Incomplete Information

Dr Paolo Guagliardo

dbs-lecturer@ed.ac.uk

THE UNIVERSITY of EDINBURGH
informatics

Fall 2018

This page is intentionally left blank

Motivation

In today's world data is collected in many ways:

- Acquired through sensors
- Exchanged between enterprises
- Crawled from the web
- Automatically generated by software
- ...

We almost never have complete knowledge

We need to accept as a fact of life that

some information will be missing

Outline

Theory of incomplete information

- What is incomplete information?
- Which queries can be **evaluated correctly** on incomplete data?
- What does "correctly" mean?

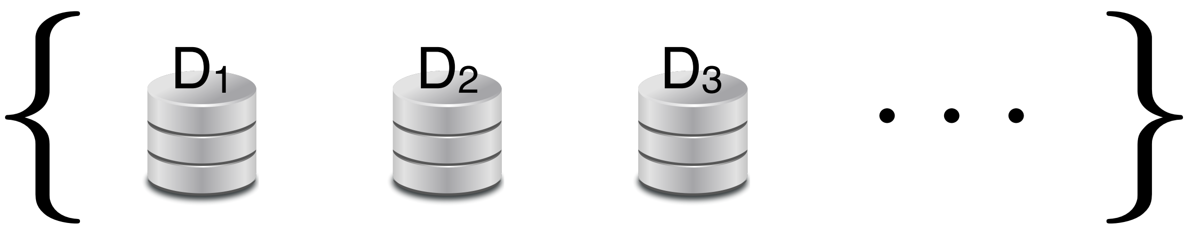
Incomplete information in SQL

- Very coarse and rudimental
- Leads to inconsistent answers
- Can we correctly evaluate queries?

Theory of incomplete information

Incomplete database

A (possibly infinite) **set of possible worlds**



Possible world = **complete database** with perfect information

Representation system

A pair $(\mathbb{D}, [\![\,]\!])$ where

- $\mathbb{D}$ is a set of "objects" (representations)
- $[\![\,]\!]$ maps elements of $\mathbb{D}$ to **sets of possible worlds**
  (interprets each representation as an incomplete database)

For $D \in \mathbb{D}$

$$[\![D]\!] = \Bigg\{$$

$$\cdots \Bigg\}$$

Naive tables

Populated by two kinds of elements:

- constants (usual database values)
- nulls (existential variables)

Each null represents a **currently unknown** value

Name	**Age**
Jane	$\perp_1$
John	$\perp_2$
Mark	$\perp_1$
Mary	27

Intuition

- Age of Jane, John and Mark is unknown
- Jane and Mark have the same age

Valuation

A map v from nulls to constants

Name	Age
Jane	$\bot_1$
John	$\bot_2$
Mark	$\bot_1$
Mary	27

$$\xrightarrow{v = \{\bot_1 \mapsto 20,\ \bot_2 \mapsto 31\}}$$

Name	Age
Jane	20
John	31
Mark	20
Mary	27

Replacing nulls with constants yields a complete database

Semantics of incompleteness (1)

Let D be a naive database (= a finite set of naive tables)

Closed-world assumption (CWA)

$$[\![D]\!]_{\text{CWA}} = \{\, v(D) \mid v \text{ is a valuation} \,\}$$

Each possible world is obtained by replacing nulls with constants

- Restores **information already present** in the database
- Most common semantics in many database applications

Semantics of incompleteness (2)

Open-world assumption (OWA)

$$[\![D]\!]_{\mathrm{OWA}} = \{\, v(D) \cup D' \mid v \text{ is a valuation, } D' \text{ is complete} \,\}$$

Each possible world is obtained by

1. replacing nulls with constants
2. possibly adding new (constant) tuples

- Open to the addition of new facts
- Common in applications involving reasoning (e.g., OBDA)

Semantics of incompleteness (3)

Weak closed-world assumption (WCWA)

$$[\![D]\!]_{\mathrm{CWA}} = \{\, v(D) \cup D' \mid v \text{ is a valuation, } D' \text{ is complete, } \mathrm{adom}(D') \subseteq \mathrm{adom}\big(v(D)\big) \,\}$$

Each possible world is obtained by

1. replacing nulls with constants
2. possibly adding new tuples
   that **use constants already stored in the database**

For every naive database D

$$[\![D]\!]_{\mathrm{CWA}} \subseteq [\![D]\!]_{\mathrm{WCWA}} \subseteq [\![D]\!]_{\mathrm{OWA}}$$

Answering queries on incomplete databases

An incomplete database is a **set of complete databases**

represented by an object D (e.g., naive tables)

via an interpretation $[\![\ ]\!]$ (e.g., $\textsc{cwa}$)

Question: What is the correct way of answering a query Q on D?

- We know how to answer Q on complete databases, so we let

$$Q\big([\![D]\!]\big) \;=\; \{\, Q(D') \mid D' \in [\![D]\!] \,\}$$

- The answer to Q on D is an object T (if it exists) such that

$$[\![T]\!] \;=\; Q\big([\![D]\!]\big)$$

Strong representation systems

Definition

$\big(\mathbb{D}, [\![\ ]\!]\big)$ is a **strong representation system** for a query language $\mathcal{L}$ if for every $D \in \mathbb{D}$ and every $Q \in \mathcal{L}$ there exists $T \in \mathbb{D}$ such that

$$[\![T]\!] \;=\; \{\, Q(D') \mid D' \in [\![D]\!] \,\}$$

$$\begin{tikzcd}
 & D_1 \arrow[rr, "Q"] & & T_1 & \\
 & D_2 \arrow[rr, "Q"] & & T_2 & \\
D \arrow[uur] \arrow[ur] \arrow[dr] \arrow[ddr] & & & & T \arrow[uul] \arrow[ul] \arrow[dl] \arrow[ddl] \\
 & D_3 \arrow[rr, "Q"] & & T_3 & \\
 & \vdots & & \vdots &
\end{tikzcd}$$

Bad news

Naive tables are not a strong representation system for RA under CWA

Consider $D: \left\{ \begin{array}{c} R \\ \begin{array}{|c|c|} \hline \textbf{A} & \textbf{B} \\ \hline 1 & 2 \\ \bot & 4 \\ \hline \end{array} \end{array} \right\}$ and $Q\colon\ \sigma_{A=3}(R)$

Suppose there exists T such that $[\![T]\!] = \{Q(D') \mid D' \in [\![D]\!]\}$

$$[\![D]\!] = \left\{ D_1\colon \left\{ \begin{array}{c} R \\ \begin{array}{|c|c|} \hline \textbf{A} & \textbf{B} \\ \hline 1 & 2 \\ 2 & 4 \\ \hline \end{array} \end{array} \right\},\quad D_2\colon \left\{ \begin{array}{c} R \\ \begin{array}{|c|c|} \hline \textbf{A} & \textbf{B} \\ \hline 1 & 2 \\ 3 & 4 \\ \hline \end{array} \end{array} \right\} \quad \cdots \right\}$$

$Q(D_1) = \varnothing$ and $Q(D_2) = \{(3,4)\}$

Therefore T cannot exist because $\varnothing \in [\![T]\!] \iff T = \varnothing$

What can we do?

We have two options:

1. Increase our firepower
   by using more powerful representation systems

   $\Longrightarrow$ **Conditional tables**

2. Lower our expectations
   by weakening the notion of correct answers

   $\Longrightarrow$ **Certain answers**

Conditional tables

Naive tables where each tuple is associated with a condition

Conditions are Boolean combinations of equalities and inequalities between constants and nulls

A	B	condition
0	3	$\bot_1 > 1$
1	1	
$\bot_1$	3	$\bot_3 = 0$
$\bot_2$	4	$\bot_3 = 1$
5	$\bot_1$	$\bot_1 \neq 5 \vee \bot_2 = 1$

Conditional tables: semantics

Applying a valuation v on a c-table T gives

$$v(T) = \{v(\bar{a}) \mid (\bar{a}, \phi) \in T,\ v \models \phi\}$$

A	B	condition
0	3	$\bot_1 > 1$
1	1	
$\bot_1$	3	$\bot_3 = 0$
$\bot_2$	4	$\bot_3 = 1$
5	$\bot_1$	$\bot_1 \neq 5 \vee \bot_3 = 1$

$$\xrightarrow{v = \{\bot_i \mapsto 1\}}$$

A	B
1	1
1	4
5	1

$[\![\ ]\!]_{\mathrm{CWA}}$ and $[\![\ ]\!]_{\mathrm{OWA}}$ are defined as before

Conditional tables: results

Conditional tables are a strong representation system
for relational algebra queries under CWA (but not under OWA)

For every conditional database D and every RA query Q
there exists a c-table T such that

$$[\![T]\!]_{\text{CWA}} \;=\; \{\, Q(D') \mid D' \in [\![D]\!] \}$$

T can be obtained via conditional evaluation of Q on D

Conditional evaluation

$$\pi_X(T) = \{(\bar{a}[X], \phi) \mid (\bar{a}, \phi) \in T\}$$

$$\sigma_\theta(T) = \{(\bar{a}, \phi \wedge \theta(\bar{a})) \mid (\bar{a}, \phi) \in T\}$$

$$T_1 \times T_2 = \{(\bar{a}_1\bar{a}_2, \phi_1 \wedge \phi_2) \mid (\bar{a}_1, \phi_1) \in T_1, (\bar{a}_2, \phi_2) \in T_2\}$$

$$T_1 \cup T_2 = \{(\bar{a}, \phi) \mid (\bar{a}, \phi) \in T_1 \text{ or } (\bar{a}, \phi) \in T_1\}$$

$$T_1 - T_2 = \{(\bar{a}_1, \phi_1 \wedge \phi_{\bar{a}_1}) \mid (\bar{a}_1, \phi_1) \in T_1\}$$

$$\text{where } \phi_{\bar{a}_1} = \bigwedge_{(\bar{a}_2, \phi_2) \in T_2} \neg(\phi_2 \wedge \bar{a}_1 = \bar{a}_2)$$

Query processing becomes quite complicated

Certain answers

For a query Q and an incomplete database D, defined as

$$\mathsf{cert}(Q, D) = \bigcap_{D' \in [\![D]\!]} Q(D')$$

Answers to Q in every possible world represented by D

Standard approach, used in all applications:

- data integration
- data exchange
- inconsistent data
- querying with ontologies
- data cleaning
- ...

Weak representation systems

Definition

$(\mathbb{D}, [\![\,]\!])$ is a **weak representation system** for a query language $\mathcal{L}$ if for every $D \in \mathbb{D}$ and every $Q \in \mathcal{L}$

$$Q(D) \;=\; \mathsf{cert}(Q, D)$$

Intuition

We can get the certain answers by evaluating the query directly on the representation D

Naive evaluation

1. Evaluate Q on D by treating nulls as regular values
   - $\bot_i = c$ evaluates to false
   - $\bot_i = \bot_j$ is true iff $\bot_i$ and $\bot_j$ are the same element

2. Discard tuples with nulls from $Q(D)$

Does this give us cert(Q, D)?

Yes if Q is a UCQ (both under CWA and OWA)

Problems arise with negation

Naive evaluation does **not** give the certain answers

D: R

Name	**City**
Jane	$\bot_1$
John	$\bot_2$
Mary	London

Q: $\pi_{\text{Name}}\big(\sigma_{\text{City}\neq\text{London}}(R)\big)$

$Q(D) = \{\text{Jane, John}\}$

but cert$(Q, D) = \varnothing$

Computing certain answers

Input: a database D and a tuple $\bar{a}$

Output: yes if $\bar{a} \in \mathsf{cert}(Q, D)$, no otherwise

For relational algebra queries:

coNP-complete under CWA

- it is in coNP: just guess $D' \in [\![D]\!]_{\text{CWA}}$ so that $\bar{a} \notin Q(D)$
- it is complete for coNP: 3-colourability

Undecidable under OWA

- the same as validity problem in logic (undecidable)
- but can be solved efficiently (polynomial time) for simpler classes of queries

Theory of incomplete information: Summary

- Simple representation: naive tables
  but we cannot even evaluate simple selections over them
- With conditional tables we can evaluate all RA queries
  but query processing becomes cumbersome
- If we settle for just certain answers and use naive tables
  we can evaluate UCQs
- **Tradeoff**: Semantic correctness vs Complexity of queries

Incomplete information in SQL

SQL and incomplete data

> *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

Wrong answers in SQL

ORDERS

ord_id	ord_date
ord1	2015-06-12
ord2	2015-07-11
ord3	2015-07-20

PAYMENTS

pay_id	ord_id	pay_date
pay1	ord1	2015-06-14
pay2	**NULL**	2015-07-25

A typical query we teach students to write: "Unpaid orders"

```
SELECT O.ord_id
FROM   Orders O
WHERE  NOT EXISTS
     ( SELECT *
       FROM   Payments P
       WHERE  P.ord_id=O.ord_id )
```

Answer: $\{$**ord2**, **ord3**$\}$ but there are **no certain answers**

SQL and correctness

Wrong answers = answers that SQL returns but are not certain

- For UCQ$^{\neq}$ there are no wrong answers
- Problems arise in queries with **difference**

Can SQL evaluation give precisely the certain answers? **No**

- Finding certain answers for RA queries is coNP-hard
- SQL evaluation is very efficient (AC^0)

We need to settle for an **approximation** (which SQL does not provide)

Questions

- Are we addressing a **real** problem?
  Do real-life SQL queries produce non-certain answers?
- The existing solution works well in theory
  but does it work in **practice**?
- If it doesn't, **why**? And what can be done?
- How good is a practically applicable solution?
  Hope for a **reasonable slowdown** over SQL

Are wrong answers a real problem?

Experiment on the TPC-H Benchmark:
models a business scenario with associated decision support queries

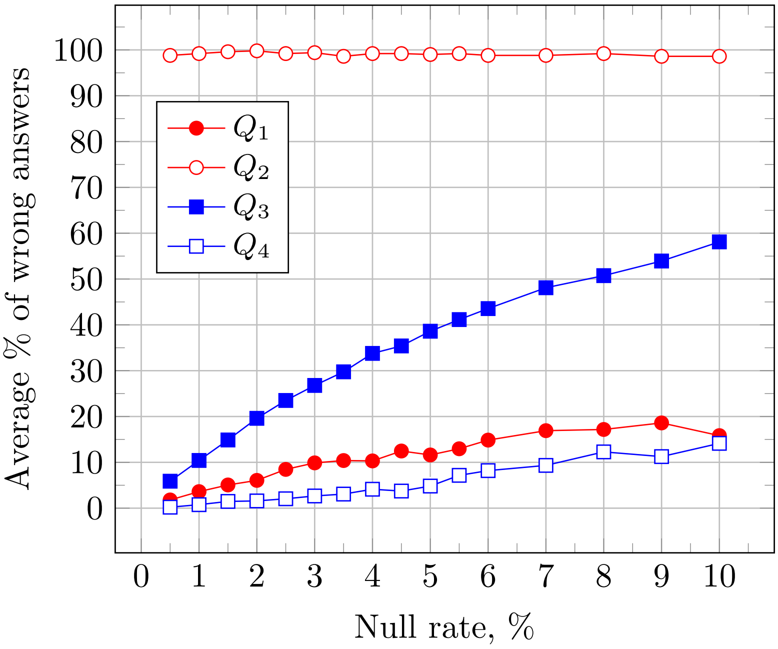
Issues

- The TPC-H data generator **does not produce nulls**
  $\Longrightarrow$ Generate nulls randomly in nullable attributes
- Only 2 queries have **difference** (**`NOT EXISTS`**)
  $\Longrightarrow$ Complement with 2 typical textbook queries
- Computing certain answers is **coNP-hard**
  $\Longrightarrow$ Design ad-hoc algorithms to detect wrong answers

Wrong answers: Lots of them

Horizontal axis: null rate (probability that a null occurs in an attribute not declared as **NOT NULL**)

Vertical axis: lower bound on the percentage of wrong answers



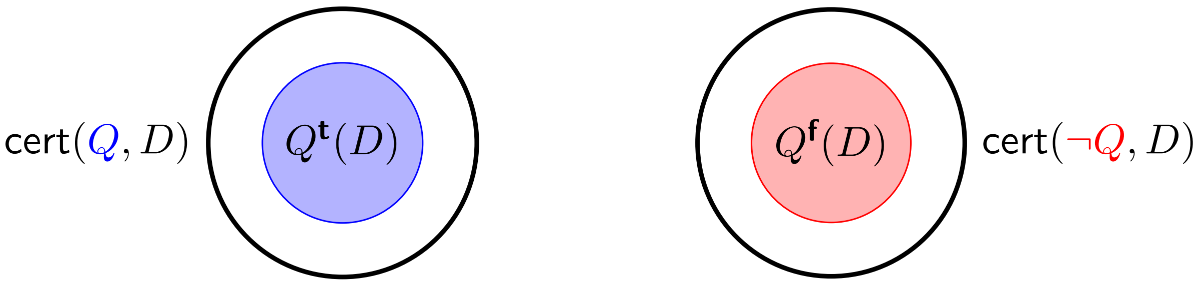
A simple approximation scheme

There is an effective translation of queries

$$Q \quad \mapsto \quad (\, Q^{\mathbf{t}}, Q^{\mathbf{f}} \,)$$

such that:

