eBIS

Enterprise Data Warehouse Implementations:
Best Practice Recommendations

Agenda

1) **Technical Best Practices For Enterprise Data Warehouse Implementations**
   a. Data Architecture
      i. *Data Layers*
      ii. *Dimensions*
      iii. *Facts & Rules*
      iv. *Measures*
   b. Data Processing
   c. Back-End Infrastructure
   d. Procedures
2) **Lessons Learned from Prior Engagements**
3) **The eBIS Solution Toolkit**
4) **About eBIS**

eBIS

Enterprise Data Warehouse Implementations: Technical Best Practices

Technical aspects of an enterprise data warehouse implementation focus on four primary categories:

1. **Data Architecture**
2. **Data Processing**
3. **Back-end Infrastructure**
4. **Procedures**

Upfront identification and application of best practices in these categories leads to long-term cost savings, solution extensibility and increased business adoption. For large, diversified financial institutions such as Washington Mutual, the need for a proven implementation approach is even more acute due to varied source inputs and high data volumes, complex data modeling requirements and significant regulatory compliance considerations. The ensuing content will identify recommended technical implementation strategies to address these risks.

1. Data Architecture According to Layers of Data Modeling

Conceptually, a data warehouse is a series of data movements. The data model within a warehouse should reflect the reporting and processing requirements unique to each data layer as data progresses from initial capture through to reporting structures. A comprehensive data warehouse approach, which includes not just data capture, but both intra-warehouse and outbound analytic processing and reporting, should include the following 10 modeling layers, at a minimum:

a) **Staging Layer**: Captures input data in the format resident in source systems

b) **Operational Data Store**: Models data for offload reporting in source system formats

c) **Core Data Warehouse**: Conforms all data sources into a normalized data model

d) **Analytic Processing Model**: Data structures necessary to feed a 3rd party application that functions internal to the warehouse database

e) **Analytic Output / Return Feed Model**: Models analytic processing output, both intra-warehouse and as fed back from external analytic vendors (return loop)

f) **Outbound Vendor Model**: Data structures necessary to feed a 3rd party application that functions external to the warehouse database

g) **Metadata Model**: Structures that house data about data. Useful in application integration and reporting.

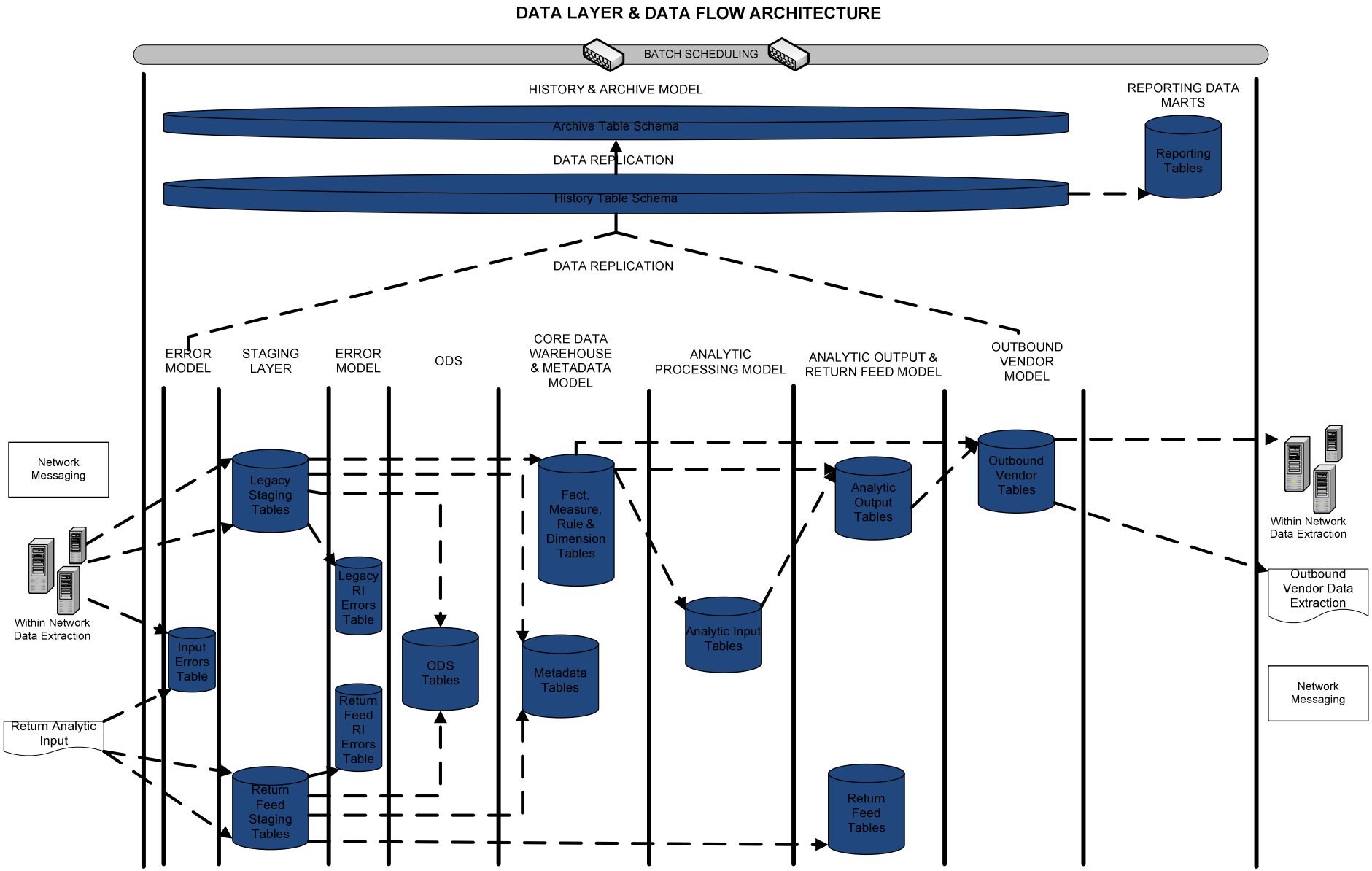
h) **Error Model**: Tables that capture any input data error (dirty data) or processing error

i) **History & Archive Model**: Schemas that replicate underlying data models for reporting (History schema) and long term storage (Archive schema)

j) **Reporting Data Marts**: Star Schema and Snowflake tables relationships constructed according to reporting technology employed: ROLAP, MOLAP, HOLAP.

Data progresses through these layers in specific sequences, depending on data type and reporting need, as shown in the following diagram.

Best Practices: Data Architecture/Data Layers



Best Practices: Data Architecture/Data Layers

Architecture Aspects by Data Layer

Data Layer	Architecture Aspect	Description
Staging Layer	Data Model	✓Embed all processing and reporting requirements (current and potential future) in staging data model ✓Replicate structure of source input data model for required data elements ✓Model time dimensions to allow for data re-extraction
	Data Sources	✓Capture data from both legacy source systems and return loop analytics
	Processing Approach	✓Fully refresh data on every load
	Error Checking	✓Perform only technical data validation on import: duplicate key, field formatting, etc.
	Data Retention	✓Limit data retention to period needed for potential re-processing (short retention period). Reportable rows moved to History and Archive table schemas.
Operational Data Store	Data Model	✓Store portions of staging columns and rows for offload reporting in native data structures (likely a subset of Staging)
	Data Sources	✓Source all data elements from Staging Layer

Best Practices: Data Architecture/Data Layers

Data Layer	Architecture Aspect	Description
Operational Data Store	Processing Approach	✓Fully refresh data on every load in order to replicate source system structures
	Error Checking	✓Perform little or no data transformation from Staging tables to ODS tables to facilitate reporting of native legacy system attributes
	Data Retention	✓Apply data retention and archive rules according to reporting need. Reportable rows moved to History and Archive table schemas.
Core Data Warehouse	Data Model	✓Create conformed, 6th normal form dimensional data relationships with temporal consistency ✓Model complex dimensions encompassing analysis requirements across all data consumers ✓Normalize data model for storage efficiency ✓Employ extensive use of surrogate keys to accommodate incremental source systems ✓Model all data elements identified for conformed analytics and reporting. Absorb future modeling requirements via normalized key structure ✓Model business date and data extraction date as keys (minimum time dimension requirement)
	Data Sources	✓Source from Staging layer and perform extensive data translation into conformed data elements

Best Practices: Data Architecture/Data Layers

Data Layer	Architecture Aspect	Description
Core Data Warehouse	Processing Approach	✓Capture data using a Type 6 slowly changing data compare process if row volumes/row change frequency warrant this approach
	Error Checking	✓Perform referential integrity checks on all rule, fact and measure rows that contain foreign key relationships to dimension tables
	Data Retention	✓Apply data retention rules according to analytic output and reporting traceability requirements. Reportable rows moved to History and Archive table schemas.
Analytic Processing Model	Data Model	✓Use pre-packaged vendor processing model, if applicable, with little or no customization ✓Leverage metadata model to understand relationships between analytic keys and core warehouse keys ✓Build business rules tables to enable multiple versions of rules
	Data Sources	✓Migrate from Core Warehouse Model with conformed key structures intact wherever possible
	Processing Approach	✓Capture data according to delivered processing model. Construct processing engines to translate data from Core Warehouse into Processing Model. *Note:* Analytic processing model is only necessary if using a delivered vendor model. If building custom analytic engines, source directly from Core Warehouse layer.

Best Practices: Data Architecture/Data Layers

Data Layer	Architecture Aspect	Description
Analytic Processing Model	Error Checking	✓Perform error checking according to processing approach. However, source input row and referential integrity checks already performed in Staging and Core Warehouse layers, so likely error checking requirements are minimal.
	Data Retention	✓Apply data retention rules according to analytic and reporting traceability requirements. Reportable rows moved to History and Archive table schemas.
Analytic Output / Return Feed Model	Data Model	✓Employ sophisticated time dimension modeling to track processing of data re-extracted from source systems or re-processed based on new business rules.
	Data Sources	✓Analytic Processing Model, if using pre-packaged vendor processing engines ✓Core Data Warehouse Layer, if using custom calculation engines ✓Staging Layer for return-loop analytics as received from external calculation systems
	Processing Approach	✓Use delivered 3rd party vendor analytic processing engines ✓Develop custom analytic processing engines according to business rules ✓Migrate data from staging layer, transformed to meet conformed modeling standards
	Error Checking	✓In analytic processing, check for processing errors and unexpected data patterns ✓In return feed processing, check for validity of surrogate keys created in data warehouse and returned in a loop after external analytic calculations. Validate referential data as with core warehouse processing.

Best Practices: Data Architecture/Data Layers

Data Layer	Architecture Aspect	Description
Analytic Output / Return Feed Model	Data Retention	✓Apply data retention rules according to analytic and reporting traceability requirements. Reportable rows moved to History and Archive table schemas.
Outbound Vendor Model	Data Model	✓Construct model to meet input data processing requirements in external processing systems. Acts as outbound staging area.
	Data Sources	✓Analytic Output / Return Feed Model ✓Core Data Warehouse
	Processing Approach	✓Select from applicable data input sources and apply transformations to meet format of output data model
	Error Checking	✓Capture processing errors due to code problems or unexpected data patterns
	Data Retention	✓As with Staging layer, limit data retention to period needed for potential re-processing (short retention period). Any reportable rows moved to History and Archive table schemas.
Metadata Model	Data Model	✓Constructed to capture all rules, decode, reference, link, translation, and transformation metadata across technologies in conformed, normalized model ✓Model permits single source for data regression analysis, enabling "cradle to grave" reporting ✓Isolates physical definitions from logical representations, especially within reporting tools

Best Practices: Data Architecture/Data Layers

Data Layer	Architecture Aspect	Description
Metadata Model	Data Model (Cont.)	✓Catalogs object definitions by object category ✓Defines process scheduling periodicity, dependencies, sequencing and relationships to Service Level Agreements (SLAs)
	Data Sources	✓Staging Layer ✓Direct data input via user interfaces
	Processing Approach	✓Migrate data from staging layer or directly insert to metadata tables ✓Slowly change data in target, treated much like rules *Note:* Metadata itself used by all other applications during processing to interpret data inputs before transforming and manipulating them
	Error Checking	✓Capture processing errors due to code problems or unexpected data patterns ✓Referential data checks likely not applicable, as metadata treated as the reference for integrity
	Data Retention	✓Apply data retention rules according to analytic and reporting traceability requirements. Reportable rows moved to History and Archive table schemas.

Best Practices: Data Architecture/Data Layers

Data Layer	Architecture Aspect	Description
Error Model	Data Model	✓Built to capture the following types of errors into various (Data Layers): 1. Duplicate Rows (Staging) 2. Field Formatting Errors (Staging) 3. Business Validity Checks: Sums and Tolerances Thresholds (Staging) 4. Dimensional Referential Integrity (Core Warehouse, Return Feed Model) 5. Surrogate Key Referential Integrity (Return Feed Model) 6. Processing Errors and Data Anomalies (Analytic Output Model, Reporting Data Marts) ✓Separate physical error data capture by layer: Staging, Core Warehouse, Return Feed, and Analytic Output/Reporting *Note:* Return feed model must delineate from other error capture, as the entire row is in error if the surrogate key validation fails
	Data Sources	✓External data sources ✓Staging Layer ✓Core Warehouse Layer ✓Analytic Processing Model ✓History Model

Best Practices: Data Architecture/Data Layers

Data Layer	Architecture Aspect	Description
Error Model	Processing Approach	✓Parse data into Error Model as part of all application processing. Within a technology, build a common procedure that can be called by any routine to check for and report errors in a consistent manner.
	Error Checking	✓Not applicable
	Data Retention	✓Apply data retention rules according to analytic and reporting traceability requirements. Reportable rows moved to History and Archive table schemas.
History and Archive Model	Data Model	✓Replicates data model across all upstream data layers twice: once for core historical reporting, and a second time for long-term archiving ✓History tables indexed and partitioned for optimal data retrieval ✓Archive tables non-indexed to reduce storage overhead
	Data Sources	✓All upstream data sources required for historical reporting
	Processing Approach	✓Data replication, not data manipulation. Options to use 3rd party tool or replication technologies packaged with chosen database software. ✓Synchronicity with upstream layers dependent on reporting latency requirements ✓Facilitates de-coupling of data acquisition and analytic processing from historical reporting, allowing more efficient management of SAN, database, and server resources

Best Practices: Data Architecture/Data Layers

Data Layer	Architecture Aspect	Description
History and Archive Model	Error Checking	✓Capture processing errors due to code problems or unexpected data patterns
	Data Retention	✓Not applicable
Reporting Data Marts	Data Model	✓De-normalizes data into highly efficient star schema and snowflake reporting structures to support ROLAP inquiry ✓Create aggregate tables to support MOLAP/HOLAP inquiry
	Data Sources	✓History Model
	Processing Approach	✓Pre-populate reporting structures from applicable History tables ✓Dynamically access star schemas/snowflake reporting structures for dimensional reporting ✓Perform data aggregation functions ✓Cross-reference reporting vendor tool metadata captured in metadata model during report generation to logically represent fact and measure data mart elements
	Error Checking	✓Capture processing errors due to code problems or unexpected data patterns
	Data Retention	✓Not applicable. Data is populated in custom data marts according to reporting need. All historical input data captured in History and Archive Model and can be pulled into data marts incrementally.

Best Practices: Data Architecture/Data Layers

2. Normalization/De-normalization Strategies

Model data according to dominant need:

a) Reduced storage cost and modeling flexibility **OR**

b) Data retrieval efficiency

The layers of data capture generally adhere to the following normalization approach:

Data Layer	Normalization Approach	Justification
Staging	De-Normalize	✓Source data structures change very slowly ✓Data elements required by system are relatively static
Operational Data Store	De-Normalize	✓Generally mirror source data as captured in staging layer ✓Required reporting elements out of ODS are relatively static ✓Heavy data access volume from reporting tools
Core Warehouse Model	Normalize	✓Long-term storage required for traceability back from reporting structures ✓Normalized key allows flexibility to add source systems and data rows without data model changes
Analytic Processing Model	Normalize	✓Rules based processing requires flexibility in referencing input rows ✓Data retrieval path static based on processing logic
Analytic Output Model	Normalize	✓Maintain normalized analytic keys to capture business rule calculation output
Outbound Vendor Model	De-Normalize	✓Data elements required by external vendor relatively static
Metadata Model	Normalize	✓Normalized key required to absorb incremental data processing logic or technologies
Error Model	Normalize	✓Normalized key required to absorb incremental input sources or technologies
Reporting Data Marts	De-Normalize	✓Dynamic data retrieval efficiency top priority for ROLAP tools ✓Specific reporting structures driven by defined reporting needs

Best Practices: Data Architecture/Data Layers

3. Atomic Data Capture

a) Model data capture the lowest required data grain in Core Warehouse Layer

b) Model data aggregations and hierarchy relationships in Reporting Layer

4. Transformation Management through Conformed Metadata Capture

a) Create metadata repository that spans applications and technologies

i. Transformation / Expression-Centric, with first transformations occurring from Staging into the Core Warehouse Layer

ii. Catalogs source system data model structure for dynamic extraction updates into Staging Layer via messaging

iii. Create transformation tree back to originating data pattern – reverse engineer source input value from reporting keys

b) Facilitates analysis of "The Life of a Row" and meets regulatory mandates such as Basel II "Cradle to Grave" tracking

5. Use of Surrogate Keys for Dimensions, Rules, Facts, and Measures

a) Use surrogate keys to easily adapt to source system additions and changes within existing feeder system rows

b) Provides reporting flexibility to demote surrogate key and display attributes only

c) Eliminates need to maintain integrity between smart key string and non-key attributes

d) Conform all dimensions, facts, and measures, with surrogate keys even if only one source system is applicable; Merger and acquisition activity and technology changes can make the surrogate key very useful in the future

e) Key assignment should occur at the database level: technology agnostic and easily called by any application

f) Use Numeric key string for efficient joins

Best Practices: Data Architecture/Data Layers

6. Reporting Foundation

a) Translate base grain dimension and reference values into hierarchies in reporting layer

b) Use trees and flattened trees to represent hierarchy relationships

c) Use time spans against time dimension grain to drive report generation

d) Determine reporting requirements before building reporting tables. Use a set of requirement variables to determine the appropriate technology to choose (ROLAP, MOLAP, HOLAP):

i. Number of users
ii. Frequency of data refresh
iii. Frequency of report access
iv. Amount of data
v. Grain of data
vi. Complexity of calculations
vii. Drill down requirements

e) Use varied reporting delivery methods, depending on business need:

i. Pull: reporting portal, worklists, other intermediate report staging for analysis at user discretion
ii. Push: email, auto-print
iii. Dynamic creation: ROLAP user inquiry

f) Avoid an attempt to embed reporting of synchronous data via a data warehouse. Heterogeneous sources, including both synchronous and asynchronous data, may feed a reporting tool.

i. Example: Combine total exposure to Countrywide Financial (data warehouse, asynchronous) with current stock price of Countrywide Financial (Bloomberg, synchronous) via chosen reporting tool

Best Practices: Data Architecture/Dimensions

1. Dimensions to Model

a) Model dimensions that are necessary for robust reporting, such as:

Dimension	Description
Account	✓Fed from Financials General Ledger definition of Balance Sheet, Income Statement, or Off Balance Sheet Accounts
Balance Type	✓Type 1: Identifies calculation method related to time dimension, e.g., Average, High, Low, Beginning, End, Mid-Point ✓Type 2: Groups Accounts for analysis, e.g., Accrued Interest, Notional Amount, Principal Balance
Channel	✓An avenue through which sales are executed ✓Examples: Internet, Branch, Private Bank, Wholesale
Customer	✓Defines a counterparty in a transaction ✓Can be either external, internal, or potentially internal
Department	✓Defines the lowest grain of entity definition in a warehouse
Geography	✓Identifies geographic regions ✓A one-to-one or many relationship to the locations defined on other dimensions, e.g., Department, Customer

Best Practices: Data Architecture/Dimensions

Dimension	Description
Legal Entity	✓Defines the grain of entity used for external reporting ✓A one-to-one or many relationship to Department
Time	✓Defines the temporal context of a row of data ✓Meaning dependent on type of data (Rule, Fact, Balance, etc.) relative to data population method (Change Data Capture, Full Refresh) ✓Multiple time dimensions can be used together on a single row to create detailed representations of data applicability
Product	✓A good or service that is bought or sold internally or externally ✓A product can be modeled several ways, often simultaneously, against a common grain of data a) Financial Identifies the financial attributes necessary to price, and determine the value of, a product for external reporting b) Profitability Identifies the financial attributes necessary to price, value, and assess the riskiness of a product for internal management reporting c) Credit Identifies common credit risk attributes: Extension of Credit, Business Context, Term, Mitigation, etc

b) Model every required dimension with only those attributes that are unique within its key definition and populate the dimension key as a foreign key on facts and measures

c) Dimension attributes may also be foreign keys of other dimensions, e.g., Time

Best Practices: Data Architecture/Dimensions

2. Time Dimension Requirements

- Fact/Measure Time Dimensions
  - a) Extraction Date
    - i. Facilitates re-extraction of bad data and back values instead of updates to existing rows
  - b) Business Analysis Date
    - i. Represents the date applicable for analysis of a row in a business context
    - ii. Intra-day volatility considerations can necessitate grain modeling below day, e.g., hour or minute
    - iii. Regional / Business Unit specific holidays should be applied to the analysis of this date
  - c) Fiscal Year / Accounting Period Calendar
    - i. Tied to Legal Entity dimension to allow for varying fiscal years, e.g., April 1st is Accounting Period 1
  - d) Warehouse Processing Date
    - i. Date the row of data was processed within the warehouse
  - e) Publication Date
    - i. Date the row of data was deemed valid for analysis by downstream users

