Getting Started with EDB Postgres™ Advanced Server on Windows®

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

March 7, 2016

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

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 Microsoft Windows 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 Windows introduces the following basics:

- identifying the database service on Windows
- 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

The Windows operating system includes a graphical service controller that offers point-and-click management of Advanced Server and Advanced Server component services. In this section, you will learn how to use the Windows `Services` Applet to discover or control the state of the Advanced Server service through a Windows-specific graphical interface.

2.1 *Opening the Windows Services Applet*

The `Services` applet icon is located under the `Administrative Tools` menu in the Windows `Apps` page. Click the `Services` icon to open the `Services` applet. (see Figure 2.1).

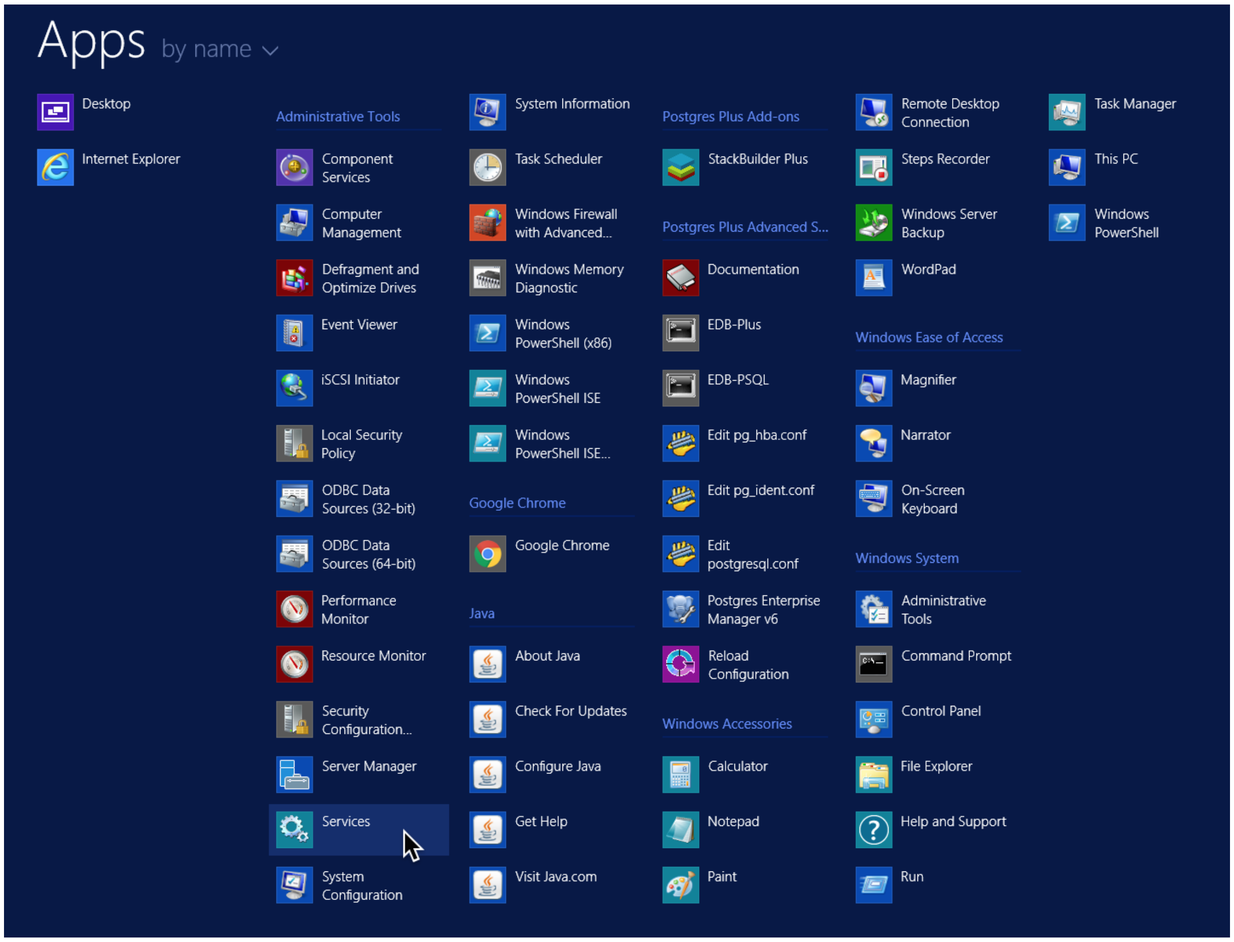
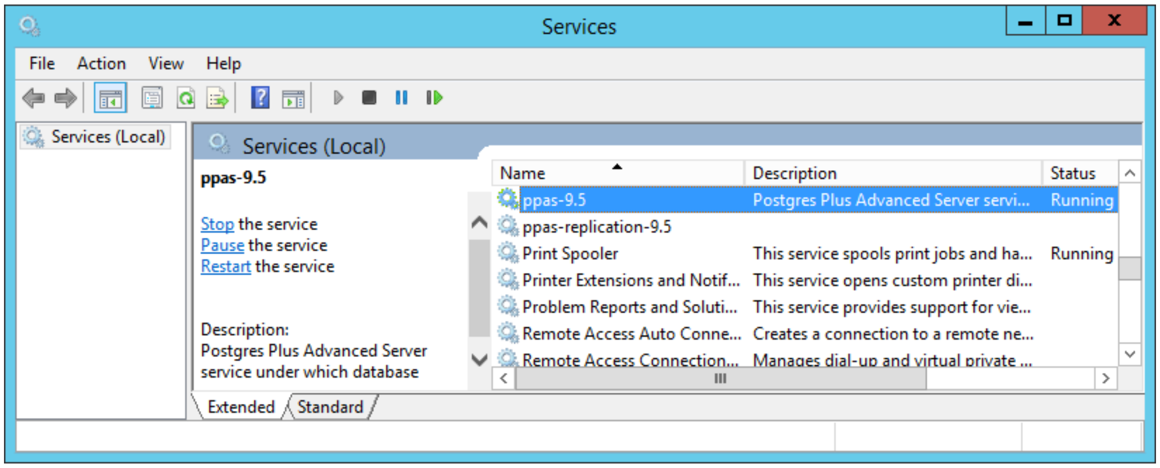


Figure 2.1 – The shortcut to Services in the Apps window.

2.2 Determining the Server Status

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. When the Windows applet opens, use the scroll bar in the Windows `Services` dialog to move through the `Name` list and click on `ppas-9.5` (see Figure 2.2).



ppas-9.5

Stop the service
Pause the service
Restart the service

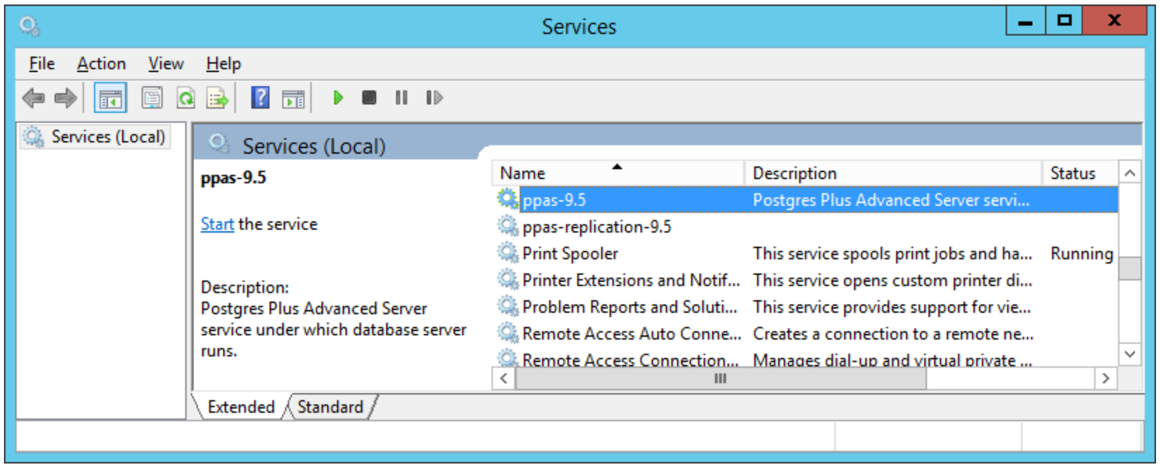
Description:
Postgres Plus Advanced Server service under which database

Name	Description	Status
ppas-9.5	Postgres Plus Advanced Server servi...	Running
ppas-replication-9.5		
Print Spooler	This service spools print jobs and ha...	Running
Printer Extensions and Notif...	This service opens custom printer di...	
Problem Reports and Soluti...	This service provides support for vie...	
Remote Access Auto Conne...	Creates a connection to a remote ne...	
Remote Access Connection...	Manages dial-up and virtual private ...	

Figure 2.2 – Selecting ppas-9.5 in the Services dialog box.

The Windows `Services` applet displays the current state of the `ppas-9.5` service in the `Status` column. The `Services` dialogue in Figure 2.2 shows that the Advanced Server service is currently `Running`.

The `Services` dialog box displays an empty `Status` column when `ppas-9.5` is stopped (see Figure 2.3).



ppas-9.5

Start the service

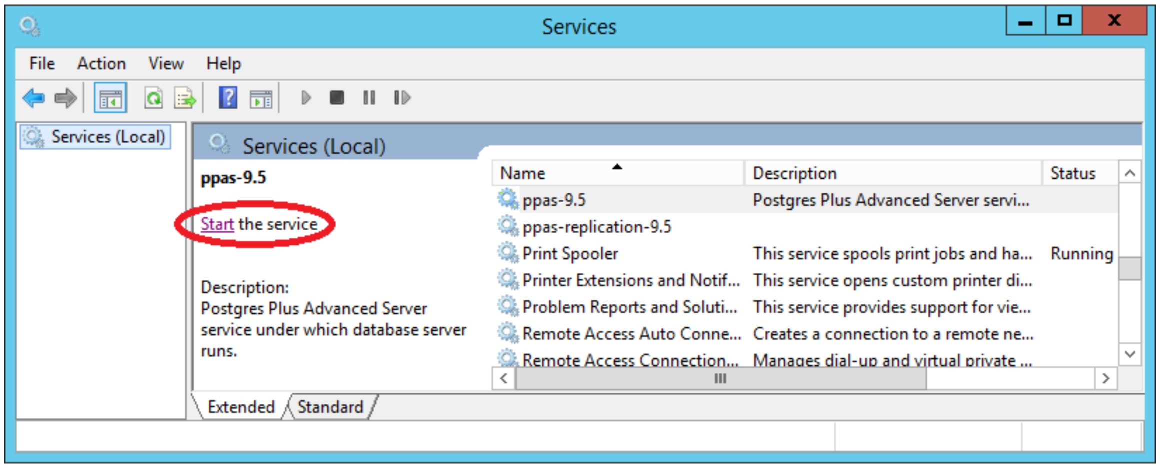
Description:
Postgres Plus Advanced Server service under which database server runs.

