

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
```

All
Equivalent!

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

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

Don't use natural joins as two tables might have same ID or the database might change over time which breaks the code

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

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

<u>sid</u>	<u>bid</u>	<u>day</u>
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

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

<u>sid</u>	<u>bid</u>	<u>day</u>
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

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

<u>sid</u>	<u>bid</u>	<u>day</u>
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

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

<u>sid</u>	<u>bid</u>	<u>day</u>
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

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

<u>sid</u>	<u>bid</u>	<u>day</u>
------------	------------	------------

Sailors x Reserves

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

Result

sid	name	bid
-----	------	-----

Result

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

Can Left Outer Join be expressed with Cross-Product?

Sailors

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

<u>sid</u>	<u>bid</u>	<u>day</u>
------------	------------	------------

Sailors ⋈ Reserves

U

$(\text{Sailors} - (\text{Sailors} \bowtie \text{Reserves})) \times \{(\text{null}, \dots)\}$



How to compute this with a query?

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   Reserves r RIGHT OUTER JOIN Sailors S  
ON     s.sid = r.sid
```

FULL OUTER JOIN

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

Sailors

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

<u>sid</u>	<u>bid</u>	<u>day</u>
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?

As selecting s.sid

Would not be NULL if select r.sid

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

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

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

Result

Left
Right

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

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

Runs when table is not empty

```
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))
```

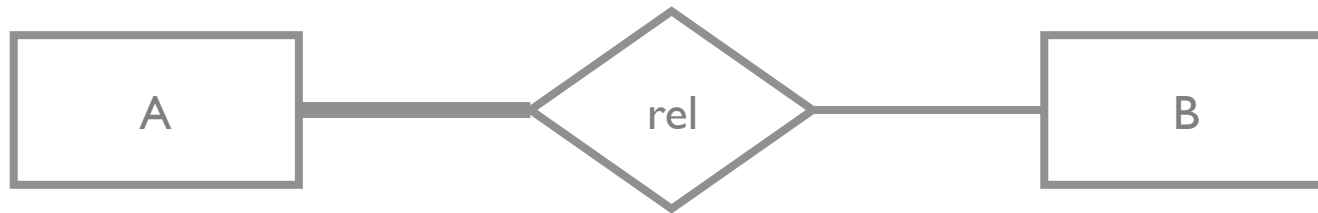
Nested subqueries
Named constraints

WHAT!

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

Assertions

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

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
220
390
270

A simple UDF (lang = SQL)

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

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

Sailors

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
220
390
270

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

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
3.285
20.5
3.75

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

<u>sid</u>	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
3.285
20.5
3.75

Procedural Language/SQL(lang = plsql)

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

Boilerplate



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;
```


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>

Triggers (logical)

def: procedure that runs automatically if specified changes in DBMS happen

```
CREATE TRIGGER name
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Triggers (logical)

def: procedure that runs automatically if specified changes in DBMS happen

```
CREATE TRIGGER name  
    [BEFORE | AFTER | INSTEAD OF] event_list  
    ON table
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Triggers (logical)

def: procedure that runs automatically if specified changes in DBMS happen

```
CREATE TRIGGER name  
    [BEFORE | AFTER | INSTEAD OF] event_list  
    ON table  
  
    WHEN trigger_qualifications
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Triggers (logical)

def: procedure that runs automatically if specified changes in DBMS happen

```
CREATE TRIGGER name  
    [BEFORE | AFTER | INSTEAD OF] event_list  
    ON table  
    [FOR EACH ROW]  
    WHEN trigger_qualifications  
    procedure
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Copy new young sailors into special table

(logical)

```
CREATE TRIGGER youngSailorUpdate
  AFTER INSERT ON SAILORS
  REFERENCING NEW TABLE NewInserts
  FOR EACH STATEMENT
  INSERT
    INTO YoungSailors(sid, name, age, rating)
    SELECT sid, name, age, rating
    FROM NewInserts N
    WHERE N.age <= 18
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Copy new young sailors into special table

(logical)

```
CREATE TRIGGER youngSailorUpdate
  AFTER INSERT ON SAILORS
  FOR EACH ROW
  WHEN NEW.age <= 18
  INSERT
    INTO YoungSailors (sid, name, age, rating)
    VALUES (NEW.sid, NEW.name, NEW.age, NEW.rating)
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Triggers (logical)

Can be complicated to reason about

Triggers may (e.g., insert) cause other triggers to run

If >1 trigger match an action, which is run first?

¬_(ツ)_/

```
CREATE TRIGGER recursiveTrigger
  AFTER INSERT ON SAILORS
FOR EACH ROW
  INSERT INTO Sailors(sid, name, age, rating)
    SELECT sid, name, age, rating
    FROM Sailors S
```

Triggers vs Constraints

Constraint

Statement about state of database

Upheld by the database for *any* modifications

Doesn't modify the database state

Triggers

Operational: X should happen when Y

Specific to statements

Very flexible

Triggers (postgres)

```
CREATE TRIGGER name
  [BEFORE | AFTER | INSTEAD OF] event_list
  ON table
  FOR EACH (ROW | STATEMENT)
  WHEN trigger_qualifications
  EXECUTE PROCEDURE user_defined_function();
```

PostgreSQL only runs *trigger* UDFs

<http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html>

<http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html>

Trigger Example

```
CREATE FUNCTION copyrecord() RETURNS trigger
AS $$
BEGIN
    INSERT INTO blah VALUES(NEW.a);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Signature: no args, return type is trigger

Returns NULL or same record structure as modified row

Special variables: OLD, NEW

```
CREATE TRIGGER t_copyinserts BEFORE INSERT ON a
FOR EACH ROW
EXECUTE PROCEDURE copyrecord();
```

<http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html>

<http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html>

Total boats and sailors < 100

```
CREATE FUNCTION checktotal() RETURNS trigger
AS $$
BEGIN
    IF ((SELECT COUNT(*) FROM sailors) +
        (SELECT COUNT(*) FROM boats) < 100) THEN
        RETURN NEW
    ELSE
        RETURN null;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER t_checktotal BEFORE INSERT ON sailors
FOR EACH ROW
EXECUTE PROCEDURE checktotal();
```

<http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html>

<http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html>

You can get into trouble...

```
CREATE FUNCTION addme_bad() RETURNS trigger
AS $$
BEGIN
    INSERT INTO a VALUES (NEW.*);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER t_addme_bad BEFORE INSERT ON a
FOR EACH ROW
EXECUTE PROCEDURE addme_bad();
```

<http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html>

<http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html>

You can get into trouble...

```
CREATE FUNCTION addme_stillwrong() RETURNS trigger
AS $$
BEGIN
    IF (SELECT COUNT(*) FROM a) < 100 THEN
        INSERT INTO a VALUES (NEW.a + 1);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER t_addme_stillwrong BEFORE INSERT ON a
FOR EACH ROW
EXECUTE PROCEDURE addme_stillwrong();
```

<http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html>

<http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html>

You can get into trouble...

```
CREATE FUNCTION addme_works() RETURNS trigger
AS $$
BEGIN
    IF (SELECT COUNT(*) FROM a) < 100 THEN
        INSERT INTO a VALUES (NEW.a + 1);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER t_addme_works AFTER INSERT ON a
FOR EACH ROW
EXECUTE PROCEDURE addme_works();
```

<http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html>

<http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html>

WITH

```
WITH RedBoats(bid, count) AS
    (SELECT    B.bid, count(*)
     FROM      Boats B, Reserves R
     WHERE     R.bid = B.bid AND B.color = 'red'
     GROUP BY  B.bid)
SELECT    name, count
FROM      Boats AS B, RedBoats AS RB
WHERE     B.bid = RB.bid AND count < 2
```

Names of unpopular boats

WITH

```
WITH RedBoats(bid, count) AS
    (SELECT    B.bid, count(*)
     FROM      Boats B, Reserves R
     WHERE     R.bid = B.bid AND B.color = 'red'
     GROUP BY  B.bid)
SELECT    name, count
FROM      Boats AS B, RedBoats AS RB
WHERE     B.bid = RB.bid AND count < 2
```

```
WITH tablename(attr1, ...) AS (select_query)
    [,tablename(attr1, ...) AS (select_query)]
main_select_query
```

Views

```
CREATE VIEW view_name  
AS select_statement
```

“tables” defined as query results rather than inserted base data

Makes development simpler

Used for security

Not *materialized*

References to *view_name* replaced with *select_statement*

Similar to WITH, lasts longer than one query

Names of popular boats

```
CREATE VIEW boat_counts
AS SELECT      bid, count(*)
   FROM        Reserves R
   GROUP BY    bid
   HAVING      count(*) > 10
```

Used like a normal table

```
SELECT bname
FROM   boat_counts bc, Boats B
WHERE  bc.bid = B.bid
```

```
SELECT bname
FROM
  (SELECT bid, count(*)
   FROM Reserves R
   GROUP BY bid
   HAVING count(*) > 10) bc,
  Boats B
WHERE  bc.bid = B.bid
```

Names of popular boats

Rewritten expanded query

CREATE TABLE

```
CREATE TABLE <table_name> AS  
<SELECT STATEMENT>
```

Guess the schema:

```
CREATE TABLE used_boats1 AS  
  SELECT r.bid  
  FROM   Sailors s,  
         Reservations r  
  WHERE  s.sid = r.sid  
  
used_boats1(bid int)
```

```
CREATE TABLE used_boats2 AS  
  SELECT r.bid as foo  
  FROM   Sailors s,  
         Reservations r  
  WHERE  s.sid = r.sid  
  
used_boats2(foo int)
```

How is this different than views?

What if we insert a new record into Reservations?

Summary

SQL is pretty complex

Superset of Relational Algebra SQL99 turing complete!

Human readable

More than one way to skin a horse

Many alternatives to write a query

Optimizer (theoretically) finds most efficient plan

