DATALAB
@Northeastern

Database Research needs an Abstract Relational Query Language

Wolfgang Gatterbauer, Diandre Sabale

CIDR 2026

https://RelationalDiagrams.com

Anonymous feedback:

Recorded talk: https://www.youtube.com/watch?v=BZYlQYEtLXw

A discussion of the evolution of the database industry over the past half century, and why the relational database concepts introduced by E.F. Codd have proven to be so resilient over several decades.

BY DONALD CHAMBERLIN

50 Years of Queries

CACM 2024. https://doi.org/10.1145/3649887

"the (SQL) Standard… cannot serve as a formal semantics… because it is … inherently ambiguous."

A Formal Semantics of SQL Queries, Its Validation, and Applications

Paolo Guagliardo
School of Informatics
University of Edinburgh
pguaglia@inf.ed.ac.uk

Leonid Libkin
School of Informatics
University of Edinburgh
libkin@inf.ed.ac.uk

PVLDB 2017

1 Can SQL really serve as <u>reference</u> language?

A discussion of the evolution of the database industry over the past half century, and why the relational database concepts introduced by E.F. Codd have proven to be so resilient over several decades.

BY DONALD CHAMBERLIN

50 Years of Queries

CACM 2024. https://doi.org/10.1145/3649887

A Formal Semantics of SQL Queries, Its Validation, and Applications

Paolo Guagliardo
School of Informatics
University of Edinburgh
pguaglia@inf.ed.ac.uk

Leonid Libkin
School of Informatics
University of Edinburgh
libkin@inf.ed.ac.uk

PVLDB 2017

"if we want to capture the meaning of SQL queries in the real world we cannot just pretend that nulls do not exist."

Rel: A Programming Language for Relational Data

Molham Aref
RelationalAI
molham.aref@relational.ai

Paolo Guagliardo
University of Edinburgh
paolo.guagliardo@ed.ac.uk

George Kastrinis
RelationalAI
george.kastrinis@relational.ai

Leonid Libkin
RelationalAI & Univ of Edinburgh
leonid.libkin@relational.ai

Victor Marsault
LIGM, Univ. Gustave Eiffel, CNRS
victor.marsault@univ-eiffel.fr

Wim Martens
RelationalAI & University of Bayreuth
wim.martens@relational.ai

Mary McGrath
RelationalAI
mary.mcgrath@skylight.digital

Filip Murlak
University of Warsaw
f.murlak@uw.edu.pl

Nathaniel Nystrom
RelationalAI
nate.nystrom@relational.ai

Liat Peterfreund
Hebrew University
liat.peterfreund@mail.huji.ac.il

Allison Rogers
RelationalAI
allison.rogers@relational.ai

Cristina Sirangelo
Université Paris Cité, CNRS, IRIF
cristina@irif.fr

Domagoj Vrgoč
PUC Chile
vrdomagoj@uc.cl

David Zhao
RelationalAI
david.zhao@relational.ai

Abdul Zreika
RelationalAI
abdul.zreika@relational.ai

SIGMOD 2025

".. the pure relational model … has neither multiplicities (bags) nor nulls…"

2 Set or bag semantics as reference? Null values or not?

"algebra with its explicit operator ordering is a good foundation for a modern query language." "nicer ... way to express queries"

A Critique of Modern SQL And A Proposal Towards A Simple and Expressive Query Language

Thomas Neumann
Technische Universität München
neumann@in.tum.de

Viktor Leis
Technische Universität München
leis@in.tum.de

CIDR 2024

"(SQL's) 'inside-out' structure makes ... SQL logic difficult." "pipe syntax ... can make the query ... easier to understand."

SQL Has Problems. We Can Fix Them: Pipe Syntax In SQL

Jeff Shute
Google, Inc.

Shannon Bales
Google, Inc.

Matthew Brown
Google, Inc.

Jean-Daniel Browne
Google, Inc.

Brandon Dolphin
Google, Inc.

Romit Kudtarkar
Google, Inc.

Andrey Litvinov
Google, Inc.

Jingchi Ma
Google, Inc.

John Morcos
Google, Inc.

Michael Shen
Google, Inc.

David Wilhite
Google, Inc.

Xi Wu
Google, Inc.

Lulan Yu
Google, Inc.

PVLDB 2024

3 Linear dataflow is easier to understand than inside-out.

"modularity ... SaneQL allows reusing logic across queries, which is not traditionally done in SQL"

A Critique of Modern SQL And A Proposal Towards A Simple and Expressive Query Language

Thomas Neumann
Technische Universität München
neumann@in.tum.de

Viktor Leis
Technische Universität München
leis@in.tum.de

CIDR 2024

"SQL's syntactic structure for composing queries using those operations is terrible."

SQL Has Problems. We Can Fix Them: Pipe Syntax In SQL

Jeff Shute
Google, Inc.

Shannon Bales
Google, Inc.

Matthew Brown
Google, Inc.

Jean-Daniel Browne
Google, Inc.

Brandon Dolphin
Google, Inc.

Romit Kudtarkar
Google, Inc.

Andrey Litvinov
Google, Inc.

Jingchi Ma
Google, Inc.

John Morcos
Google, Inc.

Michael Shen
Google, Inc.

David Wilhite
Google, Inc.

Xi Wu
Google, Inc.

Lulan Yu
Google, Inc.

PVLDB 2024

4 Linear dataflow allows modularity, i.e. composing queries

Pig Latin: A Not-So-Foreign Language for Data Processing

Christopher Olston* Yahoo! Research
Benjamin Reed† Yahoo! Research
Utkarsh Srivastava‡ Yahoo! Research
Ravi Kumar§ Yahoo! Research
Andrew Tomkins¶ Yahoo! Research

SIGMOD 2008

"Dataflow Language ... making it easier for programmers to understand ... their data processing task"

DFQL: Dataflow query language for relational databases

Gard J. Clark, C. Thomas Wu *
Naval Postgraduate School, Department of Computer Science, Code CS, Monterey, CA 93943, USA

1994

"humans think of complex problems in a sequential fashion"

Human Factors Comparison of a Procedural and a Nonprocedural Query Language

CHARLES WELTY
University of Southern Maine
and
DAVID W. STEMPLE
University of Massachusetts

1981

"subjects using the more procedural language wrote difficult queries better than subjects using the less procedural language."

5 How can we shed new light into the decades-old pipe/dataflow debate?

Pig Latin: A Not-So-Foreign Language for Data Processing

Christopher Olston* Yahoo! Research
Benjamin Reed† Yahoo! Research
Utkarsh Srivastava‡ Yahoo! Research
Ravi Kumar§ Yahoo! Research
Andrew Tomkins¶ Yahoo! Research

SIGMOD 2008

On The Reasonable Effectiveness of Relational Diagrams*

Explaining Relational Query Patterns and the Pattern Expressiveness of Relational Languages

WOLFGANG GATTERBAUER, Northeastern University, USA
CODY DUNNE, Northeastern University, USA

SIGMOD 2024, SIGMOD record 2025

"relational calculus can express a larger class of patterns than the basic operators of relational algebra."

6 Can we have both: easy-to-understand and full pattern expressiveness?

The Cambridge Report on Database Research

Anastasia Ailamaki, Samuel Madden, Daniel Abadi, Gustavo Alonso, Sihem Amer-Yahia, Magdalena Balazinska, Philip A. Bernstein, Peter Boncz, Michael Cafarella, Surajit Chaudhuri, Susan Davidson, David DeWitt, Yanlei Diao, Xin Luna Dong, Michael Franklin, Juliana Freire, Johannes Gehrke, Alon Halevy, Joseph M. Hellerstein, Mark D. Hill, Stratos Idreos, Yannis Ioannidis, Christoph Koch, Donald Kossmann, Tim Kraska, Arun Kumar, Guoliang Li, Volker Markl, Renée Miller, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Aditya Parameswaran, Ippokratis Pandis, Jignesh M. Patel, Andrew Pavlo, Danica Porobic, Viktor Sanca, Michael Stonebraker, Julia Stoyanovich, Dan Suciu, Wang-Chiew Tan, Shiv Venkataraman, Matei Zaharia, and Stanley B. Zdonik

arXiv 2025

"LLMs ... make mistakes, effective explanation mechanisms ... will become increasingly important"

NL2SQL is a solved problem... Not!

Avrilia Floratou[1], Fotis Psallidas[1], Fuheng Zhao[2,*], Shaleen Deep[1], Gunther Hagleither[3], Wangda Tan[4], Joyce Cahoon[1], Rana Alotaibi[1], Jordan Henkel[1], Abhik Singla[1], Alex van Grootel[1], Brandon Chow[1], Kai Deng[1], Katherine Lin[1], Marcos Campos[1], Venkatesh Emani[1], Vivek Pandit[1], Victor Shnayder[1], Wenjing Wang[1], Carlo Curino[1]
[1]Microsoft, [2] University of California, Santa Barbara, [3,4] Waii

CIDR 2024

"determine the intent of the user" ... "focusing on the alignment with the user's intention"

7 Shouldn't we make easier:
- for machines to write queries, and
- for humans to understand queries?

We need a Rosetta Stone for relational queries:

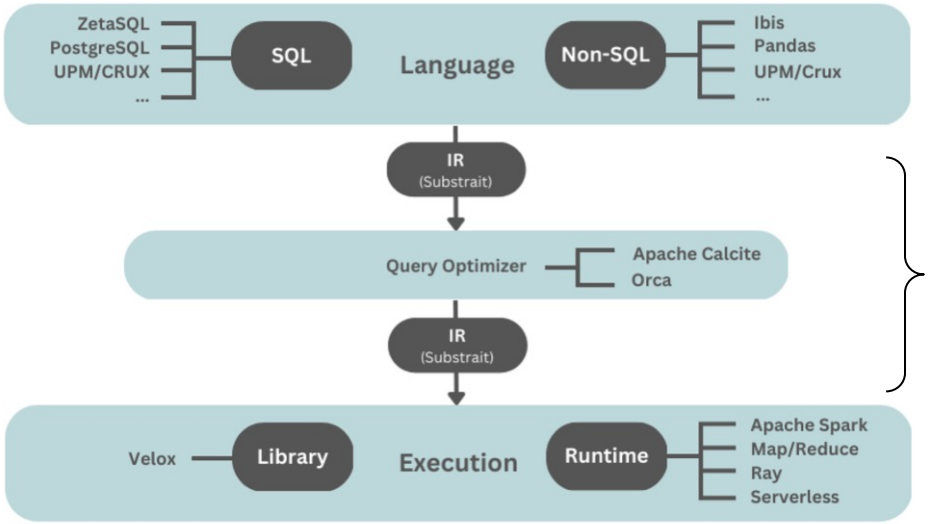
1. a canonical representation of relational intent ("*an abstract relational query language*")
2. a refined vocabulary to separate intent from:
   - *modalities* (interface, and
   - *conventions* (semantic parameters)

The Composable Data Management System Manifesto

Pedro Pedreira, Meta Platforms Inc., pedroerp@meta.com
Orri Erling, Meta Platforms Inc., oerling@meta.com
Konstantinos Karanasos, Meta Platforms Inc., kkaranasos@meta.com
Scott Schneider, Meta Platforms Inc., scottas@meta.com
Wes McKinney, Voltron Data, wes@voltrondata.com
Satya R Valluri, Databricks Inc., satya.valluri@databricks.com
Mohamed Zait, Databricks Inc., mohamed.zait@databricks.com
Jacques Nadeau, Sundeck, jacques@sundeck.io

PVLDB 2023

"new modular data stack ... provides a stronger separation between language and execution ... the execution is language-independent"



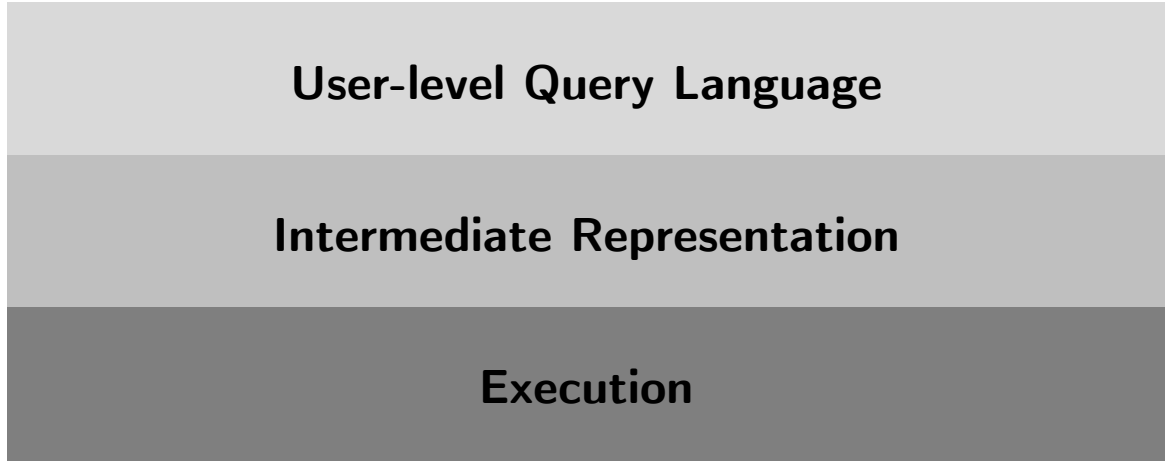
User-level QLs
(e.g. SQL, SaneQL, Rel, ...)

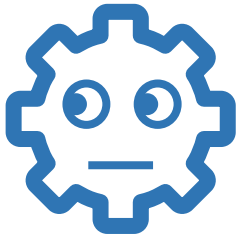
Intermediate Representations
(e.g. SDQL, Substrait, ...)

Figure 1: Open source modular data stack outline.

An Abstract Relational Query Language

Human





Machine

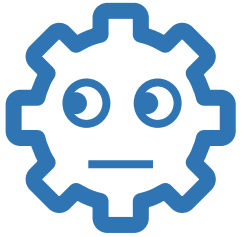
An Abstract Relational Query Language

Human

User-level Query Language

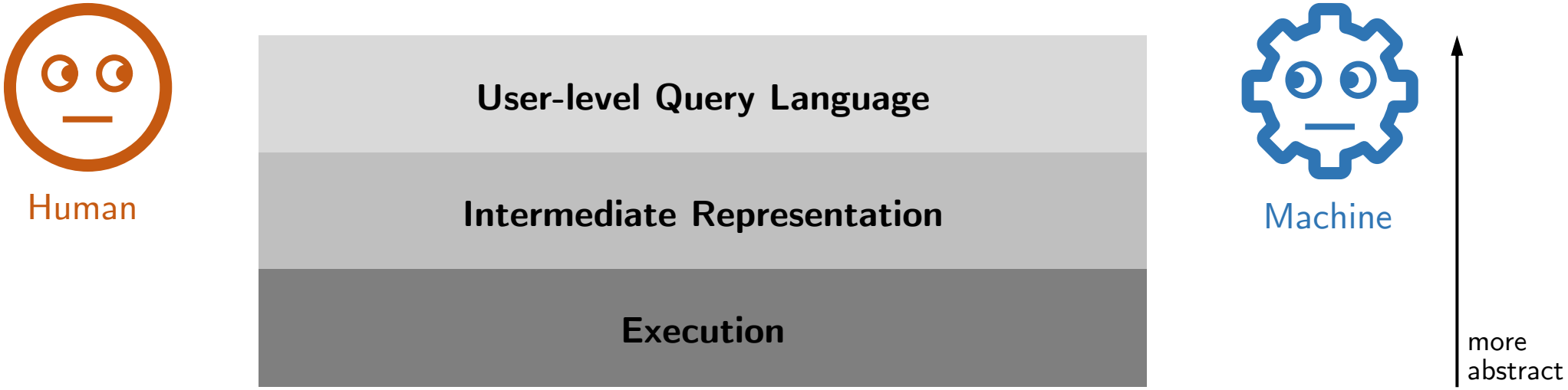
Intermediate Representation

Execution



Machine

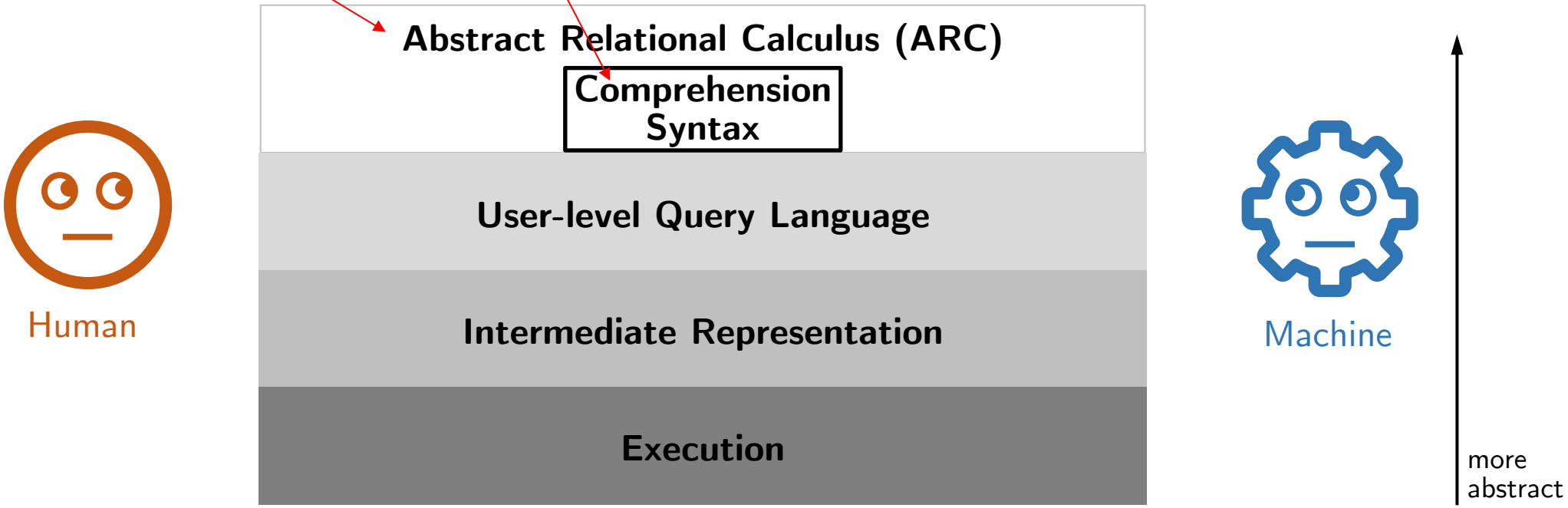
An Abstract Relational Query Language



An Abstract Relational Query Language

Reference language. Focus is on the intent: Separates "relational intent" from representation.

