SQL: Constraints & Triggers

Chapter 5.6-5.10
Controlling Output Order

- SQL’s “ORDER BY” clause is used to sort tuples in either ascending or descending order.
- ORDER BY specifies attributes used in the sort

```sql
SELECT *
FROM Sailors
WHERE age > 18
ORDER BY rating
```

```sql
SELECT *
FROM Sailors
WHERE age > 18
ORDER BY rating DESC
```

```sql
SELECT *
FROM Sailors
WHERE age > 18
ORDER BY rating DESC, sname ASC
```
Controlling output size

- The “LIMIT” clause is used to limit the number of tuples returned by a “SELECT” statement.
- Useful for seeing a small number of examples, or “top-X” in combination with “ORDER BY”

```
SELECT *
FROM Sailors
LIMIT 5
```

```
<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
```

```
SELECT *
FROM Sailors
ORDER BY rating DESC
LIMIT 5
```

```
<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
</tbody>
</table>
```
Null Values

- Field values in a tuple are sometimes unknown (e.g., a rating has not been assigned) or inapplicable (e.g., 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 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.)

- Joins can also generate null entries
Types of JOINS

- Recall our “Baby” sailor database

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>1996-10-10</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>1996-11-12</td>
</tr>
</tbody>
</table>

- An “implied” join

```sql
SELECT S.sname, R.day
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
```

- An “explicit” join (inner join)

```sql
SELECT S.sname, R.day
FROM Sailors S JOIN Reserves R ON S.sid=R.sid
```

```sql
SELECT S.sname, R.day
FROM Sailors S INNER JOIN Reserves R ON S.sid=R.sid
```

```sql
SELECT S.sname, R.day
FROM Sailors S NATURAL JOIN Reserves R
```

“inner” implies only tuples that share the join condition appear in the result set.
Left and Right JOINS

- A “Left” JOIN returns a tuple for every row of the first, “left”, relation, even if it requires adding “Null” values

```
SELECT S.sname, R.day
FROM Sailors S LEFT JOIN Reserves R ON S.sid=R.sid
```

- Likewise a “Right” join returns a tuple for every row in the second, “right”, relation

```
SELECT R.day, B.bname
FROM Reserves R NATURAL RIGHT JOIN Boats B
```

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>1996-10-10</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>1996-11-12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>1996-10-10</td>
</tr>
<tr>
<td>Null</td>
<td>Interlake</td>
<td>Null</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>1996-11-12</td>
</tr>
</tbody>
</table>
FULL OUTER Joins

- The FULL OUTER JOIN keyword returns all rows from all tables with the specified attributes joined or null if there is no match.

SELECT S.sname, R.day, B.bname
FROM (Sailors S NATURAL LEFT JOIN Reserves R)
   FULL OUTER JOIN Boats B ON R.bid=B.bid

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>1996-10-10</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>1996-11-12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sname</th>
<th>day</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>dustin</td>
<td>1996-10-10</td>
<td>Interlake</td>
</tr>
<tr>
<td>lubber</td>
<td>Null</td>
<td>Null</td>
</tr>
<tr>
<td>Null</td>
<td>Null</td>
<td>Interlake</td>
</tr>
<tr>
<td>rusty</td>
<td>1996-11-12</td>
<td>Clipper</td>
</tr>
</tbody>
</table>
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.
  - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a nonempty 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 Constraint CHECKs

- CHECK clause
- Useful when more general ICs than keys are involved.
- Example: All ratings must be between 1 and 10

```sql
CREATE TABLE Sailors(
sid INTEGER,
sname TEXT,
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CHECK (rating >= 1 AND rating <= 10)
)
More complicated CHECKs

- Constraints can be named.
- Checks can contain nested subqueries
- Example: Disallow reservations of boats named “Interlake” by sailors with ratings less than 7
- “bid” and “sid” refer to values from the associated INSERT or UPDATE

CREATE TABLE Reserves(
sid INTEGER,
bid INTEGER,
day DATE,
PRIMARY KEY (bid, day),
CONSTRAINT NoInterlakeIfLessThan7 CHECK (‘Interlake’ <> (SELECT B.bname FROM Boats B WHERE B.bid=bid) OR 7 <= (SELECT S.rating FROM Sailor S WHERE S.sid=sid))
Constraints Over Multiple Relations

- Awkward and wrong!
- If Sailors is empty, the number of Boats tuples can be anything!
- ASSERTION is the right solution; not associated with either table.

```sql
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 )
```

```
Number of boats plus number of sailors is < 100
```
Triggers

❖ Trigger: procedure that starts 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 that “no one with a rating less than 5 can reserve a green boat”. The following trigger would enforce this rule, and generate a failure message:

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

Note the special variable “new” for accessing parameters of the invoking INSERT query.
Triggers: Another Example

- Changes in one table can cause 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:

```sql
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;
```
Summary

- NULLs provide a means for representing “unspecified” attribute values
- NULLs can be generated by special JOINs
- Wide range of JOIN operations-- Some retain the cardinality of specified relations
- SQL allows specification of rich integrity constraints
- Triggers respond to changes in the database