Assembling cohort queries with FunSQL

http://github.com/MechanicalRabbit/FunSQL.jl

Kyrylo Simonov

OHDSI DevCon 2023

FunSQL is...

- a library for constructing SQL queries
- released under open-source MIT license
- written in Julia, but can also be used with R

and

- designed for assembling analytical queries from reusable components

Introduction

Cohort definitions and SQL query builders

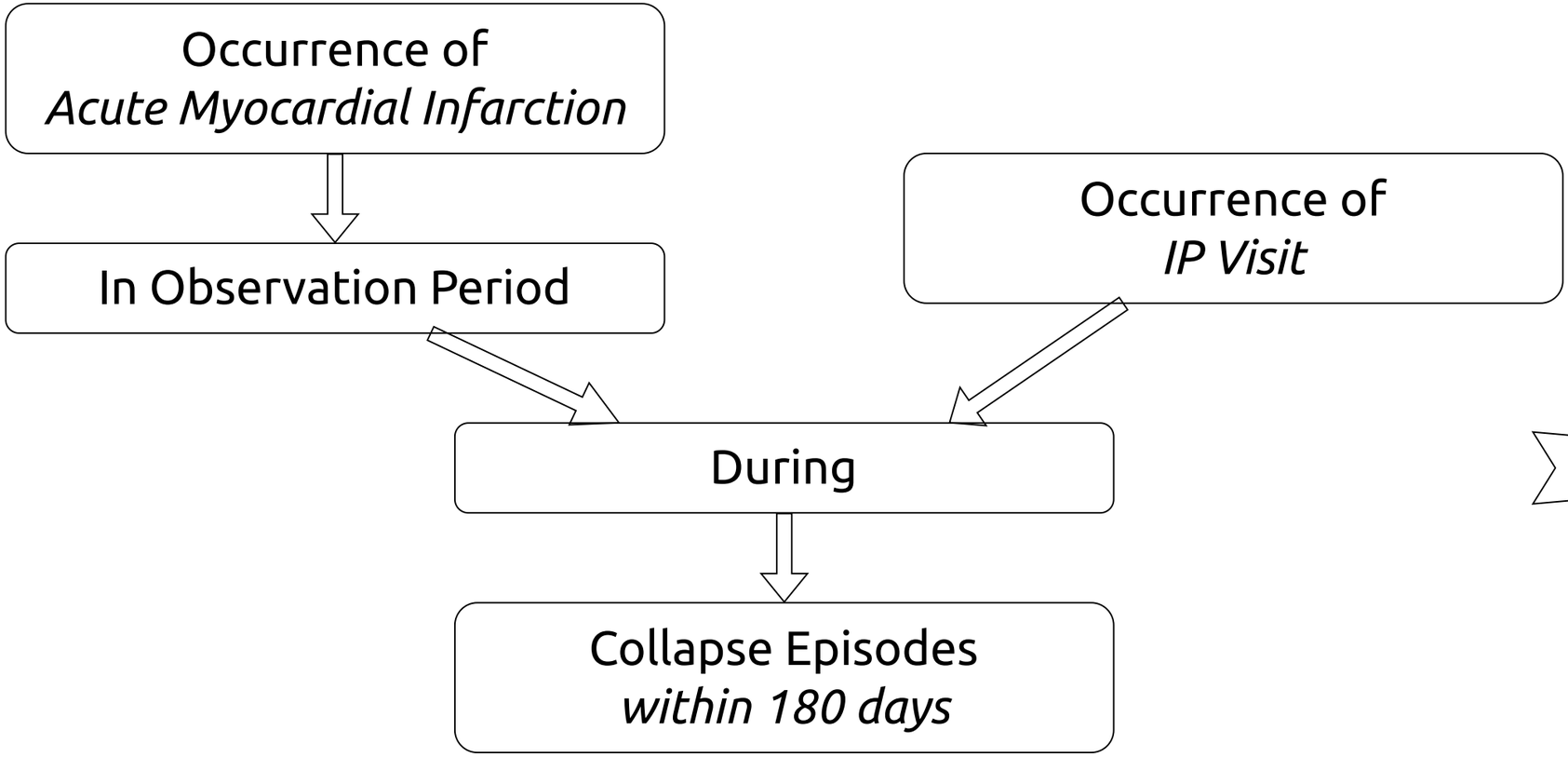
Examples

Reusable query components with joins, recursion, and aggregate and window functions

Conclusion

Build yourself a query language

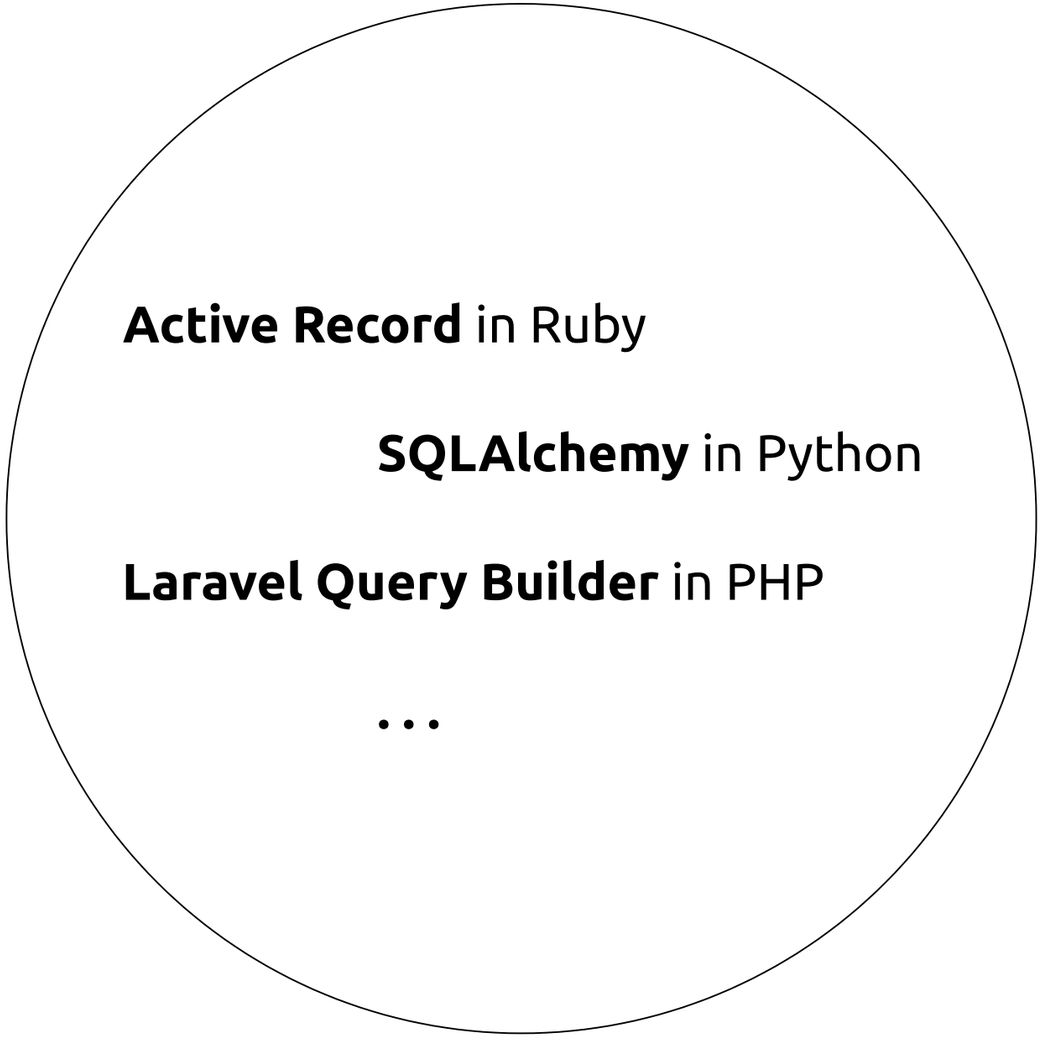
What makes translating **cohort definitions** difficult?



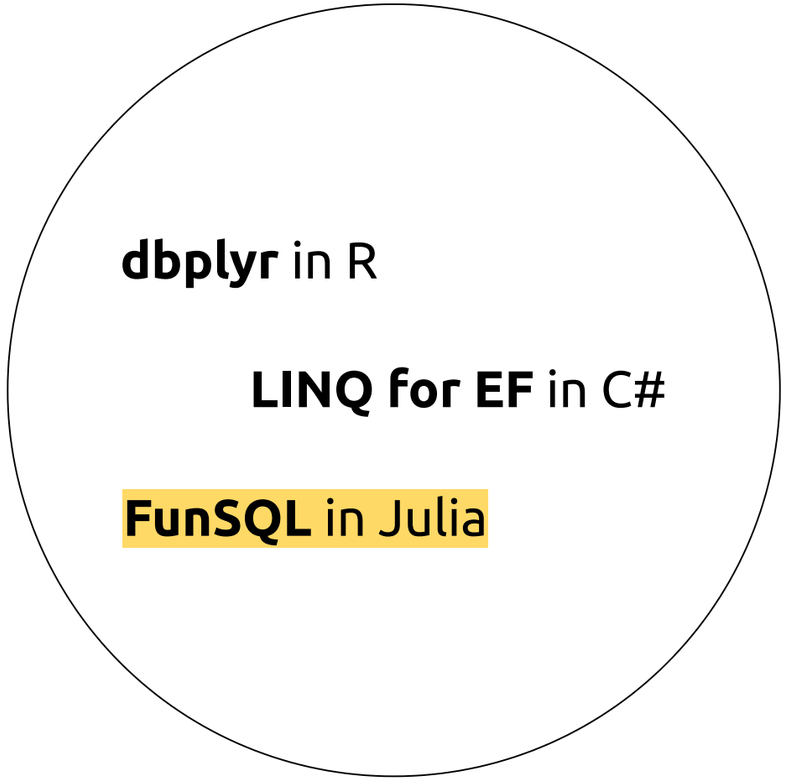

```
SELECT
  person_id,
  cohort_start_date,
  cohort_end_date
FROM
  ...
```

- Expressed in **multi-stage logic**
- Requires **advanced SQL**
- Translated **dynamically**

SQL query building libraries



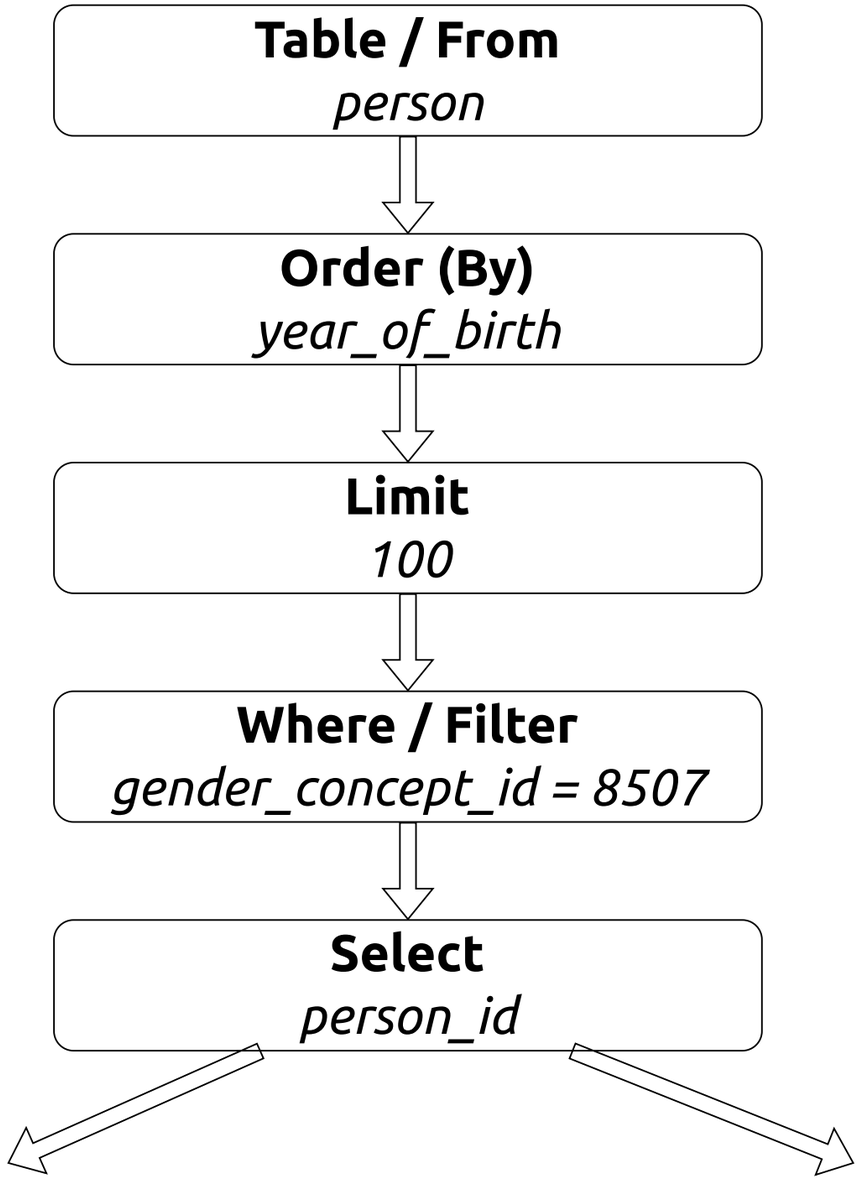
Assemble SQL **syntax** tree



Assemble **data** processing pipeline

Laravel Query Builder in PHP:

```php
DB::table('person')
->orderBy('year_of_birth')
->limit(100)
->where('gender_concept_id',
        '=', 8507)
->select('person_id')
```

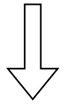


FunSQL in Julia:

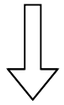
```julia
from(person)
order(year_of_birth)
limit(100)
filter(gender_concept_id
       == 8507)
select(person_id)
```

Laravel Query Builder in PHP:

```
DB::table('person')
->orderBy('year_of_birth')
->limit(100)
->where('gender_concept_id', '=', 8507)
->select('person_id')
```

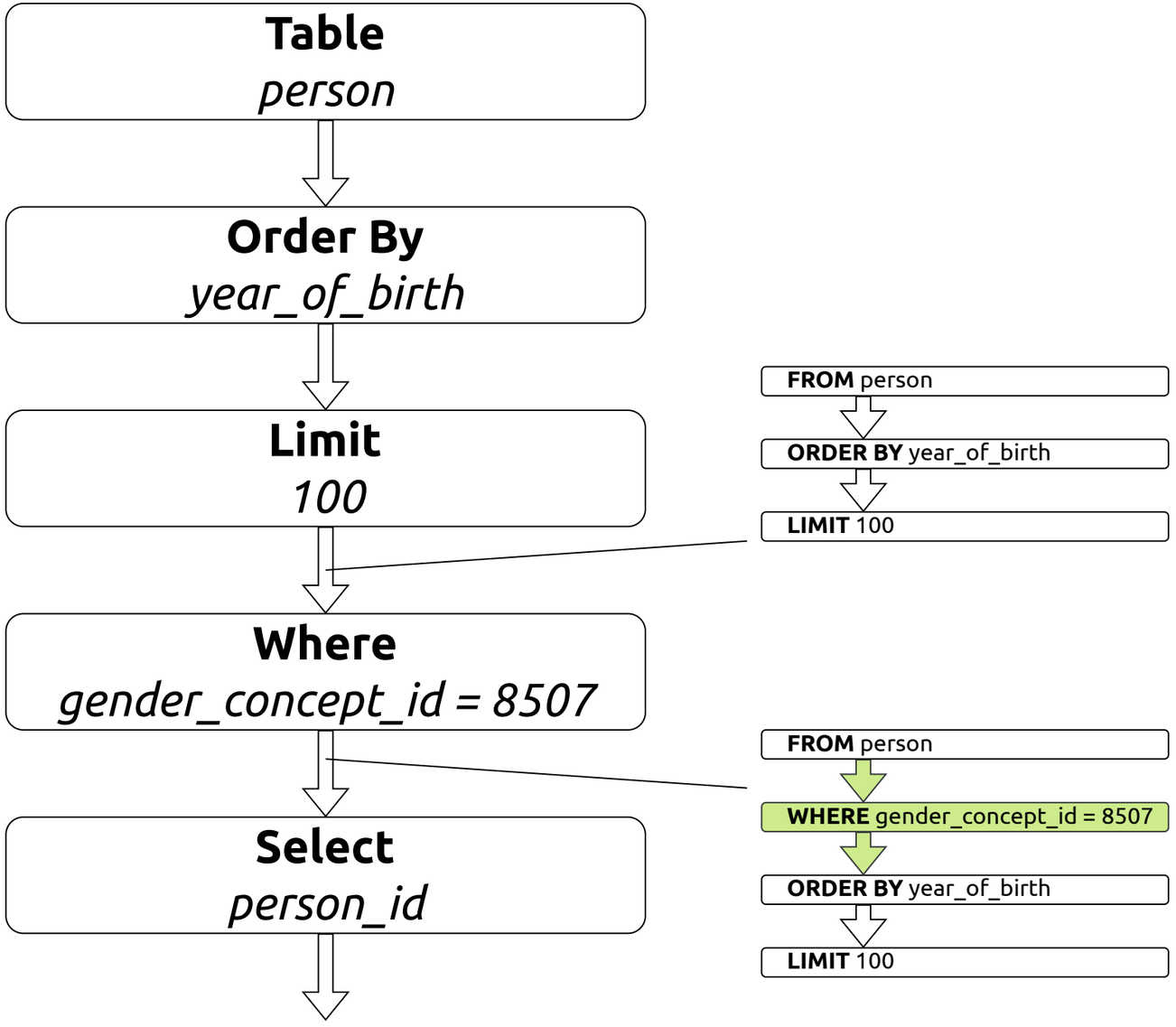


```
SELECT person_id
FROM person
WHERE gender_concept_id = 8507
ORDER BY year_of_birth
LIMIT 100
```

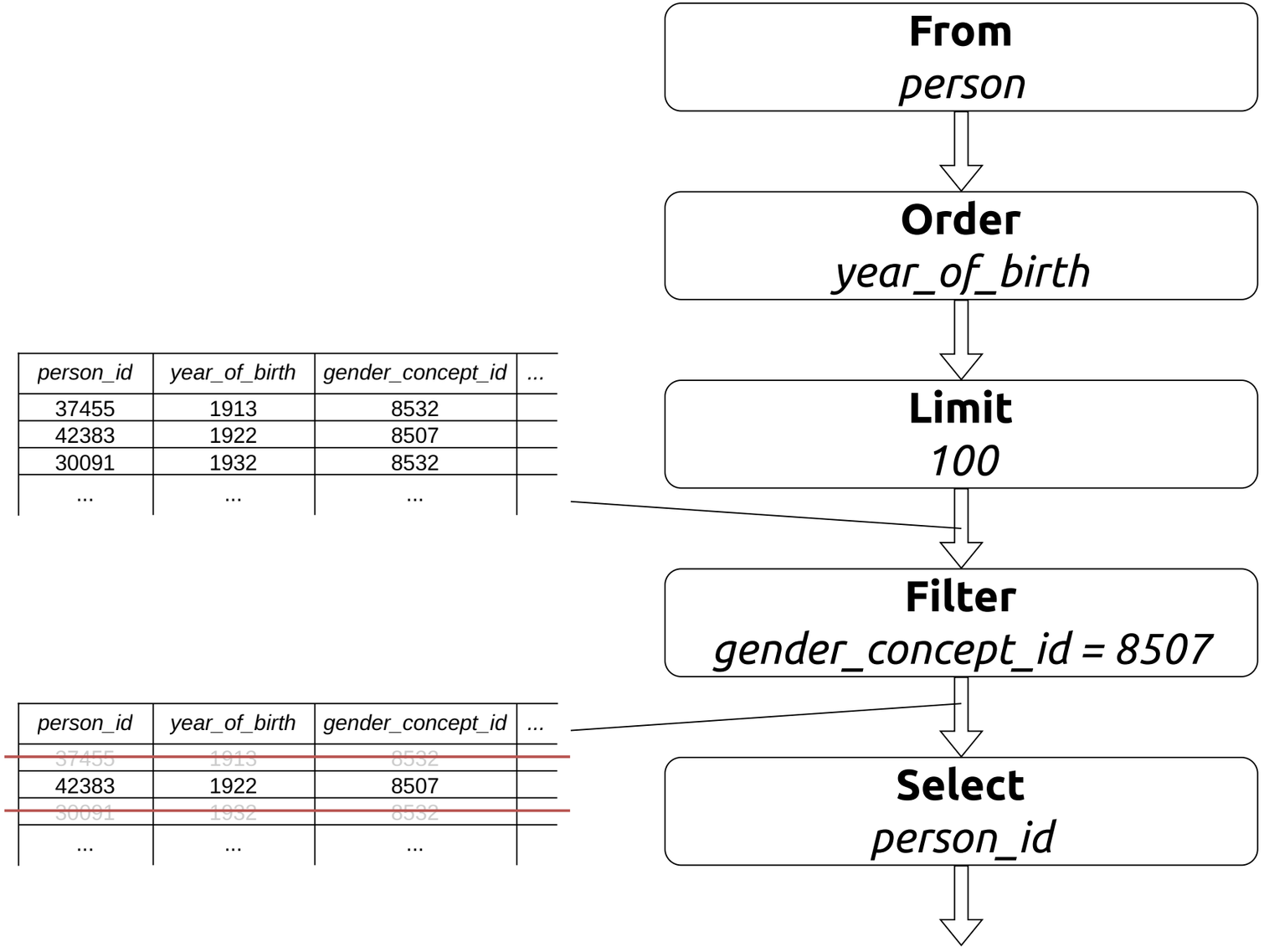


100 oldest male patients

Pipeline that builds SQL **syntax** tree:



Pipeline that processes **data**:



FunSQL in Julia:

```
from(person)
order(year_of_birth)
limit(100)
filter(gender_concept_id == 8507)
select(person_id)
```

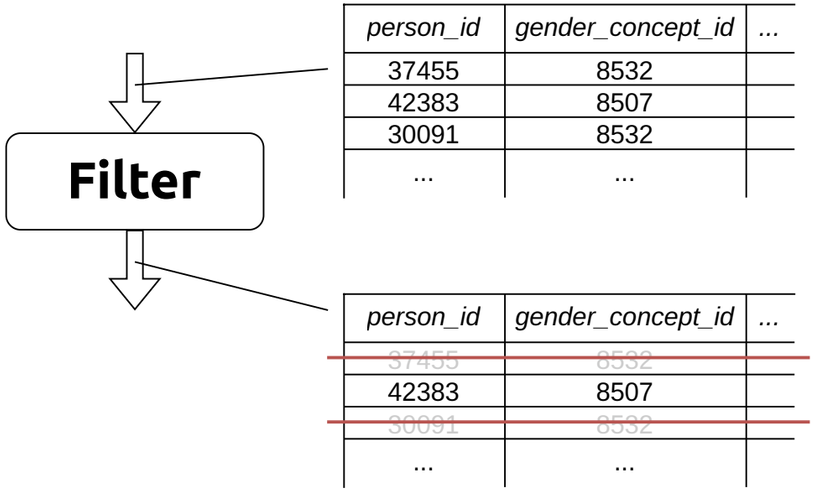
⇩

```
SELECT person_id
FROM (
  SELECT person_id,
         gender_concept_id
  FROM person
  ORDER BY year_of_birth
  LIMIT 100) AS person
WHERE gender_concept_id = 8507
```

⇩

Males among 100 oldest patients

In FunSQL, pipeline **components** are...



