Beginning PHP and PostgreSQL 8

From Novice to Professional

W. Jason Gilmore and Robert H. Treat

Apress®

Beginning PHP and PostgreSQL 8: From Novice to Professional

Copyright © 2006 by W. Jason Gilmore

ISBN-13 (pbk): 978-1-59059-547-3

ISBN-10 (pbk): 1-59059-547-5

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Matthew Moodie
Technical Reviewers: Greg Sabino Mullane, Matt Wade
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Matt Wade
Project Manager: Beth Christmas
Copy Edit Manager: Nicole LeClerc
Copy Editor: Bill McManus
Assistant Production Director: Kari Brooks-Copony
Production Editor: Laura Cheu
Compositor: Susan Glinert Stevens
Proofreader: Nancy Sixsmith
Indexer: John Collin
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Source Code section.

CHAPTER 27

The Many PostgreSQL Clients

PostgreSQL is bundled with quite a few utilities, or *clients*, each of which provides interfaces for carrying out various tasks pertinent to server administration. This chapter offers an in-depth introduction to the most prominent of the bunch, namely psql. Because the psql manual already does a great job at providing a general overview of each client, we'll instead focus on those features that you're most likely to use regularly in your daily administration activities. We'll show you how to log on and off a PostgreSQL server, explain how to set key environment variables both manually and through configuration files, and offer general tips intended to help you maximize your interaction with psql. Also, because many readers prefer to use a graphical user interface (GUI) to manage PostgreSQL, the chapter concludes with a brief survey of three GUI-based administration applications.

As is the goal with all chapters in this book, the following topics are presented in an order and format that are conducive to helping a novice learn about psql's key features while simultaneously acting as an efficient reference guide for all readers. Therefore, if you're new to psql, begin with the first section and work through the material and examples. If you're a returning reader, feel free to jump around as you see fit. Specifically, the following topics are presented in this chapter:

- **An introduction to psql**: This chapter introduces the psql client along with many of the options that you'll want to keep in mind to maximize its usage.
- **Commonplace psql tasks**: You'll see how to execute many of psql's commonplace commands, including how to log on and off a PostgreSQL server, use configuration files to set environment variables and tweak psql's behavior, read in and edit commands found within external files, and more.
- **GUI-based clients**: Because not all users prefer or even have access to the command line, considerable effort has been put into commercial- and community-driven GUI-based PostgreSQL administration solutions, several of the more popular of which are introduced in this chapter.

What Is psql?

For those of you who prefer the command-line interface over GUI-based alternatives, psql offers a powerful means for managing every aspect of the PostgreSQL server. Bundled with the PostgreSQL distribution, psql is akin to MySQL's mysql client and Oracle's SQL*Plus tool. With it, you can create and delete databases, tablespaces, and tables, execute transactions, execute general queries such as table selections and insertions, and do much more. In this section, you'll learn about the many features at your disposal when using this terse yet powerful client.

psql Options

The psql utility is executed from the command line by executing the `psql` command generally alongside one or several options. Its prototype looks like this:

```
psql [option...][dbname [username]]
```

At a minimum, you need to pass along the `dbname` and `username` parameters if these values aren't stored within the `.psqlrc` configuration file or specified within certain global variables (see the later section "Storing psql Variables and Options"). Therefore, to connect a user `website` to the database `corporate` found on the PostgreSQL server located on IP address 192.168.3.45, you'd execute the following command:

```
%>psql -h 192.168.3.45 corporate website
```

To see the other syntax variations for this task, see the section "Logging Onto and Off the Server," later in this chapter.

In most cases, these three parameters are all that you will require for typical operations (unless you're connecting locally, meaning the host address won't be required), but you may occasionally wish to pass along various options that will affect psql's behavior. The most commonly used options are presented in Table 27-1.

Table 27-1. *Common psql Client Options*

Option	Description
`-c COMMAND`	Executes a single command and then exits.
`-d NAME`	Declares the destination database. The default is `your current username`.
`-f FILENAME`	Executes commands located within the file specified by `FILENAME`, and then exits.
`-h HOSTNAME`	Declares the destination host.
`--help`	Shows the help menu and then exits.
`-l`	Lists the available databases and then exits.
`-L FILENAME`	Sends a session log to the file specified by `FILENAME`.
`-p PORT`	Declares the database port used for the connection. The default is `5432`.
`-U NAME`	Declares the connecting database username. The default is `the current username`.
`-X`	Does not read the system-wide or user-specific startup file (`psqlrc` or `~/.psqlrc`, respectively).

Although manually passing these options along is fine if you need to do so only once or a few times, it can quickly become tedious and error-prone if you have to do so repeatedly. To eliminate these issues, consider storing this information in a configuration file, as discussed in the later section "Storing psql Variables and Options."

Commonplace psql Tasks

psql offers administrators, particularly those who prefer or are particularly adept at working with the command line, a particularly efficient means for interacting with all aspects of a PostgreSQL server. Of course, unlike the point-and-click administration solutions introduced later in this chapter, you need to know the command syntax to make the most of psql. This section shows you how to execute the most commonplace tasks using this powerful utility.

Logging Onto and Off the Server

Before you can do anything with psql, you need to pass along the appropriate credentials. The most explicit means for passing these credentials is to preface each parameter with the appropriate option flag, like so:

```
%>psql -h 192.168.3.45 -d corporate -U websiteuser
```

Upon execution, you are prompted for user `websiteuser`'s password. If the username and corresponding password are validated, you are granted access to the server.

If the database happens to reside locally, you can forego specifying the hostname, like so:

```
%>psql corporate websiteuser
```

In either case, once you've successfully logged in, you see output similar to the following:

```
Welcome to psql 8.1.2, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

corporate=>
```

Note that the prompt specifies the name of the chosen database, which can be useful particularly if you're simultaneously logged in to numerous servers. If you're logged in as a superuser, the prompt will appear a bit differently, like so:

```
corporate=#
```

Once you've completed interacting with the PostgreSQL server, you can exit the connection using `\q`, like so:

```
corporate=> \q
```

Doing so returns you to the operating system's command prompt.

psql Commands

Once you've entered the psql utility, execute `\?` to review a list of psql-specific commands. This produces a list of more than 50 commands divided into six categories. Because this summary does a great job of succinctly defining each command, this section highlights just a few of the commands that you might find particularly useful. Further, many of the commands pertinent to the review of existing databases, schemas, tables, and users are introduced in the coming chapters.

■ **Note** psql's tab-completion feature can save you a great deal of typing when executing commands. As you work through the following examples, tap the Tab key on occasion to review its behavior.

Connecting to a New Database

Over the course of a given session, you'll often need to work with more than one database. To change to a database named `vendor`, execute the following command:

```
corporate=> \connect vendor
```

You can save a few keystrokes by using the abbreviated version of this command, `\c`.

Executing Commands Located Within a Specific File

Repeatedly entering a predetermined set of commands can quickly become tedious, not to mention error-prone. Save yourself from such repetition by storing the commands within a separate file and then executing those commands by invoking the `\i` command and passing along the name of the file, like so:

```
corporate=> \i audit.sql
```

Editing a File Without Leaving psql

If you are relying on commands found in a separate file, the task of repeatedly executing the command and then exiting psql to make adjustments to those commands from within an editor can become quite tedious. To save yourself from the tedium, you can edit these files without ever leaving psql by executing `\e`. For example, to edit the `audit.sql` file used in the previous example, execute the following command:

```
corporate=> \e audit.sql
```

This will open the file within whatever editor has been assigned via the `PSQL_EDITOR` variable (see Table 27-2 for more information about this variable). Once you've completed editing the file, save the file using the editor's specific save command and exit the editor (`:wq` in vim, for instance). You will be returned directly back to the psql interface, and can again execute the file using `\i` if you wish.

Sending Query Output to an External File

Sometimes you may wish to redirect query output to an external file for later examination or additional processing. To do so, execute the `\o` command, passing it the name of the desired output file. For instance, to redirect all output to a file named `output.sql`, execute the `\o` command, like so:

```
corporate=> \o output.sql
```

Storing psql Variables and Options

Of course, heavy-duty command-line users know that repeatedly entering commonly used commands can quickly become tedious. To eliminate such repetition, you should take advantage of aliases, configuration files, and environment variables at every possibility.

To set an environment variable from within psql, just execute the `\set` command followed by the variable name and a corresponding value. For example, suppose your database consists of a table named `apressproduct`. You're constantly working with this table and, accordingly, are growing sick of typing in its name. You can forego the additional typing by assigning an environment variable, like so:

```
corporate=> \set ap 'apressproduct'
```

Now it's possible to execute queries using the abbreviated name:

```
corporate=> SELECT name, price FROM :ap;
```

Note that a colon must prefix the variable name in order for it to be interpolated. psql also supports a number of predefined variables. A list of the most commonly used psql variables are presented in Table 27-2.

Table 27-2. *Commonly Used psql Variables*

Variable	Description
`PAGER`	Determines which paging utility is used to page output that requires more space than a single screen.
`PGDATABASE`	The presently selected database.
`PGHOST`	The name of the server hosting the PostgreSQL database.
`PGHOSTADDR`	The IP address of the server hosting the PostgreSQL database.
`PGPORT`	The post on which the PostgreSQL server is listening for connections.
`PGPASSWORD`	Can be used to store a connecting password. However, this variable is deprecated, so you should use the `.pgpass` file instead for password storage.
`PGUSER`	The name of the connected user.
`PSQL_EDITOR`	The editor used for editing a command prior to execution. This feature is particularly useful for editing and executing long commands that you may wish to store in a separate file. After looking to `PSQL_EDITOR`, psql will then examine the contents of the `EDITOR` and `VISUAL` variables, if they exist. If examination of all three variables proves inconclusive, `notepad.exe` is executed on Windows, and `vi` on all other operating systems.

To view a list of all presently set variables, execute `\set` without passing it any parameters, like so:

```
corporate=> \set
```

For instance, executing this command on our Ubuntu server produces:

```
VERSION = 'PostgreSQL 8.1.2 on i686-pc-linux-gnu, compiled by GCC gcc(GCC) 3.3.5
(Debian 1:3.3.5-8ubuntu2)'
AUTOCOMMIT = 'on'
VERBOSITY = 'default'
PROMPT1 = '%/%R%# '
PROMPT2 = '%/%R%# '
PROMPT3 = '>> '
DBNAME = 'corporate'
USER = 'websiteuser'
PORT = '5432'
ENCODING = 'SQL_ASCII'
HISTFILE = '~/.psql_history'
HISTSIZE = '500'
```

Storing Configuration Information in a Startup File

PostgreSQL users have two startup files at their disposal, both of which can be used to affect psql's behavior on the system-wide and user-specific levels, respectively. The system-wide `psqlrc` file is located within PostgreSQL's `etc/` directory on Linux and within `%APPDATA\postgresql\` on Windows, whereas the user-specific file is stored within the user's home directory and prefixed with a period (`.`), as is standard for configuration files of this sort.

■ **Note** On Windows, the system-wide `psqlrc` file should use `.conf` as the extension. Also, to determine the location of `%APPDATA%`, open a command prompt and execute `echo %APPDATA%`. Further, on both Linux and Windows, you can create version-specific startup files by appending a dash and specific version number to `psqlrc`. For example, a system-wide startup file named `psqlrc-8.1.0` will be read only when connecting to a PostgreSQL server running version 8.1.0.

Both files support the same syntax, and anything stored in the system-wide file can also be stored in the user-specific version. However, keep in mind that if both files contain the same setting, anything found in the user-specific version will override the value declared in the system-wide version, because the user-specific version is read last. So what might one of these files look like? The following presents an example of what you might expect to find within a user's `.psqlrc` file:

```
# Set the prompt
\set PROMPT1 '%n@%m::%`date +%H:%M:%S`> '

# Set the location of the history file
\set HISTFILE ~/pgsql/.psql_history
```

Learning More About Supported SQL Commands

Once you're logged into the server, execute `\h` to view all available commands. At the time of this writing, there were 109 commands. To view all of them, execute the following:

```
corporate=> \h
```

