UDM Framework Philosophy

Core Principles and Design Approach

Introduction

The Universal Data Model (UDM) Framework is built on a set of fundamental design principles that address common data warehousing challenges while providing flexibility for diverse business requirements. This document explores the philosophical foundations that make the framework both powerful and maintainable.

---

1. Metadata-Driven ETL: Configuration Over Code

The Principle

Traditional ETL requires custom code for each data source. The UDM Framework replaces this with a metadata-driven approach where configurations in reference tables orchestrate the entire data pipeline.

```
graph TB
    subgraph Traditional["TRADITIONAL ETL APPROACH"]
        Source1[New Source System]
        Code1[Write Custom<br/>Extraction Code]
        Code2[Write Custom<br/>Transformation Logic]
        Code3[Write Custom<br/>Load Procedures]
        Test1[Test & Deploy]

        Source1 --> Code1
        Code1 --> Code2
        Code2 --> Code3
        Code3 --> Test1

        Time1[⏱️ 4-12 Weeks]
    end
```

```
    subgraph UDM["UDM FRAMEWORK APPROACH"]
        Source2[New Source System]
        Config1[Configure SourceSystem<br/>Table Entry]
        Config2[Map Columns in<br/>SourceColumnMapping]
        SP[Run Standard<br/>Stored Procedures]
        Pipeline[Configure/Reuse<br/>Dataflow Template]

        Source2 --> Config1
        Config1 --> Config2
        Config2 --> SP
        SP --> Pipeline

        Time2[⏱ 3-5 Days]
    end

    style Traditional fill:#ffcdd2
    style UDM fill:#c8e6c9
    style Time1 fill:#f44336,color:#fff
    style Time2 fill:#4caf50,color:#fff
```

The SourceSystem Table: Central Orchestrator

At the heart of the UDM Framework is the `Reference.SourceSystem` table, which serves as the master configuration for all data integrations:

```
CREATE TABLE [Reference].[SourceSystem](
    [SourceSystemId] [bigint] IDENTITY(1,1) NOT NULL,
    [SourceDescription] [nvarchar](100) NULL,
    [SourceDatabase] [nvarchar](100) NULL,
    [SourceSchema] [nvarchar](50) NULL,
    [SourceTable] [nvarchar](100) NULL,
    [SourceIdentifierColumn] [nvarchar](100) NULL,
    [TargetSchema] [nvarchar](50) NULL,
    [TargetTable] [nvarchar](100) NULL,
    [TargetIdentifierColumn] [nvarchar](100) NULL,
    [ValidFromDate] [date] NULL,
    [ValidToDate] [date] NULL,
    [ParentSourceSystemId] [bigint] NULL,
    [SourceKeyColumn] [nvarchar](100) NULL,
```

```
    [SourceValueColumn] [nvarchar](100) NULL,
    CONSTRAINT [Pk_SourceSystem] PRIMARY KEY CLUSTERED
    ([SourceSystemId] ASC)
)
```

Key Concepts

Source-to-Target Mapping

- `SourceDatabase`, `SourceSchema`, `SourceTable`: Where data comes from
- `TargetSchema`, `TargetTable`: Where data goes
- `SourceIdentifierColumn` / `TargetIdentifierColumn`: Primary key mapping

Temporal Validity

- `ValidFromDate` / `ValidToDate`: Control when source mappings are active
- Allows for source system versioning and deprecation
- Supports parallel runs during migration periods

Hierarchical Relationships

- `ParentSourceSystemId`: Links related source systems
- Enables parent-child data relationships
- Supports EAV pattern implementations

EAV Pattern Support

- `SourceKeyColumn`: Property name column in source
- `SourceValueColumn`: Property value column in source
- Enables flexible attribute storage without schema changes

Benefits

1. **Rapid Integration**: Add a new data source by inserting a row, not writing code
2. **Consistency**: All sources follow the same integration pattern
3. **Visibility**: Easy to see all data sources and their mappings
4. **Maintainability**: Changes to source systems require metadata updates, not code changes
5. **Testability**: Framework logic is tested once; configurations are validated separately

Column-Level Metadata

The `Reference.SourceColumnMapping` table extends source-level metadata to individual columns:

```
CREATE TABLE [Reference].[SourceColumnMapping](
    [SourceColumnMappingId] [bigint] IDENTITY(1,1) NOT NULL,
    [SourceSystemId] [bigint] NOT NULL,
    [SourceColumnName] [nvarchar](100) NULL,
    [TargetAlias] [nvarchar](100) NOT NULL,
    CONSTRAINT [PK_SourceColumnMapping] PRIMARY KEY CLUSTERED
    ([SourceColumnMappingId] ASC)
)
```

Purpose:

- Maps source column names to standardized target column names
- Handles column naming differences across sources
- Enables automatic generation of ETL transformations
- Supports the Flow table pattern for staging

---

2. Generic Data Modeling: One Model, Many Industries

The Principle

Rather than building industry-specific data models, the UDM Framework provides universal domain schemas that adapt to any business context through flexible design patterns.

Universal Domains

The framework organizes data into logical domains that exist across industries:

Sales Domain

- Orders, leads, invoices, transactions
- Revenue and sales performance tracking
- Applicable to: Retail, B2B, Subscriptions, Services

Finance Domain

- Invoicing, payments, revenue recognition
- Financial reporting and accounting
- Applicable to: Any business with financial transactions

Product Domain

- Products, services, offers, SKUs
- Product hierarchies and catalogs
- Applicable to: E-commerce, Manufacturing, SaaS, Services

Contact Domain

- Customers, users, households, organizations
- Relationships and account hierarchies
- Applicable to: B2C, B2B, any customer-facing business

Marketing Domain

- Campaigns, channels, publishers, ad spending
- Attribution and marketing performance
- Applicable to: Any business with marketing activities

Reference Domain

- Configuration tables, lookup values, metadata
- Framework orchestration tables
- Universal across all implementations

Fallout Domain

- Data quality tracking and error management
- Universal across all implementations

Industry Adaptability

The same core domains adapt to different industries:

Example 1: Price Comparison (Pricewise)

- Products = Energy plans, insurance policies, mortgage products
- Sales = User orders for comparison services

- Marketing = Affiliate channels, SEO, paid advertising

Example 2: Mobile Subscriptions (Creative Clicks)

- Products = Mobile subscription plans, carriers
- Sales = Subscription activations, renewals
- Marketing = Publisher networks, campaign tracking

Example 3: E-Commerce (Generic)

- Products = Physical goods, inventory
- Sales = Online orders, transactions
- Marketing = Email campaigns, social ads

Design for Extension

The generic model supports industry-specific extensions:

- **Schema extension**: Add industry-specific schemas alongside universal ones
- **Property tables**: Handle variable attributes through EAV
- **Custom views**: Build industry-specific reporting views on universal base tables

---

3. Separation of Concerns: Multi-Layer Architecture

The Principle

Different data lifecycle stages have different requirements. The UDM Framework separates these into distinct layers, each optimized for its purpose.

Layer 1: Stage / Raw Layer

Purpose: Land raw data exactly as it comes from source systems

Characteristics:

- Minimal transformation (data type casting only)
- Preserves source data structure
- History tracking for change data capture
- Fast, simple loading

Databases: `DS_Stage`, `DS_Stage_*` (per source system)

Key Tables:

- Raw source tables (various schemas)
- `History.*` tables for CDC
- Reference tables (currencies, countries, etc.)

Philosophy: "*Store now, transform later*"

- Don't lose data due to transformation errors
- Maintain source system fidelity
- Enable reprocessing if business rules change

Layer 2: DWH Layer (Integrated)

Purpose: Integrated, cleansed, business-ready data

Characteristics:

- Conformed dimensions across sources
- Cleansed and validated data
- Timeslices for historical accuracy
- Source system traceability
- Fallout tracking for quality issues

Database: `DS_DWH` or `Mara`

Key Schemas:

- Sales, Finance, Product, Contact, Marketing (business domains)
- Reference (metadata and configuration)
- Fallout (data quality)
- Flow (incremental load staging)

Philosophy: "*Single source of truth*"

- One integrated view of the business
- Consistent definitions across sources
- Quality-assured and trustworthy

Layer 3: Reporting / Marts Layer

Purpose: Optimized for consumption by BI tools and end users

Characteristics:

- Denormalized views and aggregations
- Business-friendly naming
- Pre-calculated metrics
- Performance-optimized
- Self-service friendly

Database: `DS_Reporting` or `Mara_Marts`

Key Schemas:

- Dim (dimension views)
- Fact (fact views)
- Model (business-specific models)
- Industry-specific schemas (Mobile, Medgen, etc.)

Philosophy: "*Make it easy for users*"

- Hide complexity from business users
- Optimize query performance
- Present data in business terms

Layer 4: Sync Layer (Optional)

Purpose: Replicate and synchronize data between systems

Characteristics:

- Exact replicas of source systems
- Supports hybrid architectures
- Enables cloud migration
- Reduces load on production systems

Database: `DS_SYNC` or `Mara_Sync`

Philosophy: "*Safe, isolated replication*"

- Don't impact production systems
- Enable parallel processing

- Support gradual migration

Benefits of Separation

1. **Performance**: Each layer optimized for its workload
2. **Maintainability**: Changes in one layer don't cascade
3. **Flexibility**: Can swap technology in individual layers
4. **Security**: Different access controls per layer
5. **Debugging**: Easier to isolate issues to specific layer
6. **Reprocessing**: Can reload DWH from Stage without touching sources

---

4. Data Quality & Governance Philosophy: Fallout System

The Principle

Data quality issues are inevitable. Rather than failing silently or blocking pipelines, the UDM Framework treats quality issues as first-class data that must be tracked, managed, and resolved.

The Fallout System

Core Table:

```
CREATE TABLE [Fallout].[Fallout](
    [FalloutId] [int] IDENTITY(1,1) NOT NULL,
    [ErrorCodeId] [int] NOT NULL,
    [SourceSystemId] [bigint] NOT NULL,
    [SourceSystemIdentifier] [nvarchar](100) NOT NULL,
    [InsertDate] [datetime] NOT NULL,
    [SolveDate] [datetime] NULL,
    CONSTRAINT [PK_Fallout2] PRIMARY KEY CLUSTERED ([FalloutId] ASC)
)
```

Philosophy: "Fail Gracefully, Track Everything"

Traditional Approach:

- ETL fails completely when encountering bad data
- Error logs are difficult to analyze
- Business users don't know what's missing
- Data quality issues compound over time

UDM Approach:

- Pipeline continues running, "splits off" problem records
- Each issue logged with context (source, error type, identifier)
- Business users see what data is missing and why
- Resolution workflow tracks fixes

Fallout Workflow

1. **Detection**: ETL identifies data quality issue (missing FK, invalid value, etc.)
2. **Logging**: Record inserted into `Fallout.Fallout` with error code and context
3. **Notification**: Monitoring alerts triggered for new fallouts
4. **Analysis**: Data stewards review fallout records
5. **Resolution**: Fix at source or adjust business rules
6. **Reprocessing**: Corrected records flow through normally
7. **Closure**: `SolveDate` updated to mark resolution

Error Code Categories

The framework uses standardized error codes:

- **Code 11**: Missing foreign key for dimension A
- **Code 12**: Missing foreign key for dimension B
- **Code 13**: Missing foreign key for dimension C
- **Code 1x**: Additional FK validation errors
- **Code 2x**: Data type / format errors
- **Code 3x**: Business rule violations
- **Code 4x**: Duplicate detection

Benefits

1. **Pipeline Resilience**: ETL doesn't fail due to data quality issues
2. **Visibility**: Complete transparency into data quality
3. **Prioritization**: Focus on high-impact quality issues
4. **Trend Analysis**: Track data quality improvements over time

5. **Root Cause Analysis**: Trace issues back to source systems
6. **Accountability**: Clear ownership of data quality

Governance Through Traceability

Every record in the DWH includes `SourceSystemId`:

- **Complete lineage**: Know exactly where data came from
- **Impact analysis**: Understand downstream effects of source changes
- **Compliance**: Provide audit trails for regulatory requirements
- **Debugging**: Quickly isolate issues to specific sources

---

5. Timeslice-Based Historical Tracking

The Principle

Business data changes over time. The UDM Framework treats time as a first-class dimension, enabling accurate historical reporting and trend analysis.

The Timeslice Pattern

Core Concept: Every entity has validity periods defined by `ValidFromDate` and `ValidToDate`

```
-- Example: Product dimension with timeslices
CREATE TABLE [Product].[Product](
    [ProductId] [bigint] IDENTITY(1,1) NOT NULL,
    [ProductNK] [nvarchar](100) NOT NULL,  -- Natural key from source
    [ProductName] [nvarchar](200) NULL,
    [Price] [decimal](18,2) NULL,
    [ValidFromDate] [date] NOT NULL,
    [ValidToDate] [date] NOT NULL,         -- 9999-12-31 for current
    [SourceSystemId] [bigint] NOT NULL,
    CONSTRAINT [PK_Product] PRIMARY KEY CLUSTERED ([ProductId] ASC)
)
```

Change Tracking Approach

When source data changes:

1. **Current record**: Update `ValidToDate` to date of change
2. **New record**: Insert with `ValidFromDate` = change date, `ValidToDate` = 9999-12-31
3. **Natural key preserved**: Same `ProductNK`, different surrogate keys
4. **Full history**: Complete timeline of changes maintained

History Table Pattern

The Stage layer tracks changes in source systems:

Stored Procedure: `BuildHistoryTracking`

```
-- Creates history tracking for any source table
EXEC [dbo].[BuildHistoryTracking] '[dbo].[ProductTable]'

-- Results in History.ProductTable with:
-- - ChangeType: INSERT, UPDATE, DELETE
-- - ChangeTime: When change occurred
-- - All original columns: Full record snapshot
```

Point-in-Time Queries

Timeslices enable "as of" queries:

```
-- What was the price of Product X on 2023-06-15?
SELECT ProductName, Price
FROM Product.Product
WHERE ProductNK = 'ProductX'
  AND '2023-06-15' BETWEEN ValidFromDate AND ValidToDate
```

Benefits

1. **Historical Accuracy**: Report on past exactly as it was
2. **Trend Analysis**: Track changes over time (price changes, customer migrations)
3. **Regulatory Compliance**: Maintain complete audit history
4. **Debugging**: Understand when data changed and why
5. **What-if Analysis**: Compare current state to historical states

Philosophy: "Preserve the Past, Track the Present"

- Never delete historical data
- Every change is an insert, not an update
- Storage is cheap; lost history is expensive
- Enable time-travel queries for business insights

---

6. EAV Pattern: Flexible Properties

The Principle

Different entities have different attributes. Rather than forcing a rigid schema, the UDM Framework supports dynamic attributes through the Entity-Attribute-Value (EAV) pattern.

The Problem

Traditional Approach:

```
-- Must add column for every new attribute
ALTER TABLE Product ADD Color VARCHAR(50)
ALTER TABLE Product ADD Size VARCHAR(20)
ALTER TABLE Product ADD Material VARCHAR(100)
-- Leads to:
-- - Wide tables with many NULLs
-- - Schema changes for new attributes
-- - Migration complexity
```

UDM Approach: Property Tables

```
-- Base entity table
CREATE TABLE [Sales].[Order](
    [OrderId] [bigint] IDENTITY(1,1) NOT NULL,
    [OrderNK] [nvarchar](100) NOT NULL,
    [OrderDate] [datetime] NULL,
    -- Core attributes only
)
```

```
-- Properties table (EAV)
CREATE TABLE [Sales].[OrderProperties](
    [OrderPropertyId] [bigint] IDENTITY(1,1) NOT NULL,
    [OrderId] [bigint] NOT NULL,              -- Entity reference
    [PropertyName] [nvarchar](100) NOT NULL,   -- Attribute
    [PropertyValue] [nvarchar](500) NULL,      -- Value
    [ValidFromDate] [date] NULL,
    [ValidToDate] [date] NULL,
    CONSTRAINT [PK_OrderProperties] PRIMARY KEY CLUSTERED
)
```

SourceSystem Support for EAV

The `SourceKeyColumn` and `SourceValueColumn` in `Reference.SourceSystem` enable EAV loading:

```
-- Example SourceSystem entry for EAV source
SourceTable = 'ProductAttributes'
SourceKeyColumn = 'AttributeName'    -- Column containing property names
SourceValueColumn = 'AttributeValue' -- Column containing property values
ParentSourceSystemId = 42            -- Link to parent Product entity
```

Property Reference Table

`Reference.PropertyReference` controls which properties are active:

```
CREATE TABLE [Reference].[PropertyReference](
    [PropertyReferenceId] [bigint] IDENTITY(1,1) NOT NULL,
    [PropertyTypeId] [int] NOT NULL,
    [SourceSystemId] [bigint] NOT NULL,
    [SourceColumn] [nvarchar](100) NOT NULL,
    [Ignore] [bit] NOT NULL DEFAULT 1,  -- Control property visibility
)
```

Workflow:

1. Run `InitPropertyReference` to discover available properties
2. Set `Ignore = 0` for properties to include in DWH
3. Daily ETL automatically loads non-ignored properties

Benefits

1. **Schema Flexibility**: Add attributes without DDL changes
2. **Source Variety**: Different sources can have different attributes
3. **Sparse Data**: No wasted space on NULL columns
4. **Backward Compatibility**: Adding properties doesn't break existing queries
5. **Self-Service**: Business users can request new attributes without IT dependency

When to Use EAV vs. Columns

Use EAV for:

- Attributes that vary by source
- Frequently changing attribute sets
- Optional attributes with sparse data
- User-defined fields

Use Columns for:

- Core business attributes
- Attributes used in frequent queries
- Foreign keys to dimensions
- High-performance requirements

---

7. Incremental Loading Philosophy

The Principle

Full reloads are wasteful. The UDM Framework uses intelligent incremental loading to process only changed data.

The Flow Pattern

Temporary staging tables in the `Flow` schema hold incremental batches:

```
-- Created dynamically per source system
Flow.Sales_Orders_SourceSystemId42
```

```
-- Contains:
-- - Only new/changed records since last load
-- - All mapped columns from SourceColumnMapping
-- - Ready for dataflow processing
```

Stored Procedure: `ReadSourceTable_NewRecords`

```
-- Populates Flow table with incremental data
EXEC [dbo].[ReadSourceTable_NewRecords] @SourceSystemId = 42

-- Logic:
-- 1. Compare source SourceIdentifierColumn to existing DWH records
-- 2. Select only records not yet in DWH
-- 3. Apply SourceColumnMapping transformations
-- 4. Insert into Flow table
```

Incremental Loading Strategy

1. **Stage Layer**: Sync/replicate full source tables
2. **Flow Tables**: Extract only new records using identifier comparison
3. **DWH Load**: Process Flow tables through dataflows
4. **Timeslice Management**: Update validity periods for changed records

Benefits

1. **Performance**: Process 1% of data instead of 100%
2. **Scalability**: Handles growing data volumes efficiently
3. **Reduced Load**: Less impact on source systems
4. **Faster Pipelines**: Minutes instead of hours
5. **Cost Efficiency**: Lower compute and storage costs

---

8. Design Principles Summary

Principle 1: Configuration Over Code

Reduce custom development through metadata-driven processes

Principle 2: Generic Over Specific

Build universal models that adapt to any industry

Principle 3: Separate Over Combine

Isolate concerns across layers for maintainability

Principle 4: Track Over Block

Handle errors gracefully and maintain quality visibility

Principle 5: Preserve Over Discard

Keep complete history for accuracy and compliance

Principle 6: Flexible Over Rigid

Support dynamic attributes without schema changes

Principle 7: Incremental Over Full

Process only what changed for efficiency

Conclusion

The UDM Framework philosophy represents years of real-world data warehousing experience distilled into reusable patterns. These principles work together to create a system that is:

- **Fast to implement**: Days instead of months
- **Easy to maintain**: Configurations instead of custom code
- **Adaptable to change**: Flexible patterns support evolving requirements
- **Quality-focused**: Built-in tracking and governance
- **Historically accurate**: Complete timeline of all changes
- **Business-friendly**: Self-service analytics enabled

By adhering to these principles, organizations can build data warehouses that scale with their business, adapt to changing requirements, and provide trustworthy insights for years

to come.

Next: 02_Architecture_Overview.md - Detailed technical architecture and infrastructure