Principles of Managing Uncertain Data

Lecture 5: Incomplete Databases

SQL NULL

Incompleteness via Possible Worlds

Representation Systems

Query Answering

Table of Contents

1 SQL NULL

2 Incompleteness via Possible Worlds

3 Representation Systems

4 Query Answering

Table of Contents

SQL NULL

Incompleteness via Possible Worlds

Representation Systems

Query Answering

References

SQL NULL

NULL in Queries

3-Valued Logic

Table of Contents

1 SQL NULL

2 Incompleteness via Possible Worlds

3 Representation Systems

4 Query Answering

3-Valued Logic

Running Example

To properly support NULL, we need to understand how to treat NULL in queries

<table>
<thead>
<tr>
<th>Takes</th>
<th>course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>PL</td>
</tr>
<tr>
<td>Anat</td>
<td>AI</td>
</tr>
<tr>
<td>Alon</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Teaches</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>↓</td>
<td>Keren</td>
</tr>
<tr>
<td>AI</td>
<td>↓</td>
</tr>
</tbody>
</table>
Semantics of SQL NULL

What is the meaning of “$x > 5$ or $y < 2$” when $x$ is unknown?

- Comparison to an unknown value (NULL) is an unknown truth value
- The answer depends on $y$; there are different answers for $y = 1$ and $y = 3$
- SQL adapts the 3-valued logic

3 truth values: True (T), False (F) and Unknown (U)

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x \land y$</th>
<th>$x \lor y$</th>
<th>$\neg x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>U</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>F</td>
<td>F</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>U</td>
<td>U</td>
<td>T</td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

Follows the rules of $T = 1$, $F = 0$, $U = 1/2$, and then $\land = \min$, $\lor = \max$, $\neg x = (1 - x)$

Example Revisited

```
SELECT student
FROM Takes
WHERE course='PL' OR course!='PL'
```

Takes

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>PL</td>
</tr>
<tr>
<td>Anat</td>
<td>AI</td>
</tr>
<tr>
<td>Alon</td>
<td></td>
</tr>
</tbody>
</table>

Why so?

Another Example (1)

Example due to Leonid Libkin:

<table>
<thead>
<tr>
<th>Orders</th>
<th>Paid</th>
</tr>
</thead>
<tbody>
<tr>
<td>ord</td>
<td>price</td>
</tr>
<tr>
<td>o11</td>
<td>100</td>
</tr>
<tr>
<td>o22</td>
<td>200</td>
</tr>
<tr>
<td>o33</td>
<td>300</td>
</tr>
<tr>
<td>cust</td>
<td></td>
</tr>
<tr>
<td>c11</td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td></td>
</tr>
</tbody>
</table>

```
SELECT * FROM Orders O
WHERE NOT EXISTS (SELECT ord FROM Paid P
WHERE P.ord = O.ord)
```

empty

```
SELECT * FROM Orders O
WHERE NOT IN (SELECT ord FROM Paid)
```

```
SELECT * FROM Orders O
WHERE ord NOT IN
(SELECT ord FROM Paid P
WHERE P.ord = 0.ord)
```

Another Example (2)

Example due to Leonid Libkin:

<table>
<thead>
<tr>
<th>Orders</th>
<th>Paid</th>
</tr>
</thead>
<tbody>
<tr>
<td>ord</td>
<td>price</td>
</tr>
<tr>
<td>o11</td>
<td>100</td>
</tr>
<tr>
<td>o22</td>
<td>200</td>
</tr>
<tr>
<td>o33</td>
<td>300</td>
</tr>
</tbody>
</table>

```
SELECT * FROM Orders O
WHERE NOT IN
(SELECT ord FROM Paid)
```

empty

```
SELECT * FROM Orders O
WHERE ord NOT IN
(SELECT ord FROM Paid P
WHERE P.ord = 0.ord)
```

```
SELECT * FROM Orders O
WHERE ord NOT IN
(SELECT ord FROM Paid)
```

From Christopher J. Date

“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. In my opinion, this state of affairs is a complete showstopper.”
To avoid the semantic deficiencies of SQL nulls and 3-valued logic, database research has developed different approaches to modeling and handling missing information.

A principal approach is the possible-world semantics. An incomplete database represents a collection of possible completions; each is a “possible world.”

