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:

T

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

- 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:

$$\hat{Q}(T) \quad = \quad \{Q(R) \mid R \in \mathsf{POSS}(T)\}$$

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

$$\mathsf{POSS}(Q(T)) \quad = \quad \{Q(R) \mid R \in \mathsf{POSS}(T)\}$$

- 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

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

  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: $T =$

A	B
0	1
x	2

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

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

$R_1 =$

A	B
0	1
2	2

and $R_2 =$

A	B
0	1
3	2

and

$Q(R_1) = \emptyset$, $Q(R_2) = \{(3,2)\}$, and hence T' cannot exist, because

$\emptyset \in$ POSS(T') if and only if $T' = \emptyset$

Weak representation systems

- Idea: consider certain answers:

$$\text{certain}(Q, T_1, \ldots, T_n) \;=\; \bigcap \left\{ Q(R_1, \ldots, R_n) \;\middle|\; \begin{array}{c} R_1 \in \text{POSS}(T_1), \\ \ldots, \\ R_n \in \text{POSS}(T_n) \end{array} \right\}$$

- 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,

$$\text{certain}\big(Q(T_1, \ldots, T_n)\big) \;=\; \text{certain}(Q, T_1, \ldots, T_n)$$

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:

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	x	c_5

- 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`
- Evaluation: $\begin{array}{cc} \mathsf{A} & \mathsf{B} \\ \hline 1 & \mathsf{x} \\ 2 & \mathsf{y} \end{array} \bowtie \begin{array}{cc} \mathsf{B} & \mathsf{C} \\ \hline \mathsf{x} & 3 \\ \mathsf{y} & 4 \end{array} = \begin{array}{ccc} \mathsf{A} & \mathsf{B} & \mathsf{C} \\ \hline 1 & \mathsf{x} & 3 \\ 2 & \mathsf{y} & 4 \end{array}$
- Heavily used in data exchange.

Conditional tables

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

A	B	C	condition
a_1	b_1	c_1	$x > 1$
a_2	b_2	c_2	
x	b_3	c_3	$t = 0$
y	b_4	c_4	$t = 1$
a_5	x	c_5	

$x \neq 5 \ \vee \ y = 1$

- 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 σ and π can be evaluated over Codd tables, but not $\cup, -, \bowtie$.
- 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
  `SELECT * FROM R WHERE R.A=1` and
  `SELECT * FROM R WHERE R.A<>1` should be the same as
  `SELECT * 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 $\frac{\text{A}}{1}$
- `SELECT * FROM R WHERE R.A<>1` returns $\frac{\text{A}}{2}$
- How to check $=$ null? New comparison: `IS NULL`.
- `SELECT * FROM R WHERE R.A IS NULL` returns $\frac{\text{A}}{\text{null}}$
- `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*?

-
`x`	`NOT x`
true	false
false	true
unknown	unknown

-
`AND`	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

-
`OR`	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

- 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
- `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 `SELECT SUM(R.A) FROM R` returns 1 if `R.A`=$\{$1,null$\}$.
- Most common rule for aggregate functions:

  first, ignore all nulls,

  and then compute the value.
- The only exception: `COUNT(*)`.

Nulls in subqueries: more surprises

- `R1.A` $= \{1,2\}$ `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` $= \{1,2,$ 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:

A	condition
3	$x = 0$
4	$x \neq 0$
3	$y = 0$
4	$y = 0$

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?

```
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

#	Target
M1	A
M2	B
M3	C

Intercept

I#	Missile	Status
I1	M1	active
I2	null	active

- {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 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 mathcing
- If T is a naive table, the problem is NP-complete
  - 3-colorability reduction
- (blackboard)

Computing certain answers

$$\begin{array}{ll} \textsc{Input:} & \text{a table } T, \ \text{a tuple } t \\ \textsc{Output:} & \begin{cases} 1 & \text{if } t \in \text{certain}(Q, T) \\ 0 & \text{otherwise} \end{cases} \end{array}$$

- 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)