CSC 261/461 – Database Systems
Lecture 4

Spring 2018
Topics Covered

1. NULLs
2. Outer Joins
3. With and Case
4. Constraint
5. Schema Change Statements
## Table 7.1 Logical Connectives in Three-Valued Logic

<table>
<thead>
<tr>
<th></th>
<th>AND</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td></td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>OR</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td></td>
<td>UNKNOWN</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>NOT</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c)</td>
<td>TRUE</td>
<td>FALSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>TRUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NULLS in SQL

• Whenever we don’t have a value, we can put a NULL

• Can mean many things:
  – Value does not exists
  – Value exists but is unknown
  – Value not applicable
  – Etc.

• The schema specifies for each attribute if can be null (nullable attribute) or not
• Each individual NULL value 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`

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

Now it includes all Persons!
```
By default, joins in SQL are “inner joins”:

```
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!
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!
• 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 $(a, \text{NULL})$!

• Left outer joins in SQL:

  ```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:

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>prodName</th>
<th>store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>

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

Note: another equivalent way to write an INNER JOIN!
### LEFT OUTER JOIN:

#### Product

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

#### Purchase

<table>
<thead>
<tr>
<th>prodName</th>
<th>store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>

#### SQL Query

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