CHAPTER 37

Importing and Exporting Data

Back in the Stone Age, cavemen never really had any issues with data incompatibility, as slabs of rock and one's own memory were the only storage media. Copying data involved pulling out the old chisel and getting busy on a new piece of granite. Of course, these days the situation is much different, as hundreds of data storage solutions exist. For instance, how would one go about converting data found in a PostgreSQL table into a format suitable for viewing in a spreadsheet, or vice versa? If this is done in a non-optimal fashion, you could spend hours, and even days or weeks, massaging the converted data into a usable format. It's unlikely the marketing department or company president is going to be willing to wait more than a few minutes for such data, much less want to put in a special request to have it prepared for them.

So how can you programmatically create mechanisms for easily importing and exporting data into other formats? In this chapter, you'll learn how to do so with ease, using a variety of SQL commands, PostgreSQL-specific commands, and programming techniques. Specifically, this chapter introduces the following topics:

- **PostgreSQL's `COPY` Command**: PostgreSQL's `COPY` command and its PHP equivalents, `pg_copy_to()` and `pg_copy_from()`, make importing and exporting table data a snap. You'll see how to accomplish these tasks both from the command line and from a PHP script.
- **Importing and exporting data with phpPgAdmin**: phpPgAdmin offer user-friendly yet powerful tools for easily importing and exporting data without having to jump through programmatic hoops.

Note In Chapter 26, you learned about several of PostgreSQL's backup- and recovery-related utilities, including `pg_dump`, `pg_dumpall`, and `pg_restore`, that are capable of helping you to eliminate these issues. However, these commands are generally most efficiently used when importing data from and restoring data to a PostgreSQL database, rather than readying it for use within another data manager or viewer.

The COPY Command

The `COPY` command is a PostgreSQL-specific command used to quickly copy data between a database table and a file. This section introduces the syntax necessary both for copying data from a table to a file, and vice versa. The section that follows shows you how to execute `COPY` from a PHP script using the `pg_copy_from()` and `pg_copy_to()` functions, first introduced in Chapter 30.

Copying Data to and from a Table

Copying data from a table to a text file or standard output is accomplished using the `COPY` *tablename* `TO {`*filename* `| STDOUT}` variant of the `COPY` command. The complete syntax follows:

```
COPY tablename [(column [, ...])]
   TO {'filename' | STDOUT}
   [[WITH]
      [BINARY]
      [OIDS]
      [DELIMITER [AS] 'delimiter']
      [NULL [AS] 'null string']
      [CSV [HEADER]
         [QUOTE [AS] 'quote']
         [ESCAPE [AS] 'escape']
         [FORCE NOT NULL column [, ...]]
```

Copying data residing in a text file to a table or standard output is accomplished using the same syntax as that for copying from a table, except for three slight variations of the syntax. These changes are bolded in the syntax that follows:

```
COPY tablename [(column [, ...])]
   FROM {'filename' | STDIN}
   [[WITH]
      [BINARY]
      [OIDS]
      [DELIMITER [AS] 'delimiter']
      [NULL [AS] 'null string']
      [CSV [HEADER]
         [QUOTE [AS] 'quote']
         [ESCAPE [AS] 'escape']
         [FORCE QUOTE column [, ...]]
```

As you can see, `COPY` has quite a bit to offer. Perhaps the best way to understand its many capabilities is through several examples.

Copying Data from a Table

To begin, let's dump data from a table containing employee information to standard output:

```
psql>COPY employee TO STDOUT;
```

This returns the following:

```
1       JG100011        Jason Gilmore        jason@example.com
2       RT435234        Robert Treat         rob@example.com
3       GS998909        Greg Sabino Mullane  greg@example.com
4       MW777983        Matt Wade            matt@example.com
```

To redirect this output to a file, simply specify a filename, like so:

```
psql>COPY employee TO '/home/jason/sqldata/employee.sql';
```

Keep in mind that the PostgreSQL daemon user will require the necessary privileges for writing to the specified directory. Also, an absolute path is required, because COPY will not accept relative pathnames.

