# 10 Information Technology Systems

Shawn N. Murphy and Jeffrey G. Klann

**Learning Objectives**

At the end of the chapter, the reader will be able to:

- Describe the difference between structured and unstructured data
- Understand how data typically need to be changed to fit into a database
- Define the ACID concept of a database
- Describe the differences and tradeoffs between relational and non-relational systems, as well as cloud vs. on-premise databases
- Discuss the essential components of data interoperability, including Common Data Models and Health Information Exchange
- Identify the basic concepts behind Knowledge Discovery and Data Mining
- Cite various types of network topology
- Understand how a system architecture is represented
- Describe a three-tier software architecture
- Explain the design considerations in choosing a programming language, including compiled vs. interpreted and object-oriented vs. procedural
- List three software design considerations
- List four safeguards that HIPAA describes
- List three types of security attacks
- Describe how FISMA moderate compliance helps prevent security attacks

S. N. Murphy
Department of Neurology, Massachusetts General Hospital, Boston, MA, USA

Department of Biomedical Informatics, Harvard Medical School, Boston, MA, USA
e-mail: snmurphy@partners.org

J. G. Klann (✉)
Department of Medicine, Massachusetts General Hospital, Boston, MA, USA

Harvard Medical School, Boston, MA, USA
e-mail: jeff.klann@mgh.harvard.edu

**Practice Domains: Tasks, Knowledge, and Skills**

- K006. Computer programming fundamentals and computational thinking
- K007. Basic systems and network architecture
- K060. Enterprise architecture (databases, storage, application, interface engine)
- K062. Network communications infrastructure and protocols between information systems
- K076. Approaches to knowledge repositories
- K077. Data storage options and their implications
- K089: Data life cycle
- K090. Transactional and reporting/research databases
- K091. Techniques for the storage of disparate data types
- K092. Techniques to extract, transform, and load data
- K094. Data management and validation techniques
- K096. Types and uses of specialized and emerging data sources (e.g., imaging, bioinformatics, internet of things
- K098. Information architecture
- K099. Query tools and techniques
- K100. Flat files, relational and non-relational/NoSQL database structures, distributed file systems

**Case Vignette**

Jane is the CMIO of a large healthcare system and wants her enterprise to invest in a new electronic medical record (EMR) system. She will need to make a convincing argument, hoping to keep the technically-oriented CIO happy by showing the new system will indeed scale to the requirements of an upcoming merger with another health system. She would like to justify some of the claims made in the sales-oriented, splashy presentations of the EMR companies with her hard-hitting, factual presentation. It turns out the EMR companies are different in several ways. First, they use different types of databases. The first company uses a MUMPS hierarchical database, while the other companies use relational databases. The first company also uses a waterfall programming methodology, while the other com-

J. T. Finnell, B. E. Dixon (eds.), *Clinical Informatics Study Guide*, https://doi.org/10.1007/978-3-030-93765-2_10

panies use agile programming methodologies. One of the EMR companies is pushing a novel NoSQL-based system as part of its platform, but she doubts it can handle the transaction flow and wants to make that point to the CIO. Ultimately, Jane would like her health system to adopt an EMR with agile programming practices and a standards-based Application Programming Interface that uses a relational database system. How could she best present her arguments to the CIO? See if you can help Jane build her presentation as you navigate this chapter.

## Introduction

The events of clinical practice can be represented in an Information Technology (IT) system. Medical software is at the pinnacle of all IT system development in many ways, because these systems have a great responsibility towards the patient. Therefore, systems must be carefully designed to embody the following characteristics: sharing, proper formulation, quality measures, and fulfillment of use cases. Sharing includes proper authorization practices for those assessing the system, distinctions between data types that can be shared, and harmonization methods that allow sharing. The proper formulation includes focusing on data sources and data types, accounting for temporal aspects of clinical data, and accounting for various levels of data granularity and missingness in IT systems design. Quality measures for IT systems include working under many failure situations regarding data, code, and system security. This chapter will introduce practical decisions that must be made to formulate the components of a health IT system, including data, networks, and programs. We will carefully consider these characteristics as we discuss each component.

## Data and Databases

Data is at the heart of every health IT (HIT) system. The purpose of all HIT systems is gathering, storing, sharing, and utilizing data. This section will discuss the data itself, which will allow us to dive into further topics on building HIT systems, such as programming and system architecture, in later sections.

### Getting Data

#### Data Sources

HIT systems constantly generate data, which in the context of medical practice are pieces of information, especially those that are part of a collection to analyze a problem. In HIT parlance, these data fall into three broad categories:

- **Structured data** make up most of the information clinicians, and technicians enter into electronic health record (EHR) systems for record-keeping and billing purposes. Structured data are stored in various standard formats and terminologies (as discussed in Chap. 13) that computers can interpret and manipulate. As a rule, structured data come at the cost of clinicians' time and effort; these are not part of normal communication between clinicians that normally occurs with written unstructured discourse. However, structured data are much more useful to HIT systems for data processing.
  - *Examples of structured data:* billing data (e.g., diagnosis codes, procedure codes), demographic data, laboratory results, vital signs, and coded medication and problem lists.
- **Unstructured data** refer to data not stored in an easily computable format. Primarily this includes all the notes about a patient—from reports to discharge summaries, including data that may not be stored in a computer system at all (such as, in many environments, daily nursing notes). Images are often considered unstructured, as well as lab results that are supplied as fax documents. This category also includes some financial and legal data that are not readily available in computable format (such as consent forms, DNR orders, etc.). Unstructured data tend to be much richer than structured data, but they usually cannot be used directly in a computable environment such as a decision support system. Natural language processing (NLP) [1] is a way to extract computable meaning from this text. However, due to the many variations of how things can be said in human languages and how text is structured, NLP is fraught with difficulty and error-prone.
  - *Examples of unstructured data:* patient notes, financial and legal documents.
- **"Big" data** is an emerging category of data that are generally unstructured but is put in this separate category because it is difficult to process [2]. It is difficult, because the data has either an extremely large storage footprint (like radiology images or genomics from sequencing machines) or is so extraordinarily complex that it takes enormous computing resources. Sometimes these are data collected by continuous-monitoring machines. Home health monitoring (such as home blood glucose monitors) is an example of continuous monitoring data working into medical records.
  - *Examples of "big" data:* radiological images, genomic, and exomic data.

Another source of data besides HIT is patient-reported data and community information, as elaborated in Chaps. 24 and 25. Patient-reported data is used to reconcile the medical record with patient experiences and collect subjective information on patient perception of disease burden. Community information (such as public data about the number of parks in a city) is becoming more important as medical data is used for public health. Understanding local health policy, regional socioeconomic statuses, communicable diseases, and disease trends are becoming integrated into health data analysis.

### Interoperability: Mapping and ETL

Data are stored in many different systems throughout the hospital. To be retrieved or used for analysis, data must be extracted from their source system. Typically, when data are retrieved on a single patient, software interfaces exist that allow the clinician to browse their patients' information using a combination of proprietary and standard solutions. Many of these interfaces are based on standards developed by Health Level Seven (HL7). Data retrieval becomes more difficult when gathering cohorts of patient data for research or quality improvement. Data retrieved for this purpose undergo a three-step process known as Extract, Transform, and Load (ETL). Chapter 14 discusses interoperability in more detail. Here, we provide a brief overview of the ETL steps and major stumbling blocks [3].

- **Extract**. Data must be retrieved from the source system using available programming interfaces. The biggest stumbling block in this step is knowing what data resides where and what it means. For example, an ambulatory EHR system might be separate from billing systems, and thus the data from these systems must be merged to understand patient encounters. Diagnosis codes that represent billing diagnosis might not represent a patient's actual disease, so these would need to be stored separately from the problem list. For example, the billing diagnosis code for a visit to rule out diabetes is the same as a billing diagnosis code to manage diabetes.
- **Transform**. Because data are stored in various proprietary formats, it is necessary to align all these formats so that the data can be analyzed together. This task, known as data mapping, is often quite complex and is discussed at length in Chap. 13.
- **Load**. This step involves transferring data in large quantities into a data warehouse, which requires careful attention to some of the performance concerns discussed in "storing data" below.

### Data Representation

When data is in transit or being processed, structured data is often represented in one of the following formats: XML, JSON, or CSV [4]. These are largely interchangeable ways of organizing data. Text data (notes) often also have some structure in the header section, which defines to whom the note belongs, who transcribed it, and on what date, among other "metadata" fields (data about the data).

- **XML** uses tags, or text within brackets, to separate pieces of the document. Tags can be embedded in other tags, thus creating a hierarchy of information with a document.
- **JSON** is a similar format that uses colons, commas, and tabs instead of brackets and has become popular as it is generally more readable.
- **CSV** is a nonhierarchical structured format that essentially represents data as a spreadsheet, with columns and rows—commas separate columns, and each row appears on a separate line. A simple example of information in all three formats is below.

A sample data structure represents a patient's weight at an encounter as XML, JSON, and CSV in Table 10.1.

Chapter 13 will discuss data exchange standards, which define the specific tags, element names, and headings used to transit various data in these three formats. These data exchange standards build on the underlying structures of XML, JSON, and CSV. Most notable among these are the Fast Healthcare Interoperability Resources (FHIR), which describe EMR data in a standard way using these data structures [5]. FHIR's use is being accelerated by the 21st Century Cures Act, which mandates that EMR vendors support this standard in some circumstances [6]. These include serving Medicare patients and getting a certification from the US Office of the National Coordinator for Health Information Technology (ONC).

## Storing Data (Databases)

In enterprise systems, data are stored in databases.

**Table 10.1** Data representation of patient's weight in XML, JSON, and CSV

| XML | JSON | CSV |
|---|---|---|
| <encounter id='111'><br><vitals><br><weight units="lbs">140</weight><br></vitals><br></encounter> | {<br>"encounter": {<br>"id" : "111",<br>"vitals": {<br>"weight": {<br>"units" : "lbs",<br>"weight": "140"<br>} } } } | (This "vitals" csv would be one of several csv files needed to represent these data.)<br>Encounter,weight,units<br>111,140,lbs |

## Relational Databases

The gold standard for database storage is the **relational database** [7]. These are also known as SQL databases because database programming is done in the Structured Query Language (SQL) [8]. SQL 92, the version of the language released in 1992, is a standard across most database systems. Since then, many changes have been made to the standard language, but there is incompatibility across database platforms concerning features introduced since SQL 92. Although database platforms implement the features introduced in SQL 99 and some features found in even more recent versions, they all do so slightly differently. Therefore, database programmers tend to become experts in one platform, such as Microsoft SQL Server or Oracle.

The most common relational database brands used in HIT systems are those from Oracle and Microsoft. They have a reasonable equivalence of features, though, as mentioned, their SQL dialects are quite different. Postgres is a popular open-source database used in smaller health IT projects (such as for research systems), which offers many of the same features as the commercial equivalents but without the same level of support or guarantee of functionality.

## Database Schema Design

In SQL databases, data are stored in *tables* where each entry is a *row* with a predetermined set of *columns*. Conceptually, this is very similar to a spreadsheet. Like spreadsheets, various aggregate functions can be performed on tables to characterize the data. Unlike spreadsheets, tables can be *joined* to answer questions that cannot be gleaned from a single table. These joins are performed using the relationships between the tables, which is why these databases are *relational*.

The structure of the database tables for any particular application is known as the database *schema*. These tables are usually designed to store data so that information is not duplicated across tables. This is known as *normalizing* the data [9]. For example, a patient table might contain the patient's date of birth. A normalized schema would not duplicate the patient date of birth in, for example, the encounters table.

Structuring a database schema, so tables are normalized can be quite complex. Normalization should be done only up to the point that makes sense for the database application. There are more than six normal forms of data. However, the third normal form (3NF) is the level of normalization proposed by relational database pioneer EF Codd and is the general standard for minimizing data repetition [10]. 3NF specifies that every piece of data in a row only depends on the information in the primary key (the primary identifier for the table, such as a patient id or encounter id). For example, an encounter table might have a provider identifier. The encounter table should not also have the provider's name and address, as these are properties of the provider and not the encounter. These should go in a separate provider table.

To understand how joins are used, consider how many encounters occurred in 2014 involving patients born in the 1960s. A database programmer would issue a query that "joins" the patient and encounter tables. The power of relational databases is that these joins are dynamic and ad hoc and do not require a priori definition of relationship hierarchy. To join two tables, a common column must exist between these two tables. This is an exception to the "do not duplicate data" rule of normalization. These two columns are the *primary key* (the column[s] of the primary table to be joined) and the *foreign* key (the column[s] of the secondary table to be joined). In the above example of the patient and encounter table, both tables would include some type of patient identifier. More technical details of SQL joins can be found in the section "Programming" below.

Schema designs are frequently visualized with an Entity Relationship Diagram (ERD). These simple diagrams use boxes to represent each table in the schema. Each box lists the columns in the table and their data types. Usually, the keys of the table are demarcated by boldfacing or otherwise highlighting them. Lines are drawn between boxes where a relationship exists (i.e., indicating that the two tables can be joined). The lines are annotated with the type of relationship: one-to-one, many-to-one, or many-to-many. The patient-to-encounter relationship would be one-to-many because a single patient can have many encounters, but each encounter is about only one patient. A many-to-many relationship might be a provider and patient table. A provider has many patients, and a patient likewise has many providers. Many-to-many relationships are often shown on ERD diagrams as a pair of one-to-many relationships, with an intermediate table in the middle that provides the many-to-many linkage. In this case, the encounter table might be the intermediary table for the many-to-many linkage (assuming that a patient can have only one provider per encounter). A variety of schemes for annotating the relationship exists. An ERD diagram based on this discussion that uses the popular "crow's foot" annotation method is shown in Fig. 10.1.

One complexity to consider when defining a database schema is balancing usability with resilience to future changes in that schema. It is generally faster and easier to access data with predefined columns (such as columns in a patient table, e.g., gender, race, and ethnicity). Still, suppose the available data could change dramatically over time. In that case, it is often better to use an entity attribute value (EAV) format, a special way of normalizing the data that provides great flexibility for schema changes [11]. In pure EAV format, a table has only three columns. In a patient table, the Entity column would be a patient identifier. The Attribute column would define what is being measured in that row (e.g., birthdate). The Value column would have the value of the measurement (e.g., January 1, 1960). Thus each patient's data in the patient table would take up many rows. Without careful indexing, this can have poor performance.

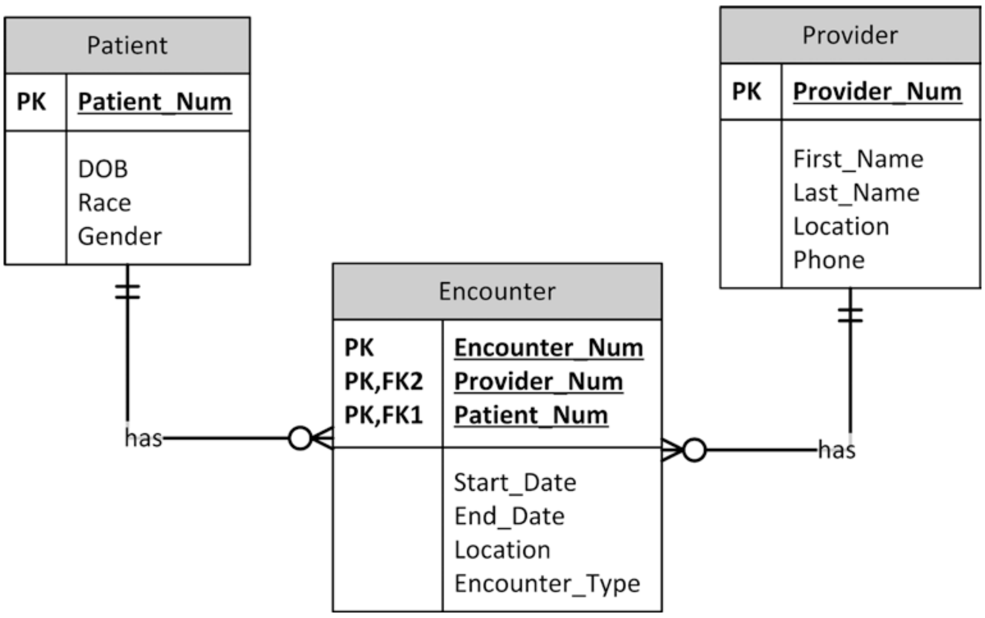

**Fig. 10.1** Entity relationship diagram for a simple database schema

Furthermore, it is not particularly human-readable. However, it is immediately adaptable to new data types without changing the underlying database schema. Because of this, EAV is used in many data warehouses. In practice, schema styles are used that combine EAV and standard tables. The star schema and the snowflake schema prototypes are most common, both of which involve one or more EAV tables and *dimension* tables that define additional attributes using standard normalized data. The dimensions are linked to the EAV table through additional columns (foreign keys). The Informatics for Integrating Biology and the Bedside (i2b2) database framework for clinical data warehousing, free and in use at over 200 sites worldwide, uses a star schema format [12].

### Cloud Database Providers

Of increasing importance are cloud-based data storage providers. Although due to security concerns around clinical data, such data are traditionally stored on-premises ("on-prem") on an institutionally managed database server, the flexibility of storing data in the cloud is attractive. It does not require institutional investment in and maintenance of database servers, and database size can be "elastic" and dynamically allocated by the hosting provider. All major cloud providers support healthcare data in some way, and it is becoming more commonplace for major institutions to sign BAAs (business associate agreements) with commercial cloud providers.

New variants of SQL and other query languages come with these cloud providers that informaticians must become familiar with. Google provides an implementation called BigQuery, which is compatible with SQL 2011 standards [13]. Amazon promotes a scalable database solution called RedShift, which implements its own dialect of SQL to support very large datasets and high-performance analytics [14]. Microsoft Azure suggests using Azure SQL, which uses a SQL language like Microsoft SQL Server [15]. Although all these companies offer more traditional databases on the cloud, they claim that the highest performance is achieved with one of their cloud-native approaches.

There are many cloud-based NoSQL solutions (i.e., databases that provide programming interfaces not based on SQL). See the next section for a discussion of NoSQL.

### Database Integrity and Performance

Because databases are often accessed by many systems simultaneously, it is critical that no two systems modify the database simultaneously. Furthermore, databases must be resilient to failures (such as power or hardware). Data integrity in relational databases is achieved through the ACID principles [3]. In this framework, database operations that must occur together are said to be a single *transaction*. The elements of ACID are:

- **Atomicity**. If one part of a transaction fails, the entire transaction is reversed.
- **Consistency**. No transaction will violate the rules of the database (such as the schema and other constraints).
- **Isolation**. If transactions are run concurrently, the database and results must be the same as if they were run consecutively. This can be achieved by configuring the system to run all transactions consecutively. Still, in practice, complex database scheduling programs determine which transactions can be run simultaneously (for example, read-only transactions can always be run simultaneously).

- **Durability**. Once a transaction succeeds (is *committed*), the changes are resilient to failures and visible to all other running transactions. Database designers must balance this requirement with performance because true durability means that committed database changes must be immediately written to permanent storage (i.e., they cannot be stored in memory), which is much slower than using memory.

Columns on tables can be *indexed*, which speeds up searches significantly. Defining indices depends on the intended application and the database's *query optimize*r, which maximizes the performance of index use. Every database engine (e.g., Oracle or SQL Server) has a unique query optimizer, so index designs must be tweaked for each database engine supported. As a rule of thumb, the performance of a database table will not degrade until the relevant parts of the index are too large to fit into memory. Therefore, it is possible to have tables with hundreds of millions of queryable rows in milliseconds if the query optimizer uses indices. At that scale, index design and query optimization become very important, and there are many tutorials and technical documents on this subject. Unfortunately, the optimal query design also varies between database platforms. For example, Oracle often excels on complex, large queries, whereas SQL Server tends to do better when each step is computed separately and stored in a temporary table.

## Non-relational Databases (NoSQL)

Non-relational databases (collectively called NoSQL) are becoming popular for some specific tasks, although relational databases remain the highest performing systems for general use. However, NoSQL databases can be very powerful for in-memory and distributed querying (i.e., when there are extremely large amounts of memory and many compute nodes).

Popular NoSQL approaches include:

- ***Massachusetts General Hospital Utility Multi-Programming System (MUMPS):*** MUMPS is particularly important to the medical informatics community [16]. This is a database format developed in the 1970s at the Massachusetts General Hospital before relational databases. It is still widely used in medical informatics. It is both a programming language and a database, and all data are stored in *sparse matrices* rather than in tables. (See the section "Knowledge Discovery and Data Mining (KDDM)" below for more information on sparse matrices.) It is very efficient at complex data manipulation. Because MUMPS was developed when memory was costly, it tends to be very terse—all MUMPS commands can be reduced to a one-to-three letter abbreviation.

  Additionally, spaces are important (which is not true in most languages). A space is used to separate commands, for example. Therefore, MUMPS programs tend to be more cryptic than SQL. Entire systems have been written in MUMPS, but many modern systems (such as Epic's EHR platform) use MUMPS similarly to SQL and use a more traditional language for user interaction (see the section "Programming" below). MUMPS implementations include M and Caché®. The latter is the most popular MUMPS implementation, sold by InterSystems, Inc. Caché® is now part of the company's suite of tools called IRIS, which exposes a multi-model datastore built on MUMPS and provides an approach to use SQL and noSQL in the same environment [17].
- **MapReduce databases**: MapReduce is an algorithm developed by Google that allows optimized querying in "massively parallel" environments [18], where hundreds of computers execute portions of queries simultaneously. Each query is split into many small subtasks. When the hardware is available, parallelizing complex computing tasks into inexpensive computing nodes is very appealing. Hadoop is a popular open-source MapReduce database [19].
- **Document databases**: Whole-document storage and processing is a feature of many NoSQL databases that support MapReduce. This simplifies the Load process of ETL because the data can be stored and queried as structured documents. Thus, the transformation from the transport format (e.g., XML) into a database schema becomes unnecessary. Document databases are computer-processing intensive, but in a massively parallel environment, this can be mitigated.
- **Graph Databases**: In cases where the relationships between tables can be predefined into a schema of linear relationships (such as "patients have encounters" and "encounters have data on medications"), a graph database allows such data to be traversed faster than the dynamic data relationships of a relational database. The difficulty is that the data relationships are static and must be traversed linearly. In this example, it is not possible to directly join patients and medications. This can create performance problems and limit query design when the data are not used as anticipated. On the other hand, the performance is very good if the schema fits these constraints. Neo4J is a popular open-source graph database [20].

NoSQL databases frequently relax some of the constraints of ACID to achieve high performance. Therefore, in many cases, NoSQL is better at analytics on massive, slow-to-update datasets than live systems that are continuously updated (e.g., an EHR).

*Examples of NoSQL Databases* Neo4J [20] (a popular open-source graph database); MongoDB, CouchDB, and Hadoop [19] (MapReduce Document databases); Caché® and Iris (a widely-used MUMPS database and its successor).

Many NoSQL databases are open source but frequently provide recovery and support contracts for commercial use. Additionally, cloud providers offer many NoSQL solutions, such as Amazon's DynamoDB and Google Firestore (both document databases).

## Using Data

Data serve no purpose without a reason to use them. Here we briefly discuss some important uses of data in HIT systems.

### Health Information Systems

Health Information Systems (HIS) are the clinical systems used to retrieve patient data for review by their caregivers [21]. Structured data are presented in easy-to-understand formats such as flow sheets. The data sometimes power useful applications that run alongside the health record, such as decision support systems, which provide helpful suggestions to improve patient care (e.g., reminders about vaccinations). Most systems can search within a patient chart to find keywords in unstructured data or draft a patient note for a visit based on the structured data entered for that visit. Many innovations continue to emerge. Homegrown HIS used to be common. Commercial systems have largely replaced these. Still, recent government initiatives, such as the 21st Century Cures Act, are encouraging open standards for integrating smaller, single-purpose "apps" with larger HIS [22–24].

### Data Warehouses

Data warehouses are increasingly used within hospital systems for, among other uses, quality improvement, public health reporting, research, and clinical trial recruitment. The ETL process described earlier copies data into data warehouses out of production systems. These data warehouses may be refreshed as frequently as daily, or they might be created on a one-off basis (for a research project, for example), depending on the applications for the warehouse and the amount of data. The COVID-19 pandemic motivated many healthcare organizations to develop faster data warehouse refresh pipelines. Daily or weekly updates on COVID patients could occur to speed up research on the disease. This will have the effect of faster ETL pipelines post-pandemic.

Data warehouses define a Common Data Model (CDM) and may offer various data analytic tools that will run on the CDM. Several open-source clinical data warehouses are in widespread use. The most widely used freely available platforms are i2b2, OMOP, and PCORnet. Additionally, EHR vendors frequently offer a data warehouse (Epic Caboodle), and home-grown data warehouses built by individual hospital systems are still widely used. Here we will briefly introduce the freely available platforms.

**Informatics for integrating biology in the bedside** (i2b2) is the oldest, freely available data warehouse system, first developed over a decade ago and used at over 200 sites worldwide. It is also used in large data research networks, including NCATS's national Accrual to Clinical Trials (ACT) network. In addition to a data model, it provides an Application Programming Interface (API) for query and data retrieval, supporting database-independent app design. It also offers a client tool for developing queries and viewing results [12, 25, 26]. i2b2's greatest strength is its flexibility and ability to ingest and analyze new types of data without changing the core data model. Besides EHR data, i2b2 is used in many other unique domains, including patient-reported outcomes, genomics, and social determinants of health.

The **Observational Health Data Sciences and Informatics** (or OHDSI, pronounced "Odyssey") Collaborative provides a CDM known as **Observational Medical Outcomes Partnership** (OMOP). The collaborative offers a variety of analytic tools, from cohort design to regression analysis to data sharing. However, the platform's greatest strengths are probably its well-specified data model and comprehensive, regularly updated data dictionary of curated terms from many standard terminologies. These make OHDSI/OMOP very appealing for data analysts because SQL queries are readable and relatively easy to write. As of this writing, OMOP is implemented at over 100 organizations worldwide [27, 28].

The Patient-Centered Outcomes Research Network develops the PCORnet CDM, an OMOP-like relational data model representing EHR data. It is used at PCORnet sites in the US, which currently encompasses 70 million patients' data. Network participants gain access to data characterization and quality checking programs that run on the commercial SAS analytics platform and produce reports used for quality improvement [29].

### Health Information Exchange

Many initiatives to share information across health systems are collectively dubbed "health information exchange" or HIE. This can be as small-scale as electronically transferring a single patient's records to a new hospital system, such as the Direct project from the National Coordinator's Office for Health Information Technology [30]. HIE can also be as large-scale as distributed analytics across an entire state or country.

Early efforts in HIE took data from local sites and built regional data repositories (Regional Health Information Organizations, or RHIOs) for analytics. Several of these projects were successful, such as the Indiana Network for Patient Care operated by Indiana Health Information Exchange, which aggregates data on millions of patients from dozens of hospitals, as well as independent laboratories and insurance companies for a comprehensive record of the patient's medical history [31].

In general, however, regulatory issues around sharing patient data hamper the success of this approach. Therefore, in the past decade, the "federated network" has emerged as the most prominent modality of health information exchange. In this model, data stay at home institutions. Rather than creating central repositories of data, the "questions are brought to the data"—queries are distributed across networks of health systems, and only the results are aggregated. This solves a variety of privacy and security problems at the expense of performance. Many large government-sponsored national networks take this approach, such as PCORnet [32], the NIH ACT [25] network, and the Mini-Sentinel network [33]. The new NIH "Long COVID" research network (RECOVER DRC—Researching COVID to Enhance Recovery Data Resource Core) is also planning to use a federated approach [34]. Emerging advancements allow much more complex distributed analysis, taking advantage of new technologies like homomorphic encryption to exchange patient-level information while ensuring patient privacy [35].

## Knowledge Discovery and Data Mining (KDDM)

KDDM refers to using statistical methods on data to discover patterns that are not intuitively obvious upon inspection [36]. In practice, preliminary knowledge discovery frequently occurs through simple searches in databases of patient data (such as keyword searches in notes or "cohort finding queries" on data warehouses). Still, KDDM can also be much more complex [37]. One popular use of KDDM is for *predictive analytics*, such as predicting 30-day hospital readmissions or risk of heart failure. This type of KDDM uses *classification* algorithms, such as *regression analysis* or *support vector machines* [38]. Classification algorithms are known as *supervised learning* because the correct outcome is known and *supervises* the algorithm as it trains its parameters. Supervised learning involves a *training set* of data, meaning that the final statistical model is developed from a set of data where the true positives are known. Then the model is tested on a *test set*, where the true positives are unknown to the algorithm, and the algorithm's accuracy is evaluated by how closely the algorithm correctly labels the test set.

A popular approach for robust testing involves repeatedly splitting the data into different training and test sets and comparing performance across all parameterizations of the algorithm. This is known as *cross-validation*. A related technique, *bootstrapping*, creates additional training data by resampling the existing training set (i.e., creating additional simulated data based on the statistical properties of the training set). For algorithms where sensitivity can be varied, the algorithm's output is often presented as a *Receiver Operator Curve (ROC)*, which is a plot of sensitivity against one-specificity for each parameterization of the algorithm.

*Unsupervised learning* is also becoming popular in medical KDDM. Unsupervised learning looks for patterns or relationships in data where there is no known "goal". The most popular example of unsupervised learning is recommendation algorithms used in consumer e-commerce platforms such as Netflix and Amazon to suggest purchases to customers based on the previous purchase history [39]. This type of algorithm has been used, e.g., to generate drafts of decision support, suggest ontological relationships among data elements in standardized vocabularies, and find the most important variables in a dataset (feature selection) [40–42]. One of the most recent popular unsupervised techniques is the autoencoder, which essentially uses a single dataset for training and testing. The goal is an algorithm that can efficiently reproduce the input data from a smaller set of parameters. These smaller representations of the original data can then be used in a variety of ways, such as data compression, noise reduction, synthetic data creation, etc. [43] Autoencoders are a form of Deep Learning, which is becoming an important term of art in medical informatics [44]. Deep Learning networks are essentially complex, multilayer neural networks (a classic machine learning technique that dates back to the 1960s). However, today's extremely powerful computing resources allow very complex networks, inspiring a revolution of new KDDM tools and applications.

The data format required for KDDM is somewhat different than data transport or storage. Whereas databases store information in normalized tables and transport formats tend to store data hierarchically, KDDM usually requires data in a *sparse matrix*, in which there are perhaps hundreds of columns, each representing a parameter that could be predictive of the desired outcome. This is the same format used by MUMPS. These matrices are known as sparse because most of the entries in the matrix are empty.

## Data Quality

It is important to remember that representing data efficiently and with semantic standards does not guarantee sufficient quality to be used in KDDM algorithms and large-scale HIE. Because EHR data are entered by busy humans whose primary goal is to *provide* healthcare, the *documentation* of such healthcare can at times be lacking. Moreover, EHR documentation is largely driven by billing needs, so information needed for analytics (a secondary use of the data) is often inadequately recorded. A range of problems are possible, from the use of unexpected (albeit standard) codes to information not being present at all. The core elements of data quality are: *conformance* (does data adhere to the required standards?), *completeness* (are data present?), and *plausibility* (are data believable?) [45]. It is possible to ensure conformance through well-written ETL, but completeness and plausibility are much more difficult. For this reason, CDMs like OHDSI and PCORnet have made quality checks a cornerstone of their tools. Still, data quality is a major limiting factor in the use of EHR data for research. It is, therefore, very important to validate the accuracy of algorithms and data-based discoveries across multiple locations to detect potential differences in information entry and coding [46]. Data Quality is covered in more detail in Chap. 16.

## Networks and Network Architecture

In this section, we will discuss how computers communicate with each other and with various devices that may be instrumental in collecting medical data, such as imaging and laboratory machines.

### Networks

Computer systems connect to each other via *networks*. Networks operate over various physical media, including copper wire, fiber optic cable, and wireless radio transmission. Networks convey various information, including text, sound, and video, over the Internet, medical orders within a health care system, and the exchange of medical data between care providers.

*Enterprise networks*, sometimes called corporate networks, link computer systems within an organization to support the organization's business processes. Networks or subnetworks within a building or campus are known as Local Area Networks (LAN). The characteristics of a LAN include high network speeds, routing at lower layers of the network, local ownership, and a high degree of trust between nodes.

LANs contrast with Wide Area Networks (WANs), which employ different technologies than LANs to connect campuses or buildings across longer distances. *Telecommunications* refers to the technologies employed to send data, voice, or video over distances of more than a few hundred meters. Telecommunication technology is highly specialized, and most organizations rent either shared or private long-distance connections from telecommunications companies.

As one might imagine, a private telecommunication connection is more secure than a shared connection. However, a Virtual Private Network (VPN) achieves something similar to a private connection by encrypting all communications between two locations over a shared network.

#### Network Topology

*Network topology* is an abstract representation of the way computer systems connect. Computer systems are visualized as nodes on a graph in network topology and network connections as lines between nodes. Simple network topologies in include:

- **Point-to-point**, in which two computers connect directly to each other.
- **Star topology**, a central system such as a large computer or router connects to each of the other computer systems. The satellite systems communicate with each other through the central node.
- **Backbone topology**, in which a shared communications channel such as an Ethernet cable serves as a backbone linking nodes at multiple drop points. The Internet Cloud is a variant of a Backbone topology—the essential feature being multiple drop points from a communication medium into which we have no visibility.
- **Ring topology**, a backbone circles around to connect its ends to form a large ring. The ring topology provides increased reliability since cutting the ring at any point produces a backbone that can continue communications.
- **Hybrid topology**, in which multiple backbones, stars, and rings connect. An enterprise network is likely a hybrid.

In a hybrid topology, the constituent network segments connect via specialized network devices. Network devices may boost the physical signal to allow networks to extend over longer distances.

#### Seven-Layer Network Model

Another way to think about networks is by looking at how atomic data (binary 0's and 1's) are organized and transferred. We categorize network devices as hubs, switches, routers, and firewalls by the *network layer* at which the device connects subnets. Table 10.2 shows the network layers of the seven-layer Open Systems Interconnection (OSI) network model [47] of the International Standards Organization (ISO). Note that HL7 was aptly named as it focuses on the 7th layer of the OSI model.

*Firewalls* are a special case in that they are security devices that operate at multiple network layers. The firewall passes approved network packets, and it blocks unapproved or suspicious network packets, per a list of approved network addresses, application port numbers, and network protocols. Firewalls may also scan network traffic for known viruses or leaks of confidential information.

#### Network Speed

As any user of the Internet knows, network speed matters. Several factors affect network speed. *Network speed* is the time it takes for a fixed amount of data, such as a message or a file, to cross the network from one computer system to another. The raw network speed, known as *bandwidth*, is the rate at which binary 0's and 1's (bits) cross the network (bits per second). Modern networks transfer megabits (millions of bits per second) or gigabits (billions of bits per second).

**Table 10.2** Network layers of the International Standards Organization (ISO) model

| Layer | Name | Description and examples |
|---|---|---|
| 7 | Application | The application layer defines the message format between computer systems or the human-machine interface. Examples are HTTP for web browsers or HL7 for communicating health information between servers |
| 6 | Presentation | The presentation layer handles encryption and compression of data packets. Examples are SSL encryption, ASCII text or JPEG images |
| 5 | Session | The session layer performs authentication, authorization and session restoration. An application connects to a session via a socket, which is assigned by port number |
| 4 | Transport | The transport layer provides end-to-end error control, since data may pass over many physical layers and routers between ends. TCP is a common transport layer protocol. When combined with an IP Address, TCP/IP is the transport method used by the Internet |
| 3 | Network | The network address is an external (unique globally) or internal (unique within the enterprise) address assigned by the network, such as an Internet Protocol Address (IP Address). The network layer connects via routers |
| 2 | Data Link | The data link layer performs error detection and flow of control on the physical link, i.e. controls which end is transmitting and which is receiving. This layer uses physical device addresses known as Media Access Control (MAC) addresses. Each networked device has a unique MAC that does not change if you move the device to a different part of the network. Ethernet is a common data link protocol. The data link layer connects via switches |
| 1 | Physical | Physical medium, such as copper wire, optical fiber or wireless radio transmission. Physical segments connect via hubs |

However, there is much more to network speed than bandwidth. Any modestly large data set, say a web page, is broken down into smaller data packets to cross the network. A packet header of routing information is added to each data packet for the network to correctly route and reassemble the packets at the destination. Therefore, the actual number of bits transferred increases by some amount, typically in the 5–10% range.

In addition to the packet-header overhead, there will be some delay in getting the first byte of the packet transferred, called *network latency*. Network latency usually results from (1) the time it takes a network device (hub, switch, router, or firewall) to receive the packet, process its header for the relevant routing information, and then retransmit the packet toward the appropriate target; and (2) waiting time due to competition for network resources from other computer systems using the network.

Networks are fundamental to any modern enterprise computer application, with LANs connecting local computer systems and WANs connecting the enterprise to other organizations. Network topology affects the reliability, scalability, maintainability, and cost of a network. Network speed is influenced by different types of network devices (hubs, switches, routers, and firewalls), which operate at different network layers to route data packets and reassemble them at the correct destination.

## Network Architecture

Architecture is about the big picture—how the parts relate to the whole. In *systems architecture,* we break the computer system down into *components* and *relationships* among these components. There are multiple ways to divide a system into components, depending on what aspect is most important to the analysis or the target audience. The most common of these are network topology, application structure, the flow of data among components, and a summary of the most important features of each breakdown.

### Architectural Diagrams

Let's consider a hypothetical obstetrics system as an example. This system collects and manages pregnancy information during clinic visits, makes that information available to the hospital at the time of delivery, and eventually sends the data to a data warehouse for research.

*Architectural diagrams* are the most common way to represent a system of components and relationships. The ability to read and understand common architectural diagrams is a key to communicating with IT professionals.

Figure 10.2 shows a *Network Architecture Diagram* of the network used by our hypothetical system. This diagram conveys information about the hybrid network topology at the lower layers of the OSI network model:

- A star topology centered on the Internet cloud, connected via Firewalls to the Clinic, Hospital, and University networks
- A single Ethernet backbone at the University, connecting servers, data storage, and user devices
- Two Ethernet backbones connected with a (Layer 2) switch at the Clinic
- A wireless network at the Clinic, connecting to a wireless table for user interaction,
- A ring network connected to a (Layer 3) router at the Hospital

Note that a Network Diagram shows how the servers, data storage, and user interface devices are connected but doesn't show what is happening at the application level (Layer 7).

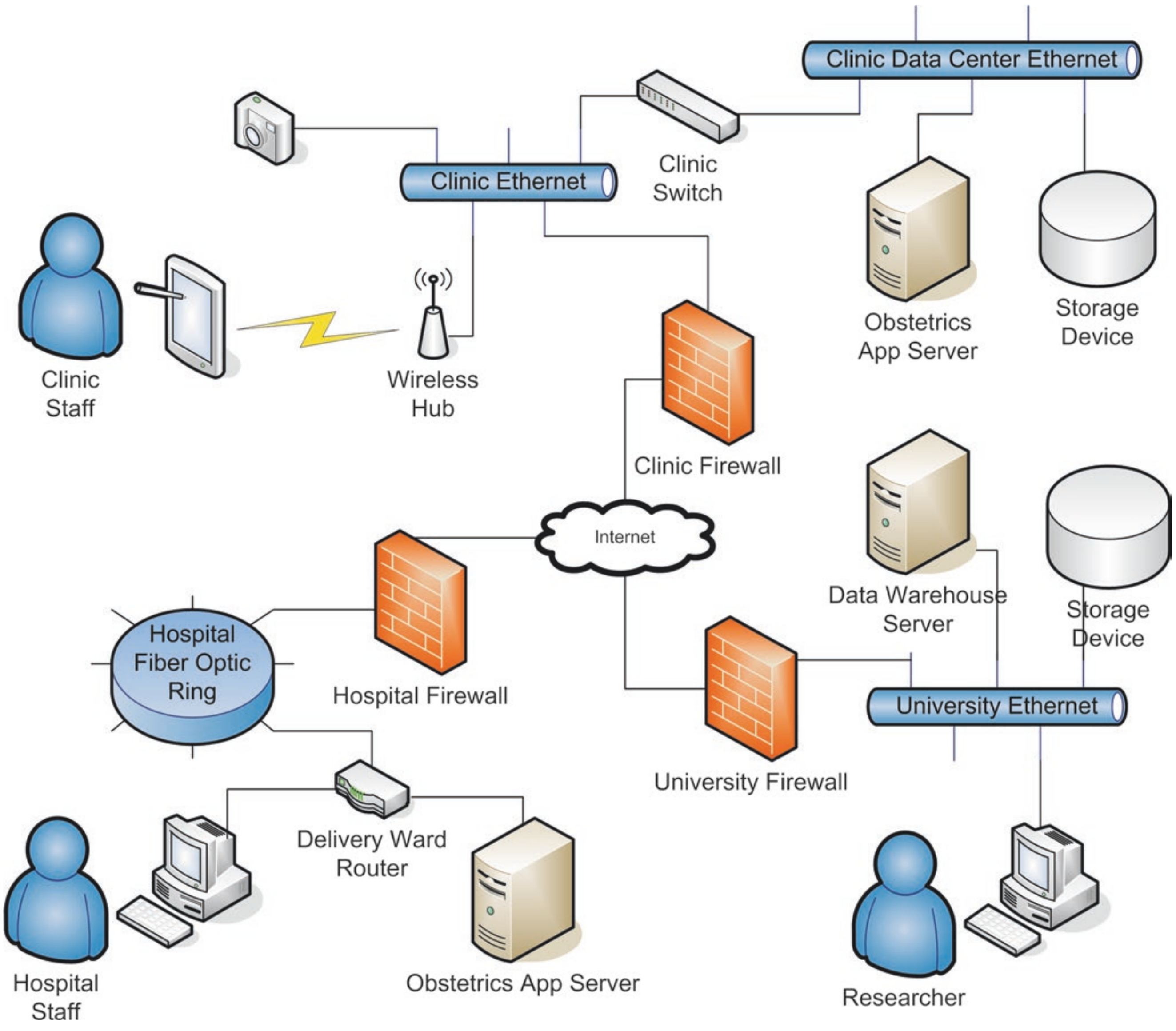

**Fig. 10.2** Network architecture diagram of sample obstetrics system

In Fig. 10.3, a *UML Activity Diagram* shows how the application logic works at Layer 7. The major features of the UML Activity Diagram are:

- Swimlanes are vertical boxes that group the activities according to who and where the actor is (Clinic Provider, Obstetrics Application, Hospital Provider, Data Warehouse, or University Researcher)
- Processes, boxes with rounded corners
- Datastores, boxes with less rounded corners
- Flow of control, represented as solid arrows
- Flow of data, represented as dashed arrows
- Split and join operations on the flow of control, shown as dark bars. In our system, this occurs where the clinic provider performs the sonogram and note & observation entry

UML stands for Unified Modeling Language, which Grady Booch, Ivar Jacobson, and James Rumbaugh developed in the mid-1990s [48]. In 2000, the ISO adopted UML as a software design standard. An activity diagram is only one type of diagram in the UML family, including many other diagrams for software structure, behavior, and deployment.

A *Data Flow Diagram* describes the movement of data through a system, with emphasis on data transformations. Circular nodes represent data transformation processes, and labeled lines show data flow from one process to another. The Data Flow Diagram in Fig. 10.4 shows:

- A starting point at a double circle
- Every line is labeled with the data elements in motion

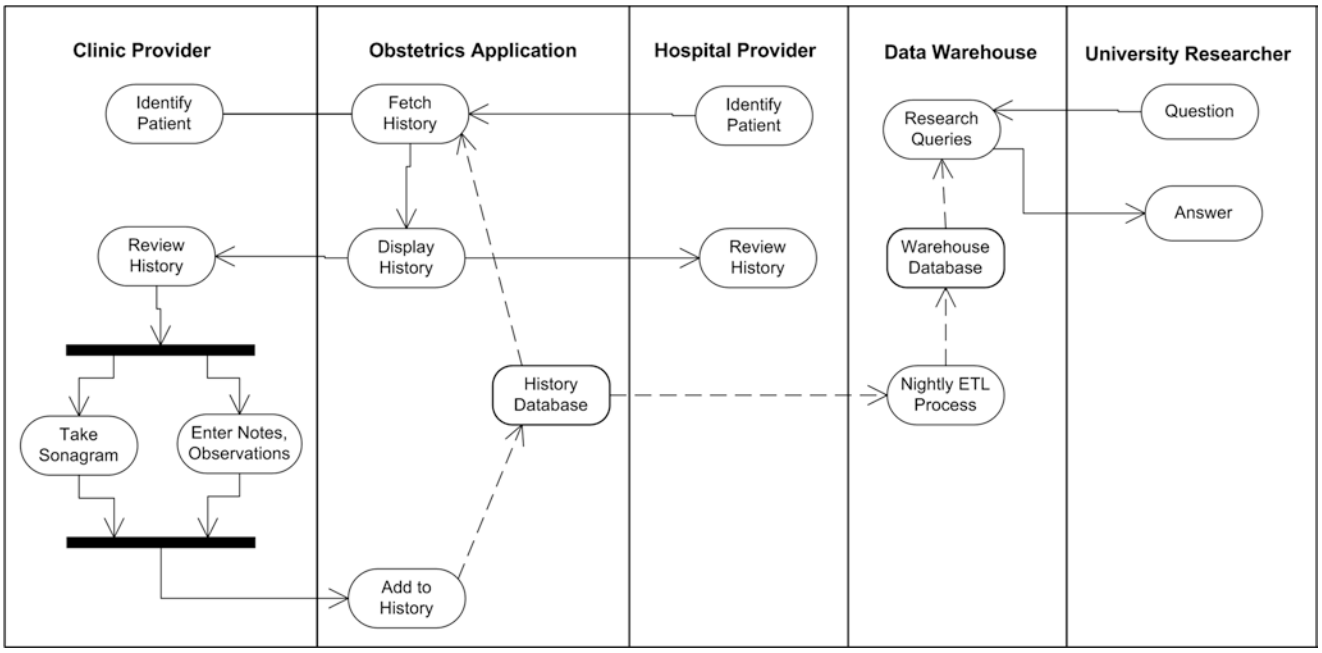

**Fig. 10.3** UML activity diagram of sample obstetrics system

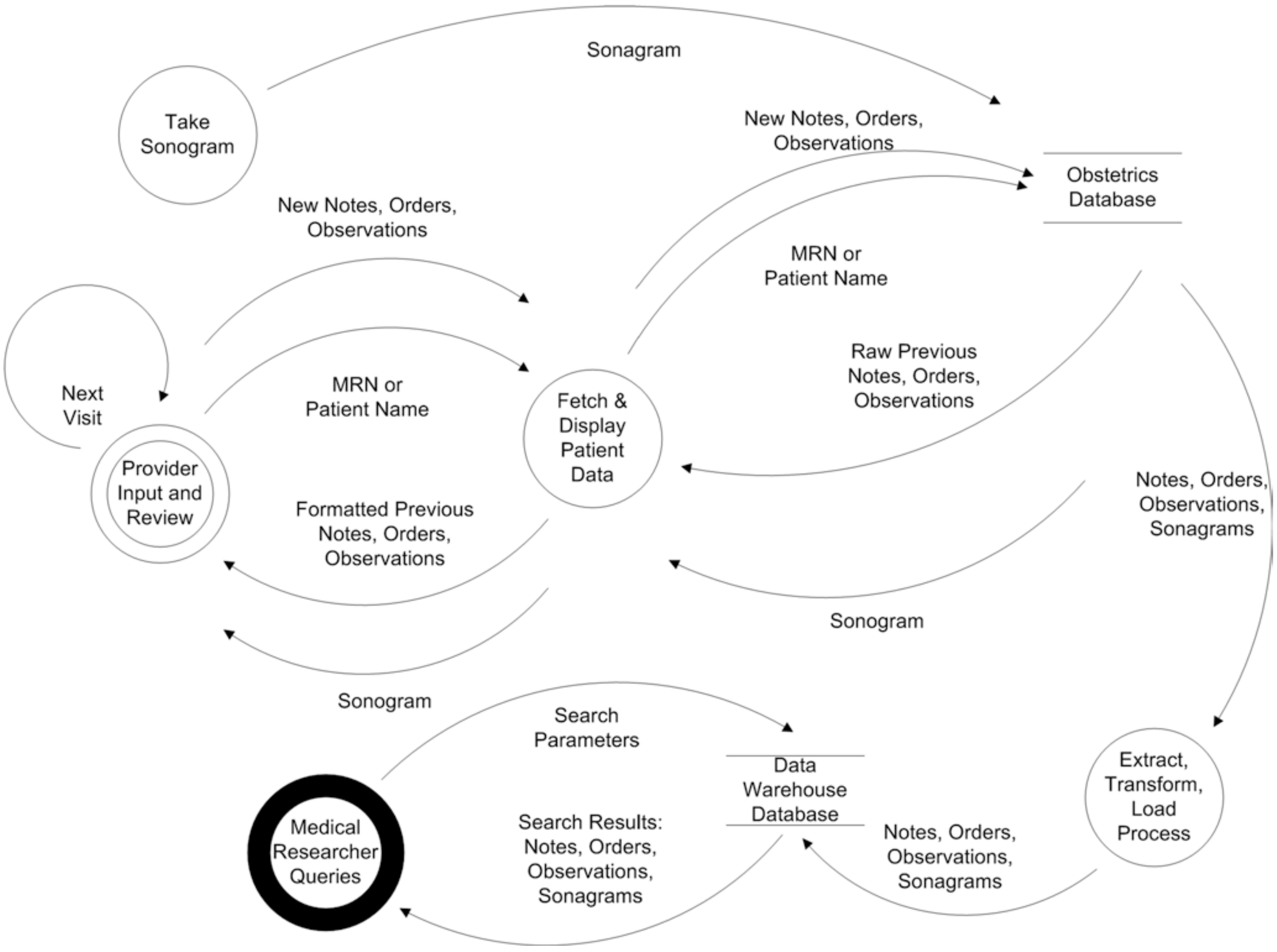
Sonagram
Take Sonogram
New Notes, Orders, Observations
Obstetrics Database
New Notes, Orders, Observations
MRN or Patient Name
Next Visit
MRN or Patient Name
Raw Previous Notes, Orders, Observations
Fetch & Display Patient Data
Provider Input and Review
Formatted Previous Notes, Orders, Observations
Notes, Orders, Observations, Sonagrams
Sonogram
Sonogram
Search Parameters
Data Warehouse Database
Medical Researcher Queries
Search Results: Notes, Orders, Observations, Sonagrams
Notes, Orders, Observations, Sonagrams
Extract, Transform, Load Process

**Fig. 10.4** Dataflow diagram of sample obstetrics system

- Every circle is labeled with a data transformation process
- Permanent data stores (obstetrics and data warehouse databases) are represented as open rectangles
- An ending point at the darkened circle

Sometimes the goal is to communicate the overall structure and behavior of a system with only the main features of each aspect of the system. An *Enterprise Architecture Diagram*, as in Fig. 10.5, shows how to accomplish this.

- The main feature of the network shown is the Internet cloud
- Additional network connections are shown as arrows labeled with the data elements being transported, emphasizing the data flow at the application layer (Layer 7) and not the underlying network topology, protocols, and physical structure
- The system users, Clinic Providers, Hospital Providers, and Researchers appear in all types of architecture diagrams. This is appropriate because these actors are essential in defining how the system interacts with the real world
- Computer servers and PCs show how the application is divided and distributed
  - The application displays information on PCs and tablets, organizes information on application servers, and stores data on database servers.
  - The obstetrics application runs on two servers, one at the clinic and one at the hospital, and on multiple user workstations.

## Application Architecture

*Application architecture* refers to the way the software is broken down into components, especially on different servers. *Software tiers* are the layers from user interaction to the database and back. A three-tier architecture is common: (1) user interface (front end) on a PC or tablet, (2) application server, which may serve multiple users, and (3) database server, which may serve multiple applications.

If the user interface layer is simple, such as a web browser, we call it a *thin-client* application. We call it a thick-client application if some or all the application logic is encoded in the front-end tier. If the application resides on multiple servers, then it is called a *distributed application*, and similarly,

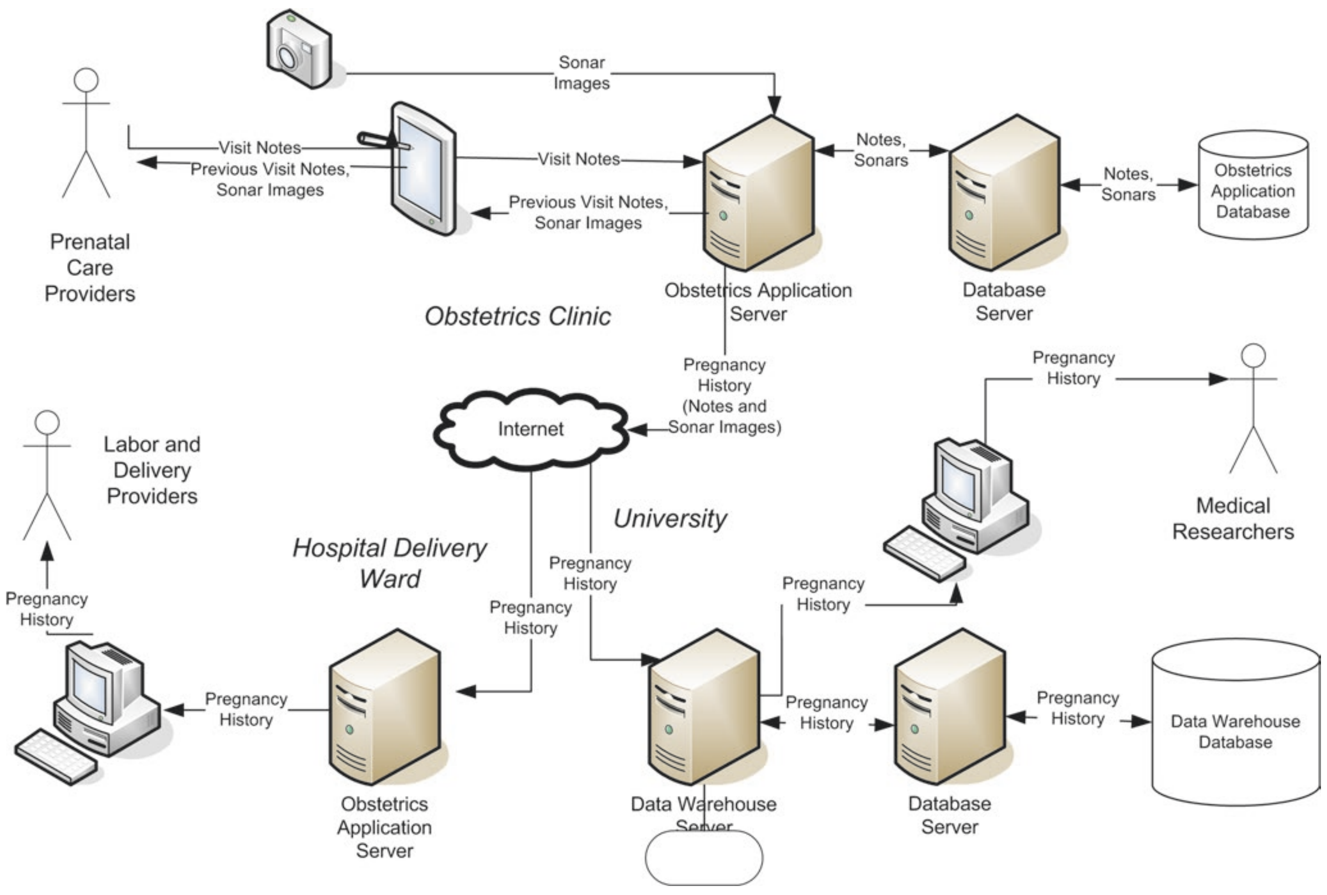

**Fig. 10.5** Enterprise architecture diagram of sample obstetrics system

if the database resides in multiple locations, it is called a *distributed database*. Distributed systems are more reliable and scalable, but they come at a greater cost and add complexity to maintain and support.

### Non-functional Requirements

The decisions embodied in selecting system architecture have a significant impact on meeting *non-functional requirements*. Non-functional requirements are not features but things like usability, reliability, response time, maintainability, security, disaster recovery, and system cost.

For example, in our diagrams, we represented *servers* as individual computers. This was always true when computers first came into wide use in the 1900s, but it is often no longer the case. *Virtual servers,* or, more precisely, *guest virtual servers,* are emulations of *physical servers* on a larger *host virtual server*. Virtual servers do everything a physical server does, but because they share resources with other virtual servers on the same host, they are more economical and maintainable. *Cloud computing* places the host virtual server on the internet, where a third party manages the host and sells guest computing capacity, capitalizing even further on economies of scale.

Other extensions of the simple physical server include *parallel computing*, in which multiple processing units share the computational load. This is very common in recent years, even on inexpensive PCs. *Grid computing* extends the parallel computing notion to groups of physical servers, such as all the PCs in a building or all servers in a data center. Some applications can leverage parallel or grid computing to speed themselves up many times (such as MapReduce discussed earlier in this chapter) [18], but other applications may be a series of sequential steps that cannot benefit from parallel computing.

### Integration and Interfaces

Another key aspect of application architecture is whether the relationship between two components is tight and private (*integrated*) or loose and public (*interfaced*). Interfaced components allow for interoperability. This is especially true for interfaces defined by public standards. For example, the World Wide Web (WWW) depends on two public standards: TCP/IP for transport and HTTP for formatting data for use by web browsers.

When computers provide services to other servers on a network via a standard application interface, it is sometimes called a Service-Oriented Architecture (SOA) [49]. Some common frameworks for general-purpose SOAs include SOAP (Simple Object Access Protocol) [49], REST (REpresentational State Transfer) [50, 51], CORBA (Common Object Request Broker Architecture) [52] and ICE (Internet Communications Engine) [53].

REST is heavily used in medical informatics, as it builds on Internet motifs and is simple to implement and operate. Many of the systems discussed in earlier sections utilize REST, including FHIR, i2b2, and OHDSI tools. Other communications standards that typically rely on REST include HL7, CCD, and standard terminologies like ICD-10, LOINC, and RXNORM (detailed in Chap. 13).

## Software, Computer Languages, and Programming

Software is the command center that controls the components in the system architecture. Like spoken language, the software can be written in a variety of programming languages. These vastly differ from one another. Most programming languages are extensively documented in other reference books and online [54–57]. Here, we will cover the most important approaches from the perspective of medical informatics, focusing on data.

### Data Types

In programming languages, data are stored in *variables*. Variables are temporary holding cells for data that *vary* as a program executes. Data can be stored longer-term in files on disk or in relational database tables. In MUMPS, this distinction between database and variable is blurred—variables can be either in-memory holding cells or locations in a database.

No matter where data are stored, each variable or database column has a specific *data type* that constrains the data type that can be stored. Languages can be *strongly typed* or *weakly typed*, depending on the degree of computer verification that variables correctly match their defined data type. Weakly typed languages, which do not enforce such checks, are harder to debug and run less efficiently. Still, they offer more flexibility and the potential for data types to change as the program is running. Common data types include:

- **Numbers**: usually defined as integers or floating-point numbers (numbers with decimals)
- **Letters**: single characters and strings (sequences of characters, or what we commonly think of as text)
- **Dates and times**: specialized storage of these temporal data, which supports computer interpretation and manipulation
- **Lists and sets and other collections**: groups of numbers or letters stored in a way conducive to performing iterative operations
- **Binary data**: information such as image data that is not meant to be directly manipulated by a programmer but transported to specialized software. In databases, columns of this type are known as *blobs*. In programming languages, the name for binary data varies widely.

## Programming

In informatics, a distinction is frequently made between "software development" and "database programming". The former are programs run directly on the computer and correspond to either the user interface or application server layers in the three-tier architecture. In, for example, an EHR system, the software development component provides the user interface and control structure that guides the system's functionality. The database programming involves subprograms that process data, such as loading a patient's record, pulling up today's appointments, or analyzing quality deficits in the treatment of diabetic patients.

### Database Programming

As discussed previously, relational database programming is done in SQL.

The core of all SQL code is the SELECT statement, which implements set theory to ask questions about the data. If we wanted to ask questions about the PATIENT table with one row per unique patient, we would use this format: SELECT <data elements> FROM PATIENT WHERE <constraint>. We can use aggregate functions, such as

```
SELECT avg(income) FROM PATIENT WHERE birth_
date>'01/01/1979'
```

This will return the average income of all patients born after January 1, 1979. We would use a join with a common column between the tables known as a "key" to answer questions involving multiple tables. A full discussion of SQL SELECT statements, including more complex joins and aggregate operators, is out of the scope of this chapter, but excellent online tutorials are readily available. SQL commands can be collected into small programs that are more complex than a single statement. These are called *stored procedures*.

### Software Development

Traditional software development is done through imperative languages, which issue a series of commands to the computer. There are a variety of styles, each with advantages and disadvantages. Broadly, these can be grouped into object-oriented and procedural styles.

#### Object-Oriented vs. Procedural Programming

In **object-oriented programming**, data structures can be built to have *properties* and *methods*. Properties are variables that the object holds, and methods are actions that one can perform on the variables. For example, there might be objects named Patient and Appointment. Patients could have a method named hasAppointment, which verifies whether the patient has a given appointment. This method would take an *argument*, a piece of data upon which the method operates. Our hasAppointment method's argument is an appointment object. The appointment object might have various properties such as date, time, clinic, and physician ID. The object definitions are templates for actual appointments and patients. These object definitions are *instantiated* for each specific case.

Java is a very popular object-oriented language. The language and many tools associated with it are freely available. Also, it is a cross-platform language, meaning that it will run on many types of machines. This is because Java runs on a *virtual machine* that interprets the Java program when it is run and converts it to the machine language of that particular machine.

Other notable object-oriented-only languages include: C++, which is the grandfather of all object-oriented languages and continues to remain the most efficient due to its native compilation; C#, Microsoft's virtual-machine Java-like language, which is easier to develop in but only runs on Windows; Ruby, a popular more recent language which also improves upon Java and is popularly used for web applications using the "Ruby on Rails" framework.

**Procedural programming** is more straightforward: entire programs share methods and global variables, and there are no objects. This structure has a significant disadvantage: the object paradigm makes it easier to organize and conceptualize large programs. Therefore, most languages that support procedural programming also support some type of object-oriented programming. Procedural languages are particularly useful as scripting languages. *Scripts* are short programs that control the functionality of other computer programs, most frequently webpages. Popular procedural languages that also support object-oriented programming and are widely used for scripting include Python (widely used in scientific programming), PHP, JavaScript (which powers the world wide web), and R (which is widely used in statistical modeling and data visualization).

A notable exception is the language C, a procedural language that does not support object-oriented programming and is also not well suited to scripting. Many of today's most complex software underpinnings (e.g., most operating systems) are written in some variant of C. C was developed long before object-oriented programming was invented or scripting was envisioned. Because it continues to be the most powerful and efficient high-level language available (despite its complexity), it is still widely used today.

New languages continue to appear to address the problems of the modern era. For example, Google's Go language (sometimes called Golang) is built around concurrency—executing multiple tasks simultaneously. This is in many ways a response to the growing popularity and prevalence of parallel and grid computing.

**Other programming paradigms**, such as functional programming, are of primary interest to mathematicians and computer scientists and are therefore out of the scope of this chapter.

### Control Structures

Programs don't just issue commands in order. Most imperative languages make extensive use of *control structures* to manipulate the flow of commands. SQL is an exception; SQL has control structures, but control structures are not a central component of the language because the primary motif is set theory. In imperative languages, control structures are central to the design of the program. Broadly, control structures can be broken down into *looping* and *branching*. A variant of looping is *recursion*, but the differences between these are out of the scope of this chapter.

A common programming design is to repeat some operation until a condition is true. This is done with a loop. A list of names could be looped over until all the names are processed. This is known as a *for* loop because operations are performed for all elements in a collection. There are also other types of loops, such as *while* loops, which operate while a certain condition is true (such as accepting new patients until the clinic closes). Branching occurs when the program takes a different direction depending on the value of a variable. This is done through an *if… then* statement.

### Compiled and Interpreted Languages

Languages are either *compiled* or *interpreted*. Compiled languages are converted into code that the computer can understand before running the program. Interpreted languages are converted to this machine language from scratch each time the program is run. Languages that run on virtual machines are a special case. A language run on a virtual machine is first compiled to *byte code*, a pseudo-machine language that is quickly translatable into machine language.

Therefore, a performance hierarchy emerges among programming languages: the fastest languages are natively compiled, the second-fastest languages run on virtual machines, and the slowest languages are interpreted. Of course, this hierarchy has some exceptions because of how specific features in the language are implemented. For example, Jython, a version of Python that runs in the Java virtual machine, is generally slower than the interpreted language Python. As computer speeds increase, this hierarchy is becoming less important, at least for high-level application development. Although an operating system or other core computer code that is run constantly should be written in a compiled language, many of today's user-facing applications are written in Java or Python. It is typical in scientific code to use an interpreted or bytecode language for most of the application and then write core functions (like image processing) in a compiled language for speed.

## Software Design Considerations

## Code Modularity, Reuse, and Performance

**Code Reuse** The ability of a programmer to understand the programming code she or others on a team have written is imperative to the success of a project. Therefore, many software development methodologies highly emphasize software documentation. Also, there are frequently multiple approaches to solving computational problems. In a team-based environment, frequently, the approach that is most easily understood by others (the most readable approach) is preferred.

**Modularity** Self-contained software code can be distributed in "libraries" that other software developers can use. Thousands of these libraries exist for any given language; they provide the functionality to the programmer quickly without the programmer having to dive into the source code of another developer. Because the libraries do not require source code, many commercial products provide libraries while retaining the confidentiality of their proprietary software code. Examples of libraries include packages to manipulate Microsoft Office documents from within a software program, packages to perform statistical analysis of data, or packages for animation and visualization. One well-known source for quality, free libraries is the Apache Software Foundation at www.apache.org.

**Performance** Often code readability is more important than performance, but for computation-intensive tasks (such as KDDM), performance is very important. The performance of computer algorithms can be determined mathematically through *complexity analysis*. The practical performance of computer programs is often judged through *profilers*, which are special programs that measure the speed of software under a variety of conditions.

## Methodology and Quality Assurance

**Software Development Methodology** A variety of organizational designs for developing software have been proposed. These tend to be combinations of two overall types:

- **Waterfall**: this is the traditional method of software development. A phase of requirements gathering occurs before any software is developed and requirements documents are assembled. Then the software development commences, followed by testing. This is a very robust and thorough method, but the final product is often either not what was envisioned by those providing the requirements or changes during the product cycle.

- **Iterative**: This is the antithesis of the waterfall model, in which a minimum of planning occurs at the beginning of the project. Rather, the software is developed in short cycles of planning, development, and testing. The iterative approach offers closer alignment with shifting user needs and complex changing environments. However, it also tends to focus on immediate needs instead of long-term goals. This can tend to make the developed software less thoroughly developed and less modular.

The two overall types are combined in many methodologies. The **spiral** method directly combines these two types. Each project is defined as a collection of many development cycles, some of which use more of a waterfall approach, and some are more of an iteration.

**Agile** methodologies collectively refer to a variety of rapid cycling software development, in which development, testing, and requirements gathering revision are closely fused [58]. Agile methodologies use the same approach as the spiral method (shifting between iterations and planning phases). Still, they try to be more flexible by doing less pre-planning of cycles and being more able to change as a project moves forward.

A popular agile approach is the **scrum** methodology, in which work is broken down into 30-day **sprints**, which begin with planning and requirements gathering and end with a new release of the product. The sprints are not defined before the sprint's beginning, making this approach very resilient to changing needs. All these methods still have the danger of focusing too heavily on short-term development goals, however.

**Quality Metrics and Testing** Many methodologies exist for ensuring quality software and for subsequently testing that software. Popular methods to build software with quality from the outset include *pair programming, code reviews, and software documentation before writing the code* [58]. Also, there is some evidence that more readable languages tend to lead to higher-quality software. Software testing is fundamentally important, no matter how much a development methodology emphasizes up-front quality. One robust approach is that the software developer creates *unit tests* as they develop their software. Unit tests are tests of an individual function of the software for a specific combination of inputs. A finished piece of software might have thousands of unit tests. If these tests are written as the software is developed, it is simple to perform *regression testing*, or running all the old unit tests, to verify that new features have not broken any old features. If a test that used to work no longer does, it becomes straightforward to find the change that broke that particular test.

**Verification and Validation** Software *verification* testing, like unit tests, compares the software to what it was designed to do and may be performed by the software developers or by dedicated testers. The end-users and requirements gatherers perform software validation testing. Validation makes sure that the software performs the function that it was originally intended to. Whereas verification finds bugs in the software, validation finds problems in design or requirements gathering.

