Understanding electronic health records and third-party application databases

Choose the proper data model prior to a database purchase

Shahid N Shah

November 02, 2010

Proper healthcare IT data modeling is difficult because the healthcare domain is both broad and deep. The number of data concepts and ways in which they can relate to one another is extensive. There are a variety of participants, such as patients (as users and subjects of record keeping), physicians, nurses, clinicians, business staff, and administrators. Each participant has a slightly different view of the care processes they are involved in. After you incorporate the participants, you run into problems due to the convoluted nature of clinical knowledge and how very few processes are perfectly repeatable across time and patient populations. These and other difficulties cause IT developers, from a variety of disciplines, to take shortcuts in their data models to meet deadlines. This article explains why data models in healthcare matter, and how you can ensure that new electronic health records application purchases you make have extensible databases to meet future business and clinical needs.

The data model matters

In healthcare enterprises, applications come and go but protected health information (PHI) and data live on forever. Since the beginning of the computer industry when we move from legacy systems to modern architectures, we often leave behind applications but we usually bring the data along into new systems. Even though data is so important, we in healthcare IT don't seem to spend the necessary time to structure our schemas and databases so that they are easier to maintain in the future. Often, we often don't design our data models solidly nor test them well. Although things should have improved when we began using object-oriented programming, it has actually caused us to compromise data management principles unintentionally.

Large-scale, object-oriented applications development, which is new to healthcare, has not improved our ability to manage data using sound data management principles. This is because in the world of objects, data is considered simply part of the state of an object. For example, a person's object demographics are considered the state of the person object. An encounter's data is considered the state of a transaction object. That makes sense from an application point of view; if an application contains a person object and a transaction object then the data actually is the state of those objects. When multiple applications have to share the data, such as a MUMPS (the
programming language) application alongside a Java application, things get more complicated and the data in the database is no longer just a temporary state of an object but a long-term record.

### Definition of terms

- **EHR**: Electronic Health Records
- **PHI**: Protected Health Information
- **NHIN**: Nationwide Health Information Network

Most modern object-relational mapping tools try to hide SQL and the underlying relationships of the database. This is great if a relational modeler designs the data model from the ground up and then the object-relational mapping is applied on top of it. Many times, though, the object-relational mapping is used to generate the schema, which means that an application programmer is determining the data model, not a data modeler. What's wrong with that? If the application were the most important thing, that's fine. However, if we want the data to live forever, we need to define the data, its usage, and how it's structured before we design the application, instead of the other way around.

Our friends in the healthcare IT applications development community need to learn that data modeling is not just a technical exercise. You can't define a data model with a bunch of engineers and other geeks sitting around a table. Data modeling is about understanding all of the uses of the data, the relationships and attributes involved in the data, and, most importantly, how the data management approach will grow and change in the future. It's the last part (extensibility of the database) that developers often forget when designing most systems. All this involves direct communication with end users, stakeholders, and other non-technical personnel. Too often, databases are treated as a file cabinet—just let your application toss whatever is necessary in there and then deal with organizing it later.

When you're building your own systems, you should attempt to use tested and proved data models from existing sources. If you don't have experienced data modelers, look at Len Silverson's three-volume work *The Data Model Resource Book: A Library of Universal Data Models for All Enterprises* (see Related topics). Buy the three books, try to comprehend what an extensible data model looks and works like, and then hold your developers and database administrators accountable for it. If you're building a healthcare meta model, there are also books available that cover universal meta models. Unlike a few years ago, there are books that provide de facto, well-designed, data models so you're not on your own. If a developer comes to you with a data model but can't provide references and patterns for how he arrived at his model, tell him to use one described in a book. There's no time to keep reinventing the wheel.

When you're buying new systems, spend more time evaluating the database design than the application's user interface (UI). You can change the UI easily in the future but changing the data model is not trivial, and it is worse if the vendor changes the data model. If a vendor does not let you analyze or study their data model, don't buy from them. That's like a car dealer selling you a car without allowing you to check the engine. When you're evaluating a data management system (and in healthcare IT, almost all systems are data management systems) focus on the data, not the applications. You could even use well-known designs from books, such as the Silverston books, to make your vendor compare their data models against.
With all of the talk around patient-centric architecture, you'd think that the vendors would have a data model to support it. Of course, we all know that they don't and we need to hold them accountable for it. Moreover, because regional health information organizations, the Nationwide Health Information Network (NHIN), personal health records, and electronic health records are identifying new integration needs, data models are even more crucial. Applications will continue to be chopped up in a service-oriented architecture, making the central database even more powerful.

The lesson here is applications come and go but data lives on forever, so don't ignore the data model.

**Plan for your EHR software escape hatch**

You will spend weeks or months in the sales and demo cycle selecting an EHR application. You will take time to consider all workflows, test drive the UI, and make sure training goes smoothly. You will also try to ensure that deployment will be easy. Do not forget, however, to plan how to get out of an application or system after it's been installed and running for a while.

Why is getting out important? Every application looks better in a demo than in a working environment and every solution becomes legacy sooner or later. Almost every system will be replaced or augmented at some point in time, but the data you store in the system won't be discarded. You understand that the cost of acquisition (barrier to entry) is something you need to calculate. But, the barrier to exit or switching cost is also something you must calculate at the time you decide which systems to purchase.

If you can't answer "In six, eighteen, or twenty-four months, how will we be able to move on to the next better technology or system?", then you've not completed your due diligence in the sales cycle. Vendor salespeople are reluctant to answer "How do I leave your system?". You will need to press hard and ask for a plan before signing any contracts. It is crucial that you focus on the database and how well it's designed for export and import. When preparing a Request for Information, or a Request for Price, ask vendors specific questions about how easy it is to get out of their technology in addition to how easy it is to deploy and interoperate. Put in specific test cases and have your staff consider this when they are looking at all new purchases. Here are some specific factors to consider:

- Do you own your data or does the vendor? If you don't have crystal clear statements in writing that the data is yours and that you can do whatever you want with it, don't sign the contract. Look for a new vendor.
- Is the database structure and all data easily accessible to you without involving the vendor? If only your vendor can see the data, you're locked in so be wary. Find out what database the vendor is using and make sure you can get to it directly without needing their permission.
- Are the data formats that the system uses to communicate with other vendors open? If not, you don't own your data. Be sure that at least Continuity of Care Record and Continuity of Care Document formats are available and that all document data is accessible in standard PDF or Microsoft Office formats. Discrete data should be extractable in XML or Health Level 7 (HL7) formats.
• How much of the technology stack is based on industry standards? The more proprietary the technology, the more you’re locked in.
• Are all the programming APIs open, documented, and available without paying royalties or license costs? If not, when you try to get out it will cost you a lot.

Key features of extensible healthcare data models

The following are some of the key features you should look for in a healthcare data model. This list is not exhaustive but does form the basis of good models. Consider these features as conceptual requirements for extensibility, not necessarily for performance or scalability. The physical database schemas should capture the appropriate conceptual requirements in a high-performance and scalable manner.

Person and organization models

Modern and extensible databases model patient (consumer), physician, nurse, staff member, administrator, contact, insurance policy holders, and related data as Person records. Instead of having a separate table for each type of person (for example, a different table for a patient versus a physician), you should try to model the different person types in a single inheritable and related table.

The same goes for organizations. Facilities, tenants, hospitals, insurance providers, departments, clinics, administration, and related data should be grouped into something conceptually called an organization. Any entity that isn't a Person type will likely fall into the Organization record type category—a single table with appropriate attributes should work fine.

Figure 1 shows how in the top (suboptimal and harder to manage) tables we have duplicated the same kind of data in the different tables. In the bottom we see the optimal approach where all of the common fields are in the Person and Organization tables while the unique elements are one-to-one related and contain the special fields.
Support for robust patient identification and de-duplication

Most healthcare databases will only have a single identification value for a person (patient, provider) or maybe just a few. Good data models, however, allow an unlimited number of identifiers for any entity—a primary key for internal consistency plus any number of external identifiers. Every Person record should allow an extensible set of identification values to use for both ID lookups and de-duplication requirements that crop up when integrating multiple systems.

In Figure 2 we see on the left the suboptimal way to create identifiers; attach each identifier as another column in the Person table. On the right we see the optimal and easier to maintain mechanism where we create a new table called Person_Identifier and a related Identity_Type table. The SSN and Medical Record Number are rows in the Identity_Type table and the Person_Identifier table references the appropriate identity_type_id (for example, SSN) and store the actual 000-00-9999 type value in the identifier column. This model allows you to have as many different identities as necessary into the future.
Support for separation of PHI from clinical and transactional attributes

Typically, databases are modeled with all related attributes in one table. For example, a Person table might contain a person type (for example, patient), the patient’s name, ID, and related demographic information. While easy to model, when identification data and transactional data reside together they are more of a security risk than if separated into different databases. A good design is to put PHI data into one database (configured with proper security), and put the clinical, business, and other attributes into another database.

In Figure 3 we see that DBx, an arbitrary patient database, has PHI intermingled with non-PHI data. It’s easier to protect data when the most important data (PHI) is separated into DBy, a separate database, and the original DBx has only the non-protected information. When the PHI is segregated this way, application developers have to use additional services and do more work to follow PHI release rules; they are less likely to expose PHI in applications when it is inconvenient for them.

Support for multiple simultaneous entity roles

Each entity in the database, such as person or organization, should be able to support multiple entity roles. We have already described that a Person record should be created in a common table for patients, physicians, nurses, and so on, and why that makes sense. However, think about
the scenario where a nurse at a hospital may also be a patient in the same hospital. Instead of duplicating records, the same record could take on dual roles; in this case as both a nurse and a patient.

In Figure 4, on the left we see the enhanced person model and how it is limited to a person having only a single role (type) in an organization, meaning a person can still only be a patient or a physician but not both simultaneously.

If, instead, we break off a new table called Person_Type_In_Org, we can easily model the same person record with multiple simultaneous types (that is, in one organization they could be a patient while in another organization they could be a physician).

**Figure 4. The enhanced person model**

Support for viewing data based on security roles

Modern healthcare databases will model hundreds of resources, which include granular items such as columns, rows, and tables. Resources should be classified and grouped into permissions, and multiple permissions grouped into security roles. The types of roles include the following, and whenever possible should be managed inside the database not in the applications:

**Setup, Infrequent, Routine**

These roles are used most frequently by policies enforcing the principle that damage is avoided by restricting infrequent and setup functions to more senior people. For example, creating an organization and a fee schedule is set-up, updating a fee schedule is infrequent, and actually using a fee schedule is routine.

**Internal Organization, External Organization, Employee, Patient**

A portal approach to all major entities such as people, organizations, and encounters allow each entity and its associated data access to be grouped into special roles based on who is viewing or modifying the data. For example, patients may view and modify their own demographics but not modify their physician notes or medications. As another example, organizations at a roll-up level may see aggregate data but not detail data.

**Medical, General, Financial**

There is a great deal of data grouped by work being performed at particular locations or times. For example, a nurse may get access to medical data but not financial data. Likewise, an administrator may view claims and billing information but not get details about medications prescribed. Temporal data, or time-based roles, are also possible by allowing users to have access only at particular times and at a particular location.
Component, Workflow-related

These technical buildingblocks allow security at lower levels so that hacking attempts do not bypass application-level security. Workflow-related security can actually allow the security roles to change for various steps of a process if necessary.

Support for long-term storage and change management (revision control) of all entity attributes

When data lives for a long time, it can change. Extensible databases support long-term storage, change management of the database structures, and revision control of the data (records). A simple example is the gender of a person, something we assume never changes. However, realistically we know that gender change operations do occur and when they do it's important for a physician to know that a current gender is a particular value today and that it was another value prior to changing. Good database designers treat all data as changeable over time and keep track of all prior versions of every record over time. Good revision control must be a part of every healthcare data model.

Support for multiple applications and devices within the same structures

Many engineers create databases as containers for specific applications, which usually is a bad idea. When creating a database always assume that multiple applications will write to the same database. This means that for every record written you should track which application wrote or changed the record and perhaps on which device.

Conclusion

This article covered why data models in healthcare matter and how you can ensure that new EHR application purchases you make have extensible databases that meet business and clinical needs well into the future. Follow some simple rules, such as those presented in this article, to help ensure that you can identify shortcuts that may have been taken in data models designs, which lead to long-term maintenance problems, and avoid them.
Related topics

- **The Data Model Resource Book: A Library of Universal Data Models for All Enterprises**: Len Silverson's three-volume work provides a foundation of patterns for data modelers to expand upon.
- **Special Publication 800-122: Guide to Protecting the Confidentiality of Personally Identifiable Information (PII)** (National Institute of Standards and Technology, April 2010): Use this publication to help you determine what is privacy data and how you protect it.
- **National Institute of Standards and Technology (NIST)**: Visit the NIST website.
- **Personally Identifiable Information (PII) and Health Insurance Portability and Accountability Act (HIPAA)**: Read the Wikipedia references.
- **A Tour of the Healthcare New Media Marketing Landscape**: Read a presentation about the various techniques hospitals can use to market their services online in this blog by Shahid N. Shah.
- **HITECH Survival Guide**: Learn more about the Health Information Technology for Economic and Clinical Health Act (HITECH) Act
- **Ten Usability Heuristics**: Get more detail on the ten general principles for user interface design from Jakob Nielson.
- **NHIN Direct**: Learn more about this project to expand the standards and service definitions that, with a policy framework, constitute the NHIN.
- **Open Health Tools**: Learn more about this open source community that is dedicated to improving the health of people through the transformation of health information technologies.
- **developerWorks WebSphere Portal zone**: Find information and resources on IBM WebSphere Portal, a composite application or business mashup framework and the advanced tooling needed to build flexible, SOA-based solutions.
- **developerWorks podcasts**: Listen to interesting interviews and discussions for software developers.
- **IBM product evaluation versions**: Download and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

© Copyright IBM Corporation 2010
**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)