Informatics 1: Data & Analysis

Lecture 3: The Relational Model

Ian Stark

School of Informatics
The University of Edinburgh

Tuesday 21 January 2014
Semester 2 Week 2

BE THE CHANGE. JOIN FRESHSIGHT

Join Edinburgh's unique student-led consultancy organisation.

We train and empower students of all disciplines to realise their talents as consultants for real-life charities.

Apply now at **www.freshsight.com** by **23 Jan, 5pm**

Find out more at our information sessions:
17th & 20th January
M2A Appleton Tower, 6pm

www.freshsight.com
info@freshsight.com
www.facebook.com/freshsightedinburgh

FRESHSIGHT

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*

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

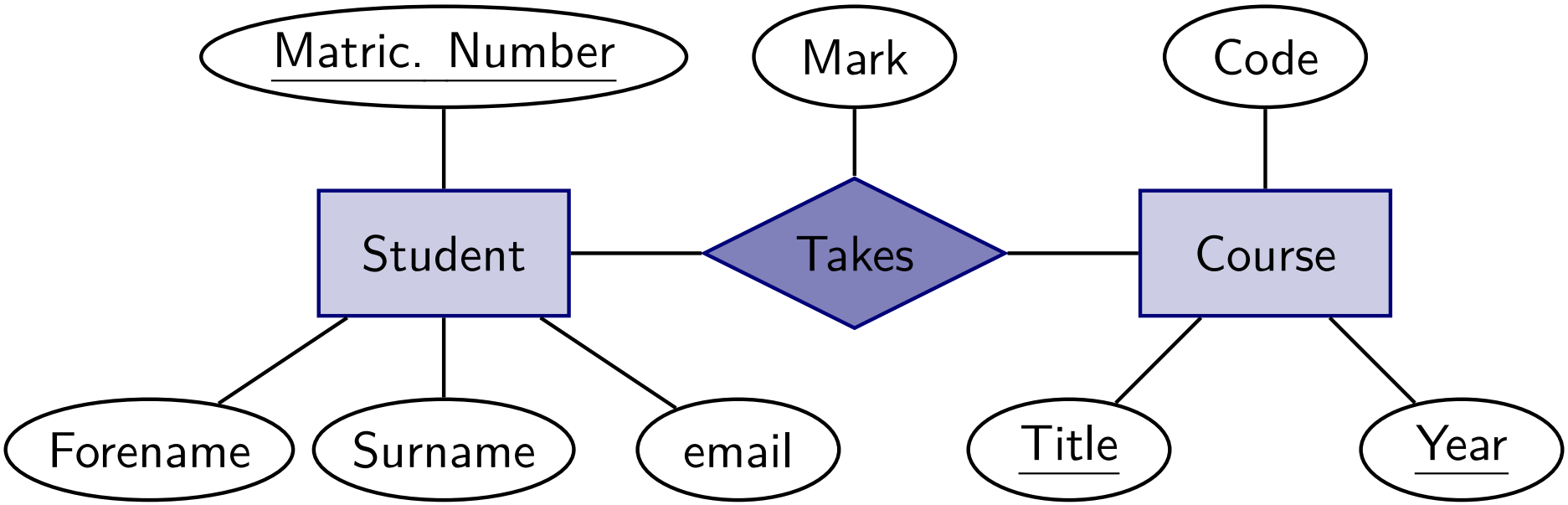
This is followed by some methods for manipulating data in the relational model and using it to extract information.

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

Entity-Relationship Modelling

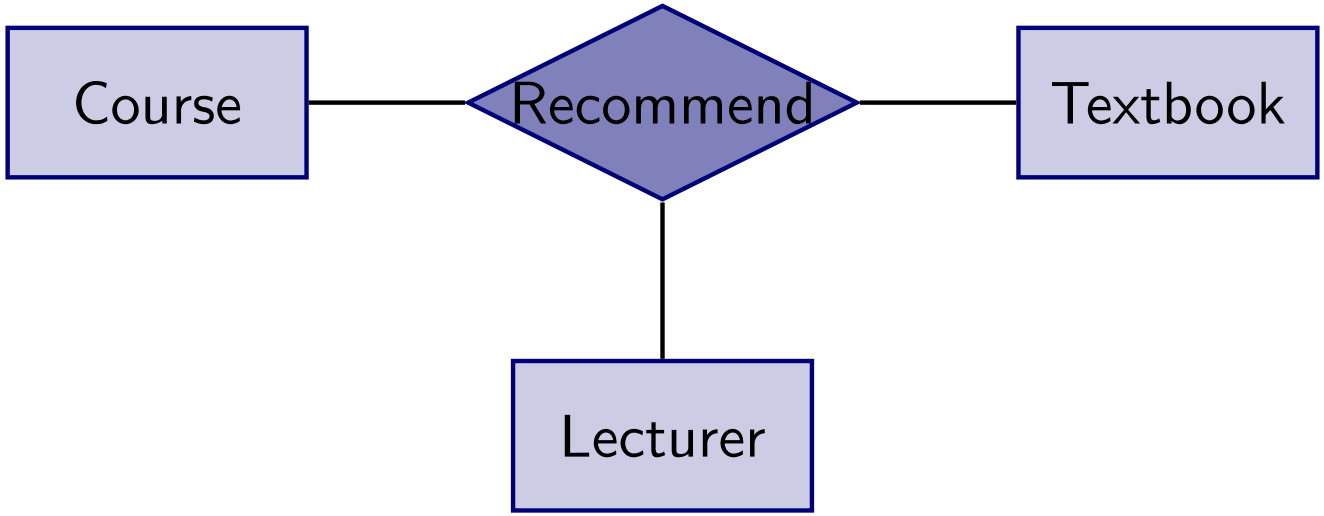
The Story so Far

- Requirements Analysis; Conceptual Design; Logical Design
- Entities: Entity Instances, Entity Sets
- Attributes, Domains
- Keys: Candidate Keys, Primary Keys, Composite Keys
- Relationships: Relationship Instances, Relationship Sets



Classifying Relationships

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



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.

Classifying Relationships

A binary relationship set R between entity sets 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?

City In Country; Engine Pulls Carriage; Student Takes Course.

Key Constraints

We can extend the idea of many-to-one beyond binary relationships.

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

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

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 set E in relationship set R means that every entity x in E appears in at least one relationship instance from 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

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

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

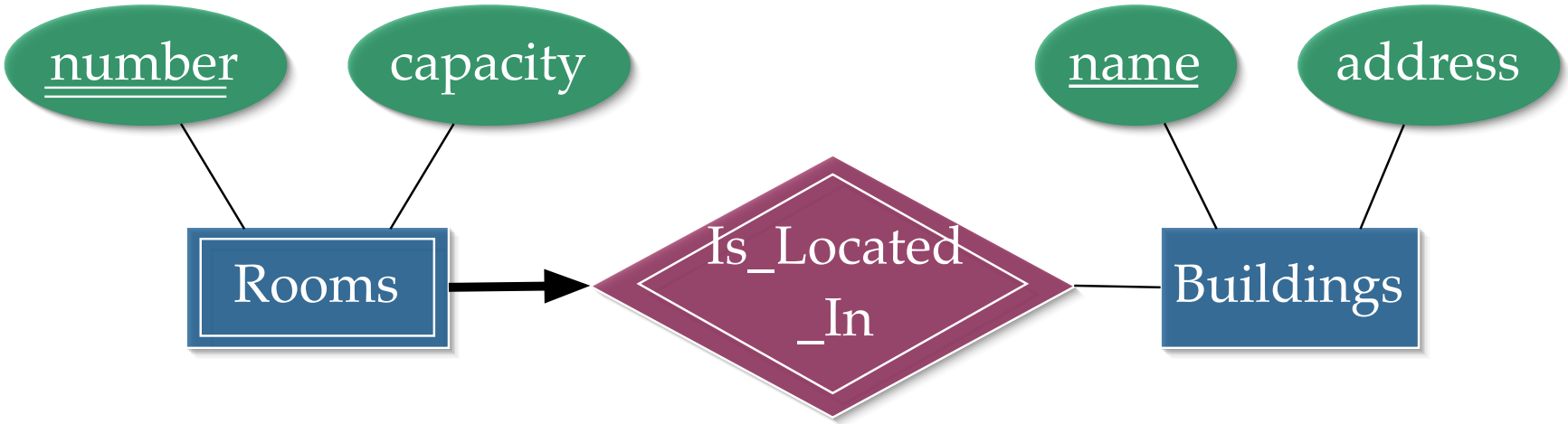
Weak Entity Sets

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

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

This can only work if the entity set has a key constraint and total participation in the *identifying relationship*.

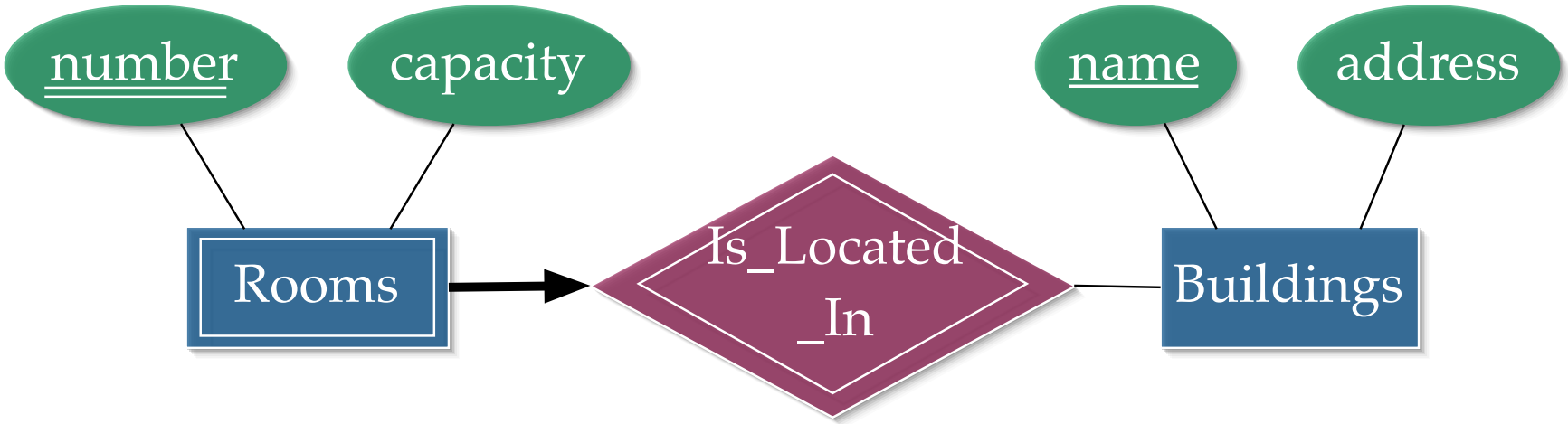
This is then a *weak entity set*, with borrowed attributes coming from the uniquely-defined *identifying owner*.



Weak Entity Sets

ER diagrams represent weak entity sets by:

- A double or thick border on the rectangle of the weak entity set;
- 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 set that contribute to the composite key.

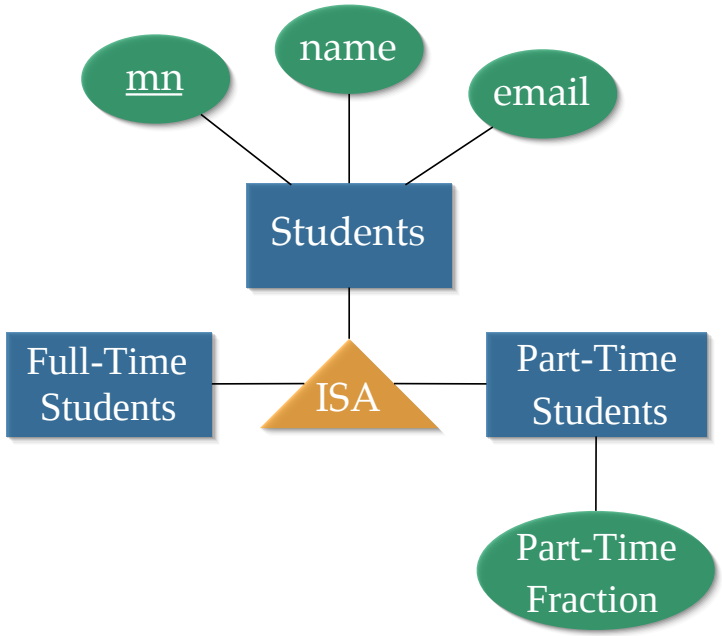


Entity Hierarchies and Inheritance

Sometimes one or more kinds of 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

- One-to-Many relationships, Many-to-One, Many-to-Many
- Key constraints
- Participation constraints: Total participation, Partial participation
- Weak entities, Identifying relationship, Identifying owner
- Entity hierarchies, Inheritance

Entity-Relationship Refinements

What We've Just Seen

- One-to-Many relationships, Many-to-One, Many-to-Many
- Key constraints
- Participation constraints: Total participation, Partial participation
- Weak entities, Identifying relationship, Identifying owner
- Entity hierarchies, Inheritance

But why?

Entity-Relationship Refinements

What We've Just Seen

- One-to-Many relationships, Many-to-One, Many-to-Many
- Key constraints
- Participation constraints: Total participation, Partial participation
- Weak entities, Identifying relationship, Identifying owner
- Entity hierarchies, Inheritance

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

Entity-Relationship Refinements

What We've Just Seen

- One-to-Many relationships, Many-to-One, Many-to-Many
- Key constraints
- Participation constraints: Total participation, Partial participation
- Weak entities, Identifying relationship, Identifying owner
- Entity hierarchies, Inheritance

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

Operations of thought are like cavalry charges in a battle — they are strictly limited in number, they require fresh horses, and must only be made at decisive moments. Introduction to Mathematics, 1911

Homework

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.

The first tutorial sheet will be out by tomorrow — look for announcement on the course blog and mailing list. It's about designing an entity-relationship model, and you should start work on it this week.

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

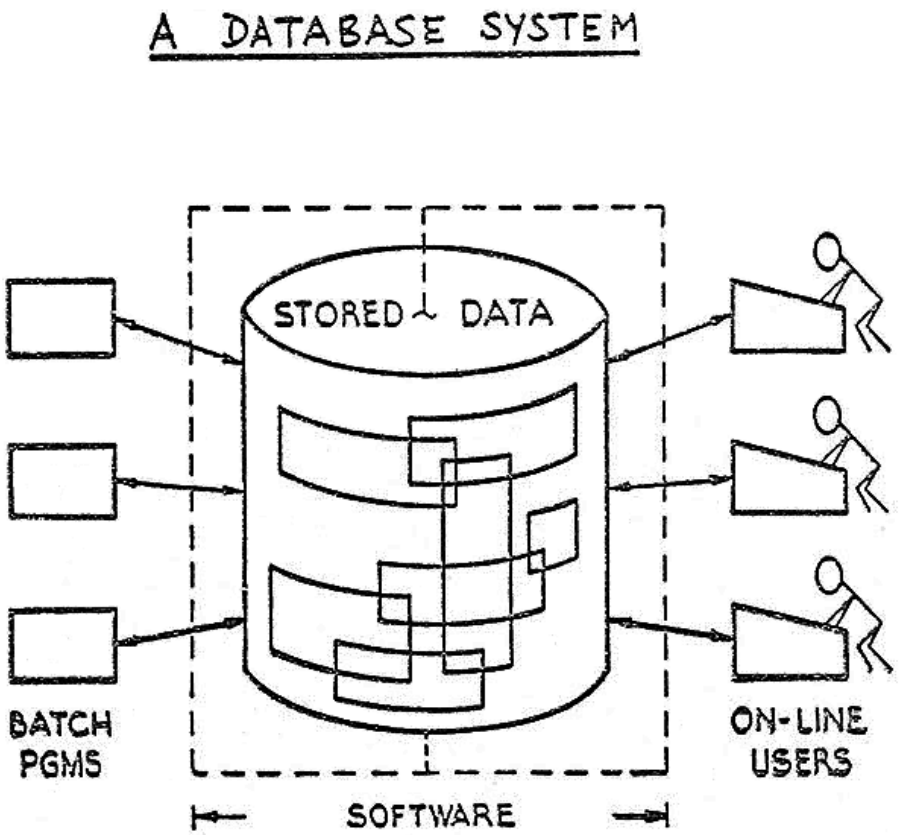
Note slightly different naming:
-relation**ship** vs. relation**al**

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*

History of the Relational Model



IBM's 100 Icons of Progress: The Relational Database

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).

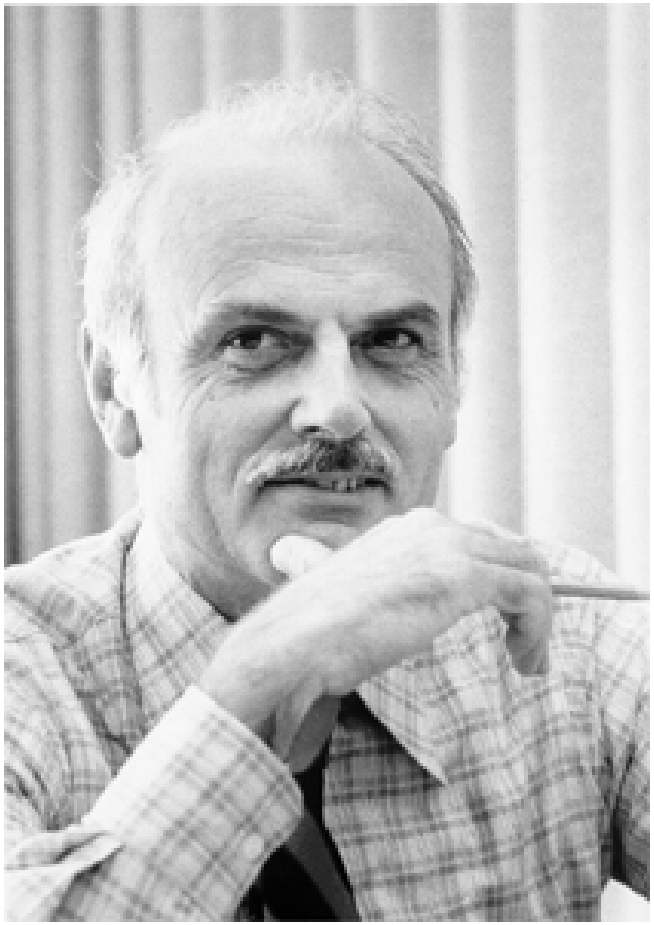
IBM were initially reluctant to exploit Codd's idea, but did in due course develop the System R database platform which led the commercial development of *relational database management systems* (RDBMS).

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$36 market size in 2011, and growing.

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

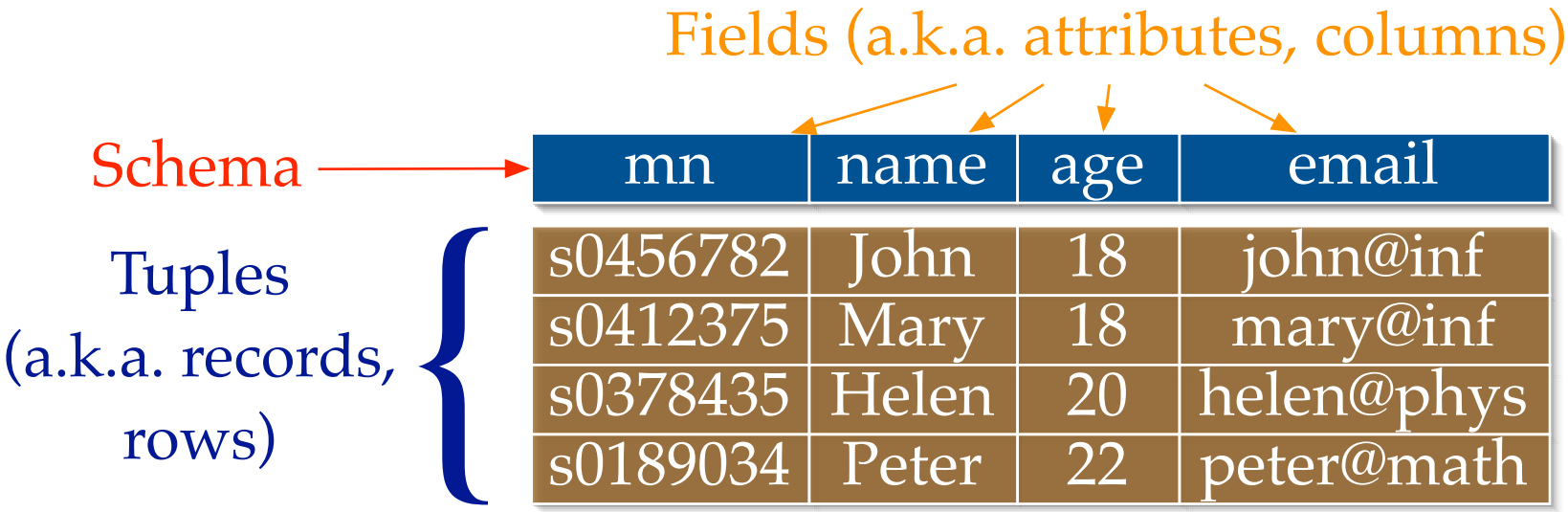
Codd



Credits: IBM, Ferkel

The Relational Model

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



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

The Relational Model

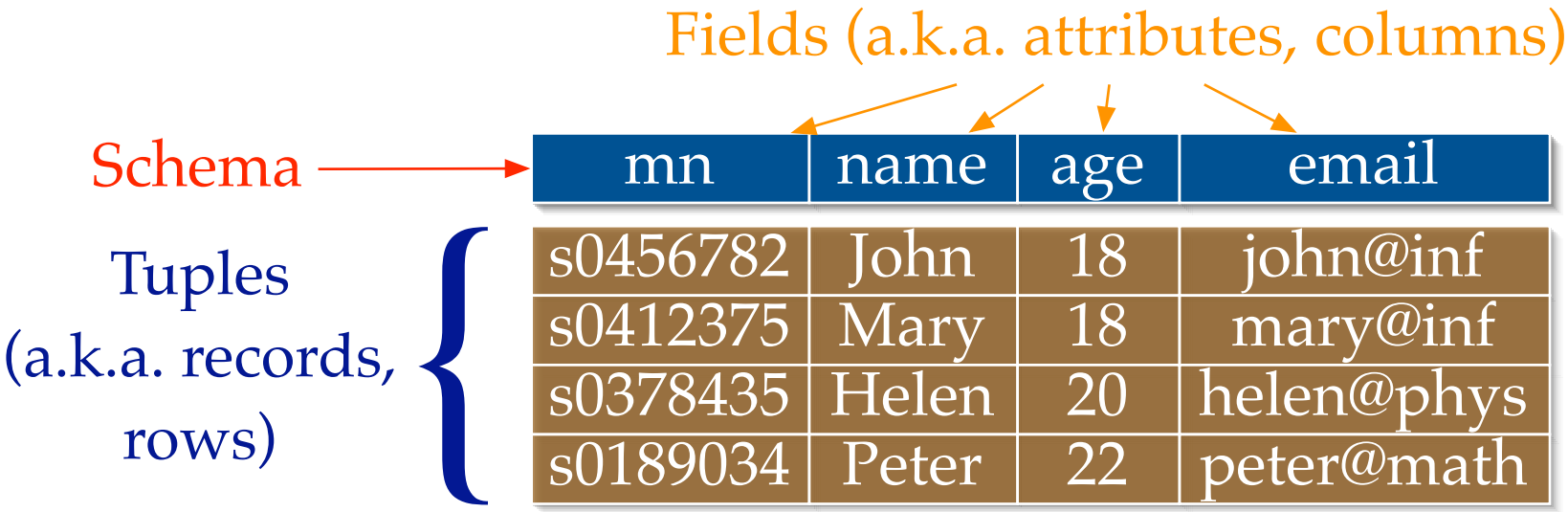
Relational databases take as fundamental the idea of a *relation*, comprising a *schema* and an *instance*.

- 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

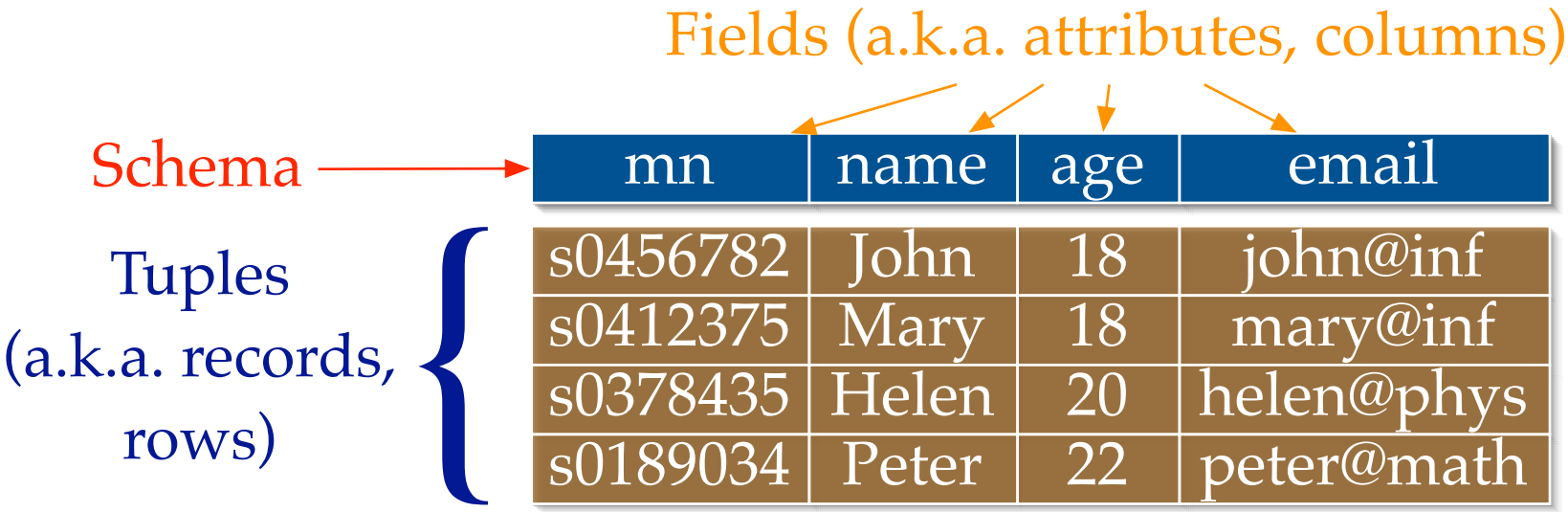
Relational databases take as fundamental the idea of a *relation*, comprising a *schema* and an *instance*.



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*.



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

SQL: Structured Query Language

- SQL is the standard language for interacting 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 some other time.

DDL Example

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

The general form of this statement is

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

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

DDL Example

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

The final line declares a *primary key constraint*, that matric is the primary key for the students table.

This states that no two rows in the students table can share the same value for matric.

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

DDL Example

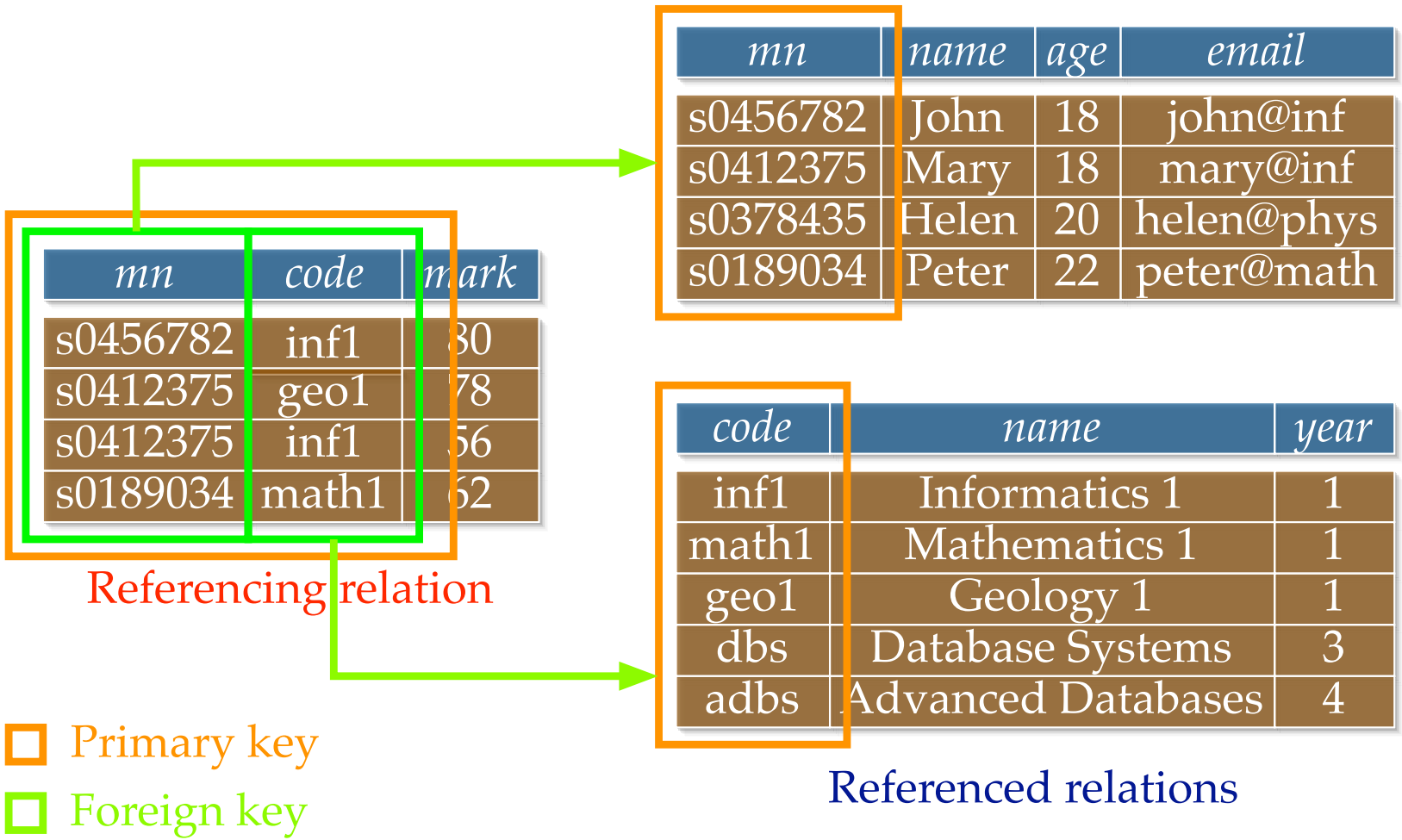
```
CREATE TABLE Takes (
    matric  VARCHAR(8),
    code    VARCHAR(20),
    mark    INTEGER,
    PRIMARY KEY (matric, code),
    FOREIGN KEY (matric) REFERENCES students,
    FOREIGN KEY (code) REFERENCES courses )
```

This relation has a *composite key* with both matric and code needed to identify a record in the table.

Additional *foreign key constraints* declare that matric will always take a value that appears in the students table, and code will always be from the courses table.

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

Constraints from DDL



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 Students 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.