Certain Answers meet Zero-One Laws

Leonid Libkin
University of Edinburgh

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	Ord2

Customer

CUST_ID	NAME
c1	John
c2	Mary

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	“Big Data”	30
Ord2	“SQL”	35
Ord3	“Logic”	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	Ord2

Customer

CUST_ID	NAME
c1	John
c2	Mary

Typical queries, as we teach students to write them:

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	Ord2

Customer

CUST_ID	NAME
c1	John
c2	Mary

Typical queries, as we teach students to write them:

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	Ord2

Customer

CUST_ID	NAME
c1	John
c2	Mary

Typical queries, as we teach students to write them:

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	Ord2

Customer

CUST_ID	NAME
c1	John
c2	Mary

Typical queries, as we teach students to write them:

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Answer: Ord3.

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

Answer: none.

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	“Big Data”	30
Ord2	“SQL”	35
Ord3	“Logic”	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	Ord2

Customer

CUST_ID	NAME
c1	John
c2	Mary

Typical queries, as we teach students to write them:

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	“Big Data”	30
Ord2	“SQL”	35
Ord3	“Logic”	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	Ord2

Customer

CUST_ID	NAME
c1	John
c2	Mary

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	Ord2

Customer

CUST_ID	NAME
c1	John
c2	Mary

In the real world, information is often missing

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

Customer

CUST_ID	NAME
c1	John
c2	Mary

In the real world, information is often missing

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	--

Customer

CUST_ID	NAME
c1	John
c2	Mary

In the real world, information is often missing

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

A company database: orders, customers, payments

Orders

ORDER_ID	TITLE	PRICE
Ord1	“Big Data”	30
Ord2	“SQL”	35
Ord3	“Logic”	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	--

Customer

CUST_ID	NAME
c1	John
c2	Mary

In the real world, information is often missing

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

Old Answer: Ord3 New: NONE!

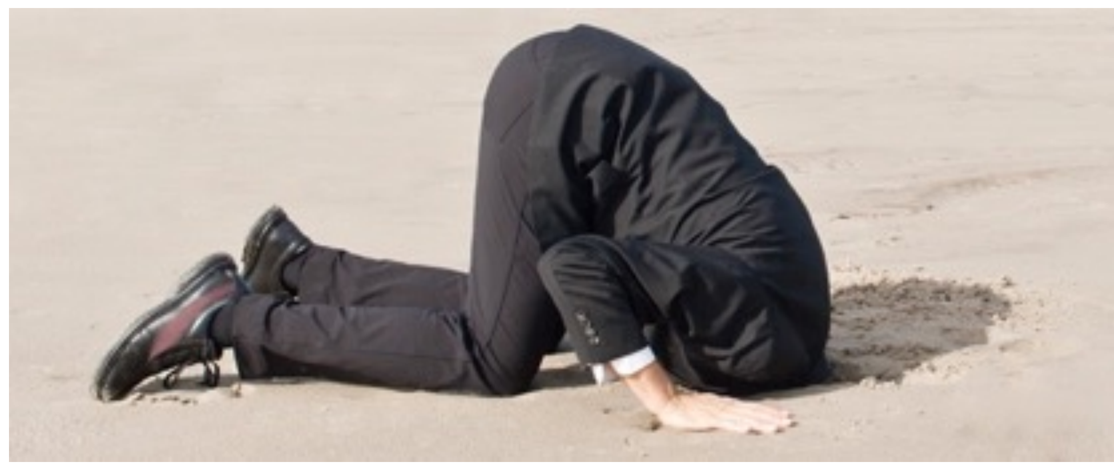
Old answer: none New: c2!

Problems

- There is a good understanding of what correctness is: certain answers
  - true in all completions (restricted validity)
- Computationally hard: coNP-hard for basic SQL
  - Hence DBMSs sacrifice correctness to ensure efficiency
  - SQL DBMSs use special rules based on 3-valued logic to get query answers
  - and these answers can be very wrong…

How to solve it

For many years, the community adopted this approach



Recently, a new idea emerged: approximations of certain answers

Can be found efficiently for all relational algebra queries

Behave well in theory (L., ACM TODS 2016)
and practice (Guagliardo, L. PODS'16 + followups)

Naive Evaluation

- Treat nulls as new constants
- Evaluate query using standard techniques
- Heavily used: data integration/exchange, OBDA etc

Orders

ORDER_ID	TITLE	PRICE
Ord1	“Big Data”	30
Ord2	“SQL”	35
Ord3	“Logic”	50

Pay

Customer

CUST_ID	NAME
c1	John
c2	Mary

Naive Evaluation

- **Treat nulls as new constants**
- **Evaluate query using standard techniques**
- **Heavily used: data integration/exchange, OBDA etc**

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

Customer

CUST_ID	NAME
c1	John
c2	Mary

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Naive Evaluation

- **Treat nulls as new constants**
- **Evaluate query using standard techniques**
- **Heavily used: data integration/exchange, OBDA etc**

Orders

ORDER_ID	TITLE	PRICE
Ord1	“Big Data”	30
Ord2	“SQL”	35
Ord3	“Logic”	50

Pay

Customer

CUST_ID	NAME
c1	John
c2	Mary

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

Naive Evaluation

- **Treat nulls as new constants**
- **Evaluate query using standard techniques**
- **Heavily used: data integration/exchange, OBDA etc**

Orders

ORDER_ID	TITLE	PRICE
Ord1	“Big Data”	30
Ord2	“SQL”	35
Ord3	“Logic”	50

Pay

Customer

CUST_ID	NAME
c1	John
c2	Mary

Unpaid orders:

select O.order_id
from Orders O
where O.order_id not in
(select order from Pay P)

Answer: Ord2, Ord3.

Customers without an order:

select C.cust_id from Customer C
where not exists
(select * from Orders O, Pay P
where C.cust_id=P.cust_id
and P.order=O.order_id)

Answer: c2.

Naive Evaluation

- Treat nulls as new constants
- Evaluate query using standard techniques
- Heavily used: data integration/exchange, OBDA etc

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

Customer

CUST_ID	NAME
c1	John
c2	Mary

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

Naive Evaluation

- **Treat nulls as new constants**
- **Evaluate query using standard techniques**
- **Heavily used: data integration/exchange, OBDA etc**

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	$\perp$

Customer

CUST_ID	NAME
c1	John
c2	Mary

Unpaid orders:

```
select O.order_id
from Orders O
where O.order_id not in
        (select order from Pay P)
```

Customers without an order:

```
select C.cust_id from Customer C
where not exists
        (select * from Orders O, Pay P
         where C.cust_id=P.cust_id
         and P.order=O.order_id)
```

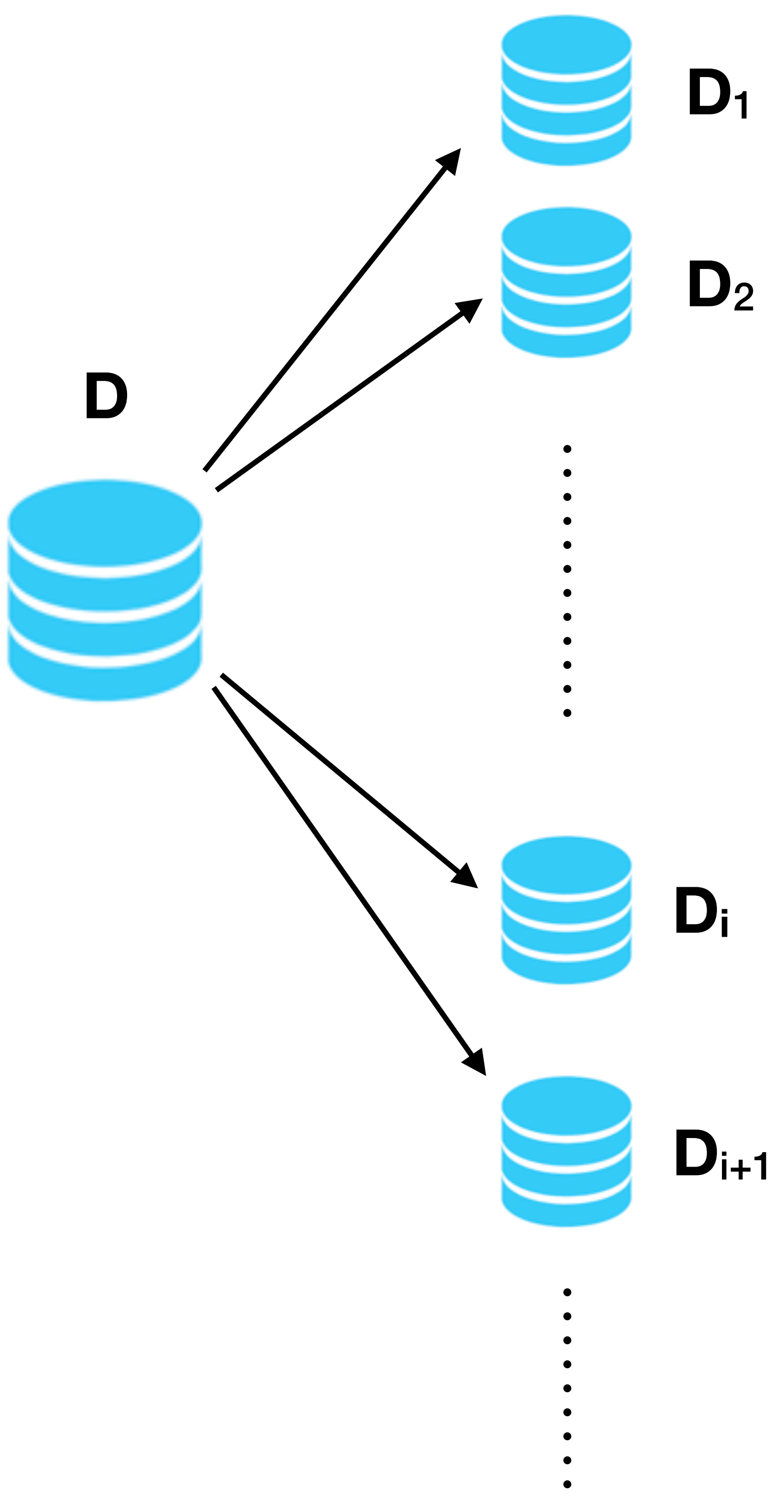
How bad are bad answers?

- What if $\perp$ is a constant that is different from Ord1, Ord2, Ord3?
- Then naive evaluation actually produces correct answers!
- If we know nothing about it is not such an unreasonable assumption: there could be many orders?
- But what if we know $\perp \in \{Ord1, Ord2, Ord3\}$?
- Then answer to the first query is Ord2 with 50% chance and Ord3 with 50% chance. Answer to the second query is empty.

Questions

- Is naive evaluation always good without constraints on nulls, or we just got lucky?
  - Yes, it always is
- Can we get the second type of answers, with constraints?
  - Yes, but with more work
- Now revisit certain answers, and connect with a well know subject in logic and probability

Incomplete data and certain answers

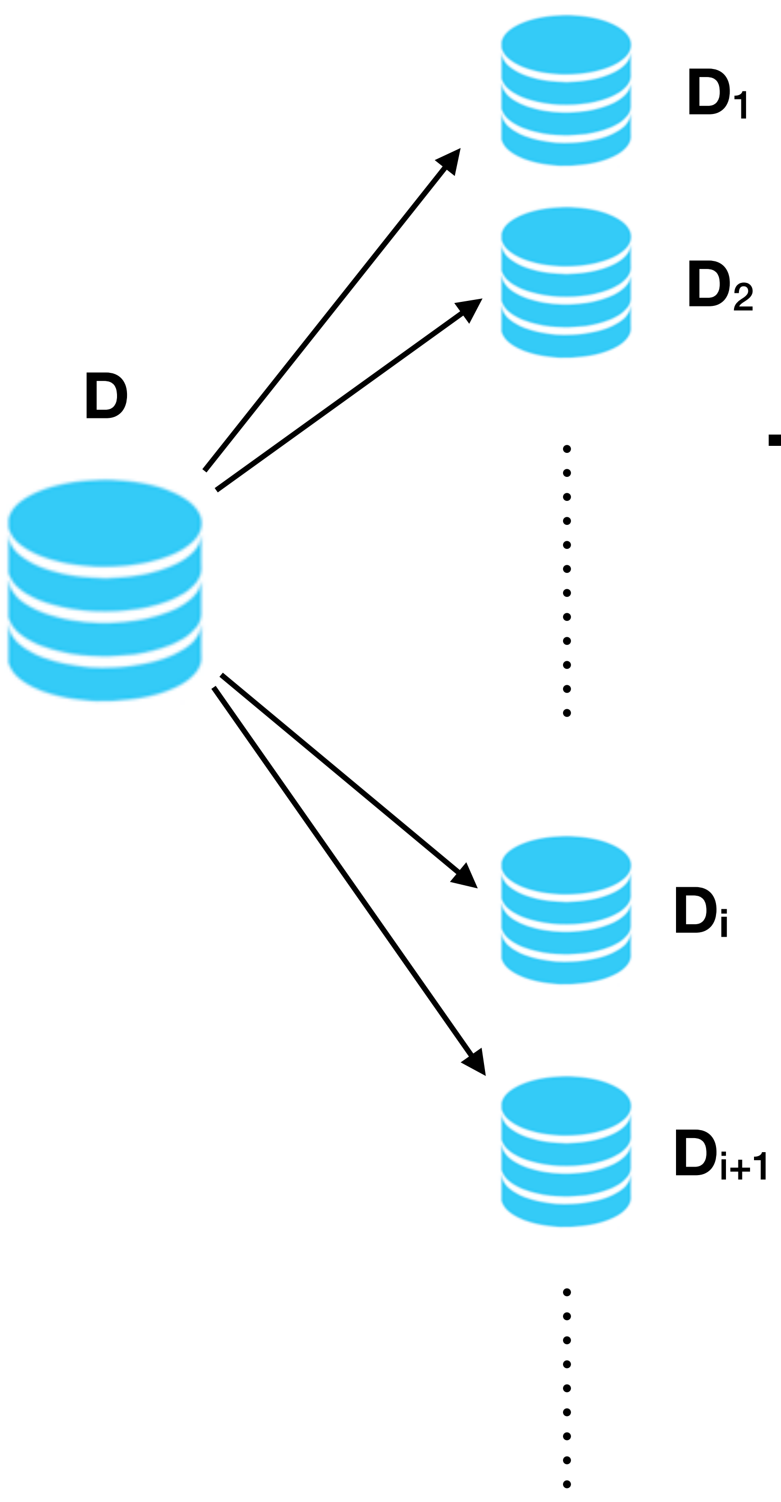


Incomplete database D represents many complete databases D_1, D_2, …

This is done by interpreting incompleteness

For example, by assigning values to every null that occurs in D

Incomplete data and certain answers



Tuple a is certain answer to query Q in D
⇔ a is an answer to Q in every D_i

Certainty is hard computationally:
coNP-hard for relational algebra queries

Zero-One Laws

A formula α over graphs; green = true; red = false

α is almost surely valid: true in almost all graphs

Examples:

- **μ(has isolated node) = 0**
- **μ(is a tree)=0**
- **μ(connected) = 1**
- **μ(has diameter at most 2) = 1**

Zero-One Laws

A formula α over graphs; green = true; red = false

…..

α is almost surely valid: true in almost all graphs

- **pick a graph G at random**
- **calculate the probability $\mu(\alpha)$ that α is true in G**
- **$\mu(\alpha) = 1 \Leftrightarrow \alpha$ is almost surely valid**

Examples:

- **μ(has isolated node) = 0**
- **μ(is a tree)=0**
- **μ(connected) = 1**
- **μ(has diameter at most 2) = 1**

Zero-One Laws

Extended to many other logics: Fixed-point, Infinitary logics, Fragments of second-order logic; Other distributions too

A very active subject in logic/combinatorics

Zero-One Laws

Fagin 1976:
if α is first-order, then $\mu(\alpha)$ is 0 or 1

Extended to many other logics: Fixed-point, Infinitary logics, Fragments of second-order logic; Other distributions too

A very active subject in logic/combinatorics

Zero-One Laws

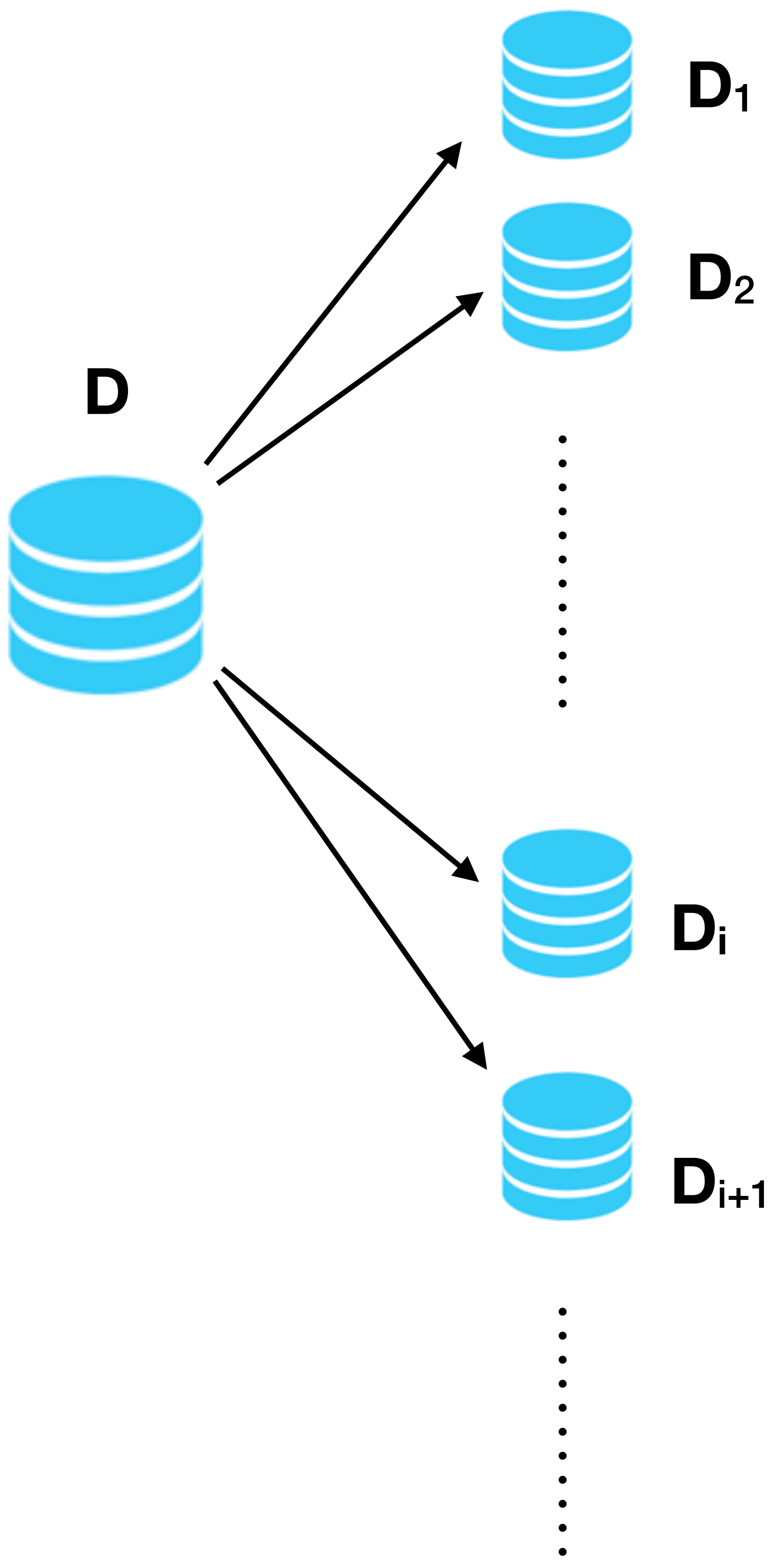
Fagin 1976:
if α is first-order, then $\mu(\alpha)$ is 0 or 1

α is valid (true in all graphs) - undecidable.
α is almost surely valid ($\mu(\alpha) = 1$) - easy to decide.

Extended to many other logics: Fixed-point, Infinitary logics, Fragments of second-order logic; Other distributions too

A very active subject in logic/combinatorics

Certainty and Zero-One Laws



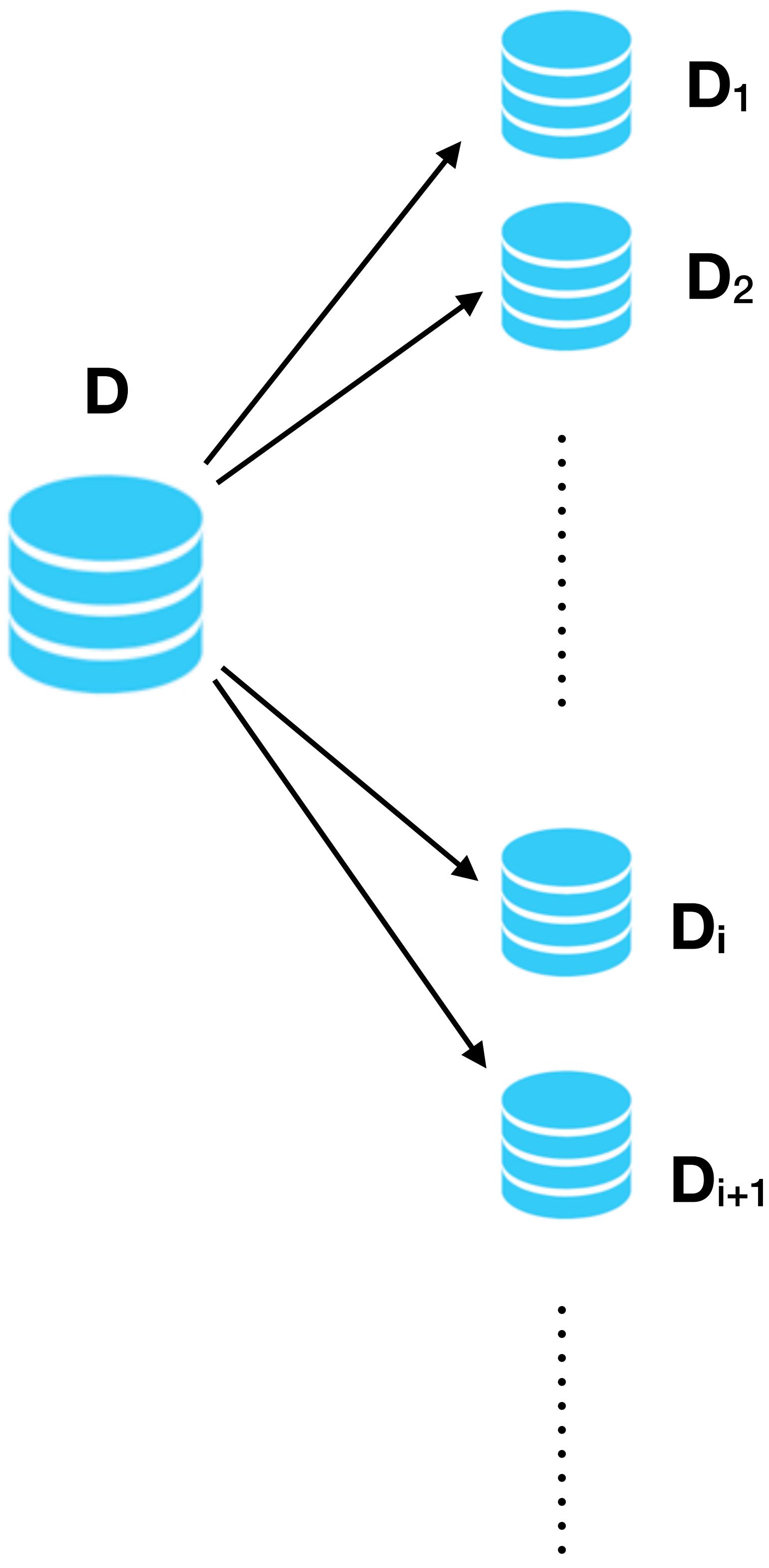
For query Q:

- **pick a complete database D_i at random**
- **$\mu(Q,D,a)$: probability that $a \in Q(D_i)$**

$$\mu(Q,D,a) = 1 \Rightarrow$$

a = almost certainly true answer to Q in D

Certainty and Zero-One Laws



For query Q:

- pick a complete database D_i at random
- $\mu(Q,D,a)$: probability that $a \in Q(D_i)$

$$\mu(Q,D,a) = 1 \Rightarrow$$

a = almost certainly true answer to Q in D

Questions

1. When is $\mu(Q,D,a) = 1$?
2. How easy is it to compute?
3. Can an answer be 50% true?
4. Is one tuple a better answer than another?

The model

Marked nulls - common in data integration, exchange, OBDA, generalize SQL nulls

D

A	B	C
1	$\perp_1$	$\perp_2$
2	$\perp_2$	3
3	$\perp_3$	$\perp_1$

$\perp_1 \to 1$

$\perp_2 \to 2$

$\perp_3 \to 3$

A	B	C
1	**1**	**2**
2	**2**	3
3	**3**	**1**

⋮

$\perp_1 \to 5$

$\perp_2 \to 4$

$\perp_3 \to 7$

A	B	C
1	**5**	**4**
2	**4**	3
3	**7**	**5**

Valuations v: Nulls → Constants

Certain Answers

A tuple of constants c is a certain answer:

$c \in Q(v(D))$ for each valuation v

An arbitrary tuple a is a certain answer:

$v(a) \in Q(v(D))$ for each valuation v

Definition of certain answers from Lipski 1984;
unfortunately forgotten for years in favour of the constants-only definition

Certain Answers

A tuple of constants c is a certain answer:

$c \in Q(v(D))$ for each valuation v

An arbitrary tuple a is a certain answer:

$v(a) \in Q(v(D))$ for each valuation v

Definition of certain answers from Lipski 1984;
unfortunately forgotten for years in favour of the constants-only definition

Support of a:

$$\mathrm{Supp}(Q,D,a) = \{\text{valuations } v \mid v(a) \in Q(v(D))\}$$

Certain Answers

Certain Answers

Support of a: Supp(Q,D,a) = {valuations v | v(a) ∈ Q(v(D)) }

Answer a is certain ⇔ every valuation v is in Supp(Q,D,a)

Certain Answers

Support of a: $\mathrm{Supp}(Q,D,a) = \{\text{valuations } v \mid v(a) \in Q(v(D))\}$

Answer a is certain ⇔ every valuation v is in Supp(Q,D,a)

Idea: answer a is almost certainly true
⇔ a randomly chosen valuation v is in Supp(Q,D,a)

Certain Answers

Support of a: Supp(Q,D,a) = {valuations v | v(a) ∈ Q(v(D)) }

Answer a is certain ⇔ every valuation v is in Supp(Q,D,a)

Idea: answer a is almost certainly true
⇔ a randomly chosen valuation v is in Supp(Q,D,a)

A small problem: there are infinitely many valuations.
But techniques from zero-one laws help: look at finite approximations.

Measuring Certainty

Constants (non-nulls) = $\{c_1, c_2, c_3, \ldots\ldots\}$

$Valuation_k$ = finite set of valuations with range $\subseteq \{c_1, \ldots, c_k\}$

$$Supp_k(Q,D,a) = Supp(Q,D,a) \cap Valuation_k$$

Measuring Certainty

Constants (non-nulls) = $\{c_1, c_2, c_3, \ldots.. \}$

$Valuation_k$ = finite set of valuations with range $\subseteq \{c_1, \ldots, c_k\}$

$$Supp_k(Q,D,a) = Supp(Q,D,a) \cap Valuation_k$$

$$\mu_k(Q,D,a) = \frac{|Supp_k(Q,D,a)|}{|Valuation_k|}$$

(a number in [0,1])

Interpretation: Probability that a randomly chosen valuation with range in $\{c_1, \ldots, c_k\}$ witnesses that a is an answer to Q

Measuring Certainty

$$\mu(Q,D,a) = \lim_{k\to\infty} \mu_k(Q,D,a)$$

Interpretation: Probability that a randomly chosen valuation witnesses that a is an answer to Q

Observation: the value $\mu(Q,D,a)$ does not depend on a particular enumeration of $\{c_1, c_2, c_3, \ldots\}$

Zero-One Law

Zero-One Law

- **Q: any reasonable query**
    - **definable in a query language such as relational algebra, datalog, second-order logic etc - formally, generic**

Zero-One Law and Naive Evaluation

Zero-One Law and Naive Evaluation

- **$\mu(Q,D,a) = 1 \Leftrightarrow a$ is returned by the naive evaluation of Q**
  - **thus almost certainly true answers are much easier to compute than certain answers**
  - **and naive evaluation is justified as being very close to certainty**

Naive evaluation: treat nulls as values

A	B
1	$\perp_1$
2	$\perp_1$
2	$\perp_2$

$-$

A	B
1	$\perp_2$
2	$\perp_1$

$=$

A	B
1	$\perp_1$
2	$\perp_2$

Naive evaluation: treat nulls as values

A	B
1	$\perp_1$
2	$\perp_1$
2	$\perp_2$

-

A	B
1	$\perp_2$
2	$\perp_1$

=

A	B
1	$\perp_1$
2	$\perp_2$

Certain answer is empty because of valuations $\perp_1, \perp_2 \rightarrow c$

If the range of nulls is infinite, such valuations are unlikely.
Returned tuples are almost certainly true answers - but not certain.

Naive evaluation: treat nulls as values

A	B
1	$\perp_1$
2	$\perp_1$
2	$\perp_2$

-

A	B
1	$\perp_2$
2	$\perp_1$

=

A	B
1	$\perp_1$
2	$\perp_2$

Certain answer is empty because of valuations $\perp_1, \perp_2 \to c$

If the range of nulls is infinite, such valuations are unlikely.
Returned tuples are almost certainly true answers - but not certain.

In general, naive evaluation ≠ certain answers. Exceptions:

- **unions of conjunctive queries**
- **their extension with $Q \div R$ where R is a relation**

Naive evaluation: treat nulls as values

A	B
1	$\perp_1$
2	$\perp_1$
2	$\perp_2$

-

A	B
1	$\perp_2$
2	$\perp_1$

=

A	B
1	$\perp_1$
2	$\perp_2$

What if:

1. **A is a key of the second relation (which forces $\perp_1 = \perp_2$), or**
2. **there is a restriction on the range of B?**

The reasoning that valuations $\perp_1, \perp_2 \rightarrow c$ are unlikely no longer works

This is due to the presence of constraints.

Certainty with constraints

- **Only interested in databases satisfying integrity constraints Σ - for example, keys or foreign keys**
- **Standard approach: find certain answers to $\Sigma \rightarrow Q$**
- **Not very informative (because $\Sigma \rightarrow Q$ is $\neg\Sigma \vee Q$)**
    - **if $\mu(\Sigma,D) = 0$, then $\mu(\Sigma \rightarrow Q,D,a) = 1$**
    - **if $\mu(\Sigma,D) = 1$, then $\mu(\Sigma \rightarrow Q,D,a) = \mu(Q,D,a)$**

Certainty with constraints

- **A better idea: use conditional probability $\mu(Q \mid \Sigma, D, a)$**
    - **probability that a randomly chosen valuation that satisfies Σ also witnesses that a is an answer to Q**
- **Still defined as a limit since there are infinitely many valuations**

Measuring certainty with constraints

$$Supp_k(Q,D,a) = \{\text{valuations } v \in Valuation_k \mid v(a) \in Q(v(D))\}$$

$$\mu_k(Q \mid \Sigma, D, a) = \frac{|Supp_k(Q \wedge \Sigma, D, a)|}{|Supp_k(\Sigma, D, a)|}$$

Interpretation: Probability that a randomly chosen valuation with range in $\{c_1, \ldots, c_k\}$ that witnesses constraints Σ also witnesses that a is an answer to Q

Measuring certainty with constraints

$$\mu(Q \mid \Sigma, D, a) = \lim_{k\to\infty} \mu_k(Q \mid \Sigma, D, a)$$

Interpretation: Probability that a randomly chosen valuation that witnesses constraints Σ also witnesses that a is an answer to Q

Observation: the value $\mu(Q \mid \Sigma, D, a)$ does not depend on a particular enumeration of $\{c_1, c_2, c_3, \ldots\}$

Zero-One Law fails with constraints

- **Database D: $R = \{\perp\}$, $S = \{1\}$, $U = \{1,2\}$**
- **Constraint: $R \subseteq U$**
- **Query Q: is $R \subseteq S$?**
- ***$\mu(Q \mid \Sigma, D, a) = 0.5$***

What if zero-one fails?

- **The best next thing: convergence**
- **Consider, for example, ordered graphs.**
- **Zero-one law fails: μ(edge between the smallest and the largest element) = 0.5**
- **But $\mu(\alpha)$ exists for every first-order α**
  - **and is a rational of the form $n/2^m$ (Lynch 1980)**

Convergence with constraints

Convergence with constraints

- **Q: any reasonable query, Σ: any reasonable constraints (both generic)**

Convergence with constraints

- **Q: any reasonable query, Σ: any reasonable constraints (both generic)**
- **Theorem: $\mu(Q \mid \Sigma, D, a)$ always exists**
    - **$\mu(Q \mid \Sigma, D, a)$ is a rational number between 0 and 1**

Convergence with constraints

- **Q: any reasonable query, Σ: any reasonable constraints (both generic)**
- **Theorem: $\mu(Q \mid \Sigma, D, a)$ always exists**
  - **$\mu(Q \mid \Sigma, D, a)$ is a rational number between 0 and 1**
- **Every rational number in $[0,1]$ can appear as $\mu(Q \mid \Sigma, D, a)$ for a conjunctive query Q and inclusion constraints Σ**

Convergence with constraints

Convergence with constraints

- **Computing $\mu(Q \mid \Sigma, D, a)$**

Convergence with constraints

- **Computing $\mu(Q \mid \Sigma, D, a)$**
- **It is a rational number, so need a function complexity class**

Convergence with constraints

- Computing $\mu(Q \mid \Sigma, D, a)$
- It is a rational number, so need a function complexity class
- It can be computed in $FP^{\#P}$
  - functions computable in polynomial time with access to #P oracle

Convergence with constraints

- **Computing $\mu(Q \mid \Sigma, D, a)$**
- **It is a rational number, so need a function complexity class**
- **It can be computed in $FP^{\#P}$**
    - **functions computable in polynomial time with access to #P oracle**
- **Computing $\mu(Q \mid \Sigma, D, a)$ could be hard for $FP^{\#P}$**
    - **under the appropriate definition of hardness for function classes**

Constraints and zero-one laws

- Zero-one law still holds for some constraints, e.g., functional dependencies
- Σ: a set of functional dependencies.
- Known: if Q is a conjunctive query, then certain answers under Σ = $Q(\text{chase}(D,\Sigma))$
- If Q is an arbitrary query, then almost certainly true answers under Σ = $Q(\text{chase}(D,\Sigma))$
    - $\mu(Q \mid \Sigma, D, a) = \mu(Q, \text{chase}(D, \Sigma), a)$

Qualitative Measures

- **We can also use supports Supp(Q,D,a) to define qualitative measures:**

Qualitative Measures

- **We can also use supports Supp(Q,D,a) to define qualitative measures:**
  - **a is at least as good an answer as b, to query Q if Supp(Q,D,b) $\subseteq$ Supp(Q,D,a)**

Qualitative Measures

- **We can also use supports Supp(Q,D,a) to define qualitative measures:**
  - **a is at least as good an answer as b, to query Q if Supp(Q,D,b) $\subseteq$ Supp(Q,D,a)**
  - **a is a better answer than b, to query Q if Supp(Q,D,b) $\subsetneq$ Supp(Q,D,a)**

Qualitative Measures

- **We can also use supports Supp(Q,D,a) to define qualitative measures:**
  - **a is at least as good an answer as b, to query Q if Supp(Q,D,b) $\subseteq$ Supp(Q,D,a)**
  - **a is a better answer than b, to query Q if Supp(Q,D,b) $\subsetneq$ Supp(Q,D,a)**
  - **a is a best answer to Q if there is no better answer**

Qualitative measure: example

A	B
1	$\perp_1$
2	$\perp_1$
2	$\perp_2$

-

A	B
1	$\perp_2$
2	$\perp_1$

Qualitative measure: example

A	B
1	$\bot_1$
2	$\bot_1$
2	$\bot_2$

\-

A	B
1	$\bot_2$
2	$\bot_1$

- **No certain answers**
- **Naive evaluation gives $(1, \bot_1)$ and $(2, \bot_2)$**
- **$(2, \bot_2)$ is a better answer than $(1, \bot_1)$**
- **Best answer = $(2, \bot_2)$**

Qualitative measure: example

A	B
1	$\perp_1$
2	$\perp_1$
2	$\perp_2$

$-$

A	B
1	$\perp_2$
2	$\perp_1$

- **No certain answers**
- **Naive evaluation gives $(1, \perp_1)$ and $(2, \perp_2)$**
- **$(2, \perp_2)$ is a better answer than $(1, \perp_1)$**
- **Best answer = $(2, \perp_2)$**

Unlike certain answers, best answers always exist

Qualitative measures: complexity

- **Fix a query Q of relational algebra/calculus**
- **Input: database D, tuples a and b**

Qualitative measures: complexity

- **Fix a query Q of relational algebra/calculus**
- **Input: database D, tuples a and b**
- **Is a at least as good as b?** **coNP-complete**

Qualitative measures: complexity

- **Fix a query Q of relational algebra/calculus**
- **Input: database D, tuples a and b**
- **Is a at least as good as b?** **coNP-complete**
- **Is a better than b?** **DP-complete**

Qualitative measures: complexity

- **Fix a query Q of relational algebra/calculus**
- **Input: database D, tuples a and b**
- **Is a at least as good as b?** **coNP-complete**
- **Is a better than b?** **DP-complete**
- **Identify the set of best answers:** **$P^{NP[\log n]}$-complete**

Qualitative measures: complexity

- **Fix a query Q of relational algebra/calculus**
- **Input: database D, tuples a and b**
- **Is a at least as good as b?** **coNP-complete**
- **Is a better than b?** **DP-complete**
- **Identify the set of best answers:** **$P^{NP[\log n]}$-complete**
- **For unions of conjunctive queries, all in PTIME.**
    - Does not go via naive evaluation; the algorithm is of very different nature

Quantitative vs qualitative

- **Quantitative: $\mu(Q,D,a) = 0$ or $\mu(Q,D,a) = 1$**
- **Qualitative: best or not best**
- **All 4 combinations are possible, even for first-order queries**

Summary

Almost certainly true answers to queries on incomplete databases

- **Very good - though not always perfect**
- **Much better than certain answers computationally**

Summary

Almost certainly true answers to queries on incomplete databases

- Very good - though not always perfect
- Much better than certain answers computationally

Future

Summary

Almost certainly true answers to queries on incomplete databases

- Very good - though not always perfect
- Much better than certain answers computationally

Future

- Other data models
- SQL nulls and SQL evaluation
- Other distributions
- Applications of certain answers (integration, exchange, etc)