- *Coherent*

  Every component is interpreted as a data transformation

- *Composable*

  Any compatible components can be connected

- *Comprehensive*

  Components can represent aggregate and window functions, correlated subqueries, lateral joins, CTEs, recursive queries, and more

Introduction

Cohort definitions and SQL query builders

Examples

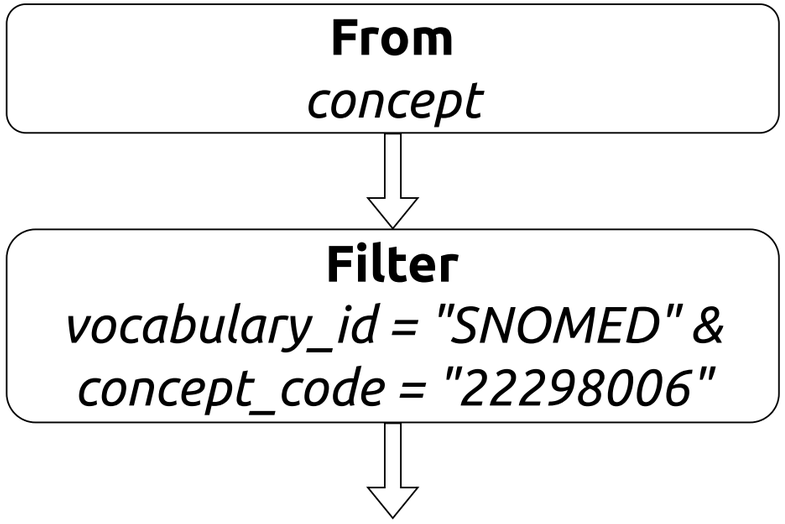
Reusable query components with joins, recursion, and aggregate and window functions

Conclusion

Build yourself a query language

SNOMED 22298006 "Myocardial infarction"

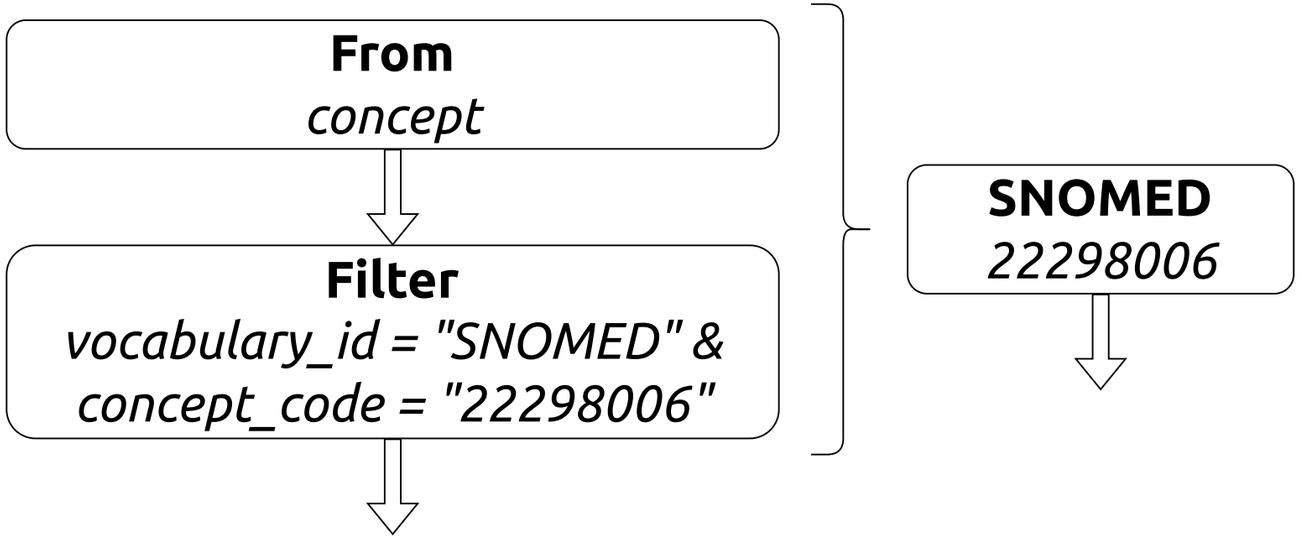
```
begin
  from(concept)
  filter(
    vocabulary_id == "SNOMED" &&
    concept_code == "22298006")
end
```

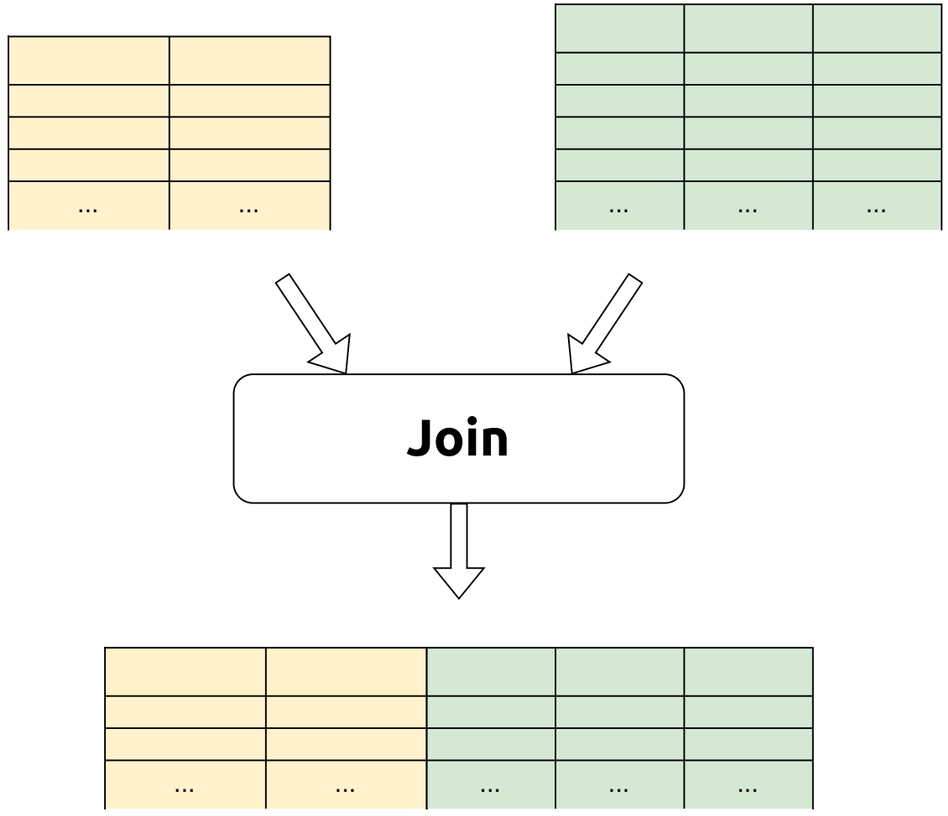


SNOMED 22298006 "Myocardial infarction"

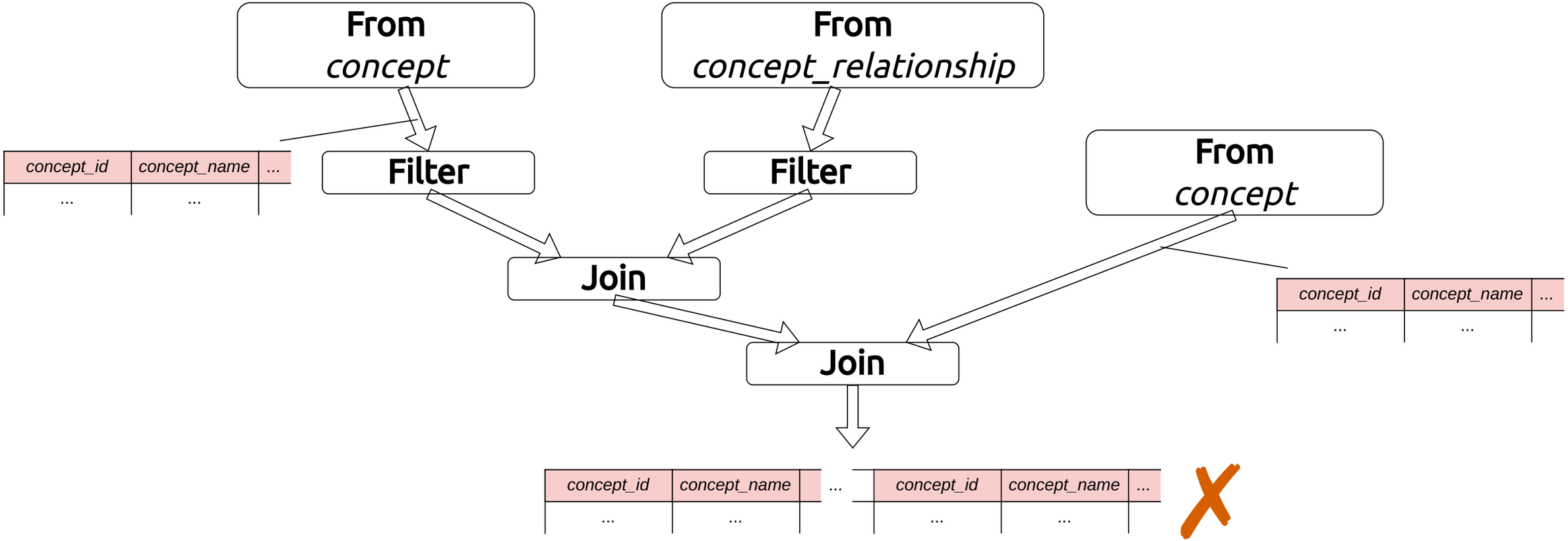
```
snomed("22298006")
```

```
snomed(code) = begin
  from(concept)
  filter(
    vocabulary_id == "SNOMED" &&
    concept_code == code)
end
```




...
...
...
...
...
Join
...
...
...
...
...

Find immediate children of the concept
SNOMED 22298006 "Myocardial infarction"

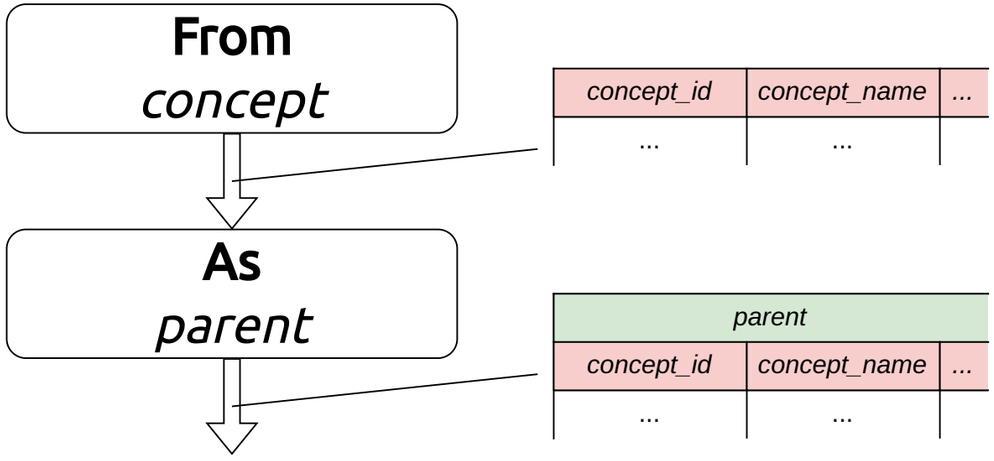


How to deal with **duplicate** column names?

SQL:
table **aliases**

dplyr:
mangled column names

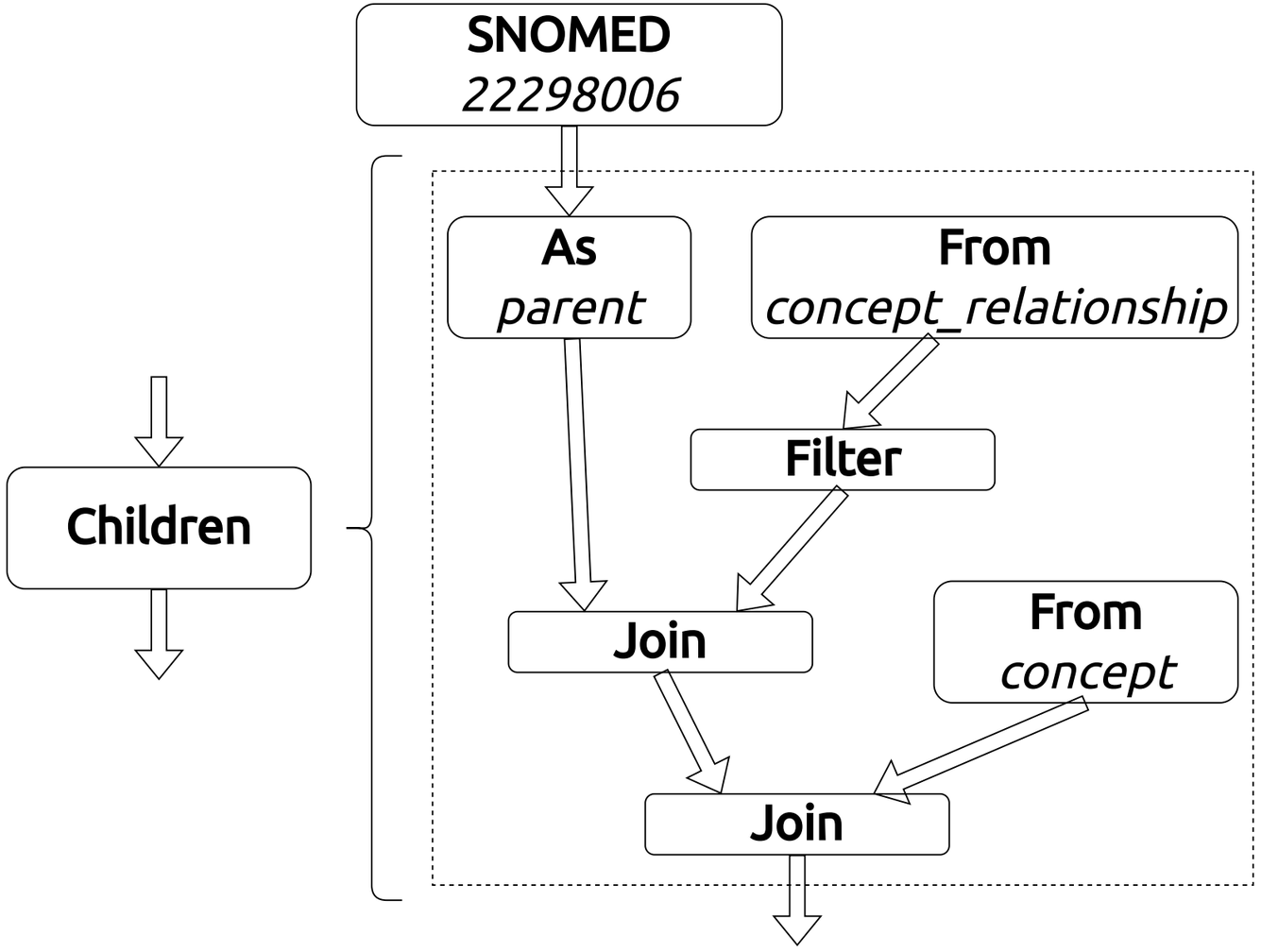
FunSQL:
nested records



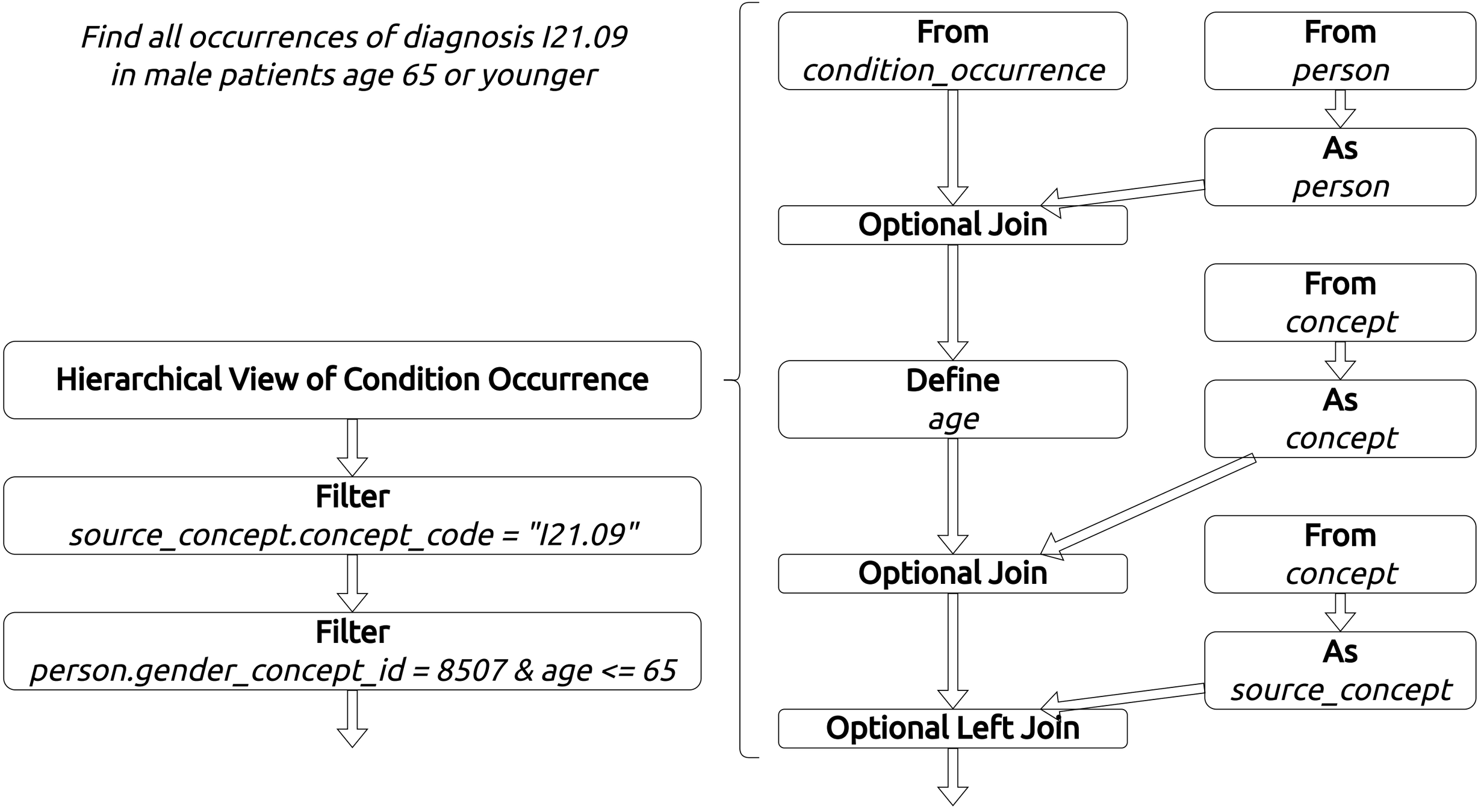
Find immediate children of the concept
SNOMED 22298006 "Myocardial infarction"

```
snomed("22298006")
children()
```

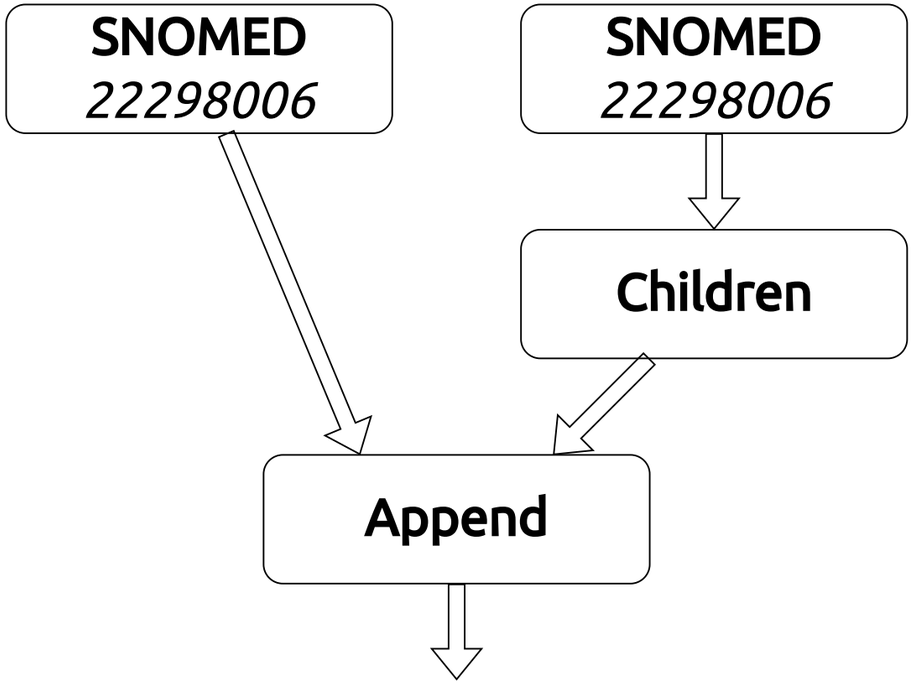
```
children() = begin
  as(parent)
  join(
    from(concept_relationship).
    filter(
      relationship_id == "Subsumes"),
    parent.concept_id == concept_id_1)
  join(
    from(concept),
    concept_id_2 == concept_id)
end
```

