Material about anexo lesson 1

1. Data models, Schemas, and Instances

One fundamental characteristic of the database approach is that it provides some level of **data abstraction**. Data abstraction generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.

A **data model** is a collection of concepts that can be used to describe the structure of a database. By structure of a database we mean the data types, relationships, and constraints that apply to the data.

1.1 Categories of Data Models

High-level or semantic/conceptual data models provide concepts that are close to the way many users perceive data, whereas **low-level or physical data models** provide concepts that describes the details of how data is stored on the computer storage media. Concepts provided by physical data models are generally meant for computer specialists, not for end users.
Between the two extrems is a class of **representational data models**, which provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage.

Conceptual/semantic data models use concepts such as entities, attributes, and relationships. An **entity** represents a real-world object or concept, such as an employee or a project from the miniworls that is described in the database. An **attribute** represents some property of interest that further describes an entity such as the employee's name or salary. A **relationship** among two or more entities represents an association among the entities, for exemple, a works-on relationship between an employee and a project. Chapter 3 presents the **entity-relationship model**, a popular high level conceptual data model.

Representational data models are the models used most frequently in traditional DBMSs. These include the widely used **relational data model**, as well as the **network** and **hierarchical models** – that have been widely used in the past.

SGBD	**Modelo de datos**	**Estructura de datos**
Jerárquico	Jerárquico	segmento (registro), árbol
Red	Red	registro, fichero, set (lista)
Relacional	Relacional	tupla (registro), relación
Orientado a objetos (OO)	Orientado a objetos	constructores de tipos: conjunto, lista, tupla, etc.

Part 3 of the text is devoted to the relational data model, and its constraints, operations and languages. The SQL stantard for relational databases is described in Chapters 6 and 7.

1.2 Schemas, Instances, and Database State

In a data model, it is important to distinguish between the description of the database and the database itself. The description of the database is called the **database schema**, which is specified during database design and is not expected to change frequently.

A display schema is called a **schema diagram**. Figure 2.1 shows a schema diagram for the database shown in Figure 1.2; the diagram displays the structure of each record type but not the actual instances of records.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

We call each component in the schema – such as STUDENT or COURSE – a **schema construct.**

A schema diagram displays only some aspects of a schema, such as the data items. Other aspects are not specified in the schema diagram, for exemple, Figure 2.1 shows neither the data type of each data item nor the relationship among the various files. Many types of constraints are not represented in schema diagrams. A constraint such as *student majoring in computer science mus take CS1310 before the end of their sophomore year is quite difficult to represent diagrammatically.*

The actual data in a database may change quite frequently. For example, the data shown in Figure 1.2 changes every time we add a new student or enter a new grade. The data in the database at a particular moment in time is called **database state** or **snapshot**. It is also called the current set of **occurrences** or **instances** in the database. In a given databas estate, each schema construct has its own current set of instances, for example, the STUDENT construct will contain the set of individual

student entities (records) as its instances. Every time we inserto r delete a record or change the value of a data item in a record, we change one state of the database into another state.

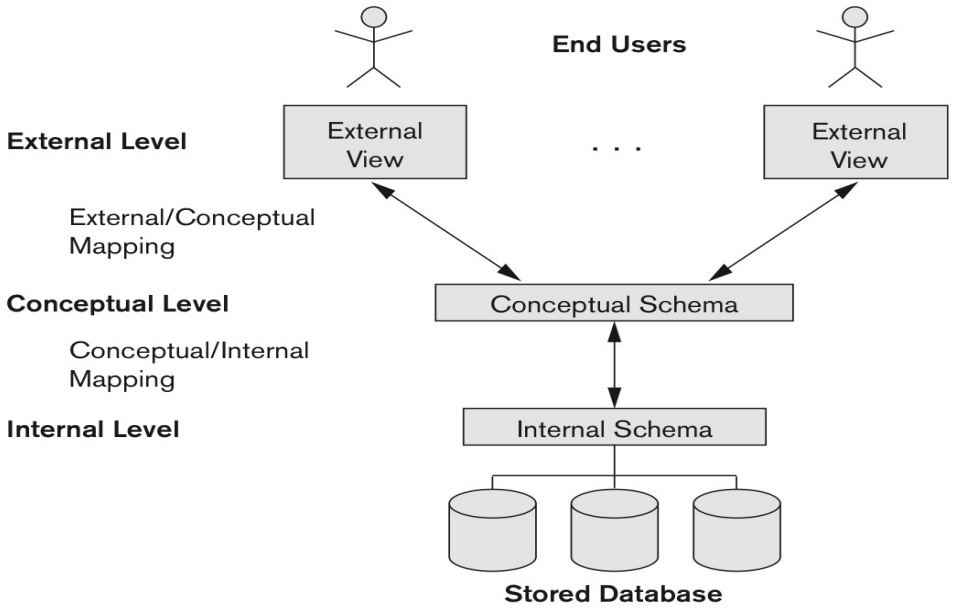
The distinction between database schema and database state is very important. When we **define** a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the *empty state* with no data. We get the *initial state* of the database when the database is first **populated** or **loaded** with the initial data. From then on, every time an update operation is applied to the database, we get another database state. At any point in time, the database has a current state. The DBMS is partly responsable for ensuring that every state of the database is a **valid state** – that is, a state that satisfies the structure and constraints specified in the schema. Hence, specifying a correct schema to the DBMS is extremely important and the schema must be designed with utmost care. The DBMS stores the descriptions of the schema constructs and constraints – also called **meta-data** – in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to. The schema is sometimes called the **intension**, and the database state is called **extension** of the schema.

2. The Three-Schema Architecture

The goal of the three-schema architecture, illustrated in Figure 2.2, is to separate the user applications from the physical database.

To achieve this goal, database management systems allow the definition of the database at three levels of abstraction: physical/internal, logical/conceptual and extern/view. The definition of the database in each of these levels is called a schema.

In this architecture, schemas can be defined at the following three levels:



1. The **internal level** has an **internal schema**, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The **conceptual level** has a **conceptual schema**, which describes of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on decsribing entities, data types, relationships and constraints.
3. The **external** or **view level** includes a number of **external schemas** or user views. Each external schema describes the part of the database that particular user group is interested in and hides the rest of the database from that user group.

The three-level ANSI architecture has an important place in database technology development because ir clearly separates the users' external level, the databases' conceptual level, and the internal storage level for designing a database.

The DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformated to match the user's external view. The process of transforming requests and results between levels are called **mappings.**

3. Data Independence

The three-schema architecture can be used to further explain the concept of data Independence., which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data Independence:

a) **Logical data Independence** is the capacity to change the conceptual schema without having to change external schemas or application programs.
b) **Physical data indepence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schema need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized –for example, by creating additional acces structures—to improve the performance of retrieval or update.