# Building Data Apps with RelationalAI

**Northwest Database Society**
**May 20, 2022**

**Martin Bravenboer**
**VP Engineering**

RelationalAI

RelationalAI

# The next-generation database system for intelligent data apps based on relational knowledge graphs

Our target workloads:

1. Reasoning
2. Graph analytics
3. Relational machine learning
4. Optimization

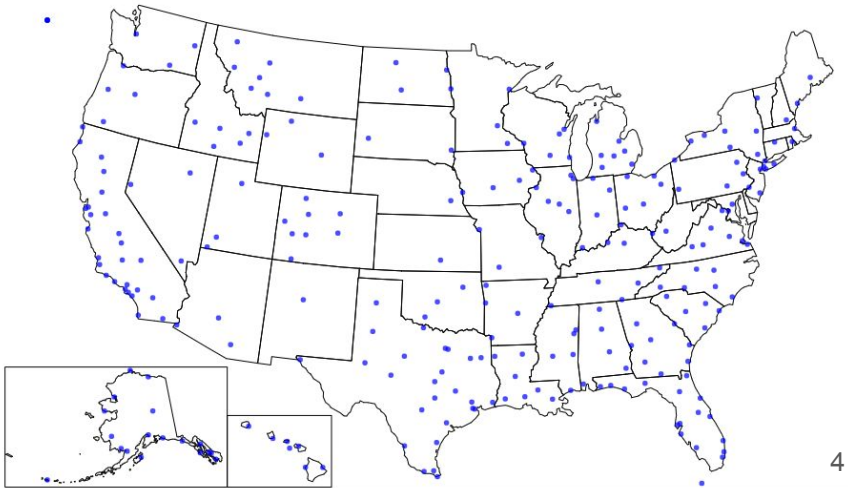
# Key Innovations for Relational Knowledge Graphs

1. Immutability - Cloud native architecture
   *Data + metadata is versioned and immutable*
2. Declarative and expressive relational language (Rel)
   *Designed for complex business logic, abstraction, schema abstraction, libraries*
3. Join algorithms
   *Worst-case optimal join algorithms specifically suitable for knowledge graphs*
4. Semantic optimization
   *Use knowledge to apply deep and structural optimization*
5. Vectorized and JIT compilation of WCOJ
   *Compile complex joins to native code. Vectorized engine for simpler joins*
6. Live - Incrementality (for data and logic)
   *Maintain computations incrementally*
   *Compile model incrementally*

# Flight Data: How to build an intelligent data app

Opportunities in such an application

- Performance of carriers, airports, aircraft models and individual aircraft
- Discover relationships between seasons, time of day, weather and delays
- Planning and optimization
- Historical trends
- If live, act on issues that are arising



RelationalAI

Primary key?

Hours? Minutes?

Period? Minutes? Hours?

Miles? Kilometers?

Are these exclusive?

| carrier | origin | destination | flight_num | flight_time | tail_num | dep_time | arr_time | dep_delay | arr_delay | taxi_out | taxi_in | distance | cancelled | diverted | id2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F9 | MCI | DEN | 818 | 89 | N916FR | 2005-09-26 08:23:00 UTC | 2005-09-26 09:07:00 UTC | -7 | -8 | 9 | 6 | 533 | N | N | 36606824 |
| WN | LBB | ELP | 819 | 41 | N708SW | 2000-08-18 07:30:00 UTC | 2000-08-18 07:20:00 UTC | 0 | -5 | 7 | 2 | 295 | N | N | 4369021 |
| NW | ATL | MEM | 819 | 52 | N607NW | 2001-11-16 07:15:00 UTC | 2001-11-16 07:29:00 UTC | -5 | -9 | 19 | 3 | 332 | N | N | 11838308 |
| DL | SLC | BOI | 819 | 48 | N296WA | 2001-12-04 22:12:00 UTC | 2001-12-04 23:53:00 UTC | 7 | 4 | 49 | 4 | 291 | N | N | 12060416 |
| WN | PHX | SAN | 819 | 51 | N391SW | 2001-12-05 09:11:00 UTC | 2001-12-05 09:17:00 UTC | 11 | | | | 304 | N | N | 12383068 |
| WN | LAS | AUS | 819 | 135 | N519SW | 2002-04-05 08:18:00 UTC | 2002-04-05 12:47:00 UTC | 8 | | | | 1085 | N | N | 13763279 |
| WN | SJC | LAS | 819 | 63 | N528SW | 2002-07-14 06:30:00 UTC | 2002-07-14 07:47:00 UTC | 0 | -3 | 11 | 3 | 386 | N | N | 15284777 |
| WN | LAS | AUS | 819 | 137 | N502SW | 2002-09-16 08:20:00 UTC | 2002-09-16 12:52:00 UTC | 0 | -8 | 11 | 4 | 1085 | N | N | 16027516 |
| DL | MSP | SLC | 819 | 149 | N3754A | 2002-09-29 19:34:00 UTC | 2002-09-29 21:35:00 UTC | 9 | 15 | 25 | 7 | 991 | N | N | 16399211 |
| DL | MSP | SLC | 819 | 145 | N3745B | 2002-12-06 19:27:00 UTC | 2002-12-06 21:16:00 UTC | -3 | -9 | 18 | 6 | 991 | N | N | 17417961 |
| WN | SJC | LAS | 819 | 54 | N730SW | 2003-06-26 06:30:00 UTC | 2003-06-26 07:44:00 UTC | 0 | -11 | 15 | 5 | 386 | N | N | 20460576 |
| WN | SJC | LAS | 819 | 60 | N501SW | 2003-09-15 06:30:00 UTC | 2003-09-15 07:50:00 UTC | 0 | 0 | 18 | 2 | 386 | N | N | 22113592 |
| NW | ATL | MEM | 819 | 51 | N785NC | 2003-11-20 07:11:00 UTC | 2003-11-20 07:15:00 UTC | -9 | -23 | 10 | 3 | 332 | N | N | 23383534 |
| DL | MSP | ATL | 819 | 120 | N906DE | 2004-02-10 09:59:00 UTC | 2004-02-10 13:36:00 UTC | -6 | 0 | 30 | 7 | 906 | N | N | 25206983 |
| US | CHS | CLT | 820 | 38 | N592US | 2002-07-01 19:32:00 UTC | 2002-07-01 20:54:00 UTC | 37 | 64 | 9 | 35 | 168 | N | N | 15142411 |
| FL | TPA | ATL | 820 | 64 | N955AT | 2003-01-31 11:29:00 UTC | 2003-01-31 12:55:00 UTC | -6 | -5 | 13 | 9 | 406 | N | N | 17949329 |
| WN | RDU | PHL | 820 | 73 | N382SW | 2004-12-02 11:10:00 UTC | 2004-12-02 12:31:00 UTC | 0 | -19 | 5 | 3 | 336 | N | N | 30796766 |
| HP | PHX | BOS | 820 | 0 | N826AW | 2005-10-25 00:00:00 UTC | 2005-10-25 00:00:00 UTC | 0 | 0 | 0 | 0 | 2300 | Y | N | 37174931 |
| US | PBI | CLT | 821 | 90 | N624AU | 2002-05-18 06:35:00 UTC | 2002-05-18 [illegible]:00 UTC | -5 | -8 | 10 | 6 | 590 | N | N | 14247814 |
| US | PBI | CLT | 821 | 87 | [illegible]US | [illegible] | [illegible]4 07:43:00 UTC | -7 | -7 | 16 | 7 | 590 | N | N | 20289351 |
| DL | GSO | CVG | 821 | 70 | N943[illegible] | [illegible] | [illegible]08:12:00 UTC | 1 | 11 | 14 | 7 | 330 | N | N | 21396171 |

Not a delay

Include helicopters?

Risk of messing up aggregates

Possible values?

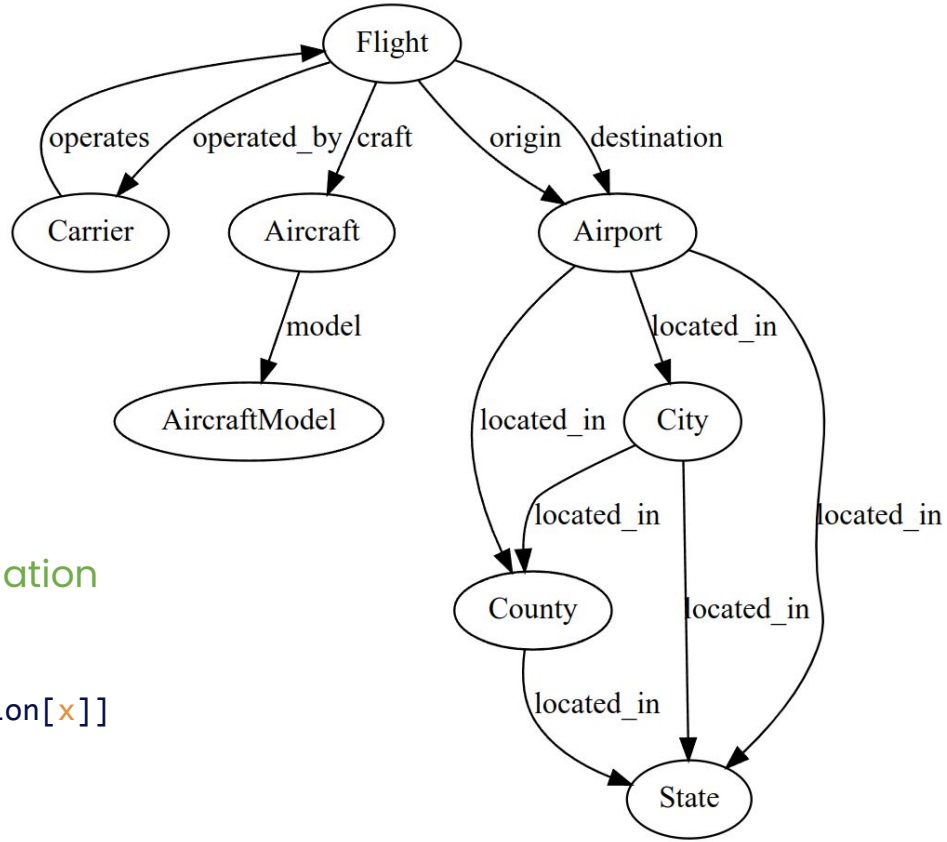
# Model

## Better conceptual model

```
def Heliport(x in Airport) =
    fac_type(x, "HELIPORT")

def cancelled(f in Flight) =
    flight(f) and flight_cancelled(f, "Y")

def arrival_delay[f in Flight] =
    ^Minute[maximum[0, arr_delay[f]]]
```

## Units of measurements to prevent miscalculation

```
def coordinate[x in Airport] =
    ^LLA[latitude[x], longitude[x], elevation[x]]

def LengthUnit = :Feet; :Meters; :Miles
value type Length = LengthUnit, Number
value type Degree = Number
value type LLA = Degree, Degree, Length
```



# Data Integrity

## Nodes involved in relationships

```
ic forall(f, ap: origin(f, ap) implies Flight(f) and Airport(ap))
```

## Required relationships

```
ic forall(f: Flight(f) implies exists origin[f])
```

## Functional dependency (flight can have only one origin)

```
ic function(origin)
```

## Arbitrarily complex

```
ic forall(f in cancelled: not exists flight_duration[f])
ic forall(f in flight: cancelled(f) xor diverted(f) xor arrived(f))
```

# Aggregation

Total number of flights

```
count[Flight]
```

```
37,561,525
```

Carrier with most flights

```
c: count[f: operated_by(f, c)]
```

```
Southwest          5,775,777
Delta              4,477,929
American           4,434,727
```

Carriers mean arrival delay

```
c: mean[f.arrival_delay for f where operated_by(f, c)]
```

```
Airtran            15 min
Atlantic Coast     13 min
United Airlines    13 min
...
Aloha Airlines     6 min
Hawaiian Airlines  3 min
```