This produces the following output:

```
Available help:
  ABORT                     CREATE LANGUAGE          DROP VIEW
  ALTER AGGREGATE           CREATE OPERATOR CLASS    END
  ALTER CONVERSION          CREATE OPERATOR          EXECUTE
  ALTER DATABASE            CREATE ROLE              EXPLAIN
  ALTER DOMAIN              CREATE RULE              FETCH
  ALTER FUNCTION            CREATE SCHEMA            GRANT
  ALTER GROUP               CREATE SEQUENCE          INSERT
  ALTER INDEX               CREATE TABLE             LISTEN
  ALTER LANGUAGE            CREATE TABLE AS          LOAD
  ALTER OPERATOR CLASS      CREATE TABLESPACE        LOCK
  ALTER OPERATOR            CREATE TRIGGER           MOVE
  ALTER ROLE                CREATE TYPE              NOTIFY
  ALTER SCHEMA              CREATE USER              PREPARE
  ALTER SEQUENCE            CREATE VIEW              PREPARE TRANSACTION
  ALTER TABLE               DEALLOCATE               REINDEX
  ALTER TABLESPACE          DECLARE                  RELEASE SAVEPOINT
  ALTER TRIGGER             DELETE                   RESET
  ALTER TYPE                DROP AGGREGATE           REVOKE
  ALTER USER                DROP CAST                ROLLBACK
  ANALYZE                   DROP CONVERSION          ROLLBACK PREPARED
  BEGIN                     DROP DATABASE            ROLLBACK TO SAVEPOINT
  CHECKPOINT                DROP DOMAIN              SAVEPOINT
  CLOSE                     DROP FUNCTION            SELECT
  CLUSTER                   DROP GROUP               SELECT INTO
  COMMENT                   DROP INDEX               SET
  COMMIT                    DROP LANGUAGE            SET CONSTRAINTS
  COMMIT PREPARED           DROP OPERATOR CLASS      SET ROLE
  COPY                      DROP OPERATOR            SET SESSION AUTHORIZATION
  CREATE AGGREGATE          DROP ROLE                SET TRANSACTION
  CREATE CAST               DROP RULE                SHOW
  CREATE CONSTRAINT TRIGGER DROP SCHEMA              START TRANSACTION
  CREATE CONVERSION         DROP SEQUENCE            TRUNCATE
  CREATE DATABASE           DROP TABLE               UNLISTEN
  CREATE DOMAIN             DROP TABLESPACE          UPDATE
  CREATE FUNCTION           DROP TRIGGER             VACUUM
  CREATE GROUP              DROP TYPE
  CREATE INDEX              DROP USER
```

To learn more about a particular command, execute `\h` again, but this time pass the command as a parameter. For example, to learn more about the `INSERT` command, execute the following:

```
corporate=> \h INSERT
```

This produces the following output:

```
Command:     INSERT
Description: create new rows in a table
Syntax:
INSERT INTO table [ ( column [, ...] ) ]
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) | query }
```

Therefore, `\h` is useful not only for determining what psql commands are at your disposal, but also for recalling what syntax is required for a particular command.

Executing a Query

Once connected to a PostgreSQL server, you're free to execute any supported query. For example, to retrieve a list of all company employees, execute a `SELECT` query, like so:

```
corporate=>SELECT lastname, email, telephone FROM employee ORDER by lastname;
```

Executing a `DELETE` query works just the same:

```
corporate=> DELETE FROM hr.employee WHERE lastname='Gilmore';
```

If you're interested in executing a single query, you can do so when invoking psql, like so:

```
%>psql -d corporate -U hrstaff
-c "SELECT lastname, email, telephone FROM employee ORDER by lastname"
```

Once the appropriate query result has been displayed, psql exits and returns to the command line.

For automation purposes, you can dump query output to a file with the `-o` option:

```
%>psql -d corporate -U hrstaff
-c "SELECT lastname, email, telephone FROM employee ORDER by lastname"
-o "/dataimport/employeeinfo.txt"
```

■**Note** In the next chapter, you'll learn how to execute commonplace administration tasks such as managing users and creating and destroying databases and schemas.

Modifying the psql Prompt

Because of the lack of visual cues when using the command line, it's easy to forget which database you're presently using, or even which server you're logged into if you're working on multiple database servers simultaneously. However, you can avoid any such confusion by modifying the psql prompt to automatically display various items of information. For example, if you'd like your prompt to include the name of the server host, the username you're logged in as, and the name of the current database, set the PROMPT1 variable, like so:

```
corporate=> \set PROMPT1 '%n@%m::%/> '
```

Once set, the prompt contains the username, server hostname, and presently selected database, like this example:

```
corporate@apress::test>
```

Two other prompt variables exist, namely PROMPT2 and PROMPT3. PROMPT2 stores the prompt for subsequent lines of a multiline statement. PROMPT3 represents the prompt used while entering data passed to the COPY command. All three variables use the same substitution sequences to determine what the rendered prompt will look like. Many of the most common sequences are presented in Table 27-3.

Table 27-3. *Common Prompt Substitution Sequences*

Sequence	Description
%~	The name of the presently selected database. Alternatively, the %/ sequence can be used.
%#	The hash mark if the present user is a superuser. Alternatively, the greater-than sign (>) is used.
%>	The server port number.
%`*command*`	Output of the command represented by command. For instance, you might set this (on a Unix system) to %`date +%H:%M:%S` to include the present time on each prompt.
%m	The server hostname.
%n	The presently connected user's username.

Controlling the Command History

Three variables control psql's command history capabilities:

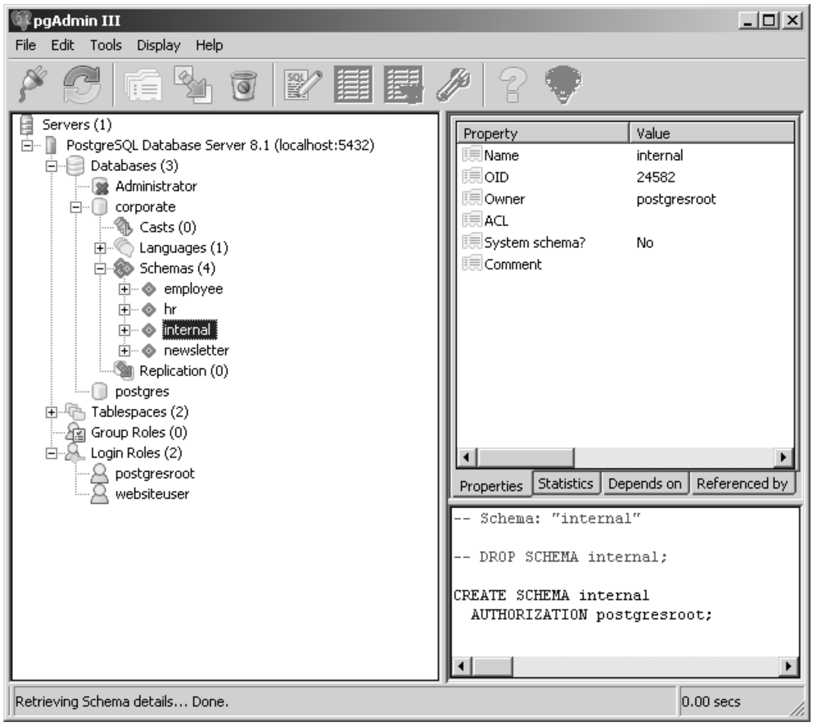
- HISTCONTROL: This variable determines whether certain lines will be ignored. If set to ignoredups, any repeatedly entered lines occurring directly following the first line will not be logged. If set to ignorespace, any lines beginning with a space are ignored. If set to ignoreboth, both ignoredups and ignorespace are enforced.
- HISTFILE: By default, a user's history information is stored within ~/.psql_history. However, you're free to change this to any location you please, ~/pgsql/.psql_history for instance. On Windows, the preceding period is omitted (psql_history).
- HISTSIZE: By default, 500 of the most recent lines are stored within the history file. Using HISTSIZE, you can change this to any size you please.

GUI-based Clients

Although a command-line-based client such as psql offers an amazing degree of efficiency, its practical use comes at the cost of having to memorize a great number of often-complex commands. The memorization process not only is tedious, but can also require a great deal of typing (although using the tab-completion feature can greatly reduce that). To make commonplace database administration tasks more tolerable, both the PostgreSQL developers and third-party vendors have long offered GUI-based solutions. This section introduces several of the most popular products.

pgAdmin III

pgAdmin III is a powerful, client-based administration utility that is capable of managing nearly every aspect of a PostgreSQL server, including the various PostgreSQL configuration files, data and data structures, users, and groups. Figure 27-1 shows the interface you might encounter when reviewing the `corporate` database's schemas.



