Course 1. Introduction in databases

Database?

A **database** is a large collection of related data items stored for record-keeping and analysis that exist over a long period of time one models the real-world aspect through a Data Model.

Simply put, <u>databases are used to store and manage data</u>. Databases are probably one of the most common uses of computers and are available on just about every type of computer.

We usually deal with many different databases everyday without realizing it. Anytime we lookup a phone number in directory, do a transaction at the bank, pay for something with a credit card, or purchase something on the Web, you're using a database.

Databases are not specific to computers. Examples of non-computerized databases abound: phone book, dictionaries, almanacs, etc. (although in actuality these are examples of printed reports generated from databases).

Data Model

A **data model** is a collection of concepts for describing data. These concepts are used to define data's *structure*, *semantics*, *consistency constraints* or its relations with other data.

A **schema** is a description of structure of a collection of data items, using a given data model. Schema is defined at set-up time, and rarely changes afterwards (part of the "metadata")

Since a schema describes how data is to be structured, **data** is actual an "instance" of database. Data may change rapidly and we may compare it with "*variables and types*" from programming languages.

Examples:

- data model is a *graph*. E.g., nodes might represent cities, edges represent airline routes
- data model is an *XML document*. E.g., document might contain list of books with ISBN numbers as ID's, titles and author names as sub-elements
- data model is *set of records*. E.g., records might each have student-ID, name, address, courses, photo

The first important data model is *Hierarchical Model*. It was defined in the late of 1960s and evolved from file-based processing systems. A schema using this data model organizes data into a tree-like structure.

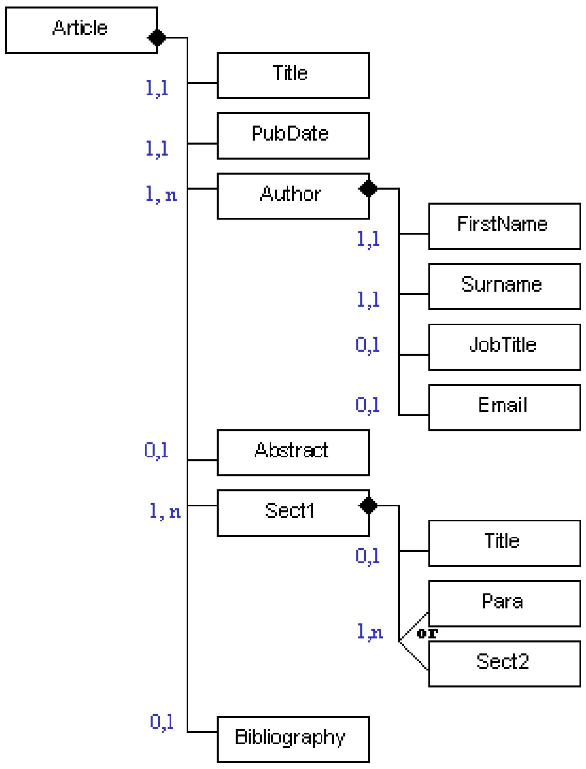


Figure 1.1 Schema of an *Article* entity using *Hierarchical Model*

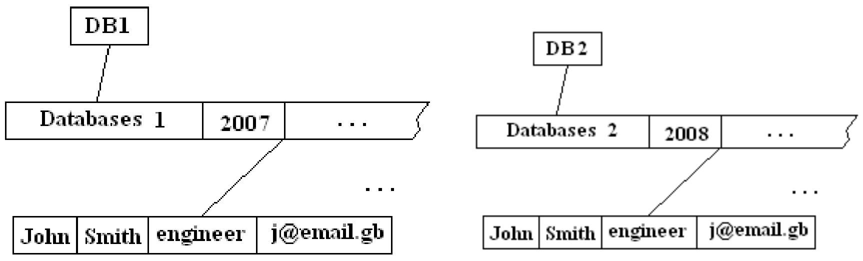


Figure 1.2 Two particular instances of schema presented in Fig. 1.1

Another data model is *Network Model* which is an extension of *Hierarchical Model*, organizing data into a graph-like structure. Figure 1.3 shows a database schema using the *Network Model.*

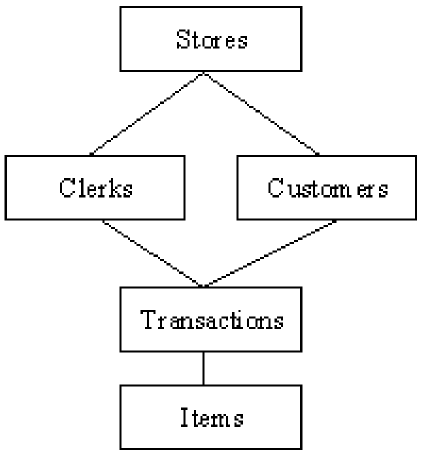
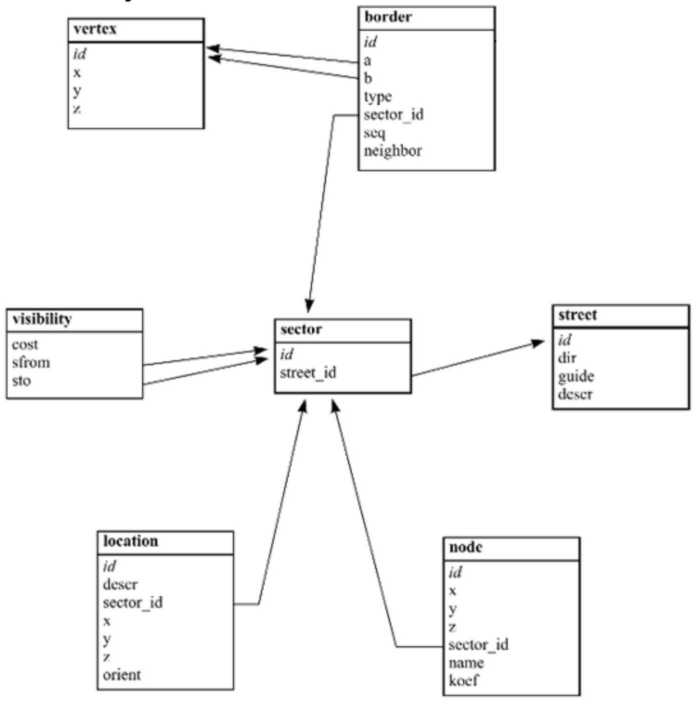


Figure 1.3.

Early 1970's Ted Codd invented *Relational Model* and the concept of data abstraction. Relational model is the most popular data model used in today databases and it will be extensively described in next courses.



vertex

id	x	y	z
1	30	30	30
2	30	30	0
3	30	0	0
4	0	0	0
5	0	0	30
6	0	30	30

Figure 1.4 Relational model example

Object Oriented Model – introduces concepts (like class, attribute, method) and relationships between them (association, aggregation, inheritance). Object oriented model is very popular as analysis, design and programming philosophy. In databases, due to efficiency reasons it remains has only a "scientific" interest.

Schema vs. Data

Database schema describes the structure of the database

- Changed infrequently;
- a.k.a. *database intension;*
- a.k.a. *metadata* (= data about data);

Database state refers to the data in the database at any given moment (snapshot)

- Changes frequently;
- a.k.a. *database extension;*
- DBMS assures that all database states are *valid* states.

Database Instance refers to the combination of schema and state

Database Management Systems

While a database is simply the structured collection of data (usually handled by a database engine), applications that utilize information contained in databases are typically created with a database development environment. These development tools are unlike traditional programming environments in that they provide tools to create applications that allow users to manage data easily, without having to deal with the low-level details normally associated with traditional programming (such as memory management, etc.) Instead, database development environments typically have tools for creating forms using a form editor, some kind of scripting environment or specialized high-level language for controlling the interface and access to the database engine, and other high-level tools specifically designed for manipulating information

Such a database development environment is part of a so called Database Management System (DBMS)

DBMS is a collection of software that facilitates the implementation and management of database applications
Database Management System (DBMS): Provides efficient, convenient, and safe multi-user storage of and access to massive amounts of persistent data

Modern DBMS hide the physical complexity from users through several levels of abstraction while providing simple and efficient access methods

Examples of DBMS

- Record-based data models:
    - *Relational model* (e.g. DB2, Informix, Oracle, MS SQL Server, MS Access, FoxBase, Paradox, MySQL)
    - *Hierarchical model* (e.g. IBM's IMS DBMS)
    - *Network model* (used in IDMS)
- Object-based data models:
    - *Object-oriented model* (e.g. Objectsore and Versant)
    - *Object-relational model* (e.g. Illustra, O2, UniSQL).

Some recent trends:

- DBMS are getting smaller and smaller
    - DBMS that can store GB of data can run on PC
- Databases are getting bigger and bigger
    - Multiple TBs (terabyte = 10^{12} bytes) not uncommon
    - Databases also able to store images, video, audio
    - Database stored on secondary storage devices
- DBMS Supporting Parallel Computing
    - Speed-up query processing through parallelism (e.g., read data from many disks)
    - However, need special algorithms to partition data correctly

When Use Databases? When we have to deal with:

- Persistence
- Large Amounts of Data
- Structured Data
- Concurrent and Distributed Access
- Integrity
- Security
- Data shared with other applications

When NOT to Use Databases?

- Initial investment too high
- Too much overhead
- Application is simple, well-defined, not expected to change
- Multi-user access to data is not required

Flat files

A flat file database is a primitive database that stores data in a plain text file. The main drawbacks of using flat files are:

- Updates and deletions are expensive
- Search is expensive (always have to read the entire file)
- "Brute force" query processing
- No buffer management - application must stage large datasets between main memory and secondary storage (e.g., buffering, page-oriented access, 32-bit addressing, etc.)
- Special code for different queries
- No concurrency control – must protect data from inconsistency due to multiple concurrent users
- No reliability (can lose data or live operations half done)
- No API or GUI
- No security and access control
- No crash recovery

Relational Model

Use a simple data structure: *the Table*

- simple to understand
- useful data structure (capture many situations)
- leads to useful not too complex query lang.

Use mathematics to describe and represent records and collections of records: *the Relation*

- can be understood formally
- leads to formal query languages
- properties can be explained and proven

Relation – formal definition

A **domain** is a set of scalar values (i.e. limited to atomic types - integer, string, boolean, date, blobs). A **relation** or **relation schema R** is a list of **attribute names** [A_1, A_2, …, A_n].

D_i = Dom(A_i) - domain of A_i, i=1..n

Relation instance ([R]) is a subset of

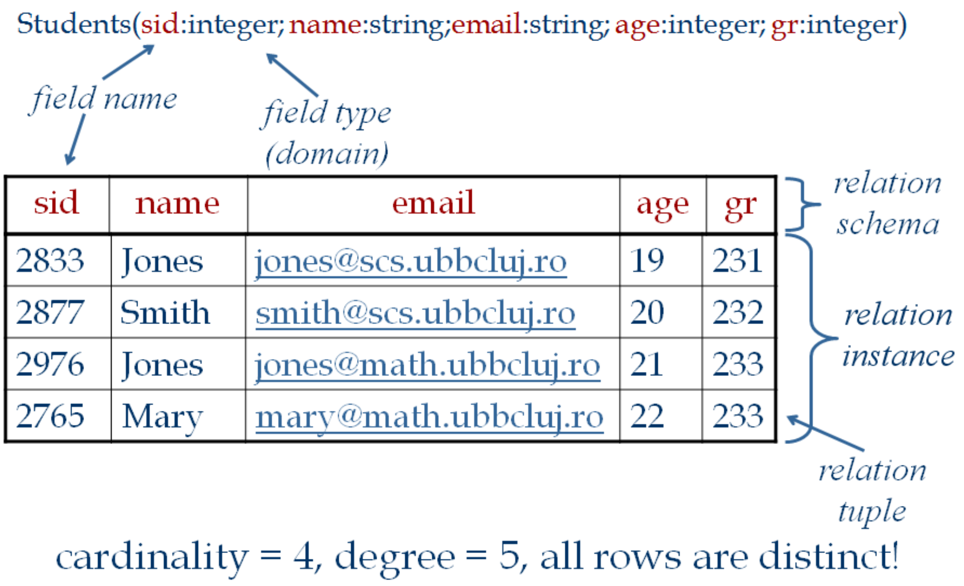
$D_1 \times D_2 \times \ldots$ □ D_n

Degree (**arity**) = the number of attributes in a relation scheme

Tuple = an element of the relation instance, a record. All tuples of a relation are distinct!

Cardinality = the number of tuples of a relation

Relation example


Students(sid:integer; name:string;email:string; age:integer; gr:integer)

sid	name	email	age	gr
2833	Jones	jones@scs.ubbcluj.ro	19	231
2877	Smith	smith@scs.ubbcluj.ro	20	232
2976	Jones	jones@math.ubbcluj.ro	21	233
2765	Mary	mary@math.ubbcluj.ro	22	233

cardinality = 4, degree = 5, all rows are distinct!

Very often we will confuse…

- the *relation*, its *schema*, and its *instance*
- the *instance* and the *table*
- the *attribute*, the *field* and the *column*
- the *tuple*, the *record* and the *row*

Ask for precision if there is ambiguity!

A **database** is a set of relations. A **database schema** is the set of schemas of the relations in the database. A **database instance** (**state**) is the set of instances of the relations in the database.

Integrity Constraints

Integrity Constraints (ICs) are conditions that must be true for *any* instance of the database. ICs are specified when schema is defined and are checked when relations are modified.

A *legal* instance of a relation is one that satisfies all specified ICs.

Integrity Constraints – samples:

Students*(sid:string, name:string, email:string, age:integer, gr:integer)*

Domain constraints: *gr:integer*

Range constraints: $18 \leq age \leq 70$

TestResults*(sid:string, TotalQuestions:integer, NotAnswered:integer, CorrectAnswers:integer, WrongAnswers:integer)*

TotalQuestions = NotAnswered+CorrectAnswers+WrongAnswers – **not an IC!**

Primary Key Constraint

A set of fields is a **key** for a relation if :

1. No two tuples can have same values for all fields

AND

2. This is not true for any subset of the key.

If the 2nd statement is false → **superkey**.

If there's >1 key for a relation → **candidate keys**

One of the candidate keys is chosen as **primary key**

Foreign Keys, Referential Integrity

A **foreign key** is a set of fields in one relation that is used to `*refer*' to a tuple in another relation (Like a `*logical pointer*') It must correspond to primary key of the second relation.

E.g. *sid* is a foreign key referring to Students:

Enrolled (*sid*: string, *cid*: string, *grade*: double)

Referential integrity = all foreign key constraints are enforced → no dangling references.

Let's consider ***Students*** and ***Enrolled*** tables: *sid* in ***Enrolled*** is a foreign key that references a record from ***Students*** table.

What should be done if an ***Enrolled*** record with a non-existent student id is inserted? The DBMS will reject it.

What should be done if a ***Students*** tuple is deleted? There are 4 approaches:

- Also delete all ***Enrolled*** tuples that refer to it.
- Disallow deletion of a ***Students*** tuple that is referred to.
- Set *sid* in ***Enrolled*** tuples that refer to it to a *default sid*.
- Set *sid* in ***Enrolled*** tuples to a special value *null*, denoting *`unknown'* or *`inapplicable'*.

