

Databases course book

Version 2.3 (17 September 2009)

Free University of Bolzano Bozen – A.Y. 2009-10 – Prof. 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:

- students of the Introduction to Information Science course A.Y. 2008-09 for the Bachelor in Agricultural Science and Agricultural Economics use only level 1;
- students of the Introduction to Programming course for the Bachelor in Logistics and Production Engineering use only level 1;
- students of the Advanced Computer Skills course at the School of Economics and Management use levels 1 and 2;
- students of the Introduction to Information Science course (10 credits) A.Y. 2008-09 for the Bachelor in Logistics and Production Engineering use levels 1, 2 and 3.

This book refers to Microsoft Access 2007, with referrals to Microsoft Access 2003 in footnotes. This book is in continuous development, please take a look at the version date.

Disclaimers

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

Table of Contents

INTRODUCTION	1	4.2. FIELD PROPERTIES (2)	13
TABLE OF CONTENTS	1	4.3. IMPORTING TABLES (2)	14
1. RELATIONAL DATABASE (1)	2	5. FORMS (2)	14
1.1. SINGLE TABLE DATABASE	2	6. QUERIES (1)	15
1.2. PRIMARY KEY.....	5	6.1. SELECTION QUERIES (1)	15
2. DATABASE ARCHITECTURE (1)	6	6.2. EXPRESSION BUILDER (2)	16
2.1. ONE-TO-MANY RELATION (1)	6	6.3. SUMMARY QUERY (2).....	17
2.2. ONE-TO-ONE RELATION (1).....	7	6.4. NON SELECTION QUERIES (3)	17
2.3. MANY-TO-MANY RELATION (1).....	7	7. REPORTS (2)	18
2.4. REFERENTIAL INTEGRITY (2).....	9	8. SQL LANGUAGE (3)	19
2.5. TEMPORAL VERSUS STATIC DATABASES (2)	10	8.1. BASIC SYNTAX	19
3. MICROSOFT ACCESS (1)	10	8.2. INNER JOINS	19
3.1. BASIC OPERATIONS (1).....	10	8.3. SUMMARY QUERIES.....	20
3.2. NORTHWIND EXAMPLE (2).....	11	8.4. OTHER QUERIES.....	20
3.3. RELATIONSHIPS DIAGRAM (1)	11	9. DESIGNING A DATABASE (1)	21
4. TABLES (1)	12	9.1. PAPER DIAGRAM (1)	21
4.1. FIELD TYPES (1).....	12	9.2. BUILDING THE TABLES (1).....	22

9.3. INSERTING DATA (1)	23	10.3. TABLES AND FIELDS (2)	25
9.4. TECHNICAL DOCUMENTATION (1)	23	10.4. FORMS (2)	25
10. MYFARM DOCUMENTATION EXAMPLE (1) .	24	10.5. QUERIES (1)	26
10.1. STRUCTURE (1)	24	10.6. REPORTS (2)	26
10.2. TABLES AND FIELDS (1 ONLY)	24	INDEX	26

1. Relational database (1)

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

1.1. Single table database

The easiest form of database, which can be handled even by Microsoft Excel, is a single table. To be a database table, 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

Moreover, we add this other requisite:

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 1 st semester	Tax 2 nd 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 columns. There are however situations in which a single table is not an efficient way to handle the information.

1.1.1. Information redundancy

For example, when trying to put inside the 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 face the problem of information redundancy.

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 means that the same information is repeated over and over. This 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 sort 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 quickly leads 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.

1.1.2. Empty cells

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:

- even 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	
Person	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 appears several times in the Telephones table. This is normal, 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. Primary key

Each table must 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 programs however handling a composite key can create problems and therefore it is a better idea to use, in this case, an ID.

1.2.1. 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 program when it has to check referential integrity (see section 2.4).

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

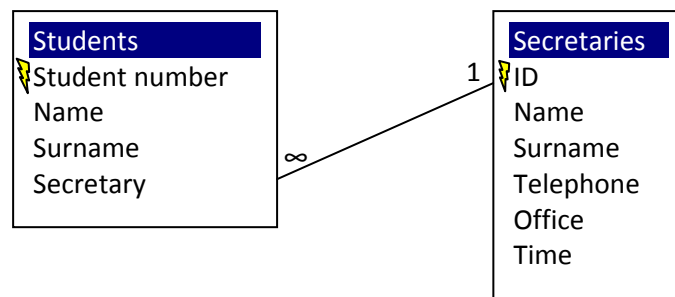
2. Database architecture (1)

A relational database is defined as a collection of structured tables connected via relations. It is important to note that tables must be structured, which means compliant with the requisites of section 1.1, and that they are connected via relations.

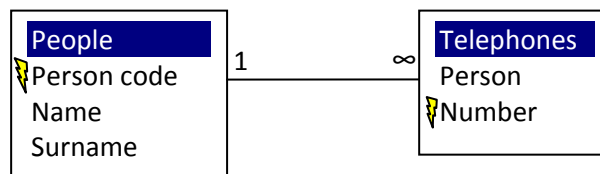
2.1. One-to-many relation (1)

A relation is a connection between a field of table A (which becomes a foreign key) and a 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.

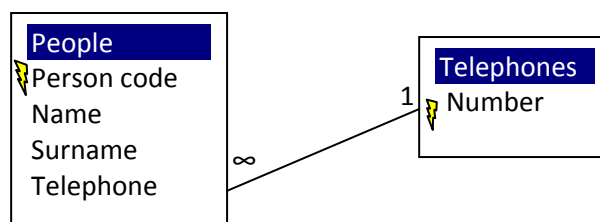
For the example of section 1.1.1, the tables are indicated in this way, meaning that for each student there is exactly one secretary and for each secretary there are many students. This relation is called many-to-one.



For the example of section 1.1.2, the tables are instead indicated in this way, meaning that for each person there can be none, one or several telephone numbers and for each number there is only one corresponding owner. This relation is called one-to-many.

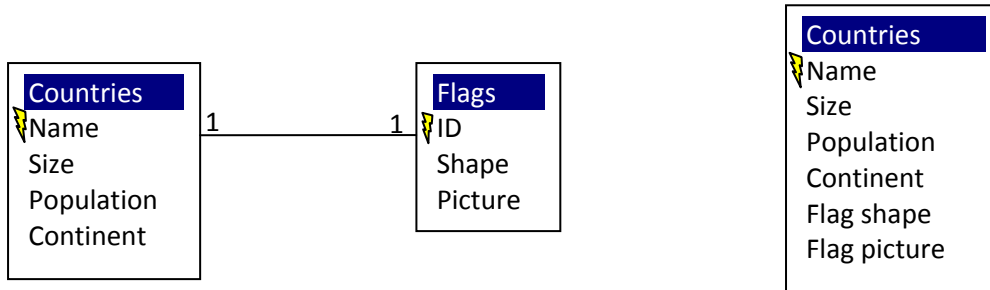


Clearly one-to-many and many-to-one are the same relation, the only difference being the order of drawn tables. It is however very important to correctly identify the “1” side, since it has several implications on the correct working of the database. For example, in the previous example putting the “1” side on the Telephones table means that for each person there is only one telephone and that for each telephone there are many people, a situation which is possible (for example, it is typical of the beginning of the telephone technology, when there was only one telephone for a whole family used by all its components) but which is not what we want to describe with the current database. Moreover, reversing the relation also need to change a little the structure of the tables, putting the foreign key Telephone in the People table instead of the foreign key Person in the Telephones table.



2.2. One-to-one relation (1)

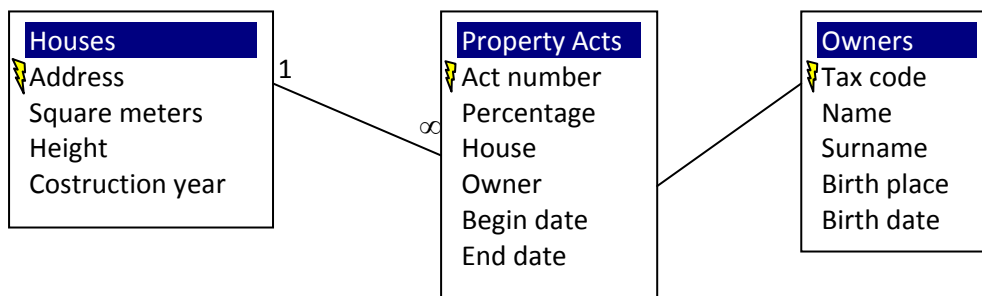
One-to-one relations consist of a direct connection between two primary keys. Each record of the first table has exactly one corresponding record in the second table and vice versa. An example can be countries and national flags. This relation can sometimes be useful to represent two distinct objects with a lot of fields, but it should be avoided, since the two tables can be easily joined together in a single table.



2.3. Many-to-many relation (1)

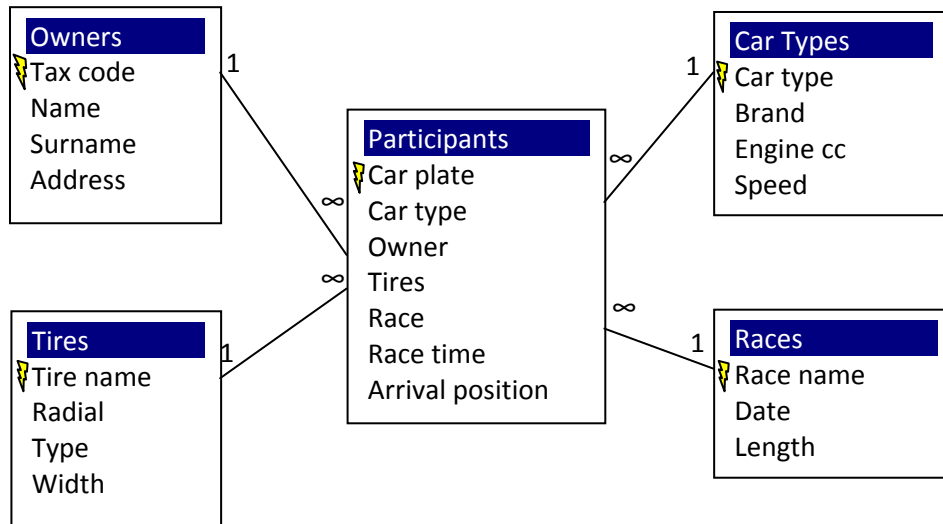
Even through many-to-many relations are very common in real applications, unfortunately they can not be handled automatically by relational databases. In order to deal with them, relational databases use a junction table, which is an extra table with the task of connecting together the two fields which are many-to-many related; sometimes this junction table has an effective corresponding meaning in everyday experience, other times it is only an abstract representation of the relation. In any case, it is always a good idea to give a meaningful name to the junction table, often using a question form such as “What is owned by whom”, to have always clearly in mind its meaning.

For example, we build a database with houses and owners. Each house may be owned by several people (with different percentages or, if we are building an historical database, with different starting and ending dates), and on the other hand each person may own several portions of houses. In order to represent this many-to-many relation between houses and owners we use a junction table which can be called either “What is owned by whom” or “Who owns what” or, using a more tangible name, Property Acts.



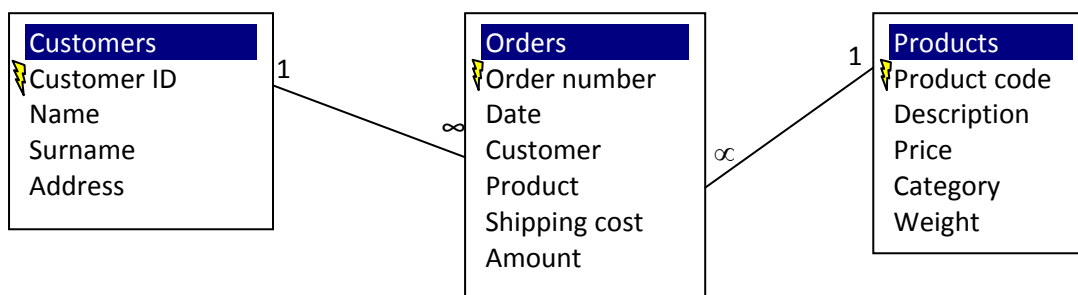
Each owner can therefore have many property acts and each house can have many property acts which refer to that house. On the other hand each property act has written on it only one owner and one house.

This is the typical structure of the junction table: it contains two or more foreign keys on the “many” side of the relation. An example where the junction table contains four foreign keys is this database of non-professional car competitions with Car Types, Tires, Races, Owners.



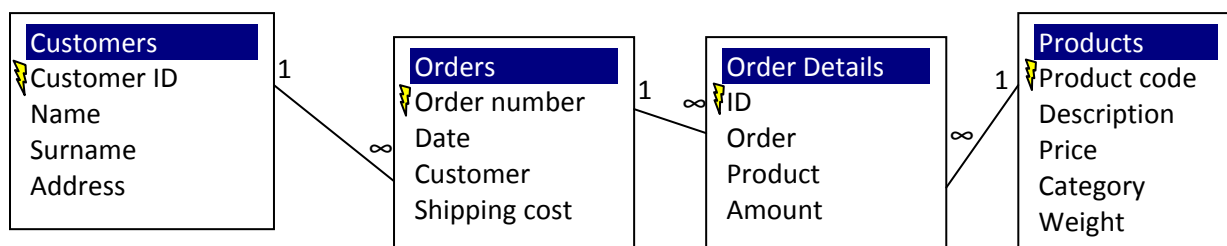
2.3.1. Details table (2)

Many times in everyday applications the relation is so complicated that a junction table is not enough. This is the case, for example, of a selling database, with table Customers and table Products. Clearly each customer may order different products and each products is hopefully ordered by several customers, therefore we need an Orders junction table. This table contains also all the details of the order, such as the amount of products, the date and the shipping cost.



However, while it is correct that for each order there is one and only one customer, for each order there is also one and only one product, which is not what usually happens in real applications where a customer orders several products at the same time and wants also to pay them all together with combined shipping costs.

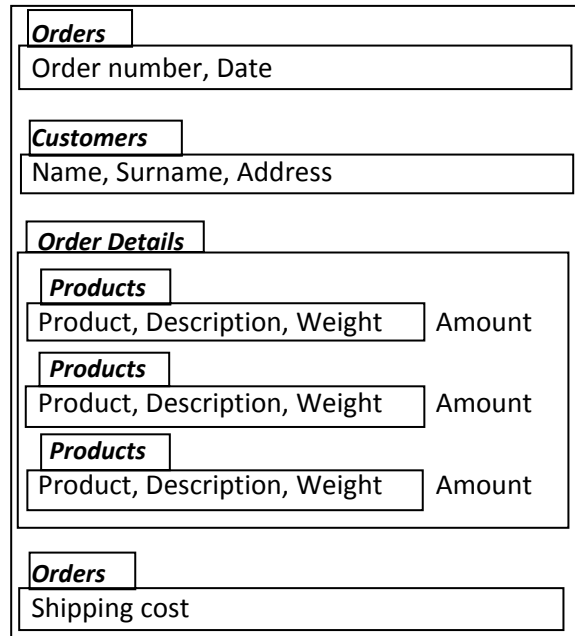
In order to deal with this situation, we need a details table. We leave all the order's administrative information, including the customer relation, in the Orders table and we move the list of ordered products into the details table, which will look like the Telephones table of section 1.1.2.



Each record in the Order Details table represents a product which is ordered with its amount and clearly an order can have several details. In this way an entire order can be represented taking from the Customers

table the information on who ordered it, from the Products through the Order Details table the information on the products and from the Orders table itself the administrative information.

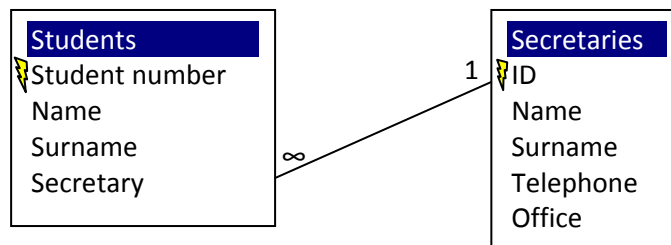
Using queries and reports (explained respectively in section 6 and section XXX) all these data can be conveniently put together, taking them from the tables and automatically joining them following the relations, into a report like this one.



A details table is in general used every time the junction table, even with several foreign keys, is not enough to describe the relation. In some cases further sub-detail tables may be even necessary.

2.4. Referential integrity (2)

If two tables are related via a many-to-one relation, like the one between students and secretaries of section 1.1.1, we are no more free to modify the data on the “1” side at our will. For example, if we delete a secretary or if we change its ID, there are probably corresponding students in the Students table which becomes orphans, i.e. they do not have their corresponding secretary anymore and following their relation to the Secretaries table leads to a nonexistent ID. This issue is known as referential integrity, which is the property of a database to have all the foreign key’s data correctly related to primary key’s data. When a record on the “1” side table is deleted, referential integrity can be broken and this results in a non consistent database.



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

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

→ orphan

→ orphan

On the other hand, if a table has only “many” side relations, its records can be freely deleted and modified without breaking the referential integrity.

2.5. Temporal versus static databases (2)

It is a common mistake when deciding the architecture of the database of limiting it to an instantaneous view of reality, instead of building, often with the same architectural effort, a database which can handle historical information. A temporal database does not only offer the opportunity to handle past data, but also gives the chance to easily revert to the previous situation in case of input errors, which for a static database is often impossible since data have been overwritten.

For example, the Property Acts table in section 2.3 could be a static table, reflecting the current status quo of the property, or an historical table with the beginning and ending date of property. Simply introducing two date's fields has converted our database from static to temporal opening a wide range of new possibilities.

3. Microsoft Access (1)

Microsoft Access 2007 is a database management program, a program which is in charge of handling data and extracting them following correctly the relations and doing other sorting and filtering operations. Moreover, Access takes care of preserving the correct structure of the database enforcing referential integrity (see section 2.4).

Other famous database management programs are Oracle, MySQL, PostgreSQL.

3.1. Basic operations (1)

Microsoft Access behaves in a slight different way with respect to Microsoft Word or Excel and Office users can easily become confused.

The most important thing to keep in mind is that Access automatically saves every data operation as soon as it is done, without needing to give the save command. Exceptionally only the last data modification can be undone, but not the others. This behavior is typical of databases, where several people must access the data at the same time and therefore data must always be up-to-date. There is a Office button → Save¹ command in Access, but it is typically used to save the objects (tables, queries, reports, forms) inside the database file, while there is no need to use it to save the whole database file on the hard disk. If we want to save the database file with another name or in another format, the right tool to do it is the command Office button → Save As → Save the database in another format², which allows to duplicate the database file with another name or in another format, or Office button → Manage → Back Up Database.

¹ For Microsoft Access 2003 File → Save.

² Pay attention that in Microsoft Access 2003 there is only the option File → Backup Database.

When the database is opened, the main window usually appears on the left³. Choose from the drop down menu Object Types and then select All Access Objects. Each object category presents the list of all that objects together with two buttons to manually create another object or to create it with the help of a wizard tool.

3.2. Northwind example (2)

Microsoft Access provides an official database example called Northwind. It is better to use 2003 version, which is much simpler and usually located in directory C:\Program Files\Microsoft Office\Office11\Samples.⁴

When this database (or another one containing Visual Basic macros) is opened, Access usually displays a Security Warning⁵, which can be ignored clicking on Options → Enable this Content.

This database example also presents at its activation a splash screen, i.e. a graphical interface which leads the user to the most common operations⁶. Building graphical interfaces is beyond the scope of this book and they can easily be ignored to arrive directly at the database main window.

3.3. Relationships diagram (1)

Access provides a tool to automatically display the database's architecture through command Database Tools → Show/Hide → Relationships⁷. This opens a graphical interface displaying tables, fields and relations.

This tool has however small problems:

- if some tables are missing, right-click → Show all;
- if nonexistent tables are displayed, often with an _1 extension, select them and press Del;
- when closing the relationships diagram Access asks to save it. This only saves the layout of the diagram, the modified relations are instead saved immediately.

In the relationships diagram relations can be created, deleted and modified:

- to delete a relation, select it → right-click → Delete. Warning: this operation cannot be undone;
- to modify a relation, double-click on it. A graphical interface appears, which displays the two tables and the related fields together with the relation type, which is automatically decided by Access according to the structure of the tables and to the fields involved. Moreover, there is also a checkbox to enforce referential integrity: it is very important that this box is checked because in this way Access will forbid the deletion of records in the "1" side table when this operation breaks the referential integrity of this relation;

³ In Microsoft Access 2003 it is a floating window with the object types' menu on the left and All Access Objects does not need to be selected.

⁴ Also Microsoft Access 2007 has a Northwind 2007 database, but it must be downloaded or generated from a template file. Moreover, it has a much more complicated structure which can mislead the novice user. Office 2007 users should copy the Northwind.mdb file from an Office 2003 system, or, if a computer with Office 2003 is not available, search for this file on the Internet with the help of a search engine.

⁵ Microsoft Access 2003 displays instead three pop-up windows to which "No" then "Yes" then "Open" should be answered; they can be permanently disabled choosing Tools → Macro → Security → Low.

⁶ Microsoft Access 2003 displays two splash windows.

⁷ For Microsoft Access 2003 File → Relationships.

- to create a relation between two fields, simply drag a field above the other. If at least one of the two fields is a primary key, Access automatically recognizes the relation's type and the only remaining thing to do is to apply referential integrity.

It is always better to do any structure's modification before filling the tables with data. Once data are inside, Access may refuse to do certain operations on relations when the present data are inconsistent with the new relations, or may delete data inside foreign key fields when they do not complain with the new relation.

Relations can also be built with Lookup Wizard (see section 4.1.1). Using this tool Access creates at the same time the relation and, in the foreign key field, a user-friendly drop-down menu which speeds up data insertion.

4. Tables (1)

Tables are accessed choosing Tables object in the main database window.

The best way to create a new table is Create → Tables → Table Design⁸. This icon opens an empty table in Design View, where fields can be added with the indication of the primary key, the field name, the field type and the field detailed description. Right-clicking on the left column gives the possibility to define or undefine a primary key, while the field type can be chosen from a drop down menu in third column.

Each existing table in the main database window can be opened double-clicking on it. The table is displayed in Datasheet View, which is an Excel-like way to look at the table and which is also a convenient way to edit or insert data. In order to see the table in Design View, we choose Home → View → Design View⁹.

4.1. Field types (1)

It is very important to choose the correct field type because it helps the database to minimize the space allocation and to avoid wrong insertions. In Access the most important field types are:

- Text, which contains up to 255 alphanumeric characters. This type is proper for names, addresses and every short text;
- Memo, which contains up to 65,536 alphanumeric characters. This type is proper for long texts, such as curricula, abstracts and small articles;
- Number, which contains numbers which can be manipulated through mathematical operations. This type must be used only for numerical information. It is a very common mistake to use it for numeric codes, such as telephone numbers, version numbers and ZIP codes. Numeric codes must use the text type, since they may start with 0 (telephone 0471012343 or ZIP 00100) or have a 0 as last decimal digit (version 7.10) and, in any case, mathematical operations with them must be forbidden;
- Date/Time, which contains dates and times. Exactly like Excel, Access memorizes date and time together, using integer numbers for days. Therefore dates can be subtracted to obtain the difference in days, or numbers can be added or subtracted to them to go ahead in the future or back in the past;
- Currency, which contains numbers with automatically a currency symbol;
- Autonumber, a type which is used only by Access to create IDs;
- Yes/No, a type with only two values;
- OLE object, which contains other files, such as images or documents. These files may be embedded, which means that the database file automatically contains a duplicate of this file (thus making the database file larger) or linked, which mean that the database file simply contains the link to the file (thus making it unusable when the external file is not available);

⁸ For Microsoft Access 2003 we instead choose "Create table in Design View" from the main database window.

⁹ For Microsoft Access 2003 View → Design View.

- Hyperlink, which contains an hyperlink usually to a web page or to an email address;
- Lookup Wizard is not a real field type but the possibility to take field's values from a predetermined list or from other tables.

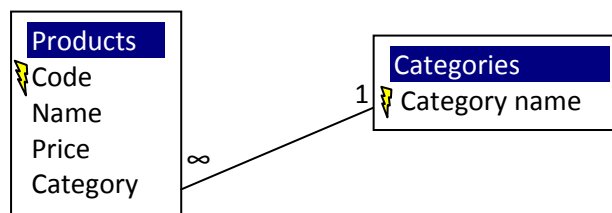
4.1.1. Lookup Wizard (2)

From the Lookup Wizard, choosing "I will take the values from another table", Access automatically builds a relation with another table taking the current field as foreign key of the "many" side and the primary key of the selected table as "1" side of the relation. At the same time, during the building of this relation, Access asks which fields the user wants to display in the drop down menu. Since these fields are simply what will be displayed, not the real field involved in the relation (which is instead the primary key), it is convenient to choose the most meaningful fields for data editing (for example, name, surname and birth date of a person). At the end of the wizard procedure the relation is created but without referential integrity, which should be added manually from the relationships diagram (see section 3.3).

From the Lookup Wizard, choosing instead "I will type in the values that I want", Access lets the user type in a predetermined list of values which will be offered every time the field is filled. The list is not mandatory and a different value can be manually typed in the field.

4.1.2. Mandatory predetermined list (2)

If the list of predetermined values should be instead mandatory, it is better to build another table which simply contains the list of values as primary key and build, via Lookup Wizard, a relation which takes values for this field from the primary key of the new table, such as in this example.



With this solution the user may not choose values which are not in the list; however, he can add other values to the list simply adding more records to the second table, without having to modify the fields' features in the first table.

4.2. Field properties (2)

According to the chosen field type, several field properties appear on the bottom window. The most interesting are:

- Field Size, which restricts the number of characters which can be inserted for both text and numeric fields;
- Format, which defines how does the data look like;
- Decimal Places, which fixes the decimal digits for numbers;
- Default Value, a value which is automatically assigned to the field whenever the user does not type anything;
- Validation Rule, a rule to which values of this field must adhere to be accepted. This rule can be quite complex, but may not use values taken from other fields. For example, to indicate that field Age must be 18 or above, the validation rule is as following ≥ 18 ;
- Validation Text, the warning displayed to the user when the validation rule is broken;
- Required, which indicates that the field must be filled. When required is set to yes, records with no value in this field are not accepted;
- Allow Zero Length, which indicates whether an empty sequence of characters (which is not considered an empty value) can be inserted in a text field or not;

- **Indexed**, which indicates to Access that this field is going to be used for searches. Access organizes this field in a special way to speed up future searches. It can be Duplicated Allowed or No Duplicates, depending on whether we want to allow two records to have the same value for this field. This is also a good trick to force a field, which is not a primary key, to have all different values.

4.2.1. Table validation rule (2)

While a field validation rule cannot involve other fields, sometimes it is necessary to put a validation rule on entered data that crosschecks the values of different fields of the same table. To do this, in Design View choose Show/Hide → Properties¹⁰ and add a table validation rule (using also the Expression Builder, see section 6.2) and its corresponding validation text. For example, for an hotel booking table it is necessary to have departure dates not before arrival dates and therefore the condition here is

[Departure Date] <= [Arrival Date].

If more rules are needed, they must be combined with the And operator, for example

([Departure Date] <= [Arrival Date]) And ([Booking Date] <= [Arrival Date]).

Unfortunately the validation text is only one and it is not possible to tell the user exactly to which part of the rule his data do not adhere.

Complex expressions can be built with the Expression Builder (see section 6.2).

4.3. Importing tables (2)

Access can obviously import data from several sources, typically tab-delimited text files, comma-separated text files and Excel files. The importing operation for text files is very similar to importing a text file into Excel. The importing operation of an Excel file is very easy with the command External Data → Import → Excel¹¹.

The only thing to pay attention to is that data must already be well structured before importing them into a table, otherwise the importing procedure will stop several times.

5. Forms (2)

A **form** is a graphical interface which lets the user look, modify, insert, delete data. When the user is not the database administrator, it is better that tables are not accessed directly, since a wrong value, especially in a foreign key, can lead to a non consistent database. Forms can present the data in a more user-friendly way and can restrict access to some data or forbid some data operations.

To produce a form in Access we build it using wizard choosing Create → Forms → Other Forms → Create using Wizard¹². This guides us through a step by step procedure, where we:

- choose the tables and the fields from which data are taken. If data come from more than one table, Access automatically takes into consideration the existing relations and builds an appropriate subform for data on the “many” side. Data can also be taken from queries (see section 6), but usually they are not;
- choose the layout of the form;
- choose the style of the form;
- give the form a name. The form, being an object, needs to be saved inside the database file.

¹⁰ For Microsoft Office 2003 View → Properties.

¹¹ For Microsoft Access 2003 File → External data → Import.

¹² For Microsoft Access 2003, we instead choose “Create Form using wizard” from the database main window.

The form can then be opened in Form View to access the data, paying attention to the fact that, as usual, every datum modification is automatically reflected in the corresponding table.

The form can also be opened in Design View to change its layout and style, or to add and remove fields. Choosing command Tools → Properties¹³ opens all the form's parts' properties from which the layout can be precisely defined. Among these properties the most important ones are in the Form drop down menu Data tab, where modifying the fields "Allow Modifications", "Allow Deletions" and "Allow Insertions" restricts the user from modifying, deleting, inserting records through this form.

6. Queries (1)

A query is a question posed to the database, which answers with a virtual table called view. The question can be, for example, "Which students are born in 2007?" or "Who are the German students, which is their address, and what are their grades' averages?". The view would be in these cases a table with a single column with the students' numbers, or a table with four columns with student's name, student's surname, student's address and the average of the grades of that student.

The view therefore contains all the values corresponding to the fields selected in the query, organized following correctly the underlying relations, sorted and filtered according to the query's indication and together with virtual fields created using formulas contained in the query. It is thus a powerful tool to extract information from the database.

The view, even though it is virtual, is directly linked to the real data and any modification to its data is automatically reflected in the original tables. The view does not contain any formatting: to present query's results in a better-looking format, report is the appropriate tool (see section 7).

6.1. Selection queries (1)

The select query is the standard type of query, a direct question to the database which involves only data extraction following correctly the relations and sometimes doing calculations. For example, a question such "Which exams has Jack passed?" or "How many students has each secretary in charge?".

To produce a select query in Access we simply use the command Create → Other → Query Wizard¹⁴. This guides us through a step by step procedure, where we:

- choose the tables and the fields from which data are taken. If data come from more than one table, Access automatically takes into consideration the existing relations. Data can also be taken from other queries;
- give the query a name. The query, being an object, needs to be saved inside the database file.

The query can then be opened in Datasheet View to access the data, paying attention to the fact that, as usual, every datum modification is automatically reflected in the corresponding tables.

The query can also be opened in Design View, where we have full control over what is displayed in the view. In the upper side of this window we see the involved tables with their relations and we can add other tables with right-click → Add tables. In the lower side we have the selected fields, which can be removed with right-click → Del or to which other fields can be added simply dragging them from the tables above. To a field we can also put a sorting option, ascending or descending, or a Show option to display/hide the field (obviously hiding makes sense only when the field is used for something else, otherwise it would be better to remove it directly from the query).

¹³ For Microsoft Access 2003 View → Properties.

¹⁴ In Microsoft Access 2003, we select the Queries objects in the main database window and choose "Build Query using Wizard".

We can also use a criterion to filter out some records. For example, we can type in the criteria space directly a value (enclosed in quotations if that field is a textual field) and only the data having this value in this field are displayed. We can also put more complicated conditions, such as equalities and inequalities or conditions involving other fields. For example, to filter out underage students, we can put in the Age field criterion

`>=18;`

to consider only students enrolled from 2006, we can put in the EnrollDate field criterion

`>= #1/1/2006#.`

However, when the condition becomes too complicated, it is better to use the Expression Builder (see section 6.2). If two criteria are put on the same field, typing them on two spaces one above the other, they automatically are alternative (it is enough that one of them be valid for those data to be displayed); on the other hand if two criteria are put on different fields, typing them on two spaces of the same row, they are considered together (both must be valid for those data to be displayed). Again, complicated conditions with logical operators are more easily built with the Expression Builder.

6.1.1. Virtual fields (2)

Other fields can be automatically generated taking values from any field, even fields not present in the query (but their tables must be present in the query's window upper part), and applying mathematical, logical or textual operations. We simply type in the field's name space of an existing query the virtual field's name followed by a colon and by its expression, using the Expression Builder (see section 6.2) or typing fields' names involved in the operation enclosed in square brackets. For example, to build virtual field ToPay which contains the price of an order with a single product, we simply type in the field's name space the following

`ToPay: [Price] * [Quantity].`

If we instead type in the name's space or in the criteria's space something between square parenthesis which does not correspond to any field in the present tables, Access stops the execution of the query and asks us the value of that thing. This is a good trick to force Access to ask the user a value. For example, putting in the criteria space of the EnrolmentYear field

`[Please, tell me the enrolment year]`

and switching to Datasheet View, forces Access to stop the execution of the query, realize that a field with name "Please, tell me the enrolment year" does not exist, and display a dialog box which says exactly "Please, tell me the enrolment year".

6.2. Expression Builder (2)

The Expression Builder is a powerful tool to build expressions, which can be called clicking on the three dots anytime there is the need for an expression, e.g. validation rules, table validation rules, query's criteria and query's new fields.

The Expression Builder has a main space where the expression appears. It can be directly typed in or it can be built using the mathematical and logical operators presented below. If it is possible (i.e. we are not inside a field's validation rule), other fields' values can be inserted in the expression choosing them from the menus below: they appear as their field's name enclosed in square parenthesis, sometimes preceded by the table's name if the same field's name is used in more than one table. Also many functions can be inserted, the most interesting being:

- Date to get the current date;
- DateDiff to get the difference in months or years between two dates;
- DateAdd to get a date plus or minus a certain amount of months or years;
- Year to get the year from a date;
- the usual mathematical functions Abs, Exp, Sqr, Log, Int;

- Like, which lets the user check whether the value corresponds to an expression using wildcards ? (any character) and * (any amount of characters). For example, to check that a ZIP code has exactly 5 characters and starts with 39, we can use the condition

Like "39????".

To check that an email address is reasonable, we can use the condition

Like "*@*.***".

6.3. Summary query (2)

A simple query is able to perform operations, but only acting within the same record of the view and is not able to perform operations involving different records, such as sums or averages. For example, "How many students has each secretary in charge in 2009?" or "What is the average students' grade this year for each country?".

To do this, we need a summary query: we create a simple query with the involved fields, then we press Show/Hide → Σ Totals button¹⁵ and a new option appears in the query's field. This option is originally set to Group By, meaning that the view will now be squeezed trying to group identical values of those fields. If some view's records have the same values in all the fields which have the Group By option, only a single record appears in the view. This feature used alone can be applied to some rare situations (and, on the other hand, causes some problems when the summary query is wrongly chosen instead of a simple query), but usually it works in conjunction with the selection of another option for one field, such as Sum, Count or Avg. In this case, the query tries to group using the fields with the Group By and, for every group, it calculates the sum, average or count of the values of the field with the other option.

For example, if we have a database with Students table with Country field and we have an Exams table with Grade field and we want to calculate the average grade for each country, we need to create a query with these two fields, convert it to summary query, and then select Avg option to the Grade field while leaving Group By option to the Country field. The result is a view with field Country with the list of countries, each one appearing only once since the results are grouped by countries, and field Grade (sometimes called instead Avg of Grade) with the average of the values of the Grade fields for students in that country.

It is very important to remember that every field present in the query with the Group By option is used to group, even when this field has the no-show option or when this field is simply used for a criterion. A wrong grouping, with extra fields, clearly produces more records in the view than what should be. Criteria in fields with Group By option can thus lead to problems and therefore it is always suggested to switch the Group By option to Where whenever a field is used only for filtering and not for grouping.

For example, if in the database above we want to calculate the average grade per country in 2008, we need to build a simple query with the fields Country, Grade and Date. Then we convert it to summary query, we select Avg option in the Grade field and we select Where in the Date field putting as criterion

Between #1/1/2008# and #31/12/2008#

and we unselect the Show option. If we instead leave Group By option in the Date field with no-show option and with this criterion, the result is grouped by country and by date which means that the view shows the average per country per exam session.

6.4. Non selection queries (3)

While selection queries extract data from the database, there are other queries which actively modify data in the database. The best way to handle these queries is to build them as selection queries in Design View and Datasheet View, then from the Query Type tab¹⁶ switch them to the appropriate query type, check

¹⁵ In Microsoft Access 2003 it is simply the Σ button.

¹⁶ In Microsoft Access 2003 from the Query menu.

carefully in Design View that the query is doing exactly what is wanted and then finally press Results → Run¹⁷ to do changes to the data.

Make Table query converts the view into a real independent table. From now on those data are independent from the data contained in the original tables.

Update query selects records and can assign a value to the present fields through a new space that appears.

Delete query selects records and deletes all the records involved (and not only the content of the present fields) from their corresponding tables.

Append query appends the records of the view to an existing table. Obviously the view's fields must match exactly the fields of the table.

7. Reports (2)

A report is a way to present database's data in a good-looking format.

To produce a report in Access we simply use the command Create → Reports → Report Wizard¹⁸. This guides us through a step by step procedure, where we:

- choose the tables or the queries and their fields from which data are taken;
- choose whether data must be grouped or not. For example, students can be grouped by countries if the Country field is selected. If data from several tables are involved, Access always suggests a grouping based on the relation's "1" side. Once the first grouping is selected, Access asks whether further grouping is required. For example, students can be grouped by corresponding secretary and then, inside each group, grouped again by country;
- choose whether data must be sorted;
- choose the layout of the report;
- choose the style of the report;
- give the report a name. The report, being an object, needs to be saved inside the database file.

The report is automatically opened in Print Preview¹⁹. Closing Print Preview, it can then be opened in Report View to look at the final result, or in Design View to change its layout and style, or to add and remove fields, titles, header and footer. Reports can be exported from Access in RTF format from Data → Word.

¹⁷ For Microsoft Access 2003 Query → Run.

¹⁸ In Microsoft Access 2003, we select the Reports objects in the main database window and choose "Build Report using Wizard".

¹⁹ In Microsoft Access 2003 it is opened automatically in Report View.

8. SQL language (3)

Standard Query Language is a programming language for querying and modifying data and managing databases. Although every database operation can be done using SQL commands, in this book only SQL commands for queries are presented.

From inside an existing query, switching to SQL View displays the SQL code corresponding to the current query. Modifying the SQL code in this window immediately reflects to the Design View and to the Datasheet View.

8.1. Basic syntax

The simplest SQL syntax is

```
SELECT {fields} FROM {table},
```

where {fields} is a list of fields separated by commas and {table} is the name of a table. For example,

```
SELECT FirstName, LastName, BirthDate FROM Students.
```

To add a filtering or sorting condition we just need to add

```
WHERE {condition} ORDER BY {field} {ASC|DESC}.
```

For example,

```
SELECT FirstName, LastName, Age FROM Students WHERE BirthDate >= #1/1/1988# ORDER BY LastName ASC ORDER BY FirstName ASC.
```

If we want to create another field, the syntax is

```
SELECT {expression} AS {name} FROM {table}.
```

For example,

```
SELECT FirstTax + SecondTax AS FinalTax FROM Students.
```

8.2. Inner joins

If we want to take values from two tables, SQL language is not able to correctly follow the relations. This is due to the fact that SQL is a generic programming language which works for every database management program, even those which do not handle referential integrity nor automatic relations. Therefore, we need every time to explicitly remind SQL which relations are involved. The easiest way to do it is

```
SELECT {fields} FROM {tables} WHERE {field1} = {field2},
```

where {field1} and {field2} are the foreign and primary key of related tables, while {fields} are the names of the selected fields from {tables}, where each name may be preceded by table's name and a dot in case the same field's name is used in both tables. For example,

```
SELECT Secretaries.FirstName, Secretaries.LastName, Students.LastName FROM Students, Secretaries WHERE Secretaries.Code = Students.Secretary.
```

If three tables are involved, there must be a where condition for every relation, even when a table (typically a junction table) does not use any field. Using the example in section 2.3,

```
SELECT Houses.SquareMeters, Owners.LastName FROM Houses, PropertyActs, Owners WHERE Address = PropertyActs.House AND PropertyActs.Owner = TaxCode.
```

Another way to do the same thing is

```
SELECT {fields} FROM {table1} INNER JOIN {table2} ON {field1} = {field2},
```

where {field1} and {field2} are the foreign and primary key involved in the relation. For example,

```
SELECT Secretaries.FirstName, Secretaries.LastName, Students.LastName FROM Secretaries INNER JOIN Students ON Secretaries.Code = Students.Secretary.
```

If three tables are involved, there must be a compound inner join, even when a table (typically a junction table) does not use any field. Using the example in section 2.3,

```
SELECT Houses.SquareMeters, Owners.LastName FROM ( Houses INNER JOIN PropertyActs ON Address = PropertyActs.House ) INNER JOIN Owners ON PropertyActs.Owner = TaxCode.
```

Even though the inner join seems more difficult to use, it exactly describes what the query is doing: taking all three tables and squeezing their content relation by relation until a single table is reached. In fact, when switching from a query built with the wizard to SQL View, we can see that Access by default uses the inner join technique.

8.3. Summary queries

Summary queries are easier and more comprehensible when presented in SQL View. Their syntax is exactly the same of the simple query, with the difference that a function (Count, Sum, Avg) is before the field on which a record operation must be done and that a

GROUP BY {fields}

is added at the end of the SQL code. For example, if we have a database with Students table with Country field and we have an Exams table with Grade field and we want to calculate the average grade for each country, the SQL is

```
SELECT Country, AVG(Grade) FROM Students INNER JOIN Exams ON Students.StudentNumber = Exams.StudentNumber GROUP BY Country.
```

For example, if we want to calculate the number of passed exams per country in 2008, the SQL is

```
SELECT Country, COUNT(Exams.ID) FROM Students INNER JOIN Exams ON Students.StudentNumber = Exams.StudentNumber WHERE Year(Exams.Date) = 2008 GROUP BY Country.
```

8.4. Other queries

Create query's SQL syntax is

```
SELECT {fields} INTO {new table} FROM {table},
```

to which obviously inner joins, where conditions, grouping and all the previous elements can be added.

Update query's SQL syntax is

```
UPDATE {tables} SET {field} = {value},
```

to which inner joins, where conditions and all the previous elements are usually be added. For example, to set grade to 27 to all the exams done by German students, SQL is

```
UPDATE ( Exams INNER JOIN Students ON Exams.StudentNumber = Students.StudentNumber ) SET Grade = 27 WHERE Students.Country = "Germany".
```

Delete query's SQL syntax is

```
DELETE FROM {table},
```

to which inner joins, where conditions and all the previous elements are usually be added. Note that no field needs to be specified. For example, if we want to delete all the German students we simply type

```
DELETE FROM Students WHERE Country = "Germany".
```

Append query's SQL syntax is

```
INSERT INTO {table} ({fields}) SELECT
```

followed by a standard selection query. Fields in the {fields} list must have the same type as fields in the selection query.

9. Designing a database (1)

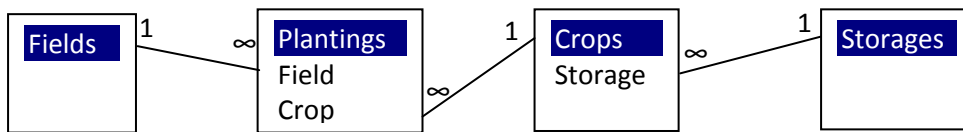
The most important rule to keep in mind when designing a database is that the architecture must be fully completed before data are entered. Entering data is usually a time-consuming procedure, either because they are manually typed in or because they are imported from other tables which must be well structured before the import operation. Modifying the architecture, or just a single relation, after data have been entered, often means huge data cancellation or restructure while Access continuously emits warnings that data are not compliant with the new structure.

9.1. Paper diagram (1)

The first step is a paper diagram of the architecture, containing all the tables with relations and foreign keys. It is very important to check that

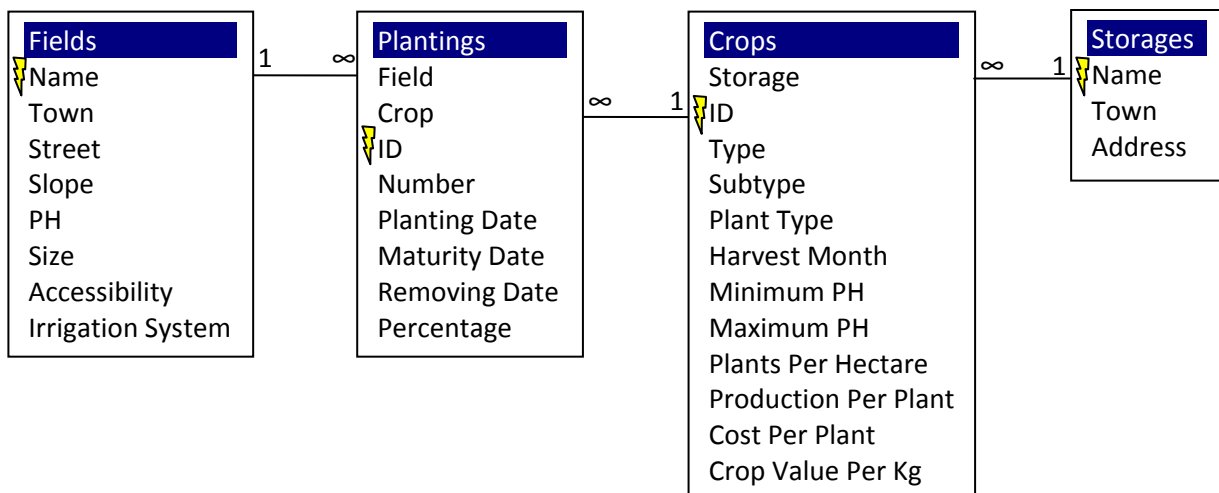
- one-to-many and many-to-one relations are properly oriented,
- no relation is instead a one-to-one,
- many-to-many relations are dealt with appropriate junction tables and eventually details table,
- the "1" side of relations is really a "1" side (either because there are obvious reasons that guarantee so, for example in a relation between Exams and Students, or because we want it to be like this, for example in the relation between Secretaries and Students) and does not instead need a many-to-many relation.

For example, if we want to build database MyFarm for a small fruit farm company, we start with this architecture



Considering carefully the fact that putting a many-to-one relation between Crops and Storages automatically implies that each crop is stored in only one storage while a storage can accept different crops. Plantings table is a junction table between Fields and Crops: in fact in the same field we can plant several crops (either dividing the field in parts or considering an historical database where each field can be used for different crops in different years) and, clearly, the same crop can be planted in several fields.

Now we can enrich the diagram putting all the other fields and assigning primary keys, checking that no duplicate values of the primary key exist.



We assign primary key to Name field for Fields and Storages tables, keeping in mind that neither two Fields nor two Storages with the same name may exist. For Plantings we decide to use an ID, while for Crops a

composite primary key could be Type and Subtype, but in order to avoid problems with composite primary key we choose to use an ID.

Especially when the paper diagram has to be submitted to somebody else, it is a good idea to write now a short description of the database and of the data that are going to be put inside, in order to sum up everything and to check that the relations are correct.

9.2. Building the tables (1)

Now we can start to build tables. It is better to start from the external ones, i.e. the tables which do not have foreign keys, since these table do not take values from other tables, and then going on to the other tables. For each field the appropriate type must be chosen, and primary keys must be defined, while other fields' options can be decided later.

For example, for table Fields: Name, Town, and Street are text; Slope, PH, and Size are number; Accessibility and Irrigation system are yes/no. For table Storages: Name, Town, and Address are text. For table Crops: ID is autonumber; Type, Subtype, Plant Type, and Harvest Month are number; Minimum PH, Maximum PH, Plants Per Hectare, Production Per Plant, Cost Per Plant, and Crop Value per Kg are number. For table Plantings: ID is autonumber, Number and Percentage are number, Planting Date, Maturity Date, and Removing Date are date/time

Foreign keys can be set:

- (1) to the same type of the corresponding primary key, and then the relation can be built in the relationships diagram, remembering to enforce referential integrity;
- (2) (3) as Lookup Wizard choosing as displayed values the appropriate meaningful fields from the related table, hiding the primary key when it is an ID (and thus not meaningful for a human operator) or unhiding it when it is meaningful. Then referential integrity is enforced in the relationships diagram.

For example, we set Storage field in Crops table to

- (1) text (like Name in Storages table) and we create a relation in the relationships diagram enforcing referential integrity;
- (2) (3) Lookup Wizard, choosing to take values from Storages table and to display, for example, Name and Town, remembering to unhide Name (usually Access tries to hide the primary key) since it is meaningful to have it in the drop-down menu. Then we enforce referential integrity.

9.2.1. Field options (2)

Once fields are defined we can set also all the options. For example, in the Plantings table:

Field and Crops have the required option;

Number is set to long integer with 0 decimal digits, with validation rule ≥ 0 and a validation text "Number of planted plants must be zero or positive". We also write the description "Number of planted plants";

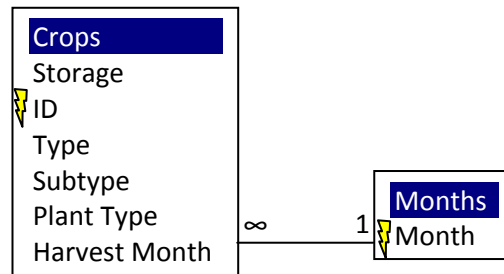
Planting Date is set to required and we set also the option index (duplicates allowed) since we are going to do searches based on the Planting date;

Maturity Date and Removing Date are not set to required since we want to leave the possibility to have these fields empty (for example, if we do not know when we will remove this planting);

Percentage is set to single with 2 decimal digits and percentage format, with validation rule Between 0 And 1 and validation text "Percentage of used field must be between 0% and 100%". We can also set default value to 0 and option required, since it is important to always know how much field each planting uses.

Some fields clearly admit predetermined values. A drop-down menu can be easily created using the Lookup Wizard with values directly typed in. For example, to Plant Type we add a drop-down menu with a non mandatory predetermined list containing the most common plant types (tree, bush, vine, herb); we choose a non mandatory list since other values can exist. Same thing for the Harvest Month, we insert the twelve months in a predetermined list through Lookup Wizard.

(3) However, if these values are mandatory and we are sure that no other value is allowed, we should instead build another table with the predetermined list of values and convert the considered field into a foreign key. For example, for Harvest Month the list is mandatory since only twelve months exists. Thus we build a Months table and we connect the Harvest Month to it building the relation and the drop-down menu using the Lookup Wizard displaying simply the Month, always remembering to enforce referential integrity.



9.2.2. Table validation rule (2)

When applying table validation rules we must always keep in mind that a rule can usually involve only fields which are required. If we involve fields which are not required, we must always add to the validation rule the possibility that the non required fields be empty otherwise the validation rule will automatically fail.

For example, a validation rule for table Plantings which condition that Maturity Date and Removing Date be not before Planting Date, keeping in mind that Maturity Date and Planting Date are not required, is

```
( [Maturity date] Is Null Or [Planting date] <= [Maturity date] ) And ( [Removing date] Is Null Or [Planting date] <= [Removing date] )
```

Each part of the rule is true when the field is either empty or after Planting Date.

9.3. Inserting data (1)

Now data can be inserted, either manually or importing them from tables in other formats. The only thing to remember is to have enough and appropriate data such that all the planned forms, queries and reports make sense and display something meaningful. This can also be a double-checking to control that queries work correctly.

9.4. Technical documentation (1)

At the end of every database it is mandatory to write a technical documentation. Building a database without a documentation means forcing other people to go through all the relations, tables, tables' options, forms, queries, and reports to understand what they do and how are they built. This is especially true when the architecture is complicated, in particular to justify the presence or the absence of junction and details tables.

Writing the technical documentation while the database is built is a good idea to produce a full documentation without forgetting any step. During its writing, we should remember that it is addressed to technicians, i.e. it is not necessary to explain how Access works, what are the motivations of the database and its cultural background and that the documentation can also be schematic.

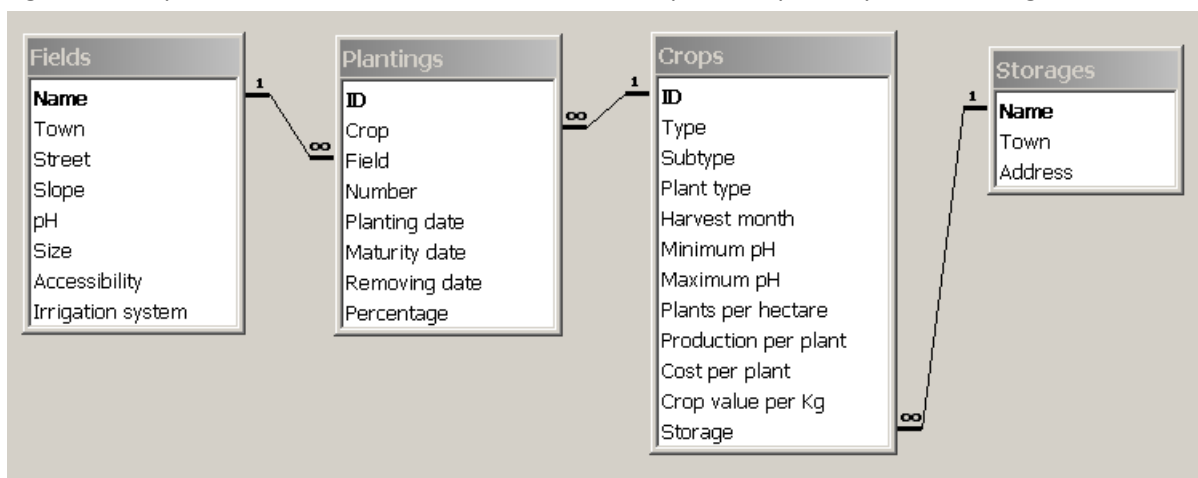
Technical documentation starts with a general description of the database, of the data it contains and of how they are handled. Then it contains an overview of the relationships, better with a picture of the tables and relations, and a justification of the type of each relation, especially of the ones which are not obvious. Then, going through table by table, each field is presented with its type and all its options, giving also a further explanation for the fields whose content is not obvious from the field's name. Now queries are presented, clearly writing their criteria, sorting, grouping, virtual fields, and also how they follow the relations when retrieving data. This last thing, frequently overlooked, is crucial since often two tables can be connected following different paths of relations, with the query resulting in different results. Finally we present forms and reports, with all their features.

10. MyFarm documentation example (1)

MyFarm database contains data about the crops organization of a small fruit farm. We handle data about which crops are planted, the fields and their physical features and where the fruits are stored once removed from the plants. The database contains also historical records, since when a planting is removed it is not deleted from the database but simply a removing date is added.

10.1. Structure (1)

The architecture has the main tables Storages, which contains storage's places data, Fields, which contains the physical data of the fields, and Crops, which contains the biological and commercial features of the crops. Moreover, a junction table Plantings is used to describe what, where and when is planted: this table connects Crops and Fields via a many-to-many relation. The other relation is a one-to-many between Storages and Crops, which underlines the fact that each crop can be put only in one storage.



10.2. Tables and fields (1 only)

Table Fields contains

Name: text, primary key;

Town, Street: text;

Slope, pH, Size: number;

Accessibility: yes/no. This indicates whether a road arrives to the field;

Irrigation system: yes/no, required, default value no.

Table Storages contains

Name: text, primary key;

Town, Address: text;

Table Crops contains

ID: autonumber, primary key;

Type, Harvest month: text;

Plant type: text. This contains the plant type such as tree, bush or herb.

Subtype: text. This contains the specific type of fruit, for example Pear Kaiser, Apple Golden;

Minimum pH, Maximum pH, Plants per hectare, Production per plant: number;

Cost per plant, Crop value per Kg: currency;

Storage: foreign key, takes values from Storages table.

Table Plantings contains

ID: autonumber, primary key;

Crop: foreign key, takes values from Crops table.

Field: foreign key, takes values from Fields table.

Number: number. This indicates how many plants of Crop type are planted in Field;
Planting date, Maturity date, Removing date: date/time;
Percentage: number. This indicates which percentage of the field is used for this crop.

10.3. Tables and fields (2)

Table Fields contains

Name: text limited to 50 characters, primary key, required, no zero length;
Town, Street: text limited to 50 characters;
Slope: number, integer with 0 decimal digits, default value 0;
pH: number, single with 2 decimal digits, default value 5, must be between 0 and 10;
Size: number, single with 2 decimal digits, default value 0, must be positive;
Accessibility: yes/no. This field indicates whether a road arrives to the field;
Irrigation system: yes/no, required, default value no.

Table Storages contains

Name: text limited to 50 characters, primary key, required, no zero length;
Town, Address: text limited to 50 characters;

Table Crops contains

ID: autonumber, primary key;
Type: text limited to 50 characters, required, no zero length text, indexed (with duplicates), non mandatory predetermined list with values Apple, Pear, Strawberry, Grapevine;
Subtype: text limited to 50 characters, required. This contains the specific type of fruit, for example Pear Kaiser, Apple Golden;
Plant type: text limited to 50 characters, non mandatory predetermined list with values Tree, Bush, Vine, Herb;
Harvest month: text limited to 50 characters, (2) non mandatory predetermined list with the twelve months as values, (3) there is an extra table Months, containing the twelve months, for which this field is a foreign key;
Minimum pH, Maximum pH: number, single with 2 decimal digits, required, default value 5, must be between 0 and 10;
Plants per hectare: number, integer with 0 decimal digits, default value 0, must be positive;
Production per plant: number, single with 2 decimal digits, default value 0, must be non negative;
Cost per plant: currency, euro format, single with 2 decimal digits, default value 0;
Crop value per Kg: currency, euro format, single with 2 decimal digits, default value 0;
Storage: foreign key, takes values from Storages table.

Table Plantings contains

ID: autonumber, primary key;
Crop: foreign key, takes values from Crops table.
Field: foreign key, takes values from Fields table.
Number: number, integer with 0 decimal digits, default value 0, must be non negative. This field indicates how many plants of Crop type are planted in Field;
Planting date: date/time, format dd/mm/yyyy, required, indexed (duplicates allowed);
Maturity date, Removing date: date/time, format dd/mm/yyyy;
Percentage: number, percentage format, single with 2 decimal digits, default value 0%, must be between 0% and 100%. This indicates which percentage of the field is used for this crop.

(3) To table Plantings there is also a table validation rule that checks that Maturity date and Removing date, when they exist, are not before Planting date.

10.4. Forms (2)

Form Insert/Modify Crops shows the crops with their storage, using the relation between Storages and Crops. It is possible to modify them or to insert new ones, but deletion of crops is forbidden.

10.5. Queries (1)

Query Type+Subtype+Field Without Irrigation Query shows the field without an irrigation system with the crop type and crop subtype that are and were planted in that field. This query retrieves the fields with a No in the Irrigation system field and then, thanks to the junction table Plantings and following the two relations, retrieves the crop's type and subtype.

10.6. Reports (2)

Report Crops by harvest month displays, grouped by harvest month and then grouped by crop type, the crop subtype and the corresponding field and town where the crop is or was planted. It takes data from tables Crops and Fields, connected through the junction table Plantings and the two relations.

Index

- Σ button, 17
- 1 side, 6, 9, 11, 13, 18, 21
- ∞ side, 6
- Abs, 16
- Allow Deletion, 15
- Allow Edit, 15
- Allow Insertion, 15
- allow zero length, 13
- AND, 19, 20
- append query, 18, 20
- architecture, 21
- AS, 19
- ASC, 19
- ascending, 15, 19
- autonumber, 12
- Avg, 17, 20
- AVG, 20
- backup database, 10
- comma-separated text file, 14
- composite key, 5
- composite primary key, 22
- compound key, 5
- Count, 17, 20
- COUNT, 20
- create query, 18, 20
- create using wizard, 11
- criterion, 16, 17
- currency, 12
- data, 12
- database file, 10
- database management program, 10, 19
- Datasheet View, 12, 15, 16, 17, 18, 19
- date, 12
- Date, 16
- DateAdd, 16
- DateDiff, 16
- decimal digits, 22
- decimal positions, 13
- default value, 13, 22
- DELETE FROM, 20
- delete query, 20
- DESC, 19
- descending, 15, 19
- Design View, 12, 14, 15, 17, 18, 19
- details table, 8, 9, 21
- dialog box, 16
- disordered, 3
- drop down menu, 13
- drop-down menu, 12, 22
- empty field, 4
- enforce referential integrity, 11, 22, 23
- enforcing referential integrity, 10, 22
- Excel file, 14
- Exp, 16
- expression, 16
- Expression Builder, 14, 16
- field, 3, 6, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22
- field type, 12, 22
- filter, 10
- foreign key, 5, 6, 7, 9, 12, 13, 14, 19, 21, 22, 23
- form, 10, 14
- Form View, 15
- format, 13
- FROM, 19, 20
- graphical interface, 11
- Group By, 17
- GROUP BY, 20
- header, 2
- hyperlink, 13
- ID, 5, 9, 21, 22
- import data, 14
- importing tables, 14
- independent, 2
- index, 14, 22
- information redundancy, 3
- inner join, 19, 20
- INNER JOIN, 19, 20
- INSERT INTO, 20
- Int, 16
- INTO, 20
- junction table, 7, 8, 9, 19, 21
- Like, 17
- Log, 16
- Lookup Wizard, 12, 13, 22, 23
- main database window, 12
- management program, 10
- many side, 6, 7, 10, 13, 14
- many-to-many, 7, 21
- many-to-one, 6, 9, 21

memo, 12
 Microsoft Access, 10, 12
 MySQL, 10

 Northwind, 11
 no-show, 15, 17
 number, 12

 OLE object, 12
 ON, 19, 20
 one-to-many, 6, 21
 one-to-one, 7, 21
 Oracle, 10
 ORDER BY, 19
 orphan, 9

 paper diagram, 21, 22
 PostgreSQL, 10
 predetermined list, 13, 22, 23
 predetermined values, 22
 primary key, 5, 6, 7, 9, 12, 13, 19,
 21, 22
 Print View, 18

 query, 9, 10, 14, 15, 16, 17, 18,
 19, 20

 record, 3, 5, 7, 8, 10, 17, 18
 redundancy, 3
 referential integrity, 5, 9, 10, 11,
 12, 13, 19
 relation, 6, 7, 9, 10, 11, 12, 13,
 14, 15, 18, 19, 20, 21, 22
 relational database, 2, 6, 7
 relationships diagram, 11, 13, 22
 report, 9, 10, 15, 18
 required, 13, 22, 23

 save, 10, 14, 18
 SELECT, 19, 20
 select query, 15, 17, 20
 SET, 20
 simple query, 17, 20
 single table database, 3, 4
 size, 13
 sort, 10, 15, 18
 splitting the table, 3, 4
 SQL, 19, 20
 SQL View, 19, 20
 Sqr, 16
 Standard Query Language, 19
 Sum, 17, 20
 summary query, 17

 Summary query, 20
 surrogate key, 5

 tab-delimited text file, 14
 table, 2, 3, 6, 7, 8, 9, 10, 11, 12,
 14, 15, 18, 19, 20, 21, 22, 23
 table validation rule, 14, 16, 23
 technical documentation, 23
 text, 12
 time, 12

 UPDATE, 20
 update query, 18, 20

 validation rule, 13, 14, 16, 22
 validation text, 13, 14, 22
 view, 15, 17, 18
 virtual table, 15
 Visual Basic macros, 11

 where, 19
 Where, 17
 WHERE, 19, 20
 wildcard, 17

 Year, 16
 yes/no, 12