MySQL wizardry

By Giuseppe Maxia

Table of Contents

Introduction

Cross tabulations are statistical reports where you de−normalize your data and show results grouped by one field, having one column for each distinct value of a second field.

*Basic problem definition.
Starting from a list of values, we want to group them by field A and create a column for each distinct value of field B.*

The desired result is a table with one column for field A, several columns for each value of field B, and a total column.

*According to some authoritative sources (Joe Celko, "SQL for smarties") we should use specialized (and expensive) statistical tools to achieve this purpose with a database server.
My recent experience with MySQL has shown that you don't have to invest a fortune to have server−side cross−tabulations. The way I found it is littered with errors and disappointment, and in perspective it should appear quite boring. This is the chronicle of how I would have liked to find out a solution.*

A call for help − Defining the problem

I called in the Wizard on a Friday evening. It was almost six p.m. and I was afraid he had already left for the day. Instead, he answered at the first ring.

Wizards never go home early.

He recognized me and asked how he could help. I told him. He listed patiently, without asking silly questions in between, and finally said "I think I could provide you with some useful tool. See you in half an hour" and hung up.

Twenty five minutes later he was in my office, sitting in front of me and mentioning coffee. Then he started to ask questions.
"So, what exactly do you need to do?"
"I need to get a cross−tab out of my database."
"How did you solve the problem so far?"
"In the beginning, I used to export my data to a spreadsheet, and let the users do the work. You know, nowadays these spreadsheets can do almost anything, provided that you are smart enough. Therefore, I gave my users the possibility of exporting records to their favorite application so that they could twist the data as they liked."
"And why you can't do it anymore?"
"We're growing, you know. When I started the database, we had just a few hundred records, and everything worked just fine. But then, we hit the market really hard, and before I could realize it, we had more than ninety thousand records that were about to go down to the spreadsheets and the users complained that it was slow and asked me to integrate the x−tab into the main application. They say that, after all, these nice desktop databases that I don't want to mention −− the Wizard nodded approvingly −− have such features, and my application should have it as well."
"Then you did what they asked for. You created a function to integrate the cross−tabulation into your application." He was reading me like an open book.

"Yes, of course I did it, even though it was not easy. To minimize the amount of data transferred from the server to the client I was only dealing with summarized data, you know, GROUP BY two fields. An then I found a way of translating the values of the second field into columns and summing up the data."
The wizard looked at his coffee mug for a long time and said "You seem to have solved your problem then. Haven't you?"
"Well, not exactly. Dealing with a large matrix of data is not what C language seems to be made for. I mean, it is, but I can't cope with it as well as I can work out a few spreadsheet macros. And the algorithm has to take care of peculiar cases where the server does not return a value for the second field, and then the management wanted to have the cross−tab broken down by additional rows and columns, and they want such reports to be available into the main application by Monday, and I don't think my algorithm could cope with such request. This is why I asked your help."

I looked at him expectantly. I knew that now he could do one of two things: he could either say that the problem was trivial and uninteresting and he would leave me in the glue, or he should ask to see the algorithm and tell me what was wrong with it and give me a simple and efficient solution, which will make my application scalable to solve the darn problem. He had done that before for me. He had looked at my application, found the reason for inefficiency, suggested a simpler approach, and left me happy, wiser and puzzled.

That day, though, he did none of that. He didn't want to see the code at all. Instead, he asked for a refill of coffee and told me "Give me a tour of your company." I understood what he wanted. He wasn't looking for a walk among the desks, but he wanted to see the data. I explained that I could not show him the real data, since my boss was really concerned about the competition learning what we are up to, but happily I had some dummy data that I was using with our real structure when I was testing the database, and I showed him that.

"This is the design of the main tables in the DB." I explained. "We have an *employees* table, code−related with *departments* and *locations*, which is also related to c*ountries*. The *sales* table has references to the *employees* and *categories* tables. Only employees belonging to the Sales department can be involved in sales, but they can sell anything, from software to services to education."

The Wizard studied the diagram for a few minutes, nodding from time to time, as if recognizing an old friend. Then he looked at me and said "OK. Let's do some cross tabulation. Do you have anything especially urgent?"

