# FunSQL:
# A library for compositional construction of SQL queries

*Find all patients born at or after 1950.*

```
function find_patients(conn)
    sql = """
    SELECT person_id
    FROM patient
    WHERE year_of_birth >= 1950
    """

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

- What is SQL? Data is often stored in relational databases, and to retrieve it, we write queries in SQL.
- Popular databases with SQL interface include MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Redshift, BigQuery, and many others.
- Julia already has a number of libraries that let you interact with SQL databases.
- Why another tool?

*Find all patients born between* [ 1950 ▲▼ ]

*and* [ ▲▼ ]

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

*Find all patients born between* [ 1950 ]

*and* [ ]

*living in* [ IL ▾ ]    "location"

*with conditions* [ Hypertension | Arrhythmia | ]    "condition_occurrence"

*who visited a doctor* [ within last 12 months ▾ ]

"visit_occurrence"

```
function find_patients(conn; start_year = nothing,
                             end_year = nothing,
                             state = nothing,
                             conditions = [],
                             last_visit = nothing)
    sql = """
    SELECT person_id
    FROM patient
    """
    ???
    DBInterface.execute(conn, sql)
end
```

- Clearly, a more systematic approach is necessary.

**location**

| PK | location_id |
|----|-------------|
|    | city |
|    | state |
|    | zip |

**condition_occurrence**

| PK | condition_occurrence_id |
|----|-------------------------|
| FK | person_id |
|    | condition_concept_id |
|    | condition_start_date |
|    | condition_end_date |

**person**

| PK | person_id |
|----|-----------|
|    | year_of_birth |
| FK | location_id |

**visit_occurrence**

| PK | visit_occurrence_id |
|----|---------------------|
| FK | person_id |
|    | visit_concept_id |
|    | visit_start_date |
|    | visit_end_date |

- A small diversion to introduce the database schema which we will use in subsequent examples.
- OMOP Common Data Model is a popular open-source used in healthcare of observational research.
- As typical in healthcare, the schema is patient-centric. The *person* table stores information about patients including basic demographic information. Their address is stored in a separate table called *location*.
- Most of the patient data consists of clinical events: encounters with healthcare providers, recorded observations, diagnosed conditions, performed procedures, etc.

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

const visit_occurrence =
    SQLTable(name = :visit_occurrence,
             columns = [:visit_occurrence_id, :person_id,
                        :visit_concept_id,
                        :visit_start_date, :visit_end_date])
```
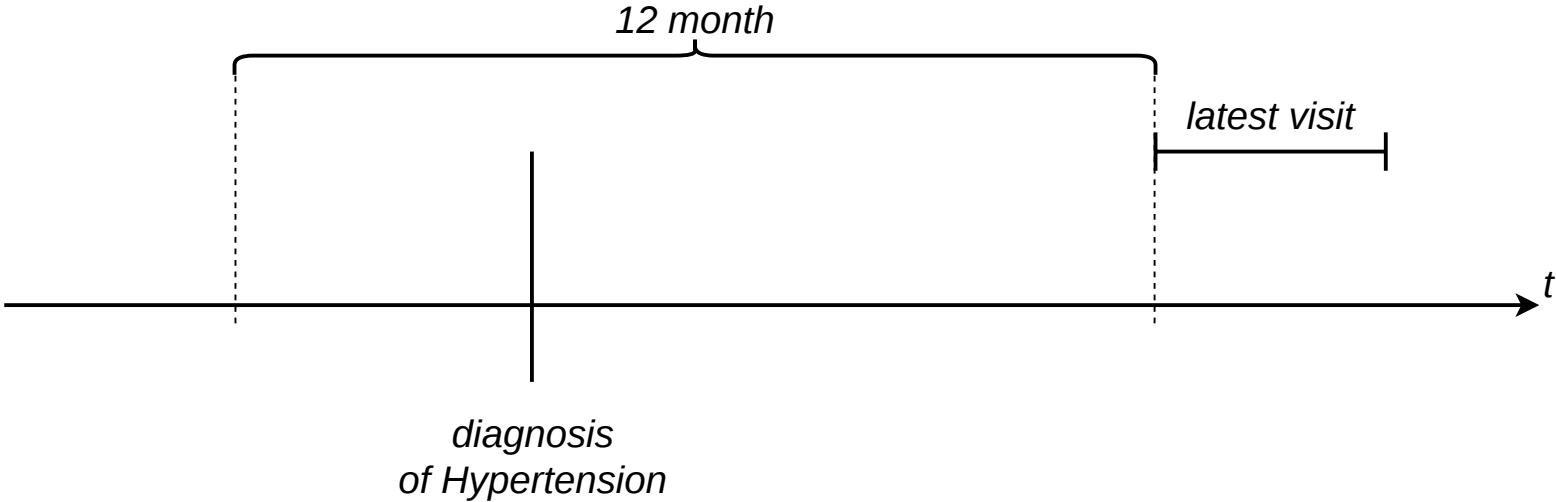
*Find all patients born in 1970 or later and living in Illinois who have been diagnosed with Hypertension no more than 12 months before their latest visit to a healthcare provider.*

*Find patients*

- *born in 1970 or later,*
- *living in Illinois,*
- *such that their latest visit to the healthcare provider satisfies:*

*Find all patients born in 1970 or later.*

```
FROM person p
```
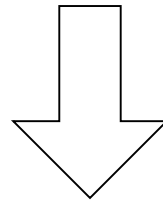
⬇
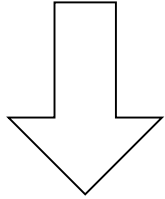
```
FROM person p
WHERE p.year_of_birth >= 1950
```

