

# Null and Void?

Dealing with Nulls in DB2

**Craig S. Mullins**

Corporate Technologist  
NEON Enterprise Software, Inc.

<http://www.neonesoft.com>

<http://www.CraigSMullins.com>



# Agenda

---

Definition

Some History

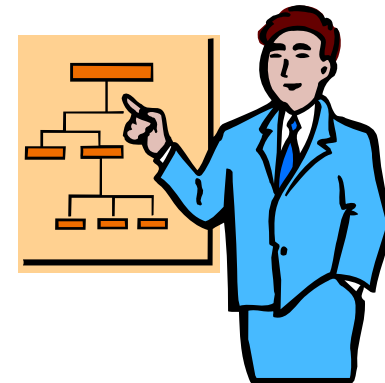
Types of Nulls

- Inapplicable versus Applicable Data
- Nulls and Keys
- Distinguished Nulls

Using Nulls in DB2

Problems with Nulls

Guidance and Advice



# What is a NULL?

---

NULL represents the absence of a value.

It is **not** the same as zero or an empty string.

A null is not a “null value”

- Because there is no value
- Maybe a “Null Lack of Value”



# What is the Difference, You Ask?

---

Consider the following columns:



- TERMINATION\_DATE - null or a valid date?
- SALARY - null or zero?
- SSN - non-US resident?
- ADDRESS - different composition by country, so let some components be null?
- HAIR\_COLOR - what about bald men?

# Ted Codd on Nulls

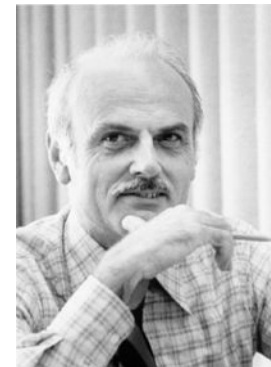
---

Dr. E.F. Codd built the notion of nulls into the relational model, but not in the original 1970 paper

- The extensions to RM/V1 to support nulls weren't defined until the 1979 paper *Extending the Database Relational Model to Capture More Meaning*, *ACM TODS 4*, No. 4 (December 1979).

He later defined two types of nulls in RM/V2

- These were called marks
  - A-mark: applicable, but unknown
  - I-mark: inapplicable (known to be inapplicable)



# Inapplicable versus Applicable

---



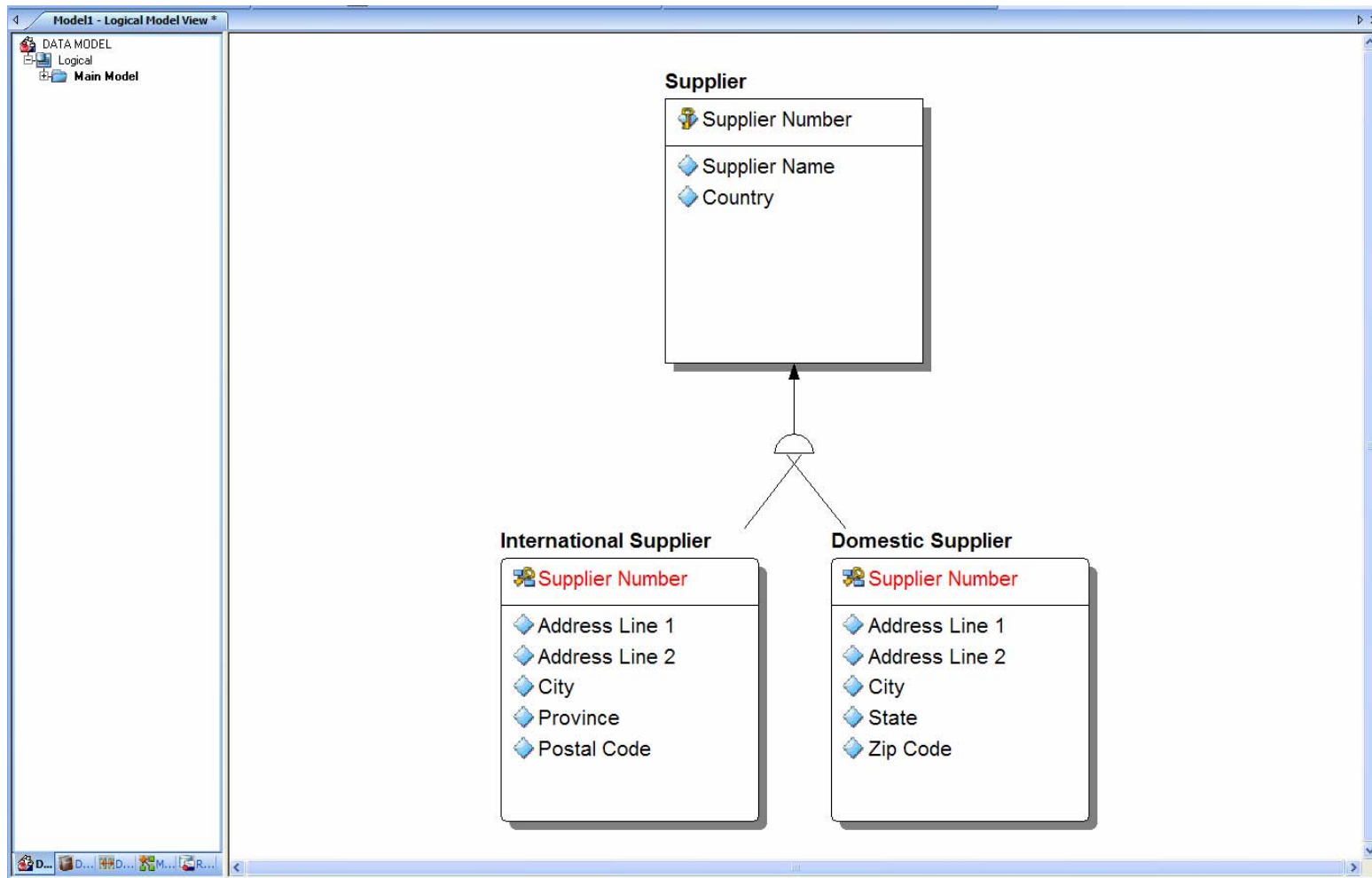
## Inapplicable:

