

# SQL's Handling of Nulls: Can It Be Fixed?

Leonid Libkin (University of Edinburgh)

## Problematic SQL's nulls

*“... this topic cannot be described in a manner that is simultaneously both comprehensive and comprehensible”*  
*“Those SQL features are ... fundamentally at odds with the way the world behaves”*

*C. Date & H. Darwen, 'A Guide to SQL Standard'*

*“If you have any nulls in your database, you're getting wrong answers to some of your queries. What's more, you have no way of knowing, in general, just which queries you're getting wrong answers to; all results become suspect. You can never trust the answers you get from a database with nulls”*

*C. Date, 'Database in Depth'*

## But nulls must be handled...

- ▶ Incomplete data is **everywhere**.
- ▶ The more data we accumulate, the more incomplete data we accumulate.
- ▶ Sources:
  - ▶ Traditional (missing data, wrong entries, etc)
  - ▶ The Web
  - ▶ Integration/translation/exchange of data, etc
- ▶ The importance of it was recognized early
  - ▶ Codd, "*Understanding relations (installment #7)*", 1975.
- ▶ And yet the state is **very poor**.

# SQL: example 1

## Orders

order_id	title
ord1	<i>'SQL Standard'</i>
ord2	<i>'Database Systems'</i>
ord3	<i>'Logic'</i>

## Payments

pay_id	order_id	amount
p1	ord1	-
p2	-	\$50

## SQL: example 1

### Orders

order_id	title
ord1	<i>'SQL Standard'</i>
ord2	<i>'Database Systems'</i>
ord3	<i>'Logic'</i>

### Payments

pay_id	order_id	amount
p1	ord1	-
p2	-	\$50

Query: all payment ids. Written as:

```
SELECT pay_id FROM Payments  
WHERE amount  $\geq$  50 OR amount < 50
```

# SQL: example 1

## Orders

order_id	title
ord1	'SQL Standard'
ord2	'Database Systems'
ord3	'Logic'

## Payments

pay_id	order_id	amount
p1	ord1	-
p2	-	\$50

Query: all payment ids. Written as:

```
SELECT pay_id FROM Payments
WHERE amount ≥ 50 OR amount < 50
```

Answer: **only p2!**

## SQL: example 2

Query: **unpaid** orders:

```
SELECT order_id FROM Orders
WHERE order_id NOT IN (SELECT order_id FROM Payments)
```

Answer:

## SQL: example 2

Query: **unpaid** orders:

```
SELECT order_id FROM Orders
WHERE order_id NOT IN (SELECT order_id FROM Payments)
```

Answer: **EMPTY!**



## SQL: example 2

Query: **unpaid** orders:

```
SELECT order_id FROM Orders
WHERE order_id NOT IN (SELECT order_id FROM Payments)
```

Answer: **EMPTY!**

- ▶ This goes against our intuition: 3 orders, 2 payments.
- ▶ At least **one must be unpaid!**

## SQL: example 2

Query: **unpaid** orders:

```
SELECT order_id FROM Orders
WHERE order_id NOT IN (SELECT order_id FROM Payments)
```

Answer: **EMPTY!**

- ▶ This goes against our intuition: 3 orders, 2 payments.
- ▶ At least **one must be unpaid!**
- ▶ SQL tells us that  $|X| > |Y|$  and  $X - Y = \emptyset$  are compatible.
- ▶ This is cast in stone (SQL standard).

## Source of problems: 3-valued logic

SQL used **3-valued logic**, or **3VL**, for databases with nulls.

Comparisons involving nulls evaluate to **unknown**: for instance,  $5 = \text{null}$  results in **unk**.

They are propagated using 3VL rules:

$$\begin{array}{lll} \text{unk} \vee \text{unk} = \text{unk} & \text{unk} \vee \text{true} = \text{true} & \text{unk} \wedge \text{unk} = \text{unk} \\ \text{unk} \wedge \text{false} = \text{false} & \neg \text{unk} = \text{unk} & \text{etc} \end{array}$$

- ▶ Committee design from 30 years ago, leads to many problems,
- ▶ but is efficient and used **everywhere**

## What does theory have to offer?

The notion of **correctness** — **certain answers**.

- ▶ Answers independent of the interpretation of missing information.
- ▶ Typically defined as

$$\text{certain}(Q, D) = \bigcap Q(D')$$

over all possible worlds  $D'$  described by  $D$

- ▶ Standard approach, used in all applications: data integration and exchange, inconsistent data, querying with ontologies, data cleaning, etc.

Question: **What is the relationship between SQL and certain answers?**

# Can they be the same?

No!

# Can they be the same?

No!

Complexity argument:

- ▶ Finding certain answers for relational calculus queries in **coNP**-hard
- ▶ SQL is very efficient (**DLOGSPACE**)

# Can they be the same?

No!

## Complexity argument:

- ▶ Finding certain answers for relational calculus queries in **coNP**-hard
- ▶ SQL is very efficient (**DLOGSPACE**)

Now we:

- ▶ Analyze the behavior of SQL
- ▶ See what it can do **wrong** (when **right** is certain answers)
- ▶ See what we can do to make it go wrong less often

## Wrong behaviors: false negatives and false positives

**False negatives:** missing some of the certain answers

**False positives:** giving answers which are not certain

Complexity tells us:

SQL query evaluation cannot avoid both!



## Wrong behaviors: false negatives and false positives

**False negatives:** missing some of the certain answers

**False positives:** giving answers which are not certain

Complexity tells us:

SQL query evaluation cannot avoid both!

SQL **must** generate at least one type of errors.

## SQL's errors

False positives are **worse**: they tell you something blatantly false rather than hide part of the truth

But examples we've seen only have **false negatives**.

Perhaps SQL only generates one type of errors – and milder ones?

Since it is impossible to avoid errors altogether, this wouldn't be so bad.

And complexity doesn't rule this out.

## But design by committee does...

Relations:  $R = \begin{array}{|c|} \hline A \\ \hline 1 \\ \hline \end{array}$        $S = \begin{array}{|c|} \hline A \\ \hline \text{null} \\ \hline \end{array}$

Query:

```
SELECT R.A FROM R
WHERE R.A NOT IN (SELECT R1.A FROM R R1
                  WHERE R1.A NOT IN (SELECT * FROM S))
```

Essentially  $R - (R - S)$

Certain answer:  $\emptyset$       SQL answer: **1**

# What we do

Identify the source of the problem:

- ▶ SQL's evaluation is based on **3-valued logic**
- ▶ Some of the rules for handling **true**, **false**, and **unknown** are quite arbitrary.

## What we do

Identify the source of the problem:

- ▶ SQL's evaluation is based on 3-valued logic
- ▶ Some of the rules for handling true, false, and unknown are quite arbitrary.

We show that a slight fix of the rules avoids false positives.

## What we do

Identify the source of the problem:

- ▶ SQL's evaluation is based on **3-valued logic**
- ▶ Some of the rules for handling **true**, **false**, and **unknown** are quite arbitrary.

We show that a slight fix of the rules **avoids false positives**.

Idea of the fix: be **faithful** to 3-valuedness and classify answers not into **(certain, the rest)** but rather:

**certainly true — certainly false — unknown**

## Evaluation procedures for first-order queries

Given a database  $D$ , a query  $Q(\bar{x})$ , a tuple  $\bar{a}$

$\text{Eval}(D, Q(\bar{a})) \in$  set of truth values

- ▶ 2-valued logic: truth values are 1 (true) and 0 (false)
- ▶ 3-valued logic: 1, 0, and  $\frac{1}{2}$  (unknown)

Meaning: if  $\text{Eval}(D, Q(\bar{a}))$  evaluates to

- ▶ 1, we know  $\bar{a} \in Q(D)$
- ▶ 0, we know  $\bar{a} \notin Q(D)$
- ▶  $\frac{1}{2}$ , we don't know whether  $\bar{a} \in Q(D)$  or  $\bar{a} \notin Q(D)$

## Not reinventing the wheel...

All evaluation procedures are completely standard for  $\vee, \wedge, \neg, \forall, \exists$ :

$$\text{Eval}(D, Q \vee Q') = \max(\text{Eval}(D, Q), \text{Eval}(D, Q'))$$

$$\text{Eval}(Q \wedge Q', D) = \min(\text{Eval}(D, Q), \text{Eval}(D, Q'))$$

$$\text{Eval}(D, \neg Q) = 1 - \text{Eval}(D, Q)$$

$$\text{Eval}(D, \exists x Q(x, \bar{a})) = \max\{\text{Eval}(D, Q(a', \bar{a})) \mid a' \in \text{adom}(D)\}$$

$$\text{Eval}(D, \forall x Q(x, \bar{a})) = \min\{\text{Eval}(D, Q(a', \bar{a})) \mid a' \in \text{adom}(D)\}$$



## FO evaluation procedure

We only need to give rules for atomic formulae.

$$\text{Eval}_{\text{FO}}(D, R(\bar{a})) = \begin{cases} 1 & \text{if } \bar{a} \in R \\ 0 & \text{if } \bar{a} \notin R \end{cases}$$

$$\text{Eval}_{\text{FO}}(D, a = b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases}$$

## SQL evaluation procedure

All that changes is the rule for comparisons.

We write  $\text{Null}(a)$  if  $a$  is a null and  $\text{NotNull}(a)$  if it is not.

$$\text{Eval}_{\text{SQL}}(D, a = b) = \begin{cases} 1 & \text{if } a = b \text{ and } \text{NotNull}(a, b) \\ 0 & \text{if } a \neq b \text{ and } \text{NotNull}(a, b) \\ \frac{1}{2} & \text{if } \text{Null}(a) \text{ or } \text{Null}(b) \end{cases}$$

SQL's rule: if one attribute of a comparison is null, the result is unknown.

## What's wrong with it?

We are **too eager** to say **no** or **unknown**.

If we say **no** to a result that ought to be **unknown**, when negation applies, **no** becomes **yes**! And that's how false positives creep in.

Consider  $R =$ 

A	B
1	null

What about  $(\text{null}, \text{null}) \in R$ ?

SQL says **no** but correct answer is **unknown**: what if **null** is really **1**?

## A word about the model

We go a bit beyond SQL in fact — **marked** nulls.

Semantics: missing values (aka **closed world** semantics)

A	B	C
1	2	$\perp_1$
$\perp_2$	$\perp_1$	3
$\perp_3$	5	1
2	$\perp_3$	3

## A word about the model

We go a bit beyond SQL in fact — **marked** nulls.

Semantics: missing values (aka **closed world** semantics)

A	B	C
1	2	$\perp_1$
$\perp_2$	$\perp_1$	3
$\perp_3$	5	1
2	$\perp_3$	3

$$h(\perp_1) = 4$$

$$h(\perp_2) = 3$$

$$h(\perp_3) = 5$$



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

## A word about the model

We go a bit beyond SQL in fact — **marked** nulls.

Semantics: missing values (aka **closed world** semantics)

A	B	C
1	2	$\perp_1$
$\perp_2$	$\perp_1$	3
$\perp_3$	5	1
2	$\perp_3$	3

$$h(\perp_1) = 4$$

$$h(\perp_2) = 3$$

$$h(\perp_3) = 5$$



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

SQL model: a special case when all nulls are distinct.

## Correctness: what do we know?

$$\text{Eval}(Q, D) = \{\bar{a} \mid \text{Eval}(D, Q(\bar{a})) = 1\}$$

We want at least **simple correctness guarantees**

$$\text{constant tuples in } \text{Eval}(Q, D) \subseteq \text{certain}(Q, D)$$

where  $\text{certain}(Q, D)$  is the standard definition of certain answers:

$$\text{certain}(Q, D) = \bigcap \{Q(D') \mid D' = h(D) \text{ for a valuation } h\}$$

## Correctness: what do we know?

$$\text{Eval}(Q, D) = \{\bar{a} \mid \text{Eval}(D, Q(\bar{a})) = 1\}$$

We want at least **simple correctness guarantees**

$$\text{constant tuples in } \text{Eval}(Q, D) \subseteq \text{certain}(Q, D)$$

where  $\text{certain}(Q, D)$  is the standard definition of certain answers:

$$\text{certain}(Q, D) = \bigcap \{Q(D') \mid D' = h(D) \text{ for a valuation } h\}$$

Sometimes we want/get even more:

$$\text{constant tuples in } \text{Eval}(Q, D) = \text{certain}(Q, D)$$



## Languages for correctness

UCQ: unions of conjunctive queries, or positive relational algebra

$\pi, \sigma, \bowtie, \cup$ .

$\text{FO}_{\text{certain}}$  — UCQs extended with the division operator  $\div$

- ▶ but only  $Q \div R$  queries
- ▶ meaning: find tuples  $\bar{a}$  that occur in  $Q(D)$  together with every tuple  $\bar{b}$  in  $R$

For  $\text{FO}_{\text{certain}}$  queries,

constant tuples in  $\text{Eval}_{\text{FO}}(Q, D) = \text{certain}(Q, D)$

For UCQs,

constant tuples in  $\text{Eval}_{\text{SQL}}(Q, D) \subseteq \text{certain}(Q, D)$

## Towards a good evaluation: unifying tuples

Two tuples  $\bar{t}_1$  and  $\bar{t}_2$  **unify** if there is a mapping  $h$  of nulls to constants such that  $h(\bar{t}_1) = h(\bar{t}_2)$ .

$$\begin{pmatrix} 1 & \perp & 1 & 3 \\ \perp' & 2 & \perp' & 3 \end{pmatrix} \implies \begin{pmatrix} 1 & 2 & 1 & 3 \end{pmatrix}$$

but  $\begin{pmatrix} 1 & \perp & 2 & 3 \\ \perp' & 2 & \perp' & 3 \end{pmatrix}$  do not unify.

This can be checked in **linear time**.

## Proper 3-valued procedure

$$\text{Eval}_{3v}(D, R(\bar{a})) = \begin{cases} 1 & \text{if } \bar{a} \in R \\ 0 & \text{if } \bar{a} \text{ does not unify with any tuple in } R \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

$$\text{Eval}_{3v}(D, a = b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \text{ and } \text{NotNull}(a, b) \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

## Simple correctness guarantees: no false positives

If  $\bar{a}$  is a tuple without nulls, and  $\text{Eval}_{3v}(D, Q(\bar{a})) = 1$  then  $\bar{a} \in \text{certain}(Q, D)$ .

Simple correctness guarantees:

constant tuples in  $\text{Eval}_{3v}(Q, D) \subseteq \text{certain}(Q, D)$

Thus:

- ▶ Fast evaluation (checking  $\text{Eval}_{3v}(D, Q(\bar{a})) = 1$  in **DLOGSPACE**)
- ▶ Correctness guarantees: no false positives

## Strong correctness guarantees: involving nulls

How can we give correctness guarantees for tuples with nulls? By a natural extension of the standard definition.

A tuple **without nulls**  $\bar{a}$  is a **certain answer** if

$\bar{a} \in Q(h(D))$  for every valuation  $h$  of nulls.

## Strong correctness guarantees: involving nulls

How can we give correctness guarantees for tuples with nulls? By a natural extension of the standard definition.

A tuple **without nulls**  $\bar{a}$  is a **certain answer** if

$$\bar{a} \in Q(h(D)) \text{ for every valuation } h \text{ of nulls.}$$

An arbitrary tuple  $\bar{a}$  is a **certain answers with nulls** if

$$h(\bar{a}) \in Q(h(D)) \text{ for every valuation } h \text{ of nulls.}$$

Notation:  $\text{certain}_{\perp}(Q, D)$

## Certain answers with nulls: properties

$$\text{certain}(Q, D) \subseteq \text{certain}_{\perp}(Q, D) \subseteq \text{Eval}_{\text{FO}}(Q, D)$$

Moreover:

- ▶  $\text{certain}(Q, D)$  is the set of null free tuples in  $\text{certain}_{\perp}(Q, D)$
- ▶  $\text{certain}_{\perp}(Q, D) = \text{Eval}_{\text{FO}}(Q, D)$  for  $\text{FO}_{\text{certain}}$  queries

## Correctness with nulls: strong guarantees

- ▶  $D$  – a database,
- ▶  $Q(\bar{x})$  – a first-order query
- ▶  $\bar{a}$  – a tuple of elements from  $D$ .

Then:

$$\text{▶ Eval}_{3v}(D, Q(\bar{a})) = 1 \implies \bar{a} \in \text{certain}_{\perp}(Q, D)$$

$$\text{▶ Eval}_{3v}(D, Q(\bar{a})) = 0 \implies \bar{a} \in \text{certain}_{\perp}(\neg Q, D)$$

3-valuedness extended to answers: **certainly true**, **certainly false**, **don't know**.



## When things are easy: UCQs with inequalities

Follow the two-valued standard FO procedure, and add one rule:

$$\text{Eval}_{\text{UCQ}\neq}(D, a \neq b) = \begin{cases} 1 & \text{if } a \neq b \text{ and NotNull}(a, b) \\ 0 & \text{otherwise} \end{cases}$$

Then, for UCQs with inequalities

$$\text{Eval}_{3v}(D, Q(\bar{a})) = 1 \quad \Leftrightarrow \quad \text{Eval}_{\text{UCQ}\neq}(D, Q(\bar{a})) = 1$$

**Corollary:** correctness guarantees for  $\text{Eval}_{\text{UCQ}\neq}$  over UCQs with inequalities.

## Open world assumption (OWA) semantics

Tuples can be added:

A	B	C
1	2	$\perp_1$
$\perp_2$	$\perp_1$	3
$\perp_3$	5	1
2	$\perp_3$	3

# Open world assumption (OWA) semantics

Tuples can be added:

A	B	C
1	2	$\perp_1$
$\perp_2$	$\perp_1$	3
$\perp_3$	5	1
2	$\perp_3$	3

$$h(\perp_1) = 4$$

$$h(\perp_2) = 3$$

$$h(\perp_3) = 5$$

$$\implies$$

A	B	C
1	2	4
3	4	3
5	5	1
2	5	3
7	8	9
17	18	19

## Open world assumption (OWA) semantics

Tuples can be added:

A	B	C
1	2	$\perp_1$
$\perp_2$	$\perp_1$	3
$\perp_3$	5	1
2	$\perp_3$	3

$$h(\perp_1) = 4$$

$$h(\perp_2) = 3$$

$$h(\perp_3) = 5$$



A	B	C
1	2	4
3	4	3
5	5	1
2	5	3
7	8	9
17	18	19

Observation:  $Eval_{3v}$  doesn't work under OWA

## OWA: problems

- ▶ We can never be sure that a tuple is **not** in a relation
- ▶ We can never be sure a universal  $\forall$  query holds
- ▶ We can never be sure an existential  $\exists$  query does not hold.

**Solution:** make the result of evaluation  $\frac{1}{2}$  in the worst case, forget 0

## OWA: solution

$$\text{Eval}_{3v}^{\text{owa}}(D, R(\bar{a})) = \begin{cases} 1 & \text{if } \bar{a} \in R \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

$$\text{Eval}_{3v}^{\text{owa}}(D, \exists x \varphi(x, \bar{a})) = \max \left\{ \frac{1}{2}, \max_{a' \in \text{adom}} \{ \text{Eval}_{3v}^{\text{owa}}(D, \varphi(a', \bar{a})) \} \right\}$$

$$\text{Eval}_{3v}^{\text{owa}}(D, \forall x \varphi(x, \bar{a})) = \min \left\{ \frac{1}{2}, \min_{a' \in \text{adom}} \{ \text{Eval}_{3v}^{\text{owa}}(D, \varphi(a', \bar{a})) \} \right\}$$

$\text{Eval}_{3v}^{\text{owa}}$  has correctness guarantees under OWA.

## Relational algebra queries

This is how it will be implemented after all.

Why not

Relational algebra  $Q \implies$  equivalent FO  $\varphi \implies \text{Eval}_{3v}(D, \varphi)?$

Because the algebra-to-calculus translation works in the 2-valued world and doesn't provide 3-valued guarantees.

Also we become dependent on a particular translation.

So we need to work directly on relational algebra queries.

## 3-valued implementation of RA with correctness guarantees

Classify tuples into:

- ▶ certainly true
- ▶ certainly false
- ▶ don't know

To do this, define a translation

$$Q \mapsto (Q^+, Q^-)$$

with **certainty guarantees**, i.e.

$$Q^+(D) \subseteq \text{certain}_\perp(Q, D) \quad Q^-(D) \subseteq \text{certain}_\perp(\bar{Q}, D)$$



## Relational algebra translations: basic rules

For a relation  $R$ :

- ▶  $R^+ = R$
- ▶  $R^- = \{t \mid t \text{ doesn't unify with anything in } R\} \subseteq \bar{R}$

For union (intersection is dual)

- ▶  $(Q_1 \cup Q_2)^+ = Q_1^+ \cup Q_2^+$
- ▶  $(Q_1 \cup Q_2)^- = Q_1^- \cap Q_2^-$

For difference:

- ▶  $(Q_1 - Q_2)^+ = Q_1^+ \cap Q_2^-$
- ▶  $(Q_1 - Q_2)^- = Q_1^- \cup Q_2^+$

## Slightly trickier rules

Cartesian product:

$$\blacktriangleright (Q_1 \times Q_2)^+ = Q_1^+ \times Q_2^+$$

$$\blacktriangleright (Q_1 \times Q_2)^- = Q_1^- \times \text{adom}^{\text{arity}(Q_2)} \cup \text{adom}^{\text{arity}(Q_1)} \times Q_2^-$$

Projection:

$$\blacktriangleright (\pi_{\alpha}(Q))^+ = \pi_{\alpha}(Q^+)$$

$$\blacktriangleright (\pi_{\alpha}(Q))^- = \pi_{\alpha}(Q^-) - \pi_{\alpha}(\text{adom}^{\text{arity}(Q)} - Q^-)$$

## The last bit: selection

Translate conditions  $\theta \mapsto \theta^*$ :

- ▶  $(A = B)^* = (A = B)$ .
- ▶  $(A = \text{const})^* = (A = \text{const})$ .
- ▶  $(A \neq B)^* = (A \neq B) \wedge \text{NotNull}(A, B)$ .
- ▶  $(A \neq \text{const})^* = (A \neq \text{const}) \wedge \text{NotNull}(A)$ .
- ▶  $(\theta_1 \vee \theta_2)^* = \theta_1^* \vee \theta_2^*$ .
- ▶  $(\theta_1 \wedge \theta_2)^* = \theta_1^* \wedge \theta_2^*$ .

Translate selections:

- ▶  $(\sigma_\theta(Q))^+ = \sigma_{\theta^*}(Q^+)$
- ▶  $(\sigma_\theta(Q))^- = Q^- \cup \sigma_{(-\theta)^*}(\text{adom}^{\text{arity}(Q)})$

## Correctness and efficiency

Translations have correctness guarantees.

## Correctness and efficiency

Translations have correctness guarantees.

**Problem** Some of the rules are not very efficient:

$$(\sigma_{\theta}(Q))^{-} = Q^{-} \cup \sigma_{(-\theta)^{*}}(\text{adom}^{\text{arity}(Q)})$$

$$(Q_1 \times Q_2)^{-} = Q_1^{-} \times \text{adom}^{\text{arity}(Q_2)} \cup \text{adom}^{\text{arity}(Q_1)} \times Q_2^{-}$$

$$(\pi_{\alpha}(Q))^{-} = \pi_{\alpha}(Q^{-}) - \pi_{\alpha}(\text{adom}^{\text{arity}(Q)} - Q^{-})$$

## Efficiency: solution

We defined a **family** of translations.

Taking a smaller, with respect to containment, query **preserves correctness** guarantees.

For example:

$$(\sigma_{\theta}(Q))^{-} = Q^{-}$$

$$(Q_1 \times Q_2)^{-} = Q_1^{-} \times Q_2^{-}$$

$$(\pi_{\alpha}(Q))^{-} = \emptyset$$

## Some extensions

- ▶ A dual approach: no false negatives
  - ▶ just take negation twice
- ▶ Other derived useful operations of relational algebra
  - ▶ especially for implementing correlated subqueries
  - ▶ semi-join, anti-join
  - ▶ explain why SQL's handling of **EXCEPT** and nested queries for implementing difference is not the same

## What is so special about 3VL?

Is it the only possibility?

Perhaps we can find a smarter procedure based on the usual **two-valued** logic that has correctness and efficiency guarantees?

If not, can there be other many-valued logics that give us correctness?  
And if so, what makes them such?



We need  $\geq 3$  values. But there are many choices.

(new results with Marco Console and Paolo Guagliardo)

We need  $\geq 3$  values. But there are many choices.

(new results with Marco Console and Paolo Guagliardo)

2 values do not work.

- ▶ every procedure based on Boolean 2VL gives both false positives and false negatives.

We need  $\geq 3$  values. But there are many choices.

(new results with Marco Console and Paolo Guagliardo)

2 values do not work.

- ▶ every procedure based on Boolean 2VL gives both false positives and false negatives.

Lots of many-valued logics will work.

- ▶ What makes it work: **monotonicity** of  $\wedge, \vee, \neg$
- ▶ Many-valued logics have the **truth ordering**  $0 \leq \frac{1}{2} \leq 1$  and the **knowledge ordering**  $\frac{1}{2} \preceq 0$  and  $\frac{1}{2} \preceq 1$
- ▶  $\wedge, \vee, \neg$  are monotone with respect to  $\preceq$
- ▶ Every such many-valued logic can be lifted to an efficient and correct procedure for all relational calculus queries.

We need  $\geq 3$  values. But there are many choices.

(new results with Marco Console and Paolo Guagliardo)

2 values do not work.

- ▶ every procedure based on Boolean 2VL gives both false positives and false negatives.

Lots of many-valued logics will work.

- ▶ What makes it work: **monotonicity** of  $\wedge, \vee, \neg$
- ▶ Many-valued logics have the **truth ordering**  $0 \leq \frac{1}{2} \leq 1$  and the **knowledge ordering**  $\frac{1}{2} \preceq 0$  and  $\frac{1}{2} \preceq 1$
- ▶  $\wedge, \vee, \neg$  are monotone with respect to  $\preceq$
- ▶ Every such many-valued logic can be lifted to an efficient and correct procedure for all relational calculus queries.

3VL. The logic of choice for most – but not all – applications.

## What's next

Implement it.

- ▶ Theoretical complexity is good but too many cartesian products.
- ▶ Hence approximations.
- ▶ The price of correctness: it will be slower, but how much slower?
- ▶ Handle at the level of algebra? SQL?

## References

- ▶ A. Gheerbrant, L. Libkin, C. Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM TODS* 39(4): (2014)
  - ▶ (when  $\text{Eval}_{\text{FO}}$  works as-is)
- ▶ L Libkin, Certain answers as objects and knowledge. *KR 2014*
  - ▶ (a framework for providing correctness guarantees)
- ▶ L. Libkin. SQL's three-valued logic and certain answers. *ICDT 2015*: 94-109.
  - ▶ (the new 3VL procedure with correctness guarantees)
- ▶ M. Console, P. Guagliardo, L. Libkin. Computing certain answers efficiently with many-valued logic.
  - ▶ (new results on many-valued logics)

## References

- ▶ A. Gheerbrant, L. Libkin, C. Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM TODS* 39(4): (2014)
  - ▶ (when  $\text{Eval}_{\text{FO}}$  works as-is)
- ▶ L Libkin, Certain answers as objects and knowledge. *KR 2014*
  - ▶ (a framework for providing correctness guarantees)
- ▶ L. Libkin. SQL's three-valued logic and certain answers. *ICDT 2015*: 94-109.
  - ▶ (the new 3VL procedure with correctness guarantees)
- ▶ M. Console, P. Guagliardo, L. Libkin. Computing certain answers efficiently with many-valued logic.
  - ▶ (new results on many-valued logics)

Questions?