Rémy LENTZNER

Getting started with SQL

Exercises with PhpMyAdmin and MySQL

PROFESSIONAL TRAINING
ÉDITIONS REMYLENT

REMY LENTZNER

Getting started with SQL

French original title : Bien débuter avec SQL
EDITIONS REMYLENT, Paris, 1ère édition, 2023
R.C.S. 399 397 892 Paris
25 rue de la Tour d'Auvergne - 75009 Paris
REMYLENT@GMAIL.COM
www.REMYLENT.FR

ISBN EPUB : 978-2-38518-037-9

This book is dedicated to Anna and Tama

I could not have written it without their support, advice, encouragements and proofreading.

Graphic illustration : Anna LENTZNER

Getting started with Adobe Acrobat Pro

Getting started with Sparkle

INTRODUCTION

Numerous business people use spreadsheets to enter numbers and formulas. These numerical data are placed in cells within a group of columns and rows. When it comes to manipulating values, a spreadsheet is the ideal tool. But the amount of stored information will be limited by the maximum number of rows allowed. For instance, Excel Mac version 16 offers you 1,048,676 rows and moreover 10,000 columns. This may seem like a lot, but if you have to manage millions of parts like ship or spacecraft manufacturers, one spreadsheet won't be enough.

Applications that enable you to store huge amounts of information are called databases. These are computer tools that manage information stored in many tables and linked together by relationships. In this way, data can be pooled, retrieved, queried or added to. The overall system is designed to secure the information. For instance, a bank manages all its customers by assigning them with an individual user number and a password. Each customer can consult his account, withdraw funds, make transfers and perform many other operations. Thanks to the Internet network from his phone, banking operations can be carried out at home, at the office or while travelling. Each operation is considered as a transaction that can succeed or fail. To manage these countless transactions, the computer application must be able to provide an extremely high degree of security. This is the purpose of a database that links numerous pieces of information and is called a RDBMS or Relational Database Management System.

There are several of them in the market such as Oracle, Sybase, SQL server, MySQL, PostgreSQL and many others.The common language used by all these information management applications is SQL (Structured Query Language). It has evolved over time with the development of operating systems.

SQL is platform independent and has a small instruction set. In 1970, the professor and mathematician Edgar Frank Codd (1924-2003) was working as a researcher for the IBM company at the San Jose laboratory. He wrote a theoretical paper (*A relational model of data for large shared data banks*) published in the Association for Computing Machinery journal. The article laid the foundation for a simple programming language in English that could handle data stored on any computer platform.

A few years later, IBM created a prototype for a relational database called System/R with SEQUEL or Structured English Query Language, which was later renamed SQL. It is the Oracle Corp. that produced the first marketable version.

SQL is a standardized language that has evolved over time: SQL86, SQL89, SQL92, SQL1999, SQL2003, SQL2006, SQL2008, SQL2011 and SQL2016. It has adapted to Internet applications that manage huge volumes of information.

The SQL language is based on relational algebra and is broken down into different subsets: DDL (Data Definition Language), DML (Data Manipulation Language), DCL (Data Control Language) and TCL (Transaction Control Language).

The DDL or Data Definition Language groups together the commands that allow the creation, modification or deletion of table structures, indexes, views, etc.

A table is the physical information support and the index is a feature that accelerates searches and permits relations between tables. The view enables the extraction of data in a flexible and efficient way.

The DML or Data Manipulation Language contains the commands that are used to select and manipulate the data contained in the database (SELECT, INSERT, DELETE, UPDATE, etc).

The DCL or Data Control Language contains the orders that manage the security of data access, such as GRANT or REVOKE.

The TCL or Transaction Control Language groups the commands that manage the validation (or not) of the transactions, such as COMMIT, or ROLLBACK.

It would be tedious to describe all the SQL commands one after the other. In my opinion, it is more interesting to discover the most used commands by doing progressive exercises.

In this book, you will practice with the free MySQL language accessible to everyone and thanks to the PhpMyAdmin database management environment, you will be able to manipulate data. I have chosen the free local MAMP server which you can download on PC or Mac.

The book is divided into 6 chapters.

Chapter 1 describes how to install the local MAMP server, the PhpMyAdmin relational database management feature and how to create a database with its different objects.

Chapter 2 shows how to create a table, enter data, search for information, and query it using advanced criteria or groupings. You will learn how to use the SELECT command, that enables you to select information from one or more tables. With the UPDATE command, you will perform updates and calculations. You will learn how to create views, stored procedures and triggers.

Chapter 3 explains the techniques that allow you to define relationships between tables. You will be able to define foreign keys and constraints.

Chapter 4 focuses on functions that manipulate text, dates, or numbers and can be used in SELECT queries. You will learn about the aggregate functions that are used with the GROUP BY clause.

Chapter 5 looks at more advanced SQL queries. You will discover how to integrate a SELECT command into another SELECT command. You will learn how to sort in substrings. You will manipulate several set queries with the UNION operator as well as the left and right outer joins between a primary key and a foreign key from two tables.

A glossary summarizes the principal key words of the language

I hope that this book will interest you and enhance your SQL knowledge.

Do not hesitate to contact me at remylent@gmail.com if you have any comments or questions about this book.

Enjoy your reading.

The author

TABLE OF CONTENTS

Chapter 1
The working environment

To work with SQL, you must have a database environment that enables you to create tables, insert or modify information and run SQL queries. A database server is therefore necessary whether it is local or on an Internet platform.

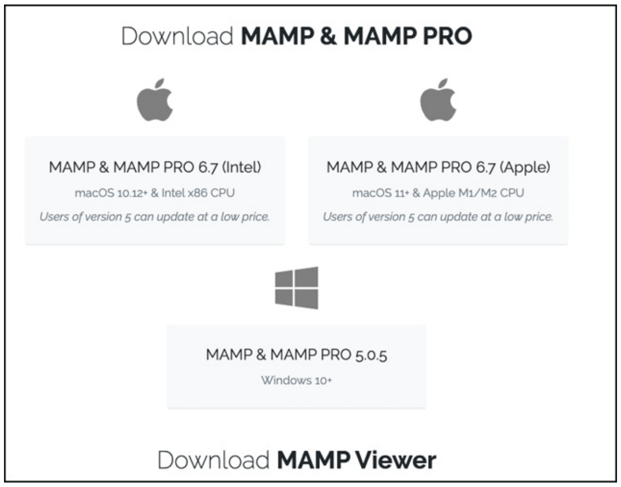
This chapter shows how to use the local and free MAMP server, where you can work with the relational database system PhpMyAdmin in order to manipulate SQL.

Once the local server is ready to use, you can create a database, tables and other objects.

1.1 Installing the local MAMP server

In your favorite browser, search for MAMP then select the link *Downloads - MAMP & MAMP PRO*.

Figure 1.1 shows the different platforms that are available to you.



Download **MAMP & MAMP PRO**

MAMP & MAMP PRO 6.7 (Intel)
macOS 10.12+ & Intel x86 CPU
Users of version 5 can update at a low price.

MAMP & MAMP PRO 6.7 (Apple)
macOS 11+ & Apple M1/M2 CPU
Users of version 5 can update at a low price.

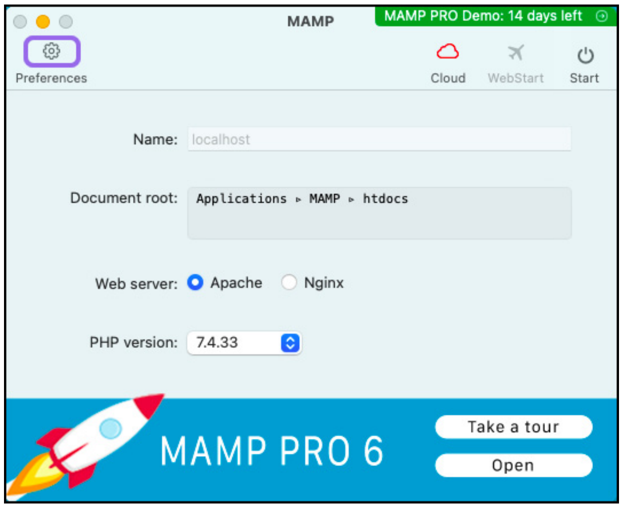
MAMP & MAMP PRO 5.0.5
Windows 10+

Download **MAMP Viewer**

Figure 1.1 : MAMP platforms

- Click on the needed MAMP link.
- Choose your platform, then install the package on your computer.

Figure 1.2 shows the MAMP application.



Name: localhost

Document root: Applications ▸ MAMP ▸ htdocs

Web server: [✓ Apache] Nginx

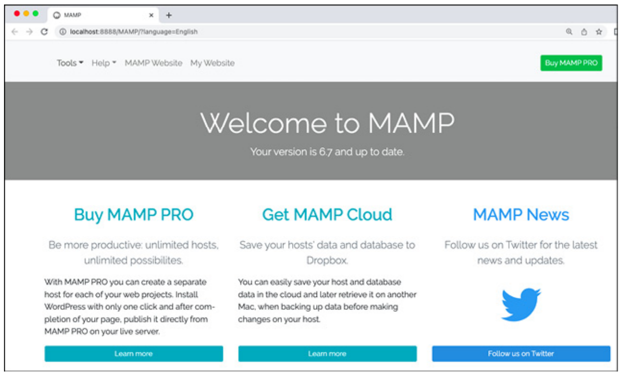
PHP version: 7.4.33

Figure 1.2 : MAMP is ready to start

- To start the local server, click on the *Start* icon. To stop the server, click on the *Stop* icon.

The Open WebStart page link will enable you to run the PhpMyAdmin database manager.

Figure 1.3 shows the different tools that are available at the address : http://localhost:8888/MAMP/?language=English



Welcome to MAMP

Your version is 6.7 and up to date.

Buy MAMP PRO

Be more productive: unlimited hosts, unlimited possibilites.

With MAMP PRO you can create a separate host for each of your web projects. Install WordPress with only one click and after completion of your page, publish it directly from MAMP PRO on your live server.

Get MAMP Cloud

Save your hosts' data and database to Dropbox.

You can easily save your host and database data in the cloud and later retrieve it on another Mac, when backing up data before making changes on your host.

MAMP News

Follow us on Twitter for the latest news and updates.

Figure 1.3 : MAMP tools

To create a database, use the *phpMyAdmin* tool.

1.2 Starting the phpMyAdmin tool

- Click on *Tools / phpMyAdmin*.

Figure 1.4 shows the *Tools* menu.

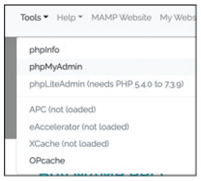
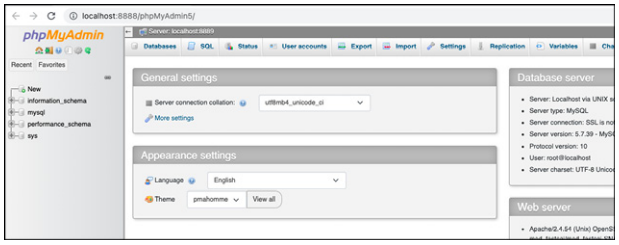


Figure 1.4 : The Tools menu

Once phpMyAdmin is running, the window is split into 3 parts (figure 1.5).

In this working environment, you will be able to create your databases.



Server connection collation: utf8mb4_unicode_ci

Language: English

Theme: pmahomme

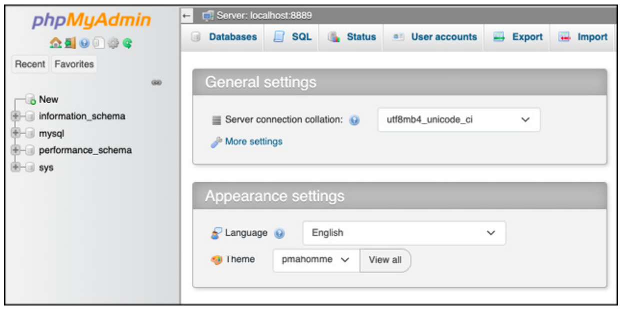
- Server: Localhost via UNIX s
- Server type: MySQL
- Server connection: SSL is no
- Server version: 5.7.39 - MyS
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unico

- Apache/2.4.54 (Unix) OpenS

Figure 1.5 : The SQL working environment

The window has two important parts: the left panel for the objects and the central panel for their properties. Notice the *home* icon at the top of the left panel.

By clicking on this *home* icon, you can change the display language, thanks to the drop-down list in the *Language* area.



Server connection collation: utf8mb4_unicode_ci

Language: English

Theme: pmahomme

Figure 1.6 : Changing the language

The other icons enable you to get information about phpMyAdmin and to set other parameters concerning the databases.

The central panel displays tabs that will allow you to manipulate the database objects (figure 1.7).

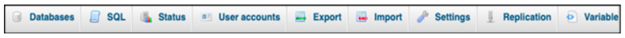
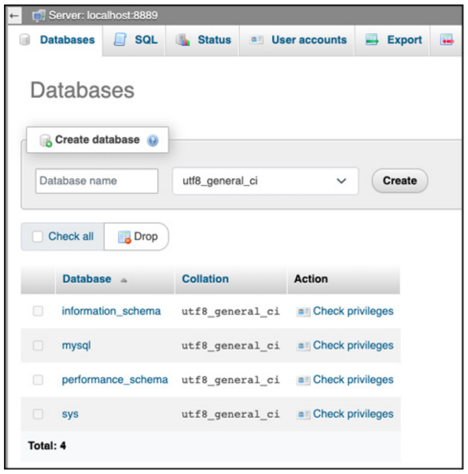


Figure 1.7 : The objects menu

The following list details these options:

- *Database*. Here you can create new databases.

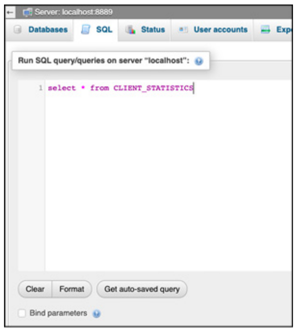


Database	Collation	Action
information_schema	utf8_general_ci	Check privileges
mysql	utf8_general_ci	Check privileges
performance_schema	utf8_general_ci	Check privileges
sys	utf8_general_ci	Check privileges

Total: 4

Figure 1.8 : Creating a new database

- *SQL*. This window allows you to write SQL queries. With phpMyAdmin, you can see the SQL code that prefigures any action.

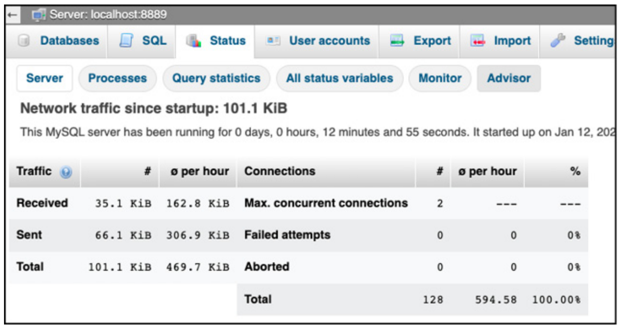


Run SQL query/queries on server "localhost":

```sql
select * from CLIENT_STATISTICS
```

Figure 1.9 : Writing the SQL code

- *Status*. This feature displays general information about the started server and the running processes, as well as the status variables. Other actions are possible to monitor the database management.



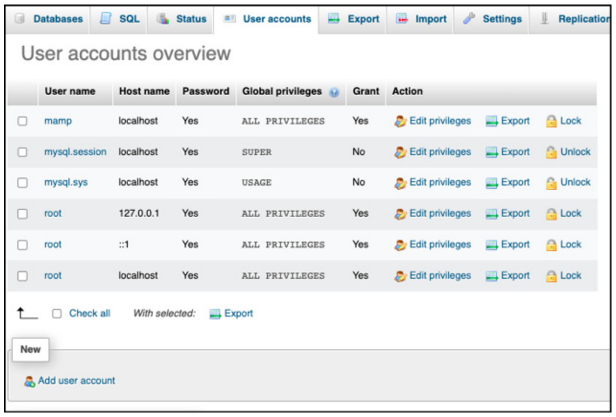
Network traffic since startup: 101.1 KiB

This MySQL server has been running for 0 days, 0 hours, 12 minutes and 55 seconds. It started up on Jan 12, 202

Traffic	#	ø per hour	Connections	#	ø per hour	%
Received	35.1 KiB	162.8 KiB	Max. concurrent connections	2	---	---
Sent	66.1 KiB	306.9 KiB	Failed attempts	0	0	0%
Total	101.1 KiB	469.7 KiB	Aborted	0	0	0%
			Total	128	594.58	100.00%

Figure 1.10 : The current status information

- *User account*. This option displays user names, host names, passwords and privileges. You can create a new user in this area.

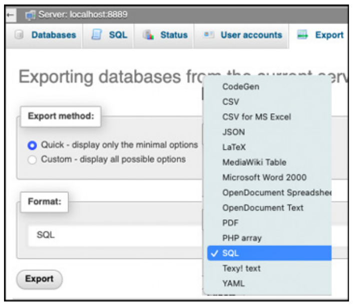


User accounts overview

	User name	Host name	Password	Global privileges	Grant	Action		
☐	mamp	localhost	Yes	ALL PRIVILEGES	Yes	Edit privileges	Export	Lock
☐	mysql.session	localhost	Yes	SUPER	No	Edit privileges	Export	Unlock
☐	mysql.sys	localhost	Yes	USAGE	No	Edit privileges	Export	Unlock
☐	root	127.0.0.1	Yes	ALL PRIVILEGES	Yes	Edit privileges	Export	Lock
☐	root	::1	Yes	ALL PRIVILEGES	Yes	Edit privileges	Export	Lock
☐	root	localhost	Yes	ALL PRIVILEGES	Yes	Edit privileges	Export	Lock

Figure 1.11 : User accounts overview

- *Export*. This feature enables you to export a database in different formats.



Server: localhost:8889

Exporting databases from the current serv

Export method:

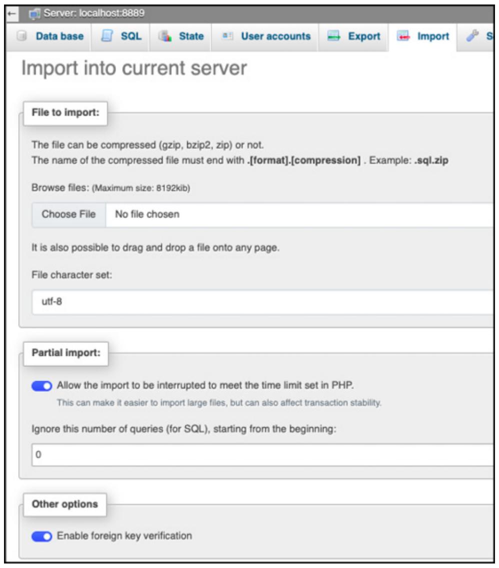
[✓ Quick - display only the minimal options]
Custom - display all possible options

Format:

SQL

Figure 1.12 : Exporting a database

- *Import*. This option permits you to import a file into the current database.



Server: localhost:8889

Import into current server

File to import:

The file can be compressed (gzip, bzip2, zip) or not.
The name of the compressed file must end with **.[format].[compression]** . Example: **.sql.zip**

Browse files: (Maximum size: 8192kib)

Choose File No file chosen

It is also possible to drag and drop a file onto any page.

File character set:

utf-8

Partial import:

[✓ Allow the import to be interrupted to meet the time limit set in PHP.]
This can make it easier to import large files, but can also affect transaction stability.

Ignore this number of queries (for SQL), starting from the beginning:

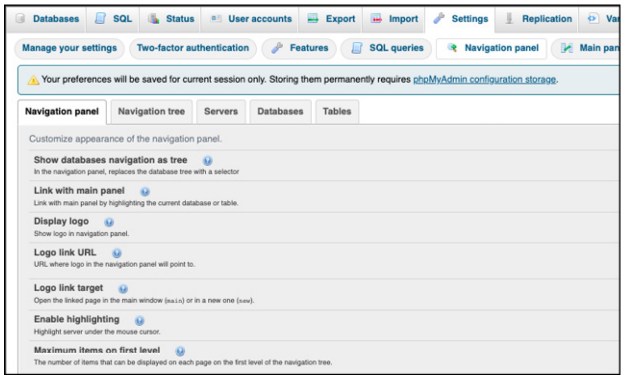
0

Other options

[✓ Enable foreign key verification]

Figure 1.13 : Importing a file into a database

- *Settings*. You can change the general settings of the system, for instance, by redefining the dialog display.



Your preferences will be saved for current session only. Storing them permanently requires phpMyAdmin configuration storage.

Customize appearance of the navigation panel.

Show databases navigation as tree
In the navigation panel, replaces the database tree with a selector

Link with main panel
Link with main panel by highlighting the current database or table.

Display logo
Show logo in navigation panel.

Logo link URL
URL where logo in the navigation panel will point to.

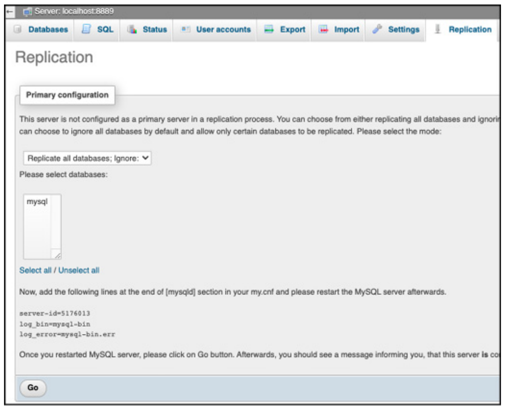
Logo link target
Open the linked page in the main window (`main`) or in a new one (`new`).

Enable highlighting
Highlight server under the mouse cursor.

Maximum items on first level
The number of items that can be displayed on each page on the first level of the navigation tree.

Figure 1.14 : Managing settings

- *Replication*. This feature allows you to define database replication settings. It is a process for copying, storing and saving data between a master and a target database.



Replication

This server is not configured as a primary server in a replication process. You can choose from either replicating all databases and ignori
can choose to ignore all databases by default and allow only certain databases to be replicated. Please select the mode:

Replicate all databases; Ignore:

Please select databases:

mysql

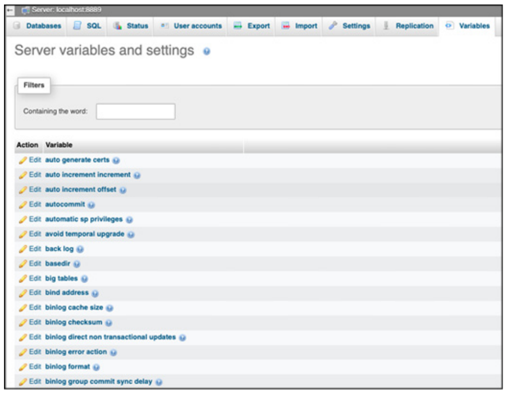
Now, add the following lines at the end of [mysqld] section in your my.cnf and please restart the MySQL server afterwards.

```
server-id=5176013
log_bin=mysql-bin
log_error=mysql-bin.err
```

Once you restarted MySQL server, please click on Go button. Afterwards, you should see a message informing you, that this server **is** co

Figure 1.15 : Replication settings

- *Variables*. Displays the contents of all system status variables.



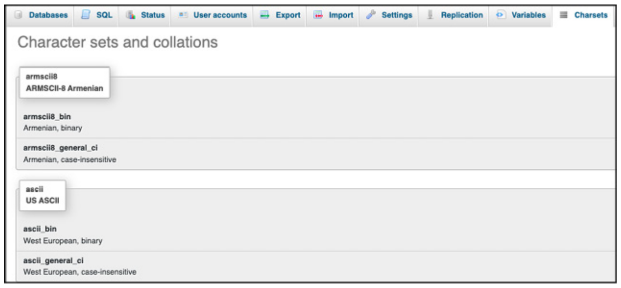
Server variables and settings

Containing the word:

Action	Variable
Edit	auto generate certs
Edit	auto increment increment
Edit	auto increment offset
Edit	autocommit
Edit	automatic sp privileges
Edit	avoid temporal upgrade
Edit	back log
Edit	basedir
Edit	big tables
Edit	bind address
Edit	binlog cache size
Edit	binlog checksum
Edit	binlog direct non transactional updates
Edit	binlog error action
Edit	binlog format
Edit	binlog group commit sync delay

Figure 1.16 : The system variables

- *Charsets*. You can define a character set from an international set.



Character sets and collations

armscii8
ARMSCII-8 Armenian

armscii8_bin
Armenian, binary

armscii8_general_ci
Armenian, case-insensitive

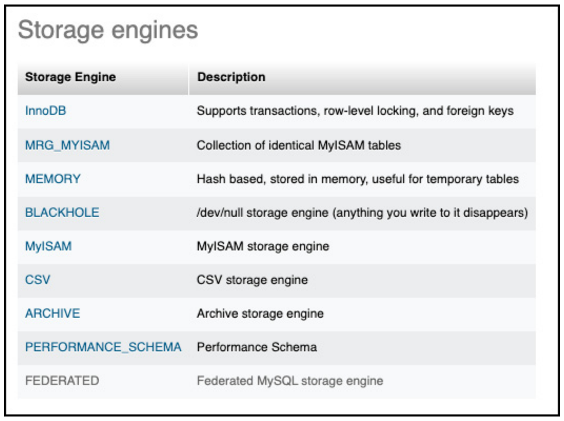
ascii
US ASCII

ascii_bin
West European, binary

ascii_general_ci
West European, case-insensitive

Figure 1.17 : Choosing the charsets

- *Storage engine*. Several internal storage programs are available.



Storage engines

Storage Engine	Description
InnoDB	Supports transactions, row-level locking, and foreign keys
MRG_MYISAM	Collection of identical MyISAM tables
MEMORY	Hash based, stored in memory, useful for temporary tables
BLACKHOLE	/dev/null storage engine (anything you write to it disappears)
MyISAM	MyISAM storage engine
CSV	CSV storage engine
ARCHIVE	Archive storage engine
PERFORMANCE_SCHEMA	Performance Schema
FEDERATED	Federated MySQL storage engine

Figure 1.18 : The storage engines

1.3 Creating a database

Using the PhpMyAdmin system, the following shows how to create a database called MANAGEMENT:

- Start the MAMP server or your own server.
- Go to the phpMyAdmin utility.
- Click on the *Databases* tab.

- Enter the word MANAGEMENT in the *Database* area then click on the *Create* button (figure 1.19).

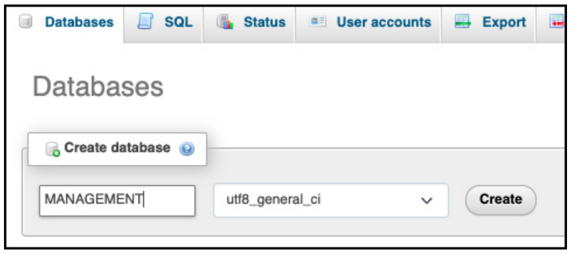


Figure 1.19 : Creating a database

As soon as the database is created, the system offers you to create a new table with 4 columns by default (figure 1.20).

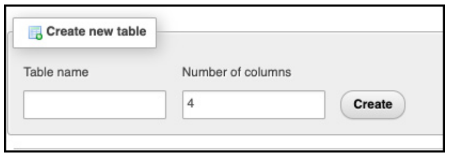


Figure 1.20 : Waiting for the creation of tables

A database is a structure that permits the storage of large amounts of information thanks to an organization in the form of tables.

The tables are structured in columns which are also called fields. Tables have numerous properties or *attributes*. If the information is well structured, the tables can be linked together by relations or special joins. This relational feature enables you to link a large number of tables in order to extract records based on several criteria.

Figure 1.21 shows the creation of a table name *FLOWERS* with 2 fields.

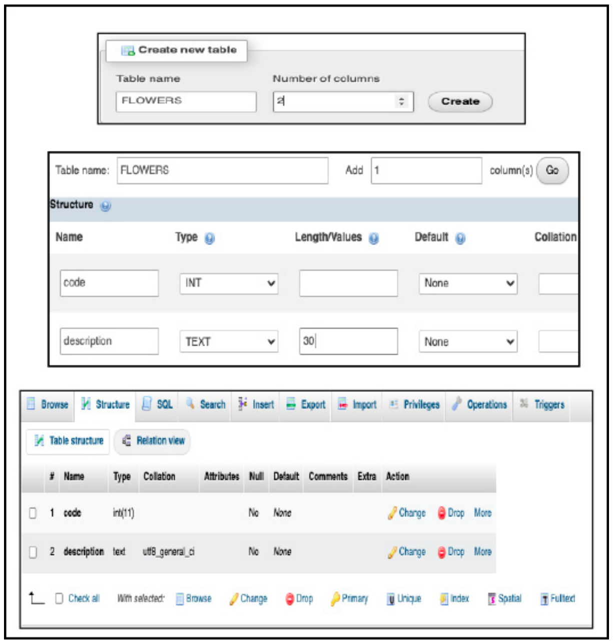


Table name: FLOWERS
Number of columns: 2

Table name: FLOWERS Add 1 column(s)

Name	Type	Length/Values	Default	Collation
code	INT		None	
description	TEXT	30	None	

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	code	int(11)			No	None			Change Drop More
2	description	text	utf8_general_ci		No	None			Change Drop More

Figure 1.21 : Creating a table with 2 fields

Once the table has been created, you can add data in it. The SQL tab enables you to view the SQL code created for each action you define afterwards.

For example, the following code creates a database named MANAGEMENT:

CREATE DATABASE IF NOT EXISTS MANAGEMENT

or

CREATE DATABASE MANAGEMENT

In brief

To work with a database, you need a dedicated server: local or remote. The phpMyAdmin database system enables you to manipulate and create as many databases and tables as you want. Many system parameters can be modified within this organization.

Chapter 2 will deal with the table structure.

Chapter 2
Tables

This chapter shows how to create tables, columns and primary/foreign keys that are fundamental inside the structure. You will be able to enter and modify data while studying the provided SQL code. You will discover the SELECT and UPDATE commands that respectively enable you to display data with criteria and modify information in depth. You will probably be interested in the views feature and stored procedures which permit to save actions in order to re-use them if necessary.

2.1 The table, the fields and the primary key

An information system defines data that is organized in a structured way. The information can be manipulated simply thanks to the SQL language, which has a small number of instructions. With it, you can insert, modify or delete data. Calculations can also be performed. The result of an SQL query is a result set or a view that can be saved for later use.

2.1.1 Creating a table

A table is a group of columns or fields. A set of rows is called records or tuples. You can create as many tables as you want in the database, the only limit is the storage capacity of the database which depends on the system used.

A table has fields of different types. For instance, you can specify numerical, text or date fields, and within these categories several subtypes are available.

Figure 2.1 shows these different types.

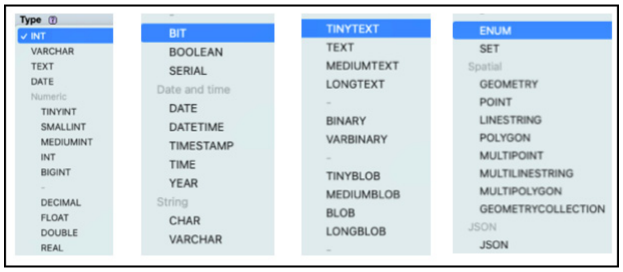


Figure 2.1 : Field types

The following list details the main types:

- *VARCHAR*. You can enter characters up to the number you define as a limit.
- *TEXT*. You can enter text.
- *CHAR*. In this type of column, you have to enter exactly the defined number of characters.
- *DATE*. It is a date type field.
- *TIMESTAMP*. This column type enables you to enter both a date and a time.
- *TINYINT*. A tiny integer is therefore a small positive integer with values from 0 to 255.
- *SMALLINT*. In this type of column, you can enter long integers in the range -2^{15} to 2^{15-1}.
- *INT*. In this type of column, you can enter long integers in the range -2^{31} to 2^{31-1}.
- *BIGINT*. It is a larger positive or negative integer with values from -2^{63} to 2^{62}.
- *BLOB*. It is a field that is used to store pictures or sound files.
- *BINARY*. This type of column enables you to define values between 0 and 1. It is often used to manage checkboxes.
- *DECIMAL*. This field allows you to enter large numbers with a precision of (+/-)231.
- *REAL*. In this type of column you can enter numbers in the

range -2^{1074} à $(2-2^{52})*2^{1023}$.

- *FLOAT*. Use it if you have memory constraint because it takes almost half as much space as double.
- DOUBLE. Use it if you need more precise and accurate results.

2.1.2 Workshop

Consider a book publisher that works with booksellers. He needs to have a list of both customers and booksellers.

To perform a good data organization, some rules must be respected:

- Each bookseller must have a unique identifier called the primary key that is defined when the table is created.
- Each column must contain only consistent data. For instance, in an address column, do not put either city or phone number.

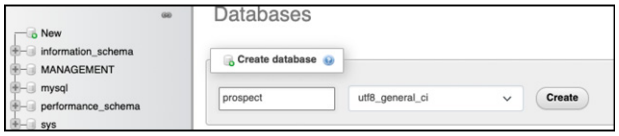
Usually a table can have millions of rows but each of them is identified by a primary key field. It can be of text or integer type. It is more practical to choose the *integer* type with the *auto-increment* feature because the value of the primary key will be increased by 1 each time a new record is created. This auto-incrementing property is defined at the time of the table creation. The primary key is very important because it also enables you to join data between several tables.

The table below shows the structure of a *bookstore* table.

Attribute	Description	Property
id_bookst	int	auto-increment primary key
name	text	40
address	text	40
zip	text	10
city	text	40
discount	decimal	
date_contact	date	

The following procedure shows how to create the *bookstore* table according to this structure:

- Start the server.
- Access *PhpMyAdmin*.
- *New / Create Database*
- Enter the name of the database (*prospect*) then click on the *Create* button (figure 2.2).



prospect
utf8_general_ci

Figure 2.2 : Creating the database.

- Click on the *Create New table* button.
- Type the name of the table (*bookstore*) in the *Name table* field.
- Enter the number of columns (7) in the *Number of columns* field.
- Finish with the *Create* button (figure 2.3).

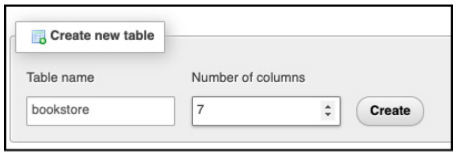


Table name: bookstore
Number of columns: 7

Figure 2.3 : Creating the bookstore table

The system displays the columns to be defined.

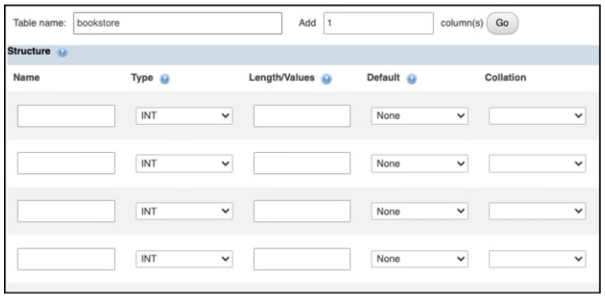


Table name: bookstore Add 1 column(s)

Name	Type	Length/Values	Default	Collation
	INT		None	
	INT		None	
	INT		None	
	INT		None	

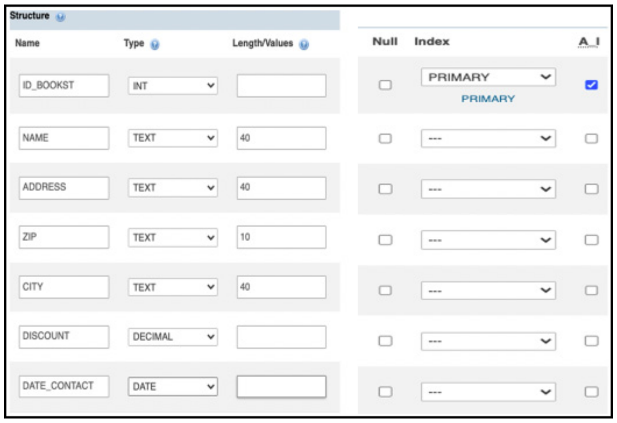
Figure 2.4 : The fields to define

Notice the column *A_I* which indicates that the field will be *auto-*

incremented.

Figure 2.5 shows the different field names. Once the fields are filled in, finish with the *Save* button.

The first field is the primary key. It is a good idea to specify it with an INT type and auto-incremented. Thus, each time you will create a new record, this field will be incremented and the unicity will be set without error.


Structure

Name	Type	Length/Values	Null	Index	A_I
ID_BOOKST	INT		☐	PRIMARY PRIMARY	✓
NAME	TEXT	40	☐	---	☐
ADDRESS	TEXT	40	☐	---	☐
ZIP	TEXT	10	☐	---	☐
CITY	TEXT	40	☐	---	☐
DISCOUNT	DECIMAL		☐	---	☐
DATE_CONTACT	DATE		☐	---	☐

Figure 2.5 : The different fields

Caution: The *Null* checkbox enables you to leave a field without information when editing the record.

Note the *SQL* Preview button at the bottom left of the window. It allows you to see the SQL code that creates the table.

CREATE TABLE `PROSPECT`.`BOOKSTORE` (`ID_BOOKST` INT NOT NULL AUTO_INCREMENT , `NAME` TEXT NOT NULL , `ADDRESS` TEXT NOT NULL , `ZIP` TEXT NOT NULL , `CITY` TEXT NOT NULL , `DISCOUNT` DECIMAL NOT NULL , `DATE_CONTACT` DATE NOT NULL , PRIMARY KEY (`ID_BOOKST`)) ENGINE = INNODB;

- Click on the *Save* button to finish.

PhpMyAdmin shows the field structure on the left corner (figure 2.6).

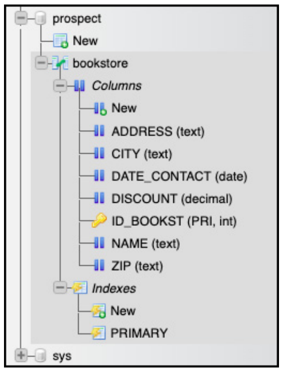


Figure 2.6 : The field structure

If you want to change the name of a field afterwards, perform the following:

- Select the table.
- Click on the *Structure* tab.
- Check the field you want the name to change.
- Click on the *Change* link.
- Modify the field name.
- Finish with the *Save* button.

The SQL code below shows the modification of the CITY field using the ALTER command.

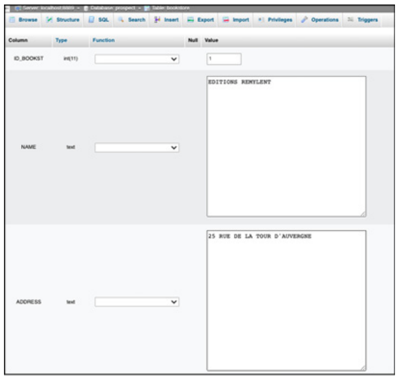
ALTER TABLE `BOOKSTORE` CHANGE `CITY` `CITIES` TEXT CHARACTER SET UTF8 COLLATE UTF8_GENERAL_CI NOT NULL;

2.1.3 Adding information to a table

To add data to a table, you must use the *Insert* tab.

- Select the table.
- Click on the *Insert* tab.
- Enter the data in the dedicated fields (the big rectangles).
- Finish with the *Go* button.

Figure 2.7 shows the entry of the first record. Do not fill in the primary key field, the system will take care of that automatically.



Column	Type	Function	Null	Value
ID_BOOKST	int(11)			1
NAME	text			EDITIONS REMYLENT
ADDRESS	text			25 RUE DE LA TOUR D'AUVERGNE

Figure 2.7 : Adding a record to the table

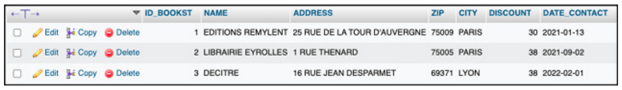
The SQL code that inserts data is as follows:

INSERT INTO \`BOOKSTORE\` (\`ID_BOOKST\`, \`NAME\`, \`ADDRESS\`, \`ZIP\`, \`CITY\`, \`DISCOUNT\`, \`DATE_CONTACT\`) VALUES (NULL, 'LIBRAIRIE EYROLLES', '1 RUE THENARD', '75005', 'PARIS', '38', '2/9/21')

The values to insert are surrounded by inverted commas.

Once all the data is entered, you can see a summary by clicking on the *Browse* tab. Each line can be modified. Check the line then click on the *Edit* link.

Figure 2.8 shows 3 rows entered in the table.



	ID_BOOKST	NAME	ADDRESS	ZIP	CITY	DISCOUNT	DATE_CONTACT
Edit Copy Delete	1	EDITIONS REMYLENT	25 RUE DE LA TOUR D'AUVERGNE	75009	PARIS	30	2021-01-13
Edit Copy Delete	2	LIBRAIRIE EYROLLES	1 RUE THENARD	75005	PARIS	38	2021-09-02
Edit Copy Delete	3	DECITRE	16 RUE JEAN DESPARMET	69371	LYON	38	2022-02-01

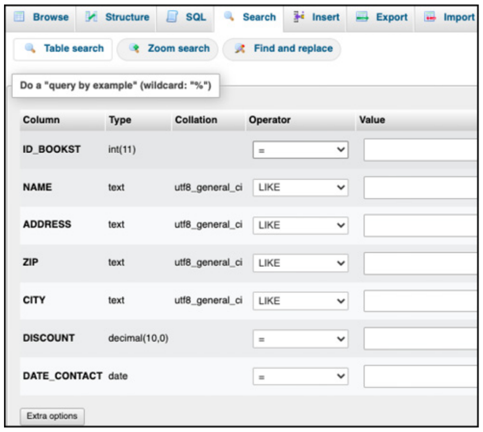
Figure 2.8 : Several rows in the table

Multiple actions are possible regarding records: editing or deleting a row, adding other information, searching, etc. All operations that can be done manually with PhpMyAdmin can be done with the SQL language.

2.1.4 Finding information in a table

A table can contain millions of rows. You can search for information using the *Search* tab.

Figure 2.9 shows the fields in the dialog.

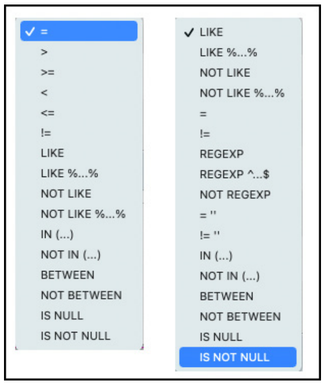


Do a "query by example" (wildcard: "%")

Column	Type	Collation	Operator	Value
ID_BOOKST	int(11)		=	
NAME	text	utf8_general_ci	LIKE	
ADDRESS	text	utf8_general_ci	LIKE	
ZIP	text	utf8_general_ci	LIKE	
CITY	text	utf8_general_ci	LIKE	
DISCOUNT	decimal(10,0)		=	
DATE_CONTACT	date		=	

Figure 2.9 : Several rows in the table

There are operators available in the *Operator* column. Figure 2.10 shows the numeric operators and the text ones.

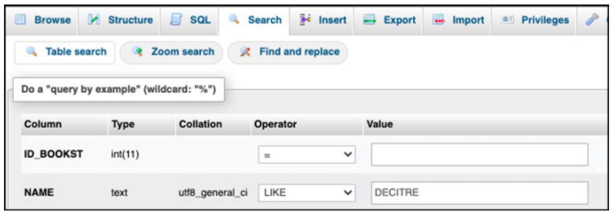


✓ =	✓ LIKE
>	LIKE %...%
>=	NOT LIKE
<	NOT LIKE %...%
<=	=
!=	!=
LIKE	REGEXP
LIKE %...%	REGEXP ^...$
NOT LIKE	NOT REGEXP
NOT LIKE %...%	= ''
IN (...)	!= ''
NOT IN (...)	IN (...)
BETWEEN	NOT IN (...)
NOT BETWEEN	BETWEEN
IS NULL	NOT BETWEEN
IS NOT NULL	IS NULL
	IS NOT NULL

Figure 2.10 : The different operators

To search for information, choose the needed conditional operator then enter a value in the *Value* column. Finish with the *Go* button. If the value is found, PhpMyAdmin will display the result.

Figure 2.11 shows a research for the library DECITRE.



Do a "query by example" (wildcard: "%")

Column	Type	Collation	Operator	Value
ID_BOOKST	int(11)		=	
NAME	text	utf8_general_ci	LIKE	DECITRE

Figure 2.11 : Searching a value

The SQL command that performs the upward research is as follows:

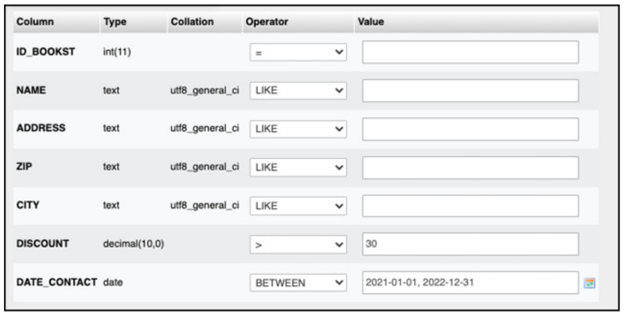
SELECT * FROM `BOOKSTORE` WHERE `NAME` LIKE 'DECITRE'

The sign * displays all the table fields without naming them.

2.1.5 Multiple operators

Several logical operators can be used together to narrow the research. For instance, suppose you need to list the bookstores between the years 2021 and 2022 and whose discount was greater than 30.

Figure 2.12 shows a mix of criteria.



Column	Type	Collation	Operator	Value
ID_BOOKST	int(11)		=	
NAME	text	utf8_general_ci	LIKE	
ADDRESS	text	utf8_general_ci	LIKE	
ZIP	text	utf8_general_ci	LIKE	
CITY	text	utf8_general_ci	LIKE	
DISCOUNT	decimal(10,0)		>	30
DATE_CONTACT	date		BETWEEN	2021-01-01, 2022-12-31

Figure 2.12 : A search with several conditions

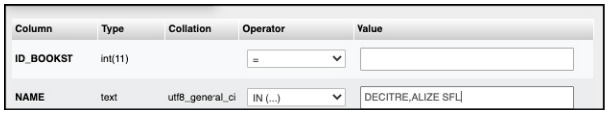
SELECT * FROM `BOOKSTORE` WHERE `DISCOUNT` > 30 AND `DATE_CONTACT` BETWEEN '2021-01-01' AND '2022-12-31'

2.1.6 Miscellaneous operators

The operator IN permits you to search for rows based on multiple contents inside a single column. For instance, you want to see all the records coming from two companies.

The SQL query below searches for all records where the names of the BOOKSTORE are *DECITRE* as well as *ALIZE SFL*.

SELECT * FROM `BOOKSTORE` WHERE `NAME` IN ('DECITRE', 'ALIZE SFL')



Column	Type	Collation	Operator	Value
ID_BOOKST	int(11)		=	
NAME	text	utf8_general_ci	IN (...)	DECITRE,ALIZE SFL

Figure 2.13 : The operator IN

The values are wrapped with apostrophes.

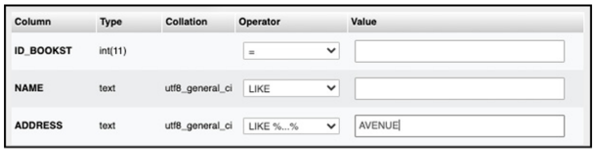
The opposite of the previous formulation would be written with the NOT IN operator:

SELECT * FROM `BOOKSTORE` WHERE `NAME` NOT IN ('DECITRE', 'ALIZE SFL')

The figure below shows another example where you search for all the bookstores that have their store in an 'AVENUE'.

The operator used is LIKE with the wildcard % feature.

It finds all expressions that have the word AVENUE, either at the beginning, in the middle or at the end of the field.



Column	Type	Collation	Operator	Value
ID_BOOKST	int(11)		=	
NAME	text	utf8_general_ci	LIKE	
ADDRESS	text	utf8_general_ci	LIKE %...%	AVENUE

Figure 2.14 : The operator wildcard

The SQL expression is as follow:

SELECT * FROM `BOOKSTORE` WHERE `ADDRESS` LIKE '%AVENUE%'

The operator IS NULL indicates that the field contains the value NULL.

If you want to search for columns that do not contain any value, you must use the operator ='.

The ! character indicates to negate the expression that follows. For instance, the operator !=' (different from nothing) searches for everything that is filled in.

2.2 The SELECT command

SELECT is the main order of an SQL query. It retrieves a set of results according to the chosen fields and the defined criteria. With SELECT, you can specify multiple tables with joins (relationships) that are written in a WHERE clause.

The complete syntax of the query is as follows:

SELECT list-of-fields [INTO table_name]

FROM source_table]

WHERE search condition

GROUP BY grouping expression

HAVING condition in the group

ORDER BY sort field ASC | DESC

The list below details the different clauses.

- *SELECT*. This clause enables you to define the columns of the table (or tables) that can be related.
- *FROM*. With this clause you define the tables that are involved in the query.
- *WHERE*. This clause permits you to define equalities or comparisons between values. You can place join features between tables in case relationships are needed between them.
- *GROUP BY*. This clause enables you to group values in order to use calculation functions such as sums or averages.
- *HAVING*. This statement allows you to define equalities or comparisons within a grouping defined by the *GROUP BY* clause.
- *ORDER BY*. This clause enables the result of a search to be sorted by one or more columns. The sorting can also be defined in an ascending or descending manner.

UNION, *EXCEPT* and *INTERSECT* operators can also be used between multiple queries to combine or compare their results in a single result set. They depend on the database system.

Caution. A SELECT query can be included in another SELECT query.

2.2.1 Examples with SELECT

Here are some examples that produce a result set.

1. Displaying bookstores with zip codes beginning with 69.

SELECT * FROM BOOKSTORES WHERE SUBSTRING(`ZIP`,1,2)='69'

2. Displaying the bookstores whose zip code does not start with 69.

SELECT * FROM BOOKSTORES WHERE NOT SUBSTRING(`ZIP`,1,2)='69'

3. Displaying the bookstores from Lyon (ZIP begins with 69) and Paris (ZIP begins with 75).

SELECT * FROM BOOKSTORES WHERE SUBSTRING(`ZIP`,1,2)='69' OR SUBSTRING(`ZIP`,1,2)='75

4. Displaying the bookstores that are given a 30% discount.

SELECT * FROM BOOKSTORES WHERE DISCOUNT=30

5. Displaying the bookstores that have a discount higher than 30%.

SELECT * FROM BOOKSTORES WHERE DISCOUNT>30

6. Displaying the bookstores that have a discount between 35% and 40%.

SELECT * FROM BOOKSTORES WHERE DISCOUNT BETWEEN 35 AND 40

7. Displaying the name of the bookstores and the city.

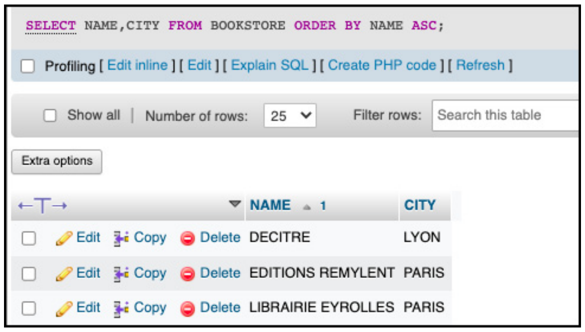
SELECT NAME, CITY FROM BOOKSTORES

2.2.2 Sorting a column with Order By

A result set is the successful return of an SQL query. It can contain one or more columns sorted in the ascending or descending direction.

The example below displays the names of the bookstores sorted alphabetically from A to Z with the cities.

SELECT NAME, CITY FROM BOOKSTORE ORDER BY NAME ASC



```
SELECT NAME,CITY FROM BOOKSTORE ORDER BY NAME ASC;
```

NAME	CITY
DECITRE	LYON
EDITIONS REMYLENT	PARIS
LIBRAIRIE EYROLLES	PARIS

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25
Filter rows: Search this table
Extra options
Edit Copy Delete

Figure 2.15 : The names are sorted alphabetically.

The DESC operator enables to sort in descending order.

You can also sort within several columns. The following example displays all the columns by sorting the ZIP column on the first 2 characters in ascending order, then the CITY column in ascending order.

SELECT * FROM `BOOKSTORE` ORDER BY LEFT(ZIP,2) ASC,CITY ASC

2.2.3 Grouping with GROUP By

The GROUP BY operator enables you to group rows according to one or several criteria.

The operator creates subsets on which calculations will be performed. The GROUP BY clause takes into account the columns that are used to compose each subset. It is very easy to count elements.

The following example counts the number of times a city appears.

SELECT CITY, COUNT(*) as NUMBER FROM BOOKSTORE GROUP BY CITY

If you want to display the cities excluding duplicates, you can write the following expression:

SELECT CITY FROM BOOKSTORE GROUP BY CITY

The following table shows the calculation functions that can be used in group:

Functions	Description
AVG	Returns the value of the average.
COUNT	Returns the number of rows including duplicates and NULL values.
MAX	Returns the maximum value.
MIN	Returns the minimum value.
SUM	Returns the sum of the values.
STDEV	Returns the standard deviation.

2.2.4 The HAVING clause

It limits the number of lines displayed after a GROUP BY operation. Therefore you can act on the rows that come from a grouping operation.

The following SQL displays the names of the cities and the number of times the city appears.

SELECT CITY, COUNT(*) AS NUMBER FROM BOOKSTORE GROUP BY CITY

The following example displays the names of the cities that have 2 bookstores.

SELECT CITY, COUNT(*) AS NUMBER FROM BOOKSTORE GROUP BY CITY HAVING NUMBER>2

The HAVING command is applied on a group of elements defined in the GROUP BY clause. HAVING accepts calculations on the fields.

We could have written COUNT(`CITY`) instead of COUNT(*).

2.3 Managing records

You can act on the records thanks to the links *Edit*, *Copy*, *Delete* and *Export*.

2.3.1 Deleting a record

To delete a row, perform the following operations:

- Select the table.
- Click on the *Browse* tab.
- Check the row (or the rows) you want to delete then click on *Delete*.
- Confirm with OK.

The SQL code that deletes a row is as follows:

DELETE FROM `BOOKSTORE` WHERE `BOOKSTORE`.`K_LIB` = 7;

2.3.2 Copying a record

Here is the procedure that enables you to copy a row by changing some values:

- Select the table.
- Click on the *Browse* button.
- Check the row you want to copy then click on *Copy*.
- Change the values and finish by clicking on the *Go* button.

The SQL code uses the expression INSERT INTO.

INSERT INTO `BOOKSTORE` (`ID_BOOKST`, `NAME`, `ADDRESS`, `ZIP`, `CITY`, `DISCOUNT`, `DATE_CONTACT`) VALUES (NULL, 'DECITRE', '16 AVENUE JEAN DESPARMET', '69371', 'LYON', '38', '2022-02-01');

2.3.3 The UPDATE command

UPDATE enables you to replace one value by another for a whole group of rows defined by a criterion in the WHERE clause.

You have to be careful with this UPDATE command because it acts on all rows. After the confirmation, you will not be able to go back, unless the database (or the table) has been saved.

The syntax is as follows:

UPDATE 'Table name'

SET 'field' = 'new value'

WHERE condition

Here are some examples with UPDATE.

The following query changes the word TOULOUSE to TOULOUSE CEDEX in the CITY field.

UPDATE `BOOKSTORE` SET `CITY`='TOULOUSE CEDEX' WHERE `CITY` LIKE '%TOULOUSE%'

The % character is a wildcard.

And to go back :

UPDATE `BOOKSTORE` SET `CITY`='TOULOUSE' WHERE `CITY`='TOULOUSE CEDEX'

You can also use string functions inside an UPDATE statement.

The following expression transforms the address field into upper case:

UPDATE `BOOKSTORE`

SET `ADDRESS`=UPPER(`ADDRESS`)

The following expression changes the discount to 31% for the BOOKSTORE between the year 2019:

UPDATE `BOOKSTORE` SET `DISCOUNT` = 31 WHERE `DATE_CONTACT` BETWEEN '2019-01-01' AND '2019-12-31'

With SQL, dates are always written in the form YYYY-MM-DD.

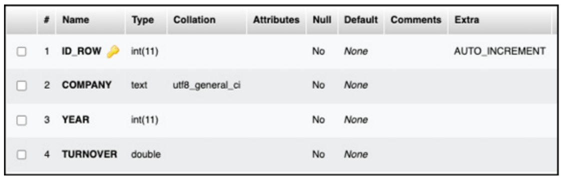
2.4 Calculating

You can perform calculations with numeric fields.

The following SQL code creates a table with three fields: *company*, *year*, *turnover*.

CREATE TABLE `PROSPECT`.`REPORT` (`ID_ROW` INT NOT NULL AUTO_INCREMENT , `COMPANY` TEXT NOT NULL , `YEAR` INT NOT

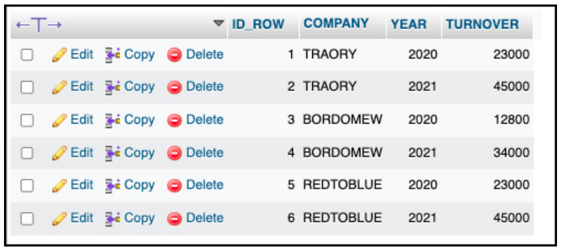
NULL , `TURNOVER` DOUBLE NOT NULL) ENGINE = InnoDB;



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	ID_ROW	int(11)			No	*None*		AUTO_INCREMENT
2	COMPANY	text	utf8_general_ci		No	*None*		
3	YEAR	int(11)			No	*None*		
4	TURNOVER	double			No	*None*		

Figure 2.16 : Creating a table

Figure 2.17 shows some records inserted in the table.



	ID_ROW	COMPANY	YEAR	TURNOVER
Edit Copy Delete	1	TRAORY	2020	23000
Edit Copy Delete	2	TRAORY	2021	45000
Edit Copy Delete	3	BORDOMEW	2020	12800
Edit Copy Delete	4	BORDOMEW	2021	34000
Edit Copy Delete	5	REDTOBLUE	2020	23000
Edit Copy Delete	6	REDTOBLUE	2021	45000

Figure 2.17 : Some data in the table

Caution. With the *Browse* feature and only if your table has a primary key, you can edit data directly by double-clicking on a field.

2.4.1 Grouping and totaling

The code below calculates a total of the turnover grouping by company and with a sort descending. The result of the turnover is displayed in a column renamed TOTAL.

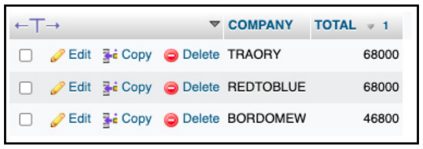
SELECT `COMPANY`,SUM(`TURNOVER`) AS TOTAL

FROM REPORT

GROUP BY `COMPANY`

ORDER BY TOTAL DESC

Figure 2.18 shows the result.



	COMPANY	TOTAL 1
Edit Copy Delete	TRAORY	68000
Edit Copy Delete	REDTOBLUE	68000
Edit Copy Delete	BORDOMEW	46800

Figure 2.18 : Grouping and calculating

2.4.2 Statistics

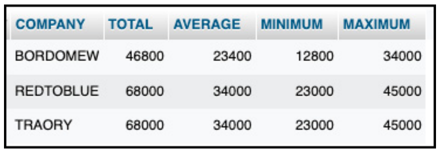
The code below shows how to extract statistics thanks to several functions

SELECT `COMPANY`,SUM(`TURNOVER`) AS TOTAL, AVG(`TURNOVER`) AS AVERAGE, MIN(`TURNOVER`) AS MINIMUM, MAX(`TURNOVER`) AS MAXIMUM

FROM REPORT

GROUP BY `COMPANY`

Figure 2.19 shows the result.



COMPANY	TOTAL	AVERAGE	MINIMUM	MAXIMUM
BORDOMEW	46800	23400	12800	34000
REDTOBLUE	68000	34000	23000	45000
TRAORY	68000	34000	23000	45000

Figure 2.19 : Statistics

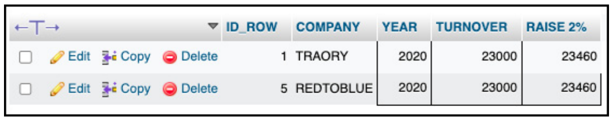
2.4.3 Calculating with a condition WHERE

The clause WHERE enables you to select records according to a logical condition.

The code below calculates a raise of 2% from the turnover solely for both the year 2020 and a turnover > 20000.

SELECT *,`TURNOVER`*1.02 AS 'RAISE 2%' FROM `REPORT` WHERE `YEAR` LIKE '2020' AND `TURNOVER` > 20000;

Figure 2.20 shows the result.



	ID_ROW	COMPANY	YEAR	TURNOVER	RAISE 2%
Edit Copy Delete	1	TRAORY	2020	23000	23460
Edit Copy Delete	5	REDTOBLUE	2020	23000	23460

Figure 2.20 : Calculating after two logical conditions

2.4.4 Calculating with UPDATE

The UPDATE clause modifies the data for all the records or a group of records defined with the clause WHERE.

Be careful with UPDATE because once completed, you cannot reverse the action.

You can define a calculation in the UPDATE expression. For instance, the code below adds 8% to all turnover for the year 2020.

UPDATE REPORT

SET `TURNOVER`=1.08*`TURNOVER`

WHERE `YEAR`='2020'

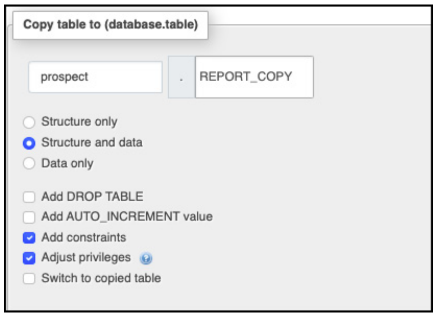
2.4.5 Duplicating a table

It is a good idea to copy a table before acting on it with the UPDATE order.

Follow the steps below to copy any table:

- Select the table to copy.
- Click on the *Operation* tab.
- Type the name of the new table
- Go to perform the operation.

Figure 2.21 shows the settings. You can copy the structure only, the structure and the date or the data only. You can add the constraints and the privileges if needed.


Copy table to (database.table)

prospect . REPORT_COPY

( ) Structure only
(●) Structure and data
( ) Data only

☐ Add DROP TABLE
☐ Add AUTO_INCREMENT value
☑ Add constraints
☑ Adjust privileges
☐ Switch to copied table

Figure 2.21 : Copying a table

2.5 The routines

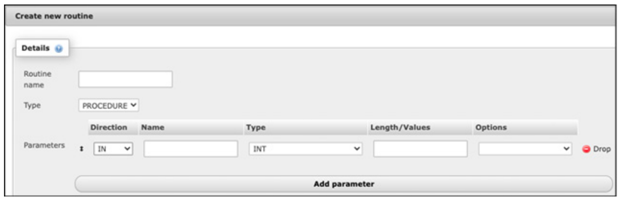
To save the SQL code that contains an UPDATE order, you can create a routine (also called stored procedure) that will be saved in the database. A routine is always defined with a name and is invoked by a CALL command inside another routine.

When a SQL query is performed, it is only kept in memory for the current session. On the contrary, a routine is permanently stored in the database.

Follow the steps below to create a routine with PhpMyAdmin:

- Select the database.
- Click on the *Routines* tab.
- Click on *Create new routine*.

Figure 2.22 shows the dialog box that requests information.

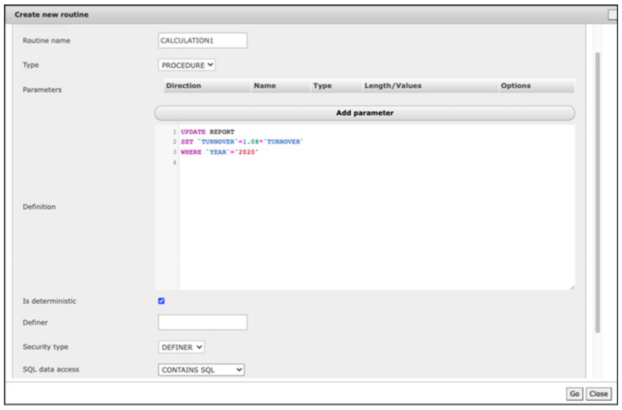


Routine name
Type PROCEDURE

	Direction	Name	Type	Length/Values	Options
Parameters	IN		INT		

Figure 2.22 : Creating a routine.

- Enter the routine name, for instance, enter the expression CALCULATION1.
- Choose whether the routine is a procedure or a function. With a function, parameters are defined. Here, it is a routine.
- Click on the *Drop* link at the right corner to delete the parameter line.
- In the *Definition* area, copy the SQL code that updates the data.
- Check *Is deterministic*. A calculation function that returns an average is deterministic because you know exactly what will happen. On the other hand, a calculation that depends on the current date is not always deterministic. In the case of an update order of a column in relation to others, you can check this parameter.



Routine name: CALCULATION1
Type: PROCEDURE
Parameters:

Direction	Name	Type	Length/Values	Options

Definition:

```
UPDATE REPORT
SET `TURNOVER`=1.08*`TURNOVER`
WHERE `YEAR`='2020'
```

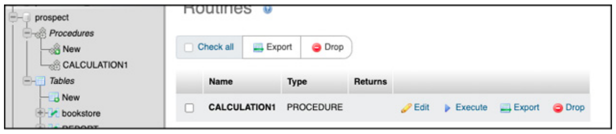
Is deterministic: ✓
Definer:
Security type: DEFINER
SQL data access: CONTAINS SQL

Figure 2.23 : The routine body

- *Definer*. Specify the name of the person who creates the routine.
- *Security type*. With DEFINER, the person who creates the routine or the function must have the necessary rights. With INVOKER, it is the person who starts the routine or the function who must have the necessary rights.
- SQL data access. With the NO SQL setting, the database is not relational and is intended for Big Data. It is another form of storage that is used by applications (Facebook or others) that do not structure data in a SQL form. With the CONTAINS SQL parameter, the routine contains SQL statements that do not involve reading or writing data. With the READS SQL DATA parameter, the procedure contains SQL statements that read data. With the parameter MODIFIES SQL DATA, SQL expressions (UPDATE, INSERT, DELETE or ALTER) can write data.
- *Comments*. Specify remarks if necessary.
- Finish by clicking on the *Go* button.

Click the Edit button to return to the dialog box.

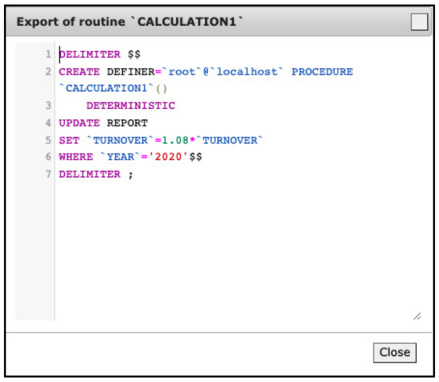
Figure 2.24 shows the name of the new routine stored in the database.



Name	Type	Returns
CALCULATION1	PROCEDURE	

Figure 2.24 : The routine is stored in the database

To see the SQL code created automatically, select the routine then click on *Export*. Figure 2.25 shows the code.



```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`CALCULATION1`()
    DETERMINISTIC
UPDATE REPORT
SET `TURNOVER`=1.08*`TURNOVER`
WHERE `YEAR`='2020'$$
DELIMITER ;
```

Figure 2.25 : The code of the stored procedure

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `CALCULATION1`()

DETERMINISTIC

UPDATE REPORT

SET `TURNOVER`=1.08*`TURNOVER`

WHERE `YEAR`='2020'$$

DELIMITER ;

To perform a routine, click on the *Execute* button.

It is possible to call a routine from another with the CALL statement

The code below calls the routine CALCULATION1.

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `LAUNCH`()

MODIFIES SQL DATA

```
DETERMINISTIC
CALL CALCULATION1()$$
DELIMITER ;
```

In general when writing SQL statements, a semicolon is used to separate them. But in a routine, you have to redefine this separator character. Thus, the whole set of statements is considered as a single block.

2.6 The views

Views are virtual tables created by assembling other tables according to several criteria. Views are created using a SELECT query and can be saved for re-use. For instance one view can display certain columns of a table while another view can perform calculations.

The general syntax of a view is as follows:

```
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { USER | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column(s))]
AS query_select
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

The following list details the different parameters of this syntax:

- OR REPLACE. If a view exists, it is deleted and replaced by the new one.
- ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}. The UNDEFINED parameter is the default value. The MERGE parameter uses the SQL query that was used to create the

view as the basis for the operation. The result set comes from the table specified in the query. The TEMPTABLE parameter uses a temporary table created to store the results. It enables faster release of locks that apply to the underlying tables. With the MERGE value, the result set will be editable unlike the TEMPTABLE value.

- DEFINER = { USER | CURRENT_USER }. This parameter enables you to assign a creator to the view which is CURRENT_USER by default.
- SQL SECURITY { DEFINER | INVOKER }. This parameter allows you to define the user's rights.
- WITH [CASCADED | LOCAL] CHECK OPTION. This parameter checks the constraints specified in the WHERE clause of a modifiable view, when modifying its data. The CASCADED parameter is the default value. It allows to check the constraint for the view as well as for the underlying derived views. With the LOCAL parameter, the constraint is checked only on the view and not on its derivatives.

The notions of referential integrity constraints will be explained in the next chapter.

Once the view has been created, it can be modified with the ALTER VIEW command.

To delete a view, use the DROP VIEW expression.

The following steps show how to create a view with the SQL code:

```
CREATE VIEW `FEBRUARY` AS

SELECT * FROM `REPORT` WHERE `COMPANY` LIKE 'TRAORI'
```

Once the query is run, you will see a new feature inside the database that allows you to create or edit views.

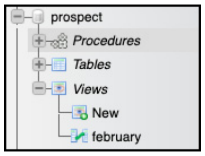


Figure 2.26 : Creating a view from the database

You can save as many views as you wish.

2.7 The triggers

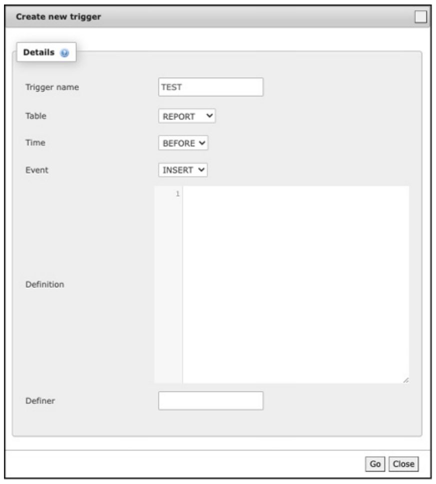
A trigger is a sequence of stored code that the database server runs if a specific event occurs (UPDATE, INSERT or DELETE). For instance, one can imagine that before inserting a row in a table, some controls must be tested. Triggers are defined and stored at the database level.

You can create them both with PhpMyAdmin or SQL.

The following list details the creation of a trigger:

- Select the database.
- Click on the *Triggers* tab.
- Click on the *Create new trigger* button.

Figure 2.27 shows the dedicated dialog box.



Create new trigger

Details

Trigger name	TEST
Table	REPORT
Time	BEFORE
Event	INSERT
Definition	
Definer	

Figure 2.27 : The trigger parameters

The list below shows the parameters to be filled in:

- *Trigger name*. In this field, enter the name of the trigger.
- *Table*. Use the drop-down list to choose the name of the table.
- *Time*. Use the drop-down list to choose when (BEFORE or AFTER) the trigger should be activated.
- *Event*. Choose the event that will activate the trigger (INSERT, UPDATE or DELETE).
- *Definition*. In this field, place the code that the database server will run.
- *Definer*. Indicate the user (he must have the rights) who can run the trigger.
- Finish with the *Go* button to create the trigger.

Afterwards you will be able to modify or delete the trigger. An *Export* button enables you to view the trigger code.

The code below shows the syntax that can be changed according to the database system.

```
CREATE

TRIGGER=TRIGGER_NAME

DEFINER = USER_NAME

INSERT | UPDATE | DELETE

BEFORE or AFTER

ON TABLE_NAME

THE CODE
```

For instance, the following code replaces a value after an update action on a table:

```
DELIMITER //

CREATE TRIGGER TEST AFTER INSERT ON STUDENT

FOR EACH ROW
```

BEGIN

SET STUDENT.TOTAL = '12'

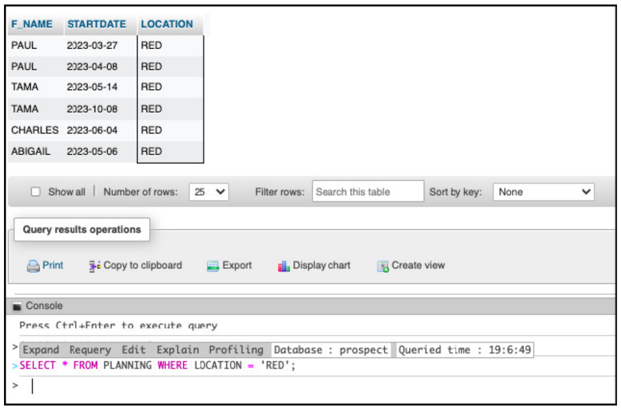
END//

DELIMITER;

The SQL code can be tested with the console feature that allows you to test any action.

2.8 The console

If you want to test a SQL query and study the result, you use the *console* that is located at the bottom of the window as shown figure 2.28.

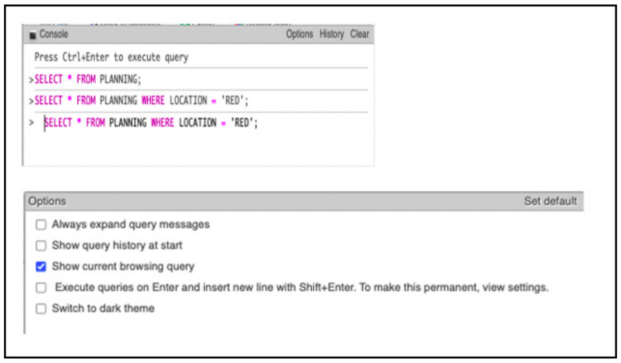


F_NAME	STARTDATE	LOCATION
PAUL	2023-03-27	RED
PAUL	2023-04-08	RED
TAMA	2023-05-14	RED
TAMA	2023-10-08	RED
CHARLES	2023-06-04	RED
ABIGAIL	2023-05-06	RED

```
Press Ctrl+Enter to execute query
Database : prospect   Queried time : 19:6:49
>SELECT * FROM PLANNING WHERE LOCATION = 'RED';
>
```

Figure 2.28 : The console

By default the query is running with the keys *CTRL ENTER* but you can change it with the options (figure 2.29).



```
Press Ctrl+Enter to execute query
>SELECT * FROM PLANNING;
>SELECT * FROM PLANNING WHERE LOCATION = 'RED';
> SELECT * FROM PLANNING WHERE LOCATION = 'RED';
```

- ☐ Always expand query messages
- ☐ Show query history at start
- ☑ Show current browsing query
- ☐ Execute queries on Enter and insert new line with Shift+Enter. To make this permanent, view settings.
- ☐ Switch to dark theme

Figure 2.29 : The console options

If you want to modifiy the previous code, place the pointer on the line, then click on the *Edit* link.

The console is a very practical feature to test the SQL code directly.

2.9 The privileges

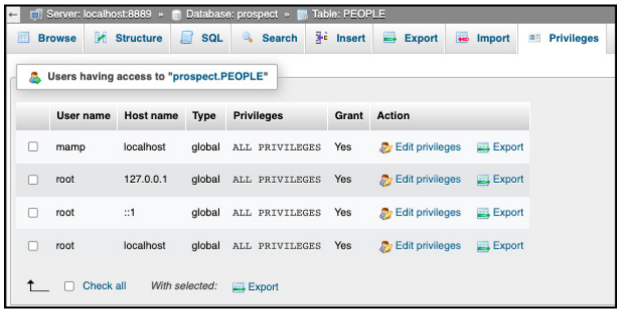
There are orders that give (or not) a right to manipulate the tables and even more so, all the database objects. You can define several privileges to several accounts on several tables with the SELECT, UPDATE, DELETE and INSERT clause.

The main command is GRANT.

To see all the possible privileges and the SQL code associated to user accounts, follow the steps below:

- Select a table.
- Click on the *Privileges* tab.

Figure 2.30 shows the available privileges scope.



Users having access to "prospect.PEOPLE"

	User name	Host name	Type	Privileges	Grant	Action	
☐	mamp	localhost	global	ALL PRIVILEGES	Yes	Edit privileges	Export
☐	root	127.0.0.1	global	ALL PRIVILEGES	Yes	Edit privileges	Export
☐	root	::1	global	ALL PRIVILEGES	Yes	Edit privileges	Export
☐	root	localhost	global	ALL PRIVILEGES	Yes	Edit privileges	Export

☐ Check all *With selected:* Export

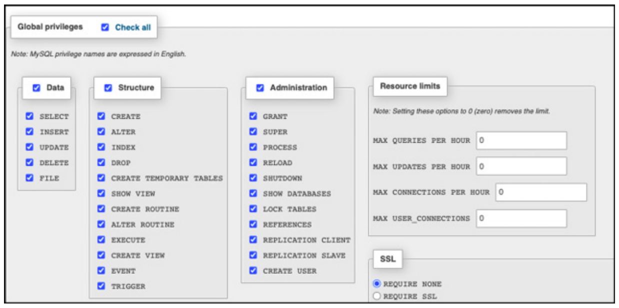
Figure 2.30 : Privileges for users

In case of several users, privileges can be granted. For instance some people have the right to see information from the tables, but cannot change the data.

When you want to grant privileges, you must determine the actions for each table.

- Click on *Edit privileges*.

Figure 2.31 shows the available actions. Each checkbox allows you to define the granting degree in a very specific way.



Global privileges ✓ Check all

Note: MySQL privilege names are expressed in English.

✓ Data
- ✓ SELECT
- ✓ INSERT
- ✓ UPDATE
- ✓ DELETE
- ✓ FILE

✓ Structure
- ✓ CREATE
- ✓ ALTER
- ✓ INDEX
- ✓ DROP
- ✓ CREATE TEMPORARY TABLES
- ✓ SHOW VIEW
- ✓ CREATE ROUTINE
- ✓ ALTER ROUTINE
- ✓ EXECUTE
- ✓ CREATE VIEW
- ✓ EVENT
- ✓ TRIGGER

✓ Administration
- ✓ GRANT
- ✓ SUPER
- ✓ PROCESS
- ✓ RELOAD
- ✓ SHUTDOWN
- ✓ SHOW DATABASES
- ✓ LOCK TABLES
- ✓ REFERENCES
- ✓ REPLICATION CLIENT
- ✓ REPLICATION SLAVE
- ✓ CREATE USER

Resource limits

Note: Setting these options to 0 (zero) removes the limit.

MAX QUERIES PER HOUR	0
MAX UPDATES PER HOUR	0
MAX CONNECTIONS PER HOUR	0
MAX USER_CONNECTIONS	0

SSL

[✓ REQUIRE NONE] REQUIRE SSL

Figure 2.31 : Granting privileges

You can prevent the user from selecting, inserting, updating or deleting data from the tables. You can define the rights to modify the table structure and so on.

The following code gives the rights to use the SELECT clause for the user *remy* with database *propects*.

GRANT SELECT ON `prospect`.* TO 'remy';

In brief

In this chapter you have studied how to create tables and fields. The SQL language proposes a simple set of instructions that you can discover thanks to the PhpMyAdmin environment. Routines and views enable you to save actions for re-use.

Chapter 3 will treat the techniques of relations between tables thanks to SQL schemas and joins.

Chapter 3
Relationships

This chapter shows how to create relationships between tables using joins and SQL.

3.1 Defining relationships

As we saw in the previous chapter, a database is used to store various information in tables. The following is a list of important rules when dealing with tables:

- *Rule 1*. Never repeat the same information in more than one table.
- *Rule 2*. Respect the consistency of information in each column of a table. For instance, do not put invoice dates in a column that should contain city names.
- *Rule 3*. Define a primary key for each table created. Prefer an auto-incrementing field that increases the previous value by 1, each time a new record is created.
- *Rule 4*. A relationship between two tables is always created between the primary key of the first table and a foreign key of the second. The foreign key is formed by one or more columns of the second table.

The most common types of relationships you may encounter are the following:

- *The 1-to-1 relationship*. A value in the primary key column of a first table is linked to the same value in a column of a second table.

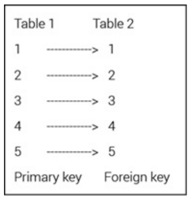


Figure 3.1 : The 1 to 1 relationship

This 1 to 1 relationship is used when two tables contain different columns, linked together by an identical number stored in each table. This case rarely occurs because usually all columns are stored in one table.

- *The 1-to-many relationship*. A value in a table 1 is linked to several identical values in a table 2.

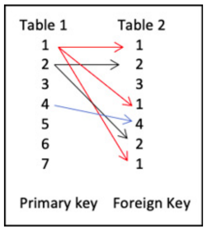


Figure 3.2 : The 1-to-many relationship

This *1-to-many* relationship is very common in daily information management. For instance, imagine a manager who deals with several buildings. For each building, he will have to know information about apartments, tenants, owners etc. An other example could be the management of several company departments such as communication, general services, human resources, legal and so on.

The organization defined by the form of *1-to-many* must respect a *parent-child* rule.

- The parent table is the one that has a primary key field.
- The child table is the one that has a foreign key field in addition to its own primary key.

The primary/foreign key pairing defines a relationship. Keep in mind that both primary/foreign key columns must have the same field type.

3.1.1 The BOOKS table

Figure 3.3 shows a table with 4 fields to manage different books.

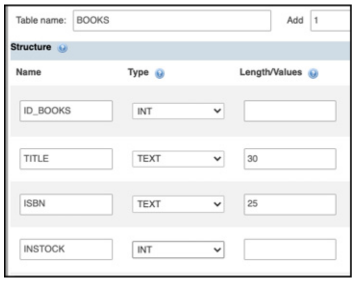


Table name: BOOKS Add 1

Structure

Name	Type	Length/Values
ID_BOOKS	INT	
TITLE	TEXT	30
ISBN	TEXT	25
INSTOCK	INT	

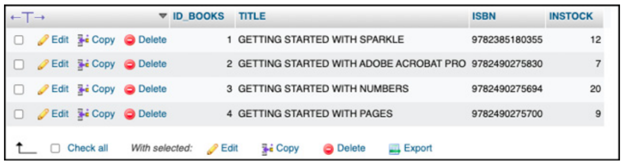
Figure 3.3 : The BOOKS table

The first field ID_BOOKS is the primary key of the table. It is defined as auto-incremented.

The code below creates this table.

CREATE TABLE `prospect`.`BOOKS` (`ID_BOOKS` INT NOT NULL AUTO_INCREMENT , `TITLE` TEXT NOT NULL , `ISBN` TEXT NOT NULL , `INSTOCK` INT NOT NULL , PRIMARY KEY (`ID_BOOKS`)) ENGINE = InnoDB;

Figure 3.4 shows four inserted records



	ID_BOOKS	TITLE	ISBN	INSTOCK
Edit Copy Delete	1	GETTING STARTED WITH SPARKLE	9782385180355	12
Edit Copy Delete	2	GETTING STARTED WITH ADOBE ACROBAT PRO	9782490275830	7
Edit Copy Delete	3	GETTING STARTED WITH NUMBERS	9782490275694	20
Edit Copy Delete	4	GETTING STARTED WITH PAGES	9782490275700	9

Figure 3.4 : The data in the BOOKS table

The code below shows how to insert a record.

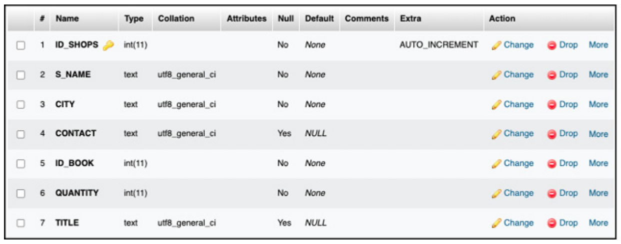
INSERT INTO `BOOKS` (`ID_BOOKS`, `TITLE`, `ISBN`, `INSTOCK`) VALUES (NULL, 'GETTING STARTED WITH NUMBERS', '9782490275694', '20'), (NULL, 'GETTING STARTED WITH PAGES', '9782490275700', '9');

The primary key is automatically incremented.

3.1.2 The SHOPS table

A publisher sells books to several shops. To know the information of each of them, a table is necessary.

Figure 3.5 shows the structure of the SHOPS table.



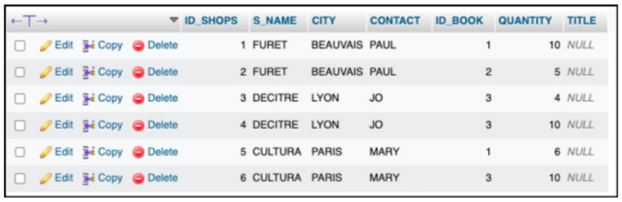
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action		
1	ID_SHOPS	int(11)			No	*None*		AUTO_INCREMENT	Change	Drop	More
2	S_NAME	text	utf8_general_ci		No	*None*			Change	Drop	More
3	CITY	text	utf8_general_ci		No	*None*			Change	Drop	More
4	CONTACT	text	utf8_general_ci		Yes	*NULL*			Change	Drop	More
5	ID_BOOK	int(11)			No	*None*			Change	Drop	More
6	QUANTITY	int(11)			No	*None*			Change	Drop	More
7	TITLE	text	utf8_general_ci		Yes	*NULL*			Change	Drop	More

Figure 3.5 : The SHOPS table

The list below details the different fields:

- *ID_SHOPS*. It is the primary key of the table which totally identifies each record.
- *S_NAME*. It is the name of the shop.
- CITY. It is the city of the shop.
- *CONTACT*. It is the contact of the shop that can be empty if necessary.
- *ID_BOOK*. This number defines a book that is sold to the shop.This field is the foreign key that will be linked to the primary key of the BOOKS table.
- *Quantity*. It is the number of books sold to a shop.
- TITLE. It is the name of the book that can be empty because it will come from the table BOOKS.

Figure 3.6 shows several records inserted in this table.



←T→			ID_SHOPS	S_NAME	CITY	CONTACT	ID_BOOK	QUANTITY	TITLE
Edit	Copy	Delete	1	FURET	BEAUVAIS	PAUL	1	10	*NULL*
Edit	Copy	Delete	2	FURET	BEAUVAIS	PAUL	2	5	*NULL*
Edit	Copy	Delete	3	DECITRE	LYON	JO	3	4	*NULL*
Edit	Copy	Delete	4	DECITRE	LYON	JO	3	10	*NULL*
Edit	Copy	Delete	5	CULTURA	PARIS	MARY	1	6	*NULL*
Edit	Copy	Delete	6	CULTURA	PARIS	MARY	3	10	*NULL*

Figure 3.6 : The data of the SHOPS data

To retrieve data from the two tables, you need to create a relationship between them.

3.1.3 Displaying the schema

The schema is a chart that shows the representation of the linked tables. Several tools are at your disposal.

Follow the steps below to view the schema:

- Select the database.
- Click on the *Designer* tab.
- Click on the first icon at the left corner.
- *Show/Hide* tables list
- Uncheck the tables that are not necessary.

Figure 3.7 shows the tools available.

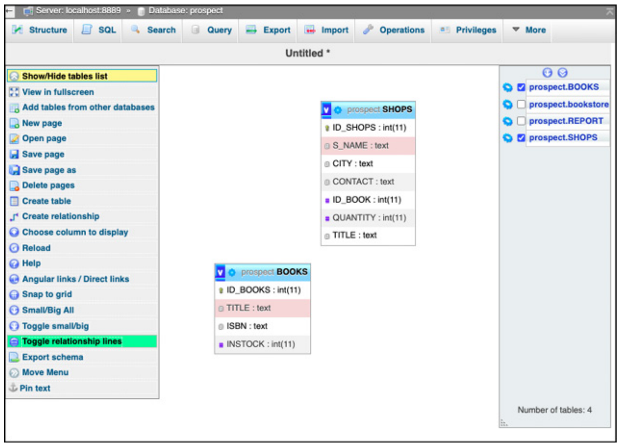


Figure 3.7 : The two tables BOOKS and SHOPS in the schema

Some icons are dedicated to the creation of relationships.

3.1.4 Creating a foreign key

A first table can only be linked to a second table if the following conditions are set:

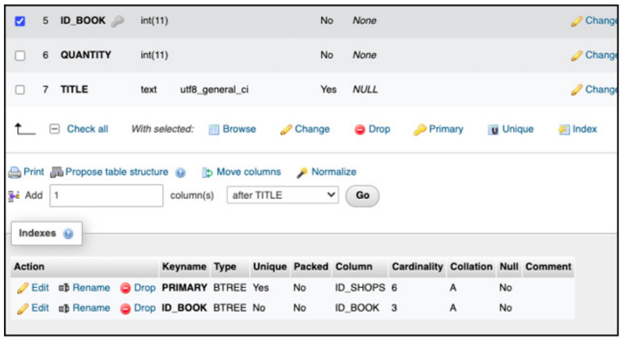
- The first table has a primary key.
- The second table has a foreign key with the same type that the primary key.

In the previous diagram, the BOOKS table has a primary key called *ID_BOOKS*. It can be linked to the *ID_BOOK* field which can have a different name. Here, the two field names are not exactly the same.

To create a foreign key, perform the following steps:

- Display the structure of the table that contains the next foreign key.
- Check the field.
- Click on the *Index* button.
- Go

Figure 3.8 shows the index characteristics.



✓	5	ID_BOOK	int(11)		No	*None*	Change
	6	QUANTITY	int(11)		No	*None*	Change
	7	TITLE	text	utf8_general_ci	Yes	*NULL*	Change

Action			Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit	Rename	Drop	PRIMARY	BTREE	Yes	No	ID_SHOPS	6	A	No	
Edit	Rename	Drop	ID_BOOK	BTREE	No	No	ID_BOOK	3	A	No	

Figure 3.8 : Creating an index for the foreign key

3.1.5 Creating a relationship

To create a link between the two tables, you must use the tool *Create relationship* in the *Designer* feature

- Select the database.
- Click on the *Designer* tab.
- Click on the first icon at the left corner.

- *Show/Hide* tables list
- Uncheck the tables that are not necessary.
- Click on *Create relationship*.
- Click on the primary key.
- Click on the foreign key.
- *OK*
- Click on *Toggle relationship lines.*
- Double-click on the database name to reload it.

Figure 3.9 shows the link between the two tables.

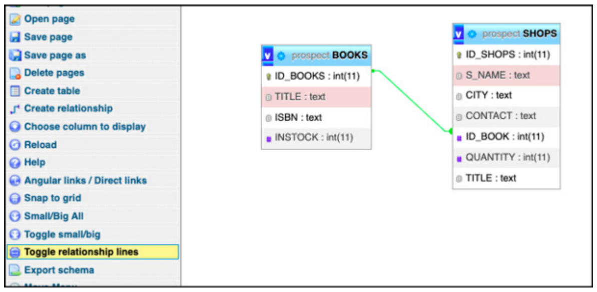


Figure 3.9 : The relationship lines

3.1.6 The table constraints

When creating a table, you can specify constraints that will be applied to the data. These are rules that apply to the columns in a table, or at the level of the table itself. A constraint ensures the accuracy and reliability of the information in the database.

The most common constraints are

- *NOT NULL*. A column may or may not contain NULL values. A NULL constraint allows you to leave the field blank. If you want the column to be filled in, specify NOT NULL in the line of code.

The following code modifies the structure of the table SALES:

```
ALTER TABLE `SALES` CHANGE `display_rate` `display_rate` DECIMAL(10,2) NOT NULL;
```

- *DEFAULT*. This constraint provides a default value for a column when the INSERT INTO statement does not provide a specific value.

```
ALTER TABLE `SALES` CHANGE `display_rate` `display_rate` DECIMAL(10,2) NOT NULL DEFAULT '30';
```

- *UNIQUE*. The UNIQUE constraint prevents two rows from having identical values in a column.

```
ALTER TABLE `SALES` ADD UNIQUE( `CODE1`);
```

- *CHECK*. This constraint sets a condition that must be checked at record validation time. If the condition fails, the record is not validated.

```
ALTER TABLE LIBRARIES ADD CONSTRAINT chk_remise CHECK (discount>=30)
```

- *INDEX*. An index is used both to enhance the search of a value and to create a foreign key. The index can be created using a single column or a group of columns.

```
ALTER TABLE `LIBRARIES` ADD INDEX( `CONTACT`);
```

- *PRIMARY KEY*. A primary key constraint specifies that a column will be used as a unique, non-NULL identifier for a record. A primary key automatically creates a unique B-tree index.

The following shows the creating of a primary key:

```
CREATE TABLE OUTPUT (

x integer, y integer, z integer,

PRIMARY KEY (x, z)

);
```

- *FOREIGN KEY*. A foreign key constraint indicates that the values in a column (or group of columns) must match the values that appear in the rows of another table (usually at the primary key level).

```
ALTER TABLE SALES

ADD CONSTRAINT bookstore_code

FOREIGN KEY (bookstore_code)
```

3.1.7 The UPDATE or DELETE constraints

When a primary key of a table is linked to a foreign key of another table, you can apply foreign key constraints of type DELETE or UPDATE. It is often referred to as *Referential Integrity.*

If a primary key points to several identical values at the foreign key level, you can say that the parent (primary key) has several children (foreign key).

What happens if you change the value of a parent? Will there be any repercussions towards the children ?

What happens if you delete a parent record ? Will there be children deleted ?

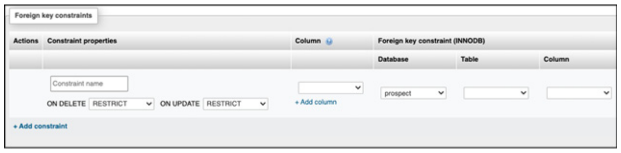
With an UPDATE constraint, any change to the parent primary key (if it is possible) will change the values of the children's foreign key field.

With a DELETE constraint, any deletion of a parent record will delete all the children's records linked.

Perform the steps below to investigate them:

- Select the database.
- Check the table that contains the primary key (BOOKS).
- Click on the *Structure* link at the right of the table name.
- Click on the *Relation View* button.

Figure 3.10 shows the settings.


Foreign key constraints

Actions	Constraint properties	Column	Foreign key constraint (INNODB)		
			Database	Table	Column
	Constraint name ON DELETE RESTRICT ON UPDATE RESTRICT	+ Add column	prospect		

Figure 3.10 : Setting up foreign key constraints

You must define a constraint name, the primary key column, the database name, the table that contains the foreign key and finally the foreign key column name.

For a delete constraint, click on the ON DELETE drop-down list. For a modification constraint, click on the ON UPDATE drop-down list. The possible values are: CASCADE, SET NULL, NO ACTION and RESTRICT.

The *SQL Preview* button shows the code that will be executed.

The list below details the different parameters:

- *CASCADE*. If a parent record is deleted or modified, all children will be also deleted or modified.
- *SET NULL*. A foreign key with this parameter means that if a record is deleted, the one that matches it will have a *NULL* value, but will not be deleted.
- *NO ACTION*. This means that no action is performed on the child record, if the parent record is deleted or modified.
- *RESTRICT*. This parameter prevents a parent from being deleted if it has children.

The following code creates a constraint between both BOOKS and SHOPS tables.

ALTER TABLE \`BOOKS\` ADD CONSTRAINT \`CONST1\` FOREIGN KEY (\`ID_BOOKS\`) REFERENCES \`SHOPS\`(\`ID_SHOPS\`) ON DELETE RESTRICT ON UPDATE CASCADE;

3.2 Multi-table queries

Once the relationships between the tables are well defined, PhpMyAdmin enables you to visually query the columns. A SQL query is created automatically and nothing prevents you from modifying it.

3.2.1 The query window

Perform the following steps to create a query based on multiple tables:

- Select the database.
- Click the *Query* tab.

The Query window is divided in three parts in which you can define the tables, the columns and the criteria.

The *Update Query* button will display the code SQL.

Figure 3.11 shows the structure that enables you to define the tables.

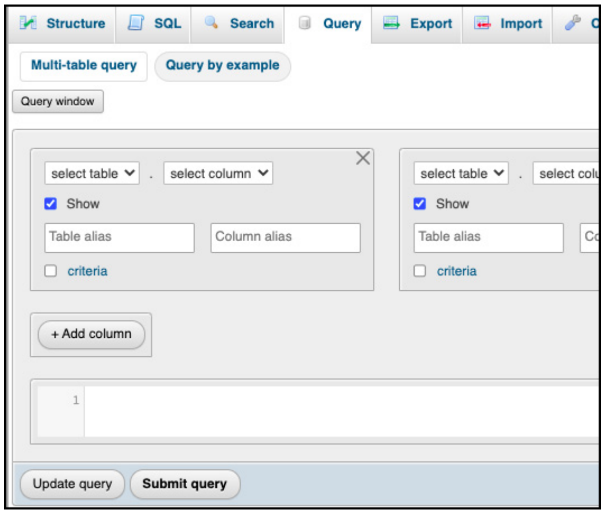


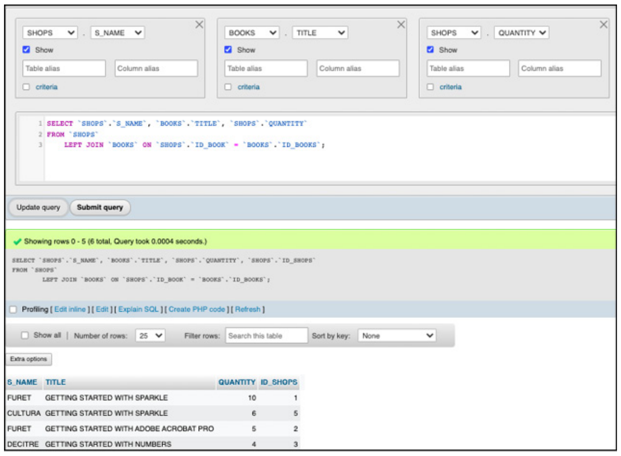
Figure 3.11 : Specifying tables and fields

The *Update query* button will display the SQL query in the dedicated white area.

The *Submit query* button will run the instructions. The result will appear at the bottom of the window.

The code below shows the shops to which books have been sent.

SELECT `SHOPS`.`S_NAME`, `BOOKS`.`TITLE`, `SHOPS`.`QUANTITY` FROM `SHOPS` LEFT JOIN `BOOKS` ON `SHOPS`.`ID_BOOK` = `BOOKS`.`ID_BOOKS`;



```
SELECT `SHOPS`.`S_NAME`, `BOOKS`.`TITLE`, `SHOPS`.`QUANTITY`
FROM `SHOPS`
    LEFT JOIN `BOOKS` ON `SHOPS`.`ID_BOOK` = `BOOKS`.`ID_BOOKS`;
```

✓ Showing rows 0 - 5 (6 total, Query took 0.0004 seconds.)

```
SELECT `SHOPS`.`S_NAME`, `BOOKS`.`TITLE`, `SHOPS`.`QUANTITY`, `SHOPS`.`ID_SHOPS`
FROM `SHOPS`
        LEFT JOIN `BOOKS` ON `SHOPS`.`ID_BOOK` = `BOOKS`.`ID_BOOKS`;
```

S_NAME	TITLE	QUANTITY	ID_SHOPS
FURET	GETTING STARTED WITH SPARKLE	10	1
CULTURA	GETTING STARTED WITH SPARKLE	6	5
FURET	GETTING STARTED WITH ADOBE ACROBAT PRO	5	2
DECITRE	GETTING STARTED WITH NUMBERS	4	3

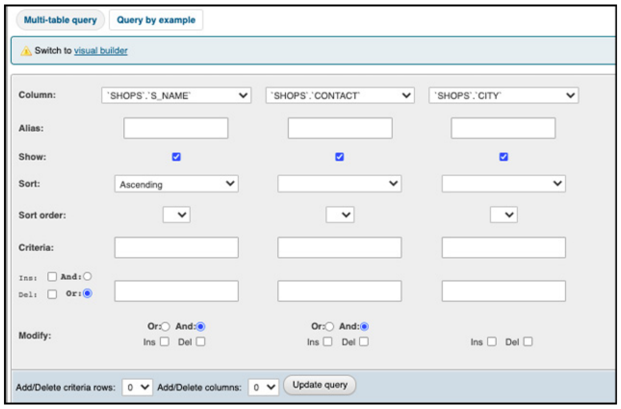
Figure 3.12 : Information from several tables

3.2.2 Performing a query by example

You can also define the tables and the columns thanks to the visual query system called *Query by Example*.

- Select the database.
- Click on the *Query* tab then choose *Query by Example*.

Figure 3.13 shows this visual feature.



Column:	`SHOPS`.`S_NAME`	`SHOPS`.`CONTACT`	`SHOPS`.`CITY`
Show:	✓	✓	✓
Sort:	Ascending		

Figure 3.13 : The query visual feature

SELECT `SHOPS`.`S_NAME`, `SHOPS`.`CONTACT`, `SHOPS`.`CITY`

```
FROM `SHOPS` ORDER BY `SHOPS`.`S_NAME` ASC
```

In this example, the name of the shops, the contacts and the city are displayed. An ascending sort is applied to the name. It is possible to add criteria in the query with the WHERE order.

In brief

Tables can be linked together by relationships defined at the database level. A relationship always starts from the primary key of a first table to a foreign key of a second table. They are automatically used when queries are created and enable access to all the fields in the relational schema.

Chapter 4 will deal with the functions that can be used in a SQL expression.

Chapter 4
Functions

This chapter deals with different functions of text, numerical and date types that can be used in a SQL expression.

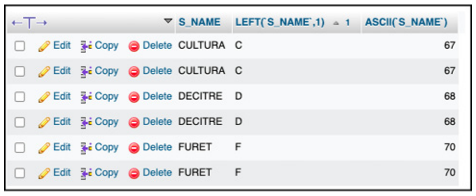
4.1 The text functions

They enable you to manipulate strings of characters inside the fields.

- *ASCII()*. Returns the ASCII numerical value of the first character of the string.

The query below displays: the name of the libraries, the first character of the name (sorted out), and the corresponding ASCII code.

SELECT \`S_NAME\`, LEFT(\`S_NAME\`,1), ASCII(\`S_NAME\`) FROM \`SHOPS\` ORDER BY \`LEFT(\`\`S_NAME\`\`,1)\` ASC



	S_NAME	LEFT(\`S_NAME\`,1) ▲ 1	ASCII(\`S_NAME\`)
Edit Copy Delete	CULTURA	C	67
Edit Copy Delete	CULTURA	C	67
Edit Copy Delete	DECITRE	D	68
Edit Copy Delete	DECITRE	D	68
Edit Copy Delete	FURET	F	70
Edit Copy Delete	FURET	F	70

Figure 4.1 : ASCII code extraction

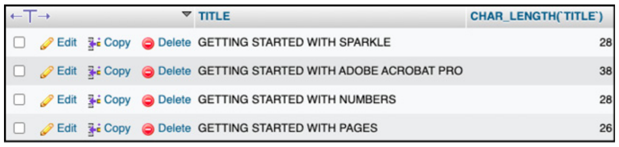
The LEFT() function extracts several characters of the field from the left.

- *TRIM()*. This function removes the invisible characters at the beginning and at the end of the string, such as space, tab, line feed or carriage return.

SELECT \`S_NAME\`, TRIM(\`S_NAME\`) FROM \`SHOPS\`

- *CHAR_LENGTH()*. Counts the number of characters in a string.

SELECT `TITLE`, CHAR_LENGTH(`TITLE`) FROM `BOOKS`



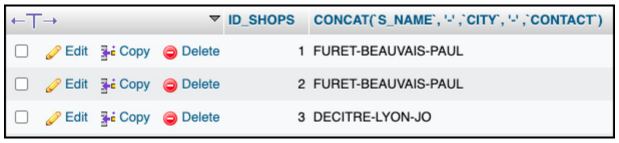
←T→	TITLE	CHAR_LENGTH(`TITLE`)
Edit Copy Delete	GETTING STARTED WITH SPARKLE	28
Edit Copy Delete	GETTING STARTED WITH ADOBE ACROBAT PRO	38
Edit Copy Delete	GETTING STARTED WITH NUMBERS	28
Edit Copy Delete	GETTING STARTED WITH PAGES	26

Figure 4.2 : Displaying the number of characters in a string

- *CONCAT()*. This function enables you to concatenate (paste) several strings together.

SELECT `ID_SHOPS`, CONCAT(`S_NAME`, '-' ,`CITY`, '-' ,`CONTACT`) FROM `SHOPS` WHERE 1;

Figure 4.3 concatenates the fields separated by a dash.



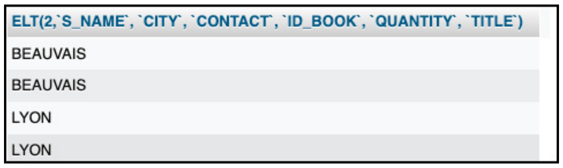
←T→	ID_SHOPS	CONCAT(`S_NAME`, '-' ,`CITY`, '-' ,`CONTACT`)
Edit Copy Delete	1	FURET-BEAUVAIS-PAUL
Edit Copy Delete	2	FURET-BEAUVAIS-PAUL
Edit Copy Delete	3	DECITRE-LYON-JO

Figure 4.3 : Concatenating of several fields

- *ELT()*. This function returns a string from a specified index.

The following code returns information based on the value 2 (that is the city) inside the fields separated by a comma:

SELECT ELT(2,`S_NAME`, `CITY`, `CONTACT`, `ID_BOOK`, `QUANTITY`, `TITLE`) FROM `SHOPS` WHERE 1;

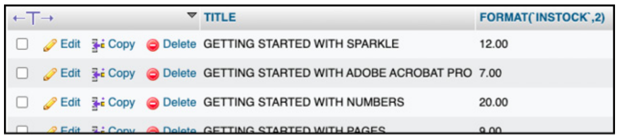


ELT(2,`S_NAME`, `CITY`, `CONTACT`, `ID_BOOK`, `QUANTITY`, `TITLE`)
BEAUVAIS
BEAUVAIS
LYON
LYON

Figure 4.4 : Retrieving information according to an index

- *FORMAT(field, decimals)*. This function formats a value according to the number of decimals.

```
SELECT `TITLE`, FORMAT(`INSTOCK`,2) FROM `BOOKS` WHERE 1;
```



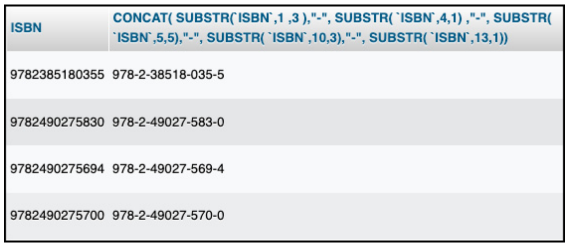
←T→	TITLE	FORMAT(`INSTOCK`,2)
☐ Edit Copy Delete	GETTING STARTED WITH SPARKLE	12.00
☐ Edit Copy Delete	GETTING STARTED WITH ADOBE ACROBAT PRO	7.00
☐ Edit Copy Delete	GETTING STARTED WITH NUMBERS	20.00
☐ Edit Copy Delete	GETTING STARTED WITH PAGES	9.00

Figure 4.5 : Formatting a field

- *SUBSTR(string, position, length,).* This function extracts a substring from a given position with a defined length.
- For instance, the following SQL expression extracts the numbers from the ISBN number then inserts a dash every 2 digits.

```
SELECT `ID_BOOKS`, `TITLE`, `ISBN`,
CONCAT( SUBSTR(`ISBN`,1 ,3 ),"-",
SUBSTR( `ISBN`,4,1) ,"-",
SUBSTR( `ISBN`,5,5),"-",
SUBSTR( `ISBN`,10,3),"-",
SUBSTR( `ISBN`,13,1))
FROM `BOOKS`;
```

Figure 4.6 shows the result.



ISBN	CONCAT(SUBSTR(`ISBN`,1 ,3),"-", SUBSTR(`ISBN`,4,1) ,"-", SUBSTR(`ISBN`,5,5),"-", SUBSTR(`ISBN`,10,3),"-", SUBSTR(`ISBN`,13,1))
9782385180355	978-2-38518-035-5
9782490275830	978-2-49027-583-0
9782490275694	978-2-49027-569-4
9782490275700	978-2-49027-570-0

Figure 4.6 : Extraction of a substring

- *LCASE(string).* This function transforms the text into lower case.

```
SELECT S_NAME, LCASE(S_NAME) FROM BOOKS
```

- *UCASE(string).* This function transforms any lower case text into upper case.

SELECT S_NAME, UCASE(S_NAME) FROM BOOKS

- *RIGHT(string,length)*. This function extracts characters from the right.

The following code extracts the last 3 characters of the name:

SELECT RIGHT(S_NAME,3) FROM BOOKS

- *LENGTH(string)*. This function returns the length of the string.

SELECT LCASE(S_NAME), RIGHT(LCASE(S_NAME),3), LENGTH(LCASE(S_NAME)) FROM BOOKS

Here, two functions are embedded.

- *REPEAT(string , number)*. Returns a string repeated a number of times.

SELECT `ID_BOOKS`, `TITLE`, REPEAT("X",5),`ISBN` FROM `BOOKS`;

- *REPLACE(input string, text to replace, replacement string)*. This function replaces alphanumeric characters in a string.

For instance, one can imagine email addresses that end with *.com*. You want replace *.com* by *.fr* using the UPDATE command.

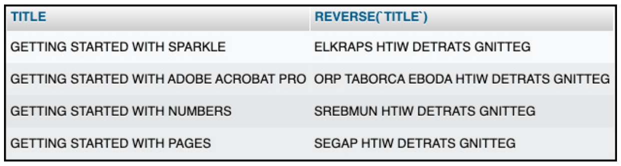
SELECT `EMAIL`, REPLACE(`EMAIL`,".COM",".FR") FROM LIBRARIES

The example below uses the UPDATE command.

UPDATE LIBRARIES SET `EMAIL2`=REPLACE(`EMAIL`,".COM",".FR")

- *REVERSE(string)*. This function reverses the direction of the letters in the whole expression.

SELECT `TITLE`, REVERSE(`TITLE`) FROM `BOOKS`;



TITLE	REVERSE(`TITLE`)
GETTING STARTED WITH SPARKLE	ELKRAPS HTIW DETRATS GNITTEG
GETTING STARTED WITH ADOBE ACROBAT PRO	ORP TABORCA EBODA HTIW DETRATS GNITTEG
GETTING STARTED WITH NUMBERS	SREBMUN HTIW DETRATS GNITTEG
GETTING STARTED WITH PAGES	SEGAP HTIW DETRATS GNITTEG

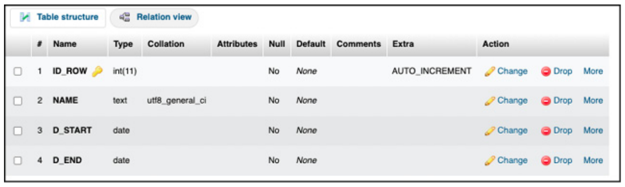
Figure 4.7 : The REVERSE function

4.2 The date functions

They enable you to manipulate both dates and times. For instance you can perform calculations such as adding a period to a date or extracting a part of a date.

4.2.1 Creating a table with dates

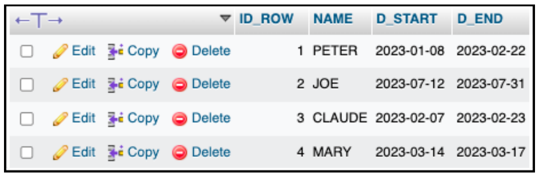
Figure 4.8 shows the structure of a table that contains the name of employee, a starting date and an ending date. Several SQL functions can manipulate these dates and perform calculations.



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	ID_ROW	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
2	NAME	text	utf8_general_ci		No	*None*			Change Drop More
3	D_START	date			No	*None*			Change Drop More
4	D_END	date			No	*None*			Change Drop More

Figure 4.8 : The training table structure

Figure 4.9 shows the data entered in this table.



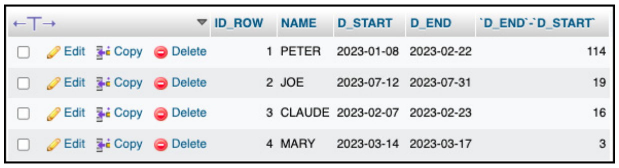
	ID_ROW	NAME	D_START	D_END
Edit Copy Delete	1	PETER	2023-01-08	2023-02-22
Edit Copy Delete	2	JOE	2023-07-12	2023-07-31
Edit Copy Delete	3	CLAUDE	2023-02-07	2023-02-23
Edit Copy Delete	4	MARY	2023-03-14	2023-03-17

Figure 4.9 : Training dates

4.2.2 Calculations with dates

<u>A) Displaying the number of days between two dates</u>

```
SELECT `ID_ROW`, `NAME`, `D_START`, `D_END`,
`D_END`-`D_START` FROM `TRAINING`;
```



←T→	ID_ROW	NAME	D_START	D_END	\`D_END\`-\`D_START\`
☐ Edit Copy Delete	1	PETER	2023-01-08	2023-02-22	114
☐ Edit Copy Delete	2	JOE	2023-07-12	2023-07-31	19
☐ Edit Copy Delete	3	CLAUDE	2023-02-07	2023-02-23	16
☐ Edit Copy Delete	4	MARY	2023-03-14	2023-03-17	3

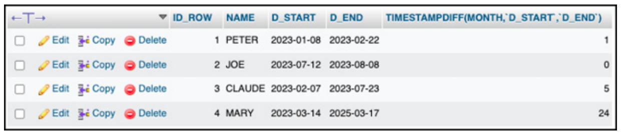
Figure 4.10 : Difference between two dates

You can also use the TIMESTAMPDIFF function to display the number of days between two dates.

SELECT \`ID_ROW\`, \`NAME\`, \`D_START\`, \`D_END\`, TIMESTAMPDIFF(DAY,\`D_START\`,\`D_END\`) FROM \`TRAINING\`;

<u>B) Displaying the number of months between two dates</u>

SELECT \`ID_ROW\`, \`NAME\`, \`D_START\`, \`D_END\`, TIMESTAMPDIFF(MONTH,\`D_START\`,\`D_END\`) FROM \`TRAINING\`;



←T→	ID_ROW	NAME	D_START	D_END	TIMESTAMPDIFF(MONTH,\`D_START\`,\`D_END\`)
☐ Edit Copy Delete	1	PETER	2023-01-08	2023-02-22	1
☐ Edit Copy Delete	2	JOE	2023-07-12	2023-08-08	0
☐ Edit Copy Delete	3	CLAUDE	2023-02-07	2023-07-23	5
☐ Edit Copy Delete	4	MARY	2023-03-14	2025-03-17	24

Figure 4.11 : Number of months between two dates

<u>C) Displaying the number of years between two dates</u>

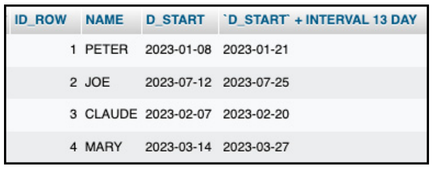
SELECT \`ID_ROW\`, \`NAME\`, \`D_START\`, \`D_END\`, TIMESTAMPDIFF(YEAR,\`D_START\`,\`D_END\`) FROM \`TRAINING\`;

Other possible parameters for the TIMESTAMPDIFF function are: SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER and YEAR.

<u>D) Adding 13 days to a date</u>

SQL proposes the expression INTERVAL with parameters to add a value to a date. Use the sign + in the formula.

SELECT \`ID_ROW\`, \`NAME\`, \`D_START\`, \`D_START\` + INTERVAL 13 DAY FROM TRAINING;



ID_ROW	NAME	D_START	\`D_START\` + INTERVAL 13 DAY
1	PETER	2023-01-08	2023-01-21
2	JOE	2023-07-12	2023-07-25
3	CLAUDE	2023-02-07	2023-02-20
4	MARY	2023-03-14	2023-03-27

Figure 4.12 : Adding a value to a date

E) Subtracting 10 days from a date

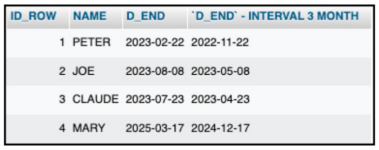
SQL proposes the expression INTERVAL with parameters to subtract a value from a date. Use the sign - in the formula.

SELECT `ID_ROW`, `NAME`, `D_START`,
`D_START` - INTERVAL 10 DAY FROM TRAINING

F) Subtracting 3 months from a date

The INTERVAL function also enables you to subtract months from a date. The following SQL expression subtracts 3 months from the date:

SELECT `ID_ROW`, `NAME`, `D_END`, `D_END` - INTERVAL 3 MONTH FROM TRAINING;



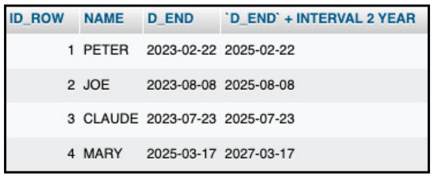
ID_ROW	NAME	D_END	`D_END` - INTERVAL 3 MONTH
1	PETER	2023-02-22	2022-11-22
2	JOE	2023-08-08	2023-05-08
3	CLAUDE	2023-07-23	2023-04-23
4	MARY	2025-03-17	2024-12-17

Figure 4.13 : Subtracting months from a date

G) Adding years to a date

In the same way, you can add years to a date with the INTERVAL function.

SELECT `ID_ROW`, `NAME`, `D_END`, `D_END` + INTERVAL 2 YEAR FROM TRAINING



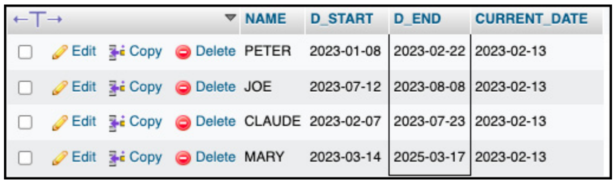
ID_ROW	NAME	D_END	`D_END` + INTERVAL 2 YEAR
1	PETER	2023-02-22	2025-02-22
2	JOE	2023-08-08	2025-08-08
3	CLAUDE	2023-07-23	2025-07-23
4	MARY	2025-03-17	2027-03-17

Figure 4.14 : Adding years to a date

4.2.3 The current date and time

Use the CURRENT_DATE function to retrieve the current date.

SELECT `NAME`, `D_START`, `D_END`, CURRENT_DATE FROM `TRAINING` WHERE `D_END`>CURRENT_DATE;

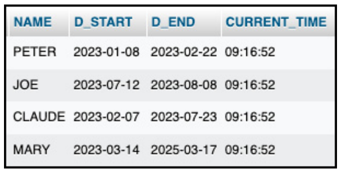


NAME	D_START	D_END	CURRENT_DATE
PETER	2023-01-08	2023-02-22	2023-02-13
JOE	2023-07-12	2023-08-08	2023-02-13
CLAUDE	2023-02-07	2023-07-23	2023-02-13
MARY	2023-03-14	2025-03-17	2023-02-13

Figure 4.15 : The current date function

The CURRENT_TIME() function returns the time.

SELECT `NAME`, `D_START`, `D_END`, CURRENT_TIME FROM `TRAINING`;

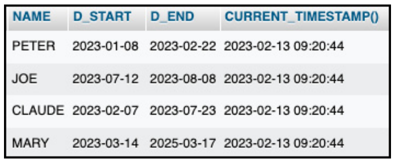


NAME	D_START	D_END	CURRENT_TIME
PETER	2023-01-08	2023-02-22	09:16:52
JOE	2023-07-12	2023-08-08	09:16:52
CLAUDE	2023-02-07	2023-07-23	09:16:52
MARY	2023-03-14	2025-03-17	09:16:52

Figure 4.16 : The current time function

The CURRENT_TIMESTAMP() function groups the two forms.

SELECT `NAME`, `D_START`, `D_END`, CURRENT_TIMESTAMP() FROM `TRAINING`;



NAME	D_START	D_END	CURRENT_TIMESTAMP()
PETER	2023-01-08	2023-02-22	2023-02-13 09:20:44
JOE	2023-07-12	2023-08-08	2023-02-13 09:20:44
CLAUDE	2023-02-07	2023-07-23	2023-02-13 09:20:44
MARY	2023-03-14	2025-03-17	2023-02-13 09:20:44

Figure 4.17 : The current_timestamp function

4.2.4 Extracting parts of the date

You can use the DATE_FORMAT function with parameters to extract several parts of the date.

The syntax is as follows:

DATE_FORMAT(CURRENT_TIMESTAMP, 'parameter')

The table below shows the parameters:

Parameter	Description

%k	The hour
i	The minute
%s	The second
%d	The day
%m	The month
%Y	The year

The following code displays the hour, minute and second of the current date:

SELECT CURRENT_TIMESTAMP,

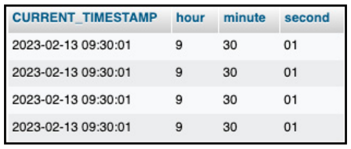
DATE_FORMAT(CURRENT_TIMESTAMP, '%k') as hour,

DATE_FORMAT(CURRENT_TIMESTAMP, '%i') as minute,

DATE_FORMAT(CURRENT_TIMESTAMP, '%s') as second

FROM `TRAINING`

The expression *as minute* is an alias i.e. an expression that renames the column.

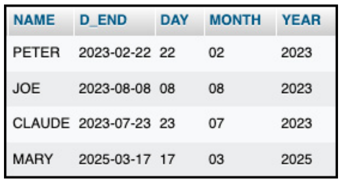


CURRENT_TIMESTAMP	hour	minute	second
2023-02-13 09:30:01	9	30	01
2023-02-13 09:30:01	9	30	01
2023-02-13 09:30:01	9	30	01
2023-02-13 09:30:01	9	30	01

Figure 4.18 : Extracting of the hour, minute and second

The following code displays the day, the month, and the year of the D_END date.

SELECT NAME,`D_END`, DATE_FORMAT(`D_END`,'%d') AS DAY, DATE_FORMAT(`D_END`,'%m') AS MONTH, DATE_FORMAT(`D_END`,'%Y') AS YEAR FROM `TRAINING`;



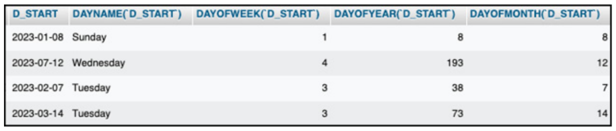
NAME	D_END	DAY	MONTH	YEAR
PETER	2023-02-22	22	02	2023
JOE	2023-08-08	08	08	2023
CLAUDE	2023-07-23	23	07	2023
MARY	2025-03-17	17	03	2025

Figure 4.19 : the day, the month and the year

4.2.5 Day of month, week and year

The functions DAYNAME, DAYOFWEEK and DAYOFYEAR return respectively the name of the day, the number of the day in the week (1 for Sunday, 2 for Monday, etc.) and the number of the day in the year.

SELECT NAME,`D_START`, DAYNAME(`D_START`), DAYOFWEEK(`D_START`), DAYOFYEAR(`D_START`), DAYOFMONTH(`D_START`) FROM TRAINING;



```
D_START     DAYNAME(`D_START`)  DAYOFWEEK(`D_START`)  DAYOFYEAR(`D_START`)  DAYOFMONTH(`D_START`)
2023-01-08  Sunday                                 1                     8                      8
2023-07-12  Wednesday                              4                   193                     12
2023-02-07  Tuesday                                3                    38                      7
2023-03-14  Tuesday                                3                    73                     14
```

Figure 4.20 : Other results

4.3 Functions about numbers

Several functions enable you to act on numerical values.

The table below details them:

Functions	Description
ABS	Returns the absolute value of the expression.
ACOS	Returns the cosine arc of a number.
ASIN	Returns the sine arc of a number.
ATAN	Returns the tangent arc of a number.
AVG	Returns the average.
CEIL	Returns the smallest integer value of a number.
COS	Returns the cosine of a number.
COT	Returns the cotangent.
COUNT	Counts a series of values.
DEGREES	Converts a radian value into degrees.
DIV	Returns the result of integer division.
EXP	Returns a value to the power.
FLOOR	Returns the largest integer value.
GREATEST	Returns the largest value of a series of arguments.
LEAST	Returns the smallest value of a series of arguments.
LN	Returns the natural logarithm of 2.
MAX	Returns the maximum value of a series of values.
MIN	Returns the minimum value of a series of values.
MOD	Returns the remainder of the division of a number by another.

PI	Returns the value of Pi.
POWER	Returns the value of a number increased to the power of another.
RADIANS	Converts a value from degrees to radians.
RAND	Returns a random number between 0 and 1.
ROUND	Returns a number rounded to a certain number of decimal places.
SIGN	Returns the sign of a number.
SIN	Returns the sine of a number.
SQRT	Returns the square root.
SUM	Calculates the sum of a series of values.
TAN	Returns the tangent of a number.
TRUNCATE	Truncates a number to a certain number of decimal places.

The following code rounds the column *price* to one decimal with the ROUND and TRUNCATE commands.

```
SELECT `order`, `price`, ROUND(`price`,1), TRUNCATE(`price`) FROM `SALES`
```

The next code calculates the sum, the average the maximum and the minimum of a column by rounding to 0 decimal places.

```
SELECT ROUND( SUM(`price`),0) , ROUND(AVG(`price`),0) , ROUND( MAX(`price`),0) , ROUND( MIN(`price`),0) FROM `SALES`
```

Column can be renamed with the *AS* expression considered as an alias.

```
SELECT
ROUND(SUM(`price`),0) AS SUM,
ROUND(AVG(`price`),0) AS AVERAGE,
ROUND(MAX(`price`),0) AS MAXIMUM,
ROUND(MIN(`price`),0) AS MINIMUM
FROM `SALES
```

4.4 The aggregate functions

These functions are used to calculate a numerical value within a column, by grouping records according to one or more criteria.

The calculation functions are the following:

- COUNT() or COUNT(*) : number of values
- SUM() : sum of values
- MAX() : maximum value
- MIN() : minimum value
- AVG() : average of the values

The structure used for the queries is :

SELECT fields

FROM table

WHERE conditions

GROUP BY fields

HAVING conditions

ORDER BY fields

The GROUP BY clause is used to group records. The COUNT function can be applied to count elements. The SUM function will be used to calculate totals, as will the AVG function that will be used to return an average of the values.

Figure 4.21 shows the structure of a table SALES with the name of the seller, his sector, the amount and the city of each sale.

The field ID_SALE is the primary key and specifies each row.

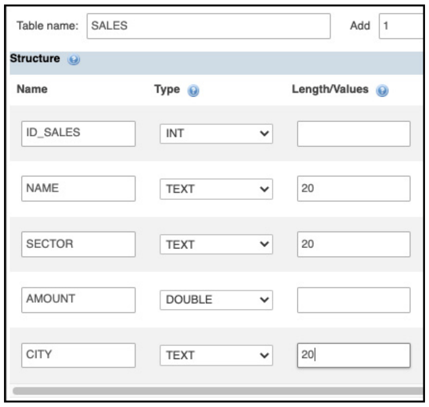


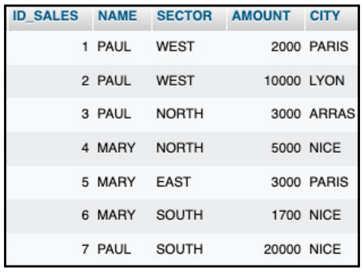
Table name: SALES Add 1

Structure

Name	Type	Length/Values
ID_SALES	INT	
NAME	TEXT	20
SECTOR	TEXT	20
AMOUNT	DOUBLE	
CITY	TEXT	20

Figure 4.21 : The structure of the table SALES

Figure 4.22 shows some data entered in this table.

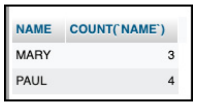


ID_SALES	NAME	SECTOR	AMOUNT	CITY
1	PAUL	WEST	2000	PARIS
2	PAUL	WEST	10000	LYON
3	PAUL	NORTH	3000	ARRAS
4	MARY	NORTH	5000	NICE
5	MARY	EAST	3000	PARIS
6	MARY	SOUTH	1700	NICE
7	PAUL	SOUTH	20000	NICE

Figure 4.22 : The data

The following query counts the number of rows by name:

SELECT NAME, COUNT(`NAME`)FROM `SALES` GROUP BY `NAME`;

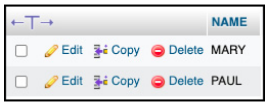


NAME	COUNT(`NAME`)
MARY	3
PAUL	4

Figure 4.23 : Number of rows by name

The following query displays the name of each seller:

SELECT NAME FROM `SALES` GROUP BY `NAME`;



←T→	NAME
Edit Copy Delete	MARY
Edit Copy Delete	PAUL

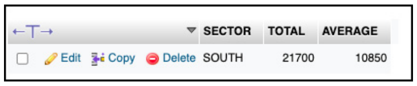
Figure 4.24 : The name of the sellers

The following query displays the sum and the average of amounts grouping by sector and in the result, the average must be greater or equal than 6000 and the sector equal to 'SOUTH':

SELECT `SECTOR`, SUM(AMOUNT) AS TOTAL, AVG(AMOUNT) AS AVERAGE FROM `SALES` GROUP BY `SECTOR` HAVING (AVERAGE >= 6000 AND SECTOR LIKE 'SOUTH');

The query uses the HAVING clause to test values inside a result set.

Figure 4.25 shows the result.

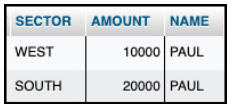


←T→	SECTOR	TOTAL	AVERAGE
Edit Copy Delete	SOUTH	21700	10850

Figure 4.25 : Grouping by sectors and testing a result set

The following query selects a first result set that will be used inside a WHERE clause. Here, two SELECT are embedded:

SELECT `SECTOR`, `AMOUNT`, `NAME` FROM SALES WHERE `AMOUNT`> (SELECT AVG(`AMOUNT`) FROM SALES);



SECTOR	AMOUNT	NAME
WEST	10000	PAUL
SOUTH	20000	PAUL

Figure 4.26 : Two SELECT expressions are embedded.

In brief

In this chapter you have studied text, numerical and date functions that you can use in SQL queries. Aggregate functions enable you to group information by using the GROUP BY clause.

Chapter 5 will expose advanced SQL queries.

Chapter 5
Advanced SQL Queries

This chapter focuses on complex SQL expressions that can enhance your productivity and your skills.

5.1 A CASE expression in a SELECT

You can embed a CASE condition within a SELECT expression to retrieve results based on multiple conditions.

The following SQL expression tests whether sales are greater than several values by returning a text corresponding to the status of the salespeople:

The CASE expression begins the series of conditional tests and finishes with the END command. The result of the condition is assigned to an alias.

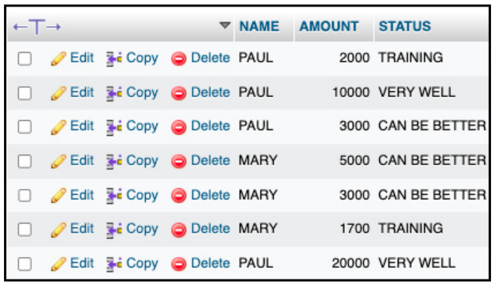
```
SELECT `NAME`,`AMOUNT`,

CASE WHEN `AMOUNT` >5000 THEN 'VERY WELL'

WHEN `AMOUNT` >2000 THEN 'CAN BE BETTER'

ELSE 'TRAINING'

END AS STATUS FROM SALES;
```



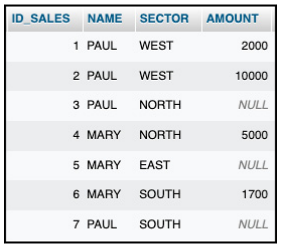
	NAME	AMOUNT	STATUS
Edit Copy Delete	PAUL	2000	TRAINING
Edit Copy Delete	PAUL	10000	VERY WELL
Edit Copy Delete	PAUL	3000	CAN BE BETTER
Edit Copy Delete	MARY	5000	CAN BE BETTER
Edit Copy Delete	MARY	3000	CAN BE BETTER
Edit Copy Delete	MARY	1700	TRAINING
Edit Copy Delete	PAUL	20000	VERY WELL

Figure 5.1 : A testing CASE expression

5.2 The COALESCE function and NULL values

This very powerful function enables you to replace NULL values with other values. The following query replaces NULL values with the value 0 in the sales column:

Figure 5.2 shows the field AMOUNT containing NULL values.

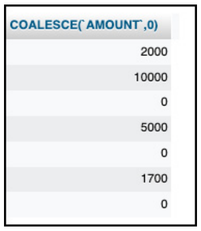


ID_SALES	NAME	SECTOR	AMOUNT
1	PAUL	WEST	2000
2	PAUL	WEST	10000
3	PAUL	NORTH	NULL
4	MARY	NORTH	5000
5	MARY	EAST	NULL
6	MARY	SOUTH	1700
7	PAUL	SOUTH	NULL

Figure 5.2 : NULL values in a field

The code below displays 0 in place of NULL.

SELECT COALESCE(`AMOUNT`,0) FROM SALES



COALESCE(`AMOUNT`,0)
2000
10000
0
5000
0
1700
0

Figure 5.3 : NULL are replaced with 0

Another way to perform this operation is with a SELECT CASE expression.

SELECT CASE

WHEN `AMOUNT` IS NULL THEN 0

ELSE `AMOUNT`

END

FROM SALES

It is much more interesting to use the COALESCE function.

To change the value inside the table, use the UPDATE clause.

UPDATE SALES SET AMOUNT = COALESCE(`AMOUNT`,0)

5.3 More complex sorting

In the previous chapters you have learned that sorting a SQL query is done using the ORDER BY clause. It can also be applied to text, date or numerical values in an ascending (ASC) or a descending (DESC) direction.

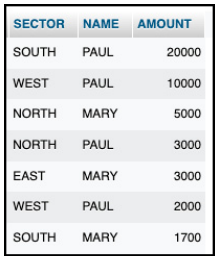
5.3.1 Sorting with an index

It is not necessary to specify the name of the column used as a criterion to sort out a field because you can specify a number that corresponds to its position in the SELECT clause starting from 1.

For instance, the following expression sorts the fields by the third column:

SELECT `SECTOR`,`NAME`,`AMOUNT` FROM SALES ORDER BY 3 DESC;

Figure 5.4 shows the result.



SECTOR	NAME	AMOUNT
SOUTH	PAUL	20000
WEST	PAUL	10000
NORTH	MARY	5000
NORTH	PAUL	3000
EAST	MARY	3000
WEST	PAUL	2000
SOUTH	MARY	1700

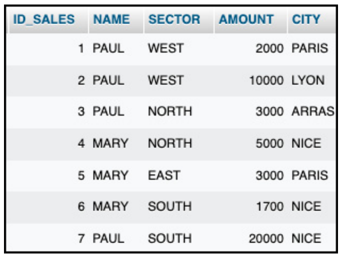
Figure 5.4 : Sorting according to an index

5.3.2 Sorting with several columns

You can also sort with several columns. To do this, separate the fields by commas. The priority is defined from left to right.

The SQL code below sorts the rows of the table according to three fields.

SELECT `SECTOR`,`CITY`,`AMOUNT`,`NAME` FROM SALES ORDER BY `SECTOR`,`CITY`,`AMOUNT` DESC;



ID_SALES	NAME	SECTOR	AMOUNT	CITY
1	PAUL	WEST	2000	PARIS
2	PAUL	WEST	10000	LYON
3	PAUL	NORTH	3000	ARRAS
4	MARY	NORTH	5000	NICE
5	MARY	EAST	3000	PARIS
6	MARY	SOUTH	1700	NICE
7	PAUL	SOUTH	20000	NICE

Figure 5.5 : Sorting with three fields

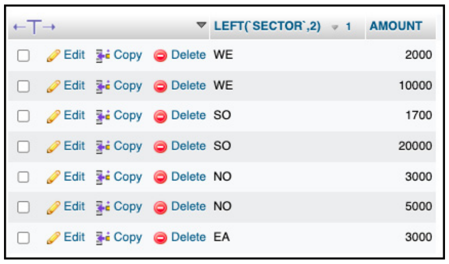
5.3.3 Sorting by substrings

It is quite possible to sort by substrings using the string manipulation functions.

The following code sorts by the first two characters of the field SECTOR.

SELECT LEFT(`SECTOR`,2),`AMOUNT` FROM SALES

ORDER BY LEFT(`SECTOR`,2) DESC

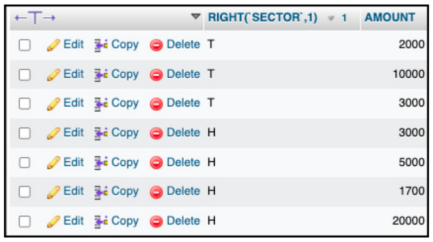


	LEFT(`SECTOR`,2)	AMOUNT
Edit Copy Delete	WE	2000
Edit Copy Delete	WE	10000
Edit Copy Delete	SO	1700
Edit Copy Delete	SO	20000
Edit Copy Delete	NO	3000
Edit Copy Delete	NO	5000
Edit Copy Delete	EA	3000

Figure 5.6 : Sorting by two first characters

The following code sorts by the last character of the field SECTOR from the right.

SELECT RIGHT(`SECTOR`,1), `AMOUNT` FROM SALES ORDER BY RIGHT(`SECTOR`,1) DESC

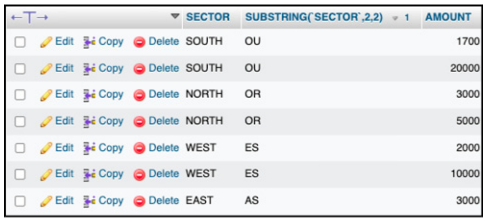


	RIGHT(`SECTOR`,1)	AMOUNT
Edit Copy Delete	T	2000
Edit Copy Delete	T	10000
Edit Copy Delete	T	3000
Edit Copy Delete	H	3000
Edit Copy Delete	H	5000
Edit Copy Delete	H	1700
Edit Copy Delete	H	20000

Figure 5.7 : Sorting by the last character

The following code sorts by two characters of the field starting at the 2nd character.

SELECT `SECTOR`, SUBSTRING(`SECTOR`,2,2),`AMOUNT` FROM SALES ORDER BY SUBSTRING(`SECTOR`,2,2) DESC;



	SECTOR	SUBSTRING(`SECTOR`,2,2)	AMOUNT
Edit Copy Delete	SOUTH	OU	1700
Edit Copy Delete	SOUTH	OU	20000
Edit Copy Delete	NORTH	OR	3000
Edit Copy Delete	NORTH	OR	5000
Edit Copy Delete	WEST	ES	2000
Edit Copy Delete	WEST	ES	10000
Edit Copy Delete	EAST	AS	3000

Figure 5.8 : Sorting by a substring

5.4 Ensemble queries

With SQL, you can act with two tables using the following set operators: UNION, INTERSECT and MINUS or EXCEPT.

- *UNION*. The union query enables you to join information from two tables end to end, systematically eliminating duplicates.
- *INTERSECT*. This command allows you to obtain the intersection of the results of two queries by retrieving the common records. The two queries must have the same number of columns and same type of fields presentation.
- *MINUS*. Retrieves the records of a first query without including the results of a second query.

5.4.1 The UNION operator

The syntax is as follows:

SELECT field1, field2, field3, etc

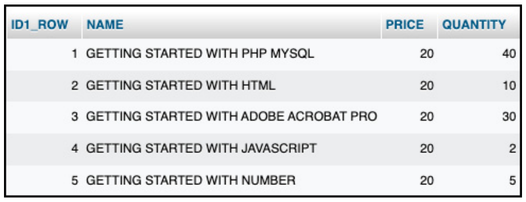
FROM Table1

UNION

SELECT field1, field2, field3, etc

FROM Table2

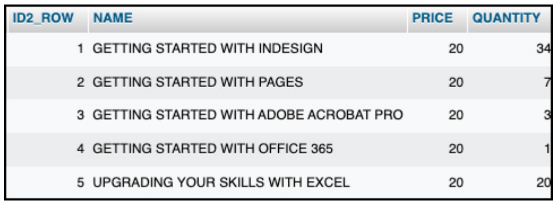
Figure 5.9 shows a table BOOKS2020 that contains book titles.



ID1_ROW	NAME	PRICE	QUANTITY
1	GETTING STARTED WITH PHP MYSQL	20	40
2	GETTING STARTED WITH HTML	20	10
3	GETTING STARTED WITH ADOBE ACROBAT PRO	20	30
4	GETTING STARTED WITH JAVASCRIPT	20	2
5	GETTING STARTED WITH NUMBER	20	5

Figure 5.9 : The table BOOK2020

Figure 5.10 shows a table BOOKS2021 that contains other book titles.



ID2_ROW	NAME	PRICE	QUANTITY
1	GETTING STARTED WITH INDESIGN	20	34
2	GETTING STARTED WITH PAGES	20	7
3	GETTING STARTED WITH ADOBE ACROBAT PRO	20	3
4	GETTING STARTED WITH OFFICE 365	20	1
5	UPGRADING YOUR SKILLS WITH EXCEL	20	20

Figure 5.10 : The table BOOK2021

The two tables have the same structure and the same field names except the primary key. Only one book is common to both tables (GETTING STARTED WITH ADOBE ACROBAT PRO).

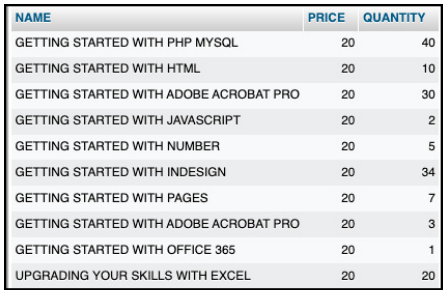
To group all the book titles from the two tables, a UNION query is needed.

```
SELECT `NAME`,`PRICE`,`QUANTITY` FROM BOOK2020

UNION

SELECT `NAME`,`PRICE`,`QUANTITY` FROM BOOK2021
```

Figure 5.11 shows the result:



NAME	PRICE	QUANTITY
GETTING STARTED WITH PHP MYSQL	20	40
GETTING STARTED WITH HTML	20	10
GETTING STARTED WITH ADOBE ACROBAT PRO	20	30
GETTING STARTED WITH JAVASCRIPT	20	2
GETTING STARTED WITH NUMBER	20	5
GETTING STARTED WITH INDESIGN	20	34
GETTING STARTED WITH PAGES	20	7
GETTING STARTED WITH ADOBE ACROBAT PRO	20	3
GETTING STARTED WITH OFFICE 365	20	1
UPGRADING YOUR SKILLS WITH EXCEL	20	20

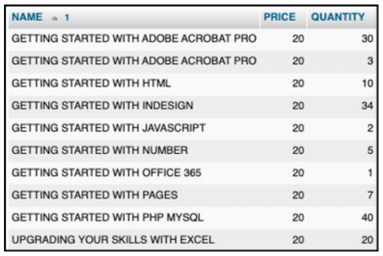
Figure 5.11 : The result of a UNION query

The UNION query can be sorted as follows:

```
SELECT `NAME`,`PRICE`,`QUANTITY` FROM BOOK2020

UNION

SELECT `NAME`,`PRICE`,`QUANTITY` FROM BOOK2021 ORDER BY `NAME`
```



NAME ▲ 1	PRICE	QUANTITY
GETTING STARTED WITH ADOBE ACROBAT PRO	20	30
GETTING STARTED WITH ADOBE ACROBAT PRO	20	3
GETTING STARTED WITH HTML	20	10
GETTING STARTED WITH INDESIGN	20	34
GETTING STARTED WITH JAVASCRIPT	20	2
GETTING STARTED WITH NUMBER	20	5
GETTING STARTED WITH OFFICE 365	20	1
GETTING STARTED WITH PAGES	20	7
GETTING STARTED WITH PHP MYSQL	20	40
UPGRADING YOUR SKILLS WITH EXCEL	20	20

Figure 5.12 : The result of a UNION query can be sorted

5.4.2 Creating a table with the UNION operator

To extract the data from a UNION query result to a new table, you can use the CREATE TABLE AS clause, as shown in the query below:

```
CREATE TABLE THE_GROUP AS

SELECT `NAME`,`PRICE`,`QUANTITY` FROM BOOK2020

UNION

SELECT `NAME`,`PRICE`,`QUANTITY` FROM BOOK2021
```

In this example, a new table called THE_GROUP will be created and will contain all the data coming from the two tables.

5.4.3 The MINUS operator

The SQL MINUS operator is used to return all rows in a first SELECT statement that do not exist in a table 2. In other words, the MINUS operator removes the results from the second dataset.

The syntax is as follows:

```
SELECT * FROM table1

MINUS
```

SELECT * FROM table2

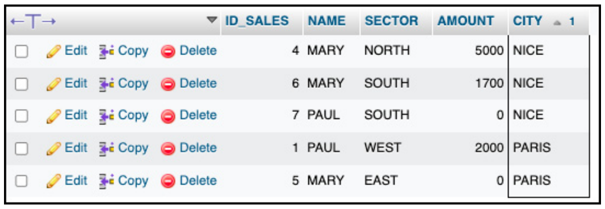
The MINUS does not work in every database system.

5.4.4 The IN operator

The IN operator enables you to retrieve values based on a choice. For instance, the following SQL query retrieves records only for the cities PARIS and NICE from the table SALES.

SELECT * FROM SALES WHERE `CITY` IN ('PARIS','NICE')

ORDER BY `CITY`



	ID_SALES	NAME	SECTOR	AMOUNT	CITY
Edit Copy Delete	4	MARY	NORTH	5000	NICE
Edit Copy Delete	6	MARY	SOUTH	1700	NICE
Edit Copy Delete	7	PAUL	SOUTH	0	NICE
Edit Copy Delete	1	PAUL	WEST	2000	PARIS
Edit Copy Delete	5	MARY	EAST	0	PARIS

Figure 5.13 : The operator IN

The following query shows the opposite with the NOT operator:

SELECT * FROM SALES WHERE `CITY` NOT IN ('PARIS','NICE')

The IN operator is equivalent to the OR operator.

SELECT * FROM SALES WHERE (`CITY` = 'PARIS' OR `CITY` = 'NICE')

and its opposite:

SELECT * FROM SALES WHERE NOT (`CITY` = 'PARIS' OR `CITY` = 'NICE')

5.5 Joining several tables

When tables have common values, you can join them inside a query thanks to a primary key that points to a foreign key. The primary key has unique values and the foreign key has duplicated values.

Figure 5.14 shows the structure of a table called PEOPLE that contains two fields: THENAME and SPORT. The first field is the primary key. Both the fields are of type VARCHAR.

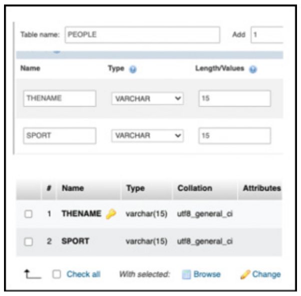


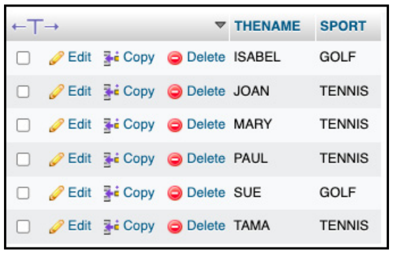
Table name: PEOPLE   Add 1

Name	Type	Length/Values
THENAME	VARCHAR	15
SPORT	VARCHAR	15

	#	Name	Type	Collation	Attributes
☐	1	THENAME	varchar(15)	utf8_general_ci	
☐	2	SPORT	varchar(15)	utf8_general_ci	

Figure 5.14 : The table PEOPLE structure

Figure 5.15 shows the data.



	THENAME	SPORT
Edit Copy Delete	ISABEL	GOLF
Edit Copy Delete	JOAN	TENNIS
Edit Copy Delete	MARY	TENNIS
Edit Copy Delete	PAUL	TENNIS
Edit Copy Delete	SUE	GOLF
Edit Copy Delete	TAMA	TENNIS

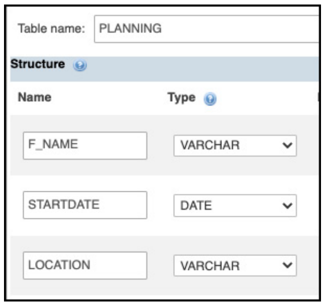
Figure 5.15 : The table data

For each person, several sport trainings are scheduled. A training starting date and a location will be defined. Because there are several trainings, a second table PLANNING will store the dates and the location.

The structure of this second table contains the following fields:

- *F_NAME*. It is the name of the person and it is also a foreign key that will be defined later in order to join the two tables.
- *STARTDATE*. It is the starting date of the training.
- *LOCATION*. It is the location of the training

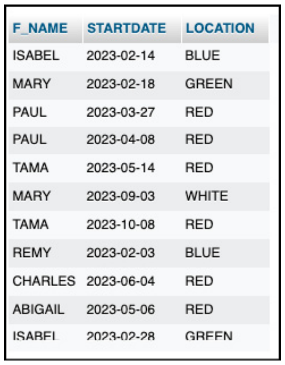
Figure 5.16 shows the structure of the table PLANNING.



Name	Type
F_NAME	VARCHAR
STARTDATE	DATE
LOCATION	VARCHAR

Figure 5.16 : The structure of the second table

Figure 5.17 shows the data inside this second table.



F_NAME	STARTDATE	LOCATION
ISABEL	2023-02-14	BLUE
MARY	2023-02-18	GREEN
PAUL	2023-03-27	RED
PAUL	2023-04-08	RED
TAMA	2023-05-14	RED
MARY	2023-09-03	WHITE
TAMA	2023-10-08	RED
REMY	2023-02-03	BLUE
CHARLES	2023-06-04	RED
ABIGAIL	2023-05-06	RED
ISABEL	2023-02-28	GREEN

Figure 5.17 : The second table

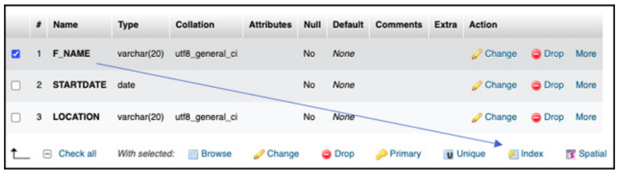
In this table, two names (ABIGAIL and CHARLES) are not in the first table. Thanks to the LEFT JOINT and the RIGHT JOINT, SQL allows you to easily control the content between the two tables.

5.5.1 Creating a foreign key

To join two tables in a SQL query, a primary key (from the first table) must point to a foreign key (from a second table).

To create a such foreign key, follow the steps below:

- Display the structure of the second table PLANNING.
- Check the field F_NAME.
- Click on the *Index* link (figure 5.18).

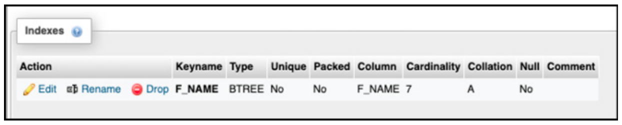


	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
✓	1	F_NAME	varchar(20)	utf8_general_ci		No	*None*			Change Drop More
☐	2	STARTDATE	date			No	*None*			Change Drop More
☐	3	LOCATION	varchar(20)	utf8_general_ci		No	*None*			Change Drop More

Figure 5.18 : Creating an index

- Click on the *Go* link to finish.

Figure 5.19 shows the index settings. In this case, the column *Null* is specified to *No.* It means that you cannot enter an empty value for the name.



Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	F_NAME	BTREE	No	No	F_NAME	7	A	No	

Figure 5.19 : The index settings

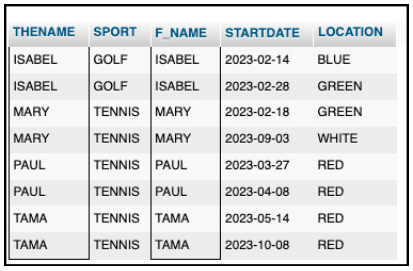
Once the index has been defined, you can create queries with relations between tables.

5.5.2 Finding identical names

The next query extracts information from two tables by establishing a relation from the primary key to the foreign key, using the WHERE clause. The result will display the same names.

SELECT PEOPLE.THENAME, PEOPLE.SPORT, PLANNING.F_NAME, PLANNING.STARTDATE, PLANNING.LOCATION FROM PEOPLE,PLANNING WHERE PEOPLE.THENAME=PLANNING.F_NAME

Figure 5.20 shows the result from both tables.



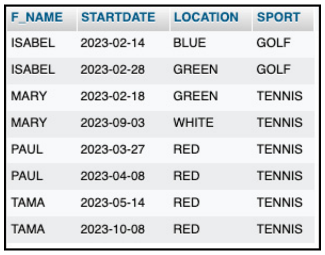
THENAME	SPORT	F_NAME	STARTDATE	LOCATION
ISABEL	GOLF	ISABEL	2023-02-14	BLUE
ISABEL	GOLF	ISABEL	2023-02-28	GREEN
MARY	TENNIS	MARY	2023-02-18	GREEN
MARY	TENNIS	MARY	2023-09-03	WHITE
PAUL	TENNIS	PAUL	2023-03-27	RED
PAUL	TENNIS	PAUL	2023-04-08	RED
TAMA	TENNIS	TAMA	2023-05-14	RED
TAMA	TENNIS	TAMA	2023-10-08	RED

Figure 5.20 : Data coming from two tables

The SQL query below displays the fields using another syntax with the INNER JOIN expression.

```
SELECT F_NAME, STARTDATE, LOCATION, PEOPLE.SPORT
FROM PLANNING
INNER JOIN PEOPLE
ON PEOPLE.THENAME=PLANNING.F_NAME
```

Figure 5.21 shows the result.



F_NAME	STARTDATE	LOCATION	SPORT
ISABEL	2023-02-14	BLUE	GOLF
ISABEL	2023-02-28	GREEN	GOLF
MARY	2023-02-18	GREEN	TENNIS
MARY	2023-09-03	WHITE	TENNIS
PAUL	2023-03-27	RED	TENNIS
PAUL	2023-04-08	RED	TENNIS
TAMA	2023-05-14	RED	TENNIS
TAMA	2023-10-08	RED	TENNIS

Figure 5.21 : Linking data with INNER JOIN

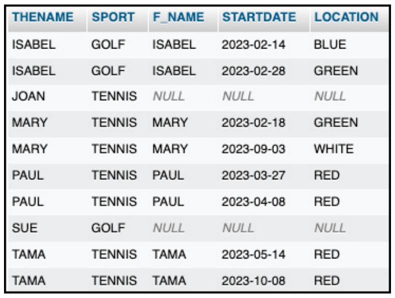
Such a relation (or joint) finds the values of the keys common to both tables either with the WHERE clause or with the INNER JOIN expression.

5.5.3 The LEFT OUTER JOIN operator

If you want to search for information from two tables, taking all the keys from the left table, you must write the following query with the LEFT OUTER JOIN ON clause:

```
SELECT PEOPLE.THENAME, PEOPLE.SPORT, PLANNING.F_NAME,
PLANNING.STARTDATE, PLANNING.LOCATION
FROM PEOPLE LEFT OUTER JOIN PLANNING
ON PEOPLE.THENAME=PLANNING.F_NAME
```

Figure 5.22 shows the data that does not match between the two tables. The result is a *NULL* value in the column.



THENAME	SPORT	F_NAME	STARTDATE	LOCATION
ISABEL	GOLF	ISABEL	2023-02-14	BLUE
ISABEL	GOLF	ISABEL	2023-02-28	GREEN
JOAN	TENNIS	NULL	NULL	NULL
MARY	TENNIS	MARY	2023-02-18	GREEN
MARY	TENNIS	MARY	2023-09-03	WHITE
PAUL	TENNIS	PAUL	2023-03-27	RED
PAUL	TENNIS	PAUL	2023-04-08	RED
SUE	GOLF	NULL	NULL	NULL
TAMA	TENNIS	TAMA	2023-05-14	RED
TAMA	TENNIS	TAMA	2023-10-08	RED

Figure 5.22 : The result from a LEFT OUTER JOIN clause

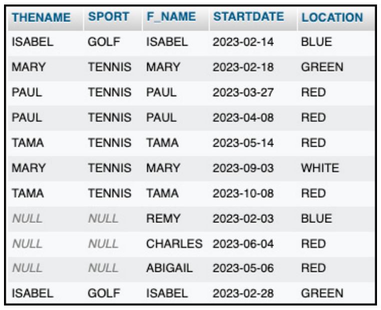
SQL considers the primary key (THENAME) and searches for the matching foreign key (F_name). If it is not the case, a NULL value is displayed.

5.5.4 The RIGHT OUTER JOIN operator

In the opposite direction, SQL will display all the foreign keys and searches the primary keys that match.

SELECT PEOPLE.THENAME, PEOPLE.SPORT, PLANNING.F_NAME, PLANNING.STARTDATE, PLANNING.LOCATION
FROM PEOPLE RIGHT OUTER JOIN PLANNING
ON PEOPLE.THENAME=PLANNING.F_NAME

Figure 5.23 shows the result where all the foreign keys
(F_NAME) are displayed and the primary keys that match (THENAME). The value NULL is set to the contrary.



THENAME	SPORT	F_NAME	STARTDATE	LOCATION
ISABEL	GOLF	ISABEL	2023-02-14	BLUE
MARY	TENNIS	MARY	2023-02-18	GREEN
PAUL	TENNIS	PAUL	2023-03-27	RED
PAUL	TENNIS	PAUL	2023-04-08	RED
TAMA	TENNIS	TAMA	2023-05-14	RED
MARY	TENNIS	MARY	2023-09-03	WHITE
TAMA	TENNIS	TAMA	2023-10-08	RED
NULL	NULL	REMY	2023-02-03	BLUE
NULL	NULL	CHARLES	2023-06-04	RED
NULL	NULL	ABIGAIL	2023-05-06	RED
ISABEL	GOLF	ISABEL	2023-02-28	GREEN

Figure 5.23 : The result from a RIGHT OUTER JOIN clause

These features can be very interesting in case errors are found in the database between primary and foreign keys that do not match.

5.6 Deletions

SQL enables you to delete records but also other objects like table constraints.

The following examples show you how to perform deletions:

5.6.1 Removing records in a table

Use the DELETE order to remove records from a table. Other clauses are possible in the SQL query

The following example deletes all the records for which the date is less than '2023-02-01'

```
DELETE FROM PLANNING WHERE `STARTDATE` < '2023-02-01'
```

5.6.2 Deleting a table

The DROP command allows you to delete a table inside the database. The following code deletes the table
THE_GROUP in the PROSPECT database

```
DROP TABLE `PROSPECT`.`THE_GROUP`
```

5.5.3 Deleting an index in a table

The ALTER command is used with a DROP command to delete an index.

```
ALTER TABLE PLANNING DROP INDEX `F_NAME`
```

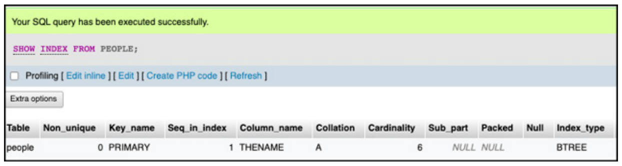
5.7 Enumerating objects in a table

SQL enables you to display embedded information in tables, such as index or key names.

The following code shows the index from the table PEOPLE:

```
SHOW INDEX FROM PEOPLE
```

Figure 5.24 shows the result.



Your SQL query has been executed successfully.

```
SHOW INDEX FROM PEOPLE;
```

Profiling [Edit inline] [Edit] [Create PHP code] [Refresh]
Extra options

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
people	0	PRIMARY	1	THENAME	A	6	NULL	NULL		BTREE

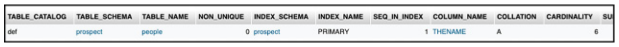
Figure 5.24 : A list of indexes

You can display other parameters with the feature called INFORMATION_SCHEMA.STATISTICS.

SELECT * FROM INFORMATION_SCHEMA.STATISTICS

WHERE table_name = 'PEOPLE'

Figure 5.25 shows the result:



TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	NON_UNIQUE	INDEX_SCHEMA	INDEX_NAME	SEQ_IN_INDEX	COLUMN_NAME	COLLATION	CARDINALITY	SU
def	prospect	people	0	prospect	PRIMARY	1	THENAME	A	6	

Figure 5.25 : Statistics information

In brief

A CASE condition that can be applied to a SELECT query. The NULL values can be managed with the COALESCE function. It is possible to sort data with substring functions. The UNION query allows you to retrieve information from two tables with the same structure. The JOIN clause permits to link a primary key to a foreign key in many ways.

Glossary

ACID	Acronym for Atomicity, Consistency, Isolation and Durability. It concerns transactions in a database.
AGREGAT	Information returned by a statistical calculation and grouped by another criteria.
ALIAS	Enables a column to be renamed for greater clarity.
ASC	Enables a column to be sorted in the ascending direction.
ASCII	American Standard Code for Information Interchange. It is an American standard used for coding alphanumeric characters. ASCII codes are essential for computer keyboards.
ATTRIBUT	A column in a table.
DATABASE	It is a structured set of data stored on a server or computer. A database is intended to meet the needs of users in a context of sharing and security.
CARDINALITY	Number of rows in a table.
CATALOG	It is a collection of SQL schemas.
CLAUSE	A command in a SELECT query.
CLIENT	It is a process that establishes a session to communicate with a database.
CLUSTER	It is a group of networks or nodes that distribute a workload.
COMMIT	To commit a transaction.
CONSTRAINT	It is a type of constraint defined on a table. For instance, you can restrict the values allowed in a column.
DDL	Data Definition Language. It is the acronym that designates all SQL commands that can modify the structure of tables (CREATE TABLE, ALTER, DROP, INDEX, etc).
DELETE	To delete records in a table.
DML	Data Manipulation Language. It is the set of SQL commands that manipulate records in a table (INSERT, UPDATE, DELETE, SELECT, etc).
DSC	Enables a column to be sorted in the downward direction.
FOREIGN KEY	It is a constraint defined on one or more columns and used to create a relationship between two tables, in the context of a one-to-many relationship.
FROM	Enables you to define the tables that are involved in the SQL query.
FUNCTIONS	These are functions that enable you to act on characters, dates or numbers. The functions can be used in a query.
GRANT	Enables you to grant rights to users or applications. This feature is part of the toolkit to enhance the security of the database.
GROUP BY	Enables you to group records based on a common criterion.
INSERT	Inserts information into a table.
INTEGRITY REFERENTIAL	This value integrity mechanism intervenes between a primary key and a foreign key as part of a constraint. You can restrict the modification or deletion of their values.
ORDER BY	Enables you to sort columns.
OUTER JOIN	Enables you to create joins between tables via the primary and foreign key pair. It can be an equal join or an outer join, right or left.
PL/SQL	Procedural Language / Structured Query Language. It is the proprietary procedural language of

	Oracle databases.
PRIMARY KEY	It is a table constraint that guarantees the uniqueness of the row. It can be auto-incremented and is used in joins between tables.
PROCEDURE	A stored procedure enables you to save a group of SQL commands. One can re-use the code that has been created.
RELATION	It is a logical link between two tables through a foreign key and a primary key. Relationships can also be set up by joins.
REQUEST	It is a request formulated in a language to provide a result. It can also be a command to the server in order to perform a task.
REVOKE	Enables you to revoke rights to users or applications.
ROLLBACK	Cancel a transaction.
SCHEMA	It is the objects organization in the database.
SELECT	It is the main order when retrieving data from one or more tables.
SERVER	A group of applications that provide access to information in a multi-user context.
RDBMS	Relational Database Management System. It contains tables and information that can be linked through the SQL language.
SQL	Structured Query Language. It contains a small set of instructions capable of querying data on multiple platforms within a relational database management system.
TABLE	It is a data structure containing a collection of information of the same kind. A table contains rows or records.
TRANSACTION	A set of SQL code that determines actions that are completely independent of other transactions.
TUPPLE	A record organized into columns also called attributes.
UPDATE	Enables you to update information in a table.
VIEW	It is a virtual table created with the SELECT command. It can be saved for re-use.
WHERE	Enables you to define logical conditions to retrieve a specific result set.

I hope that this book about the SQL language has met your expectations and will enable you to manage your future relational databases.

Do not hesitate to contact me at REMYLENT@GMAIL.COM if you have any comments or questions.

I will be sure to answer you.