The same approaches we have if primary key of ***Students*** tuple is updated.

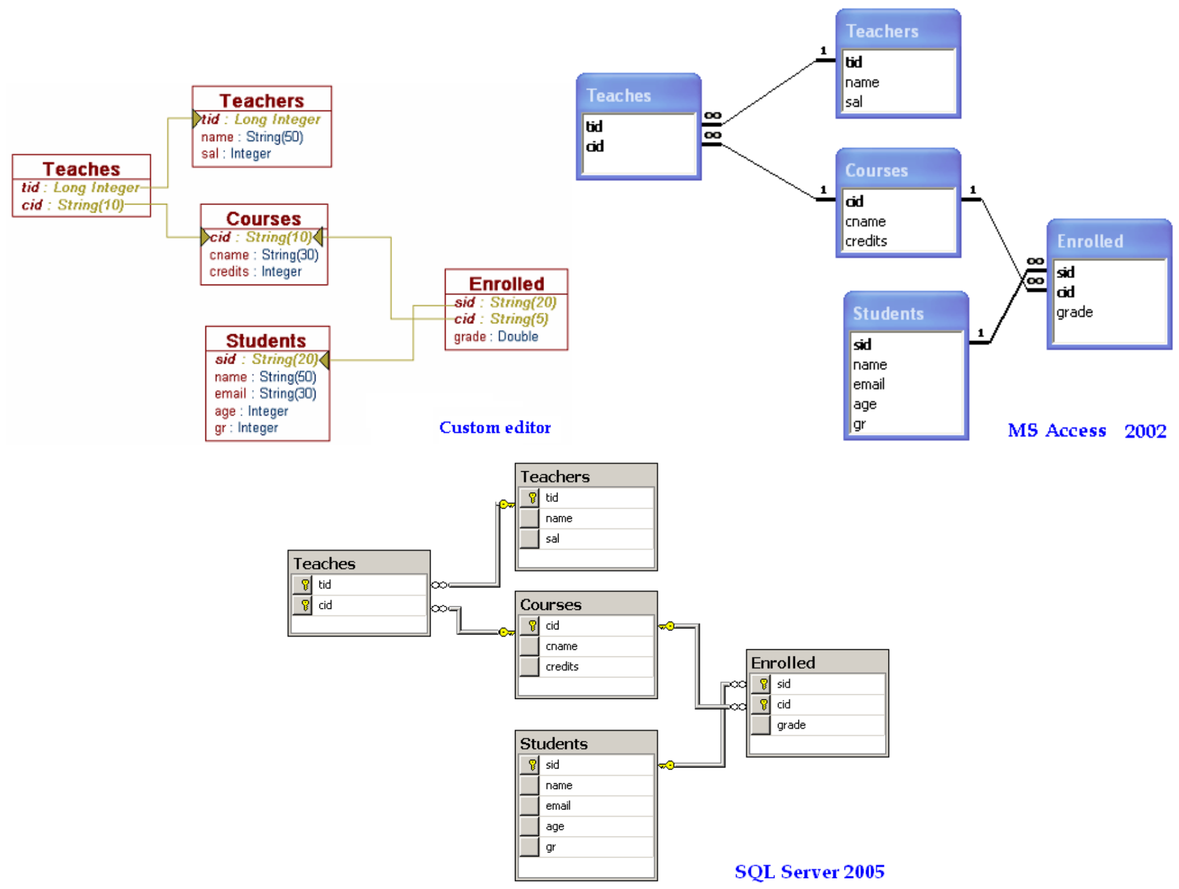


Figure 1.5. Samples of graphical representation of tables, fields, primary keys and foreign keys in

Keys and foreign keys are the most common ICs. ICs are based upon the semantics of the real-world enterprise that is being described in the database relations. We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.

An IC is a statement about *all possible* instances of a particular table! For example, for ***Students*** table we know *name* is not a key, but the assertion that *sid* is a key is given to us.