SQL

Lecture 11

Based on slides by R. Ramakrishnan and J. Gehrke

CS 638 Web Programming

Structured Query Language

- Developed by IBM (system R) in the 1970s
- Used by all major vendors of relational databases
- SQL has been standardized by ISO and ANSI
- SQL has many components
  - Data Description Language is used for creating databases (creating tables, specifying integrity constraints) – not covered
  - Data Manipulation Language is used for querying database, inserting, deleting, modifying rows – subject of today's lecture
  - Other components not covered in this class: triggers, transactions, stored procedures

Basic SQL query

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

- *relation-list* A list of relation names (possibly with a *range-variable* after each name).
- *target-list* A list of attributes of relations in *relation-list*
- *qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of $<, >, =, \leq, \geq, \neq$) combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are *not* eliminated!

Conceptual evaluation strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list* (see example).
  - Discard resulting tuples if they fail *qualifications.*
  - Delete attributes that are not in *target-list.*
  - If DISTINCT is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

Conceptual evaluation ex.

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Other query examples

- Find all sailors who have reserved at least one boat
  - Why doesn't this query find all sailors?

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

- Find all sailors who have reserved a red or a green boat

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
  AND (B.color='red' OR B.color='green')
```

- Find all sailors who have reserved a red and a green boat

```
SELECT  S.sid
FROM  Sailors S, Boats B1, Reserves R1,
          Boats B2, Reserves R2
WHERE  S.sid=R1.sid AND R1.bid=B1.bid
  AND  S.sid=R2.sid AND R2.bid=B2.bid
  AND (B1.color='red' AND B2.color='green')
```

Ordering the results

```
SELECT     [DISTINCT] target-list
FROM       relation-list
WHERE      qualification
ORDER BY  column1 [ASC, DESC] [, column2 [ASC, DESC]] …
```

- Allows user to control ordering of tuples in result
- Can specify multiple columns to order the results by
  - Second column used as tie breaker when values in the first column are equal
- By default rows ordered in ascending order, but can request descending order using DESC

ORDER BY examples

- List the names and ages of all sailors, sorted by name

```
SELECT  S.sname,  S.age
FROM  Sailors S
ORDER BY  S.sname
```

- List the names and ratings of all sailors with a rating larger than 5 ordered by rating and within each rating by the age of the sailors

```
SELECT  S.sname, S.rating
FROM  Sailors S
WHERE  S.rating > 5
ORDER BY S.rating, S.age
```

Expressions and strings

```
SELECT  S.age, age1=S.age-5, 2*S.age AS age2
FROM  Sailors S
WHERE  S.sname LIKE ‘B_%B’
```

- Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with B and contain at least three characters.
  - Uses arithmetic expressions and string pattern matching
- AS and = are two ways to name fields in result.
- LIKE is used for string matching. `_’ stands for any one character and `%’ stands for 0 or more arbitrary characters.

Nested queries

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid
                  FROM  Reserves R
                  WHERE  R.bid=103)
```

- Find the names of sailors who have reserved boat 103
  - There are multiple ways to formulate every query
- A very powerful feature of SQL: a WHERE clause can itself contain a SQL query! (So can FROM and HAVING clauses.)
- To find sailors who’ve *not* reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a <u>*nested loops*</u> evaluation: *For each Sailors tuple, check the qualification by computing the subquery.*

Nested queries w/ correlation

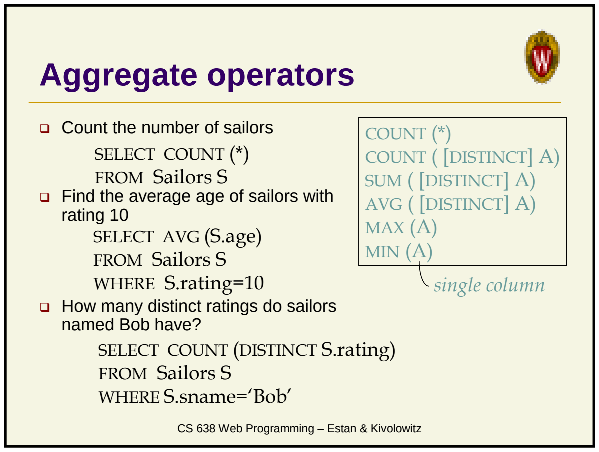
```
SELECT  S.sname
FROM  Sailors S
WHERE   EXISTS  (SELECT  *
                 FROM  Reserves R
                 WHERE  R.bid=103 AND S.sid=R.sid)
```

- EXISTS is another set comparison operator, like IN.
- If UNIQUE is used, and * is replaced by *R.bid*, finds sailors with at most one reservation for boat #103. (UNIQUE checks for duplicate tuples; * denotes all attributes. Why do we have to replace * by *R.bid*?)
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple.

More set comparison operators

- We’ve already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL, *op* IN   >,<,=,≥,≤,≠
- Find sailors whose rating is greater than that of some sailor called Horatio:

```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ANY  (SELECT  S2.rating
                        FROM  Sailors S2
                        WHERE S2.sname=‘Horatio’)
```



Aggregate operators

- Count the number of sailors

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

- Find the average age of sailors with rating 10

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

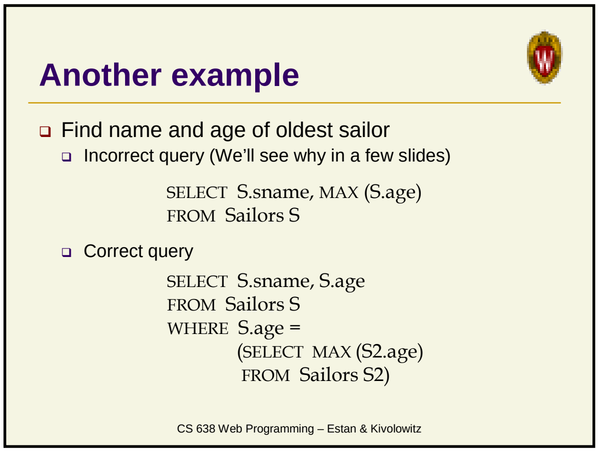
- How many distinct ratings do sailors named Bob have?

```
SELECT COUNT (DISTINCT S.rating)
FROM Sailors S
WHERE S.sname='Bob'
```

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

single column

CS 638 Web Programming – Estan & Kivolowitz



Another example

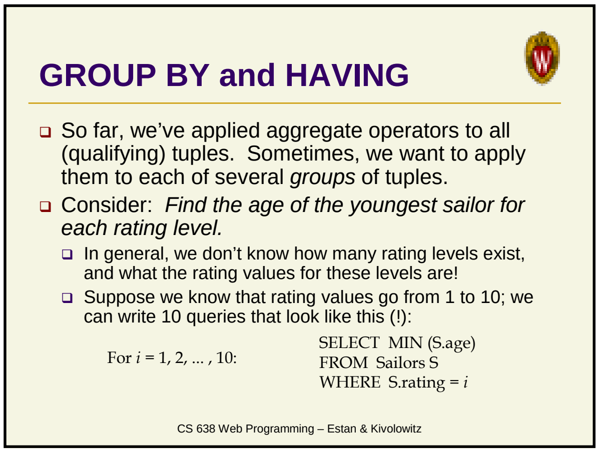
- Find name and age of oldest sailor
  - Incorrect query (We'll see why in a few slides)

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

  - Correct query

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

CS 638 Web Programming – Estan & Kivolowitz



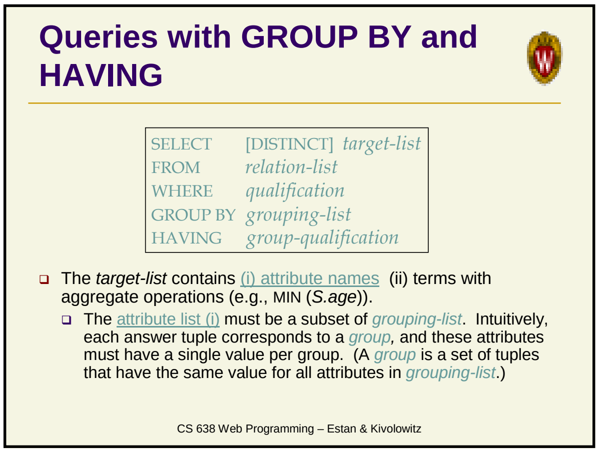
GROUP BY and HAVING

- So far, we've applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several *groups* of tuples.
- Consider: *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; we can write 10 queries that look like this (!):

For *i* = 1, 2, ... , 10:

```
SELECT MIN (S.age)
FROM Sailors S
WHERE S.rating = i
```

CS 638 Web Programming – Estan & Kivolowitz

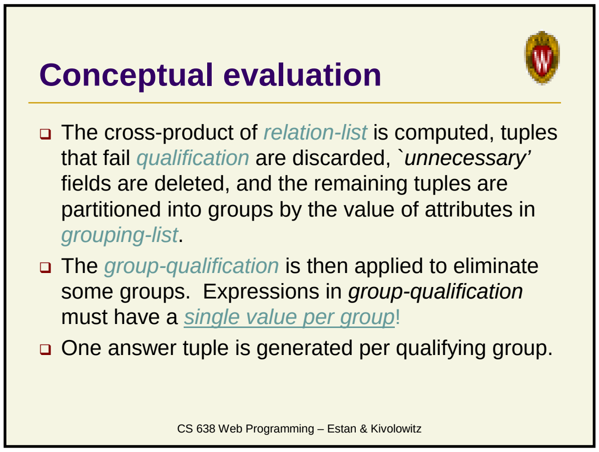


Queries with GROUP BY and HAVING

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

- The *target-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
  - The attribute list (i) must be a subset of *grouping-list*. Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group. (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

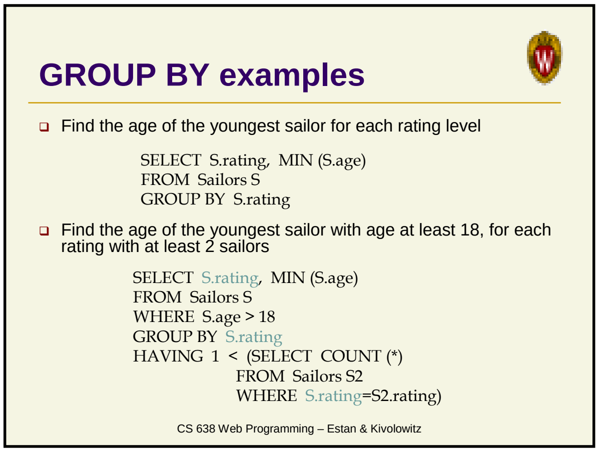
CS 638 Web Programming – Estan & Kivolowitz



Conceptual evaluation

- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, `*unnecessary'* fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The *group-qualification* is then applied to eliminate some groups. Expressions in *group-qualification* must have a *single value per group*!
- One answer tuple is generated per qualifying group.

CS 638 Web Programming – Estan & Kivolowitz



GROUP BY examples

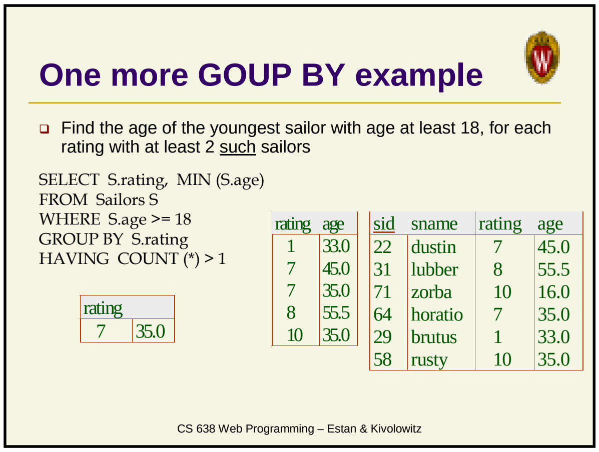
- Find the age of the youngest sailor for each rating level

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

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

```
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)
```

CS 638 Web Programming – Estan & Kivolowitz



One more GOUP BY example

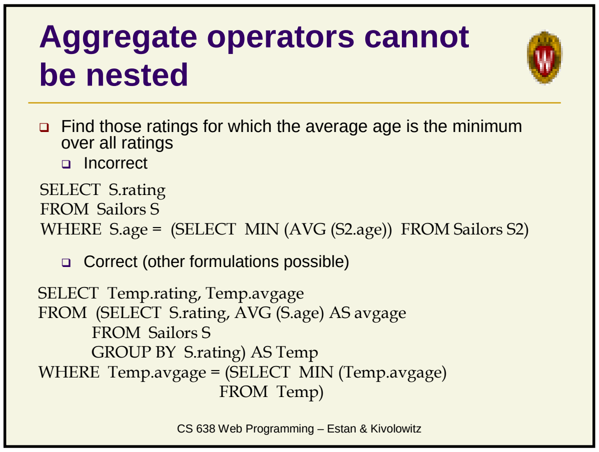
- Find the age of the youngest sailor with age at least 18, for each rating with at least 2 such sailors

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

rating	
7	35.0

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

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



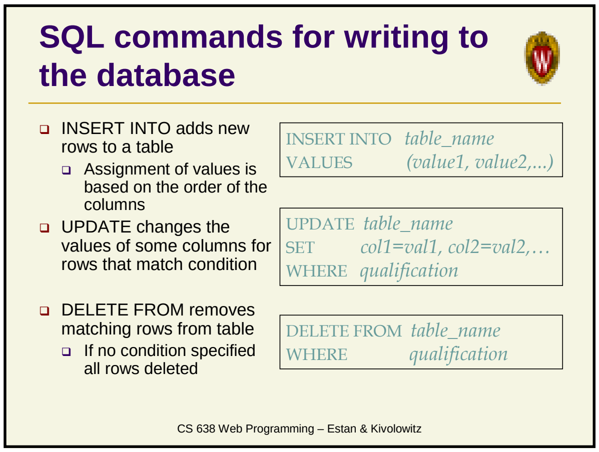
Aggregate operators cannot be nested

- Find those ratings for which the average age is the minimum over all ratings
  - Incorrect

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

  - Correct (other formulations possible)

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



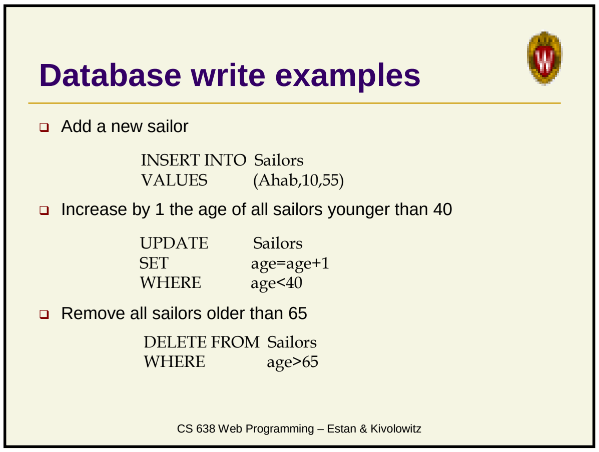
SQL commands for writing to the database

- INSERT INTO adds new rows to a table
  - Assignment of values is based on the order of the columns
- UPDATE changes the values of some columns for rows that match condition
- DELETE FROM removes matching rows from table
  - If no condition specified all rows deleted

```
INSERT INTO   table_name
VALUES          (value1, value2,...)
```

```
UPDATE  table_name
SET         col1=val1, col2=val2,…
WHERE   qualification
```

```
DELETE FROM  table_name
WHERE            qualification
```



Database write examples

- Add a new sailor

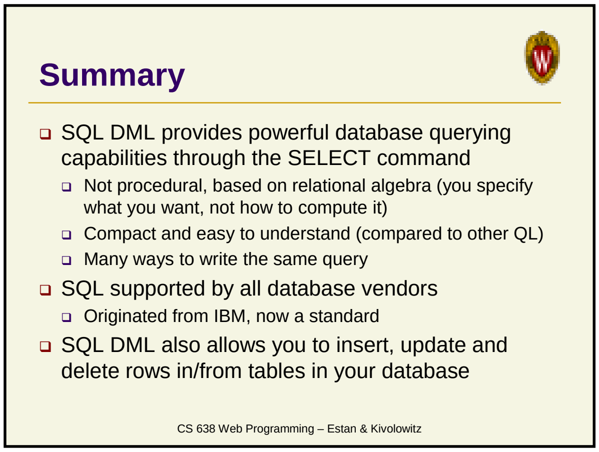
```
INSERT INTO  Sailors
VALUES          (Ahab,10,55)
```

- Increase by 1 the age of all sailors younger than 40

```
UPDATE        Sailors
SET              age=age+1
WHERE         age<40
```

- Remove all sailors older than 65

```
DELETE FROM  Sailors
WHERE             age>65
```



Summary

- SQL DML provides powerful database querying capabilities through the SELECT command
  - Not procedural, based on relational algebra (you specify what you want, not how to compute it)
  - Compact and easy to understand (compared to other QL)
  - Many ways to write the same query
- SQL supported by all database vendors
  - Originated from IBM, now a standard
- SQL DML also allows you to insert, update and delete rows in/from tables in your database