Administrative Notes – February 9, 2023

- Feb 17: Assignment 3 (pairs if you want) due
  - Make sure you sign up in pairs beforehand as we need time to configure Canvas to allow both partners to view the submission and subsequent feedback
    - Everyone who has signed up as of last night has been put into a group on Canvas

- Feb 20 – 24: Reading Break! (yay!)
  - No lectures, tutorials, or office hours during this week

Motivating Example for Nested Queries

Find ids and names of female stars who have been in movie with ID 28:

```
SELECT M.StarID, name
FROM MovieStar M, StarsIn S
WHERE M.StarID = S.starID AND S.MovieID = 28
      AND gender = 'female';
```

Find ids and names of female stars who have not been in movie w/ ID 28 w/o using EXCEPT/MINUS:

- Would the following be correct?

```
SELECT M.StarID, name
FROM MovieStar M, StarsIn S
WHERE M.StarID = S.starID AND S.MovieID <> 28
      AND gender = 'female';
```

Motivating Example for Nested Queries

MovieStar

StarID	Name	Gender
1	Jessica Wong	Female
2	Jia Lu	Male
3	Carol Huang	Female

StarsIn

MovieID	StarID	Character
28	1	A tree
28	3	Background
29	3	A tired grad student

Does "WHERE StarsIn.MovieID <> 28" correctly remove Carol from the results?

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 stars who have been in movie with ID 28:

Nested Queries (IN/Not IN)

Find ids and names of stars who have been in movie with ID 28:

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

There's also NOT IN

- To find stars who have *not* been in movie 28, use **NOT IN**.
- To understand nested query semantics, think of a *nested loops* evaluation:
  - *For each MovieStar tuple, check the qualification by computing the subquery.*

Nested Queries (IN/Not IN)

Find ids and names of stars who have been in movie with ID 28:

```
SELECT M.StarID, M.Name
FROM MovieStar M
WHERE M.Gender =  'female' 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=28
```

StarID
1026
1027

```
SELECT M.StarID, M.Name
FROM MovieStar M
WHERE M.Gender = 'female' AND
      M.StarID IN (1026,1027)
```

Rewriting EXCEPT Queries Using In

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

Rewriting EXCEPT Queries Using In

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

```
SELECT snum
FROM enrolled e1
WHERE e1.cname = 'Operating System Design'
       AND snum NOT IN
       (SELECT snum
        FROM enrolled e2
        WHERE e2.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 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 28:

```
SELECT M.Name
FROM MovieStar M
WHERE EXISTS  (SELECT *
               FROM  StarsIn S
               WHERE  MovieID=28 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).

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)
```

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?

SQL EXISTS Condition

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

Student s2

snum	Name	…
…	…	…
…	…	…
…	…	…

Student s1

snum	Name	…
…	…	…
…	…	…
…	…	…

In-Class Exercise (SQL 2)

- Canvas → Modules → In Class Exercises
- You can work on it with other people around you. If you work with others, you must **write their names on your submission to acknowledge the collaboration**.
  - Everyone must submit to Canvas
- Reminder: no late submissions accepted

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: **>**, **<**, **=**, **<=**, **>=**, **<>**

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 Question

```
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 Question

Clickernested.sql

```
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

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

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

Example

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

Example

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

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)
```

```
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 that are older than the youngest student(s)

E. None of the above

Division

- Useful for expressing queries that include a notion of "**for all**" or "**for every**"

- E.g., *Find movie stars who were in **all** movies*.

Examples for Division

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

Examples for Division

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

Examples for Division

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

Examples for Division

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

Division in SQL

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))
```

(method 1)

The hard way (without EXCEPT):

```
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)

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

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).

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))
```

Select Student S such that there is no class C which is not taken by S.