CPSC 304 – Administrative notes October 25 & October 29, 2024

- Midterm is back!
  - See Piazza – in general you all did quite well
- Project:
  - Milestone 3: Project check in – due October 25
    - Sign up 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!
  - Look out for conflict form to come soon!

Now where were we…

- SQL!
- When in doubt, start with
  SELECT
  FROM
  WHERE

Set Operations: EXCEPT/MINUS

- Find the sids of all students who took Operating System Design but did not take Database Systems

```
SELECT snum
FROM enrolled
WHERE cname = 'Operating System Design'
EXCEPT  ← Oracle uses MINUS rather than EXCEPT
SELECT snum
FROM enrolled
WHERE cname = 'Database Systems'
```

Can we do it in a different way?
(We'll come back to this)

But what about…

- Select the IDs of all students who have not taken “Operating System Design”
  - One way to do is to find all students that taken “Operating System Design”.
  - Do all students MINUS those who have taken “Operating System Design”

```
SELECT snum
FROM student
EXCEPT  ← Oracle uses MINUS rather than EXCEPT
SELECT snum
FROM enrolled
WHERE cname = ‘Operating System Design'
```

Motivating Example for Nested Queries

Find ids and names of male stars who have been in movie with ID 248:

Motivating Example for Nested Queries

Find ids and names of male stars who have been in movie with ID 248:

```
SELECT  M.StarID, name
FROM    MovieStar M, StarsIn S
WHERE   M.StarID = S.starID AND S.MovieID = 248
AND gender = ‘male’;
```

- *Find ids and names of male stars who have been in some movie but not been in movie w/ ID 248 w/o using EXCEPT/MINUS:*

Would the following be correct?

~~SELECT DISTINCT M.StarID, name~~
~~FROM   MovieStar M, StarsIn S~~
~~WHERE   M.StarID = S.starID AND S.MovieID <> 248~~
~~and gender = ‘male’;~~

Nested Queries

- A very powerful feature of SQL:

```
SELECT A1,A2,…,An
FROM   R1,R2, …,Rm
WHERE  condition
```

- A nested query is a query that has another query embedded with it.
  - A **SELECT, FROM, WHERE, or HAVING** clause can itself contain an SQL query!
  - Being part of the WHERE clause is the most common

Nested Queries (IN/Not IN)

Find ids and names of male stars who have been in movie with ID 248:

Nested Queries (IN/Not IN)

Find ids and names of male stars who have been in movie with ID 248:

```
SELECT  DISTINCT M.StarID, M.Name
FROM    MovieStar M                 There's also NOT IN
WHERE   M.Gender =  'male' AND
        M.StarID IN  (SELECT  S.StarID
                      FROM  StarsIn S
                      WHERE  MovieID=248)
```

- To find stars who have *not* been in movie 248, use **NOT IN**.
- To understand nested query semantics, think of a <u>*nested loops*</u> evaluation:
  - *For each MovieStar tuple, check the qualification by computing the subquery.*

Nested Queries (IN/Not IN)

Find ids and names of male stars who have been in movie with ID 248:

```
SELECT  DISTINCT M.StarID, M.Name
FROM    MovieStar M
WHERE   M.Gender =  `male` AND
        M.StarID IN  (SELECT  S.StarID
                      FROM  StarsIn S
                      WHERE  MovieID=28)
```

- In this example in inner query does not depend on the outer query so it could be computed just once.
- Think of this as a function that has no parameters.

```
SELECT  S.StarID
FROM  StarsIn S
WHERE  MovieID=248
```

StarID
1245
1246

```
SELECT  M.StarID, M.Name
FROM    MovieStar M
WHERE   M.Gender = `male` AND
        M.StarID IN
        (1245,1246)
```

Rewriting EXCEPT Queries Using In

- Using nested queries, find the snums of all students who took Operating System Design but did not take Database Systems

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

Rewriting EXCEPT Queries Using In

- Using nested queries, find the snums of all students who took Operating System Design but did not take Database Systems

```
SELECT snum
FROM enrolled
WHERE cname = 'Operating System Design' and snum not in
          (SELECT snum
           FROM enrolled
           WHERE cname = 'Database Systems')
```

Rewriting INTERSECT Queries Using IN

Find IDs of stars who have been in movies in 1944 and 1974

Rewriting INTERSECT Queries Using IN

Find IDs of stars who have been in movies in 1944 and 1974

```
SELECT DISTINCT S.StarID
FROM    Movie M, StarsIn S
WHERE  M.MovieID = S.MovieID AND M.year = 1944 AND
 S.StarID IN (SELECT  S2.StarID
                    FROM Movie M2, StarsIn S2
                    WHERE   M2.MovieID = S2.MovieID AND M2.year = 1974)
```

The subquery finds stars who have been in movies in 1974

Nested Queries with Correlation Same idea, subtle difference

Find names of stars who have been in movie w/ ID 248:

```
SELECT  DISTINCT M.Name
FROM    MovieStar M
WHERE   EXISTS  (SELECT  *
                 FROM  StarsIn S
                 WHERE  MovieID=248 AND S.StarID = M.StarID)
```

- **EXISTS**: *returns true if the set is not empty*.
- **UNIQUE**: *returns true if there are no duplicates*.
- Illustrates why, in general, subquery must be re-computed for each StarsIn tuple.

SQL EXISTS Condition

- The SQL EXISTS condition is used in combination with a subquery and is considered to be met, if the subquery returns at least one row. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement.
- We can also use NOT EXISTS

SQL EXISTS Condition

- Using the EXISTS/ NOT EXISTS operations and correlated queries, find the name and age of the oldest student(s)

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

SQL EXISTS Condition

- Using the EXISTS/ NOT EXISTS operations and correlated queries, find the name and age of the oldest student(s)

```
SELECT sname, age
FROM student s2
WHERE NOT EXISTS(SELECT *
                 FROM student s1
                 WHERE s1.age >s2.age)
```

SQL EXISTS

```
SELECT sname, age
FROM student s2
WHERE NOT EXISTS(SELECT *
                 FROM student s1
                 WHERE s1.age >s2.age)
```

Does there exist a tuple in s1 such that the age of the s1 tuple is greater than the age of the tuple in s2?

Student s2

snum	Name	…
…	…	…
…	…	…
…	…	…

Student s1

snum	Name	…
…	…	…
…	…	…
…	…	…

More on Set-Comparison Operators

- We've already seen **IN and EXISTS**. Can also use **NOT IN**, **NOT EXISTS**.
- Also available: **op ANY**, **op ALL,**
  where op is one of: **>**, **<**, **=**, **<=**, **>=**, **<>**
- Find movies made after "Fargo"

More on Set-Comparison Operators

- We’ve already seen **IN and EXISTS**. Can also use **NOT IN**, **NOT EXISTS**.
- Also available: **op ANY**, **op ALL**, where op is one of: **>**, **<**, **=**, **<=**, **>=**, **<>**
- Find movies made after “Fargo”

```
SELECT   *
FROM     Movie
WHERE   year > ANY  (SELECT   year
                      FROM    Movie
                      WHERE   Title ='Fargo')
```

Just returning one column

If we have multiple movies named Fargo then we can use ALL instead of ANY

Clicker nested question

Determine the result of:

```
SELECT Team, Day
FROM Scores S1
WHERE Runs <= ALL
    (SELECT Runs
    FROM Scores S2
    WHERE S1.Day = S2.Day )
```

Which of the following is in the result:

A. (Carp, Sun)
B. (Bay Stars, Sun)
C. (Swallows, Mon)
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	Giants	2
Swallows	Sun	Dragons	7
Bay Stars	Sun	Tigers	2
Giants	Sun	Carp	4
Dragons	Mon	Carp	6
Tigers	Mon	Bay Stars	5
Carp	Mon	Dragons	3
Swallows	Mon	Giants	0
Bay Stars	Mon	Tigers	7
Giants	Mon	Swallows	5

Clicker nested question

Clickernested.sql

Determine the result of:

```
SELECT Team, Day
FROM Scores S1
WHERE Runs <= ALL
    (SELECT Runs
    FROM Scores S2
    WHERE S1.Day = S2.Day )
```

Which of the following is in the result:

A. (Carp, Sun)

B. (Bay Stars, Sun)

C. (Swallows, Mon)

D. All of the above Correct

E. None of the above

Scores:

Team	Day	Opponent	Runs
Dragons	Sun	Swallows	4
Tigers	Sun	Bay Stars	9
Carp	Sun	Giants	2
Swallows	Sun	Dragons	7
Bay Stars	Sun	Tigers	2
Giants	Sun	Carp	4
Dragons	Mon	Carp	6
Tigers	Mon	Bay Stars	5
Carp	Mon	Dragons	3
Swallows	Mon	Giants	0
Bay Stars	Mon	Tigers	7
Giants	Mon	Swallows	5

Team/Day pairs such that the team scored the minimum number of runs for that day.

Example

- Using the any or all operations, find the name and age of the oldest student(s)

Example

- Using the any or all operations, find the name and age of the oldest student(s)

```
SELECT sname, age
FROM student s2
WHERE NOT EXISTS(SELECT *
                 FROM student s1
                 WHERE s1.age >s2.age)
```

You can rewrite queries that use any or all with queries that use exist or not exist

```
SELECT sname, age
FROM student s2
WHERE s2.age >= all (SELECT age
                     FROM student s1)
```

Clicker Question

- Consider the following SQL query

```
SELECT DISTINCT s1.sname, s1.age
FROM student s1, student s2
WHERE s1.age > s2.age
```

- This query returns
- A: The name and age of one of the oldest student(s)
- B: The name and age of all of the oldest student(s)
- C: The name and age of all of the youngest student(s)
- D: The name and age of all students that are older than the youngest student(s)
- E: None of the above

Clicker Question

- Consider the following SQL query

```
SELECT DISTINCT s1.sname, s1.age
FROM student s1, student s2
WHERE s1.age > s2.age
```

- This query returns
- A: The name and age of one of the oldest student(s)
- B: The name and age of all of the oldest student(s)
- C: The name and age of all of the youngest student(s)
- D: The name and age of all students who are older than the youngest student(s)
- E: None of the above

Reminder: Division A/B

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

sno
s1
s4

A/B3

sno
s1

(method 1)

Division in SQL

Find students who've taken all classes.

```
SELECT sname
FROM   Student S
WHERE NOT EXISTS
            ((SELECT  C.name
               FROM  Class C)        All classes
               EXCEPT
              (SELECT  E.cname
Classes         FROM  Enrolled E
taken by S      WHERE E.snum=S.snum))
```

The hard way (without EXCEPT): (method 2)

```
SELECT sname
FROM    Student S
WHERE NOT EXISTS (SELECT  C.name
                  FROM  Class C
                  WHERE  NOT EXISTS (SELECT  E.snum
                                     FROM  Enrolled E
                                     WHERE  C.name=E.cname
                                     AND E.snum=S.snum))
```

Method 2:

select Student S such that ...

there is no Class C…

which is not taken by S

Division in SQL (method 1- use EXCEPT)

Find students who've taken all classes.

```
SELECT sname
FROM   Student S
WHERE NOT EXISTS
          ((SELECT  C.name      All classes
            FROM  Class C)
            EXCEPT
             (SELECT  E.cname
              FROM  Enrolled E
              WHERE E.snum=S.snum))
```

Division in SQL (method 1- use EXCEPT)

Find students who've taken all classes.

```
SELECT sname
FROM   Student S
WHERE NOT EXISTS
             ((SELECT  C.name
               FROM  Class C)
               EXCEPT
                (SELECT  E.cname
                 FROM  Enrolled E
                 WHERE E.snum=S.snum))
```

All classes taken by S

Division in SQL (method 1- use EXCEPT)

Find students who've taken all classes.

```
SELECT sname
FROM   Student S
WHERE NOT EXISTS
          ((SELECT  C.name
            FROM  Class C)
            EXCEPT
             (SELECT  E.cname
              FROM  Enrolled E
              WHERE E.snum=S.snum))
```

All classes that have not been taken by S

Division in SQL (method 1- use EXCEPT)

Find students who've taken all classes.

```
SELECT sname
FROM   Student S
WHERE NOT EXISTS
          ((SELECT  C.name
            FROM  Class C)
            EXCEPT
             (SELECT  E.cname
              FROM  Enrolled E
              WHERE E.snum=S.snum))
```

Only true if there is no class that has not been taken by S (i.e., S must have taken all the classes)

Division in SQL (method 2- without using EXCEPT)

Find students who've taken all classes.

```
SELECT sname
FROM    Student S
WHERE NOT EXISTS (SELECT  C.name
                  FROM  Class C
                  WHERE  NOT EXISTS (SELECT  E.snum
                                     FROM  Enrolled E
                                     WHERE  C.name=E.cname
                                     AND E.snum=S.snum))
```

Returns a result if student S is enrolled in class C

Division in SQL (method 2- without using EXCEPT)

Find students who've taken all classes.

```
SELECT sname
FROM    Student S
WHERE NOT EXISTS (SELECT  C.name
                  FROM  Class C
                  WHERE  NOT EXISTS (SELECT  E.snum
                                     FROM  Enrolled E
                                     WHERE  C.name=E.cname
                                     AND E.snum=S.snum))
```

Only true if student S has never been enrolled in class C.

Division in SQL (method 2- without using EXCEPT)

Find students who've taken all classes.

```
SELECT sname
FROM    Student S
WHERE NOT EXISTS (SELECT  C.name
                  FROM  Class C
                  WHERE  NOT EXISTS (SELECT  E.snum
                                     FROM  Enrolled E
                                     WHERE  C.name=E.cname
                                     AND E.snum=S.snum))
```

Find the classes that student S has not enrolled in.

Division in SQL (method 2- without using EXCEPT)

Find students who’ve taken all classes.

```
SELECT sname
FROM    Student S
WHERE NOT EXISTS (SELECT  C.name
                  FROM  Class C
                  WHERE  NOT EXISTS (SELECT  E.snum
                                     FROM  Enrolled E
                                     WHERE  C.name=E.cname
                                     AND E.snum=S.snum))
```

Only true if there is no class that student S has never been enrolled in (i.e., student S has been enrolled in all the classes).

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.