1 Introduction

The invention of the Internet and the emergence of the World Wide Web revolutionized people's access to digital data stored on electronic devices. Today, we take for granted the ability to specify a search query into a browser or smartphone and tap into millions of documents and into databases of local businesses, recommendations, and coupon offers. Similarly, we also assume we can order a computer configured just the way we want and receive it within *days*, even if the computer requires assembling parts from dozens of companies scattered around the world. To provide such services, systems on the Internet must efficiently and accurately process and serve a significant amount of data. But unlike traditional data management applications, such as maintaining a corporation's payroll, the new services require the ability to *share* data among multiple applications and organizations, and to *integrate* data in a flexible and efficient fashion. This book covers the principles of *data integration*, a set of techniques that enable building systems geared for flexible sharing and integration of data across multiple autonomous data providers.

1.1 What Is Data Integration?

We illustrate the need for data integration with two examples, one representing a typical scenario that may occur within a company and the other representing an important kind of search on the Web.

Example 1.1

Consider FullServe, a company that provides Internet access to homes, but also sells a few products that support the home computing infrastructure, such as modems, wireless routers, voice-over-IP phones, and espresso machines. FullServe is a predominantly American company and recently decided to extend its reach to Europe. To do so, FullServe acquired a European company, EuroCard, which is mainly a credit card provider, but has recently started leveraging its customer base to enter the Internet market.[1]

The number of different databases scattered across a company like FullServe could easily reach 100. A very simplified version of FullServe's database collection is shown in Figure 1.1. The Human Resources Department has a database storing information about each employee,

[1]The names of these companies and the combination of businesses in which they are involved are purely fictional, but not unimaginable in today's economy.

Principles of Data Integration, DOI: 10.1016/B978-0-12-416044-6.00001-6
Copyright © 2012 by Elsevier Inc. All rights reserved.

Employee Database
FullTimeEmps(ssn, empID, firstName, middleName, lastName)
Hire(empID, hireDate, recruiter)
TempEmployees(ssn, hireStart, hireEnd, name, hourlyRate)

Resume Database
Interviews(interviewDate, pID, recruiter, hireDecision, hireDate)
CVs(ID, resume)

Training Database
Courses(courseID, name, instructor)
Enrollments(courseID, empID, date)

Services Database
Services(packName, textDescription)
Customers(name, ID, zipCode, streetAdr, phone)
Contracts(custID, packName, startDate)

Sales Database
Products(prodName, prodID)
Sales(prodID, customerID, custName, address)

HelpLine Database
Calls(date, agent, custID, text, action)

FIGURE 1.1 Some of the databases a company like FullServe may have. For each database, we show some of the tables and for each table, some of its attributes. For example, the Employee database has a table FullTimeEmps with attributes ssn, empID, firstName, middleName, and lastName.

separating full-time and temporary employees. They also have a separate database of resumes of each of their employment candidates, including their current employees. The Training and Development Department has a database of the training courses (internal and external) that each employee went through. The Sales Department has a database of services and its current subscribers and a database of products and their customers. Finally, the Customer Care Department maintains a database of calls to their help-line center and some details on each such call.

Upon acquiring EuroCard, FullServe also inherited their databases, shown in Figure 1.2. EuroCard has some databases similar to those of FullServe, but given their different location and business focus, there are some obvious differences.

There are many reasons why data reside in multiple databases throughout a company, rather than sitting in one neatly organized database. As in the case of FullServe and EuroCard, databases are acquired through mergers and acquisitions. When companies go through internal restructuring they don't always align their databases. For example, while the services and products divisions of FullServe are currently united, they probably didn't start out that way, and therefore the company has two separate databases. Second, most databases originate from a particular group in a company that has an information need at a particular point in time. When the database is created, its authors cannot anticipate all information needs of the company in the future, and how the data they are producing today may be used differently at some other point in time. For example, the Training database at FullServe may have started as a small project by a few employees in the company to keep track of who attended certain training sessions in the company's early days. But as the company grew, and the Training and Development Department was created, this database needed to be broadened quite a bit. As a result

Employee Database
Emp(ID, firstNameMiddleInitial, lastName, salary)
Hire(ID, hireDate, recruiter)

Resume Database
Interviews(ID, date, location, recruiter)
CVs(candID, resume)

Credit Card Database
Cards(CustID, cardNum, expiration, currentBalance)
Customers(CustID, name, address)

HelpLine Database
Calls(date, agent, custID, description, followup)

FIGURE 1.2 Some of the databases of EuroCard. Note that EuroCard organizes its data quite differently from FullServe. For example, EuroCard does not distinguish between full-time and part-time employees. FullServe records the hire data of employees in the Resume database and the Employee database, while EuroCard only records the hire date in the Employee database.

of these factors and others, large enterprises typically have dozens, if not hundreds, of disparate databases.

Let us consider a few example queries that employees or managers in FullServe may want to pose, all of which need to span *multiple* databases.

- Now that FullServe is one large company, the Human Resources Department needs to be able to query for all of its employees, whether in the United States or in Europe. Because of the acquisition, data about employees are stored in multiple databases: two databases (for employees and for temps) on the American side of the company and one on the European side.
- FullServe has a single customer support hotline, which customers can call about any service or product they obtain from the company. It is crucial that when a customer representative is on the phone with a customer, he or she sees the entire set of services the customer is getting from FullServe, whether it be Internet service, credit card, or products purchased. In particular, it is useful for the representative to know that the customer on the phone is a big spender on his or her credit card, even if he or she is calling about a problem with his or her Internet connection. Obtaining such a complete view of the customer requires obtaining data from at least three databases, even in our simple scenario.
- FullServe wants to build a Web site to complement its telephone customer service line. On the Web site, current and potential customers should be able to see all the products and services FullServe provides, and also select bundles of services. Hence, a customer must be able to see his or her current services and obtain data about the availability and pricing of any other services. Here, too, we need to tap into multiple databases of the company.
- To take the previous example further, suppose FullServe partners with a set of other vendors to provide *branded* services. That is, you can get a credit card issued by your favorite sports team, but the credit card is actually served by FullServe. In this case, FullServe needs to provide a Web service that will be accessed by other Web sites (e.g., those of the sports teams) to provide a single login point for customers. That Web service needs to tap into the appropriate databases at FullServe.
- Governments often change reporting or ethic laws concerning how companies can conduct business. To protect themselves from possible violations, FullServe may want to be proactive. As

a first step, the company may want to be aware of employees who've worked at competing or partner companies prior to joining FullServe. Answering such a query would involve combining data from the Employee database with the Resume database. The additional difficulty here is that resumes tend to be unstructured text, and not nicely organized data.

- Combining data from multiple sources can offer opportunities for a company to obtain a competitive advantage and find opportunities for improvement. For an example of the former, combining data from the HelpLine database and the Sales database will help FullServe identify issues in their products and services early on. Discovering trends in the usage of different products can enable FullServe to be proactive about building and maintaining inventory levels. Going further, suppose we find that in a particular area of the country FullServe is receiving an unusual number of calls about malfunctions in their service. A more careful look at this data may reveal that the services were installed by agents who had taken a particular training course, which was later found to be lacking. Finding such a pattern in the data requires combining data from the Training database, HelpLine database, and Services database, all residing in very different parts of the company.

Example 1.2

Consider a very different kind of example where data integration is also needed. Suppose you are searching for a new job, and you'd like to take advantage of resources on the Web. There are thousands of Web sites with databases of jobs (see Figure 1.3 for two examples). In each of these sites, you'd typically see a form that requires you to fill out a few fields (e.g., job title, geographical location of employer, desired salary level) and will show you job postings that are relevant to your query. Unfortunately, each form asks for a slightly different set of attributes. While the Monster form on the left of Figure 1.3 asks for job-related keywords, location, company, industry, and job category, the CareerBuilder form on the right allows you to select a location and job category from a menu of options and lets you further specify your salary range.

Consequently, going to more than a handful of such Web sites is tiresome, especially if you're doing this on a daily basis to keep up with new job postings. Ideally, you would go to a *single* Web site to pose your query and have that site integrate data from all relevant sites on the Web.

More generally, the Web contains millions of databases, some of which are embedded in Web pages and others that can be accessed through Web forms. They contain data in a plethora of domains, ranging from items typically found in classified ads and products to data about art, politics, and public records. Leveraging this incredible collection of data raises several significant challenges. First, we face the challenge of schema heterogeneity, but on a much larger scale: millions of tables created by independent authors and in over 100 languages. Second, extracting the data is quite difficult. In the case of data residing behind forms (typically referred to as the *Deep Web* or *Invisible Web*) we need to either crawl through the forms intelligently or be able to pose well-formed queries at run time. For data that are embedded in Web pages, extracting the tables from the surrounding text and determining its schema is

