Null Values

- Field values in a tuple may sometimes be
  - unknown: e.g., a rating has not been assigned, or a new column is added to the table
  - inapplicable: e.g., CEO has no manager, single person has no spouse
- SQL provides a special value NULL for such situations
  - Special operators IS NULL, IS NOT NULL
  - SELECT FROM Sailors WHERE rating IS NOT NULL
  - Note: NULL must not be used as constant in expressions!
  - A field can be declared as NOT NULL, means NULL values are not allowed (by default, PK fields are NOT NULL)

Dealing with Null Values

- The presence of NULL complicates some issues
  - NULL op value has as result NULL (op is +,*,/,−)
  - What does rating>8 evaluate to if rating is equal to NULL?
  - Answer: unknown
  - 3-valued logic: true, false and unknown
    - Recall that WHERE eliminates rows that don’t evaluate to true
    - What about AND, OR and NOT connectives?
      - unknown AND true = unknown
      - unknown OR false = unknown
      - NOT unknown = unknown
    - Also, <NULL_value> = <NULL_value> is unknown!

Null Values and Aggregates

- The COUNT(*) result includes tuples with NULL
- COUNT(A) only counts tuples where value of attribute A is not NULL
- All other aggregates skip NULL values (if aggregate is on the field that is NULL)
  - If all values are NULL on the aggregated field, the result of aggregate is also NULL (except COUNT which returns 0)

Null Values and Aggregates

Following two queries DO NOT RETURN SAME RESULT if there are NULLs in field name:

SELECT COUNT(*) FROM Sailors S

SELECT COUNT(S.name) FROM Sailors S

Following two queries DO NOT RETURN SAME RESULT if there are NULLs in field rating:

SELECT COUNT(*) FROM Sailors S

SELECT COUNT(*) FROM Sailors
WHERE (rating>8) OR (rating <= 8)

Null Values and Duplicates

- Comparing two NULL values gives as result unknown
  - But there are anomalies when checking for duplicates!
    - NULL values are considered equal in this case!
    - Two tuples are duplicates if they match in all non-NULL attributes
  - Implications for DISTINCT, UNIQUE subqueries, set operations!
    - Tuples with NULL in some group-by attributes placed in same group if all non-NUL-
      group-by attributes match!
    - DISTINCT: if multiple tuples have equal values in all non-
      NULL attributes only one of them is output
Outer Joins

- Include in join result non-matching tuples
- Result tuple padded with NULL values
- Variants
  - FULL: non-matching tuples in both relations included in result
  - LEFT: only non-matching tuples in left relation included in result
  - RIGHT: only non-matching tuples in right relation included in result

Join Expressions

- SQL shorthands for expressions we already saw
  - Cross Product: \( \text{Sailors CROSS JOIN Reserves} \)
  - Condition Join: \( \text{Sailors JOIN Reserves ON } \langle \text{condition} \rangle \)
  - Natural Join: \( \text{Sailors NATURAL JOIN Reserves} \)

Usage Example:

```
SELECT *
FROM Sailors JOIN Reserves ON Sailors.sid = Reserves.sid
```

Sample Schema

<table>
<thead>
<tr>
<th>ENROLLED</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>CID</td>
<td>CID</td>
<td>CID</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STUDENT</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>SNAME</td>
<td>MAJOR</td>
<td>AGE</td>
</tr>
<tr>
<td>1</td>
<td>Alice</td>
<td>CS</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>Economics</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>Carl</td>
<td>CS</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>Denise</td>
<td>History</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COURSE</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CID</td>
<td>CNAME</td>
<td>ROOM</td>
<td>FID</td>
</tr>
<tr>
<td>1</td>
<td>Java</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Micro</td>
<td>203</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>US Hist</td>
<td>201</td>
<td>1</td>
</tr>
</tbody>
</table>

Outer Joins

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
</tr>
</tbody>
</table>

Selected sid, sname, rating, age, bid, day
FROM Sailors NATURAL LEFT OUTER JOIN Reserves

Integrity Constraints (Review)

- An IC describes conditions that every legal instance of a relation must satisfy.
- Inserts/deletes/updates that violate IC’s are disallowed.

Types of IC’s:
- Domain constraints
  - Field values must be of right type - always enforced
- Primary key constraints
- Foreign key constraints
- General constraints

Enforcing Referential Integrity

- What should be done if an Enrolled tuple with a non-existent student id is inserted?
  - Reject the insert!

- What should be done if a Students tuple is deleted?
  - Delete all Enrolled tuples that refer to it
  - Correct as far as IC is concerned, but data is lost!
  - Disallow deletion of a Students tuple that is referred to
  - More appropriate in practice
  - Set sid in Enrolled tuples that refer to it to a default sid
  - Or, set it to NULL
Referential Integrity in SQL

- SQL/92 and SQL:1999 support all options on deletes and updates.
- Default is **NO ACTION** (delete/update is rejected)
- **CASCADE** (delete/update all tuples that refer to deleted/updated tuple)
- **SET NULL** / **SET DEFAULT** (sets foreign key value of referencing tuple)

```sql
CREATE TABLE Enrolled
(sid CHAR(20) default '00',
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid)
REFERENCES Students (sid)
ON DELETE SET DEFAULT
ON UPDATE CASCADE)
```

Complex Constraints: CHECK clause

- Useful when more general ICs than keys are involved
- Can use queries to express constraint
- Constraints can be named
- Not checked if table is empty!
- Standalone CHECK for single table only!

```sql
CREATE TABLE Sailors
(sid INTEGER,
sname CHAR(10),
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CONSTRAINT RatingRange CHECK ( rating >= 1 AND rating <= 10 )
)
```

Complex Constraints: Assertions

- **Number of boats plus number of sailors is < 100**
- Not associated with a particular table
- Constraint may apply to multiple tables!

```sql
CREATE ASSERTION smallClub
CHECK
  ( SELECT COUNT(S.sid) FROM Sailors S ) +
  ( SELECT COUNT(B.bid) FROM Boats B ) < 100
```

---

**Complex Constraints: Assertions**

- **Number of boats plus number of sailors is < 100**
- Not associated with a particular table
- Constraint may apply to multiple tables!

```sql
CREATE ASSERTION smallClub
CHECK
  ( SELECT COUNT(S.sid) FROM Sailors S ) +
  ( SELECT COUNT(B.bid) FROM Boats B ) < 100
```