Incomplete Information: Null Values

• Often ruled out: not null in SQL.
• Essential when one integrates/exchanges data.
• Perhaps the most poorly designed and the most often criticized part of SQL:

  “… [this] topic cannot be described in a manner that is simultaneously both comprehensive and comprehensible.”

  “… those SQL features are not fully consistent; indeed, in some ways they are fundamentally at odds with the way the world behaves.”

  “A recommendation: avoid nulls.”

  “Use [nulls] properly and they work for you, but abuse them, and they can ruin everything”
Part I: theory of incomplete information

- What is incomplete information?
- Which relational operations can be evaluated correctly in the presence of incomplete information?
- What does “evaluated correctly” mean?

Part II: incomplete information in SQL

- Simplifies things too much.
- Leads to inconsistent answers.
- One needs to understand this for asking queries over integrated/exchanged data.
Sometimes we don’t have all the information

• Null is used if we don’t have a value for a given attribute.

• What could null possibly mean?
  ◦ Value exists, but is unknown at the moment.
  ◦ Value does not exist.
  ◦ There is no information.
Representing relations with nulls: Codd tables

- In Codd tables, we put distinct variables for null values:

\[
\begin{array}{ccc}
A & B & C \\
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
x & b_3 & c_3 \\
y & b_4 & c_4 \\
a_5 & z & c_5 \\
\end{array}
\]

- Semantics of a Codd table \( T \) is the set \( \text{POSS}(T) \) of all tables without nulls it can represent.

- That is, we substitute values for all variables.
Tables in $POSS(T)$

- Closed World Assumption:
  - simply replace each variable by a value
- Open World Assumption:
  - replace each variable by a value
  - and possibly add tuples
Querying Codd tables

• Suppose \( Q \) is a relational algebra, or SQL query, and \( T \) is a Codd table. What is \( Q(T) \)?

• We only know how to apply \( Q \) to usual relations, so we can find:

\[
\hat{Q}(T) = \{ Q(R) \mid R \in \text{POSS}(T) \}
\]

• If there were a Codd table \( T' \) such that \( \text{POSS}(T') = \hat{Q}(T) \), then we would say that \( T' \) is \( Q(T) \). That is,

\[
\text{POSS}(Q(T)) = \{ Q(R) \mid R \in \text{POSS}(T) \}
\]

• Question: Can we always find such a table \( T' \)?
Strong representation systems

- Let $L$ be a language (e.g. a fragment of relational algebra).
- Assume that for every query $Q$ in $L$, and every table $T$, we can find a table $T'$ so that
  \[ \text{POSS}(T') = \{ Q(R) \mid R \in \text{POSS}(T) \} \]
- Then $T'$ is the answer to $Q$ on $T$.
- If we can do it, we say that Codd tables form a strong representation system for $L$.
- **Bad news:** We may not have a strong representation system even for a small subset of relational algebra.
No strong representation system for Codd tables

Table: \[ T = \begin{array}{cc} A & B \\ 0 & 1 \\ x & 2 \end{array} \]

Query: \[ Q = \sigma_{A=3}(T) \]

Suppose there is \( T' \) such that \( \text{POSS}(T') = \{ Q(R) \mid R \in \text{POSS}(T) \} \).
Consider:

\[ R_1 = \begin{array}{cc} A & B \\ 0 & 1 \\ 2 & 2 \end{array} \quad \text{and} \quad R_2 = \begin{array}{cc} A & B \\ 0 & 1 \\ 3 & 2 \end{array} \]

\[ Q(R_1) = \emptyset, \quad Q(R_2) = \{(3, 2)\}, \] and hence \( T' \) cannot exist, because \( \emptyset \in \text{POSS}(T') \) if and only if \( T' = \emptyset \).
Weak representation systems

- Idea: consider certain answers:

\[
\text{certain}(Q, T_1, \ldots, T_n) = \bigcap \left\{ Q(R_1, \ldots, R_n) \left| \begin{array}{c}
R_1 \in \text{POSS}(T_1), \\
\vdots \\
R_n \in \text{POSS}(T_n)
\end{array} \right. \right\}
\]

- \(\text{certain}(T)\) – the set of tuples in \(T\) without null values.

