Part I — Structured Data

Data Representation:

I.1 The entity-relationship (ER) data model

I.2 The relational model

Data Manipulation:

I.3 Relational algebra

I.4 Tuple relational calculus

I.5 The SQL query language

Required reading: Chapter 3 of [DMS], §§ 3.1,3.2,3.4,3.5

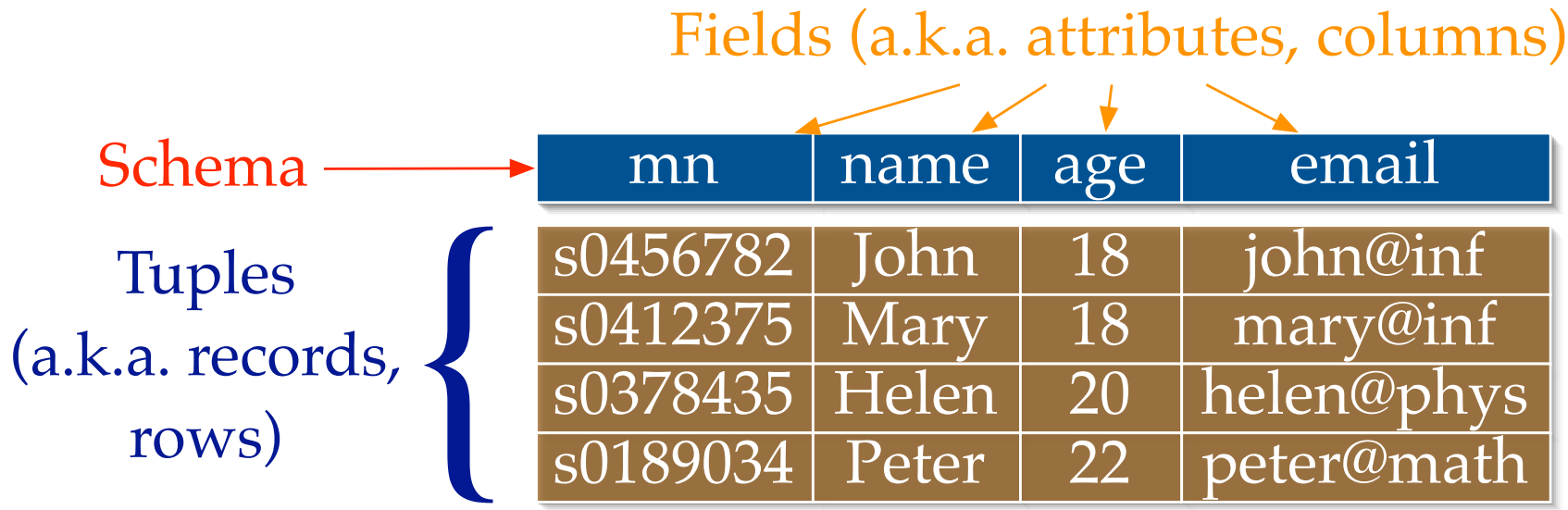
History of relational model

- The *relational model* was introduced in 1970 by Edgar F. Codd, a British computer scientist working at IBM's Almaden Research Center in San Jose, California.
- IBM was initially slow to exploit the idea, but by the mid 1970's IBM was at the forefront of the commercial development of relational database systems with its System R project, which included the development and first implementation of SQL. (Codd was sidelined from this project!)
- Around the same time, the relational model was developed and implemented at UC Berkeley (the Ingres project)
- Nowadays relational databases are a multi-billion pound industry.
- A major reason for the success of the relational model is its simplicity
- Codd received the 1981 *Turing Award* for his pioneering work on relational databases

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 from the specified domain 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

SQL: The 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.

Data definition in SQL

- 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 (floating point), and (iii) **`char(`**n**`)`** for a string of maximum length n
  - There is also a special **`null`** value, which we see briefly later.

General form of a DDL statement

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

Example 1

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

The example defines the **Students** table.

The last line implements a *primary key constraint*, it declares **mn** to be the chosen primary key for **Students**.

This constraint requires that the **Students** table contains at most one row with any given **mn** value. This is enforced by the system.

Any attempt to insert a new row with an **mn** value that already exists in some other row of the table will fail.