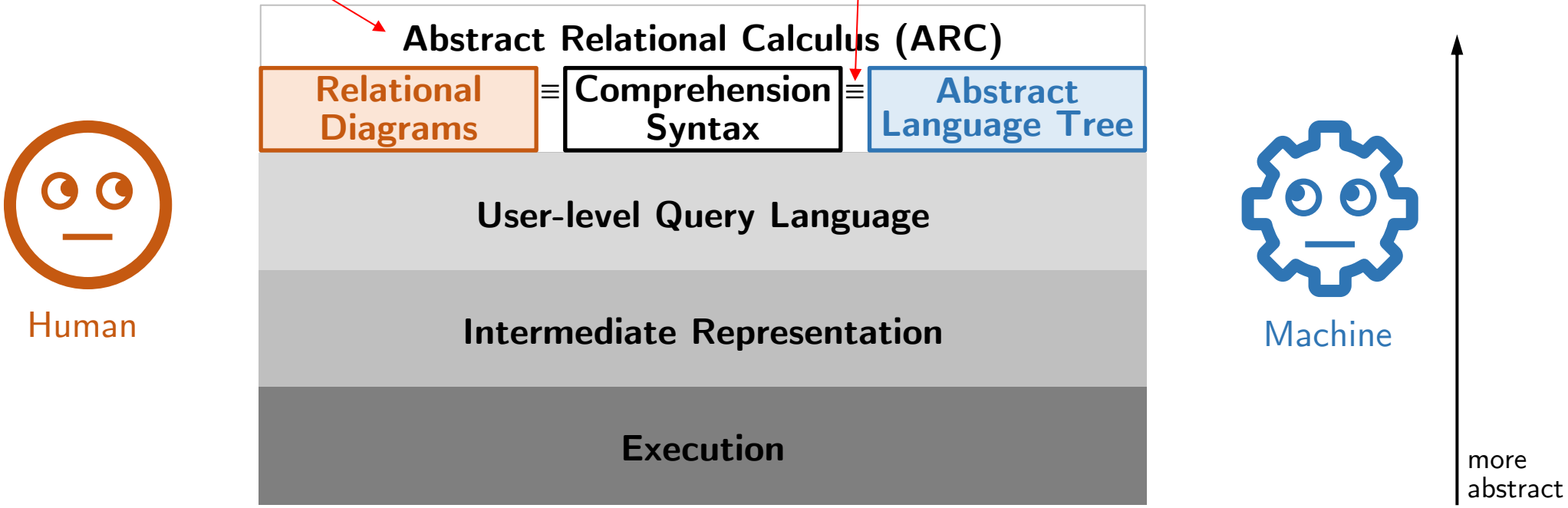
Textual representation



An Abstract Relational Query Language

Reference language. Focus is on the intent: Separates "relational intent" from representation.

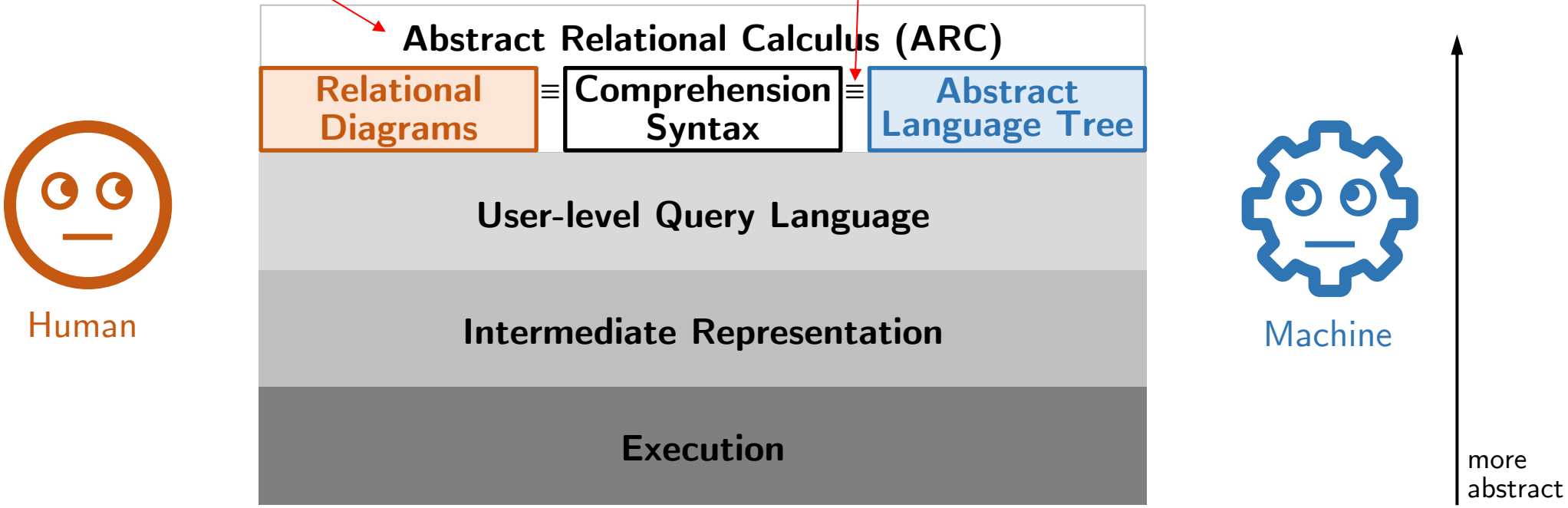
Modality: alternative representation of the intent tailored to different audiences



An Abstract Relational Query Language

Reference language. Focus is on the intent: Separates "relational intent" from representation.

Modality: alternative representation of the intent tailored to different audiences



Convention: orthogonal environment-level semantic parameters under which the relational intent is interpreted (e.g., set vs. bag semantics, or treatment of null values)

An Abstract Relational Query Language

Reference language. Focus is on the intent: Separates "relational intent" from representation.

Modality: alternative representation of the intent tailored to different audiences

Our suggestion: ***Abstract Relational Calculus* (ARC)**:

- generalization of Tuple Relational Calculus (TRC) (grouping, aggregation, recursion, outer joins, IN with nulls, built-in predicates, external relations, ... etc.)

see paper

- allows split into 1 intent / 2 modality / 3 convention
- vocabulary for more fine-grained discussion, incl.:
- modularity through abstract relations

talk

Convention: orthogonal environment-level semantic parameters under which the intent is interpreted (e.g., set vs. bag semantics, or treatment of null values)

Modalities & Conventions

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

SQL

```
select A as G
from Left, Right
where A = C
```

Datalog

$Q(x) :- Left(x,_), Right(x,_)$

Tuple Relational Calculus (TRC)

$\{Q(G) \mid \exists l \in Left, r \in Right\ [$
$\quad Q.G = l.A \wedge l.A = r.C]\}$

Languages differ in how explicitly they encode and surface the primitives of relational languages

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

SQL

```
select A as G
from Left, Right
where A = C
```

Datalog

$Q(x) :- Left(x,_), Right(x,_)$

assignment can be from left or right (logically equivalent)

Languages differ in how explicitly they encode and surface the primitives of relational languages

Tuple Relational Calculus (TRC)

$\{Q(G) \mid \exists l \in Left, r \in Right\ [\boxed{Q.G = l.A} \wedge l.A = r.C]\}$

assignment of the output from Left

The tuple perspective is more explicit

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

SQL

```
select A as G
from Left, Right
where A = C
```

```
select A as G
from Left, Right
where A > C
```

assignment can be from left or right (logically equivalent)

Datalog

Q(x) :– Left(x,_), Right(x,_)

Q(x) :– Left(x,_), Right(y,_), x>y

normalized

Tuple Relational Calculus (TRC)

{Q(G) | ∃l∈Left, r∈Right [Q.G=l.A ∧ l.A=r.C]}

{Q(G) | ∃l∈Left, r∈Right [Q.G=l.A ∧ l.A>r.C]}

assignment of the output from Left

The tuple perspective is more explicit

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

Languages

SQL

```
select A as G
from Left, Right
where A = C
```

Datalog

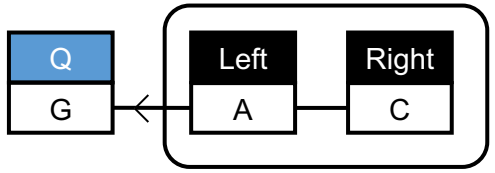
Q(x) :– Left(x,_), Right(x,_)

Tuple Relational Calculus (TRC)

{Q(G) | ∃l∈Left, r∈Right [
  Q.G=l.A ∧ l.A=r.C]}

modality: alternative representation of the intent tailored to different audiences

```
COLLECTION
├─ HEAD: Q(G)
└─ QUANTIFIER ∃
   ├─ BINDING: l ∈ Left
   ├─ BINDING: r ∈ Right
   └─ AND ∧
      ├─ PREDICATE: Q.G = l.A
      └─ PREDICATE: l.A = r.C
```



Relational Diagram ***Set comprehension based textual notation*** ***Abstract Language Tree***

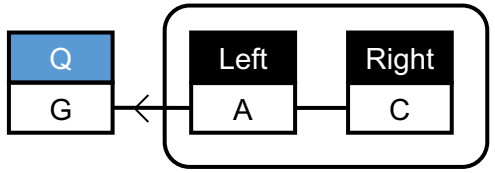
Modalities

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

modality: alternative representation of the intent tailored to different audiences

one-to-one correspondence b/w relational concepts across the modalities



Tuple Relational Calculus (TRC)

$$\{Q(G) \mid \exists l \in Left, r \in Right\ [Q.G = l.A \land l.A = r.C]\}$$

```
COLLECTION
├─ HEAD: Q(G)
└─ QUANTIFIER ∃
   ├─ BINDING: l ∈ Left
   ├─ BINDING: r ∈ Right
   └─ AND ∧
      ├─ PREDICATE: Q.G = l.A
      └─ PREDICATE: l.A = r.C
```

Relational Diagram — *Set comprehension based textual notation* — *Abstract Language Tree*

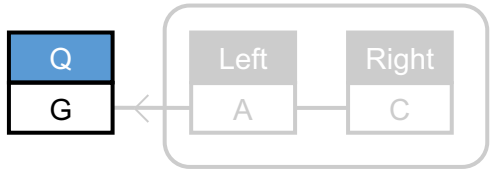
Modalities

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

modality: alternative representation of the intent tailored to different audiences

one-to-one correspondence b/w relational concepts across the modalities



Tuple Relational Calculus (TRC)

$\{Q(G) \mid \exists l \in Left, r \in Right [Q.G = l.A \wedge l.A = r.C]\}$

```
COLLECTION
├─ HEAD: Q(G)
└─ QUANTIFIER ∃
   ├─ BINDING: l ∈ Left
   ├─ BINDING: r ∈ Right
   └─ AND ∧
      ├─ PREDICATE: Q.G = l.A
      └─ PREDICATE: l.A = r.C
```

Relational Diagram — *Set comprehension based textual notation* — *Abstract Language Tree*

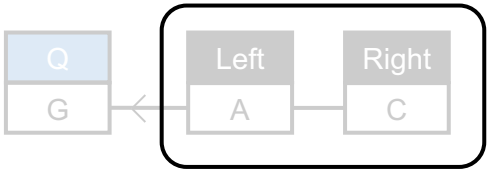
Modalities

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

modality: alternative representation of the intent tailored to different audiences

one-to-one correspondence b/w relational concepts across the modalities



Tuple Relational Calculus (TRC)

$$\{Q(G) \mid \exists l \in Left, r \in Right\ [Q.G = l.A \land l.A = r.C]\}$$

```
COLLECTION
├─ HEAD: Q(G)
└─ QUANTIFIER ∃
   ├─ BINDING: l ∈ Left
   ├─ BINDING: r ∈ Right
   └─ AND ∧
      ├─ PREDICATE: Q.G = l.A
      └─ PREDICATE: l.A = r.C
```

Relational Diagram — *Set comprehension based textual notation* — *Abstract Language Tree*

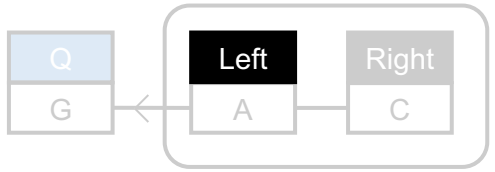
Modalities

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

modality: alternative representation of the intent tailored to different audiences

one-to-one correspondence b/w relational concepts across the modalities



Tuple Relational Calculus (TRC)

$\{Q(G) \mid \exists l \in Left, r \in Right [Q.G = l.A \land l.A = r.C]\}$

```
COLLECTION
├─ HEAD: Q(G)
└─ QUANTIFIER ∃
   ├─ BINDING: l ∈ Left
   ├─ BINDING: r ∈ Right
   └─ AND ∧
      ├─ PREDICATE: Q.G = l.A
      └─ PREDICATE: l.A = r.C
```

Relational Diagram ← *Set comprehension based textual notation* → *Abstract Language Tree*

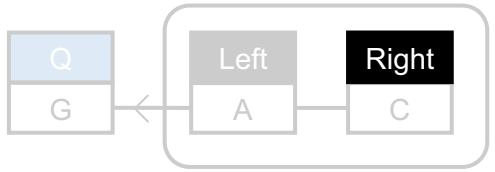
Modalities

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

modality: alternative representation of the intent tailored to different audiences

one-to-one correspondence b/w relational concepts across the modalities



Tuple Relational Calculus (TRC)

$\{Q(G) \mid \exists l \in Left, r \in Right\ [Q.G = l.A \land l.A = r.C]\}$

```
COLLECTION
├─ HEAD: Q(G)
└─ QUANTIFIER ∃
   ├─ BINDING: l ∈ Left
   ├─ BINDING: r ∈ Right
   └─ AND ∧
      ├─ PREDICATE: Q.G = l.A
      └─ PREDICATE: l.A = r.C
```

Relational Diagram ⟷ *Set comprehension based textual notation* ⟷ *Abstract Language Tree*

Modalities

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

modality: alternative representation of the intent tailored to different audiences

one-to-one correspondence b/w relational concepts across the modalities

conjunction = juxtaposition

Tuple Relational Calculus (TRC)

$$\{Q(G) \mid \exists l \in Left, r \in Right\ [Q.G = l.A \land l.A = r.C]\}$$

```
COLLECTION
├─ HEAD: Q(G)
└─ QUANTIFIER ∃
   ├─ BINDING: l ∈ Left
   ├─ BINDING: r ∈ Right
   └─ AND ∧
      ├─ PREDICATE: Q.G = l.A
      └─ PREDICATE: l.A = r.C
```

Relational Diagram ⟷ *Set comprehension based textual notation* ⟷ *Abstract Language Tree*

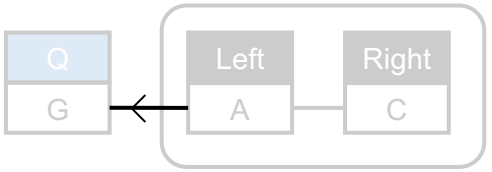
Modalities

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

modality: alternative representation of the intent tailored to different audiences

one-to-one correspondence b/w relational concepts across the modalities



Tuple Relational Calculus (TRC)

$\{Q(G) \mid \exists l \in Left, r \in Right [$
$Q.G = l.A \wedge l.A = r.C]\}$

```
COLLECTION
├─ HEAD: Q(G)
└─ QUANTIFIER ∃
   ├─ BINDING: l ∈ Left
   ├─ BINDING: r ∈ Right
   └─ AND ∧
      ├─ PREDICATE: Q.G = l.A
      └─ PREDICATE: l.A = r.C
```

Relational Diagram — *Set comprehension based textual notation* — *Abstract Language Tree*

Modalities

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

modality: alternative representation of the intent tailored to different audiences

one-to-one correspondence b/w relational concepts across the modalities

Tuple Relational Calculus (TRC)

$$\{Q(G) \mid \exists l \in Left, r \in Right\ [\ Q.G = l.A \wedge l.A = r.C\ ]\}$$

```
COLLECTION
├─ HEAD: Q(G)
└─ QUANTIFIER ∃
   ├─ BINDING: l ∈ Left
   ├─ BINDING: r ∈ Right
   └─ AND ∧
      ├─ PREDICATE: Q.G = l.A
      └─ PREDICATE: l.A = r.C
```

Relational Diagram — *Set comprehension based textual notation* — *Abstract Language Tree*

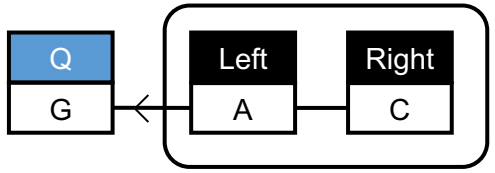
Modalities

Languages vs. Modalities

Relational schema: Left(A,B), Right(C,D)

modality: alternative representation of the intent tailored to different audiences

one-to-one correspondence b/w relational concepts across the modalities



Tuple Relational Calculus (TRC)

$$\{Q(G) \mid \exists l \in Left, r \in Right\ [Q.G = l.A \land l.A = r.C]\}$$

```
COLLECTION
├─ HEAD: Q(G)
└─ QUANTIFIER ∃
   ├─ BINDING: l ∈ Left
   ├─ BINDING: r ∈ Right
   └─ AND ∧
      ├─ PREDICATE: Q.G = l.A
      └─ PREDICATE: l.A = r.C
```

Relational Diagram ⟷ *Set comprehension based textual notation* ⟷ *Abstract Language Tree*

Modalities

Conventions

Left

A	B
a	1

Right

C	D
a	1
a	2

SQL

```
select A,
  (select sum(D) as sm
  from Right
  where A = C)
from Left
```

A	sm
a	3

Datalog (Soufflé)

```
Q(x, sum y: {Right(x, y)}) :- Left(x,_).
```

A	sm
a	3

Conventions

Left

A	B
a	1
b	2

Right

C	D
a	1
a	2
b	3
b	4

SQL

```
select A,
  (select sum(D) as sm
  from Right
  where A = C)
from Left
```

A	sm
a	3
b	7

Datalog (Soufflé)

```
Q(x, sum y: {Right(x, y)}) :- Left(x,_).
```

A	sm
a	3
b	7

Conventions

Left

A	B
a	1
b	2
c	3

Right

C	D
a	1
a	2
b	3
b	4

SQL

```
select A,
  (select sum(D) as sm
  from Right
  where A = C)
from Left
```

A	sm
a	3
b	7
c	null

Datalog (Soufflé)

```
Q(x, sum y: {Right(x, y)}) :- Left(x,_).
```

A	sm
a	3
b	7
c	0

?

