Updated 2/23/2022

Topic 1: Data models and query languages
Unit 4: Datalog
Lecture 08

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

https://northeastern-datalab.github.io/cs7240/sp22/

2/11/2022

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.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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

- 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.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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

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

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

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

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

?

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

?

Source: Dan Suciu, CSE 554, 2011.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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

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

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

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

- 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)

?

Source: Dan Suciu, CSE 554, 2011.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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

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

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

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

- 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 know expressible in RA

  Next: Datalog: extends monotone RA with recursion

Source: Dan Suciu, CSE 554, 2011.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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 $D \times D$. The transitive closure R^+ of R is the smallest subset of $D \times D$ that contains R and satisfies the following condition:

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

There is no relational algebra expression $E(R)$ taking R as a variable argument that produces R^+. This can be proved using the fact that, given a relational expression E for which it is claimed that $E(R) = R^+$, where R is a variable, we can always find an instance $\boldsymbol{r}$ of R (and a corresponding domain $\mathbf{d}$) such that $E(\boldsymbol{r}) \neq \boldsymbol{r}^+$.[14]

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

Source: https://en.wikipedia.org/wiki/Relational_algebra#Transitive_closure

Datalog

E(S,T)

501

- 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

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

- Few open source implementations, mostly academic implementations
- Recently a hot topic, beyond databases:
  - network protocols, static program analysis, DB+ML

dlvsystem SPIN-OFF OF UNIVERSITY OF CALABRIA

LogicBlox

Based on slides by Dan Suciu
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Recursion with SQL server vs. Datalog

Proprietary SQL

LISTING 4.7 Using Common Table Expressions for Recursive Operations

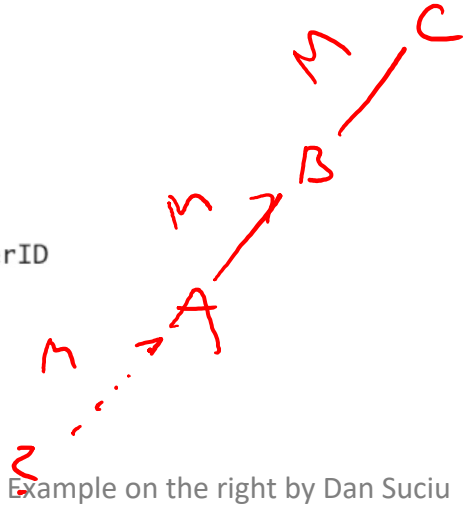
```
USE AdventureWorks;
WITH DirectReports (ManagerID, EmployeeID, EmployeeName, Title)
AS
(
-- Anchor member definition
  SELECT e.ManagerID, e.EmployeeID, c.FirstName + ' ' + c.LastName, e.Title
  FROM HumanResources.Employee AS e
  INNER JOIN Person.Contact as c
            ON e.ContactID = c.ContactID
  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?

Query on the left from Bieker, Lee. Mastering SQL server 2008. Example on the right by Dan Suciu Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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 Expression = 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
  - 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').
Casts(344759, 7909).
Casts(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).

?

Examples by Dan Suciu
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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').
Casts(344759, 7909).
Casts(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).

?

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').
Casts(344759, 7909).
Casts(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).

?

Examples by Dan Suciu
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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').
Casts(344759, 7909).
Casts(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

?

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').
Casts(344759, 7909).
Casts(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 from 1940

OR

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

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

Examples by Dan Suciu
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/