More “Basic” SQL: Set Operations & NULL Values

COM 3563: Database Implementation

Avraham Leff

Yeshiva University

avraham.leff@yu.edu

COM3563: Fall 2020

Today's Lecture: Overview

1. Miscellania
2. Set Operations
3. The Problem of NULL

Many figures taken from lectures created by Professor Zvi Kedem, NYU Courant (used with permission), and from lectures by Professor Greiner (used with permission)

Context

- We've started exploring SQL capabilities, beginning with the fundamental *SELECT FROM WHERE* incantation
  - Focused on the relational algebra semantics of the underlying "selection", "projection", and "Cartesian product" operators
- Today
  - More "basic SQL" material, especially SQL set operations
  - Semantics and implications of NULL
- But first, short, but important "miscellania"
  - The material isn't very deep . . . ("syntax", not "semantics")
  - These are "pointer slides": be aware of the material, read textbook & Internet as necessary

Miscellania

Ambiguous Attribute Names

- Different relations may use the same attribute name in their schema
  - That's perfectly legal ☺
- Q: but what if your query must refer to both relations in the same query and must distinguish between the identical attribute name?
- A: resolve syntactic ambiguity by qualifying the attribute name with the appropriate relation name

```sql
SELECT Fname, EMPLOYEE.Name, Address
  FROM EMPLOYEE, DEPARTMENT
  WHERE DEPARTMENT.Name = 'Research' AND DEPARTMENT.Dnumber
      = EMPLOYEE.Dnumber;
```

Aliasing and Renaming (I)

- The ambiguity issue can even arise with respect to relation names
- Example: *"For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor"*
  - The problem: the query refers to the same relation twice
- Solution: have the query declare alternative relation names
  - These are called aliases or tuple variables
  - Syntax: use the AS clause
  - (Note: the AS may be dropped in most SQL implementations, but don't do that! ☺)

```
SELECT E.Fname, E.Lname, S.Fname, S.Lname
  FROM EMPLOYEE AS E, EMPLOYEE AS S
  WHERE E.Super_ssn = S.Ssn;
```

Aliasing and Renaming (II)

- You can use this renaming mechanism in any query to specify tuple variables for every table in the WHERE clause
  - Even when there no ambiguity exists ☺
  - Recommended practice to make your queries more readable
- Example: *"Retrieve the name and address of all employees who work for the 'Research' department"*

```sql
SELECT E.Fname, E.LName, E.Address
  FROM EMPLOYEE AS E, DEPARTMENT AS D
  WHERE D.DName = 'Research' AND D.Dnumber = E.Dno;
```

More selection examples

```
SELECT *
FROM Product
WHERE prod_price IS NOT NULL;
```

Inclusive

```
SELECT *
FROM Product
WHERE prod_price BETWEEN 20 AND 40;
```

```
SELECT *
FROM Product
WHERE prod_price > 20 AND prod_manufacturer = 'GizmoWorks';
```

```
SELECT *
FROM Product
WHERE prod_manufacturer IN ('GizmoWorks', 'WidgetsRUs');
```

Selection with pattern matching

```
SELECT *
FROM Product
WHERE prod_name LIKE '%Gizmo%';
```

%	= Match any sequence of 0-or-more characters
_	= Match any single character
[abc]	= Match any one character listed
[a-c]	= Match any one character in range

Making results distinct

```
SELECT DISTINCT manufacturer
FROM Product;
```

Product

prod_name	prod_price	prod_manufacturer
Gizmo	$19.99	GizmoWorks
Powergizmo	$39.99	GizmoWorks
Widget	$19.99	WidgetsRUs
HyperWidget	$203.99	Hyper

prod_manufacturer
GizmoWorks
WidgetsRUs
Hyper

Ensures results are a set.

Computation in SELECT clauses

```
SELECT location, time, celsius * 1.8 + 32 AS fahrenheit
FROM SensorReading;
```

Use AS. Otherwise, get a default field name.

```
SELECT player_id, Floor(height) AS feet, (height – Floor(height)) * 12 AS inches
FROM Player;
```

```
SELECT student_id, CASE WHEN lastname < 'N' THEN 1 ELSE 2 END AS group
FROM Student;
```

```
SELECT lastname || ', ' || firstname AS name
FROM Student;
```

Syntax details

- Strings use single quotes, not double 'Houston'
- Equality test uses single =, not double x = 5
- Amount of whitespace doesn’t matter

Sorting

```
SELECT prod_name, prod_manufacturer
FROM Product
ORDER BY prod_price DESC, prod_manufacturer;
```

Product

prod_name	prod_price	prod_manufacturer
Gizmo	$19.99	GizmoWorks
Powergizmo	$39.99	GizmoWorks
Widget	$19.99	WidgetsRUs
HyperWidget	$203.99	Hyper

→

prod_name	prod_manufacturer
HyperWidget	Hyper
Powergizmo	GizmoWorks
Gizmo	GizmoWorks
Widget	WidgetsRUs

Sorting on computation results

```
SELECT item, price * quantity AS total
FROM Order
ORDER BY price * quantity;
```

```
SELECT item, price * quantity AS total
FROM Order
ORDER BY total;
```

Order

item	price	quantity
apple	$0.50	3
orange	$0.60	2
banana	$0.40	4
peach	$0.80	1

→

item	total
peach	$0.80
orange	$1.20
apple	$1.50
banana	$1.60

Multiset semantics vs. Sorting

Table rows are unordered, except when theyre ordered.

More accurately, unless you use ORDER BY:

- Cant assume anything about ordering.
- Ordering depends on implementation, which can vary.
- Query results dont necessarily maintain order of original table.

Subset of results

Product

prod_name	prod_price	prod_manufacturer
Gizmo	$19.99	GizmoWorks
Powergizmo	$39.99	GizmoWorks
Widget	$19.99	WidgetsRUs
HyperWidget	$203.99	Hyper

```
SELECT prod_name, prod_manufacturer
FROM Product
LIMIT 2;
```

Some SQLs:
SELECT TOP 2 ...

```
SELECT prod_name, prod_manufacturer
FROM Product
ORDER BY prod_price
LIMIT 2;
```

prod_name	prod_manufacturer
Gizmo	GizmoWorks
HyperWidget	Hyper

Arbitrary products

prod_name	prod_manufacturer
Gizmo	GizmoWorks
Widget	WidgetsRUs

Cheapest products

String Matching

- You can always do an exact match on string value

```
delete from instructor
where dept_name='Finance'
```

- SQL includes a string-matching operator for comparisons on character strings
- The LIKE operator uses patterns that are described using two special characters
  - Percent (%): matches any substring

    ```
    select name
    from instructor where name like '%dar%'
    ```

    - Can use the ESCAPE operator to define the "escape" character and thus allow comparison against the % character itself

    ```
    like '100\%' escape  '\'
    ```

  - Underscore (_): matches any character

String Comparisons: Some Examples

- Even though SQL in general is case insensitive, patterns are case sensitive!
- Pattern matching examples
  - 'Intro%' matches any string beginning with "Intro"
  - '%Comp%' matches any string containing "Comp" as a substring
  - '___' matches any string of exactly three characters
  - '___%' matches any string of at least three characters
- Many built-in String functions
  - Concatenation
  - Convert between "upper case" and "lower case"
  - Substring extraction
  - String length

Set Operations

Introduction

- You may have heard that SQL is based on the relational algebra which, in turn, is based on a formal set-based definition of relations and tuples ☺
- We've already discussed the "selection" and "projection" operators which were developed specifically for relational databases
- We've already mentioned that the semantics of "Cartesian product" are changed from classic set theory
  - The Cartesian product of a set of n-tuples with a set of m-tuples yields a set of "flattened" $(n + m)$-tuples
  - "Basic" set theory would require a set of 2-tuples, each containing an n-tuple and an m-tuple
- Unsurprisingly, SQL also includes the set union, set difference, and set intersection operators
  - Respectively: SQL UNION, EXCEPT (some dialects have MINUS), and INTERSECT
- However: SQL adds an additional constraint to these operators: union compatibility

Union Compatibility

- In order to be used in a UNION, the two relations must have the "same attribute characteristics"
  - The attributes and their domains must be compatible: they share the same number of columns and their corresponding columns share the same or compatible domains
- This property holds for the INTERSECT and EXCEPT operators as well

Examples:

- $R(id, name)$ and $S(id, name, grades)$ are not union compatible: R has 2 attributes, but S has 3 attributes
- $R(id, name)$ and $S(id, grades)$ are not union compatible: the domains of "name" and "grades" are different ("string" versus "numeric")
- $R(id, name)$ and $S(id, StudentName)$ are union compatible: R and S both have 2 attributes and their domains are also identical (only the column names differ)

Set Operations: Examples

Find courses that ran in Fall 2009 or in Spring 2010

```
(select course_id from section
where sem = 'Fall' and year = 2009)
union
(select course_id from section
where sem = 'Spring' and year = 2010)
```

Find courses that ran in Fall 2009 and in Spring 2010

```
(select course_id from section
where sem = 'Fall' and year = 2009)
intersect
(select course_id from section
where sem = 'Spring' and year = 2010)
```

Find courses that ran in Fall 2009 but not in Spring 2010

```
(select course_id from section
where sem = 'Fall' and year = 2009)
except
(select course_id from section
where sem = 'Spring' and year = 2010)
```

Tables as Sets in SQL (I)

- The "multi-set" issue is one of the most important differences between "commercial" SQL and the formal relational algebra
- This course eschews "DML formalism" as much as possible, and therefore focuses on SQL's way of doing things …
  - But you must be aware that SQL allows a relation to include multiple tuples that are identical in all their attribute values
- This implies that an SQL table is not a set of tuples
  - Instead: an SQL table is a multi-set (or "bag") of tuples
  - By default, SQL will not eliminate duplicate tuples
- Note: we can constrain some SQL relations to be sets because of a "key constraint" or because we use DISTINCT in a SELECT statement

Tables as Sets in SQL (I)

- Q: why doesn't SQL simply eliminate duplicates in result-sets?
- A_1: because sometimes the client does want to see duplicate tuples ☺
  - Example: you do a JOIN between a "users" and "tasks" tables, and want to get all the tasks associated with a given user
  - See discussion here
  - OTOH: sometimes the client is introducing a "join duplication" bug ☹
- A_2: because that can be a very expensive operation ☹
  - Consider implementation: e.g., "first sort, then eliminate duplicates"
- Hold this discussion in mind when we (subsequent lecture) discuss aggregate functions
  - Typically we don't want to eliminate duplicates

Set Operations & Issue of Duplicates

- UNION operation automatically eliminates duplicates from the result relation
  - To override this behavior: must specify UNION ALL
- INTERSECT operation automatically eliminates duplicates from the result relation
  - To override this behavior: must specify INTERSECT ALL
  - The number of duplicate tuples is then the minimum of the number of duplicates in the two relations
- EXCEPT operation automatically eliminates duplicates from the result relation
  - To override this behavior: must specify EXCEPT ALL
  - The number of duplicate tuples is then the number of duplicates in the first relation minus the number of duplicates in the second relation
    - So long as that difference is positive ☺

Tables are *multisets*

So, what do we know about multisets?

Two ways to think about multisets

Tuple
(1, a)
(1, a)
(1, b)
(2, c)
(2, c)
(2, c)
(1, d)
(1, d)

=

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	1
(2, c)	3
(1, d)	2

Multiset union

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

$\cup$

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Tuple	$\lambda(Z)$
(1, a)	7
(1, b)	1
(2, c)	5
(1, d)	2

$$\boldsymbol{\lambda(Z) = \lambda(X) + \lambda(Y)}$$

Multiset intersection

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

$\cap$

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Tuple	$\lambda(Z)$
(1, a)	2
(1, b)	0
(2, c)	2
(1, d)	0

$$\boldsymbol{\lambda}(\boldsymbol{Z}) = min\left(\boldsymbol{\lambda}(\boldsymbol{X}), \boldsymbol{\lambda}(\boldsymbol{Y})\right)$$

Multiset difference

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

$-$

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Tuple	$\lambda(Z)$
(1, a)	0
(1, b)	0
(2, c)	1
(1, d)	0

$$\boldsymbol{\lambda(Z) = \lambda(X) - \lambda(Y)}$$

SQL syntax

Sets:

```
SELECT a
FROM R
UNION
SELECT a
FROM S;
```

```
…
INTERSECT
…
```

```
…
EXCEPT
…
```

Multisets:

```
SELECT a
FROM R
UNION ALL
SELECT a
FROM S;
```

```
…
INTERSECT ALL
…
```

```
…
EXCEPT ALL
…
```

- Remainder of lecture is our "drill-down" into the topic of NULL
- Besides its intrinsic importance ...this topic is connected to our discussion of SQL set operations
  - As we shall see, the semantics of NULL imply that the semantics of "duplicates" is much more complicated than we'd like ☹

The Problem of NULL

> *The simple scientific fact is that an SQL table that contains a null isn't a relation; thus, relational theory doesn't apply, and all bets are off.*
>
> *Chris Date*

NULL in a nutshell

- Each domain is augmented with a NULL
- NULL, intuitively stands for one of the following
  - Value unknown
  - Value not permitted to be known (to some of us)
  - Value not applicable

We've been using NULL in many of our examples: what is the fuss all about?

The Problem of NULL: What Semantics Is It Modeling?

- We just said that "NULL can have several interpretations"
  - Example: a query returns NULL as the value of the hair_color attribute
- Does that mean that the person is bald?
- Or: the person has hair, but you just don't know what color?
- Or: can the person be bald or have hair, but you just don't know which one applies?
- Or: the person in the midst of a hair coloring exercise and you only temporarily don't know the color?
- Or: (even more likely), the data-collection person simply forgot to record the data ☺

"Hair color" example taken from here

Dealing With The Consequences Of NULL

- SQL is forced to replace traditional boolean logic with a new 3-Valued logic
  - Once we introduce NULL (regardless of whether this is a good idea or not) . . .
  - We have no choice but to change very basic ideas of how logic behaves ☹
- Abbreviations
  - T for TRUE
  - F for FALSE
  - U for UNKNOWN
- Let's start with a boolean logic refresher ☺