  Note: Assumes single date dimension, business date, is modeled in source systems

- Rule Time Dimensions
  - f) Effective Date
    - i. Beginning date for analysis; applicable until newer row of data received against like non-time keys
    - ii. Requires change data capture data population for appropriate analysis
  - g) Last Update Date/Time
    - i. Date and time that a user last updated the row of data
    - ii. Non-key time dimension to track user activity

Best Practices: Data Architecture/Dimensions

3. Use of Reference Tables

Model all types of reference data, not just complex dimensions. Categorize table creation according to the following reference types:

a) Reference (Simple Dimension)
   i. Conformed high level key with limited attributes
   ii. Includes both custom reference data, e.g., Industry Code, and common reference data, e.g., Standard Industry Code (SIC)

b) Decode
   i. High level key with description field

c) Translate
   i. Translates source system specific value into conformed warehouse value

d) Conversion
   i. Tracks relationships between old data values and current data values, e.g., merged dimensions

4. Restatements and Conforming old data values to new Dimension Members

a) Use conversion tables for merging dimensions

b) Use conversion tables in combination with surrogate keys for consistent analysis of changing source system inputs through time

Best Practices: Data Architecture/Facts & Rules

1. Abstraction of non-key attributes to minimize data model complexity

a) Normalize fact and rule records by key value of 'Attribute Type'

b) Model Non-key columns based on common input field formats

c) An important balance must be drawn between logical groupings of attributes by table and data model transparency

   i. Grouping at too low a level leads to confusion on row placement and complexity in data extraction

   ii. Grouping at too high a level leads to significant non-key column abstraction and low data cardinality

2. Use of 'Master' tables

a) Implement master tables for facts and rules, as well as dimensions, which contain every row at a given data grain and are resolved for surrogate key assignment

b) At a minimum, master tables must contain the columns used for surrogate key assignment, but may also contain elements common to that data grain, depending on the chosen data model approach

3. Use of status indicators on fact tables vs. rule tables

a) Fact: Use conformed status value to add texture to observed balance value

   i. A fact attribute status in isolation is meaningless, other than to validate the absence of a balance

   ii. In situations where a fact status conflicts with an attendant measure, the measure dominates

b) Rule: Drives interpretation and applies to equivalent processing or analysis date

   i. An inactive rule should be bypassed during analysis, whereas an inactive fact adds a point of analysis to a balance, sometimes indicating a source system problem if the two conflict

Best Practices: Data Architecture/Measures

1. **Modeling of Zero balances**
   a) Do not populate, except in back value and aggregation feeds
      i. In most situations, the absence of a row can be interpreted the same way as a zero balance
   b) Useful in back value situations, to update a non-zero position to zero
   c) Useful in aggregate feeds, where non-zero balances can indicate a meaningful summed calculation to zero (Example: Mark-to-Market balances for derivatives pools)
   d) Trend analysis on back value frequency allows for monitoring of improvements in source system balance capture
2. **Non-Daily system balance modeling: Pull forward balances vs. single input balance**
   a) Model non-daily system balances based on frequency of data capture in Staging, ODS and Core Warehouse Model
   b) Pull forward non-daily system balances for consistent daily portfolio modeling into analytic model and reporting layers
      i. Care must be taken to match the balance frequency between source values and two points of analysis: risk/profitability analytics and external reporting tools, such as the General Ledger

Best Practices: Data Architecture/Measures

3. Multiple representations of balance calculations via Normalized Model or Scenario key

a) Create a normalized analytic output key to represent the rules used to calculate a balance. Examples of balance calculation rule types:

i. Regulatory

ii. Accounting

iii. Credit

iv. Financial Profitability

b) Quells risk and profitability modeling frustrations, such as: "I won't be held hostage to an accounting rule"

4. Currency modeling in Measures

a) Identify currency balances required across all reporting in Core Warehouse layer

i. Transaction Currency

ii. Base Currency of Legal Entity

iii. Consolidated Currency

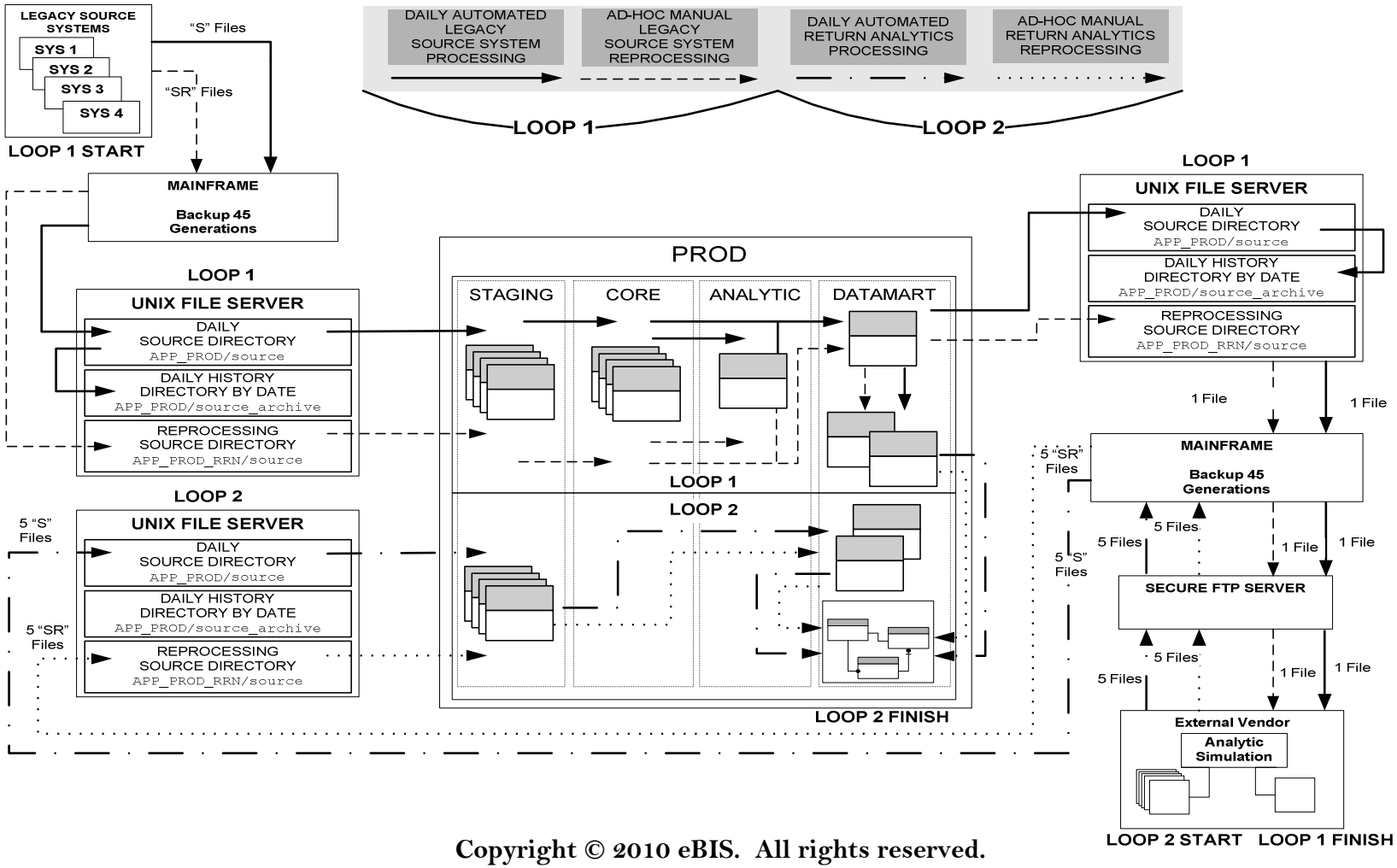
b) Convert from transaction currency to required currencies during staging to core warehouse transformation step

c) Additional currency translations can be performed for specific reports against the transaction currency amount in the Reporting Layer of the warehouse; convert in the Warehouse Model only the currency balances that are consistently analyzed

