Chapter 2
Database Design from Requirements to Implementation

In this chapter we discuss the differences between traditional database modeling and database design, the UML diagrams that can be used for database design, and some of the differences between using the UML and more traditional techniques. We also explain the case study that will be used throughout the remainder of the book.

Database Modeling versus Database Design

Database Modeling

Database modeling is generally focused on logical and physical database models. A logical database model is composed of entities and attributes, and it includes relationships between the different entities, which can be mandatory or not. The logical model consists of a normalized model that is generally set to third normal form. It includes many elements that make up a database, but it is not specific to any software or database implementation. Performance factors are not a major consideration at this point nor are the applications that will be using the database. The main concern is building a model of what the database would look like when capturing the data needed by the users.

The denormalization process begins with the physical database model. The database team takes the work performed in the logical model and starts to optimize it for queries, specific database implementations, and applications that may talk to the database. More specific items, like views, are added to make
working with the database easier for users. The physical database model also needs to be mapped back to the logical database model; they can be very different, and each has its own purpose. As changes occur, the team must update one model from the other and maintain that mapping between them.

Database Design

While database modeling focuses mostly on depicting the database, database design encompasses the entire process from the creation of requirements, business processes, logical analysis, and physical database constructs to the deployment of the database. For example, in database design, physical data modeling includes the modeling of not only tables and columns but also tablespaces, partitions, hardware, and the entire makeup of the database system. Database design includes uncovering and modeling requirements, the business processes (as they are today and where they are going in the future), the activities of the business, the logical models, and the physical database models, as well as addressing issues of what information is needed, how the different parts relate, how applications communicate with the data, and how the entire system is implemented.

Data Modeling Today

There are many tools and notations that address data modeling, but generally they are very focused on the implementation of the database. These tools and notations concentrate on logical and physical database modeling, usually ignoring other aspects of the business and its requirements. If you want to model and understand the business processes, requirements, and rules, you need additional tools and notations. If you want to understand the applications and how they relate to the database, once again you must move to other resources. This makes communication, reuse, and interoperability quite difficult, if not impossible.

There have been many methodologies, tools, and notations that championed different ways to best model, design, and build software applications from analysis through to database implementation. Some of these methodologies were very strict in the process and very tool intensive. If you wanted to customize these to fit how your organization did business, often you were out of luck. You had to do it the way prescribed by the tool or methodology. Since software development is an iterative process performed by different teams with different levels of experience and often in varying phases of the project’s life cycle, it became quite difficult to work in such an environment. Working in such a linear path made it very difficult to accomplish work that needed to be done immediately without going through the other steps first. It was also quite
difficult to go back and fill in some of the blanks once a step was skipped. The tools that support such a life cycle are often called upper CASE tools or full life-cycle CASE tools. Over time, people decided that a very strict methodology wasn’t going to work in an environment that needed flexibility. They moved away from the upper CASE tools into what we call lower CASE tools. The organizations that moved to these types of tools and methodologies began to adopt best-of-breed solutions.

**Best-of-Breed Solutions versus Full Life-Cycle Solutions**

The majority of companies that moved to best-of-breed solutions have run into a new problem. How do the different teams involved in the development effort communicate requirements and changes? Using the full life-cycle tools and methodologies, although quite cumbersome, gave the business analyst, application developer, and database designer the ability to work together in the same environment with the same artifacts and information. Moving to a best-of-breed solution gives the different developers the ability to choose what is best for their subject matter but leaves the other teams behind. Often the business analyst is working with a documentation tool while the software developer is working with code or some UML models and the database designer is working with his or her own database modeling tools. The teams, which are working toward a common goal to solve a specific business problem, have now branched off into their own worlds and the door of communication has been slammed shut.

The relative inflexibility of the full life-cycle tools may have made doing an entire job difficult, but they at least brought the different teams together to solve the business problem that was placed before them. We have been involved with many companies that bring the business analysts, application developers, and database designers into a meeting and discover it’s the first time they have met each other. The first question we ask in such a situation is, “If you don’t even know each other and you are all working together to solve the same business problem, how do you communicate requirements, especially when they change?” The answers are usually the same. The company has some really large requirements document that gets updated occasionally and an e-mail goes out to notify everyone. For most people, this is a pretty difficult and ineffective way to communicate changes. We have never worked on a project that had no requirements changes from inception to the end of the project. (If there were a company with such projects, everyone would want to work there.) How can teams, sometimes with hundreds of people involved, communicate the requirements, especially if the requirements change often?

Using the full life-cycle tools, since everyone was working in the same environment, made it easy to communicate the requirements and their changes.
through some sort of repository or data dictionary that was common across the
tool and methodology. But going to a best-of-breed solution has put an end to
the common information. Often teams meet to define requirements for build-
ing enterprise architectures without taking into account the other teams that
should be involved. Because the teams are using different tools and processes,
they often create their own plans and architecture, which the teams then pass
on to the other teams involved, many times causing wasted effort. If the ana-
ysts are working with the developers, for example, to build an enterprise archi-
tecture without bringing some of the database design team into the fold, there
is a good chance that the architecture being designed will miss some important
artifacts needed by the database team. We are not recommending that every-
one from every team be involved in all meetings, but each team should be rep-
resented. This will help to avoid the problems of six different definitions for
customer or architectures that have to be thrown away because the database
that has been implemented can’t support it.

A Happy Medium

Having full life-cycle tools hasn’t proved to be a good answer for bringing teams
together. Also, nobody likes a process that is so strict that they can’t customize
it to do what they want, when they want, and with the parts they want. The
best-of-breed option has provided a good solution for individuals, but it ignores
the fact that even though the reporting structures may be different and each
person may be working on different parts of applications, everyone still needs
to work together as a team and share information readily.

The UML has the best of both worlds. There are many tools like Rational
Rose, which will support the life cycle using the UML, yet they are flexible
enough to do what you want when you need it. Using the UML as a language
for the entire life cycle of the development effort allows all of the teams in-
volved to work together in one way but to do their own part as needed. Since
the UML is “process agnostic,” its use can be altered as needed to fit into your
company’s structure and processes.

UML Diagrams for Database Design

There are many types of UML diagrams available to help database designers do
their jobs. These diagrams can be used for capturing requirements, depicting
deployment, and everything in between. The different UML diagram types are
listed in Table 2–1.
Table 2-1  Descriptions of the UML Diagrams

<table>
<thead>
<tr>
<th>Diagram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case</td>
<td>The use case diagram is a model of the system’s intended functions and its environment that supports the business processes. This model serves as a contract between the customer and the developers.</td>
</tr>
<tr>
<td>Interaction</td>
<td>Interaction diagrams are either sequence or collaboration diagrams, both of which show the interaction of objects within the system. They can be used to understand queries that will affect the database and even help build indexes based on the information modeled.</td>
</tr>
<tr>
<td>Activity</td>
<td>Activity diagrams show the flow of a process. They can be used to show a high-level view of the business and how it operates.</td>
</tr>
<tr>
<td>Statechart</td>
<td>Statecharts capture the dynamic behavior of the system or objects within the system.</td>
</tr>
<tr>
<td>Class</td>
<td>Class diagrams are logical models that show the basic structure of the system.</td>
</tr>
<tr>
<td>Database</td>
<td>The database diagram depicts the structure of the database including tables, columns, constraints, and so on.</td>
</tr>
<tr>
<td>Component</td>
<td>Component diagrams show the physical storage of the database, including the database management system, tablespaces, and partitions. They can also include applications and their interfaces used to access a database.</td>
</tr>
<tr>
<td>Deployment</td>
<td>Deployment diagrams show the hardware configuration that is used for the database and applications.</td>
</tr>
</tbody>
</table>

Why Use the Various UML Diagrams?

UML modeling has come into mainstream use through the pathway of object-oriented analysis and design (OOAD), but it can be used for many different types of analysis and design modeling without ever building object-oriented applications. Development of the language was driven by the need to understand object-oriented development and how the architecture of a project affects the outcome. However, don’t exclude the use of the UML just because you are not building object-oriented applications—the UML can be used for basically any type of analysis and design project, whether or not the database or systems are object oriented, even outside of software. An example of this is an auto manufacturer that models the way its cars are assembled, particularly the specifications of the parts that third-party vendors supply. The company uses these UML models for quality control and to provide the models to prospective vendors who are bidding to build a specific part.

Being able to visualize the requirements helps the development team understand more easily and quickly the impacts of changes. Having models like
use cases and activity diagrams helps the teams examine such questions as, “What will happen if . . . ?” It is much easier for people to understand a quick picture that shows who will do activity x, then y, then z, rather than trying to read numerous paragraphs of text that describe the same information. Textual descriptions can be interpreted differently by the different people who are reading them.

The goal of using the UML for business processes, application development, and database modeling is to tie development teams together and make sure that organizations no longer build enterprise architectures without involving all teams that are important to the process. Building a team that is cross functional with the specific areas of expertise helps create an architecture that can be built and supported by all parties involved. If the different teams work in isolated silos, they eventually lose touch with the rest of the project. In many situations, individuals uncover new requirements, but often this information is not fed back into the overall project. As changes occur throughout the project life cycle, the UML diagrams can be updated so that everyone involved can understand the changes impacting their respective areas.

Sometimes just the diagrams are not enough. Metadata is very important to describe what has been modeled. Making sure, for example, that the word *customer* means the same to all groups and isn’t used in different ways is critical. You need to do a very good job of capturing this ancillary information. Again, this is where having everybody working in one language and notation helps. From the beginning, you can all get together to build the initial requirements and develop an early understanding of what *customer* means before breaking off into separate teams to begin deeper analysis and design in your specific areas.

Also, with the use of stereotypes, you can extend the UML to fit your needs even more specifically. A stereotype is a UML modeling element that extends the existing elements within the UML metamodel without directly changing the metamodel. Stereotyping a UML element causes it to act as something else, and by acting as that new element it has specific properties. To put it in database terms, for example, consider a table as the base element. When you stereotype it as an entity, this table now behaves as an entity and has attributes instead of columns. However, in the metamodel they are mapped to the same place. You may also add some additional tagged values based on the stereotype, which are then added to the metamodel.

### The UML Differs from Traditional Database Modeling

Traditional database modeling promotes the basic theory that the database is the backbone of the system and everything revolves around that database.
Although it is true that much—if not most—of the important information of
the organization lives within the database, the database cannot stand on its
own, and there are many other things that make up the company and its infor-
mation. Without the applications to open the database to employees, there
would be no accessible data. Without customers and transactions, there would
be no information for the database. Without a business for which to build the
database, there would be no reason to have a database. For these and many
other reasons, the database must exist together with the rest of the organiza-
tion and must be considered just one piece of the puzzle that must coexist with
all the others.

The previous statement may seem obvious, but it is not evidenced in many
of the companies we have worked in and visited. The database team often
works on its own without open doors of communication. The information it
captures is based on the database the team members are building and not the
entire system that is needed. The blame should not be put onto the database
team, though; the tools and methodologies that support the different types of
design have led the database team down this road, and most people have not
yet begun to move outside of the box into which they have been put. Using the
most commonly available tools and the most prevalent methodologies, organi-
zations have chosen to split the teams involved in requirements definition,
development, and database management, enforcing the barriers and making
communication difficult.

Bringing in the UML enables a common language for all teams involved and
starts to break down those walls, reuniting people into one development team.
Traditional database modeling concerns itself mainly with the database. The
database is very important, and you must concentrate on your subject matter
area. The problem begins when people focus so hard on their subject matter
areas that they don’t think outside of that subject and don’t even communicate
beyond their walls. Bringing a common language like the UML into the mix
doesn’t require you to do your database modeling differently—it just means that
the database modeling that you are doing today must expand outside of just the
database structures and become a part of the entire analysis and design process.

In the process that we follow throughout this book, we explore the various
parts of the development process, focusing on the job of the database analyst
and designer. Using the UML doesn’t inhibit the way that you traditionally de-
sign the database, although the notations may look a little different compared
with the familiar old ones. When designing the database with the UML, you still
have tables, columns, triggers, constraints, and the other elements that you gen-
erally use when modeling and designing a database. They just may be described
a little bit differently and will definitely be more easily communicated to the
rest of the teams involved with the development process.
The UML gives you the ability to model, in a single language, the business, application, database, and architecture of the systems. By having one single language, everybody involved can communicate their thoughts, ideas, and requirements. As described earlier, you can use the parts of the UML that are pertinent to your job and not be left out when other teams do their tasks. Also, by using the UML, you can always go back in the process to update information and include new information that may be discovered or required later.

In this book, we cover the different ways to model during the life cycle of database design and how the different teams can work together to accomplish common goals through modeling. By having all teams working together to understand and define the business, uncover needed changes, prioritize those changes, and model the business, they can all understand the job ahead. This will also help them recognize changes to the requirements as the project proceeds to ensure that each team takes advantage of changes made by others and to successfully use the changes that are made.

The Case Study

In the book, we follow a case study of a fictitious company, EAB Healthcare, Inc., to demonstrate how they began to understand their business systems, how they made changes to those systems to better serve their customers (both internal and external), and how they designed new elements in their database to best serve their purposes.

EAB Healthcare is a fictional provider of physical rehabilitation and nursing care to older adults. While traditional healthcare facilities are suffering financial difficulties and are undergoing industry-wide consolidation, EAB has been tremendously successful. This is primarily due to their innovative rehabilitative therapy programs and their focus on efficient facility operation.

In an effort to remain a leader in their industry through improving their operational excellence, EAB has undertaken a project to computerize the vast volume of paper medical records that all of their nationwide facilities must handle on a daily basis.

The Vision

The business client’s ultimate vision for this project is to have a fully automated, online medical records system that will

- Eliminate the need to manually handle the large volume of paper medical records by providing these records in electronic form
- Fully integrate their records into a medical records database
I Enforce regulatory record-keeping requirements
I Eliminate the manual transcription of information between currently paper-based information sets

The desired outcome is improved patient care through better information management and more effective staff operations.

A Typical Scenario

Using the new system, the staff of EAB will typically access the medical records via touch-sensitive information display panels that will be present in each resident’s room, in all treatment areas, and in all staff offices. The following describes a typical usage scenario.

A nurse enters a resident’s room to assess the resident’s status. Noticing the resident’s trend toward excessive weight loss, the nurse needs to review the patient’s dietary orders in the medical records. She goes to the informational display and swipes her personal access card. Once recognized by the system, she enters her personal identification number (PIN). Security verification allows her to access the system. She enters the resident’s name and the system displays the resident’s records to her. (Depending on the person’s role, the system may grant access as read-only, read/write, read/write/create/destroy, or other appropriate combinations. The system may also make available only certain sections of the medical records, also established by role.) The nurse finds and reviews the information she needs. She then ends her session with the system.

The Project Goals

The ultimate business goals for this project are as follows:

1. To reduce errors and improve the accuracy of the medical records
2. To improve efficiency by reducing the amount of physical paper the workers handle in daily operations
3. To streamline the interaction between the EAB facilities and the myriad external regulatory agencies and other private service providers with whom EAB interacts

Due to their commitment to quality and since there is such intense oversight of the healthcare industry by external government agencies, EAB has committed to a thorough analysis and design of this new system. The process will begin with business modeling, then system analysis and design using the UML and object-oriented techniques.
Our Focus

We use this case study throughout the book to illustrate how such a system may be developed using UML-based techniques. However, we focus primarily on the database design aspects of the project, not on the development of the application software. Thus, while many of the designs in this book may address application issues, we ultimately focus on the database design supporting the EAB Healthcare system.