Part II: Let's get real

The following example is of a general enough nature as to provide cases that you are bound to find in applications that are of a medium level of complexity, in order to help you translate these ideas into your own projects

1. Case Example: Now we need a problem...

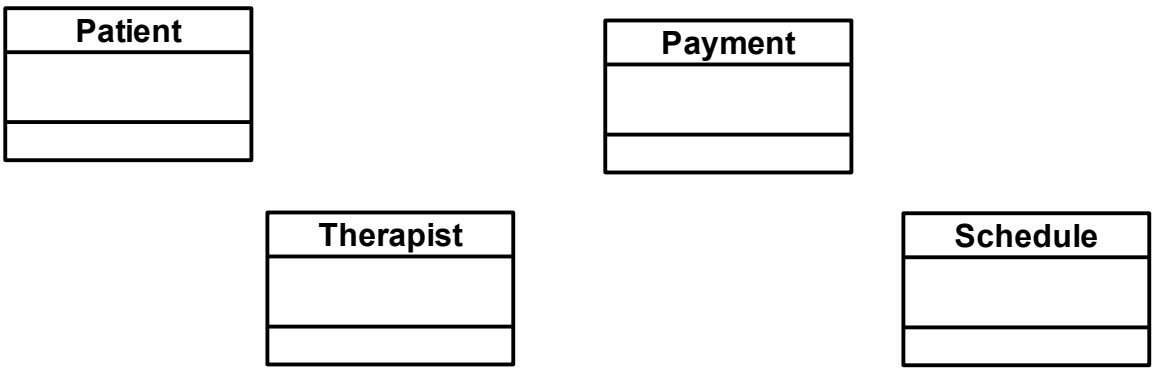
Let's apply the phases of database design to a concrete and possible problem. We will consider the needs of a small psychotherapy center. The director will be the client and you and I will be the experts.

Our first step as experts will be to try and identify the presenting problem. Because this is a stylized version of the work, the most important questions -and answers- will be identified quite quickly. In real life, this can easily take several meetings or a good chunk of time devoted to thinking about the operations or business processes we will be aiding.

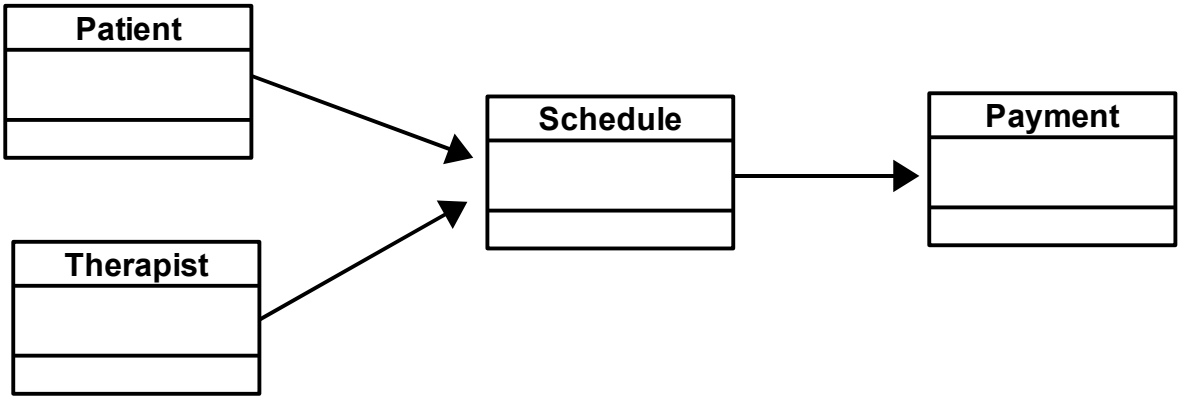
In our example we start by asking the client and learning what is the business about and who are the principal players. The clinic receives people looking for psychotherapy services. They are usually referred by their medical doctor, but not always. Once evaluated and admitted, they are assigned to a therapists according to relevant training and time availability. Then they show up for a number of sessions they, or their insurances, need to pay for. The director will then pay the therapists according to the number of patients they saw in a month.

Now that we have a broad understanding of the process, we want to know what the needs of our client are: the client needs to record data from his patients and decide if they should be admitted to the Clinic. He needs to organize the admitted patients, matching them to a therapists and finding an available time slot. He needs to keep track of the performed sessions, by therapists so he can pay them, and by patient so he can charge them. He also needs to keep info about the therapists, their training and other data for tax purposes.

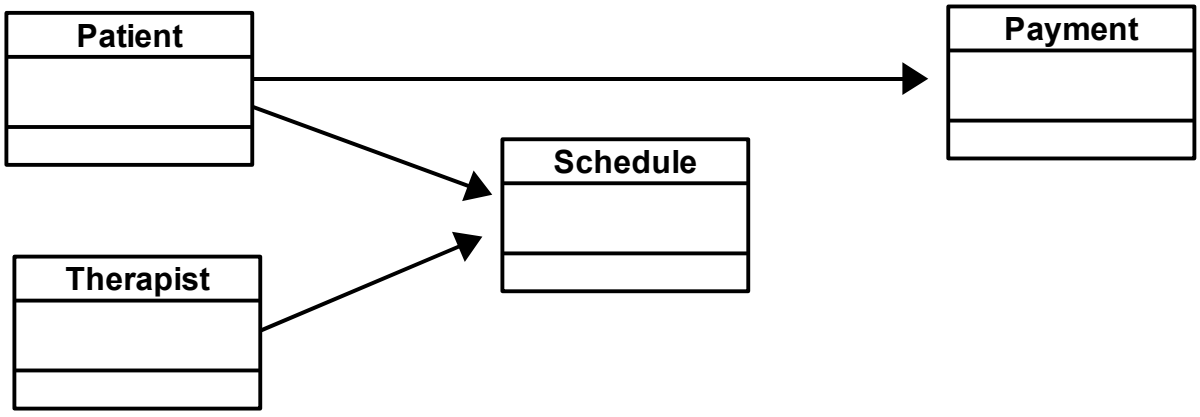
So we need to record info about the clients, the therapists, about their meetings and about the payments. After our first (and very stylized and lean) approach we can start to identify some classes we are going to need: The Patient class, the Therapists class, the Schedule class and the Payment class.




We can imagine that a patient and a therapists will be assigned to a schedule and that a payment will be associated with a performed session, as recorded in the schedule. So in very general terms we can imagine a structure of connections more or less like the following:




Now, here comes the first problem with assumptions: they can be wrong! So it is always better to ask about what we are thinking instead of taking it for granted: When we show this diagram to the Client, he tells us that, although our first assumption -that patients and therapists will be assigned to a schedule- is correct, our second assumption -that payments are associated to a performed session- is not. First, patients will have to pay for a session they didn't show up for if they do not cancel it at least 24 hours prior, so payment is not necessarily associated with performed sessions. Second, patients might request a payment plan, allowing them to make -say- weekly payments of a fraction of the cost of the session. This way, they will make small payments until the debt is complete even if they are no longer receiving therapy. All right, good we asked! Now we know that the schedule will have to record if the session took place or not and if it was properly canceled, in order to keep a tally of money owed; and that the payments will only be associated to the client and not to the Schedule. We can modify our model to reflect this in the following way:




It might seem like a small modification but, at this very early stage in the process, this change has taken our design in a different direction.

We go back to our client for more information: We heard that some patients are referred by their medical doctors. Do we need to collect this information? The clients says yes; furthermore, every patient needs to have one medical doctor in record. Are there other professionals we need to register for every patient? The client informs us that some patients might also need services by a psychiatrist and

that this data, along with the medication they are assigned, needs to be collected. Do all patients need a psychiatrists? The client informs us that no, only some patients could needed it.

We know by now that he needs to collect info about the potential patient but that he also needs to perform an evaluation of several factors and later make a decision to admit him/her or not. Now we can see that there is contact info, evaluation info and resolution info. The dialog between Client and Expert continues: At what moment is the patient registered into the database? When he calls seeking services or when it is resolved that he be admitted? Will all the information requested be available at the time that the record is entered? If not, what information is essential to decide to register a record? (remember the NOT NULL attribute of columns?)

By asking the former, we the experts learn that the patients will be registered into the database only after they have been admitted, which means that the evaluation has already been performed and a diagnosis been made; this way there is no need to collect all the information generated by the evaluation or the resolution. Being registered in the database is a very important milestone as it marks the moment when the patient is incorporated to the clinic and becomes a responsibility of it. For this reason, it is also important to record the date at which the patient is registered into the database. However, at this point in time, there could still be some information that has not been properly collected from the patients. For example, some could be taking medication and, although know the name of it, not be sure of the dosage. Others could have forgotten the phone number of their medical doctor and promise to phone later with it. This shows us that some information is essential while other is not... The Client gives us a list of the information he wants to collect about the patients but also informs us that the name, address, phone number and the diagnosis -proof of the evaluation- would be enough to make an entry.

Let's continue the dialog between the Client and the Expert: What data do we need from the therapists? Of course, name and contact information, their degrees and specialties and tax information so they can be properly paid. And what data is relevant to collect in the payments class?. The Client wants to know who made the payment and how much he paid. At this point we can use our imagination and identify data that the Client has not though of but that sounds important to us: For example, would the Client like to record the date when the payment was done? Yes, he answers. Would he like to record the date a therapists is hired or stops working at the clinic? At first the Client seems unsure about this info, that it would not be commonly used, but later realizes that it would do proper administration and decides to include that data.

What about the schedule class? The Client states that he will want to record the time slots the patients are assigned and the therapists they are assigned to, when was the patient assigned and the date the case is closed. He also wants to record whether the patients came or not to a scheduled session and, if not, if they properly canceled the session. We, the experts, ask him what happens if it is the therapists who has to cancel a session. In that case, he answers, the patient should not be billed for that session and therefore, who made the cancellation should also be recorded.

Reviewing the paper forms the clinic has been using so far we discover other elements of interest: some patients have a home phone number and live happily with that. Other patients have a home number, a cell or mobile number, a job number and maybe even a second job phone number. We ask why and discover that being able to get in touch with the patients is very important for the clinic, particularly if an emergency arises. This also leads us to the possibility that the Client could need extra contact information to reach the medical doctor and the psychiatrist in case of an emergency. We ask the Client about this but he says that the staff of those professionals will know how to reach them if needed, so we

only need to record their office numbers.

What reports does the Client need to make? At first, the Client can think of three principal ones: clinical reports, financial reports and schedule reports. The clinical reports should contain contact information about the patients, diagnosis, assigned therapist and medical doctor, and psychiatrist and medication info if any. Financial reports need to provide information in three different ways: The overall number of billable sessions performed in a set period of time, patient information that includes the total number of sessions performed and the total amount of money payed by him and, lastly, the assigned sessions to be payed to every therapists. The schedule report should indicate what therapist has been assigned to what patient and in what time slot. After thinking about this for a while, the Client also tells us that he would like to have a summary of all the sessions to be performed next day with the name and contact data of the patients so they can be called and confirmed. Hum! Challenging. Let's see if we can accommodate these requests!

Now, just in case this has been going too fast, let's make a summary of what has happened so far. A client, the director of a mental health clinic, has approached us, the experts, to develop a database to organize his clients, therapists and to keep track of payments. We, the experts, have inquired about the way this clinic handles business, pre-visualized the reports required by the client and analyzed the paper forms he currently uses. We also stressed our own imaginations and offered some ideas, all of which helps in fleshing out the classes and attributes we could need. With this work we have identified four main classes: clients, therapists, schedules and payments.

So far, so good.

2. Possible solution.

Now that we have relevant information, we should sit down and write a formal statement about what the goals or purpose of our database will be. By this we mean the functionality that, we as experts, believe that is possible to provide for the situation and needs revealed in the previous research.

So after thinking about it for a while we write down that we will develop a database that stores contact and clinical information about patients, that stores and maintains information about services provided, including dates of sessions and assigned therapist, that stores and maintains information about payments received by them and that stores information about therapists and their qualifications. The database will be able to provide the following reports: Clinical summary of patients, summary of services rendered to individual patients, payments made by individual patients, sessions performed by individual therapists, summary of all services rendered by the clinic for a set period of time and a log of next-day sessions with patient contact info.

You can see that this statement is quite short and representative of the needs of the client and the information that needs to be collected. It provides concise goals and boils down the needs of the client into concrete procedures and final reports. It also sets a standard of completion: When our Base application can do what is stated in the previous paragraph then this project has been completed (and a new one can be started). Professional developers usually present this to their Clients and make them sign a copy. When clients later complain that the application does not, for example, provide a summary of the most common mental health illnesses, they answer: “Well, that was not in the initial specification. If you want that we can develop it for you and it will cost you $$$ extra”. Pure genius!

The other important thing is that this statement will help us make decisions about data modeling. As

you might know by now, there is more than one way to organize the same data. If one of those options makes producing one of the required reports difficult or complicates the collection of information then that is an option we will surely drop in favor of an option that makes the tasks easier. This should become self evident the more you practice your data modeling skills.

With the elements collected so far we can start drafting the forms we might need. For patients we will want to record: name, date of birth, address, one or more phone numbers, diagnosis, medical doctor, psychiatrist and medication -if present- and the date of registry. For therapists, we will want to record name, address, phone numbers, tax id number, degree, specialization and date hired. For the Payment class we will record the patient, the amount and the date of the payment. For the schedule class we will collect the patient, the therapists, the assignment date, the time slot and whether the session took place or not.

We are now ready to start modeling data. However, and to be truly honest, we have not cleared all possible questions relating our data. As we try to conform the classes, their attributes and connections, new questions will pop up that will force us to go back to the client, rethink our designs, include new data or even exclude some of the data we now find necessary. Let's not fear this and consider it part of the job. However,our goal statement provides a direction and, whatever changes we do, they must all aid in achieving the stated goals.

3. Data Modeling.

Intuitively, the Client class and the Schedule class seem the most complex. Let's star by analyzing and fleshing them out.

In our initial draft, the Client class was connected to the Assignment and the Payment classes. This means that we will need a primary key. We will let Base produce a numeric key for each record. With the information collected so far, we can propose the following attributes for this class:

| **Patient** |
| --- |
| ID Number (Primary key)<br>Name<br>Date of Birth<br>Address<br>Phone number1, number2, number3, etc.<br>Diagnosis<br>Medical Doctor Name, address, phone number<br>Psychiatrist Name, address, phone number<br>Medication<br>Time of registry |
| |

Any problems with this? Several.

Maybe you have already spotted the multivalued information for phone numbers. This is because this table is not in first normal form. Normalization here requires us to take the phone numbers and produce a new table with the multivalued item and the primary key as a foreign key. If we didn't do this, we would have an arbitrary number of columns (Phone1, Phone2, Phone3.) that sometimes are populated and oftentimes not. We would also be unable to record a fourth phone number if the client has one. All

this is solved by extracting the phone numbers and making them their own class.

Do we have problems with second normal form? No, because this class does not have a compound primary key, so there is no way a subset of it could also be a primary key.

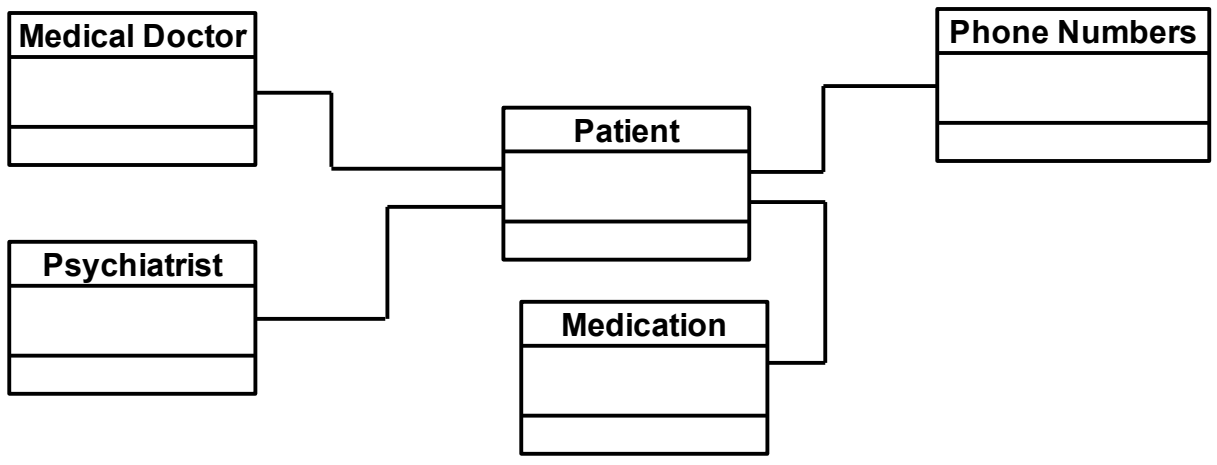
But there is something fishy about the data for the medical doctor. Notice this. I do need the primary key to know what medical doctor a particular client has. But I don't need the key to know the medical doctor's address or phone number, all I need is the medical doctor's name and I can tell you the other data. This could be better visualized if you imagine this class as a table with several records that have the same doctor. Only the information particular to each patient is unique, but the medical doctor's information would be repeated again and again and you could identify it by just knowing the medical doctor's name. Aha, we are in violation of third normal form here! We need to extract the data 'Medical Doctor' and make a new table with it.

The same thinking applies to the information about the psychiatrist. I only need the psychiatrist's name and I can tell you the psychiatrist's address or phone number. This info also requires a table of its own. If you think about it for a minute, what we have here is a set-subset situation. We have a set: patients, and a subset: patients with psychiatrists. Not all patients have psychiatrists. This also suggests that we need to extract this information and organize it in its own class. It also points in the direction of one of the cardinalities we are going to need: 1..1.

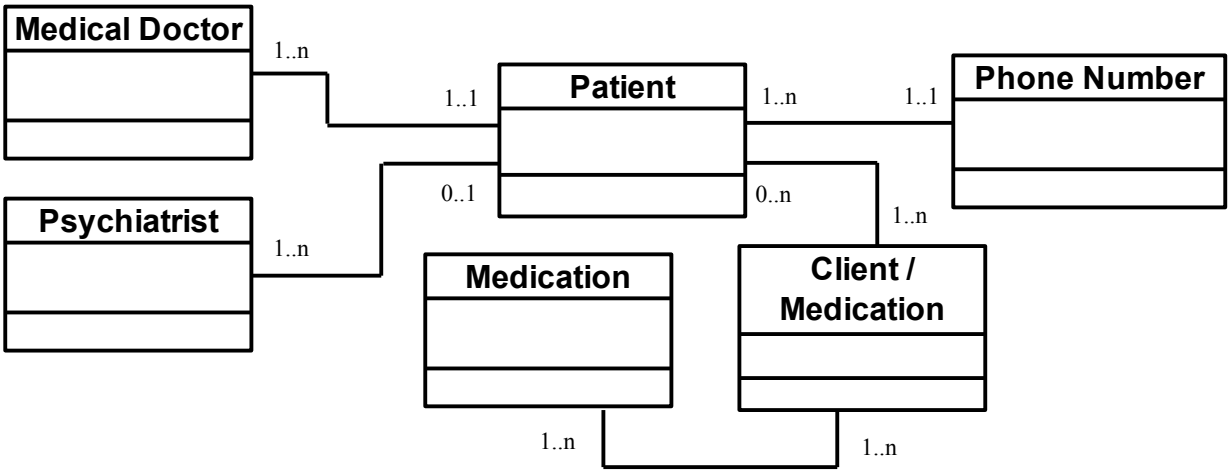
The other way to look at this situation is to recall the first principle in class formation that states that a class must be about one topic and one topic only. Our first draft for the Patient class does not follow this rule and includes other topics like medical doctors and psychiatrists.

Something similar happens with medication. Not all patients need medication. This is, again, a set-subset situation and medication needs to be made its own class. Would normalization spot that medication needs to be in its own class? Yes, but by a different route: people could need one but also two, there or more different medications, reviving the problem of multiple values for one item of information alluded by first normal form. Therefore, medication would have been made a table of its own. Both ways of thinking reach the same conclusion and are, in consequence, equivalent.

After applying normalization to the Patient class we have developed four new classes that relate to it. Using UML we can describe their relationships as follows:



Let's now think about the cardinalities: By asking our Client, and using our imagination, we can realize that one medical doctor could refer several Patients (1..n) but that each patient will have exactly one head medical doctor (1..1). Similarly, one psychiatrist could be seeing several patients from this clinic (1..n) but Patients will have either exactly one psychiatrist or none (1..0). Phone numbers are not a big mystery: one patient can have one or more phone numbers (1..n) while phone numbers can be *thought as* belonging to exactly one patient (1..1, although family members could have the same number). What is the situation with the medication? We already know that one client would be using either none, exactly one or even several medications (0..n) while several medications could be being used by several patients at the same time (n..n). Aha! n..n relationships force us to use an intermediate table between Client and Medication! We will call this table Client/Medication. Let's add all these elements to the diagram:




What seemed like a simple Patient class was decomposed to form five new classes. However, making this decomposition was not difficult at all as it only took applying the rules of Normalization and understanding well the needs of our client and the conditions under which business operations take place. Intuition also suggest that this new design is far more flexible and reliable than the first one-table draft for a Patient class

Consequently, the new Patient Class will be structured in the following way:

| Patient |
| --- |
| ID Number (Primary key)<br>Name<br>Date of Birth<br>Address<br>Diagnosis<br>Time of registry |
| |

Any problem with this? Again, yes.

