1. Introduction
   1. Purpose of the Reference Architecture

This document describes the foundation of the Enterprise Data Warehouse (EDW) and Business Intelligence platform, with the details showing each element fits together in the architecture. The document also provides the principles and guidelines to enable the design and development of Business Intelligence applications together with a Data Warehouse foundation that is scalable, maintainable and flexible to meet business needs.

These high level designs and principles direct the technical implementation and components, which are intended to be capture into a Solution Architecture document for each delivery. The Solution Architecture captures which elements of the architecture are used and the reasoning for this, which is deployment-specific.

For every project it is required to fine-tune and match the Solution Architecture using these reference designs. Only by carefully selecting which parts should be adopted a fit-for-purpose implementation will be achieved.

### Objectives from a business perspective

To the business, the architecture focuses on:

* Facilitating business decision-making and performance-monitoring processes through timely delivery of integrated and consistent information across the entire organisation.
* Enable the business to focus less on data gathering activities and more on information analysis and use.

### Objectives from a technology perspective

In the context of technology and implementation this reference architecture provides the boundaries (‘playing field and rules’) where ETL patterns, concepts and best-practices can be developed. The framework ensures that:

* All the data in the environment is accessible through a common and user-friendly interface.
* The right information is provided to the right users in the right format, and in a timely manner.
* The physical location and structures of the data are transparent to the users.
* The data is integrated (where applicable), consistent and can be reconciled.
* The information (query results, reports, etc.) provided to the users is consistent and can be reconciled.
* Application components are ‘built once, reused many times’.

To achieve this, the architecture describes on a high level:

* What the steps for data integration are, and in what order they should be processed.
* What the options and considerations are during these steps.
* What the structure of the database should look like.
* How metadata is used and linked to the architecture layers.
* How error handling and recycling is used and linked to the architecture layers.

This document references the individual documents for each layer in the architecture and provides the overview of how these documents are related. These detailed conceptual architecture designs are used to document specific ETL requirements such as:

* Ability to be rerun.
* Data traceability (audit trail).
* Operational metadata and analysis.
* Standardisation.
  1. Using the reference architecture for the ETL Framework

This document should be used to obtain an overview of the reference architecture and its components. The most important aspect is to understand which ETL and EDW concepts are used in what Layer of the architecture and the reasoning behind this.

The reference architecture serves as an outline to position ETL patterns and best-practices. The main purpose is to create a common ground where every developer starts with the same background knowledge to relate new concepts and techniques to.

The reference architecture also provides the basic structure for the documentation of the ETL Framework, as every component of the design and implementation relates back to this architecture, including tips, tricks and examples of implementation options. The implementation solutions (solution patterns) for this architecture are designed to be as generic as possible without losing practical value.

Being a reference architecture, not all components have to be necessarily deployed for every project. In most scenarios specific components are integrated in existing solutions or structures. For this reason the solutions will be as designed to be as modular as possible to enable the utilisation of specific components independently.

1. Principles of the Reference Architecture

No reference architecture or model can exist without adopting basic principles. This paragraph lists these principles along with the fundamental ideas.

* **Hybrid model approach**. Traditionally Data Warehouse models have been classified as either fully normalized (early Inmon) and fully denormalised (Kimball). A hybrid approach utilises components of both concepts. A degree of normalisation ensures that every meaningful (business) entity has its own separate table for distributing surrogate keys and one where history is stored in a traditional Type-2 (denormalised) fashion. Depending on the chosen modelling technique relationships can also be modelled separately. Examples of hybrid approaches are Data Vault, Anchor, Head-and-Version and Matter.
* **Separation of Data Warehouse concepts**. The core functionality within a Data Warehouse is divided in separate ETL steps. This is different to loading a typical Kimball dimension (Dimensional Bus Architecture) where keys, structure and history are combined in the ETL process. Separating these functions provides additional flexibility and maintainability in the future. This includes:
  + Surrogate key distribution or hashing of business keys
  + Storing and tracking history (managing time-variant data)
  + Structure and hierarchy
  + Cleaning and integration (business rules)
* **Flexibility**. A common pitfall of Data Warehouse models is the design (modelling) for a current need of information or specific business requirement. This leads to data being modelled specifically for a set purpose as defined in a given project. However, this often also limits the future usage of the Data Warehouse a future requirements may impact the design or there are differences in the interpretation of data across the organisation. Designing for flexibility in this context means designing to be future proof. This is major goal of the reference architecture that includes:
  + The ability to load data even when relationships between data change. Most hybrid modelling techniques use many-to-many relationships separate of the main entities, even when the data could currently be modelled as one-to-many without a separate relationship table.
  + Catering for different levels of completeness of data. This impacts the way errors are handled / rejected and the specification of failure.
  + Providing multiple versions of the truth (multiple Information Marts for the same data) to support different interpretations of data.
  + Handling changing business rules. To be able to always represent data in another way, this impacts the integration approach and the Information Mart concept.
  + Separate original and transformed data while still retaining the relationship between the two to support lineage and auditability, as well as refactoring of ETL and data models
  + Applying business rules / logic as late as possible: in the delivery of the information (Presentation Layer)
* **Real-time ready**. All concepts are designed for a possible future of handling real-time data sources
* **Modular approach**. The architecture and all related products can be used independently with relatively little customisation for different technologies. Every concept that can be modularised contains a separate paragraph explaining how this is achieved. Modules that can be used regardless of architecture include (but not limited to):
  + Metadata and process control, the complete metadata concept can be implemented regardless of the chosen architecture by creating the necessary metadata attributes and tables. This will require table and mapping customisation.
  + Error handling and recycling. Concepts such as the Error Bitmap and recycling can be added to any architecture (but will require table and mapping customisation).
  + Persistent Staging Area / Historical Area. An archive of transactions can be added to the Data Warehouse by creating a separate schema with the described process attributes (record validity attributes).
* **Corporate memory**. The Data Warehouse collects, integrates stores and manages all data, but does not ‘invent’ new data or addresses data quality issues directly. These should be handled by the operational systems, supported by exception reporting by the Data Warehouse. Data quality and interpretation of information is managed via Data Governance. This principle also means that no information is physically deleted from the Data Warehouse and logical deletes are supported at all times.

1. Architecture overview

This chapter describes the detail overview of the reference architecture.

* 1. High level architecture overview

The high level reference architecture is as follows:

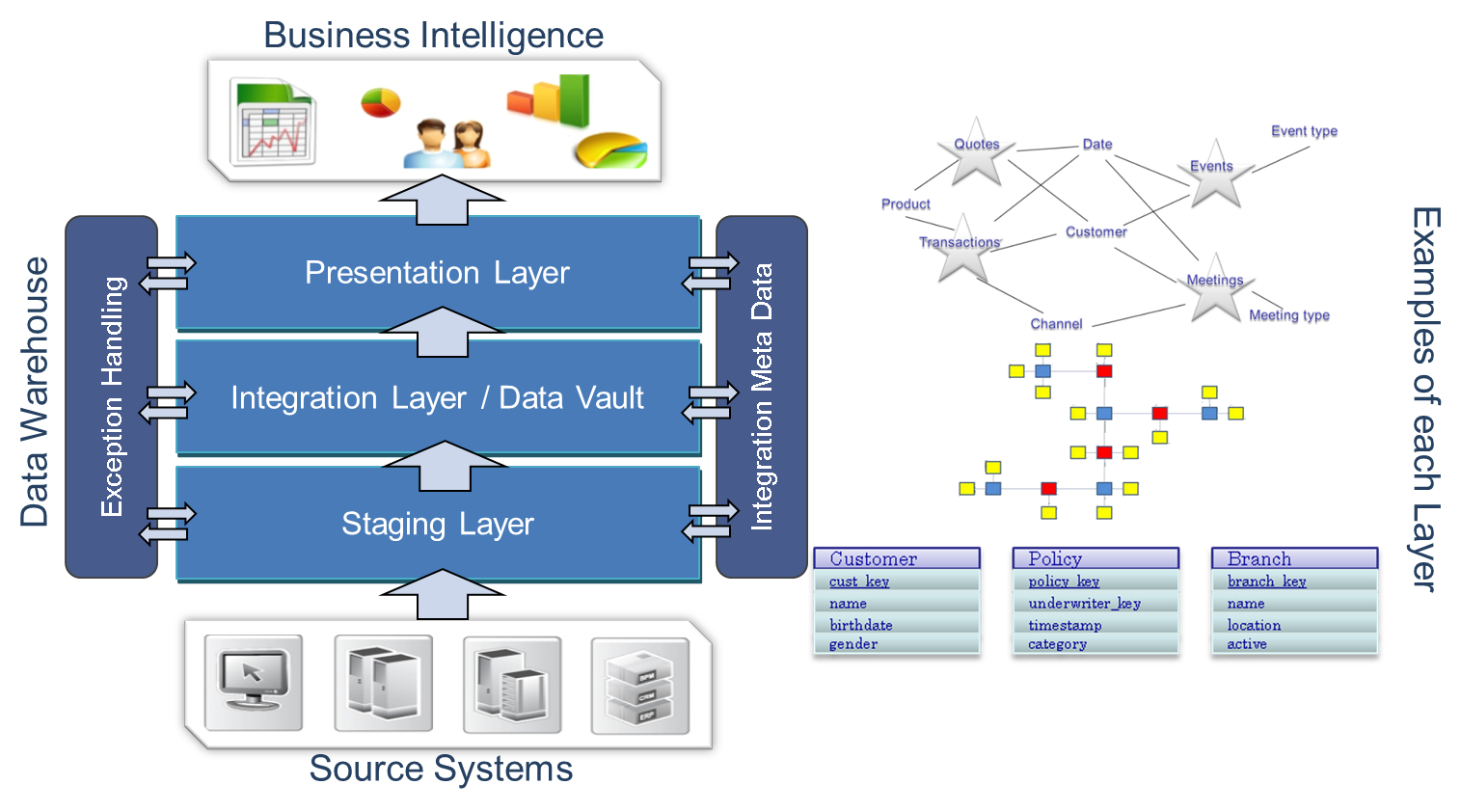


Figure 1: High level reference architecture

In this overview the basic flow of data as specified in the Reference Architecture is:

Data Warehouse Sources

1. Data is loaded to a **Staging Layer** (STG) with the purpose of bringing the information into the Data Warehouse environment. This layer focuses on correct interfacing of the data sources, managing deltas and archiving. The Staging Layer design and development can be executed without a core Data Warehouse model in place
2. The data is modelled for Data Warehouse purposes in the core **Integration Layer** (INT). This Layer focuses on mapping the data against the Data Warehouse model and correctly storing the history of the information
3. Data is converted to information in Information Marts as part of the **Presentation Layer** (PRES) design. In the Presentation Layer the information is optimised for use in the Business Intelligence or analytics software

Error handling and metadata / process control are applicable to every process in the architecture.

This high level approach is in line with virtually all Data Warehouse and Business Intelligence models. The difference lies in the exact location of the architecture where the specific concepts are enabled and the flexibility in applying these concepts.

The Data Warehouse Sources are operational systems that act as the key data suppliers to the Data Warehouse, and may also be consumers of the information. The integration between the Data Warehouse and operational systems can be direct database connection, flat files, xml, spread sheets or application messages.

The Data Warehouse may also receive and send data to external sources and other specialised systems like actuarial, analytics, budget and forecast, etc.

* 1. Detailed architecture overview

The following diagram is a bottom-up overview of the detailed steps taken and choices made. In this picture the architectural layers are divided into areas which contain specific steps for the Data Warehouse. This is a refinement of the high level architecture where optional choices can be made.

Each layer contains two (2) separate ‘areas’. These areas cover specific ETL functionality that support the overall purpose of the layer of which they are part of. Each area inherits the data modelling approach of the parent layer.

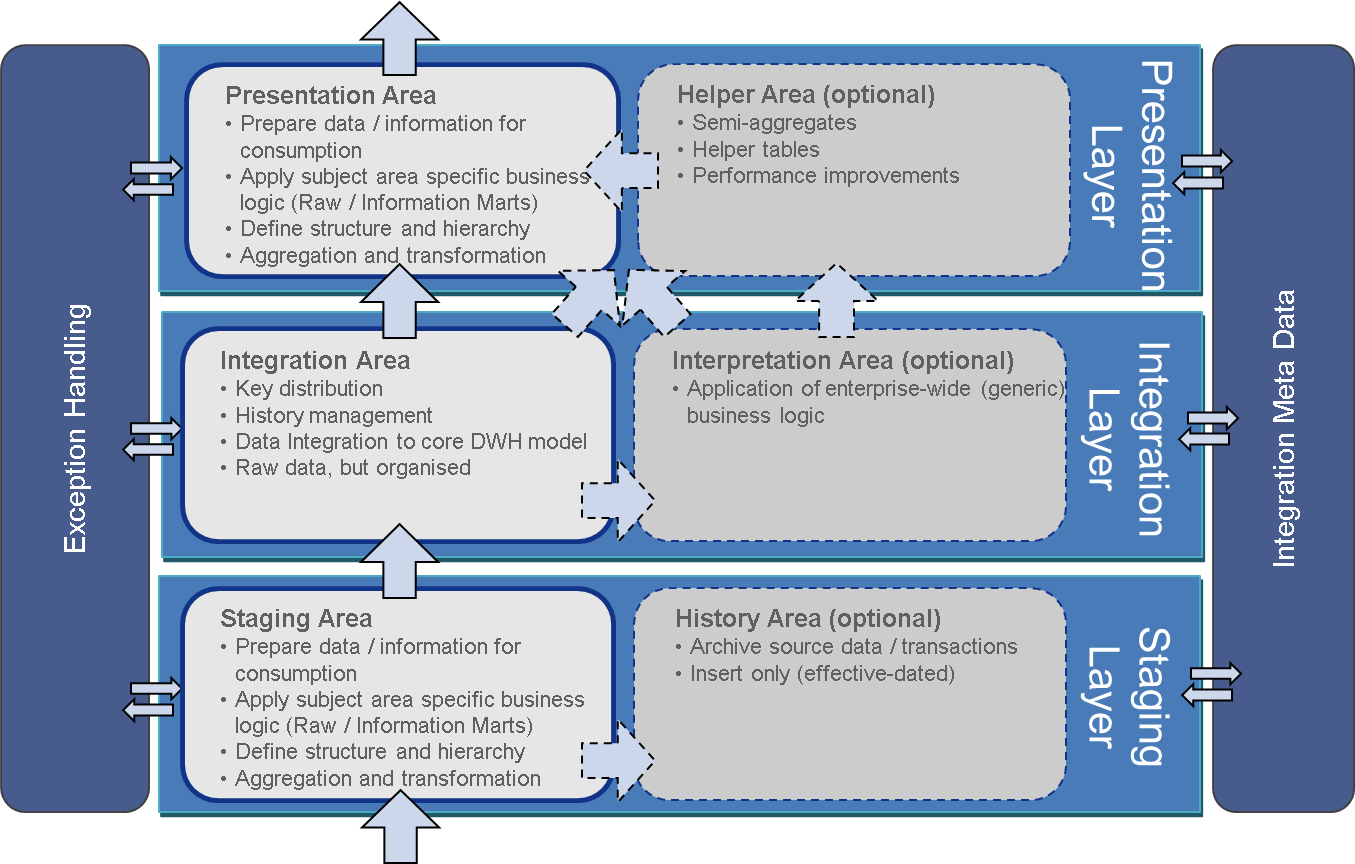


Figure 2: Layers and areas

In this diagram Error/Exception handling and Operational Metadata are positioned as applicable for every process in the architecture. The left column of areas in the diagram specifies the mandatory areas (Staging, Integration and Reporting Structure).

The right column of areas specifies the optional areas for the Data Warehouse, such as the History, Interpretation and Helper areas. Detailed description of the purpose of each Layer and Area is documented in the designated Layer document but an overview is given in the next sections.

The Areas that are part of the same Layer highlight that they share the same data modelling technique. For instance both the Staging Area and History Area inherit the source structure and are therefore modelled using the same technique.

Each layer is documented in detail in the subsequent reference architecture documentation.

* + 1. Staging Layer

The Staging Layer consists of the **Staging Area** and the **History Area**. The main purpose of this layer is to collect source data and optionally store it in a source data archive. The Staging Layer prepares and collects data for further process into the Integration Layer.

The Staging Area within the Staging Layer streamlines data types and loads source data into the Data Warehouse environment. This is done by utilising different Change Data Capture (CDC) techniques depending on the source system, files or options / restrictions of the available technology. Another important role for the Staging Area is the correct definition of time in the Data Warehouse. Depending on the type of source and interface dynamics extreme care has to be taken to ensure timelines are setup correctly for proper management of historical information in the subsequent steps.

The design is to load the source data delta into the History Area. Here the data is stored in the structure of the providing source but changes are tracked over time. The History Area is an important component in Data Recovery (DR) and re-initialisation of data (initial load) and is also used as part of the Full Outer Join comparison against the source systems.

An option in the Data Warehouse design is to load the source data into a History Area. Here the data is stored in the structure of the providing source but changes are tracked using the Slowly Changing Dimensions (SCD type 2) mechanism. The History Area is an important component in Disaster Recovery (DR) and re-initialisation of data (initial loads). When Change Data Capture, Change Tracking or messaging sources are part of the design the addition of a History Area is strongly recommended. A History Area can also be used for full outer join comparison against the source system and/or a full data dump interface.

Objects in the Staging Layer are not accessible for end-users or Business Intelligence and analytics software (e.g. Cognos). This is because for most scenarios information has not yet been prepared for consumption. There is an exception to this rule; for specific data mining or statistical analysis it is often preferable for analysts to access the raw / unprocessed data. This means this access can be granted for the Staging Layer which contains essentially raw time variant data. Allow access serves a purpose in prototyping and local self-service BI / visualisation.

* + 1. Integration Layer

The Integration Layer consists of the **Raw Data Vault** area and the **Business Data Vault** area. The main purpose of this Layer is to function as the ‘core Data Warehouse layer’ where all the data is collected in a normalised Data Warehouse model. To achieve optimal flexibility and error handling business rules are implemented as late as possible in the ETL process.

The Raw Data Vault stores the source data without changing the contents in the core Data Warehouse model. The system collects the data from all source systems in a generic way which is suitable for further expansion. The main Data Warehouse functionalities such as surrogate key distribution, storing history and maintaining relationships are done in this area.

The Business Data Vault uses the same modelling standards as the Raw Data Vault but provided interpretations or alternate views on the granular data. Both areas link closely to each other and in most cases provides separate cleaned or changed instances of tables that already exist in the Raw Data Vault.

The Business Data Vault is not a full copy of the Raw Data Vault. In most cases the Interpretation Area tables will refer to Integration Area surrogate key tables and provide an alternative perspective to Integration Area historical tables.

Examples of logic that can be applied in the Business Data Vault are generic business rules such as de-duplication or determining a single customer view. Additionally, the Business Data Vault is also used to design cross-references between similar datasets from different source systems. These cross-references are essentially recursive or intersection entities between business entities in the Raw Data Vault, but contain (business) rules to identify the main keys.

The important factor is that in this layer, business rules that alter the contents of the data are not yet applied. In the case of derivations, for example in the Business Data Vault, this means the original values will always need to stay available. Also, records are not checked for errors to keep the system as flexible as possible towards the Information Marts.

The Integration Layer (Integration Area and Interpretation Area) will be created using a **Data Vault 2.0** model which decouples key distribution using main entities (Hubs) but de-normalises reference information (Satellites) for these entities. Relationships between the main entities (Links) can be managed and tracked over time. This is a loosely-coupled data modelling approach which reduces dependencies and timing issues which are expected to occur in the data delivery.

As an example, this approach allows information related to the same customer or prospect to be delivered and integrated independently. It also supports ongoing linking of customer information to tie in various elements of information to the unique prospect or customer over time without losing flexibility; the logic for de-duplication can be changed and/or recalculated across historical information if required.

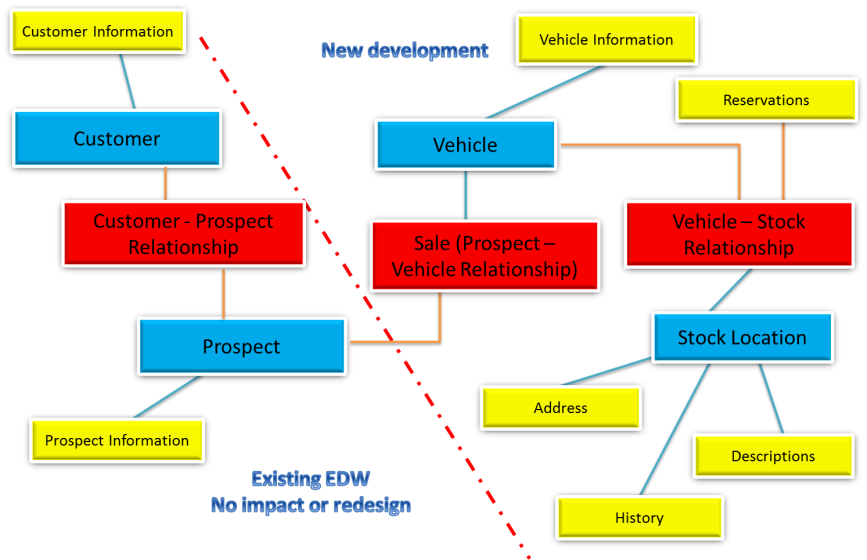


Figure 3: Combining Raw and Business Data Vault

Objects in the Integration Layer are not accessible for end-users or Business Intelligence and analytics software (e.g. Cognos). This is because for most scenarios information has not yet been prepared for consumption; only Data Warehouse logic is implemented. There is an exception to this rule; for specific data mining or statistical analysis it is often preferable for analysts to access the raw / unprocessed data. This means this access can be granted for the Integration Layer which contains essentially raw, but indexed and time variant data in the right context (e.g. related to the correct business keys). This is an ideal structure for statistical analysis.

* + 1. Presentation Layer

The Presentation Layer consists of the **Helper Area** and the **Reporting Structure Area**. This layer provides the data in a structure that is suitable for reporting and applies any specific business logic. By design information can be provided in any format and/or historical view since the presentation itself is decoupled from the core data store. Where the Integration Layer focuses on optimally storing anything that happens to the data (manage the data itself) and its relationships the Presentation Layer combines these relationships to form Facts and Dimensions. Since historical information is maintained in the previous layer these structures can be easily changed or re-deployed. Deriving dimensional models from a properly structured Integration Layer is very straightforward and development is made very easy because templates are provided and both facts and dimensions can be emptied (truncated) and reloaded at any point in time without losing information.

The Helper Area of the Presentation Layer is an optional area where semi-aggregates or useful tables can be stored to simplify or speed up processing. These types of tables are usually added for either performance reasons or the wish to implement the same business logic in as few places as possible. Helper tables can be modelled in any way as long as they benefit the Reporting Structure Area. They are not accessible by users or front-end reporting and analysis software.

By thoughtfully creating aggregate tables which can be shared by the Information Mart one could for instance create a fact table on a certain aggregate level and have different Information Marts aggregate this table further depending on their needs. This way the business logic and performance demanding calculations only have to be done once.

The Reporting Structure Area is the final part of the reference architecture. An Information Mart is modelled for a specific purpose, audience and technical requirement. The complete Data Warehouse can contain very different Information Marts with different models and different ‘versions of the truth’ depending on the business needs.

In the process from loading the data from the Integration Layer to the Presentation Layer most of the business logic is implemented.

* 1. Context of the Reference Architecture

The previous diagrams focused primarily on the place ETL has in the Data Warehouse, but in the broader architecture other factors are in play as well. Not all components are applicable in every scenario but the following diagram shows their place in the overall architecture in case they are required.

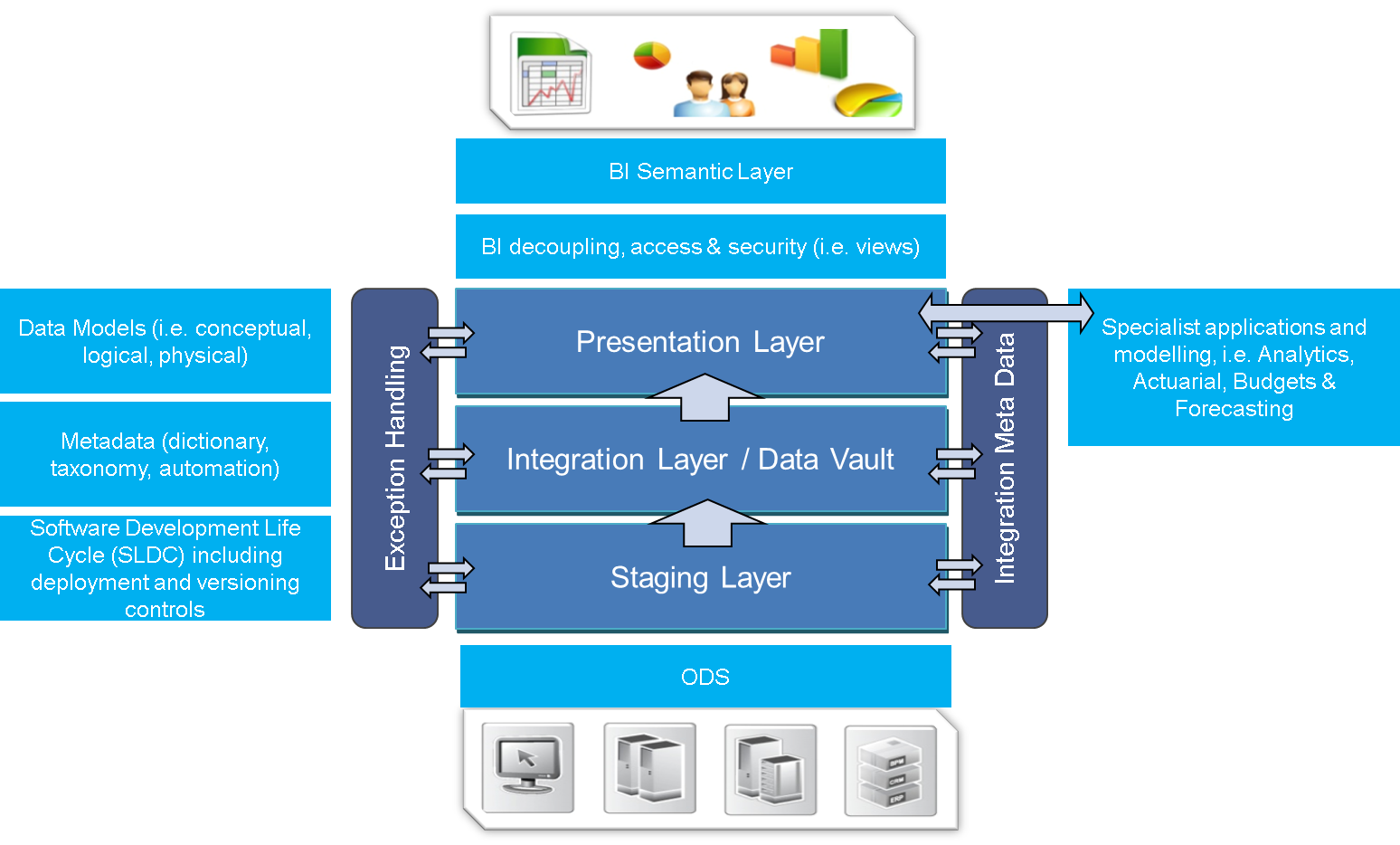
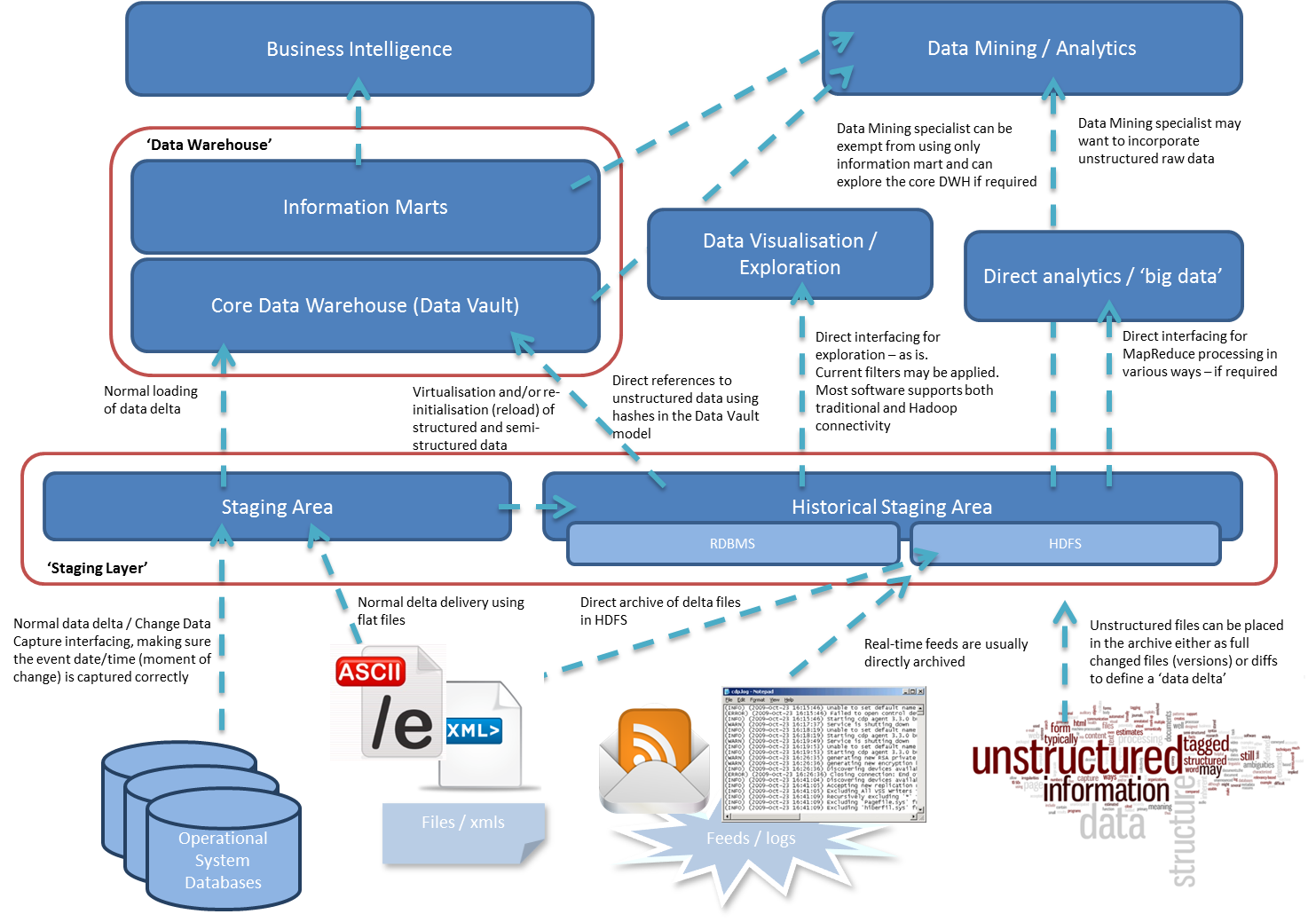


Figure 4: Overall Solution Design

This diagram shows the non-ETL considerations that form part of the Reference Architecture.

* BI Semantic Layer. This is essentially a business friendly view of the underlying physical database. A Semantic Layer prepares the joins (relationships) between database tables and defines for instance which measures can be summarised and which ones would result in double counting. Another purpose is to rename technical database attribute names to a name more suitable for reporting. A Semantic Layer aims to support the data exploration by defining these central requirements once so they do not have to become part of each query
* In most software platforms, the definition of a Semantic Layer adds intelligence and awareness of neighbouring data entities to assist users in the creation of the reports
* BI Views. The reference architecture incorporates views only ‘on top off’’ the Presentation Layer (Information Marts) to act as ‘decoupling’ mechanism between the physical table structure and the Business Objects semantic layer (business model). This is explained in more detail in the Modelling section
* Specialist Applications. In some cases applications are defined that ‘live’ in the Presentation Layer and/or the (typically) OLAP applications that are updated through the Presentation Layer. This occurs often in Finance related scenarios where OLAP is used as Forecasting application and various Forecast scenarios are saved back into the OLAP cube or Presentation Layer structure
* Dependent / Independent Information Marts. In some cases not all or even none of the data is provided by the Data Warehouse. This occurs typically in prototype scenarios or for purposes which have limited requirements including a short term use. While the directive is to integrate all data into the Data Warehouse it is acknowledged that in some cases Information Marts exist in the Presentation Layer that are sourced directly from operational systems. Similarly, dependent Information Marts are always updated via the Data Warehouse (Integration Layer)
* ODS. Provisions may be taken to introduce an Operational Data Store (ODS) which is an integrated repository that enables operational use and maintenance of data. This may be applicable to one or more source systems, or for information that does not have an authoritative source. It is important to note that not all information needs to pass through the ODS before entering the Data Warehouse. The ODS is a fit-for-purpose solution that provides a new operational use for a subset of information that may be integrated from various sources that fit the requirement. The remaining data can be sourced in the conventional way
* Data Models / Data Modelling. The design supports different approaches for modelling of information both in the context of Information Modelling (Logical Models and Physical Models) of the information for an organisation and for specific techniques in Data Warehouse modelling. Ultimately, data models, as well as a selected technique, are a core requirement for developing under the Reference Architecture
* Metadata. Information about the data is a key design input and describes how, when and by whom the data is collected, formatted and used. Metadata is essential for understanding information stored in the data layer. Metadata is vital to the understanding the impact that results when data or its meanings is altered
* SDLC. The Data Warehouse is essentially a custom developed application and is subject to the same Software Development Life Cycle (SDLC) processes as any other application. This also defines the involved DTAP environments (Development, Test, Acceptance, and Production) and processes around the use of these environments.
  1. Expanding for unstructured data

The following diagram expands on the broader architecture considerations, especially how the Staging Layer can be used to incorporate unstructured data that is not suitable to be directly loaded into a (relational) database.



The key message is that not all data needs to be routed through a Data Warehouse, in some cases it may not be worth doing so. However, this approach maintains the option to integrate data at a later stage and preserving the audit trail.

1. Data Modelling
   1. Data Warehouse modelling and design

The following principles apply:

* The Staging Layer is always in the same structure as the providing operational system, but all attributes are nullable to avoid load errors
* The Integration Layer can be modelled using a hybrid (Data Vault, Anchor Modelling) technique. For the Enterprise Data Warehouse, which integrates many sources and is subject to change a Data Vault 2.0 approach is adopted
* Information Marts are created as full Type 2 by default, but can be mixed with Type 1 attributes as well
* For performance reasons, specific Type 1 dimensions can be defined or in advanced systems hybrid solutions where the temporality is defined on an attribute level
* The Integration Layer, as the core Data Warehouse Layer, is the default data platform. Information Marts are essentially redundant as all information is maintained in the Integration Layer and may even be virtualised
* Data is always maintained at the most granular level in the Integration Layer. It can be aggregated in the Information Marts for performance and usability reasons
  1. Modelling Reasoning

In traditional Data Warehouse modelling there are currently three main modelling approaches available and in wide use today:

* 3NF (3rd normal Form) (Inmon)
* Data Vault (Linstedt)
* Dimensional Modelling (Kimball)

The recommended modelling technique for the Integration Layer is Data Vault, with the complete architecture supporting Data Vault 2.0 concepts. Data Vault Modelling is a database modelling method that is designed to provide long-term historical storage of data coming in from multiple operational systems. It is also a method of looking at historical data that, apart from the modelling aspect, deals with issues such as auditing, tracing of data, loading speed and resilience to change.

Data Vault Modelling focuses on several things. First, it emphasizes the need to trace where all the data in the database came from. This means that every row in a Data Vault must be accompanied by record source and load date attributes, enabling an auditor to trace values back to the source.

Second, it makes no distinction between good and bad data ("bad" meaning not conforming to business rules). This is summarized in the statement that a Data Vault stores "a single version of the facts" (also expressed by Dan Linstedt as "all the data, all of the time") as opposed to the practice in other data warehouse methods of storing "a single version of the truth" where data that does not conform to the definitions is removed or "cleansed".

Third, the modelling method is designed to be resilient to change in the business environment where the data being stored is coming from, by explicitly separating structural information from descriptive attributes. Finally, Data Vault is designed to enable parallel loading as much as possible, so that very large implementations can scale out without the need for major redesign.

This recommendation is based on the existing situation at AGA:

* There is a need to integrate disparate systems
* The business and its requirements change over time
* The storage of information needs to happen over time
* The data quality of the existing source systems is evolving
* The requirements and expected usage of the information is fluid
  1. Use of Views
     1. Decoupling views

The Data Warehouse design incorporates views ‘on top off’’ the Presentation Layer (Information Mart). This is applied for the following reasons:

* Views allow a more flexible implementation of data access security (in addition to the security applied in the BI Layer)
* Views act as ‘decoupling’ mechanism between the physical table structure and the Semantic Layer (business model)
* Views allow for flexible changing of information delivery (historical views)

These views are meant to be 1-to-1, meaning that they represent the physical table structure of the Information Mart. However, during development and upgrades these views can be altered to temporarily reduce the impact of changes in the table structure from the perspective of the BI platform. This way changes in the Information Mart can be made without the necessity to immediately change the Semantic Layer and/or reports. In this approach normal reporting can continue and the switch to the new structure can be done at a convenient moment.

This is always meant as a temporary solution to mitigate the impact of these changes and the end state after the change should always include the return to the 1-to-1 relationship with the physical table.

A very specific use which includes the only allowed type of functionality to be implemented in the views is the way they deliver the historical information. Initially these views will be restricted to Type 1 information by adding the restriction of showing only the most recent state of the information (where the Expiry Date/Time = ‘9999-12-31’). Over time however it will be possible to change these views to provide historical information if required. On a full Type2 Information Mart, views can be used to deliver any type of history without changing the underlying data or applying business logic.

* + 1. Views for virtualisation

Another use case for view is for virtualising the Presentation Layer. As all granular and historic information is stored in the Integration Layer it is possible, if the hardware allows it, to use views to present information in any specific format. This removes the need for ETL – physically moving data – from the solution design. Applicability of virtualisation depends largely on the way the information is accessed and the infrastructure that is in place. Possible application includes when the BI platform uses the information to create cubes, when information is infrequently accessed or with a smaller user base.

* 1. Referential Integrity and constraints

The fundamental approach of the Data Warehouse modelling is to enforce Referential Integrity (RI) on database level. This is a Data Warehouse best practice that allows the database to efficiently manage the consistency of the solution. Exceptions can be made where RI is temporarily disabled when certain ETLs can be run in different order and/or parallel (especially in the case of a 3NF Integration Layer) but RI must be enabled after the processing to ensure integrity. Only for very large datasets (>250TB), or for very light hardware RI is disabled altogether. For this purpose all ETL designs must take RI into account using placeholders and key distribution.

The Data Vault 2.0 approach still requires enforcing of RI, however due to options of parallel loading the integrity cannot be implemented in ETL. For this purpose a Batch level verification process needs to be executed, to make sure the RI is in order after ETL processing.

The Data Warehouse design implements various levels of predefined constraints and placeholder mechanisms to support this principle. The Operational Meta Data repository, as a component managed by the Data Warehouse team, is exempt from these conventions and is allowed a greater freedom in implementation options (data types, keys, constraints).

Every Data Warehouse table contains a predefined set of metadata attributes, which are – with the exception of the Update process attributes – always set to NOT NULL.

| **Layer / area** | **Constraint / concept** |
| --- | --- |
| Staging Area (STG) | All source attributes are nullable (NULL). |
| History Area (HSTG) | All source attributes are nullable. HSTG tables have a meaningless key as Primary Key and a unique constraint on the combination of the source key and the event date/time. This means only one value can be valid at a point in time. The source to staging interface design ensures that no duplicates can ever occur by the (correct) assignment of this event date/time. |
| Integration Layer | Data Warehouse key tables will always have a -1 placeholder value to server as the ‘unknown’ record.  Data Warehouse history tables will always have a complete time interval. This means there is never a ‘gap’ or ‘island’ in the time intervals and inner joins can always be used. This is implemented by insert a starting record every time a new DWH key is created.  All record sets that are loaded to the Integration Layer support their own ‘keying’ processes to reduce dependencies, but also to ensure the referential integrity requirements are always met. This also means that the system will always provide a correct view of the data when it was processed, and how it improves over time. |
| Presentation Layer | Every Dimension will contain a -1 dummy record to link orphan records to. This means all Fact records will have an inner join link to the Dimensions.  Additionally, if transactions refer to business entities (Data Warehouse keys) that have no match to other reference data when joining the various entities into Dimensions (through the intersection / ‘link’ entities in the Integration Layer) the upper levels are set to ‘Unknown’. No loss of data is ensured in this process because the standard use of outer joins when implementing business logic in Dimensions. No NULL values are allowed in the Dimensions. |

1. Error and Exception handling

Error handling and exception is applicable to every layer and area in the architecture. Each individual layer definition document will describe how error handling is used for that particular section of the architecture since the exception handling is very different between layers. The error handling and recycling document itself lists and explains the concepts that can be used and will provide an overview of the complete error and exception handling solution.

Figure 5: Exception handling options

The details for error and exception handling are defined in the ETL Framework - 7 - Exception Handling v1.0’ document.

1. ETL control process model

Similar to exception and error handling concepts the Metadata Model links in with every process, layer or area in the architecture. Each individual layer definition document will describe how the metadata model is used for that particular section of the architecture.

The Metadata Model document itself will list and explain the available concepts and will provide an overview of the complete framework.



Figure 6: Metadata Logical Model

In general the metadata process model supports the ability to trace back what data has been loaded, when and in what way for every interface. A single attribute in any place in the architecture should be auditable back to the originating source system.

This means that the following information must be available (at date/time level):

* When a record was inserted
* When a record was update.
* What the source system was where the record originated from
* When the event took place that changed the source data
* Which interface has loaded the data (the Module)
* Which workflow has loaded the data (the Batch)
* On which platform the ETL took place (also true for source data)
* When data was most recently offered to the model (integration layer specific)

For tables that store history the metadata includes the regular start and end date/time information as well as flags whether the record is the most actual one. The metadata model will largely be based on the Operational Metadata (OMD) Framework, but it will be updated and tailored to suite the reference architecture.

