SQL: NULLs and Triggers Part 3

Chapter 5.5-5.10

Resources, extra office hours this week

Null Values

- Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., single have no spouse's name).
  - SQL provides a special value *null* for such situations.
- The presence of *null* complicates many issues. E.g.:
  - Special operators needed to check if value is/is not *null*.
  - Is *rating>8* true or false when *rating* is equal to *null*? What about AND, OR and NOT connectives?
  - We'll need a 3-valued logic (true, false and *unknown*).
  - Meaning of constructs must be defined carefully. (e.g., WHERE clause eliminates rows that don't evaluate to true.)
  - New operators (in particular, *outer joins*) possible/needed.

Dealing with Nulls

- Let's add a row to our small example (perhaps it takes a while to get a rating)
- How do Nulls impact queries?

Sailors:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
99	nubie	null	19.0

Reserves:

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

Finding NULLs

- SELECT * FROM Sailors WHERE rating IS NULL;

sid	sname	rating	age
99	nubie	null	19.0

- SELECT * FROM Sailors WHERE rating IS NOT NULL;

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

- SELECT COUNT(*) FROM Sailors
  WHERE rating IS NULL;

Null Math

- SELECT sname, rating*0.1 FROM Sailors;
- SELECT sname, rating*0 FROM Sailors;

sname	rating
dustin	0.7
lubber	0.8
rusty	1.0
nubie	null

sname	rating
dustin	0
lubber	0
rusty	0
nubie	null

- Any arithmetic operation involving a NULL results in a NULL, even if the result could be knowable like (fieldname * 0) or (fieldname – fieldname)

Null Relations

- When NULLs are compared with any value the result is UNKNOWN. UNKNOWN is a third logical value.
- SELECT * FROM Sailor
  WHERE rating > 7 AND AGE < 40.0;
- SELECT * FROM Sailor
  WHERE rating > 7 OR AGE < 40.0;

sid	sname	rating	age
58	rusty	10	35.0

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0
99	nubie	null	19.0

Null Logic

- Table of logic combinations

x	y	x AND y	x OR y	not x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Specifying the type of JOIN

- SELECT * FROM Sailors NATURAL JOIN Reserves;

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

- gives the same result as:
  SELECT * FROM Sailors S, Reserves R
  WHERE S.sid=R.sid;

- SELECT * FROM Sailors JOIN Reserves
  ON Sailors.sid = Reserves.sid;

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Cross Joins

- SELECT * FROM Sailors CROSS JOIN Reserves;

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96
99	nubie	null	19.0	22	101	10/10/96
99	nubie	null	19.0	58	103	11/12/96

- same as SELECT * FROM Sailors JOIN Reserves;

Left Outer Joins

- SELECT * FROM Sailors S LEFT OUTER JOIN Reserves R
  ON S.sid=R.sid;

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
31	lubber	8	55.5	31	null	null
58	rusty	10	35.0	58	103	11/12/96
99	nubie	null	19.0	99	null	null

- All members of the left-side relation appear in the result set with nulls filling in the unmatched right-side entries
- Forces total participation
- same result as:
  SELECT * FROM Sailors S NATURAL LEFT OUTER JOIN Reserves R;

Right Outer Joins

- SELECT * FROM Sailors S RIGHT OUTER JOIN Reserves R
  ON S.sid=R.sid;
- All members of the right-side relation appear in the result set with nulls filling in the unmatched right-side entries

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

- Gives same result as NATURAL JOIN in this case
- If we:
  INSERT INTO Reserves VALUES(60,101,'09/09/14');

More on Right Outer Joins

- INSERT INTO Reserves VALUES(60,101,’09/09/14’);
- SELECT * FROM Sailors S RIGHT OUTER JOIN Reserves R ON S.sid=R.sid;

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96
60	null	null	null	60	101	09/09/14

Full Outer Joins

- Lastly, to include all rows from the left and right relations
- SELECT * FROM Sailors S FULL OUTER JOIN Reserves R
  ON S.sid=R.sid;

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
31	lubber	8	55.5	31	null	null
58	rusty	10	35.0	58	103	11/12/96
99	nubie	null	19.0	99	null	null
60	null	null	null	60	101	09/09/14

Integrity Constraints (Review)

- An IC describes conditions that every *valid instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC's are disallowed.
  - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- *Types of IC's*: Domain constraints, primary key constraints, foreign key constraints, general constraints.
  - *Domain constraints*: Field values must be of right type. Always enforced.