Name	Description	Status
ppas-9.5	Postgres Plus Advanced Server servi...	
ppas-replication-9.5		
Print Spooler	This service spools print jobs and ha...	Running
Printer Extensions and Notif...	This service opens custom printer di...	
Problem Reports and Soluti...	This service provides support for vie...	
Remote Access Auto Conne...	Creates a connection to a remote ne...	
Remote Access Connection...	Manages dial-up and virtual private ...	

Figure 2.3 – The ppas-9.5 service is stopped.

2.2.1 Starting the Service

On the upper left side of the `Services` dialog, the service controller displays commands that allow you to change the status of a server. To view these controls, the `Services` applet must be in `Extended View`. When Advanced Server is not running, `Start the service` will be the only management option available (see Figure 2.4).



ppas-9.5

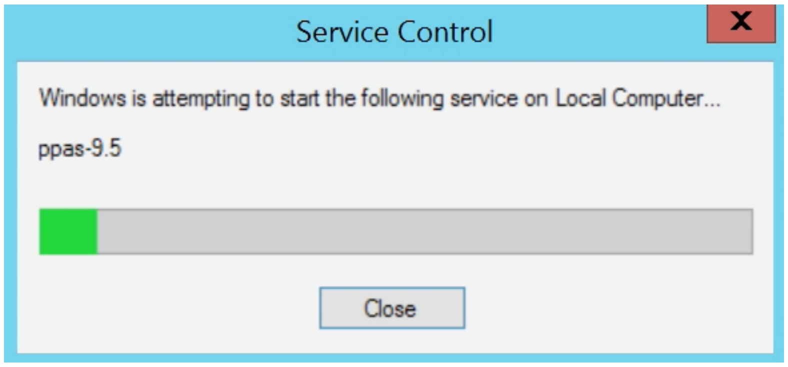
Start the service

Description:
Postgres Plus Advanced Server service under which database server runs.

Name	Description	Status
ppas-9.5	Postgres Plus Advanced Server servi...	
ppas-replication-9.5		
Print Spooler	This service spools print jobs and ha...	Running
Printer Extensions and Notif...	This service opens custom printer di...	
Problem Reports and Soluti...	This service provides support for vie...	
Remote Access Auto Conne...	Creates a connection to a remote ne...	
Remote Access Connection...	Manages dial-up and virtual private ...	

Figure 2.4 – Start the service.

Use the `Start the service` option to start the Advanced Server service. A pop-up window will confirm the action (see Figure 2.5).



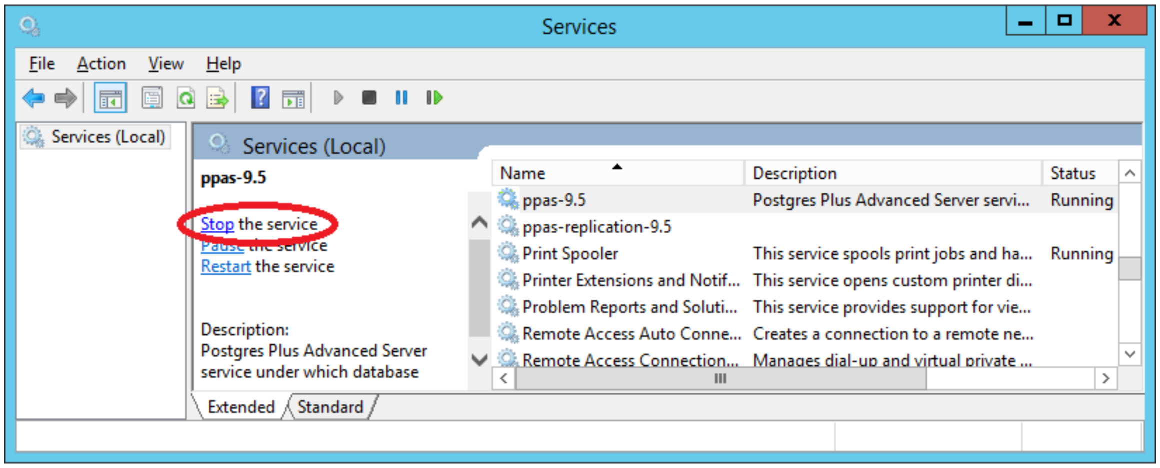
Windows is attempting to start the following service on Local Computer...

ppas-9.5

Figure 2.5 – A popup will confirm the action.

2.2.2 Stopping the Service

Select `Stop the service` (see Figure 2.6) to stop the server. When you stop the server, any user (or client application) connected to the Advanced Server instance will be disconnected.

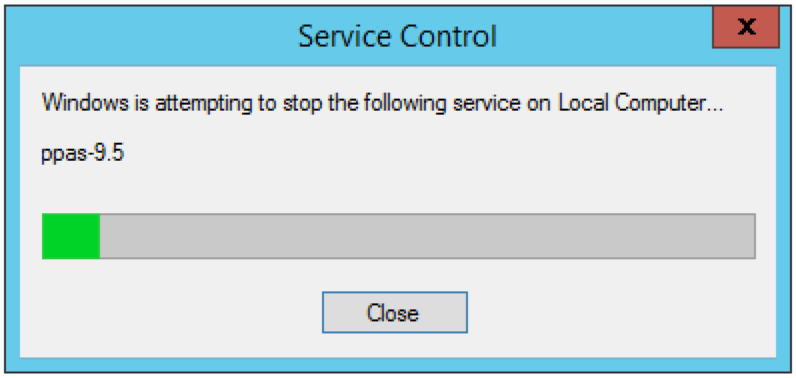


Description:
Postgres Plus Advanced Server service under which database

Name	Description	Status
ppas-9.5	Postgres Plus Advanced Server servi...	Running
ppas-replication-9.5		
Print Spooler	This service spools print jobs and ha...	Running
Printer Extensions and Notif...	This service opens custom printer di...	
Problem Reports and Soluti...	This service provides support for vie...	
Remote Access Auto Conne...	Creates a connection to a remote ne...	
Remote Access Connection...	Manages dial-up and virtual private ...	

Figure 2.6 – Stop the service.

A pop-up window will confirm the action (see Figure 2.7).



Windows is attempting to stop the following service on Local Computer...

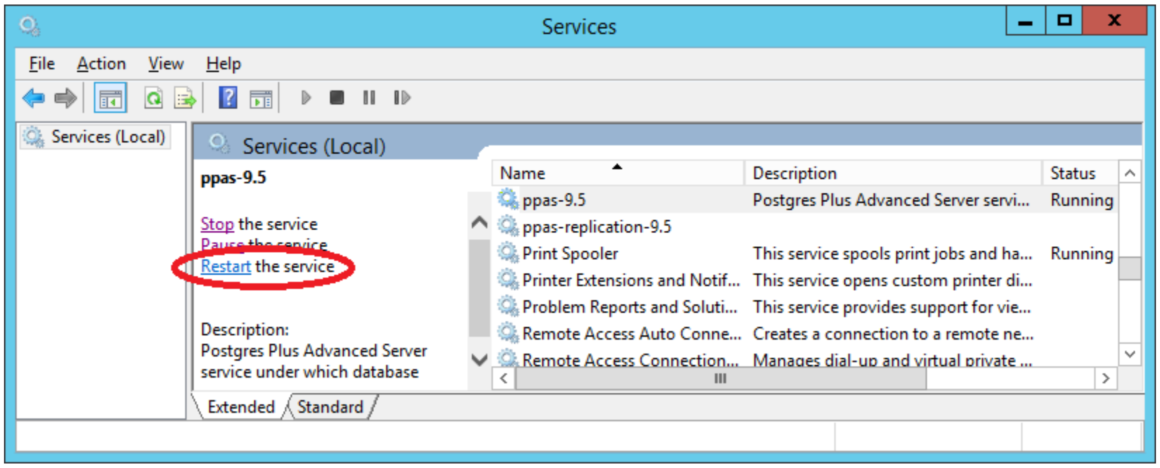
ppas-9.5

Figure 2.7 – A popup will confirm the action.

Please note: When you stop the service, any dependent services will also be stopped. Upon restarting Advanced Server, you can review the list of services displayed in the `Services` applet to confirm that your supporting services are running.

2.2.3 Restart the Service

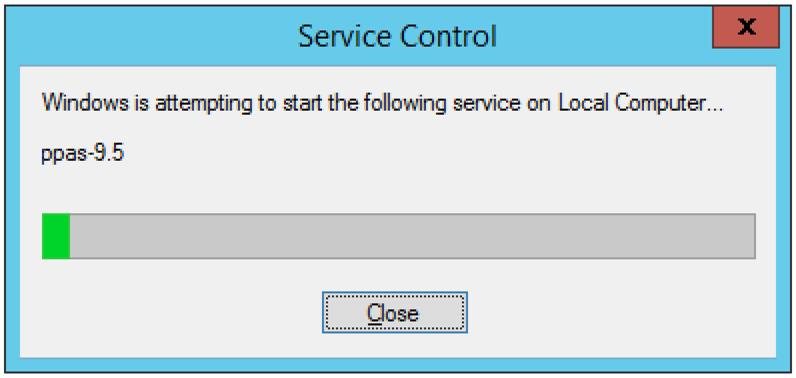
To reload configuration parameters in the Windows `Services` applet, press `Restart the service` to stop and then start the `ppas-9.5` service (see Figure 2.8). Please note that any user sessions will be terminated when you stop the service.



Name	Description	Status
ppas-9.5	Postgres Plus Advanced Server servi...	Running
ppas-replication-9.5		
Print Spooler	This service spools print jobs and ha...	Running
Printer Extensions and Notif...	This service opens custom printer di...	
Problem Reports and Soluti...	This service provides support for vie...	
Remote Access Auto Conne...	Creates a connection to a remote ne...	
Remote Access Connection...	Manages dial-up and virtual private ...	

Figure 2.8 – Restart the service.

A pop-up window will confirm the action (see Figure 2.9).



Windows is attempting to start the following service on Local Computer...

ppas-9.5

Figure 2.9 – A popup will confirm the action.

Please note: When you stop the service, any dependent services will also be stopped. Upon restarting Advanced Server, you can review the list of services displayed in the `Services` applet to confirm that your supporting services are running.

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 the Windows `Apps` menu (see Figure 3.1).

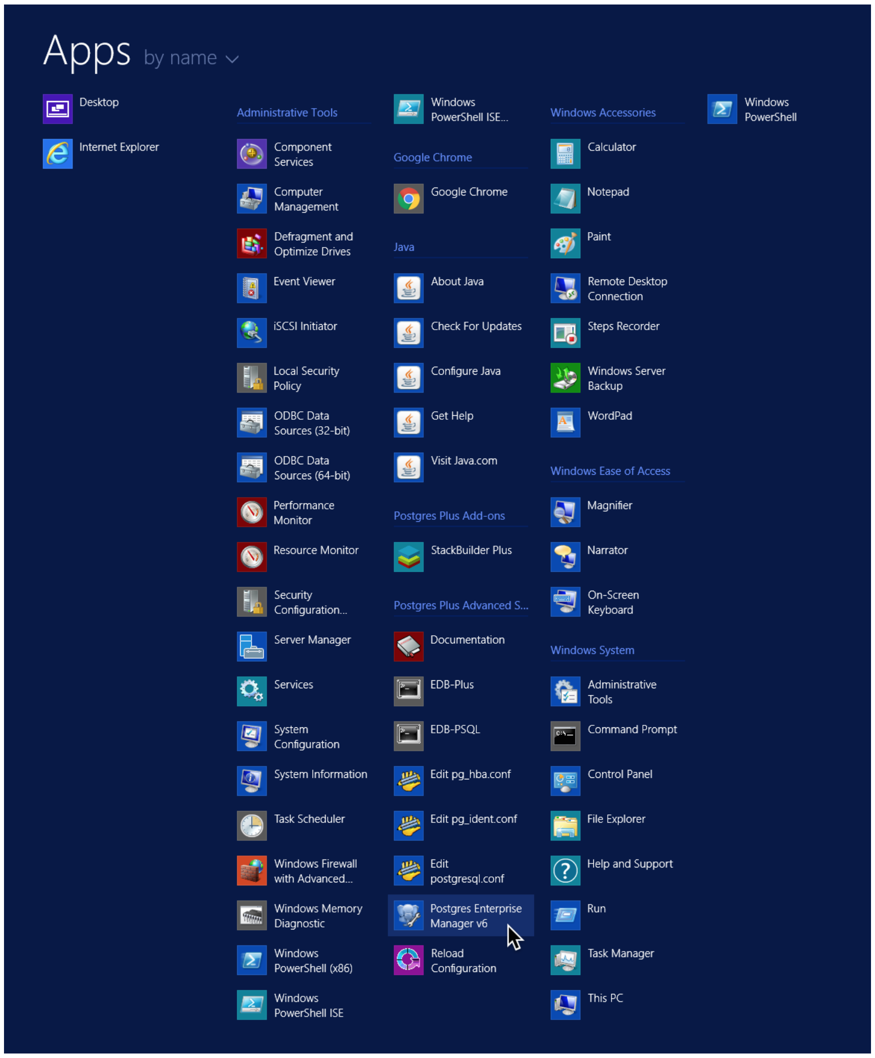


Figure 3.1 - Selecting Postgres Enterprise Manager v6.

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).

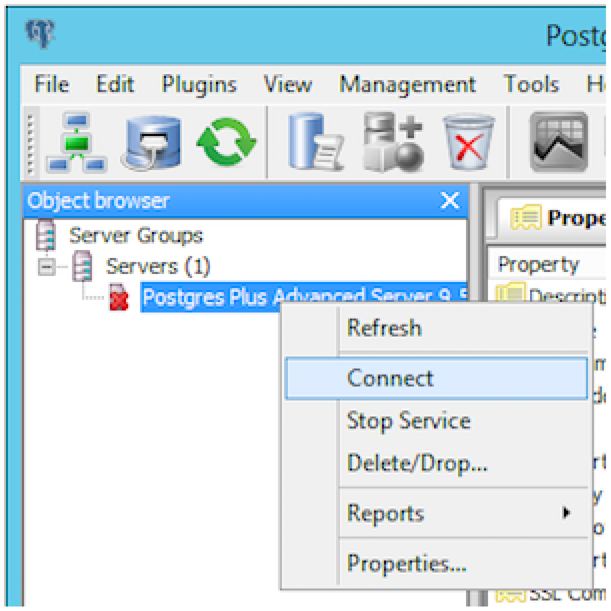
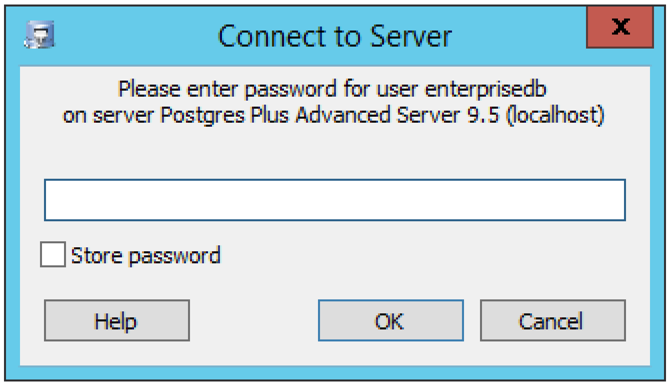


Figure 3.2 – Connecting to the PEM client.

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



Please enter password for user enterprisedb
on server Postgres Plus Advanced Server 9.5 (localhost)

Figure 3.3 – If prompted, provide a password.

3.2 The PEM Client User Interface

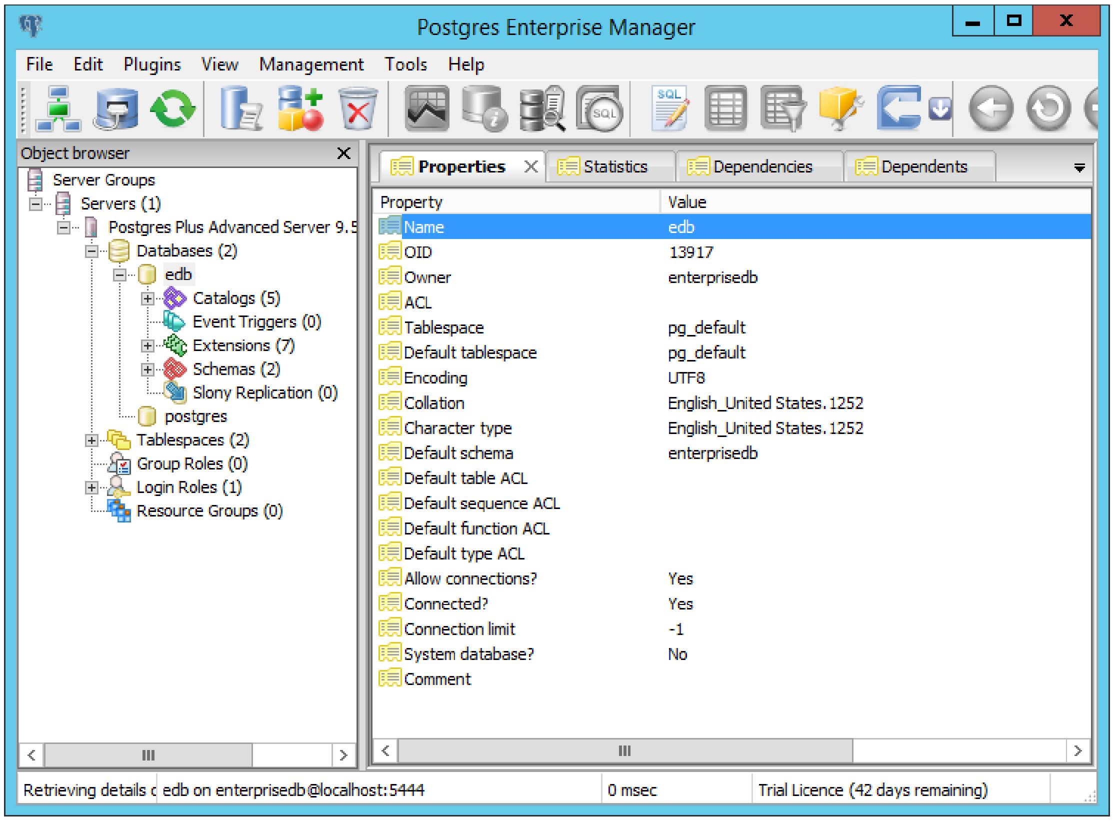
After authenticating with the server, the server's node of the tree control will be populated with the objects that reside on that server. You can expand the tree control to view the database objects that reside on each server. Use the plus sign (+) to the left of a node to expand a segment of the tree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



Property	Value
Name	edb
OID	13917
Owner	enterprisedb
ACL	
Tablespace	pg_default
Default tablespace	pg_default
Encoding	UTF8
Collation	English_United States.1252
Character type	English_United States.1252
Default schema	enterprisedb
Default table ACL	
Default sequence ACL	
Default function ACL	
Default type ACL	
Allow connections?	Yes
Connected?	Yes
Connection limit	-1
System database?	No
Comment	

Retrieving details c edb on enterprisedb@localhost:5444 | 0 msec | Trial Licence (42 days remaining)

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).

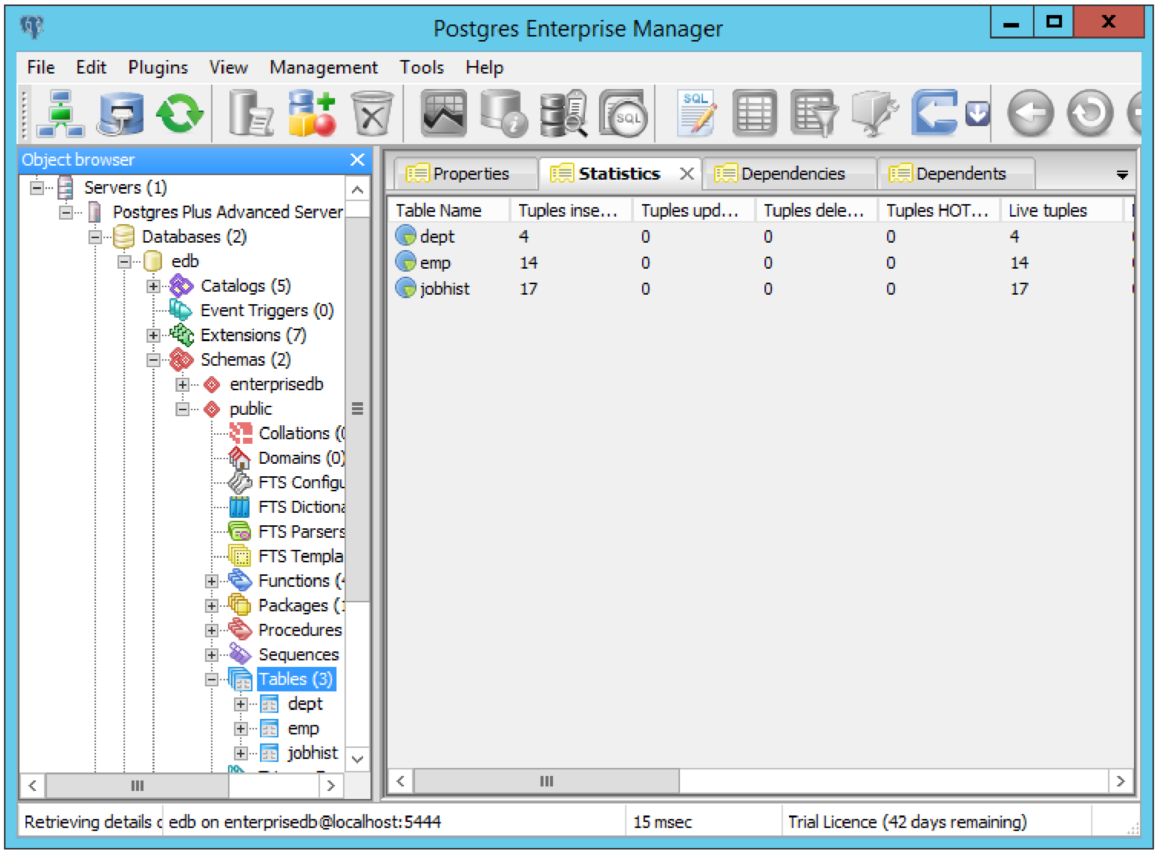


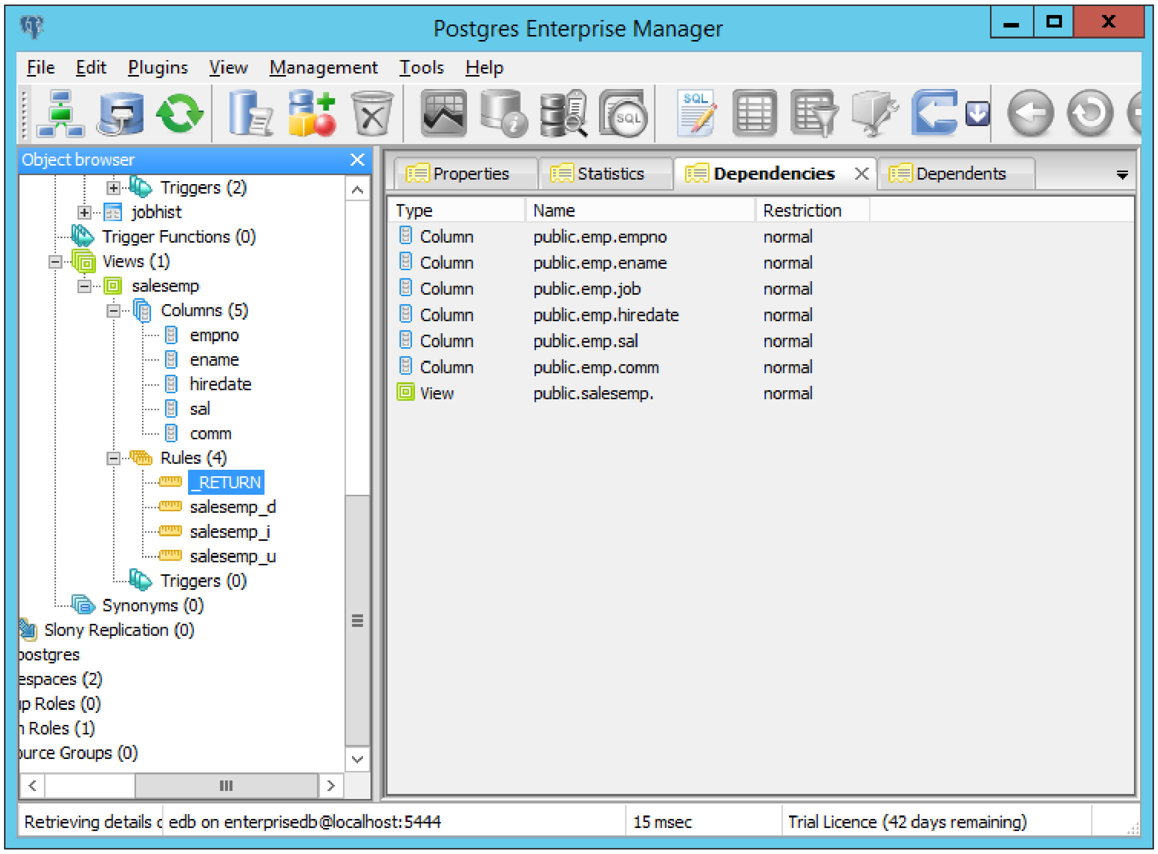
Table Name	Tuples inse...	Tuples upd...	Tuples dele...	Tuples HOT...	Live tuples
dept	4	0	0	0	4
emp	14	0	0	0	14
jobhist	17	0	0	0	17

Retrieving details c edb on enterprisedb@localhost:5444 | 15 msec | Trial Licence (42 days remaining)

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).



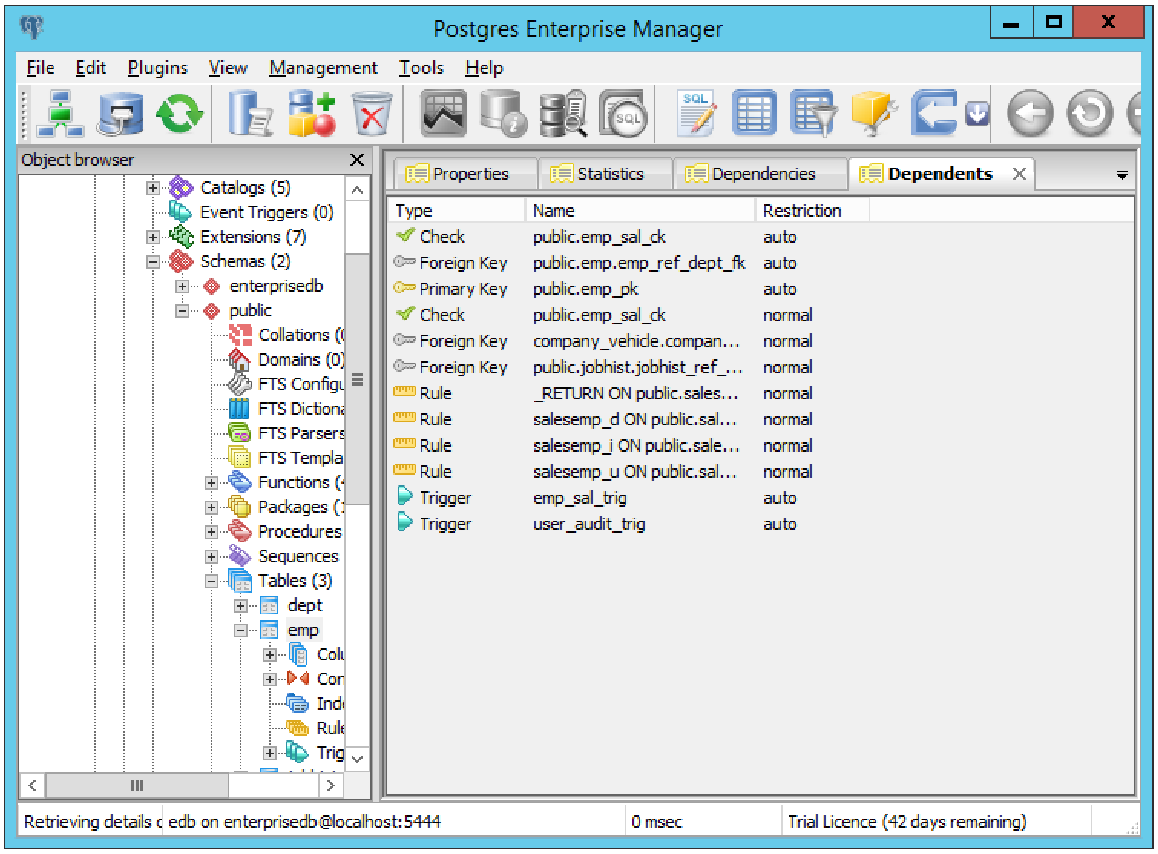
Type	Name	Restriction
Column	public.emp.empno	normal
Column	public.emp.ename	normal
Column	public.emp.job	normal
Column	public.emp.hiredate	normal
Column	public.emp.sal	normal
Column	public.emp.comm	normal
View	public.salesemp.	normal

```
Retrieving details c edb on enterprisedb@localhost:5444    15 msec    Trial Licence (42 days remaining)
```

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).



Type	Name	Restriction
Check	public.emp_sal_ck	auto
Foreign Key	public.emp.emp_ref_dept_fk	auto
Primary Key	public.emp_pk	auto
Check	public.emp_sal_ck	normal
Foreign Key	company_vehicle.compan...	normal
Foreign Key	public.jobhist.jobhist_ref_...	normal
Rule	_RETURN ON public.sales...	normal
Rule	salesemp_d ON public.sal...	normal
Rule	salesemp_i ON public.sale...	normal
Rule	salesemp_u ON public.sal...	normal
Trigger	emp_sal_trig	auto
Trigger	user_audit_trig	auto

Retrieving details c edb on enterprisedb@localhost:5444 | 0 msec | Trial Licence (42 days remaining)

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 Postgres Enterprise Manager Getting Started Guide, available at

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

3.3 Using the PEM Client

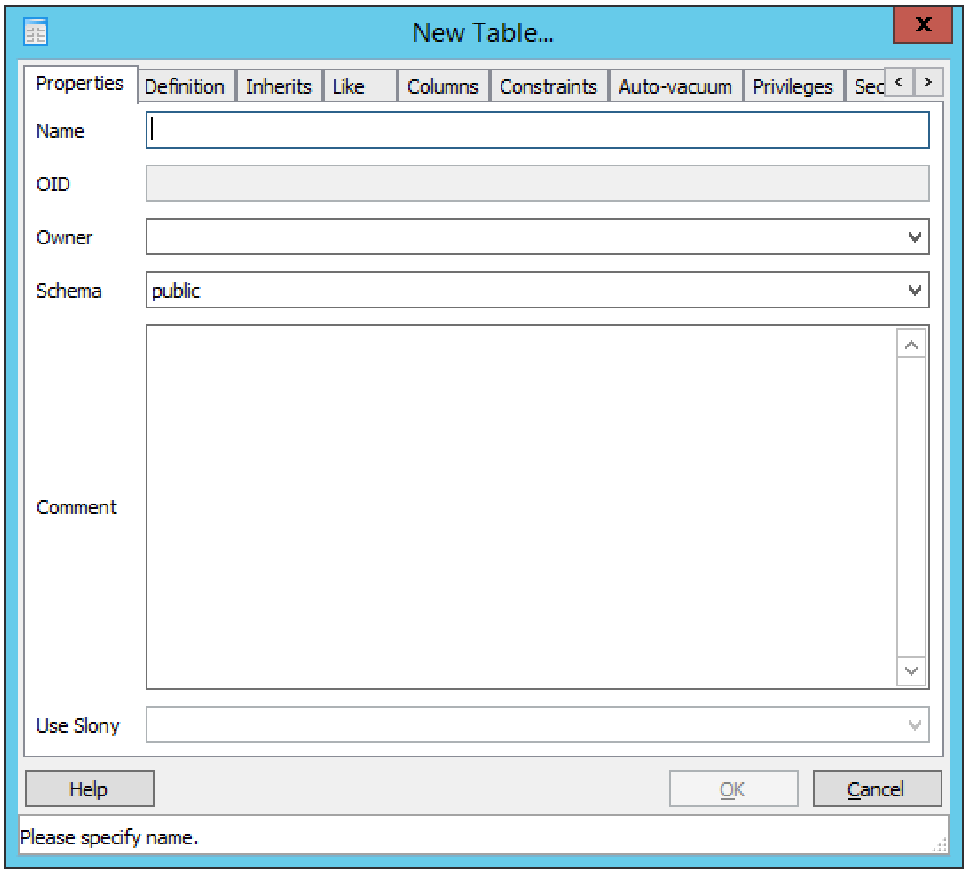
You can use the PEM client to:

- 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 attributes 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).

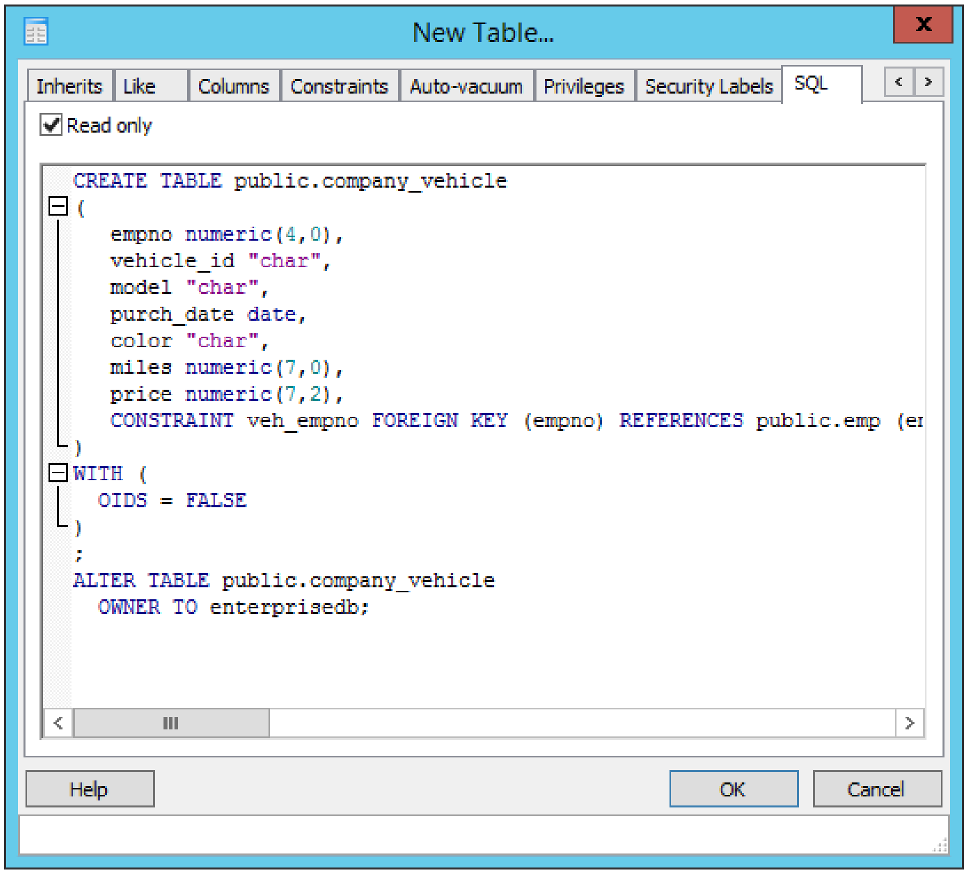


Schema: public

Please specify name.

Figure 3.8 - The New Table dialog.

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



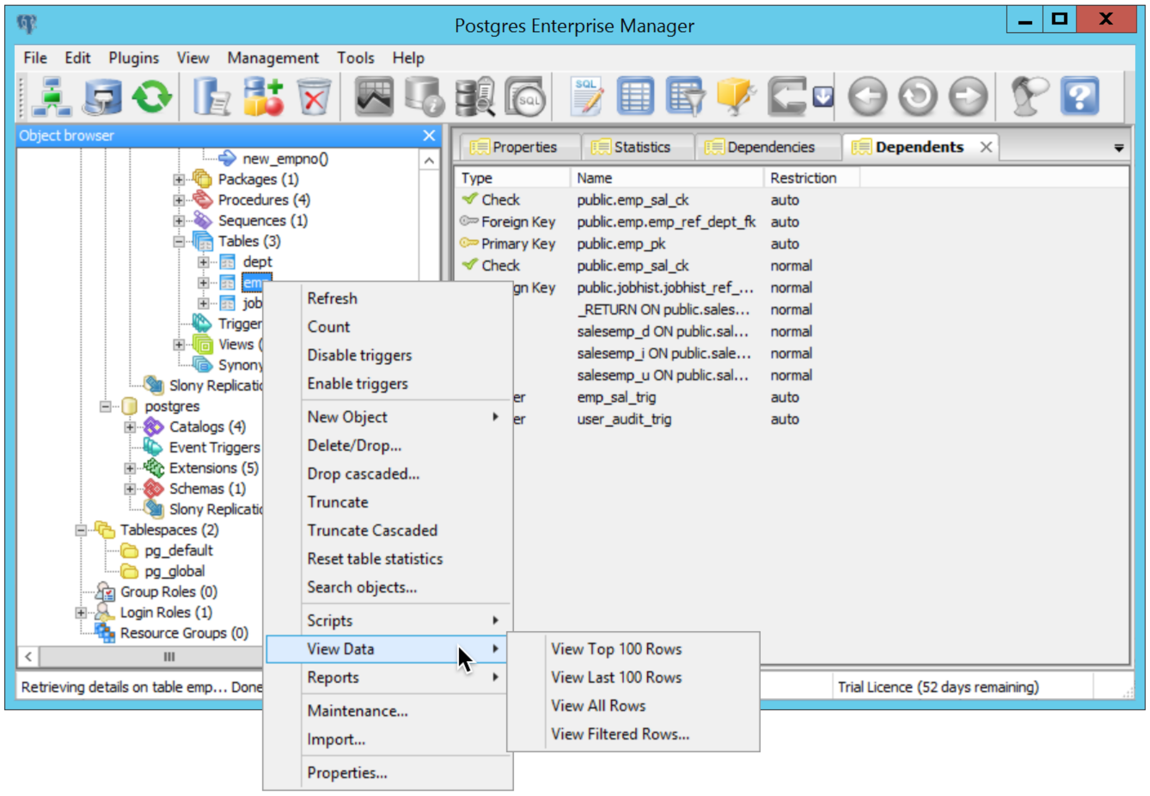
```
CREATE TABLE public.company_vehicle
(
   empno numeric(4,0),
   vehicle_id "char",
   model "char",
   purch_date date,
   color "char",
   miles numeric(7,0),
   price numeric(7,2),
   CONSTRAINT veh_empno FOREIGN KEY (empno) REFERENCES public.emp (er
)
WITH (
  OIDS = FALSE
)
;
ALTER TABLE public.company_vehicle
  OWNER TO enterprisedb;
```

Figure 3.9 - The New Table dialog.

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

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).



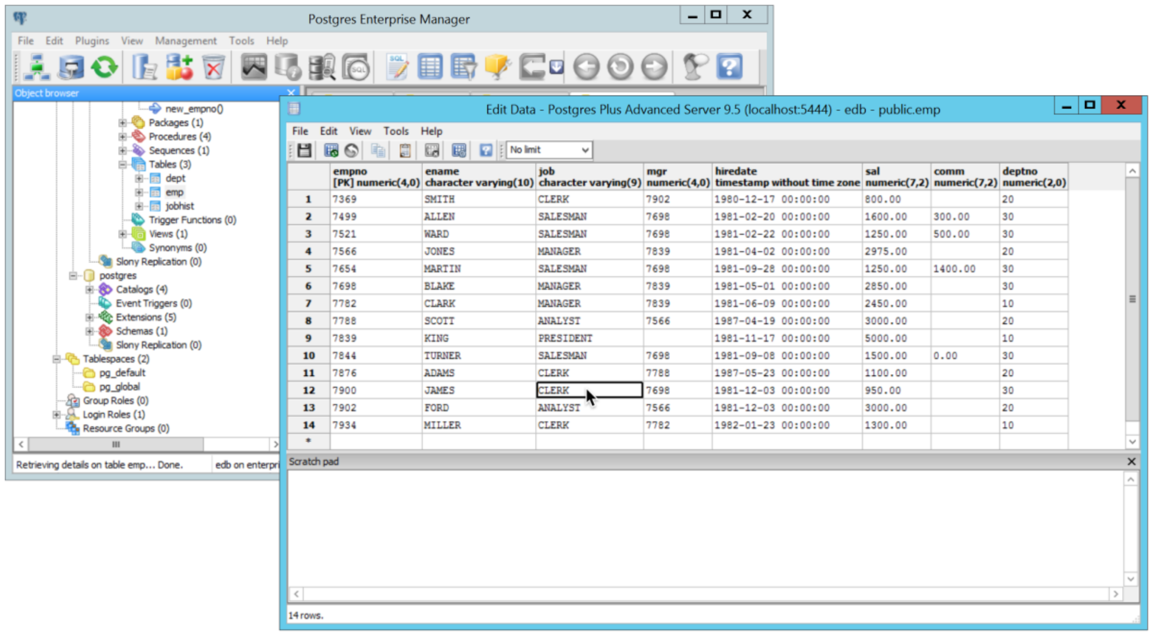
Type	Name	Restriction
Check	public.emp_sal_ck	auto
Foreign Key	public.emp.emp_ref_dept_fk	auto
Primary Key	public.emp_pk	auto
Check	public.emp_sal_ck	normal
gn Key	public.jobhist.jobhist_ref_...	normal
	_RETURN ON public.sales...	normal
	salesemp_d ON public.sal...	normal
	salesemp_i ON public.sale...	normal
	salesemp_u ON public.sal...	normal
er	emp_sal_trig	auto
er	user_audit_trig	auto

Retrieving details on table emp... Done

Trial Licence (52 days remaining)

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).


Edit Data - Postgres Plus Advanced Server 9.5 (localhost:5444) - edb - public.emp

	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	7902	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
*								

14 rows.

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).

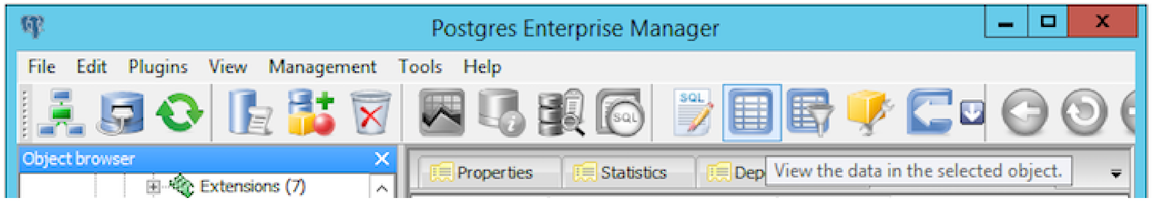


Figure 3.12 - The View Data Icon.

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.

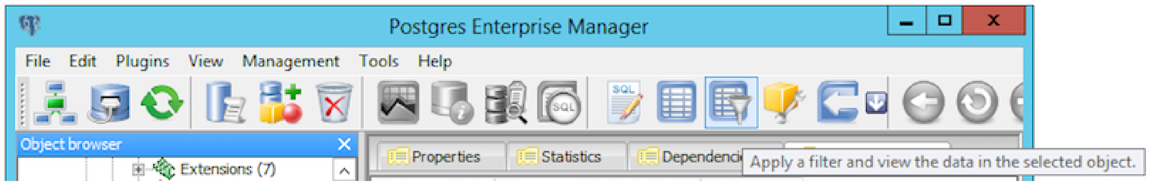
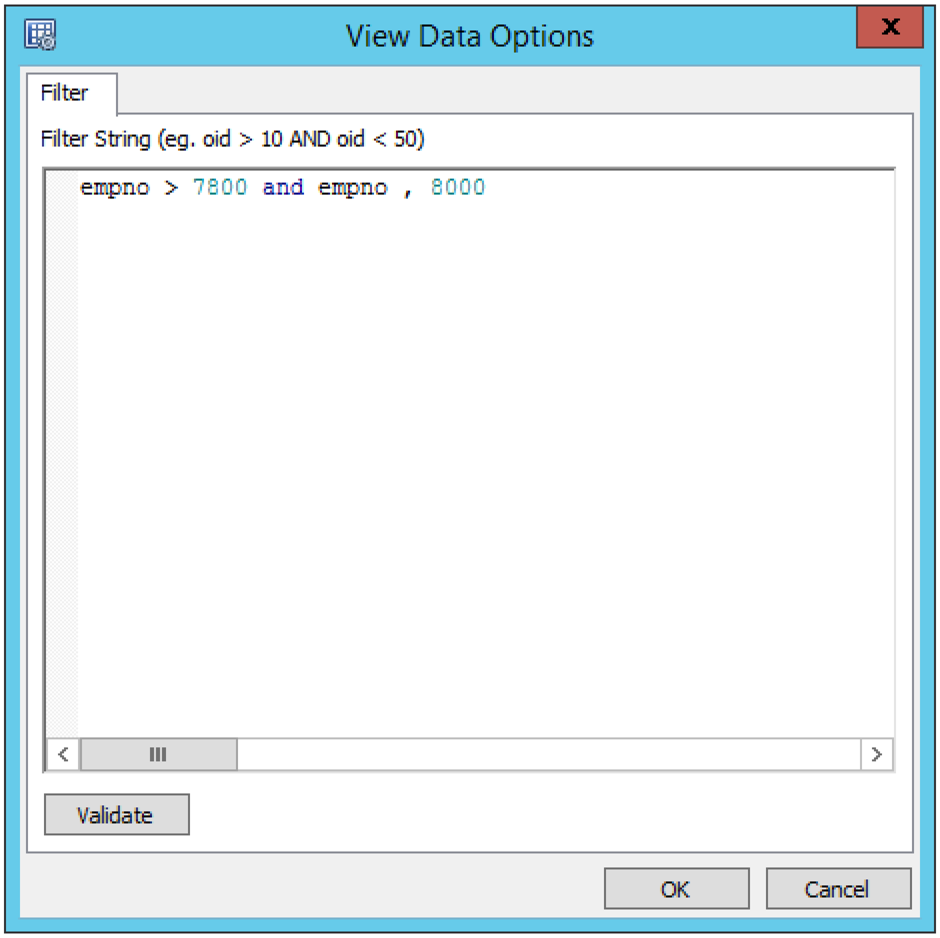

Postgres Enterprise Manager
File Edit Plugins View Management Tools Help
Object browser
Extensions (7)
Properties
Statistics
Dependenci
Apply a filter and view the data in the selected object.

Figure 3.13 - The View Filtered Data Icon.

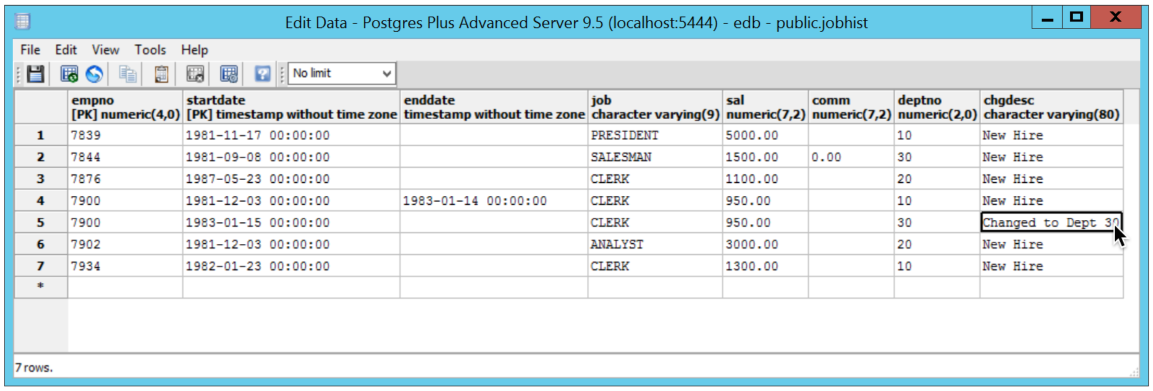
Specify a condition in the `View Data Options` dialog to filter and view data with the condition applied by entering a `Filter String` (see Figure 3.14).



```
empno > 7800 and empno , 8000
```

Figure 3.14 - The View Data Options dialog.

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


Edit Data - Postgres Plus Advanced Server 9.5 (localhost:5444) - edb - public.jobhist
File
Edit
View
Tools
Help
No limit

	empno [PK] numeric(4,0)	startdate [PK] timestamp without time zone	enddate timestamp without time zone	job character varying(9)	sal numeric(7,2)	comm numeric(7,2)	deptno numeric(2,0)	chgdesc character varying(80)
1	7839	1981-11-17 00:00:00		PRESIDENT	5000.00		10	New Hire
2	7844	1981-09-08 00:00:00		SALESMAN	1500.00	0.00	30	New Hire
3	7876	1987-05-23 00:00:00		CLERK	1100.00		20	New Hire
4	7900	1981-12-03 00:00:00	1983-01-14 00:00:00	CLERK	950.00		10	New Hire
5	7900	1983-01-15 00:00:00		CLERK	950.00		30	Changed to Dept 30
6	7902	1981-12-03 00:00:00		ANALYST	3000.00		20	New Hire
7	7934	1982-01-23 00:00:00		CLERK	1300.00		10	New Hire
*								

7 rows.

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

To open the psql client, open the `Apps` menu and select `EDB-PSQL`. The `EDB-PSQL` client icon is located in the `Postgres Plus Advanced Server` section of the `Apps` menu (see Figure 4.1).

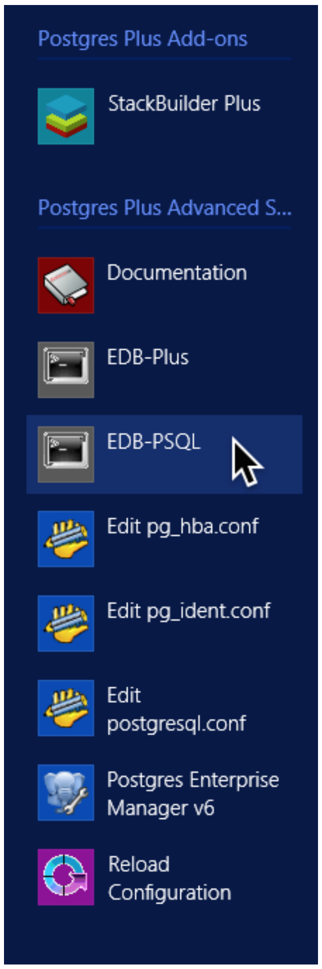
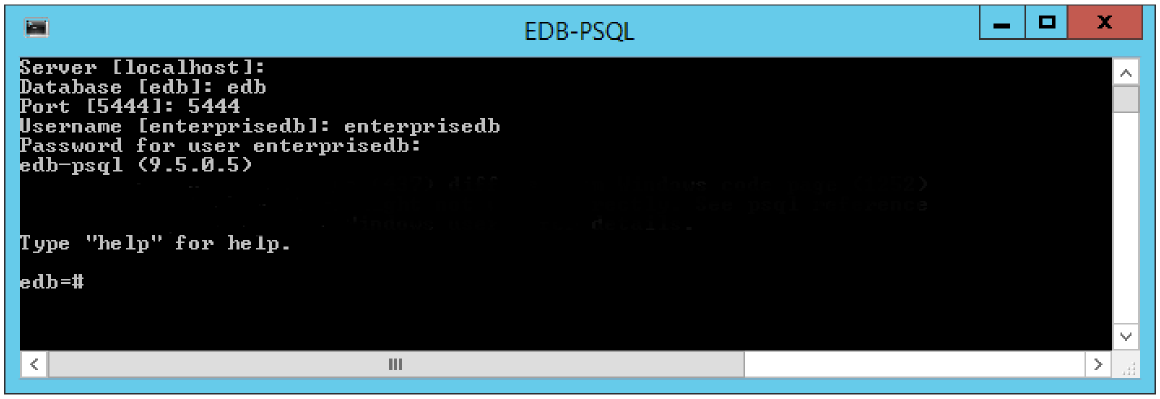


Figure 4.1 – Selecting the EDB-PSQL client icon.

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



```
Server [localhost]:
Database [edb]: edb
Port [5444]: 5444
Username [enterprisedb]: enterprisedb
Password for user enterprisedb:
edb-psql (9.5.0.5)


Type "help" for help.

edb=#
```

Figure 4.2 – The PSQL Client.

A `psql` meta-command is a command that prefaced with an unquoted backslash that (unlike SQL commands) work 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**
`\c [ dbname ]` or `\connect [ dbname [ username ] [ host ] [ port ] ] | conninfo`	Establishes a new connection to a database.
`\d table_name`	Shows the structure of the specified table.
`\d+`	Examine a table and its child tables.
`\dt`	Lists all tables in current database.
`\l`	Lists all available databases.
`\q`	Quits.
`\s`	Runs in single-step mode.
`\U username`	Connects to the database as the user username instead of the default.
`\W`	Forces psql to prompt for a password before connecting to a database.
`\x`	Display results in expanded view. This command acts as a toggle; the next \x disables the functionality.
`\? Or \h`	Display 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

4.2 Using the PSQL Client

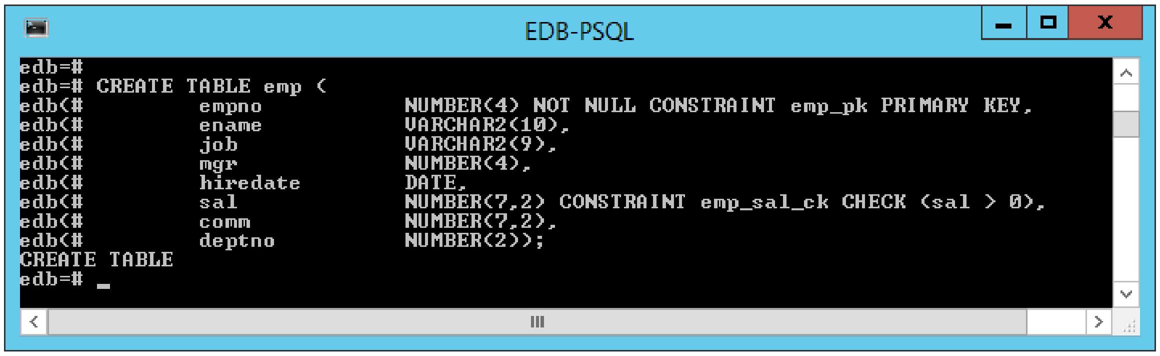
After connecting with the psql client, you can use psql meta-commands, SQL commands, and Postgres functions to create, manage, and query database objects and roles.

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 4.3).



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

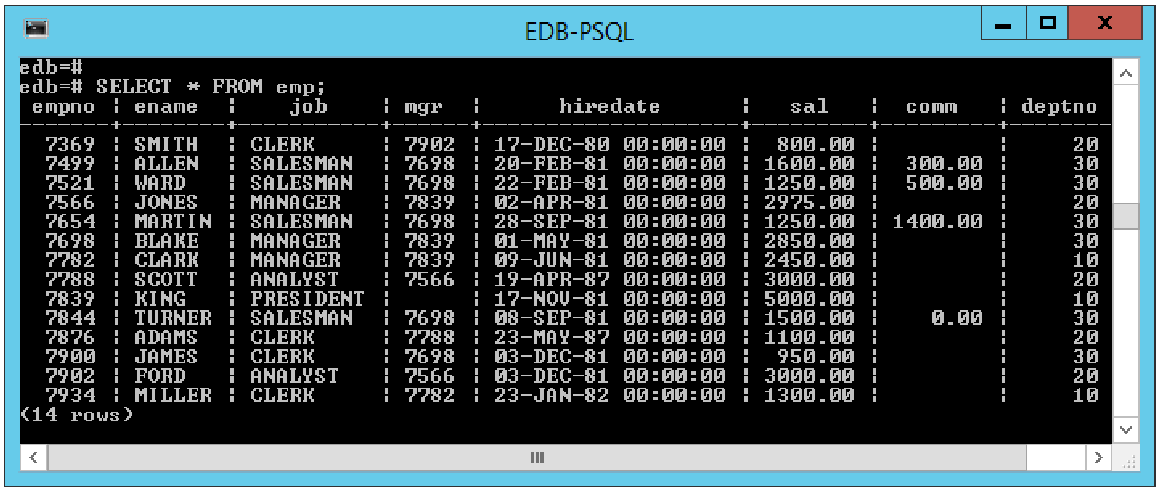
Figure 4.3 - The Create Table command.

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 4.4):



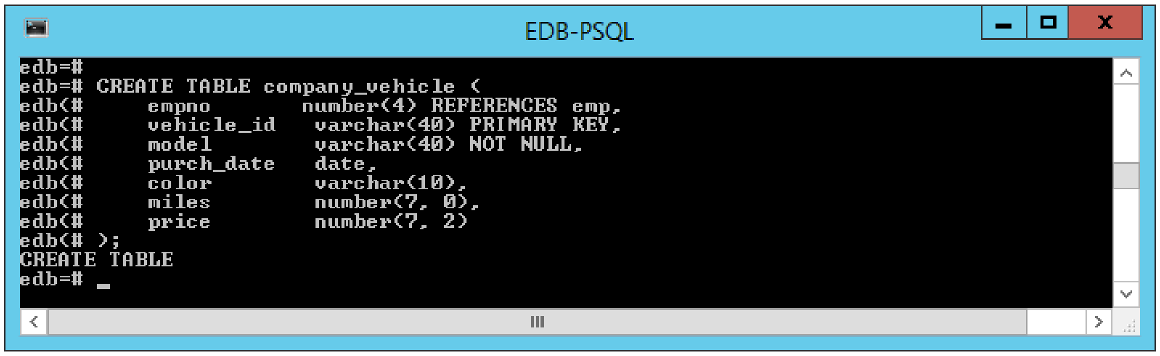
```
edb=#
edb=# SELECT * FROM emp;
 empno | ename  |    job    | mgr  |      hiredate      |   sal   |  comm   | deptno
-------+--------+-----------+------+--------------------+---------+---------+--------
  7369 | SMITH  | CLERK     | 7902 | 17-DEC-80 00:00:00 |  800.00 |         |     20
  7499 | ALLEN  | 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)
```

Figure 4.4 – 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 4.5).



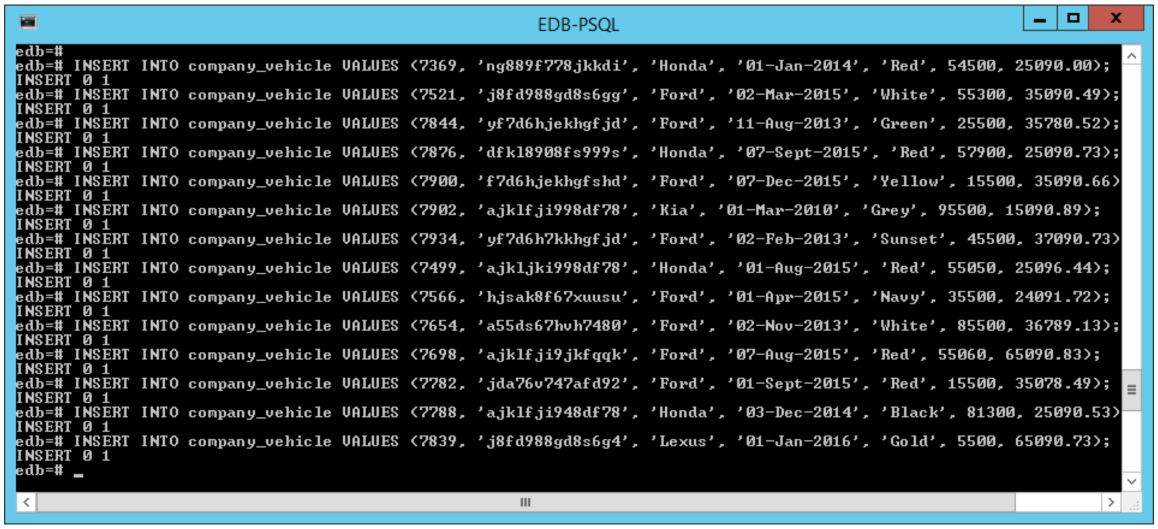
```
edb=#
edb=# CREATE TABLE company_vehicle (
edb(#     empno        number(4) REFERENCES emp,
edb(#     vehicle_id   varchar(40) PRIMARY KEY,
edb(#     model        varchar(40) NOT NULL,
edb(#     purch_date   date,
edb(#     color        varchar(10),
edb(#     miles        number(7, 0),
edb(#    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

Figure 4.5 – Creating the company_vehicle table.

Then, use `INSERT` statements to add rows 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 4.6).



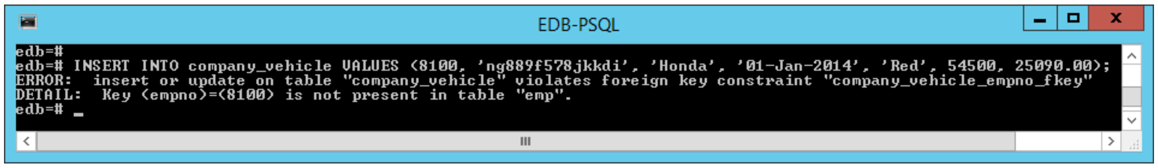
```
edb=#
edb=# INSERT INTO company_vehicle VALUES (7369, 'ng889f778jkkdi', 'Honda', '01-Jan-2014', 'Red', 54500, 25090.00);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (7521, 'j8fd988gd8s6gg', 'Ford', '02-Mar-2015', 'White', 55300, 35090.49);
INSERT 0 1
edb=# INSERT INTO company_vehicle VALUES (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, 'f7d6hjekhgfshd', '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, 'yf7d6h7kkhgfjd', '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, 'a55ds67hvh7480', '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
edb=# _
```

Figure 4.6 –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 4.7):

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

The server will return an error:



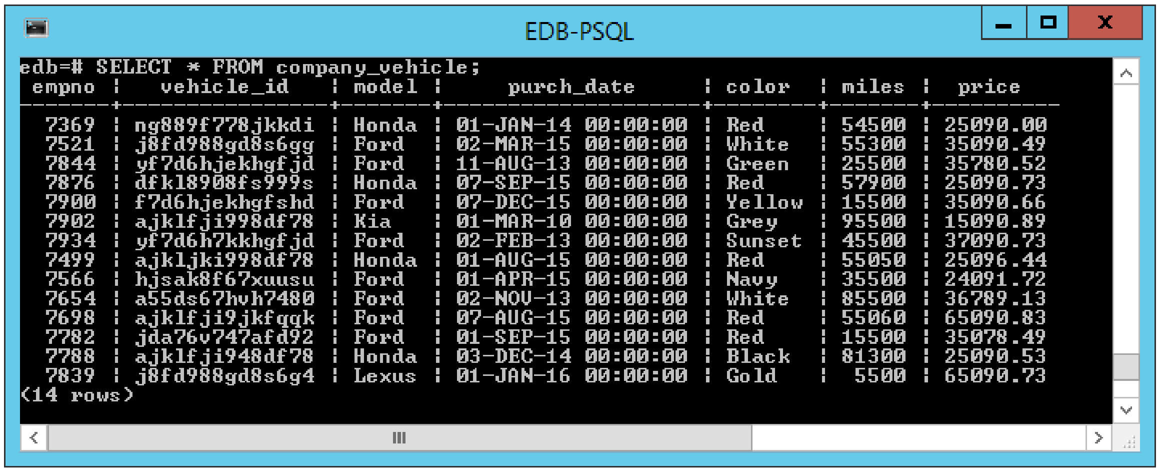
```
edb=#
edb=# INSERT INTO company_vehicle VALUES (8100, 'ng889f578jkkdi', 'Honda', '01-Jan-2014', 'Red', 54500, 25090.00);
ERROR:  insert or update on table "company_vehicle" violates foreign key constraint "company_vehicle_empno_fkey"
DETAIL:  Key (empno)=(8100) is not present in table "emp".
edb=# _
```

Figure 4.7 – 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 4.8).

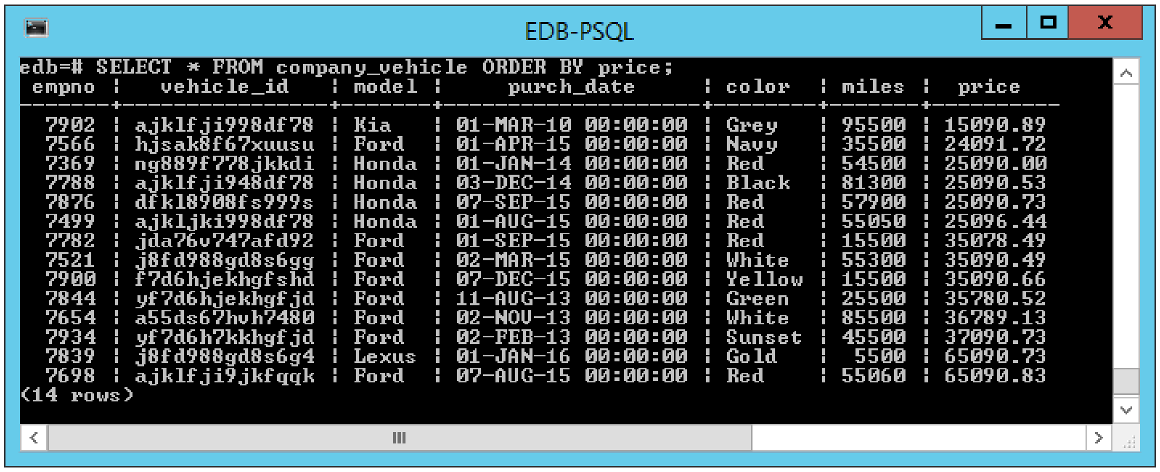


```
edb=# SELECT * FROM company_vehicle;
 empno |    vehicle_id     | model |       purch_date       | color  | miles |  price
-------+-------------------+-------+------------------------+--------+-------+----------
  7369 | ng889f778jkkdi    | Honda | 01-JAN-14 00:00:00     | Red    | 54500 | 25090.00
  7521 | j8fd988gd8s6gg    | Ford  | 02-MAR-15 00:00:00     | White  | 55300 | 35090.49
  7844 | yf7d6hjekhgfjd    | 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 | f7d6hjekhgfshd    | 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 | yf7d6h7kkhgfjd    | 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-15 00:00:00     | Navy   | 35500 | 24091.72
  7654 | a55ds67hvh7480    | Ford  | 02-NOV-13 00:00:00     | White  | 85500 | 36789.13
  7698 | ajklfji9jkfqgk    | Ford  | 07-AUG-15 00:00:00     | Red    | 55060 | 65090.83
  7782 | jda76v747afd92    | Ford  | 01-SEP-15 00:00:00     | Red    | 15500 | 35078.49
  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
(14 rows)
```

Figure 4.8 - 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 4.9).

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

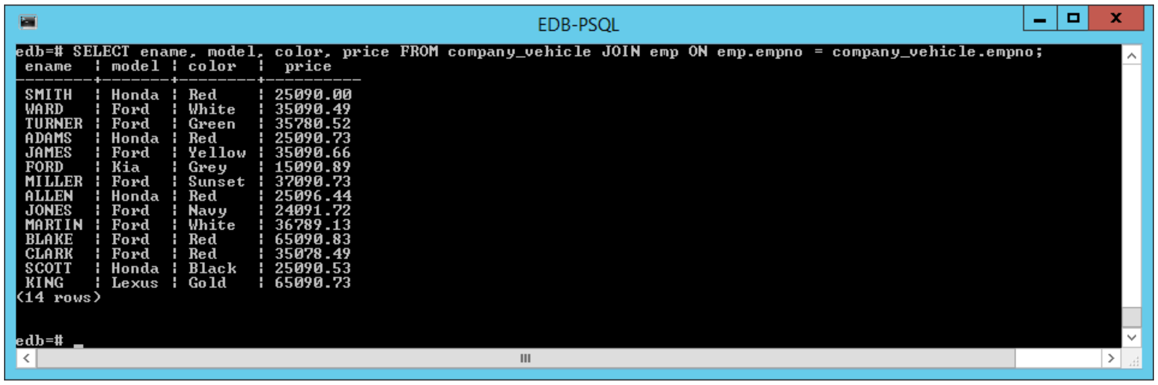


```
edb=# SELECT * FROM company_vehicle ORDER BY price;
 empno |    vehicle_id     | model |       purch_date       | color  | miles |  price
-------+-------------------+-------+------------------------+--------+-------+----------
  7902 | ajklfji998df78    | Kia   | 01-MAR-10 00:00:00     | Grey   | 95500 | 15090.89
  7566 | hjsak8f67xuusu    | Ford  | 01-APR-15 00:00:00     | Navy   | 35500 | 24091.72
  7369 | ng889f778jkkdi    | Honda | 01-JAN-14 00:00:00     | Red    | 54500 | 25090.00
  7788 | ajklfji948df78    | 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 | a55ds67hvh7480    | 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 | ajklfji9jkfqgk    | Ford  | 07-AUG-15 00:00:00     | Red    | 55060 | 65090.83
(14 rows)
```

Figure 4.9 – 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 4.10).

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



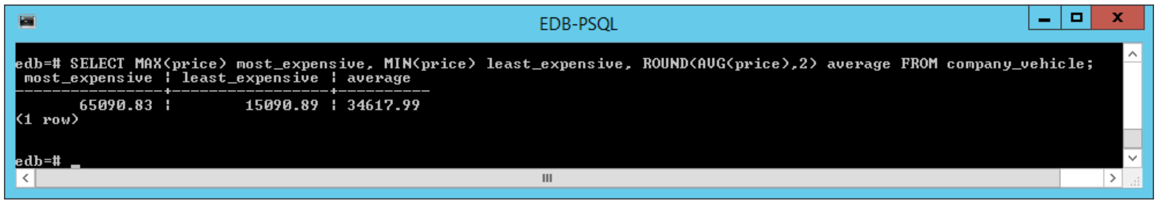
```
edb=# SELECT ename, model, color, price FROM company_vehicle JOIN emp ON emp.empno = company_vehicle.empno;
 ename  | model | color  |  price
--------+-------+--------+----------
 SMITH  | Honda | Red    | 25090.00
 WARD   | Ford  | White  | 35090.49
 TURNER | Ford  | Green  | 35780.52
 ADAMS  | 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 4.10 – Linking Tables with JOIN.

You can also use aggregate functions to view the high value, low value, and average value of the vehicles (see Figure 4.11):

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



```
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 4.11 – Using aggregate functions.