UDM Framework Architecture Overview

Technical Architecture and Infrastructure

Introduction

The Universal Data Model (UDM) Framework implements a multi-layer data warehouse architecture designed for cloud-hybrid deployments. This document provides a comprehensive technical overview of the architecture, technology stack, data flows, and deployment patterns.

---

1. High-Level Architecture

Conceptual Overview

```
graph TB
    subgraph Sources["SOURCE SYSTEMS"]
        CRM[CRM Systems]
        ERP[ERP Systems]
        Marketing[Marketing Platforms]
        APIs[External APIs]
    end

    subgraph Extraction["EXTRACTION LAYER"]
        SSIS[SSIS Packages<br/>ETL Scripts]
        Functions[Azure Functions<br/>API Connectors]
    end

    subgraph Stage["STAGE / RAW LAYER"]
        StageCommon[DS_Stage<br/>Common/Reference]
        StageHL[DS_Stage_HL<br/>HealthLabs]
        StageOther[DS_Stage_*<br/>Other Sources]
        direction LR
        StageCommon --> StageHL
```

```
    StageHL --> StageOther
end

subgraph ETL["ETL ORCHESTRATION"]
    ADF[Azure Data Factory<br/>Pipelines & Dataflows<br/>Metadata-dri
end

subgraph DWH["DWH / INTEGRATION LAYER<br/>(DS_DWH / Mara)"]
    Sales[Sales Schema]
    Finance[Finance Schema]
    Product[Product Schema]
    Contact[Contact Schema]
    MarketingDWH[Marketing Schema]
    Reference[Reference Schema]
    Fallout[Fallout Schema]
    Flow[Flow Schema]
end

subgraph Reporting["REPORTING / MARTS LAYER<br/>(DS_Reporting / Mara_
    Dim[Dim Views<br/>Dimensions]
    Fact[Fact Views<br/>Facts]
    Model[Model Views<br/>Business Logic]
end

subgraph BI["BI & ANALYTICS LAYER"]
    PowerBI[Power BI]
    Tableau[Tableau]
    Sisense[Sisense]
    Grafana[Grafana]
    Excel[Excel]
    Custom[Custom Apps]
end

Sources --> Extraction
CRM --> SSIS
ERP --> SSIS
Marketing --> Functions
APIs --> Functions

SSIS --> Stage
Functions --> Stage
```

```
    Stage --> ADF
    ADF --> DWH

    DWH --> Reporting
    Sales --> Dim
    Finance --> Fact
    Product --> Model

    Reporting --> BI
    Dim --> PowerBI
    Fact --> Tableau
    Model --> Sisense

    style Sources fill:#e1f5ff
    style Stage fill:#fff4e1
    style DWH fill:#e8f5e9
    style Reporting fill:#f3e5f5
    style BI fill:#fce4ec
```

---

2. Layer-by-Layer Architecture

Layer 1: Stage / Raw Layer

Purpose: Landing zone for raw source data with minimal transformation

Database Naming Convention:

- `DS_Stage`: Common/reference data (currencies, countries, lookup tables)
- `DS_Stage_[SourceName]`: Source-specific staging databases
  - Example: `DS_Stage_HL` (HealthLabs)
  - Example: `DS_Stage_Stripe` (Payment data)
  - Example: `DS_Stage_GA` (Google Analytics)

Key Schemas:

```
-- Stage Database Structure
DS_Stage
├── [dbo]                    -- Default schema for source tables
├── [History]                -- Change data capture tables
```

```
├── [Reference]             -- Common reference data
│   ├── Countries
│   ├── Currencies
│   └── BU_List
└── [TaxJar]                -- Example: external API data
```

Key Components:

1. **Raw Tables**: Mirror source system structure
    - Preserve source data types where possible
    - Minimal transformations (casting only)
    - Include load timestamp columns
2. **History Tracking**: Automatic change data capture

```
-- Example: History.ProductTable
CREATE TABLE [History].[ProductTable](
    [ChangeType] [nvarchar](100) NULL,    -- INSERT, UPDATE, DELETE
    [ChangeTime] [datetime] NULL,         -- When change occurred
    -- All columns from source table
)
```

3. **Stored Procedures**:
    - `BuildHistoryTracking`: Creates history tables for CDC
    - `CreateHistoryTable`: Generates history table DDL

Data Flow:

```
Source Systems → ETL/Replication → DS_Stage_* → History Tables
```

Technology:

- SQL Server (on-premises or Azure SQL Database)
- SSIS packages for complex extractions
- Azure Data Factory for cloud sources
- Custom connectors for APIs

Layer 2: DWH / Integration Layer

Purpose: Integrated, cleansed, business-ready data warehouse

Database Name: `DS_DWH` or `Mara`

Core Schemas:

```
DS_DWH
├── [Sales]              -- Sales transactions and orders
├── [Finance]            -- Financial data, invoicing, payments
├── [Product]            -- Product catalog and offers
├── [Contact]            -- Customers, users, households
├── [Marketing]          -- Campaigns, channels, publishers
├── [Reference]          -- Framework metadata and lookups
├── [Fallout]            -- Data quality tracking
├── [Flow]               -- Incremental load staging
├── [Bridge]             -- Many-to-many relationships
├── [Interaction]        -- Customer interactions
├── [Inventory]          -- Stock and inventory (if applicable)
├── [ResultLog]          -- Customer journey tracking
├── [Logging]            -- ETL logging
└── [Grafana]            -- Monitoring views
```

Key Table Types:

1. **Dimension Tables** (Type 2 SCD with timeslices)

```
CREATE TABLE [Product].[Product](
    [ProductId] [bigint] IDENTITY(1,1) NOT NULL,
    [ProductNK] [nvarchar](100) NOT NULL,
    [ProductName] [nvarchar](200) NULL,
    [ValidFromDate] [date] NOT NULL,
    [ValidToDate] [date] NOT NULL,
    [SourceSystemId] [bigint] NOT NULL,
    [SourceSystemIdentifier] [nvarchar](100) NULL,
    CONSTRAINT [PK_Product] PRIMARY KEY ([ProductId])
)
```

2. **Fact Tables**

```
CREATE TABLE [Sales].[Order](
    [OrderId] [bigint] IDENTITY(1,1) NOT NULL,
    [OrderNK] [nvarchar](100) NOT NULL,
    [OrderDate] [datetime] NULL,
    [CustomerId] [bigint] NULL,
    [ProductId] [bigint] NULL,
    [Amount] [decimal](18,2) NULL,
    [SourceSystemId] [bigint] NOT NULL,
    CONSTRAINT [PK_Order] PRIMARY KEY ([OrderId])
)
```

3. **Property Tables** (EAV)

```
CREATE TABLE [Sales].[OrderProperties](
    [OrderPropertyId] [bigint] IDENTITY(1,1) NOT NULL,
    [OrderId] [bigint] NOT NULL,
    [PropertyName] [nvarchar](100) NOT NULL,
    [PropertyValue] [nvarchar](500) NULL,
    [ValidFromDate] [date] NULL,
    [ValidToDate] [date] NULL
)
```

4. **Flow Tables** (Dynamic, incremental staging)

```
-- Example: Flow.Sales_Order_42
-- Created by ReadSourceTable_NewRecords
-- Contains only new/changed records for SourceSystemId=42
```

5. **Reference Tables** (Framework metadata)
   - `Reference.SourceSystem`: Master configuration
   - `Reference.SourceColumnMapping`: Column-level mappings
   - `Reference.PropertyReference`: EAV property control

6. **Fallout Tables** (Data quality)

```
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
)
```

Data Flow:

```
DS_Stage_* → Flow Tables → Azure Data Factory Dataflows → DS_DWH Tables -
```

Layer 3: Reporting / Marts Layer

Purpose: Optimized, business-friendly views for analytics and reporting

Database Name: `DS_Reporting` or `Mara_Marts`

Core Schemas:

```
DS_Reporting
├── [Dim]                -- Dimension views (flattened)
├── [Fact]               -- Fact views (with surrogate key joins)
├── [Model]              -- Business logic views
├── [Mobile]             -- Mobile-specific analytics (CC example)
├── [Medgen]             -- Publisher analytics (CC example)
├── [Ecommerce]          -- E-commerce views (CC example)
├── [Finance]            -- Financial reporting views
├── [HL]                 -- HealthLabs specific (CC example)
└── [Tracking]           -- Tracking and attribution
```

Key View Patterns:

1. **Dimension Views**: Denormalized, current-state dimensions

```
CREATE VIEW [Dim].[Product] AS
SELECT
    p.ProductId,
    p.ProductNK,
    p.ProductName,
```

```
    pg.ProductGroupName,
    c.CategoryName
FROM Product.Product p
JOIN Product.ProductGroup pg ON p.ProductGroupId = pg.ProductGroupId
JOIN Product.Category c ON pg.CategoryId = c.CategoryId
WHERE p.ValidToDate = '9999-12-31'  -- Current records only
```

2. **Fact Views**: Pre-joined with dimension names

```
CREATE VIEW [Fact].[Orders] AS
SELECT
    o.OrderId,
    o.OrderDate,
    c.CustomerName,
    p.ProductName,
    o.Amount
FROM Sales.Order o
LEFT JOIN Contact.Customer c ON o.CustomerId = c.CustomerId
LEFT JOIN Product.Product p ON o.ProductId = p.ProductId
```

3. **Aggregation Views**: Pre-calculated metrics

```
CREATE VIEW [Model].[DailySalesSummary] AS
SELECT
    CAST(OrderDate AS DATE) as OrderDate,
    ProductName,
    COUNT(*) as OrderCount,
    SUM(Amount) as TotalRevenue
FROM Fact.Orders
GROUP BY CAST(OrderDate AS DATE), ProductName
```

4. **Hash-Based Group Keys**: For complex dimensionality

```
CREATE FUNCTION [Mobile].[fn_ARPU_GroupKey]
(
    @CarrierId NVARCHAR(150),
    @NetworkId NVARCHAR(150),
    -- ... multiple parameters
)
```

```
RETURNS VARBINARY(32)
AS
BEGIN
    RETURN HASHBYTES('SHA2_256',
        Mobile.fn_HashNVar_NullAware(@CarrierId) +
        Mobile.fn_HashNVar_NullAware(@NetworkId) +
        -- ... concatenate all dimensions
    )
END
```

Data Flow:

```
DS_DWH → Views (Dim, Fact, Model) → BI Tools
```

Layer 4: Sync Layer (Optional)

Purpose: Replication and synchronization between systems

Database Name: `DS_SYNC` or `Mara_Sync`

Schemas:

```
DS_SYNC
├── [Source]            -- Replicas of source system tables
└── [Target]            -- Replicas of target system tables
```

Use Cases:

1. **Cloud Migration**: Replicate on-prem to cloud during transition
2. **Load Isolation**: Reduce impact on production systems
3. **Cross-Region**: Sync data between geographic regions
4. **Hybrid Architecture**: Bridge on-prem and cloud

Key View:

```
-- VW_SyncTableSelection: Identifies tables for merge operations
CREATE VIEW [dbo].[VW_SyncTableSelection] AS
SELECT
    s.name AS SchemaName,
```

```
    t.name AS TableName,
    IDENT_CURRENT(s.name + '.' + t.name) AS CurrentIdentity
FROM sys.tables t
JOIN sys.schemas s ON t.schema_id = s.schema_id
WHERE EXISTS (
    SELECT 1 FROM sys.identity_columns ic
    WHERE ic.object_id = t.object_id
)
```

---

3. Technology Stack

Database Platform

SQL Server

- **Versions**: SQL Server 2017+, Azure SQL Database
- **Features Used**:
  - Schemas for logical organization
  - Identity columns for surrogate keys
  - User-defined functions (UDFs)
  - Stored procedures for framework logic
  - Views for reporting layer
  - System tables for metadata introspection

Configuration:

```
-- Database settings (from DS_DWH.sql)
ALTER DATABASE [DS_DWH] SET COMPATIBILITY_LEVEL = 150
ALTER DATABASE [DS_DWH] SET RECOVERY SIMPLE
ALTER DATABASE [DS_DWH] SET AUTO_UPDATE_STATISTICS ON
ALTER DATABASE [DS_DWH] SET QUERY_STORE = OFF  -- Or ON for performance r
```

ETL & Orchestration

Azure Data Factory (ADF)

Components:

1. **Pipelines**: Orchestration logic

```
{
  "name": "Load_Sales_Orders",
  "activities": [
    {
      "type": "Switch",
      "typeProperties": {
        "on": "@pipeline().parameters.SourceSystemId",
        "cases": [
          {
            "value": "42",
            "activities": [
              {
                "type": "ExecuteDataFlow",
                "dataflow": "LoadSalesOrders_SS42"
              }
            ]
          }
        ]
      }
    }
  ]
}
```

2. **Dataflows**: Transformation logic
    - Flow Source: Read from Flow tables
    - Lookups: Resolve foreign keys to dimensions
    - Conditional Splits: Route to DWH or Fallout
    - Sinks: Write to DWH tables or Fallout tables
3. **Parameters**: Enable metadata-driven execution
    - `SourceSystemId`: Which source to process
    - `SourceTable`: Which source table
    - `FlowTable`: Which Flow table to read

SSIS (Legacy/On-Premises)

Used in earlier implementations (Pricewise/Mara):

- Mara_ETL packages for staging
- Mara_Marts_ETL for reporting layer
- Deployed via SSDT to SQL Server

Migration Path: SSIS → Azure Data Factory

Cloud Services

Azure Functions

Example: `CloudTransferAzureApp`

```
[FunctionName("CloudTransferFromAzureToGoogle")]
public static async Task<IActionResult> CloudTransferFromAzureToGoogle(
    [HttpTrigger(AuthorizationLevel.Anonymous, "post")] HttpRequest req,
    ILogger log)
{
    // Transfer files between Azure Blob and Google Cloud Storage
    string pathFrom = data?.from;    // Azure Blob path
    string pathTo = data?.to;        // GCS path
    string contentType = data?.content_type;

    // Logic for bidirectional transfer
}
```

Azure Blob Storage

- Staging for file uploads
- Archive for historical data
- Integration with ADF pipelines

Google Cloud Storage (GCS)

- Cloud storage for multi-cloud scenarios
- Integration with BigQuery

Google BigQuery

- Cloud data warehouse option
- Analytics at scale

- Integration with GCS and ADF

BI & Analytics Tools

Power BI

- Primary BI tool for modern deployments
- Direct Query or Import mode
- Row-level security support
- Cloud or on-premises gateway

Sisense

- Used in Pricewise implementation
- ElastiCube for in-memory analytics
- Custom dashboards and reports

Tableau

- Enterprise BI platform support
- Live connections to SQL Server

Grafana

- Real-time monitoring dashboards
- Custom schema in DWH: `[Grafana]`
- Time-series visualizations

Excel

- Ad-hoc analysis via ODBC
- Pivot tables and charts
- Power Query for data refresh

---

4. Data Flow Patterns

End-to-End Data Flow

```
flowchart TD
    subgraph Phase1["PHASE 1: EXTRACTION"]
        Source[Source Systems]
        ETLTools[SSIS / ADF / Custom Scripts]
        StageDB[DS_Stage_*<br/>Raw Tables]
        History[History.*<br/>CDC Tracking]

        Source --> ETLTools
        ETLTools --> StageDB
        StageDB --> History
    end

    subgraph Phase2["PHASE 2: PREPARATION"]
        SP[Stored Procedure<br/>ReadSourceTable_NewRecords]
        Flow[Flow.* Tables<br/>Incremental Batches]

        History --> SP
        SP --> Flow
    end

    subgraph Phase3["PHASE 3: TRANSFORMATION"]
        ADFFlow[Azure Data Factory<br/>Dataflows]
        Lookups[FK Lookups]
        Validations[Business Rules<br/>Validation]
        Splits[Conditional Splits]
        DWHTables[DS_DWH Tables<br/>Success Records]
        FalloutTable[Fallout.Fallout<br/>Error Records]

        Flow --> ADFFlow
        ADFFlow --> Lookups
        ADFFlow --> Validations
        Lookups --> Splits
        Validations --> Splits
        Splits -->|Valid| DWHTables
        Splits -->|Invalid| FalloutTable
    end

    subgraph Phase4["PHASE 4: REPORTING"]
        ReportingViews[DS_Reporting Views<br/>Dim, Fact, Model]
        BITools[BI Tools & Applications]
```

```
        DWHTables --> ReportingViews
        ReportingViews --> BITools
    end

    Phase1 --> Phase2
    Phase2 --> Phase3
    Phase3 --> Phase4

    style Phase1 fill:#e1f5ff
    style Phase2 fill:#fff4e1
    style Phase3 fill:#e8f5e5
    style Phase4 fill:#f3e5f5
```

Incremental Load Pattern

```
sequenceDiagram
    participant Source as Source System
    participant Stage as DS_Stage
    participant SP as ReadSourceTable_NewRecords SP
    participant Flow as Flow Table
    participant ADF as Azure Data Factory
    participant DWH as DS_DWH Tables
    participant Fallout as Fallout.Fallout

    Note over Source,Stage: Step 1: Extract Raw Data
    Source->>Stage: Load raw data<br/>(minimal transformation)
    Stage->>Stage: Track changes in History.*

    Note over Stage,Flow: Step 2: Prepare Incremental Batch
    Stage->>SP: Execute ReadSourceTable_NewRecords<br/>@SourceSystemId =
    SP->>SP: Compare SourceIdentifierColumn<br/>to existing DWH records
    SP->>Flow: Insert only NEW records<br/>Flow.Sales_Order_42

    Note over Flow,DWH: Step 3: Transform & Load
    Flow->>ADF: Trigger pipeline<br/>Read Flow.Sales_Order_42
    ADF->>ADF: Apply transformations<br/>FK lookups, validations
    ADF->>DWH: Insert valid records<br/>Sales.Order
    ADF->>Fallout: Log invalid records<br/>Fallout.Fallout

    Note over Flow: Step 4: Cleanup
```

```
    DWH->>Flow: Success confirmed
    Flow->>Flow: TRUNCATE TABLE<br/>(ready for next run)
```

Fallout Handling Pattern

```
flowchart TD
    Start[Flow Table Records] --> Lookup[Lookup Dimension<br/>e.g., Custo
    Lookup --> Check{Customer<br/>Found?}

    Check -->|Yes| Success[Continue Pipeline<br/>Add CustomerId]
    Check -->|No| FalloutPath[Route to Fallout]

    FalloutPath --> CheckExisting{Already in<br/>Fallout?}
    CheckExisting -->|No| AddFallout[Add to Fallout.Fallout<br/>ErrorCode
    CheckExisting -->|Yes| Skip[Skip Duplicate]

    Success --> NextLookup[Next FK Lookup<br/>e.g., Product]
    NextLookup --> NextCheck{Product<br/>Found?}

    NextCheck -->|Yes| FinalSuccess[Load to DWH Table<br/>Sales.Order]
    NextCheck -->|No| NextFallout[Route to Fallout<br/>ErrorCodeId = 14]

    AddFallout --> FalloutTable[(Fallout.Fallout)]
    NextFallout --> FalloutTable

    style Start fill:#e1f5ff
    style Success fill:#c8e6c9
    style FinalSuccess fill:#4caf50,color:#fff
    style FalloutPath fill:#ffcdd2
    style FalloutTable fill:#f44336,color:#fff
```

---

5. Deployment Patterns

Pattern 1: On-Premises

Architecture:

```
graph LR
    subgraph OnPrem["ON-PREMISES DATACENTER"]
        Sources[On-Prem Sources<br/>CRM, ERP, Files]
        SQLServer[SQL Server<br/>Stage, DWH, Reporting]
        SSIS[SSIS<br/>ETL Orchestration]
        BITools[On-Prem BI Tools<br/>Sisense, Excel]
    end

    Sources --> SQLServer
    SQLServer --> SSIS
    SSIS --> SQLServer
    SQLServer --> BITools

    style OnPrem fill:#e1f5ff
```

Characteristics:

- All components in local datacenter
- SSIS for ETL orchestration
- SQL Server for all layers
- Direct network connectivity

Use Cases:

- Data sovereignty requirements
- Low network latency needs
- Existing infrastructure investment

Pattern 2: Cloud-Native

Architecture:

```
graph TB
    subgraph Cloud["AZURE CLOUD"]
        CloudSources[Cloud Sources<br/>Salesforce, APIs, Files]
        ADF[Azure Data Factory<br/>ETL Orchestration]
        AzureSQL[Azure SQL Database<br/>Stage, DWH, Reporting]
        Functions[Azure Functions<br/>API Connectors]
        Blob[Azure Blob Storage<br/>File Staging]
```

```
        PowerBI[Power BI Service<br/>Cloud BI]
    end

    CloudSources --> ADF
    CloudSources --> Functions
    Functions --> Blob
    Blob --> ADF
    ADF --> AzureSQL
    AzureSQL --> PowerBI

    style Cloud fill:#e8f5e9
```

Characteristics:

- Fully cloud-based
- Azure Data Factory for orchestration
- Azure SQL Database or Synapse
- Serverless compute where possible

Use Cases:

- New implementations
- Cloud-first organizations
- Global accessibility requirements

Pattern 3: Hybrid (Most Common)

Architecture:

```
graph TB
    subgraph OnPrem["ON-PREMISES"]
        OnPremSources[On-Prem Sources<br/>Legacy Systems]
        Sync[DS_SYNC<br/>Replication Layer]
        Gateway[On-Prem Data Gateway]
    end

    subgraph Azure["AZURE CLOUD"]
        ADFHybrid[Azure Data Factory]
        AzureSQLHybrid[Azure SQL Database<br/>DWH & Reporting]
        PowerBIHybrid[Power BI Service]
```

```
    end

    OnPremSources --> Sync
    Sync --> Gateway
    Gateway --> ADFHybrid
    ADFHybrid --> AzureSQLHybrid
    AzureSQLHybrid --> PowerBIHybrid

    style OnPrem fill:#fff4e1
    style Azure fill:#e8f5e9
```

Migration Path:

```
graph LR
    P1[Phase 1<br/>Replicate to DS_SYNC] --> P2[Phase 2<br/>Build Cloud I
    P2 --> P3[Phase 3<br/>Migrate BI Tools]
    P3 --> P4[Phase 4<br/>Migrate Sources]

    style P1 fill:#e1f5ff
    style P2 fill:#fff4e1
    style P3 fill:#e8f5e9
    style P4 fill:#f3e5f5
```

Characteristics:

- Sources remain on-premises
- Sync layer replicates to cloud
- Cloud-based processing and BI
- On-premises data gateway for connectivity

Benefits:

- Gradual migration reduces risk
- Maintains existing source systems
- Enables cloud BI tools
- Supports legacy and modern simultaneously

Pattern 4: Multi-Cloud

Architecture:

```
graph TB
    Sources[Various Sources<br/>APIs, Files, Databases]

    subgraph Azure["AZURE"]
        ADFMulti[Azure Data Factory]
        AzureSQLMulti[Azure SQL Database]
        BlobMulti[Azure Blob Storage]
        PowerBIMulti[Power BI]
    end

    subgraph GCP["GOOGLE CLOUD"]
        BigQuery[Google BigQuery]
        GCS[Google Cloud Storage]
        Looker[Looker / Tableau]
    end

    subgraph Transfer["CROSS-CLOUD TRANSFER"]
        CloudTransfer[CloudTransfer<br/>Azure Function]
    end

    Sources --> ADFMulti
    ADFMulti --> AzureSQLMulti
    ADFMulti --> BlobMulti
    ADFMulti --> CloudTransfer
    CloudTransfer --> GCS
    GCS --> BigQuery
    BigQuery --> Looker
    AzureSQLMulti --> PowerBIMulti

    style Azure fill:#e8f5e9
    style GCP fill:#fff4e1
    style Transfer fill:#f3e5f5
```

Characteristics:

- Multiple cloud platforms
- CloudTransfer Azure Function for cross-cloud data movement
- Best-of-breed services from each platform

Use Cases:

- Avoid vendor lock-in
- Leverage specialized services (BigQuery for analytics, Azure for BI)
- Global presence across cloud regions

---

6. Integration Points & APIs

Inbound Integration

SQL Server Native:

- Linked servers for cross-database queries
- SSIS packages for complex extractions
- Bulk insert for flat files

Azure Data Factory Connectors:

- REST APIs
- SFTP/FTP
- Azure Blob Storage
- Google Cloud Storage
- Various SaaS platforms (Dynamics, Salesforce, etc.)

Custom Azure Functions:

- HTTP triggers for webhook-based ingestion
- Timer triggers for scheduled pulls
- Event Grid triggers for event-driven processing

Outbound Integration

Direct Query:

- BI tools connect via SQL protocols
- ODBC/JDBC drivers
- Direct Query in Power BI

Data Export:

- Views expose data to external systems
- Stored procedures for batch exports
- Azure Data Factory pipelines for data publishing

APIs (Custom):

- REST APIs built on top of DWH
- Authentication and row-level security
- JSON/XML output formats

---

7. Security Architecture

Authentication

SQL Server Authentication:

- User accounts per application/team
- Windows Authentication for domain users
- Azure AD integration for cloud deployments

Example Users (from DS_Stage):

```
CREATE USER [CREATIVECLICKS\bi-data-gateway]
CREATE USER [CREATIVECLICKS\SQL-Read-bi-damsquare-sq]
```

Authorization

Schema-Based Security:

```
-- Grant read-only access to reporting layer
GRANT SELECT ON SCHEMA::Dim TO [BI_Users]
GRANT SELECT ON SCHEMA::Fact TO [BI_Users]

-- Deny access to sensitive schemas
DENY SELECT ON SCHEMA::Reference TO [BI_Users]
DENY SELECT ON SCHEMA::Fallout TO [BI_Users]
```

Role-Based Access Control:

```
ALTER ROLE [db_datareader] ADD MEMBER [Sisense]
ALTER ROLE [db_datareader] ADD MEMBER [GrafanaReader]
```

Data Protection

Encryption:

- TDE (Transparent Data Encryption) for at-rest
- SSL/TLS for data in transit
- Column-level encryption for sensitive fields (optional)

Masking:

- Dynamic data masking for PII
- Row-level security for multi-tenant scenarios

---

8. Monitoring & Operations

Monitoring Components

Logging Schema:

```
[Logging]
├── ETL_Log                  -- Pipeline execution logs
├── Error_Log                -- Error details
└── Performance_Metrics      -- Query performance tracking
```

Fallout Monitoring:

```
-- View: Fallout.VW_TotalFallout
SELECT
    ss.SourceDescription,
    ec.ErrorDescription,
    COUNT(*) as FalloutCount,
    MAX(f.InsertDate) as LatestFallout
FROM Fallout.Fallout f
JOIN Reference.SourceSystem ss ON f.SourceSystemId = ss.SourceSystemId
```

```
JOIN Fallout.ErrorCode ec ON f.ErrorCodeId = ec.ErrorCodeId
WHERE f.SolveDate IS NULL
GROUP BY ss.SourceDescription, ec.ErrorDescription
```

Grafana Dashboards:

- Real-time ETL pipeline status
- Data quality metrics
- System performance
- User activity

Operational Procedures

Daily Operations:

1. Staging job runs (sync from sources)
2. Flow tables populated (incremental extraction)
3. ADF pipelines execute (load DWH)
4. Fallout review and resolution
5. Reporting layer refresh

Initialization (one-time setup):

1. Initialize Simple Dimensions
2. Initialize SourceSystem metadata
3. Initialize SourceColumnMapping
4. Initialize PropertyReference

---

9. Performance Considerations

Indexing Strategy

DWH Layer:

- Clustered indexes on surrogate keys (Id columns)
- Non-clustered indexes on natural keys (NK columns)
- Non-clustered indexes on foreign keys
- Filtered indexes on ValidToDate for current records

Reporting Layer:

- Indexed views for frequent aggregations
- Columnstore indexes for large fact tables

Partitioning

Time-Based Partitioning:

```
-- Example: Partition Orders by OrderDate
CREATE PARTITION FUNCTION PF_OrderDate (datetime)
AS RANGE RIGHT FOR VALUES
('2020-01-01', '2021-01-01', '2022-01-01', '2023-01-01')
```

Benefits:

- Faster queries on recent data
- Efficient archival of old data
- Partition switching for maintenance

Query Optimization

Reporting Views:

- Pre-filter to current records (ValidToDate = '9999-12-31')
- Avoid SELECT * in views
- Use NOLOCK for read-heavy queries (where appropriate)

ADF Dataflows:

- Partition data sources in parallel
- Optimize lookup sizes
- Use broadcast joins for small dimensions

---

10. Scalability & Growth

Vertical Scaling

Database Tier Upgrades:

- On-premises: Add CPU, RAM, faster storage
- Azure SQL: Scale up service tier

When to Scale Up:

- Slow query performance
- High CPU utilization during ETL
- Memory pressure

Horizontal Scaling

Read Scale-Out:

- Azure SQL Read Replicas for BI tools
- Separate reporting database (replicated)

ETL Parallelization:

- Process multiple SourceSystems in parallel
- Partition large tables across multiple pipelines

Data Volume Management

Archival Strategy:

- Move old data to archive tables
- Partition by date ranges
- Compress historical data

Incremental Processing:

- Always use Flow tables (never full reloads)
- CDC for change detection
- Optimize SourceIdentifierColumn for fast comparisons

Conclusion

The UDM Framework architecture provides:

1. **Flexibility**: Multiple deployment patterns (on-prem, cloud, hybrid)

2. **Scalability**: Handles growing data volumes and user bases
3. **Maintainability**: Clear separation of concerns across layers
4. **Reliability**: Built-in quality monitoring and error handling
5. **Performance**: Optimized for both ETL and query workloads
6. **Security**: Role-based access and data protection
7. **Observability**: Comprehensive monitoring and logging

This architecture has been proven across multiple industries and use cases, from price comparison platforms to mobile subscription services to e-commerce operations.

Next: 03_Core_Domains.md - Detailed domain schemas and data models