NULL VALUES

CHAPTER 5 (6/E)
CHAPTER 8 (5/E)
LECTURE OUTLINE

- Dealing with null values
  - Three-valued logic
  - Effects in WHERE clauses
    - IS NULL
  - Effects on aggregation
  - Effects on GROUP BY, set operations, and SELECT DISTINCT
  - Treatment in ORDER BY clauses
  - Effects in CHECK constraints

- Outer joins
SEMANTICS OF NULL

- Recall possible meanings of NULL
  - Unknown value
  - Unavailable or withheld value
  - Not applicable attribute
- Each stored NULL value incomparable to every other stored value
  - Even if other value also NULL
    - \( \text{unknown} \not\leq 5 \rightarrow \text{unknown} \)
    - \( \text{unknown} \not\geq \text{unknown} \rightarrow \text{unknown} \)
  - Comparisons involving unknown values are neither true nor false.
- Thus, SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN
Similarly, any operation involving an unknown value produces an unknown value for the result.

- e.g., \( \text{unknown} + 5 \rightarrow \text{unknown} \)
Recall that *WHERE* clause evaluates each tuple in turn and returns only those for which the condition evaluates to *true*.

Tuples that evaluate to *false* or *unknown* are rejected.

- Cannot use

  ```sql
  WHERE phone = NULL
  ```

  to test for null value in a tuple.

Many tautologies do not hold for columns with NULLs.

- e.g., no “law of the excluded middle”

  ```sql
  SELECT *
  FROM Student
  WHERE age > 18 OR NOT age > 18
  ```

might *not* return all Student tuples
Most SQL operators are **NULL-intolerant**.
- They return unknown if an operand is NULL.
SQL provides special test that is NULL-tolerant

\[
\text{IS [NOT] NULL}
\]

**Query 18.** Retrieve the names of all employees who do not have supervisors.

```
Q18: SELECT Fname, Lname
    FROM EMPLOYEE
    WHERE Super_ssn IS NULL;
```

Need to account for NULLs when formulating queries
- Not handling NULLs is a common source of errors
Consider aggregating values for budget in the following.

- e.g., `max(budget)`, `sum(budget)`, `average(budget)`

- NULL values in tuples *ignored* for aggregation (even for `COUNT`)
  - Only non-NULL values included in aggregations.
  - i.e., `sum()` handled differently from `+`

- Example:

```sql
SELECT COUNT(*), COUNT(budget), AVERAGE(gross-budget)
FROM Film
WHERE genre = 'comedy';
```

- all comedies counted for first aggregation;
- only comedies with non-NULL budget counted for second aggregation;
- only comedies with non-NULL budget and non-NULL gross included in third aggregation
WHEN ALL NULLS ARE TREATED EQUAL

- Grouping and set operations treat all NULLs as the same value
  - e.g., GROUP BY budget forms separate group for all tuples with NULL value in budget
  - Similarly for set operations: all NULLs treated as if a single value
    - e.g., \{(A,B,NULL),(A,B,C)\} \cap \{(A,B,D),(A,B,NULL)\} = \{(A,B,NULL)\}

(SELECT genre, budget
 FROM Film
 WHERE gross > 15000000)
 UNION
 (SELECT genre, budget
 FROM Film
 WHERE year > 2000)

- Similarly, too, for duplicate elimination with SELECT DISTINCT

- Finally ORDER BY

- NULLs sorted together, but sort order with respect to other values is implementation-dependent
NULLS IN SQL’S DDL

- By default, must be aware of possible NULLs for all columns.
- Recall, however, a column can be declared **NOT NULL**.
  - NULL values cannot occur; querying simplified
  - Recall: Primary key columns must be declared **NOT NULL**

- **Unlike WHERE clause**, **CHECK constraints and FOREIGN KEY constraints ensure that no tuple returns false.**
  - Therefore NULLs accepted
  - e.g.,
    
    ```
    CHECK (age > 18)
    ```
    
    allows tuples with NULL value for age
JOIN OPERATOR

- For convenience, SQL’s join operator (algebra’s $\bowtie$) permits users to specify a table resulting from a join operation.
  
  \[ \text{Table1 [INNER] JOIN Table2 ON <condition>} \]

  - May appear in the **FROM** clause of a query.
  - Keyword **INNER** is optional.
  - Result is a single **joined table**.
  - Equivalent to including `<condition>` in **WHERE** clause.
  - Number of rows in result in range \([0, |\text{Table1}| \times |\text{Table2}|]\)
    - Data from Table1 appear in result only if matching row exists in Table2.
    - Data from Table2 appear in result only if matching row exists in Table1.

