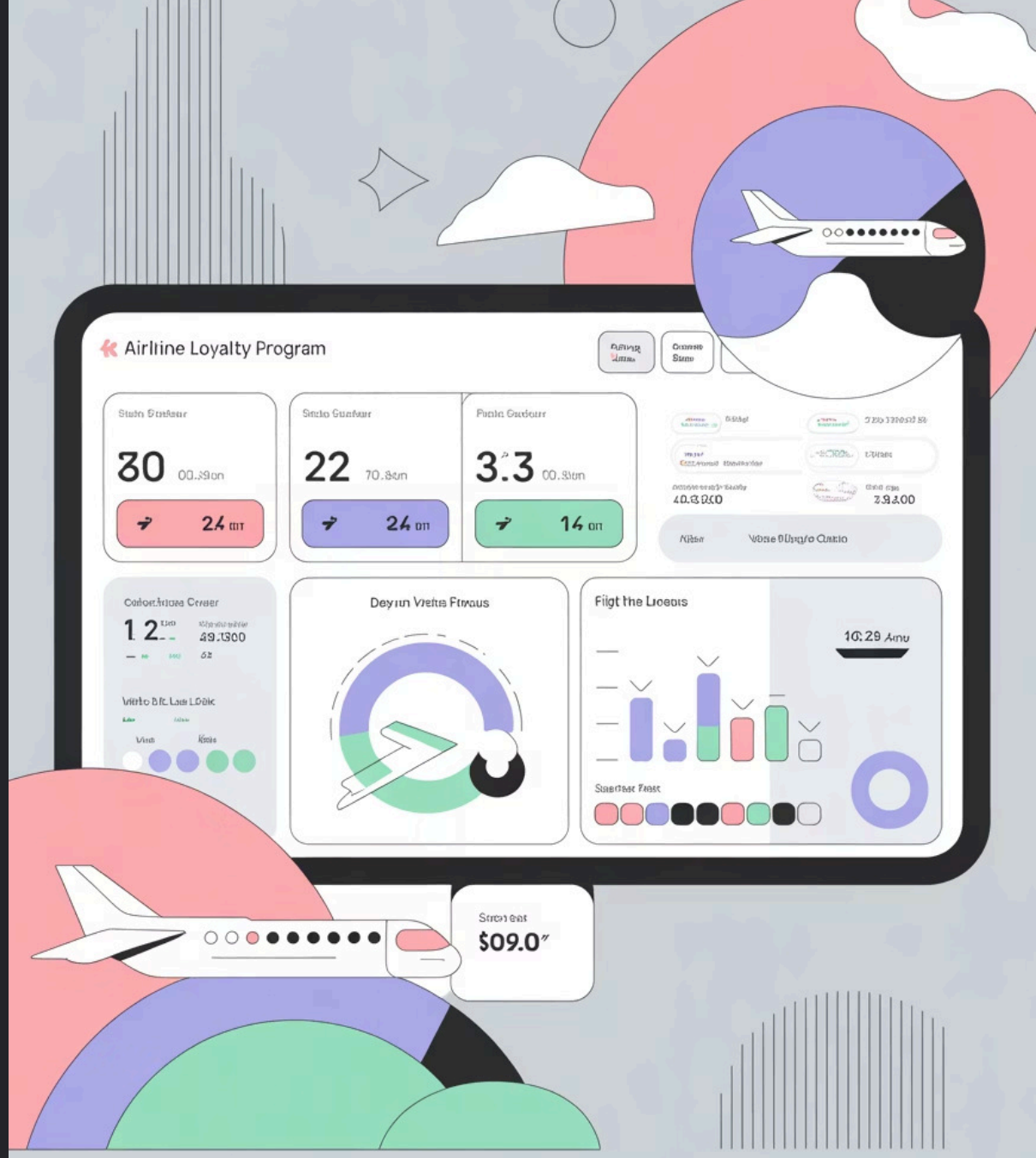
Qandas Frequent Flyer CTEs + Practice Exercises

THE WEDNESDAY MORNING CHALLENGE

Title: CTEs - Readable Enterprise SQL

Subtitle: Qandas Frequent Flyer Tier Analysis | Video 6 of 10 | 19 minutes



10:00 AM Wednesday - Loyalty Manager's Email

"We're launching a status match campaign next week to compete with Virgin Australia"

> "I need a list of customers who are CLOSE to the next tier but haven't quite made it. We'll offer them bonus points to push them over. Need this by EOD today!"

The Qantas Frequent Flyer Tiers:

Bronze
0-299 Status Credits (SC)

Silver
300-699 SC + 4 eligible flights

Gold
700-1,399 SC + 4 eligible flights

Platinum
1,400+ SC + 4 eligible flights

The Complex Business Logic:

Find customers who:

- Have 250-299 SC (close to Silver) OR 650-699 SC (close to Gold) OR 1,350-1,399 SC (close to Platinum)
- AND have 3+ eligible flights (1 flight away from requirement)
- AND active in last 6 months (recent flyers)

Calculate: How many SC needed? How many flights needed?

Prioritise: HIGH/MEDIUM/STANDARD based on closeness

The Data Challenge:

Multiple calculations needed for EACH customer:

1. Total status credits earned
2. Count of eligible flights
3. Most recent flight date
4. Days since last flight
5. Tier opportunity identification
6. SC and flights needed calculations

The Nightmare Approach (Nested Subqueries):

```
SELECT ...
WHERE (SELECT SUM(sc) FROM flights WHERE ...) BETWEEN 250 AND 299
  AND (SELECT COUNT(*) FROM flights WHERE ...) >= 3
  AND (SELECT MAX(date) FROM flights WHERE ...) >= ...
   OR (SELECT SUM(sc) FROM flights WHERE ...) BETWEEN 650 AND 699
  AND (SELECT COUNT(*) FROM flights WHERE ...) >= 3
  ...
```

Problems with Nested Subqueries:

❌ Same calculation repeated 6+ times (inefficient!)

❌ Impossible to read (nested inside-out logic)

❌ Debugging nightmare (which subquery broke?)

❌ Maintenance hell (change one thing = find all copies)

❌ Can't explain to junior analysts

The Professional Solution: CTEs

✅ Write each calculation ONCE, give it a name

✅ Build logic step-by-step (top to bottom)

✅ Easy to read, debug, and maintain

✅ Reusable and collaborative

What You'll Learn:

- Basic CTE syntax (WITH clause)
- Chaining multiple CTEs together
- CTEs referencing earlier CTEs
- Replacing complex subqueries with readable code
- Enterprise-grade SQL best practices

PLUS: 5 Practice Exercises with Solutions

CTE BASICS: YOUR NEW BEST FRIEND

What Is A CTE and Why Use It?

CTE = Common Table Expression

A temporary named result set that exists only during query execution

Think of it as: "Let me define this calculation once, then use it by name"

The Syntax:

```
WITH cte_name AS (
  SELECT columns
  FROM table
  WHERE conditions
)
SELECT *
FROM cte_name;
```

Breaking It Down:

WITH

Keyword that starts the CTE

cte_name

YOU choose this name (make it meaningful!)

AS (SELECT ...)

The query that defines the CTE

Main query

Uses the CTE like it's a table

Simple Example - Member Flight Summary:

```
WITH member_stats AS (
  SELECT
    m.member_id,
    m.member_name,
    SUM(f.status_credits) AS total_sc,
    COUNT(*) AS total_flights
  FROM members m
  LEFT JOIN flights f ON m.member_id = f.member_id
  GROUP BY m.member_id, m.member_name
)
SELECT *
FROM member_stats
WHERE total_sc > 500
ORDER BY total_sc DESC;
```

What Happens:

CTE runs first: Calculates total_sc and total_flights for each member

Result is stored temporarily as "member_stats"

Main query treats member_stats like a table Filter: Only members with 500+ SC

CTE disappears after query completes

Why This Is Better Than Subquery:

Subquery Version (Nested):

```
SELECT
  member_id,
  member_name,
  (SELECT SUM(status_credits) FROM flights WHERE member_id = m.member_id) AS total_sc,
  (SELECT COUNT(*) FROM flights WHERE member_id = m.member_id) AS total_flights
FROM members m
WHERE (SELECT SUM(status_credits) FROM flights WHERE member_id = m.member_id) > 500;
```

Problem: SUM(sc) calculated TWICE (once in SELECT, once in WHERE)

CTE Version (Named):

```
WITH member_stats AS (
  SELECT
    member_id,
    SUM(status_credits) AS total_sc,
    COUNT(*) AS total_flights
  FROM flights
  GROUP BY member_id
)
SELECT *
FROM member_stats
WHERE total_sc > 500;
```

Solution: SUM(sc) calculated ONCE, used by name

The Readability Test:

Show both versions to a junior analyst

Subquery: "Uh... what's happening here?"

