

Temp Tables & Table
Variables

Complex Multi-Step Workflows

NAD Banking Month-End Reconciliation

Video 10 of 15 | Duration: 17 minutes | Level: Advanced



5:00 PM Friday – Finance Manager Walks In (Stressed)

"Month-end reconciliation is due Monday morning for the board meeting!"

"I need the full December reconciliation report... but it takes **3 HOURS** to complete manually!"



The 10-Step Manual Nightmare

01	02	03
Extract Transactions	Categorise	Calculate Balances
Run query to extract all December transactions → Copy 2,500+ rows to Excel → Save as "Step1_Transactions.xlsx"	Run categorisation query → Open Step1_Transactions.xlsx → VLOOKUP to categorise → Save as "Step2_Categories.xlsx"	Calculate daily running balances → Open Step2_Categories.xlsx → Manual formulas, drag down 2,500 rows → Save as "Step3_RunningBalances.xlsx"
04	05	
Identify Overdrafts	Continue Processing	
Identify overdraft accounts → Filter Step3 for negative balances → Save as "Step4_Overdrafts.xlsx"	Step 5-10: More Excel files, more manual work...	

📌 Total: 10 Excel files, 3 hours of copy-paste hell!

The Problem – Sequential Dependencies

- Can't do Step 5 without Step 3 results
- Can't do Step 8 without Steps 6 & 7 complete
- Each step DEPENDS on previous steps!

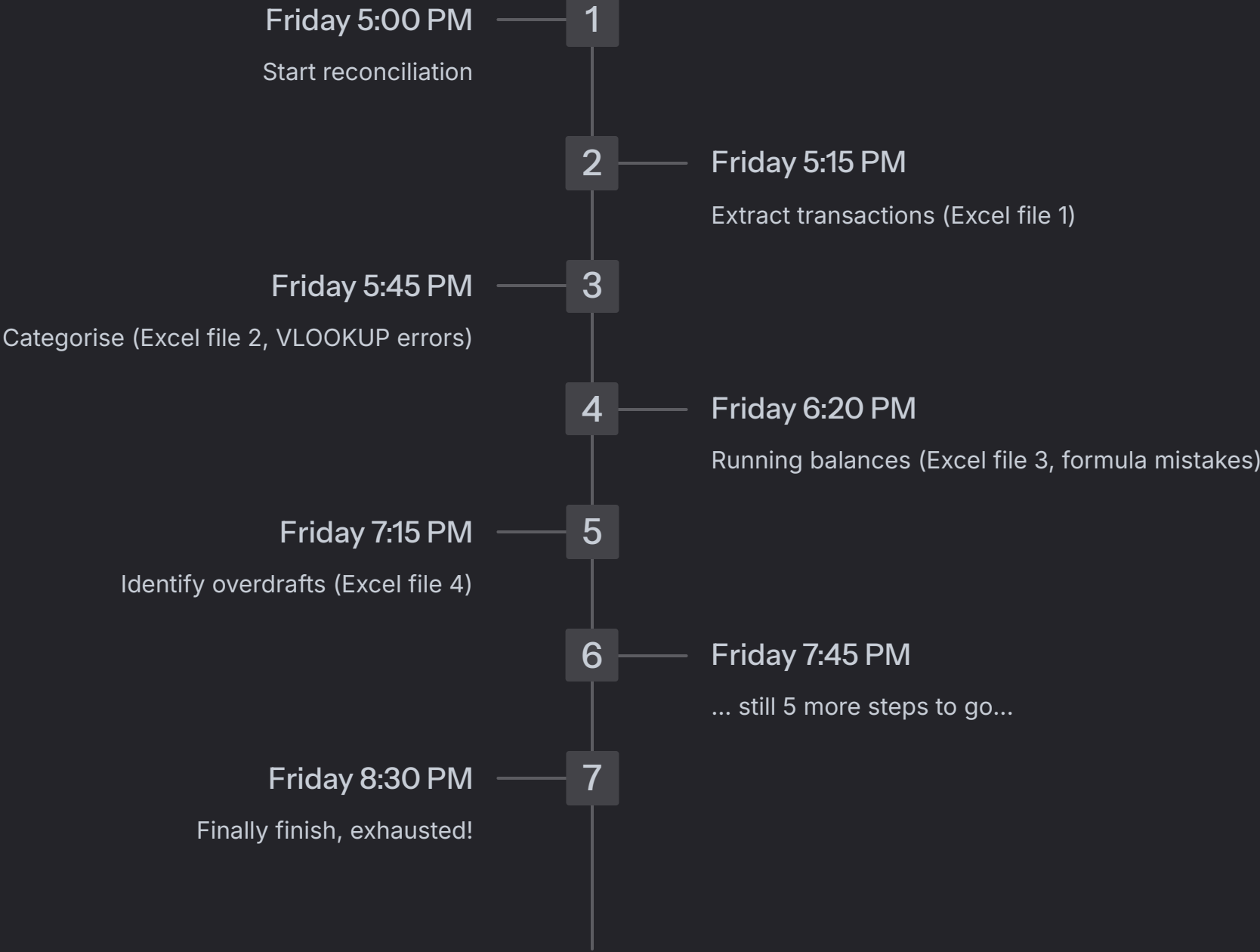
Traditional SQL Problem:

Run query 1 → Can't save results for query 2 Run query 2 → Loses query 1 results Can't chain queries together!

Solution needed:

- Store intermediate results
- Each step reads previous step's data
- Automated chain of processing

The Copy-Paste Disaster



Errors discovered:

- Wrong VLOOKUP range (Step 2)
- Formulas not copied to all rows (Step 3)
- Manual filter missed rows (Step 4)

Weekend ruined fixing errors! 😞

What If We Could...

Run ONE script?	Store intermediate results automatically?
Chain all 10 steps together?	Complete in 10 minutes instead of 3 hours?

That's what TEMP TABLES do!

The Solution: Temporary Tables

Instead of 10 Excel files:

Create TEMP TABLES in SQL

Instead of manual VLOOKUP:

JOIN temp tables automatically

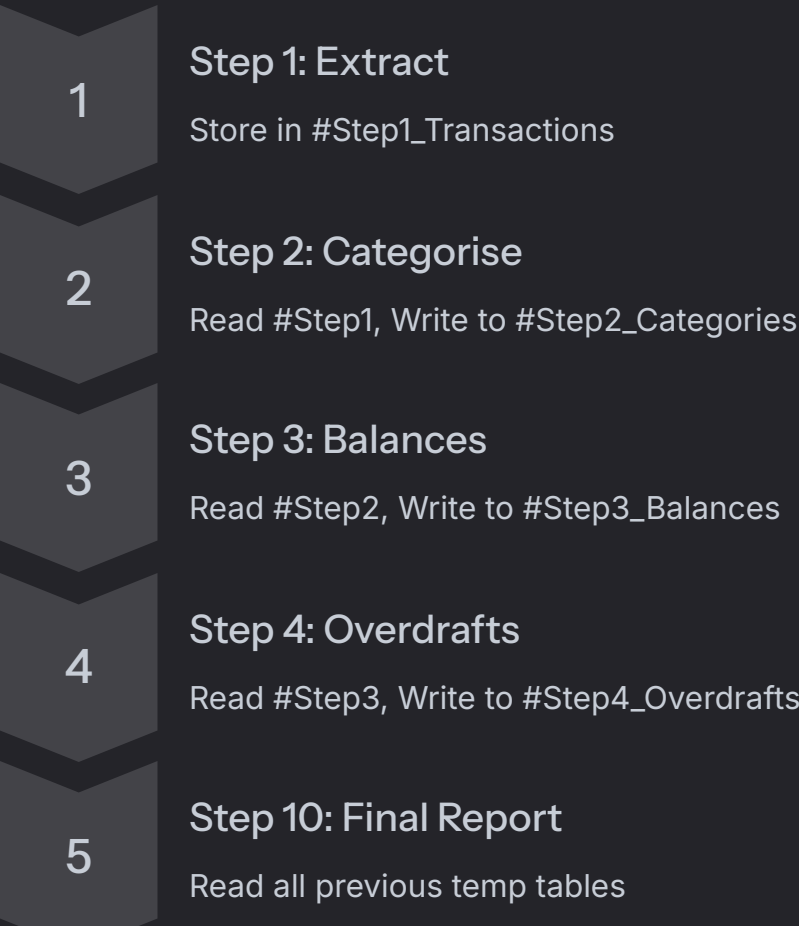
Instead of copy-paste:

INSERT INTO #TempTable SELECT...

Instead of 3 hours:

10 minutes automated execution!

The Magic Pattern



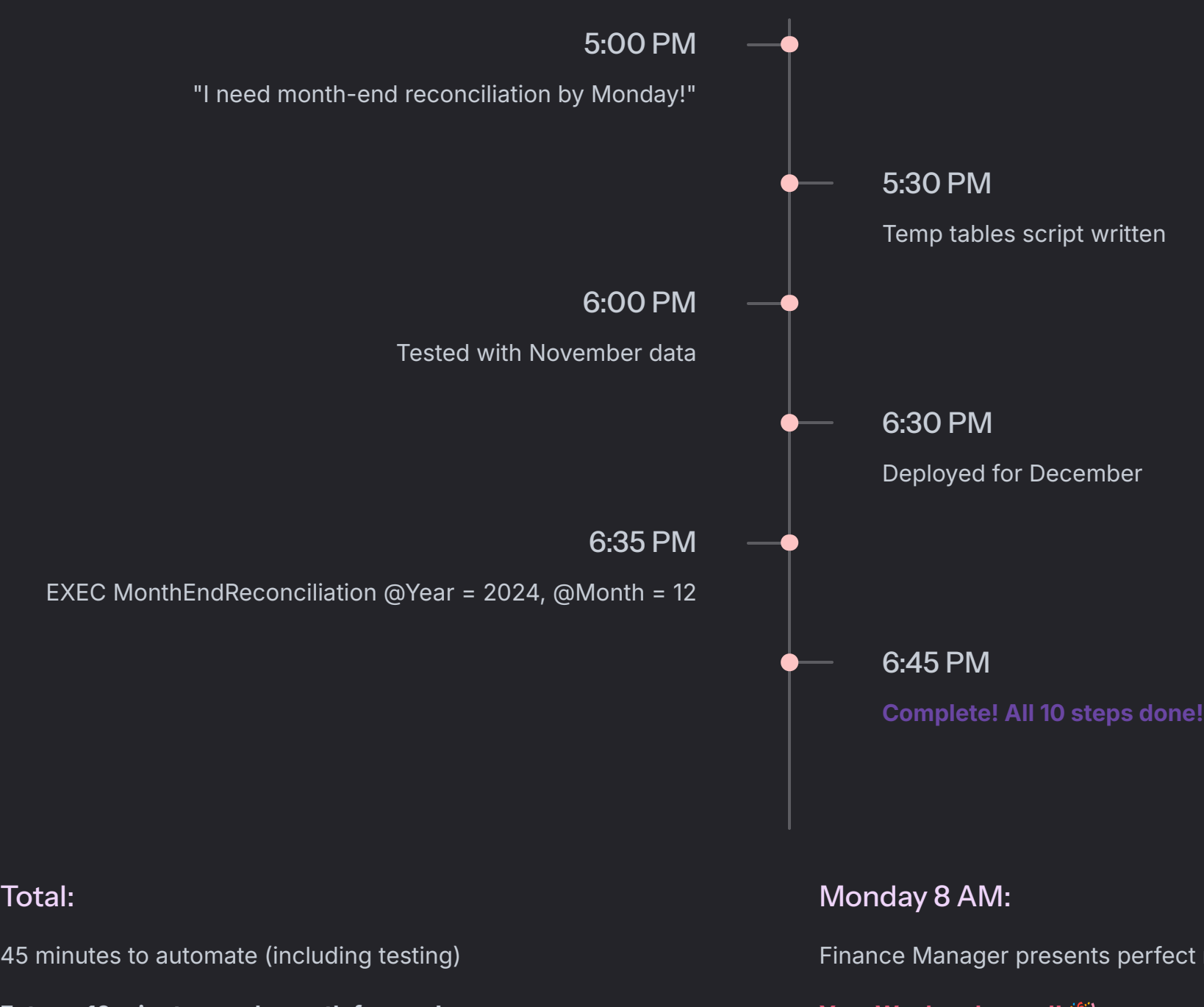
Automated! Fast! Error-free!

What You'll Master

 Temp Tables (#TempTable) <ul style="list-style-type: none">• Session-scoped storage• When to use for large datasets• Indexing for performance• Multi-step workflow patterns	 Table Variables (@TableVariable) <ul style="list-style-type: none">• Batch-scoped storage• When to use for small datasets• Lightweight alternative
 Global Temp Tables (##GlobalTemp) <ul style="list-style-type: none">• Cross-session sharing• Team collaboration scenarios	 Comparison & Decision Guide <ul style="list-style-type: none">• #TempTable vs @TableVariable vs CTE• Performance considerations• Production best practices

PLUS: 5 Practice Exercises with Solutions

The Friday Evening Victory



Session-Scoped Intermediate Storage

What Are Temp Tables?

Definition: <ul style="list-style-type: none">Temporary tables stored in tempdb databaseExist only for your session (connection)Automatically dropped when session endsPrefix with # (single hash mark)	Think of them as: <ul style="list-style-type: none">Scratch paper for complex calculationsStaging area for multi-step processesIntermediate results holder	Lifetime: <ul style="list-style-type: none">Created: When you run CREATE TABLE #NameExists: Throughout your entire sessionDropped: When session disconnects OR you DROP manually
---	---	--

Creating a Temp Table

Syntax:

```
CREATE TABLE #TempTableName (  
    column1 datatype,  
    column2 datatype,  
    ...  
);
```

Real NAB Example:

```
CREATE TABLE #MonthlyTransactions (  
    account_id INT,  
    account_number NVARCHAR(20),  
    customer_name NVARCHAR(100),  
    transaction_count INT,  
    total_deposits DECIMAL(15,2),  
    total_withdrawals DECIMAL(15,2),  
    net_change DECIMAL(15,2)  
);
```

Looks like regular table! Just has # prefix

Populating Temp Tables

Method 1: INSERT INTO ... SELECT

```
INSERT INTO #MonthlyTransactions  
SELECT  
    a.account_id,  
    a.account_number,  
    a.customer_name,  
    COUNT(*) AS transaction_count,  
    SUM(CASE WHEN amount > 0 THEN amount ELSE 0 END) AS total_deposits,  
    SUM(CASE WHEN amount < 0 THEN ABS(amount) ELSE 0 END) AS total_withdrawals,  
    SUM(amount) AS net_change  
FROM accounts a  
LEFT JOIN transactions t ON a.account_id = t.account_id  
WHERE t.transaction_date >= '2024-12-01'  
GROUP BY a.account_id, a.account_number, a.customer_name;
```

Standard INSERT syntax!

Method 2: SELECT INTO (shorthand)

```
SELECT  
    account_id,  
    SUM(amount) AS total  
INTO #QuickTemp  
FROM transactions  
GROUP BY account_id;
```

Creates temp table automatically!

Column names and types inferred!

Querying Temp Tables

Just like regular tables:

```
SELECT *  
FROM #MonthlyTransactions  
WHERE net_change > 0  
ORDER BY transaction_count DESC;
```

Can use in JOINS:

```
SELECT  
    mt.customer_name,  
    mt.net_change,  
    a.account_type  
FROM #MonthlyTransactions mt  
JOIN accounts a  
    ON mt.account_id = a.account_id;
```

Full SQL power available!

Where Are Temp Tables Stored?

Physically:

tempdb database

Check with:

```
SELECT name, create_date  
FROM tempdb.sys.tables  
WHERE name LIKE '#MonthlyTransactions%';
```

SQL Server adds unique suffix:

#MonthlyTransactions__000000000123

Why?

Prevent naming conflicts between sessions

Session Scope – What Does It Mean?

YOUR session ONLY:

- You create #MyTemp
- You can see #MyTemp
- You can query #MyTemp

OTHER sessions:

- They create #MyTemp (same name!)
- They see THEIR #MyTemp (different table!)
- They CANNOT see YOUR #MyTemp

Each session has isolated temp tables!

Cleanup – Two Ways

Automatic (recommended):

Session ends → All temp tables automatically dropped

No action needed!

Manual (when needed):

DROP TABLE #MonthlyTransactions;

Use manual DROP when:

- Reusing same session for multiple tasks
- Large temp table (free up tempdb space)
- Testing/development (clean slate between runs)

Temp Table Characteristics

- Can add indexes
- CREATE INDEX IX_Account ON #MonthlyTransactions(account_id);
- Can add constraints
- ALTER TABLE #Temp ADD CONSTRAINT PK_ID PRIMARY KEY (id);
- Has statistics (query optimiser uses them)
- UPDATE STATISTICS #MonthlyTransactions;
- Supports transactions
- BEGIN TRANSACTION;
INSERT INTO #Temp...;
COMMIT;
- Can participate in JOIN, WHERE, ORDER BY
- Just like permanent tables!

When to Use Temp Tables

Use temp tables when:

- ✔ Large datasets (1,000+ rows)
- ✔ Reused multiple times in script
- ✔ Multi-step workflows (step results)
- ✔ Need indexes for performance
- ✔ Complex transformations
- ✔ Data persists across batches (GO statements)

Examples:

- Month-end reconciliation (step-by-step processing)
- Complex ETL workflows (extract → transform → load)
- Report generation (aggregate → filter → format)
- Data quality checks (identify → categorise → report)

When NOT to Use Temp Tables

Don't use temp tables when:

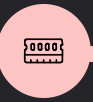
- ✗ Small dataset (< 100 rows) → Use table variable
- ✗ Single query only → Use CTE instead
- ✗ Simple aggregation → Use subquery
- ✗ Immediate one-time result → Just SELECT

Overkill examples:

- Quick COUNT(*) check
- Simple INNER JOIN query
- One-time data exploration
- Ad-hoc stakeholder question


Batch-Scoped Quick Storage

What Are Table Variables?




Definition:

- In-memory table structure
- Declared with DECLARE (not CREATE)
- Batch-scoped (shorter lifetime than temp tables)
- Prefix with @ (single at-sign)



Think of them as:

- Variables that hold table data
- Quick staging for small datasets
- Lightweight temp table alternative



Lifetime:

- **Declared:** In current batch only
- **Exists:** Until end of batch (GO statement)
- **Dropped:** Automatically at batch end

Declaring a Table Variable

Syntax:

```
DECLARE @VariableName TABLE (  
    column1 datatype,  
    column2 datatype,  
    ...  
);
```

Real NAB Example:

```
DECLARE @AccountSummary TABLE (  
    account_id INT,  
    customer_name NVARCHAR(100),  
    current_balance DECIMAL(15,2),  
    risk_level NVARCHAR(20)  
);
```

Looks like temp table, but DECLARE instead of CREATE!

Populating Table Variables

INSERT INTO @VariableName

```
INSERT INTO @AccountSummary  
SELECT  
    a.account_id,  
    a.customer_name,  
    a.opening_balance + ISNULL(SUM(t.amount), 0) AS current_balance,  
    CASE  
        WHEN a.opening_balance + ISNULL(SUM(t.amount), 0) < 1000 THEN 'High Risk'  
        WHEN a.opening_balance + ISNULL(SUM(t.amount), 0) < 5000 THEN 'Medium Risk'  
        ELSE 'Low Risk'  
    END AS risk_level  
FROM accounts a  
LEFT JOIN transactions t ON a.account_id = t.account_id  
GROUP BY a.account_id, a.customer_name, a.opening_balance;
```

Same INSERT syntax as temp tables!

Querying Table Variables

```
SELECT *  
FROM @AccountSummary  
WHERE risk_level = 'High Risk'  
ORDER BY current_balance;
```

Can JOIN with other tables:

```
SELECT  
    a.account_type,  
    av.risk_level,  
    COUNT(*) AS account_count  
FROM @AccountSummary av  
JOIN accounts a  
    ON av.account_id = a.account_id  
GROUP BY a.account_type, av.risk_level;
```

Full query capabilities!

Key Difference - Batch Scope




Batch = Code between GO statements

Example - Table Variable Dies at GO:

```
DECLARE @Temp TABLE (id INT);  
INSERT INTO @Temp VALUES (1);  
SELECT * FROM @Temp; --
```

The Complete Comparison & Decision Guide

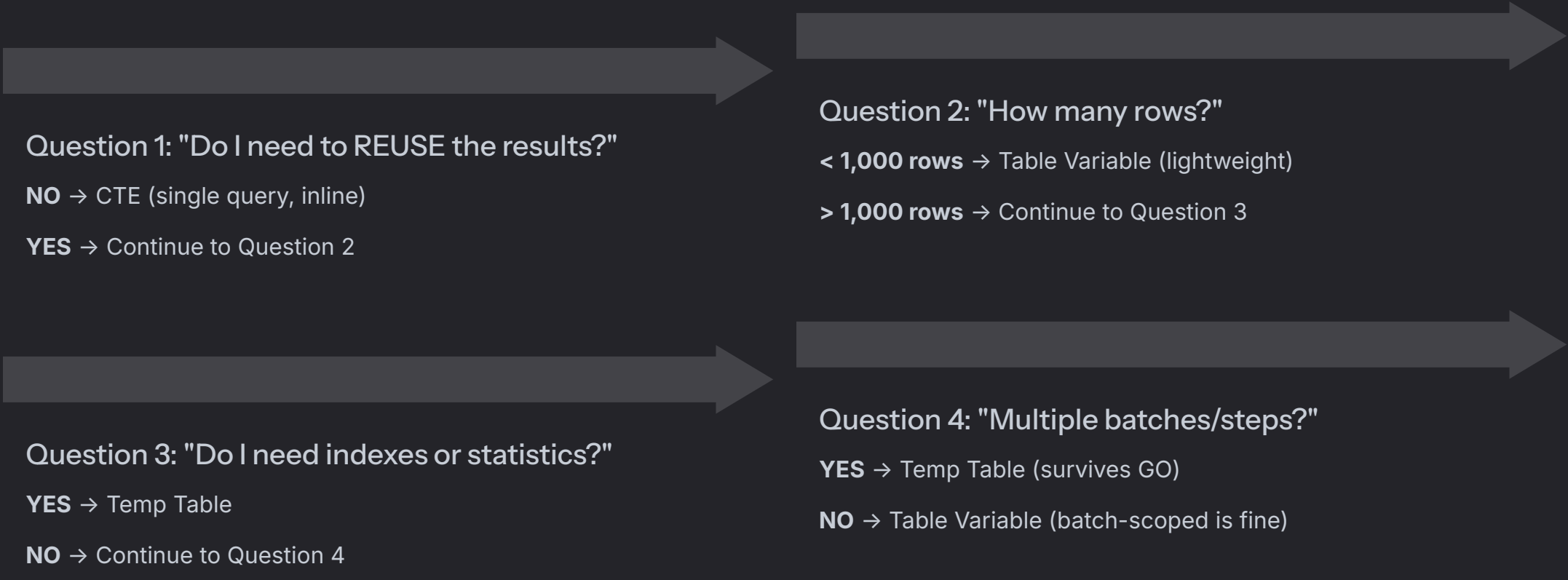
The Three Options

<div></div> <div>Option 1: Temp Tables (#TempTable)</div> <div><ul style="list-style-type: none">Physical storage in tempdbSession-scopedCan add indexesHas statisticsBest for large datasets & multi-step workflows</div>	<div></div> <div>Option 2: Table Variables (@TableVariable)</div> <div><ul style="list-style-type: none">In-memory structure (actually in tempdb too!)Batch-scopedLimited indexingNo statisticsBest for small datasets & simple staging</div>	<div></div> <div>Option 3: CTEs (Common Table Expressions)</div> <div><ul style="list-style-type: none">Query-scopedNo storage (inline with query)Calculated on demandReadable syntaxBest for single-query readability</div>
--	--	---





Feature Comparison Table

Feature	Temp Table	Table Variable	CTE
Storage location	tempdb	tempdb	None
Lifetime	Session	Batch	Query
Add indexes after create	✔ Yes	✗ No	N/A
Statistics	✔ Yes	✗ No	✗ No
Reuse across batches	✔ Yes	✗ No	✗ No
Transactions	✔ Yes	✔ Yes	✔ Yes
Explicit DROP needed	Optional	No	No
Ideal dataset size	> 1,000	< 1,000	Any
Performance (large data)	Excellent	Poor	Good

When to Use Each - Decision Tree



Real-World Scenarios

<div></div> <div>Scenario 1: Month-end reconciliation (10 steps, large data)</div> <div>Answer: TEMP TABLES</div> <div>Reason: Multi-step, large datasets, need indexes</div>	<div></div> <div>Scenario 2: Quick parameter list for IN clause (20 IDs)</div> <div>Answer: TABLE VARIABLE</div> <div>Reason: Small data, single use, simple</div>
<div></div> <div>Scenario 3: Readable subquery for rankings</div> <div>Answer: CTE</div> <div>Reason: Single query, no reuse needed</div>	<div></div> <div>Scenario 4: Data quality report (identify → categorise → report)</div> <div>Answer: TEMP TABLES</div> <div>Reason: Multi-step, persists across batches</div>

CTE Refresher – The Readable Option

```
WITH SalesSummary AS (  
    SELECT  
        account_id,  
        SUM(amount) AS total_sales  
    FROM transactions  
    WHERE transaction_date >= '2024-12-01'  
    GROUP BY account_id  
)  
SELECT  
    a.customer_name,  
    s.total_sales  
FROM accounts a  
JOIN SalesSummary s ON a.account_id = s.account_id  
WHERE s.total_sales > 10000;
```

CTE = Inline, readable, single query

Perfect for organising complex SELECT statements!

- When CTE is NOT enough:
- ✗ Need to query the CTE multiple times (recalculated each time!)
 - ✗ Need to store for later processing
 - ✗ Dataset too large for inline processing

Temp Tables vs CTEs – The Reuse Test

<div>With CTE (BAD - recalculates 3 times):</div> <div>WITH Summary AS (SELECT ... complex calculation ...) SELECT * FROM Summary WHERE condition1; -- Calculate once SELECT * FROM Summary WHERE condition2; -- Calculate again! SELECT * FROM Summary WHERE condition3; -- Calculate third time!</div> <div>Inefficient! Complex query runs 3 times!</div>	<div>With Temp Table (GOOD - calculate once):</div> <div>CREATE TABLE #Summary AS SELECT ... complex calculation ...; SELECT * FROM #Summary WHERE condition1; -- Use stored result SELECT * FROM #Summary WHERE condition2; -- Use stored result SELECT * FROM #Summary WHERE condition3; -- Use stored result</div> <div>Efficient! Complex query runs once!</div>
--	--

 **Rule: Reuse results → Temp table, not CTE!**

Global Temp Tables (##GlobalTemp)

Special case: Cross-session sharing

```
CREATE TABLE ##SharedData (  
    id INT,  
    value NVARCHAR(100)  
);
```

- Visible to ALL sessions! Prefix with ## (double hash)
- Use case:
- Session A creates ##Lookup table
 - Sessions B, C, D all read ##Lookup
 - Shared reference data across team
- Caution:
- Dropped when LAST session using it disconnects
 - Naming conflicts possible (other users may create same name)
 - Use sparingly!
- Best practice: Permanent table better for true shared data

Performance Tips

- Tip 1: Index temp tables for large datasets

CREATE INDEX IX_AccountID ON #Transactions(account_id);

Massive performance boost for JOINS and WHERE!
- Tip 2: Use table variables for < 1,000 rows

Less overhead, faster for small data
- Tip 3: Use CTEs for readable one-time queries

No storage overhead, clean syntax
- Tip 4: DROP temp tables when done (in long scripts)

Frees up tempdb space immediately
- Tip 5: Monitor tempdb usage

Large temp tables can fill tempdb

Watch for "tempdb is full" errors

Chaining Temp Tables for Complex Processing

The Month-End Reconciliation Challenge

01	02
Extract 15,000 December transactions	Calculate daily running balances per account
03	04
Identify overdraft instances	Flag suspicious large transactions
05	06
Calculate fee revenue	Match deposits to withdrawals
07	08
Identify unmatched transactions	Calculate interest accruals
09	10
Generate variance report	Final reconciliation summary

Each step feeds the NEXT step!

The Temp Table Chain Pattern

```
Step 1 → #Step1_Transactions
↓
Step 2 reads #Step1, writes #Step2_DailyBalances
↓
Step 3 reads #Step2, writes #Step3_Overdrafts
↓
Step 4 reads #Step1, writes #Step4_SuspiciousTransactions
↓
Step 5 reads #Step1 + #Step2, writes #Step5_FeeRevenue
... continues through Step 10
↓
Final: Read ALL temp tables, generate report!
```

Automated pipeline!

Code Structure - The Template

```
-- STEP 1: Extract base data
CREATE TABLE #Step1_Transactions (...);
INSERT INTO #Step1_Transactions SELECT ... FROM source;
CREATE INDEX IX_Account ON #Step1_Transactions(account_id);

-- STEP 2: Transform/Calculate
CREATE TABLE #Step2_Calculated (...);
INSERT INTO #Step2_Calculated SELECT ... FROM #Step1_Transactions;
CREATE INDEX IX_Date ON #Step2_Calculated(date_column);

-- STEP 3: Further processing
CREATE TABLE #Step3_Results (...);
INSERT INTO #Step3_Results SELECT ... FROM #Step2_Calculated;

-- FINAL: Generate report
SELECT ...
FROM #Step1_Transactions t1
JOIN #Step2_Calculated t2 ON ...
JOIN #Step3_Results t3 ON ...;

-- CLEANUP
DROP TABLE #Step1_Transactions;
DROP TABLE #Step2_Calculated;
DROP TABLE #Step3_Results;
```

Clear, organised, maintainable!

Real NAB Example - Step by Step

STEP 1: Extract Transactions

```
CREATE TABLE #Step1_Transactions (
    transaction_id INT,
    account_id INT,
    transaction_date DATETIME,
    amount DECIMAL(15,2)
);

INSERT INTO #Step1_Transactions
SELECT
    transaction_id,
    account_id,
    transaction_date,
    amount
FROM transactions
WHERE transaction_date >= '2024-12-01'
    AND transaction_date < '2025-01-01';

CREATE INDEX IX_Account ON #Step1_Transactions(account_id);
```

Result: 15,248 December transactions stored

STEP 2: Daily Running Balances

```
CREATE TABLE #Step2_DailyBalances (
    account_id INT,
    balance_date DATE,
    running_balance DECIMAL(15,2)
);

INSERT INTO #Step2_DailyBalances
SELECT
    t.account_id,
    CAST(t.transaction_date AS DATE),
    a.opening_balance + SUM(SUM(t.amount)) OVER (
        PARTITION BY t.account_id
        ORDER BY CAST(t.transaction_date AS DATE)
    ) AS running_balance
FROM #Step1_Transactions t
JOIN accounts a ON t.account_id = a.account_id
GROUP BY t.account_id, CAST(t.transaction_date AS DATE), a.opening_balance;
```

Result: Daily balances calculated using previous step's data!

STEP 3: Identify Overdrafts

```
CREATE TABLE #Step3_Overdrafts (
    account_id INT,
    overdraft_date DATE,
    overdraft_amount DECIMAL(15,2)
);

INSERT INTO #Step3_Overdrafts
SELECT
    account_id,
    balance_date,
    running_balance
FROM #Step2_DailyBalances
WHERE running_balance < 0;
```

Result: 23 overdraft instances identified!

Uses STEP 2 results!

STEP 4: Flag Suspicious Transactions

```
CREATE TABLE #Step4_Suspicious (
    transaction_id INT,
    amount DECIMAL(15,2),
    flag_reason NVARCHAR(100)
);

INSERT INTO #Step4_Suspicious
SELECT
    transaction_id,
    amount,
    CASE
        WHEN ABS(amount) > 10000 THEN 'Large transaction (>$10K)'
        WHEN ABS(amount) > 5000 THEN 'Medium alert (~$5K)'
    END
FROM #Step1_Transactions
WHERE ABS(amount) > 5000;
```

Result: 47 suspicious transactions flagged!

Uses STEP 1 results!

Final Report - Combining All Steps

```
SELECT
    a.account_number,
    a.customer_name,
    a.opening_balance,
    SUM(t.amount) AS total_movement,
    a.opening_balance + SUM(t.amount) AS closing_balance,
    COUNT(DISTINCT o.overdraft_date) AS overdraft_days,
    COUNT(DISTINCT s.transaction_id) AS suspicious_count
FROM accounts a
LEFT JOIN #Step1_Transactions t ON a.account_id = t.account_id
LEFT JOIN #Step3_Overdrafts o ON a.account_id = o.account_id
LEFT JOIN #Step4_Suspicious s ON a.account_id = s.account_id
GROUP BY a.account_number, a.customer_name, a.opening_balance
ORDER BY closing_balance DESC;
```

One query pulls from ALL temp tables!

Complete reconciliation report! 🟢

Wrapping in a Stored Procedure

```
CREATE PROCEDURE MonthEndReconciliation
    @Year INT,
    @Month INT
AS
BEGIN
    -- Calculate date range
    DECLARE @StartDate DATE = DATEFROMPARTS(@Year, @Month, 1);
    DECLARE @EndDate DATE = DATEADD(MONTH, 1, @StartDate);

    -- STEP 1: Extract
    CREATE TABLE #Step1_Transactions (...);
    INSERT INTO #Step1_Transactions ...

    -- STEP 2: Calculate
    CREATE TABLE #Step2_Balances (...);
    INSERT INTO #Step2_Balances ...

    -- STEP 3-10: Continue processing
    ...

    -- FINAL: Report
    SELECT ... FROM all temp tables;

    -- CLEANUP
    DROP TABLE #Step1_Transactions;
    DROP TABLE #Step2_Balances;
    ...


END
GO
```

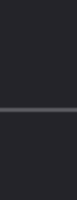
Execute with:


```
EXEC MonthEndReconciliation @Year = 2024, @Month = 12;
```

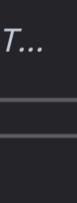
Reusable for any month! 🔄


Benefits of This Pattern

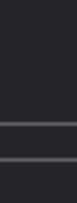
**Organised**
Each step clearly labelled

**Debuggable**
Run steps individually to find issues

**Maintainable**
Change one step without affecting others

**Performant**
Indexes on temp tables speed up joins

**Auditable**
Can save temp table results for review

**Reusable**
Wrap in procedure, call with parameters

The Friday Evening Transformation

BEFORE (Manual Excel):

- 5:00 PM: Start
- 5:15 PM: Query 1 → Excel file 1
- 5:45 PM: VLOOKUP errors, fix formulas
- 6:20 PM: Query 2 → Excel file 2
- 7:15 PM: More errors, more fixes
- 8:30 PM: Finally finish step 10

Total: 3.5 hours, exhausted, errors

AFTER (Temp Table Automation):

- 5:30 PM: Write script with temp tables (one time!)
- 6:00 PM: Test with sample data
- 6:30 PM: EXEC MonthEndReconciliation @Year = 2024, @Month = 12
- 6:40 PM: Complete! Perfect results!

Total: 10 minutes execution (after initial 1-hour setup)

Future months: 10 minutes forever! 🚀

PRACTICE EXERCISES & MASTERY SUMMARY

PRACTICE EXERCISES

1

Basic Temp Table ★

- Create temp table #HighValueAccounts
- Insert accounts with opening_balance > \$50,000
- Query and display all columns
- Manually DROP the temp table

Hint: CREATE TABLE #Name (...); INSERT INTO #Name SELECT...

2

Table Variable for Small Dataset ★★

- Declare @TransactionSummary table variable
- Columns: account_id, total_amount, tx_count
- Populate with aggregated transaction data by account
- Display accounts with tx_count > 10

Hint: DECLARE @Name TABLE (...); INSERT INTO @Name...

3

Multi-Step Workflow ★★★

- Step 1: Create #AllTransactions with Dec 2024 data
- Step 2: Create #DailyTotals aggregating by date from #AllTransactions
- Step 3: Query #DailyTotals to show daily trends
- Clean up both temp tables

Hint: Each step reads from previous temp table

4

Indexed Temp Table Performance ★★★★

- Create temp table with all transactions
- Add index on account_id column
- Query for specific account's transactions
- Compare execution plan with/without index

Hint: CREATE INDEX IX_Name ON #Table(column); View execution plan

5

Complete Reconciliation Procedure ★★★★★

- Create procedure: QuarterlyReconciliation
- Parameters: @Year INT, @Quarter INT
- Calculate quarter start/end dates
- Use temp tables for multi-step processing
- Return quarterly summary by account

*Hint: @StartMonth = (@Quarter - 1) * 3 + 1*

All solutions in SQL file! Try first, then check answers.

WHAT YOU MASTERED

Technical Skills:

- ✓ Temp table creation and management (#TempTable)
- ✓ Table variable declaration (@Table/Variable)
- ✓ Global temp tables (##GlobalTemp)
- ✓ Session vs batch vs query scope
- ✓ Indexing temp tables for performance
- ✓ Multi-step workflow patterns
- ✓ When to use each storage type
- ✓ Production procedure patterns

Business Skills:

- ✓ Month-end reconciliation automation
- ✓ Complex multi-step processing
- ✓ Financial reporting workflows
- ✓ Data quality validation patterns
- ✓ Audit trail creation
- ✓ Intermediate results storage

Career Skills:

- ✓ Enterprise-grade workflow design
- ✓ Performance optimisation techniques
- ✓ Production-ready code structure
- ✓ Debugging multi-step processes
- ✓ Code organisation best practices
- ✓ Interview-ready for complex SQL scenarios

Real-World Applications

Banking (NAB example):

- Month-end reconciliation
- Daily balance calculations
- Overdraft detection
- Suspicious transaction flagging

Retail:

- Inventory reconciliation
- Sales aggregation pipelines
- Customer segmentation workflows
- Promotion effectiveness analysis

Manufacturing:

- Production quality checks
- Multi-stage inspection reports
- Defect tracking workflows
- Equipment maintenance schedules

Healthcare:

- Patient billing reconciliation
- Claims processing workflows
- Multi-step diagnosis validation
- Treatment outcome tracking

The Friday Evening Miracle

BEFORE Temp Tables:

- 5:00 PM: Month-end reconciliation needed
- 5:00-8:30 PM: Manual Excel nightmare (10 files, 3.5 hours)
- Errors:** VLOOKUP mistakes, formula errors, copy-paste failures
- Error rate:** 15% (requires weekend fixes)
- Stress level:** Maximum 😫
- Weekend:** Ruined fixing errors

AFTER Temp Tables:

- 5:00 PM: Month-end reconciliation needed
- 5:30 PM: One-time script development (temp table chain)
- 6:00 PM: Testing complete
- 6:30 PM: EXEC MonthEndReconciliation @Year = 2024, @Month = 12
- 6:40 PM: Complete! Perfect! 🟢
- Error rate:** 0%
- Stress level:** Zero 😊
- Weekend:** Free!

95%

Time Reduction
3.5 hours → 10 minutes

0%

Error Rate
Down from 15%

\$4,200+

Annual Savings
42 hours × \$100/hr per analyst

Decision Guide Summary

Question: What should I use?

< 100 rows, single use → CTE (inline query)	< 1,000 rows, single batch → TABLE VARIABLE (@Variable)
> 1,000 rows OR multi-step → TEMP TABLE (#TempTable)	Cross-session sharing → GLOBAL TEMP TABLE (##GlobalTemp)

Performance:

Need indexes? → TEMP TABLE

Scope:

Need across batches? → TEMP TABLE

Simplicity:

Quick staging? → TABLE VARIABLE

Readability:

Single query? → CTE

Best Practices Checklist

- ✓ Use temp tables for large datasets (>1,000 rows)
- ✓ Add indexes to temp tables for performance
- ✓ Use table variables for small datasets (<1,000 rows)
- ✓ Use CTEs for single-query readability
- ✓ DROP temp tables when done (in long scripts)
- ✓ Name temp tables descriptively (#Step1_Transactions)
- ✓ Document multi-step workflows with comments
- ✓ Wrap complex workflows in stored procedures
- ✓ Test with sample data before production
- ✓ Monitor tempdb usage in production

Interview Value

"How would you handle complex multi-step processing?"
→ Temp tables!

"What's the difference between temp table and table variable?"
→ Scope, performance, statistics!

"When would you use a CTE vs temp table?"
→ Single use vs reuse!

"How do you optimise temp table queries?"
→ Indexes!

📌 Temp tables/table variables appear in 65% of advanced SQL interviews!

SQL Mastery Journey

- Videos 1-3: Foundation (SELECT, JOIN, GROUP BY)
- Videos 4-5: Intermediate (Time, Subqueries)
- Video 6: CTEs (Readable SQL)
- Video 7: Window Functions (Rankings)
- Video 8: PIVOT (Matrix reports)
- Video 9: Stored Procedures (Automation)
- Video 10: Temp Tables & Variables ← COMPLEX WORKFLOWS MASTERED! 🟢

Next Video Preview

Video 11: Transactions & Error Handling

Commonwealth Bank - Data Integrity & ACID Principles

Duration: 18 minutes

Learn:

- BEGIN TRANSACTION, COMMIT, ROLLBACK
- ACID principles (Atomicity, Consistency, Isolation, Durability)
- Savepoints for partial rollback
- Transaction isolation levels
- Deadlock handling

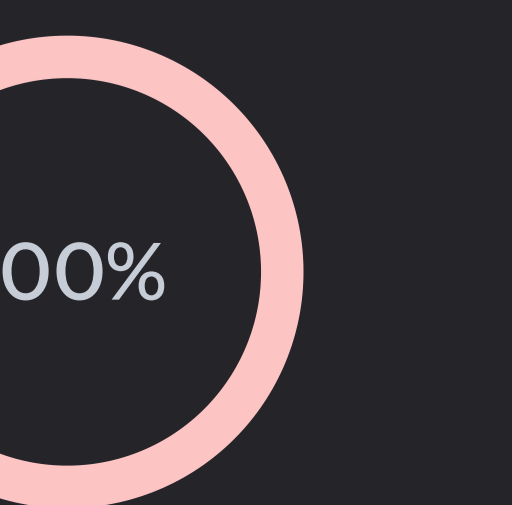
Business scenario:

Banking fund transfers (must be atomic!)

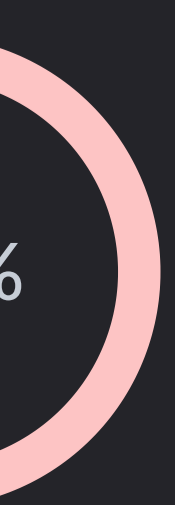
Problem: Transfer \$1,000 from Account A to Account B

Requirement: BOTH debit AND credit must succeed (or both fail!)

Solution: Transactions ensure data integrity



You've Unlocked Complex Workflow Automation!



100%

Can chain multi-step processes

100%

Can optimise with indexes

100%

Can choose right storage type

100%

Can automate month-end reconciliations

Career advancement: Senior Analyst level! 📈

5 more videos to complete the course!