SDMX Indonesia

Rilis 0.1

Satrio Waskitho

17 Nov, 2021

Daftar Isi

SDMX (*Statistical Data and Metadata eXchange*) menyediakan *metamodel*[1] untuk mendeskripsikan data dalam cakupan statistik. Asal-usul model SDMX dapat ditelusuri ke dalam model *Generic Statistical Message* (GESMES)[2]. Pada kenyataannya inti dari model SDMX adalah evolusi yang telah terjadi dari waktu ke waktu sejak tahun 1970-an. Setelah peluncuran SDMX pada tahun 2001, model ini telah ditingkatkan secara signifikan dan diimplementasikan sepenuhnya dalam XML (SDMX-ML) dan JSON (untuk mendukung diseminasi data melalui web).

Inisiatif SDMX disponsori oleh tujuh organisasi, di antaranya:

Bank for International Settlements (BIS); European Central Bank (ECB); Eurostat; International Monetary Fund (IMF); Organisation for Economic Co-operation and Development (OECD); United Nations Statistics Division (UNSD); World Bank.

SDMX mendukung banyak aktivitas statistik dan proses yang mendukung aktivitas tersebut di antaranya:

- *Data collection*— data registrasi dan pencacahan, serta validasi data.
- *Data reporting* dan *data mapping*.
- *Data dissemination— data discovery, data query,* dan portal data.
- Repositori metadata struktural untuk manajemen metadata, *query*, dan pencacahan.

[1] *Metamodeling* merupakan pembuatan sekumpulan "konsep" (seperti benda, istilah, dan lain lain.) dalam suatu cakupan tertentu.
[2] GESMES merupakan standar UN/EDIFACT.

BAB 1

Daftar Isi

1.1 Mengapa Menggunakan SDMX?

1.1.1 Dukungan untuk Beberapa *Use Cases*

Pertama, penting untuk diketahui bahwa SDMX dapat memberikan lebih dari sekadar format umum bagi *data collectors* dan *data reporters* yang digunakan dalam pertukaran data dan metadata, meskipun akronimnya menggambarkan sebaliknya.

Kedua, jangan khawatir tentang bagaimana representasi sintaks dari data dan metadata. Ada banyak *tools* dan aplikasi *open source* yang dapat mengatasi kerumitan ini dan dengan demikian memungkinkan Kamu untuk menggunakan SDMX dalam memecahkan masalah tersebut. Misalnya, bagaimana cara menggunakan SDMX di aplikasi statistik favorit Kamu seperti SAS, R, atau Excel. Bagaimana membuat situs web diseminasi maupun membangun sistem pengumpulan data yang tangguh.

Dan terakhir, perhatikan Model Informasi SDMX dan pahami bagaimana model tersebut dapat mendukung *use cases* yang Kamu miliki. Ketika Kamu mempelajari model ini (kami akan menjelaskan model ini nanti) kami yakin bahwa Kamu akan menyadari bahwa SDMX dapat memenuhi kebutuhan-kebutuhanmu. Kemudian dengan aplikasi *open source*, Kamu juga akan mendapati bahwa mengimplementasinya pada sistemmu akan jauh lebih cepat dari apa yang Kamu bayangkan sebelumnya.

1.1.2 Model yang sudah Matang

SDMX, pada intinya, adalah Model Informasi yang diimplementasikan dalam sintaks tertentu (terutama XML). SDMX responsif terhadap teknologi baru karena representasi sintaksis dapat dibangun dengan mudah serta memiliki basis Model Informasi. Sebagai contoh, format JSON terbaru telah dikembangkan dan format tersebut semakin populer di kalangan pengembang web.

Kami telah menerapkan sistem pengumpulan data, validasi data, dan diseminasi data untuk sejumlah klien. Kami juga telah membangun aplikasi SDMX sejak tahun 2005, dan kami telah mengembangkan dan merilis SdmxSource (*open source*). Kami senang untuk berbagi pengetahuan ini untuk menunjukkan bagaimana sistem dengan "SDMX Inside" dapat memungkinkan Kamu untuk menemukan solusi dengan biaya yang sedikit, sumber daya yang sedikit baik pada tahap inisiasi maupun pada tahap yang sedang berlangsung, serta dalam waktu yang lebih singkat.

Model Informasi adalah sintaks dengan format agnostik[1] yang sebagian besar proses dan fungsi dapat dikembangkan di sekitar model, dan bukan di sekitar sintaks. Ini adalah inti dari SDMX Source yang terdiri dari aplikasi *open source* yang digunakan untuk mengembangkan aplikasi berbasis SDMX. Banyak aplikasi telah dikembangkan untuk memproses SDMX (*reading*, *writing*, validasi, transformasi, pemetaan). Aplikasi-aplikasi tersebut dapat digabung dan dijadikan satu ke dalam sistem tanpa menghiraukan pembuat dari suatu komponen tersebut telah memiliki pengetahuan untuk mematuhi API yang sama.

SDMX Source dikembangkan oleh *Metadata Technology* dan digunakan oleh banyak institusi besar, termasuk Eurostat yang telah mengadopsi *SDMX Source* sebagai *framework* dasar untuk *tools* dan aplikasinya. Pembangunan aplikasi menggunakan *framework* ini dapat menjamin penyesuaian SDMX baik untuk impor maupun ekspor informasi.

1.1.3 Standar ISO

SDMX hadir dengan sponsor yang berasal dari organisasi internasional (BIS, ECB, Eurostat,IMF, OECD, PBB, Bank Dunia) dan memiliki status ISO (standar internasional 17369). SDMX dijaga dan dipertahankan secara aktif dengan merespons permintaan fungsi baru melalui proses yang terbuka.

1.1.4 Pengembalian Investasi

Sering disebutkan, bahwa sulit untuk melakukan justifikasi terhadap investasi pada standar yang spesifik jika hanya digunakan untuk satu hal saja. SDMX dapat melakukan lebih dari sekadar bertindak sebagai format umum untuk bertukar data statistik dan metadata. Jika Kamu menggunakan SDMX sebagai model pada sistem pengumpulan data, *data reporting*, dan diseminasi data, maka akan ada banyak manfaat yang Kamu dapatkan. Semakin banyak Kamu menggunakan kelebihan dari SDMX di sistemmu, semakin banyak manfaat yang akan Kamu peroleh.

1.2 Penjelasan Singkat Model Informasi SDMX

1.2.1 Cakupan Permasalahan

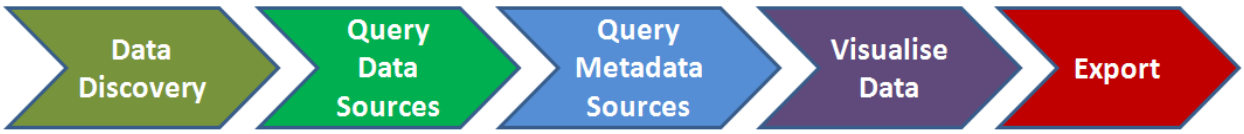
Suatu sistem statistik terdiri dari banyak sub sistem dan komponen. Persoalan utama sistem komputer termasuk sistem statistik adalah sistem —atau dapat disebut perangkat lunak— dibangun secara silo atau setidaknya dibangun dengan komponen yang tidak independen dan tidak dapat digunakan kembali.

Pertimbangkan alur proses sederhana berikut ini, yaitu alur impor data ke basis data dan alur diseminasi data.

Alur Impor Data

Alur Diseminasi Data

[1] Dalam komputasi, perangkat atau program perangkat lunak dikatakan agnostik atau data agnostik jika metode atau format transmisi data tidak relevan dengan fungsi perangkat atau program. Ini berarti bahwa perangkat atau program dapat menerima data dalam berbagai format atau dari berbagai sumber, dan masih memproses data tersebut secara efektif.



Berikut adalah beberapa kelemahan yang kemungkinan terdapat di dalam sistem yang saat ini mendukung proses tersebut.

Desain Basis Data

Skema basis data dirancang secara internal sehingga dapat menampung semua data yang dikerjakan di dalam suatu departemen (selanjutnya akan disebut institusi). Institusi dapat memutuskan untuk mengadopsi basis data Oracle, dan tabel basis data disesuaikan untuk menyimpan data spesifik yang relevan sesuai dengan cakupan tempat institusi bekerja. Tidak perlu menggunakan lebih dari 2 bahasa, setiap tabel berisi label khusus untuk setiap bahasa di mana pun data tersebut muncul.

Ada satu tabel basis data yang besar berisi nilai observasi untuk setiap dataset. Ada banyak tabel basis data terkait yang berisi informasi lebih lanjut tentang pengamatan. Tabel tertaut digunakan untuk memfasilitasi pemfilteran saat membuat kueri basis data. Data dan informasi di dalam tabel tersebut berguna untuk menyusun hasil kueri yang akan disajikan kepada pengguna.

Impor Data

Institusi menerima data dari banyak penyedia data lain, dan dengan demikian format data di setiap dataset bergantung pada si pengirim data. Pengimpor data dicatat untuk mendukung dokumentasi ketika terdapat penawaran data. Dalam beberapa situasi, tabel pemetaan perlu didefinisikan untuk memetakan klasifikasi pengguna data ke pemetaan yang digunakan di internal institusi. Terdapat suatu aturan validasi yang ditetapkan untuk memastikan data sesuai dengan apa yang diharapkan. Setiap importir mengimplementasikan logika validasinya sendiri. Terdapat juga validasi tambahan yang digunakan setelah data berada di dalam basis data.

Diseminasi Data (1)

Untuk melakukan diseminasi data, tim analisis bisnis menentukan jenis kueri mana yang diperlukan. Kemudian, tim developer menulis kueri basis data untuk mendefinisikan *use cases*. *Use cases* yang didefinisikan memiliki output untuk mengekstrak dataset yang relevan. Tidak ada model yang baku untuk data, sehingga output sintaks bergantung pada penerima data. Jika pengguna data memerlukan format output yang berbeda, tim developer akan menulis kueri basis data baru atau menulis transformasi dari format yang sudah ada.

Situs web dibangun di atas API yang ditentukan oleh tim analisis bisnis, dengan desainer web dan developer *backend* bekerja sama untuk membangun halaman web. Semakin banyak *use cases* yang berkembang, tim developer akan menulis API baru untuk mendukung situs web.

Secara internal, unit lain di dalam institusi diberikan akses langsung ke tabel basis data. Unit-unit tersebut menulis logika kueri mereka sendiri berdasarkan struktur tabel untuk mendapatkan data dari sistem mereka sendiri.

Pemeliharaan Sistem

Sistem ini menggabungkan kedua data, termasuk metadata yang diperlukan untuk memahami data. Aplikasi internal dibangun untuk memberi akses pengguna ke informasi pencarian seperti klasifikasi. Pengguna lain juga diizinkan untuk memodifikasi dan menambahkan informasi.

Dalam beberapa situasi, tidak ada antarmuka pengguna untuk jenis informasi tertentu, sehingga pengguna harus menggunakan kueri untuk mengakses basis data secara langsung. Dalam beberapa kasus, satu-satunya cara untuk memodifikasi jenis metadata tertentu adalah dengan memodifikasi basis data secara langsung.

Apa yang Salah dengan Pendekatan Ini?

Pendekatan ini memberikan contoh yang baik dari sistem yang sangat memiliki ketergantungan satu dengan yang lain di hampir setiap titik. Sistem yang tidak independen berarti bahwa mengubah satu aspek sistem memiliki dampak di seluruh sistem secara keseluruhan. Sistem yang saling memiliki ketergantungan memiliki ketahanan yang rendah untuk berubah dan biaya pemeliharaan yang tinggi dalam hal waktu dan uang.

Contoh pertama dari tingkat ketergantungan yang tinggi adalah skema basis data itu sendiri, yang desainnya digabungkan dengan dataset dan *use cases* pada saat melakukan desain. Setiap perubahan skema akan berdampak pada situs web, semua output untuk setiap pengguna data, dan karena basis data ini ditanyakan oleh unit lain, itu juga akan mempengaruhi semua pengguna data internal. Pengguna data internal juga dihubungkan ke platform Oracle.

Backend API digabungkan sesuai persyaratan situs web dan aplikasi *maintenance* yang ada. Karena kueri basis data telah dibangun untuk melayani API ini, kueri tersebut juga digabungkan sesuai dengan persyaratan. Karena situs web dibangun dengan langsung memanggil *backend API*, situs web digabungkan sesuai dengan bahasa pemrograman tempat sistem dibangun. Setiap perubahan terhadap persyaratan berdampak pada API tersebut dan setiap perubahan pada API memengaruhi situs web.

Tiap data dari *subject matter* (importir data) digabungkan menjadi format yang sesuai dengan datanya masing-masing. Jika *subject matter* (klien) mengubah format data mereka, ini akan berdampak pada pemrosesan terhadap data *subject matter* tersebut. Tiap format yang memiliki logika impornya masing-masing dapat menghasilkan *bugs* di beberapa data *subject matter* yang tidak ada di tempat lain (data hanya dimiliki oleh satu *subject matter* tersebut). Hal ini akan mengarah pada *high testing overhead* karena terdapat sedikit logika yang digunakan bersama-sama di antara tiap-tiap *subject matter*. Ketika format impor yang baru diperlukan, maka beban pemeliharaan akan meningkat.

Tidak ada model internal yang dapat mengarahkan kueri ad hoc[1] untuk ditulis sesuai kebutuhan agar mendukung *subject matter* internal, *subject matter* eksternal, situs web, dan aplikasi *maintenance*. Hal ini menyebabkan banyak API memiliki duplikasi logika dari waktu ke waktu sehingga sebagian kode menjadi kurang terstruktur dan menghasilkan *Spaghetti Code*. Aplikasi menjadi lebih kompleks untuk di- *maintenance* dan jauh lebih rentan terhadap *bugs*. Hal tersebut disebabkan karena perubahan pada satu bagian aplikasi dapat memiliki dampak yang tidak diinginkan pada bagian aplikasi yang tampaknya tidak terkait.

Tidak ada ketentuan untuk mengubah platform basis data (DBMS). Jika suatu institusi lebih menyukai SQL Server daripada Oracle, maka akan ada beban pemeliharaan yang sangat besar untuk menyesuaikan semua logika kueri. Skrip migrasi perlu ditulis untuk memigrasikan data dan semua kode perangkat lunak perlu dimodifikasi.

Aplikasi yang dibangun untuk melihat dan memelihara metadata pendukung akan menjadi beban *maintenance*, termasuk DBMS dan desain basis data. Pengenalan jenis metadata baru atau memodifikasi metadata yang telah ada tidak hanya membutuhkan perubahan basis data, tetapi juga modifikasi pada aplikasi *maintenance*.

Tidak ada ketentuan untuk mendukung data yang memiliki cakupan baru. Jika suatu unit diminta untuk menyimpan jenis data baru, maka perubahan pada kode dan modifikasi basis data akan diperlukan. Karena tingkat ketergantungan di dalam sistem yang tinggi, perubahan ini membutuhkan banyak *testing* tambahan.

[1] Kueri ad hoc adalah kueri tunggal yang tidak disertakan dalam *Stored Procedure* dan tidak diparameterisasi atau disiapkan untuk tujuan secara umum. Contohnya: `var newSqlQuery = "SELECT * FROM table WHERE id = " + myId;`

Kueri ke basis data membutuhkan banyak *join* tabel yang mengarah ke performa yang buruk. Karena situs web, aplikasi *maintenance*, *subject matter* eksternal dan internal digabung menjadi satu struktur tabel, tentu tidak mungkin untuk meningkatkan performa dengan mudah.

Apakah Masalah Ini akan Terus Terjadi?

Penjelasan sebelumnya mungkin melukiskan gambaran yang hampir apokaliptik tentang apa yang bisa terjadi. Kami tentunya tidak menginginkan situasi tersebut hadir dalam suatu sistem. Kami telah mengamati semua aspek dalam sistem dan kami telah memberikan saran konsultasi tentang pendekatan berbasis model (SDMX).

Terdapat cara yang lain untuk merancang sistem yang digunakan untuk melakukan pengumpulan, pelaporan, diseminasi data dan metadata, serta integrasi dengan *tools* analisis data yang digunakan oleh organisasi. Cara tersebut adalah dengan menggunakan pendekatan berbasis model dan arsitektur komponen yang mendukung model.

1.2.2 Pendekatan Berbasis Model

Dengan menyelaraskan bahasa yang digunakan untuk menjelaskan data dan metadata terkait, sangat mungkin untuk mengintegrasikan sumber data yang berbeda. Hal tersebut juga memungkinkan aplikasi perangkat lunak untuk dapat mengakses beragam dataset menggunakan *common language* terlepas dari produk perangkat lunak yang digunakan untuk menyimpan data.

Solusi SDMX memperkenalkan model data dan metadata internal yang *powerful*. Model Informasi SDMX dibangun dengan menganalisis proses internal banyak lembaga statistik dan bank sentral, serta menyadari bahwa meskipun masing-masing aplikasi lembaga tersebut berbeda, mereka semua melakukan hal yang sama. Mampu menggambarkan data dan metadata yang mendukung aplikasi statistik apa pun dengan cara generik, tujuan tersebut mengarahkan pada kemampuan untuk mengembangkan modul perangkat lunak generik yang dapat memproses data dalam domain statistik apa pun dengan cara yang lazim digunakan.

Model Informasi SDMX adalah model data. Hal itu tidak menentukan *behaviour* (misalnya *behaviour* apa yang harus dimiliki sistem saat memproses Kode) meskipun berbagai spesifikasi mungkin tergolong *high level behaviour* yang spesifik, seperti mengirimkan metadata struktural ke Registri SDMX.

Pada dasarnya, model data menentukan ruang lingkup sistem atau standar dalam hal:

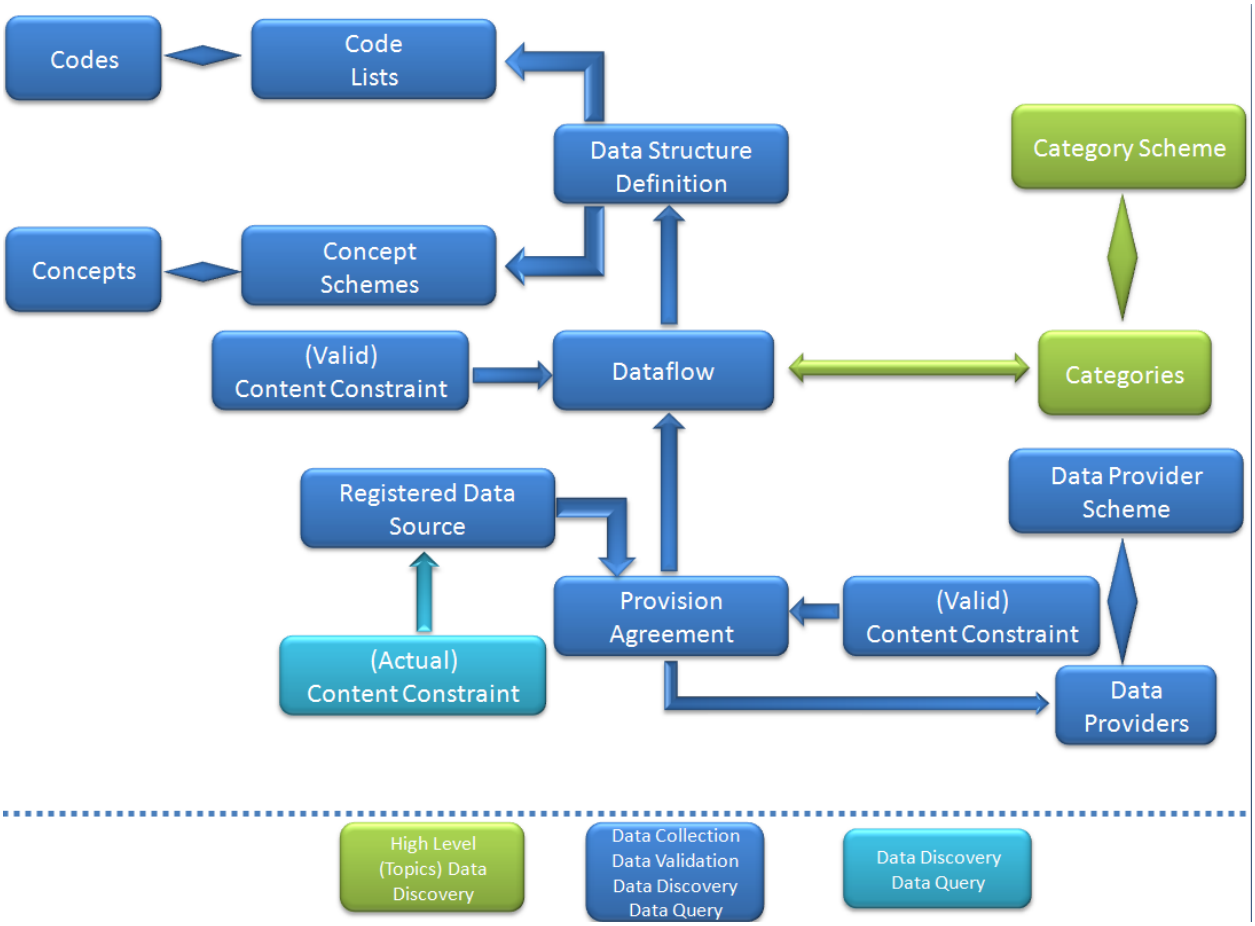
- Informasi yang akan dibagi antara proses atau institusi dalam hal objek informasi (misalnya Kode) dan isi objek (misalnya id kode, label kode);
- Hubungan antar objek informasi.

Agar model menjadi berguna, maka harus memiliki implementasi. Sebagai contoh, harus ada cara untuk mewakili daftar kode tertentu termasuk kodenya dalam suatu sintaks seperti XML dan hal ini telah ada di dalam SDMX. SDMX memiliki lebih dari satu cara untuk mewakili contoh tertentu dari objek informasi. Ini merupakan poin penting dan menjadi manfaat utama dari model informasi karena representasi sintaks yang berbeda dapat didukung. Jika arsitektur sistem dirancang dengan baik, maka tidak perlu lagi sebagian besar komponen sistem untuk memikirkan implementasi terkait sintaks. Kumpulan komponen dibangun untuk memahami objek model bukan sintaksnya, yang mana objek-objek inilah yang diimpor ataupun diekspor.

Pada dasarnya, hal ini merupakan pendekatan berbasis model yang digunakan untuk *system engineering*. Bekerja dengan objek di dalam suatu model harus direalisasikan sebagai objek juga yang memiliki *behaviour*. Kemudian, komponen perangkat lunak dapat dibangun dengan menerapkan *behaviour* ini (misalnya mengembalikan Id dan Nama Kode). Yang terpenting, *behaviour* ini (untuk sebagian besar) merupakan *context free*, misalnya suatu komponen yang mengembalikan Id Kode dan Nama Kode tidak tahu mengapa potongan-potongan informasi ini diperlukan dan juga tidak perlu tahu. Komponen ini hanya melakukan tugasnya untuk melayani Kode.

Oleh karena itu, pendekatan berbasis model untuk *system engineering* menghasilkan komponen yang dapat digunakan kembali, yaitu komponen yang tidak memiliki ketergantungan satu dengan yang lain dan kohesif. Sehingga, sistem tidak rapuh dan mudah untuk di- *maintenance* dan dikembangkan.

SDMX memiliki *Common Component Architecture* berdasarkan Model Informasi SDMX dan implementasi *open source* dari arsitektur tersebut. Arsitektur ini tersedia di SDMX Source.



Dataflow adalah komponen penting di dalam Model Informasi SDMX. *Dataflow* merupakan konsep yang digunakan untuk menyajikan dan melakukan diseminasi data. Hal tersebut memanfaatkan informasi struktural yang didefinisikan oleh *Data Structure Definition* (DSD), tetapi memungkinkan restriksi lebih lanjut yang akan ditentukan dan hanya digunakan untuk konten yang diizinkan ((**Valid**) **Content Constraint**).

Data Structure Definition (DSD) adalah struktur fundamental yang mendefinisikan konten valid dari kumpulan data di dalam dimensi, variabel, konsep, dan kontennya yang valid sebagai variabel (misalnya daftar kode atau tipe data lainnya).

Provision Agreement berisi informasi tentang penyediaan data oleh satu **Penyedia Data** untuk satu **Dataflow**. Dalam lingkungan pengumpulan data, hal tersebut dapat berisi tautan ke **Valid Content Constraint** yang selanjutnya membatasi nilai yang diizinkan dan dapat dilaporkan oleh Penyedia Data. Dalam lingkungan Diseminasi Data, hal tersebut dapat ditautkan ke *Registered Data Source* yang mengidentifikasi konten sumber data ((**Aktual**) **Content Constraint**) serta lokasi data dan bagaimana data dapat diambil (misalnya kueri SDMC).

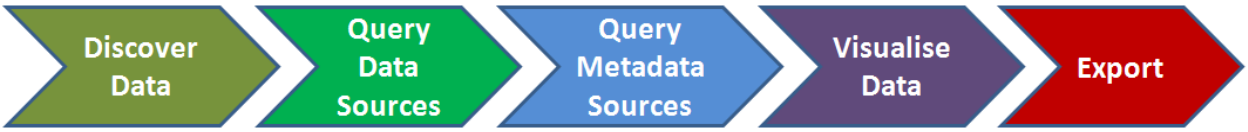
Tiap *Dataflow* dapat dihubungkan ke satu atau lebih **Kategori** dan setiap Kategori dapat dihubungkan ke nol atau lebih *Dataflow*. Koneksi ini mendukung pencarian data dengan topik yang terorganisasi seperti Demografi, Sensus, Kesehatan, maupun Keuangan.

1.2.3 Beberapa *Use Cases* yang Didukung oleh Model

Data Collection

Database Load

Diseminasi Data (2)



1.2.4 Dukungan SDMX untuk Pemrosesan

Proses	Konsep	Role
Mendaftarkan data yang tersedia	*Provision Agreement*; *Data Registration*.	• Pengumpulan data dapat diotomatisasi oleh penyedia data yang mendaftarkan lokasi data baru yang akan disajikan.
Menerima/Mengumpulkan data	*Provision Agreement*; *Data Registration*.	• Aplikasi dapat menginformasikan tentang data pendaftaran baru dan data tersebut dapat diperoleh; • Sebagai alternatif, data dapat dikirim secara langsung ke *data collector* (misalnya melalui e-mail, dll); • Terhadap dua kasus tersebut, data yang terkait dengan *Provision Agreement* akan membantu identifikasi yang dilakukan oleh penyedia Data.
Read dataset	*Data Structure Definition* serta Konsep dan Daftar Kode terkait.	• Data dapat bervariasi ke dalam berbagai format dan *reader* khusus untuk format ini tentu diperlukan; • *Reader* mungkin memerlukan akses ke *Data Structure Definition* untuk melakukan fungsi ini.
Data validation	*Data Structure Definition* serta Konsep dan Daftar Kode terkait; *Dataflow* dan *Content Constraint* terkait (valid). *Provision Agreement* dan *Content Constraint* terkait (valid).	• Memvalidasi data menggunakan spesifikasi DSD yang dibatasi oleh *Dataflow*. • Memvalidasi data menggunakan batasan (*constrain*) tambahan dari *Provision Agreement*.
Membuat tabel basis data	*Data Structure Definition*	• Tabel basis data dapat dibuat secara otomatis dari metadata di *Data Structure Definition*.
Transform/Map	*Data Structure Definition* serta Konsep dan Daftar Kode terkait; *Dataflow*	• Input data mungkin perlu dipetakan dalam hal dimensi dan/atau skema pengkodean yang digunakan.
Discover data	*Category Scheme*; *Concepts*; Penyedia Data; *Dataflow*	• Untuk memungkinkan pembangunan *high level data discovery* yang memungkinkan pengguna untuk menelusuri topik data secara luas dari data yang penting (Dataflow).

1.3 Struktur SDMX

1.3.1 Terminologi

Maintainable

Semua struktur yang di- *submit* atau dijadikan kueri dari registri adalah struktur yang *maintainable*. *Maintainable* mungkin saja berisi sub-struktur, tetapi *maintainable* adalah tingkatan tertinggi dari *containership*. *Maintainable* tidak dapat digunakan ke dalam jenis struktur lain dan strukturnya tidak dapat di- *submit* ke registri kecuali mereka adalah *maintainable* atau didefinisikan ke dalam *maintainable*. Hal ini mungkin merupakan pernyataan yang agak kompleks dan akan lebih mudah jika dijelaskan menggunakan contoh secara nyata:

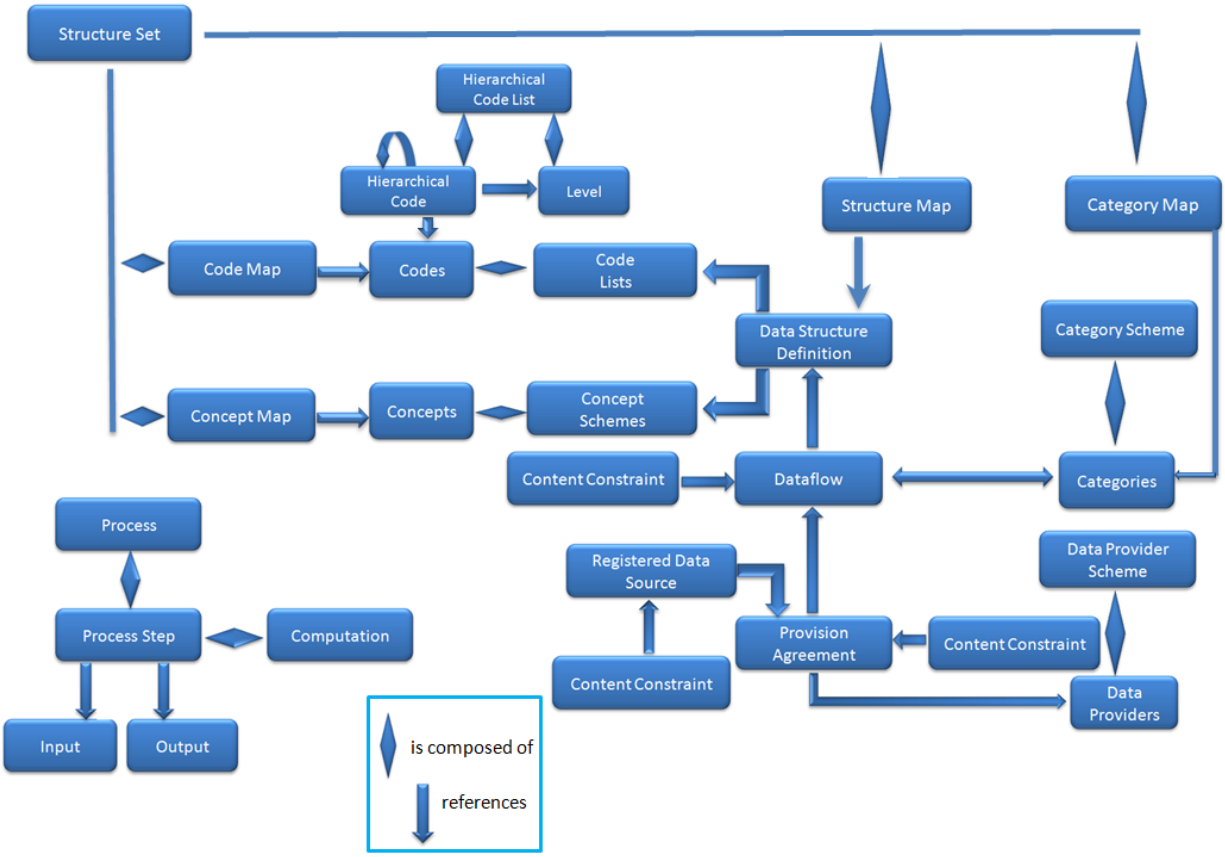
- Daftar kode adalah struktur yang *maintainable* dan dengan demikian dapat di- *submit* ke *Fusion Registry*;
- Daftar kode dapat berisi kode yang tidak *maintainable*.
- Daftar kode tidak memiliki tipe struktur *parent* dan tidak dapat berisi Daftar Kode lainnya;
- Kode tidak dapat di- *submit* ke registri dengan sendirinya. Mereka harus didefinisikan ke dalam Daftar Kode;

Jadi, struktur yang *maintainable* dapat dianggap sebagai wadah informasi. Informasi tidak bisa di- *maintain* di luar *parent*-nya yang *maintainable*, maka istilah itu disebut *maintainable*.

Struktur yang *maintainable* memiliki referensi ke *maintenance agency* yang bertindak sebagai 'pemilik' struktur. Hal tersebut merupakan *mandatory version*, secara bawaan menjadi 1.0 dan *mandatory identifier*. Kombinasi dari jenis struktur, Id agensi, Id, dan versi dapat digunakan untuk mengidentifikasi struktur yang *maintainable* secara unik di SDMX.

Semua jenis struktur yang *maintainable* dalam SDMX tercantum di bawah ini:

- *Skema Kategori*
- *Kategorisasi*
- *Daftar Kode*
- *Skema Konsep*
- *Constraint*
- *Data Structure Definition (DSD)*
- *Dataflow*
- *Daftar Kode Hierarkis*
- *Metadata Structure Definition (MSD)*
- *Metadataflow*
- *Skema Organisasi*
- *Reporting Taxonomy*
- *Proses*
- *Structure Set*
- *Provision Agreement*



Menampilkan model UML dari Tipe Struktur SDMX.

