EUROPEAN CENTRAL BANK

EUROSYSTEM

Edition 2.1

July 2026

SMCube Metadata Model

Contents

0. Document Control and Model Versioning

Manual Edition	Date	Description	Model Version	Manual Status
1.0.0	2018-03	Initial version	SMCube 1.0	Archived
1.1.0	2024-03	Updated for restructuring and improvements	SMCube 1.0	Archived
2.0.0	2025-09	Updated according to SMCube 2.0	SMCube 2.0	Archived
2.1.0	2026-07	Updated and reviewed	SMCube 2.0	Shared

1. Introduction

The **Single Multidimensional Metadata Model (SMCube)** provides the common metadata foundation for the **Single Data Dictionary (SDD)** and the **Banks' Integrated Reporting Dictionary (BIRD)**. It defines how metadata is structured, historised, governed, and delivered to support both regulatory and statistical frameworks.

SMCube is a **Metadata Implementation Model** that specifies how concepts such as V**ariables,** M**embers,** C**ubes,** h**ierarchies,** and t**ransformations a**re represented within the metadata system. It establishes a unified approach to modelling structural and descriptive metadata, enabling consistent application across multiple frameworks (e.g. AnaCredit, BIRD, Financial Reporting (FINREP), Balance Sheet Information (BSI), Common Reporting (COREP), or Securities Holding Statistics (SHS)), while preserving their specific business content. The SMCube model is implemented in the BIRD dictionary.

SMCube 2.0 has been further developed to address BIRD-related use cases[1].

A fundamental design principle of SMCube is the **clear separation between logical modelling and rendering / technical structures**. This separation allows business concepts, transformations, and validation logic to be modelled independently of how information is ultimately presented or exchanged.

[1] This development benefited from close collaboration with the BIRD Operational Tasks project team.

As a result, logical metadata can be reused across multiple reporting artefacts and physical representations, significantly improving consistency, maintainability, and the ability to support complex logical transformations.

The target audience includes **Data modellers**, **Metadata managers**, **IT architects**, **Integration engineers**, and **Business analysts** (including BIRD users) who rely on metadata to support regulatory and statistical processes.

1.1 SMCube Metadata Model: purpose, requirements and use cases

The SMCube metadata model provides a unified, metadata-driven framework to support automation, governance, and interoperability across different modelling standards.

Purpose and Core Principles

- **Metadata-driven design:** use metadata as a configuration layer to reduce coding, improve automation, and support both IT systems and end users
- **Cross-standard integration:** introduce an abstraction layer above modelling methodologies (e.g., DPM/XBRL, SDMX, ER models) to align and jointly use datasets
- **Governance and traceability:** embed historicisation and maintenance agency attribution to track changes over time
- **Consistency and clarity:** standardise naming conventions, remove redundancies, and define clear relationships between metadata objects

Functional Capabilities

- **Complex mappings:** connect related information across frameworks and integrate existing dictionaries
- **ER model compatibility:** support representation of logical data structures using Entity Relationship principles
- **Extensibility:** adapt to evolving regulatory and statistical requirements and allow reuse by ECB and external organisations

Main Use Cases

- **Regulatory and statistical frameworks:** manage metadata for datasets such as AnaCredit, BSI, FINREP, COREP, including template rendering
- **Metadata governance:**
    - support generic and cross-domain datasets

    - enable harmonised mappings across frameworks
- **Interoperability and integration:** allow IT systems to use metadata as a consistent integration layer across standards
- **Auditability and traceability:** ensure all metadata changes are historised, transparent, and attributable

Overall Value

- Enable scalable, metadata-driven data ecosystems
- Improve integration, transparency, and reuse of data across domains

1.2 Purpose of this Document

This document presents the **SMCube metadata model** as the reference high-level specification of the metadata structures underpinning the **Single Data Dictionary (SDD)** and the **Banks' Integrated Reporting Dictionary (BIRD)**. It provides a comprehensive overview of the concepts, structures, and metadata rules that together form the foundation of metadata-driven regulatory and statistical reporting.

The document is organised into the following chapters:

- **Core package**: introduces the dictionary's fundamental objects such as **Variables**, **Domains**, and **Members**.
- **Data definition**: explains how core concepts are organised into complete data models through **Frameworks**, **Cubes**, **Cube Structure Items**, and their **Relationships**.
- **Transformations & Validations**: describes how data models can be derived or checked against each other (e.g. transforming **BIRD Input Models** into **AnaCredit collection data model**, or applying consistency rules).
- **Other Metadata**: covers additional packages such as:
    - **Rendering:** (e.g. expressing a FINREP 05.01 business template layout),
    - **Mapping:** only in the SDD context (e.g. aligning ITS concepts with BIRD concepts),
    - **Legal and Classification references** (e.g. linking a Variable or Member to a regulation or supervisory category).

2. Definition of a data set in SMCube methodology

The conceptual foundations of the SMCube methodology are defined as those most basic and reusable metadata objects that constitute the semantic layer upon which all other packages and

structures of the information model are based. These are cross-cutting concepts, not tied to a specific framework, but applicable in multiple contexts. They include:

- the definition of the **Maintenance Agency**, which is defined in every table of the Information Model.
- the principles of **historicisation**, which uniformly govern the temporal evolution of all metadata.
- the identification and description of business objects

2.1 Maintenance agency

The **Maintenance agency** is the institution or organisational unit responsible for defining and updating metadata. It ensures that metadata is managed, traceable, and aligned with institutional mandates.

All metadata created in accordance with the SMCube methodology must be assigned to the relevant maintenance agency identified by the unique attribute MAINTENANCE_AGENCY_ID. For example, BIRD is the maintenance agency for metadata defined by the BIRD Experts Group.

2.2 Historicisation Principles

Historical traceability is a fundamental principle of SMCube. All metadata objects are chronologically bound and contain explicit information regarding their operational validity:

- **Valid from (VALID_FROM)**: the date from which the object is valid; this forms part of each object's primary key.
- **Valid to (VALID_TO):** the date until which the object is valid[2];

Versioning ensures that metadata can be reconstructed exactly as it was at a given point in time, supports version comparison within the Framework, and creates an audit trail of all changes. It is not possible for two distinct versions of any metadata to overlap in terms of validity. This constraint ensures the chronological consistency of all metadata.

2.2.1 Framework version historicisation

Frameworks are historised through the **Framework Version** table of the information model, which contains the following attributes:

- **Maintenance Agency ID:** the responsible institution.
- **Framework_ID**: a unique identifier of the Framework.

[2] The default value 31/12/9999 is assigned to currently valid business objects.

- **Version ID:** a unique identifier of the Framework version.
- **Is Draft:** indicator of draft versus production status.
- **Valid From / Valid To:** the validity timeframe.

This provides a reliable basis for managing the lifecycle of entire Frameworks and linking them to their associated Cubes, Variables, and Transformations.

2.3 Identification and Description of Business Objects

Most SMCube metadata share a common set of descriptive attributes and identifiers. These attributes ensure that metadata remains **human-readable, machine-processable** and suitable for usability, governance, system integration and referential integrity.

Common attributes:

CODE:

A technical code that uniquely identifies the metadata object within its scope. For some objects (e.g. **Variables**, **Members**) the CODE is expected to be directly implemented in the dataset's technical system. For other objects (e.g. **Cubes**) it is strongly recommended to use the CODE consistently, so that users experience a uniform modelling approach across frameworks. Codes may be abbreviated or technical for efficiency or organisational reasons. For some type of metadata (e.g. **Subdomain**) such codes are not directly representing something that can be translated into a technical system, but they are used for reference.

NAME:

A human-readable, business-friendly label for the object. While the CODE may be cryptic, the NAME should communicate the business meaning clearly[3].

DESCRIPTION: A detailed explanation of the concept, clarifying its business meaning and scope. Descriptions should be as complete as possible, especially for regulatory or statistical concepts.

For certain types of metadata (e.g. **R**outines) the description provides a semantic description of the algorithm.

<Object type>_ID:

A unique (business) identifier for the object.

[3] Example: CODE = CRRYNG_AMT, NAME = Carrying Amount.

Note that, normally, object business IDs do not include the **Valid From** information, that is considered a separate component of the primary key.

Rationale:

This layered identification (CODE, NAME, DESCRIPTION, ID) ensures that

- Technical users have stable and consistent identifiers.
- Business users can interpret metadata meaningfully.
- Metadata remains portable across frameworks and systems.

3. The Core Package

The **Core Package** defines the fundamental, reusable dictionary objects in SMCube. These objects are dataset-agnostic and are referenced by the other packages (Data Definition, Transformation, Rendering, and Mappings). The Core Package provides a common semantic foundation across Frameworks and use cases, ensuring that shared meanings are modelled consistently and can be reused throughout the information model.

Principles:

- **Semantic integration and consistency:** Each business concept (e.g. a Variable or a Member) is defined once with a clear and unambiguous semantic meaning and is consistently referenced across contexts.
- **Reusability**: Core objects are designed for reuse across Cubes and Frameworks to minimise duplication, prevent semantic drift, and support long-term maintainability.
- **Governance & Historicisation**: every Core object is subject to chronological management (VALID_FROM, VALID_TO) and is associated to its **Maintenance Agency**.
- **Separation of concerns**: Domains describe value spaces; Variables attach business meaning to those spaces; Subdomains restrict them in contextual scope.
- **Explicit cardinalities**: relationships between Core objects are formally specified and constrained (see **Cardinalities & Integrity**).

3.1 Domain

A **Domain** defines the set of permissible values associated with a given (logical) data type (e.g. Country, Currency, Date, String, Monetary Amount). It provides the value space from which a Variable can take its values. Domains can be classified as:

- **Enumerated**: a finite set of allowed **Members** (e.g. ISO currency codes, country codes).
- **Non-Enumerated**: value spaces can be governed by **Facets** (e.g. a String pattern, a numeric interval, a date range).

Characteristics:

- Domains can be reused.
- Domains are containers for all possible values of a Variable for all usage.