- These are perhaps better dealt with by creating better data models
  - Social Security Number for a European
  - Consulting fee for a full-time employee

## Applicable, but unknown:

- These are the ones that are “true” nulls
  - Future dates not yet known
  - Information not supplied
  - Not everyone has a middle name

# Modeling *Fixes* Inapplicable Columns



TAKE CONTROL

# Nulls and Keys

---

## Primary Key

- A primary key cannot be null



## Foreign Key(s)

- Foreign keys *can* be null
- ON DELETE SET NULL - when the PK that refers to the FK row is deleted, the FK is set to null

# More on Foreign Keys

Which are valid foreign key rows?

TB1	PK1	PK2	cols...
	1	1	
	2	2	
	3	3	
	...	...	



TB2	PK	FK1	FK2	cols...
	A	1	1	
	B	2	NULL	
	C	5	NULL	
	D	NULL	NULL	

All of Them!

TAKE CONTROL

# How to Get Around That?

---

Consider adding a CHECK constraint:

- Either both are NULL or neither are NULL

CHECK (FK1 IS NULL AND FK2 IS NULL)  
OR (FK1 IS NOT NULL AND FK2 IS NOT NULL)



# More on NULLs and Constraints

NULLs are treated differently by CHECK constraints than VIEWS using WITH CHECK OPTION:

```
CREATE TABLE T1 (COL1 INTEGER);
```

```
CREATE VIEW V1 (COL1) AS  
  SELECT COL1 FROM T1 WHERE COL1 > 0  
WITH CHECK OPTION;
```

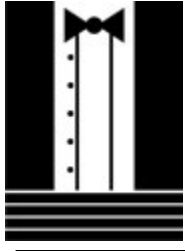
```
INSERT INTO T1 (COL1)  
VALUES (NULL);
```

**Failure**

```
CREATE TABLE T2  
(COL2 INTEGER CHECK (COL2 > 0));
```

```
INSERT INTO T2 (COL2)  
VALUES (NULL);
```

**Success**



# Distinguished Nulls:

*Not all Nulls are Equal*



Sometimes, based on domains and knowledge of the data a null may not really be a null.

Consider the **COLOR** column

- With a domain of BLACK, WHITE, and RED
- And a UNIQUE constraint on it

Now, say we have three rows in the table, one of which has “BLACK” for the COLOR and the other two are null.

- Are those nulls really completely null? We know *something* about them.

What if one of the nulls is changed to WHITE? That last null is not really null, is it?

