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!

```
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:

  ```
  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:

  ```
  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!
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

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

- Name and age of oldest Sailors?

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

Third query equivalent to second query →

- in SQL 92 standard, but not supported in some systems.
More Examples: Group By

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

\[
\text{SELECT } R\text{.bid, COUNT(*)}
\text{FROM Reserves } R
\text{WHERE } R\text{.sid }\neq 42
\text{GROUP BY } R\text{.bid;}
\]

Try Exercise (2-4)

\[
\text{SELECT } S\text{.rating, MIN (S\text{.age})}
\text{FROM Sailors } S
\text{WHERE } S\text{.age }\geq 18
\text{GROUP BY } S\text{.rating;}
\]

\[
\text{SELECT } B\text{.bid, COUNT(*) AS boatCount}
\text{FROM Boats } B, \text{ Reserves } R
\text{WHERE } R\text{.bid }= B\text{.bid}
\text{AND } B\text{.color}=\text{'red'}
\text{GROUP BY } B\text{.bid;}
\]

\[
\text{SELECT } R\text{.bid, R\text{.day, COUNT(*) AS reserveCount}
\text{FROM Reserves } R
\text{GROUP BY } R\text{.bid, R\text{.day;}
}\]

Aggregates: Iterator Perspective

\[
\text{SELECT MAX(age)}
\text{FROM Sailors;}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{sid} & \text{sn} & \text{rating} & \text{age} \\
\hline
22 & \text{dust} & 7 & 45.0 \\
31 & \text{lubber} & 8 & 55.5 \\
71 & \text{zorba} & 10 & 16.0 \\
64 & \text{horatio} & 7 & 35.0 \\
29 & \text{brutus} & 1 & 33.0 \\
58 & \text{rusty} & 10 & 35.0 \\
\hline
\end{array}
\]

SimpleDb Aggregate

```java
public class Aggregate extends Operator {
    public Aggregate(ObIterator child, int ofield, int gfield, Aggregator.Op Op)
    The index of the field you'll be aggregating. Determine its Type to decide whether to create an IntegerAggregator or a StringAggregator.
    The aggregation function, e.g., MAX.
    Which field you'll be grouping on, or Aggregator.NO_GROUPING.
}
```

SimpleDb Aggregator

```java
public class IntegerAggregator implements Aggregator {
    public IntegerAggregator(int gbfield, Type gbfieldtype, int ofield, Op Op)
    Which field you'll be grouping on, or Aggregator.NO_GROUPING.
    The type of the group by field.
    The aggregate function.
    Which field is being aggregated.
}
```
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

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 ≥ 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>

Query: Find the age of the youngest sailor with age ≥ 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 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
Types of JOINs

`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.

```sql
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:

```sql
SELECT s.sid, s.sname, r.bid
FROM Sailors s NATURAL JOIN Reserves r
WHERE s.sid = r.sid;
```

“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)

```sql
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

### Left Outer Join

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

<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>

### Full Outer Join

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>sname</th>
<th>rating</th>
<th>age</th>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>105</td>
<td>Lubber</td>
<td>purple</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
<td>null</td>
<td>null</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>bid</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>105</td>
<td>Lubber</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>101</td>
<td>Interlake</td>
</tr>
</tbody>
</table>