SQL: Duplicate Semantics and NULL Values

Fall 2015

David Toman

School of Computer Science
University of Waterloo

Databases CS338

Multisets and Duplicates

- SQL uses a **MULTISET**/**BAG** semantics rather than a **SET** semantics:
  ⇒ SQL tables are **multisets** of tuples
  ⇒ originally for efficiency reasons
- What does "*allows duplicates*" mean?

part
bolt
bolt
bolt
nut
nut

$\Longleftrightarrow$

part	cnt
bolt	3
nut	2

How does this impact Queries?

Example (Cheap Quantification–Projection)

EMP	
Name	Dept
Bob	CS
Sue	CS
Fred	PMath
Barb	Stats
Jim	Stats

$\xrightarrow{\{y \mid \exists x.\mathrm{EMP}(x,y)\}}$

Dept
CS
CS
PMath
Stats
Stats

$\Longleftrightarrow$

Dept	cnt
CS	2
PMath	1
Stats	2

Duplicates and Queries

So how do we define what an answer to a query is now?

Ideas

1. an answer tuple can appear k times ($k > 0$) in $Q(D)$
2. the number of duplicates is a function of the numbers of duplicates in subqueries

Definition (Duplicate Semantics (for Relational Calculus))

we write $\mathbf{t}\{k\} \in Q$ for "the tuple $\mathbf{t}$ appears k times in the answer to Q"

$\mathbf{t}\{k\} \in R(\mathbf{x})$	if	$\mathbf{t} \in \mathbf{R}^D$ k times
$\mathbf{t}\{1\} \in (x_i = x_j)$	if	$\mathbf{t}(x_i) = \mathbf{t}(x_j)$
$\mathbf{t}\{m \cdot n\} \in Q_1 \wedge Q_2$	if	$\mathbf{t}\{m\} \in Q_1$ and $\mathbf{t}\{n\} \in Q_2$
$\mathbf{t}\{\sum_{v \in D} n_v\} \in \exists x_i.Q$	if	$\mathbf{t}[x_i \mapsto v]\{n_v\} \in Q$
$\mathbf{t}\{m + n\} \in Q_1 \vee Q_2$	if	$\mathbf{t}\{m\} \in Q_1$ and $\mathbf{t}\{n\} \in Q_2$
$\mathbf{t}\{\max(0, m - n)\} \in Q_1 \wedge \neg Q_2$	if	$\mathbf{t}\{m\} \in Q_1$ and $\mathbf{t}\{n\} \in Q_2$
$\mathbf{t}\{1\} \in \mathrm{DISTINCT}(Q)(D)$	if	$\mathbf{t}\{n\} \in Q$

Duplicates and SQL

Allowing duplicates leads to additional *syntax*.

- a *duplicate elimination operator*
  ⇒ "`SELECT DISTINCT x`" v.s. "`SELECT x`" in `SELECT`-blocks
- MULTISET (BAG) operators
  ⇒ equivalents of *set operations*
  ⇒ but with multiset semantics.

Example

```
SQL> select r1.publication
  2  from wrote r1, wrote r2
  3  where r1.publication=r2.publication
  4    and r1.author<>r2.author;

PUBLICAT
--------
ChSa98
ChSa98
ChTo98
ChTo98
ChTo98a
ChTo98a
```

⇒ for publications with n authors we get $O(n^2)$ answers!

Bag Operations

- bag union: `UNION ALL`
  ⇒ additive union: bag containing all in Q_1 *and* Q_2.
- bag difference: `EXCEPT ALL`
  ⇒ subtractive difference (monus):
  ⇒ a bag all tuples in Q_1 for which
  there is no "matching" tuple in Q_2.
- bag intersection: `INTERSECT ALL`
  ⇒ a bag of all tuples taking the
  maximal number common to Q_1 and Q_2

Example

```
SQL> ( select author
  2    from wrote, book
  3    where publication=pubid )
  4  union all
  5  ( select author
  6    from wrote, article
  7    where publication=pubid );

    AUTHOR
----------
         2
         3
         1
         2
         1
         2
         1
```

Summary

- SQL covered so far:
  1. Simple SELECT BLOCK
  2. Set operations
  3. Duplicates and Multiset operations
  4. Formulation of complex queries, nesting of queries, and views
  5. Aggregation
- this covers pretty much all of standard SQL queries (i.e., they can be expressed in the syntax introduced so far, but it might be quite cumbersome)

  $\Rightarrow$ (lots of) syntactic sugar coming next ...

What is a "null" value?

Phone		
Name	Office	Home
Joe	1234	3456
Sue	1235	?

- *Sue doesn't have home phone* (value inapplicable)
- *Sue has home phone, but we don't know her number* (value unknown)

Value Inapplicable

- Essentially *poor schema design*.
- Better design:

Office Phone	
Name	Office
Joe	1234
Sue	1235

Home Phone	
Name	Home
Joe	3456

- Queries should behave *as if asked* over the above decomposition.

  $\Rightarrow$ (relatively) easy to implement

Value Unknown

Idea

Unknown values can be replaced by any domain value (that satisfies integrity constraints).

$\Rightarrow$ *many possibilities (possible worlds)*

Phone		
Name	Office	Home
Joe	1234	3456
Sue	1235	?

$\rightarrow$

Phone		
Name	Office	Home
Joe	1234	3456
Sue	1235	0000

$\vdots$

Phone		
Name	Office	Home
Joe	1234	3456
Sue	1235	9999

Value Unknown and Queries

How do we answer queries?

Idea

Answers true in all possible worlds W of an incomplete D.

Certain Answer

$$Q(D) = \bigcap_{W \text{ world of } D} Q(W)$$

$\Rightarrow$ answer common to all possible worlds.

Is this (computationally) feasible?

$\Rightarrow$ NO (NP-hard to *undecidable* except in trivial cases)

SQL's solution: a (crude) approximation

What can we do with NULLs in SQL?

expressions

- general rule: a NULL as a parameter to an operation makes (should make) the result NULL
- $1 +$ NULL $\rightarrow$ NULL, 'foo' || NULL $\rightarrow$ NULL, etc.

predicates/comparisons

- three-valued logic (crude approximation of "value unknown")

set operations

- unique *special value* for *duplicates*

aggregate operations

- doesn't "*count*" (i.e., "value inapplicable")

Comparisons Revisited

Idea

Comparisons with a NULL *value return* UNKNOWN

Example

$1 = 1$	TRUE
$1 =$ NULL	UNKNOWN
$1 = 2$	FALSE

Still short of *proper logical* behaviour:

$$x = 0 \vee x \neq 0$$

should be always true (no matter what x is, including NULL!), but...

UNKNOWN and Boolean Connectives

Idea

Boolean operations have to handle UNKNOWN

$\Rightarrow$ *extended truth tables for boolean connectives*

$\wedge$	T	U	F
T	T	U	F
U	U	U	U
F	F	U	F

$\vee$	T	U	F
T	T	T	T
U	T	U	U
F	T	U	F

$\neg$	
T	F
U	U
F	T

...for tuples in which x is assigned the NULL value we get:

$$x = 0 \vee x \neq 0 \rightarrow \text{UNKNOWN} \vee \text{UNKNOWN} \rightarrow \text{UNKNOWN}$$

which is not the same as TRUE.

UNKNOWN in WHERE Clauses

How is this used in a WHERE clause?

- Additional syntax IS TRUE, IS FALSE, and IS UNKNOWN
  ⇒ WHERE <cond> shorthand for WHERE <cond> IS TRUE
- Special comparison IS NULL

List all authors for which we don't know a URL of their home page:

```
SQL> select aid, name
  2  from author
  3  where url IS NULL

       AID NAME
---------- -----------
         3 Saake, Gunter
```


Counting NULLS

How do NULLs interact with counting (and aggregates in general)?

- count(URL) counts only non-NULL URL's
  ⇒ count(*) counts "rows"

```
db2 => select count(*) as RS, count(url) as US
db2 (cont.) => from author

RS          US
----------------
3           2

  1 record(s) selected.
```


Outer Join

Idea

allow "NULL-padded" answers that "fail to satisfy" a conjunct in a conjunction

- extension of syntax for the FROM clause
  ⇒ FROM R <j-type> JOIN S ON C
  ⇒ the <j-type> is one of FULL, LEFT, RIGHT, or INNER
- semantics (for $R(x,y)$, $S(y,z)$, and $C = (r.y = s.y)$).
  1. $\{(x,y,z) : R(x,y) \wedge S(y,z)\}$
  2. $\{(x,y,\text{NULL}) : R(x,y) \wedge \neg(\exists z.S(y,z))\}$ for LEFT and FULL
  3. $\{(\text{NULL},y,z) : S(y,z) \wedge \neg(\exists x.R(x,y))\}$ for RIGHT and FULL

  ⇒ syntactic sugar for UNION ALL

Example

```
db2 => select aid,publication
db2 (cont.) => from author left join wrote
db2 (cont.) =>                  on aid=author

AID         PUBLICATION
----------- -----------
          1 ChTo98
          1 ChTo98a
          1 Tom97
          2 ChTo98
          2 ChTo98a
          2 ChSa98
          3 ChSa98
          5 -
  8 record(s) selected.
```

Counting with OJ

For every author count the number of publications:

```
db2 => select aid, count(publication) as pubs
db2 (cont.) => from author left join wrote
db2 (cont.) =>                   on aid=author
db2 (cont.) => group by aid

AID         PUBS
----------- -----------
          1           3
          2           3
          3           1
          5           0

  4 record(s) selected.
```


Summary

- NULLs are necessary evil

  ⇒ used to account for (small) irregularities in data
  ⇒ should be used sparingly

- can be **always** avoided

  ⇒ however some of the solutions may be inefficient

- you can't escape NULLs in practice

  ⇒ *easy fix* for blunders in schema design
  ⇒ . . . also due to schema evolution, etc.