Database Theory

Unit 1 — Relational Query Languages

Matthias Lanzinger, WS2024/25

TU WIEN Informatics

The Relational Model

Setup

- The relational database model is essentially first-order structures + names for attributes.
- This extra information on attributes can be useful when describing queries.
- Sometimes it is unnecessary and we just work on plain first-order structures.

Reminder First-order structures:

Relations $R_1, R_2, \ldots, R_n$ that each have an arity $\#R_i$.

Assigned meaning through an interpretation I over a domain Dom:

$$I(R_i) \subseteq Dom_n^{\#R_i}$$

Schemas

- Let **Rel** and **Att** be (countably infinite) vocabularies of relation names and attribute names.

- A database schema $\mathcal{S}$ is a partial function $\mathcal{S} : \mathsf{Rel} \to 2^{\mathsf{Att}}$ such that $Dom(\mathcal{S})$ is finite and every image under $\mathcal{S}$ is finite.

- The arity of $R \in Dom(\mathcal{S})$ is defined as $|\mathcal{S}(R)|$.

Example:

In a database we have a table $Student$ with columns for id, name and birthdate.

Formally, there is a relation name $Student \in \mathsf{Rel}$ and we use the schema $\mathcal{S}(Student) = (id, name, brithdate)$ where these are names in **Att**.

Relation Instances

- Each attribute $A \in \mathsf{Att}$ has a domain $Dom(A)$.
- A tuple for relation name R (under schema $\mathcal{S}$) is an element of $Dom(A_1) \times Dom(A_2) \times \cdots \times Dom(A_n)$ where $\mathcal{S}(R) = (A_1, A_2, \ldots, A_n)$.
- A relation (instance) for R (under $\mathcal{S}$) is a *finite* set of tuples for R
- A database is a finite set of relations under some schema $\mathcal{S}$.

Continuing our example:

$Dom(id) = \mathbb{N}$,

$Dom(name) =$ all strings (Σ^* for some alphabet Σ)

$Dom(birthdate)$ = e.g., all strings of certain format

Example tuple:

$(1,$ Bob, 12-03-4567$) \in Dom(id) \times Dom(name) \times Dom(birthdate)$

Some Helpful Notation

1. Relation Names vs. Relation Instances

   When we have a database D we use R^D to denote the relation instance for relation name R

2. Attributes of Tuples

   For a relation R^D with schema $\mathcal{S}(R) = (A_1, \ldots, A_k)$, we use $A_i(t)$ to denote the i-th position of tuple $t \in R^D$.

Id	Name	DoB
13	Student A	14.06.19
22	Student B	23.06.19
...	...	..

$ID(t_1) = 13, \quad Name(t_1) =$ Student A

Relational Query Languages

Formal Query Languages

- As in any discussion of formal languages, query languages always have two core parts.
- Syntax
  How do terms of the language look like. Can be analogous to logic or more operational.
- Semantics
  How are expressions of the languages evaluated. The formal definition of what answers we want from the data.

```
SELECT MIN(s_acctbal), MAX(s_acctbal)
FROM part, partsupp, supplier,
     nation, region
WHERE p_partkey    = ps_partkey
  AND s_suppkey    = ps_suppkey
  AND n_nationkey = s_nationkey
  AND r_regionkey = n_regionkey
  AND p_price >
      (SELECT avg (p_price) FROM part)
  AND r_name IN ('Europe', 'Asia')
```

$$\exists y\ x \xrightarrow{a^*b} y \wedge x \xrightarrow{(a+b)^*c} y$$

$$\pi_{\{\text{pid},\text{pname}\}}(\text{Person}) - \pi_{\{\text{pid},\text{pname}\}}(\text{Person} \ltimes \text{City})$$

$$(\forall x)(\exists yz)(y \neq z \wedge E(x,y) \wedge E(x,z) \wedge (\forall w)(E(x,w) \rightarrow (w = y \vee w = z)))$$

```
SELECT ?capital
       ?country
WHERE
  {
    ?x  ex:cityname        ?capital
        ex:isCapitalOf     ?y
    ?y  ex:countryname     ?country
        ex:isInContinent   ex:Africa
  }
```

$$\begin{aligned} \exists z \mathsf{Drive}(z,x,y) &\leftarrow (\diamondminus_{[0,\infty]}\mathsf{Depart}(x,y))\ \mathcal{U}_{[0,\infty)}\mathsf{Arrive}(x,y), \\ \mathsf{Working}(z) &\leftarrow \mathsf{Drive}(z,x,y), \\ \mathsf{Dangerous}(x) &\leftarrow \boxminus_{[0,8]}\mathsf{Working}(z) \wedge \mathsf{Drive}(z,x,y). \end{aligned}$$

Example Schema

Course

Name	Sem	Lecturer
Logic	W24	L1
Complexity	S24	L2
Logic	W23	L1

Student

Id	Name	DoB	Active
13	Student A	14.06.1903	TRUE
22	Student B	23.06.1912	FALSE
...	...	..	...

Enrolled

Course	Sem	Student
Logic	W24	13
Logic	W24	22
Complexity	S24	13

Relational Algebra

Overview

Syntax:

Relational Algebra (RA) expressions e are formed inductively:

- every relation name R is an RA expression

- If e_1, e_2 is an RA expressions, so are:

$$\sigma_\theta(e_1) \qquad \pi_\alpha(e_1) \qquad \rho_{A \to B}(e_1)$$
$$e_1 \times e_2 \qquad e_1 \cup e_2 \qquad e_1 - e_2$$

Semantics:

An expression e applied to a database D evaluates to a new relation, we write $e(D)$.

If e is relation name R, then $e(D) = R^D$.

Semantics of other operators are defined on the following slides.

$$\sigma$$

Selection

Syntax:

$e = \sigma_\theta(e_1)$ where

- e_1 is a RA expression of sort U
- θ is a propositional formula over attributes in U, $=$, and constants.

Semantics:

$e(D) = \{t \in e_1(D) \mid \theta(t)$ is true $\}$
with schema $\mathcal{S}(e) = \mathcal{S}(e_1)$

$$\sigma$$

Selection

$$\sigma_{Sem='W24'}(Enrolled) \longrightarrow$$

Course	Sem	Student
Logic	W24	13
Logic	W24	22
Complexity	S24	13

$$\pi$$

Projection

Syntax:

$e = \pi_\alpha(e_1)$ where

- e_1 is a RA expression of sort U
- α is sequence of attributes in U

Semantics:

$$e(D) = \{(\alpha_1(t), \alpha_2(t), \ldots, \alpha_{|\alpha|}) \mid t \in e_1(D)\}$$

with schema $\mathcal{S}(e) = \alpha$

$$\pi$$

Projection

$$\pi_{Active,ID}(Student) \longrightarrow$$

Active	Id
TRUE	13
FALSE	22
...	...

ρ

Renaming

Syntax:

$e = \rho_{A \to B}(e_1)$ where

- e_1 is a RA expression of sort U
- $A \in U$, and $B \in \mathsf{Att} \setminus U$

Semantics:

$e(D) = e_1(D)$

with schema $\mathcal{S}(e) = \mathcal{S}(e_1)[A/B]$

Replace A with B

$$\rho$$

Renaming

$$\rho_{Name \to Course, Student \to ID}(Enrolled) \longrightarrow$$

Course	Sem	ID
Logic	W24	13
Logic	W24	22
Complexity	S24	13

$\times$

Product

Syntax:

$e = e_1 \times e_2$ where

- e_1, e_2 are RA expressions with schema $(A_1, \ldots, A_n)$ and $(B_1, \ldots, B_m)$, respectively.

Semantics:

$$e_1(D) = \{(a_1, \ldots, a_n, b_1, \ldots, b_m) \mid (a_1, \ldots, a_n) \in R, (b_1, \ldots, b_m) \in S\}$$

with schema $\mathcal{S}(e) = (A_1, \ldots, A_n, B_1, \ldots, B_n)$

$\times$

Product

$$Enrolled \times \pi_{ID,Name}(Student) \longrightarrow$$

Course	Sem	Student	ID	Name
Logic	W24	13	13	Student A
Logic	W24	13	22	Student B
Logic	W24	22	13	Student A
Logic	W24	22	22	Student B
Complexity	S24	13	13	Student A
Complexity	S24	13	22	Student B
...	...	...	...	...

$-$

Difference

Syntax:

$e = e_1 - e_2$ where

- e_1, e_2 are RA expressions of sort U

Semantics:

$e(D) = e_1(D) \setminus e_2(D)$

$\cup$

Union

Syntax:

$e = e_1 \cup e_2$ where

- e_1, e_2 are RA expressions of sort U

Semantics:

$e(D) = e_1(D) \cup e_2(D)$

$\bowtie$

(Natural) Join

Very common in database queries.
Can be expressed via other operators:

$$e_1 \bowtie e_2 := \sigma_{A=A'}(e_1 \times \rho_{A\to A'}(e_2))$$

Where A is the only shared attribute between e_1, e_2.
(generalisation to more shared attributes is straightforward)

Example Query

List lectures together with the students that attend them in WS24:

$$\overbrace{\pi_{Course,Student}\left(\overbrace{\overbrace{\pi_{Course,Student}(Enrolled)}^{\text{Remove Sem attribute}} \bowtie \overbrace{\rho_{Name\to Course}\,\overbrace{\sigma_{Sem=WS24}(Course)}^{\text{Limit to WS24}}}^{\text{Rename for Join}} \bowtie \overbrace{\rho_{Id\to Student}(Student)}^{\text{Rename for Join}}}^{\text{Connect student data to course data via enrolment}}\right)}^{\text{Keep only the two attributes that we want}}$$

The natural database theory question:
Do we need all of these operators?

Yes 👍 — but how do we show this?

(Except for $\bowtie$ of course)

Example: Renaming

Relation R

A
1
2
3

Relation S

B
3
4
5

Consider the expression

$R \cup \rho_{B \to A}(S)$

A
1
2
3
4
5

We can show that ρ is necessary by showing that in RA without renaming there is no expression that gives the same output!

Relational Domain Calculus

= First-Order Queries

Queries from Logic

Let φ be a formula with free variables $x_1, \ldots, x_k$, then

$$\{(x_1, x_2, \ldots, x_k) \in \mathsf{Dom}^k \mid D \vDash \varphi(x_1, x_2, \ldots, x_k)\}$$

Is a k-ary query, i.e., it returns a set of k tuples that represent "solutions" for φ on database D.

Logics can be seen as query languages!

Relational Domain Calculus

The query language induced by *first-order logic* is called relational (domain) calculus.

Quick reminder — Semantics of first-order logic:

$$
\begin{aligned}
I \models R(x_1, \ldots, x_n) \quad &\iff \quad R(I(x_1), \ldots, I(x_n)) \\
I \models x = y \quad &\iff \quad I(x) = I(y) \\
I \models x = c \quad &\iff \quad I(x) = c \\
I \models \neg\phi \quad &\iff \quad I \not\models \phi \\
I \models \phi_1 \wedge \phi_2 \quad &\iff \quad I \models \phi_1 \text{ and } \phi_2 \\
I \models \exists x \,.\, \phi \quad &\iff \quad I \models \phi[x/c] \text{ for every } c \in Dom
\end{aligned}
$$

with $\forall x \,.\, \phi := \neg \exists x \,.\, \neg\phi$ and $\phi_1 \vee \phi_2 := \neg(\neg\phi_1 \wedge \neg\phi_2)$

Example Query

Recall our example query: list lectures together with the students that attend them in WS24.

$$
\begin{aligned}
\{(c, s) \mid \exists sem_1, sem_2, sid, l, dob, active\ .\ & \\
& Enrolled(c, sem_1, sid) \land Course(c, sem_2, z) \land \\
& sem_1 = \text{WS24} \land Student(sid, s, dob, active)\}
\end{aligned}
$$

Course

Name	Sem	Lecturer
Logic	W24	L1
Complexity	S24	L2
Logic	W23	L1

Student

Id	Name	DoB	Active
13	Student A	14.06.1903	TRUE
22	Student B	23.06.1912	FALSE
...	...	..	...

