Data Integration

Data integration: process of integrating data from multiple sources to obtain a single view over all sources.

- Integration can be:
  - Virtual: keep data in original sources, and have external keys/identifiers to link individual records across the data sources
  - Physical: copy data into one source/location and perform data integration in that single source

Reasons for data integration:

- Refuse data from various legacy databases and systems
- Reconcile different points of views adopted by different systems
- Integrate external data

Main challenge of data integration: Heterogeneity

- At differential levels: source type, schemas, data types, data values, semantics

Three main tasks of data integration:

1. **Schema mapping and matching**
   - Identify which attributes or attribute sets across database tables contain the same type of information (corresponding columns)
   - Analyse attribute names, not the attribute content

2. **Record linkage / data matching / entity resolution**
   - Identify which records in one or more databases correspond to the same real-world entity
   - Analyse content of attributes
   - A special case is deduplication (or duplicate detection) in a single database

3. **Data fusion**

- Merge pairs or groups of records that correspond to the same entity into one clean, up-to-date and consistent record that represents the entity
- Issues: spelling variations, incorrect values

<u>Example: Woo (Web of Objects)</u>
Aim: To enable various products in Yahoo! to synthesis knowledge-bases of entities relevant to their domains

- Knowledge graph of real-world entities which are connected based on their relationships

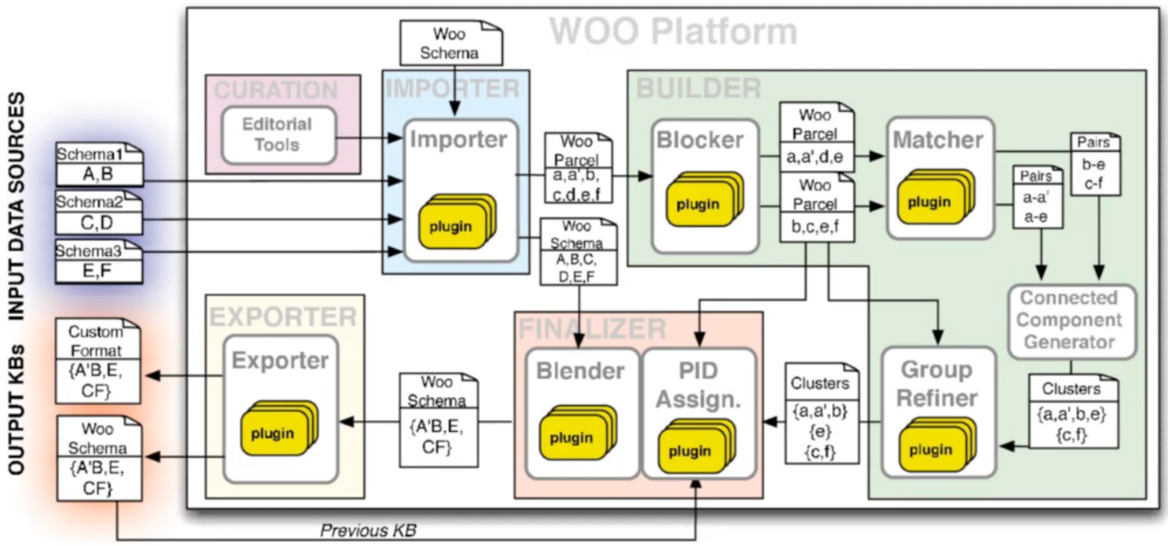
Requirements:

- **Coverage**: the fraction of real-world entities
  - High coverage - knowledge graph/search engine - need a large fraction of what is in the real world to be represented
- **Accuracy**: information must be accurate
  - Data must be accurate (data quality) - no point in having incorrect information
  - Difficult to assess whether something is accurate without looking at the source of the data itself
- **Linkage**: the level of connectivity of entities
  - Basic information that is not linked with entities or other information, then the search engine is useless
  - Provide better experience for consumers
- **Identifiability**: one and only one identifier for a real-world entity
  - Every unique entity should have a unique identifier
  - Need to query the entity using this identifier
- **Persistence/ content continuity**: variants of the same entity across time must be linked
  - There has to be a way to look at the past of an entity - continuity of information has to be correctly linked

    - Concept of entity should continue over time even when characteristics of it change
- **Multi-tenant**: be useful to multiple portals
    - If we have a large database of entities, it needs to be possible that different applications/web portals can use the database (extract data into their own databases)
    - Should facilitate providing data to other systems

Knowledge-base synthesis is the process of ingestion, disambiguation and enrichment of entities from a variety of **structured** and **unstructured** data sources. Enrich data from various sources.
Challenges in this kind of system:
- Sheer scale of the data → hundreds of millions of entities daily
- Diverse domains → from hundreds of data sources
- Diverse requirements → multiple tenants, such as locals, movies, deals and events in the Yahoo website

Woo Architecture



Importer	Takes a collection of data sources as input (such as XML feeds, RDF content, relational databases, or other custom formats) • For each source, there will be a plug in that allows the data to be imported and convert them to a specific internal structure called 'Woo schema' • Each data source is converted into a common format called *WOO schema* • The *WOO Parcel*, containing only the attributes needed for matching, is pushed to the Builder ○ Create WOO parcels for each of the entities in the system that only contains attributes strictly needed for matching - more compact representation of whole data ○ Ineffective to pass all information in this architecture ○ Only contains attributes strictly required for matching - much more compact than passing huge amounts of data ○ Large information about entities - inefficient to pass all information from one model to another within this architecture.
Builder	Performs the entity deduplication (record linkage/entity resolution) and produces a clustering decision. It includes the subfaces: (1) Blocker, (2) Matcher, (3) Connected Component Generator, and (4) Group Refiner. • ***Blocker***: Very expensive - cannot compare every record in one source with all records in another source. To combat quadratic complexity, we use blocker / blocking ○ Identifies smaller blocks of data which contains information about entities, which may be the same entity • E.g., Group records with same postcodes • Further processing is only done within these blocks ○ Blocking reduces the complexity of comparing all records with one another

<table>
<tr><td></td><td>

Matcher: Software which compares individual records and calculates similarities between them

Output: pairs of records which have similarities
Combination of ML and rule-based algorithms

Connected Component Generator: Clustering process, where connected components are computed to generate clusters of entities that are highly likely to be matched

Use calculated similarities in previous step to generate clusters of entities that are very likely to be matched
Input: pairs of records and calculated similarities
Output: clusters built on similarities

Grouper Refiner: (Optional stage) can further refine clusters (large clusters) into smaller clusters

Output: set of clusters

</td></tr>
<tr><td>Finalizer</td><td>
Responsible for handling the persistence of object identifiers and the blending (fusion) of the attributes of the (potentially many) entities that are being merged

Input: set of clusters
Output: WOO schema (all combined records)

Two subfaces: PID Assignment and Blender
PID Assignment: keeps continuity of content/entity that are being matched with the algorithms in clustering

Keeps track of how they evolve over time

Blender: full entities are being fused/blended together according to a defined set of functions

Get all the information for record pairs in order to combine them together into a single record

</td></tr>
<tr><td>Exporter</td><td>
Generates a fully integrated and de-duplicated knowledge-based, either in a format consistent with the WOO schema or in any custom format

Exports the combined data into certain formats required for different systems
Output: Creating a fully integrated and deduplicated knowledge graph of integrated data that is meaningful

</td></tr>
<tr><td>Curation</td><td>
Enables domain experts to influence the system behaviour through a set of GUIs, such as forcing or disallowing certain matches between entities, or by editing attribute values

Set of editorial tools that enable domain experts to influence the system behaviour manually

Manual alterations

Influence based on their domain expertise and knowledge
Want to ensure the WOO architecture produces high quality output

</td></tr>
</table>

Schema Mapping and Matching

Schema matching problem: generating correspondences between elements of two database schemas
- Difficult with large, complex databases
- Schema matching tries to solve this problem