Best Practices: Data Processing

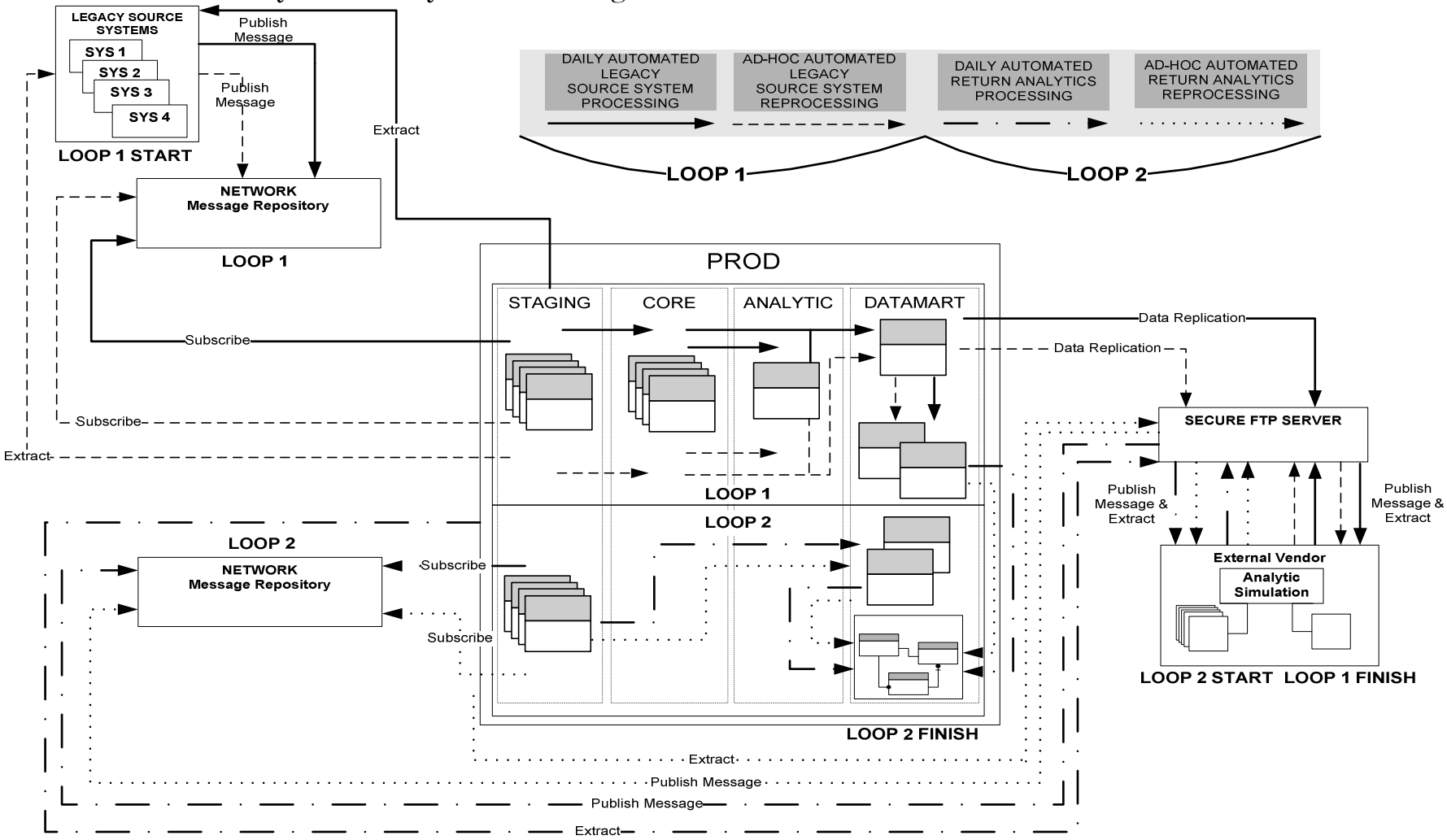
1. Inbound/Outbound Data Processing

a) Endeavor to leverage a Service Oriented Architecture driven by publish and subscribe network messaging as opposed to file system based triggers. A typical file system-driven warehouse processing framework is depicted below.



Best Practices: Data Processing

b) Architect network messaging to be either synchronous or asynchronous, via publish and subscribe technology, with data extraction and replication according to business need and processing capacity constraints. A message-driven processing architecture is depicted below, which reduces data redundancy and latency and file storage costs.



Best Practices: Data Processing

2. Tool-Based Load Validation

a) Develop self-correcting mechanisms, such as stored procedures, to correct input data before it is loaded

Example: Trigger re-creation of staging table to match source input if appropriate change notification is received

3. The Change Data Capture (CDC) Implementation Decision

a) Follow 2 vectors to determine the appropriate data capture approach: CDC or data refresh

i. Slow vs. Fast Changes

ii. Data Volume vs. Join Complexity

Examples: A source with fast changes and low data volume is a good candidate for data refresh.

A source with slow changes and high data volume is a good candidate for CDC, depending on its use in reporting/join complexity

b) If implemented, Change Data Capture should be target based

i. Execute all transformations before comparing input row to target

4. Re-Extract vs. Re-Process Paradigm: Integration with Error Reporting

a) Re-extract rather than re-process error rows and columns

b) Source systems own source data and should correct duplicate rows, formatting errors and dimensional referential inconsistencies and then re-extract data for downstream use

c) Re-processing of error rows, with error correction within the data warehouse, obviates source system data ownership, which is counter to a warehouse's purpose. A warehouse should act as an analysis arena, with data remediation executed in providing systems, and then re-extracted to a warehouse.

d) For referential integrity, over-write realized errors in row columns to a conformed representation of an error, e.g., '00000'

e) Realized errors in rows (duplicates) are more difficult to interpret. Capture the first of the duplicate rows for onward processing and parse the duplicates to an error table for remediation.

f) Endeavor to automate notification to source systems of data errors via network messaging and automate the re-extraction of data via publish and subscribe as if it were a normal daily feed

Note: The time dimensions modeled in all of the warehouse data layers should accommodate both re-extraction and normal daily processing

Best Practices: Data Processing

5. Technology Agnostic Batch Scheduling/Message Management

a) Use APIs to centrally schedule and manage batch load

b) Employ significant use of dependencies for event-based triggers

c) Employ both time and event-based processing to ensure process execution in cases where event triggers are not realized

d) Ensure that time-based triggers do non pre-maturely instantiate processing in instances of early arriving facts or late arriving dimensions

   i. If scheduling is not configured properly, misleading errors could result in Error Model

e) Create system specific scheduling metadata that identifies a calendar of data availability

f) Consolidated daily processing dependencies read from universe of system-specific metadata calendars

g) Create a logical processing flow according to processing schedule dependencies that can be validated against observed execution

   i. Create negative error reports to highlight data inputs that were expected but not received

h) Subscribe at the lowest temporal frequency, e.g., daily, regardless of projected data availability, e.g., daily, monthly, quarterly, and incorporate expected input periodicity in negative error reporting

Best Practices: Data Processing

6. Processing Based on Common Run Control Parameters

a) Define a set of parameters that drive processing and feed those parameters into a common program to resolve from source and delete from target

b) In the context of complex time dimension modeling, endeavor to supply a single time dimension to drive processing, the interpretation of which depends on the type of table being resolved (Balance vs. Fact vs. Rule) and the type of process being executed (Source data capture vs. Analytics vs. Reporting)

i. If parallel processing is required against multiple dates, supply a processing time-span to the run control and let the resolve process parse data into date threads

7. Table Reads

a) Employ the use of uncommitted reads, tied to batch dependencies, during processing

b) Dependency metadata ensures all relevant data is available when a read is invoked

c) Uncommitted database reads avoid table locking and database-level time-outs

8. Use of inserts in processing

a) Avoid update statements, which can cause temporal inconsistencies

b) Recommendation for partitioned data processing to facilitate multi-threaded process execution within (data chunking) or across (multi-batch) time dimensions

c) Normalization strategies enable inserts on target tables

d) Time dimension modeling enables inserts on re-extract or back-value feeds

e) Inserts used in combination with target delete logic based on source input time dimension combinations and other processing keys

Best Practices: Data Processing

9. Use of non-indexed tables

a) Employ non-indexed tables wherever possible for efficient processing and inserts

b) Rely on Data Resolve processing, e.g., parsing data into temp table space for processing, and target table index keys for data integrity; allow intermediate processing to occur without index overhead

10. Consolidations and Eliminations

a) Provide visibility to rules that govern data consolidation as defined within General Ledger/Financial Reporting application

b) Apply rules consistently during processing across data grains (Instrument, Facility, Ledger) and data layers (ODS, Core Warehouse, Reporting, etc.)

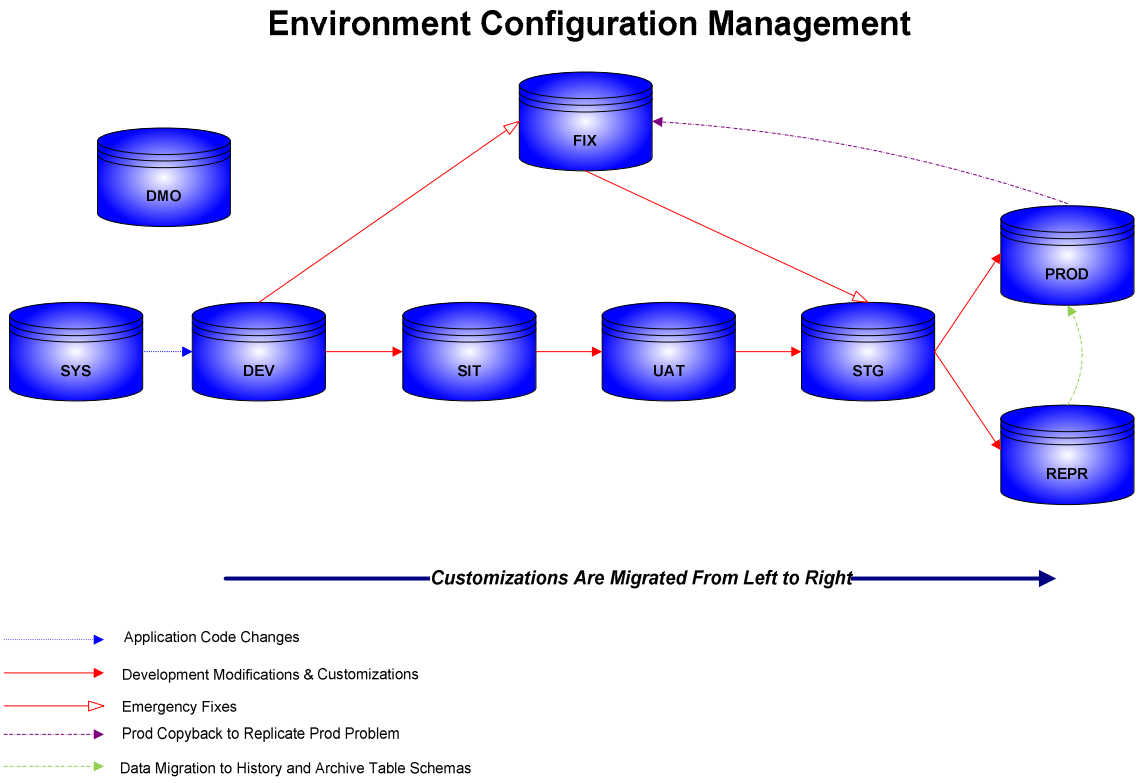
c) Consistent application of consolidation rules within the warehouse allows for tight reconciliation of low data grains, e.g., Instrument and Facility, with General Ledger

Best Practices: Back-end Infrastructure

1. Database Instances

a) Recommended databases and their functions

i. DMO: Training/Demonstrations/Sandbox Exploration
ii. SYS: Application code changes & Upgrades
iii. DEV: Development and unit testing
iv. SIT: End-to-End batch integration and capacity tests
v. UAT: Business logic user acceptance testing
vi. STG: Production migration staging
vii. PROD: Production Processing, Historical reporting, and Archiving
viii. FIX: Production Fixes
ix. REPR: Historical processing or re-processing



Best Practices: Back-end Infrastructure

2. Data Archive Strategies: Infrastructure impacts

a) Buy big server boxes! Favor multiple servers servicing a single database as opposed to multiple databases on multiple servers

b) Consolidate History and Archive table schemas on same database instance as Production schema

c) Rule-based data migrations from Production → History → Archive table schemas within single database environment

3. Historical Re-Processing

a) Re-processing is needed in cases of business rule restatements, mergers, and acquisitions

b) Should occur in REPR environment configured for significant temporal parallelism and variable, but usually high, data volumes; contrasts with PROD, which is configured for a set of daily processes, where input data volumes are stable and output data normally grows linearly.

c) Data migration should occur from REPR environment to history and archive table schemas within PROD environment, with tight archive rule coordination

Best Practices: Back-end Infrastructure

4. Refresh and Restore Strategies

a) All environments should be refreshed from PROD instance

b) Incremental development in test environments (DEV, SIT, UAT) and not in PROD, as identified through compare reports, should be backed up prior to refresh and then restored

c) The frequency of environment refresh depends on factors such as testing need (DEV, SIT, UAT), realized production problems (FIX) and historical reprocessing requirements (REPR)

d) Endeavor to not allow test environments to become stale; the efficacy of test cycles rely on accurate and relevant test data

5. Redundancy Approaches

a) Create two levels of database environment redundancy: Failover and Disaster Recovery

i. Failover environment writes to SAN synchronously with PROD environment and provides a real-time image of PROD data

ii. Disaster recovery is a cold database backup of PROD, replicated at set intervals, usually weekly. A disaster recovery restore, combined with PROD database logs, provides a method to restore and recover from a hot database "double default"

6. Levels of Security Administration

- Apply Security at Three Levels:

a) Server / Operating System

i. Database command restrictions – update, delete, create, select – based on OS User Id

b) Application

i. User Id level security for application interfaces and batch scheduling

ii. Row or column level security to address data sensitivity concerns in reporting layer

c) Network

i. User Id sign-on integration across applications, e.g., LDAP

ii. I.P. address authentication for data transmission encryption within and across networks

Best Practices: Procedures

1. **Development standards**
   a) Define best practice, repeatable development standards for
      i. Object naming and formatting
      ii. Object structure
      iii. API use
      iv. Batch scheduling
      v. Security integration
   b) Apply consistently across technologies and project teams
2. **Test technical development in three distinct phases**
   a) Unit Testing
      i. Developer-centric testing of defined business requirements during development cycle
   b) Batch integration and performance testing
      i. Full data cycle testing of batch metadata dependencies and subscribe technology
      ii. Full production processing simulation with capture of capacity utilization metrics; analyze performance against existing SAN, O/S, DB, and server configurations
   c) Business user acceptance testing
      i. Source table to Target table (across warehouse data layers) and User Interface testing
      ii. Row counts, then attribute and measure validation with scripts that replicate processing code
      iii. Regression test all previous passed tests once the last business test is satisfied
      iv. Validate all downstream processing output as business integration test

Best Practices: Procedures

3. Object Migration Procedures

a) Match environments to definition of test phases

i. DEV → Unit Testing

ii. SIT → Batch integration and performance testing

iii. UAT → Business User acceptance testing

b) Create standard migration request form template to track acceptable objects and data for migration, and define procedures to rebuild the objects in the target environment

c) Define criteria that must be satisfied, with appropriate oversight and approval, for promotion across environments

d) Separate migration from development duties by designating an application administrator who manages all migrations

e) Validate objects and data in target to ensure successful migration and environment consistency

4. Implement version control with all warehouse database objects

a) Accomplished through delivered vendor package, versioning add-on or through file-based project backups of all objects migrated across environments

i. Enables roll-back of object development should it be necessary during testing cycles or as a Production fix

Note: Data versioning accomplished via time dimension keys within tables

Best Practices: Procedures

5. Change Management

a) Employ a central calendar for Data, Technology, etc. changes with explicit approval by effected parties

b) Apply consistent procedure both upstream and downstream (Warehouse affecting other applications)

6. Error Remediation

a) Analyze errors captured in Error Model across layers of data movement

b) Implement communication plan with source system and warehouse technology owners to disseminate reporting of errors

c) Establish remediation procedures to improve source data capture or processing logic

d) Track efficacy of fixes against relative frequency of error capture

Experience Behind Recommendations: Lessons Learned from Prior Engagements

Best practices emerge from experience. Successes, and perhaps more importantly, failures, form extensible knowledge capital. The preceding technical recommendations were distilled from experience at over 15 enterprise data warehouse implementations at large banking or broker/dealer institutions. The following chart summarizes the areas in which we learned important lessons, both from our successes and failures:

Category	Lesson Number	Area	Lesson Description
Project Planning	**1.**	Business & Information Technology Cooperation	Project success is usually constrained by divergent Business and IT goals. Ensure Sponsorship clearly defines and communicates Program goals and obtains buy-in from both business and IT stakeholders.
	2.	Effort/Time Estimation	Large, complex data warehouse projects take longer than initially anticipated to implement successfully. Don't be afraid of the truth! Allocate sufficient time for project planning at two key junctures: end of Scope phase and end of Design phase. The level of implementation effort is not know with certainty until a design is approved. Take the path of best design, which lowers the cost of long-term ownership, and revisit project planning against that design. Allocate sufficient project slack to account for Subject Matter Expert (SME) over-capacity throughout a project lifecycle, especially leading up to and during the test phase.
	3.	Implementation Phases	Implement a project in phases to reduce scope risk and allow calibration of implementation techniques. Recommended Phases: 1) Proof of Concept; 2) Source Data Capture; 3) Analytics/Outbound Integration; 4) History/Archive; 5) Reporting; 6) Parallel/Steady State
	4.	Production Parallel Phase	Allocate sufficient time to execute a new data warehouse in parallel with legacy reporting solutions. Before cut-over, a significant effort is involved to migrate, convert and cleanse production data used in the parallel phase that does not meet cut-over standards.

Lessons Learned From Prior Engagements

Category	Lesson Number	Area	Lesson Description
Project Planning	**5.**	Oversight	Structure an implementation program with adequate checks, balances, and approvals through oversight committees. Leverage skill sets to review and approve across program layers and development lifecycle.
	6.	Resource Skill Sets	Clearly define the roles and responsibilities of required resources. Identify attendant required skill sets, and acquire only those resources who have those skills. Do not assign resources to roles simply because they are available or of low cost!
	7.	Estimation of Support Need	Include Support estimation as part of Design Phase. Identify required skill sets for long-term support and recruit, hire, and train support staff during Development Phase. Allocate sufficient time for SMEs to transfer knowledge to internal staff.
Data Architecture	**8.**	Delivered Data Models	Don't buy into a delivered vendor data model if Enterprise-wide data reporting is required. Model data based on source systems and create fact records based on logical groupings and common physical columns. Invest time to model robust dimensions. A warehouse is only as good as the attributes of its dimensions.
	9.	Data Acquisition & Reporting	Reporting requirements should drive data acquisition. Analyze reporting and data security requirements of constituents upfront. The latency in project roll-out between data acquisition and reporting does not mean reporting requirements should be analyzed last. The grain of data acquisition should be at least as granular as that of the lowest level report detail. Grain and volume of data to report against should influence the selection of a vendor tool by required technology (ROLAP, MOLAP, HOLAP) and resulting data mart schemas.
	10.	Normalization	Normalize the core warehouse, metadata and error models as much as possible. Change is inevitable, and normalized table structures, along with surrogate keys, can adapt easily to new data sources and modeling requirements.

Lessons Learned From Prior Engagements

Category	Lesson Number	Area	Lesson Description
Data Architecture	**11.**	The Time Dimension	The time dimension requires the most forethought for proper modeling. It allows analysis of back valued and re-extracted data patterns if modeled correctly. Lowest temporal grain modeling permits translation to varied fiscal calendars.
	12.	The Customer Dimension	Take care to identify potentially internal customers. Regulations O, 23A, and 23B all govern reporting of transactions with closely held or related subsidiaries. An internal designation is often a function of a reporting view, e.g., Parent vs. Bank, so "potentially internal" is a warehouse-assigned flag that should drive analysis in specific reports.
	13.	Metadata Management	Define a metadata repository that conforms object definition and data transformation across technologies. In particular, clear definition of surrogate keys allows for consistent key use by warehouse tools. The handshake between technologies is one of the tallest implementation hurdles to overcome, and conformed metadata helps integrate disparate toolsets.
Data Processing	**14.**	Application Messaging vs. File System Data Flow	Implement seamless application integration via a Service Oriented Architecture with heavy use of publish and subscribe network messaging. File-based data movement is time instead of event-based, consumes significant file system resources, and requires manual source system intervention in cases of data re-extraction.
	15.	Processing/Batch Management	Generalize the batch management tool to process independent of application technology. Establish batch scheduling metadata that drives batch execution according to input messages and defined processing predecessors. Create processing calendars, defined at the lowest temporal execution grain.
	16.	Run Control Configuration	Use consistent run control parameters to govern batch execution. Create categories of run controls, usually by technology, that define the parameters fed to processes at execution. Each category should have a consistent set of parameters that drive processing. Otherwise, automated update management of the run controls is much more difficult.