Enrolled

Course	Sem	Student
Logic	W24	13
Logic	W24	22
Complexity	S24	13

$$\{(c, s) \mid \exists sem_1, sem_2, sid, l, dob, active\ . \\ Enrolled(c, sem_1, sid) \wedge Course(c, sem_2, z) \wedge \\ sem_1 = \mathrm{WS24} \wedge Student(sid, s, dob, active)\}$$

(Logic, Student A) is an answer to the query:

$$sem_1 \mapsto W24, \quad sid \mapsto 13, \quad dob \mapsto 14.06.1903, \quad active \mapsto \mathrm{TRUE}, \quad l \mapsto L1$$

What about sem_2?

Course

Name	Sem	Lecturer
Logic	W24	L1
Complexity	S24	L2
Logic	W23	L1

Student

Id	Name	DoB	Active
13	Student A	14.06.1903	TRUE
22	Student B	23.06.1912	FALSE
...	...	..	...

Enrolled

Course	Sem	Student
Logic	W24	13
Logic	W24	22
Complexity	S24	13

$$\{(c, s) \mid \exists sem_1, sem_2, sid, l, dob, active \,. \\ Enrolled(c, sem_1, sid) \wedge Course(c, sem_2, z) \wedge \\ sem_1 = \text{WS24} \wedge Student(sid, s, dob, active)\}$$

(Logic, Student A) is an answer to the query:

$$sem_1 \mapsto W24, \quad sid \mapsto 13, \quad dob \mapsto 14.06.1903, \quad active \mapsto \text{TRUE}, \quad l \mapsto L1$$

What about sem_2? Could be$W24$ or $W23$ such that $Enrolled(c, sem_1, l)$ is true.

SQL

SQL Overview

- *The* standard language for relational databases.
- Originally developed in the 1970s inspired by the relational model and especially relational algebra.

SQL Query Syntax…

We are not going to formally define SQL.

- Syntax changes between implementations.

- Contains constructs that specify details of the actual execution of the query, e.g.,

  WITH … AS MATERIALIZED

  which makes it challenging to specify formal semantics.

Image source: https://www.postgresql.org/docs/current/sql-select.html

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    [ { * | expression [ [ AS ] output_name ] } [, ...] ]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
    [ HAVING condition ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } { ONLY | WITH TIES } ]
    [ FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF from_reference [, ...] ] [ NOWAIT | SKIP LOCKED ] [.

where from_item can be one of:

    [ ONLY ] table_name [ * ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
                [ TABLESAMPLE sampling_method ( argument [, ...] ) [ REPEATABLE ( seed ) ] ]
    [ LATERAL ] ( select ) [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    [ LATERAL ] function_name ( [ argument [, ...] ] )
                [ WITH ORDINALITY ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    [ LATERAL ] function_name ( [ argument [, ...] ] ) [ AS ] alias ( column_definition [, ...] )
    [ LATERAL ] function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
    [ LATERAL ] ROWS FROM( function_name ( [ argument [, ...] ] ) [ AS ( column_definition [, ...] ) ] [, ...] )
                [ WITH ORDINALITY ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    from_item join_type from_item { ON join_condition | USING ( join_column [, ...] ) [ AS join_using_alias ] }
    from_item NATURAL join_type from_item
    from_item CROSS JOIN from_item

and grouping_element can be one of:

    ( )
    expression
    ( expression [, ...] )
    ROLLUP ( { expression | ( expression [, ...] ) } [, ...] )
    CUBE ( { expression | ( expression [, ...] ) } [, ...] )
    GROUPING SETS ( grouping_element [, ...] )

and with_query is:

    with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ] ( select | values | insert | update |
        [ SEARCH { BREADTH | DEPTH } FIRST BY column_name [, ...] SET search_seq_col_name ]
        [ CYCLE column_name [, ...] SET cycle_mark_col_name [ TO cycle_mark_value DEFAULT cycle_mark_default ] USI

TABLE [ ONLY ] table_name [ * ]
```

Core SQL Queries

$Q_1, Q_2 :=$ **SELECT** <select_list>
FROM <from_list>
WHERE <condition>

| Q_1 **UNION** Q_2

| Q_1 **EXCEPT** Q_2

Core SQL Queries

$Q_1, Q_2 :=$ **SELECT** <select_list>
FROM <from_list>
WHERE <condition>

| Q_1 **UNION** Q_2

| Q_1 **EXCEPT** Q_2

Constants or attributes of relation names from the <from_list>

Core SQL Queries

$Q_1, Q_2 :=$ **SELECT** <select_list>
FROM <from_list>
WHERE <condition>

| Q_1 **UNION** Q_2

| Q_1 **EXCEPT** Q_2

List of relation names and subqueries used in the query. Can be aliased (renamed) to use relation repeatedly.

Core SQL Queries

$Q_1, Q_2 :=$ **SELECT** <select_list>
FROM <from_list>
WHERE <condition>

| Q_1 **UNION** Q_2

| Q_1 **EXCEPT** Q_2

An *expression* consisting of:

- Equalities between attributes, e.g., R.a = S.a.
- Equalities between attributes and constants, e.g., R.a = 7
- Combinations of expressions using **AND**, **OR**, and **NOT**.

Core SQL Queries

Theorem

Core SQL queries are equivalent in expressiveness to Relational Algebra.

That is, for every Core SQL query q, there exists an RA query q' such that $q(D) = q'(D)$ for every database D, and vice versa.

For details, see Arenas, et al. "Database Theory.", Section 5.

Informally, this means that we can focus our theoretical analysis only on one of these languages!

SQL — Example

List lectures together with the students that attend them in WS24:

```sql
SELECT course.name, student.name
FROM course, student, enrolled
WHERE course.name = enrolled.course AND
      course.sem = enrolled.sem AND
      student.id = enrolled.student AND
      enrolled.sem = 'WS24';
```

Warning: Bag vs. Set Semantics

Set Semantics: Answers to queries are *sets* of tuples. That is, there is no repetition in answers and operations ignore repeating tuples.

Bag Semantics: Answers to queries are *bags (or multisets)* of tuples. Repetition matters!

A	B
1	2
1	2

$\bowtie$

B	C
2	3
2	4

=

A	B	C
1	2	3
1	2	4

Set Semantics

/

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

Bag Semantics

Warning: Bag vs. Set Semantics

Set Semantics: Answers to queries are *sets* of tuples. That is, there is no repetition in answers and operations ignore repeating tuples.

Bag Semantics: Answers to queries are *bags (or multisets)* of tuples. Repetition matters!

Our definition of relational algebra and relational calculus uses set semantics.
In the statement on the previous slide we assume set semantics for core SQL queries.
However, SQL in practical systems usually uses bag semantics.

Not a problem, it is also straightforward to define relational algebra with bag semantics.
But it is important to always keep in mind which type of semantics we are talking about.

SQL is More than SELECT

Data Description

CREATE: To create new tables, databases, views, or indexes.

ALTER: To modify existing database objects (e.g., add columns to a table).

Data Manipulation

GRANT: To provide specific privileges (e.g., SELECT, INSERT) to users or roles.

REVOKE: To remove previously granted privileges.

Data Control

INSERT: Adds new rows (records) to a table.

UPDATE: Modifies existing rows in a table based on certain conditions.

DELETE: Removes rows from a table based on specified conditions.

Which is Best?

Codd's Theorem

Theorem (Codd 1972, informal)
Relational algebra, relational domain calculus, and Core SQL Queries have the same expressive power.

Limitations apply:

- Relational calculus queries have to be "safe"

We will work through the details of Codd's Theorem in Lecture 3!

Qualitative Comparison

As a consequence of the languages being equivalent
we can switch between them depending on the task.

Relational Algebra

Operational semantics are well suited for topics where we care about the steps taken to execute a query.

Well-suited for the study of query optimisation and execution.

Relational Calculus

Declarative language with cleanest semantics. Direct connection to extensive body of work on logic.

Well-suited for theoretical study of complexity and expressivity.

SQL

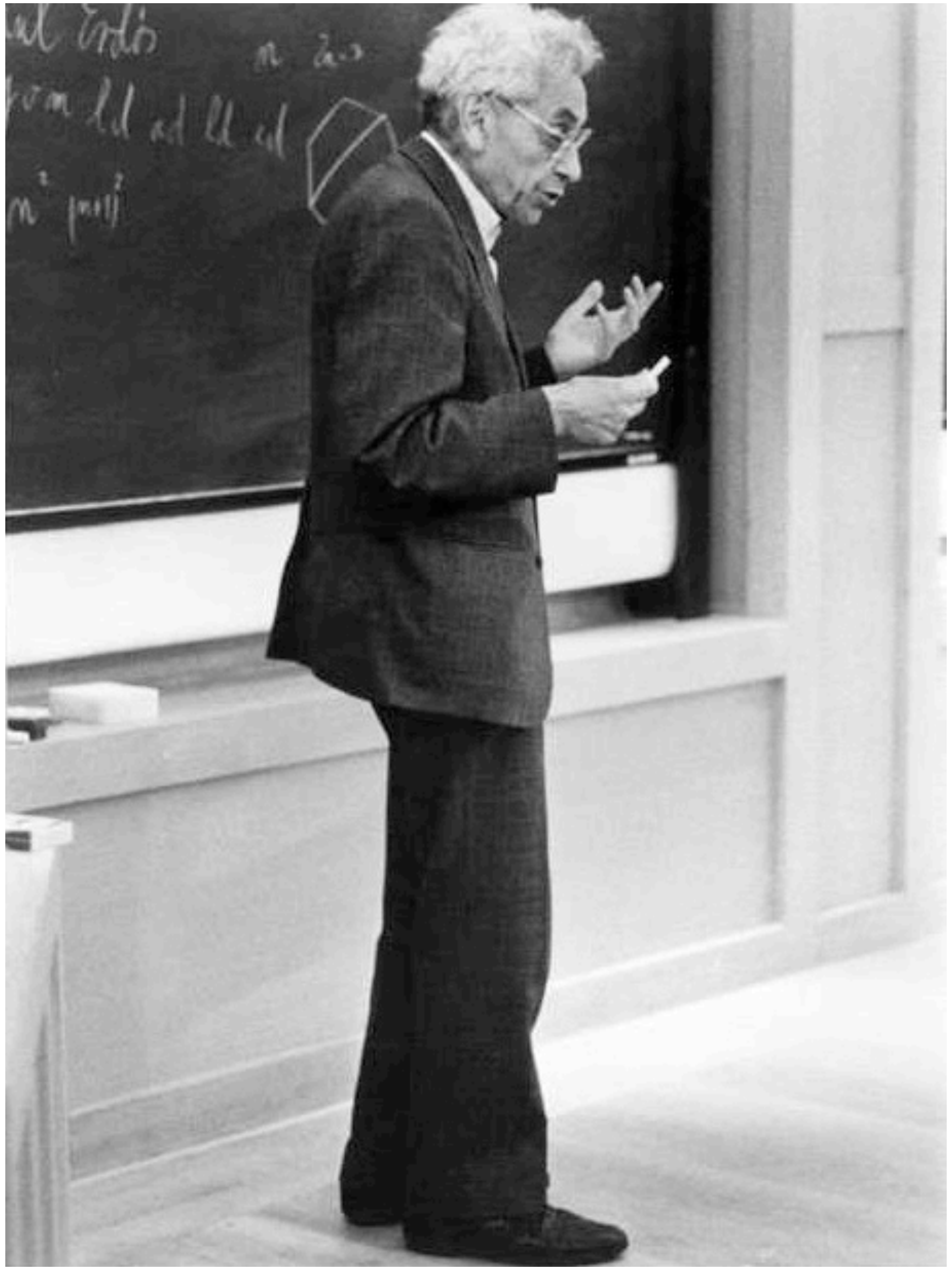
"User-friendly" language aimed at end-users of actual systems. Extremely wide-spread in the real world.

Limitations

Limitations?

- Paul Erdős was one of the most prolific mathematicians of all time. He wrote over 1500 articles, many of them highly influential.
  He had 509 direct collaborators!

- The Erdős Number is a way of describing the "collaboration distance" Paul Erdős.
  - Erdős has an Erdős number of 0
  - The Erdős Number of author M is the minimum among the Erdős Numbers of all the coauthors of M, plus 1



https://sites.math.rutgers.edu/~sg1108/People/Math/Erdos

Example

Paul Erdős
EN = 0

Coauthor

Noga Alon
EN = 1

Coauthor

Mike Fellows
EN = 2

Coauthor

Igor Razgon
EN = 3

Coauthor

Matthias Lanzinger
EN = 4

We see, it's simply a fun way of describing shortest path in the coauthor graph.

Isaac Asimov

EN = ∞

Querying the Erdős Number

Assume a database with schema:

Author(aid, name), **Paper**(pid, title), **Wrote**(aid, pid)

We can query the authors with EN $\leq$ 1 easily:

$$P := \pi_{pid}\left(\sigma_{name=\text{'Paul Erdos'}}(Author) \bowtie Write\right)$$ get the ids of Erdős' papers

$$Q := \pi_{aid}(P \bowtie Write)$$ get the authors of those papers

Can we also get the authors with EN = 1?

Querying the Erdős Number

Assume a database with schema:

Author(aid, name), **Paper**(pid, title), **Wrote**(aid, pid)

We can query the authors with EN $\leq$ 1 easily:

$$P := \pi_{pid}\left(\sigma_{name=\text{'Paul Erdos'}}(Author) \bowtie Write\right)$$ get the ids of Erdős' papers

$$Q := \pi_{aid}(P \bowtie Write)$$ get the authors of those papers

Can we also get the authors with EN = 1?

Yes – $Q - \pi_{aid}\,\sigma_{name=\text{'Paul Erdos'}}(Author)$

Querying the Erdős Number

Assume a database with schema:

Author(aid, name), **Paper**(pid, title), **Wrote**(aid, pid)

We can query the authors with EN ≤ 2 just as easily:

$P_0 := \pi_{pid}\left(\sigma_{name=\text{'Paul Erdos'}}(Author) \bowtie Write\right)$ get the ids of Erdős' papers

$Q_1 := \pi_{aid}(P_0 \bowtie Write)$ get the authors with EN at most 1

$P_1 := \pi_{pid}(Q_1 \bowtie Write)$ get their papers

$Q_2 := \pi_{aid}(P_1 \bowtie Write)$ and get those papers' coauthors

Querying the Erdős Number

Assume a database with schema:

Author(aid, name), **Paper**(pid, title), **Wrote**(aid, pid)

Let's be more ambitious. Can we write RA queries for the following questions:

- AIDs of authors with EN < ∞, i.e., those with finite EN?
- AIDs of authors with EN = ∞, i.e., those with no EN?

Querying the Erdős Number

Assume a database with schema:

Author(aid, name), **Paper**(pid, title), **Wrote**(aid, pid)

Let's be more ambitious. Can we write RA queries for the following questions:

- AIDs of authors with EN < ∞, i.e., those with finite EN?
- AIDs of authors with EN = ∞, i.e., those with no EN?

No

Equal expressive power also means that all languages that we've discussed so far share the same limitations!

🙁

Looking Forward

How do we know this?

How can we prove that there cannot be a RA query for these questions?

We use Codd's Theorem in combination with results from logic, e.g., Ehrenfeucht-Fraïsse Games or the Compactness Theorem.

Solutions

Are there query languages that can answer these queries?

Yes! Datalog, a prominent example of such languages will be the topic of the next lecture.