INFORMATION INTEGRATION

Diego Garcia-Olano, Elsa Mullor

OUTLINE

1. What IS Information Integration?
2. The Heterogeneity Problem
3. Distributed Architectures
4. Schema Generation
5. Unstructured Data
6. Yahoo Pipes
7. Open Refine
8. II as it pertains to Open Data
9. Scary Exercises

WHAT IS INFORMATION INTEGRATION

1. WHAT IS INFORMATION INTEGRATION

Information Integration

is the process of taking several databases

or other information sources and

making the data in theses sources work together

as if they were a single database.

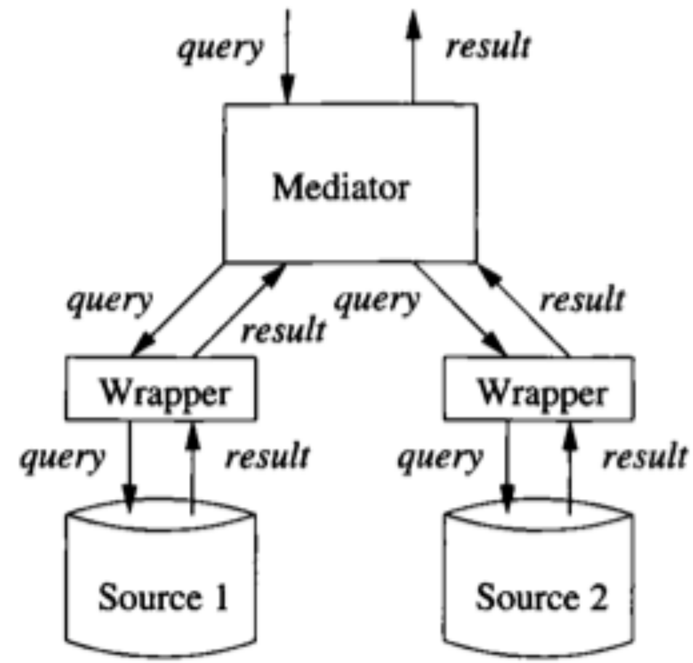
1. WHAT IS INFORMATION INTEGRATION

The integrated database may be

- **physical (a "warehouse")** or
- **virtual (a "mediator**" or "middleware”) that may be queried even though it does not typically exist)

Sources may include conventional databases or other types of unstructured information, such as collections of web pages.

1. WHAT IS INFORMATION INTEGRATION



Even similar databases

can embody conflicts that are

hard to resolve, & the solution

Lies in the design of **wrappers**

which are translators between

the schema and data values at a source and

the schema and data values at the integrated DB.

1. WHAT IS INFORMATION INTEGRATION

Mediator (virtual) systems

can be divided into two classes:

1) “global-as-view” (GAV) where

the data at the integrated database is defined by how it is constructed from the sources and

2) “local-as-view” (LAV) where

the content of the sources is defined in terms of the schema that the integrated database supports.

1. WHAT IS INFORMATION INTEGRATION

Some reasons why you'll need it:

1) Databases are created independently even if they later need to work together

2) The use of databases evolves so we cannot design a DB to support every possible future use.

The need to use and support legacy data sources.

1. WHAT IS INFORMATION INTEGRATION

Example:

Incorporating many different databases in a school system to be able to share information.

Use **middleware**, a layer of abstraction on top of all legacy databases, which allows them to serve their current applications

This could be relational views – materialized or virtual , and then SQL can be used query to this layer (or XML/Xquery)

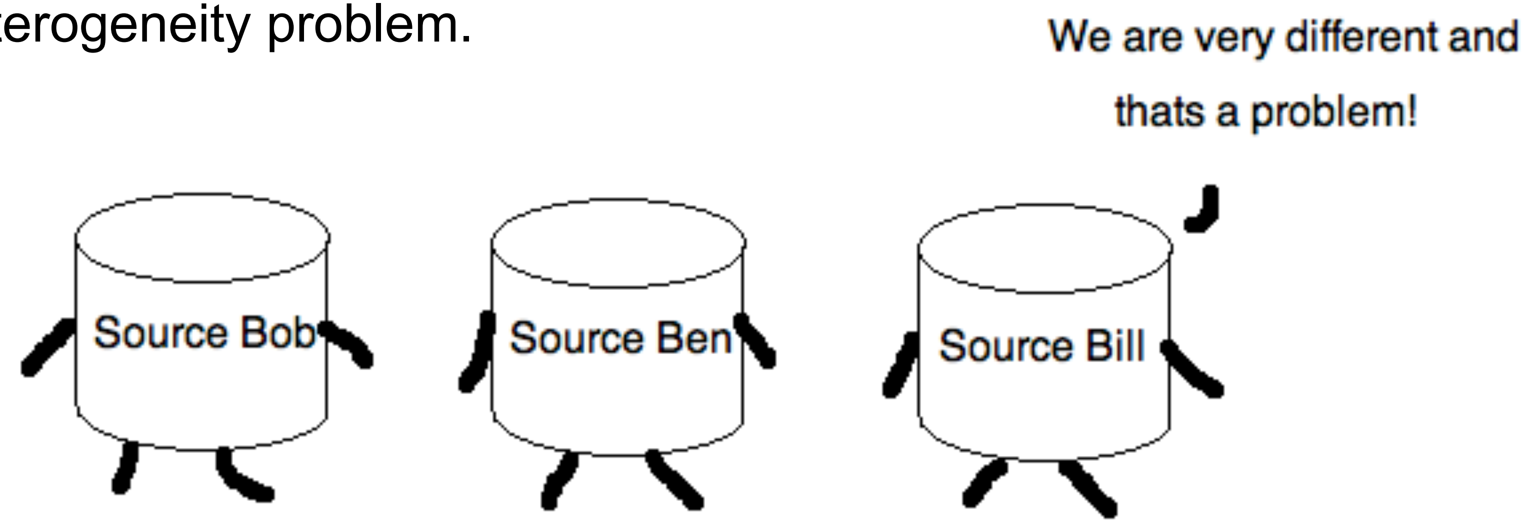
This middleware is often defined by a collection of classes and queried in an object oriented language.

As a result new applications can be written to access this middleware while legacy apps continue to use legacy db.

HETEROGENEITY PROBLEM

2. HETEROGENEITY PROBLEM

When we try to connect information sources that were developed independently we invariably find that the sources differ in many ways, even if they are intended to store the same kinds of data. Such sources are called heterogeneous, and the problem of integrating them is the heterogeneity problem.



2. HETEROGENEITY PROBLEM

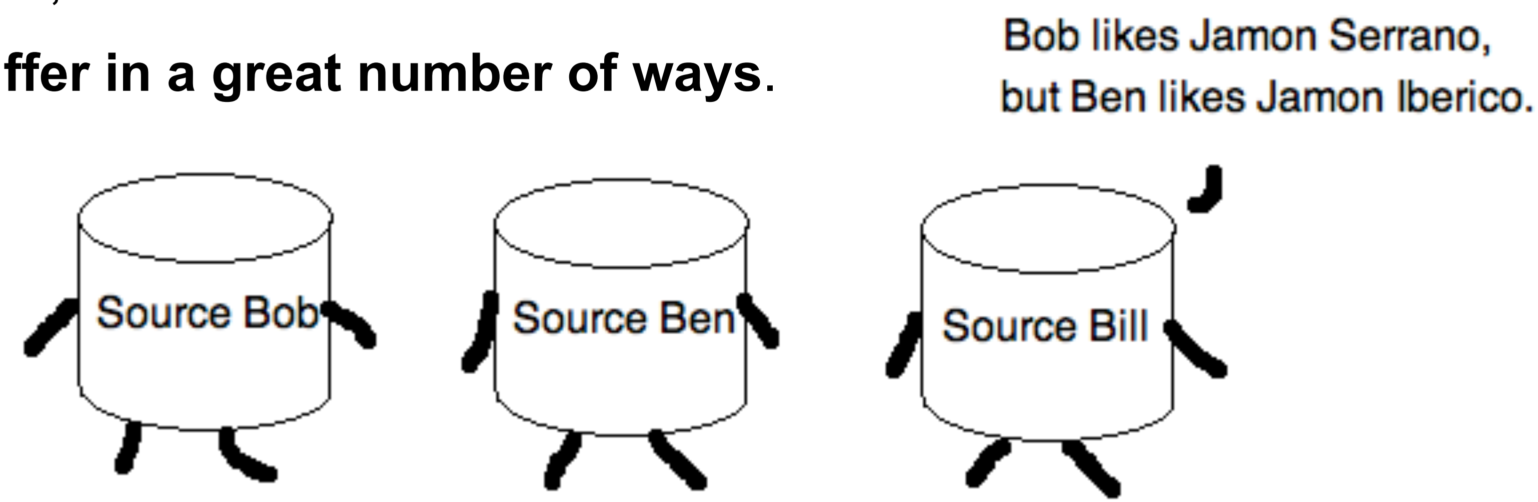
A car company has 1000 dealers, each of which maintains a database of their cars in stock.

The company wants to create an integrated database containing the information of all 1000 sources that will help dealers locate a particular model at another dealer, if they don't have one in stock.

It also can be used by corporate analysts to predict the' market and adjust production to provide the models most likely to sell.

However, the dealers' databases

may **differ in a great number of ways**.



2. HETEROGENEITY PROBLEM

Communication Heterogeneity

Bob allows access to his dealership information using HTTP whereas

Ben only accepts remote accesses via remote procedure calls or anonymous FTP.

Query-Language Heterogeneity

The manner in which we query or modify a dealer's database may vary.

The DB may or may not accept SQL queries and modifications. Bill doesn't.

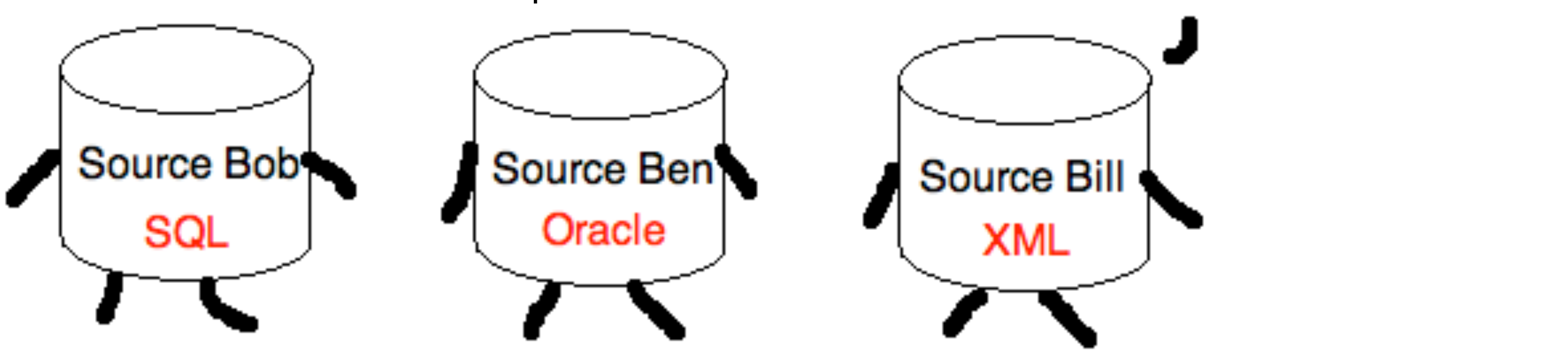
Schema Heterogeneity

Even if they all use relational DBMS supporting SQL as the query language, their schemas can differ.

Bill and Ben call them "Cars", but Bob calls them "Autos". Seriously.

Ben doesn't keep track of the date when he sells a car at all!

Missing schema elements are a common problem.



2. HETEROGENEITY PROBLEM

Data type differences

Bob stores serial numbers as strings of length 64, whereas Ben only stores the first 32 characters

Ben stores them as a long numeric value just because he can.

Value Heterogeneity

The same concept might be represented by different constants at different sources.

The color black is stored by Bob as “black”.

Ben and Bill both use the value “BL”, but for Bill that means the car blue!

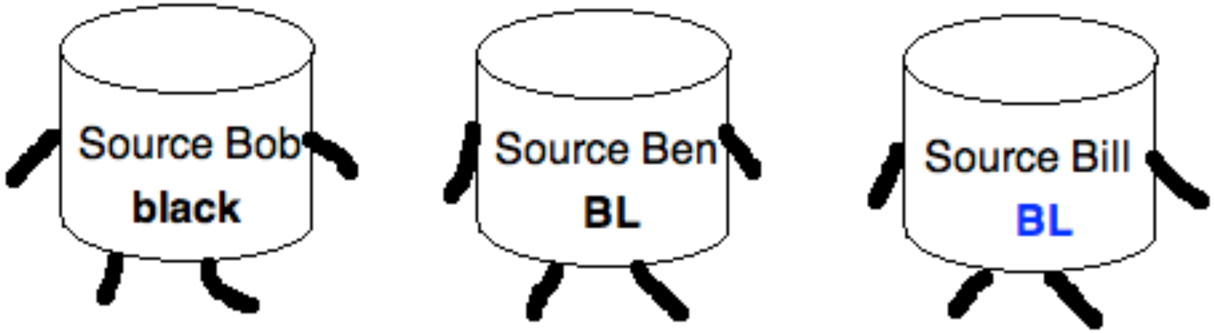
Semantic Heterogeneity

Terms may be given different interpretations at different sources**.**

Bill is from Texas and includes “Trucks” in the Cars relation whereas neither Bob or Ben do.

Bob is from Barcelona and includes “Motos” in the Cars relation.

.



3 – DISTRIBUTED ARCHITECTURES

DDBS/DBMS

DDBS: Distributed DB system :

Collection of multiple databases, **distributed** over a computer network

DBMS: Distributed DB Management System :

software systems. Management of DDBS, makes distribution transparent to user

More reliable, more **responsive**, process better, cope with **large scale** data management problems (divide & conquer)

+ Economically better: add processing and storage power

TRANSPARENCY

Autonomy: Control distributed, individual can operate independently (change Data structure, exchange data, execute transactions)

Data **independence**: user can change organization of data (logical: change schema/ physical: change structure).

Data **localization**: each site handles only a portion of data

⇒Less contention CPU, I/O services

⇒Less remote access delay (wide area networks)

Network Transparency.

TRANSPARENCY

Replication: partition/fully replicated/partially replicated

Distributed DBMS improve **reliability** : replicated components eliminates single-point failure.

Replication transparency: Who handle **management of copies**.

$\Rightarrow$Have to propagates update (even if failure/recovery)

Fragmentation transparency: **translation** from global query to several fragmented queries.

Reliability/availability: implementation of **consistency**

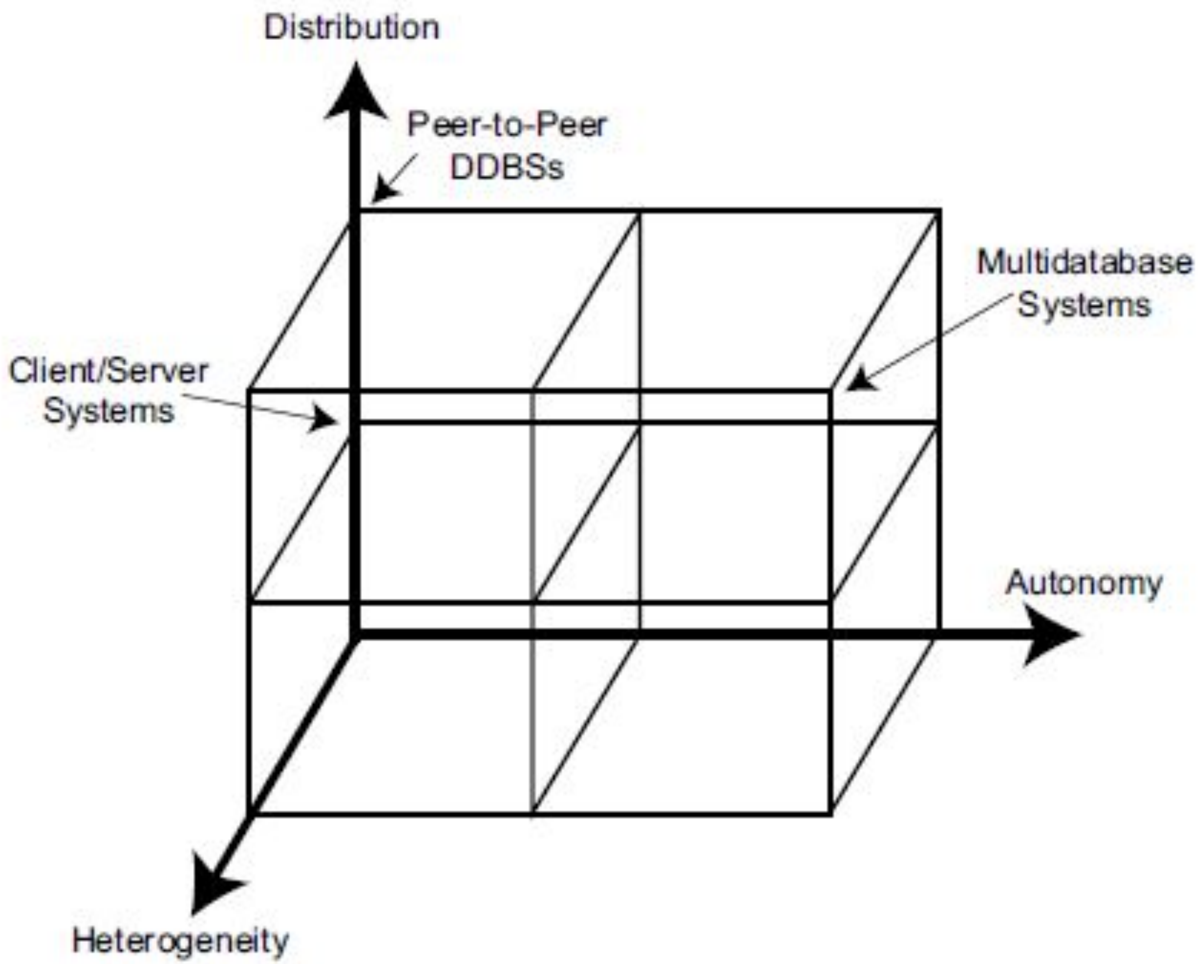


Fig. 1.10 DBMS Implementation Alternatives

FEDERATED DATABASE

One-to-one connections between all pairs of sites

Any site can query any other one => n*(n-1) connections (query translation)

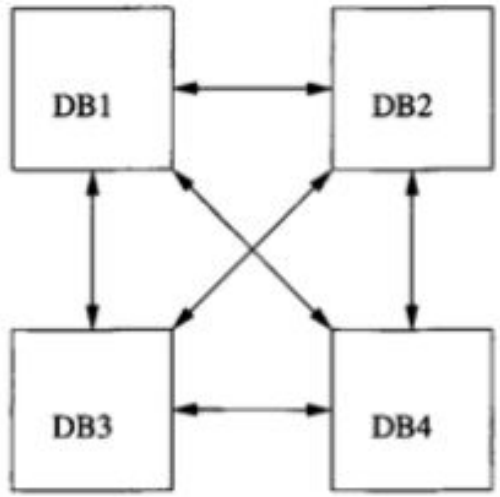


Figure 21.1: A federated collection of four databases needs 12 components to translate queries from one to another

Easiest to build when communications between DB limited

MDBS

Individuals DBMSs **fully autonomous**

+ **No** concept of **cooperation**

Global Database : **Union** of some **parts** of **local DB**

GCS = conceptual view of SOME of local DB <- Integration of LCSs or LESs (**bottom-up**)

MDBS layer: added on top of DBMS: facilitates acces to various DB for user.

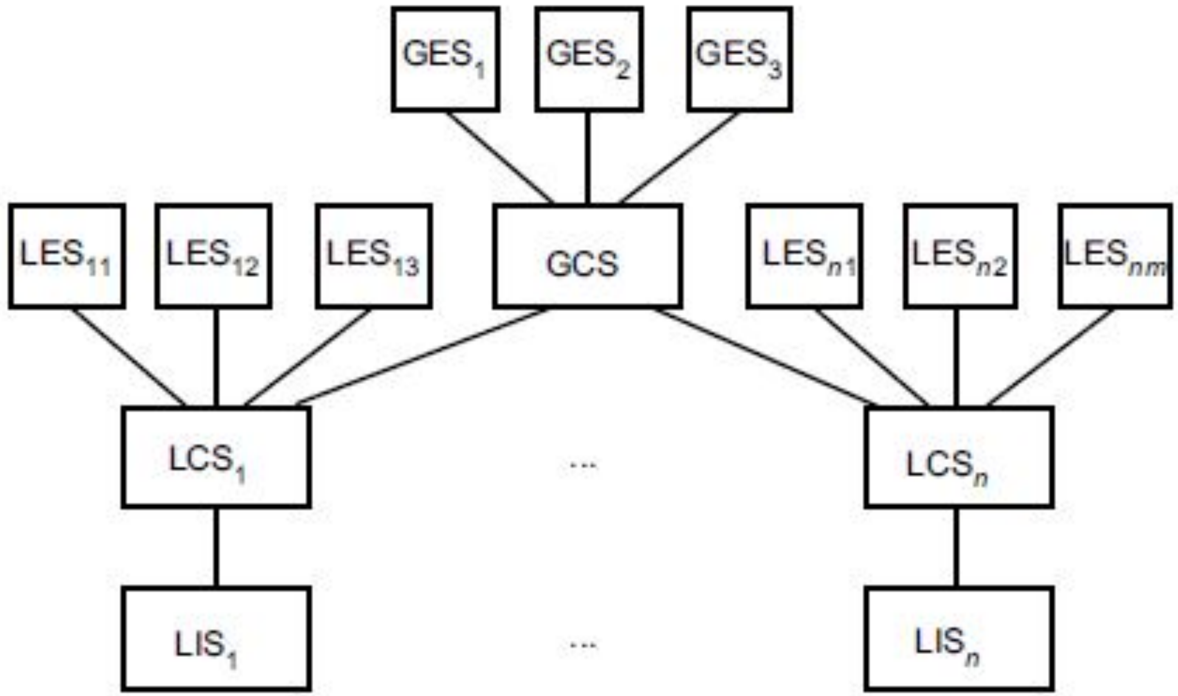


Fig. 1.16 MDBS Architecture with a GCS

MEDIATORS/WRAPPERS

Mediator: exploit knowledge of data for higher layers applications. Performs specific functions with defined Interfaces.

Each module in the MDBS layer is a Mediator.

Mediator implement GCS + handles User queries over it.

Collection of Mediators could be seen as Middleware

Wrappers: provide mapping between sources DBMS and mediator's views.

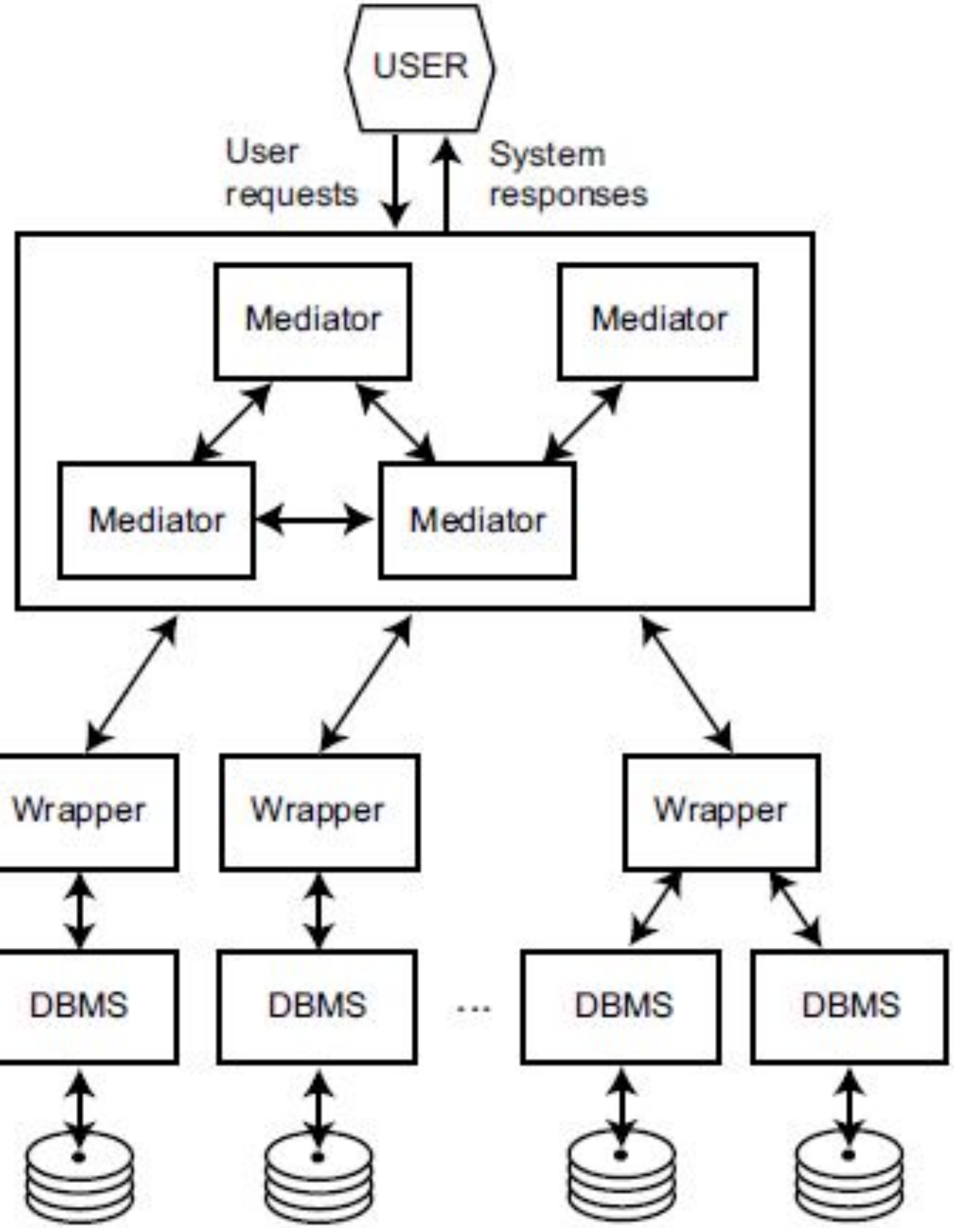


Fig. 1.18 Mediator/Wrapper Architecture

DATA WAREHOUSING

Source DB integrated in Global DB: **Data Warehouse**

Physical integration: gather data from local DB and **materialize** them in Global integrate DB.

Updates on local operational DB propagated to DW.

Support Decision Support Application (OLAP).

Query not directly over Distributed Database but over **DW**.

+ performances, data process (cleaning…)

- Not up-to-date data, manage updates, large-scale database

DATABASE INTEGRATION

Logical Integration: GCS **virtual**, not materialized.

Local systems: high autonomy + heterogeneity

Queries to Global Schema have to be **decomposed and translated to local** operational Databases.

Difficult Global updates (read-only).

Underlying data sources do not have to be databases.

+ up-to-date data, easy to add site, large-scale, distributed effort

- performances, query translation, heterogeneity of sites

PEER TO PEER

High heterogeneity, high autonomy, massive distribution.

Fully distributed: **no distinctions**. Each one has full DBMS functionalities

Local Internal Schema (data organization) can be **different** at each site.

Local conceptual Schema (LCS) supplementary layer

Global Conceptual Schema (GCS) = **Union of LCSs**

External Schema (ES) : support user applications + user access to DB.

PEER TO PEER

User Processor: interactions with Users

User Interface Handler : User command + results

Semantic Data Controller : Integrity constraints + authorizations (GCS)

Global Query Optimizer:Translate queries (global to local)+Minimize cost

Distributed Execution Monitor : Coordinates request

Data Processor : Storage

Local query optimizer: Choose best access path

Local Recovery manager : Consistency

Run-time Support Processor : Data Access

Expect to find **Both Data and User Processor on each machine**.

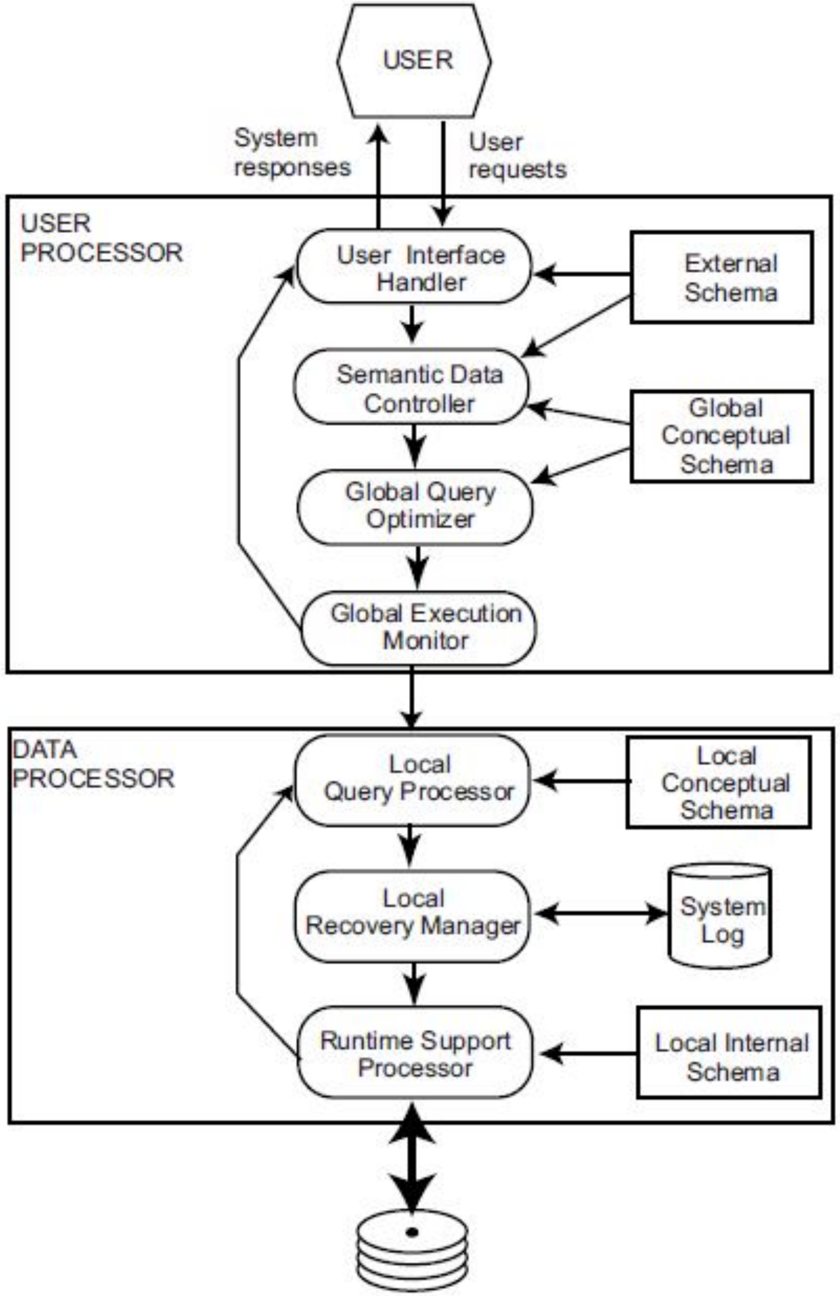


Fig. 1.15 Components of a Distributed DBMS

4 – SCHEMA GENERATION

SCHEMA GENERATION

Integrate Local to Global (**Physical** Vs **Logical**)

GCS = Union **subset of LCSs**

GCS defined first (**up-front**) then LCSs mapped to it. (**DW**)

Or define as integration of LCSs (**bottom up**) (**database Integration**)

Schema **translation**: Common representation. Only if heterogeneity

Schema generation : Matching / Integration / Mapping

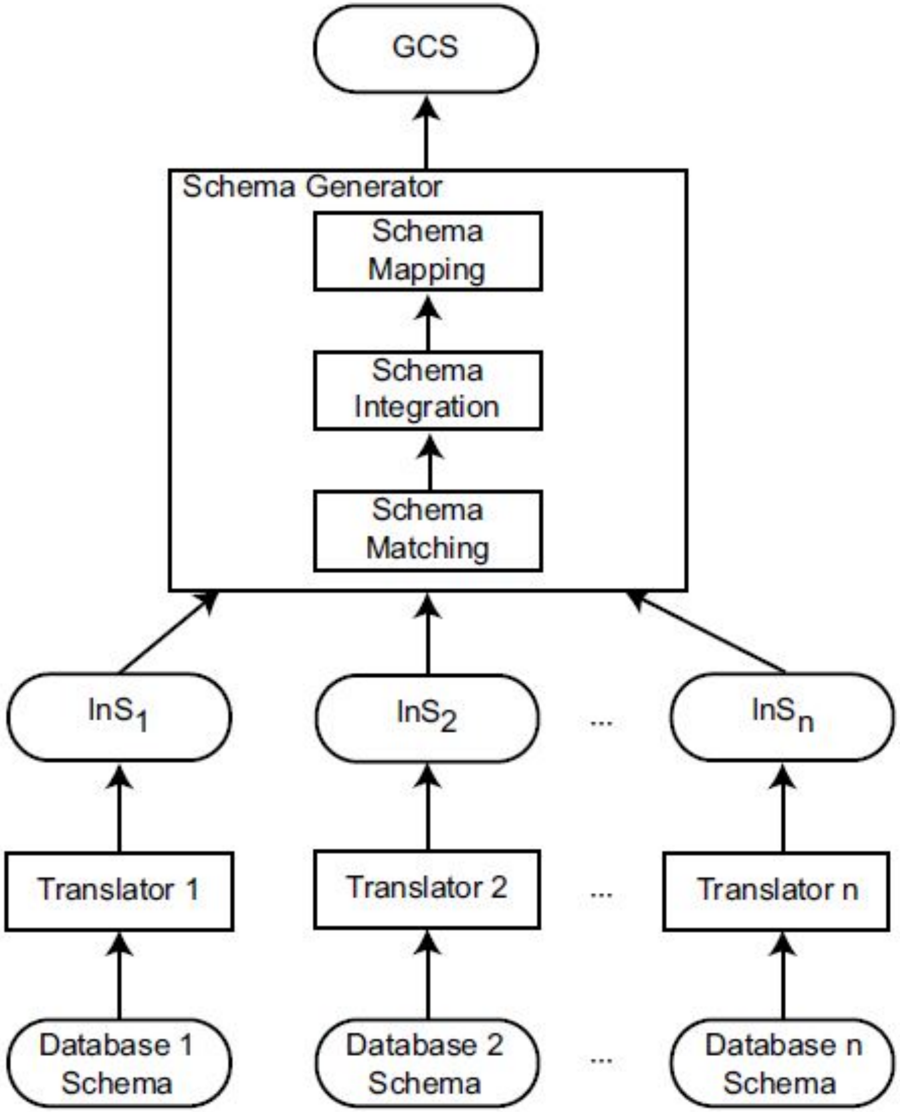


Fig. 4.3 Database Integration Process

LAV/GAV/GLAV

How elements of **GCS derived** from elements of **LCSs**:

LAV: Local-as-view: GCS definition exists. LCS as views over it.
-> query constrained on info in LCS (GCS may be richer)

Add new site easier

GAV: Global-as-view: GCS defined as view over LCS -> query constrained objects in GCS (local DBMS may be richer)

Query translation simpler

GLAV: Global-local-as-view: Combination of both

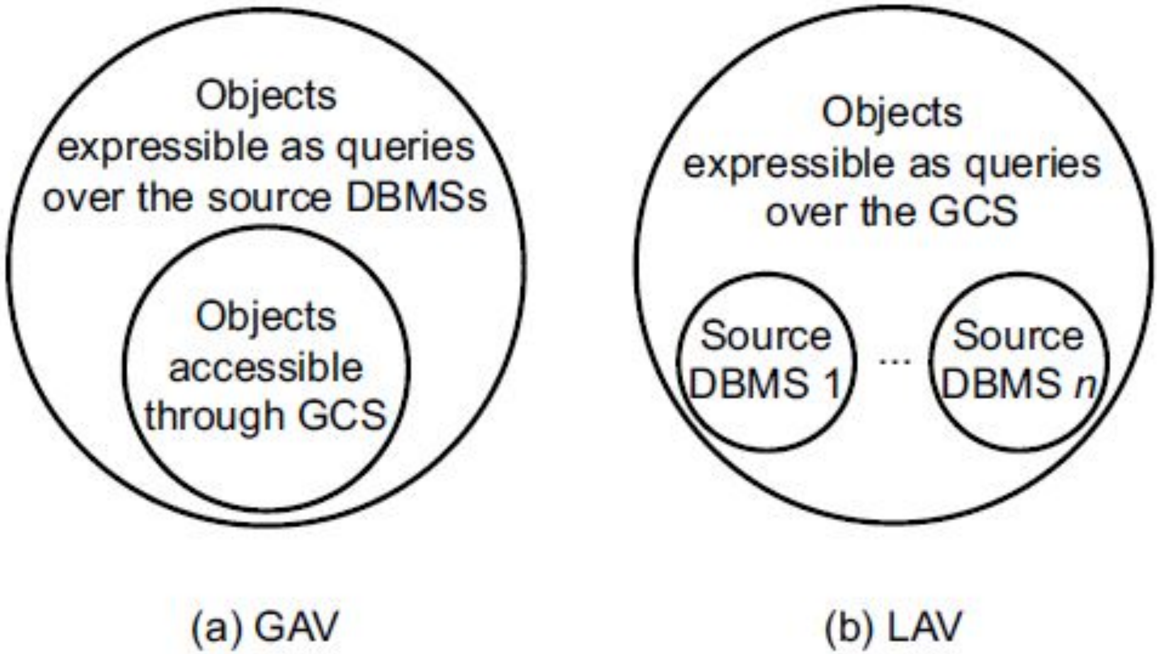


Fig. 4.2 GAV and LAV Mappings (Based on [Koch, 2001])

MATCHING

LCS to LCS (or GCS if exists)

Set of **rules r=<correspondence, predicate, similarity value>**

Schema integration or Instance based (peer to peer)

Ex: *Linguistic matching (names synonyms, homonyms, information retrieval techniques, affixes, n-grams, distances…)*

Structural similarities (similar concepts in neighborhood, PK/FK relationship),

Learning based (machine learning)

Combined (hybrid or composite)

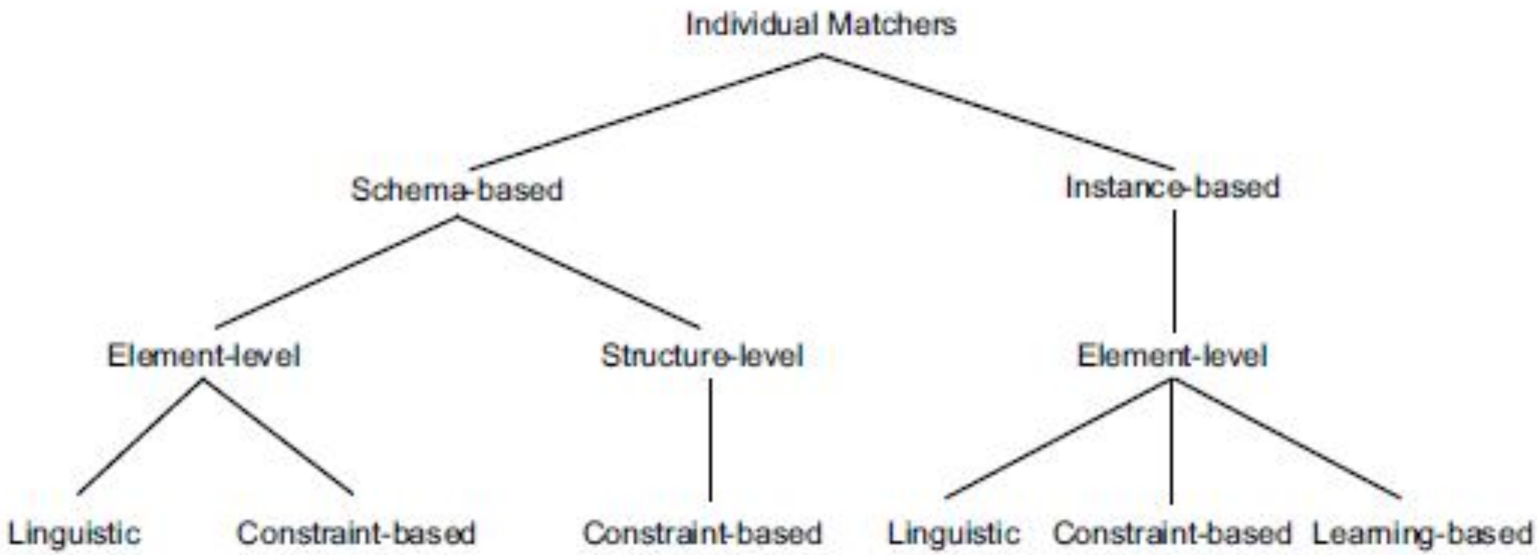


Fig. 4.7 Taxonomy of Schema Matching Techniques

INTEGRATION

If GCS wasn't defined (determined up-front).

Use correspondances between LCSs to integrate them => **Create GCS**

If GCS determine up-front, done in matching step.

Binary: two schema at a time

N-ary integration : several schema per iteration

One-pass integration

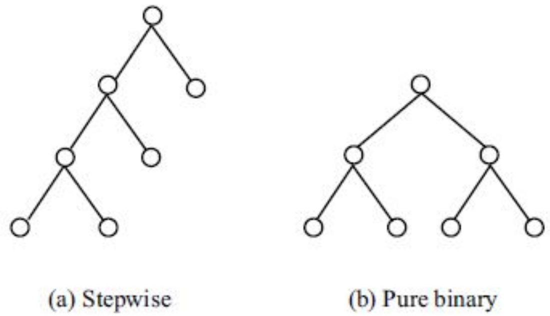


Fig. 4.11 Binary Integration Methods

MAPPING (I)

Explicit how obtain Global Database.

DW: extract data from sources + translate them to populate it

Database integration: mapping used during query processing

Mapping **creation**: Create **set of queries** to create GCS data instances

Algorithm takes into account semantics over sources but not over target = GAV mapping

More complex: extend to target semantics = GLAV mapping

MAPPING (II)

Mapping **maintenance**: detection, correction of inconsistencies (schema evolution)

CLEANING

DW: cleaning as global Database is created

Can be done **off line**

Database Integration: During query processing

Need to be **on-line** -> may prefer tolerant system

UNSTRUCTURED DATA

UNSTRUCTURED DATA

Data **not from databases**.

Large amount of Unstructured data in web (80-90% Merrill Lynch)

Easy to share, query.

Can be image, text, number, facts.

No pre-defined **schema or organization**

insufficient semantic information:

provide precise and complete answers, enforce integrity constraints, combine information in meaningful ways

Semi-structured: Not conform with structure data but contains some form of organization : XML, JSON…

YAHOO! PIPES

PIPES

-Free online service: aggregates, manipulate mashup web's content.

- Output as RSS, JSON, XML…

- Saved/Published/Shared

- User-friendly Graphical interface: Wire modules together, each one = specific task

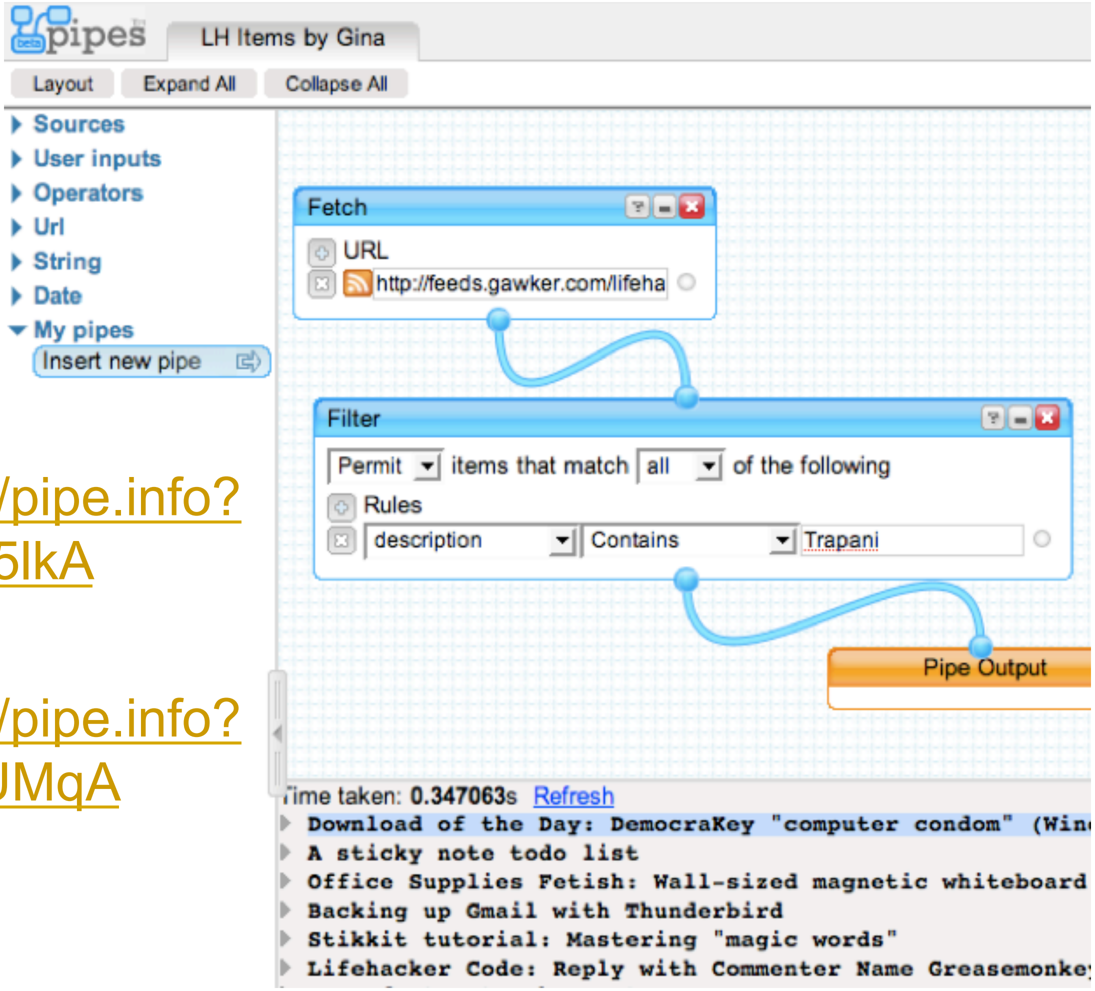
-> build mashups, produce enriched results, use and modify content from different sources

PIPES

-Two examples:

Aggregates news alert: http://pipes.yahoo.com/pipes/pipe.info?_id=fELaGmGz2xGtBTC3qe5IkA

E-bay price: http://pipes.yahoo.com/pipes/pipe.info?_id=avkEShi32xG_EF6KZVUMqA



OPEN REFINE

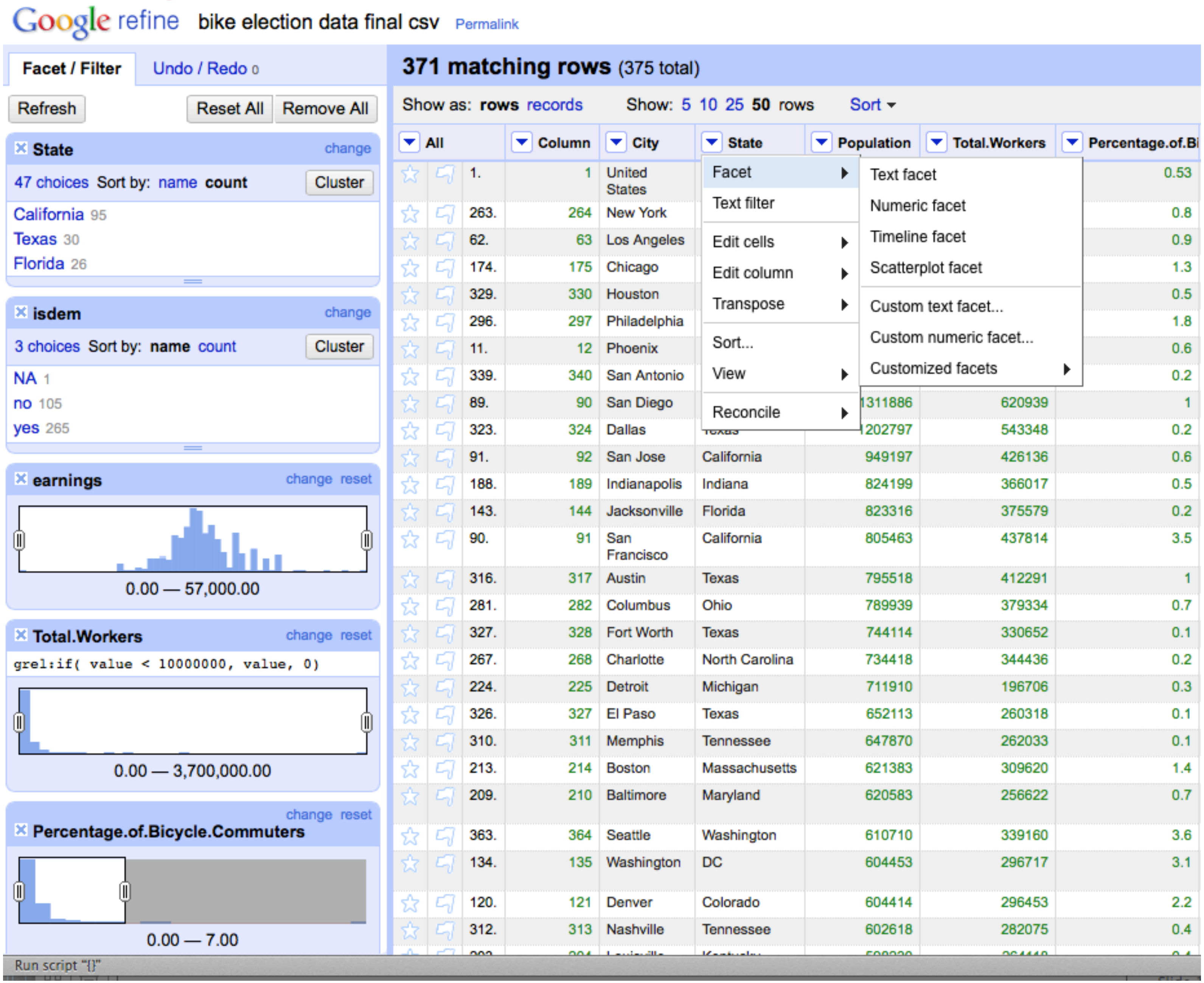
7. OPEN REFINE

OpenRefine (openrefine.org)

a free open-source desktop tool (windows/linux/mac) that allows you to load data, understand it, clean it up, reconcile it to a master database, and augment it with data coming from Freebase or other web sources.

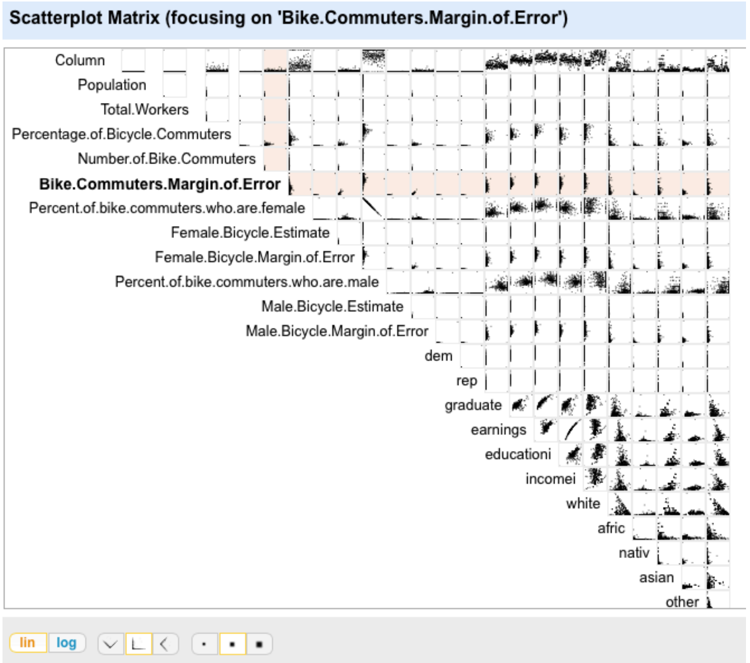
1. Import data in various formats
2. Explore datasets in a matter of seconds
3. Apply basic and advanced cell transformations,
4. Deal with cells that contain multiple values
5. Create instantaneous links between datasets
6. Filter and partition your data easily with regular expressions
7. Use named-entity extraction on full-text fields to automatically identify topics
8. Perform advanced data operations with the GREL

7. OPEN REFINE



7. OPEN REFINE

1. Explore / Clean Data :



Facets!
<u>"text facet" over a categorical column</u>
to group the values in the column, telling you what values are there and how many of each you have in it.

You can sort the groups by count (to see the biggest) or by alphabetical order the **"Cluster" feature** will intelligently provide you a view of what it thinks may be same values over a column(based on different heuristics that you may select from)

Excellent for merging values into one group (ie, changing all their cell values to be the same) by showing you the inconsistencies in your data.

7. OPEN REFINE

Explore/ Clean Data:
use "numerical facets" for continous/discrete valued columns,
Do log transforms over a column and instantly preview

Select a window over a distribution curve to select only rows within that range, and see the errors or missing values along the distribution
Timeline facets, scatterplot facets!

2. Transform Data :
The idea with open refine is:

1. use a filter or facet to isolate the rows you want to change
2. run a command on those rows to change them in one shot

- Run a text transformation on a given column via a regex and create a new column with the results.
- Run an API call over a give column, setting the throttle rate, and storing results in new column
- Export history of transformations (ie, a template) you've done so you may use it on new data.

7. OPEN REFINE

Reconcile and Match:
Open Refine can be used to link and extend your data with various web services.

Scenario 1:
Given a dataset of addresses, get latitude and longitude of each.
1. On address column, select "Edit Column" -> "add column by fetching URLs" , set url to
"http://nominatim.openstreetmap.org/search?format=json&email=name@gmail.com&app=google-refine&q=" + escape(value, 'url')
2. Set Throttle Delay to 1500 milliseconds, and name new column “JSON”
3. On JSON column, "Edit column" -> "Add column based on this column”
   in GREL: with(value.parseJson()[0], pair, pair.lat + "," + pair.lon)
   and name new column "Latitude, Longitude"

Reconciliation is the process of associating names to database keys.

Scenario 2:
Given dataset of movie titles, get director and year info using Freebase.
1. On film column, select "Reconcile " then "Freebase Reconciliation Service”
   Refine tries to guess what type of freebase object this column contains and
   shows you a list for you to choose from a set.
2. Click "Start Reconciling"
   This turns each film name, into a link to the appropriate page on Freebase or
   a list of choices for the user to choose from if uncertain (Terminator, Ocean's Eleven)
3. Once done with manual reconciliation, add more information from Freebase to dataset by going to film column, selecting "edit column" -> "add column from freebase based on this column" , and then selecting from a list of properties you’d like to add (director, year, genre, etc)

8. RELEVANCE TO OPEN DATA

More & More available Open Data and usage

=> Increase need for Data **Integration**.

Want to **connect** independent OD objects which may be heterogeneous, and highly distributed. Ie, Disorganized information.

In order to gather data **from different sources** over the web, you must establish some sort of global system.

⇒So need for Schema Generation (as in OD project)+ Need to clean/ process.

⇒Database Integration, can also be Data Warahouse

8. RELEVANCE TO OPEN DATA

- Information Integration is essential to your system's **middleware**, and to how you can build on legacy systems via API's.
- It encompasses different schemes (physical / virtual) and provides flexibility in incorporating new data sources into a system.
- It provides strategies for **mapping sources** (either structured or unstructured) so that they appear unified for querying purposes.
- It is still a evolving field, but does provide ways to do **entity matching** (clustering in Open Refine for instance)
- Yahoo Pipes and Open Refine are great tools for constructing datasets from disparate sources using **linked data** (freebase, dbpedia extensions) and are usable by non technical users as well, though outside of that use case, it would be clumsy way of implementing a middleware solution
- Other alternatives: Stanford Data Wrangler project (vis.stanford.edu/wrangler/) and R reshape package (http://had.co.nz/reshape/)

THANKS.

QUESTIONS?

9. OUR QUESTIONS FOR YOU:

1. We are trying to combine multiple sources into an integrated database. There are 3 professors who each independently have SQL DBs of their student's records. They want to share the information amongst themselves, but while one has the students in a table named "students", another has them in a table named "alumnos", while the third uses a table "clients" to store them.
   What type of heterogeneity problem does this situation reflect?

a) Semantic
b) Schema
c) Value
d) Query Language

9. OUR QUESTIONS FOR YOU:

2. You want to create a web application that gathers information about cities (hotels, restaurants, demographic information, weather, universities, museums…,etc).

- You want to have a large number of users.
- You need to integrate information from lots of local DBs (most of them will be from Open Data sources).
- You are using a lot of different sources (with high heterogeneity and autonomy) and different schemas among the local DBs.
- You even want to integrate some unstructured Data.
- You know that with this kind of data, new sites will regularly be added/ suppressed from your distributed system.
- You need up-to-date data but the queries will be quite simple.
- It will be a read-only system. The users won't have to store data, nor communicate between each other.

Which Structure will you choose?
DataWarehouse/ DB Integration/ peer-to-peer/ Federated DB?
Explain why and try to be precise...

BIBLIOGRAPHY

- HALEVY, Alon Y., ETZIONI, Oren, DOAN, AnHai, *et al.* Crossing the Structure Chasm. In : *CIDR*. 2003.
- ÖZSU, M. Tamer et VALDURIEZ, Patrick. *Principles of distributed database systems*. Springer, 2011.
- ULLMAN, Jeffrey D., GARCIA-MOLINA, Hector, et WIDOM, Jennifer. *Database systems: the complete book*. 2nd Edition. Pearson, 2009.