AJL Energy Subqueries + Practice Exercises

THE MONDAY MORNING EMERGENCY

Subqueries - Finding Outliers & Exceptions

AJL Energy High Usage Analysis | Video 5 of 15 | 18 minutes

9:00 AM Monday - Operations Manager Rushes Over

"We have a problem! Some customers are using 3-4 times more electricity than normal. This could be faulty equipment, leaks, or meter issues. We need to call them BEFORE they get a shock bill or equipment catches fire!"

The Data

- 50,000 residential customers across NSW, VIC, QLD
- December 2024 electricity usage
- Usage varies by state (QLD hotter = more aircon)

The Business Question

"Which customers are using MORE than normal for THEIR state?"

Why This Is Tricky

"Normal" varies by state:

- **NSW average:** ~450 kWh (cooler, less aircon)
- **VIC average:** ~520 kWh (heating in winter)
- **QLD average:** ~680 kWh (hot, heavy aircon)

You CAN'T use a fixed threshold like "anyone over 1000 kWh"

A QLD customer using 800 kWh is NORMAL

A NSW customer using 800 kWh is HIGH

The Challenge

For EACH customer, you need to ask: "What is the average in THIS customer's state?" This question CHANGES for every row! You need a "query within a query" - that's a **SUBQUERY!**

What You'll Learn

01

Subquery in WHERE clause (filtering)

02

Subquery in SELECT clause (calculated columns)

03

Subquery in FROM clause (derived tables)

04

Correlated subqueries (row-by-row comparisons)

05

EXISTS operator (checking existence)

06

PLUS: 5 Practice Exercises with Solutions

The Excel Nightmare

- Calculate NSW average manually
- Calculate VIC average manually
- Calculate QLD average manually
- VLOOKUP each customer to their state average
- Calculate % over average with formulas
- **Time: 3+ hours**

The SQL Solution

- Correlated subquery calculates state average per row
- One query, automatic comparison
- Time: 15 minutes!

SUBQUERY BASICS: QUERY WITHIN A QUERY

What Is A Subquery?

The Simple Definition: A subquery is a query INSIDE another query. The inner query runs first, provides a value. The outer query uses that value.

Visual Structure

```
SELECT columns
FROM table
WHERE column > (SELECT AVG(column) FROM table)
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
               This is the SUBQUERY
```

Real Example - Overall Average

Question: "Find customers using more than the overall average"

Step 1 - Calculate overall average (subquery):

```
SELECT AVG(kwh_used) FROM monthly_usage;
Result: 844 kWh
```

Step 2 - Find customers above that (main query):

```
SELECT customer_name, kwh_used
FROM customers c
JOIN monthly_usage m ON c.customer_id = m.customer_id
WHERE kwh_used > (SELECT AVG(kwh_used) FROM monthly_usage);
```

What Happens

1. Subquery runs: AVG(kwh_used) = 844
2. Main query becomes: WHERE kwh_used > 844
3. All customers above 844 kWh returned

The Three Locations

Subqueries can appear in:

1

WHERE Clause

```
WHERE kwh > (SELECT AVG(kwh) FROM usage)
```

Filter: Only customers above average

2

SELECT Clause

```
SELECT name, kwh, (SELECT AVG(kwh) FROM usage) AS average
```

Column: Show the average for every row

3

FROM Clause

```
FROM (SELECT state, AVG(kwh) FROM usage GROUP BY state) AS summary
```

Derived table: Create temp summary, then query it

Why Subqueries Matter

- Answer complex "compared to what?" questions
- Calculate values dynamically
- Break complex logic into readable steps
- Essential for finding outliers and exceptions

THE PROBLEM WITH OVERALL AVERAGE

Why We Need Per-State Comparison

The Naive Approach

"Just find anyone using more than the overall average!"

```
WHERE kwh_used > (SELECT AVG(kwh_used) FROM monthly_usage)
```

Overall average = 844 kWh (mixing NSW, VIC, QLD)

What You Get

Customer	State	Usage	Result
Charlotte G.	QLD	2,450 kWh	✅ Real outlier
Evelyn T.	QLD	2,280 kWh	✅ Real outlier
Noah W.	VIC	2,100 kWh	✅ Real outlier
Emma T.	NSW	1,850 kWh	✅ Real outlier
Sophia A.	QLD	720 kWh	❌ FALSE ALARM!
Lucas M.	QLD	650 kWh	❌ FALSE ALARM!

The Problem

QLD average is 680 kWh

Sophia using 720 kWh = only 6% above QLD average (normal!)

But overall average is 844 kWh

So Sophia flagged even though she's NORMAL for QLD

The Business Impact of Wrong Approach

- **FALSE POSITIVES:** Waste time calling normal QLD customers
- **FALSE NEGATIVES:** Miss NSW customers at 600 kWh (33% over NSW avg!)
- Operations team loses trust in data
- Real emergencies get buried in noise

What We Actually Need

1

NSW Customers

Compare to NSW average (450 kWh)

2

VIC Customers

Compare to VIC average (520 kWh)

3

QLD Customers

Compare to QLD average (~680 kWh)

This requires a **CORRELATED subquery!**

The Key Insight

The question "What is the average?" has a DIFFERENT answer for each row:

- For NSW customers, answer is 450 kWh
- For VIC customers, answer is 520 kWh
- For QLD customers, answer is 680 kWh

The subquery needs to CHANGE based on the current row's state. That's what **"correlated"** means!

CORRELATED SUBQUERY: THE RIGHT SOLUTION

Per-State Comparison with Correlated Subquery

What Makes It "Correlated"

Normal subquery: Runs once, returns one value

Correlated subquery: Runs for EVERY row, uses that row's values

The Magic Line

```
WHERE c2.state = c.state
      ^^^       ^^^
      Inner     Outer
      query     query
              (current row)
```

This connects inner query to outer query. Makes subquery re-run with EACH row's state value

The Complete Query

```
SELECT c.customer_name, c.state, m.kwh_used,
      (SELECT AVG(m2.kwh_used)
       FROM monthly_usage m2
       JOIN customers c2 ON m2.customer_id = c2.customer_id
       WHERE c2.state = c.state) AS state_average
FROM customers c
JOIN monthly_usage m ON c.customer_id = m.customer_id
WHERE m.kwh_used > (SELECT AVG(m2.kwh_used)
                  FROM monthly_usage m2
                  JOIN customers c2 ON m2.customer_id = c2.customer_id
                  WHERE c2.state = c.state);
```

How It Works - Row by Row

Row 1 - Emma (NSW customer)

1. Outer query: Emma's state = NSW
2. Subquery: WHERE c2.state = 'NSW'
3. Subquery calculates: AVG for NSW only = 450 kWh
4. Comparison: Emma's 1,850 > 450? **YES → Flagged!**

Row 2 - Sophia (QLD customer)

1. Outer query: Sophia's state = QLD
2. Subquery: WHERE c2.state = 'QLD'
3. Subquery calculates: AVG for QLD only = 680 kWh
4. Comparison: Sophia's 720 > 680? **NO → Not flagged!**

Row 3 - Noah (VIC customer)

1. Outer query: Noah's state = VIC
2. Subquery: WHERE c2.state = 'VIC'
3. Subquery calculates: AVG for VIC only = 520 kWh
4. Comparison: Noah's 2,100 > 520? **YES → Flagged!**

The Perfect Result

Customer	State	Usage	State Avg	% Over	Status
Charlotte G.	QLD	2,450	680	260%	✅
Evelyn T.	QLD	2,280	680	235%	✅
Noah W.	VIC	2,100	520	304%	✅
Mia J.	VIC	1,920	520	269%	✅
Emma T.	NSW	1,850	450	311%	✅
Alexander L.	NSW	1,620	450	260%	✅

All 6 customers are 200-300% OVER their state average. These are **REAL outliers** requiring immediate attention. Zero false positives!

Business Impact

- ✅ 6 proactive calls made
- ✅ Equipment issues found before damage
- ✅ Bill shock prevented
- ✅ Customer satisfaction protected
- ✅ Operations team trusts the data

Performance Note: Correlated subqueries CAN be slow on huge datasets. For 50,000+ rows, consider window functions or JOINs. For this use case (20 customers), perfect solution!

OTHER SUBQUERY TYPES & EXISTS

SELECT Clause, FROM Clause & EXISTS Operator

TYPE 1: Subquery in SELECT Clause

Add calculated columns to every row

```
SELECT customer_name, state, kwh_used,
      (SELECT AVG(kwh_used)
       FROM monthly_usage m2
       JOIN customers c2 ON m2.customer_id = c2.customer_id
       WHERE c2.state = c.state) AS state_avg,
      (SELECT MAX(kwh_used)
       FROM monthly_usage m2
       JOIN customers c2 ON m2.customer_id = c2.customer_id
       WHERE c2.state = c.state) AS state_max
FROM customers c
JOIN monthly_usage m ON c.customer_id = m.customer_id;
```

Result: Each row shows: customer usage, state average, state maximum. All calculated dynamically per state

Use Case: Adding reference columns for comparison. Showing "you vs average" on dashboards

TYPE 2: Subquery in FROM Clause (Derived Table)

Create a temporary result set, then query it

```
SELECT state, avg_usage
FROM (
   SELECT c.state, AVG(m.kwh_used) AS avg_usage
   FROM customers c
   JOIN monthly_usage m ON c.customer_id = m.customer_id
   GROUP BY c.state
) AS state_summary
WHERE avg_usage > 600;
```

How It Works:

1. Inner query creates state_summary table in memory
2. Outer query treats it like a normal table
3. Can filter, join, or aggregate the derived table

Use Case: Multi-step logic (summarize, then filter summary). Breaking complex queries into readable chunks. Temporary calculations not needed in final result

TYPE 3: EXISTS - Check If Something Exists

Returns TRUE/FALSE, doesn't return actual data

```
SELECT customer_name
FROM customers c
WHERE EXISTS (
   SELECT 1
   FROM monthly_usage m
   WHERE m.customer_id = c.customer_id
   AND m.kwh_used > 1500
);
```

What It Does: For each customer, check: "Do they have ANY usage over 1500?" If YES (at least one row found) → Include customer. If NO (zero rows found) → Exclude customer

EXISTS vs COUNT

❌ WHERE (SELECT COUNT(*) ...) > 0 (slow, counts all)

✅ WHERE EXISTS (SELECT 1 ...) (fast, stops at first match)

EXISTS is more efficient!

NOT EXISTS - Find Missing

```
SELECT customer_name
FROM customers c
WHERE NOT EXISTS (
   SELECT 1
   FROM monthly_usage m
   WHERE m.customer_id = c.customer_id
);
```

Find customers with NO usage records at all. Useful for: Missing data, dormant accounts, orphaned records

Common EXISTS Patterns

- "Find customers WHO HAVE done X" → **EXISTS**
- "Find customers WHO HAVE NOT done X" → **NOT EXISTS**
- "Find products THAT HAVE BEEN ordered" → **EXISTS**
- "Find products NEVER ordered" → NOT EXISTS

PRACTICE EXERCISES & WHAT YOU MASTERED

5 Exercises + Solutions & Career Impact

PRACTICE EXERCISES (Try Yourself First!)

1

Exercise 1: Reverse The Logic ⭐

Find customers whose usage is BELOW their state average

Hint: Change one comparison operator!

Difficulty: Beginner

2

Exercise 2: Top N Per Group ⭐⭐

Find the top 3 highest usage customers in each state

Hint: Use ROW_NUMBER() OVER (PARTITION BY state) or correlated subquery

Difficulty: Intermediate

3

Exercise 3: NOT EXISTS Practice ⭐

Find customers who have NO usage records at all

Hint: Use NOT EXISTS with the monthly_usage table

Difficulty: Beginner

4

Exercise 4: Derived Table Challenge ⭐⭐

Create a state summary (state, total_customers, avg_usage, max_usage). Then filter to show only states with avg_usage > 600

Hint: Subquery in FROM clause

Difficulty: Intermediate

5

Exercise 5: Bill Analysis ⭐⭐⭐

Find customers whose bill is more than 150% of their state's average bill

Hint: Same pattern as usage query, but use bill_amount

Difficulty: Advanced

SOLUTIONS PROVIDED IN SQL FILE: All 5 exercises have complete solutions with explanations. Try yourself first, then check answers!

WHAT YOU MASTERED IN 18 MINUTES

Technical Skills

- ✓ Subquery in WHERE clause (filtering with calculated values)
- ✓ Subquery in SELECT clause (adding calculated columns)
- ✓ Subquery in FROM clause (derived tables)
- ✓ Correlated subqueries (row-by-row dynamic calculations)
- ✓ EXISTS and NOT EXISTS operators
- ✓ Finding outliers and exceptions systematically
- ✓ Per-group comparisons (state averages)

Business Skills

- ✓ Identifying abnormal patterns
- ✓ Preventing customer issues proactively
- ✓ State-by-state analysis (geographical segmentation)
- ✓ Threshold-based alerting
- ✓ Root cause investigation triggers

SQL Mastery Progression

1 Video 1
Basic SELECT, WHERE, JOIN

2 Video 2
INNER vs LEFT JOIN (missing data)

3 Video 3
GROUP BY aggregations (summaries)

4 Video 4
Time intelligence (YoY, MoM, rolling avg)

5 Video 5
Subqueries ← YOU ARE HERE

6 Next
CTEs (readable complex queries)

Real-World Applications

Every outlier detection system uses subqueries:

- **Fraud detection:** "Transactions above user's average"
- **Quality control:** "Defects above factory average"
- **Sales alerts:** "Reps below regional average"
- **Healthcare:** "Patients with vitals outside normal range"

Career Value

Subqueries = Intermediate→Advanced analyst skill. Interview question staple: "Find top N per group". Production systems use correlated subqueries daily. **You can now handle 70% of business analytics scenarios!**

Business Impact Delivered

6
High-usage customers
200-300% over average

$1K+
Bill shock avoided
Per customer saved

0
False positives
Perfect accuracy

The Monday Afternoon Victory

- **9:00 AM:** "Find high-usage customers urgently!"
- **9:15 AM:** Query written and tested
- **9:30 AM:** Results delivered to Operations
- **10:00 AM:** First proactive calls made
- **4:00 PM:** 2 faulty heaters found, 1 pool pump stuck on
- **Result:** 3 customers saved from $2,000+ bills

Practice Challenge: Complete all 5 exercises before Video 6. Post your solutions to team chat. Help others who get stuck. Real learning happens through practice!

Next Video Preview

Video 6: CTEs - Common Table Expressions
Qantas Frequent Flyer Tier Qualification
Learn: WITH clause, recursive CTEs, readability
Duration: 17 minutes
Why: Subqueries get messy - CTEs make them readable!

You're halfway through the intermediate tier! Keep crushing it! 🚀