- For a query language \(L\), Codd tables form a weak representation system if for any query \(Q\) in \(L\),

\[
\text{certain}\left(Q(T_1, \ldots, T_n)\right) = \text{certain}(Q, T_1, \ldots, T_n)
\]
Weak representation systems cont’d

• **Good news:** Codd tables form a weak representation system for the selection-projection queries in relational algebra.

• That is, Codd tables form a weak representation system for SQL queries of the form `SELECT-FROM-WHERE` such that the `FROM` clause only has one relation.

• **Bad News:** If we add either union or join (that is, allow `UNION` or multiple relations in the `FROM` clause), then Codd tables no longer form a weak representation system.

• **Reason:** we cannot use conditions of the form $x = y$, where $x$ and $y$ are variables, and this causes problems in computing joins.

• **Conclusion:** SQL’s nulls semantics is very very problematic.
**Naive tables**

- Codd tables in which some of the variables can coincide. One often refers to *marked nulls*.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td></td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₂</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>b₃</td>
<td>c₃</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>b₄</td>
<td>c₄</td>
<td></td>
</tr>
<tr>
<td>a₅</td>
<td>x</td>
<td>c₅</td>
<td></td>
</tr>
</tbody>
</table>

- Naive tables form a weak representation system for SPJU queries (that is, $\pi$, $\sigma$, $\bowtie$, $\cup$).

- In SQL terms: no INTERSECT, EXCEPT, NOT IN, NOT EXISTS

- **Naive evaluation:**

  \[
  \begin{array}{ccc}
  A & B & C \\
  1 & x & \bowtie \\
  2 & y & 4 \\
  \end{array}
  \begin{array}{ccc}
  B & C \\
  x & 3 \\
  \end{array}
  =
  \begin{array}{ccc}
  A & B & C \\
  2 & y & 4 \\
  \end{array}
  \begin{array}{ccc}
  1 & x & 3 \\
  \end{array}
  \]

- Heavily used in data exchange.
Naive evaluation of conjunctive queries

- $Q$ is a conjunctive query
- $T_1, \ldots, T_n$ are tables
- Compute $Q(T_1, \ldots, T_n)$ naively.
- Remove all tuples containing nulls from the result.
- The result is $\text{certain}(Q, T_1, \ldots, T_n)$
Naive evaluation of conjunctive queries: example

\[
R = \begin{array}{cc}
A & B \\
1 & x \\
2 & y \\
\end{array} \quad S = \begin{array}{cc}
B & C \\
x & y \\
y & 4 \\
\end{array}
\]

\[
Q = \pi_{AC}(R \bowtie_B S)
\]

Naive evaluation:

\[
\begin{array}{ccc}
A & B & C \\
R \bowtie_B S = & 1 & x \\
& 2 & y \\
\end{array}
\]

\[
\begin{array}{cc}
A & C \\
\pi_{AC}(R \bowtie_B S) = & 1 & y \\
& 2 & 4 \\
\end{array}
\]

• Remove tuples with nulls
• Get (2,4) as the certain answer.
Conditional tables

- Naive tables do not form a weak representation system for full relational algebra
- Conditional tables do.
- Example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$x &gt; 1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td></td>
</tr>
<tr>
<td>$x$</td>
<td>$b_3$</td>
<td>$c_3$</td>
<td>$t = 0$</td>
</tr>
<tr>
<td>$y$</td>
<td>$b_4$</td>
<td>$c_4$</td>
<td>$t = 1$</td>
</tr>
<tr>
<td>$a_5$</td>
<td>$x$</td>
<td>$c_5$</td>
<td></td>
</tr>
</tbody>
</table>

$x \neq 5 \lor y = 1$

- Query evaluation is quite complicated.
Theory of incomplete information: summary

- Simple representation: Codd tables. But we cannot even evaluate simple selections over them.
- If we settle for less – just certain answers must be represented correctly – then $\sigma$ and $\pi$ can be evaluated over Codd tables, but not $\cup$, $-$, $\ni$.
- If we use naive tables (variables can coincide), then SPJU queries can be evaluated.
- If we use conditional tables, all relational algebra queries can be evaluated, but conditional tables are very hard to deal with.
- Tradeoff:
  
  Semantic correctness vs Complexity of queries
Incomplete information in SQL

• SQL approach: there is a single general purpose NULL for all cases of missing/inapplicable information

• Nulls occur as entries in tables; sometimes they are displayed as null, sometimes as ‘–’

• They immediately lead to comparison problems

• The union of
  
  \[
  \text{SELECT } * \text{ FROM } R \text{ WHERE } R.A=1 \quad \text{and} \\
  \text{SELECT } * \text{ FROM } R \text{ WHERE } R.A<>1
  \]

should be the same as

\[
\text{SELECT } * \text{ FROM } R.
\]

• But it is not.

• Because, if \( R.A \) is null, then neither \( R.A=1 \) nor \( R.A<>1 \) evaluates to \( true \).
Nulls cont’d

- R.A has three values: 1, null, and 2.

- SELECT * FROM R WHERE R.A=1 returns \( \frac{A}{1} \)

- SELECT * FROM R WHERE R.A<>1 returns \( \frac{A}{2} \)

- How to check = null? New comparison: IS NULL.

- SELECT * FROM R WHERE R.A IS NULL returns \( \frac{A}{\text{null}} \)

- SELECT * FROM R is the union of
  SELECT * FROM R WHERE R.A=1,
  SELECT * FROM R WHERE R.A<>1, and
  SELECT * FROM R WHERE R.A IS NULL.
Nulls and other operations

- What is 1+null? What is the truth value of '3 = null'? 
- Nulls cannot be used explicitly in operations and selections: WHERE R.A=NULL or SELECT 5-NULL are illegal. 
- For any arithmetic, string, etc. operation, if one argument is null, then the result is null. 
- For R.A={1,null}, S.B={2},

```
SELECT R.A + S.B
FROM R, S
```

returns {3, null}. 
- What are the values of R.A=S.B? When R.A=1, S.B=2, it is false. When R.A=null, S.B=2, it is unknown.
The logic of nulls

- How does unknown interact with Boolean connectives? What is NOT unknown? What is unknown OR true?

\[
\begin{array}{cccc}
 x & \text{NOT } x & \text{AND} & \text{true} & \text{false} & \text{unknown} \\
 \text{true} & \text{false} & \text{true} & \text{true} & \text{false} & \text{unknown} \\
 \text{false} & \text{true} & \text{true} & \text{false} & \text{false} & \text{false} \\
 \text{unknown} & \text{unknown} & \text{unknown} & \text{unknown} & \text{false} & \text{unknown} \\
 \end{array}
\]

- Problem with null values: people rarely think in three-valued logic!
Nulls and aggregation

- Be ready for big surprises!

```
SELECT * FROM R
A
--------
  1
  -
```

SELECT COUNT(*) FROM R
returns 2

SELECT COUNT(R.A) FROM R
returns 1
Nulls and aggregation

- One would expect nulls to propagate through arithmetic expressions
- \(\text{SELECT SUM(R.A) FROM R}\) is the sum
  \[a_1 + a_2 + \ldots + a_n\]
  of all values in column \(A\); if one is null, the result is null.
- But \(\text{SELECT SUM(R.A) FROM R}\) returns 1 if \(R.A=\{1,\text{null}\}\).
- Most common rule for aggregate functions:
  first, ignore all nulls,
  and then compute the value.
- The only exception: \(\text{COUNT(*)}\).
Nulls in subqueries: more surprises

- \( R1.A = \{1,2\} \quad R2.A = \{1,2,3,4\} \)

- `SELECT R2.A
  FROM R2
  WHERE R2.A NOT IN (SELECT R1.A
  FROM R1)`

- Result: \( \{3,4\} \)

- Now insert a null into \( R1 \): \( R1.A = \{1,2, \text{null}\} \)
  and run the same query.

- The result is \( \emptyset \)!
Nulls in subqueries cont’d

- Although this result is counterintuitive, it is correct.
- What is the value of 3 NOT IN (SELECT R1.A FROM R1)?
  \[
  3 \text{ NOT IN } \{1,2,\text{null}\} \\
  = \quad \text{NOT (3 IN } \{1,2,\text{null}\}) \\
  = \quad \text{NOT((3 = 1) OR (3=2) OR (3=null))} \\
  = \quad \text{NOT(false OR false OR unknown)} \\
  = \quad \text{NOT (unknown)} \\
  = \quad \text{unknown}
  \]
- Similarly, 4 NOT IN \{1,2,\text{null}\} evaluates to unknown, and 1 NOT IN \{1,2,\text{null}\}, 2 NOT IN \{1,2,\text{null}\} evaluate to false.
- Thus, the query returns \emptyset.
Nulls in subqueries cont’d

- The result of

```
SELECT R2.A
FROM R2
WHERE R2.A NOT IN (SELECT R1.A
                    FROM R1)
```

can be represented as a conditional table:

<table>
<thead>
<tr>
<th>A</th>
<th>condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$x = 0$</td>
</tr>
<tr>
<td>4</td>
<td>$x \neq 0$</td>
</tr>
<tr>
<td>3</td>
<td>$y = 0$</td>
</tr>
<tr>
<td>4</td>
<td>$y = 0$</td>
</tr>
</tbody>
</table>
Nulls could be dangerous!

- Imagine US national missile defense system, with the database of missile targeting major cities, and missiles launched to intercept those.
- Query: Is there a missile targeting US that is not being intercepted?

```sql
SELECT M.#, M.target
FROM Missiles M
WHERE M.target IN (SELECT Name
                     FROM USCities) AND
                 M.# NOT IN (SELECT I.Missile
                              FROM Intercept I
                              WHERE I.Status = 'active')
```

- Assume that a missile was launched to intercept, but its target wasn’t properly entered in the database.
Nulls could be dangerous!

- Missile Intercept

<table>
<thead>
<tr>
<th>#</th>
<th>Target</th>
<th>Intercept</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>A</td>
<td>I1</td>
</tr>
<tr>
<td>M2</td>
<td>B</td>
<td>I2</td>
</tr>
<tr>
<td>M3</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

- A, B, C are in USCities

- The query returns the empty set: M2 NOT IN {M1, null} and M3 NOT IN {M1, null} evaluate to unknown.

- although either M2 or M3 is not being intercepted!

- Highly unlikely? Probably (and hopefully). But never forget what caused the Mars Climate Orbiter to crash!
Complexity of nulls

- Several problems related to nulls.
- We shall look at two:
  - recognizing relations in POSS($T$)
  - query answering (i.e., computing certain answers)
Recognising tables in \( \text{POSS}(T) \)

**Input:** a table \( T \), relation \( R \)

**Output:**
\[
\begin{cases}
1 & \text{if } R \in \text{POSS}(T) \\
0 & \text{otherwise}
\end{cases}
\]

Complexity depends on what type of table \( T \) is:

- If \( T \) is a Codd table, there is a polynomial \( O(n^2 \sqrt{n}) \) algorithm
  - bipartite graph matching
- If \( T \) is a naive table, the problem is NP-complete
  - 3-colorability reduction
- (blackboard)
Computing certain answers

**INPUT:** a table $T$, a tuple $t$

**OUTPUT:**

\[ \begin{cases} 
1 & \text{if } t \in \text{certain}(Q, T) \\
0 & \text{otherwise} 
\end{cases} \]

- Complexity: coNP-complete, under CWA.
  - it is in coNP: just guess $R \in \text{POSS}(T)$ so that $t \notin Q(R)$
  - it is complete for coNP: 3-colourability

- Complexity: undecidable for relational algebra queries under OWA
  - the same as validity problem in logic – undecidable
  - but can be solved efficiently (polynomial time) for simpler classes of queries (e.g. conjunctive or $\sigma, \pi, \Join, \cup$-queries)