database systems

formal definitions
of the
relational data model
[ 03 ]

s. yurttaş
1 the relational data model

The relational model of data was introduced by E.F. Codd (1970).

1.1 relational model concepts

The relational model represents the data in a database as a collection of relations. When a relation is thought of as a table of values, each row in the table represents a collection of related data values. These values can be interpreted as a fact describing an entity or relationship instance.

In relational database terminology, a row is called a tuple, a column name is called an attribute, and the table is called a relation. The data type describing the types of values that can appear in each column is called a domain.

1.2 domains, tuples, attributes, and relations

A domain $D$ is a set of atomic values. By atomic we mean that each value in the domain is indivisible as far as the relational model is concerned.

A relation schema $R$, denoted by $R(A_1, A_2, \ldots, A_n)$, is a set of attributes $R = \{A_1, A_2, \ldots, A_n\}$. Each attribute $A_i$ is the name of a role played by some domain $D$ in the relation schema $R$. $D$ is called the domain of $A_i$ and is denoted by $\text{dom}(A_i)$. A relation schema is used to describe a relation; $R$ is called the name of this relation. The degree of a relation is the number of attributes $n$ of its relation schema.

A relation (or relation instance) $r$ of the relation scheme $R(A_1, A_2, \ldots, A_n)$, also denoted by $r(R)$, is a set of $n$-tuples $r = \{t_1, t_2, \ldots, t_m\}$. Each $n$-tuple $t$ is an ordered list of $n$ values $t = (v_1, v_2, \ldots, v_n)$, where each value $v_i$, $i \leq i \leq n$, is an element of $\text{dom}(A_i)$ or is a special null value. The terms relation intension for the schema $R$ and relation extension for a relation instance $r(R)$ are also commonly used.

The above definition of a relation can be restated as follows: A relation $r(R)$ is a subset of the Cartesian product of the domains that define $R$. 
\[ r(R) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times \ldots \times \text{dom}(A_n)) \]

The Cartesian product specifies all possible combinations of values from the underlying domains. Hence, if we denote the number of values or cardinality of a domain \( D \) by \( |D| \), then, assuming all domains are finite, the total number of tuples in the Cartesian product would be

\[ |\text{dom}(A_1)| \times |\text{dom}(A_2)| \times \ldots \times |\text{dom}(A_n)| \]

Out of all these possible combinations, a relation instance at a given time – the current relational instance – reflects only the valid tuples that represent a particular state of the real world. In general, as the state of the real world changes, so does the relation by being transformed to another relation instance. However, the schema \( R \) is relatively static and does not change except very infrequently – for example, by adding an attribute to represent new information that was not originally stored in the relation.

It is possible for several attributes to have the same domain. The attributes indicate different roles, or interpretations, for the domain.

An alternative definition of a relation can be given, making the ordering of values in a tuple unnecessary. In this definition a relation \( r \) of relation schema \( R = \{A_1, A_2, \ldots, A_n\} \) is a finite set of mappings \( r = \{t_1, t_2, \ldots, t_m\} \), where each tuple \( t_i \) is a mapping from \( R \) to \( D \), and \( D \) is the union of the attribute domains; that is, \( D = \text{dom}(A_1) \cup \text{dom}(A_2) \cup \ldots \cup \text{dom}(A_n) \). In this definition, \( t(A_i) \) must be in \( \text{dom}(A_i) \) for \( 1 \leq i \leq n \) for each mapping \( t \) in \( r \). Each mapping \( t_i \) is called a tuple.

According to this definition, a tuple can be considered as a set of \(<\text{attribute}>, <\text{value}>\) pairs, where each pair gives the value of the mapping from an attribute \( A_i \) to a value \( v_i \) from \( \text{dom}(A_i) \). The ordering of attributes is not important because the attribute name appears with its value.

When a relation is implemented as a file, the attributes can be physically ordered as fields within a record. We will continue to use the first definition of relation, where the values within tuples are ordered, because it simplifies much of the notation.

Each value in a tuple is an atomic value; that is, it is not divisible into components within the framework of the relational model. Hence, composite and multivalued attributes are not allowed. Much of the theory behind the relational model was developed with this assumption in mind, which is called the first normal form assumption.
The values of some attributes within a particular tuple may be unknown or may not apply to this particular tuple. A special value, called **null**, is used for these cases.

- The relation schema can be interpreted as declaration or a type of **assertion**.
- An alternative interpretation of a relation schema is a **predicate**; in this case, the values in each tuple are interpreted as values that satisfy the predicate.

