Nested Column Queries

This lesson discusses nested queries that return a column of values.

Nested Column Queries

In the previous lesson we examined nested queries that returned a single value. In this lesson we'll see nested queries that return values belonging to the same column.

> Connect to the terminal below by clicking in the widget. Once connected, the command line prompt will show up. Enter or copy and paste the command **./DataJek/Lessons/31lesson.sh** and wait for the MySQL prompt to start-up.

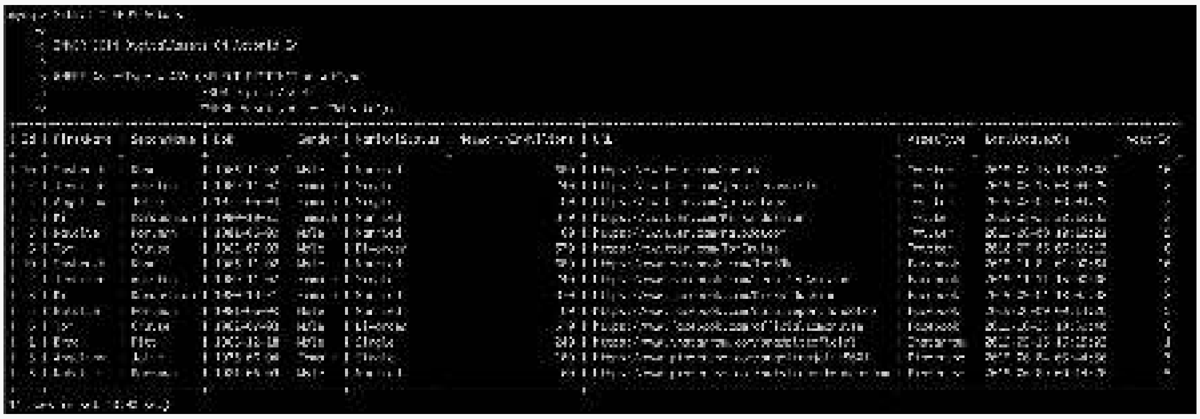
```
-- The lesson queries are reproduced below for convenient copy/paste into the terminal.

-- Query 1
SELECT * FROM Actors
INNER JOIN DigitalAssets ON ActorId=Id
WHERE AssetType = ANY (SELECT DISTINCT AssetType
                       FROM DigitalAssets
                       WHERE AssetType != 'Website');

-- Query 2
SELECT * FROM Actors
INNER JOIN DigitalAssets ON ActorId=Id
WHERE AssetType != 'Website';

-- Query 3
SELECT FirstName, SecondName
FROM Actors
WHERE Id = ANY (SELECT ActorId
                FROM DigitalAssets
                WHERE AssetType = 'Facebook');

-- Query 4
SELECT FirstName, SecondName
```

```
FROM Actors
WHERE Id IN (SELECT ActorId
             FROM DigitalAssets

             WHERE AssetType = 'Facebook');

-- Query 5
SELECT FirstName, SecondName
FROM Actors
WHERE NetworthInMillions > ALL (SELECT NetworthInMillions
                                FROM Actors
                                WHERE FirstName LIKE "j%");
```

Terminal

1. We'll use a slightly contrived example this time. Imagine we want to list all the social media accounts for all the actors, except for their personal websites. From our database schema we know that the table **DigitalAssets** has a column **AssetType**, which is essentially an enum and has a value website to denote an actor's personal website. The DigitalAssets table by itself can't give us the names of the actors since it only contains actor IDs. We'll require an inner join with the **Actors** table to get the actor names. The complete query is shown below:

   ```
   SELECT * FROM Actors

   INNER JOIN DigitalAssets ON ActorId=Id

   WHERE AssetType = ANY (SELECT DISTINCT AssetType
                          FROM DigitalAssets
                          WHERE AssetType != 'Website');
   ```

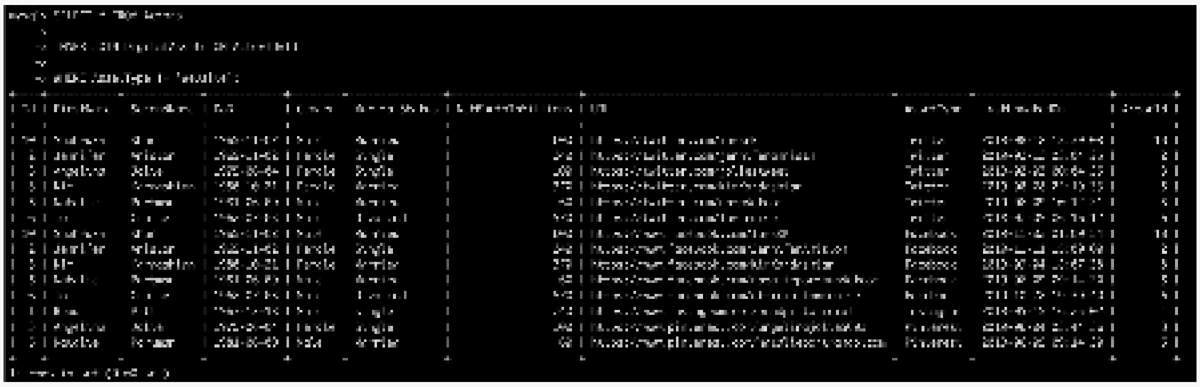
   ```
   [illegible]
   ```

   The subquery returns all the enum values for the column **AssetType** except the value "Website". The **WHERE** clause of the outer query

sets up a condition which evaluates to true whenever the column **AssetType** of the resulting inner join equals *any* of the values returned by the inner query. The **ANY** operator allows us to match the column **AssetType** with *any one of* the values returned for the column **AssetType**.

Granted, the same query can be written much simpler as follows without the need for an inner query, but the intention was to demonstrate a column subquery.

```sql
-- A much simpler approach to get the same result

SELECT * FROM Actors

INNER JOIN DigitalAssets ON ActorId=Id

WHERE AssetType != 'Website';
```

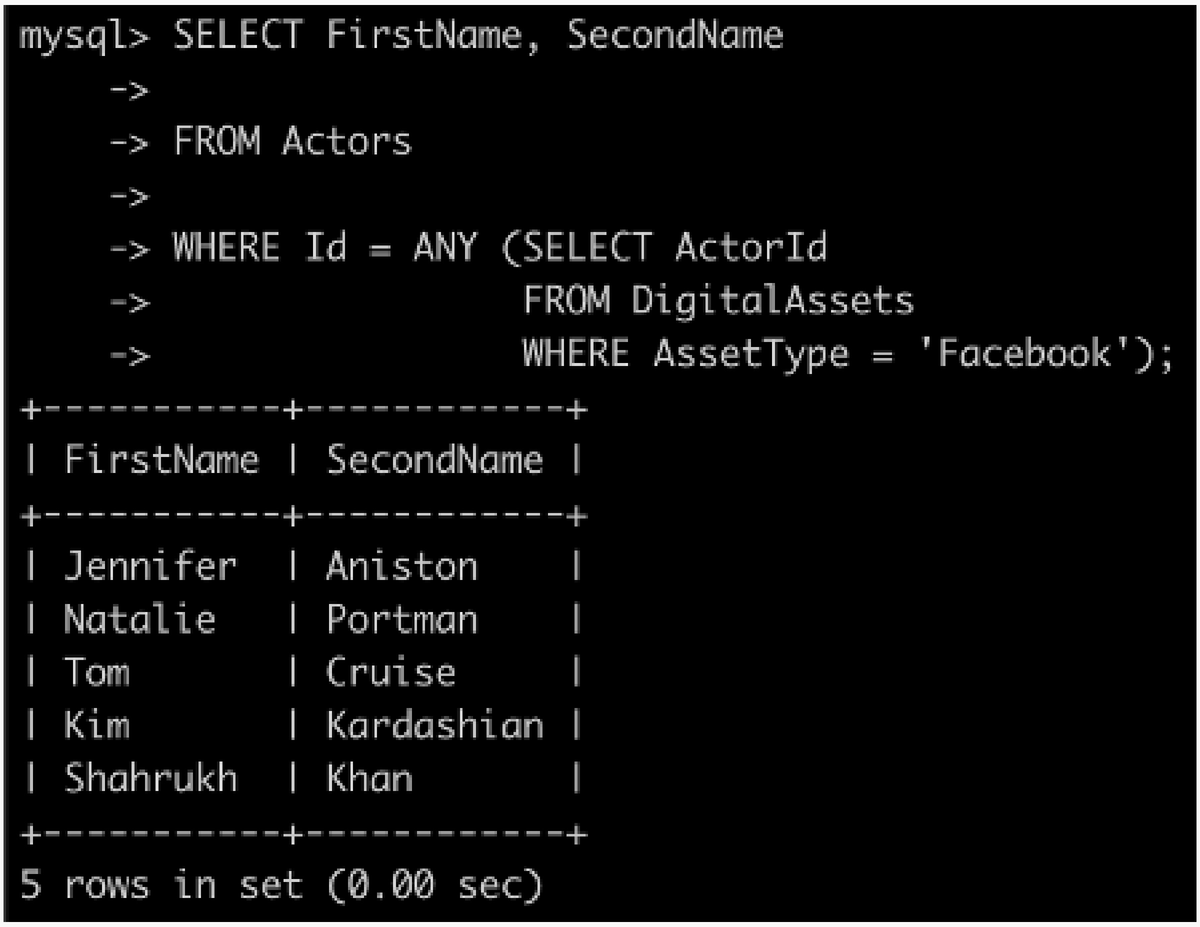


[illegible]

2. Let's work another example. Say we now want to find the names of all the actors that have a Facebook presence. One way we can answer this query is to first collect all the actor IDs from the **DigitalAssets** table that have Facebook asset types. Next, we select all those rows from the **Actors** table whose ID matches any of the IDs from the first query:

```sql
SELECT FirstName, SecondName

FROM Actors

WHERE Id = ANY (SELECT ActorId
                FROM DigitalAssets
                WHERE AssetType = 'Facebook');
```

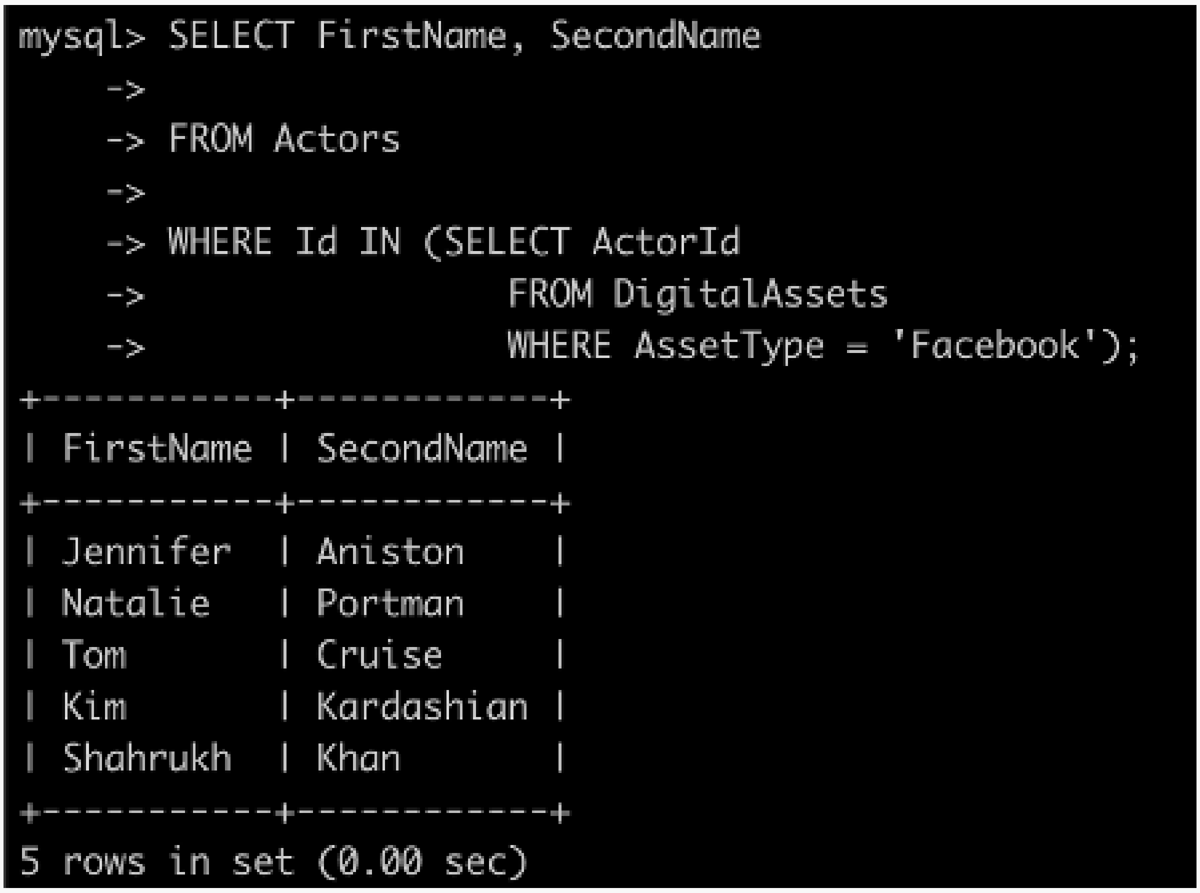
```
                 WHERE AssetType = 'Facebook');
```



```
mysql> SELECT FirstName, SecondName
    ->
    -> FROM Actors
    ->
    -> WHERE Id = ANY (SELECT ActorId
    ->                 FROM DigitalAssets
    ->                 WHERE AssetType = 'Facebook');
+-----------+------------+
| FirstName | SecondName |
+-----------+------------+
| Jennifer  | Aniston    |
| Natalie   | Portman    |
| Tom       | Cruise     |
| Kim       | Kardashian |
| Shahrukh  | Khan       |
+-----------+------------+
5 rows in set (0.00 sec)
```

The **ANY** clause has an alias **IN** that can be used interchangeably. We can rewrite the above query as follows:

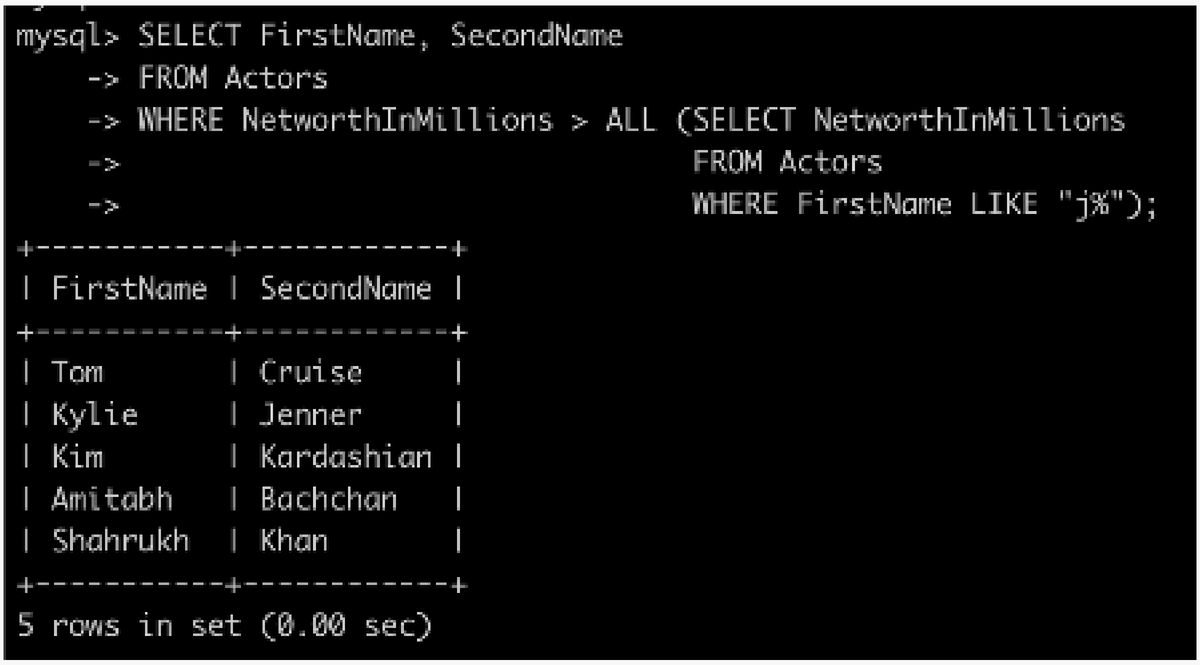
```
SELECT FirstName, SecondName

FROM Actors

WHERE Id IN (SELECT ActorId
             FROM DigitalAssets
             WHERE AssetType = 'Facebook');
```



```
mysql> SELECT FirstName, SecondName
    ->
    -> FROM Actors
    ->
    -> WHERE Id IN (SELECT ActorId
    ->                  FROM DigitalAssets
    ->                  WHERE AssetType = 'Facebook');
+-----------+------------+
| FirstName | SecondName |
+-----------+------------+
| Jennifer  | Aniston    |
| Natalie   | Portman    |
| Tom       | Cruise     |
| Kim       | Kardashian |
| Shahrukh  | Khan       |
+-----------+------------+
5 rows in set (0.00 sec)
```

3. The operator **ANY**, and its alias **IN**, match at least one value from a group of values. On the contrary, there's another operator, **ALL**, that must match all the values in the group. As an example, imagine we are asked to find out the list of actors that have a net worth greater than all the actors whose first name starts with the letter 'J'. The query would look as follows:

```
SELECT FirstName, SecondName
FROM Actors
WHERE NetworthInMillions > ALL (SELECT NetworthInMillions
                                FROM Actors
                                WHERE FirstName LIKE "j%");
```



```
mysql> SELECT FirstName, SecondName
    -> FROM Actors
    -> WHERE NetworthInMillions > ALL (SELECT NetworthInMillions
    ->                                 FROM Actors
    ->                                 WHERE FirstName LIKE "j%");
+-----------+------------+
| FirstName | SecondName |
+-----------+------------+
| Tom       | Cruise     |
| Kylie     | Jenner     |
| Kim       | Kardashian |
| Amitabh   | Bachchan   |
| Shahrukh  | Khan       |
+-----------+------------+
5 rows in set (0.00 sec)
```