SMR - Database Design

- Detailed Database Design
  - Data Software Objects and Resultant Data Structures
    - The RDL
    - The SDS
    - Metadata Repository
  - Database Management System Files
    - Data Types
    - Physical Constraints
    - Indexes
    - Triggers

Detailed Database Design

NM HSD Medicaid data is exchanged across multiple systems in various formats. It is difficult and time consuming for agency staff to gather all related data from various data sources. The SMR will provide consistent and consolidated enterprise data in a centralized repository. To achieve this, the SMR maintains two data stores: the RDL and the SDS.

The SMR will also maintain a metadata repository, along with the RDL and SDS, that contains the metadata of the source data. The following subsections identify the structure of the metadata repository as well. The following subsections provide details on nature and data structures of the RDL and SDS.

The design discussed here addresses the business requirements that have been documented in the Jama project (Link).

The design follows the architectural strategies, system design and detailed database design documented in the deliverable SIPLT1: System Design Document.

Data Software Objects and Resultant Data Structures

The SMR will have the three data store components, with each having its own data objects and structures as explained in the following subsections.

- The RDL
- The SDS
- Metadata Repository

Note:

- The Operational Data Store (ODS) will be built on top of the SDS. So, the design for SDS will be applicable to the ODS as well.
- The Master Data Management (MDM) data stores will be developed in future, and this design documentation does not capture the MDM design.

The RDL

The RDL stores the data as loaded and it follows the native data format from the legacy source system. The RDL supports the following data formats:

- Structured data from relational databases in rows and columns.
- Semi-structured data, such as CSV, logs, extensible Markup Language (XML), and JavaScript Object Notation (JSON).
- Binary data, such as Portable Document Format (PDF) files, images, audio, and video
- Document or Object-based NoSQL data.

For example, the native structure of the Omnicaid data is preserved along with the original integrity, type, length, and structural constraints. When a row of tabular data is loaded into the MarkLogic, it is represented as an XML document.

The RDL structure for a given source is the As-Is data model of that source. The As-Is data model of Omnicaid and ASPEN are captured in the document SIDM1: Conceptual Data Model (CDM).

The SDS

A high-level logical ERD is a pictorial representation of the relationships between the data domains described within the Physical Data Models (PDM). The high-level data models contain the MMISR "To Be" view. The following figure provides a very high-level view of how the data domains interact with each other at the conceptual level. This figure does not show the details of the domains. However, the details of each data domain along with the business entities and the data elements contained in those data domains are addressed in the following deliverables:

- SIDM1: Conceptual Data Model (CDM)
- SIDM2: Logical Data Model (LDM)
- SIDM3: Physical Data Model (PDM)
- SIDM4: Information Governance Catalog
An Enterprise Data Model (EDM) is an integrated view of the data produced and consumed across an entire organization. It follows the standard message exchange models and guidelines like NIEM and FIHR. An EDM represents a single integrated definition of data that is unbiased of a system or application and is independent on how the data is sourced, stored, processed, or accessed. It unites, formalizes, and represents the information important to an organization, as well as the rules that govern them.

A high-level definition of data domains and business entities is provided below:

- **A data domain** is a high-level functional area (or subsystem) of the given source system through which the source system coordinates with the related business workflows and Business Entities.
- **Business entities** are recognizable concepts – such as a person, places, things, or events – that represent the given Data Domain.

For example, "Provider" is a data domain and the Provider License or Provider Demographics are business entities.

The MMISR data is modeled into the following nine data domains:

- **Client**
  A client is a person or entity that is or has been eligible and enrolled in the State's Medicaid Management Information System (MMIS) program.

- **Claim**
  A claim is a bill for services submitted by a Provider to the State or depending on the claim type, a line item of service on a bill, or all services for one Member within a bill.

- **Drug Rebate**
  The Drug Rebate extracts pharmacy and medical claims, provider, and drug reference files to create invoices requesting rebates for specified drugs. Payments made have to be tracked and related back to the invoices.

- **Financial**
  The financial captures Gross Level Payouts, Cost Settlement, Reconciliations, Recoupments, Accounts Receivables, and Creation of interfaces to Statewide Human Resources Accounting Reports (SHARE) and Department of Finance and Administration (DFA).

- **Managed Care Organizations (MCO)**
  Managed Care is Information sharing member affiliations with Primary Care Physicians (PCPs), Health Homes, Disability status, Long Term Care status, Care Coordination, and member assessments and reporting on MCO activities such as pilot projects, value-based purchasing agreements, etc.

