### How To Handle Missing Information Without Using NULL

#### Hugh Darwen

hugh@dcs.warwick.ac.uk www.dcs.warwick.ac.uk/~hugh

for Warwick University, CS319

SQL's NULL Is A Disaster	
Relational Database Writings 1985-1989 by C.J.Date with a special contribution by H.D. (as Andrew War	rden)
Relational Database Writings 1989-1991 by C.J.Date with Hugh Darwen	
Relational Database Writings 1991-1994 by C.J.Date	
Relational Database Writings 1994-1997 by C.J.Date	
Database Explorations by C.J. Date and Hugh Darwen (2010)	
	2

See

#### NULL

Cause of more debate and anguish than any other Fatal Flaw.

There's even a split in the relational camp (E.F. Codd proposed "A-marks", "I-marks" and a 4-valued logic).

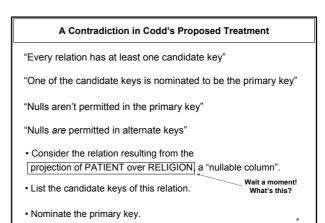
There's only one NULL. How many different reasons can there be for something being "missing"?

Why NULL ruins everything –

- UNION of sets, cardinality of sets.

Destruction of functional dependency theory

SQL's implementation of NULL is even worse than the best suggested by theoreticians. And it's not completely BYPASSABLE, because SQL thinks that the sum of the empty set is NULL! Nor is it CORRECTABLE - the Shackle of Compatibility! <sup>3</sup>



### Surprises Caused by SQL's NULL

- 1. SELECT \* FROM T WHERE X = Y UNION SELECT \* FROM T WHERE NOT ( X = Y ) is not equal to SELECT \* FROM T
- 2. SELECT SUM(X) + SUM(Y) FROM T is not equal to SELECT SUM(X + Y) FROM T
- IF X = Y THEN 'Yes'; ELSE 'No' is not equal to IF NOT ( X = Y ) THEN 'No'; ELSE 'Yes'

Why NULL Hurts Even More Than It Once Did

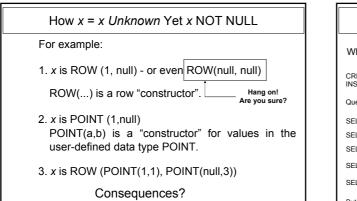
Suppose "*x* = *x*" returns *Unknown* 

Can we safely conclude "x IS NULL" ?

Suppose x "is not the null value"?

Can we conclude "x IS NOT NULL"?

Not in modern SQL!



x IS NULL (Case 1)						
What does x IS NULL MEAN? Think you know? Well, think again!						
CREATE TABLE T ( C1 INT, C2 ROW ( F1 INT, F2 INT ) ) ; INSERT INTO T VALUES ( NULL, NULL ) ;						
Query	Result Cardina	ality				
SELECT * FROM T WHERE C1 IS NULL	1					
SELECT * FROM T WHERE C2 IS NULL	1					
SELECT * FROM T WHERE ( C1, C1 ) IS NULL	1					
SELECT * FROM T WHERE ( C1, C2 ) IS NULL	1	So far,				
SELECT * FROM T WHERE ( C2, C2 ) IS NULL	1	so good?				
But even this depends on our charitable interpretation of the ISO SQL standard. $$\ensuremath{_{8}}$						

x IS NULL (Case 2)
TABLE T ( C1 INT, C2 ROW ( F1 INT, F2 INT ) ) ; INTO T VALUES ( NULL, ROW ( NULL, NULL ) ) ; - note the difference from Case 1
Result Cardinality

CREATE

Query

SELECT * FROM T WHERE C1 IS NULL	1		
SELECT * FROM T WHERE C2 IS NULL	1	!!!	
SELECT * FROM T WHERE ( C1, C1 ) IS NULL	1		
SELECT * FROM T WHERE ( C1, C2 ) IS NULL	0	!!!	
SELECT * FROM T WHERE ( C2, C2 ) IS NULL	0	!!!	

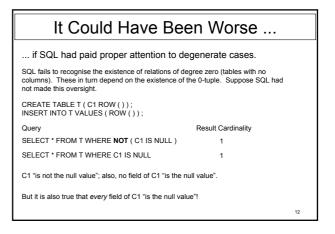
x IS NOT NULL					
So, what does x IS NOT NULL MEAN?					
CREATE TABLE T ( C1 INT, C2 ROW ( F1 INT, F2 INT ) ) ; INSERT INTO T VALUES ( NULL, ROW ( NULL, NULL ) ) ;					
Query	Result Cardi	nality			
SELECT * FROM T WHERE C1 IS NOT NULL	0				
SELECT * FROM T WHERE C2 IS NOT NULL	0				
SELECT * FROM T WHERE ( C1, C1 ) IS NOT NULL	0				
SELECT * FROM T WHERE ( C1, C2 ) IS NOT NULL	0				
SELECT * FROM T WHERE ( C2, C2 ) IS NOT NULL	1				
			10		

### Effects of Bad Language Design

There are general language design lessons to be learned from this tangled web, as well as lessons about NULL:

- Enclosing an expression in parens should not change its meaning. ( C1 ) is not the only example in SQL. Think of "scalar subqueries"
- Great caution is needed when considering pragmatic shorthands.
  (C1, C2) IS NULL was originally shorthand for C1 IS NULL AND C2 IS NULL.
- All data types supported by a language should be "first-class", for orthogonality. ROW types were originally not first-class – could not (for example) be the declared types of columns.

11



### 3-Valued Logic: The Real Culprit

Relational theory is founded on classical, 2-valued logic.

A relation r is interpreted as a representation of the extension of some predicate P.

Let *t* be a tuple with the same heading as *r*.

If tuple *t* is a member of *r*, then the proposition P(t) is taken to be TRUE; otherwise (*t* is not a member of *r*), P(t) is taken to be FALSE.

There is no middle ground. The Law of The Excluded Middle applies.

There is no way of representing that the truth of P(t) is unknown, or inapplicable, or otherwise concealed from us.

13

SQL's WHERE clause arbitrarily splits at the TRUE/UNKNOWN divide.

Case Study Example						
PERS_INFO						
	<u>ld</u>	Name	Job	Salary		
	1234	Anne	Lawyer	100,000		
	1235	Boris	Banker	?		
	1236	Cindy	?	70,000		
	1237	Devinder	?	?		
Meaning (a predicate): The person identified by <i>Id</i> is called <i>Name</i> and has the job of a <i>Job</i> , earning <i>Salary</i> pounds per year.						
BU	T WHAT DC	THOSE QUESTION	N MARKS MEAN	???	14	

### Summary of Proposed Solution

- 1. Database design:
- a. "vertical" decomposition
- b. "horizontal" decomposition
- 2. New constraint shorthands:
  - a. "distributed key"
- b. "foreign distributed key"
- 3. New database updating construct: "multiple assignment"
- Recomposition by query to derive (an improved) PERS\_INFO when needed

#### Database Design

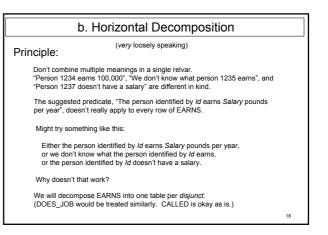
a. "vertical" decomposition

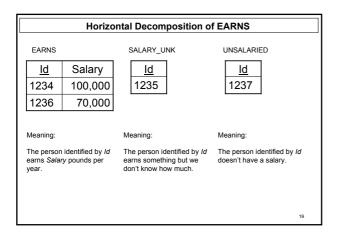
Decompose into 2 or more relvars by *projection* Also known as *normalization*. Several degrees of normalization were described in the 1970s: 1NF, 2NF, 3NF, BCNF, 4NF, 5NF.

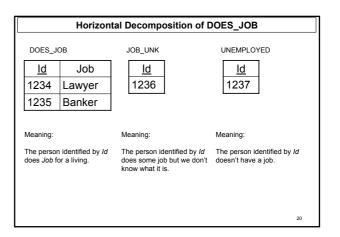
The ultimate degree, however, is 6NF: "irreducible relations". (See "Temporal Data and The Relational Model", Date/Darwen/Lorentzos, 2003.)

A 6NF relvar consists of a key plus at most one other attribute.

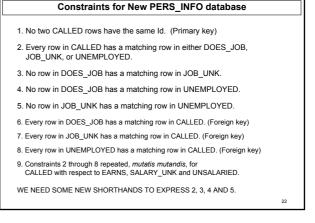
	Vertical Decomposition of PERS_INFO						
CALLED		DOES_JOB				EARNS	
ld	Name		ld	Job		<u>ld</u>	Salary
1234	Anne		1234	Lawyer		1234	100,000
1235	Boris		1235	Banker		1235	?
1236	Cindy		1236	?		1236	70,000
1237	Devinder		1237	?		1237	?
Meaning:	Meaning: Meaning: Meaning:						
					earns Salai	identified by <i>Id</i> y pounds per	
BUT WHAT DO THOSE QUESTION MARKS MEAN? (reprise) 17					17		







What We Have Achieved So Far	Constrain
What started as a single table (PERS_INFO) is now a <i>database (sub)schema</i> (let's call it PERS_INFO again), consisting of:	1. No two CALLED rows
CALLED ( Id, Name )	2. Every row in CALLED JOB_UNK, or UNEMF
DOES_JOB ( ld, Job )	3. No row in DOES_JOB
JOB_UNK ( ld )	4. No row in DOES_JOB
UNEMPLOYED ( Id )	5. No row in JOB_UNK h
EARNS ( Id, Salary )	6. Every row in DOES_JOB
SALARY_UNK ( ld )	7. Every row in JOB_UNK h
UNSALARIED ( ld )	8. Every row in UNEMPLOY
Next, we must consider the constraints needed to hold this design together (so to	9. Constraints 2 through 8 r CALLED with respect to
speak). 21	WE NEED SOME NEW SH



#### Proposed Shorthands for Constraints

- 1. Id is a *distributed key* for (DOES\_JOB, JOB\_UNK, UNEMPLOYED). This addresses Constraints 3, 4 and 5.
- 2. Id is a distributed key for (EARNS, SALARY\_UNK, UNSALARIED).
- Id is a foreign distributed key in CALLED, referencing (DOES\_JOB, JOB\_UNK, UNEMPLOYED).
   This addresses Constraint 2.
- Id is a foreign distributed key in CALLED, referencing (EARNS, SALARY\_UNK, UNSALARIED).

Plus regular foreign keys in each of DOES\_JOB, JOB\_UNK, UNEMPLOYED, EARNS, SALARY UNK, UNSALARIED, each referencing CALLED. (Might also want UNEMPLOYED to *imply* UNSALARIED – how would that be expressed?)

So, now we have a schema and constraints. Next, how to add the data and subsequently update it? Are the regular INSERT/UPDATE/DELETE operators good enough? 23

#### Updating the Database: A Problem

How can we add the first row to any of our 7 tables?

Can't add a row to CALLED unless there is a matching row in DOES\_JOB, JOB\_UNK or UNEMPLOYED and also a matching row in EARNS, SALARY\_UNK or UNSALARIED.

Can't add a row to DOES\_JOB unless there is a matching row in CALLED. Ditto JOB\_UNK, UNEMPLOYED, EARNS, SALARY\_UNK and UNSALARIED.

Impasse!

24

28

30

#### Updating the Database: Solution

"Multiple Assignment": doing several updating operations in a single "mouthful".

For example:

INSERT\_TUPLE INTO CALLED { Id 1236, Name 'Cindy' } , INSERT\_TUPLE INTO JOB\_UNK { Id 1236 } , INSERT\_TUPLE INTO EARNS { Id 1236, Salary 70000 } ;

Note very carefully the punctuation!

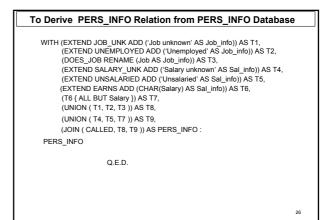
This triple operation is "atomic". Either it all works or none of it works.

Loosely speaking: operations are performed in the order given (to cater for the same target more than once), but intermediate states might be inconsistent and are not visible.

25

27

So, we now have a *working* database design. Now, what if the user wants to derive that original PERS\_INFO table from this database?



The New PERS_INFO					
PERS_INFO					
ld	Name	Job_info	Sal_info		
1234	Anne	Lawyer	100,000		
1235	Boris	Banker	Salary unknown		
1236	Cindy	Job unknown	70,000		
1237	Devinder	Unemployed	Unsalaried		

LOOK - NO QUESTION MARKS, NO NULLS!