Supported data types:

- NUMERIC

- INTEGER
- DECIMAL
- TEMPORAL
- DATE
- TIMESTAMP
- STRING
- BOOLEAN

3.2 Member

A **Member** is an allowed value within an Enumerated Domain[4]. Members are unique within their Domain and may be grouped into **Subdomains** and organised into **Member Hierarchies**.

Characteristics:

- Members do not intrinsically encode aggregation semantics; such semantics are introduced, specified, and managed exclusively via **Member Hierarchies**.
- Members should follow recognised coding schemes (e.g., ISO standards) whenever applicable. Any deviations from these standards should be documented.
- **Expected presence in data**: The CODE of a Member is the value expected in physical datasets[5].

3.3 Subdomain and Subdomain enumeration

A **Subdomain** is a reusable restriction of a Domain tailored to a context[6] (e.g. Framework-wide, cube-specific or purpose-specific).

Membership & ordering (Enumerated Domains)

Subdomains enumerate a subset of Members via Subdomain Enumeration.

Faceted restrictions (Non-Enumerated Domains)

Subdomains can bind Facets (e.g. MIN, MAX, min value, max value, PATTERN) that further restrict the value space (see next sections)

---

[4] Example: DOMAIN = Currency; MEMBERS = {Euro (EUR); United States Dollars (USD); Japanese Yen (JPY); …}.

[5] Example: If the Currency Domain includes the Member Euro with CODE = “EUR”, then in the data, the Currency field must use the value "EUR" for Euro-denominated records.

[6] Examples: A) Currency Domain → EU Currencies Subdomain. B) String Domain → Strings with Characters up to 20 Subdomain (length <= 20).

Characteristics:

- A Subdomain **inherits the behaviour** of its parent Domain:
    - If the Domain is Enumerated, the Subdomain is also Enumerated.
    - If the Domain is Non-Enumerated, the Subdomain applies additional Facets.
- A Subdomain can be created and explicitly bound to a **specific Cube**, ensuring it can only be used in that Cube's context (e.g. restricting Currency choices in a reporting template).
- For Subdomains based on **numeric Domains**, the Subdomain can further specify the **semantic meaning** of the values. Typical examples include:
    - Integer
    - Decimal
    - Percentage
    - Monetary
        - Monetary (decimal)
        - Monetary (integer)
    - Rate
    - Index
    - Quantity
    - Ratio
    - Score
        - Score (decimal)
        - Score (integer)

3.4 Variable

A Variable is a reusable business concept, defined independently of any specific data structure, which can assume allowed values belonging to a specific Domain. Variables are defined and detailed in the core package of the information model and are applied in data definitions to provide semantic meaning for Cube attributes.

Main characteristics:

Independence and reuse:

- Variables are defined outside of Cubes and are not tied to any specific Cube implementation.
- The same Variable can be reused across multiple Cubes.
- A Variable is associated with a Cube only through Cube Structure Items, which establish its contextual role within a specific Cube.

Domain, data type, and constraints:

- Each Variable is bound to a Domain, which defines its data type and general constraints.
- When a Variable is used within a Cube, its Domain may be restricted to a Subdomain, if required by the Cube's specific context.

- The effective data type and constraints of a Variable in a Cube are therefore derived from its Domain and, where applicable, from the selected Subdomain.

Codes and identification in datasets:

Each Variable is identified by a CODE, which normally/ideally corresponds to the expected technical representation of the corresponding data element in datasets. When a Variable is used in a Cube, datasets implementing that Cube must include a field or column with the same CODE[7].

3.5 Variable Set & Variable Set Enumeration

A Variable Set is an explicitly enumerated collection of Variables that may be referenced as values by another Variable. This modelling approach, known as *verticalisation*, consolidates multiple measures or attributes into a single dimension, with the set membership governed by a Variable Set Enumeration. The specific Variables included in the set are defined by a Variable Set Enumeration. This approach is particularly valuable for managing time-varying attributes or complex lists, as new attributes can be introduced without the physical structure of the tables being altered.

Note: Variable Sets are not currently available for BIRD.

Characteristics:

- **Variable Sets** are most useful when:
  - The list of measures is open or extendable.
  - The structure benefits from a *pivoted layout*.
- **Separate Variables** (explicit attribute/Cube Structure Items) are preferred when:
  - The list of measures is stable and fixed.
  - An explicit tabular structure is clearer and more efficient.

[D] Instrument Type	Carrying Amount	Fair Value
Reverse repurchase loan	31	29
Factoring	19	23
…	…	…

Table 1: Dataset representation without a Variable Set (horizontal format)

[7] Example: If a Cube includes the Variable “Carrying Amount”, which is defined in the core package with CODE = CRRYNG_AMT, the Cube structure must contain the cube structure item linked to that Variable.

[D] Instrument Type	[D] Type of Value	Value
Reverse repurchase loan	Carrying Amount	31
Reverse repurchase loan	Fair Value	29
Factoring	Carrying Amount	19
Factoring	Fair Value	23
…	…	…

Table 2: Dataset representation of the same information as Table 1, using a Variable Set (vertical format)

3.6 Member Hierarchy & Member Hierarchy Node

A **Member Hierarchy** organises **Members** of a Domain into parent–child structures. These hierarchies are used for navigation, presentation, aggregation, and validation. A **Member Hierarchy Node** represents a member within such a hierarchy and contains the metadata required to describe its role and relationships.

Characteristics:

- **Parent-child structure and comparators**: the hierarchies rely on parent–child links that define the overall tree, while aggregation operators determine how sibling nodes are combined (for example, the + operator for summation). Comparators express checks between parents and children (such as = to indicate that a parent equals the sum of its children, or >= when it must be greater than or equal to them).
- **Multiple member hierarchies:** for Domains where multiple hierarchies coexist, a distinction is guaranteed between the **main hierarchy**[8] and any **additional, framework-specific hierarchies** created for purposes such as presentation or analytical views. Multiple hierarchies are historicised independently.

[8] Identified through the Is main field.

3.7 Facet & Facet Item

A **Facet** is a set of constraints attached to a **Non-Enumerated Domain** or **Subdomain**. **Facet Items** define each individual constraint within the Facet. Together, they make a Domain restricted with greater precision than the Domain's general data type alone.

The **Facet value type**[9] defines the general characteristics of the values to which the constraints apply. This is conceptually similar to a Domain data type but can provide finer detail. The concept is derived from the **SDMX standard**.

Common Facet Item Types

- **String constraints**: *Minimum Length*, *Maximum Length*
- **Numeric constraints**: *Minimum Value*, *Maximum Value*, *Decimal* etc.
- **Pattern constraints**: *Pattern* (regular expression). A Pattern can capture multiple restrictions at once (e.g. length, character set, format) in a compact definition.

Characteristics:

- **Purpose and function:** facets support both modelling and validation by allowing metadata designers to specify input restrictions precisely, ensuring that constraints remain transparent, reusable, and governed as metadata rather than embedded in custom code.
- **Technical behaviour:** facets can be associated with either a Domain or a Subdomain, and the same Facet may be reused across multiple contexts, providing a consistent mechanism for defining and applying restrictions.
- **System integration:** systems can consume Facets to enforce input checks, **perform consistency checks on incoming values**, and apply type casting consistently, ensuring uniform behaviour across implementations.

3.8 Worked Examples (End-to-End)

The following examples illustrate how the concepts introduced in the **Core Package** work together to provide a consistent modelling framework.

Example 1 – Currency Domain with Subdomain and Members

- **Domain**: *Currency* (Enumerated)
- **Members**: EUR, USD, JPY, GBP, … (based on ISO 4217 codes)
- **Subdomain**: *EU Currencies* (subset of Members restricted to EUR, SEK, CZK, …)
- **Variable**: *Reporting Currency* bound to Domain *Currency*

[9] Examples of Facet value types: Gregorian Date; DateTime

- **Facet**: none (Domain is Enumerated)

Use case: ensures that reported financial data includes only permitted currency codes, restricted to EU currencies in specific Cubes.

Example 2 – Carrying Amount Variable with Subdomain and Facet

- **Domain**: *Monetary Amount* (Non-Enumerated, numeric type)
- **Facet**:
    - MIN = 0 (amounts cannot be negative)
    - DECIMALS = 2 (precision restricted to cents)
- **Variable**: *Carrying Amount* bound to Domain *Monetary Amount*
- **Subdomain**: *Positive Amounts Only* (applies Facet constraints)
- **Variable Set**[10]: *Type of Value* = { *Carrying Amount*, *Fair Value* }

Use case: enables consistent reporting of amounts in multiple Cubes and supports pivoted structures where multiple measures are consolidated into one dimension.

Example 3 – Member Hierarchy for Instrument Types

- **Domain**: *Instrument Type* (Enumerated)
- **Members**: Loans, Advances, Securities, Debt Securities, Equity Securities
- **Member Hierarchy**:
    - *Financial Instruments*
        - Loans
        - Advances
        - Securities
            - Debt Securities
            - Equity Securities
- **Operators and Comparators**:
    - Parent *Securities* = Debt Securities + Equity Securities (= comparator)
    - Parent *Financial Instruments* ≥ Loans + Advances + Securities (>= comparator)

Use case: ensures that reported totals comply with accounting structures and validation rules.

[10] Currently the Variable set is not used in BIRD.

4. Data definition

The **Logical Structures** of SMCube describe how conceptual elements from the Core Package are assembled to build data models. They define the organisation of datasets (**Cubes),** their attributes (**Cube Structure Items),** and the ways they relate to each other within and across **Frameworks.** Logical Structures act as the "blueprint" of data models: they specify how **Variables** are combined, constrained, and linked to represent real-world reporting requirements. The **Data definition package** thus provides the modelling layer where business concepts (**Variables, Members) a**re assembled into complete data models. It connects the semantic definitions of the **Core Package** with higher-level metadata such as **Transformations, Validations,** and **Rendering,** forming the backbone of the SMCube modelling approach.

4.1 Frameworks

A **Framework**[11] groups together related **Cubes** under a regulatory, statistical, or business context.

Each Framework is time-framed through the **Framework Version** object, which defines its validity period and governance attributes (e.g. whether it is a draft or production version). Frameworks ensure that all contained Cubes share a consistent structural and governance context.

Characteristics:

- All Entity Cubes must be associated with a Framework to associate them with a business context.

Subdomains and Variable Sets[12] can be associated directly to a Framework.

4.2 Cubes

A Cube is the central object of SMCube. It represents either a logical entity type or a specific combination of values for that entity type. At the conceptual level, a Cube defines a structured set of Variables and Members. Such a structure is suitable for rendering tabular representations of data or for use as an input data collection template, without constraining the underlying storage technology

Cube Categories (expressed by CUBE_TYPE)

- **Entity (type) Cubes**: the direct components of a data model:
    - **ERMC (Entity Relationship Model Cube)**: Cubes of an entity–relationship logical model (for example logical data model cubes such as BIRD Logical Data Model Instrument).

[11] Examples include AnaCredit, FINREP, and COREP.

[12] Currently Variable sets are not used in BIRD .

- **RMC (Relational Model Cube)**: Cubes that can be directly translated into relational tables (for example Input Cube such as BIRD Input Layer Instrument).
- **TC (Template Cube)**: Cubes corresponding to the logical structure of a reporting template (for example FINREP template Cube F_01.01L).
- **GC (Generic Cube)**: flexible category for other types of logical structures (for example AnaCredit cubes).

– **Value Combination Cubes** – normally complement Entity Cubes by expressing data point-level or time-series combinations, e.g.:

- **DPC (Data Point Cube)**: Cubes representing individual data points (e.g. FINREP data points of a Template Cube). These concepts correspond to the DPM 2.0 variables.
- **SDPC (Section Data Point Cube)**: Cubes representing section data points.

These types of Cubes restrict the space of their related Cube by listing allowed tuples over its Dimensions.

4.2.1 Cube Structure Items

Every **Cube Structure Item** of a Cube represents a specific attribute of that Cube by linking it to a corresponding **Variable defined and described in the core package** (see Chapter 3). Cube Structure Items specify the **role** and **constraints** of Variables in the context of the Cube.

Roles[13]:

– **Dimension**: identifies records in the Cube, similar to primary keys in a relational table. The Dimension types are:

- **Time dimension** represents the temporal aspect of the Cube (e.g. *Reference Date*).
- **Unit dimension** represents the unit of measurement (e.g. *Currency*, *Percentage*).
- **Measure dimension** indicates which measure the Observation refers to (used in aggregated Cubes with verticalised structures).
- **Breakdown dimension** represents classification or breakdown criteria (e.g. *Instrument Type*, *Institutional Sector*).
- **Open Breakdown dimension** represents open breakdown criteria.

[13] The roles available for BIRD are: B-breakdown; M-Measure; OB - Open Breakdown; SB - Section Breakdown.

    - **Section Breakdown dimension** represents section breakdown criteria (for example Z-axes in EBA templates).
- **Observation**: Cube Structure Items providing values for each unique combination of Dimensions (e.g. *Carrying Amount*).
- **Complementary attribute**: Cube Structure Item associated with another Cube Structure Item, either within the same Cube or in a different Cube, that provides complementary information for the associated item (e.g. the *Null Explanatory Value* associated with *Carrying Amount*).

Restrictions on Cube Structure Items can be applied via:

- **Subdomain**: the value space is limited to a subset of Members (via Subdomain Enumeration) or Facets (for Non-Enumerated Domains).
- **Variable Set**[14]: the value space is defined by a set of Variables rather than Members, supporting verticalised layouts.
- **Member**: the value is fixed to a single predefined Member. Changing it would alter the Cube's scope, as the Member provides contextual information about the Cube.

4.3 Cube Relationships & Cube Structure Item Relationships

A Cube Relationship defines the semantic and structural links between Cubes within a metadata model. Cube Relationships make the logical structure of the model explicit and provide the foundation for data integration, consistency, and validation. According to the SMCube methodology, the following types of Cube Relationships are supported:

- **Association (ASS)** represents associative relationships between Cubes, similar to foreign-key links in relational modelling (e.g. linking an *Instrument Cube* to a *Counterparty Cube*). For this type of relationship, the following metadata is specified:
    - **Is Identifying**: indicates whether the relationship contributes to the identity of the referencing (primary) Cube.
    - **Cardinalities**[15]: define the multiplicity of the relationship from both the primary cube and the foreign cube perspectives.

[14] Currently the Variable set is not used in BIRD.

[15] Examples: N → 1 (many-to-one); 1 → 1 (one-to-one)

- **Mandatoriness**[16] **(Primary and Foreign):** specifies whether participation in the relationship is mandatory or optional for each Cube.

– **Generalisation (GEN)**: A Generalisation represents a parent–child (supertype–subtype) relationship between Cubes. These relationships correspond to inheritance in Entity–Relationship (ER) modelling and are used to model hierarchies of entity type[17]s. Specifically for this type of relationship, the following information is defined:

- **Parent Discriminator** identifies the attribute in the parent (supertype) Cube used to discriminate among child entity types.

Associated Member identifies the specific Member value of the discriminator that corresponds to the child (subtype) Cube. At a finer level of granularity, a **Cube Structure Item Relationship** links specific Cube Structure Items across Cubes, representing attribute-level links (e.g. a foreign key in one Cube referencing a primary key in another). Cube Structure Item Relationships are defined only for Association relationships, as they require explicit attribute-to-attribute connection.

4.4 Cube Hierarchies and Cube Groups

4.4.1 Cube Hierarchies

A Cube Hierarchy is an organisational structure for Cubes, used primarily for classification and navigation rather than logical modelling. Unlike Generalisation Relationships, which represent true subtyping and enforce logical rules, Cube hierarchies are **business-oriented classifications**. Their main purpose is to improve the usability of the model by grouping cubes in a meaningful manner.

In particular, Cube Hierarchies:

– cluster Cubes to make data models easier to understand, explore, and present.

– provide visual and thematic groupings, rather than structural or semantic constraints.

– support multiple levels of classification (e.g. separating Loan-related Instruments from OTC Derivative Instruments).

The number of levels is flexible. However, in BIRD, only one level of business classification is currently used.

[16] Example: If an Instrument Cube must always reference a Counterparty, then the relationship is mandatory on the Instrument (foreign) side. Otherwise, if not all Counterparties necessarily relate to Instruments, then the relationship is optional on the Counterparty (primary) side.

[17] Example: the logical entity “Instrument” is a parent Cube of “Debt Instrument” and “Equity Instrument” child entities.

A **Cube Hierarchy** is composed of **Cube Hierarchy Nodes**, which act as folders or clusters within the hierarchy. Structurally:

- the **leaf nodes** of a Cube Hierarchy are typically linked to a **Cube Group**.
- a Cube Group enumerates the Cubes belonging to that cluster.
- the hierarchy node does **not** directly reference individual Cubes, but instead points to a Cube Group, which acts as the container for those Cubes.

4.4.2 Cube Groups

Cube Groups can also be used independently of hierarchies to create lists of Cubes for other modelling or organisational purposes, making them a **generic and flexible construct**.

5. Transformations & Validations

The Transformations & Validations package defines how logical rules for metadata denormalisation, derivation, generation, and validation are formally modelled within SMCube.

This package provides the conceptual bridge between the **semantic** and **logical layer** (Cubes, Variables, Members) and the **operational layer**, where metadata is wrapped up, derived, generated and validated.

Rather than describing physical processing steps, the package introduces logical, reusable, and interpretable metadata objects that support **business transparency**, **regulatory traceability** and **automated technical implementation**.

The package supports the full spectrum of reporting and regulatory use cases, such as:

- generation of EBA Template and Data Point Cubes, AnaCredit Cubes, etc.
- formalisation of business and regulatory validation rules.

5.1 Routines

A **Routine** is a reusable *building block* that encapsulates **transformation or validation logic**. It does **not** represent a standalone transformation; instead, it is a fundamental component used within **Transformation Rules** or **Validation Rules**. From an operational perspective, a Routine can be described as a **multipurpose, reusable algorithm** applied to one or more source variables in order to derive or verify one or more destination variables. By abstracting common logic, Routines promote consistency, reusability, and maintainability across transformation and validation processes. The following list summarises some purposes of routines:

- Manage metadata (such as a **wrap-up and denormalisation** routine which is applied to a logical attribute to obtain an input variable).
- Derive metadata (such as a **derivation** routine which is applied to input variables to obtain an enriched variable).
- Filter and transform metadata (such as a **generation** routine which is applied to an input variable in order to obtain an output variable).
- Validate metadata (such as **validation** routines).

Characteristics:

- A Routine defines a set of algorithmic steps.
- The algorithm can be expressed in any language.
- In BIRD, Routines are typically written in pseudo-SQL for clarity and consistency.

- If needed, a Routine may be linked to an Annex document[18] describing the logic in detail.

5.1.1 Routine Variable Assignment

The Routine variable assignment is the process of identifying for each routine the following objects:

- **Input Variables**: supplying source data to the Routine.
- **Output Variables**: representing the output results of the Routine.

This ensures transparent lineage of destination Variables and allows impact analysis across Cubes and Cube Structure Items.

5.2 Logical Transformation Rule

A Logical Transformation Rule describes a transformation at the logical modelling level. It is used to formalise how output metadata objects (destination Cubes) are obtained from input metadata objects (source Cubes), independently of any physical implementation. The following types of logical transformation rules are defined:

- **Wrap-up and Denormalisation (WU-DEN)**[19]: Define how LDM structures are reorganised and flattened to form the IL.
- **Derivations**[20]: Specify how new attributes are created within the EIL based on IL/EIL content.
- **Generations**[21]: Describe how EIL content is filtered, combined, and aggregated to produce OL requirements.

Characteristics:

- A Logical Transformation Rule may include one **algorithm** which is written in a logical language (e.g. Pseudo-SQL) and can be expressed through one or more linked **Routines** representing single processing steps.
- Transformation Rules may be linked to an **Annex document** providing a human-readable description of the logic.

---

[18] That can be embedded in the metadata or stored in an official repository and linked via document URL.