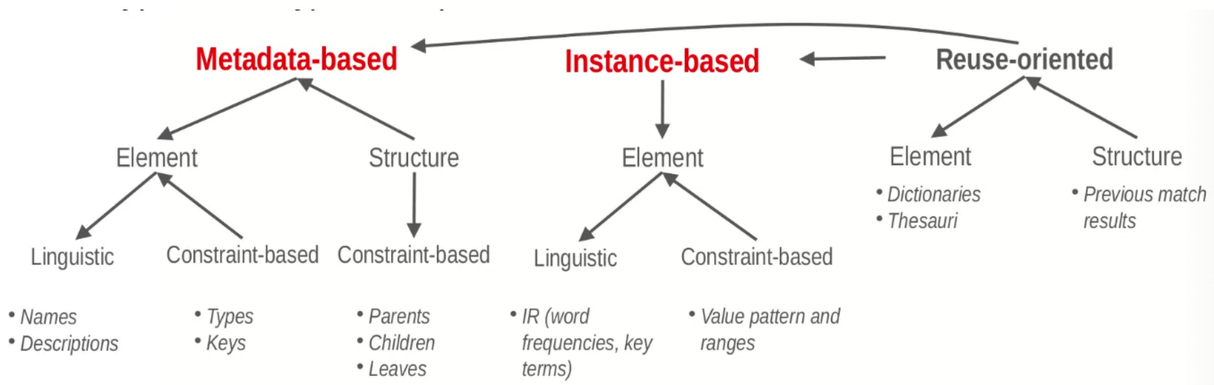
Basic input to schema matching techniques:
- schema structures;
- element (attribute) names; and
- constraints, such as data types and keys.

Other inputs to basic schema matching:
- Synonyms
    - Code = Id = Num = No
    - Zip = Postal [code]
- Acronyms
    - PO = Purchase Order
    - UOM = Unit of Measure
    - SS# = Social Security Number
- Data instances (attribute values)
    - Key insight: Elements match if they have similar instances or value distributions

Many applications need correspondences:
- **Data translation**
    - Object-to-relational mapping
    - XML message translation (e.g., between different applications) - applications may exchange data using XML schemas
    - Data warehousing loading (ETL) - integrate data and ensure it is clean and consistent, integrating different databases into the same warehouse

- **Data integration**
    - ER (entity relationship) design tools
    - Schema evolution (temporal changes) - new version of database system, new regulations, merging companies or departments
    - Record linkage

<u>Taxonomy of Automatic Match Techniques</u>
- Matcher combinations are either *hybrid* matches (e.g., that consider name and type similarity), or *composite* matches
- Metadata-based: only look at the attributes and structure of databases
- Instance-based: look at the content of the databases
- Reuse-oriented: previously matched databases (e.g., dictionary book from previous databases)
    - Past information can be fed into both metadata-based and instance-based matches



Schema Matching Techniques

- ***Linguistic matchers***
  - Names of attributes
  - (String) similarity of concept/element names
  - Based on dictionaries or thesauri, such as WordNet/ UMLS

- ***Structure-based matcher***s
  - Consider similarity of ancestors/descendants
  - Graph-based matching such as *Similarity Flooding* (Melnik et al., ICDE 2002)

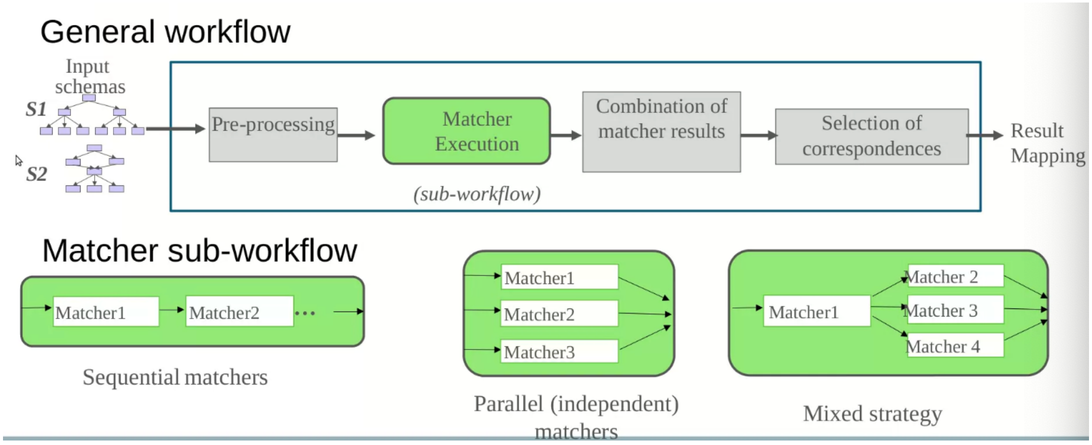
- ***Instance-based matchers***
  - Record values
  - Concepts with similar instances/annotated objects should match
  - Consider all instances of a concept as a document and utilise document similarity (such as TF-IDF) to find matching concepts

Instance-based ontology matching

- Concepts with most similar instances should match (requires shared/ similar instances for most concepts)
  - Structured trees that have concepts - each node corresponds to concept. Find concepts that have the most similar instances
  - Find elements which are in similar concepts which have similar descriptions
  - Look at instances (individual nodes/concepts) of two ontologies
- Mutual treatment of entity resolution (instance matching) and ontology matching
- Promising for link discovery in the Linked Open Web of Data
  - Identify links between similar objects and different websites/databases - related to WOO

Schema-matching is a multi-step process

- Input to schema matching: a set of schemas
  - E.g., simple database schemas, ontology (concept descriptions)



Matcher Execution (sub-workflow)	○ Different ways to do this, often have more than one matcher ○ Similarity-based matcher, instance-based matcher and concept-based matcher ○ Can run the matchers sequentially or independently (parallel), or a mixed strategy • Sequential manner: the output of one matcher may flow into the second matcher • Mixed: initial matcher, and results flow into a set of other matchers
Combination of matcher results	○ Information about attributes in different databases/ concepts in sub-trees ○ May have contradicting results ○ Must combine into a single result

Selection of correspondences	○ Best correspondences ○ Good coverage is important - high number of correspondences - good mapping • Can use domain knowledge to do more matching ○ May not be able to identify correspondences between all concepts

Issue: Large-scale matching

- Very large ontologies/ schemas (>10,000 elements)
    - Quadratic complexity of evaluating the Cartesian product (match efficiency)
        - Cannot compare every element/ concept
        - Have to do blocking
    - Difficult to find all right correspondences (match quality)
        - Match quality will decrease as correspondences increase - more links between matches
        - Have to involve user - semi-automatic way to match
    - Support for user interaction

- Many (>>2) ontologies/ schemas
    - Holistic ontology/ schema matching
        - May have to mix record linkage approach with schema matching approach
        - Schema matching (identify correspondences) then do linking - iteratively to improve overall linkage
    - Clustering of equivalent concepts/ elements or linking to some hubs
        - Hub = one core database/ ontology to which we add or map other smaller databases/ ontologies

Schema matching often requires user input - often not done fully automatically. User has to decide matchers and their order.

<u>Self-Tuning Match Workflows</u>

- ***Semi-automatic configuration*** - user input required
    - Selection and ordering of matchers
    - Combination of match results
    - Selection of correspondences (top-k, threshold, …)

- ***Prototype tuning frameworks*** (Apfel, eTuner, YAM)
    - Use of supervised machine learning
        - Training data from earlier schema-matching - can be used for related databases
    - Need previously solved match problems for training
        - Need large training dataset - time consuming to build or validate
    - Difficult to support large schemas

- ***Heuristic approaches***
    - Use linguistics and structural similarity of input schemas to select matchers and their weights
        - Assign weights to different matchers depending on their quality and appropriateness
    - Favour matchers that give higher similarity values in the combination of matcher results
    - Often, user with expertise in understanding the matcher technology and the domain will choose.

- ***Rule-based approach***
    - Comprehensive rule set to determine and tune match workflow
    - Rules often have to be developed manually using domain expertise
    - Generally, developing rule-based learning approaches/systems are more time consuming and difficult than ML-based, probabilistic approaches
    - Use of schema features and intermediate match results