Outer Joins

The joins you know from RA

These can go in a FROM clause:

Expression	Meaning
R, S	R×S
R cross join S	1
R natural join S	R ⋈ S
R join S on Condition	$R \bowtie_{condition} S$



In practise, natural join is brittle

- A working query can be broken by adding a column to a schema.
 - Example:

```
SELECT sID, instructor
FROM Student NATURAL JOIN Took
NATURAL JOIN Offering;
```

- What if we add a column called campus to Offering?
- Also, having implicit comparisons impairs readability.
- Best practise: Don't use natural join.

Dangling tuples

- With joins that require some attributes to match, tuples lacking a match are left out of the results.
- We say that they are "dangling".
- An outer join preserves dangling tuples by padding them with NULL in the other relation.
- A join that doesn't pad with NULL is called an inner join.



Three kinds of outer join

- LEFT OUTER JOIN
 - Preserves dangling tuples from the relation on the LHS by padding with nulls on the RHS.
- RIGHT OUTER JOIN
 - The reverse.
- FULL OUTER JOIN
 - Does both.

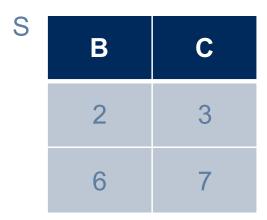


Example: joining R and S various ways

A B

1 2

4 5



R NATURAL JOIN S

Α	В	С
1	2	3



Example

A B

1 2

4 5

S B C 2 3 6 7

R NATURAL FULL JOIN S

Α	В	С
1	2	3
4	5	NULL
NULL	6	7



Example

A B

1 2

4 5

S B C 2 3 6 7

R NATURAL LEFT JOIN S

Α	В	С
1	2	3
4	5	NULL



Example

A B

1 2

4 5

S B C 2 3 6 7

R NATURAL RIGHT JOIN S

Α	В	С
1	2	3
NULL	6	7



Summary of join expressions

Cartesian product

```
A CROSS JOIN B
```

same as A, B

Theta-join

```
A JOIN B ON C
```

✓A {LEFT|RIGHT|FULL} JOIN B ON C

Natural join

```
A NATURAL JOIN B
```

✓A NATURAL {LEFT|RIGHT|FULL} JOIN B

✓ indicates that tuples are padded when needed.



Keywords INNER and OUTER

- There are keywords INNER and OUTER, but you never need to use them.
- Your intentions are clear anyway:
 - You get an outer join iff you use the keywords LEFT, RIGHT, or FULL.
 - If you don't use the keywords LEFT, RIGHT, or FULL you get an inner join.



Impact of having null values

Missing Information

- Two common scenarios:
 - Missing value.
 E.g., we know a student has some email address, but we don't know what it is.
 - Inapplicable attribute.
 E.g., the value of attribute spouse for an unmarried person.



Representing missing information

- One possibility: use a special value as a placeholder. E.g.,
 - If age unknown, use 0.
 - If StNum unknown, use 9999999999.
- Implications?
- Better solution: use a value not in any domain.
 We call this a null value.
- Tuples in SQL relations can have NULL as a value for one or more components.



Checking for null values

- You can compare an attribute value to NULL with
 - IS NULL
 - IS NOT NULL
- Example:

```
SELECT *
FROM Course
WHERE breadth IS NULL;
```



In SQL we have 3 truth-values

- Because of NULL, we need three truth-values:
 - If one or both operands to a comparison is NULL, the comparison *always* evaluates to UNKNOWN.
 - Otherwise, comparisons evaluate to TRUE or FALSE.



Combining truth values

- We need to know how the three truth-values combine with AND, OR and NOT.
- Can think of it in terms of the truth table.
- Or can think in terms of numbers:
 - TRUE = I, FALSE = 0, UNKNOWN = 0.5
 - AND is min, OR is max,
 - NOT x is (I-x), i.e., it "flips" the value



The three-valued truth table

A	В	A and B	A or B
Т	Т	Т	Т
TF c	r FT	F	Т
F	F	F	F
TU c	r UT	U	Т
FU c	r UF	F	U
U	U	U	U



Thinking of the truth-values as numbers

A	В	as nums	A and B	min	A or B	max
Т	Т	1, 1	Т	1	Т	1
TF	or FT	1, 0	F	0	Т	1
F	F	0, 0	F	0	F	0
TU	or UT	1, 0.5	U	0.5	Т	1
FU	or UF	0, 0.5	F	0	U	0.5
U	U	0.5, 0.5	U	0.5	U	0.5



Thinking of the truth-values as

A	as a num, x	not A	1 - x
Т	1	F	0
F	0	Т	1
U	0.5	U	0.5



Surprises from 3-valued logic

- Some laws you are used to still hold in threevalued logic. For example,
 - AND is commutative.
- But others don't. For example,
 - The law of the excluded middle breaks:
 (p or (NOT p)) might not be TRUE!
 - (0*x) might not be 0.



Impact of null values on WHERE

- A tuple is in a query result iff the WHERE clause is TRUE.
- UNKNOWN is not good enough.
- "WHERE is picky."
- Example: where-null



Impact of null values on aggregation

- Summary: Aggregation ignores NULL.
 - NULL never contributes to a sum, average, or count, and
 - can never be the minimum or maximum of a column (unless every value is NULL).
- If there are no *non*-NULL values in a column, then the result of the aggregation is NULL.
 - Exception: COUNT of an empty set is 0.



Aggregation ignores nulls

	some nulls in A	All nulls in A	
min(A)			
max(A)		null	
sum(A)	ignore the nulls		
avg(A)			
count(A)		0	
count(*)	all tuples count		

Example: aggregation-nulls

More re the impact of null values

- Other corner cases to think about:
 - SELECT DISTINCT: are 2 NULL values equal?
 - natural join: are 2 NULL values equal?
 - set operations: are 2 NULL values equal?
- And later, when we learn about constraints:
 - UNIQUE constraint: do 2 NULL values violate?

This behaviour may vary across DBMSs.



Summary re NULL

- Any comparison with NULL yields UNKNOWN.
- WHERE is picky: it only accepts TRUE.
- Therefore NATURAL JOIN is picky too.
- Aggregation ignores NULL.
- In other situations where NULLs matter
 - when a truth-value may be NULL
 - when it matters whether two NULL are considered the same
 - Don't assume. Behaviour may vary by DBMS.

