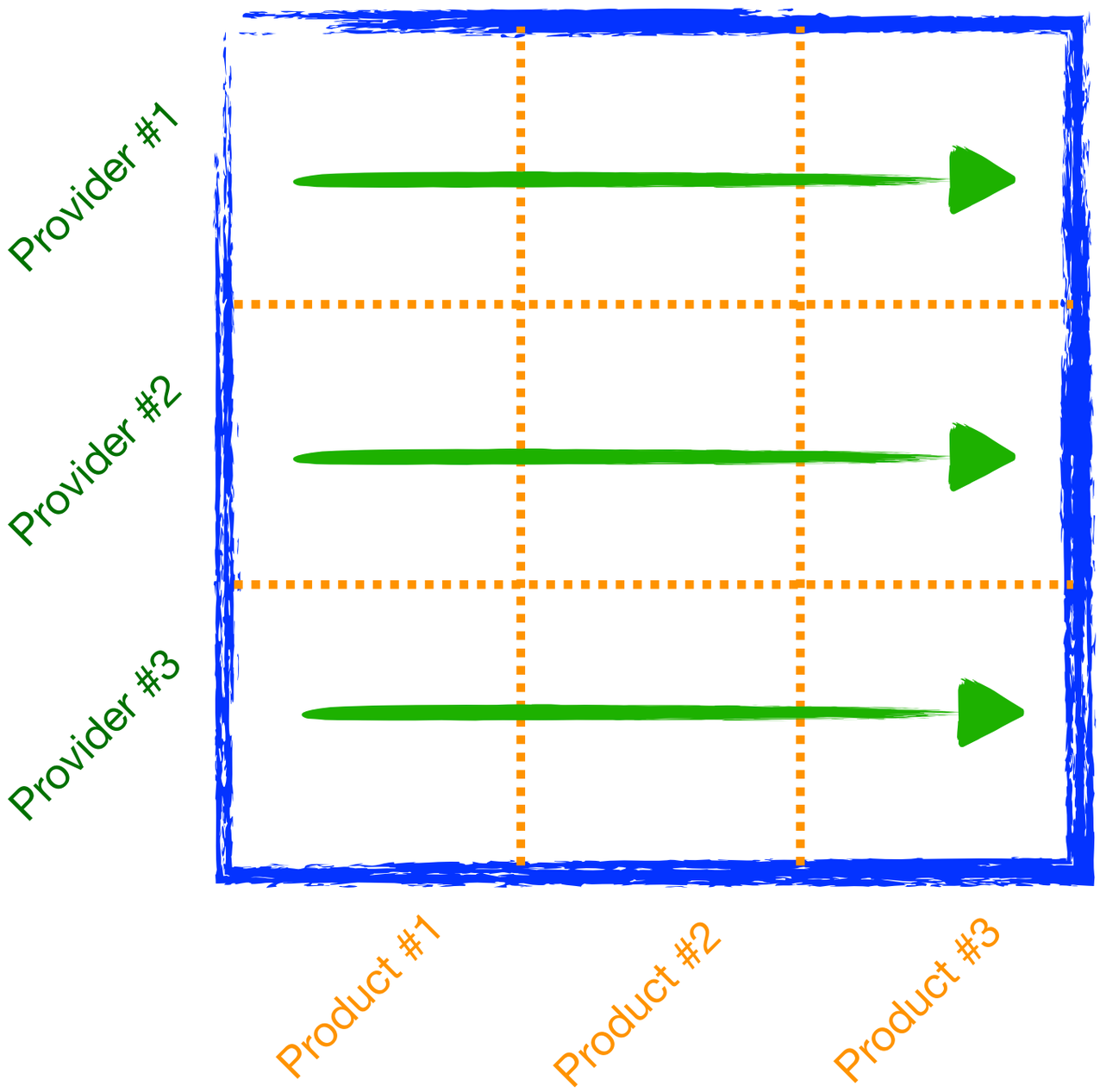






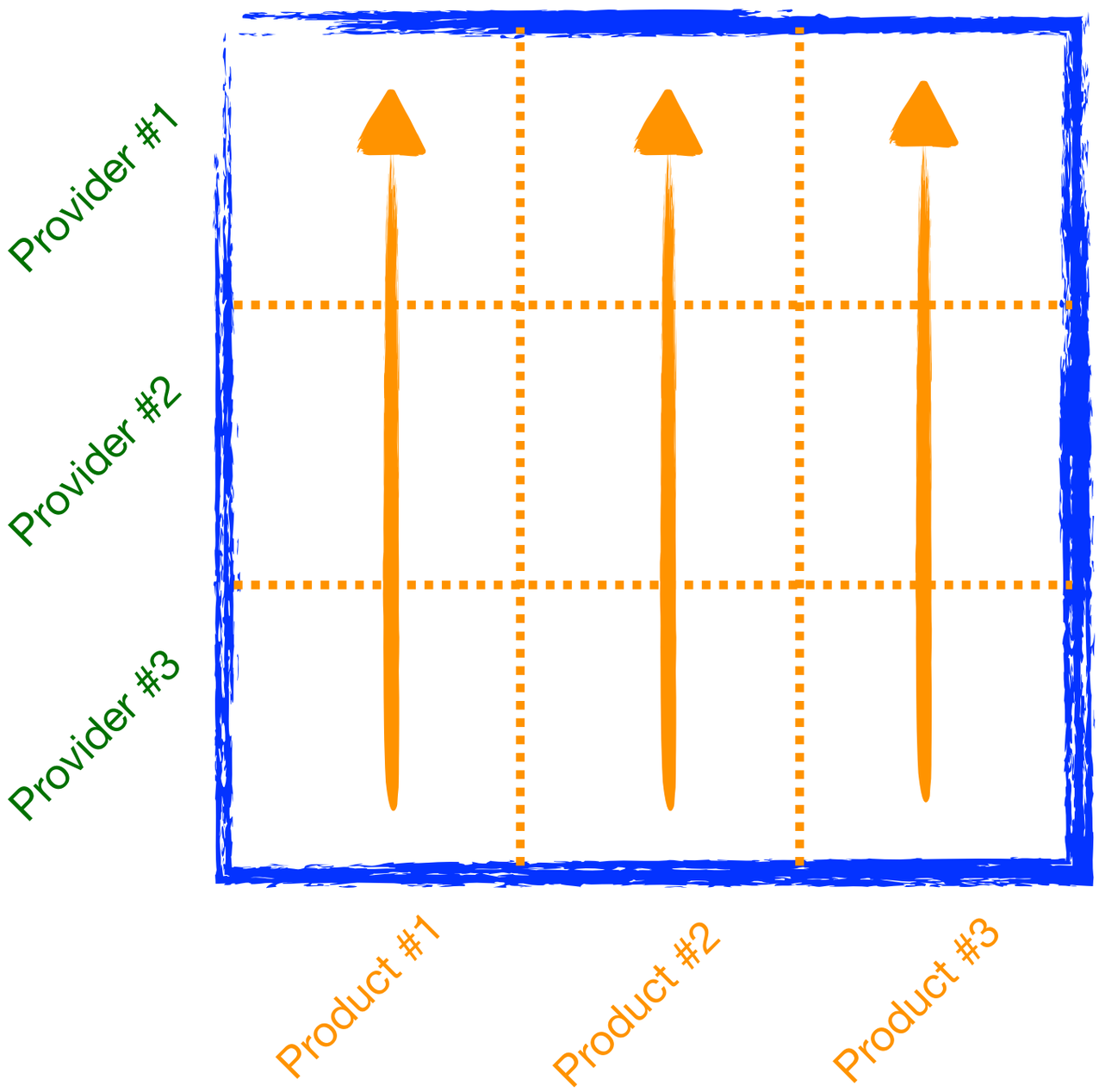
- Rotate
- Drill Up
- Drill Down

Drill Up



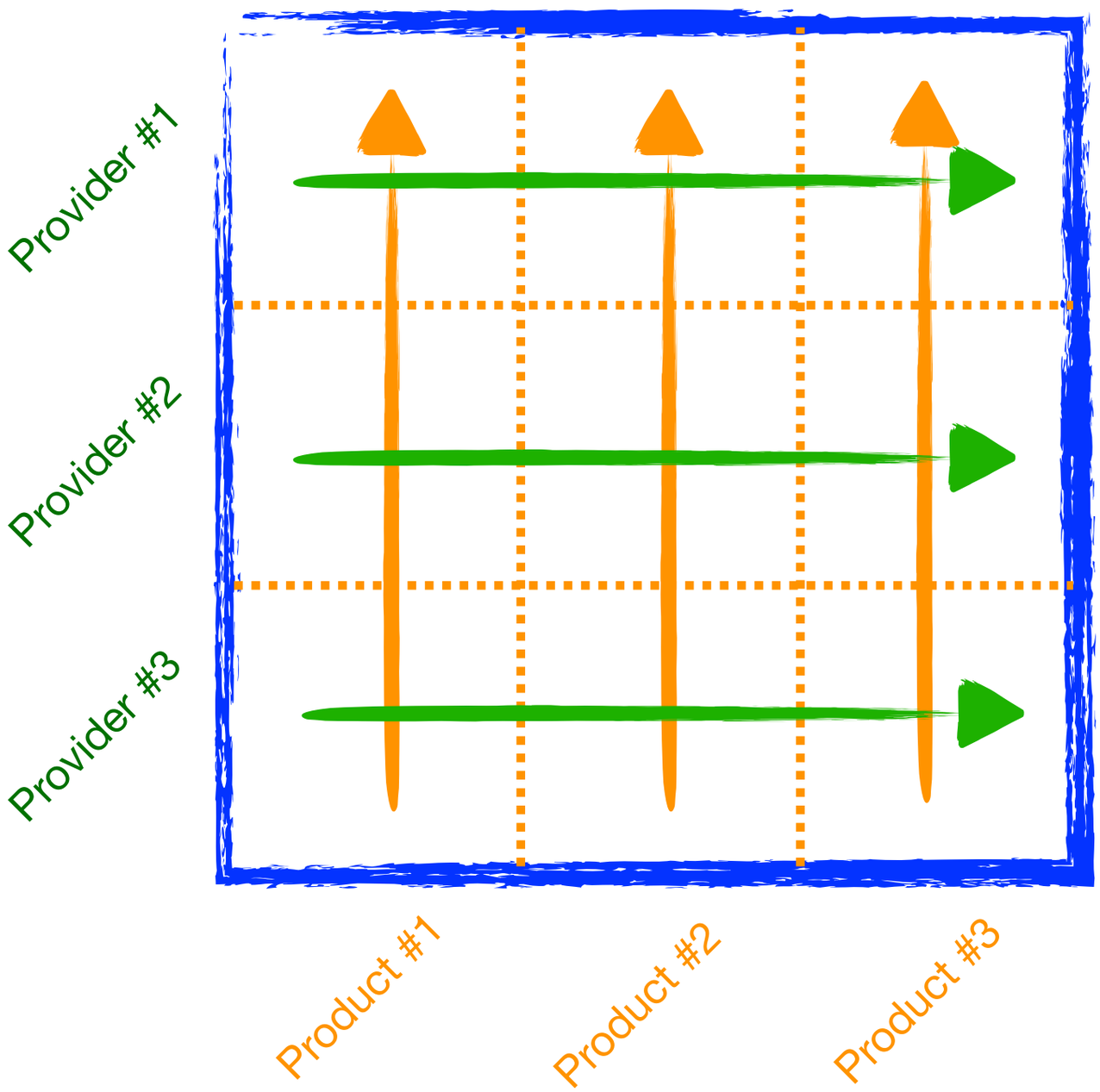

```
SELECT provider,
       count(product)
FROM table_fact
GROUP BY provider
```

Rotate + Drill Up

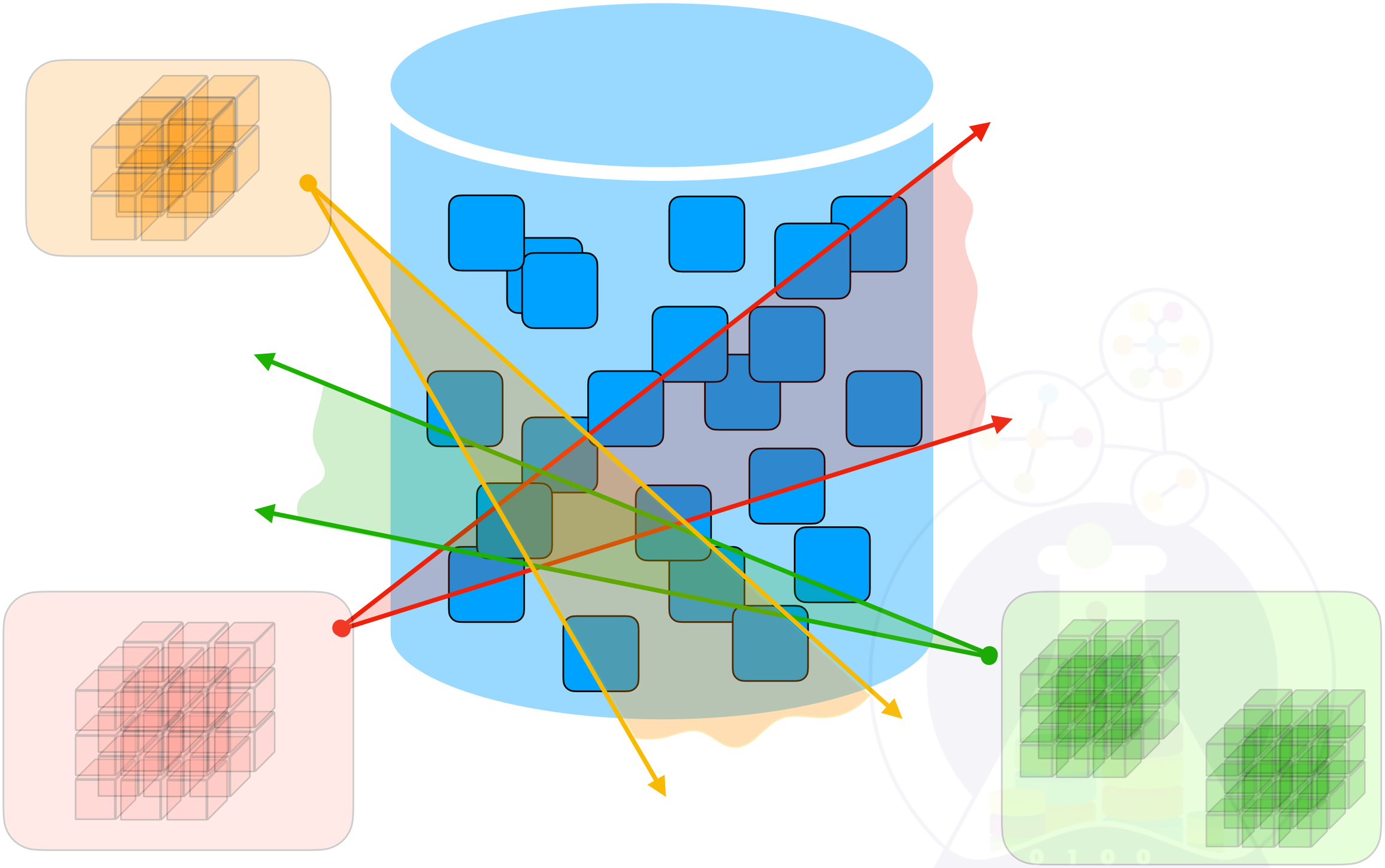



```
SELECT product,
       count(provider)
FROM table_fact
GROUP BY product
```

Drill Down




```
SELECT *
  FROM table_fact
```



+

provider	product	amount
s1	p1	300
s1	p2	200
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200

```
SELECT sum(amount) AS sm
  FROM delivery
```

sm
1600

provider	product	amount
s1	p1	300
s1	p2	200
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200

```
SELECT provider,
       sum(amount) AS sm
  FROM delivery
GROUP BY provider
ORDER BY provider
```

provider	sm
s1	500
s2	700
s3	200
s4	200

provider	product	amount
s1	p1	300
s1	p2	200
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200

```
SELECT product,
       sum(amount) AS sm
  FROM delivery
GROUP BY product
ORDER BY product
```

product	sm
p1	600
p2	1000

provider	product	amount
s1	p1	300
s1	p2	200
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200

```
SELECT provider, product,
       sum(amount) AS sm
  FROM delivery
GROUP BY provider, product
ORDER BY provider, product
```

provider	product	amount
s1	p1	300
s1	p2	200
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200

provider	product	amount
s1	p1	300
s1	p2	200
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200

```
SELECT provider,
       sum(amount) AS sm
  FROM delivery
GROUP BY provider
HAVING sum(amount) > 300
ORDER BY provider
```

provider	sm
s1	500
s2	700

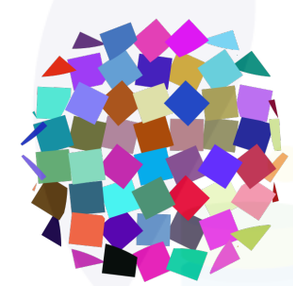
provider	product	amount
s1	p1	300
s1	p2	200
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200

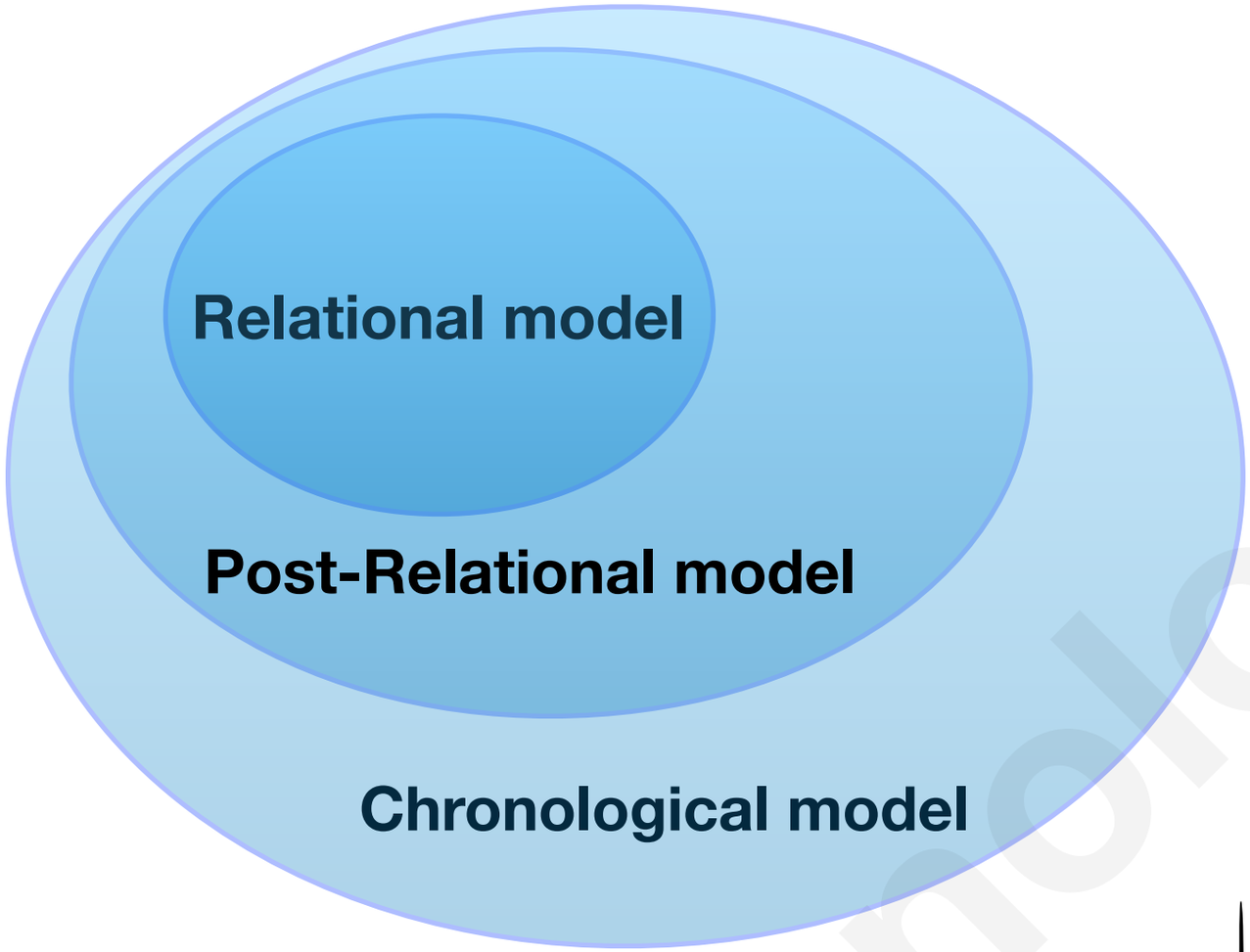
```
SELECT provider,
  sum(amount) FILTER
          (WHERE amount>300) AS sm
   FROM delivery
GROUP BY provider
ORDER BY provider
```

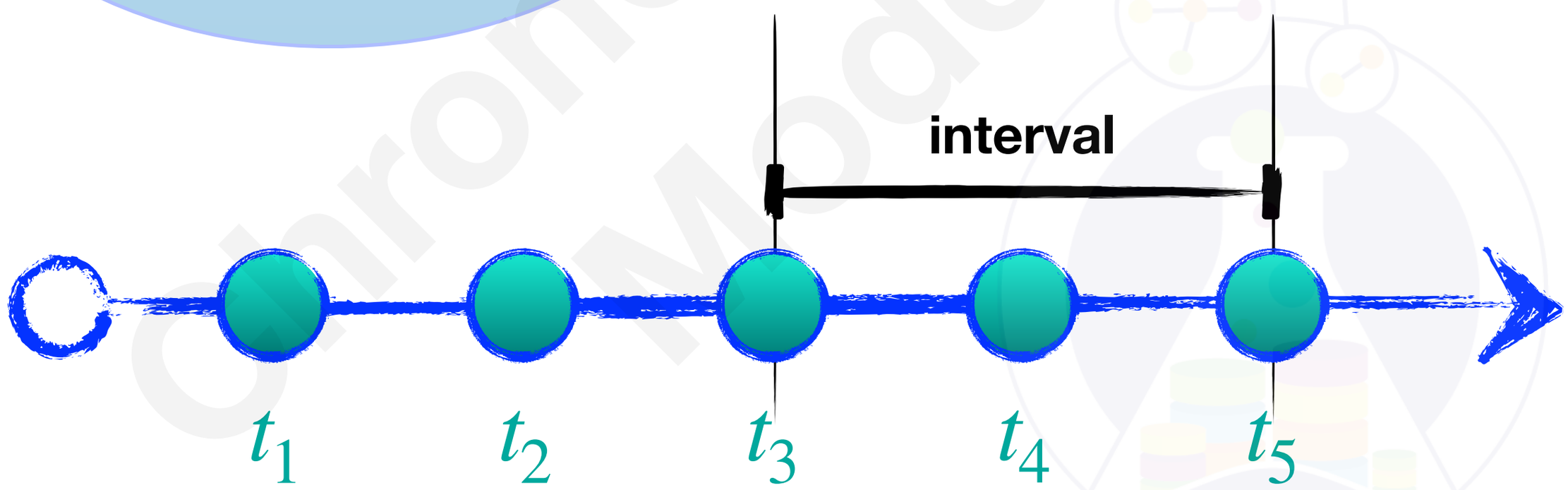
provider	sm
s1	*null*
s2	400
s3	*null*
s4	*null*

provider	product	amount
s1	p1	300
s1	p2	200
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200

```
SELECT provider,
       sum(amount) AS sm
  FROM delivery
GROUP BY provider
ORDER BY sm DESC
LIMIT 2
```







