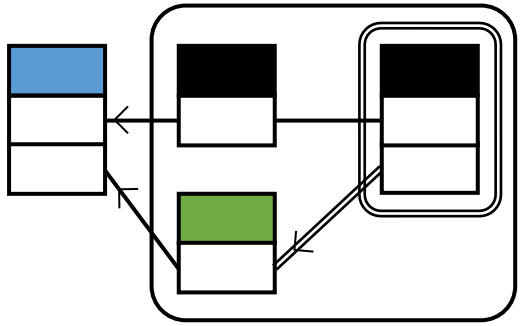
DATALAB
@Northeastern



A Tutorial on Relational Language Design

Wolfgang Gatterbauer

June 3, 2026 (SIGMOD'26)

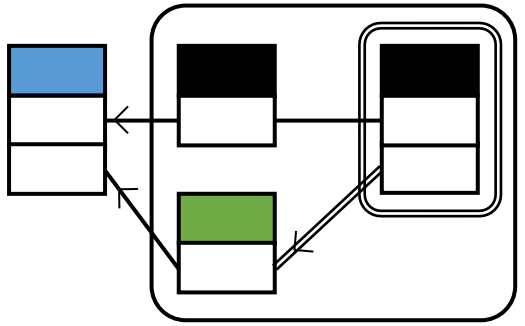
PLEASE ask questions! If you spot an error, please let me know, incl. the page number ☺

feedback QR

Corrected version from July 26, 2026

For the latest version, check the tutorial web page: https://northeastern-datalab.github.io/relational-language-tutorial/

DATALAB
@Northeastern



A Tutorial on Relational Language Design

Wolfgang Gatterbauer

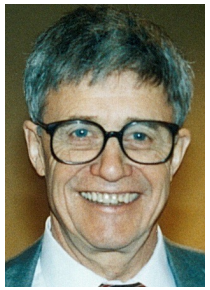
June 3, 2026 (SIGMOD'26)

PLEASE ask questions! If you spot an error, please let me know, incl. the page number ☺

feedback QR

Corrected version from July 26, 2026

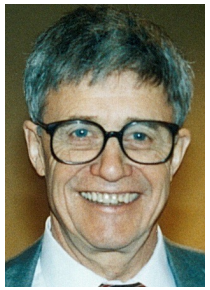
For the latest version, check the tutorial web page: https://northeastern-datalab.github.io/relational-language-tutorial/



?

KENNETH E. IVERSON
IBM Thomas J. Watson Research Center

1979 Turing Award Lecture



Notation as a Tool of Thought

1979 Turing Award Lecture

KENNETH E. IVERSON
IBM Thomas J. Watson Research Center

The importance of nomenclature, notation, and language as tools of thought has long been recognized. In chemistry and in botany, for example, the establishment of systems of nomenclature by Lavoisier and Linnaeus did much to stimulate and to channel later investigation. Concerning language, George Boole in his *Laws of Thought* [1, p. 24] asserted "That language is an instrument of human reason, and not merely a medium for the expression of thought, is a truth generally admitted."

Mathematical notation provides perhaps the best-known and best-developed example of language used consciously as a tool of thought. Recognition of the important role of notation in mathematics is clear from the quotations from mathematicians given in Cajori's *A History of Mathematical Notations* [2, pp. 332, 331]. They are well worth reading in full, but the following excerpts suggest the tone:

> By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race.
>
> A. N. Whitehead
>
> The quantity of meaning compressed into small space by algebraic signs, is another circumstance that facilitates the reasonings we are accustomed to carry on by their aid.
>
> Charles Babbage

[CACM 1980]

Left: Iverson, "*Notation as a Tool of Thought*", 1979 Turing award lecture, CACM 1980. https://doi.org/10.1145/358896.358899, nicer print: https://www.jsoftware.com/papers/tot.htm .
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Notation as a Tool of Thought

1979 Turing Award Lecture

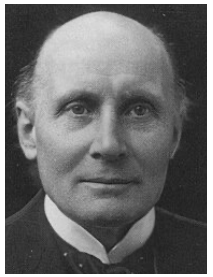
KENNETH E. IVERSON
IBM Thomas J. Watson Research Center

The importance of nomenclature, notation, and language as tools of thought has long been recognized. In chemistry and in botany, for example, the establishment of systems of nomenclature by Lavoisier and Linnaeus did much to stimulate and to channel later investigation. Concerning language, George Boole in his *Laws of Thought* [1, p. 24] asserted "That language is an instrument of human reason, and not merely a medium for the expression of thought, is a truth generally admitted."

Mathematical notation provides perhaps the best-known and best-developed example of language used consciously as a tool of thought. Recognition of the important role of notation in mathematics is clear from the quotations from mathematicians given in Cajori's *A History of Mathematical Notations* [2, pp. 332, 331]. They are well worth reading in full, but the following excerpts suggest the tone:

> By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race.
>
> A. N. Whitehead
>
> The quantity of meaning compressed into small space by algebraic signs, is another circumstance that facilitates the reasonings we are accustomed to carry on by their aid.
>
> Charles Babbage

[CACM 1980]



[1911]

721. *A. N. Whitehead*[3] writes on the power of symbols: "By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race. Before the introduction of the Arabic notation, multiplication was difficult, and the division even of integers called into play the highest mathematical faculties. Probably nothing in the modern world would have more astonished a Greek mathematician than to learn that, under the influence of compulsory education, the whole population of Western Europe, from the highest to the lowest, could perform the operation of division for the largest numbers. . . . . Mathematics is often considered a difficult and mysterious science, because of the numerous symbols which it employs. Of course, nothing is more incomprehensive than a symbolism which we do not understand. Also a symbolism, which we only partially understand and are unaccustomed to use, is difficult to follow. . . . . By the aid of symbolism, we can make transitions in reasoning almost mechanically by the eye, which otherwise would call into play the higher faculties of the brain. It is a profoundly erroneous truism, repeated by all copy-books and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them."

[Cajori 1929]

influence of Hindu-Arabic numerals

Left: Iverson, "*Notation as a Tool of Thought*", 1979 Turing award lecture, CACM 1980. https://doi.org/10.1145/358896.358899, nicer print: https://www.jsoftware.com/papers/tot.htm . Right: Cajori. "*A history of mathematical notations (books 1 and 2)*", 1928/1929. https://archive.org/details/historyofmathema031756mbp/ , (https://en.wikipedia.org/wiki/A_History_of_Mathematical_Notations). Whitehead. "*An introduction to mathematics*", 1911. https://archive.org/details/introductiontoma00whitiala.

Good Notational Design helps problem solving

- "*Have you ever tried multiplying Roman numerals? It's incredibly, ridiculously difficult. That's why, before the 14th century, everyone thought that multiplication was an incredibly difficult concept, and only for the mathematical elite.*
- *Then (Hindu-Arabic numerals) came along, with their nice place values, and we discovered that even seven-year-olds can handle multiplication just fine. There was nothing difficult about the concept of multiplication — the problem was that numbers, at the time, had a bad user interface.*" [Victor 2011]

Bret Victor, cited by FastCompany 2011: https://www.fastcompany.com/1664508/ex-apple-designer-creates-teaching-ui-that-kills-math-using-data-viz . But recall the earlier quote by Whitehead and likely many before: Whitehead. "*An introduction to mathematics*", 1911. https://archive.org/details/introductiontoma00whitiala . Fibonacci's Liber abaci (Leonardo of Pisa, 1202) is commonly cited as a major route by which Hindu-Arabic numerals and place-value arithmetic entered European mathematical culture. https://en.wikipedia.org/wiki/Liber_Abaci

Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

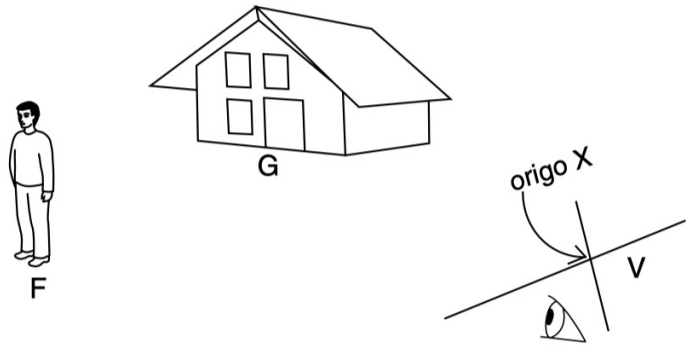
Linguistics: the scientific study of language

- Language as a Tool of Thought:
  - The study of how language influences thought and vice versa. "*Language can also be used for thought by framing and modifying thinking with a precision that was not possible without language*." [Wiki]
  - Linguistic determinism (Sapir-Whorf hypothesis): "*language determines thought and linguistic categories limit and restrict cognitive categories*" (controversial) [Wiki]
  - Linguistic relativity: "*language influences worldview or cognition*" [Wiki]

Wikipedia: https://en.wikipedia.org/wiki/Linguistic_relativity, https://en.wikipedia.org/wiki/Language_and_thought
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Linguistic relativity

RELATIVE
"He's to the left of the house."



ABSOLUTE
"He's north of the house."

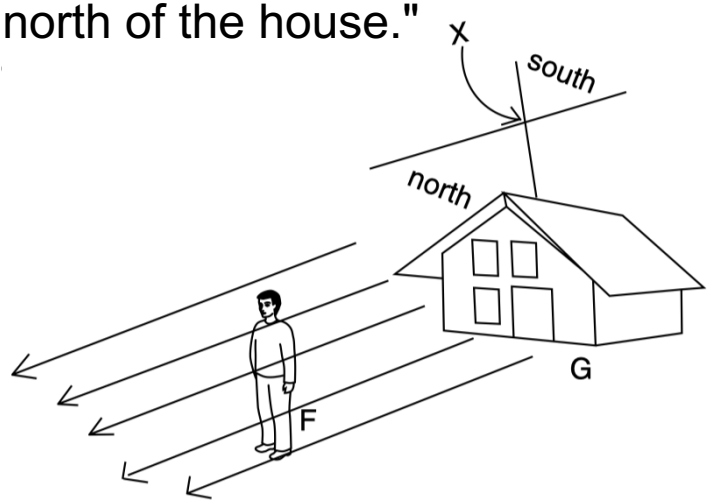
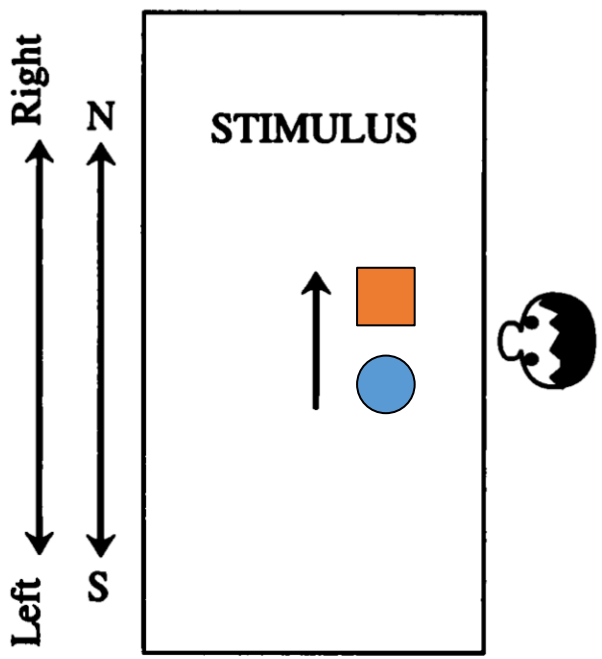


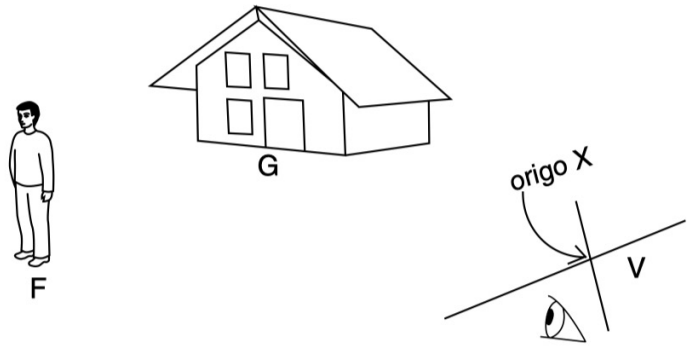
Table 1



Left: Excerpt of Figure 2.2 (3 frames of reference) from: Levinson. "*Space in Language and Cognition: Explorations in Cognitive Diversity*", Cambridge University Press 2003. https://doi.org/10.1017/CBO9780511613609
Right: Annotated Excerpt of Figure 4.2 from: Levinson. "*Frames of Reference and Molyneux's Question: Crosslinguistic Evidence*", MIT press, 1996. https://doi.org/10.7551/mitpress/4107.003.0006
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Linguistic relativity

RELATIVE
"He's to the left of the house."



ABSOLUTE
"He's north of the house."

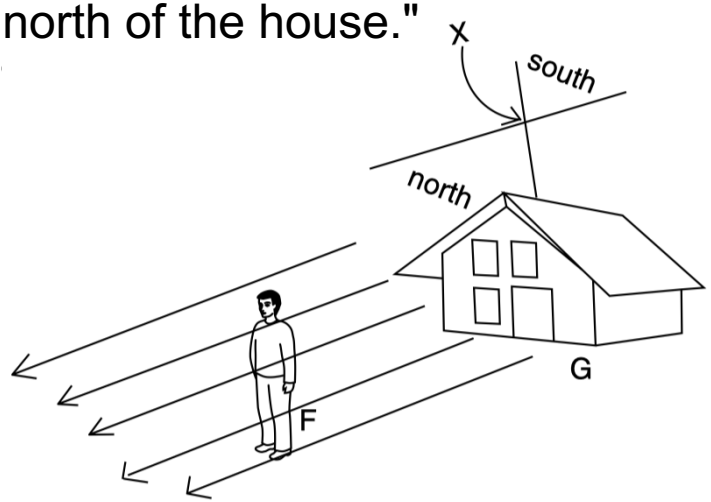
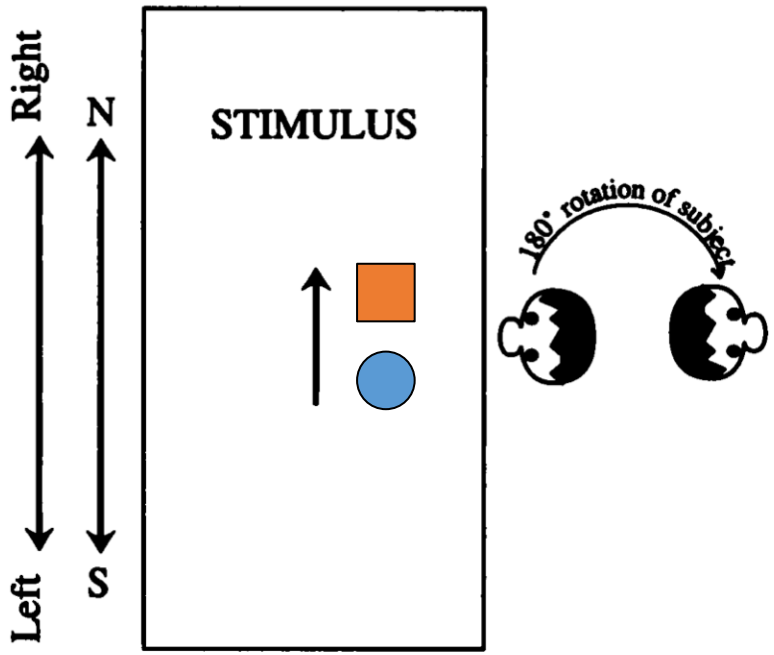


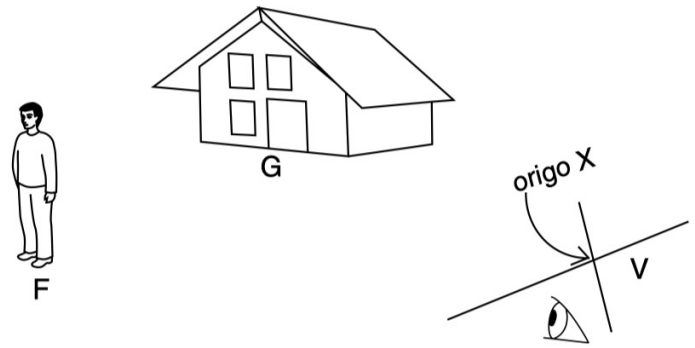
Table 1



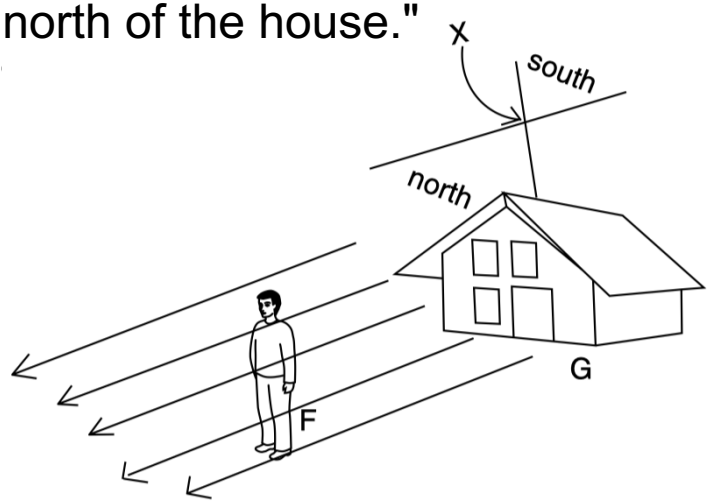
Left: Excerpt of Figure 2.2 (3 frames of reference) from: Levinson. "*Space in Language and Cognition: Explorations in Cognitive Diversity*", Cambridge University Press 2003. https://doi.org/10.1017/CBO9780511613609
Right: Annotated Excerpt of Figure 4.2 from: Levinson. "*Frames of Reference and Molyneux's Question: Crosslinguistic Evidence*", MIT press, 1996. https://doi.org/10.7551/mitpress/4107.003.0006
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

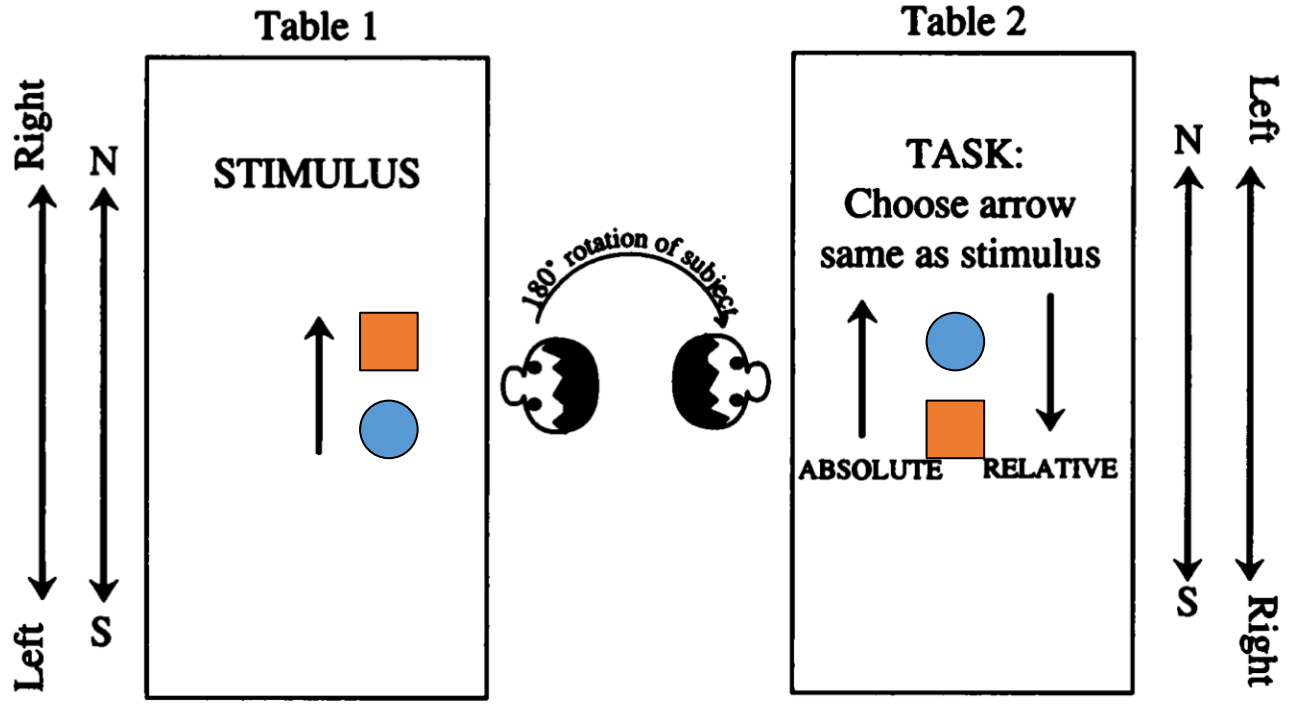
Linguistic relativity

RELATIVE
"He's to the left of the house."



ABSOLUTE
"He's north of the house."





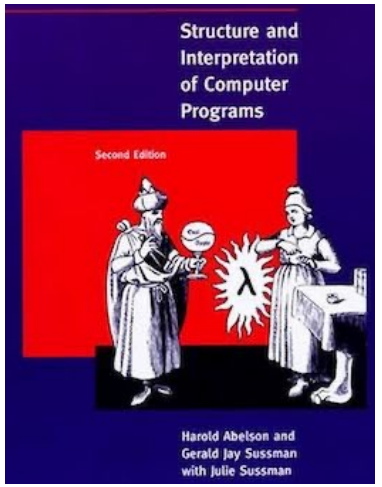
Comparing these two languages by "expressiveness" misses the point. Both can express the same situations.

They differ in what a language makes visible, i.e. what distinctions they make easier to encode or decode. Here, by their frame of reference

Left: Excerpt of Figure 2.2 (3 frames of reference) from: Levinson. "*Space in Language and Cognition: Explorations in Cognitive Diversity*", Cambridge University Press 2003. https://doi.org/10.1017/CBO9780511613609
Right: Annotated Excerpt of Figure 4.2 from: Levinson. "*Frames of Reference and Molyneux's Question: Crosslinguistic Evidence*", MIT press, 1996. https://doi.org/10.7551/mitpress/4107.003.0006
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

(Programming) Language as a Tool of Thought

- Language as a Tool of Thought
- Programming Languages as Tools of Thought
  - "*...the computer language you use influences how you understand and solve problems.*"[Metzger'81]
  - "*We control complexity by building abstractions that hide details* ..." [Abelson, Sussman'96]



[Metzger'81] Metzger, "*APL thinking finding array-oriented solutions*," SIGAPL quote quad 1981. https://doi.org/10.1145/390007.805361 .
[Abelson, Sussman'96]. "*Structure and interpretation of computer programs*," MIT press 1996. https://dl.acm.org/doi/10.5555/547755, https://web.mit.edu/6.001/6.037/sicp.pdf
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Different Notations invite different mental models

Python (element-by-element thinking)

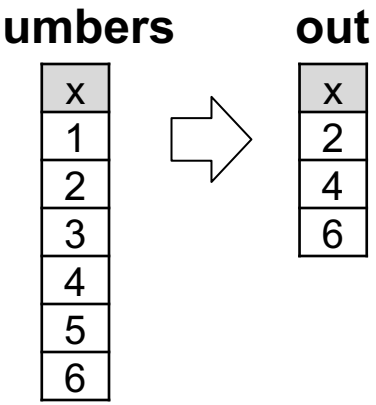
```
out = []
for x in numbers:
    if x % 2 == 0:
        out.append(x)
```

NumPy-style Python (whole collection thinking)

```
out = numbers[numbers % 2 == 0]
```

SQL

```
select x
from numbers
where x % 2 = 0
```



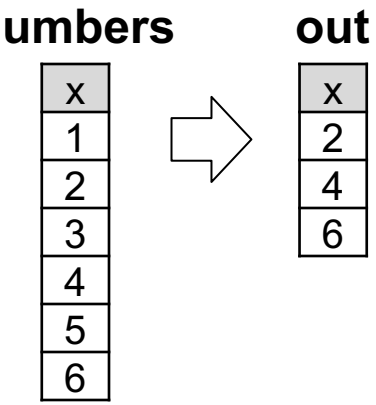
Different Notations invite different mental models

Python (element-by-element thinking)

```
out = []
for x in numbers:
    if x % 2 == 0:
        out.append(x)
```

Language Pattern

Walk through the numbers one by one, apply a filter (whether the number is even), and keep it if it passes the test.



NumPy-style Python (whole collection thinking)

```
out = numbers[numbers % 2 == 0]
```

Take a property (evenness) and apply it to the whole collection at once

SQL

```
select x
from numbers
where x % 2 = 0
```

Return the values for those rows where a condition (x % 2 = 0) holds.

A **procedural language** makes the procedure visible (a loop: visit, test, append).
A **declarative language** makes the predicate visible (keep rows where x is even)

Same computation, different thought.

We will revisit this argument later ☺

Relational Language Design (for users, not logicians)

- Language as a Tool of Thought
- Programming Languages as Tools of Thought
- Relational Languages as Tools of Thought

The bigger question: *How do relational languages help humans and machines write, read, understand, and modify queries?*

Our more modest goal today towards that bigger question: Starting to develop a vocabulary for questions like: *How do different relational languages realize the same relational intent while making different structure explicit?*

Outline

- What is a relational query? A vocabulary
- Various aspects of relational languages
  - Join queries
  - Lateral Joins
  - Declarativity of Languages
  - Grouping
  - Linear Composition
  - Relational Division
- Outlook

PLEASE ask questions and let me know if you spot any error anywhere ☺

An anonymous feedback form is linked from the tutorial web page:

What is a query?

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26

Out

eid	sm
a	30

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

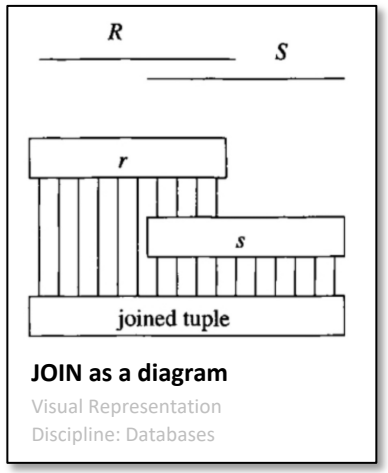
This is SQL behavior (for various reasons).

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

We use Relational Diagrams as "notional machines"

- "*A notional machine is the idealized model of the computer implied by the constructs of the programming language*" [du Boulay+'81]
- "*A notional machine is a pedagogic device to assist the understanding of some aspect of programs or programming.*" [Fincher+'20]
- *an example* [Miedema+'25]



Relational Diagram

SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

[du Boulay+'81] du Boulay, O'Shea, Monk: *The black box inside the glass box: presenting computing concepts to novices*, International Journal of Man-Machine Studies, 1981. https://doi.org/10.1016/S0020-7373(81)80056-9, [Fincher+'20] Fincher et al. *Notional Machines in Computing Education: The Education of Attention*, ITiCSE-WGR, 2020. https://doi.org/10.1145/3437800.3439202 , [Miedema+'25] *Explanations of Data Systems Concepts in CS Education: An Updated View on Notional Machines*, ITiCSE-WGR, 2025. https://doi.org/10.1145/3760545.3783971
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

We use Relational Diagrams as "notional machines"

- "*A notional machine is the idealized model of the computer implied by the constructs of the programming language*" [du Boulay+'81]
- "*A notional machine is a pedagogic device to assist the understanding of some aspect of programs or programming.*" [Fincher+'20]

Relational Diagram

SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

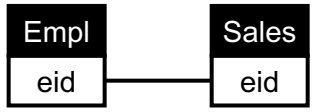
[du Boulay+'81] du Boulay, O'Shea, Monk: *The black box inside the glass box: presenting computing concepts to novices*, International Journal of Man-Machine Studies, 1981. https://doi.org/10.1016/S0020-7373(81)80056-9,
[Fincher+'20] Fincher et al. *Notional Machines in Computing Education: The Education of Attention*, ITiCSE-WGR, 2020. https://doi.org/10.1145/3437800.3439202
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

We use Relational Diagrams as "notional machines"

- "*A notional machine is the idealized model of the computer implied by the constructs of the programming language*" [du Boulay+'81]
- "*A notional machine is a pedagogic device to assist the understanding of some aspect of programs or programming.*" [Fincher+'20]

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

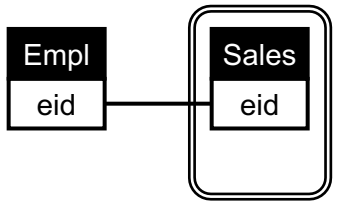
[du Boulay+'81] du Boulay, O'Shea, Monk: *The black box inside the glass box: presenting computing concepts to novices*, International Journal of Man-Machine Studies, 1981. https://doi.org/10.1016/S0020-7373(81)80056-9,
[Fincher+'20] Fincher et al. *Notional Machines in Computing Education: The Education of Attention*, ITiCSE-WGR, 2020. https://doi.org/10.1145/3437800.3439202
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

We use Relational Diagrams as "notional machines"

- "*A notional machine is the idealized model of the computer implied by the constructs of the programming language*" [du Boulay+'81]
- "*A notional machine is a pedagogic device to assist the understanding of some aspect of programs or programming.*" [Fincher+'20]

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

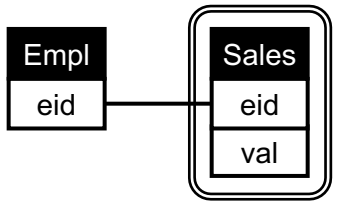
[du Boulay+'81] du Boulay, O'Shea, Monk: *The black box inside the glass box: presenting computing concepts to novices*, International Journal of Man-Machine Studies, 1981. https://doi.org/10.1016/S0020-7373(81)80056-9,
[Fincher+'20] Fincher et al. *Notional Machines in Computing Education: The Education of Attention*, ITiCSE-WGR, 2020. https://doi.org/10.1145/3437800.3439202
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

We use Relational Diagrams as "notional machines"

- "*A notional machine is the idealized model of the computer implied by the constructs of the programming language*" [du Boulay+'81]
- "*A notional machine is a pedagogic device to assist the understanding of some aspect of programs or programming.*" [Fincher+'20]

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

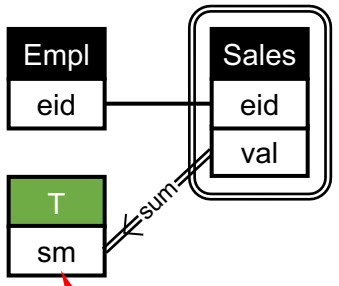
[du Boulay+'81] du Boulay, O'Shea, Monk: *The black box inside the glass box: presenting computing concepts to novices*, International Journal of Man-Machine Studies, 1981. https://doi.org/10.1016/S0020-7373(81)80056-9,
[Fincher+'20] Fincher et al. *Notional Machines in Computing Education: The Education of Attention*, ITiCSE-WGR, 2020. https://doi.org/10.1145/3437800.3439202
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

We use Relational Diagrams as "notional machines"

- "*A notional machine is the idealized model of the computer implied by the constructs of the programming language*" [du Boulay+'81]
- "*A notional machine is a pedagogic device to assist the understanding of some aspect of programs or programming.*" [Fincher+'20]

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

The sm values are yielded/returned from the scope.
Thus we show them *outside* (in contrast to SQL).

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

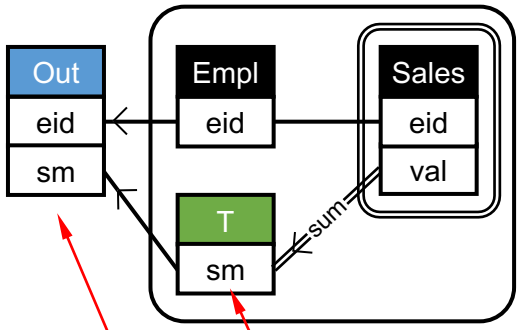
[du Boulay+'81] du Boulay, O'Shea, Monk: *The black box inside the glass box: presenting computing concepts to novices*, International Journal of Man-Machine Studies, 1981. https://doi.org/10.1016/S0020-7373(81)80056-9,
[Fincher+'20] Fincher et al. *Notional Machines in Computing Education: The Education of Attention*, ITiCSE-WGR, 2020. https://doi.org/10.1145/3437800.3439202
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

We use Relational Diagrams as "notional machines"

- "*A notional machine is the idealized model of the computer implied by the constructs of the programming language*" [du Boulay+'81]
- "*A notional machine is a pedagogic device to assist the understanding of some aspect of programs or programming.*" [Fincher+'20]

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

The sm values are yielded/returned from the scope.
Thus we show them *outside* (in contrast to SQL).