- $Q^{\mathbf{t}}$ approximates certain answers to Q
- $Q^{\mathbf{f}}$ approximates certain answers to the negation of Q
- both queries have $\mathbf{AC}^0$ data complexity



Relational algebra translations: $Q^{\mathbf{t}}$

Libkin, "SQL's 3-valued logic and certain answers", ICDT'15

For a relation R: $\quad R^{\mathbf{t}} = R$

For op $\in \{\cap, \cup, \times\}$: $\quad (Q_1 \text{ op } Q_2)^{\mathbf{t}} = Q_1^{\mathbf{t}} \text{ op } Q_2^{\mathbf{t}}$

For projection: $\quad \pi_\alpha(Q)^{\mathbf{t}} = \pi_\alpha(Q^{\mathbf{t}})$

For difference: $\quad (Q_1 - Q_2)^{\mathbf{t}} = Q_1^{\mathbf{t}} \cap Q_2^{\mathbf{f}}$

For selection: $\quad \sigma_\theta(Q)^{\mathbf{t}} = \sigma_{\theta^*}(Q^{\mathbf{t}})$

$$\begin{aligned} \text{where} \quad (A = B)^* &= (A = B) \\ (A \neq B)^* &= (A \neq B) \wedge \text{not\_null}(A) \wedge \text{not\_null}(B) \\ (\theta_1 \text{ op } \theta_2)^* &= \theta_1^* \text{ op } \theta_2^* \qquad \text{for op} \in \{\wedge, \vee\} \end{aligned}$$

Relational algebra translations: $Q^{\mathbf{f}}$

Libkin, "SQL's 3-valued logic and certain answers", ICDT'15

$$\begin{aligned} R^{\mathbf{f}} &= \{\bar{r} \in \text{adom}^{\text{ar}(R)} \mid \bar{r} \text{ does not match any tuple in } R\} \\ (Q_1 \cup Q_2)^{\mathbf{f}} &= Q_1^{\mathbf{f}} \cap Q_2^{\mathbf{f}} \\ (Q_1 \cap Q_2)^{\mathbf{f}} &= Q_1^{\mathbf{f}} \cup Q_2^{\mathbf{f}} \\ (Q_1 - Q_2)^{\mathbf{f}} &= Q_1^{\mathbf{f}} \cup Q_2^{\mathbf{t}} \\ (\sigma_\theta(Q))^{\mathbf{f}} &= Q^{\mathbf{f}} \cup \sigma_{(\neg\theta)^*}(\text{adom}^{\text{ar}(Q)}) \\ (Q_1 \times Q_2)^{\mathbf{f}} &= Q_1^{\mathbf{f}} \times \text{adom}^{\text{ar}(Q_2)} \cup \text{adom}^{\text{ar}(Q_1)} \times Q_2^{\mathbf{f}} \\ (\pi_\alpha(Q))^{\mathbf{f}} &= \pi_\alpha(Q^{\mathbf{f}}) - \pi_\alpha(\text{adom}^{\text{ar}(Q)} - Q^{\mathbf{f}}) \end{aligned}$$

Does it work in practice?

Not a chance: With as few as 1000 tuples and 3 attributes bad queries start computing relations with **billions** of tuples!

Inefficient translations

$$R^{\mathsf{f}} = \left\{ \bar{r} \in \mathsf{adom}^{\mathsf{ar}(R)} \mid \bar{r} \text{ does not match any tuple in } R \right\}$$

$$(\sigma_\theta(Q))^{\mathsf{f}} = Q^{\mathsf{f}} \cup \sigma_{(\neg\theta)^*}\left(\mathsf{adom}^{\mathsf{ar}(Q)}\right)$$

$$(Q_1 \times Q_2)^{\mathsf{f}} = Q_1^{\mathsf{f}} \times \mathsf{adom}^{\mathsf{ar}(Q_2)} \cup \mathsf{adom}^{\mathsf{ar}(Q_1)} \times Q_2^{\mathsf{f}}$$

$$\left(\pi_\alpha(Q)\right)^{\mathsf{f}} = \pi_\alpha(Q^{\mathsf{f}}) - \pi_\alpha\left(\mathsf{adom}^{\mathsf{ar}(Q)} - Q^{\mathsf{f}}\right)$$

With the best tricks we can only handle a few hundred tuples:

AC^0 and efficiency are **NOT** the same!

Let's rethink the basics

We only needed Q^{f} to handle **difference**: $(Q_1 - Q_2)^{\mathsf{t}} = Q_1^{\mathsf{t}} \cap Q_2^{\mathsf{f}}$

Intuition: A tuple is for sure in $Q_1 - Q_2$ if

- it is certainly in Q_1 and
- it is certainly not in Q_2

This is not the only possibility

A tuple is for sure in $Q_1 - Q_2$:

- it is certainly in Q_1 and
- it **does not match** any tuple that **could be in** Q_2

What is "match"?

Unification: Two tuples **unify** if there is an instantiation of nulls with constants that makes them equal

Left unification semijoin

$$R \ltimes_{\Uparrow} S = \{\, \bar{r} \in R \mid \exists \bar{s} \in S \colon \bar{s} \text{ unifies with } \bar{r} \}$$

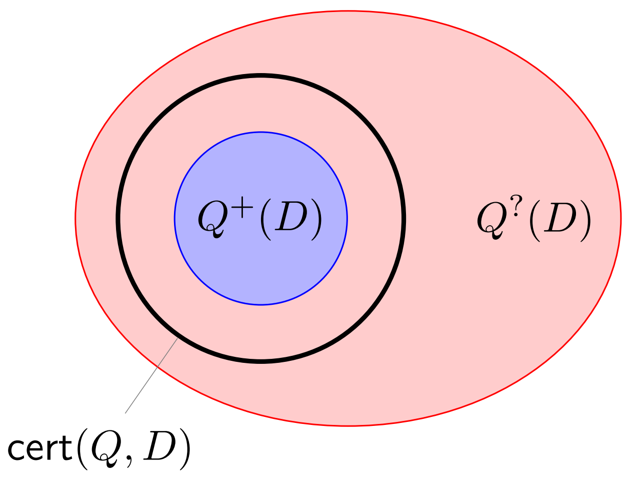
Left unification antijoin

$$R \,\overline{\ltimes}_{\Uparrow}\, S = R - R \ltimes_{\Uparrow} S$$

New translation

Translate Q into $(\, Q^{+}, Q^{?} \,)$ where:

- Q^{+} approximates certain answers
- $Q^{?}$ **represents possible answers**



$$(Q_1 - Q_2)^{+} = Q_1^{+} \,\overline{\ltimes}_{\Uparrow}\, Q_2^{?}$$

$$R^{?} = R$$

$$(Q_1 \cup Q_2)^{?} = Q_1^{?} \cup Q_2^{?}$$

$$(Q_1 \cap Q_2)^{?} = Q_1^{?} \ltimes_{\Uparrow} Q_2^{?}$$

$$(Q_1 - Q_2)^{?} = Q_1^{?} - Q_2^{+}$$

$$\big(\sigma_{\theta}(Q)\big)^{?} = \sigma_{\neg(\neg\theta)^{*}}(Q^{?})$$

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

$$\big(\pi_{\alpha}(Q)\big)^{?} = \pi_{\alpha}(Q^{?})$$

Does it work in practice?

We ran our queries and translations on TPC-H instances with nulls and measured the relative runtime performance of Q^+ w.r.t. Q

- SQL was designed for **efficiency**
  $\Longrightarrow$ we cannot expect to outperform native SQL
- but we can hope for the **overhead** to be acceptable

We observed the following behaviors:

- **The good:** small overhead
  (less than $< 4\%$)
- **The fantastic:** significant speed-up
  (more than 10^3 times faster)
- **The tolerable:** moderate slow-down
  (half the speed on 1GB instances, a quarter on 10GB ones)

Summary

- The way SQL handles incomplete data is disastrous
  (wrong answers are a **real problem**)
- The only existing solution (prior to ours) works well in theory
  but **not at all in practice**
- There is hope for a **practically feasible** solution

What does it take to introduce **SELECT CERTAIN** in SQL?

- Deal with bags (and, later, aggregation)
- Direct SQL-to-SQL translations
- Incorporate constraints
- Query optimization with disjunctions