Student

Name	ID*
Ivan	1
Anna	2
Peter	3
Kate	4

Student_Fun

StudentID*	FunID*
1	1
1	3
2	1
2	2
4	3

Fun

Name	ID*
Party	1
Disco	2
Library	3

Student

Name	ID*	From
Ivan	1	01.09.2022
Anna	2	01.09.2022
Peter	3	01.09.2021
Kate	4	01.09.2020

```
SELECT *
  FROM student
WHERE From >= '01.09.2021'
```

```
SELECT *
  FROM fun
WHERE From >= '01.01.0000'
```

Fun

Name	ID*	From
Party	1	01.09.1982
Disco	2	01.09.1960
Library	3	01.01.0000

Student_Fun

StudentID*	FunID*	From
1	1	01.02.2021
1	3	01.01.2019
2	1	01.09.2022
2	2	01.09.2022
4	3	01.09.2020

```
SELECT *
  FROM student_fun
WHERE From >= '01.01.2020'
```

Student

Name	ID*	From*	To*
Ivan	1	01.09.2022	01.09.2023
Anna	2	01.09.2022	*null*
Peter	3	01.09.2021	*null*
Kate	4	01.09.2020	01.09.2025
Ivan	1	02.09.2023	*null*

```
SELECT *
  FROM student
WHERE To IS NULL
```

Student

Name	ID*	From*	To*
Ivan	1	01.09.2022	01.09.2023
Anna	2	01.09.2022	*null*
Peter	3	01.09.2021	*null*
Kate	4	01.09.2020	01.09.2025
Ivan	1	02.09.2023	*null*

```
SELECT *
  FROM student
 WHERE now( ) BETWEEN
                 From AND To
```

Student_Fun

StudentID*	FunID*	From*	To*
1	1	01.02.2021	31.05.2021
1	3	01.01.2019	31.05.2019
2	1	01.09.2022	31.12.2022
2	2	01.09.2022	*null*
4	3	01.09.2020	*null*

```
SELECT *
  FROM student_fun
WHERE now( ) BETWEEN
                 From AND To
```

```
SELECT *
  FROM fun
WHERE now( ) BETWEEN
                 From AND To
```

Fun

Name	ID*	From*	To*
Party	1	01.09.1982	01.01.9999
Disco	2	01.09.1960	01.01.9999
Library	3	01.01.0000	01.01.9999

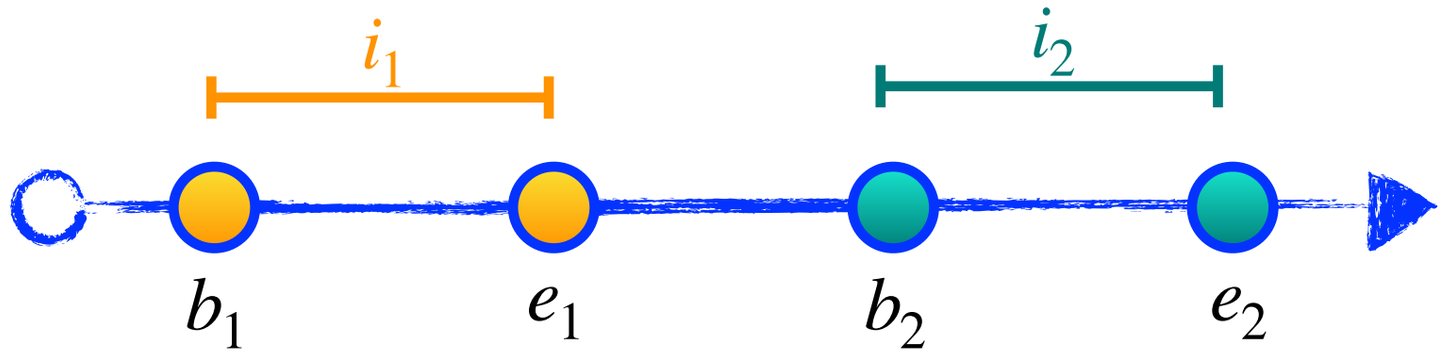
Student

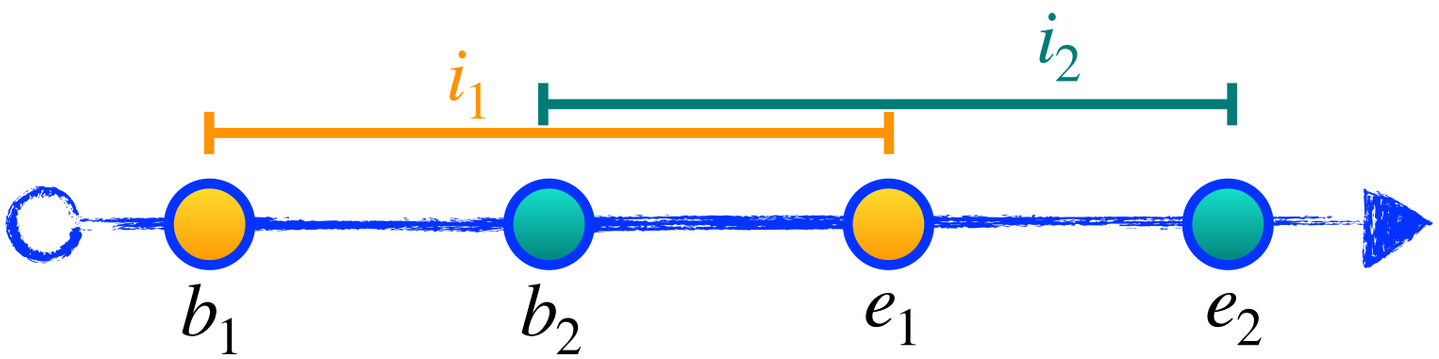
Name	ID*	From*	To*
Ivan	1	01.09.2022	01.09.2023
Anna	2	01.09.2022	*null*
Peter	3	01.09.2021	*null*
Kate	4	01.09.2020	01.09.2025
Ivan	1	02.09.2023	*null*

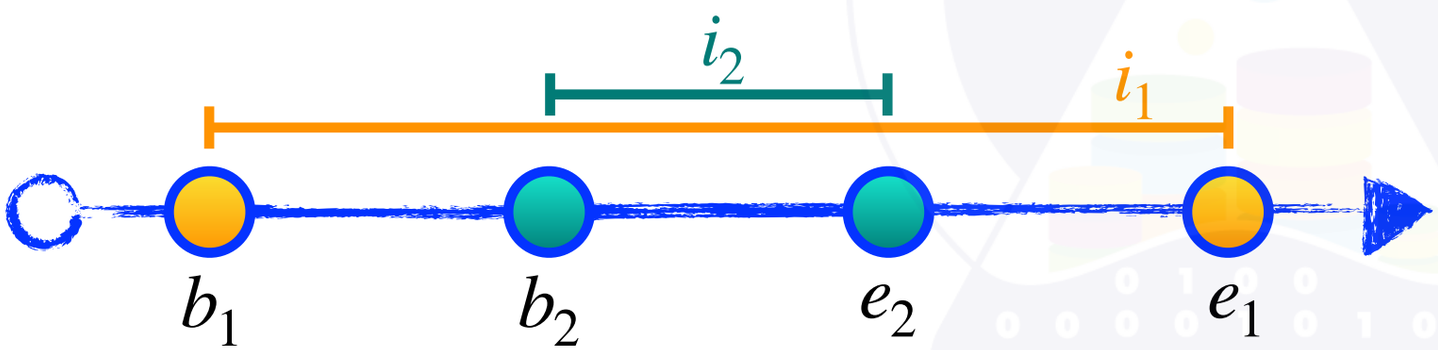
Student

Name	ID*	During*
Ivan	1	[01.09.2022 : 01.09.2023]
Anna	2	[01.09.2022 : 01.01.9999]
Peter	3	[01.09.2021 : 01.01.9999]
Kate	4	[01.09.2020 : 01.09.2025]
Ivan	1	[02.09.2023 : 01.01.9999]

$$i_1 = [b_1 : e_1],\ i_2 = [b_2 : e_2]$$







$$i_1 = [b_1 : e_1], i_2 = [b_2 : e_2]$$

$$i_1 = i_2 \;\Rightarrow b_1 = b_2 \; AND \; e_1 = e_2$$

$$i_1 \supseteq i_2 \;\Rightarrow b_1 \leqslant b_2 \; AND \; e_1 \geqslant e_2$$

$$i_1 \supset i_2 \;\Rightarrow i_1 \supseteq i_2 \; AND \; i_1 \neq i_2$$

$$i_1 \; BEFORE \; i_2 \;\Rightarrow e_1 < b_2;$$

$$i_1 \; AFTER \; i_2 \;\equiv i_2 \; BEFORE \; i_1$$

$$i_1 = [b_1 : e_1],\ i_2 = [b_2 : e_2]$$

$$i_1\ MEETS\ i_2 \ \Rightarrow b_2 = e_1 + 1\ OR\ b_1 = e_2 + 1$$

$$i_1\ OVERLAPS\ i_2 \ \Rightarrow b_1 \leqslant e_2\ AND\ b_2 \leqslant e_1$$

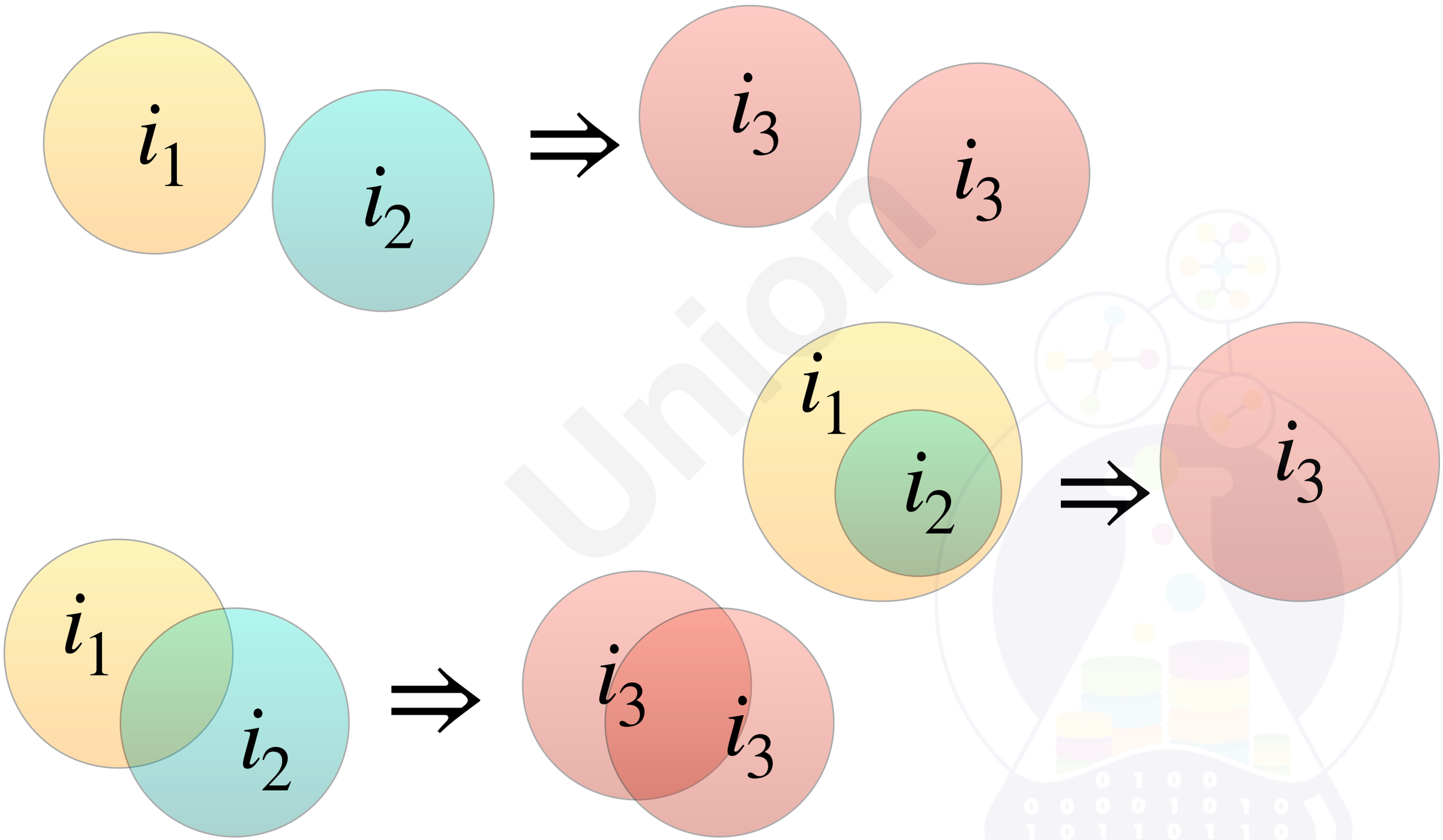
$$i_1\ MERGES\ i_2 \ \Rightarrow i_1\ OVERLAPS\ i_2$$

$$OR\ i_1\ MEETS\ i_2$$

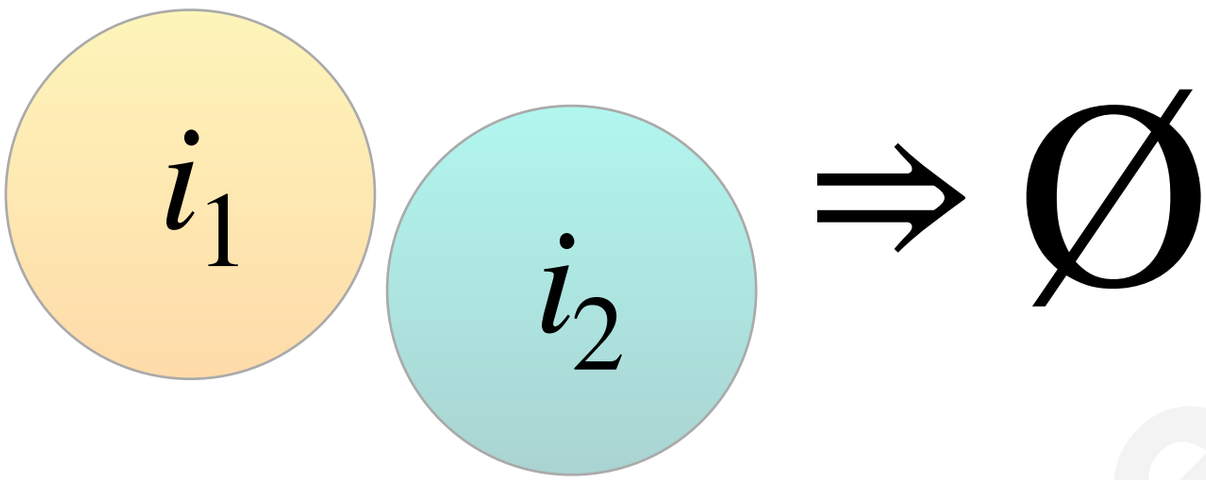
$$i_1\ BEGINS\ i_2 \ \Rightarrow b_1 \ = \ b_2\ AND\ e_1 \ \leqslant \ e_2$$

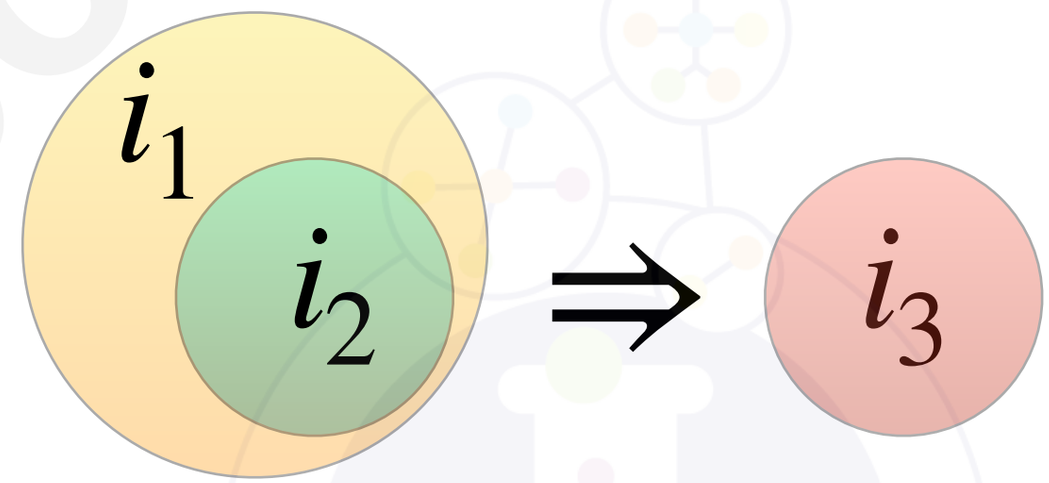
$$i_1\ ENDS\ i_2 \ \Rightarrow e_1 \ = \ e_2\ AND\ b_1 \ \geqslant \ b_2$$

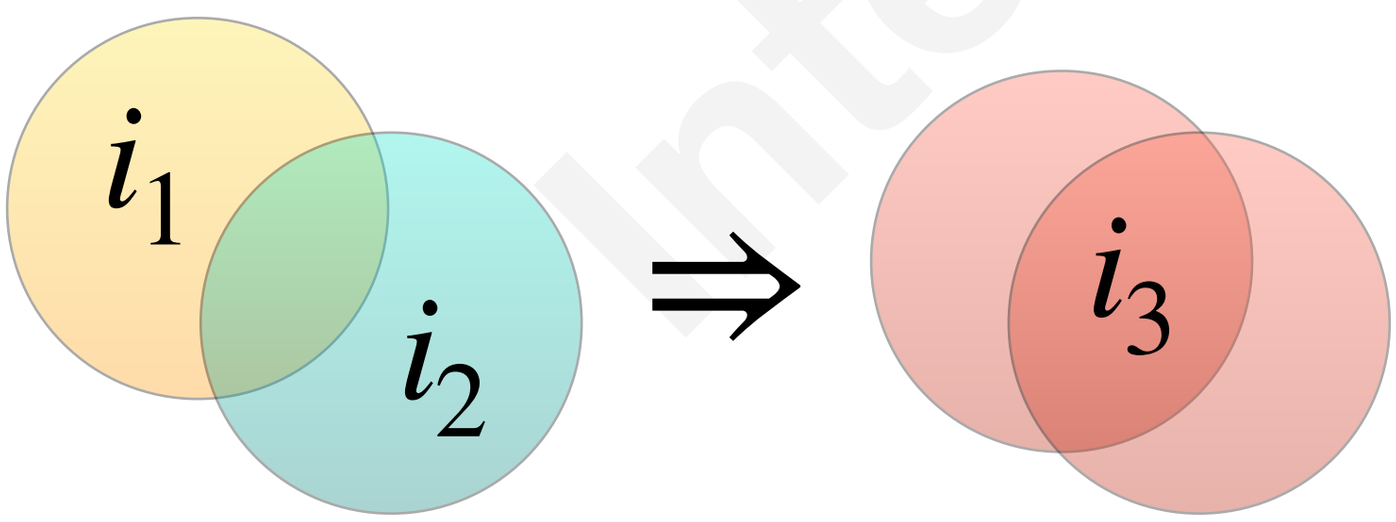
$$i_1 = [b_1 : e_1],\ i_2 = [b_2 : e_2]$$



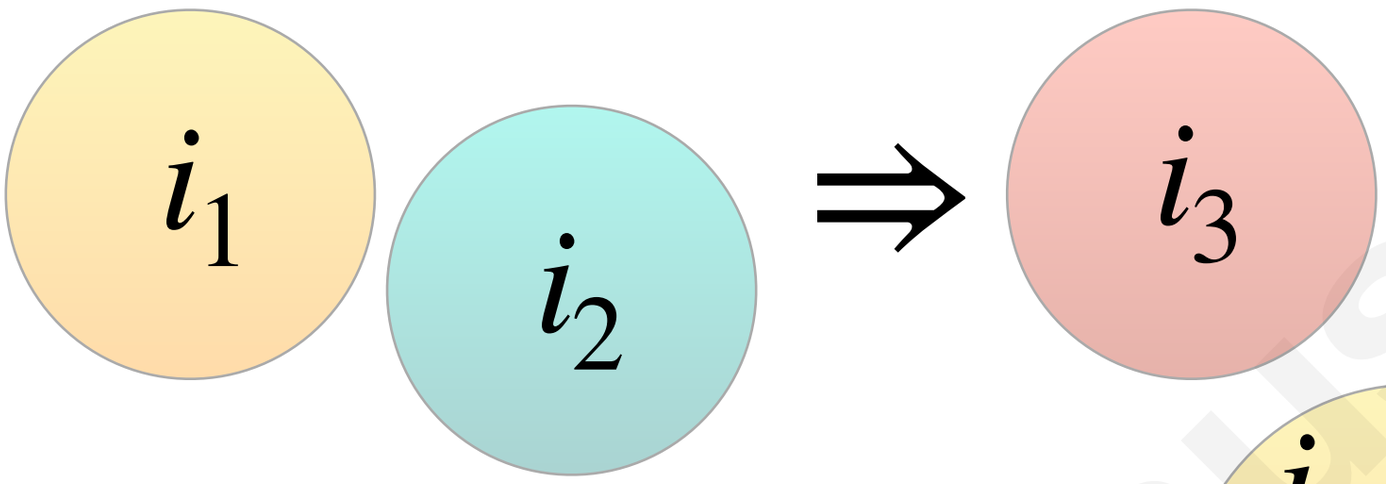
$$i_1 = [b_1 : e_1],\ i_2 = [b_2 : e_2]$$

