Topic 2.5: Degrees of Data Abstraction

Another way of classifying data models is by level or degrees of data abstraction. When designing a database, the designer starts with an abstract view of the overall data environment and adds details as the design approaches implementation. The design of a database can be divided into four models with decreasing levels of abstraction.

2.5.1 The Conceptual Model

The conceptual model represents a global and high level view of the database. It is an enterprise-wide representation of data as viewed by high-level managers. Therefore, it is located at the abstraction apex in Figure 2.10. It is the basis for the identification and description of the main data objects, avoiding details. The most widely used conceptual model is the entity relationship (ER) model.

The ER model is illustrated with the help of the entity relationship diagram (ERD), which is, in effect, the basic database blueprint. ERD is used for the graphical representation of the conceptual model.

Example: To illustrate the use of conceptual model, let us examine the data environment of Tiny College. Tiny College’s main objects are its students, professors, courses, classes, and classrooms. These objects are the main entities for which data are collected and stored.
Figure 2.11 displays the Tiny College entities STUDENT, PROFESSOR, COURSE, CLASS, and ROOM as well as some sample STUDENT attributes.

**FIGURE 2.11 TINY COLLEGE ENTITIES**

Given the entities above, we can define and describe the relationships (also called associations or interactions) among those entities. Recall that a relationship may be classified as one-to-one (1:1), one-to-many (1:M), or many-to-many (M:N).

Having identified the entities, we use a conceptual model, graphically represented by an ERD (Figure 2.12), to relate one entity to another. As you examine the ERD in Figure 2.12, note that the relationships are described by the verbs generates, teaches, contains, and is used for. That is, a COURSE generates a CLASS, a PROFESSOR teaches a CLASS, a CLASS contains STUDENT(s), and a ROOM is used for a CLASS.
The conceptual model yields some very important advantages. First, it provides a relatively easily understood bird’s-eye (macro level) view of the data environment. For example, you can get a summary of Tiny College’s data environment by examining the conceptual model presented in Figure 2.12.

Second, the conceptual model is independent of both software and hardware. **Software independence** means that the model does not depend on the DBMS software used to implement the model. **Hardware independence** means that the model does not depend on the hardware used in the implementation of the model. Therefore, changes in either the hardware or the DBMS software will have no effect on the database design at the conceptual level.

**Advantages of Conceptual Model**

- Provides a relatively easily understood macro level view of data environment
- Independent of both software and hardware
  - Does not depend on the DBMS software used to implement the model
  - Does not depend on the hardware used in the implementation of the model
  - Changes in either the hardware or the DBMS software have no effect on the database design at the conceptual level
2.5.2 The Internal Model

Once a specific DBMS has been selected, the internal model adapts the conceptual model to the DBMS. The internal model is the representation of the database as “seen” by the DBMS. In other words, the internal model requires the designer to match the conceptual model’s characteristics and constraints to those of the selected implementation model. Because the internal model depends on the existence of specific database management software, it is said to be software-dependent. Therefore, a change in the DBMS software requires that the internal model be changed to fit the characteristics and requirements of the DBMS.

In the case of Figure 2.12, we implement the internal model by creating the database for Tiny College using the tables PROFESSOR, COURSE, STUDENT, and ROOM. Unfortunately, between STUDENT and CLASS is M:N. The M:N relationship is undesirable, because it produces the data redundancies you learned about in unit 1. In addition, the enrollment of students in classes is an activity that requires its own entity. Therefore, we must create an entity to implement the relationship between CLASS and STUDENT. We will name this entity ENROLL, as shown in the internal model segment in Figure 2.13. The internal model is hardware-independent because it is unaffected by the choice of the computer in which the software is installed. Therefore, a change in storage devices or even a change in operating system will not affect the internal model’s design requirements.
2.5.3 The External Model

The external model, based on the internal model, is the end user's view of the data environment. By end users we mean the people who use the application programs as well as those who designed and implemented them. End users usually operate in an environment in which an application has specific business unit focus. Businesses are generally divided into several business units such as
sales, finance, and marketing. Each business unit subject to specific constraints and requirements, and each one uses a data subset. Therefore, applications programmers working within those business units view their data subsets as separate from, or external to, the internal model from which those subsets were derived.

The applications programmer's working environment thus requires that the modeler subdivide the set of requirements and constraints into functional modules that can be examined within the framework of their external models. For example, the internal model depicted in Figure 2.13 is broken up into two functional modules that yield the two external modules. Each of the external models summarizes the theater of operations for each applications programmer.

Each external model includes the appropriate entities, relationships, processes, and constraints imposed by the business unit. Note that each of the external models in Figure 2.14 represents a subset of the internal model depicted in Figure 2.13. Also note that, although the application views are isolated from other views, each view shares a common entity with the other view. For example, programmers Jim and Anne share the entity CLASS. A good design should consider such relationships between views and provide the programmers with a set of restrictions that govern common entities. The external model for Tiny College may be represented as shown in Figure 2.14.

**Advantages of External Model**

- Use of database subsets makes application program development much simpler
- Facilitates designer’s task by making it easier to identify specific data required to support each business unit’s operations
- Provides feedback about the conceptual model’s adequacy
- Creation of external models helps to ensure security constraints in the database design

The external model is DBMS-dependent and hardware-independent, because it is designed for specific DBMS but does not depend on the hardware used to implement the system. Therefore, the database designer can concentrate on the design without having to address the constraints imposed by the hardware. Consequently, well-designed external schemes do not need to be changed when new physical devices are installed.
2.5.4 The Physical Model

The physical model operates at the lowest level of abstraction, describing the way data are saved on storage media such as disks or tapes. The physical model requires the definition of both the physical storage devices and the (physical) access methods required to reach the data within those storage devices.

Because the physical model requires such precise task assignments, it is both software- and hardware-dependent. The storage structures used are dependent on the software (DBMS, operating system) and on the type of storage devices that the computer can handle. The precision required in the physical model's definition demands that database designers who work at this level have a detailed knowledge of the hardware and software used to implement the database design.
Although the relational model does not require the designer to be concerned about the data’s physical storage characteristics, the implementation of a relational model may require low level fine-tuning for increased performance.

Fine-tuning is especially important when very large databases are installed in a mainframe environment. Yet even such performance fine-tuning at the physical level does not require knowledge of the relational models physical data storage characteristics. In short, the relational model is much more designer friendly than the hierarchical and network models.

Data modeling requirements are a function of different data views (global vs. local) and the level of data abstraction. Characteristics of the four levels of data abstraction are illustrated in Table 2.2.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>DEGREE OF ABSTRACTION</th>
<th>DATA MODEL</th>
<th>FOCUS</th>
<th>INDEPENDENT OF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual</td>
<td>High</td>
<td>Entity</td>
<td>Global</td>
<td>Hardware and software</td>
</tr>
<tr>
<td>External</td>
<td></td>
<td>ER components</td>
<td>Subset</td>
<td>Hardware</td>
</tr>
<tr>
<td>Internal</td>
<td></td>
<td>Relational and others</td>
<td>Global</td>
<td>Hardware</td>
</tr>
<tr>
<td>Physical</td>
<td>Low</td>
<td>Physical storage methods</td>
<td>N/A</td>
<td>Neither hardware nor software</td>
</tr>
</tbody>
</table>

**Concept Check**

What is data abstraction?

How does data abstraction apply to data modeling?

Explain the four levels of data abstraction?