SQL: Queries, Constraints, Triggers
Part 2

Chapter 5.5-5.10
SQL’s Aggregate Operators

- Significant extension of relational algebra.
- Computation and summarization operations
- Appears in target-list of query
- Results aggregate rather than appear individually
- E.x. How many instances in the sailor relation?

```sql
SELECT COUNT (*)
FROM Sailors S
```

COUNT (*), COUNT ([DISTINCT] A), SUM ([DISTINCT] A), AVG ([DISTINCT] A), MAX (A), MIN (A)
More examples

- Average age of Sailors with a rating of 10?
  
  ```sql
  SELECT AVG(S.age)
  FROM Sailors S
  WHERE S.rating=10
  ```

- Names of all Sailors who have achieved the maximum rating
  
  ```sql
  SELECT S.sname
  FROM Sailors S
  WHERE S.rating=(SELECT MAX(S2.rating)
  FROM Sailors S2)
  ```
More examples (cont)

- How many distinct ratings for Sailors less than 40 years of age?

  \[
  \text{SELECT COUNT(DISTINCT S.rating) FROM Sailors S WHERE S.age < 40.0}
  \]

- How many reservations were made by Sailors less than 40 years old?

  \[
  \text{SELECT COUNT(*) FROM Sailors S, Reserves R WHERE S.sid = R.sid AND S.age < 40}
  \]
Find name and age of the oldest sailor(s)

- The first query is invalid! (We’ll look into the reason a bit later, when we discuss GROUP BY.)
- The third query is equivalent to the second query, and is allowed in the SQL/92 standard, but is not supported in some systems.

```sql
SELECT S.sname, MAX (S.age)
FROM Sailors S

SELECT S.sname, S.age
FROM Sailors S
WHERE S.age =
    (SELECT MAX (S2.age)
     FROM Sailors S2)

SELECT S.sname, S.age
FROM Sailors S
WHERE (SELECT MAX (S2.age)
       FROM Sailors S2)
    = S.age
```
Motivation for Grouping

- So far, we’ve applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to subgroups.

- Consider: Find the age of the youngest sailor for each rating level.
  - In general, we don’t know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):

    For $i = 1, 2, \ldots, 10$:
    ```sql
    SELECT MIN (S.age)
    FROM Sailors S
    WHERE S.rating = i
    ```
Queries With GROUP BY and HAVING

- The target-list contains
  (i) attribute names
  (ii) terms with aggregate operations (e.g., MIN (S.age)).

- The attribute list (i) must be a subset of grouping-list.
  Intuitively, each answer tuple corresponds to a group, and these attributes must have a single value per group. (A group is a set of tuples that have the same value for all attributes in grouping-list.)
Conceptual Evaluation

- The cross-product of relation-list is computed, tuples that fail qualification are discarded, unnecessary fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in grouping-list.

- The group-qualification is then applied to eliminate some groups. Expressions in group-qualification must have a single value per group!
  - In effect, an attribute in group-qualification that is not an argument of an aggregate op also appears in grouping-list. (SQL does not exploit primary key semantics here!)

- One answer tuple is generated per qualifying group.
Find age of the youngest sailor with age $\geq 18$, for each rating with at least 2 such sailors

SELECT S.rating, 
    MIN (S.age) AS minage 
FROM  Sailors S 
WHERE  S.age $\geq$ 18 
GROUP BY  S.rating 
HAVING  COUNT (*) > 1

Sailors instance:

<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>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>71</td>
<td>zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>74</td>
<td>horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>85</td>
<td>art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>bob</td>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>96</td>
<td>frodo</td>
<td>3</td>
<td>25.5</td>
</tr>
</tbody>
</table>

Answer relation:

<table>
<thead>
<tr>
<th>rating</th>
<th>minage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>
Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rating</th>
<th>minage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>
Find age of the youngest sailor with age $\geq 18$, for each rating with at least 2 such sailors and where every sailor is under 60.

HAVING COUNT (*) > 1 AND EVERY (S.age <=60)

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rating</th>
<th>minage</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>

What is the result of changing EVERY to ANY?
Find age of the youngest sailor with age $\geq 18$, for each rating with at least 2 sailors between 18 and 60.

$$\text{SELECT } \text{S.rating, } \text{MIN}(\text{S.age}) \text{ AS minage} \text{ FROM Sailors S WHERE S.age} \geq 18 \text{ AND S.age} \leq 60 \text{ GROUP BY S.rating HAVING COUNT(*)} > 1$$

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</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>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>71</td>
<td>zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>74</td>
<td>horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>85</td>
<td>art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>bob</td>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>96</td>
<td>frodo</td>
<td>3</td>
<td>25.5</td>
</tr>
</tbody>
</table>

Answer relation:
For each red boat, find the number of reservations for this boat

SELECT  B.bid,  COUNT (*) AS scount
FROM   Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color='red'
GROUP BY  B.bid

- Grouping over a join of three relations.
- What do we get if we remove $B.color='red'$ from the WHERE clause and add a HAVING clause with this condition?
- What if we drop Sailors and the condition involving $S.sid$?
Find age of the youngest sailor with age > 18, for each rating with at least 2 sailors (of any age)

```
SELECT  S.rating, MIN (S.age)
FROM   Sailors S
WHERE  S.age >= 18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT (*)
               FROM   Sailors S2
               WHERE S.rating=S2.rating)
```

- Shows HAVING clause can also contain a subquery.
- Compare this with the query where we considered only ratings with 2 sailors over 18!
- What if HAVING clause is replaced by:
  - HAVING COUNT(*) > 1
Find those ratings for which the average age is the minimum over all ratings

- Aggregate operations cannot be nested! **WRONG:**
  
  \[
  \text{SELECT S.rating} \\
  \text{FROM Sailors S} \\
  \text{WHERE S.age} = (\text{SELECT MIN (AVG (S2.age)) FROM Sailors S2})
  \]

- Correct solution (in SQL/92):

  \[
  \text{SELECT Temp.rating, Temp.avgage} \\
  \text{FROM (SELECT S.rating, AVG (S.age) AS avgage FROM Sailors S} \\
  \text{GROUP BY S.rating) AS Temp} \\
  \text{WHERE Temp.avgage} = (\text{SELECT MIN (Temp.avgage) FROM Temp})
  \]
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.)
  - New operators (in particular, outer joins) possible/needed.
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 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.

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

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

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

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** (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:

```sql
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 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:

```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

- 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 many complications
- SQL allows specification of rich integrity constraints
- Triggers respond to changes in the database