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:

<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>z</td>
<td>c₅</td>
<td></td>
</tr>
</tbody>
</table>

• Semantics of a Codd table $T$ is the set 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). Suppose $Q$ is a query, over $n$ tables, in language $L$.

• If for all $T_1, \ldots, T_n$ which are inputs to $Q$, there exists a table $T'$ such that

$$\text{POSS}(T') = \{Q(R_1, \ldots, R_n) \mid R_1 \in \text{POSS}(T_1), \ldots, R_n \in \text{POSS}(T_n)\}$$

then we call $T'$ the answer to $Q$, that is, $Q(T_1, \ldots, T_n)$.

• If we can always find $Q(T_1, \ldots, T_n)$, 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: \[ T = \begin{array}{cc}
A & B \\
0 & 1 \\
\times & 2 \\
\end{array} \]

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:

\[
\text{certain}(Q, T_1, \ldots, T_n) = \bigcap \left\{ Q(R_1, \ldots, R_n) \mid R_1 \in \text{POSS}(T_1), \ldots, R_n \in \text{POSS}(T_n) \right\}
\]

- 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 \(L\),

\[
\text{certain}(Q(T_1, \ldots, T_n)) = \text{certain}(Q, T_1, \ldots, T_n)
\]
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.
Naive tables

• Codd tables in which some of the variables can coincide:

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

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

• In SQL terms: no \texttt{INTERSECT}, \texttt{EXCEPT}, \texttt{NOT IN}, \texttt{NOT EXISTS}

  \[
  \begin{array}{c|c|c} 
  A & B & C \\
  \hline
  1 & x & \bowtie \\
  2 & y & x \\
  \end{array}
  \begin{array}{c|c|c} 
  B & C \\
  \hline
  x & 3 \\
  y & 4 \\
  \end{array}
  =
  \begin{array}{c|c|c} 
  A & B & C \\
  \hline
  1 & x & 3 \\
  2 & y & 4 \\
  \end{array}
  
• Heavily used in data exchange.
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 $\cup$, $-$, $\Join$.

• 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
  \[
  \text{SELECT} \ast \text{ FROM R WHERE R.A=1} \quad \text{and} \\
  \text{SELECT} \ast \text{ FROM R WHERE R.A<>1} \\
  \text{should be the same as} \\
  \text{SELECT} \ast \text{ FROM R.}
  \]
- 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},

\[
\text{SELECT R.A + S.B} \\
\text{FROM R, S}
\]

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**?

<table>
<thead>
<tr>
<th>x</th>
<th>NOT x</th>
<th>AND</th>
<th>true</th>
<th>false</th>
<th>unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td></td>
<td>true</td>
<td>false</td>
<td>unknown</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td></td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>unknown</td>
<td>unknown</td>
<td></td>
<td>unknown</td>
<td>unknown</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>true</th>
<th>false</th>
<th>unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>unknown</td>
</tr>
<tr>
<td>unknown</td>
<td>true</td>
<td>unknown</td>
<td>unknown</td>
</tr>
</tbody>
</table>

- 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

- \( R_1.A = \{1,2\} \quad R_2.A = \{1,2,3,4\} \)
- SELECT R2.A
  FROM R2
  WHERE R2.A NOT IN (SELECT R1.A
                      FROM R1)

- Result: \{3,4\}
- Now insert a null into \( R_1 \): \( R_1.A = \{1,2, \text{null}\} \)
  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 NOT IN \{1,2,null\}
  = NOT (3 IN \{1,2,null\})
  = NOT((3 = 1) OR (3=2) OR (3=null))
  = NOT(false OR false OR unknown)
  = NOT (unknown)
  = unknown

• Similarly, 4 NOT IN \{1,2,null\} evaluates to unknown, and 1 NOT IN \{1,2,null\}, 2 NOT IN \{1,2,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!

- \[\begin{array}{c|c|c|c}
\text{#} & \text{Target} & \text{I\#} & \text{Missile} & \text{Status} \\
\hline
M1 & A & I1 & M1 & \text{active} \\
M2 & B & I2 & \text{null} & \text{active} \\
M3 & C & & & \\
\end{array}\]

- \{A, B, C\} are in USCities

- The query returns the empty set:
  \[\text{M2 NOT IN \{M1, null\} and M3 NOT IN \{M1, null\}}\]
  evaluate to \textit{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 POSS\((T)\)

\[\text{INPUT: } \text{a table } T, \text{ relation } R\]

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

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:** 
- \( 1 \) if \( t \in \text{certain}(Q, T) \)
- \( 0 \) otherwise

- 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 (blackboard)

- Complexity: **undecidable** for relational algebra queries under OWA
  - The same as validity problem in logic – undecidable
  - But in coNP for simpler classes of queries (e.g. conjunctive or \( \sigma, \pi, \bowtie, \cup \)-queries)