The query as a whole yields/returns tuples (eid, sm), and those are shown again *outside* the scope of the query

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

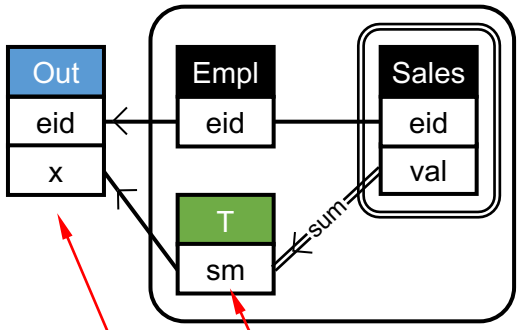
[du Boulay+'81] du Boulay, O'Shea, Monk: *The black box inside the glass box: presenting computing concepts to novices*, International Journal of Man-Machine Studies, 1981. https://doi.org/10.1016/S0020-7373(81)80056-9,
[Fincher+'20] Fincher et al. *Notional Machines in Computing Education: The Education of Attention*, ITiCSE-WGR, 2020. https://doi.org/10.1145/3437800.3439202
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

We use Relational Diagrams as "notional machines"

- "*A notional machine is the idealized model of the computer implied by the constructs of the programming language*" [du Boulay+'81]
- "*A notional machine is a pedagogic device to assist the understanding of some aspect of programs or programming.*" [Fincher+'20]

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)   as x
from Empl E
```

The sm values are yielded/returned from the scope.
Thus we show them *outside* (in contrast to SQL).

The query as a whole yields/returns tuples (eid, x), and those are shown again *outside* the scope of the query

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

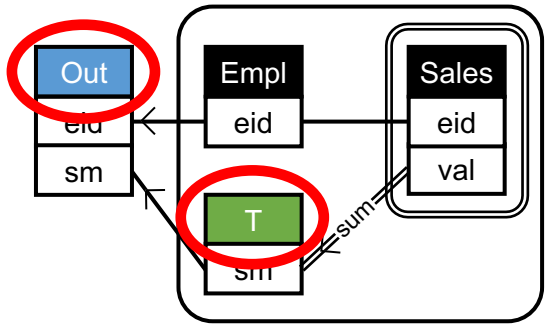
Out

eid	sm
a	30
b	40
c	null

[du Boulay+'81] du Boulay, O'Shea, Monk: *The black box inside the glass box: presenting computing concepts to novices*, International Journal of Man-Machine Studies, 1981. https://doi.org/10.1016/S0020-7373(81)80056-9,
[Fincher+'20] Fincher et al. *Notional Machines in Computing Education: The Education of Attention*, ITiCSE-WGR, 2020. https://doi.org/10.1145/3437800.3439202
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

?

Empl

eid	k
a	1
b	2
c	3

Sales

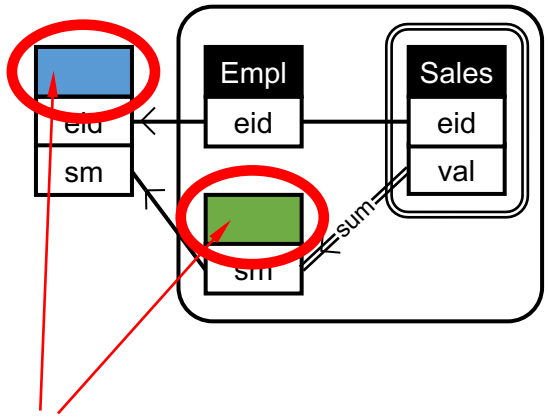
eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

What is a query?

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Intermediate relational structures without names (including the final output) can be represented as unnamed relations. We show them because the concepts of intermediate and output tables are helpful, even if some languages like SQL choose not to show them or give them names (contrast with Datalog which at least names the output table)

Empl

eid	k
a	1
b	2
c	3

Sales

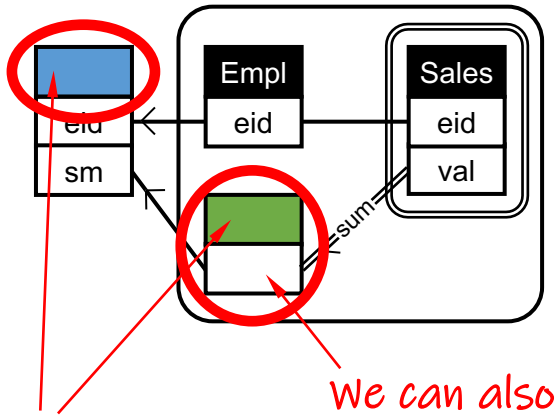
eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

What is a query?

Relational Diagram



SQL

```
select E.eid,
  (select sum(val)
  from Sales S
  where E.eid = S.eid) as sm
from Empl E
```

We can also have unnamed columns

Intermediate relational structures without names (including the final output) can be represented as unnamed relations. We show them because the concepts of intermediate and output tables are helpful, even if some languages like SQL choose not to show them or give them names (contrast with Datalog which at least names the output table)

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

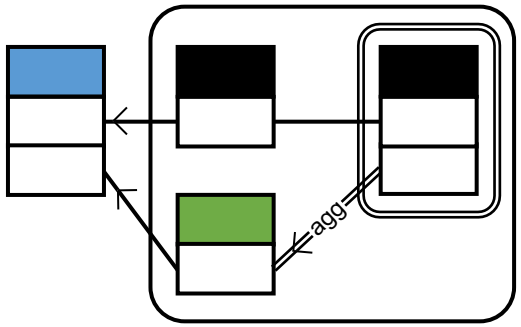
What is a query?

Relational pattern: the notation-independent abstract relational structure encoded by a query expression e: how the output Out is derived from table references (...).

Relational Pattern

Correlated subcollection summary: Given an outer collection and an inner collection, use each outer tuple to determine a matching subcollection of inner tuples. Summarize each subcollection with an aggregate, and output selected outer attributes together with the summary.

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

For a slightly different, more formal definition of relational patterns see: [Gatterbauer, Dunne'25] *Relational Diagrams and the Pattern Expressiveness of Relational Languages*, SIGMOD record 2025, https://sigmodrecord.org/?smd_process_download=1&download_id=14010

Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

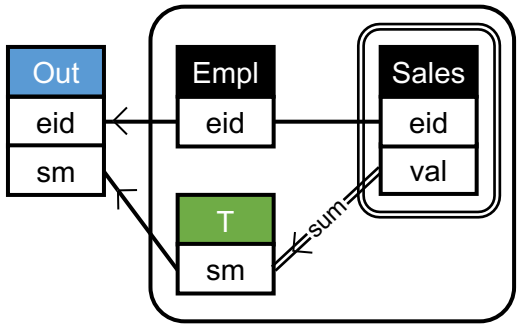
What is a query?

Relational pattern: the notation-independent abstract relational structure encoded by a query expression e: how the output Out is derived from table references (...).

Relational Pattern

Correlated subcollection summary: Given an outer collection and an inner collection, use each outer tuple to determine a matching subcollection of inner tuples. Summarize each subcollection with an aggregate, and output selected outer attributes together with the summary.

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Instantiation:

- Outer collection: `Empl`, bound as `E`.
- Inner collection: `Sales`, bound as `S`.
- Matching condition: `E.eid = S.eid`.
- Copied outer attribute: `E.eid`.
- Aggregated inner expression: `S.val`.
- Aggregate: `sum`, named `sm`.

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

For a slightly more formal definition of relational patterns see: [Gatterbauer, Dunne'25] *Relational Diagrams and the Pattern Expressiveness of Relational Languages*, SIGMOD record 2025, https://doi.org/10.1145/3733620.373363
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

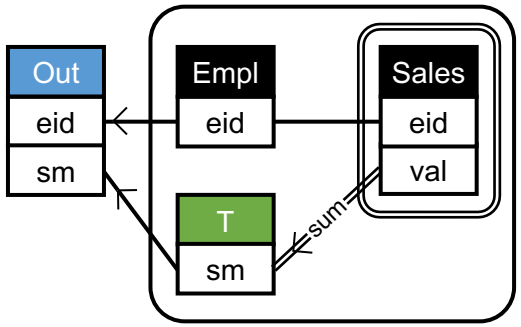
What is a query?

Relational pattern: the notation-independent abstract relational structure encoded by a query expression e: how the output Out is derived from table references (...).

Relational Pattern

Correlated subcollection summary: Given an outer collection and an inner collection, use each outer tuple to determine a matching subcollection of inner tuples. Summarize each subcollection with an aggregate, and output selected outer attributes together with the summary.

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Instantiation:

- Outer collection: `Empl`, bound as `E`.
- Inner collection: `Sales`, bound as `S`.
- Matching condition: `E.eid = S.eid`.
- Copied outer attribute: `E.eid`.
- Aggregated inner expression: `S.val`.
- Aggregate: `sum`, named `sm`.

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	null

Abstract Relational Calculus (ARC)

Abstract Language Tree

```
COLLECTION
├─ HEAD: Out(eid,sm)
└─ SCOPE ∃
   ├─ BINDING: e ∈ Empl
   ├─ BINDING: t ∈
   │  └─ COLLECTION
   │     ├─ HEAD: T(sm)
   │     └─ SCOPE γ
   │        ├─ BINDING: s ∈ Sales
   │        ├─ GROUP BY: ∅
   │        └─ AND ∧
   │           ├─ PREDICATE: e.A = s.A
   │           └─ PREDICATE: T.sm = sum(s.B)
   └─ AND ∧
      ├─ PREDICATE: Out.A = e.A
      └─ PREDICATE: Out.sm = t.sm
```

Comprehension Syntax

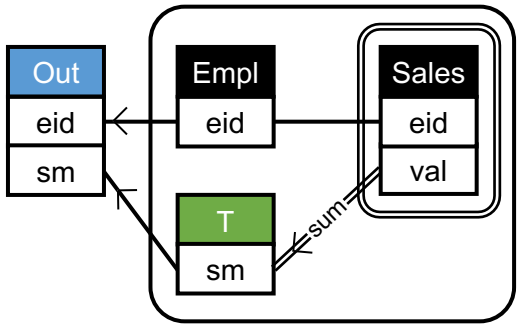
$$\{\mathrm{Out}(\mathrm{eid}, \mathrm{sm}) \mid \exists e \in \mathrm{Empl}, t \in \{\mathrm{T}(\mathrm{sm}) \mid \exists s \in \mathrm{Sales}, \gamma_{\emptyset}\ [s.\mathrm{eid}=e.\mathrm{eid} \wedge \mathrm{T.sm}=\mathrm{sum}(s.\mathrm{val})]\}\ [\mathrm{Out.eid}=e.\mathrm{eid} \wedge \mathrm{Out.sm}=t.\mathrm{sm}]\}$$

Abstract Relational Calculus is a Relational Reference Language, but is not covered in this tutorial. See recent CIDR 2026 video recording

For details on ARC, see: Gatterbauer, Sabale. *Database Research needs an Abstract Relational Query Language*, CIDR 2026. https://www.vldb.org/cidrdb/papers/2026/p27-gatterbauer.pdf
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

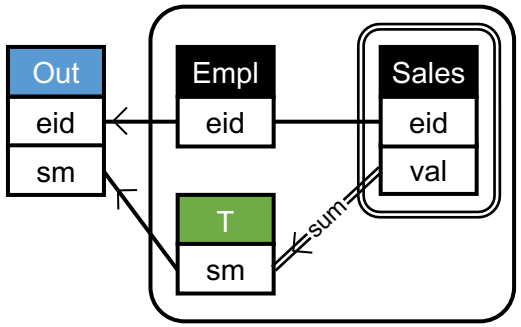
eid	sm
a	30
b	40
c	null

This is SQL behavior (for various reasons).
But we want 0 instead of null because the neutral element of summation is 0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational Diagram



SQL

```
select E.eid,
  (select coalesce(sum(val),0) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

coalesce(... , 0) works

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

User-Defined Aggregate (UDA)

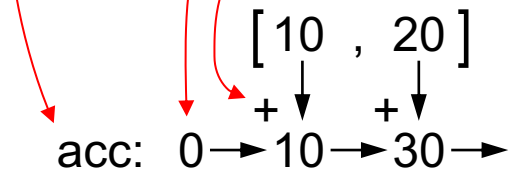
We can define a user-defined aggregate (UDA) in Postgres as follows:

```
create aggregate sum0(integer) (
  sfunc = int4pl,
  stype = integer,
  initcond = '0');
```

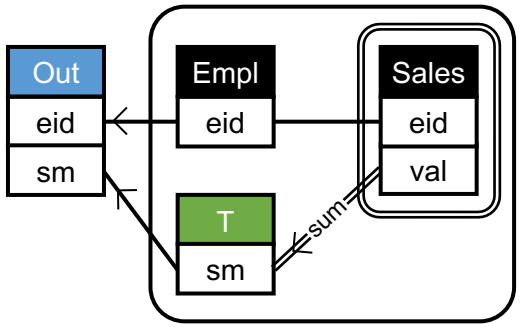
addition as step function (+)

accumulator type (Int)

initial accumulator value (0)



Relational Diagram



SQL

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql; SQLite has "total(x)" which returns 0.0 on an empty input https://sqlite.org/lang_aggfunc.html; In NumPy "np.sum([])" returns 0.0 https://numpy.org/doc/2.3/reference/generated/numpy.sum.html; Python similarly has "sum(iterable, start=0)" https://docs.python.org/3/library/functions.html#sum; Calcite has a similar "sum0" aggregate (thanks to Julian Hyde for letting me know): https://calcite.apache.org/javadocAggregate/org/apache/calcite/sql/fun/SqlStdOperatorTable.html#SUM0

Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

User-Defined Aggregate (UDA)

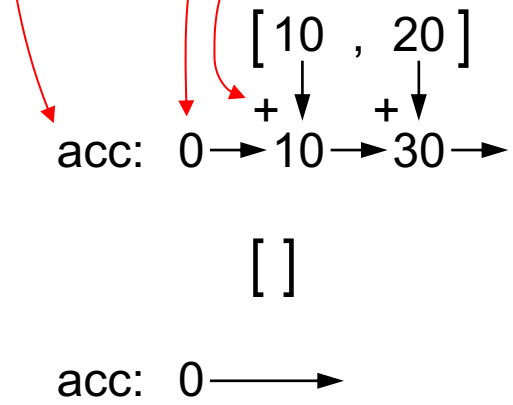
We can define a user-defined aggregate (UDA) in Postgres as follows:

```
create aggregate sum0(integer) (
  sfunc = int4pl,
  stype = integer,
  initcond = '0');
```

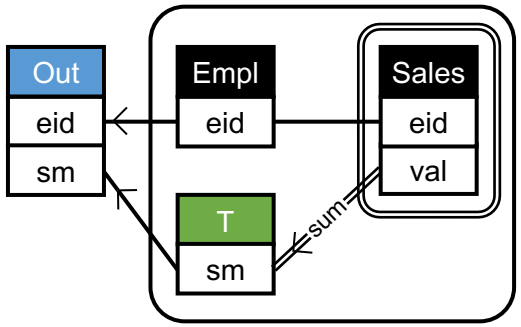
addition as step function (+)

accumulator type (Int)

initial accumulator value (0)



Relational Diagram



SQL

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

In functional PL terms, the aggregate `sum0` is just a fold with an explicit initial accumulator value:

Haskell

```
sum0 xs = foldl (+) 0 xs
sum0 [10, 20]
  foldl (+) 0 [10, 20]
  (0 + 10) + 20
  30
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql; SQLite has "total(x)" which returns 0.0 on an empty input https://sqlite.org/lang_aggfunc.html; In NumPy "np.sum([])" returns 0.0 https://numpy.org/doc/2.3/reference/generated/numpy.sum.html; Python similarly has "sum(iterable, start=0)" https://docs.python.org/3/library/functions.html#sum; Calcite has a similar "sum0" aggregate (thanks to Julian Hyde for letting me know): https://calcite.apache.org/javadocAggregate/org/apache/calcite/sql/fun/SqlStdOperatorTable.html#SUM0

Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

DETAILS

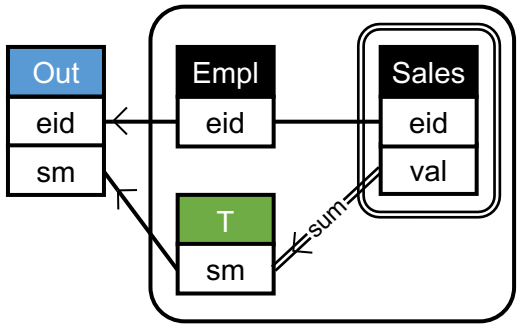
```
create schema overrides;

create aggregate overrides.sum(integer) (
  sfunc = int4pl,
  stype = integer,
  initcond = '0');

set search_path = overrides,
  pg_catalog, public;
```

shadows sum in the search path

Details: PostgreSQL lets schemas in search_path determine which unqualified function or aggregate name is used. pg_catalog is normally searched first implicitly, but you can explicitly put pg_catalog later so user-defined names override built-ins.

Also, PostgreSQL's built-in sum(integer) accumulates beyond integer; better to use bigint/numeric transition types to avoid integer overflow.

Relational Diagram



SQL

```
select E.eid,
  (select sum(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Now, this unqualified call resolves to overrides.sum(integer) instead of pg_catalog.sum(integer)

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

Details: the original sum aggregate is still available:

```
...
  (select pg_catalog.sum (val) as sm
...
```

What is a query?

Relational pattern: the notation-independent abstract relational structure encoded by a query expression e: how the output Out is derived from table references (...).

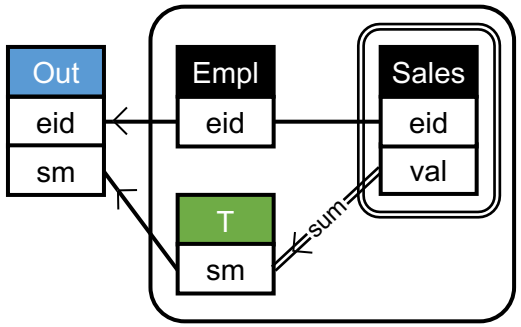
Relational Pattern

Correlated subcollection summary: Given an outer collection and an inner collection, use each outer tuple to determine a matching subcollection of inner tuples. Summarize each subcollection with an aggregate, and output selected outer attributes together with the summary.

Instantiation:

- Outer collection: `Empl`, bound as `E`.
- Inner collection: `Sales`, bound as `S`.
- Matching condition: `E.eid = S.eid`.
- Copied outer attribute: `E.eid`.
- Aggregated inner expression: `S.val`.
- Aggregate: `sum`, named `sm`.

Relational Diagram



SQL

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

Different syntax, same pattern!
Correspondence between "x-x" and the line in "Empl.eid=Sales.eid" is not explicit!

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

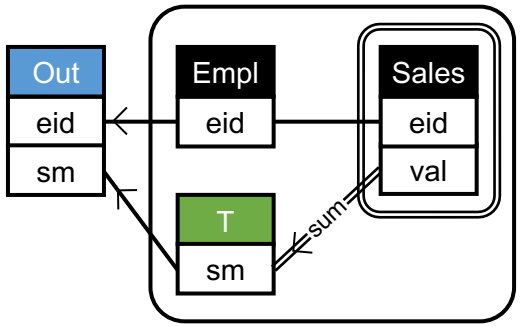
Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql; Soufflé code 750-1 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational Diagram



SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

Empl

Sales

SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

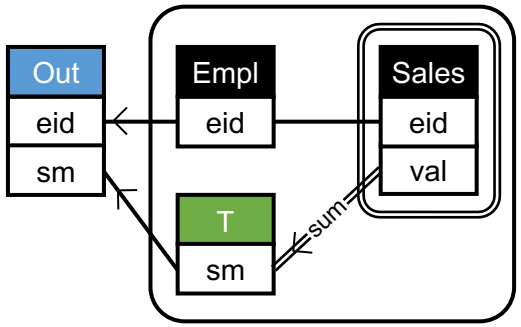
Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational Diagram

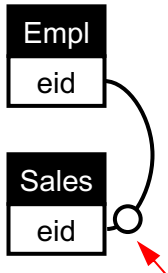


SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```



SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Inspired by crow's foot notation in ER diagrams for "optional" participation constraints

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

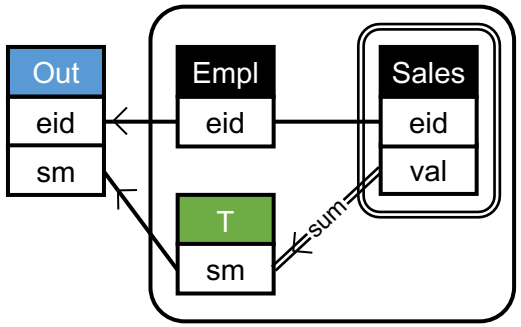
Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational Diagram

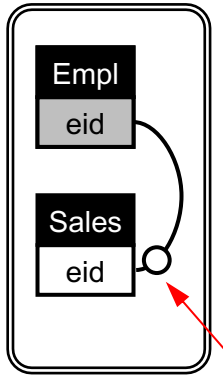


SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```



SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Inspired by crow's foot notation in ER diagrams for "optional" participation constraints

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

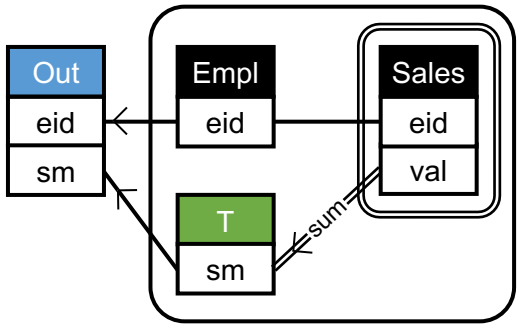
Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational Diagram

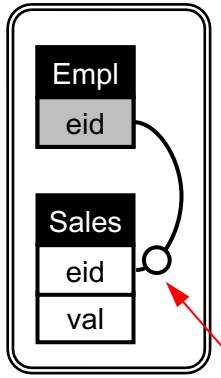


SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```



SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Inspired by crow's foot notation in ER diagrams for "optional" participation constraints

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

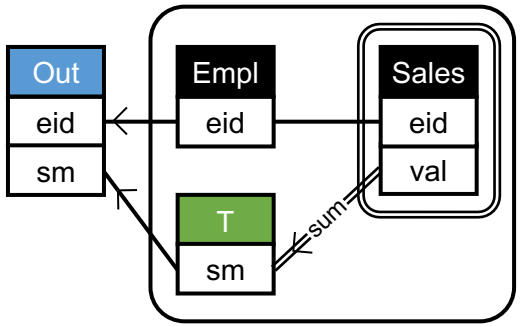
Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational Diagram

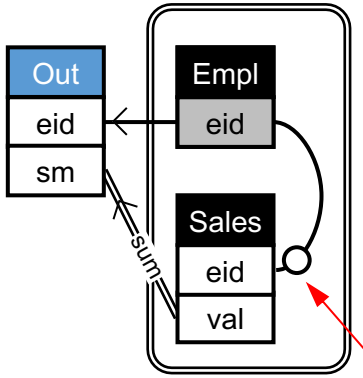


SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```



SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Inspired by crow's foot notation in ER diagrams for "optional" participation constraints

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

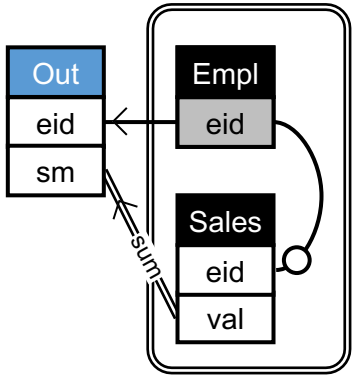
What is a query?

Relational pattern (structure): the notation-independent abstract relational structure represented by e: how output Out are derived from table-reference roles (...).

Relational Pattern

?

Relational Diagram



SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

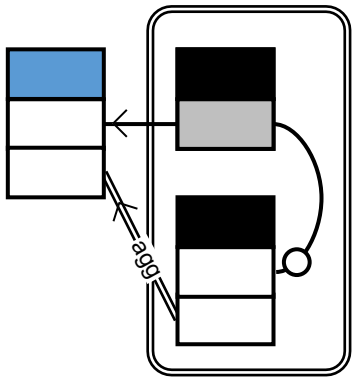
What is a query?

Relational pattern (structure): the notation-independent abstract relational structure represented by e: how output Out are derived from table-reference roles (...).

Relational Pattern

Outer-preserving grouped summary: Given an outer collection and an inner collection, pair each outer tuple with every matching inner tuple, while preserving outer tuples with no matches. Group the resulting pairs by selected outer attributes. Summarize each group with an aggregate over inner values, and return the grouping attributes together with the summary.

Relational Diagram



SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational pattern (structure): the notation-independent abstract relational structure represented by e: how output Out are derived from table-reference roles (...).

Query expression e: a well-formed syntactic object, written in the notation of a particular query language L.

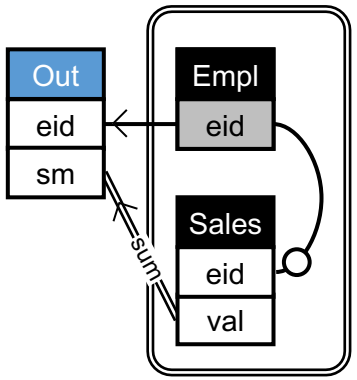
Relational Pattern

Outer-preserving grouped summary: Given an outer collection and an inner collection, pair each outer tuple with every matching inner tuple, while preserving outer tuples with no matches. Group the resulting pairs by selected outer attributes. Summarize each group with an aggregate over inner values, and return the grouping attributes together with the summary.

Instantiation:

- Outer collection: `Empl`, bound as `E`.
- Inner collection: `Sales`, bound as `S`.
- Matching condition: `E.eid = S.eid`.
- Grouping attribute: `E.eid`.
- Aggregated inner expression: `S.val`.
- Aggregate: `sum`, named `sm`.

Relational Diagram



SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Do these 3 query expressions "mean the same thing" (do they have the same denotation)? Thus, do they return the same output table for any input tables?

?

Query expression e: a well-formed syntactic object, written in the notation of a particular query language L.

SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Side topic: Do these two functions mean the same thing? (Do these two expressions define the same function?)

DETAILS

$$f(x) = x^3$$

$$g(x) = |x|^3$$

?

Side topic: Do these two functions mean the same thing? (Do these two expressions define the same function?)

DETAILS

$$f_{\mathcal{D}}(x) = x^3 \qquad (x \in \mathcal{D})$$

$$g_{\mathcal{D}}(x) = |x|^3 \qquad (x \in \mathcal{D})$$

?

Expressions do not define a fully specified function until a domain is supplied.

Equality of functions need to compare the domain and the mapping by an extensional view of functions: "A function is fully characterized by its domain ($\mathcal{D}f$ for function f) and its mapping."[Boute'10]

[Boute '05] "*Functional declarative language design and predicate calculus: a practical approach*", TOPLAS 2005. https://doi.org/10.1145/1086642.1086647. Boute treats a function as specified by two attributes: its domain and its mapping. He notes that some traditions include a codomain, but his and our function concept does not require one. In his words: A function "is fully specified by two attributes: a domain (the set on which it is defined) and its mapping, associating with every domain element a unique image." See Section 8.1 in [Boute'10] "*Pointfree expression and calculation: from quantification to temporal logic*", Form Methods Syst Des 2010. https://doi.org/10.1007/s10703-010-0100-2. Thanks to Molham Aref for introducing me to Boute's work!

Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Side topic: Do these two functions mean the same thing? (Do these two expressions define the same function?)

DETAILS

$$f_{\mathcal{D}}(x) = x^3 \qquad (x \in \mathcal{D})$$

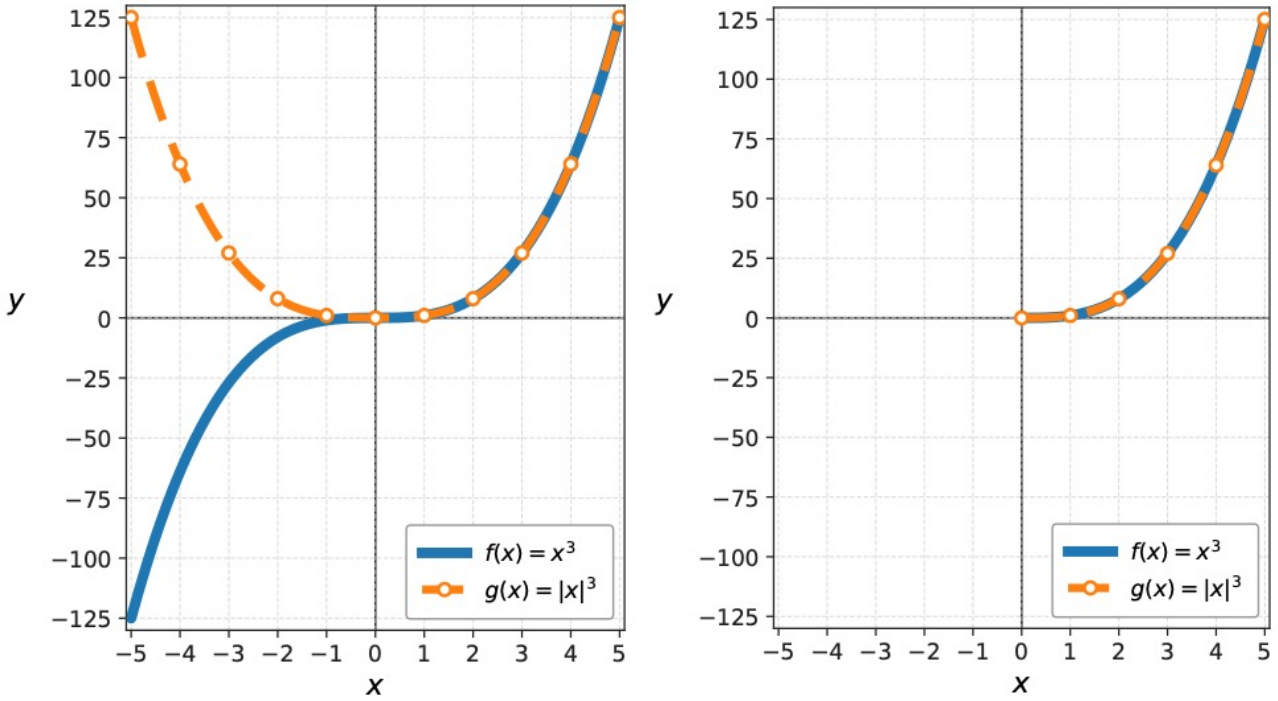
$$g_{\mathcal{D}}(x) = |x|^3 \qquad (x \in \mathcal{D})$$

Expressions do not define a fully specified function until a domain is supplied.

Equality of functions need to compare the domain and the mapping by an extensional view of functions: "A function is fully characterized by its domain ($\mathcal{D}f$ for function f) and its mapping."[Boute'10]

no ⟷ yes iff $\mathcal{D} \subseteq \mathbb{R}_{\geq 0}$

$f_{\mathbb{R}} \neq g_{\mathbb{R}}$ $\qquad$ $f_{\mathbb{R}_{\geq 0}} = g_{\mathbb{R}_{\geq 0}}$



[Boute '05] "*Functional declarative language design and predicate calculus: a practical approach*", TOPLAS 2005. https://doi.org/10.1145/1086642.1086647. Boute treats a function as specified by two attributes: its domain and its mapping. He notes that some traditions include a codomain, but his and our function concept does not require one. In his words: A function "is fully specified by two attributes: a domain (the set on which it is defined) and its mapping, associating with every domain element a unique image." See Section 8.1 in [Boute'10] "*Pointfree expression and calculation: from quantification to temporal logic*", Form Methods Syst Des 2010. https://doi.org/10.1007/s10703-010-0100-2. Thanks to Molham Aref for introducing me to Boute's work!

Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational pattern (structure): the notation-independent abstract relational structure represented by e: how output Out are derived from table-reference roles (...).

Query expression e: a well-formed syntactic object, written in the notation of a particular query language L.

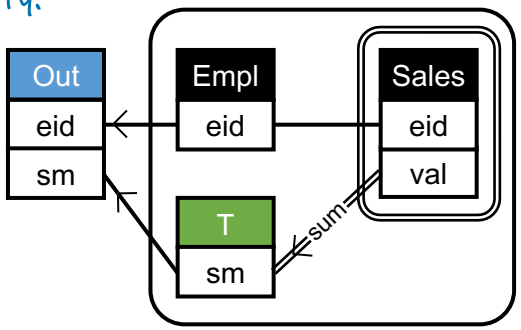
SQL1

Out

eid	sm
a	30
b	40
c	0

?

In Diagram modality:



SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Soufflé (Datalog extension)

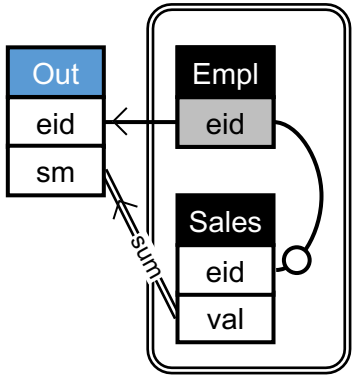
```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

Soufflé

Out

eid	sm
a	30
b	40
c	0

?



SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3
b	4

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

SQL2

Out

eid	sm
a	30
b	40
c	0

?

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql ,
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational pattern (structure): the notation-independent abstract relational structure represented by e: how output Out are derived from table-reference roles (...).

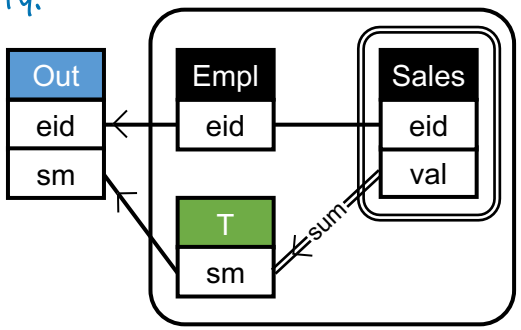
Query expression e: a well-formed syntactic object, written in the notation of a particular query language L.

SQL1

Out

eid	sm
a	30
b	40
b	40
c	0

In Diagram modality:




```
SQL1
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Language convention
- Duplicate convention: bag-preserving

Language convention
- Duplicate convention: set-oriented

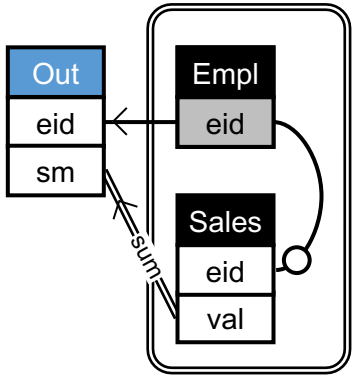
Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

Soufflé

Out

eid	sm
a	30
b	40
c	0




```
SQL2
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3
b	4

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

SQL2

Out

eid	sm
a	30
b	40
c	0

?

Semantic conventions C: language-level defaults that affect denotation but are not reflected in a relational pattern (e.g., duplicate, null, empty-aggregate behavior). Explicit constructs may override defaults.

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql ,
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a query?

Relational pattern (structure): the notation-independent abstract relational structure represented by e: how output Out are derived from table-reference roles (...).

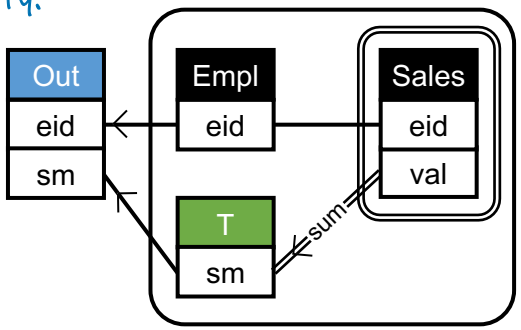
Query expression e: a well-formed syntactic object, written in the notation of a particular query language L.

SQL1

Out

eid	sm
a	30
b	40
b	40
c	0

In Diagram modality:




```
SQL1
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Language convention
- Duplicate convention: bag-preserving

Language convention
- Duplicate convention: set-oriented

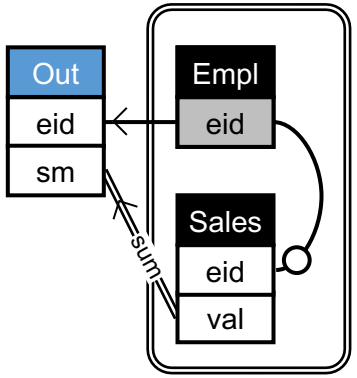
Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

Soufflé

Out

eid	sm
a	30
b	40
c	0




```
SQL2
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3
b	4

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

SQL2

Out

eid	sm
a	30
b	80
c	0

Schema $\mathcal{S}$: relation signatures + integrity constraints (e.g., keys) that determine admissible database instances $\mathcal{I}_S$

Semantic conventions C: language-level defaults that affect denotation but are not reflected in a relational pattern (e.g., duplicate, null, empty-aggregate behavior). Explicit constructs may override defaults.

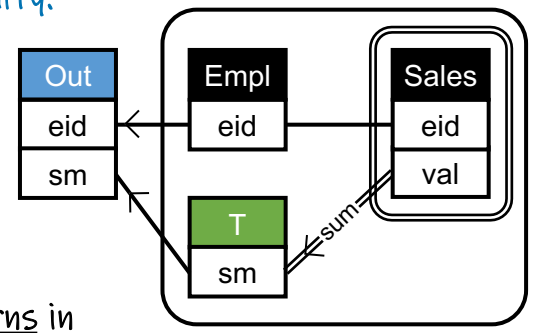
SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql ,
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Semantic Background $B = (S, C)$ is implicit

Relational pattern (structure): the notation-independent abstract relational structure represented by e: how output Out are derived from table-reference roles, up to the semantic background B.

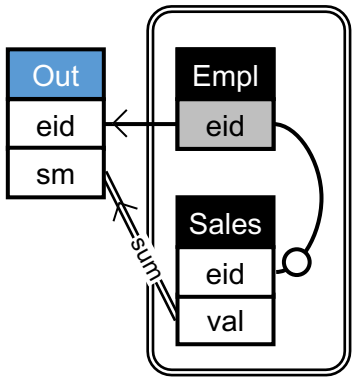
Query expression e: a well-formed syntactic object, written in the notation of a particular query language L.

In Diagram modality:



2 relational patterns in diagrammatic modality.

They realize the same relational intent under schema eid as key in Empl and no duplicates in Sales, and convention sum({})=0 and sum({null})=0



SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Language convention
- Duplicate convention: bag-preserving

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

Language convention
- Duplicate convention: set-oriented

3 query expressions in 2 different languages, realizing 2 different relational patterns

SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

Semantic conventions C: language-level defaults that affect denotation but are not reflected in a relational pattern (e.g., duplicate, null, empty-aggregate behavior). Explicit constructs may override defaults.

Schema S: relation signatures + integrity constraints (e.g., keys) that determine admissible database instances $\mathcal{I}_S$

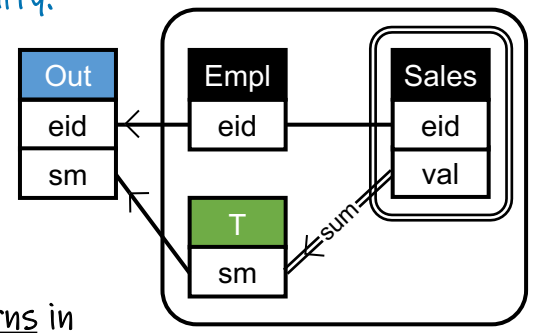
Semantic background $B = (S, C)$: implicit assumptions and conventions usually not encoded in a relational pattern or query expression e

Relational Intent ⇐ Relational Pattern + Semantic background

Relational pattern (structure): the notation-independent abstract relational structure represented by e: how output Out are derived from table-reference roles, up to the semantic background B.

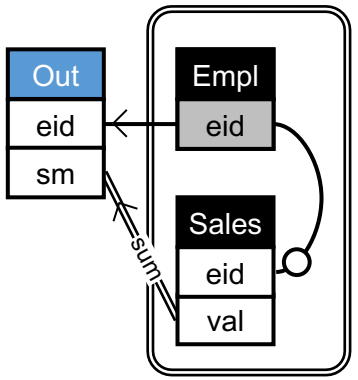
Query expression e: a well-formed syntactic object, written in the notation of a particular query language L.

In Diagram modality:



2 relational patterns in diagrammatic modality.

They realize the same relational intent under schema eid as key in Empl and no duplicates in Sales, and convention sum({})=0 and sum({null})=0



SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Language convention
- Duplicate convention: bag-preserving

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

Language convention
- Duplicate convention: set-oriented

3 query expressions in 2 different languages, realizing 2 different relational patterns

SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Relational intent, denotation q: relation-valued mapping from admissible input instances to output relation values: $I \mapsto q(I)$ for $I \in \mathcal{I}_S$

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

restricts ($\mathcal{I}_S$)

Schema $\mathcal{S}$: relation signatures + integrity constraints (e.g., keys) that determine admissible database instances $\mathcal{I}_S$

Semantic conventions C: language-level defaults that affect denotation but are not reflected in a relational pattern (e.g., duplicate, null, empty-aggregate behavior). Explicit constructs may override defaults.

Semantic background $B = (\mathcal{S}, C)$: implicit assumptions and conventions usually not encoded in a relational pattern or query expression e

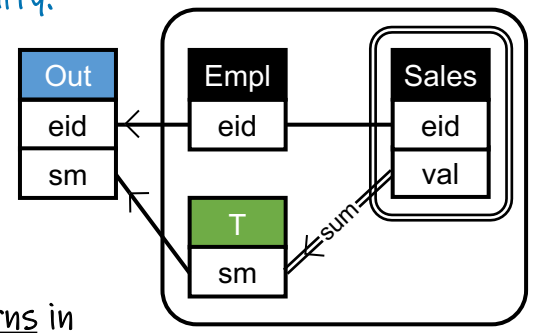
Relational Intent + Instance: can answer a Query Intent

Query intent: information need or question, independent of a schema S and language L. An evaluated query may or may not answer query intent correctly.

informal & not our focus today

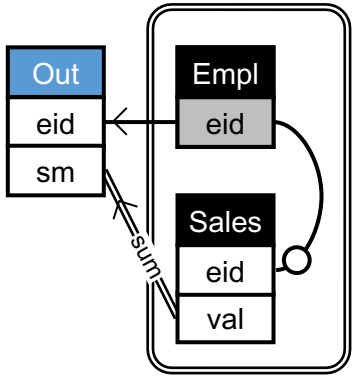
Relational pattern (structure): the notation-independent abstract relational structure represented by e: how output Out are derived from table-reference roles, up to the semantic background B.

In Diagram modality:



2 relational patterns in diagrammatic modality.

They realize the same relational intent under schema eid as key in Empl and no duplicates in Sales, and convention sum({})=0 and sum({null})=0



Query expression e: a well-formed syntactic object, written in the notation of a particular query language L.

SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Language convention
- Duplicate convention: bag-preserving

Soufflé (Datalog extension)

```
Q(x, sum y: {Sales(x, y,_)}) :- Empl(x,_).
```

Language convention
- Duplicate convention: set-oriented

3 query expressions in 2 different languages, realizing 2 different relational patterns

SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Relational intent, denotation q: relation-valued mapping from admissible input instances to output relation values: $I \mapsto q(I)$ for $I \in \mathcal{I}_S$

Empl

eid	k
a	1
b	2
c	3

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
c	0

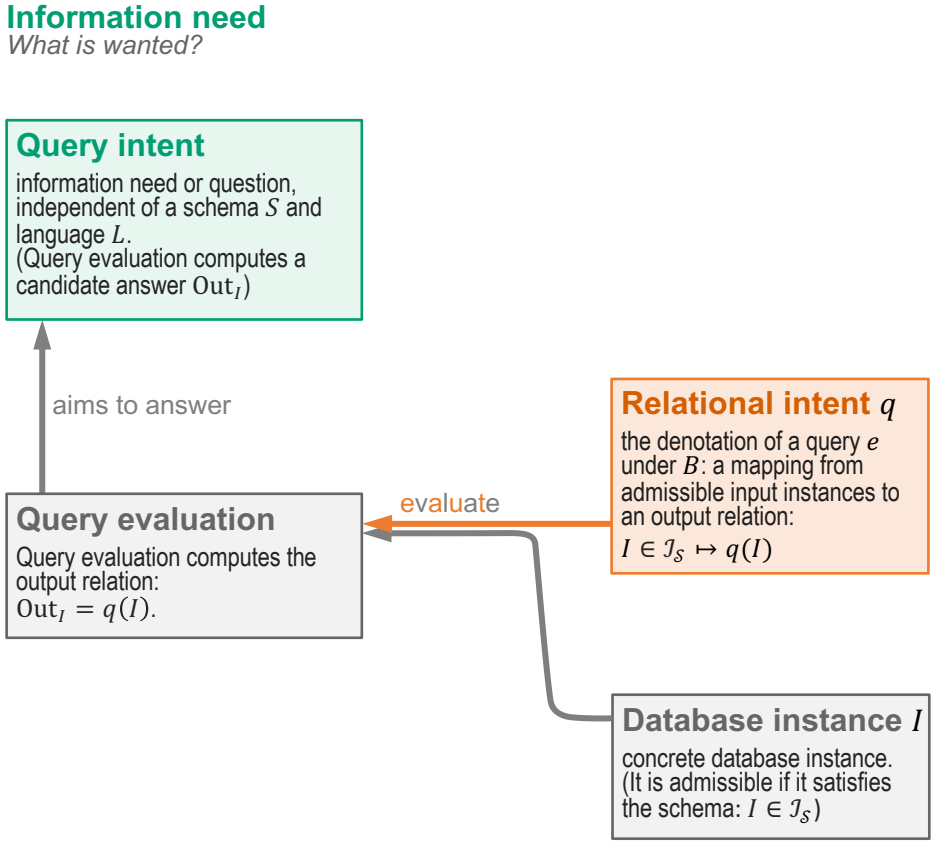
restricts ($\mathcal{I}_S$)

Schema $\mathcal{S}$: relation signatures + integrity constraints (e.g., keys) that determine admissible database instances $\mathcal{I}_S$

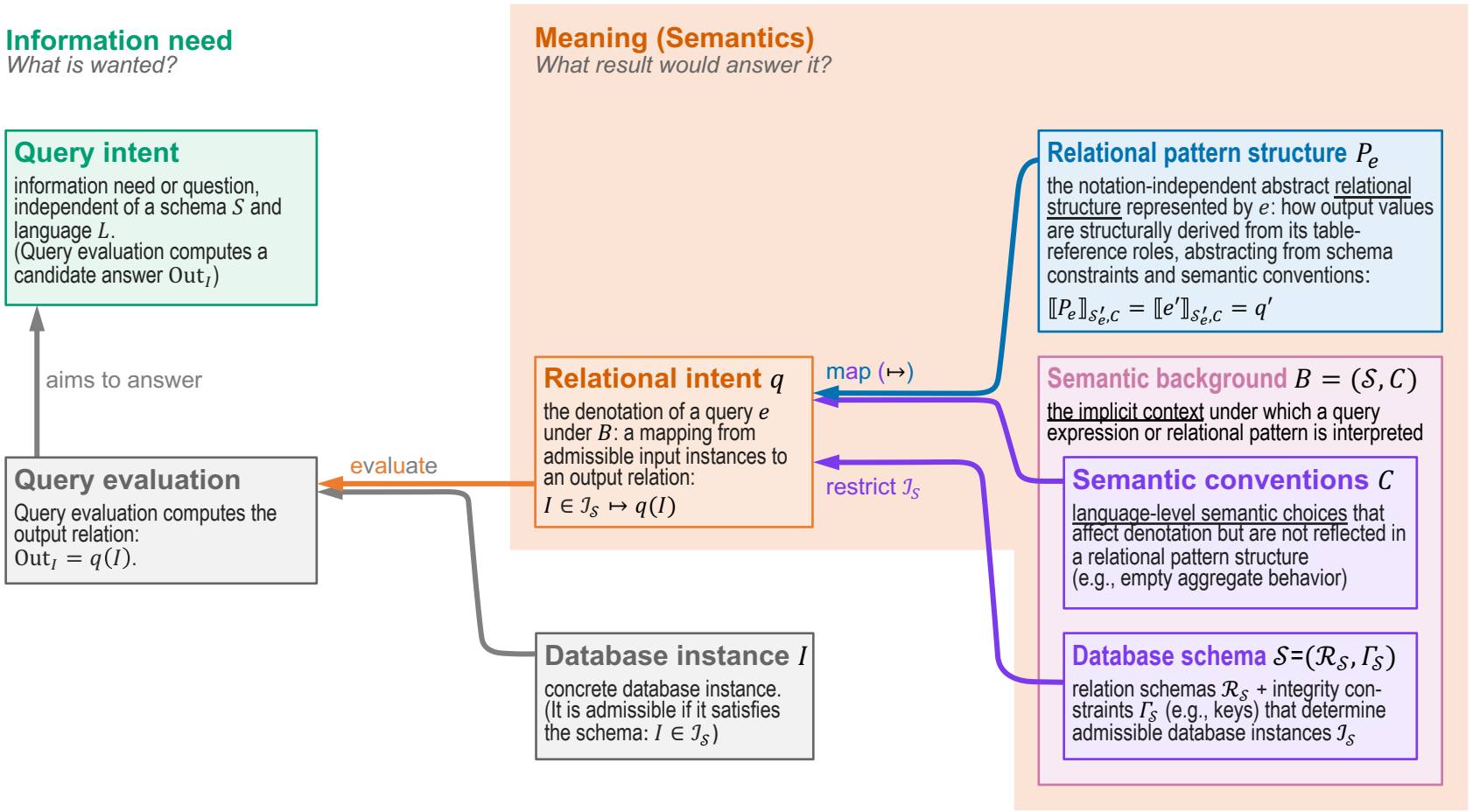
Semantic conventions C: language-level defaults that affect denotation but are not reflected in a relational pattern (e.g., duplicate, null, empty-aggregate behavior). Explicit constructs may override defaults.

Semantic background $B = (S, C)$: implicit assumptions and conventions usually not encoded in a relational pattern or query expression e

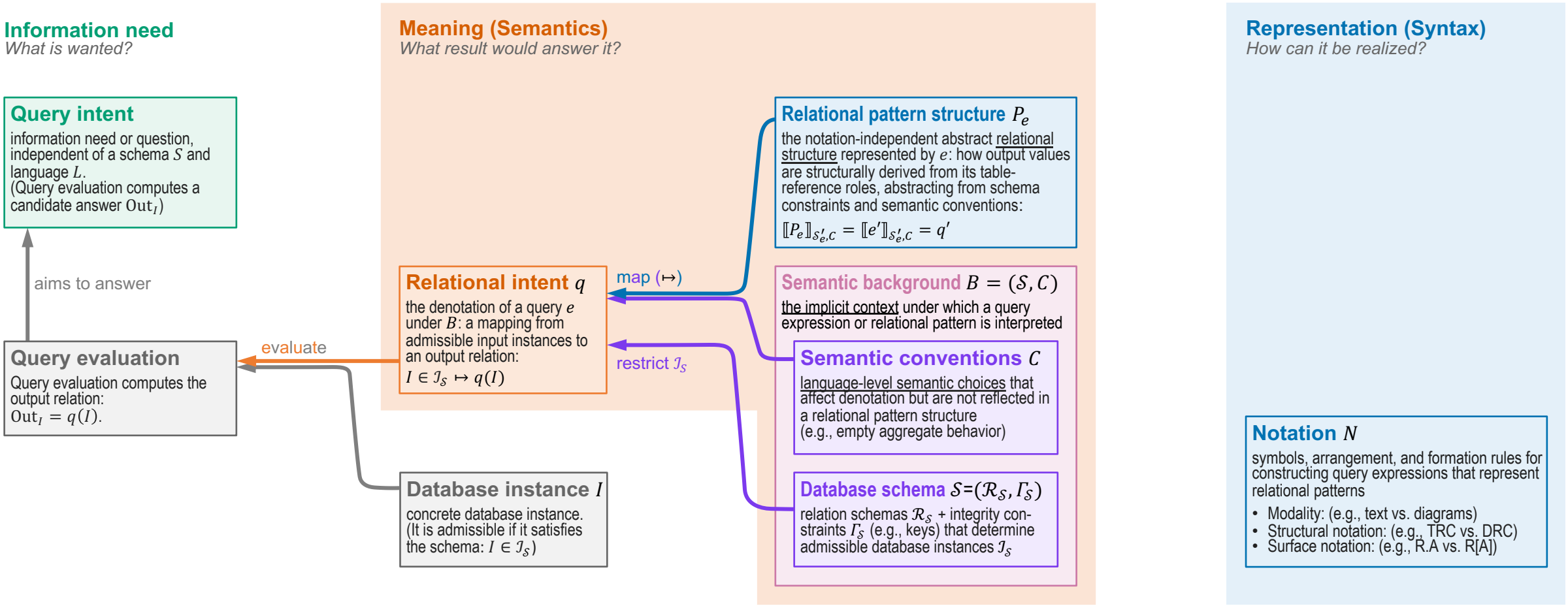
Vocabulary for Relational Language Design



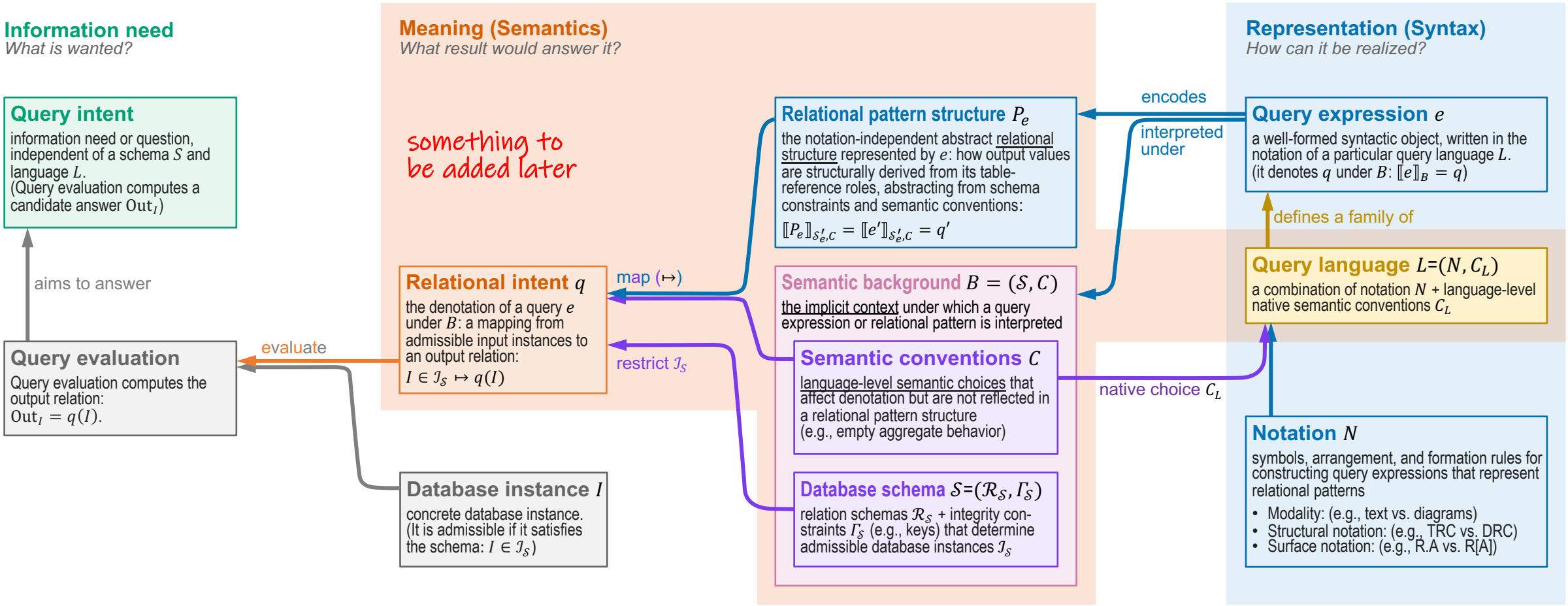
Vocabulary for Relational Language Design



Vocabulary for Relational Language Design



Vocabulary for Relational Language Design



Vocabulary: notation (syntax)

Notation N
the externally represented scheme of symbols, arrangement conventions, and formation rules by which query expressions are constructed and recognized.

2. structural notation: *the modality-independent structural notational design choices that determine how a relational pattern is encoded in a representation (e.g., nested vs. pipelined composition, named vs. positional access to relation components, tuple vs. domain variables)*

2

Domain RC — *Tuple RC*

R(x,_), S(x,_)

R(x,y), S(x,z)

∃r∈R, ∃s∈S [r.A=s.A]

∃r∈R, ∃s∈S (r[A]=s[A])

(∃r) ∧ (∃s)(S(s) ∧ R(r) ∧ r.A=s.A)

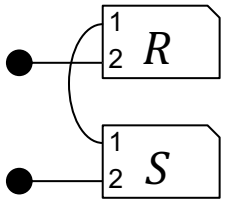
r<-rs, s<-ss, a r == a s

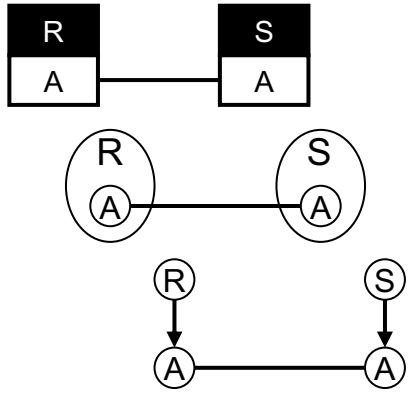
3

3. surface notation: *the modality-specific notational choices that affect only local spelling, syntax, naming, or layout (e.g., "R.A" vs. "R[A]" vs. "A(R)")*

1

1. modality: *the external representational form in which notation is realized (e.g., text or diagrams)*





3

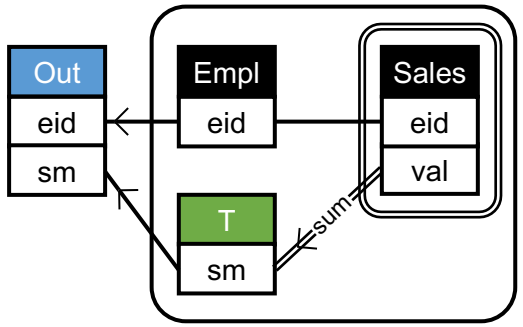
A quick Quiz

What is a relational pattern?

Out

SQL1

eid	sm
a	30
b	40
b	40
c	null



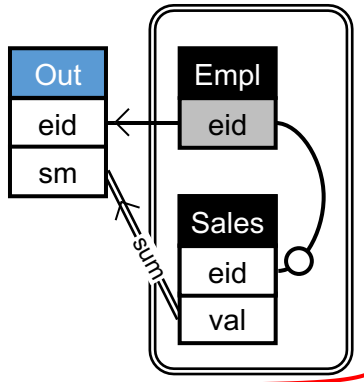
SQL1

```
select E.eid,
  (select sum(val) sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

Under native SQL conventions, SQL1 and SQL3 agree on every admissible instance.

Do they have the same pattern?

?



SQL2

```
select E.eid, sum(val) sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3
b	4

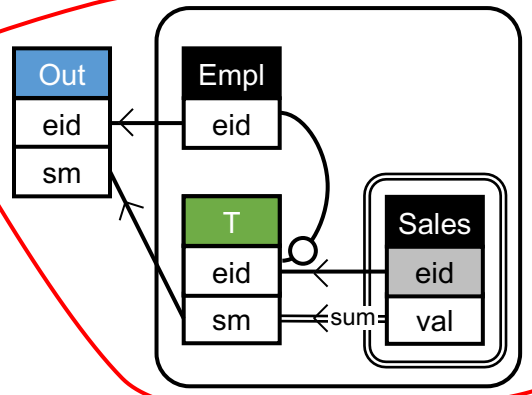
Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

SQL2

Out

eid	sm
a	30
b	80
c	null



SQL3

```
select E.eid, T.sm sm
from Empl E
left join (
  select S.eid, sum(val) sm
  from Sales S
  group by S.eid ) T
on E.eid = T.eid
```

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql ,
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a relational pattern?

SQL3'

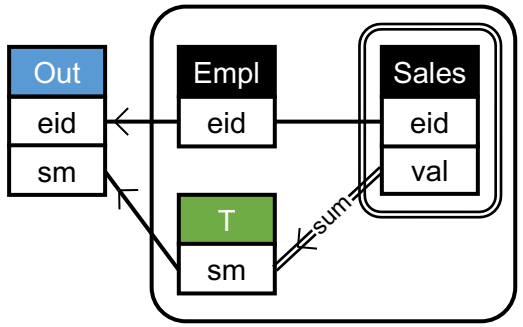
Out

eid	sm
a	30
b	40
b	40
c	null

SQL1'

Out

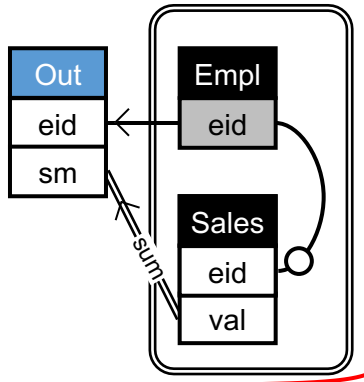
eid	sm
a	30
b	40
b	40
c	0



SQL1'

```
select E.eid,
  (select sum0(val) sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

No, because the equality relied on left joins and empty group having the same "null" behavior. We could break equality by changing the sum behavior alone!



SQL2'

```
select E.eid, sum0(val) sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3
b	4

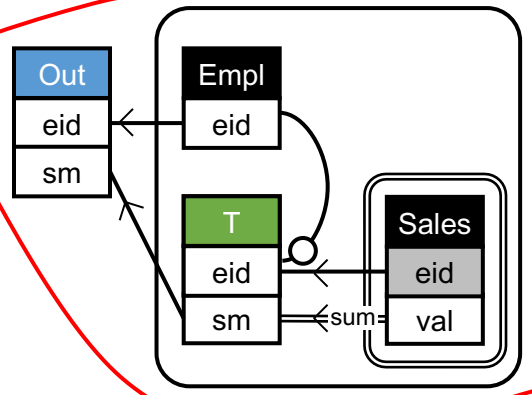
Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

SQL2'

Out

eid	sm
a	30
b	80
c	0



SQL3'

```
select E.eid, T.sm sm
from Empl E
left join (
  select S.eid, sum0(val) sm
  from Sales S
  group by S.eid ) T
on E.eid = T.eid
```

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql ,
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a relational pattern?

Out

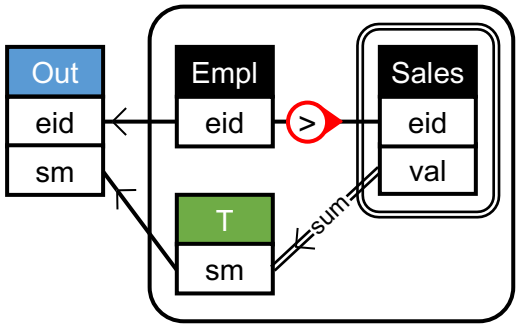
SQL3"

eid	sm
a	null
b	30
b	30
c	30
c	40

Out

SQL1"

eid	sm
a	null
b	30
b	30
c	70



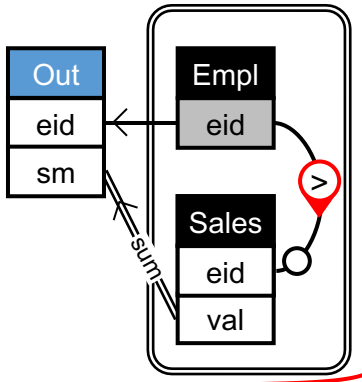
SQL1"

```
select E.eid,
  (select sum(val) sm
  from Sales S
  where E.eid > S.eid)
from Empl E
```

Or by changing the matching condition as we did here!

How to decorrelate SQL1"?

?



SQL2"

```
select E.eid, sum(val) sm
from Empl E
left outer join Sales S
on E.eid > S.eid
group by E.eid
```

Empl

eid	k
a	1
b	2
c	3
b	4

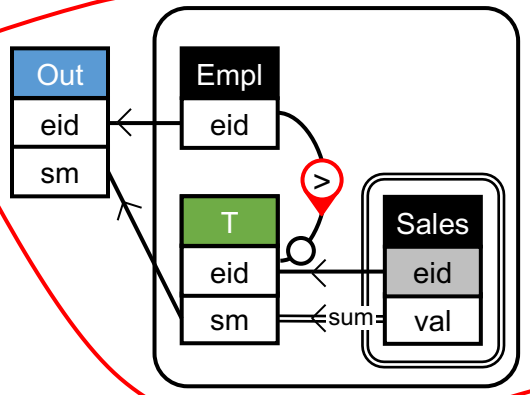
Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

SQL2"

Out

eid	sm
a	30
b	60
c	70



SQL3"

```
select E.eid, T.sm sm
from Empl E
left join (
  select S.eid, sum(val) sm
  from Sales S
  group by S.eid ) T
on E.eid > T.eid
```

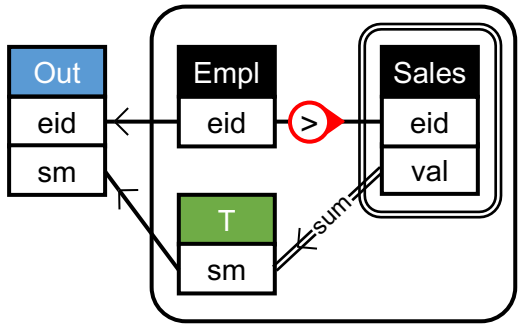
SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql ,
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is a relational pattern?

Out

SQL1" /SQL4

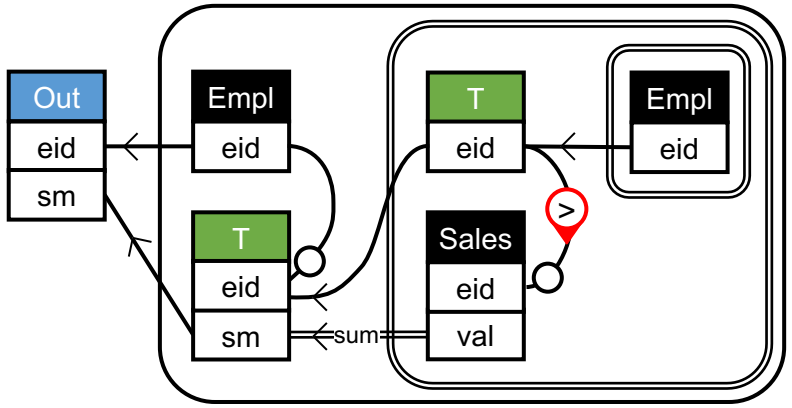
eid	sm
a	null
b	30
b	30
c	70



SQL1"

```
select E.eid,
  (select sum(val) sm
  from Sales S
  where E.eid > S.eid)
from Empl E
```

If a language does not support correlated queries, then the rewrite becomes far more complicated with non-equi-join conditions



SQL4

```
select E.eid, T.sm sm
from Empl E
left join (
  select X.eid, sum(val) sm
  from (select distinct E2.eid
        from Empl E2) X
  left join Sales S
  on X.eid > S.eid
  group by X.eid ) T
on E.eid = T.eid
```

Empl

eid	k
a	1
b	2
c	3
b	4

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql ,
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Outline

- What is a relational query? A vocabulary
- Various aspects of relational languages
  - Join queries
  - Lateral Joins
  - Declarativity of Languages
  - Grouping
  - Linear Composition
  - Relational Division
- Outlook

PLEASE ask questions and let me know if you spot any error anywhere ☺

An anonymous feedback form is linked from the tutorial web page:

Languages we will look at

- SQL
- Tuple Relational Calculus (TRC) and its generalization Abstract Relational Calculus (ARC)
- Domain Relational Calculus (DRC)
- Relational Algebra (RA)
- Soufflé (a variant of Datalog)
- Relational Diagrams (as notional device to understand patterns)
- Ibis (as representative of dataframe languages)
- Rel (by RelationalAI)
- Morel
- Haskell
- Pig Latin
- Klug's formalism for aggregate operators over sets [Klug 1982]
- SQL Pipe Syntax: https://docs.cloud.google.com/bigquery/docs/reference/standard-sql/pipe-syntax
- SaneQL
- Aggregate Logic [Hella+'01]

ARC: Gatterbauer, Sabale. *Database Research needs an Abstract Relational Query Language*, CIDR 2026. https://www.vldb.org/cidrdb/papers/2026/p27-gatterbauer.pdf; **Soufflé**: by Bernhard Scholz and team: https://souffle-lang.github.io/, https://souffle-lang.github.io/pdf/SouffleLanguage.pdf; **Relational Diagrams**: Gatterbauer. *A Principled Solution to the Disjunction Problem of Diagrammatic Query Representations*, SIGMOD 2026. https://doi.org/10.1145/3786617; **Ibis**: Wes McKinney and Ibis Project authors. *The Ibis Project: the portable Python dataframe library*. Version 12.0.0. 2026. https://ibis-project.org/; **Rel**: by RelationalAI: Aref et al. *Relational Programming in Rel*, SIGMOD record 2026. https://doi.org/10.1145/3810900.3810918; **Morel**: by Julian Hyde: https://github.com/hydromatic/morel; **SaneQL**: Neumann, Leis. *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; **Pig Latin**: Olston, Reed, Srivastava, Kumar, Tomkins. *Pig Latin: a not-so-foreign language for data processing*, SIGMOD 2008. https://doi.org/10.1145/1376616.1376726; **Klug**. *Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions*, JACM, 1982. https://doi.org/10.1145/322326.322332; **SQL Pipe Syntax**: Shute et al. *SQL Has Problems. We Can Fix Them: Pipe Syntax In SQL*, PVLDB 2024. https://vldb.org/pvldb/vol17/p4051-shute.pdf; **[Hella+'01]**: Hella, Libkin, Nurmonen, Wong. *Logics with aggregate operators*, JACM 2001. https://doi.org/10.1145/502090.502100
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Aspects of Relational Languages we will discuss

- binding structure: tuple vs. domain variables
- named vs. positional access to relation components
- pointwise (variable-based) vs. point-free (tacit) notation,
- set vs. bag semantics (and list semantics)
- declarative vs. procedural style,
- nested vs. pipelined composition
- and grouping from the inside out (FIO) vs. from the outside in (FOI)
- negation patterns
- correlated queries / lateral joins / scalar queries
- derived attributes (as in matrix multiplication)

Outline

- What is a relational query? A vocabulary
- Various aspects of relational languages
  - Join queries
  - Lateral Joins
  - Declarativity of Languages
  - Grouping
  - Linear Composition
  - Relational Division
- Outlook

PLEASE ask questions and let me know if you spot any error anywhere ☺

An anonymous feedback form is linked from the tutorial web page:

Languages vs. Modalities

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

SQL

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

Datalog

$$Q(x) :- \text{Left}(x,\_), \text{Right}(x,\_)$$

Tuple Relational Calculus (TRC)

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

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
```

assignment can be from left or right (logically equivalent)

Datalog

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

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

Tuple Relational Calculus (TRC)

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

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

assignment of the output from Left

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]}

The tuple perspective is more explicit

Languages vs. Modalities

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

Languages

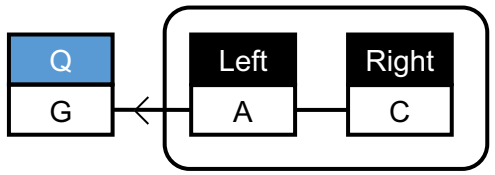
SQL

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

modality: alternative representation of the intent tailored to different audiences

Datalog

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



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*

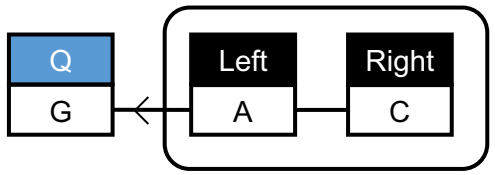
Modalities

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*

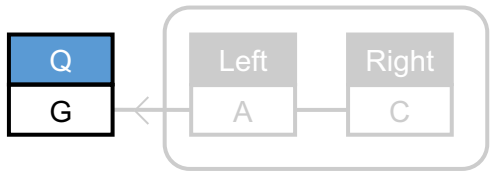
Modalities

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)

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

```
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*

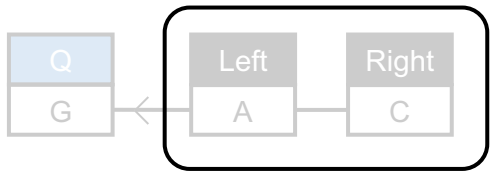
Modalities

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*

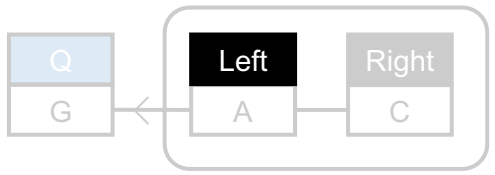
Modalities

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)

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

```
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*

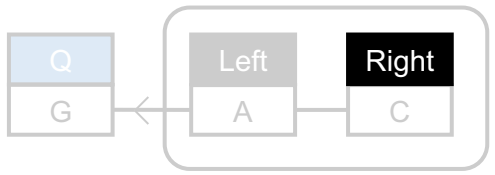
Modalities

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

conjunction = juxtaposition

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***

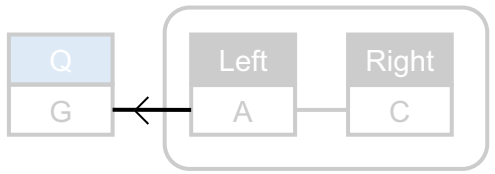
Modalities

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)

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

```
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***

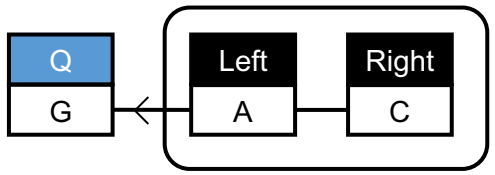
Modalities

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

AST vs. Abstract Language Tree (ALT)

Left(A,B), Right(C,D)

SQL

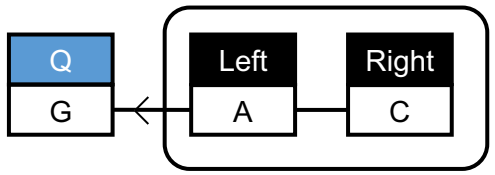
```
select A as G
from Left, Right
where A = C
```

```
Query
└─ SELECT
   ├─ SELECTLIST
   │  └─ L.A as G
   ├─ FROM
   │  └─ JOIN(comma)
   │     ├─ TABLE: "Left" as L
   │     └─ TABLE: "Right" as R
   └─ WHERE
      └─ EXPRESSION: L.A = R.C
```

Simplified AST from ZetaSQL

```
SELECT
├─ EXPRESSIONS
│  └─ L.A as G
├─ FROM
│  └─ TABLE: Left as L
├─ JOINS
│  └─ TABLE: Right as R
└─ WHERE
   └─ EXPRESSION: L.A = R.C
```

Simplified AST from SQLGlot



Relational Diagram

Tuple Relational Calculus (TRC)

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

Set comprehension based textual notation

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

Abstract Language Tree

Modalities

Bindings:
tuples / domain
named / positional
pointwise / pointfree

Variable bindings

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

~Datalog (Soufflé)

```
Q(x) :-
  Left(x, u), Right(v, _), u<v.
```

SQL

```
select A as G
from Left, Right
where B < C
```

~Datalog (Rel)

```
def Q(x) : exists((u, v) |
  Left(x, u) and Right(v, _) and u<v)
```

```
select L.A as G
from Left L, Right R
where L.B < R.C
```

?

DRC

$$\{Q(x) \mid \exists u, v, y \; [Left(x,u) \wedge Right(v, y) \wedge u<v]\}$$

TRC

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

value-oriented (or domain-oriented) — tuple-oriented†

What do the variables range over?

† Tuple Relational Calculus (or Tuple Calculus) should be called "Relation Calculus" according to [Date 2004] to mirror the name Domain RC. An alternative would be to replace domain with value and use "value relational calculus".
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Variable bindings

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

~Datalog (Souffle)

```
Q(x) :-
  Left(x, u), Right(v, _), u<v.
```

~Datalog (Rel)

```
def Q(x) : exists((u, v) |
  Left(x, u) and Right(v, _) and u<v)
```

SQL

```
select A as G
from Left, Right
where B < C
```

```
select L.A as G
from Left L, Right R
where L.B < R.C
```

Dataframe (Ibis)

