CHAPTER I - BASIC FEATURES[1]

1. Introduction

This document introduces the basic features of an information system that mimics the web-page of a Sport Centre, i.e. a multi-sports gymnasium. The application is based on a Relational Databased, developed in ACCES 2010; specifically, there are 16 Tables, 43 Forms, 22 Principal Queries and 3 Reports.

The graphical design of the above mentioned objects will be pinpointed very quickly[2], as the main goal of this document is to explain how to use Visual Basic Code to automatize recurrent operations and to create stable, secure and professional applications[3].

2. The Sport Center

2.1. The structure of the center

The sport center has:

- A gym with two weighs rooms (also equipped with cardio-fitness machines) and two fitness rooms where courses (step, pilates, yoga, FAST, etc.) are held;
- A wellness center (with SPA, Sauna, Hammam, Massage, etc.);
- 25 meters swimming pool (with 8 swim lanes) approved for FIN competitions. The swimming pool has an extra space (25x6m and 5m deep) that can be used (to enlarge the total area of the swimming pool) in case of water-polo trainings or matches. Anyhow, this additional space can be used to create 4 additional swim lanes, or it can be used for scuba diving activities.
- A small and swallow pool (20x8 and 80cm deep) that is used for soft gym in the water and/or for rehab.

2.2. Kind of Membership Cards

The center has a vast set of offers; the basic passes are the following ones:

- Water Courses pass - This pass allows to participate to all water courses (i.e., the ones held in the small swimming pool);
- Earth Courses pass - This pass allows to participate to all earth courses (i.e., the ones held outside, or in the two fitness rooms);
- Swimming Courses pass - This pass allows to participate to all swimming courses (for novice, intermediate and advance swimmers)

---

1 Programming features that are introduced for the first time are briefly pinpointed at the end of the chapter.

2 For more details concerning the creation of query, forms and record the interested readers are referred to the official manual "Microsoft Access 2013 Step by Step"

3 All the code included in the document is also available in the ACCESS file DBGym_Release03.accdb

- Weight Room pass - This pass gives access to the weight rooms; customers can use all the equipment and can ask (free of charge) for supports and/or for personal training schedule to the staff of the gym;
- Free Swim pass - This pass gives access to the 25m swimming pool, but only in the swim lanes that, according to the schedule of the swimming pool, have not been reserved to swimming courses and/or to masters' teams;
- SPA pass - This pass gives access to the Wellness Area.

To create personalized membership cards, customers can combine, as they like, all the above-mentioned passes and they can choose the period of validity of the membership card. Monthly, trimestral, annual and bi-annual subscriptions are available.

Lastly, customers can choice the daily time window during which they can access to the gym. The available time windows are listed below:

- Morning: 7.30 - 12.30;
- Lunch Time: 12.00 - 14.30;
- Night: 19.00 - 23.00;
- Open: 7.30 - 23.00.

Obviously, the full price of a membership card depends on the specific configuration made by the customer (in terms of passes, period of validity and daily time window). Discounts and special offers are also available in case of:

- Anticipated renewal and/or upgrade of the current membership card;
- University Students and Staff;
- Teenagers;
- Elder people over 60 years old.

Anyhow, the commercial staff of the center may offer, at his own discretion, additional discounts; yet, the total discount is limited to 15% of the full price.

2.3. Users' processes

The information system must be able to manage, with different levels/privileges of access, four types of users:

- Potential Customers;
- Registered Customers;
- Technical Staff i.e., Instructors;
- Center's Administrators.

Potential Customers (PCs)

PCs have a read-only access to a limited part of the data stored in the RDB.

For instance, they can see:

- General information concerning the Center;
- The list of active courses;
- The weekly time table (of the active courses);
- Available passes and prices;
- Promotions;
- …

Making an enquiry for the purchase of a membership card is the only dynamic activity that a PC can do.

Specifically, on the web site of the center a PC can fill a form (with his or her personal details) and creating his or her membership card, by selecting a combination of different passes. When the form has been completed in all its parts, the system returns a printable report containing the list of all the passes and other services selected by the PC, and a rough quotation valid for a week. With a printed copy of the quotation the PC can go to the Sport Center's reception desk to purchase the membership card.

Registered Customers (RCs)

RCs can perform all the actions allowed to PCs and, additionally, they have a personal page that can be accessed logging in to the site, using a valid User Name and Password.

Specifically, in their personal page, RCs can:

- View a summary of the state of their membership card (starting and ending date, missing days to expiration date, validity of the health certificate, list of accessible courses, etc.);
- Change their personal password;
- Check the detailed weekly time-table of the courses and the areas (fitness rooms and swim lanes) that are occupied (i.e., not available) by competitive teams or by curses;
- Make or Cancel a reservation for a specific course;
- Check the name of the instructors that will be in the fitness rooms at a certain hour of a certain day.

Concerning reservations, we observe that, for each course there is a maximum number of possible participants (this number depends both on the capacity of the room, in which the course is held, and on the type of exercises that will be made during a class). In order to clarify this concept, let us assume that the course of Pilates is held in Room#1 every Monday (from 13.00 to 14.00) and every Friday (from 12.00 to 13.00). Pilates is an "Earth course" and so all the *n* customers that have an "Earth Courses" pass with validity in the "lunch time window" can join it, provided that there is still availability. Indeed, if MCR_1 is the maximum number of people that can

attend a Pilates class, if $n > MCR_1$ not all the customers will be able to participate, together, to the same class. To obviate this fact, in order to participate to a specific class - for instance Pilates on Monday November 14th - customers that are allowed to it must make a reservation at least one day in advance.

Also, when a customer makes a reservation, if there is still room, the information system saves the reservation and adds it to the list of classes reserved by the customer. Vice versa, if the maximum admissible number has already been reached, the customer can be placed in a "waiting list". If, later on, as a result of one or more cancellations, some places become available once again, the information system warns the secretary of the center who will call the customers of the waiting list - in order of registration - to offer them the opportunity to attend the class.

Similarly, cancellations must be made at least one day before the date in which the class is scheduled. Also, in order to avoid "opportunistic behaviors", customers that have not cancelled their reservation and that miss the class are blocked by the information system. In other words they cannot make new reservations, unless they pay a (small) fine at the reception desk, in order to be unblocked.

Instructors

As for the RUs, also instructors have a personal page, but thy have a very limited access to the data of the RDB. Specifically, they can only:

- Change their password;
- Check the weekly time table of the courses;
- Check their working shifts and classes that have been assigned to them;
- Print the list of the customers that registered for one of their class;
- Signal (and block) missing customers, which did not participate to a class.

With respect to the last point, we note that, before going to a class, instructors are required to print the list of participants, take the attendance and record unjustified absents. Next, at the end of the class, instructors have to fill an on-line module to signal the unjustified absents that, as a result, will be automatically blocked by the system.

Center's Administrators (CAs)

Administrators do not have a personal page, but can access (in read only mode) to all the data stored in the RDB. In addition they should have:

- One or more forms to add new customers and/or to modify existing ones;
- One or more forms to create new offers for a specific customer;
- One or more forms to update courses, to associate courses to trainers, to create the time table, to modify passes, prices, etc.

- A control panel displaying graph and statistics concerning the current situation of the Center. For instance, it may be interesting to know the distribution of the passes, which are the most crowded times, which the more appealing courses and so on.

3. The Tables

The RDB is based on 16 Tables, described below.

MEMBERSHIP S' TYPES (ABBONAMENTI)

This table is made of records of 9 fields:

- ID - Primary Key, Type Integer;
- Membership 's Name - Type String, is the name of a specific membership card;
- Weight Room, Earth Courses, Water Courses, Swimming Pool, Swimming Courses, Wellness Center, Monthly Price - Type Booleans- These fields are used to define the services offered by a specific course.
- Price - Type Currency.

