solaris

HOW to CONFIGURE and RUN
POSTGRESQL ON SOLARIS™ 10

> Solaris™ 10 How To Guides

Sun microsystems

Robert Lor, Staff Engineer
Version 1.1 | *Last updated: 07/31/07*

About This Solaris How To Guide

The PostgreSQL How to Guide instructs Solaris system administrators and database professionals in the process of configuring and running PostgreSQL on a Solaris™ 10 system. This guide covers launching and setting up the PostgreSQL database using the Solaris Service Management Facility (SMF), and configuring PostgreSQL to run in a Solaris Container technology called *Solaris Zones*. The user will then be able to verify and test the PostgreSQL data base.

This guide assumes that the release of Solaris being used is the Solaris 10 6/06 operating system or later. Starting with Solaris 10 6/06, PostgreSQL has been integrated with the operating system and can be installed with Solaris. If an earlier version of Solaris 10 (e.g. Solaris 10 1/06 or Solaris 10 3/05) is used, PostgreSQL must be installed on the system. Instructions on finding the packages and how to perform the install are included at the end of this guide.

This guide is not exhaustive and does not cover all optional features of these technologies. However, the reference section provided at the end of the document provides pointers to where administrators can learn more.

Contributors: Robert Lor, Josh Berkus, Joost Pronk, Neha Sampat, Paul Steeves

Contents

PostgreSQL How To Guide

PostgreSQL for Solaris: Overview

PostgreSQL is a very powerful, open source, enterprise-class, feature-rich relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, accuracy and portability. It also supports storage of binary large objects, including pictures, sounds, or video. It has native programming interfaces for C/C++, Java, Perl, Python, Ruby, Tcl, and ODBC, among others.

PostgreSQL is highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate. There are active PostgreSQL systems in production environments that manage in excess of 4 terabytes of data.

To meet growing demand for open source software deployments, Sun has integrated PostgreSQL for Solaris (the Solaris optimized release of PostgreSQL) in the Solaris 10 6/06 release to provide its enterprise-class customers with breakthrough new technologies based on open standards.

Starting PostgreSQL for the First Time

PostgreSQL is shipped with both Solaris Express Developer Edition (SXDE) and Solaris 10. Depending on which Solaris 10 update you're using, the instructions may be different.

Solaris Express Developer Edition

SXDE 5/07 includes PostgreSQL 8.1 and 8.2.

PostgreSQL 8.1:

1. As root, make sure the default data directory is owned by the user postgres and su to postgres:

```
# chown postgres:postgres /var/lib/pgsql/data
# su - postgres
```

2. Create PostgreSQL DB cluster:

```
$ /usr/bin/initdb -D /var/lib/pgsql/data
```

3. As root, use the SMF's svcadm command to start PostgreSQL:

```
# /usr/sbin/svcadm enable postgresql:version_81
```

PostgreSQL 8.2:

1. As root, su to postgres:

```
# su - postgres
```

2. Create PostgreSQL DB cluster:

```
$ /usr/postgres/8.2/bin/initdb -D /var/postgres/8.2/data
```

3. As root, use the SMF's svcadm command to start PostgreSQL:

```
# /usr/sbin/svcadm enable postgresql:version_82
```

The binaries for PostgreSQL 8.1 and 8.2 are located in /usr/bin and /usr/postgres/8.2/bin, respectively. To use 8.2, make sure to add /usr/postgres/8.2/bin to PATH.

It is possible to run both PostgreSQL 8.1 and 8.2 servers at the same time as long as the port numbers are different.

The data directory used above are the defaults, but you can put the database anywhere. Just make sure the SMF property for data directory is changed appropriately. See postgres_82 man page (e.g. run "man postgres_82" from the command prompt) for more information.

Solaris 10 6/06 or 11/06

Once you have successfully installed Solaris 10, there are a few steps that you will need to perform before you can start the database.

1. Create a Solaris OS user and group that will be used to administer PostgreSQL. If you choose to use an existing user, skip this step and proceed to Step 2.
    NOTE: PostgreSQL cannot be run as root user.

    For example, to create a user called "postgres" and assign it to a "postgres" group, execute the following commands as root. Make sure the directory /export/home exists:

    ```
    # groupadd postgres
    # useradd -c 'PostgreSQL user' -d /export/home/postgres -g postgres -m -s
    /bin/bash postgres
    ```

2. The next step is to decide on a directory to create the database and ensure that the permissions are set correctly. The default location is /var/lib/pgsql/data, but it can be placed anywhere. In fact, in a production environment, you should place it in its own filesystem partition, with consideration for space, growth, performance and availability.

    To use the default directory with the Solaris user called "postgres", execute the following commands to set the ownership and permissions:

    ```
    # chown postgres /var/lib/pgsql/data
    # chmod 700 /var/lib/pgsql/data
    ```

3. You are now ready to create a database cluster. Login as "postgres" or another user you've selected to run the database and execute the initdb command.

    To create a database cluster in /var/lib/pgsql/data, execute the following command:

    ```
    $ initdb -D /var/lib/pgsql/data
    ```

4. PostgreSQL is now ready to be started using the following command:

    ```
    $ pg_ctl -D /var/lib/pgsql/data -l postmaster.log start
    ```

5. You can now test the running database.

    To connect to a database called "postgres" running on a default port, execute the following command:

    ```
    $ psql postgres
    ```

To configure the database, modify the postgresql.conf file in the database cluster directory used in step 3. For tuning tips on Solaris, visit http://www.sun.com/servers/coolthreads/tnb/applications_postgresql.jsp.

PostgreSQL Integration with Solaris Service Management Facility (SMF)

PostgreSQL has been integrated with SMF in Solaris Express Developer Edition. If you are using Solaris 10 6/06 or 11/06 and want the SMF enabled PostgreSQL, you will have to do that manually with the following steps.

SMF creates a standardized control mechanism for application services by turning them into first-class objects that administrators can observe and manage in a uniform way. These services can then be automatically restarted if they are accidentally terminated by an administrator, if they are aborted as the result of a software programming error, or if they are interrupted by an underlying hardware problem. SMF is simple to use. Developers can convert most existing applications to take full advantage of SMF features just by adding a simple service manifest (XML file) to each application and using a few SMF commands to import the service description and activate the service.

Below are the SMF service manifest and accompanying shell script needed to integrate PostgreSQL with Solaris SMF. Perform the following steps to import the manifest into the SMF repository:

1. Save the following XML code to a file called "postgresql.xml" in /var/svc/manifest/application/database. You need to create the directory if it doesn't exist and have the appropriate privileges to perform this action:

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM
"/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<!--
 Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 Use is subject to license terms.

 CDDL HEADER START

 The contents of this file are subject to the terms of the
 Common Development and Distribution License (the "License").
 You may not use this file except in compliance with the License.

 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 or http://www.opensolaris.org/os/licensing.
 See the License for the specific language governing permissions
 and limitations under the License.

 When distributing Covered Code, include this CDDL HEADER in each
```

```
  file and include the License file at usr/src/OPENSOLARIS.LICENSE.

  If applicable, add the following below this CDDL HEADER, with the
  fields enclosed by brackets "[]" replaced with your own identifying
  information: Portions Copyright [yyyy] [name of copyright owner]

  CDDL HEADER END

          ident   "%Z%%M% %I%     %E SMI"

          NOTE:  This service manifest is not editable; its contents will
          be overwritten by package or patch operations, including
          operating system upgrade.  Make customizations in a different
          file.
-->

<service_bundle type='manifest' name='postgresql'>

<service
          name='application/database/postgresql'
          type='service'
          version='1'>

          <!--
              Wait for network interfaces to be initialized.
          -->
          <dependency
                  name='network'
                  grouping='require_all'
                  restart_on='none'
                  type='service'>
                  <service_fmri value='svc:/milestone/network:default' />
          </dependency>

          <!--
              Wait for all local filesystems to be mounted.
          -->
          <dependency
                  name='filesystem-local'
                  grouping='require_all'
                  restart_on='none'
                  type='service'>
                  <service_fmri value='svc:/system/filesystem/local:default'
/>
          </dependency>

          <exec_method
                  type='method'
                  name='start'
                  exec='/lib/svc/method/postgresql start'
                  timeout_seconds='300' />
```

```
        <exec_method
                type='method'
                name='stop'
                exec='/lib/svc/method/postgresql stop'
                timeout_seconds='300' />

        <exec_method
                type='method'
                name='refresh'
                exec='/lib/svc/method/postgresql refresh'
                timeout_seconds='60' />

        <!--
          We define two instances of PostgreSQL as examples.
        -->

        <instance name='default' enabled='false'>

                <method_context>
                        <method_credential user='postgres' group='postgres'
/>
                </method_context>

                <!--
                    Make sure the data configurable property points to the
                    appropriate database directory.
                -->

                <property_group name='postgresql' type='application'>
                        <propval name='data' type='astring'
                           value='/var/lib/pgsql/data' />
                        <propval name='log' type='astring'
                           value='postmaster.log' />
                </property_group>

        </instance>

        <instance name='postgres' enabled='false'>

                <method_context>
                        <method_credential user='postgres' group='postgres'
/>
                </method_context>
                <!--
                    Make sure the data configurable property points to the
                    appropriate database directory and port number in
                    postgresql.conf is different than the first instance.
                -->
<property_group name='postgresql' type='application'>
                        <propval name='data' type='astring'
                           value='/var/lib/pgsql/data2' />
                        <propval name='log' type='astring'
                           value='postmaster.log' />
                </property_group>
```

```
        </instance>

        <stability value='Evolving' />

        <template>
                <common_name>
                        <loctext xml:lang='C'>
                                PostgreSQL RDBMS
                        </loctext>
                </common_name>
                <documentation>
                        <manpage title='postgres' section='1M' />
                        <doc_link name='postgresql.org'
                                uri='http://postgresql.org' />
                </documentation>
        </template>

</service>

</service_bundle>
```

The default instance of the manifest assumes that the database user is postgres, database cluster directory is /var/lib/pgsql/data and the postmaster log file is postmaster.log. If any of them is different, update the above XML accordingly or use the svccfg command to change this property after the manifest has been imported. See the examples below.

2. Save the following shell script to a file called "postgresql":

```
#!/sbin/sh
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END

# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%      %E% SMI"
```

```
. /lib/svc/share/smf_include.sh

# SMF_FMRI is the name of the target service. This allows multiple instances
# to use the same script.

getproparg() {
        val=`svcprop -p $1 $SMF_FMRI`
        [ -n "$val" ] && echo $val
}

PGBIN=/usr/bin
PGDATA=`getproparg postgresql/data`
PGLOG=`getproparg postgresql/log`

if [ -z $SMF_FMRI ]; then
        echo "SMF framework variables are not initialized."
        exit $SMF_EXIT_ERR
fi

if [ -z $PGDATA ]; then

        echo "postgresql/data property not set"
        exit $SMF_EXIT_ERR_CONFIG
fi

if [ -z $PGLOG ]; then
        echo "postgresql/log property not set"
        exit $SMF_EXIT_ERR_CONFIG
fi

case "$1" in
'start')
        $PGBIN/pg_ctl -D $PGDATA -l $PGDATA/$PGLOG start
        ;;

'stop')
        $PGBIN/pg_ctl -D $PGDATA stop
        ;;

'refresh')
        $PGBIN/pg_ctl -D $PGDATA reload
        ;;

*)
        echo $"Usage: $0 {start|refresh}"
        exit 1
        ;;

esac
exit $SMF_EXIT_OK
```

3. Place the shell script "postgresql" in /lib/svc/method.

4. Change the permission to 555. You need to have the appropriate write privileges to copy files into this directory.

5. Import the SMF manifest by executing the following commands:

```
# cd /var/svc/manifest/application/database
# /usr/sbin/svccfg import postgresql.xml
```

6. Initially, both service instances are disabled. Use the following command to see the state.
    NOTE: The full service name or fault management resource identifier (FMRI) for both instances are
    svc:/application/database/postgresql:default and svc:/application/database/postgresql:postgres respectively, but
    they can be shortened to postgresql:default and postgresql:postgres:

```
# svcs postgresql
```

7. Start the service (e.g start PostgreSQL) for the default instance by executing the following command:

```
# /usr/sbin/svcadm enable postgresql:default
```

From this point on, the PostgreSQL process is controlled by the Solaris SMF, and the administrator can change its state
by using the svcadm command. If the service is terminated for some reason, the SMF restarter daemon will attempt
to restart it, and at system reboot, the service will be started automatically unless it is disabled.

It is possible to change any of the configurable properties (user, data, or log) in postgresql.xml dynamically after the
manifest has been imported. To do so, you need to (a) disable the service, (b) change the property value and refresh
it, and (c) restart the service. Below are two examples for changing the administrative user and data directory.

To change the Solaris administrative user to "foo" for the default instance, execute:

```
# svcadm disable postgresql:default
# svccfg -s postgresql:default setprop method_context/user = "foo"
# svcadm refresh postgresql:default
# svcadm enable postgresql:default
```

To change the database cluster directory to "/pgdata" for the default instance, execute:

```
# svcadm disable postgresql:default
# svccfg -s postgresql:default setprop postgresql/data = "/pgdata"
# svcadm refresh postgresql:default
# svcadm enable postgresql:default
```

For more details on how to use SMF, please read the smf(5) man page or online documentation at
http://docs.sun.com/app/docs/doc/817-1985/6mhm8o5rh?a=view.

Configuring and Running PostgreSQL in Solaris Zones

Solaris Zones, a part of Solaris Containers technology, is used to virtualize operating system services and provide an isolated and secure environment for running applications. A *zone* is a virtualized operating system environment created within a single instance of the Solaris Operating System. When you create a zone, you produce an application execution environment in which processes are isolated from the rest of the system. This isolation prevents processes that are running in one zone from monitoring or affecting processes that are running in other zones. Even a process running with superuser credentials cannot view or affect activity in other zones.

- **Global Zone:** Every Solaris system contains a global zone. A global zone contains a fully functional installation of the Solaris OS that is bootable by the system hardware.
- **Non-global Zone:** It is the virtualization technology that virtualizes an operating environment and isolates namespace for processes. There are two types of non-global zone root file system models: sparse and whole root. The *sparse root* zone model optimizes the sharing of objects. The *whole root* zone model provides the maximum configurability.
  - **> Sparse root Zone:** The sparse root zone model optimizes the sharing of objects but is less flexible. For example, the following directories are shared with the global zone: /lib, /platform, /sbin, /usr. Access to files in these directories will be in read-only mode.
  - **> Whole root Zone:** This model provides maximum flexibility. All file systems are private to the zone. The advantages of this model include the capability for global administrators to customize their zones file system layout. This would be done, for example, to add arbitrary unbundled or third-party packages.

For complete details on Solaris Containers technology, including Solaris Zones, refer to this online guide: http://docs.sun.com/app/docs/doc/817-1592.

Installing a Solaris Zone

There are several approaches to consider when running PostgreSQL in Solaris Zones.

- Install PostgreSQL in the global zone and run different PostgreSQL instances in different sparse root zones. The PostgreSQL binary will be shared by all zones, but not the data. This method will simplify PostgreSQL upgrade as all zones will automatically see the same binary.
- Install PostgreSQL in a whole root zone so different zones will have their own binary. In this approach, you can run different versions of PostgreSQL in different zones.
- A combination of 1 & 2. Install PostgreSQL in the global zone, create some sparse root zones to run some instances of PostgreSQL using the shared binary, and create some whole root zones to run their own copy of PostgreSQL.

The following example will demonstrate the first approach, with PostgreSQL binary installed in the global zone and using a sparse root zone to run the process.

Before creating a Solaris Zone, decide on a directory where it will reside. In this example, the zone will be installed in /export/zones/pg_zone. Make sure to limit the access of this directory to only the user with read, write, and execute permission (e.g chmod 700 /export/zones/pg_zone).

Creating, Installing and Booting a Zone

NOTE: For more detailed, step-by-step instructions on configuring zones, visit the Solaris Containers how to guide at sun.com/solaris/howtoguides.

1. To configure and define a new zone, use the following command:

```
global# zonecfg -z pg_zone
```

This will return the message "pg_zone: No such zone configured" before prompting you to begin configuring a new

zone. You are now in the zonecfg shell that is identified by its prompt:"zonecfg:email-zone>".

2. Configuring the zone by executing the following commands:

```
zonecfg:pg_zone> create
zonecfg:pg_zone> set zonepath=/export/zones/pg_zone
zonecfg:pg_zone> set autoboot=true
zonecfg:pg_zone> add net
zonecfg:pg_zone:net> set address=10.6.222.74/24
zonecfg:pg_zone:net> set physical=ipge0
zonecfg:pg_zone:net> end
zonecfg:pg_zone> verify
zonecfg:pg_zone> commit
zonecfg:pg_zone> exit
```

NOTE: Change "address" and "physical" to the appropriate IP address and name of interface card, respectively.

At this point, a zone configuration file is created in /etc/zones/pg_zone.xml

3. Install the zone using the following command:

```
global1# zoneadm -z pg_zone install
```

This can take a few minutes.

4. When installation completes, use the following command to list the status of the installed zones:

```
global1# zoneadm list -iv
```

5. Booting a zone places the zone in the running state. A zone can be booted from the installed state or from the ready state using the following command:

```
global1# zoneadm -z pg_zone boot
```

6. Use the following command to log in to the Zone console:

```
global1# zlogin -C pg_zone
```

The first time you log in to the console, you are prompted to answer a series of questions.

At this point, the newly created zone is ready to use. You can proceed to setup PostgreSQL in the zone.

Run PostgreSQL in Solaris Zone

Follow the instructions in the section "Starting PostgreSQL for the First Time" to configure and run PostgreSQL.

Mount filesystem in Zone

1. To mount a filesystem in a non-global zone, add the following entries to the zone configuration (pg_zone.xml):

```
global# zonecfg -z pg_zone
zonecfg:pg_zone> add fs
zonecfg:pg_zone:fs> set type=ufs
zonecfg:pg_zone:fs> set special=/dev/dsk/c1t1d0s0
zonecfg:pg_zone:fs> set raw=/dev/rdsk/c1t1d0s0
zonecfg:pg_zone:fs> set dir=/pg_log
zonecfg:pg_zone:fs> end
zonecfg:pg_zone> verify
zonecfg:pg_zone> commit
zonecfg:pg_zone> exit
```

Change the properties "special", "raw", and "dir" appropriately for your environment.

Installing PostgreSQL on Earlier Versions of Solaris 10

This section only applies when using Solaris 10 3/05 or 1/06 releases.

Obtaining the Packages

The packages can be downloaded from http://pgfoundry.org/projects/solarispackages/.

PostgreSQL for Solaris Packages

The table below lists all the packages and what they are used for. For a complete list of files in each package, see the pkgmap file in each package.

Package	Description
SUNWpostgr-libs	The SUNWpostgr-libs package provides the essential shared libraries for any PostgreSQL client program or interface. You will need to install this package to use any other PostgreSQL package or any clients that need to connect to a PostgreSQL server.
SUNWpostgr	If you want to manipulate a PostgreSQL database on a local or remote PostgreSQL server, you need this package. You also need to install this package if you're installing the SUNWpostgr-server package.
SUNWpostgr-contrib	The SUNWpostgr-contrib package contains contributed packages that are included in the PostgreSQL distribution.
SUNWpostgr-devel	The SUNWpostgr-devel package contains the header files and libraries needed to compile C or C++ applications which will directly interact with a PostgreSQL database management server and the ecpg Embedded C Postgres preprocessor. You need to install this package if you want to develop applications which will interact with a PostgreSQL server.
SUNWpostgr-docs	The SUNWpostgr-docs package includes the SGML source for the documentation as well as the documentation in PDF format and some extra documentation. Install this package if you want to help with the PostgreSQL documentation project, or if you want to generate printed documentation.

Package	Description
SUNWpostgr-server	The SUNWpostgr-server package includes the programs needed to create and run a PostgreSQL server, which will in turn allow you to create and maintain PostgreSQL databases. You should install SUNWpostgr-server if you want to create and maintain your own PostgreSQL databases and/or your own PostgreSQL server. You also need to install the SUNWpostgr package and its requirements.
SUNWpostgr-server-data	The SUNWpostgr-server-data package creates the default data directories and may contain a demo database.
SUNWpostgr-tcl	The SUNWpostgr-tcl package contains the Pgtcl client library and its documentation.
SUNWpostgr-jdbc	The SUNWpostgr-jdbc package includes the .jar files needed for Java programs to access a PostgreSQL database.
SUNWpostgr-pl	The SUNWpostgr-pl package contains the PL/Perl, and PL/Python procedural languages for the backend. PL/Pgsql is part of the core server package.

Table 1—PostgreSQL for Solaris packages

Package File Locations

To remain in compliance with the Solaris OS, the PostgreSQL for Solaris packages install files in various locations which are different than the default locations found in PostgreSQL documentation. According to the PostgreSQL documentation, PostgreSQL is installed under the directory /usr/local/pgsql, with executables, source, and data existing in various subdirectories.

Different distributions have different recommended file locations. In particular, the documentation directory can be /usr/doc, /usr/doc/packages, /usr/share/doc, /usr/share/doc/packages, or some other similar path. The Solaris locations are listed below:

Files	PostgreSQL 8.1	PostgreSQL 8.2
Executables	/usr/bin	/usr/postgres/8.2/bin
Libraries	/usr/lib	/usr/postgres/8.2/lib
Documentation	/usr/share/doc/pgsql-x.y.z /usr/share/doc/pgsql-x.y.z/contrib	/usr/postgres/8.2/doc
Contrib	/usr/share/pgsql/contrib	/usr/postgres/8.2/share/contrib
Data	/var/lib/pgsql/data	/var/postgres/8.2/backup
Backup Area	/var/lib/pgsql/backup	/var/postgres/8.2/backup
Templates	/usr/share/pgsql	/usr/postgres/8.2/share

Files	PostgreSQL 8.1	PostgreSQL 8.2
Procedural Languages	/usr/lib/pgsql	/usr/postgres/8.2/lib
Development Headers	/usr/include/pgsql	/usr/postgres/8.2/include
Other shared data	/usr/share/pgsql	/usr/postgres/8.2/share

Table 2—Package file locations

Installing PostgreSQL for Solaris Packages

This section only applies when using Solaris 10 3/05 or Solaris 10 1/06 releases.

To quickly get PostgreSQL up and running, you can install a subset of the packages available. See the table above for further information. Here are a couple of scenarios:

• If you only want to run a Postgres server, install SUNWpostgr-libs, SUNWpostgr, SUNWpostgr-server-data, and SUNWpostgr-server
• If you only want to run the Postgres client, install SUNWpostgr-libs and SUNWpostgr

If a package depends on other package(s), you will need to install these dependencies first. You will be notified of these dependencies during install.

Solaris packages are installed using the pkgadd command. This command transfers the contents of a software package from the distribution medium or directory and installs it onto a system.

This section provides basic installation instructions for installing your package in order to verify that it installs correctly.

1. Download packages, unzip and untar them.
2. Add the software package to the system:

```
# pkgadd -d device-name [pkg-abbrev...]
```

device-name specifies the location of the package. Note that device-name can be a full directory path name or the identifiers for a tape, floppy disk, or removable disk.

pkg-abbrev Is the name of one or more packages (separated by spaces) to be added. If omitted, pkgadd installs all available packages.

For example, the following command will install SUNWpostgr-libs package from the current directory:

```
# pkgadd -d . SUNWpostgr-libs
```

After you have installed all the necessary packages, refer to the "Starting PostgreSQL for the First Time" section above to run the database.

Installing Solaris Patches

If you're installing PostgreSQL on Solaris 10 3/05 or 1/06, you will need to install Python patch 121606-01 before using PL/Python procedural language. The Python patch can be downloaded from: http://pgfoundry.org/projects/solarispackages.

Refer to the man pages for instructions on using 'patchadd' and 'patchrm' scripts provided with Solaris. To install the patch, follow these steps:

1. Download the patch, unzip and untar it into any directory (e.g. /var/tmp).

2.   Add the patch to the system. You must have root privileges to add a patch:

```
# patchadd /var/tmp/121606-01
```

The above command will take a few minutes, so be patient.
After the patch is installed successfully, you can proceed to use PL/Python.

NOTE: If you encounter patchadd or patchrm problems, such as "wordlist too large" messages while installing this
patch, you may need to install the following patch:

```
119254-02 (or newer) Install and Patch Utilities Patch
```

Removing PostgreSQL for Solaris Packages

You shouldn't need to remove PostgreSQL packages, but if you choose to, it is recommended that you execute a full
database dump (and possibly a filesystem level backup) before removing Solaris packages. Because the pkgrm command
updates information in the software products database, it is important when you remove a package to use the pkgrm
command, even though you might be tempted to use the rm command instead. For example, you could use the rm
command to remove a binary executable file, but that is not the same as using pkgrm to remove the software package that
includes that binary executable. Using the rm command to remove a package's files will corrupt the software products
database. (If you really only want to remove one file, you can use the removef command, which will update the software
product database correctly.)

1.   Log in to the system as superuser.
2.   Remove an installed package:

```
# pkgrm pkg-list ...
```

pkg-list Is the name of one or more packages (separated by spaces). If omitted, pkgrm removes all available packages.

3.   To verify that the package has successfully been removed, use the pkginfo command:

```
# pkginfo | egrep pkg-abbrev
```

If pkg-abbrev is installed, the pkginfo command returns a line of information about it. Otherwise, pkginfo returns the
system prompt.

You should stop all server processes before removing packages.
NOTE: If you have created database clusters in /var/lib/pgsql/data directory, any newly created files and directories
will not be removed by a pkgrm of SUNWpostgr-server-data package. If you want to remove the database content, you
have to do it manually.

Summary

Fully integrated into Solaris 10 with flexible support offerings from Sun, PostgreSQL on Solaris 10 is an enterprise-class
open source database. When combined with the reliable, stable Solaris Operating System, customers can use PostgreSQL
for a majority of the commercial database needs. Customers now have the additional reassurance of world-class, global
24x7 support from Sun.

For More information

While this How to Guide provides a user with the basic steps required to get started with a PostgreSQL database on Solaris 10, more information on varying configurations, additional Solaris 10 How to Guides and other relevant information for PostgreSQL for Solaris are referenced below.

Solaris 10 Manuals and Reference Materials	
Solaris 10 Overview	sun.com/solaris/
Solaris 10 FAQ	sun.com/solaris/faqs/index.jsp
Solaris 10 Datasheets and Resources	sun.com/solaris/teachme
Additional Solaris How to Guides	sun.com/solaris/howtoguides
Predictive Self-Healing Feature Information	sun.com/solaris/availability

PostgreSQL for Solaris Reference Materials	
PostgreSQL for Solaris Service & Support Offering	sun.com/service/osdb/index.xml
PostgreSQL for Solaris Web Site	sun.com/solaris/postgresql

Community Resources	
PostgreSQL Documentation	postgresql.org/docs/
PostgreSQL for Solaris Documentation	postgresql.org/docs/techdocs.33
PostgreSQL Packages for Solaris	pgfoundry.org/projects/solarispackages/

sun.com/solaris