Q1A: SELECT Fname, Lname, Address FROM (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber) WHERE Dname=‘Research’;
**LEFT OUTER JOIN OPERATOR**

- Every tuple in left table appears in result
  - If matching tuple(s) in right table, works like inner join
  - If no matching tuple in right table, one tuple in result with left tuple values padded with NULL values for columns of right table

```
Table1 LEFT [OUTER] JOIN Table2 ON <condition>
SELECT *
FROM Customer LEFT JOIN Sale ON Customer.custid = Sale.custid
```
OTHER OUTER JOIN OPERATORS

- **Table1 RIGHT [OUTER] JOIN Table2 ON <condition>**
  - Every tuple in right table appears in result (padded on left if needed)

- **Table1 FULL [OUTER] JOIN Table2 ON <condition>**
  - Every tuple in either table appears in result (padded if needed)

```
SELECT *
FROM Customer FULL JOIN Sale ON Customer.custid = Sale.custid
```

### Customer Table

<table>
<thead>
<tr>
<th>custid</th>
<th>name</th>
<th>address</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1205</td>
<td>Lee</td>
<td>633 S. First</td>
<td>555-1219</td>
</tr>
<tr>
<td>3122</td>
<td>Willis</td>
<td>41 King</td>
<td>555-9876</td>
</tr>
<tr>
<td>2134</td>
<td>Smith</td>
<td>213 Main</td>
<td>555-1234</td>
</tr>
<tr>
<td>1697</td>
<td>Ng</td>
<td>5 Queen N.</td>
<td>555-0025</td>
</tr>
<tr>
<td>3982</td>
<td>Harrison</td>
<td>808 Main</td>
<td>555-4829</td>
</tr>
</tbody>
</table>

### Sale Table

<table>
<thead>
<tr>
<th>saleid</th>
<th>date</th>
<th>custid</th>
</tr>
</thead>
<tbody>
<tr>
<td>A17</td>
<td>5 Dec</td>
<td>3122</td>
</tr>
<tr>
<td>B823</td>
<td>5 Dec</td>
<td>1697</td>
</tr>
<tr>
<td>B219</td>
<td>9 Dec</td>
<td>3122</td>
</tr>
<tr>
<td>C41</td>
<td>15 Dec</td>
<td>1205</td>
</tr>
<tr>
<td>X00</td>
<td>23 Dec</td>
<td>0000</td>
</tr>
</tbody>
</table>

### Result Table

<table>
<thead>
<tr>
<th>Customer.custid</th>
<th>name</th>
<th>address</th>
<th>phone</th>
<th>saleid</th>
<th>date</th>
<th>Sale.custid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1205</td>
<td>Lee</td>
<td>633 S. First</td>
<td>555-1219</td>
<td>C41</td>
<td>15 Dec</td>
<td>1205</td>
</tr>
<tr>
<td>3122</td>
<td>Willis</td>
<td>41 King</td>
<td>555-9876</td>
<td>A17</td>
<td>5 Dec</td>
<td>3122</td>
</tr>
<tr>
<td>3122</td>
<td>Willis</td>
<td>41 King</td>
<td>555-9876</td>
<td>B219</td>
<td>9 Dec</td>
<td>3122</td>
</tr>
<tr>
<td>2134</td>
<td>Smith</td>
<td>213 Main</td>
<td>555-1234</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>1697</td>
<td>Ng</td>
<td>5 Queen N.</td>
<td>555-0025</td>
<td>B823</td>
<td>5 Dec</td>
<td>1697</td>
</tr>
<tr>
<td>3982</td>
<td>Harrison</td>
<td>808 Main</td>
<td>555-4829</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>X00</td>
<td>23 Dec</td>
<td>0000</td>
</tr>
</tbody>
</table>
NULL values need careful consideration.

- Most operators are NULL-intolerant.
- Some queries must use IS [NOT] NULL to operate correctly.
- Aggregations ignore NULLs.
- Partitioning and set operators treat all NULLs as equal.
- Check constraints are NULL-tolerant.
- Include NOT NULL for column declarations where appropriate.
  - Recall: required for primary keys

Outer joins

- LEFT, RIGHT, and FULL