Database Design and Implementation

CS 645

SQL and Datalog
What you need

- Refresh your SQL:
  - http://sqlzoo.net

- Practice!

- You probably already have sqlite.
- Instructions to install Postgres on the assignments page on the website.

- Homework assignment 1!
Simple SQL query

```
SELECT *
FROM Product
WHERE category='Gadgets'
```

### Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
Simple SQL query

\[ \sigma_{\text{price} > 100} \pi_{\text{pname}, \text{price}, \text{manufacturer}} \]

\[ \begin{align*}
\text{SELECT} & \quad \text{pName, price, manufacturer} \\
\text{FROM} & \quad \text{Product} \\
\text{WHERE} & \quad \text{price} > 100
\end{align*} \]

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
Eliminating duplicates

| Product | | |
|---|---|---|---|
| PName | Price | Category | Manufacturer |
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| PowerGizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

**SELECT** category **FROM** Product

**SELECT** DISTINCT category **FROM** Product

Set vs. Bag semantics
Ordering the results

```sql
SELECT pName, price, manufacturer 
FROM Product 
WHERE category='Gadgets' 
AND price > 10 
ORDER BY price, pName
```

- Ties in `price` attribute broken by `pname` attribute
- Ordering is ascending by default. Descending:

```sql
... ORDER BY price, pname DESC
```
<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

**SELECT** DISTINCT category
**FROM** Product
**ORDER BY** category

**SELECT** category
**FROM** Product
**ORDER BY** pName

**SELECT** DISTINCT category
**FROM** Product
**ORDER BY** pName
<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

\[\text{SELECT DISTINCT category} \]
\[\text{FROM Product} \]
\[\text{ORDER BY category} \]

\[\text{SELECT category} \]
\[\text{FROM Product} \]
\[\text{ORDER BY pName} \]

\[\text{SELECT DISTINCT category} \]
\[\text{FROM Product} \]
\[\text{ORDER BY pName} \]

Syntax error*
Joins

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all products under $200 manufactured in Japan; return their names and prices!

SELECT pName, price
FROM Product, Company
WHERE manufacturer=cName
    and country='Japan'
    and price <= 200

Join between Product and Company
Joins

Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

Company

<table>
<thead>
<tr>
<th>CName</th>
<th>StockPrice</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>25</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>65</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>15</td>
<td>Japan</td>
</tr>
</tbody>
</table>

SELECT pName, price
FROM Product, Company
WHERE manufacturer=cName
    and country='Japan'
    and price <= 200

PName     | Price     |
-----------|-----------|
SingleTouch| $149.99   |
Semantics are tricky…

What do these queries compute?

\[
\text{SELECT DISTINCT } R.a \\
\text{FROM } R, S \\
\text{WHERE } R.a=S.a
\]

Returns \( R \cap S \)

\[
\text{SELECT DISTINCT } R.a \\
\text{FROM } R, S, T \\
\text{WHERE } R.a=S.a \text{ or } R.a=T.a
\]

If \( S \neq \emptyset \) and \( T \neq \emptyset \) then returns \( R \cap (S \cup T) \) else returns \( \emptyset \)

\( R(a), S(a), T(a) \)
Formal semantics of SQL queries

\[
\text{SELECT } a_1, a_2, \ldots, a_k \\
\text{FROM } R_1 \text{ as } x_1, R_2 \text{ as } x_2, \ldots, R_n \text{ as } x_n \\
\text{WHERE } \text{Conditions}
\]

Conceptual evaluation strategy (nested for loops):

\[
\text{Answer} = \{\} \\
\text{for } x_1 \text{ in } R_1 \text{ do} \\
\quad \text{for } x_2 \text{ in } R_2 \text{ do} \\
\quad \quad \ldots \\
\quad \quad \text{for } x_n \text{ in } R_n \text{ do} \\
\quad \quad \quad \text{if } \text{Conditions} \\
\quad \quad \quad \quad \text{then } \text{Answer} = \text{Answer} \cup \{(a_1, \ldots, a_k)\} \\
\text{return } \text{Answer}
\]
SELECT country
FROM Product, Company
WHERE manufacturer = cName
  and category = 'Gadgets'

Q: Find all countries that manufacture some product in the ‘Gadgets’ category!

Remember to use DISTINCT
Subqueries

- A subquery is a SQL query nested inside a larger query.
- Such inner-outer queries are called nested queries.
- A subquery may occur in:
  - A SELECT clause
  - A FROM clause
  - A WHERE clause
- Rule of thumb: avoid writing nested queries when possible; keep in mind that sometimes it’s impossible.
1. Subqueries in SELECT

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each product return the city where it is manufactured!

```
SELECT  P.pname,  (SELECT C.city
       FROM    Company C
       WHERE   C.cid = P.cid)
FROM     Product P
```

What happens if the subquery returns more than one city?

Runtime error
1. Subqueries in SELECT

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each product return the city where it is manufactured!

```
SELECT P.pname, (SELECT C.city
               FROM Company C
               WHERE C.cid = P.cid)
FROM Product P
```

"unnesting the query"

Whenever possible, don't use nested queries

```
SELECT P.pname, C.city
FROM Product P, Company C
WHERE C.cid = P.cid
```
2. Subqueries in FROM

Product (pname, price, cid)
Company (cid, cname, city)

Q: Find all products whose prices are > 20 and < 30!

```
SELECT X.pname
FROM (SELECT *
      FROM Product as P
      WHERE price > 20 ) as X
WHERE X.price < 30
```

unnesting

```
SELECT pname
FROM Product
WHERE price > 20 and price < 30
```
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Existential quantifiers $\exists$

Q: Find all companies that make some products with price $< 100$!

Using EXISTS:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE EXISTS (SELECT * 
               FROM Product P 
               WHERE C.cid = P.cid 
               and P.price < 100)
```
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Q: Find all companies that make some products with price < 100!

Using **IN**: 

```
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
                 FROM Product P
                 WHERE P.price < 100)
```
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Existential quantifiers \( \exists \)

Q: Find all companies that make some products with price < 100!

Using ANY:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 100 > ANY (SELECT price
                      FROM Product P
                      WHERE P.cid = C.cid)
```
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Existential quantifiers \( \exists \)

Q: Find all companies that make some products with price < 100!

Now, let's unnest:

SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid = P.cid
and P.price < 100

Existential quantifiers are easy 😊
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Universal quantifiers $\forall$

Q: Find all companies that make only products with price < 100!

same as:

Q: Find all companies for which all products have price < 100!

Universal quantifiers are more complicated! 😞
3. Subqueries in WHERE

1. Find the other companies: i.e. they have some product $\geq 100$

```sql
SELECT DISTINCT C.cname 
FROM Company C 
WHERE C.cid IN (SELECT P.cid 
    FROM Product P 
    WHERE P.price $\geq$ 100)
```

2. Find all companies s.t. all their products have price < 100

```sql
SELECT DISTINCT C.cname 
FROM Company C 
WHERE C.cid NOT IN (SELECT P.cid 
    FROM Product P 
    WHERE P.price $\geq$ 100)
```
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Q: Find all companies that make only products with price < 100!

Using NOT EXISTS:

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT *
                  FROM Product P
                  WHERE C.cid = P.cid
                  and P.price >= 100)
```
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Q: Find all companies that make only products with price < 100!

Using ALL:

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE 100 > ALL (SELECT price
                   FROM Product P
                   WHERE P.cid = C.cid)
```
Challenging question

How can we unnest a universal quantifier query?
A query Q is **monotone** if:

- Adding tuples to the input cannot remove tuples from the output

**Fact:** all unnested queries are monotone

**Proof:** using the “nested for loops” semantics

**Fact:** Query with universal quantifier is not monotone

- Add one tuple violating the condition. Then not “all”...

**Consequence:** we cannot unnest a query with a **universal quantifier**
The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Challenge: write these in SQL

Find drinkers that frequent some bar that serves some beer they like.

Find drinkers that frequent only bars that serve some beer they like.

Find drinkers that frequent some bar that serves only beers they like.

Find drinkers that frequent only bars that serve only beer they like.
Aggregation

```
SELECT avg(price)
FROM Product
WHERE maker='Toyota'
```

```
SELECT count(*)
FROM Product
WHERE year > 1995
```

SQL supports several aggregation operations:

- `sum`, `count`, `min`, `max`, `avg`

Except `count`, all aggregations apply to a single attribute.
Aggregation: count distinct

COUNT applies to duplicates, unless otherwise stated:

```
SELECT count (category)
FROM Product
WHERE year > 1995
```

We probably want:

```
SELECT count (DISTINCT category)
FROM Product
WHERE year > 1995
```
Simple aggregation

### Purchase

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

\[
3 \times 20 = 60 \\
2 \times 20 = 40 \\
\text{sum: } 100
\]

SQL creates attribute name

\[\text{SELECT } \text{sum (price } \times \text{ quantity)} \text{ FROM } \text{Purchase WHERE product = 'Bagel'}\]

