The next-generation database system for intelligent data apps based on relational knowledge graphs.
Innovations for Relational Knowledge Graphs

1. Immutability - Cloud native architecture
2. Expressive relational language (Rel)
3. Join algorithms
4. Semantic optimization
5. Vectorized and JIT compilation of WCOJ
6. Live - Incrementality (for data and logic)
Challenges in Database System Design and Implementation

Data structures and memory management
- In-memory performance for modern workloads exceeding available memory and disk
- Write-optimized data structures for modern workloads in cloud native architecture

Query processing
- Index selection (what indexes to define for a workload)
- Efficient evaluation of subqueries
- Relational query processing of graph workloads (complex joins)
- Materialized view selection (with views to materialize for a workload)
- Incremental computation (recursion) and maintenance wrt input changes

Concurrency and workload management
- Optimization of bottom-up vs top-down (demand-driven) evaluation
- Optimization of very large computation graphs
- Strong consistency, scalability of read-only and write workload

General Architecture
- Eliminate the split brain: moving computations to the data management system
- Maximal independence of application logic vs machine representation and organization of data (relational model)
- Language support for abstraction (libraries)
- Language support for schema abstraction (generic programming)
Dependency Graph of Tax Analysis Logic
3.6K relations, 13K dependencies
replacing millions of lines of procedural code
Dependency Graph of Tax Analysis Logic
Focus: Single strongly-connected component (recursion)
The Modern Data Stack
Modern database systems are cloud native

Modern database systems are implemented with cloud native architecture that separates storage from compute.

This architecture makes it possible to provide compelling features like:

- **Infinite storage** - store all your data regardless of structure or volume
- **Infinite compute** - run any number of workloads without concurrency limits
- **Versioning** - time-travel, zero-copy cloning
- **Fully managed** - workload management with minimal user intervention
- **Data sharing** - collaboration, live sharing, access to external data
Cloud Data Platform
(warehouse, lakehouse)

The Modern Data Stack

System of Record

Data Apps

BI Tools

ML Feature Engineering

Notebooks
The Semantic Layer
<table>
<thead>
<tr>
<th>carrier</th>
<th>origin</th>
<th>destination</th>
<th>flight_num</th>
<th>flight_time</th>
<th>tall_num</th>
<th>dep_time</th>
<th>dep_delay</th>
<th>arr_time</th>
<th>arr_delay</th>
<th>taxi_out</th>
<th>taxi_in</th>
<th>distance</th>
<th>cancelled</th>
<th>dverted</th>
<th>id2</th>
</tr>
</thead>
<tbody>
<tr>
<td>F9</td>
<td>MCI</td>
<td>DEN</td>
<td>818</td>
<td>89</td>
<td>N916FR</td>
<td>2005-09-26 08:23:00 UTC</td>
<td>-7</td>
<td>-8</td>
<td>9</td>
<td>6</td>
<td>533</td>
<td>N</td>
<td>N</td>
<td>36606834</td>
<td></td>
</tr>
<tr>
<td>WN</td>
<td>LBB</td>
<td>ELI</td>
<td>819</td>
<td>41</td>
<td>N708SW</td>
<td>2000-08-18 07:30:00 UTC</td>
<td>0</td>
<td>-5</td>
<td>7</td>
<td>2</td>
<td>295</td>
<td>N</td>
<td>N</td>
<td>4569021</td>
<td></td>
</tr>
<tr>
<td>NW</td>
<td>ATL</td>
<td>MEM</td>
<td>819</td>
<td>52</td>
<td>N609NW</td>
<td>2001-11-16 07:15:00 UTC</td>
<td>-5</td>
<td>-9</td>
<td>19</td>
<td>3</td>
<td>320</td>
<td>N</td>
<td>N</td>
<td>11838308</td>
<td></td>
</tr>
<tr>
<td>DL</td>
<td>SLC</td>
<td>BOI</td>
<td>819</td>
<td>48</td>
<td>N296WA</td>
<td>2001-12-04 22:12:00 UTC</td>
<td>7</td>
<td>-4</td>
<td>40</td>
<td>4</td>
<td>291</td>
<td>N</td>
<td>N</td>
<td>12065416</td>
<td></td>
</tr>
<tr>
<td>WN</td>
<td>PHX</td>
<td>SAN</td>
<td>819</td>
<td>51</td>
<td>N591SW</td>
<td>2001-12-05 09:11:00 UTC</td>
<td>11</td>
<td>304</td>
<td>N</td>
<td>132383068</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WN</td>
<td>LAS</td>
<td>AUS</td>
<td>819</td>
<td>135</td>
<td>N619SW</td>
<td>2002-04-05 08:18:00 UTC</td>
<td>8</td>
<td>1085</td>
<td>N</td>
<td>13762927</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WN</td>
<td>SJC</td>
<td>LAS</td>
<td>819</td>
<td>63</td>
<td>N528SW</td>
<td>2007-04-02 06:30:00 UTC</td>
<td>0</td>
<td>-3</td>
<td>11</td>
<td>3</td>
<td>386</td>
<td>N</td>
<td>N</td>
<td>15284777</td>
<td></td>
</tr>
<tr>
<td>WN</td>
<td>LAS</td>
<td>AUS</td>
<td>819</td>
<td>137</td>
<td>N502SW</td>
<td>2002-09-16 08:20:00 UTC</td>
<td>0</td>
<td>-8</td>
<td>11</td>
<td>4</td>
<td>1085</td>
<td>N</td>
<td>N</td>
<td>16027516</td>
<td></td>
</tr>
<tr>
<td>DL</td>
<td>MSP</td>
<td>SLC</td>
<td>819</td>
<td>149</td>
<td>N3754A</td>
<td>2002-09-29 19:34:00 UTC</td>
<td>0</td>
<td>-15</td>
<td>25</td>
<td>7</td>
<td>991</td>
<td>N</td>
<td>N</td>
<td>16399211</td>
<td></td>
</tr>
<tr>
<td>DL</td>
<td>MSP</td>
<td>SLC</td>
<td>819</td>
<td>145</td>
<td>N3745B</td>
<td>2002-09-29 19:27:00 UTC</td>
<td>0</td>
<td>-9</td>
<td>18</td>
<td>6</td>
<td>991</td>
<td>N</td>
<td>N</td>
<td>17417961</td>
<td></td>
</tr>
<tr>
<td>WN</td>
<td>SJC</td>
<td>LAS</td>
<td>819</td>
<td>54</td>
<td>N730SW</td>
<td>2003-06-26 06:30:00 UTC</td>
<td>0</td>
<td>-11</td>
<td>15</td>
<td>5</td>
<td>386</td>
<td>N</td>
<td>N</td>
<td>20460576</td>
<td></td>
</tr>
<tr>
<td>WN</td>
<td>SJC</td>
<td>LAS</td>
<td>819</td>
<td>60</td>
<td>N501SW</td>
<td>2003-09-15 06:30:00 UTC</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>2</td>
<td>386</td>
<td>N</td>
<td>N</td>
<td>22113592</td>
<td></td>
</tr>
<tr>
<td>NW</td>
<td>ATL</td>
<td>MEM</td>
<td>819</td>
<td>51</td>
<td>N785NC</td>
<td>2003-11-20 07:11:00 UTC</td>
<td>0</td>
<td>-23</td>
<td>10</td>
<td>3</td>
<td>332</td>
<td>N</td>
<td>N</td>
<td>23835354</td>
<td></td>
</tr>
<tr>
<td>DL</td>
<td>MSP</td>
<td>ATL</td>
<td>819</td>
<td>120</td>
<td>N996DE</td>
<td>2004-02-10 09:59:00 UTC</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>7</td>
<td>906</td>
<td>N</td>
<td>N</td>
<td>25206893</td>
<td></td>
</tr>
<tr>
<td>US</td>
<td>CYH</td>
<td>CLT</td>
<td>820</td>
<td>38</td>
<td>N592US</td>
<td>2002-07-01 19:32:00 UTC</td>
<td>37</td>
<td>64</td>
<td>9</td>
<td>35</td>
<td>166</td>
<td>N</td>
<td>N</td>
<td>15142411</td>
<td></td>
</tr>
<tr>
<td>FL</td>
<td>TPA</td>
<td>ATL</td>
<td>820</td>
<td>64</td>
<td>N555AT</td>
<td>2003-01-31 11:29:00 UTC</td>
<td>6</td>
<td>-5</td>
<td>13</td>
<td>9</td>
<td>406</td>
<td>N</td>
<td>N</td>
<td>17349259</td>
<td></td>
</tr>
<tr>
<td>WN</td>
<td>RDU</td>
<td>PHL</td>
<td>820</td>
<td>73</td>
<td>N382SW</td>
<td>2004-12-02 11:10:00 UTC</td>
<td>0</td>
<td>-19</td>
<td>5</td>
<td>3</td>
<td>336</td>
<td>N</td>
<td>N</td>
<td>30796766</td>
<td></td>
</tr>
<tr>
<td>HP</td>
<td>PHX</td>
<td>BNA</td>
<td>820</td>
<td>0</td>
<td>N826AW</td>
<td>2005-10-25 00:00:00 UTC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2300</td>
<td>Y</td>
<td>N</td>
<td>37174931</td>
<td></td>
</tr>
<tr>
<td>US</td>
<td>PBI</td>
<td>CLT</td>
<td>821</td>
<td>90</td>
<td>N642AU</td>
<td>2002-05-18 06:35:00 UTC</td>
<td>5</td>
<td>-8</td>
<td>10</td>
<td>6</td>
<td>590</td>
<td>N</td>
<td>N</td>
<td>14247614</td>
<td></td>
</tr>
<tr>
<td>US</td>
<td>PBI</td>
<td>CLT</td>
<td>821</td>
<td>87</td>
<td>N14SW</td>
<td>2002-05-18 07:43:00 UTC</td>
<td>7</td>
<td>-7</td>
<td>16</td>
<td>7</td>
<td>590</td>
<td>N</td>
<td>N</td>
<td>20209351</td>
<td></td>
</tr>
<tr>
<td>DL</td>
<td>GSO</td>
<td>CVG</td>
<td>821</td>
<td>70</td>
<td>N417SW</td>
<td>2005-07-08 12:02:00 UTC</td>
<td>1</td>
<td>11</td>
<td>14</td>
<td>7</td>
<td>330</td>
<td>N</td>
<td>N</td>
<td>21396171</td>
<td></td>
</tr>
</tbody>
</table>
The Semantic Layer and Data Apps

Let's build a data app for an order database (TPC-H, Northwind etc)

Example functionality:

- What is the average charge of orders by week
- What percentage of orders were late this year
- If two consecutive orders for a customer are late, alert the account manager

The system cannot answer such questions if it does not know what late and charge mean to begin with!
Suppose, for example, that we hand you a piece of paper with this short passage:

Two children, Chloe and Alexander, went for a walk. They both saw a dog and a tree. Alexander also saw a cat and pointed it out to Chloe. She went to pet the cat.

It is trivial to answer questions like “Who went for a walk?,” in which the answer (“Chloe and Alexander”) is directly spelled out in the text, but any competent reader should just as easily be able to answer questions that are not directly spelled out, like “Did Chloe see the cat?” and “Were the children frightened by the cat?” If you can’t do that, you aren’t really following the story. Because SQuAD didn’t include any questions of this sort, it wasn’t really a strong test of reading; as it turns out the new AI systems would not have been able to cope with them. By way of contrast, Gary tested the story on his daughter Chloe, then four and a half years old, and she had no trouble making the inference that the fictitious Chloe had seen a cat. (Her older brother, then not quite six years old, went a step further, musing about what would happen if the dog actually turned out to be a cat; no current AI could begin to do that.)
How many movies has Meryl Streep been in per decade

<table>
<thead>
<tr>
<th>count</th>
<th>decade</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1970</td>
</tr>
<tr>
<td>11</td>
<td>1980</td>
</tr>
<tr>
<td>12</td>
<td>1990</td>
</tr>
<tr>
<td>15</td>
<td>2010</td>
</tr>
<tr>
<td>22</td>
<td>2000</td>
</tr>
</tbody>
</table>

What movies has Johnny Depp acted in since 2015

<table>
<thead>
<tr>
<th>movie_id</th>
<th>title</th>
<th>release_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>168426</td>
<td>Pirates of the Caribbean: Dead Men Tell No Tales</td>
<td>2017-05-23</td>
</tr>
<tr>
<td>241259</td>
<td>Alice Through the Looking Glass</td>
<td>2016-05-25</td>
</tr>
<tr>
<td>259316</td>
<td>Fantastic Beasts and Where to Find Them</td>
<td>2016-11-16</td>
</tr>
<tr>
<td>290825</td>
<td>Yoga Hosers</td>
<td>2016-07-08</td>
</tr>
<tr>
<td>382079</td>
<td>Donald Trump's The Art of the Deal: The Movie</td>
<td>2016-02-10</td>
</tr>
</tbody>
</table>

Showing all 5 results
dimension: is_order_paid {
    type: yesno
    sql: ${status} = 'paid' ;;
}

dimension: full_name {
    type: string
    sql: CONCAT(${first_name}, ' ', ${last_name}) ;;
}

dimension: profit {
    type: number
    sql: ${revenue} - ${cost} ;;
}

dimension: distance_to_pickup {
    type: distance
    start_location_field: customer.home_location
    end_location_field: rental.pickup_location
    units: miles
}

dimension: store_location {
    type: location
    sql_latitude: ${store_latitude} ;;
    sql_longitude: ${store_longitude} ;;
}

measure: cumulative_total_revenue {
    type: running_total
    sql: ${total_sale_price} ;;
}

measure: total_gross_margin {
    type: sum
    value_format_name: usd
    sql: ${gross_margin} ;;
}

measure: percent_of_total_gross_margin {
    type: percent_of_total
    sql: ${total_gross_margin} ;;
}

https://docs.looker.com/reference
source: users is table('malloy-data.ecomm.users') {
  primary_key: id
  dimension: full_name is concat(first_name, ' ', last_name)
  measure: user_count is count()
}

source: iowa is table('malloy-data.iowa_liquor_sales.sales_deduped') {
  dimension: gross_margin is 100 * (state_bottle_retail - state_bottle_cost) / nullif(state_bottle_retail, 0)
  dimension: price_per_100ml is state_bottle_retail / nullif(bottle_volume_ml, 0) * 100
}

source: flights is table('malloy-data.faa.flights') {
  dimension: distance_km is distance / 1.609344
  measure: flight_count is count()
  rename: destination_code is destination
}
customer_orders as (  
    select  
    customer_id,  
    min(order_date) as first_order,  
    max(order_date) as most_recent_order,  
    count(order_id) as number_of_orders  
    from orders  
    group by customer_id)

gitlab_dotcom_issues_source AS (  
    SELECT *  
    FROM {{ ref('gitlab_dotcom_issues_source')}}  
    {% if is_incremental() %}  
    WHERE updated_at >= (SELECT MAX(updated_at) FROM {{this}})  
    {% endif %})

upvote_count AS (  
    SELECT  
    awardable_id  
    SUM(IF(award_emoji_name LIKE 'thumbsup%', 1, 0)) AS thumbsups_count,  
    SUM(IF(award_emoji_name LIKE 'thumbsdown%', 1, 0)) AS thumbsdowns_count,  
    thumbsups_count - thumbsdowns_count AS upvote_count  
    FROM gitlab_dotcom_award_emoji_source  
    WHERE awardable_type = 'Issue'  
    GROUP BY 1)

order_payments as (  
    select  
    order_id,  
    {% for payment_method in payment_methods -%}  
    sum(case when payment_method = '{{ payment_method }}'  
    then amount else 0 end  
    ) as {{ payment_method }}_amount,  
    {% endfor -%}  
    sum(amount) as total_amount  
    from payments  
    group by order_id)
Data Apps, Reasoning & Knowledge

Views / Reasoning / Knowledge / The Semantic Layer
The Semantic Layer – Rel
Let's build a data app for an order database (TPC-H, Northwind etc)

Example functionality:

- What is the average charge of orders by week
- What percentage of orders were late this year
- If two consecutive orders for a customer are late, alert the account manager

The system cannot answer such questions if it does not know what late and charge mean to begin with!
Data Apps, Reasoning & Knowledge