Conventions

SQL

```
select A,
  (select sum(D) as sm
  from Right
  where A = C)
from Left
```

A	sm
a	3
b	7
c	null

Queries

Left

A	B
a	1
b	2
c	3

Right

C	D
a	1
a	2
b	3
b	4

Datalog (Soufflé)

```
Q(x, sum y: {Right(x, y)}) :- Left(x,_).
```

A	sm
a	3
b	7
c	0

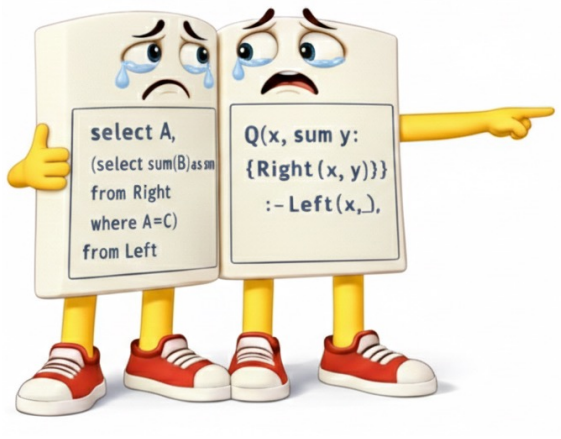
?

Conventions

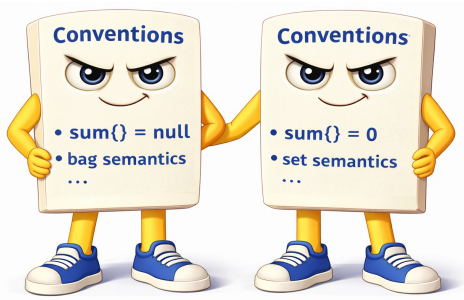
SQL

```
select A,
  (select sum(D) as sm
  from Right
  where A = C)
from Left
```

A	sm
a	3
b	7
c	null



Queries



Conventions

Left

A	B
a	1
b	2
c	3

Right

C	D
a	1
a	2
b	3
b	4

Datalog (Soufflé)

```
Q(x, sum y: {Right(x, y)}) :- Left(x,_).
```

A	sm
a	3
b	7
c	0

!

Language

orthogonal environment-level semantic parameters under which the query is interpreted

Query ↔ Convention

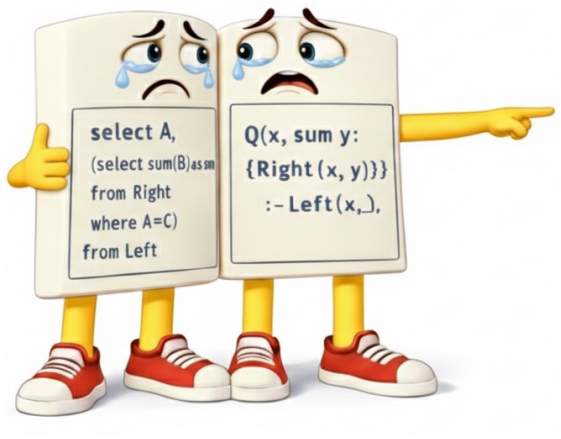
swappable

Conventions

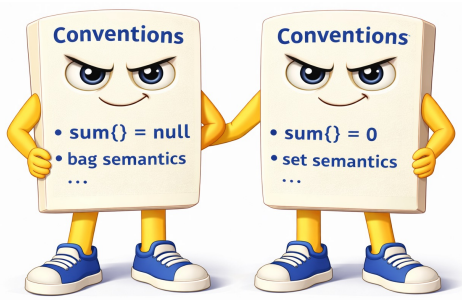
SQL

```
select A,
  (select sum(D) as sm
  from Right
  where A = C)
from Left
```

A	sm
a	3
b	7
c	null



Queries



Conventions

Left

A	B
a	1
b	2
c	3

Right

C	D
a	1
a	2
b	3
b	4

Datalog (Soufflé)

```
Q(x, sum y: {Right(x, y)}) :- Left(x,_).
```

A	sm
a	3
b	7
c	0

Language

orthogonal environment-level semantic parameters under which the query is interpreted

!

Query ↔ 3. Convention

swappable

1. Relational Intent

the relational entities & the compositional structure that determine intent

2. Modality

alternative representations of that intent tailored to different audiences

(Intent × Modality = Query) × Conventions = Semantics

orthogonal environment-level parameters under which the query is interpreted

Queries
(in various syntax)

```
select A,
  (select sum(D) as sm
  from Right
  where A = C)
from Left
```

Q(x, sum y: {Right(x, y)}) :- Left(x,_).

Conventions

Conventions
- sum({}) = 0
- set semantics
- ...

composable / modular through independence, separation of concerns, = factorization

Conventions
- sum({}) = null
- bag semantics
- ...

(Intent × Modality = Query) × Conventions = Semantics

orthogonal environment-level parameters under which the query is interpreted

Queries
(in various syntax)

```
select A,
  (select sum(D) as sm
  from Right
  where A = C)
from Left
```

Q(x, sum y: {Right(x, y)}) :- Left(x,_).

Collection Convention

Convention #539
- set semantics

Convention #538
- bag semantics

composable / modular through independence, separation of concerns, = factorization

Null Convention

Convention #1483
- sum({}) = 0

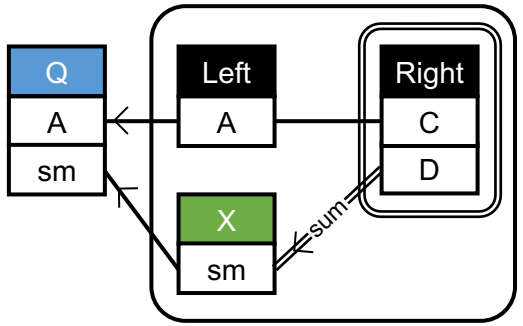
Convention #1484
- sum({}) = null

(Intent × Modality = Query) × Conventions = Semantics

the relational entities &
the compositional structure
that determine intent

orthogonal environment-level parameters
under which the query is interpreted

Relational Intent



Relational Diagram

Queries (in various syntax)

```
select A,
  (select sum(D) as sm
  from Right
  where A = C)
from Left
```

Q(x, sum y: {Right(x, y)}) :- Left(x,_).

Collection Convention

Convention #539
- set semantics

Convention #538
- bag semantics

Null Convention

Convention #1483
- sum({}) = 0

Convention #1484
- sum({}) = null

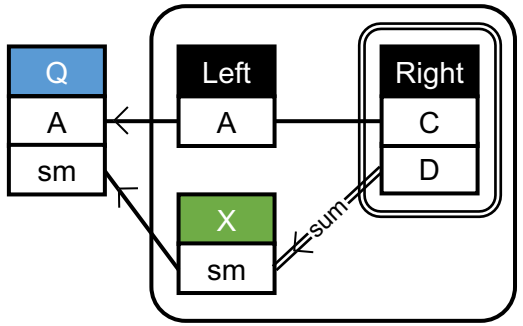
composable / modular
through independence,
separation of concerns,
= factorization

(Intent × Modality = Query) × Conventions = Semantics

the relational entities &
the compositional structure
that determine intent

orthogonal environment-level parameters
under which the query is interpreted

Relational Intent × Modality



Relational Diagram

```
COLLECTION
├─ HEAD: Q(A,sm)
└─ QUANTIFIER ∃
   ├─ BINDING: r ∈ R
   ├─ BINDING: x ∈
   │  └─ COLLECTION
   │     ├─ HEAD: X(sm)
   │     └─ SCOPING γ
   │        ├─ BINDING: s ∈ S
   │        ├─ GROUP BY: ∅
   │        └─ AND ∧
   │           ├─ PREDICATE: s.A = r.A
   │           └─ PREDICATE: X.sm = sum(s.B)
   └─ AND ∧
      ├─ PREDICATE: Q.A = r.A
      └─ PREDICATE: Q.sm = x.sm
```

Abstract Language Tree

Queries
(in various syntax)

```
select A,
  (select sum(D) as sm
  from Right
  where A = C)
from Left
```

Q(x, sum y: {Right(x, y)}) :- Left(x,_).

Collection Convention

Convention #539
- set semantics

Convention #538
- bag semantics

Null Convention

Convention #1483
- sum({}) = 0

Convention #1484
- sum({}) = null

composable / modular
through independence,
separation of concerns,
= factorization

Modality
alternative representations
of that intent tailored to
different audiences

Two debates that we analyze today:

(1) Is linear dataflow inherently : "easier to read" than inside-out?

(2) Are dataflow languages compo-sitional, and inside-out is not?

Inside-out (nesting/single-block) vs. pipelined dataflow

Claim: Pipelines are a lot <u>more intuitive</u> than SQL's "inside-out" dataflow.
Piped dataflow ≈ linear sequence of operators. Examples:

- DFQL ("*Dataflow QL*") [Clark,Wu'94] ("*ease-of-use*")
- PigLatin [Olston+'08] (*dataflow language, step-by-step where each step carries out a single data transformation, easier for programmers, explicit intermediate results*)
- Dataframes, such as Pandas/Python, or dplyr/R (*operator chaining*)
- FunSQL.jl
- PRQL ("*Pipelined Relational QL*" = "Prequel")
- Malloy
- SaneQL [Neumann, Leis'24]
- Google's SQL Pipe Syntax [Shute+'24]

[Olston+'08] Pig Latin: A Not-So-Foreign Language for Data Processing, SIGMOD 2008. https://doi.org/10.1145/1376616.1376726 [PRQL'22]: a simple, powerful, pipelined SQL replacement. https://prql-lang.org/ [Malloy'22]: A modern open source language for analyzing, transforming, and modeling data. https://www.malloydata.dev/ [FunSQL'21]: a Julia library for compositional construction of SQL queries. https://github.com/MechanicalRabbit/FunSQL.jl [Neumann, Leis'24] A Critique of Modern SQL And A Proposal Towards A Simple and Expressive Query Language, CIDR 2024, https://mail.vldb.org/cidrdb/papers/2024/p48-neumann.pdf [Shute+'24] SQL Has Problems. We Can Fix Them: Pipe Syntax In SQL, PVLDB. https://vldb.org/pvldb/vol17/p4051-shute.pdf

Inside-out (nesting/single-block) vs. pipelined dataflow

Claim: Pipelines are a lot <u>more intuitive</u> than SQL's "inside-out" dataflow.
Piped dataflow ≈ linear sequence of operators. Examples:

- DFQL ("*Dataflow QL*") [Clark,Wu'94] ("*ease-of-use*")
- PigLatin [Olston+'08] (*dataflow language, step-by-step where each step carries out a single data transformation, easier for programmers, explicit intermediate results*)
- Dataframes, such as Pandas/Python, or dplyr/R (*operator chaining*)
- FunSQL.jl
- PRQL ("*Pipelined Relational QL*" = "Prequel")
- Malloy
- **SaneQL** [Neumann, Leis'24]
- **Google's SQL pipe syntax** [Shute+'24]

allows parameterized functions (thus not strict "pipeline")

(extend instead of replace SQL)

[Olston+'08] Pig Latin: A Not-So-Foreign Language for Data Processing, SIGMOD 2008. https://doi.org/10.1145/1376616.1376726 [PRQL'22]: a simple, powerful, pipelined SQL replacement. https://prql-lang.org/ [Malloy'22]: A modern open source language for analyzing, transforming, and modeling data. https://www.malloydata.dev/ [FunSQL'21]: a Julia library for compositional construction of SQL queries. https://github.com/MechanicalRabbit/FunSQL.jl [Neumann, Leis'24] A Critique of Modern SQL And A Proposal Towards A Simple and Expressive Query Language, CIDR 2024, https://mail.vldb.org/cidrdb/papers/2024/p48-neumann.pdf [Shute+'24] SQL Has Problems. We Can Fix Them: Pipe Syntax In SQL, PVLDB. https://vldb.org/pvldb/vol17/p4051-shute.pdf

Simplified TPC-H13 (customer distribution) : Inside-out vs. pipelines

SQL

```
select custct, count(*) as custdist
from
  (select C.custkey, count(O.orderkey) as custct
  from customer C
  left outer join orders O
  on C.custkey=O.custkey
  and O.comment not like '%unusual%'
  group by C.custkey) as X
group by custct
```

You do *not* need to understand the queries to follow the argument!
Just follow the flow!

Claim: A piped linear syntax is easier-to-understand then "inside-out"

Pipe syntax

```
from customer as C
|> left outer join orders as O
   on C.custkey = O.custkey
   and O.comment not like '%unusual%'
|> aggregate count(O.orderkey) as custct
   group by C.custkey
|> aggregate count(*) as custdist
   group by custct
```

Shouldn't grouping come before aggregation, as in PigLatin?

A simplified variant of TPC-H query 13 (schema simplified, order removed), used by Jeff Shute in his talk at Andy Pavlo's seminar: https://www.youtube.com/watch?v=ZIkjAQ7AwSM

"A piped linear syntax is easier to understand than inside-out"

SQL

```
select custct, count(*) as custdist
from
  (select C.custkey, count(O.orderkey) as custct
  from customer C
  left outer join orders O
  on C.custkey=O.custkey
  and O.comment not like '%unusual%'
  group by C.custkey) as X
group by custct
```

Claim: A piped linear syntax is easier-to-understand then "inside-out"

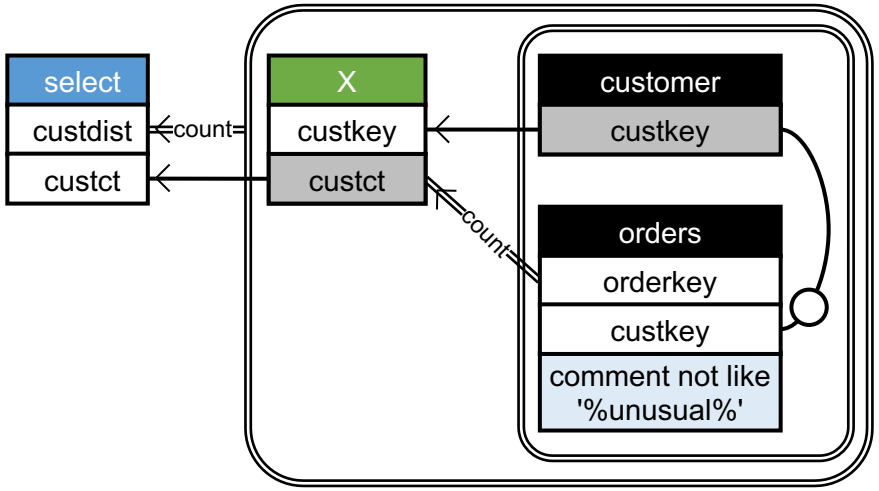
Pipe syntax

```
from customer as C
|> left outer join orders as O
   on C.custkey = O.custkey
   and O.comment not like '%unusual%'
|> aggregate count(O.orderkey) as custct
   group by C.custkey
|> aggregate count(*) as custdist
   group by custct
```

"A piped linear syntax is easier to understand than inside-out"

SQL

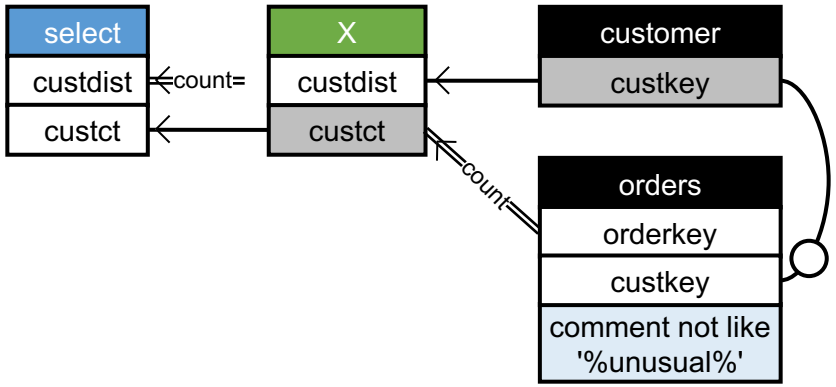
```
select custct, count(*) as custdist
from
  (select C.custkey, count(O.orderkey) as custct
  from customer C
  left outer join orders O
  on C.custkey=O.custkey
  and O.comment not like '%unusual%'
  group by C.custkey) as X
group by custct
```



Claim: A piped linear syntax is easier-to-understand then "inside-out"

Pipe syntax

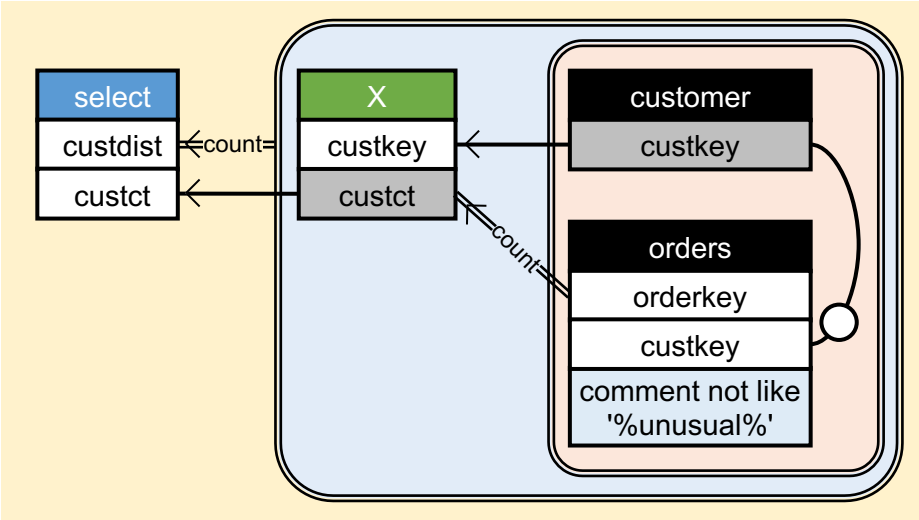
```
from customer as C
|> left outer join orders as O
   on C.custkey = O.custkey
   and O.comment not like '%unusual%'
|> aggregate count(O.orderkey) as custct
   group by C.custkey
|> aggregate count(*) as custdist
   group by custct
```



"A piped linear syntax is easier to understand than inside-out"

SQL

```
select custct, count(*) as custdist
from
  (select C.custkey, count(O.orderkey) as custct
  from customer C
  left outer join orders O
  on C.custkey=O.custkey
  and O.comment not like '%unusual%'
  group by C.custkey) as X
group by custct
```

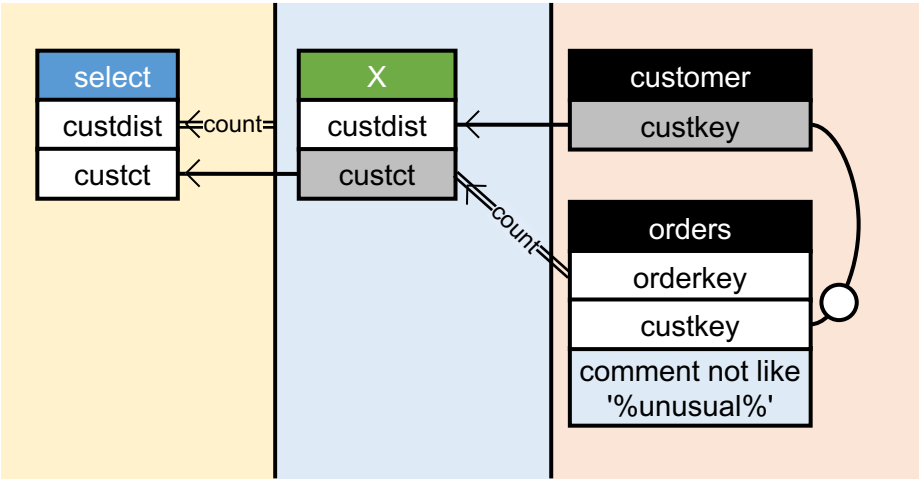


Claim: A piped linear syntax is easier-to-understand then "inside-out"

?