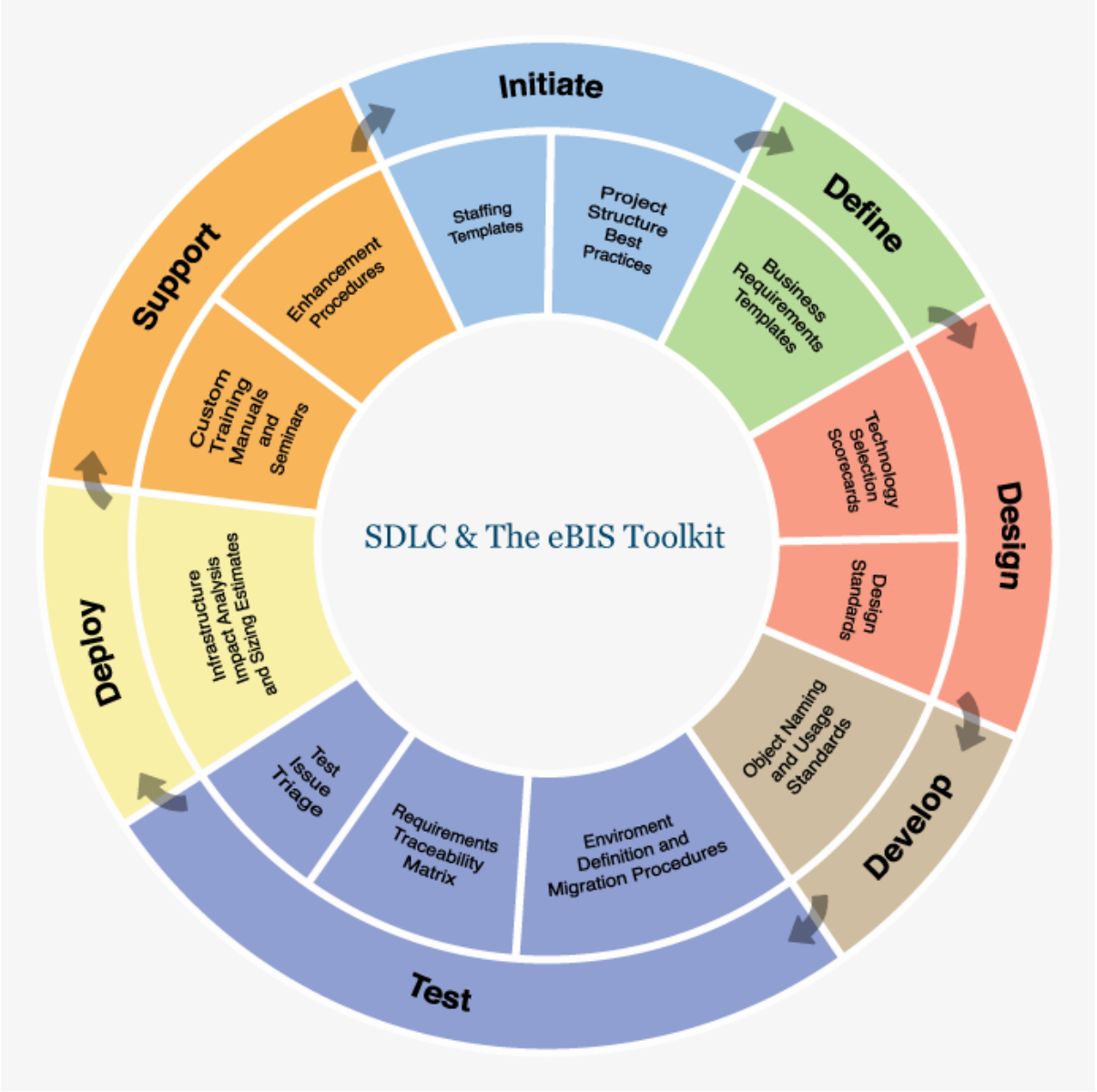
Lessons Learned From Prior Engagements

Category	Lesson Number	Area	Lesson Description
Data Processing	**17.**	Re-Extraction of data from source systems	Do not delete and re-insert data within a data warehouse. Previous versions of data may have been disseminated and analyzed in downstream applications and reports. Implement processing logic that inserts new versions of old data in concert with time dimension modeling in target tables. In this way, prior analysis can be married with previous data versions.
	18.	The Change Data Capture (CDC) Decision	Don't automatically implement CDC. While changing data rows slowly reduces storage cost, it introduces time dimension join complexity that can impede downstream processing efficiency. A data full refresh, combined with data archive rules, may offer more advantages. Strategize data capture by data type and input source, and thoroughly weigh the costs and benefits of an approach before choosing a design.
	19.	Technology agnostic surrogate key assignment	Surrogate keys are critical in data models to consistently identify a rule, fact, or dimension through time. Data sources and data patterns that identify unique rows are fluid, and a data warehouse can react dynamically to these changes if it assigns surrogate keys. The process to assign these keys should occur at the database level via a database function, not through a specific vendor tool. In this way, any technology can assign surrogate keys in any data layer through a database function call.
Back-end Infrastructure	**20.**	Development Platform	Select a vendor tool that provides a flexible development platform. Enterprise Data Warehouses for financial institutions invariably require significant customization and development work; the flexibility of the toolset, especially availability of APIs, is critical to success.
	21.	Database Instance Management	Create database instances according to testing phases, production fixes, and historical reprocessing needs. In each instance, integrate data capture, analytics, and reporting in a consolidated data model. Doing so permits a single production database instance, which minimizes data redundancy and maximizes metadata transformation visibility. Include processing, as well as history and archive, table schemas in the same instance to feed reporting technologies seamlessly.

Lessons Learned From Prior Engagements

Category	Lesson Number	Area	Lesson Description
Back-end Infrastructure	**22.**	Security Administration	Administer security at a minimum of three levels: Network, Operating System, and Application. Do not neglect the importance of data security, both while in transit across the network and once captured in a database, especially conformance to regulatory requirements.
	23.	Infrastructure Capacity and Performance	Leverage benchmarks and conduct capacity tests early in a project lifecycle to estimate hardware needs. Extrapolate observed capacity test results in relation to estimated database instances, data volume, and processing volume to project optimal hardware configuration.
Procedures	**24.**	Object Naming & Development Structure	Apply object naming and development structure standards across data modeling layers and technologies. Centrally monitor object creation and naming standards. Consistency of object structure and common identification across technologies leads to transparency and increased adoption rates.
	25.	Version control	Implement a procedure to track changes in development objects over time. Version tracking enables easier analysis of Production problems and validation of database instance migration requests.
	26.	Data & Application Changes	Establish upstream system protocols to clearly define and communicate system changes. Require downstream acknowledgement of changes prior to implementation. Establish central IT change calendar to track pending changes.
	27.	Data Error Remediation	Data is invariably dirty once enterprise standards are applied. The main benefit of data warehousing is the ability to analyze data consistently, regardless of source system or source data model. When data errors are discovered, implement a procedure to fix input data errors within source systems, rather than within the warehouse. This approach will ensure data quality forward in time. Source data owners must be willing to participate actively in the data governance process.

The eBIS Solution Toolkit



About eBIS

eBIS is a privately held strategy consulting and technology solutions company with close to ten years of experience in bridging gaps between business ideas and technology solutions for the financial services industry. Leveraging understanding of both enterprise financial risks and technologies that can quantify and mitigate them, eBIS partners with clients to deliver value-added business solutions. eBIS specializes in strategic advisory services, systems architecture engineering and risk analytics modeling using proven best practices, reusable solution toolkits and innovative problem solving. The company's client list includes top ten international and U.S. financial institutions in commercial and retail banking, investment banking and asset management. www.ebis.biz