# DMVA Delta/Incremental Loading for Iceberg Tables

**Data Migration Validation Accelerator (DMVA) - Delta Loading Guide**

## Overview

DMVA supports **incremental/delta loading patterns** from Teradata to Snowflake Iceberg tables through a sophisticated **checkpointing and partitioning mechanism**. This enables efficient synchronization of only changed data rather than full table reloads.

## Key Concepts

### 1. **Checksum-Based Checkpointing**

DMVA uses the dmva\_checksums table as the **checkpoint registry** to track what data has been loaded:

-- Core checkpoint structure  
CREATE TABLE dmva\_checksums (  
 checksum\_id NUMBER NOT NULL,  
 object\_id NUMBER NOT NULL, -- Links to source table  
 extract\_group\_id VARCHAR NOT NULL, -- Checkpoint identifier (partition)  
 checksum\_method VARIANT, -- How data is partitioned  
 source\_filter VARCHAR, -- WHERE clause for source  
 target\_filter VARCHAR, -- WHERE clause for target  
 partition\_checksum VARCHAR, -- Data signature  
 partition\_row\_count NUMBER, -- Rows in this partition  
 created\_ts TIMESTAMP\_NTZ(9), -- When checkpoint created  
 updated\_ts TIMESTAMP\_NTZ(9) -- When checkpoint updated  
);

**Key Fields:** - **extract\_group\_id**: Acts as the checkpoint - identifies which data partition has been loaded - **partition\_checksum**: Hash of the data in this partition - detects changes - **source\_filter / target\_filter**: SQL WHERE clauses defining the partition boundaries

### 2. **Checksum Methods** (Partitioning Strategies)

DMVA offers **4 checksum methods** to partition data and enable incremental loads:

| Method | Use Case | Checkpoint Logic |
| --- | --- | --- |
| **WHOLE TABLE** | Small tables, one-time full loads | No partitioning, entire table = 1 checkpoint |
| **by\_integer** | Integer column (ID, sequence) | Partition by ranges: ID >= i \* modulus AND ID < (i+1) \* modulus |
| **by\_date** | Date/timestamp column | Partition by time periods: DATE >= '2024-01-01' AND DATE < '2024-02-01' |
| **by\_substr** | String column (account ID prefix) | Partition by string prefix: SUBSTR(COLUMN, 1, 6) = 'ABC123' |

## Delta Loading Workflow - Executive Overview

**For executive and high-level audiences**

DMVA’s delta loading capability enables **smart, efficient data synchronization** from Teradata to Snowflake Iceberg tables by loading only what has changed, rather than reprocessing entire tables repeatedly. Think of it as an intelligent “sync” mechanism, similar to how cloud storage services sync only modified files rather than re-uploading everything.

### How It Works (High-Level)

**1. Initial Setup - Taking a Snapshot** - DMVA divides your data into logical chunks (like chapters in a book) - Each chunk gets a unique “fingerprint” (checksum) that identifies its contents - These fingerprints are stored as checkpoints in a tracking registry

**2. Ongoing Synchronization - Smart Updates** - On subsequent runs, DMVA checks only the chunks you’re interested in (based on your lookback period) - For each chunk, it compares the current fingerprint with the stored checkpoint - **If fingerprints match**: Data hasn’t changed → Skip (saves time and cost) - **If fingerprints differ**: Data has changed → Reload that chunk only - **If new chunks exist**: New data → Load these chunks

**3. Efficiency Gains** - **Time Savings**: Process hours or days of data changes instead of months/years of historical data - **Cost Reduction**: Use less compute power and storage bandwidth by moving only changed data - **Flexibility**: Handle late-arriving data by reprocessing recent time periods with a lookback window

### Business Benefits

| Benefit | Impact |
| --- | --- |
| **Reduced Processing Time** | Daily updates complete in minutes instead of hours |
| **Lower Cloud Costs** | Pay only for processing changed data, not entire tables |
| **Near Real-Time Data** | More frequent updates possible due to reduced load times |
| **Automatic Change Detection** | System identifies changes without manual intervention |
| **Audit Trail** | Complete logging of what was loaded, when, and why |

### Real-World Example

**Scenario:** A retail company with 500 million historical transactions, growing by 5 million new transactions daily.

**Without Delta Loading (Full Reload):** - Process all 500M rows daily - 2-3 hours per run - High compute costs

**With DMVA Delta Loading:** - Process only changed partitions (typically 1-7 days of data) - 5-15 minutes per run - 90% cost reduction - Can run multiple times per day if needed

### Key Capabilities

✅ **Incremental Loading**: Only process data from a specific point forward (e.g., “last 7 days”)  
✅ **Change Detection**: Automatically identify which data partitions have changed  
✅ **Late-Arriving Data**: Detect and reload historical partitions if late data arrives  
✅ **Parallel Processing**: Multiple data chunks processed simultaneously for speed  
✅ **Full Monitoring**: Complete visibility into what’s being loaded and its status

### When to Use Delta Loading

| Scenario | Recommendation |
| --- | --- |
| Large tables (100M+ rows) with regular updates | **Strongly Recommended** |
| Daily/hourly incremental data feeds | **Ideal Use Case** |
| Historical tables with occasional backfills | **Beneficial** |
| Small tables (<10M rows) | Full reload may be simpler |
| One-time migrations | Full load sufficient |

## Delta Loading Workflow - Technical Details

**For technical and implementation teams**

The following sections provide detailed technical implementation guidance for data engineers and developers.

### Phase 1: Initial Full Load

┌──────────────────────────────────────────────────────────────┐  
│ STEP 1: Discover Table Partitions │  
├──────────────────────────────────────────────────────────────┤  
│ Teradata: │  
│ SELECT FLOOR(transaction\_id / 1000000) AS extract\_group\_id│  
│ , HASH\_AGG(\*) AS partition\_checksum │  
│ , COUNT(\*) AS partition\_row\_count │  
│ FROM source\_table │  
│ GROUP BY extract\_group\_id │  
│ │  
│ Result: Discovers partitions 0, 1, 2, ..., N │  
└──────────────────────────────────────────────────────────────┘  
 ↓  
┌──────────────────────────────────────────────────────────────┐  
│ STEP 2: Store Checkpoints │  
├──────────────────────────────────────────────────────────────┤  
│ INSERT INTO dmva\_checksums: │  
│ extract\_group\_id: '0' │  
│ source\_filter: 'transaction\_id >= 0 AND < 1000000' │  
│ target\_filter: 'transaction\_id >= 0 AND < 1000000' │  
│ partition\_checksum: 'ABC123...' │  
│ partition\_row\_count: 999500 │  
│ │  
│ extract\_group\_id: '1' │  
│ source\_filter: 'transaction\_id >= 1000000 AND < 2000000' │  
│ ... │  
└──────────────────────────────────────────────────────────────┘  
 ↓  
┌──────────────────────────────────────────────────────────────┐  
│ STEP 3: Extract & Load Each Partition │  
├──────────────────────────────────────────────────────────────┤  
│ For EACH partition: │  
│ 1. Unload from Teradata (using source\_filter) │  
│ 2. Upload to Snowflake stage │  
│ 3. Load into Iceberg table (using target\_filter) │  
│ 4. Validate checksums match │  
└──────────────────────────────────────────────────────────────┘

### Phase 2: Incremental/Delta Load

#### **With Lookback Parameter** (Recommended)

The **lookback** parameter enables incremental loading by defining a starting checkpoint:

**Example Configuration:**

{  
 "column\_name": "transaction\_date",  
 "type": "by\_date",  
 "period": "month",  
 "lookback": "20241201"  
}

**Incremental Load Logic:**

-- DMVA generates extract\_group\_ids ONLY from the lookback point forward  
WITH partitions AS (  
 SELECT DATE\_TRUNC('month', transaction\_date) AS extract\_group\_id  
 , HASH\_AGG(\*) AS partition\_checksum  
 , COUNT(\*) AS partition\_row\_count  
 FROM source\_table  
 WHERE transaction\_date >= TO\_DATE('20241201', 'YYYYMMDD') -- ← LOOKBACK FILTER  
 GROUP BY extract\_group\_id  
)  
SELECT TO\_CHAR(extract\_group\_id, 'YYYYMMDD') AS extract\_group\_id  
 , ...  
FROM partitions

**Checkpoint Behavior:** 1. **New Partitions** (not in dmva\_checksums): Loaded as NEW data 2. **Existing Partitions** (in dmva\_checksums): - Compare partition\_checksum (hash) - If **checksums differ** → Data changed → RELOAD partition - If **checksums match** → Skip (already synchronized)

**Delta Load Workflow:**

┌──────────────────────────────────────────────────────────────┐  
│ STEP 1: Query Teradata for Partitions Since Lookback │  
├──────────────────────────────────────────────────────────────┤  
│ Teradata discovers partitions: Dec-2024, Jan-2025, Feb-2025 │  
└──────────────────────────────────────────────────────────────┘  
 ↓  
┌──────────────────────────────────────────────────────────────┐  
│ STEP 2: Compare with Existing Checkpoints │  
├──────────────────────────────────────────────────────────────┤  
│ Dec-2024: EXISTS in dmva\_checksums │  
│ Old checksum: 'ABC123' │  
│ New checksum: 'ABC123' → MATCH → SKIP │  
│ │  
│ Jan-2025: EXISTS in dmva\_checksums │  
│ Old checksum: 'DEF456' │  
│ New checksum: 'DEF789' → CHANGED → RELOAD │  
│ │  
│ Feb-2025: NOT EXISTS in dmva\_checksums → NEW → LOAD │  
└──────────────────────────────────────────────────────────────┘  
 ↓  
┌──────────────────────────────────────────────────────────────┐  
│ STEP 3: Process Only Changed/New Partitions │  
├──────────────────────────────────────────────────────────────┤  
│ Jan-2025: DELETE + INSERT (REPLACE pattern) │  
│ Feb-2025: INSERT (APPEND pattern) │  
└──────────────────────────────────────────────────────────────┘

## Load Patterns Supported

DMVA supports multiple load patterns through its **load process**:

### 1. **REPLACE (Truncate & Load)**

**Pattern:** Delete existing partition data, then insert new data

**Implementation:**

# From system\_snowflake.py  
def get\_delete\_sql(self, object\_identifier, target\_filter):  
 return f'delete from {object\_identifier} where {target\_filter}'  
  
def get\_copy\_sql(self, object\_identifier, ...):  
 return f'copy into {object\_identifier} from (...)'

**Execution Flow:**

-- Step 1: Delete old partition data  
DELETE FROM iceberg\_table   
WHERE transaction\_date >= '2025-01-01'   
 AND transaction\_date < '2025-02-01';  
  
-- Step 2: Load new partition data  
COPY INTO iceberg\_table  
FROM @stage/partition\_0001/  
...;

**When Used:** - Data in source partition has changed (checksum mismatch) - Ensures Iceberg table is **exactly synchronized** with source

### 2. **APPEND (Insert Only)**

**Pattern:** Insert new data without deleting existing data

**When Used:** - New partition (extract\_group\_id not in dmva\_checksums) - Source data is append-only (no updates/deletes)

**Execution Flow:**

-- Only load step (no delete)  
COPY INTO iceberg\_table  
FROM @stage/partition\_new/  
...;

### 3. **MERGE/SCD2** (NOT Directly Supported)

**Current State:** DMVA does **not natively support MERGE** or **SCD Type 2** patterns.

**Workaround Options:**

#### **Option A: Post-Load MERGE (Recommended)**

Use DMVA for data extraction, then apply custom MERGE logic:

-- DMVA loads into staging table  
-- Your custom procedure:  
MERGE INTO target\_iceberg\_table t  
USING staging\_table s  
ON t.primary\_key = s.primary\_key  
WHEN MATCHED AND t.checksum != s.checksum THEN UPDATE SET ...  
WHEN NOT MATCHED THEN INSERT ...;

#### **Option B: External Tool**

Use dbt or custom SQL for SCD2 transformations after DMVA load.

## Logging & Monitoring

DMVA provides **comprehensive logging** at multiple levels:

### 1. **Task-Level Logging**

**Table: dmva\_tasks / dmva\_ingested\_all**

Tracks every task (checksum, unload, upload, load, measure):

SELECT   
 task\_id,  
 task\_type, -- 'checksum\_partition', 'load\_partition', etc.  
 status\_cd, -- 'OK', 'ERROR', 'RUNNING'  
 start\_ts,  
 finish\_ts,  
 DATEDIFF('second', start\_ts, finish\_ts) AS duration\_sec,  
 source\_object\_id,  
 target\_object\_id,  
 extract\_group\_id, -- Which partition was processed  
 result\_payload -- Detailed results (rows loaded, errors, etc.)  
FROM dmva\_tasks  
WHERE task\_type = 'load\_partition'  
ORDER BY start\_ts DESC;

**Example Output:**

task\_id | task\_type | status\_cd | duration | extract\_group\_id | result\_payload  
--------|-----------------|-----------|----------|------------------|----------------  
12345 | load\_partition | OK | 45s | 202501 | {"rows": 1.2M}  
12346 | load\_partition | OK | 38s | 202502 | {"rows": 1.1M}  
12347 | load\_partition | ERROR | 12s | 202503 | {"error": "..."}

### 2. **Status Change Log**

**Table: dmva\_status\_change\_log**

Tracks status transitions for auditability:

SELECT   
 object\_id,  
 extract\_group\_id,  
 old\_status,  
 new\_status,  
 changed\_ts,  
 changed\_by  
FROM dmva\_status\_change\_log  
WHERE extract\_group\_id = '202501'  
ORDER BY changed\_ts;

### 3. **Checkpoint History**

**Table: dmva\_checksums**

Every checkpoint update is logged with timestamps:

SELECT   
 extract\_group\_id,  
 partition\_checksum,  
 partition\_row\_count,  
 created\_ts, -- When first loaded  
 updated\_ts -- When last synchronized  
FROM dmva\_checksums  
WHERE object\_id = (SELECT object\_id FROM dmva\_object\_info WHERE object\_name = 'MY\_TABLE')  
ORDER BY extract\_group\_id;

**Identify Changed Partitions:**

-- Find partitions that were reloaded (updated\_ts != created\_ts)  
SELECT   
 extract\_group\_id,  
 partition\_row\_count,  
 created\_ts,  
 updated\_ts,  
 DATEDIFF('day', created\_ts, updated\_ts) AS days\_since\_first\_load  
FROM dmva\_checksums  
WHERE updated\_ts > created\_ts  
ORDER BY updated\_ts DESC;

### 4. **Monitoring Views**

DMVA provides pre-built views for monitoring:

-- Overall migration status  
SELECT \* FROM dmva\_status\_results\_total;  
  
-- Per-table status  
SELECT \* FROM dmva\_status\_results\_detail  
WHERE source\_schema\_name = 'TERADATA\_DB'  
 AND source\_object\_name = 'MY\_TABLE';  
  
-- Active tasks  
SELECT \* FROM dmva\_tasks\_active;  
  
-- Task statistics  
SELECT \* FROM dmva\_task\_stats\_by\_date;

## Complete Delta Load Example

### Scenario: Daily Incremental Load from Teradata to Snowflake Iceberg

**Source Table:** - teradata\_db.transactions (500M rows, growing by ~5M/day) - Date column: transaction\_date

**Target Table:** - snowflake\_db.iceberg\_schema.transactions (Iceberg table)

### Step 1: Configure Checksum Method

-- Set incremental checksum method with lookback  
UPDATE dmva\_object\_info  
SET checksum\_method = PARSE\_JSON('{  
 "column\_name": "transaction\_date",  
 "type": "by\_date",  
 "period": "day",  
 "lookback": "20250101"  
}')  
WHERE system\_name = 'teradata\_source'  
 AND schema\_name = 'teradata\_db'  
 AND object\_name = 'transactions';

**Explanation:** - **type: by\_date**: Partition by date ranges - **period: day**: Each partition = 1 day of data - **lookback: 20250101**: Only process data from Jan 1, 2025 forward

### Step 2: Initial Load (Day 1)

-- Execute checksum/load tasks  
CALL dmva\_get\_checksum\_tasks('teradata\_source', PARSE\_JSON('{"teradata\_db": ["transactions"]}'));

**What Happens:** 1. **Teradata** discovers partitions: 20250101, 20250102, …, 20250108 (8 days) 2. Each partition becomes a checkpoint in dmva\_checksums 3. Data for each day is unloaded, uploaded, and loaded into Iceberg table 4. Checksums stored for validation

**Result:**

SELECT extract\_group\_id, partition\_row\_count, created\_ts  
FROM dmva\_checksums  
WHERE object\_id = (SELECT object\_id FROM dmva\_object\_info WHERE object\_name = 'transactions')  
ORDER BY extract\_group\_id;  
  
-- Output:  
-- 20250101 | 5,100,000 | 2025-01-08 10:00:00  
-- 20250102 | 5,050,000 | 2025-01-08 10:12:00  
-- 20250103 | 5,200,000 | 2025-01-08 10:24:00  
-- ...  
-- 20250108 | 5,150,000 | 2025-01-08 11:30:00

### Step 3: Delta Load (Day 2 - Next Day)

-- Update lookback to yesterday (or keep same lookback for a sliding window)  
UPDATE dmva\_object\_info  
SET checksum\_method = PARSE\_JSON('{  
 "column\_name": "transaction\_date",  
 "type": "by\_date",  
 "period": "day",  
 "lookback": "20250108"  
}')  
WHERE object\_name = 'transactions';  
  
-- Execute delta load  
CALL dmva\_get\_checksum\_tasks('teradata\_source', PARSE\_JSON('{"teradata\_db": ["transactions"]}'));

**What Happens:**

1. **Teradata** discovers partitions from lookback forward: 20250108, 20250109 (today)
2. **Compare Checksums:**

* Partition 20250108:  
   - Exists in dmva\_checksums: YES  
   - Old checksum: 'ABC123'  
   - New checksum: 'ABC456' (late-arriving data added!)  
   - Action: RELOAD (DELETE + INSERT)  
    
  Partition 20250109:  
   - Exists in dmva\_checksums: NO  
   - Action: LOAD (INSERT only)

1. **Execution:**

* -- Partition 20250108: DELETE + INSERT (REPLACE pattern)  
  DELETE FROM iceberg\_schema.transactions  
  WHERE transaction\_date >= '2025-01-08' AND transaction\_date < '2025-01-09';  
    
  COPY INTO iceberg\_schema.transactions FROM @stage/20250108/...;  
    
  -- Partition 20250109: INSERT (APPEND pattern)  
  COPY INTO iceberg\_schema.transactions FROM @stage/20250109/...;

1. **Update Checkpoints:**

* -- Partition 20250108: Update updated\_ts  
  UPDATE dmva\_checksums  
  SET partition\_checksum = 'ABC456',  
   partition\_row\_count = 5,155,000, -- Now includes late data  
   updated\_ts = CURRENT\_TIMESTAMP()  
  WHERE extract\_group\_id = '20250108';  
    
  -- Partition 20250109: Insert new checkpoint  
  INSERT INTO dmva\_checksums (extract\_group\_id, partition\_checksum, ...)  
  VALUES ('20250109', 'DEF789', ...);

### Step 4: Monitor Delta Load

-- Check which partitions were processed  
SELECT   
 t.extract\_group\_id,  
 t.task\_type,  
 t.status\_cd,  
 t.start\_ts,  
 t.finish\_ts,  
 t.result\_payload:rows\_affected::NUMBER AS rows\_loaded  
FROM dmva\_tasks t  
WHERE t.source\_object\_id = (SELECT object\_id FROM dmva\_object\_info WHERE object\_name = 'transactions')  
 AND t.start\_ts >= DATEADD('hour', -1, CURRENT\_TIMESTAMP())  
ORDER BY t.start\_ts DESC;  
  
-- Output:  
-- 20250109 | load\_partition | OK | ... | 5,120,000 rows  
-- 20250108 | load\_partition | OK | ... | 5,155,000 rows (reloaded)

## Best Practices

### 1. **Choose the Right Checksum Method**

| Data Pattern | Recommended Method |
| --- | --- |
| Append-only with timestamps | by\_date (day/month) |
| Sequential IDs (auto-increment) | by\_integer (modulus = 1M-10M) |
| Partition key in source | Match source partitioning |
| Small tables (<10M rows) | WHOLE TABLE |

### 2. **Set Appropriate Lookback Windows**

**Sliding Window (Recommended):**

{  
 "lookback": "LAST\_7\_DAYS"  
}

* Reprocesses last 7 days to catch late-arriving data
* Balances data freshness vs. processing cost

**Fixed Checkpoint:**

{  
 "lookback": "20250101"  
}

* Loads all data from a specific date forward
* Use for initial loads or backfills

### 3. **Partition Size Guidelines**

Target: **1M-10M rows per partition**

**Too Large:** - Slow processing - Memory issues - Difficult to parallelize

**Too Small:** - Too many tasks - Overhead dominates - Poor performance

**Adjust modulus/period:**

// If partitions too large (>100M rows):  
{ "modulus": 100000 } // Smaller modulus = more partitions  
  
// If partitions too small (<100K rows):  
{ "modulus": 10000000 } // Larger modulus = fewer partitions

### 4. **Monitor & Alert**

-- Create alert for failed loads  
CREATE ALERT delta\_load\_failures  
WAREHOUSE = dmva\_wh  
SCHEDULE = '1 HOUR'  
IF (EXISTS (  
 SELECT 1   
 FROM dmva\_tasks  
 WHERE status\_cd = 'ERROR'  
 AND task\_type = 'load\_partition'  
 AND start\_ts >= DATEADD('hour', -1, CURRENT\_TIMESTAMP())  
))  
THEN  
 CALL send\_notification\_sp('Delta load failures detected!');

## Limitations & Workarounds

### ❌ **NOT Supported**

1. **Native MERGE/UPSERT**
   * Workaround: Use REPLACE pattern + post-load MERGE
2. **SCD Type 2 (Slowly Changing Dimensions)**
   * Workaround: Load to staging, apply SCD2 logic separately
3. **Change Data Capture (CDC)**
   * Workaround: Use checksum method to detect changes at partition level
4. **Cross-Partition Updates**
   * Limitation: DMVA works at partition granularity
   * Workaround: Set partition size to match update patterns

### ✅ **Supported**

1. ✅ **REPLACE** (DELETE + INSERT per partition)
2. ✅ **APPEND** (INSERT only for new partitions)
3. ✅ **Late-Arriving Data** (via lookback and checksum comparison)
4. ✅ **Incremental Loads** (via checkpointing)
5. ✅ **Parallel Processing** (multiple partitions loaded simultaneously)
6. ✅ **Data Validation** (checksum verification)
7. ✅ **Comprehensive Logging** (task tracking, status changes, audit trail)

## Summary

DMVA provides **robust delta/incremental loading** capabilities through:

1. **Checkpoint Mechanism**: dmva\_checksums table tracks loaded partitions
2. **Partitioning Strategies**: 4 checksum methods for different data patterns
3. **Lookback Support**: Enables incremental loads from a specific point forward
4. **Change Detection**: Checksum comparison identifies changed partitions
5. **Load Patterns**: REPLACE (update) and APPEND (insert) patterns
6. **Comprehensive Logging**: Multi-level tracking for monitoring and troubleshooting

**For Teradata → Snowflake Iceberg migrations:** - ✅ Efficient delta loads without full table scans - ✅ Automatic detection of changed data - ✅ Parallel processing for performance - ✅ Full audit trail of all loads - ⚠️ Post-processing required for MERGE/SCD2 patterns

**Next Steps:** - Configure appropriate checksum\_method for your tables - Set lookback parameter for incremental loads - Schedule periodic delta loads (daily/hourly) - Monitor via dmva\_tasks and status views - Apply custom MERGE logic if needed for SCD patterns