Data and Analysis :: Structured Data

Stratis Viglas

January 2006

Outline

Course logistics

- *Lecturers*:
  - Stratis Viglas (*sviglas@inf.ed.ac.uk*) — structured data
  - Helen Pain (*helen@inf.ed.ac.uk*) — semi-structured, non-structured data
- *Teaching assistant*: Manuel Marques-Pita (*m.marques-pita@ed.ac.uk*)
- No required textbook, only lecture notes and selected chapters
- *Six tutorials*, *four lab sessions*
- *Four* assignments (one shared with Object-Oriented Programming)

Structured data

- For some *application domains*, data is *inherently structured*
  - For instance, all students share common information
- In such domains, it makes sense to *organise* the data in a way that *directly maps* to their *physical properties*
  - Not only that, but also *devise mechanisms* to *manipulate* them without compromising their properties
- We will deal with two main *data representation* models:
  - The *entity/relationship* model, and
  - The *relational* model
- Finally, we will deal with data *manipulation* for the *relational model*, in particular:
  - *Relational algebra*, and
  - *Tuple-relational calculus*

Steps involved

1. *Requirements analysis*
   - *Gather* data and *identify* their structure
   - At all times, keep the *mode of interaction* in mind
2. *Conceptual design*
   - *Organise* the data in a way that their semantic links are *stored* and *preserved*
   - Not only identify how data interact, but also how *users interact* with *data*
3. *Logical design*
   - *Implement* the conceptual design in terms of the application at hand
   - This involves *mapping* the conceptual design in a *logical data representation*
   - The outcome is a *logical schema*

Outline

The ER model

What is it used for?

The ER model is a way to describe *real-world entities* and the *relationships* between them

Why is it useful?

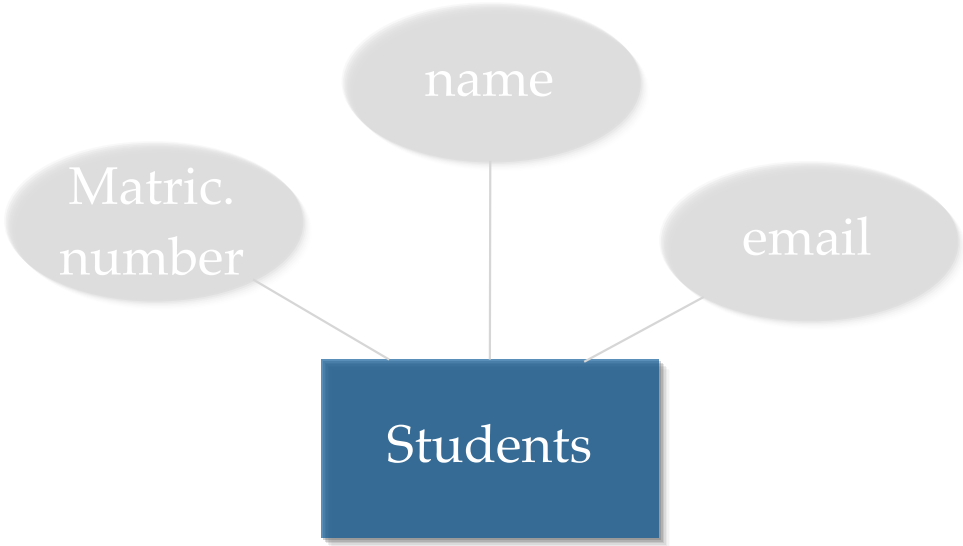
Because it *maps* to different *logical data models*, including the relational model

How is it used?

It is essentially a way to *visualise* data and their *dependencies*

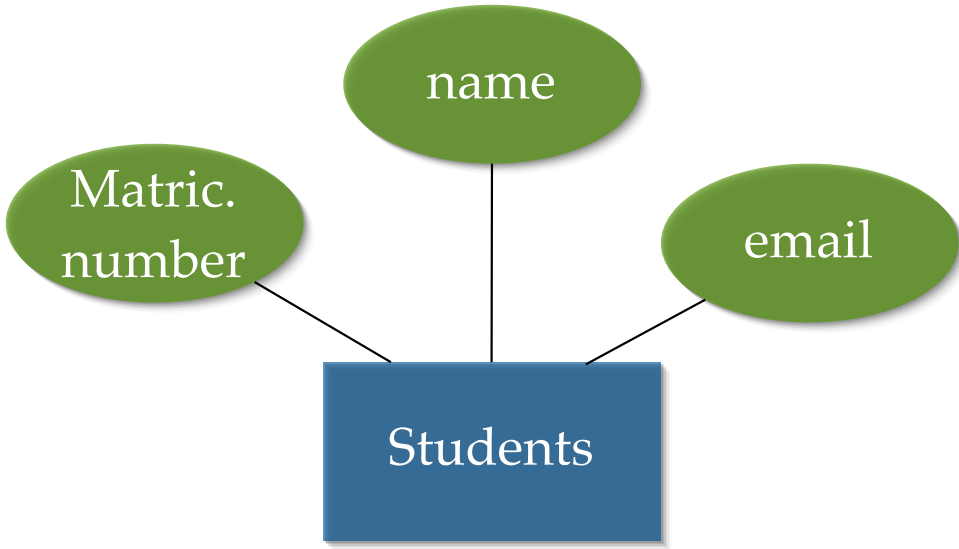
Entities and entity sets

- Any *distinguishable object* in the real world is an *entity*
- A *collection* of the *same type* of *entities*, is an *entity set*
  - Entity sets are *represented by boxes*, labeled with the entity set's name



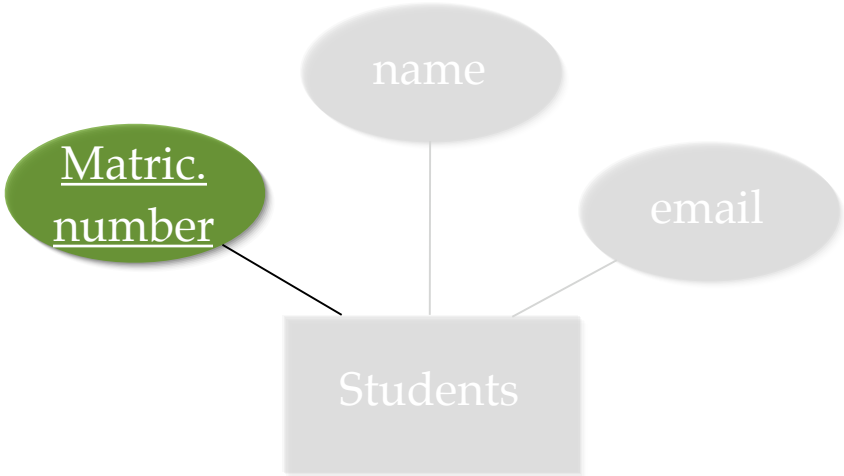
Attributes and domains

- *Each entity* of the *same entity set*, has some characteristic *attributes*
  - Attributes are *represented by ovals*, labeled with the attribute's name, connected to the entity set they belong to
  - Each attribute has a *domain* from which *allowable values* are derived



Keys

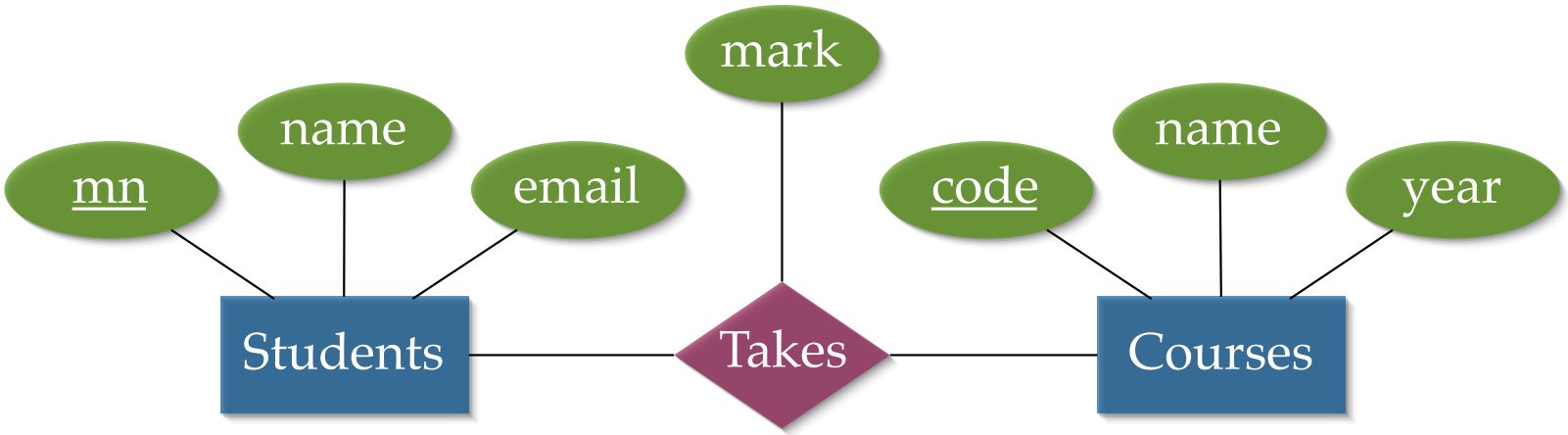
- A *key* is the *minimal set of attributes* whose *values* allow us to *uniquely identify* an *entity* in an entity set
- There *may be more than one* such minimal *sets*; they are called *candidate keys*
  - For instance, either the matriculation number, or the email address can act as keys



- If *multiple candidate keys* exist, we *choose* one and make it the *primary key*; the attributes of the primary key are *underlined* in the ER diagram

Relationships and relationship sets

- *Relationships* model *associations* between *entity sets*
  - *Grouped* into *relationship sets* (just as entities are grouped into entity sets
- Represented as *diamonds* in ER diagrams
- Relationships may *have attributes* of their *own*



Tricky bits

Instances

Entity instances and *relationship instances* are what we obtain after *instantiating* the *attributes* of an entity or a relationship

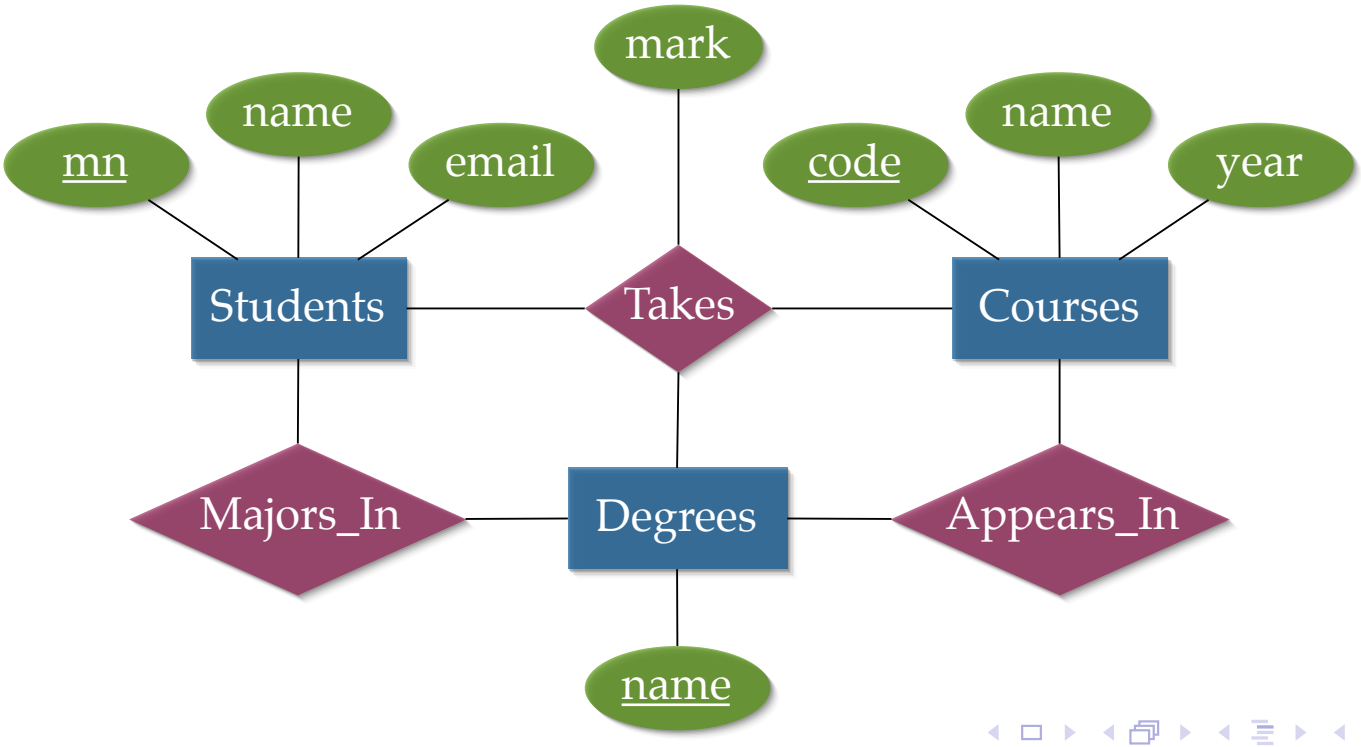
Example

- An instance of the *Students* entity set
  - (123, *Natassa*, *natassa@somewhere*)
- An instance of the *Courses* entity set
  - (*inf1*, *Informatics 1*, 1)
- An instance of the *Takes* relationship set
  - (123, *Natassa*, *natassa@somewhere*, *inf1*, *Informatics 1*, 1, 88)

Tricky bits

How many entities in a relationship?

There is *no bound* on the *number of entities participating* in a *relationship*; correspondingly, there is *no bound* on the *number of relationships* an *entity* can *participate* in



Key constraints

Semantics

Key constraints capture the *identification* connections between *entities* participating in a *relationship*

Definition

R is a *relationship* between *n entity sets*, $E_1, \ldots, E_n$; if there is a *key constraint* on one of the entities, E_k, then by *instantiating* the *attributes* of E_k we can *uniquely identify* the *relationship instance* it participates in

Example

Students, directors of studies (DoS), and the relationship between them (Directed-By)

- *Each student* has a *unique DoS*; *given a student instance*, we can *determine the Directed-By instance* it appears in

One-to-many and many-to-many relationships

Intuition

Another way of looking at things stems from *"counting"* how many *relationship instances* a *single entity* instance appears in

Definition

A *one-to-many* relationship R between entity sets E_o and E_m means that *each instance* $e_o \in E_o$ may *appear* in *at most one* relationship instance $r \in R$; there is *no constraint* on entity instances $e_m \in E_m$

Definition

A *many-to-many* relationship R between entity sets E_o and E_m means that there are *no constraints* on the *number of times* entity instances $e_o \in E_o$ and $e_m \in E_m$ may appear in relationship instances $r \in R$

One-to-many and many-to-many relationships examples

Example

The *Directed_By* relationship between *Students* and *DoSs* is a *one-to-many relationship*

- A single DoS has many students
- Many students have the same DoS

Example

The *Takes* relationship between *Students* and *Courses* is a *many-to-many relationship*

- A single student may take many different courses;
- A single course may be taken by many different students

Participation constraints

Semantics

Participation constraints capture the *mode* in which an *entity participates* in a *relationship*

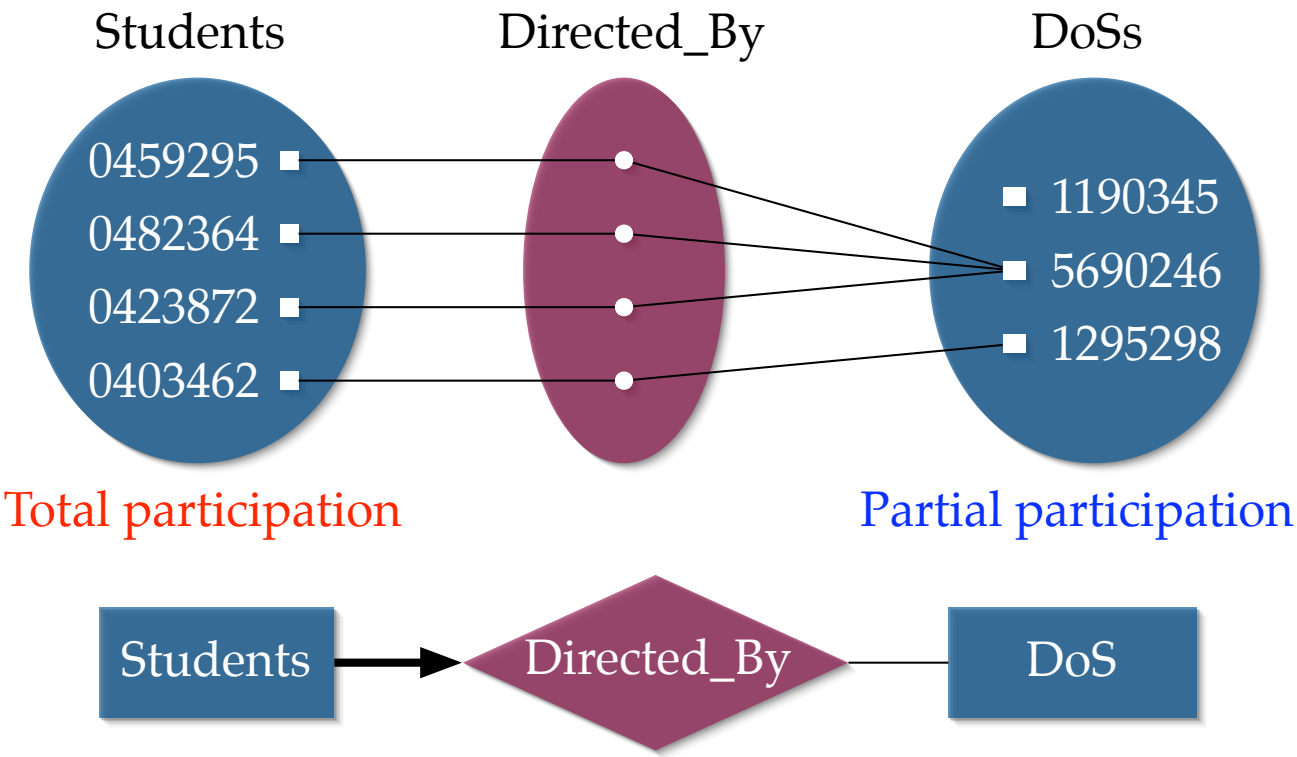
Definition

Total participation on *entity set* E for *relationship* R is declared when *every entity* $e \in E$ *appears* in a *relationship instance of* R

Definition

Partial participation on *entity set* E for *relationship* R is declared when *there exist entities* $e \in E$ that *do not appear* in *instances of* R

Participation constraint example



Notation

A *thick arrow* from the *totally participating entity* to the *relationship* denotes total participation

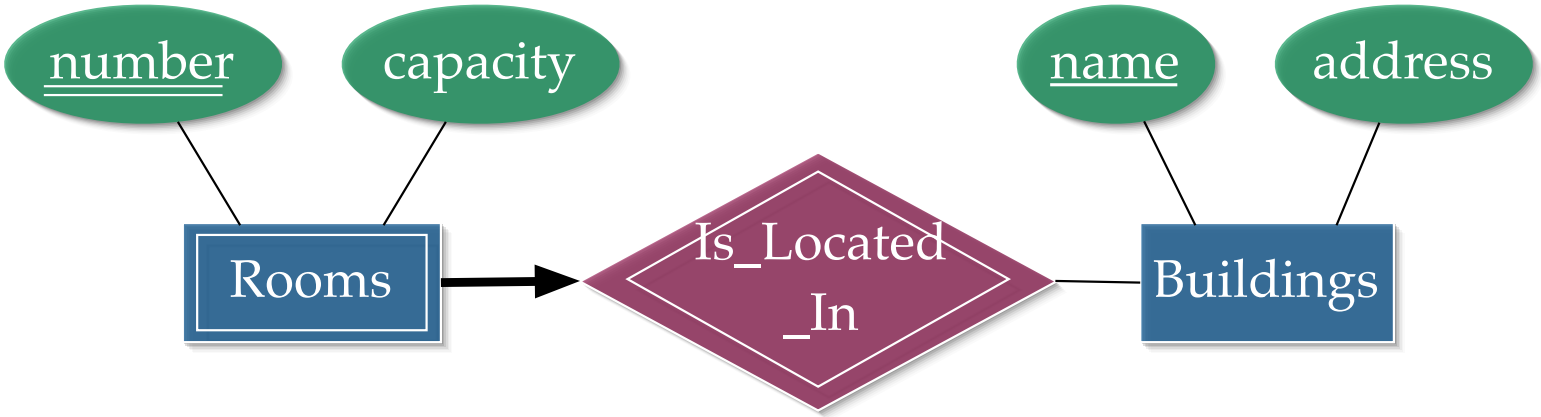
Weak entity sets

- In certain cases, it is *impossible* to *designate a primary key* for entities of an entity set
- Instead, the only way in which *set participation* can be *declared* is by *"borrowing"* the *key* of *another relation*

Definition

- A *weak entity set* is an entity set for which a *primary key* consisting only of its *own attributes cannot be identified*
- The *key* is formed by a *combination* of its *own attributes* and *attributes from another entity set* with which it has a relationship
- The *entity set* from which attributes are *borrowed* is called the *identifying owner*
- The *relationship* between the *weak entity set* and its *identifying owner* is called an *identifying relationship*

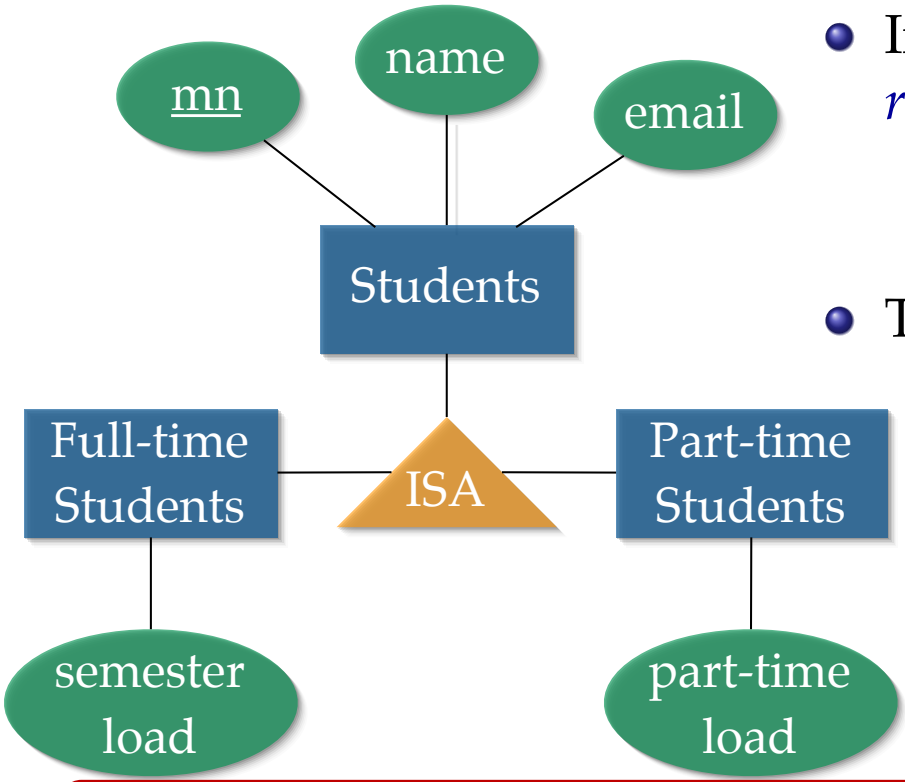
Weak entity set example



Notation

- *Double lines* for *weak entity* and *identifying relationship*
- *Doubly underlined attributes* of the *weak entity set* participating in the composite key
- *Total participation* of the *weak entity set* in the *identifying relationship*

Hierarchical entities and inheritance



- In certain cases, *entities* can be further *refined*
  - A *core set of attributes* is retained and *extra attributes* are added
- The mechanism is called *inheritance*
  - *Subclasses inherit* core properties from *superclass* and *extend* these properties
  - *Changes* to *attributes* are *visible only* at the *appropriate* level

Terminology

A *superclass* is *specialised* through inheritance into *subclasses*; *subclasses* are *generalised* by the *superclass*

Outline

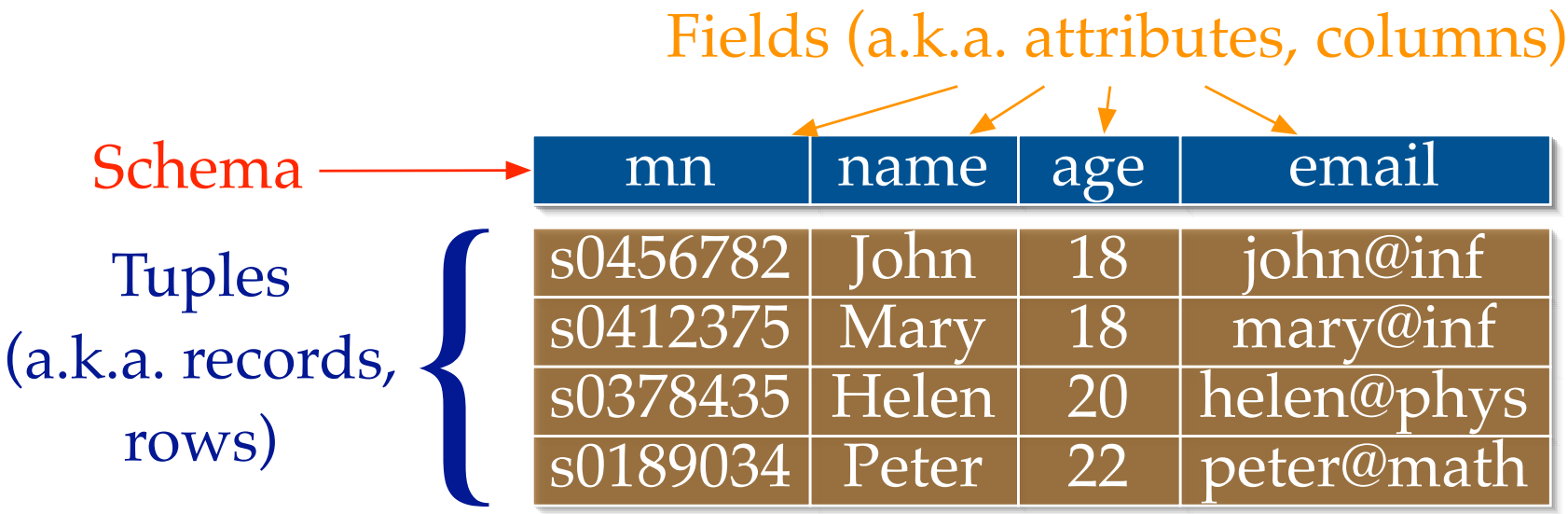
Relational databases and the relational model

- The *relational model* was *introduced* by *E.F. Codd* in *1970*
- *Adopted* by *industry* (IBM) and *academia* (UC-Berkeley)
- The first *research prototypes* of *relational database management systems* introduced in *mid 1970's*
  - IBM's System R and UC-Berkeley's Ingres
- *Nowadays* it is a *multi-billion* pound industry
- One of the *simplest data models* ever
  - It simply shows that *simplicity* goes a *long way*
  - *K.I.S.S.* (*Keep It Simple, Stupid*)

Building blocks

- The *basic construct* is a *relation*
  - It consists of a *schema* and an *instance*
  - The *schema* can be thought of as the *format* of the relation
  - A *relation instance* is also known as a *table*
- A *schema* is a *set of fields*, which are *(name, domain)* pairs
  - *Fields* may be referred to as *attributes*, or *columns*
  - *Domains* are referred to as *types*
- The *rows* of a table are called *tuples* (or *records*) and they are *value assignments* for the *fields* of the table
- The *arity* of a relation is its *number of columns* (fields)
- The *cardinality* of a table is its *number of rows* (tuples)

Example


Fields (a.k.a. attributes, columns)
Schema
mn
name
age
email
Tuples
(a.k.a. records,
rows)
s0456782
John
18
john@inf
s0412375
Mary
18
mary@inf
s0378435
Helen
20
helen@phys
s0189034
Peter
22
peter@math

Data definition in SQL

- SQL stands for *Structured Query Language*
- A special subset of SQL called the *Data Definition Language* (DDL) is used to *declare table schemata*
- *Relations* are called *tables* in SQL
- It is a *typed* language
  - For simplicity, we will assume there are only three types: (i) `integer` for *integer numbers*, (ii) `real` for *real numbers*, and (iii) `char(`*n*`)` for a *string of maximum length n*

General form of a DDL statement

`create table` *table name* `(` *attribute name* *attribute type*
`[,` *attribute name* *attribute type* `]*`
⟨*integrity constraints*⟩ `)`

Primary key constraints

- We have already seen that *entities* can have *keys*
  - Though there may be *multiple candidate keys*, we *choose* a *single primary key*
- The *same mechanism* is in place in the *relational model*
- A *primary key* essentially imposes the *set participation constraint* for table rows

SQL

Primary key constraints are implemented in SQL by the:

`primary key` (*declaration*)

construct

Semantics of primary key constraints

Example

```
create table Students (
        mn              char(8),
        name            char(20),
        age             integer,
        email           char(15),
        primary key     ( mn ) )
```

- This *definition* for the *Students* table *imposes* the following *constraint*:
  - There can only exist *a single row* with a *given* mn *value* in the table
- Any *attempts* to *insert* another row with an mn *value* that *already exists* in some other row of the table will *fail*
  - The *system enforces* the primary key *constraint*

Foreign key constraints

- *Foreign key constraints* semantically *link* table *columns* of *different tables*
  - In the same way entity sets can be linked
- In the *relational model*, they are *declared* by the user and *enforced by the model/system* itself

SQL

Foreign key constraints are implemented in SQL by the:

`foreign key` (*declaration*) `references` *table*

construct

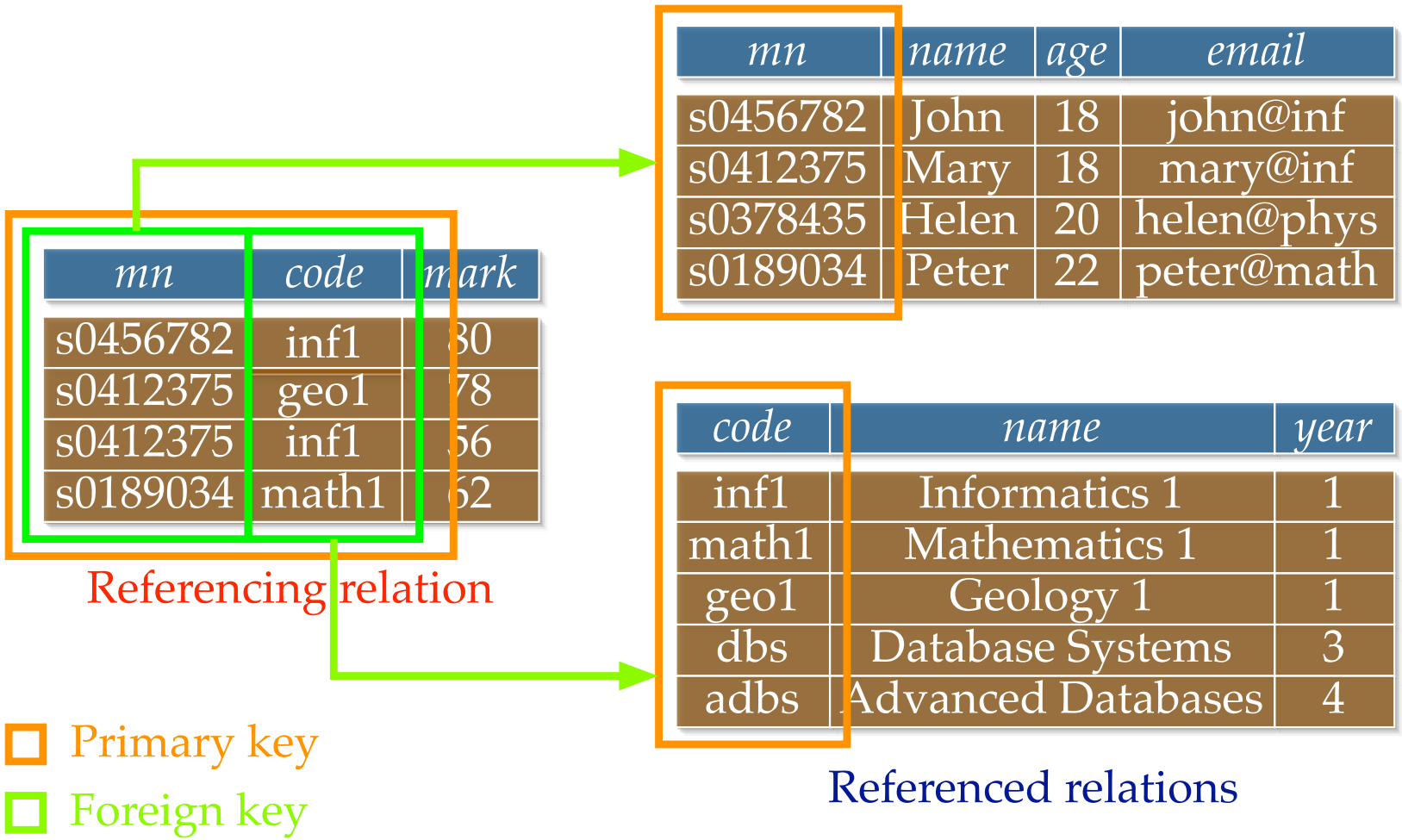
Semantics of foreign key constraints

Example

```
create table Takes (
        mn              char(8),
        code            char(20),
        mark            integer,
        primary key     ( mn, code ),
        foreign key     (mn) references Students,
        foreign key     (code) references Courses)
```

- This definition essentially *imposes* two *constraints* on the Takes table:
  - Whenever a *tuple* is *inserted*, the *value* for the mn field *must* be a value that *appears* in the *matriculation number column* of the Students table
  - The *value* for the code field *must* be a value that *appears* in the *corresponding column* of the Courses table

Foreign key constraints example



Mapping entity sets

Algorithm

- A *table* is created for the *entity set*
- *Each attribute* of the *entity set* becomes an *attribute* of the *table* with an *appropriate type*
- A *primary key* is *declared*

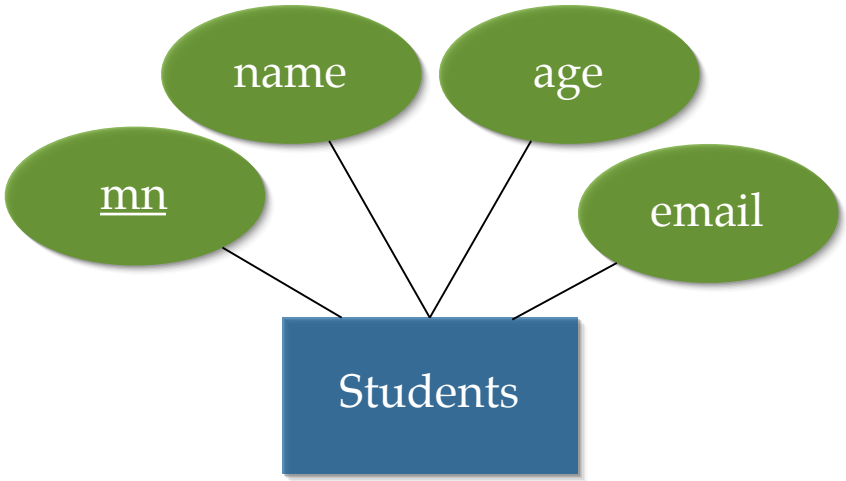


Table definition

```
create table Students (
        mn              char(8),
        name            char(20),
        age             integer,
        email           char(15),
        primary key     ( mn ) )
```

Mapping relationship sets (no key constraints)

Algorithm

- A *table* is created for the *relationship set*
- The *table contains* the *primary keys* of the *participating entity sets*
- *Descriptive attributes* of the relationship are *added*
- A *composite primary key* is *declared* on the table
- *Foreign key* constraints are *declared*

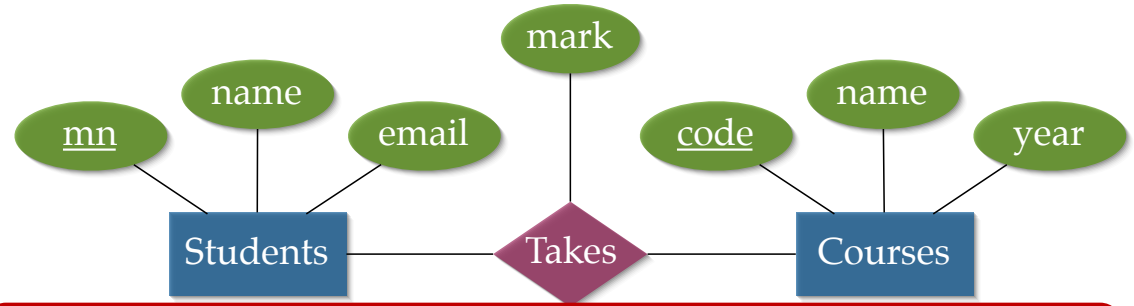


Table definition

```
create table Takes (
        mn                  char(8),
        code                char(20),
        mark                integer,
        primary key         ( mn, code ),
        foreign key         (mn)
           references       Students,
        foreign key         (code)
           references       Courses)
```

Null values and participation constraints

- In SQL, a *special value* a field can have is `null`
  - A `null` value *means* that a field is *undefined* or *missing*
- By *allowing* or *disallowing* `null` values we can *implicitly declare participation constraints*

Table definition

```
create table Takes (
        mn              char(8) not null,
        code            char(20),
        mark            integer,
        primary key     ( mn, code ),
        foreign key     (mn) references Students,
        foreign key     (code) references Courses)
```

Mapping relationship sets (with key constraints)

Algorithm

- A *table* is created for the *relationship set*
- The *primary key* of the *"source"* entity set is *declared* as the *primary key* of the *relationship set*
- *Foreign key constraints* are declared for *both source* and *target* entity sets

Table definition

```
create table Directed_By (
        mn                  char(8),
        staff_id            char(8),
        primary key         ( mn ),
        foreign key         (mn)
           references Students,
        foreign key         (staff_id)
           references DoSs)
```

Mapping weak entity and relationship sets

Algorithm

- Create a *table* for the *weak entity set*
- *Add* an *attribute set*, for the *primary key* of the *entity set's identifying owner*
- *Add* a *foreign key constraint* on the *identifying owners primary key*
- *Instruct* the system to *automatically delete* any *tuples* in the table *for which* there are *no owners*

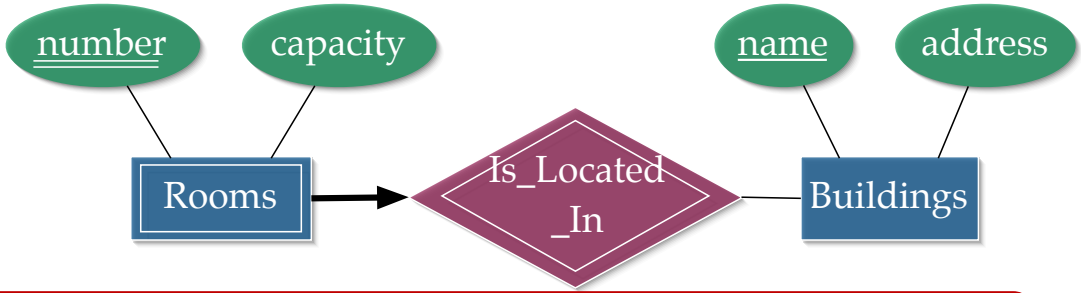


Table definition

```
create table Rooms (
        number          char(8),
        capacity        integer,
        name            char(20),
        primary key     ( number, name ),
        foreign key     (name)
           references Buildings,
           on delete cascade)
```

Mapping hierarchical entities

Algorithm

- *Declare* a *table* for the *superclass* of the hierarchy, specifying its primary key
- For *each subclass*, declare *another table*, containing the superclass's *primary key* and the *subclass's extra attributes*
- Each *subclass* has the *same primary key* as its *superclass*
- *Declare foreign key constraints*

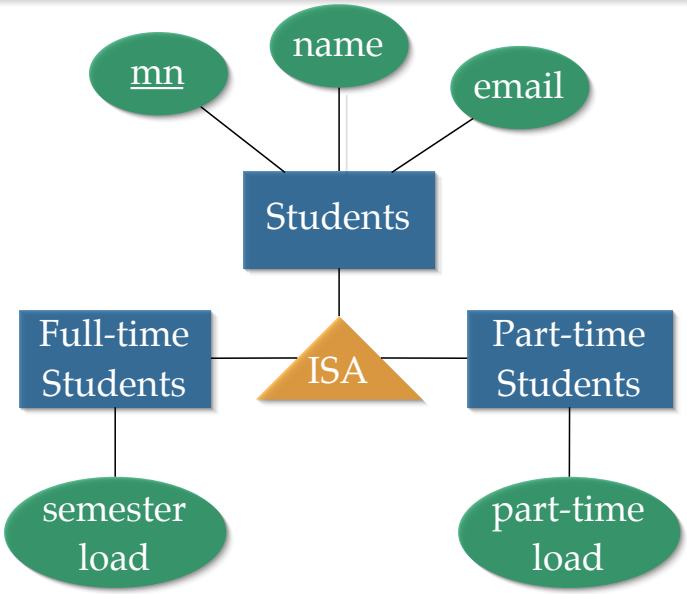


Table definition

```
create table PT_Students (
        mn                  char(8),
        pt_load             integer,
        primary key         ( mn ),
        foreign key         ( mn)
            references Students)
```

Querying

- Once the *data* is *organised* in a relational schema, the natural *next step* is *manipulating* it
  - For our purposes, this means *querying*: *identifying parts* of the data having *properties of interest*
- *Relational algebra*
  - A *procedural* way of expressing queries over relationally represented data
- *Tuple-relational calculus*
  - A *declarative* way of expressing queries, tightly coupled to *first order predicate logic*
- Both *relational algebra* and *tuple-relational calculus* have the *same expressive power*
  - A *query* that can be *expressed* in *one* formalism, can be *expressed* in the *other*

Outline

Relational algebra building blocks

- The key *concept* in relational algebra is an *operator*
- Operators accept a *single* relation or a *pair* of *relations* as *input*
- Operators produce a *single relation* as *output*
- This means that *operators* can be *composed* in order to *form complex* relational algebra *expressions*
  - *Composition* in this context means that an *operator's output* can be *used* as *input* to *another operator*
- There are *five basic operators*
  - Selection, projection, union, cross-product, and difference
- *Other operators* can be *composed* in terms of these five, but are so frequently used that they are treated as *fundamental*

Selection and projection — σ and π

- Recall that *relational data* are basically *tables*
- *Selection* and *projection* allow one to *"isolate"* any *rectangular subset* of a *single table*
  - *Selection* identifies *rows* of interest
  - *Projection* identifies *columns* of interest
- If *both* are used on a *single table*, we have a *rectangular subset* of the table

Selection — σ

- General form: $\sigma_{\text{predicate}}(\text{Relation instance})$
- A *predicate* is a *condition* that is *applied* on *each row* of the table
  - It should *evaluate* to either *true or false*
  - If it evaluates to *true*, the row is *propagated* to the output, if it evaluates to *false* it is *dropped*
  - The *cardinality* of the input is *affected*
- A *predicate* is of the Boolean form
  term_1 `bop` term_2 `bop` ... `bop` term_m
  - Where `bop` $\in \{\vee, \wedge\}$
  - term_i's are of the form attribute `rop` constant or attribute_1 `rop` attribute_2 (`rop` $\in \{>, <, =, \neq, \geq, \leq\}$)

Projection — π

- General form: $\pi_{\text{column list}}(\text{Relation instance})$
- *All rows* of the input are *propagated* in the output
- The *arity* of the output table is *different*
  - Only columns appearing in *column list* appear in the output
- The *resulting relation* has a *different schema*

Selection and projection examples

mn	name	age	email
s0456782	John	18	john@inf
s0412375	Mary	18	mary@inf
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

Students

name	age
John	18
Mary	18
Helen	20
Peter	22

$\pi_{\text{name, age}}(\text{Students})$

mn	name	age	email
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

$\sigma_{\text{age}>18}(\text{Students})$

name	age
Helen	20
Peter	22

Combination

Algebraic equivalence

- $\sigma_{\text{name}>18}(\pi_{\text{name,age}}(\text{Students}))$
- $\pi_{\text{name,age}}(\sigma_{\text{name}>18}(\text{Students}))$

Set operations

- There are *three* basic *set operations* in relational algebra
  - *Union*, *cross-product*, *difference*
- A fourth one (set *intersection*) can be expressed in terms of the first two
- *All* set operations are *binary*
- Essentially, they are the *well-known* set operations from *set theory*, but *extended* to deal with *tuples*

Set operator definitions

- Let R and S be two *relations*
- For *union*, *set difference* and *intersection*, R and S *must have compatible schemata*
  - Two schemata are *compatible* if they have the *same number of fields* and *corresponding fields* in a *left-to-right* order have the *same domains*
  - The *names* of the fields are *not used*
- The *union* $R \cup S$ of R and S is a new relation with the *same schema* as R and S
  - For naming purposes it is assumed that the output relation inherits the field names from the relation appearing first in the specification (R in the previous case)
- $R \cup S$ contains *tuples* appearing *either* in R *or* in S

Set operator definitions

- The *set difference* $R - S$ of R and S is a new relation with the *same schema* as R and S
  - Same naming conventions as in the union case apply
- $R - S$ contains *tuples* appearing *in* R *but not* in S
- The *intersection* $R \cap S$ of R and S is a new relation with the *same schema* as R and S
  - Same naming conventions as in the union and set difference cases apply
- $R \cap S$ contains *tuples* appearing *in both* R *and* S
- Intersection can be expressed in terms of set difference
  - $R \cap S = R - (R - S)$

Set operator definitions

- The *cross-product* (also known as the *Cartesian product*) $R \times S$ of two relations R and S is a new relation where
  - The *schema* of the relation is the *bag-union* of the *two schemata*
  - *Bag-union* means that *duplicate column names may appear*
  - The resulting relation contains *one tuple* $\langle r, s \rangle$ for *each pair* of tuples $r \in R$ and $s \in S$

Note

The two relations need not have the same schema to begin with

Set union example

mn	*name*	*age*	*email*
s0456782	John	18	john@inf
s0412375	Mary	18	mary@inf
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

S_1

mn	*name*	*age*	*email*
s0489967	Basil	19	basil@inf
s0412375	Mary	18	mary@inf
s9989232	Ophelia	24	oph@bio
s0189034	Peter	22	peter@math
s0289125	Michael	21	mike@geo

S_2

mn	*name*	*age*	*email*
s0456782	John	18	john@inf
s0412375	Mary	18	mary@inf
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math
s0489967	Basil	19	basil@inf
s9989232	Ophelia	24	oph@bio
s0289125	Michael	21	mike@geo

$S_1 \cup S_2$

Set intersection example

mn	*name*	*age*	*email*
s0456782	John	18	john@inf
s0412375	Mary	18	mary@inf
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

S_1

mn	*name*	*age*	*email*
s0489967	Basil	19	basil@inf
s0412375	Mary	18	mary@inf
s9989232	Ophelia	24	oph@bio
s0189034	Peter	22	peter@math
s0289125	Michael	21	mike@geo

S_2

mn	*name*	*age*	*email*
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

$S_1 \cap S_2$

Set difference example

mn	*name*	*age*	*email*
s0456782	John	18	john@inf
s0412375	Mary	18	mary@inf
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

S_1

mn	*name*	*age*	*email*
s0489967	Basil	19	basil@inf
s0412375	Mary	18	mary@inf
s9989232	Ophelia	24	oph@bio
s0189034	Peter	22	peter@math
s0289125	Michael	21	mike@geo

S_2

mn	*name*	*age*	*email*
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys

S_1-S_2

Cross-product example

mn	*name*	*age*	*email*
s0456782	John	18	john@inf
s0412375	Mary	18	mary@inf
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

S_1

code	*name*	*year*
inf1	Informatics 1	1
math1	Mathematics 1	1

R

mn	*name*	*age*	*email*	*code*	*name*	*year*
s0456782	John	18	john@inf	inf1	Informatics 1	1
s0456782	John	18	john@inf	math1	Mathematics 1	1
s0412375	Mary	18	mary@inf	inf1	Informatics 1	1
s0412375	Mary	18	mary@inf	math1	Mathematics 1	1
s0378435	Helen	20	helen@phys	inf1	Informatics 1	1
s0378435	Helen	20	helen@phys	math1	Mathematics 1	1
s0189034	Peter	22	peter@math	inf1	Informatics 1	1
s0189034	Peter	22	peter@math	math1	Mathematics 1	1

$S_1 \times R$

Renaming

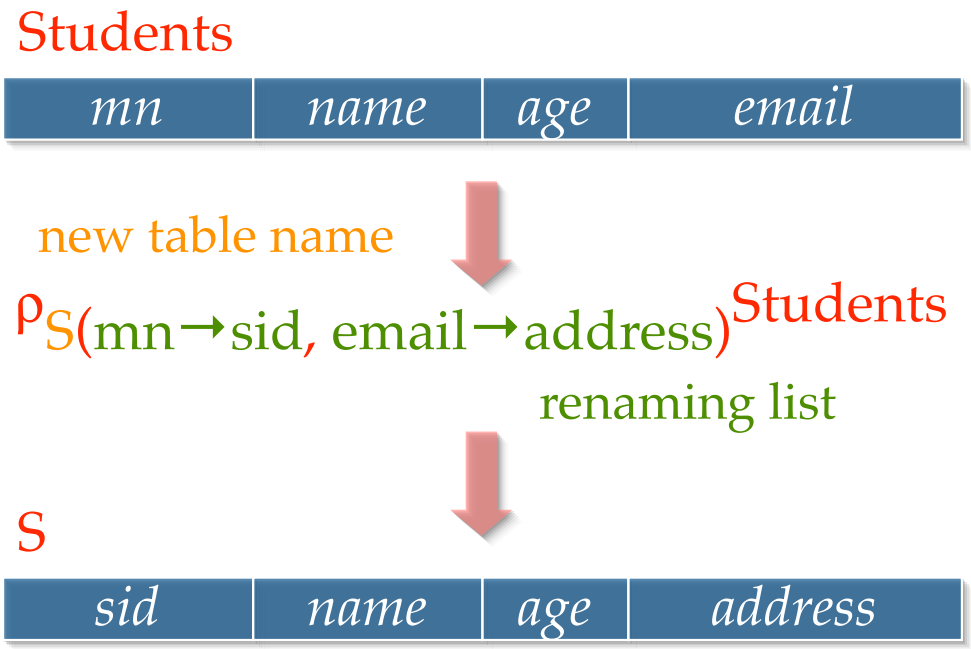
- *Problem*: certain *set operators* may lead to *duplicate column names* in the *resulting schemata*; this is known as a *naming conflict*
- *Solution*: introduce an *operator* that *changes* the *names* of tables and columns
- General form

$$\rho_{\text{New relation name}(\text{renaming list})}(\text{Original relation name})$$

with semantics

- The *original relation* is *assigned* the *new relation name*
- The *renaming list* consists of *terms* of the form oldname $\rightarrow$ newname which *rename* a filed named oldname to newname
- For ρ to be *well-defined* there should be *no conflicts* in the output

Renaming example



Note

- The *types* of the columns *do not change*
- *Either* the *renaming list*, *or* the *new table name* may be *empty*, but *not both*

Join — $\bowtie$

- The *relational join* is the *most frequently used* relational operator
  - However, there is *no need* to *explicitly define* it, *other than convenience*
- A *join* $R \bowtie_p S$ of two relations R and S is defined by use of *predicate* p
  - Predicate p (called the *join predicate*) is of the form col_1 `rop` col_2 where col_1 and col_2 are *columns of* R *or* S and `rop` $\in \{>, <, =, \neq, \geq, \leq\}$
- Formally, the relational join is defined as

$$R \bowtie_p S = \sigma_p(R \times S)$$

Join example

mn	*name*	*age*	*email*
s0456782	John	18	john@inf
s0412375	Mary	18	mary@inf
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

Students

mn	*code*	*mark*
s0412375	inf1	80
s0378435	math1	70

Takes

mn	*name*	*age*	*email*	*mn*	*code*	*mark*
s0456782	John	18	john@inf	s0412375	inf1	80
s0456782	John	18	john@inf	s0378435	math1	70
s0412375	Mary	18	mary@inf	s0412375	inf1	80
s0412375	Mary	18	mary@inf	s0378435	math1	70
s0378435	Helen	20	helen@phys	s0412375	inf1	80
s0378435	Helen	20	helen@phys	s0378435	math1	70
s0189034	Peter	22	peter@math	s0412375	inf1	80
s0189034	Peter	22	peter@math	s0378435	math1	70

$$Students \bowtie_{Students.mn = Takes.mn} Takes$$

Natural join

- A *special "flavour"* of the join operator for *attributes* having the *same name*
  - Commonly found in *key-foreign key* joins
- If the *predicate* is an *equality* predicate *on the two columns sharing a name*, then it *can be omitted*
  - The previous predicate as a natural join can be simply stated as Students $\bowtie$ Takes
  - In the sense, that this is the *"natural" way* of *joining* the two relations
  - (Because, in all probability, it has come from *mapping an* ER *diagram to a relational schema*!)

Outline

Tuple-relational calculus building blocks

- Another *way* of *querying* relational data
- Its *power* lies in the fact that it is *entirely declarative*
  - *Specify* the *properties* of the data we are *interested in retrieving*, but *not how to go about retrieving them*
- The *key concept* is a *tuple variable*
  - A *variable* that *ranges* over the *tuples* of a *relation instance*
- *Queries* are of the form

$$\{T \mid p(T)\}$$

  where T is a *tuple variable* and $p(T)$ a *first order logic formula* that evaluates to either *true* or *false*
- The *result* is an *instantiation* of all *tuples* $t \in T$ such that $p(t)$ evaluates to *true*

Tuple-relational calculus example

Example

- Find all students older than 18 years old
- $\{S \mid S \in \text{Students} \wedge S.\text{age} > 18\}$
  - Tuple variable S is introduced
  - Instantiated over all tuples in the Students table
  - Predicate $S.\text{age} > 18$ is evaluated on each indovidual tuple
  - *If and only if* the predicate evaluates to true, the tuple is propagated to the output

Formal syntax of atomic formulae

- *Rel* is a *relation name*, R and S are *tuple variables*, a is an *attribute of* R and b is an *attribute of* S
- Let op denote a *logical operator* in the set $\{>, <, =, \neq, \geq, \leq\}$
- An *atomic formula* is one of the following
  - $R \in \text{Rel}$
  - $R.a$ op $S.b$
  - $R.a$ op constant, or constant op $R.a$

Formal syntax of complex formulae

- A *formula* is *recursively defined*
- Let p and q be *formulae* and $p(R)$ denote a *formula* in which *tuple variable* R appears
  - Any *composite* formula
  - $\neg p$, $p \wedge q$, $p \vee q$, or $p \Rightarrow q$
  - $\exists R(p(R))$, where R is a tuple variable
  - $\forall R(p(R))$, where R is a tuple variable.
- It is *possible* to *implicitly declare* resulting *schemata* by *introducing tuple variables*

Example

- $\{P \mid \exists S \in \text{Students}(S.\text{age} > 20 \wedge P.\text{name} = S.\text{name} \wedge P.\text{age} = S.\text{age})\}$
- A *new relation* P with *attributes* name *and* age containing all *students older than 20*

Example 1

Example

Find the names of students who are taking Informatics 1

Relational algebra

$$\pi_{\text{Students.name}}(\text{Students} \bowtie_{\text{Students.mn}=\text{Takes.mn}} (\text{Takes} \bowtie_{\text{Takes.code}=\text{Courses.code}} (\sigma_{\text{name}=\text{'Informatics 1'}}(\text{Courses}))))$$

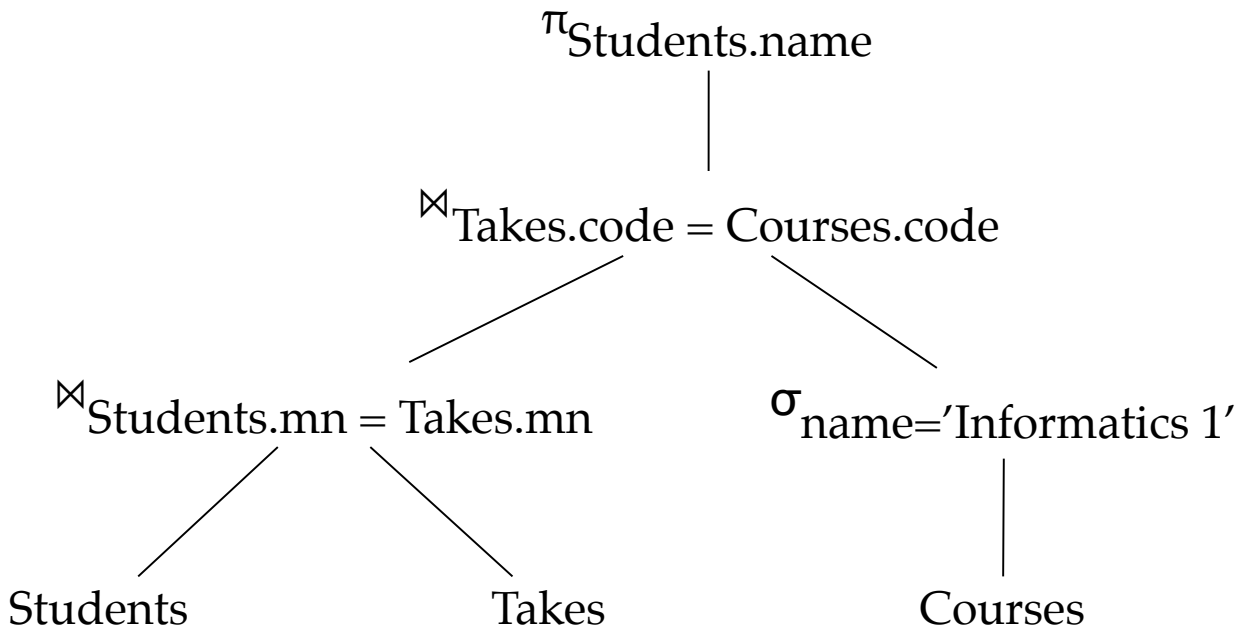
Tuple relational calculus

$$\{P \mid \exists S \in \text{Students}\ \exists T \in \text{Takes}\ \exists C \in \text{Courses}\ (C.\text{name} = \text{'Informatics 1'} \wedge C.\text{code} = T.\text{code} \wedge S.\text{mn} = T.\text{mn} \wedge P.\text{name} = S.\text{name})\}$$

Relational algebra — tree representation

Example

Find the names of students who are taking Informatics 1



Procedural representation

Relational algebra "instructs" how the query is to be evaluated

Example 2

Example

Find the names of all courses taken by Joe

Relational algebra

$$\pi_{\text{Courses.name}}((\sigma_{\text{name}='\text{Joe}'}(\text{Students})) \bowtie_{\text{Students.mn}=\text{Takes.mn}} (\text{Takes} \bowtie_{\text{Takes.code}=\text{Courses.code}} \text{Courses}))$$

Tuple-relational calculus

$$\{P \mid \exists S \in \text{Students}\ \exists T \in \text{Takes}\ \exists C \in \text{Courses}\ (S.\text{name} = \text{'Joe'} \wedge S.\text{mn} = T.\text{mn} \wedge C.\text{code} = T.\text{code} \wedge P.\text{name} = C.\text{name})\}$$

Example 3

Example

Find the names of all students who are taking Informatics 1 or Geology 1

Relational algebra

$$\pi_{\text{Students.name}}(\text{Students} \bowtie_{\text{Students.mn}=\text{Takes.mn}} (\text{Takes} \bowtie_{\text{Takes.code}=\text{Courses.code}} (\sigma_{\text{name}=\text{'Informatics 1'} \vee \text{name}=\text{'Geology 1'}}(\text{Courses}))))$$

Tuple-relational calculus

$$\{P \mid \exists S \in \text{Students}\ \exists T \in \text{Takes}\ \exists C \in \text{Courses}\ ((C.\text{name} = \text{'Informatics 1'} \vee C.\text{name} = \text{'Geology 1'}) \wedge C.\text{code} = T.\text{code} \wedge S.\text{mn} = T.\text{mn} \wedge P.\text{name} = S.\text{name})\}$$

Example 4

Example

Find the names of students who are taking both Informatics 1 and Geology 1

Relational algebra

$$\pi_{\text{Students.name}}($$
$$(\text{Students} \bowtie_{\text{Students.mn}=\text{Takes.mn}}$$
$$(\text{Takes} \bowtie_{\text{Takes.code}=\text{Courses.code}}$$
$$(\sigma_{\text{name}=\text{'Informatics 1'}}(\text{Courses})))) \cap$$
$$(\text{Students} \bowtie_{\text{Students.mn}=\text{Takes.mn}}$$
$$(\text{Takes} \bowtie_{\text{Takes.code}=\text{Courses.code}}$$
$$(\sigma_{\text{name}=\text{'Geology 1'}}(\text{Courses})))))$$

Example 4

Example

Find the names of students who are taking both Informatics 1 and Geology 1

Tuple-relational calculus

$$\{P \mid \exists S \in \text{Students} \ \wedge \ \forall C \in \text{Courses}$$
$$((\text{Cname} = \text{'Informatics 1'} \ \vee \ \text{Cname} = \text{'Geology 1'}) \Rightarrow$$
$$(\exists T \in \text{Takes}(T.\text{mn} = S.\text{mn} \ \wedge \ T.\text{code} = S.\text{code} \ \wedge$$
$$P.\text{name} = S.\text{name})))\}$$

Example 5

Example

Find the names of students who are taking all courses

Tuple-relational calculus

$$\{P \mid \exists S \in \text{Students}\ \forall C \in \text{Courses}$$
$$(\exists T \in \text{Takes}\ (C.\text{code} = T.\text{code} \wedge S.\text{mn} = T.\text{mn} \wedge$$
$$P.\text{name} = S.\text{name}))\}$$

Relational algebra and tuple-relational calculus

- *Relational algebra* and *tuple-relational calculus* have the *same expressive power*
  - That is, if a *query* can be *expressed* in *relational algebra*, it can be *expressed* in *relational calculus* and *vice-versa*
- The one is *procedural*, the other *declarative*
- This forms the basis of *query optimisation* in *relational databases*
  - Specify *what to retrieve* in *tuple-relational calculus*
  - Figure out the *best way to retrieve* it using *relational algebra*
  - *One of the most powerful frameworks in computer science*

Outline

Semi-structured data

- So far, we have dealt with *purely structured* data
  - In fact, we have spent most of our time identifying structure and semantic properties of the data and capturing both in various ways
- In other cases, *structure may not be so rigid*
- Sometimes it is a *good idea* to *mix structure* and *semantic properties* of data
- The *process* is usually called *marking up* the data and it *leads* to *semi-structured data*
  - The most dominant such paradigm is the *eXtensible Markup Language* — XML

XML and semi-structured data

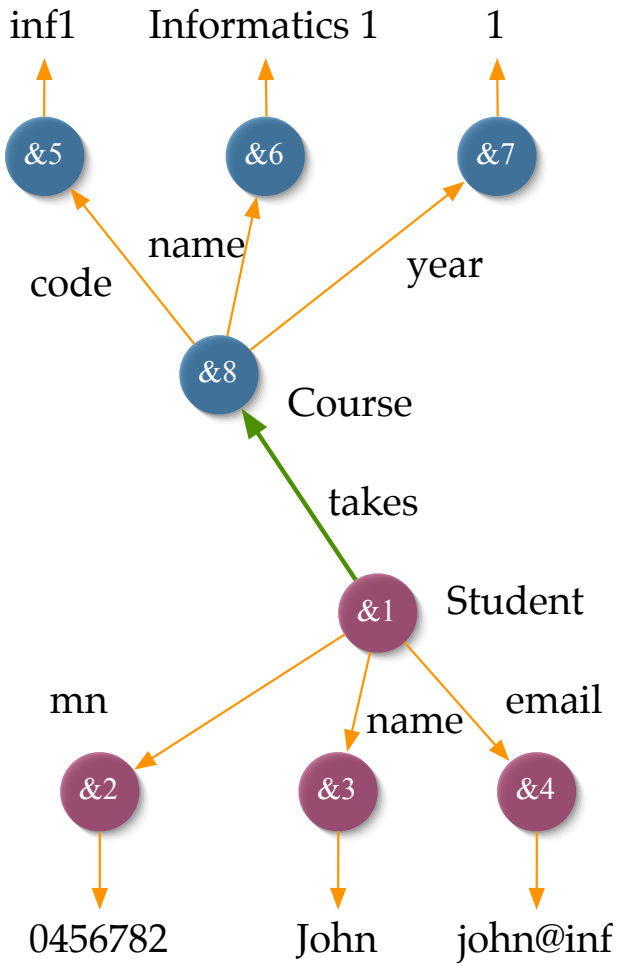
XML

```
<students>
  <student>
    <mn> s0456782 </mn> <name> John </name>
    <age> 18 </age> <email> john@inf </email>
  </student>
  <student>
    <mn> s0412375 </mn> <name> Mary </name>
    <age> 18 </age> <email> mary@inf </email>
  </student>
  <student>
    <mn> s0378435 </mn> <name> Helen </name>
    <age> 20 </age> <email> helen@phys </email>
  </student>
  <student>
    <mn> s0189034 </mn> <name> Peter </name>
    <age> 22 </age> <email> peter@math </email>
    <note> Has extension for first practical </note>
  </student>
</students>
```

- *Same information* as in the *Students table*
- The *data (values)* are *interspersed* with the *schema (markup)*
- The *schema* may *change*, *without* the rest of the *data changing*
- *Structure* is *not fixed*; *data* is *semi-structured*

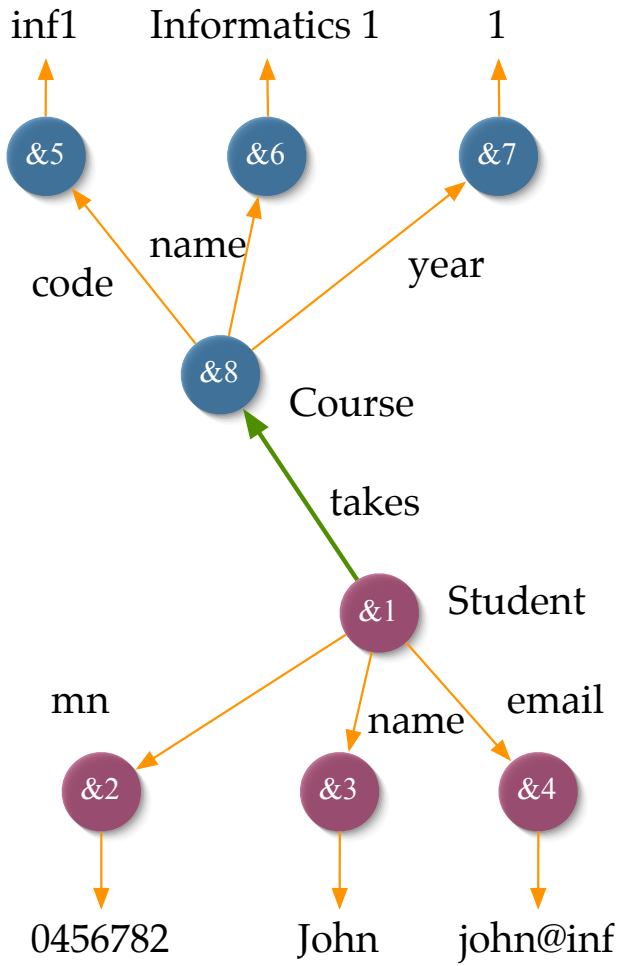
The XML graph representation

- One *node* for each *entity* and each *entity attribute*
- Each *node* has a *unique identifier*, so it can be directly referenced
- *Edges* denote a *relationship* between nodes
  - For instance, the *edge* from *Student* to *mn* means that each student has a matriculation number
  - The *edge* from *Student* to *Course* means that John takes Informatics 1



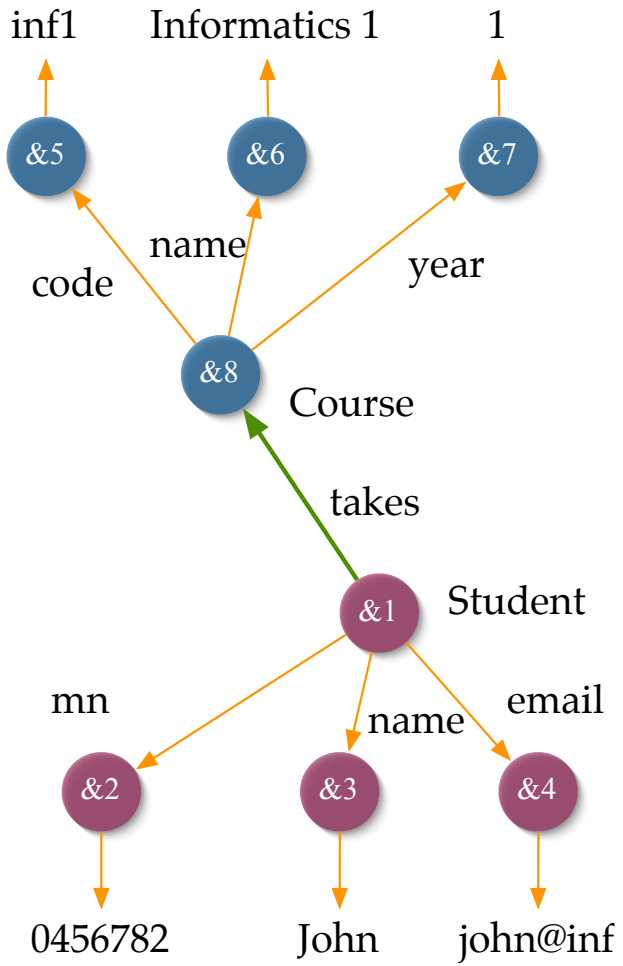
One possible representation

```
<student id="1"
   sid="#2" sname="#3" syear="#4"
   takes="#8">
</student>
<mn id="2">0456782</mn>
<name id="3">John</name>
<email id="4"> </email>
<course id="8"
   cid="#5" cname="#5" cyear="#7">
</course>
<code id="5">inf1</code>
<name id="6">
   Informatics 1
</name>
<year id="7">1</year>
```



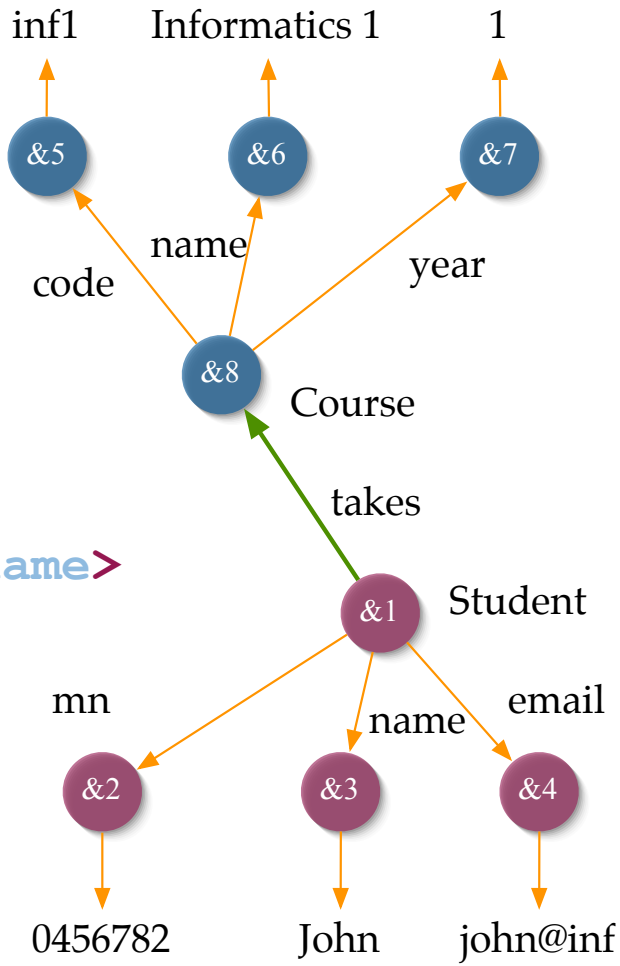
Another representation

```
<student id="1" takes="#8">
   <mn id="2">0456782</mn>
   <name id="3">John</name>
   <email id="4">john@inf</email>
</student>
<course id="8">
   <code id="5">inf1</code>
   <name id="6">Informatics 1</name>
   <year id="7">1</year>
</course>
```



A third way

```
<student id="1">
   <mn id="2">0456782</mn>
   <name id="3">John</name>
   <email id="4">john@inf</email>
   <takes>
      <course id="8">
         <code id="5">inf1</code>
         <name id="6">Informatics 1</name>
         <year id="7">1</year>
      </course>
   </takes>
</student>
```



...and finally

Congratulations!

You survived structured data!