SQL (Part 3)

Instructor: Shel Finkelstein

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

Important Notices

- Lab2 assignment was posted Sunday, Jan 22 on Piazza.
  - Due by **Sunday, Feb 5, 11:59pm** on Canvas as zip file
    - No late submissions, no make-up assignments
  - Lab2 will be discussed at this week's and next week's Labs.
  - A load script for Lab2 was posted on Piazza on Tuesday, Jan 24.
  - A change to Query6 description was also posted on Tuesday, Jan 24.
- First Gradiance Assignment is due by **Friday, Jan 27, 11:59pm**
- **Reminder:** Midterm is on **Monday, Feb 13**; no make-ups
  - You may bring a **single two-sided 8.5" x 11" sheet of paper** with as much info written (or printed) on it as you can fit and read unassisted.
    - No sharing of these sheets will be permitted.
  - For DSC accommodation, please submit forms to me well in advance
- Sign-up for LSS tutoring with Alexander Ou, if interested
  - See Piazza announcement; slots won't be available if you delay.

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*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 CMPS180.
  - Find the number of movies released in 2012.
  - 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] values in the A column.
  - SUM([DISTINCT] A)
    - Returns the sum of all [different] values in the A column.
  - AVG([DISTINCT] A)
    - Returns the average of all [different] values in the A column.
  - MAX(A)/MIN(A)
    - Returns the maximum value or minimum value in the A column.

Aggregation Example

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

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

- Finds the average of “netWorth” values for 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 StarsIn;

SELECT COUNT(starName)
FROM StarsIn;

SELECT COUNT(DISTINCT starName)
FROM StarsIn;

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

Aggregation and Grouping Example

- Movies(title, year, 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 SELECT clause has aggregates AGGOP, then $c_1, c_2, \ldots, c_m$ 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, …, t_n satisfy *condition* (i.e., condition evaluates to true), then add the resulting tuple that consists of c_1, c_2, …, 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;
```

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

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

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

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

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 query asked for B after SUM(C)?

What if query didn’t ask for B in the SELECT?

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, …, 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] $c_1, c_2, \ldots, c_m$ AGGOP(…)
FROM $R_1, R_2, \ldots, R_n$
[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, …, 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 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

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

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

HAVING Example

Find the age of the youngest sailor with age ≥ 18, for each rating that has 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 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;
```

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 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;
```

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 of HAVING Semantics: 3

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

```
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 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 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 of HAVING Semantics: 6

- Generate one tuple for each group according to 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 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(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?

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>slope-id</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>slope-id</u>	sname	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, the last query could have been written as:

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

But only if day can’t be NULL

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 <u>no attributes</u> 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

EVERY 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

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?

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".