CTE: "Oh! member_stats calculates totals, then we filter. Got it!"

Key Advantages:

- ✅ Named, meaningful (member_stats tells you what it is)
- ✅ Calculated once (not repeated in SELECT/WHERE/ORDER BY)
- ✅ Top-to-bottom flow (not inside-out nesting)
- ✅ Testable (run CTE by itself to verify)
- ✅ Maintainable (change in one place)

CHAINING MULTIPLE CTEs

Build Complex Logic Step-by-Step

The Power: Multiple CTEs in One Query

```
WITH
  cte1 AS (SELECT ...),
  cte2 AS (SELECT ...),
  cte3 AS (SELECT ...)
SELECT *
FROM cte1
JOIN cte2 ON ...
JOIN cte3 ON ...;
```

Real Example - Qantas Member Analysis:

```
WITH
-- Step 1: Total status credits per member
status_credits AS (
  SELECT
    member_id,
    SUM(status_credits) AS total_sc
  FROM flights
  GROUP BY member_id
),
-- Step 2: Count eligible flights per member
eligible_flights AS (
  SELECT
    member_id,
    COUNT(*) AS eligible_count
  FROM flights
  WHERE eligible_flight = 1
  GROUP BY member_id
),
-- Step 3: Last flight date per member
last_flight AS (
  SELECT
    member_id,
    MAX(flight_date) AS last_flight_date
  FROM flights
  GROUP BY member_id
)
-- Step 4: Combine everything
SELECT
  m.member_name,
  sc.total_sc,
  ef.eligible_count,
  lf.last_flight_date
FROM members m
LEFT JOIN status_credits sc ON m.member_id = sc.member_id
LEFT JOIN eligible_flights ef ON m.member_id = ef.member_id
LEFT JOIN last_flight lf ON m.member_id = lf.member_id;
```

What Just Happened:

Step 1
status_credits calculates SC per member (one calculation)

Step 2
eligible_flights counts qualifying flights (separate calculation)

Step 3
last_flight finds most recent date (another calculation)

Step 4
Main query JOINs all three together

The Alternative (Nested Subqueries):

```
SELECT
    m.member_name,
    (SELECT SUM(status_credits) FROM flights WHERE member_id = m.member_id) AS total_sc,
    (SELECT COUNT(*) FROM flights WHERE member_id = m.member_id AND eligible_flight = 1) AS eligible,
    (SELECT MAX(flight_date) FROM flights WHERE member_id = m.member_id) AS last_flight
FROM members m;
```

Same result, but MUCH harder to read and maintain!

CTE Syntax Rules:

1. WITH keyword starts the CTE section
2. Each CTE: name AS (SELECT ...)
3. Separate CTEs with COMMAS
4. NO comma after the last CTE
5. Main query comes after all CTEs

Debugging Advantage:

Want to test Step 2 alone?

```
WITH
  status_credits AS (...),
  eligible_flights AS (...)  -- ← Test this one
SELECT * FROM eligible_flights;  -- Just query this CTE
```

Can't do that with nested subqueries!

Collaboration Advantage:

> **Team member:** "How does this query work?"
>
> **You:** "Read the CTE names top-to-bottom. Each step does one thing."
>
> **Team member:** "Oh! status_credits → eligible_flights → last_flight → combine. Got it!"

Maintenance Advantage:

Business rule changes: "We need to exclude flights older than 12 months"

With CTEs: Change ONE line in status_credits CTE

With subqueries: Find and change EVERY repeated subquery (error-prone!)