To start with, the table records the name of the patient as only one attribute. If we later wanted to send mail to the patients we could only use the obviously computer-generated “Dear John Smith:” -for

example- unless we embark in complicated string manipulation. If we record the name in smaller categories, like first name and surname, we could have the option to produce different tones in our letters, like a formal: “Dear Mr. Smith:” or a more friendly “Dear John:”.

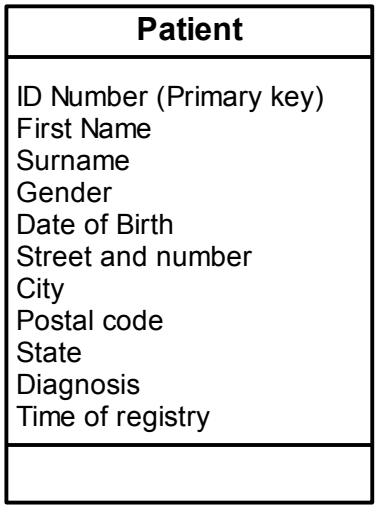
Generalizing this, it will give us more flexibility if we tend to record the smallest unity of usable information. What is the definition of usable? That will depend on the goals of your database. For example, there is no use for me to distinguish between the area code and the phone number, and I would record them both as one attribute. Yet, for someone who is analyzing patterns in phone numbers, maybe storing each individual number of the sequence in its own column would make plenty of sense.

In the same line, it is useful to me if I atomize the address information. Where I live, addresses include a street name and a number, a city, a state and a postal zone number. By storing each in its own attribute we could later, during maintenance, develop forms that produce statistics that show what cities or postal zone codes patients tend to come from, for example. Sure, this is not part of our initial goal statement (number) but that was because we were very inexperienced then and didn't know about the power and flexibility of atomizing information categories. If the client later asks us for such statistics, we will be ready to deliver and not discover with dread that our data structure does not support this.

Another helpful thing is to include gender information. I really dislike mail that reads: “Dear Sir/Madam John Smith”. This happens because the database was not able to capture the gender of the person. I might not be the only one complaining because I have seen attempts to solve this. Some forms will ask for a prefix: Mr., Mrs., Miss, etc. You might know that OO.o Writer allows you to create letter templates with conditional text that will be printed only if certain conditions are met. For this reason I would recommend to include a Boolean variable to record the gender of the patient, for example: 0=male and 1=female (this assignment is arbitrary, of course) and then use this as a condition to tailor the text. Additionally, we can later compile statistics that include gender as a factor.

Summarizing, what we have done is atomize information (e. g. name ➔ first name + surname) and added descriptors (e.g. Gender info). These same considerations will be applied to the other classes: Medical Doctor, Psychiatrists, Therapists, etc. Even a class like Medication can benefit from a descriptor, recording, for example, what do the medications do or what illnesses they treat.

With these elements, we have modified the structure of the Patient class to the following form:



Anything missing? Well, it is time to assign foreign keys.

The phone number table is quite straightforward, as we only have to follow the instructions for first normal form. This means that it will be this table that will receive the patient's primary key as a foreign key. The intermediary table “Patient/Medication”, by definition, will receive the the primary keys from the Patient table and also from the Medication table.

What about the medical doctor's table? If we made this table receive the patient's primary key, this would mean that we would be repeating the medical doctor's info for every patient that had him as a head doctor (visualize this by imagining a populated table that repeats all data for a doctor except for the foreign key that connects it to a particular patient), and this would not make any sense. In fact, it would be creating the kind of redundancy that we want to avoid because it consumes unnecessary memory and opens the door for inconsistencies. Instead, we will want the patient's table to receive, as a foreign key, the primary key of the medical doctor's table. This same logic can be applied to the psychiatrist's table.

This way we know that the medical doctor's table and the psychiatrist's table will both require primary keys and that the patient table will need to store them as foreign keys. After this analysis, we can finally complete the Patient class, which takes the following form:

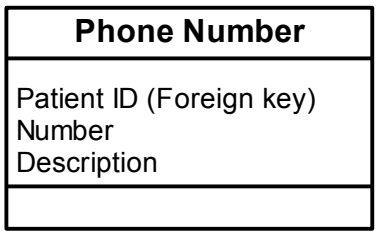
| **Patient** |
| --- |
| ID Number (Primary key)<br>First Name<br>Surname<br>Gender<br>Date of Birth<br>Street and number<br>City<br>Postal code<br>State<br>Diagnosis<br>Medical Doctor (foreign key)<br>Psychiatrist (foreign key)<br>Time of registry |
| |

Applying the same principles alluded so far, the related tables will have to following structure:

| **Medical Doctor** |
| --- |
| ID Number (Primary key)<br>First Name<br>Surname<br>Gender<br>Street and number<br>City<br>Postal code<br>State<br>Phone number |
| |

| **Psychiatrist** |
| --- |
| ID Number (Primary key)<br>First Name<br>Surname<br>Gender<br>Street and number<br>City<br>Postal code<br>State<br>Phone number |
| |

OK, we can see the primary key that Base will generate for us, we can see that atomization of name and address information. We can also see the gender descriptors. But wait a minute!! How come these tables have a phone number attribute? How come Phone Number is a table in relation to the Patient class but is an attribute in these two tables? This is dictated by the needs of the Client. He needs to record as many numbers from the patients as possible, in order to handle emergencies; but only needs one number from the professionals as their staff will know how to reach them if they are not immediately available. The defining factor is that in one case we need an undetermined number of phone numbers and in the second case we know that we need exactly one.

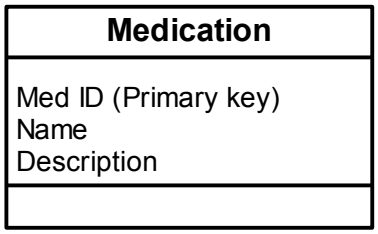



OK, we can see the foreign key, connecting the phone number with the proper patient. We can also see a descriptor with the very creative name of "description". We will use this to record if the number is the home number or the mobile or the office or the second job, etc. If it were an important piece of information, you could also add the descriptor that records the best time to call, but this didn't come up in our research.

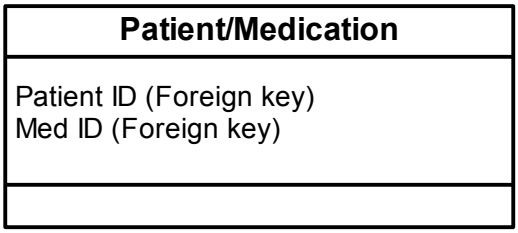
Now wait! Shouldn't there be a primary key attribute here? Strictly speaking, we don't need it, as all the information is relevant only in relationship to the Patient table. Our queries relating phone number will all have the form: "locate phone number where patient id is such". However, please note that when you are coding your tables in Base using the GUI (that is, the graphical interface as opposed to using SQL command lines) Base will automatically generate a numeric primary key in every table, just to keep things neat.

Another thing: isn't this a rather small table? Shouldn't tables be bigger? Isn't it a waste to create a table to just record two or three attributes? Well, we have seen that the power of relational databases comes from their ability to RELATE data, not from the size of its tables. It is not uncommon that databases are made of many, many rather small tables, but all really very interconnected. On the other hand, I am not sure if this table is small. It records only three attributes but, if every patient has exactly two numbers (home and office, for example; and about everybody also uses a mobile phone these days), this table will always hold -at least- twice as many records as the patient's table. Who's small, again?

With this in mind, the medication table should not surprise you:




Nor the intermediate Patient/Medication table:

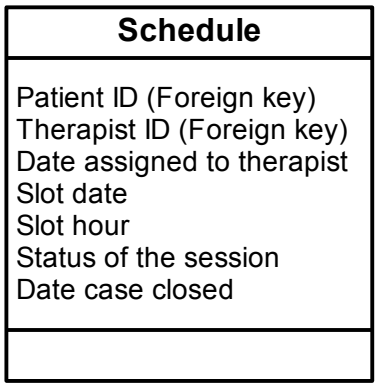


This last table uses only two attributes and doesn't even need a primary key, yet is allows us to find all the medications one particular patient is using or all the patients using a particular medication. Isn't this nice?

Could this table use a descriptor? Maybe we could have recorded the date this medication was assigned and the current dosage. And if we capture the date the patient started using this medication, we could also record the date he stopped using it, and keep this as an historical record for the patient. However, the need for this information did not arise in the goal statement and seems more relevant in a database for the psychiatrists, who is the one who is really monitoring the medication. We could ask the Client if he really needs this information or not. However, it works here to show that a table with only two attribute is not only possible, but also very important and useful.

At this point in the game, you should be able to organize all these tables in a nice UML diagram, describing the structure of the classes, connecting primary to foreign keys and indicating cardinalities. You should also be able to model the Therapist's table without help from this tutorial!

Let's focus now on the Schedule class. According to our research and the goal statement we did, this class should have the following attributes:

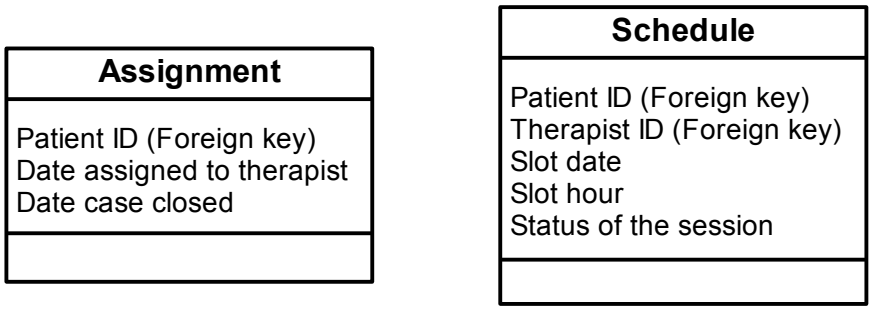



How are we feeling about this? We can see that in order to identify a slot hour, for example, we need to know both the patient id and the therapists' id. So this table uses a compound key to identify each cell. What is the rule of normalization for compound keys? Oh, right! Second normal form. That rule states that no subset of the primary key can also be a primary key. Is there anything like this here?

For the most part, I really need both elements for the key. After all, a patient needs to be assigned to a therapist from which he can have his case closed later on. Yet there is this strange feeling that the date of assignment and the space for storing the date the case is closed will be repeated unnecessarily every time a patient schedules a session. On closer inspection, the attributes Date Assigned and Date Closed depend on patient id only and not on the compound patient-therapist ids. This table is not in second normal form!

