Relations

A **relation** is a *named*, set of k-tuples defined by a cross product of k-sets.

For example we can specify a relation as the set of 3-tuples defined by the following cross product.

(INT) X (STRING) X (STRING)

Possible **instances** of this relation are:

(001, Alice, Boston)
(200, Bob, Albany)
(321, Carl, Portland)

The relation above could describe the *studentID*, *name*, and *homeCity* attributes of a Student entity.

A relation can also be viewed as a two-dimensional table of data where columns define the attributes of the relation and rows define instances of the relation.

We can define a relation using a logical schema. They are written using the following form.

- RELATION_NAME (Attribute1, Attribute2, …)

Properties of Relations

- Each relation has a unique name and is CAPITALIZED.
- Each attribute name within a relation is unique.
- Each entry for an attribute in each row must be a single (possibly null) value. Relations cannot have attributes that are composite or multi-valued. They may be derived, but we do not include [] to indicate that they are derived.
- Each row is unique.

Relational Keys

A **primary key** is an attribute or a combination of attributes (a composite key) that uniquely identify each row in a relation. We designate primary keys by underlining the attribute name(s).

EMPLOYEE (EmpID, Name, DeptName, Salary)

The **identifiers** of a strong entities form the primary keys of the relation representing the entity.

The **identifiers** of a weak entity are only part of the weak entity's primary key. More on that later.

DEPENDENT (DependentName, EmpID, DOB)

A **foreign key** is one or more attributes that serve as a primary key in another relation. Foreign keys define relationships between the relations.

We denote a foreign key by placing a dashed line under it unless it is also a primary key in which case we underline it.

DEPARTMENT (DeptName, Location, Phone)
EMPLOYEE (EmpID, Name, DeptName, Salary)

DeptName in the EMPLOYEE relation is a foreign key. It is not a primary key in the EMPLOYEE relation, but it is a primary key in the DEPARTMENT relation.

To denote the relationship between foreign keys and primary keys in a logical schema, arrows are drawn from the foreign keys to the associated primary keys.

Removing Multi-Valued Attributes

One of the properties of relations is that each value at the intersection of a row and column must be a single value. As a preliminary step, before we discuss a process called normalization, we can simply create additional rows in the relation for multi value attributes. This creates data redundancy which we will eliminate later.

Integrity Constraints

Integrity constraints are rules for limiting acceptable values and actions on the relations in order to facilitate maintaining the accuracy and integrity of data in the database.

There are 3 major integrity constraints.

- Domain constraints
- Entity constraints
- Referential constraints

Domain Constraints ensure all of the values in the column come from the same domain.

We'll be using MySQL 8.0. MySQL Documentation can be found at: https://dev.mysql.com/. MySQL has numerous data types. Your third assignment is to identify all of the numeric, date and time, and string types in MySQL.

The **entity integrity rule** states that no attribute that is part of a primary key can ever be **null.**

A **null** value indicates the value of the attribute is unknown or is not applicable.

Attributes that are not part of a primary key may have null values if when a row is created there is either no applicable value or it is not yet known.

> FaxNumber – may not be applicable (employee doesn't have a fax number)
> PreviousEmployerPhone – not known

A **referential integrity constraint** states that if there is a foreign key in one relation, either each foreign key value must match a primary key value in another relation, or the foreign key value must be null.

If you delete a row in a relation with a primary key that is used as a foreign key in another relation, you must update the relation with the foreign key by setting the foreign key value to null in all rows that reference the deleted primary key.

Well Structured Relations

A well-structured relation

1. contains minimal redundancy.
2. allows a user to insert, modify, and delete a row without errors or inconsistencies.

We call an error or inconsistency in a relation an **anomaly**. Three types of anomalies we are concerned about are:

- Insertion anomalies
- Deletion anomalies
- Modification anomalies

An **insertion anomaly** occurs when a row is created, and part of a primary key is not available thus violating the *entity integrity rule*.

We can't define an attribute as part of a primary key unless we are certain a value exists for it each time a row is inserted into the relation.

A **deletion anomaly** occurs when deleting a row in a table deletes information that the organization needed for other purposes unknown to the user who deleted the row.

For example, deleting a row in an Order relation might invalidate how the organization computes total sales.

A **modification anomaly** occurs when a value in one row of a relation is changed which causes an error in other rows.

> EMPLOYEE (EmpID, Name, …, salary, skill)
>
> If an employee is represented in a relation by multiple rows because she has multiple skills, if we modify the salary in one row, we have to modify the salary in all rows, otherwise we'll have an error.

We can prevent these anomalies in our relations by designing our relations carefully using a systematic approach.