"As a matter of fact, I have more than one specific urgent job, but if you don't mind, I would like to see the solution of a simple one, so that I can work it out on a more complex one." "It's fine with me." He said. "Show me your case."

"OK. I will query two tables, *employees* and *locations*, and get the list of employees with the town where they work."

I opened a Xterm in my Linux box and connected to MySQL. There I entered:

```
mysql> SELECT name, gender, location FROM locations INNER JOIN employees USING (loc_code);
```

name	gender	location
Luigi	M	Roma
Mario	M	Roma
Fred	M	Milano
Cinzia	F	Cagliari
Marco	M	Cagliari
Jim	M	Roma
John	M	Milano
Sue	F	Cagliari
Maria	F	Paris
Giselle	F	Marseille
Sonia	F	Marseille
Jacques	M	Marseille
Paul	M	Paris
Jennifer	F	Manchester
Julie	F	New York
Christine	F	London
Don	M	London

Sam	M	Manchester
Colette	F	New York
Connie	F	Boston
Guy	M	Boston
Steve	M	New York
Antonio	M	New York
Nina	F	Boston

```
24 rows in set (0.00 sec)
```

"What I would like to have," I explained, "Is a row for each town, with a column for each gender and a total column."

Opening the path

The wizard took a sheet of paper and drew a table

Town	M	F	total

"We will go for it manually." He said. "This way, we are going to understand what to ask the database engine to do." It was typical of the Wizard. When he was in this explaining mood, I should better let him talk.
"Let's start with the first row. It's *Roma*. The employee is male, so we write **1** under *M* and **0** under **F**. Then we get the second line. It's again *Roma*. Which gender is this one? If we have a male, then we are going to add **1** to the value under **M**, and add a **0** under **F**, and so on."

Town	M	F	total
Roma	*1+1*	*0+0*	

He looked at me, as if expecting me to see the light and have a magic understanding of the algorithm he was hinting at. My blank stare must have told him that I was still at large. "Don't you get it? It's simple. We **sum** to M **if** the employee is male, and we **sum** to F **if** she's a female." He stressed the words **sum** and **if**. Then he grabbed my keyboard and modified my previous statement:

```
mysql> SELECT location, SUM(IF(gender='M',1,0)) AS M, SUM(IF(gender='F',1,0)) AS F
FROM locations INNER JOIN employees USING (loc_code) GROUP BY location;
```

location	M	F
Boston	1	2
Cagliari	1	2
London	1	1
Manchester	1	1
Marseille	1	2
Milano	2	0
New York	2	2
Paris	1	1
Roma	3	0

```
9 rows in set (0.00 sec)
```

"So we are telling the engine to do exactly the same thing that we would have done manually. **Sum ... if**. Only the engine will do it faster."

I said "Wow!" but my mind was racing to see how this incredibly simple statement could be of help. "What about the total column?" I asked.

"Oh, that. Here you are." And he modified the statement once more:

```
mysql> SELECT location, SUM(IF(gender='M',1,0)) AS M,
SUM(IF(gender='F',1,0)) AS F, COUNT(*) AS total
GROUP by location;
```

location	M	F	total
Boston	1	2	3
Cagliari	1	2	3
London	1	1	2
Manchester	1	1	2
Marseille	1	2	3
Milano	2	0	2
New York	2	2	4
Paris	1	1	2
Roma	3	0	3

```
9 rows in set (0.00 sec)
```

"I don't think I really understand, though." I said. "We need to count, but we are summing up. How comes?"
"From the SQL point of view, we are doing the same thing. COUNT of star and SUM of one are the same thing. Try it yourself. Type a 'select COUNT star from employees'".

```
mysql> SELECT COUNT(*) from employees;
```

count(*)
24

```
1 row in set (0.00 sec)
```

"Now replace COUNT of star with SUM of one."

```
mysql> SELECT SUM(1) from employees;
```

sum(1)
24

```
1 row in set (0.00 sec)
```

"It's the same!" I said, excited.
"No, actually it's not. COUNT of star is optimized by MySQL, and it is performed from the table descriptor, without actually counting the records. You can't realize the difference in such a small table. If you had one million records, and you were actually counting by groups, you would see that SUM takes a couple of milliseconds more than COUNT, and I think we can live with that. Notice that we could not use COUNT in our cross−tab, because it would have counted all the rows anyway. Try it."

```
mysql> SELECT location, COUNT(IF(gender='M',1,0)) AS M,
COUNT(IF(gender='F',1,0)) AS F,
COUNT(*) AS total
FROM locations INNER JOIN employees USING (loc_code)
GROUP BY location;
```

(warning: gives WRONG results!)

location	M	F	total
Boston	3	3	3
Cagliari	3	3	3
London	2	2	2
Manchester	2	2	2
Marseille	3	3	3
Milano	2	2	2
New York	4	4	4
Paris	2	2	2
Roma	3	3	3

```
9 rows in set (0.00 sec)
```

"See? That's why we have to sum up, instead of counting. COUNT is a dumb function which will count any piece of junk it finds. SUM has some grace, in its choice."
It looked so trivial that I was ashamed of myself for not having found it alone.
But suddenly I saw something that didn't seem right to me. "Here we have a simple case, where we know all the values that will go into the columns. But what should we do if we don't know? What if we want the departments instead?"
The Wizard took a glance at the diagram and typed:

```
mysql> SELECT dept from departments;
```

dept
Development
Personnel
Research
Sales
Training

```
5 rows in set (0.01 sec)
```

"Yeah. I see." I said, with a hint of disappointment in my voice. "You mean that I have to compose the query manually, entering a SUM/IF statement for each value in departments?"

Some help from SQL itself

The Wizard smiled. "If you like to, you're welcome. However, you could get some help from the database engine itself, provided that you ask nicely." And while he was speaking he typed a very cryptic statement:

```
mysql> SELECT CONCAT(', SUM(IF(dept = "',dept,'", 1,0)) AS `',dept,'`')
FROM departments;
```

CONCAT(', SUM(IF(dept = "',dept,'", 1,0)) AS `',dept,'`')
, SUM(IF(dept = "Development", 1,0)) AS `Development`
, SUM(IF(dept = "Personnel", 1,0)) AS `Personnel`
, SUM(IF(dept = "Research", 1,0)) AS `Research`
, SUM(IF(dept = "Sales", 1,0)) AS `Sales`
, SUM(IF(dept = "Training", 1,0)) AS `Training`

```
5 rows in set (0.00 sec)
```

"You know," he went on, "you can also use SQL to produce SQL code. This is one of the cases. You have in front of you the list of columns that you should include in your query. Now, with some cut−and−paste, we could get the result you want. Here."

```
mysql> SELECT location
, SUM(IF(dept = "Development", 1,0)) AS `Development`
, SUM(IF(dept = "Personnel", 1,0)) AS `Personnel`
, SUM(IF(dept = "Research", 1,0)) AS `Research`
, SUM(IF(dept = "Sales", 1,0)) AS `Sales`
, SUM(IF(dept = "Training", 1,0)) AS `Training`
, COUNT(*) AS total
FROM locations INNER JOIN employees USING (loc_code)
INNER JOIN departments USING (dept_code)
GROUP BY location;
```

location	Development	Personnel	Research	Sales	Training	total
Boston	2	0	0	1	0	3
Cagliari	0	0	3	0	0	3
London	0	1	0	0	1	2
Manchester	0	0	0	2	0	2
Marseille	2	0	0	1	0	3
Milano	0	0	0	1	1	2
New York	3	0	0	1	0	4
Paris	1	0	0	1	0	2
Roma	0	1	0	1	1	3

```
9 rows in set (0.36 sec)
```

The wizard was now unstoppable. He had reached the stage where he simply couldn't help giving away his knowledge. "Before we go on," he said, taking possession of my keyboard, "let's see how this same method can do more than counting. Let's replace those 1s with a numeric field, and do real summing up."

```
mysql> SELECT location
, SUM(IF(dept = "Development", salary,0)) AS `Development`
, SUM(IF(dept = "Personnel", salary,0)) AS `Personnel`
, SUM(IF(dept = "Research", salary,0)) AS `Research`
, SUM(IF(dept = "Sales", salary,0)) AS `Sales`
, SUM(IF(dept = "Training", salary,0)) AS `Training`
, SUM(salary) AS total
FROM locations INNER JOIN employees USING (loc_code)
INNER JOIN departments USING (dept_code)
GROUP BY location;
```

