FunSQL

A library for compositional construction of SQL queries

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

Clark C. Evans,
Kyrylo Simonov

JuliaCon 2021

FunSQL? Who Needs It?

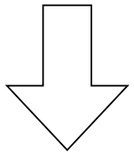
Query Algebra

Correlated Queries

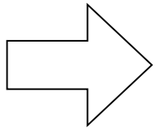
Aggregate & Window Functions

Conclusion

Find all patients born in or after 1970.



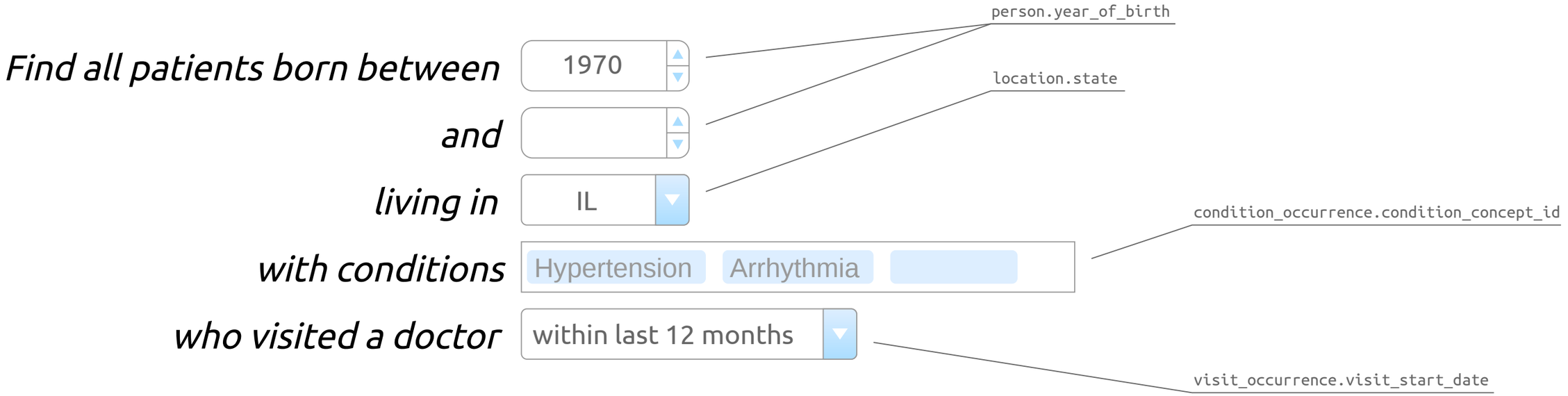
```
SELECT p.person_id
FROM person p
WHERE p.year_of_birth >= 1970
```



```
function find_patients(conn)
    sql = """
    SELECT p.person_id
    FROM person p
    WHERE p.year_of_birth >= 1970
    """
    DBInterface.execute(conn, sql)
end
```

Find all patients born between 1970 *and*

```
function find_patients(conn; start_year = nothing, end_year = nothing)
    sql = """
    SELECT p.person_id
    FROM person p
    """
    predicates = String[]
    if start_year !== nothing
        push!(predicates, "p.year_of_birth >= $start_year")
    end
    if end_year !== nothing
        push!(predicates, "p.year_of_birth <= $end_year")
    end
    if !isempty(predicates)
        sql *=  "\nWHERE " * join(predicates, " AND ")
    end
    DBInterface.execute(conn, sql)
end
```




```
function find_patients(conn; start_year = nothing,
                             end_year = nothing,
                             state = nothing,
                             condition_concepts = [],
                             latest_visit_threshold = nothing)

    sql = ???

    DBInterface.execute(conn, sql)
end
```

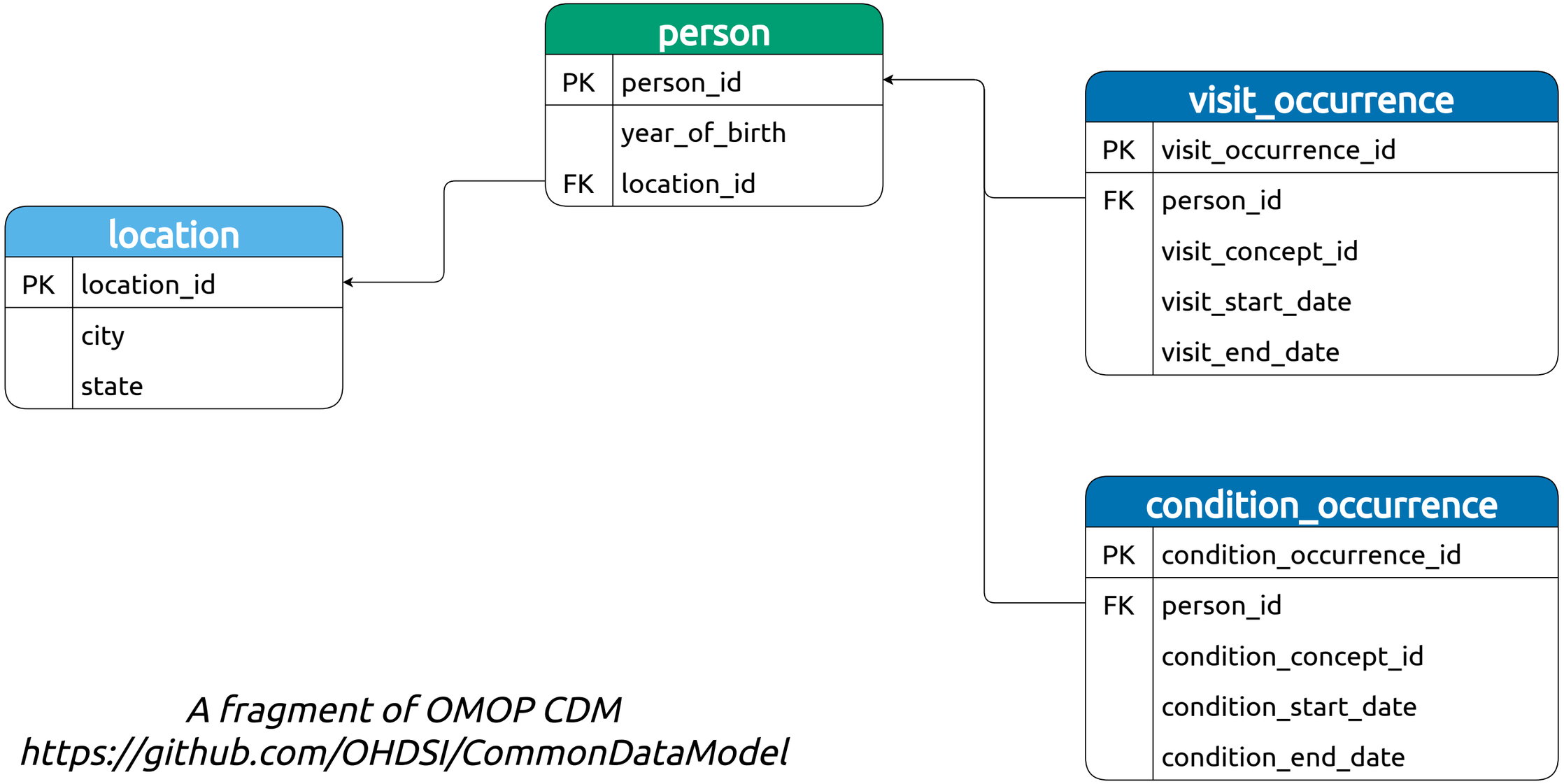
FunSQL? Who Needs It?

