Introduction (I)

- Seminal article: E.F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* 13(6):377-387 (1970)
- Commercial DBMSs such as Oracle, Informix, SQL Server, Sybase, DB2 as well as public domain DBMS such as PostgreSQL and MySQL are based on the relational data model
- Main reasons for the success of the relational data model
  - Flat two-dimensional tables (relations) as the simple underlying data structure

T	A1	A2	…	A4
	V11	V12	…	V14
	V21	V22	…	V24

  - "Flat" means: No nested complicated structures, that is, attribute fields may *not* contain values such as tables, arrays, lists, trees, etc. but only atomic values

Introduction (II)

- Main reasons for the success of the relational data model (*continued*)
  - Set oriented processing of data in contrast to record oriented processing prevailing until then (hierarchical model, network model)
    - Compare to a programming language example: The task is to copy an array A of integers to an array B
    - Usually performed element-wise by a *record oriented* loop:
      for (int i = 0; i < n; ++i) B[i] = A[i]; // “=” is the assignment operator
    - Desired *set oriented* syntax: B = A;
    - Only possible in object-oriented programming languages by means of overloading
  - Simple comprehensibility also for the unskilled user
  - Very good performance for standard, alphanumerical database applications
  - Existence of a mature, formal theory (in contrast to other data models), in particular with respect to the design of relational databases and with respect to an efficient processing of user queries

Model Definition (I)

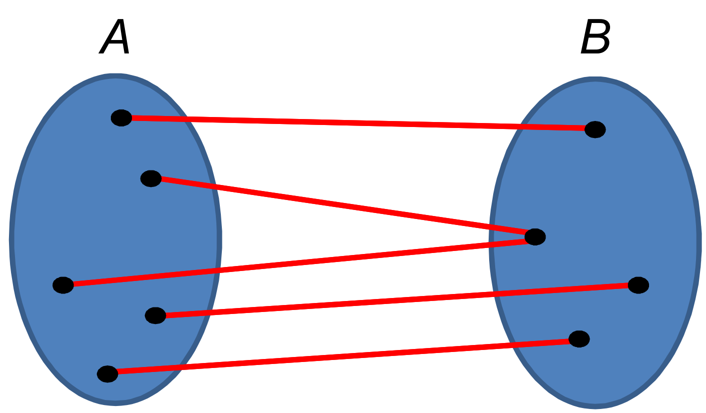
- Given n domains $D_1, D_2, ..., D_n$
  - The term “domain” is a database term for the term “data type”
  - Examples for domains: data types *integer*, *string*[20], *real*, *bool*, *date*, ...
  - Domains need not be disjoint, i.e., $D_i = D_j$ is admissible for $i \neq j$
  - Domains may contain only atomic values, they must not be structured
- A relation (instance) r_R is defined as a subset of the Cartesian product of n domains:

  $r_R \subseteq D_1 \times D_2 \times \ldots \times D_n$ (r_R finite)

- r_R is an occurrence (instance) of a pertaining relation schema R (analogously to the programming language notions of *variable* and *type*).
- An element of the set r_R is called tuple, a tuple has the arity or degree n

Model Definition (II)

- Example: Assume domains $D_1 = \{a, b, c\}$, $D_2 = \{0, 1\}$
  - Cartesian product: $D_1 \times D_2 = \{(a, 0), (a, 1), (b, 0), (b, 1), (c, 0), (c, 1)\}$
  - Examples of instances: $r_1 = \{(a, 0), (b, 0), (c, 0), (c, 1)\}$, $r_2 = \{(a, 0)\}$, $r_3 = \varnothing$
- Number of elements of $D_1 \times D_2$: $|D_1 \times D_2| = |D_1| \cdot |D_2|$ where $|A|$ denotes the (finite) cardinality, that is, the number of elements, of a set A
- Difference between a relation and a function
  - A function f between two sets A and B (notation: $f : A \rightarrow B$) is a relation such that each element in A is related (*mapped*) to exactly one element in B
  - Diagram



Model Definition (III)

- Distinction between the schema of a relation R, which is given by the n domains (data types), and the current instance of this relation schema, which is given by a subset of the Cartesian product
- Schema analogously to the programming language notion of *type*
- A relation schema R, denoted by $R(A_1, A_2, ..., A_n)$, consists of the relation name R and a list of attributes $A_1, A_2, ..., A_n$
- Each attribute A_i is the name of a role played by domain D_i in the relation schema R
  - D_i is also the domain (type) of A_i
  - Notation: $D_i = dom(A_i)$
- For the schema $R(A_1, ..., A_n)$ holds: $r_R \subseteq dom(A_1) \times ... \times dom(A_n)$
- We describe the schema of R also in the form $R(A_1 : D_1, ..., A_n : D_n)$
- Because we often do *not* make a clear distinction between the meta level (schema) and the instance level (occurrence), we also denote relation instances with the letter R

Model Definition (IV)

- Representation of a relation as tables with rows (tupels) and columns

R	A	N
	a	0
	a	1
	b	0
	b	1
	c	0
	c	1

R is a table name, A and N are attributes and have the function of column names, each horizontal line represents a row or tuple

- Example: relation Students(RegNo : *string*, Name : *string*, Age : *integer*, ...)

Students	RegNo	Name	Age	...
	123456	Meyer John	22	...
	456123	Smith Ben	23	...
	321654	Benson Jeff	27	...
	654321	Bates Allen	21	...
	...	...	...	...

Model Definition (V)

- Database schema: collection of relation schemas (more static character, changes rarely)

- Database: collection of the current relation instances (more dynamic character, changes (more) often)

- The definitions so far allow instances that cannot exist in reality
  - Example: An attribute *age* of type *integer*. Values such as -34 and 18792 are syntactically correct but make no sense semantically.
  - Hence, it makes sense to restrict the instances by suitable semantical conditions called integrity constraints (full discussion later)

Features of Relations (I)

- Difference between a *set* and a *list*
  - A set is an *unordered* homogeneous collection of values
    - For example, {3, 9, 6, 4, 1, 2}
    - Duplicates are not possible, sorting is not possible
  - A list is an *ordered* homogeneous collection of values
    - For example, ⟨5, 2, 1, 9, 17⟩
    - Duplicates are possible: for example, ⟨5, 2, 1, 5, 1, 9, 17, 5⟩
    - Sorting is possible: for example, ⟨1, 1, 2, 5, 5, 5, 9, 17⟩
- A relation is defined as a *set* of tuples
  - Tuples in a relation are not ordered since an order of the tuples is not semantically relevant
  - Defining a relation as a list of tuples would allow sorting
  - Note: The rows in a table are ordered
- Relations are based on a set model, relational tables (SQL tables) are based on a list model

Features of Relations (II)

- A tuple is defined as a *list* of *n* attribute values
  - Attribute values in a tuple are ordered
  - But the order of attributes and their values is not semantically relevant
  - It is only necessary to maintain the implicit, position-based correspondence between attributes and their values: Given the attributes $(A_1, ..., A_n)$ and the tuple $(v_1, ..., v_n)$, the value v_i corresponds to the attribute A_i
  - A tuple *t* could be defined as a *set* of (*attribute*, *attribute value*) pairs, that is, $t = \{(A_1, v_1), (A_2, v_2), \ldots, (A_n, v_n)\}$ where the A_i are attributes and the $v_i \in dom(A_i)$

- Attribute values in tuples
  - Each attribute value in a tuple is atomic (indivisible), that is, no composite or multivalued attributes are allowed (first normal form)
  - Values of attributes in a tuple can be unknown: use of a special value null for this case

Keys

- Analogously to the notion of key in the E-R model
- Due to the set property of relations there are no two tuples that have the same combination of values for all their attributes
- Let us assume $R(A_1, A_2, ..., A_n)$, and let $X \subseteq \{A_1, A_2, ..., A_n\}$. For $t \in r_R$ let $t[X]$ be the projection of t to the attributes in X. X is called key if the following two conditions are fulfilled:
  1. Uniqueness: For all relation instances r_R of R holds:

     $\forall\ t_1, t_2 \in r_R : t_1[X] = t_2[X] \Rightarrow t_1 = t_2$

     (Alternatively: $\forall\ t_1, t_2 \in r_R : t_1 \neq t_2 \Rightarrow t_1[X] \neq t_2[X]$)
  2. Minimality: There is no $Y \subset X$ so that uniqueness is fulfilled
- Candidate keys: Several possible keys, one of them is selected as the primary key, the others do not lose their key property and can be used for creating indexes on them
- Examples: SSN and UFID are candidate keys for students, ISBN and article numbers are candidate keys for books