

Null Values. SQL Constraints

CS430/630
Lecture 10

Slides based on "Database Management Systems" 3rd ed, Ramakrishnan and Gehrke

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-NULL 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

Outer Joins

Sailors

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Reserves

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

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

<u>sid</u>	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
31	lubber	8	55.5	NULL	NULL
58	rusty	10	35.0	103	11/12/96

Join Expressions

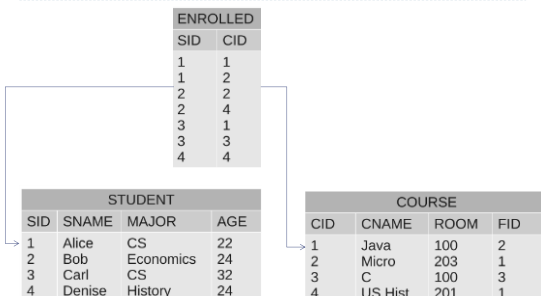
- ▶ SQL shorthands for expressions we already saw
- Cross Product:
 - Sailors **CROSS JOIN** Reserves
- Condition Join:
 - Sailors **JOIN** Reserves **ON** condition
- Natural Join:
 - Sailors **NATURAL JOIN** Reserves
- Usage Example:


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

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 ICs:
 - ▶ domain constraints
 - ▶ Field values must be of right type - always enforced
 - ▶ primary key constraints
 - ▶ foreign key constraints
 - ▶ general constraints

Sample Schema



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)

```
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)
```

15

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!

```
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!

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

16

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder