Informatics 1: Data & Analysis

Lecture 3: The Relational Model

Ian Stark

School of Informatics
The University of Edinburgh

Tuesday 22 January 2019
Semester 2 Week 2

THE UNIVERSITY
of EDINBURGH

Lecture Plan for Weeks 1–4

Data Representation

This first course section starts by presenting two common data representation models.

- The *entity-relationship (ER)* model
- The *relational* model

Note slightly different naming:
-relation**ship** vs. relation**al**

Data Manipulation

This is followed by some methods for manipulating data in the relational model and using it to extract information.

- *Relational algebra*
- The *tuple-relational calculus*
- The query language *SQL*

Outline

1. Review of Entities and Relationships

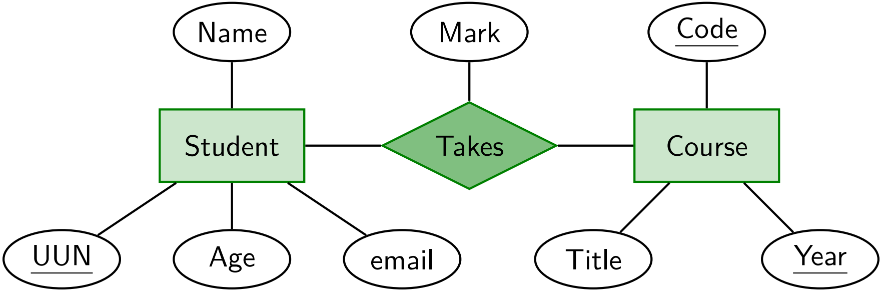
2. Constraints on Relationships

3. Basics of the Relational Model

Entity-Relationship Modelling

The Story so Far

- Requirements Analysis, Conceptual Design, Logical Design
- Entities: Entity Instances, Entity Sets; Attributes, Domains
- Keys: Superkeys, Candidate Keys, Primary Keys, Composite Keys
- Relationships: Relationship Instances, Relationship Sets ⟵ *I'll do that one again*



Relationships

A *relationship* is an association between entities. For example, the takes relationship between students and courses.

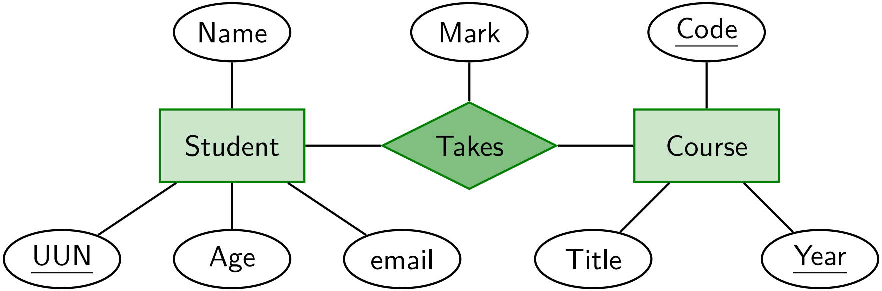
Each individual occurrence of the relationship is a *relationship instance*, and the collection of all such is a *relationship set*.

Relationship	Takes
Relationship instance	((s0456782), (INF08013, 2018))
Relationship set	Edinburgh course registrations

Relationships

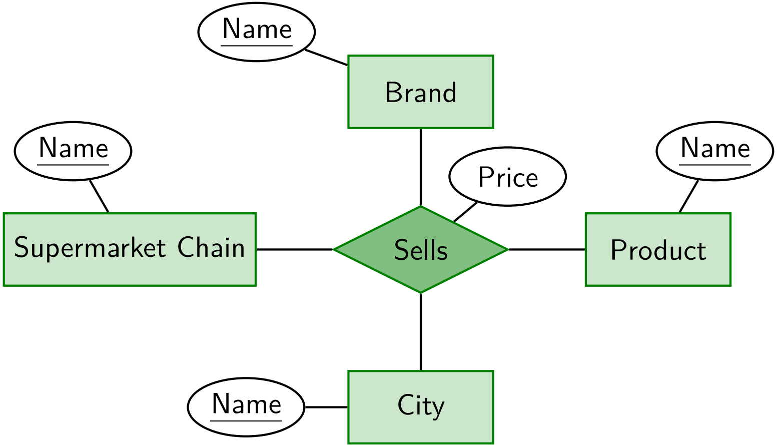
ER diagrams show relationships as diamonds, labelled with the name of the relationship and connected to all the participating entities.

A relationship may also have its own attributes.



Relationships

Relationships can be between two entities (“binary”), three (“ternary”) or more (“n-ary”).

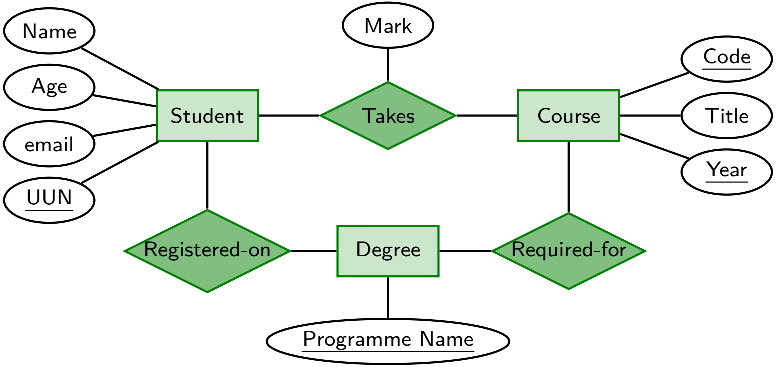


Relationship instance (ASDA, Heinz, Ketchup, Edinburgh, £2)

Relationships

There is no bound on the number of entities participating in a relationship.

An entity may be involved in any number of different relationships.



Summary

The entity-relationship (ER) model is a way to organise the description of *entities* (individual things) and the *relationships* between them.

So far we have seen the following elements of ER modelling:

- Entities, all with characteristic attributes;
- For each kind of entity there are entity instances that together make up an entity set;
- A key as a minimal set of attributes to identify and distinguish entity instances;
- Relationships between entities, which may be binary, ternary, or n-ary.

Entity-relationship modelling provides a *graphical language* for describing this structure in an ER diagram.

Outline

1 Review of Entities and Relationships

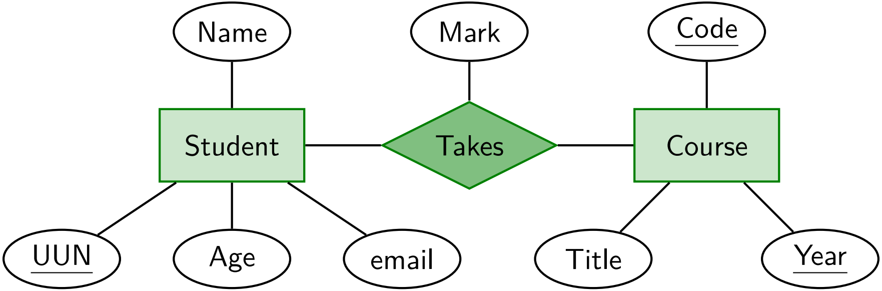
2 Constraints on Relationships

3 Basics of the Relational Model

Constraints on Relationships

In general, relationships need have no distinguished direction between the entities involved, nor limits on how many entity instances are related.

However, some relationships are more constrained.



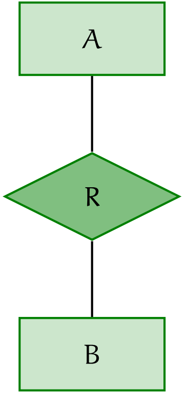
Constraints on Relationships

A binary relationship R between entities A and B can be:

Many-to-one: Several A may relate to a single B; but not the other way round.

One-to-many: Each A may relate to several B; but not the other way round.

Many-to-many: Any number of A may be related to any number of B.



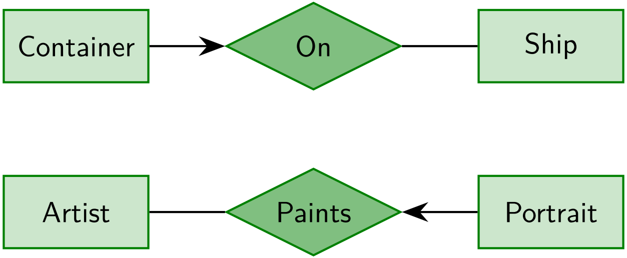
Examples?

Container On Ship; Artist Paints Portrait; Student Takes Course.

Key Constraints

An entity E has a *key constraint* in relationship R if each entity instance x of E can appear in at most one relationship instance $(x, y, z, \dots)$ from R.

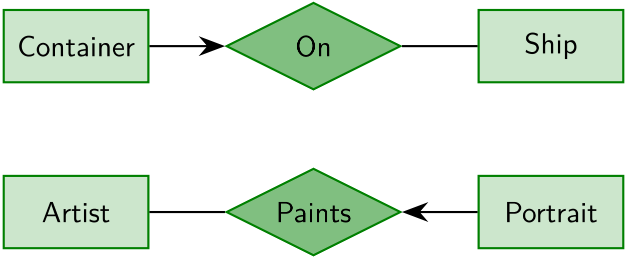
An ER diagram indicates a key constraint with an arrowhead on the line joining entity set E to relationship set R.



Key Constraints

Note

There is a subtlety here: a key constraint does not just report that entity instances happen to appear in no more than one relationship instance — instead, it asserts a structural property of the data that each entity instance cannot be in more than one relationship instance.



Participation Constraints

Total Participation of entity E in relationship R means that every entity instance x in E appears in at least one relationship instance of R.

Partial Participation of entity E in relationship R means that some entity instances in E might appear in no relationship instance from R.

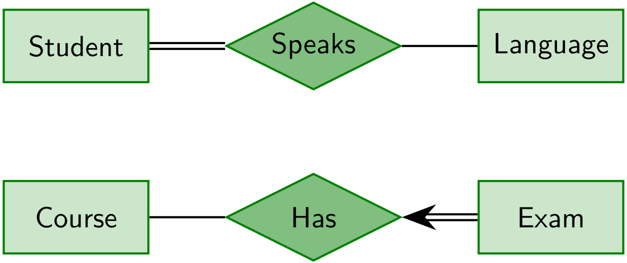
Examples?

Student Speaks Language; Country Uses Currency; Course Has Exam

Again this indicates not just chance but a structural constraint on the data model: each entity instance must appear in the relationship.

Participation Constraints

ER diagrams indicate total participation with a thick or double line from the entity to the relationship.



Again this indicates not just chance but a structural constraint on the data model: each entity instance *must* appear in the relationship.

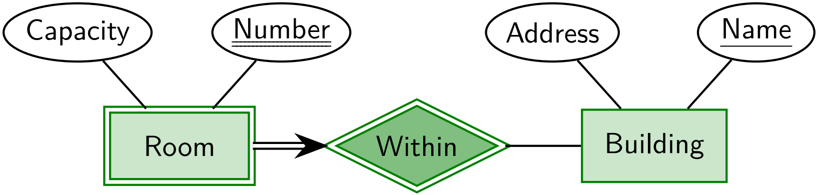
Weak Entities

Sometimes the attributes of an entity are not enough to build a key that uniquely identifies each individual.

We may be able to fix this using attributes from other, related, entities.

For this the entity must have a key constraint and total participation in some *identifying relationship*.

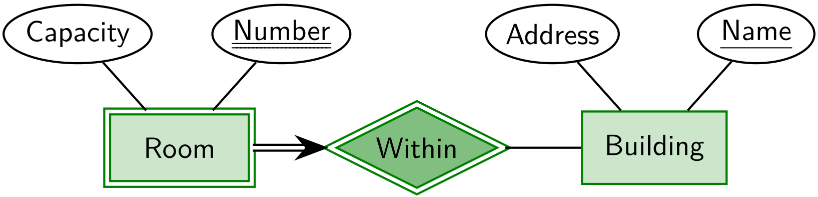
It is then a *weak entity*, with borrowed attributes coming from the *identifying owner*.



Weak Entities

ER diagrams represent weak entities by:

- A double or thick border on the rectangle of the weak entity;
- A double or thick border on the diamond of the identifying relationship;
- A double or thick line and arrow linking these;
- A double or dashed underline for the attributes of the weak entity that contribute to the composite key.

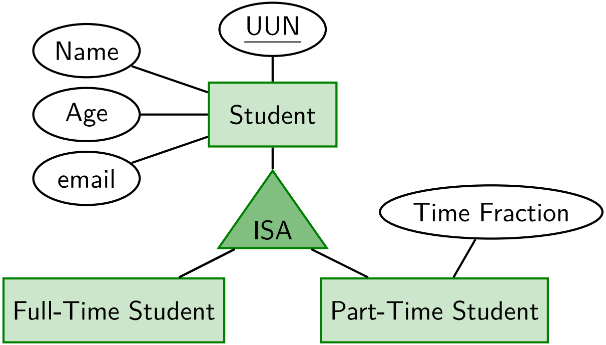


Entity Hierarchies and Inheritance

Sometimes one entity will refine another: a *subclass* entity *specializes* a *superclass* entity.

Each subclass entity *inherits* the attributes of the superclass entity, and may add its own attributes, too.

ER diagrams represent inheritance with an *IS-A* triangle.



Entity-Relationship Refinements

What We've Just Seen

- Relationships that are: One-to-Many, Many-to-One, Many-to-Many
- Key constraints
- Participation constraints: Total participation, Partial participation
- Weak entities with an identifying relationship and identifying owner
- Entity hierarchies and inheritance

Whitehead: Civilization advances by extending the number of important operations which we can perform without thinking about them.

Homework from Tuesday

Read This

Sections 2.1–2.4 of this textbook on databases: you will need to log in to *Learn* to access this chapter as a PDF.

R. Ramakrishnan and J. Gehrke.
Database Management Systems.
McGraw-Hill, third edition, 2003.

This is the recommended textbook for the third-year course Database Systems. It's a large book, with thorough and extensive material on a wide range of database topics.

It is *not* necessary to buy this book for Inf1-DA. Instead:

Do This

Step inside the library, locate the HUB and find this book. Have a look at the other textbooks there, too, and compare style and content.

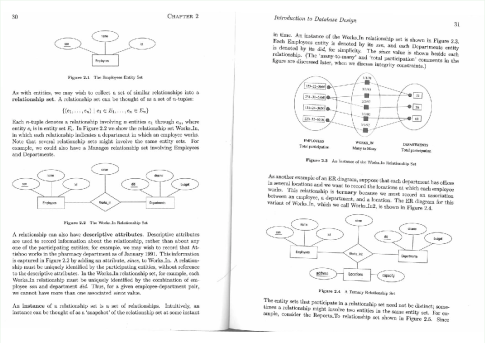
Finally: Find the video of *this lecture* online and watch the first minute. If you have any problems then ask on Piazza/Facebook.

Homework

Read This

Before the next lecture, on Friday, read the remaining sections, §§2.5 onwards, of Ramakrishnan and Gehrke, completing Chapter 2.

These consider trade-offs and choices in the design of Entity-Relationship models, as well as more on the wider context of modelling.



30 CHAPTER 2

Figure 2.1 The Employees Entity Set

As with entities, we may wish to collect a set of similar relationships into a **relationship set**. A relationship set can be thought of as a set of n-tuples:

$$\{(e_1, \ldots, e_n) \mid e_1 \in E_1, \ldots, e_n \in E_n\}$$

Each n-tuple denotes a relationship involving n entities e_1 through e_n, where entity e_i is in entity set E_i. In Figure 2.2 we show the relationship set Works_In, in which each relationship indicates a department in which an employee works. Note that several relationship sets might involve the same entity sets. For example, we could also have a Manages relationship set involving Employees and Departments.

Figure 2.2 The Works_In Relationship Set

A relationship can also have **descriptive attributes**. Descriptive attributes are used to record information about the relationship, rather than about any one of the participating entities; for example, we may wish to record that Attishoo works in the pharmacy department as of January 1991. This information is captured in Figure 2.2 by adding an attribute, *since*, to Works_In. A relationship must be uniquely identified by the participating entities, without reference to the descriptive attributes. In the Works_In relationship set, for example, each Works_In relationship must be uniquely identified by the combination of employee *ssn* and department *did*. Thus, for a given employee-department pair, we cannot have more than one associated *since* value.

An **instance** of a relationship set is a set of relationships. Intuitively, an instance can be thought of as a 'snapshot' of the relationship set at some instant

Introduction to Database Design 31

in time. An instance of the Works_In relationship set is shown in Figure 2.3. Each Employees entity is denoted by its *ssn*, and each Departments entity is denoted by its *did*, for simplicity. The *since* value is shown beside each relationship. (The 'many-to-many' and 'total participation' comments in the figure are discussed later, when we discuss integrity constraints.)

Figure 2.3 An Instance of the Works_In Relationship Set

As another example of an ER diagram, suppose that each department has offices in several locations and we want to record the locations at which each employee works. This relationship is **ternary** because we must record an association between an employee, a department, and a location. The ER diagram for this variant of Works_In, which we call Works_In2, is shown in Figure 2.4.

Figure 2.4 A Ternary Relationship Set

The entity sets that participate in a relationship set need not be distinct; sometimes a relationship might involve two entities in the same entity set. For example, consider the Reports_To relationship set shown in Figure 2.5. Since

R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, third edition, 2003.

One Idea, Many Variations

+

How to Succeed at Inf1-DA !

- Be there at every lecture
- Take notes, paper or electronic (slides will be available online and as paper handouts)
- Write up your notes after the lecture: rearrange, reflect, summarise
- Do the homework: read this, do that, watch these
- Work through the exercises before each tutorial
- Be there at every tutorial: take part, work together with others
- After each tutorial, do some of the additional example questions
- If you have questions or problems then ask for help: on Piazza or Facebook; other students, tutors, me; in person, by email, after lectures.

Tutorials and Exercise Sheets !

Tutorials start in **Week 3** of semester and and continue each week until the end of semester, except for during the *Festival of Creative Learning* week which falls between lecturing weeks 5 and 6.

Your *course tutor* leads your tutorial group; this is not the same as your *personal tutor*.

You will be able to choose your own tutorial group during Week 2: all tutorials will be one-hour time-slots on Monday, Tuesday or Wednesday.

Studying for Inf-DA requires you to work through exercise sheets and then take part in your weekly tutorials. If you are ill or otherwise unable to attend one week then email your course tutor, and if possible attend another tutorial group in the same week.

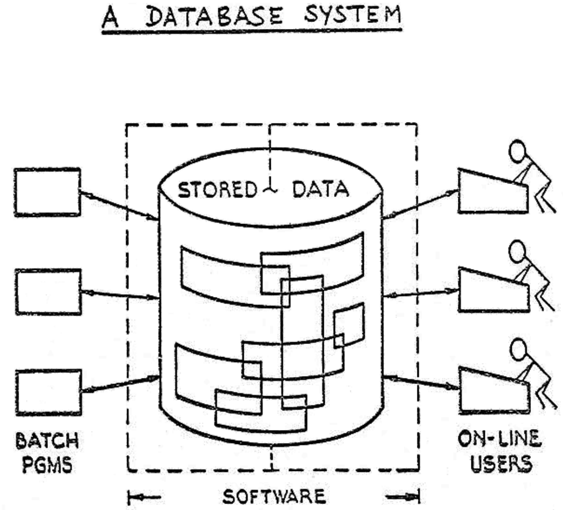
Outline

1. Review of Entities and Relationships

2. Constraints on Relationships

3. Basics of the Relational Model

History of the Relational Model



History of the Relational Model

The *relational model* was introduced in 1970 by Edgar F. Codd, a British computer scientist working for IBM in California.

A key insight, and a reason for the success of the relational model, is its separation between *specification* (what you want to find out) and *implementation* (how the system should find it out for you).

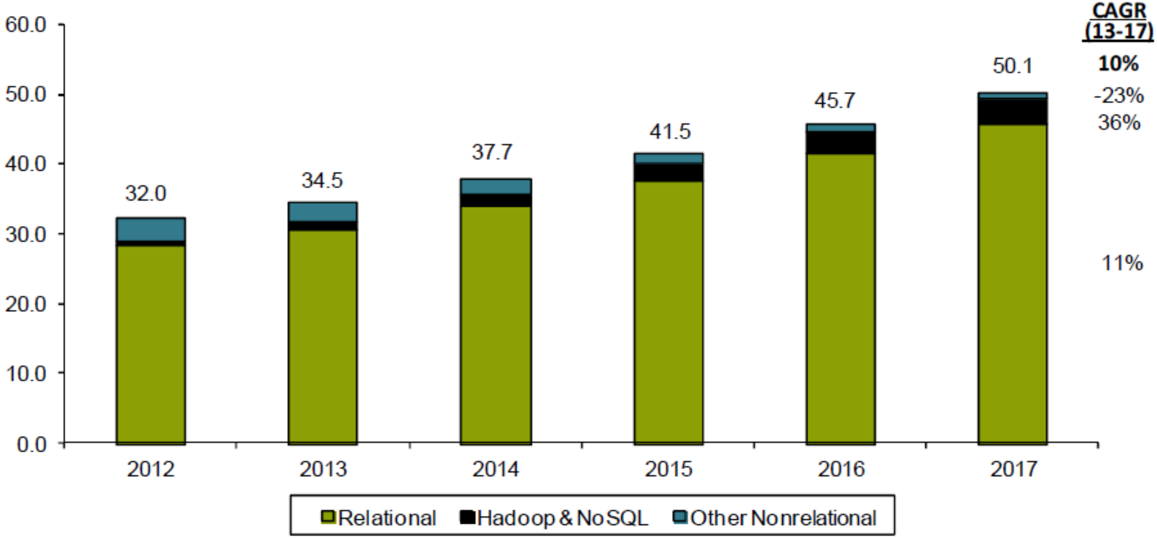
IBM were initially reluctant to exploit Codd's idea, but did in due course develop the System R database platform which led the commercial development of *relational database management systems* (RDBMS).

System R included the SEQUEL language, which then became SQL and the standard query language for all subsequent relational databases.

Relational database management systems are now a very large industry: G$38 market size in 2014, and growing.

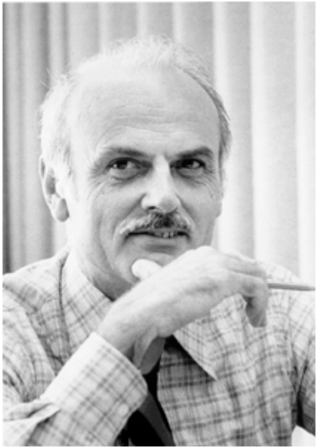
Codd received the 1981 Turing Award for his work on databases.

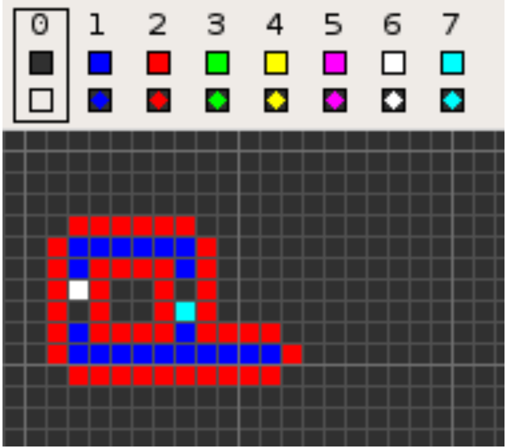
Global Database Market ($B)



Source: IDC, Bernstein analysis

Codd

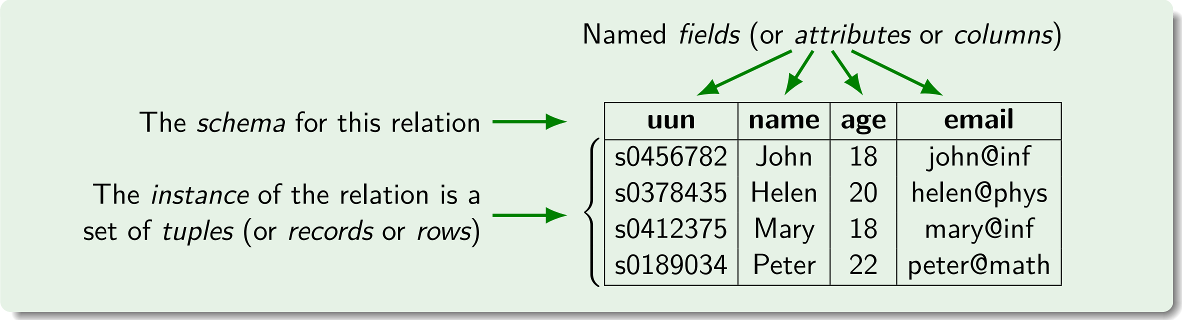




Credits: IBM, Ferkel

The Relational Model

Relational databases take as fundamental the idea of a *relation*, comprising a *schema* and an *instance*.



Named *fields* (or *attributes* or *columns*)

The *schema* for this relation

The *instance* of the relation is a set of *tuples* (or *records* or *rows*)

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

Absolutely everything in a relational database is built from relations and operations upon them.

The Relational Model

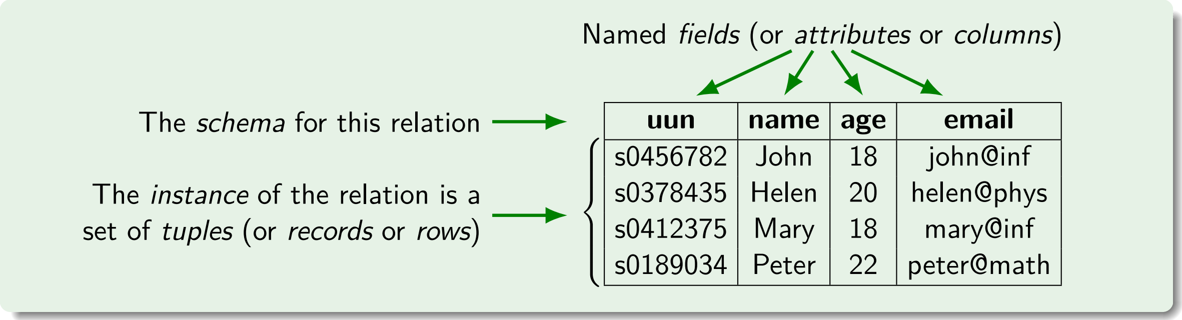
Relational databases take as fundamental the idea of a *relation*, comprising a *schema* and an *instance*.

- The schema is the format of the relation:
  - A set of named *fields* (or *attributes* or *columns*)
  - For each field its *domain* (or *type*)
- The instance of a relation is a *table*:
  - A set of *rows* (or *records* or *tuples*)
  - Each row gives a value for every field, from the appropriate domain.
- The *arity* of a relation is the number of fields in its schema.
- The *cardinality* of a relation is the number of rows in its table.

Absolutely everything in a relational database is built from relations and operations upon them.

Example

Relational databases take as fundamental the idea of a *relation*, comprising a *schema* and an *instance*.



Named *fields* (or *attributes* or *columns*)

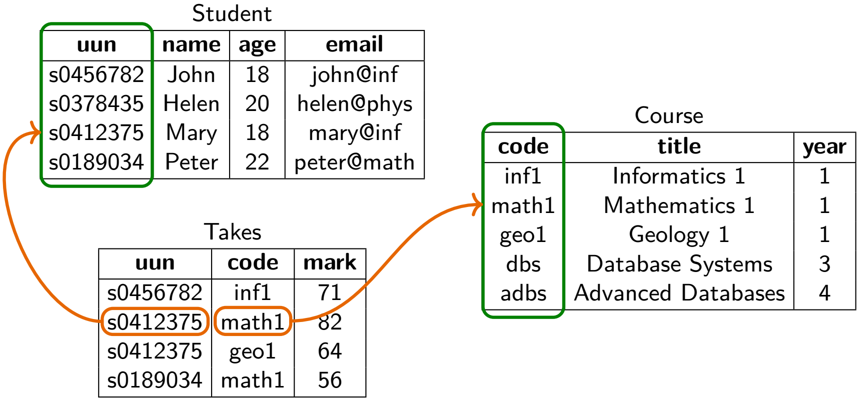
The *schema* for this relation

The *instance* of the relation is a set of *tuples* (or *records* or *rows*)

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

Every relational database is a linked collection of several tables like this: often much wider, and sometimes very, very much longer.

Some Linked Relations



Student

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

Takes

uun	code	mark
s0456782	inf1	71
s0412375	math1	82
s0412375	geo1	64
s0189034	math1	56

Course

code	title	year
inf1	Informatics 1	1
math1	Mathematics 1	1
geo1	Geology 1	1
dbs	Database Systems	3
adbs	Advanced Databases	4

- Primary Key
- Foreign Key

SQL: Structured Query Language

- SQL is the standard language for working with relational database management systems.
- Substantial parts of SQL are declarative: code states what should be done, not necessarily how to do it.
- When actually querying a large database, database systems take advantage of this to plan, rearrange, and optimize the execution of queries.
- Procedural parts of SQL do contain imperative code to make changes to the database.
- While SQL is an international standard (ISO 9075), individual implementations have notable idiosyncrasies, and code may not be entirely portable.

There is also lots of activity in *NoSQL* databases; these are interesting, too, but we won't be doing too much on them right now. As it happens, most of what is on this slide applies to NoSQL approaches as well.

DDL: SQL Data Definition Language

The *Data Definition Language* is a portion of SQL used to declare the schemas for relations.

In particular the DDL contains the imperative command **CREATE TABLE** which sets up a fresh, empty, table with a certain schema.

For simplicity, we'll use schemas with only three types:

- INTEGER for integer values;
- FLOAT for floating point real-valued numbers;
- VARCHAR(n) for strings of length up to n.

It's conventional, although not at all universal, to write SQL keywords in UPPER CASE, with lower case or Mixed Case for identifiers. This isn't enforced by the language.

There are some moderately complex rules about use of quotation marks around identifiers and strings, which I'll talk about another time.

DDL Example

```
CREATE TABLE Student (
    uun    VARCHAR(8),
    name   VARCHAR(20),
    age    INTEGER,
    email  VARCHAR(25),
    PRIMARY KEY (uun) )
```

Student

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

The general form of this statement is

CREATE TABLE *table-name* (⟨*attribute declarations*⟩ ⟨*integrity constraints*⟩)

with a body containing a comma-separated list of *attribute-name attribute-type* pairs and a series of statements that present constraints on the data in the table.

DDL Example

```
CREATE TABLE Student (
    uun    VARCHAR(8),
    name   VARCHAR(20),
    age    INTEGER,
    email  VARCHAR(25),
    PRIMARY KEY (uun) )
```

Student

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

The final line declares a *primary key constraint*, that the single field uun is the primary key for the Student table.

This states that no two rows in the Student table can share the same value for uun.

The constraint is enforced by the system: any attempt to insert a new row that duplicates an existing uun value will fail.

DDL Example

```
CREATE TABLE Takes (
    uun     VARCHAR(8),
    code    VARCHAR(20),
    mark    INTEGER,
    PRIMARY KEY (uun, code),
    FOREIGN KEY (uun)    REFERENCES Student,
    FOREIGN KEY (code)   REFERENCES Course )
```

Takes

uun	code	mark
s0456782	inf1	71
s0412375	math1	82
s0412375	geo1	64
s0189034	math1	56

This relation has a *composite key* with both uun and code needed to identify a Takes record.

Additional *foreign key constraints* declare that uun will always take a value that appears in the Student table, and that code will always be from the Course table.

These constraints are enforced by the system and no rows can be added that do not satisfy all of them.

From ER to Relational Models

Entity-relationship modelling gives a high-level conceptual design for a database system, showing what things are to be recorded and how they are connected.

The relational model supports a more explicit logical design, closer to implementation.

Some work is required to move between them: notice, for example, that in the ER model Student would have been an entity with attributes, and Takes a relationship between entities, while in the relational model both are implemented as tables.

In the next lecture we shall look at how to systematically transform an ER model into a relational one.