Databases course book

Version 4.1 (8 October 2013)
Free University of Bolzano Bozen – Paolo Coletti

Introduction

This book contains the relational databases and Access course's lessons held at the Free University of Bolzano Bozen. The book is divided into levels, the level is indicated between parenthesis after each section's title:

- students of Information Systems and Data Management 3 credits course use level 1;
- students of Information Systems and Data Management 5 credits course use levels 1, 2 and 3;
- students of Computer Science and Information Processing course use levels 1, 2 and 3;
- students of Advanced Data Analysis course use levels 2 and 5.

This book refers to Microsoft Access 2010, with referrals to 2007 and 2003 in footnotes, to MySQL Community Server version 5.5 and to HeidiSQL version 7.0.0.

This book is in continuous development, please take a look at its version number, which marks important changes.

Disclaimers

This book is designed for novice database designers. It contains simplifications of theory and many technical details are purposely omitted.

Table of Contents

1. Relational databases (level 2)

This chapter presents the basic ideas and motivations which lie behind the concept of relational database. Readers with previous experience in building schemas for relational databases can skip this part.

A relational database is defined as a collection of tables connected via relations. It is always a good idea to have this table organized in a structured was that is called normal form.

1.1. Database in Normal Form

The easiest form of database, which can be handled even by Microsoft Excel, is a single table. To be a database in normal form, the table must satisfy some requisites:

1. the first line contains the headers of the columns, which univocally define the content of the column. For example:

Student number	Name	Surname	Telephone
2345	Mary	Smith	0471 234567

2. each column contains only what is indicated in its header. For example, in a column with header "telephone number" we may not put two numbers or indication on the preferred calling time, such as in the second row of this table:

Student number	Name	Surname	Telephone
2345	Mary	Smith	0471 234567
2348	John	McFlurry	0471 234567 or 337 8765432

3. each row refers to a single object. For example, there may not be a row with information on several objects or on a group of objects, such as in the second row of this table:

Student number	Name	Surname	Degree course
2345	Mary	Smith	Economics and Management
Starting with 5			Logistics and Production Engineering

4. rows are independent, i.e. no cell has references to other rows, such as in the second row of this table:

Student number	Name	Surname	Notes
2345	Mary	Smith	
2376	John	Smith	is the brother of 2345

5. rows and columns are disordered, i.e. their order is not important. For example, these four tables are the same one:

Student number	Name	Surname
2345	Mary	Smith
2376	John	McFlurry

Student number	Name	Surname
2376	John	McFlurry
2345	Mary	Smith

Name	Student number	Surname
Mary	2345	Smith
John	2376	McFlurry

Surname	Student number	Name
McFlurry	2376	John
Smith	2345	Mary

6. cells do not contain values which can be directly calculated from cells of the same row, such as in the last column of this table:

Student number	Name	Surname	Tax 1st semester	Tax 2nd semester	Total tax
2345	Mary	Smith	550 €	430 €	980 €
2376	John	McFlurry	450 €	0 €	450 €

Database rows are called records and database columns are called fields.

Single table databases can be easily handled by many programs and by human beings, even when the table is very long or with many fields. There are however situations in which a single table is not an efficient way to handle the information.

1.1.1. Primary key

Each table should have a primary key, which means a field whose value is different for every record. Many times primary key has a natural candidate, as for example student number for a students' table, tax code for a citizens table, telephone number for a telephones table. Other times a good primary key candidate is difficult to detect, for example in a cars' table the car name is not a primary key since there are different series and different motor types of the same car. In these cases it is possible to add an extra field, called ID or surrogate key, with a progressive number, to be used as primary key. In many database programs this progressive number is handled directly by the program itself.

It is also possible to define as primary key several fields together, for example in a people table the first name together with the last name, together with place and date of birth form a unique sequence for every person. In this case the primary key is also called composite key or compound key. On some database management programs however handling a composite key can create problems and therefore it is a better idea to use, in this case, an ID.

1.2. Relations

1.2.1. Information redundancy

In some situations trying to put the information we need in a single table database causes a duplication of identical data which can be called information redundancy. For example, if we add to our students' table the information on who is the reference secretary for each student, together with other secretary's information such as office telephone number, office room and timetables, we get this table:

Student number	Name	Surname	Secretary	Telephone	Office	Time
2345	Mary	Smith	Anne Boyce	0471 222222	C340	14-18
2376	John	McFlurry	Jessy Codd	0471 223334	C343	9-11
2382	Elena	Burger	Jessy Codd	0471 223334	C343	9-11
2391	Sarah	Crusa	Anne Boyce	0471 222222	C340	14-18
2393	Bob	Fochs	Jessy Codd	0471 223334	C343	9-11

Information redundancy is not a problem by itself, but:

- storing several times the same information is a waste of computer space (hard disk and memory), which for a very large table, has a bad impact on the size of the file and on the speed of every search or sorting operation;
- whenever we need to update a repeated information (e.g. the secretary changes office), we need to do a lot of changes;
- manually inserting the same information several times can lead to typing (or copying&pasting) mistakes, which decrease the quality of the database.

In order to avoid this situation, it is a common procedure to split the table into two distinct tables, one for the students and another one for the secretaries. To each secretary we assign a unique code and to each student we indicate the secretary's code.

Students			
Student number	**Name**	**Surname**	**Secretary**
2345	Mary	Smith	1
2376	John	McFlurry	2
2382	Elena	Burger	2
2391	Sarah	Crusa	1
2393	Bob	Fochs	2

Secretaries					
Secretary code	**Name**	**Surname**	**Telephone**	**Office**	**Time**
1	Anne	Boyce	0471 222222	C340	14-18
2	Jessy	Codd	0471 223334	C343	9-11

In this way the information on each secretary is written and stored only once and can be updated very easily. The price for this is that every time we need to know who is a student's secretary we have to look at its secretary code and find the corresponding code in the Secretaries table: this can be a long and frustrating procedure for a human being when the Secretaries table has many records, but is very fast task for a computer program which is designed to quickly search through tables.

1.2.2. Empty fields

Another typical problem which arises with single table databases is the case of many empty fields. For example, if we want to build an address book with the telephone numbers of all the people, we will have somebody with no telephone numbers, many people with a few telephone numbers, and some people with a lot of telephone numbers. Moreover, we must also take into consideration that new numbers will probably be added in the future to anybody.

If we reserve a field for every telephone, the table looks like this:

Name	Surname	Phone1	Phone2	Phone3	Phone4	Phone5	Phone6	Phone7
Mary	Smith	0412345						
John	McFlurry	0412375	3396754					
Elena	Burger	0412976	3397654	0436754	3376547	0487652	3387655	0463456
Sarah	Crusa	0418765	0412345					
Bob	Fochs	0346789	0765439	3376543				

As it is clear, if we reserve several fields for the telephone numbers, a lot of cells are empty. The problems of empty cells are:

- an empty cell is a waste of computer space;
- there is a fixed limit of fields which may be used. If a record needs another field (for example, Elena Burger gets another telephone number) the entire structure of the table must be changed;
- since all these fields contain the same type of information, it is difficult to search whether an information is present since it must be looked for in every field, including the cells which are empty.

In order to avoid this situation, we again split the table into two distinct tables, one for the people and another one for their telephone numbers. This time, however, we assign a unique code to each person and we build the second table with combinations of person-telephone.

People		
Person code	**Name**	**Surname**
1	Mary	Smith
2	John	McFlurry
3	Elena	Burger
4	Sarah	Crusa
5	Bob	Fochs

Telephones	
Owner	**Number**
1	0412345
2	0412375
2	3396754
3	0412976
3	3397654
3	0436754
3	3376547
3	0487652
3	3387655
3	0463456
4	0418765
4	0412345
5	0346789
5	0765439
5	3376543

Even though it seems strange, each person's code appears several times in the Telephones table. This is correct, since Telephones table uses the exact amount of records to avoid having empty cells: people appear as many times as many telephones they have, and people with no telephone do not appear at all. The drawback is that every time we want to get to know telephone numbers we have to go through the entire Telephones table searching for the person's code, but again this procedure is very fast for an appropriate computer program.

1.2.3. Foreign key

When a field, which is not the primary key, is used in a relation with another table this field is called foreign key. This field is important for the database management program, such as Access, when it has to check referential integrity (see section 1.6).

For example, in the previous examples Owner is a foreign key for Telephones table and Secretary is a foreign key for Students table.

1.3. One-to-many relation

A relation is a connection between a field of table A (which becomes a foreign key) and the primary key of table B: on the B side the relation is "1", meaning that for each record of table A there is one and only one corresponding record of table B, while on the A side the relation is "many" (indicated with the mathematical symbol ∞) meaning that for each record of table B there can be none, one or more corresponding records in table A.

For the example of section 1.2.1, the tables are indicated in this way, meaning that for each student there is exactly one secretary and for each secretary there are many students. This relation is called many-to-one relation.

Click here to download full PDF material