Lesson 8: Advanced Queries with Subqueries

Duration: 20 minutes
Deliverable: `lesson8_advanced.sql`

🎯 Learning Objectives

By the end of this lesson, you will be able to: - Write subqueries in WHERE clauses - Use subqueries in SELECT statements - Work with IN, NOT IN, EXISTS, and NOT EXISTS - Combine subqueries with JOINs - Solve complex data retrieval problems

📚 What are Subqueries?

A **subquery** (or nested query) is a query inside another query. They allow you to: - Use the results of one query in another - Break complex problems into simpler steps - Write more readable code - Solve problems that would be difficult with just JOINs

Example Scenario: "Find all characters from the same planet as Luke Skywalker"

🛠️ Part 1: Subqueries in WHERE Clause (10 minutes)

Step 1: Create Your SQL File

1. Create: `lesson8_advanced.sql`
2. Add header:

```
-- Lesson 8: Advanced Queries with Subqueries
-- Student Name: [Your Name]
-- Date: [Today's Date]
--
-- This script demonstrates subqueries and advanced query techniques
```

Basic Subquery Syntax

```
SELECT columns
FROM table_name
WHERE column OPERATOR (SELECT column FROM table WHERE condition);
```

Step 2: Simple Subquery Example

```
-- Query 1: Find all characters from the same planet as Luke Skywalker

-- First, let's see what planet Luke is from
SELECT homeworld_id FROM characters WHERE name = 'Luke Skywalker';

-- Now use that in a subquery
SELECT name, species
FROM characters
WHERE homeworld_id = (SELECT homeworld_id FROM characters WHERE name = 'Luke Skywalke
```

Explanation: The subquery finds Luke's homeworld_id, then the main query finds all characters with that same homeworld_id.

Step 3: Subquery with Comparison Operators

```
-- Query 2: Find characters taller than the average height
SELECT name, height
FROM characters
WHERE height > (SELECT AVG(height) FROM characters)
ORDER BY height DESC;
```

Breakdown: 1. Subquery calculates average height 2. Main query finds characters above that average

Step 4: Subquery with IN Operator

```
-- Query 3: Find characters from planets with "desert" terrain
SELECT name, species
FROM characters
WHERE homeworld_id IN (
    SELECT id
    FROM planets
    WHERE terrain LIKE '%desert%'
);
```

IN allows the subquery to return multiple values.

Step 5: Subquery with NOT IN

```
-- Query 4: Find characters NOT from Tatooine or Alderaan
SELECT name, species
FROM characters
WHERE homeworld_id NOT IN (
    SELECT id
    FROM planets
    WHERE name IN ('Tatooine', 'Alderaan')
);
```

Step 6: Nested Subqueries

```
-- Query 5: Find characters from planets with above-average population
SELECT c.name, p.name AS homeworld, p.population
FROM characters c
JOIN planets p ON c.homeworld_id = p.id
WHERE p.population > (
    SELECT AVG(population)
    FROM planets
    WHERE population IS NOT NULL
)
ORDER BY p.population DESC;
```

🔍 Part 2: EXISTS and NOT EXISTS (7 minutes)

EXISTS checks if a subquery returns any rows. It's often more efficient than IN for large datasets.

EXISTS Syntax

```
SELECT columns
FROM table1
WHERE EXISTS (SELECT 1 FROM table2 WHERE condition);
```

Step 7: Using EXISTS

```
-- Query 6: Find characters who pilot at least one vehicle
SELECT name, species
FROM characters c
WHERE EXISTS (
    SELECT 1
    FROM character_vehicles cv
    WHERE cv.character_id = c.id
);
```

Explanation: For each character, check if there's a matching row in character_vehicles.

Step 8: Using NOT EXISTS

```
-- Query 7: Find characters who DON'T pilot any vehicles
SELECT name, species
FROM characters c
WHERE NOT EXISTS (
    SELECT 1
    FROM character_vehicles cv
    WHERE cv.character_id = c.id
);
```

Difference from LEFT JOIN + NULL: EXISTS often performs better and is more readable.

Step 9: EXISTS with Correlated Subquery

```
-- Query 8: Find planets that have at least one human character
SELECT p.name AS planet_name, p.climate
FROM planets p
WHERE EXISTS (
    SELECT 1
    FROM characters c
    WHERE c.homeworld_id = p.id
    AND c.species = 'Human'
);
```

Correlated subquery: The subquery references the outer query's table (p.id).

Step 10: Complex EXISTS Example

```
-- Query 9: Find characters who pilot the same type of vehicle as Luke
SELECT DISTINCT c.name
FROM characters c
WHERE EXISTS (
    SELECT 1
    FROM character_vehicles cv1
    JOIN vehicles v1 ON cv1.vehicle_id = v1.id
    WHERE cv1.character_id = c.id
    AND v1.vehicle_class IN (
        SELECT v2.vehicle_class
        FROM character_vehicles cv2
        JOIN vehicles v2 ON cv2.vehicle_id = v2.id
        JOIN characters c2 ON cv2.character_id = c2.id
        WHERE c2.name = 'Luke Skywalker'
    )
)
AND c.name != 'Luke Skywalker';
```

This is complex! It finds characters who pilot the same vehicle class as Luke.

📊 Part 3: Subqueries in SELECT (3 minutes)

You can use subqueries in the SELECT list to add calculated columns.

Step 11: Subquery in SELECT

```
-- Query 10: Show each character with a count of how many vehicles they pilot
SELECT
    c.name,
    c.species,
    (SELECT COUNT(*)
     FROM character_vehicles cv
     WHERE cv.character_id = c.id) AS vehicle_count
FROM characters c
ORDER BY vehicle_count DESC;
```

Alternative: This can also be done with LEFT JOIN and GROUP BY, but subqueries are sometimes clearer.

Step 12: Multiple Subqueries in SELECT

```
-- Query 11: Character statistics
SELECT
    name,
    height,
    (SELECT AVG(height) FROM characters) AS avg_height,
    height - (SELECT AVG(height) FROM characters) AS height_difference
FROM characters
WHERE height IS NOT NULL
ORDER BY height_difference DESC;
```

Shows: How each character's height compares to the average.

🎓 Practice Exercises

Exercise 1: Subquery with IN

```
-- Exercise 1: Find all characters from planets with "temperate" climate
SELECT name, species
FROM characters
WHERE homeworld_id IN (
    SELECT id
    FROM planets
    WHERE climate = 'temperate'
);
```

Exercise 2: Subquery with Comparison

```
-- Exercise 2: Find vehicles piloted by more characters than average
SELECT v.name, COUNT(cv.character_id) AS pilot_count
FROM vehicles v
JOIN character_vehicles cv ON v.id = cv.vehicle_id
GROUP BY v.name
HAVING COUNT(cv.character_id) > (
    SELECT AVG(pilot_cnt)
    FROM (
        SELECT COUNT(character_id) AS pilot_cnt
        FROM character_vehicles
        GROUP BY vehicle_id
    )
);
```

Exercise 3: NOT EXISTS

```
-- Exercise 3: Find planets with no characters
SELECT name, climate
FROM planets p
WHERE NOT EXISTS (
    SELECT 1
    FROM characters c
    WHERE c.homeworld_id = p.id
);
```

Exercise 4: Complex Subquery

```
-- Exercise 4: Find characters shorter than all droids
SELECT name, species, height
FROM characters
WHERE height < (
    SELECT MIN(height)
    FROM characters
    WHERE species = 'Droid' AND height IS NOT NULL
)
AND height IS NOT NULL;
```

🐛 Common Errors & Troubleshooting

Error: "Subquery returns more than 1 row"

Problem: Using = or > with a subquery that returns multiple values.

Wrong:

```
WHERE homeworld_id = (SELECT id FROM planets WHERE climate = 'temperate');
-- Returns multiple planets!
```

Correct:

```
WHERE homeworld_id IN (SELECT id FROM planets WHERE climate = 'temperate');
```

Slow Query Performance

Problem: Subquery runs for every row (correlated subquery).

Less Efficient:

```
SELECT name,
    (SELECT COUNT(*) FROM vehicles WHERE...) AS vehicle_count
FROM characters;
```

More Efficient:

```
SELECT c.name, COUNT(v.id) AS vehicle_count
FROM characters c
LEFT JOIN vehicles v ON...
GROUP BY c.name;
```

When to use each: Subqueries are clearer but JOINs are often faster.

Correlated Subquery Confusion

Problem: Forgetting to reference outer query.

Wrong:

```
WHERE EXISTS (SELECT 1 FROM planets WHERE population > 1000000);
-- Doesn't reference outer query at all!
```

Correct:

```
WHERE EXISTS (SELECT 1 FROM planets p WHERE c.homeworld_id = p.id AND p.population >
```

NULL Handling in Subqueries

Problem: NULLs cause unexpected results.

Solution: Filter NULLs in subquery:

```
WHERE height > (SELECT AVG(height) FROM characters WHERE height IS NOT NULL);
```

✅ Checkpoint: What You've Learnt

Before moving on, make sure you can:

- ✅ Write subqueries in WHERE clauses
- ✅ Use IN and NOT IN with subqueries
- ✅ Use EXISTS and NOT EXISTS
- ✅ Write subqueries in SELECT statements
- ✅ Understand correlated vs. non-correlated subqueries
- ✅ Know when to use subqueries vs. JOINs

🎯 Challenge Problem (Optional)

Task: Write a query using ONLY subqueries (no JOINs) that finds all characters who: 1. Are from the same planet as someone who pilots a starfighter 2. But do NOT pilot any starfighters themselves

Requirements: - Use subqueries only (no JOINs) - Use IN or EXISTS - Filter correctly

Click to reveal the solution

```
SELECT name, species
FROM characters
WHERE homeworld_id IN (
    -- Find planets of characters who pilot starfighters
    SELECT homeworld_id
    FROM characters
    WHERE id IN (
        SELECT character_id
        FROM character_vehicles
        WHERE vehicle_id IN (
            SELECT id
            FROM vehicles
            WHERE vehicle_class = 'Starfighter'
        )
    )
)
AND id NOT IN (
    -- Exclude characters who pilot starfighters
    SELECT character_id
    FROM character_vehicles
    WHERE vehicle_id IN (
        SELECT id
        FROM vehicles
        WHERE vehicle_class = 'Starfighter'
    )
);
```

💾 Save Your Work with Git

```
git status
git add lessons/lesson8_advanced.sql
git commit -m "Completed Lesson 8: Advanced queries with subqueries"
git push
```

📖 Key SQL Commands Learnt

Command	Purpose	Example
Subquery in WHERE	Filter using nested query	`WHERE id = (SELECT...)`
`IN`	Match any value in list	`WHERE id IN (SELECT...)`
`NOT IN`	Exclude values in list	`WHERE id NOT IN (SELECT...)`
`EXISTS`	Check if subquery returns rows	`WHERE EXISTS (SELECT...)`
`NOT EXISTS`	Check if subquery returns no rows	`WHERE NOT EXISTS (SELECT...)`
Subquery in SELECT	Calculate column value	`SELECT (SELECT COUNT(*)...)`
Correlated Subquery	References outer query	`WHERE EXISTS (SELECT 1 WHERE outer.id = inner.id)`

🔄 Subquery vs JOIN Comparison

Aspect	Subquery	JOIN
Readability	Often clearer for simple cases	Better for complex relationships
Performance	Can be slower (especially correlated)	Usually faster
Use Case	Single value comparisons, EXISTS	Combining multiple table columns
Reusability	Less reusable	Can join multiple tables once

🎉 Excellent Work!

You can now write complex nested queries! In the next lesson, you'll integrate SQL with Python to build applications.

Ready to continue? Move on to `lesson9_instructions.md`

Need Help? - Start with simple subqueries first - Test subquery separately before nesting - Use comments to explain complex logic - Ask your instructor - Consider if a JOIN might be simpler!