# Introduction to Data Management
# DSC 100

## Nested SQL Queries

# What have we learned so far

SQL features

- Projections
- Selections
- Joins (inner and outer)
- Aggregates
- Group by
- Having
- Inserts, updates, and deletes

# Subqueries

- Subquery: A query that is part of another
- Nested Query: A query that has an embedded subquery
- A subquery can be a nested query itself!
- Why? Sometimes we need to express a condition that refers to a table that must itself be computed
- A subquery may occur in:
  - A `SELECT` clause
  - A `FROM` clause
  - A `WHERE` clause

Often appear here

- Rule of thumb: avoid nested queries when possible
  - But sometimes it's impossible, as we will see

# Subqueries

- Can return a single value to be included in a `SELECT` clause
- Can return a relation to be included in the `FROM` clause
- Can return a single value to be compared with another value in a `WHERE` or `HAVING` clause
- Can return a relation to be used in the `WHERE`

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 1. Subqueries in SELECT

For each actor return the genre of movie they acted in

```
SELECT a.actorname, (SELECT genre
                     FROM   Movie m
                     WHERE  m.name=a.moviename) as genre
FROM  ActedIn a
```

"correlated subquery"

What happens if the subquery returns more than one genre?

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 1. Subqueries in SELECT

Whenever possible, don't use a nested queries:

```
SELECT a.actorname, (SELECT genre
                     FROM   Movie m
                     WHERE  m.name=a.moviename) as genre
FROM  ActedIn a
```

=

Subquery unnesting

```
SELECT a.actorname, genre
FROM   ActedIN a, Movie m
WHERE  m.name=a.moviename
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 1. Subqueries in SELECT

Compute average salary of actors for all movies with rating > 9

```
SELECT DISTINCT m.name, (SELECT AVG(salary)
                         FROM   ActedIn a
                         WHERE  m.name=a.moviename) as salary
FROM Movie m
WHERE m.rating > 9
```

Subquery unnesting

```
SELECT m.name, AVG(salary)
FROM   Movie m, ActedIn a
WHERE  m.name=a.moviename
AND m.rating > 9
GROUP BY m.name
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 1. Subqueries in SELECT

```
SELECT DISTINCT m.name, (SELECT count(*)
                         FROM ActedIN a
                         WHERE m.name=a.moviename) as anum
FROM Movie m
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 1. Subqueries in SELECT

Compute the number of actors in each movie

```
SELECT DISTINCT m.name, (SELECT count(*)
                         FROM ActedIN a
                         WHERE m.name=a.moviename) as anum
FROM Movie m
```

!!

Subquery unnesting

```
SELECT m.name, count(*)
FROM   ActedIn a, Movie m
WHERE  m.name=a.moviename
GROUP BY m.name
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 1. Subqueries in SELECT

But are these equivalent?

```
SELECT DISTINCT m.name, (SELECT AVG(salary)
                         FROM   ActedIn a
                         WHERE  m.name=a.moviename) as salary
FROM Movie m
WHERE m.rating > 9
```

```
SELECT m.name, AVG(salary)
FROM   Movie m, ActedIn a
WHERE  m.name=a.moviename AND
m.rating > 9
GROUP BY m.name
```

≠

=

```
SELECT m.name, AVG(salary)
FROM   Movie m LEFT OUTER JOIN ActedIn a  ON
m.name=a.moviename
WHERE  m.rating > 9
GROUP BY m.name
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 2. Subqueries in FROM

Find all Movie with rating > 8 and < 9

"Not a correlated subquery"

```
SELECT x.name, rating
FROM (SELECT *
      FROM Movie AS m
      WHERE rating > 8) as x
WHERE x.rating < 9
```

A subquery whose result we called myTable

```
WITH myTable AS (SELECT * FROM Movie AS m WHERE rating > 8)
  SELECT x.name, x.rating
  FROM myTable as X
  WHERE x.rating < 9
```

Sub-query refactoring

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 2. Subqueries in FROM

Find all Movie with rating > 8 and < 9

```
SELECT x.name
FROM (SELECT *
      FROM Movie AS m
      WHERE rating > 8) as x
WHERE x.rating < 9
```

=

```
SELECT m.name, rating
FROM   Movie m
WHERE  m.rating < 9 AND m.rating > 8
```

Subquery unnesting

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Find the name of actors who have acted in some Sci-Fi movie

Quantifier is a logical operator that specifies how many elements in the domain of discourse satisfy a property

Existential quantifiers

"there exists", "there is at least one", or "for some"

TRUE iff the subquery returns one or more records

Using EXISTS:

```
SELECT DISTINCT  a.actorname
FROM    ActedIn a
WHERE   EXISTS (SELECT m.name
                FROM   Movie m
                WHERE  m.name=a.moviename AND
                       m.genre='Sci-Fi')
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Find the name of actors who have acted in some Sci-Fi movie

Existential quantifiers

Allow us to test set membership

Using IN:

```
SELECT DISTINCT  a.actorname
FROM    ActedIn a
WHERE   a.moviename IN (SELECT m.name
                        FROM   Movie m
                        WHERE  m.name=a.moviename AND
                               m.genre='Sci-Fi')
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Find the name of actors who have acted in some Sci-Fi movie

Existential quantifiers

```
SELECT DISTINCT  a.actorname
FROM    ActedIn a
WHERE   a.moviename IN (SELECT m.name
                        FROM   Movie m
                        WHERE  m.name=a.moviename AND
                               m.genre='Sci-Fi')
```

Subquery unnesting

```
SELECT DISTINCT a.actorname
FROM    Movie m, ActedIn a
WHERE   m.name=a.moviename AND m.genre='Sci-Fi'
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Find the name of actors who have acted in some non-Sci-Fi movie

Existential quantifiers

```
SELECT DISTINCT  a.actorname
FROM    ActedIn a
WHERE   a.moviename NOT IN (SELECT m.name
                            FROM    Movie m
                            WHERE   m.name=a.moviename AND
                                    m.genre='Sci-Fi')
```

Subquery unnesting

```
SELECT DISTINCT a.actorname
FROM    Movie m, ActedIn a
WHERE   m.name=a.moviename AND m.genre ≠'Sci-Fi'
```

Existential quantifiers are straightforward!☺

Join queries essentially evaluate the presence of existential quantifiers


Universal quantifiers pose a challenge ☹

The SQL constructs that we have covered thus far do not have the capability to handle universal quantifiers.

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Retrieve all actor names that only acted on action movies

Same as: Every movie they acted on was an action movie

**Universal quantifiers**

"given any", "for all", "for every"

Step 1: Find all actor names that acted on some non-action movie

```
SELECT a.actorname
FROM   Movie m, ActedIn a
WHERE  m.name=a.moviename AND m.genre ≠ 'Action'
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Retrieve all actor names that only acted on action movies

Same as: every movies they acted on were action

Universal quantifiers

Step 1: Find all actor names that acted on some non-action movie

Step 2: Retrieve all the others (i.e., those that do not satisfy the result of Step 1)

```
SELECT a.actorname
FROM   Movie m, ActedIn a
WHERE  m.name=a.moviename AND m.genre ≠ 'Action'
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Retrieve all actor names that only acted on action movies

Same as: every movies they acted on were action

Universal quantifiers

Step 1: Find all actor names that acted on some non-action movie

Step 2: Retrieve all the others (i.e., those that do not satisfy the result of Step 1)

```
SELECT a.name
FROM   ActedIn a
WHERE  a.actorname NOT IN ( SELECT a.actorname
                            FROM   Movie m, ActedIn a
                            WHERE  m.name=a.moviename AND m.genre ≠ 'Action')
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Retrieve all actor names that acted on at most two action movies

```
SELECT DISTINCT  a.actorname
FROM   ActedIn a
WHERE  2 >= (SELECT count(*)
             FROM   Movie m, ActedIN b
             WHERE  m.name=b.moviename AND
                    b.actorname=a.actorname   AND
                       m.genre='Action')
```

What does this query do?

```
SELECT DISTINCT  a.actorname
FROM   ActedIn a
WHERE  0 > (SELECT count(*)
            FROM   Movie m, ActedIN b
            WHERE  m.name=b.moviename AND
                   b.actorname=a.actorname   AND
                      m.genre='Action')
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

What does this query do?

```
SELECT a.actorname
FROM ActedIn a
JOIN Movie m ON a.moviename = m.name
WHERE m.genre = `Action’
GROUP BY a.actorname
HAVING count(*) >= 2
```

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Find all movie s.t. all their actors' salaries > $100K

Universal quantifiers

```
SELECT   m.name
FROM     movie m
WHERE    $100K < ALL (SELECT a.salary
                      FROM   ActedIn a
                      WHERE  m.name=a.moviename)
```

Not supported in sqlite

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Find all movie s.t. all their actors' salaries > $100K

Universal quantifiers

```
SELECT   m.name
FROM     movie m
WHERE    $100K < ALL (SELECT a.salary
                      FROM   ActedIn a
                      WHERE  m.name=a.moviename)
```

Not supported in sqlite

Movie(name, year, genre, budget, revenue , rating)
ActedIN(actorname, moviename, salary)

# 3. Subqueries in WHERE

Find all movie s.t. all their actors' salaries > $100K

Universal quantifiers

```
SELECT m.name
FROM movie m
WHERE NOT EXISTS
(SELECT 1 FROM ActedIn a WHERE m.name = a.moviename
AND a.salary <= $100k)
```

# Unnesting Universal Quantifiers

- Is it possible to unnest the *universal quantifier* query?

Definition: A query Q is monotone if:

- Whenever we add tuples to one or more input tables, the answer to the query will not lose any output tuple

```
SELECT a.actorname
FROM   Movie m, ActedIn a
WHERE  m.name=a.moviename AND m.gendre='Crime'
```

Is this monotone?

Movie(name, year, genre)
ActedIN(actorname, moviename)

# Monotone Queries

| name | year | genre |
|---|---|---|
| Apocalypse now | 1979 | War |
| The god father | 1972 | Crime |
| Planet Earth II | 2016 | Nature documentary |
| Jack and Jill | 2011 | Comedy |

| actorname | moviename |
|---|---|
| Marlon Brando | Apocalypse now |
| Al Pacino | The god father |
| Marlon Brando | The god father |

```
SELECT a.actorname
FROM   Movie m, ActedIn a
WHERE  m.name=a.moviename AND m.gendre='Crime'
```

| actorname |
|---|
| Marlon Brando |
| Al Pacino |

Movie(name, year, genre)
ActedIN(actorname, moviename)

# Monotone Queries

| name | year | genre |
|---|---|---|
| Apocalypse now | 1979 | War |
| The god father | 1972 | Crime |
| Planet Earth II | 2016 | Nature documentary |
| Jack and Jill | 2011 | Comedy |

| actorname | moviename |
|---|---|
| Marlon Brando | Apocalypse now |
| Al Pacino | The god father |
| Marlon Brando | The god father |
| Al Pacino | Jack and Jill |

```
SELECT a.actorname
FROM   Movie m, ActedIn a
WHERE  m.name=a.moviename AND m.gendre='Crime'
```

Monotone

| actorname |
|---|
| Marlon Brando |
| Al Pacino |

Movie(name, year, genre)
ActedIN(actorname, moviename)

# Monotone Queries

| name | year | genre |
|---|---|---|
| Apocalypse now | 1979 | War |
| The god father | 1972 | Crime |
| Planet Earth II | 2016 | Nature documentary |
| Jack and Jill | 2011 | Comedy |

| actorname | moviename |
|---|---|
| Marlon Brando | Apocalypse now |
| Al Pacino | The god father |
| Marlon Brando | The god father |

```
SELECT a.actorname
FROM   ActedIn a
WHERE  a.actorname
NOT IN ( SELECT a.actorname
         FROM   Movie m, ActedIn a
         WHERE  m.name=a.moviename AND m.genre ≠ 'Crime')
```

| actorname |
|---|
| Al Pacino |

Movie(name, year, genre)
ActedIN(actorname, moviename)

# Monotone Queries

| name | year | genre |
|---|---|---|
| Apocalypse now | 1979 | War |
| The god father | 1972 | Crime |
| Planet Earth II | 2016 | Nature documentary |
| Jack and Jill | 2011 | Comedy |

| actorname | moviename |
|---|---|
| Marlon Brando | Apocalypse now |
| Al Pacino | The god father |
| Marlon Brando | The god father |
| Al Pacino | Jack and Jill |

```
SELECT a.actorname
FROM   ActedIn a
WHERE  a.actorname
NOT IN ( SELECT a.actorname
         FROM   Movie m, ActedIn a
         WHERE  m.name=a.moviename AND m.genre ≠ 'Crime')
```

| actorname |
|---|
| Al Pacino |

Movie(name, year, genre)
ActedIN(actorname, moviename)

# Monotone Queries

| name | year | genre |
|---|---|---|
| Apocalypse now | 1979 | War |
| The god father | 1972 | Crime |
| Planet Earth II | 2016 | Nature documentary |
| Jack and Jill | 2011 | Comedy |

| actorname | moviename |
|---|---|
| Marlon Brando | Apocalypse now |
| Al Pacino | The god father |
| Marlon Brando | The god father |
| Al Pacino | Jack and Jill |

```
SELECT a.actorname
FROM   ActedIn a
WHERE  a.actorname
NOT IN ( SELECT a.actorname
         FROM   Movie m, ActedIn a
         WHERE  m.name=a.moviename AND m.genre ≠ 'Crime')
```

| actorname |
|---|

Non-monotone

# Monotone Queries

- Theorem: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

- Proof. We use the nested loop semantics: if we insert a tuple in a relation $R_i$, this will not remove any tuples from the answer

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

Add a tuple to $R_2$…

```
for x1 in R1 do
  for x2 in R2 do
    …
    for xn in Rn do
      if Conditions
        output (a1,…,ak)
```

…can't lose anything here.

# Non-Monotone Queries

- If a query is monotonic, it implies that a nested query can be unnested.
- If a query is non-monotone, it implies that a nested query can NOT be unnested
- Queries with universal quantifiers or negation and aggerates are non-monotone