PostgreSQL 7.2 Administrator’s Guide

The PostgreSQL Global Development Group

PostgreSQL 7.2 Administrator's Guide

by The PostgreSQL Global Development Group

Legal Notice

Table of Contents

Preface ..... **ix**
1. What is PostgreSQL? ix
2. A Short History of PostgreSQL ix
2.1. The Berkeley POSTGRES Project x
2.2. Postgres95 x
2.3. PostgreSQL xi
3. Documentation Resources xi
4. Terminology and Notation xii
5. Bug Reporting Guidelines xiii
5.1. Identifying Bugs xiii
5.2. What to report xiv
5.3. Where to report bugs xv
6. Y2K Statement xvi
1. Installation Instructions **1**
1.1. Short Version 1
1.2. Requirements 1
1.3. Getting The Source 2
1.4. If You Are Upgrading 2
1.5. Installation Procedure 3
1.6. Post-Installation Setup 9
1.6.1. Shared Libraries 9
1.6.2. Environment Variables 10
1.7. Supported Platforms 10
2. Installation on Windows **16**
3. Server Runtime Environment **17**
3.1. The PostgreSQL user account 17
3.2. Creating a database cluster 17
3.3. Starting the database server 18
3.3.1. Server Start-up Failures 19
3.3.2. Client Connection Problems 20
3.4. Run-time configuration 21
3.4.1. Planner and Optimizer Tuning 22
3.4.2. Logging and Debugging 24
3.4.3. General operation 25
3.4.4. WAL 29
3.4.5. Short options 30
3.5. Managing Kernel Resources 31
3.5.1. Shared Memory and Semaphores 31
3.5.2. Resource Limits 35
3.6. Shutting down the server 36
3.7. Secure TCP/IP Connections with SSL 37
3.8. Secure TCP/IP Connections with SSH tunnels 37
4. Client Authentication **39**
4.1. The `pg_hba.conf` file 39
4.2. Authentication methods 43
4.2.1. Trust authentication 43
4.2.2. Password authentication 43
4.2.3. Kerberos authentication 44

4.2.4. Ident-based authentication ... 45
4.3. Authentication problems ... 46
5. Localization ... 47
5.1. Locale Support ... 47
5.1.1. Overview ... 47
5.1.2. Benefits ... 48
5.1.3. Problems ... 48
5.2. Multibyte Support ... 49
5.2.1. Enabling Multibyte Support ... 50
5.2.2. Setting the Encoding ... 51
5.2.3. Automatic encoding translation between server and client ... 51
5.2.4. About Unicode ... 53
5.2.5. What happens if the translation is not possible? ... 53
5.2.6. References ... 53
5.2.7. History ... 54
5.2.8. WIN1250 on Windows/ODBC ... 55
5.3. Single-byte character set recoding ... 56
6. Managing Databases ... 58
6.1. Creating a Database ... 58
6.1.1. Template Databases ... 58
6.1.2. Alternative Locations ... 59
6.2. Destroying a Database ... 60
7. Database Users and Permissions ... 62
7.1. Database Users ... 62
7.1.1. User attributes ... 62
7.2. Groups ... 63
7.3. Privileges ... 63
7.4. Functions and Triggers ... 63
8. Routine Database Maintenance Tasks ... 65
8.1. General Discussion ... 65
8.2. Routine Vacuuming ... 65
8.2.1. Recovering disk space ... 65
8.2.2. Updating planner statistics ... 66
8.2.3. Preventing transaction ID wraparound failures ... 67
8.3. Log File Maintenance ... 68
9. Backup and Restore ... 70
9.1. SQL Dump ... 70
9.1.1. Restoring the dump ... 70
9.1.2. Using **pg_dumpall** ... 71
9.1.3. Large Databases ... 71
9.1.4. Caveats ... 72
9.2. File system level backup ... 72
9.3. Migration between releases ... 73
10. Monitoring Database Activity ... 75
10.1. Standard Unix Tools ... 75
10.2. Statistics Collector ... 75
10.2.1. Statistics Collection Configuration ... 76
10.2.2. Viewing Collected Statistics ... 76

11. Write-Ahead Logging (WAL) 80
11.1. General Description 80
11.1.1. Immediate Benefits of WAL 80
11.1.2. Future Benefits 80
11.2. Implementation 81
11.2.1. Database Recovery with WAL 81
11.3. WAL Configuration 82
12. Disk Storage 84
13. Database Failures 85
13.1. Disk Filled 85
13.2. Disk Failed 85
14. Regression Tests 86
14.1. Introduction 86
14.2. Running the Tests 86
14.3. Test Evaluation 87
14.3.1. Error message differences 87
14.3.2. Locale differences 87
14.3.3. Date and time differences 88
14.3.4. Floating-point differences 88
14.3.5. Polygon differences 88
14.3.6. Row ordering differences 89
14.3.7. The “random” test 89
14.4. Platform-specific comparison files 89
A. Release Notes 91
A.1. Release 7.2 91
A.1.1. Overview 91
A.1.2. Migration to version 7.2 91
A.1.3. Changes 92
A.1.3.1. Server Operation 92
A.1.3.2. Performance 92
A.1.3.3. Privileges 93
A.1.3.4. Client Authentication 93
A.1.3.5. Server Configuration 93
A.1.3.6. Queries 93
A.1.3.7. Schema Manipulation 93
A.1.3.8. Utility Commands 94
A.1.3.9. Data Types and Functions 94
A.1.3.10. Internationalization 95
A.1.3.11. PL/pgSQL 95
A.1.3.12. PL/Perl 95
A.1.3.13. PL/Tcl 96
A.1.3.14. PL/Python 96
A.1.3.15. Psql 96
A.1.3.16. Libpq 96
A.1.3.17. JDBC 96
A.1.3.18. ODBC 97
A.1.3.19. ECPG 97
A.1.3.20. Misc. Interfaces 97
A.1.3.21. Build and Install 98
A.1.3.22. Source Code 98

A.1.3.23. Contrib 98
A.2. Release 7.1.3 98
A.2.1. Migration to version 7.1.3 99
A.2.2. Changes 99
A.3. Release 7.1.2 99
A.3.1. Migration to version 7.1.2 99
A.3.2. Changes 99
A.4. Release 7.1.1 99
A.4.1. Migration to version 7.1.1 100
A.4.2. Changes 100
A.5. Release 7.1 100
A.5.1. Migration to version 7.1 101
A.5.2. Changes 101
A.6. Release 7.0.3 104
A.6.1. Migration to version 7.0.3 105
A.6.2. Changes 105
A.7. Release 7.0.2 106
A.7.1. Migration to version 7.0.2 106
A.7.2. Changes 106
A.8. Release 7.0.1 106
A.8.1. Migration to version 7.0.1 106
A.8.2. Changes 106
A.9. Release 7.0 107
A.9.1. Migration to version 7.0 107
A.9.2. Changes 108
A.10. Release 6.5.3 114
A.10.1. Migration to version 6.5.3 114
A.10.2. Changes 114
A.11. Release 6.5.2 114
A.11.1. Migration to version 6.5.2 114
A.11.2. Changes 114
A.12. Release 6.5.1 115
A.12.1. Migration to version 6.5.1 115
A.12.2. Changes 115
A.13. Release 6.5 116
A.13.1. Migration to version 6.5 117
A.13.1.1. Multi-Version Concurrency Control 117
A.13.2. Changes 117
A.14. Release 6.4.2 120
A.14.1. Migration to version 6.4.2 120
A.14.2. Changes 121
A.15. Release 6.4.1 121
A.15.1. Migration to version 6.4.1 121
A.15.2. Changes 121
A.16. Release 6.4 122
A.16.1. Migration to version 6.4 122
A.16.2. Changes 122
A.17. Release 6.3.2 126
A.17.1. Changes 126
A.18. Release 6.3.1 127
A.18.1. Changes 127
A.19. Release 6.3 128

A.19.1. Migration to version 6.3 129
A.19.2. Changes 129
A.20. Release 6.2.1 132
A.20.1. Migration from version 6.2 to version 6.2.1 133
A.20.2. Changes 133
A.21. Release 6.2 133
A.21.1. Migration from version 6.1 to version 6.2 134
A.21.2. Migration from version 1.*x* to version 6.2 134
A.21.3. Changes 134
A.22. Release 6.1.1 136
A.22.1. Migration from version 6.1 to version 6.1.1 136
A.22.2. Changes 136
A.23. Release 6.1 137
A.23.1. Migration to version 6.1 137
A.23.2. Changes 137
A.24. Release 6.0 139
A.24.1. Migration from version 1.09 to version 6.0 139
A.24.2. Migration from pre-1.09 to version 6.0 139
A.24.3. Changes 139
A.25. Release 1.09 141
A.26. Release 1.02 142
A.26.1. Migration from version 1.02 to version 1.02.1 142
A.26.2. Dump/Reload Procedure 142
A.26.3. Changes 143
A.27. Release 1.01 143
A.27.1. Migration from version 1.0 to version 1.01 143
A.27.2. Changes 145
A.28. Release 1.0 146
A.28.1. Changes 146
A.29. Postgres95 Release 0.03 147
A.29.1. Changes 147
A.30. Postgres95 Release 0.02 149
A.30.1. Changes 149
A.31. Postgres95 Release 0.01 150
Bibliography 151
Index 153

List of Tables

3-1. Short option key 30
3-2. System V IPC parameters 32
5-1. Character Set Encodings 50
5-2. Client/Server Character Set Encodings 51
10-1. Standard Statistics Views 76
10-2. Statistics Access Functions 78

List of Examples

4-1. An example `pg_hba.conf` file 42
4-2. An example `pg_ident.conf` file 46

Preface

1. What is PostgreSQL?

PostgreSQL is an object-relational database management system (ORDBMS) based on POSTGRES, Version 4.2[1], developed at the University of California at Berkeley Computer Science Department. The POSTGRES project, led by Professor Michael Stonebraker, was sponsored by the Defense Advanced Research Projects Agency (DARPA), the Army Research Office (ARO), the National Science Foundation (NSF), and ESL, Inc.

PostgreSQL is an open-source descendant of this original Berkeley code. It provides SQL92/SQL99 language support and other modern features.

POSTGRES pioneered many of the object-relational concepts now becoming available in some commercial databases. Traditional relational database management systems (RDBMS) 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, this simplicity makes the implementation of certain applications very difficult. PostgreSQL offers substantial additional power by incorporating the following additional concepts in such a way that users can easily extend the system:

- inheritance
- data types
- functions

Other features provide additional power and flexibility:

- constraints
- triggers
- rules
- transactional integrity

These features put PostgreSQL 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 traditional relational database languages. So, although PostgreSQL has some object-oriented features, it is firmly in the relational database world. In fact, some commercial databases have recently incorporated features pioneered by PostgreSQL.

