More SQL: NULL, Outer Joins

NULL

(null > 0) = null
(null + 1) = null
(null = 0) = null
(null AND true) = null
null is null = true

Some truth tables

AND	T	F	NULL
T	T	F	NULL
F	F	F	F
NULL	NULL	F	NULL

OR	T	F	NULL
T	T	T	T
F	T	F	NULL
NULL	T	NULL	NULL

NULL comparisons: unknown

Null is "unknown" or "maybe"
null > 16? Unknown!
left AND right: True if BOTH left and right are true;
- NULL AND true? Could be true if NULL was true: = NULL
- NULL AND false? Can only be false

left OR right: True if any one is true
- NULL OR true? Must be true, no matter what value
- NULL OR false? Could be true if NULL was true: = NULL

JOINS

```
SELECT [DISTINCT] target_list
FROM tableA, tableB
WHERE tableA.col = tableB.col AND ...

SELECT [DISTINCT] target_list
FROM tableA JOIN tableB
ON tableA.col = tableB.col
WHERE ...
```

(explicit) JOINS

```
SELECT [DISTINCT] target_list
FROM table_name
    [INNER {LEFT |RIGHT | FULL} {OUTER}] JOIN table_name
    ON qualification_list
WHERE ...
```

INNER is default
Difference is how to deal with NULL values

PostgreSQL documentation:
http://www.postgresql.org/docs/9.4/static/tutorial-join.html

Inner/Natural Join

```
SELECT s.sid, s.name, r.bid
FROM   Sailors S, Reserves r
WHERE  s.sid = r.sid

SELECT s.sid, s.name, r.bid
FROM   Sailors s INNER JOIN Reserves r
ON     s.sid = r.sid

SELECT s.sid, s.name, r.bid
FROM   Sailors s NATURAL JOIN Reserves r
```

All Equivalent!

Natural Join means equi-join for each pair of attrs with same name

Sailor names and their reserved boat ids

```
SELECT s.sid, s.name, r.bid
FROM   Sailors s INNER JOIN Reserves r
ON     s.sid = r.sid
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Result

sid	name	bid
1	Eugene	102
2	Luis	102

Sailor names and their reserved boat ids

```
SELECT s.sid, s.name, r.bid
FROM   Sailors s INNER JOIN Reserves r
ON     s.sid = r.sid
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Result

sid	name	bid
1	Eugene	102
2	Luis	102

Prefer INNER JOIN over NATURAL JOIN. Why?

Sailor names and their reserved boat ids

```
SELECT s.sid, s.name, r.bid
FROM   Sailors s INNER JOIN Reserves r
ON     s.sid = r.sid
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Result

sid	name	bid
1	Eugene	102
2	Luis	102

Notice: No result for Ken!

Left Outer Join (or No Results for Ken)

Returns all matched rows *and all unmatched rows from table on left of join clause*
(*at least one* row for each row in left table)

```
SELECT s.sid, s.name, r.bid
FROM   Sailors s LEFT OUTER JOIN Reserves r
ON     s.sid = r.sid
```

All sailors & bid for boat in their reservations
Bid set to NULL if no reservation

Left Outer Join

```
SELECT s.sid, s.name, r.bid
FROM   Sailors s LEFT OUTER JOIN Reserves r
ON     s.sid = r.sid
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Result

sid	name	bid
1	Eugene	102
2	Luis	102
3	Ken	NULL

Can Left Outer Join be expressed with Cross-Product?

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day

Sailors x Reserves

Result

sid	name	bid

```
Sailors s LEFT OUTER JOIN Reserves r
ON  s.sid = r.sid
```

Result

sid	name	bid
1	Eugene	NULL
2	Luis	NULL
3	Ken	NULL

Right Outer Join

Same as LEFT OUTER JOIN, but guarantees result for rows in table on right side of JOIN

```
SELECT  s.sid, s.name, r.bid
FROM  Sailors s LEFT OUTER JOIN Reserves r
ON    s.sid = r.sid

SELECT  s.sid, s.name, r.bid
FROM  Reserves r RIGHT OUTER JOIN Sailors S
ON    s.sid = r.sid
```

RIGHT OUTER JOIN

```
SELECT s.sid, s.name, r.bid
FROM   Sailors s RIGHT OUTER JOIN Reserves r
ON     s.sid = r.sid
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
4	109	9/20

Result

sid	name	bid
1	Eugene	102
2	Luis	102
NULL	NULL	109

Why is sid NULL?

FULL OUTER JOIN

Returns all matched *or* unmatched rows from both sides of JOIN

```
SELECT s.sid, s.name, r.bid
FROM   Sailors s FULL OUTER JOIN Reserves r
ON     s.sid = r.sid
```

FULL OUTER JOIN

```
SELECT s.sid, s.name, r.bid
FROM   Sailors s Full OUTER JOIN Reserves r
ON     s.sid = r.sid
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
4	109	9/20

Result

	sid	name	bid
	1	Eugene	102
	2	Luis	102
Left	3	Ken	NULL
Right	NULL	NULL	109

JOIN Advice

Prefer "FROM TableA, TableB WHERE …"
Except when you need OUTER JOIN (rare)

Integrity Constraints

Conditions that every legal instance must satisfy
- Inserts/Deletes/Updates that violate ICs rejected
- Helps ensure app semantics or prevent inconsistencies

We've discussed
- domain/type constraints, primary/foreign key
- general constraints ←

Beyond Keys: Table Constraints

Additional checks to ensure all data in table is valid

```
CREATE TABLE Sailors(
    sid int,
    …
    PRIMARY KEY (sid),
    CHECK (rating >= 1 AND rating <= 10)

CREATE TABLE Reserves(
    sid int,
    bid int,
    day date,
    PRIMARY KEY (bid, day),
    CONSTRAINT no_red_reservations
    CHECK ('red' NOT IN (SELECT B.color
                         FROM Boats B
                         WHERE B.bid = bid))
```

Nested subqueries
Named constraints

WHAT!

So many things we can't express or don't work!

Nested queries in CHECK constraints

Advanced Stuff

User defined functions
Triggers
WITH
Views

Advanced Stuff
aka Not On the Midterm

User defined functions
Triggers
WITH
Views

User Defined Functions (UDFs)

Custom functions that can be called in database
Many languages: SQL, python, C, perl, etc

```
CREATE FUNCTION function_name(p1 type, p2 type, …)
RETURNS type
```

User Defined Functions (UDFs)

Custom functions that can be called in database
Many languages: SQL, python, C, perl, etc

```
CREATE FUNCTION function_name(p1 type, p2 type, …)
RETURNS type
AS $$

-- logic

$$ LANGUAGE language_name;
```

User Defined Functions (UDFs)

Custom functions that can be called in database

Many languages: SQL, python, C, perl, etc

```
CREATE FUNCTION function_name(p1 type, p2 type, …)
RETURNS type
AS $$

-- logic

$$ LANGUAGE language_name;
```

A simple UDF (lang = SQL)

```
CREATE FUNCTION mult1(v int) RETURNS int
AS $$
SELECT v * 100;
$$ LANGUAGE SQL;
```

Last statement is returned

```
CREATE FUNCTION function_name(p1 type, p2 type, …)
RETURNS type
AS $$

-- logic

$$ LANGUAGE language_name;
```

A simple UDF (lang = SQL)

```
CREATE FUNCTION mult1(v int) RETURNS int
AS $$
SELECT v * 100;
$$ LANGUAGE SQL;

SELECT mult1(S.age)
FROM sailors AS S
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
2200
3900
2700

http://www.postgresql.org/docs/9.1/static/xfunc-sql.html

A simple UDF (lang = SQL)

```
CREATE FUNCTION mult1(int) RETURNS int
AS $$
SELECT $1 * 100;
$$ LANGUAGE SQL;

SELECT mult1(S.age)
FROM sailors AS S
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
2200
3900
2700

http://www.postgresql.org/docs/9.1/static/xfunc-sql.html

Process a Record (lang = SQL)

```
CREATE FUNCTION mult2(x sailors) RETURNS int
AS $$
SELECT (x.sid + x.age) / x.rating;
$$ LANGUAGE SQL;

SELECT mult2(S.*)
FROM sailors AS S
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
3.285
20.5
3.75

http://www.postgresql.org/docs/9.1/static/xfunc-sql.html

Process a Record (lang = SQL)

```
CREATE FUNCTION mult2(sailors) RETURNS int
AS $$
SELECT ($1.sid + $1.age) / $1.rating;
$$ LANGUAGE SQL;

SELECT mult2(S.*)
FROM sailors AS S
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
3.285
20.5
3.75

http://www.postgresql.org/docs/9.1/static/xfunc-sql.html

Procedural Language/SQL (lang = plsql)

```
CREATE FUNCTION proc(v int) RETURNS int
AS $$
DECLARE
    -- define variables
BEGIN
    -- PL/SQL code
END;
$$ LANGUAGE plpgsql;
```

Boilerplate

http://www.postgresql.org/docs/9.4/static/plpgsql.html

Procedural Language/SQL (lang = plsql)

```
CREATE FUNCTION proc(v int) RETURNS int
AS $$
DECLARE
    -- define variables.   VAR TYPE [= value]
    qty int = 10;
BEGIN
    qty = qty * v;
    INSERT INTO blah VALUES(qty);
    RETURN qty + 2;
END;
$$ LANGUAGE plpgsql;
```

http://www.postgresql.org/docs/9.4/static/plpgsql.html

Procedural Code (lang = plpython2u)

```
CREATE FUNCTION proc(v int) RETURNS int
AS $$
import random
return random.randint(0, 100) * v
$$ LANGUAGE plpython2u;
```

Very powerful – can do anything so must be careful

run in a python interpreter with no security protection

plpy module provides database access

plpy.execute("select 1")

http://www.postgresql.org/docs/9.4/static/plpython.html

Procedural Code (lang = plpython2u)

```
CREATE FUNCTION proc(word text) RETURNS text
AS $$
import requests
resp = requests.get('http://google.com/search?q=%s' % v)
return resp.content.decode('unicode-escape')
$$ LANGUAGE plpython2u;
```

Very powerful – can do anything so must be careful

run in a python interpreter with no security protection

plpy module provides database access

plpy.execute("select 1")

http://www.postgresql.org/docs/9.4/static/plpython.html