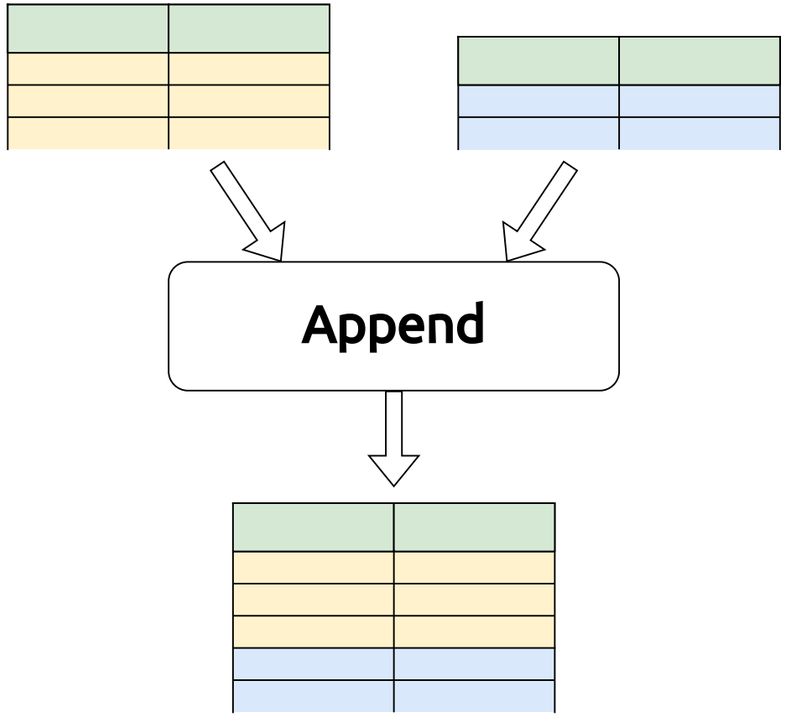


Find all occurrences of diagnosis I21.09
in male patients age 65 or younger



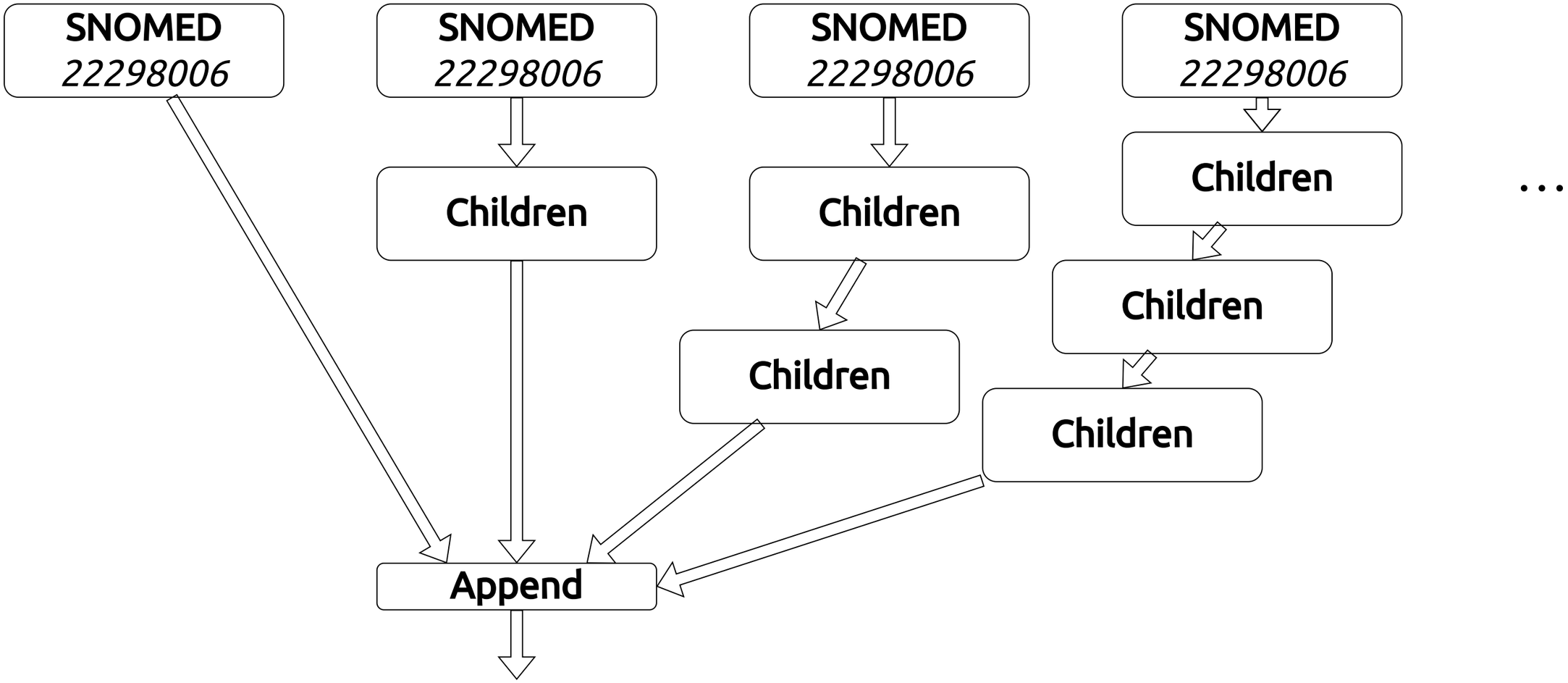
Concept SNOMED 22298006 "Myocardial infarction"
***and** its immediate children*






```
snomed("22298006")
append(snomed("22298006").children())
```

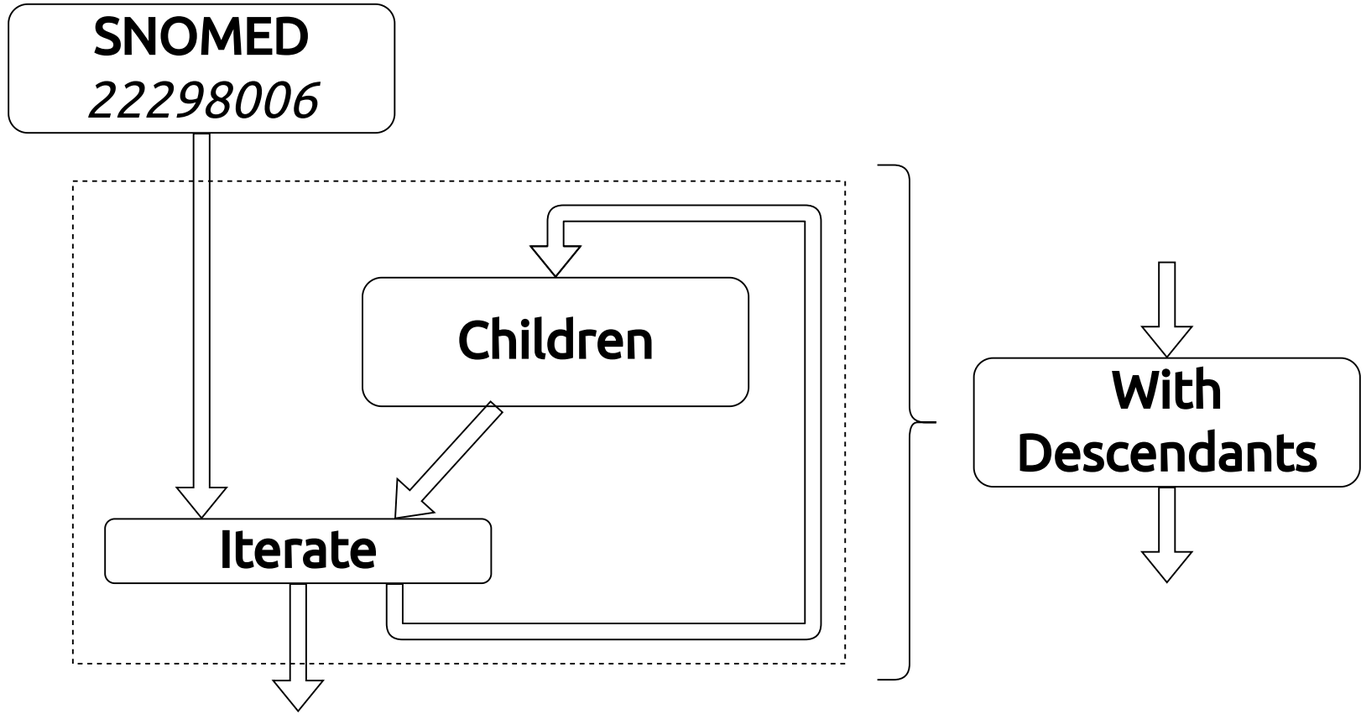
Concept SNOMED 22298006 "Myocardial infarction"
*and all its **descendants***



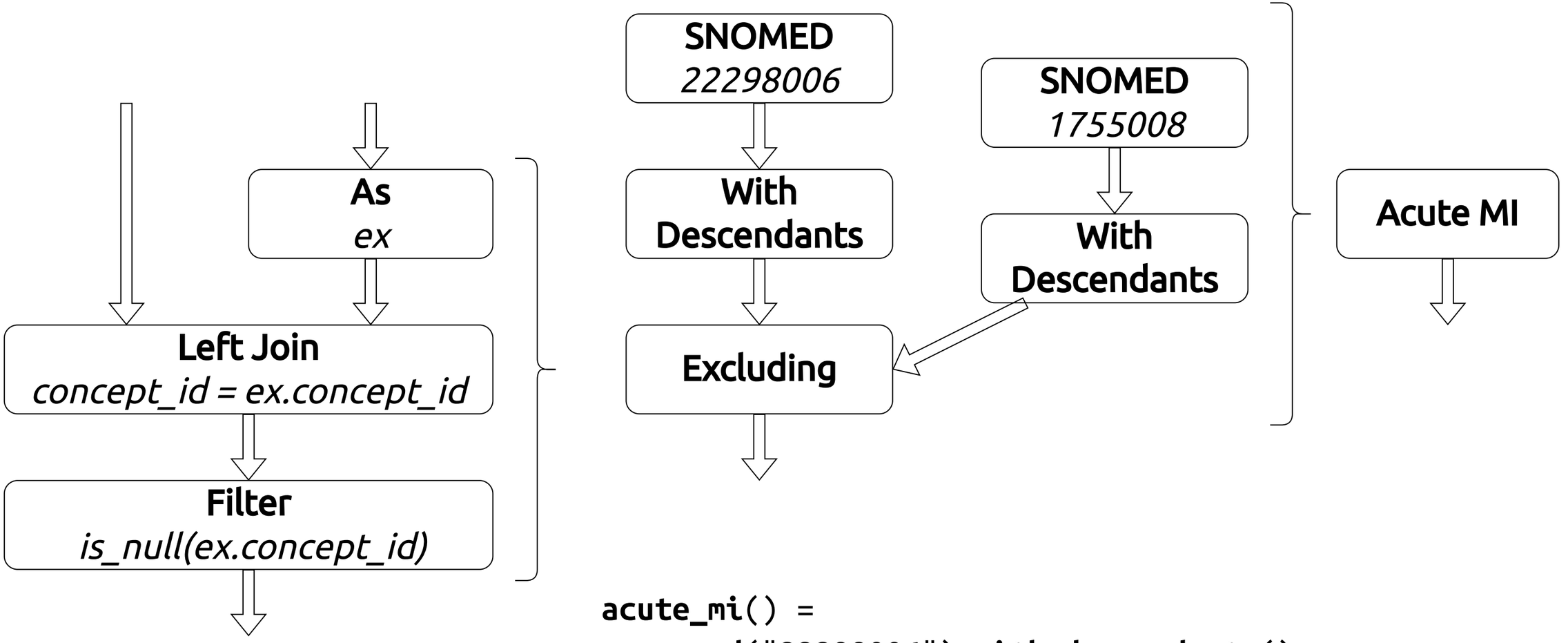
Concept SNOMED 22298006 "Myocardial infarction"
*and all its **descendants***

```
snomed("22298006")
with_descendants()
```

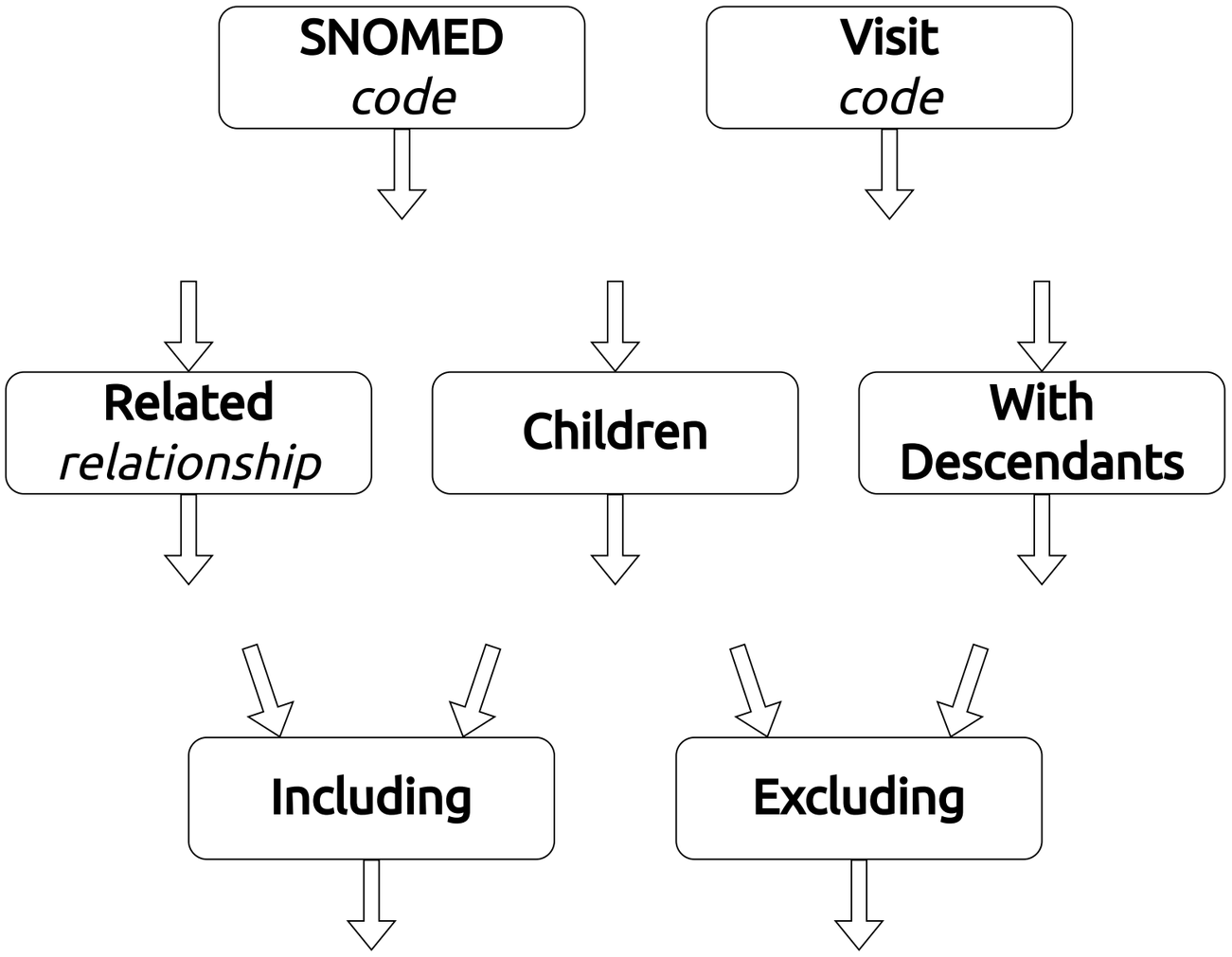
```
with_descendants() =
  iterate(children())
```



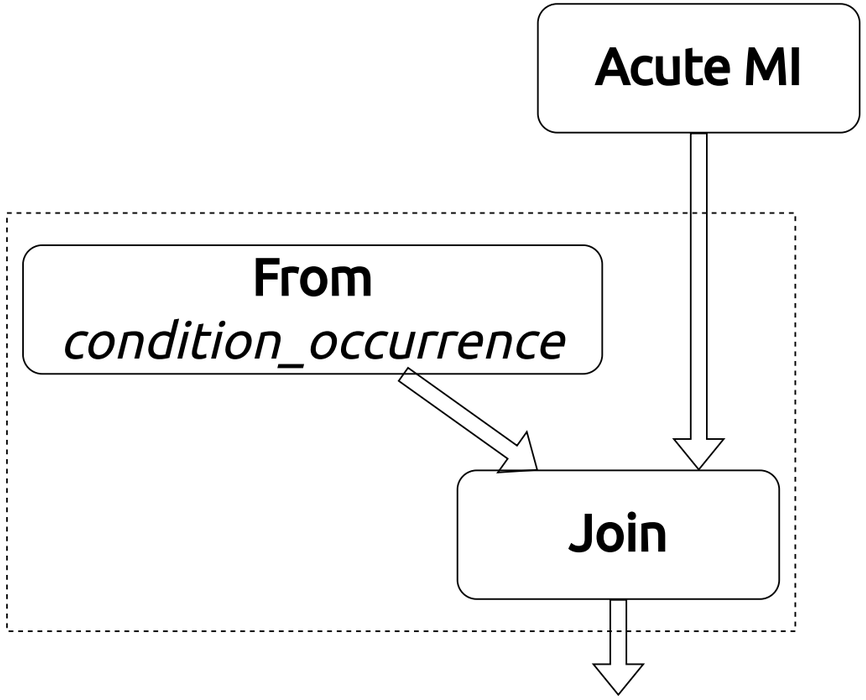
Acute MI is SNOMED 22298006 "Myocardial infarction" and all its descendants excluding SNOMED 1755008 "Old myocardial infarction" and all its descendants



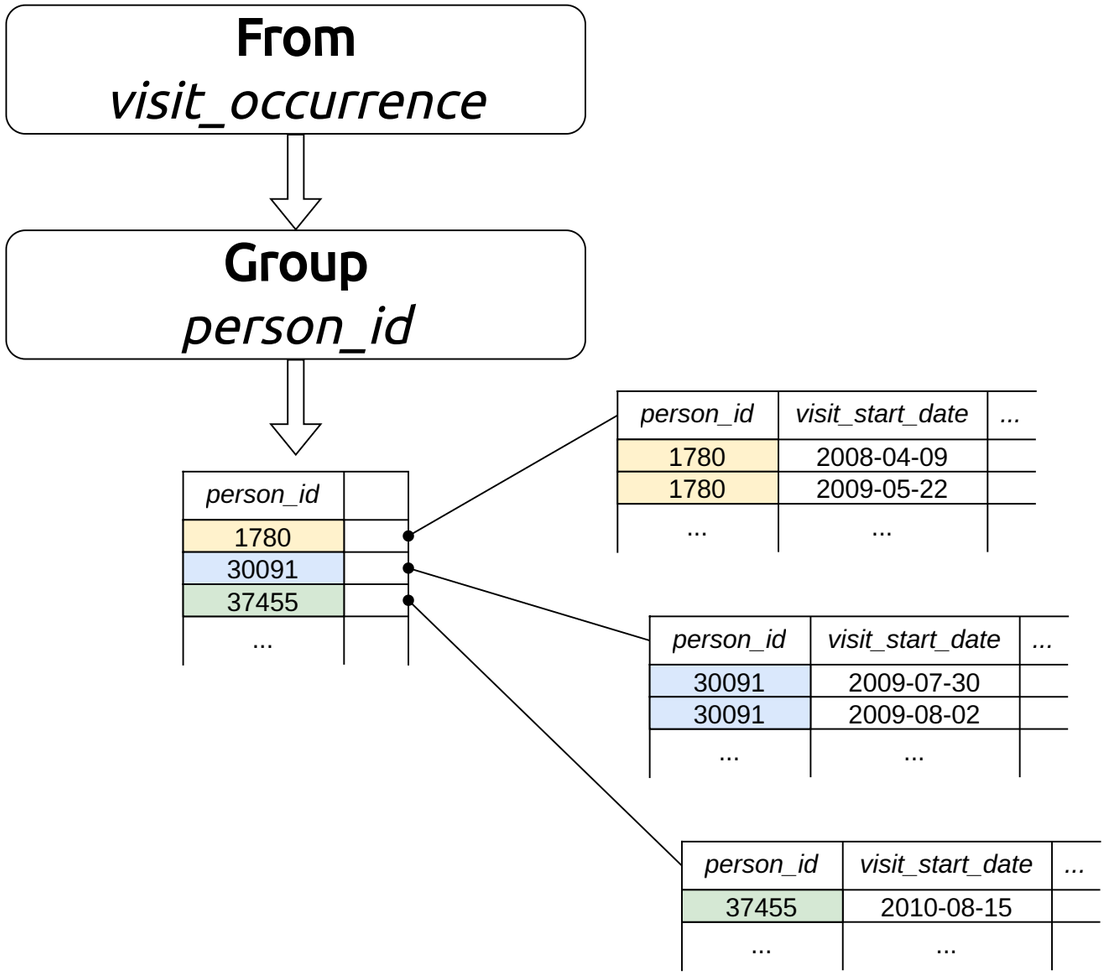

```
acute_mi() =
  snomed("22298006").with_descendants().
  excluding(snomed("1755008").with_descendants())
```

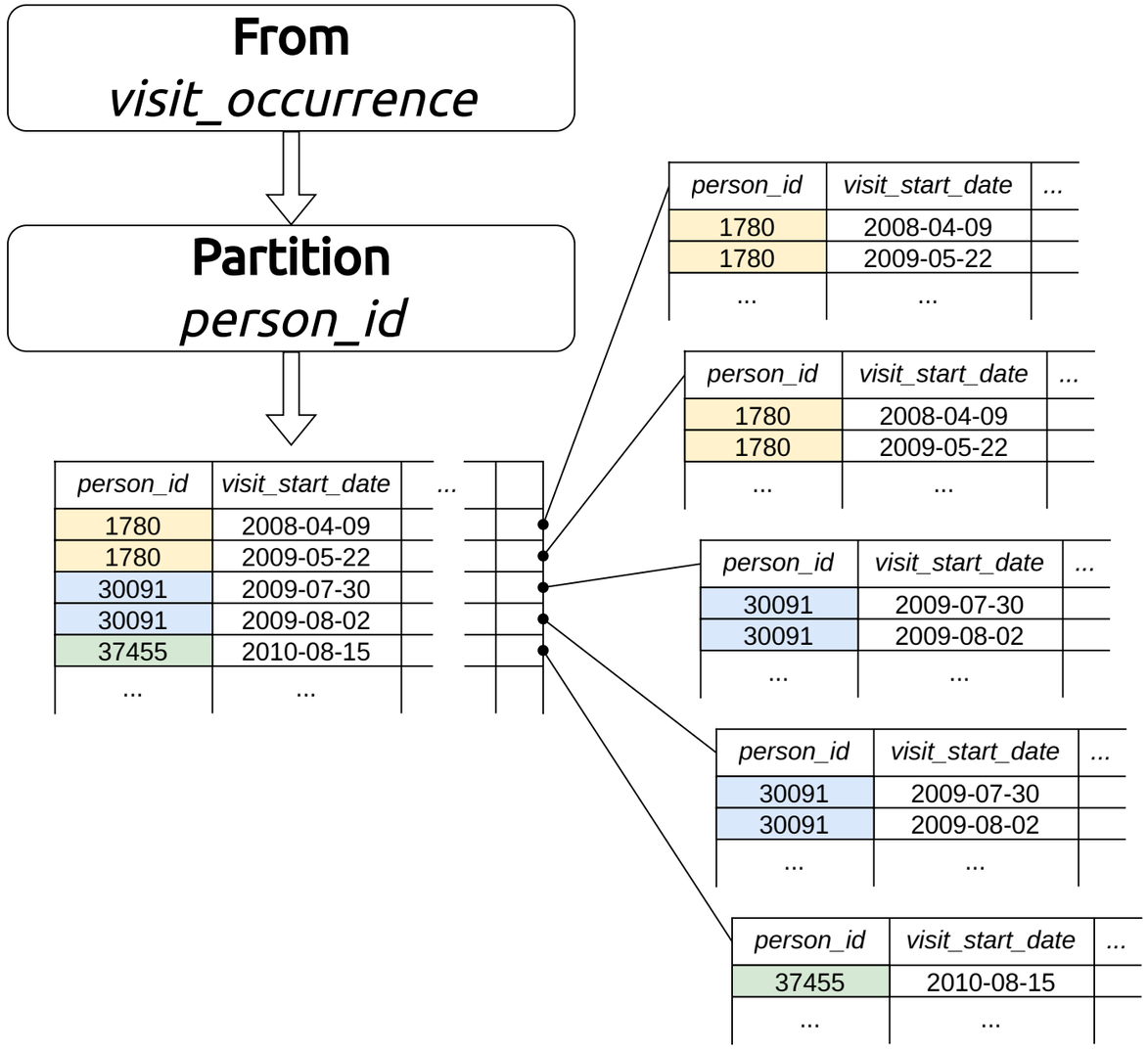


All occurrences of Acute MI




```
condition_occurrence(concept_set) = begin
  from(condition_occurrence)
  join(
    concept_set,
    condition_concept_id == concept_id)
end
```





