# The Relational Data Model: Structure

## 1 Overview

- By far the most likely data model in which you'll implement a database application today.
- Of historical interest: the relational model is not the first implementation data model. Prior to that were the network data model, exemplified by CODASYL, and the hierarchical model, implemented by IBM's IMS.
- Also of historical interest: when object-oriented programming gained prominence, there were many attempts to replace a relational database with an *object database*, but generally they did not fare well. Today, object-oriented concepts co-exist with underlying relational databases through *object-relational mapping* algorithms.
- Salient features of the relational model:
  - Conceptually simple; the fundamentals are intuitive and easy to pick up.
  - Powerful underlying theory: the relational model is the only database model that is powered by formal mathematics, which results in excellent dividends when developing database algorithms and techniques.
  - Easy-to-use database language: though not formally part of the relational model, part of its success is due to SQL, the de facto language for working with relational databases.

## 2 Structure

- A relational database is a collection of *tables*.
  - Each table has a unique name.
  - Each table consists of multiple *rows*.
  - Each row is a set of values that by definition are *related* to each other in some way; these values conform to the *attributes* or *columns* of the table.

- Each attribute of a table defines a set of permitted values for that attribute; this set of permitted set is the *domain* of that attribute.

- This definition of a database table originates from the pure mathematical concept of a *relation*, from which the term "relational data model" originates.
  - Formally, for a table $r$ with $n$ attributes $a_1 \ldots a_n$, each attribute $a_k$ has a domain $D_k$, and any given row of $r$ is an $n$-tuple $(v_1, \ldots, v_n)$ such that $v_k \in D_k$.
  - Thus, any instance of table $r$ is a subset of the Cartesian product $D_1 \times \cdots \times D_n$.
  - We require that a domain $D_k$ be *atomic* — that is, we do not consider the elements of $D_k$ to be breakable into subcomponents.
  - A possible member of any domain is *null* — that is, an unknown or non-existent value; in practice, we try to avoid the inclusion of *null* in our databases because they can cause a number of practical issues.

- This definition is virtually identical to the pure mathematical definition of a relation — the main difference is that, for a database, we assign names to the table's attributes, where mathematical relations only label their attributes by their integer indices $1 \ldots n$.

- To avoid jargon proliferation, we will stick with the more formal terms "relation" for table and "tuple" for row — which is why we used $r$ to represent a table above instead of $t$.

- More notation:
  - Given a tuple $t$ that belongs to a relation $r$, we can say that $t \in R$ since after all $r$ is a set of tuples.
    - By "set of tuples" we do mean the mathematical concept of a set; thus order doesn't matter.
    - Two relations $r$ and $s$ are the same as long as they have the same attributes $a_1 \ldots a_n$ with domains $D_1 \ldots D_n$ and contain as elements the same tuples with the same values, regardless of the order in which these tuples appear.
  - To talk about a specific attribute $a_k$ of $t$, we write $t[a_k]$.
  - This notation also applies to a set of attributes $A$ — the notation $t[A]$ refers to the "sub-tuple" of $t$ consisting only of the attributes in $A$.
  - Alternatively, $t[k]$ can refer to that same attribute of $t$, as long as we are consistent about how attributes are ordered in the relation.

## 2.1 Schemas and Instances

- Just as with classes and instances in the object-oriented world, a similar dichotomy exists between a *database schema* and a *database instance*.

  - The *schema* refers to the database's design or definition — what relations it contains, what attributes these relations have, and to what domains these attributes belong.
  - An *instance* of the database is a specific snapshot of the data in that database at a given point in time.

- Ditto with *relation schemas* — lists of attributes and their corresponding domains — and *relation instances*.
  - The SKS convention is to use names with an initial uppercase letter to represent a relation schema, and all-lowercase for a relation instance (or, simply, a relation).
  - The notation $r(R\_schema)$ indicates that the relation $r$ conforms to a schema $R\_schema$.

## 2.2 Keys: Super, Candidate, Primary, Foreign

- Since a tuple $t$ is completely defined by its values $(v_1 \ \ldots \ v_n)$, there is no way to differentiate between $t = (v_1 \ \ldots \ v_n)$ and $t' = (v'_1 \ \ldots \ v'_n)$ if $v_k = v'_k \ \forall \ 1 \leq k \leq n$. In the relational model, two such tuples are one and the same.
- Thus, we need a value-based way for differentiating tuples in a relation. Values or sets of values that distinguish one tuple from another are called *keys*.
- A *superkey* is a set of one or more attributes that uniquely identify a tuple in a relation. The superkey can range from a single attribute, if no two tuples will ever have the same value for that attribute, to the entire tuple. As long as the set of attributes will distinguish tuples from each other, then it is a superkey.
  - Thus, if $K$ is a superkey, then any superset of $K$ up to the entire attribute set is also a superkey.
  - More formally, if $R$ is a relation schema, then a subset $K$ of $R$ is a superkey for $R$ if, for a relation $r(R)$ and tuples $t_1$ and $t_2$ in $r$, $t_1 \neq t_2 \rightarrow t_1[K] \neq t_2[K]$.
  - In practice, a superkey is not very useful; we need a stricter definition for a key, which is...
- A *candidate key* is a "minimal superkey" — it is a superkey for which no proper subset is also a superkey.
  - Note that there can still be more than one candidate key for a relation, consisting of different subsets of attributes whose combined values will be unique for all tuples.
- So we finally narrow things down to the *primary key*: the primary key is a candidate key that has been chosen by the database designer as the principal means for identifying tuples in a relation.

- One of the key guidelines (ha! no pun intended, I think…) that makes a candidate key particularly suited for elevation to the status of "the one" primary key is that the value of the candidate key should never, or very rarely, change.

- Say you have two relation schemas $R_1$ and $R_2$ and these schemas share some subset of attributes. If $K_p$ is the primary key of $R_2$ and $K_p \subseteq R_1$ as well — in other words, $R_1$'s attributes include the primary key of $R_2$ — then $K_p$ is a *foreign key* from $R_1$ referencing $R_2$.

  - $R_1$, the relation that contains the foreign key, is the *referencing relation*.
  - $R_2$, the relation for which $K_p$ is a primary key, is the *referenced relation*.
  - In practice, specifying a foreign key places a *constraint* on the data that any $r_1(R_1)$ and $r_2(R_2)$ may contain: that is, $\forall$ tuples $t_1 \in r_1$, there *must* be a tuple $t_2 \in r_2$ such that $t_1[P_k] = t_2[P_k]$.
  - Quick — clearly there is only one such tuple $t_2$. Why?

## 3 Relational Schema Diagram Notation

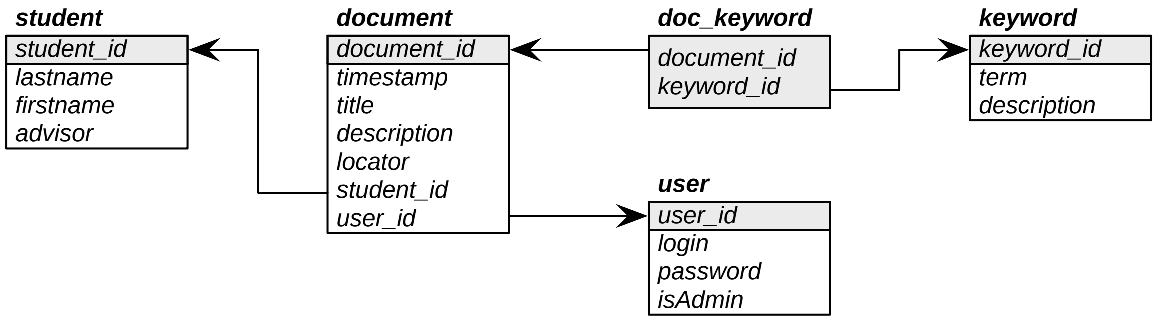

Figure 1: Sample relational schema diagram for our sample document management database application.

- A relational database schema can be depicted pictorially through a *relational schema diagram*. Yes, it's yet another notation like E-R and UML, but this notation is very focused and specific to the relational data model:

  - Relations are drawn as boxes with the relation name above the box.
  - A relation's attributes are listed within its box.
  - The attributes that belong to the relation's primary key (if it has one) are listed first, with a line separating this primary key from the other attributes and their background shaded in gray.
  - Foreign key dependencies are illustrated as arrows from the foreign key attributes of the referencing relation to the primary key attributes of the referenced relation.

- Figure 1 illustrates a possible relational schema diagram for our sample document management database application, with the *id* attribute for each class serving as the primary key for the corresponding table and foreign key attributes added to represent the associations between classes.

- Note how we have added another relation to represent the association between documents and keywords — this is a standard technique for expressing a many-to-many relationship in terms of the relational data model.

- How about a 1-to-many relationship — can you see from the diagram how that kind of relationship gets mapped to the relational model?