Airport ratio of cancelled arriving flights

```
ap: ratio[cancelled, ap.arriving_flight]
```

```
Unalaska           19%
Worcester Regional 11%
Nantucket Memorial  9%
```

# Reasoning

Rel supports general mutually recursive definitions + type inference

```
def located_in(x, y) =
    exists(t: located_in(x, t) and located_in(t, y))
```

General computations

```
def airport_distance[ap1 in Airport, ap2 in Airport] =
    distance[coordinate[ap1], coordinate[ap2]]

def distance[x in LLA, y in LLA] =
    haversine_function[earth_radius, x, y]

def earth_radius =
    ^Length[:Kilometers, 6378.1]
```

Library abstractions and aggregation in recursive definitions

```
def transfer_count[c in Carrier, ap1 in Airport, ap2 in Airport] =
    shortest_path_length[route[c], ap1, ap2]
```

# Schema Abstraction

Rel is not a dynamic language (nor a triple store). Rel exposes the schema logically as data and uses partial evaluation methods to infer and specialize the program to the schema.

## Count all nodes

```
count[x, v: flight_graph(x, v)]
```

```
38,061,144
```

## Count all nodes, grouped by type

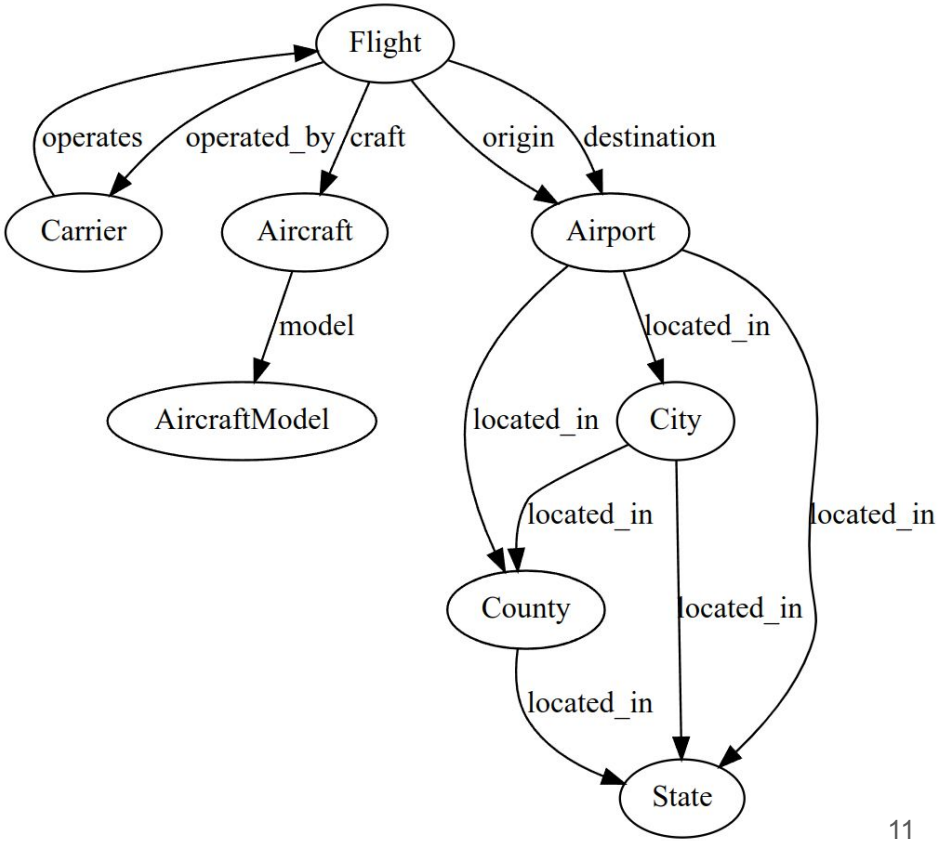
```
x: count[v: flight_graph(x, v)]
```

| | |
|---|---|
| Flight | 37,561,525 |
| Aircraft | 359,928 |
| AircraftModel | 60,461 |
| City | 50,944 |
| Airport | 19,793 |
| Heliport | 5,135 |
| County | 3,009 |
| Major | 270 |
| State | 58 |
| Carrier | 21 |