```
Q = (
    Left
    .cross_join(Right)
    .filter(Left.B < Right.C)
    .select(G=Left.A))
```

DRC

$$\{Q(x) \mid \exists u, v, y \; [Left(x,u) \wedge Right(v, y) \wedge u<v]\}$$

TRC

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

RA

$$\rho_{A \to G} \left(\pi_A \left(Left \bowtie_{B<C} Right\right)\right)$$

value-oriented (or domain-oriented) tuple-oriented[†]

What do the variables range over?

(pointfree, tacit): no variables (smallest unit are relations) vs. variable-based or pointwise[*]

Binding unit (Binding granularity): Does the language have variables, and what do they range over? Or does the language operate primarily pointfree over relations? What is the unit of reference?

[*] "Pointwise" because the values of function arguments are sometimes called points (as in "the value of f at point x")

[†] Tuple Relational Calculus (or Tuple Calculus) should be called "Relation Calculus" according to [Date 2004] to mirror the name Domain RC. An alternative would be to replace domain with value and use "value relational calculus".

Access to relation components

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

?

DRC

$$\{Q(x) \mid \exists u, v, y \; [Left(x,u) \wedge Right(v, y) \wedge u<v]\}$$

TRC

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

RA

$$\rho_{A \to G} (\pi_A (Left \bowtie_{B<C} Right))$$

value-oriented (or domain-oriented) — tuple-oriented

What do the variables range over?

(pointfree, tacit): no variables (smallest unit are relations) vs. variable-based or pointwise

Binding unit (Binding granularity): Does the language have variables, and what do they range over? Or does the language operate primarily pointfree over relations? What is the unit of reference?

Access to relation components

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

Access to relation components: are components referenced by name or by position?

attribute positions

DRC

$$\{Q(x) \mid \exists u, v, y \; [Left(x,u) \wedge Right(v, y) \wedge u<v]\}$$

?

?

names

?

TRC

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

RA

$$\rho_{A \to G} (\pi_A (Left \bowtie_{B<C} Right))$$

value-oriented (or domain-oriented) — tuple-oriented

What do the variables range over?

(pointfree, tacit): no variables (smallest unit are relations) vs. variable-based or pointwise

Binding unit (Binding granularity): Does the language have variables, and what do they range over? Or does the language operate primarily pointfree over relations? What is the unit of reference?

Access to relation components

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

Access to relation components: are components referenced by name or by position?

attribute positions

DRC

$$\{Q(x) \mid \exists u, v, y \; [Left(x,u) \wedge Right(v, y) \wedge u<v]\}$$

TRC (w/ positional access)

$$\{Q^{(1)} \mid \exists l \in Left, r \in Right \; [Q.\$1 = l.\$1 \wedge l.\$2 < r.\$1]\}$$

RA (unnamed perspective)

$$\pi_{\$1} (Left \bowtie_{\$2<\$4} Right)$$

$$\pi_{\$1} (\sigma_{\$2<\$4} (Left \times Right))$$

names

?

TRC

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

RA

$$\rho_{A \to G} (\pi_{A} (Left \bowtie_{B<C} Right))$$

value-oriented (or domain-oriented) — tuple-oriented

What do the variables range over?

(pointfree, tacit): no variables (smallest unit are relations) vs. variable-based or pointwise

Binding unit (Binding granularity): Does the language have variables, and what do they range over? Or does the language operate primarily pointfree over relations? What is the unit of reference?

Access to relation components

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

Access to relation components: are components referenced by name or by position?

attribute positions

DRC

$$\{Q(x) \mid \exists u, v, y \; [Left(x,u) \wedge Right(v, y) \wedge u<v]\}$$

TRC (w/ positional access)

$$\{Q^{(1)} \mid \exists l \in Left, r \in Right \; [Q.\$1=l.\$1 \wedge l.\$2<r.\$1]\}$$

RA (unnamed perspective)

$$\pi_{\$1} (Left \bowtie_{\$2<\$4} Right)$$

$$\pi_{\$1} (\sigma_{\$2<\$4} (Left \times Right))$$

names

DRC (with named access)

$$\{Q(G=x) \mid \exists x, u, v, y \; [Left(A=x, B=u) \wedge Right(C=v, D=y) \wedge u<v]\}$$

TRC

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

RA

$$\rho_{A \to G} (\pi_A (Left \bowtie_{B<C} Right))$$

value-oriented (or domain-oriented) | tuple-oriented | (pointfree, tacit): no variables (smallest unit are relations) vs. variable-based or pointwise

What do the variables range over?

Binding unit (Binding granularity): Does the language have variables, and what do they range over? Or does the language operate primarily pointfree over relations? What is the unit of reference?

Named vs Unnamed (positional) perspective

R

A	B
1	3
1	4
2	2

Unnamed (positional) perspective:

$R(1,3)$

Named perspective:

A tuple is an (unordered) mapping from a set of attribute names to values

$\{A \mapsto 1, B \mapsto 3\}$ Other notations: $\{A : 1, B : 3\}$ or $\{A = 1, B = 3\}$

A relation signature is is a finite map from attribute names to types (or domains or sorts).

$\tau = \{A : \text{int}, B : \text{int}\}$

A relation schema associates a relation name with a signature

$R : \tau$

A relation (value) is a set (or bag) of tuples conforming to the same signature τ.

$\{\{A \mapsto 1, B \mapsto 3\}, \{A \mapsto 1, B \mapsto 4\}, \{A \mapsto 2, B \mapsto 2\}\}$

Different TRC notations

Likes(person, drink)
Frequents(person, bar)
Serves(bar, drink)

Return persons who frequent only bars that serve at least one drink they like.
(Return persons for whom there does not exist a bar they frequent that does not serve any drink they like.)
Return persons who frequent at least one bar, and for every bar they frequent, that bar serves at least one drink they like.

Uppercase notation to overload both output tuple variable with the actual name of the resulting table

{Q(person) | ∃f∈Frequents [Q.person=f.person ∧ ¬(∃f2∈Frequents [f2.person=f.person ∧ ¬(∃l∈Likes, s∈Serves [l.drink=s.drink ∧ f2.bar=s.bar ∧ f2.person=l.person])])]} — my preferred notation ARC [CIDR'26]

repeated quantifiers do not need to be repeated

{q(person) | ∃f∈Frequents [q.person=f.person ∧ ¬(∃f2∈Frequents [f2.person=f.person ∧ ¬(∃l∈Likes, ∃s∈Serves [l.drink=s.drink ∧ f2.bar=s.bar ∧ f2.person=l.person])])]} — my preferred notation TRC [SIGMOD'26]

{t: Person | ∃f∈Frequents [t(Person)=f(Person) ∧ ¬∃f2∈Frequents [f2(person)=f(person) ∧ ¬(∃l∈Likes ∃s∈Serves) [l(Drink)=s(Drink) ∧ f2(Bar)=s(Bar) ∧ f2(Person)=l(Person)]]]} — [Deutsch'19]

allows re-using of range variables for the output, even with projection

{f.Person | Frequents(f) AND (NOT(∃f2)(Frequents(f2) AND f2.person=f.person AND (NOT(∃l)(∃s)(Likes(l) AND Serves(s) AND l.drink=s.drink AND f2.bar=s.bar AND f2.person=l.person)))))} — [Elmasri+'15]

Notations that separate creation of a tuple variable from its assignment to a table

$\{\mu^{(1)} \mid (\exists\rho^{(2)})\ (\text{Frequents}(\rho) \wedge \rho[1]=\mu[1] \wedge \neg((\exists\lambda^{(2)})(\text{Frequents}(\lambda) \wedge \lambda[1]=\rho[1] \wedge \neg((\exists\nu^{(2)})(\exists\theta^{(2)})(\text{Likes}(\nu) \wedge \text{Serves}(\theta) \wedge \nu[2]=\theta[2] \wedge \lambda[2]=\theta[1] \wedge \lambda[1]=\nu[1])))))\}$ — [Ullman'88]

{P | ∃F∈Frequents (F.person=P.person ∧ ¬∃F2∈Frequents(F2.person=F.person ∧ ¬(∃L∈Likes ∃S∈Serves (L.drink=S.drink ∧ F2.bar=S.bar ∧ F2.person=L.person))))} — [Ramakrishnan+'03]

{t | ∃f∈Frequents (t[Person]=f[Person] ∧ ¬∃f2∈Frequents (f2[person]=f[person] ∧ ¬(∃l∈Likes ∃s∈Serves) (l[Drink]=s[Drink] ∧ f2[Bar]=s[Bar] ∧ f2[Person]=l[Person])))} — [Silberschatz+'20]

scope of negation is applied only to range variables

ARC notation: "*Database Research needs an Abstract Relational Query Language*, CIDR 2026. https://www.vldb.org/cidrdb/papers/2026/p27-gatterbauer.pdf", TRC notation: "*A Principled Solution to the Disjunction Problem of Diagrammatic Query Representations*", SIGMOD 2026. https://arxiv.org/pdf/2412.08583 , Deutsch (based on Vianu), CSE132A: Database System Principles, fall 2019. https://cseweb.ucsd.edu/classes/fa19/cse132A-a/slides/relational-calculus.pdf. Elmasri , Navathe. "*Fundamentals of database systems*", (7 ed), 2015. https://dl.acm.org/doi/book/10.5555/2842853 , Ullman. "*Principles of Database and Knowledge-base Systems*", Vol. 1, 1988. https://dl.acm.org/doi/book/10.5555/42790 , Ramakrishnan, Gehrke. "*Database Management Systems*" (3 ed), 2003. https://dl.acm.org/doi/book/10.5555/560733. Silberschatz, Korth, Sudarshan. "*Database System Concepts*", 7th ed., McGraw-Hill, 2020. https://www.db-book.com/. SQL database 331 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Lateral Join

1. Inner Join in Python

```python
# result = [(x, y) for x in X for y in Y if x < y]
for x in X:
  for y in Y:
    cond = "True" if (x<y) else ""
    print(f"({x}, {y}): {cond}")
```

```
(1, 1):
(1, 2): True
(1, 3): True
(1, 4): True
(2, 1):
(2, 2):
(2, 3): True
(2, 4): True
(3, 1):
(3, 2):
(3, 3):
(3, 4): True
(4, 1):
(4, 2):
(4, 3):
(4, 4):
```

```
X = [1, 2, 3, 4]
Y = [1, 2, 3, 4]
```

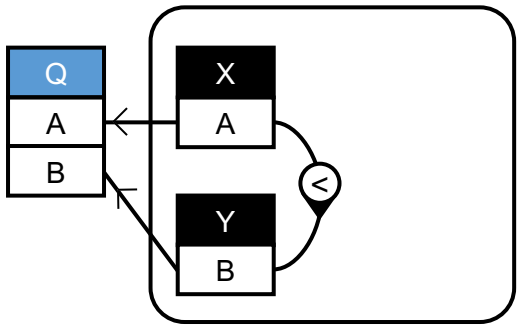
X	Y
A	B
1	1
2	2
3	3
4	4

```
select X.A, Y.B
from X, Y
where X.A<Y.B
```

```
select X.A, Y.B
from X
inner join Y
on X.A<Y.B
```

A	B
1	2
1	3
1	4
2	3
2	4
3	4

```
COLLECTION
├─ HEAD: Q(A,B)
└─ QUANTIFIER ∃
   ├─ BINDING: x ∈ X
   ├─ BINDING: y ∈ Y
   └─ AND ∧
      ├─ PREDICATE: x.A < y.B
      ├─ PREDICATE: Q.A = x.A
      └─ PREDICATE: Q.B = y.B
```



Python and SQL examples available as 735 at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql, For details on the collection syntax see: "*Database Research needs an Abstract Relational Query Language*", CIDR 2026. https://www.vldb.org/cidrdb/papers/2026/p27-gatterbauer.pdf
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

2. Lateral Join in Python

```
# result = [(x, z) for x in X for z in (y for y in Y if x < y)]
for x in X:
  Z = [y for y in Y if x < y]
  for z in Z:
    cond = "True" if (x<z) else ""
    print(f"({x}, {z}): {cond}")
```

```
(1, 2): True
(1, 3): True
(1, 4): True
(2, 3): True
(2, 4): True
(3, 4): True
```

```
X = [1, 2, 3, 4]
Y = [1, 2, 3, 4]
```

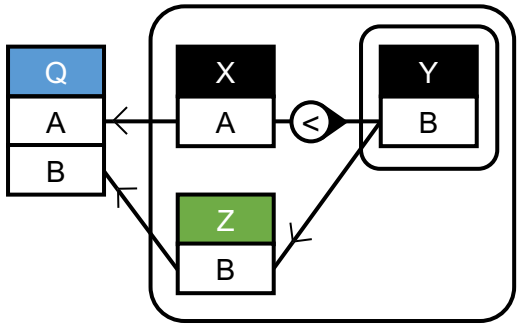
X	Y
A	B
1	1
2	2
3	3
4	4

lateral join allows reference to X

```
select X.A, Z.B
from X
cross join lateral
   (select Y.B
   from Y
   where X.A<Y.B) Z
```

A	B
1	2
1	3
1	4
2	3
2	4
3	4

```
COLLECTION
├─ HEAD: Q(A,B)
└─ QUANTIFIER ∃
   ├─ BINDING: x ∈ X
   ├─ BINDING: z ∈
   └─ COLLECTION
      ├─ HEAD: Z(B)
      └─ QUANTIFIER ∃
         ├─ BINDING: y ∈ Y
         └─ AND ∧
            ├─ PREDICATE: x.A < y.B
            └─ PREDICATE: Z.B = y.B
   └─ AND ∧
      ├─ PREDICATE: Q.A = x.A
      └─ PREDICATE: Q.B = z.B
```



Python and SQL examples available as 735 at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql, For details on the collection syntax see: "*Database Research needs an Abstract Relational Query Language*", CIDR 2026. https://www.vldb.org/cidrdb/papers/2026/p27-gatterbauer.pdf
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

3. Nested Correlated Query

```
# result = [x for x in X if any(x < y for y in Y)]
for x in X:
  cond = "True" if any(x < y for y in Y) else ""
  print(f"{x}: {cond}")
```

```
1: True
2: True
3: True
4:
```

```
X = [1, 2, 3, 4]
Y = [1, 2, 3, 4]
```

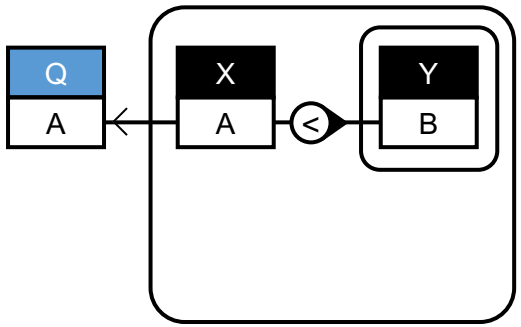
X	Y
A	B
1	1
2	2
3	3
4	4

nested correlated subquery references X

```
select X.A
from X
where exists
  (select 1
  from Y
  where X.A<Y.B)
```

A
1
2
3

```
COLLECTION
├─ HEAD: Q(A)
└─ QUANTIFIER ∃
   ├─ BINDING: x ∈ X
   └─ AND ∧
      ├─ PREDICATE: Q.A = x.A
      └─ QUANTIFIER ∃
         ├─ BINDING: y ∈ Y
         └─ PREDICATE: x.A < y.B
```



Python and SQL examples available as 735 at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql, For details on the collection syntax see: "*Database Research needs an Abstract Relational Query Language*", CIDR 2026. https://www.vldb.org/cidrdb/papers/2026/p27-gatterbauer.pdf
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Outline

- What is a relational query? A vocabulary
- Various aspects of relational languages
  - Join queries
  - Lateral Joins
  - Declarativity of Languages
  - Grouping
  - Linear Composition
  - Relational Division
- Outlook

PLEASE ask questions and let me know if you spot any error anywhere ☺

An anonymous feedback form is linked from the tutorial web page:

Declarative vs. Procedural Languages

R(A,C), S(B,D)

SQL

```
select R.A, S.B
from R, S
where R.A < S.B
```

Relational Pattern

Theta-join projection: Define the result as the selected attributes of all pairs of tuples, one from each input collection, whose values satisfy a given comparison predicate.

Instantiation:

- First collection: R, bound as R (in SQL).
- Second collection: S, bound as S (in SQL).
- Pair condition: R.A < S.B.
- Returned attribute from first collection: R.A.
- Returned attribute from second collection: S.B.

Declarative vs. Procedural Languages

R(A,C), S(B,D)

SQL

```
select R.A, S.B
from R, S
where R.A < S.B
```

Relational Algebra

$$\pi_{A,B}\,(R \bowtie_{A<B} S)$$

TRC without dedicated output variable (e.g., [Elmasri,Navathe'15])

Tuple Relational Calculus (TRC)

$\{(r.A,s.B) \mid r \in R, s \in S\ [r.A<s.B]\}$

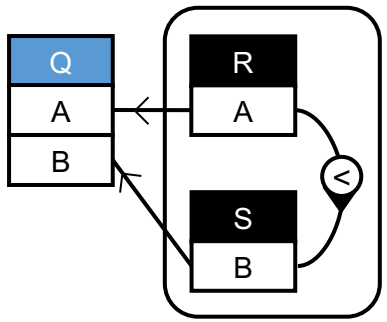
$\{q(A,B) \mid \exists r \in R, s \in S\ [q.A=r.A \wedge q.B=s.B \wedge r.A<s.B]\}$

output

predicate

a range variable s that "ranges over" relation S (non-deterministically)

Relational Diagram



Declarative: what result is meant.
Procedural: how to compute it.

What is here declarative and what not ?

Relational Pattern

Theta-join projection: Define the result as the selected attributes of all pairs of tuples, one from each input collection, whose values satisfy a given comparison predicate.

Instantiation:

- First collection: R, bound as R (in SQL).
- Second collection: S, bound as S (in SQL).
- Pair condition: `R.A < S.B`.
- Returned attribute from first collection: `R.A`.
- Returned attribute from second collection: `S.B`.

List Comprehension

Tuple Relational Calculus (TRC)

$\{(r.A,s.B) \mid r \in R, s \in S[r.A<s.B]\}$

Relational calculus: a pair (r.A, s.B) is in the answer iff the predicate holds.
Interpretation: these are the tuples that satisfy the predicate.

Domain Relational Calculus (DRC)

$\{(a,b) \mid \exists x,y\ [rs(a,x) \wedge ss(b,y) \wedge a < b]\}$

Relational calculus: a pair (a, b) is in the answer iff the predicate holds.
Interpretation: these are the tuples that satisfy the predicate.

Haskell List Comprehension

```
[(a, b) | (a, _) <- rs, (b, _) <- ss, a < b]
```

generators guard

qualifiers

Is this declarative ?

List Comprehension

Tuple Relational Calculus (TRC)

$\{(r.A,s.B) \mid r \in R, s \in S[r.A<s.B]\}$

Relational calculus: a pair (r.A, s.B) is in the answer iff the predicate holds. Interpretation: these are the tuples that satisfy the predicate.

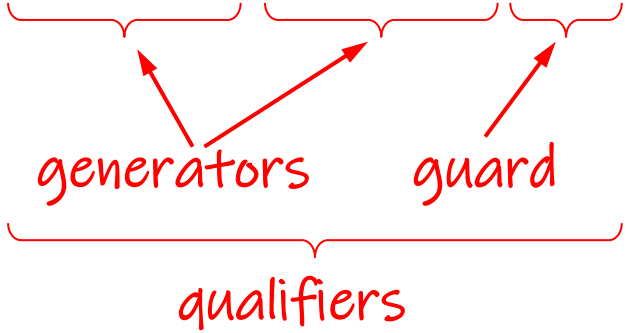
Domain Relational Calculus (DRC)

$\{(a,b) \mid \exists x,y\ [rs(a,x) \wedge ss(b,y) \wedge a < b]\}$

Relational calculus: a pair (a, b) is in the answer iff the predicate holds. Interpretation: these are the tuples that satisfy the predicate.

Haskell List Comprehension

```
[(a, b) | (a, _) <- rs, (b, _) <- ss, a < b]
```



Can be interpreted declaratively as: all pairs (a,b) such that some tuple in rs has first component a, some tuple in ss has first component b, and a < b

Can also be interpreted operationally as:
1. take each tuple from *rs* and bind the first component to *a*
2. take each tuple from *ss* and bind its first component to *b*
3. keep only those combinations where *a* < *b*
4. yield (*a*,*b*)

Haskell list comprehensions have a declarative surface resemblance to calculus. But their semantics is operational over ordered lists (generator order, output order, duplicates, and termination behavior are observable).

Haskell nested loops

```
for each (a, _) in rs
   for each (b, _) in ss
      if a < b
         produce (a, b)
```

Explicit nested loops (procedural): Now we have a specific control flow and enumeration strategy. Can be read as: "Enumerate tuples in this order, by this control structure."

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'08]

We have developed a new language called *Pig Latin* that combines the best of both worlds: high-level declarative querying in the spirit of SQL, and low-level, procedural programming à la map-reduce.

Note that although Pig Latin programs supply an explicit sequence of operations, it is not necessary that the operations be executed in that order.

In effect, writing a Pig Latin program is similar to specifying a query execution plan (i.e., a dataflow graph), thereby making it easier for programmers to understand and control how their data processing task is executed.

> "I much prefer writing in Pig [Latin] versus SQL. The step-by-step method of creating a program in Pig [Latin] is much cleaner and simpler to use than the single block method of SQL. It is easier to keep track of what your variables are, and where you are in the process of analyzing your data." – Jasmine Novak, Engineer, Yahoo!

in Pig Latin, a user specifies a sequence of steps where each step specifies only a *single*, high-level data transformation. This is stylistically different from the SQL approach where the user specifies a set of declarative constraints that collectively define the result.

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)>10^6;
output = FOREACH big_groups GENERATE
            category, AVG(good_urls.pagerank);
```

```
SELECT category, AVG(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 10^6
```

Olston, Reed, Srivastava, Kumar, Tomkins. Pig Latin: a not-so-foreign language for data processing, SIGMOD 2008. https://doi.org/10.1145/1376616.1376726
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Two interpretations of a "declarative statement"

Declarative: what result is meant (instead of procedural: how to compute it)

1) membership specification (predicate specification / property-based / characterizing answers / satisfaction based / membership test)

- a Boolean predicate that a solution must satisfy
- semantics is defined independently of any enumeration strategy

$\{q(A) \mid \exists r \in R\ [q.A=r.A]\}$

Is this declarative according to this definition ?

Two interpretations of a "declarative statement"

Declarative: what result is meant (instead of procedural: how to compute it)

1) membership specification (predicate specification / property-based / characterizing answers / satisfaction based / membership test)

- a Boolean predicate that a solution must satisfy
- semantics is defined independently of any enumeration strategy

$\{q(A) \mid \exists r \in R\ [q.A=r.A]\}$

- Yes, if "$\exists r \in R\ [q.A=r.A]$" is seen as Boolean predicate
- No, if "$r \in R$" is interpreted as generator

2) result specification (execution-independent specification / rewrites allowed)

- specification does not commit to an enumeration strategy (sometimes called "descriptive")
- a conceptual algorithm whose observable result defines the contract, while execution may use optimized alternatives that preserve its high-level relational structure

$\pi_{A,B}\ (R \bowtie_{A<B} S)$

Is this declarative according to this definition ?

Two interpretations of a "declarative statement"

Declarative: what result is meant (instead of procedural: how to compute it)

1) membership specification (predicate specification / property-based / characterizing answers / satisfaction based / membership test)

- a Boolean predicate that a solution must satisfy
- semantics is defined independently of any enumeration strategy

$\{q(A) \mid \exists r \in R\ [q.A=r.A]\}$

- Yes, if "$\exists r \in R\ [q.A=r.A]$" is seen as Boolean predicate
- No, if "$r \in R$" is interpreted as generator

2) result specification (execution-independent specification / rewrites allowed)

- specification does not commit to an enumeration strategy (sometimes called "descriptive")
- a conceptual algorithm whose observable result defines the contract, while execution may use optimized alternatives that preserve its high-level relational structure

$\pi_{A,B}\,(R \bowtie_{A<B} S)$

- Yes, if we allow rewrites
- No, if we interpret the statement as literal evaluation strategy

Outline

- What is a relational query? A vocabulary
- Various aspects of relational languages
  - Join queries
  - Lateral Joins
  - Declarativity of Languages
  - Grouping
  - Linear Composition
  - Relational Division
- Outlook

PLEASE ask questions and let me know if you spot any error anywhere ☺

An anonymous feedback form is linked from the tutorial web page:

Grouping & Aggregates: FIO (From the Inside Out) vs. FOI (From the Outside In)

Aggregates: FIO (From the Inside-Out) vs. FOI

R

A	B
a	1
a	2
b	3
b	4
c	5
e	6

Q

A	sm
a	3
b	7
c	5
e	6

SQL

```
select R.A, sum(B) sm
from R
group by R.A
```

RA

$$\gamma_{\text{A, sum(B)}\rightarrow\text{sm}}(\text{R})$$

Rel (~Datalog)

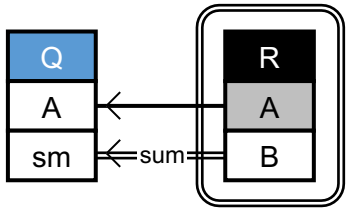
```
def Q(a, sm) : sm = sum[(b) : R(a,b)]
def Q[a] : sum[(b) : R(a,b)]
```

Soufflé (~Datalog)

```
Q(a, sm) :- R(a,_), sm=sum b: {R(a, b)}.
Q(a, sum b: {R(a, b)}) :- R(a,_).
```

Abstract Relational Calculus (ARC)

```
COLLECTION
├─ HEAD: Q(A,sm)
└─ QUANTIFIER Γ
   ├─ BINDING: r ∈ R
   ├─ GROUPING: r.A
   └─ AND ∧
      ├─ PREDICATE: Q.A = r.A
      └─ PREDICATE: Q.sm = sum(r.B)
```



$$\{Q(A, sm) \mid \Gamma(r\in R, \gamma(r.A)\ [Q.A=r.A \wedge Q.sm=sum(r.B)])\}$$

Aggregates: FIO (From the Inside-Out) vs. FOI

SQL

```
select R.A, sum(B) sm
from R
group by R.A
```

RA

$\gamma_{A,\,\mathrm{sum}(B)\to sm}(R)$

SQL

```
select distinct R1.A,
  (select sum(R2.B) sm
  from R R2
  where R1.A = R2.A)
from R R1
```

R

A	B
a	1
a	2
b	3
b	4
c	5
e	6

Q

A	sm
a	3
b	7
c	5
e	6

Rel (~Datalog)

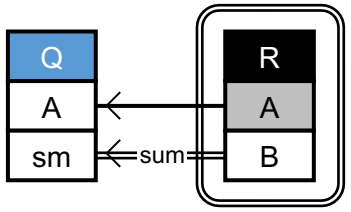
```
def Q(a, sm) : sm = sum[(b) : R(a,b)]
def Q[a] : sum[(b) : R(a,b)]
```

Soufflé (~Datalog)

```
Q(a, sm) :- R(a,_), sm=sum b: {R(a, b)}.
Q(a, sum b: {R(a, b)}) :- R(a,_).
```

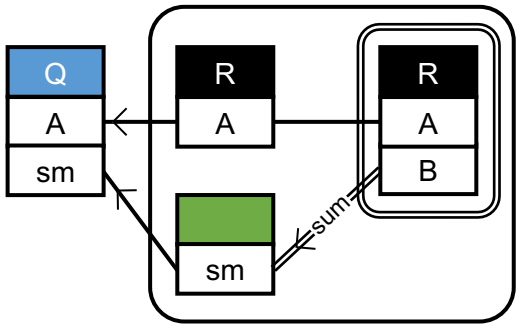
Abstract Relational Calculus (ARC)

```
COLLECTION
├─ HEAD: Q(A,sm)
└─ QUANTIFIER Γ
   ├─ BINDING: r ∈ R
   ├─ GROUPING: r.A
   └─ AND ∧
      ├─ PREDICATE: Q.A = r.A
      └─ PREDICATE: Q.sm = sum(r.B)
```

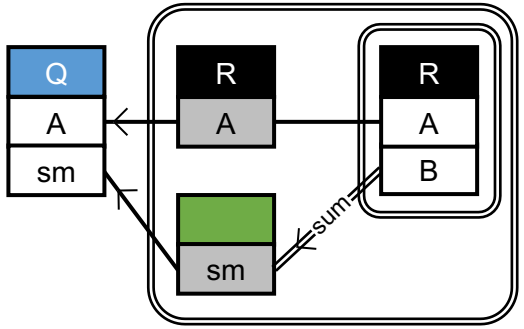


$\{Q(A, sm) \mid \Gamma(r \in R, \gamma(r.A)\ [Q.A=r.A \wedge Q.sm=sum(r.B)])\}$

ARC



set semantics



bag semantics

Database 720 to run SQL queries (you may have to exchange R with S) https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql/, Soufflé: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Aggregates: FIO (From the Inside-Out) vs. FOI

Soufflé (~Datalog)

```
Q(a, sm) :- R(a,_), sm=sum b: {R(a, b)}.
```

Rel (~Datalog)

```
def Q(a, sm) : sm = sum[(b) : R(a,b)]
```

Soufflé (~Datalog)

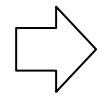
```
Q(a, sm) :- sm=sum y: {R(a, b)}.
```

SQL

```
select distinct R1.A,
  (select sum(R2.B) sm
  from R R2
  where R1.A = R2.A)
from R R1
```

S

A	B
a	1
a	2
b	3
b	4
c	5
e	6



Q

A	sm
a	3
b	7
c	5
e	6

The Witness Problem

You cannot export information from within the body of an aggregate. This means that you cannot ground a variable from within the scope of the aggregate body and expect this grounding to transfer to the outer scope. In this example, `y` is grounded by the inner scope of the aggregate and we attempt to discover its value within the head `a(x, y)`.

```
a(x,y) :- b(x), x = sum z : { c(y, z) }.
```

```
Error: Witness problem: argument grounded by an aggregator's inner scope is used ungrounded in
outer scope in a count/sum/mean aggregate in file 720-1-simplegrouping.dl at line 22
Q(a, sm) :- sm=sum b: { R(a, b) }.   // does not work (witness problem)
-------------------------^-----------------------------------------
```

Witness problem figure source: https://souffle-lang.github.io/aggregates

Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions

ANTHONY KLUG

University of Wisconsin, Madison, Wisconsin

SQL

```
select distinct R1.A,
  (select sum(R2.B) sm
  from R R2
  where R1.A = R2.A)
from R R1
```

S

A	B
a	1
a	2
b	3
b	4
c	5
e	6

Q

A	sm
a	3
b	7
c	5
e	6

Klug's original formalism (~TRC with positional placement)

```
(v1[1],
  sum2((v2[1], v2[2])
    : R(v2) : v2[1] = v1[1]))
  : R(v1)
```

Klug's formalism (translated into dotted positional placement)

```
{(r1.$1,
  sum₂{(r2.$1, r2.$2)
    | r2 ∈ R [r2.$1 = r1.$1]})
  | r1 ∈ R}
```

Klug's formalism (translated into named perspective)

```
{(r1.A,
  sum₂{(r2.A, r2.val)
    | r2 ∈ R [r2.A = r1.A]})
  | r1 ∈ R}
```

Klug. *Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions*, JACM, 1982. https://doi.org/10.1145/322326.322332
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

What is grouping without aggregation?

What happens during grouping before aggregation?

SQL

```
select R.A, sum(R.B) sm
from R
group by R.A
```

Morel

```
from r in R
  group {a = r.a}
    compute {elements}
```

Haskell (GHC.Exts)

```
q =
  [ (the a, b, c)
  | (a, b, c) <- r
  , then group by a using groupWith ]
```

R

A	B	C
a	1	3
a	2	4
b	3	5
b	4	6
c	5	7
e	6	8

`group by R.A`

	A	B	C
a →	a	1	3
	a	2	4
b →	b	3	5
	b	4	6
c →	c	5	7
e →	e	6	8

Q

A	sm
a	3
b	7
c	5
e	6

Q

A	elements
a	A B C a 1 3 a 2 4
b	A B C b 3 5 b 4 6
c	A B C c 5 7
e	A B C e 6 8

Q

A	B	C
a	1 2	3 4
b	3 4	5 6
c	5	7
e	6	8

```
[("a",[1,2],[3,4]),
 ("b",[3,4],[5,6]),
 ("c",[5],  [7]),
 ("e",[6],  [8])]
```

```
q =
  [ (the a, sum b)
  | (a, b, c) <- r
  , then group by a using groupWith]
```

```
[("a",3),
 ("b",7),
 ("c",5),
 ("e",6)]
```

SQL group by typically means: partition, then immediately aggregate into a flat result

Database and file 720 to run SQL and Morel and Haskell (you may have to exchange R with S): https://github.com/northeastern-datalab/cs3200-activities/tree/master/ ,
Jones, Wadler. Comprehensive comprehensions: comprehensions with 'order by' and 'group by', Haskell 2007. https://doi.org/10.1145/1291201.1291209
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Pig Latin: cogroup and group

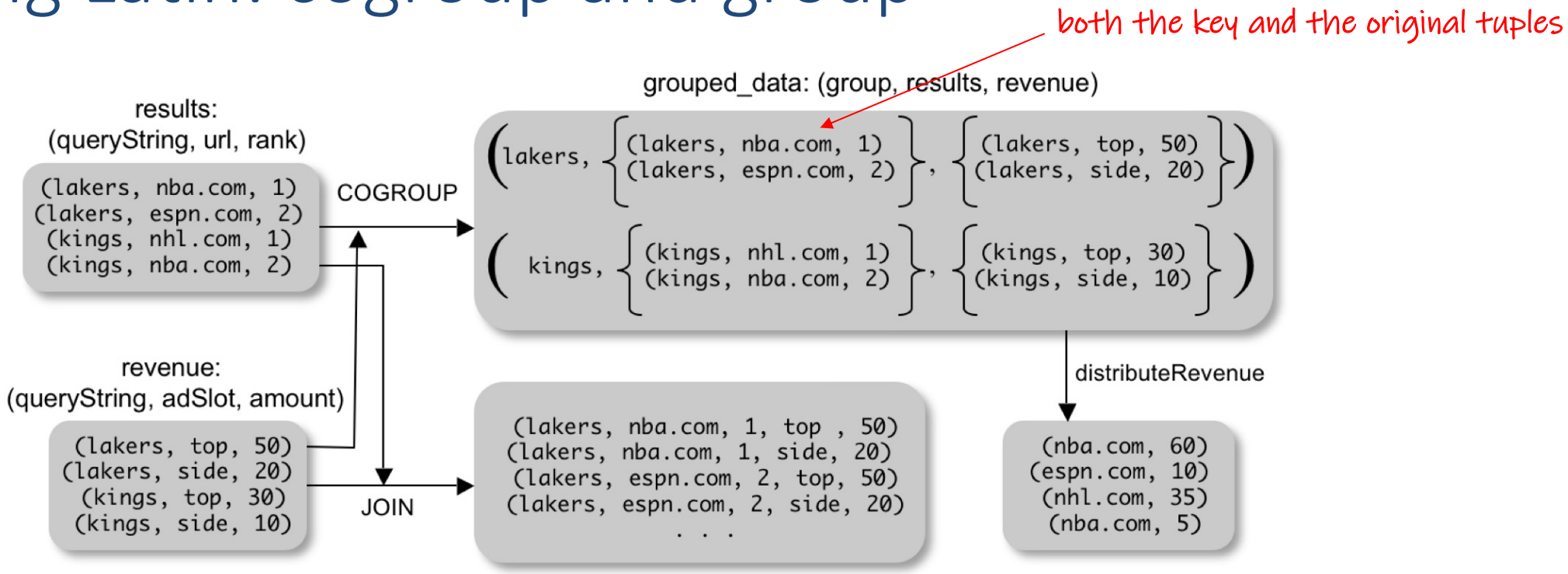


Figure 2: COGROUP versus JOIN.

- Pig Latin: GROUP / COGROUP produces a tuple with a nested bag (non-1NF). Then you decide what to do with it (aggregate or join in the case of COGROUP)
- JOIN = COGROUP + cross product of the nested bags (then flattened)

Olston, Reed, Srivastava, Kumar, Tomkins. Pig Latin: a not-so-foreign language for data processing, SIGMOD 2008. https://doi.org/10.1145/1376616.1376726
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Ibis: Grouped and Ungrouped Aggregates

SQL

```
select S.A, sum(S.B) sm
from S
group by S.A
```

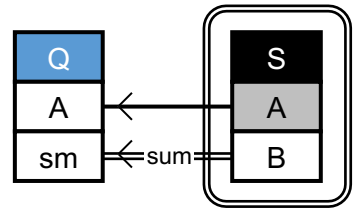
Ibis

```
Q = (S
     .group_by("A")
     .aggregate(sm=S.B.sum()) )
```

group automatically yields the grouping attribute

Q

A	sm
a	3
b	7
c	5
e	6



S

A	B
a	1
a	2
b	3
b	4
c	5
e	6

SQL

```
select sum(S.B) sm
from S
```

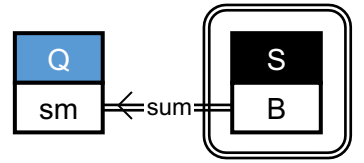
Ibis

```
Q = (S
     .aggregate(sm=S.B.sum()) )
```

aggregate(...) returns a table expression (1-row dataFrame)

Q

sm
21



a table expression with one row and one column named sm (Table[sm: int64])

S.B.sum() returns an integer scalar

```
sm = S.B.sum()
```

21

a scalar aggregate expression (Scalar[int64])

Database and file 720 to run SQL and Ibis: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql/ , https://github.com/northeastern-datalab/cs3200-activities/tree/master/ibis/
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Morel: Grouped and Ungrouped Aggregates

SQL

```
select S.A, sum(S.B) sm
from S
group by S.A
```

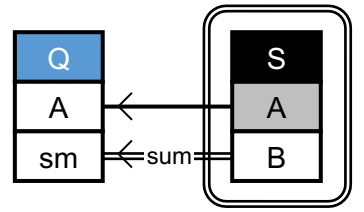
Morel

```
from s in S
  group {A = s.A}
    compute {sm = sum over s.B}
```

group automatically yields the grouping attribute

Q

A	sm
a	3
b	7
c	5
e	6



S

A	B
a	1
a	2
b	3
b	4
c	5
e	6

SQL

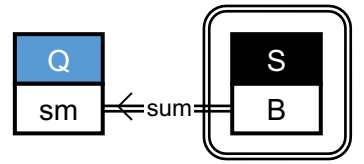
```
select sum(S.B) sm
from S
```

Morel

```
from s in S
  group {}
    compute {sm = sum over s.B}
```

Q

sm
21



[{sm=21}] a table/list with a single record, with one entry ({sm:int} list)

the empty group creates a table/list with a single record or scalar

the compute with curly bracket yields records;

```
from s in S
  group {}
    compute sum over s.B
```

[21] a list with a scalar entry (int list)

```
from s in S
  compute {sm = sum over s.B}
```

{sm=21} a record with a single entry ({sm:int})

w/o grouping, it becomes a record instead of a list of records

without the brackets, the compute yields (a list of) scalar integers instead of records

```
from s in S
  compute sum over s.B
```

21 an integer (int)

Database and file 720 to run SQL and Morel: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql/ , https://github.com/northeastern-datalab/cs3200-activities/tree/master/morel/
Contrast with Alloy's unified treatment. 2012 - Jackson - Software Abstractions -- Logic, Language, and Analysis (book, 2nd ed), p.45: "So there aren't really any scalars in Alloy?" !!!
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Two aggregates

Multiple Aggregates

R

empl	dept
1	a
2	a
3	b
4	c

S

empl	sal
1	100
2	100
3	200
4	50

Q

dpts	av
a	100
b	200

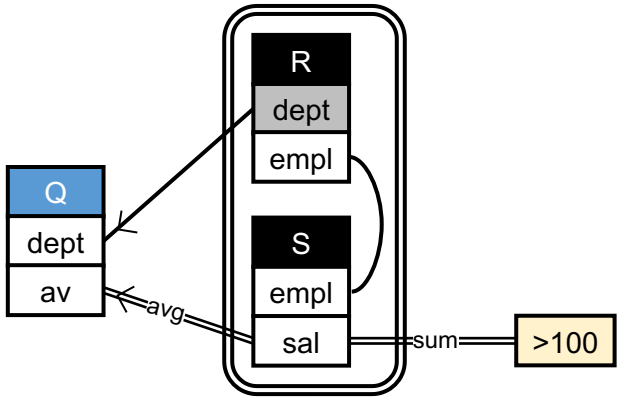
SQL

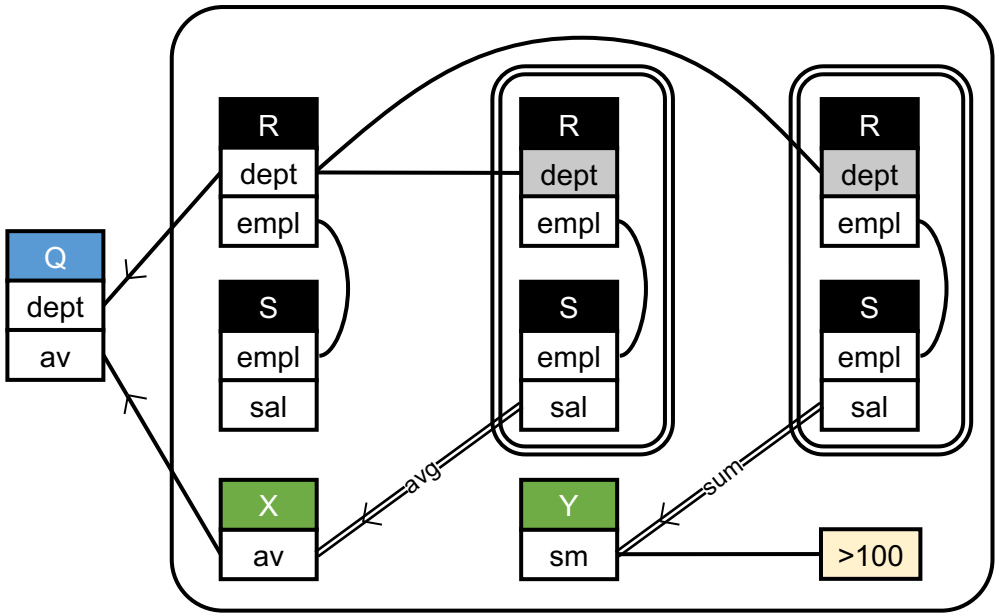
```
select R.dept, avg(S.sal) av
from R, S
where R.empl = S.empl
group by R.dept
having sum(S.sal) > 100
```

For each value of variable y (from the outer scope), iterate over all bound variables (z), i.e. salaries.

[Hella+'01]

$$Q(y, q) := (\exists x \exists z.R(x, y) \wedge S(x, z)) \wedge$$
$$(q = \text{Aggr}_{AVG}\ x, z.(R(x, y) \wedge S(x, z), z)) \wedge$$
$$(\text{Aggr}_{\Sigma}\ x, z.(R(x, y) \wedge S(x, z), z) > c_{100})$$





Multiple Aggregates

R

empl	dept
1	a
2	a
3	b
4	c

S

empl	sal
1	100
2	100
3	200
4	50

Q

dpts	av
a	100
b	200

SQL

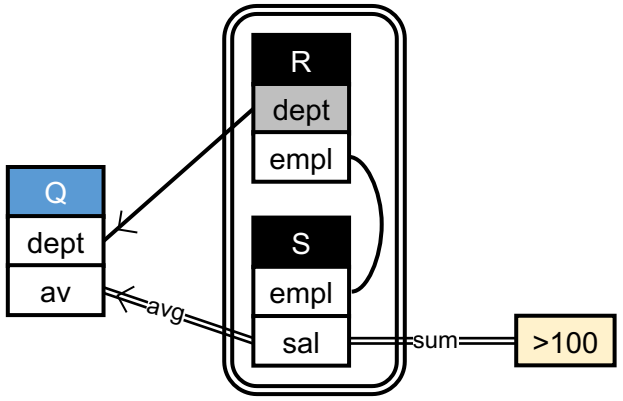
```
select R.dept, avg(S.sal) av
from R, S
where R.empl = S.empl
group by R.dept
having sum(S.sal) > 100
```

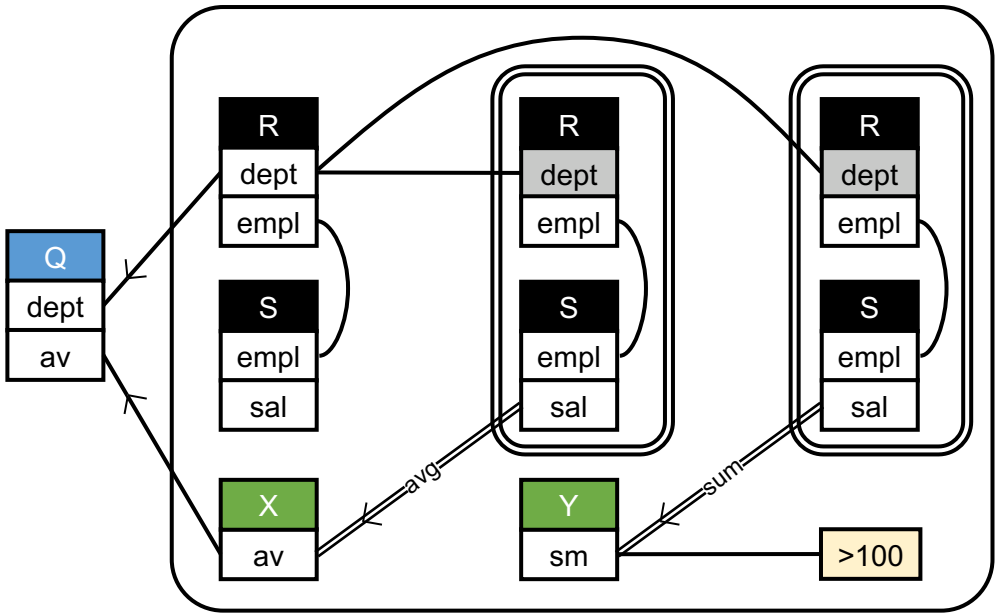
For each value of variable d (from the outer scope), iterate over all bound variables (s), i.e. salaries.

Souffle (~Datalog)

```
Q(d, sm / ct) :- R(e,d), S(e,_),
  sm=sum s: { R(e2, d), S(e2, s) },
  ct =count : { R(e2, d), S(e2, _) },
  sm>100.
```

Similar size as SQL,
but more complicated pattern





Multiple Aggregates

naming template for overlapping columns from the right table

R

A	B
a	1
a	2
b	3
c	4
c	5
d	6

S

A	B
a	1
a	2
b	3
b	4
c	5
e	6

Q

A	av
b	3.5
c	5

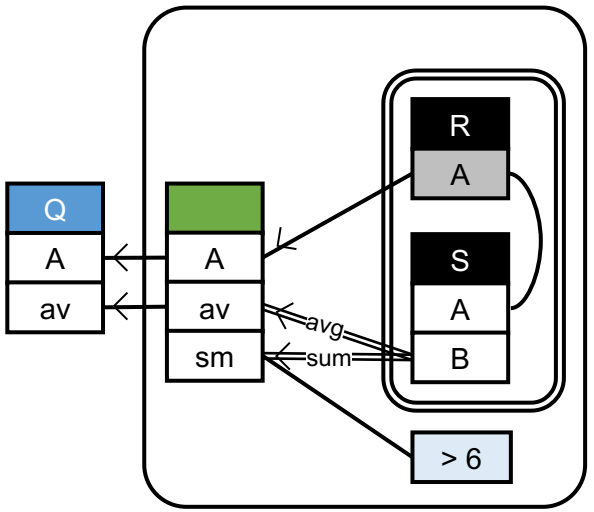
SQL

```
select S.A, avg(S.B) av
from R, S
where R.A = S.A
group by S.A
having sum(S.B) > 6
```

Ibis

```
Q = (R
     .join(S, R.A == S.A, rname = "s_{name}")
     .group_by(A=_.A)
     .aggregate(sm=_.s_B.sum(),
                av=_.s_B.mean() )
     .filter(_.sm > 6)
     .select("A", "av") )
```

The underscore "_" refers to the current table expression at this point in the chain

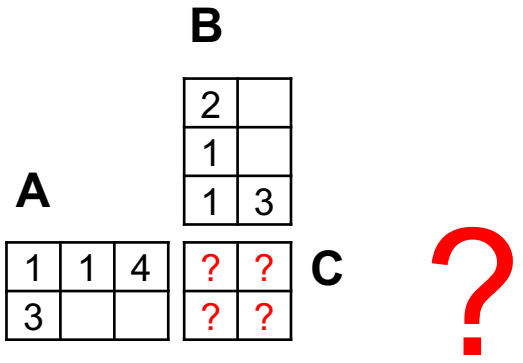


Morel

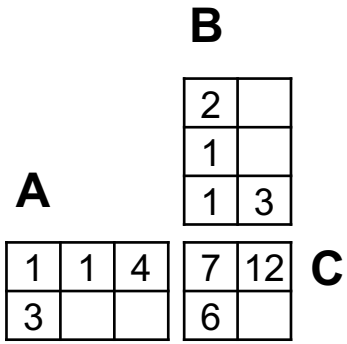
```
from r in R,
     s in S
   where r.a = s.a
   group {a = s.a}
     compute {sm = sum over s.b,
              av = real (sum over s.b) / real (count over ())}
   where sm > 6
   yield {a, av};
```

Matrix Multiplication

Matrix Multiply $C = A \cdot B$



Matrix Multiply $C = A \cdot B$



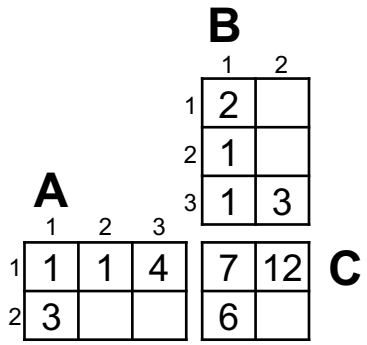
Matrix Multiply $C = A \cdot B$

A

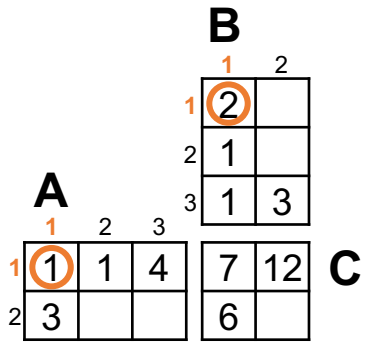
row	col	val
1	1	1
1	2	1
1	3	4
2	1	3

B

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



Matrix Multiply $C = A \cdot B$



input

A

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

B

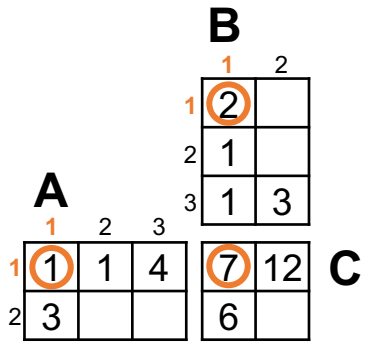
row	col	val
1	1	2
2	1	1
3	1	1
3	2	3

Matrix Multiply $C = A \cdot B$

derived attribute*

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
```



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

derived attribute

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

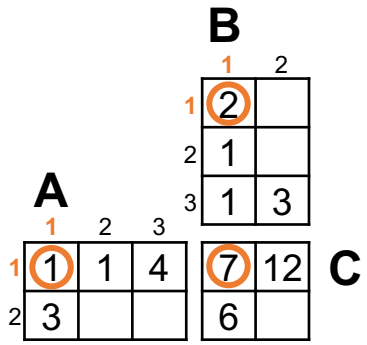
A "derived attribute" is a fresh output attribute whose value is defined by an expression over available attributes
SQL example 670 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Matrix Multiply $C = A \cdot B$

SQL

derived attribute*

```
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
```

former derived attribute, now re-interpreted as domain value in an "external relation"

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

A "derived attribute" is a fresh output attribute whose value is defined by an expression over available attributes
SQL example 670 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Matrix Multiply $C = A \cdot B$

derived attribute*

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
```

former derived attribute, now re-interpreted as domain value in an "external relation"

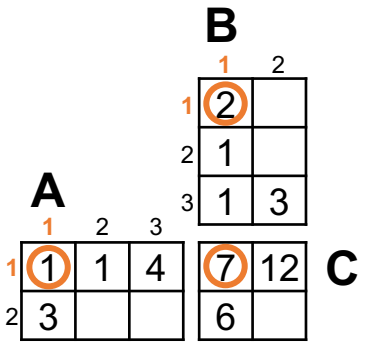
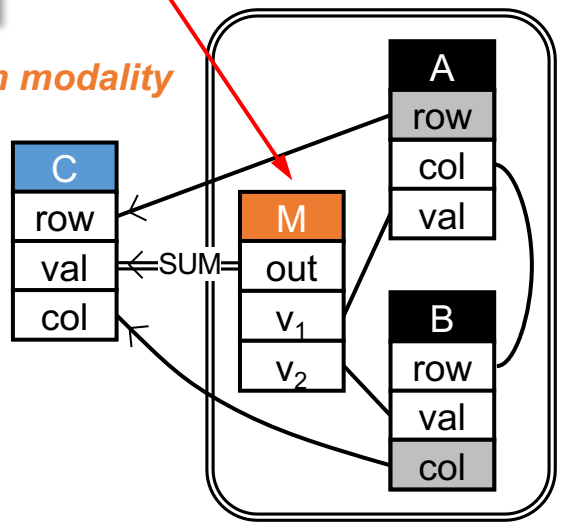


Diagram modality



Abstract Relational Calculus (ARC)

Comprehension Syntax

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

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

A "derived attribute" is a fresh output attribute whose value is defined by an expression over available attributes
SQL example 670 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

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
```

SQL Pipe Syntax

```
from A as a
|> join B as b
   on a.col = b.row
|> aggregate sum(a.val * b.val) as val
   group by a.row, b.col;
```

SaneQL

```
A.join(B, A.col = B.row)
 .group(
   {row := A.row, col := B.col},
   {val := sum(A.val * B.val)} )
```

PRQL

```
from a = A
join b = B (a.col == b.row)
group { row = a.row, col = b.col } (
  aggregate {
    val = sum (a.val * b.val) } )
```

Ibis

```
Q = (A
     .join(B, A.col == B.row, rname="{name}_b")
     .group_by(row=_.row, col=_.col_b)
     .aggregate(val=(_.val * _.val_b).sum()))
```

Morel

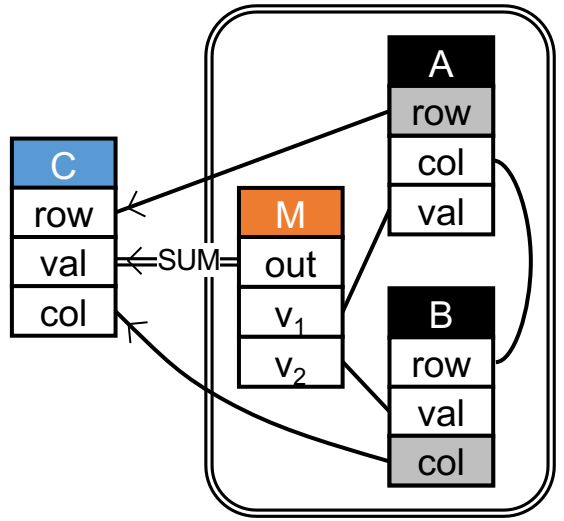
```
from a in A,
     b in B
  where a.col = b.row
  group {row = a.row, col = b.col}
    compute {val = sum over (a.val * b.val)}
```

Soufflé (Datalog extension)

```
C(i, j, sum prod) :- prod = aval * bval,
 A(i, k, aval), B(k, j, bval).
```

Rel

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



Outline

- What is a relational query? A vocabulary
- Various aspects of relational languages
  - Join queries
  - Lateral Joins
  - Declarativity of Languages
  - Grouping
  - Linear Composition
  - Relational Division
- Outlook

PLEASE ask questions and let me know if you spot any error anywhere ☺

An anonymous feedback form is linked from the tutorial web page:

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

Claim: Pipelines are a lot more intuitive 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]
- 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
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

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

Claim: Pipelines are a lot more intuitive 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]
- **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
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

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
```

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

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

SQL 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
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

"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 than "inside-out"

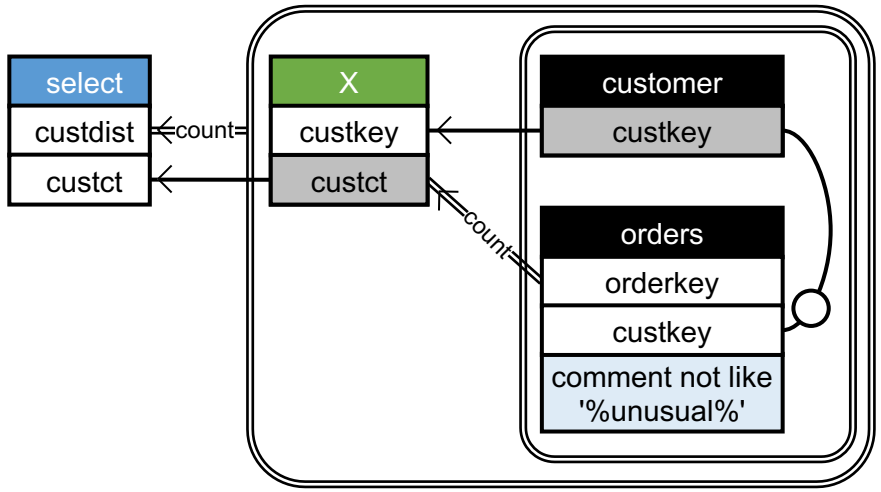
SQL Pipe Syntax

```
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

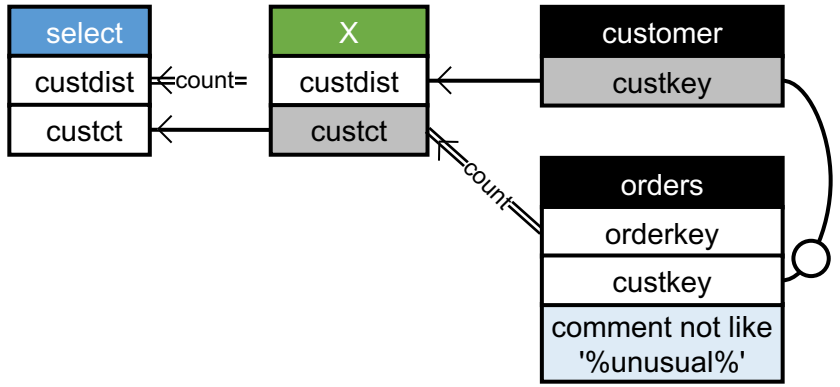
```
select custct, count(*) as custdist
from
  (select C.custkey, count(O.orderkey) as custct
  from customer C
  left outer join orders O
  on C.custkey=O.custkey
  and O.comment not like '%unusual%'
  group by C.custkey) as X
group by custct
```



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

SQL Pipe Syntax

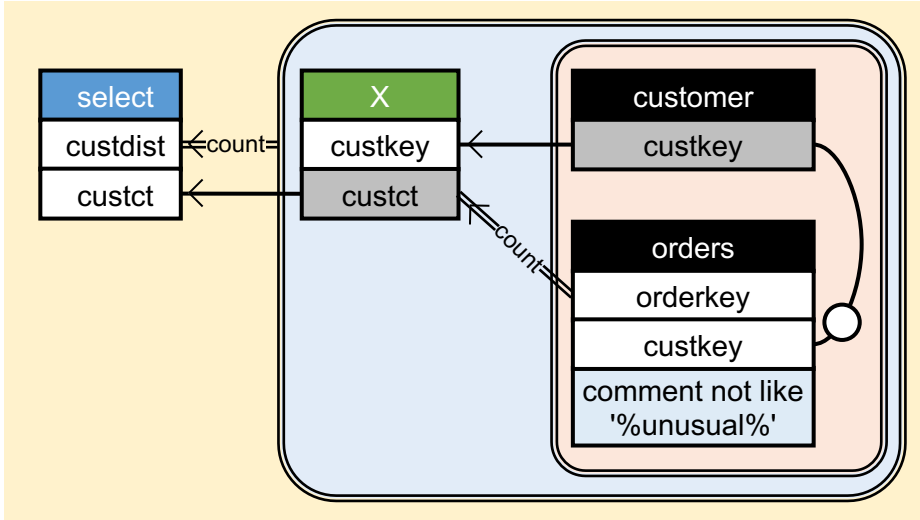
```
from customer as C
|> left outer join orders as O
   on C.custkey = O.custkey
   and O.comment not like '%unusual%'
|> aggregate count(O.orderkey) as custct
   group by C.custkey
|> aggregate count(*) as custdist
   group by custct
```



"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
```

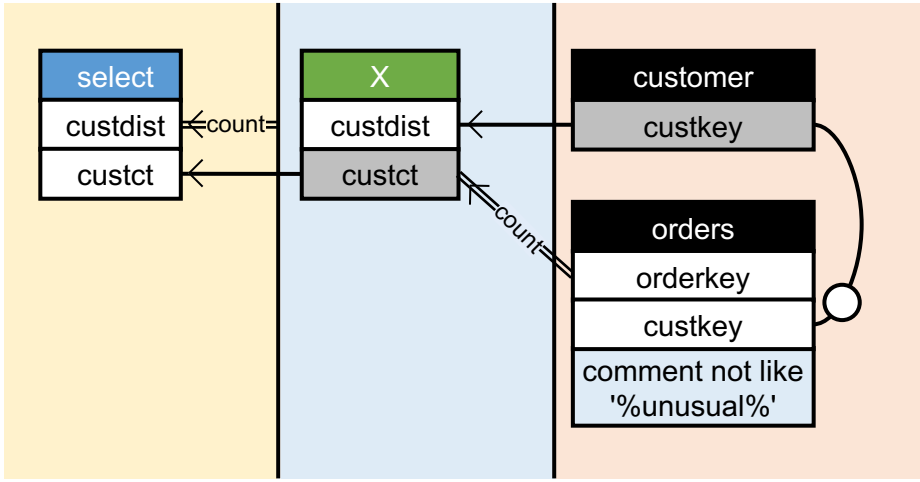


Claim: A piped linear syntax is easier to understand than "inside-out"

?

SQL Pipe Syntax

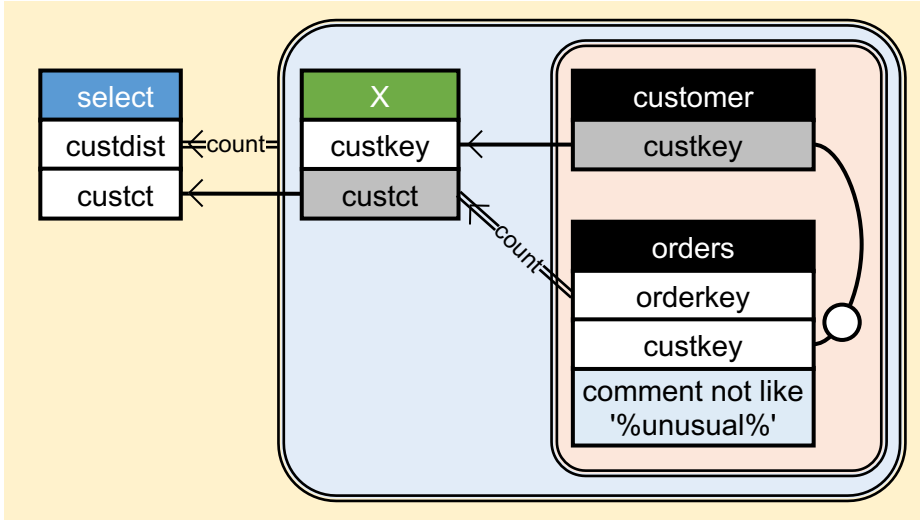
```
from customer as C
|> left outer join orders as O
   on C.custkey = O.custkey
   and O.comment not like '%unusual%'
|> aggregate count(O.orderkey) as custct
   group by C.custkey
|> aggregate count(*) as custdist
   group by custct
```



"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
```

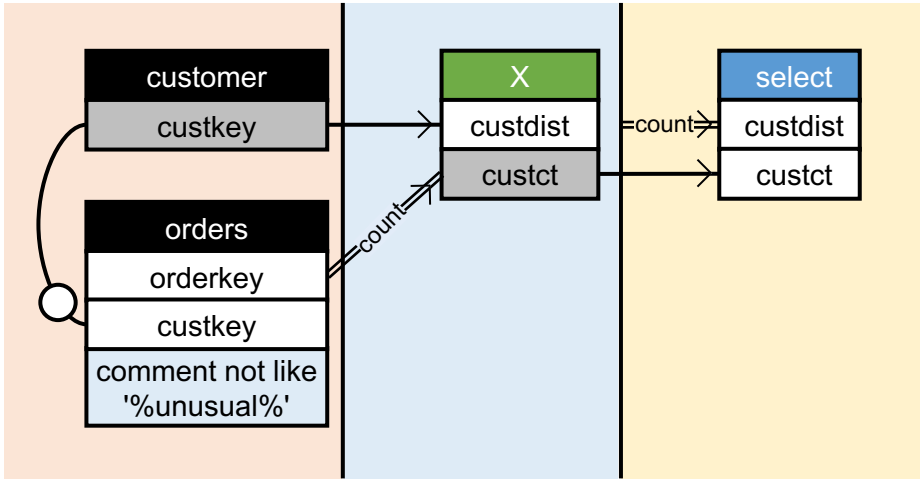


Claim: A piped linear syntax is easier to understand than "inside-out"

?

SQL Pipe Syntax

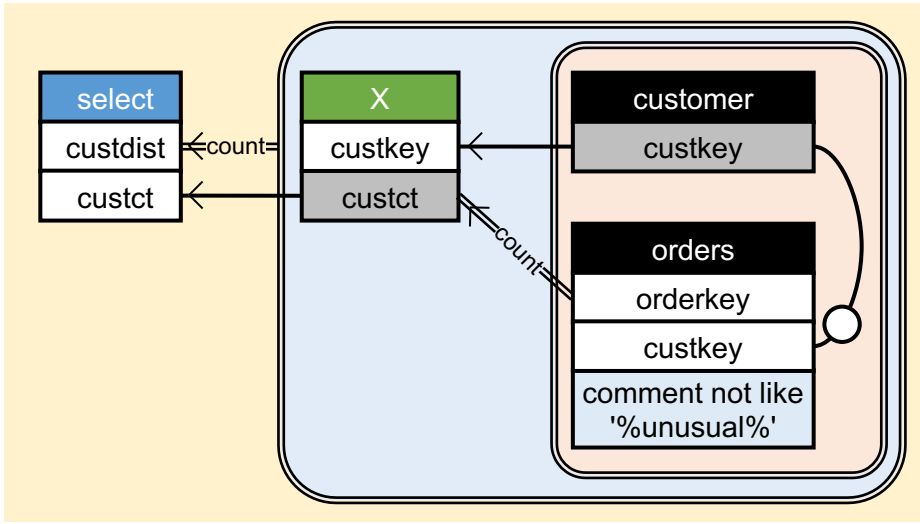
```
from customer as C
|> left outer join orders as O
   on C.custkey = O.custkey
   and O.comment not like '%unusual%'
|> aggregate count(O.orderkey) as custct
   group by C.custkey
|> aggregate count(*) as custdist
   group by custct
```



"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 than "inside-out"

SQL 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:

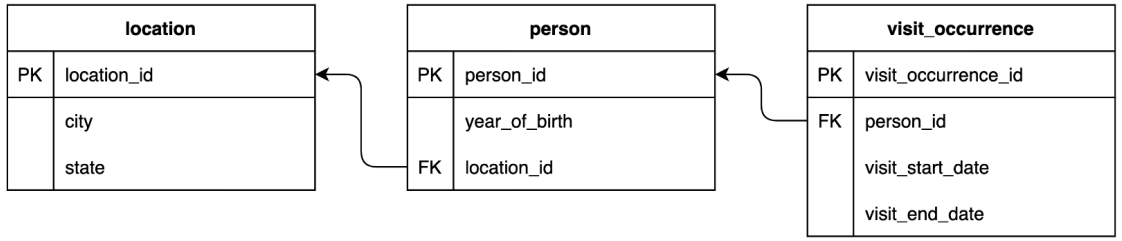
- easy-to-understand depends on "modality", not piped linear syntax vs. "inside-out"

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

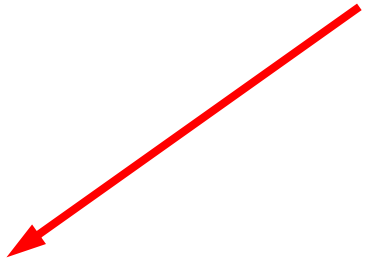
Q: When was the last time each person born between 1930 and 1940 and living in Illinois was seen by a healthcare provider?



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)

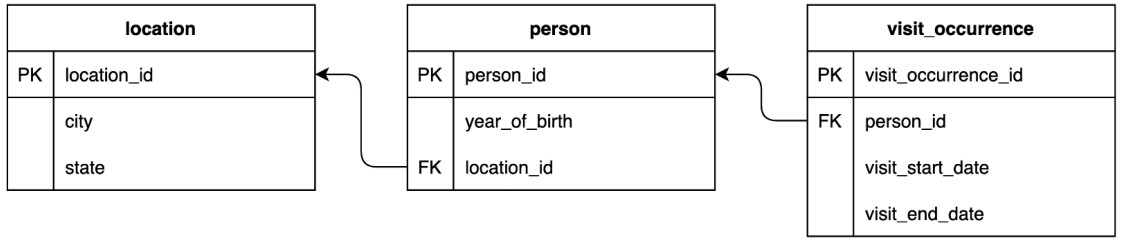
"FunSQL"



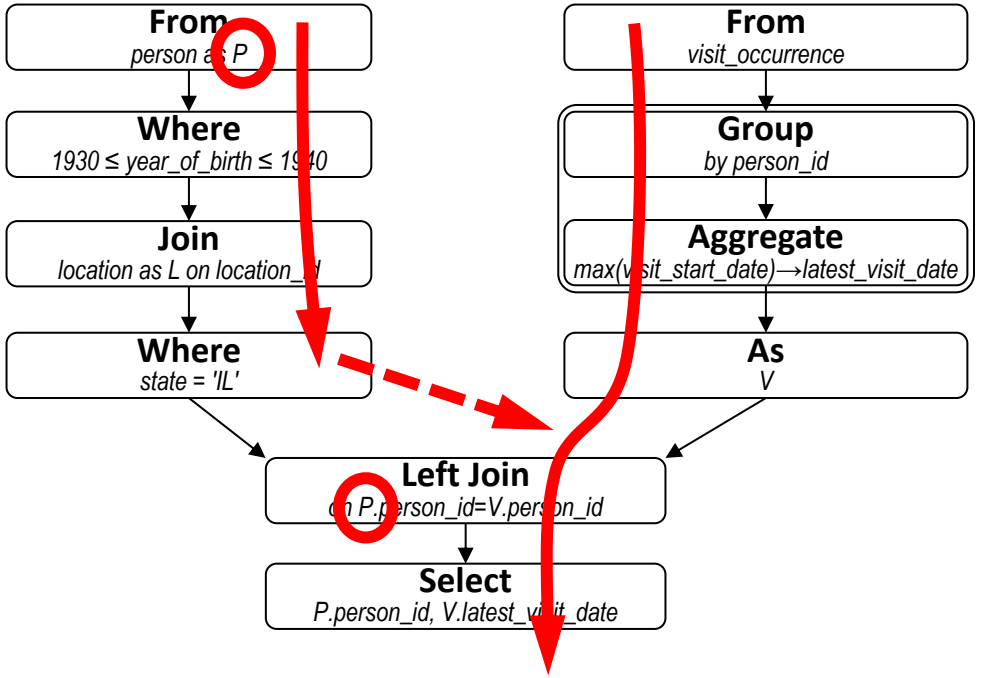
Query adapted from [FunSQL'21] (a Julia library for compositional construction of SQL queries. https://github.com/MechanicalRabbit/FunSQL.jl) and translated into SQL Pipe Syntax
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

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;
```

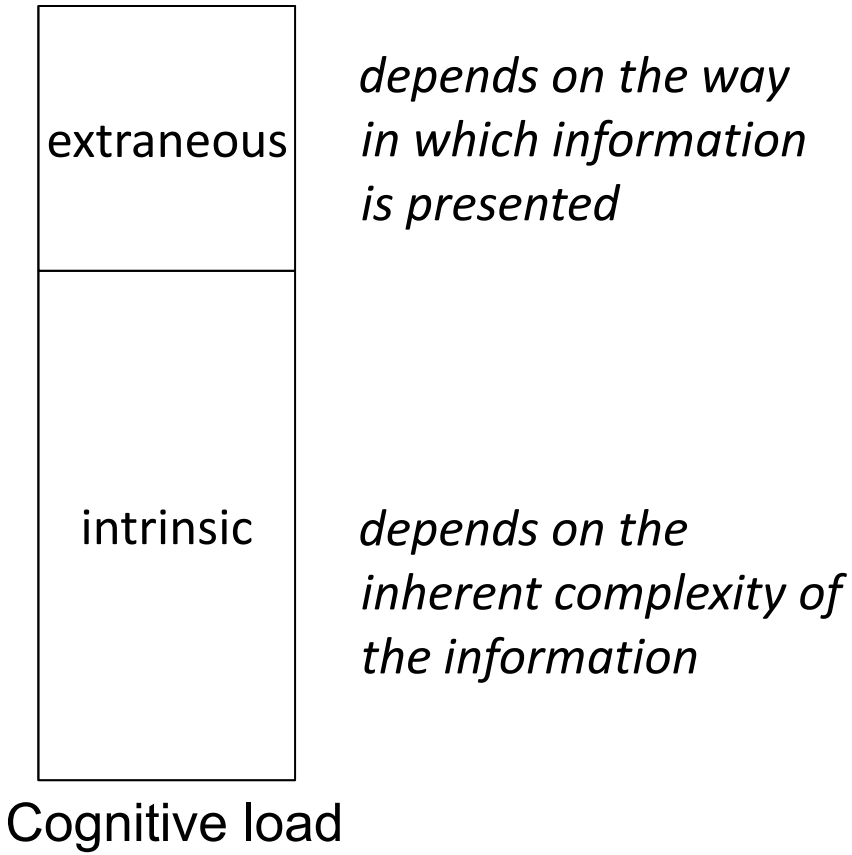


Serialization of a partial order requires jumping around via some reference (define and re-use later): this cannot be avoided!

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

Query adapted from [FunSQL'21] (a Julia library for compositional construction of SQL queries. https://github.com/MechanicalRabbit/FunSQL.jl) and translated into SQL Pipe Syntax
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Cognitive load: intrinsic vs extraneous

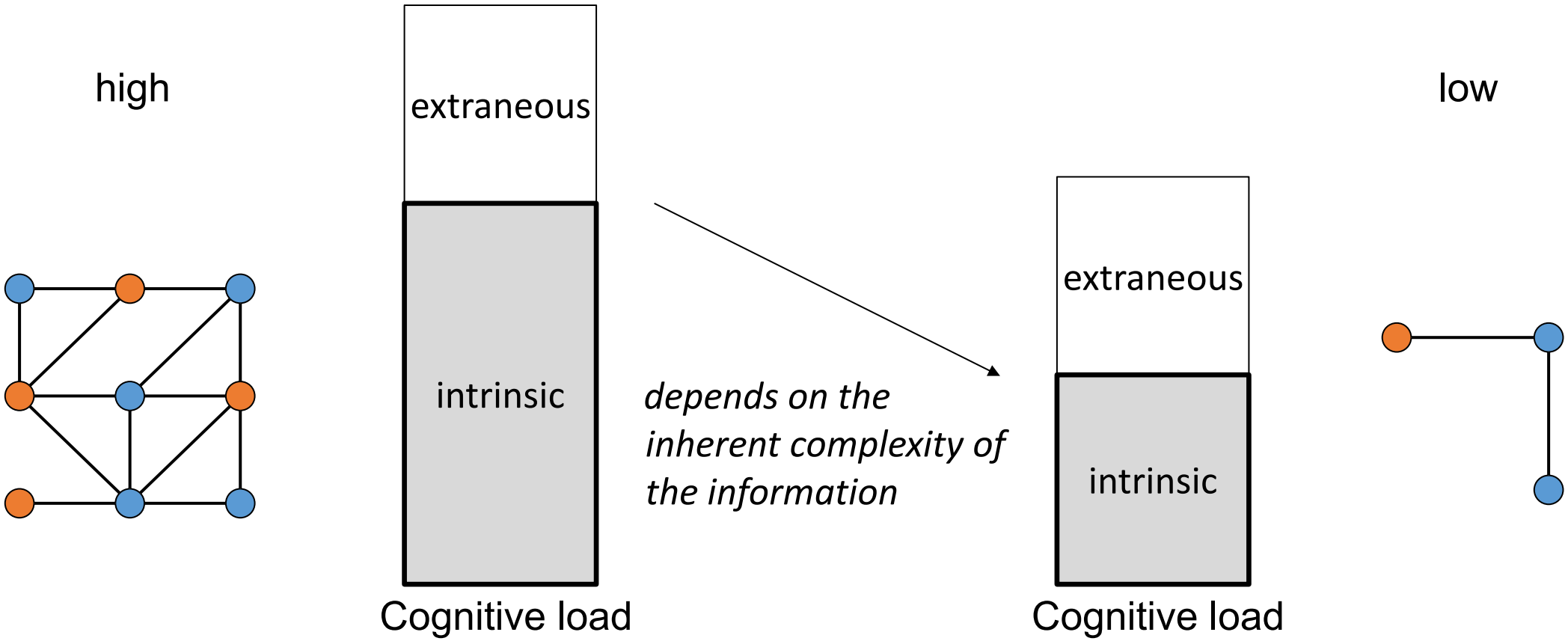


depends on the way in which information is presented

depends on the inherent complexity of the information

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. Notice a similar difference between essential and accidental complexity by Frederick Brooks ("No Silver Bullet: Essence and Accidents of Software Engineering", 1987. https://doi.org/10.1109/MC.1987.1663532), who himself quotes Aristotle.
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

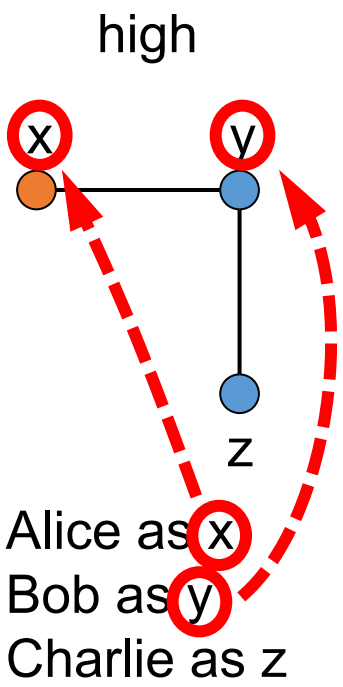
Cognitive load: intrinsic vs extraneous

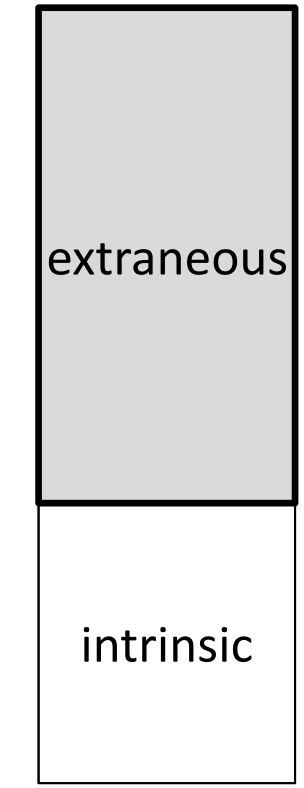


Simplified take-away from a large body of work on split-attention: "Cognitive load during problem solving: Effects on learning", Sweller, 1988. https://doi.org/10.1016/0364-0213(88)90023-7. Notice a similar difference between essential and accidental complexity by Frederick Brooks ("No Silver Bullet: Essence and Accidents of Software Engineering", 1987. https://doi.org/10.1109/MC.1987.1663532), who himself quotes Aristotle.
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

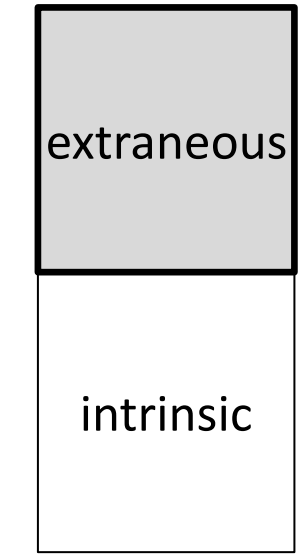
Cognitive load: intrinsic vs extraneous

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



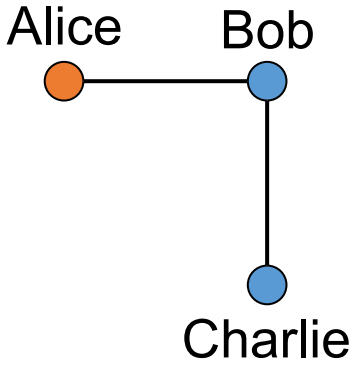


depends on the way in which information is presented



spatial contiguity: when related information is near one another

low

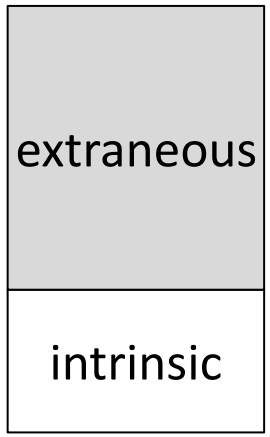


Simplified take-away from a large body of work on split-attention: "Cognitive load during problem solving: Effects on learning", Sweller, 1988. https://doi.org/10.1016/0364-0213(88)90023-7. Notice a similar difference between essential and accidental complexity by Frederick Brooks ("No Silver Bullet: Essence and Accidents of Software Engineering", 1987. https://doi.org/10.1109/MC.1987.1663532), who himself quotes Aristotle.
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Pipelines reduce split-points, diagrams remove them

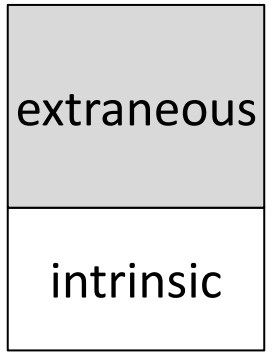
Easy-to-understand depends on "modality" first, and piped linear syntax vs. "inside-out" second

inside-out



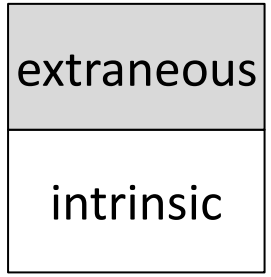
2 split points

dataflow



1 split point

inside-out



0 split points

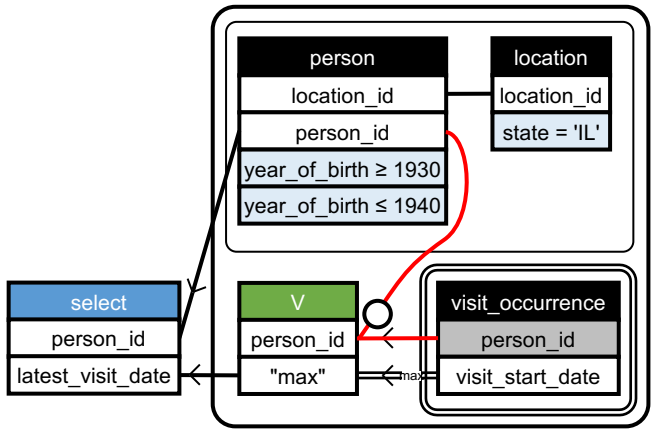
SQL (text)

```
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
```

SQL 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

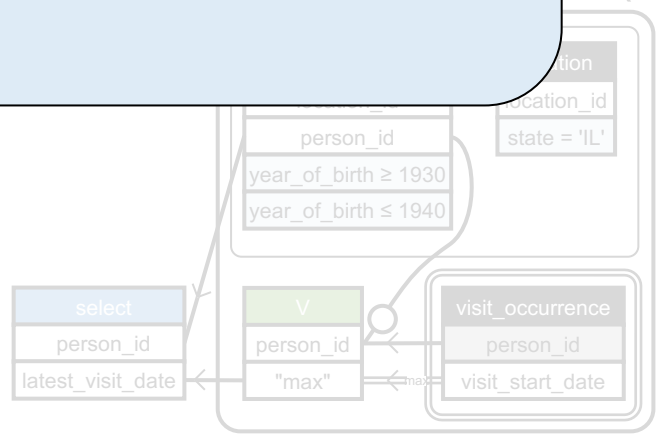
extraneous | extraneous | extraneous

- 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;
```



Correlated Sales

Pipe Syntax on Correlated Sales

Ibis

```
Q = (Empl.join(Sales.group_by("eid")
               .aggregate(sm=Sales.val.sum()),
              "eid", how="left")
      .select(eid=_.eid, sm=_.sm.fill_null(0)) )
```

SQL PipeSyntax

```
WITH T AS (
  FROM Sales AS S
  |> AGGREGATE SUM(val) AS sm
     GROUP BY eid)
FROM Empl AS E
|> LEFT JOIN T
   ON E.eid = T.eid
|> SELECT E.eid, coalesce(T.sm, 0) AS sm;
```

SaneQL

```
Empl
  .join(
    Sales
      .group({eid}, {sm := sum(val)}),
    Empl.eid = eid,
    type := leftouter )
  .project({
    eid := Empl.eid,
    sm := coalesce(sm, 0) })
```

PRQL

```
from E = Empl
join side:left ST = (
  from Sales
  group eid (aggregate { sm = sum val } )
) (E.eid == ST.eid)
select {eid = E.eid, sm = ST.sm ?? 0 }
```

Empl

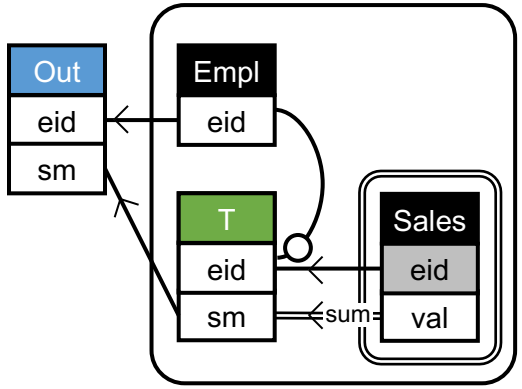
eid	k
a	1
b	2
c	3
b	4

Sales

eid	val	date
a	10	1/1/26
a	20	1/2/26
b	40	1/1/26

Out

eid	sm
a	30
b	40
b	40
c	0



SQL3

```
select E.eid, coalesce(T.sm, 0) sm
from Empl E
left join (
  select S.eid, sum(val) sm
  from Sales S
  group by S.eid ) T
on E.eid = T.eid
```

SQL database 750 available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql ,
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Outline

- What is a relational query? A vocabulary
- Various aspects of relational languages
  - Join queries
  - Lateral Joins
  - Declarativity of Languages
  - Grouping
  - Linear Composition
  - Relational Division
- Outlook

PLEASE ask questions and let me know if you spot any error anywhere ☺

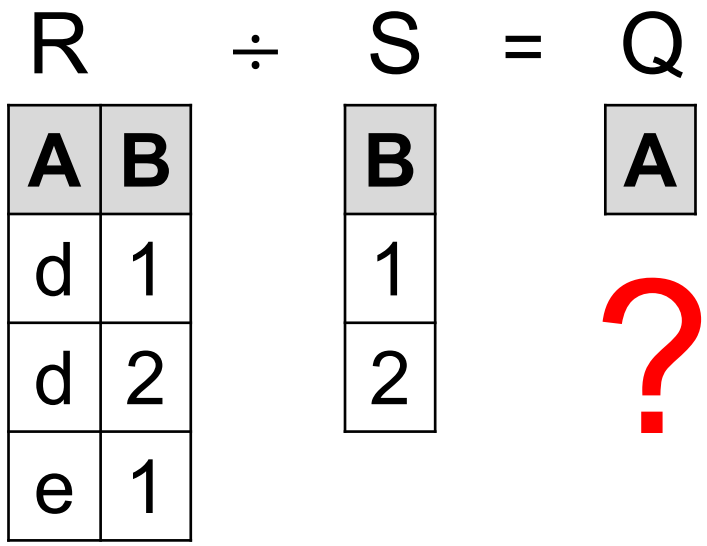
An anonymous feedback form is linked from the tutorial web page:

Relational Division

R(A,B), S(B)

Relational Division R(A,B) ÷ S(B)

"Find values from R.A that co-occur with all values from S.B"

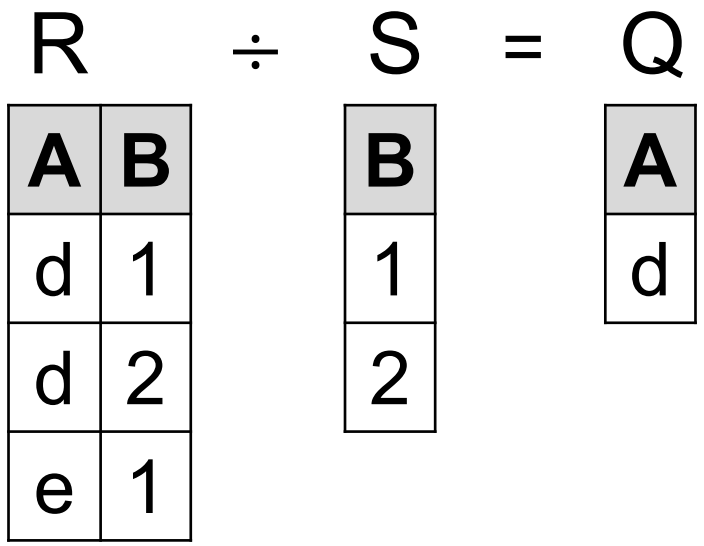


Relational Division

R(A,B), S(B)

Relational Division R(A,B) ÷ S(B)

"*Find values from R.A that co-occur with all values from S.B*"



R

A	B
d	1
d	2
e	1

÷

S

B
1
2

=

Q

A
d

Logical Expressiveness ≠ Pattern Expressiveness of QLs

Q: Find sailors who reserved all boats.

```
SELECT DISTINCT S.sid
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

LEMMA 20 (appendix): This SQL query has no pattern-equivalent representation in Relational Algebra.

Logical Expressiveness ≠ Pattern Expressiveness of QLs

Q: Find sailors who reserved all boats.

```
SELECT DISTINCT S.sid
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

LEMMA 20 (appendix): This SQL query has no pattern-equivalent representation in Relational Algebra.

Relational Algebra

$$\pi_{\text{sid}}\text{Sailor} - \pi_{\text{sid}}\,(\pi_{\text{sid,bid}}(\text{Sailor} \times \text{Boat}) - \pi_{\text{sid,bid}}\text{Reserves})$$

Relational Algebra can express *fewer* patterns

Q: Find sailors who reserved all boats.

LEMMA 20 (appendix): This SQL query has no pattern-equivalent representation in Relational Algebra.

Relational Algebra

?

```
SELECT DISTINCT S.sid
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

- logically equivalent, but
- not pattern equivalent

$$\pi_{sid}\text{Sailor} - \pi_{sid}(\pi_{sid,bid}(\text{Sailor} \times \text{Boat}) - \pi_{sid,bid}\text{Reserves})$$

Source: On the Reasonable Effectiveness of Relational Diagrams. SIGMOD 2024. https://doi.org/10.1145/3639316
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Relational Algebra can express *fewer* patterns

Q: Find sailors who reserved all boats.

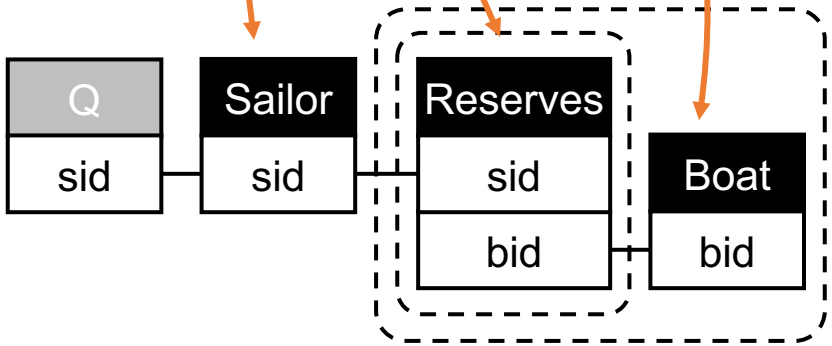
LEMMA 20 (appendix): This SQL query has no pattern-equivalent representation in Relational Algebra.

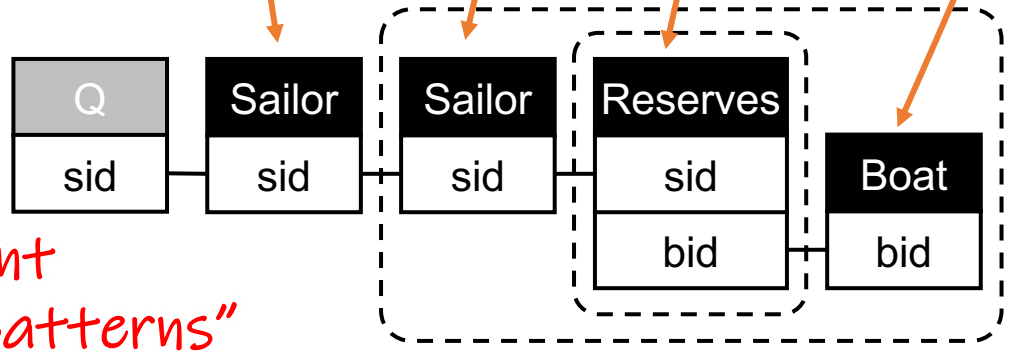
Relational Algebra

```
SELECT DISTINCT S.sid
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

- logically equivalent, but
- not pattern equivalent

$$\pi_{sid}\text{Sailor} - \pi_{sid}(\pi_{sid,bid}(\text{Sailor} \times \text{Boat}) - \pi_{sid,bid}\text{Reserves})$$





Two different "relational patterns"

Source: On the Reasonable Effectiveness of Relational Diagrams. SIGMOD 2024. https://doi.org/10.1145/3639316
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Relational Algebra can express *fewer* patterns

Q: Find sailors who reserved all boats.

LEMMA 20 (appendix): This SQL query has no pattern-equivalent representation in Relational Algebra.

- logically equivalent, but
- not pattern equivalent

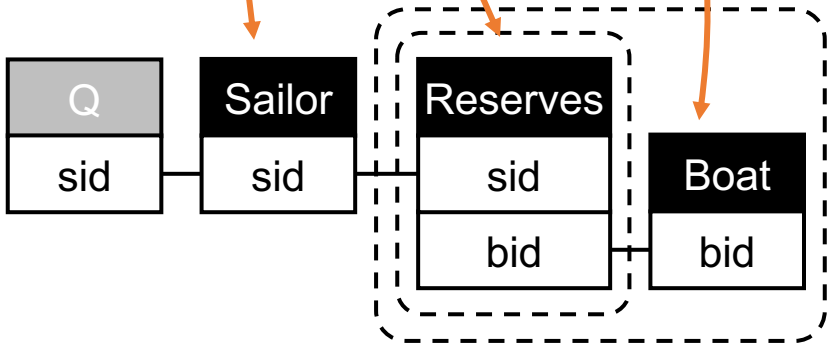
INSIGHT: Visual representations based on Relational Algebra operators cannot capture all patterns of relational queries

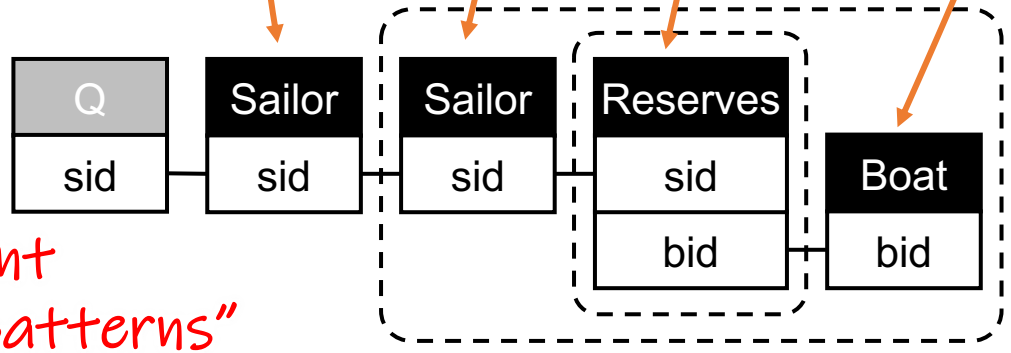
Two different "relational patterns"

```
SELECT DISTINCT S.sid
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

Relational Algebra

$$\pi_{sid}\text{Sailor} - \pi_{sid}(\pi_{sid,bid}(\text{Sailor} \times \text{Boat}) - \pi_{sid,bid}\text{Reserves})$$





Source: On the Reasonable Effectiveness of Relational Diagrams. SIGMOD 2024. https://doi.org/10.1145/3639316
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Relational Algebra can express *fewer* patterns

Q: Find sailors who reserved all boats.

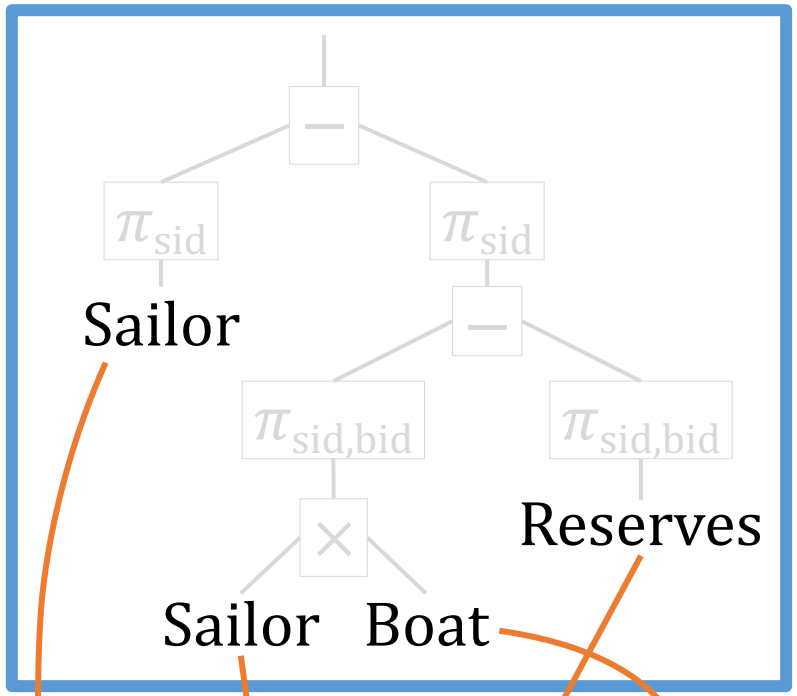
LEMMA 20 (appendix): This SQL query has no pattern-equivalent representation in Relational Algebra.

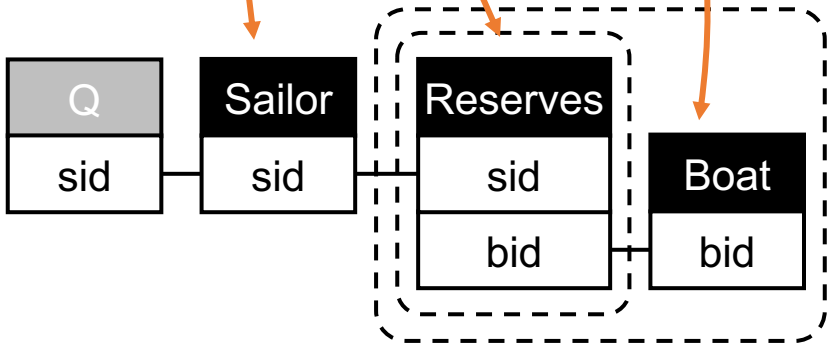
- logically equivalent, but
- not pattern equivalent

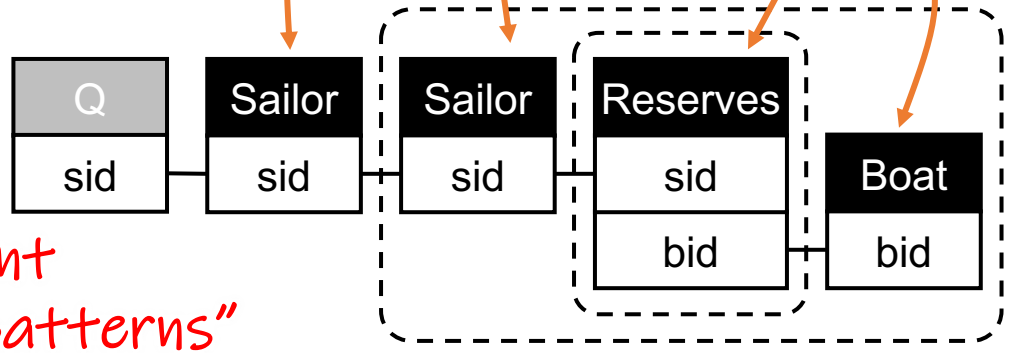
INSIGHT: Visual representations based on Relational Algebra operators cannot capture all patterns of relational queries

```
SELECT DISTINCT S.sid
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

Abstract syntax tree of RA







Two different "relational patterns"

Source: On the Reasonable Effectiveness of Relational Diagrams. SIGMOD 2024. https://doi.org/10.1145/3639316
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Visual representations of RA operators is *not* enough

Q: Find sailors who reserved all boats.

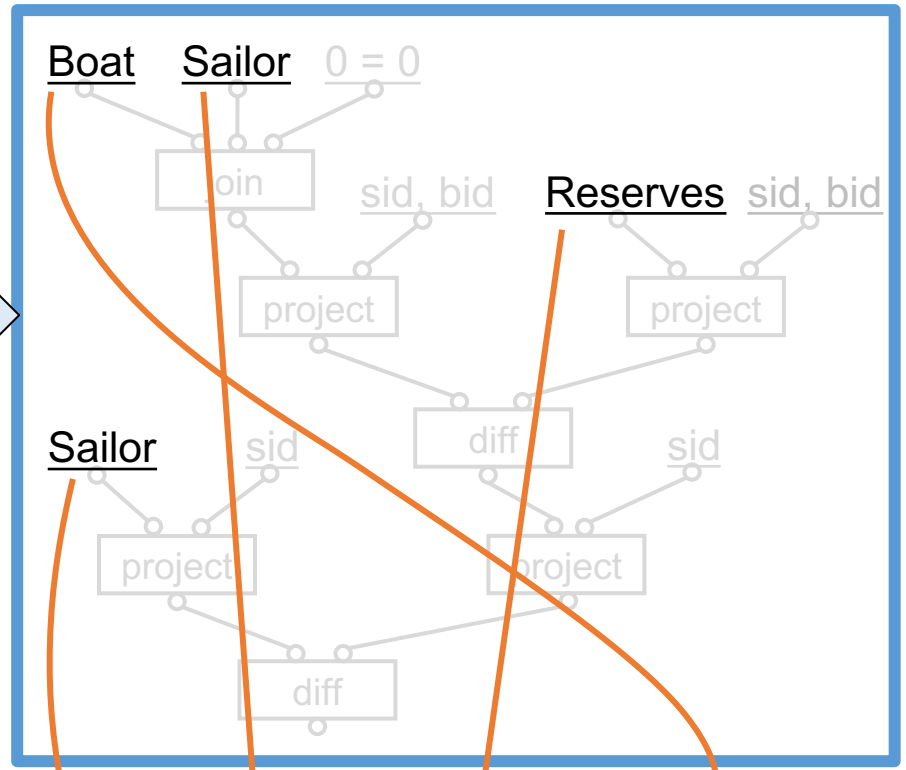
```
SELECT DISTINCT S.sid
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

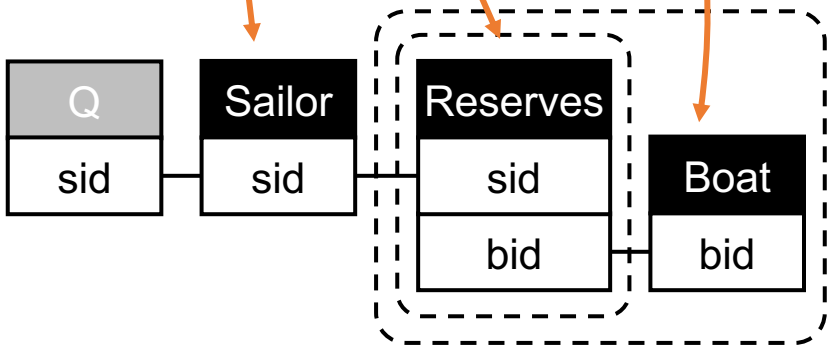
LEMMA 20 (appendix): This SQL query has no pattern-equivalent representation in Relational Algebra.

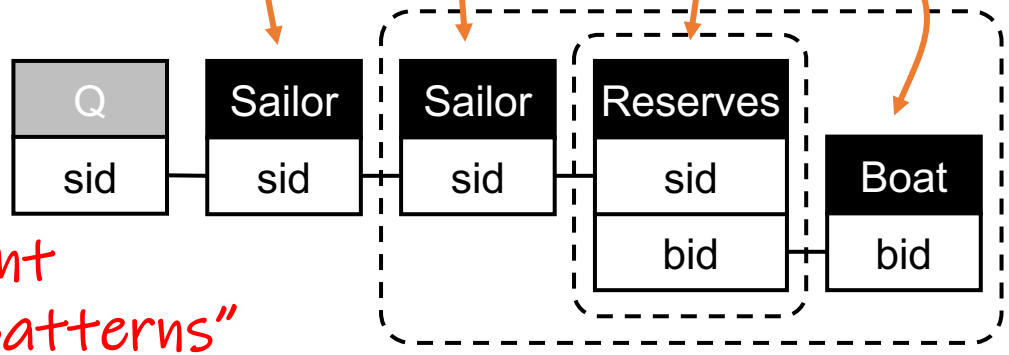
- logically equivalent, but
- not pattern equivalent

INSIGHT: Visual representations based on Relational Algebra operators cannot capture all patterns of relational queries

Example: Dataflow QL ['94]







Two different "relational patterns"

Source: On the Reasonable Effectiveness of Relational Diagrams. SIGMOD 2024. https://doi.org/10.1145/3639316
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Logically equivalent* queries for relational division

R(A,B)÷S(B)

SQL

```
SELECT DISTINCT R.A
FROM R
WHERE R.A not in
  (SELECT R3.A
  FROM R AS R3, S
  WHERE (R3.A, S.B) not in
    (SELECT R2.A, R2.B
    FROM R AS R2))
```

```
SELECT DISTINCT R.A
FROM R
WHERE not exists
  (SELECT *
  FROM R AS R3, S
  WHERE R.A = R3.A
  AND not exists
    (SELECT *
    FROM R AS R2
    WHERE R2.B=S.B
    AND R2.A=R3.A))
```

```
SELECT DISTINCT R.A
FROM R
WHERE not exists
  (SELECT *
  FROM S
  WHERE not exists
    (SELECT *
    FROM R AS R2
    WHERE R2.B=S.B
    AND R2.A=R.A))
```

Datalog with negation

$$I(x) :- R(x, \_), S(y), \neg R(x, y)$$
$$Q(x) :- R(x, \_), \neg I(x)$$

RA: $\pi_A R - \pi_A((\pi_A R \times S) - R)$

TRC

$$\{q(A) \mid \exists r \in R[r.A=q.A \wedge \neg(\exists r_3 \in R, \exists s \in S[r_3.A=r.A \wedge \neg(\exists r_2 \in R[r_2.B=s.B \wedge r_2.A=r_3.A])])]\}$$

$$\{q(A) \mid \exists r \in R[r.A=q.A \wedge \neg(\exists s \in S[\neg(\exists r_2 \in R[r_2.B=s.B \wedge r_2.A=r.A])])]\}$$

Do these 7 logically equivalent queries use the same "pattern" ?

* assuming NO NULL value constraints

Logically equivalent*, but not pattern-equivalent

R(A,B)÷S(B)

```
SELECT DISTINCT R.A
FROM R
WHERE R.A not in
  (SELECT R3.A
  FROM R AS R3, S
  WHERE (R3.A, S.B) not in
    (SELECT R2.A, R2.B
    FROM R AS R2))
```

```
SELECT DISTINCT R.A
FROM R
WHERE not exists
  (SELECT *
  FROM R AS R3, S
  WHERE R.A = R3.A
  AND not exists
    (SELECT *
    FROM R AS R2
    WHERE R2.B=S.B
    AND R2.A=R3.A))
```

```
SELECT DISTINCT R.A
FROM R
WHERE not exists
  (SELECT *
  FROM S
  WHERE not exists
    (SELECT *
    FROM R AS R2
    WHERE R2.B=S.B
    AND R2.A=R.A))
```

SQL

Datalog with negation

$$I(x) :- R(x, \_), S(y), \neg R(x, y)$$
$$Q(x) :- R(x, \_), \neg I(x)$$

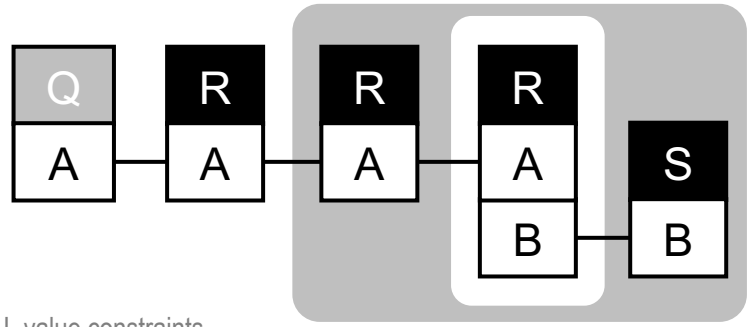
RA: $\pi_A R - \pi_A((\pi_A R \times S) - R)$

1. Two different "relational patterns"

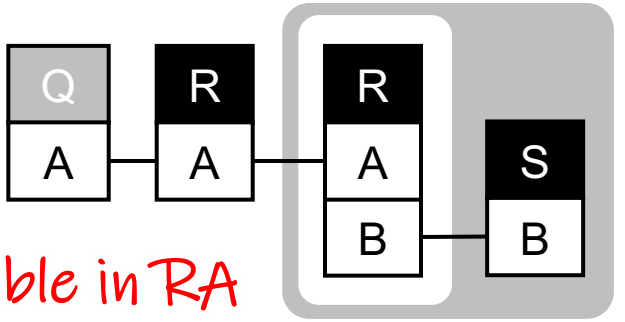
TRC

$$\{q(A) \mid \exists r \in R[r.A=q.A \wedge \neg(\exists r_3 \in R, \exists s \in S[r_3.A=r.A \wedge \neg(\exists r_2 \in R[r_2.B=s.B \wedge r_2.A=r_3.A])])]\}$$

$$\{q(A) \mid \exists r \in R[r.A=q.A \wedge \neg(\exists s \in S[\neg(\exists r_2 \in R[r_2.B=s.B \wedge r_2.A=r.A])])]\}$$



2. Relational Diagrams



3. not possible in RA

* assuming NO NULL value constraints

Intuition: Focus on the extensional tables only

R(A,B)÷S(B)

```
SELECT DISTINCT R.A
FROM R
WHERE R.A not in
  (SELECT R3.A
  FROM R AS R3, S
  WHERE (R3.A, S.B) not in
    (SELECT R2.A, R2.B
    FROM R AS R2))
```

```
SELECT DISTINCT R.A
FROM R
WHERE not exists
  (SELECT *
  FROM R AS R3, S
  WHERE R.A = R3.A
  AND not exists
    (SELECT *
    FROM R AS R2
    WHERE R2.B=S.B
    AND R2.A=R3.A))
```

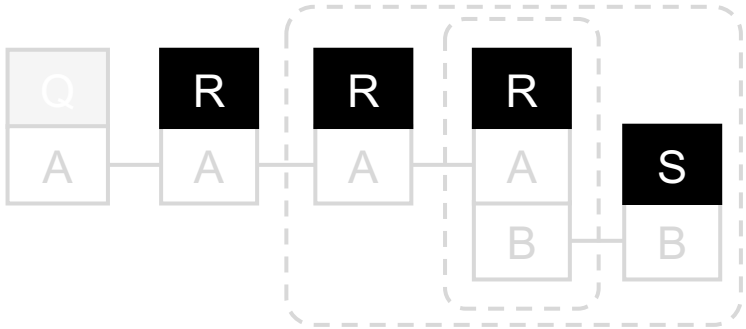
```
SELECT DISTINCT R.A
FROM R
WHERE not exists
  (SELECT *
  FROM S
  WHERE not exists
    (SELECT *
    FROM R AS R2
    WHERE R2.B=S.B
    AND R2.A=R.A))
```

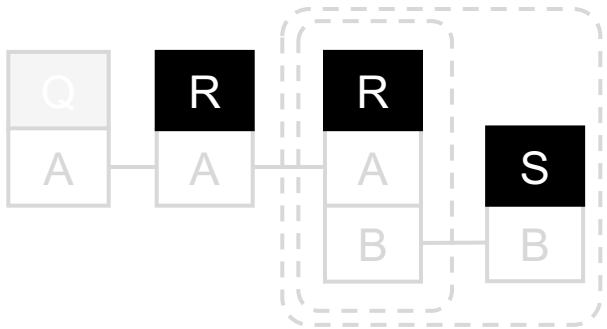
$$I(x) :- R(x, \_), S(y), \neg R(x, y)$$
$$Q(x) :- R(x, \_), \neg I(x)$$

$$\pi_A R - \pi_A((\pi_A R \times S) - R)$$

$$\{q(A) \mid \exists r \in R[r.A = q.A \wedge \neg(\exists r_3 \in R, \exists s \in S[r_3.A = r.A \wedge \neg(\exists r_2 \in R[r_2.B = s.B \wedge r_2.A = r_3.A])])]\}$$

$$\{q(A) \mid \exists r \in R[r.A = q.A \wedge \neg(\exists s \in S[\neg(\exists r_2 \in R[r_2.B = s.B \wedge r_2.A = r.A])])]\}$$





Intuition: Focus on the extensional tables only

R(A,B)÷S(B)

```
SELECT DISTINCT R.A
FROM R
WHERE R.A not in
  (SELECT R3.A
  FROM R AS R3, S
  WHERE (R3.A, S.B) not in
    (SELECT R2.A, R2.B
    FROM R AS R2))
```

```
SELECT DISTINCT R.A
FROM R
WHERE not exists
  (SELECT *
  FROM R AS R3, S
  WHERE R.A = R3.A
  AND not exists
    (SELECT *
    FROM R AS R2
    WHERE R2.B=S.B
    AND R2.A=R3.A))
```

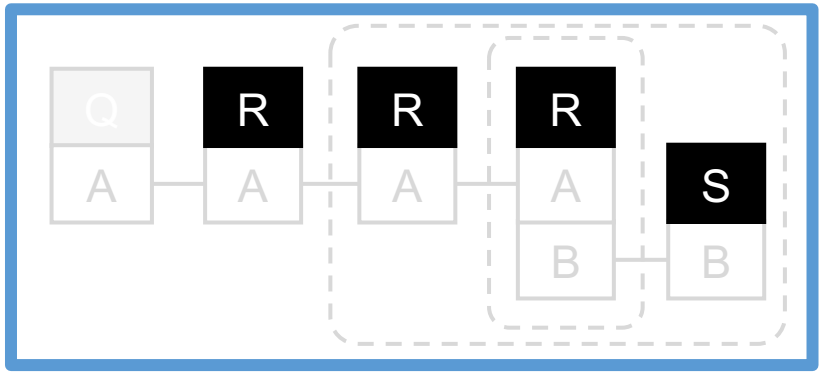
```
SELECT DISTINCT R.A
FROM R
WHERE not exists
  (SELECT *
  FROM S
  WHERE not exists
    (SELECT *
    FROM R AS R2
    WHERE R2.B=S.B
    AND R2.A=R.A))
```

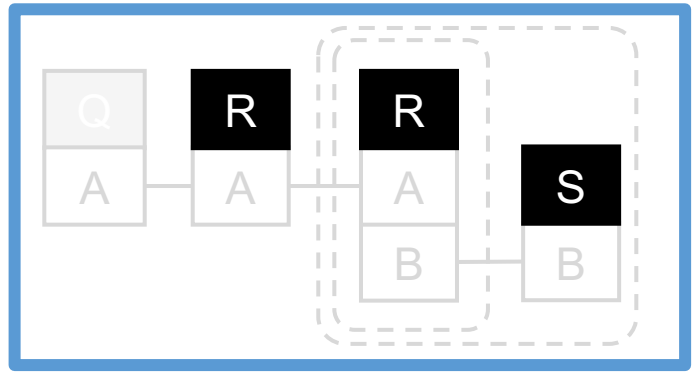
$$I(x) :- R(x, \_), S(y), \neg R(x, y)$$
$$Q(x) :- R(x, \_), \neg I(x)$$

$$\pi_A R - \pi_A((\pi_A R \times S) - R)$$

$$\{q(A) \mid \exists r \in R[r.A=q.A \wedge \neg(\exists r_3 \in R, \exists s \in S[r_3.A=r.A \wedge \neg(\exists r_2 \in R[r_2.B=s.B \wedge r_2.A=r_3.A])])]\}$$

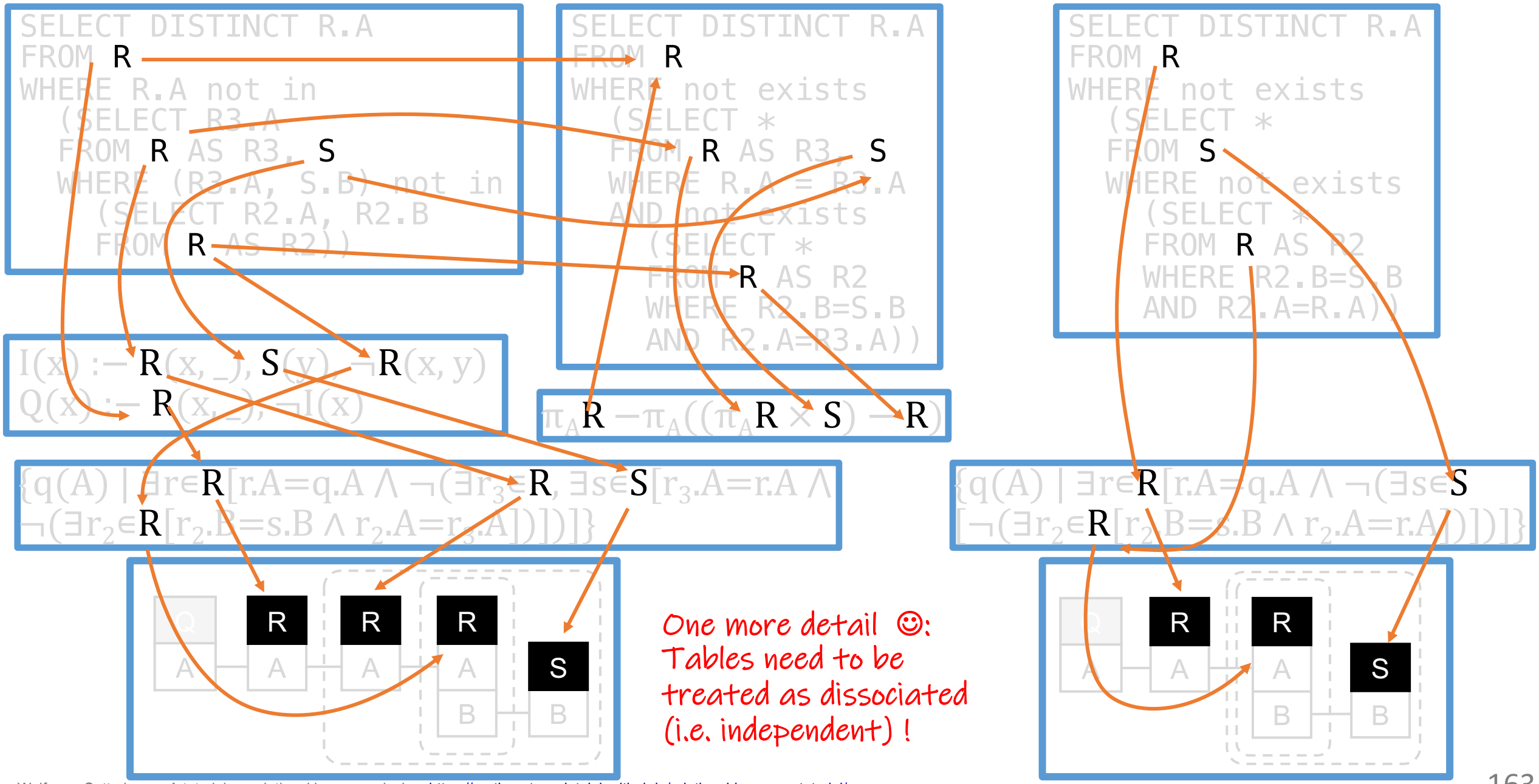
$$\{q(A) \mid \exists r \in R[r.A=q.A \wedge \neg(\exists s \in S[\neg(\exists r_2 \in R[r_2.B=s.B \wedge r_2.A=r.A])])]\}$$





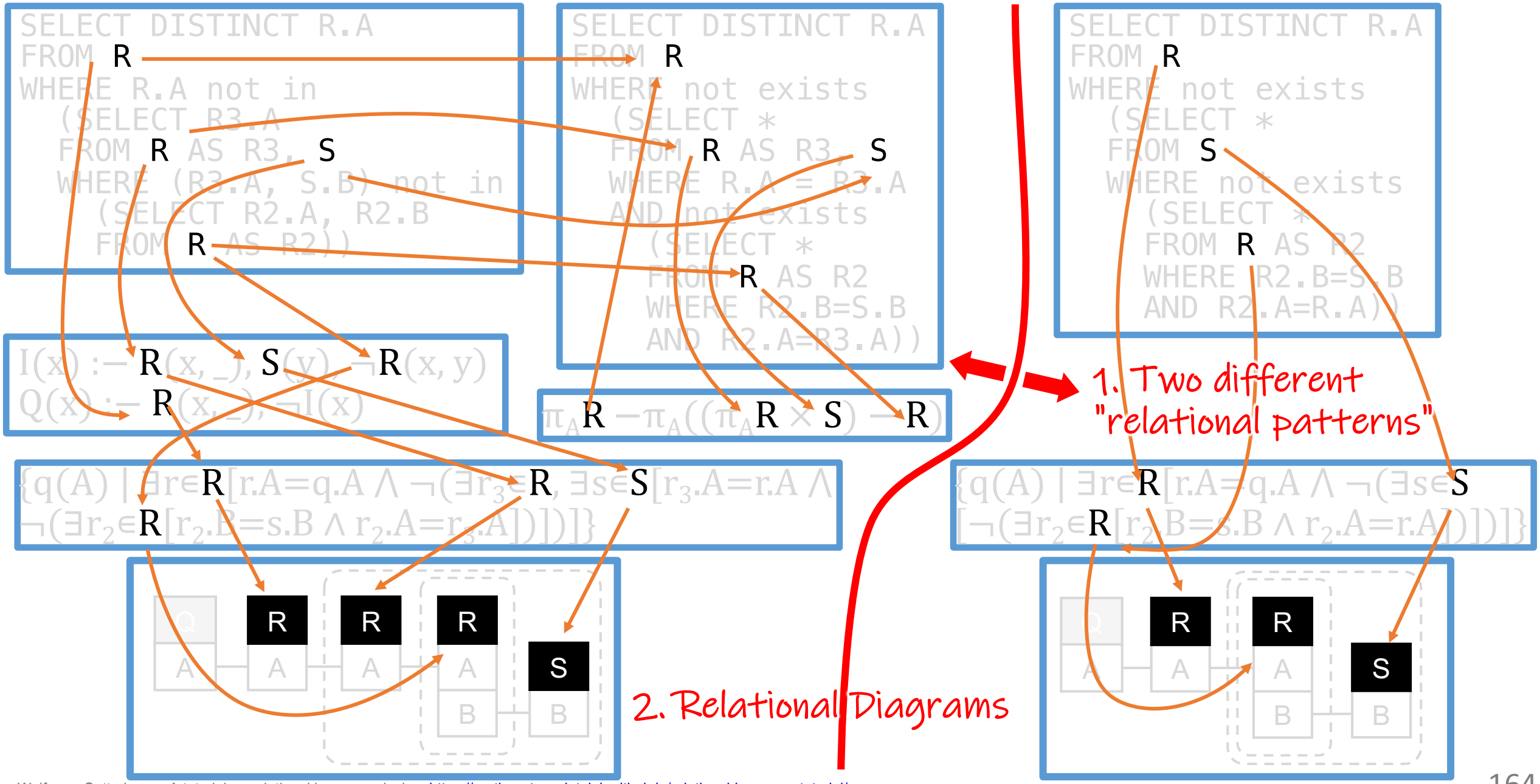
Intuition: Find mappings that preserve logical equivalence

R(A,B)÷S(B)



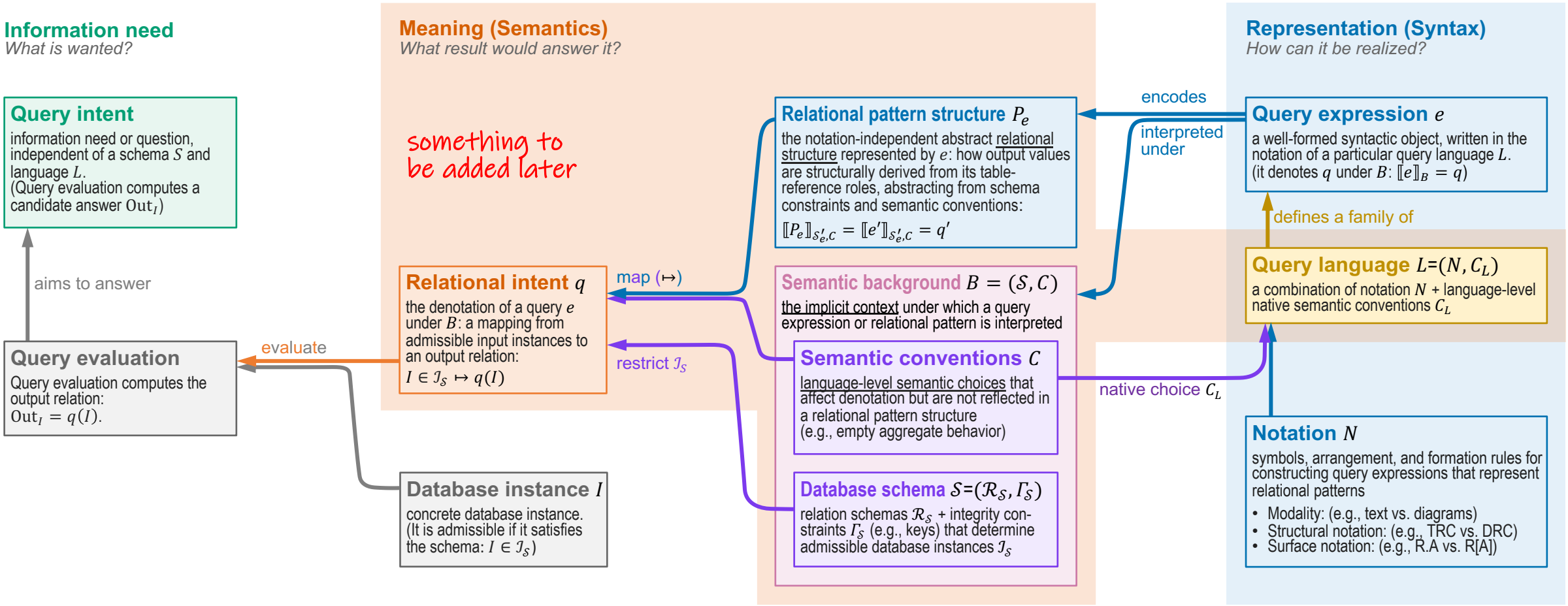
One more detail ☺:
Tables need to be treated as dissociated (i.e. independent) !

Mappings that preserve logical equivalence after dissociation R(A,B)÷S(B)

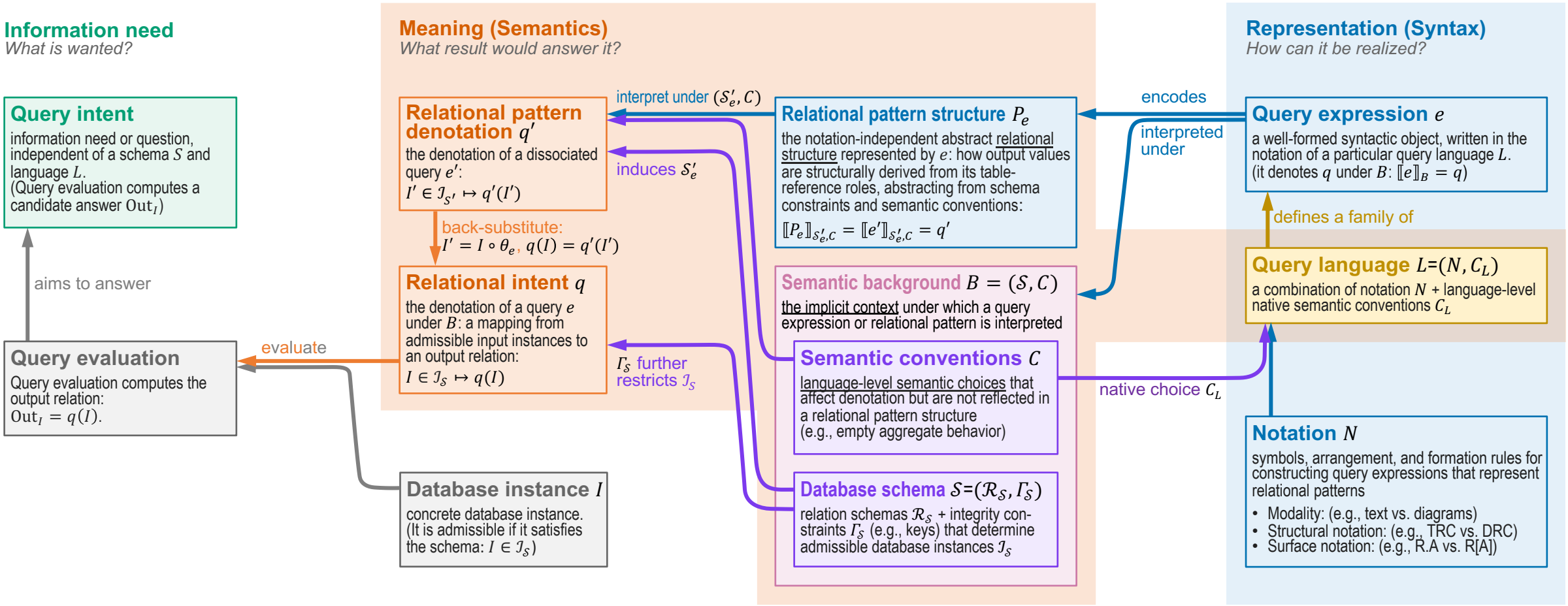


Vocabulary for Relational Language Design

REPEATED SLIDE



Vocabulary for Relational Language Design



Categorical propositions

Also see: https://en.wikipedia.org/wiki/Categorical_proposition
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

4 categorical propositions

S... subject
P... predicate

All S are P

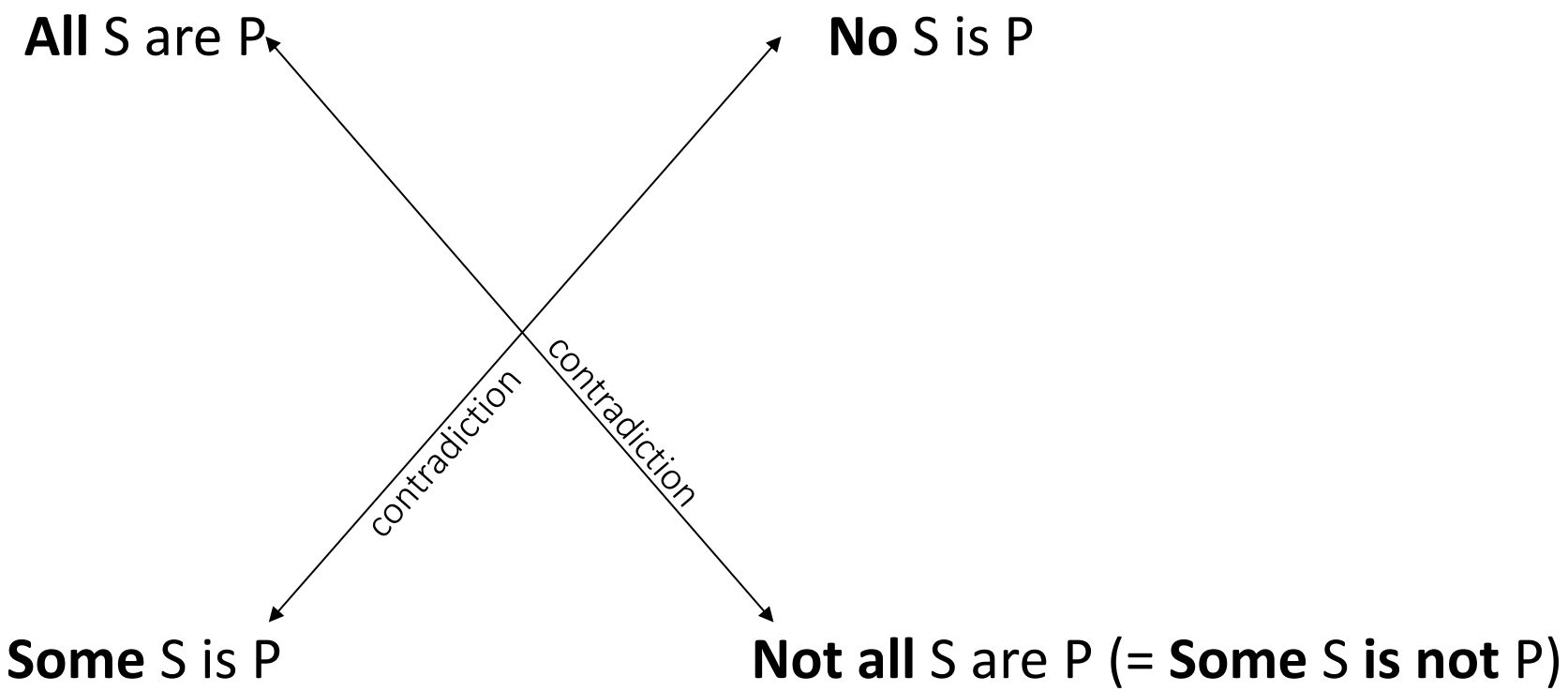
No S is P

Some S is P

Not all S are P (= **Some** S **is not** P)

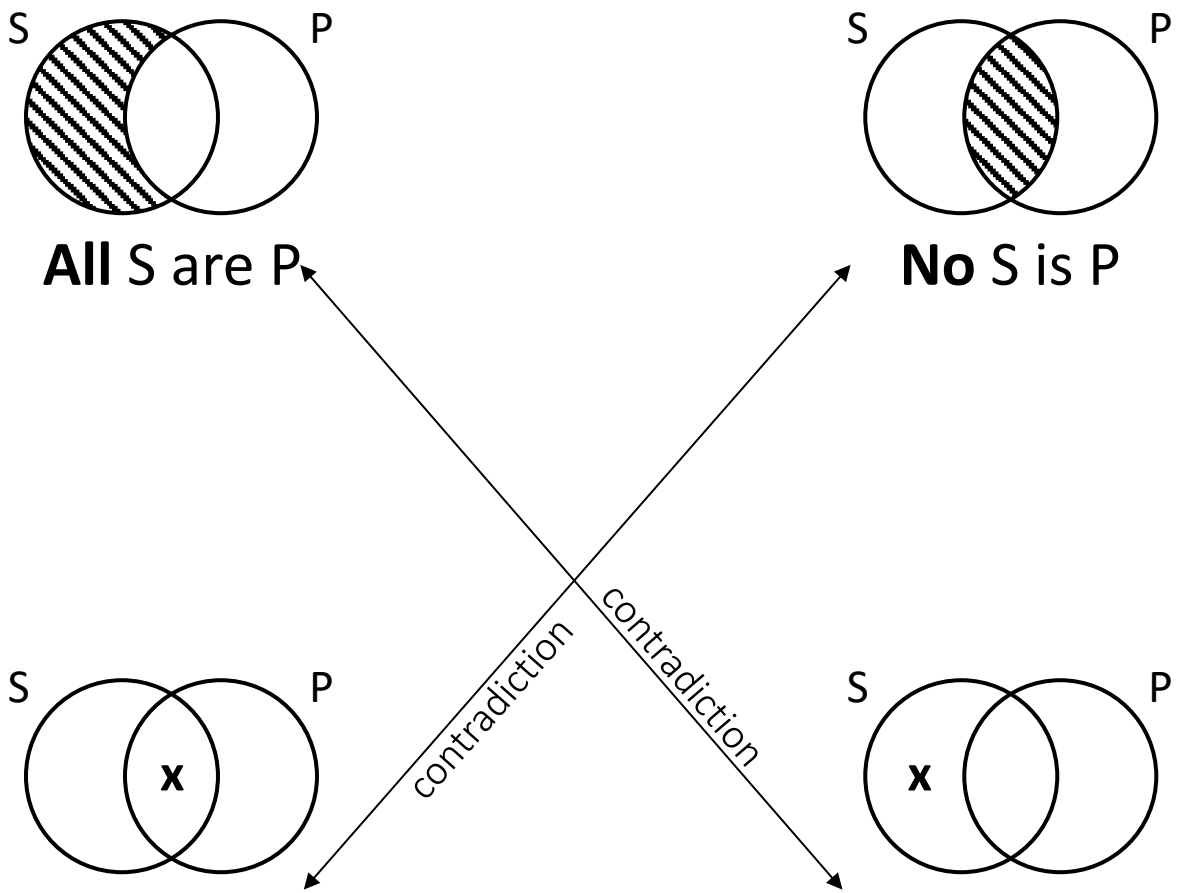
4 categorical propositions / square of opposition

S... subject
P... predicate



4 categorical propositions / square of opposition

S... subject
P... predicate



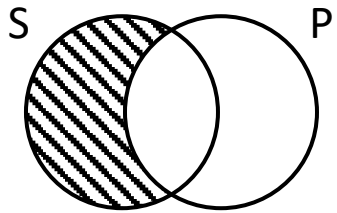
shaded areas are empty

"x" shows that something exists

Notation follows "Venn-I" diagrams by Shin (1994), itself a variant of the extension by Peirce (~1896) of Venn diagrams (1880)

4 categorical propositions / square of opposition

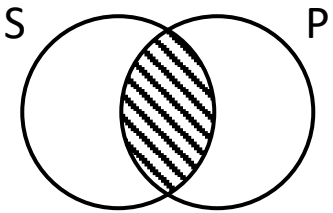
S... subject
P... predicate



All S are P

$\forall x[S(x) \Rightarrow P(x)]$

$\neg\exists x[S(x) \wedge \neg P(x)]$



No S is P

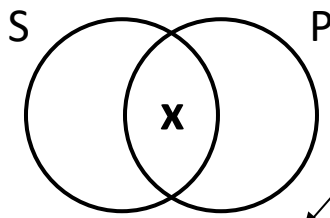
$\forall x[S(x) \Rightarrow \neg P(x)]$

$\neg\exists x[S(x) \wedge P(x)]$

shaded areas are empty

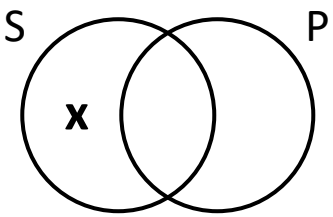
contradiction

contradiction



Some S is P

$\exists x[S(x) \wedge P(x)]$



Not all S are P (= **Some** S **is not** P)

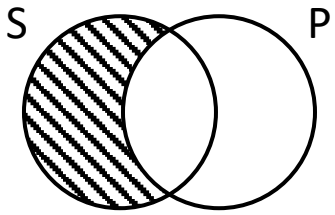
$\exists x[S(x) \wedge \neg P(x)]$

"x" shows that something exists

4 categorical propositions / square of opposition

S... subject
P... predicate

Universal propositions:

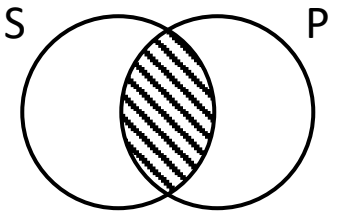


All S are P

$\forall x[S(x) \Rightarrow P(x)]$

$\neg\exists x[S(x) \wedge \neg P(x)]$

A ("Affirmo" = I affirm)



No S is P

$\forall x[S(x) \Rightarrow \neg P(x)]$

$\neg\exists x[S(x) \wedge P(x)]$

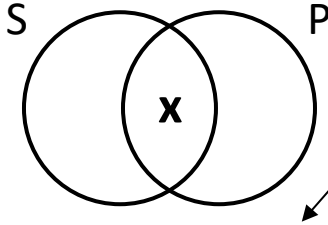
E ("nEgo" = I deny)

shaded areas are empty

contradiction

contradiction

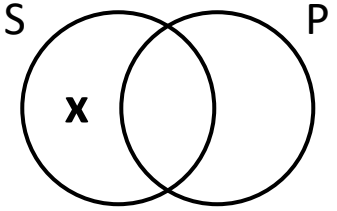
Particular propositions



Some S is P

$\exists x[S(x) \wedge P(x)]$

I ("affIrmo" = I affirm)



Not all S are P (= **Some** S **is not** P)

$\exists x[S(x) \wedge \neg P(x)]$

O ("negO" = I deny)

"x" shows that something exists

For more details see: https://en.wikipedia.org/wiki/Square_of_opposition

4 categorical propositions with Sailors

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

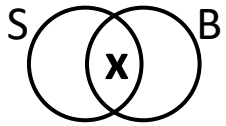
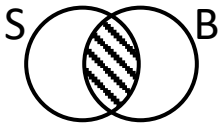
NL	Sailors who reserved **some** red boat	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

4 categorical propositions with Sailors

Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

340

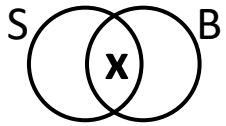
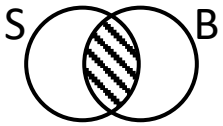
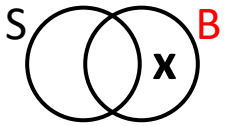
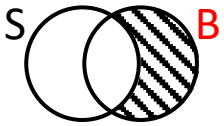
Cat.	 Some S is B. $\exists x[S(x) \wedge B(x)]$	 No S is B. (All S are not B) $\neg\exists x[S(x) \wedge B(x)]$		
NL	Sailors who reserved **some** red boat	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

4 categorical propositions with Sailors

Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

340

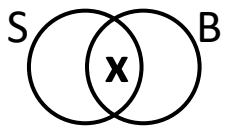
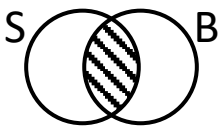
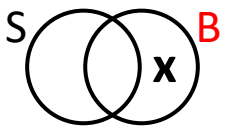
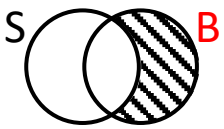
Cat.	 Some S is B. $\exists x[S(x) \wedge B(x)]$	 No S is B. (All S are not B) $\neg\exists x[S(x) \wedge B(x)]$	 Some B is not S. $\exists x[B(x) \wedge \neg S(x)]$	 All B are S. $\neg\exists x[B(x) \wedge \neg S(x)]$
NL	Sailors who reserved **some** red boat	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

4 categorical propositions with Sailors

Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

340

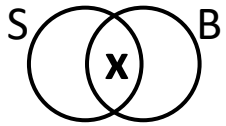
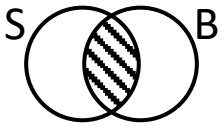
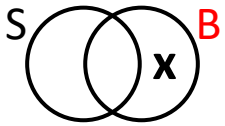
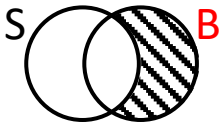
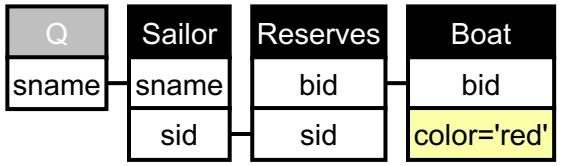
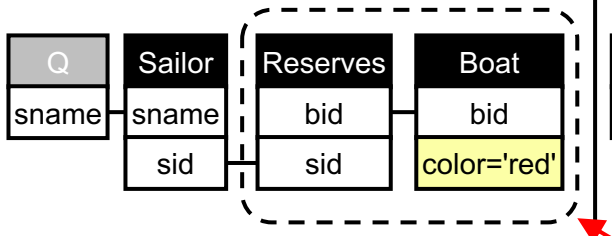
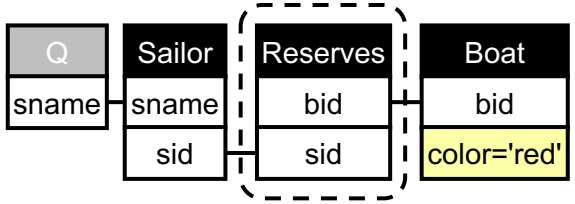
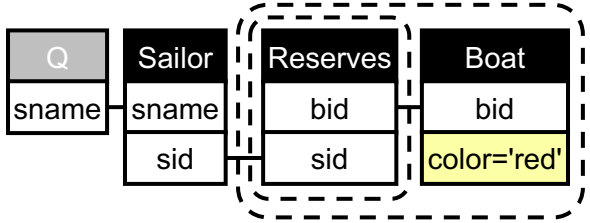
Cat.	 Some S is B. $\exists x[S(x) \wedge B(x)]$	 No S is B. (All S are not B) $\neg\exists x[S(x) \wedge B(x)]$	 Some B is not S. $\exists x[B(x) \wedge \neg S(x)]$	 All B are S. $\neg\exists x[B(x) \wedge \neg S(x)]$
NL	Sailors who reserved **some** red boat	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats

SQL	`SELECT DISTINCT S.sname`
	`FROM Sailor S`
	`WHERE EXISTS(`
	`  SELECT *`
	`  FROM Reserves R`
	`  WHERE R.sid = S.sid`
	`  AND EXISTS(`
	`  SELECT *`
	`  FROM Boat B`
	`  WHERE B.color = 'red'`
	`  AND B.bid = R.bid))`
		`SELECT DISTINCT S.sname`
		`FROM Sailor S`
		`WHERE NOT EXISTS(`
		`  SELECT *`
		`  FROM Reserves R`
		`  WHERE R.sid = S.sid`
		`  AND EXISTS(`
		`  SELECT *`
		`  FROM Boat B`
		`  WHERE B.color = 'red'`
		`  AND B.bid = R.bid))`
			`SELECT DISTINCT S.sname`
			`FROM Sailor S`
			`WHERE EXISTS(`
			`  SELECT *`
			`  FROM Boat B`
			`  WHERE B.color = 'red'`
			`  AND NOT EXISTS(`
			`  SELECT *`
			`  FROM Reserves R`
			`  WHERE R.bid = B.bid`
			`  AND R.sid = S.sid))`
				`SELECT DISTINCT S.sname`
				`FROM Sailor S`
				`WHERE NOT EXISTS(`
				`  SELECT *`
				`  FROM Boat B`
				`  WHERE B.color = 'red'`
				`  AND NOT EXISTS(`
				`  SELECT *`
				`  FROM Reserves R`
				`  WHERE R.bid = B.bid`
				`  AND R.sid = S.sid))`

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

4 categorical propositions with Sailors

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

Cat.	 Some S is B. $\exists x[S(x) \wedge B(x)]$	 No S is B. (All S are not B) $\neg\exists x[S(x) \wedge B(x)]$	 Some B is not S. $\exists x[B(x) \wedge \neg S(x)]$	 All B are S. $\neg\exists x[B(x) \wedge \neg S(x)]$
NL	Sailors who reserved **some** red boat	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats
SQL	`SELECT DISTINCT S.sname FROM Sailor S WHERE EXISTS( SELECT * FROM Reserves R WHERE R.sid = S.sid AND EXISTS( SELECT * FROM Boat B WHERE B.color = 'red' AND B.bid = R.bid))`	`SELECT DISTINCT S.sname FROM Sailor S WHERE NOT EXISTS( SELECT * FROM Reserves R WHERE R.sid = S.sid AND EXISTS( SELECT * FROM Boat B WHERE B.color = 'red' AND B.bid = R.bid))`	`SELECT DISTINCT S.sname FROM Sailor S WHERE EXISTS( SELECT * FROM Boat B WHERE B.color = 'red' AND NOT EXISTS( SELECT * FROM Reserves R WHERE R.bid = B.bid AND R.sid = S.sid))`	`SELECT DISTINCT S.sname FROM Sailor S WHERE NOT EXISTS( SELECT * FROM Boat B WHERE B.color = 'red' AND NOT EXISTS( SELECT * FROM Reserves R WHERE R.bid = B.bid AND R.sid = S.sid))`
RD				

dashed box = not exists

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Sailor (sid, sname, rating, bdate)
Reserves (sid, bid, day)
Boat (bid, bname, color, pdate)

	some	not any	not all	all
Sailors renting boats	have reserved some red boat	have not reserved any red boat	did not reserve all red boats	reserved all red boats