Appendix A: Glossary

| **Terminology** | **Definition** |
| --- | --- |
| Business Key | See Source Key |
| Business Rules | A set of rules or logic defined by the Business units of an organization and used for data validation or to derive a specific data outcome |
| Change Data Capture | In databases, change data capture (CDC) is a set of software design patterns used to determine (and track) the data that has changed so that action can be taken using the changed data. |
| Cleansing Area | An area where data that originates from integration area get standardized, validated, cleansed/scrubbed, de-duplicated and/or enriched |
| Data de-duplication | A term used to identify data that are captured, processed, or stored redundantly. It results in unknown, uncontrolled, and unmanaged data redundancy. It is not orderly and creates additional disparate data. |
| Data enrichment | The process of supplementing data with additional information, typically obtained from external sources |
| Data scrubbing | The process of filtering, merging, decoding, and translating source data to create validated data for the data warehouse. |
| Data standardization | The process of applying a defined set of standards, usually formatting standards, to the data |
| Data validation | The process of checking if that data conforms to a defined set of rules or logic |
| Information Mart / Data Mart | See Presentation Area.   * + - Dependent Information Mart; a subset of the Data Warehouse (sourced from the DWH). A dependent Information Mart may consist of both tables and materialized queries.     - Independent Information Mart. Contains data directly sourced from operational systems.     - Raw Data Mart; delivery of unaltered (raw) information. No business logic has been applied |
| Data Warehouse | A centralised storage of enterprise-wide data that becomes the single source of truth for an organization. An integrated database designed towards the enterprise business intelligence requirements. The data is integrated from several optional systems, external sources and specialised applications like budget, or actuarial. |
| History Area | An area where data, that originates from the staging area, is archived permanently |
| Integration Area | An area where data that originates from the staging area is re-modeled and stored. Data content does not change and remain the same. |
| Integration Layer | Second layer of the architecture where data obtained from the Staging layer is re-modeled and is either stored in its untreated form (Integration Area) or treated form (Cleansing Area). |
| Logical Delete | Where a flag in a record is set to represent that it has been deleted. |
| Metadata Control | See Operational Metadata |
| Metadata Model | Refers to Operational Metadata Model. A set of data entities that contain all information relating to the load process. |
| Operational Data Store (ODS) | A subject oriented, integrated, volatile, current valued data store containing only corporate detailed data |
| Operational Metadata (OMD) | Information relating to the data load process |
| Presentation Area | An area where data that originates from the integration area or cleansing area is re-modeled into a star-schema and/or snowflake model and further transformed to suit the needs of a specific group of Business units (typically at the departmental-level) |
| Presentation Layer | Third and final layer of the architecture where data is available for Business units to query |
| Process Control | See Operational Metadata |
| Source (System) Key | Key found in a source system to uniquely identify certain things such as product or customer |
| Source System | An operational system of record whose function it is to capture the transactions of the business. Often called a "legacy system" in a mainframe environment. |
| Staging Area | An area where data supplied by a source system or extracted from a source system is stored and its data type validated |
| Staging Layer | First layer of the architecture where data supplied by a source system or extracted from a source system is temporarily held (Staging Area) before being copied into the integration area and/or History Area. |
| Surrogate Key | An integer value that gets assigned to each record in, typically, a Dimension table. The value itself has no meaning and does not represent anything about the underlying data. But it is associated with a source key or a composite of attributes such as source system key, natural key and record effective start date. |

Appendix B: Guidelines

This appendix contains guidelines for various aspects of the reference architecture, most notably ETL are specified in great detail in the Design and Solution Patterns. The following provides a high level overview of the guidelines and principles of projects that are designed and developed under the architecture of the ETL Framework.

## Extraction from internal systems

* Data must be extracted from the sources that created the data (as opposed to using copied data as a source). This is a broader Data Governance principle
* Impacts on operational systems must be assessed and documented as part of the interface specification
* Source system extract processes should not include the Data Warehouse transformation, aggregation and consolidation rules, this is applied later (separation of concerns)
* Source system extracts should include control data to enable audit and reconciliation, e.g. record count, hash totals, etc. This is by default supported by the ETL process control model
* The standard data integration tool must be used to extract data from the source systems, unless another efficient data extract utility is provided as part of the application package (this may include using SQL for ETL)
* Implement incremental extracts where possible, as this is more scalable.

## Integration to the Data Warehouse

* Reconciliation and auditing must be built in all processes. By default this is managed by the ETL process control model.
* Change history to hierarchy, e.g. product hierarchy, must be captured (also known as slowly changing dimension)
* Optimise throughput without compromising data integrity
* Transformation process must include re-start and exception handling with no or minimum manual intervention. Do not rely on the database backup and recovery for process restart and retry. By default this is built in all ETL Framework templates.
* All exceptions and performance statistics must be saved in the database to enable reporting. By default this is handled by the ETL process control model.
* Do not over complicate processes more than necessary, e.g. cover scenarios that are unpractical or impossible, defects in preceding components, all possible future changes, etc.
* Keep processes modular to gain leverage over failure and reusability.
* Performance benchmarks must be done as part of the development
* Leverage the RDBMS for data integrity enforcement (e.g. referential integrity, check constraints, validation, etc.). Be aware of the performance trade-offs and always ensure that RI is handled and verified by ETL as well
* Avoid using embedded freehand SQLs in ETL tools
* Do not ‘over optimise’! Sometimes ‘good enough’ performance is acceptable. Be aware of the cost benefit ratio. Balancing performance against simplicity should by default fall in favour of simplicity
* Do not include functions that are not required, even if it ‘does not take long’. Be aware of additional costs in testing and maintenance
* Take an incremental development approach
* Development should allow more data and new types of data to be added without rework
* Ensure components are ‘built once reuse many times’ without creating additional dependencies unnecessarily
* Do not delete data from the data warehouse except by archive processes
* Design transformation and load processes so that processing schedule can change without any rework. E.g. the same processes can run daily, weekly or monthly, etc.
* Transaction data should be incrementally added to the Data Warehouse, even if the source system cannot provide incremental extracts
* Transaction data must not be physically deleted. Instead, flag the transaction as ‘delete’
* Transaction data should not be updated. E.g. should the data change, flag the old transaction as ‘deleted’ and create a new record
* All data in the Data Warehouse must be reconcilable to the sources

## Extracting data from the Data Warehouse

* Data can be extracted from the Data Warehouse only; do not extract data from the Information Marts to be incorporated into a new Information Mart to attempt the reconciliation of logic
* Design the extract process so it may be re-used for multiple consumers
* Include control to enable audit and reconciliation