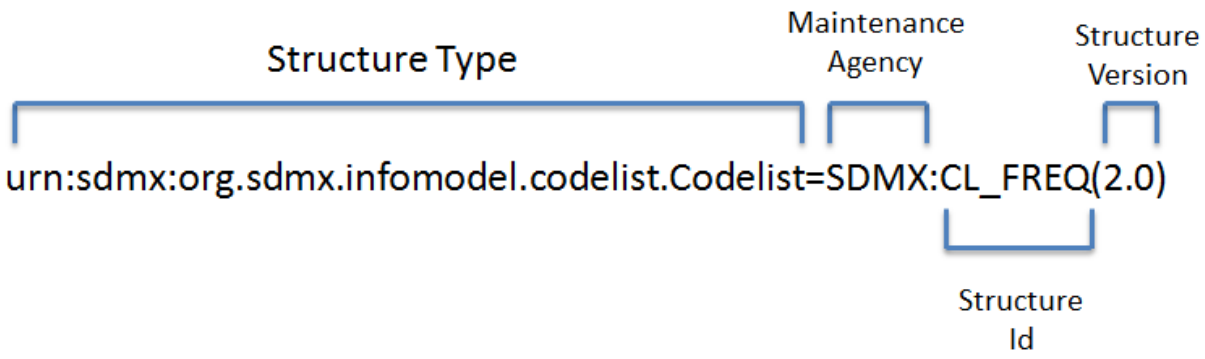
Identifiable

Struktur yang *identifiable* adalah struktur yang dapat diidentifikasi secara unik di dalam SDMX dengan menggunakan URN.

URN (*Uniform Resource Name*) adalah bentuk khusus dari URI (*Uniform Resource Identifier*). URI adalah serangkaian karakter yang digunakan untuk mengidentifikasi nama dari *web resource*. URL digunakan untuk mengidentifikasi artefak SDMX tertentu secara unik.

SDMX mendefinisikan sintaks URN dan setiap URN dibangun dengan menggabungkan jenis struktur, lembaga pemeliharaan, versi struktur, dan id struktur.

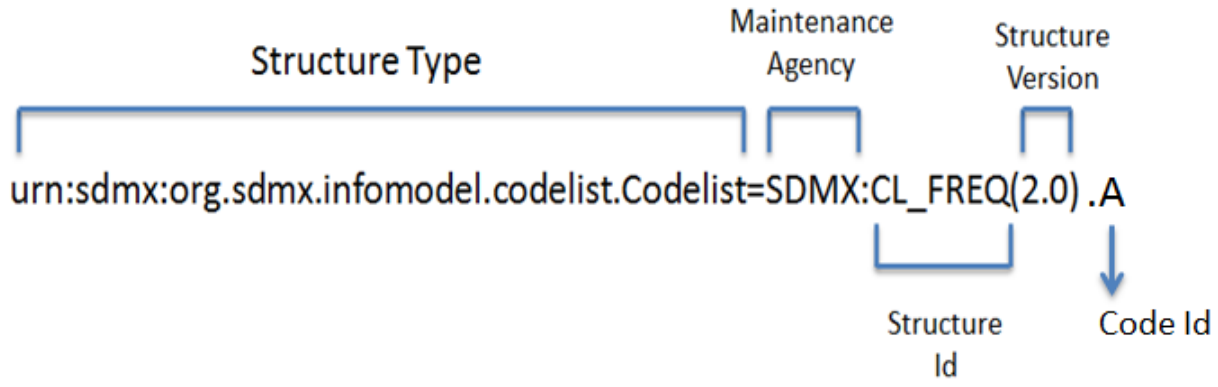
Contoh URN untuk Daftar Kode diberikan di bawah ini:



Memperlihatkan sintaks URN dengan struktur yang *maintainable*.

Semua struktur yang *maintainable* adalah struktur yang *identifiable*, karena semuanya memiliki jenis struktur, id agensi, id, dan versinya masing-masing. Ini berarti setiap struktur yang *maintainable* dapat diidentifikasi secara unik di dalam SDMX menggunakan sintaks URN.

Ada juga struktur yang dapat diidentifikasi di dalam SDMX tetapi tidak termasuk *maintainable*. Jika struktur tersebut termasuk *identifiable*, maka pasti memiliki *mandatory* id. Jika struktur tersebut tidak *maintainable*, maka ia harus selalu berada di dalam cakupan *maintainable*. Oleh karena itu, *maintainable parent* digunakan untuk bisa mendapatkan URN. Contoh *identifiable* yang tidak termasuk *maintainable* adalah Kode yang berada di dalam Daftar Kode. Gambaran sintaks URN untuk sebuah kode ditunjukkan di bawah ini. Cukup sederhana untuk melampirkan/memasangkan id kode ke akhir Daftar Kode dari URN.



Memperlihatkan sintaks URN untuk struktur yang *identifiable*.