location	Development	Personnel	Research	Sales	Training	total
Boston	11900	0	0	5950	0	17850
Cagliari	0	0	16800	0	0	16800
London	0	5700	0	0	5700	11400
Manchester	0	0	0	11550	0	11550
Marseille	11150	0	0	5800	0	16950
Milano	0	0	0	5550	4900	10450
New York	17850	0	0	6100	0	23950
Paris	5700	0	0	5400	0	11100
Roma	0	5000	0	5500	5100	15600

```
9 rows in set (0.00 sec)
```

"Don't forget to change also the total field. A simple SUM without IF, and your total is ready."

I was looking at the screen, which was showing what seemed to be the complete solution to my problem, but the wizard was shaking his head. "As a matter of fact," he was saying, there is something that we can improve here. We have two queries, in which we are reading the *departments* table. So we are reading it twice twice. The whole process could be improved, by querying for department code the first time, and omitting the join with departments the second time."

```
mysql> SELECT CONCAT(', SUM(IF(dept_code = "',dept_code,'", 1,0)) AS `',dept,'`')
FROM departments;
```

"Here. Let's get the columns once more. Good. And there it is. This one looks better."

```
SELECT location
, SUM(IF(dept_code = "1", 1,0)) AS `Personnel`
, SUM(IF(dept_code = "2", 1,0)) AS `Training`
, SUM(IF(dept_code = "3", 1,0)) AS `Research`
, SUM(IF(dept_code = "4", 1,0)) AS `Sales`
, SUM(IF(dept_code = "5", 1,0)) AS `Development`
, COUNT(*) AS total
FROM locations INNER JOIN employees USING (loc_code)
```

```
GROUP BY location;
```

He changed the previous two SQL statements, executed them, with some cut−and−pasting in the middle, and got exactly the same result. Now he was explaining me why he did it. "The first query is scanning all the departments table, and we know that its results will be used to build the second query containing the employees table, which has already a department code. Therefore, we can skip the join with departments, since the only purpose of that join was to get the names of the departments."
"I see" was the only comment I could offer, Since I was overwhelmed by his continuous insight. I got the idea, and I couldn't help thinking that he must have done that before. All those pieces of information were coming just too fast for me. Luckily, all logs were on, so I knew that I would be able to get all the statements back when he would leave. Which was not the case yet. The wizard was now ready to give me his philosophical view of cross tabulating.

The golden rules

"Rule number one:" −− wizards have always a rule n. 1 for everything −− "**cross−tabulating is a simple algorithm with a complex implementation**. Once you know the principle, you can do everything you need for your statistics. The difficult part is making the process automatic. You don't know in advance which values you should use for your columns. Hence, you have to find such values every time, before creating the second query. The real problem with cross−tabs is that **there are no simple cases.** Even the ones that look as such, have sticky problems. And you should consider this as rule number two."

I was about to lose heart, but he had more for me. "On the other hand," he continued, "This algorithm is so flexible that will let you do things that your ordinary spreadsheet won't let you. For example, let's suppose that you want a x−tab with the number of employees by gender and the total of their salaries as well. In your common graphical tool you can do either of them. With this system, you can combine both. It's no difficult, just complicated. Now, if you fetch some more coffee, I'll show you."
And when I came back from the kitchenette, there was it.

```
mysql> SELECT location
, SUM(IF(gender='M',1,0)) AS M
, SUM(IF(gender='M',salary,0)) AS salary_M
, SUM(IF(gender='F',1,0)) AS F
, SUM(IF(gender='F',salary,0)) AS salary_F
, COUNT(*) AS empl
, SUM(salary) AS tot_salary
FROM locations INNER JOIN employees USING (loc_code) GROUP BY location;
```

location	M	salary_M	F	salary_F	empl	tot_salary
Boston	1	5800	2	12050	3	17850
Cagliari	1	5600	2	11200	3	16800
London	1	5700	1	5700	2	11400
Manchester	1	5600	1	5950	2	11550
Marseille	1	5550	2	11400	3	16950
Milano	2	10450	0	0	2	10450
New York	2	11950	2	12000	4	23950
Paris	1	5400	1	5700	2	11100
Roma	3	15600	0	0	3	15600

```
9 rows in set (0.00 sec)
```

He took a sip from his mug and added, "And of course nobody could prevent you from inserting a totally unrelated line into your query, like this one:

```
mysql> SELECT location ,SUM(IF(gender='M',1,0)) AS M
, SUM(IF(gender='M',salary,0)) AS salary_M
, SUM(IF(gender='F',1,0)) AS F
, SUM(IF(gender='F',salary,0)) AS salary_F
, SUM(IF(dept_code = "4", 1,0)) AS `Sales`
, COUNT(*) AS empl
, SUM(salary) AS tot_salary FROM locations
INNER JOIN employees USING (loc_code) GROUP BY location;
```

location	M	salary_M	F	salary_F	Sales	empl	tot_salary
Boston	1	5800	2	12050	1	3	17850
Cagliari	1	5600	2	11200	0	3	16800
London	1	5700	1	5700	0	2	11400
Manchester	1	5600	1	5950	2	2	11550
Marseille	1	5550	2	11400	1	3	16950
Milano	2	10450	0	0	1	2	10450
New York	2	11950	2	12000	1	4	23950
Paris	1	5400	1	5700	1	2	11100
Roma	3	15600	0	0	1	3	15600

```
9 rows in set (0.00 sec)
```

"Here we have results coming from three different sources: the counting of employees by gender, the sum of their salaries by gender, and the total number of the employees in the Sales department, regardless of their gender. I know that this particular example doesn't make much sense, but this is something that people in the management, for reasons that completely escape my intellectual hold, want to put together. Don't underestimate such possibility, which could prove to be useful in many occasions."

I knew by then that his enthusiasm was growing thinner, and I should have expected him to leave abruptly any moment. I had to refuel his good disposition and ask some juicy questions. "This is really wonderful," I said, "but in a real application you would not use SQL alone. How can I implement this algorithm with a general purpose language?"

Putting the pieces together

The wizard startled, as if awaken from a troubled sleep and suddenly his eyes were sparkling with renewed interest. "Yes, of course," he replied. "If we stick to one−level cross−tabulations, this algorithm will fit nicely into any high level language. We should avoid the multi−level ones, for the moment, because the general concept is more important than the thorny details. Let me draw you a flow−chart diagram."
I had already relinquished my desktop, and I stayed quietly at his side, while he was skillfully drawing this diagram, which he explained almost as fast as he was designing.

"The required actions are quite simple. However, you need some planning before starting. First, you have to identify the source for the columns. It could be the same source from where you need to count or sum, but in a well organized and normalized database it should be in a separated table. Either way, your first action with the database will be to query for distinct column values. Then you will merge such values within the summary statements composing your query. And at that point you'll be ready to execute. You can only omit the initial query if you are 100% sure that your values have not changed. This could be the case, for example, if your column values are in a read−only table. But usually this is not the case, or else cross−tabulations wouldn't be that hard. Well, let's start. Any preferred language?"

This was a false democratic question, which I knew by experience. I could mention any language and he would be proficient in it, but he would reject on some ground, until I would eventually manage to name the language he had in mind. Having gone through the motions before, I had my answer ready: "I think that Perl would serve the purpose," I said hastily. He nodded, approving my wisdom, and fired a copy of vim from a xterm. "Sorry if you are an Emacs guy," but his voice betrayed his complete lack of sorrow, "Nothing like vim to highlight Perl syntax. I am sure you can follow me into this, and besides, I have the helm." That, I knew for sure. My computer was firmly in his possession, and I started doubting that I would ever have it back.

The wizard, unaware of my anxiety, was already writing

```
#!/usr/bin/perl −w


use DBI;
use strict;
```

(note: You can check here the full script, with plenty of comments)

Now, you would expect your average wizard to be adventurous and careless, keen of programming without constraints. Not this one. My wizard is a wizard because he behaves like one, but, being a database wizard, he is also very strict in matter of coding. More than once he gave me a speech on the importance of catching the errors in advance, before they can catch you. Since I knew that lecture by heart, I did not comment. My only contribution was to provide the IP address of my server and the location where the DBI could look for my user−name and password. I suspect that the wizard already knew that much, but he wanted to make me feel important by letting me provide some tiny contribution to the script.

```
my %params;


while (<>) { # gets the configuration from the input pipe
next if /^#/; # skips comments
my ($name,$value) = split '=',$_;
chomp $value;
$params{$name} = $value ;
}
$params{database} or die "no valid input given\n";
```

"We are going to read the parameters into a hash from a configuration file. They are too many for the command line, and it's better to save them to a file, which will be even clearer. In a production case, you would rather store those parameters into a database table. Four our purposes, we are going to use a text file. Actually, let's write it down, before we continue. I think it would be better to start with the simple case, location by gender."

```
title=location−gender
database=xcompany
row_name=location
row_alias=town
col_name=gender
col_alias=gender
col_value=1
col_from=FROM employees
col_where=
col_order=
row_from=FROM locations INNER JOIN employees USING(loc_code)
row_where=
row_order=ORDER BY loc_sort_order
row_group=group by location
```

He saved those parameters into a location−gender−count.xtab and explained: "See, our parameters are the description of the final query. In order to build the query, we need to set its parts. In this particular case, we are not going to use a WHERE and a ORDER BY clause for the columns, but you know that we may need them for the others, so we'll leave them. Our parsing mechanism can easily take care of the empty strings. In this script I assume that all these parameters are properly set in the configuration file. I will skip all the error checking on that, and you can implement it later on."

```
my $dbh = DBI−>connect("DBI:mysql:$params{database};"
."host=172.16.35.1;mysql_read_default_file=$ENV{HOME}/.my.cnf")
or die "can't connect $! \n";
```

"This is fairly simple. Just the connection to the database. Now we have a database handle $dbh and with it we can go for the first query."

```
my $row_statement = "";

my $sth = $dbh−>prepare("SELECT DISTINCT $params{col_name},
$params{col_alias} "
. $params{col_from} . " " . $params{col_where} .
$params{col_order});

$sth−>execute();

while (my ($colname, $colalias) = $sth−>fetchrow_array()) {
```

```
$row_statement .= ", SUM(IF($params{col_name} =
\"${colname}\","
. " $params{col_value}, 0)) AS `${colalias}` \n"

}

$sth−>finish();
$dbh−>disconnect();
```

I recognized the proceedings. He was instructing Perl to do the same thing that he was previously doing manually. I told him that much. "Yes, exactly," he approved. "And in addition to that, using a high level language will grant us some more freedom. Since we have already collected the field list from our configuration file, we are not in a hurry to compose our second query. Having the parameters in a hash will also give us some amount of control, since any missing parameter from the configuration file will be duly reported by the Perl compiler. At this point, the only thing we need to do is to gather all the pieces together and produce our final query."

```
# ------------ add total column and row details

$row_statement .= ", " . (($params{col_value} eq "1")?
"COUNT(*)" : "SUM($params{col_value})")
. " AS total \n"
. $params{row_from}
. $params{row_where} ;

print "use $params{database};\n";

$row_statement =~ s/\n\n/\n/g; # remove double EOL

print "SELECT $params{row_name} AS $params{row_alias} \n",
"$row_statement\n$params{row_group} \n $params{row_order}
;\n"; # the cross table

print "SELECT 'TOTAL'\n$row_statement;\n"; # the total line
```

He continued explaining. "Since you have seen the manual examples and the flow−chart, what we are doing shouldn't come as a surprise to you. We add a total line, which will be a real SUM or a COUNT, depending on the parameter that was entered. Then we add the FROM and WHERE clause, and we don't care if this last one is empty. The resulting script will include a USE database statement, followed by a query that should look much like the one we entered manually with cut−and−paste."
I wanted desperately to enter the discussion, so I told him what was troubling me. "Why you did not complete the $row_statement with the GROUP and ORDER−BY clauses? Why did you stop at the WHERE level?"
Almost annoyed by my lack of insight, he replied "You should see it by yourself. The $row_statement as I made it is what we have in common between the normal selection and the total line. To get the grand total you

should not GROUP, and since we are going to get only one line, it doesn't make much sense to ORDER it. Does it? Now, since you mentioned the total, this solution is just for a quick demonstration, which I can do with a single line of Perl. If your tables are those big babies with millions of record that you might have in your data warehouse, then you'd better send the output of this query to a temporary table and then get the grand total from there. For tables with less than one hundred thousand records, MySQL will slurp this kind of cross−tabs in a blink."