Pipe syntax

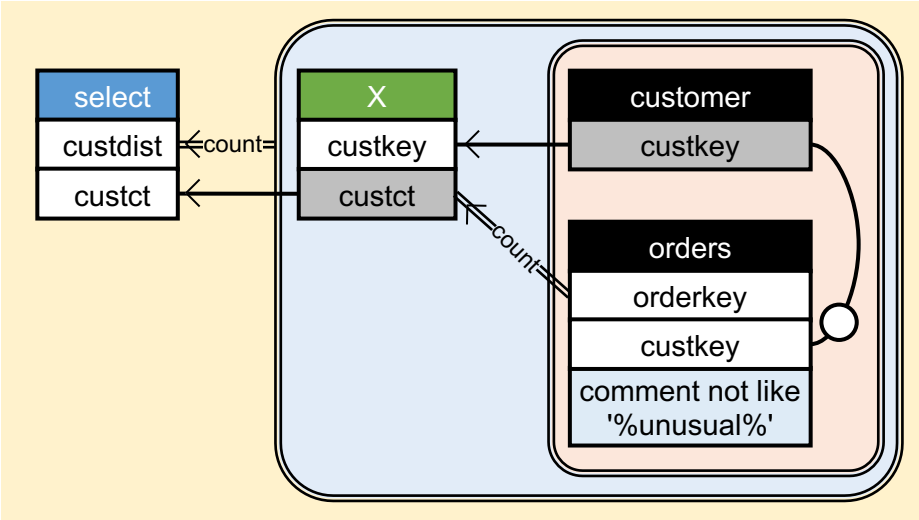
```
from customer as C
|> left outer join orders as O
   on C.custkey = O.custkey
   and O.comment not like '%unusual%'
|> aggregate count(O.orderkey) as custct
   group by C.custkey
|> aggregate count(*) as custdist
   group by custct
```



"A piped linear syntax is easier to understand than inside-out"

SQL

```
select custct, count(*) as custdist
from
  (select C.custkey, count(O.orderkey) as custct
  from customer C
  left outer join orders O
  on C.custkey=O.custkey
  and O.comment not like '%unusual%'
  group by C.custkey) as X
group by custct
```

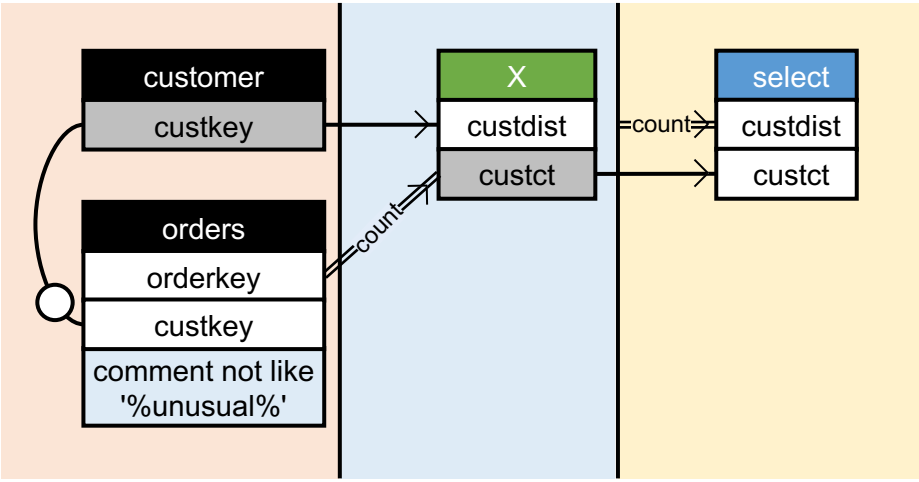


Claim: A piped linear syntax is easier-to-understand then "inside-out"

?

Pipe syntax

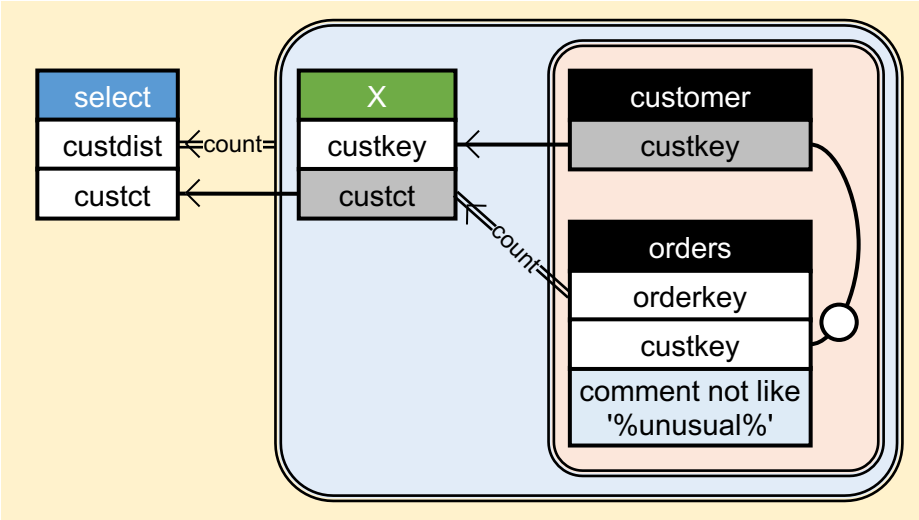
```
from customer as C
|> left outer join orders as O
   on C.custkey = O.custkey
   and O.comment not like '%unusual%'
|> aggregate count(O.orderkey) as custct
   group by C.custkey
|> aggregate count(*) as custdist
   group by custct
```



"A piped linear syntax is easier to understand than inside-out"

SQL

```
select custct, count(*) as custdist
from
  (select C.custkey, count(O.orderkey) as custct
  from customer C
  left outer join orders O
  on C.custkey=O.custkey
  and O.comment not like '%unusual%'
  group by C.custkey) as X
group by custct
```



Claim: A piped linear syntax is easier-to-understand then "inside-out"

Pipe syntax

```
from customer as C
|> left outer join orders as O
   on C.custkey = O.custkey
   and O.comment not like '%unusual%'
|> aggregate count(O.orderkey) as custct
   group by C.custkey
|> aggregate count(*) as custdist
   group by custct
```

Counter-claim:

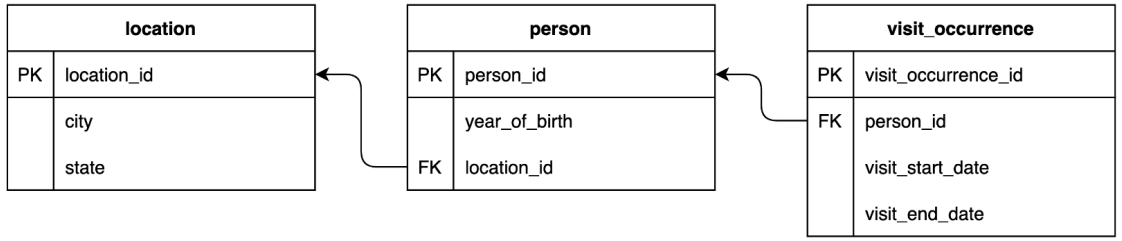
- easy-to-understand depends on "modality", not piped linear syntax vs. "inside-out"

1. Pipes (in text) reduce "split-attention"

2. but they also need nestings and can't avoid "split attention" (partial orders / DAGs)

Linearizing partial orders also leads to nesting

Q: When was the last time each person born between 1930 and 1940 and living in Illinois was seen by a healthcare provider?



You do *not* need to understand the queries to follow the argument! Just follow the flow!

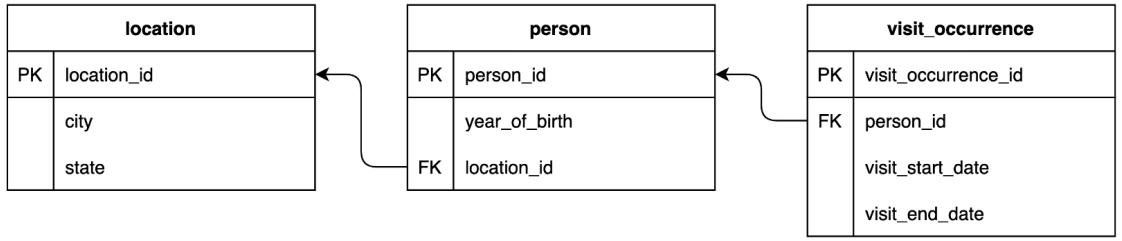
(I also have the full SQL query larger in a backup slide)

"FunSQL"

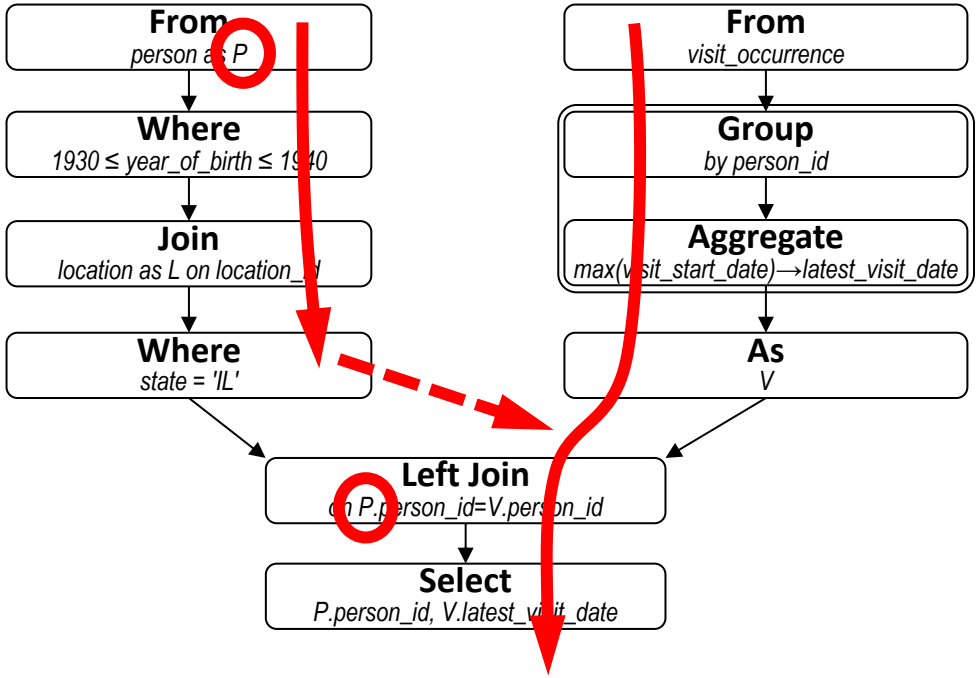
Query and taken from [FunSQL'21]: a Julia library for compositional construction of SQL queries. https://github.com/MechanicalRabbit/FunSQL.jl, translated into Google's SQL pipe syntax

Linearizing partial orders also leads to nesting

Q: When was the last time each person born between 1930 and 1940 and living in Illinois was seen by a healthcare provider?




```
from person as P
|> where year_of_birth >= 1930
   and year_of_birth <= 1940
|> join location as L
   on P.location_id = L.location_id
|> where L.state = 'IL'
|> left join (
     from visit_occurrence
     |> aggregate max(visit_start_date)
        as latest_visit_date
        group by person_id) as V
   on P.person_id = V.person_id
|> select P.person_id, V.latest_visit_date;
```

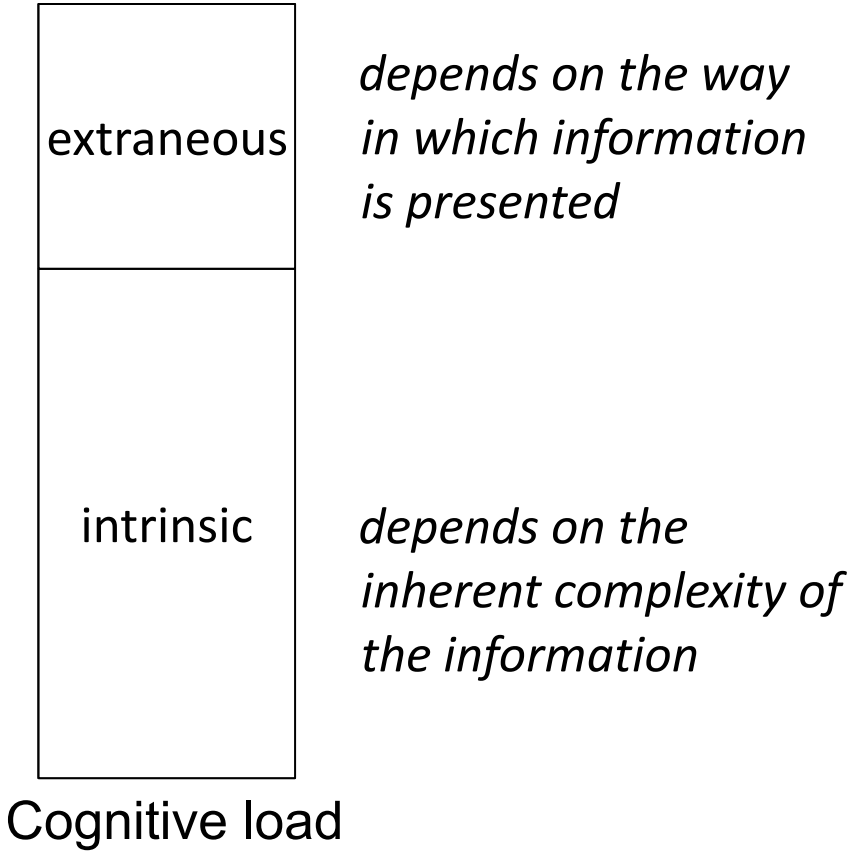


Serialization of a partial order requires jumping around via some reference (define and re-use later): this cannot be avoided!

Cognitive load theory: "split-attention" when two related pieces of information are spatially separated

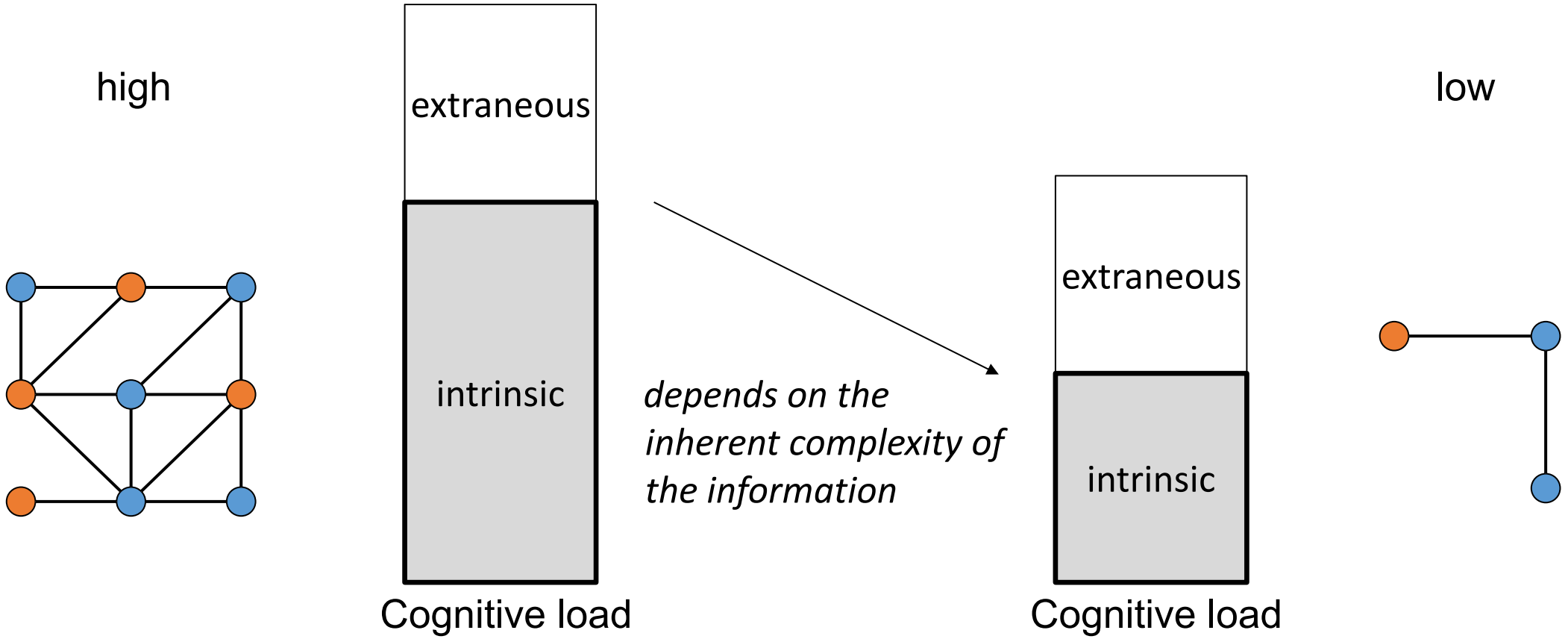
Query and taken from [FunSQL'21]: a Julia library for compositional construction of SQL queries. https://github.com/MechanicalRabbit/FunSQL.jl, translated into Google's SQL pipe syntax

Cognitive load: intrinsic vs extraneous



Simplified take-away from a large body of work on split-attention: "Cognitive load during problem solving: Effects on learning", Sweller, 1988. https://doi.org/10.1016/0364-0213(88)90023-7

Cognitive load: intrinsic vs extraneous



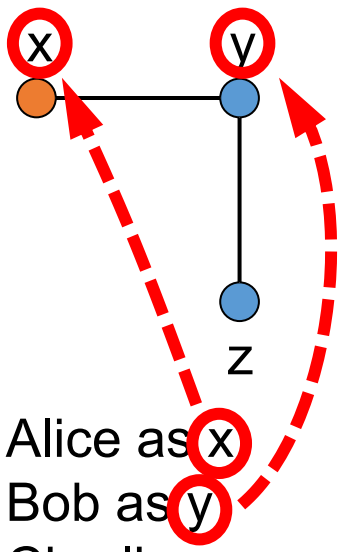
Simplified take-away from a large body of work on split-attention: "Cognitive load during problem solving: Effects on learning", Sweller, 1988. https://doi.org/10.1016/0364-0213(88)90023-7

Cognitive load: intrinsic vs extraneous

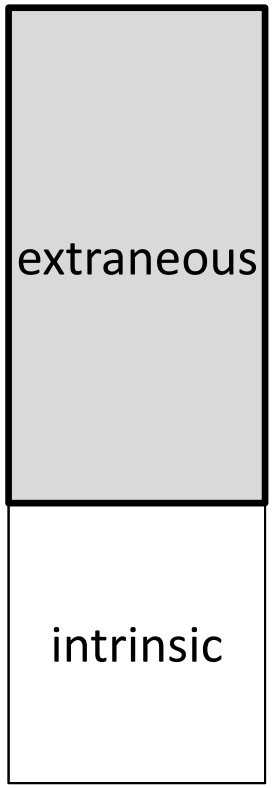
split-attention: when attention is split between spatially separated pieces of information

spatial contiguity: when related information is near one another

high

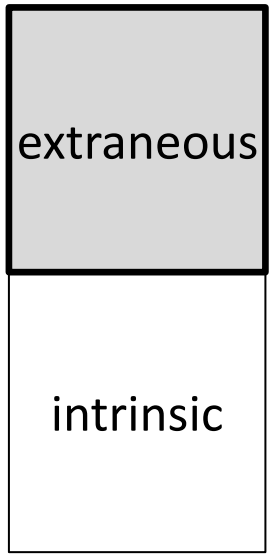


Alice as x
Bob as y
Charlie as z



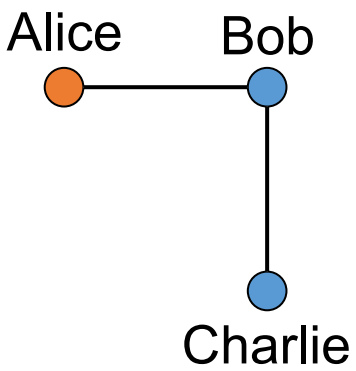
Cognitive load

depends on the way in which information is presented



Cognitive load

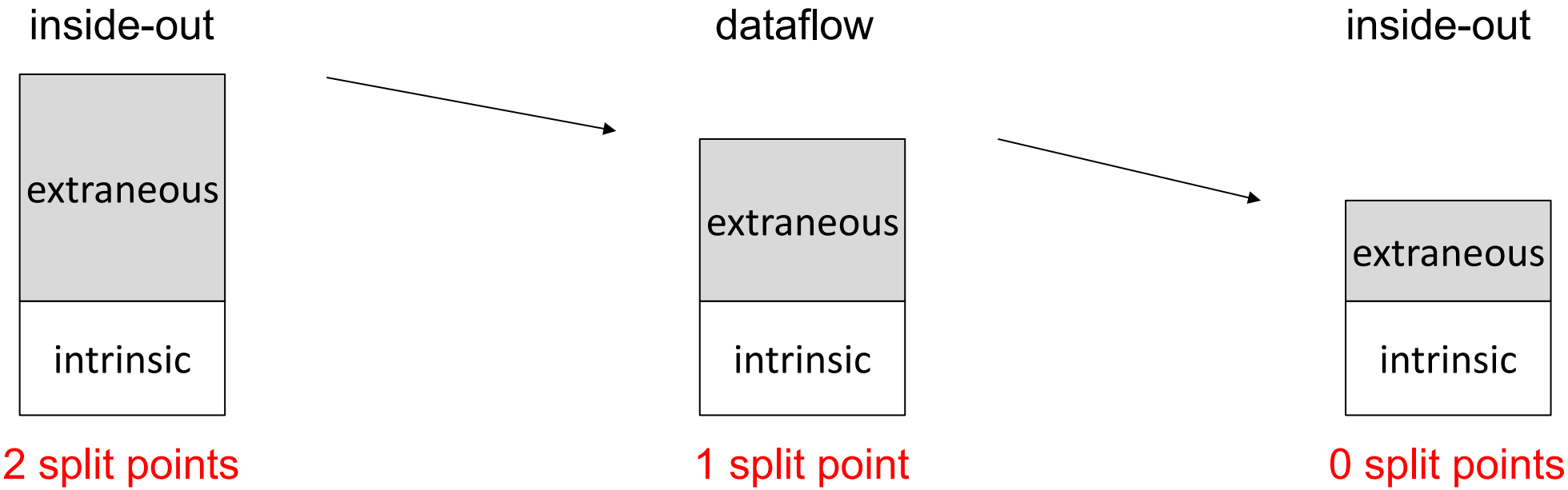
low



Simplified take-away from a large body of work on split-attention: "Cognitive load during problem solving: Effects on learning", Sweller, 1988. https://doi.org/10.1016/0364-0213(88)90023-7

Pipelines reduce split-points, diagrams remove them

Easy-to-understand depends on "modality" first, and piped linear syntax vs. "inside-out" second



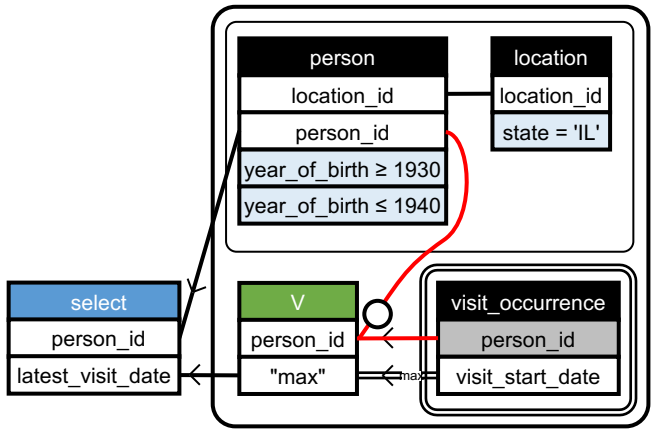
SQL (text)

```
select P.person_id, V."max" as latest_visit_date
from
  (select person_id, location_id
  from person
  where year_of_birth >= 1930
  and year_of_birth <= 1940) as P
join
  (select location_id
  from location
  where state = 'IL') as L
on P.location_id = L.location_id
left join(
  select max(visit_start_date) as "max", person_id
  from visit_occurrence
  group by person_id) as V
on P.person_id = V.person_id
```

Pipe syntax (text)

```
from person as P
|> where year_of_birth >= 1930
   and year_of_birth <= 1940
|> join location as L
   on P.location_id = L.location_id
|> where L.state = 'IL'
|> left join (
     from visit_occurrence
     |> aggregate max(visit_start_date)
        as latest_visit_date
        group by person_id) as V
   on P.person_id = V.person_id
|> select P.person_id,
V.latest_visit_date;
```

Abstract Relational Calculus (diagram)



For usability: modality instead of syntax!

Easy-to-understand depends on "modality" first, and piped linear syntax vs. "inside-out" second

inside-out | dataflow | inside-out

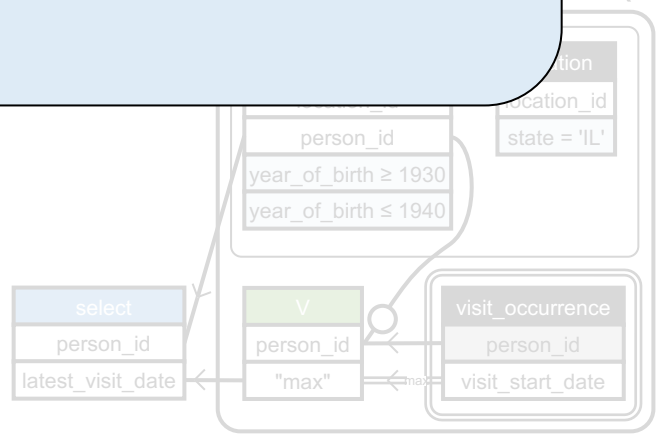
extraneous | extraneous | extraneous

- Easy-to-understand = minimizing split points
- Dataflow has indeed fewer split points than inside-out *in text*
- But to minimize split points → go into 2D → Diagrams = different modality, not different syntax!

SQL (te | us (diagram)

```
select P.pers
from
  (select person
  from person
  where year_of_birth >= 1930
  and year_of_birth <= 1940) as P
join
  (select location_id
  from location
  where state = 'IL') as L
on P.location_id = L.location_id
left join(
  select max(visit_start_date) as "max", person_id
  from visit_occurrence
  group by person_id) as V
on P.person_id = V.person_id
```

```
|> join location as L
   on P.location_id = L.location_id
|> where L.state = 'IL'
|> left join (
     from visit_occurrence
     |> aggregate max(visit_start_date)
        as latest_visit_date
     group by person_id) as V
   on P.person_id = V.person_id
|> select P.person_id,
V.latest_visit_date;
```

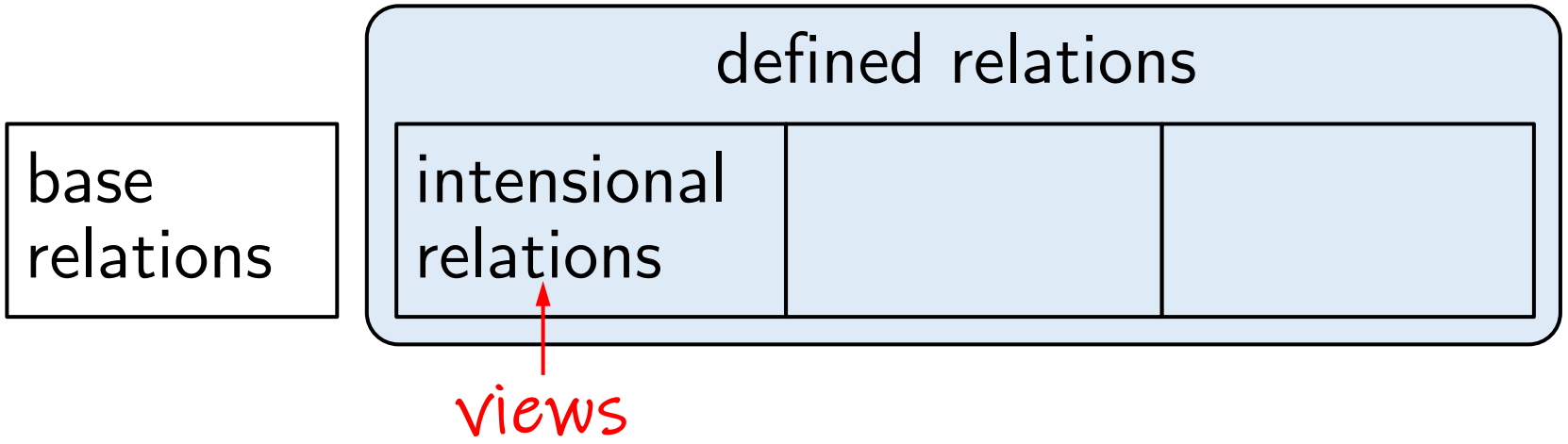


Two debates that we analyze today:

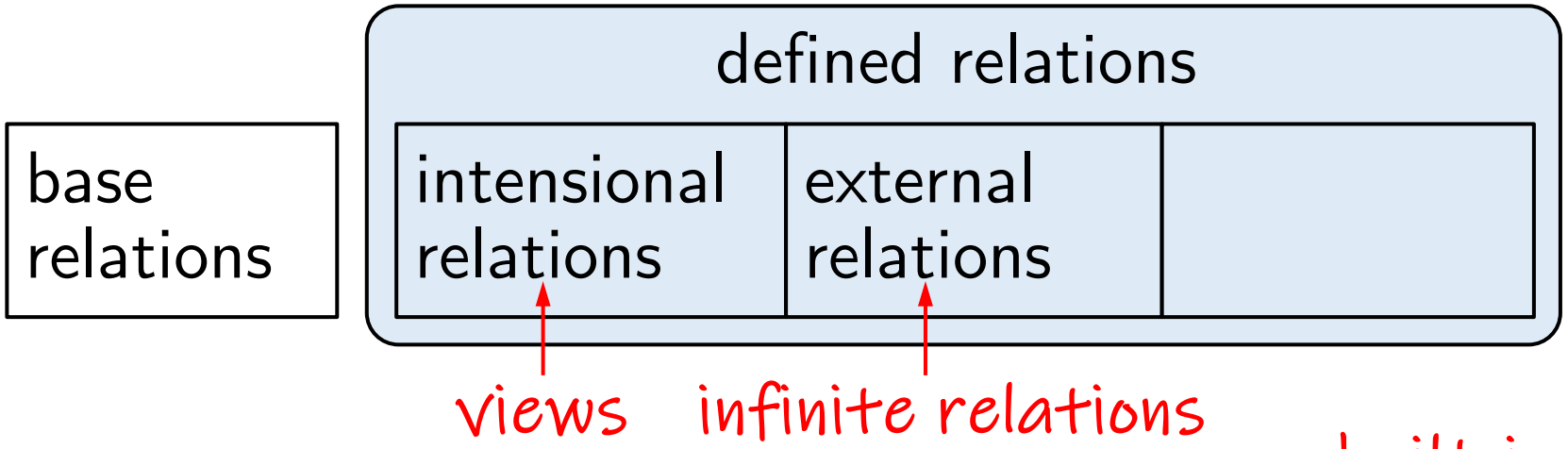
(1) Is linear dataflow inherently : "easier to read" than inside-out?

(2) Are dataflow languages compo-sitional, and inside-out is not?

4 Types of Relations in Abstract Relational Calculus (ARC)



4 Types of Relations in Abstract Relational Calculus (ARC)



defined outside relational model

built-in predicates

< (smaller)

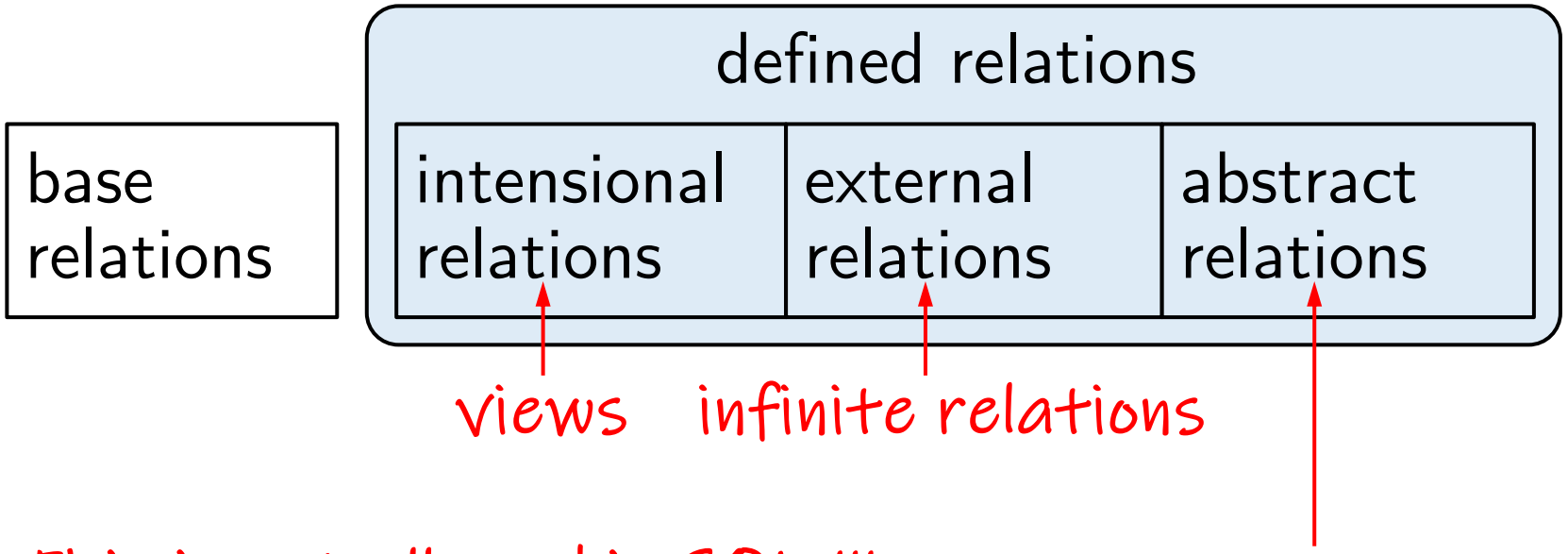
left	right
0	1
1	2
1.23	1.24
-2	-1
...	...

relationalized functions

* (multiplication)

$1	$2	out
1	0	0
2	3	6
7	8	56
5.2	2.5	13
...	...	...