Following the rule then, we will decompose this table into one table that only has the Patient id, the

date he is assigned and the date the case is closed (which seems proper to call the 'Assignment' table) and have another table that records the patient id, the therapists id and the slot date and time, like this:




This is much better. We can see that in the schedule table, slot date, hour and status of the session (whether the patient came or not or if it was canceled by the therapists or the patient) depend exclusively on the compound key made by patient id and therapists id.

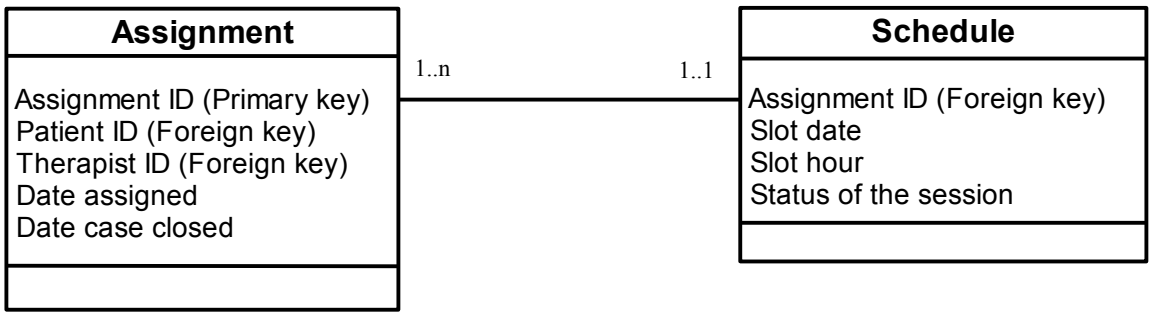
The assignment table, in turn, records the date a patient is assigned to a therapists and the date his case is closed, data that is functionally dependent only on the patient id key.

However, there seems to be a piece of missing data. I don't know you but but I feel uncomfortable by the fact that the assignment table does not record what therapist was assigned. Of course, I could simply add the therapist's id to the table. This would be correct and not affect the structure of the tables.

But if I do so, the assignment and schedule tables will both have the same two attributes of patient id and therapist id. This opens the door for possible inconsistencies that, of course, we want to avoid. How do we solve this?

Remember that I said that there is no one way to design a database? Our problem will serve as an example of that. There is no one rule that helps us here and, instead, we are going to have to use our imagination to solve this one. Different imaginations will come up with different solutions, and in all honesty, some time devoted to thinking about this could provide several options.

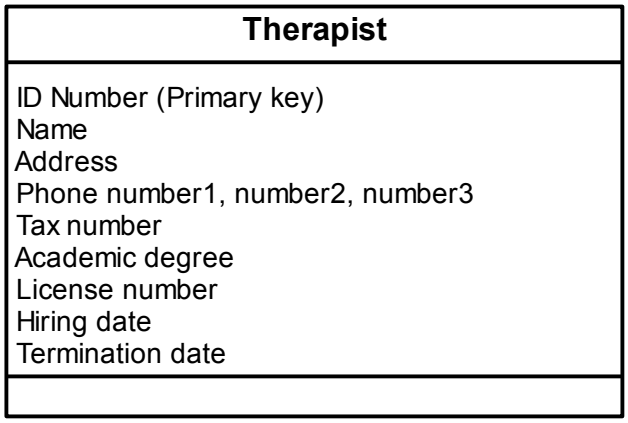
The solution that I use includes thinking of each assignment as its own entity. So I will record in the assignment table the patient id and the therapists id along with the dates of assignment and closing. I also provide each assignment with its own number primary key. Then in the schedule table, I use the assignment primary key as a foreign key instead of the patient's and therapist's id, and leave the slots time and date and the status of the session, like this:



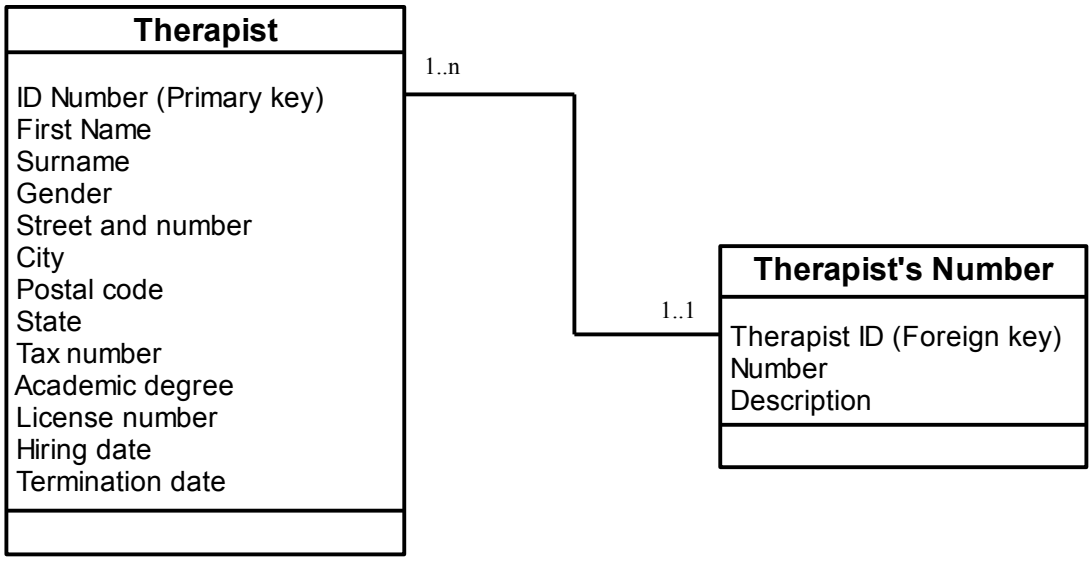

So far we have given examples of data modeling by using normalization, by using our knowledge of

the needs of the client and the way he handles business (coded in our goal statement) and now, by stretching our imaginations and creativity. Expect to do all these in your own projects!

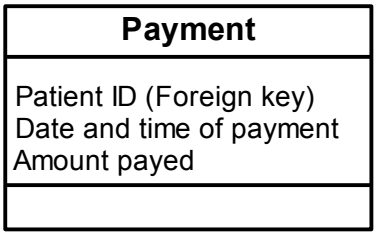
Let's finish this exercise in data modeling by conforming the therapist's table and the payment's table. For the therapist's table we will start with the necessary data identified by our research and later apply the principles of normalization, atomization of categories and applying gender descriptors. I will only render an initial and a final version. By now you should be able to describe the reasoning in the changes:




Transforms into:




Finally, we have the data required by the payments' table:

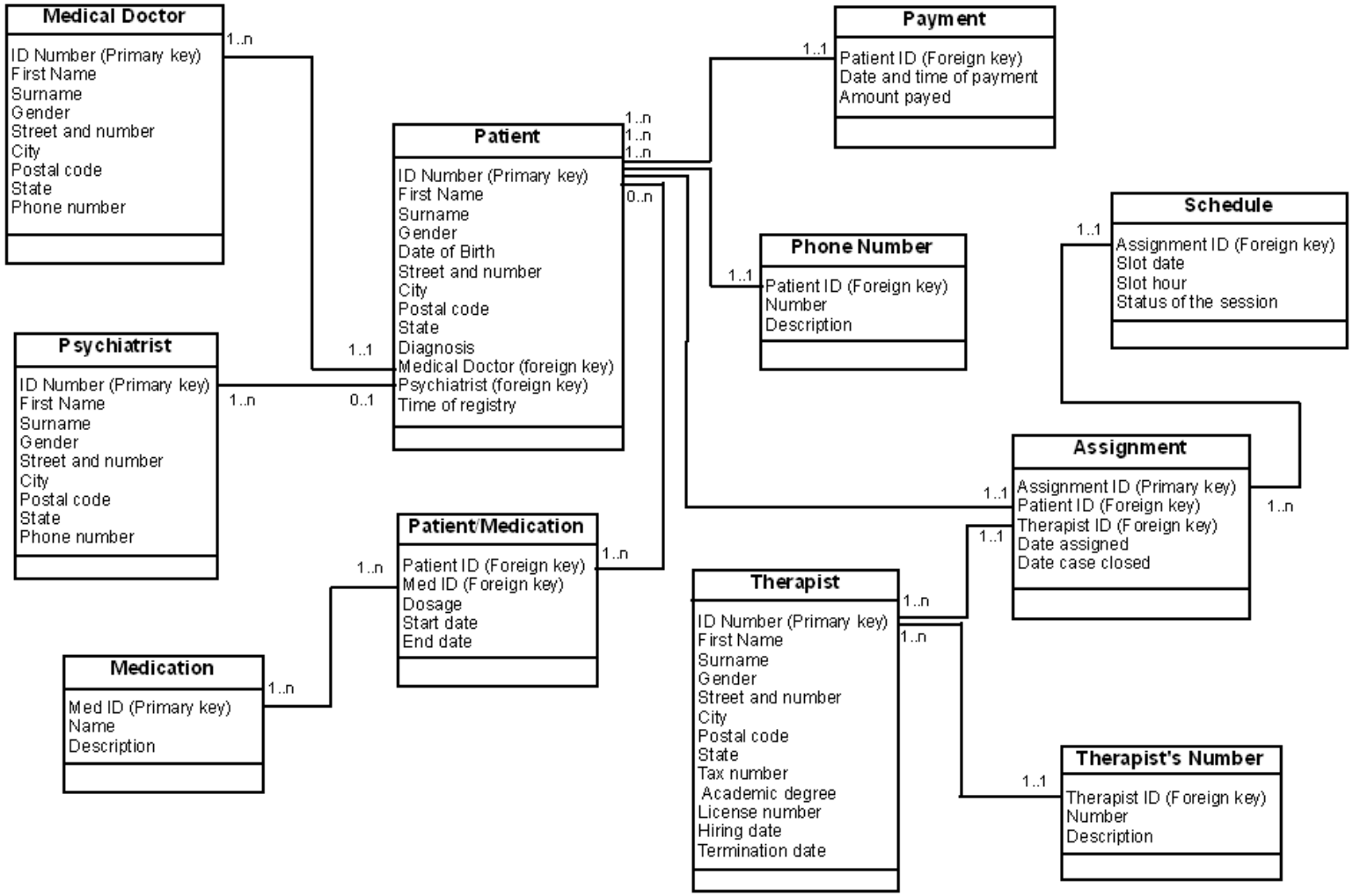


Date and time appear as one attribute because I intend to use the timestamp data type, that records both the date and the hour the record is created.

Transform this to the version by Drew Jensen

At this point we have finished the modeling of our data. We have expanded our initial draft that included only four boxes and made the necessary class extractions, we have determined the structure of each class and their structure of connections. By now you should have a UML diagram like the one in the next page.

# UML Design for Mental Health Clinic Database




**********Here the full UML diagram for this project

4. Attributes/Variable definition lists

Each one of our classes (tables) identifies the attributes (columns) they will need. These attributes will be stored as variables by the computer, of which we already know that there several types. Also, these variables will need names. These names could or could not be the ones we have been using so far... to make a long story short, each variable has a set of characteristics we need to specify. In order to make sure that we assign the most economic options and that we keep consistency in assigning these characteristics, we will make them explicit before we sit in front of the computer.

We do this by writing down a Variables Definition List. This takes the form of a table where we include the variable in question, the name we will give it, the data type it belongs to, maybe an initial value and any other remarks you might want to remember about them. This is a table that we, and not the computer, will be using; but that will make coding our application with Base much easier.

Notice that it is also possible to give our variables initial values, that will be automatically populated by Base when a new record is created. This is useful when you want to store a value by default, which will remain true until it is modified. It is also possible to create conditions that Base will look for before accepting data. For example, you can ask that Base check that the variable StartDate is always older than EndDate. Lastly, it could be possible that you are using arbitrary code conventions in your code (like 0=male, 1=female). It is a good idea to write down notes for yourself about these characteristics so that you don't forget to apply them and so that you apply them in a consistent way every time. It will also help you enormously when you sit down in front of Base and start coding.

For writing this lists, you can use Calc or a table in Writer. Let's write down the variables definition list for the Patient class:

| **Patient Class Variable Definition Lists** | | | | | |
|---|---|---|---|---|---|
| **No.** | **Description** | **Name** | **Type** | **Initial value** | **Remarks** |
| 1 | ID Number | patient_ID | Integer | 1 | Unique, auto increment |
| 2 | First Name | first_name | Var Char Ignore Case | Empty | Length = 20 chars |
| 3 | Last Name | surname | Var Char Ignore Case | Empty | Length = 20 chars |
| 4 | Gender | gender | Boolean | Null | 1= female; 0= male |
| 5 | Date of Birth | dob | Date | Empty | Format: mm/dd/ yyyy |
| 6 | Street and No. | address1 | Var Char Ignore Case | Empty | Length = 30 chars |
| 7 | City | city | Var Char Ignore Case | Empty | Length = 30 chars |
| 8 | Postal Code | zip | Var Char Ignore Case | 00000 | Length = 5 |
| 8 | State | state | Var Char | NY | Length = 2 |
| 10 | Diagnosis | diagnosis | Var Char | Empty | Length = 30 |
| 11 | Medical Doctor | md | Integer | Empty | Foreign Key |
| 12 | Psychiatrist | psy | Integer | Empty | Foreign Key |
| 13 | Time of registry | log_Date | Date | Today's date | Format: mm/ dd/ yyyy |

Notice how the names given for the UML design (second column) translate to names that our physical

table -created with Base- will actually use (third column). We explain this further in the section “On logical Names and Physical Names” bellow.

By now you should be able to understand all the information in this table. We have used an Integer data type for the patient id number, which is a signed 4 byte number with a range wide enough to store many patients. We are also writing the remark that this attribute should be unique to each record and should increment automatically for every new record, something that Base will do by itself. However, you need to code this -Base will not guess what your intentions are- so it is good practice to write this down so we don't forget about it. For Name and Surname we use the Var Char Ignore Case. Var Char means that this attribute will store **up to** the 20characters we have specified, or less depending on how many are really typed. It also means that Base will not pay attention to capitalization when it performs string comparisons, just in case there was a mistake during input. The gender has been assigned a Boolean variable that can store 0 or 1. We later remark that 0 will mean male and 1 will mean female. Date of birth has been assigned a Date data type and we remark the format used, to make sure we don't get confused later. The postal code receives an initial value of five consecutive zeros. This is a way to highlight the fact that this variable has not been properly recorded yet. Later, I can make use of OOo Writer's conditional capabilities and ask it to write the zip in red ink if value ='00000', for example. Could have I assigned this variable a Small Integer data type -that uses only two bytes per record- instead of a var char, that will use at least 5? Yes, you could have. Just bear in mind that if you do so, the general address information and postal code will belong to different data types so you don't mix them in ways that the application might not understand, Also, remember your choice so you don't attempt string manipulation on a variable that is of the numeric type. I have also assigned an initial value for the State attribute that I would most commonly find in the geographical area where I live, so I don't have to type it every time I create a new record. Finally, I made sure to make explicit that the values for medical doctor and psychiatrist are foreign keys. Of course, they have to be of the same data type that they will be assigned in their own tables. In general, all auto-generated primary keys will be an Integer type.

You will see that the SQL commands that we will use with Base to actually create the tables translates almost directly from this Class Variable Definition list. As an exercise, you could now write the variable definition lists for the other tables!

5. On logical Names and Physical Names

Throughout this tutorial we have tried to use the words “Class”, “Table”, “Attributes” and “Columns” as if they belong to different logical levels. We defined a class as a collection of object that share the same attributes. A class translates to a table and the attributes conform columns, which is why we have the temptation to use them interchangeably; but in fact they denote different things. A class is a logical relationship, formed when the abstractive powers of our mind are able to establish patterns of commonalities among the entities we are working with. A table is the physical expression of such a class, as embodied in the way the database software and hardware actually store, index and organize data.

When thinking about the name for our variables it can be useful to differentiate between a logical name for the variable and the physical name used in the application. For example, “First Name” is the name of the variable that stores the first name of our patients, like “John” or “Chloe”. “First Name” describes a data entity we a re working with. It does not matter if “First Name” is alternatively written “FIRST NAME” or “firstName” or “first_name” or even “F_N” as long as we conceptually understand that it references the first name of the patients. When we think about conceptual or logical

relationships, a clear and descriptive name is all we need. This is the kind of names we would use in our UML diagrams, which is an example of conceptual or logical data model. Unfortunately, this does not translate so simply into the names that a database software will allow us to use. For example, some software will allow us to use “First Name” but others would reject it and ask us to use “first_name” instead. This has to do with the way a particular database software has been designed and the conventions accepted then. The name actually used in the internal structure of our tables is called a “physical” name, as opposed to the logical name discussed above. As you can see in our table, our definition list does differentiate between the logical name (under the column called “Description”) and the physical name (indicated in the column labeled “Name”). This name (physical name) will have to conform to the conventions imposed by the software that we are using.

It is true but not all database software uses the same rules for variable name formation. For example, many special characters are not accepted as part of names (e.g. %, ^, @, =, etc.) and some applications do not allow spaces between words in variable names. Thus, 'First Name' would be rejected as a name. Numbers, or the character underscore '_', are accepted provided they are not the first character in a name. This way we could name our 'First Name' variable something like 'firstname' or 'first_name'. The format 'thisisthenameofmyvariable' could be quite hard to read, so many coders capitalize the first letter in each word, like 'firstName' or 'FirstName'. Keep in mind that some software is caps sensitive, so 'Var' and 'var' would be treated as different variables. If you don't keep consistency, you could go crazy trying to figure out why firstName is empty, when you really gave a value to FirstName.

To be clear, Base using the HSQL engine can accept spaces and other special characters as valid characters for names and can also act as caps sensitive.

Why do I care about how other database programs handle variable names?

You may have noticed how much thinking goes into designing the tables and their connections. Compared to that effort, writing the code to build them can seem almost trivial. Once you have the logical structure, you can easily transport it to any database application. What if the code with which you create the database in Base could be transported to other database software as easily? This is not always possible because of different physical name conventions. Furthermore, Base can act as a front end for several different applications. If you are planning to use Base in this way then you might want to make sure that the application that you will migrate to or access through Base accepts the variable names you have chosen and properly understands the variable types that you are using.

We should also be aware of a couple of problems when using long variable names like 'thisisthenameofmyvariable'. First, it is quite easy to misspell a variable like this, causing Base to believe that we are using a variable that is empty or not defined in the query we are trying to build. Second, it can make the formatting of reports a real nightmare. Let's say that you are building a report that has a column for the variable: “totalnumberofwives-freeunions” to represent a small integer with values like “1” or “0”. The length taken by the name of the variable in relation to the length of the actual values will make it difficult to calculate the final layout of the report and ensure it could be read easily.

To standardize our options we could conform to the following practice when naming variables:

1. All variable names will start with a letter.
2. Subsequent characters will be either letters, numbers or the underscore character.
3. We will not use a space between words. Instead we will separate them with the underscore.

4. We will not use special characters except for the underscore.
5. We will use abbreviations if needed to help keep the length of variables names short.

What does short mean, exactly, will be determined by experience and the context of the problem. You will see how our example uses abbreviations.

6. A little bit more: Duplicity of roles and super classes.

The example that we have developed in this tutorial belongs to the very sophisticated field of medical records and medical billing, one obvious area for database applications. This example pales when compared to the functionalities provided by professional applications in this field. For example, we have not recorded what insurance company is in charge of the patient's payment, most of which have different procedures and time lines. A pro design would be able to handle all that. Also, What happens if the patient is a minor or, for any other reason, has a legal guardian assigned to him? All this kind of exceptions -and more- can be handled by professional designs. Why hasn't this example gone further?

First of all, medical billing is a complex subject in itself, that is taught at the college level and is based on many years of trial and error by many bright minds tampering with this. In contrast, this tutorial aims to give you tools to develop more general types of databases, not only medical billing. However, most of those extra functionalities can be implemented using the same principles described so far: by understanding well the data that your clients need to record and retrieve, by harnessing the power of a RELATIONAL design, by not defaulting normal form, by thinking your designs carefully, by challenging them and by testing and testing and testing them.

However, there is a characteristic of the design we have arrived to in this tutorial that could become a problem in a different context. Maybe you have even spotted this potential problem already.

If you remember, one of the goals of proper database design, is to minimize redundancy (number). Our design does this quite well as long as a therapists of the clinic is also not a patient of it.

If you review the data structure for the therapist's and the patient's tables, you will notice that both start by asking name, gender, address and phone numbers. If a therapists were also a patient, we would have to repeat this data in the patient table, opening the chance for inconsistent data and, in any event, duplicating records.

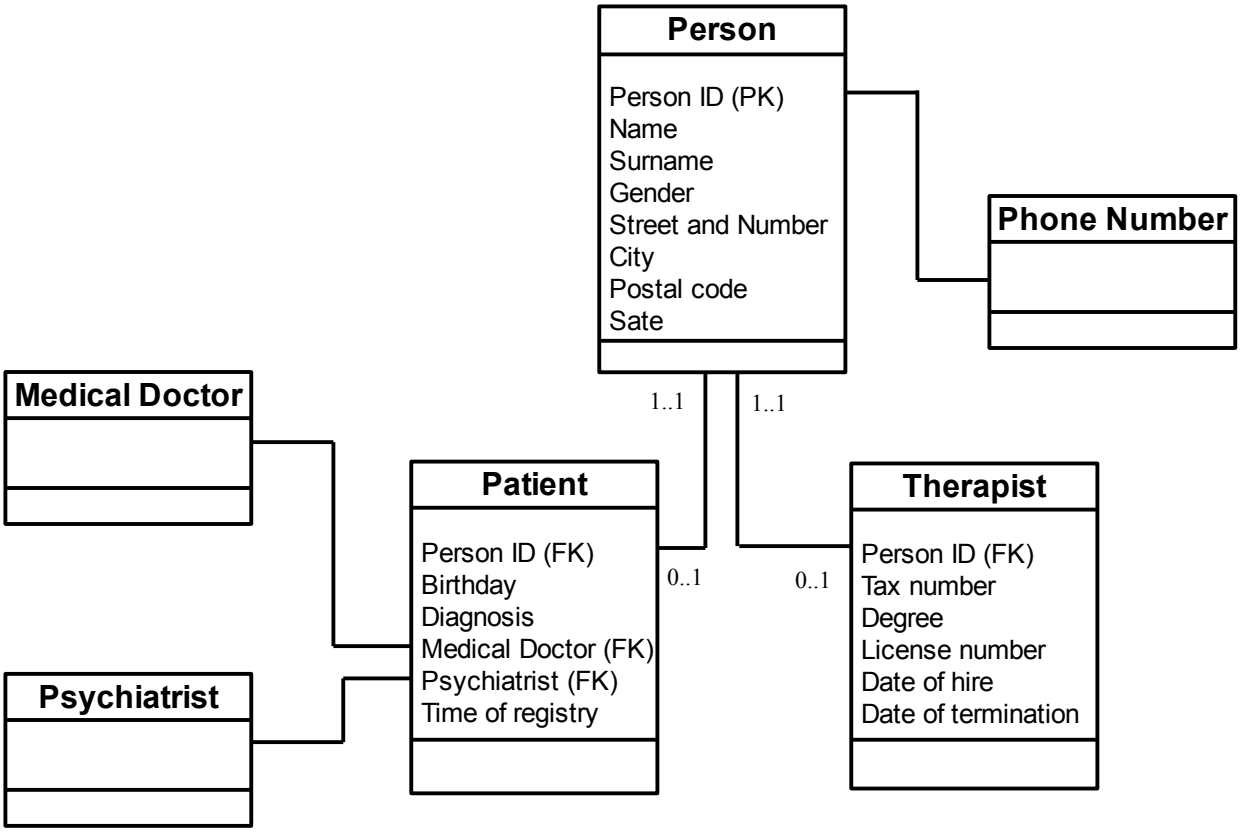
Now, maybe this design makes sense in the context of a small mental health clinic where, for ethical reasons, it is not a good idea that a therapists also becomes a patient of it. But there are many other examples where we could expect this duplicity of roles and should be ready for it. For example, in a film school we have students and professors. But it could also happen that a director of photography professor enrolls in a producing course. It would also be possible that some third year students make some extra money by assisting the audio recording professor in sound lab. It these cases, we have professors that are also students and students that are also teachers. Teachers that also study need to be enrolled, pay for the courses and receive the same benefits other students do, data that is probably not stored in the professor's tables. The same way, students that teach need to be paid and taxes deducted, and require other data that will not appear in the regular student's tables.

If we record such kind of person twice, once as a teacher and once as a student, we are duplicating the common records and opening the chance for inconsistent data.

In cases where a duplicity of roles arises, you might want to think in terms of a super class, a class that stores all the attributes common to the different roles and later relates to them as sub-classes. For example we can create the class 'Person', which will have the attributes common to students and professors (name, address, etc). We then connect this table to the student and professor tables which will record the attributes specific to each role. This way, one person can be either a student, a teacher or both.

Let's see this in the context of our own example. We have the patient and therapists tables. When we analyze their data structure, we discover that both tables require: id number, name, address, gender and phone numbers. We can extract this data to form the 'Person' table. The patient table will be left with the following attributes: a person id foreign key, birthday, diagnosis, a medical doctor foreign key, a psychiatrist foreign key and a time of enrollment. In turn, the therapist table would be left with: person id foreign key, tax number, degree, license number, date of hire and date of termination.

This structural change is summarily reflected in the following UML diagram:




Unless you have a good reason to keep the subclasses separate, like the ethical consideration in the clinic we are deriving our example from, the notion of a super class connected to related subclasses is the solution you most often will want to apply when you find duplicity of roles.

Now that we have the design of our database, let's check the forms and reports that we will want to use with ir.

7. The forms:

Let's now analyze the forms for data entry the way we envision them for the database design in this example. Let's start with the form for patient information (figure 1).

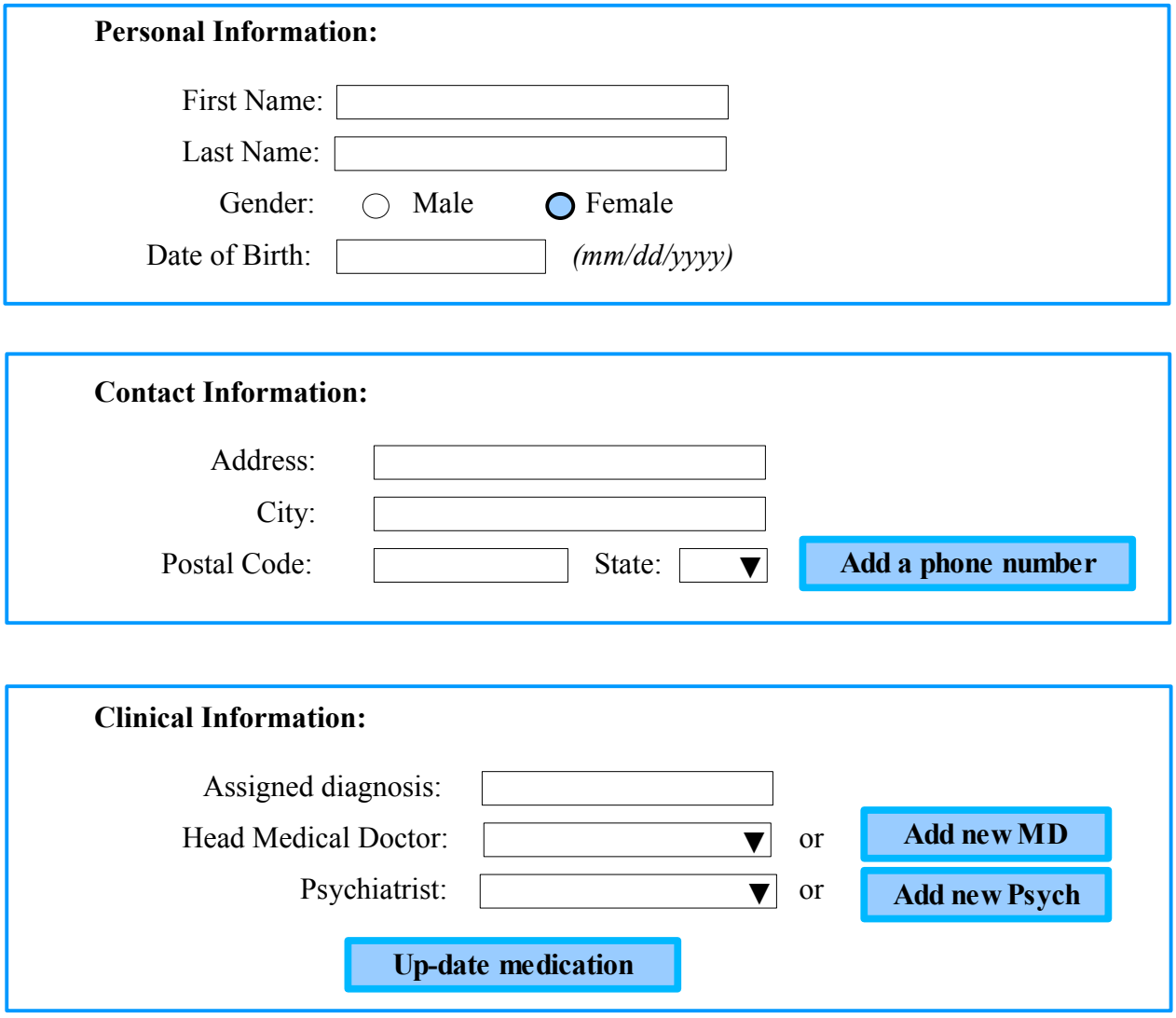



**Figure 1: Patient information entry data form.**

I have divided this form in 3 components just for ease of use. The first box, Personal Information, will receive the first name, surname and date of birth with text boxes. Nothing new here if you have studied other manuals for form design with Base. Note that for entering the gender I do not need to write anything at all: not “male” or “female” nor “M” or “F”, nothing. All I have to do is click on the appropriate radio button. Of course this means that we are going to use some programming when we actually develop this form. But that extra work will simplify our data entry and will make sure we only enter values within a set expected by our design.

The second box, Contact Information, has two interesting features: There is a drop-down menu for the input of the Sate info, which is another way to make sure that the data entered belongs to a predefined set of options. Then we have a button for adding phone numbers. If you remember, our design calls for an undefined number of phone numbers for patients, for which we have a dedicated table. If we want to enter a phone number we will click on this button, which will open the phone data entry form (see figure 2). After we enter the phone number, we close that form and return to this one. Then we can

click again on the button, and so forth, for every phone number we need to enter. We are going to need to program this new form so that it automatically associates the phone number to the primary ID number for the patient being recorded. This automatic registry will not be noticed by the user (saving him time), but is fundamental for the design to work.

The third box, Clinical Information, also has some interesting features: Notice that the assignment of a MD or a psychiatrist is done from a drop-down list. This ensures that this information is entered without errors that could make it difficult later to identify this data properly (different or wrong spelling, for instance). The form will have to read from the database to identify all available MDs or psychiatrists. This means that this is some kind of dynamic form that is created based on previous entries. What happens if the MD of the patient has not been entered before? It will not be available as an option from the drop down menu. For this reason we have another button here, labeled “Add new MD”. Clicking on it will open a form for entering the MD information. After we do and we close that form, this form will update itself and will show the name of the new MD among it's options. The same process is necessary for the entry of the psychiatrist's information. Also notice that the form will provide the name and surname of the MD but, once we identify and chose one, what Base needs to record is actually the primary key of the MD as a foreign key in the patient's table.

The third box also includes a button for adding medication, which will open the medication data entry form (see figure 3) if we click it. The procedure for this will be very similar to the the programing required for the “Add a phone number” button.

Of course we, the developers of these forms, will have to program these functions, which will take an extra effort, but they will pay with better performance.

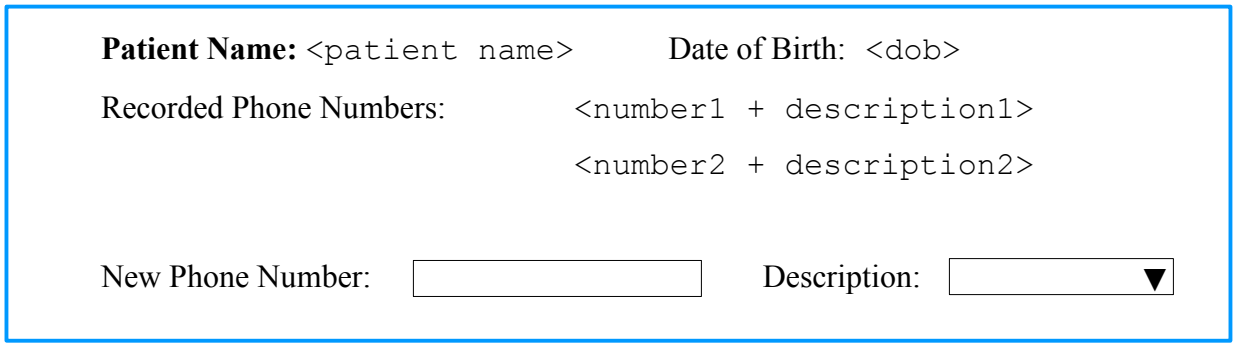



**Figure 2: Phone numbers entry form**

The form used to register the phone numbers also has a dynamic quality. Note that we expect that, when we call it, it properly identifies the patient we are making a registry for. It must also show the date of birth, just to make sure we are not adding data for another patient with the same name. Alternatively, we could use a registry number, for instance, the primary key of the patient. Both options have pros and cons: the probability of entering data for different persons with the same date of birth and name is very small but not zero; in contrast, using a registry requires that we have a validation for that number (a log, for example) at hand. We will use both options (in different forms) because we are more interested in showing how to create these dynamic forms. Which option you use will be a matter of your design goals. We also want this form to show the phone numbers for the patient that we have entered so far, to make sure that we do not duplicate records. This, again, calls for a dynamic creation of this form, based on data that we have entered previously.

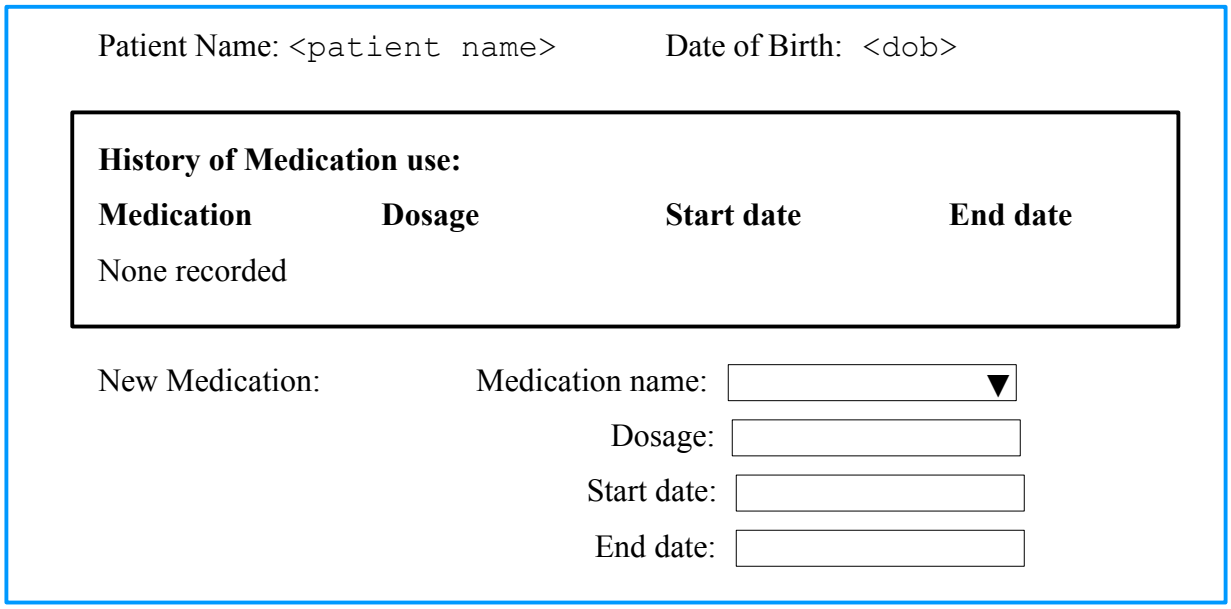



**Figure 3: Medication Assignment form**

The form for medication assignment also needs to be formed depending on previous entries (figure 3). Note that we need this form to display the date of birth (or primary key) in order to confirm identity and to include the medications we have entered so far. Note that the name of the medication will be picked from a drop-down list that will be populated from records in the medication table. This table assumes that the database design will record dosage and the date the medication stared and ended.

The Assignment Form (figure 4) is one of my favorites because it seems so simple, but many things are happening under the hood, aside from being crucial for the working of the database. First, notice that the form just looks to pair a patient with a therapist. However, the form will provide options with drop-down lists populated from records in the database. This will ensure that only recorded patients be assigned, and because this is a point and click operation, we eliminate errors in spelling. If you remember, it is also very important that we record the date of this assignment. This will be done automatically by the form (with lots of help form our programming), so the user does not need to concern himself with remembering what date is today. If the final user does not have access to the code or to modifying this record, then this date registry could also have the authority to establish a time line. Quite cool!

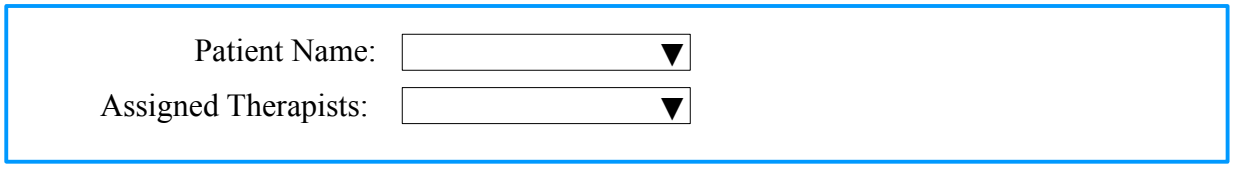



**Figure 4: Assignment form**

For closing a a case we could use the form in figure 5. Note that all data about the file is populated by the form. We have left the closing date for the user to input but we could have, as well, automated it.

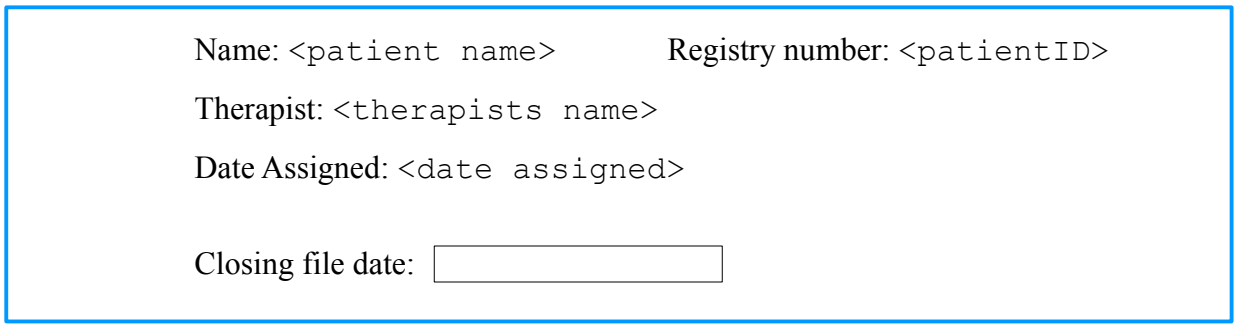



**Figure 5: File Closing form**

The Scheduling Form (figure 6) requires elements that we have already discussed. The purpose of this form is to set the date for a future appointment. The form will provide the name of the patient and a registry number to confirm identity (instead of the date of birth we have been using so far) and the name of the therapist. All these elements have been assigned previously. The user will input date and time of next session.

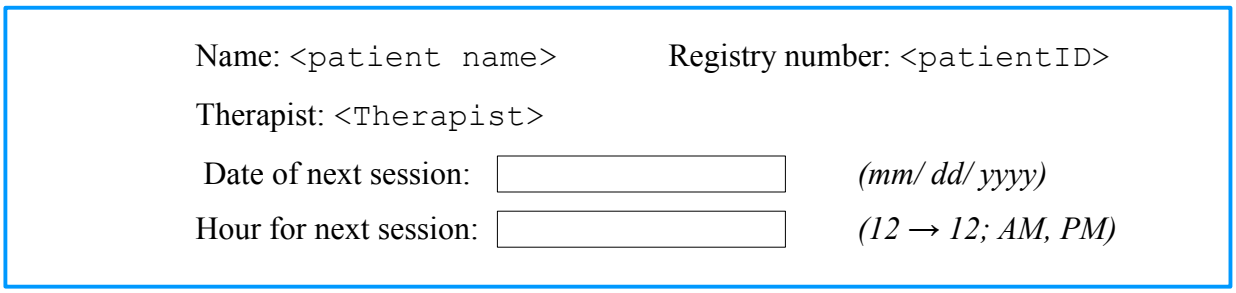



**Figure 6: Scheduling form**

The last relevant form is the Follow-Up Registry. Here is where the user records whether the patient came or not or if the session was canceled and by whom. Notice that, again, we are going to have Base provide the elements that identify this registry: The form will indicate the name of the patient, a confirmation element (the Registry Number in this case), the therapist assigned and the date and time when the session was supposed to take place. Bellow, the form offers just 4 radio buttons of which the user must chose only one. That will be a simple click but will allow complex calculations that have to do with generating bills and payments.

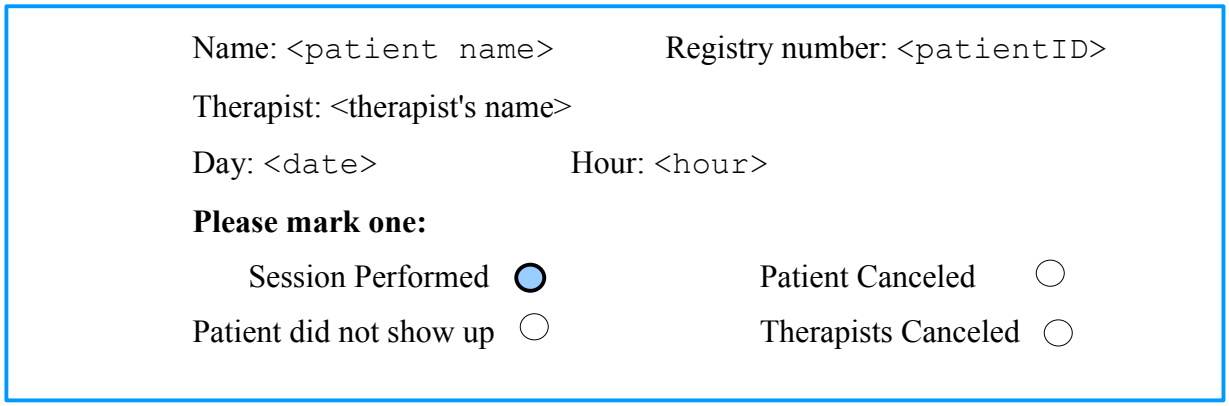



**Figure 7: Follow-Up Registry**

To sum this up, the forms that we need to create will receive and understand input from radio buttons and drop-down lists, and most of these drop-down lists will be generated based on data recorded in the database (an exception would be the Sate info in the Contact Information box and the "Description" info in the Patient Phone Number Entry form).

