CPSC 304 – Administrative notes October 30 and 31, 2024

- Midterm is back!
  - See Piazza – in general you all did quite well
- Project:
  - Make sure you have your end-to-end project tech stack working now!!!
  - Milestone 4: Project implementation – due November 29
  - Milestone 5: Group demo – week of December 2
  - Milestone 6: Individual Assessment – Due November 29
- Tutorials: basically project group work time
- Final exam: December 16 at 12pm!
  - Conflict form on Piazza – due November 12

Project notes: Group work (1/3)

- Milestone 3 included information about how your group was going to divide up tasks
- It is your responsibility to do what you agreed to do in a timely fashion
  - It is not acceptable to wait until the last possible minute and then expect others to treat your failure to plan as their emergency
- It is your responsibility to communicate with your group members

Project notes
Group work (2/3)

Per the Project Description: "In most cases, group members will share the same project grade; however, the instructors and TAs reserve the right to change the grade for each participant based on a student's contributions.

For example, students who tend to be unavailable to their team, or who contribute little to the project, cannot expect to receive a good grade, even if the rest of the group gets a good grade.

Note: Even if one of your group members volunteers to do most of the work for you, you still need to do your fair share. "

Project Notes
Group Work (3/3)

- That does not mean that we will immediately decrease your grade if a group member complains against, but keep in mind that we have mechanisms for seeing when you have done what work including:
  - Peer/individual evaluations
  - If you showed up for your Milestone 3 check in
  - Your behaviour and ability to answer questions during your demo
  - Your commits to your repository

One more project note: don't use AI to do your work

From the syllabus:

"Using ChatGPT or a similar system on any work that you turn in also constitutes a violation of academic integrity."

Don't do it.

Now where were we...

- We'd been doing an awful lot of SQL
- We'd just covered everything that we could do in relational algebra

You're Now Leaving the World of Relational Algebra

- You now have many ways of asking relational algebra queries
  - For this class, you should be able write queries using all of the different concepts that we've discussed & know the terms used
  - In general, use whatever seems easiest, unless the question specifically asks you to use a specific method.
  - Sometimes the query optimizer may do poorly, and you'll need to try a different version, but we'll ignore that for this class.

Mind the gap

- But there's more you might want to know!
- E.g., "find the average age of students"
- There are extensions of Relational Algebra that cover these topics
  - We won't cover them
- We will cover them in SQL

Aggregate Operators

- These functions operate on the multiset of values of a column of a relation, and return a value

  AVG: average value
  MIN: minimum value
  MAX: maximum value
  SUM: sum of values
  COUNT: number of values

- The following versions eliminate duplicates before applying the operation to attribute A:

  COUNT (DISTINCT A)
  SUM (DISTINCT A)
  AVG (DISTINCT A)

```
SELECT count(distinct s.snum)
FROM enrolled e, Student S
WHERE e.snum = s.snum
```

```
SELECT count(s.snum)
FROM enrolled e, Student S
WHERE e.snum = s.snum
```

Aggregate Operators: Examples

students

```
SELECT  COUNT(*)
FROM    Student
```

Find name and age of the oldest student(s)

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

Finding the average age of students whose standing is SR

```
SELECT  AVG (age)
FROM   Student
WHERE  standing='SR'
```

Student(snum,sname,major,standing,age)
Class(name,meets_at,room,fid)
Enrolled(snum,cname)
Faculty(fid,fname,deptid)

Aggregation examples

- Find the minimum student age

- How many students have taken a class with “Database” in the title

Student(snum,sname,major,standing,age)
Class(name,meets_at,room,fid)
Enrolled(snum,cname)
Faculty(fid,fname,deptid)

Aggregation examples

- Find the minimum student age

```
SELECT min(age)
FROM student;
```

- How many students have taken a class with “Database” in the title

```
SELECT count(distinct snum)
FROM enrolled
WHERE cname like '%Database%'
```

GROUP BY and HAVING

- Divide tuples into groups and apply aggregate operations to each group.
- Example: *Find the age of the youngest student for each major.*

For *i* = 'Computer Science',
'Civil Engineering'…

```
SELECT  MIN (age)
FROM    Student
WHERE   major = i
```

- Problem:

  We don't know how many majors exist, not to mention this is not good practice

Grouping Examples

Find the age of the youngest student who is at least 19, for each major

```
SELECT    major, MIN(age)
FROM      Student
WHERE     age >= 19
GROUP BY  major
```

Snum	Major	Age
115987938	Computer Science	20
112348546	Computer Science	19
280158572	Animal Science	18
351565322	Accounting	19
556784565	Civil Engineering	21

No Animal Science

Major	Age
Computer Science	19
Accounting	19
Civil Engineering	21

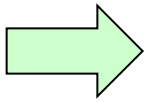
Grouping Examples with Having

Find the age of the youngest student who is at least 19, for each major with at least 2 such students

```
SELECT    major, MIN(age)
FROM      Student
WHERE     age >= 19
GROUP BY  major
HAVING  COUNT(*) > 1
```

Snum	Major	Age
115987938	Computer Science	20
112348546	Computer Science	19
280158572	Animal Science	18
351565322	Accounting	19
556784565	Civil Engineering	21

Major	Age
Computer Science	20
Computer Science	19
Accounting	19
Civil Engineering	21



Major	
Computer Science	19

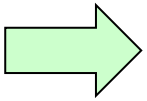
And there are rules

Find the age of the youngest student who is at least 19, for each major with at least 2 such students

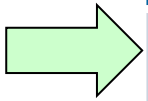
```
SELECT    major, MIN(age)
FROM      Student
WHERE     age >= 19
GROUP BY  major
HAVING    COUNT(*) > 1
```

- Would it make sense if I select age instead of MIN(age)?
- *Would it make sense if I select snum to be returned?*
- *Would it make sense if I select major to be returned?*

Major	Age
Computer Science	20
Computer Science	19
Accounting	19
Civil Engineering	21
…	…



Major	Age
Computer Science	20
Computer Science	19



Major	Age
Computer Science	19

GROUP BY and HAVING (cont)

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

- Each answer tuple corresponds to a *group:* a set of tuples with same value for all attributes in *grouping-list*
  - selected attributes must have a single value per group.
- The *target-list* may only contain:
  (i) attribute names in the *grouping-list*
  (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
- Attributes in *group-qualification* are either in the *grouping-list* or are arguments to an aggregate operator.

GROUP BY and HAVING (cont)

- Example1: *For each class, find the class name and age of the youngest student who has enrolled in this class:*

```
SELECT    cname,  MIN(age)
FROM      Student S, Enrolled E
WHERE     S.snum= E.snum
GROUP BY  cname
```

- Example2: *For each course with more than 1 enrollment, find the course name and age of the youngest student who has taken this class:*

```
SELECT    cname,  MIN(age)
FROM      Student S, Enrolled E
WHERE     S.snum = E.snum
GROUP BY  cname
HAVING    COUNT(*) > 1
```

Student(snum,sname,major,standing,age)
Class(name,meets_at,room,fid)
Enrolled(snum,cname)
Faculty(fid,fname,deptid)

← per group qualification!

Clicker question: grouping

- Compute the result of the query:

```
SELECT a1.origin, a2.dest, COUNT(*)
FROM   Flight a1, Flight a2
WHERE a1.dest = a2.origin
GROUP BY a1.origin, a2.dest
```

(The query asks for how many ways you can take each 2 hop plane trip, e.g., flights 1 and 3 is 2 hops SFO to SEA)

Which of the following is in the result?

A. (SFO,SEA,2)
B. (PIT,YVR,6)
C. (PIT,SEA,1)
D. All of the above
E. None of the above

FLIGHT:

num	origin	dest
1	SFO	YVR
2	SFO	YVR
3	YVR	SEA
4	SEA	PIT
5	SEA	PIT
6	PIT	SFO
7	PIT	SFO
8	PIT	SFO
9	PIT	YVR

clickergrouping2.sql

Clicker question: grouping

FLIGHT:

num	origin	dest
1	SFO	YVR
2	SFO	YVR
3	YVR	SEA
4	SEA	PIT
5	SEA	PIT
6	PIT	SFO
7	PIT	SFO
8	PIT	SFO
9	PIT	YVR

- Compute the result of the query:

```
SELECT a1.origin, a2.dest, COUNT(*)
FROM   Flight a1, Flight a2
WHERE a1.dest = a2.origin
GROUP BY a1.origin, a2.dest
```

(The query asks for how many ways you can take each 2 hop plane trip, e.g., flights 1 and 3 is 2 hops SFO to SEA)

Which of the following is in the result?

A. (SFO,SEA,2)
B. (PIT,YVR,6)
C. (PIT,SEA,1)
D. All of the above — correct
E. None of the above

Groupies of your very own

- Find the average age for each standing (e.g., Freshman)

Student(snum,sname,major,standing,age)
Class(name,meets_at,room,fid)
Enrolled(snum,cname)
Faculty(fid,fname,deptid)

- Find the deptID and # of faculty members for each department having a department id > 20

Groupies of your very own

- Find the standing and average age for each standing (e.g., Freshman)

```
SELECT standing, avg(age)
FROM Student
GROUP BY standing
```

- Find the deptID and # of faculty members for each department having a department id > 20

(1)
```
SELECT count(*), deptid
FROM faculty
WHERE deptid > 20
GROUP BY deptid
```

(2)
```
SELECT count(*), deptid
FROM faculty
GROUP BY deptid
HAVING deptid > 20
```

Which one works?

A: just 1

B: just 2

C: both Correct

D: neither

Grouping Examples (cont')

For each standing, find the standing and number of students who took a class with "System" in the title

```
SELECT  s.standing,  COUNT(DISTINCT s.snum) AS scount
FROM     Student S, enrolled E
WHERE   S.snum = E.snum and E.cname like '%System%'
GROUP BY  s.standing
```

- What if we do the following:
  (a) remove *E.cname like '%System%'* from the WHERE clause, and then
  (b) add a HAVING clause with the dropped condition?

```
SELECT  s.standing,  COUNT(DISTINCT s.snum) AS
scount
FROM     Student S, enrolled E
WHERE   S.snum = E.snum
GROUP BY  s.standing
HAVING  E.cname like '%System%'
```

E.Cname not in groupby
Error!

Clicker question: having

Suppose we have a relation with schema R(A, B, C, D, E). If we issue a query of the form:

```
SELECT ...
  FROM R
  WHERE ...
  GROUP BY B, E
  HAVING ???
```

Identify, in the list below, the term that CANNOT appear in the HAVING clause (where the ??? is).

A. A (unaggregated)
B. B (unaggregated)
C. Count(B)
D. All can appear
E. None can appear

Clicker question: having

Suppose we have a relation with schema R(A, B, C, D, E). If we issue a query of the form:

```
SELECT ...
  FROM R
  WHERE ...
  GROUP BY B, E
  HAVING ???
```

Any aggregated term can appear in HAVING clause. An attribute not in the GROUP-BY list cannot be unaggregated in the HAVING clause. Thus, B or E may appear unaggregated, and all five attributes can appear in an aggregation. However, A, C, or D cannot appear alone.

Identify, in the list below, the term that CANNOT appear in the HAVING clause (where the ??? is).

A. A (u
B. B (unaggregated)
C. Count(B)
D. All can appear
E. None can appear

A cannot appear unaggregated

Grouping Examples (cont')

Find the major and age of the youngest student with age > 18, for each major with at least 2 students(of age > 18)

Student(snum,sname,major,standing,age)
Class(name,meets_at,room,fid)
Enrolled(snum,cname)
Faculty(fid,fname,deptid)

Grouping Examples (cont')

Find the major and age of the youngest student with age > 18, for each major with at least 2 students(of age > 18)

```
SELECT  S.major, MIN(S.age)
FROM    Student S
WHERE   S.Age > 18
GROUP BY  S.major
HAVING  COUNT(*) > 1
```

Grouping Examples (cont')

Find the age of the youngest student with age > 18, for each major for which the average age of the students who are >18 is higher than the average age of all students across all majors.

Grouping Examples (cont')

Find the age of the youngest student with age > 18, for each major for which the average age of the students who are >18 is higher than the average age of all students across all majors.

```
SELECT   S.major,  MIN(S.age), avg(S.age)
FROM     Student S
WHERE    S.age > 18
GROUP BY  S.major
HAVING   avg(S.age) > (SELECT avg(age)
                       FROM Student)
```

Note: avg(S.age) is included as a piece of information for your reference. The question doesn't indicate that you need to include this in the answer

Grouping Examples (cont')

Would this work?

```
SELECT S.major, MIN(S.age), avg(S.age)
FROM Student S, Student S2
WHERE S.age > 18 AND
                S.snum = S2.snum
GROUP BY S.major
HAVING avg(S.age) > avg(S2.age)
```

A. Yes

B. No

Grouping Examples (cont')

Would this work?

```
SELECT S.major, MIN(S.age), avg(S.age)
FROM Student S, Student S2
WHERE S.age > 18 AND
                S.snum = S2.snum
GROUP BY S.major
HAVING avg(S.age) > avg(S2.age)
```

A. Yes

B. No

Grouping Examples (cont')

Student table (some attributes omitted)

snum	major	age
1	CS	18
2	Music	20
3	Music	19
4	English	17
5	Business	21

Joining two instances of student and removing S.age <= 18

S.snum	S.major	S.age	S2.snum	S2.major	S2.age
~~1~~	~~CS~~	~~18~~	~~1~~	~~CS~~	~~18~~
2	Music	20	2	Music	20
3	Music	19	3	Music	19
~~4~~	~~English~~	~~17~~	~~4~~	~~English~~	~~17~~
5	Business	21	5	Business	21

Grouping Examples (cont')

Grouping by major (each group shown separately)

S.snum	S.major	S.age	S2.snum	S2.major	S2.age
2	Music	20	2	Music	20
3	Music	19	3	Music	19

S.snum	S.major	S.age	S2.snum	S2.major	S2.age
5	Business	21	5	Business	21

Taking the average age of S.age would be the same as taking the average age of S2.age.

Grouping Examples (cont')

Find those majors for which their average age is the minimum over all majors

```
SELECT major, avg(age)
FROM  student S
GROUP BY major
HAVING min(avg(age))
```

- WRONG, cannot use nested aggregation
  - One solution would be to use subquery in the FROM Clause

A bit ugly

```
SELECT  Temp.major, Temp.average
FROM(SELECT  S.major, AVG(S.age) as average
        FROM  Student S
        GROUP BY S.major) AS Temp
WHERE Temp.average in (SELECT  MIN(Temp.average) FROM  Temp)
```

Grouping Examples (cont')

Find those majors for which their average age is the minimum over all majors

~~SELECT major, avg(age)~~
~~FROM student S~~
~~GROUP BY major~~
~~HAVING min(avg(age))~~

- WRONG, cannot use nested aggregation
  - Another would be to use subquery with ALL in HAVING

```
SELECT major, avg(age)
FROM  student S
GROUP BY major
HAVING avg(age) <= all (SELECT  AVG(S.age)
                        FROM  Student S
                        GROUP BY S.major)
```

Easiest method would be to use Views

What are views

- Relations that are defined with a create table statement exist in the physical layer
  - do not change unless explicitly told so
- Virtual views do not physically exist, they are defined by expression over the tables.
  - Can be queries (most of the time) as if they were tables.

Why use views?

- Hide some data from users
- Make some queries easier
- Modularity of database
  - When not specified exactly based on tables.

Example: UBC has one table for students. Should the CS Department be able to update CS students info? Yes, Biology students? NO
Create a view for CS to only be able to update CS students

Defining and using Views

- Create View <view name><attributes in view> As <view definition>
  - View definition is defined in SQL
  - From now on we can use the view almost as if it is just a normal table
- View V (R_1,…R_n)
- query Q involving V
  - Conceptually
    - V (R_1,…R_n) is used to evaluate Q
  - In reality
    - The evaluation is performed over R_1,…R_n

Defining and using Views

- Example: Suppose tables
  Course(Course#,title,dept)
  Enrolled(Course#,sid,mark)

```
CREATE  VIEW  CourseWithFails(dept, course#, mark) AS
      SELECT   C.dept, C.course#, mark
      FROM     Course C, Enrolled E
      WHERE    C.course# = E.course# AND mark<50
```

This view gives the dept, course#, and marks for those courses where someone failed

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given CourseWithFails, but not Course or Enrolled, we can find the course in which some students failed, but we can't find the students who failed.

Course(Course#,title,dept)
Enrolled(Course#,sid,mark)
VIEW CourseWithFails(dept, course#, mark)

View Updates

- View updates must occur at the base tables.
  - Ambiguous
  - Difficult

- Example:

```
CREATE  VIEW  CourseWithFails(dept, course#, mark) AS
              SELECT   C.dept, C.course#, mark
              FROM     Course C, Enrolled E
              WHERE    C.course# = E.course# AND mark<50
```

- If you tried to delete a row from CourseWithFails, what does that mean? Do you want to delete the course info or the student info? How can you delete it such that other rows won't be affected?
- DBMS's restrict view updates only to some simple views on single tables (called updatable views)

View Deletes

- Drop View <view name>
  - Dropping a view does not affect any tuples of the in the underlying relation.

- How to handle DROP TABLE if there's a view on the table?
- DROP TABLE command has options to prevent a table from being dropped if views are defined on it:
  - DROP TABLE Student RESTRICT
    - drops the table, unless there is a view on it
  - DROP TABLE Student CASCADE
    - drops the table, and recursively drops any view referencing it

The Beauty of Views

Find those majors for which their average age is the minimum over all majors

With views:

```
Create View Temp(major, average) as
        SELECT      S.major, AVG(S.age) AS average
        FROM        Student S
        GROUP BY  S.major;

SELECT major, average
FROM Temp
WHERE average = (SELECT MIN(average) FROM Temp)
```

Without views:

```
   SELECT  Temp.major, Temp.average
FROM(SELECT  S.major, AVG(S.age) as average
      FROM  Student S
      GROUP BY S.major) AS Temp
WHERE Temp.average in (SELECT  MIN(Temp.average) FROM  Temp)
```

A bit ugly