We will review the framework originally proposed in a seminal paper by Imielinski and Lipski [IL84], which has been commonly adopted and developed by database researchers.

Reminder: Basic Concepts

- A **schema** (without constraints) is a collection of relation schemas, each having a **name** and a **heading** (attribute list).
- A **relation** $r$ over a relation schema $R$ consists of a collection of **tuples** over the heading of $R$.
- An **instance** (database) $I$ over a schema $S$ associates with every relation symbol $R$ a relation $R_I$ over $R$.
- We denote by $\text{Inst}(S)$ the set of all the instances over $S$.

Incomplete Instance

An **incomplete instance** $\mathcal{I}$ over a schema $S$ is a representation of a set of ordinary instances over $S$; we denote this set by $[\mathcal{I}]$, and each $I \in [\mathcal{I}]$ is called a **possible world**.
v-Instances

- We assume two (countably infinite) disjoint sets:
  - \( \text{Const} \) is the set of constants that are used as (ordinary) values in relations
  - \( \text{Var} \) is a set of variables or labeled nulls
- A \( v \)-instance \( I \) over a schema \( S \) is defined similarly to an ordinary instance, except that a value can be from either constant or a variable
- A relation of a \( v \)-instance is called a \( v \)-table or a naive table

Valuation

- Let \( I \) be a \( v \)-instance
- We denote by \( \text{Var} \) the set of variables that occurs in \( I \)
- A valuation over \( I \) is a mapping \( \mu : \text{Var} \rightarrow \text{Const} \)
- If \( \mu \) is a valuation over \( I \), then we denote by \( \mu(I) \) the (ordinary) instance that is obtained from \( I \) by replacing every variable \( v \) with the constant \( \mu(v) \)

Semantics of \( v \)-Instances

- There are two conventional ways to interpret the meaning \( [I] \) of a \( v \)-instance \( I \) over a schema \( S \)
  - Under the closed-world assumption:
    \[ [I] = \{ \mu(I) \mid \mu \text{ is a valuation over } I \} \]
  - Under the open-world assumption:
    \[ [I] = \{ I' \in \text{Inst}(S) \mid \mu(I') \subseteq I \text{ for some valuation } \mu \text{ over } I \} \]
- By default, we assume the closed-world assumption

Example

\[
\begin{array}{c|c|c|c|c|c}
\text{student} & \text{course} & \text{teaches} & \text{course} & \text{lecturer} \\
\hline
\text{Ahuva} & \text{PL} & \text{PL} & \text{Eran} \\
\text{Anat} & \text{AI} & \text{AI} & \text{Keren} \\
\text{Alon} & \text{AI} & \text{AI} & \text{Roy} \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\text{Student} & \text{Course} & \text{Lecturer} & \text{Student} & \text{Course} \\
\hline
\text{Ahuva} & \text{AI} & \text{Al} & \text{Ahuva} & \text{PL} \\
\text{Anat} & \text{AI} & \text{Keren} & \text{Anat} & \text{AI} \\
\text{Alon} & \text{AI} & \text{Roy} & \text{Alon} & \text{AI} \\
\end{array}
\]
In the next lecture we will discuss IBM’s Clio project (part of DB2) that uses v-instances to represent the result of transforming a source database into a target schema.

A Codd-instance is a v-instance such that no variable occurs more than once.

This corresponds to SQL nulls.

A relation of a Codd-instance is called a Codd-table.

Semantics carries over from v-instances.

Before presenting conditional instances (c-instances), let us see a small motivating example.

Example

\[ I: \]

<table>
<thead>
<tr>
<th>Takes</th>
<th>student</th>
<th>course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>PL</td>
<td></td>
</tr>
<tr>
<td>Alon</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Teaches</th>
<th>course</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL</td>
<td>Eran</td>
<td></td>
</tr>
<tr>
<td>Al</td>
<td></td>
<td>Keren</td>
</tr>
</tbody>
</table>

\[ I': \]

<table>
<thead>
<tr>
<th>Takes</th>
<th>student</th>
<th>course</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>PL</td>
<td>Eran</td>
<td></td>
</tr>
<tr>
<td>Alon</td>
<td></td>
<td></td>
<td>Keren</td>
</tr>
</tbody>
</table>