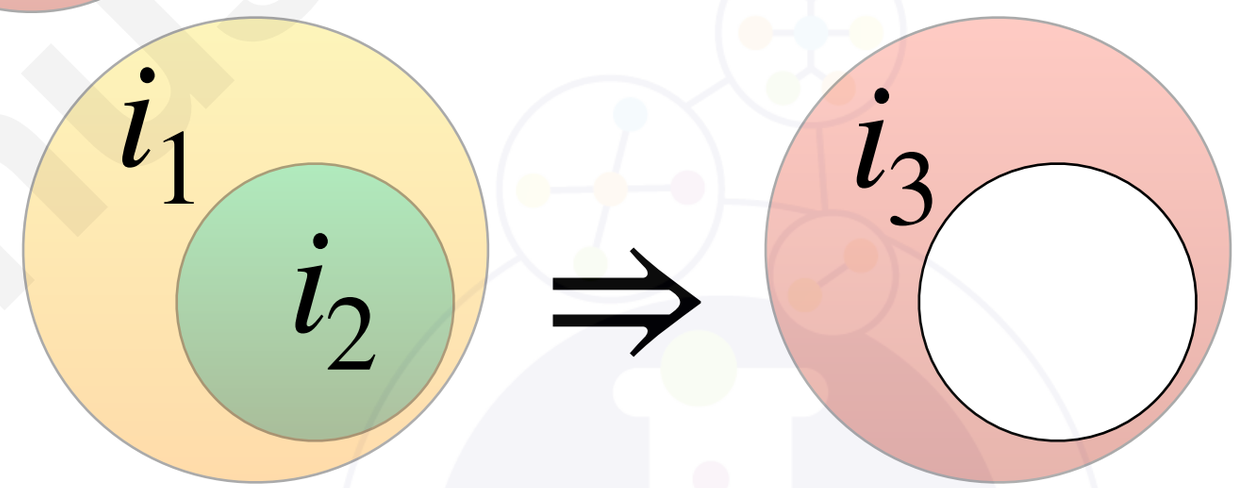


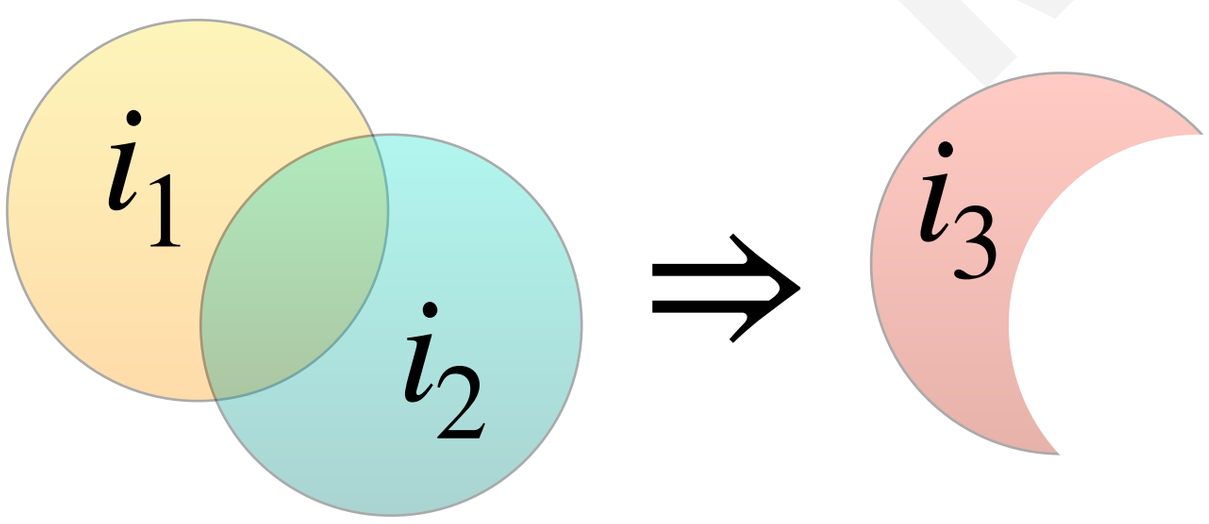




$$i_1 = [b_1 : e_1],\ i_2 = [b_2 : e_2]$$







$$i_1 = [b_1 : e_1],\ i_2 = [b_2 : e_2]$$

$$i_1\ UNION\ i_2\ \Rightarrow [MIN(b_1, b_2) : MAX\,(e_1, e_2)]$$

$$i_1\ UNION\ i_2\ \Rightarrow (i_1\ MERGES\ i_2) = True$$

$$i_1\ INTERSECT\ i_2\ \Rightarrow [MAX(b_1, b_2) : MIN\,(e_1, e_2)]$$

$$i_1\ INTERSECT\ i_2\ \Rightarrow (i_1\ OVERLAPS\ i_2) = True$$

$$i_1\ MINUS\ i_2\ \Rightarrow [b_1 : MIN(b_2 - 1, e_1)] \mid b_1 < b_2\ AND\ e_1 \leq e_2$$

$$[MAX(e_2 + 1, b_1) : e_1] \mid b_1 \geq b_2\ AND\ e_1 > e_2$$

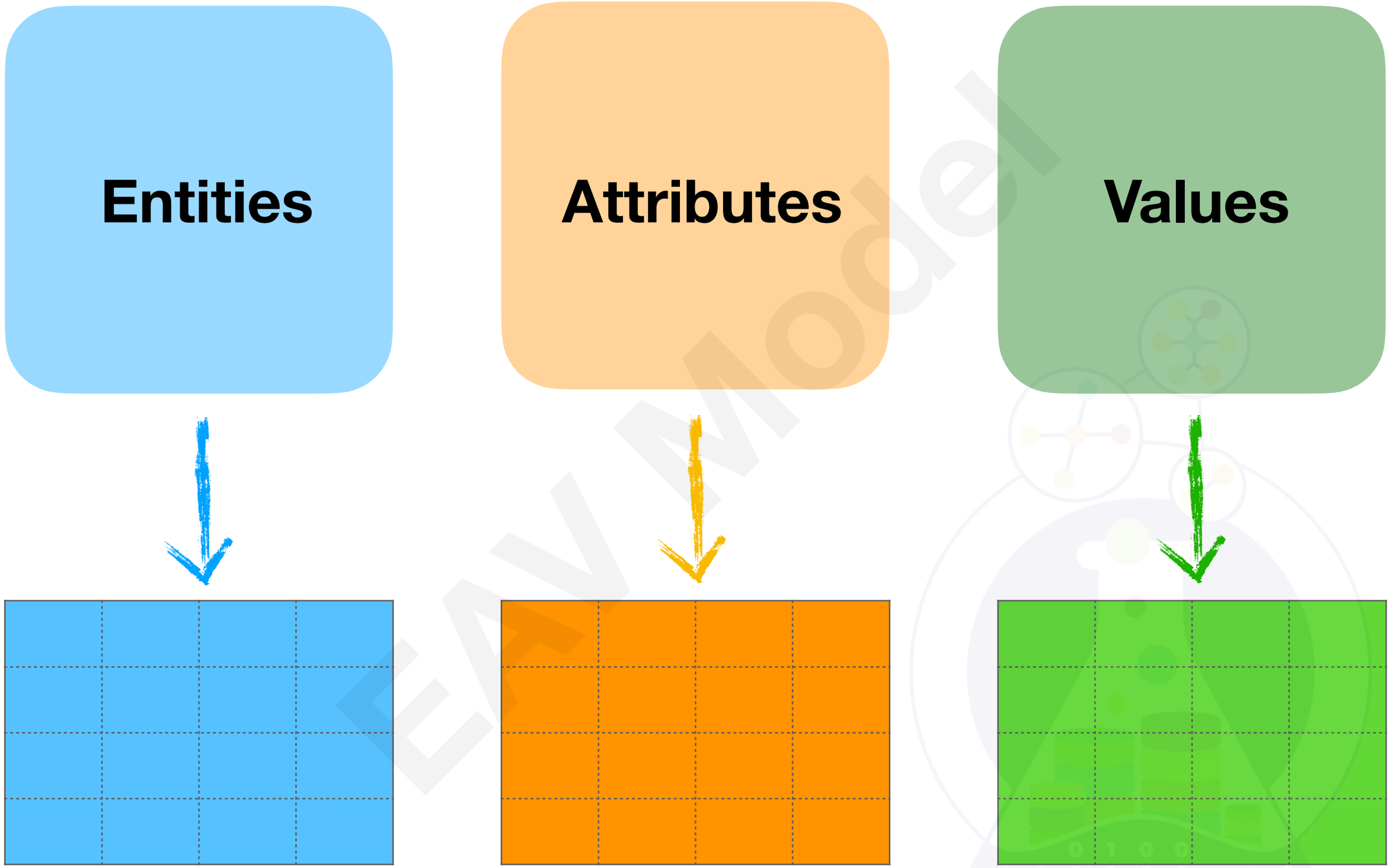
```
CREATE TABLE reservation (
    during TSTZRANGE,
    EXCLUDE USING GIST (during WITH &&)
);

INSERT INTO reservation VALUES
    ('[2023-01-01 11:30, 2023-01-15 15:00)');

INSERT INTO reservation VALUES
    ('[2023-01-15 15:01, 2023-01-16 15:00]');
```

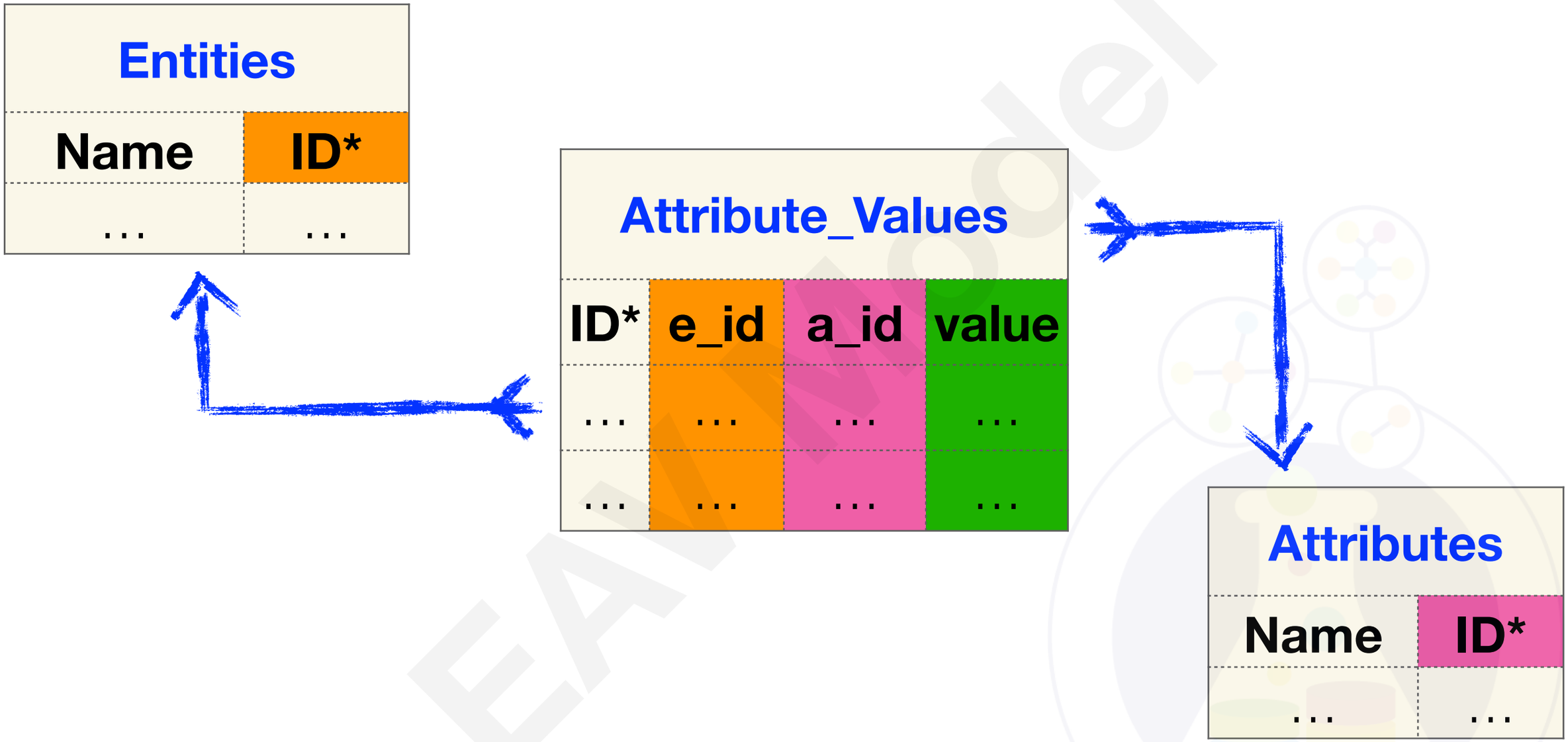
```
SELECT *
  FROM reservation;
```

During
["2023-01-01 11:30:00+00","2023-01-15 15:00:00+00")
["2023-01-15 15:01:00+00","2023-01-16 15:00:00+00"]

```
SELECT *
  FROM reservation
WHERE during @>
 '[2023-01-01 11:30, 2023-01-12 15:00]';
```

During
["2023-01-01 11:30:00+00","2023-01-15 15:00:00+00")





Products

Name	ID*
TV	1
Fridge	2

Products_Properties

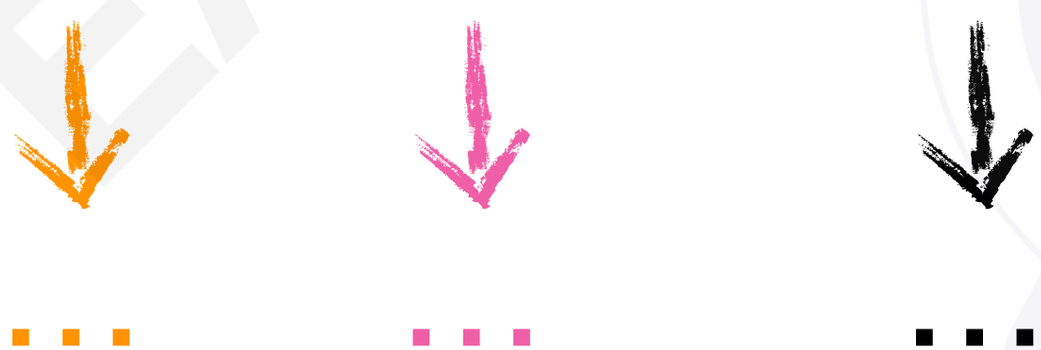
ID*	e_id	a_id	value
1	1	1	25
2	1	2	true
3	2	3	-15
4	2	1	60

Properties

Name	ID*
weight	1
is_led	2
min_temp	3

EAV model
(Entity-Attribute-Value)

user_id	key	value
1	first_name	Petr
1	last_name	Petrov
2	first_name	Ivan
2	last_name	Ivanov
...	...	...



Relational model

user_id	first_name	last_name	...
1	Petr	Petrov	...
2	Ivan	Ivanov	...

→ ...

Indicator

Name	ID*
boiler	1

Indicator_Property

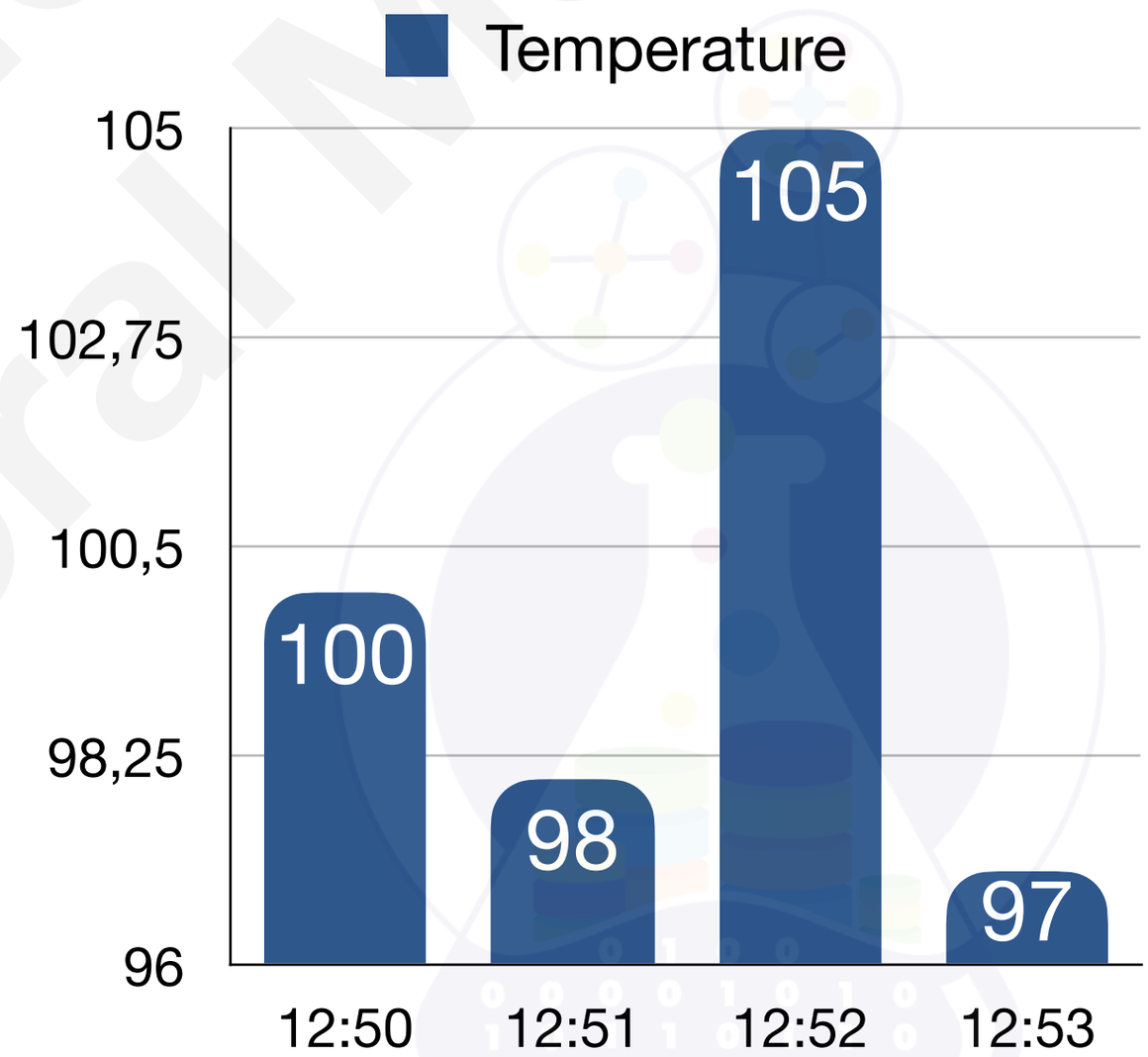
ID*	i_id	p_id	value	time
1	1	100	100	12:50
2	1	100	98	12:51
3	1	100	105	12:52
4	1	100	97	12:53

Property

Name	ID*
temperature	100

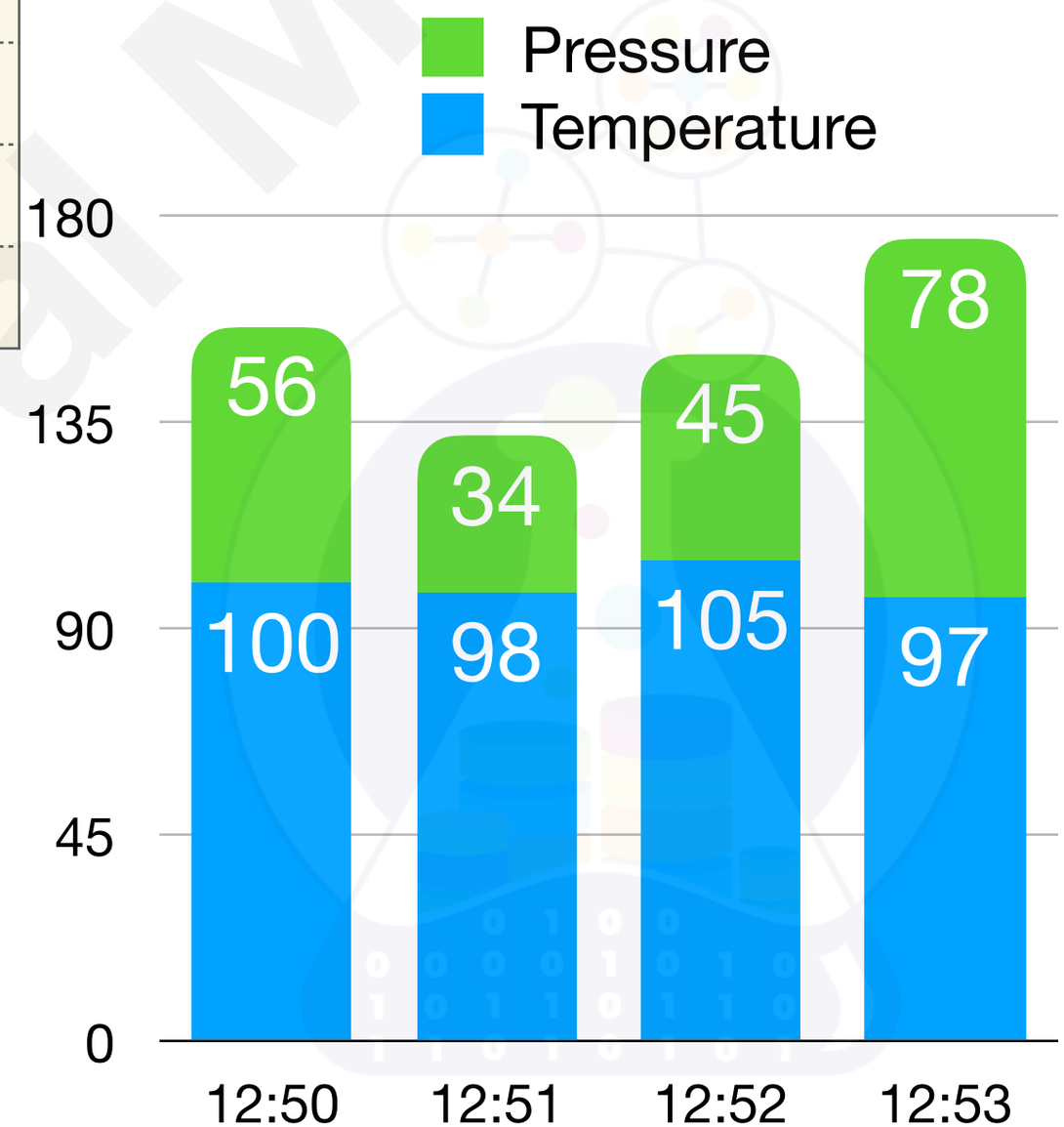
```
SELECT value, time
  FROM Indicator_Property
WHERE i_id = 1 AND p_id = 100
ORDER BY time;
```

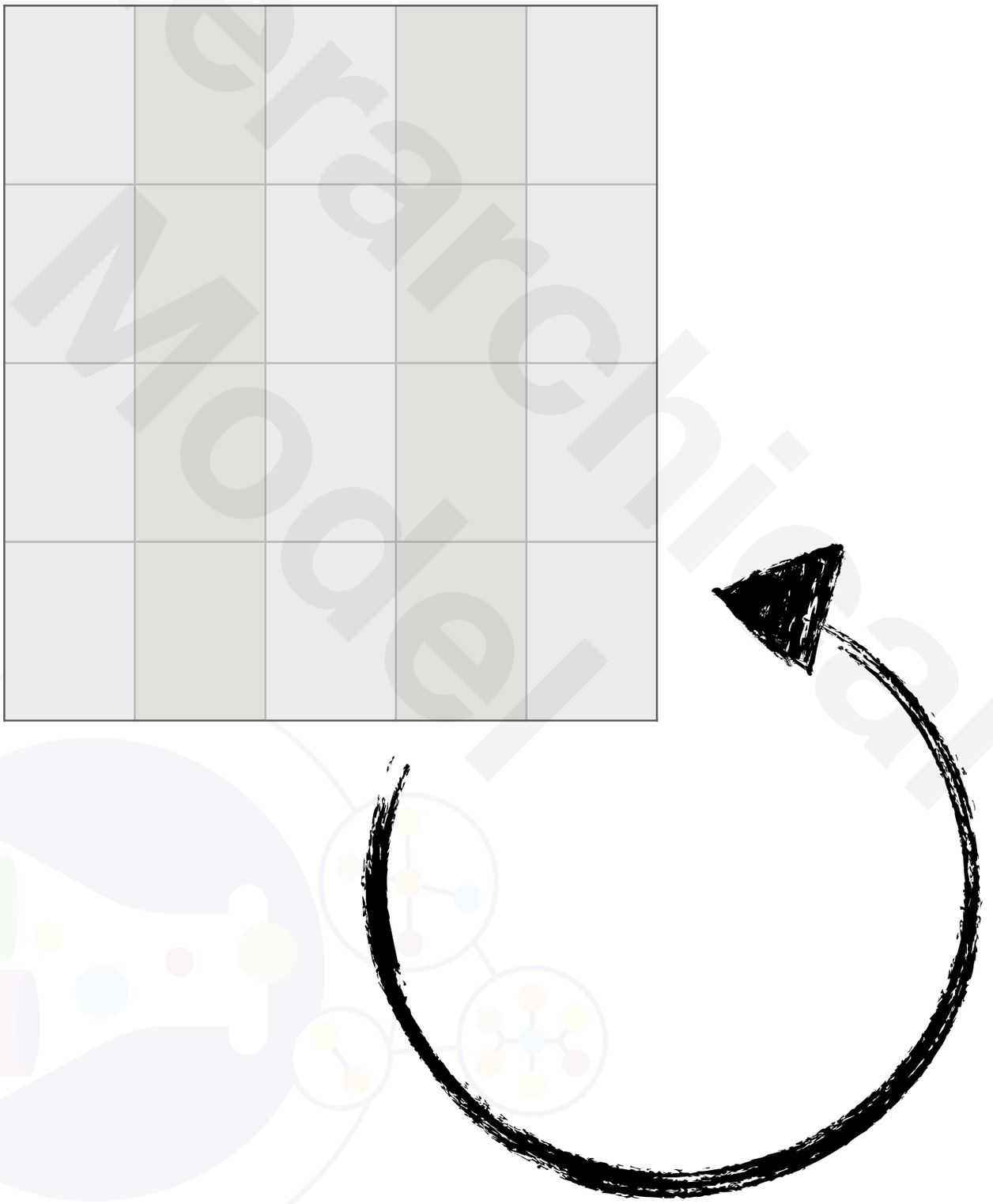
value	time
100	12:50
98	12:51
105	12:52
97	12:53

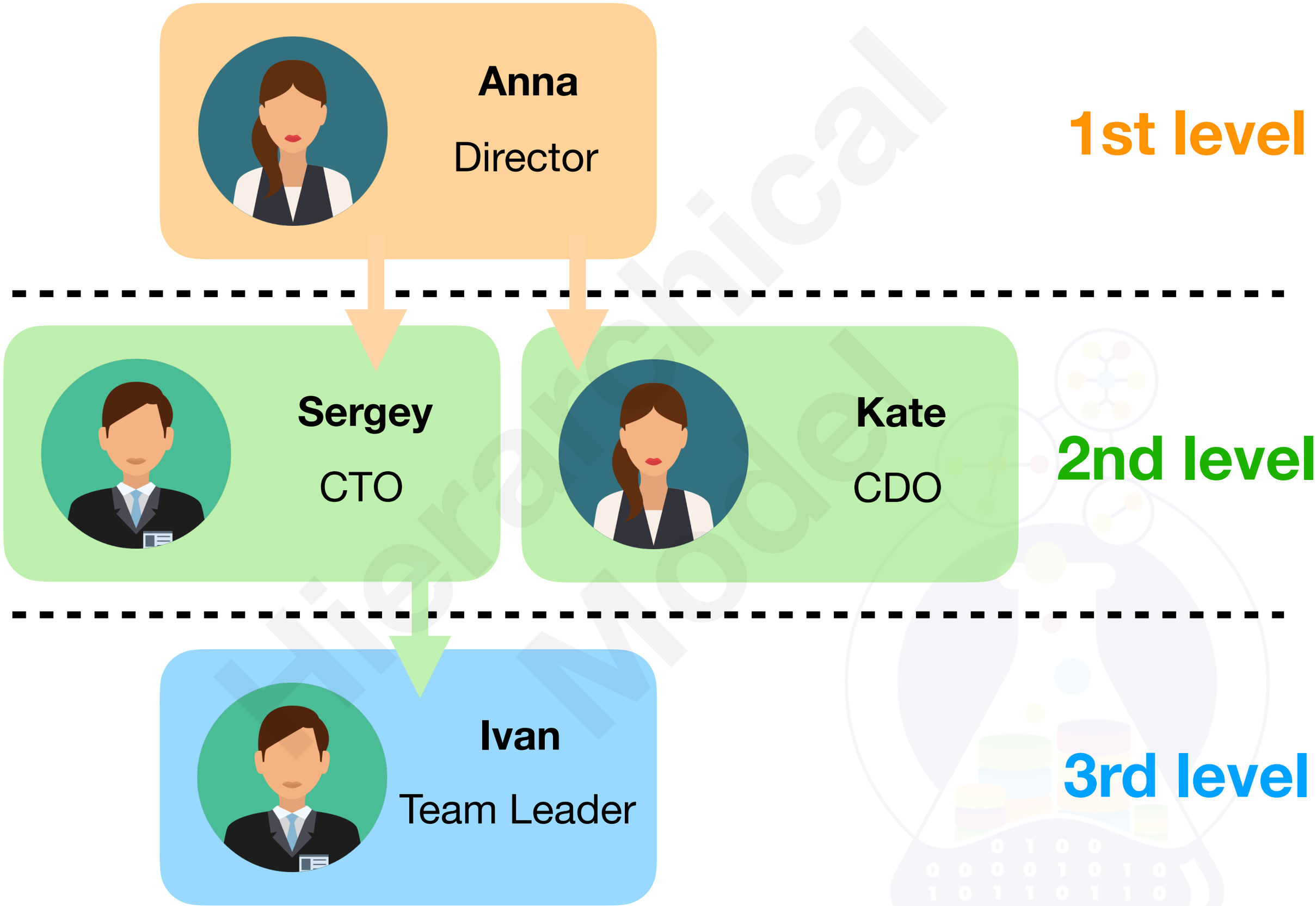


```
SELECT i.name, p.name, value, time
  FROM Indicator_Property ip
        INNER JOIN indicator i ON
              i.id = ip.i_id
        INNER JOIN property p ON
              p.id = ip.p_id
ORDER BY time;
```

i.name	p.name	value	time
boiler	temperature	100	12:50
boiler	temperature	98	12:51
boiler	temperature	105	12:52
boiler	temperature	97	12:53
boiler	pressure	56	12:50
boiler	pressure	34	12:51
boiler	pressure	45	12:52
boiler	pressure	78	12:53







```
CREATE TABLE managers
( ID SERIAL PRIMARY KEY,
  NAME VARCHAR NOT NULL,
  POST VARCHAR NOT NULL,
  MANAGER_ID INTEGER,

    CONSTRAINT manager_fk
    FOREIGN KEY (MANAGER_ID)
    REFERENCES managers (ID)
);
```

foreign key

id	name	post	manager_id
100	Anna	Director	*null*
200	Sergey	CTO	100
300	Kate	CDO	100
400	Ivan	Team Leader	200

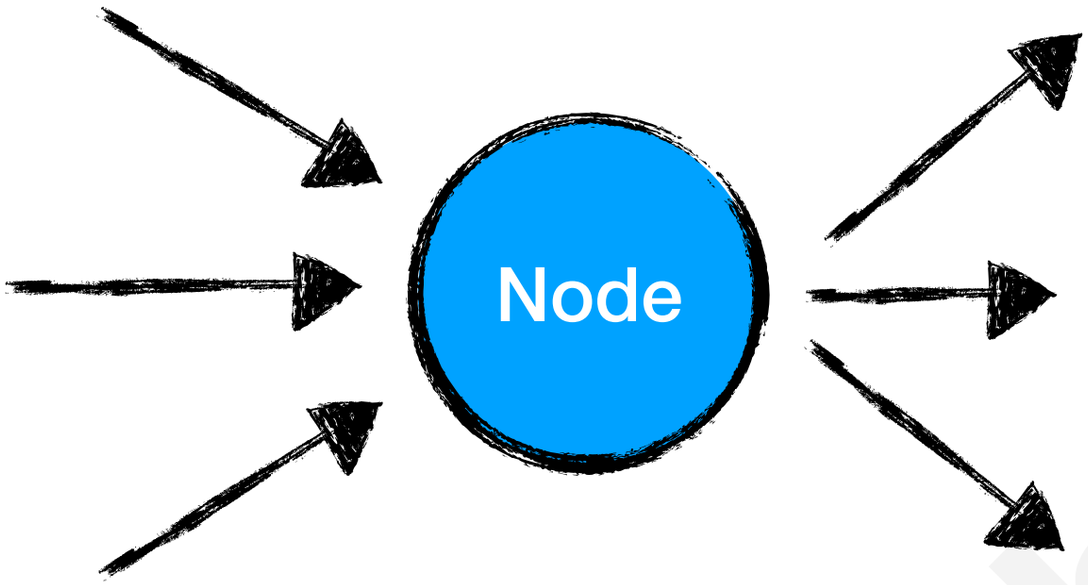
```
SELECT m1.name AS manager_name,
       m2.name
  FROM managers AS m1
       INNER JOIN managers AS m2
       ON  m1.id = m2.manager_id;
```

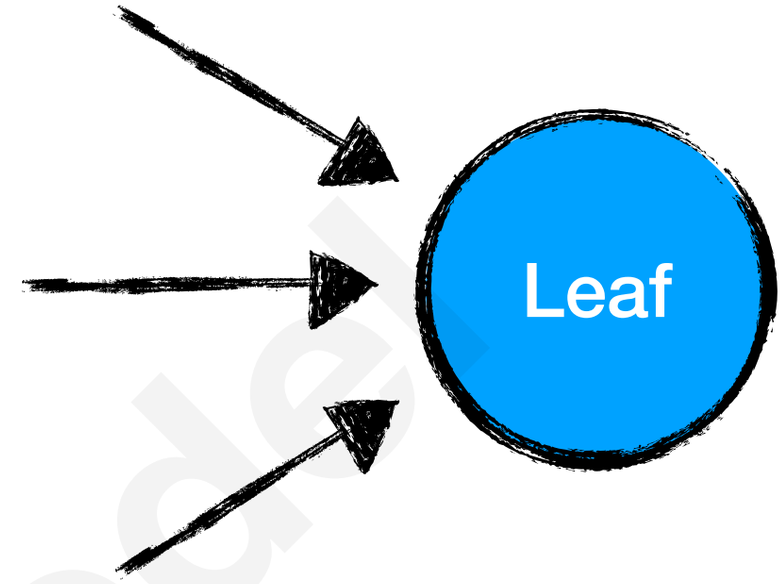
manager_name	name
Anna	Sergey
Anna	Kate
Sergey	Ivan

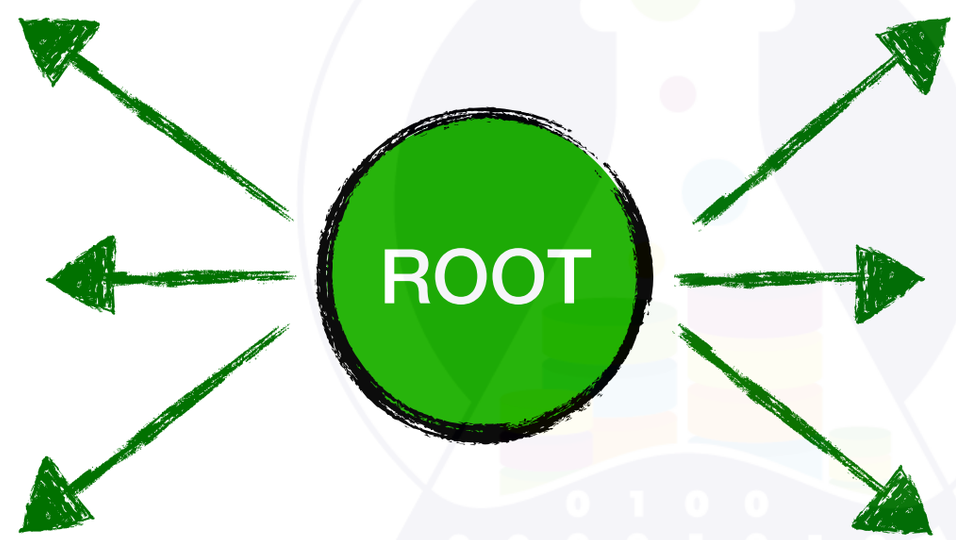
```
WITH RECURSIVE _managers AS
  (
    SELECT id,
           manager_id,
           name,
           1 AS level,
           array[name] AS path
     FROM managers
    WHERE manager_id IS NULL
     UNION
     SELECT m1.id,
            m1.manager_id,
            m1.name,
            level + 1 AS level,
            path || m1.name  AS path
        FROM managers AS m1 INNER JOIN
             _managers AS m2 ON m1.manager_id = m2.id
  )
SELECT name,
       level,
       path
FROM _managers
```