Given: extendedprice, discount, tax

```python
def item_revenue[o, num] =
    extendedprice[o, num] * (1 - discount[o, num])

def revenue[o] =
    sum[num: item_revenue[o, num]]

def item_charge[o, num] =
    item_revenue[o, num] * (1 + tax[o, num])

def charge[o] =
    sum[num: item_charge[o, num]]
```
Given: commitdate, receiptdate

```python
def received_late(o, num):
    commitdate[o, num] < receiptdate[o, num]

def late(o):
    exists(num: received_late(o, num))
```
| 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 07:10:00 UTC | -5        | -9        | 19       | 3       | 332      | N         | N         | 11838808 |
| DL      | SLC    | BOI         | 819        | 48          | N296WA   | 2001-12-01 22:12:00 UTC | 2001-12-01 23:53:00 UTC | 7         | -4        | 40       | 4       | 291      | N         | N         | 12058416 |
| WN      | PHX    | SAN         | 819        | 51          | N9115W   | 2001-12-05 09:11:00 UTC | 2001-12-05 09:17:00 UTC | 11        |          |          |          |          | N         | N         | 12383068 |
| WN      | LAS    | AUS         | 819        | 135         | N519SW   | 2002-04-05 08:18:00 UTC | 2002-04-05 12:47:00 UTC | 8         |          |          |          |          | N         | N         | 13762927 |
| WN      | SJC    | LAS         | 819        | 63          | N529SW   | 2002-07-14 06:30:00 UTC | 2002-07-14 07:47:00 UTC | 0         | -3        | 11       | 3       | 386      | N         | N         | 15294777 |
| 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         | N7544A   | 2002-05-29 19:34:00 UTC | 2002-05-29 21:35:00 UTC | 9         | 15        | 25       | 7       | 991      | N         | N         | 16399211 |
| DL      | MSP    | SLC         | 819        | 145         | N7545B   | 2002-05-29 19:27:00 UTC | 2002-05-29 21:16:00 UTC | -3        | -9        | 18       | 6       | 991      | N         | N         | 17417961 |
| WN      | SJC    | LAS         | 819        | 54          | N730SW   | 2003-06-06 06:30:00 UTC | 2003-06-06 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 |
| 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         | 23838534 |
| DL      | MSP    | ATL         | 819        | 120         | N905DE   | 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   | 2004-07-01 19:32:00 UTC | 2004-07-01 20:54:00 UTC | 37        | 64        | 9        | 35      | 166      | N         | N         | 15142411 |
| FL      | TPA    | ATL         | 820        | 64          | N555AT   | 2003-03-31 11:29:00 UTC | 2003-03-31 12:55:00 UTC | -6        | -5        | 13       | 9       | 406      | N         | N         | 17940929 |
| WN      | RDU    | PHL         | 820        | 73          | N832SW   | 2004-12-02 11:10:00 UTC | 2004-12-02 12:31:00 UTC | 0         | -19       | 5        | 3       | 336      | N         | N         | 30797666 |
| HP      | PHX    | BOS         | 820        | 0           | N826SW   | 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 08:12:00 UTC | -5        | -8        | 10       | 6       | 590      | N         | N         | 14247614 |
| US      | PBI    | CLT         | 821        | 87          | N624US   | 2002-05-18 06:35:00 UTC | 2002-05-18 08:12:00 UTC | -7        | -7        | 16       | 7       | 590      | N         | N         | 20269351 |
| DL      | GSO    | CVG         | 821        | 70          | N417DR   | 2003-07-08 07:43:00 UTC | 2003-07-08 10:09:00 UTC | 1         | 11        | 14       | 7       | 330      | N         | N         | 21396171 |

- **Primary key**: id2
- **Include helicopters**: Not applicable
- **Possible values**: Risk of messing up aggregates
- **Are these exclusive**: Not applicable
- **Not a delay**: Yes
Better Conceptual Model

```python
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[\text{maximum}[0, arr\_delay[f]]]\}

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

def airport_distance[a1 in Airport, a2 in Airport] =
    distance[coordinate[a1], coordinate[a2]]
```
Reasoning manages app logic with the data

**Reasoning** subsumes business logic now implemented procedurally in languages like Java, C#, Python, Scala, PL/SQL, T/SQL etc.

Fixing the “split brain” problem where the data is managed in one layer and knowledge/semantics in another will have huge impact.

Bringing the app logic to the data makes it possible for one (cloud native) system to manage the semantics, integrity, and resources needed for the application.

With thanks to Peter Bailis...
Relational Models
Directed Graphs as a Relation

edge(2, 1)
edge(2, 4)
edge(3, 1)
edge(3, 2)
edge(3, 4)
Labelled Property Graphs as Relational Graphs

Actor
name: Chalamet
id: 1
role: Paul Atreides

Movie
title: Dune
year: 2021
id: 3

Director
name: Villeneuve
id: 2
directed: 2

Writer
name: Villeneuve
id: 2
writer: 2

Movie
movie(3)
title(3, "Dune")
year(3, 2021)
director(2)
writer(2)
name(2, "Villeneuve")
directed(2, 3)

Actor
name: Chalamet
id: 1
acted(1, 3)
role(1, 3, "Paul Atreides")
Tables as a Collection of Relations

<table>
<thead>
<tr>
<th>orderkey</th>
<th>customer</th>
<th>date</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>500</td>
<td>2022-03-27</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>2022-03-27</td>
<td>43</td>
</tr>
</tbody>
</table>

customer(1, 500)
customer(2, 23)
date(1, 2022-03-27)
date(2, 2022-03-27)
price(1, 75)
price(2, 43)

SQL tables are in a sense a modularity construct, grouping relations with the same primary key.
Recall ...

Tensors are relations!

\[
\begin{bmatrix}
1 & 2 & 0 \\
0 & 1 & 0 \\
2 & 0 & 1
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Tensors as Relations

A relational database system that is effective for tensors would be an outstanding proof-point for the relational model.

(and imagine the data management benefits this would have for ML systems!)
Tensors as Relations: Matrix Multiplication

**Math**

\[ c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \]

**Rel**  
*Our new relational language*

```python
def C[i, j] = sum[k: A[i, k] * B[k, j]]
```

**SQL**

```sql
SELECT A.row, B.col, SUM(A.val * B.val)
FROM A INNER JOIN B ON A.col = B.row
GROUP BY A.row, B.col
```
The Essence of the Relational Model

Have relational database systems been sufficiently ambitious on this point.
Architecture
Cloud Native Deployment Architecture

Cloud Region

- **Engine**
  - Transient
  - RAM, SSD cache

- **Engine**
  - Transient
  - RAM, SSD cache

- **Serverless Engine**
  - Transient
  - RAM, SSD cache

- **Scalable, durable object storage**
  - Immutable, versioned, write-optimized, paged data structures

- **CAS Key-Value Store**
  - (Only database root pointers)

RAI SDK (Python, Julia, JS, Go, Java, C#)

Services (JSON, Arrow)

VSCode  RAI Console  RAI CLI  Data Apps  SQL Apps  Legend Apps
Coexist as One Happy Relational Family

Core Rel IR

Relational Knowledge Graph System

Future

GQL

SPARQL

GraphQL

Rel

SQL

Legend

CSV

RDF

SQL RDBMS CDC

Binary objects

Tensor data

Parquet, Iceberg

LPG

JSON

LLVM

julia

Specialized Solvers
Internal Engine Architecture

Rel Model

Parse

Dependency analysis

Type Inference

Specialization to first-order logic

Evaluation (vectorized + JIT)

Physical optimization

Semantic optimization

Dependency analysis

Metadata database (Salsa + Arroyo)

Demand-driven computation and provenance for incrementality and live programming

JuliaCon 2020 – Salsa.jl – Nathan Daly
Core Innovations for Relational Knowledge Graphs

Immutable Data Structures for Cloud Object Storage
RAI Storage and Memory Management
(inspired by Snowflake and Umbra/Leanstore)

Scalable, durable object storage

Ephemeral SSD cache

RAM cache (buffer pool)

fetch and evict

fetch
evict
commit
RAI databases are immutable, including the catalog
RAI databases are immutable, including the catalog

key/value store with CAS

demo
RAI databases are immutable, including the catalog

key/value store with CAS
- demo-2022-03-25
- demo

transaction updates C

rel A
rel B
rel C
rel C'
RAI databases are immutable, including the catalog

<table>
<thead>
<tr>
<th>key/value store with CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>demo-2022-03-25</td>
</tr>
<tr>
<td>demo</td>
</tr>
</tbody>
</table>
RAI databases are immutable, including the catalog.
Key: immutable tables → immutable catalog

Isolation: strict serializability
- **Must**: Anything weaker causes inconsistencies for data apps *(depending on lock granularity)*
- No locks need to be acquired *(concurrent writes can be executed optimistically)*
- Effectively unlimited read scalability
- No limit on the duration of a transaction

DDL is atomic
- **Must**: Schema changes are common in data apps and live programming
- Cloning a database is an atomic O(1) operation
- Perfect for as-of (system time) queries, what-if analysis

Write-optimized data structures 💕 immutable object storage
- **Must**: Removing write amplification is critical for object storage *(Bε-tree)*
- Group commits and variable page sizes to reduce write throughput needs

No transaction log is needed for durability or recovery
- Previous version immutable. Commits atomic in KV store *(CAS)*
Storage Management: Influences and Resources

Elastic Storage Management

- The Snowflake Elastic Data Warehouse
  Dageville et al., SIGMOD 2016
- Building an Elastic Query Engine on Disaggregated Storage
  Vuppalapati et al., NSDI 2020

Write Optimization

- Lower Bounds for External Memory Dictionaries
  Brodal et al., SODA 2003
- An Introduction to Bε -trees and Write-Optimization
  Bender et al., :login: magazine, 2015
- Design and Implementation of the LogicBlox System
  Aref et al. SIGMOD 2015

In-Memory Performance

- LeanStore: In-Memory Data Management Beyond Main Memory
  Leis et al., ICDE 2018
- Umbra: A Disk-Based System with In-Memory Performance
  Neumann et al., CIDR 2020
Core Innovations for Relational Knowledge Graphs

Rel
A Productive and Expressive Relational Language
Datalog and First-order Logic

Transitive closure

\[
\text{ancestor}(x, y) :- \text{parent}(x, y)
\]
\[
\text{ancestor}(x, y) :- \text{parent}(x, t) \text{ and ancestor}(t, y)
\]
\[
\text{reachable}(x, y) :- \text{edge}(x, y)
\]
\[
\text{reachable}(x, y) :- \text{edge}(x, t) \text{ and reachable}(t, y)
\]

Functional dependency

\[
\text{function_age}() \ :- \textit{forall}(x, v, w:\text{age}(x, v) \text{ and age}(x, w) \implies v = w)
\]
\[
\text{function_name}() \ :- \textit{forall}(x, v, w:\text{name}(x, v) \text{ and name}(x, w) \implies v = w)
\]
\[
\text{function_address}() \ :- \textit{forall}(x, v, w:\text{address}(x, v) \text{ and address}(x, w) \implies v = w)
\]

Average

\[
\text{average_sales}(x, y, w) \ :- \text{sum_sales}(x, y, u) \text{ and count_sales}(x, y, v) \text{ and } w = u / v
\]
\[
\text{average_returns}(x, y, w) \ :- \text{sum_returns}(x, y, u) \text{ and count_returns}(x, y, v) \text{ and } w = u / v
\]
Datalog

Good

- Outstanding formal foundation
- Mutually recursive definitions

More is needed

- Classic Datalog (globally stratified) is too limited for graph workloads:
  - Value creation in recursion
  - Aggregation in recursion
  - Negation in recursion

- Datalog does not support abstraction (similar to SQL, Cypher, SPARQL etc)
  - Abstract over concrete relations
  - Abstract over schema

Rel: Datalog is the IR
# Rel – Design Objectives

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small core</td>
<td>Designed for growth: whole is greater than sum of the parts</td>
</tr>
<tr>
<td>Declarative</td>
<td>Maximize opportunities for executing programs in different ways</td>
</tr>
<tr>
<td>Relational</td>
<td>Data independence <em>(representation, ordering, semantic stability)</em></td>
</tr>
</tbody>
</table>
| Abstraction               | Libraries of reusable functionality *(eg statistics, graph analytics)*  
                             | Encourage an ecosystem of reusable components |
| Abstraction without regret| Aggressive optimizations to compile abstraction cost aways. |
| Schema abstraction        | Logically treating schema as data to support schema-generic logic  
                             | Prevent the need for code generators  
                             | Support interactive schema discovery *(reflection)* |
| Live programming          | Support arbitrary schema changes  
                             | Ingest data without upfront schema into an efficient representation  
                             | Incorrect application logic is a valid state  
                             | Support gradually enforcing a schema with integrity constraints |
| Carrier | Origin | Destination | Flight_num | Flight Time | Tall Num | Dep Time | Arr Time | Dep Delay | Arr Delay | Taxi Out | Taxi In | Distance | Cancelled | Dverted | 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       | 36608634 |
| WN      | LBB    | ELI         | 819        | 41          | N708SW   | 2005-09-26 07:00:00 UTC | 2005-09-26 07:20:00 UTC | 0         | -5        | 7        | 2       | 295     | N        | N       | 4369021  |
| NW      | ATL    | MEM         | 819        | 52          | N697NW   | 2011-11-16 07:15:00 UTC | 2011-11-16 07:29:00 UTC | -5        | -9        | 19       | 3       | 332     | N        | N       | 11838308 |
| DL      | SLG    | BOI         | 819        | 48          | N296WA   | 2001-12-02 22:12:00 UTC | 2001-12-03 07:53:00 UTC | 7         | 4         | 40       | 4       | 291     | N        | N       | 12063416 |
| WN      | PHX    | SAN         | 819        | 51          | N9115W   | 2001-12-05 09:11:00 UTC | 2001-12-05 09:17:00 UTC | 11        | 304       |          |         |         | N        | N       | 12383058 |
| WN      | LAS    | AUS         | 819        | 135         | N619SW   | 2002-04-05 08:18:00 UTC | 2002-04-05 12:47:00 UTC | 8         | 1085      |          |         |         | N        | N       | 13762979 |
| WN      | SJC    | LAS         | 819        | 63          | N529SW   | 2002-07-14 06:30:00 UTC | 2002-07-14 07:47:00 UTC | 0         | 113       | 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         | N754LA   | 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         | N7545B   | 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 |
| WN      | ATC    | 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       | 23838354 |
| DL      | MSP    | ATL         | 819        | 120         | N990DE   | 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   | 2005-07-01 19:32:00 UTC | 2005-07-01 20:54:00 UTC | 37        | 64        | 9        | 35      | 168     | N        | N       | 15142411 |
| FL      | TPA    | ATL         | 820        | 64          | N555AT   | 2005-03-31 11:29:00 UTC | 2005-03-31 12:55:00 UTC | -6        | -5        | 13       | 9       | 406     | N        | N       | 17349329 |
| WN      | RDU    | PHI         | 820        | 73          | N382SW   | 2004-12-02 11:10:00 UTC | 2004-12-02 12:31:00 UTC | 0         | -19       | 5        | 3       | 336     | N        | Y       | 30796766 |
| HP      | PHX    | BOS         | 820        | 0           | N826AW   | 2005-10-25 09:00:00 UTC | 2005-10-25 23:00:00 UTC | 0         | 0         | 0        | 0       | 2300    | Y        | N       | 37174931 |
| US      | PBI    | CLT         | 821        | 90          | N624AU   | 2005-05-18 06:35:00 UTC | 2005-05-18 08:12:00 UTC | -5        | -8        | 10       | 6       | 590     | N        | Y       | 14247614 |
| US      | PBI    | CLT         | 821        | 87          | N624AS   | 2005-05-18 06:35:00 UTC | 2005-05-18 08:43:00 UTC | 7         | -7        | 16       | 7       | 590     | N        | N       | 20289351 |
| DL      | GSO    | CVG         | 821        | 70          | N947SV   | 2005-05-18 06:35:00 UTC | 2005-05-18 08:12:00 UTC | 1         | 11        | 14       | 7       | 330     | N        | Y       | 21398171 |
**Better Conceptual Model**

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

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

def origin(f in Flight, a in Airport) =
    flight-origin(f, code) and
    airport_code(a, code)
    from code

def destination(f in Flight, a in Airport) =
    flight-destination(f, code) and
    airport_code(a, code)
    from code

def airport_distance[a1 in Airport, a2 in Airport] =
    distance[coordinate[a1], coordinate[a2]]

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

Nodes involved in relationships

\[
\text{ic \, forall}(f, \, ap: \, \text{origin}(f, \, ap) \, \text{implies} \, \text{Flight}(f) \, \text{and} \, \text{Airport}(ap))
\]

Required relationships

\[
\text{ic \, forall}(f: \, \text{Flight}(f) \, \text{implies} \, \text{exists} \, \text{origin}[f])
\]

Functional dependency (flight can have only one origin)

\[
\text{ic \, forall}(x, \, v, \, w: \, \text{origin}(x, \, v) \, \text{and} \, \text{origin}(x, \, w) \, \text{implies} \, v = w)
\]

Arbitrarily complex

\[
\text{ic \, forall}(f \, \text{in} \, \text{cancelled:} \, \text{not} \, \text{exists} \, \text{flight_duration}[f])
\]
\[
\text{ic \, forall}(f \, \text{in} \, \text{flight:} \, \text{cancelled}(f) \, \text{xor} \, \text{diverted}(f) \, \text{xor} \, \text{arrived}(f))
\]
Aggregation

Total number of flights
\[
\text{count[Flight]}
\]

Carrier with most flights
\[
\text{c: count[f: operated_by(f, c)]}
\]

Carriers mean arrival delay
\[
\text{c: mean[f.arrival_delay for f where operated_by(f, c)]}
\]

Airport ratio of cancelled arriving flights
\[
\text{ap: ratio[cancelled, ap.arriving_flight]}
\]

Southwest: 5,775,777
Delta: 4,477,929
American: 4,434,727
Airtran: 15 min
Atlantic Coast: 13 min
United Airlines: 13 min
... Aloha Airlines: 6 min
Hawaiian Airlines: 3 min
Unalaska: 19%
Worcester Regional: 11%
Nantucket Memorial: 9%
Abstraction and Value Types