`max[visit_start_date]`

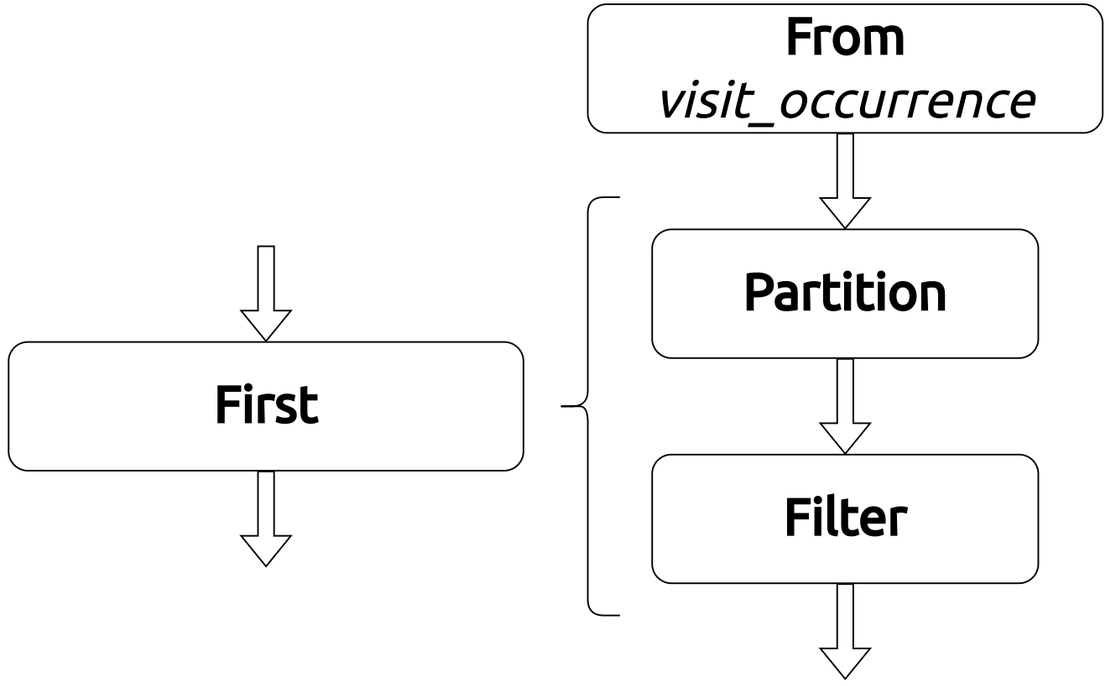
An aggregate function when applied to the output of the *universal aggregate function*

A window function when applied to the output of the *universal window function*

For each patient, find their latest visit

```
from(visit_occurrence)
first(visit_start_date.desc())
```

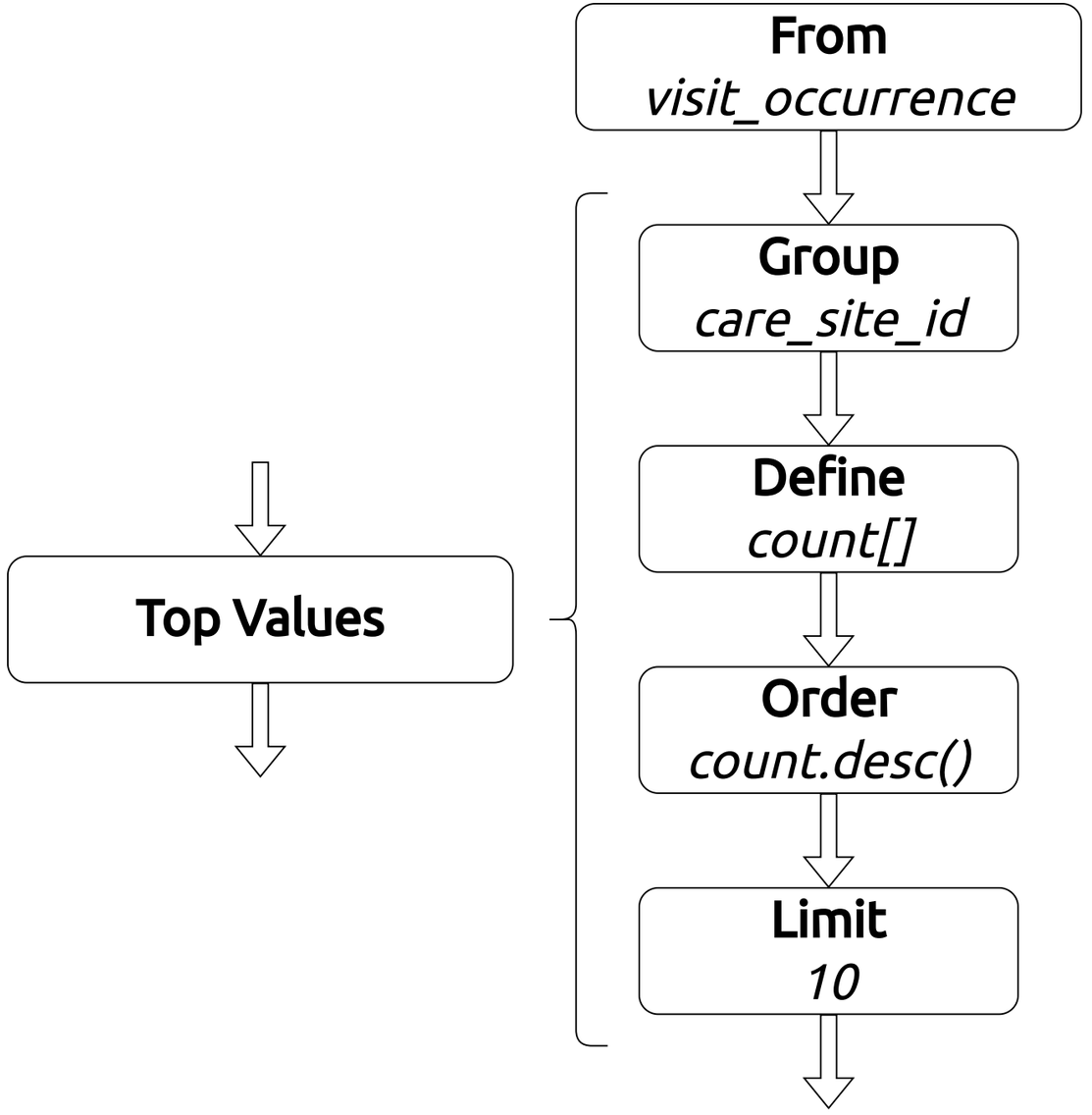
```
first(order_by...) = begin
  partition(
    person_id,
    order_by = [order_by...])
  filter(row_number[] == 1)
end
```



Find 10 most common care sites and the number of associated visits

```
from(visit_occurrence)
top_values(care_site_id)
```

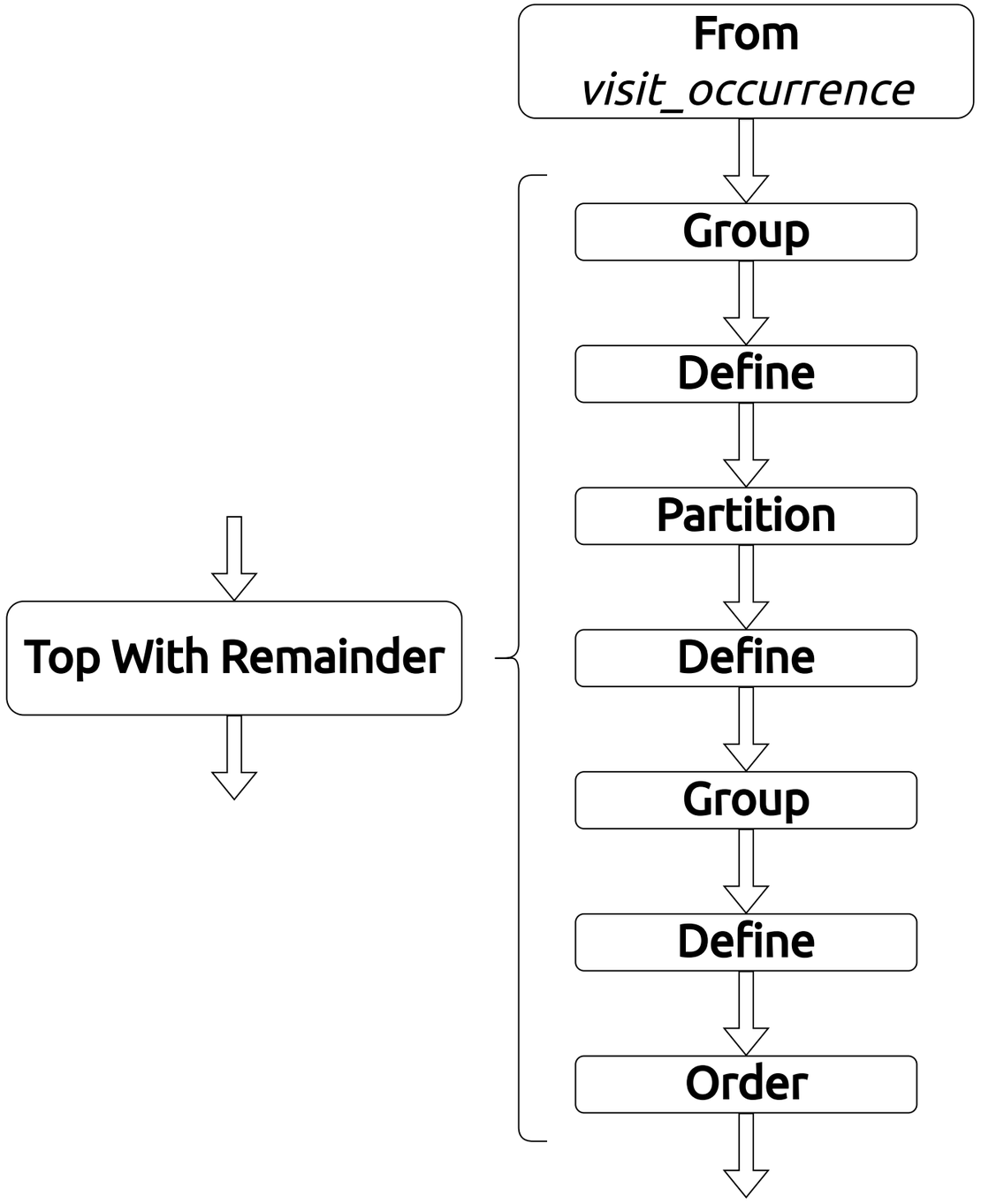
```
top_values(column, n = 10) = begin
  group(column)
  define(count => count[])
  order(count.desc())
  limit(n)
end
```



Find 10 most common care sites with their visit counts and the remainder

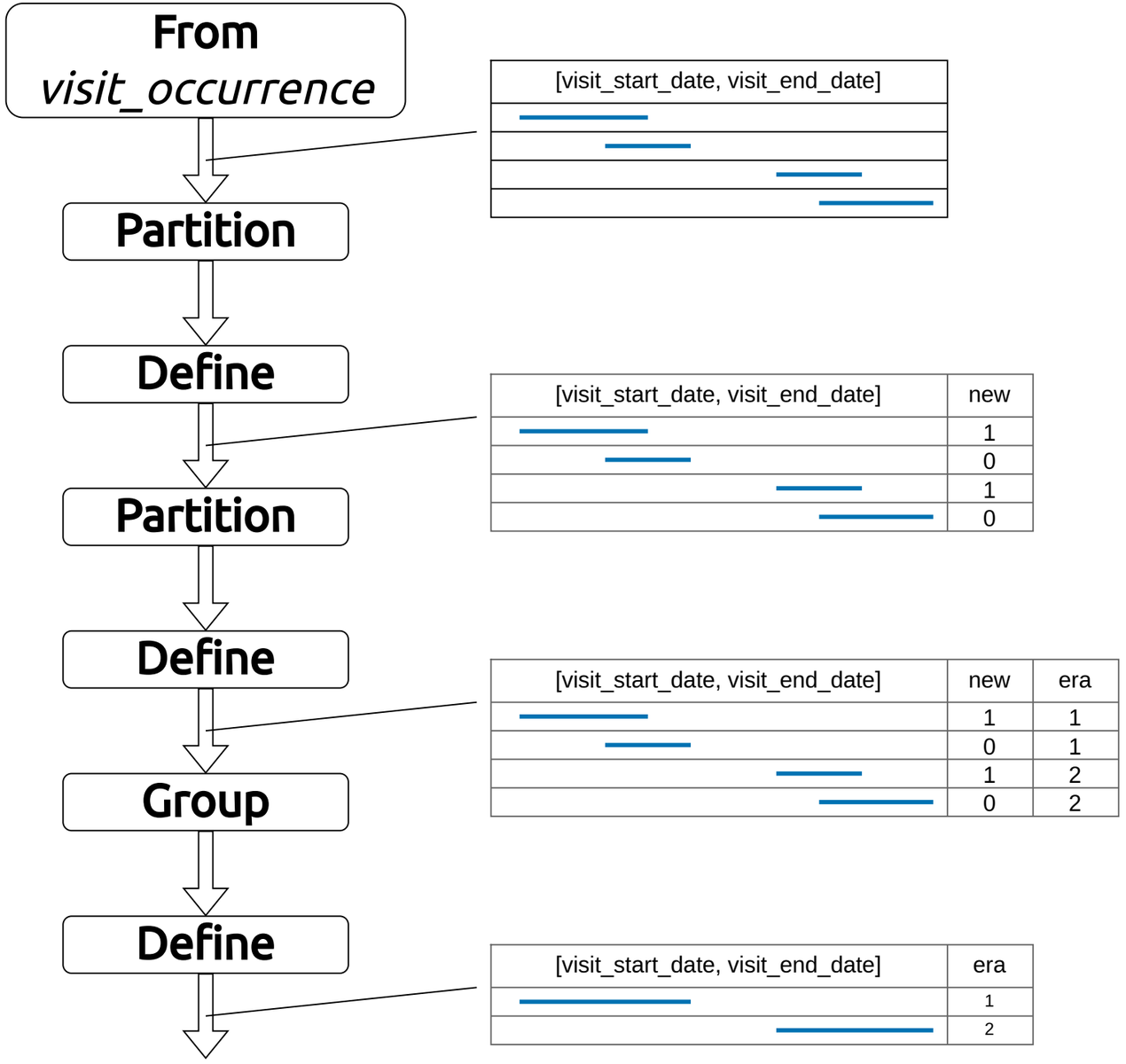
```
from(visit_occurrence)
top_with_remainder(care_site_id)
```

```
top_with_remainder(column, n = 10) = begin
  group(column)
  define(count => count[])
  partition(order_by = [count.desc()])
  define(remainder => row_number[] > n ? 1 : 0)
  group(
    column => remainder == 0 ? column : missing,
    remainder)
  define(count => sum[count])
  order(remainder, count.desc())
end
```

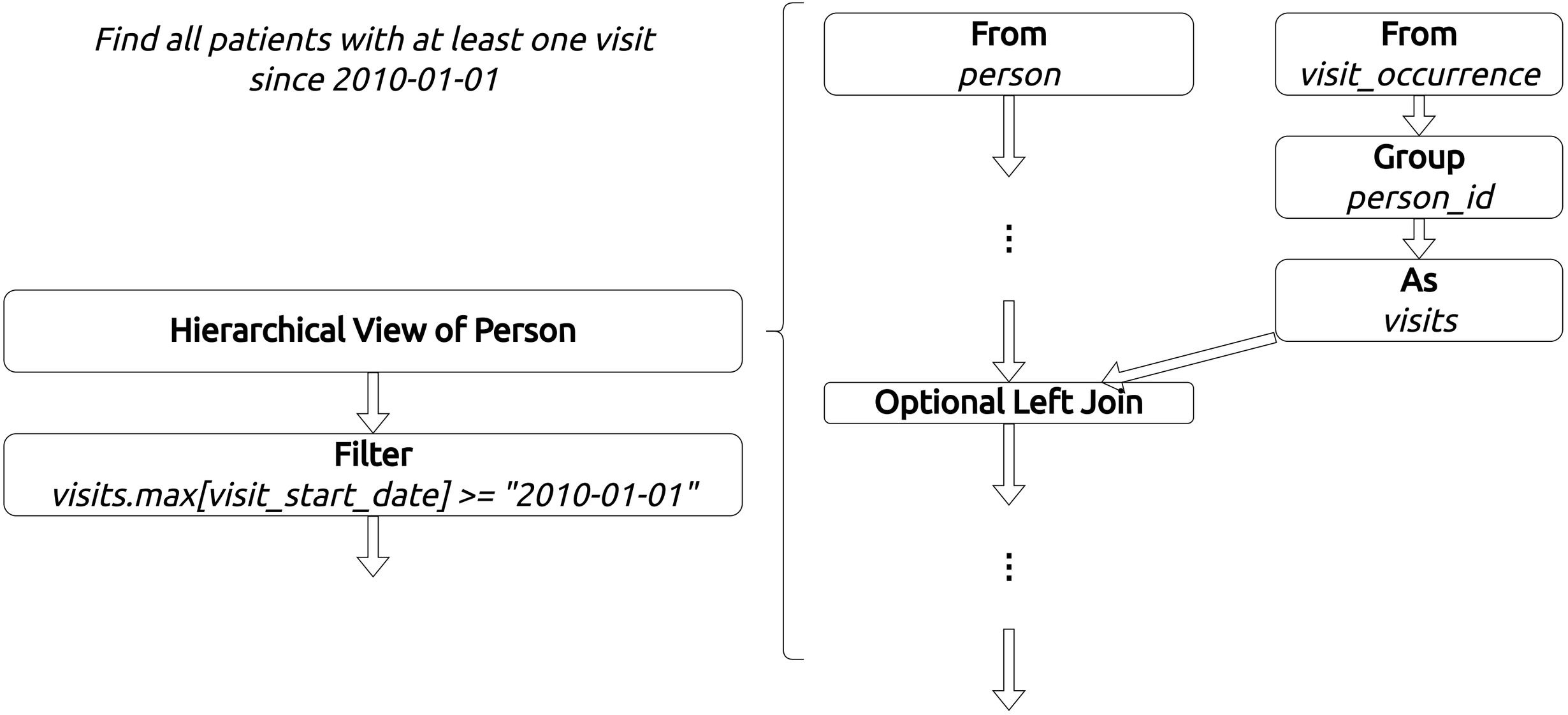


Merge overlapping visits

```
from(visit_occurrence)
partition(
  person_id,
  order_by = [visit_start_date],
  frame = (mode = rows,
           start = -Inf, finish = -1))
define(
  new =>
    visit_start_date <=
      max[visit_end_date] ? 0 : 1)
partition(
  person_id,
  order_by = [visit_start_date, -new],
  frame = (mode = rows))
define(era => sum[new])
group(person_id, era)
define(
  visit_start_date => min[visit_start_date],
  visit_end_date => max[visit_end_date])
```



Find all patients with at least one visit since 2010-01-01



Introduction

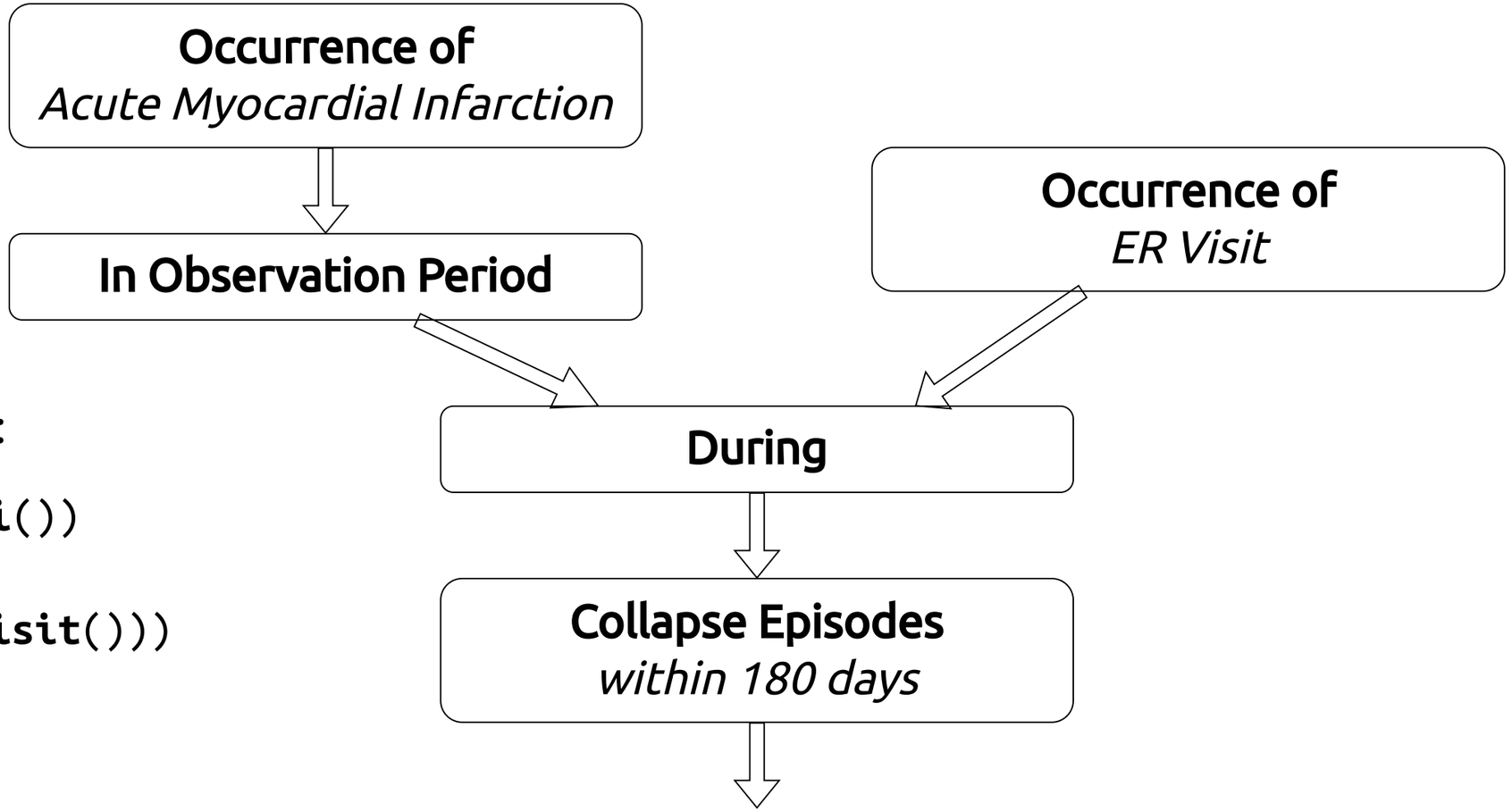
Cohort definitions and SQL query builders

Examples

Reusable query components with joins, recursion, and aggregate and window functions

Conclusion

Build yourself a query language



Build yourself a query **language**:

```
condition_occurrence(acute_mi())
in_observation_period()
during(visit_occurrence(ip_visit()))
collapse_episodes(180)
```