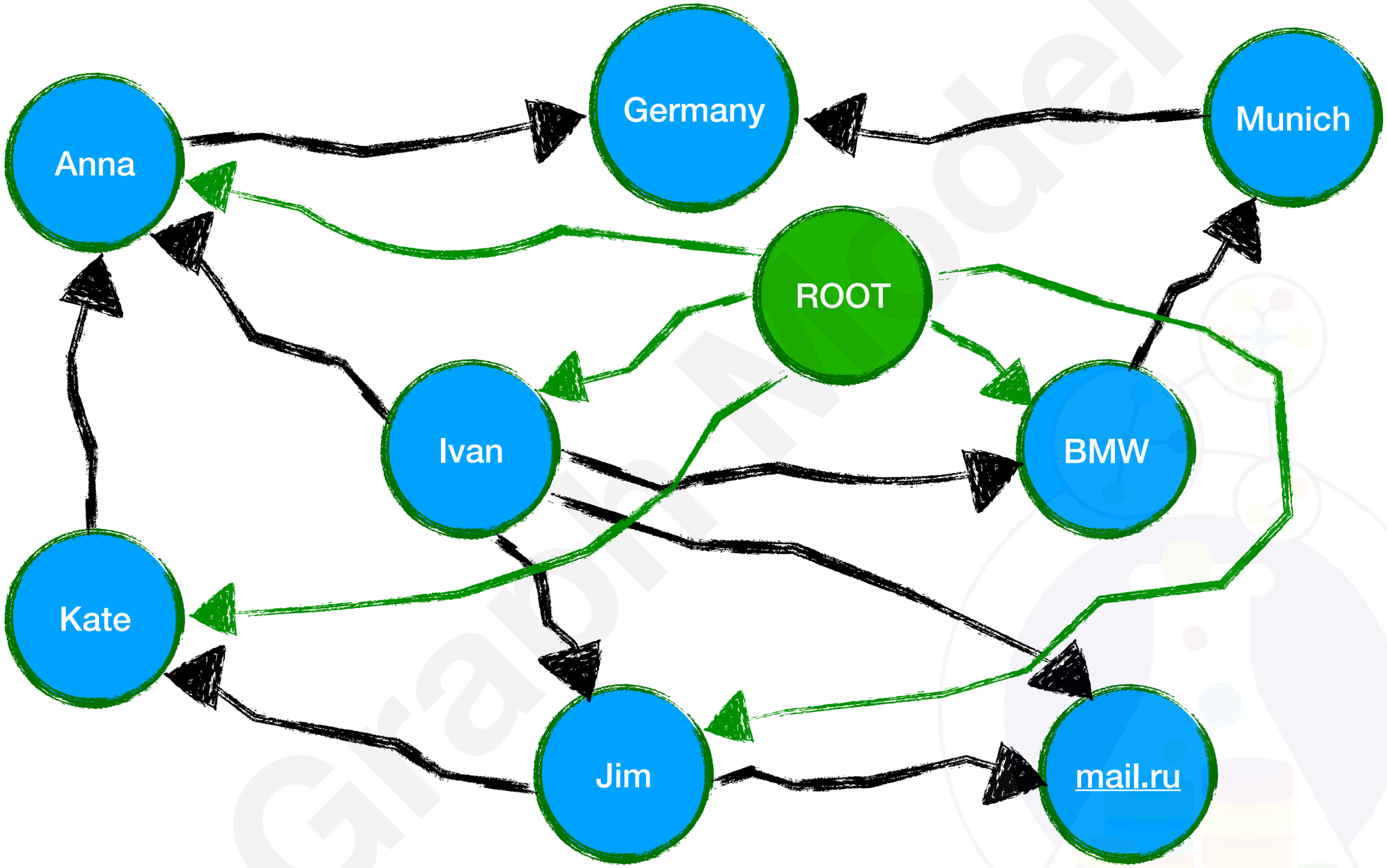
name	level	path
Anna	1	{Anna}
Sergey	2	{Anna,Sergey}
Kate	2	{Anna,Kate}
Ivan	3	{Anna,Sergey,Ivan}

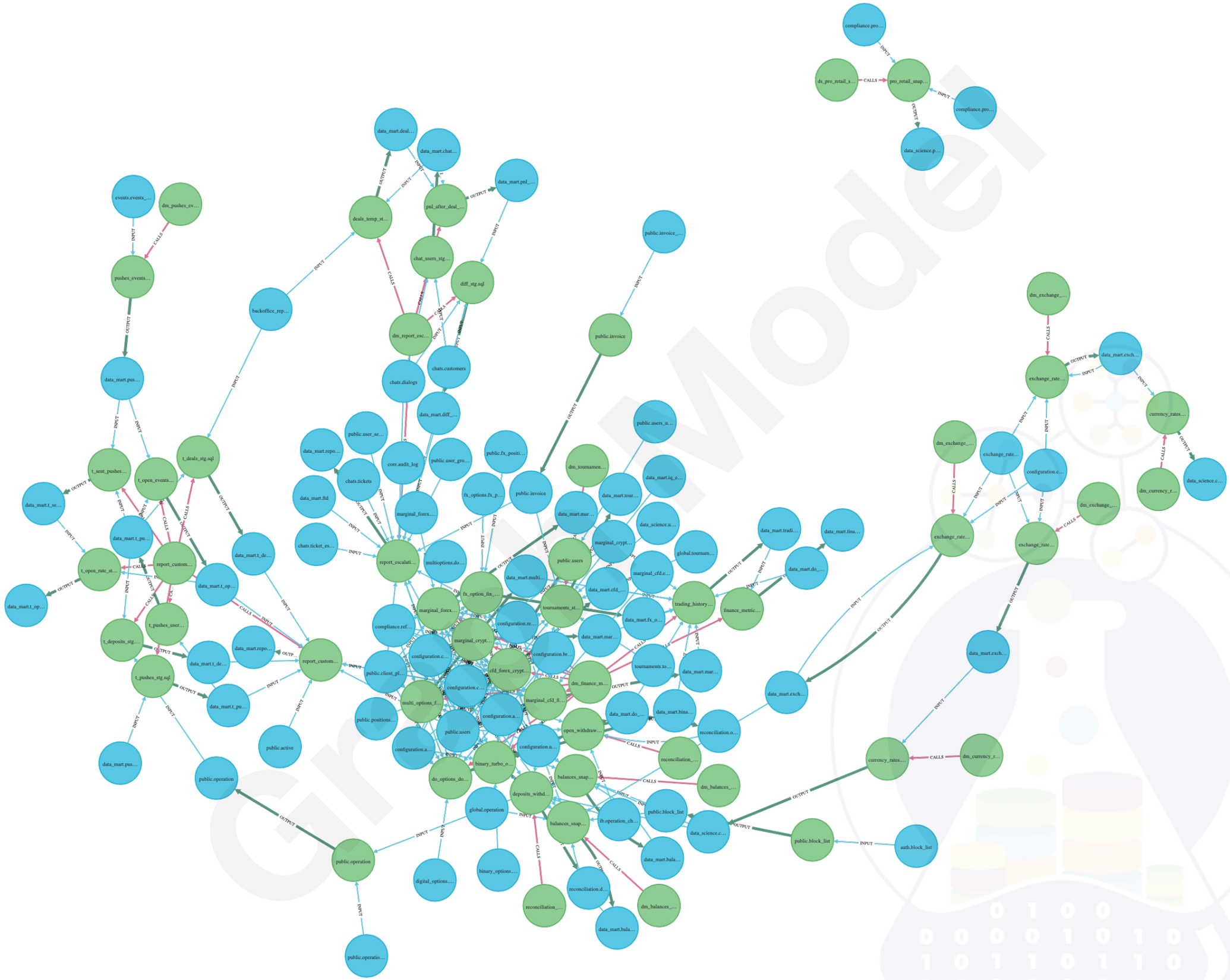
Hierarchical + Temporal Models











```
MATCH
  (a:entity),
  (b:job)
WHERE a.name = 'public.block_list' AND
          b.name = 'public.block_list'
CREATE (a)<-[r:OUTPUT]-(b)
RETURN type(r);
```

Relational model

id	node_1	node_2
100	Anna →	Germany
200	Munich →	Germany
300	BMW →	Munich
400	Ivan →	BMW
500	Ivan →	Anna
...	...	...


#1
#8
#12
EXHIBIT #6
#3
MAY 2016
24601
DPD
?
#10
RELATED ?
THE DARRINGTON POST
MISSING PERSON TOPS AUGUST NEWS
Alan Millan disappears
#4
EXHIBIT 2
#7
EXHIBIT 13

eXtensible Markup Language

```
<? XML VERSION = “1.0” ?>
<STAR-MOVIE-DATA>
  <STAR>
    <NAME>Carrie Fisher</NAME>
    <ADDRESS><CITY>Hollywood</CITY></ADDRESS>
    <ADDRESS><CITY>Malibu</CITY></ADDRESS>
  </STAR>
  <MOVIE>
    <TITLE>Star Wars</TITLE>
    <YEAR>2017</YEAR>
  </MOVIE>
 <MOVIE>
    <TITLE>Star Wars II</TITLE>
  </MOVIE>
</STAR-MOVIE-DATA>
```

Document Type Definition

```
< !DOCTYPE Stars [
    < !ELEMENT STARS (STAR*) >
    < !ELEMENT STAR (NAME, ADDRESS+,MOVIES) >
    < !ELEMENT NAME (#PCDATA) >
    < !ELEMENT ADDRESS (STREET) >
    < !ELEMENT STREET (#PCDATA) >
    < !ELEMENT CITY (#PCDATA) >
    < !ELEMENT MOVIES (MOVIE*) >
    < !ELEMENT MOVIE (TITLE, YEAR) >
    < !ELEMENT TITLE (#PCDATA) >
    < !ELEMENT YEAR (#PCDATA) >
] >
```

```
<? XML VERSION = “1.0” ?>
<! DOCTYPE Stars SYSTEM “star.dtd”>
<STAR-MOVIE-DATA>
 <STAR>
  <NAME>Carrie Fisher</NAME>
  <ADDRESS><CITY>Hollywood</CITY></ADDRESS>
  <ADDRESS><CITY>Malibu</CITY></ADDRESS>
 </STAR>
 <MOVIE>
  <TITLE>Star Wars</TITLE>
  <YEAR>2017</YEAR>
 </MOVIE>
 <MOVIE>
  <TITLE>Star Wars II</TITLE>
  <YEAR>1982</YEAR>
 </MOVIE>
</STAR-MOVIE-DATA>
```

XPath

#get all titles of movies
/STAR-MOVIE-DATA/MOVIE/TITLE

#get a first title of movies
/STAR-MOVIE-DATA/MOVIE[1]/TITLE

#get a list of title of movies with year is 2017
/STAR-MOVIE-DATA/MOVIE [YEAR = 2017] /TITLE

XQuery

```
#a list of title movies with year is 2017
for $x in doc("stars.xml")/STAR-MOVIE-DATA/MOVIE
where $x/YEAR=2017
order by $x/TITLE
return $x/TITLE
```

```
CLASS Account = { accountNo: integer;
                  balance: real;
                  owner: REF Customer;
                  deposit(a: Account, m: real);
                }


CLASS Customer = { name: string;
                   phone : string;
                 }
```

```
class Movie {
attribute string title;
attribute integer year;
attribute integer length;
attribute enum Film {color, BlackAndWhite} filmType;
};
```

(“Gone With the Wind”, 1939, 231, BlackAndWhite)

(“Star Wars. Last Jedi”, 2017, 152, color)

```
class Star {
attribute string name;
attribute Struct Addr {string street, string city} address;
relationship Set<Movie> starredIn;
};

class Movie {
attribute string title;
attribute integer year;
attribute integer length;
attribute enum Film {color, blackAndWhite} filmType;
relationship Set<Star> starsOf;
};
```

Star

property	type
name	string
address	address
Movie	**CLASS**

Movie

property	type
title	string
year	integer
length	integer
Star	**CLASS**

```
class Movie {
attribute string title;
attribute integer year;
attribute integer length;
attribute enum Film {color, BlackAndWhite} filmType;
relationship Set<Star> starsOf;
};
```

```
class Cartoon extends Movie {
relationship Set<Star> voices;
};
```

```
class Mystery extends Movie {
attribute boolean isScreaming;
};
```

```
class Cartoon extends Movie {
relationship Set<Star> voices;
};
```

```
class Mystery extends Movie {
attribute boolean isScreaming;
};
```

```
class CartoonMystery extends Mystery : Cartoon ;
```

```
class CarSharing {
    attribute class Car;
    attribute class Person;
    attribute string district;
}
```

```
class Car {
    attribute string model;
    attribute string regNumber;
    attribute string status;
}
```

```
class Person {
    attribute string name;
    attribute class Document;
    attribute string phone;
}
```

```
SELECT c.model, c.regNumber
  FROM c IN Cars
WHERE status = ‘working’
```

```
count( SELECT c.model
         FROM c IN Cars
        WHERE status = ‘working’)
```

COMMIT;