- **Prior Authorization (PA)**
  The PA subsystem collects and maintains comprehensive current and historical information about Pas, which are submitted for the determination of medical and dental necessity for Medicaid, and waivered services for the clients of NM's Medicaid Program. ([HHS Procurement Library: XEROX 07D-Prior Authorization 1narr](#)). Prior Authorization PDCS Association [A_PA_PDCS_TB]).

- **Provider**
  A Provider is an individual, institution, facility, agency, physician, health care practitioner, non-medical individual agency, or other entity that is licensed or otherwise authorized to provide any of the Covered Services in the State to HHS 2020 enterprise agencies. Providers include individuals and vendors providing services under a Managed Care contract agreement to ([HHS Procurement Library: 8-HHS 2020 Terms and Definitions](#)).

- **Third-Party Liability (TPL)**
The TPL subsystem narrative maintains comprehensive current and historical information to support the benefit recovery functions of the NM Omnicaid MMIS. The Medical Assistance Division (MAD) uses this information to reduce its liability to pay for client Medicaid claims. The TPL Subsystem ensures that Medicaid is the payer of last resort by identifying, cost avoiding and recovering from liable third-parties.

The figure below provides a high-level Entity/Relationship (E/R) diagram across the above listed data domains.

**Figure 5: High-Level Entity Relationship Diagram across Data Domains**

---

**Metadata Repository**

Metadata is data about data and provides context for source and target datasets. The Metadata repository containing all source schemas and the associated source record names uses a format similar to the following sample format.

Along with the versioned source schemas, the metadata repository also stores SMR's Physical Data Models (PDM). The PDM represents identified data domains and business entities, data elements contained in those entities, and the relationship between each of them. It also contains the physical properties of the data format, such as the schema definitions. Every PDM is depicted as an XSD, which also functions as metadata and a specification that the data, stored as an XML, will conform to. More information on the PDM as metadata can be found in deliverable SIDM3: Physical Data Models.

**Sample Format 1: Metadata Structure**
Database Management System Files

This subsection provides a detailed design of the MarkLogic NoSQL database management system and the MMISR data in the SMR.

Data Types

Data elements are also represented within the PDM as XSD element tags. These data elements contain the following metadata:

- 'name' attribute that defines what would be the corresponding XML tag
- 'type' attribute that defines the data type of the element. It can be an XSD simpleType or complexType
- 'minOccurs' attribute that defines the minimum times the element should appear within the entity. A value of '0' means, the element is optional, and a value of '1' means the element is mandatory
- 'maxOccurs' attribute that defines maximum times the element can appear within the entity. A value of '1' means the element can appear only once and has a 'one to one' relationship within the entity. A value of 'unbounded' means the element has a 'one to many' relationships within the entity

Element definitions are accompanied by 'annotation' tags, which contain 'documentation' tags that include the definition and vocabulary of the data element. Each data element definition contains the 'type' attribute, which is used to define the data type of the element. These types are always associated with the W3C standard data types or data types that extend the W3C standard type with restrictions. They can be defined as a simpleType or complexType. The following are the 'simpleType' definitions found within the SMR's Physical Data Models (PDM):

- xs:string – This is a W3C standard simpleType. The string data type can contain characters, line feeds, carriage returns, and tab characters. It is alphabetic when associated with regex validations.
- com:AlphabeticDataType – This is a custom data type defined within the 'Common' package. This is an extension of the xs:string data type but restricts this to only alphabetic characters using regular expression (Regex) specifications.
- xs:unsignedLong – This is a W3C standard simpleType. The unsignedLong data type represents an unsigned 64-bit integer. This data type is generally applied to representing an identifier that is numeric in nature.
- xs:nonNegativeInteger – This is a W3C standard simpleType. This represents an integer containing only non-negative values (0, 1, 2, etc.). This data type is generally applied to representing numeric units.
- com:AmountDataType – This is a custom data type defined within the 'Common' package. This is an extension of the W3C standard simpleType of xs:decimal. This restricts the digits after the decimal to 2. This data type is generally applied to representing monetary amounts.
- com:UserDataType – This is a custom data type defined within the 'Common' package. This is an extension of the W3C standard simpleType of xs:string. This data type is generally applied to representing a user name.
- xs:date – This is a W3C standard simpleType. The date data type used to specify a date in the following form ‘YYYY-MM-DD’ where:
  - ’YYYY’ indicates four digits of the year
  - ’MM’ indicates two numeric digits of the month
  - ’DD’ indicates two digits of the day
- xs:dateTime – This is a W3C standard simpleType. The dateTime data type is used to specify a date and a time combined in the following form ‘YYYY-MM-DDThh:mm:ss’ where:
  - ’YYYY’ indicate four digits of the year
  - ’MM’ indicate two numeric digits of the month
  - ’DD’ indicate two digits of the day
  - ‘T’ indicates the start of the required time section
  - ’hh’ indicates two digits of the hour
  - ’mm’ indicate two digits of the minute
  - ’ss’ indicate two digits the second.
  - When an element is defined as a dateTime, all components of the date and time are mandatory.
- xs:anyURI – This is a W3C standard simpleType. The anyURI data type is used to specify a URI. This data type is generally applied as a pointer, or reference, to other documents within the database.
• **xs:Boolean** – This is a W3C standard simpleType. The Boolean data type is used to specify a true or false value. This data type is generally applied to indicator type fields in the source data. Typically, ‘Y’ value in the source is converted to ‘true’ and ‘N’ value in the source is converted to ‘false’, unless specified otherwise.

The data type associated with an element can be a complexType also, and as appropriate, those data types are defined within the domain-specific package or the ‘common’ package.

**Physical Constraints**

Constraints illustrate conditions that are applicable by enforcing limits on the data or type of data that can be inserted, updated, or deleted from a business entity. The whole purpose of constraints is to maintain data integrity during ‘update,’ ‘delete,’ and ‘insert’ operations on a business entity. The source RDBMS constraints are converted to XSD constraints for storage within MarkLogic.

The following constraints are within the ERD:

• **Keys** – Keys are data elements or sets of data elements that uniquely identify an entity within its entity set. There are two different types of keys applicable within the PDMs:
  
  • **Primary Keys** – Primary keys uniquely identify each record in a table. They must have unique values and cannot contain nulls. There can be one or more elements within a business entity that uniquely identifies a business entity record. ERDs show the primary keys with a `<<PK>>` notation. Within the XSD definition, primary keys are defined as ‘xs:selector’ tags that contains the ‘XPath’ expression of the business entity, and a set of one or more ‘xs:field’ tag that contains the ‘XPath’ attributes pointing to the elements included as part of the key.

  • **Foreign Keys** – Foreign keys are the elements of a business entity that point to the primary key of another business entity, thereby acting as a cross-reference between business entities. The foreign keys within a business entity can be one or many, and they must be available in the other business entity it relates to. Every relationship within an ERD has to refer to a particular foreign key constraint. ERDs show the foreign keys with a `<<FK>>` notation. Within the XSD, definition foreign keys are defined as ‘xs:keyref’ tags. It contains ‘xs:selector’ tag that contains the ‘XPath’ expression of the business entity, and a set of one or more ‘xs:field’ tag that contains the ‘XPath’ attributes pointing to the elements included as part of the key.

• **Nullable/Not null constraints** – Nullable constraints mean that the element value can be null. Not null constraints mandate that there have to be a value populated and associated with the business element, meaning that the business element value cannot be null or blank. Not null constraints are shown within the ERD alongside the data element wherever applicable with a ‘[0..1]’ or ‘[0..unbounded]’ for the nullable field. The omission of the nullable expression denotes the element is mandatory. Within XSD definitions, the ‘minOccurs’ attribute and the ‘maxOccurs’ attributes are used to define if an element is mandatory or optional. Nullable constraints mean that the element value can be null. Not null constraints mandate that there have to be a value populated and associated with the business element, meaning that the business element value cannot be null or blank. Not null constraints are shown within the ERD alongside the data element wherever applicable with a ‘[0..1]’ or ‘[0..unbounded]’ for the nullable field. The omission of the nullable expression denotes the element is mandatory. Within XSD definitions, the ‘minOccurs’ attribute and the ‘maxOccurs’ attributes are used to define if an element is mandatory or optional.

• **Defaults** – Sometimes the Not null constraint is also associated with a default value. If a default value is associated with a data element it is shown right next to the data element with a ‘=’ sign leading to the default value for the data element when no value is explicitly assigned. Within XSD, the ‘default’ attribute is used wherever applicable to define defaults if they exist.

• **Check** – Check constraints are conditions that apply for one or more data elements within a business entity. ERDs show check constraints with a `<<check>>` notation. XSDs only refer to unique constraints automatically using the ‘xs:unique’ tag, but the other constraints of this type are handled by the code accompanying the data.

**Indexes**

MarkLogic applies a universal index that indexes every word in the database. However, SMR leverages specific indexing capabilities offered by MarkLogic.

**Triggers**

The typical usage of triggers does not apply to SMR; thus, no triggers will be used.