He got up from my chair, and I knew that my lesson was over. "Wait a minute." I said hastily. " This script doesn't execute the cross−tab query. How do you use it?"

Still halfway from my chair, he bent to the keyboard, and typed

```
$ perl xtab.pl < location−gender−count.xtab | mysql −t
```

"Just add your password and host, if you need it, and it will give you the X−tab."

"But? wait! I have a few questions about multi−level cross−tabulations."
He smiled in a devilish way. "I am sure you do. But I have a girlfriend who wants to discuss database theory with me," he looked at his watch and added "in exactly ten minutes at that new Chinese restaurant downtown, and I shouldn't make her wait. I think you have enough food for thought to fill your weekend. Give me a call next week, and perhaps, if I can spare half an hour, I can give you a hand." And while offering his metaphoric limb, he extended his real one to shake mine, and off he went before I could say 'son of a wizard!'

Conclusion

DISCLAIMER

The Wizard is a fictitious character. He does not exist in the wild, although some laboratories in Outer China are collecting evidence about the theoretical possibility that he could be real. Any resemblance with any individual in the physical world is purely coincidental. Translated into plain English: if you think that the Wizard looks like somebody you know, especially if he or she is a pain in the neck, it is because you are unlucky. Wizards are grumbling fellows who call Perl or SQL their mother tongue and feel ill at ease while speaking English (or Italian, Hungarian, Flemish or whichever language they use d to speak before learning C++. They are usually kind and willing to please you, but they often manage to upset you while doing so. Wizards also enjoy gathering with others of their kind, since they find common people boring. However, if you put two wizards in the same room, you are likely to see some sparks after a while. That's why wizards are usually alone, and they like to be left as such.

ACKNOWLEDGEMENT

I want to thank many people who made this article possible. All the programmers who created GNU, Linux, MySQL, StarOffice, perl, DBI, nedit (I am not the Wizard. I can't cope with vi!), and many utilities without which I would be most likely writing about embedded databases in C. (There is nothing wrong about embedding databases, and doing that in C is quite honorable, since I have done that for a while, but MySQL is a toy on a totally different scale). In a word, I am grateful to all the open source community, which has made profitable exchanging knowledge (not to mention the fun!).

RESOURCES

You must have realized that I (or perhaps the Wizard) have taken for granted that you know the basics of MySQL, SQL, client server architecture, perl, the DBI, and maybe something else that I don't recall now. Don't be angry with me. Teaching the basics is a social duty, but sometimes it makes you feel good when you can exchange some unusual experience. You might be interested in a few links that offer the basics that I have so happily skipped: http://www.mysql.com/documentation One of the best manuals you can find about MySQL, the one written by the developers themselves. In addition, http://www.devshed.com offers many tutorials and articles for beginners (and intermediate and expert users as well). DBI can be learned online at http://www.symbolstone.org/technology/perl/DBI About Perl, I don't dare suggesting anything that could contravene the official policy that TIMTOWTDI (there is more than one way to do it). Larry Wall has written some thousand pages on this subject, with the conclusion that Perl's official policy is that there is no official policy. I would leave it at that.

SOURCE CODE

To let you appreciate the examples in this article, I prepared some scripts: The Xcompany dump file, when fed into MySQL, will re−create the database for you. The xtab.pl script, general purpose cross−tabulation builder, will let you feel the thrill first hand.
Some **configuration files** for xtab.pl will spare you the boredom of re−typing.
location−gender−count.xtab is the simplest one. Counting of employees by ocations and gender.

age−gender−count.xtab will count the employees by age range (using MySQL "WHEN" instruction) and gender.
country−dept−salary−sum.xtab sums the salaries by country and department.
month−category−sales−sum.xtab will summarize monthly sales by category.

Now, please. Don't let me do too much. Try it on your databases, and tell me how it worked.