### Other Considerations

**Open-Source** Much commercial software is closed source, meaning that the programming code used to develop the software is not available to the licensees. In open-source software, the code *is* made available [59]. However, the code could still be copyrighted and might have restrictions on changing or using it. Dozens of open-source licenses define exactly how to program code that can be used, changed, and redistributed in open-source software. Because the program code in commercial applications is often a trade secret, commercial software is more frequently closed source. The commercial software that is open source tends to have restrictive licenses to protect the copyright holders. For products where the goal is that their code is used for further innovation, licenses tend to be less restrictive, and the vendor company's main financial gain is through support contracts.

**Platform** The computer platform on which the software runs is another important consideration in choosing or developing software. As discussed previously, software written in some languages, such as Java, can be run on multiple computer platforms. Generally, however, the software is written for a particular operating system (such as Windows, Macintosh, or UNIX), a particular database platform (such as Oracle or SQL Server), or a particular web browser (such as Google Chrome or Microsoft Internet Explorer).

## Security

Computer security is a balance of two things: (1) preventing misuse of computer systems and data, and (2) enabling proper use of computer systems and data. We could ensure no misuse by turning off a computer and locking it in a vault, but that would defeat the second objective. The goal must be a balance of usability and minimal risk of misuse.

This section will frame the discussion of computer security in terms of the HIPAA Security Rule, which is the law for all computer systems containing patient-specific medical data. Still, the principles embedded in these regulations are good security practices for any type of data. Chapter 17 goes into much more detail on all the various considerations around cybersecurity.

## The Health Insurance Portability and Accountability Act

The Health Insurance Portability and Accountability Act of 1996 (HIPAA) includes a Security Rule section to establish security standards for the protection of *Electronic Protected Health Information* (*e-PHI*). The HITECH Act of 2009 extends HIPAA with additional penalties for e-PHI security breaches and additional rights for patients to view and limit access to their own data. The latest "Omnibus HIPAA Final Rule" was published in 2013 [60]. Many states have additional patient privacy regulations.

Here is the core of the Security Rule:

The HIPAA Security Rule [60] addresses the confidentiality, integrity, and availability of e-PHI on any computer system that creates, receives, maintains, or transmits such information. Organizations handling e-PHI (referred to as *Covered Entities* and their *Business Associates,* with whom they exchange e-PHI) are required to

1. Ensure the confidentiality, integrity, and availability of all e-PHI the covered entity creates, receives, maintains, or transmits.
2. Protect against any reasonably anticipated threats or hazards to the security or integrity of such information.
3. Protect against any reasonably anticipated misuses or disclosures of such information.
4. Ensure compliance by its workforce.

There are four types of technical safeguards to ensure the security of e-PHI: (a) access control, (b) audit controls, (c) integrity controls, and (d) transmission security. We will discuss each of these in turn.

(a) *Access Control* determines who has access to the data and consists of two parts: authentication and authorization.

    *Authentication* ensures that the user is who they say they are. Usually, this is by a username and password. Other options include smart cards and biometrics, such as fingerprint readers. Physical security, such as limiting access to selected workstations or smartphones, also aids authentication. *Two-factor authentication* means that two types of authentication are required in combination, such as a smart card plus a PIN (Personal Identification Number) or a password plus a controlled network location.

    The second component of Access Control is *Authorization*, which enables the user to access the computer systems, applications, and data necessary for their job function. A researcher is authorized to access data only for patients in their study. A physician is authorized to order lab tests and medications in a CPOE (Computerized Physician Order Entry) system for patients under his care.

(b) *Audit Controls* require computer systems to log activity, such as who viewed or modified a patient record, protect the audit logs from alteration and make the audit logs available for inspection.

(c) *Integrity Controls* consist of implementing policies and procedures to ensure that e-PHI is not improperly altered or destroyed.

(d) *Transmission security* refers to measures taken to prevent unauthorized access to e-PHI when transmitted over a network. Data encryption is a must, either by using a VPN (Virtual Private Network) or a point-to-point protocol such as SSL (Secure Sockets Layer). Computer systems first exchange encryption keys and then use those keys to scramble the data during transmission. Firewalls between organizations ensure that only authorized computer systems of Business Associates can receive e-PHI.

## The Common Rule

The Common Rule is a 1981 rule of ethics (revised in 2018) regarding biomedical and behavioral research involving human subjects in the United States [61]. It establishes the regulations governing Institutional Review Boards for oversight of human research through the Department of Health and Human Services Title 45 CFR 46 (Public Welfare) Subparts A, B, C, and D. The Common Rule is the baseline standard of ethics by which any government-funded research in the US is held; nearly all academic institutions hold their researchers to these statements of rights regardless of funding.

The main elements of the Common Rule [62] include requirements for assuring compliance by research institutions, requirements for researchers' obtaining, waiving, and documenting informed consent, and requirements for Institutional Review Board (IRB) membership, function, operations, review of research, and record keeping. The Common Rule includes additional protections for certain vulnerable research subjects: for pregnant women, in vitro fertilization, and fetuses, plus additional protections for prisoners and children.

## Malicious Attacks

The HIPAA Security Rule obligates the organization to protect e-PHI against reasonably anticipated threats. One such threat is a *brute force attack,* which consists of the attacker trying to guess an encryption key or a password by trying many different combinations until one works. The length of the key or password determines how long it will take an unauthorized party to guess correctly—the longer the key, the better.

In the *Man-in-the-middle attack*, the attacker inserts a malicious computer system on the network somewhere between two systems exchanging data. The system in the middle acts as a router, receiving and retransmitting data, but it also copies or even alters the data packets as they pass through, potentially compromising e-PHI or stealing passwords, without either legitimate computer system realizing that anything is wrong.

Malicious actors may exploit weaknesses in computer applications and operating systems to place their own software on a computer, which can open e-PHI to the intruder. Two of the most common exploits, buffer overflow, and code injection, are described below.

The *buffer overflow attack* sends the target system a larger data packet than it expects. The computer system accepts the packet into a reserved area of memory, called a buffer. The extra data in the super-sized packet exceeds the buffer size, writing the extra data past the end of the buffer into an area of memory used by executable code. Later, the computer executes the attacker's code, thinking it is the original code that was overwritten, and the attacker's code can do anything it wants to on the target system.

In the *code injection attack*, the malicious actor puts executable code into the input data fields of an application. The attacker surrounds the code with special "escape characters" that cause subroutines within the computer application to end their intended operation prematurely and misinterpret the rest of the input as code to execute. For example, an application might insert user input directly into an SQL statement sent to the database for execution. An *SQL injection attack* might answer an MRN prompt with "; SELECT * FROM ALL_USERS;" The ";" tells the database query engine to start a new command, and the select statement returns a list of all database accounts to the attacker.

## The Federal Information Security Management Act (FISMA)

The Federal Information Security Management Act (FISMA) is a United States federal law passed as part of the E-Government Act of 2002 [63]. It set the requirements for each federal agency to create, document, and implement programs that ensure security for the agencies' data and the systems that support the agencies' operations and assets, creating documented programs to use for securing said agencies' data and the systems they use for their operations and assets. Many federal research programs must obtain FISMA Authority to Operate (ATO) when participating in government Contracts or "Other Transaction" funding mechanisms. FISMA aims to prevent unauthorized access, use, disruption, modification, or destruction of information and information systems and requires a "FISMA boundary" around the software systems managing the data for the federal research program. By preventing misuse or attacks on the data and software, confidentiality, integrity, and availability are ensured. These systems must obtain an ATO to authorize system processing before and after operations begin, signifying the systems have detailed security plans, assigned security responsibilities to appropriate officials, and regularly reviewed the systems' security mechanisms.

## De-identified Data

The HIPAA Security Rule only applies to e-PHI, namely data that a third party can identify as belonging to a specific individual. HIPAA specifies 18 identifiers of an individual, such as name, social security number, address, certain dates, and implanted device serial numbers. You can *anonymize* or *deidentify* e-PHI by removing all of these identifiers. The modified data set is not e-PHI and is not subject to HIPAA regulations. As discussed in the HIE section earlier, this allows organizations to share de-identified data sets for research purposes.

When in doubt, seek out the advice of your organization's HIPAA Compliance Officer or IT Security Officer. They can help interpret and advise on security regulations for e-PHI. The consequences of a mistake can be devastating. The HITECH Act of 2009 provides penalties for negligence leading to an e-PHI breach that can add up to millions of dollars. Major breaches of security, defined as unauthorized access to 500 or more unencrypted patient records, requires notification of the local media and reporting the breach to the HHS, where the breach will be listed on the HHS public internet site, sometimes referred to as "The Wall of Shame" [64].

## Emerging Trends

As this chapter is being written, four emerging technologies appear to capture the most press and excitement. Only time will tell if they prove to be fruitful. The NoSQL database is considered to be a cheap, scalable solution that will become highly competitive with the relational database that is currently the mainstay of data analytics. Although that destination is premature, it clearly will open new worlds for extracting data from documents that could not be performed in a scalable manner 10 years ago.

The "App store for health" is another emerging trend that holds promise for opening the user interface of the electronic healthcare system to novel ways of presenting data and providing decision support. Such marketplaces allow Apps to be bought and sold to accommodate niche needs throughout the system by a large workforce of developers.

Big Data represents a third emerging trend, with sensors on the body collecting massive amounts of data. The ability to sift through the data to extract insights will define much of how we view physiology in the future.

Data Enclaves, especially on the cloud, is a fourth emerging trend [65]. The Data Enclave provides a release of data, often with PHI, into a computing environment that has been pre-loaded with programming tools and libraries. However, the networking configuration does not allow the data to travel out of a firewall boundary around the Enclave. The data can be viewed (often on a Virtual Machine) but not removed.

## Summary

Information technology is how all clinical informatics is ultimately expressed. The knowledge one has on the details of IT will figure into many implementation decisions. Data optimization, program efficiency, and attention to security will contribute greatly to the success of the informatician.

## Query Tools and Techniques: Resources

We have attempted to include relevant resources (including websites, articles, and books) in the References section when possible. However, we also wanted to highlight the following resources for query tools and common data models:

- **Patient-centered Outcomes Research Network:** https://pcornet.org/resources/
- **National COVID Cohort Collaborative:** https://covid.cd2h.org/n3c
- **Informatics for Integrating Biology and the Bedside:** https://www.i2b2.org/software
- **Observational Health Data Science and Informatics:** https://www.ohdsi.org/software-tools/

## Questions for Discussion

1. Clinical Data Warehouses store structured data in various Common Data Models (CDMs) and homegrown data models. What data models have your organization adopted or considered, and what economic and data considerations led to that decision?
2. If you were an informatics director tasked with developing software to build interactive reports on the various data facts in the hospital clinical data warehouse (e.g., the prevalence of uncontrolled diabetes over time), what programming language and software development methodology would you choose? Would you instruct your team to write the program as a series of SQL queries or use a higher-level language like Java?
3. In designing a physical network for a new informatics research lab, what key decisions must be made around network topology and network architecture? Draw several possible network architecture diagrams and discuss the pros and cons of each.
4. Imagine your organization wants to join a popular clinical data research network and participate in federated queries. What do you think are the most important considerations in ensuring that the shared data are appropriately de-identified and patient identity remains protected? What data sources would you use (e.g., notes, flowsheets, problem lists, etc.), and how would you think through designing an ETL process?
5. You have been awarded a federal contract to manage the data from a new clinical study sponsored by the NIH. What are the expected FISMA requirements you will need to adhere to within your "FISMA boundary?"

**Acknowledgements** Thanks to *Jim Meeks-Johnson, MA,* for his important contributions to the first edition of this chapter. His knowledge of network architecture and software design were invaluable in preparing both editions of the chapter.

## References

1. Ware H, Mullett CJ, Jagannathan V. Natural language processing framework to assess clinical conditions. J Am Med Inform Assoc. 2009;16:585–9.
2. Cook JA, Collins GS. The rise of big clinical databases. Br J Surg. 2015;102:e93–101.
3. Kimball R. The data warehouse toolkit: practical techniques for building dimensional data warehouses. New York: Wiley; 1996.
4. Livne OE, Schultz ND, Narus SP. Federated querying architecture with clinical & translational health IT application. J Med Syst. 2011;35:1211–24.
5. Overview—FHIR v4.0.1 [Internet] [cited 11 Aug 2021]. Available from: https://www.hl7.org/fhir/overview.html
6. Health and Human Services Department. 21st Century Cures Act: interoperability, information blocking, and the ONC Health

IT Certification Program [Internet]. Fed Regist. 2020;25642–961. Available from: https://www.federalregister.gov/d/2020-07419
7. Date CJ. An introduction to database systems: International edition. 8th ed. Upper Saddle River, NJ: Pearson; 2004.
8. Chamberlin DD. Early history of SQL. IEEE Ann Hist Comput. 2012;34:78–82.
9. Codd EF. A relational model of data for large shared data banks. Commun ACM 1983;26:64–9.
10. Codd EF. Further normalization of the data base relational model. Data base systems, vol. 6. Englewood Cliffs, NJ: Prentice-Hall; 1972. p. 33–64.
11. Nadkarni PM, Brandt C. Data extraction and ad hoc query of an entity-attribute-value database. J Am Med Inform Assoc. 1998;5:511–27.
12. Murphy SN, Weber G, Mendis M, Gainer V, Chueh HC, Churchill S, et al. Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2). J Am Med Inform Assoc. 2010;17:124–30.
13. BigQuery standard SQL [Internet] [cited 13 Aug 2021]. Available from: https://cloud.google.com/bigquery/docs/reference/standard-sql/migrating-from-legacy-sql
14. Amazon Redshift SQL [Internet]. Available from: https://docs.aws.amazon.com/redshift/latest/dg/c_redshift-sql.html
15. Sack J. Azure SQL databases compatibility with MSSQL Server [Internet] [cited 13 Aug 2021]. Available from: https://azure.microsoft.com/en-us/blog/default-compatibility-level-140-for-azure-sql-databases/
16. Bowie J, Barnett GO. MUMPS—an economical and efficient time-sharing system for information management. Comput Programs Biomed. 1976;6:11–22.
17. InterSystems IRIS Data Platform 2021.1 [Internet] [cited 23 Aug 2021]. Available from: https://irisdocs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls
18. Ghemawat JDS. MapReduce: simplified data processing on large clusters. In: 6th symposium on operating systems design and implementation. p. 137–49.
19. Apache Hadoop [Internet] [cited 13 Aug 2021]. Available from: https://hadoop.apache.org/
20. Neo4J Graph Database Platform [Internet] 2020 [cited 13 Aug 2021]. Available from: https://neo4j.com/
21. Van de Velde R, Degoulet P. Clinical information systems: a component-based approach. New York: Springer; 2003.
22. Mandl KD, Mandel JC, Murphy SN, Bernstam EV, Ramoni RL, Kreda DA, et al. The SMART Platform: early experience enabling substitutable applications for electronic health records. J Am Med Inform Assoc. [Internet]. 2012. Available from: http://jamia.bmj.com/content/early/2012/03/16/amiajnl-2011-000622
23. Mandel JC, Kreda DA, Mandl KD, Kohane IS, Ramoni RB. SMART on FHIR: a standards-based, interoperable apps platform for electronic health records. J Am Med Inform Assoc. 2016;23:899–908.
24. Klann JG, Mendis M, Phillips LC, Goodson AP, Rocha BH, Goldberg HS, et al. Taking advantage of continuity of care documents to populate a research repository. J Am Med Inform Assoc. 2015;22:370–9.
25. Visweswaran S, Becich MJ, D'Itri VS, Sendro ER, MacFadden D, Anderson NR, et al. Accrual to clinical trials (ACT): a Clinical and Translational Science Award Consortium Network. JAMIA Open. 2018;1:147–52.
26. Klann JG, Phillips LC, Herrick C, Joss MAH, Wagholikar KB, Murphy SN. Web services for data warehouses: OMOP and PCORnet on i2b2. J Am Med Inform Assoc. 2018;25:1331–8.
27. OHDSI Data Network 2020 [Internet] [cited 13 Aug 2021]. Available from: https://www.ohdsi.org/web/wiki/doku.php?id=resources:2020_data_network
28. Hripcsak G, Duke JD, Shah NH, Reich CG, Huser V, Schuemie MJ, et al. Observational health data sciences and informatics (OHDSI): opportunities for observational researchers. Stud Health Technol Inform. 2015;216:574–8.
29. PCORnet Common Data Model (CDM) [Internet] [cited 14 Aug 2021]. Available from: http://www.pcornet.org/pcornet-common-data-model/
30. Klann JG, Buck MD, Brown J, Hadley M, Elmore R, Weber GM, et al. Query health: standards-based, cross-platform population health surveillance. J Am Med Inform Assoc. 2014;21:650–6.
31. McDonald CJ, Overhage JM, Barnes M, Schadow G, Blevins L, Dexter PR, et al. The Indiana Network for Patient Care: a working local health information infrastructure. Health Aff. 2005;24:1214–20.
32. Collins FS, Hudson KL, Briggs JP, Lauer MS. PCORnet: turning a dream into reality. J Am Med Inform Assoc. 2014;21:576–7.
33. Platt R, Carnahan RM, Brown JS, Chrischilles E, Curtis LH, Hennessy S, et al. The U.S. Food and Drug Administration's Mini-Sentinel program: status and direction. Pharmacoepidemiol Drug Saf. 2012;21 Suppl 1:1–8.
34. NIH makes first infrastructure awards to support research on post COVID conditions [Internet]. 2021 [cited 16 Aug 2021]. Available from: https://www.nih.gov/about-nih/who-we-are/nih-director/statements/nih-makes-first-infrastructure-awards-support-research-post-covid-conditions
35. Raisaro JL, Troncoso-Pastoriza JR, Misbach M, Sousa JS, Pradervand S, Missiaglia E, et al. MedCo: enabling secure and privacy-preserving exploration of distributed clinical and genomic data. IEEE/ACM Trans Comput Biol Bioinform. 2019;16:1328–41.
36. Witten IH, Frank E, Hall MA, Pal C. Data mining: practical machine learning tools and techniques (Morgan Kaufmann series in data management systems). 4th ed. San Francisco, CA: Morgan Kaufmann; 2016.
37. Hastie T, Tibshirani R, Friedman J. The elements of statistical learning: data mining, inference, and prediction (Springer series in statistics). 2nd ed. New York: Springer; 2016.
38. Bramer M. Principles of data mining. 4th ed. London: Springer; 2020.
39. Linden G, Smith B, York J. Amazon.com recommendations: item-to-item collaborative filtering. IEEE Internet Comput. 2003;7:76–80.
40. Klann J, Schadow G, McCoy JM. A recommendation algorithm for automating corollary order generation. AMIA Annu Symp Proc. 2009;333–7.
41. Carter JS, Brown SH, Erlbaum MS. Initializing the VA medication reference terminology using UMLS metathesaurus co-occurrences. Proc AMIA Symp. 2002;116–20.
42. Estiri H, Strasser ZH, Murphy SN. Individualized prediction of COVID-19 adverse outcomes with MLHO. Sci Rep. 2021;11:5322.
43. Mantri N. Applications of autoencoders [Internet]. OpenGenus IQ: computing expertise & legacy. 2019 [cited 16 Aug 2021]. Available from: https://iq.opengenus.org/applications-of-autoencoders/
44. Miotto R, Li L, Kidd BA, Dudley JT. Deep patient: an unsupervised representation to predict the future of patients from the electronic health records. Sci Rep. 2016;6:26094.
45. Kahn MG, Callahan TJ, Barnard J, Bauck AE, Brown J, Davidson BN, et al. A harmonized data quality assessment terminology and framework for the secondary use of electronic health record data. eGEMs [Internet]. 2016;4. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5051581/
46. Klann JG, Weber GM, Estiri H, Moal B, Avillach P, Hong C, et al. Validation of an internationally derived patient severity phenotype to support COVID-19 analytics from electronic health record data. J Am Med Inform Assoc [Internet]. 2021. Available from: https://doi.org/10.1093/jamia/ocab018
47. The OSI model's seven layers defined and functions explained [Internet]. 2015. Available from: https://support.microsoft.com/kb/103884.
48. Booch G, Rumbaugh J, Jacobson I. The unified modeling language user guide (Object technology series). 2nd ed. Boston, MA: Addison-Wesley; 2005.

49. Hirsch F, Kemp J, Ilkka J. Mobile web services: architecture and implementation. Chichester: Wiley; 2007.
50. Richardson L, Ruby S. RESTful web services. Farnham: O'Reilly; 2008.
51. Fielding RT. Architectural styles and the design of network-based software architectures [PhD]. Taylor R, editor. University of California, Irvine; 2000.
52. Orfali R. Client server survival guide. New York: Wiley; 1999.
53. An overview of the ice platform 2015 [Internet]. 2015. Available from: http://zeroc.com/overview.html
54. McConnell S, McConnell SM, McConnell SC. Code complete: a practical handbook of software construction. Microsoft; 1993.
55. Gamma E, Helm R, Johnson RE, Vlissides J. Design patterns: elements of reusable object-oriented software. New York: Addison-Wesley; 2015.
56. Hutton DM. Clean code: a handbook of agile software craftsmanship. Kybernetes. 2009;38:1035. Available from: https://doi.org/10.1108/03684920910973252.
57. Martin RC. Clean code: a handbook of agile software craftsmanship. Upper Saddle River, NJ: Prentice-Hall; 2009. £27.99, ISBN: 9-780-13235-088-4
58. Beck K, Andres C. Extreme programming explained: embrace change. Upper Saddle River, NJ: Pearson; 2005.
59. Raymond ES. The cathedral & the bazaar: musings on linux and open source by an accidental revolutionary. Sebastopol, CA: O'Reilly; 2001.
60. Health and Human Services Department. Modifications to the HIPAA privacy, security, enforcement, and breach notification rules under the health information technology for economic and clinical health act and the genetic information nondiscrimination act; other modifications to the HIPAA rules [Internet]. Fed Regist. 2013;5565–702. Available from: https://www.federalregister.gov/d/2013-01073
61. Department of Homeland Security, Department of Agriculture, Department of Energy, National Aeronautics and Space Administration, Department of Commerce, Social Security Administration, et al. Federal policy for the protection of human subjects. Final rule. Fed Regist. 2017;82:7149–274.
62. Korenman SG, Shipp AC. Teaching the responsible conduct of research through a case study approach: a handbook for instructors. Washington, DC: Association of American Medical Colleges; 1994.
63. Rodrigues JJPC, de la Torre I, Fernández G, López-Coronado M. Analysis of the security and privacy requirements of cloud-based electronic health records systems. J Med Internet Res. 2013;15:e186.
64. U.S. Department of Health & Human Services Breach Portal: breach of unsecured protected health information [Internet] [cited 17 Aug 2021]. Available from: https://ocrportal.hhs.gov/ocr/breach/breach_report.jsf
65. Al-Qahtani MS, Farooq HM. Securing a large-scale data center using a multi-core enclave model [Internet]. In: 2017 European modelling symposium (EMS). 2017. Available from: https://doi.org/10.1109/ems.2017.45.