# Schema Abstraction

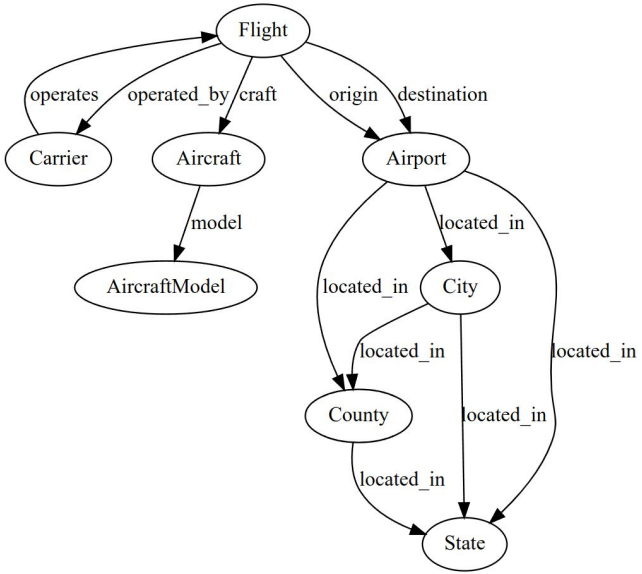
Query the schema and visualize with graphviz

```
module schema_graph[G]
    def node(x) = G(x, _)
    def edge(e, tx, ty) =
        G(e, x, y) and
        G(tx, x) and
        G(ty, y) and
        Entity(x) and
        Entity(y)
        from x, y
end

def output = graphviz[schema_graph[flight_graph]]
```




Schema = data: library applies to both

# Schema Abstraction




## Schema: shortest path from Flight to State

```
shortest_path[schema_graph[flight_graph], :Flight, :State]
```

```
Flight -> destination -> Airport -> located_in -> State
Flight -> origin      -> Airport -> located_in -> State
```

## Schema: all acyclic paths from Flight to State

```
acyclic_path[schema_graph[flight_graph], :Flight, :State]
```

```
Flight -> destination -> Airport -> located_in -> City -> located_in -> County -> located_in -> State
Flight -> destination -> Airport -> located_in -> City -> located_in -> State
Flight -> destination -> Airport -> located_in -> County -> located_in -> State
Flight -> destination -> Airport -> located_in -> State
...
```

*Note: The path algorithms are written in Rel (not foreign functions)*

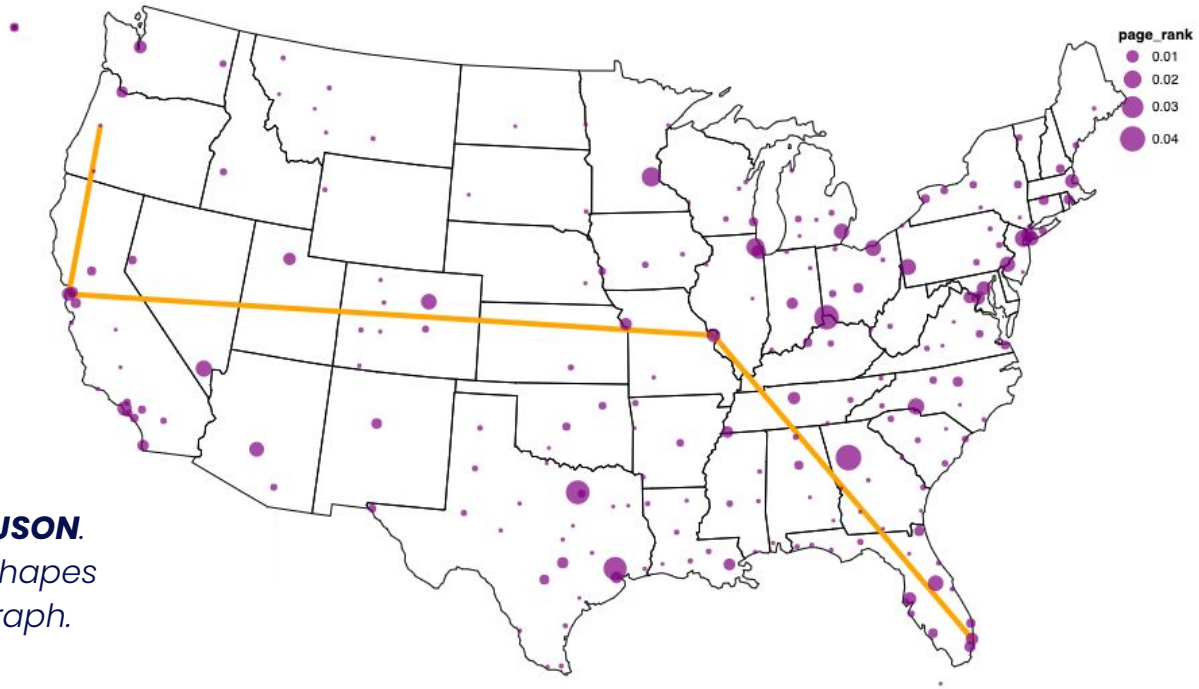
RelationalAI

# Graph Analytics

Rel can express graph algorithms, for example **pagerank** and **shortest path**.

Shown: pagerank for major airports

Highlighted is a shortest path between two nodes.




*Rel supports **geographical data** and **JSON**. The maps are computed in Rel from shapes of the states, part of the knowledge graph. Visualization is Vega-Lite.*

# Feature Engineering: Describe

Similar to Dataframes, **`describe`**, **implemented in Rel**, generically reports statistics for a collection of relations.

```
describe[airport]
```

| | Elevation | State | Facility | .... | |
|---|---|---|---|---|---|
| min | -210 | AK | AIRPORT | | *(Furnace Creek, CA)* |
| max | 12,442 | WY | ULTRALIGHT | | *(Berthoud Pass, CO)* |
| mean | 1,143 | | | | |
| std | 1,444 | | | | |
| 25% | 270 | | | | |
| 50% | 745 | | | | |
| 75% | 1,220 | | | | |
| unique | | 58 | 7 | | |
| mode | | TX | AIRPORT | | |

```
describe[t: Airport <: airport[t]]
```

| | Elevation | ... | |
|---|---|---|---|
| max | 9,927 | | *(Lake County, CO)* |

# Relational Machine Learning

## Generic models

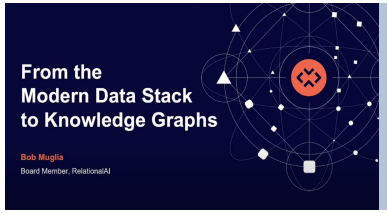
*This is in a reusable library. Note this uses Rel schema abstraction (features is schema)*

```
def predict_linear[X, M][k...] = sum[f: M[f] * X[f, k...]] + sum[f: M[f, X[f, k...]]] + M[:bias]

def mse[Yhat, Y] = sum[x: (Y[x] - Yhat[x]) ^ 2] / count[Y]

def rmse[Yhat, Y] = sqrt[mse[Yhat, Y]]

def linear_regression[X, Y, M] = minimize[rmse[predict_linear[X, M], Y]]
```

## Application-specific instantiation

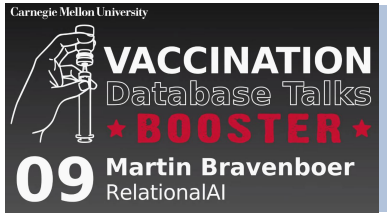
```
def features:hour_of_day = arrival_time.hour_of_day
def response = arrival_delay

def model = linear_regression[features, response, initial_point]
```

# Learn More




**"KGC Bob Muglia" for modern data stack and relational knowledge graph**

Youtube




**"CMU RelationalAI" for RAI system overview**

Youtube


**"DSDSD Bravenboer" for different RAI system overview**

Youtube

https://twitter.com/RelationalAI


# Thank you!