Property	Value
Name	internal
OID	24582
Owner	postgresroot
ACL	
System schema?	No
Comment	

```
-- Schema: "internal"

-- DROP SCHEMA internal;

CREATE SCHEMA internal
  AUTHORIZATION postgresroot;
```

Figure 27-1. *Viewing the corporate database's internal table schema*

Availability

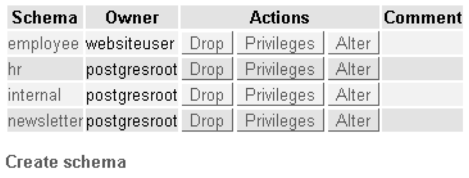
Licensed under the open source Artistic license, pgAdmin III is freely available for download, use, distribution, and modification in accordance with the Artistic license's terms. For most users, their concern applies solely to usage; in this case you're free to use pgAdmin III for both personal and commercial uses free of charge.

If you'd like to use pgAdmin III on a Unix-based platform, you first need to download it from the pgAdmin Web site (`http://www.pgadmin.org/`) or from the appropriate directory within the PostgreSQL FTP server (`http://www.postgresql.org/ftp/`). Offering binaries for Fedora Core 4, FreeBSD, Mandriva Linux, OS X, and Slackware, in addition to the source code, you're guaranteed to be able to use pgAdmin III regardless of platform. If you're using Windows, pgAdmin III is bundled and installed along with the PostgreSQL server download; therefore, no special installation steps are necessary for this platform.

phpPgAdmin

Managing your database using a Web-based administration interface can be very useful because it not only enables you to log in from any computer connected to the Internet, but also enables you to easily secure the connection using SSL. Additionally, not all hosting providers allow users to log in to a command-line interface, nor connect remotely through any but a select few, well-defined ports, negating the possibility that a client-side application could be easily used. For all of these reasons and more, you might consider installing a Web-based PostgreSQL manager. While there are several such products, the most prominent is phpPgAdmin, an open source, Web-based PostgreSQL administration application written completely in PHP.

Modeled after the extremely popular phpMyAdmin (`http://www.phpmyadmin.net/`) application (used to manage the MySQL database), phpPgAdmin has been in active development since 2002, and is presently collaboratively developed by a team of seven. It supports all of the features one would expect of such an application, including the ability to manage users and databases, generate reports and view server statistics, import and export data, and much more. For instance, Figure 27-2 depicts the interface you'll encounter when viewing the schemas found within the example `corporate` database.



Schema	Owner	Actions			Comment
employee	websiteuser	Drop	Privileges	Alter	
hr	postgresroot	Drop	Privileges	Alter	
internal	postgresroot	Drop	Privileges	Alter	
newsletter	postgresroot	Drop	Privileges	Alter	

Create schema

Figure 27-2. *Viewing the corporate database's schemas*

■ **Note** phpPgAdmin requires PHP 4.1 or greater, and supports all versions of PostgreSQL 7.0 and greater.

Availability

phpPgAdmin is freely available for download and use under the GNU GPL license. To install phpPgAdmin, proceed to the phpPgAdmin Web site (`http://phppgadmin.sourceforge.net/`) and download the latest stable version. It is compressed using three different formats, bz2, gz, and zip, so download the version that's most convenient to your platform and uncompress it to an appropriate location within the Web server document root.

Next, open the `conf/config.inc.php-dist` file, located in this newly uncompressed directory (which at the time of writing is titled `phpPgAdmin`), and save it as `config.inc.php` to the same directory. Open a Web browser and proceed to the phpPgAdmin home directory—for example, `http://www.example.com/phpPgAdmin/index.php`. You will be presented with a welcome screen, which prompts for a username, password, choice of language, and a target server (provided more than one was defined within the `config.inc.php` file; open this file for more details).

This interface prompts you for a username and password, referring to one of the accounts created within the PostgreSQL server. For security reasons, you cannot log in without a password, nor with the usernames `administrator`, `pgsql`, `postgresql`, or `root`, as this presumes you're attempting to log in using the superuser account and therefore could be transmitting the password in an unencrypted format. Because the `config.inc.php` file can store information for any number of PostgreSQL servers via the `$conf['servers']` configuration array, you'll be able to choose which server to connect to using the Server drop-down list box. You can also change the interface's language. At the time of writing, phpPgAdmin supports 26 different languages, including English, Spanish, Italian, and Romanian, to name a few.

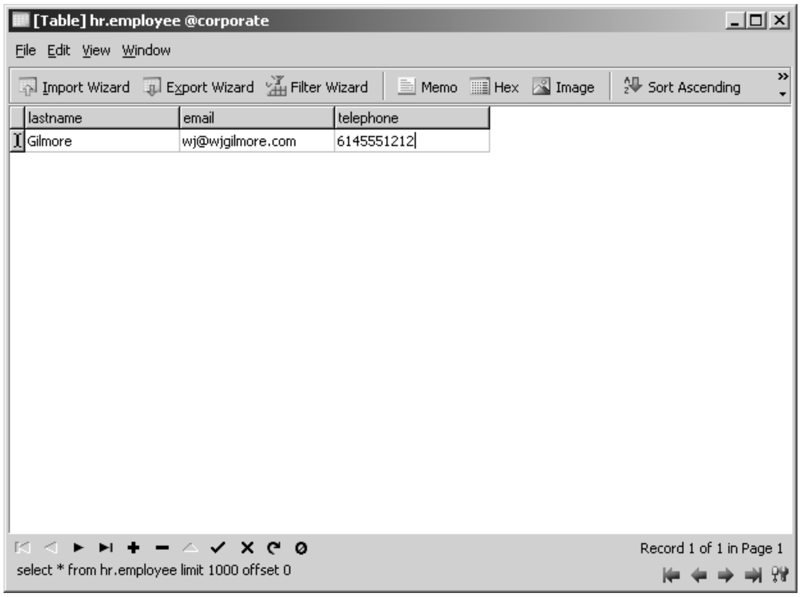
If you've already gone ahead and tried to log in, depending upon how your PostgreSQL installation is configured, you might have been surprised to learn that you are allowed in even if you entered an incorrect or blank password. This is not a flaw in phpPgAdmin, but rather is a byproduct of PostgreSQL's default configuration of using trust-based authentication! See Chapter 29 for more information about how to modify this feature.

Navicat

Navicat is a commercial PostgreSQL database administration client application that presents a host of user-friendly tools through a rather slick interface. Under active development for several years, Navicat offers users a feature-rich and stable solution for managing all aspects of the database server. Navicat offers a number of compelling features:

- An interface that provides easy access to 10 different management features, including backups, connections, data synchronization, reporting, scheduled tasks, stored procedures, structure synchronization, tables, users, and views.
- Comprehensive user management features, including a unique tree-based privilege administration interface that allows you to quickly add and delete database, table, and column rights.
- A mature, full-featured interface for creating and managing views.
- Most tools offer a means for managing the database by manually entering the command, as one might via the psql client, and a wizard for accomplishing the same via a point-and-click interface.

Figure 27-3 depicts Navicat's data-viewing interface.



lastname	email	telephone
Gilmore	wj@wjgilmore.com	6145551212

Record 1 of 1 in Page 1

```
select * from hr.employee limit 1000 offset 0
```

Figure 27-3. *Viewing the contents of corporate.hr.employee*

Availability

Navicat is a product of PremiumSoft CyberTech Ltd. and is available for download at `http://www.navicat.com/`. Unlike the previously discussed solutions, Navicat is not free, and at the time of writing costs $129, $79, and $75 for the enterprise, standard, and educational versions, respectively. You can download a fully functional 30-day evaluation version. Binary packages are available for Microsoft Windows, Mac OS X, and Linux platforms.

Summary

You need to have a capable utility at your disposal to effectively manage your PostgreSQL server. Regardless of whether your particular situation or preference calls for a command-line or graphical interface, this chapter demonstrated that you have a wealth of options at your disposal.

The next chapter discusses how PostgreSQL organizes its data hierarchies, introducing the concepts of clusters, databases, schemas, and tables. You'll also learn about the many datatypes PostgreSQL supports for representing a wide variety of data, how table attributes affect the way tables operate, and how to enforce data integrity.