Relational Database Design

Table of Contents

Relational Database Design

Welcome to Relational Database Basics and Design. This class was developed to introduce the concepts used to create and maintain relational database systems. A *relational database system (RDBS)* is one in which two or more linked tables are used to track information. A *table* is a list of data about a specific subject.

Flat File Databases vs. Relational Databases

You may already be familiar with a flat file database. A flat file database (FFDB) is one in which a single table is used to hold all the information for a particular application. Flat file databases are commonly used to track very simple lists of information such as a personal address list. Commonly though, what seems like a simple list of information quickly becomes a complicated system. This is where good planning and a relational database become very useful.

Let's set up a simple Campus Calendar database as an example of a flat file database system. In this calendar you are going to track events that are happening on campus.

This appears at first glance to be a simple list of information but let's see what happens as you start to enter the information. Say you have information about 50 events to add to this calendar but that each event has at least two different sponsors. Your calendar would have a minimum of 100 records to start (50 events multiplied by 2 sponsors equals 100 records.) Now if an event were to be rescheduled you would have to change at least 2 records per event to ensure that the calendar was properly updated. How do you enter information about events that happen more than once, like a concert on three consecutive nights? Also what happens if a sponsor changed their name or phone number? First you would have to locate all the events that were sponsored by them and then change the information in each record (there could be up to 50 records that need to be changed). Think about what adding ticket or contact information would involve.

As you can see from the Campus Calendar database, information tends to be repeated within flat file databases. This makes flat file databases hard to maintain and wasteful of resources such as disk space. In the Campus Calendar database, the sponsor name, address and phone number were unnecessarily repeated for each event they sponsored and date that the event occurred. Small changes such as a sponsor name change can be very difficult when it has been repeated for hundreds of records. Relational databases avoid these problems by avoiding redundancies of information.

Term Definitions

Relational databases employ the same basic database terminology as flat file databases. But because they have the ability to describe relations between tables, relational databases do use some different terms.

Database

A collection of tables

Table

An organization of records

Records

Records group fields into logical units. For example, all fields dealing with a particular event, person, department or project are included in the same record.

Fields

These are the atoms of a database. They are the smallest, discreet pieces of information that we want to store. Examples include a sponsor name, event name, surname, start date, department name, salary, etc.

To visualize the relationship between fields and records you can also think of records as the rows and fields as the columns of a table. See Figure 1 for an example.

Fields ↓

Record →

EmployeeID *	Name	Salary	Department
1	Liz	2000	3
2	Joe	1795	1
3	Susan	3500	2

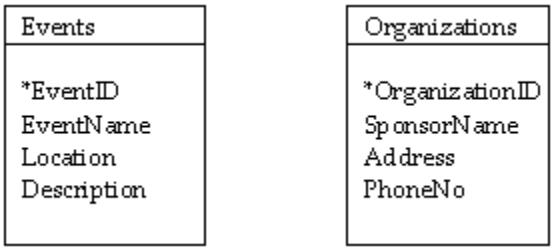
Figure 1

Primary Key
A field (or a combination of fields) that uniquely identifies a record. Employee ID is the primary key in Figure 1.

Relationship
A connection between two tables based on a common field. This sounds simple and really it is. An important thing to remember is that the relationships are just as much part of the database as the fields, records and tables. The relations provide information.

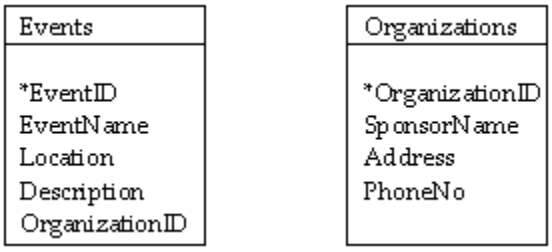
An Example Relational Database

Let's look at just the Event and Sponsor relationship in the Campus Calendar database. There are two tables in this relationship Events and Organizations. (We'll talk more about the choice of "Organizations" for the table name later.) The Events table contains the basic information about an event; the Organization table holds information about the organizations that sponsor events. Each has a primary key, EventID and OrganizationID that is just a unique number assigned to each record.



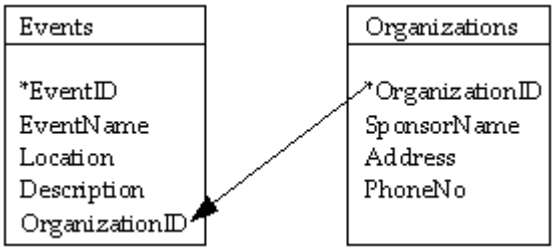
* Indicates the Primary key

To relate these two tables we first want to define the relationship, events are sponsored by an organization. Which indicates that somehow we need to associate the Organization table with the Events table. We can do this by placing an OrganizationID in each Event record.



* Indicates the Primary key

Looking at these two tables, it becomes obvious quite quickly where at least one relationship exists. Both the Events and Organization tables have a field called OrganizationID. An event will have at least one sponsor, so a link to the Organizations table is provided through the OrganizationID field.



* Indicates the Primary key

Creating Relationships

An important thing to know and remember when creating relationships between tables is that *links can only be created if one of the sides of the link is a primary key.* The other side of the link is known as the *foreign key*. This means our Campus Calendar database can only link the Organizations table to the Events table by the OrganizationID; no other field would work since it is not the primary key of the Organizations table.

One-to-Many Relationships
The relationship between the Events and Organizations tables shown earlier is an example of a **One-to-Many** relationship. For every event, there is only *one* sponsor, but for every sponsor there are *many* events. In a One-to-Many relationship, like the Events and Organizations, the table on the "Many" side (the Events table) has a foreign key, OrganizationID that corresponds to the primary key, OrganizationID, of the Organizations table.

You can see how using this relationship has improved the Campus Calendar database. Rather than each event record storing information about the sponsor, just one field is needed. This is a huge saving of storage space and update time for the Database Administrator.

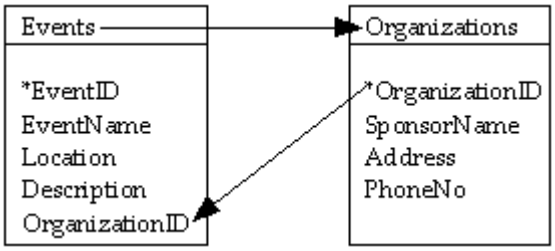
Just think, for example, if an organization changes its name frequently. In a flat file database, if this happened someone would have to search the database to find all the events sponsored by the organization and make the change for each record in the Events table. Imagine that this person makes a few typographical errors, or has in the past and therefore misses an event they should have

updated. Suddenly the event's sponsor information would be incorrect!

With the relational database, a change to an organization's name would not even necessitate a change to the Events table. All that needs changing would be the OrganizationName field in the Organization table.

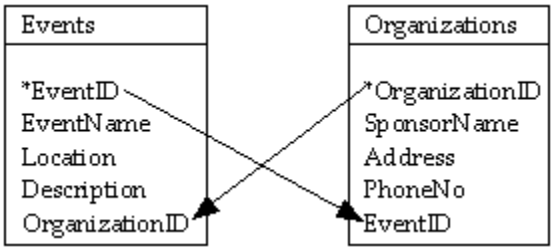
Many-to-Many Relationships

Another type of relationship between the two tables is a Many-to-Many relationship. **Many-to-Many** relationships are really just One-to-Many relationships going both directions between the tables. Typically Many-to-Many relationships are handled by using a third table to relate the other two. Take for example our Campus Calendar database. Let's say that an event can have more than one sponsor. This means that an organization can sponsor more than one event **and** an event can have more than one organization sponsoring it, a one-to-many relationship in both directions.



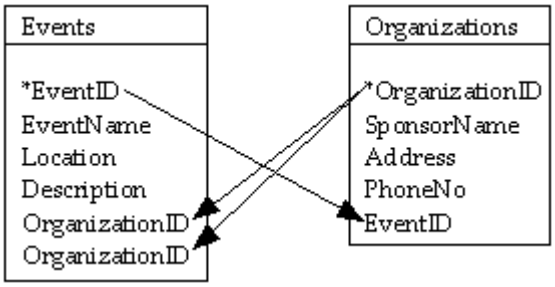
* Indicates the Primary key

One way we could handle this is by adding the EventID to the Organization table. But as you can see this causes problems. First, we are back to repeating organization information since the organization record would have to be copied for each event it sponsored. The event information is repeated as well since an event record needs to be copied for each organization that sponsors it. Both of these problems lead to problems with the original Events to Organization link since there could be multiple OrganizationIDs for an organization and multiple EventIDs for a specific event.



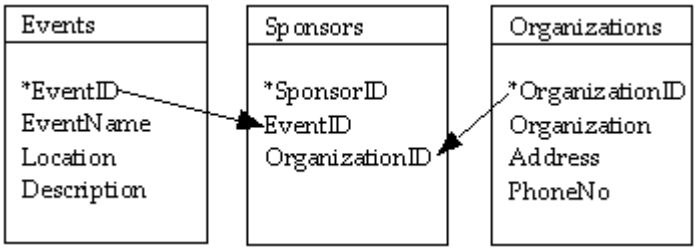
* Indicates the Primary key

Another way to deal with multiple sponsors of an event is to add another OrganizationID to the Events table. This could work but limits us to two sponsors per event. What about events that have 3 or even 10 sponsors, how do you enter the sponsors for those events?



* Indicates the Primary key

The best way to handle the many-to-many relationship between the Events and Organizations tables is to create a third table called Sponsors. This table exists only to define the many-to-many relationship between Events and Organizations. It has only two fields, EventID and OrganizationID, because these are the only fields that are needed to define the relationship.



* Indicates the Primary key

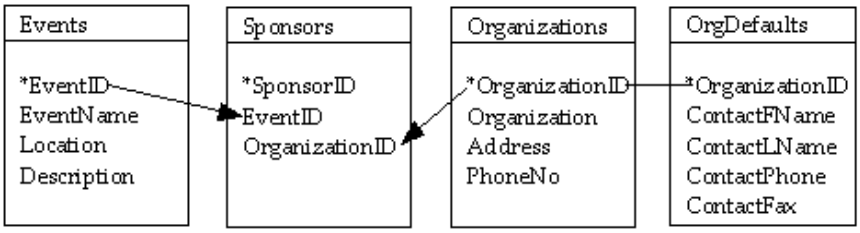
Notice that both the EventID and OrganizationID fields are used to create the primary key for the Sponsors table. This ensures that each record in the Sponsors table is unique since it does not make sense to list an organization as a sponsor of an event more than once. Also note that in the Sponsors table we really have two one-to-many relationships, one between Sponsors and Events and the other between Sponsors and Organizations. These relationships are one-to-many since a record in the Sponsors table can only point to one record at a time in both the Events and Organizations tables. But looking the other way, from the Events or Organizations tables, the EventID and OrganizationIDs are repeated many times in the Sponsors table. Hence we have one-to-one relationships in

both directions and a many-to-many relationship between the Events and Organizations tables.

One-to-One Relationships

There is a third type of relationship called a **One-to-One** relationship. In this type of relationship, there are two tables that are specifically linked; for each record in one table there is one and only one record in the other table. Another way to think of this is to realize that their primary keys are essentially identical. You might be wondering why there are two tables in this case. Usually the reason to split the data into two tables has to do with security, size of the data, frequency of use or just for organizational purposes. We will discuss this in detail later.

Let's use as an example of this adding default contact information in our Campus Calendar database. The default settings are saved per organization and include some generic information about the events that they sponsor. Because this information is not really about the organization we will store it in its own table, OrgDefaults, and not in the Organizations table.



* Indicates the Primary key

So, for every record in the OrgDefault table there is only one corresponding record in the Organizations table, i.e., all organizations only have one set of defaults. You see how this has improved our database organization. We have logically grouped like-information, defaults, into one place and left a more simplistic record in the Organization table.

Exercise

1. Add date and time information to the to the Campus Calendar database. Keep in mind that an event can happen on more than one date.

2. List the tables and fields you would need to track software. The information you want to know about is the software title, a description, cost, publisher and publisher address. Remember that a publisher can produce more than one software package.

Designing a Relational Database System

Taking time initially to plan your relational database will save you time and frustration in the long run. When working on the design phase, don't worry about making mistakes or leaving things out. A good design is flexible enough to accommodate changing requirements.

There are five basic guidelines to keep in mind when designing a relational database. The process applying these guidelines is commonly referred to as *normalizing* a relational database.

1. **Determine the purpose of your database.** This will help you determine the information that you want to store and track.

2. **Determine the tables you need.** Decide what information you want to keep in each table of your RDBMS. Remember a table should hold information about one subject.