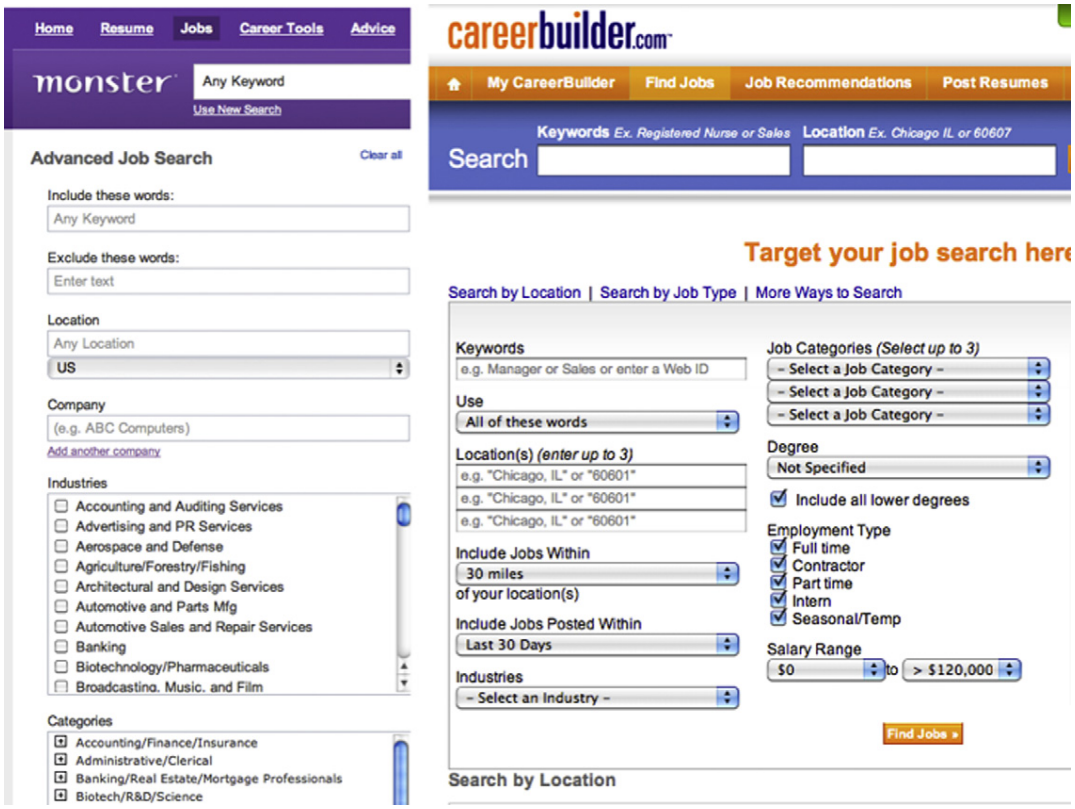

FIGURE 1.3 Examples of different forms on the Web for locating jobs. Note that the forms differ on the fields they request and formats they use.

challenging. Of course, data on the Web are often dirty, out of date, and even contradictory. Hence, obtaining answers from these sources requires a different approach to ranking and data combination.

While the above two examples illustrate common data integration scenarios, it is important to emphasize that the problem of data integration is pervasive. Data integration is a key challenge for the advancement of science in fields such as biology, ecosystems, and water management, where groups of scientists are independently collecting data and trying to collaborate with one another. Data integration is a challenge for governments who want their different agencies to be better coordinated. And lastly, mash-ups are now a popular paradigm for visualizing information on the Web, and underlying every mash-up is the need to integrate data from multiple disparate sources.

To summarize, the goal of a data integration system is to offer uniform access to a set of autonomous and heterogeneous data sources. Let us expand on each of these:

- **Query**: The focus of most data integration systems is on querying disparate data sources. However, updating the sources is certainly of interest.
- **Number of sources:** Data integration is already a challenge for a small number of sources (fewer than 10 and often even 2!), but the challenges are exacerbated when the number of sources grows. At the extreme, we would like to support Web-scale data integration.
- **Heterogeneity:** A typical data integration scenario involves data sources that were developed independently of each other. As a consequence, the data sources run on different systems: some of them are databases, but others may be content management systems or simply files residing in a directory. The sources will have different schemata and references to objects, even when they model the same domains. Some sources may be completely structured (e.g., relational databases), while others may be unstructured or semi-structured (e.g., XML, text).
- **Autonomy:** The sources do not necessarily belong to a single administrative entity, and even when they do, they may be run by different suborganizations. Hence, we cannot assume that we have full access to the data in a source or that we can access the data whenever we want, and considerable care needs to be given to respecting the privacy of the data when appropriate. Furthermore, the sources can change their data formats and access patterns at any time, without having to notify any central administrative entity.

1.2 Why Is It Hard?

In order to approach the problem of data integration effectively, it is important to first examine the reasons for which it is hard. Later in the book we cover these challenges in more detail, but here we give a high-level description of the challenges. We classify the reasons into three different classes: systems reasons, logical reasons, and social reasons. We examine each one separately.

1.2.1 Systems Reasons

The systems challenges that occur in data integration are quite obvious and appear very early in the process of trying to build an integration application. Fundamentally, the challenge is to enable different systems to talk seamlessly to one another. Even assuming that all systems are running on the same hardware platform and all the sources are relational database systems supporting SQL standard and ODBC/JDBC, the problem is already not easy. For example, while SQL is a standard query language for relational databases, there are some differences in the way different vendors implement it, and these differences need to be reconciled.

Executing queries over multiple systems efficiently is even more challenging. Query processing in distributed databases (i.e., databases where the data are partitioned into multiple nodes) is already a hard problem. The saving grace in distributed databases is that the data are *a priori* distributed to the different nodes by one entity and in an organized and known fashion. In data integration we are faced with a preexisting collection of sources, and the organization of the data in the sources is much more complex and not necessarily well known. Furthermore, the capabilities of each of the data sources in terms of the query processing powers they have can be very different. For example, while one source may be a full SQL engine and therefore may be able to accept very complex queries, another source may be a Web form and therefore only accept a very small number of query templates.

1.2.2 Logical Reasons

The second set of challenges has to do with the way data are logically organized in the data sources. For the most part, structured data sources are organized according to a schema. In the common case of relational databases, the schema specifies a set of tables, and for each table a set of attributes with their associated data types. In other data models the schema is specified by tags, classes, and properties.

Human nature is such that if two people are given *exactly* the same requirements for a database application, they will design very different schemata.[2] Hence, when data come from multiple sources, they typically look very different.

We can see several differences when we compare the schemata of the databases of FullServe and EuroCard:

- EuroCard models temporary employees and full-time employees in the same database table, while FullServe maintains two separate tables. This may be because FullServe uses an outside agency to contract and manage its temporary employees.
- Even when modeling employees, FullServe and EuroCard do so differently and cover slightly different attributes. For example, EuroCard uses IDs (corresponding to national ID cards) to identify employees, while FullServe records the social security number but also assigns an employee ID (since social security numbers may not be used as IDs in certain cases, for privacy reasons). FullServe records the hireDecision and hireDate attributes in their Resume database, but EuroCard simply assumes these attributes can be obtained by querying the Employee database with the appropriate ID. On the other hand, EuroCard records *where* each interview was conducted, whereas FullServe does not.
- Even when modeling the exact same attribute, FullServe and EuroCard may use different attribute names. For example, the HelpLine database of FullServe uses the attributes text and action to record the same as the attributes description and followup in EuroCard's database.

[2] This aspect of human nature is well known to database professors and is often used to detect cheating.

Finally, the representation of the data can be significantly different as well. For example, in FullServe's Employee database, the company records the first, middle, and last name of every employee in a separate field. In the Training database, it only records the full name of the employee in a single field, and as a result, names often occur in various formats, such as (First, Last) or (Last, First Initial). Consequently, it may be hard to match records from these two databases. The same problem occurs between the Sales and Services databases. While the Sales database only records a field for the name and a field for the address, the Services database records much finer-grained information about each subscriber. Of course, the units of measure will differ as well: FullServe uses American dollars for prices, while EuroCard uses Euros. Since the exchange rate between the currencies constantly changes, the correspondence between the values cannot be set *a priori*.

Data from multiple sources can only be integrated if we bridge this so-called *semantic heterogeneity*. In fact, semantic heterogeneity turns out to be a major bottleneck for data integration.

1.2.3 Social and Administrative Reasons

The last set of reasons we mention are not so technical as the previous ones, but are often as hard and can easily be the reason a data integration project fails. The first challenge may be to *find* a particular set of data in the first place. For example, it may turn out that EuroCard did not save the resumes of their employees electronically, and a special effort is required to locate and scan them all.

Even when we know where all the data are, owners of the data may not want to cooperate with an integration effort. When owners come from different companies or universities, there are some obvious reasons for not wanting to share, but even within an enterprise, owners are often reluctant. In some cases, the reason is that their data are part of a carefully tuned and mission-critical function of the enterprise. Allowing additional queries from the data integration system can put a prohibitively heavy load on their system. In other cases, the reason is that some data can only be viewed by particular people within the enterprise, and the data owners are worried (for a good reason!) that the data integration system will not be able to enforce these restrictions. Finally, in some cases people create *data fiefdoms* — here, access to the data means more power within the organization. For example, the head of the Sales Department may not want to share the data on sales representatives' work, as it may reveal some internal problems.

It is worth noting that in a few circumstances — for instance, those involving medical records or law enforcement — there may be legitimate legal reasons why a data owner cannot share data. Because such situations arise, the problem of anonymizing data remains a hot topic in computer science research.

While we cannot expect to solve the social problems with technology alone, technology can help data owners capitalize on some of the benefits of data integration, thereby providing additional incentives to participate. For example, an important benefit of participating in a data integration project is that one's data can be reached by many more

people and have a broader impact (e.g., if the data are included in relevant search results by a Web search engine). As another example, a data integration system can be designed so that the attribution of the data is always clear (even when results are assembled from multiple sources), therefore ensuring that appropriate credit is given to the data owners.

1.2.4 Setting Expectations

Data integration is a hard problem, and some say it will never be completely solved (thereby guaranteeing a steady stream of readers for this book). Before we can discuss solutions to different aspects of data integration, it is important to set the expectations appropriately.

In the ideal world, we would like to provide a data integration system access to a set of data sources and have the system automatically configure itself so that it can correctly and efficiently answer queries that span multiple sources. Since this ideal goal is unlikely to be achieved, we focus on two more modest goals. Our first goal is to build tools that *reduce the effort* required to integrate a set of data sources. For example, these tools should make it easy to add a new data source, relate its schema to others, and automatically tune the data integration system for better performance.

Our second goal is to improve the ability of the system to answer queries in uncertain environments. Clearly, if the data integration system must always return correct and complete answers, then uncertainty cannot be tolerated at all. But in applications where data integration facilitates exploratory efforts (e.g., on the Web), the system should be able to answer queries under uncertainty. For example, when we look for jobs on the Web, it's okay to find 29 out of 30 relevant sources, or to return answers that don't completely satisfy the user query.

In some sense, user effort and accuracy are competing goals in setting up a data integration application. The more time we spend, the more accurate our system is going to be. Hence, in cases where we are willing to trade off lower accuracy for decreased setup time and labor, our goal can be stated as follows: *reduce the user effort needed to obtain high-quality answers from a data integration system.*

1.3 Data Integration Architectures

As background for the discussion later in the book, we briefly describe the architecture of a data integration system. There are a variety of possible architectures for data integration, but broadly speaking, most systems fall somewhere on the spectrum between *warehousing* and *virtual integration*. In the warehousing end of the spectrum, data from the indvidual data sources are loaded and materialized into a physical database (called a *warehouse*), where queries over the data can be answered. In virtual integration, the data remain in the sources and are accessed as needed at query time. Despite the differences in approach, many of the hard challenges are shared across these architectures.

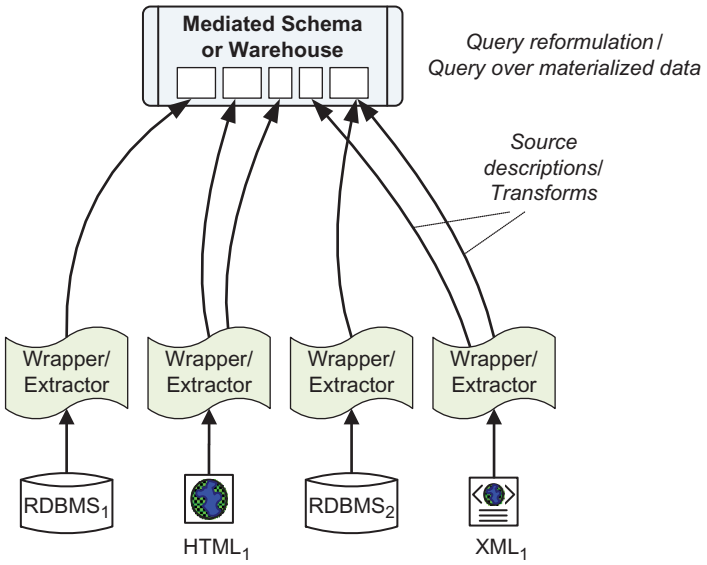

FIGURE 1.4 The basic architecture of a general-purpose data integration system. Data sources can be relational, XML, or any store that contains structured data. The *wrappers* or *loaders* request and parse data from the sources. The *mediated schema* or central *data warehouse* abstracts all source data, and the user poses queries over this. Between the sources and the mediated schema, *source descriptions* and their associated *schema mappings*, or a set of *transformations*, are used to convert the data from the source schemas and values into the global representation.

1.3.1 Components of the Data Integration System

Figure 1.4 shows the logical components of both kinds of data integration systems. We now describe each of these components, beginning with the components used in the virtual approach to integration. We contrast the warehouse model afterwards.

On the bottom of the figure we see the *data sources*. Data sources can vary on many dimensions, such as the data model underlying them and the kinds of queries they support. Examples of structured sources include database systems with SQL capabilities, XML databases with an XQuery interface, and sources behind Web forms that support a limited set of queries (corresponding to the valid combinations of inputs to its fields). In some cases, the source can be an actual application that is driven by a database, such as an accounting system. In such a case, a query to the data source may actually involve an application processing some data stored in the source.

Above the data sources are the programs whose role is to communicate with the data sources. In virtual data integration, these programs are called *wrappers*, and their role is to send queries to a data source, receive answers, and possibly apply some basic transformations on the answer. For example, a wrapper to a Web form source would accept a query and translate it into the appropriate HTTP request with a URL that poses the query on

the source. When the answer comes back in the form of an HTML file, the wrapper would extract the tuples from the HTML file.

The user interacts with the data integration system through a single schema, called the *mediated schema*. The mediated schema is built for the data integration application and contains *only* the aspects of the domain that are relevant to the application. As such, it does not necessarily contain all the attributes we see in the sources, but only a subset of them. In the virtual approach the mediated schema is not meant to store any data. It is purely a logical schema that is used for posing queries by the users (or applications) employing the data integration system.

The key to building a data integration application is the *source descriptions*, the glue that connects the mediated schema and the schemas of the sources. These descriptions specify the properties of the sources that the system needs to know in order to use their data. The main component of source descriptions is the *semantic mappings*, which relate the schemata of the data sources to the mediated schema. The semantic mappings specify how attributes in the sources correspond to attributes in the mediated schema (when such correspondences exist), and how the different groupings of attributes into tables are resolved. In addition, the semantic mappings specify how to resolve differences in how data values are specified in different sources. It is important to emphasize that the virtual data integration architecture only requires specifying mappings between the data sources and the mediated schema and not between every *pair* of data sources. Hence, the number of mappings we specify is the same as the number of sources and not the square of that number. Furthermore, the semantic mappings are specified *declaratively*, which enables the data integration system to reason about the contents of the data sources and their relevance to a given query and optimize the query execution.

In the warehousing approach, instead of a mediated schema the user poses queries in terms of the warehouse schema. In addition to being a schema that contains the necessary attributes from the sources, the warehouse schema is also a physical schema with an underlying database instance. Instead of wrappers, the system includes *ETL* or extract-transform-load tool pipelines that periodically extract data from the sources and load them into the warehouse. Unlike wrappers, ETL tools typically apply more complex transformations to the data that may involve cleaning, aggregation, and value transformations. These transformations play the role of schema mappings in the virtual data integration architecture, but tend to be more procedural in nature.

Some of the properties of data warehousing stem from the fact that these systems were not originally developed for the purpose of data integration. Instead, they were developed as a tool for performing deeper analysis, such as uploading data from transactional systems (e.g., databases recording every sale made at a store) into a database where the data is aggregated and cleaned so decision support queries can be posed (e.g., sales of particular products by region). Converting data from the transactional systems to the warehouse can involve rather sophisticated transformations and aggregation. We discuss data warehousing and some of its variants in detail in Chapter 10, but the vast majority of the book discusses data integration in terms of the

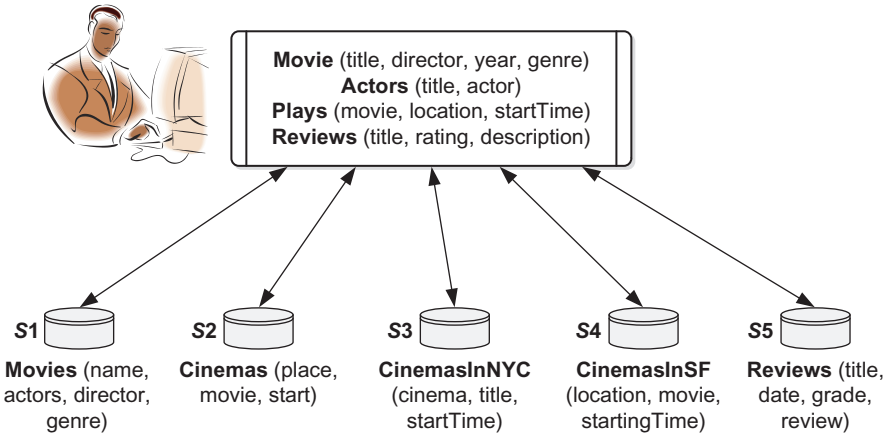

FIGURE 1.5 An example data integration scenario. Queries about movies are answered using a combination of sources on movie details, cinema listings, and review databases.

virtual data integration approach because it best illustrates the main concepts of data integration.

1.3.2 Example Data Integration Scenario

The following example, shown in Figure 1.5, illustrates a complete data integration scenario.

Data Sources and Mediated Schema

In the example, we have five data sources. The first one on the left, S1, stores data about movies, including their names, actors, director, and genre. The next three sources, S2-S4, store data about showtimes. Source S2 covers the entire country, while S3 and S4 consider only cinemas in New York City and San Francisco, respectively. Note that although these three sources store the same type of data, they use different attribute names. The rightmost source, S5, stores reviews about movies.

The mediated schema includes four relations, Movie, Actors, Plays, and Reviews. Note that the Review in the mediated schema does not contain the date attribute, but the source storing reviews does contain it.

The semantic mappings in the source descriptions describe the relationship between the sources and the mediated schema. For example, the mapping of source S1 will state that it contains movies, and that the attribute name in Movies maps to the attribute title in the Movie relation of the mediated schema. It will also specify that the Actors relation in the mediated schema is a *projection* of the Movies source on the attributes name and actors.

Similarly, the mappings will specify that tuples of the Plays relation in the mediated schema can be found in S2, S3, or S4, and that the tuples in S3 have their location city set to New York (and similarly for San Francisco and S4).

In addition to the semantic mappings, the source descriptions also specify other aspects of the data sources. First, they specify whether the sources are *complete* or not. For example, source S2 may not contain all the movie showtimes in the entire country, while source S3 may be known to contain *all* movie showtimes in New York. Second, the source descriptions can specify limited access patterns to the sources. For example, the description of S1 may specify that in order to get an answer from the source, there needs to be an input for at least one of its attributes. Similarly, all the sources providing movie playing times also require a movie title as input.

Query Processing

We pose queries to the data integration system using the terms in the mediated schema. The following query asks for showtimes of movies playing in New York and directed by Woody Allen.

```
SELECT title, startTime
FROM Movie, Plays
WHERE Movie.title = Plays.movie AND location="New York"
  AND director="Woody Allen"
```

As illustrated in Figure 1.6, query processing proceeds in the following steps.

Query reformulation. As described earlier, the user query is posed in terms of the relations in the mediated schema. Hence, the first step the system must do is *reformulate* the query into queries that refer to the schemas of the data sources. To do so, the system uses the source descriptions. The result of the reformulation is a set of queries that refer to the schemata of the data sources and whose combination will yield the answer to the original query. We refer to the result of reformulation as a *logical query plan*.

The following would be derived during reformulation:

- Tuples for Movie can be obtained from source S1, but the attribute title needs to be reformulated to name.
- Tuples for Plays can be obtained from either source S2 or S3. Since the latter is complete for showings in New York City, we choose it over S2.
- Since source S3 requires the title of a movie as input, and such a title is not specified in the query, the query plan must first access source S1 and then feed the movie titles returned from S1 as inputs to S3.

Hence, the first logical query plan generated by the reformulation engine accesses S1 and S3 to answer the query. However, there is a second logical query plan that is also correct (albeit possibly not complete), and that plan accesses S1 followed by S2.

Query optimization. The next step in query processing is query optimization, as in traditional database systems. Query optimization takes as input a logical query plan and produces a *physical query plan*, which specifies the exact order in which the data sources are accessed, when results are combined, which algorithms are used for performing operations on the data (e.g., join between sources), and the amount of resources allotted to

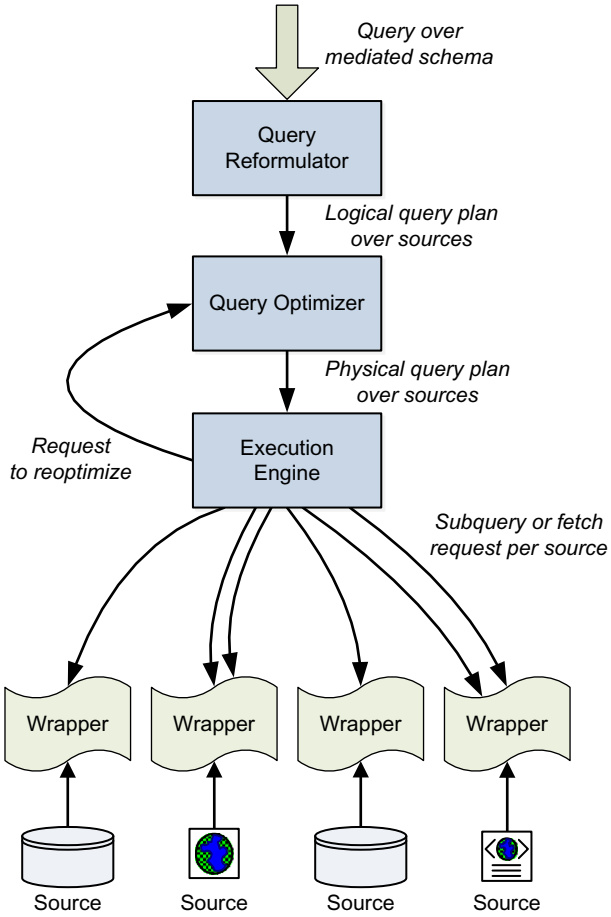

FIGURE 1.6 Query processing in a data integration system differs from traditional database query processing in two main ways. First, the query needs to be reformulated from the mediated schema to the schema of the sources. Second, query execution may be adaptive in that the query execution plan may change as the query is being executed.

each operation. As described earlier, the system must also handle the challenges that arise from the distributed nature of the data integration system.

In our example, the optimizer will decide which join algorithm to use to combine results from S1 and S3. For example, the join algorithm may stream movie titles arriving from S1 and input them into S3, or it may batch them up before sending them to S3.

Query execution. Finally, the execution engine is responsible for the actual execution of the physical query plan. The execution engine dispatches the queries to the individual sources through the wrappers and combines the results as specified by the query plan.

Herein lies another significant difference between a data integration system and a traditional database system. Unlike a traditional execution engine that merely executes

the query plan given to it by the optimizer, the execution engine of a data integration system may decide to ask the optimizer to reconsider its plan based on its monitoring of the plan's progress. In our example, the execution engine may observe that source `S3` is unusually slow and therefore may ask the optimizer to generate a plan that includes an alternate source.

Of course, an alternative would be for the optimizer to already build certain contingencies into the original plan. However, if the number of unexpected execution events is large, the original plan could grow to an enormous size. Hence, one of the interesting technical challenges in designing the query processing engine is how to balance the complexity of the plan and its ability to respond to unexpected execution events.

1.4 Outline of the Book

The remainder of this book will elaborate on each of the components and processes described above. The following outlines some of the main topics covered by each chapter.

Chapter 2 lays a few theoretical foundations that are needed for later discussions. In particular, the chapter describes algorithms for manipulating and reasoning about query expressions. These algorithms enable us to determine whether one query is *equivalent* to a second query even if they are written differently and to determine whether a query can be answered from previously computed views on the database. These algorithms will be essential for query reformulation and optimization in data integration, but are also useful in other data management contexts.

Chapter 3 describes the formalisms proposed for specifying source descriptions and in particular the semantic mappings. We describe two languages that use query expressions for specifying semantic mappings—Global-as-View (GAV) and Local-as-View (LAV)—and the GLAV language that combines the two. For each of these languages, we describe the appropriate query reformulation algorithms. We also describe how we can handle information about source completeness and about limitations on accessing data in sources.

In Chapter 4 we begin our discussion of techniques for *creating* semantic mappings. As it turns out, creating mappings is one of the main bottlenecks in building data integration applications. Hence, our focus is on techniques that reduce the time required from a person to create mappings. Chapter 4 discusses the fundamental problem of determining whether two strings refer to the same real-world entity. String matching plays a key role in matching data and schema from multiple sources. We describe several heuristics for string matching and methods for scaling up these heuristics to large collections of data. We then discuss the problems of creating mappings at the schema level (Chapter 5). One of the interesting aspects of these techniques is that they leverage methods from machine learning, enabling the system to improve over time as it sees more correct schema and object matchings.

Beyond the problem of establishing mappings between schemas, there are a variety of other operations one performs on metadata. Chapter 6 discusses general schema manipulation or *model management operators*, which can be incredibly useful in comparing and composing schemas and mappings.

Next we consider techniques for identifying mappings or entity correspondences at the data level (Chapter 7). Again, as with the schema mapping problem, machine learning techniques prove useful.

Chapter 8 discusses query processing in data integration systems. The chapter describes how queries are optimized in data integration systems and query operators that are especially appropriate for this context. The important new concept covered in this chapter is *adaptive query processing*, which refers to the ability of a query processor to change its plan during execution.

Chapter 9 then discusses how Web information extractors or *wrappers* are constructed, in order to acquire the information to be integrated. Wrapper construction is an extremely challenging problem, especially given the idiosyncrasies of real-world HTML. It typically requires a combination of heuristic pattern matching, machine learning, and user interaction.

Chapter 10 reviews *data warehousing* and its variants. The main advantage of data warehousing is that it can support complex queries more efficiently. We also discuss *data exchange*, an architecture in which we have a *source* database and our goal is to translate the data and store them in a *target* database that has a different schema, and answer queries over the target database.

?? of the book focuses on richer ways of representing the data, incorporating hierarchy, class relationships, and annotations. Chapter 11 discusses the role of XML in data integration. XML has played an important role in data integration because it provides a syntax for sharing data. Once the syntax problem was addressed, people's appetites for sharing data in semantically meaningful ways were whetted. This chapter begins by covering the XML data model and query language (XQuery). We then cover the techniques that need to be developed to support query processing and schema mapping in the presence of XML data.

Chapter 12 discusses the role of knowledge representation (KR) in data integration systems. Knowledge representation is a branch of artificial intelligence that develops languages for representing data. These languages enable representing more sophisticated constraints than are possible in models employed by database systems. The additional expressive power enables KR systems to perform sophisticated reasoning on data. Knowledge representation languages have been a major force behind the development of the *Semantic Web*, a set of techniques whose goal is to enrich data on the Web with more semantics in order to ultimately support more complex queries and reasoning. We cover some of the basic formalisms underlying the Semantic Web and some of the challenges this effort is facing.

Chapter 13 discusses how to incorporate uncertainty into data integration systems. When data are integrated from multiple autonomous sources, the data may not all be

correct or up to date. Furthermore, the schema mappings and the queries may also be approximate. Hence, it is important that a data integration system be able to incorporate these types of uncertainty gracefully. Then Chapter 14 describes another form of annotation: data provenance that "explains," for every tuple, how the tuple was obtained or derived. We describe the close relationship between provenance and probabilistic scores.

Finally, **??** discusses some new application contexts in which data integration is being used and the challenges that need to be addressed in them. Chapter 15 discusses the kinds of structured data that are present on the Web and the data integration opportunities that arise there. The Web offers unique challenges because of the sheer number and diversity of data sources.

Chapter 16 considers how one can combine ideas from keyword search and machine learning to provide a more lightweight form of integration "on demand": even in the absence of preexisting mappings and mediated schemas, a system can find ways of joining sources' data to "connect" data values matching search terms, thus answering ad hoc queries. The system might even use machine learning techniques to refine its results. This is a form of "pay as you go" data integration that may be useful in domains where little administrator input is available.

Chapter 17 describes a peer-to-peer (P2P) architecture for data sharing and integration. In the P2P architecture, we don't have a single mediated schema, but rather a loose collection of collaborating peers. In order to join a P2P data integration system a source would provide semantic mappings to *some* peer already in the system, the one for which it is most convenient. The main advantage of the P2P architecture is that it frees the peers from having to agree on a mediated schema in order to collaborate.

Chapter 18 describes how ideas from Web-based data integration and P2P data sharing have been further extended to support collaborative exchange of data. Key capabilities in collaborative systems include the ability to make annotations and updates to shared views of data.

Finally, we conclude with a brief look at some of the directions that seem especially open for high-impact developments in the data integration field.

Bibliographic Notes

Data integration has been a subject of research since the early 1980s, beginning with the Multi-Base System [366]. Since then, there has been considerable research and commercial development in the area. The paper on *mediators* [571] helped fuel the field and ultimately obtain more government funding for research in the area. The emergence of the World Wide Web and the large number of databases available online led to considering large-scale data integration [229]. Özsu and Valduriez provide a book-length treatment of distributed databases [471].

Since the late 1990s, data integration has been a well-established field in the enterprise market typically referred to as *Enterprise Information Integration* (EII) [285]. There

is also a significant market for data integration on the Web. In particular, numerous companies created domain-specific interfaces to a multitude of data sources in domains such as jobs, travel, and classifieds. These sites integrate data from hundreds to thousands of sources. Chaudhuri et al. [123] provide a good overview of the state of the art of business intelligence and of the data integration challenges that arise there.

This chapter offered only a very cursory introduction to the relational data model and query languages. We refer the reader to standard database textbooks for further reading on the topic [245, 489]. For a more theoretical treatment of query languages and integrity constraints, we refer the reader to [7], and for a comprehensive treatment of datalog, see [554].