4 Types of Relations in Abstract Relational Calculus (ARC)



This is not allowed in SQL !!!

```
select L1.drinker as left,
       L2.drinker as right
where not exists
  (select 1
  from Likes L1
  and not exists
    (select 1
    from Likes L2
    where L2.drinker = L1.drinker))
```

may not have a well-defined extension on their own

defined within a relational language

serve to modularize a query

Find CIDR drinkers w/ a unique whisky taste

Likes(drinker,whisky)

Alice: {1}
Bob: {1,2}
Charly: {1,2}
Dora: {1,2,3}

not 1NF

Likes

drinker	whisky
A	1
B	1
B	2
C	1
C	2
D	1
D	2
D	3

1NF

?

Find CIDR drinkers w/ a unique whisky taste

Likes(drinker,whisky)

```
select distinct L1.drinker
from Likes L1
where not exists
  (select 1
  from Likes L2
  where L1.drinker <> L2.drinker
  and not exists
    (select 1
    from Likes L3
    where L3.drinker = L2.drinker
    and not exists
      (select 1
      from Likes L4
      where L4.drinker = L1.drinker
      and L4.whisky = L3.whisky))
  and not exists
    (select 1
    from Likes L5
    where L5. drinker = L1. drinker
    and not exists
      (select 1
      from Likes L6
      where L6.drinker = L2.drinker
      and L6.whisky = L5.whisky)))
```

Alice: {1}
Bob: {1,2}
Charly: {1,2}
Dora: {1,2,3}

not 1NF

pipelines won't help

Likes

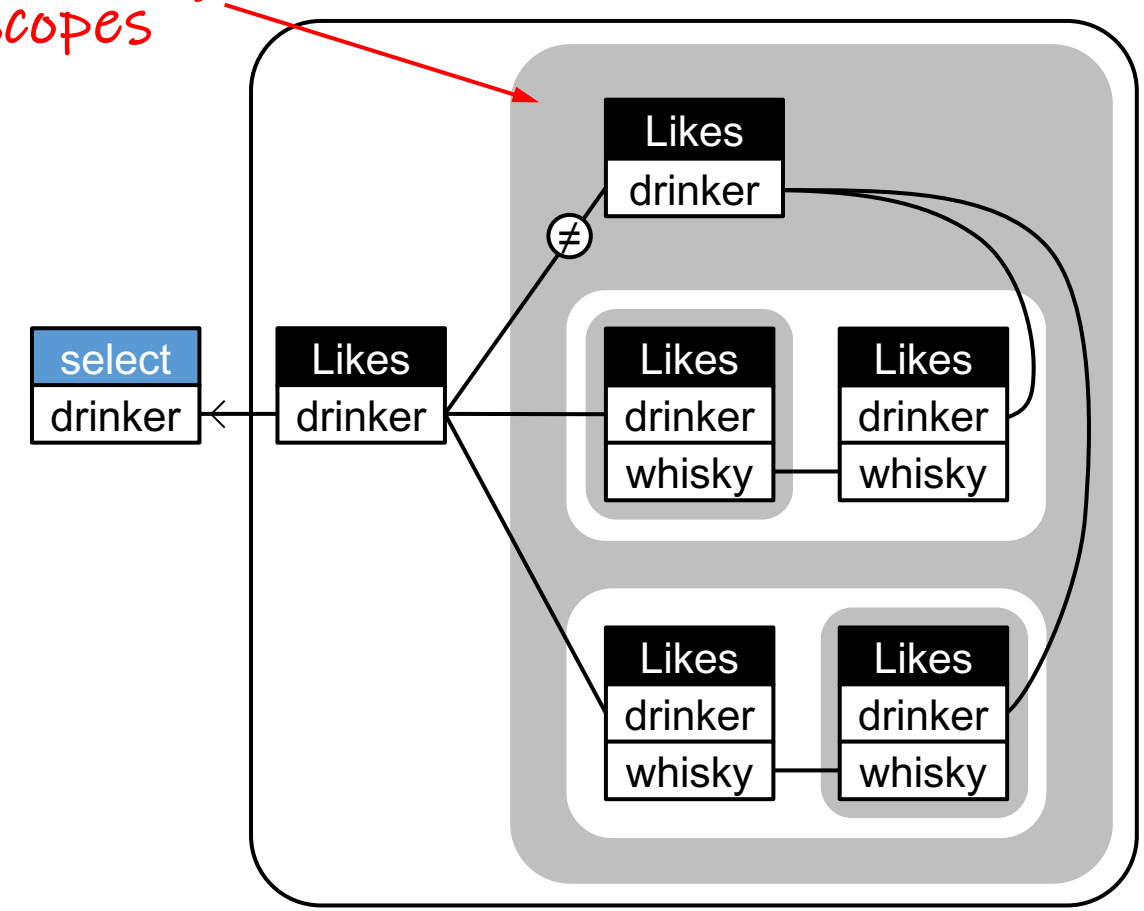
drinker	whisky
A	1
B	1
B	2
C	1
C	2
D	1
D	2
D	3

Find CIDR drinkers w/ a unique whisky taste

Likes(drinker,whisky)

alternating shades of gray show negation scopes

```
select distinct L1.drinker
from Likes L1
where not exists
  (select 1
  from Likes L2
  where L1.drinker <> L2.drinker
  and not exists
    (select 1
    from Likes L3
    where L3.drinker = L2.drinker
    and not exists
      (select 1
      from Likes L4
      where L4.drinker = L1.drinker
      and L4.whisky = L3.whisky))
  and not exists
    (select 1
    from Likes L5
    where L5. drinker = L1. drinker
    and not exists
      (select 1
      from Likes L6
      where L6.drinker = L2.drinker
      and L6.whisky = L5.whisky)))
```

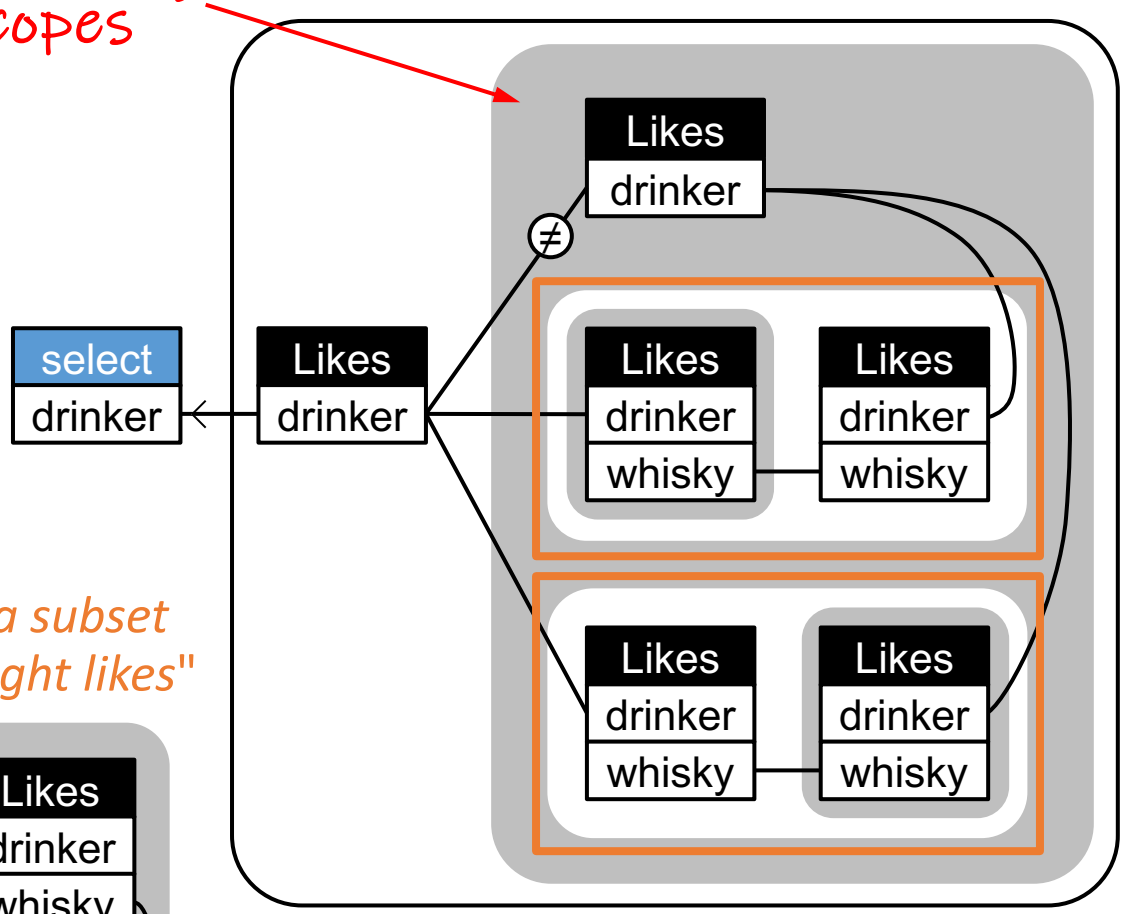


Find CIDR drinkers w/ a unique whisky taste

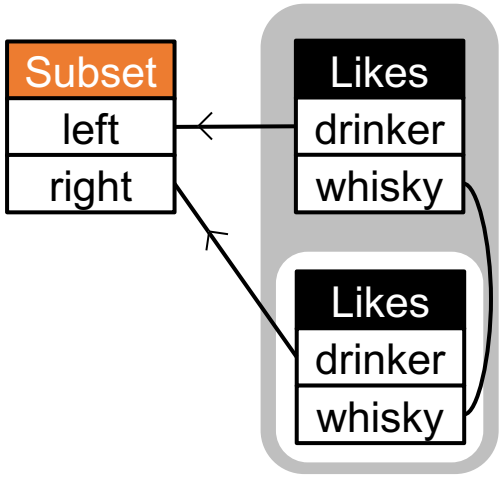
Likes(drinker,whisky)

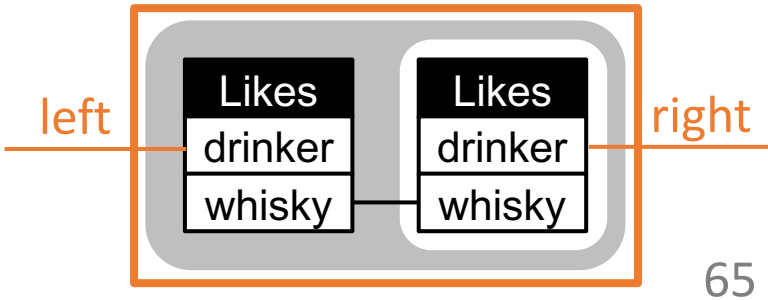
```
select distinct L1.drinker
from Likes L1
where not exists
  (select 1
  from Likes L2
  where L1.drinker <> L2.drinker
  and not exists
    (select 1
    from Likes L3
    where L3.drinker = L2.drinker
    and not exists
      (select 1
      from Likes L4
      where L4.drinker = L1.drinker
      and L4.whisky = L3.whisky))
  and not exists
    (select 1
    from Likes L5
    where L5. drinker = L1. drinker
    and not exists
      (select 1
      from Likes L6
      where L6.drinker = L2.drinker
      and L6.whisky = L5.whisky)))
```

alternating shades of gray show negation scopes



"left drinker likes a subset of whiskies that right likes"

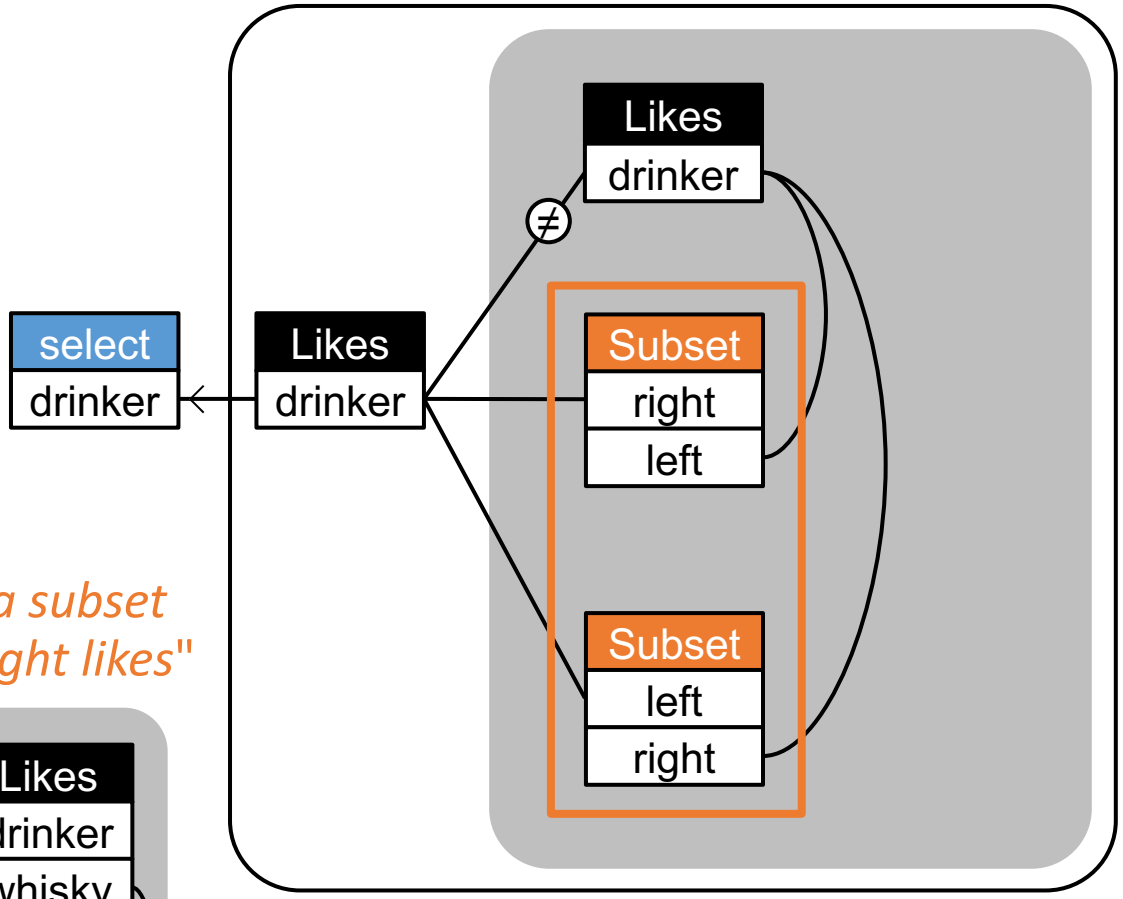




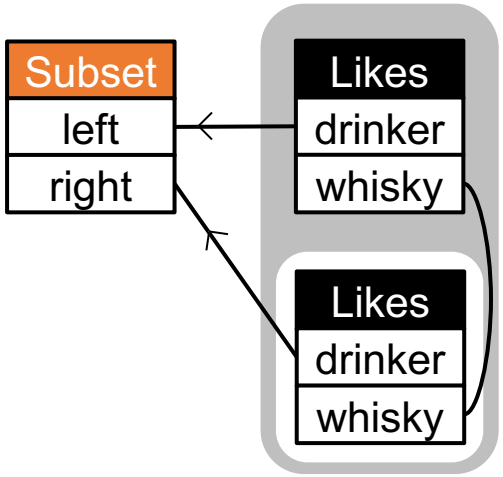
Find CIDR drinkers w/ a unique whisky taste

Likes(drinker,whisky)

```
select distinct L1.drinker
from Likes L1
where not exists
  (select 1
  from Likes L2, Subset S1, Subset S2
  where L1.drinker <> L2.drinker
  and S1.left = L2.drinker
  and S1.right = L1.drinker
  and S1.left = L1.drinker
  and S1.right = L2.drinker
                                    )
```

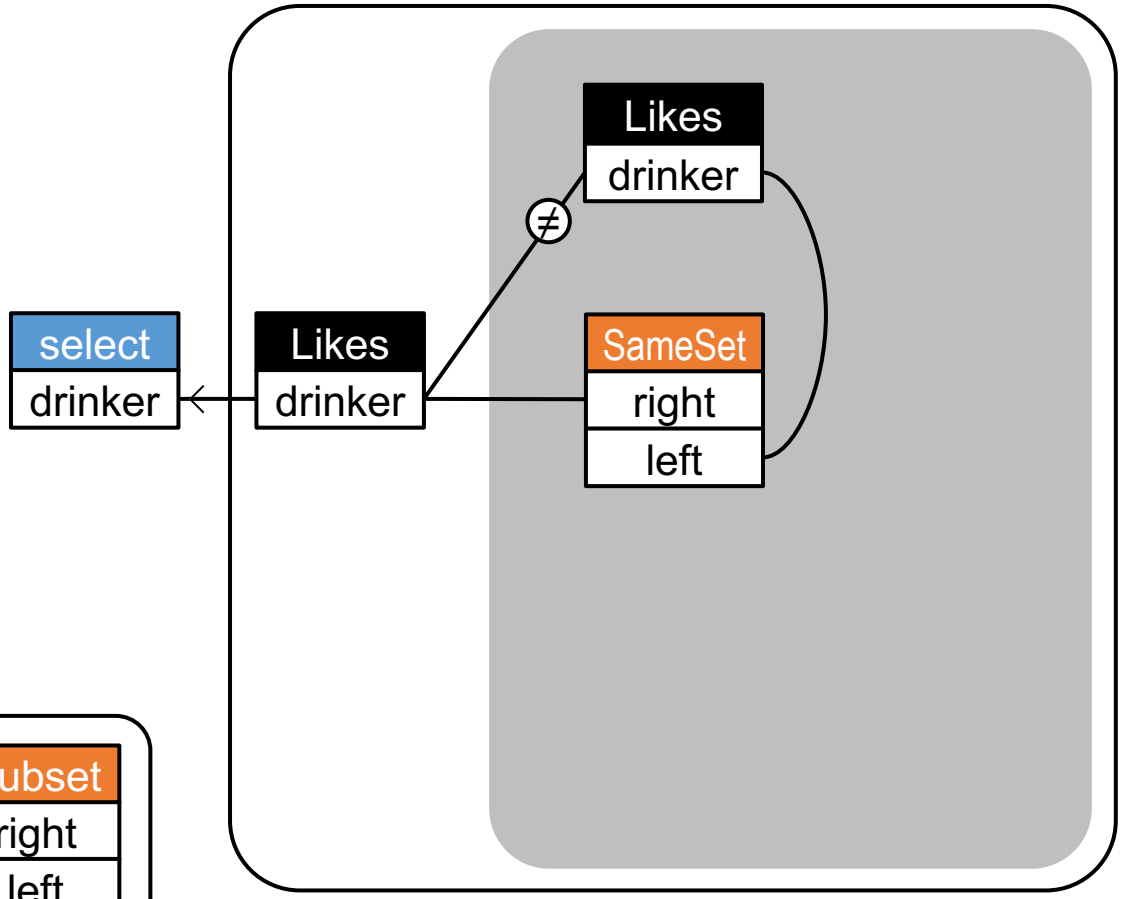


"left drinker likes a subset of whiskies that right likes"

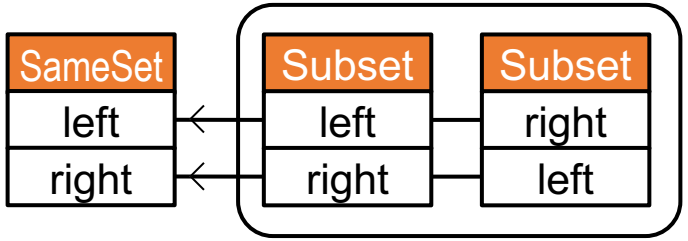


Find CIDR drinkers w/ a unique whisky taste Likes(drinker,whisky)

```
select distinct L1.drinker
from Likes L1
where not exists
  (select 1
  from Likes L2, SameSet S
  where L1.drinker <> L2.drinker
  and S.left = L2.drinker
  and S.right = L1.drinker)
```



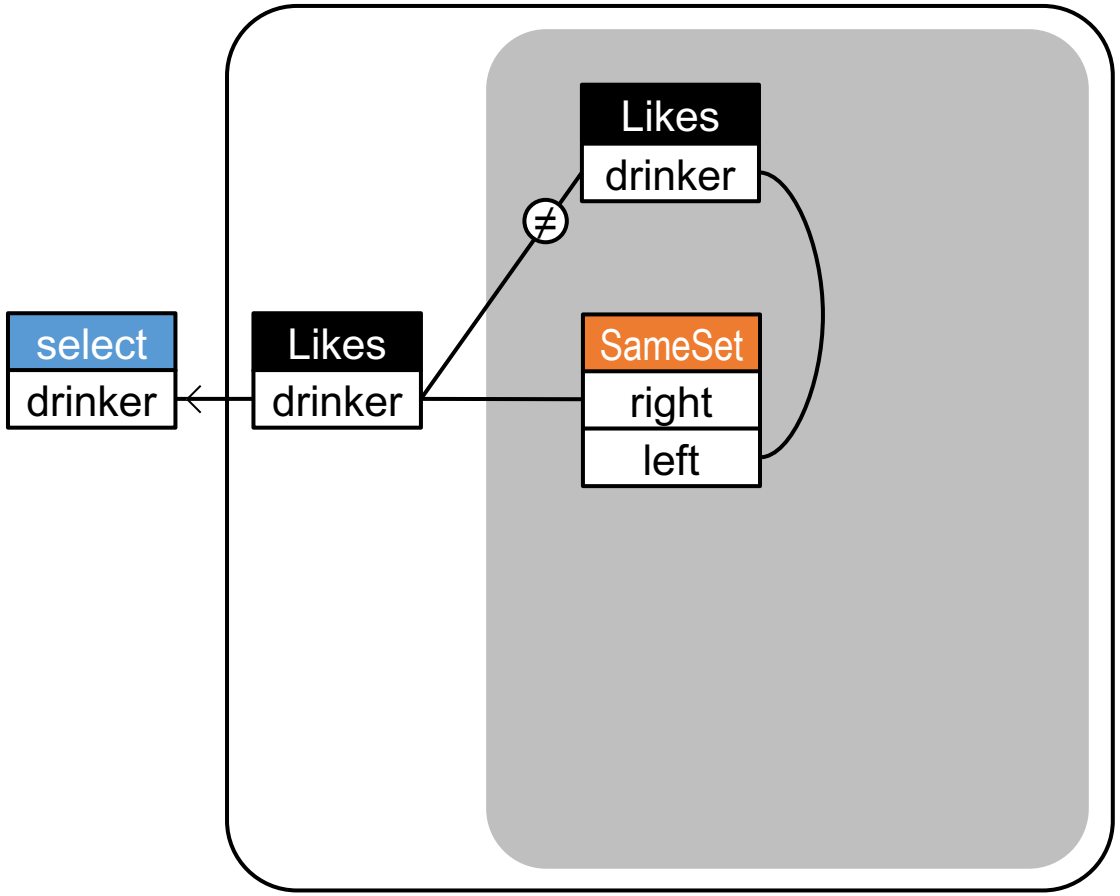
"pairs of drinkers who like the same set of whiskies"



Find CIDR drinkers w/ a unique whisky taste

Likes(drinker,whisky)

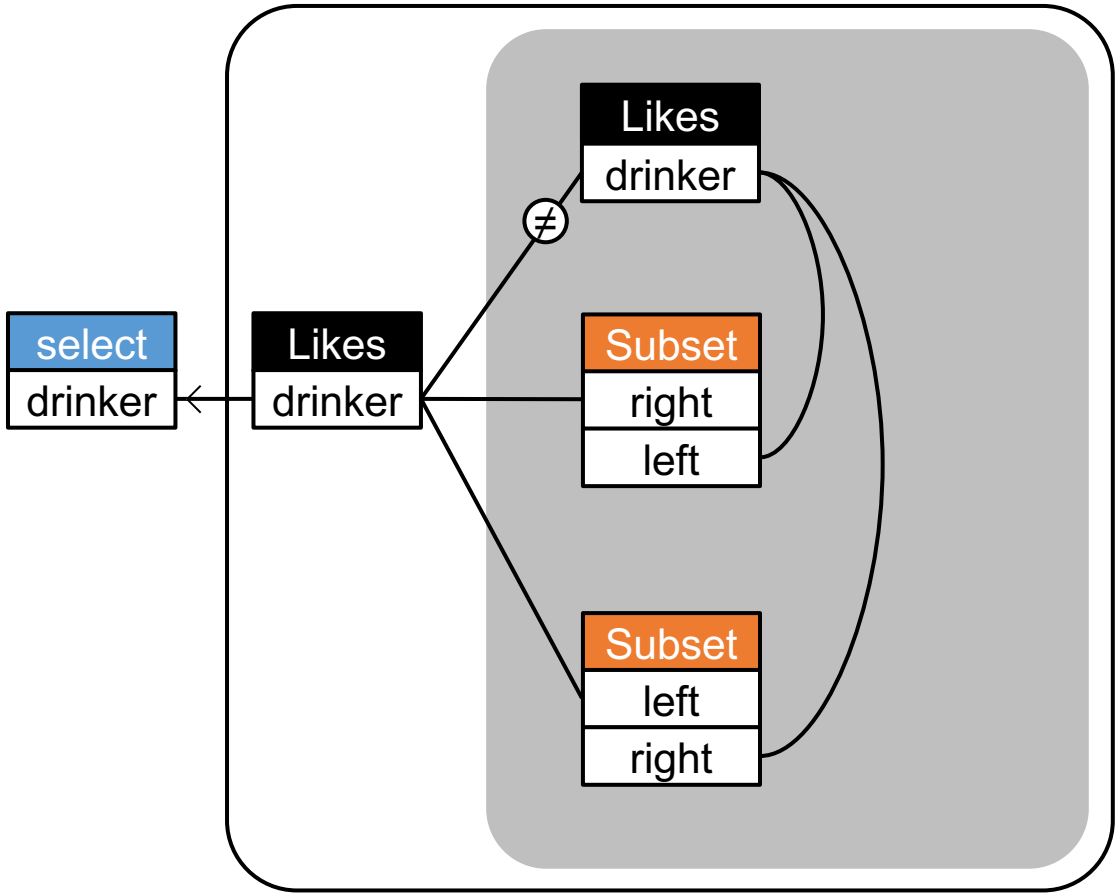
```
select distinct L1.drinker
from Likes L1
where not exists
  (select 1
  from Likes L2, SameSet S
  where L1.drinker <> L2.drinker
  and S.left = L2.drinker
  and S.right = L1.drinker)
```



Find CIDR drinkers w/ a unique whisky taste

Likes(drinker,whisky)

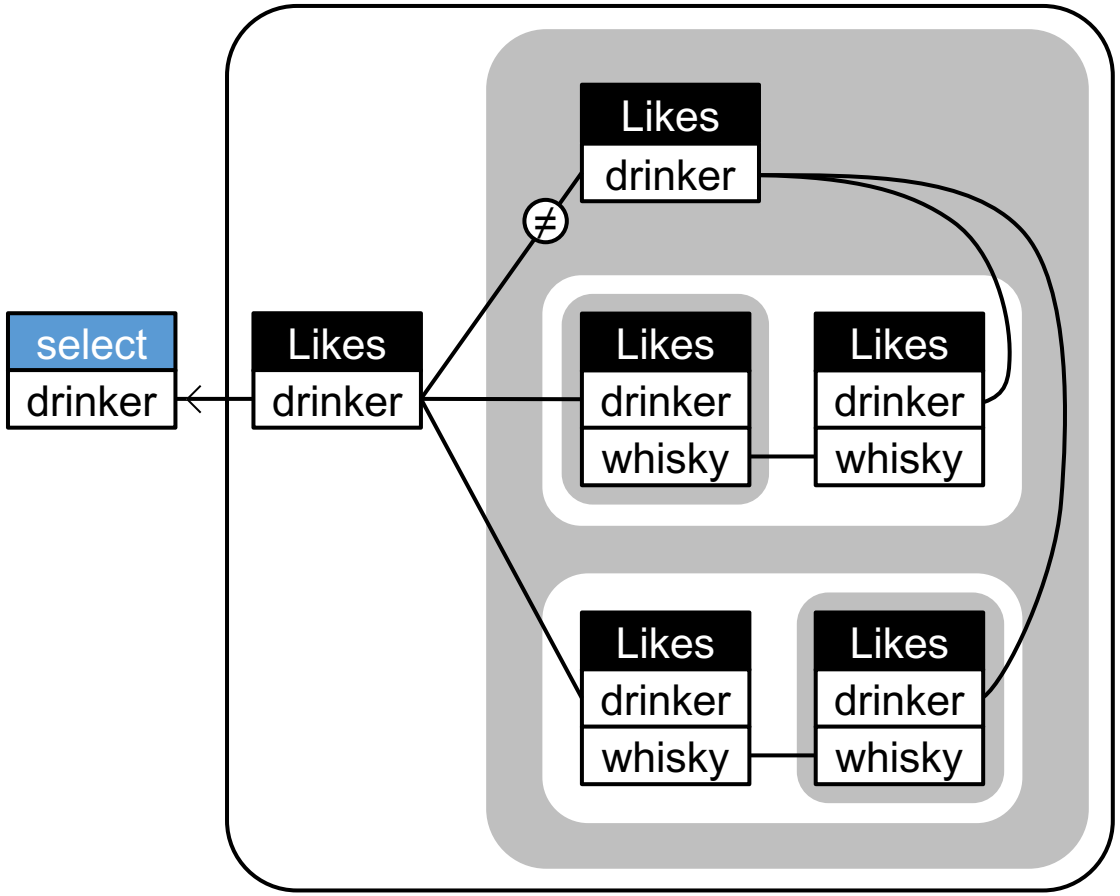
```
select distinct L1.drinker
from Likes L1
where not exists
  (select 1
  from Likes L2, Subset S1, Subset S2
  where L1.drinker <> L2.drinker
  and S1.left = L2.drinker
  and S1.right = L1.drinker




  and S1.left = L1.drinker
  and S1.right = L2.drinker



)
```



Find CIDR drinkers w/ a unique whisky taste

Likes(drinker,whisky)

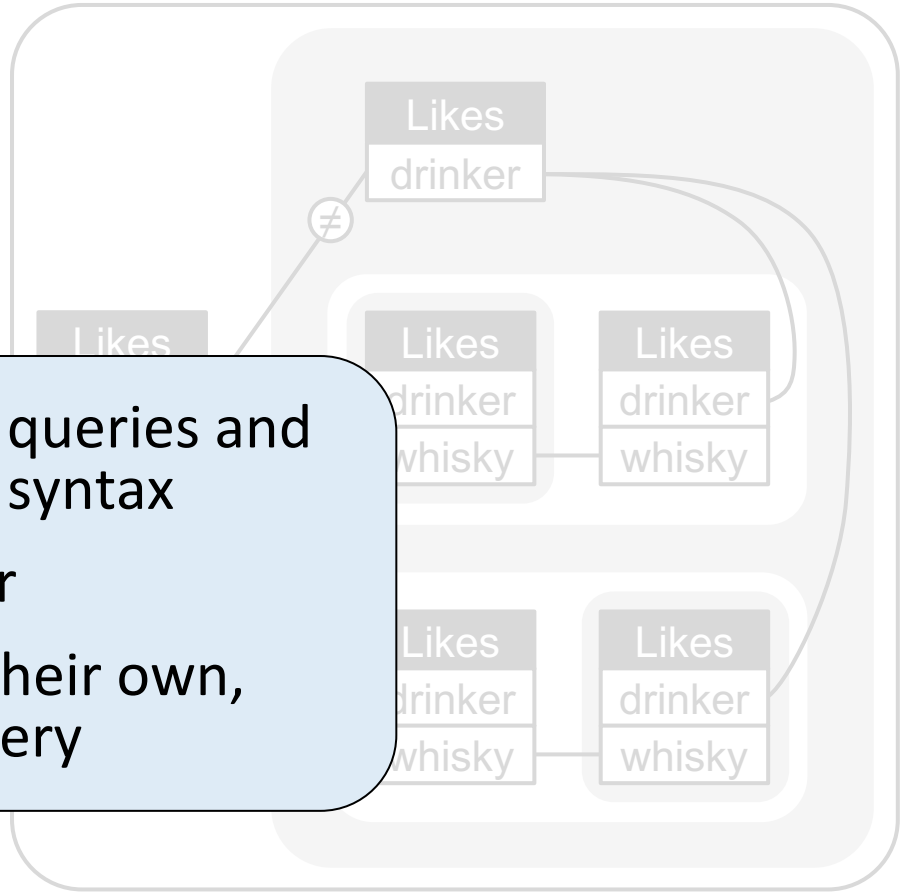
```
select distinct L1.drinker
from Likes L1
where not exists
  (select 1
  from Likes L2
  where L1.drinker <> L2.drinker
  and not exists
    (select 1
    from Likes L3
    where L3.drinker = L2.drinker
    and not exists
      (select 1
      from Likes L4
      where L4.drinker = L1.drinker
      and L4.whisky = L3.whisky))
  and not exists
    (select 1
    from Likes L5
    where L5. drinker = L1. drinker
    and not exists
      (select 1
      from Likes L6
      where L6.drinker = L2.drinker
      and L6.whisky = L5.whisky)))
```



Find CIDR drinkers w/ a unique whisky taste

Likes(drinker,whisky)

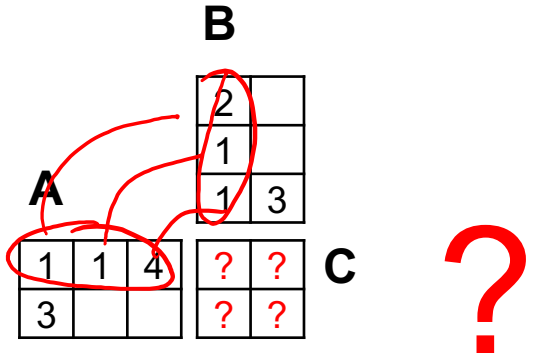
```
select distinct L1.drinker
from Likes L1
where not exists
  (select 1
  from Likes L2
  where L1.drinker <> L2.drinker
  and not exists
    (select 1
    from
    whe
    and
      (
      f
      w
      a
  and n
    (se
    from
    where L5. drinker = L1. drinker
    and not exists
      (select 1
      from Likes L6
      where L6.drinker = L2.drinker
      and L6.whisky = L5.whisky)))
```



- Abstract relations can "modularize" complicated queries and allow "collapse and expand", even for inside-out syntax
  - no need for pipes to make programs modular
- They may not have a well-defined extension on their own, but are well defined within the context of the query

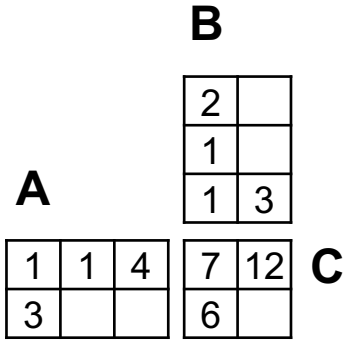
An example: Matrix Multiplication

Matrix Multiply $C = A \cdot B$



?

Matrix Multiply $C = A \cdot B$



Matrix Multiply $C = A \cdot B$

A

row	col	val
1	1	1
1	2	1
1	3	4
2	1	3

B

row	col	val
1	1	2
2	1	1
3	1	1
3	2	3

B

	1	2
1	2	
2	1	
3	1	3

A

	1	2	3
1	1	1	4
2	3		

7	12
6	

C

Matrix Multiply $C = A \cdot B$

input

A

row	col	val
1	1	1
1	2	1
1	3	4
2	1	3

B

row	col	val
1	1	2
2	1	1
3	1	1
3	2	3

B

	1	2
1	2	
2	1	
3	1	3

A

	1	2	3
1	1	1	4
2	3		

C

7	12
6	

Matrix Multiply $C = A \cdot B$

670

SQL

```
select A.row, B.col, sum(A.val*B.val) val
from A, B
where A.col = B.row
group by A.row, B.col
```

A

	1	2	3
1	1	1	4
2	3		

B

	1	2
1	2	
2	1	
3	1	3

C

	7	12
	6	

input

A

row	col	val
1	1	1
1	2	1
1	3	4
2	1	3

B

row	col	val
1	1	2
2	1	1
3	1	1
3	2	3

equijoin

a.row	a.col	a.val	b.row	b.col	b.val	a.val*b.val
1	1	1	1	1	2	2
1	2	1	2	1	1	1
1	3	4	3	1	1	4
1	3	4	3	2	3	12
2	1	3	1	1	2	6

group by (a.row,b.col)

	a.row	a.col	a.val	b.row	b.col	b.val	a.val*b.val
(1,1) →	1	1	1	1	1	2	2
	1	2	1	2	1	1	1
	1	3	1	3	1	1	4
(1,2) →	1	3	1	3	2	1	12
(2,1) →	2	1	3	1	1	2	6

aggregation

a.row				b.col		sum(a.val*b.val)
1				1		7
1				2		12
2				1		6

SQL example 670 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Matrix Multiply $C = A \cdot B$

670

SQL

```
select A.row, B.col, sum(A.val*B.val) val
from A, B
where A.col = B.row
group by A.row, B.col
```

```
select A.row, B.col, sum(M.out) val
from A, B, M
where A.col = B.row
and A.val = M.v1
and B.val = M.v2
group by A.row, B.col
```

A

	1	2	3
1	1	1	4
2	3		

B

	1	2
1	2	
2	1	
3	1	3

C

	7	12
	6	

input

A

row	col	val
1	1	1
1	2	1
1	3	4
2	1	3

B

row	col	val
1	1	2
2	1	1
3	1	1
3	2	3

M(ultiplication)

$m.v_1$	$m.v_2$	m.out
1	1	1
1	2	2
3	2	6
4	1	4
4	3	12

equijoin

a.row	a.col	a.val	b.row	b.col	b.val	$m.v_1$	$m.v_2$	m.out
1	1	1	1	1	2	1	2	2
1	2	1	2	1	1	1	1	1
1	3	4	3	1	1	4	1	4
1	3	4	3	2	3	4	3	12
2	1	3	1	1	2	3	2	6

group by (a.row,b.col)

	a.row	a.col	a.val	b.row	b.col	b.val	$m.v_1$	$m.v_2$	m.out
(1,1) →	1	1	1	1	1	2	1	2	2
	1	2	1	2	1	1	1	1	1
	1	3	1	3	1	1	4	1	4
(1,2) →	1	3	1	3	2	1	4	3	12
(2,1) →	2	1	3	1	1	2	3	2	6

aggregation

a.row				b.col				sum(m.out)
1				1				7
1				2				12
2				1				6

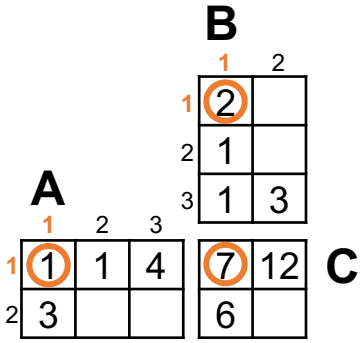
SQL example 670 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Matrix Multiply $C = A \cdot B$

SQL

```
select A.row, B.col, sum(A.val*B.val) val
from A, B
where A.col = B.row
group by A.row, B.col
```

```
select A.row, B.col, sum(M.out) val
from A, B, M
where A.col = B.row
and A.val = M.v1
and B.val = M.v2
group by A.row, B.col
```



input

A

row	col	val
1	1	1
1	2	1
1	3	4
2	1	3

B

row	col	val
1	1	2
2	1	1
3	1	1
3	2	3

M(ultiplication)

$m.v_1$	$m.v_2$	m.out
1	1	1
1	2	2
3	2	6
4	1	4
4	3	12

equijoin

a.row	a.col	a.val	b.row	b.col	b.val	$m.v_1$	$m.v_2$	m.out
1	1	1	1	1	2	1	2	2
1	2	1	2	1	1	1	1	1
1	3	4	3	1	1	4	1	4
1	3	4	3	2	3	4	3	12
2	1	3	1	1	2	3	2	6

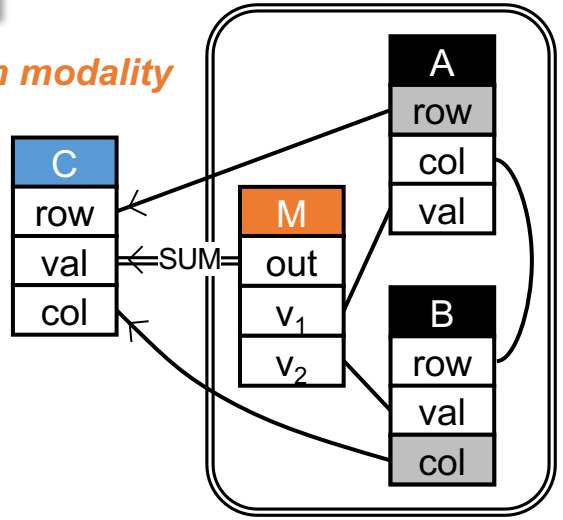
group by (a.row,b.col)

	a.row	a.col	a.val	b.row	b.col	b.val	$m.v_1$	$m.v_2$	m.out
(1,1) →	1	1	1	1	1	2	1	2	2
	1	2	1	2	1	1	1	1	1
	1	3	1	3	1	1	4	1	4
(1,2) →	1	3	1	3	2	1	4	3	12
(2,1) →	2	1	3	1	1	2	3	2	6

aggregation

a.row				b.col				sum(m.out)
1				1				7
1				2				12
2				1				6

Diagram modality



Abstract Relational Calculus (ARC)

Comprehension Syntax

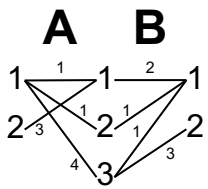
{C(row,col,val) | γa∈A, b∈B, m∈M,
group by({a.row, b.col})
[C.row=a.row ∧ C.col=b.col ∧ C.val=sum(m.out)
$m.v_1$=a.val ∧ $m.v_2$=b.val] }

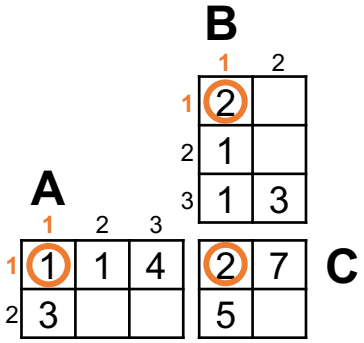
Shortest two-hop path

SQL

```
select A.row, B.col, min(A.val+B.val) val
from A, B
where A.col = B.row
group by A.row, B.col
```

```
select A.row, B.col, min(S.out) val
from A, B, S
where A.col = B.row
and A.val = S.v1
and B.val = S.v2
group by A.row, B.col
```





input

A

row	col	val
1	1	1
1	2	1
1	3	4
2	1	3

B

row	col	val
1	1	2
2	1	1
3	1	1
3	2	3

S(um)

$s.v_1$	$s.v_2$	s.out
1	1	2
1	2	3
3	2	5
4	1	5
4	3	7

equijoin

a.row	a.col	a.val	b.row	b.col	b.val	$s.v_1$	$s.v_2$	s.out
1	1	1	1	1	2	1	2	3
1	2	1	2	1	1	1	1	2
1	3	4	3	1	1	4	1	5
1	3	4	3	2	3	4	3	7
2	1	3	1	1	2	3	2	5

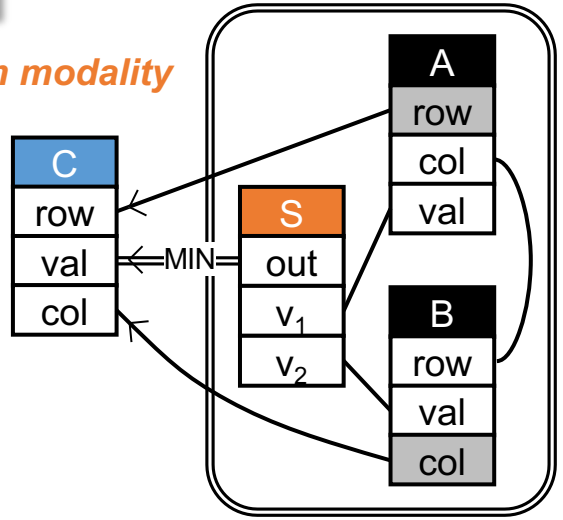
group by (a.row,b.col)

	a.row	a.col	a.val	b.row	b.col	b.val	$s.v_1$	$s.v_2$	s.out
(1,1) →	1	1	1	1	1	2	1	2	3
	1	2	1	2	1	1	1	1	2
	1	3	1	3	1	1	4	1	5
(1,2) →	1	3	1	3	2	1	4	3	7
(2,1) →	2	1	3	1	1	2	3	2	5

aggregation

a.row	b.col	min(s.out)
1	1	2
1	2	7
2	1	5

Diagram modality



Abstract Relational Calculus (ARC)

Comprehension Syntax

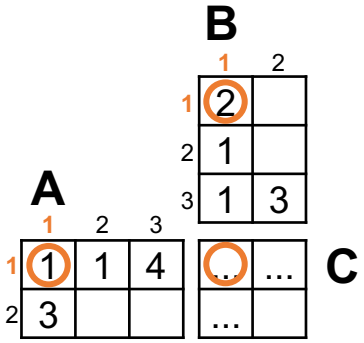
{C(row,col,val) | γa∈A, b∈B, s∈S,
group by({a.row, b.col})
[C.row=a.row ∧ C.col=b.col ∧ C.val=min(s.out)
$s.v_1$=a.val ∧ $s.v_2$=b.val] }

In Semiring Framework

SQL

```
select A.row, B.col, ⊕(A.val⊗B.val) val
from A, B
where A.col = B.row
group by A.row, B.col
```

```
select A.row, B.col, ⊕(P.out) val
from A, B, ⊗ as P
where A.col = B.row
and A.val = P.v1
and B.val = P.v2
group by A.row, B.col
```



input

A

row	col	val
1	1	1
1	2	1
1	3	4
2	1	3

B

row	col	val
1	1	2
2	1	1
3	1	1
3	2	3

P (⊗)

$p.v_1$	$p.v_2$	p.out
1	1	...
1	2	...
3	2	...
4	1	...
4	3	...

equijoin

a.row	a.col	a.val	b.row	b.col	b.val	$p.v_1$	$p.v_2$	p.out
1	1	1	1	1	2	1	2	...
1	2	1	2	1	1	1	1	...
1	3	4	3	1	1	4	1	...
1	3	4	3	2	3	4	3	...
2	1	3	1	1	2	3	2	...

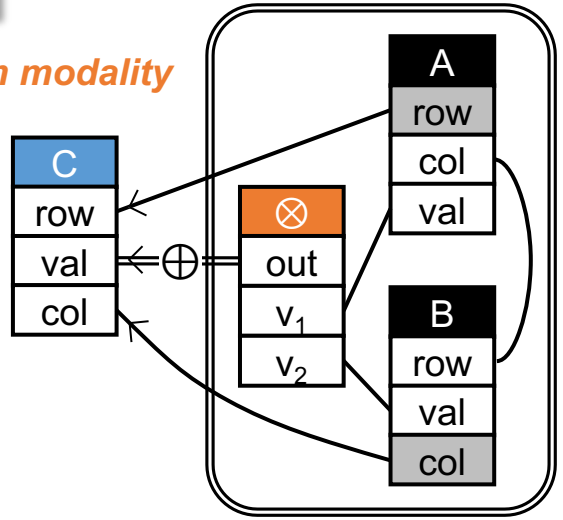
group by (a.row,b.col)

	a.row	a.col	a.val	b.row	b.col	b.val	$p.v_1$	$p.v_2$	p.out
(1,1) →	1	1	1	1	1	2	1	2	...
	1	2	1	2	1	1	1	1	...
	1	3	1	3	1	1	4	1	...
(1,2) →	1	3	1	3	2	1	4	3	...
(2,1) →	2	1	3	1	1	2	3	2	...

aggregation

a.row	b.col	min(s.out)
1	1	...
1	2	...
2	1	...

Diagram modality



Abstract Relational Calculus (ARC)

Comprehension Syntax

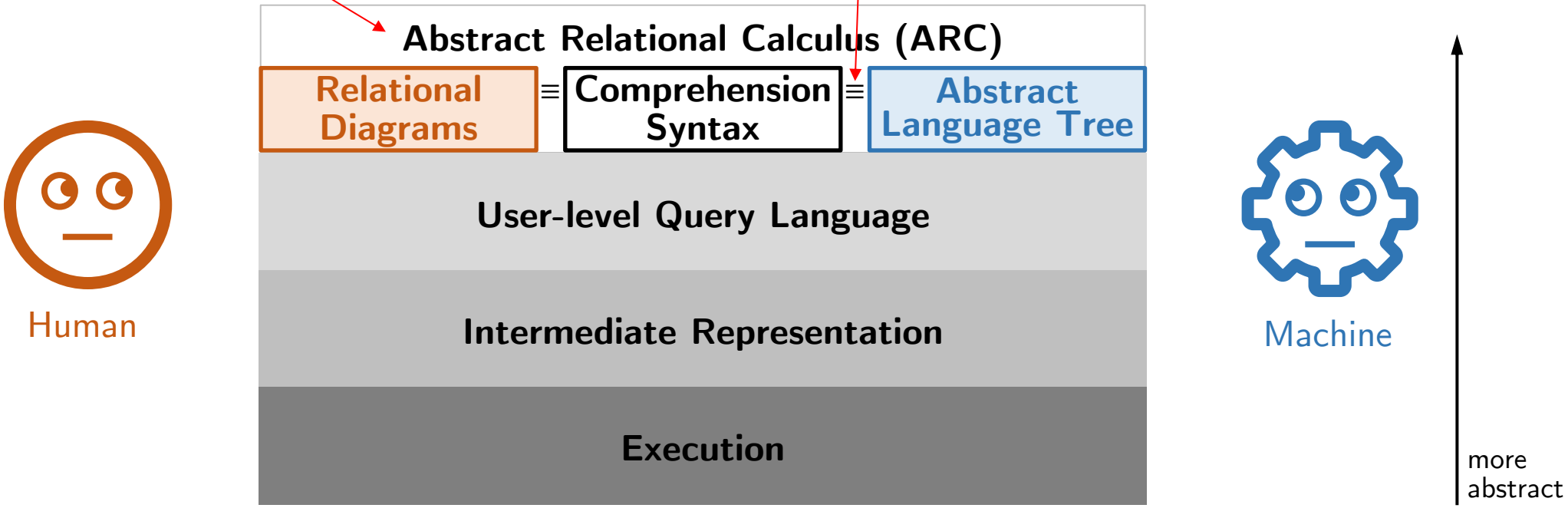
$\{C(row,col,val) \mid \gamma a \in A, b \in B, p \in \otimes,$
$\quad$ group by($\{a.row, b.col\}$)
$\quad [C.row=a.row \wedge C.col=b.col \wedge C.val=\oplus(p.out)$
$\quad p.v_1=a.val \wedge p.v_2=b.val] \}$

Take-aways

An Abstract Reference Language with refined vocabulary

Reference language. Focus is on the intent: Separates "relational intent" from representation.

modality: alternative representation of the intent tailored to different audiences



convention: orthogonal environment-level semantic parameters under which the relational intent is interpreted (e.g., set vs. bag semantics, or treatment of null values)

Choose an (optional) example:

Select example...

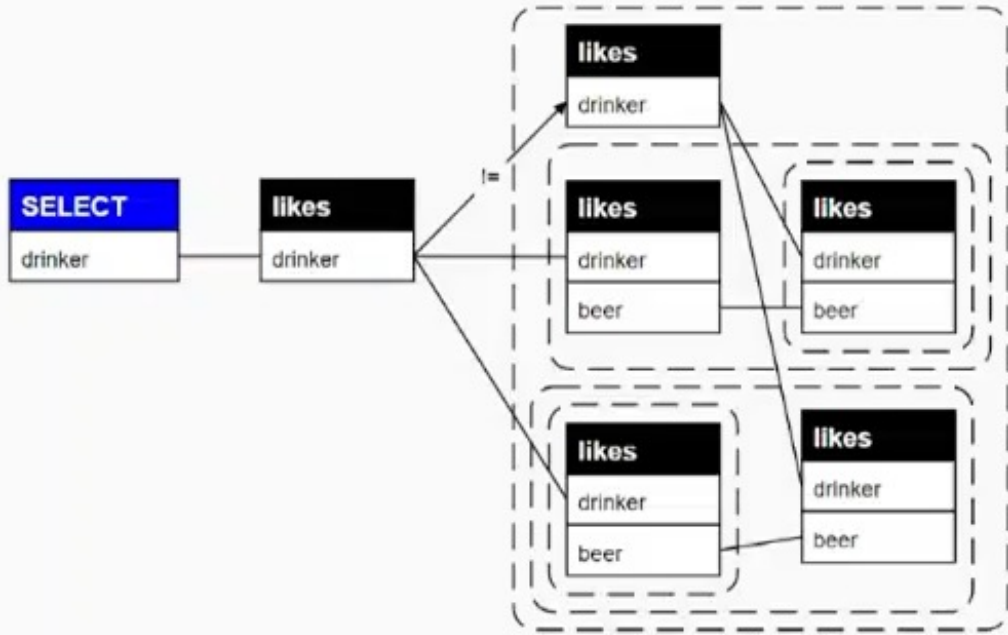
Enter a schema:

```
Likes(drinker, beer)
```

Enter a SQL query:

```
select distinct L1.drinker
from Likes as L1
where not exists
 (select 1
 from Likes as L2
 where L1.drinker <>
L2.drinker
 and not exists
   (select 1
   from Likes as L3
   where L3.drinker =
L2.drinker
   and not exists
     (select 1
```

Submit



Take-aways

- Abstract Relational Calculus: strict generalization of Tuple RC
  - intended as reference language, separating relational intent / modality (representation for audience) / convention (e.g., sets or bags)
- Most relevant papers (all available from RelationalDiagrams.com)
  - VLDB'23/ICDE'24: 500+ slide tutorial on diagrammatic representations
  - SIGMOD'24/Sigmod record'25: showing calculus can express more 'relational patterns' than algebra / reproducible user study
  - CIDR'26: today's talk: extensions to a collection framework
  - SIGMOD'26: disjunctions in diagrams
- Next steps
  - theory: formal proofs / nested relational model / sorting
  - practice: transpilers from other languages / NL2SQL

Thanks ☺

Anonymous feedback:

Project page: https://relationaldiagrams.com/