Nulls

In place of a value in a tuple’s component.

- Interpretation is not exactly “missing value.”
- There could be many reasons why no value is present, e.g., “value inappropriate.”

Comparing Nulls to Values

- 3rd truth value UNKNOWN.
- SELECT clause only lists tuples if the condition evaluates to TRUE (UNKNOWN is not sufficient).
Example

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's bar</td>
<td>Bud</td>
<td>NULL</td>
</tr>
</tbody>
</table>

SELECT bar
FROM Sells
WHERE price < 2.00 OR price >= 2.00;

UNKNOWN
UNKNOWN

UNKNOWN
3-Valued Logic

Think of true = 1; false = 0, and unknown = 1/2. Then:

- \( \text{AND} = \min \).
- \( \text{OR} = \max \).
- \( \text{NOT}(x) = 1 - x \).

Some Key Laws Fail to Hold

Example: Law of the excluded middle, i.e.,

\[
p \text{ OR NOT } p = \text{ TRUE}
\]

- For 3-valued logic: if \( p \) = unknown, then left side = \( \max(1/2,(1-1/2)) = 1/2 \neq 1 \).
- Like bag algebra, there is no way known to make 3-valued logic conform to all the laws we expect for sets/2-valued logic, respectively.
Outerjoin

\( R \Join S = R \Join S \) with *dangling* tuples padded with nulls and included in the result.

- A tuple is dangling if it doesn’t join with any other tuple.

\( R = \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

\( S = \)

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

\( R \Join S = \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Outerjoin in SQL2

A number of forms are provided.

- Can be used either stand-alone (in place of a select-from-where) or to define a relation in the FROM-clause.

  \[
  R \text{ NATURAL JOIN } S \\
  R \text{ JOIN } S \text{ ON condition} \\
  \text{e.g., condition: } R.B = S.B \\
  R \text{ CROSS JOIN } S \\
  R \text{ OUTER JOIN } S \\
  \]

- The last of these can be modified by:
  1. Optional \text{NATURAL} in front.
  2. Optional \text{ON} condition at end.
  3. Optional \text{LEFT, RIGHT, or FULL} before \text{OUTER}.

  \[ \text{LEFT} = \text{pad dangling tuples of } R \text{ only;} \]

  \[ \text{RIGHT} = \text{pad dangling tuples of } S \text{ only.} \]
Oracle Outerjoin

There is a rudimentary facility that allows either left or right outerjoin.

- Add (+) to one side of the equality that forms a join between two tables.

Example

List the beers sold by Joe’s Bar, with their manufacturers, but include the beer even if the manufacturer is not known.

Beers(name, manf)
Sells(bar, beer, price)

SELECT beer, manf
FROM Sells, Beers
WHERE bar = 'Joe’s Bar' AND
    beer = name(+);
Constraints

Commercial relational systems allow much more “fine-tuning” of constraints than do the modeling languages we learned earlier.

- In essence: SQL programming is used to describe constraints.

Outline

1. Primary key declarations (covered).
2. Foreign-keys = referential integrity constraints.
3. Attribute- and tuple-based checks = constraints within relations.
4. SQL2 Assertions = global constraints.
   - Not found in Oracle.
5. Oracle Triggers.
   - A substitute for assertions.
6. SQL3 triggers and assertions.
**Foreign Keys**

In relation $R$ a clause that “attribute $A$ references $S(B)$” says that whatever values appear in the $A$ column of $R$ must also appear in the $B$ column of relation $S$.

- $B$ must be declared the primary key for $S$.

**Example**

```sql
CREATE TABLE Beers (  
  name CHAR(20) PRIMARY KEY,  
  manf CHAR(20)  
);  

CREATE TABLE Sells (  
  bar CHAR(20),  
  beer CHAR(20) REFERENCES Beers(name),  
  price REAL  
);  
```
• Alternative: add another element declaring the foreign key, as:

```sql
CREATE TABLE Sells (  
    bar CHAR(20),  
    beer CHAR(20),  
    price REAL,  
    FOREIGN KEY beer REFERENCES Beers(name)  
);
```

• Extra element essential if the foreign key is more than one attribute.
What Happens When a Foreign Key Constraint is Violated?

- Two ways:

1. Insert or update a Sells tuple so it refers to a nonexistent beer.
   ♦ Always rejected.

2. Delete or update a Beers tuple that has a beer value some Sells tuples refer to.
   a) Default: reject.
   b) Cascade: Ripple changes to referring Sells tuple.

Example

- Delete “Bud.” Cascade deletes all Sells tuples that mention Bud.

- Update “Bud” → “Budweiser.” Change all Sells tuples with “Bud” in beer column to be “Budweiser.”
c) *Set Null*: Change referring tuples to have NULL in referring components.

**Example**

- Delete “Bud.” Set-null makes all *sells* tuples with “Bud” in the *beer* component have NULL there.
- Update “Bud” → “Budweiser.” Same change.
Selecting a Policy

Add ON [DELETE, UPDATE] [CASCADE, SET NULL] to declaration of foreign key.

Example

```sql
CREATE TABLE Sells (  
  bar CHAR(20),  
  beer CHAR(20),  
  price REAL,  
  FOREIGN KEY beer REFERENCES  
  Beers(name)  
  ON DELETE SET NULL  
  ON UPDATE CASCADE 
);```

- “Correct” policy is a design decision.
  - E.g., what does it mean if a beer goes away? What if a beer changes its name?
Attribute-Based Checks

Follow an attribute by a condition that must hold for that attribute in each tuple of its relation.

- Form: CHECK (condition).
  - Condition may involve the checked attribute.
  - Other attributes and relations may be involved, but *only* in subqueries.
  - Oracle: *No subqueries allowed in condition.*

- Condition is checked only when the associated attribute changes (i.e., an insert or update occurs).
Example

```
CREATE TABLE Sells (    
  bar CHAR(20),    
  beer CHAR(20) CHECK(        
    beer IN (SELECT name        
      FROM Beers)    
  ),    
  price REAL CHECK(        
    price <= 5.00    
  )    
);
```

- Check on `beer` is like a foreign-key constraint, except:
  - The check occurs only when we add a tuple or change the beer in an existing tuple, not when we delete a tuple from `Beers`. 
Tuple-Based Checks

Separate element of table declaration.

- Form: like attribute-based check.
- But condition can refer to any attribute of the relation.
  - Or to other relations/attributes in subqueries.
  - Again: Oracle forbids the use of subqueries.
- Checked whenever a tuple is inserted or updated.
Example

Only Joe’s Bar can sell beer for more than $5.

CREATE TABLE Sells (  
  bar CHAR(20),  
  beer CHAR(20),  
  price REAL,  
  CHECK(bar = 'Joe''s Bar' OR  
  price <= 5.00)  
);

16