### 1.3 relational model notation

- A relation schema $R$ of degree $n$ is represented as $R(A_1, A_2, \ldots, A_n)$.
- An $n$-tuple $t$ in a relation $r(R)$ is represented as $t = \langle v_1, v_2, \ldots, v_n \rangle$, where $v_i$ is the value corresponding to attribute $A_i$. We use the following notation to refer to **component values** of tuples:
  
  - $t[A_i]$ refers to the value $v_i$ in $t$ for attribute $A_i$.
  - $t[A_u, A_w, \ldots, A_z]$, where $A_u, A_w, \ldots, A_z$ is a list (or set) of attributes from $R$, refers to subtuple of values $\langle v_u, v_w, \ldots, v_z \rangle$ from $t$ corresponding to the attributes specified in the list.

- The letters $Q, R, S$ denote relation names.
- The letters $q, r, s$ denote relation instances.
- The letters $t, u, v$ denote tuples.
- In general, the name of a relation such as $STUDENT$ indicates the current set of tuples in that relation – the current relation instance – whereas $STUDENT(\text{Name, SSN,} \ldots)$ refers to the relation schema.
- Attribute names are sometimes qualified with the relation name to which they belong, for example, $STUDENT.\text{Name}$ or $STUDENT.\text{Age}$. 
1.4 key attributes of a relation

A relation is defined as a set of tuples. By definition, all elements of a set are distinct; hence, all tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for all their attributes. Usually, there are other subsets of attributes of a relation schema $R$ with the property that no two tuples in any relation instance $r$ of $R$ should have the same combination of values for these attributes. Suppose we denote one such subset of attributes by $SK$; then for any two distinct tuples $t_1$ and $t_2$ in a relation instance $r$ of $R$, we have:

$$t_1[SK] \neq t_2[SK]$$

Any such set of attributes $SK$ is called a superkey of the relation schema $R$. Every relation has at least one superkey – the set of all its attributes. A key $K$ of a relation schema $R$ is a superkey of $R$ with the additional property that removing any attribute $A$ from $K$ leaves a set of attributes $K'$ that is not superkey of $R$. Hence, a key is a minimal superkey, a superkey from which we cannot remove any attributes.

In general, a relation schema may have more than one key. In this case, each of the keys is called a candidate key.

It is common to designate one of the candidate keys as the primary key of the relation. This is the candidate key whose values are used to identify tuples in the relation. We use the convention that the attributes that form the primary key of a relation schema are underlined. When a relation schema has several candidate keys, the choice of one to become primary key is arbitrary; however, it is usually better to choose a primary key with single attribute or a small number of attributes.

1.5 relational database schemas and integrity constraints

A relational database schema $S$ is a set of relation schemas $S = \{R_1, R_2, \ldots, R_m\}$ and a set of integrity constraints $IC$. A relational database instance $DB$ of $S$ is a set of relation instances $DB = \{r_1, r_2, \ldots, r_m\}$ such that the $r_i$'s satisfy the integrity constraints specified in $IC$.

Integrity constraints are specified on database schema and are expected to hold on every database instance of that schema. Key constraints that specify the candidate keys of each relation schema; candidate key values must
be unique for every tuple in any relation instance of that relation schema. In addition to the key constraints, two other types of constraints are considered part of the relational model – entity integrity and referential integrity.

The **integrity constraint** states that no primary key value can be null. Key constraints and entity integrity constraints are specified on individual relations.

The **referential integrity constraint** is a constraint that is specified between two relations and is used to maintain consistency among tuples of the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an **existing tuple** in that relation.

To define referential integrity more formally, we need first to define the concept of a foreign key. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas $R_1$ and $R_2$. A set of attributes $FK$ in relation schema $R_1$ is a **foreign key** of $R_1$ if it satisfies the following two rules:

1. The attributes in $FK$ have the same domain as the primary key attributes $PK$ of another relation schema $R_2$; the attributes $FK$ are said to **reference** or **refer to** the relation $R_2$.

2. A value of $FK$ in a tuple $t_1$ of $R_1$ either occurs as a value of $PK$ for some tuple $t_2$ in $R_2$ or is null. In the former case, we have $t_1[FK] = t_2[PK]$, and we say that the tuple $t_1$ **references** or **refers to** the tuple $t_2$.

In a database of many relations, there will usually be many referential integrity constraints. To specify these constraints we must first have a clear understanding of the meaning or role that each set of attributes plays in the various relation schemas of the database. Referential integrity constraints typically arise from the **relationships among the entities** represented by the relation schemas.

A foreign key can **refer to its own relation**.

We can **diagrammatically display referential integrity constraints** by drawing a directed arc from each foreign key to the relation it references.

All integrity constraints should be specified on the relational database schema if we are interested in maintaining these constraints on all database instances. Hence, in a relational system, the data definition language (DDL) should include provisions for specifying the various types of constraints so
that the DBMS can automatically enforce them. Most relational database management systems support key and entity integrity constraints but unfortunately not referential integrity, although some systems are starting to support referential integrity.

The above types of constraints do not include a large class of general constraints, sometimes called semantic integrity constraints, that may need to be specified and enforced on a relational database.