	NOT
F	T
T	F

OR	F	T
F	F	T
T	T	T

AND	F	T
F	F	F
T	F	T

Three-Valued Logic: Intuition

- May help your intuition if you think of U as being “in between” F and T
  - But more accurate interpretation on next slide
- If you think of
  - NOT(x) as 1 - x
  - x OR y as max(x,y)
  - x AND y as min(x,y)
- For 2-valued logic we have
  - FALSE is 0
  - TRUE is 1
- For 3-valued logic we have
  - FALSE is 0
  - UNKNOWN is 0.5
  - TRUE is 1

Three-Valued Logic

> Instead of thinking of NULL as being "in between" F and T, better to say "NULL means: maybe T or maybe F"

	NOT
F	T
U	U
T	F

OR	F	U	T
F	F	U	T
U	U	U	T
T	T	T	T

AND	F	U	T
F	F	F	F
U	F	U	U
T	F	U	T

- Each tuple in the FROM clause is tested against the WHERE predicate
- Here are the rules
  - If P(tuple) is TRUE, it is passed to SELECT
  - If P(tuple) is UNKNOWN, it is not passed to SELECT
  - If P(tuple) is FALSE, it is not passed to SELECT
- Summary: For SELECT queries, UNKNOWN behaves exactly the same as FALSE
- Unfortunately this behavior is different for DDL and INSERT statements
  - Where UNKNOWN behaves as TRUE ☹

NULL & SELECT: AND Example

Any comparison in which one side is NULL is UNKNOWN

R	A	B	C
	1	6	8
	2	7	9
	3	NULL	8
	4	NULL	9

```
select A from R where B = 6 AND C =
    8
```

	A
	1

Any comparison in which one side is NULL is UNKNOWN

R	A	B	C
	1	6	8
	2	7	9
	3	NULL	8
	4	NULL	9

```
select A from R where B = 6 OR C = 8
```

	A
	1
	3

NULL & SELECT: NULL = NULL is FALSE!

Any comparison in which one side is NULL is UNKNOWN

R	A	B	C
	1	6	8
	2	7	9
	3	NULL	8
	4	NULL	9

```sql
select A from R where B = NULL;
```

The result of this query is: empty table!

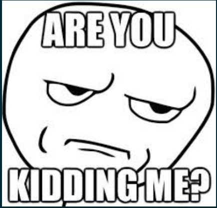
NULL & SELECT: NULL ≠ NULL is FALSE!

Any comparison in which one side is NULL is UNKNOWN

R	A	B	C
	1	6	8
	2	7	9
	3	NULL	8
	4	NULL	9

```
select A from R where B <> NULL;
```

The result of this query is: empty table!



Any comparison in which one side is NULL is UNKNOWN

R	A	B	C
	1	6	8
	2	7	9
	3	NULL	8
	4	NULL	9

```sql
select A from R where B = B;
```

	A
	1
	2

Because, going row by row:

1. 6 = 6 is TRUE
2. 7 = 7 is TRUE
3. NULL = NULL is UKNOWN
4. NULL = NULL is UNKNOWN

Alleviate Some of the Problem: Introduce New Keywords

- A new keyword made of three words: IS NOT NULL
- A new keyword made of two words: IS NULL

IS NOT NULL Example

R	A	B	C
	1	6	8
	2	7	9
	3	NULL	8
	4	NULL	9

```
select A from R where B is not null
```

	A
	1
	2

IS NULL Example

R	A	B	C
	1	6	8
	2	7	9
	3	NULL	8
	4	NULL	9

```
select A from R where B is null
```

	A
	3
	4

NULL & Arithmetic Wierdness

If one of the operands is NULL, the result is NULL!

- 5 + NULL = NULL
- NULL - NULL = NULL
- 0 * NULL = NULL
- NULL / 0 = NULL

> Key point: given the semantics of NULL, these results are inevitable

Quick review of SQL & duplicates

- Standard SELECT FROM WHERE statement does not remove duplicates at any stage of its execution
  - Use SELECT DISTINCT FROM WHERE to remove duplicates
- Standard UNION, EXCEPT, INTERSECT do remove duplicates
- UNION ALL, EXCEPT ALL, INTERSECT ALL do not remove duplicates with rather interesting semantics
- All NULLs are duplicates of one another
  - Implication: so if you use DISTINCT, you'll get at most NULL tuple
  - Even though it is UNKNOWN whether they are equal to each other ☺

Miscellania

Set Operations

The Problem of NULL

Readings

- "Miscellania" material mostly covered in Textbook, Chapter 3.4
- Set operations covered in Textbook, Chapter 3.5
- NULL covered in Textbook, Chapter 3.6