SQL's Logic of Incompleteness: Can It Be Fixed?

Leonid Libkin (University of Edinburgh)

DBMSs and incomplete information

For most data processing tasks, we still use commercial DBMSs for storing and querying data.

These are mainly relational products from IBM, Oracle, Microsoft, and the likes; this $25B/year business is doing well.

Hence for lots of data processing tasks we still use SQL – that committee-designed reincarnation of first-order logic.

But even for what we view as first-order queries, SQL is actually more than that, when it comes to handling incomplete information.

SQL's handling of incompleteness is problematic

"... 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'

SQL example

Orders

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

Payments

pay_id	order_id	amount
p1	ord1	–
p2	–	$50

SQL example

Orders

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

Payments

pay_id	order_id	amount
p1	ord1	–
p2	–	$50

Query: unpaid orders:

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

SQL example

Orders

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

Payments

pay_id	order_id	amount
p1	ord1	–
p2	–	$50

Query: unpaid orders:

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

Answer: EMPTY!

SQL example

Orders

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

Payments

pay_id	order_id	amount
p1	ord1	–
p2	–	$50

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!
- This is cast in stone (SQL standard).

What it's blamed on: 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:

$$\text{unk} \vee \text{unk} = \text{unk} \qquad \text{unk} \vee \text{true} = \text{true} \qquad \text{unk} \wedge \text{unk} = \text{unk}$$
$$\text{unk} \wedge \text{false} = \text{false} \qquad \neg\, \text{unk} = \text{unk} \qquad \text{etc}$$

- 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.

The *real* source of the problem

Can SQL evaluation and certain answers be the same?

No!

The *real* source of the problem

Can SQL evaluation and certain answers be the same?
No!

Complexity argument:

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

The *real* source of the problem

Can SQL evaluation and certain answers be the same?
No!

Complexity argument:

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

- So perhaps it's not that bad after all?
- We need to approximate answers that are hard to find; of course we'll miss some.
- Let's see what else can go wrong.

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 the example we've seen only has 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: $R =$ | A | / | 1 | $\qquad S =$ | A | / | null |

Query $R - S$:

```
SELECT R.A FROM R
WHERE NOT EXISTS (SELECT * FROM S WHERE R.A=S.A)
```

Certain answer: $\emptyset$ SQL answer: 1

Is there a Boolean solution?

Perhaps the committee design missed something and we don't need 3VL?

Actually, we do....

> Theorem (Console, Guagliardo, L.) Every query evaluation that uses the Boolean semantics for $\wedge, \vee, \neg$ generates false positives on databases with nulls.

Fixing the 3VL semantics

- Some of the rules for handling true, false, and unknown are quite arbitrary.

Fixing the 3VL semantics

- 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.

Fixing the 3VL semantics

- 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 $\bar{a}$

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

- 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 $\vee, \wedge, \neg, \forall, \exists$:

$$
\begin{aligned}
\mathsf{Eval}(D, Q \vee Q') &= \max(\mathsf{Eval}(D, Q), \mathsf{Eval}(D, Q')) \\
\mathsf{Eval}(Q \wedge Q', D) &= \min(\mathsf{Eval}(D, Q), \mathsf{Eval}(D, Q')) \\
\mathsf{Eval}(D, \neg Q) &= 1 - \mathsf{Eval}(D, Q) \\
\mathsf{Eval}(D, \exists x\ Q(x, \bar{a})) &= \max\{\mathsf{Eval}(D, Q(a', \bar{a})) \mid a' \in \mathit{adom}(D)\} \\
\mathsf{Eval}(D, \forall x\ Q(x, \bar{a})) &= \min\{\mathsf{Eval}(D, Q(a', \bar{a})) \mid a' \in \mathit{adom}(D)\}
\end{aligned}
$$

FO evaluation procedure

We only need to give rules for atomic formulae.

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

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

SQL evaluation procedure

All that changes is the rule for comparisons.

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

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

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

$h(\perp_1) = 4$
$h(\perp_2) = 3$
$h(\perp_3) = 5$
$\Longrightarrow$

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

$h(\perp_1) = 4$
$h(\perp_2) = 3$
$h(\perp_3) = 5$
$\Longrightarrow$

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?

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

We want at least simple correctness guarantees

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

Correctness: what do we know?

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

We want at least simple correctness guarantees

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

Sometimes we want/get even more:

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

Languages for correctness

UCQ: unions of conjunctive queries, or positive relational algebra $\pi, \sigma, \bowtie, \cup$.

$\mathsf{FO}_{\mathsf{certain}}$ — UCQs extended with the formation rule

$$\forall \bar{y}\ (\mathsf{atom}(\bar{y}) \rightarrow \varphi(\bar{x}, \bar{y}))$$

(Gheerbrant, L., Sirangelo) For $\mathsf{FO}_{\mathsf{certain}}$ queries,

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

For UCQs,

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

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 $h(\bar{t}_1) = h(\bar{t}_2)$.

$$\begin{pmatrix} 1 & \bot & 1 & 3 \\ \bot' & 2 & \bot' & 3 \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & 2 & 1 & 3 \end{pmatrix}$$

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

This can be checked in linear time.

Proper 3-valued procedure

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

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

Simple correctness guarantees: no false positives

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

Simple correctness guarantees:

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

Thus:

- Fast evaluation (checking $\mathsf{Eval}_{3v}(D, Q(\bar{a})) = 1$ 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 (proposed by Lipski in 1984 but quickly forgotten).

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

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

Strong correctness guarantees: involving nulls

How can we give correctness guarantees for tuples with nulls? By a natural extension of the standard definition (proposed by Lipski in 1984 but quickly forgotten).

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

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

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:

- $\mathrm{Eval}_{3v}(D, Q(\bar{a})) = 1 \implies \bar{a} \in \mathrm{certain}_{\perp}(Q, D)$
- $\mathrm{Eval}_{3v}(D, Q(\bar{a})) = 0 \implies \bar{a} \in \mathrm{certain}_{\perp}(\neg Q, D)$

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

Relational algebra queries

This is how it will be implemented after all.

Why not

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

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

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

with certainty guarantees, i.e.

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

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:

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

Projection:

- $(\pi_{\boldsymbol{\alpha}}(Q))^+ = \pi_{\boldsymbol{\alpha}}(Q^+)$
- $(\pi_{\boldsymbol{\alpha}}(Q))^- = \pi_{\boldsymbol{\alpha}}(Q^-) - \pi_{\boldsymbol{\alpha}}(adom^{\mathsf{arity}(Q)} - Q^-)$

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:

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

What is so special about 3VL?

At the first glance, not much (with Marco Console and Paolo Guagliardo)

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.
- The procedure shown above is just one example, for 3VL.

Another example: 4VL

An extra truth value **s** – sometimes

Meaning: we know for sure it's not certainly true nor certainly false

$$\bar{a} \in Q(D') \quad \text{and} \quad \bar{a} \notin Q(D'') \quad \text{for some} \quad D, D'' \in [\![D]\!]$$

Computing $R - S$ for $R = \{1\}$ and $S = \{\bot\}$:

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

SQL and Eval_{3v} assign unk to 1 but now we can assign **s**.

Gives extra information but at a price (some optimization rules don't apply)

Many-valued framework

We need:

- ordering on truth values
- proper many-valued semantics (never defined before) including both
  - semantics of queries and
  - semantics of query answers (viewed as databases)
- By doing this, we ensure the basic principle holds:
  - additional knowledge about the input translates into additional knowledge about the output

This framework both explains what happens with 3VL, and lets us lift logics to evaluation procedures.

Does the 3VL translation work?

It does! (ongoing work with Paolo Guagliardo)

Setup:

- Take the standard database benchmark TPC-H
- generate decent size instances (up to 2GB) and throw in nulls (between 1% and 10% of values)
- Take some TPC-H queries involving negation

Results:

- False positives are everywhere (for some queries, nearly all results are false) – hence the problem is real
- Translations don't slow down queries much:
  - for most queries between 1% and 4% of execution time
  - and sometimes they even improve things

What's next

- What does it take to introduce SELECT CERTAIN into SQL?
- Applications:
  - data integration
  - data exchange
  - consistent query answering
- Dealing with the open-world semantics (trickier!)
  - Crucial application: OBDA
- Other data models: XML, graphs, key-value stores.

When things are easy: UCQs with inequalities

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

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

Then, for UCQs with inequalities

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

Corollary: correctness guarantees for $\mathrm{Eval}_{\mathrm{UCQ}^{\neq}}$ 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

$h(\perp_1) = 4$
$h(\perp_2) = 3$
$h(\perp_3) = 5$
$\Longrightarrow$

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

$h(\perp_1) = 4$
$h(\perp_2) = 3$
$h(\perp_3) = 5$
$\Longrightarrow$

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

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

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

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

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