# Relational DB Structure

23 September 2023 18:05

Thus, in the relational model:
- the term **relation** is used to refer to a **table**
- The term **tuple** is used to refer to a **row**.
- The term **attribute** refers to a **column** of a table.

e (or list) of values. A
by an n-tuple of values,

Attribute or Column

| ID | name | dept_name | salary |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | History | 62000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

Record or Tuple

Instructor table or Relation

- Relational instance: A specific instance of a relation, which includes a particular set of rows
- The relation is a set of tuples. So like in any other set, the Order of arrangement of tuples doesn't matter. The tables are considered to be same.
- Domain of attribute: Permitted values of attribute
  - Domains are atomic i.e indivisible
- Null value to represent that there is no available data for that particular attribute of a record number for the instructor.
  - Can cause challenges when accessing or updating the database
  - May lead to issues in calculations, comparisons, and querying operations, so best to eliminate.
- Relational schema: List of attributes and their domains
- Contents of relational instance may change over time but schema does not.
- Using common attributes in relation schemas is one way of relating tuples of distinct relations.

## Constraints

Constraints determine which values are permissible and which are not in the database.
They are of three main types:

- **Inherent or Implicit Constraints**:
  - Based on the data model itself. (E.g., relational model does not allow a list as a value for any attribute)
- **Schema-based or Explicit Constraints**:
  - Expressed in the schema by using the facilities provided by the model. (E.g., max. cardinality ratio constraint in the ER model)
- **Application based or semantic constraints**:
  - Beyond the expressive power of the model and must be specified and enforced by the application programs.

The Main types of (**explicit**/ **schema-based**) constraints that can be expressed in the relational model:
- **Domain** constraint.
- **Key constraints** and **Constraints on NULL** values
- **Integrity** Constraints
  - **Entity integrity** constraints
  - **Referential integrity** constraints

## Domain Constraints

Domain Constraint specifies that
- Within each tuple, the value of each attribute A must be an atomic value from the domain dom(A)
- Every value in a tuple must be from the domain of its attribute (or it could be null, if allowed for that attribute)

- The data types associated with domains include
  - Standard **numeric** data types for **integers** (such as short integer, integer, and long integer)
  - **Real numbers** (float and double-precision float).
  - **Characters,**
  - **Booleans**,
  - Fixed-length strings, Variable-length strings, as are date, time, timestamp, and other special data types.
  - Domains can also be described by a **subrange of values** from a data type or
  - as an **enumerated** data type in which all possible values are explicitly listed

**Keys**

Values of attributes must be such that each tuple can be uniquely identified.
No two tuples can have exactly same values.

**Superkey**

One or more attributes together can uniquely identify the tuple i.e 2 tuples cannot have same super key value.

Instructor (ID, name, dept_name, salary)

- Now as seen before {ID} is a superkey as it uniquely identifies each tuple in the relation
- What about the set {ID, name} is this also a superkey?

- The answer is yes
- A superkey may contain extraneous attributes like shown above
- If SK is a superkey, then so is any superset of SK.
- We are often interested in superkeys for which no proper subset is a superkey. Such **minimal superkeys** are called **candidate keys**.

**Car** (State, Reg#, SerialNo, Make, Model, Year)

- If we notice, the SerialNo is an candidate key as it is unique for the Car
- But we also know that {State, Reg#} together are also unique for a given car

- So in the above case {SerialNo} and {State, Reg#} both are minimal superkeys and also called candidate keys

- Thereby we can conclude that A relation schema may have more than one minimal superkey. In this case, each of them is called a candidate key.

- If a relation has several candidate keys, one is chosen arbitrarily to be the **primary key**.
- The primary key attributes are underlined.
  - Example: Consider the CAR relation schema:
  - **CAR**(State, Reg#, <u>SerialNo</u>, Make, Model, Year)
  - We chose SerialNo as the primary key

- The primary key value is used to uniquely identify each tuple in a relation
- Provides the tuple identity
- Also used to reference the tuple from another tuple
  - General rule: Choose as primary key the smallest of the candidate keys (in terms of size)
  - Not always applicable – choice is sometimes subjective

  - It is customary to list the primary key attributes of a relation schema before the other attributes;

- Choose attributes that are rarely changed for primary key
- Valid state: State of db that satisfies all integrity constraints

**Entity Integrity constraint**
Primary key cannot have null values

**Foreign key Constraint**

A foreign-key constraint from attribute(s) A of relation r1 to the primary-key B of relation r2 states that
- on any database instance, the value of A for each tuple in r1 must also be the value of B for some tuple in r2.
- Attribute set A is called a foreign key from r1, referencing r2.
- The relation r1 is also called the **referencing relation** of the foreign-key constraint, and r2 is called the **referenced relation**.

For example, let us consider the following schema:
Instructor (<u>ID</u>, name, Dept_name, Salary)
Department (<u>Dept_name</u>, building, budget)

- attribute "dept_name" in instructor is a foreign key from the instructor table, referencing the department table;
- Note that dept_name is the primary key of the department.

So here,
- Instructor  referencing relation
- Department  referenced relation

**** Note that in a foreign-key constraint, the referenced attribute(s) must be the primary key of the referenced relation ****

**Foreign Key constraint**
Referenced attributes must be primary key of referenced relation.

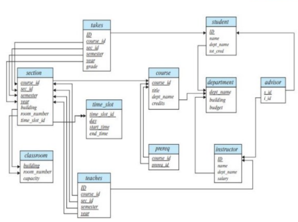
**Referential Integrity constraint**
Values appearing in specified attributes of any tuple in the referencing relation also appear in specified attributes of at least one tuple in the referenced relation.

- Note that in the above example, the time slot does not form a primary key of the time slot relation, although it is a part of the primary key;
- thus, we cannot use a foreign-key constraint to enforce the above constraint.

- In fact, **foreign-key constraints are a special case of referential integrity constraints**, where the referenced attributes form the primary key of the referenced relation.

- Database systems today typically support foreign-key constraints, but they do not support referential integrity constraints where the referenced attribute is not a primary key

- Each relation appears as a box, with the relation name at the top in blue and the attributes listed inside the box.
- Primary-key attributes are shown underlined.
- Foreign-key constraints appear as arrows from the foreign-key attributes of the referencing relation to the primary key of the referenced relation.
- We use a two-headed arrow, instead of a single-headed arrow, to indicate a referential integrity constraint that is not a foreign-key constraint.



Schema diagram for university Database

- We use the term **entity** to refer to any such distinctly identifiable item.
  - Considering a university database, the entities would be:
    - ➢ instructors,
    - ➢ students,
    - ➢ departments,
    - ➢ courses, etc.

- The various entities are **related** to each other in a variety of ways, all of which need to be captured in the database design.
  - For example:
    - ➢ A student takes a course offering
    - ➢ An instructor teaches a course offering

- While designing a database we must try to avoid two major pitfalls:
  - Redundancy
  - Incompleteness

- **Problem with Redundancy:**
  - Redundant copies of data can become inconsistent if updated without care.
  - Different sections of a course could end up having different titles if not properly managed.

- **Drawbacks of Workaround:**
  - Using null values to represent missing information is not an elegant solution.
  - It might run into issues due to constraints like primary-key requirements.

## Design Alternatives

- **Avoiding Poor Designs is Insufficient:**
  - Simply avoiding bad designs is not enough in database design.
  - Multiple good design options might be available, creating the need to select wisely.
- **The Complexity of Design Choices:**
  - For instance, consider a simple scenario where a customer buys a product.
  - A key decision arises: Is the sale a relationship between customer and product, or is the sale itself a distinct entity connected to both customer and product?
- **Impact of Design Choices:**
  - This choice can significantly influence the effective modeling of various enterprise aspects.
  - Different design approaches might lead to different insights and representations of business operations.

- **Scale of Decision-Making:**
  - In real-world scenarios, numerous entities and relationships require similar decisions.
  - This makes database design a challenging task, demanding careful consideration.
- **Database Design Challenge:**
  - The complexity of design choices highlights the intricate nature of database design.
  - It involves determining suitable structures for various entities and relationships.
- **Combination of Science and Aesthetic Judgment:**
  - Database design necessitates a balanced approach of scientific principles and intuitive "good taste."
  - Successful design involves a mix of methodical analysis and informed decision-making.