This structure makes it possible to define all the possible combinations of the 6 basic passes. For instance, a Basic and VIP memberships may be defined as follows:

Tab. 1.1. *Membership's Types*

ID	NAME	WR	ECs	WCs	WP	SCs	WC	Price
1	BASIC	1	0	0	0	0	0	50 €
2	VIP	1	1	1	1	1	1	150 €

Concerning the Price field, this is the standard monthly price for a basic duration of one month. Clearly, when a customer creates his own membership configuration, the standard price will be modified taking into account: the total duration of the membership card, the time window and other discount features. For instance, if a teenager selects a basic membership card for 6 months in the "lunch" time window, the price could be calculated as follow:

$$Total\ Price = (Basic\ monthly\ price \times Total\ duration) \times (1 - Min(MD\%; (D_{W,l} + D_{D,6} + D_T))$$

Where:

- MD% is the maximum allowable discount rate;
- $D_{W,l}$ is the discount rate for the "lunch time window";
- $D_{D,6}$ is the discount rate for a six month duration;
- D_T is the discount rate for a teenager.

DURATIONS AND TIME WINDOWS (DURATE E FASCE ORARIE)

DURATION lists all the possible durations of a membership card (i.e., month, trimester, six months, etc.) and the corresponding discount rate D_{di}.

Specifically, records have the following three fields:

- ID - Primary Key, Type Integer;
- Duration - Type String (with limited input values, i.e., month, trimester, etc.)
- Discount Rate - Type Percentage.

Clearly there is not discount in case of a duration of one month, i.e., $D_{D,1} = 0$.

Similarly, TIME WINDOW lists all the available time windows (i.e., morning, lunch time, etc.) and the corresponding discount rate $D_{W,i}$. Obviously the full time discount rate equals zero, i.e., $D_{T,f} = 0$.

DISCOUNTS (SCONTI)

This table defines the different kind of discounts (or even extra prices) in use in the Center. Specifically, it is a support table (i.e., a table that is not linked to any other table and that is used only to store useful data) and has records of six fields:

- ID - Primary Key, Type Integer;
- Name - Type string, is the name of the discount;
- Description - Type Memo, is a brief textual explanation of the discount;
- Condition - Type Integer, it is a field that define the logical condition used to decide if an user can take advantage of a certain discount;
- Discount - Type Percentage, it is the discount rate;
- Extra - Type Percentage, it is the extra rate.

For instance, the teenager discount, the student discount, and the extra price needed to add the Wellness Center to a standard membership card could be codified as in Table 1.2:

Tab. 1.2. Discounts' *Types*

ID	*Name*	*Description*	*Condition*	*Discount*	*Extra*
1	*Teenager*	*Users that are 19 years old or younger*	*≤19*	*25%*	*0%*
2	*Student*	*Users that study at high school or College*	*= Student*	*15%*	*0%*
	…	*…*	*…*	*…*	
N	*Wellness*	*Users that want to add the Wellness Center*	*…*	*0%*	*10%*

COURSES and COURSES_INSTRUCTORS (CORSI, CORSI ISTRUTTORI)

COURSES is the table that contains and describes all the courses activated by the Sport Center.

Specifically it is made of record of six fields:

- ID - Primary Key, Type Integer;
- Name - Type String;
- Description - Type Memo, it is a brief description of the courses;
- Level - Type String with a limited number of input (i.e., very easy, easy, …, challenging)
- Category - Type String with a limited number of input (i.e., Earth, Water, etc.)
- Max Number - Type Integer, is the maximum number of people that can attend a class.

COURSES_INSTRUCTORS

This is a bridge table, which is needed to split the MTM relations between COURSES and INSTRUCTORS in two OTM relations.

This table is made of records of three fields:

- ID_Course - Primary and Forward Key, Type Integer;
- ID_Instructor - Primary and Forward Key, Type Integer;
- Principal - Type Boolean, used to specify if an instructor is the main instructor or a possible replacement.

TIME TABLE and WEIGTH'S ROOM TIME TABLE (PALINSESTO E PALINSESTO PESI)

TIME TABLE is used to define the weekly schedule of the fitness and water courses. Specifically, the time and the day on which a course will take place. Its records are made of the following fields:

- ID - Primary Key, Type Integer;
- ID_Course - Forward Key, Type Integer;
- ID_Instructor - Forward Key, Type Integer;
- Start Time - Type Date/Time;
- Day - Type String with a limited number of input (i.e., Monday, Tuesday, …)
- Room - Type String with a limited number of input (i.e., Room1, Room2, …, Swimming Pool)

The date of a class of a course is not included in the above-mentioned table. This is absolutely logic; in fact, using the date instead of the day, the number of records to be created would increase significantly. Let us suppose there are m different courses repeated k times each week. If the time table was updated every n weeks, then $(n \times m \times k)$ records should be manually updated, as shown in Tab. 1.3 (for n = 10, k = 2 and m = 4):

Tab. 1.3. *The TIME TABLES with the Date field*

ID	Course	Date	…	…
1	Zumba	09/01/2017		
2	Zumba	12/01/2015		
3	Zumba	16/01/2017		
…	…	…	…	…
8	Zumba	02/02/2017	…	…
…	…	…	…	…
80	Pilates	04/02/2017	…	…

Conversely, using the day of the week, the updating procedure is much easier because:

- There is no need to specify a precise date (but only a day), a fact that reduces the probability of making input errors;
- The number of records that must be updated drops to $(m \times k)$;
- Records are inserted only once; later, in case of changes to the time table, only records that are affected by a modification will be (partially changed).

An example is given in Table 1.4, where Zumba is moved from Monday to Wednesday

Tab. 1.4. *An example of modification*

ID	ID_Course	ID_Instrctor	Time	Day	Room
Original Table					
1	1 - Zumba	1 - Keats	12:00	Monday	S1
2	1 - Zumba	1 - Keats	08:30	Friday	S2
…	…	…	…	…	…
2n	n - Pilates	5 - Denson	14:00	Sunday	S1
Modified Table					
1	1 - Zumba	1 - Keats	12:00	Monday	S1
2	1 - Zumba	1 - Keats	08:30	**Wednesday**	S2
…	…	…	…	…	…
2n	n - Pilates	5 - Denson	14:00	Sunday	S1

We anticipate that, in order to register to a class (of a course) one will need to indicate a specific date. As we will see at the end of this chapter, this is not a major problem. Let us suppose, for instance, that on Tuesday, November 16th, a user decides to sign to a Zumba Class. If Zumba is held every Monday, every Wednesday and every Friday, the user is free to sign, in the current week, only to the classes held on Wednesday and/or on Friday; in the following weeks he is free to sign in any days. Generating the dates of these classes is not difficult (we will see the VBA code later on), indeed:

- Wednesday of the current week is one day from now i.e., November 16th
- Friday of the current week is three days from now i.e., November 18th

- Monday of the next week is six days from now i.e., November 21th
- …

The WEIGTH'S ROOM Time Table is analogous to the precedent one and will not be discussed any more.

RESERVATIONS (PRENOTAZIONI)

Reservation is a sort of bridge table connecting COURSES and CUSTOMERS, indeed, it should be clear that a customer (of the Sport Centre) can participate to several classes and that a class can be joined by several customers. However, since the same course is repeated many times during the year, ID_Customer and ID_Course (two forward keys of this table) are not enough to define a unique primary key. In other words, in order to identify a specific class there is the need to consider, jointly, ID_Course and Date. Additionally, if a course can be repeated more than once during the same day (at different times), also the Time field is needed to identify a specific class. Owing to these issues, to identify a specific reservation made by a customer there is the need to consider, jointly, four fields: *(i)* ID_Course, *(ii)* Date, *(iii)* Time and *(iv)* ID_Customer.

Due to this complexity, for the sake of easiness, we decided to introduce a unique ID field to be used as Primary Key, as shown in the fields' list that follows:

- ID - Primary Key, Type Integer;
- ID_Course - Forward Key, Type Integer;
- ID_Customer - Forward Key, Type Integer;
- Registration Date - Type Date/Time;
- Class Date - Type Date/Time;
- Class Time - Type Date/Time;
- Registration's State - Type Boolean;
- Absent - Type Boolean.

The last two fields deserve some further comments.

Registrations' State is used to specify if, at the time of the registration, the customer has been enrolled to the class or if he has been placed in the waiting list. More precisely, if a customer decides to sign to a class that has some vacancies, then a value equal to one (i.e., true) will be registered in the Registrations' State field. Conversely, if the class is already full, a value equal to zero (i.e., false) will be inserted.

The Absent field is used to take trace of the customers that signed to a class but, later, they did not join it. More precisely, if a customer having a State Registration equals to 1 (i.e., he registered and he did not cancel the registration) does not show to the class, the Absent field will be set to one (i.e., true).

CUSTOMERS, INSTRUCTORS and ADMINISTRATORS (UTENTI, ISTRUTTORI, AMMINISTRATORI)

These tables contain the master data of the three kinds of users that can access to the web site of the Sport Centre. Clearly the tables are very similar and have a common number of fields. These are: name, surname, sex, address, user name and password (the last two are needed for logging in). Concerning the Address field, we note that, according to the second normal form, address should be divided in many fields such as country, region, city, street and zip code, because only in this way it is possible to use the address as a filtering parameter. Nonetheless, in this case, for the sake of simplicity, we preferred no to comply with the second normal form; as a matter of fact it is rather unusual that a gym has the necessity to filter customers depending on their address. Among these three tables, CUSTOMERS is the most "complex" one, since, in addition to personal fields, it includes other fields needed to specify the membership type chosen by a customer. Specifically, the used fields are the following ones:

- ID - Primary Key, Type Integer;
- Name - Type String;
- Surname - Type String;
- Sex - Type String with limited number of input (Male, Female);
- Birth Date - Type Date/Time;
- Telephone - Type String (to include special characters such as +39/0187/420531);
- Email - Type String;
- Address - Type String;
- ID_Membership - Forward Key, Type Integer;
- ID_Duration - Forward Key, Type Integer;
- ID_Time_Window - Forward Key, Type Integer;
- Wellness - Type Boolean, one if the customer has access to the wellness area;
- Starting Date - Type Date/Time, the starting date of the membership card;
- Total Price - Type Currency, the price paid by the customer for the membership card;
- Applied Discount - Type Percentage;
- Health Certificate - Type Integer (0 $\rightarrow$ not handed over, 1 $\rightarrow$ standard, 2 $\rightarrow$ competitive, …);
- Certificate Expiring Date - Type Date/Time
- Class Time - Type Date/Time;
- Blocked - Type Boolean (if true, for unjustified absence, the customer cannot make new reservations).

As above mentioned, membership cards can be customized by combining different passes, durations and time windows. The selection is coded using the three foreign keys of this table; for example, a possibility could the

following one: ID_Membership = VIP, ID_Durtion = Year, ID_Time_Window = Open. We also note that the Wellness Boolean field is used to indicate whether or not a customer has access to the wellness area.
Total Price, Applied Discount and Starting Date define the last features of the membership card. Please note that there is not an Ending Date field. As a matter of fact this would be a calculated field, since its value can be easily determined knowing the starting date and the duration of the membership.
We conclude this section that, due to the similarity of the CUSTOMERS, INSTRUCTORS and ADMINISTRATORS, instead of three tables one could have used a single table called USERS. However, this approach, although correct, has some limits:

- There is the need to add an additional field User_Type {Administrator, Customer, Instructor} to classify the users;
- Some fields are exclusive for a specific kind of user. For example all the fields relative to a membership card are specific to the customers; thus, these fields would be null for all the other kinds of users;
- Using a single table some relationships would not be clear. For instance there is a relationship between customers and courses and, similarly, there is a relationship between instructors and courses. Yet, the nature of these two relations is different: the first one specifies the courses joined by a customer, the second one species the courses that are assigned by an instructor. Using a single table it would be impossible to clarify the different nature of these relations.

ACCESSES (INGRESSI)

Customers have a magnetic badge that opens turnstiles (at the entrance of the Sport Centre) and the doors of each different area. Also, when the user uses the badge to access or the leave a certain area, the badge records (on the information system) the ID of the customer, the date, the time and the area where the user has gone.
The ACCESSES table is used, exclusively to store data concerning the accesses of the customers. These data are precious, as they could be used to make statistical analysis in terms of arrival rate (per day, time and area), peak hours, etc.
The fields of the records of this table are the following ones:

- ID - Primary Key, Type Integer;
- Date - Type Date/Time;
- Time In - Type Date/Time;
- Time Out - Type Date/Time;
- Area - Type String with limited number of input (Fitness area, Swimming Pool, Wellness, etc.);

QUOTATIONS (PREVENTIVI)

This table is used to save quotes (for new membership cards and/or for membership's renewals) asked by potential or registered customers. This table is very similar to the CUSTOMERS one, as it has all the personal fields and the memberships fields that appear, also, in the CUSTOMER table. However, QUOTES has also four specific fields:

- ID_Customer - Forward Key, Type Integer;
- Offered Price - Type Currency;
- Validity - Type Date/Time;
- Note - Type Memo, contains particular requests made by the customers.

ID_Customer is the forward key that links QUOTES and CUSTOMERS; as above mentioned also registered customers can ask for a quotation for a new membership card or for the renewal of the current one. Clearly, if the quotation is asked by a potential customer, the ID_Customer field will be null (a licit value for a forward key). Validity displays the date within which the customer has to sign the offer.

Offered Price is the price automatically proposed by the system and that has to be confirmed by an administrator at the time of the contract stipulation.

OFFERS (OFFERTE)

This table contains specific offers made by the administration to a selected customer. At present this table is just a draft, as it only contains the following fields:

- ID_Offer - Primary Key, Type Integer;
- ID_Customer - Forward Key, Type Integer, the customer to whom the offer refers to;
- Offered Price - Type Currency;
- Validity - Type Date/Time, the expiring date of the offer;
- Description - Type Memo, the description of the offer.

BLOCKED_USERS (TblAPPOGGIO_BLOCCATI)

This is a support table (i.e., a table that is not linked to any other table and that is used only for updating purposes) that is used to update, in a safe way, the CUSTOMERS table, any time the value of the Blocked field (of one or more customers) changes from True to False. Its fields are the following ones:

- ID - Primary Key, Type Integer;
- ID_Customer - Type Integer (in this case this is not a forward key, but just a reference field);
- ID_Course - Type Integer (in this case this is not a forward key, but just a reference field);
- Date - Type Date/Time;
- Time - Type Date/Time.

3. Relationships

All the relationship among the 17 tables of the RDB are shown in Figure 1.2. As it can be seen:

- USERS is in OTM relation with:
  - QUOTES - A customer may have asked for different quotes;
  - OFFERS - A customer may have received different offers;
  - ACCESSES - A customer enters many time in the Sport Centre;
  - RESERVATIONS - A customer makes many reservations to one or more courses.
- USERS is also in MTO relation with DURATIONS, TIME WINDOWS and MEMBERSHIPS; as above mentioned these relations codify the kind of membership chosen by a customer.
- COURSES is in OTM relation with:
  - TIME TABLES - A course may be repeated many time during a week (in some cases it may be repeated more than one even during the same day)
  - RESERVATIONS - A course may be booked more than once by customer (clearly for classes held at different dates);
- INSTRUCTORS is in OTM relation with:
  - TIME TABLES - An instructor may have more than a class during a week;
  - WEIGTH TIME TABLES- An instructor has to stay in the weights' area (with support and control duty) for a couple of days per week.

COURSES and INSTRUCTORS are in MTM relation, since a course may be assigned to two or more instructors (the first is the principal instructor, the other one are the replacements) and an instructors may have more courses. Thus this relation is split using the bridge table COURSES_INSTRUCTORS

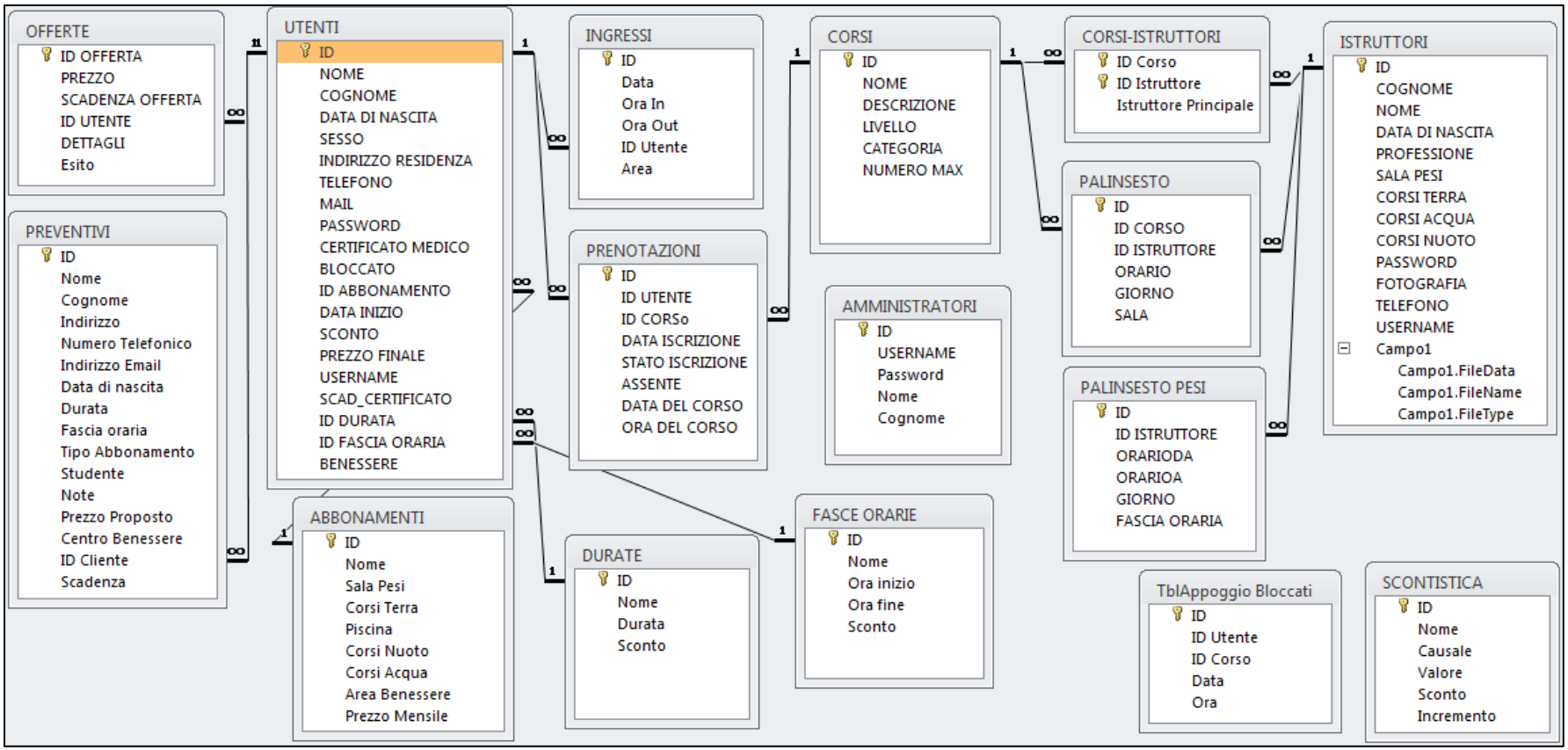

Fig. 1.2. *The entities – relations model*

Lastly we note that there are three tables, namely ADMINISTRATORS, DISCOUNTS and BLOCKED_USERS, that are not linked to any other table. Concerning DISCOUNTS and BLOCKED_USERS, as we have mentioned before, they are support tables and, as such, they do not have any relationships. Concerning ADMINISTRATORS, the absence of relationships is due to the hypothesis that administrators play a purely administrative role and did not participate actively to the sport activities of the center (i.e., they do not join courses, do not make reservations, etc.). To include this possibility, the ADMINISTRATORS table should be opportunely modified to include all the needed relationships or, in a more easy way, ADMINISTRATORS and CUSTOMERS should be joined in a single table.

4. Queries

We conclude the first chapter with a few hints concerning some of the saved queries included in the RDB.

4.1 Queries used to create forms

Earth Courses' Time-Table (Orari Terra Campi Incrociati) - This is a Cross-Table Query used to generate the time table of the Earth Courses.

Specifically, as shown in Table 1.5 it specifies the day, time and fitness room where each class (of each earth course) will be held.

Tab. 1.5.

The Earth Courses' Time Table

Lunedì	Martedì	Mercoledì	Giovedì	Venerdì	Sabato	Domenica	ORARIO	SALA
	TOTAL BODY						09:00	SALA 1
PILATES		PILATES		PILATES			09:00	SALA 2
			TOTAL BODY	TOTAL BODY			10:00	SALA 1
						20'20'20'	11:00	SALA 1
					SPINNING	SPINNING	11:00	SALA 2
PUMP	GAG	STEP & TONE	FAT BURN	BODY SCULPT	TOTAL BODY	FAT BURN	12:00	SALA 1

It is interesting to note that, as shown in Figure 2, this query does not operate on a table, but it operates on a (pretty simple) join query, namely OrariTerra, whose SQL code is reported below:

```
SELECT COURSES.Name, [TIME TABLE].[Day], [TIME TABLE].[Start Time], [TIME TABLE].[Room]
FROM COURSES INNER JOIN [TIME TABLE] ON COURSES.ID = [TIME TABLE].[ID_Courses]
WHERE COURSES.Category = "Earth Courses"
```

We briefly recall that, in order to create a Cross-Table query, one has to select *Design Query* on the *Create* menu and, next, one has to select the *Cross-Table* option in the same menu. By doing so, the input form changes as shown in Figure 1.2, below. Specifically, in the additional row called *Cross-Table* one has to specify which fields must be used as row and column headers and which field must be displayed (as value) in the output table. For the latter one in the Formula field one has to specify also the computation that must be made. Possible options are: Sum, Average, Min, Max, First, Count, SDev, etc.

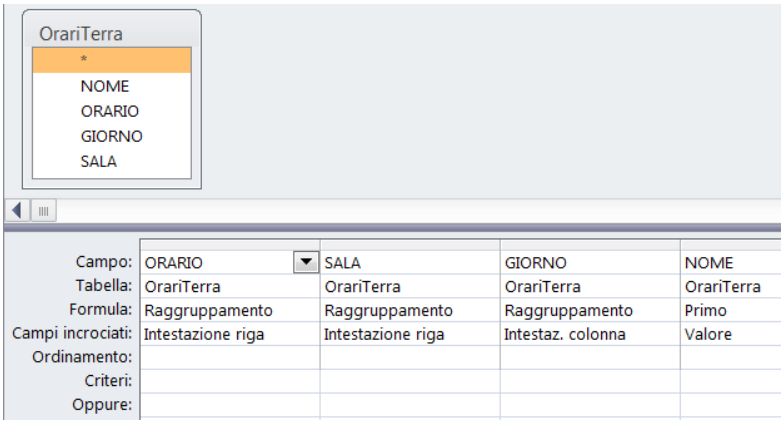

Fig. 1.2. *Creating The Cross Table Query*

Weights' Time-Table (Palinsesto Pesi Campi Incrociati) - This is a Cross-Table Query, similar to the previous one, which shows, for each day, the instructor in charge of the weights' room.

Full Time-Table (Orari) - This is a Pivot query that shows the whole Time Table of the Sport Centre. As shown by Figure 1.3, as in the Cross Table previously described, also in this case Days are on the columns and different Times are on the rows. However, this time, each cell displays all the courses (no only the earth ones) that are scheduled at the same time and day. It is also possible to filter the Time Table using Category, Instructor and Room as filtering criteria.

Orari

CATEGORIA	ISTRUTTORE	SALA
CORSI TERRA	(Più elementi)	Escludi: PISCINA

	GIORNO						
	Domenica	Lunedì	Martedì	Mercoledì	Giovedì	Venerdì	Sabato
ORARIO	CORSO	CORSO	CORSO	CORSO	CORSO	CORSO	CORSO
11:00	20'20'20'						SPINNING
	SPINNING						
12:00			GAG	STEP & TONE		BODY SCULPT	
13:00				SPINNING		SPINNING	
14:00		GAG		GAG			
18:00					BODY SCULPT		
19:00				20'20'20'	STEP & TONE	SPINNING	
				SPINNING			
20:00						GAG	

Fig. 1.3. *The Pivot Query*

Briefly, we remember that, in order to create a Pivot Query, it is sufficient to create a standard selection query (by operating in design view or, straightly, by typing the SQL code).

For instance, in the present case the query is the following one:

```
SELECT INSTRUCTORS.Surname AS Instructor, COURSE.Name AS Course, COURSE.Category, _
                                    [TIME TABLE].Day, [TIME TABLE].[Start Time],[TIME TABLE].[Room]
FROM COURSES INNER JOIN (INSTRUCTORS INNER JOIN [TIME TABLE] ON _
                        INSTRUCTORS.ID = [TIME TABLE].[ID_Instructor]) ON COURSES.ID = [TIME TABLE].[ID_Course]
```

Once the query has been created, one has to select, from the *View Tab*, the *Pivot View* option. This creates an input form (as the one shown in Figure 4) that allows defining rows and columns' headings, values to be displayed and, eventually, one or more filtering parameters. Obviously, headings, values and filtering parameters must be returned by the underlying selection query

Courses & Instructors (Corsi Istruttori) - This is a simple selection query that shows all the instructors that have the adequate skills to be in charge of a course. The interest thing is that this query is dynamic; as we will see in the next sections, the information system has a navigation form (namely Instructors' List), accessible both by registered and potential customers, that allows one to select a specific instructor to see all the courses assigned to him. To this aim the underlying query operates on three tables (namely COURSES, COURSES_INSTRUCTORS and INSTRUCTORS) and makes use of Public functions written in VBA. The SQL code is the following one:

```
SELECT COURSES.Name
FROM COURSES INNER JOIN (INSTRUCTOR INNER JOIN COURSES_INSTRUCTORS ON _
                                    INSTRUCTORS.ID = COURSES_INSTRUCTORS.ID_Instructor) ON _
                                                COURSES.ID = [COURSES_INSTRUCTORS].[ID_Course]
WHERE (ISTRUCTORS .Name= GetName()) AND (ISTRUCTORS.Surname = GetSurname())
```

GetName[4] is a public function that reads and returns a public variable (i.e., Instructor_Name), as shown below:

```
Public Function GetName() As String
  GetName = Instructor_Name
End Function
```

[4] *GetSurname() is identical, but reads and returns the public variable Instructor_Surname*

Also, when the user double clicks an instructor (on the list displayed on the form), the following code is triggered:

```
Private Sub Cmb_DblClick (Cancel As Integer)
Dim FullName() As String
   ' On the Form, instructors are written according to the following convention: Name-Surname
    FullName = Split(Istruttore, "-") 'The split function is used to get the name and the surname
   ' The global variables are set
   Instructor_Name = FullName(0)
   Instructor_Surname = FullName(1)
   ' The query is called
   DoCmd.Openquery ("Courses & Instructors")
End Sub
```

In this way, before the query is executed the global variables Instructor_Name and Instructor_Surname are set and the GetName and GetSurname functions return the correct values used in the Where Condition of the query. Also note that *Split(Text As String, [Delimiter As String])* is a VBA function that receives a text as input and returns a zero-based, one-dimensional array containing a specified number of substrings. Substrings are identified by the element passed as delimiter. For instance, Disney-Characters = Split("Pippo#Pluto#Topolino", "#") returns a one dimensional array with three elements, i.e., Disney-Characters(0) = "Pippo", Disney-Characters(1) = "Pluto", Disney-Characters(2) = "Topolino".

Docmd is a useful object that has several properties and methods and that makes it possible to execute saved queries, to open form and report, etc.; in the example above the *OpenQuery* method was used.

Time-Tables With Dates (Palinsesto Datato) - This is a fundamental query that shows all the classes scheduled in the next seven days, starting from the current date. More specifically, this is the main query used by all the forms that are used (by the registered customers) to make or cancel a reservation to a class. As we previously noted, the Time-Tables does not have any date, but dates become essential to make a reservation.

This query uses all the information of the Time-Table and RESERVATION tables, and takes advantage of some public functions to create the following calculated fields:

- Date - The date of a specific class, calculated with the *ClDate* function;
- Free Places - The number of places that are still available for a class, calculated with the *FrPlaces* function;
- People In - The number of customers that have already make a reservation for a class, calculated with the *TotReg* function;

- People Waiting - The number of customers that are on the waiting lists (clearly queue may be non-null, provided that there are no vacancies in the class), calculated with *InQueue* function.

The SQL code is the following one:

```
SELECT COURSES.*, [TIME TABLE].*, _
        ClDate([TIME TABLE].Day) AS Date,_
        FrPlaces([TIME TABLE].ID_Course, [TIME TABLE].Day, [TIME TABLE].[Start Time]) AS [Free Places],_
        TotReg([TIME TABLE].ID_Course, [TIME TABLE].Day, [TIME TABLE].[Start Time]) AS [People In],_
        InQueue([TIME TABLE].ID_Course, [TIME TABLE].Day, [TIME TABLE].[Start Time]) AS [People Waiting],_
FROM COURSES INNER JOIN [TIME TABLE] ON COURSES.ID = [TIME TABLE].ID_Course
```

The user defined public functions used by this query are listed below.

```
Public Function ClDate(Day As String) As Date
' The function takes a string representing a day and returns the corresponding date, within the next seven days.
' Declaration section
Dim Today, CourseDay As Integer
' Code starts here
  Today = Weekday(Date(), vbMonday) 'A number is associated to the actual date, Monday = 1
  ' The number of days from today to the day of the class are computed and used to calculate the class's date
  CourseDay = ConvertDay(Day) 'A user defined function that operates as Weekday.
  If CourseDay > Today Then ' Courseday is this week
      ClDate = DateAdd("d", CourseDay - Today, Date())
  Else ' CourseDay is Next week
      ClDate= DateAdd("d", 7 - Today + CourseDay, Date())
  End If
End Function
```

Briefly, *ClDate* takes a string representing a day, let say Friday, and returns the date of the next Friday within the next seven days. For instance if today is Monday January 1st 2016 and Day = Wednesday, the function will return January 3th 2016.

At first, the current date (i.e., Date()) is converted in a number with the WeekDay function; next the day of the class is converted in a number with ConvertDay function. For instance, if Date() = #17/11/2016#, then the actual day is Thursday and so WeekDay returns 4 (Thursday is the fourth day of a week). Similarly, if the day of the course is Saturday then ConvertDay returns 6.

After the days have been converted into numbers, the number of days from the current date to the date of the class is calculated. For instance, if today is Thursday and the class in on Saturday then a total of (6 - 4) = 2 days must pass. Conversely, if today is Thursday and the class in on Monday, then the class will take place on the following weeks; consequently, a total of [(7 - 4) + 1] = 4 days must pass.

Specifically:

- The quantity (7 - 4) corresponds to the number of residual days of the current week;
- The value 1 corresponds to the number of days from the end of the current week to the next Monday.

This logic is performed inside the IF … THEN … statement of the code.

Lastly, once we know the number of days (*nd*) that have to pass, we can easily compute the date of the class; this can be done adding a total of *nd* days to the current date. To this aim the DateAdd function can be used.

Please, also note that:

- The VBA *Date()* functions does not require an input and simply returns the current date;
- The VBA *DateAdd()* function adds a time interval to a supplied date and/or time, and returns the resulting date/time. The syntax of the DateAdd function is: *Dateadd(Interval, Number, Date)*;
- The VBA Weekday function returns an integer (from 1 to 7), representing the day of the week for a supplied date. The syntax of the function is: *Weekday(Date, [FirstDayOfWeek]);*
- *ConvertDay* is a user defined function shown below. It operates as Weekday, as it takes a string representing a day (such as “Friday”) and convert it in a natural number starting from Monday = 1. The difference is that, Weekday operates on a Date, instead ConvertDay operates on a string.

```
Private Function ConvertDay(Day As String) As Integer
' This function is similar to Weekday: it takes a string representing a day (such as “Friday”) and converts it in a
' natural number starting from Monday = 1. This is done using a basic Select Case statement
  Select Case Day
    Case "Monday"
        ConvertDay = 1
    Case "Tuesday"
        ConvertDay = 2
    Case "Wednesday"
        ConvertDay = 3
    Case "Thursday"
        ConvertDay = 4
    Case "Friday"
```

```
            ConvertDay = 5
        Case "Saturday"
            ConvertDay = 6
        Case "Sunday"
            ConvertDay = 7
    End Select
End Function
```

```
Public Function TotReg(IdCourse As Integer, CDay As String, CTime As Date) As Integer
' This function calculates the total number of people that made a reservation for a class.
' Declaration Section
Dim CDate As Date
Dim Criteria As String
' Code starts here
    CDate = ClDate(CDay) ' It converts the day (CDay) of the class in a Date
    ' The filtering criteria, used in the DFunction operating on the RESERVATIONS table, is obtained by combining
    ' many substrings with the & operator
    Criteria = "[Class Date] = " & Format(CDate, "\#mm\/dd\/yyyy\#") ' This is the American format
    Criteria = Criteria & " AND [Class Time] = #" & Format(Ctime, ShortTime) & "#"
    Criteria = Criteria & " AND ID_Course = " & IdCorso
    Criteria = Criteria & " AND [Registration's State] = 1"
    ' Use a Domain function to sum all the valid registration to a class
     TotReg = Nz(DSum("[Registration's State]", "RESERVATIONS", Criteria)) ' State = 1 if registration is valid
  End Function
```

Briefly, TotReg takes three inputs (i.e., an Integer, a String and a Date/Time) and returns the total number of customers that registered to a specific class.

Specifically, the integer is the ID of the course (IdCourse), the string is the day of the class (CDay) and date/time is the time of the class (CTime). These values are used to create a filtering criteria used in a Domain Function. Let us suppose that IdCourse = 1, CDay = Monday, that Time is 12:00 and that today is Thursday #17/11/2016#.

At first, Monday is converted into #21/11/2016#, next the following filtering condition is obtained by combining three sub strings:

```
Criteria = "[Class Date] = #11/21/2106# AND [Class Time] = #12:00# AND ID_Course = 1 AND _ [Registration's State] = 1"
```

Please note that the date is converted in the American Format[5] (with months preceding the days) and that, in SQL, both a date and a time must be placed inside two hashtags i.e., #date# or #time#.

This criterion is used in the following Domain Function:

```
Bookings = Nz(DSum("[Registration's State]", "RESERVATIONS", Criteria))
```

In this way only the records of the selected class (identified through ID, Date and Time), which have a non-null value of the Registration's State field, are summed and the total number of bookings is returned.

Please note that *Format* is a VBA function that makes it possible to format in a standard way numbers, date, time and even string. In the above example we used the following expression to convert a date in the standard format and to place it inside two hashtags:

```
Format(CDate, "\#mm\/dd\/yyyy\#")
```

The desired format is passed as a string (inside quotation marks), also \ is a jolly character that indicates that the characters that comes next must be included in the desired format. So, we are saying that we want the month followed by the day and by the years; and we are also saying that month, day and year must be separated by / and that the date must be preceded and followed by #.

A similar function, namely MaxAllowed, is used to compute the maximum number of people that can join a class. The code is the following one:

```
Public Function MaxAllowed(IdCourse As Integer, CDay As String, CTime As Date) As Integer
' This function calculates the maximum number of people that can join a class
Dim CDate As Date
Dim Criteria As String
Criteria = "ID = " & IdCourse
   ' We use a Domain function to get the maximum number of people that can join a class
   MaxAllowed = Nz(DLookup("[Max Number]", "COURSES", Criteria))
End Function
```

These basic functions are used to compute also the number of vacancies (number of free places) of a course and/or the people that have been placed in the waiting list.

[5] *To avoid possible errors when comparing dates, it is better to use the American format*

```
Public Function FrPlaces(IdCourse As Integer, CDay As String, CTime As Date) As Integer
Dim Bookings, MaxAl As Integer
   Bookings = TotReg(IdCourse, CDay, CTime)
   MaxAL = MaxAllowed(IdCourse, CDay, CTime)
   If Bookings <= MaxAl Then
        FRPlaces = MaxAl - Bookings
   Else
        FRPlaces = 0
   End If
End Function
```

```
Public Function InQueue(IdCourse As Integer, CDay As String, CTime As Date) As Integer
Dim CData As Date
Dim Criterio As String
   CData = DataCorso(GIORNO)
   Criteria = "[Class Date] = " & Format(CDate, "\#mm\/dd\/yyyy\#") ' This is the American format
   Criteria = Criteria & " AND [Class Time] = #" & Format(Ctime, ShortTime) & "#"
   Criteria = Criteria & " AND ID_Course = " & IdCorso
   Criteria = Criteria & " AND [Registration's State] <> 1"
   If FrPlaces(IdCourse, CDay, CTime) = 0 Then
      InQueue = Nz(DCount("[Registration's State]", "RESERVATIONS", Criteria))
   Else
      InQueue = 0
   End If
End Function
```

4.2 Queries used to make statistics

Average Time Inside the Center (Tempo Medio In) - This query returns the average time spent by a customer inside the center. It is a query based on a subquery and on a calculated field that takes advantage of the *DateDiff* function. DateDiff Function returns a Variant specifying the number of time intervals between two specified dates. *Its syntax is as follows: DateDiff(interval, date1, date2[, first day of week[, first week of year]])*

The SQL is the following one:

```
SELECT Format(Avg(Minutes),"Standard") AS [Average Minutes In]
FROM (SELECT Datediff('n', [Time In], [Time Out]) AS Minutes FROM ACCESSES)
```

As it can be seen, the inner query is a column query that returns the difference in minutes ('n' indicates minutes) between all the entrances and exits of the customer. The outer query operates on this column, taking the average.

Sum of Memberships (Somma Abbonamenti) - This is a parametric query (i.e., some inputs are variables that must be specified by the user when the query is executed) that counts the total number of valid memberships. The summation may be done filtering for membership's type, for date, etc. The SQL is shown below:

```
SELECT Count([CUSTOMERS].[ID]) AS Total
FROM MEMBERSHIPS INNER JOIN (DURATIONS INNER JOIN ([TIME WINDOWS] INNER JOIN _
CUSTOMERS ON [TIME WINDOWS].ID = CUSTOMERS.[ID_TimeWindow]) ON _
DURATIONS.ID = USERS.[ID_Duration]) ON MEMBERSHIPS.ID = CUSTOMERS.[ID_Membership]
HAVING MEMBERSHIPS.Name=[Insert the name of the membership] AND_
DURATIONS.Name = [Insert a Duration] AND [TIME WINDOWS].Name = [Insert a time window]
```

Average number of participants to a course (Partecipanti medi ad un corso) - This is a parametric query that returns the average number of customers that join a course. The SQL is as follows:

```
SELECT QPARTICIPANS.Name AS Course, Format(Avg(QPARTICIPANS.Participans), "Standard") AS [Avg N]
FROM QPARTICIPANS
WHERE QPARTICIPANS.[Class Date] >= [Insert Filtering Date]
GROUP BY QPARTICIPANS.Name
```

As it can be seen, the query operates on a saved query called QPartecipants, whose SQL code is shown below:

Query QPARTICIPANTS:

```
SELECT COURSES.Name, REGISTRATIONS.[Class Date], Count(RESERVATION.ID_Course) AS Participants
FROM COURSES INNER JOIN REGISTRATIONS ON COURSES.ID = RESERVATIONS.ID_Course
GROUP BY COURSES.Name, REGISTRATIONS.[Class Date]
```

QPARTICIPANTS returns a table that, for each day of the year (where at least a course was scheduled), shows the total number of people that participated to a course.

Sum Of Membership operates on these data and takes the average of the participants to each courses starting from a date passed as input by the user.

5. Code Insights and Snippets[6]

5.1. Variables

When you create a table in Access, you need to define the data type of every field of the table. Data types in tables include Text (for storing short strings of text), Memo (larger chunks of text), Number (for numbers), and Date/Time (for dates and times).

Clearly, VBA can work with data stored in tables, too. But just like tables, VBA often needs to know the type of information it is working with. Thus, VBA has a way of storing little chunks of information (data) in temporary cubbyholes called variables. Although it is not compulsory, before using a variable it is wise to declare it, using the more appropriate data type among the ones handled by VBA. These are:

- *Boolean:* used to hold a value that can be either true or false;
- *Byte: used to store integer values in the range* [0 – 255];
- *Currency (@)*: a real number formatted as currency with the euro or dollar symbol;
- *Date: used to store dates;*
- *Integer (%)*: as for byte, integers are used to store whole number, in a wider range [-32768: 32767];
- *Long (&)*: used to store whole number in an extremely wide range (i.e. coded using 8 byte);
- *Double (#)*: doubles are more accurate than integers and can also store numbers after the comma;
- *Single (!):* similar to a double, but the representation is less precise (i.e. single-precision numbers store an approximation of a real number);
- *String ($):* a variable that contains text;
- *Variant:* a variable that can contain any kind of data;
- *Object:* a specific variable used to store every kind of objects (a Data Base, a Form, etc.)

Note that the symbol in parenthesis is a short cut to define the corresponding data type.

Creating a variable is a two-step process: *(i)* first you declare the variable's name and data type with a *Dim…As* statement and next *(ii)* you assign a value to the variable as needed.

The syntax usually involves two lines of code that follow this structure:

```
Dim name As type
name = value
```

where name is a name of your own choosing, type is one of the data types mentioned above and value is the data you want to store in the variable.

[6] A full list of all the predefined functions available in ACCESS can be found at https://support.office.com/en-us/article/Access-Functions-by-category-B8B136C3-2716-4D39-94A2-658CE330ED83

For instance:

```
Dim x As Integer
x = 10
```

5.2 Loops

Occasionally, a situation occurs in which you want to execute one or more VBA statements multiple times. Suppose you write some VBA statements that need to operate on each record in a table, and the table holds 1000 records. You have two choices: (i) write each set of statements 1000 times or (ii) create a loop that repeats the same set of statements 1000 times. Needless to say, typing the statements once rather than 1000 times saves you a lot of time. A loop is your best bet.

The *Do...Loop* block is one method of setting up a loop in code to execute statements repeatedly.

The loop requires two lines of code: one at the top and one at the bottom. You have a lot of flexibility when defining a *Do...Loop*. In fact, there are two forms of syntax for creating these loops.

The first is the following one:

```
Do [{While | Until} condition]
    [statements]
[Exit Do]
    [statements]
Loop
```

The second form of syntax provides the option of defining the condition at the bottom of the loop, like this:

```
Do [statements]
    [Exit Do]
    [statements]
Loop [{While | Until} condition]
```

In both instances, *statements* refer to any number of VBA instructions, while *condition* is an expression that can result in either True or False. The code inside the *Do … Loop* statement is executed repeatedly (from top to bottom), depending on the value of the condition:

- If the *While* clause is used, then the Do … Loop cycle ends as soon as the condition becomes false.
- Vice versa, when *Until* is used, the cycle terminates when the condition becomes true.

Also note that in the first case, depending on the value of the condition, the code inside the Do … Loop cycle may be not executed. Conversely, in the second case, it is executed at least once.

Lastly, the *Exit Do* statement makes it possible to exit from the cycle before the exit condition becomes true.

The *While…Wend* statement is similar to *Do...Loop*, but it uses the simpler (and less flexible) syntax shown in the following code:

```
While condition
    [statements]
Wend
```

As can be seen this loop It has the following parts:

- *Condition* (Required.) → A numeric or string expression that evaluates to True or False. If condition is Null, condition is treated as false
- *Statements* (Optional) → One or more statements executed while condition is True

If condition is True, all statements are executed until the Wend statement is encountered. Control then returns to the While statement and condition is again checked. If condition is still True, the process is repeated. If it is not True, execution resumes with the statement following the Wend statement.

While...Wend loops can be nested to any level. Each Wend matches the most recent While.

A simple example follows. It uses the *While...Wend* statement to increment a counter variable. The statements in the loop are executed as long as the condition evaluates to True.

```
Dim Counter As Integer
Counter = 0 ' We initialize a variable
While Counter < 20 ' We test value of the Counter variable
    Counter = Counter + 1 ' We Increment Counter
Wend ' We End the While loop when Counter > 19.
Debug.Print Counter ' We Print 20 in the Immediate window.
```

Another set of commands for creating loops in code is the *For...Next* block of statements.

The syntax for a *For...Next* loop is shown below:

```
For counter = start To end [Step step]
    [statements]
[Exit For]
    [statements]
Next [counter]
```

where:

- *counter* is any name that you want to give to the variable that keeps track of passes through the loop;
- *start* is a number that indicates where the loop should start counting;
- *end* is a number that indicates when the loop should end;

- *step* is an optional parameter and it indicates how much to increment or decrement the counter variable any time a single loop has been completed. If it is omitted, counter is incremented by 1, with each pass through the loop;
- *statements* are any number of VBA statements that execute with each pass through the loop.

5.3 Arrays

If you think of a variable or constant as one little cubbyhole in which you can tuck away information, a collection of cubbyholes is an array. Each cubbyhole is an element of the array, in which you can store information. All the cubbyholes in an array have the same name (i.e., the name given to the array). In order to specify on which cubbyhole you want to operate to, you need to use a subscript in parentheses, which defines its specific position in the array.

Declaring an array is a lot like declaring a single variable, but you have to tell VBA how many items are in the array. The syntax looks like this:

```
Dim ArrayName(dimensions) As Type
```

where *ArrayName* is a name that you give the array, *dimensions* is a parameter that specifies how many items there are in the array, and *type* is one of the data types available in VBA.

For example, the following statement create an array of names and assign the name “Bill” to the first element (note that, by default, the first element has the index 0):

```
Dim Names(5) As String
Name(0) = “Francesco”
```

The following example creates a bi-dimensional 5×5 array (i.e., a matrix) containing Integers numbers:

```
Dim Matrix(5, 5) As Integer
```

There are several useful functions that can be used to manipulate arrays. The most useful ones are:

- *Length* - it represents the total number of elements currently contained in the array/matrix, or the total number of elements contained in a specific dimension of a matrix;
- *Ubound* - it returns the last index of an array or the last index of one of the dimensions of a matrix.

It is also possible to create *dynamic arrays*, that is you just say to VBA that a variable is an array, but you do not specify its dimension. To this aim you just do not place any number inside the parenthesis, as in the following example:

```
Dim DynamicArray() As Integer
```

Next, before assigning values to the element of the array you have to define its dimensions (at run time), using the keywords Redim:

```
Redim DynamicArray(5)
```

In this way you have created an array that can contain six integer values. You can use Redim as many time you want to change the dimension of your array. However in doing so you lose the values that you've stored in the array. If you want to save yours values you need to use the *keyword Preserve* immediately after *Redim*:

```
Redim Preserve DynamicArray(Ubound(DynamicArray) + 2)
```

Loops are very useful to work with arrays. Suppose you have just created a 4×4 matrix ***M*** and you want the value of each element being equal to the sum of its row and columns, that is ***M****(r,c) = r + c* for each r and for each c. How could you do so? A simple code follows:

```
Dim M (1 To 4, 1 To 4) As Integer
Dim r, c As Integer
  For r = 1 To 4
     For c= 1 To 4
          M(r,c) = (r + c)
     Next c
  Next r
```

5.3 Making Decisions

Decision-making is a big part of programming because most programs need. Often, you want your code to do one thing if such-and-such is true but do something else if such-and-such is false. You use conditional expressions to determine whether something is true or false.

