SQL Aggregate Queries

Aggregate Operators

Significant extension of relational algebra

### Aggregate Operators

- `COUNT(*)`
- `COUNT(DISTINCT A)`
- `SUM(DISTINCT A)`
- `AVG(DISTINCT A)`
- `MAX(A)`
- `MIN(A)`

_A is a single column_

Result is _single_ value obtained by applying aggregate over all qualifying tuples

- `SELECT COUNT(*)`
  
  FROM Sailors S

Common Mistake with Aggregates

- Can’t have both aggregates and non-aggregates in SELECT
  - Reason: it is not guaranteed that there is only one tuple with the MAX value
  - Another way to look at it:
    - `_S.name_` has a value for each row
    - `MAX(S.age)`: one value over all rows, no per-row value defined
    - Select loops over rows (later; groups of rows with GROUP BY)

Illegal Query!

Aggregate Queries Examples

- `SELECT AVG(S.age)`
  
  FROM Sailors S

- `SELECT COUNT(DISTINCT S.rating)`
  
  FROM Sailors S

- `SELECT MIN(S.age)`
  
  FROM Sailors S

Grouping Results

- So far, aggregates applied to all (qualifying) tuples
  - We may want to apply them to each of several groups
  - **“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

- `SELECT MIN(S.age)`
  
  FROM Sailors S

- `SELECT MIN(S.age)`
  
  FROM Sailors S

- `SELECT MIN(S.age)`
  
  FROM Sailors S

- `SELECT MIN(S.age)`
  
  FROM Sailors S

- `SELECT MIN(S.age)`
  
  FROM Sailors S

How to achieve this?
GROUPBY Query Example

“Find the age of the youngest sailor for each rating level”

```sql
SELECT S.rating, MIN(S.age) AS minage
FROM Sailors S
GROUP BY S.rating
```

<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>
<tr>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>

GROUPBY Query Example, with join

“Find the number of reservations by sailors for each rating level”

```sql
SELECT S.rating, COUNT(*) AS n_res
FROM Sailors S, Reserves R
WHERE S.sid = R.sid
GROUP BY S.rating
```

<table>
<thead>
<tr>
<th>rating</th>
<th>n_res</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

GROUPBY Query Example, with join

“Find the average rating of sailors by boat they reserve (bid)”

```sql
SELECT bid, AVG(S.rating) AS rating
FROM Sailors S, Reserves R
WHERE S.sid = R.sid
GROUP BY R.bid
```

<table>
<thead>
<tr>
<th>bid</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>7.33</td>
</tr>
<tr>
<td>101</td>
<td>7.5</td>
</tr>
<tr>
<td>104</td>
<td>7.5</td>
</tr>
<tr>
<td>103</td>
<td>8.0</td>
</tr>
</tbody>
</table>

Queries With GROUP BY and HAVING

“Find the average rating of sailors by boat they reserve (bid)”

```sql
SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
GROUP BY grouping-list
HAVING group-qualification
```

- The target-list contains:
  (i) attribute names list (column names, possibly with corr. names)
  (ii) terms with aggregate operations (e.g., \( \text{MIN} (S\text{.age}) \))
- The attribute list (i) must be a subset of grouping-list
  - A group is a set of tuples that have the same value for all attributes in grouping-list
  - Each answer tuple corresponds to a group, so these attributes must have a single value per group.

GROUPBY Query Example

“Find the age of the youngest sailor with age at least 18, for each rating with at least 2 such sailors”

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

<table>
<thead>
<tr>
<th>sid</th>
<th>name</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>

Conceptual Evaluation

1. Compute cross-product of relation-list
2. Discard tuples that fail qualification of WHERE clause, ‘unnecessary’ fields are deleted
3. Remaining tuples are partitioned into groups by the value of attributes in grouping-list
4. Discard groups that fail group-qualification of HAVING clause

- Expressions in group-qualification must have a single value per group!
- An attribute in group-qualification that is not an argument of an aggregate operation in the group-qualification must appear in grouping-list (unless EVERY or ANY used)
5. Generate single answer tuple per qualifying group
GROUPBY Conceptual Evaluation Example

```plaintext
<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>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>
<tr>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
```

Find age of the youngest sailor with age \( \geq 18 \), for each rating with at least 2 sailors (of any age)

```plaintext
<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>
```

Find age of the youngest sailor with age \( \geq 18 \), for each rating with at least 2 sailors (of any age)

```plaintext
<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>
```

"Summary of cases" – INFORMAL!

- Can group validation condition be evaluated on "intermediate" relation alone?
  - If NO, then we need a subquery in HAVING, in order to process the original table rows as needed.
  - If YES, then we usually do not need a subquery:
    - Usual case: HAVING clause contains only aggregates and group-by attributes
    - Unusual case: the groups of rows have the data but it needs further processing using select to implement the desired filtering.

- Intermediate relation: result of from … where … groupby … part of query (groups of rows)

Pay attention to order of steps!

- HAVING executes AFTER WHERE

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

Wrong!!!

Example of unusual case

```
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) AND S.age > 25
```

Here the qualifying rows are in the groups, but we don’t have a way to reference them directly with select, so we go back to the original tables.
Example 1a

"Find the maximum age of students taking some course with 3 credits"

```
SELECT MAX (S.age)
FROM Students S, Enrolled E, Courses C
WHERE S.sid = E.sid AND E.cid = C.cid AND C.credits = 3
```

Example 1

"Find the average age of students taking some course with 3 credits"

```
SELECT AVG (S.age)
FROM Students S, Enrolled E, Courses C
WHERE S.sid = E.sid AND E.cid = C.cid AND C.credits = 3
```

Example 1b

"Find the average age of students taking some course with 3 credits"

Simple solution, but if same student takes multiple 3-credit courses, it may not be what we expect

```
SELECT AVG (S1.age)
FROM Students S1
WHERE S1.sid IN
(SELECT S.sid
FROM Students S, Enrolled E, Courses C
WHERE S.sid = E.sid AND E.cid = C.cid AND C.credits = 3)
```

Example 2

"Find the average age of enrolled students for each course with at least 10 enrolled students. List the course name(s) as well."

```
SELECT C.cname, AVG(S.age)
FROM Students S, Enrolled E, Courses C
WHERE S.sid = E.sid AND E.cid = C.cid AND COUNT(*)>=10
GROUP BY C.cid, C.cname
HAVING COUNT(*)>=10
```
More Group Qualification Functions

- So far, we have seen group qualification based on a property of the group
  - E.g., aggregate function computed for entire group
- But recent SQL standard version allow group qualification (HAVING) based on a property of individual records
  - `EVERY(condition)`: TRUE if condition holds for every group tuple
  - `ANY(condition)`: TRUE if condition holds for some group tuple
- Neither Oracle 12c (current version) nor `mysql` 5.6/5.7 support these constructs. I believe even DB2 can't do this.
- Thus we will ignore these HAVING clause extensions.

Recall previous Example

```
Find age of the youngest sailor with age 18, for each rating with at least 2 sailors (of any age) and every sailor under 60.
```

**Wrong by standard!!**

```
SELECT S.rating, MIN (S.age)  
FROM Sailors S  
WHERE S.age = (SELECT MIN (AVG (S2.age)) FROM Sailors S2)  
GROUP BY S.rating
```

**But this works on Oracle! (Extension of standard SQL)**

```
SELECT MIN (AVG (S.age)) AS avgage  
FROM Sailors S  
GROUP BY S.rating
```

Oracle use the AVG on groups and then the Min on those averages.
Aggregates and FROM Subqueries

- Aggregate operations cannot be nested!

  “Find rating that has lowest average sailor age”

  Text’s solution, p. 162: isn’t quite right either

  SELECT Temp.rating, Temp.avgage
  FROM (SELECT S.rating, AVG (S.age) AS avgage
           FROM Sailors S
           GROUP BY S.rating) Temp
  WHERE Temp.avgage = (SELECT MIN (Temp.avgage)
                          FROM Temp)

  • Fix: replace “FROM Temp” with “FROM (SELECT S.rating, AVG (… ) Temp, that is, write the first subquery out a second time. Noone says SQL is pretty!

  • Also, the text has FROM (...) AS Temp, but Oracle does not allow (subquery) as alias, only (subquery) alias in the FROM clause, in defiance of the standard

Example 3

Students (sid:integer, sname:string, age:integer)
Enrolled (sid:integer, cid:integer, grade:integer)
Courses (cid:integer, cname:string, credits:integer)

“Find the average age over enrolled students that are 25 or younger for each course with at least 10 enrolled students (of any age)”

SELECT C.cname, AVG(S.age)
FROM Students S, Enrolled E, Courses C
WHERE S.sid = E.sid AND E.cid = C.cid AND S.age <= 25
GROUP BY C.cid, C.cname
HAVING 10 <= (SELECT COUNT(*)
               FROM Enrolled E1
               WHERE E1.cid = C.cid)

Example 4

Students (sid:integer, sname:string, age:integer)
Enrolled (sid:integer, cid:integer, grade:integer)
Courses (cid:integer, cname:string, credits:integer)

“Find the average age of enrolled students for each course satisfying following two conditions: (1) course has at least 10 enrolled students and (2) average enrolled student age higher than 20”

SELECT C.cname, AVG(S.age)
FROM Students S, Enrolled E, Courses C
WHERE S.sid = E.sid AND E.cid = C.cid
GROUP BY C.cid, C.cname
HAVING AVG(S.age) > 20 AND COUNT(*) >= 10

Example 5

Students (sid:integer, sname:string, age:integer)
Enrolled (sid:integer, cid:integer, grade:integer)
Courses (cid:integer, cname:string, credits:integer)

“Find the average age of enrolled students for each course satisfying following two conditions: (1) course has at least 10 enrolled students and (2) at least one enrolled student has age higher than 20”

SELECT C.cname, AVG(S.age)
FROM Students S, Enrolled E, Courses C
WHERE S.sid = E.sid AND E.cid = C.cid
GROUP BY C.cid, C.cname
HAVING AVG(S.age) > 20 AND COUNT(*) >= 1
Example 5

**Enrolled (sid:integer, cid:integer, grade:integer)**

**Courses (cid:integer, cname:string, credits:integer)**

"Find the average age of enrolled students for each course satisfying following two conditions: (1) course has at least 10 enrolled students and (2) at least one enrolled student has age higher than 20"

```
SELECT C.cname, AVG(S.age) FROM Students S, Enrolled E, Courses C WHERE S.sid = E.sid AND E.cid = C.cid GROUP BY C.cid, C.cname HAVING MAX(S.age) > 20 AND COUNT(*) >= 10
```

---

Example 6

**Enrolled (sid:integer, cid:integer, grade:integer)**

**Courses (cid:integer, cname:string, credits:integer)**

"Find the courses that have the highest average age computed among students 25 or younger (i.e., highest among all courses) and also have at least 10 students of any age enrolled. Output the course name and the above-mentioned average age value."

This requires two stages. We can find average age for each course among students 25 or younger in courses having at least 10 students of any age, call that T:

```
SELECT C.cname, AVG(S.age) AS avg_age FROM Students S, Enrolled E, Courses C WHERE S.sid = E.sid AND E.cid = C.cid AND S.age <= 25 GROUP BY C.cid, C.cname HAVING 10 <= (SELECT COUNT(*) FROM Enrolled E1 WHERE E1.cid = C.cid)
```

Then:

```
SELECT C.cname, AVG(S.age) AS avg_age FROM Students S, Enrolled E, Courses C WHERE S.sid = E.sid AND E.cid = C.cid AND S.age <= 25 GROUP BY C.cid, C.cname HAVING 10 <= (SELECT COUNT(*) FROM Enrolled E1 WHERE E1.cid = C.cid)
```

```
<table>
<thead>
<tr>
<th>Cname</th>
<th>Avg_age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calc 1</td>
<td>18</td>
</tr>
<tr>
<td>Philo 1</td>
<td>22</td>
</tr>
</tbody>
</table>
```

All in one query by using a subquery in the FROM and the same one in the WHERE clause:

```
SELECT Temp.cname, Temp.avg_age FROM (SELECT C.cname, AVG(S.age) AS avg_age FROM Students S, Enrolled E, Courses C WHERE S.sid = E.sid AND E.cid = C.cid AND S.age <= 25 GROUP BY C.cid, C.cname HAVING 10 <= (SELECT COUNT(*) FROM Enrolled E1 WHERE E1.cid = C.cid)) Temp WHERE Temp.avg_age = (SELECT MAX(Temp1.avg_age) FROM (SELECT C.cname, AVG(S.age) AS avg_age FROM Students S, Enrolled E, Courses C WHERE S.sid = E.sid AND E.cid = C.cid AND S.age <= 25 GROUP BY C.cid, C.cname HAVING 10 <= (SELECT COUNT(*) FROM Enrolled E1 WHERE E1.cid = C.cid)) Temp1)
```