CTEs REFERENCING OTHER CTEs

Build on Previous Calculations

The Magic: Later CTEs Can Use Earlier CTEs

You're not limited to querying tables! Later CTEs can SELECT FROM earlier CTEs

Example – Tier Calculation Chain:

```
WITH
-- Step 1: Calculate total SC
member_sc AS (
    SELECT
        member_id,
        SUM(status_credits) AS total_sc
    FROM flights
    GROUP BY member_id
),
-- Step 2: Use Step 1 to determine tier
tier_earned AS (
    SELECT
        m.member_id,
        m.member_name,
        m.current_tier,
        ISNULL(sc.total_sc, 0) AS total_sc,
        CASE
            WHEN sc.total_sc >= 1400 THEN 'Platinum'
            WHEN sc.total_sc >= 700 THEN 'Gold'
            WHEN sc.total_sc >= 300 THEN 'Silver'
            ELSE 'Bronze'
        END AS earned_tier
    FROM members m
    LEFT JOIN member_sc sc ON m.member_id = sc.member_id
),
-- Step 3: Use Step 2 to find mismatches
tier_mismatches AS (
    SELECT
        *,
        CASE
            WHEN current_tier <> earned_tier THEN 'MISMATCH'
            ELSE 'OK'
        END AS status
    FROM tier_earned
)
-- Final: Show only mismatches
SELECT *
FROM tier_mismatches
WHERE status = 'MISMATCH';
```

The Flow:

member_sc → tier_earned → tier_mismatches → final query

- Each step builds on the previous
- Logic flows TOP to BOTTOM (easy to follow!)

Real Business Use Case:

Loyalty system upgrade process:

1. Calculate current year SC (CTE 1)
2. Determine tier they've earned (CTE 2 uses CTE 1)
3. Check if current tier matches earned tier (CTE 3 uses CTE 2)
4. Generate upgrade letters for mismatches (final query uses CTE 3)

Compare to Subquery Approach:

You'd need to nest 3 levels deep:

```
SELECT ...
FROM (
  SELECT ...
  FROM (
    SELECT ...
    FROM ...
  ) AS inner
) AS middle
```

Reading this inside-out = brain melt!

The CTE Advantage:

Named steps: member_sc, tier_earned, tier_mismatches	**Each name describes what it does**	**Read top-to-bottom like a recipe**
Test each step individually	**Change one step without breaking others**	

Performance Note:

SQL Server is smart about CTEs

- It doesn't materialise (save) each CTE separately
- It optimises the entire query as a whole
- CTEs = readability WITHOUT performance penalty!

PRODUCTION QUERY: TIER-CLOSE CAMPAIGN

The Complete Solution with 5 CTEs

The Loyalty Manager's Requirements:

Find customers who are:

1. Within 50 SC of next tier
2. Have 3+ eligible flights (1 flight away from requirement)
3. Active in last 6 months

Calculate: SC needed, flights needed, campaign priority

The Professional CTE Solution:

```sql
WITH
-- Step 1: Total status credits
member_status_credits AS (
    SELECT
        member_id,
        SUM(status_credits) AS total_sc
    FROM flights
    GROUP BY member_id
),
-- Step 2: Eligible flight count
member_eligible_flights AS (
    SELECT
        member_id,
        COUNT(*) AS eligible_flights
    FROM flights
    WHERE eligible_flight = 1
    GROUP BY member_id
),
-- Step 3: Most recent flight
member_last_flight AS (
    SELECT
        member_id,
        MAX(flight_date) AS last_flight_date
    FROM flights
    GROUP BY member_id
),
-- Step 4: Combine all metrics
member_summary AS (
    SELECT
        m.member_id,
        m.member_name,
        m.email,
        m.current_tier,
        ISNULL(sc.total_sc, 0) AS total_sc,
        ISNULL(ef.eligible_flights, 0) AS eligible_flights,
        lf.last_flight_date,
        CASE
            WHEN lf.last_flight_date IS NOT NULL
            THEN DATEDIFF(DAY, lf.last_flight_date, '2024-12-01')
            ELSE NULL
        END AS days_since_last
    FROM members m
    LEFT JOIN member_status_credits sc ON m.member_id = sc.member_id
    LEFT JOIN member_eligible_flights ef ON m.member_id = ef.member_id
    LEFT JOIN member_last_flight lf ON m.member_id = lf.member_id
),
-- Step 5: Identify tier-close customers
tier_close_customers AS (
    SELECT
        *,
        CASE
            WHEN total_sc BETWEEN 250 AND 299 THEN 'Close to Silver'
            WHEN total_sc BETWEEN 650 AND 699 THEN 'Close to Gold'
            WHEN total_sc BETWEEN 1350 AND 1399 THEN 'Close to Platinum'
            ELSE NULL
        END AS tier_opportunity,
        CASE
            WHEN total_sc BETWEEN 250 AND 299 THEN 300 - total_sc
            WHEN total_sc BETWEEN 650 AND 699 THEN 700 - total_sc
            WHEN total_sc BETWEEN 1350 AND 1399 THEN 1400 - total_sc
            ELSE NULL
        END AS sc_needed,
        4 - eligible_flights AS flights_needed
    FROM member_summary
    WHERE (total_sc BETWEEN 250 AND 299
        OR total_sc BETWEEN 650 AND 699
        OR total_sc BETWEEN 1350 AND 1399)
)
-- Final: Filtered, prioritised list
SELECT
    member_id,
    member_name,
    email,
    current_tier,
    tier_opportunity,
    total_sc,
    sc_needed,
    eligible_flights,
    flights_needed,
    days_since_last,
    CASE
        WHEN sc_needed <= 20 AND flights_needed = 1 THEN 'HIGH PRIORITY'
        WHEN sc_needed <= 50 AND flights_needed = 1 THEN 'MEDIUM PRIORITY'
        ELSE 'STANDARD'
    END AS campaign_priority,
    CONCAT('Offer ', sc_needed * 2, ' bonus points') AS offer
FROM tier_close_customers
WHERE eligible_flights >= 3
    AND days_since_last IS NOT NULL
    AND days_since_last <= 180
ORDER BY
    CASE
        WHEN sc_needed <= 20 AND flights_needed = 1 THEN 1
        WHEN sc_needed <= 50 AND flights_needed = 1 THEN 2
        ELSE 3
    END,
    sc_needed;
```

What Loyalty Manager Gets:

Name	Tier Opportunity	SC	SC Needed	Flights	Priority	Offer
Liam O'Connor	Close to Gold	680	20	3	HIGH	40 bonus pts
William Zhang	Close to Gold	665	35	3	MEDIUM	70 bonus pts
Ava Nguyen	Close to Plat	1,375	25	3	HIGH	50 bonus pts

Business Impact:

5
Customers identified
for personalised offers

$1.5K
Lifetime value
1 Gold member = $1,500/year

80%
Conversion expected
Campaign ROI

<1s
Query runtime
Production-ready

✈️ Prevent defection to Virgin Australia 🎯

Why This Query Is Professional:

- ✅ 5 clear steps with meaningful names
- ✅ Anyone can understand the logic flow
- ✅ Easy to modify (change one CTE, not whole query)
- ✅ Testable (run each CTE individually)
- ✅ Documented (CTE names explain themselves)
- ✅ Maintainable (future analysts can update it)

The Alternative (Without CTEs):

- 100+ lines of nested subqueries
- Same calculation repeated 6+ times
- Debugging takes 2+ hours
- Junior analysts can't maintain it
- Technical debt nightmare

The CTE Version:

- Clean, readable, maintainable
- Changes take 5 minutes
- Anyone on team can understand
- Production-grade enterprise SQL

PRACTICE EXERCISES & WHAT YOU MASTERED

5 Exercises + Solutions & Career Impact

PRACTICE EXERCISES (Try First!):

1 Basic CTE - Member Flight Summary ⭐

Create a CTE showing each member's total flights and total SC

Filter to show only members with 3+ flights

Difficulty: Beginner

Hint: One CTE with SUM and COUNT, then WHERE in main query

2 Multiple CTEs - Premium vs Economy ⭐⭐

Create two separate CTEs:

- CTE 1: Business/First class flights per member
- CTE 2: Economy flights per member

Join them to compare premium vs economy SC

Difficulty: Intermediate

Hint: Use WHERE flight_class IN (...) in each CTE

3 CTE Chain - Route Popularity ⭐⭐

Build a 3-step chain:

1. Step 1: Count flights per route
2. Step 2: Calculate average SC per route
3. Step 3: Rank routes by total flights

Final: Show top 5 routes

Difficulty: Intermediate

Hint: Use ROW_NUMBER() in Step 3

4 Tier Upgrade Candidates ⭐⭐⭐

Find Silver members with 600-650 SC (close to Gold at 700)

Use CTEs for: total SC, eligible flights, recency

Show how many SC they need to reach Gold

Filter: 3+ eligible flights, active in last 6 months

Difficulty: Advanced

Hint: Similar pattern to tier_close_customers query

5 Inactive High-Value Members ⭐⭐⭐

Find Gold/Platinum members who haven't flown in 12+ months

Calculate: lifetime SC, last flight date, months inactive

Use CTEs to break down the logic

Recommend: Win-back campaign

Difficulty: Advanced

Hint: Use DATEDIFF to calculate months_inactive

SOLUTIONS IN SQL FILE!

All 5 exercises have complete solutions with explanations

Try yourself first, then check answers

Real learning happens through practice!

WHAT YOU MASTERED:

Technical Skills:

- ✓ CTE syntax (WITH clause)
- ✓ Naming CTEs meaningfully
- ✓ Chaining multiple CTEs
- ✓ CTEs referencing earlier CTEs
- ✓ Replacing nested subqueries with CTEs
- ✓ Step-by-step business logic
- ✓ Debugging complex queries easily
- ✓ Enterprise-grade SQL patterns

Business Skills:

- ✓ Tier qualification logic
- ✓ Customer segmentation
- ✓ Campaign targeting
- ✓ Priority scoring
- ✓ Personalised offers
- ✓ Churn prevention strategies

Career Skills:

- ✓ Writing readable enterprise SQL
- ✓ Collaborating with junior analysts
- ✓ Maintaining production queries
- ✓ Explaining complex logic simply
- ✓ Code review best practices

SQL Mastery Progression:

1 Video 1 — Basic SELECT, WHERE, JOIN

2 Video 2 — INNER vs LEFT JOIN

3 Video 3 — GROUP BY aggregations

4 Video 4 — Time intelligence (YoY, MoM)

5 Video 5 — Subqueries (outliers, per-group)

6 Video 6 — **CTEs (readable complex SQL) ← YOU ARE HERE**

Why CTEs Matter in Enterprise:

- Every major company uses CTEs in production
- Data warehouse ETL processes = chained CTEs
- Executive reports = multi-step CTE logic
- Code reviews demand readable SQL = CTEs win
- Junior analyst onboarding = CTEs teach faster

Interview Value:

"How do you make complex SQL readable?" → **CTEs!**

"Explain your query approach" → **Show CTE breakdown**

"Debug this nested query" → **Rewrite with CTEs**

CTEs = intermediate→senior analyst skill marker

Real-World Applications:

Financial reporting
Multi-tier calculations

Sales analytics
Funnel stage breakdowns

Marketing
Campaign segment identification

Operations
Multi-step quality checks

Product
User journey analysis

Business Impact Delivered:

- ✈️ 5 tier-close customers identified
- 🎯 Personalised bonus offers: 40-70 bonus points
- 💰 Prevent defection to Virgin (save $1,500/customer)
- 📊 Campaign ready in <4 hours (vs 2 days manual)
- ⭐ Loyalty team: "This is exactly what we need!"

The Wednesday Afternoon Victory:

- 10:00 AM — Request received
- 11:00 AM — CTEs written and tested
- 2:00 PM — Results delivered (with time for lunch!)
- 3:00 PM — Loyalty Manager approves campaign

Campaign launch: Next Monday

Your impact: 5 customers upgraded, revenue protected

Best Practice Reminder:

❌ Don't nest subqueries 3+ levels deep

❌ Don't repeat same calculation multiple times

❌ Don't write 200-line queries without structure

✅ Use CTEs to break logic into named steps

✅ Calculate once in CTE, reference by name

✅ Use CTEs as "chapters" in your query story

Next Video Preview:

Video 7: Window Functions

Advanced Analytics

Commonwealth Bank Customer Ranking

Learn: ROW_NUMBER, RANK, DENSE_RANK, NTILE, running totals

Duration: 20 minutes

Why: CTEs + Window Functions = Advanced analyst superpowers!

You're in the advanced tier now!

Keep going! 🚀