Data Integration

Giorgos Alexiou
ATHENA Research Center
&
Athens University of Economics and Business
galexiou@athenarc.gr

Data

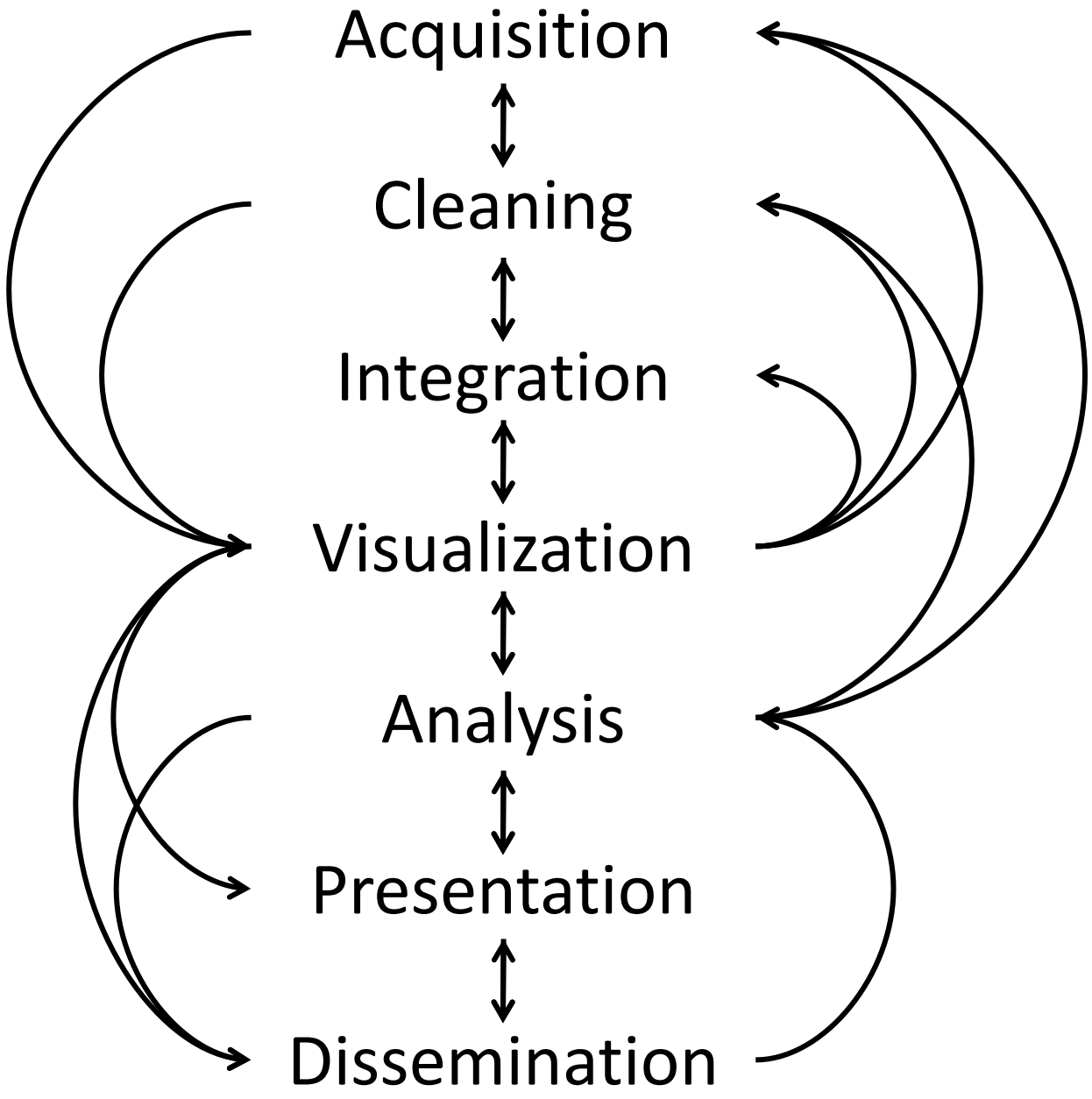
- **Data** is your most valuable asset.

- Utilising it correctly can allow you to
  - make intelligent business decisions,
  - drive growth and
  - improve profitability.

- Nonetheless, many companies lack a centralised approach to data, with data siloes being one of the most common issues.

Data

- **The problem with data silos:**
  - Most of the businesses use many different applications, therefore data is scattered amongst disparate systems.
  - This results in poor communication between departments, systems and processes, leading to inability to meet both clients' needs and business goals.
  - Moreover, important decisions are based on misinformation, which can have disastrous effects.


Acquisition
Cleaning
Integration
Visualization
Analysis
Presentation
Dissemination

What is Data Integration?
Why is it important?

Data Integration

The problem of combining data **residing in different sources** (e.g databases) and providing users with a **unified view** of these data.

Data integration is occurring with **increasing frequency** as the **volume** and **the need to share** existing data continues to grow.

Or, for our purposes: **how can analysts effectively leverage multiple data sources?**

Data Integration

- Data integration allows businesses to combine data residing in different sources to provide users with a real-time view of business performance.
- As a strategy, integration is the first step toward transforming data into meaningful and valuable information.
- According to the Smart Data Collective, “If businesses want the right kind of data to underpin advanced analytics processes or to create multi-dimensional views of customers, data integration must be pursued as a strategic function that aligns with business objectives.”

The benefits of data integration

- **Improve decision making**
  - Access to real-time data presented in an easy to digest format will provide you with invaluable acumen, helping you to be proactive, uncovering opportunities and identifying potential bottlenecks before they occur.

- **Improve customer experience**
  - Siloed data sets prevent a complete view of customers which impact on sales and ultimately revenue. Only when you have access to real-time customer information as well as the historical data can you target your customers with the right message at the right time to improve your customer experience, loyalty and ultimately revenue.

The benefits of data integration cont’d

- **Streamline operations**
  - From procurement and supply chain to manufacturing and product management, real-time access is useful for improving processes, increasing production, and lowering costs across various departments, including sales, production, distribution and more.
- **Increase productivity**
  - If you have to constantly move between numerous systems to gather insight, your productivity is significantly reduced. With automatic data integration, your data from all the different sources is pooled into a single customer view, allowing you to be more productive.

The benefits of data integration cont’d

- **Predict the future**
  - Combine historical data with sales pipeline information to make forecasts and anticipate customer demands. This will help you evaluate your products and services, while providing you with the ability to remain ahead of the competition.

Example Applications

1. **Enterprise Information Integration**: making separate DB's, all owned by one company, work together.
2. **Scientific DB's**, e.g., genome DB's.
3. **Catalog integration**: combining product information from all your suppliers.

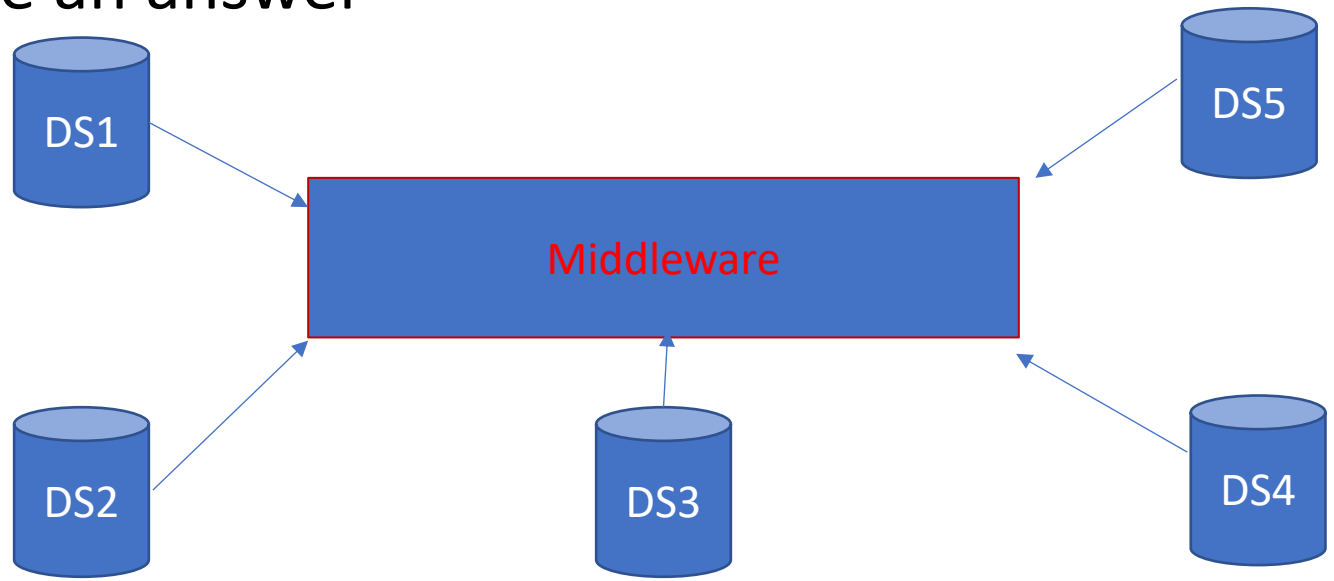
Challenges

1. **Legacy databases** : DB's get used for many applications.
   - Starting over is not always possible
     - High development cost
     - Incompatibility issues with legacy software
     - You can't change its structure for the sake of one application, because it will cause others to break.

2. **Incompatibilities** : Two, supposedly similar databases, will mismatch in many ways.

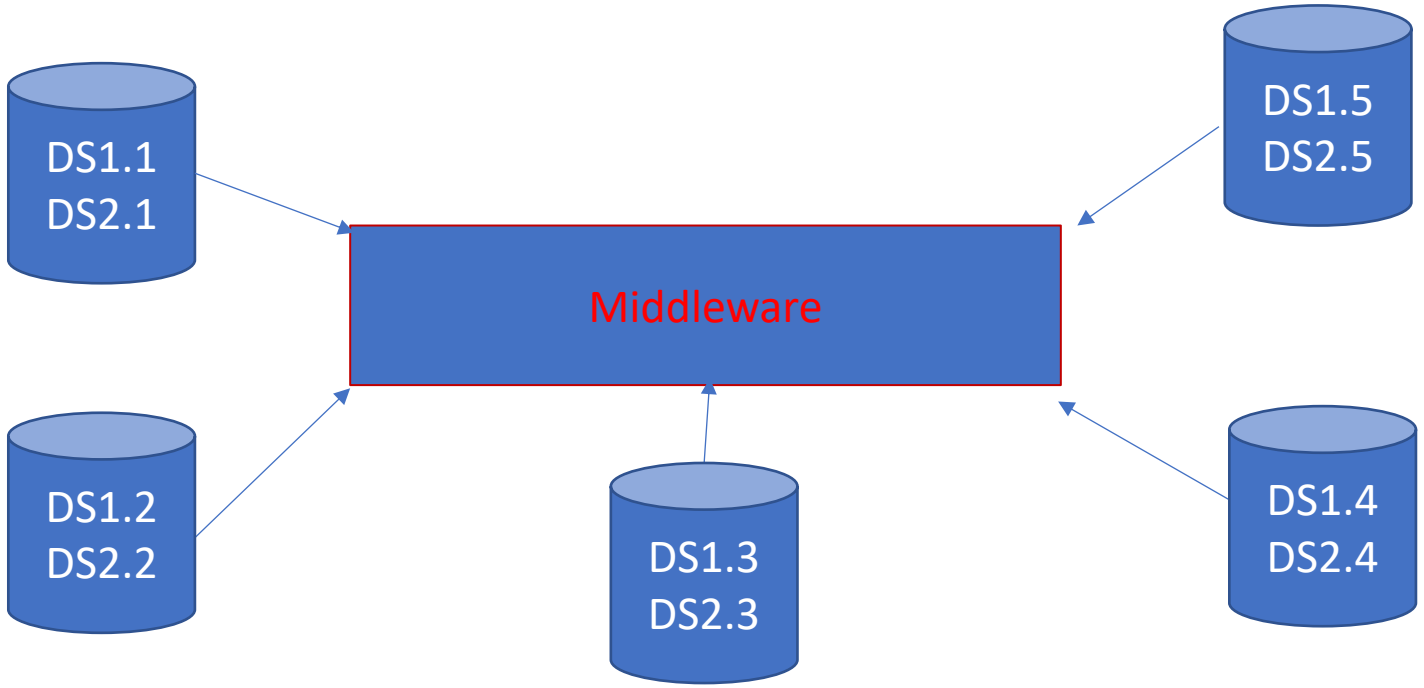
Example Scenario

- As mentioned earlier (slide2), businesses uses many different applications, therefore data is scattered amongst disparate systems
- Management may need to pose queries that require the combination of these data sources (DS) in order to provide an answer



Example Scenario OR

- A business A acquired a business B
- Both companies have data sources that describe similar entities through different schemas and these datasets need to be integrated



Incompatibilities

- Data sources vary-differ in many possible ways, even in cases that are built to store the same kind of datasets

- In such cases the aforementioned data sources are called **Heterogeneous,** and present many incompatibilities

Examples: Incompatibilities

1. **Lexical** : addr in one DB is address in another.

2. **Value mismatches** : is a “red” car the same color in each DB? Is 20 degrees Fahrenheit or Centigrade?

3. **Semantic** : are “employees” in each database the same? What about consultants? Retirees? Contractors?

Examples: Incompatibilities

1. Communication Heterogeneity
    1. Communication protocol (e.g. http, mqtt)
    2. Tunnelled connection
    3. Remote Connection

2. Query Language Heterogeneity
    1. Different SQL Dialects (e.g. SQL, HiveQL, Sparql)
    2. NoSQL (e.g .MongoDB)
    3. URI (e.g. Rest APIs)
    4. File-based (e.g. parquet, avro etc)

What Do You Do About It?

- Grubby, handwritten translation at each interface.
  - Some research on automatic inference of relationships.

- ***Wrapper*** (aka “adapter”) translates incoming queries and outgoing answers.
  - An abstraction layer (middleware) on top of all the available data sources.
  - The layer of abstraction could be a set of relational views.
  - These views could be either virtual or materialized.
  - A form of SQL could be used to access this abstraction layer.

Integration Architectures

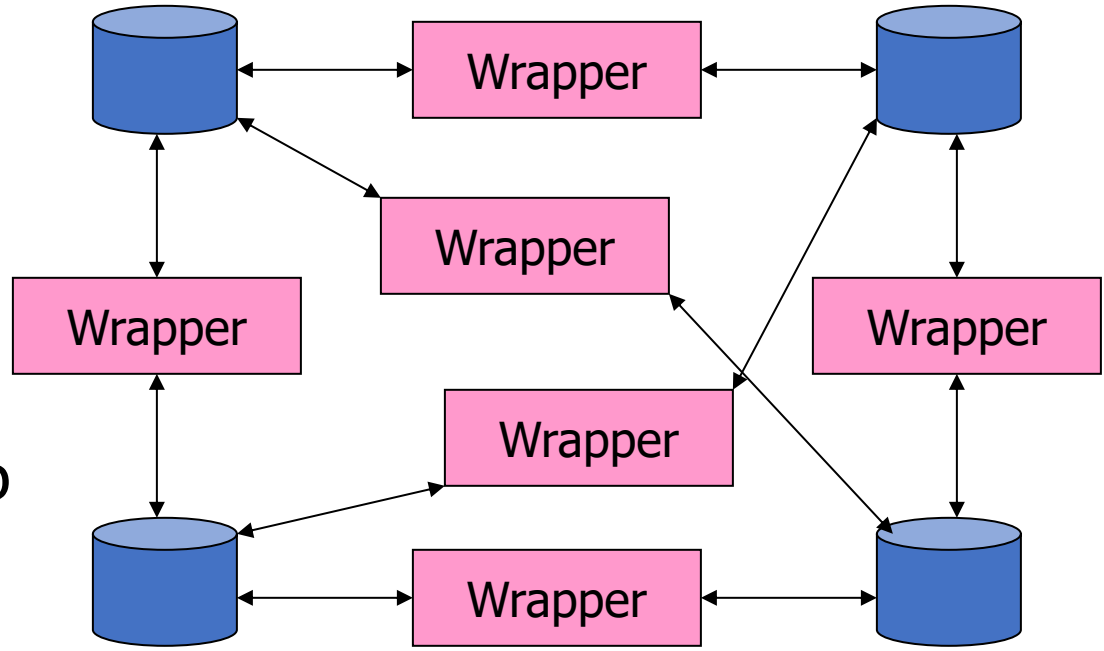
1. **Federation** : everybody talks directly to everyone else.

2. **Warehouse** : Sources are translated from their local schema to a global schema and copied to a central DB.

3. **Mediator** : Virtual warehouse --- turns a user query into a sequence of source queries.

Federations

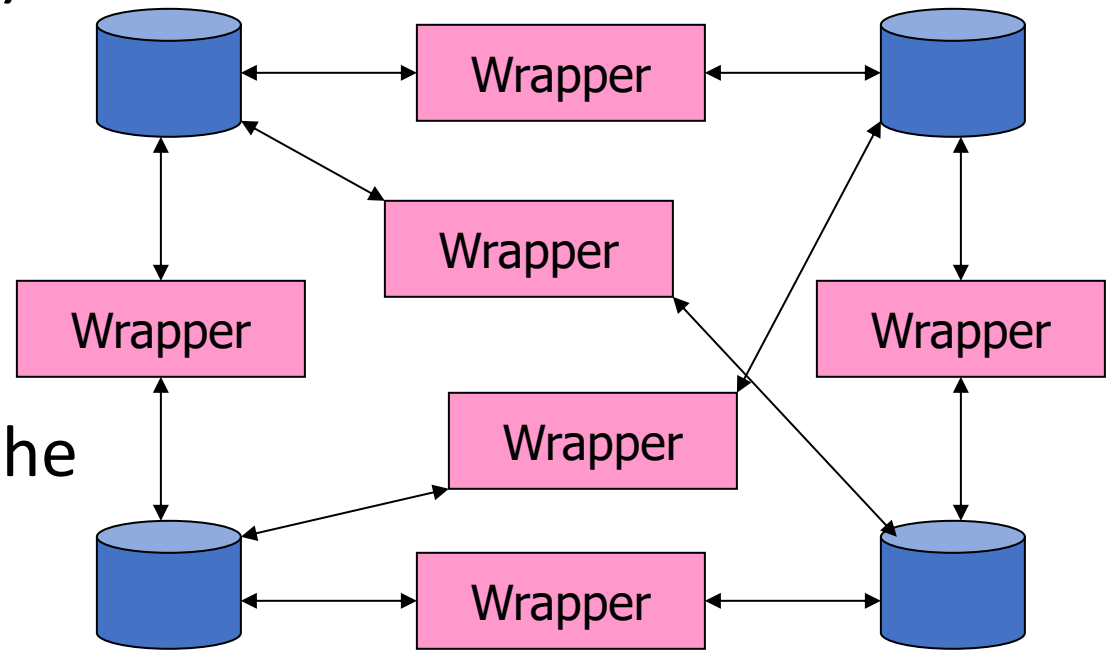
- Perhaps the simplest architecture for integrating several databases
- One source can call on others to supply information
- Each query works properly for the database to which it is addressed



Federations (cont'd)

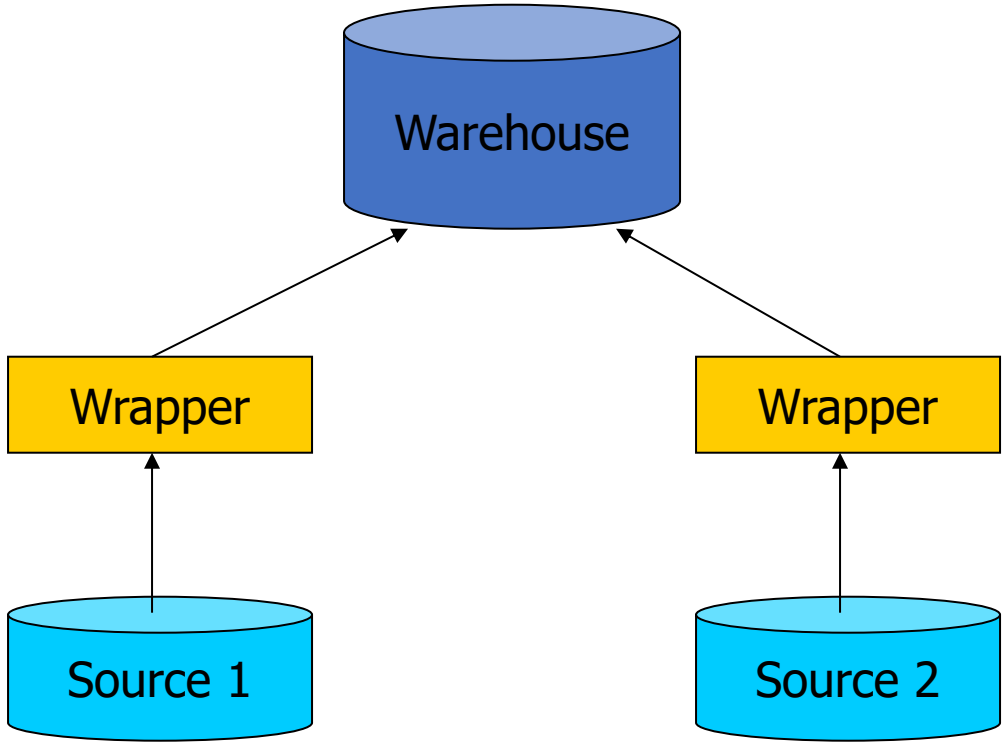
- If n databases each need to talk to the *n — 1* other databases, then we must write *n(n — 1)/2* pieces of code to support queries between systems

- Could be useful if the systems need to query the database of a very limited number of other systems



Warehouses

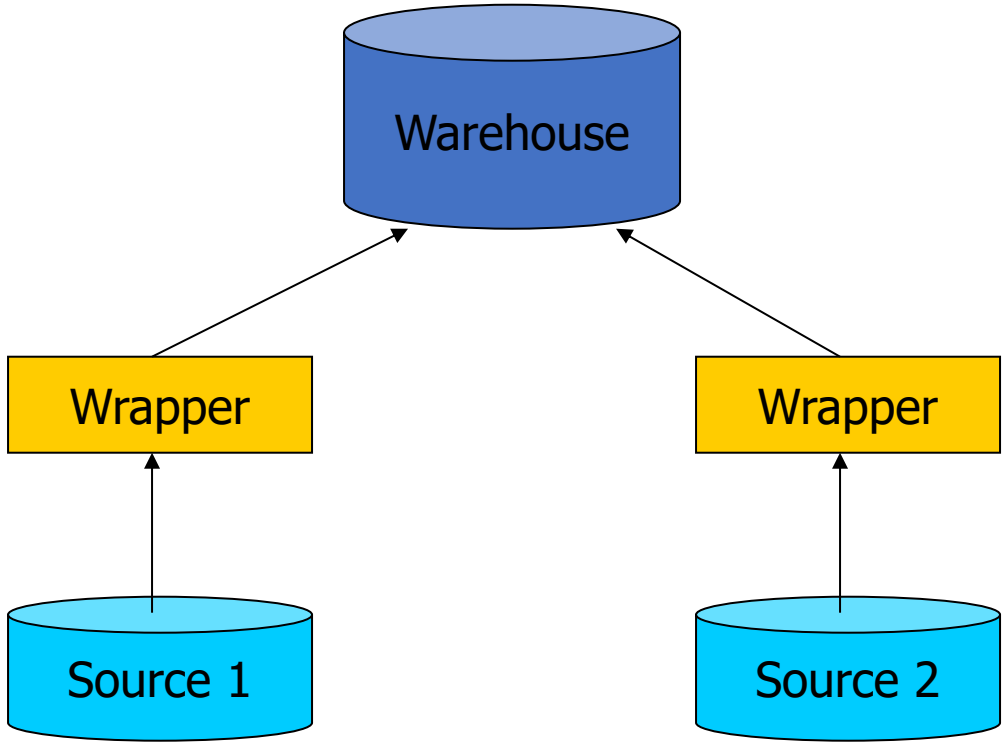
- Data from several sources is extracted and combined into a global schema
- In a data warehouse system, the source extractors consist of:
  - One or more predefined queries that are executed at the source to produce data for the warehouse.
- Suitable communication mechanisms, so the extractor can:
  - Pass queries to the source
  - Receive responses from the source
  - Pass information to the warehouse.



Warehouses (cont'd)

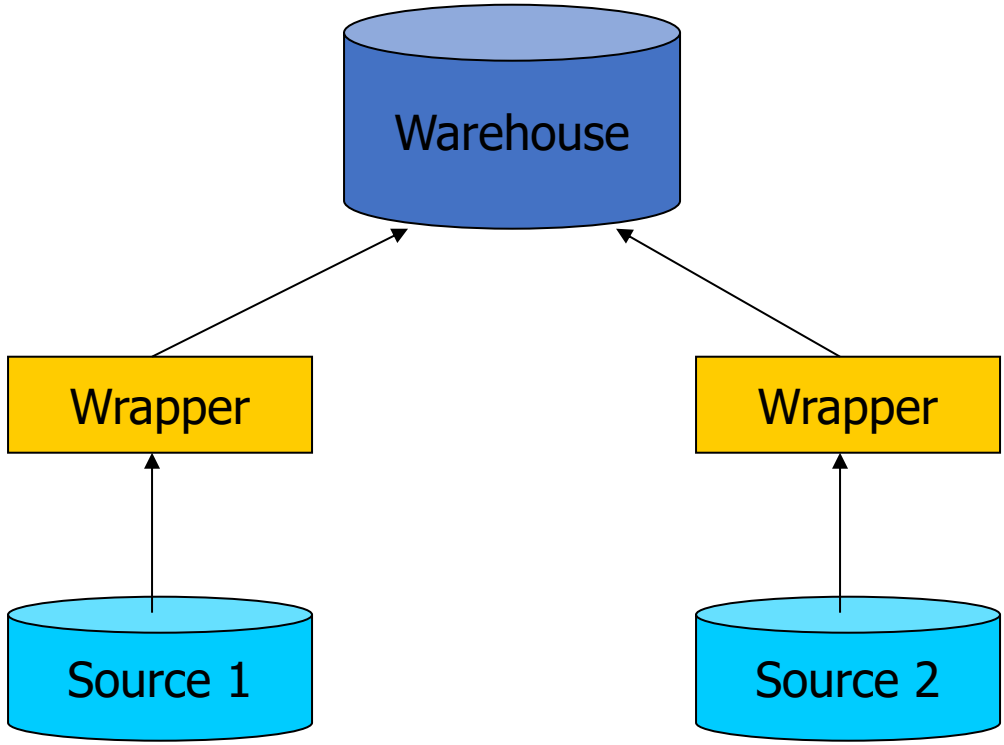
- The predefined queries to the source could be:
  - SQL queries if the source is a SQL database
  - Operations in whatever language was appropriate for a source that was not a database system

- Queries may be issued by the user exactly as they would be issued to any database



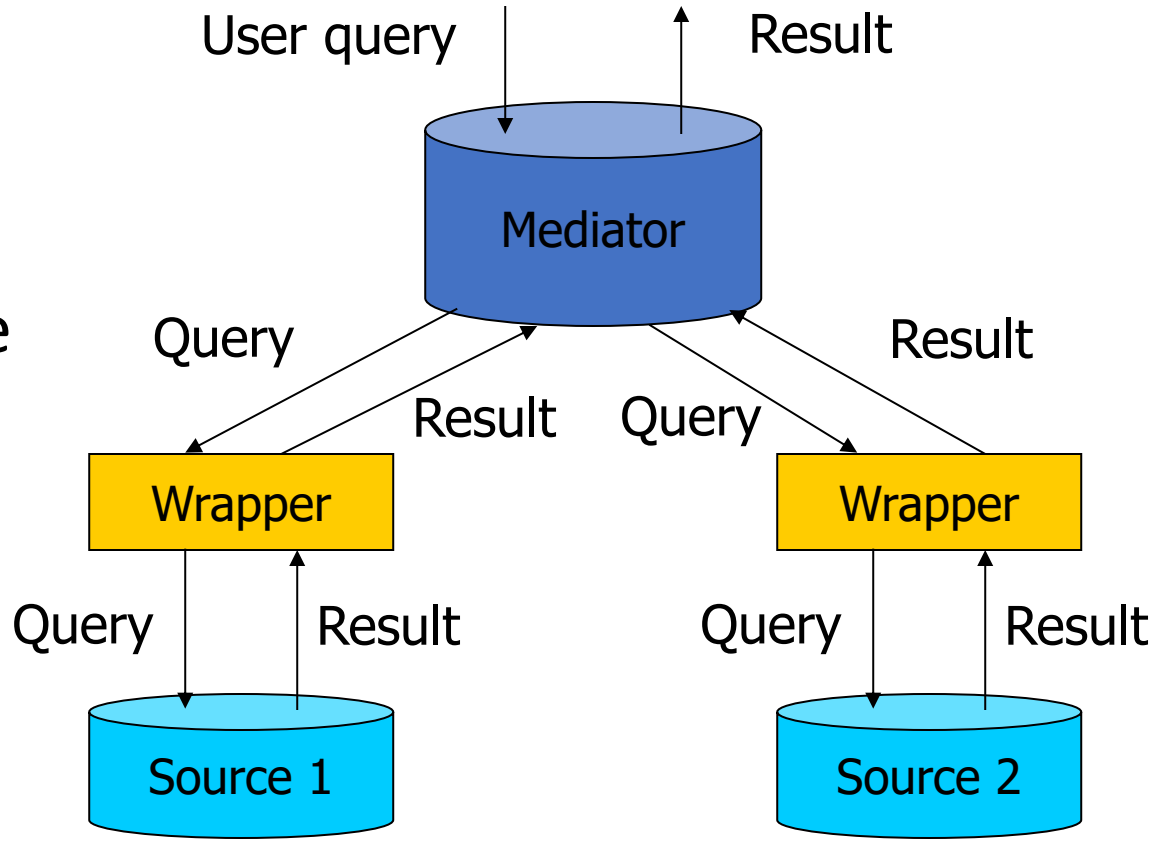
Warehouses (cont'd)

- Two approaches on updating the DW:
  - The warehouse is periodically closed to queries and reconstructed from the current data in the sources. (e.g. once a night or at even longer intervals)
  - The warehouse is updated periodically (e.g., each night), based on the changes that have been made to the sources since the last time the warehouse was modified. (incremental update)
- It is generally too expensive to reflect immediately, at the warehouse, every change to the underlying databases.



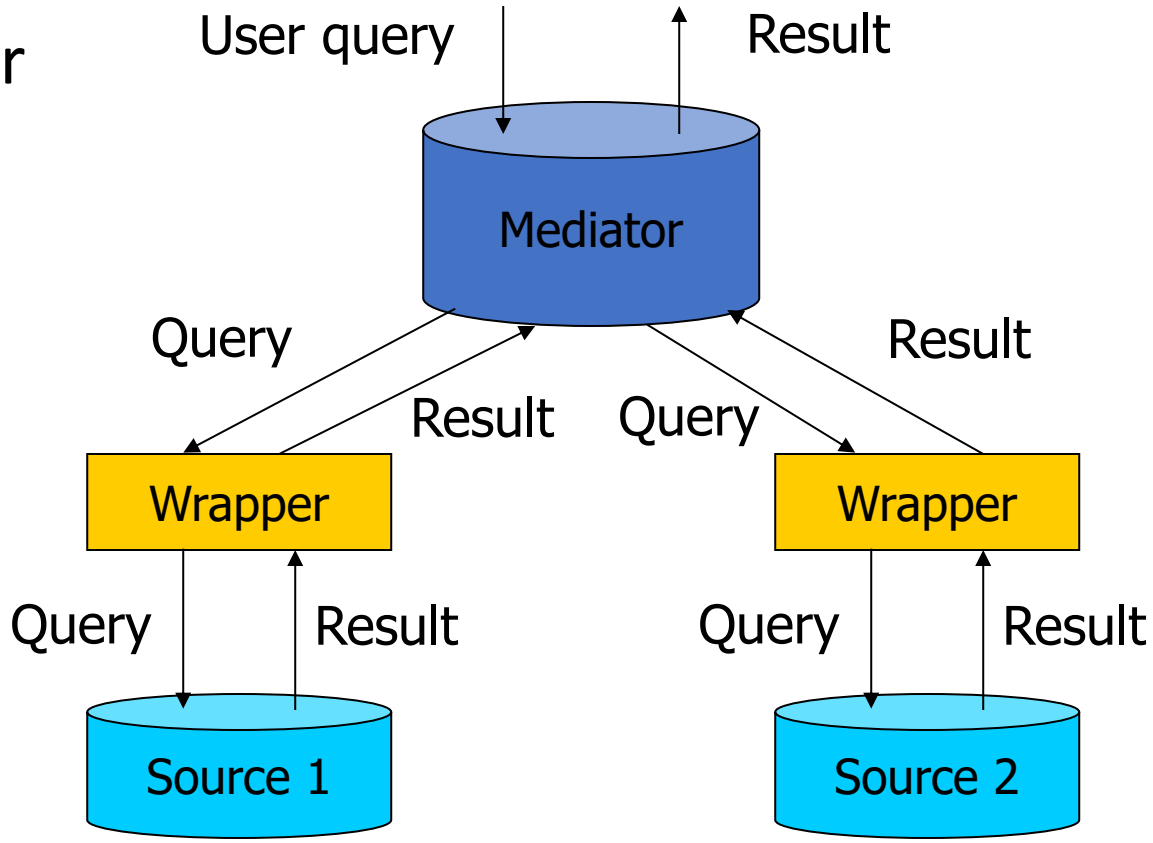
Mediators

- A mediator supports a virtual view, or collection of views, that integrates several sources.
- The mediator doesn't store any data. The mechanics of mediators and warehouses are rather different.
- To begin, the user or application program issues a query to the mediator.
- Since the mediator has no data of its own, it must get the relevant data from its sources and use that data to form the answer to the user's query.



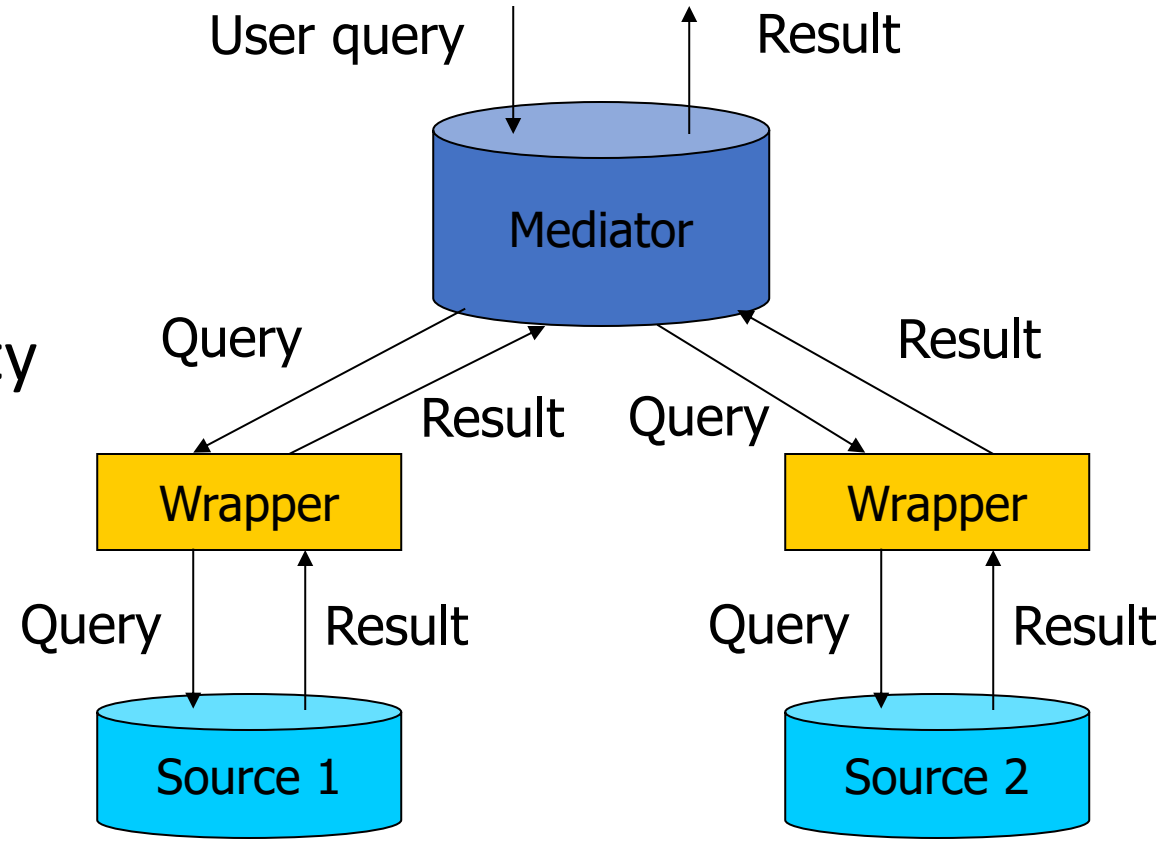
Mediators (cont'd)

- The mediator sends a query to each of its wrappers, which in turn send queries to their corresponding sources.
- The mediator may send several queries to a specific wrapper, and may not query all wrappers.
- The results come back and are combined at the mediator.



Mediators (cont'd)

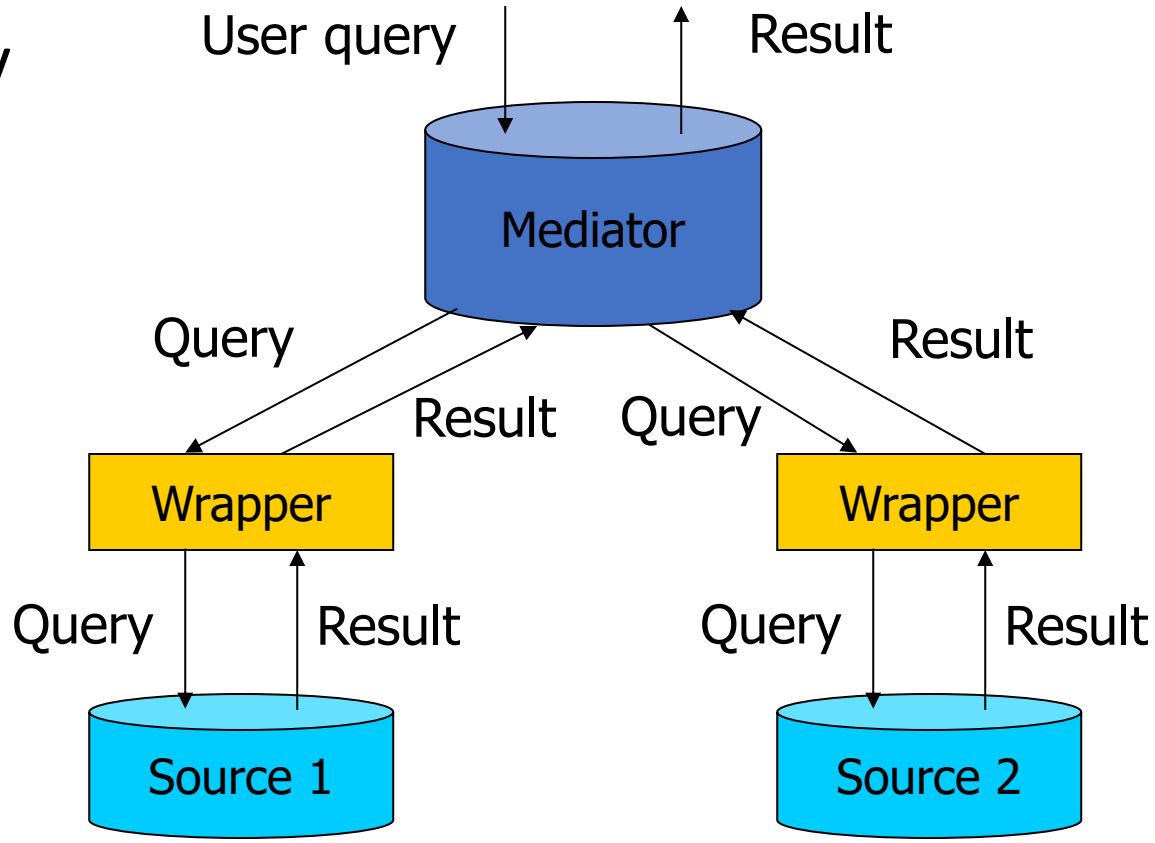
- Mediator systems require more complex wrappers than most of the warehouse systems
- The wrapper must be able to accept a variety of queries from the mediator and translate any of them to the terms of the source
- The wrapper must then communicate the result back to the mediator



Mediators (cont'd)

- A systematic way to design a wrapper that connects a mediator to a source is to classify the possible queries that the mediator can pose, into templates.

- Templates are queries with parameters that represent constants.



Templates for Query Patterns

- Suppose we want to build a wrapper for the source of Car Dealer 1, which has the schema:
  - Cars(SerialNo, model, color, trans,…)

- For use by a mediator with schema:
  - CarMed(dealer, serialNo, model, color, trans)

Templates for Query Patterns

- Suppose we want to build a wrapper for the source of Car Dealer 1, which has the schema:
  - Cars(SerialNo, model, color, trans,…)

- For use by a mediator with schema:
  - CarMed(dealer, serialNo, model, color, trans)

- A possible solution:
  - SELECT * FROM CarMed WHERE color = $c;

  =>

  SELECT “dealer1”, serialNo, model, color, trans FROM Cars WHERE color = $c;

Templates for Query Patterns

- In this case, there are eight choices, if queries are allowed to specify any of three attributes: *model, color, and trans*.

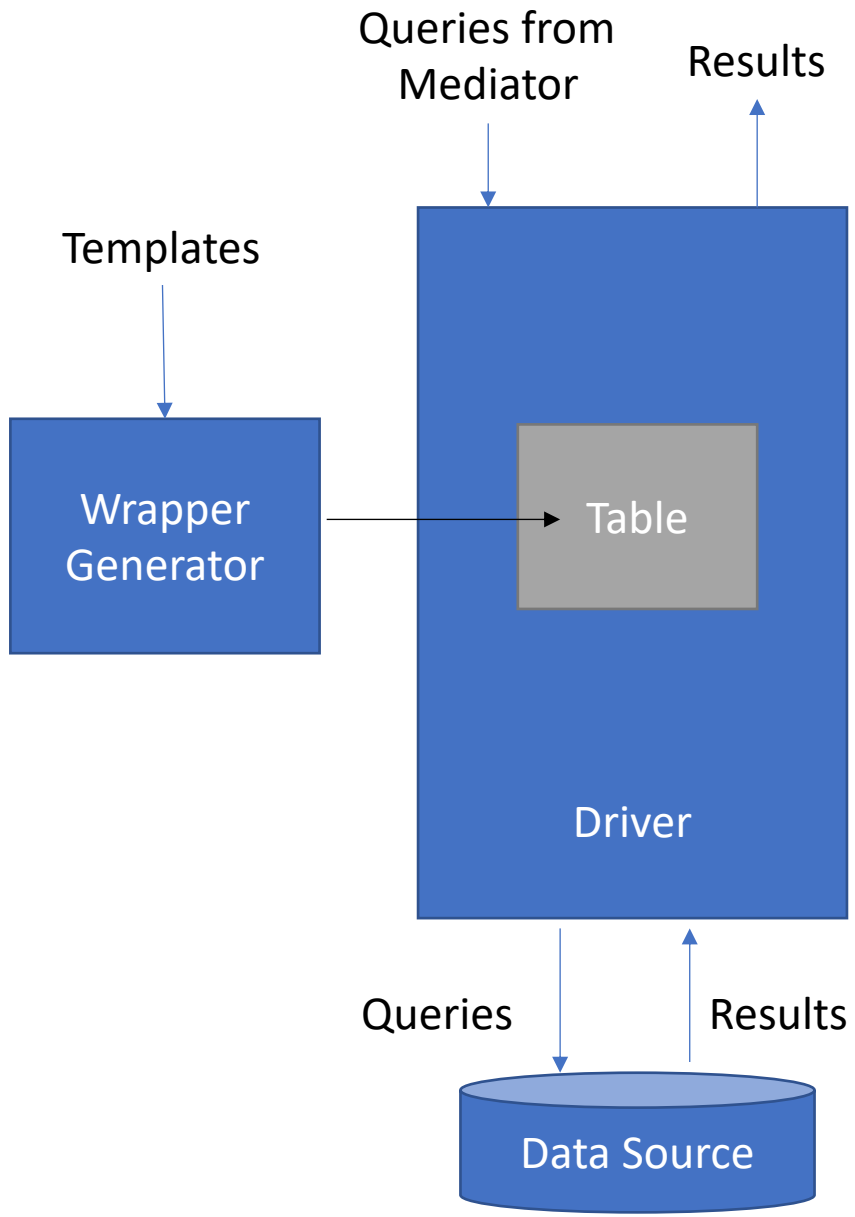
- In general, there would be 2^n templates if we have the option of specifying n attributes

- The number of templates could grow unreasonably large

```
SELECT * FROM CarMed WHERE color = $c;
=>
SELECT “dealer1”, serialNo, model, color,
trans FROM Cars WHERE
color = $c;
```

Templates for Query Patterns

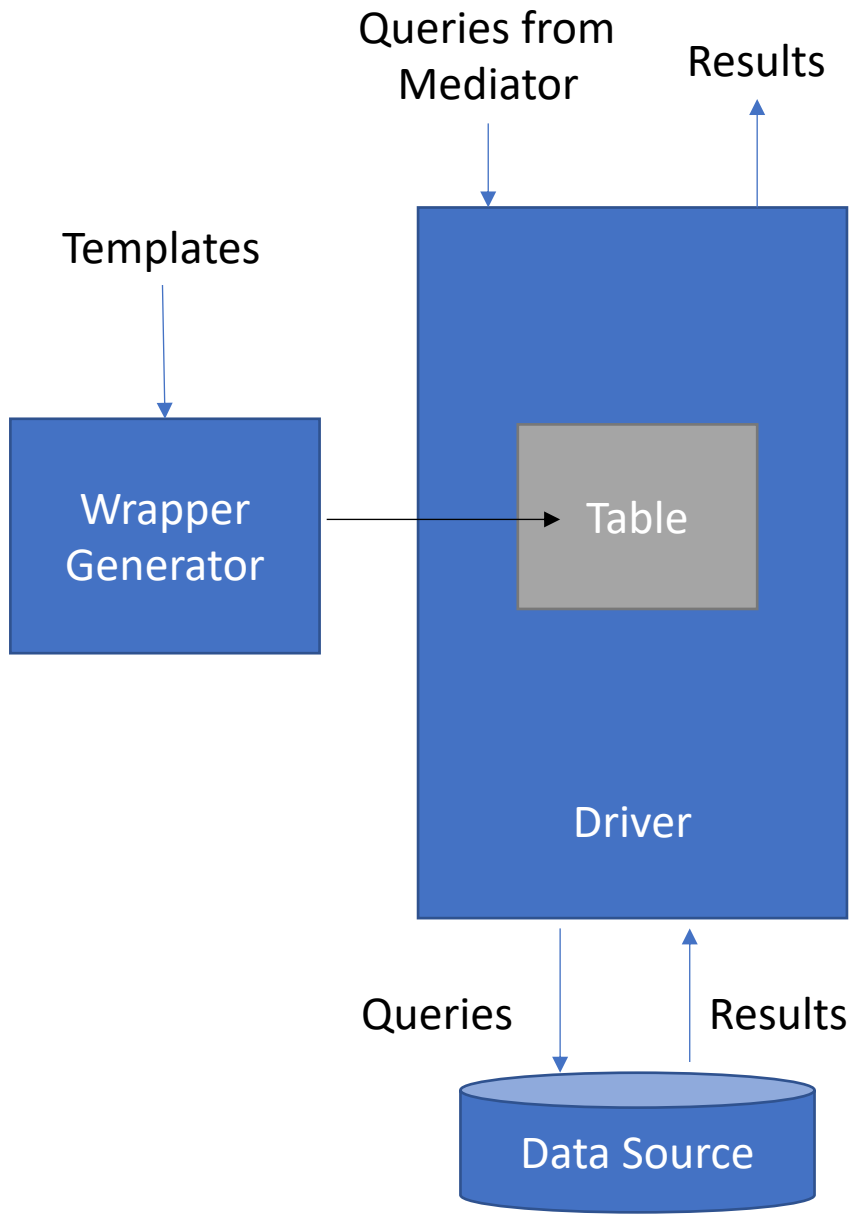
- The templates defining a wrapper must be turned into code for the wrapper itself.
- The software that creates the wrapper is called a wrapper generator
- The wrapper generator creates a table that holds
  - the various query patterns contained in the templates
  - the source queries that are associated with each



Templates for Query Patterns

- A driver is used in each wrapper; in general the driver can be the same for each generated wrapper.

- The task of the driver is to:
  - Accept a query from the mediator
  - Search the table for a template that matches the query
  - Send the query to the source
  - Process the response and return it to the mediator



Filters

- It is not always realistic to write a template for every possible form of query
- Suppose that a wrapper on a car dealer's (dealer1) database has the template for finding cars by color
- The mediator is asked to find cars of a particular model and color

```
SELECT "dealer1", serialNo, model, color,
trans FROM Cars WHERE
color = $c;
```

Filters

- As long as the wrapper has a template that (after proper substitution for the parameters) returns a superset of what the query wants, then it is possible to filter the dataset at the wrapper and pass only the desired tuples to the mediator.

SELECT “dealer1”, serialNo, model, color, trans FROM Cars WHERE
color = $c;

- In practice, the tuples could be produced one-at-a-time and filtered one-at-a-time, in a pipelined fashion, rather than having the entire stored at the wrapper and then filtered

Capability – Based Optimization

- A typical DBMS estimates the cost of each query plan and picks what it believes to be the best.
- Optimization by a mediator usually follows a strategy known as capability-based optimization.
- The central issue is not what a query plan costs, but whether the plan can be executed at all.
- Only among plans found to be executable (“feasible”) we try to estimate costs.
- There are several reasons why a source may be incapable of providing parts of the information needed for a query to be executed successfully.

Limited Source Capabilities

- Reasons why a source may limit the ways in which queries can be posed:
  - Protection against a rival exploiting its database (e.g. Amazon.com would never allow the equivalent of a “SELECT * FROM books”)
  - Privacy Protection (e.g. a medical database may answer queries about averages, but won’t disclose the details of a particular patient’s medical history)
  - Lack of proper Indexes may make certain kinds of queries too expensive to execute.
  - The interface provide is not designed to support such queries (e.g. an API that provides a specific number of predefined HTTP requests)

Adornments

- We need a way to describe the capabilities of a data source.
- In order to describe the legal forms of queries, we may use adornments.
- Adornments are sequences of codes that represent the requirements for the attributes of the relation, in their standard order.

Adornments

- The codes we shall use for adornments reflect the most common capabilities of sources. They are:

Symbol	Description
f	(free) means that the attribute can be specified or not, as we choose.
b	(bound) means that we must specify a value for the attribute, but any value is allowed.
u	(unspecified) means that we are not permitted to specify a value for the attribute.
c[S]	(choice from set S) means that a value must be specified, and that value must be one of the values in the finite set S. (e.g. values from a dropdown menu)
o[S]	(optional, from set S) means that we either do not specify a value, or we specify one of the values in the finite set S.

- In addition, we place a prime (e.g., f ') on a code to indicate that the attribute is not part of the output of the query

Example: Adornments

- Suppose we have the following schema:
  - Cars(SerialNo, model, color, trans)

- **Scenario 1**: The user specifies a serial number. All the information about the car with that serial number (i.e., the other three attributes) is produced as output.

- The adornment for this query form is:

  b'uuu

Example: Adornments

- Suppose we have the following schema:
  - Cars(SerialNo, model, color, trans)

- **Scenario 2**: The user specifies a model and color, and perhaps whether or not automatic transmission is wanted. All four attributes are printed for all matching cars.

- The adornment for this query form is:

  ubbo[yes, no]

Query Plan Selection

- Given a query at the mediator, a capability-based query optimizer first considers what queries it can ask at the sources to help answer the query.
- If we imagine those queries asked and answered, then we have bindings for some more attributes, and these bindings may make some more queries at the sources possible.
- We repeat this process until either:
  - We have asked enough queries at the sources to resolve all the conditions of the mediator query, and therefore we may answer that query. Such a plan is called feasible.
  - We can construct no more valid forms of source queries, yet we still cannot answer the mediator query, in which case the mediator must give up; it has been given an impossible query

Example: Query Plan Selection

- Suppose we have the following sources:

  Cars(serialNo, model, color)
  Options(serialNo, option)

- The adornment for Autos is: ***ubf***
- Options has two adornments: ***bu*** and ***uc[trans, navi]***
- Let the query be: “find the serial numbers and colors of “Audi Q3” models with a navigation system.”

Example: Query Plan Selection

- Suppose we have the following sources:

  Cars(serialNo, model, color)
  Options(serialNo, option)

- The adornment for Autos is: ***ubf***
- Options has two adornments: ***bu*** and ***uc[trans, navi]***
- Let the query be: “find the serial numbers and colors of “Audi Q3” models with a navigation system.”

Approach 1

Specifying that the model is “Q3”, query Cars and get the serial numbers and colors of all Q3s. Then, using the **bu** adornment for Options, for each such serial number, find the options for that car and filter to make sure it has a navigation system.

Example: Query Plan Selection

- Suppose we have the following sources:

  Cars(serialNo, model, color)
  Options(serialNo, option)

- The adornment for Autos is: ***ubf***
- Options has two adornments: ***bu*** and ***uc[trans, navi]***
- Let the query be: “find the serial numbers and colors of “Audi Q3” models with a navigation system.”

Approach 2

Specifying the navigation-system option, query Options using the uc[trans, navi] adornment and get all the serial numbers for cars with a navigation system. Then query Cars as in (1), to get all the serial numbers and colors of Q3s, and intersect the two sets of serial numbers.

Example: Query Plan Selection

- Suppose we have the following sources:

  Cars(serialNo, model, color)
  Options(serialNo, option)

- The adornment for Autos is: ***ubf***
- Options has two adornments: ***bu*** and ***uc[trans, navi]***
- Let the query be: “find the serial numbers and colors of “Audi Q3” models with a navigation system.”

Approach 3

Query Options as in (2) to get the serial numbers for cars with a navigation system. Then use these serial numbers to query Cars and see which of these cars are Q3s. But this approach will **fail**. The system **does not** have the capability to execute this plan.

Cost-Based Optimization

- Having found the feasible plans, the Mediator must choose among them.
- Since the sources are usually independent of the mediator, it is difficult to estimate the cost. (E.g.: A source may take less time during periods when it is lightly loaded, but when are those periods?)
- Long-term observation by the mediator is necessary for the mediator even to guess what the response time might be.
- Consider the previous example. Approach (2) uses only two source queries, while Approach (1) uses one plus the number of Q3s found in the Cars. Thus, it appears that plan (2) has lower cost.

Chain Algorithm for Answering Queries

- The algorithm is called "**chain**".
- It is not guaranteed to provide the most efficient solution, but it will provide a solution whenever one exists, and in practice, it is very likely to obtain the most efficient solution.
- Lets agree on the following notation for data sources:

$Cars^{buu}$(serialNo, model, color)
$Options^{uc[trans,navi]}$(serialNo, option)

- And on the following notation for expressing queries:

Answer(s,c) ← $Cars^{fbf}$(s,"Q3",c) AND $Options^{fb}$(s,"navi")

Notice the simplified adornment notation used to represent the arguments of the subgoals that are bound to a set of constants

"find the serial numbers and colors of Q3 models with a navigation system"

Chain Algorithm for Answering Queries (cont'd)

- The algorithm maintains two kinds of information:
  - An adornment is maintained for each subgoal.
    - Initially, the adornment for a subgoal has ***b*** if and only if the mediator query provides a constant binding for the corresponding argument of that subgoal.
    - In all other places, the adornment has ***f*** 's.
    - After every step of the algorithm, the adornment of each subgoal is updated.
  - A relation ***X*** that is (a projection of) the join of the relations for all the subgoals that have been resolved.
    - Initially, since no subgoals have been resolved, ***X*** is a relation over no attributes, containing just the empty tuple.
    - As the algorithm progresses, ***X*** will have attributes that are variables of the rule — those variables that correspond to **b**'s in the adornments of the subgoals in which they appear.

Answer(s,c) ← $Cars^{fbf}$(s,"Q3",c) AND $Options^{fb}$(s,"navi")

Chain Algorithm for Answering Queries (cont'd)

- The core of the Chain Algorithm is as follows:
  1. Initialize a relation ***X*** and the adornments of the subgoals.
  2. **Select** a subgoal that can be resolved.
  3. **Join X** and the result of the subgoal.
  4. **Project** out of **X** all components that correspond to variables that do not appear in the head or in any unresolved subgoal.
  5. **Update** the adornments of the unresolved subgoals.
  6. **Repeatedly select** a subgoal that can be resolved and **update X** accordingly until no unresolved subgoal has been left.
  7. If we succeed in resolving every subgoal, then relation **X** will be the answer to the query. If at some point, there are unresolved subgoals, yet none can be resolved, then the algorithm fails. In that case, there can be no other sequence of resolution steps that answers the query.

Example: Chain Algorithm

- Mediator query:
  - *Q*: Answer(*c*) ← $R^{bf}(1,a)$ AND $S^{ff}(a,b)$ AND $T^{ff}(b,c)$
- Example:

Relation R

Data

w	x
1	2
1	3
1	4

Relation S

x	y
2	4
3	5

Relation T

y	z
4	6
5	7
5	8

Adornment *bf* *c'*[2,3,5]*f* *bu*

Example: Chain Algorithm (cont'd)

- Initially, the adornments on the subgoals are as shown in query ***Q***, and ***X*** initially contains an empty tuple.
  - ***S*** and ***T*** cannot be resolved because they each have ***ff*** adornments, but the sources have either **a *b* or *c***.

- ***R(1,a)*** can be resolved because its adornments are *matched* by the source's adornments.

- Fortunately, the first subgoal, ***R(1,a)***, can be resolved, since the ***bf*** adornment at the corresponding source is matched by the adornment of the subgoal. Thus, send ***R(w,x)*** with ***w=1*** to get the respective tables

Relation	R		S		T	
Data	w	x	x	y	y	z
	1	2	2	4	4	6
	1	3	3	5	5	7
	1	4			5	8
Adornment	*bf*		*c'*[2,3,5]*f*		*bu*	

Q: Answer(*c*) ← $R^{bf}(1,a)$ AND $S^{ff}(a,b)$ AND $T^{ff}(b,c)$

Example: Chain Algorithm (cont'd)

- Project the subgoal's relation onto its second component, since only the second component of ***R(1,a)*** is a variable.

a
2
3
4

Relation	R		S		T	
Data	w	x	x	y	y	z
	1	2	2	4	4	6
	1	3	3	5	5	7
	1	4			5	8
Adornment	*bf*		*c'*[2,3,5]*f*		*bu*	

Q: Answer(*c*) ← $R^{bf}(1,a)$ AND $S^{ff}(a,b)$ AND $T^{ff}(b,c)$

- This relation is joined with X, which currently has no attributes and only the empty tuple.
- Since ***a*** is now bound, we change the adornment on the S subgoal from ***ff*** to ***bf***

Example: Chain Algorithm (cont'd)

- At this point, the second subgoal, ***S^{bf}(a,b)***, can be resolved.

- We obtain bindings for the first component by projecting ***X*** onto ***a***; the result is ***X*** itself. That is, we can go to the source for ***S(x,y)*** with bindings 2, 3, and 4 for **x**.

- We do not need bindings for **y**, since the second component of the adornment for the source is ***f***.

Relation	R		S		T	
Data	w	x	x	y	y	z
	1	2	2	4	4	6
	1	3	3	5	5	7
	1	4			5	8
Adornment	*bf*		*c'*[2,3,5]*f*		*bu*	

Q: Answer(*c*) ← $R^{bf}(1,a)$ AND $S^{ff}(a,b)$ AND $T^{ff}(b,c)$

Example: Chain Algorithm (cont'd)

- The ***c'[2,3,5]*** code for **x** says that we can give the source the value 2, 3, or 5 for the first argument.

- Since there is a prime on the **c**, we know that only the corresponding **y** value(s) will be returned, not the value of **x** that we supplied in the request.

Relation	R		S		T	
Data	w	x	x	y	y	z
	1	2	2	4	4	6
	1	3	3	5	5	7
	1	4			5	8
Adornment	*bf*		*c'*[2,3,5]*f*		*bu*	

Q: Answer(c) ← $R^{bf}(1,a)$ AND $S^{ff}(a,b)$ AND $T^{ff}(b,c)$

- We care about values 2, 3, and 4, but 4 is not a possible value at the source for 5, so we never ask about it.

Example: Chain Algorithm (cont'd)

- When we ask about x = 2, we get one response: y = 4.

- We pad this response with the value 2 we supplied to conclude that (2,4) is a tuple in the relation for the 5 subgoal.

Relation	R		S		T	
Data	w	x	x	y	y	z
	1	2	2	4	4	6
	1	3	3	5	5	7
	1	4			5	8
Adornment	*bf*		*c'*[2,3,5]*f*		*bu*	

Q: Answer(*c*) ← $R^{bf}(1,a)$ AND $S^{ff}(a,b)$ AND $T^{ff}(b,c)$

- Similarly, when we ask about x = 3, we get y = 5 as the only response and we add (3,5) to the set of tuples constructed for the S subgoal

Example: Chain Algorithm (cont'd)

- Now we resolve $S^{bf}(a,b)$:
  - Project **x** onto ***a***, resulting in ***X***.
  - Now, search *S* for tuples with attribute ***a*** equivalent to attribute ***a*** in *X*.

Q: Answer(c) ← $R^{bf}(1,a)$ AND $S^{ff}(a,b)$ AND $T^{ff}(b,c)$

a	b
2	4
3	5

Relation	R		S		T	
Data	w	x	x	y	y	z
	1	2	2	4	4	6
	1	3	3	5	5	7
	1	4			5	8
Adornment	*bf*		*c'*[2,3,5]*f*		*bu*	

- Join this relation with ***X***, and remove ***a*** because it doesn't appear in the head nor any unresolved subgoal:

b
4
5

Example: Chain Algorithm (cont'd)

- Now we resolve $T^{bf}(b,c)$:

b	c
4	6
5	7
5	8

Q: Answer(c) ← $R^{bf}(1,a)$ AND $S^{ff}(a,b)$ AND $T^{ff}(b,c)$

Relation	R		S		T	
Data	w	x	x	y	y	z
	1	2	2	4	4	6
	1	3	3	5	5	7
	1	4			5	8
Adornment	*bf*		*c'*[2,3,5]*f*		*bu*	

- Join this relation with ***X*** and project onto the ***c*** attribute to get the relation for the head.

- That is, the answer to the query at the mediator is {(6), (7), (8)}.

Incorporating Union Views at the Mediator

- In our description of the Chain Algorithm we assumed that each query subgoal was a “view” of data at one particular source.
- It is common for there to be several sources that can contribute tuples.
- If specific sources have tuples to contribute that other sources may not have, it adds complexity.
- To resolve this, we can consult all sources, or make best efforts to return all the answers.

Incorporating Union Views at the Mediator (cont'd)

- If more than one sources are available:
  - The sources may contain replicated information. In that case, we can turn to anyone of the sources. However, there may be several adornments that allow us to query that source.
  - Each source contributes some tuples that the other sources may not contribute. In that case, we should consult all the sources for the predicate.
    - Less practical because it makes queries harder to answer and impossible if any source is down.
- There is a policy choice to be made:
  - Either we can refuse to answer the query unless we can consult all the sources
  - Or we can make **best efforts** to return all the answers to the query that we can obtain by combinations of sources.

Example: Best Effort (Union Views)

- Suppose we have the following query:

  Answer(a,c) ← R^{ff}(a,b) AND S^{ff}(b,c)

- We have the following sources for ***R: R_1^{ff} and R_2^{fb}***

- We have the following sources for ***S: S_1^{ff} and S_2^{bf}***

- Suppose we start with ***R***'s source. We query this source and get some tuples for ***R(using R_1^{ff}).***

- Now, we have some bindings, but perhaps not all, for the variable ***b***.

Example: Best Effort (Union Views) (cont'd)

- We can now use both sources for ***S*** to obtain tuples and the relation for ***S*** can be set to their union.

Answer(a,c) ← R^{ff}(a,b) AND S^{ff}(b,c)

- At this point, we can project the relation for ***S*** onto variable ***b*** and get some ***b***-values. These can be used to query the second source for ***R***, the one with adornment ***fb***.

- In this manner, we can get some additional ***R***-tuples. It is only at this point that we can join the relations for ***R*** and ***S***, and project onto ***a*** and ***c*** to get the best-effort answer to the query

GAV & LAV Mediators

- The mediators discussed so far are called **global-as-view** (**GAV**) mediators.

- **GAV** mediators are easy to construct. You decide on the global predicates or relations that the mediator will support, and for each source, you consider which predicates it can support, and how it can be queried.

- In a **local-as-view** (**LAV**) mediator, we define global predicates at the mediator, but we **do not** define these predicates as views of the source data.

GAV & LAV Mediators (cont'd)

- Rather, we define, for each source, one or more expressions involving the global predicates that describe the tuples that the source is able to produce.

- Queries are answered at the mediator by discovering all possible ways to construct the query using the views provided by the sources.

GAV & LAV Mediators (cont'd)

Two Mediation Approaches

1. ***Global as View*** : Mediator processes queries into steps executed at sources.
2. ***Local as View*** : Sources are defined in terms of global relations; mediator finds all ways to build query from views.

Example: Catalog Integration

- Suppose a tech company wants to buy a bus and a disk that share the same protocol.

- **Global schema**: `Buses(manf,model,protocol)`
  `Disks(manf,model,protocol)`

- **Local schemas**: each bus or disk manufacturer has a (`model,protocol`) relation --- `manf` is implied.

Example: Global-as-View

- Mediator might start by querying each bus manufacturer for model-protocol pairs.
  - The wrapper would turn them into triples by adding the manf component.
- Then, for each protocol returned, mediator queries disk manufacturers for disks with that protocol.
  - Again, wrapper adds manf component.

Example: Local-as-View

- Sources' capabilities are defined in terms of the global predicates.
  - E.g.,Quantum's disk database could be defined by QuantumView(M,P) = Disks('Quantum',M,P).

- Mediator discovers all combinations of a bus and disk "view," equijoined on the protocol components.

Example: LAV

- We shall look at an example where the mediator is intended to provide a single predicate ***Par(c,p)***, meaning that ***p*** is a parent of ***c***.
- Suppose we have a database that provides some parent facts (DS_1). And a an another database (DS_2), maintained by the Association of Grandparents, that supports only grandparent facts
- **GAV** mediators do not allow us to use a grandparents source at all, if our goal is to produce a ***Par*** relation. However, **LAV** mediators allow us to say that a certain source provides grandparent facts.

Example: LAV (cont'd)

- The DS_1 can be described as:

$$V_1(c,p) \leftarrow Par(c,p)$$

- The DS_2 can be described as:

$$V_2(c,g) \leftarrow Par(c,p) \text{ AND } Par(p,g)$$

- Our query at the mediator will ask for great-grandparent facts that can be obtained from the sources. The mediator query is:

$$ggp(c,x) \leftarrow Par(c,u) \text{ AND } Par(u,v) \text{ AND } Par(v,x)$$

- How can the sources provide solutions that provide all available answers?

Example: LAV (cont'd)

- Our query at the mediator will ask for great-grandparent facts that can be obtained from the sources. That is, the mediator query is

ggp(c,x) ← Par(c,u) AND Par(u,v) AND Par(v,x)

- The possible solutions are:
  - Using only V_1 : **ggp(c,x) ← V_1(c,u) AND V_1(u,v) AND V_1(u,x)**
  - Using a combination of V_1 and V_2 :
    - Alternative 1: **ggp (c,x) ← V_1(c,u) AND V_2(u,x)**
    - Alternative 2: **ggp(c,x) ← V_2(c,v) AND V_1(v,x)**

- No other queries involving the views can provide more ggp facts.
- Deep theory needed to explain.

Comparison: LAV Vs. GAV

- GAV is simpler to implement.
  - Lets you control what the mediator does.

- LAV is more extensible.
  - Add a new source simply by defining what it contributes as a view of the global schema.
  - Can get some use from grandparent info., even if par(c,p) is the only mediator data.