CS 133: Databases

Fall 2019
Lec 9 – 10/03
SQL II

Prof. Beth Trushkowsky

Plan for Today

- Continue to develop strategies for thinking about conceptual SQL query evaluation
- Build more expressive SQL queries using aggregates and JOINs
- Gain foundation for tackling aggregation and HeapFile mutability in Lab 2

Aggregate Operators

- Extension of relational algebra!

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

SELECT AVG (S.age)
FROM Sailors S
WHERE S.rating = 10

SELECT COUNT (DISTINCT R.day)
FROM Reserves R
WHERE R.sid = 42;
```

GROUP BY: Intuition

- Consider the query:

```sql
SELECT MIN (S.age)
FROM Sailors S
WHERE S.rating = 5;
```

- What if we want the age of the youngest sailor for each rating level?
  - If we knew that all possible rating values range 1 to 10; we could write 10 queries that look like this:

```sql
For i = 1, 2, ..., 10:
SELECT MIN (S.age)
FROM Sailors S
WHERE S.rating = i;
```

- In general, we don’t know how many rating levels exist, and what the rating values for these levels are!

Aggregates of a field consider only non-NULL values!

Aggregate without a GROUP BY clause?
The whole relation is one group!
GROUP BY

• To get youngest age per rating group:

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

Output relation schema: rating, minAge

• How many tuples do we expect in the output relation?

• What will be true of the rating field for each tuple in a particular group?

Queries With GROUP BY

• To generate values for a field based on groups of tuples, use aggregate functions in SELECT statements along with the GROUP BY clause.

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

WHERE clause is evaluated before GROUP BY

The target-list can contain:
(i) terms with aggregate operations
(ii) list of column names

*Note:* list of column names (ii) can contain only attributes from the grouping-list.

Query: Find name and age of oldest sailors

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

What’s wrong with this query?

- Cannot use fields from other relations in FROM

Third query equivalent to second query

- in SQL 92 standard, but not supported in some systems.

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

Nested Queries in FROM Clause

• Supported by most database systems
  – Cannot use fields from other relations in FROM

• Name and age of oldest Sailors?

```sql
SELECT S.sname, S.age
FROM Sailors S,
(SELECT MAX(age) AS oldest
FROM Sailors)
AS MaxAgeRelation
WHERE S.age = MaxAgeRelation.oldest;
```
More Examples: Group By

For each bid, find the number of reservations that have not been reserved by sid 42

```sql
SELECT R.bid, COUNT(*)
FROM Reserves R
WHERE R.sid <> 42
GROUP BY R.bid;
```

Exercise (2-4)

```sql
SELECT S.rating, MIN(S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating;
```

```sql
SELECT B.bid, COUNT(*) AS boatCount
FROM Boats B, Reserves R
WHERE R.bid = B.bid
AND B.color = 'red'
GROUP BY B.bid;
```

```sql
SELECT R.bid, R.day, COUNT(*) AS reserveCount
FROM Reserves R
GROUP BY R.bid, R.day;
```

Aggregates: Iterator Perspective

SELECT MAX(age) FROM Sailors;

<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>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>71</td>
<td>zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Aggregator call's `next()` on its child Operator to get Tuples

SimpleDb Aggregate

```java
public class Aggregate extends Operator {
    // ...
    public Aggregate(DbIterator child, int ofield, int gfield, Aggregator.Op aop) {
        // ...
    }
}
```

```java
public class IntegerAggregator implements Aggregator {
    // ...
    public IntegerAggregator(int gbfield, Type gbfieldtype, int ofield, Op what) {
        // ...
    }
    // ...
    public void mergeTupleIntoGroup(Tuple tup) {
        // ...
    }
}
```
Lab 2: HeapFile Mutability

- Insert (you’ll do the actual Insert operator later)
  - Will call insertTuple() in BufferPool
    - Which calls insertTuple() in HeapFile
      - Which calls insertTuple() in HeapPage

- Will be adding more to HeapFile and HeapPage
  - Have to find a page to put the tuple (how to tell?)
  - When inserting, if no pages have room, may need a new
    page — HeapPage.createEmptyPageData() will be useful

  - Pages will get dirty!
  - BufferPool will set a page as dirty or not-dirty

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

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

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33.0</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>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

GROUP BY and HAVING

- Use the HAVING clause with the GROUP BY clause
  to restrict which group-rows are returned in the result set

- Conceptual evaluation (after evaluating WHERE clause)
  - Form groups according to grouping-list
  - Then group-qualification is applied to eliminate some groups.

Expressions in group-qualification must have a single value per group!
→ Fields in group-qualification must either (1) appear in grouping-list or
  (2) be part of an aggregation

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

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

How could you change this query to instead include only the rating
  groups that have rating greater than the average rating for all
  sailors?

One possibility is to change the HAVING clause to:

```
HAVING S.rating >=
    (SELECT AVG(rating) from Sailors)
```

Answer
JOIN Variety

```
SELECT (column_list)
FROM  table_name1
[INNER | NATURAL | {LEFT | RIGHT | FULL } OUTER] JOIN
    table_name2
ON qualification_list
```

Inner and Natural Join

Only the rows that match the search conditions are returned.

```
SELECT S.sid, S.sname, R.bid
FROM Sailors S INNER JOIN Reserves R
ON S.sid = R.sid;
```

Returns only those sailors who have reserved boats

SQL92 also allows:

```
SELECT s.sid, s.sname, r.bid
FROM Sailors s NATURAL JOIN Reserves r
```

“NATURAL JOIN” is an equi-join for each pair of attributes with the same name, removing duplicate columns

Left Outer Join

Left Outer Join returns all matched rows, and also all unmatched rows from the table on the left of the join clause
(uses NULLs in fields of non-matching tuples)

```
SELECT s.sid, s.sname, r.bid
FROM Sailors s LEFT OUTER JOIN Reserves r
ON s.sid = r.sid
```

Returns all sailors & information on whether they have reserved boats

<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>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</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>10/10/96</td>
</tr>
<tr>
<td>95</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>12/3/97</td>
</tr>
</tbody>
</table>
Exercise 5: JOINs

SELECT S.sid, S.sname, count(R.bid) FROM Sailors S LEFT OUTER JOIN Reserves R ON S.sid = R.sid GROUP BY S.sid, S.sname;

Full Outer Join

Returns all (matched or unmatched) rows from both the tables.

Returns all (matched or unmatched) rows from both the tables.

SELECT r.sid, s.sname, b.bid, b.bname FROM Sailors s FULL OUTER JOIN Boats b ON s.sname = b.bname

<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>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</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>105</td>
<td>Lubber</td>
<td>purple</td>
</tr>
</tbody>
</table>

Types of JOINs

Choices:

- INNER JOIN
- NATURAL JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

Src: https://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins