Snowflake Advanced Learning – Phase 2

Date: December 12, 2025
Learning Path: Data Engineering – Files, Stages, Tables & Views

---

Table of Contents

1. File Formats (CSV, JSON, AVRO, Parquet, ORC)
2. Data Lifecycle (Storage, Querying, Removal)
3. Stages (Internal & External)
4. Tables (Permanent, External, Iceberg, Transient, Temporary)
5. Views (Standard, Secure, Materialized)
6. Hands-on Practices
7. Next Steps

---

1. File Formats (CSV, JSON, AVRO, Parquet, ORC)

Overview

File formats define how data is structured, encoded, and parsed when loading/unloading data in Snowflake. Different formats suit different use cases.

Supported Formats

❖ **CSV (Comma-Separated Values)**

- Best For: Tabular data, legacy systems, human-readable exports
- Characteristics: Plain text, line-delimited rows, column-delimited values
- Pros: Simple, widely supported, human-readable
- Cons: No native schema support, schema inference needed

sql

```
CREATE OR REPLACE FILE FORMAT FF_CSV
  TYPE = CSV
  FIELD_DELIMITER = ','
  RECORD_DELIMITER = '\n'
  SKIP_HEADER = 1
  FIELD_OPTIONALLY_ENCLOSED_BY = '"'
  NULL_IF = ('NULL', 'null', '');
```

Example CSV data:

text

order_id,customer_id,amount,order_date

1,101,500.50,2025-01-15

2,102,750.00,2025-01-16

3,103,1200.25,2025-01-17

❖ **JSON (JavaScript Object Notation)**

- Best For: Semi-structured data, APIs, nested hierarchies
- Characteristics: Key-value pairs, nested objects/arrays
- Pros: Flexible schema, supports nesting, widely used in APIs
- Cons: Larger file size than columnar formats

sql

CREATE OR **REPLACE FILE** FORMAT FF_JSON

TYPE = JSON

STRIP_OUTER_ARRAY = TRUE

COMPRESSION = AUTO;

Example JSON data:

json

```
[
 {"order_id": 1, "customer_id": 101, "amount": 500.50, "order_date": "2025-01-15"},
 {"order_id": 2, "customer_id": 102, "amount": 750.00, "order_date": "2025-01-16"},
 {"order_id": 3, "customer_id": 103, "amount": 1200.25, "order_date": "2025-01-17"}
]
```

❖ **AVRO (Apache Avro)**

- Best For: Schema evolution, serialization in streaming pipelines (Kafka, etc.)
- Characteristics: Binary format with schema metadata embedded
- Pros: Schema versioning, compact, handles schema evolution
- Cons: Requires schema definition, less human-readable

sql

```
CREATE OR REPLACE FILE FORMAT FF_AVRO
  TYPE = AVRO
  COMPRESSION = SNAPPY;
```

❖ **Parquet (Apache Parquet)**

- Best For: Analytics, big data, columnar storage
- Characteristics: Columnar binary format with compression
- Pros: Highly compressed, fast column access, efficient for analytics
- Cons: Not human-readable, requires schema

sql

```
CREATE OR REPLACE FILE FORMAT FF_PARQUET
  TYPE = PARQUET
  COMPRESSION = SNAPPY;
```

❖ **ORC (Optimized Row Columnar)**

- Best For: Large-scale analytics, Hadoop ecosystems
- Characteristics: Columnar format with built-in compression and statistics
- Pros: High compression, built-in indexing, statistics
- Cons: Less universal support than Parquet

sql

```
CREATE OR REPLACE FILE FORMAT FF_ORC
  TYPE = ORC
  COMPRESSION = ZSTD;
```

Creating & Using File Formats

sql

```
-- Create a custom CSV format
CREATE OR REPLACE FILE FORMAT FF_SALES
  TYPE = CSV
  FIELD_DELIMITER = '|'
  RECORD_DELIMITER = '\n'
  SKIP_HEADER = 1
  NULL_IF = ('NA', 'NULL', '');
```

-- Use with a stage

CREATE OR **REPLACE** STAGE STG_SALES

FILE_FORMAT = FF_SALES;

-- Or specify format in COPY INTO

COPY **INTO** SALES_TABLE

FROM @STG_SALES

FILE_FORMAT = (**TYPE** = 'CSV' FIELD_DELIMITER = ',')

ON_ERROR = 'SKIP_FILE';

---

2. Data Lifecycle (Storage, Querying, Removal)

Storage Phase

What happens:

- Data is loaded into Snowflake tables from stages or external sources.
- Data is compressed and partitioned into micro-partitions.
- Encryption is applied automatically.
- Metadata is stored in the Cloud Services layer.

sql

-- Load data into table

COPY **INTO** SALES_TABLE

FROM @STG_SALES

FILE_FORMAT = FF_CSV

ON_ERROR = 'ABORT_STATEMENT';

Querying Phase

What happens:

- Virtual warehouses execute SQL queries.
- Metadata pruning identifies relevant micro-partitions.
- Only required data is scanned (columnar access).
- Results are returned to the user.

sql

```
-- Simple query
SELECT CUSTOMER_ID, SUM(AMOUNT) AS TOTAL
FROM SALES_TABLE
WHERE ORDER_DATE >= '2025-01-01'
GROUP BY CUSTOMER_ID;

-- With TIME TRAVEL (query historical data)
SELECT * FROM SALES_TABLE
AT (TIMESTAMP => '2025-01-10 10:00:00'::TIMESTAMP_NTZ);

-- Before a specific statement execution
SELECT * FROM SALES_TABLE
BEFORE (STATEMENT => 'c81e728d9f35d5d9');
```

Time Travel Retention:

- Permanent tables: 90 days (default)
- Transient tables: 1 day
- Temporary tables: 1 day

Removal Phase

Methods:

- DELETE – Remove specific rows
- TRUNCATE – Remove all rows (faster)
- DROP – Remove table/schema/database
- Automatic purge after Time Travel retention expires

sql

```
-- Delete specific rows
DELETE FROM SALES_TABLE
WHERE ORDER_DATE < '2023-01-01';

-- Truncate entire table (faster, cannot be rolled back after commit)
TRUNCATE TABLE SALES_TABLE;
```

-- Drop table

DROP TABLE SALES_TABLE;

-- Restore dropped table (within 90 days for permanent tables)

UNDROP **TABLE** SALES_TABLE;

Data Retention Policy

sql

-- Set custom retention for a table

ALTER TABLE SALES_TABLE **SET** DATA_RETENTION_TIME_IN_DAYS = 30;

-- Extend retention for compliance

ALTER TABLE SENSITIVE_DATA **SET** DATA_RETENTION_TIME_IN_DAYS = 365;

---

3. Stages (Internal & External)

Stage Types

User Stage (@~)

- Private stage per user
- 160 GB per user
- Used for personal file uploads

sql

-- Upload to user stage

PUT **file**:*///home/user/data.csv @~ AUTO_COMPRESS = TRUE;*

-- Query from user stage

COPY **INTO** TABLE_NAME **FROM** @~ FILE_FORMAT = FF_CSV;

Table Stage (@%table_name)

- Associated with a specific table
- Useful for table-specific landing data
- Namespace prevents conflicts

sql

-- Upload to table stage

PUT **file**:*///data/orders.csv @%ORDERS_TABLE;*

-- Load from table stage

COPY **INTO** ORDERS_TABLE

FROM @%ORDERS_TABLE

FILE_FORMAT = FF_CSV;

Internal Named Stage

- Permanent, named storage inside Snowflake
- Shared across users with proper grants
- Can be versioned and accessed from anywhere in the database

sql

-- Create internal stage

CREATE OR **REPLACE** STAGE STG_LANDING

FILE_FORMAT = FF_CSV

ENCRYPTION = (**TYPE** = 'SNOWFLAKE_SSE');

-- Upload files

PUT **file**:*///data/sales/*.csv @STG_LANDING AUTO_COMPRESS = TRUE;*

-- List files in stage

LIST @STG_LANDING;

-- Copy to table

COPY **INTO** SALES_TABLE

FROM @STG_LANDING

PATTERN = '.*csv'

FILE_FORMAT = FF_CSV;

External Stage

- References cloud storage (S3, Azure Blob, GCS)

- Data stays outside Snowflake
- Useful for data lakes and shared infrastructure

sql

```
-- Create external stage (AWS S3)
CREATE OR REPLACE STAGE STG_S3_LAKE
  URL = 's3://my-data-lake/raw/'
  CREDENTIALS = (
    AWS_KEY_ID = 'AKIA...'
    AWS_SECRET_KEY = '...'
  )
  ENCRYPTION = (
    TYPE = 'SSE_S3'
  )
  FILE_FORMAT = FF_PARQUET;

-- Create external stage (Azure Blob)
CREATE OR REPLACE STAGE STG_AZURE_LAKE
  URL = 'azure://storageaccount.blob.core.windows.net/container/path/'
  CREDENTIALS = (
    AZURE_SAS_TOKEN = 'sp=...'
  )
  FILE_FORMAT = FF_PARQUET;

-- Query external data
SELECT * FROM SALES_TABLE
WHERE DATA_SOURCE = (
  SELECT * FROM @STG_S3_LAKE/orders/
);
```

Stage Operations

sql

```
-- List files in stage
```

```
LIST @STG_LANDING;

-- Remove files from stage
REMOVE @STG_LANDING/processed/;

-- Describe stage
DESCRIBE STAGE STG_LANDING;

-- Grant permissions on stage
GRANT READ ON STAGE STG_LANDING TO ROLE DATA_ENGINEER;
```

---

4. Tables (Permanent, External, Iceberg, Transient, Temporary)

Permanent Table (Default)

- Fully managed by Snowflake
- 90-day Time Travel + 7-day Fail-safe
- Supports all features (clustering, streams, etc.)
- Highest cost, best for production data

sql

```
CREATE OR REPLACE TABLE SALES_ORDERS (
  ORDER_ID NUMBER PRIMARY KEY,
  CUSTOMER_ID NUMBER,
  ORDER_AMOUNT NUMBER(10,2),
  ORDER_DATE DATE,
  CREATED_AT TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP()
);

-- Restore from Time Travel
UNDROP TABLE SALES_ORDERS;
```

Transient Table

- No Fail-safe (only Time Travel for 1 day)
- Cheaper than permanent tables

- Good for intermediate/staging data

sql

```
CREATE OR REPLACE TRANSIENT TABLE STAGING_ORDERS (
  ORDER_ID NUMBER,
  CUSTOMER_ID NUMBER,
  ORDER_AMOUNT NUMBER(10,2)
);

-- Set data retention
ALTER TABLE STAGING_ORDERS SET DATA_RETENTION_TIME_IN_DAYS = 1;
```

Temporary Table

- Lives only for the current session
- Automatically dropped when session ends
- Private to the session
- Fast for session-specific operations

sql

```
CREATE TEMPORARY TABLE SESSION_ORDERS AS
SELECT * FROM SALES_ORDERS
WHERE ORDER_DATE >= CURRENT_DATE - 7;

-- Use in session
SELECT * FROM SESSION_ORDERS;

-- Automatically dropped at session end
```

External Table

- References data stored outside Snowflake (S3, Azure, GCS)
- Not physically stored in Snowflake
- Read-only (no INSERT/UPDATE/DELETE)
- Useful for data lake queries without ingesting

sql

```
CREATE OR REPLACE EXTERNAL TABLE EXT_SALES_ORDERS (
```

```
  ORDER_ID NUMBER,
  CUSTOMER_ID NUMBER,
  ORDER_AMOUNT NUMBER(10,2),
  ORDER_DATE DATE
)
WITH LOCATION = @STG_S3_LAKE/orders/
FILE_FORMAT = FF_PARQUET
AUTO_REFRESH = FALSE;

-- Query external data
SELECT * FROM EXT_SALES_ORDERS LIMIT 100;

-- Refresh metadata
ALTER TABLE EXT_SALES_ORDERS REFRESH;
```

Iceberg Table

- Uses Apache Iceberg open format
- Interoperable with other engines (Spark, Trino, etc.)
- Copy-on-write updates (ACID compliant)
- Better for data sharing across ecosystems

sql

```
CREATE OR REPLACE ICEBERG TABLE ORDERS_ICEBERG (
  ORDER_ID NUMBER PRIMARY KEY,
  CUSTOMER_ID NUMBER,
  ORDER_AMOUNT NUMBER(10,2),
  ORDER_DATE DATE
);

-- Insert data (creates new version)
INSERT INTO ORDERS_ICEBERG VALUES (1, 101, 500.50, '2025-01-15');

-- Update triggers version change
```

UPDATE ORDERS_ICEBERG

SET ORDER_AMOUNT = 525.50

WHERE ORDER_ID = 1;

-- Query specific version

SELECT * **FROM** ORDERS_ICEBERG

FOR TIMESTAMP AS OF '2025-01-15 10:00:00';

Table Comparison

Feature	Permanent	Transient	Temporary	External	Iceberg
Time Travel	90 days	1 day	1 day	N/A	90 days
Fail-safe	7 days	None	None	N/A	7 days
Cost	High	Medium	Medium	Low	High
Mutable	Yes	Yes	Yes	No	Yes
Clustering	Yes	Yes	Yes	No	Yes
Streams	Yes	Yes	No	No	Yes
Interop	No	No	No	Limited	Yes

---

5. Views (Standard, Secure, Materialized)

Standard View

- Logical layer over base tables
- Always reflects current underlying data
- Security is not row-level (visible to anyone with SELECT)
- Minimal overhead

sql

```
CREATE OR REPLACE VIEW V_SALES_SUMMARY AS
SELECT CUSTOMER_ID,
    COUNT(*) AS ORDER_COUNT,
    SUM(ORDER_AMOUNT) AS TOTAL_AMOUNT,
    AVG(ORDER_AMOUNT) AS AVG_AMOUNT,
    MAX(ORDER_DATE) AS LAST_ORDER_DATE
FROM SALES_ORDERS
GROUP BY CUSTOMER_ID;

-- Query the view
SELECT * FROM V_SALES_SUMMARY WHERE TOTAL_AMOUNT > 10000;

-- Show view definition
SHOW VIEWS;
DESCRIBE VIEW V_SALES_SUMMARY;
```

Secure View

- Hides underlying table structure and query logic
- Used for sensitive aggregations or data masking
- Recommended for Secure Data Sharing
- Slight performance overhead due to internal optimization restrictions

sql

```
-- Secure view hides underlying tables from users
CREATE OR REPLACE SECURE VIEW V_SALES_MASKED AS
SELECT CUSTOMER_ID,
    TOTAL_AMOUNT
FROM V_SALES_SUMMARY
WHERE TOTAL_AMOUNT > 500;

-- Users cannot see:
-- - Underlying table structure
```

-- - Query logic inside the view

-- - Related tables

-- Share securely with other accounts

GRANT SELECT ON VIEW V_SALES_MASKED **TO SHARE** sales_share;

When to use Secure Views:

- Sensitive data aggregations
- Secure data sharing with external accounts
- Hiding query complexity
- Compliance requirements (PII masking)

Materialized View

- Physically stores query results in a table-like structure
- Auto-updates when underlying tables change
- Fast queries on pre-computed results
- Uses storage and compute for maintenance
- Ideal for repeated expensive aggregations

sql

-- Create materialized view

CREATE OR **REPLACE** MATERIALIZED **VIEW** MV_SALES_DAILY **AS**

SELECT ORDER_DATE,

COUNT(*) **AS** ORDER_COUNT,

SUM(ORDER_AMOUNT) **AS** TOTAL_SALES,

AVG(ORDER_AMOUNT) **AS** AVG_SALE

FROM SALES_ORDERS

GROUP BY ORDER_DATE;

-- Query materialized view (very fast)

SELECT * **FROM** MV_SALES_DAILY

WHERE ORDER_DATE >= **CURRENT_DATE** - 30;

-- Refresh materialized view manually

ALTER MATERIALIZED **VIEW** MV_SALES_DAILY REFRESH;

-- Drop materialized view

DROP MATERIALIZED **VIEW** MV_SALES_DAILY;

Materialized View Refresh:

- Automatic (default): Refreshes when underlying data changes
- Manual: Use ALTER MATERIALIZED VIEW ... REFRESH
- On Demand: Controlled via AUTO_REFRESH_ON_CHANGE = FALSE

View Comparison

Feature	**Standard**	**Secure**	**Materialized**
Query Logic Hidden	No	Yes	No
Real-time Data	Yes	Yes	No (refreshed)
Performance	Depends on query	Slightly slower	Very fast
Storage	None	None	Yes
Update Cost	None	None	Yes (refresh)
Best For	Simple queries	Sensitive data	Heavy aggregations

---

6. Hands-on Practices

Practice 1: Create & Use Different File Formats

sql

-- Create CSV format with pipe delimiter

CREATE OR **REPLACE FILE** FORMAT FF_PIPE

TYPE = CSV

FIELD_DELIMITER = '|'

SKIP_HEADER = 1;

```
-- Create JSON format
CREATE OR REPLACE FILE FORMAT FF_JSON_DATA
  TYPE = JSON
  STRIP_OUTER_ARRAY = TRUE;

-- Create Parquet format
CREATE OR REPLACE FILE FORMAT FF_PARQUET_DATA
  TYPE = PARQUET
  COMPRESSION = SNAPPY;

-- List all formats
SHOW FILE FORMATS;
```

Practice 2: Internal & External Stages

sql

```
-- Create internal stage
CREATE OR REPLACE STAGE STG_INTERNAL
  FILE_FORMAT = FF_CSV;

-- Create external stage (S3)
CREATE OR REPLACE STAGE STG_S3_EXTERNAL
  URL = 's3://my-bucket/data/'
  CREDENTIALS = (AWS_KEY_ID = '...' AWS_SECRET_KEY = '...')
  FILE_FORMAT = FF_PARQUET_DATA;

-- Upload to internal stage
PUT file:///local/data.csv @STG_INTERNAL;

-- List files
LIST @STG_INTERNAL;

-- Grant access
```

```
GRANT READ ON STAGE STG_INTERNAL TO ROLE DATA_ENGINEER;
```

Practice 3: Create Different Table Types

sql

```
-- Permanent table (production)
CREATE OR REPLACE TABLE PROD_ORDERS (
  ORDER_ID NUMBER PRIMARY KEY,
  CUSTOMER_ID NUMBER,
  AMOUNT NUMBER(10,2),
  ORDER_DATE DATE
);

-- Transient table (staging)
CREATE OR REPLACE TRANSIENT TABLE STAGING_ORDERS (
  ORDER_ID NUMBER,
  CUSTOMER_ID NUMBER,
  AMOUNT NUMBER(10,2)
);

-- External table (data lake)
CREATE OR REPLACE EXTERNAL TABLE LAKE_ORDERS (
  ORDER_ID NUMBER,
  CUSTOMER_ID NUMBER,
  AMOUNT NUMBER(10,2),
  ORDER_DATE DATE
)
WITH LOCATION = @STG_S3_EXTERNAL
FILE_FORMAT = FF_PARQUET_DATA;

-- Iceberg table (interoperable)
CREATE OR REPLACE ICEBERG TABLE ICEBERG_ORDERS (
  ORDER_ID NUMBER PRIMARY KEY,
```

```
 CUSTOMER_ID NUMBER,
 AMOUNT NUMBER(10,2),
 ORDER_DATE DATE
);
```

Practice 4: Create Views

sql

```
-- Standard view
CREATE OR REPLACE VIEW V_TOP_CUSTOMERS AS
SELECT CUSTOMER_ID,
    COUNT(*) AS ORDERS,
    SUM(AMOUNT) AS TOTAL_SPENT
FROM PROD_ORDERS
GROUP BY CUSTOMER_ID
HAVING ORDERS > 5;

-- Secure view (for data sharing)
CREATE OR REPLACE SECURE VIEW V_CUSTOMER_SUMMARY AS
SELECT CUSTOMER_ID,
    TOTAL_SPENT
FROM V_TOP_CUSTOMERS;

-- Materialized view (for performance)
CREATE OR REPLACE MATERIALIZED VIEW MV_DAILY_SALES AS
SELECT ORDER_DATE,
    COUNT(*) AS ORDERS,
    SUM(AMOUNT) AS REVENUE
FROM PROD_ORDERS
GROUP BY ORDER_DATE;

-- Query materialized view
SELECT * FROM MV_DAILY_SALES ORDER BY ORDER_DATE DESC LIMIT 10;
```

Practice 5: Data Lifecycle

sql

-- Insert data

INSERT INTO PROD_ORDERS **VALUES**

(1, 101, 500.50, '2025-01-15'),

(2, 102, 750.00, '2025-01-16'),

(3, 103, 1200.25, '2025-01-17');

-- Query current data

SELECT * **FROM** PROD_ORDERS;

-- Query historical data (time travel)

SELECT * **FROM** PROD_ORDERS

AT (**TIMESTAMP** => '2025-01-16 12:00:00'::TIMESTAMP_NTZ);

-- Delete old records

DELETE FROM PROD_ORDERS

WHERE ORDER_DATE < '2025-01-01';

-- Restore deleted rows (within retention)

UNDROP **TABLE** PROD_ORDERS;

---

7. Quick Reference: File Formats & Stages

File Format Commands

sql

CREATE FILE FORMAT format_name **TYPE** = 'CSV' ...;

DROP FILE FORMAT format_name;

SHOW FILE FORMATS;

DESCRIBE FILE FORMAT format_name;

Stage Commands

sql

CREATE STAGE stage_name FILE_FORMAT = format_name;

DROP STAGE stage_name;

LIST @stage_name;

REMOVE @stage_name/file_pattern;

PUT **file**:*///local_path @stage_name;*

GET @stage_name/filename /local_path;

GRANT READ ON STAGE stage_name **TO** ROLE role_name;

Table Commands

sql

CREATE TABLE table_name (...);

CREATE EXTERNAL **TABLE** ext_table (...) **WITH** LOCATION = @stage;

CREATE ICEBERG **TABLE** iceberg_table (...);

CREATE TRANSIENT **TABLE** temp_table (...);

DROP TABLE table_name;

UNDROP **TABLE** table_name;

ALTER TABLE table_name **SET** DATA_RETENTION_TIME_IN_DAYS = 30;

SHOW TABLES;

DESCRIBE TABLE table_name;

View Commands

sql

CREATE VIEW view_name **AS SELECT** ...;

CREATE SECURE **VIEW** secure_view **AS SELECT** ...;

CREATE MATERIALIZED **VIEW** mv_name **AS SELECT** ...;

DROP VIEW view_name;

ALTER MATERIALIZED **VIEW** mv_name REFRESH;

SHOW VIEWS;

DESCRIBE VIEW view_name;

---

8. Key Concepts Summary

Data Lifecycle Best Practices

1. Staging: Use internal stages for sensitive data, external stages for data lakes

2. Loading: Choose file format based on data structure (CSV for simple, Parquet for analytics, Avro for streaming)
3. Storage: Use permanent tables for production, transient for staging, external for lakes
4. Querying: Create views for abstraction, materialized views for expensive aggregations
5. Retention: Set appropriate DATA_RETENTION_TIME_IN_DAYS based on compliance needs
6. Cleanup: Archive old data, delete temporary records, remove unused stages

Format Selection Guide

Data Type	Recommended Format	Why
Tabular/CSV sources	CSV	Simple, native support
APIs/Semi-structured	JSON	Flexible, nesting support
Streaming pipelines	AVRO	Schema evolution, serialization
Analytics/Big data	Parquet	Compression, columnar, fast
Large-scale analytics	ORC	High compression, statistics

Table Type Selection

- Permanent: Production data, long history (compliance)
- Transient: Staging, intermediate results (cost optimization)
- Temporary: Session-specific, temporary calculations
- External: Data lake queries without ingestion
- Iceberg: Multi-engine ecosystems, open format sharing

Resources

- Snowflake File Formats: **https://docs.snowflake.com/en/sql-reference/sql-ddl-file-format**
- Stages: **https://docs.snowflake.com/en/user-guide-data-load-local-file-system-create-stage**
- Table Types: **https://docs.snowflake.com/en/user-guide/tables-iceberg-create**
- Views: **https://docs.snowflake.com/en/user-guide-views**
- Data Lifecycle: **https://docs.snowflake.com/en/user-guide-data-retention**

Learning Summary

✅ Covered:

- File formats (CSV, JSON, AVRO, Parquet, ORC) with practical examples
- Data lifecycle: storage, querying, and removal phases
- Internal and external stages for data ingestion
- Table types: permanent, transient, temporary, external, Iceberg
- Views: standard, secure, and materialized for data abstraction
- Time Travel for data recovery
- Hands-on practices with SQL examples

🚀 Ready For:

- Building complete ETL pipelines with stages and tables
- Querying external data lakes
- Creating efficient views and materialized views
- Setting up data retention policies
- Moving to Snowpipe, Streams, and Tasks
- Performance optimization and clustering

Last Updated: December 12, 2025
Learning Status: Phase 2 Complete | Advanced Features Next