Chapter 2: Database System Concepts and Architecture

Introduction to Data Models

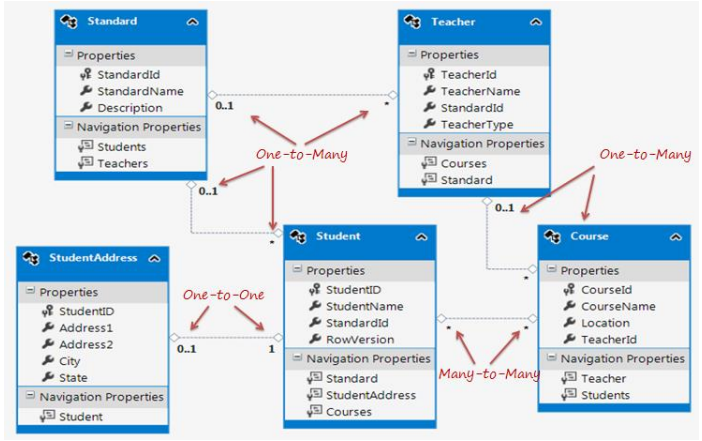
- Consider:
  - What are the steps to construct a house
    - What do need in the building?
    - Put it in the form design on the paper
    - Gather all the materials to construct the building

Introduction to Data Models (cont.)

- Consider again
  - How to construct a database for a particular requirement?
    - What are components of requirement?
    - How it can be structured in the database

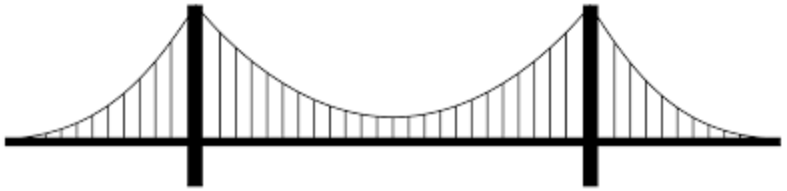
Data modeling

- **Planning the structure** of database is called **data models**
  - Tables, columns, mappings, storage



Introduction to Data Models (cont.)

Real world requirement



Data model

Database Design

Highlight drawback

Data Models

- **Data Model**:
  - A set of *concepts* to describe
    - The *structure* of a database
      - Data types, relationships
      - The **operations** for manipulating these structures
      - The **constraints** that the database should obey

Data Model Concepts

- **Entity**
  - A class of real world objects having common attributes
- **Attribute**
  - A characteristic or property of an entity
- **Relationship**
  - An association between two or more entities

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A

Data Modeling Exercise

- Consider:
  - What is the "**entity**"?
  - What are the "**attributes**" of the entity

Data Modeling Exercise (cont.)

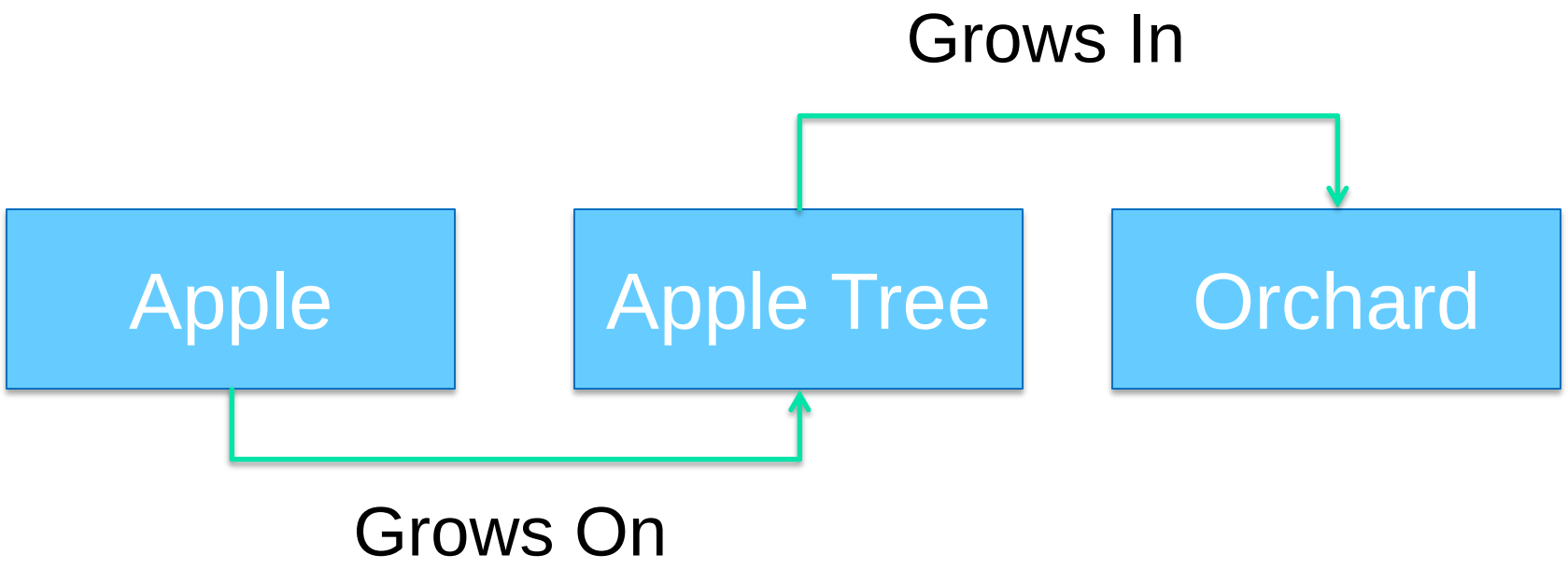
- What is the **entity**?
- What are the **attributes**?

Data Modeling Exercise (cont.)

- What is the **entity**?
- What are the **attributes**?

Data Modeling Exercise (cont.)

- What are the **relationships**?



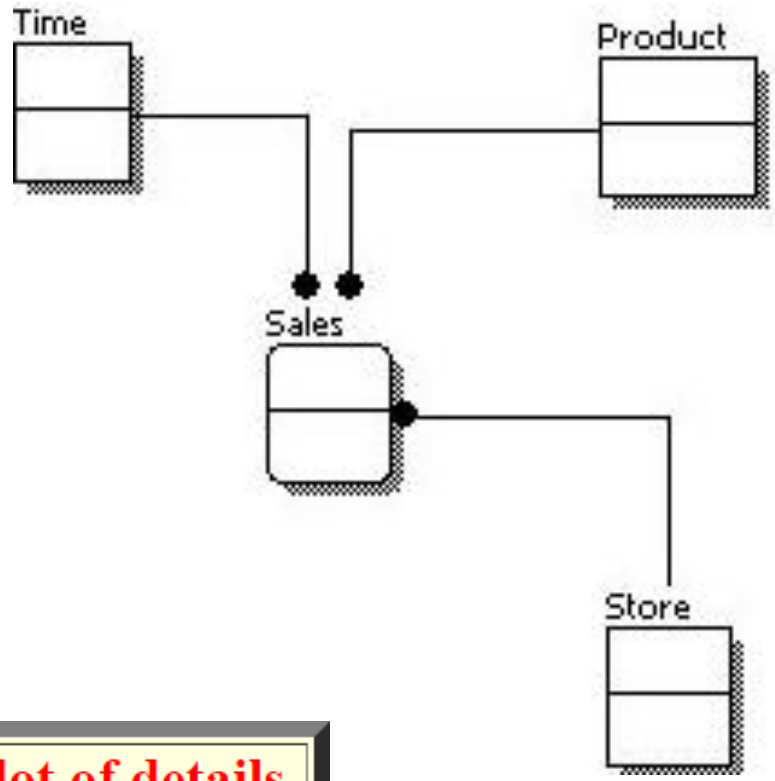
Categories of Data Models

- Goal – make sure all data objects required by a database are completely and accurately represented

- Three categories (stages)
  - **Conceptual data** model
  - **Implementation data** model
  - **Physical data** model

Conceptual Data Models

- **Conceptual** (high-level, semantic) data models:
  - Provide concepts that are **close** to the way many users perceive data.
    - Includes the important entities and the relationships among them
    - No attribute is specified
    - No primary key is specified

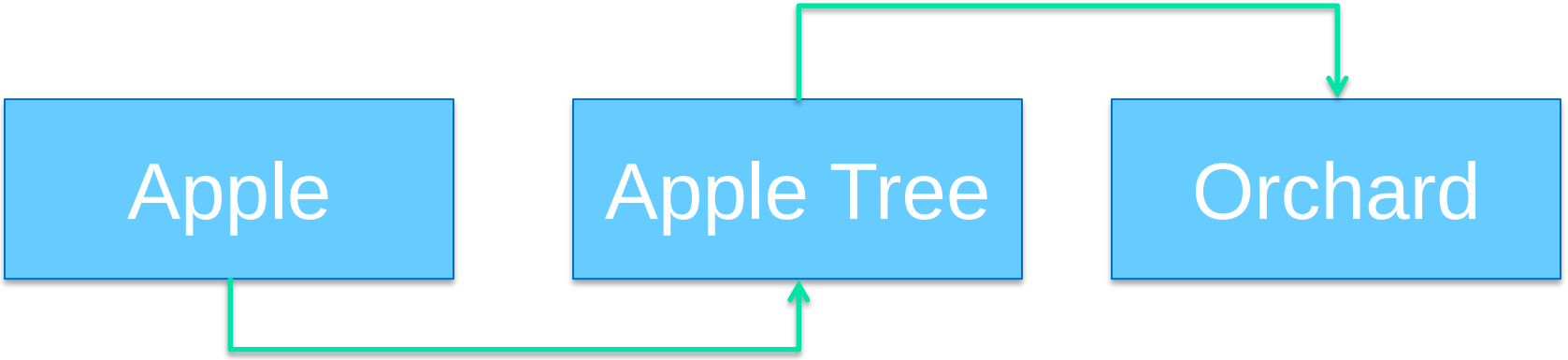


- This **type** of **model** **omits a lot of details.**

Who do use this model?

Apple/Tree/Orchard Conceptual Model

Grows In



Grows On

Implementation Data Model

- **Implementation** data models:

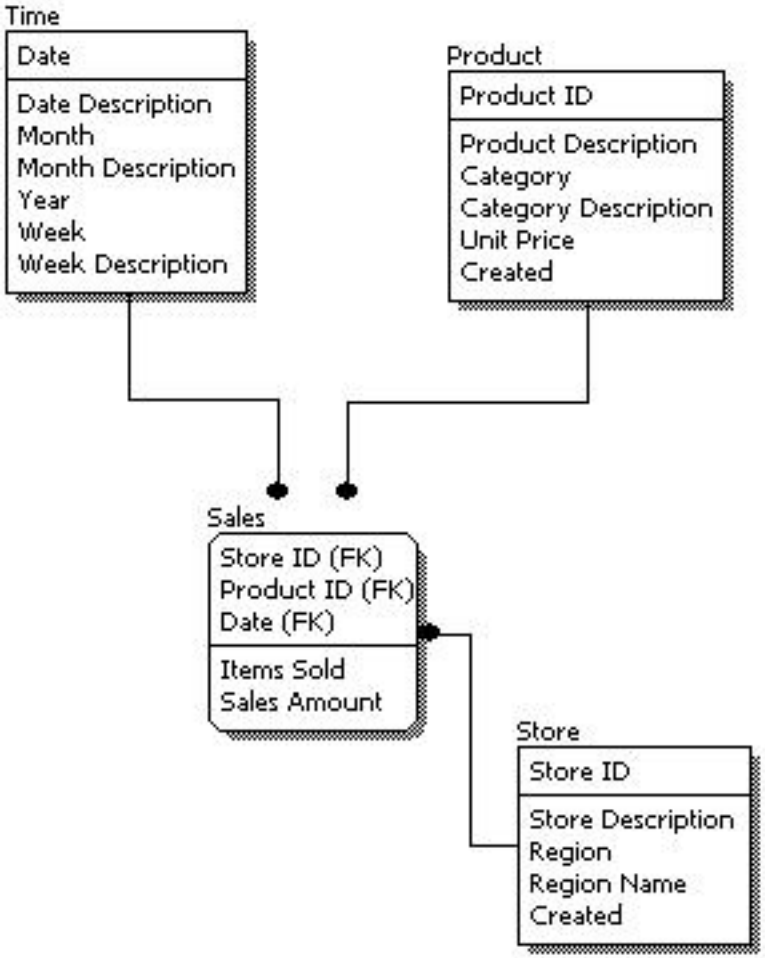
  - This **type** of **model** has ***more*** **details** on ***what*** **data** is **stored** in the **database** and ***how*** **(what files)** the **data** is **stored**
  - The model contains enough detail to let user **see** the **structure** of the **data** and **how** the **data** is **stored** inside the **database**.

  - Defines
    - Entities and their attributes
    - Relationships
    - Constraints
    - Primary key, foreign key

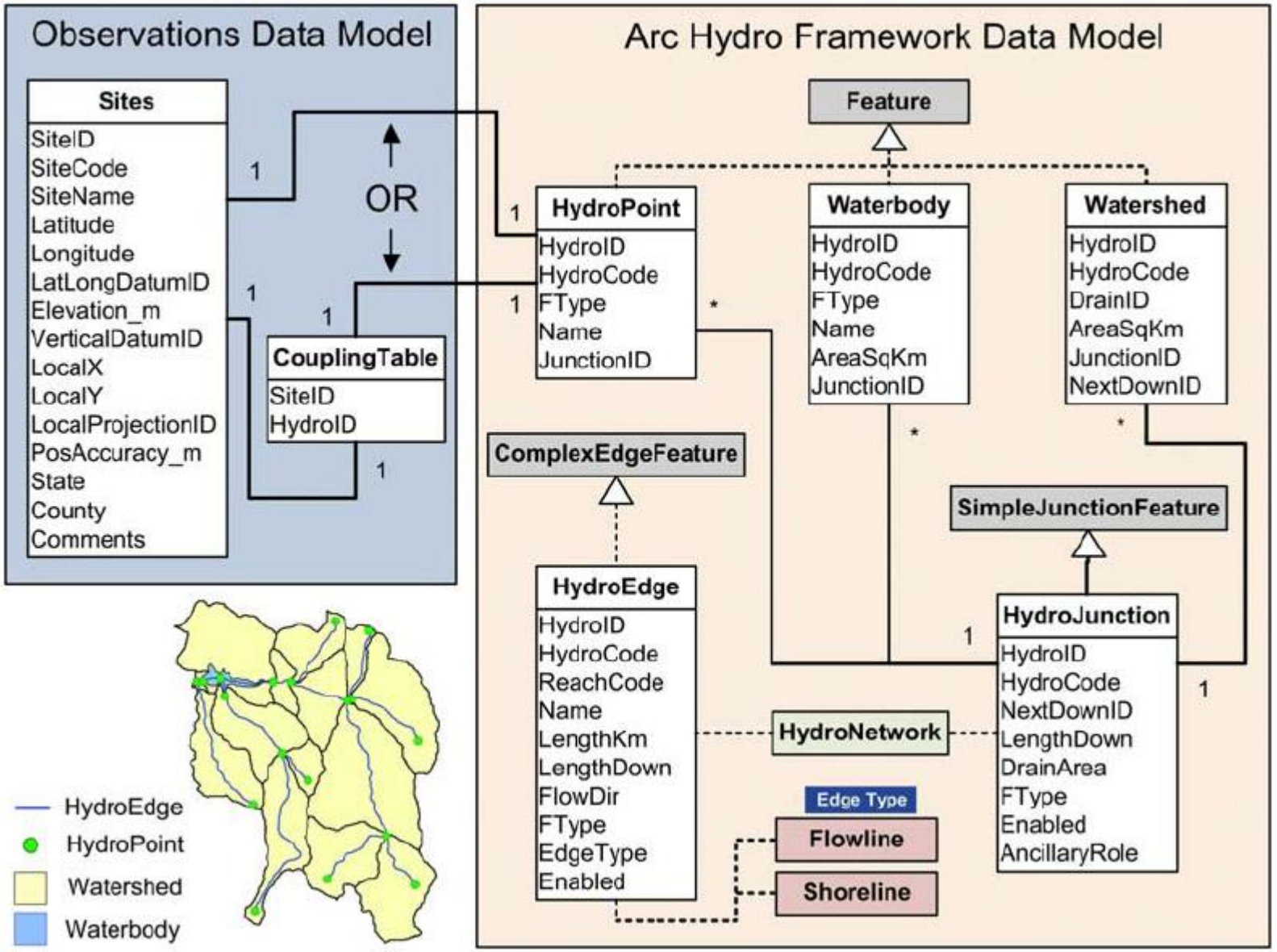
Who do use this model?

Implementation Data Model (cont.)

- **Implementation** data models:



Example: ODM Implementation Data Model ER Diagram



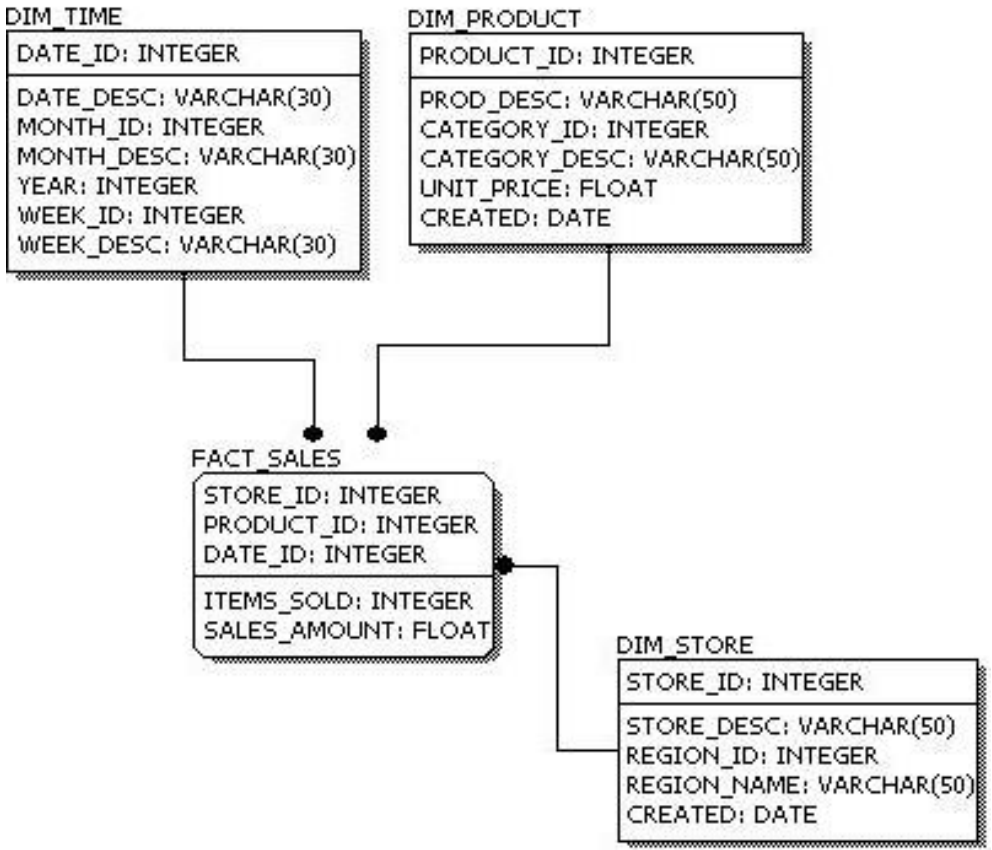
Physical Data Model

- **Physical** data models:
  - This type of model contains minute (very precise) detail on how the data is store inside the **computer**
  - Example of some of the details:

- **File names**
- **Indices** on the **files**
- **Statistics** on the **data** in files (e.g.: number of records, record length, etc)
- **Which *disk*** will be used to **store** the data

Physical Data Model (cont.)

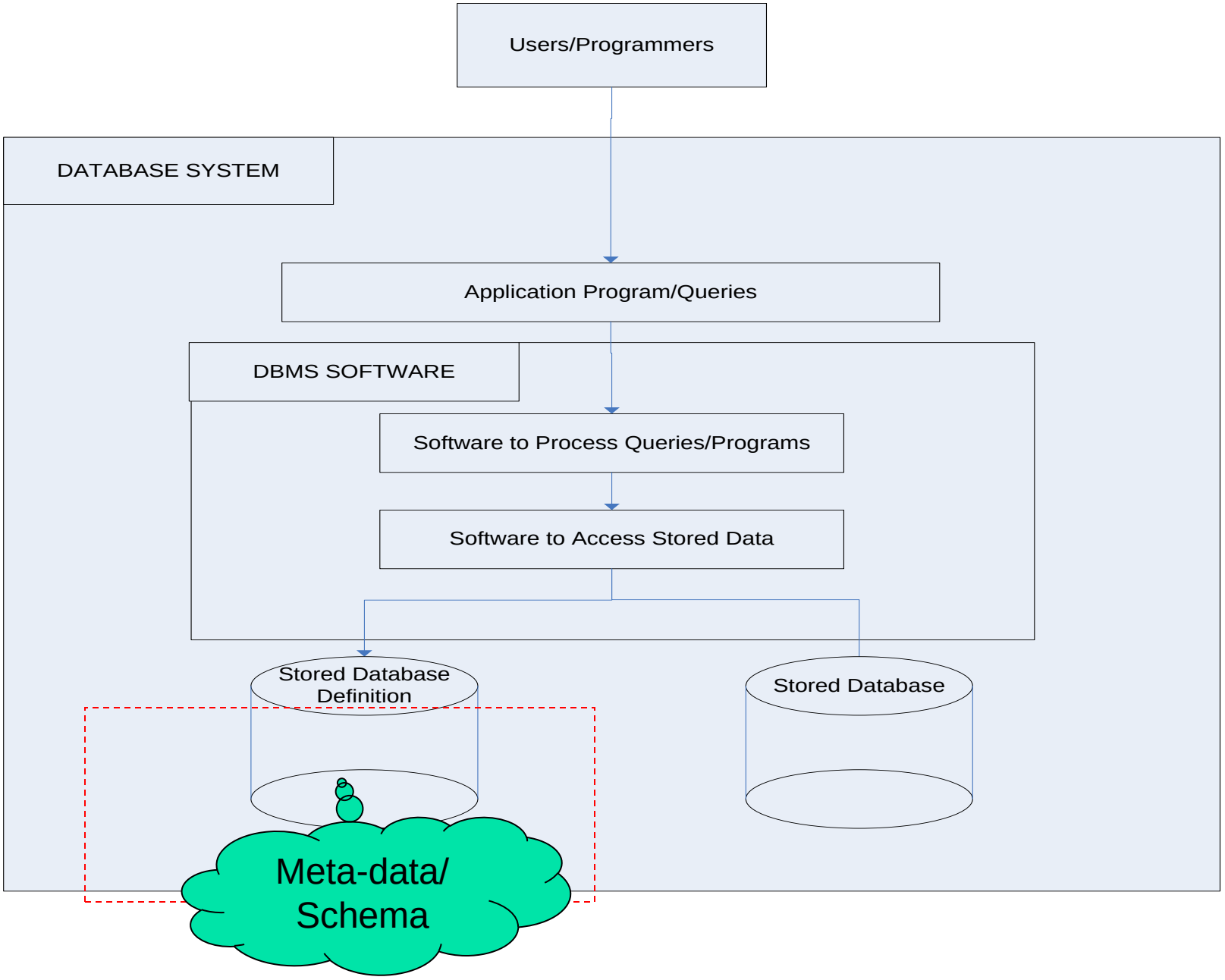
- **Physical** data models:
  - Provide concepts that describe details of how data is stored in the computer



Database Schema

- Description of the dataset v.s. Dataset
  - Similar to Types v.s. Variables in programming languages

> - **Database schema** = a **description** of a **database**
>
> (It's **actually** a **description** of the **data** stored inside a **database**)


Users/Programmers
DATABASE SYSTEM
Application Program/Queries
DBMS SOFTWARE
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition
Stored Database
Meta-data/ Schema

A Running Example

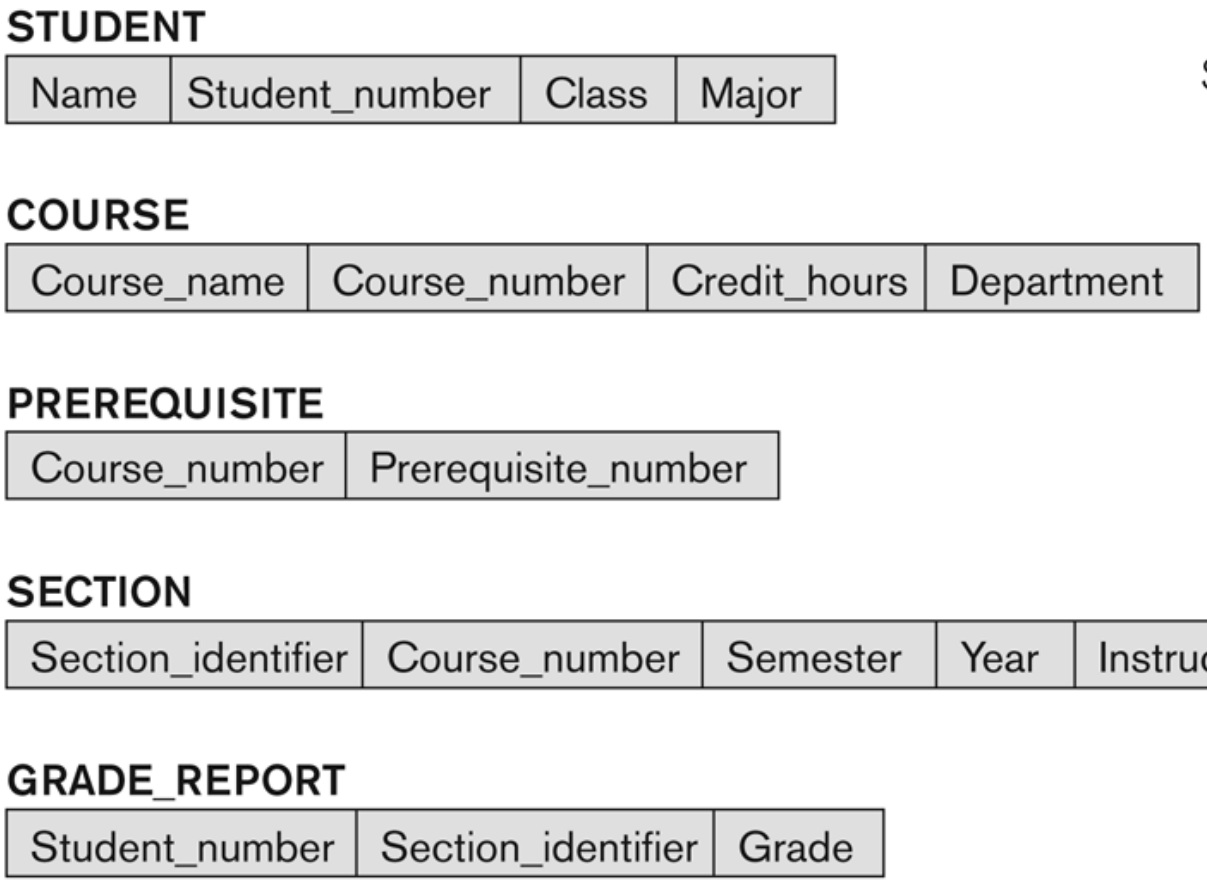


Figure 2.1
Schema diagram for the database in Figure 1.2.

Database State

- **Database State:**
  - The actual content of the database at a particular point of time.
    - Analogous to the value of a variable
  - Also called **database instance**

A Running Example

At 5:00PM, EMPLOYEE and DEPARTMENT tables

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEPT_ID
100	Joseph	Clinton Town	10
101	Rose	Fraser Town	20
102	Mathew	Lakeside Village	10
103	Stewart	Troy	30

DEPARTMENT

DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

At 5:01 PM, EMPLOYEE and DEPARTMENT tables – New records is added to EMPLOYEE table

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEPT_ID
100	Joseph	Clinton Town	10
101	Rose	Fraser Town	20
102	Mathew	Lakeside Village	10
103	Stewart	Troy	30
104	William	Holland	30

DEPARTMENT

DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

At 5:02 PM, EMPLOYEE and DEPARTMENT tables – EMP_ID 102 has been deleted from EMPLOYEE table

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEPT_ID
100	Joseph	Clinton Town	10
101	Rose	Fraser Town	20
~~102~~	~~Mathew~~	~~Lakeside Village~~	~~10~~
103	Stewart	Troy	30
104	William	Holland	30

DEPARTMENT

DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

Database Schema vs. Database State

- Distinction
  - The ***database schema*** changes very infrequently.
  - The ***database state*** changes every time the database is updated.

- **Schema** is also called **Intension**.

- **State** is also called **Extension**.

Why Introduce Database Schema

- A database system is used by users from very diverse backgrounds:

  - from **Computer Science nerds**
  - to **Managers** who have no idea what a "byte" is....

- As such, a database system will use *different levels* of schemas to describe the data stored inside the database

Three-Schema Architecture

- Proposed to support DBMS characteristics of:
  - Use of a **catalog** to store the schema
  - **Program-data independence.**
  - Support of **multiple views** of the data.

- Defines DBMS schemas at ***three*** levels:
  - **External schemas**
  - **Conceptual schema**
  - **Internal schema**

External Schema

- **External schemas**
  - Describes how a class of users views/perceives their data
  - Used by non-experts (users of the DBMS) to specify their **needs**

    - Different users will have different needs and have different perceptions of the data

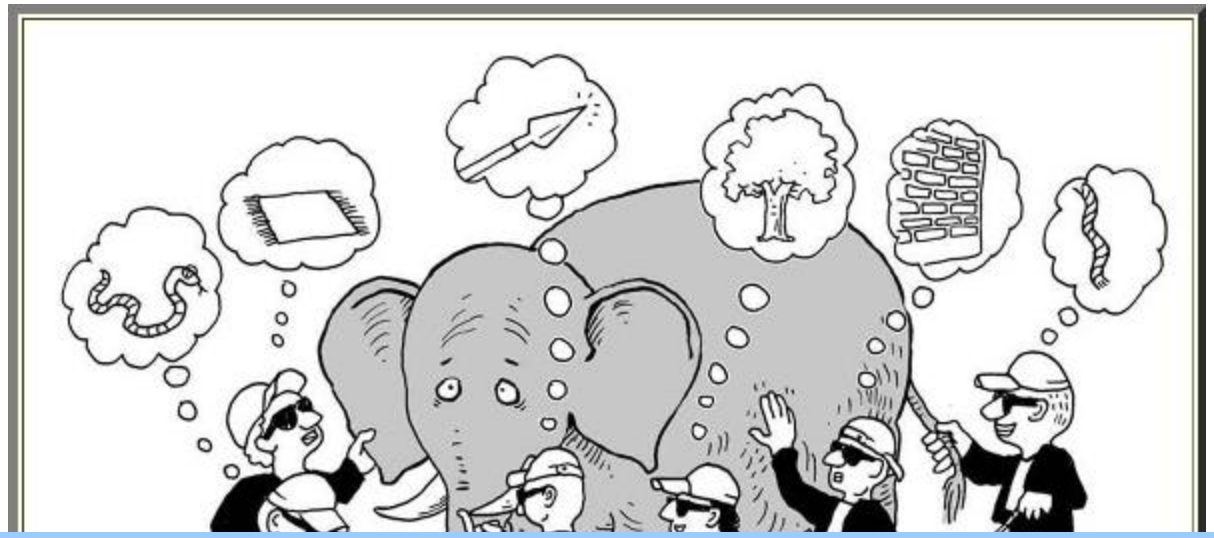
      E.g., a CEO and a clerk of the same company will have different needs and perceptions of the same company

  - There are multiple **external schemas** in one database schema

The **external schema** is an example of **conceptual data model**

A Running Example

- Analogy: Blind men examining an elephant…

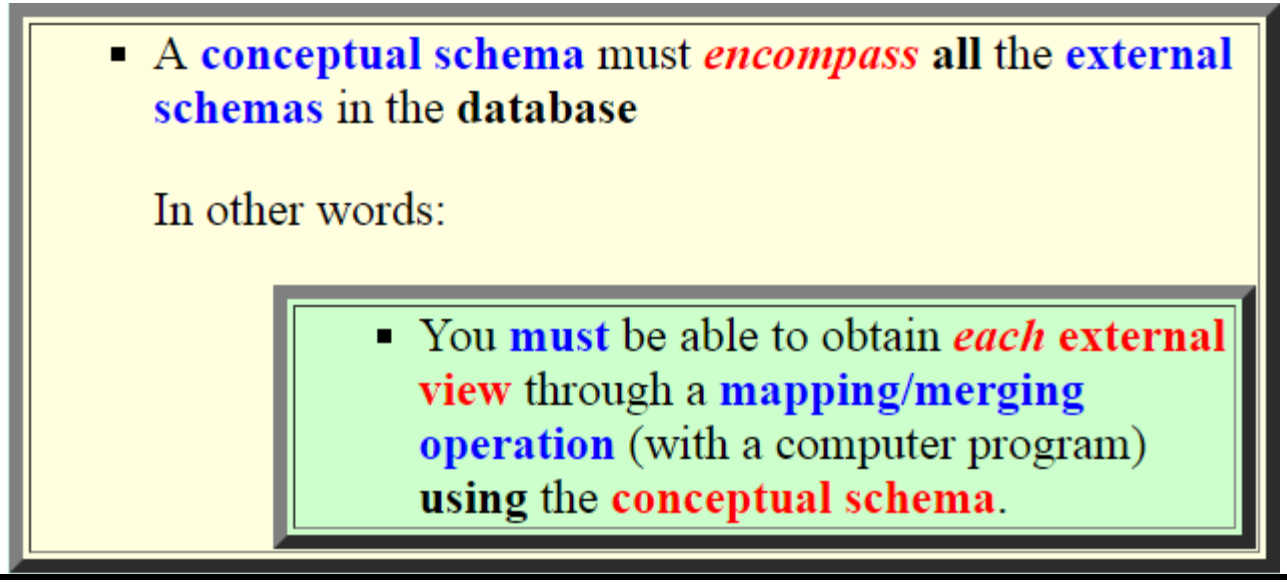


Different users **perceive** the data stored differently

One man said it's like a tree trunk (he felt the feet)
Another said it's like a snake (he felt the tail)
And so on…

Conceptual Schema

- **Conceptual schema**
  - Describe what data is stored in database and relationship among the data
  - The way perceived by the DBA & programmers
  - There is only one **Conceptual Schema** for each database



The **conceptual schema** is an example of **implementation data model**

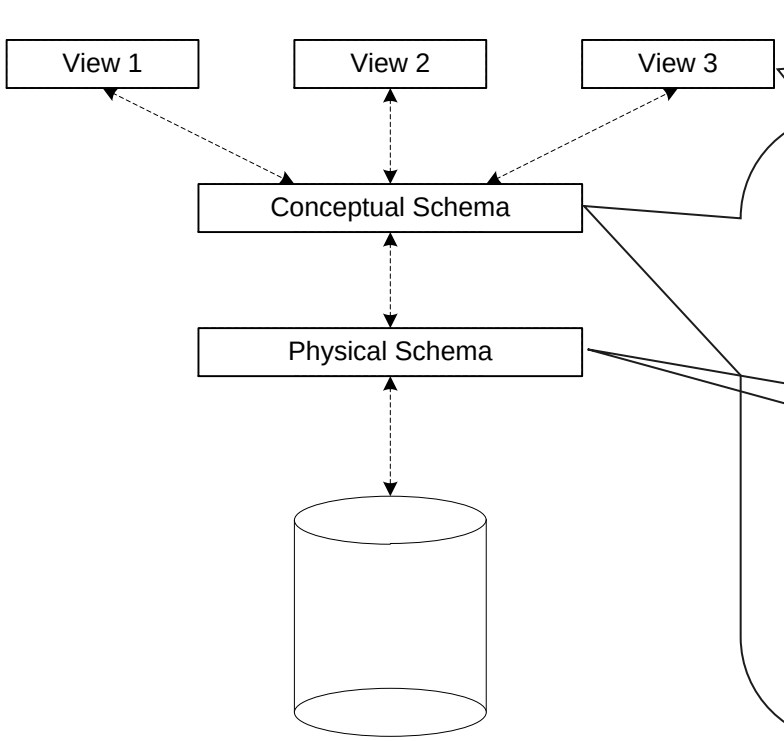
Three-Schema Architecture (cont)

- **Internal schema**
  - The **Internal schema** describes **every detail** on how the data files of the database are stored inside the **computer**
  - There is only one **Physical schema** for each database

  - **Format** of each file
  - **Identifying fields** (e.g., **SSN**, **Student number**) of each file
  - **Indexes** (used to **access record** with **specific values** ***quickly***)
  - And so on.

The **Internal schema** is an example of **Physical data model**

Three-Schema Architecture (cont)



Grade_Report(StudentNumber: **01**, SectionID: **11285**, Grade: **A**)

Prerequisite(CourseNumber: **4710**, PrerequisiteNumber: **3410**)

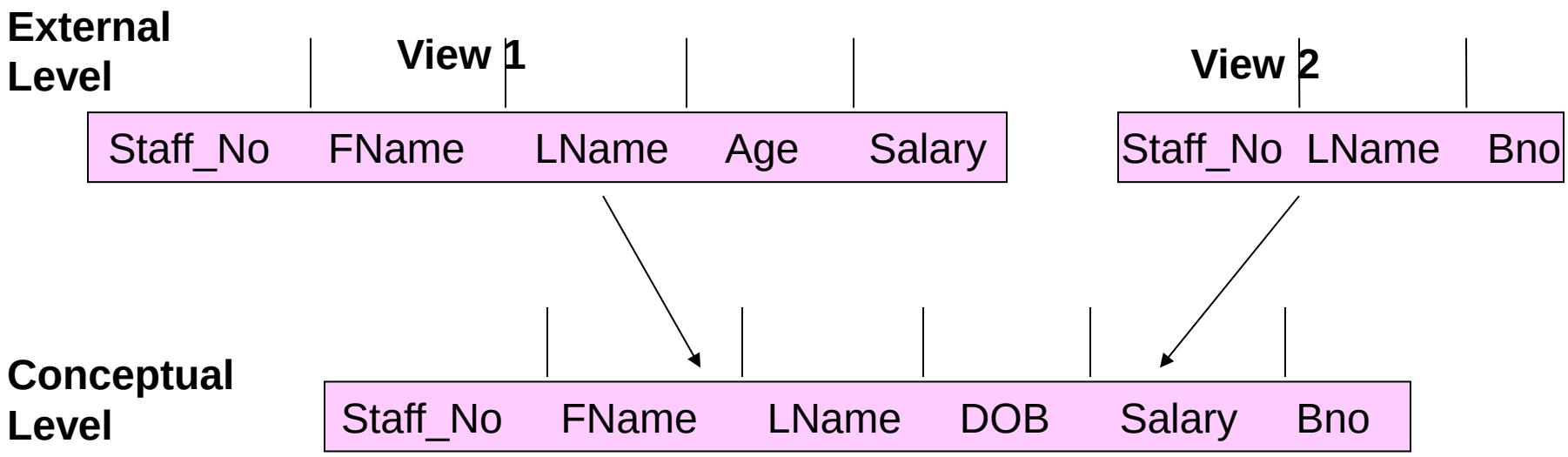
Student(Name:string, StudentNumber:integer, Class:integer, Major:string)

Course(CourseName:string, CourseNumber:string, CreditHours:integer, Department:string)

Section(SectionID:integer, CourseNumber:string, Semester:string, Year:integer, Instructor:String)

- Block size, unordered files:
- Index on second column of ***Student***:

Three-Schema Architecture (cont)