General Constraints

- Useful when more general ICs than keys are involved.
- Can use queries to express constraint.
- Constraints can be named.

```
CREATE TABLE  Sailors(
        sid         INT,
        sname   TEXT,
        rating    INT,
        age         REAL,
        PRIMARY KEY  (sid),
        CHECK  (rating >= 1
                    AND rating <= 10)
);
```

General Constraints

- Useful when more general ICs than keys are involved.
- Constraints can be named.
- Can use queries to express constraint.

```
CREATE TABLE  Reserves(
        sname  CHAR(10),
        bid  INTEGER,
        day  DATE,
        PRIMARY KEY  (bid,day),
        CONSTRAINT  noInterlakeRes CHECK  (
              'Interlake' <> ( SELECT  B.bname
                               FROM  Boats B
                               WHERE  B.bid=bid)
        )
)
```

Constraints Over Multiple Relations

- Awkward and wrong!
- Adding Boats can break rule.
- ASSERTION is the right solution; not associated with either table.

Number of boats plus number of sailors is < 100

```
CREATE TABLE  Sailors(
        sid  INTEGER,
        sname  CHAR(10),
        rating  INTEGER,
        age  REAL,
        PRIMARY KEY  (sid),
        CHECK (
          (SELECT COUNT (S.sid) FROM Sailors S) +
          (SELECT COUNT (B.bid) FROM Boats B) < 100)
        )
)
```

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

Triggers

- Trigger: A procedure that is invoked automatically if specified changes occur to the DBMS

- Triggers have three parts:
  - *Event* (that activates the trigger)
  - *Condition* (tests whether the triggers should run)
  - *Action* (what happens if the trigger runs)

Triggers: Example

- Suppose there was a rule than no one with a rating less than five can reserve a green boat. The following trigger would enforce this rule:

```
CREATE TRIGGER RatingRuleForGreen
    BEFORE INSERT ON Reserves
BEGIN
    SELECT RAISE(FAIL, 'Sailor is not qualified’)
    WHERE EXISTS (SELECT * FROM Sailors S, Boats B
                  WHERE S.sid = new.sid AND S.rating < 5
                  AND B.bid = new.bid AND B.color = 'green');
END;
```

- Note the special variable “new” for accessing parameters of the original INSERT query

Triggers: Another Example

- Queries of one table can be made to have side-effects in other tables via triggers
- Example “Event Logging”
- We know dates of reservations, but not when they were made. This can be remedied using a trigger as follows:

```
CREATE TRIGGER insertLog
   AFTER INSERT ON Reserves
BEGIN
   INSERT INTO ReservesLog (sid, bid, resDate, madeDate)
   VALUES (new.sid, new.bid, new.date, DATE(‘NOW’);
END;
```

Another Trigger Example

- What does this trigger do?

```
CREATE TRIGGER StartOfficalRating
  AFTER INSERT ON Reserves
BEGIN
   UPDATE Sailor S SET rating = 1
   WHERE S.sid = new.sid and rating IS NULL;
END;
```

Summary

- SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- Relationally complete; in fact, significantly more expressive power than relational algebra.
- Even queries that can be expressed in RA can often be expressed more naturally in SQL.
- Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
  - In practice, users need to be aware of how queries are optimized and evaluated for best results.

Summary (Contd.)

- NULL for unknown field values brings in new capabilities for join operations many along with complications
- SQL allows specification of rich integrity constraints
- Triggers respond to changes in the database

Next time we'll embed SQL into real code