CPSC 304 – March 15, 2018 Administrative Notes

- Office hours for me tomorrow are moved from 11-12 to 1-2
- Reminder: Midterm #2 regrades due next Tuesday before lecture
- Reminder: Tutorial is due this week

Before we go back to SQL

- Let's talk about Unix a minute
  - Basic file listing
  - Editing files – try pico if you want easy, emacs if you want powerful
  - Unix file permissions - https://www.linuxnix.com/chmod-command-explained-linuxunix/ 6

Now where were we...

- We'd just discussed GROUP BY and HAVING, two more clauses you can add in SQL
- GROUP BY allows you to write queries that form "groups" of tuples
- HAVING allows you to write per group qualification

Clicker question: grouping

- Compute the result of the query:

```
SELECT a1.x, a2.y, COUNT(*)
FROM   Arc a1, Arc a2
WHERE a1.y = a2.x
GROUP BY a1.x, a2.y
```

(think of Arc as being a flight, and the query as asking for how many ways you can take each 2 hop plane trip)
Which of the following is in the result?

A. (1,3,2)
B. (4,2,6)
C. (4,3,1)
D. All of the above
E. None of the above

x	y
1	2
1	2
2	3
3	4
3	4
4	1
4	1
4	1
4	2

clickergrouping.sql

Clicker question: grouping

- Compute the result of the query:

```
SELECT a1.x, a2.y, COUNT(*)
FROM   Arc a1, Arc a2
WHERE a1.y = a2.x
GROUP BY a1.x, a2.y
```

x	y	COUNT(*)
1	3	2
2	4	2
3	1	6
3	2	2
4	2	6
4	3	1

x	y
1	2
1	2
2	3
3	4
3	4
4	1
4	1
4	1
4	2

A. (1,3,2) (1,2)(2,3), (1,2)(2,3)

B. (4,2,6) 3 ways to do (4,1) and two ways to do (1,2)

C. (4,3,1) (4,2)(2,3)

D. All of the above Correct

E. None of the above

Clicker question: grouping

- Compute the result of the query:

```
SELECT a1.x, a2.y, COUNT(*)
FROM   Arc a1, Arc a2
WHERE a1.y = a2.x
GROUP BY a1.x, a2.y
```

(The query asks for how many ways you can take each 2 hop plane trip. 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:

origin	dest
SFO	YVR
SFO	YVR
YVR	SEA
SEA	PIT
SEA	PIT
PIT	SFO
PIT	SFO
PIT	SFO
PIT	YVR

clickergrouping2.sql

Clicker question: grouping

- Compute the result of the query:

```
SELECT a1.x, a2.y, COUNT(*)
FROM   Arc a1, Arc a2
WHERE a1.y = a2.x
GROUP BY a1.x, a2.y
```

(The query asks for how many ways you can take each 2 hop plane trip. 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

FLIGHT:

origin	dest
SFO	YVR
SFO	YVR
YVR	SEA
SEA	PIT
SEA	PIT
PIT	SFO
PIT	SFO
PIT	SFO
PIT	YVR

Groupies of your very own

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

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

Groupies of your very own

- Find the 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 an 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 is correct?

A: just 1

B: just 2

C: both Correct

D: neither

Grouping Examples (cont')

For each standing, find the 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 ???
```

What terms can appear in the HAVING condition (represented by ??? in the above query)? Identify, in the list below, the term that CANNOT appear.

A. A
B. B
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.

What terms can appear in the HAVING condition (represented by ??? in the above query)? Identify, in the list below, the term that CANNOT appear.

A. A — A cannot appear unaggregated
B. B
C. Count(B)
D. All can appear
E. None can appear

Grouping Examples (cont')

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

Grouping Examples (cont')

Find the 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 their average age 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 their average age is higher than the average age of all students across all majors.

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

Grouping Examples (cont')

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

Grouping Examples (cont')

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

```
SELECT  S.major,  MIN(S.age)
FROM    Student S
WHERE   S.age > 18
GROUP BY S.major
HAVING  1  <  (SELECT  COUNT(*)
               FROM  Student S2
               WHERE  S.major=S2.major)
```

- Subqueries in the HAVING clause can be correlated with fields from the outer query.

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

```
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)
```

Hideously ugly
Not supported in all systems

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.

Defining and using Views

- Create View <view name> 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

CourseWithFails(dept, course#, mark)

Course(Course#,title,dept)

Enrolled(Course#,sid,mark)

- DBMS's restrict view updates only to some simple views on single tables (called updatable views)

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

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)
```

Hideously ugly

Clicker question: views

Suppose relation R(a,b,c):

Define the view *V* by:

```
CREATE VIEW V AS
SELECT a+b AS d, c
FROM R;
```

What is the result of the query:

```
SELECT d, SUM(c)
FROM V
GROUP BY d
HAVING COUNT(*) <> 1;
```

a	b	c
1	1	3
1	2	3
2	1	4
2	3	5
2	4	1
3	2	4
3	3	6

Identify, from the list below, a tuple in the result of the query:

A. (2,3)
B. (3,12)
C. (5,9)
D. All are correct
E. None are correct

Clickerview.sql

Clicker question: views

Suppose relation R(a,b,c):

Define the view *V* by:

```
CREATE VIEW V AS
SELECT a+b AS d, c
FROM R;
```

What is the result of the query:

```
SELECT d, SUM(c)
FROM V
GROUP BY d
HAVING COUNT(*) <> 1;
```

a	b	c
1	1	3
1	2	3
2	1	4
2	3	5
2	4	1
3	2	4
3	3	6

V

d	c
2	3
3	3
3	4
5	5
6	1
5	4
6	6

d	Sum(C)
3	7
5	9
6	7

Identify, from the list below, a tuple in the result of the query:

A. (2,3) Wrong. In view

B. (3,12)

C. (5,9) Right

D. All are correct

E. None are correct

Null Values

- Tuples may have a null value, denoted by *null*, for some of their attributes
- Value *null* signifies an unknown value or that a value does not exist.
- The predicate **IS NULL** (**IS NOT NULL**) can be used to check for null values.
  - E.g. *Find all student names whose age is not known.*

```
SELECT name
FROM Student
WHERE age IS NULL
```

- The result of any arithmetic expression involving *null* is *null*
  - E.g. 5 + *null* returns *null.*

Null Values and Three Valued Logic

- null requires a 3-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*, (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: *(true* **and** *unknown)* = *unknown, (false* **and** *unknown)* = *false,*
    (unknown **and** *unknown)* = *unknown*
  - NOT*: (***not** *unknown)* = *unknown*
  - "*P* **is unknown"** evaluates to true if predicate *P* evaluates to *unknown*
- Any comparison with *null* returns *unknown*
  - *E.g. 5 < null or null <> null or null = null*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.

```
SELECT count(*)          SELECT count(fid)
FROM class               FROM class
```

Clicker null query

Determine the result of:

```
SELECT COUNT(*),
    COUNT(Runs)
FROM Scores
WHERE Team = 'Carp'
```

Which of the following is in the result:

A. (1,0)
B. (2,0)
C. (1,NULL)
D. All of the above
E. None of the above

Scores:

Team	Day	Opponent	Runs
Dragons	Sun	Swallows	4
Tigers	Sun	Bay Stars	9
Carp	Sun	NULL	NULL
Swallows	Sun	Dragons	7
Bay Stars	Sun	Tigers	2
Giants	Sun	NULL	NULL
Dragons	Mon	Carp	NULL
Tigers	Mon	NULL	NULL
Carp	Mon	Dragons	NULL
Swallows	Mon	Giants	0
Bay Stars	Mon	NULL	NULL
Giants	Mon	Swallows	5

Clicker null query

Start clickernull.sql

Determine the result of:

```
SELECT COUNT(*),
   COUNT(Runs)
FROM Scores
WHERE Team = 'Carp'
```

Which of the following is in the result:

A. (1,0)
B. (2,0) Right
C. (1,NULL)
D. All of the above
E. None of the above

Scores:

Team	Day	Opponent	Runs
Dragons	Sun	Swallows	4
Tigers	Sun	Bay Stars	9
Carp	Sun	NULL	NULL
Swallows	Sun	Dragons	7
Bay Stars	Sun	Tigers	2
Giants	Sun	NULL	NULL
Dragons	Mon	Carp	NULL
Tigers	Mon	NULL	NULL
Carp	Mon	Dragons	NULL
Swallows	Mon	Giants	0
Bay Stars	Mon	NULL	NULL
Giants	Mon	Swallows	5

Natural Join

- The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with same name of associate tables will appear once only.
- Natural Join : Guidelines
  - The associated tables have one or more pairs of identically named columns.
  - The columns must be the same data type.
  - Don’t use ON clause in a natural join.

```
SELECT *
FROM student s natural join enrolled e
```

- Natural join of tables with no pairs of identically named columns will return the cross product of the two tables.

```
SELECT *
FROM student s natural join class c
```