Additionally, the forms themselves will be generated dynamically, including information previously recorded in the database. Other required forms (like the forms for the MDs or the psychiatrists, for example) that are not described here either do not require these elements or require elements that have already been analyzed.

8. The reports

Let's now analyze the design of the reports that our client seems to need. We will produce six reports:

1. Clinical summary, which displays the patient information and the clinical information for each patient;
2. History of services and payment summary, which displays the history of sessions (whether they happened or were canceled) and the history of payments performed in a time window;
3. Therapist's summary of all sessions performed by therapist in a time window;
4. Clinic's summary of all sessions performed by the Clinic and all payments received in a time window;
5. Next day Schedule confirmation table, which lists all patients that have scheduled a session for a specified day, with contact data so someone can call and confirm the appointment; and
6. Daily Schedule, which lists all patients for each therapist, organized by time.

Let's review them in more detail:

1. Clinical Summary:

---

**Patient Information:**

Date: `<today's date>`

| | | | |
|---|---|---|---|
| Name: | `<Name Surname>` | Registry: | `<patientID>` |
| Date of birth: | `<dob>` | Gender: | `<gender>` |
| Address: | `<address1>` | | |
| City | `<city>` | Sate: | `<state>` |
| Postal Code | `<zip>` | | |
| Phone Numbers: | | | |
| | (xxx) xxx-xxxx | Home number | |
| | (yyy) yyy-yyyy | Work Number | |
| | (zzz) zzz-zzzz | Cell Number | |

---

**Clinical Information:**

Date of admission: <date of admission>  Date of termination: active

Therapist: <Therapist>

Diagnosis: <diagnosis>

Head Doctor: <medical Doctor>  Tel.: <phone number>

Psychiatrist: <psychiatrist>  Tel.: <phone number>

| Medications: | | | |
|---|---|---|---|
| | Abilify 20 mgrs./day | From: 09/12/2007 | until: Today |
| | Prozac 0.5 mgrs/day | From: 05/23/2007 | Until: 09/12/2007 |

---

**End of Clinical Summary**

This report relates a patient ID will all contact information and the corresponding phone numbers, MD, psychiatrist, medication history and assigned diagnosis. The most recent medications are listed first. Notice that if there is no date of termination, the report will write “Active”. In the same line, if a medication has no termination date, the form will write: “Today”. Also, notice that this report sets today's date in the right hand corner, so we know that the report is valid as of that day, for any information that could change (like address, phone numbers or medication). Finally, notice that the report ends with a line under which we write : “End of Clinical Summary”. The purpose of this is that, if this report had more than one page, you can easily find where does it end. Picture yourself producing several of theses reports and you can see why a clear end like this could be useful.

2. Patient history of services and payment summary:

---

**Patient History of Services Summary**

rendered between <date1> and <date2>

Date: <today's date>

Name: <Name Surname>  Registry: <patientID>

Date of services:

| Date: | Status: | Value: |
|---|---|---|
| 08/12/2007 | No Show | $35 |
| 08/19/2007 | Patient Canceled | $0.0 |
| 08/26/2007 | Performed | $35 |
| 09/03/2007 | Performed | $35 |
| 09/10/2007 | Performed | $35 |
| 09/17/2007 | Therapist Canceled | $0.0 |
| 09/24/2007 | Performed | $35 |
| 09/30/2007 | Performed | $35 |
| | Total: | $210 |

---

**Payment History**

Name: <Name Surname>  Registry: <patientID>

as of 10/05/2007

| Payments Received on: | Amount: |
|---|---|
| 08/26/2007-13:30:22 | $35 |

| | |
|---|---|
| 09/03/2007-13:34:53 | $35 |
| 09/07/2007-11:12:33 | $35 |
| 09/10/2007-13:32:41 | $35 |
| 09/24/2007-13:37:12 | $10 |
| 09/30/2007-13:29:34 | $55 |
| Total Payments: | $205 |

---

**Balance Summary**

total services amount: $210 - total payments: $205 = -$5

---

**End of Patient History of Services Summary**

This report is built using the patient ID and all sessions scheduled, in the top half, and all payments made, in the bottom, between a specified time window (that is, between `<date1>` and `<date2>`). The amounts are added. Then a balance is calculated. Patient and date information are repeated to simplify lecture (maybe this gets to be along list). We also display the date the query is run and terminate the report with a clear notice.

3. Therapist's summary:

---

**Therapist's Services Summary:**

Date: `<today's date>`

Therapist: `<Therapist>` Registry: `<therapistID>`

rendered between `<date1>` and `<date2>`

| Date | Patient | Status | Value |
|---|---|---|---|
| 07/01/2007 | `<surname>` | No Show | $20 |
| 07/01/2007 | `<other>` | Performed | $20 |
| 07/01/2007 | `<andother>` | Performed | $20 |
| 07/01/2007 | `<yetother>` | Canceled | $0.0 |
| and so on... | | | |
| 07/30/2007 | `<lastone>` | Performed | $20 |
| | | Total Services: | $1207 |

---

**End of Therapist's Services Summary**

This form is built with all scheduled sessions for a particular Therapist's ID and a time range, The list is chronologically ordered and, within the same date, alphabetically ordered by the surname of patients. He report would never write “and so on...” this is just something I wrote so that you get the idea. The ellipsis represents a continuation of the list of events. Date and end of report are also clearly identified.

4. Clinic's summary of services rendered and payments received:

---

**Clinic's Summary of Services**

rendered between `<date1>` and `<date2>`

| Date | Therapist | Status | Value |
|---|---|---|---|
| 09/15/2007 | `<Atherapists>` | No Show | $35 |
| 09/15/2007 | `<Atherapist>` | Performed | $35 |
| ... | | | |
| 09/15/2007 | `<Btherapist>` | Performed | $35 |
| 09/15/2007 | `<Btherapist>` | Canceled | $0.0 |
| ... | | | |
| | | Total Services: | $5700 |

---

**Summary of Payments:**

rendered between `<date1>` and `<date2>`

| Payment Received On: | By Patient | Value: |
|---|---|---|
| 09/15/2007-13:30:22 | `<Apatient>` | $35 |
| 09/22/2007-13:34:53 | `<Apatient>` | $35 |
| *etc. for* `<Apatient>` | | |
| 09/16/2007-11:12:33 | `<Bpatient>` | $35 |
| 09/23/2007-13:32:41 | `<Bpatient>` | $35 |
| 09/24/2007-13:37:12 | `<Bpatient>` | $10 |
| *etc. for* `<Bpatient>` | | |
| 09/30/2007-13:29:34 | `<Cpatient>` | $55 |
| *etc. for* `<Cpatient>` | | |
| | Total Payments: | $5670 |

---

**End of Clinic's Summary of Services**

This report has two components: first it states all sessions performed by all therapists (in alphabetical order) in a selected time range. The form includes the status of the session (performed, canceled, etc) and the value charged to the client. The 'Value' column has a total sum at the bottom. The second component lists all payments made for the same time range, ordered by patient's surname and chronology of payment; and includes the timeestamp and the actual values payed by them. A sum of all payments is also done at the end of the 'Value' column.

5. Next day Schedule confirmation table

---

**Future Sessions Confirmation Table**

**For Date:** <selected day>

**Name of patient:** <name surname> **Tel.:** <phone number>, [<phone number>]

**Therapist:**<therapist>

**Time slot:** <hour> ❑: Confirmed

❑: Re-scheduled for: _____ ❑: Canceled

**Name of patient:** <name surname> **Tel.:** <phone number>

**Therapist:**<therapist>

**Time slot:** <hour> ❑: Confirmed

❑: Re-scheduled for: _____ ❑: Canceled

**Name of patient:** <name surname> **Tel.:** <phone number>, [<phone number>]

**Therapist:**<therapist>

**Time slot:** <hour> ❑: Confirmed

❑: Re-scheduled for: _____ ❑: Canceled

---

**End of Confirmation Table**

The boxes will have no other functionality than letting the user of a printed version of this report check the result of the confirmation. This report will list every patient scheduled for the specified day.

6. Daily Schedule

| Hour | <therapist1> | <therapist2> | <therapist3> | Etc. |
|---|---|---|---|---|
| 9:00 AM | <Patient> | <Patient> | **Empty** | Etc. |
| 10:00 AM | **Empty** | <Patient> | <Patient> | |
| 11:00 AM | <therapist><br><Patient> | <Patient> | <Patient> | |
| 12:00 Noon | <Patient> | **Empty** | <Patient> | |
| 01:00 PM | **Empty** | <Patient> | **Empty** | |
| 2:00 PM | <Patient> | **Empty** | **Empty** | |
| Etc. | | | | |

Here we organize the sessions by therapist and by time slot. We could provide colored backgrounds to make it easier to find a particular patient or time slot. The Etc. Means that there could be more (of the same kind) of information and is not something that would actually appear in this report..

9. Turn on the computers

We are finally ready to sit in front of the computer and create our application with Base. We have a database design (we will use the one in page xx which handles therapists and patients separately), we know the forms that we need and we know the reports that we want. We have also built variable definition lists for all our tables. All the thinking about our particular database has been done so now we can focus on the complexities of coding it with Base.

Our first step will be to actually build the tables. You can do this with the graphical user interface of design mode, with the help of the wizard or by coding them with SQL commands. There are excellent tutorials that explain how to build your tables using the first two methods and I truly hope that you have studied them by now (cite references). However in this tutorial we are going to create our tables using SQL commands. This might seem less glamorous than creating the tables by using the graphical interface (GUI), but you will quickly notice how it is as easy and avoids some limitations of the GUI, like the imposition that all tables have a  numeric primary key.

We will then build the forms for this database. Make sure to have also studied the excellent tutorials on this topic(cite references) as we will, again, only focus on some advanced capabilities, like using drop-down menus and other ways of providing input. This, in turn, will require us to understand how to use some OOBasic in order to give our buttons the desired functionality

Finally, we will produce the reports that, as you have seen, also focus on advanced functionalities. This is because these reports not only retrieve particular records but also offer summary information. Let's get started!