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!

```
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;
```

COUNT (*)

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

single attribute

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

Aggregates of a field consider only *non-NULL* values!

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:

```
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

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

WHERE clause is evaluated *before* GROUP BY

Can have multiple fields in 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

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

What's wrong with this query?

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

Third query equivalent to second query →

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

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

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 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

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

Exercise (2-4)

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

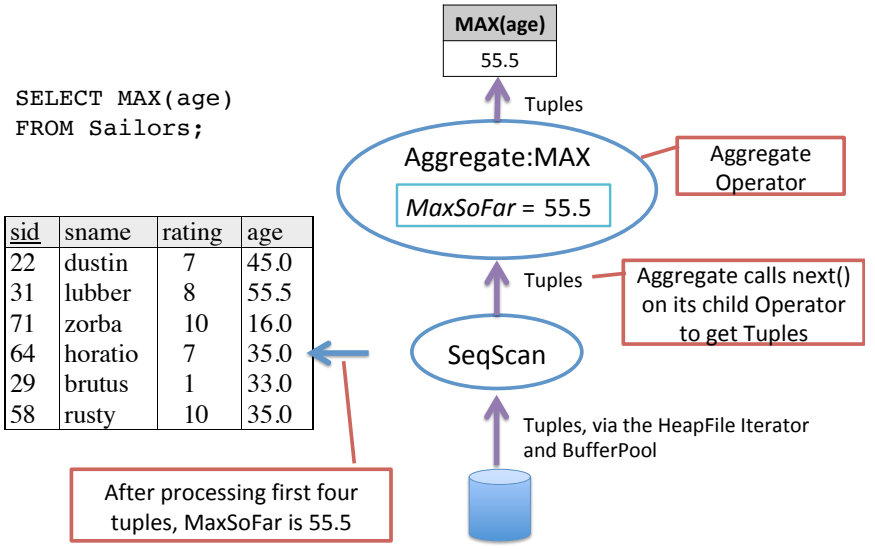
```
SELECT B.bid, COUNT(*) AS boatCount
FROM Boats B, Reserves R
WHERE  R.bid = B.bid
       AND B.color='red'
GROUP BY  B.bid;
```

```
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;
```

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0



SimpleDb Aggregate

```
public class Aggregate extends Operator {
  public Aggregate(DbIterator child, int afield, int gfield, Aggregator.Op aop)
```

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

```
public class IntegerAggregator implements Aggregator {
  public IntegerAggregator(int gbfield, Type gbfieldtype, int afield, Op what)
```

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

The type of the group by field

Which field is being aggregated

The aggregate function

```
/**
 * Merge a new tuple into the aggregate, grouping as indicated in the
 * constructor
 *
 * @param tup
 *           the Tuple containing an aggregate field and a group-by field
 */
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

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

- 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
```

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

rating	age	count
1	{33.0}	1
7	{35.0, 45.0}	2
8	{55.0}	1
10	{35.0}	1

rating	MIN(age)
7	35.0

Answer

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 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)
```

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

Same as:
SELECT S.sid, S.sname, R.bid
FROM Sailors S, Reserves R
WHERE S.sid = R.sid;

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

Left Outer Join

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

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

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

sid	sname	bid
22	Dustin	101
22	Dustin	102
31	Lubber	*null*
95	Bob	103

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

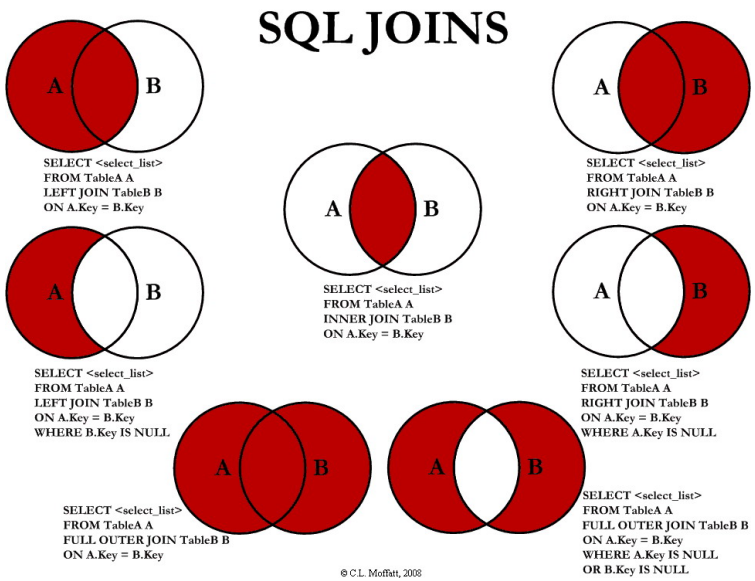
Returns all (matched or unmatched) rows from both the tables.

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

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

bid	bname	color
101	Interlake	blue
105	Lubber	purple

sid	sname	bid	bname
22	Dustin	*null*	*null*
31	**Lubber**	**105**	**Lubber**
95	Bob	*null*	*null*
null	*null*	101	Interlake



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