Table of Contents

This document discusses database terms and concepts from a design point of view. This is not a database design class (that is CS 159A); but we cannot talk about tables without talking about design.

We need to talk about collections of tables/relations and the associations/relationships between these tables and part of this is terminology.

1. Entities

An **Entity** is something that we care about and for which we need to store data. This could be a person, a sales order, anything that the company is concerned with.

1.1. Entity Instance

The collection of data values about an individual person, a single sales order or an animal at the zoo.

1.2. Entity Set

The collection of entity instances that we are currently storing

1.3. Entity Class

The logical description of the entity

This describes the set of all possible entity instances that could ever be stored.

In terms of our databases, we can think of each row as an entity instance. In the employees table each row refers to a particular employee. We can think of the current collection of rows, the data contents of the table, as the entity set. This entity set will change as we add more rows, delete rows or change the values in the rows. The Create table statement defines the structure of the table and the rules for the data that can be entered. This is a logical description of the entity and we can describe it as the entity class.

2. Attributes, domains, tuples

2.1. Attribute

A characteristic of an entity that we need to store.

The collection of attributes defines the entity class. An attribute is represented as a column in a table.

2.2. Domain

A description of the legitimate values for an attribute

A domain can be given a name and used to determine the values for more than one attribute. A domain can also define the actions that can be performed with these attribute values. When you design a database, you use data types and constraints to implement the domain. Unfortunately most dbms do not implement domains as a database object.

2.3. Domain Compatible

Two data types are domain compatible if they can be converted into a common data type. This is a logical concept. In the Locations table we have an attribute for the city. In the Employees table we have an attribute for the employee last name. These are both character data but is would not make sense to talk about comparing these two attributes to see if an employee has the same last name as a city. These two attributes belong to different domains.

2.4. Tuple

A collection of attributes about a single entity instance

This is represented as a row in a table.

3. Relations

A **RELATION** is a collection of tuples. A relation is represented as a two-dimensional table. A relation has the following characteristics:

- Each relation has a name that is unique within the schema. We cannot create two tables with the same name.
- Each row contains information about a single entity instance. We should not have a row in a table that includes detailed information about an employee and also information about their department. These are two separate entities and the data is stored in two tables.
- No two rows are identical. It would not be helpful to have two rows in the employee table for the same employee. We use a primary key to avoid duplicate rows.
- Each column contains attribute data values. A column is defined with a name that represents an attribute we want to store and we should use that column for that purpose.
- All of the values in a column are from the same domain.

- Each column has a name that is unique within the relation. We cannot create two columns in the same table with the same name. We can use the same name for columns in different tables and we often do this for the columns that form the relationships between tables.
- Each cell contains a single data value. If we want to keep salary history for our employees, we would have to set up a new table for this. We should not try to keep salary history as a list of values in the salary attribute.
- The relation may not have repeating columns. Suppose we decided to keep track of our employees' phone numbers and we wanted to store multiple phone numbers for each employee. It would not be appropriate to create columns such as phone_1, phone_2, phone_3 etc.
- The order of the rows has no logical significance. We cannot have any logic in our table concepts that tries to reflect that a row in a table has some meaning because it follows another row. We can display the rows in a certain order, but we do not store the rows in any particular order. Tables reflect sets which are not ordered.
- The order of the columns has no logical significance. We are not required to have the column for the primary key be the first column in the table, although it often is the first. In fact we cannot meaningfully speak of the first column. We speak of columns by the name of the column.

3.1. Base Tables

These are the tables that store data about an entity.

3.2. Degree of a Table

The number of columns; it must be at least 1

4. Keys

We talk a lot about keys in databases; there are several types of keys.

4.1. Candidate Key

A candidate key is one which is never null and is always unique within its table. In the employee table we have two candidate keys, the employee ID and the SSN.

4.2. Primary Key, Identifier

The designated or chosen candidate key

Most tables have only one candidate key. If there are multiple candidate keys, then any one of them could be chosen to be the primary key. A table has only one primary key.

The employee_id is the best choice for the primary key of the Employee table. We do not want to duplicate an employee's SSN value in multiple tables for privacy reasons. A generated pk is sometimes the best choice.

4.3. Foreign Key

Foreign keys are defined with respect to two tables. The foreign key is the attribute in the child table that has values matching a candidate key's values (normally the primary key) in the parent table. The foreign key must have the same underlying domain as the associated candidate key. The foreign key values must either be null or have values that match existing values in the parent table.

4.4. Alternate Key

Any candidate key that was not chosen to be the primary key

4.5. Surrogate Key

Any key that is not composed of naturally occurring values. Often this is a system generated value.

4.6. Composite Key

A key that is made up of more than one attribute

4.7. Entity Integrity Rule

Each table must have a primary key, which cannot be null. If this is a composite primary key, then no component of the key can be null.

4.8. Rules for Foreign Keys

The foreign key and the referenced candidate key must have the same data type. They should have the same domain. A foreign key can be nullable.

The foreign key and the associated candidate key can have different names.

A row in the parent table can have zero, one, or more associated rows in the child table.

If the primary key of a table is composite, then any related foreign key must also be composite. You can have a table with a composite foreign key in which each of the attributes refers back to a different parent table.

It is possible that the parent table and the child table are actually the same table.

5. Relationships and participation in relationships

5.1. Relationship

A Relationship is an association between two or more relations based on data values in common between the tables. Relationships can be classified as One-to-One, One-to-Many, or Many-to-Many.

We have mentioned that we could have a table with data about books and another table with data about publishers. For our data each book has a single publisher and each publisher could have published multiple books. The relationship between the publisher table and the books table is One-to-Many. The term "many" does not mean that we must have multiple rows on the many side- only that we could. A one-to-many relationship allows a situation where we could have a row for a publisher and no books for that publisher.

We might want to keep data about the authors of our books. We could set up a table for authors. Some authors might write many books and some books have more than one author. The relationship between these books and authors is Many-to-Many.

We might want to store a picture of the cover art for our books. We decide to put these images in a different table for storage reasons. Assuming that a book has only one cover art picture the relationship between the tables is One-to-One. Each book has at most one picture and each picture belongs to at most one book.

5.2. Parent/Child

The terms parent and child are often used to describe the tables in a one to many (1:N) relationship. For example, one customer may place many orders. In this case, the customer table is the parent and the order table is the child. One order may be associated with multiple order lines. In this relationship, the parent is the order table and the child is the order line table.

5.3. Relationship Tables

These are the tables that are designed to allow the many-to-many relationships. A pure relationship table is "all key" but a table such as a BookAuthor table may also contain a column for the book identifier and a column for the author identifier. It might also include some columns that are attributes of the relationship.

5.4. Optional Relationship

An association that does not require the parent tuple to have any associated child tuple. For example, a customer can exist without have any associated orders.

5.5. Mandatory Relationship

An association where the parent tuple has to have at least one associated child tuple. We could have a business rule that an order cannot exists unless it has at least one line item.

5.6. Cardinality of a relationship

The number of instances in a child table that can (or must) be associated with an instance in the parent table. There may be a minimum and a maximum cardinality. We could have a business rule that an order can have at most 10 order line rows.

6. Schemas and schema objects

6.1. Relational Database

There are several definitions of a relational database that have different focuses. The first definition below is based on the dbms and has a dba attitude; the second is oriented towards a developer; the third definition is more of a system analyst's/designer's definition.

- A collection of data, schema, indexes, rules, and procedures for manipulating the data.
- A collection of tables, queries, forms, reports, and modules that are used for an application.
- A model of the user's model of some aspect of their business and business rules.

6.2. Schema

A named collection of tables and related objects, such as views, indexes, sequences, and user-defined types

This is one term that differs in the dbms particularly in what it means to create a schema.

With MySQL the Create Schema statement is considered a synonym for Create Database.

With SQL Server, a schema is a logical organization within a database. Tables and views can be placed inside the schema.

With Oracle, a schema is a logical organization that can contain tables, views, stored routines and other objects. Each user (in the sense of a user account) owns a schema which has the same name as the user's login name. A user can create objects in his own schema, assuming his account was granted with the appropriate permissions.

6.3. Table

The basic unit of data storage

Each table has a unique name relative to its schema.

6.4. View

A derived table; a virtual table; a stored, named query

A view can be used to limit the data that a user sees or to combine tables to supply the data values needed by a user.

6.5. Stored procedures, including Triggers

Code units that perform specific tasks

Procedures help implement an application that accesses and manipulates the database.

7. Database and table design

Good database design depends on good table design. Good table design reduces redundancy and avoids errors in adding new rows, deleting current rows and updating existing data.

These are some general rules for designing good tables for a relational database. We will formalize these later, when we discuss normalization.

- Store each fact one time only. One purpose of a database is to control redundancy, so you do not want to store facts more than once.
- Each attribute should contain a single value. For example, if you are storing information about books and a book has two co-authors, you cannot store both authors' names in the same attribute.

- Each attribute should be atomic. Do not have an attribute named Address that contains the street address, city, state, country, and postal code. This should be five separate attributes.
- Each row should have an attribute or combination of attributes that is a unique identifier- a primary key. Naturally occurring attributes, such as a customer name attribute, are not good primary keys because you cannot guarantee that they will be unique.
- The primary key should not be subject to change. For example, a phone number is generally not a good primary key since people change their phone numbers and the phone company can reassign phone numbers to other people.
- Every attribute in a row should contain information about the entity identified by the primary key. For example, a table with information about customers will store the customer's name and address values — but not information on the items that they purchased. That information will be stored in related tables.
- In general, every table in the database will have a relationship with at least one other table.
- Usually, a well-designed database will have many narrow tightly focused tables- rather than a few tables that try to store a lot of different columns of data.
- Database logic is based on set logic- a set is an unordered collection of items, all of the same type. A well designed table is a set of rows.
- SQL allows you to create tables which have duplicate rows; therefore SQL tables correspond to logical constructs called multi-sets or bags.
- A table is a set of rows; a row is a set of columns; a column is an atomic value which has a data type.
- With object-relational databases, the column value might be an object rather than a simple data value.