■ **Note** On Windows, forward slashes should be used to specify the absolute path. So, for example, to COPY data to PostgreSQL's data directory, the path might look like c:/pgsql/data/employee.sql.

Copying Data from a Text File

Copying data from a text file to a table is as simple as copying data to it. Let's begin by importing the employee data dumped to employee.sql in the earlier example into an identical but newly named and empty employee table:

```
psql>COPY employeenew FROM '/home/jason/sqldata/employee.sql';
```

Now, SELECT the data from employeenew and you'll see the following output:

```
 employeeid | employeecode |     name             |       email
------------+--------------+----------------------+---------------
          1 | JG100011     | Jason Gilmore        | jason@example.com
          2 | RT435234     | Robert Treat         | rob@example.com
          3 | GS998909     | Greg Sabino Mullane  | greg@example.com
          4 | MW777983     | Matt Wade            | matt@example.com
(4 rows)
```

Note that an absolute path must be used to refer to the file. Additionally, the PostgreSQL daemon user must be capable of reading the target file. Finally, keep in mind that COPY will not attempt to perform any processing on the file to determine whether the data in each field can legally be placed in a particular table column; rather, it will simply incrementally match each field in the text file to the table column of the same offset.

COPY FROM presumes each field is delimited by a predefined character string, which is by default a tab (\t) for text files, and a comma for CSV files (see the later section "Working with CSV Files"). Furthermore, each row is presumed to be delimited by a similar predefined string, which is by default newline (\n). These predefined characters can be changed if necessary; see the later section "Changing the Default Delimiter" for more information.

Binary

This clause tells PostgreSQL to copy data using a custom format, resulting in a slight increase in performance. However, executing COPY FROM ... BINARY can only be used when the data was previously written using COPY TO ... BINARY. Furthermore, this has nothing to do with storing data such as Word documents or images. It's merely a slightly more efficient way to copy large files.

Exporting the Table OIDs

If the target table was created with OIDs (object identifiers), you can specify that they are output along with the rest of the table data by using the `OIDS` clause. For example:

```
psql>COPY employee TO STDOUT OIDS;
```

This produces:

```
24627  1       GM100011       Jason Gilmore        jason@example.com
24628  2       RT435234       Robert Treat         rob@example.com
24629  3       GS998909       Greg Sabino Mullane  greg@example.com
24630  4       MW777983       Matt Wade            matt@example.com
```

Changing the Default Delimiter

Note the apparent white space found in between each column in the previous example's output. This is actually a tab (`\t`), which is the default delimiter. By using the `DELIMITER` clause, you can change the default to, for instance, a vertical pipe (`|`):

```
psql>COPY employee TO STDOUT DELIMITER '|';
```

This returns the preceding output (minus the OIDs) in the following format:

```
1|GM100011|Jason Gilmore|jason@example.com
2|RT435234|Robert Treat|rob@example.com
3|GS998909|Greg Sabino Mullane|greg@example.com
4|MW777983|Matt Wade|matt@example.com
```

Likewise, if a text file you'd like to import does not use tab characters to delimit fields, specify it similarly to the previous command:

```
psql>COPY employeenew FROM '/home/jason/sqldata/employee.sql' DELIMITER |;
```

Copying Only Specific Columns

If you wanted to copy just the employees' names and e-mail addresses to standard output, specify the column names like so:

```
psql>COPY employee (name,email) TO STDOUT;
```

This produces the following:

```
Jason Gilmore        jason@example.com
Robert Treat         rob@example.com
Greg Sabino Mullane  greg@example.com
Matt Wade            matt@example.com
```

Likewise, if a text file contains only some of the fields to be inserted in a table, you can specify them as was done previously. Keep in mind, however, that the other table columns need to either be nullable or possess default values.

Dealing with Null Values

While e-mail is a crucial communications tool for individuals working in an office environment, suppose some of the employees work solely in the warehouse, negating the need for e-mail. Therefore, some of the e-mail values in the `employee` table might be null. When exporting data using `COPY`, the default for null values is `\N`, and when using CSV mode (discussed later in this section), it's an empty string. However, what if you want to declare a custom string for such instances, `no email` for example? You should use the `NULL` clause, like so:

```
psql>COPY employee TO STDOUT NULL 'no email';
```

This produces output similar to this (presuming some of the employee e-mail addresses have been set to null):

```
Jason Gilmore         no email
Robert Treat          rob@example.com
Greg Sabino Mullane   greg@example.com
Matt Wade             no email
```

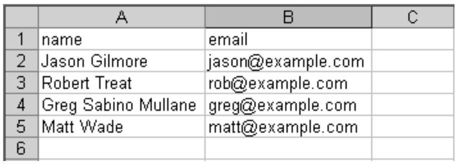
Similarly, if you are importing data from a text file and a `NULL` value is specified, anytime that value is located, the corresponding column will be nulled.

Working with CSV Files

A comma-separated value (CSV) file is a format accepted by possibly every mainstream relational database in existence today, not to mention a wide variety of products such as Microsoft Excel. You can easily create a CSV file from a PostgreSQL table by using `COPY` accompanied by the `CSV` clause. For instance, to create a file capable of immediately being viewed in Microsoft Excel or OpenOffice.org Calc, execute the following command:

```
psql>COPY employee (name, email) TO '/home/jason/sqldata/employee.csv' CSV HEADER;
```

Specifying the `HEADER` clause as indicated above causes the names of the retrieved columns to be listed in the first row as column headers. For example, executing this command and opening the `employee.csv` file in Microsoft Excel produces output similar to that shown in Figure 37-1.



	A	B	C
1	name	email	
2	Jason Gilmore	jason@example.com	
3	Robert Treat	rob@example.com	
4	Greg Sabino Mullane	greg@example.com	
5	Matt Wade	matt@example.com	
6			

Figure 37-1. *Viewing the employee.csv file in Microsoft Excel*

If you are reading a CSV file into a table using `COPY FROM` and the `HEADER` clause is declared, the first line will be ignored.

Some data may be delimited by single or double quotes, which have special significance within PostgreSQL, so you need to be aware of them to make sure they are properly accounted for. You can use the `QUOTE` clause to specify this character, which by default is set to double quotes. The specified quotation character can then be escaped using the character identified by the `ESCAPE` clause, which also defaults to double quotes.

If you're exporting data from a table and use `FORCE NOT NULL`, it is presumed that no value is null; if any null value is encountered, it will be inserted as an empty string.

If you're importing data into a table and use `FORCE QUOTE`, then all non-null values will be quoted, either using the default double quotes or whatever value is specified if the `QUOTE` clause is declared.

Calling COPY from a PHP Script

While the `COPY` command as described previously is useful for developers and database administrators, certainly a more intuitive solution is required for end users. To satisfy this need, the `pg_copy_from()` and `pg_copy_to()` functions (introduced in Chapter 30) are made available via PHP's PostgreSQL extension. Both functions operate identically to the previously introduced `COPY FROM` and `COPY TO` commands, respectively, except that they're also easily executable from within your Web application.

In this section, we'll consider a real-world example in which `pg_copy_to()` is used to copy data from a PostgreSQL table to a text file.

Copying Data from a Table to a Text File

Suppose you want to create an interface that allows managers to create CSV files consisting of employee contact information. These files are saved by date to a folder made available to a directory placed on a shared drive. The code for doing so is found in Listing 37-1.

Listing 37-1. *Saving Employee Data to a CSV File (saveemployeedata.php)*

```
<?php

   $pg = pg_connect("host=localhost user=jason password=secret dbname=corporate")
         or die("Could not connect to db server.");

   // Copy the employee table data to an array
   $array = pg_copy_to($pg, "employee", ",");

   // Retrieve current date for file-naming purposes
   $date = date("Ymd");

   // Open the file
   $fh = fopen("/home/reports/share/employees-$date.csv", "w");
```

```
    // Collapse the array to a newline-delimited set of rows
    $contents = implode("\n", $array);

    // Write $contents to the file
    fwrite($fh, $contents);

    // Close the file
    fclose($fh);

?>
```

Once the script has executed, open the newly created file and you'll see output similar to this:

```
1,JG1000011,Jason Gilmore,jason@example.com
2,RT435234,Robert Treat,rob@example.com
3,GS998909,Greg Sabino Mullane,greg@example.com
4,MW777983,Matt Wade,matt@example.com
```

Now try opening this in a spreadsheet program such as Microsoft Excel or OpenOffice.org Calc!

You can also easily add a header to the CSV file by writing a line to it before outputting the array contents, like so:

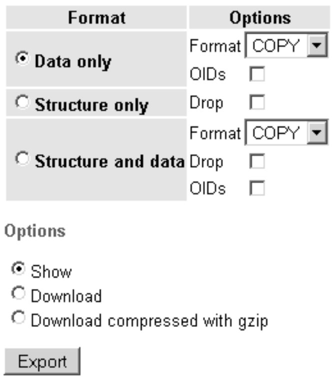
```
fwrite($fh, "Employee ID,Name,Email\n");
```

Importing and Exporting Data with phpPgAdmin

If you're looking for a convenient and powerful administration utility that is capable of being accessed from anywhere you have a Web browser, phpPgAdmin (`http://www.phppgadmin.net/`) is the most capable solution around. First introduced in Chapter 27, phpPgAdmin is capable of managing your database with ease, in addition to both importing and exporting data in a variety of formats.

■ **Note** At the time of writing, using this phpPgAdmin feature with Windows is not supported.

To export data, navigate to the target table and click the Export link located in the top-right corner of the page. Doing so produces the interface found in Figure 37-2.



Format	Options
[✓ Data only]	Format: COPY; OIDs ☐
Structure only	Drop ☐
Structure and data	Format: COPY; Drop ☐; OIDs ☐

Options

[✓ Show]
Download
Download compressed with gzip

Figure 37-2. *phpPgAdmin's export interface*

As you can see, you can export data in three ways:

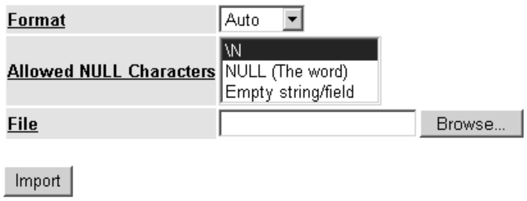
- **Only the table data**: If you want to export only the data, you can do so in six different formats, including COPY, CSV, SQL, Tabbed, XHTML, and XML.

- **Only the table structure**: If you want to export just the table structure, then the table creation SQL syntax is exported. Checking the corresponding Drop checkbox causes table DROP commands to be inserted at the top of the output file so that any preexisting tables of the same name are dropped before being re-created.

- **Both the data and structure**: If you want to export both the table structure and the table data, you can choose to export it in both COPY and SQL formats. Checking the corresponding Drop checkbox causes table DROP commands to be inserted at the top of the output file so that any preexisting tables of the same name are dropped before being re-created.

Note that in all cases, you can either output the information to the browser or download it. Choosing to download it saves the information in a file with the extension `.sql` before prompting you to download it to your local computer.

Importing data is accomplished by navigating to the target table and clicking the Import menu item. Doing so produces the interface found in Figure 37-3.



Format	Auto
Allowed NULL Characters	[✓ \N] NULL (The word) Empty string/field
File	Browse...

Figure 37-3. *phpPgAdmin's import interface*

Imported files are accepted in four formats: Auto, CSV, Tabbed, and XML. The purpose of each format should be obvious, except for perhaps Auto. Selecting Auto causes phpPgAdmin to choose one of the other three formats by examining the file extension (valid extensions are `.csv`, `.tab`, and `.xml`). Also, if any null characters are found in the file, you can specify whether they appear using the sequence `\N`, the word `NULL`, or as an empty string.

Summary

As you learned in this chapter, you have several options at your disposal for importing data into and exporting data from your PostgreSQL database. You can do it manually via the command line or through scripting by using the `COPY` command. Or, you can incorporate these features into a Web application by using PHP's `pg_copy_to()` and `pg_copy_from()` functions. Alternatively, you can rely on applications such as phpPgAdmin to facilitate the process.

This concludes the book. We hope you enjoyed reading it as much as we enjoyed the process of writing it. Good luck!