2. A Short History of PostgreSQL

The object-relational database management system now known as PostgreSQL (and briefly called Postgres95) is derived from the POSTGRES package written at the University of California at Berkeley. With over a decade of development behind it, PostgreSQL is the most advanced open-source database available anywhere, offering multiversion concurrency control, supporting almost all SQL

1. http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/postgres.html

constructs (including subselects, transactions, and user-defined types and functions), and having a wide range of language bindings available (including C, C++, Java, Perl, Tcl, and Python).

2.1. The Berkeley POSTGRES Project

Implementation of the POSTGRES DBMS began in 1986. The initial concepts for the system were presented in *The design of POSTGRES* and the definition of the initial data model appeared in *The POSTGRES data model*. The design of the rule system at that time was described in *The design of the POSTGRES rules system*. The rationale and architecture of the storage manager were detailed in *The design of the POSTGRES storage system*.

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. Version 1, described in *The implementation of POSTGRES*, was released to a few external users in June 1989. In response to a critique of the first rule system (*A commentary on the POSTGRES rules system*), the rule system was redesigned (*On Rules, Procedures, Caching and Views in Database Systems*) 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, subsequent releases until Postgres95 (see below) 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 (later merged into Informix[2], which is now owned by IBM[3].) picked up the code and commercialized it. POSTGRES became the primary data manager for the Sequoia 2000[4] scientific computing project in late 1992.

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 Berkeley POSTGRES project officially ended with Version 4.2.

2.2. Postgres95

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

Postgres95 code was completely ANSI C and trimmed in size by 25%. Many internal changes improved performance and maintainability. Postgres95 release 1.0.x ran about 30-50% faster on the Wisconsin Benchmark compared to POSTGRES, Version 4.2. Apart from bug fixes, the following were the major enhancements:

- The query language PostQUEL was replaced with SQL (implemented in the server). Subqueries were not supported until PostgreSQL (see below), but they could be imitated in Postgres95 with user-defined SQL functions. Aggregates were re-implemented. Support for the GROUP BY query clause was also added. The `libpq` interface remained available for C programs.
- In addition to the monitor program, a new program (psql) was provided for interactive SQL queries using GNU Readline.

2. http://www.informix.com/
3. http://www.ibm.com/
4. http://meteora.ucsd.edu/s2k/s2k_home.html

- A new front-end library, `libpgtcl`, supported Tcl-based clients. A sample shell, **pgtclsh**, provided new Tcl commands to interface Tcl programs with the Postgres95 backend.
- The large-object interface was overhauled. The Inversion large objects were the only mechanism for storing large objects. (The Inversion file system was removed.)
- The instance-level rule system was removed. Rules were still available as rewrite rules.
- A short tutorial introducing regular SQL features as well as those of Postgres95 was distributed with the source code
- GNU make (instead of BSD make) was used for the build. Also, Postgres95 could be compiled with an unpatched GCC (data alignment of doubles was fixed).

2.3. PostgreSQL

By 1996, it became clear that the name "Postgres95" would not stand the test of time. We chose a new name, PostgreSQL, to reflect the relationship between the original POSTGRES and the more recent versions with SQL capability. At the same time, we set the version numbering to start at 6.0, putting the numbers back into the sequence originally begun by the Berkeley POSTGRES project.

The emphasis during development of Postgres95 was on identifying and understanding existing problems in the backend code. With PostgreSQL, the emphasis has shifted to augmenting features and capabilities, although work continues in all areas.

Major enhancements in PostgreSQL include:

- Table-level locking has been replaced by multiversion concurrency control, which allows readers to continue reading consistent data during writer activity and enables hot backups from pg_dump while the database stays available for queries.
- 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 coercion, 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 20-40%, and backend start-up time has decreased by 80% since version 6.0 was released.

3. Documentation Resources

This manual set is organized into several parts:

Tutorial

An informal introduction for new users

User's Guide

Documents the SQL query language environment, including data types and functions.

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 server management information

Reference Manual

Reference pages for SQL command syntax and client and server programs

Developer's Guide

Information for PostgreSQL developers. This is intended for those who are contributing to the PostgreSQL project; application development information appears in the *Programmer's Guide*.

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

man pages

The *Reference Manual*'s pages in the traditional Unix man format.

FAQs

Frequently Asked Questions (FAQ) lists document both general issues and some platform-specific issues.

READMEs

README files are available for some contributed packages.

Web Site

The PostgreSQL web site[5] carries details on the latest release, upcoming features, and other information to make your work or play with PostgreSQL more productive.

Mailing Lists

The mailing lists are a good place to have your questions answered, to share experiences with other users, and to contact the developers. Consult the User's Lounge[6] section of the PostgreSQL web site for details.

Yourself!

PostgreSQL is an open-source effort. As such, it depends on the user community for ongoing support. As you begin to use PostgreSQL, 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 them.

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 <pgsql-docs@postgresql.org> mailing list is the place to get going.

5. http://www.postgresql.org
6. http://www.postgresql.org/users-lounge/

4. Terminology and Notation

The terms “PostgreSQL” and “Postgres” will be used interchangeably to refer to the software that accompanies this documentation.

An *administrator* is generally a person who is in charge of installing and running the server. A *user* could be anyone who is using, or wants to use, any part of the PostgreSQL system. These terms should not be interpreted too narrowly; this documentation set does not have fixed presumptions about system administration procedures.

We use `/usr/local/pgsql/` as the root directory of the installation and `/usr/local/pgsql/data` as the directory with the database files. These directories may vary on your site, details can be derived in the *Administrator’s Guide*.

In a command synopsis, brackets (`[` and `]`) indicate an optional phrase or keyword. Anything in braces (`{` and `}`) and containing vertical bars (`|`) indicates that you must choose one alternative.

Examples will show commands executed from various accounts and programs. Commands executed from a Unix shell may be preceded with a dollar sign (“$”). Commands executed from particular user accounts such as root or postgres are specially flagged and explained. SQL commands may be preceded with “`=>`” or will have no leading prompt, depending on the context.

> **Note:** The notation for flagging commands is not universally consistent throughout the documentation set. Please report problems to the documentation mailing list `<pgsql-docs@postgresql.org>`.

5. Bug Reporting Guidelines

When you find a bug in PostgreSQL we want to hear about it. Your bug reports play an important part in making PostgreSQL more reliable because even the utmost care cannot guarantee that every part of PostgreSQL will work on every platform under every circumstance.

The following suggestions are intended to assist you in forming bug reports that can be handled in an effective fashion. No one is required to follow them but it tends to be to everyone’s advantage.

We cannot promise to fix every bug right away. If the bug is obvious, critical, or affects a lot of users, chances are good that someone will look into it. It could also happen that we tell you to update to a newer version to see if the bug happens there. Or we might decide that the bug cannot be fixed before some major rewrite we might be planning is done. Or perhaps it is simply too hard and there are more important things on the agenda. If you need help immediately, consider obtaining a commercial support contract.

5.1. Identifying Bugs

Before you report a bug, please read and re-read the documentation to verify that you can really do whatever it is you are trying. If it is not clear from the documentation whether you can do something or not, please report that too; it is a bug in the documentation. If it turns out that the program does something different from what the documentation says, that is a bug. That might include, but is not limited to, the following circumstances:

- A program terminates with a fatal signal or an operating system error message that would point to a problem in the program. (A counterexample might be a “disk full” message, since you have to fix that yourself.)

- A program produces the wrong output for any given input.
- A program refuses to accept valid input (as defined in the documentation).
- A program accepts invalid input without a notice or error message. But keep in mind that your idea of invalid input might be our idea of an extension or compatibility with traditional practice.
- PostgreSQL fails to compile, build, or install according to the instructions on supported platforms.

Here "program" refers to any executable, not only the backend server.

Being slow or resource-hogging is not necessarily a bug. Read the documentation or ask on one of the mailing lists for help in tuning your applications. Failing to comply to the SQL standard is not necessarily a bug either, unless compliance for the specific feature is explicitly claimed.

Before you continue, check on the TODO list and in the FAQ to see if your bug is already known. If you cannot decode the information on the TODO list, report your problem. The least we can do is make the TODO list clearer.

5.2. What to report

The most important thing to remember about bug reporting is to state all the facts and only facts. Do not speculate what you think went wrong, what "it seemed to do", or which part of the program has a fault. If you are not familiar with the implementation you would probably guess wrong and not help us a bit. And even if you are, educated explanations are a great supplement to but no substitute for facts. If we are going to fix the bug we still have to see it happen for ourselves first. Reporting the bare facts is relatively straightforward (you can probably copy and paste them from the screen) but all too often important details are left out because someone thought it does not matter or the report would be understood anyway.

The following items should be contained in every bug report:

- The exact sequence of steps *from program start-up* necessary to reproduce the problem. This should be self-contained; it is not enough to send in a bare select statement without the preceding create table and insert statements, if the output should depend on the data in the tables. We do not have the time to reverse-engineer your database schema, and if we are supposed to make up our own data we would probably miss the problem. The best format for a test case for query-language related problems is a file that can be run through the psql frontend that shows the problem. (Be sure to not have anything in your `~/.psqlrc` start-up file.) An easy start at this file is to use pg_dump to dump out the table declarations and data needed to set the scene, then add the problem query. You are encouraged to minimize the size of your example, but this is not absolutely necessary. If the bug is reproducible, we will find it either way.

  If your application uses some other client interface, such as PHP, then please try to isolate the offending queries. We will probably not set up a web server to reproduce your problem. In any case remember to provide the exact input files, do not guess that the problem happens for "large files" or "mid-size databases", etc. since this information is too inexact to be of use.

- The output you got. Please do not say that it "didn't work" or "crashed". If there is an error message, show it, even if you do not understand it. If the program terminates with an operating system error, say which. If nothing at all happens, say so. Even if the result of your test case is a program crash or otherwise obvious it might not happen on our platform. The easiest thing is to copy the output from the terminal, if possible.

> **Note:** In case of fatal errors, the error message reported by the client might not contain all the information available. Please also look at the log output of the database server. If you do not keep your server's log output, this would be a good time to start doing so.

