

# CSC 261/461 – Database Systems

## Lecture 4

Fall 2017

# Announcement

- You can now form a team of:

Three!

# Topics Covered

1. NULLs
2. Outer Joins
3. With and Case
4. Constraint
5. Schema Change Statements

# Comparisons Involving NULL and Three-Valued Logic (cont'd.)

**Table 7.1** Logical Connectives in Three-Valued Logic

(a)	<b>AND</b>	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	FALSE	UNKNOWN
	FALSE	FALSE	FALSE	FALSE
	UNKNOWN	UNKNOWN	FALSE	UNKNOWN
(b)	<b>OR</b>	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE	UNKNOWN
	UNKNOWN	TRUE	UNKNOWN	UNKNOWN
(c)	<b>NOT</b>			
	TRUE	FALSE		
	FALSE	TRUE		
	UNKNOWN	UNKNOWN		

# NULLS in SQL

- Whenever we don't have a value, we can put a NULL
- Can mean many things:
  - Value does not exist
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- The schema specifies for each attribute if it can be null (*nullable* attribute) or not
- Each individual NULL value is considered to be different from every other NULL value
- SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN (like Maybe)
- **NULL = NULL comparison is avoided**
- How does SQL cope with tables that have NULLs?

# Null Values

Unexpected behavior:

```
SELECT *  
FROM Person  
WHERE age < 25 OR age >= 25
```

Some Persons are not included !

# Null Values

Can test for NULL explicitly:

- x IS NULL
- x IS NOT NULL

```
SELECT *  
FROM Person  
WHERE age < 25 OR age >= 25  
      OR age IS NULL
```

Now it includes all Persons!

# RECAP: Inner Joins

By default, joins in SQL are “inner joins”:

```
Product(name, category)
Purchase(prodName, store)
```

```
SELECT Product.name, Purchase.store
FROM   Product
       JOIN Purchase ON Product.name = Purchase.prodName
```

```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

Both equivalent:  
Both INNER JOINS!



# Inner Joins + NULLS = Lost data?

By default, joins in SQL are “inner joins”:

```
Product(name, category)
Purchase(prodName, store)
```

```
SELECT Product.name, Purchase.store
FROM   Product
       JOIN Purchase ON Product.name = Purchase.prodName
```

```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

However: Products that never sold (with no Purchase tuple) will be lost!

# Outer Joins

- An **outer join** returns tuples from the joined relations that don't have a corresponding tuple in the other relations
  - I.e. If we join relations A and B on  $a.X = b.X$ , and there is an entry in A with  $X=5$ , but none in B with  $X=5$ ...
    - A LEFT OUTER JOIN will return a tuple (a, NULL)!

- Left outer joins in SQL:

```
SELECT Product.name, Purchase.store
FROM   Product
LEFT OUTER JOIN Purchase ON
       Product.name = Purchase.prodName
```

Now we'll get products even if they didn't sell

# INNER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Purchase.store
FROM Product
INNER JOIN Purchase
ON Product.name = Purchase.prodName
```



name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Note: another equivalent way to write an INNER JOIN!

# LEFT OUTER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```



name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

# Other Outer Joins

- Left outer join:
  - Include the left tuple even if there's no match
- Right outer join:
  - Include the right tuple even if there's no match
- Full outer join:
  - Include the both left and right tuples even if there's no match

# Acknowledgement

- Some of the slides in this presentation are taken from the slides provided by the authors.
- Many of these slides are taken from cs145 course offered by Stanford University.
- Thanks to YouTube, especially to [Dr. Daniel Soper](#) for his useful videos.