Recall from the model

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

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

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

Units of measurements to prevent miscalculation

```python
def LengthUnit = :Feet; :Meters; :Miles; :Kilometers

value type Length = LengthUnit, Number
value type Degree = Number
value type LLA = Degree, Degree, Length

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

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

The type system of Rel prevents a runtime cost of tracking units of measurement.

Statically Rel guarantees that the correct conversions are applied and no incompatible values can be used in operations.
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

\[ \text{count}[x, \, v: \text{flight}\_\text{graph}(x, \, v)] \]

Count all nodes, grouped by type

\[ x: \text{count}[v: \text{flight}\_\text{graph}(x, \, v)] \]

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flight</td>
<td>37,561,525</td>
</tr>
<tr>
<td>Aircraft</td>
<td>359,928</td>
</tr>
<tr>
<td>AircraftModel</td>
<td>60,461</td>
</tr>
<tr>
<td>City</td>
<td>50,944</td>
</tr>
<tr>
<td>Airport</td>
<td>19,793</td>
</tr>
<tr>
<td>Heliport</td>
<td>5,135</td>
</tr>
<tr>
<td>County</td>
<td>3,009</td>
</tr>
<tr>
<td>Major</td>
<td>270</td>
</tr>
<tr>
<td>State</td>
<td>58</td>
</tr>
<tr>
<td>Carrier</td>
<td>21</td>
</tr>
</tbody>
</table>

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.
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.
Basic graph algorithms

Neighbor (undirected edge)

\[
def \text{neighbor}(x, y) = \text{edge}(x, y) \text{ or } \text{edge}(y, x)
\]
\[
def \text{cn}[x, y] = \text{count}[\text{intersect}[\text{neighbor}[x], \text{neighbor}[y]]]
\]

Degree

\[
def \text{outdegree}[x] = \text{count}[\text{edge}[x]]
\]
\[
def \text{degree}[x] = \text{count}[\text{neighbor}[x]]
\]

Similarity

\[
def \text{cosine_sim}[x, y] = \frac{\text{cn}[x, y]}{\sqrt{\text{degree}[x] \times \text{degree}[y]}}
\]
\[
def \text{jaccard_sim}[x, y] = \frac{\text{cn}[x, y]}{\text{count}[\text{neighbor}[x]] + \text{count}[\text{neighbor}[y]] - \text{cn}[x, y]}
\]

Transitive closure (reachability)

\[
def \text{reachable}(x, y) = \text{edge}(x, y)
\]
\[
def \text{reachable}(x, y) = \exists (t: \text{edge}(x, t) \text{ and } \text{reachable}(t, y))
\]
Basic graph algorithms

Weakly connected components

```python
def wcc[x] = min[reachable_undirected[x]]
```

The purpose of the semantic optimizer of RelationalAI is to automate this optimization by using the algebraic properties of minimum.

Weakly connected components (without reachable)

```python
def wcc[x] = minimum[ min[neighbor[x]], min[wcc[z] for z in neighbor[x] ] ]
```

Strongly connected components

```python
def scc[x] = min[v: reachable(x, v) and reachable(v, x)]
```
Basic graph algorithms

Breadth-first search

```python
def bfs[x in root] = 0
def bfs[x] = min[ bfs[x]; bfs[y: edge(y, x)] + 1 ]
```
Shortest Distance

Shortest distance between two nodes

```python
def path[x, y] = distance[x, y]
def path[x, y] = path[x, t] + distance[t, y] from t

def shortest_distance[x, y] = min[path[x, y]]
```

Shortest distance between two nodes (Bellman-Ford)

```python
def shortest_distance[x, y] =
    min[ distance[x, y];
        (shortest_distance[x, t] + distance[t, y] from t)]
```
Semantic Optimizer: Push Demand into Recursion

Optimize all-pairs shortest path to single-source shortest path using demand transformation

```python
def bacon_number[p] =
    shortest_distance[(co_star, 1)[KevinBacon, p]
```

---

```python
def bacon_number[p] =
    min[num:
        co_star(KevinBacon, p) and num = 1
    or exists(t: co_star(t, p) and num = bacon_number[t] + 1)
]  ```
**Pagerank**

Non-monotonic, relying on reaching a fixpoint

```python
def damping = 0.85
def pagerank[x in node] = 1.0, not(pagerank(x, _))

def pagerank[y in node] =
    (1.0 - damping) +
    damping * sum[pagerank[x] / outdegree[x] for x where edge(x, y)]
```

Iterative

```python
def damping = 0.85
def pagerank[x in node, 0] = 1.0

def pagerank[y in node, i in range[0, 20, 1]] =
    (1.0 - damping) +
    damping * sum[pagerank[x, i - 1] / outdegree[y] for x where edge(x, y)]
```
### Pagerank

HeapAccum<Vertex_Score>(top_k, score DESC) @@top_scores_heap;
MaxAccum<FLOAT> @@max_diff = 9999;
SumAccum<FLOAT> @sum_recd_score = 0;
SumAccum<FLOAT> @sum_score = 1;
SetAccum<EDGE> @@edge_set;

Start = \{v_type\};
WHILE @@max_diff > max_change
  LIMIT max_iter DO
    @@max_diff = 0;
    V = SELECT s
      FROM Start:s -(e_type:e)- v_type:t
      ACCUM
        t.@sum_recd_score += s.@sum_score/(s.outdegree(e_type))
      POST-ACCUM
        s.@sum_score = (1.0-damping) + damping * s.@sum_recd_score,
        s.@sum_recd_score = 0,
        @@max_diff += abs(s.@sum_score - s.@sum_score');
  END;  # END WHILE loop

### WCC

MinAccum<INT> @min_cc_id = 0;
MapAccum<INT, INT> @@comp_sizes_map;
MapAccum<INT, ListAccum<INT>> @@comp_group_by_size_map;

Start = \{v_type\};

S = SELECT x
  FROM Start:x
  POST-ACCUM x.@min_cc_id = getvid(x);

WHILE (S.size()>0) DO
  S = SELECT t
    FROM S:s -(e_type:e)- v_type:t
    ACCUM t.@min_cc_id += s.@min_cc_id
  HAVING t.@min_cc_id != t.@min_cc_id';
Recursion: Program Analysis (Doop)

```python
def VarPointsTo(var, heap) =
  AssignHeapAllocation(var, heap)

def VarPointsTo(to, heap) =
  Assign(from, to) and
  VarPointsTo(from, heap)

def VarPointsTo(to, heap) =
  LoadInstanceField(base, signature, to) and
  VarPointsTo(base, baseheap) and
  InstanceFieldPointsTo(baseheap, signature, heap)

def InstanceFieldPointsTo(baseheap, signature, heap) =
  StoreInstanceField(from, base, signature) and
  VarPointsTo(base, baseheap) and
  VarPointsTo(from, heap)
```
Syntactic Second-order Features

Transitive closure (reachability)

```python
def ancestor(x, y) = parent(x, y)
def ancestor(x, y) = exists(t: parent(x, t) and ancestor(t, y))
```

Abstract

```python
def tc[E](x, y) = E(x, y)
def tc[E](x, y) = exists(t: E(x, t) and tc[E](t, y))
```

Use

```python
def ancestor = tc[parent]
```
Syntactic Second-order Features

Mean (average)

\[
\text{sum[sales]} / \text{count[sales]}
\]

Abstract

\[
\text{def mean[F] = sum[F] / count[F]}
\]

Use

mean[sales]
Syntactic Second-order Features

Functional dependency

\[ \forall (x, v, w: \text{origin}(x, v) \text{ and } \text{origin}(x, w) \implies v = w) \]

Abstract

```python
def function(R) =
    \forall (k..., v1, v2 \text{ where } R(k..., v1) \text{ and } R(k..., v2): v1 = v2)
```

Use

`function(origin)`
module graph_analytics[G]
    with G use node, edge

    def neighbor(x, y) = edge(x, y) or edge(y, x)
def outdegree[x] = count[edge[x]]
def degree[x] = count[neighbor[x]]
def cn[x, y] = count[intersect[neighbor[x], neighbor[y]]]
def reachable = edge; reachable.edge
    def reachable_undirected = neighbor; reachable_undirected.neighbor

def scc[x] = min[v: reachable(x, v) and reachable(v, x)]
def wcc[x] = min[reachable_undirected[x]]

def cosine_sim[x, y] = cn[x, y] / sqrt[degree[x] * degree[y]]
def jaccard_sim[x, y] = cn[x, y] / count[neighbor[x]] + count[neighbor[y]] - cn[x, y]
...
end
Library Example: Relational Algebra to Calculus

```python
def intersect[R, S](x...) = R(x...) and S(x...)
def union[R, S](x...) = R(x...) or S(x...)
def diff[R, S](x...) = R(x...) and not S(x...)
def subset[R, S] = forall(x... where R(x...): S(x...))
def disjoint(R, S) = empty(R ∩ S)
def empty(R) = not exists(x...: R(x...))

def (∩) = intersect
def (∪) = union
def (∅) = cart
def (⊆) = proper_subset
def (⊂) = subset
```
Library Example: Statistics

RelationalAI features a large library of reusable functionality implemented in Rel.

```python
def mean[F] = sum[F] / count[F]

def frequency[R, elem] = count[x...: R(x...), elem]


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

\[
MSE = \frac{1}{n} \sum (y - \hat{y})^2 \\
RMSE = \sqrt{\frac{1}{n} \sum (\hat{y}_i - y_i)^2}
\]
Library Example: Machine Learning

Generic abstractions for feature scaling

