CSCI 403: Databases
2 - The Relational Model

High Level

A *relation* resembles a table of values, or a flat file of records. A record is a collection of named data values representing some fact about the world. In the tabular view of things, each row is a record. Each column represents a specific field or *attribute* from every record in the table.

Tables and attributes in the relational model are named. For example, figure 1 shows a few rows and columns from the table **mines_courses**.

Definitions & Formalism

Tuples

In the relational model, rows are called *tuples*, and relations are simply *sets* of tuples. A tuple is an ordered and named collection of values. For instance, from figure 1, one tuple is
('Mehta, Dinesh', 'CSCI406', 'ALGORITHMS', 46).
The ordered collection
('instructor', 'course_id', 'title', 'enrollment')
contains the *attribute names* for the tuple.

The formal model is based on set theory, and formally the attributes of a tuple form a set. However, sets are intrinsically unordered, meaning we'd need to attach names to every value in every tuple to know what is going on unless we assume some convention of ordering. So by convention we order the attributes and then apply the same order to the values in the corresponding tuples.

Each attribute in a tuple has an associated *domain* of values which the values are constrained to belong to (e.g., strings of length 4, floating point numbers, dates).

Relation

We denote a *relation schema* R as $R(A_1, A_2, \ldots, A_n)$. The relation schema R has *degree* n and attributes $A_1, A_2, \ldots, A_n$. Each attribute has associated domain $D_i = \mathrm{dom}(A_i)$. (Notation here is not particularly important, e.g., something you will be tested on, but sometimes it helps in thinking about what is going on.)

A *relation state*, or simply *relation*, $r(R)$, is a set of tuples conforming to the relation schema R. That is, we can say for any tuple $t \in r$:

$$t = (v_1, v_2, \ldots, v_n) : v_i \in \mathrm{dom}(A_i).$$

We can additionally denote the value corresponding to the specific attribute A_i as $t.A_i$ or $t[A_i]$.

We can equivalently define a relation as any subset of the Cartesian product of the domains of the attributes:

$$r(R) \subseteq (\mathrm{dom}(A_1) \times \mathrm{dom}(A_2) \times \cdots \times \mathrm{dom}(A_n))$$

Some Notes

The statement that "a relation is a set" implies that there is no ordering of tuples in a relation. This is true in practice as well, in that a relational DBMS may choose to store tuples (rows) of a relation (table) in any order or fashion according to the design of the DBMS. Formally, two relations are equivalent if they have the same tuples, even if the tuples are in different orders.

The same statement also implies that there are no duplicates in a relation, i.e., that no two tuples contain the exact same values. In practice, this restriction is not enforced by relational DBMSs, so technically relations in a DBMS form a *multiset*. The implications of this will be discussed more when we talk about SELECT queries and the DISTINCT operator.

In the same vein, formally the attributes of a relation schema have no order, thus two relations

instructor	course_id	title	enrollment
Painter-Wakefield, Christopher	CSCI262	DATA STRUCTURES	90
Painter-Wakefield, Christopher	CSCI403	DATABASE MANAGEMENT	73
Fisher, Wendy	CSCI303	DATA SCIENCE	60
Mehta, Dinesh	CSCI406	ALGORITHMS	46

Figure 1: Sample rows from the table **mines_courses** (only some columns shown)

that are equivalent except for the ordering of their attributes are equivalent.

NULL

A few words need be said about NULL at this point. NULL is a "state" of a data element that can exist in a relational database to represent a variety of concepts. Basically it is the absence of a value. NULL can mean that the value is unknown, or missing, or simply irrelevant to a particular tuple. In the mines_courses table there are many rows with NULL instructor; this likely represents the fact that when this data was obtained, not all course instructors had been assigned.

It is very important to note that NULL values cannot be compared. In particular, two NULL values are never equal to each other. In queries (see next lecture), there is a special operator (IS NULL) for detecting the presence of a NULL value.

NULL has some other interesting properties; for instance, the result of arithmetic or string operations involving NULL is NULL. E.g., `NULL` + 42 results in NULL, `'Hello' || NULL` results in NULL, even `NULL / 0` yields NULL.

NULL results in a special "3-value logic" for handling Boolean comparisons when NULL is a possible state. The result of a Boolean operation can be "true", "false", or "unknown", where "unknown" is typically represented by NULL. You can find the resulting truth table various places, but it basically comes down to whether or not you can determine the outcome of the operation knowing just the non-NULL values. For instance, `TRUE AND NULL` yields NULL ("unknown"), since the evaluation could go either way depending on how you replace NULL with an actual value. On the other hand, `TRUE OR NULL` evaluates to TRUE, since it doesn't matter what the other value is in an OR expression when one is TRUE.

Constraints

In addition to relations as described above, the relational model is also concerned with *constraints*. Constraints are simply restrictions on a relation. There are three flavors of constraints:

1. implicit (model-based)
2. explicit (schema-based)
3. application-based

Implicit constraints are those which are implied by the model of the world that a schema represents. For example, the instructor field of the mines_courses table is assumed to contain person's names. Application-based constraints are constraints which are not defined in the database, but which are enforced by the applications which use the database; these are often known as the "business rules" of an application, and can be things like "no employee can have a hire date that is not on the first of some month". Such constraints tend to be more complex to implement on the DBMS itself, and thus are relegated to application code.

In terms of the relational model, we are mostly interested in the explicit constraints. One trivial constraint enforced by the database are the domain constraints requiring that values in a column be part of the domain as defined for the column (attribute) in the relation schema. The next constraint of interest is called a *primary key constraint*, and to explain that we need to define some more terms.

Keys and Superkeys

A *superkey* of a relation schema R is some subset of R's attributes with the property that no relation of R may contain tuples with exactly the same values for the attributes in the superkey. For example, figure 2 shows some rows

from the mines_courses table, this time with some different attributes shown. If we consider the set (course_id, section, instructor), no two rows shown (and in fact, in the actual table) have the same values for all three attributes. Thus (course_id, section, instructor) is a superkey of mines_courses.

Some facts about superkeys that are immediately obvious; first, if we consider the formal relational model in which duplicate tuples are not allowed, then every relation schema has at least one superkey, which is the set of all attributes of the schema. Second, clearly any superset of a superkey is also a superkey.

A *key* is simply a superkey with the property that no attribute can be removed from the key without destroying the superkey property. For example, the set (course_id, section, instructor) is not a key, as we can remove the instructor attribute and have the superkey (course_id, section). On the other hand, if we have (course_id, section), clearly we cannot remove either attribute, since both columns have duplicates by themselves. Therefore (course_id, section) is a key of mines_courses. Another way of defining key is that it is a *minimal* superkey.

Multiple keys are possible; in the mines_courses table, (crn) is also a key. All keys of a relation schema are called the *candidate keys*. Conventionally, we select one key to identify as the *primary key* for a relation. The primary key is a unique identifier for any tuple in the relation. Typically we prefer smaller sets over larger ones for the primary key, thus (crn) is a good choice for the primary key of the mines_courses table.

Note that no tuple in a relation can contain a NULL value for any attribute of a key; since NULLs cannot be compared, it is impossible to uniquely identify a record if part or all of its key may be NULL.

In practice, we can set a primary key constraint as part of the relation schema in a DBMS. This constraint is then enforced by the DBMS, requiring uniqueness for the attributes in the primary key. (Also, the DBMS will not allow NULL values.) Typically other keys can be enforced as well by a uniqueness constraint + a not null constraint, but the convention is to have only one primary key identified for a relation. In PostgreSQL, you can have only one primary key on a table.

Relational Database

So far, we have been discussing constraints as applied to a single relation schema. In the larger picture, a relational database is made up of multiple tables and associated constraints, and conforms to what is called the *relational database schema*. The relational DB schema also allows for constraints that represent relationships between relation schemas. These are called *referential integrity constraints*, or *foreign key constraints*.

Basically, a foreign key constraint is used to require that entries in one table have corresponding entries in another. Formally, a set of attributes is a foreign key of a relation schema if its values are either NULL or exist in the specified attributes of a referenced relation schema (note that the domains must match). For example, suppose there exists a table named instructors containing name, office, email, and other information about every instructor at Mines. Then we can create a foreign key constraint relating the mines_courses table (instructor attribute) to the instructors table (name attribute). Each instructor value in mines_courses would then be required to be either in the instructors table, or NULL.

As another example, it is possible to have a referential integrity constraint from a table to itself; an example from the book is of a table containing employee information. The primary key of the employee table is the employee's SSN, and the table also contains the SSN of the supervisor of every employee. Thus there exists a foreign key constraint on employee (superssn) referencing employee (ssn), meaning that the supervisor of any employee must also be an employee. (This table is also in the csci403 database.)

crn	course_id	section	instructor	title
10120	CSCI262	B	Painter-Wakefield, Christopher	DATA STRUCTURES
12693	CSCI262	R02	Painter-Wakefield, Christopher	DATA STRUCTURES
12048	CSCI403	A	Painter-Wakefield, Christopher	DATABASE MANAGEMENT
12623	CSCI406	A	Mehta, Dinesh	ALGORITHMS
10300	CSCI406	A	Mehta, Dinesh	ALGORITHMS
12621	CSCI303	A	Fisher, Wendy	DATA SCIENCE

Figure 2: Some more rows from the mines_courses tables, this time selecting different attributes