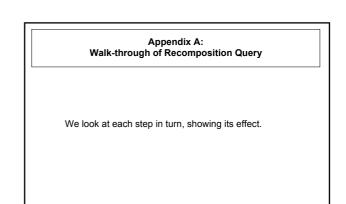
# How Much of All That Can Be Done Today? Vertical decomposition: can be done in SQL Horizontal decomposition: can be done in SQL

- Primary and foreign keys: can be done in SQL
- Distributed keys: can be done in (awful) longhand, if at all
- Foreign distributed keys can be done in (awful) longhand, if at all
- Multiple assignment: hasn't caught the attention of SQL DBMS vendors, but Alphora's D4 supports it. (So does Dave Voorhis's *Rel.*)

 Recomposition query: can be done but likely to perform horribly. Might be preferable to store PERS\_INFO as a single table under the covers, so that the tables resulting from decomposition can be implemented as mappings to that. But current technology doesn't give clean separation of physical storage from logical design.

Perhaps something for the next generation of software engineers to grapple with?

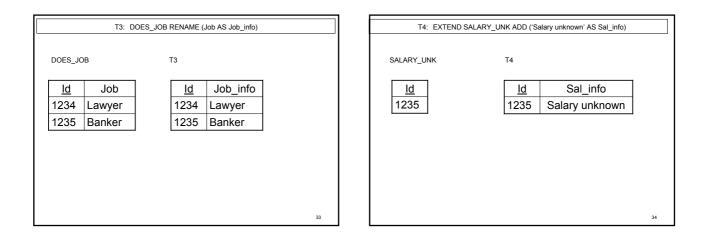
The End	
(Appendix A follows)	
	20

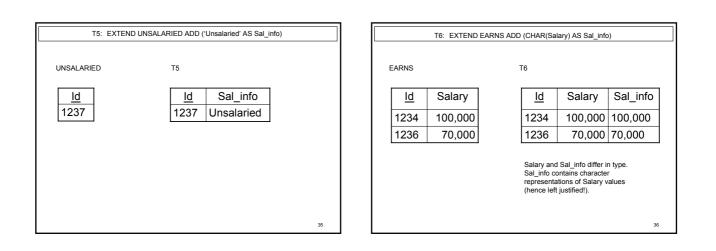


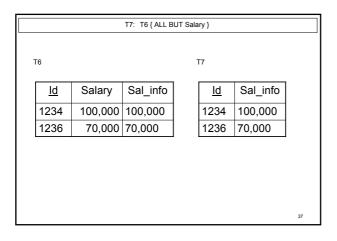
CS319: Theory of Databases

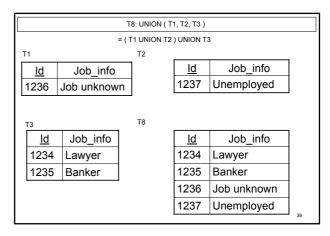
T1: EXTEND JOB_UNK ADD ('Job unknown' AS Job_info)			
JOB_UNK	T1		
<u>ld</u> 1236	<u>ld</u> 1236	Job_info Job unknown	
			31

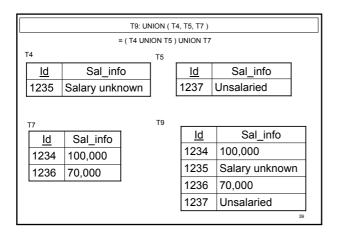
T2: EXTEND UNE	MPLOYED ADD ('	Unemployed' AS Job_info	))
UNEMPLOYED	T2		
ld	ld	Job_info	
1237	1237	Unemployed	
			32











PERS INFO: JOIN ( CALLED, T8, T9 )									
CALLED T8 T9									
ld	Name		ld		Job_info		ld	Sal_info	
1234	ŀ	Anne		34	Lawyer		1234	100,000	
1235	E	Boris		35	Banker		1235	Salary unknown	
1236	C	Cindy	1236		Job unknown		1236	70,000	
1237	De	vinder	1237		Unemployed		1237	Unsalaried	
PERS_INFO									
<u>ld</u>		Name		Job_info		Sal_info			
1234		Anne		Lawyer		100,000			
1235		Boris		Banker		Salary unknown			
1236		Cindy		Job unknown		70,000			
1237		Devinder		Unemployed		Unsalaried			40