```python
def mean_normalization[F][x...] =
    (F[x...] - mean[F]) / (max[F] - min[F]), (max[F] > min[F])

def min_max_normalization[F][x...] =
    (F[x...] - min[F]) / (max[F] - min[F]), (max[F] > min[F])

def zscore_normalization[F][x...] =
    (F[x...] - mean[F]) / standard_deviation[F]
```
{%- if include_columns=='*' -%}
{% set all_source_columns = adapter.get_columns_in_relation(source_table) | map(attribute='quoted') -%}
{% set include_columns = all_source_columns %}
{%- endif -%}

-- generate a CTE for each source column, a single row containing the aggregates

with
{% for source_column in source_columns %}
    {{ source_column }}_aggregates as (  
        select  
            min({{ source_column }}) as min_value,  
            max({{ source_column }}) as max_value  
        from {{ source_table }}  
    )
{% if not loop.last %}, {% endif %}
{% endfor %}

select
    {% for column in include_columns %}
        source_table.{{ column }},
    {% endfor %}
    {% for source_column in source_columns %}
        ({{ source_column }} - {{ source_column }}_aggregates.min_value)  
        / ({{ source_column }}_aggregates.max_value - {{ source_column }}_aggregates.min_value) as {{ source_column }}_scaled
    {% if not loop.last %}, {% endif %}
    {% endfor %}
from
    {% for source_column in source_columns %}
        {{ source_column }}_aggregates,
    {% endfor %}
    {{ source_table }} as source_table
The (simplified) linear prediction function uses schema abstraction \( f \) to compute a prediction for a module of features \( \text{Feature} \).

\[
\text{def linear_predict}[\text{Feature}, \text{Weight}][x...] = \\
\text{sum}[f: \text{Weight}[f] * \text{Feature}[f, x...]] + \\
\text{sum}[f: \text{Weight}[f, \text{Feature}[f, x...]]] + \\
\text{Weight}[:\text{bias}]
\]

\[
\text{def linear_regression}[\text{Feature}, \text{Response}, \text{Weight}] = \\
\text{minimize}[\text{rmse}[\text{linear_predict}[\text{Feature}, \text{Weight}], \text{Response}]]
\]

Rel \( \Rightarrow \) Core Rel generates a sum of the features (which typically have a specific schema).
Example: Gradient Descent

Simplified batch gradient descent:

```python
def max_k = 200
def alpha = 0.01

def predict[i] = linear_predict[features, weight[i]]
def predict_error[i] = rmse[response, predict[i]]
def gradient = jacobian[predict_error, weight]

def weight[i, f] =
    weight[i - 1, f] - alpha * gradient[i - 1, i - 1, f],
i < max_k

Instantiation:

def features:gdp_per_capita = min_max_normalization[gdp_per_capita]
def response = life_satisfaction
```

(This is for illustration purposes: linear regression does not normally use gradient descent)
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

\[
\text{shortest\_path}[\text{schema\_graph}[\text{flight\_graph}]], \text{:Flight, :State}]
\]

- Flight \rightarrow \text{destination} \rightarrow \text{Airport} \rightarrow \text{located\_in} \rightarrow \text{State}
- Flight \rightarrow \text{origin} \rightarrow \text{Airport} \rightarrow \text{located\_in} \rightarrow \text{State}

Schema: all acyclic paths from Flight to State

\[
\text{acyclic\_path}[\text{schema\_graph}[\text{flight\_graph}]], \text{:Flight, :State}]
\]

- Flight \rightarrow \text{destination} \rightarrow \text{Airport} \rightarrow \text{located\_in} \rightarrow \text{City} \rightarrow \text{located\_in} \rightarrow \text{County} \rightarrow \text{located\_in} \rightarrow \text{State}
- Flight \rightarrow \text{destination} \rightarrow \text{Airport} \rightarrow \text{located\_in} \rightarrow \text{City} \rightarrow \text{located\_in} \rightarrow \text{State}
- Flight \rightarrow \text{destination} \rightarrow \text{Airport} \rightarrow \text{located\_in} \rightarrow \text{County} \rightarrow \text{located\_in} \rightarrow \text{State}
- Flight \rightarrow \text{destination} \rightarrow \text{Airport} \rightarrow \text{located\_in} \rightarrow \text{State}
- ...

Note: The path algorithms are written in Rel (not foreign functions)
Feature Engineering: Describe

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

```
describe[airport]
```

<table>
<thead>
<tr>
<th>Elevation</th>
<th>State</th>
<th>Facility</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>-210</td>
<td>AIRPORT</td>
</tr>
<tr>
<td>max</td>
<td>12,442</td>
<td>ULTRALIGHT</td>
</tr>
<tr>
<td>mean</td>
<td>1,143</td>
<td></td>
</tr>
<tr>
<td>std</td>
<td>1,444</td>
<td></td>
</tr>
<tr>
<td>25%</td>
<td>270</td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>745</td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>1,220</td>
<td></td>
</tr>
<tr>
<td>unique</td>
<td>58</td>
<td>7</td>
</tr>
<tr>
<td>mode</td>
<td>TX</td>
<td>AIRPORT</td>
</tr>
</tbody>
</table>

```
(Furnace Creek, CA)
(Berthoud Pass, CO)
```

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

<table>
<thead>
<tr>
<th>Elevation</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>9,927</td>
</tr>
</tbody>
</table>

```
(Lake County, CO)
```
Describe Implementation in Rel

```python
def describe[R][column] = describe_full[R[column]]

def describe_full[R, :count] = count[R]
def describe_full[R, :min] = min[R]
def describe_full[R, :max] = max[R]

def describe_full[R, :unique] = count[last[R :> (x: not Number(x))]]
def describe_full[R, :mode] = mode[R :> (x: not Number(x))]
def describe_full[R, :mode_freq] = max[frequency[R :> (x: not Number(x))]]

def describe_full[R, :mean] = mean[R :> Number]
def describe_full[R, :std] = sample_stddev[R :> Number]
def describe_full[R, :"25\%"] = percentile[(R :> Number), 25]
def describe_full[R, :"50\%"] = median[R :> Number]
def describe_full[R, :"75\%"] = percentile[(R :> Number), 75]
```

This implementation feels very dynamic in nature but this is all handled at compile-time and the logic is specialized to the actual R.
Core Innovations for
Relational Knowledge Graphs

Incremental Computation
Incremental Computation

changes

View / Reasoning / Knowledge / Semantics Layer
Dependency Graph of Tax Analysis Logic
3.6K relations, 13K dependencies
replacing millions of lines of procedural code
Dependency Graph of Tax Analysis Logic
Focus: Single strongly-connected component (recursion)
Incremental Computation

Goal: maintain computations (views) incrementally wrt changes in the inputs.

Inputs can change along two dimensions:

i) Changes caused by changes to the state of the database
**Incremental Computation**

Goal: maintain computations (views) incrementally wrt changes in the inputs.

Inputs can change along two dimensions:

I) Changes caused by changes to the state of the database

II) Changes caused by iterative computations
The Incremental Maintenance Stack

RAI aims to support incremental processing of changes to code as well as data.

**Dependency tracking** to determine which computations are affected by a change.

**Demand-driven execution** to only compute what users are actively interested in.

**Differential computation** to incrementally maintain even general recursion.

**Semantic information** to determine that a recursive computation is monotonic.

**Semantic optimization** to recover better maintenance algorithms where possible.
Algorithms for Incremental Computation

- Semi-naive evaluation for stratified Datalog
- Generalized semi-naive evaluation (recognize more logic as monotonic)
- Differential dataflow for general non-monotonic logic

**Naive**

\[
\text{for } t = 1, 2, \ldots \text{ do } \\
\quad R_t = F(R_{t-1}) \\
\quad \text{if } R_t = R_{t-1} \text{ return } R_t \\
\text{end}
\]

**Generalized Semi-naive**

\[
\text{for } t = 1, 2, \ldots \text{ do } \\
\quad \delta R_t = F(R_{t-1}) - R_{t-1} \\
\quad R_t = R_{t-1} \oplus \delta R_t \\
\quad \text{if } \delta R_t = \emptyset \text{ return } R_t \\
\text{end}
\]
Incremental Computation: Resources and Influences

- **Convergence of Datalog over (Pre-) Semirings**
  Abo Khamis, Ngo, Pichler, Suciu, Wang, PODS 2022 (Best paper award)

- **Differential dataflow**
  McSherry, Murray, Isaacs, Isard, CIDR 2013

- **Reconciling Differences**
  Green, Ives, Tannen, Theory of Computing Systems 2011

- **F-IVM: Incremental View Maintenance with Triple Lock Factorization Benefits**
  Nikolic and Olteanu, SIGMOD 2018
Join algorithms used in SQL-based relational databases are binary join algorithms. For knowledge graphs, intermediate results are too large. Example:

\[ \text{directed}(d, m) \text{ and } \text{child}(d, a) \text{ and } \text{acted_in}(a, m) \]

Binary join options:

- directed\((d, m)\) and child\((d, a)\)
  - not selective: most directors have children!

- directed\((d, m)\) and acted_in\((a, m)\)
  - not selective: every movie has a director and actors!

- child\((d, a)\) and acted_in\((a, m)\)
  - not selective: every actor has parents!

This is one reason for the stigma 'joins are bad'
Three ways of looking at WCOJ

We use **worst-case optimal join algorithms**. This is a new class of algorithms whose properties and trade-offs are not yet well understood.

Leapfrog Triejoin (LFTJ), GenericJoin and Dovetail Join are WCOJ algorithms.

We look at the properties from three angles:

⇒ *Exploit sparsity in data*

⇒ *Recast the subquery problem and embrace correlation*

⇒ *Recast index selection problem*
WCOJ uses sparsity of all relations to narrow down search

Worst-case optimal join (WCOJ) algorithms use the sparsity of all relations to narrow down the search.
Worst-case Optimal Joins: Basic Background

Multi-way joins are used **continuously**, not just for unary joins

```
child(d, a) and directed(d, m) and acted_in(a, m)
```

Given a variable ordering of `d`, `a`, `m` (determined by query optimizer)

```
child(d, _)
directed(d, _)  find directors `d` who `directed` some movie and have some `child`

child[d](a)  find children `a` of director `d` who `acted_in` some movie
acted_in(a, _)

directed[d](m)  find movies `m` `directed` by `d` and `acted_in` by actor `a` (intersection)
acted_in[a](m)
```

WCOJ exploits all correlation **simultaneously**
How we recast the **subquery** problem

Two undesirable approaches
*(SQL systems attempt to rewrite and decorrelate to avoid these)*

---

**Top-down: Nested Loop**

- Outer query
  - for each row do
  - Subquery

**Bottom-up: (over)-compute once and reuse**

- $S = \text{Subquery}$
- Outer query
  - join with $S$

---

We address subqueries with two powerful and general methods

1. Uncorrelated subqueries are handled by **semantic optimizer**
2. Embrace correlation: **WCOJ is also a correlated join device!**
How we recast the index selection problem

Index-selection and auto-tuning is an unsolved problem.

RelationalAI users cannot be asked to manually define indexes, and even supervised tuning approaches are not acceptable.

Our solution:

- **Everything is an index in our graph-like schemas**
  
  Compare: RDF triple stores that create indexes for all orderings
  Compare: SQL table stores with an index for every functional dependency

- **WCOJ is a device to create composite indexes on-the-fly, cheaply**
How we recast the **index selection problem**

WCOJ is a device to create composite indexes on-the-fly, cheaply.
Our Evaluation Strategy: Compiled and Vectorized

Compiler and vectorized interpreter are implemented in Julia, which helps with the maintenance concerns of two back-ends.

Compiled and vectorized evaluation can be mixed in single plan!
Dovetail Join is a new join algorithm invented in January 2019. It addresses typical sources of inefficiency with worst-case optimal join algorithms:

<table>
<thead>
<tr>
<th>OVERHEAD</th>
<th>ADDRESSED VIA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime bookkeeping for join state</td>
<td>Encode as finite state machine</td>
</tr>
<tr>
<td>Overhead from abstract iterators</td>
<td>Works directly on raw iterators</td>
</tr>
<tr>
<td>Dynamic dispatch</td>
<td>Specialization</td>
</tr>
</tbody>
</table>

Dovetail/FSM is an implementation of Dovetail that leverages Julia's runtime code generation to produce ultra-efficient join kernels.
Join Algorithms: Resources and Influences

Worst-case optimal join algorithms

- **Worst-case Optimal Join Algorithms**
  Ngo, PODS 2012 (Best paper award)

- **Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm**
  Veldhuizen, ICDT 2015 (Best Newcomer Award)

- **A Worst-case Optimal Join Algorithm for SPARQL**
  Hogan, ISWC 2019

- **Worst-Case Optimal Graph Joins in Almost No Space**
  Arroyuelo, SIGMOD 2021

Correlated Subqueries

- **Unnesting Arbitrary Queries**
  Neumann, BTW 2015

- **How Materialize and other databases optimize SQL subqueries**
  Brandon, Materialize Deep Dive, March 2021
Core Innovations for Relational Knowledge Graphs

Semantic Optimization
Semantic Optimization

- Rel model
- Data
- Answer
- Semantic Optimizer
- Optimized Rel model
- Equivalent Rel models
- Knowledge
What Knowledge

User-specified constraints

- Functional dependencies etc
- Total functions, disjoint etc

Mathematical axioms

- Semirings, rings, fields, lattices, ...

Learned from the data

- Data: Summary statistics, histograms
- Query: Samples cardinality estimation
Semantic Optimization

Using mathematical knowledge in semantic optimization

\[ \min_{i, j} f[i] + g[j] \]

\[ \min_{i} f[i] + g[i] \]

\[ \text{count}[f \times g] \]

\[ \min[f] + \min[g] \]

\[ \min[f] + \min[g] \]

\[ \text{count}[f] \times \text{count}[g] \]
Semantic Optimization is not Syntactic or Ad-hoc

\[
\text{count}\{a, b, c : R(a) \land S(b) \land T(c) \land a < b < c\}
\]

\[
\text{sum}\{b : \text{count}\{a : R(a) \land S(b) \land a < b\} * \text{count}\{c : S(b) \land T(c) \land b < c\}\}
\]
Semantic Optimization is not Syntactic or Ad-hoc

\[
\text{count}[x, y: R(x) \text{ and } S(y) \text{ and } x \neq y]
\]

\[
\text{count}[R] \times \text{count}[S] - \text{count}[x, y: R(x) \text{ and } S(y) \text{ and } x = y]
\]

\[
1_{x \neq y} = 1 - 1_{x = y}
\]
def q6 =
  count[p1, p2, p3, tag:
    knows(p1, p2) and
    knows(p2, p3) and
    interest(p3, tag) and
    p1 != p3]

def q6 = sum[tmp[p3] for p1, p2, p3 where knows(p1, p2) and knows(p2, p3)] - err1 - err2

def err2 = sum[tmp[p3] for p1, p2, p3 where knows(p1, p2) and knows(p2, p3) and p1 = p3]
def err1 = sum[tmp[p1] for p1, p2 where knows(p1, p2)]
def tmp[p3] = count[tag: interest(p3, tag)]
Semantic Optimization: Running Total

```python
def running_total[t] =
sum[series[prev] for prev where prev <= t]
```

Knowledge: ordering on the temporal dimension

```python
def running_total[t] =
series[t], first(t)
```

```python
def running_total[t] =
running_total[previous[t]] + series[t]
```

(imagine not having to remember window function syntax!)
Semantic Optimization: Push Agg into Recursion

Push min aggregation into a recursive path to derive Dijkstra's algorithm

```python
def path[x, y] = edge[x, y]
def path[x, y] = path[x, t] + edge[t, y] from t
def shortest_path[x, y] = min[path[x, y]]
```

```
def shortest_path[x, y] =
    min[edge[x, y]; shortest_path[x, t] + edge[t, y] from t]
```
Semantic Optimizer: Push Demand into Recursion

Optimize all-pairs shortest path to single-source shortest path using demand transformation

```python
def bacon_number[p] =
    shortest_path[co_star × 1][KevinBacon, p]
```

```python
def bacon_number[p] =
    min[num:
        co_star(KevinBacon, p) and num = 1
        or exists(t: co_star(t, p) and num = bacon_number[t] + 1)
    ]
```
Optimization supports Abstraction

```python
def shortest_path[x, y] = min[path[x, y]]
```

No need for separate single-source vs all-pairs definitions
Reuse the very large path relation.

```python
def scc[x] = min[v: reachable(x, v) and reachable(v, x)]
```

Reuse the very large reachable relation.

```python
def wcc[x] = min[reachable_undirected[x]]
```

Reuse the very large reachable_undirected relation.

```python
def mean[R] = sum[R] / count[R]
```

Pretty bad without aggregation optimization
Semantic Optimization: Resources and Influences

- **FAQ: Questions Asked Frequently**
  Khamis, Ngo, Rudra, PODS 2016 (Best Paper Award)

- **What Do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another**
  Khamis, Ngo, Suciu, PODS 2017

- **Precise complexity analysis for efficient Datalog queries**
  Tekle et al., PPDP 2010

- **Functional Aggregate Queries with Additive Inequalities**
  Khamis et al., PODS 2019

- **Convergence of Datalog over (Pre-) Semirings**
  Khamis, Ngo, Pichler, Suciu, Wang, PODS 2022 (Best paper award)

- **Factorised representations of query results: size bounds and readability**
  Olteanu, Zavodny, ICDT 2012 (2022 Test of time award)
Core Innovations for Relational Knowledge Graphs

Live Programming and Incrementality
The Incremental Maintenance Stack

RAI aims to support incremental processing of changes to **code** as well as **data**.

**Dependency tracking** to determine which computations are affected by a change.

**Demand-driven execution** to only compute what users are actively interested in.

**Differential computation** to incrementally maintain even general recursion.

**Semantic information** to determine that a recursive computation is monotonic.

**Semantic optimization** to recover better maintenance algorithms where possible.
Eagerly maintaining the entire model is not a good idea at this scale.

RAI is entirely demand-driven, which means that computations only happen when the result is needed (or when executed in the background to catch up). The architecture is based on PL incremental compiler research for IDEs.

Challenges:
- when to do invalidation and evaluation
- incrementally maintaining cyclic computation (scc)
Eager maintenance is bad

Lazy maintenance is bad
detecting dirty computations is too expensive when an output is queried.

Best: Eager invalidation
lazy evaluation
Dependency Graph of Tax Analysis Logic
3.6K relations, 13K dependencies
replacing millions of lines of procedural code
Dependency Graph of Tax Analysis Logic
Focus: Single strongly-connected component (recursion)
Dependency Graph of Tax Analysis Logic
Focus: Single node with many dependencies
The architecture is based on PL incremental compiler research for IDEs.

Key ingredients:
- Precise dependency tracking (treat access to the catalog as queries)
- Memoization and invalidation (on input changes)

We've open-sourced **Salsa.jl**, our framework for writing responsive compilers.
- **Responsive compilers** – Nicholas Matsakis – PLISS 2019
- **JuliaCon 2020** – Salsa.jl – Nathan Daly
Core Innovations for Relational Knowledge Graphs

Relational Models for Machine Learning

Unconstrained Optimization Models
Current Practice in Machine Learning

Beautiful relational schema without redundancy

Feature extraction query

Step 1: **throw away all** the structure and knowledge on the data set (eg dependencies).

Design matrix: the ultimate denormalization

<table>
<thead>
<tr>
<th>k</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>...</th>
<th>y</th>
</tr>
</thead>
</table>

TensorFlow

PyTorch

scikit-learn
<table>
<thead>
<tr>
<th>sku</th>
<th>store</th>
<th>date</th>
<th>sold</th>
<th>color</th>
<th>price</th>
<th>city</th>
<th>size</th>
<th>state</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>2022-03-26</td>
<td>5</td>
<td>Red</td>
<td>$5.14</td>
<td>Seattle</td>
<td>4000 sqft</td>
<td>WA</td>
<td>53</td>
</tr>
<tr>
<td>1</td>
<td>S1</td>
<td>2022-03-27</td>
<td>7</td>
<td>Red</td>
<td>$5.14</td>
<td>Seattle</td>
<td>4000 sqft</td>
<td>WA</td>
<td>53</td>
</tr>
<tr>
<td>1</td>
<td>S1</td>
<td>2022-03-28</td>
<td>3</td>
<td>Red</td>
<td>$5.14</td>
<td>Seattle</td>
<td>4000 sqft</td>
<td>WA</td>
<td>53</td>
</tr>
</tbody>
</table>
Relational Modelling for Machine Learning

With our research network we have developed training methods that do not require creating a design matrix of features and operate directly on the relational structure.

Key innovations:

● **Rel language** – concisely expressing generic machine learning models
● **Automatic differentiation** of relational cost function
● **Semantic optimizer** – exploit relational structure and independence
● Optimization method executed iteratively in RAI system
● Execute large numbers of aggregations efficiently
Rel - Math for Linear Regression

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

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

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

Application-specific instantiation

```python
def features[:gdp_per_capita] = ...
def response = life_satisfaction

def model = linear_regression[features, response, initial_point]
```
Semantic Optimization for Covariance Matrix

<table>
<thead>
<tr>
<th>sku</th>
<th>store</th>
<th>date</th>
<th>sold</th>
<th>Red</th>
<th>Green</th>
<th>price</th>
<th>Seattle</th>
<th>San Diego</th>
<th>size</th>
<th>WA</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>2022-03-26</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>$5.14</td>
<td>1</td>
<td>0</td>
<td>4000 sqft</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>S1</td>
<td>2022-03-27</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>$5.14</td>
<td>1</td>
<td>0</td>
<td>4000 sqft</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Generic covariance matrix:

```python
def covariance[j, k] =
    sum[st, sk, d: design_matrix[j, st, sk, d] * design_matrix[k, st, sk, d]]
```

Imagine the specialize to price and size:

```python
def covariance[:price, :size] =
    sum[st, sk, d: design_matrix[:size, st, sk, d] * design_matrix[:price, st, sk, d]]
```

**Price** is independent of **store** and **date**

**Size** is independent of **sku** and **date**

```python
def covariance[:price, :size] =
    (sum[st: features[:price, st]] * count[stores] * count[dates]) *
    (sum[sk: features[:size, sk]] * count[skus] * count[dates])
```
Relational Machine Learning: Resources and Influences

- **A Layered Aggregate Engine for Analytics Workloads**
  Schleich, Olteanu, Khamis, Ngo, Nguyen, SIGMOD 2019

- **Learning Models over Relational Data Using Sparse Tensors and Functional Dependencies**
  Khamis, Ngo, Nguyen, Olteanu, Schleich, PODS 2018, TODS 2020

- **The Relational Data Borg is Learning**
  Olteanu, VLDB 2020 Keynote (youtube recording: [watch?v=0ic0jMjoP0M](https://www.youtube.com/watch?v=0ic0jMjoP0M), [watch?v=kWm-0BnbEoU](https://www.youtube.com/watch?v=kWm-0BnbEoU))

- **Structure-Aware Machine Learning over Multi-Relational Databases**
  Schleich, PhD thesis, Honorable mention for the 2021 SIGMOD Jim Gray Doctoral Dissertation Award

- **Relational Knowledge Graphs as the Foundation for Artificial Intelligence**
  Aref (youtube recording: [watch?v=VpyGbjUzG7Y](https://www.youtube.com/watch?v=VpyGbjUzG7Y))

- **Rk-means: Fast Clustering for Relational Data**
  Curtin, Moseley, Ngo, Nguyen, Olteanu, Schleich, AISTATS 2020
Core Innovations for Relational Knowledge Graphs

Relational Models for Mathematical Optimization

Constrained Optimization Models
Optimization

Unconstrained Optimization

- Objective: the error/loss function
- Solver: differentiable function, often gradient descent
- All solutions are acceptable

Constrained optimization

- Objective: minimize or maximize the function
- Solver: LP, ILP, MIP etc
- Not all solutions are acceptable: constraints
- Mathematical optimization problems are specified in high-level math expressions (AMPL, JuMP). The problems are easily written in Rel
Model for Manufacturing Problem

Given:  
- $P$, a set of products
- $a_j$ = tons per hour of product $j$, for each $j \in P$
- $b$ = hours available at the mill
- $c_j$ = profit per ton of product $j$, for each $j \in P$
- $u_j$ = maximum tons of product $j$, for each $j \in P$

Define variables: $X_j$ = tons of product $j$ to be made, for each $j \in P$

Maximize:  
$$\sum_{j \in P} c_j X_j$$

Subject to:  
$$\sum_{j \in P} \left(\frac{1}{a_j}\right) X_j \leq b$$
$$0 \leq X_j \leq u_j, \text{ for each } j \in P$$

@variable(model, make[products])
@objective(model, Max, sum(prod_profit[p] * make[p] for p in products))
@constraint(model, sum(1 / prod_rate[p] * make[p] for p in products) <= 40)
@constraint(model, [p in products], 0 <= make[p] <= prod_max[p])

var Make{p in PROD}
maximize Profit: sum{p in PROD} prod_profit[p] * Make[p];
subject to Time: sum{p in PROD} (1 / prod_rate[p]) * Make[p] <= 40;
subject to Limit{p in PROD}: 0 <= Make[p] <= prod_max[p]
Relational Model

Rel supports expressing the objective function and constraints.

The system grounds the constraint in the database and pass the problem to a solver (eg CPLEX, Gurobi, Xpress)

```python
def total_profit =
    sum[prod_profit[p] * make[p] for p in products]

def time_avail() =
    sum[(1 / prod_rate[p]) * make[p] for p in products] ≼ avail

def demand_market() =
    forall(p in products: make[p] ≼ prod_market[p])
```

Optimization happens in the dependency graph, so inputs to the solver can computed Rel definitions or even other optimization problems.
Core Innovations for Relational Knowledge Graphs

Interfaces: SQL ❤ Rel
DuckDB-based SQL Interface

DuckDB is an embeddable SQL OLAP database management system with great performance, excellent quality, small footprint and enjoying quick adoption.

RAI uses DuckDB for SQL support. Rel is used to model SQL tables, which are used by DuckDB for SQL query evaluation. Individual 'columns' can be data vs views.

DuckDB has outstanding support for working with a dynamic catalog.

Other approaches we evaluated:
- Calcite
- DuckDB query plan
- PostgreSQL parser

```
module order
    def orderkey[o] = ...
    def customer[o] = ...
    def orderdate[o] = ...
    def totalprice[o] = sum[1:: charge[o, num]]
end

SELECT orderkey, customer, orderdate
FROM order
WHERE totalprice > 100
```
Recap
Incremental computation for fixpoint computation and database changes

Rel - An expressive relational language

Relational models for mathematical optimization

Semantic optimization

SQL support with DuckDB engine

Large scale reasoning

Immutable database in durable object storage, including immutable catalog. Write-optimized.

Vectorized engine and compiled WCOJ algorithms, addressing subquery and index selection.

Relational machine learning utilizing semantic optimization.

Large scale reasoning

SQL support with DuckDB engine

Rel - An expressive relational language

Relational models for mathematical optimization

Semantic optimization

Incremental computation for fixpoint computation and database changes

Immutable database in durable object storage, including immutable catalog. Write-optimized.

Vectorized engine and compiled WCOJ algorithms, addressing subquery and index selection.
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!