Example taken from: https://queryvis.com/example.html
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Student (sid, sname)
Takes (sid, cid, semester)
Class (cid, cname, depart)

Sailor (sid, sname, rating, bdate)
Reserves (sid, bid, day)
Boat (bid, bname, color, pdate)

	some	not any	not all	all
Sailors renting boats	have reserved some red boat	have not reserved any red boat	did not reserve all red boats	reserved all red boats
Students taking classes	took some art class	took no art class	took not all art classes	took all art classes

Example taken from: https://queryvis.com/example.html
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Actor (id, lname)
Play (aid, mid, role)
Movie (id, name, dir)

Student (sid, sname)
Takes (sid, cid, semester)
Class (cid, cname, depart)

Sailor (sid, sname, rating, bdate)
Reserves (sid, bid, day)
Boat (bid, bname, color, pdate)

	some	not any	not all	all
Sailors renting boats	have reserved some red boat	have not reserved any red boat	did not reserve all red boats	reserved all red boats
Students taking classes	took some art class	took no art class	took not all art classes	took all art classes
Actors playing in movies	played in some Hitchcock movie	did not play in a Hitchcock movie	did not play in all Hitchcock movies	played in all Hitchcock movies

Example taken from: https://queryvis.com/example.html
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Patterns

Actor (id, lname)
Play (aid, mid, role)
Movie (id, name, dir)

Student (sid, sname)
Takes (sid, cid, semester)
Class (cid, cname, depart)

Sailor (sid, sname, rating, bdate)
Reserves (sid, bid, day)
Boat (bid, bname, color, pdate)

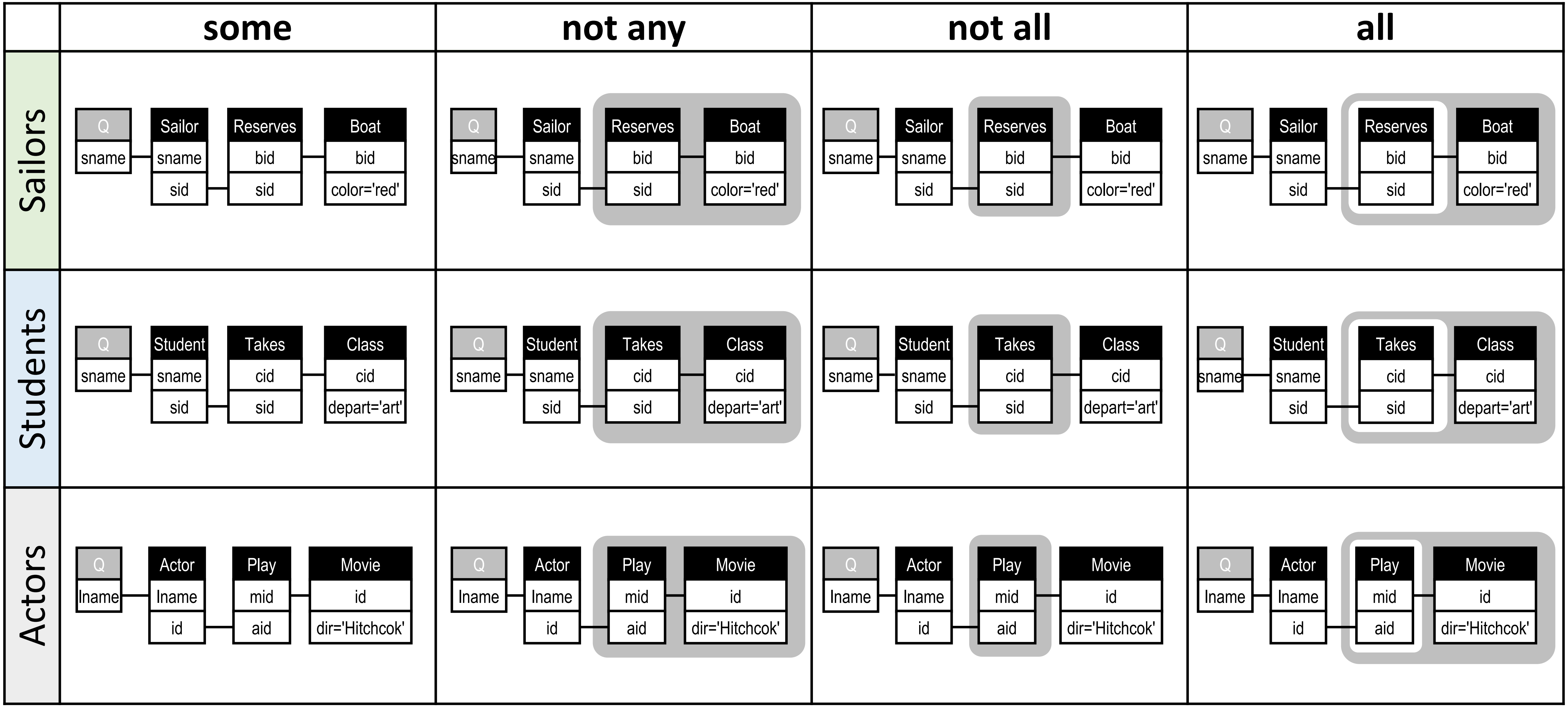
	some	not any	not all	all
Sailors	`SELECT DISTINCT S.sname FROM Sailor S WHERE EXISTS( SELECT * FROM Reserves R WHERE R.sid = S.sid AND EXISTS( SELECT * FROM Boat B WHERE B.color = 'red' AND B.bid = R.bid))`	`SELECT DISTINCT S.sname FROM Sailor S WHERE EXISTS( SELECT * FROM Boat B WHERE B.color = 'red' AND NOT EXISTS( SELECT * FROM Reserves R WHERE R.bid = B.bid AND R.sid = S.sid))`	`SELECT DISTINCT S.sname FROM Sailor S WHERE NOT EXISTS( SELECT * FROM Reserves R WHERE R.sid = S.sid AND EXISTS( SELECT * FROM Boat B WHERE B.color = 'red' AND B.bid = R.bid))`	`SELECT DISTINCT S.sname FROM Sailor S WHERE NOT EXISTS( SELECT * FROM Boat B WHERE B.color = 'red' AND NOT EXISTS( SELECT * FROM Reserves R WHERE R.bid = B.bid AND R.sid = S.sid))`
Students	`SELECT DISTINCT S.sname FROM Student S WHERE EXISTS( SELECT * FROM Takes T WHERE T.sid = S.sid AND EXISTS( SELECT * FROM Class C WHERE C.depart = 'art' AND C.cid = T.cid))`	`SELECT DISTINCT S.sname FROM Student S WHERE EXISTS( SELECT * FROM Class C WHERE C.depart = 'art' AND NOT EXISTS( SELECT * FROM Takes T WHERE T.cid = C.cid AND T.sid = S.sid))`	`SELECT DISTINCT S.sname FROM Student S WHERE NOT EXISTS( SELECT * FROM Takes T WHERE T.sid = S.sid AND EXISTS( SELECT * FROM Class C WHERE C.depart = 'art' AND C.cid = T.cid))`	`SELECT DISTINCT S.sname FROM Student S WHERE NOT EXISTS( SELECT * FROM Class C WHERE C.depart = 'art' AND NOT EXISTS( SELECT * FROM Takes T WHERE T.cid = C.cid AND T.sid = S.sid))`
Actors	`SELECT DISTINCT A.lname FROM Actor A WHERE EXISTS( SELECT * FROM Play P WHERE P.aid = A.id AND EXISTS( SELECT * FROM Movie M WHERE M.dir = 'Hitchcock' AND M.id = P.mid))`	`SELECT DISTINCT A.lname FROM Actor A WHERE EXISTS( SELECT * FROM Movie M WHERE M.dir = 'Hitchcock' AND NOT EXISTS( SELECT * FROM Play P WHERE P.mid = M.id AND P.aid = A.id))`	`SELECT DISTINCT A.lname FROM Actor A WHERE NOT EXISTS( SELECT * FROM Play P WHERE P.aid = A.id AND EXISTS( SELECT * FROM Movie M WHERE M.dir = 'Hitchcock' AND M.id = P.mid))`	`SELECT DISTINCT A.lname FROM Actor A WHERE NOT EXISTS( SELECT * FROM Movie M WHERE M.dir = 'Hitchcock' AND NOT EXISTS( SELECT * FROM Play P WHERE P.mid = M.id AND P.aid = A.id))`

Patterns

Actor (id, lname)
Play (aid, mid, role)
Movie (id, name, dir)

Student (sid, sname)
Takes (sid, cid, semester)
Class (cid, cname, depart)

Sailor (sid, sname, rating, bdate)
Reserves (sid, bid, day)
Boat (bid, bname, color, pdate)

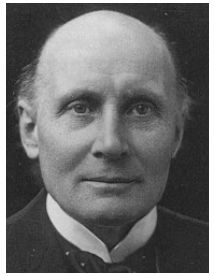


Outline

- What is a relational query? A vocabulary
- Various aspects of relational languages
  - Join queries
  - Lateral Joins
  - Declarativity of Languages
  - Grouping
  - Linear Composition
  - Relational Division
- Outlook

PLEASE ask questions and let me know if you spot any error anywhere ☺

An anonymous feedback form is linked from the tutorial web page:



[2050 ?]

721. *A. N. Whitehead*[3] writes on the power of symbols: "By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race. Before the introduction of the Arabic notation, multiplication was difficult, and the division even of integers called into play the highest mathematical faculties. Probably nothing in the modern world would have more astonished a Greek mathematician than to learn that, under the influence of compulsory education, the whole population of Western Europe, from the highest to the lowest, could perform the operation of division for the largest numbers. . . . . Mathematics is often considered a difficult and mysterious science, because of the numerous symbols which it employs. Of course, nothing is more incomprehensive than a symbolism which we do not understand. Also a symbolism, which we only partially understand and are unaccustomed to use, is difficult to follow. . . . . By the aid of symbolism, we can make transitions in reasoning almost mechanically by the eye, which otherwise would call into play the higher faculties of the brain. It is a profoundly erroneous truism, repeated by all copy-books and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them."

[Cajori 1929]

"... better relational language notation"

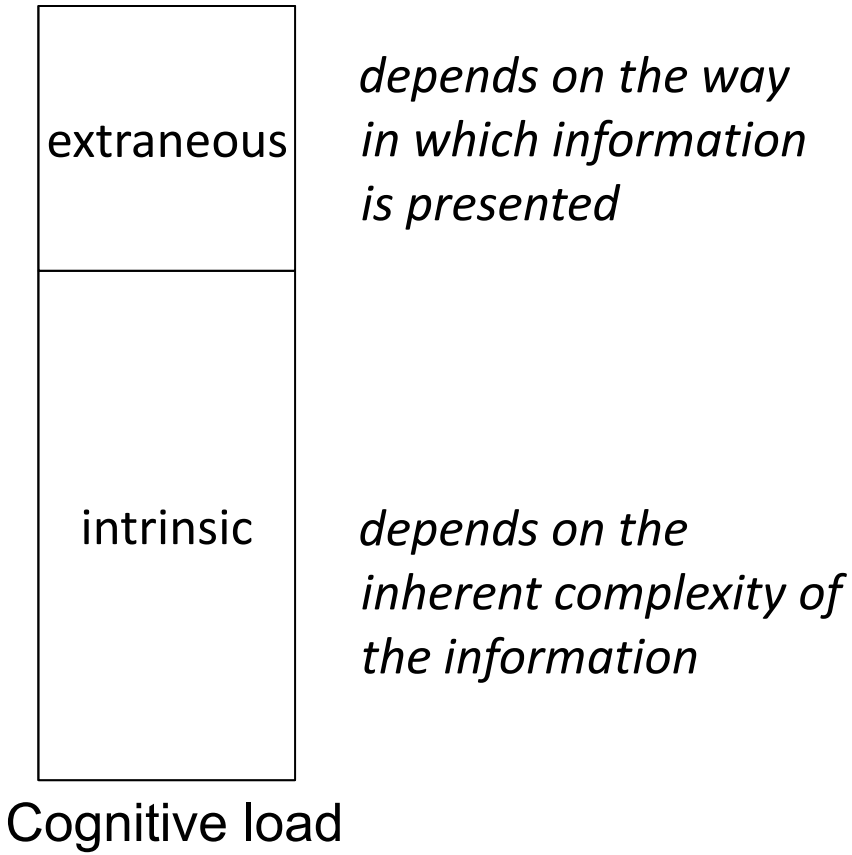
"... understanding relational queries became easy."

What is the most natural notation for relational queries ?

What is the intrinsic complexity of relational queries ?

Cajori. "A history of mathematical notations (books 1 and 2)", 1928/1929. https://archive.org/details/historyofmathema031756mbp/ , (https://en.wikipedia.org/wiki/A_History_of_Mathematical_Notations). Whitehead. "An introduction to mathematics", 1911. https://archive.org/details/introductiontoma00whitiala.

Cognitive load: intrinsic vs extraneous



depends on the way in which information is presented

depends on the inherent complexity of the information

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
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

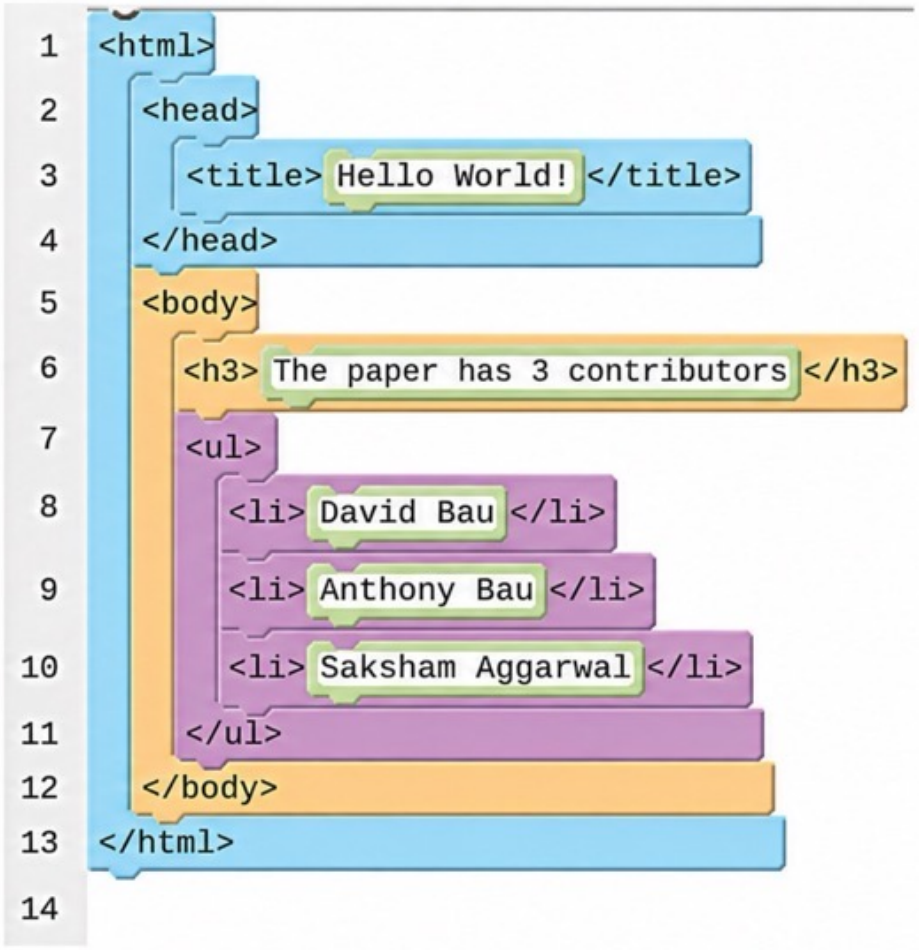
How much does indentation actually help?

```
1  <html>
2    <head>
3      <title>Hello World!</title>
4    </head>
5    <body>
6      <h3>The paper has 3 contributors</h3>
7      <ul>
8        <li>David Bau</li>
9        <li>Anthony Bau</li>
10       <li>Saksham Aggarwal</li>
11     </ul>
12   </body>
13 </html>
14
```

Original figure: Aggarwal, Bau, Bau. A blocks-based editor for HTML code, Blocks and Beyond 2015. https://doi.org/10.1109/BLOCKS.2015.7369008
Slightly modified from: Shapiro, Ahrens. Education Beyond Blocks: Syntax and Semantics, CACM 2016. https://doi.org/10.1145/2903751
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

How much does indentation actually help?

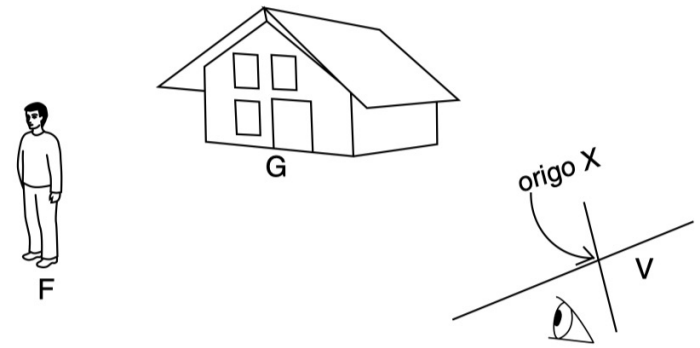
```
1  <html>
2    <head>
3      <title>Hello World!</title>
4    </head>
5    <body>
6      <h3>The paper has 3 contributors</h3>
7      <ul>
8        <li>David Bau</li>
9        <li>Anthony Bau</li>
10       <li>Saksham Aggarwal</li>
11     </ul>
12   </body>
13 </html>
14
```



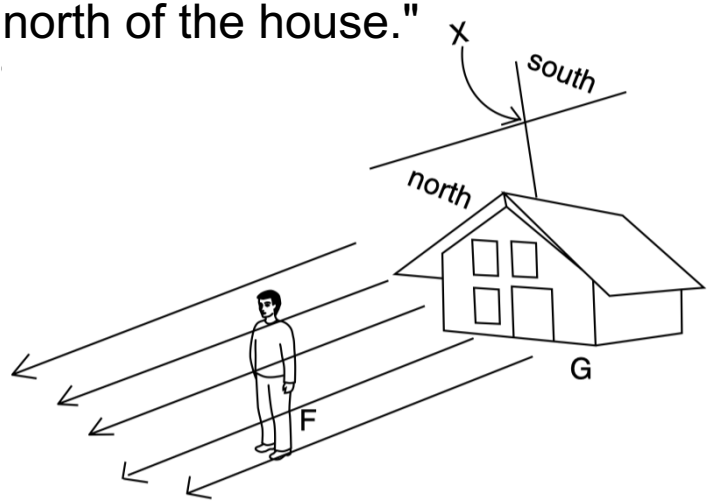
Original figure: Aggarwal, Bau, Bau. A blocks-based editor for HTML code, Blocks and Beyond 2015. https://doi.org/10.1109/BLOCKS.2015.7369008
Slightly modified from: Shapiro, Ahrens. Education Beyond Blocks: Syntax and Semantics, CACM 2016. https://doi.org/10.1145/2903751
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

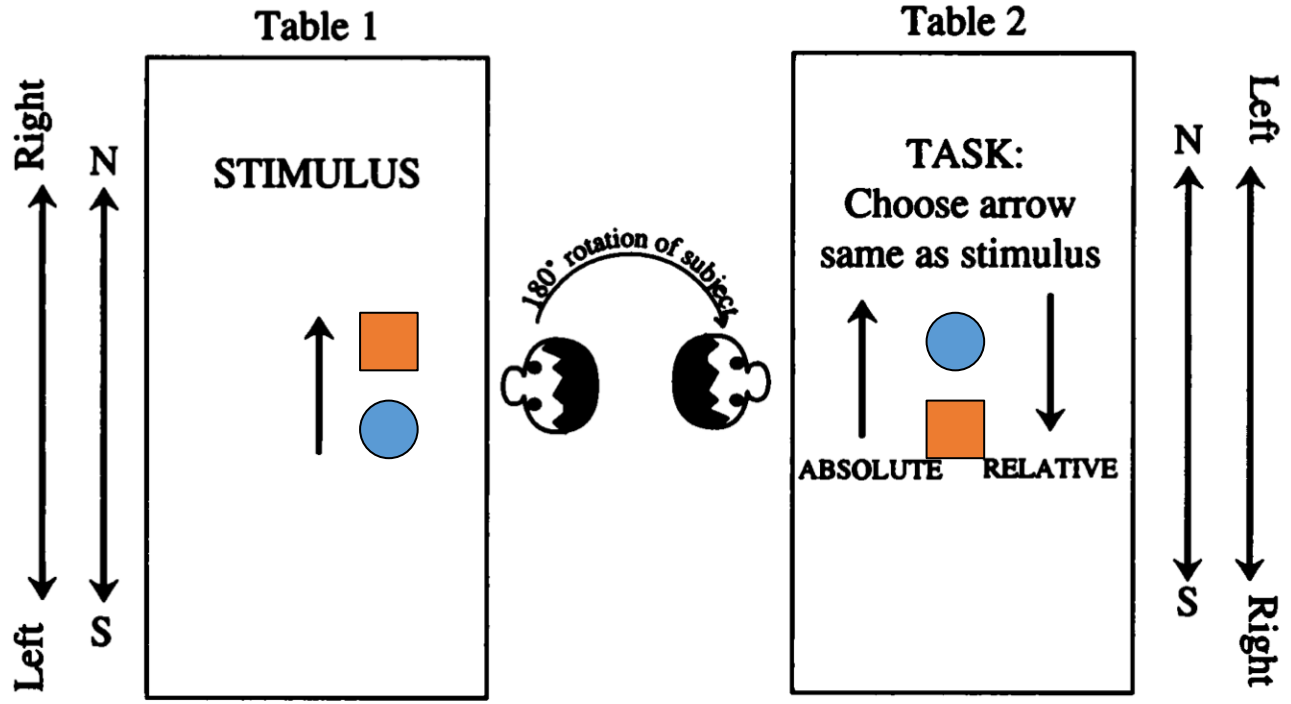
Linguistic relativity: Relative vs. Absolute

RELATIVE
"He's to the left of the house."



ABSOLUTE
"He's north of the house."





Comparing these two languages by "expressiveness" misses the point. Both can express the same situations.

They differ in what a language makes visible, i.e. what distinctions they make easier to encode or decode

Left: Excerpt of Figure 2.2 (3 frames of reference) from: Levinson. "Space in Language and Cognition: Explorations in Cognitive Diversity", Cambridge University Press 2003. https://doi.org/10.1017/CBO9780511613609
Right: Annotated Excerpt of Figure 4.2 from: Levinson. "Frames of Reference and Molyneux's Question: Crosslinguistic Evidence", MIT press, 1996. https://doi.org/10.7551/mitpress/4107.003.0006
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Absolute vs relative statements in Logic

SQL 1

```
select exists
  (select 1
  from R R2
  where not exists
    (select 1
    from S
    where not exists
      (select 1
      from R
      where R.A = R2.A
      and R.B = S.B)))
```

local edit

SQL 2

```
select not exists
  (select 1
  from R R2
  where not exists
    (select 1
    from S
    where not exists
      (select 1
      from R
      where R.A = R2.A
      and R.B = S.B)))
```

relative system of reference for negation scopes

? Can we have an absolute system of reference

Database 337 to run SQL queries https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql/
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Absolute vs relative statements in Logic

SQL 1

```
select exists
  (select 1
  from R R2
  where not exists
    (select 1
    from S
    where not exists
      (select 1
      from R
      where R.A = R2.A
      and R.B = S.B)))
```

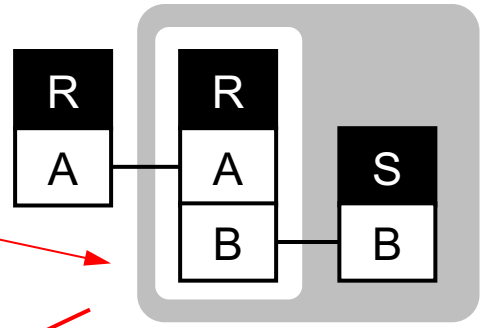
local edit

SQL 2

```
select not exists
  (select 1
  from R R2
  where not exists
    (select 1
    from S
    where not exists
      (select 1
      from R
      where R.A = R2.A
      and R.B = S.B)))
```

relative system of reference for negation scopes

shading indicates polarity of a zone: positive/monotone or negative/antitone



absolute system of reference for negation scopes with gray shades

$\exists r_2 \in R\ [\neg(\exists s \in S\ [\neg(\exists r \in R\ [r.A = r_2.A \wedge r.A = s.B])])]$

global edit

?

$\neg(\exists r_2 \in R\ [\neg(\exists s \in S\ [\neg(\exists r \in R\ [r.A = r_2.A \wedge r.A = s.B])])])$

Database 337 to run SQL queries https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql/
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Absolute vs relative statements in Logic

SQL 1

```
select exists
  (select 1
  from R R2
  where not exists
    (select 1
    from S
    where not exists
      (select 1
      from R
      where R.A = R2.A
      and R.B = S.B)))
```

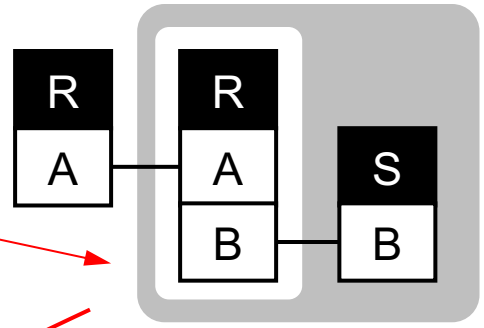
local edit

SQL 2

```
select not exists
  (select 1
  from R R2
  where not exists
    (select 1
    from S
    where not exists
      (select 1
      from R
      where R.A = R2.A
      and R.B = S.B)))
```

relative system of reference for negation scopes

shading indicates polarity of a zone: positive/monotone or negative/antitone

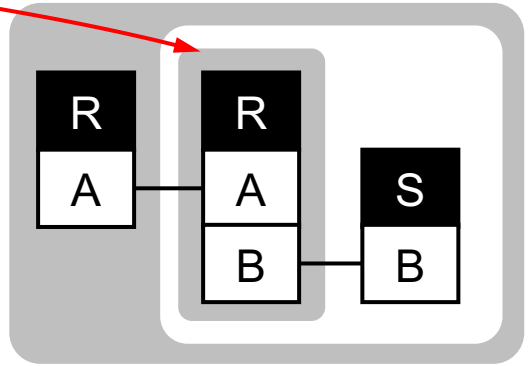


absolute system of reference for negation scopes with gray shades

$\exists r_2 \in R\ [\neg(\exists s \in S\ [\neg(\exists r \in R\ [r.A = r_2.A \wedge r.A = s.B])])]$

$\exists r_2 \in R\ [\ \ \forall s \in S\ [\ \ \exists r \in R\ [r.A = r_2.A \wedge r.A = s.B]\ ]\ ]$

global edit



$\neg(\exists r_2 \in R\ [\neg(\exists s \in S\ [\neg(\exists r \in R\ [r.A = r_2.A \wedge r.A = s.B])])])$

$\neg(\exists r_2 \in R\ [\ \ \forall s \in S\ [\ \ \exists r \in R\ [r.A = r_2.A \wedge r.A = s.B]\ ]\ ])$

$\forall r_2 \in R\ [\ \ \exists s \in S\ [\neg(\exists r \in R\ [r.A = r_2.A \wedge r.A = s.B])]\ ]$

Database 337 to run SQL queries https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql/

Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

Absolute vs relative statements in Logic

today 13:30 R-21: Query Languages & Execution:
"A Principled Solution to the Disjunction Problem of Diagrammatic Query Representations"

SQL 1

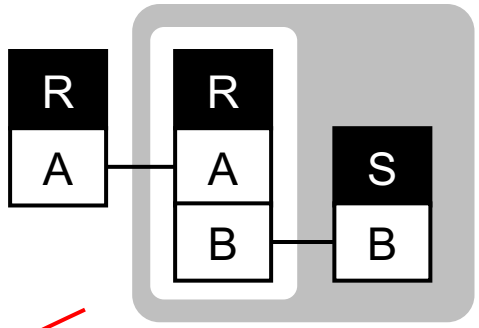
```
sel
  (
  f
  w
    where not exists
      (select 1
      from R
      where R.A = R2.A
      and R.B = S.B)))
```

local edit

SQL 2

```
select not exists
  (select 1
  from R R2
  where not exists
    (select 1
    from S
    where not exists
      (select 1
      from R
      where R.A = R2.A
      and R.B = S.B)))
```

relative system of reference for negation scopes

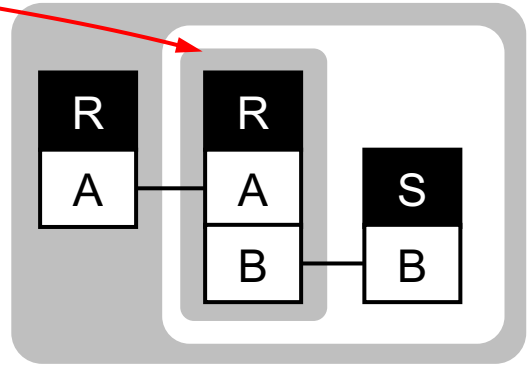


absolute system of reference for negation scopes with gray shades

$\exists r_2 \in R\ [\neg(\exists s \in S\ [\neg(\exists r \in R\ [r.A = r_2.A \wedge r.A = s.B])])]$

$\exists r_2 \in R\ [\ \ \forall s \in S\ [\ \ \exists r \in R\ [r.A = r_2.A \wedge r.A = s.B]\ ]\ ]$

global edit



$\neg(\exists r_2 \in R\ [\neg(\exists s \in S\ [\neg(\exists r \in R\ [r.A = r_2.A \wedge r.A = s.B])])])$

$\neg(\exists r_2 \in R\ [\ \ \forall s \in S\ [\ \ \exists r \in R\ [r.A = r_2.A \wedge r.A = s.B]\ ]\ ])$

$\forall r_2 \in R\ [\ \ \exists s \in S\ [\neg(\exists r \in R\ [r.A = r_2.A \wedge r.A = s.B])]\ ]$

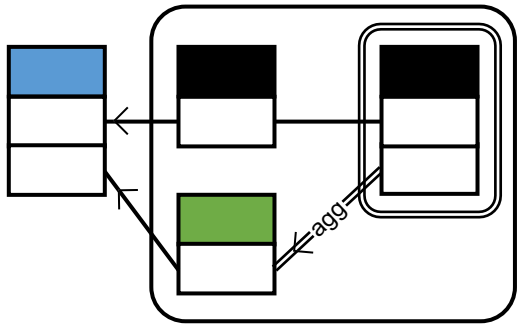
Database 337 to run SQL queries https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql/
Wolfgang Gatterbauer. A tutorial on relational language design: https://northeastern-datalab.github.io/relational-language-tutorial/

A Catalogue of Relational Patterns

SQL1

```
select E.eid,
  (select sum0(val) as sm
  from Sales S
  where E.eid = S.eid)
from Empl E
```

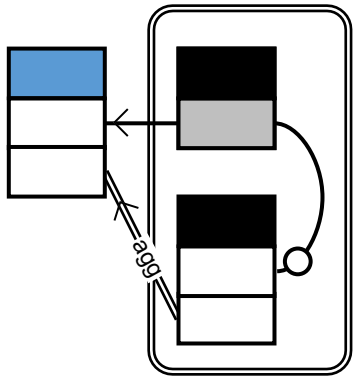
Correlated subcollection summary: Given an outer collection and an inner collection, use each outer tuple to select a matching subcollection of inner tuples. Summarize each subcollection with an aggregate, and output selected outer attributes together with the summary.



SQL2

```
select E.eid, sum0(val) as sm
from Empl E
left outer join Sales S
on E.eid = S.eid
group by E.eid
```

Outer-preserving grouped summary: Given an outer collection and an inner collection, match outer tuples with inner tuples while preserving outer tuples that have no match. Group the resulting tuples by selected outer attributes. Summarize each group with an aggregate over inner values, and output the grouping attributes together with the summary.

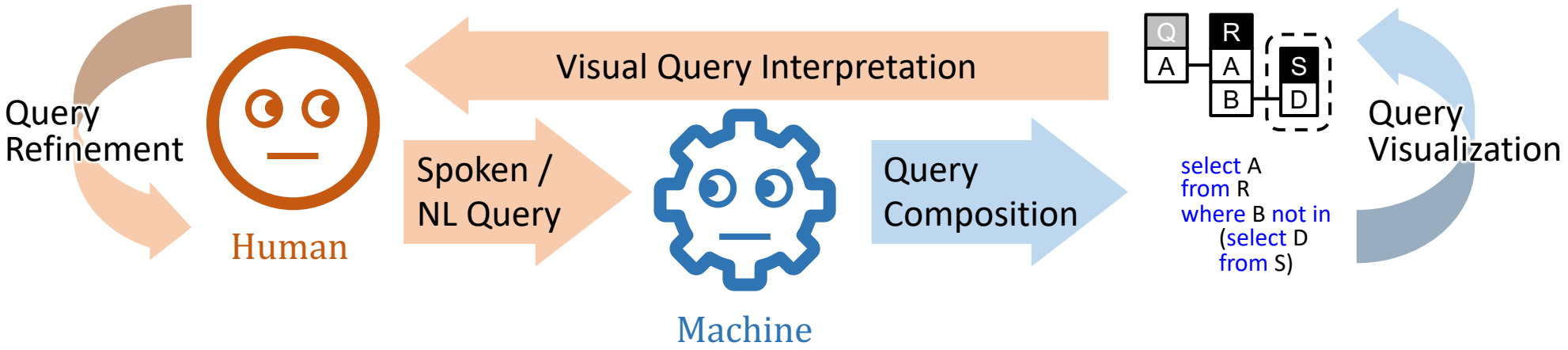


What is the catalogue of most common patterns used today ?

A possible future human-query interaction

What is the most natural notation for relational queries ?

What is the intrinsic complexity of relational queries ?



A number of open questions

1. What is the most natural notation for relational queries?
   - One single notation or it depends: Generating by LLMs, reading & debugging by humans, optimizing, or may each favor a different notation.
2. What is the intrinsic complexity of relational queries?
   - When a query is hard to understand (e.g. relational division), which difficulty comes from the pattern itself, and which comes from the chosen notation? Which aspects of a pattern (a subpattern) is hard?
3. Can we build a pattern catalogue for relational queries, analogous to design patterns in programming languages?
   - e.g. inside-out vs. outside in groupings
4. What are the right units of meaning? What should be atomic vs. composed building blocks? What should make a language visible?
   - e.g., individual operators, bigger patterns like the ones we saw, ...

A number of open questions

5. What is the right way to describe patterns (notation independent)?
   - e.g. "an outer-preserving grouped summary" instead of "left join plus group by"
6. What is the right granularity for a catalogue of relational patterns?
   - Too small, and it collapses important distinctions.
   - Too large, and it may become a list of SQL idioms.
7. How do we design empirical studies that measure how notational design choices affect human readability?
   - Synthetic isolated questions vs. in-vivo complex environments?
   - How to make such human studies scalable today (LLM used by AMT workers)?
   - Even questions like "What is the most understandable SQL indentation?" are open
8. Mapping from language aspects to granular pattern recognitions:
   - What notation is better for seeing grouping / for seeing a pipeline / for seeing join conditions / for conciseness

A number of open questions

9. How to estimate the "cognitive cost" for understanding a query?
   - Based on smaller building blocks and the cost for "gluing together"
10. What makes a query "locally editable"?
   - e.g. employees with sales → employees with no sales
11. What should be explicit vs. implicit (semantic conventions) in a language?
   - e.g. empty aggregates
12. Different languages for understanding intent vs. optimizing?
   - related to the question on what are the primitive building blocks exposed; logical vs. physical; ("intermediate representations")
13. What is the right target for NL-to-x translations by LLMs?
   - What should a query language expose to make machine assistance reliable? E.g. explicit scopes, explicit grouping, explicit join intent, making null behavior explicit
   - With language transpilers, we could quickly create tons of training sets.

Not considered today, but intended for VLDB later this summer

- Recursion
- Nested Relational Model
- Window functions
- Order
- Different Join Definitions & Graph QLs

Acknowledgements

- Thanks especially to Molham Aref, Val Tannen, Leonidas Fegaras, Julian Hyde, Diandre Sabale, Mitch Wand, Jun Yang for stimulating discussions on various aspects of the topics today and great feedback

Today 13:30 R-21: Query Languages & Execution:
"A Principled Solution to the Disjunction Problem of Diagrammatic Query Representations"

Thanks ☺

An anonymous feedback form is linked from the tutorial web page: