PostgreSQL Administrator’s Guide

The PostgreSQL Development Team

PostgreSQL Administrator's Guide
Covering v6.3 for general release
by The PostgreSQL Development Team

Edited by Thomas Lockhart

(last updated 1998-02-23)

Table of Contents

List of Tables

List of Figures

Summary

Postgres, developed originally in the UC Berkeley Computer Science Department, pioneered many of the object-relational concepts now becoming available in some commercial databases. It provides SQL92/SQL3 language support, transaction integrity, and type extensibility. PostgreSQL is a public-domain, open source descendant of this original Berkeley code.

Chapter 1. Introduction

This document is the user manual for the PostgreSQL database management system, originally developed at the University of California at Berkeley. PostgreSQL is based on Postgres release 4.2. The Postgres project, led by Professor Michael Stonebraker, has been sponsored by the Defense Advanced Research Projects Agency (DARPA), the Army Research Office (ARO), the National Science Foundation (NSF), and ESL, Inc.

What is Postgres?

Traditional relational database management systems (DBMSs) support a data model consisting of a collection of named relations, containing attributes of a specific type. In current commercial systems, possible types include floating point numbers, integers, character strings, money, and dates. It is commonly recognized that this model is inadequate for future data processing applications. The relational model successfully replaced previous models in part because of its "Spartan simplicity". However, as mentioned, this simplicity often makes the implementation of certain applications very difficult. Postgres offers substantial additional power by incorporating the following four additional basic concepts in such a way that users can easily extend the system:

classes
inheritance
types
functions

Other features provide additional power and flexibility:

constraints
triggers
rules
transaction integrity

These features put Postgres into the category of databases referred to as object-relational. Note that this is distinct from those referred to as object-oriented, which in general are not as well suited to supporting the traditional relational database languages. So, although Postgres has some object-oriented features, it is firmly in the relational database world. In fact, some commercial databases have recently incorporated features pioneered by Postgres.

A Short History of Postgres

The Berkeley Postgres Project

Implementation of the Postgres DBMS began in 1986. The initial concepts for the system were presented in [STON86] and the definition of the initial data model appeared in [ROWE87]. The design of the rule system at that time was described in [STON87a]. The rationale and architecture of the storage manager were detailed in [STON87b].

Postgres has undergone several major releases since then. The first "demoware" system became operational in 1987 and was shown at the 1988 ACM-SIGMOD Conference. We released Version 1, described in [STON90a], to a few external users in June 1989. In response to a critique of the first rule system ([STON89]), the rule system was redesigned ([STON90b]) and Version 2 was released in June 1990 with the new rule system. Version 3 appeared in 1991 and added support for multiple storage managers, an improved query executor, and a rewritten rewrite rule system. For the most part, releases since then have focused on portability and reliability.

Postgres has been used to implement many different research and production applications. These include: a financial data analysis system, a jet engine performance monitoring package, an asteroid tracking database, a medical information database, and several geographic information systems. Postgres has also been used as an educational tool at several universities. Finally, Illustra Information Technologies picked up the code and commercialized it. Postgres became the primary data manager for the Sequoia 2000 scientific computing project in late 1992. Furthermore, the size of the external user community nearly doubled during 1993. It became increasingly obvious that maintenance of the prototype code and support was taking up large amounts of time that should have been devoted to database research. In an effort to reduce this support burden, the project officially ended with Version 4.2.

Postgres95

In 1994, Andrew Yu and Jolly Chen added a SQL language interpreter to Postgres, and the code was subsequently released to the Web to find its own way in the world. Postgres95 was a public-domain, open source descendant of this original Berkeley code.

Postgres95 is a derivative of the last official release of Postgres (version 4.2). The code is now completely ANSI C and the code size has been trimmed by 25%. There are a lot of internal changes that improve performance and code maintainability. Postgres95 v1.0.x runs about 30-50% faster on the Wisconsin Benchmark compared to v4.2. Apart from bug fixes, these are the major enhancements:

- The query language Postquel has been replaced with SQL (implemented in the server). We do not yet support subqueries (which can be imitated with user defined SQL functions). Aggregates have been re-implemented. We also added support for ‘‘GROUP BY’’. The libpq interface is still available for C programs.
- In addition to the monitor program, we provide a new program (psql) which supports GNU readline.
- We added a new front-end library, libpgtcl, that supports Tcl-based clients. A sample shell, pgtclsh, provides new Tcl commands to interface tcl programs with the Postgres95 backend.
- The large object interface has been overhauled. We kept Inversion large objects as the only mechanism for storing large objects. (This is not to be confused with the Inversion file system which has been removed.)
- The instance-level rule system has been removed. Rules are still available as rewrite rules.
- A short tutorial introducing regular SQL features as well as those of ours is distributed with the source code.
- GNU make (instead of BSD make) is used for the build. Also, Postgres95 can be compiled with an unpatched gcc (data alignment of doubles has been fixed).

PostgreSQL

By 1996, it became clear that the name Postgres95 would not stand the test of time. A new name, PostgreSQL, was chosen to reflect the relationship between original Postgres and the more recent versions with SQL capability. At the same time, the version numbering was reset to start at 6.0, putting the numbers back into the sequence originally begun by the Postgres Project.

The emphasis on development for the v1.0.x releases of Postgres95 was on stabilizing the backend code. With the v6.x series of PostgreSQL, the emphasis has shifted from identifying and understanding existing problems in the backend to augmenting features and capabilities, although work continues in all areas.

Major enhancements include:

- Important backend features, including subselects, defaults, constraints, and triggers, have been implemented.
- Additional SQL92-compliant language features have been added, including primary keys, quoted identifiers, literal string type coersion, type casting, and binary and hexadecimal integer input.
- Built-in types have been improved, including new wide-range date/time types and additional geometric type support.
- Overall backend code speed has been increased by approximately 20%, and backend startup speed has decreased 80%.

About This Release

From now on, We will use Postgres to mean PostgreSQL.

PostgreSQL is available without cost. This manual describes version 6.3 of PostgreSQL.

Check the Administrator's Guide for a list of currently supported machines. In general, PostgreSQL is portable to any Unix/Posix-compatible system with full libc library support.

Resources

This manual set is organized into several parts:

Tutorial

: An introduction for new users. Does not cover advanced features.

User's Guide

: General information for users, including available commands and data types.

Programmer's Guide

: Advanced information for application programmers. Topics include type and function extensibility, library interfaces, and application design issues.

Administrator's Guide

: Installation and management information. List of supported machines.

Developer's Guide

: Information for Postgres developers. This is intended for those who are contributing to the Postgres project; application development information should appear in the Programmer's Guide.

Reference Manual

: Detailed reference information on command syntax. At the moment, this manual is very sparse, but eventually should contain information similar to that in the man pages.

In addition to this manual set, there are other resources to help you with Postgres installation and use:

man pages

The man pages have general information on command syntax.

FAQs

The Frequently Asked Questions (FAQ) documents address both general issues and some platform-specific issues.

READMEs

README files are available for some contributed packages.

Web Site

The Postgres web site has some information not appearing in the distribution. There is a mhonarc catalog of mailing list traffic which is a rich resource for many topics.

Mailing Lists

The Postgres Questions mailing list is a good place to have user questions answered. Other mailing lists are available; consult the web page for details.

Yourself!

Postgres is an open source product. As such, it depends on the user community for ongoing support. As you begin to use Postgres, you will rely on others for help, either through the documentation or through the mailing lists. Consider contributing your knowledge back. If you learn something which is not in the documentation, write it up and contribute it. If you add features to the code, contribute it. Even those without a lot of experience can provide corrections and minor changes in the documentation, and that is a good way to start. The Postgres Documentation mailing list is the place to get going.

Copyrights and Trademarks

Chapter 2. Ports

Currently Supported Platforms

Postgres is available free of charge. This manual describes version 6.3 of Postgres. The authors have compiled and tested Postgres on the following platforms:

Table 2-1. Supported Platforms

OS	Processor	Version	Reported	Remarks
AIX 4.1.x-4.2	RS6000	v6.3	1998-03-01	4.1.4.0,4.2 (Darren King), 4.1.5 (Andreas Zeugswetter); 3.2.5 confirmed on v6.2.1 (Frank Dana)
BSDi	x86	v6.3	1998-03-01	(Bruce Momjian
FreeBSD 2.2.x-3.x	x86	v6.3	1998-03-01	(Tatsuo Ishii, Marc Fournier)
NetBSD 1.3	x86	v6.3	1998-03-01	(Brook Milligan)
NetBSD 1.3	Sparc	v6.3	1998-03-01	(Tom I Helbekkmo)
NetBSD 1.3	VAX	v6.3	1998-03-01	(Tom I Helbekkmo)
DGUX 5.4R4.11	m88k	v6.3	1998-03-01	(Brian E Gallew)
HPUX 10.20	PA-RISC	v6.3	1998-03-01	9.0.x confirmed on v6.2.1 (Stan Brown)
IRIX 6.x	MIPS	v6.3	1998-03-01	5.x is different (Andrew Martin)
Digital 4.0	Alpha	v6.3	1998-03-01	in progress; v6.2.1 confirmed working (Pedro J. Lobo)
linux 2.0.x	Alpha	v6.3	1998-03-01	partial success (Ryan Kirkpatrick, Jeff Sturm)
linux 2.0.x	x86	v6.3	1998-03-01	(Thomas Lockhart, Tatsuo Ishii)
linux 2.0.x	Sparc	v6.3	1998-03-01	(Tom Szybist)
mklinux	PPC	v6.3	1998-03-01	(Tatsuo Ishii)
SCO	x86	v6.3	1998-03-01	partial success (Billy G. Allie)
Solaris	x86	v6.3	1998-03-01	(Marc Fournier)
Solaris 2.5.1-2.6	x86	v6.3	1998-03-01	(Marc Fournier, Tatsuo Ishii)
SunOS 4.1.4	Sparc	v6.3	1998-03-01	patches submitted (Tatsuo Ishii)
SVR4	MIPS	v6.3	1998-03-01	similar to v6.2.1; "mostly working" (Frank Ridderbusch)

SVR4 4.4	m88k	v6.2.1	1998-03-01	confirmed with patching (Doug Winterburn)
Unixware	x86	v6.3	1998-03-01	aka UNIVEL (Billy G. Allie)
NextStep	x86	v6.x	1998-03-01	client-only support; v1.0.9 worked with patches (David Wetzel)

Unsupported Platforms

A few platforms which have been attempted and which have been reported to not work with the standard distribution. Others listed here do not provide sufficient library support for an attempt.

Table 2-2. Possibly Incompatible Platforms

OS	Processor	Version	Reported	Remarks
MacOS	all	v6.3	1998-03-01	not library compatible; use ODBC/JDBC
NetBSD	arm32	v6.3	1998-03-01	not yet working (Dave Millen)
NetBSD	m68k	v6.3	1998-03-01	Amiga, HP300, Mac; not yet working (Henry Hotz)
Ultrix	MIPS,VAX?	v6.x	1998-03-01	no recent reports; obsolete?
Windows NT	all	v6.3	1998-03-01	not library compatible; client side maybe; use ODBC/JDBC
Windows	x86	v6.3	1998-03-01	not library compatible; client side maybe; use ODBC/JDBC

Note that Windows ports of the frontend are apparently possible using third-party Posix porting tools and libraries.

Chapter 3. Installation

Complete installation instructions for Postgres v6.3.

This procedure is This is based on the installation instructions for Postgres v6.3 found in $PGROOT/INSTALL. Up to date information on Postgres may be found at www.postgresql.org.

The installation notes below assume the following (except where noted):

Commands are Unix-compatible. See note below.
Defaults are used except where noted.
User postgres is the Postgres superuser.
The source path is /usr/src/pgsql (other paths are possible).
The runtime path is /usr/local/pgsql (other paths are possible).

Commands were tested on RedHat Linux version 4.2 using the tcsh shell. Except where noted, they will probably work on most systems. Commands like ps and tar vary wildly on what options you should use on each platform. Use common sense before typing in these commands.

Our Makefiles require GNU make (called gmake in this document) and also assume that install accepts BSD options. The INSTALL variable in the Makefiles is set to the BSD-compatible version of install. On some systems, you will have to find a BSD-compatible install (eg. bsdinst, which comes with the MIT X Window System distribution).

Requirements to Run Postgres

Information on supported platforms is another chapter. In general, most Unix-compatible platforms with modern libraries should be able to run Postgres.

You should have at least 8 MB of memory and at least 45 MB of disk space to hold the source, binaries, and user databases. After installation you may reduce this to about 3 Mbytes plus space for user databases.

Installation Procedure

Procedure 3.1. Postgres Installation

For a fresh install or upgrading from previous releases of Postgres:

1. Read any last minute information and platform specific porting notes. There are some platform specific notes at the end of this file for Ultrix4.x, Linux, BSD/OS and NeXT. There are other files in directory /usr/src/pgsql/doc, including files FAQ-Irix and FAQ-Linux. Also look in directory ftp://ftp.postgresql.org/pub. If there is a file called INSTALL in this directory then this file will contain the latest installation information.

   Please note that a "tested" platform in the list given earlier simply means that someone went to the effort at some point of making sure that a Postgres distribution would compile and run on this platform without modifying the code. Since the current developers will not have access to all of these platforms, some of them may not compile cleanly and pass the regression tests in the current release due to minor problems. Any such known problems and their solutions will be posted in ftp://ftp.postgresql.org/pub/INSTALL.

2. Create account postgres if it does not already exist.

3. Log into account postgres.

    a. Check that you have sufficient disk space. You will need about 17 Mbytes for /usr/src/pgsql, about 2 Mbytes for /usr/local/pgsql (excluding your database) and 1 Mbyte for an empty database. The database will temporarily grow to about 20 Mbytes during the regression tests. You will also need about 3 Mbytes for the distribution tar file.

        We therefore recommend that during installation and testing you have well over 20 Mbytes free under /usr/local and another 25 Mbytes free on the disk partition containing your database. Once you delete the source files, tar file and regression database, you will need 2 Mbytes for /usr/local/pgsql, 1 Mbyte for the empty database, plus about five times the space you would require to store your database data in a flat file.

        To check for disk space, use df -k.

4. Ftp file ftp://ftp.postgresql.org/pub/postgresql-v6.3.tar.gz from the Internet. Store it in your home directory.

5. Some platforms use flex. If your system uses flex then make sure you have a good version. To check, type flex --version.

    If the flex command is not found then you probably do not need it. If the version is 2.5.2 or 2.5.4 or greater then you are okay. If it is 2.5.3 or before 2.5.2 then you will have to upgrade flex. You may get it at ftp://prep.ai.mit.edu/pub/gnu/flex-2.5.4.tar.gz.

    If you need flex and don't have it or have the wrong version, then you will be told so when you attempt to compile the program. Feel free to skip this step if you aren't sure you need it. If you do need it then you will be told to install/upgrade flex when you try to compile.

    To install it, type the following:

    ```
    cd
    gunzip -c flex-2.5.4.tar.gz | tar xvf -
    cd flex-2.5.4
    configure --prefix=/usr
    make
    make check
    # You must be root when typing the next line.
    make install
    cd
    rm -rf flex-2.5.4
    ```

    This will update files /usr/man/man1/flex.1, /usr/bin/flex, /usr/lib/libfl.a, /usr/include/FlexLexer.h and will add link /usr/bin/flex++ which points to flex.

6. If you are upgrading an existing system then back up your database. For alpha- and beta-level releases, the database format is liable to change often every few weeks with no notice besides a quick comment in the HACKERS mailing list. Full releases always require a dump/reload from previous releases. It is therefore a bad idea to skip this step. Also, do not use the pg_dumpall script from v6.0 or everything will be owned by the Postgres super user. Type (with the gunzip line and the following line typed as one line):

    ```
    cd
    gunzip -c postgresql-v6.3.tar.gz |
    tar xvf - src/bin/pg_dump/pg_dumpall
    chmod a+x src/bin/pg_dump/pg_dumpall
    src/bin/pg_dump/pg_dumpall > db.out
    rm -rf src
    ```

    If you wish to preserve object id's (oids), then use the -o option when running pg_dumpall. However, unless you have a special reason for doing this, don't do it.

If the pg_dumpall command seems to take a long time and you think it might have died, then, from another terminal, use "ls -l db.out" several times to see if the size of the file is growing.

Please note that if you are upgrading from a version prior to Postgres95 v1.09 then you must back up your database, install Postgres95 v1.09, restore your database, then back it up again. You should also read files /usr/src/pgsql/migration/*.

You must make sure that your database is not updated in the middle of your backup. If necessary, bring down postmaster, edit the permissions in file /usr/local/pgsql/data/pg_hba.conf to allow only you on, then bring postmaster back up.

7. If you are upgrading an existing system then kill the postmaster. Type

   ```
   ps -ax | grep postmaster
   ```

   This should list the process numbers for a number of processes. Type the following line, with "???" replaced by the process id for process "postmaster". (Do not use the id for process "grep postmaster".) Type kill ??? with "???" modified as indicated.

8. If you are upgrading an existing system then move the old directories out of the way. If you are short of disk space then you may have to back up and delete the directories instead. If you do this, save the old database in the /usr/local/pgsql/data directory tree. At a minimum, save file /usr/local/pgsql/data/pg_hba.conf.

   Type the following: su cd /usr/src mv pgsql pgsql_6_0 cd /usr/local mv pgsql pgsql_6_0 exit

   If you are not using /usr/local/pgsql/data as your data directory (check to see if environment variable PGDATA is set to something else) then you will also want to move this directory in the same manner.

9. Make new source and install directories. The actual paths can be different for your installation; be consistant throughout this procedure. Type

   ```
   su
   cd /usr/src
   mkdir pgsql
   chown postgres:postgres pgsql
   cd /usr/local
   mkdir pgsql
   chown postgres:postgres pgsql
   exit
   ```

10. Unzip and untar the new source file. Type

    ```
    cd /usr/src/pgsql
    gunzip -c ~/postgresql-v6.3.tar.gz | tar xvf -
    ```

11. Configure the source code for your system. It is this step at which you can specify your actual source path and installation paths for the build process (see the --prefix option below). Type

    ```
    cd /usr/src/pgsql/src
    ./configure
    ```

    The configure program will list the template files available and ask you to choose one. A lot of times, an appropriate template file is chosen for you, and you can just press Enter to accept the default. If the default is not appropriate, then type in the appropriate template file and press Enter. (If you do this, then send email to scrappy@hub.org stating the output of the program './config.guess' and what the template file should be.)

Once you have entered the template file, you will be asked a number of questions about your particular configuration. These can be skipped by adding parameters to the configure command above. The following parameters can be tagged onto the end of the configure command:

```
       --prefix=BASEDIR   Selects a different base directory for
the
                          installation of the Postgres
configuration.
                          The default is /usr/local/pgsql.

       --enable-hba       Enables Host Based Authentication
(DEFAULT)

       --disable-hba      Disables Host Based Authentication

       --enable-locale    Enables USE_LOCALE

       --disable-locale   Disables USE_LOCALE (DEFAULT)

       --enable-cassert   Enables ASSERT_CHECKING

       --disable-cassert  Disables ASSERT_CHECKING (DEFAULT)

       --with-template=TEMPLATE
                          Use template file TEMPLATE - the template
                          files are assumed to be in the directory
                          src/template, so look there for proper
values.
                          (If the configure script cannot find the
                          specified template file, it will ask you
for
                          one).

       --with-pgport=PORT Sets the port that the postmaster process
                          listens for incoming connections on.  The
                          default for this is port 5432.
```

As an example, here is the configure script I use on a Sparc Solaris 2.5 system with /opt/postgres being the install base.

```
./configure --prefix=/opt/postgres \
    --with-template=sparc_solaris-gcc --with-pgport=5432 \
    --enable-hba --disable-locale
```

Of course, in a real shell, you would type these three lines all on the same line.

12. Compile the program. Type

    ```
    cd /usr/src/pgsql/src
    gmake all >& make.log &
    tail -f make.log
    ```

    The last line displayed will hopefully be "All of PostgreSQL is successfully made. Ready to install." At this point, or earlier if you wish, type control-C to get out of tail. (If you have problems later on you may wish to examine file make.log for warning and error messages.)

    If your computer does not have gmake (GNU make) then try running make instead throughout the rest of these notes.

    Please note that you will probably find a number of warning messages in make.log. Unless you have problems later on, these messages may be safely ignored.

    If the compiler fails with an error stating that the flex command cannot be found then install flex as described earlier. Next, change directory back to this directory, type "make clean", then recompile again.

13. Install the program. Type

    ```
    cd /usr/src/pgsql/src
    gmake install >& make.install.log &
    tail -f make.install.log
    ```

    The last line displayed will be "gmake[1]: Leaving directory '/usr/src/pgsql/src/man'". At this point, or earlier if you wish, type control-C to get out of tail.

14. If necessary, tell UNIX how to find your shared libraries. If you are using Linux-ELF do ONE of the following, preferably the first:

    a. As root, edit file /etc/ld.so.conf. Add line /usr/local/pgsql/lib to the file. Then run command /sbin/ldconfig.

    b. In a bash shell, type

       ```
       export LD_LIBRARY_PATH=/usr/local/pgsql/lib
       ```

    c. In a csh shell, type

       ```
       setenv LD_LIBRARY_PATH /usr/local/pgsql/lib
       ```

    Please note that the above commands may vary wildly for different operating systems. Check the platform specific notes, such as those for Ultrix4.x or and for non-ELF Linux.

    If, when you create the database, you get the message "pg_id: can't load library 'libpq.so'" then the above step was necessary. Simply do this step, then try to create the database again.

15. If it has not already been done, then prepare account postgres for using Postgres. Any account that will use Postgres must be similarily prepared. (The following instructions are for a bash shell. Adapt accordingly for other shells.)

    Add the following lines to your login shell, ~/.bash_profile:

    ```
    PATH=$PATH:/usr/local/pgsql/bin
    MANPATH=$MANPATH:/usr/local/pgsql/man
    PGLIB=/usr/local/pgsql/lib
    PGDATA=/usr/local/pgsql/data
    export PATH MANPATH PGLIB PGDATA
    ```

    Make sure that you have defined these variables before continuing with the remaining steps. The easiest way to do this is to type:

    ```
    source ~/.bash_profile
    ```

16. Create the database. Do not do the following as root! This would be a major security hole. Type

    ```
    initdb
    ```

17. Set up permissions to access the database system. Do this by editing file /usr/local/pgsql/data/pg_hba.conf. The instructions are included in the file. (If your database is not located in the default location, i.e. if PGDATA is set to point elsewhere, then the location of this file will change accordingly.) This file should be made read only again once you are finsihed. If you are upgrading from v6.0 you can copy file pg_hba.conf from your old database on top of the one in your new database, rather than redoing this from scratch.

18. You may wish to skip the regression tests. However, we think skipping the tests is a BAD idea!

    The file /usr/src/pgsql/src/test/regress/README has detailed instructions for running and interpreting the regression tests. A short version follows here:

Start the postmaster daemon running in the background by typing

```
cd
nohup postmaster > regress.log 2>&1 &
```

Run postmaster from your Postgres super user account (typically account postgres). DO NOT RUN POSTMASTER FROM THE ROOT ACCOUNT.

19. Run the regression tests. Type

    ```
    cd
    cd /usr/src/pgsql/src/test/regress
    gmake clean
    gmake all runtest
    ```

    You do not need to type "gmake clean" if this is the first time you are running the tests.

    You should get on the screen (and also written to file ./regress.out) a series of statements stating which tests passed and which tests failed. Please note that it can be normal for some of the tests to "fail". For the failed tests, use diff to compare the files in directories ./results and ./expected. If float8 failed, type something like:

    ```
    cd /usr/src/pgsql/src/test/regress
    diff -w expected/float8.out results
    ```

    "Failed" tests may have failed due to slightly different error messages, output formatting, failure to set the timezone correctly for your platform, etc. "Failures" of this type do not indicate a problem with Postgres.

    For a i686/Linux-ELF platform, no tests failed since this is the v6.3 regression testing reference platform.

    For the SPARC/Linux-ELF platform, using the 970525 beta version of Postgres v6.2 the following tests "failed": float8 and geometry "failed" due to minor precision differences in floating point numbers. select_views produces massively different output, but the differences are due to minor floating point differences.

    Conclusion? If you do see failures, try to understand the nature of the differences and then decide if those differences will affect your intended use of Postgres. However, keep in mind that this is likely to be the most solid release of Postgres to date, incorporating many bug fixes from v6.2.1, and that previous versions of Postgres have been in use successfully for some time now.

    After running the tests, type

    ```
    destroydb regression
    cd /usr/src/pgsql/src/test/regress
    gmake clean
    ```

20. Stop the postmaster as described in step 7. Then restore the timezone to it’s normal setting. If you changed the timezone by modifying environment variable TZ then one way to do this is to log out of, then back into, account postgres.

21. Start the postmaster daemon running. Type

    ```
    cd
    nohup postmaster > server.log 2>&1 &
    ```

    Run postmaster from your Postgres super user account (typically account postgres). DO NOT RUN POSTMASTER FROM THE ROOT ACCOUNT.

22. If you haven’t already done so, this would be a good time to modify your computer so that it will automatically start postmaster whenever you boot your computer. Here are some suggestions on how to do this, contributed by various users. Whatever you do, postmaster must be run by user postgres AND NOT BY ROOT. This is why all of the examples below start by switching user (su) to postgres. These commands also take into account the fact that environment variables like PATH and PGDATA may not be set properly. The examples are as follows. Use them with extreme caution. a) Edit file rc.local on NetBSD

or file rc2.d on SPARC Solaris 2.5.1 to contain the following single line: su postgres -c "/usr/local/pgsql/bin/postmaster -S -D /usr/local/pgsql/data" b) In FreeBSD 2.2-RELEASE edit /usr/local/etc/rc.d/pgsql.sh to contain the following lines and make it chmod 755 and chown root:bin. #!/bin/sh [-x /usr/local/pgsql/bin/postmaster] && { su -l pgsql -c 'exec /usr/local/pgsql/bin/postmaster -D/usr/local/pgsql/data -S -o -F > /usr/local/pgsql/errlog' & echo -n ' pgsql' } You may put the line breaks as shown above. The shell is smart enough to keep parsing beyond end-of-line if there is an expression unfinished. The exec saves one layer of shell under the postmaster process so the parent is init. Note: Unlike most other examples, this one has been tested. c) In RedHat v4.0 Linux edit file /etc/inittab to contain the following single line: pg:2345:respawn:/bin/su - postgres -c "/usr/local/pgsql/bin/postmaster -D/usr/local/pgsql/data >> /usr/local/pgsql/server.log 2>&1" /dev/null (The author of this example says this example will revive the postmaster if it dies, but he doesn't know if there are other side effects.) d) The contrib/linux area of the Postgres distribution has an example init.d script compatible with and tested using recent RedHat packages.

23. If you haven't already done so, this would be a good time to modify your computer to do regular maintainence. The following should be done at regular intervals: a) Run the SQL command vacuum. This will clean up your database. b) Back up your system. (You should probably keep the last few backups on hand.) Ideally, no one else should be using the system at the time. Ideally, the above tasks should be done by a shell script that is run nightly or weekly by cron. Look at the man page for crontab for a starting point on how to do this. (If you do it, please e-mail us a copy of your shell script. We would like to set up our own systems to do this too.)

24. If you are upgrading an existing system then install your old database. Type

```
cd
psql -e template1 < db.out
```

If your pre-v6.2 database uses either path or polygon geometric data types, then you will need to upgrade any columns containing those types. To do so, type (from within psql)

```
update YourTable set PathCol = UpgradePath(PathCol);
update YourTable set PolyCol = UpgradePoly(PolyCol);
...
vacuum;
```

UpgradePath() checks to see that a path value is consistant with the old syntax, and will not update a column which fails that examination. UpgradePoly() cannot verify that a polygon is in fact from an old syntax, but RevertPoly() is provided to reverse the effects of a mis-applied upgrade.

25. If you are a new user, you may wish to play with Postgres as described below.

26. Clean up after yourself. Type

```
rm -rf /usr/src/pgsql_6_0
rm -rf /usr/local/pgsql_6_0
# Also delete old database directory tree if it is not in
#  /usr/local/pgsql_6_0/data
rm ~/postgresql-v6.2.1.tar.gz
```

27. You will probably want to print out the documentation. Here is how you might do it if you have Ghostscript on your system and are writing to a laserjet printer. alias gshp='gs -sDEVICE=laserjet -r300 -dNOPAUSE' export GS_LIB=/usr/share/ghostscript:/usr/share/ghostscript/fonts # Print out the man pages. man -a -t /usr/local/pgsql/man/*/* > manpage.ps gshp -sOUTPUTFILE=manpage.hp manpage.ps rm manpage.ps lpr -l -s -r manpage.hp # Print out the Postgres95 User Manual, version 1.0, # Sept. 5, 1996. cd /usr/src/pgsql/doc gshp -sOUTPUTFILE=userguide.hp userguide.ps lpr -l -s -r userguide.hp If you are a developer, you will probably want to also print out the Postgres Implemention Guide, version 1.0, October 1, 1995. This is a WWW document located at http://www.postgresql.org/docs/impguide.