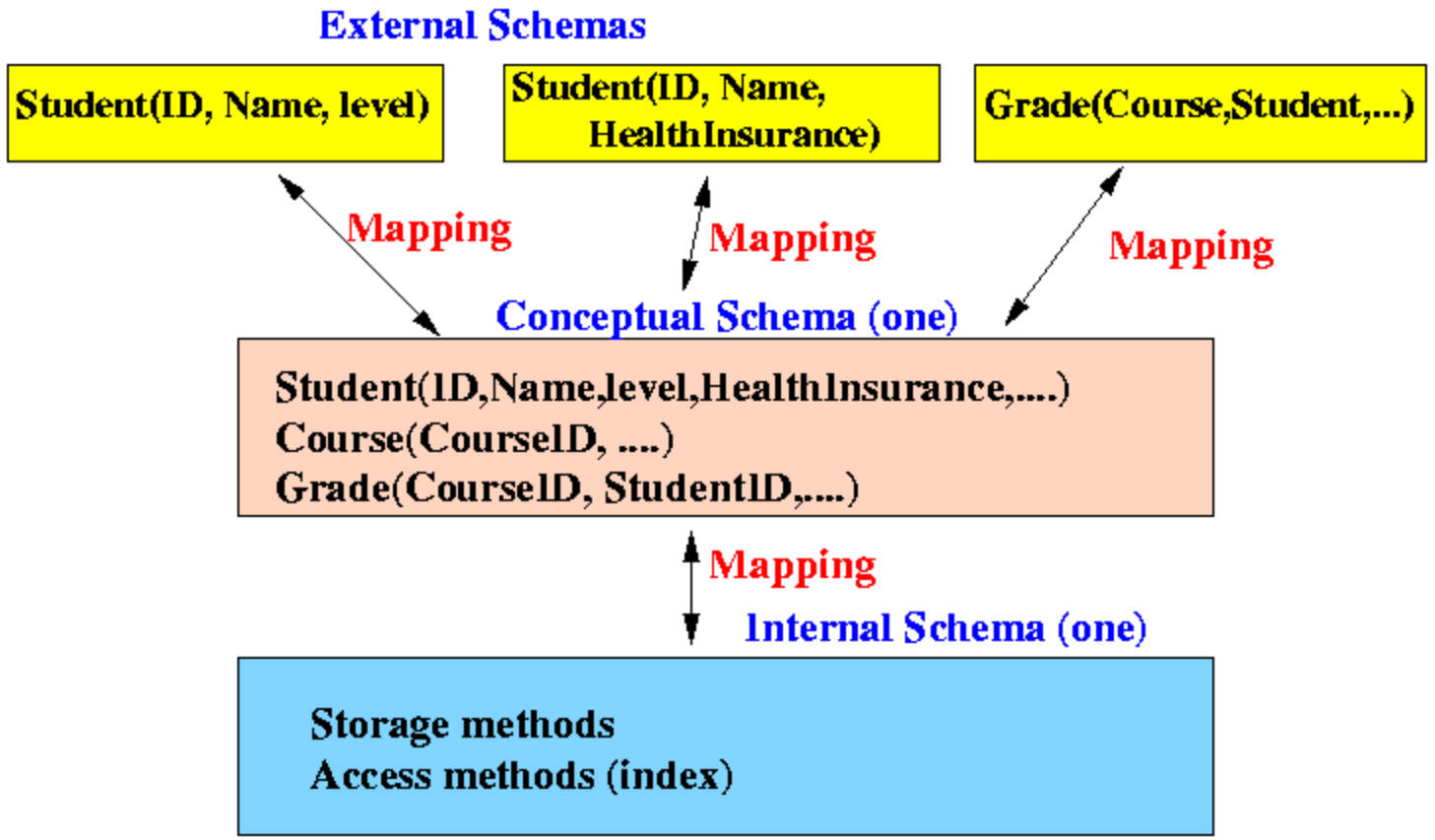


Internal Level

```
Struct STAFF {
        int Staff_No;
        char FName [15];
        char LName [15];
        date DOB;
        float Salary;
        int Bno;
        struct STAFF * next;
};
```

Three-Schema Architecture (cont.)

- Ma
  be
  - h the
  - he



Data Independence

- Logical Data Independence
- Physical Data Independence

Why Called Logical Data Independency

- Question:
  - Suppose we need to **make a change** to the **conceptual schema**

> - **Can** we still obtain the same **external schema** ?

  - In other words

> - ***Can*** we still **present the data** in the **same format** to the user ?

- The answer is clearly: **Yes**!

But Why?

Why Called Logical Data Independency (cont.)

- Reason:
  - **External schema(s)** is obtained from **data stored** in the **conceptual schema** by running some program to **merge** the data files
  - If we need to change **the conceptual schema** (how we store the data), we **can** obtain the **same external schema(s)** by:

    - Using a ***different*** **merging program** !!!

So we need to **update** the **mapping operation** if we change the **conceptual schema**

A Running Example

- Previously, we used the following **mapping operation** to obtain the desired output format:

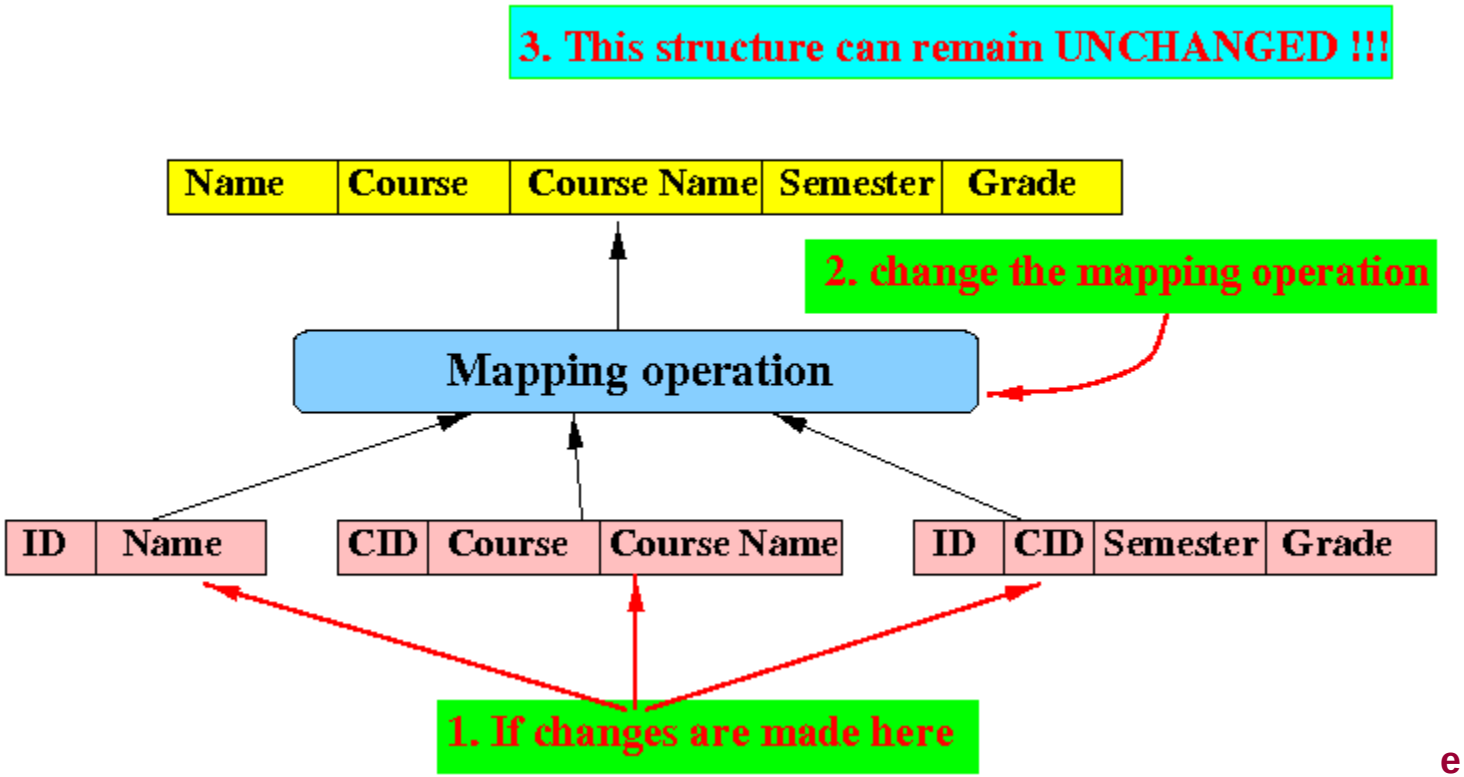
```
create view my_report(stud_name, course_number, course_name, semester, grade)
as
(
   select name, cnum, cname, sem, grade
   from  student, course, grade_report
   where  id = s_id
     and cid = c_id
);
```

- When the **conceptual scheme** changes (i.e., the structure of one ore more of the tables *student*, *course* and *grade_report* changes), we must:

  - **update** the above **SQL statement** to **re-map** the new **conceptual scheme** onto the ***old*** **view** !!!

A Running Example (cont.)

- To accommodate the change in the **conceptual schema**, all we need to do is to change the **mapping operation** between the **external schema** and the **conceptual schema:**



Why is Such a Big Deal

- The end user may have their *own* non-database applications that perform *further processing* (One such application is Excel !!!)

  - The **data analysis tools (programs)** used by the **end user** ***requires*** a ***specific*** **data format**

    (Or else, the **end user** **cannot** import the **data** into their tool ofr **further analysis** !!!)

- The **Logical Data Independence** technique can maintain the end user data file structure in the same format

It will make end users very happy customers !

Summary of Logical Data Independency

- The ability to keep presenting the user with the same format **even though** the way the actual data is stored has changed, is known as:

Logical Data Independency

- The term logical refers to the fact that:

  - How you ***view*** the **data** **"logically (conceptually)"** (= **use** the data), is **independent** from the way the data is **stored**

Physical Data Independency

- Make sure you **understand** what **Physical Data *Dependence*** is before we move on to solve this problem
- **Physical Data Dependency** is **caused** by the fact that:

  - the ***structure (format)* of the data** is **embedded *inside*** the **computer program** itself.

Physical Data Independency (cont.)

- Example:
  - Look inside the file access program, you see this definition of the format of the data:

```
class StudentRecord
{
   String ID;
   String name;
   String level;
}
```

With program statements that are AWARE of the structure:

```
x.ID = myFile.ReadString(10);
x.name = myFile.ReadString(30);
x.level = myFile.ReadString(4);
```

Achieving Physical Data Independency

- **Step 1**: create a "**meta data**" file that contains the *description* of the **structure/format** of the **data file**
  - The **content** of the **meta data** file is fixed:

```
+-----------+-----------+-----------+
| fieldName | fieldType | fieldSize |
+-----------+-----------+-----------+

fieldName = the name of a field (This is a String)

fieldType = the data type of the filed
            "C" = Characters (String)
            "I" = Integer
            "F" = Floating point number or double

fieldSize = the number of bytes used to store the value
```

Achieving Physical Data Independency (cont.)

- **Step 1**: create a "**meta data**" file that contains the *description* of the **structure/format** of the **data file**
  - The **content** of the **meta data** file is fixed: **Example**

- In order to **describe** the ***structure/format*** of the **student data file** with the following **format**:

ID	Name	level
10 chars	30 chars	4 chars

we use a **meta data file** with the following entries:

fieldName	fieldType	fieldSize	
ID	C	10	<---- entry 1
name	C	30	<---- entry 2
level	C	4	<---- entry 3

Achieving Physical Data Independency (cont.)

- **Step 2**: a **program** that achieve **physical data independence** is written as follows

  - The **program** must ***first*** read the ***meta data*** **file** and process the **structure/format description**

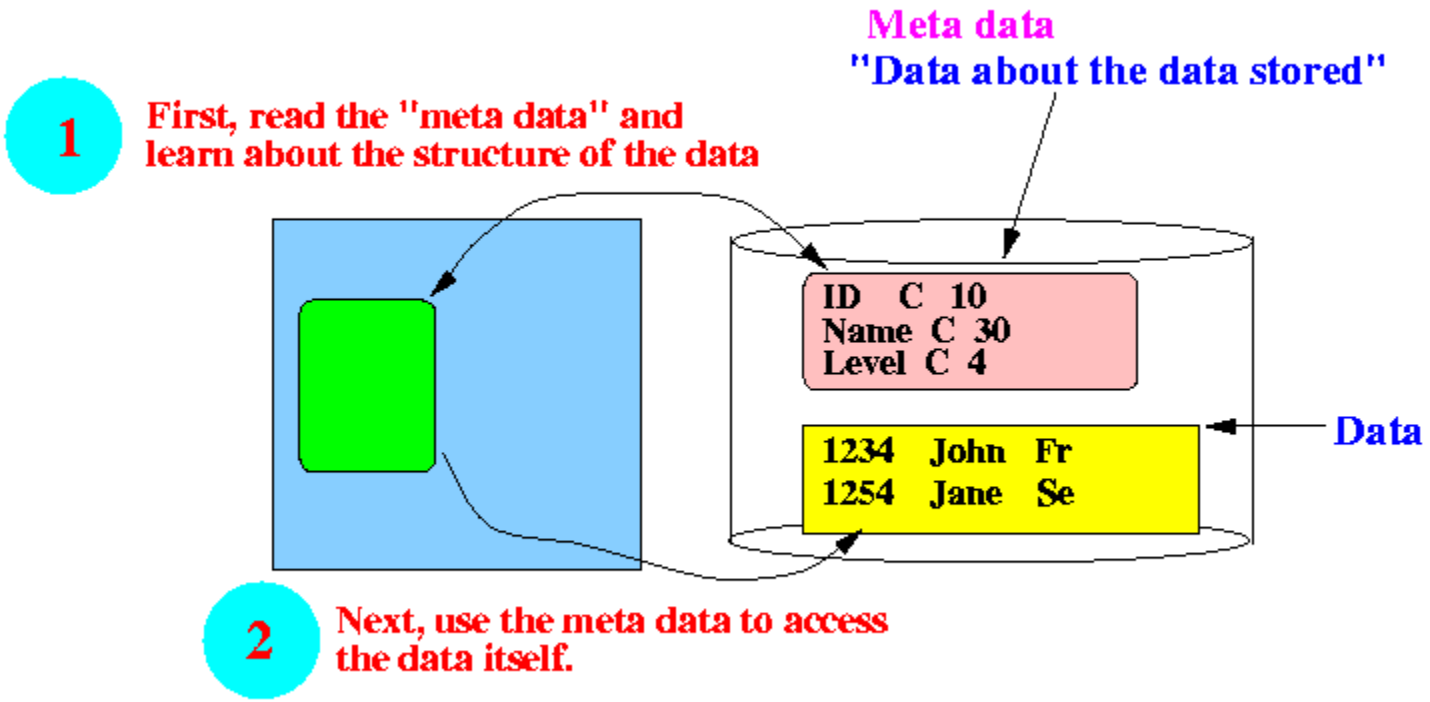
    I.e.:

    - The **program** must first ***learn*** **about** the **structure of the data file** that it will access !!!

  - **Armed** with the ***information*** **on the structure**, the program **accesses** the **actual data file** by reading the **fields** in the **format and size (# bytes)** given in the ***meta data*** **file**

Achieving Physical Data Independency (cont.)

- Schematically:



Summary of Physical Data Independency

- The ability to keep presenting the user with the same format **even though** the way the actual data is stored has changed, is known as: