Introduction to Data Structure (Data Management) Lecture 9

Felipe P. Vista IV

INTRO TO DATA STRUCTURE

DATALOG
(CH 5.3 & 5.4)

Reminder

- Everybody, make sure that your name in ZOOM is in the following format:
  - Ex: 202054321 Juan Dela Cruz

  Not changing your name to this format

  * you might be marked Absent * → absent?

- Our class will still be **online/Zoom** starting Monday 19 Oct 2020

- Datalog

- Relational Algebra vs Datalog

What is Datalog?

SQL

- Another query language for relational model
  - Simple and elegant
  - Initially designed for recursive queries
    - Increased interest due to recursive analytics
  - Some companies use it (or derivatives) for data analytics, i.e. LogiQL (**Logi**c **Q**uery **L**anguage) from LogicBox
    - *Predict consumer demand (retail)*
    - *Optimize supply chain (retail, manufacturing, distribution)*
    - *Select optimal assortments (retail)*

What is Datalog?

- Another query language for relational model
  - Simple and elegant
  - Initially designed for recursive queries
    - Increased interest due to recursive analytics
  - Some companies use it (or derivatives) for data analytics, i.e. LogiQL (**Logic Query Language**) from LogicBox
    - *Predict consumer demand (retail)*
    - *Optimize supply chain (retail, manufacturing, distribution)*
    - *Select optimal assortments (retail)*
- We take up only non-recursive Datalog, and add negation (NOT)

Why Learn About Datalog?

- Datalog can be translated to SQL
  - Help express complex queries

Why Learn About Datalog?

```
USE AdventureWorks2008R2;
GO
WITH DirectReports (ManagerID, EmployeeID, Title, DeptID, Level)
AS
(
-- Anchor member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        0 AS Level
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    WHERE ManagerID IS NULL
    UNION ALL
-- Recursive member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        Level + 1
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    INNER JOIN DirectReports AS d
        ON e.ManagerID = d.EmployeeID
)
-- Statement that executes the CTE
SELECT ManagerID, EmployeeID, Title, DeptID, Level
FROM DirectReports
INNER JOIN HumanResources.Department AS dp
    ON DirectReports.DeptID = dp.DepartmentID
WHERE dp.GroupName = N'Sales and Marketing' OR Level = 0;
GO
```

Why Learn About Datalog?

```
USE AdventureWorks2008R2;
GO
WITH DirectReports (ManagerID, EmployeeID, Title, DeptID, Level)
AS
(
-- Anchor member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        0 AS Level
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    WHERE ManagerID IS NULL
    UNION ALL
-- Recursive member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        Level + 1
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    INNER JOIN DirectReports AS d
        ON e.ManagerID = d.EmployeeID
)
-- Statement that executes the CTE
SELECT ManagerID, EmployeeID, Title, DeptID, Level
FROM DirectReports
INNER JOIN HumanResources.Department AS dp
    ON DirectReports.DeptID = dp.DepartmentID
WHERE dp.GroupName = N'Sales and Marketing' OR Level = 0;
GO
```

```
DirectReports(eID,0) ←
    Employee(eID) AND
    NOT Manages(_,eID)
DirectReports(eID,level+1) ←
    DirectReports(mid,level) AND
    Manages(mid, eID)
```

Why Learn About Datalog?

```
USE AdventureWorks2008R2;
GO
WITH DirectReports (ManagerID, EmployeeID, Title, DeptID, Level)
AS
(
-- Anchor member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        0 AS Level
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    WHERE ManagerID IS NULL
    UNION ALL
-- Recursive member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        Level + 1
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    INNER JOIN DirectReports AS d
        ON e.ManagerID = d.EmployeeID
)
-- Statement that executes the CTE
SELECT ManagerID, EmployeeID, Title, DeptID, Level
FROM DirectReports
INNER JOIN HumanResources.Department AS dp
    ON DirectReports.DeptID = dp.DepartmentID
WHERE dp.GroupName = N'Sales and Marketing' OR Level = 0;
GO
```

```
DirectReports(eID,0) ←
   Employee(eID) AND
   NOT Manages(_,eID)
DirectReports(eID,level+1) ←
   DirectReports(mid,level) AND
   Manages(mid, eID)
DirectReports(eID,0) :-
   Employee(eID),
   NOT Manages(_,eID)
DirectReports(eID,level+1) :-
   DirectReports(mid,level),
   Manages(mid, eID)
```

Why Learn About Datalog?

SQL

```
USE AdventureWorks2008R2;
GO
WITH DirectReports (ManagerID, EmployeeID, Title, DeptID, Level)
AS
(
-- Anchor member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        0 AS Level
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    WHERE ManagerID IS NULL
    UNION ALL
-- Recursive member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        Level + 1
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    INNER JOIN DirectReports AS d
        ON e.ManagerID = d.EmployeeID
)
-- Statement that executes the CTE
SELECT ManagerID, EmployeeID, Title, DeptID, Level
FROM DirectReports
INNER JOIN HumanResources.Department AS dp
    ON DirectReports.DeptID = dp.DepartmentID
WHERE dp.GroupName = N'Sales and Marketing' OR Level = 0;
GO
```

Datalog

```
DirectReports(eID,0) ←
   Employee(eID) AND
   NOT Manages(_,eID)
DirectReports(eID,level+1) ←
   DirectReports(mid,level) AND
   Manages(mid, eID)
```

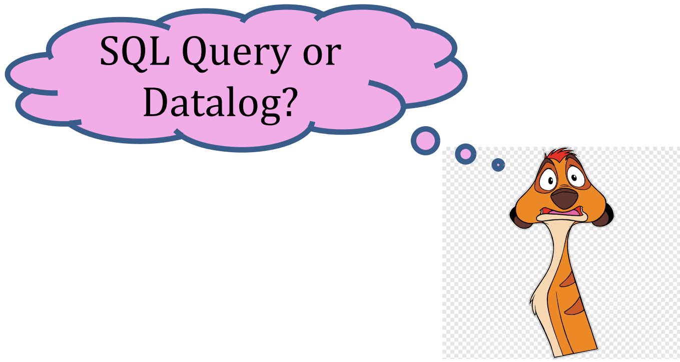
```
DirectReports(eID,0) :-
   Employee(eID),      AND
   NOT Manages(_,eID)
DirectReports(eID,level+1) :-
   DirectReports(mid,level),
   Manages(mid, eID)
```

Why Learn About Datalog?

```
USE AdventureWorks2008R2;
GO
WITH DirectReports (ManagerID, EmployeeID, Title, DeptID, Level)
AS
(
-- Anchor member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        0 AS Level
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    WHERE ManagerID IS NULL
    UNION ALL
-- Recursive member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        Level + 1
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    INNER JOIN DirectReports AS d
        ON e.ManagerID = d.EmployeeID
)
-- Statement that executes the CTE
SELECT ManagerID, EmployeeID, Title, DeptID, Level
FROM DirectReports
INNER JOIN HumanResources.Department AS dp
    ON DirectReports.DeptID = dp.DepartmentID
WHERE dp.GroupName = N'Sales and Marketing' OR Level = 0;
GO
```

```
DirectReports(eID,0) ←
    Employee(eID) AND
    NOT Manages(_,eID)
DirectReports(eID,level+1) ←
    DirectReports(mid,level) AND
    Manages(mid, eID)
```

```
DirectReports(eID,0) :-
    Employee(eID),
    NOT Manages(_,eID)
DirectReports(eID,level+1) :-
    DirectReports(mid,level),
    Manages(mid, eID)
```



Why Learn About Datalog?

- Datalog can be translated to SQL
  - Help express complex queries

Why Learn About Datalog?

- Datalog can be translated to SQL
  - Help express complex queries
- Increased interest in Datalog due recursive analytics

Why Learn About Datalog?

- Datalog can be translated to SQL
  - Help express complex queries
- Increased interest in Datalog due recursive analytics
- A query language closest to mathematical logic
  - Good language to reason about query properties
  - Can show:
    - Non-recursive Datalog & RA have same power
    - Recursive Datalog more powerful than RA
    - Extended RA & SQL92 more powerful than Datalog

Power

NRD = RA

RD > RA

D < ERA/SQL92

Some History

- Early database history
  - 60's: network data models
  - 70's: relational DBMS
  - 80's: OO-DBMS (Object oriented)

myFile.txt

NAME

Some History

- Early database history
  - 60's: network data models
  - 70's: relational DBMS
  - 80's: OO-DBMS (Object oriented)

- Ullman (1988) predicted KBMS will replace DBMS, the way it replaced what came before it
  - Knowledge Base Management System
  - Integration of DB technology and AI (data and logic/inferences)

Some History

- Early database history
  - 60's: network data models
  - 70's: relational DBMS
  - 80's: OO-DBMS (Object oriented)

Relational DBMSs still dominate

- Ullman (1988) predicted KBMS will replace DBMS, the way it replaced what came before it
  - Knowledge Base Management System
  - Integration of DB technology and AI (data and logic/inferences)

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Datalog: Facts & Rules

record/row

Facts = tuples in the DB

```
Actor(344759, 'Jenny', 'Divan').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night at Grizz', 2000).
Movie(29000, 'Janjer', 2020).
Movie(29445, 'Ohlala', 2020).
```

Rules = queries

Actor

pid	fname	lname
344759	Jenny	Divan

Casts

PID	MID

Movie

MID	NAME	YR

S M

Find movies made in 2020

S M.name F Movie
W/ M.Yr = 2020

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

x y z

Datalog: Facts & Rules

Facts = tuples in the DB

```
Actor(344759, 'Jenny', 'Divan').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night at Grizz', 2000).
Movie(29000, 'Janjer', 2020).
Movie(29445, 'Ohlala', 2020).
```

Rules = queries

```
Q1(y)← Movie(x, y, 2020).
```

Find movies made in 2020

→ Q1

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Datalog: Facts & Rules

Facts = tuples in the DB

```
Actor(344759, 'Jenny', 'Divan').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night at Grizz', 2000).
Movie(29000, 'Janjer', 2020).
Movie(29445, 'Ohlala', 2020).
```

Rules = queries

```
Q1(y)← Movie(x,y,2020).
```

Find artists who acted in movies made in 2020

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Datalog: Facts & Rules

R ⋈ S → equi join

Facts = tuples in the DB

```
Actor(344759, 'Jenny', 'Divan').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night at Grizz', 2000).
Movie(29000, 'Janjer', 2020).
Movie(29445, 'Ohlala', 2020).
```

Rules = queries

```
Q1(y)← Movie(x,y,2020).
```

```
Q2(f,l)← Actor(z,f,l) AND
         Casts(z,x) AND
         Movie(x,y,2020).
```

Find artists who acted in movies made in 2020

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Datalog: Facts & Rules

Facts = tuples in the DB

```
Actor(344759, 'Jenny', 'Divan').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night at Grizz', 2000).
Movie(29000, 'Janjer', 2020).
Movie(29445, 'Ohlala', 2020).
```

Rules = queries

```
Q1(y)← Movie(x,y,2020).
```

```
Q2(f,l)← Actor(z,f,l) AND
         Casts(z,x) AND
         Movie(x,y,2020).
```

Find artists who acted in a movie in 2020 & in one in 2000.

?

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Datalog: Facts & Rules

Facts = tuples in the DB

```
Actor(344759, 'Jenny', 'Divan').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night at Grizz', 2000).
Movie(29000, 'Janjer', 2020).
Movie(29445, 'Ohlala', 2020).
```

Rules = queries

```
Q1(y)← Movie(x,y,2020).
```

```
Q2(f,l)← Actor(z,f,l) AND
         Casts(z,x) AND
         Movie(x,y,2020).
```

```
Q3(f,l)← Actor(z,f,l) AND
   2020  Casts(z,x1) AND
         Movie(x1,y1,2020) AND
   2000  Casts(z,x2) AND
         Movie(x2,y2,2000).
```

Find artists who acted in a movie in 2020 & in one in 2000.

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Datalog: Facts & Rules

Facts = tuples in the DB

```
Actor(344759, 'Jenny', 'Divan').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night at Grizz', 2000).
Movie(29000, 'Janjer', 2020).
Movie(29445, 'Ohlala', 2020).
```

Rules = queries

```
Q1(y)← Movie(x,y,2020).
```

```
Q2(f,l)← Actor(z,f,l) AND
         Casts(z,x) AND
         Movie(x,y,2020).
```

```
Q3(f,l)← Actor(z,f,l) AND
         Casts(z,x1) AND
         Movie(x1,y1,2020)AND
         Casts(z,x2) AND
         Movie(x2,y2,2000).
```

Extensional Database Predicates (EDB) = Actor, Casts, Movie

Intensional Database Predicates (IDB) = Q1, Q1, Q3

Datalog: Terminology

$$\underbrace{Q2(f, l)}_{\text{head}} \leftarrow \underbrace{Actor(z, f, l)\ \text{AND}\ Casts(z, x)\ \text{AND}\ Movie(x, y, 2020)}_{\text{body}}$$

z = 2020

R P

"IF"

Datalog: Terminology

Body

$$\overbrace{\text{Q2(f, l)}}^{\text{head}} \leftarrow \overbrace{\overbrace{\text{Actor(z, f, l)}}^{\text{atom}} \text{ AND } \overbrace{\text{Casts(z, x)}}^{\text{atom}} \text{ AND } \overbrace{\text{Movie(x, y, 2020)}}^{\text{atom (subgoal)}}}^{\text{body}}$$

Datalog: Terminology

$$\underbrace{Q2(f, l)}_{\text{head}} \leftarrow \underbrace{\underbrace{Actor(z, f, l)}_{\text{atom}} \text{ AND } \underbrace{Casts(z, x)}_{\text{atom}} \text{ AND } \underbrace{Movie(x, y, 2020)}_{\text{atom (subgoal)}}}_{\text{body}}$$

f, l = head variables   to also body

x, y, z = existential variables   ≠ head

Datalog: More Terminology

ATOM ATOM

Q(args) ← R1(args) AND R2(args) AND …

* $R_i(args_i)$ is called an atom, or a relational predicate

Datalog: More Terminology

$$Q(args) \leftarrow R1(args) \text{ AND } R2(args) \text{ AND } \ldots$$

* $R_i(args_i)$ is called an atom, or a relational predicate
* $R_i(args_i)$ evaluates to TRUE when relation R_i contains the tuples described by the $args_i$
  - Ex: Actor(344759, 'Jenny', 'Divan')

Datalog: More Terminology

$$Q(args) \leftarrow R1(args) \text{ AND } R2(args) \text{ AND } \ldots$$

* $R_i(args_i)$ is called an atom, or a relational predicate
* $R_i(args_i)$ evaluates to TRUE when relation R_i contains the tuples described by the $args_i$
  - Ex: Actor(344759, ‘Jenny’, ’Divan’)

→ T/F

* In addition to relational predicates, we can also have arithmetic predicates
  - Ex: z = 2020

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Semantics

- Meaning of a Datalog rule = a logical statement *

$$\boxed{Q1(y) \leftarrow Movie(x, y, z) \text{ AND } z=2020}$$

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Semantics

- Meaning of a Datalog rule = a logical statement

$$\boxed{Q1(y) \leftarrow Movie(x, y, z) \text{ AND } z=2020}$$

- Means:
  - $\forall x.\ \forall y.\ \forall z.\ [(Movie(x, y, z) \text{ and } z=2020) \Longrightarrow Q1(y)]$
  - and Q1 is the smallest relation that has this property

* ∀ = for all
* ∃ = there exists

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Semantics

- Meaning of a Datalog rule = a logical statement

$$\boxed{Q1(y) \leftarrow Movie(x, y, z) \text{ AND } z=2020}$$

head

∃ ∀

- Means:
  - $\forall x.\ \forall y.\ \forall z.\ [(Movie(x, y, z) \text{ and } z=2020) \Longrightarrow Q1(y)]$
  - and Q1 is the smallest relation that has this property

* ∀ = for all
* ∃ = there exists

- Note, logically equivalent to:
  - $\forall y.\ [(\exists x.\ \exists z\ Movie(x, y, z) \text{ and } z=2020 \Longrightarrow Q1(y)]$
  - That's why vars not in head are called "existential variables"

Body

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Datalog program

- A Datalog program is a collection of one or more rules.

Each **rule** expresses the idea that, from certain combinations of tuples in certain relations, we may **infer** that some other tuple must be in some other relation or in the query answer.

assume
understand
get info.

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Datalog program

- A Datalog program is a collection of one or more rules.

> Each **rule** expresses the idea that, from certain combinations of tuples in certain relations, we may **infer** that some other tuple must be in some other relation or in the query answer.

- Example: Find all actors with 'Khan', numbering ≤ 2

```
B0(x)  ← Actor(x,'Khan','Boy')
B1(x)  ← Actor(x,f,l) AND Casts(x,z) AND Casts(y,z) AND B0(y)
B2(x)  ← Actor(x,f,l) AND Casts(x,z) AND Casts(y,z) AND B1(y)
Q4(x)  ← B0(x)
Q4(x)  ← B1(x)
Q4(x)  ← B2(x)
```

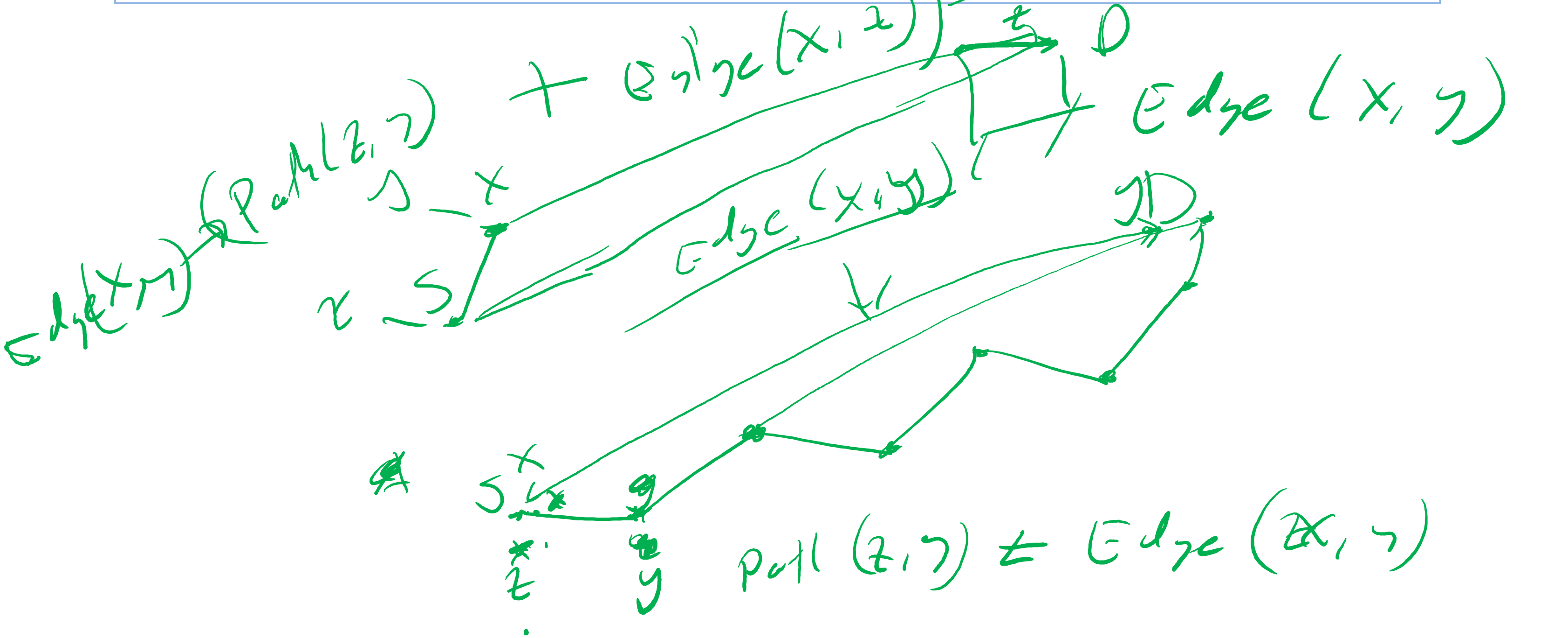
Note: Q4 means the union of B0, B1, and B2

Recursive Datalog

- In Datalog, rules can be recursive

```
(1) Path(x, y) ← Edge(x, y).
(2) Path(x, y) ← Edge(x, z) AND Path(z, y).
```

Path (z, y)

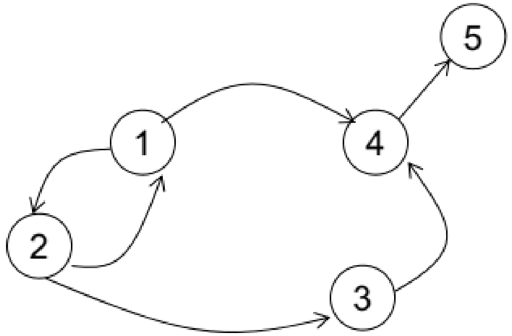


Recursive Datalog

- In Datalog, rules can be recursive

```
(1) Path(x, y) ← Edge(x, y).
(2) Path(x, y) ← Edge(x, z) AND Path(z, y).
```

- We focus on non-recursive Datalog



Edge: encodes a graph
Path: finds all paths

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Datalog with Negation

- Find all actors who do have a ‘Khan’ , numbering < 2

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Datalog with Negation

not

- Find all actors who do have a ‘Khan’ , numbering < 2

```
B0(x) ← Actor(x,'Khan','Boy')
B1(x) ← Actor(x,f,l) AND Casts(x,z) AND Casts(y,z) AND B0(y)
Q6(x) ← Actor(x,f,l) AND NOT B0(x) AND NOT B1(x)
```

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Safe Datalog Rules

- Here are examples of “unsafe” Datalog rules. Why unsafe?

H B

```
U1(x, y) ← Movie(x, z, 2014) AND y > 2000
```

```
U1(x, y) ← Movie(x, z, 2014) AND NOT Casts(u, x)
```

```
Actor(pid, fname, lname)
Casts(pid, mid)
Movie(mid, name, year)
```

Safe Datalog Rules

- Here are examples of “unsafe” Datalog rules. Why unsafe?

```
U1(x, y) ← Movie(x, z, 2014) AND y > 2000
```

```
U1(x, y) ← Movie(x, z, 2014) AND NOT Casts(u, x)
```

A Datalog rule is safe if every variable appears in some positive relational form

Datalog vs. Relational Algebra

- Every expression in standard relational algebra can be expressed as a Datalog query

Datalog vs. Relational Algebra

- Every expression in standard relational algebra can be expressed as a Datalog query ERA
- But operations in the extended algebra (grouping, aggregation, & sorting) have no corresponding features in the version of Datalog that we are discussing

Datalog vs. Relational Algebra

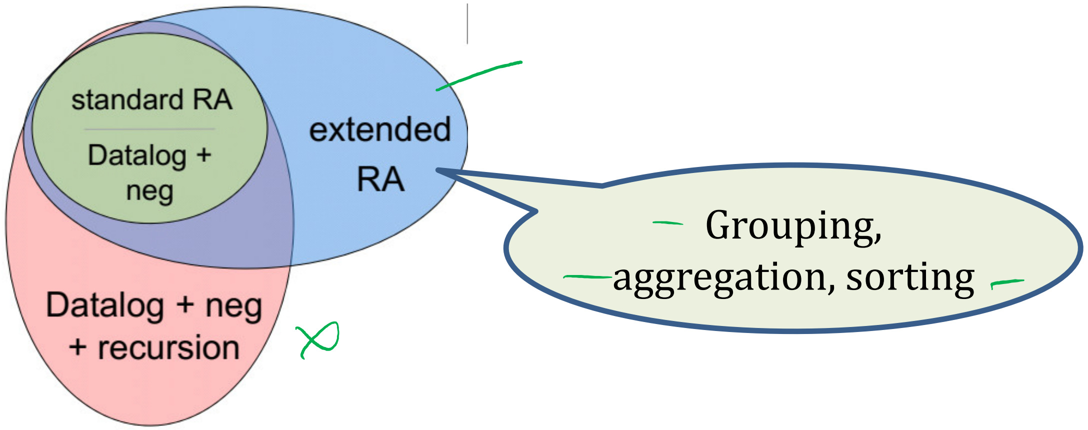
- Every expression in standard relational algebra can be expressed as a Datalog query
- But operations in the extended algebra (grouping, aggregation, & sorting) have no corresponding features in the version of Datalog that we are discussing
- Similarly, Datalog can express recursion, which relational algebra cannot

ERA → X DLog

DLog (recursive) RA X

Datalog vs. Relational Algebra

- Every expression in standard relational algebra can be expressed as a Datalog query
- But operations in the extended algebra (grouping, aggregation, & sorting) have no corresponding features in the version of Datalog that we are discussing
- Similarly, Datalog can express recursion, which relational algebra cannot



RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

OR → (+)

Union: R(A, B, C) ∪ S(D, E, F)

Q1 ← RULE1 ∪
Q1 ← RULE2

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

OR

Union: R(A, B, C) ∪ S(D, E, F)

IF

D. Log

```
U(x, y, z) ← R(x, y, z)
U(x, y, z) ← S(x, y, z)
```

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

~~∪ uni~~ ∪ - union

???

Intersection: R(A, B, C) ∩ S(D, E, F)

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

(x∧y)

Intersection: R(A, B, C) ∩ S(D, E, F)

x y z x y z

```
I(x, y, z) ← R(x, y, z) AND S(x, y, z))
```

x y z

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

- Selection: $\sigma_{x > 100 \text{ AND } y=\text{'some string'}}$ (R)

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

•Selection: $\sigma_{x > 100 \text{ AND } y=\text{'some string'}}$ (R)

```
L(x, y, z) ← R(x, y, z) AND x>100 AND
                 y='some string'
```

S R.A, R.B, R.E

F R

W x >100 AND y = 'some string';

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

•Selection: $\sigma_{x > 100 \text{ AND } y=\text{'some string'}}$ (R) AND z < 30

```
L(x, y, z) ← R(x, y, z) AND x>100 AND
                 y='some string' AND z < 30
```

AND/(+)

•Selection: $\sigma_{x > 100 \text{ OR } y=\text{'some string'}}$ (R) ~~OR z < 30~~

x>100 →
y=' ' →

```
L(x, y, z) ← R(x, y, z) AND x>100
L(x, y, z) ← R(x, y, z) AND y='some string'
L(x, y, z) ← R(x, y, z) AND z<30
```

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

•Selection: $\sigma_{x > 100 \text{ AND } y=\text{'some string'}}(R)$

```
L(x, y, z) ← R(x, y, z) AND x>100 AND
                y='some string'
```

•Selection: $\sigma_{x > 100 \text{ OR } y=\text{'some string'}}(R)$

```
L(x, y, z) ← R(x, y, z) AND x>100
L(x, y, z) ← R(x, y, z) AND y='some string'
```

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

Equi-join: $R \bowtie_{R.A.=S.D.\ AND\ R.B.=S.E.} S$ → R A

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

x y z
u v w
3
3
6 44

Equi-join: R ⋈$_{R.A.=S.D.\ AND\ R.B.=S.E.}$ S

```
J(x, y, z, u, v, w) ← R(x, y, z) AND S(u, v, w) AND
                       x=u AND y=v
```

R.A.=S.D. R.B.=S.E.

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

Equi-join: $R \bowtie_{R.A.=S.D.\ AND\ R.B.=S.E.} S$

```
J(x, y, z, u, v, w) ← R(x, y, z) AND S(u, v, w) AND
                     x=u AND y=v
```

~~J(x, y, z, w) ← R(x, y, z) AND S(x, y, w)~~

RA to Datalog Examples

Schema for given examples →

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

Projection: $\pi_{x}(R)$

$\boxed{P(x) \leftarrow R(x, y, z)}$

$\pi_y(R)$?

$P(y) \leftarrow R(x, y, z)$

$\pi_{x,z}(R)$

$P(x, z) \leftarrow R(x, y, z)$

$\pi_y(T)$

$P(y) \leftarrow T(x, y)$

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

Projection: $\pi_x(R)$

```
P(x) ← R(x, y, z)
```

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

negate → NOT

To express set difference R – S, we add negation

R-S = R NOT S

N(x, y, z) ← R(x, y, z) AND NOT S(x, y, z)

RA to Datalog Examples

Schema for given examples

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

To express set difference R – S, we add negation

```
D(x, y, z) ← R(x, y, z) AND
             NOT S(x, y, z)
```

D(e, j, n) ← R(e, j, n) AND NOT S(e, j, n)

=? ~~D(e, j, n) ← R(e, j, n) AND NOT S(x, y, z)~~

D(e, j, n) ← R(e, j, n) AND NOT (j, e, n)

Examples

Schema

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

B < 3

Translate $\pi_A(\sigma_{B=3}(R))$

$T(x) \leftarrow R(x, y, z)$ AND $R(x, 3, z)$

$T(x) \leftarrow R(x, 3, z)$

Examples

Schema

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

Translate $\pi_A(\sigma_{B=3}(R))$

R(x, 3, z)

```
B(a, b, c) ← R(a, b, c) AND b=3
```

Examples

Schema

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

Translate $\pi_A(\sigma_{B=3}(R))$

```
B(a, b, c) ← R(a, b, c) AND b=3
```

=

```
A(a) ← R(a, 3, _)
```

= A(a) ← R(a, 3, c)

Note: Underscore "_" represents an "anonymous variable", a variable that appears only once

Examples

Schema

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

Translate $\pi_A(\sigma_{B=3}(R) \bowtie_{R.A.=S.D.} \sigma_{E=5}(S))$

T(A) ← R(a, 3, _) → S(d, 5, _)

AND a = d

Examples

Schema

```
R(A, B, C)
S(D, E, F)
T(G, H)
```

Translate $\pi_A(\sigma_{B=3}(R) \bowtie_{R.A.=S.D.} \sigma_{E=5}(S))$

```
A(a) ← R(a, 3, _) AND S(a, 5, _)
```

Note: Underscore "_" represents an "anonymous variable", a variable that appears only once

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

1. Find Joe's friends and friends of Joe's friends.

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

1. Find Joe's friends and friends of Joe's friends.

JF

JF-F's

```
A(x) ← Friend('Joe', x)
A(x) ← Friend('Joe', z) AND Friend(z, x)
```

Joe → John → ?

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

2. Find all of Joe's friends who do not have any friends except for Joe:

<u>JF's</u>

$F_1 = J$ & $1F$

$F_2 = J$ $\neq$?

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

2. Find all of Joe's friends who do not have any friends except for Joe:

- NonAns(x): all people (of Joe's friends) who have some friends who are not Joe

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

2. Find all of Joe's friends who do not have any friends except for Joe:

- NonAns(x): all people (of Joe's friends) who have some friends who are not Joe

```
JoeFriends(x) ← Friend('Joe', x)
NonAns(x) ← Friend(y, x) AND y!='Joe'
A(x) ← JoeFriends(x) AND NOT NonAns(x)
```

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

3. Find all people such that all their enemies' enemies are their friends:

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

3. Find all people such that all their enemies' enemies are their friends:

- NonAns(x): all people such that some of their enemies' enemies are not their friends

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

3. Find all people such that all their enemies' enemies are their friends:

- NonAns(x): all people such that some of their enemies' enemies are not their friends

```
NonAns(x) ← Enemy(x, y) AND Enemy(y, z) AND
                NOT Friend(x, z)
A(x) ← Everyone(x) AND NOT NonAns(x)

Everyone(x) ← Friend(x, y)
Everyone(x) ← Friend(y, x)
Everyone(x) ← Enemy(x, y)
Everyone(x) ← Enemy(y, x)
```

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

4. Find all people X who have only friends all of whose enemies are x's enemies

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

4. Find all people X who have only friends all of whose enemies are x's enemies

- NonAns(x): all people X who have some friends some of whose enemies are not X's enemies

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

4. Find all people X who have only friends all of whose enemies are x's enemies

- NonAns(x): all people X who have some friends some of whose enemies are not X's enemies

```
NonAns(x) ← Friend(x, y) AND Enemy(y, z) AND
                 NOT Enemy(x, z)
A(x) ← NOT NonAns(x)
```

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

4. Find all people X who have only friends all of whose enemies are x's enemies

- NonAns(x): all people X who have some friends some of whose enemies are not X's enemies

```
NonAns(x) ← Friend(x, y) AND Enemy(y, z) AND
              NOT Enemy(x, z)
A(x) ← NOT NonAns(x)
```

What's wrong with this?

```
Friend(name1, name2)
Enemy(name1, name2)
```

More Examples

4. Find all people X who have only friends all of whose enemies are x's enemies

- NonAns(x): all people X who have some friends some of whose enemies are not X's enemies

```
NonAns(x) ← Friend(x, y) AND Enemy(y, z) AND
               NOT Enemy(x, z)
A(x) ← NOT NonAns(x)
```

What's wrong with this?

- x

```
NonAns(x) ← Friend(x, y) AND Enemy(y, z) AND
               NOT Enemy(x, z)
A(x) ← Everyone(x) AND NOT NonAns(x)
```

Datalog Summary

- Facts (extensional) & rules (intensional)
  - rules can use relations, arithmetic, union, intersect, etc…

Datalog Summary

- Facts (extensional) & rules (intensional)
  - rules can use relations, arithmetic, union, intersect, etc…

- As with SQL, existential quantifiers are easier
  - Use negation to handle universal

Datalog Summary

- Facts (extensional) & rules (intensional)
  - rules can use relations, arithmetic, union, intersect, etc…
- As with SQL, existential quantifiers are easier
  - Use negation to handle universal

RA $\underset{\longrightarrow}{X}$ nRD

nRD $\xrightarrow{X}$ RA

- Everything expressible in Ra is expressible in non-recursive Datalog and vice-versa
  - Recursive Datalog can express more than extended RA
  - Extended RA can express more than recursive Datalog

Thank you.