



Getting Started with EDB Postgres™ Advanced Server on Linux®

EDB Postgres™ Advanced Server 9.5
formerly Postgres Plus Advanced Server 9.5

March 7, 2016

**Getting Started with EDB Postgres Advanced Server on Linux
by EnterpriseDB® Corporation
Copyright © 2016 EnterpriseDB Corporation. All rights reserved.**

EnterpriseDB Corporation, 34 Crosby Drive, Suite 100, Bedford, MA 01730, USA
T +1 781 357 3390 **F** +1 978 589 5701 **E** info@enterprisedb.com **www**.enterprisedb.com

Table of Contents

1	Introduction.....	4
1.1	Typographical Conventions Used in this Guide.....	5
2	Controlling the Advanced Server Service.....	6
2.1	Controlling a Service on CentOS or RHEL 7.x.....	6
2.2	Controlling a Service on CentOS or RHEL 6.x.....	9
3	Getting Started with the PEM Client.....	12
3.1	Opening the PEM Graphical Client.....	12
3.2	The PEM Client User Interface.....	14
3.3	Using the PEM Client.....	18
3.3.1	Creating a Table.....	18
3.3.2	Viewing and Managing Data.....	20
3.3.3	Querying Data.....	22
4	Getting Started with EDB-PSQL.....	24
4.1	Connecting with the EDB-PSQL Client on CentOS or RHEL 7.x.....	24
4.2	Connecting with the EDB-PSQL Client on CentOS or RHEL 6.x.....	26
4.3	PSQL Meta-Commands.....	27
4.4	Using the PSQL Client.....	28

1 Introduction

Notice: The names for EDB's products have changed.

The product formerly referred to as Postgres Plus Advanced Server is now referred to as EDB Postgres Advanced Server (Advanced Server).

The product formerly referred to as Postgres Enterprise Manager (PEM) is now referred to as EDB Postgres Enterprise Manager (EDB Enterprise Manager).

Until a new version of this documentation is published, wherever you see an earlier version of a product name, you may substitute it with the current name. Name changes in software and software outputs will be phased in over time.

EDB Postgres Advanced Server (Advanced Server) provides all of the power and flexibility of open-source PostgreSQL, with additional functionality that provides simplified database administration, enhanced SQL capabilities, extended database and application security, performance monitoring and analysis, and application development utilities.

This EnterpriseDB Tutorial will familiarize you with Advanced Server in a Linux environment. We assume that you have already downloaded and installed Advanced Server on your desktop or laptop computer. For detailed information about installing Advanced Server, refer to the EDB Postgres Advanced Server Installation Guide, available at

<http://www.enterprisedb.com/products-services-training/products/documentation/enterpriseedition>

Getting Started with EDB Postgres Advanced Server on Linux introduces the following basics:

- determining the server status
- starting, stopping and restarting the server
- opening the Postgres Enterprise Manager (PEM) graphical client
- using the PEM client
- opening the EDB-PSQL command line client
- using the EDB-PSQL client

The PEM client is a graphical client interface for Postgres databases that is based on the pgAdmin open-source project. The PEM client provides a point-and-click environment where you can create and manage database objects, and roles and their privileges.

The EDB-PSQL client is a command line client based on PostgreSQL psql; for more information about psql, please see the PostgreSQL core documentation at:

<http://www.postgresql.org/docs/9.5/static/app-psql.html>

Throughout this guide, the term Postgres refers to either a PostgreSQL or EDB Postgres Advanced Server installation, where either is appropriate.

1.1 *Typographical Conventions Used in this Guide*

Certain typographical conventions are used in this manual to clarify the meaning and usage of various commands, statements, programs, examples, etc. This section provides a summary of these conventions.

In the following descriptions a *term* refers to any word or group of words that are language keywords, user-supplied values, literals, etc. A term's exact meaning depends upon the context in which it is used.

- *Italic font* introduces a new term, typically, in the sentence that defines it for the first time.
- Fixed-width (mono-spaced) font is used for terms that must be given literally such as SQL commands, specific table and column names used in the examples, programming language keywords, etc. For example, `SELECT * FROM emp;`
- *Italic fixed-width font* is used for terms for which the user must substitute values in actual usage. For example, `DELETE FROM table_name;`
- A vertical pipe | denotes a choice between the terms on either side of the pipe. A vertical pipe is used to separate two or more alternative terms within square brackets (optional choices) or braces (one mandatory choice).
- Square brackets [] denote that one or none of the enclosed terms may be substituted. For example, [a | b] means choose one of “a” or “b” or neither of the two.
- Braces { } denote that exactly one of the enclosed alternatives must be specified. For example, { a | b } means exactly one of “a” or “b” must be specified.
- Ellipses ... denote that the preceding term may be repeated. For example, [a | b] ... means that you may have the sequence, “b a a b a”.

2 Controlling the Advanced Server Service

In this section, you will learn how to discover or control the state of the Advanced Server service through the Linux Terminal window.

The Advanced Server service is named `ppas-9.5`. By default, the `ppas-9.5` service runs in the background without user notification or interaction. The commands that control the Advanced Server service on a Linux platform are *version* specific.

2.1 Controlling a Service on CentOS or RHEL 7.x

You can access the command line Terminal in CentOS or RHEL 7.x by clicking Applications, pointing through Utilities to click on Terminal (see Figure 2.1).

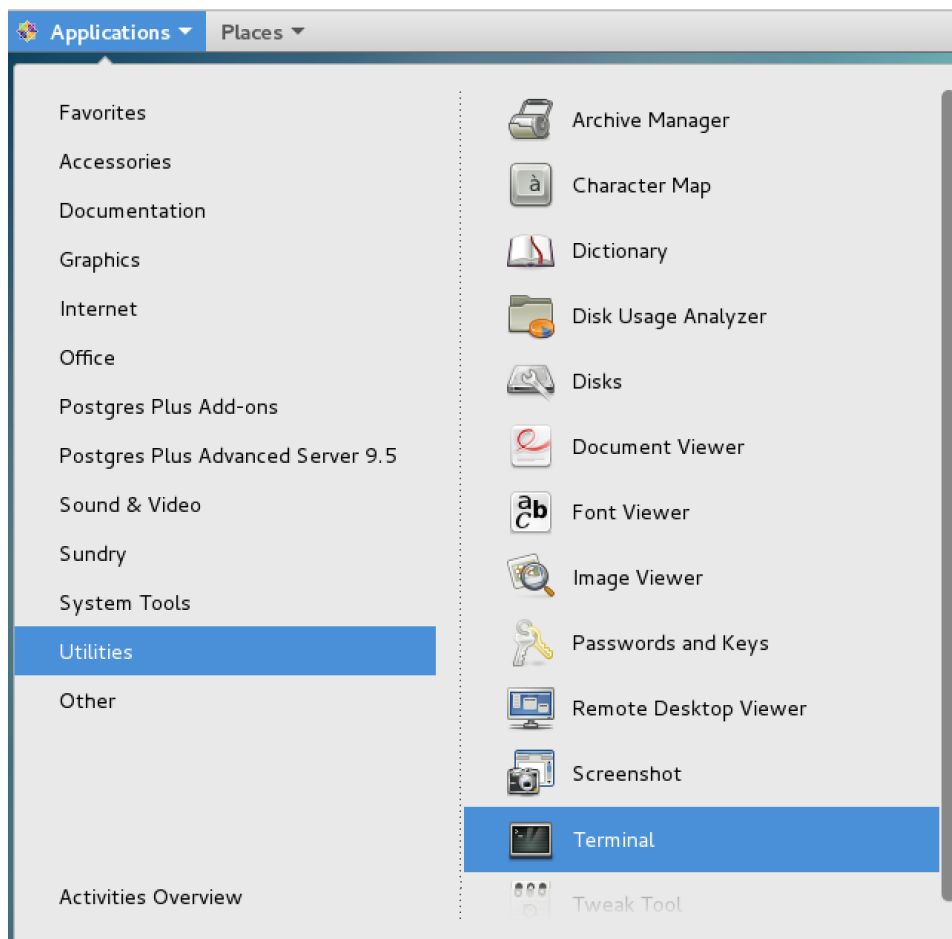


Figure 2.1 – Opening the Terminal window on CentOS or RHEL 7.x.

If your installation of Advanced Server resides on version 7.x of RHEL and CentOS, you must use the `systemctl` command to control the Advanced Server service and supporting components. To see if the service is running, assume superuser privileges with the command:

```
su -
```

If prompted, provide a password (see Figure 2.3).

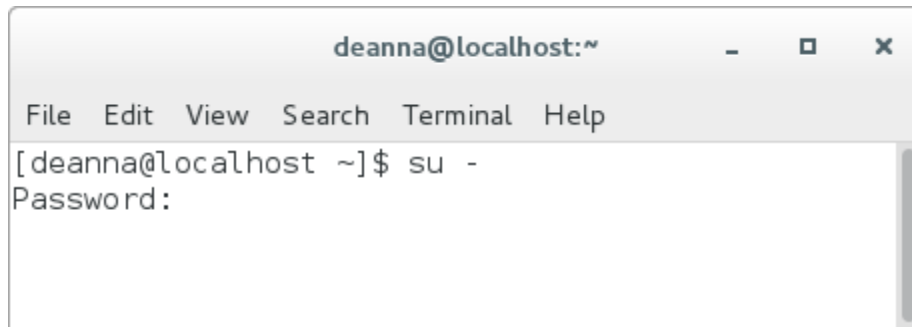


Figure 2.3 – The prompt in the Terminal window.

Note that the last character of the prompt changes to a pound sign, #, when you are an operating system superuser (see Figure 2.4).

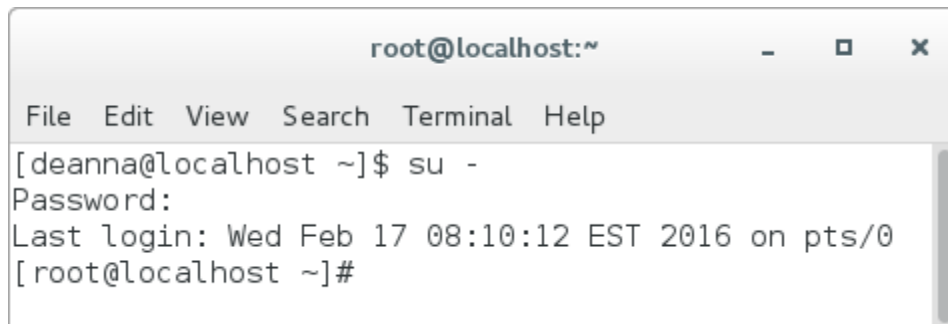


Figure 2.4 – Assuming superuser privileges.

Use the following command to determine the status of `ppas-9.5`:

```
systemctl status ppas-9.5
```

A message displays the current state of the `ppas-9.5` service. It will indicate whether `ppas-9.5` is active (running) or inactive (dead) (see Figure 2.5).

Getting Started with EDB Postgres Advanced Server on Linux

```
root@localhost:~  
File Edit View Search Terminal Help  
[root@localhost ~]# systemctl status ppas-9.5  
● ppas-9.5.service - Postgres Plus Advanced Server 9.5  
   Loaded: loaded (/usr/lib/systemd/system/ppas-9.5.service; enabled; vendor preset: disabled)  
   Active: active (running) since Wed 2016-02-17 08:30:04 EST; 9min ago  
   Main PID: 40936 (edb-postgres)  
   CGroup: /system.slice/ppas-9.5.service  
           └─40936 /opt/PostgresPlus/9.5AS/bin/edb-postgres -D /opt...  
             └─40937 postgres: logger process  
               └─40939 postgres: checkpointer process  
                 └─40940 postgres: writer process  
                   └─40941 postgres: wal writer process  
                     └─40942 postgres: autovacuum launcher process  
                       └─40943 postgres: stats collector process  
                         └─41096 postgres: enterprisedb edb ::1[50325] idle  
  
Feb 17 08:30:04 localhost.localdomain bash[40914]: waiting for serv...  
Feb 17 08:30:04 localhost.localdomain bash[40914]: server started  
Feb 17 08:30:04 localhost.localdomain systemd[1]: Started Postgres ...  
Feb 17 08:30:15 localhost.localdomain systemd[1]: [/usr/lib/systemd...  
Feb 17 08:30:15 localhost.localdomain systemd[1]: [/usr/lib/systemd...  
Feb 17 08:30:37 localhost.localdomain systemd[1]: [/usr/lib/systemd...  
Feb 17 08:30:52 localhost.localdomain systemd[1]: [/usr/lib/systemd...  
Feb 17 08:30:57 localhost.localdomain systemd[1]: [/usr/lib/systemd...  
Feb 17 08:31:07 localhost.localdomain systemd[1]: [/usr/lib/systemd...  
Feb 17 08:31:07 localhost.localdomain systemd[1]: [/usr/lib/systemd...  
Hint: Some lines were ellipsized, use -l to show in full.
```

Figure 2.5 – Determining the Status of ppas-9.5.

To manually stop, start, or restart the service, enter the appropriate command from the following table:

Behavior	Enter command
To check the status	systemctl status ppas-9.5
To start	systemctl start ppas-9.5
To stop*	systemctl stop ppas-9.5
To restart**	systemctl restart ppas-9.5
To stop the service without displaying a notice if the server is already stopped*	service ppas-9.5 condstop
To restart a running service **	service ppas-9.5 condrestart
To restart a running service**	service ppas-9.5 try-restart

*Please note: When you stop the service, any dependent services will also be stopped. When restarting Advanced Server, you can confirm that your supporting services are running by checking their status .

**The restart command is useful when reloading configuration parameters. As with the stop command, any user sessions will be terminated when the service stops. Please check dependent services are running at the ppas-9.5 service restart.

2.2 Controlling a Service on CentOS or RHEL 6.x

The Terminal window is listed with other System Tools in the Applications context menu. Click Applications, point to System Tools, proceed and click the Terminal icon to use the command line (see Figure 2.2).

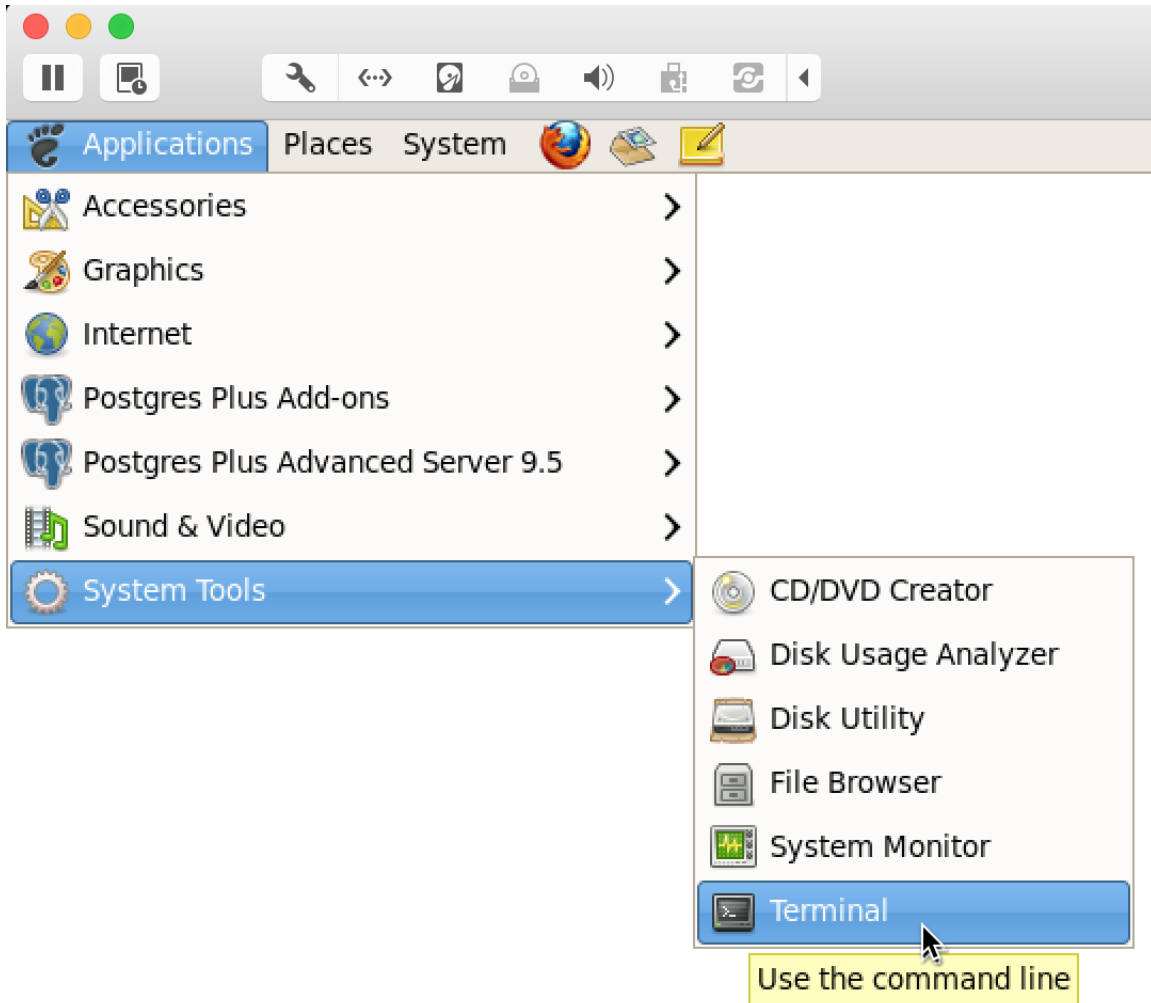


Figure 2.2 – Opening the Terminal window on CentOS or RHEL 6.x.

To find out if the service is running, assume superuser privileges with the following command:

```
su -
```

If prompted, provide a password (see Figure 2.6).

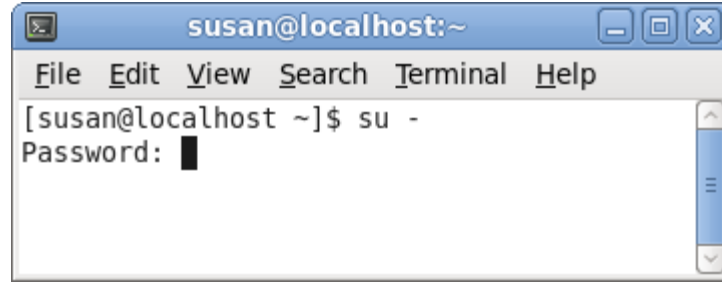


Figure 2.6 – The prompt in the Terminal window.

Note that the last character of the prompt changes to a pound sign, #, when you have administrative privileges (see Figure 2.7).

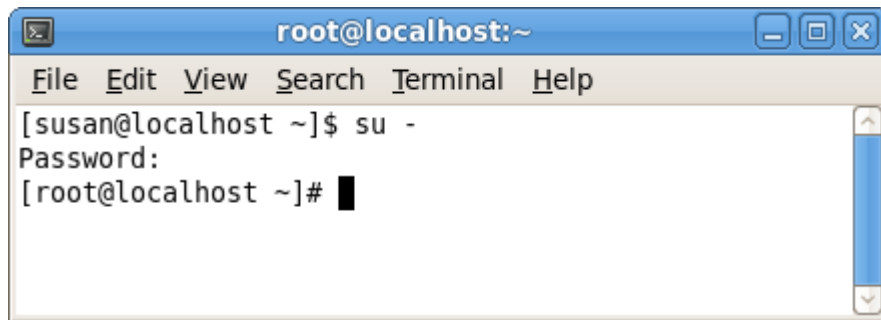


Figure 2.7 – Assuming superuser privileges.

You can determine the status of the service with the command:

```
service ppas-9.5 status
```

A message displays the current state of the ppas-9.5 service (see Figure 2.8).

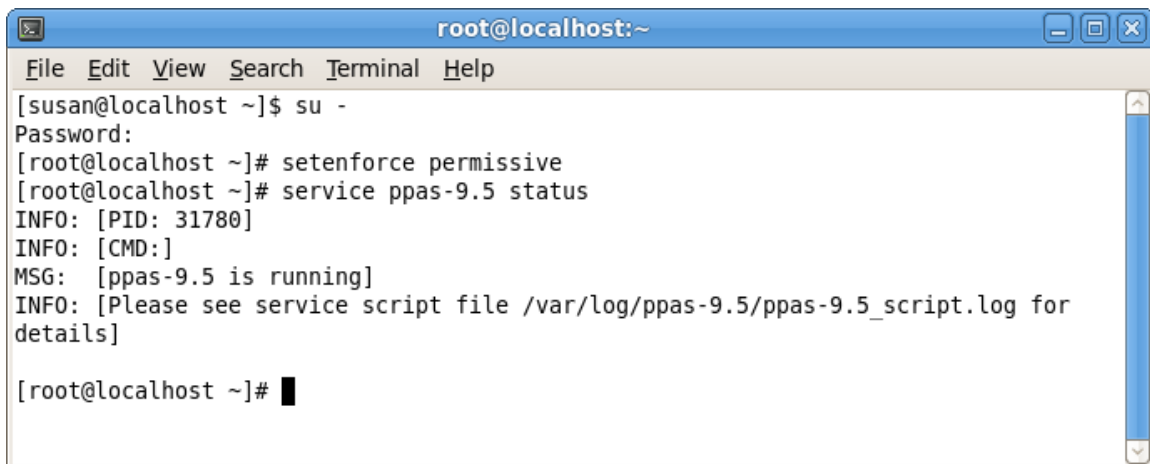


Figure 2.8 – Determining the Status of ppas-9.5.

Getting Started with EDB Postgres Advanced Server on Linux

Should you wish to manually stop, start, or restart the service, enter the appropriate command from the following table after the prompt:

Behavior	Enter command	Confirmation message or MSG:
To check the status	<code>service ppas-9.5 status</code>	[ppas-9.5 is running] or [ppas-9.5 is not running]
To start	<code>service ppas-9.5 start</code>	[ppas-9.5 started]
To stop*	<code>service ppas-9.5 stop</code>	[ppas-9.5 stopped]
To restart**	<code>service ppas-9.5 restart</code>	[ppas-9.5 restarted]
To stop the service without displaying a notice if the server is already stopped*	<code>service ppas-9.5 condstop</code>	[ppas-9.5 stopped]
To restart a running service **	<code>service ppas-9.5 condrestart</code>	[ppas-9.5 restarted]
To restart a running service**	<code>service ppas-9.5 try-restart</code>	[ppas-9.5 restarted]

*Please note: When you stop the service, any dependent services will also be stopped. Upon restarting Advanced Server, you can confirm that your supporting services are running by checking their status .

**The restart command is useful when reloading configuration parameters. As with the stop command, any user sessions will be terminated when the service stops. Please check dependent services are running at the ppas-9.5 service restart.

3 Getting Started with the PEM Client

The Postgres Enterprise Manager (PEM) client allows you to graphically manage multiple Postgres database servers from a single graphical user interface. The PEM client's dialogs allow you to create, query, and manage database objects and roles, and provides simplified configuration of supporting database functionality (such as replication and job scheduling).

3.1 Opening the PEM Graphical Client

The PEM client is distributed with the Advanced Server installer. To start the PEM client, select `Postgres Enterprise Manager v6` from `Advanced Server 9.5` in the `Applications` menu (see Figure 3.1).

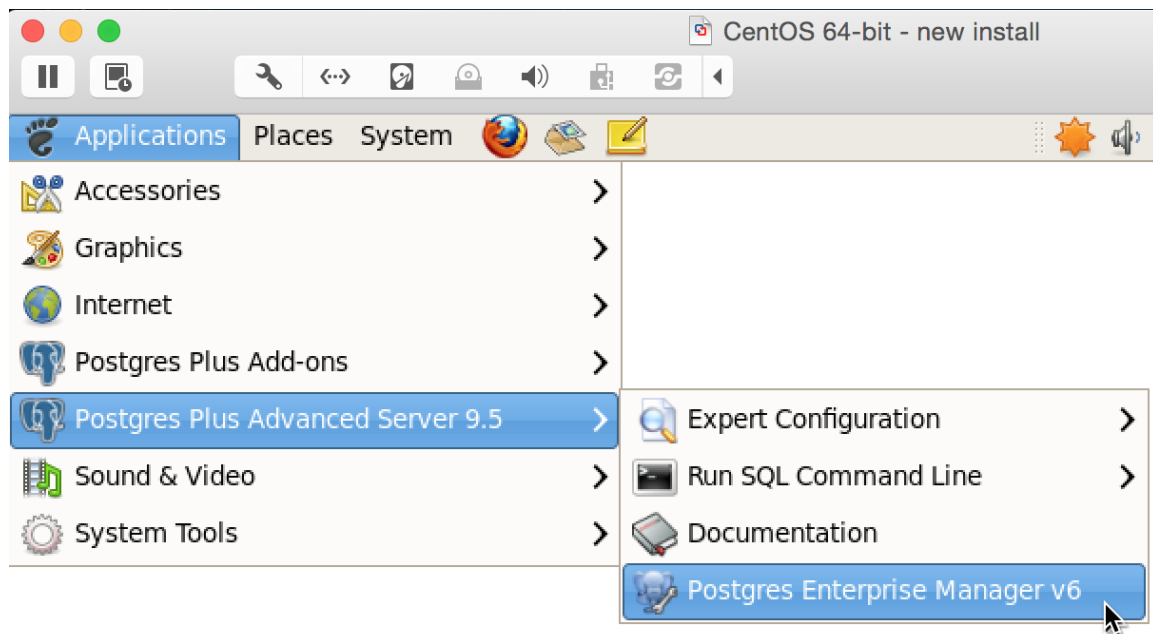


Figure 3.1 – Opening the PEM client.

When the PEM client opens, right-click on a server name in the `Object browser` tree control and select `Connect` from the context menu to connect to the server (see Figure 3.2).

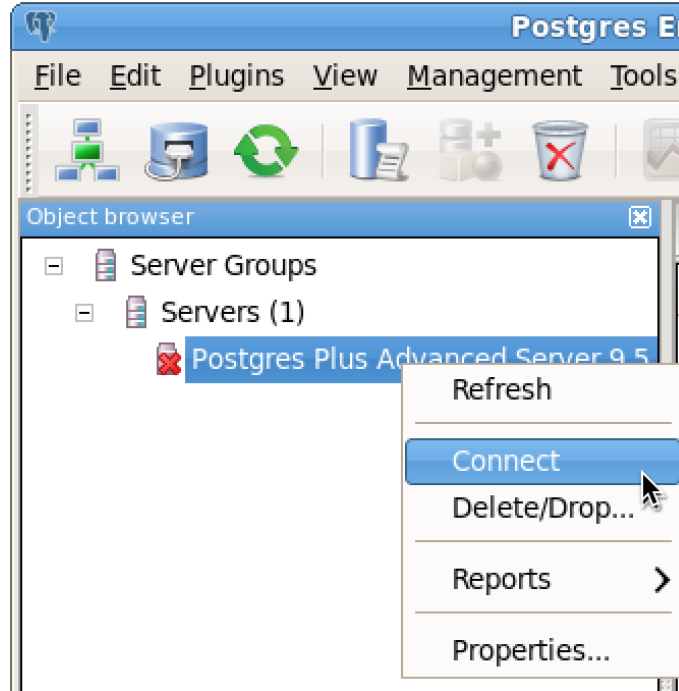


Figure 3.2 – Connecting to the PEM client.

If prompted, provide your password for authentication (see Figure 3.3).

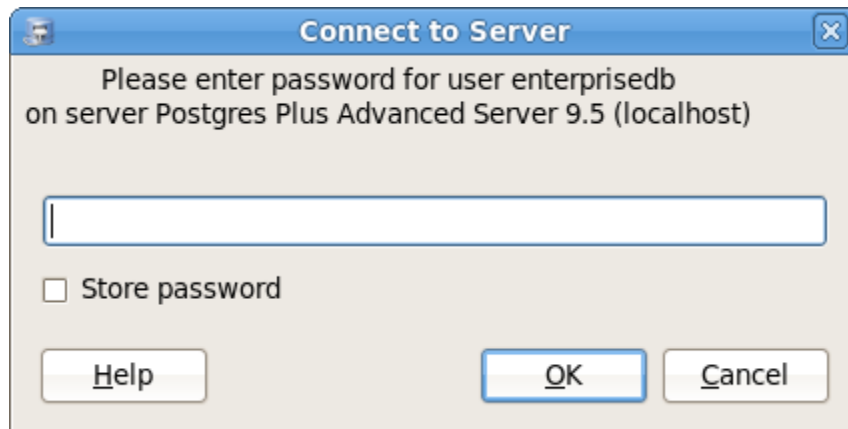


Figure 3.3 – If prompted, provide a password.

3.2 The PEM Client User Interface

After authenticating with the server, the server's node of the tree control will be populated with the objects that reside on that server. You can expand the tree control to view the database objects that reside on each server. Use the plus sign (+) to the left of a node to expand a segment of the tree control so you can review the objects that reside under a node; click the minus sign (-) to the left of a node to close that node.

Menus across the top of the client provide easy access to PEM functionality, and are context sensitive so only those tasks that are appropriate for the selected object are active. The graphical toolbar provides quick access to the most commonly used tasks and utilities.

The right pane of the client interface allows you to use tabbed browsing to review details about selected objects in the `Servers` tree control. The four tabs are `Properties`, `Statistics`, `Dependencies`, and `Dependents`.

The Properties Tab

The `Properties` tab displays the attributes of the object currently selected in the `Object Browser`. (see Figure 3.4).

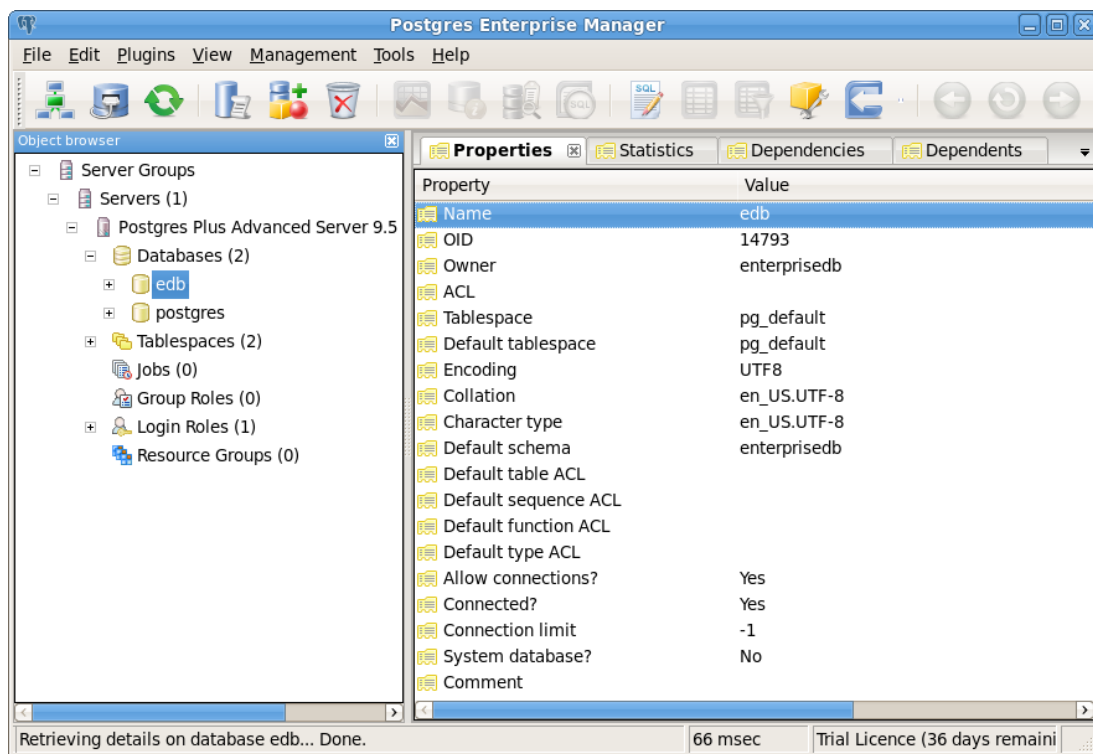


Figure 3.4 - The Properties tab in the PEM client.

The Statistics Tab

The `Statistics` tab displays available statistical information about the object currently selected in the Object Browser. (see Figure 3.5).

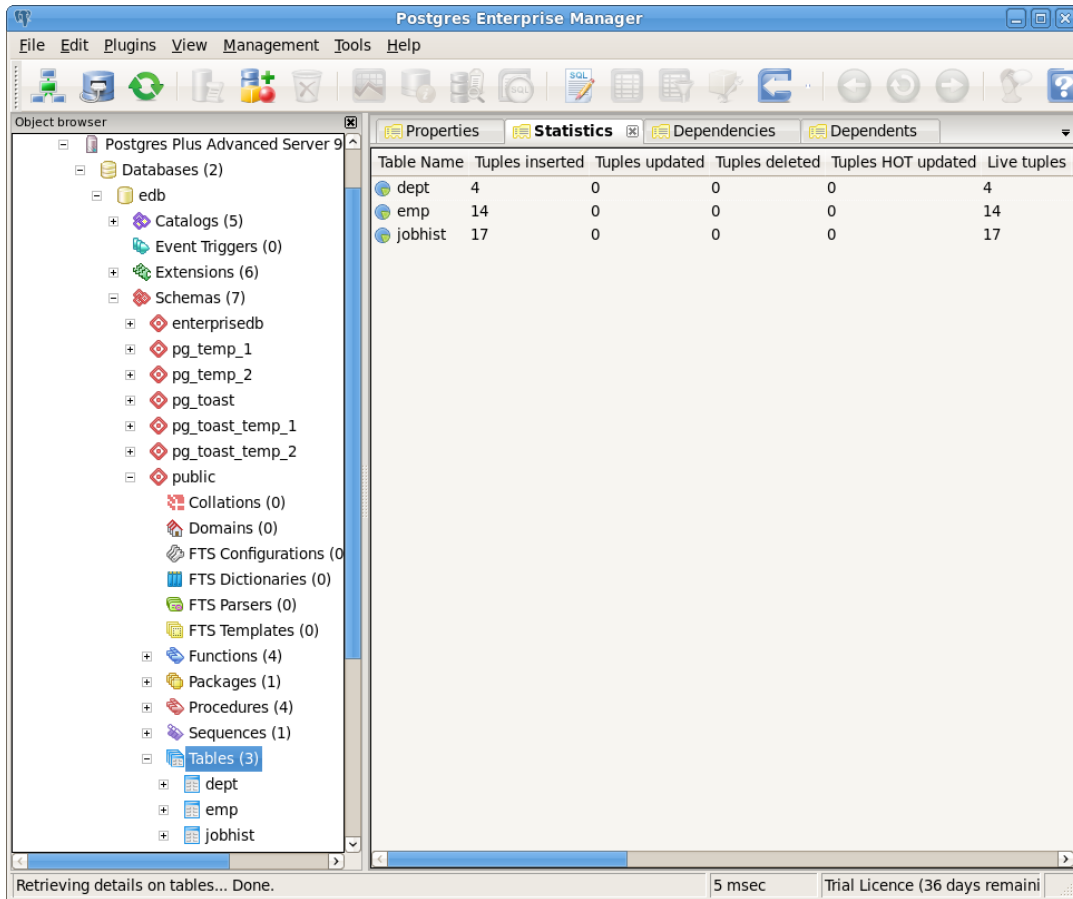


Figure 3.5 - The Statistics tab in the PEM client.

The Dependencies Tab

The Dependencies tab displays a list of objects on which the object currently selected in the Object Browser depends (see Figure 3.6).

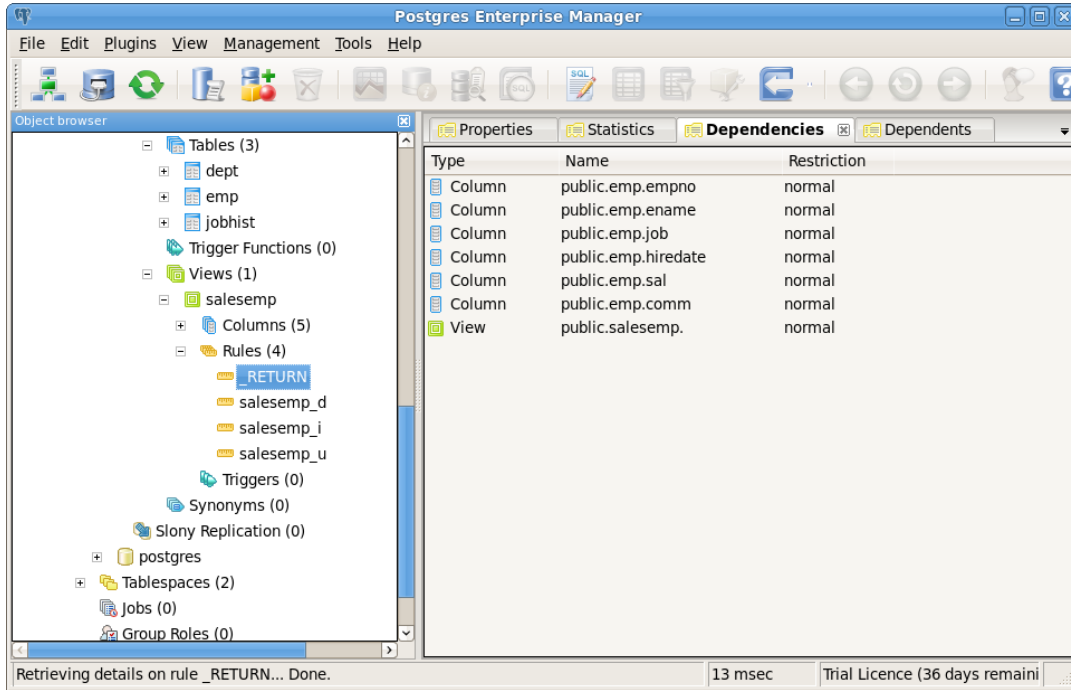


Figure 3.6 - The Dependencies tab in the PEM client.

The Dependents Tab

The Dependents tab displays a list of objects that depend on the object currently selected in the Object Browser (see Figure 3.7).

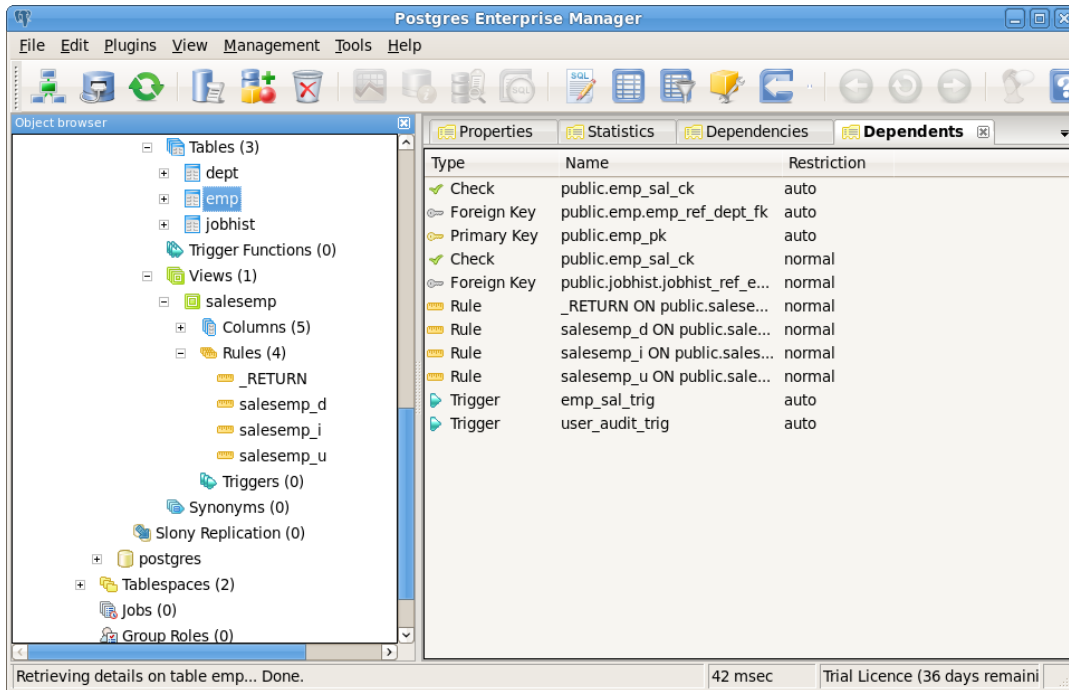


Figure 3.7 - The Dependents tab in the PEM client.

The PEM server works with the PEM client to provide a variety of tools and utilities that can help monitor and manage your Postgres servers. For more information about other PEM tools, see the PostgreSQL Enterprise Manager Getting Started Guide, available at

<http://www.enterprisedb.com/products-services-training/products/documentation/enterpriseedition>

3.3 Using the PEM Client

You can use the PEM client to:

- create database objects
- populate a table with data
- create roles
- manage privileges

and more...

3.3.1 Creating a Table

The `New Table` dialog contains fields that describe the properties of a table. To open the `New Table` dialog, right click on the `Tables` node of the tree control, and choose `New Table` from the context menu. The `New Table` dialog opens (see Figure 3.8).

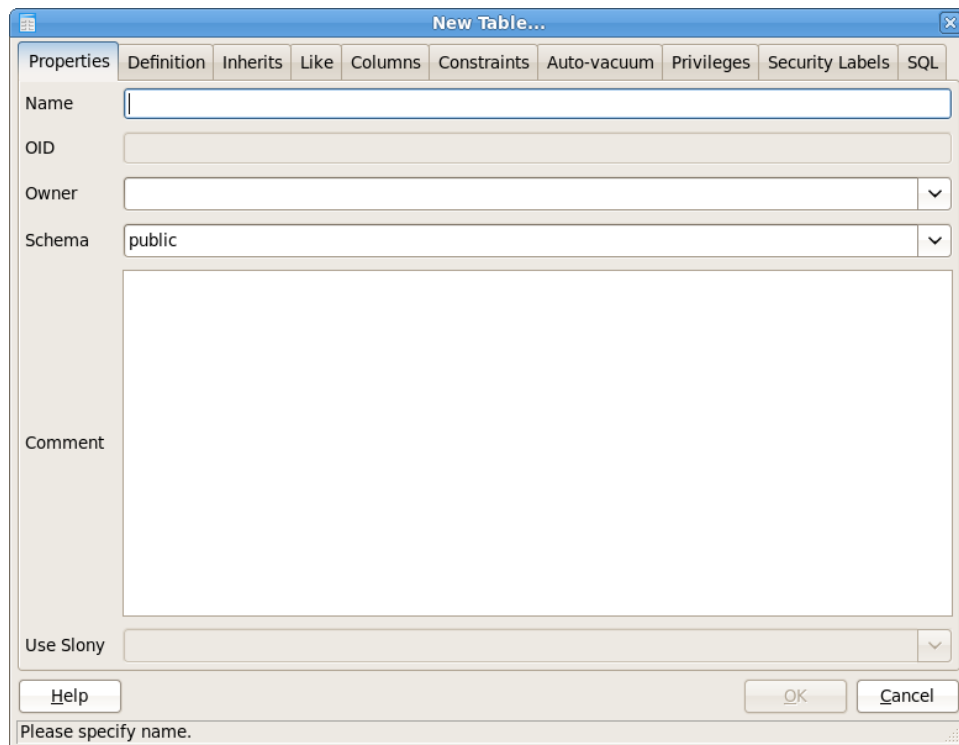


Figure 3.8 - The New Table dialog.

Use the tabs on the `New Table` dialog to define the attributes of a table.

When you've specified the table properties on the tabs of the `New Table` dialog (the columns, constraints, privileges and other properties), you can review the SQL code that creates the table on the `SQL` tab (see Figure 3.9).

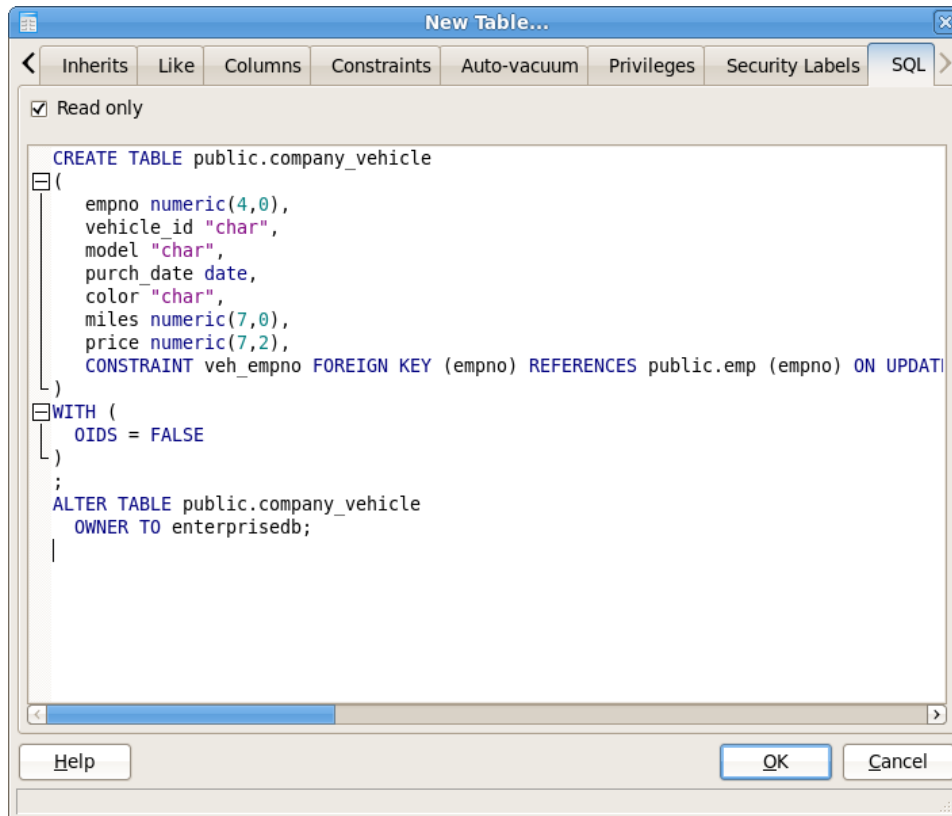


Figure 3.9 – Review the SQL code on the SQL tab.

3.3.2 Viewing and Managing Data

You can use the PEM client to review data that resides in the tables on your server. If you installed Advanced Server with the sample data (the dept, emp, and jobhist, tables) you can view the data in these tables by right-clicking the table name in the Object Browser and selecting View Data (see Figure 3.10).

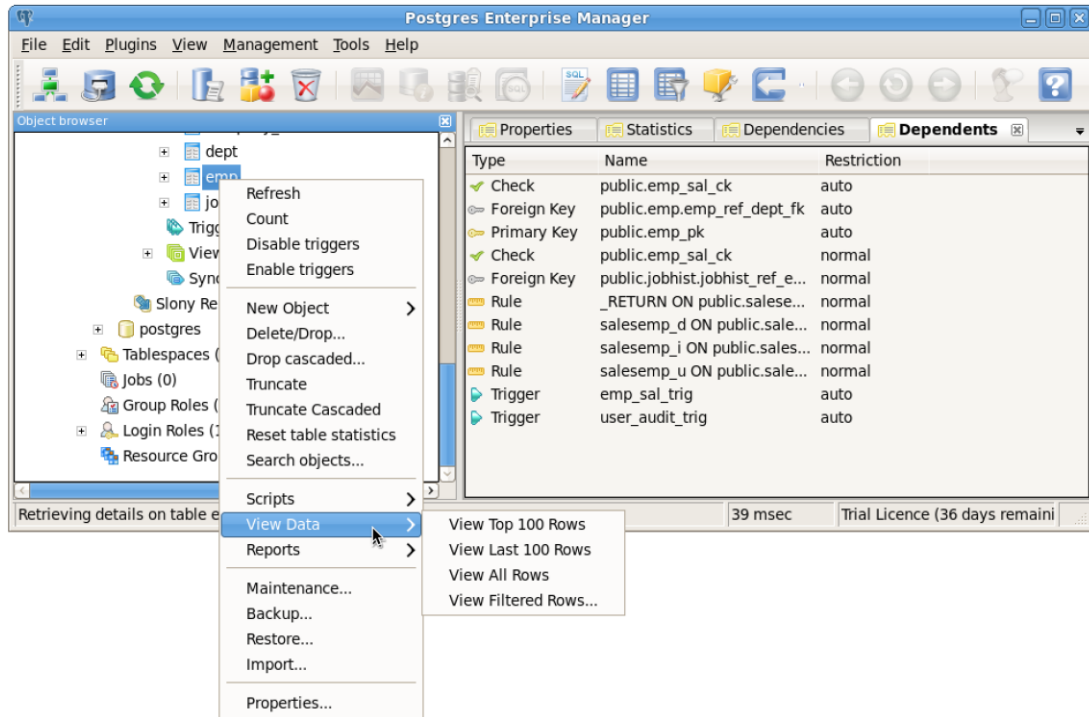


Figure 3.10 - The View Data context menu.

The Edit Data dialog opens, displaying the table's data. Click inside a cell to change a value (see Figure 3.11).

The screenshot shows the 'Edit Data' dialog for the 'emp' table. The table displays 14 rows of data with columns: empno, ename, job, mgr, hiredate, sal, comm, and deptno. A mouse cursor is pointing to the 'CLERK' job title in row 12.

	empno [PK] numeric(4,0)	ename character varying(10)	job character varying(9)	mgr numeric(4,0)	hiredate timestamp without time zone	sal numeric(7,2)	comm numeric(7,2)	deptno numeric(2,0)
1	7369	SMITH	CLERK	7962	1980-12-17 00:00:00	800.00		20
2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00	1600.00	300.00	30
3	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00	1250.00	500.00	30
4	7566	JONES	MANAGER	7839	1981-04-02 00:00:00	2975.00		20
5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00	1250.00	1400.00	30
6	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00	2850.00		30
7	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00	2450.00		10
8	7788	SCOTT	ANALYST	7566	1987-04-19 00:00:00	3000.00		20
9	7839	KING	PRESIDENT		1981-11-17 00:00:00	5000.00		10
10	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00	1500.00	0.00	30
11	7876	ADAMS	CLERK	7788	1987-05-23 00:00:00	1100.00		20
12	7900	JAMES	CLERK	7698	1981-12-03 00:00:00	950.00		30
13	7902	FORD	ANALYST	7566	1981-12-03 00:00:00	3000.00		20
14	7934	MILLER	CLERK	7782	1982-01-23 00:00:00	1300.00		10
*								

Figure 3.11 - The Edit Data dialog.

Alternately, you can access the `Edit Data` dialog box from an icon on the toolbar (see Figure 3.12).

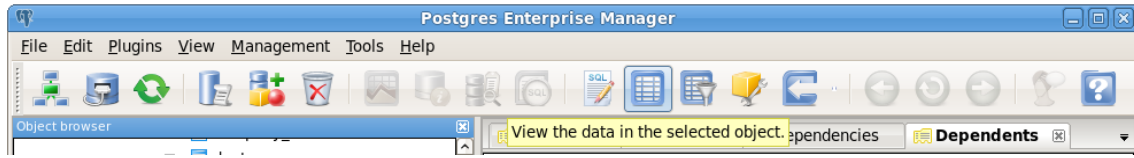


Figure 3.12 - The View Data Icon.

3.3.3 Querying Data

To the right of the `View Data` icon is the `View Filtered Rows` icon (see Figure 3.13). Click the icon to open a dialog that allows you to apply a filter to a set of data.

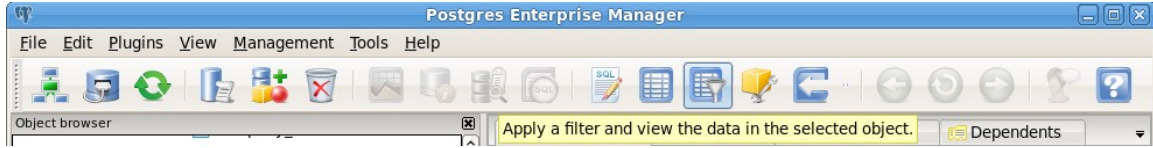


Figure 3.13 - The View Filtered Data Icon.

Specify a condition in the `View Data Options` dialog to filter and view data with the condition applied by entering a `Filter String` (see Figure 3.14).

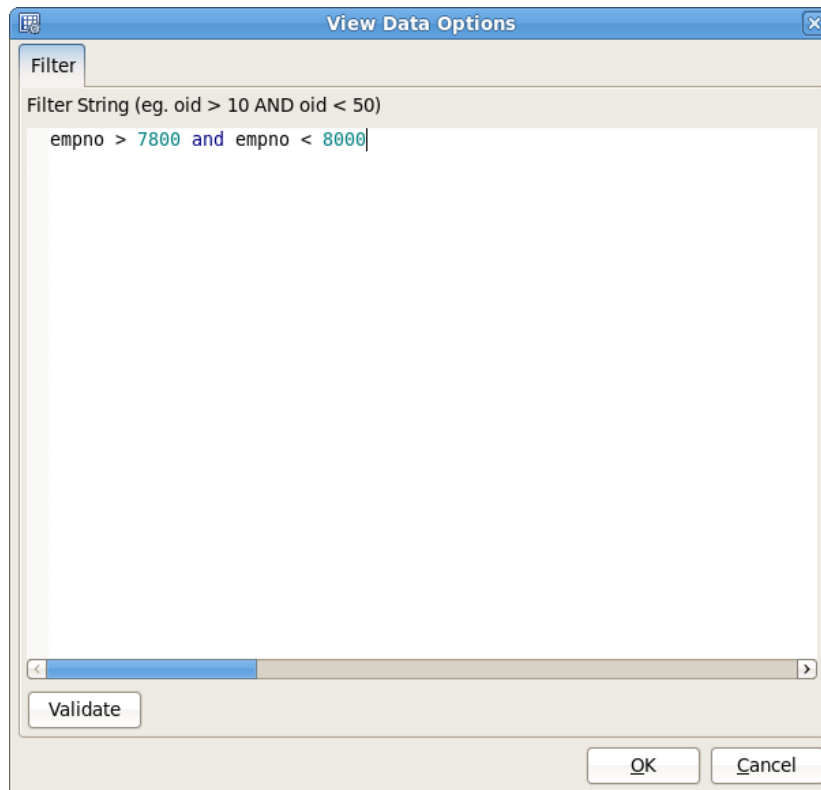
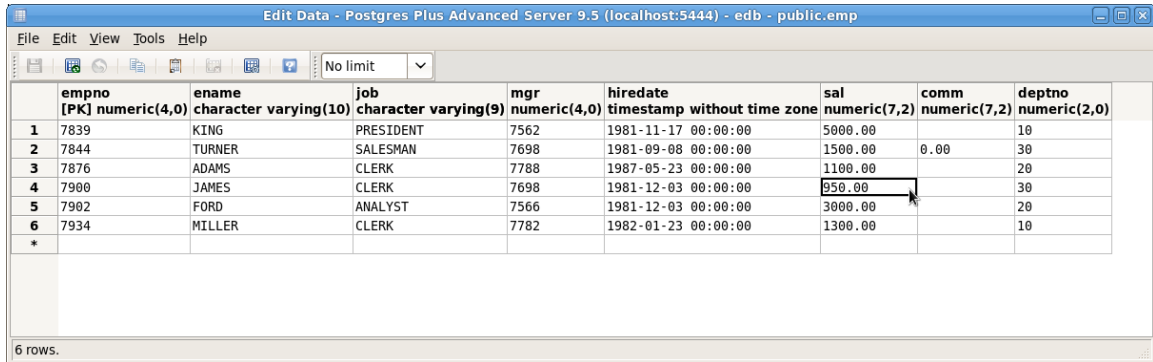


Figure 3.14 - The View Data Options dialog.

Getting Started with EDB Postgres Advanced Server on Linux

When you've defined the filter, click OK to display the result set in an editable table (see Figure 3.15).



The screenshot shows the 'Edit Data' dialog for the 'public.emp' table. The table has 6 rows and 8 columns. The columns are: empno [PK] numeric(4,0), ename character varying(10), job character varying(9), mgr numeric(4,0), hiredate timestamp without time zone, sal numeric(7,2), comm numeric(7,2), and deptno numeric(2,0). The data is as follows:

	empno [PK] numeric(4,0)	ename character varying(10)	job character varying(9)	mgr numeric(4,0)	hiredate timestamp without time zone	sal numeric(7,2)	comm numeric(7,2)	deptno numeric(2,0)
1	7839	KING	PRESIDENT	7562	1981-11-17 00:00:00	5000.00		10
2	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00	1500.00	0.00	30
3	7876	ADAMS	CLERK	7788	1987-05-23 00:00:00	1100.00		20
4	7900	JAMES	CLERK	7698	1981-12-03 00:00:00	950.00		30
5	7902	FORD	ANALYST	7566	1981-12-03 00:00:00	3000.00		20
6	7934	MILLER	CLERK	7782	1982-01-23 00:00:00	1300.00		10

The status bar at the bottom indicates '6 rows.' A mouse cursor is pointing to the '950.00' value in the 'sal' column of the fifth row.

Figure 3.15 – Filter Results in the Edit Data dialog.

4 Getting Started with EDB-PSQL

You can use the `psql` client to create and manage your database; for more detailed information about the `psql` client, please see the PostgreSQL core documentation, available at:

<http://www.postgresql.org/docs/9.5/static/app-psql.html>

4.1 Connecting with the EDB-PSQL Client on CentOS or RHEL 7.x

If you have performed an installation with the Advanced Server interactive installer, you can easily access the `psql` client through the Applications menu. Navigate through the Advanced Server 9.5 menu and select EDB-PSQL (see Figure 4.1).

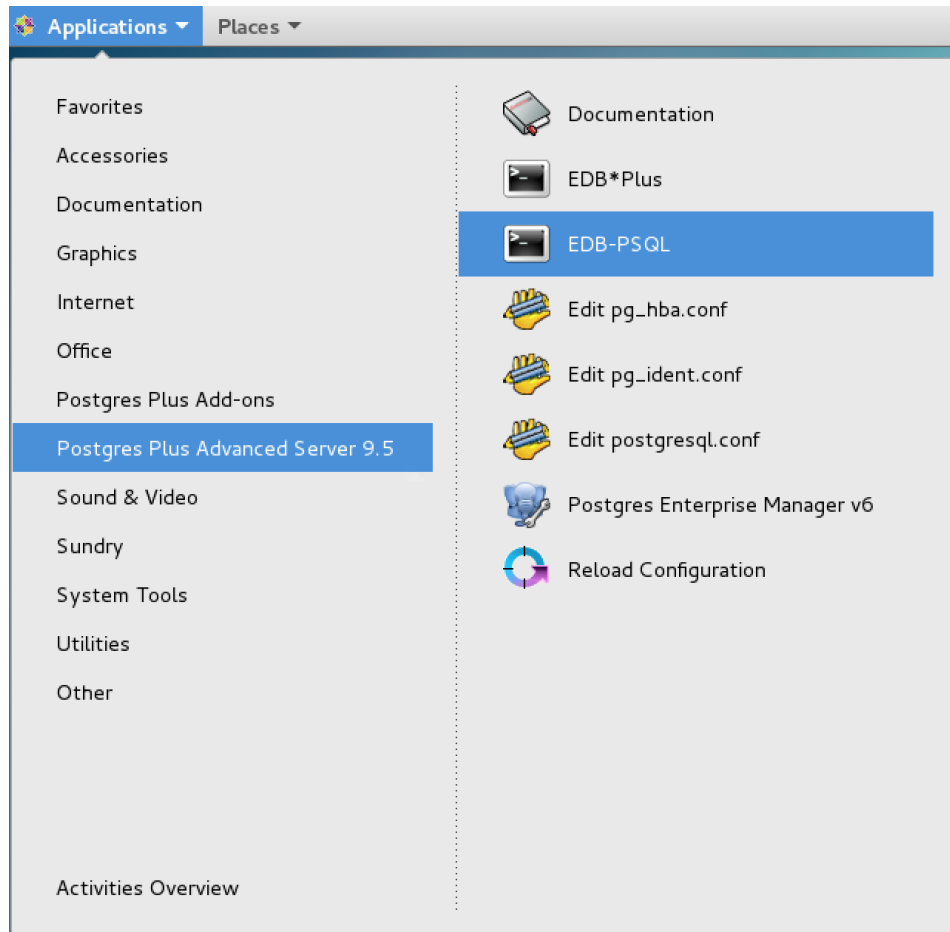


Figure 4.1 – Opening the PSQL client.

When the psql client opens, provide connection and authentication information for your server (see Figure 4.2).

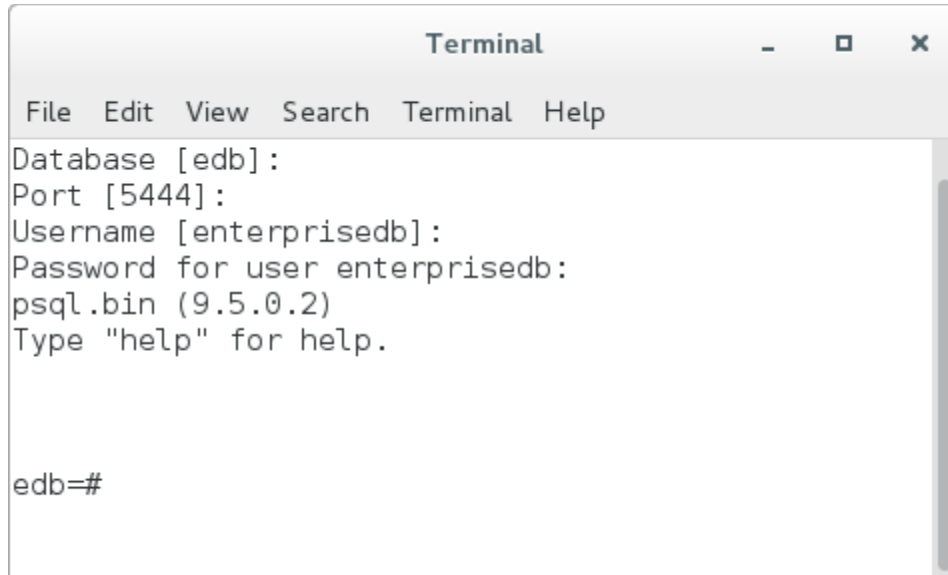


Figure 4.2 – The PSQL Client.

4.2 Connecting with the EDB-PSQL Client on CentOS or RHEL 6.x

If you have performed an installation with the Advanced Server interactive installer, you can easily access the `psql` client through the Applications or Start menu. Navigate through the Advanced Server 9.5 menu to the Run SQL Command Line menu, and select EDB-PSQL (see Figure 4.3).

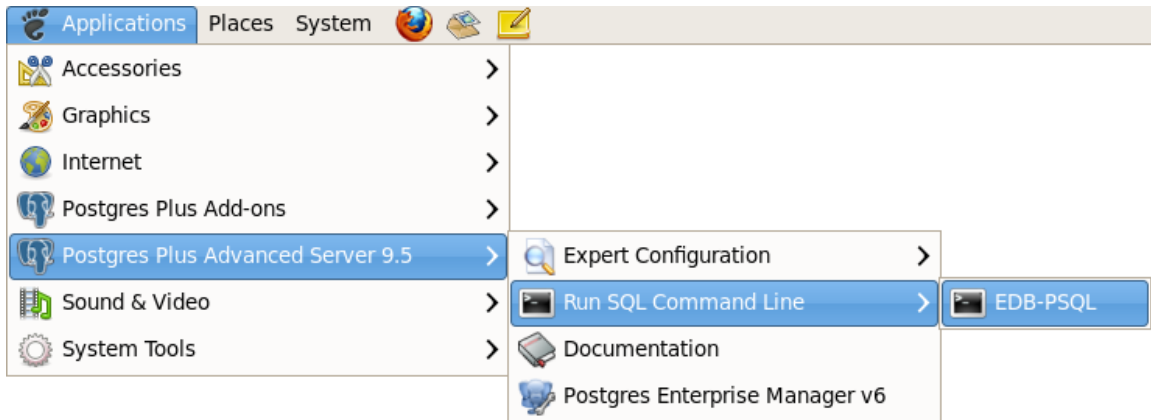


Figure 4.3 – Opening the PSQL client.

When the `psql` client opens, provide connection and authentication information for your server (see Figure 4.4).

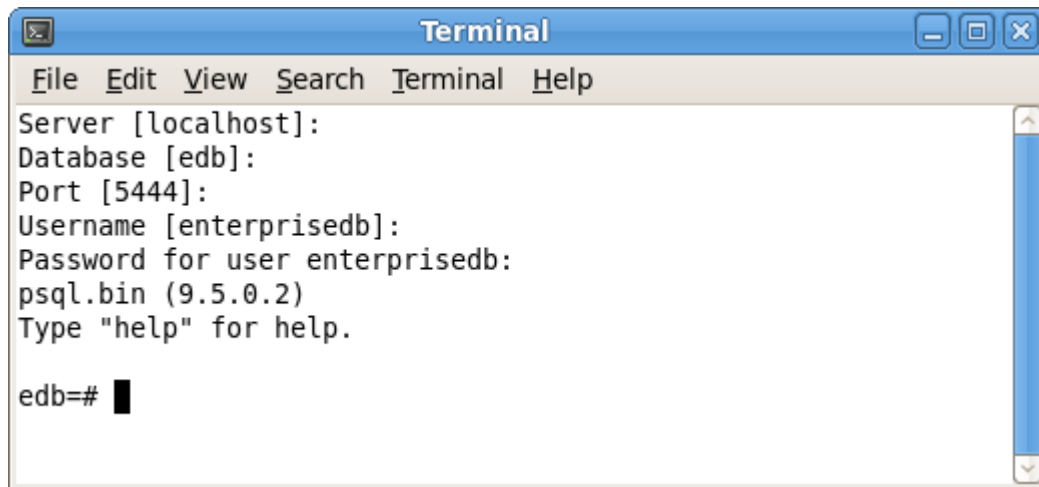


Figure 4.4 – The PSQL Client.

4.3 PSQL Meta-Commands

A psql meta-command is a command prefaced with an unquoted backslash that (unlike SQL commands) works only in the psql client. You can use psql meta-commands to retrieve information about your server and the objects that reside on the server, or to make changes to the psql environment. Some useful meta-commands are:

Meta-Command	Description
<code>\c [dbname]</code> or <code>\connect [dbname [username]</code> <code>[host] [port]] conninfo</code>	Establishes a new connection to a database.
<code>\d table_name</code>	Shows the structure of the specified table.
<code>\d+</code>	Examine a table and its child tables.
<code>\dt</code>	Lists all tables in current database.
<code>\l</code>	Lists all available databases.
<code>\q</code>	Quits.
<code>\s</code>	Runs in single-step mode.
<code>\U username</code>	Connects to the database as the user username instead of the default.
<code>\w</code>	Forces psql to prompt for a password before connecting to a database.
<code>\x</code>	Displays results in expanded view. This command acts as a toggle; the next <code>\x</code> disables the functionality.
<code>\? Or \h</code>	Displays psql help.

To view a complete list of meta-commands, please see the PostgreSQL Core Documentation, available at:

<http://www.postgresql.org/docs/9.5/static/app-psql.html>

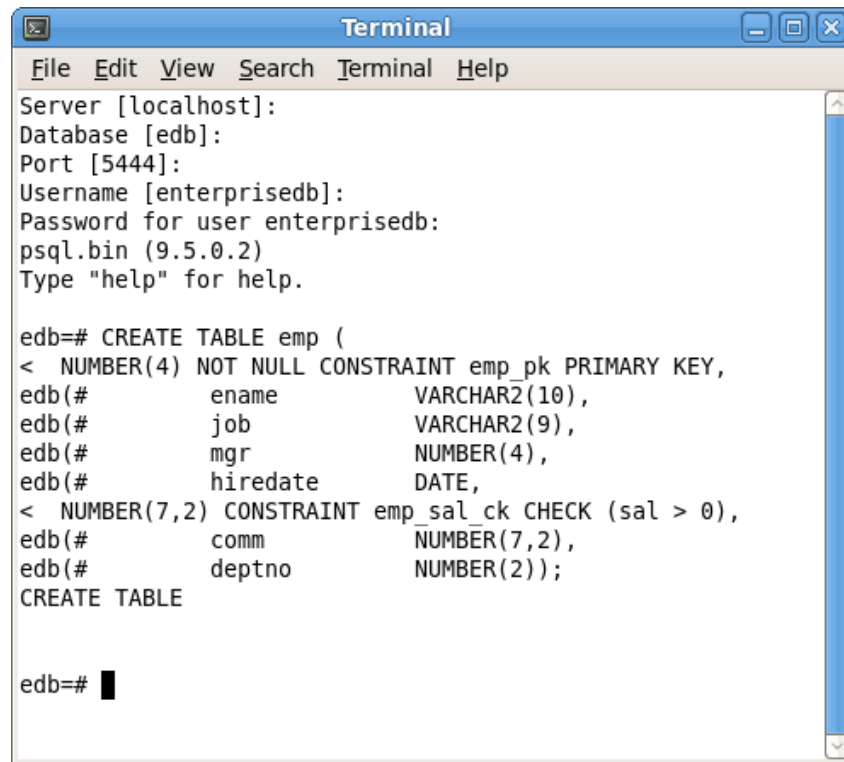
4.4 Using the PSQL Client

After connecting with the psql client, you can use psql meta-commands, SQL commands, and Postgres functions to create, manage, and query database objects and roles.

The following examples create a table that works with the existing sample database distributed with the Advanced Server graphical installer. If you have installed Advanced Server with sample tables (dept, emp, and jobhist), you do not need to create the emp table before creating the company_vehicle table defined in the example and performing the queries. If you have not installed the sample tables, create the emp table with the following command:

```
edb=# CREATE TABLE emp (
  empno      NUMBER(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
  ename      VARCHAR2(10),
  job        VARCHAR2(9),
  mgr        NUMBER(4),
  hiredate   DATE,
  sal        NUMBER(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
  comm       NUMBER(7,2),
  deptno     NUMBER(2));
```

When the command completes successfully, the psql client will display CREATE TABLE (see Figure 4.5).



```
Terminal
File Edit View Search Terminal Help
Server [localhost]:
Database [edb]:
Port [5444]:
Username [enterisedb]:
Password for user enterisedb:
psql.bin (9.5.0.2)
Type "help" for help.

edb=# CREATE TABLE emp (
< NUMBER(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
edb(#      ename          VARCHAR2(10),
edb(#      job           VARCHAR2(9),
edb(#      mgr           NUMBER(4),
edb(#      hiredate      DATE,
< NUMBER(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
edb(#      comm          NUMBER(7,2),
edb(#      deptno        NUMBER(2));
CREATE TABLE

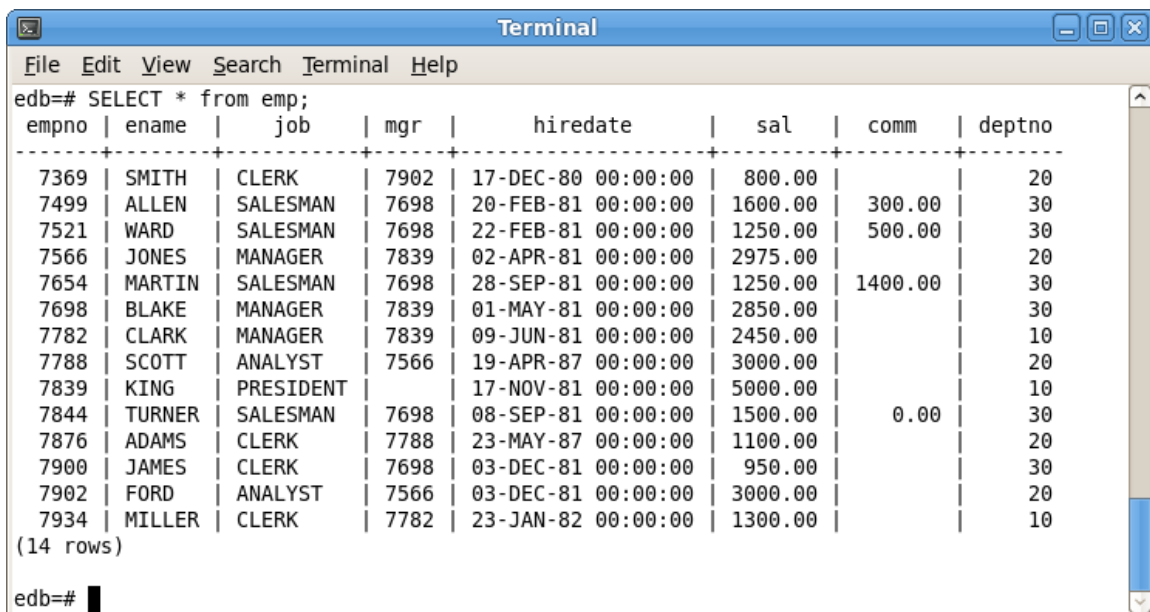
edb=# █
```

Figure 4.5 – Creating the emp table.

After creating the sample `emp` table, you can use the following commands to populate the `emp` table:

```
edb=# INSERT INTO emp VALUES (7369, 'SMITH', 'CLERK', 7902, '17-DEC-80', 800, NULL, 20);
INSERT INTO emp VALUES (7499, 'ALLEN', 'SALESMAN', 7698, '20-FEB-81', 1600, 300, 30);
INSERT INTO emp VALUES (7521, 'WARD', 'SALESMAN', 7698, '22-FEB-81', 1250, 500, 30);
INSERT INTO emp VALUES (7566, 'JONES', 'MANAGER', 7839, '02-APR-81', 2975, NULL, 20);
INSERT INTO emp VALUES (7654, 'MARTIN', 'SALESMAN', 7698, '28-SEP-81', 1250, 1400, 30);
INSERT INTO emp VALUES (7698, 'BLAKE', 'MANAGER', 7839, '01-MAY-81', 2850, NULL, 30);
INSERT INTO emp VALUES (7782, 'CLARK', 'MANAGER', 7839, '09-JUN-81', 2450, NULL, 10);
INSERT INTO emp VALUES (7788, 'SCOTT', 'ANALYST', 7566, '19-APR-87', 3000, NULL, 20);
INSERT INTO emp VALUES (7839, 'KING', 'PRESIDENT', NULL, '17-NOV-81', 5000, NULL, 10);
INSERT INTO emp VALUES (7844, 'TURNER', 'SALESMAN', 7698, '08-SEP-81', 1500, 0, 30);
INSERT INTO emp VALUES (7876, 'ADAMS', 'CLERK', 7788, '23-MAY-87', 1100, NULL, 20);
INSERT INTO emp VALUES (7900, 'JAMES', 'CLERK', 7698, '03-DEC-81', 950, NULL, 30);
INSERT INTO emp VALUES (7902, 'FORD', 'ANALYST', 7566, '03-DEC-81', 3000, NULL, 20);
INSERT INTO emp VALUES (7934, 'MILLER', 'CLERK', 7782, '23-JAN-82', 1300, NULL, 10);
```

You can then use the `SELECT` statement to retrieve and view a list of employees (see Figure 4.6):



```
edb=# SELECT * from emp;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	17-DEC-80 00:00:00	800.00		20
7499	ALLEN	SALESMAN	7698	20-FEB-81 00:00:00	1600.00	300.00	30
7521	WARD	SALESMAN	7698	22-FEB-81 00:00:00	1250.00	500.00	30
7566	JONES	MANAGER	7839	02-APR-81 00:00:00	2975.00		20
7654	MARTIN	SALESMAN	7698	28-SEP-81 00:00:00	1250.00	1400.00	30
7698	BLAKE	MANAGER	7839	01-MAY-81 00:00:00	2850.00		30
7782	CLARK	MANAGER	7839	09-JUN-81 00:00:00	2450.00		10
7788	SCOTT	ANALYST	7566	19-APR-87 00:00:00	3000.00		20
7839	KING	PRESIDENT		17-NOV-81 00:00:00	5000.00		10
7844	TURNER	SALESMAN	7698	08-SEP-81 00:00:00	1500.00	0.00	30
7876	ADAMS	CLERK	7788	23-MAY-87 00:00:00	1100.00		20
7900	JAMES	CLERK	7698	03-DEC-81 00:00:00	950.00		30
7902	FORD	ANALYST	7566	03-DEC-81 00:00:00	3000.00		20
7934	MILLER	CLERK	7782	23-JAN-82 00:00:00	1300.00		10

(14 rows)

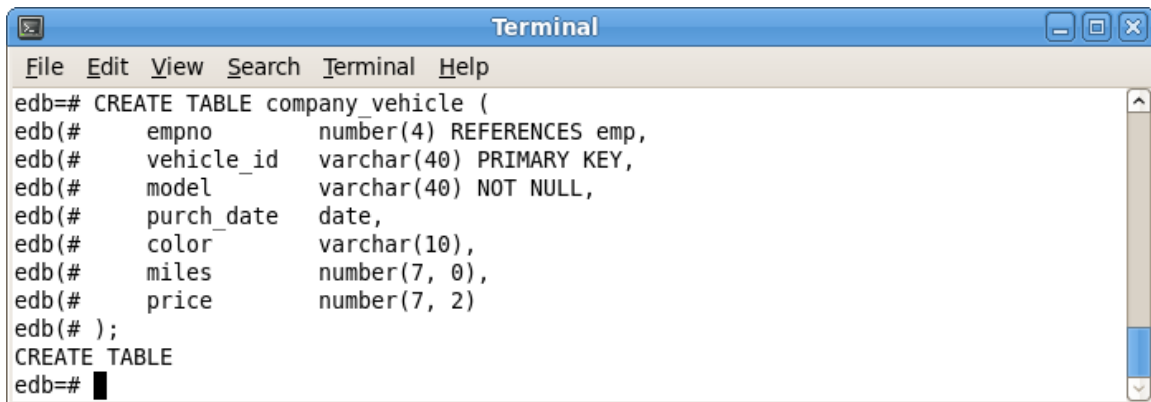
```
edb=#
```

Figure 4.6 – The contents of the `emp` table.

In our example, we will assign a company car to each of the current members of the `emp` table. The car information will be stored in a table named `company_vehicle`. Use the `CREATE TABLE` SQL command at the `psql` command line to define a table that holds a company car assignment for each employee:

```
edb=# CREATE TABLE company_vehicle (
  empno      number(4) REFERENCES emp,
  vehicle_id varchar(40) PRIMARY KEY,
  model      varchar(40) NOT NULL,
  purch_date date,
  color      varchar(10),
  miles      number(7, 0),
  price      number(7, 2)
);
```

When the command completes successfully, the client returns `CREATE TABLE` (see Figure 4.7).



```
Terminal
File Edit View Search Terminal Help
edb=# CREATE TABLE company_vehicle (
edb(#      empno      number(4) REFERENCES emp,
edb(#      vehicle_id varchar(40) PRIMARY KEY,
edb(#      model      varchar(40) NOT NULL,
edb(#      purch_date date,
edb(#      color      varchar(10),
edb(#      miles      number(7, 0),
edb(#      price      number(7, 2)
edb(# );
CREATE TABLE
edb=# █
```

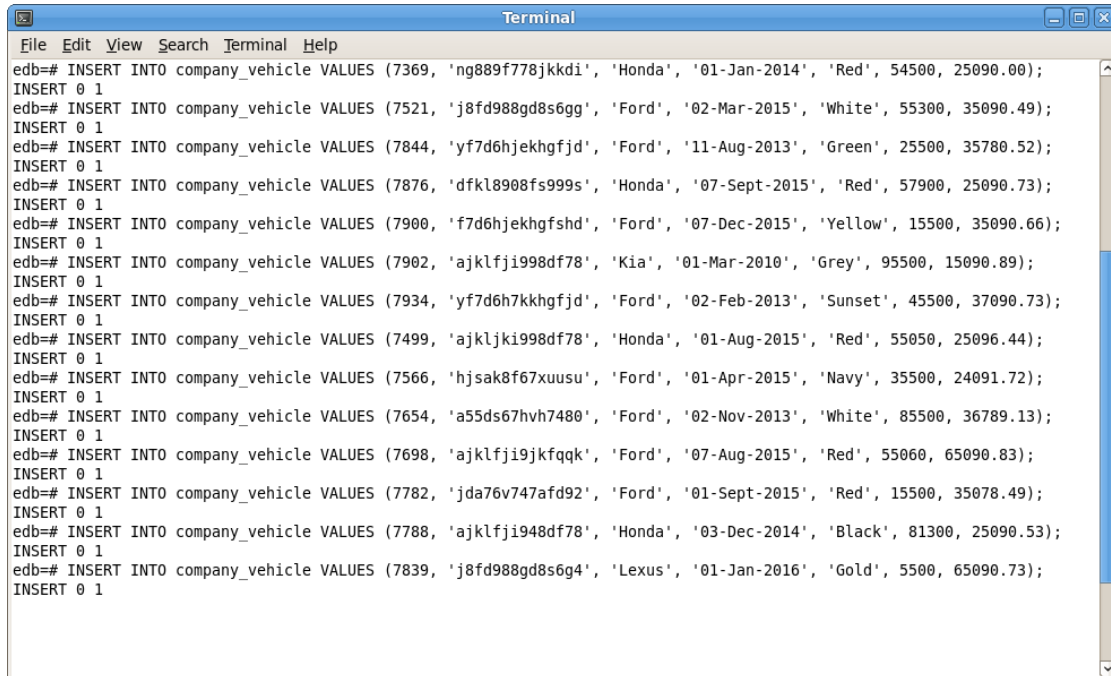
Figure 4.7 – Creating the `company_vehicle` table.

Then, use `INSERT` statements to add data to the table:

```
INSERT INTO company_vehicle VALUES (7369, 'ng889f778jkkdi', 'Honda', '01-Jan-2014', 'Red', 54500, 25090.00);
INSERT INTO company_vehicle VALUES (7521, 'j8fd988gd8s6gg', 'Ford', '02-Mar-2015', 'White', 55300, 35090.49);
INSERT INTO company_vehicle VALUES (7844, 'yf7d6hjekhgfjd', 'Ford', '11-Aug-2013', 'Green', 25500, 35780.52);
INSERT INTO company_vehicle VALUES (7876, 'dfkl8908fs999s', 'Honda', '07-Sept-2015', 'Red', 57900, 25090.73);
INSERT INTO company_vehicle VALUES (7900, 'f7d6hjekhgfshd', 'Ford', '07-Dec-2015', 'Yellow', 15500, 35090.66);
INSERT INTO company_vehicle VALUES (7902, 'ajklfji998df78', 'Kia', '01-Mar-2010', 'Grey', 95500, 15090.89);
INSERT INTO company_vehicle VALUES (7934, 'yf7d6h7kkhgfjd', 'Ford', '02-Feb-2013', 'Sunset', 45500, 37090.73);
INSERT INTO company_vehicle VALUES (7499, 'ajkljki998df78', 'Honda', '01-Aug-2015', 'Red', 55050, 25096.44);
INSERT INTO company_vehicle VALUES (7566, 'hjsak8f67xuusu', 'Ford', '01-Apr-2015', 'Navy', 35500, 24091.72);
INSERT INTO company_vehicle VALUES (7654, 'a55ds67hvh7480', 'Ford', '02-Nov-2013', 'White', 85500, 36789.13);
```

```
INSERT INTO company_vehicle VALUES (7698, 'ajklfji9jkfqqk', 'Ford', '07-Aug-2015', 'Red', 55060, 65090.83);
INSERT INTO company_vehicle VALUES (7782, 'jda76v747afd92', 'Ford', '01-Sept-2015', 'Red', 15500, 35078.49);
INSERT INTO company_vehicle VALUES (7788, 'ajklfji948df78', 'Honda', '03-Dec-2014', 'Black', 81300, 25090.53);
INSERT INTO company_vehicle VALUES (7839, 'j8fd988gd8s6g4', 'Lexus', '01-Jan-2016', 'Gold', 5500, 65090.73);
```

The `vehicle_id` column is the unique key for the table, so each car must have a unique vehicle ID number (see Figure 4.8).



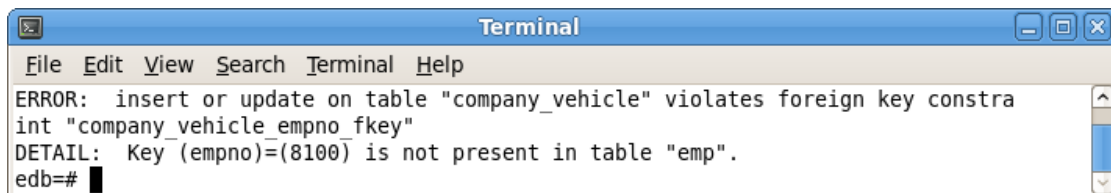
```
Terminal
File Edit View Search Terminal Help
edb=# INSERT INTO company_vehicle VALUES (7369, 'ng889f778jkkdi', 'Honda', '01-Jan-2014', 'Red', 54500, 25090.00);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7521, 'j8fd988gd8s6gg', 'Ford', '02-Mar-2015', 'White', 55300, 35090.49);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7844, 'yf7d6hjekhgfjd', 'Ford', '11-Aug-2013', 'Green', 25500, 35780.52);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7876, 'dfkl8908fs999s', 'Honda', '07-Sept-2015', 'Red', 57900, 25090.73);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7900, 'f7d6hjekhghsd', 'Ford', '07-Dec-2015', 'Yellow', 15500, 35090.66);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7902, 'ajklfji998df78', 'Kia', '01-Mar-2010', 'Grey', 95500, 15090.89);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7934, 'yf7d6h7kkgfjd', 'Ford', '02-Feb-2013', 'Sunset', 45500, 37090.73);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7499, 'ajkljki998df78', 'Honda', '01-Aug-2015', 'Red', 55050, 25096.44);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7566, 'hjsak8f67xuusu', 'Ford', '01-Apr-2015', 'Navy', 35500, 24091.72);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7654, 'a55ds67vhv7480', 'Ford', '02-Nov-2013', 'White', 85500, 36789.13);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7698, 'ajklfji9jkfqqk', 'Ford', '07-Aug-2015', 'Red', 55060, 65090.83);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7782, 'jda76v747afd92', 'Ford', '01-Sept-2015', 'Red', 15500, 35078.49);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7788, 'ajklfji948df78', 'Honda', '03-Dec-2014', 'Black', 81300, 25090.53);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7839, 'j8fd988gd8s6g4', 'Lexus', '01-Jan-2016', 'Gold', 5500, 65090.73);
INSERT 0 1
```

Figure 4.8 – Adding rows to the `company_vehicle` table.

Note that if you try to INSERT a row for a vehicle with an invalid employee number (see Figure 4.9):

```
edb=# INSERT INTO company_vehicle VALUES ('8100', 'VIN-8100-0004', 'Hyundai', '2011/07/07', 'Blue', '101780', '11000.00');
```

The server will return an error:



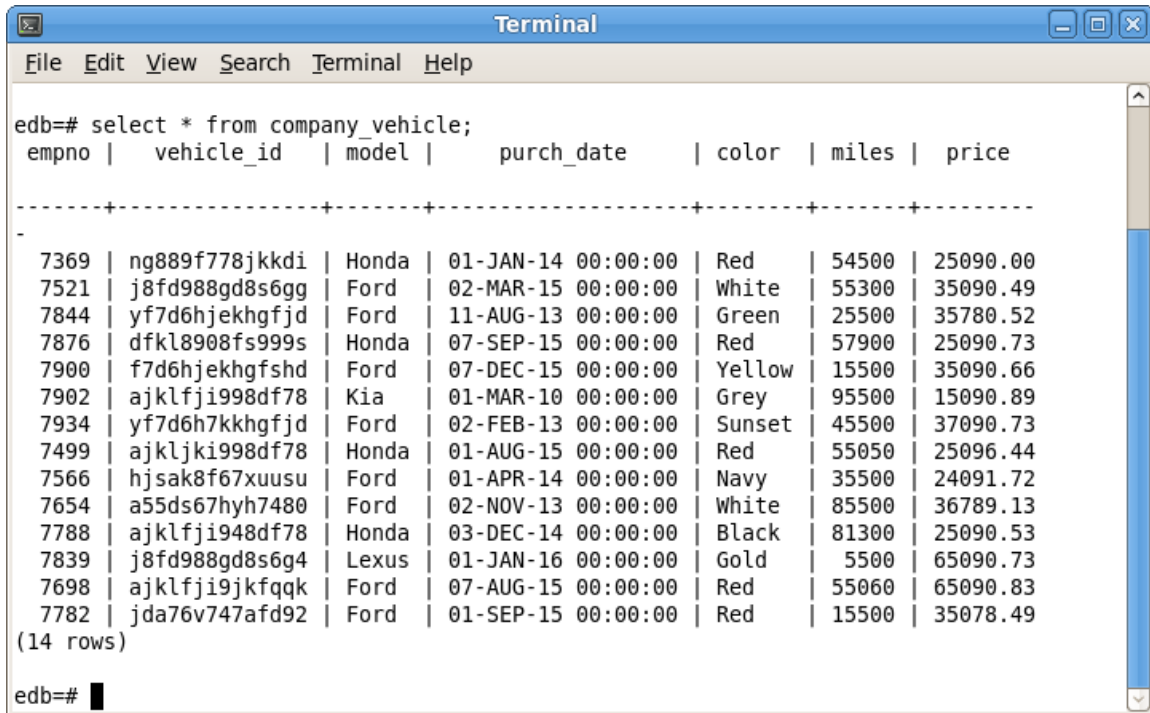
```
Terminal
File Edit View Search Terminal Help
ERROR: insert or update on table "company_vehicle" violates foreign key constraint "company_vehicle_empno_fkey"
DETAIL: Key (empno)=(8100) is not present in table "emp".
edb=# █
```

Figure 4.9 – The employee number is not present in the `emp` table.

After populating the `company_vehicle` table, you can use a `SELECT` statement to review your work:

```
edb=# SELECT * FROM company_vehicle;
```

The `psql` client displays the table's contents (see Figure 4.10).



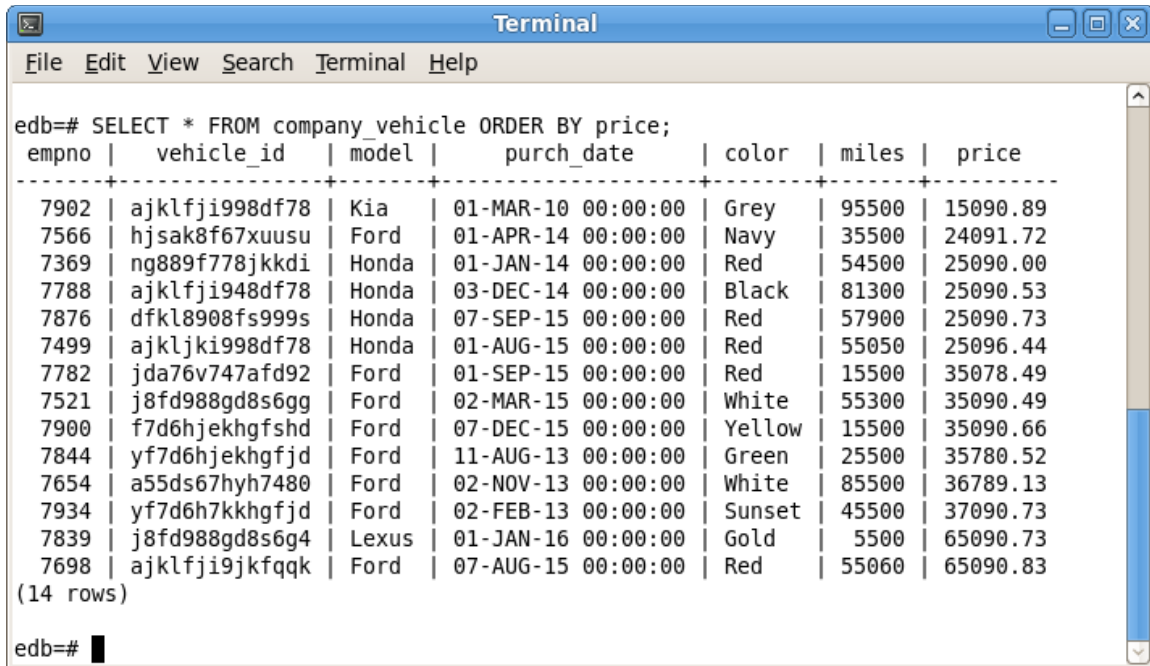
```
edb=# select * from company_vehicle;
 empno |  vehicle_id  | model |   purch_date   | color | miles | price
-----+-----+-----+-----+-----+-----+-----
 7369 | ng889f778jkkdi | Honda | 01-JAN-14 00:00:00 | Red   | 54500 | 25090.00
 7521 | j8fd988gd8s6gg | Ford  | 02-MAR-15 00:00:00 | White | 55300 | 35090.49
 7844 | yf7d6hjekhgfd | Ford  | 11-AUG-13 00:00:00 | Green | 25500 | 35780.52
 7876 | dfkl8908fs999s | Honda | 07-SEP-15 00:00:00 | Red   | 57900 | 25090.73
 7900 | f7d6hjekhgfdshd | Ford  | 07-DEC-15 00:00:00 | Yellow | 15500 | 35090.66
 7902 | ajklfji998df78 | Kia   | 01-MAR-10 00:00:00 | Grey  | 95500 | 15090.89
 7934 | yf7d6h7kkhgfd | Ford  | 02-FEB-13 00:00:00 | Sunset | 45500 | 37090.73
 7499 | ajkljki998df78 | Honda | 01-AUG-15 00:00:00 | Red   | 55050 | 25096.44
 7566 | hjsak8f67xuusu | Ford  | 01-APR-14 00:00:00 | Navy  | 35500 | 24091.72
 7654 | a55ds67hyh7480 | Ford  | 02-NOV-13 00:00:00 | White | 85500 | 36789.13
 7788 | ajklfji948df78 | Honda | 03-DEC-14 00:00:00 | Black | 81300 | 25090.53
 7839 | j8fd988gd8s6g4 | Lexus | 01-JAN-16 00:00:00 | Gold  | 5500  | 65090.73
 7698 | ajklfji9jkfqkqk | Ford  | 07-AUG-15 00:00:00 | Red   | 55060 | 65090.83
 7782 | jda76v747afd92 | Ford  | 01-SEP-15 00:00:00 | Red   | 15500 | 35078.49
(14 rows)

edb=#
```

Figure 4.10 - The Table's Contents View.

You can also include the `ORDER BY` keywords in a `SELECT` statement to sort the data by a specified column (see Figure 4.11).

```
edb=# SELECT * FROM company_vehicle ORDER BY price;
```

```

edb=# SELECT * FROM company_vehicle ORDER BY price;
 empno |  vehicle_id  | model |  purch_date  | color | miles | price
-----+-----+-----+-----+-----+-----+-----
 7902 | ajklfjji998df78 | Kia   | 01-MAR-10 00:00:00 | Grey  | 95500 | 15090.89
 7566 | hjsak8f67xuusu | Ford  | 01-APR-14 00:00:00 | Navy  | 35500 | 24091.72
 7369 | ng889f778jkkdi | Honda | 01-JAN-14 00:00:00 | Red   | 54500 | 25090.00
 7788 | ajklfjji948df78 | Honda | 03-DEC-14 00:00:00 | Black | 81300 | 25090.53
 7876 | dfkl8908fs999s | Honda | 07-SEP-15 00:00:00 | Red   | 57900 | 25090.73
 7499 | ajkljki998df78 | Honda | 01-AUG-15 00:00:00 | Red   | 55050 | 25096.44
 7782 | jda76v747afd92 | Ford  | 01-SEP-15 00:00:00 | Red   | 15500 | 35078.49
 7521 | j8fd988gd8s6gg | Ford  | 02-MAR-15 00:00:00 | White | 55300 | 35090.49
 7900 | f7d6hjekhgfshd | Ford  | 07-DEC-15 00:00:00 | Yellow | 15500 | 35090.66
 7844 | yf7d6hjekhgfjd | Ford  | 11-AUG-13 00:00:00 | Green | 25500 | 35780.52
 7654 | a55ds67hyh7480 | Ford  | 02-NOV-13 00:00:00 | White | 85500 | 36789.13
 7934 | yf7d6h7kkhgfjd | Ford  | 02-FEB-13 00:00:00 | Sunset | 45500 | 37090.73
 7839 | j8fd988gd8s6g4 | Lexus | 01-JAN-16 00:00:00 | Gold  | 5500  | 65090.73
 7698 | ajklfjji9jkfqqk | Ford  | 07-AUG-15 00:00:00 | Red   | 55060 | 65090.83
(14 rows)

edb=#

```

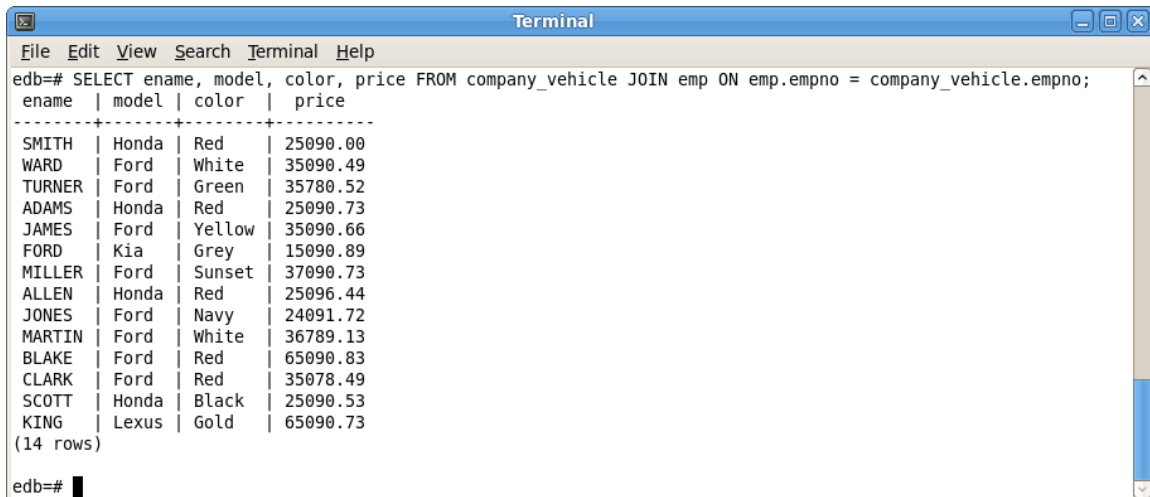
Figure 4.11 – Using an ORDER BY clause to sort data.

You can perform a simple JOIN on the tables to link the employee name with their car (see Figure 4.12).

```

edb=# SELECT ename, model, color, price FROM company_vehicle JOIN emp ON
emp.empno = company_vehicle.empno;

```



```

edb=# SELECT ename, model, color, price FROM company_vehicle JOIN emp ON emp.empno = company_vehicle.empno;
 ename | model | color | price
-----+-----+-----+-----
SMITH  | Honda | Red   | 25090.00
WARD   | Ford  | White | 35090.49
TURNER | Ford  | Green | 35780.52
ADAMS  | Honda | Red   | 25090.73
JAMES  | Ford  | Yellow | 35090.66
FORD   | Kia   | Grey  | 15090.89
MILLER | Ford  | Sunset | 37090.73
ALLEN  | Honda | Red   | 25096.44
JONES  | Ford  | Navy  | 24091.72
MARTIN | Ford  | White | 36789.13
BLAKE  | Ford  | Red   | 65090.83
CLARK  | Ford  | Red   | 35078.49
SCOTT  | Honda | Black | 25090.53
KING   | Lexus | Gold  | 65090.73
(14 rows)

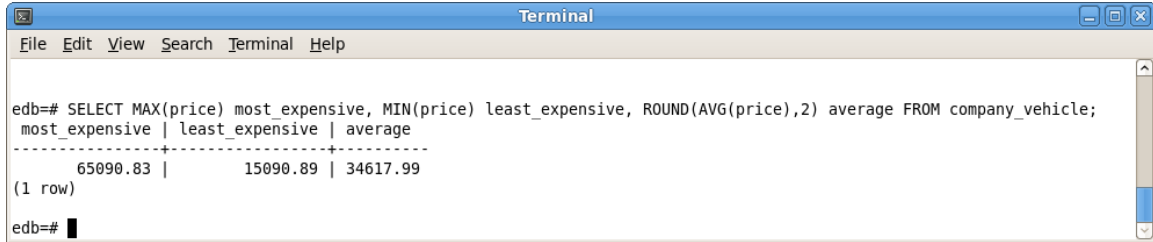
edb=#

```

Figure 4.12 – Linking Tables with JOIN.

You can also use an aggregate function to view the high value, low value, and average value of the vehicles (see Figure 4.13):

```
edb=# SELECT MAX(price) most_expensive, MIN(price) least_expensive,  
ROUND(AVG(price),2) average FROM company_vehicle;
```



The image shows a terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal displays the execution of a SQL query. The query is: `edb=# SELECT MAX(price) most_expensive, MIN(price) least_expensive, ROUND(AVG(price),2) average FROM company_vehicle;`. The output is a table with three columns: `most_expensive`, `least_expensive`, and `average`. The values are 65090.83, 15090.89, and 34617.99 respectively. Below the table, it says "(1 row)". The prompt `edb=#` is visible at the bottom left.

```
edb=# SELECT MAX(price) most_expensive, MIN(price) least_expensive, ROUND(AVG(price),2) average FROM company_vehicle;  
most_expensive | least_expensive | average  
-----  
65090.83 | 15090.89 | 34617.99  
(1 row)  
edb=#
```

Figure 4.13 – Using an aggregate function.