Updated 2/15/2023

Topic 1: Data models and query languages
Unit 4: Datalog
Lecture 8

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp23)

https://northeastern-datalab.github.io/cs7240/sp23/

2/3/2023

Where We Are

- Relational query languages we have seen so far:
  - SQL
  - Relational Calculus
  - Relational Algebra
- They can express the same class of relational queries (ignoring extensions, such as grouping, aggregates, or sorting)
  - How powerful are they? What is missing?

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

?

- Find all people who are friends with everyone who is not a friend of Bob

?

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

?

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

?

Source: Dan Suciu, CSE 554, 2011.

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

NO: needs higher math; not possible with RA
(unless we have access to a relation Prime(x)...)

- Find all people who are friends with everyone who is not a friend of Bob

?

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

?

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

?

Source: Dan Suciu, CSE 554, 2011.

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

  NO: needs higher math; not possible with RA (unless we have access to a relation Prime(x)...)

- Find all people who are friends with everyone who is not a friend of Bob

  YES: $\{x \mid \forall y.(\neg \text{Friend}(y, \text{'Bob'}) \Rightarrow \text{Friend}(x,y)\}$ DI?

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

  ?

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

  ?

Source: Dan Suciu, CSE 554, 2011.

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

  NO: needs higher math; not possible with RA
  (unless we have access to a relation Prime(x)...)

- Find all people who are friends with everyone who is not a friend of Bob

  YES: $\{x \mid \forall y.(\neg \text{Friend}(y, \text{'Bob'}) \Rightarrow \text{Friend}(x,y)\}$

  DI?

  $\{x \mid \text{Person}(x) \wedge \forall y.[\text{Person}(y) \wedge \neg \text{Friend}(y,\text{'Bob'}) \Rightarrow \text{Friend}(x,y)]\}$

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

  ?

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

  ?

Source: Dan Suciu, CSE 554, 2011.

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

  NO: needs higher math; not possible with RA
  (unless we have access to a relation Prime(x)...)

- Find all people who are friends with everyone who is not a friend of Bob

  YES: $\{x \mid \forall y.(\neg \text{Friend}(y, \text{'Bob'}) \Rightarrow \text{Friend}(x,y)\}$

  $\{x \mid \text{Person}(x) \wedge \forall y.[\text{Person}(y) \wedge \neg \text{Friend}(y,\text{'Bob'}) \Rightarrow \text{Friend}(x,y)]\}$

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

  NO: equivalent to 3-coloring; NP-complete

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

  ?

Source: Dan Suciu, CSE 554, 2011.

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

  NO: needs higher math; not possible with RA
  (unless we have access to a relation Prime(x)...)

- Find all people who are friends with everyone who is not a friend of Bob

  YES: $\{x \mid \forall y.(\neg Friend(y, \text{'Bob'}) \Rightarrow Friend(x,y)\}$

  $\{x \mid Person(x) \land \forall y.[Person(y) \land \neg Friend(y, \text{'Bob'}) \Rightarrow Friend(x,y)]\}$

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

  NO: equivalent to 3-coloring; NP-complete

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

  NO: recursive query; PTIME yet not expressible in RA

  Next: Datalog: extends monotone RA with recursion

Source: Dan Suciu, CSE 554, 2011.

Transitive closure [edit]

Although relational algebra seems powerful enough for most practical purposes, there are some simple and natural operators on relations that cannot be expressed by relational algebra. One of them is the transitive closure of a binary relation. Given a domain *D*, let binary relation *R* be a subset of *D*×*D*. The transitive closure R^+ of *R* is the smallest subset of *D*×*D* that contains *R* and satisfies the following condition:

$$\forall x \forall y \forall z \left((x,y) \in R^+ \land (y,z) \in R^+ \Rightarrow (x,z) \in R^+ \right)$$

It can be proved using the fact that there is no relational algebra expression *E*(*R*) taking *R* as a variable argument that produces R^+.[7]

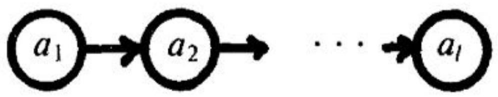
SQL however officially supports such fixpoint queries since 1999, and it had vendor-specific extensions in this direction well before that.

Appendix

In this appendix, we prove that the transitive closure of a relation cannot be couched as an expression of relational algebra.† It is interesting to note that both Bancilhon [B] and Paredaens [P] in essence characterize relational algebra as equivalent to the set of mappings obeying principle 2 with respect to an empty set of predicates. However, transitive closure obeys this principle. There is no contradiction. In [B,P] it is shown that for every relation r there is a relational algebra expression E such that $E(R)=R^+$, the transitive closure of R. What we show is that for no relational algebra expression E is $E(R)=R^+$ for all r.

Theorem 6. For an arbitrary binary relation R, there is no expression $E(R)$ in relational algebra equivalent to R^+, the transitive closure of R.

Suppose we have an expression $E(R)$ that is the transitive closure of R. Let $\Sigma_l = \{a_1, a_2, \ldots, a_l\}$ be a set of l arbitrary symbols. Let R_l be the finite relation $\{a_1 a_2, a_2 a_3, \ldots, a_{l-1} a_l\}$. R_l represents the graph



We shall show that, for any relational expression E, there is some value of l for which $E(R_l)$ is not R_l^+. In particu-

Appendix from: Aho, Ullman. "Universality of data retrieval languages". POPL 1979. https://doi.org/10.1145%2F567752.567763
Source: https://en.wikipedia.org/wiki/Relational_algebra#Transitive_closure
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Datalog

- Database query language designed in the 80's
- Simple, concise, elegant
  - "Clean" restriction of Prolog with DB access
  - Expressive & declarative:
    - Set-of-rules semantics
    - Independence of execution order
    - Invariance under logical equivalence
- Few open source implementations, mostly academic implementations
- Recently a hot topic, beyond databases:
  - network protocols, static program analysis, DB+ML

```
Path(x,y) :- Arc(x,y).
Path(x,z) :- Arc(x,y), Path(y,z).
InCycle(x) :- Path(x,x).
```

dlvsystem SPIN-OFF OF UNIVERSITY OF CALABRIA

LogicBlox

RelationalAI

Recursion with SQL server vs. Datalog

SQL

LISTING 4.7 Using Common Table Expressions for Recursive Operations

```
USE AdventureWorks;
WITH DirectReports (ManagerID, EmployeeID, EmployeeName, Title)
AS
(
-- Anchor member definition
  SELECT e.ManagerID, e.EmployeeID, c.FirstName + ' ' + c.LastName, e.Title
  FROM HumanResources.Employee AS e
  INNER JOIN Person.Contact as c
            ON e.ContactID = c.ContactID
  WHERE ManagerID IS NULL
  UNION ALL
-- Recursive member definition
  SELECT e.ManagerID, e.EmployeeID,c.FirstName + ' ' + c.LastName ,e.Title
  FROM HumanResources.Employee AS e
  INNER JOIN DirectReports AS d
    ON e.ManagerID = d.EmployeeID
  INNER JOIN Person.Contact as c
            ON e.ContactID = c.ContactID
)
-- Statement that executes the CTE
SELECT EmployeeID, EmployeeName, Title, ManagerID
FROM DirectReports
GO
```

Datalog

```
Manager(eid) :- Manages(_, eid)

DirectReports(eid, 0) :-
    Employee(eid), not Manager(eid)

DirectReports(eid, level+1) :-
    DirectReports(mid, level), Manages(mid, eid)
```

SQL Query vs. Datalog: which would you rather write?

Possible scribe: to fix that example ☺

Query on the left from Bieker, Lee. Mastering SQL server 2008. Example on the right by Dan Suciu

Smallest set of features that would make relational algebra Turing complete

Asked 8 years, 4 months ago Active 5 years, 5 months ago Viewed 296 times

4

You need just two things: new values and recursion/while.

New values means the ability to execute some external function that returns values that were not already to be found in the database. Obviously most implementations (including SQL) have that.

CTE = Common Table Expession = WITH clause

Recursion/while means the ability to execute a loop or iterative computation that may not terminate. The CTE RECURSIVE feature of SQL is one such.

SQL with CTE RECURSIVE is Turing Complete (without stored procedures).

See the Alice book http://webdam.inria.fr/Alice/ for a detailed treatment.

Share Cite Improve this answer Follow

answered Sep 1 2016 at 5:47

david.pfx
176 4

https://cs.stackexchange.com/questions/14694/smallest-set-of-features-that-would-make-relational-algebra-turing-complete

Jan Hidders, Database researcher
Answered 2 years ago · Author has **615** answers and **840K** answer views

Why is SQL not Turing complete?

Some variants of SQL, including some of the ISO standards, are actually Turing complete.

The most obvious example is SQL:1999 with the SQL/PSM extension, which adds stored procedures and therefore recursive functions and programming constructs that were intended to turn SQL into a programming language.

A less obvious example is SQL:2003 without stored procedures. It can be shown to be Turing complete using a clever combination of recursive queries (using Common Table Expressions) and Windowing, the first introduced in SQL:1999 and the latter since SQL:2003. See: http://assets.en.oreilly.com/1/event/27/High%20Performance%20SQL%20with%20PostgreSQL%20Presentation.pdf).

Nevertheless, it is true that the core of SQL was deliberately designed to be not Turing complete. The main reasons for this are:

1. By restricting the query language the programmer is encouraged to separate the computational task into a part that can be efficiently computed and optimised by the DBMS (namely the part that can be formulated in SQL) and a part that the programmer probably can better implement by themselves.
2. By restricting the query language to computations that always terminate and can be computed in polynomial time and logarithmic space, we can reduce the risk of burdening the database server with a workload that it cannot deal with.

1.4K views · View upvotes

Cyclic Tag System

Fun Snippets
Cyclic Tag System
Works with PostgreSQL 8.4
Written in
SQL
Depends on
Nothing

This SQL query (requires PostgreSQL 8.4) forms a cyclic tag system (wikipedia), which is sufficient to demonstrate that SQL is Turing-complete. It is written entirely in SQL:2003-conformant SQL.

Thanks to Andrew (RhodiumToad) Gierth, who came up with the concept and wrote the code.

The productions are encoded in the table "p" as follows:

```
"iter" is the production number;
"rnum" is the index of the bit;
"tag" is the bit value.
```

This example uses the productions:

```
110 01 0000
```

The initial state is encoded in the non-recursive union arm, in this case just '1'

The mod(r.iter, n) subexpression encodes the number of productions, which can be greater than the size of table "p", because empty productions are not included in the table.

Parameters:

```
the content of "p"
the content of the non-recursive branch
the 3 in mod(r.iter, 3)
```

"p" encodes the production rules; the non-recursive branch is the initial state, and the 3 is the number of rules

The result at each level is a bitstring encoded as 1 bit per row, with rnum as the index of the bit number.

At each iteration, bit 0 is removed, the remaining bits shifted up one, and if and only if bit 0 was a 1, the content of the current production rule is appended at the end of the string.

```
WITH RECURSIVE
p(iter,rnum,tag) AS (
    VALUES (0,0,1),(0,1,1),(0,2,0),
           (1,0,0),(1,1,1),
           (2,0,0),(2,1,0),(2,2,0),(2,3,0)
),
r(iter,rnum,tag) AS (
    VALUES (0,0,1)
UNION ALL
    SELECT r.iter+1,
           CASE
               WHEN r.rnum=0 THEN p.rnum + max(r.rnum) OVER ()
               ELSE r.rnum-1
           END,
           CASE
               WHEN r.rnum=0 THEN p.tag
               ELSE r.tag
           END
    FROM
          r
    LEFT JOIN p
        ON (r.rnum=0 and r.tag=1 and p.iter=mod(r.iter, 3))
    WHERE
          r.rnum>0
    OR p.iter IS NOT NULL
)
SELECT iter, rnum, tag
FROM r
ORDER BY iter, rnum;
```

https://www.quora.com/Why-is-relational-algebra-not-Turing-complete , https://wiki.postgresql.org/wiki/Cyclic_Tag_System , https://en.wikipedia.org/wiki/Tag_system#Cyclic_tag_systems

Cyclic tag systems [edit]

A cyclic tag system is a modification of the original tag system. The alphabet consists of only two symbols, **0** and **1**, and the production rules comprise a list of productions considered sequentially, cycling back to the beginning of the list after considering the "last" production on the list. For each production, the leftmost symbol of the word is examined—if the symbol is **1**, the current production is appended to the right end of the word; if the symbol is **0**, no characters are appended to the word; in either case, the leftmost symbol is then deleted. The system halts if and when the word becomes empty.

Example [edit]

```
Cyclic Tag System
 Productions: (010, 000, 1111)

Computation
 Initial Word: 11001
    Production          Word
    ----------          --------------
       010              11001
       000               1001010
       1111               001010000
       010                 01010000
       000                  1010000
       1111                  010000000
       010                    10000000
        .                       .
        .                       .
```

Cyclic tag systems were created by Matthew Cook and were used in Cook's demonstration that the Rule 110 cellular automaton is universal. A key part of the demonstration was that cyclic tag systems can emulate a Turing-complete class of tag systems.

Cyclic Tag System

Fun Snippets
Cyclic Tag System
Works with PostgreSQL 8.4
Written in
SQL
Depends on
Nothing

This SQL query (requires PostgreSQL 8.4) forms a cyclic tag system (wikipedia), which is sufficient to demonstrate that SQL is Turing-complete. It is written entirely in SQL:2003-conformant SQL.

Thanks to Andrew (RhodiumToad) Gierth, who came up with the concept and wrote the code.

The productions are encoded in the table "p" as follows:

```
"iter" is the production number;
"rnum" is the index of the bit;
"tag" is the bit value.
```

This example uses the productions:

```
110 01 0000
```

The initial state is encoded in the non-recursive union arm, in this case just '1'

The mod(r.iter, n) subexpression encodes the number of productions, which can be greater than the size of table "p", because empty productions are not included in the table.

Parameters:

```
the content of "p"
the content of the non-recursive branch
the 3 in mod(r.iter, 3)
```

"p" encodes the production rules; the non-recursive branch is the initial state, and the 3 is the number of rules

The result at each level is a bitstring encoded as 1 bit per row, with rnum as the index of the bit number.

At each iteration, bit 0 is removed, the remaining bits shifted up one, and if and only if bit 0 was a 1, the content of the current production rule is appended at the end of the string.

```
WITH RECURSIVE
p(iter,rnum,tag) AS (
    VALUES (0,0,1),(0,1,1),(0,2,0),
           (1,0,0),(1,1,1),
           (2,0,0),(2,1,0),(2,2,0),(2,3,0)
),
r(iter,rnum,tag) AS (
    VALUES (0,0,1)
UNION ALL
    SELECT r.iter+1,
           CASE
               WHEN r.rnum=0 THEN p.rnum + max(r.rnum) OVER ()
               ELSE r.rnum-1
           END,
           CASE
               WHEN r.rnum=0 THEN p.tag
               ELSE r.tag
           END
    FROM
        r
    LEFT JOIN p
      ON (r.rnum=0 and r.tag=1 and p.iter=mod(r.iter, 3))
    WHERE
        r.rnum>0
    OR p.iter IS NOT NULL
)
SELECT iter, rnum, tag
FROM r
ORDER BY iter, rnum;
```

https://www.quora.com/Why-is-relational-algebra-not-Turing-complete , https://wiki.postgresql.org/wiki/Cyclic_Tag_System , https://en.wikipedia.org/wiki/Tag_system#Cyclic_tag_systems

Query Language Design

Query language design is still a popular topic, especially for graphs. See e.g. https://www.tigergraph.com/gsql/

And the slides
https://courses.cs.washington.edu/courses/csed516/20au/lectures/lecture05-advanced-query-evaluation.pdf
from “DATA516/CSED516: Scalable Data Systems and Algorithms!” Dan Suciu
https://courses.cs.washington.edu/courses/csed516/20au/

Outline: T1-4: Datalog

- Datalog
  - Datalog rules
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Datalog$^{\neg}$: Negation, stratification
  - Datalog$^{\pm}$
  - Stable model semantics (Answer set programming)
  - Datalog vs. RA
  - Naive and Semi-naive evaluation (incl. Incremental View Maintenance)

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

```
Q1(y) :- Movie(x,y,z), z=1940.
```

?

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

?

```
Q3(f,l) :- Actor(z,f,l), Plays(z,x1), Movie(x1,y1,1910),
           Plays(z,x2), Movie(x2,y2,1940).
```

?

Datalog: Facts and Rules

Schema

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

```
Q1(y) :- Movie(x,y,z), z=1940.
```

Find movies from 1940

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

?

```
Q3(f,l) :- Actor(z,f,l), Plays(z,x1), Movie(x1,y1,1910),
                          Plays(z,x2), Movie(x2,y2,1940).
```

?

Examples by Dan Suciu

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

```
Q1(y) :- Movie(x,y,z), z=1940.
```

Find movies from 1940

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

Find actors who played in a movie before 1940

```
Q3(f,l) :- Actor(z,f,l), Plays(z,x1), Movie(x1,y1,1910),
           Plays(z,x2), Movie(x2,y2,1940).
```

?

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

```
Q1(y) :- Movie(x,y,z), z=1940.
```

Find movies from 1940

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

Find actors who played in a movie before 1940

```
Q3(f,l) :- Actor(z,f,l), Plays(z,x1), Movie(x1,y1,1910),
           Plays(z,x2), Movie(x2,y2,1940).
```

Find actors who played in a movie from 1910 and from 1940

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

```
Q1(y) :- Movie(x,y,z), z=1940.
```

Find movies from 1940

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

Find actors who played in a movie before 1940

```
Q3(f,l) :- Actor(z,f,l), Plays(z,x1), Movie(x1,y1,1910),
           Plays(z,x2), Movie(x2,y2,1940).
```

Find actors who played in a movie from 1910 ~~and~~ from 1940

OR

?

Examples by Dan Suciu

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

```
Q1(y) :- Movie(x,y,z), z=1940.
```

Find movies from 1940

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

Find actors who played in a movie before 1940

```
Q4(f,l) :- Actor(z,f,l), Plays(z,x1), Movie(x1,y1,1910).
Q4(f,l) :- Actor(z,f,l), Plays(z,x2), Movie(x2,y2,1940).
```

Find actors who played in a movie from 1910 ~~and~~ OR from 1940

Extensional Database (EDB) predicates: Actor, Plays, Movie

Intensional Database (IDB) predicates: Q1, Q2, Q3, Q4

Example with Souffle

Schema

Actor(id, fname, lname) Plays(aid, mid) Movie(id, name, year)

movie

command line w/ input/output directories

```
souffle -F. -D. movie.dl
```

movie.dl

```
.decl Actor(id:number, fname:symbol, lname:symbol)
.decl Plays(aid:number, mid:number)
.decl Movie(id:number, name:symbol, year:number)
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).

.decl Q2(fname:symbol, lname:symbol)
Q2(f,l) :- Actor(u,f,l), Plays(u,x), Movie(x,_,z), z<1940.
.output Q2
```

output

tab-separated output, filename: ".csv"

Q2.csv

Douglas	Fowley

For more help on Souffle, see: https://souffle-lang.github.io/simple
Datalog example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle

Syntax of rules

- evaluates to true when relation R_i contains the tuple described by $args_i$
- e.g. Actor(344759,"Douglas","Fowley") is true

$R_i(args_i)$: relational predicate with arguments (= atom)

arithmetic predicate

Q2(f,l) :- Actor(u,f,l), Casts(u,x), Movie(x,y,z), z<1940.

head

(or consequent)

single IDB subgoal

body

(or antecedent)

conjunction of subgoals

{f,l}: head variables

{u,x,y,z}: existential variables

Alternative notation: Q(args) <- R1(args) AND R2(args) .... / or variables begin with a capital, predicates with lower-case (problem: can't have "Boston")

Source: Dan Suciu, CSE 554, 2018.

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<1940.

Meaning of a Datalog rule is a logical statement:

?

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<1940.

Meaning of a Datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<1940 then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\text{Movie}(x,y,z) \land z<1940) \Rightarrow Q(y)]$$

logically equivalent to

?

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<1940.

Meaning of a Datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<1940 then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\text{Movie}(x,y,z) \wedge z<1940) \Rightarrow Q(y)]$$

logically equivalent to

$$\forall y\ [\exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940] \Rightarrow Q(y)\ ]$$

Thus, non-head variables are called "existential variables"

compare with RC

?

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<1940.

Meaning of a Datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<1940 then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\text{Movie}(x,y,z) \wedge z<1940) \Rightarrow Q(y)]$$

logically equivalent to

$$\forall y\ [\exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940] \Rightarrow Q(y)\ ]$$

Thus, non-head variables are called "existential variables"

We want the smallest set Q with this property (why?)

compare with RC

$$\{(y) \mid \exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940]\ \}$$

Syntactic Constraints

$Q(\mathbf{x})$:- $R_1(\mathbf{x}_1,\mathbf{y}_1),\ldots,R_m(\mathbf{x}_m,\mathbf{y}_m)$.

$\mathbf{x}_i \subseteq \mathbf{x}, \mathbf{y}_i \subseteq \mathbf{y}$

(bold = vector notation)

The rule stands for the following logical formula:

$$\forall \mathbf{x}[Q(\mathbf{x}) \Leftarrow \exists \mathbf{y}[R_1(\mathbf{x}_1,\mathbf{y}_1) \wedge \cdots \wedge R_m(\mathbf{x}_m,\mathbf{y}_m)]]$$

Recall we want the smallest set Q with this property

Two restrictions:

1. Safety: every head variable should occur in the body at least once

R(x,z) :- S(x,y), R(y,x).

X(z)

?

Syntactic Constraints

$Q(\mathbf{x})$:- $R_1(\mathbf{x}_1,\mathbf{y}_1),\ldots,R_m(\mathbf{x}_m,\mathbf{y}_m)$.

$\mathbf{x}_i \subseteq \mathbf{x}, \mathbf{y}_i \subseteq \mathbf{y}$

(bold = vector notation)

The rule stands for the following logical formula:

$$\forall \mathbf{x}[Q(\mathbf{x}) \Leftarrow \exists \mathbf{y}[R_1(\mathbf{x}_1,\mathbf{y}_1) \wedge \cdots \wedge R_m(\mathbf{x}_m,\mathbf{y}_m)]]$$

Two restrictions:

1. Safety: every head variable should occur in the body at least once

~~R(x,z) :- S(x,y), R(y,x).~~

forbidden rule: z not in body

2. The head predicate must be an IDB (Intensional) predicate
   (Body can include both EDBs and IDBs)

Arc(x,y) :- Arc(x,z),Arc(z,y).

?

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Syntactic Constraints

Q(**x**) :- $R_1(\mathbf{x}_1,\mathbf{y}_1),\ldots,R_m(\mathbf{x}_m,\mathbf{y}_m)$.

$\mathbf{x}_i \subseteq \mathbf{x}, \mathbf{y}_i \subseteq \mathbf{y}$

(bold = vector notation)

The rule stands for the following logical formula:

$$\forall \mathbf{x}\big[Q(\mathbf{x}) \Leftarrow \exists \mathbf{y}[R_1(\mathbf{x}_1,\mathbf{y}_1) \wedge \cdots \wedge R_m(\mathbf{x}_m,\mathbf{y}_m)]\big]$$

Two restrictions:

1. Safety: every head variable should occur in the body at least once

~~R(x,z) :- S(x,y), R(y,x).~~

forbidden rule: z not in body

2. The head predicate must be an IDB (Intensional) predicate
   (Body can include both EDBs and IDBs)

~~Arc(x,y) :- Arc(x,z),Arc(z,y).~~

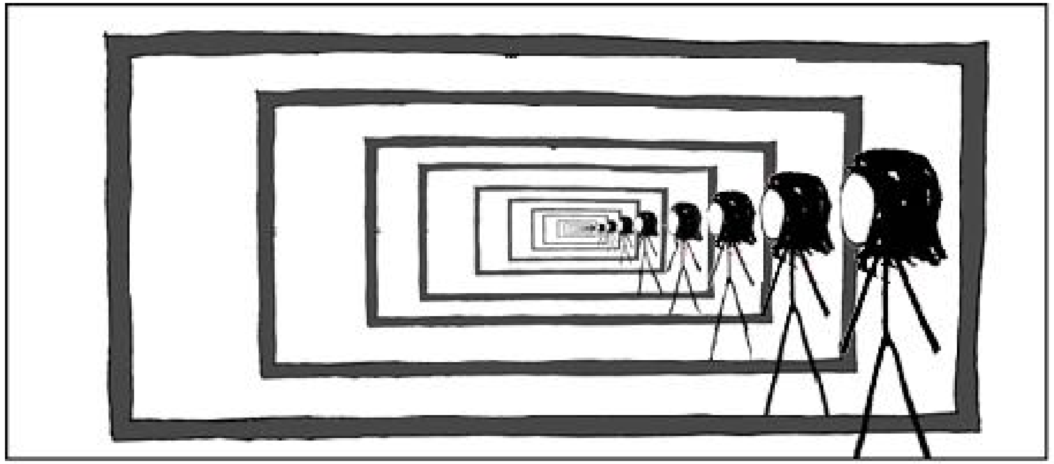
assuming Arc is EDB

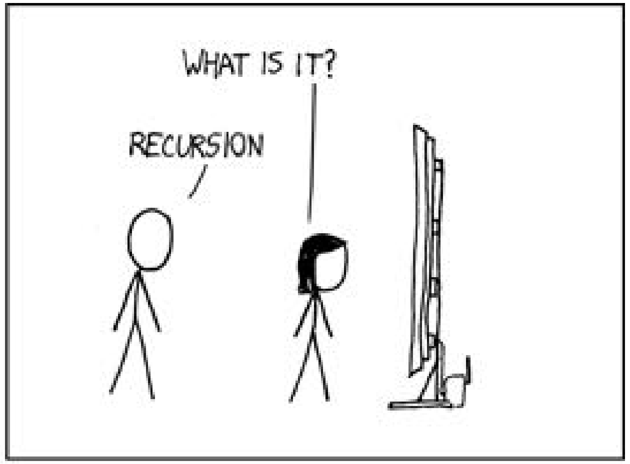
Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Outline: T1-4: Datalog

- Datalog
  - Datalog rules
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Datalog$^{\neg}$: Negation, stratification
  - Datalog$^{\pm}$
  - Stable model semantics (Answer set programming)
  - Datalog vs. RA
  - Naive and Semi-naive evaluation (incl. Incremental View Maintenance)

Recursion





Recursion occurs when a thing is defined in terms of itself (self-repetition).

Recursion and Iteration both repeatedly execute a set of instructions.

- Recursion (self-similarity) is when a statement in a function calls itself repeatedly.
- Iteration (repetition) is when a loop repeatedly executes until the controlling condition becomes false.

A datalog program consists of several rules:

- Usually there is one distinguished predicate that's the output
- Interestingly, rules may be recursive!

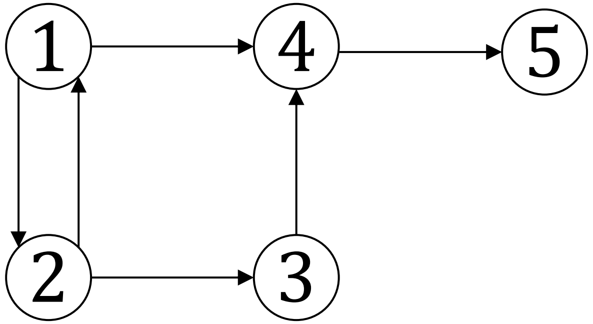
Figure Source: Fake XKCD: http://xkcdsw.com/1105

Example

EDB P(x,y) :- A(x,y).
IDB P(x,y) :- A(x,z), P(z,y).

recursion due to head in rule body

A(S,T)



What does this query compute?

?

A

1	2
2	1
2	3
1	4
3	4
4	5

Based upon an example by Dan Suciu from CSE 554, 2018.

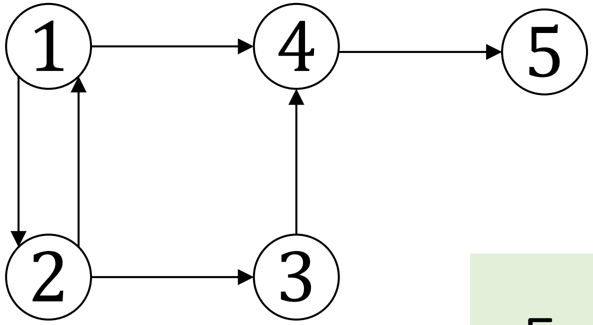
Example

A(S,T)

EDB P(x,y) :- A(x,y).
IDB P(x,y) :- A(x,z), P(z,y).

recursion due to head in rule body

Calculates all paths (transitive closure)



A

1	2
2	1
2	3
1	4
3	4
4	5

For all nodes x and y:
If there is an arc from x to y,
then there is a path from x to y.

For all nodes x, z, and y:
If there is an arc from x to z, and there is a path from z to y
then there is a path from x to y.

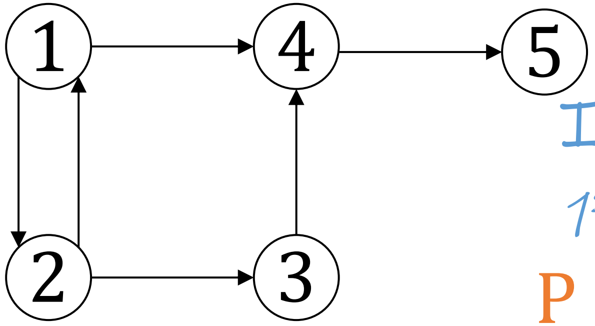
Based upon an example by Dan Suciu from CSE 554, 2018.

Example

A(S,T) 

EDB P(x,y) :- A(x,y).
IDB P(x,y) :- A(x,z), P(z,y).

recursion due to head in rule body



Initially: P is empty

1st iteration

P

?

A

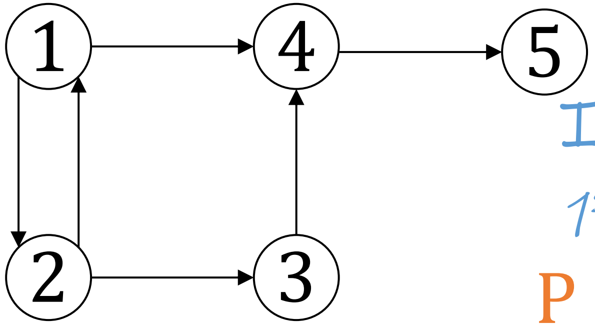
1	2
2	1
2	3
1	4
3	4
4	5

Example

A(S,T)

EDB P(x,y) :- A(x,y).
IDB P(x,y) :- A(x,z), P(z,y).

recursion due to head in rule body



Initially: P is empty

1st iteration

P

1	2
2	1
2	3
1	4
3	4
4	5

P=A from 1st rule

2nd iteration

P

?

A

1	2
2	1
2	3
1	4
3	4
4	5

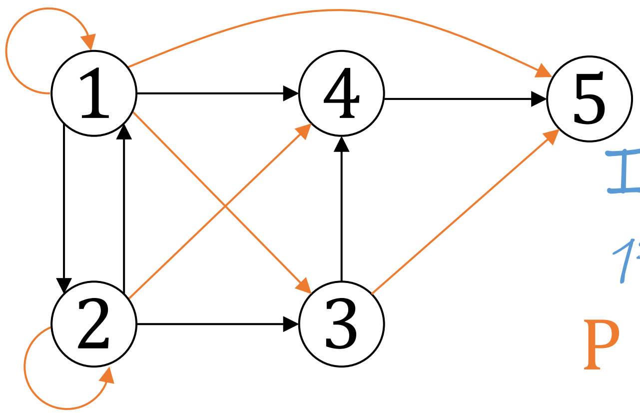
2nd rule generates nothing (because P is empty)

Example

A(S,T) 

EDB P(x,y) :- A(x,y).
IDB P(x,y) :- A(x,z), P(z,y).

recursion due to head in rule body



A

1	2
2	1
2	3
1	4
3	4
4	5

Initially: P is empty

1st iteration

P

1	2
2	1
2	3
1	4
3	4
4	5

P=A from 1st rule

2nd rule generates nothing (because P is empty)

2nd iteration

P

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

1st rule (first 6 rows); *2nd rule* (last 6 rows)

New facts from 2nd rule

Based upon an example by Dan Suciu from CSE 554, 2018.

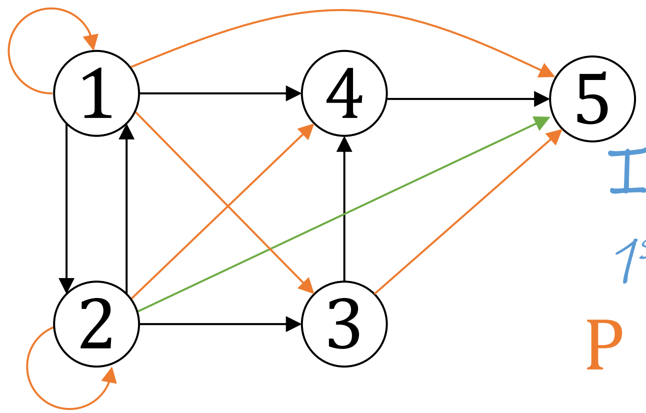
Example

A(S,T)

EDB $P(x,y)$:- $A(x,y)$.

IDB $P(x,y)$:- $A(x,z)$, $P(z,y)$.

recursion due to head in rule body



A

1	2
2	1
2	3
1	4
3	4
4	5

Initially: P is empty

recall set semantics!

1st iteration

P

1	2	P=A from 1st rule
2	1	
2	3	
1	4	
3	4	
4	5	

2nd rule generates nothing (because P is empty)

2nd iteration

P

1	2	1st rule
2	1	
2	3	
1	4	
3	4	
4	5	
1	1	2nd rule
2	2	
1	3	
2	4	
1	5	
3	5	

3rd iteration = 4th iteration (No new facts)

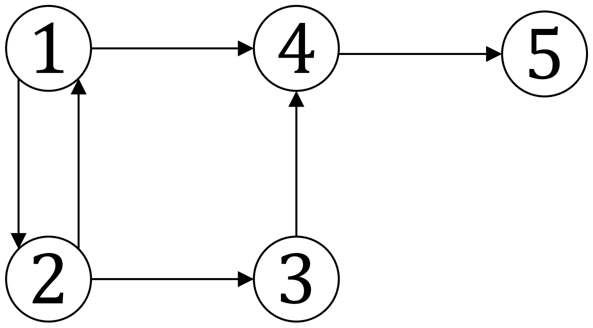
P

1	2	1st + 2nd rule
2	1	
2	3	1st rule
1	4	
3	4	
4	5	
1	1	2nd rule
2	2	
1	3	
2	4	
1	5	
3	5	
2	5	

New facts from 2nd rule

Example with Souffle Soufflé

A(S,T) graph1




```
souffle -F. -D. graph1.dl
```

graph1.dl

```
.decl A(x:number, y:number)
A(1,2).
A(2,1).
A(2,3).
A(1,4).
A(3,4).
A(4,5).

.decl P(x:number, y:number)
P(x, y) :- A(x, y).
P(x, y) :- A(x, z), P(z, y).

.output P
```

output

tab-separated, output filename: ".csv"

P.csv

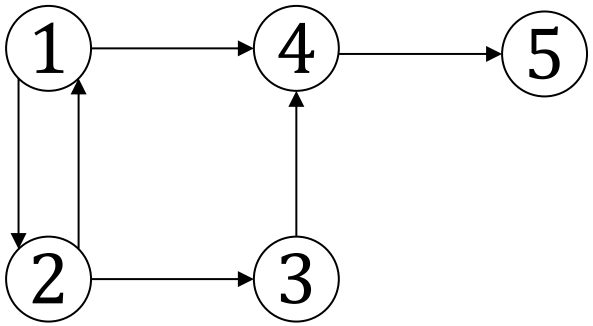
1	1
1	2
1	3
1	4
1	5
2	1
2	2
2	3
2	4
2	5
3	4
3	5
4	5

For more help on Souffle, see: https://souffle-lang.github.io/simple
Datalog example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle

Example with Souffle

A(S,T)

graph2



command line w/ input/output directories

```
souffle -F. -D. graph2.dl
```

datalog file

graph2.dl

```
.decl A(x:number, y:number)
.decl P(x:number, y:number)
.input A
.output P

P(x, y) :- A(x, y).
P(x, y) :- A(x, z), P(z, y).
```

input

output

A.facts

1	2
2	1
2	3
1	4
3	4
4	5

tab-separated, input filename: ".facts"

P.csv

1	1
1	2
1	3
1	4
1	5
2	1
2	2
2	3
2	4
2	5
3	4
3	5
4	5

tab-separated, output filename: ".csv"

For more help on Souffle, see: https://souffle-lang.github.io/simple
Datalog example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle