Course 2. The Relational Model

Layered Approach to Database Implementation

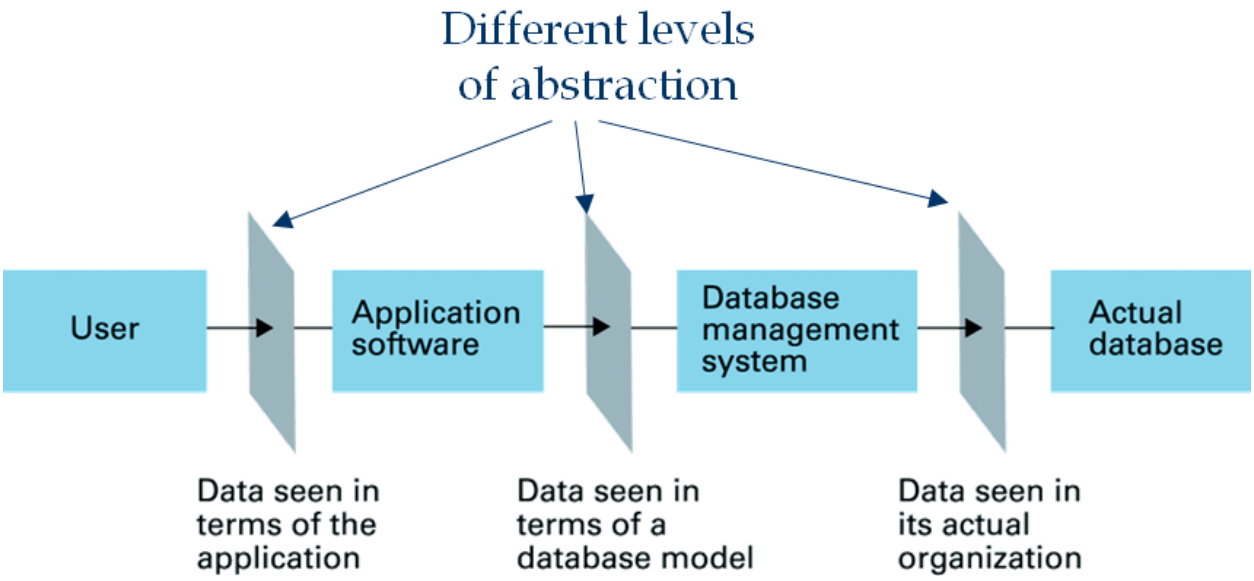


Figure 2.1.

Levels of Abstraction:

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

Example: ***Faculty*** database
Conceptual schema:
Students(sid:string, name:string, email:string, age:integer, gr:integer)
Courses(cid: string, cname: string, credits:integer)
Exams(sid:string, cid:string, grade:integer)

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

External Schema (View):
Course_info(cid:string, enrollment:integer)

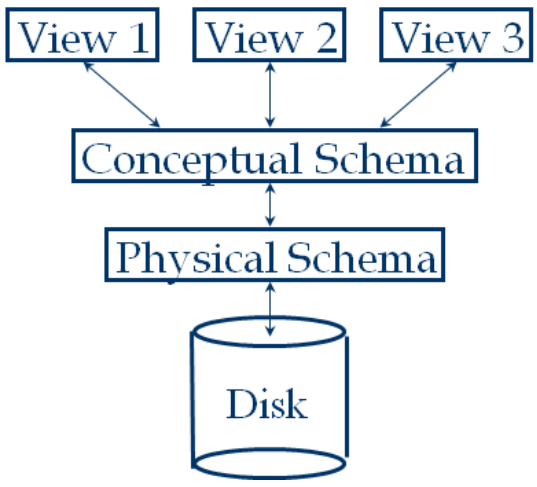


Figure 2.2.

One of the most important benefits of using a DBMS is that the applications are isolated from how

data is structured and stored.

- *Logical data independence*: Protection from changes in *logical* structure of data.
- *Physical data independence*: Protection from changes in *physical* structure of data.

Queries in a DBMS

For the sample Faculty Database, here are some questions that users may ask:

- "*What is the name of the student with sid 2833*"?
- "*What is the salary of the professor who teaches the course with cid Alg100*"?
- "*How many students are enrolled in course Alg100*"?

Such questions involving data stored in a DBMS are called ***queries*.** A DBMS provides a specialized language, called the ***query language***, in which queries can be posed. Each query language is composed by:

- *Data Definition Language (DDL)*
  - Used to define the conceptual and internal schemas;
  - Includes constraint definition language (CDL) for describing conditions that database instances must satisfy;
  - Includes storage definition language (SDL) to influence layout of physical schema (some DBMSs);
- *Data Manipulation Language (DML)*
  - Used to describe operations on the instances of a database
  - Procedural DML (how) vs. declarative DML (what).

Query Languages for Relational DBs

SQL (Structured Query Language):

SELECT *name* FROM *Students* WHERE *age* > 20

Algebra:

$\pi_{name}(\sigma_{age > 20}(Students))$

Domain Calculus:

$\{<X> | \exists V \exists Y \exists Z \exists T : Students(V, X, Y, Z, T) \wedge Z>20\}$

T-uple Calculus:

$\{X | \exists Y : Y \in Students \wedge Y.age > 20 \wedge X.name = Y.name\}$

Databases related actors

- System Analyst: Designs entity-relationship diagram

- Database Designer: Designs logical /physical schemas
- Application Programmer
- Database Administrator
    - Handles security and authorization
    - Data availability, crash recovery
    - Database tuning as needs evolve
- System Administrator
- End Users (Naive / Sophisticated end users)

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

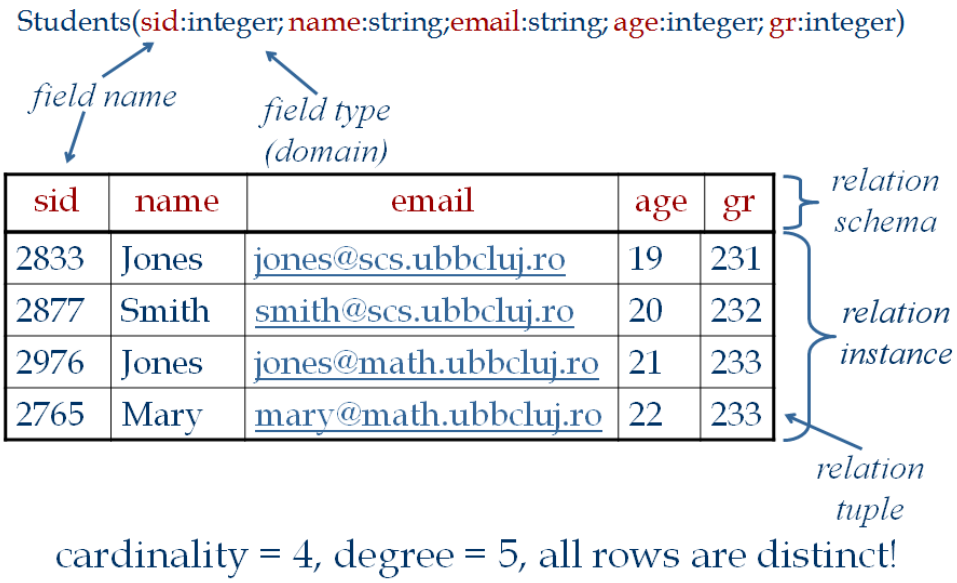
$$D_1 \times D_2 \times \ldots \square D_n$$

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



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.

