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

Thinking About Writing SQL

- What do I want?
  - Output relation schema?
  - What does each tuple in output relation mean?
- How do I get what I want?
  - Which relations will I need to consult?
  - How should they be combined?
    - For fields that are joined... what will that look like?
    - Are duplicates possible? What do they mean?

Administrivia

- Lab 2:
  - Ex 1 due tomorrow
  - Ex 2-3 out tomorrow
- Looking ahead:
  - Next topic: Query evaluation
  - Midterm goes out end of next week (2/24)
  - Midterm due 2/28, no class that day
- Modified grutoring schedule for this week and next
  - See Piazza
Aggregate Operators

- Extension of relational algebra!

| COUNT(*) |
| COUNT([DISTINCT]A) |
| SUM([DISTINCT]A) |
| AVG([DISTINCT]A) |
| MAX(A) |
| MIN(A) |

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

```
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: Motivation

- 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 all possible rating values go from 1 to 10; we could write 10 queries that look like this:
    ```
    For \(i = 1, 2, \ldots, 10\):
    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:

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

- How many tuples do we expect in the output relation?
- What is 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

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

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

column name list (ii) can contain only attributes from the grouping-list.
Query: Find name and age of oldest sailors

- The first query is incorrect!

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

- Third query equivalent to second query
  – allowed in SQL 92 standard, but not supported in some systems.

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;

Conceptual Evaluation

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

1. The cross-product of relation-list is computed
2. Tuples that fail qualification are discarded
3. The remaining tuples are partitioned into groups by the value of attributes in grouping-list.
4. One answer tuple is generated per group!

Nested Queries in FROM Clause

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

Name and age of oldest Sailors?

SELECT S.sname, S.age
FROM Sailors S,
     (SELECT MAX(age) AS oldest
      FROM Sailors)
AS OldAge
WHERE S.age = OldAge.oldest;

More Examples: Group By

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

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

Try Exercise (2-4)

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

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

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;

Aggregate: MAX

<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>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>

After processing first four tuples, MaxSoFar is 55.5

Lab 2: Aggregate

- Aggregate is an Operator
  - Input: tuples from its Child DbIterator
  - Output tuples’ schema:
    - (group, result)
    - (result)
      - if group == NO_GROUPING
  - Need to process all input before producing any output
  - Think about the Field types in the output tuples you produce!

- Aggregator Interface
  - Helper class used by Aggregate to handle the running stats
  - Implement both IntegerAggregator and StringAggregator

SimpleDb Aggregate

```java
public class Aggregate extends Operator {
    public Aggregate(DbIterator 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 what)
```

Which field you’ll be grouping on, or `Aggregator.NO_GROUPING`

The type of the group by field

The aggregate function

```java
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

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

- Form groups as before
- The group-qualification is then applied to eliminate some groups.
  - Expressions in group-qualification must have a single value per group!
  - That is, fields in group-qualification must be arguments of an aggregate operator or must also appear in the grouping-list.

  (SQL does not [necessarily] exploit primary key semantics here!)

- One answer tuple is generated per qualifying group

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

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

### Types of JOINs

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

**Choices:**
- INNER JOIN
- NATURAL JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

---

**Inner 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

*Same as:*

```
SELECT S.sid, S.sname, R.bid
FROM Sailors S, Reserves R
WHERE S.sid = R.sid;
```

As with Relational Algebra, “NATURAL” means equi-join for each pair of attributes with the same name

---

**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
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>

```
sid  bid  day
22    101  10/10/96
95    103  11/12/96
22    102  12/3/97
```

Right Outer Join

Right Outer Join returns all matched rows, plus all unmatched rows from the table on the right of the join clause.

```
SELECT r.sid, b.bid, b.bname
FROM Reserves r RIGHT OUTER JOIN Boats b
ON r.bid = b.bid
```

```
<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>102</td>
<td></td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>95</td>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>104</td>
<td></td>
<td>Marine</td>
<td>red</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;
```

```
sid  sname  bid
22    Dusty  101
22    Dusty  102
31    Lubber null
95    Bob    103
```

```sql
SELECT sid, bname
FROM Boats
```
Full Outer Join

Full Outer Join returns all (matched or unmatched) rows from the tables on both sides of the join clause.

```
SELECT r.sid, 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>
</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>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>
```

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