- But DB2 does not automatically change it to RED...
- Can you (or your program)?

TAKE CONTROL

# A Lot of Arguments

---

Some relational experts have argued against nulls (e.g. Date, Darwen, Pascal)

There are many reasons; many of them centered around the “quirks” of how nulls are implemented and the difficulty of “three-valued” logic.

- Also, nulls violate “relational” - *at least according to some*
  - A null is not a value, and therefore cannot exist in a mathematical relation



# What About DB2?

---

Well, DB2 is not a “pure” relational DBMS

- It supports and uses nulls

Let’s take a look at how DB2 uses nulls and how you use DB2 to specify nulls and manipulate nulls in your data

- ...as well as some of the “oddities” this can create.

# Nulls in DB2

---



All columns are nullable unless defined with:

- NOT NULL or NOT NULL WITH DEFAULT

DB2 uses a null-indicator to set a column to null

- One byte
  - Zero or positive value means not null
  - Negative value means null

Not stored in column, but associated with column

### Nulls do **NOT** save space - *ever!*

- Nulls are not variable columns but...
- *Can be* used with variable columns to (*perhaps*) save space

Programs have to be coded to explicitly deal with the possibility of null

- (example on next slide)

# Using Null Indicator Variables

---



You can use null indicator variables with corresponding host variables in the following situations:

- SET clause of the UPDATE statement
- VALUES clause of the INSERT statement
- INTO clause of the SELECT or FETCH statement

# Using a Null Indicator in SQL

---



```
EXEC SQL
    SELECT  EMPNO , SALARY
    INTO    :EMPNO ,
           :SALARY :SALARY-IND
    FROM    EMP
    WHERE   EMPNO = '000100'
END-EXEC .
```

A common error is to specify the indicator, but then not check it and just use the column "value" ...

TAKE CONTROL

---

# Defining Null Indicators

## 01 EMP-INDICATORS .

```
10  WORKDEPT-IND    PIC S9 (4)  USAGE COMP .
10  PHONENO-IND     PIC S9 (4)  USAGE COMP .
10  HIREDATE-IND    PIC S9 (4)  USAGE COMP .
10  JOB-IND         PIC S9 (4)  USAGE COMP .
10  EDLEVEL-IND     PIC S9 (4)  USAGE COMP .
10  SEX-IND         PIC S9 (4)  USAGE COMP .
10  BIRTHDATE-IND  PIC S9 (4)  USAGE COMP .
10  SALARY-IND      PIC S9 (4)  USAGE COMP .
10  BONUS-IND       PIC S9 (4)  USAGE COMP .
10  COMM-IND        PIC S9 (4)  USAGE COMP .
```

# Defining a Null Indicator Structure



Null indicator structures can be coded instead of individual null indicators. A null indicator structure is defined as a null indicator variable with an OCCURS clause.

Null indicator structures enable host structures to be used when nullable columns are selected. The variable should occur once for each column in the corresponding host structure, as shown in the following section of code:

```
01  IDEPT PIC S9(4) USAGE COMP  
                                OCCURS 5 TIMES.
```

# Using a Null Indicator Structure

---



```
EXEC SQL
      SELECT  DEPTNO, DEPTNAME, MGRNO,
              ADMRDEPT, LOCATION
      FROM    DEPT
      INTO    :DCLDEPT:DEPT-IND
      WHERE   DEPTNO = 'A00'
      END-EXEC.
```

# And What If...

---

...You do not specify a null indicator or structure for a nullable column?

- SQLCODE -305
- SQLSTATE 22002



THE NULL VALUE  
CANNOT BE ASSIGNED TO  
OUTPUT HOST VARIABLE  
NUMBER *position-number*  
BECAUSE NO INDICATOR  
VARIABLE IS SPECIFIED

## Predicates not the same when checking for NULL

- Remember, a null is a lack of value, so the normal operators are not applicable
- That means no  $>$ ,  $<$ ,  $=$ ,  $<>$ , etc. when checking for null
- Instead use: IS [NOT] NULL

# Coding Proper Predicates

---

This is Correct

```
SELECT  EMPNO, WORKDEPT, SALARY
FROM    EMP
WHERE   SALARY IS NULL;
```

This is Incorrect

```
SELECT  EMPNO, WORKDEPT, SALARY
FROM    EMP
WHERE   SALARY = NULL;
```



## But, This is Correct, Too

---



In UPDATE statements you can set a (nullable) column to be null using =

For example, this statement sets the MGRNO to null for every department managed by manager number '000010':

```
UPDATE  DSN8810.DEPT
        SET  MGRNO = NULL
WHERE   MGRNO = '000010';
```

# IS NOT DISTINCT FROM

Version 8



## New predicate operator in DB2 Version 8.

- Two columns are not equal if both are NULL, but sometimes you want to know that:

```
WHERE COL1 = COL2  
OR COL1 IS NULL AND COL2 IS NULL
```

- As of V8, the following SQL is logically equivalent to that above, but perhaps simpler to code and understand:

```
WHERE COL1 IS NOT DISTINCT FROM COL2
```

- Also, IS NOT DISTINCT FROM is a Stage 1 predicate
  - But not indexable

TAKE CONTROL



# What About INSERT?

---



Here is an example:

```
DECLARE GLOBAL TEMPORARY TABLE ALERT_BUS_NM
(  ALERT_ID          INTEGER
  BUS_NM            CHAR(40) ) ;
```

```
INSERT INTO SESSION.ALERT_BUS_NM(ALERT_ID,BUS_NM)
VALUES(0001,'NEON ENTERPRISE SOFTWARE') ;
```

```
INSERT INTO SESSION.ALERT_BUS_NM(ALERT_ID,BUS_NM)
VALUES(NULL, NULL) ;
```

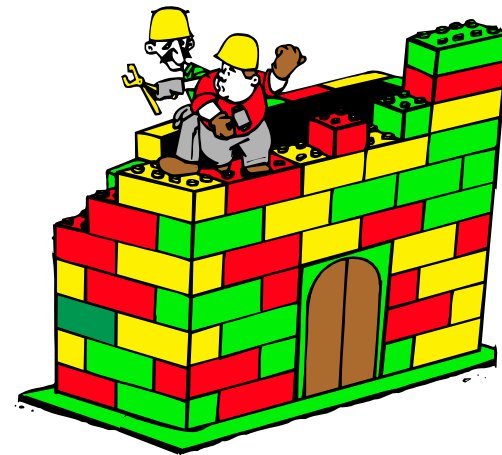
# Converting Nulls

---

Sometimes you will want to convert a null to a “normal” value to better be able to deal with it in the context of your application.

DB2 offers three functions to help you manage and/or convert nulls in your SQL:

- NULLIF
- COALESCE
- IFNULL



# NULLIF

---

The NULLIF function returns null if the two arguments are equal; otherwise, it returns the value of the first argument.

Assume that host variables PROFIT, EXPENSES, and INCOME have decimal data types with the values of 2500.00, 1500.00, and 4000.00 respectively.

The following function will then return a null:

```
NULLIF (:INCOME - :EXPENSES , :PROFIT)
```

# NULLIF Gotcha?

Consider the following function spec:

**NULLIF** (x1, x2)

But think about these situations:

- x1 <null>, x2 <372> (or any other value)
- x1 <null>, x2 <null>
- x1 <0>, x2 (a - a)

null

null

when a not null; then null  
when a is null; then 0

What is the result of the function for these?

# COALESCE

The COALESCE function returns the value of the first non-null expression.

- Here is an example from the sample database that comes with DB2.
- The HIREDATE column of DSN8810.EMP is nullable.
- The following query selects all employee rows for which HIREDATE is either unknown or earlier than Jan 1, 1960.

```
SELECT *  
FROM DSN8810.EMP  
WHERE COALESCE(HIREDATE,DATE('1959-12-31')) < '1960-01-01';
```

## COALESCE (continued)



COALESCE may contain more than two arguments.

For example:

```
SELECT COALESCE (C1, C2, "X")
```

In this case, if both C1 and C2 are null, then the constant value "X" is returned.

# Common Usage of COALESCE



To convert nulls in a SELECT list where there are columns that have to be added, but one can be a NULL

```
SELECT EMPNO, FIRSTNME, LASTNME,  
       (COALESCE (SALARY, 0) +  
        COALESCE (COMM, 0) +  
        COALESCE (BONUS, 0)) AS TOTAL_COMP  
FROM   DSN8810.EMP;
```

# IFNULL

IFNULL is almost identical to the COALESCE function...

- ...except that IFNULL is limited to two arguments instead of multiple arguments.

Do not confuse IFNULL with NULLIF; they are two different functions with different purposes.

- **IFNULL** returns the value of the first non-null expression
- **NULLIF** returns null if the two arguments are equal; otherwise, it returns the value of the first argument

## Inconsistent Treatment?

- Not equal in predicates
  - That is, WHERE NULL = NULL is not true, but unknown
- “Equal” in terms of sort order though
  - That is, all nulls group together for ORDER BY and GROUP BY
- Is this inconsistent?
  - Would you really want a separate row for each null in your output for GROUP BY and ORDER BY?

# GROUP BY and ORDER BY

---

When a nullable column participates in an ORDER BY or GROUP BY clause, the returned nulls are grouped together at the high end of the sort order.



# Grouping Nulls

---

Nulls group together (“equal” for sort order)

What if you want to GROUP BY an alternate column if the first is null?



Could use the COALESCE function (where x can be null and y cannot):

```
SELECT COALESCE(x, y), . . .  
FROM Table1  
GROUP BY 1
```

Of course, if y is a constant, then all of the nulls still group together...

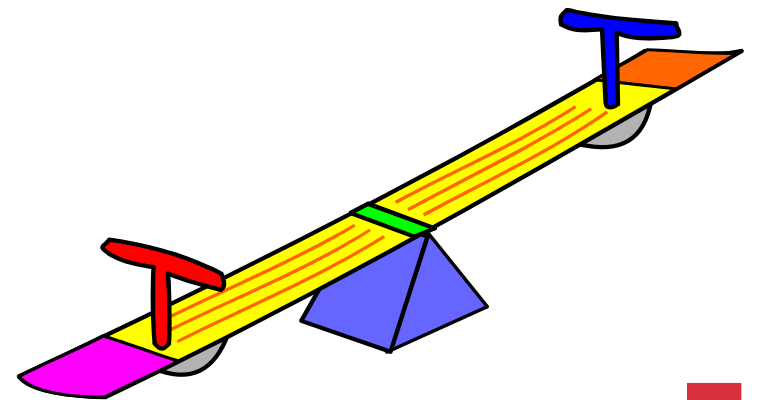
# When is a Null Equal to a Null?

---

In an ORDER BY or GROUP BY clause

When duplicates are eliminated by SELECT DISTINCT or COUNT(DISTINCT column)

In unique indexes without the WHERE NOT NULL specification



# The Logic of Nulls?

## Two- Versus Three-Valued Logic

P	Q	P AND Q	P OR Q
True	True	True	True
True	False	False	True
True	Unknown	Unknown	True
False	True	False	True
False	False	False	False
False	Unknown	False	Unknown
Unknown	True	Unknown	True
Unknown	False	False	Unknown
Unknown	Unknown	Unknown	Unknown

# Problems with Nulls

---

These “problem” areas will be covered in the ensuing slides:

- Arithmetic with Nulls
- Functions and Nulls
- EXISTS and Nulls
- IN and Nulls



TAKE CONTROL

---

# Arithmetic with Nulls

Any time a null participates in an arithmetic expression the results is null.

For example:

- $5 + \text{NULL}$
- $\text{NULL} / 501324$
- $102 - \text{NULL}$
- $51235 * \text{NULL}$
- $\text{NULL}^{**}3$
- $\text{NULL} + \text{NULL}$

The answer is NULL

# Here's a Strange One

---

What is the result of NULL/0?

At first glance a mathematician would say it should return an error because dividing any number by zero is invalid.



But, the result in DB2 will be NULL.

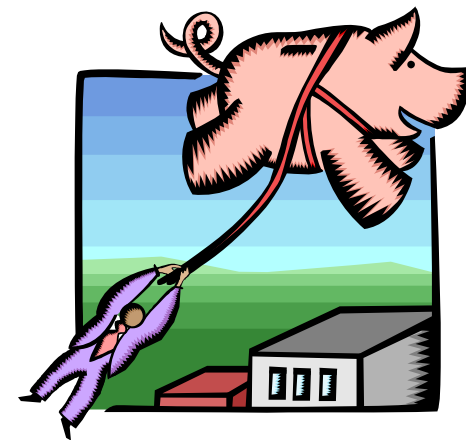
- As it would be in any SQL-compliant DBMS.

# Functions and Null

---

**AVG, COUNT DISTINCT, SUM, MAX and MIN omit column occurrences that are set to null.**

- So nulls are discounted by these functions
- This can cause some “interesting” anomalies



# SUM and Nulls

---

Functions can complicate things. Consider this SQL:

```
SELECT SUM(SALARY)
FROM EMP
WHERE DEPTNO > 999;
```

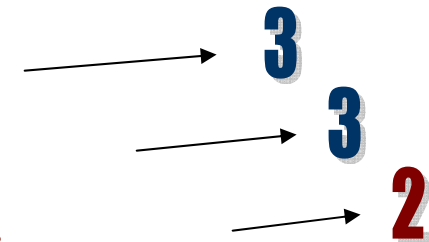
Even if SALARY is defined as NOT NULL  
this can return a NULL

- When no DEPTNO is greater than 999

# The Effect of Null on COUNT()

Consider the following three statements:

- `SELECT COUNT(*) FROM DSN8810.EMP;`
- `SELECT COUNT(EMPNO) FROM DSN8810.EMP;`
- `SELECT COUNT(HIREDATE) FROM DSN8810.EMP;`



Along with the following data:

<u>EMPNO</u>	<u>LASTNAME</u>	<u>HIREDATE</u>
'000010'	HAAS	1995-10-23
'000030'	KWAN	1980-04-03
'000555'	MULLINS	< null >

# AVG and Nulls

---

Sometimes what seems<sup>1</sup> to be the logical result is not so with nulls. Consider:

- $AVG(col)$
- $SUM(col)/COUNT(*)$

Although it would seem that these two functions should return the same result, they may not.

- AVG and SUM eliminate NULLs, but COUNT(\*) does not

# EXISTS

- Although a null is not a value, it does exist.
- So, unknowns do exist in an EXISTS statement.
- Consider, for example, an unknown (that is, NULL) celebrity birthday in the following SQL.
- The birthday is unknown, so it may be the same!

```
SELECT P1.emp_name, 'born on a day w/o a famous person'  
FROM Personnel AS P1  
WHERE NOT EXISTS  
    (SELECT *  
     FROM Celebrities AS C1  
     WHERE P1.birthday = C1.birthday);
```

# IN

Nulls can be confusing when using the IN predicate

Consider two tables:

- TableA contains: 20, 40, 60, 80
- TableB contains: 20, <null>, 40

What is the result of the following query?

```
SELECT *  
FROM TableA  
WHERE Column NOT IN  
      (SELECT Column FROM TableB);
```

Answer  
Empty set

# IN (continued)

OK, let's reverse that. Consider two tables:

- TableC contains: 20, 40, <null>, 80
- TableD contains: 20, 40, 60

What is the result of the similar query?

```
SELECT *  
FROM TableC  
WHERE Column NOT IN  
      (SELECT Column FROM TableD);
```

Answer  
80

# Outer Joins

---

## NULLs are generated when you issue outer joins

```
SELECT PART, SUPPLIER, A.PRODNO, PRODUCT  
FROM Parts A FULL OUTER JOIN Products P  
ON A.PRODNO = P.PRODNO;
```

- When there is a Part w/o a Product
    - PART, SUPPLIER, and PRODNO are NULL
  - When there is a Product w/o a Part
    - PRODUCT is NULL
- Unless you convert them using COALESCE

# Outer Joins *(continued)*



## NULLs are generated when you issue outer joins

- Unless you convert them using COALESCE

```
SELECT COALESCE(PART, "No Part"),  
       COALESCE(SUPPLIER, "No Supplier"),  
       COALESCE(A.PRODNO, P.PRODNO),  
       COALESCE(PRODUCT, "No Product")  
FROM Parts A FULL OUTER JOIN Products P  
ON A.PRODNO = P.PRODNO;
```

# Advice on Nulls

---



Define the meaning of each null for each column

Use data modeling to avoid inapplicable nulls

Try to avoid when arithmetic is involved

But do not bury your head in the sand

Study the theory

Know the reality

TAKE CONTROL

---

# Define Their Meaning

---

Define the meaning of each null, for each column

If a column is to be nullable, the application and database designers must define what it means when the column is null.

The best “meaning” for a null is usually applicable, but unknown, but you may have exceptions.

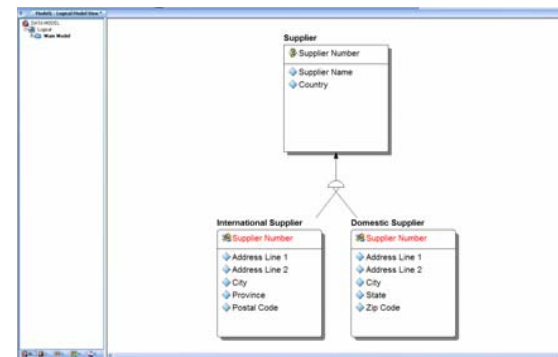
- If you have exceptions, be sure to document the meaning and train users appropriately.

# Use Data Modeling to Remove Inapplicable Nulls

Whenever possible try to create entities where every attribute is applicable

In doing so, then the only nulls will be applicable, but unknown

Recall the data modeling discussion from slide 6



# Try to Avoid When...

---

If possible, avoid nullable columns when they will be involved in arithmetic.

If nulls are involved in equations and functions then:

- You will need to use COALESCE to avoid null or realize that the result for every equation or math function involving a null will be null.
- Remember the situation with AVG and SUM/COUNT(\*)

# Do Not “Bury Your Head”

---

Learn what nulls are and how they work

Even if you define every column as not nullable  
your queries can still return a null

- Remember the slides  
on functions



# Study the Theory, Know the Reality

---



Understand the legitimate “problems” that experts like Chris Date have with nulls

Read the literature on the web and database books for guidance and enlightenment

- *Details on next two slides*

Understand the reality that existing SQL database systems like DB2 are created with nullability built into them

- As long as you use them, you need to understand how they use nulls to deal with missing information

# Literature on Nulls on the Web

---



Codd's 12 Rules

[http://www.itworld.com/nl/db\\_mgr/05072001/](http://www.itworld.com/nl/db_mgr/05072001/)

On Nulls and Empty Sets (Fabian Pascal, Chris Date, Hugh Darwen)

<http://www.dbdebunk.com/page/page/2296478.htm>

Old Approach, True Implementations by Fabian Pascal

<http://www.dbazine.com/ofinterest/oi-articles/pascal28>

The Final Null in the Coffin by Fabian Pascal

<http://www.dbdebunk.com/page/page/1396241.htm>

How To Handle Missing Information Without Using Nulls

<http://web.onetel.com/~hughdarwen/TheThirdManifesto/Missing-info-without-nulls.pdf>

There's Only One Relational Model by Chris Date

<http://www.dbdebunk.com/page/page/622839.htm>

An Interview with Chris Date

<http://www.oreillynet.com/pub/a/network/2005/07/29/cjdate.html?page=1>

Nulls - A Conversion Headache by John Hindmarsh

[http://www.embarcadero.com/resources/tech\\_papers/nullsconversion.html](http://www.embarcadero.com/resources/tech_papers/nullsconversion.html)

Using Nulls in DB2 by Craig S. Mullins

<http://www.craigsmullins.com/bp7.htm>

# Books for Additional Research



Codd, E.F., *The Relational Model for Database Management Version 2*, Reading, MA: Addison-Wesley (1990): ISBN 0201141922 (out of print)

Date, C.J., *An Introduction to Database Systems, 8<sup>th</sup> ed.*, Reading, MA: Addison-Wesley (2003): ISBN 0321197844

Date, C.J., *Database In-Depth*, Reading, MA: Addison-Wesley (2005): ISBN: 0596100124

Date, C.J., *The Database Relational Model*, Reading, MA: Addison-Wesley (2001): ISBN 0201612941

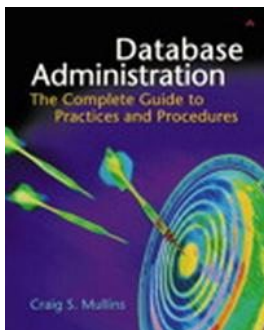
Celko, Joe, *SQL For Smarties: Advanced SQL Programming, 3<sup>rd</sup> ed.*, San Francisco, CA: Morgan Kaufmann (2005): ISBN 0123693799

Web links to amazon.com for these books available at:  
<http://www.craigsmullins.com/booklnks.htm>



**Craig S. Mullins**  
NEON Enterprise Software

[craig.mullins@neonesoft.com](mailto:craig.mullins@neonesoft.com)



[www.neonesoft.com](http://www.neonesoft.com)

[www.craigsmullins.com](http://www.craigsmullins.com)

[www.DB2portal.com](http://www.DB2portal.com)



Intelligent Solutions for Enterprise Data.