You have a couple of ways to write VBA code that's capable of making a decision; the simplest and by far the most common is the *If...End* If block of code, which uses this syntax:

```
If condition Then
     [statements]...
[Else]
     [statements]...
End If
```

Where: condition is an expression that results in True or False (or a Boolean variable), and statements refers to any number of valid VBA statements. If the condition proves True, the statements between *Then* and *Else* execute, and all other statements are ignored. If the condition proves *False*, only the statements after the Else statement execute. *If…End* blocks can be nested and you can also have a little bit of flexibility when using *If...End If*. If only one line of code executes for a True result and only one line executes for a False result, you can put the whole statement on a single line and omit the End If statement, as this line shows:

```
If State = "NY" Then TaxRate=0.075 Else TaxRate=0
```

In some situations, you might need to have your code make a decision based on several possibilities. For example you may need to perform different statements depending on which product type a person has ordered. In that case, you can set up a *Select Case* block of code, which performs a particular set of instructions depending on some value. Typically, the value is stored in a variable or field in a table and is also a number that represents some previously made selection.

The basic syntax of a Select Case block of code looks like this:

```
Select Case value
    [Case possible Value [To possible Value]
        [statements]]
    [Case possible Value [To possible Value]
        [statements]]...
    [Case Else]
        [statements]]
End Select
```

Where: *value* is some value (like a number), and *possible Value* is any value that could match the value. You can have any number of Case possible Value statements between the Select Case and End Select statements. Optionally, you can include a *Case Else* statement, which specifies statements that execute only if none of the preceding Case Value statements proves True.

Each Case statement can have any number of statements beneath it. When the code executes, only those statements after the Case statement that matches the value at the top of the block execute.

5.4 Procedures: functions and subroutines

In VBA a set of commands to perform a specific task is placed into a procedure, which can be a *function* or a *subroutine*. The main difference between a VBA function and a VBA subroutine is that a function returns a result, whereas a subroutine does not.

If you wish to perform a task that returns a result (e.g. summing of a group of numbers), you will generally use a function, but if you just need a set of actions to be carried out (e.g. formatting a set of cells), you might choose to use a subroutine.

In other words, we can say that a function is a junk of code that receives some input values and returns an output as result. Any programming language has its own (predefined) functions. For example, the Date() function does not require any input and returns the current date. Similarly the Abs(x) function requires x as input and returns its absolute value. You can immediately check this issue by typing *? Date()*, or ? Abs(-10) in the *VBA Editor*

Immediate window[7]; when you press enter the Immediate window will show the current date or 10, respectively. In this respect, it is useful to know that in VBA for Access, all the functions that are available in Access are also available in VBA (the *Expression Builder* is a good tool for finding out which functions are available as well as how to use them).

The basic syntax of a function is as follows:

```
Public|Private Function Function_Name (Input1 As Type, …, InputN As Type) As Type
    [Statements]
[Exit Function]
    [Statements]
End Function
```

Where: *Public* or *Private* define the visibility of the functions (if public you can call it from any modules of your code), *Input1* to *InputN* are the values that must be passed to the function in order to execute.

All the statements included in between the function's name and the End Function statement are executed from top to bottom (these statements can contain a variable's declaration section, loops, if conditions etc.). Also, if a certain condition applies the function can be halted using the *Exit Function* statement.

Another important point to stress is the following one: since a function always returns a value, the data type of the returned value must be declared in the function header. The value to be returned must be assigned to a variable having the same name of the function. This variable does not need to be declared, as it already exists as a part of the function. The value returned from the function is the last value to be assigned to this variable before the function is exited.

This concept may be better explained with a simple function that accepts three real numbers and returns a real number, being the sum of the first two, minus the third one:

```
Private Function SumMinus(dNum1 As Double, dNum2 As Double, dNum3 As Double) As Double
        SumMinus = dNum1 + dNum2 - dNum3
End Function
```

[7] *The immediate window is available (for debug purposes in the VBA development environment). It allows you yo execute and test part of code. The symbol ? is used to ask to VBA to perform some operations*

De facto, subroutines are a lot similar to functions. As mentioned above, the main difference is that they perform some tasks, but they do not return a value. Thus the syntax simplify as follows:

```
Public|Private Sub Function_Name (Input1 As Type … InputN As Type)
    [Statements]
[Exit Sub]
    [Statements]
End Sub
```

As we said, both functions and subroutines can be passed data via arguments, which are declared in the VBA function or subroutine definition. Some arguments may be *Optional*. These arguments may be supplied or not; in either case the procedure will still work. To this aim you just need to precede the name of the arguments with the *Optional* key words. An example follows

```
Private Function SumMinus2(Num1 As Double, Num2 As Double, Optional Num3 As Double) As Double
        If IsMissing(Num3) then Num 3 = 0
        SumMinus2 = dNum1 + dNum2 - dNum3
End Function
```

In this case the third input may not be provided. If so it is set to zero; note that we have introduced an If…End statement to check whether the third input has been supplied. We did so using the VBA functions IsMissing() that returns a Boolean value and so can be used as condition in the If…End Statement.

Of course VBA allows you to use multiple Optional arguments in a procedure, as long as the Optional arguments are positioned at the end of the argument list.

Another important thing (especially if you are using a subroutine) is the possibility to pass arguments By Value or By Reference.

- *ByVal* - The argument is passed by Value. This means that just the value (i.e., a copy of the argument) is passed to the procedure and therefore, any changes that are made to the argument inside the procedure will be lost when the procedure is exited.
- *ByRef*- The argument is passed by Reference. This means that the actual address of the argument is passed to the procedure. Any changes that are made to the argument inside the procedure will be remembered when the procedure is exited.

You can specify whether an argument is passed to a VBA function by value or by reference by using the ByVal or the ByRef keyword just before the argument's name.

It should be noted that, by default, in VBA, arguments are passed by Reference, so if you do not use the ByVal or the ByRef keyword, the arguments will be passed by Reference.

Lastly we observe that a variable number of parameter can be passed to a function. In this case you need to precede the name of the argument passed to the function with the *ParamArray* keyword.

An example follows:

```
Sub ParamArrayExample(ParamArray varParameterArray() As Variant
   ' ParamArray must be the last defined parameter
   Dim lngCount As Long
         'On the immediate windows the index, the value and the type of the parameter will be displayed
         For lngCount = 0 To UBound(varParameterArray)
               Debug.Print lngCount, varParameterArray(lngCount), VarType(varParameterArray(lngCount))
         Next
End Sub
```

We conclude this section with an useful function, written in VBA for Excel, that takes a range as input and computes the moving average of order *N* of the data contained in the first column of the range.

```
Public Function MAVG(Values As Range, Optional Order As Integer = 2) As Variant
' This function is an array formula, i.e., a function that returns a vector.
' To this aim it must be declared as Variant
Dim Col As Range ' This will be our column
Dim R, Rtot, Rin As Integer ' This are reference variables
Dim i As Integer ' This is a looping variable
Dim MA() As Varian  ' The vector were we will store the moving average
' Set is used to assign a value to an object. We are assigning the first column of the input range
  Set Col = Values.Columns(1)
  Rtot = Col.Rows.Count ' Total number of rows of the column
  ReDim MA(1 To Rtot) ' The vector is correctly dimensioned
' All elements are set to zeor
    For i = 1 To Rtor
          MA(i) = 0
    Next i
' The order of the Moving Average must be a natural value and must be lower than the number of values
  Order = Application.WorksheetFunction.Floor(Order, 1) ' We use an excel function to take the integer part
    If Rtot >= Order And Order > 0 Then
        ' Two nested loops are used to compute and save the moving average
```

```
            For R = Order To Rtot
                For Rin = R - (Order - 1) To R
                    MA(R) = MA(R) + Col.Cells(Rin, 1)
                Next Rin
                MA(R) = MA(R) / Order
            Next R
        End If
' MA is a row vector, to have a column we need to take the transpose
  MAVG = Application.WorksheetFunction.Transpose(MA)
End Function
```