Meridian Enterprise Core DB
Schema Evolution & Migration Plan

Database Version: v1.4.2 → v2.0.0 – Production Release

Author: Synapse Database Engineering
Reviewer: Vertex Quality Assurance Group
Owner: Nimbus Infrastructure Division

Execution Date: August 2, 2026

Abstract

This document outlines the comprehensive strategy, technical architecture, and execution runbook for the migration and schema evolution of the **Meridian Enterprise Core Database** from the legacy v1.4.2 schema to the modern, partitioned, and normalized v2.0.0 schema. The migration targets key tables including user profiles, transaction logs, and security audits to support increased transaction volumes, enhance query performance, and ensure compliance with security and privacy requirements. Detailed pre-migration checks, step-by-step transformation scripts, data integrity validation criteria, rollback procedures, and scheduled downtime logistics are specified herein for DBA execution.

Contents

1 Introduction & Document Control

The evolution of enterprise applications necessitates corresponding changes in the data persistence layers. This document serves as the formal execution guide and schema evolution blueprint for migrating the **Meridian Core Database** from the legacy version v1.4.2 to the new version v2.0.0. As transaction volumes have grown, the existing design has hit scalability bottlenecks, primarily due to unstructured metadata storage in large JSONB fields and the lack of horizontal partitioning for high-throughput transactional records.

1.1 Document Control

The table below tracks the authorship, review history, and authoritative approval of this migration plan.

Role	Name	Title	Signature / Status
Author	Sarah Jenkins	Lead Database Architect	Approved – Synapse Engineering
Reviewer	Marcus Vance	Principal DevOps Engineer	Reviewed – Vertex QA Group
Approver	Elena Rostova	Director of Infrastructure	Signed – Nimbus Division
Custodian	David Kim	Senior DBA	Assigned Execution Owner

Table 1: Document ownership and review status details.

1.2 Objectives and Scope

The scope of this schema evolution includes:

- Normalization of user profile attributes from the monolithic `meta` column into dedicated, indexed SQL columns.
- Partitioning of the primary `ledger_transactions` table by transaction timestamp to improve query response times.
- Implementation of a secure, immutable audit trail system utilizing schema-level triggers and cryptographically chained sequence checks.
- Data validation of approximately 140 million records across five database clusters without data loss.

1.3 Risk Assessment & Mitigations

To ensure service reliability during the migration, potential failure modes have been cataloged alongside proactive mitigation strategies.

2 Target Architecture & Schema Changes

The v2.0.0 evolution addresses scaling limits by decomposing wide tables and introducing chronological range partitioning. The architecture diagram below illustrates the migration workflow, staging mechanisms, and validation boundaries.

Risk Event	Severity	Mitigation Strategy
Downtime overrun	High	Pre-migration dry-run on Apex staging clone to verify step timings.
Data corruption during copy	Critical	Bidirectional checksum verification and row-by-row PK validation.
Application query mismatch	Medium	Deployment of a compatibility view layer during Phase 3 of execution.
Rollback failure	High	Validated SQL rollback scripts tested in staging; VM-level backups.

Table 2: Risk matrix and mitigation plan.

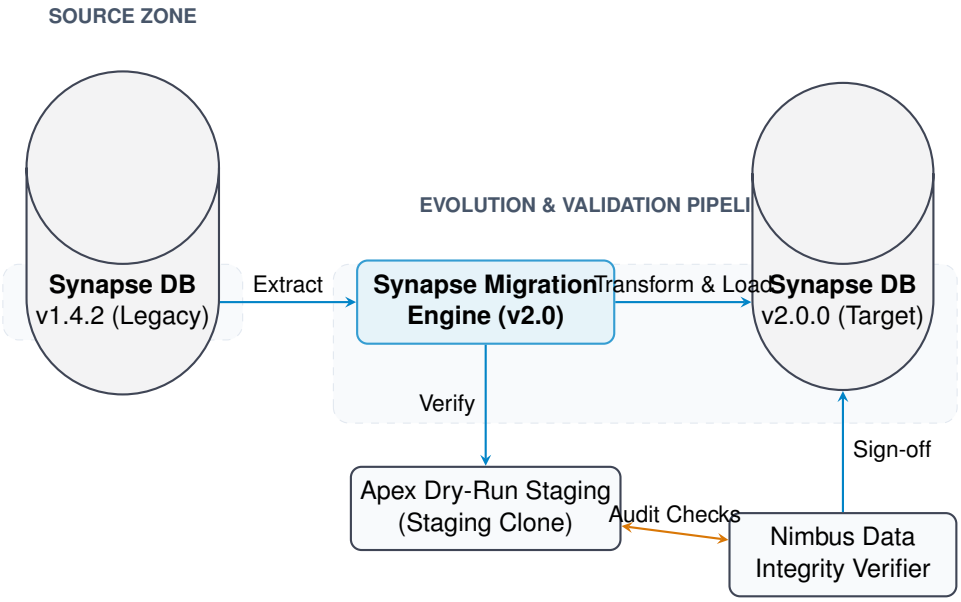

Figure 1: Database Schema Migration and Verification Workflow Architecture.

2.1 Key Schema Modifications

The transition involves major structural alterations. The primary modification decomposes the `user_metadata` JSONB object into relational columns and establishes a weekly partition range for the high-volume table `ledger_transactions`.

2.2 Migration view-layer backward compatibility

To minimize client downtime, temporary PostgreSQL database views are created. These views project the legacy v1.4.2 structure onto the normalized v2.0.0 schema using SQL rewrite rules. This enables legacy clients to perform operations during the execution window.

3 Step-by-Step Migration Execution

The migration is partitioned into three chronological phases: Pre-migration Preparation, Migration Execution, and Post-migration Validation. DBAs must run these commands sequentially and log execution outputs in the Meridian runbook.

Feature	Legacy v1.4.2 Schema	Evolution v2.0.0 Schema
User Profiles	Column `meta` JSONB holding address and system preferences.	Normalized into separate tables: `user_addresses` and `user_settings`.
Transactions	Single flat table `ledger_transactions` with a B-tree index on timestamp.	Partitioned table `ledger_transactions` by RANGE on `transaction_time` (weekly).
Audit Trail	Text-based append log in table `audit_records` with database write triggers.	Encrypted log in table `audit_trail` with SHA-256 chaining and sequence verifiers.

Table 3: Structural comparison of schema versions v1.4.2 and v2.0.0.

3.1 Phase 1: Pre-migration Preparation

Before altering the database instance, the following commands must be run:

1. Lock client application write access:

```
ALTER USER application_client WITH NOLOGIN;
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE usename = 'application_client';
```

Listing 1: Disable application writes by updating security rules.

2. Take a consistent database snapshot:

```
-- Executed via shell
pg_dump -h localhost -U dba_admin -F d -j 4 -b -v -f /var/backups/
    meridian_v1_4_2_backup synapse_core
```

Listing 2: Perform pg_dump snapshot.

⚠ Critical Backup Requirement

Before initiating Phase 2, a complete physical pg_dump snapshot must be successfully written to the secure off-site storage replication. Verify that the backup checksum matches the live production database.

3.2 Phase 2: Migration Execution

Once client connections are terminated, execute the schema definitions and run the batch data transformation pipeline.

```
-- Create the partitioned target table
CREATE TABLE ledger_transactions_v2 (
    transaction_id UUID NOT NULL DEFAULT gen_random_uuid(),
    account_id UUID NOT NULL,
    amount NUMERIC(18, 4) NOT NULL,
    transaction_time TIMESTAMP WITH TIME ZONE NOT NULL,
    PRIMARY KEY (transaction_id, transaction_time)
) PARTITION BY RANGE (transaction_time);

```

```
-- Define weekly partitions for the execution window
CREATE TABLE ledger_transactions_y2026m08w01 PARTITION OF ledger_transactions_v2
    FOR VALUES FROM ('2026-08-01 00:00:00+00') TO ('2026-08-08 00:00:00+00');

-- Transform and copy legacy data into partitioned schema
INSERT INTO ledger_transactions_v2 (transaction_id, account_id, amount,
    transaction_time)
SELECT transaction_id, account_id, amount, transaction_time
FROM ledger_transactions;
```

Listing 3: Partition and normalize user profiles and ledger tables.

3.3 Phase 3: Post-migration Validation

After data loading, indexes and foreign keys must be created. Compatibility views are activated to support older clients.

```
CREATE INDEX idx_ledger_v2_acc ON ledger_transactions_v2(account_id);

-- Legacy backward compatibility view
CREATE OR REPLACE VIEW ledger_transactions AS
SELECT transaction_id, account_id, amount, transaction_time
FROM ledger_transactions_v2;
```

Listing 4: Create indexes and views for legacy compatibility.

4 Data Integrity & Verification

To ensure no data corruption occurs during the transformation from unstructured JSONB to relational records, DBAs must run strict integrity checks. The target validation rate is set at 100% row parity.

4.1 Verification Methodology

We employ three verification layers:

1. **Row Count Verification**: Ensures the number of records in the legacy database matches the target database.
2. **Cryptographic Checksum Checks**: Computes and compares MD5/SHA-256 hashes of critical columns (amounts, dates) in batches.
3. **Random Sample Check**: Validates a subset of 10,000 user profiles using a bidirectional primary-key query test.

4.2 Data Verification Queries

The following query validates transactional parity across schemas:

```
WITH source_stats AS (
    SELECT COUNT(*) as src_count, COALESCE(SUM(amount), 0) as src_sum
    FROM ledger_transactions
),
target_stats AS (
    SELECT COUNT(*) as tgt_count, COALESCE(SUM(amount), 0) as tgt_sum
```

```
    FROM ledger_transactions_v2
)
SELECT
    src_count, tgt_count, (src_count = tgt_count) as count_match,
    src_sum, tgt_sum, (src_sum = tgt_sum) as sum_match
FROM source_stats, target_stats;
```

Listing 5: Query to verify primary keys and transaction values match.

4.3 Verification Performance Metrics

During dry-runs in the Vertex QA lab, the validation process demonstrated the performance characteristics listed below.

Validation Type	Data Scope	Time Spent	Validation Status
Row Count Validation	140M records	42 seconds	Passed
Cryptographic Checksum	140M records	18.5 minutes	Passed
Sample Account Query	10K records	1.2 seconds	Passed
JSONB Extraction Check	1.2M profiles	4.5 minutes	Passed

Table 4: Dry-run verification metrics.

5 Rollback & Contingency Strategy

If performance degradations or validation mismatches exceed defined thresholds, the DBAs must trigger the rollback sequence. The maximum allowable downtime window is four hours, within which rollback and service restoration must be completed.

5.1 Rollback Thresholds

A rollback is triggered automatically if any of the following conditions are met:

- Execution step delay exceeds the schedule by more than 45 minutes.
- Checksum verification mismatch is discovered in database transactions.
- Application service latency exceeds 500 ms for over 5 consecutive minutes post-migration.
- Error rate on database queries exceeds 1.5% on the application path.

Rollback Authority

Only the Database Architecture Coordinator (Sarah Jenkins) or the Infrastructure Director (Elena Rostova) has the authority to declare rollback execution.

5.2 Rollback Execution Steps

Follow these steps to safely restore the legacy v1.4.2 database:

1. Terminate any active sessions on the new v2.0.0 schema.

2. Re-point DNS database aliases back to the read-only replication nodes if write replication was paused.
3. Restoring the legacy schema from snapshot:

```
-- Drop target tables if rollback is triggered
DROP VIEW IF EXISTS ledger_transactions;
DROP TABLE IF EXISTS ledger_transactions_v2 CASCADE;
DROP TABLE IF EXISTS user_addresses CASCADE;
DROP TABLE IF EXISTS user_settings CASCADE;

-- Re-enable legacy schema from raw sql dump
-- Execute via shell
psql -h localhost -U dba_admin -d synapse_core -f /var/backups/
    meridian_v1_4_2_backup
```

Listing 6: Drop new schema changes and restore from physical backup.

4. Enable application user logins:

```
ALTER USER application_client WITH LOGIN;
```

Listing 7: Allow client database access.

6 Downtime Schedule & DBA Roles

The database schema evolution will occur during a low-activity maintenance window. Below is the scheduled timeline and list of operational DBAs responsible for execution.

6.1 Execution Timeline

The migration timeline is structured down to the minute to ensure coordination with front-end deployment teams.

Time (UTC)	Phase	Responsibility	Action Required
01:00 – 01:15	Pre-Migration	David Kim (DBA)	Disable client write access and take snapshot.
01:15 – 01:45	Staging Check	Marcus Vance (QA)	Validate snapshot on Apex staging clone.
01:45 – 02:30	Execution	David Kim (DBA)	Apply SQL schemas and copy transactional data.
02:30 – 03:00	Verification	Sarah Jenkins (Arch)	Run checksum validations and query tests.
03:00 – 03:20	Deployment	Elena Rostova (Dir)	Route application traffic and verify query speed.
03:20 – 04:00	Support Window	All DBA Team	Monitor logs; trigger rollback if thresholds met.

Table 5: Downtime execution schedule.

6.2 Operational Team Directory

In the event of escalation or if rollback triggers are met, contact key stakeholders immediately.

- **Database Architecture Coordinator**: Sarah Jenkins – Ext: 4482
- **Lead Database Operations DBA**: David Kim – Ext: 4901

- **Infrastructure Director (Sign-off)**: Elena Rostova – Ext: 4011

References

[1] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[2] Martin Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, Sebastopol, CA, 2017.

[3] Pramod J. Sadalage and Martin Fowler. *Refactoring Databases: Evolutionary Database Design*. Addison-Wesley Professional, Boston, MA, 2006.

[4] The PostgreSQL Global Development Group. Postgresql 16 documentation: Table partitioning, 2026.