

Lecture 5:

SQL3, Aggregates and GROUP BY

Instructor: Shel Finkelstein

Reference:

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

Important Notices

- Lecture slides and other class information will be posted on [Piazza](#) (not Canvas).
 - Slides are posted under Resources→Lectures
 - Lecture Capture recordings are available to all students under Yuja.
 - Recording for CSE 180 lecture for Wednesday, January 17 was posted directly on Yuja, since I was at a conference on that date.
 - There's no Lecture Capture for Lab Sections.
- All Lab Sections (and Office Hours) normally meet In-Person, with rare exceptions announced on Piazza.
 - Some slides for Lab Sections have been posted on Piazza under Resources→Lab Section Notes
- Office Hours for TAs and me are posted in Syllabus and Lecture1, as well as in Piazza notice [@7](#); that post also now includes hours for Group Tutors.
- Some suggestions about use (and non-use) of Generative AI systems such as ChatGPT were posted on Piazza in [@41](#), with specific discussion about the Movies tables and the four Avatar queries in Lecture 4.

Important Notices

- Lab2 assignment was posted on Piazza under Resources→Lab2 on Wed, January 24. General Information about appears under Resources-->General Resources.
 - See Piazza announcement about Lab2, which is much more difficult than Lab1.
 - Lab2 is due **Tuesday, February 6**, by 11:59pm.
 - Late Lab Assignments will not be accepted; be sure that you post the correct file!
 - Be sure to do Lab assignments individually, but you can get help on concepts.
 - Your solution should be submitted via Canvas as a zip file.
 - Canvas will be used for both Lab Assignment submission and grading.
 - You will have enough information to complete Lab2 after Lecture on Mon, January 29.
 - A load file to help you test your Lab2 solution was posted under Resources→Lab2
 - We won't provide answers for your Lab2 queries on that test data, but in [@55](#), we told you how many tuples should appear in the results of your query.
 - But testing can only prove that a solution is incorrect, **not** that it is correct.
 - Using ChatGPT to help understand CSE 180 concepts (or even ask what SQL queries you wrote actually do) is great ...
 - ... but if you do use systems such as ChatGPT to do Lab Assignments, you're mostly cheating yourself ...
 - ... because you can't use such systems on the Midterms or Final!
 - Lab2 has been (and will be) discussed at Lab Sections and Tutoring.

Important Notices

- Second Gradiance Assignment, CSE 180 Winter 2024 #2, was assigned on Tuesday, January 30.
 - It is due on **Tuesday, February 6, by 11:59pm.**
 - The day of the week on which Gradiance Assignments are due will vary.
 - There are 9 questions in this Gradiance Assignment.
 - Some of them may be difficult.
 - You should treat NULL as the **smallest** value for the ORDER BY question in Gradiance #2.
 - Website and Class Token for Gradiance are in the Syllabus and first Lecture.
 - If you forget your Gradiance password, you can recover it by providing your Gradiance login and email.
 - You may take each Gradiance Assignment as many time as you like.
 - Questions stay the same, but answers change.
 - Your Gradiance Assignment score is the score on your **Final** Gradiance submission before the deadline.
 - Gradiance does not allow late submissions, so no extensions are possible.

Very Important Notice

- The Winter 2024 CSE 180 Final is on **Wednesday, March 20** from 8:00am – 11:00am in our usual classroom.
 - The Syllabus and First Lecture originally said that it was on Tuesday, March 19.
 - That's been fixed. Please adjust your calendars accordingly!
- The Winter 2024 CSE 180 Midterm is on Wednesday, February 14
 - That was correctly described in the Syllabus and First Lecture!
 - The Winter 2023 CSE 180 Midterm was posted under Resources → Exams on Piazza on Thursday, February 1.
 - Answers to that Midterm will be posted at the same location on Wednesday, February 7.
 - However, we strongly suggest that you take the parts of that exams that we've covered in class, rather than just read the answers.
 - But last year's Midterm used that quarter's database schema, which is different from this quarter's schema.
 - More info on the Midterm will appear soon in Lectures and on Piazza.

Aggregates

- Basic SQL has 5 aggregation operators: SUM, AVG, MIN, MAX, COUNT.
- Aggregation operators are applied on scalar values, that is, a scalar attribute such as salary or $1.1 * \text{salary}$.
 - An exception: COUNT(*) which counts the number of tuples.
- Used for computing summary results over a table. Examples:
 - Find the average/min/max score of all students who took CSE180.
 - Find the number of movies released in 2020.
 - Find the 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 [different] non-NULL values in the A column.
 - SUM([DISTINCT] A)
 - Returns the sum of all [different] non-NULL values in the A column.
 - AVG([DISTINCT] A)
 - Returns the average of all [different] non-NULL values in the A column.
 - MAX(A) and MIN(A)
 - Return the maximum value or minimum non-NULL value in the A column.

Aggregation Example

- MovieExec(name, address, cert#, netWorth)
SELECT AVG(netWorth)
FROM MovieExec;

We're looking at all the movie executives together as a single group, and there will be one tuple in the result.

- Finds the average of “netWorth” values for all tuples in the relation MovieExec.

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

More Aggregation Examples

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

```
SELECT COUNT(length)  
FROM Movies;
```

```
SELECT COUNT(DISTINCT length)  
FROM Movies;
```

```
SELECT MAX(length), MIN(length)  
FROM Movies;
```

Aggregation and Grouping Example

- Movies(movieTitle, movieYear, length, genre, studioName, producerC#)
SELECT studioName, SUM(length)
FROM Movies
GROUP BY studioName;
- For each studio, find the sum of lengths of all movies from that studio.

Movies

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

Aggregation and Grouping

- GROUP BY clause follows the WHERE clause.

```
SELECT [DISTINCT] c1, c2, ..., cm, AGGOP(...)  
FROM R1, R2, ..., Rn  
[WHERE condition]  
[GROUP BY <list of grouping attributes>]  
[ORDER BY <list of attributes [ASC | DESC] >]
```

If there's a GROUP BY clause, or if the SELECT clause has aggregates AGGOP, then c_1, c_2, \dots, c_m must come from the list of grouping attributes.

- Let Result begin as an empty multiset of tuples.
- For every tuple t_1 from R_1 , t_2 from R_2 , ..., t_n from R_n
 - If t_1, \dots, t_n satisfy *condition* (i.e., condition evaluates to true), then add the resulting tuple that consists of c_1, c_2, \dots, c_m components (including attributes of AGGOP operators) of the t_i into Result.
- Group the 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(s) on tuples in each group to get tuple put in Result.
- If ORDER BY <list of attributes> exists, order the tuples in Result according to the ORDER BY clause.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- Return the final Result.

Grouping and Aggregation Examples

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

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

But never use GROUP BY unnecessarily as an alternative way to get rid of duplicates when you're not sure whether or not to use DISTINCT.

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

Movies(movieTitle, movieYear, length, genre, studioName, producerC#)
MovieExec(execName, address, cert#, netWorth)

```
SELECT e.execName, AVG(m.length)  
FROM MovieExec e, Movies m  
WHERE m.producerC# = e.cert#  
GROUP BY e.execName;
```

For each exec, show the exec's name, and the average length of movies made by that exec.

Wrong: All execs with the same name will be grouped together, even if they are have different cert# values!

What's the Result?

A	B	C	D
a1	b1	1	7
a1	b1	2	8
a2	b1	3	9
a3	b2	4	10
a2	b1	5	11
a1	b1	6	12

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

What if the query asked for B after SUM(C)?

What if the query didn't ask for A in the SELECT?

Does SELECT DISTINCT have any effect when there's a GROUP BY?

Grouping, Aggregation, and Nulls

- NULLs are ignored in any aggregation.
 - They do not contribute to the SUM, AVG, COUNT, MIN, MAX of an attribute.
 - COUNT(*) = number of tuples in a relation (**even if some columns are null**)
 - COUNT(A) is the number of tuples with **non-null** values for attribute A
- SUM, AVG, MIN, MAX on an empty result (no tuples) is NULL.
 - COUNT of an empty result is 0.
 - I think that SUM on an empty result should be 0 ... **but it isn't, it's NULL.**
- GROUP BY does not ignore NULLs.
 - The groups that are formed with a GROUP BY on attributes A_1, \dots, A_k may have NULL values for one or more of these attributes.

Examples with NULL

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

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

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

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

HAVING Clause

```
SELECT [DISTINCT] c1, c2, ..., cm, AGGOP(...)
FROM   R1, R2, ..., Rn
[WHERE condition]
[GROUP BY <list of grouping attributes>]
[HAVING condition]
[ORDER BY <list of attributes [ASC | DESC] >]
```

Note that HAVING clause
cannot exist without
GROUP BY

- HAVING: Choose groups based on some aggregate property of the group itself.
 - Think of it as like a WHERE clause applied to groups.
- The same attributes and aggregates that can appear in the SELECT can appear in the HAVING clause condition.
 - Which attributes and aggregates? Why?

Semantics of HAVING

- Let Result begin as an empty multiset of tuples.
- For every tuple t_1 from R_1 , t_2 from R_2 , ..., t_n from R_n
 - If t_1, \dots, t_n satisfy *condition* (i.e., condition evaluates to true), then add the resulting tuple that consists of c_1, c_2, \dots, c_m (including attributes in AGGROP operators) components of the t_i into Result.
- Group the 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.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- Return the final Result.

HAVING Example 1

(Assume that MovieExec.execName is UNIQUE)

```
SELECT e.execName, SUM(m.length)
FROM MovieExec e, Movies m
WHERE m.producerC# = e.cert#
GROUP BY e.execName
HAVING MIN(m.movieYear) < 1930;
```

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

HAVING Example 2

(Assume that MovieExec.execName is UNIQUE)

```
SELECT e.execName, SUM(m.length)
FROM MovieExec e, Movies m
WHERE m.producerC# = e.cert#
GROUP BY e.execName
HAVING COUNT(DISTINCT m.movieYear) >= 4;
```

Find the total film length for just those producers who made films in at least 4 different years.

What would happen if MovieExec.execName wasn't UNIQUE?

HAVING Example 3

```
SELECT e.execName, SUM(m.length), MAX(m.movieYear)
FROM MovieExec e, Movies m
WHERE m.producerC# = e.cert#
GROUP BY e.execName
HAVING COUNT(DISTINCT m.movieYear) >= 4
      AND MIN(m.movieYear) < 1930;
```

Find the total film length and the latest movie year, for just those producers who made movies in at least 4 different years, and who made at least one film prior to 1930.

HAVING Example

For each rating that has at least 2 sailors whose age is greater than or equal to 18, find the age of the youngest sailor whose age is greater than or equal to 18.

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

<u>sid</u>	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 of HAVING Semantics: 1

- 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;
```

<u>sid</u>	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 of HAVING Semantics: 2

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

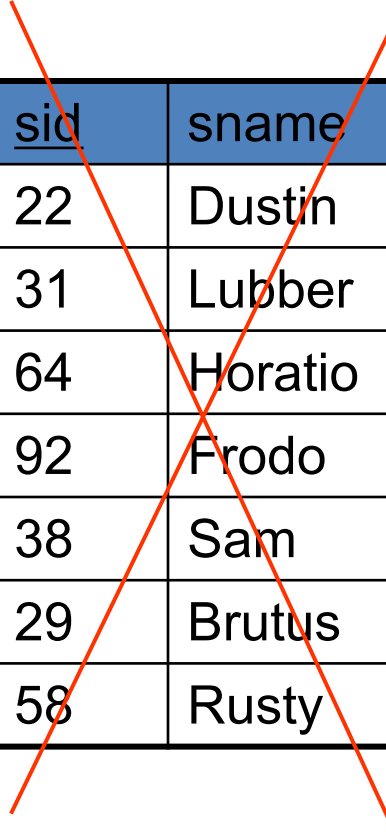
```
SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1;
```

<u>sid</u>	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 of HAVING Semantics: 3

- For simplicity, let's ignore the rest of the columns (since they are not needed).

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



<u>sid</u>	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 of HAVING Semantics: 4

- **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

Note: Don't actually have to sort to do GROUP BY

Example of HAVING Semantics: 5

- Apply condition of HAVING clause to each group. Eliminate the groups which do not satisfy the condition in the HAVING clause.
- Next, we evaluate the 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 of HAVING Semantics: 6

- Generate one tuple for each group, according to the 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
7	35.0

More Examples

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

```
SELECT S.rating, MIN(S.age)
FROM Sailors S
GROUP BY S.rating
HAVING MIN(S.age) > ( SELECT AVG(S2.age)
                      FROM Sailors S2 );
```

More Examples

- Find the second minimum age of sailors.

```
SELECT MIN(S.age)
FROM Sailors S
WHERE S.age > ( SELECT MIN(S2.age)
                FROM Sailors S2 );
```

- What happens when there is only one sailor?
- What happens when all sailors have the same age?
- What happens when there are no sailors?
- Can you figure how to find the third minimum age of sailors?

Some Incorrect Examples

- Find the second minimum age of sailors.
All examples below are not legal SQL!

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

```
SELECT MIN(S.age)
FROM Sailors S
WHERE S.age > MIN(S2.age);
```

```
SELECT MIN(S.age)
FROM Sailors S
WHERE S.age > ( MIN(S2.age)
                FROM Sailors S2 );
```

```
SELECT MIN(S.age)
FROM Sailors S
WHERE S.age > MIN ( SELECT S2.age
                    FROM Sailors S2 );
```

Aggregates can't appear in WHERE clauses, except in legal subqueries, as on previous slides.

More Examples

Customers

<u>cid</u>	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

<u>cid</u>	<u>slopeid</u>	<u>day</u>
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

<u>slopeid</u>	name	color
s1	Mountain Run	blue
s2	Olympic Lady	black
s3	Magic Carpet	blue
s4	KT-22	green

COUNT Examples

- Find the total number of customers

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

- Find the total number of days that customers engaged in activities

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

- Compare to:

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

Alternatively, that last query could have been written as:

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

But only because day can't be NULL

COUNT Examples, with Join

- Find the number of activities done by advanced customers .

```
SELECT COUNT(a.cid)
FROM Customers c, Activities a
WHERE a.cid = c.cid
      AND c.level = 'Advanced';
```
- Find the number of activities done by different advanced customers.

```
SELECT COUNT(DISTINCT a.cid)
FROM Customers c, Activities a
WHERE a.cid = c.cid
      AND c.level = 'Advanced';
```

Would these queries have same results with COUNT(c.cid) instead of COUNT(a.cid)?

What about if queries had COUNT(*)/COUNT(DISTINCT *) instead?

COUNT Examples, with JOIN and GROUP BY

- For each day, find the number of activities that were done by advanced customers.

```
SELECT a.day, COUNT(a.cid)
FROM Customers c, Activities a
WHERE a.cid = c.cid
      AND c.level = 'Advanced'
GROUP BY a.day;
```

- For each day, find the number of different advanced customers who did at least one activity.

```
SELECT a.day, COUNT(DISTINCT a.cid)
FROM Customers c, Activities a
WHERE a.cid = c.cid
      AND c.level = 'Advanced'
GROUP BY a.day;
```

COUNT Examples, with JOIN and GROUP BY with Two Attributes

- For each level and day, find the number of activities done that day by customers at that level.

```
SELECT c.level, a.day, COUNT(a.cid)
FROM Customers c, Activities a
WHERE a.cid = c.cid
GROUP BY c.level, a.day;
```

- For each level and day, find the number of activities done that day by different customers at that level..

```
SELECT c.level, a.day, COUNT(DISTINCT a.cid)
FROM Customers c, Activities a
WHERE a.cid = c.cid
GROUP BY c.level, a.day;
```

COUNT Examples, with JOIN and GROUP BY and HAVING

- For each customer level, find the number of times that customers who are at that level went on a red slope, giving level as well as number of times, ... but only if the number of times is at least 3.
 - The number of times should appear as redCount in the result.

```
SELECT c.level, COUNT(*) AS redCount
FROM Customers c, Activities a, Slopes s
WHERE a.cid = c.cid
      AND a.slopeid = s.slopeid
      AND s.color = 'Red'
GROUP BY c.level
HAVING COUNT(*) >= 3;
```

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 with Subquery

- Find the name and age of the oldest snowboarders.

```
SELECT c.cname, c.age
FROM Customers c
WHERE c.age = (SELECT MAX(c2.age)
               FROM Customers c2
               WHERE c2.type='snowboard');
```

Will this query execute correctly?

NO!

If not, how would you correct it?

MIN, MAX

- Find the age of the youngest participant for each type of activity that Beginners participate in.

```
SELECT c.type, MIN(c.age)
FROM Customers c
WHERE c.level='Beginner';
```

- Wrong!
- The non-aggregate columns in the SELECT clause must come from the attributes in the GROUP BY clause.
 - If there is an aggregate in the SELECT clause but no GROUP BY, then all tuples satisfying the WHERE clause are in a single group, and no attributes are in the GROUP BY clause, hence ... (what?).

MIN, MAX with GROUP BY

- Find the age of the youngest participant for each type of activity that Beginners participate in.

```
SELECT c.type, MIN(c.age)
FROM Customers c
WHERE c.level='Beginner'
GROUP BY c.type;
```

- Wrong!
- Selects age of youngest Beginner for activity types that Beginners participate in.

MIN, MAX with GROUP BY and EXISTS

- Find the age of the youngest participant for each type of activity that Beginners participate in.

```
SELECT c.type, MIN(c.age)
FROM Customers c
WHERE EXISTS ( SELECT *
                FROM Customers c2
                WHERE c2.level='Beginner'
                  AND c2.type=c.type )
GROUP BY c.type;
```

- Right!
- Selects age of youngest participant for activity types that at least one Beginner participates in.

MIN, MAX with GROUP BY and HAVING

- Find the age of the youngest participant for each type of activity that Beginners participate in.

```
SELECT c.type, MIN(c.age)
FROM Customers c
GROUP BY c.type
HAVING SOME ( c.level='Beginner' );
```

- Right
- Selects age of youngest participant for activity types that at least one Beginner participates in.

ANY/SOME 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

ANY/SOME in HAVING is in the SQL standard
... but PostgreSQL doesn't support it!
So don't use it in Lab Assignments, but treat
it as legal on exams.

EVERY (not ALL) 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 in HAVING is in the SQL standard
... and PostgreSQL does support it!

Careful ...

- Find the activities of Luke.

```
SELECT *  
FROM Activities a  
WHERE a.cid = (SELECT c.cid  
                FROM Customers c  
                WHERE c.cname='Luke');
```

If there is only one Luke in the Customers table, the subquery returns only one cid value. SQL returns that single cid value to be compared with a.cid.

However, if the subquery returns more than one value, a **run-time error** occurs.

Reminder: Use of ALL

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

```
SELECT c.name
FROM Customers c
WHERE c.age > ALL (SELECT c2.age
                   FROM Customers c2
                   WHERE c2.type = 'snowboard');
```

What happens if there are no snowboarders?

```
SELECT c.name
FROM Customers c
WHERE c.age > (SELECT MAX(c2.age)
               FROM Customers c2
               WHERE c2.type = 'snowboard');
```

What happens if there are no snowboarders?

Reminder: Use of ANY/SOME (Synonyms)

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

```
SELECT c.name  
FROM Customers c  
WHERE c.age > SOME (SELECT c2.age  
                     FROM Customers c2  
                     WHERE c2.type = 'snowboard');
```

What happens if there
are no snowboarders?

```
SELECT c.name  
FROM Customers c  
WHERE c.age > (SELECT MIN(c2.age)  
              FROM Customers c2  
              WHERE c2.type = 'snowboard');
```

What happens if there
are no snowboarders?

GROUP BY: Relaxing a Restriction

- For each cid, give cid and the days on which customer did an activity, with COUNT of number of activities done on that day.

```
SELECT c.cid, a.day, COUNT(*)  
FROM Customers c, Activities a  
WHERE a.cid = c.cid  
GROUP BY c.cid, a.day;
```

- For each cid, give cid, level and the days on which customer did an activity , with COUNT of number of activities done on that day.

```
SELECT c.cid, c.level, a.day, COUNT(*)  
FROM Customers c, Activities a  
WHERE a.cid = c.cid  
GROUP BY c.cid, a.day;
```

- Is this legal SQL?
- We'll accept this on tests and homeworks, if you do it right.

Even though level isn't a GROUP BY attribute, this is okay in most SQL databases!

Why? cid is the entire primary key of Customers, so level is not ambiguous.

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 a beer that is cheaper than all beers sold by the bar '99 Bottles'.