⬇
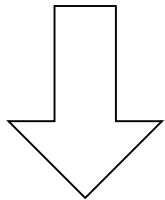
```
SELECT p.person_id
FROM person p
WHERE p.year_of_birth >= 1950
```

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

⬇
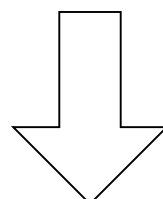
```
q = From(person) |>
    Where(Get.year_of_birth .>= 1950)
```

⬇

```
q = From(person) |>
    Where(Get.year_of_birth .>= 1950) |>
    Select(Get.person_id)
```

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

## Bound References

```
q.year_of_birth        q.person_id

q1 = From(person)
q2 = q1 |> Where(q1.year_of_birth .>= 1950)
q3 = q2 |> Select(q2.person_id)
```

Both bound and unbound references are supported.
Unbound references make decomposition easier.

## Unbound References

```
Get.year_of_birth        Get.person_id

q = From(person) |>
    Where(Get.year_of_birth .>= 1950) |>
    Select(Get.person_id)
```
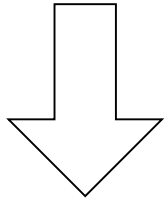
```
BornInOrLater(y) =
    Get.year_of_birth .>= y

q = From(person) |>
    Where(BornInOrLater(1950)) |>
    Select(Get.person_id)
```

*Show patients with their state of residence.*

```
FROM person p
```

```
From(person)
```

⬇

⬇

```
FROM person p
JOIN location l
  ON (p.location_id = l.location_id)
```
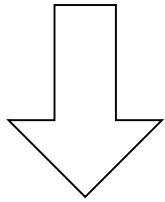
```
person |>
Join(:location => location,
     Get.location_id .==
     Get.location.location_id)
```

⬇

⬇

```
SELECT p.person_id, l.state
FROM person p
JOIN location l
  ON (p.location_id = l.location_id)
```
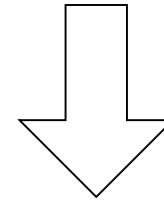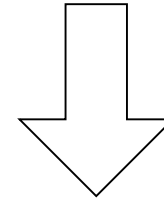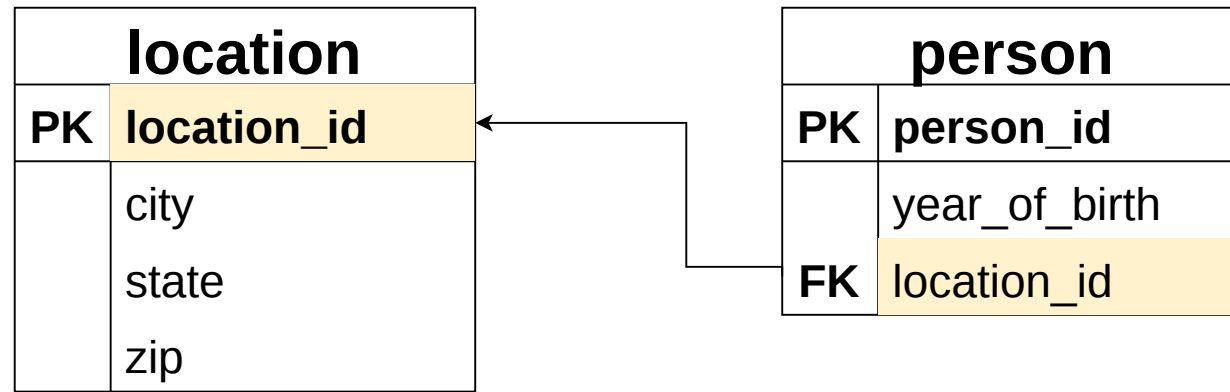
```
person |>
Join(:location => location,
     Get.location_id .==
     Get.location.location_id) |>
Select(Get.person_id,
       Get.location.state)
```

| location | | | person | |
|---|---|---|---|---|
| **PK** | **location_id** | | **PK** | **person_id** |
| | city | | | year_of_birth |
| | state | | **FK** | location_id |
| | zip | | | |

```
q1 = From(person)
q2 = From(location)
q3 = q1 |> Join(q2, q1.location_id .== q2.location_id)
```

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

*Find patterns*

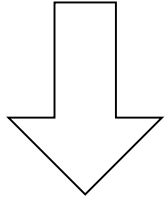- *born in 1970 or later,*
- *living in Illinois*

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

```
From(location) |>
Where(Get.state .== "IL")
```
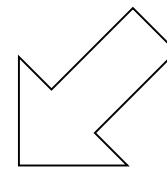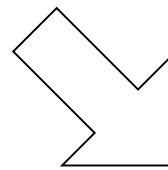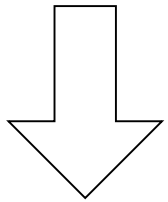
```
From(person) |>
Where(Get.year_of_birth .>= 1970)
Join(:location => From(location) |>
                  Where(Get.state .== "IL"),
     on = Get.location_id .==
          Get.location.location_id)
```
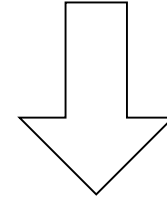
```
FROM person p                          From(person)



        ⬇                                      ⬇



FROM person p                          From(person) |>
WHERE p.year_of_birth >= 1950          Where(Get.year_of_birth .>= 1950)


    ⬇          ✖                           ⬇          ✔


FROM person p                          From(person) |>
WHERE p.year_of_birth >= 1950          Where(Get.year_of_birth .>= 1950) |>
JOIN location l                        Join(:location => location,
  ON (p.location_id = l.location_id)         Get.location_id .==
                                                 Get.location.location_id)
```