SQL

Instructor: Wang-Chiew Tan

Reference:
A First Course in Database Systems,
3rd edition, Chapter 6.4.3-6.4.7

Aggregates

- SQL has 5 aggregation operators: SUM, AVG, MIN, MAX, COUNT.
- Aggregation operators are applied on scalar value expressions. E.g., a scalar attribute such as salary or 1.1*salary.
  - An exception: COUNT(*) which counts the number of tuples.

- Used for computing summary results over a table. E.g.,
  - find the average/min/max score of all students who took CMPS180
  - find the total number of movies released in 2014.
  - find total salary of employees in Sales department.

Aggregates (cont'd)

- Aggregate operators are specified in the SELECT clause.
- Suppose A is a column in a table.
  - COUNT([DISTINCT] A)
    - Returns the number of [unique] values in the A column
  - SUM([DISTINCT] A)
    - Returns the sum of all [unique] values in the A column
  - AVG([DISTINCT] A)
    - Returns the average of all [unique] values in the A column
  - MAX(A)/MIN(A)
    - Returns the maximum value or minimum value in the A column.

Example

- MovieExec(name, address, cert#, netWorth).

  ```
  SELECT AVG(netWorth)
  FROM MovieExec;
  ```

- Finds the average of "netWorth" values of tuples in the relation MovieExec.

MovieExec

name	address	cert#	netWorth
S. Speilberg	X	38120	3000000
G. Lucas	Y	43918	4000000
W. Disney	Z	65271	5000000

Example (cont'd)

- MovieExec(name, address, cert#, netWorth).

```
SELECT AVG(netWorth)
FROM MovieExec;
```

```
SELECT AVG(DISTINCT netWorth)
FROM MovieExec;
```

- Finds the average of "netWorth" values of tuples in the relation MovieExec.

MovieExec

name	address	cert#	netWorth
S. Speilberg	X	38120	3000000
G. Lucas	Y	43918	4000000
W. Disney	Z	65271	3000000

More examples

```
SELECT COUNT(*)
FROM StarsIn;
```

```
SELECT COUNT(starName)
FROM StarsIn;
```

```
SELECT COUNT(DISTINCT starName)
FROM StarsIn;
```

StarsIn(movieTitle, movieYear, starName)

Example

- Movies(title, year, length, genre, studioName, producerC#)

```
SELECT studioName, SUM(length)
FROM Movies
GROUP BY studioName;
```

- Find the sum of lengths of all movies from each studio.

Movies

…	studioName	length
…	Dreamworks	120
…	Dreamworks	162
…	Fox	152
…	Universal	230
…	Fox	120

Aggregates and Grouping

- GROUP BY clause that follows the WHERE clause.

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$ AGGOP(…)
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]
[GROUP BY <list of grouping attributes>]
[ORDER BY <list of attributes>] [DESC]

If SELECT clause has aggregates, then $c_1, c_2, \ldots, c_m$ must come from the list of grouping attributes.

- Let Result denote an empty collection.
- For every tuple t_1 from R_1, t_2 from R_2, …, t_n from R_n
  - if $t_1, \ldots, t_n$ satisfy *condition (*i.e., condition evaluates to true), then add the resulting tuple that consists of $c_1, c_2, \ldots, c_m$ components (including attributes of AGGOP operators) of t_i into Result.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- Group tuples in Result according to list of grouping attributes. If GROUP BY is omitted, the entire table is regarded as ONE group.
- If ORDER BY <list of attributes> exists, order the tuples in Result according to ORDER BY clause.
- Apply aggregate operator on tuples of each group.
- Return the final Result.

More examples - Grouping and Aggregates

```
SELECT studioName
FROM Movies
GROUP BY studioName;
```

```
SELECT DISTINCT studioName
FROM Movies;
```

- It is possible to write GROUP BY without aggregates (and aggregates without GROUP BY). See earlier slides.
- The two queries above are equivalent.

Movies(title, year, length, genre, studioName, producerC#)
MovieExec(name, address, cert#, netWorth)

```
SELECT name, SUM(length)
FROM MovieExec, Movies
WHERE producerC# = cert#
GROUP BY name;
```

A	B	C	D
a1	b1	1	7
a1	b1	2	8
a2	b1	3	9
a3	b1	4	10
a2	b1	5	11
a1	b1	6	12

```
SELECT      A, SUM(C), MAX(D)
FROM        R
GROUP BY    A, B
```

Grouping, Aggregation, and Nulls

- NULLs are ignored in any aggregation.
  - It does not contribute to the SUM, AVG, COUNT, MIN, MAX of an attribute.
  - If the result is an empty bag, then SUM, AVG, MIN, MAX on the empty bag is NULL. COUNT of an empty bag is 0.
  - COUNT(*) = number of tuples in a relation.
  - COUNT(A) is the number of tuples with non-null values for attribute A.

- GROUP BY does not ignore NULLs.
  - The groups that are formed with a GROUP BY on attributes A_1, …, A_k may have one or more NULLs on these attributes.

Examples

- Suppose R(A,B) is a relation with a single tuple (NULL, NULL).

```
SELECT A, COUNT(B)
FROM R
GROUP BY A;


SELECT A, SUM(B)
FROM R
GROUP BY A;
```

HAVING clause

SELECT [DISTINCT] c_1, c_2, …, c_m AGGOP(…)
FROM R_1, R_2, …, R_n
[WHERE *condition*]
[GROUP BY <list of grouping attributes>
[HAVING *condition*]]
[ORDER BY <list of attributes>] [DESC]

Note that HAVING clause cannot exists by itself.

- Choose groups based on some aggregate property of the group itself.

- Let Result denote an empty collection.
- For every tuple t_1 from R_1, t_2 from R_2, …, t_n from R_n
  - if t_1, …, t_n satisfy *condition (*i.e., condition evaluates to true), then add the resulting tuple that consists of c_1, c_2, …, c_m (including attributes in AGGOP operators) components of t_i into Result.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- Group tuples in Result according to list of grouping attributes. If GROUP BY is omitted, the entire table is regarded as ONE group.
- Apply aggregate operator on tuples of each group.
- Apply condition of HAVING clause to each group. Remove groups that do not satisfy the HAVING clause.
- If ORDER BY <list of attributes> exists, order the tuples in Result according to ORDER BY clause.
- Return the final Result.

Example

```
SELECT name, SUM(length)
FROM MoveExec, Movies
WHERE producerC# = cert#
GROUP BY name
HAVING MIN(year) < 1930;
```

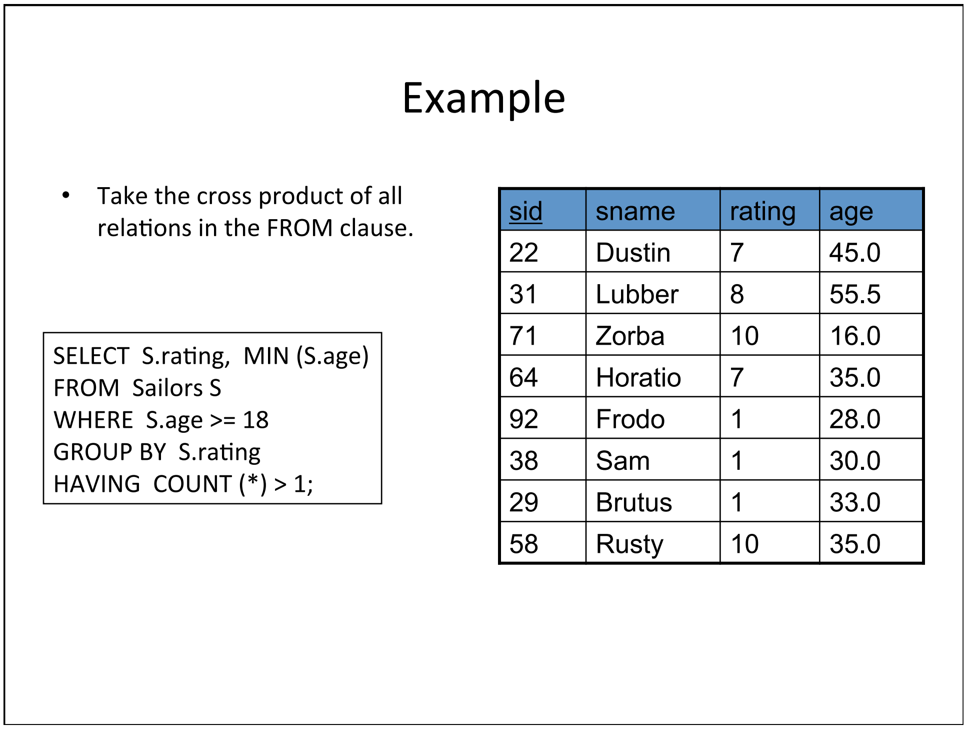
Find the total film length for only those producers who made at least one film prior to 1930.

Example

- 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
92	Frodo	1	28.0
38	Sam	1	30.0
29	Brutus	1	33.0
58	Rusty	10	35.0

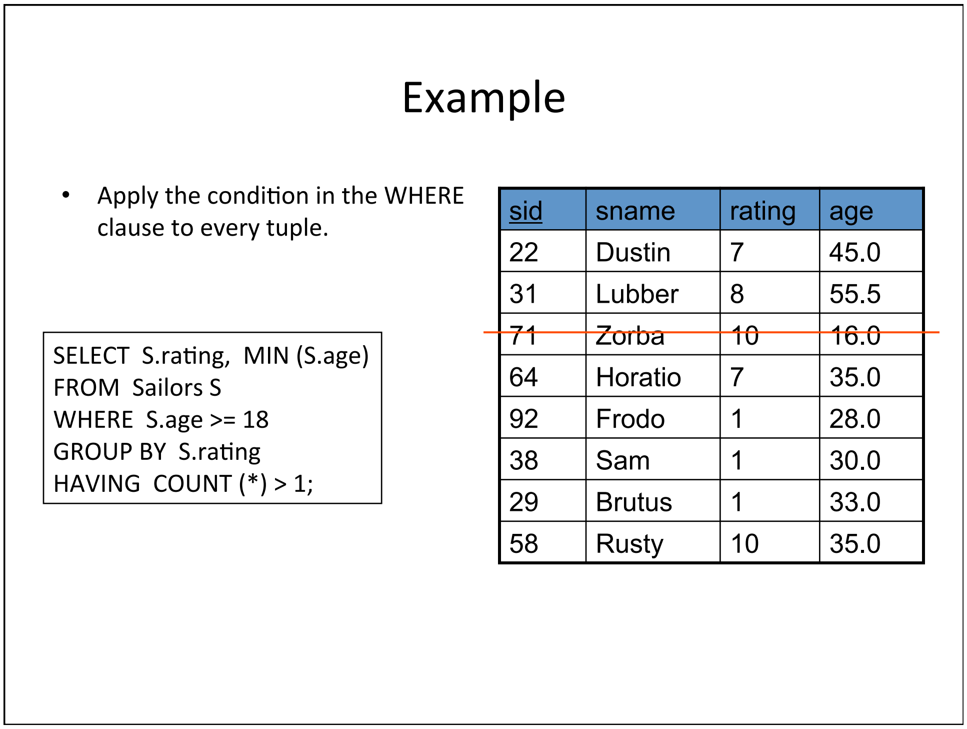


Example

- Take the cross product of all relations in the FROM clause.

```
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
92	Frodo	1	28.0
38	Sam	1	30.0
29	Brutus	1	33.0
58	Rusty	10	35.0



Example

- Apply the condition in the WHERE clause to every tuple.

```
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
92	Frodo	1	28.0
38	Sam	1	30.0
29	Brutus	1	33.0
58	Rusty	10	35.0

Example

- For simplicity, let's ignore the rest of the columns (as they are not needed by SELECT, GROUP BY, or HAVING).

```
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
~~64~~	~~Horatio~~	7	35.0
~~92~~	~~Frodo~~	1	28.0
~~38~~	~~Sam~~	1	30.0
~~29~~	~~Brutus~~	1	33.0
~~58~~	~~Rusty~~	10	35.0

Example

- **Sort** the table according to the GROUP BY columns.

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

rating	age
7	45.0
8	55.5
7	35.0
1	28.0
1	30.0
1	33.0
10	35.0

rating	age
1	28.0
1	30.0
1	33.0
7	35.0
7	45.0
8	55.5
10	35.0

Example

- Apply condition of HAVING clause to each group. Eliminate groups which do not satisfy the condition of HAVING clause.
- Evaluate SELECT clause.

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

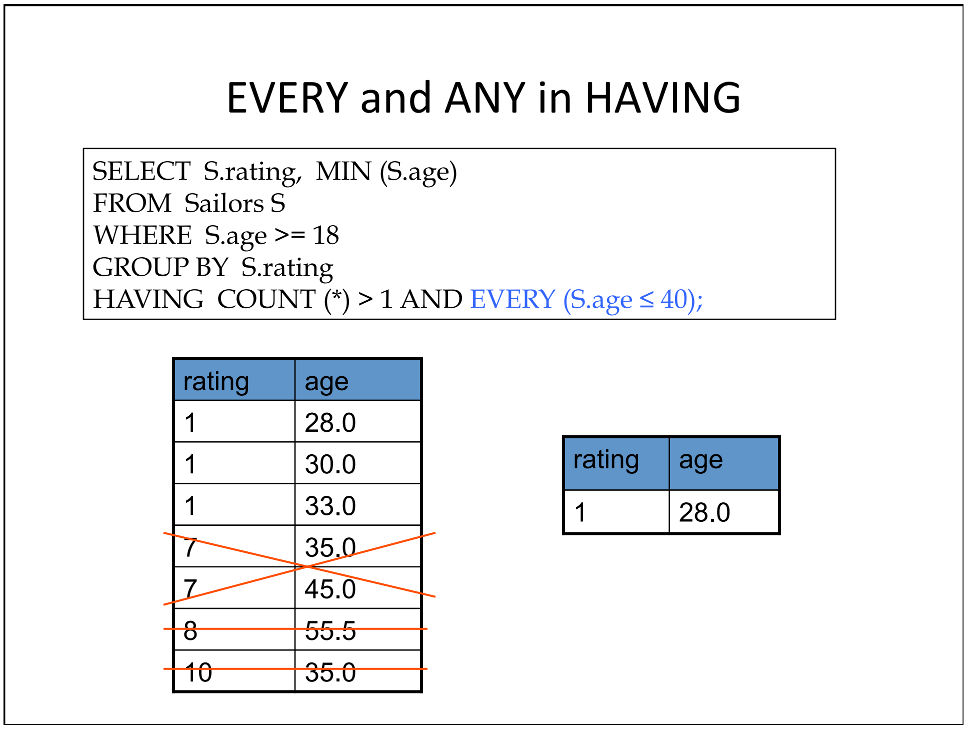
rating	age
1	28.0
1	30.0
1	33.0
7	35.0
7	45.0
~~8~~	~~55.5~~
~~10~~	~~35.0~~

Example

- Generate one tuple for each group according to SELECT clause.

rating	age
1	28.0
7	35.0

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

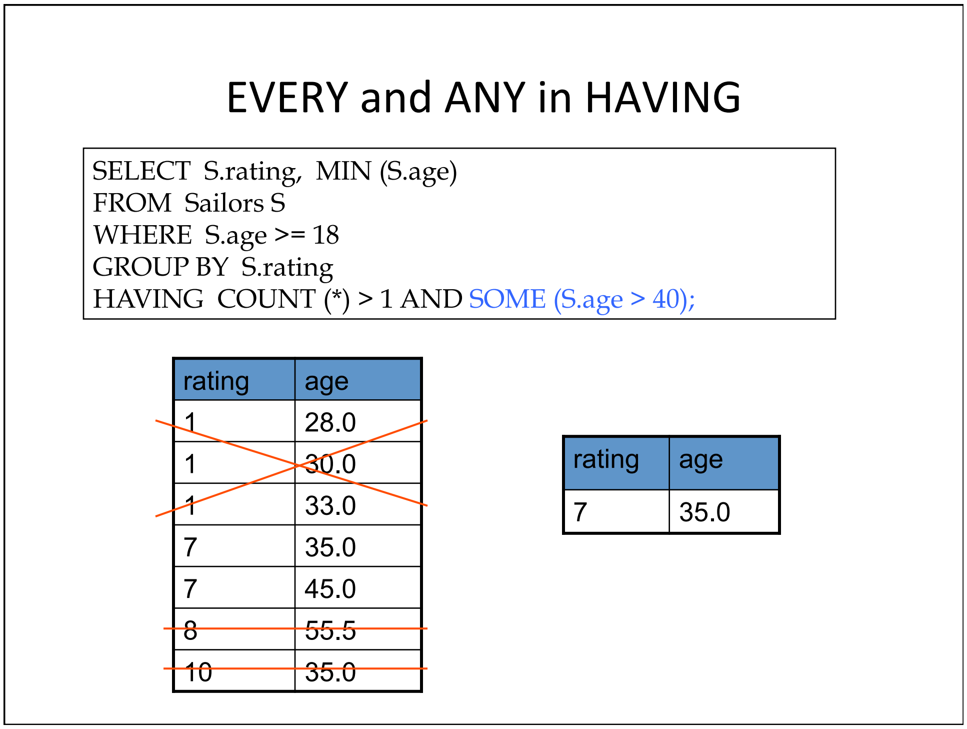


EVERY and ANY in HAVING

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1 AND EVERY (S.age ≤ 40);
```

rating	age
1	28.0
1	30.0
1	33.0
~~7~~	~~35.0~~
~~7~~	~~45.0~~
~~8~~	~~55.5~~
~~10~~	~~35.0~~

rating	age
1	28.0



EVERY and ANY in HAVING

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1 AND SOME (S.age > 40);
```

rating	age
~~1~~	~~28.0~~
~~1~~	~~30.0~~
~~1~~	~~33.0~~
7	35.0
7	45.0
~~8~~	~~55.5~~
~~10~~	~~35.0~~

rating	age
7	35.0

More examples

- Find the minimum age of sailors in each rating category such that the average age of sailors in that category is greater than the minimum age of all sailors.

```
SELECT S.rating, MIN(S.age)
FROM Sailors S
GROUP BY S.rating
HAVING AVG(S.age) > (SELECT MIN(age)
                     FROM Sailors);
```

- Find the second minimum age of sailors.

```
SELECT MIN(age)
FROM Sailors
WHERE age > (SELECT MIN(age)
             FROM Sailors);
```

- What happens when there is only one sailor?
- What happens when all sailors have the same age?
- Find the third minimum age of sailors?

More examples

Customers

sid	Cname	level	type	age
36	Cho	Beginner	snowboard	18
34	Luke	Inter	snowboard	25
87	Ice	Advanced	ski	20
39	Paul	Beginner	ski	33

Activities

sid	slope-id	day
36	s3	01/05/09
36	s1	01/06/09
36	s1	01/07/09
87	s2	01/07/09
87	s1	01/07/09
34	s2	01/05/09

Slopes

slope-id	name	color
s1	Mountain Run	blue
s2	Olympic Lady	black
s3	Magic Carpet	green
s4	KT-22	black

COUNT

- Find the total number of customers

```
SELECT COUNT(sid)
FROM Customers;
```

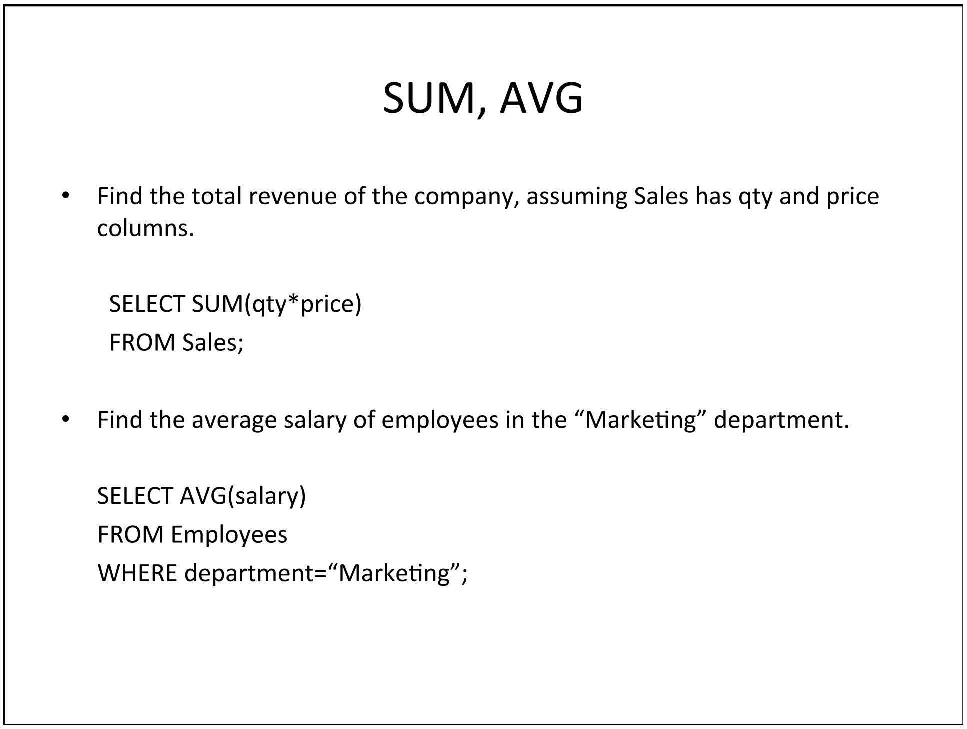
- Find the total number of days of operation

```
SELECT COUNT(distinct(day))
FROM Activities;
```

```
SELECT COUNT(day)
FROM Activities;
```

•Alternatively, the last query could have been written as

```
SELECT COUNT(*)
FROM Activities
```



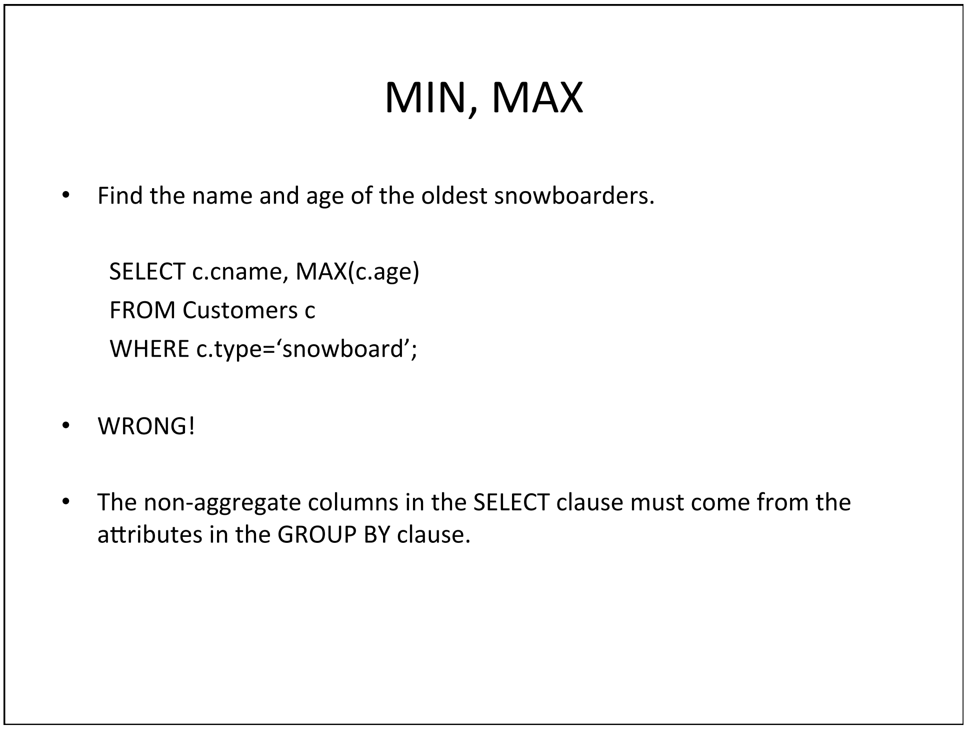
SUM, AVG

- Find the total revenue of the company, assuming Sales has qty and price columns.

```
SELECT SUM(qty*price)
FROM Sales;
```

- Find the average salary of employees in the “Marketing” department.

```
SELECT AVG(salary)
FROM Employees
WHERE department=“Marketing”;
```

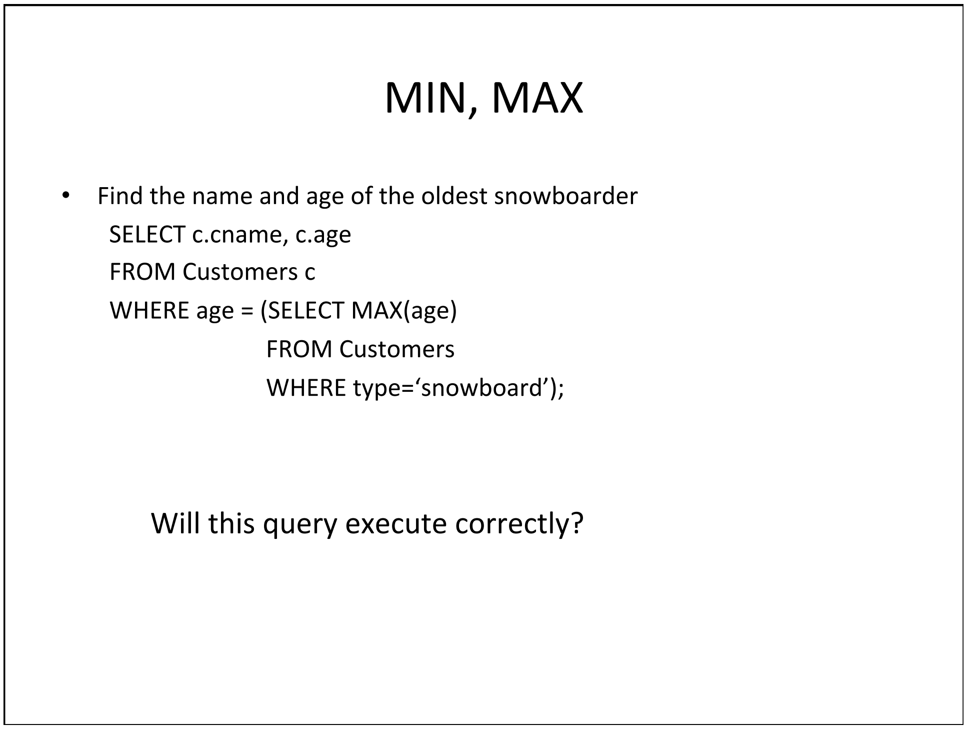


MIN, MAX

- Find the name and age of the oldest snowboarders.

```
SELECT c.cname, MAX(c.age)
FROM Customers c
WHERE c.type=‘snowboard’;
```

- WRONG!
- The non-aggregate columns in the SELECT clause must come from the attributes in the GROUP BY clause.

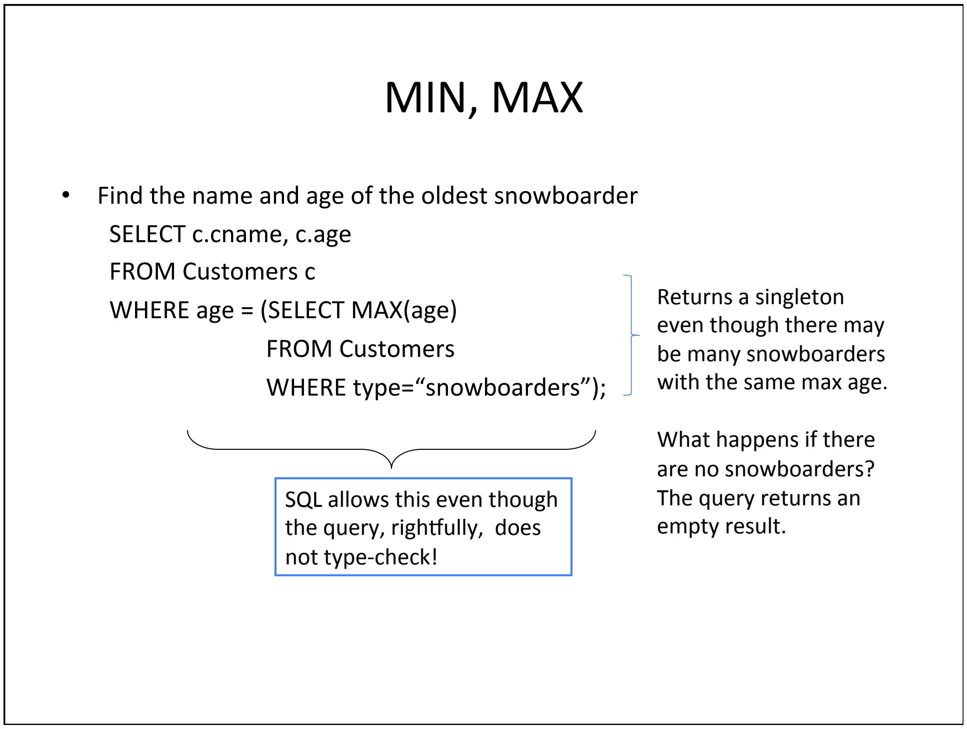


MIN, MAX

- Find the name and age of the oldest snowboarder

```
SELECT c.cname, c.age
FROM Customers c
WHERE age = (SELECT MAX(age)
             FROM Customers
             WHERE type='snowboard');
```

Will this query execute correctly?



MIN, MAX

- Find the name and age of the oldest snowboarder

```
SELECT c.cname, c.age
FROM Customers c
WHERE age = (SELECT MAX(age)
             FROM Customers
             WHERE type="snowboarders");
```

Returns a singleton even though there may be many snowboarders with the same max age.

SQL allows this even though the query, rightfully, does not type-check!

What happens if there are no snowboarders? The query returns an empty result.

On a similar note…

- Find the activities of Luke.

```
SELECT *
FROM Activities a
WHERE a.sid = (SELECT sid
               FROM Customers c
                WHERE cname=‘Luke’);
```

•If there is only one Luke in the Customers table, the subquery returns only one sid value. SQL returns that single sid value to be compared with a.sid.
•However, if the subquery returns more than one value, a run-time error occurs.

More Examples

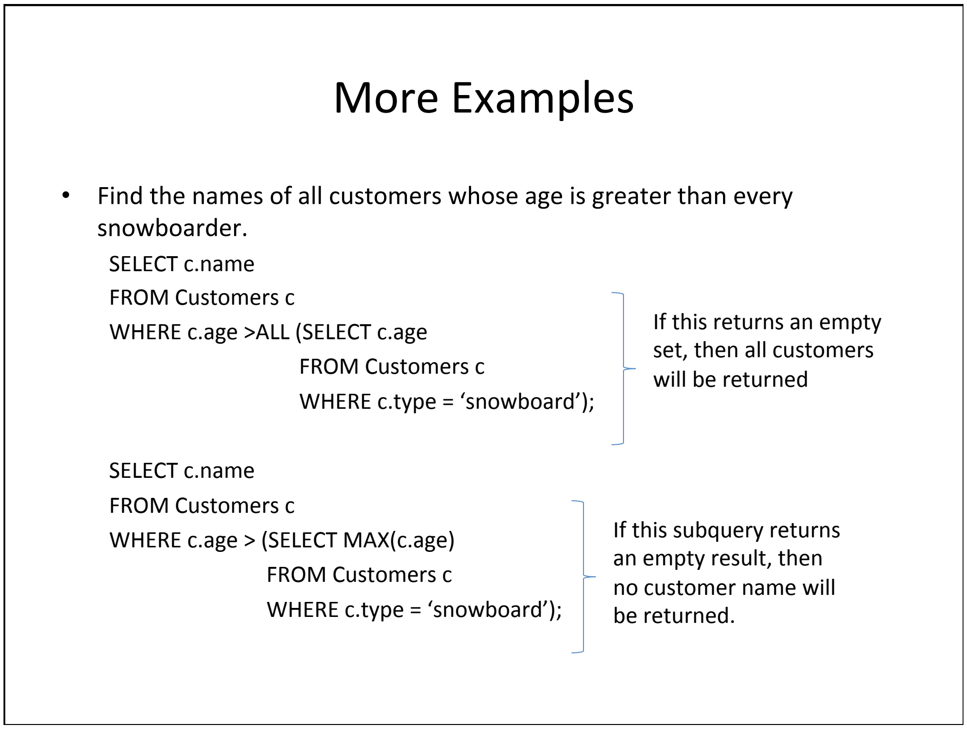
- Find the names of all customers whose age is greater than every snowboarder.

```
SELECT c.name
FROM Customers c
WHERE c.age >ALL (SELECT c.age
                  FROM Customers c
                  WHERE c.type = ‘snowboard’);
```

What happens if the there are no snowboarders?

```
SELECT c.name
FROM Customers c
WHERE c.age > (SELECT MAX(c.age)
               FROM Customers c
               WHERE c.type = ‘snowboard’);
```

What happens there are no snowboarders?



More Examples

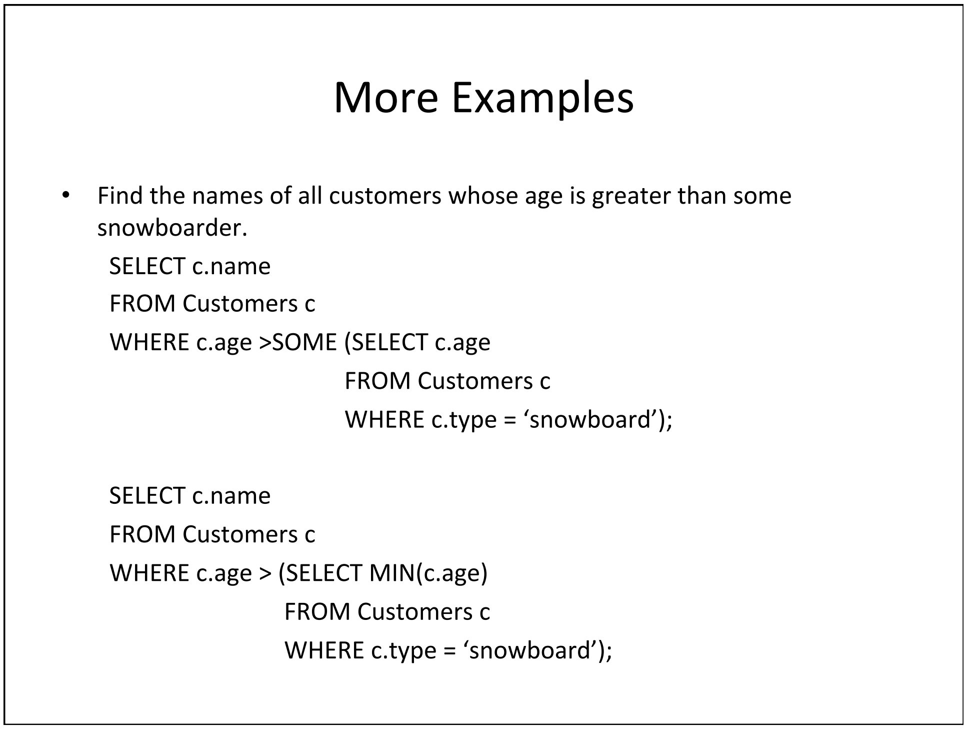
- Find the names of all customers whose age is greater than every snowboarder.

```
SELECT c.name
FROM Customers c
WHERE c.age >ALL (SELECT c.age
                  FROM Customers c
                  WHERE c.type = ‘snowboard’);
```

If this returns an empty set, then all customers will be returned

```
SELECT c.name
FROM Customers c
WHERE c.age > (SELECT MAX(c.age)
               FROM Customers c
               WHERE c.type = ‘snowboard’);
```

If this subquery returns an empty result, then no customer name will be returned.

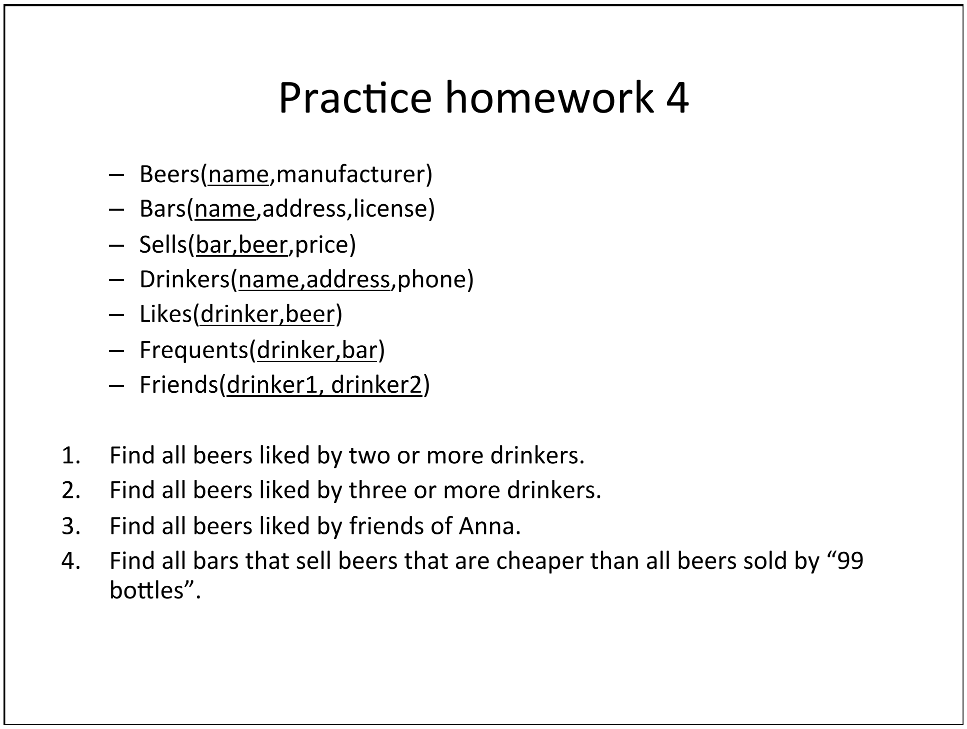


More Examples

- Find the names of all customers whose age is greater than some snowboarder.

```
SELECT c.name
FROM Customers c
WHERE c.age >SOME (SELECT c.age
                   FROM Customers c
                   WHERE c.type = ‘snowboard’);

SELECT c.name
FROM Customers c
WHERE c.age > (SELECT MIN(c.age)
               FROM Customers c
               WHERE c.type = ‘snowboard’);
```



Practice homework 4

- Beers(name,manufacturer)
- Bars(name,address,license)
- Sells(bar,beer,price)
- Drinkers(name,address,phone)
- Likes(drinker,beer)
- Frequents(drinker,bar)
- Friends(drinker1, drinker2)

1. Find all beers liked by two or more drinkers.
2. Find all beers liked by three or more drinkers.
3. Find all beers liked by friends of Anna.
4. Find all bars that sell beers that are cheaper than all beers sold by “99 bottles”.