100

can use arithmetic expressions
Find total quantities for all sales over $1, by product.
### SQL Query

```sql
SELECT product, sum(quantity) as TotalSales
FROM Purchase
WHERE price > 1
GROUP BY product
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

### Result

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
</tr>
<tr>
<td>Banana</td>
<td>20</td>
</tr>
</tbody>
</table>

Select contains:
- grouped attributes
- and aggregates
Another example

SELECT product,
    sum(quantity) as SumQuantity,
    max(price) as MaxPrice
FROM Purchase
GROUP BY product

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
<th>MaxPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
<td>3</td>
</tr>
<tr>
<td>Banana</td>
<td>70</td>
<td>4</td>
</tr>
</tbody>
</table>

Next, focus only on products with at least 50 sales
Q: Similar to before, but only products with at least 50 sales.
General form of grouping and aggregation

```sql
SELECT S
FROM R_1, ..., R_n
WHERE C1
GROUP BY a_1, ..., a_k
HAVING C2
```

**Evaluation**

1. Evaluate From → Where, apply condition C1
2. Group by the attributes a_1, ..., a_k
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

**C1**: is any condition on the attributes in R_1, ..., R_n

**C2**: is any condition on aggregates and on attributes a_1, ..., a_k
Finding witnesses

Store(sid, sname)
Product(pid, pname, price, sid)

Q: For each store, find its most expensive products

Finding the maximum price is easy...

```
SELECT Store.sid, max(Product.price)
FROM Store, Product
WHERE Store.sid = Product.sid
GROUP BY Store.sid
```

But we want the “witnesses”, i.e. the products with max price
Finding witnesses

- Compute max price in a subquery
- Compare it with each product price

```sql
SELECT Store.sname, Product.pname
FROM Store, Product,
     (SELECT Store.sid as sid,
        max(Product.price) as p
     FROM Store, Product
     WHERE Store.sid = Product.sid
     GROUP BY Store.sid) X
WHERE Store.sid = Product.sid
    AND Store.sid = X.sid
    AND Product.price = X.p
```
Finding witnesses

There is a more concise solution here:

```sql
SELECT Store.sname, x.pname
FROM Store, Product x
WHERE Store.sid = x.sid
    and x.price >=
        ALL (SELECT y.price
              FROM Product y
              WHERE Store.sid = y.sid)
```
NULLS in SQL

Whenever we don’t have a value, we can put a NULL

Can mean many things:
- Value does not exist
- Value exists but is unknown
- Value not applicable
- Etc.

The schema specifies for each attribute if it can be NULL or not

How does SQL cope with tables that have NULLs?
Null values

If x = NULL then
- Arithmetic operations produce NULL. E.g.: 4*(3-x)/7
- Boolean conditions are also NULL. E.g.: x = ‘Joe’

In SQL there are three boolean values:
   FALSE, TRUE, UNKNOWN

Reasoning:

\[
\begin{align*}
\text{FALSE} &= 0 \\
\text{TRUE} &= 1 \\
\text{UNKNOWN} &= 0.5 \\
x \text{ AND } y &= \min(x,y) \\
x \text{ OR } y &= \max(x,y) \\
\text{NOT } x &= (1 - x)
\end{align*}
\]
SELECT * 
FROM Person 
WHERE (age < 25) and (height > 6 or weight > 190)

Rule in SQL: include only tuples that yield TRUE

<table>
<thead>
<tr>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>NULL</td>
<td>200</td>
</tr>
<tr>
<td>NULL</td>
<td>6.5</td>
<td>170</td>
</tr>
</tbody>
</table>

SELECT * 
FROM Person 
WHERE age < 25 or age >= 25

Unexpected behavior

SELECT * 
FROM Person 
WHERE age < 25 or age >= 25 or age IS NULL

Test NULL explicitly
Outer joins

If we want the never-sold products, we need an “outerjoin”:

```
SELECT Product.name, Purchase.store
FROM Product LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Category</td>
<td>Name</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Camera</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
<td>Camera</td>
</tr>
</tbody>
</table>

Inner join does not produce this tuple
Example

Compute, for each product, the total number of sales in ‘September’

```
SELECT Product.name, count(*)
FROM Product, Purchase
WHERE Product.name = Purchase.prodName
    and Purchase.month = 'September'
GROUP BY Product.name
```

What’s wrong?
Example

Compute, for each product, the total number of sales in ‘September’

```sql
SELECT Product.name, count(*)
FROM Product
LEFT OUTER JOIN Purchase ON Product.name = Purchase.prodName
WHERE Purchase.month = 'September'
GROUP BY Product.name
```

What’s wrong?
Example

Compute, for each product, the total number of sales in ‘September’

We need to use the attribute to get the correct 0 count.

```
SELECT Product.name, count(month) 
FROM Product LEFT OUTER JOIN Purchase ON 
    Product.name = Purchase.prodName 
WHERE Purchase.month = 'September' 
GROUP BY Product.name
```
Datalog
Datalog

- Friendly notation for queries
- Designed for recursive queries in the 80s.
- Today: in a couple of commercial products, e.g., LogicBlox, Datomic

Today: recursion-free datalog with negation
Datalog: Facts and Rules

Facts = tuples in the database

Actor(34524, ’Johnny’, ’Depp’)  
Casts(34524, 28756)  
Casts(67725, 28756)  
Movie(28756, ’Sweeney Todd’, 2007)  
Movie(28757, ’The Da Vinci Code’, 2006)

Rules = queries

Q1(y) :- Movie(x,y,z), z=’2007’

Q2(f,l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’2007’)

Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y,’2007’), Casts(z,x2), Movie(x2,y2,’2006’)

Facts = tuples in the database

Rules = queries

Find movies made in 2007

Find actors who acted in a movie in 2007

Find actors who acted in a movie in 2007 and in 2006
EDB and IDB

Extensional Database Predicates: EDB
- Actor, casts, movie

Intentional Database Predicates: IDB
- Q1, Q2, Q3

Q2(f,l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’2007’)
Terminology

Q2(f,l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,'2007')

head

atom

atom

atom

body

f, l : head variables
x, y, z : existential variables
Datalog Program

\[
\begin{align*}
B0(x) & : - \text{Actor}(x, 'Kevin', 'Bacon') \\
B1(x) & : - \text{Actor}(x, f, l), \text{Casts}(x, z), \text{Casts}(y, z), B0(y) \\
B2(x) & : - \text{Actor}(x, f, l), \text{Casts}(x, z), \text{Casts}(y, z), B1(y) \\
Q4(x) & : - B1(x) \\
Q4(x) & : - B2(x)
\end{align*}
\]

union

Find actors with Bacon number ≤ 2
Simple datalog programs

What does this compute?

\[
T(x, y) :- R(x, y) \\
T(x, y) :- R(x, z) T(z, y)
\]

\[
R = \\
\begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]
Simple datalog programs

What does this compute?

\[
\begin{align*}
T(x,y) & : - R(x,y) \\
T(x,y) & : - R(x,z) \ T(z,y)
\end{align*}
\]

T is initially empty

\[
R = \\
\begin{array}{|c|c|}
\hline
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\hline
\end{array}
\]
Simple datalog programs

What does this compute?

\[
\text{T}(x,y) \Leftarrow \text{R}(x,y) \\
\text{T}(x,y) \Leftarrow \text{R}(x,z) \ 	ext{T}(z,y)
\]

1st iteration

\[
\begin{array}{c|c}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

R =
Simple datalog programs

What does this compute?

$T(x,y) :- R(x,y)$
$T(x,y) :- R(x,z) T(z,y)$

1\text{st iteration}  
\begin{tabular}{|c|c|}
\hline
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\hline
\end{tabular}

2\text{nd iteration}  
\begin{tabular}{|c|c|}
\hline
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\hline
\end{tabular}

$R =$
\begin{tabular}{|c|c|}
\hline
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\hline
\end{tabular}
Simple datalog programs

What does this compute?

\[
\begin{align*}
    T(x,y) & \iff R(x,y) \\
    T(x,y) & \iff R(x,z) \land T(z,y)
\end{align*}
\]

1\textsuperscript{st} iteration

\[
\begin{array}{c|c}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

2\textsuperscript{nd} iteration

\[
\begin{array}{c|c}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
1 & 1 \\
1 & 3 \\
2 & 2 \\
2 & 4 \\
1 & 5 \\
3 & 5 \\
2 & 5 \\
\end{array}
\]

3\textsuperscript{rd} iteration

\[
\begin{array}{c|c}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
1 & 1 \\
1 & 3 \\
2 & 2 \\
2 & 4 \\
1 & 5 \\
3 & 5 \\
2 & 5 \\
\end{array}
\]
Datalog with Negation

B0(x) :- Actor(x,’Kevin’,’Bacon’)
B1(x) :- Actor(x,f,l), Casts(x,z),Casts(y,z),B0(y)
Q5(x) :- Actor(x,f,l), not B1(x), not B0(x)

Find actors with Bacon number ≥ 2
Recursion and negation: 😞

EDB: \( R(a) \)

\[
S(x) :- R(x), \text{ not } T(x) \\
T(x) :- R(x), \text{ not } S(x)
\]

The fixpoint is unclear!
Unsafe Datalog Rules

What is unsafe about these rules?

\[ U_1(x,y) :\text{-} \text{Movie}(x,z,\text{'2007'}, y > \text{'2000'} \]

\[ U_2(x,u) :\text{-} \text{Movie}(x,z,\text{'2007'}, \text{not Casts}(u,x) \]

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