This is the first example, where we wrote each \( I_i \) as 1.
Recall that in the case of a \( v \)-instance \( I' \) we have

\[
[I'] = \{ \mu(I') \mid \mu \text{ is a valuation over } I' \}
\]

For a \( c \)-instance \( I = (I', \Phi, \varphi) \), two differences:
- We restrict to the valuations \( \mu \) that satisfy \( \Phi \)
- Instead of \( \mu(I') \), we take \( \mu_c(I') \), which is similar to \( \mu(I') \) except that we first remove the facts \( f \) such that \( \mu \) violates \( \varphi(f) \)

In notation:

\[
[I] = \{ \mu_c(I') \mid \mu \text{ is a valuation over } I' \text{ such that } \mu \models \Phi \}
\]
Example Revisited

$I$:

\[
\begin{array}{ccc}
\text{student} & \text{course} & \text{lecturer} \\
Ahuva & 1 & Eran \\
Alon & 1 & Eran \\
\end{array}
\]

What about this instance?

\[
\begin{array}{ccc}
\text{student} & \text{course} & \text{lecturer} \\
Ahuva & DB & Eran \\
Alon & AI & Eran \\
\end{array}
\]

Query Answering

- Recall that a query $Q$ over a schema $S$ maps every $I \in \text{Inst}(S)$ into a relation over the heading of $Q$.
- Given $Q$ and $I$, we call the relation $Q(I)$ the result and each tuple $t \in Q(I)$ an answer.
- What is the semantics of evaluating a query $Q$ over an incomplete instance $I$?
- We will consider two standard concepts:
  - Incomplete answer (representing all possible results)
  - Certain answers

Possible Results

Querying an Incomplete Instance

The result of applying a query $Q$ over an incomplete instance $I$ is the set $(Q(I) | I \in \mathcal{I})$; we denote this result by $Q(\mathcal{I})$.

Representing the Possible Results

- Given a representation system REP and query language QL, we ask whether we can represent $Q(\mathcal{I})$ in REP for every $Q \in \text{QL}$.
- In this case we say that REP is a strong representation system for QL.

Example

\[
\begin{array}{ccc}
\text{student} & \text{course} & \text{lecturer} \\
Ahuva & 1 & PL \\
Ahuva & 2 & PL \\
Ahuva & 3 & Eran \\
\end{array}
\]

This set of possible worlds cannot be represented by any v-table (or Codd-table)!
In conclusion, Codd-instances and v-instances are not strong representation systems for any query language QL that includes natural join.

- e.g., CQs, UCQs, and RA

**Theorem**
The class of c-instances is a strong representation system for relational algebra.

- That is, for every c-instance $I$ and RA query $Q$ there exists a c-table $J$ such that $[J] = \{Q(I) \mid I \in [I]\}$

**Definition**
Given a query $Q$ and an incomplete instance $I$, we denote by $\text{Certain}_Q(I)$ the relation that consists of all the tuples that occur in every relation inside $Q(I)$; that is:

$$\text{Certain}_Q(I) \overset{\text{def}}{=} \bigcap \{Q(I) \mid I \in [I]\}$$

A tuple in $\text{Certain}_Q(I)$ is called a certain answer.

**Finding the Certain Answers over v-Instances**
- Let $I$ be a v-instance, and let $Q$ be a UCQ
  - Equivalently, let $Q$ be an expression in positive RA
  - The following algorithm computes the certain answers
    1. Evaluate $Q$ over $I$, assuming that each variable is a distinct constant
    2. Remove all the resulting tuples that contain variables (labeled nulls)
  - That’s it!
  - A proof is required on why it works ...
  - (Home assignment)

**Example**

<table>
<thead>
<tr>
<th>Takes</th>
<th>Teaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>student</td>
<td>course</td>
</tr>
<tr>
<td>Ahuva</td>
<td>12</td>
</tr>
</tbody>
</table>

$Q = \pi_{\text{student, lecturer}}(\text{Takes} \bowtie \text{Teaches})$

**What is $\text{Certain}_Q(I)$?**

<table>
<thead>
<tr>
<th>student</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>Eran</td>
</tr>
</tbody>
</table>

**References I**
End of lecture 5

Incomplete Databases