28. The Postgres team wants to keep Postgres working on all of the supported platforms. We therefore ask you to let us know if you did or did not get Postgres to work on you system. Please send a mail message to pgsql-ports@postgresql.org telling us the following: - The version of Postgres (v6.2.1, 6.1.1, beta 970703, etc.). - Your operating system (i.e. RedHat v4.0 Linux v2.0.26). - Your hardware (SPARC, i486, etc.). - Did you compile, install and run the regression tests cleanly? If not, what source code did you change (i.e. patches you applied, changes you made, etc.), what tests failed, etc. It is normal to get many warning when you compile. You do not need to report these.
29. Now create, access and manipulate databases as desired. Write client programs to access the database server. In other words, ENJOY!

Playing with Postgres

After Postgres is installed, a database system is created, a postmaster daemon is running, and the regression tests have passed, you'll want to see Postgres do something. That's easy. Invoke the interactive interface to Postgres, psql:

```
% psql template1
```

(psql has to open a particular database, but at this point the only one that exists is the template1 database, which always exists. We will connect to it only long enough to create another one and switch to it.)

The response from psql is:

```
Welcome to the POSTGRESQL interactive sql monitor:
  Please read the file COPYRIGHT for copyright terms of POSTGRESQL

   type \? for help on slash commands
   type \q to quit
   type \g or terminate with semicolon to execute query
 You are currently connected to the database: template1

template1=>
```

Create the database foo:

```
template1=> create database foo;
CREATEDB
```

(Get in the habit of including those SQL semicolons. Psql won't execute anything until it sees the semicolon or a "\g" and the semicolon is required to delimit multiple statements.)

Now connect to the new database:

```
template1=> \c foo
connecting to new database: foo
```

("slash" commands aren't SQL, so no semicolon. Use \? to see all the slash commands.)

And create a table:

```
foo=> create table bar (i int4, c char(16));
CREATE
```

Then inspect the new table:

```
foo=> \d bar

Table    = bar
+--------------------------+-----------------------------------+-------+
|          Field           |              Type                 | Length|
+--------------------------+-----------------------------------+-------+
| i                        | int4                              |     4 |
| c                        | (bp)char                          |    16 |
+--------------------------+-----------------------------------+-------+
```

And so on. You get the idea.

The Next Step

Questions? Bugs? Feedback? First, read the files in directory /usr/src/pgsql/doc. The FAQ in this directory may be particularly useful.

If Postgres failed to compile on your computer then fill out the form in file /usr/src/pgsql/doc/bug.template and mail it to the location indicated at the top of the form.

Mail questions to pgsql-questions@postgresql.org. For more information on the various mailing lists, see http://www.postgresql.org and look for the mailing lists.

Porting Notes

NOTE: For some ports, these notes may be out of date.

Ultrix4.x

You need to install the libdl-1.1 package since Ultrix 4.x doesn't have a dynamic loader. It's available in s2k-ftp.CS.Berkeley.EDU:pub/personal/andrew/libdl-1.1.tar.Z

Linux

Linux ELF

The regression test reference machine is a linux-2.0.30/libc-5.3.12/RedHat-4.2 installation running on a dual processor i686. The linux-elf port installs cleanly. See the Linux FAQ for more details.

Linux a.out

For non-ELF Linux, the dld library MUST be obtained and installed on the system. It enables dynamic link loading capability to the Postgres port. The dld library can be obtained from the sunsite linux distributions. The current name is dld-3.2.5. Jalon Q. Zimmerman

BSD/OS

For BSD/OS 2.0 and 2.01, you will need to get the GNU dld library.

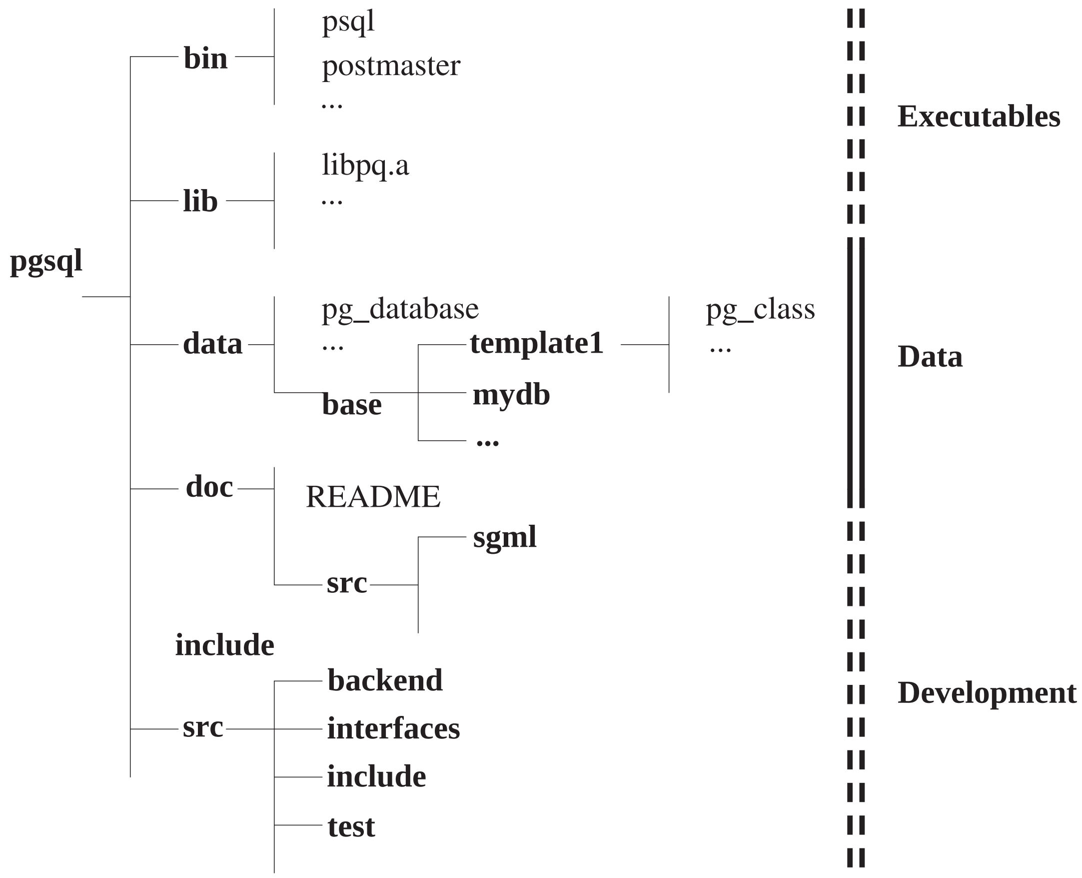
NeXT

The NeXT port for v1.09 was supplied by Tom R. Hageman. It requires a SysV IPC emulation library and header files for shared libary and semaphore stuff. Tom just happens to sell such a product so contact him for information. He has also indicated that binary releases of Postgres for NEXTSTEP will be made available to the general public. Contact Info@RnA.nl for information.

We have no recent reports of successful NeXT installations (for v6.2.1). However, the client-side libraries should work even if the backend is not supported.

Chapter 4. Runtime Environment

Figure 4-1. Postgres file layout



Postgres file layout shows how the Postgres distribution is laid out when installed in the default way. For simplicity, we will assume that Postgres has been installed in the directory /usr/local/pgsql. Therefore, wherever you see the directory /usr/local/pgsql you should substitute the name of the directory where Postgres is actually installed. All Postgres commands are installed in the directory /usr/local/pgsql/bin. Therefore, you should add this directory to your shell command path. If you use a variant of the Berkeley C shell, such as csh or tcsh, you would add

```
set path = ( /usr/local/pgsql/bin path )
```

in the .login file in your home directory. If you use a variant of the Bourne shell, such as sh, ksh, or bash, then you would add

```
PATH=/usr/local/pgsql/bin PATH
export PATH
```

to the .profile file in your home directory. From now on, we will assume that you have added the Postgres bin directory to your path. In addition, we will make frequent reference to "setting a shell variable" or "setting an environment variable" throughout this document. If you did not fully understand the last paragraph on modifying your search path, you should consult the UNIX manual pages that describe your shell before going any further.

If your site administrator has not set things up in the default way, you may have some more work to do. For example, if the database server machine is a remote machine, you will need to set the PGHOST environment variable to the name of the database server machine. The environment variable PGPORT may also have to be set. The bottom line is this: if you try to start an application program and it complains that it cannot connect to the postmaster, you should immediately consult your site administrator to make sure that your environment is properly set up.

Locale Support

> NOTE: Written by Oleg Bartunov. See Oleg's web page for additional information on locale and Russian language support.

While doing a project for a company in Moscow, Russia, I encountered the problem that postgresql had no support of national alphabets. After looking for possible workarounds I decided to develop support of locale myself. I'm not a C-programer but already had some experience with locale programming when I work with perl (debugging) and glimpse. After several days of digging through the Postgres source tree I made very minor corections to src/backend/utils/adt/varlena.c and src/backend/main/main.c and got what I needed! I did support only for LC_CTYPE and LC_COLLATE, but later LC_MONETARY was added by others. I got many messages from people about this patch so I decided to send it to developers and (to my surprise) it was incorporated into postgresql distribution.

People often complain that locale doesn't work for them. There are several common mistakes:

Didn't properly configure postgresql before compilation. You must run configure with --enable-locale option to enable locale support. Didn't setup environment correctly when starting postmaster. You must define environment variables $LC_CTYPE and $LC_COLLATE before running postmaster because backend gets information about locale from environment. I use following shell script (runpostgres):

```
#!/bin/sh

export LC_CTYPE=koi8-r
export LC_COLLATE=koi8-r
postmaster -B 1024 -S -D/usr/local/pgsql/data/ -o '-Fe'
```

and run it from rc.local as

```
/bin/su - postgres -c "/home/postgres/runpostgres"
```

Broken locale support in OS (for example, locale support in libc under Linux several times has changed and this caused a lot of problems). Latest perl has also support of locale and if locale is broken perl -v will complain something like: 8:17[mira]:~/WWW/postgres>setenv LC_CTYPE not_exist 8:18[mira]:~/WWW/postgres>perl -v perl: warning: Setting locale failed. perl: warning: Please check that your locale settings: LC_ALL = (unset), LC_CTYPE = "not_exist", LANG = (unset) are supported and installed on your system. perl: warning: Falling back to the standard locale ("C").

Wrong location of locale files! Possible location: /usr/lib/locale (Linux, Solaris), /usr/share/locale (Linux), /usr/lib/nls/loc (DUX 4.0) Check man locale for right place. Under Linux I did a symbolical link between /usr/lib/locale and /usr/share/locale to be sure next libc will not break my locale.

What are the Benefits?

You can use ~* and order by operators for strings contain characters from national alphabets. Non-english users definitely need that. If you won’t use locale stuff just undefine USE_LOCALE variable.

What are the Drawbacks?

There is one evident drawback of using locale - it’s speed ! So, use locale only if you really need it.

Chapter 5. Starting postmaster

Nothing can happen to a database unless the postmaster process is running. As the site administrator, there are a number of things you should remember before starting the postmaster. These are discussed in the section of this manual titled, "Administering Postgres." However, if Postgres has been installed by following the installation instructions exactly as written, the following simple command is all you should need to start the postmaster:

```
% postmaster
```

The postmaster occasionally prints out messages which are often helpful during troubleshooting. If you wish to view debugging messages from the postmaster, you can start it with the -d option and redirect the output to the log file:

```
% postmaster -d >& pm.log &
```

If you do not wish to see these messages, you can type

```
% postmaster -S
```

and the postmaster will be "S"ilent. Notice that there is no ampersand ("&") at the end of the last example.

Chapter 6. Adding and Deleting Users

createuser enables specific users to access Postgres. destroyuser removes users and prevents them from accessing Postgres. Note that these commands only affect users with respect to Postgres; they have no effect on users other privileges or status with regards to the underlying operating system.

Chapter 7. Disk Management

Alternate Locations

It is possible to create a database in a location other than the default location for the installation. Remember that all database access actually occurs through the database backend, so that any location specified must be accessible by the backend.

Either an absolute path name or an environment variable may be specified as a location. Note that for security and integrity reasons, all paths and environment variables so specified have some additional path fields appended.

> NOTE: The environment variable style of specification is to be preferred since it allows the site administrator more flexibility in managing disk storage.

Remember that database creation is actually performed by the database backend. Therefore, any environment variable specifying an alternate location must have been defined before the backend was started. To define an alternate location PGDATA2 pointing to /home/postgres/data, type

```
% setenv PGDATA2 /home/postgres/data
```

Usually, you will want to define this variable in the Postgres superuser's .profile or .cshrc initialization file to ensure that it is defined upon system startup.

To create a data storage area in /home/postgres/data, ensure that /home/postgres already exists and is writable. From the command line, type

```
% initlocation $PGDATA2
Creating Postgres database system directory /home/postgres/data

Creating Postgres database system directory /home/postgres/data/base
```

To test the new location, create a database test by typing

```
% createdb -D PGDATA2 test
% destroydb test
```

Chapter 8. Troubleshooting

Assuming that your site administrator has properly started the postmaster process and authorized you to use the database, you (as a user) may begin to start up applications. As previously mentioned, you should add /usr/local/pgsql/bin to your shell search path. In most cases, this is all you should have to do in terms of preparation.

If you get the following error message from a Postgres command (such as psql or createdb):

```
connectDB() failed: Is the postmaster running at ’localhost’ on port
’4322’?
```

it is usually because either the postmaster is not running, or you are attempting to connect to the wrong server host. If you get the following error message:

```
FATAL 1:Feb 17 23:19:55:process userid (2360) != database owner (268)
```

it means that the site administrator started the postmaster as the wrong user. Tell him to restart it as the Postgres superuser.

Chapter 9. Managing a Database

Now that Postgres is up and running we can create some databases to experiment with. Here, we describe the basic commands for managing a database.

Creating a Database

Let's say you want to create a database named mydb. You can do this with the following command:

```
% createdb mydb
```

Postgres allows you to create any number of databases at a given site and you automatically become the database administrator of the database you just created. Database names must have an alphabetic first character and are limited to 16 characters in length. Not every user has authorization to become a database administrator. If Postgres refuses to create databases for you, then the site administrator needs to grant you permission to create databases. Consult your site administrator if this occurs.

Accessing a Database

Once you have constructed a database, you can access it by:

running the Postgres terminal monitor programs (monitor or psql) which allows you to interactively enter, edit, and execute SQL commands.
writing a C program using the LIBPQ subroutine library. This allows you to submit SQL commands from C and get answers and status messages back to your program. This interface is discussed further in section ??.

You might want to start up psql, to try out the examples in this manual. It can be activated for the mydb database by typing the command:

```
% psql mydb
```

You will be greeted with the following message:

```
Welcome to the Postgres interactive sql monitor:

  type \? for help on slash commands
  type \q to quit
  type \g or terminate with semicolon to execute query
You are currently connected to the database: mydb

mydb=>
```

This prompt indicates that the terminal monitor is listening to you and that you can type SQL queries into a workspace maintained by the terminal monitor. The psql program responds to escape codes that begin with the backslash character, "\". For example, you can get help on the syntax of various Postgres SQL commands by typing:

```
mydb=> \h
```

Once you have finished entering your queries into the workspace, you can pass the contents of the workspace to the Postgres server by typing:

```
mydb=> \g
```

This tells the server to process the query. If you terminate your query with a semicolon, the backslash-g is not necessary. psql will automatically process semicolon terminated queries. To read queries from a file, say myFile, instead of entering them interactively, type:

```
mydb=> \i fileName
```

To get out of psql and return to UNIX, type

```
mydb=> \q
```

and psql will quit and return you to your command shell. (For more escape codes, type backslash-h at the monitor prompt.) White space (i.e., spaces, tabs and newlines) may be used freely in SQL queries. Single-line comments are denoted by -- . Everything after the dashes up to the end of the line is ignored. Multiple-line comments, and comments within a line, are denoted by /* ... */

Destroying a Database

If you are the database administrator for the database mydb, you can destroy it using the following UNIX command:

```
% destroydb mydb
```

This action physically removes all of the UNIX files associated with the database and cannot be undone, so this should only be done with a great deal of forethought.

Chapter 10. Database Recovery

This section needs to be written. Volunteers?

Chapter 11. Regression Test

Regression test instructions and analysis.

The PostgreSQL regression tests are a comprehensive set of tests for the SQL implementation embedded in PostgreSQL developed by Jolly Chen and Andrew Yu. It tests standard SQL operations as well as the extended capabilities of PostgreSQL.

These tests have recently been revised by Marc Fournier and Thomas Lockhart and are now packaged as functional units which should make them easier to run and easier to interpret. From PostgreSQL v6.1 onward the regression tests are current for every official release.

Some properly installed and fully functional PostgreSQL installations can fail some of these regression tests due to artifacts of floating point representation and time zone support. The current tests are evaluated using a simple "diff" algorithm, and are sensitive to small system differences. For apparently failed tests, examining the differences may reveal that the differences are not significant.

The regression testing notes below assume the following (except where noted):

Commands are Unix-compatible. See note below.
Defaults are used except where noted.
User postgres is the Postgres superuser.
The source path is /usr/src/pgsql (other paths are possible).
The runtime path is /usr/local/pgsql (other paths are possible).

Regression Environment

The regression test is invoked by the make command which compiles a C program into a shared library in the current directory. Localized shell scripts are also created in the current directory. The output file templates are massaged into the ./expected/*.out files. The localization replaces macros in the source files with absolute pathnames and user names.

Normally, the regression test should be run as the pg_superuser since the 'src/test/regress' directory and sub-directories are owned by the pg_superuser. If you run the regression test as another user the 'src/test/regress' directory tree should be writeable to that user.

The postmaster should be invoked with the system time zone set for Berkeley, California. This is done automatically by the regression test script. However, it does require machine support for the PST8PDT time zone.

To verify that your machine does have this support, type the following:

```
setenv TZ PST8PDT
date
```

The "date" command above should have returned the current system time in the PST8PDT time zone. If the PST8PDT database is not available, then your system may have returned the time in GMT. If the PST8PDT time zone is not available, you can set the time zone rules explicitly:

```
setenv PGTZ PST8PDT7,M04.01.0,M10.05.03
```

Directory Layout

NOTE: This should become a table in the previous section.

```
  input/ ... .source files that are converted using 'make all' into
              some of the .sql files in the 'sql' subdirectory

  output/ .. .source files that are converted using 'make all' into
              .out files in the 'expected' subdirectory

  sql/ ..... .sql files used to perform the regression tests

  expected/  .out files that represent what we *expect* the results to
              look like

  results/ . .out files that represent what the results *actually* look
              like. Also used as temporary storage for table copy
testing.
```

Regression Test Procedure

Commands were tested on RedHat Linux version 4.2 using the bash shell. Except where noted, they will probably work on most systems. Commands like ps and tar vary wildly on what options you should use on each platform. Use common sense before typing in these commands.

Procedure 11.1. Postgres Regression Configuration

For a fresh install or upgrading from previous releases of Postgres:

1. Build the regression test. Type

   ```
   cd /usr/src/pgsql/src/test/regress
   gmake all
   ```

2. If you have prevously invoked the regression test, clean up the working directory with:

   ```
   cd /usr/src/pgsql/src/test/regress
   make clean
   ```

3. The file /usr/src/pgsql/src/test/regress/README has detailed instructions for running and interpreting the regression tests. A short version follows here:

   If the postmaster is not already running, start the postmaster on an available window by typing

   ```
   postmaster
   ```

   or start the postmaster daemon running in the background by typing

   ```
   cd
   nohup postmaster > regress.log 2>&1 &
   ```

   Run postmaster from your Postgres super user account (typically account postgres).

   NOTE: Do not run postmaster from the root account.

4. Run the regression tests. Type

   ```
   cd /usr/src/pgsql/src/test/regress
   gmake runtest
   ```

   You do not need to type "gmake clean" if this is the first time you are running the tests.

5. You should get on the screen (and also written to file ./regress.out) a series of statements stating which tests passed and which tests failed. Please note that it can be normal for some of the tests to "fail". For the failed tests, use diff to compare the files in directories ./results and ./expected. If float8 failed, type something like:

   ```
   cd /usr/src/pgsql/src/test/regress
   diff -w expected/float8.out results
   ```

6. After running the tests, type

   ```
   destroydb regression
   cd /usr/src/pgsql/src/test/regress
   gmake clean
   ```

Regression Analysis

Failed tests may have failed due to slightly different error messages, math libraries, or output formatting. "Failures" of this type do not indicate a problem with Postgres.

For a i686/Linux-ELF platform, no tests failed since this is the v6.2.1 regression testing reference platform.

For the SPARC/Linux-ELF platform, using the 970525 beta version of Postgres v6.2 the following tests "failed": float8 and geometry "failed" due to minor precision differences in floating point numbers. select_views produces massively different output, but the differences are due to minor floating point differences.

Conclusion? If you do see failures, try to understand the nature of the differences and then decide if those differences will affect your intended use of Postgres. However, keep in mind that this is likely to be the most solid release of Postgres to date, incorporating many bug fixes from v6.1, and that previous versions of Postgres have been in use successfully for some time now.

Comparing expected/actual output

The results are in files in the ./results directory. These results can be compared with results in the ./expected directory using 'diff'. The files might not compare exactly. The following paragraphs attempt to explain the differences.

Error message differences

Some of the regression tests involve intentional invalid input values. Error messages can come from either the Postgres code or from the host platform system routines. In the latter case, the messages may vary between platforms, but should reflect similar information. These differences in messages will result in a "failed" regression test which can be validated by inspection.

OID differences

There are several places where PostgreSQL OID (object identifiers) appear in 'regress.out'. OID's are unique 32-bit integers which are generated by the PostgreSQL backend whenever a table row is inserted or updated. If you run the regression test on a non-virgin database or run it multiple times, the OID's reported will have different values. The following SQL statements in 'misc.out' have shown this behavior: QUERY: SELECT user_relns() AS user_relns ORDER BY user_relns; The 'a,523676' row is composed from an OID.

Date and time differences

On many supported platforms, you can force PostgreSQL to believe that it is running in the same time zone as Berkeley, California. See details in the section on how to run the regression tests. If you do not explicitly set your time zone environment to PST8PDT, then most of the date and time results will reflect your local time zone and will fail the regression testing. There appears to be some systems which do not accept the recommended syntax for explicitly setting the local time zone rules. Some systems using the public domain time zone package exhibit minor problems with pre-1970 PDT times, representing them in PST instead.

Floating point differences

Some of the tests involve computing 64-bit (float8) number from table columns. Differences in results involving mathematical functions of float8 columns have been observed. These differences occur where different operating systems are used on the same platform ie: BSDI and SOLARIS on Intel/86, and where the same operating system is used used on different platforms, ie: SOLARIS on SPARC and Intel/86. Human eyeball comparison is needed to determine the real significance of these differences which are usually 10 places to the right of the decimal point. Some systems signal errors from pow() and exp() differently from the mechanism expected by the current Postgres code.

Polygon differences

Several of the tests involve operations on geographic date about the Oakland/Berkley CA street map. The map data is expressed as polygons whose vertices are represented as pairs of float8 numbers (decimal latitude and longitude). Initially, some tables are created and loaded with geographic data, then some views are created which join two tables using the polygon intersection operator (##), then a select is done on the view. When comparing the results from different platforms, differences occur in the 2nd or 3rd place to the right of the decimal point. The SQL statements where these problems occur are the folowing:

```
QUERY: SELECT * from street;
QUERY: SELECT * from iexit;
```

Random differences

There is at least one test case in random.out which is intended to produce random results. This causes random to fail the regression testing. Typing

```
diff results/random.out expected/random.out
```

should produce only one or a few lines of differences for this reason, but other floating point differences on dissimilar architectures might cause many more differences. See the release notes below.

The expected files

The ./expected/*.out files were adapted from the original monolithic expected.input file provided by Jolly Chen et al. Newer versions of these files generated on various development machines have been substituted after careful (?) inspection. Many of the development machines are running a Unix OS variant (FreeBSD, Linux, etc) on Ix86 hardware. The original expected.input file was created on a SPARC Solaris 2.4 system using the postgres5-1.02a5.tar.gz source tree. It was compared with a file created on an I386 Solaris 2.4 system and the differences were only in the floating point polygons in the 3rd digit to the right of the decimal point. (see below) The original sample.regress.out file was from the postgres-1.01 release constructed by Jolly Chen and is included here for reference. It may have been created on a DEC ALPHA machine as the Makefile.global in the postgres-1.01 release has PORTNAME=alpha.

Chapter 12. Release Notes

> NOTE: Should include the migration notes from migration/.

The release notes have not yet been integrated into the new documentation. Check for plain text files in the top of the distribution directory tree and in the migration/ directory for current information.

Release 6.3

TBD

Release 6.2.1

> NOTE: v6.2.1 was a bug-fix and usability release on v6.2. Needs only a few notes.

Release 6.2

> NOTE: This should include information based on Bruce’s release summary.

Release 6.1

> NOTE: This should include information based on Bruce’s release summary.

The regression tests have been adapted and extensively modified for the v6.1 release of PostgreSQL.

Three new data types (datetime, timespan, and circle) have been added to the native set of PostgreSQL types. Points, boxes, paths, and polygons have had their output formats made consistant across the data types. The polygon output in misc.out has only been spot-checked for correctness relative to the original regression output.

PostgreSQL v6.1 introduces a new, alternate optimizer which uses genetic algorithms. These algorithms introduce a random behavior in the ordering of query results when the query contains multiple qualifiers or multiple tables (giving the optimizer a choice on order of evaluation). Several regression tests have been modified to explicitly order the results, and hence are insensitive to optimizer choices. A few regression tests are for data types which are inherently unordered (e.g. points and time intervals) and tests involving those types are explicitly bracketed with set geqo to ’off’ and reset geqo.

The interpretation of array specifiers (the curly braces around atomic values) appears to have changed sometime after the original regression tests were generated. The current ./expected/*.out files reflect this new interpretation, which may not be correct!

The float8 regression test fails on at least some platforms. This is due to differences in implementations of pow() and exp() and the signaling mechanisms used for overflow and underflow conditions.

The "random" results in the random test should cause the "random" test to be "failed", since the regression tests are evaluated using a simple diff. However, "random" does not seem to produce random results on my test machine (Linux/gcc/i686).

Timing Results

These timing results are from running the regression test with the command

```
% time make runtest
```

Timing under Linux 2.0.27 seems to have a roughly 5% variation from run to run, presumably due to the timing vagaries of multitasking systems.

v6.3

```
  Time   System
  02:30  Dual Pentium Pro 180, 96MB, UW-SCSI, Linux 2.0.30, gcc 2.7.2.1
-O2 -m486
  04:12  Dual Pentium Pro 180, 96MB, EIDE, Linux 2.0.30, gcc 2.7.2.1
-O2 -m486
```

v6.1

```
  Time   System
  06:12  Pentium Pro 180, 32MB, Linux 2.0.30, gcc 2.7.2 -O2 -m486
  12:06  P-100, 48MB, Linux 2.0.29, gcc
  39:58  Sparc IPC 32MB, Solaris 2.5, gcc 2.7.2.1 -O -g
```

SQL References

Selected references and readings for SQL and Postgres.

SQL Reference Books

Reference texts for SQL features.

The Practical SQL Handbook, Judity Bowman, Sandra Emerson, and Marcy Damovsky, ISBN: 0-201-44787-8, Addison-Wesley, 1997.

A Guide to The SQL Standard, C. J. Date and Hugh Darwen, ISBN: 0-201-96426-0, Addison-Wesley, 1997.

Understanding the New SQL, Jim Melton and Alan R. Simon, ISBN: 1-55860-245-3, Morgan Kaufmann, 1993.

PostgreSQL-Specific Documentation

This section is for related documentation.

The PostgreSQL Administrator's Guide, The PostgreSQL Global Development Group.

The PostgreSQL Programmer's Guide, The PostgreSQL Global Development Group.

The PostgreSQL Reference Manual, The PostgreSQL Global Development Group.

The PostgreSQL Tutorial Introduction, The PostgreSQL Global Development Group.

The PostgreSQL User's Guide, The PostgreSQL Global Development Group.

The Postgres95 User Manual, A. Yu and J. Chen, The POSTGRES Group, University of California, Berkeley CA.

Proceedings and Articles

This section is for articles and newsletters.

A Unified Framework for Version Modeling Using Production Rules in a Database System, L. Ong and J. Goh, ISSN: ERL Technical Memorandum M90/33, University of California, Berkeley CA.

The Postgres Data Model, L. Rowe and M. Stonebraker.

The Design of Postgres, M. Stonebraker and L. Rowe.

The Design of the Postgres Rules System, M. Stonebraker, E. Hanson, and C. H. Hong.

The Postgres Storage System, M. Stonebraker.

A Commentary on the Postgres Rules System, M. Stonebraker, M. Hearst, and S. Potamianos.

The Implementation of Postgres, M. Stonebraker, L. A. Rowe, and M. Hirohama.

On Rules, Procedures, Caching and Views in Database Systems, M. Stonebraker and et. al..