[19] Example of WU-DEN: The Party Input Layer cube is obtained from a Logical hierarchy in the BIRD Logical Data Model via a Wrap-up/Denormalisation process.

[20] Example of Derivation: the Carrying Amount derived variable can be derived from the Gross Carrying Amount input variable and included in an Enriched Input Layer cube structure.

[21] Example of Generation: the AnaCredit Instrument Cube can be generated from the BIRD Enriched Input Layer Instrument Cube.

5.2.1 Links between Cubes, Items, and Members (lineage)

SMCube introduces a set of **link objects** to describe how metadata elements relate across Cubes, layers, and Frameworks (lineage). These links are closely tied to **Logical Transformation Rules**, as they document how metadata evolves across layers.

- **Cube Link:** expresses a lineage link between source and destination Cubes. Unlike Cube Relationships which operate *within* a single layer, Cube Links describe **cross-layer or cross-framework** connections.
- **Cube Structure Item Link:** provides a finer-grained view of a Cube Link by specifying which Cube Structure Items correspond between the source and destination Cubes. This allows lineage to be expressed at the attribute level[22].
- **Member Link[23]:** complements Cube Structure Item Links for enumerated items by connecting their members.

Links may be direct when the same concept is reused (e.g. *Credit Card Debt → Credit Card Debt*) or hierarchical/aggregated when a source value is mapped to a broader destination value (e.g. *Credit Card Debt → Loans and Advances*).

Purpose of Lineage Links:

Together, Cube, Item, and Member Links contribute to achieving **a full data lineage** between Cubes.

They can be used to **visualise data process lifecycles** showing dependencies between input and output Cubes.

This provides a **business-readable view** of transformations, supporting both impact analysis and governance.

Often, these links can be **automatically generated** based on the definitions of Transformation Rules and Routines.

5.3 Logical Validation Rule

In SMCube, Validation Rules are expressed as reusable, machine-readable metadata objects that specify consistency checks across Cubes and Cube Structure Items. This ensures that validation logic

---

[22] Example: Carrying Amount in the BIRD Input Layer transformed into Gross Carrying Amount in the Enriched Input Layer.

[23] **Note:** In BIRD, Member links will not be implemented for EBA ITS templates.

is fully integrated with the rest of the modelling framework, historised, and transparent to all stakeholders. From a modelling perspective:

- Validation Rules include an **algorithm** or reference one or more **validation Routines**.
- Unlike Transformation Rules, they do not distinguish between source and output cubes; instead, they identify the set of Cubes and Cube Structure Items involved in the check.
- The **Cube Structure Item Validation** object explicitly records which Cube Structure Items are subject to the rule.
- Validation Rules are associated with metadata lineage.

Logical Validation Rules can be broadly classified into:

- **Structural validation rules**[24] which enforce constraints that are inherent to the data model (e.g. subtype applicability, hierarchy consistency).
- **Business validation rules**[25] which enforce additional rules defined by supervisory or statistical requirements (e.g. date comparisons, accounting identities).

[24] Example of structural validation rule (description): If "Party type by address" is equal to "no registered postal code system party" then "Postal code" must be NULL

[25] Example of business validation rule (description): “Instrument Inception Date must precede Instrument Maturity Date"

6. Rendering Package

The Rendering Package is designed to ensure the **structured representation of EBA reporting templates**. It enables the translation of SMCube metadata into EBA templates, ensuring that the data representation complies with the presentation requirements defined by the Data Point Model (DPM). Its aim is to enable the same metadata used for **semantic and logical modelling** to also drive the **generation of individual data points** and, consequently, of reporting templates.

Thanks to the close alignment with the structures of the DPM 2.0, the translation between SMCube and the DPM is straightforward, consistent and unambiguous.

6.1 Rendering Table

A **Rendering Table** is the central object representing a business-facing template (e.g. balance sheet, profit and loss statement).

Characteristics:

- Each Rendering Table is uniquely identified and historised.
- It defines the full structure of a reportable template, including **rows, columns, and sheets.**
- Tables may be linked to **Template Cubes** or **Data Point Cubes**, ensuring each element of the template corresponds to its logical definition.
- Rendering Tables can also include **open axes** (commonly the Y-axis/rows, but any axis can be open). Open axes allow flexible, potentially infinite sets of values, useful for cases where the content is not predefined.

6.2 Headers and Table Headers

Headers structure the axes of a Rendering Table.

Characteristics:

- Header objects represent **rows, columns, or sheets**.
- Headers are identified with both a **code** and a **label**.

Table Header objects link Headers to Tables and define their hierarchical order (e.g. A*ssets → Loans → Mortgage Loans)*.

Characteristics:

- This hierarchy is for representation purposes and does not necessarily reflect logical hierarchies.

6.3 Table Cell

A **Table Cell** represents the intersection of **Headers** within a **Rendering Table**. A cell can be described as a specific row–column–sheet position.

Characteristics:

- **Active vs. excluded cells**: Cells may be *active* (data expected) or *excluded* (*is excluded* = TRUE, no data expected).
- **Link to Data Point Cubes**: Each Cell can be linked to one or more **Data Point Cubes**, ensuring that the data expected in the cell is precisely defined.
- **Link to system data codes**: Cells may also be linked to a **system data code** representing the technical identifier used for reporting or storage (e.g. the **DPM/XBRL data code** to be reported).
- **Regulatory alignment**: Linking Cells to Data Point Cubes ensures full alignment with **FINREP/COREP datapoints** and other regulatory requirements.

6.4 Cube to Table and Cube Structure Item to Header

The **Cube to Table** and **Cube Structure Item to Header** objects define the explicit mappings between the **logical model** (Cubes and Cube Structure Items) and the **rendering model** (Rendering Tables, Headers, and Cells).

Cube to Table: establishes the link between a **Cube** (logical dataset) and one (or more) **Rendering Tables** that represent it.

- This makes clear which Cube underpins a given template.
- Technically, multiple tables could represent the same Cube in different business layouts.

Cube Structure Item to Header: defines the link between individual **Cube Structure Items** (e.g. Variables, Members, or Dimensions) and the corresponding **Header(s)** in a Rendering Table.

- **Assignment of Cube Structure Item/Members** ensures that each **Cube Structure Item** or **Member** from a Cube is mapped to the corresponding **Header(s)** of a Rendering Table[26].
- **Reusability** allows different Rendering Tables to reuse the same logical metadata but present it in distinct layouts (e.g. different reporting templates using the same Cube and Variables).

Characteristics:

[26] Example: the Cube Structure Item Instrument Type with the Member Loans and Advances is assigned to row 0080 of the FINREP F 05.01 template.

- **Consistency**: guarantees that reported templates are always anchored to the correct logical metadata definitions.
- **Flexibility**: enables a single Cube to feed multiple templates, and conversely, for one template to combine information from several Cubes.
- **Governance**: mappings are historised and attributable, ensuring traceability when template structures or Cube definitions evolve.

7. Legal Reference and Classification Package

The **Legal Reference and Classification Package** links SMCube metadata objects to **legal sources** and **business classifications**, reinforcing governance, compliance, and thematic organisation. By doing so, it ensures that metadata can be transparently tied to its regulatory basis and categorised for analytical or supervisory use.

Key objects include:

- **Legal Reference**: associates a metadata object (e.g. a **Variable**, **Member**, or **Cube**) with a specific Legal Reference. This provides traceability from the modelled concept back to its legal foundation.
- **Legal Text**: references an external legal act, regulatory document, or guidance (e.g. *EBA ITS on Supervisory Reporting*).
- **Classification**: a tag or label applied to metadata objects, often grouped under defined categories (e.g. *Risk-related*, *Credit Quality*, *Liquidity*).
- **Classification Assignment**: establishes many-to-many links between metadata objects and Classifications, allowing the same object to be tagged under several perspectives.

Characteristics:

- **Traceability**: each metadata element can be traced back to its regulatory basis or guidance document.
- **Categorisation**: metadata can be grouped into thematic sets, supporting business analysis, supervision, or internal organisation.
- **Flexibility**: Classifications are not limited to legal contexts; they can also capture organisational categories (e.g. “Statistical vs. Supervisory”).
- **Governance**: all links are historised and attributed to a Maintenance Agency, ensuring accountability when regulations or classifications evolve.

8. Mapping Package

The **Mapping Package** provides the functionality to **align and reconcile metadata concepts** across Frameworks or between SMCube and external standards. It ensures that different codification systems can interoperate seamlessly within the SDD.

Note: The Mapping Package is **not currently available for BIRD**.

Key Objects:

- **Mapping Definition**: the central component of the package, capturing the logic of a mapping between **Variables, Members**, or **Cubes**. Mapping Definitions can take different forms:
    - **Equivalence Mapping**: links enumerated Variables or Members directly (e.g. ISO country codes → ECB reference codes).
    - **Algorithm Mapping**: applies transformation logic when equivalence is not sufficient.
    - **Generation Mapping**: introduces implicit or generated information in the target structure.
    - **Deletion Mapping**: removes Variables that are not required in the target structure.
    - **Variable Set Mapping**: manages mappings when Variables are represented as members of a Variable Set.
- **Variable Mapping** and **Member Mapping**: describe correspondences between individual Variables or Members.
- **Cube Mapping**: aligns entire Cubes, or subsets of Cube Structure Items, across systems.
    - Example: the FINREP F 05.01 Cube as defined by the EBA ITS dictionary can be mapped to the equivalent Cube in the ECB/BIRD dictionary.
- **Variable Set Mapping** (extended use): defines how a group of Mapping Definitions apply to specific measures of a Cube within a Variable Set.
    - Example: mapping Carrying Amount in the context of a specific Variable Set of FINREP F 05.01.

Characteristics:

- **Historicisation**: all mappings are versioned with VALID_FROM and VALID_TO, ensuring full traceability over time.
- **Reusability**: once defined, mappings can be applied consistently across multiple Frameworks and datasets.
- **Interoperability**: mappings allow systems based on different standards (e.g. ECB, EBA, ISO) to exchange and reconcile metadata seamlessly.