Query Algebra

Correlated Queries

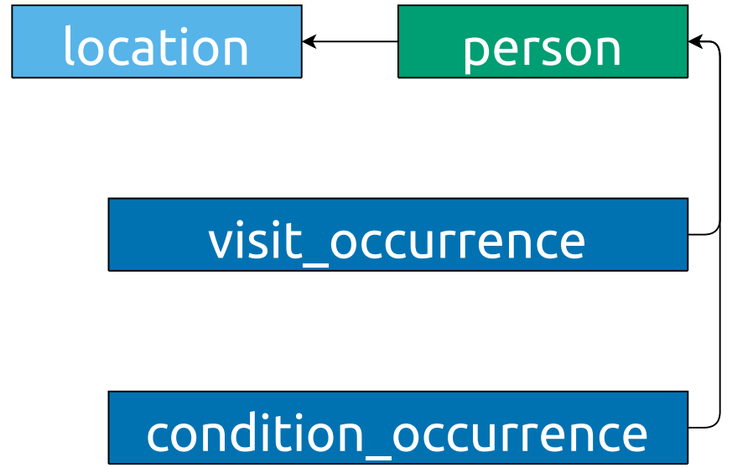
Aggregate & Window Functions

Conclusion



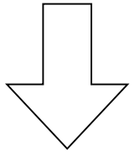
A fragment of OMOP CDM
https://github.com/OHDSI/CommonDataModel

```
using FunSQL: SQLTable

const person =
    SQLTable(name = :person,
             columns = [:person_id, :year_of_birth, :location_id])

const location =
    SQLTable(name = :location,
             columns = [:location_id, :city, :state, :zip])

const visit_occurrence =
    SQLTable(name = :visit_occurrence,
             columns = [:visit_occurrence_id, :person_id, :visit_concept_id,
                        :visit_start_date, :visit_end_date])

const condition_occurrence =
    SQLTable(name = :condition_occurrence,
             columns = [:condition_occurrence_id, :person_id, :condition_concept_id,
                        :condition_start_date, :condition_end_date])
```

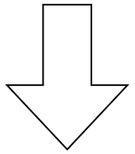


Find all patients born in or after 1970.

```
FROM person p
```



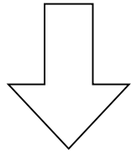
```
FROM person p
WHERE p.year_of_birth >= 1970
```



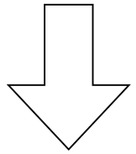
```
SELECT p.person_id
FROM person p
WHERE p.year_of_birth >= 1970
```

```
using FunSQL: From, Get, Select, Where, render
```

```
q = From(person)
```



```
q = From(person) |>
    Where(Get.year_of_birth .>= 1970)
```

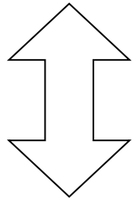


```
q = From(person) |>
    Where(Get.year_of_birth .>= 1970) |>
    Select(Get.person_id)
```

```
sql = render(q, dialect = :postgresql)
```

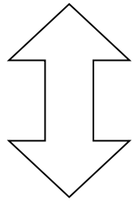
```
q₁ = From(person)
q₂ = q₁ |> Where(q₁.year_of_birth .>= 1970)
q  = q₂ |> Select(q₂.person_id)
```

bound references



```
q = From(person) |>
    Where(Get.year_of_birth .>= 1970) |>
    Select(Get.person_id)
```

unbound references



```
BornInOrAfter(Y) = Get.year_of_birth .>= Y

q = From(person) |>
    Where(BornInOrAfter(1970)) |>
    Select(Get.person_id)
```

person	
PK	person_id
	year_of_birth
FK	location_id

```
SELECT p.person_id
FROM person p
WHERE p.year_of_birth >= 1970
```

```
using FunSQL: Agg, Fun
```

"Fun" notation

```
Fun.">="(Get.year_of_birth, 1970)
```

or

```
Get.year_of_birth .>= 1970
```

broadcasting

```
WHERE p.year_of_birth >= 1970 AND
      p.year_of_birth <= 2000
```

```
Fun.and(Get.year_of_birth .>= 1970,
        Get.year_of_birth .<= 2000)
```

```
WHERE p.year_of_birth
      BETWEEN 1970 AND 2000
```

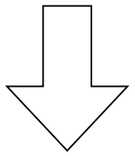
```
Fun.between(Get.year_of_birth, 1970, 2000)
```

```
SELECT AVG(p.year_of_birth)
FROM person p
```

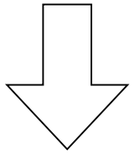
```
Agg.avg(Get.year_of_birth)
```

Show patients with their state of residence.

```
FROM person p
```



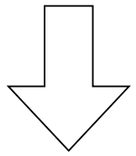
```
FROM person p
JOIN location l
  ON (p.location_id = l.location_id)
```



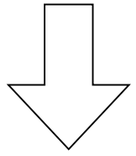
```
SELECT p.person_id, l.state
FROM person p
JOIN location l
  ON (p.location_id = l.location_id)
```

```
using FunSQL: Join
```

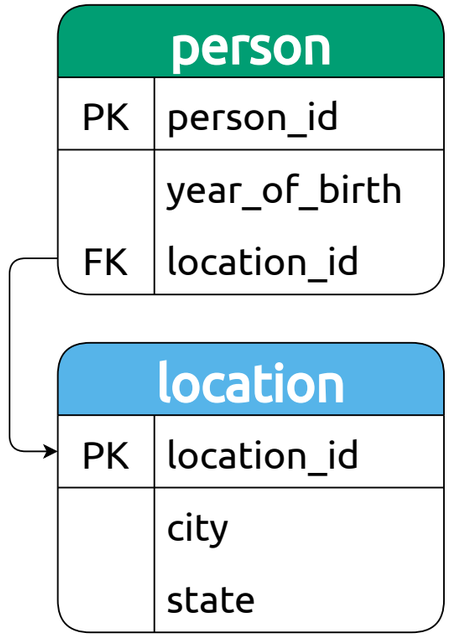
```
q = From(person)
```



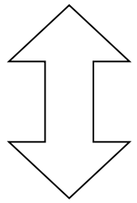
```
q = From(person) |>
    Join(:location => location,
         Get.location_id .==
         Get.location.location_id)
```

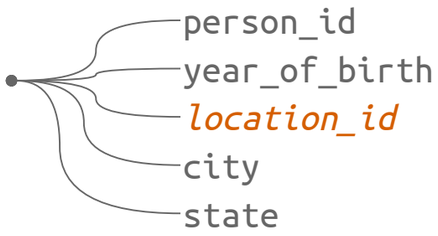


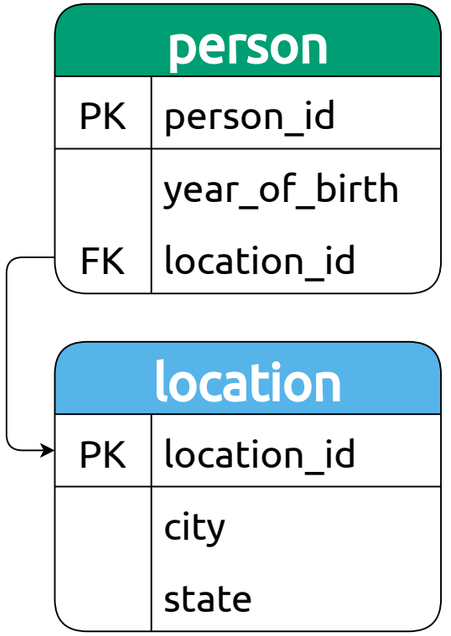
```
q = From(person) |>
    Join(:location => location,
         Get.location_id .==
         Get.location.location_id) |>
    Select(Get.person_id, Get.location.state)
```



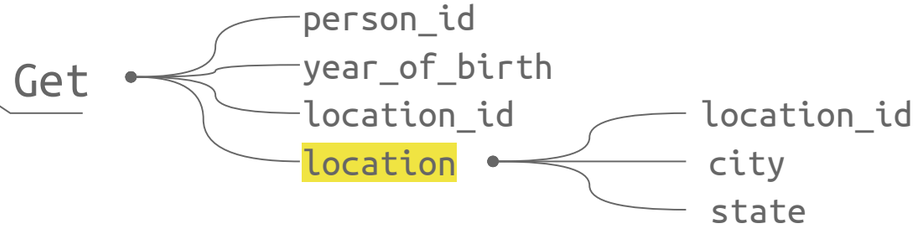
```
q_p = From(person)
q_l = From(location)
q   = q_p |> Join(q_l, q_p.location_id .== q_l.location_id)
```





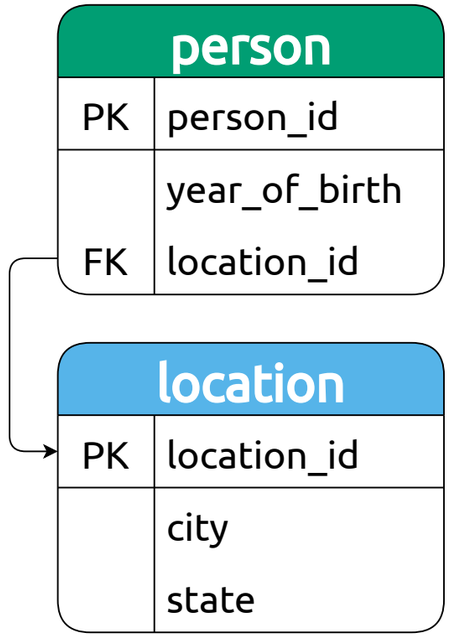



```
q = From(person) |>
    Join(:location => From(location),
         Get.location_id .== Get.location.location_id)
```



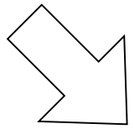
Find patients

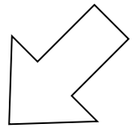
- *born in or after 1970*
- *living in Illinois*




```
q_p = From(person) |>
      Where(Get.year_of_birth .>= 1970)
```

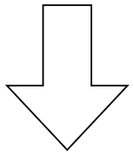
```
q_l = From(location) |>
      Where(Get.state .== "IL")
```



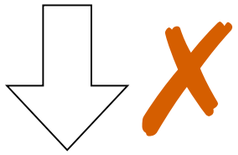


```
q_p |> Join(q_l, q_p.location_id .== q_l.location_id) |>
       Select(q_p.person_id)
```

```
FROM person p
```

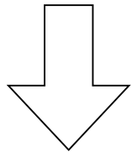


```
FROM person p
WHERE p.year_of_birth >= 1970
```

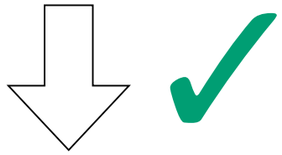


```
FROM person p
WHERE p.year_of_birth >= 1970
JOIN location l
  ON (p.location_id = l.location_id)
```

```
From(person)
```

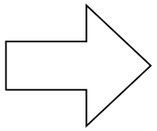


```
From(person) |>
Where(Get.year_of_birth .>= 1970)
```

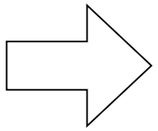


```
From(person) |>
Where(Get.year_of_birth .>= 1970) |>
Join(:location => From(location),
     Get.location_id .==
     Get.location.location_id)
```

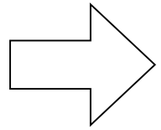
```
FROM ...
```



```
FROM ...
JOIN ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...
```



```
SELECT ...
FROM ...
JOIN ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...
```



```
FROM (
    SELECT ...
    FROM ...
    JOIN ...
    WHERE ...
    GROUP BY ...
    HAVING ...
    ORDER BY ...)
```

```
From(table)
```

```
SELECT ???
FROM $table
```

```
        |>
Where(condition)
```

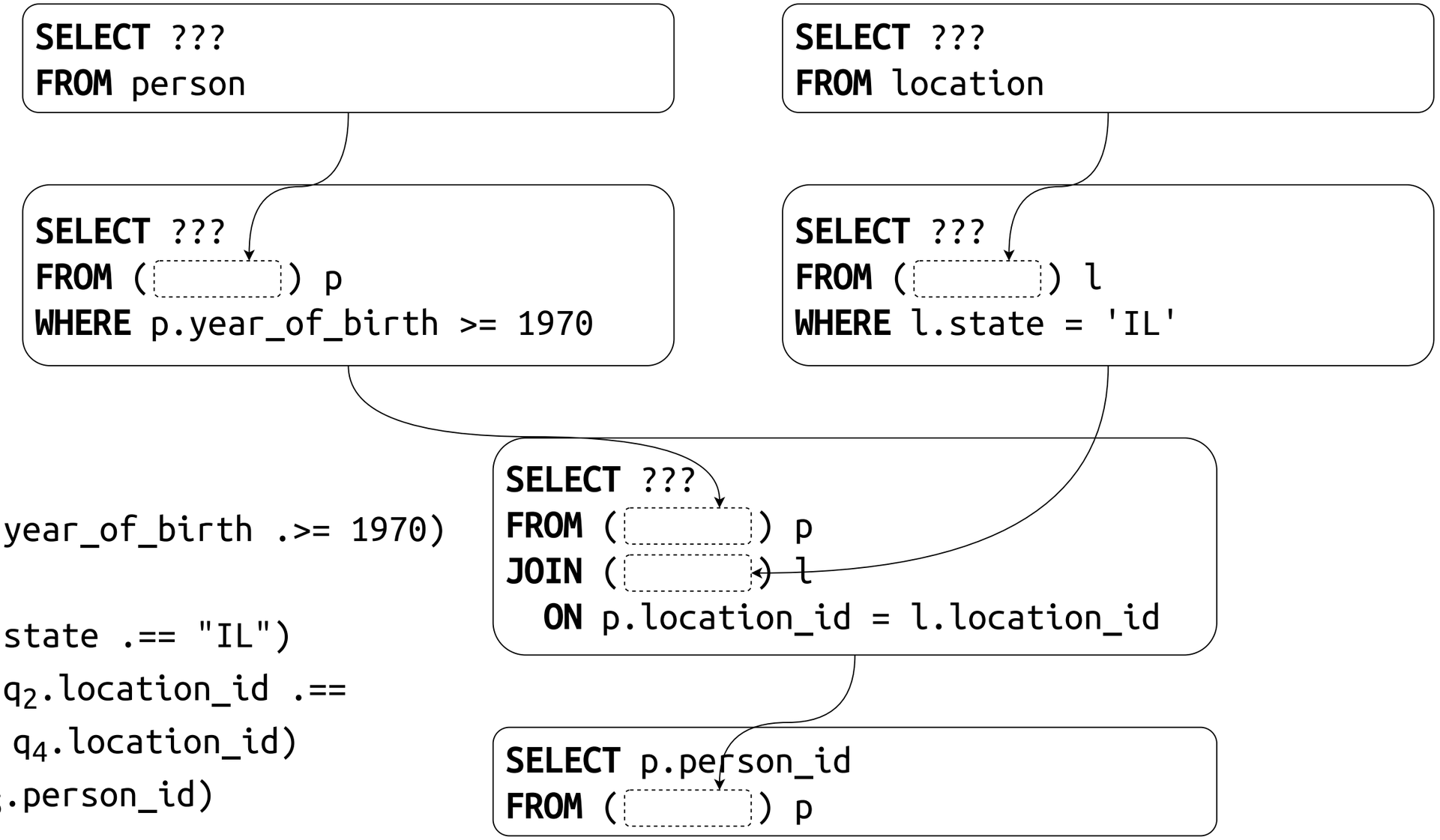
```
SELECT ???
FROM (        )
WHERE $condition
```

```
        |>
Join(        , on)
```

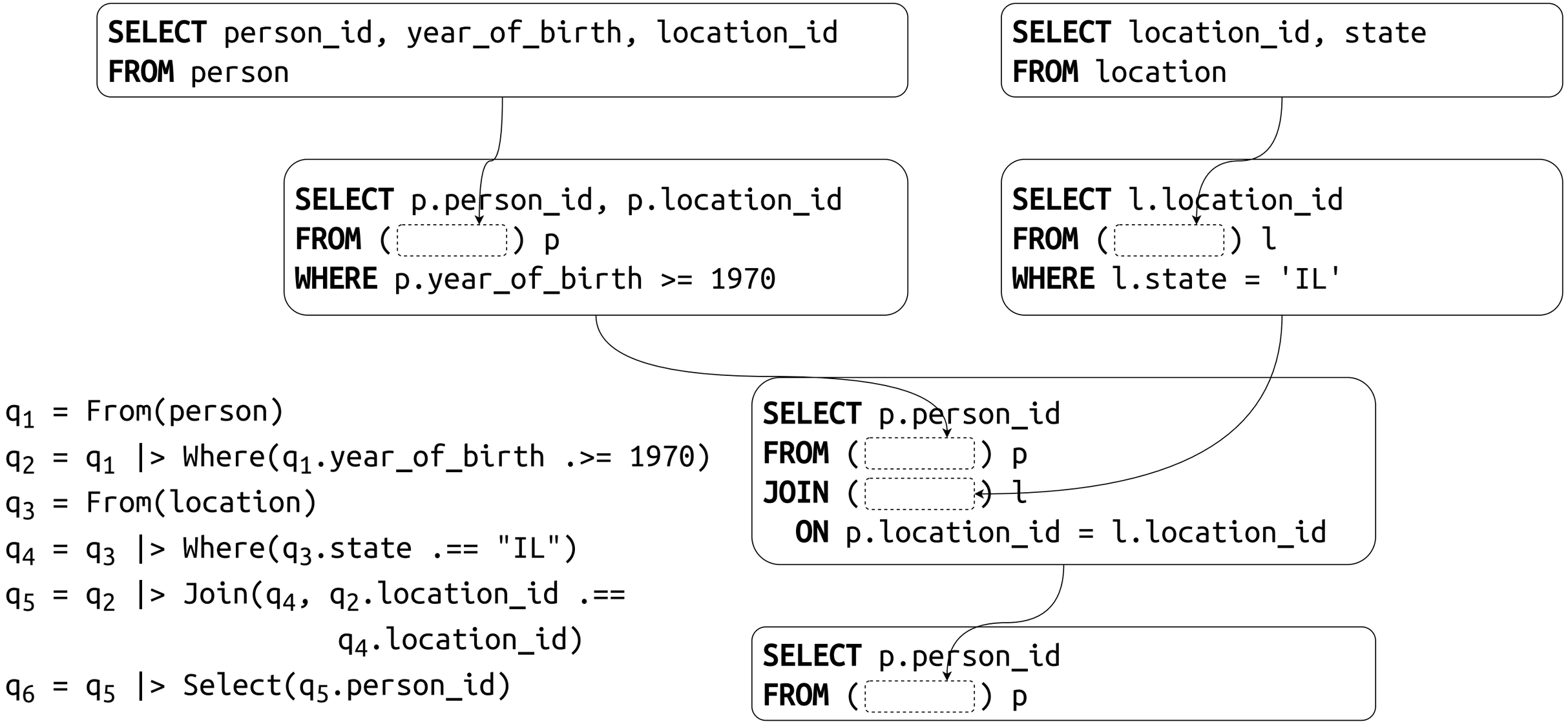
```
SELECT ???
FROM (        )
JOIN (        )
  ON $on
```

```
        |>
Select(list...)
```

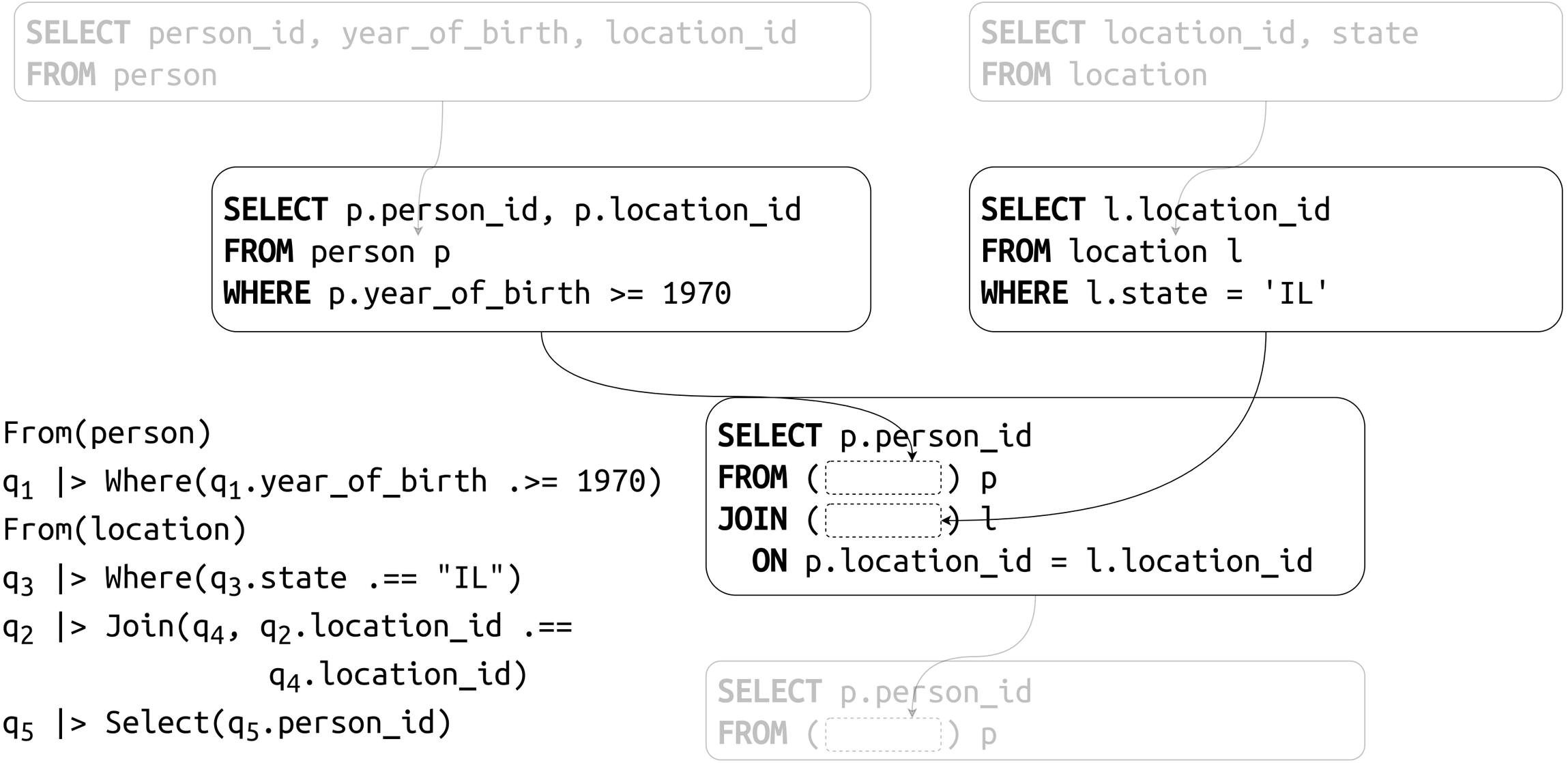
```
SELECT $(list...)
FROM (        )
```




```
q1 = From(person)
q2 = q1 |> Where(q1.year_of_birth .>= 1970)
q3 = From(location)
q4 = q3 |> Where(q3.state .== "IL")
q5 = q2 |> Join(q4, q2.location_id .==
                    q4.location_id)
q6 = q5 |> Select(q5.person_id)
```




```
q1 = From(person)
q2 = q1 |> Where(q1.year_of_birth .>= 1970)
q3 = From(location)
q4 = q3 |> Where(q3.state .== "IL")
q5 = q2 |> Join(q4, q2.location_id .==
                    q4.location_id)
q6 = q5 |> Select(q5.person_id)
```




```
q1 = From(person)
q2 = q1 |> Where(q1.year_of_birth .>= 1970)
q3 = From(location)
q4 = q3 |> Where(q3.state .== "IL")
q5 = q2 |> Join(q4, q2.location_id .==
                    q4.location_id)
q6 = q5 |> Select(q5.person_id)
```

Find patients

- *born in or after 1970*
- *living in Illinois*

```
q1 = From(person)
q2 = q1 |> Where(q1.year_of_birth .>= 1970)
q3 = From(location)
q4 = q3 |> Where(q3.state .== "IL")
q5 = q2 |> Join(q4, q2.location_id .==
                    q4.location_id)
q6 = q5 |> Select(q5.person_id)
```

```
using FunSQL: AS, FROM, JOIN, OP, SELECT, WHERE

FROM(
  FROM(:person |> AS(:p)) |>
  WHERE(OP(">=", (:p, :year_of_birth), 1970)) |>
  SELECT((:p, :person_id),
         (:p, :location_id)) |>
  AS(:p)) |>
JOIN(
  FROM(:location |> AS(:l)) |>
  WHERE(OP("=", (:l, :state), "IL")) |>
  SELECT((:l, :location_id)) |>
  AS(:l),
  OP("=", (:p, :location_id),
          (:l, :location_id))) |>
SELECT((:p, :person_id))
```

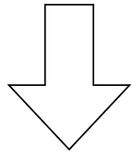
Find patients

- *born in or after 1970*
- *living in Illinois*

```
q1 = From(person)
q2 = q1 |> Where(q1.year_of_birth .>= 1970)
q3 = From(location)
q4 = q3 |> Where(q3.state .== "IL")
q5 = q2 |> Join(q4, q2.location_id .==
                    q4.location_id)
q6 = q5 |> Select(q5.person_id)
```

```
SELECT p.person_id
FROM (SELECT p.person_id, p.location_id
      FROM person p
      WHERE p.year_of_birth >= 1970) p
JOIN (SELECT l.location_id
      FROM location l
      WHERE l.state = 'IL') l
  ON p.location_id = l.location_id
```

```
using FunSQL: Define

const ObservationYear = 2000

FromPerson() =
    From(person) |>
    Define(:approx_age => ObservationYear .- Get.year_of_birth)

FromAdult() =
    FromPerson() |>
    Where(Get.approx_age .>= 18)
```



```
julia> q = FromAdult() |> Select(Get.person_id)
```

```
julia> q = FromAdult() |> Select(Get.person_id)
let person = SQLTable(:person, …),
    q₁ = From(person),
    q₂ = q₁ |> Define(Fun."-"(Lit(2000), Get.year_of_birth) |>
                      As(:approx_age)),
    q₃ = q₂ |> Where(Fun.">="(Get.approx_age, Lit(18))),
    q₄ = q₃ |> Select(Get.person_id)
    q₄
end
```

FunSQL? Who Needs It?

Query Algebra

Correlated Queries

Aggregate & Window Functions

Conclusion

Find all patients born in or after $YEAR.

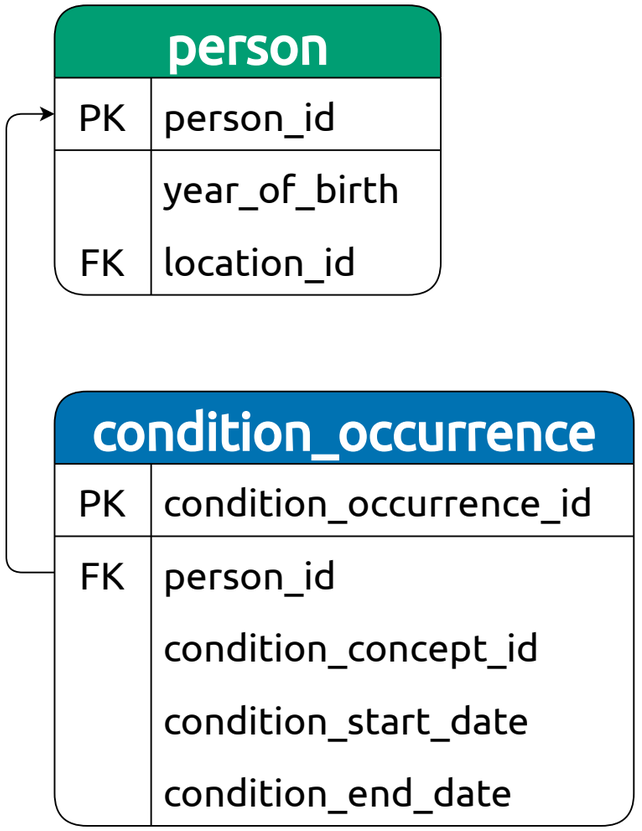
```
sql = """
SELECT p.person_id
FROM person p
WHERE p.year_of_birth >= :YEAR
"""

params = (YEAR = 1970,)
```

```
using FunSQL: Var, pack

q = From(person) |>
    Where(Get.year_of_birth .>= Var.YEAR) |>
    Select(Get.person_id)

sql = render(q, dialect = :sqlite)

params = pack(sql, (YEAR = 1970,))
```

```
DBInterface.execute(conn, sql, params)
```

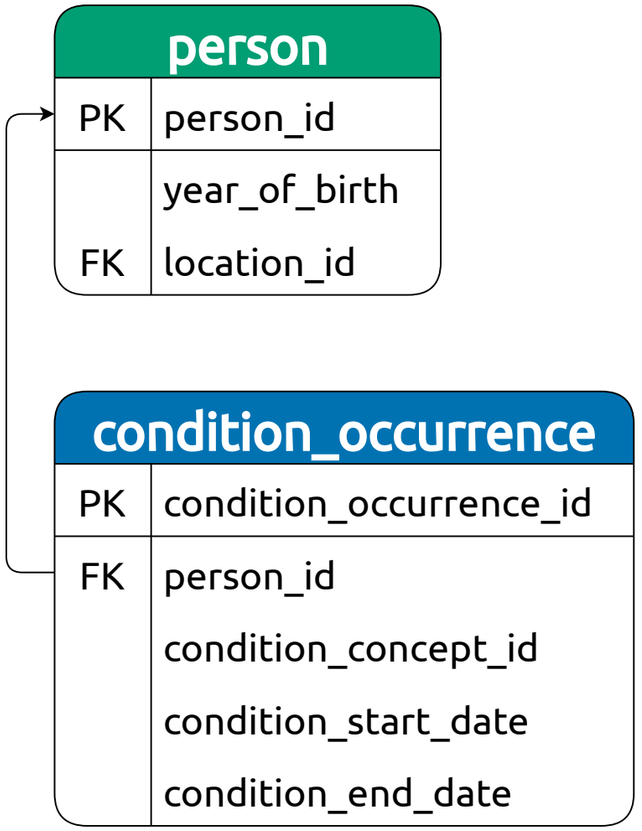
Find patients with at least one medical condition.

```
SELECT p.*
FROM person p
WHERE EXISTS (SELECT NULL
              FROM condition_occurrence c
              WHERE c.person_id = p.person_id)
```



Find patients with at least one medical condition.

```
SELECT p.*
FROM person p
WHERE EXISTS (SELECT NULL
              FROM condition_occurrence c
              WHERE c.person_id = p.person_id)
```

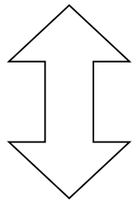



```
q_p = From(person)
q_c = From(condition_occurrence)
q_corr = q_c |> Where(q_c.person_id .== q_p.person_id)
q = q_p |> Where(Fun.exists(q_corr))
```

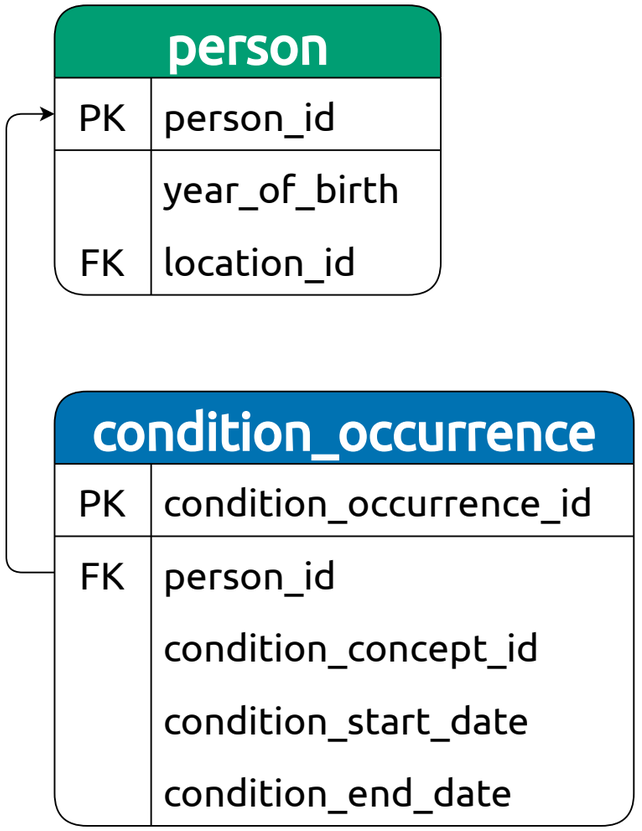
```
ERROR: Cannot find person_id
```

Find patients with at least one medical condition.

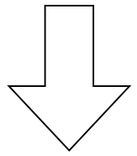
```
SELECT p.*
FROM person p
WHERE EXISTS (SELECT NULL
              FROM condition_occurrence c
              WHERE c.person_id = p.person_id)
```



```
                                SELECT NULL
CorrelatedCondition : X ↦       FROM condition_occurrence c
                                WHERE c.person_id = X

SELECT p.*
FROM person p
WHERE EXISTS CorrelatedCondition(p.person_id)
```



```
From(condition_occurrence) |>
Where(Get.person_id .== Var.X)
```



```
using FunSQL: Bind

CorrelatedCondition(X) =
    From(condition_occurrence) |>
    Where(Get.person_id .== Var.X) |>
    Bind(:X => X)
```

```
CorrelatedCondition(6)
```

```
SELECT c.*
FROM condition_occurrence c
WHERE c.person_id = :X
```

```
SELECT c.*
FROM condition_occurrence c
WHERE c.person_id = 6
```

```
CorrelatedCondition(X) =
    From(condition_occurrence) |>
    Where(Get.person_id .== Var.X) |>
    Bind(:X => X)
```

```
From(person) |>
Where(Fun.exists(
        CorrelatedCondition(Get.person_id)))
```

```
SELECT p.*
FROM person p
WHERE EXISTS (SELECT NULL
                FROM condition_occurrence c
                WHERE c.person_id = p.person_id)
```

FunSQL? Who Needs It?

Query Algebra

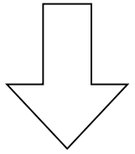
Correlated Queries

Aggregate & Window Functions

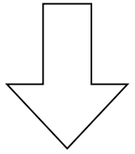
Conclusion

Number of patients by the year of birth.

```
FROM person p
```



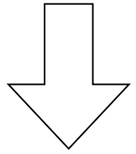
```
FROM person p
GROUP BY p.year_of_birth
```



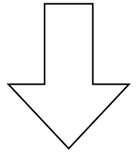
```
SELECT p.year_of_birth, COUNT(*)
FROM person p
GROUP BY p.year_of_birth
```

```
using FunSQL: Agg, Group
```

```
From(person)
```



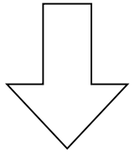
```
From(person) |>
Group(Get.year_of_birth)
```



```
From(person) |>
Group(Get.year_of_birth) |>
Select(Get.year_of_birth, Agg.count())
```

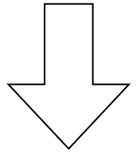
Average year of birth.

```
FROM person p
```

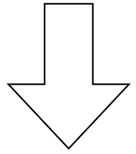


```
SELECT AVG(p.year_of_birth)
FROM person p
```

```
From(person)
```



```
From(person) |>
Group()
```

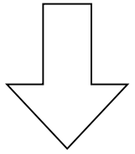


```
From(person) |>
Group() |>
Select(Agg.avg(Get.year_of_birth))
```

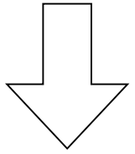
Patients who saw a doctor within the last 12 months.

visit_occurrence	
PK	visit_occurrence_id
FK	person_id
	visit_concept_id
	visit_start_date
	visit_end_date

```
FROM visit_occurrence v
```

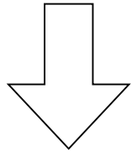


```
FROM visit_occurrence v
GROUP BY v.person_id
```

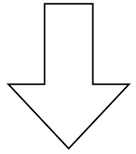


```
SELECT v.person_id
FROM visit_occurrence v
GROUP BY v.person_id
HAVING CURRENT_DATE -
       MAX(v.visit_end_date) <= 365
```

```
From(visit_occurrence)
```



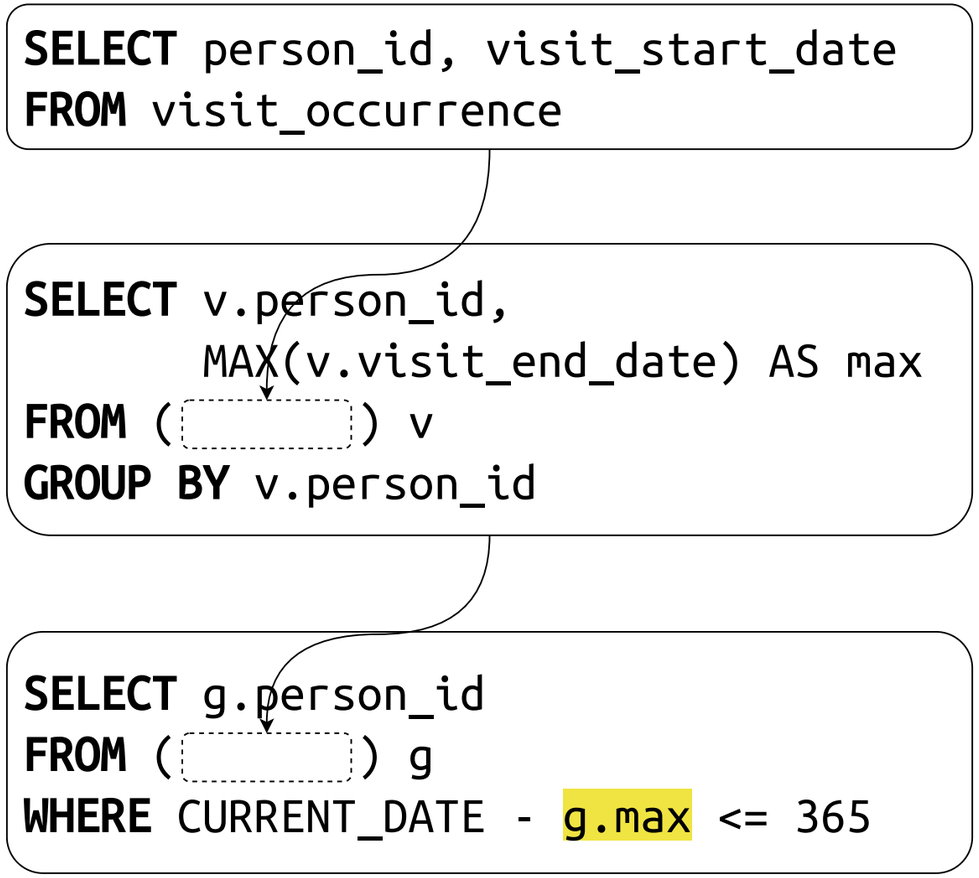
```
From(visit_occurrence) |>
Group(Get.person_id)
```



```
From(visit_occurrence) |>
Group(Get.person_id) |>
Where(Fun.current_date() .-
      Agg.max(Get.visit_end_date) .<= 365)
```

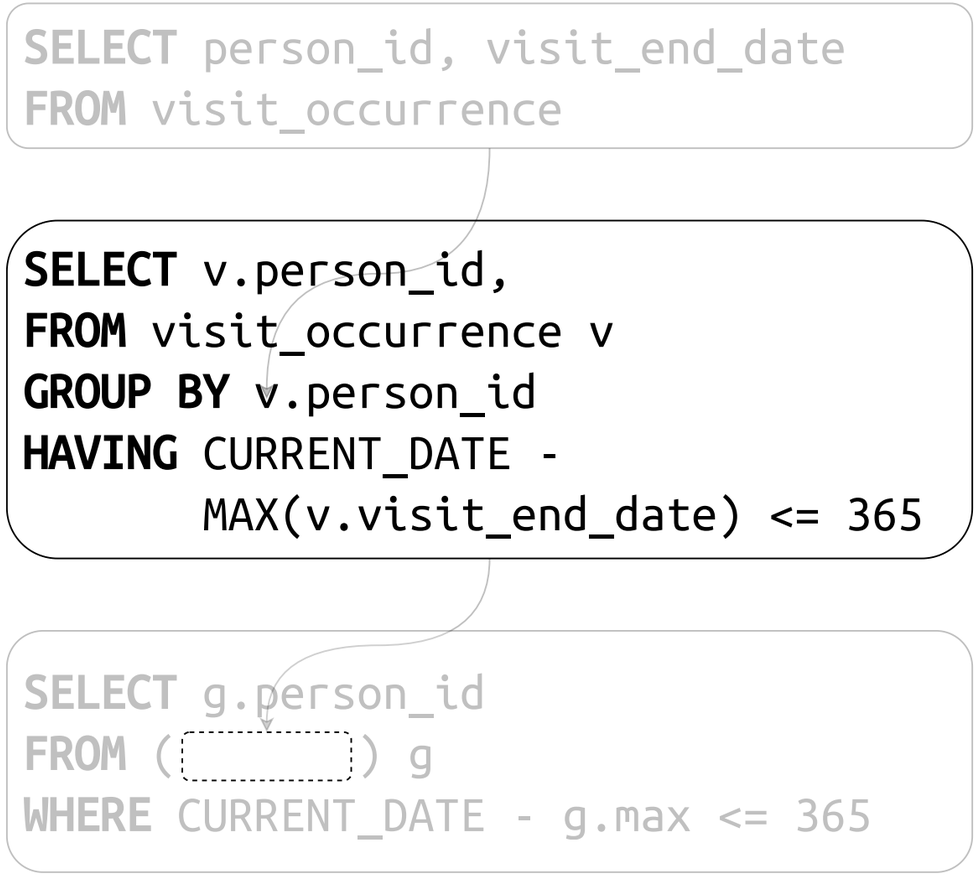
Patients who saw a doctor within the last 12 months.

```
From(visit_occurrence) |>
Group(Get.person_id) |>
Where(Fun.current_date() .-
      Agg.max(Get.visit_end_date) .<= 365)
```



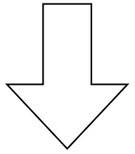
Patients who saw a doctor within the last 12 months.

```
From(visit_occurrence) |>
Group(Get.person_id) |>
Where(Fun.current_date() .-
      Agg.max(Get.visit_end_date) .<= 365)
```



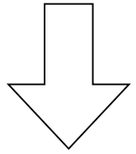
For each visit, show the time passed since the previous visit.

```
FROM visit_occurrence v
```



```
FROM visit_occurrence v
WINDOW w AS (
  PARTITION BY v.person_id
  ORDER BY v.visit_start_date
  ROWS BETWEEN UNBOUNDED PRECEDING AND
               1 PRECEDING)
```

```
using FunSQL: Partition


From(visit_occurrence)
```



```
From(visit_occurrence) |>
Partition(Get.person_id,
          order_by = [Get.visit_start_date],
          frame = (mode = :rows,
                   start = -Inf,
                   finish = -1))
```

visit_occurrence	
PK	visit_occurrence_id
FK	person_id
	visit_concept_id
	visit_start_date
	visit_end_date

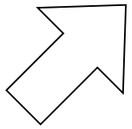
For each visit, show the time passed since the previous visit.

visit_occurrence	
PK	visit_occurrence_id
FK	person_id
	visit_concept_id
	visit_start_date
	visit_end_date

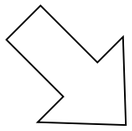
```
SELECT v.visit_start_date,
       v.visit_start_date -
       MAX(v.visit_end_date) OVER w
FROM visit_occurrence v
WINDOW w AS (
  PARTITION BY v.person_id
  ORDER BY v.visit_start_date
  ROWS BETWEEN UNBOUNDED PRECEDING AND
               1 PRECEDING)
```

```
From(visit_occurrence) |>
Partition(Get.person_id,
          order_by = [Get.visit_start_date],
          frame = (mode = :rows,
                   start = -Inf,
                   finish = -1)) |>
Select(Get.visit_start_date,
       :gap => Get.visit_start_date .-
               Agg.max(Get.visit_end_date))
```

```
From(visit_occurrence)
```



```
From(visit_occurrence) |>
Group(Get.person_id) |>
Where(Fun.current_date() .-
      Agg.max(Get.visit_end_date) .<= 365)
```



```
From(visit_occurrence) |>
Partition(Get.person_id,
          order_by = [Get.visit_start_date],
          frame = (mode = :rows,
                   start = -Inf,
                   finish = -1)) |>
Select(Get.visit_start_date,
       :gap => Get.visit_start_date .-
               Agg.max(Get.visit_end_date))
```

```
From(visit_occurrence)
```

person_id	
1	■
1	◀
1	●
2	▱
2	▯

```
Group(Get.person_id)
```

person_id
1
2

1	■
1	◀
1	●

2	▱
2	▯

```
Partition(Get.person_id, order_by = [Get.visit_start_date],
          frame = (mode = :rows, start = -Inf, finish = -1))
```

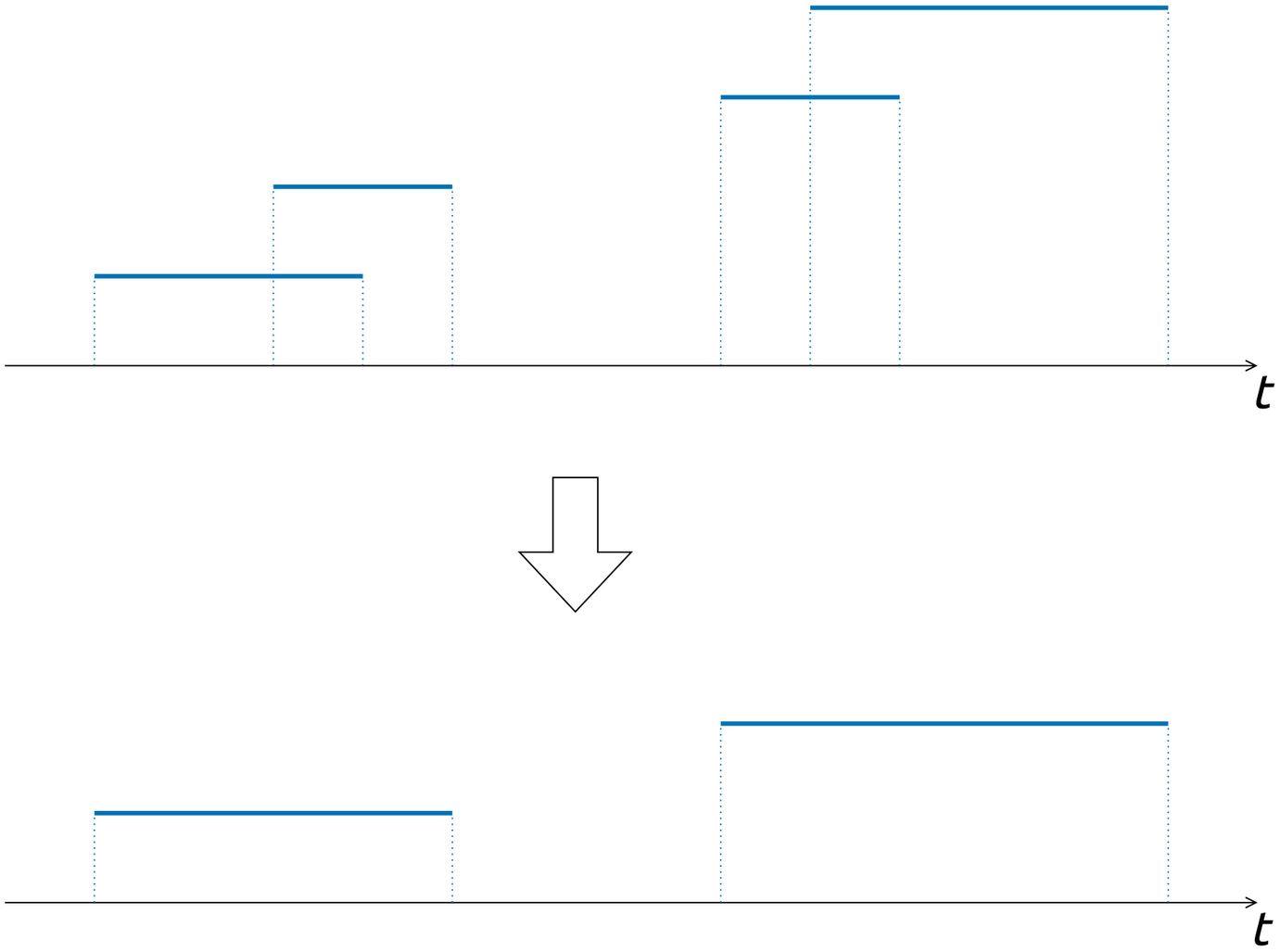
person_id	
1	■
1	◀
1	●
2	▱
2	▯

1	■

1	■
1	◀

2	▱

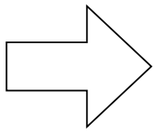
Merge overlapping visits.



visit_occurrence	
PK	visit_occurrence_id
FK	person_id
	visit_concept_id
	visit_start_date
	visit_end_date

Merge overlapping visits.

[visit_start_date, visit_end_date]
———
——
——
———

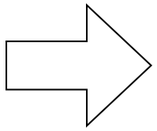


[visit_start_date, visit_end_date]	new
———	1
——	0
——	1
———	0

```
From(visit_occurrence) |>
Partition(Get.person_id, order_by = [Get.visit_start_date],
          frame = (mode = :rows, start = -Inf, finish = -1)) |>
Define(:boundary => Agg.max(Get.visit_end_date)) |>
Define(:gap => Get.visit_start_date .- Get.boundary) |>
Define(:new => Fun.case(Get.gap .<= 0, 0, 1))
```

Merge overlapping visits.

[visit_start_date, visit_end_date]	new
	1
	0
	1
	0

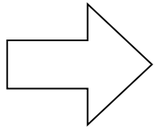


[visit_start_date, visit_end_date]	new	group
	1	1
	0	1
	1	2
	0	2

```
Partition(Get.person_id, order_by = [Get.visit_start_date],
          frame = (mode = :rows, start = -Inf, finish = -1)) |>
Define(:boundary => Agg.max(Get.visit_end_date)) |>
Define(:gap => Get.visit_start_date .- Get.boundary) |>
Define(:new => Fun.case(Get.gap .<= 0, 0, 1)) |>
Partition(Get.person_id, order_by = [Get.visit_start_date, .- Get.new],
          frame = :rows) |>
Define(:group => Agg.sum(Get.new))
```

Merge overlapping visits.

[visit_start_date, visit_end_date]	new	group
———	1	1
——	0	1
——	1	2
———	0	2



[start_date, end_date]	group
———	1
———	2

```
Define(:new => Fun.case(Get.gap .<= 0, 0, 1)) |>
Partition(Get.person_id, order_by = [Get.visit_start_date, .- Get.new],
          frame = :rows) |>
Define(:group => Agg.sum(Get.new)) |>
Group(Get.person_id, Get.group) |>
Define(:start_date => Agg.min(Get.visit_start_date),
       :end_date => Agg.max(Get.visit_end_date)) |>
Select(Get.person_id, Get.start_date, Get.end_date)
```

Merge overlapping visits.

visit_occurrence	
PK	visit_occurrence_id
FK	person_id
	visit_concept_id
	visit_start_date
	visit_end_date

```
CollapseIntervals(start_date, end_date) =
    Partition(Get.person_id, order_by = [start_date],
              frame = (mode = :rows, start = -Inf, finish = -1)) |>
    Define(:boundary => Agg.max(end_date)) |>
    Define(:gap => start_date .- Get.boundary) |>
    Define(:new => Fun.case(Get.gap .<= 0, 0, 1)) |>
    Partition(Get.person_id, order_by = [start_date, .- Get.new],
              frame = :rows) |>
    Define(:group => Agg.sum(Get.new)) |>
    Group(Get.person_id, Get.group) |>
    Define(:start_date => Agg.min(start_date),
           :end_date => Agg.max(end_date))

q = From(visit_occurrence) |>
    CollapseIntervals(Get.visit_start_date, Get.visit_end_date) |>
    Select(Get.person_id, Get.start_date, Get.end_date)
```

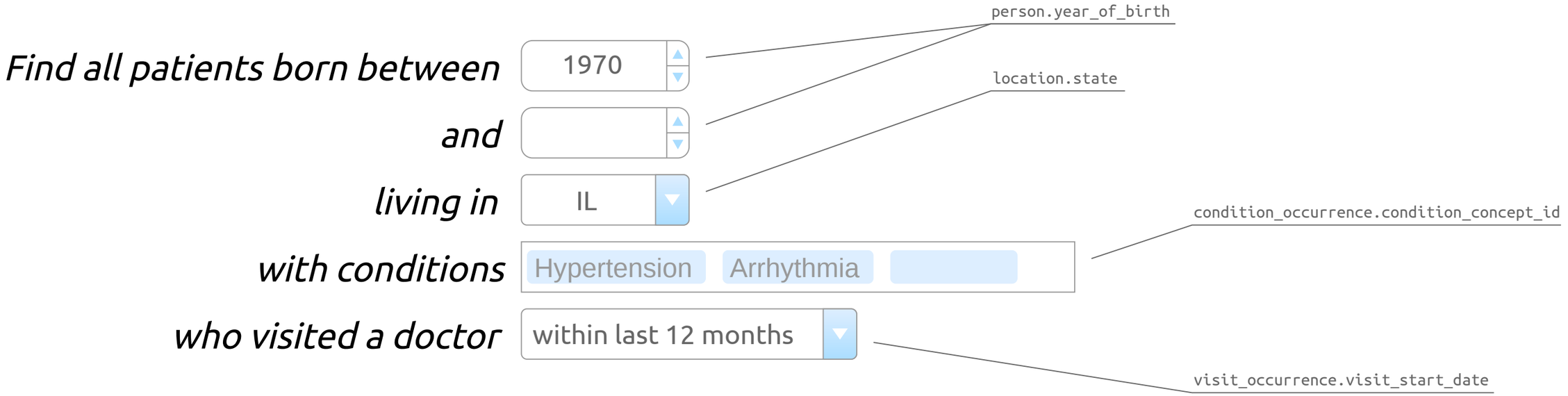
FunSQL? Who Needs It?

Query Algebra

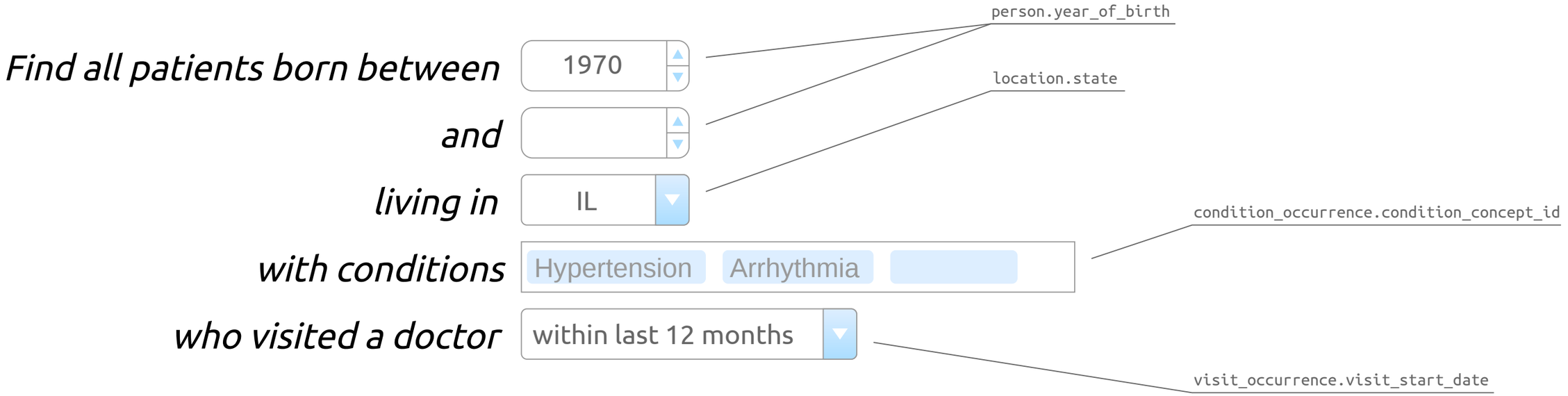
Correlated Queries

Aggregate & Window Functions

Conclusion




```
function find_patients(conn; start_year = nothing,
                             end_year = nothing,
                             state = nothing,
                             condition_concepts = [],
                             latest_visit_threshold = nothing)
    q = FindPatients(; start_year, end_year, state,
                       condition_concepts, latest_visit_threshold)
    sql = render(q, dialect = :postgresql)
    DBInterface.execute(conn, sql)
end
```




```
FindPatients(; start_year = nothing, end_year = nothing,
               state = nothing,
               condition_concepts = [],
               latest_visit_threshold = nothing) =
    From(person) |>
    FilterByYearOfBirth(; start_year, end_year) |>
    FilterByState(; state) |>
    FilterByConditions(; condition_concepts) |>
    FilterByLatestVisit(; latest_visit_threshold) |>
    Select(Get.person_id)
```

```
FilterByYearOfBirth(; start_year, end_year) =
    if start_year !== nothing && end_year !== nothing
        Where(Fun.between(Get.year_of_birth, start_year, end_year))
    elseif start_year !== nothing
        Where(Get.year_of_birth .>= start_year)
    elseif end_year !== nothing
        Where(Get.year_of_birth .<= end_year)
    else
        identity
    end


FilterByState(; state) =
    if state !== nothing
        Join(:location => From(location) |>
                          Where(Get.state .== state),
             Get.location_id .== Get.location.location_id)
    else
        identity
    end
```

```
ConditionsByPerson(person_id; condition_concepts) =
    From(condition_occurrence) |>
    Where(Fun.and(Fun.in(Get.condition_concept_id, condition_concepts...),
                  Get.person_id .== Var.person_id)) |>
    Bind(:person_id => person_id)

FilterByConditions(; condition_concepts) =
    if !isempty(condition_concepts)
        Where(Fun.exists(ConditionsByPerson(Get.person_id; condition_concepts)))
    else
        identity
    end
```

```
FilterByLatestVisit(; latest_visit_threshold) =
    if latest_visit_threshold !== nothing
        Join(:visit_group => From(visit_occurrence) |>
                             Group(Get.person_id),
             Get.person_id .== Get.visit_group.person_id) |>
        Define(:latest_visit_date =>
                    Agg.max(Get.visit_start_date, over = Get.visit_group)) |>
        Where(Fun.current_date() .- Get.latest_visit_date .<= latest_visit_threshold)
    else
        identity
    end
```

DONE

- Select, Where, Join, Define,
  Group, Partition, Append (UNION ALL),
  Parameterized and Correlated Queries
- Tested with SQLite, PostgresSQL, RedShift,
  Microsoft SQL Server

TODO (June 2021)

- ORDER BY and LIMIT, CTE and WITH clause,
  WITH RECURSIVE, CUBE/ROLLUP and more
- Documentation and examples
- Comprehensive support for SQL dialects
- Tracking column and expression types
- CRUD (INSERT, UPDATE, DELETE)
- DDL (CREATE TABLE) and schema introspection

Example

https://github.com/MechanicalRabbit/OHDSICohortExpressions.jl

Contact Us

julialang.zulipchat.com: funsql.jl