**Relational Database Design**

A relational database organizes data in tables (or relations). A table is made up of rows and columns. A row is also called a record (or tuple). A column is also called a field (or attribute). A database table is similar to a spreadsheet. However, the relationships that can be created among the tables enable a relational database to efficiently store huge amount of data, and effectively retrieve selected data.

Database Design Objective

A well-designed database shall eliminate Data Redundancy: the same piece of data shall not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies.

Ensure Data Integrity and Accuracy

Relational databases differ from other databases in their approach to organizing data and performing transactions. In an RDD, the data are organized into tables and all types of data access are carried out via controlled transactions.

Relational Database Design Process

Databases are usually customized to suit a particular application. No two customized applications are alike, and hence, no two databases are alike. Guidelines (usually in terms of what not to do instead of what to do) are provided in making these design decisions, but the choices ultimately rest on the designer.

Step 1: Define the Purpose of the Database (Requirement Analysis)

Gather the requirements and define the objective of your database.

Drafting out the sample input forms, queries and reports, often helps.

Step 2: Gather Data, organize in tables and Specify the Primary Keys

Choose one column (or a few columns) as the so-called primary key, which uniquely identify the each of the rows.

Most RDBMSs build an index on the primary key to facilitate fast search and retrieval.

The primary key is also used to reference other tables.

You have to decide which column(s) is to be used for primary key. The decision may not be straight forward but the primary key shall have these properties:

The values of primary key shall be unique (i.e., no duplicate value).

For example, customerName may not be appropriate to be used as the primary key for the Customers table, as there could be two customers with the same name.

The primary key shall always have a value. In other words, it shall not contain NULL.

Consider the followings in choose the primary key:

The primary key shall be simple and familiar, e.g., employeeID for employees table and isbn for books table.

The value of the primary key should not change. Primary key is used to reference other tables. If you change its value, you have to change all its references; otherwise, the references will be lost. For example, phoneNumber may not be appropriate to be used as primary key for table Customers, because it might change.

Let's illustrate with an example: a table customers contains columns lastName, firstName, phoneNumber, address, city, state, zipCode. The candidates for primary key are name=(lastName, firstName), phoneNumber, Address1=(address, city, state), Address1=(address, zipCode). Name may not be unique. Phone number and address may change. Hence, it is better to create a fact-less auto-increment number, say customerID, as the primary key.

Step 3: Create Relationships among Tables

A database consisting of independent and unrelated tables serves little purpose. The power of relational database lies in the relationship that can be defined between tables. The most crucial aspect in designing a relational database is to identify the relationships among tables. The types of relationship include:

one-to-many

many-to-many

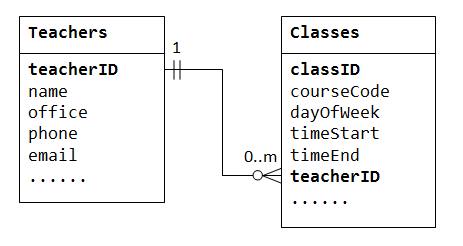
one-to-one

One-to-Many

In a "class roster" database, a teacher may teach zero or more classes, while a class is taught by one (and only one) teacher. In a "company" database, a manager manages zero or more employees, while an employee is managed by one (and only one) manager. In a "product sales" database, a customer may place many orders; while an order is placed by one particular customer. This kind of relationship is known as one-to-many.

One-to-many relationship cannot be represented in a single table. For example, in a "class roster" database, we may begin with a table called Teachers, which stores information about teachers (such as name, office, phone and email). To store the classes taught by each teacher, we could create columns class1, class2, class3, but faces a problem immediately on how many columns to create. On the other hand, if we begin with a table called Classes, which stores information about a class (courseCode, dayOfWeek, timeStart and timeEnd); we could create additional columns to store information about the (one) teacher (such as name, office, phone and email). However, since a teacher may teach many classes, its data would be duplicated in many rows in table Classes.

To support a one-to-many relationship, we need to design two tables: a table Classes to store information about the classes with classID as the primary key; and a table Teachers to store information about teachers with teacherID as the primary key. We can then create the one-to-many relationship by storing the primary key of the table Teacher (i.e., teacherID) (the "one"-end or the parent table) in the table classes (the "many"-end or the child table), as illustrated below.



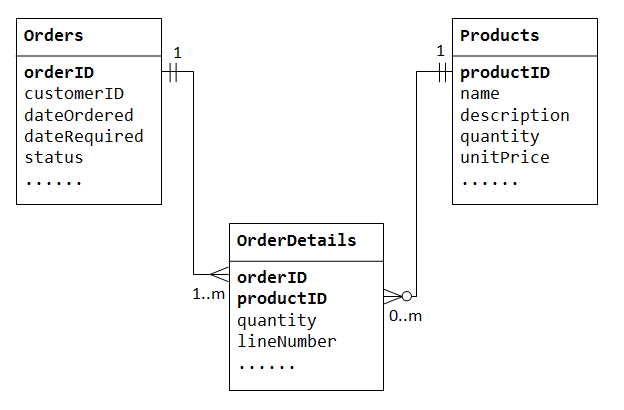
The column teacherID in the child table Classes is known as the foreign key. A foreign key of a child table is a primary key of a parent table, used to reference the parent table.

Many-to-Many

In a "product sales" database, a customer's order may contain one or more products; and a product can appear in many orders. In a "bookstore" database, a book is written by one or more authors; while an author may write zero or more books. This kind of relationship is known as many-to-many.

Let's illustrate with a "product sales" database. We begin with two tables: Products and Orders. The table products contains information about the products (such as name, description and quantityInStock) with productID as its primary key. The table orders contains customer's orders (customerID, dateOrdered, dateRequired and status). Again, we cannot store the items ordered inside the Orders table, as we do not know how many columns to reserve for the items. We also cannot store the order information in the Products table.

To support many-to-many relationship, we need to create a third table (known as a junction table), say OrderDetails (or OrderLines), where each row represents an item of a particular order. For the OrderDetails table, the primary key consists of two columns: orderID and productID, that uniquely identify each row. The columns orderID and productID in OrderDetails table are used to reference Orders and Products tables, hence, they are also the foreign keys in the OrderDetails table.



The many-to-many relationship is, in fact, implemented as two one-to-many relationships, with the introduction of the junction table.

An order has many items in OrderDetails. An OrderDetails item belongs to one particular order.

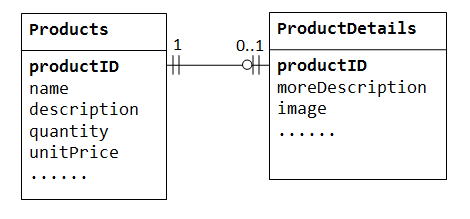
A product may appears in many OrderDetails. Each OrderDetails item specified one product.

One-to-One

In a "product sales" database, a product may have optional supplementary information such as image, moreDescription and comment. Keeping them inside the Products table results in many empty spaces (in those records without these optional data). Furthermore, these large data may degrade the performance of the database.

Instead, we can create another table (say ProductDetails, ProductLines or ProductExtras) to store the optional data. A record will only be created for those products with optional data. The two tables, Products and ProductDetails, exhibit a one-to-one relationship. That is, for every row in the parent table, there is at most one row (possibly zero) in the child table. The same column productID should be used as the primary key for both tables.

Some databases limit the number of columns that can be created inside a table. You could use a one-to-one relationship to split the data into two tables. One-to-one relationship is also useful for storing certain sensitive data in a secure table, while the non-sensitive ones in the main table.



Step 4: Choose appropriate Column Data Types

You need to choose an appropriate data type for each column. Commonly data types include: integers, floating-point numbers, string (or text), date/time.

Step 5: Refine & Normalize the Design

For example, adding more columns, create a new table for optional data using one-to-one relationship, split a large table into two smaller tables.

Normalization

Apply the so-called normalization rules to check whether your database is structurally correct and optimal.

Step 6: Apply Integrity Rules

You should also apply the integrity rules to check the integrity of your design:

Entity Integrity Rule: The primary key cannot contain NULL. Otherwise, it cannot uniquely identify the row. For composite key made up of several columns, none of the column can contain NULL. Most of the RDBMS check and enforce this rule.

Referential Integrity Rule: Each foreign key value must be matched to a primary key value in the table referenced (or parent table).

You can insert a row with a foreign key in the child table only if the value exists in the parent table.

If the value of the key changes in the parent table (e.g., the row updated or deleted), all rows with this foreign key in the child table(s) must be handled accordingly.

Features of Good Relational Database Design

It should reflect real-world structure of the problem

It can represent all expected data with in time

It should avoid redundant storage of data items

It should provide efficient access to data

It supports the maintenance of data integrity with in time

It can be consistent, and easy to understand