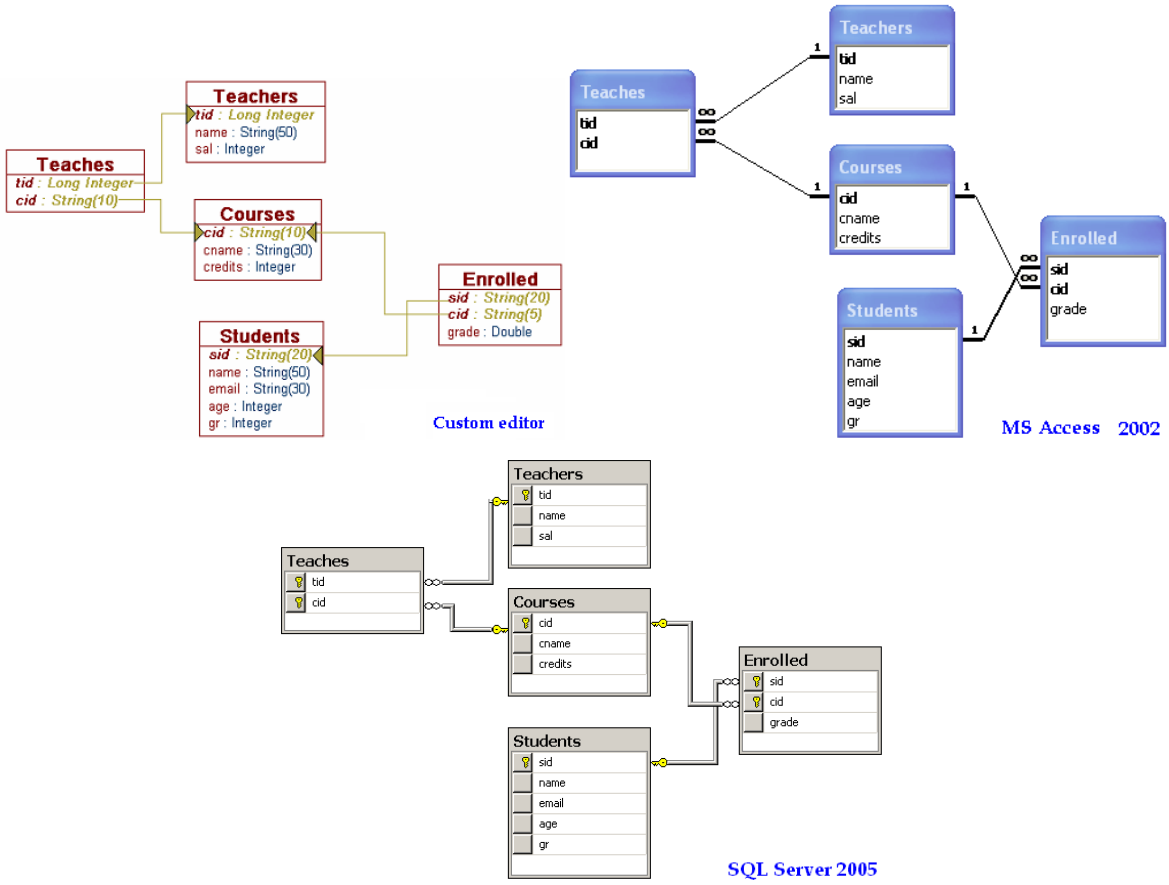


Figure 2.3. Samples of graphical representation of tables, fields, primary keys and foreign keys in

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.

Relational Query Languages

A major strength of the relational model is that it supports simple, powerful *querying* of data. Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.

- The key: precise semantics for relational queries.
- Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

Structured Query Language (SQL) became de-facto standard for query languages based on the relational data model. It was developed by IBM (system R) in the 1970. It was a need for a standard since it is used by many vendors

SQL Standards:

- SQL-86
- SQL-89 (minor revision)
- SQL-92 (major revision) - *1,120 pages*
- SQL-99 (major extensions) - *2,084 pages*
- SQL-2003 (SQL/XML – new section) - *3,606 pages*
- SQL-2008
- SQL- 2011

SQL Levels

Data-definition language (DDL):

- Create / destroy / alter *relations* and *views*.
- Define *integrity constraints* (IC's).

Data-manipulation language (DML)

- Allows users to pose queries
- Insert /delete / modify (update) tuples.

Access Control:

- Can grant / revoke the right to access and manipulate tables (relations / views).

DDL Commands

The following command creates the *Students* table (relation). Observe that the type (domain) of each field (attribute) is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
        (sid CHAR(20),
         name CHAR(50),
         email CHAR(30),
         age INTEGER,
         gr INTEGER)
```

As another example, the *Enrolled* table holds information about courses that students take.

```
CREATE TABLE Enrolled
          (sid CHAR(20),
           cid CHAR(5),
           grade REAL)
```

To destroy (remove) the relation *Students* the following command could be used. Of course, both schema information and the tuples are deleted.

```
DROP TABLE Students
```

Using the following command, the schema of *Students* table is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

```
ALTER TABLE  Students
ADD COLUMN firstYear INTEGER
```

The schema of *Students* is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

SQL could be used to declare many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.

Sample: *For a given student and course, there is a single grade*.

```
CREATE TABLE Enrolled
    (sid CHAR(20),
     cid  CHAR(20),
     grade CHAR(2),
PRIMARY KEY (sid,cid))
```

Sample: *Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade*. (this is an example about how to not define candidate keys; used carelessly, an IC can prevent the storage of database instances that arise in practice!)

```
CREATE TABLE Enrolled
    (sid CHAR(20),
     cid  CHAR(20),
     grade CHAR(2),
 PRIMARY KEY(sid),
 UNIQUE (cid, grade))
```

Sample of defining foreign keys "*Only students listed in the Students relation should be allowed to enroll for courses*".

```
CREATE TABLE Enrolled
   (sid CHAR(20),  cid CHAR(20),  grade CHAR(2),
     PRIMARY KEY  (sid,cid),
     FOREIGN KEY (sid) REFERENCES Students )
```

Referential Integrity

Starting with SQL-99 there is support for all 4 approaches preserving referential integrity in case of deletes and updates:

- NO ACTION (*delete/update is rejected*) – it is the default approach is
- CASCADE (also delete all tuples that refer to deleted tuple)
- SET NULL/SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
   (sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid,cid),
    FOREIGN KEY (sid)
      REFERENCES Students
  ON DELETE CASCADE
  ON UPDATE SET NULL )
```

General constraints are useful when more general ICs than keys are involved:

```
CREATE TABLE Students
  (sid CHAR(20),
   name CHAR(50),
   email CHAR(30),
   age INTEGER,
   gr INTEGER,
   PRIMARY KEY (sid),
   CONSTRAINT ageInterv
   CHECK (age >= 18
         AND age<=70))
```