Data Models

- A **data model** is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
  - A data model is a collection of high-level data description constructs that hide many low-level storage details. A DBMS allows a user to define the data to be stored in terms of a data model.

- A **schema** is a description of a particular collection of data, using a given data model.

- The **relational model of data** is the most widely used model today.
  - Main concept: **relation**, basically a table with rows and columns.
  - Every relation has a **schema**, which describes the columns, or fields.

The data models can be classified into **four different categories**:

1. **Relational Model:** The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as relations.

A description of data in terms of a data model is called a schema. In the relational model, the schema for a relation specifies its name, the name of each field (or attribute or column), and the type of each field. As an example, student information in a university database may be stored in a relation with the following schema:

Students(sid: string, name: string, login: string, age: integer, gpa: real)

The overall design of the database is called the database schema.

sid	*name*	*login*	*age*	*gpa*
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

An Instance of the Students Relation

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database.

Instance: the actual content of the database at a particular point in time

2. **Entity Relationship data model** (mainly for database design):

3. **Object based data models** (Object oriented and Object relational)
   Object-oriented programming (Java, C++, and C#)

4. **Semi structured data model** (XML Extensible Markup Language)

Data Independence

One of the most important benefits of using a DBMS

Applications insulated from how data is structured and stored.
Hiding the physical or low-level details to the external users

DBMS is made to provide data independence.
Without knowing the physical details, we must be able to access the data.

To provide data independence, there should be at least two levels of abstraction.

Levels of Abstraction

Since many database-system users are not computer trained, developers hide the complexity from users through several levels of data abstraction, to simplify users' interactions with the system:

1. Conceptual schema

Conceptual schema (sometimes called the logical schema) describes the stored data in terms of the data model of the DBMS. It describes **what data are stored** in the database, and **what relationships** exist among those data. In a relational DBMS, the conceptual schema describes **all relations that are stored in the database**.

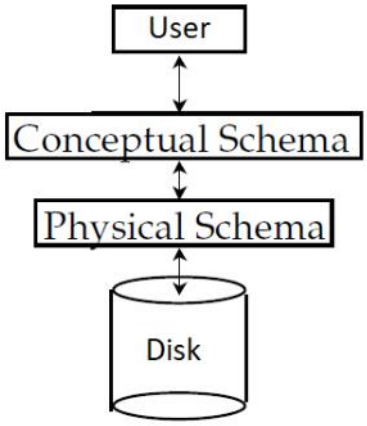
The conceptual schema **hides the details of physical storage structures** and concentrates on describing entities, data types, relationships, user operations, and constraints.

Hiding the physical or low-level details to the external users

Example: University Database

- Conceptual schema:
  Students(sid: string, name: string, login: string, age: integer, gpa:real)
  Courses(cid: string, cname:string, credits:integer)
  Enrolled(sid:string, cid:string, grade:string)

It knows that we have Students, Courses, Enrolled tables in the database.



Middle level or logical level

Low level

Database

2. Physical Schema or Internal Schema

The lowest level of abstraction describes **how the data are actually stored**.

The physical schema describes the complete details of data storage and access paths for the database. Essentially, the physical schema summarizes how the relations described in the conceptual schema are actually stored on secondary storage devices such as disks and tapes.

Metadata - Data about data
Physical Schema is metadata, so it knows that how data is physically stored in the DB (File format, record format, indexing)

- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.

It has all the physical details.

Data independence can be achieved by using the above two levels of abstraction.
We need a third level to provide different views.

3. External Schema or Views

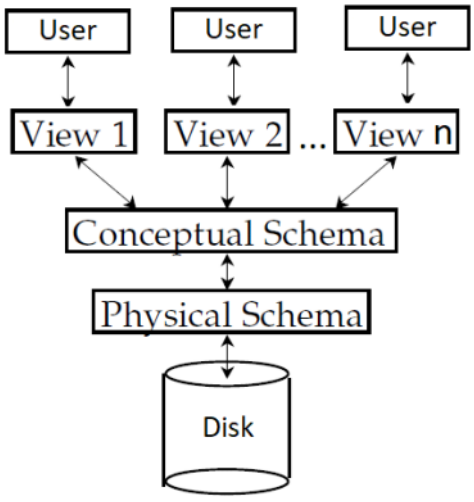
External schemas allow data access to be customized (and authorized) at the level of

individual users or groups of users.

Each external schema describes the **part of the database** that a particular user group is interested in and hides the rest of the database from that user group.

Many **views**, single **conceptual (logical) schema** and **physical schema**.

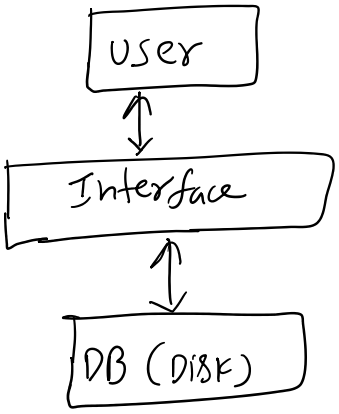
- Views describe how users see the data.
- Conceptual schema defines logical structure
- Physical schema describes the files and indexes used.



Some part of conceptual details

For example, Students can view fewer details while the instructor can view more details, HoD can view even more details. There may be different different views possible as required.

DBMS is an interface in between the user and database.



DDL and DML

Schemas are defined using DDL; data is modified/queried using DML.

A data definition language (DDL) is used to define the external and conceptual schemas.

A database system provides a data-definition language (DDL) to specify the database schema and a data-manipulation language (DML) to express database queries and updates.

DML is a Language for accessing and updating the data organized by the appropriate data model. DML also known as query language.

There are basically two types of data-manipulation language

- Procedural DML --require a user to specify what data are needed and how to get those data.
- Declarative DML or Non-procedural DMLs --require a user to specify what data are needed without specifying how to get those data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs.

In practice, the data-definition and data-manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the SQL language.

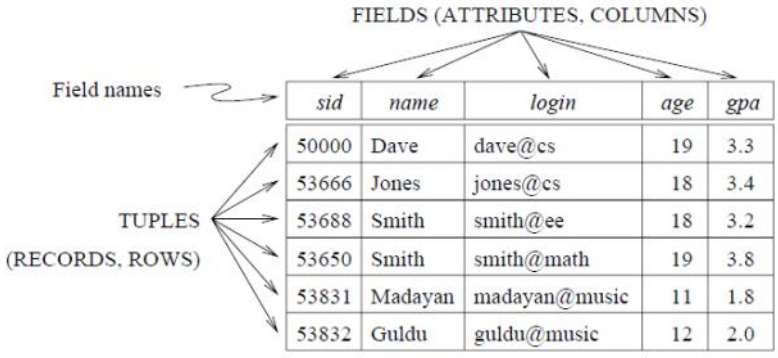
Relational Model

Codd proposed the relational data model in 1970.

All the data is stored in various tables.

Relation = Table (Table: Collection of Rows and Columns)

A database is a collection of one or more relations, where each relation is a table with rows and columns.



sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure An Instance $S1$ of the Students Relation

Relation: made up of 2 parts:

- Instance : A table, with rows and columns (the actual content of the database at a particular point in time).
  #Rows = cardinality, #fields = degree / arity.
- Schema : specifies name of relation, plus name and type of each column.
  E.G.: Students(sid: string, name: string, login: string, age: integer, gpa: real).

Cardinality = 6, degree = 5, all rows distinct

Can think of a relation as a set of rows or tuples (i.e., all rows are distinct).

No two tuples of a relation should be the same.

Relations are Unordered: Order of tuples is irrelevant (tuples may be stored in an arbitrary order)

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53666	Jones	jones@cs	18	3.4
50000	Dave	dave@cs	19	3.3

Figure An Alternative Representation of Instance $S1$ of Students

If the fields are named, as in instances, the order of fields does not matter either.

Attribute Types

A relation schema specifies the domain of each field or column in the relation instance.

The set of allowed values for each attribute is called the domain of the attribute.

These domain constraints in the schema specify an important condition that we want each instance of the relation to satisfy: The values that appear in a column must be drawn from the domain associated with that column. Thus, the domain of a field is essentially the type of that field, in programming language terms, and restricts the values that can appear in the field.

Attribute values are (normally) required to be **atomic** ; that is, indivisible

- E.g. the value of an attribute can be an account number, but cannot be a set of account numbers

The special value null is a member of every domain

The null value is a special value that signifies that the value is **unknown or does not exist**.

A comparison of two NULL values leads to ambiguities—if both Customer A and B have NULL addresses, it does not mean they have the same address.