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

General form of a DDL statement

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

Example 2

```
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 )
```

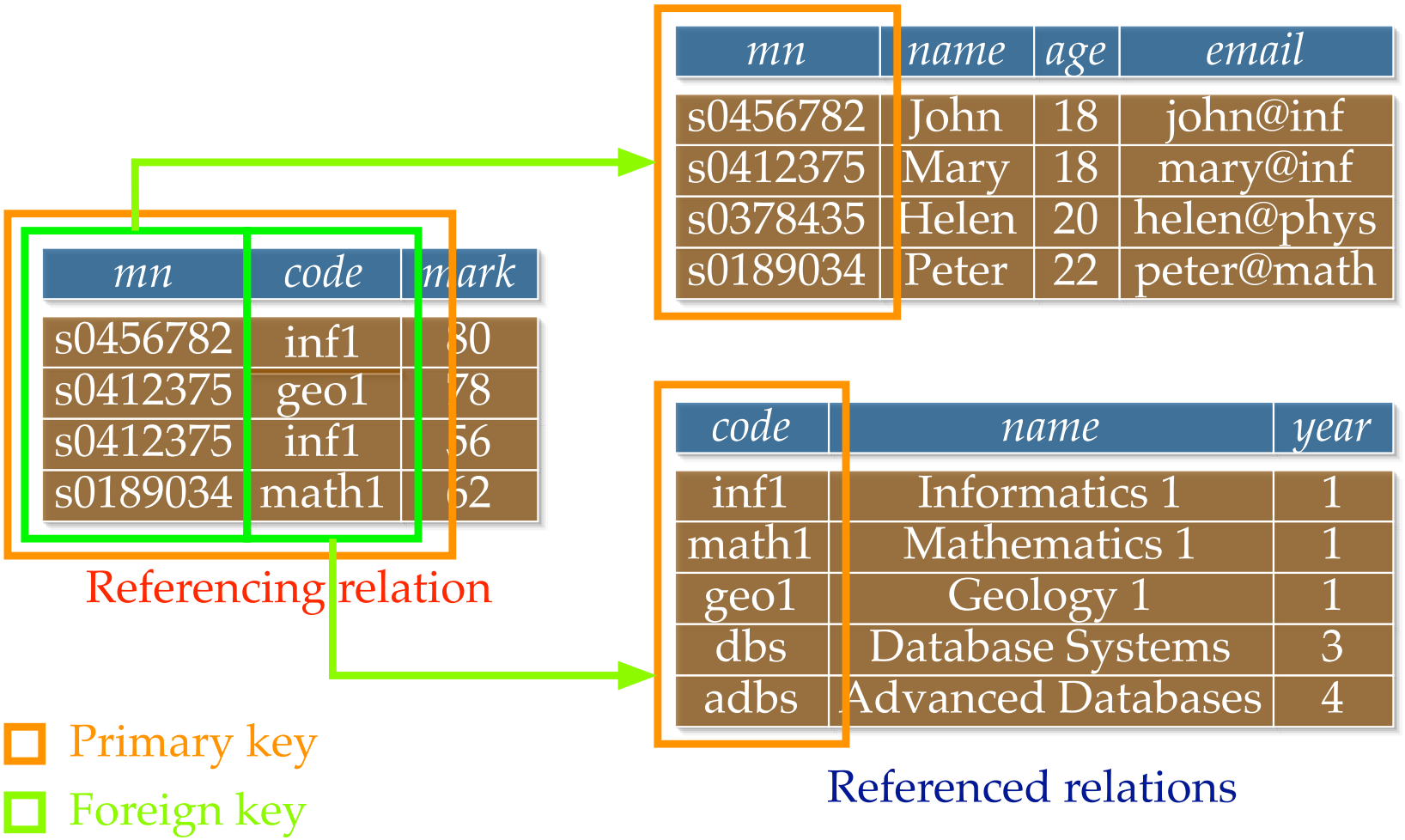
In this case, the primary key is a composite key using a pair of fields.

The *foreign key constraints* given here enforce two further properties:

- Whenever a tuple is inserted, the value for the **mn** field must be a value that appears in the primary key column of the **Students** table
- Similarly, the value for the **code** field must be a value that appears in the primary key column of the **Courses** table

```
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 )
```

Key constraints example



Summary

We have seen two forms of constraint:

`primary key` `(`*declaration*`)`

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

- Primary key constraints declare primary keys.
- Foreign key constraints link columns of one table to the primary key columns of another table.

Both are declared by the user, but enforced by the system itself.
(Attempting to enter a tuple that violates the constraint results in failure.)

N.B. In the ER model, **`Students`** was an entity set and **`Takes`** a relationship. In the relational model, *both* are (necessarily!) implemented as tables.