3. **Determine the fields you need.** Decide what information you want to keep in each table and how small that information should be made. For example, if you need to store a name, you must decide whether you need to break it into first name, last name and middle name or if just name will suffice.

4. **Determine the relationships of the tables.** Look at each table and decide how the data in it is related to the other tables. Add the necessary tables and/or fields to clarify these relationships.

5. **Refine and test your design**. Create the tables and enter some data. Test to see if you can produce the reports you required. If not, reevaluate your design using the other four guidelines.

Normalizing a Database

Let's use the five steps described above to create a relational database to track class information.

Determine the purpose of your database

The first step in the process of normalizing a database is to define the purpose of the relational database and how it will be used. This tells you what information you need to be able to get from the database and starts the process of dividing that information into subjects.

During this process you should be talking to the people who will be maintaining the database along with those who will use reports from the database. Sketch out the reports and forms you want to be able to produce. Also look at the existing methods that are being used to track the information. Talk to the people that use it, be sure to get their opinions about its strengths and weaknesses.

The more information you gather at this point, the easier the other steps will be.

Example
Suppose the Class database you are setting up is going to be used to track students, classes and instructors. Students will use the database to look up classes. The staff and faculty members will use the database to develop class lists and schedules.

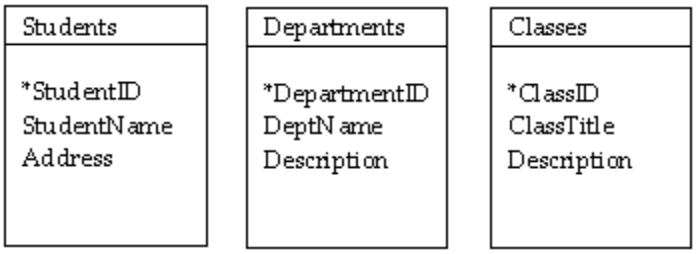
One place to start is to talk to students, staff and faculty members to get their input about what types of things they would like to be able to do with the system. Find out the questions that they would like answered such as, is there a particular class offered? If so, when and where? What requirements does a class have or fulfill? When was the last time a class was offered and what was the enrollment? Etc. Also ask them what lists of information they would like to see produced.

Next, gather up any information that exists about what is currently being done. Look at how data got into the system, i.e. forms. Talk to the people who have entered the data and use the system especially noting what they perceive as its limitations, strengths and weaknesses. Look at any reports or other documents that may have been produced by the system, especially transcripts, fee statements, list of classes by instructor, time locations, etc.

Determining the Tables You Need

Look at all the information that you want to get out of your relational database and divide it into subjects. This is not always as obvious as it seems. One approach is to look at the reports you've gathered and determine what each fact on the report is actually about. For example the student name on a transcript is about a student not a class. This would suggest that you need a separate table for student information.

Example
While gathering information during step one of the Class relational database design process you receive a copy of a fee statement. On the fee statement there was information about the following subjects: student, class, instructor and department. From that list you may determine that the following tables are necessary.



Determine The Fields You Need

In this step, you determine how much information you need to track about a person, place or thing. Think of the fields as characteristics of the person, place or thing. Make sure that the fields only describe the subject of this table. Foreign keys will be added later.

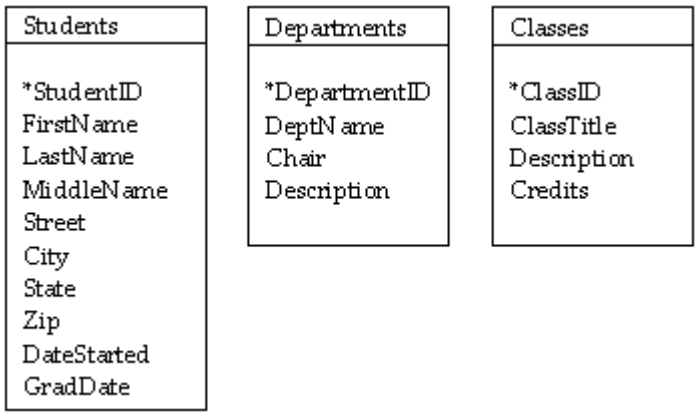
While doing this, make sure that you include all the information you will need and that you break it down if necessary. For example, if you need to track names and sort them by first and last name, use at least two fields to hold the name.

You will also identify primary keys in this step. Primary keys contain information that uniquely identifies a specific record; such as a student ID number, social security number, or class number. It is very common to use a number field as the primary field. One reason this is done is because computers can compare numbers quicker than text strings. Numbers also avoid duplicate values that

could arise from using names as the primary field. Remember duplicate values are not allowed in a primary field.

Example

A closer look at each table in the Class database and the reports that were produced from the old system show that you need to be able to sort and search on first and last names of students, class names and department names. From this information you may now adjust your table structure and determine the primary keys, indicated by an asterisk.

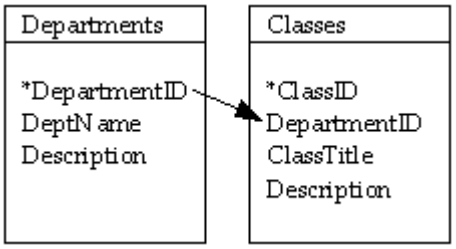


* Indicates the Primary key

Determine The Relationships Of The Tables

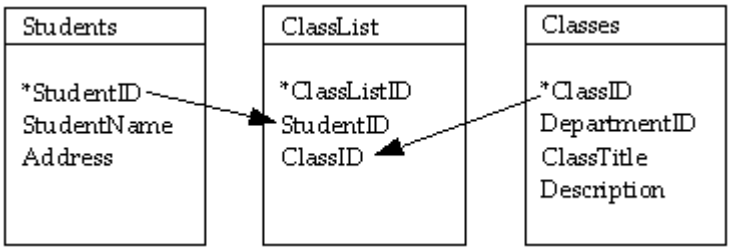
In this step you will define how each table in your relational database is related to the others. To do this you need to identify the type of relationship the tables have to each other, one–to–many, many–to–many or one–to–one. Once defined you can create the relationship by either creating a foreign key (one–to–many and one–to–one) or creating another table (many–to–many).

Example
In our Class database let's say only one department can offer a class. The Departments and Classes tables would then have a one to many relationship, a class can only have one department but a department can offer many classes. The primary field from the Departments table, DeptID, would be added to the Classes table.



* Indicates the Primary key

Also the Students and Classes tables have a many–to–many relationship since a student can take more than one class and a class has more than one student. To handle this we will create another table called Class List. This table would have use the primary fields from both tables, StudentID and ClassID, to create the relationship between the Students and Classes tables. Combining these two fields would create the primary field of the Class List table.



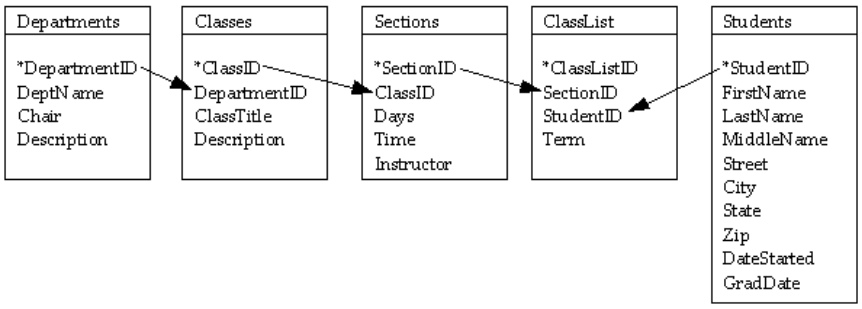
* Indicates the Primary key

Refine And Test Your Design

Once you have mapped your tables, fields and relationships it is time to create a mini version of your database. You will use this version to test the system by trying to create the reports and forms that are required.

During this testing you should look for any missing or forgotten information along with redundancies of information. You can also verify that the primary field you selected for each table works properly and is easy to use. You should also check that you have not inadvertently limited the amount of information you can enter, which may cause problems.

Example
Let's say that we have set up the Class database just as it has been defined so far. As we start to enter data into the test system we realize that we have forgotten to account for multiple sections of a class. If not corrected, this omission would create a number of redundancies in the database when implemented. To correct this, add a new table, Sections, that will hold the information about the time and location of a particular class section. The Sections table would be related to the Classes and Class List tables.



* Indicates the Primary key

Take Home Practice

Develop a relational database to keep track of your kitchen supplies and family recipes. It should be able generate a grocery list for each recipe and create a daily menu. The system should also be flexible enough to provide for enhancements such as tracking individual's food likes and dislikes.

Your RDBMS Design

Recipe RDBMS Design