4.4: Designing a Database

4.4.1: Designing a Database

Suppose a university wants to create a database to track participation in student clubs. After interviewing several people, the design team learns that implementing the system is to give better insight into how the university funds clubs. This will be accomplished by tracking how many members each club has and how active the clubs are. The team decides that the system must keep track of the clubs, their members, and their events. Using this information, the design team determines that the following tables need to be created:

- Clubs: this will track the club name, the club president, and a short description of the club.
- Students: student name, e-mail, and year of birth.
- Memberships: this table will correlate students with clubs, allowing us to have any given student join multiple clubs.
- Events: this table will track when the clubs meet and how many students showed up.

Now that the design team has determined which tables to create, they need to define the specific information that each table will hold. This requires identifying the fields that will be in each table. For example, Club Name would be one of the fields in the Clubs table. First Name and Last Name would be fields in the Students table. Finally, since this will be a relational database, every table should have a field in common with at least one other table (in other words: they should have a relationship with each other).

To properly create this relationship, a primary key must be selected for each table. This key is a unique identifier for each record in the table. For example, in the Students table, it might be possible to use students' first names to identify them uniquely. However, it is more than likely that some students will share the last name (like Mike, Stefanie, or Chris), so a different field should be selected. A student's email address might be a good choice for a primary key since e-mail addresses are unique. However, a primary key cannot change, so this would mean that if students changed their email addresses, we would have to remove them from the database and then re-insert them – not an attractive proposition. Our solution is to create a value for each student — a user ID — that will act as a primary key. We will also do this for each of the student clubs. This solution is quite common and is the reason you have so many user IDs!

You can see the final database design in the figure below:

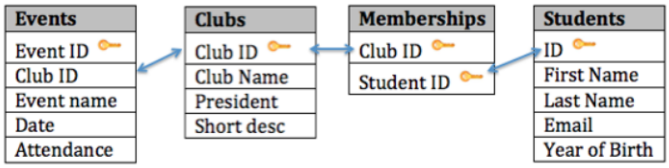

Figure 4.4.1: Data design flow. Image by David Bourgeois, Ph.D. is licensed under CC BY 4.0

With this design, not only do we have a way to organize all of the information we need to meet the requirements, but we have also successfully related all the tables together. Here's what the database tables might look like with some sample data. Note that the Memberships table has the sole purpose of allowing us to relate multiple students to multiple clubs.

Club ID	Club name	President	Short desc
1	Cheese Club	14	To talk about our love of cheese.
2	Chess Club	1	To learn how to become better chess players.
3	Archery Club	6	To compete in archery tournaments.

Table: Clubs

Figure 4.4.2 Table: Clubs. Image by David Bourgeois, Ph.D. is licensed under CC BY 4.0

ID	First Name	Last Name	Email	Year of Birth
1	Peter	Lee	plee@university.edu	1992
2	Jonathan	Edwards	jedwards@university.edu	1994
3	Marilyn	Johnson	mjohnson@university.edu	1993
6	Joe	Kim	jkim@university.edu	1992
12	Haley	Martinez	hmartinez@university.edu	1993
14	John	Mfume	jmfume@university.edu	1991
15	David	Letty	dletty@university.edu	1995

Table: Students

Figure 4.4.3 Table: Students. Image by David Bourgeois, Ph.D. is licensed under CC BY 4.0

Club ID	Student ID
1	1
1	2
1	14
2	1
2	3
2	5
2	6
3	1
3	6
3	12
3	14
3	15

Table: Memberships

Figure 4.4.4 Table: Memberships. Image: by David Bourgeois, Ph.D. is licensed under CC BY 4.0

4.4.2: Normalization

When designing a database, one important concept to understand is normalization. In simple terms, to normalize a database means to design it in a way that:

- Reduces redundancy of data between tables easier mapping
- Takes out inconsistent data.
- Information is stored in one place only.
- Gives the table as much flexibility as possible.

In the Student Clubs database design, the design team worked to achieve these objectives. For example, to track memberships, a simple solution might have been to create a Members field in the Clubs table and then list all of the members' names. However, this design would mean that if a student joined two clubs, then his or her information would have to be entered a second time. Instead, the designers solved this problem by using two tables: Students and Memberships.

In this design, when a student joins their first club, we must add the student to the Students table, where their first name, last name, e-mail address, and birth year are entered. This addition to the Students table will generate a student ID. Now we will add a new entry to denote that the student is a specific club member. This is accomplished by adding a record with the student ID and the club ID in the Memberships table. If this student joins a second club, we do not have to duplicate the student's name, e-mail, and birth year; instead, we only need to make another entry in the Memberships table of the second club's ID and the student's ID.

The Student Clubs database design also makes it simple to change the design without major modifications to the existing structure. For example, if the design team was asked to add functionality to the system to track faculty advisors to the clubs, we could easily accomplish this by adding a Faculty Advisors table (similar to the Students table) and then adding a new field to the Clubs table to hold the Faculty Advisor ID.

4.4.3: Data Types

When defining the fields in a database table, we must give each field a data type. For example, the field Birth Year is a year, so it will be a number, while First Name will be text. Most modern databases allow for several different data types to be stored. Some of the more common data types are listed here:

- Text: for storing non-numeric data that is brief, generally under 256 characters. The database designer can identify the maximum length of the text.
- Number: for storing numbers. There are usually a few different number types selected, depending on how large the largest number will be.
- Yes/No: a special form of the number data type that is (usually) one byte long, with a 0 for “No” or “False” and a 1 for “Yes” or “True.”
- Date/Time: a special form of the number data type can be interpreted as a number or a time.
- Currency: a special form of the number data type that formats all values with a currency indicator and two decimal places.
- Paragraph Text: this data type allows for text longer than 256 characters.
- Object: this data type allows for data storage that cannot be entered via keyboards, such as an image or a music file.

The importance of properly defining data type is to improve the data's integrity and the proper storing location. We must properly define the data type of a field, and a data type tells the database what functions can be performed with the data. For example, if we wish to perform mathematical functions with one of the fields, we must tell the database that the field is a number data type. So if we have a field storing birth year, we can subtract the number stored in that field from the current year to get age.

Allocation of storage space for the defined data must also be identified. For example, if the First Name field is defined as a text(50) data type, fifty characters are allocated for each first name we want to store. However, even if the first name is only five characters long, fifty characters (bytes) will be allocated. While this may not seem like a big deal, if our table ends up holding 50,000 names, we allocate 50 * 50,000 = 2,500,000 bytes for storage of these values. It may be prudent to reduce the field's size, so we do not waste storage space.

In defining a data field in a database, we have to give all specific requirements for computers to know what constitutes a valid entry entered by a user. Data entry errors must be caught immediately to avoid having bad data in the database since it can be replicated by different organizations, which spreads the mistake far and wide.

Example 4.4.1

A first-name field is defined to contain up to 10 letters long from A to Z or a to z. We have to be very specific in denoting upper or lower case letters. A valid entry for a field containing a first name can be any upper or lower letter, up to 10 letters.

Question: Is 'space' or 'hyphen' allowed in the above definition?

Solution

No, since we did not define it. If we wish to have these two characters, we will modify it to say: they can contain up to 10 letters long from A to Z, or a to z, '', "or - (hyphen).

Example 4.4.2

How would you define a date if we want to accept DD/MM/YYYY?

Solution

We would define the date field as follows:

Date field: starting with two digits, where DD can be from 01 to 31, followed by a /, followed by two numbers that can be from 01 to 12, followed by a /, and four digits that can be from 1900 to 9999.