- The output you expected is very important to state. If you just write "This command gives me that output." or "This is not what I expected.", we might run it ourselves, scan the output, and think it looks OK and is exactly what we expected. We should not have to spend the time to decode the exact semantics behind your commands. Especially refrain from merely saying that "This is not what SQL says/Oracle does." Digging out the correct behavior from SQL is not a fun undertaking, nor do we all know how all the other relational databases out there behave. (If your problem is a program crash, you can obviously omit this item.)
- Any command line options and other start-up options, including concerned environment variables or configuration files that you changed from the default. Again, be exact. If you are using a prepackaged distribution that starts the database server at boot time, you should try to find out how that is done.
- Anything you did at all differently from the installation instructions.
- The PostgreSQL version. You can run the command `SELECT version();` to find out the version of the server you are connected to. Most executable programs also support a `--version` option; at least `postmaster --version` and `psql --version` should work. If the function or the options do not exist then your version is more than old enough to warrant an upgrade. You can also look into the `README` file in the source directory or at the name of your distribution file or package name. If you run a prepackaged version, such as RPMs, say so, including any subversion the package may have. If you are talking about a CVS snapshot, mention that, including its date and time.

  If your version is older than 7.2 we will almost certainly tell you to upgrade. There are tons of bug fixes in each new release, that is why we make new releases.

- Platform information. This includes the kernel name and version, C library, processor, memory information. In most cases it is sufficient to report the vendor and version, but do not assume everyone knows what exactly "Debian" contains or that everyone runs on Pentiums. If you have installation problems then information about compilers, make, etc. is also necessary.

Do not be afraid if your bug report becomes rather lengthy. That is a fact of life. It is better to report everything the first time than us having to squeeze the facts out of you. On the other hand, if your input files are huge, it is fair to ask first whether somebody is interested in looking into it.

Do not spend all your time to figure out which changes in the input make the problem go away. This will probably not help solving it. If it turns out that the bug cannot be fixed right away, you will still have time to find and share your work-around. Also, once again, do not waste your time guessing why the bug exists. We will find that out soon enough.

When writing a bug report, please choose non-confusing terminology. The software package in total is called "PostgreSQL", sometimes "Postgres" for short. If you are specifically talking about the backend server, mention that, do not just say "PostgreSQL crashes". A crash of a single backend server process is quite different from crash of the parent "postmaster" process; please don't say "the postmaster crashed" when you mean a single backend went down, nor vice versa. Also, client programs such as the interactive frontend "psql" are completely separate from the backend. Please try to be specific about whether the problem is on the client or server side.

5.3. Where to report bugs

In general, send bug reports to the bug report mailing list at <pgsql-bugs@postgresql.org>. You are requested to use a descriptive subject for your email message, perhaps parts of the error message.

Another method is to fill in the bug report web-form available at the project's web site http://www.postgresql.org/. Entering a bug report this way causes it to be mailed to the <pgsql-bugs@postgresql.org> mailing list.

Do not send bug reports to any of the user mailing lists, such as <pgsql-sql@postgresql.org> or <pgsql-general@postgresql.org>. These mailing lists are for answering user questions and their subscribers normally do not wish to receive bug reports. More importantly, they are unlikely to fix them.

Also, please do *not* send reports to the developers' mailing list <pgsql-hackers@postgresql.org>. This list is for discussing the development of PostgreSQL and it would be nice if we could keep the bug reports separate. We might choose to take up a discussion about your bug report on pgsql-hackers, if the problem needs more review.

If you have a problem with the documentation, the best place to report it is the documentation mailing list <pgsql-docs@postgresql.org>. Please be specific about what part of the documentation you are unhappy with.

If your bug is a portability problem on a non-supported platform, send mail to <pgsql-ports@postgresql.org>, so we (and you) can work on porting PostgreSQL to your platform.

> **Note:** Due to the unfortunate amount of spam going around, all of the above email addresses are closed mailing lists. That is, you need to be subscribed to a list to be allowed to post on it. (You need not be subscribed to use the bug report web-form, however.) If you would like to send mail but do not want to receive list traffic, you can subscribe and set your subscription option to nomail. For more information send mail to <majordomo@postgresql.org> with the single word help in the body of the message.

6. Y2K Statement

> **Author:** Written by Thomas Lockhart (<lockhart@fourpalms.org>) on 1998-10-22. Updated 2000-03-31.

The PostgreSQL Global Development Group provides the PostgreSQL software code tree as a public service, without warranty and without liability for its behavior or performance. However, at the time of writing:

- The author of this statement, a volunteer on the PostgreSQL support team since November, 1996, is not aware of any problems in the PostgreSQL code base related to time transitions around Jan 1, 2000 (Y2K).
- The author of this statement is not aware of any reports of Y2K problems uncovered in regression testing or in other field use of recent or current versions of PostgreSQL. We might have expected to hear about problems if they existed, given the installed base and the active participation of users on the support mailing lists.

- To the best of the author's knowledge, the assumptions PostgreSQL makes about dates specified with a two-digit year are documented in the current *User's Guide* in the chapter on data types. For two-digit years, the significant transition year is 1970, not 2000; e.g. `70-01-01` is interpreted as 1970-01-01, whereas `69-01-01` is interpreted as 2069-01-01.
- Any Y2K problems in the underlying OS related to obtaining the "current time" may propagate into apparent Y2K problems in PostgreSQL.

Refer to The GNU Project[8] and The Perl Institute[9] for further discussion of Y2K issues, particularly as it relates to open source, no fee software.

8. http://www.gnu.org/software/year2000.html
9. http://language.perl.com/news/y2k.html

Chapter 1. Installation Instructions

1.1. Short Version

```
./configure
gmake
su
gmake install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data >logfile 2>&1 &
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

The long version is the rest of this chapter.

1.2. Requirements

In general, a modern Unix-compatible platform should be able to run PostgreSQL. The platforms that had received specific testing at the time of release are listed in Section 1.7 below. In the `doc` subdirectory of the distribution there are several platform-specific FAQ documents you might wish to consult if you are having trouble.

The following prerequisites exist for building PostgreSQL:

- GNU make is required; other make programs will *not* work. GNU make is often installed under the name `gmake`; this document will always refer to it by that name. (On some systems GNU make is the default tool with the name `make`.) To test for GNU make enter

  `gmake --version`

  It is recommended to use version 3.76.1 or later.
- You need an ISO/ANSI C compiler. Recent versions of GCC are recommendable, but PostgreSQL is known to build with a wide variety of compilers from different vendors.
- gzip is needed to unpack the distribution in the first place. If you are reading this, you probably already got past that hurdle.
- The GNU Readline library (for comfortable line editing and command history retrieval) will automatically be used if found. You might wish to install it before proceeding, but it is not essential. (On NetBSD, the `libedit` library is readline-compatible and is used if `libreadline` is not found.)
- GNU Flex and Bison are needed to build from scratch, but they are *not* required when building from a released source package because pre-generated output files are included in released packages. You will need these programs only when building from a CVS tree or if you changed the actual scanner and parser definition files. If you need them, be sure to get Flex 2.5.4 or later and Bison 1.28 or later. Other yacc programs can sometimes be used, but doing so requires extra effort and is not recommended. Other lex programs will definitely not work.
- To build on Windows NT or Windows 2000 you need the Cygwin and cygipc packages. See the file `doc/FAQ_MSWIN` for details.

If you need to get a GNU package, you can find it at your local GNU mirror site (see http://www.gnu.org/order/ftp.html for a list) or at ftp://ftp.gnu.org/gnu/.

Also check that you have sufficient disk space. You will need about 30 MB for the source tree during compilation and about 10 MB for the installation directory. An empty database cluster takes about 20 MB, databases take about five times the amount of space that a flat text file with the same data would take. If you are going to run the regression tests you will temporarily need an extra 20 MB. Use the **df** command to check for disk space.

1.3. Getting The Source

The PostgreSQL 7.2 sources can be obtained by anonymous FTP from ftp://ftp.postgresql.org/pub/postgresql-7.2.tar.gz. Use a mirror if possible. After you have obtained the file, unpack it:

```
gunzip postgresql-7.2.tar.gz
tar xf postgresql-7.2.tar
```

This will create a directory `postgresql-7.2` under the current directory with the PostgreSQL sources. Change into that directory for the rest of the installation procedure.

1.4. If You Are Upgrading

The internal data storage format changes with new releases of PostgreSQL. Therefore, if you are upgrading an existing installation that does not have a version number “7.2.x”, you must back up and restore your data as shown here. These instructions assume that your existing installation is under the `/usr/local/pgsql` directory, and that the data area is in `/usr/local/pgsql/data`. Substitute your paths appropriately.

1. Make sure that your database is not updated during or after the backup. This does not affect the integrity of the backup, but the changed data would of course not be included. If necessary, edit the permissions in the file `/usr/local/pgsql/data/pg_hba.conf` (or equivalent) to disallow access from everyone except you.

2. To dump your database installation, type:

   ```
   pg_dumpall > outputfile
   ```

   If you need to preserve OIDs (such as when using them as foreign keys), then use the `-o` option when running **pg_dumpall**.

   pg_dumpall does not save large objects. Check Section 9.1.4 if you need to do this.

   Make sure that you use the **pg_dumpall** command from the version you are currently running. 7.2’s **pg_dumpall** should not be used on older databases.

3. If you are installing the new version at the same location as the old one then shut down the old server, at the latest before you install the new files:

   ```
   kill -INT `cat /usr/local/pgsql/data/postmaster.pid`
   ```

   Versions prior to 7.0 do not have this `postmaster.pid` file. If you are using such a version you must find out the process id of the server yourself, for example by typing **`ps ax | grep postmaster`**, and supply it to the **kill** command.

On systems that have PostgreSQL started at boot time, there is probably a start-up file that will accomplish the same thing. For example, on a Red Hat Linux system one might find that

```
/etc/rc.d/init.d/postgresql stop
```

works. Another possibility is **pg_ctl stop**.

4. If you are installing in the same place as the old version then it is also a good idea to move the old installation out of the way, in case you have trouble and need to revert to it. Use a command like this:

```
mv /usr/local/pgsql /usr/local/pgsql.old
```

After you have installed PostgreSQL 7.2, create a new database directory and start the new server. Remember that you must execute these commands while logged in to the special database user account (which you already have if you are upgrading).

```
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data
```

Finally, restore your data with

```
/usr/local/pgsql/bin/psql -d template1 -f outputfile
```

using the *new* psql.

You can also install the new version in parallel with the old one to decrease the downtime. These topics are discussed at length in Section 9.3, which you are encouraged to read in any case.

1.5. Installation Procedure

1. Configuration

   The first step of the installation procedure is to configure the source tree for your system and choose the options you would like. This is done by running the `configure` script. For a default installation simply enter

   ```
   ./configure
   ```

   This script will run a number of tests to guess values for various system dependent variables and detect some quirks of your operating system, and finally will create several files in the build tree to record what it found.

   The default configuration will build the server and utilities, as well as all client applications and interfaces that require only a C compiler. All files will be installed under `/usr/local/pgsql` by default.

   You can customize the build and installation process by supplying one or more of the following command line options to `configure`:

   `--prefix=`*`PREFIX`*

   Install all files under the directory *`PREFIX`* instead of `/usr/local/pgsql`. The actual files will be installed into various subdirectories; no files will ever be installed directly into the *`PREFIX`* directory.

   If you have special needs, you can also customize the individual subdirectories with the following options.

`--exec-prefix=`*`EXEC-PREFIX`*

You can install architecture-dependent files under a different prefix, *`EXEC-PREFIX`*, than what *`PREFIX`* was set to. This can be useful to share architecture-independent files between hosts. If you omit this, then *`EXEC-PREFIX`* is set equal to *`PREFIX`* and both architecture-dependent and independent files will be installed under the same tree, which is probably what you want.

`--bindir=`*`DIRECTORY`*

Specifies the directory for executable programs. The default is *`EXEC-PREFIX`*`/bin`, which normally means `/usr/local/pgsql/bin`.

`--datadir=`*`DIRECTORY`*

Sets the directory for read-only data files used by the installed programs. The default is *`PREFIX`*`/share`. Note that this has nothing to do with where your database files will be placed.

`--sysconfdir=`*`DIRECTORY`*

The directory for various configuration files, *`PREFIX`*`/etc` by default.

`--libdir=`*`DIRECTORY`*

The location to install libraries and dynamically loadable modules. The default is *`EXEC-PREFIX`*`/lib`.

`--includedir=`*`DIRECTORY`*

The directory for installing C and C++ header files. The default is *`PREFIX`*`/include`.

`--docdir=`*`DIRECTORY`*

Documentation files, except "man" pages, will be installed into this directory. The default is *`PREFIX`*`/doc`.

`--mandir=`*`DIRECTORY`*

The man pages that come with PostgreSQL will be installed under this directory, in their respective `man`*`x`* subdirectories. The default is *`PREFIX`*`/man`.

Note: Care has been taken to make it possible to install PostgreSQL into shared installation locations (such as `/usr/local/include`) without interfering with the namespace of the rest of the system. First, the string "`/postgresql`" is automatically appended to `datadir`, `sysconfdir`, and `docdir`, unless the fully expanded directory name already contains the string "`postgres`" or "`pgsql`". For example, if you choose `/usr/local` as prefix, the documentation will be installed in `/usr/local/doc/postgresql`, but if the prefix is `/opt/postgres`, then it will be in `/opt/postgres/doc`. Second, the installation layout of the C and C++ header files has been reorganized in the 7.2 release. The public header files of the client interfaces are installed into `includedir` and are namespace-clean. The internal header files and the server header files are installed into private directories under `includedir`. See the *Programmer's Guide* for information about how to get at the header files for each interface. Finally, a private subdirectory will also be created, if appropriate, under `libdir` for dynamically loadable modules.

`--with-includes=`*`DIRECTORIES`*

`DIRECTORIES` is a colon-separated list of directories that will be added to the list the compiler searches for header files. If you have optional packages (such as GNU Readline) installed in a non-standard location, you have to use this option and probably also the corresponding `--with-libraries` option.

Example: `--with-includes=/opt/gnu/include:/usr/sup/include`.

`--with-libraries=`*`DIRECTORIES`*

`DIRECTORIES` is a colon-separated list of directories to search for libraries. You will probably have to use this option (and the corresponding `--with-includes` option) if you have packages installed in non-standard locations.

Example: `--with-libraries=/opt/gnu/lib:/usr/sup/lib`.

`--enable-locale`

Enables locale support. There is a performance penalty associated with locale support, but if you are not in an English-speaking environment you will most likely need this.

`--enable-recode`

Enables single-byte character set recode support. See Section 5.3 about this feature.

`--enable-multibyte`

Allows the use of multibyte character encodings (including Unicode) and character set encoding conversion. Read Section 5.2 for details.

Note that some interfaces (such as Tcl or Java) expect all character strings to be in Unicode, so this option will be required to correctly support these interfaces.

`--enable-nls[=`*`LANGUAGES`*`]`

Enables Native Language Support (NLS), that is, the ability to display a program's messages in a language other than English. *`LANGUAGES`* is a space separated list of codes of the languages that you want supported, for example `--enable-nls='de fr'`. (The intersection between your list and the set of actually provided translations will be computed automatically.) If you do not specify a list, then all available translations are installed.

To use this option, you will need an implementation of the gettext API. Some operating systems have this built-in (e.g., Linux, NetBSD, Solaris), for other systems you can download an add-on package from here: http://www.postgresql.org/~petere/gettext.html. If you are using the gettext implementation in the GNU C library then you will additionally need the GNU gettext package for some utility programs. For any of the other implementations you will not need it.

`--with-pgport=`*`NUMBER`*

Set *`NUMBER`* as the default port number for server and clients. The default is 5432. The port can always be changed later on, but if you specify it here then both server and clients will have the same default compiled in, which can be very convenient. Usually the only good reason to select a non-default value is if you intend to run multiple PostgreSQL servers on the same machine.

`--with-CXX`

Build the C++ interface library.

`--with-perl`

Build the Perl interface module. The Perl interface will be installed at the usual place for Perl modules (typically under `/usr/lib/perl`), so you must have root access to perform the installation step (see step 4). You need to have Perl 5 installed to use this option.

`--with-python`

Build the Python interface module. You need to have root access to be able to install the Python module at its default place (`/usr/lib/python`*`x.y`*). To be able to use this option, you must have Python installed and your system needs to support shared libraries. If you instead want to build a new complete interpreter binary, you will have to do it manually.

`--with-tcl`

Builds components that require Tcl/Tk, which are libpgtcl, pgtclsh, pgtksh, PgAccess, and PL/Tcl. But see below about `--without-tk`.

`--without-tk`

If you specify `--with-tcl` and this option, then programs that require Tk (pgtksh and PgAccess) will be excluded.

`--with-tclconfig=`*`DIRECTORY`*
`--with-tkconfig=`*`DIRECTORY`*

Tcl/Tk installs the files `tclConfig.sh` and `tkConfig.sh`, which contain configuration information needed to build modules interfacing to Tcl or Tk. These files are normally found automatically at their well-known locations, but if you want to use a different version of Tcl or Tk you can specify the directory in which to find them.

`--enable-odbc`

Build the ODBC driver. By default, the driver will be independent of a driver manager. To work better with a driver manager already installed on your system, use one of the following options in addition to this one. More information can be found in the *Programmer's Guide*.

`--with-iodbc`

Build the ODBC driver for use with iODBC.

`--with-unixodbc`

Build the ODBC driver for use with unixODBC.

`--with-odbcinst=`*`DIRECTORY`*

Specifies the directory where the ODBC driver will expect its `odbcinst.ini` configuration file. The default is `/usr/local/pgsql/etc` or whatever you specified as `--sysconfdir`. It should be arranged that the driver reads the same file as the driver manager.

If either the option `--with-iodbc` or the option `--with-unixodbc` is used, this option will be ignored because in that case the driver manager handles the location of the configuration file.

`--with-java`

Build the JDBC driver and associated Java packages. This option requires Ant to be installed (as well as a JDK, of course). Refer to the JDBC driver documentation in the *Programmer's Guide* for more information.

`--with-krb4[=`*`DIRECTORY`*`]`
`--with-krb5[=`*`DIRECTORY`*`]`

Build with support for Kerberos authentication. You can use either Kerberos version 4 or 5, but not both. The *`DIRECTORY`* argument specifies the root directory of the Kerberos installation; `/usr/athena` is assumed as default. If the relevant header files and libraries are not under a common parent directory, then you must use the `--with-includes` and `--with-libraries` options in addition to this option. If, on the other hand, the required files are in a location that is searched by default (e.g., `/usr/lib`), then you can leave off the argument.

`configure` will check for the required header files and libraries to make sure that your Kerberos installation is sufficient before proceeding.

`--with-krb-srvnam=`*`NAME`*

The name of the Kerberos service principal. `postgres` is the default. There's probably no reason to change this.

`--with-openssl[=`*`DIRECTORY`*`]`

Build with support for SSL (encrypted) connections. This requires the OpenSSL package to be installed. The *`DIRECTORY`* argument specifies the root directory of the OpenSSL installation; the default is `/usr/local/ssl`.

`configure` will check for the required header files and libraries to make sure that your OpenSSL installation is sufficient before proceeding.

`--with-pam`

Build with PAM (Pluggable Authentication Modules) support.

`--enable-syslog`

Enables the PostgreSQL server to use the syslog logging facility. (Using this option does not mean that you must log with syslog or even that it will be done by default, it simply makes it possible to turn that option on at run time.)

`--enable-debug`

Compiles all programs and libraries with debugging symbols. This means that you can run the programs through a debugger to analyze problems. This enlarges the size of the installed executables considerably, and on non-GCC compilers it usually also disables compiler optimization, causing slowdowns. However, having the symbols available is extremely helpful for dealing with any problems that may arise. Currently, this option is recommended for production installations only if you use GCC. But you should always have it on if you are doing development work or running a beta version.

`--enable-cassert`

Enables *assertion* checks in the server, which test for many "can't happen" conditions. This is invaluable for code development purposes, but the tests slow things down a little. Also, having the tests turned on won't necessarily enhance the stability of your server! The assertion checks are not categorized for severity, and so what might be a relatively harmless bug will still lead to server restarts if it triggers an assertion failure. Currently, this option is not recommended for production use, but you should have it on for development work or when running a beta version.

`--enable-depend`

Enables automatic dependency tracking. With this option, the makefiles are set up so that all affected object files will be rebuilt when any header file is changed. This is useful if you

are doing development work, but is just wasted overhead if you intend only to compile once and install. At present, this option will work only if you use GCC.

If you prefer a C or C++ compiler different from the one `configure` picks then you can set the environment variables `CC` or `CXX`, respectively, to the program of your choice. Similarly, you can override the default compiler flags with the `CFLAGS` and `CXXFLAGS` variables. For example:

```
env CC=/opt/bin/gcc CFLAGS='-O2 -pipe' ./configure
```

2. Build

   To start the build, type

   ```
   gmake
   ```

   (Remember to use GNU make.) The build may take anywhere from 5 minutes to half an hour depending on your hardware. The last line displayed should be

   ```
   All of PostgreSQL is successfully made. Ready to install.
   ```

3. Regression Tests

   If you want to test the newly built server before you install it, you can run the regression tests at this point. The regression tests are a test suite to verify that PostgreSQL runs on your machine in the way the developers expected it to. Type

   ```
   gmake check
   ```

   (This won't work as root; do it as an unprivileged user.) It is possible that some tests fail, due to differences in error message wording or floating point results. Chapter 14 contains detailed information about interpreting the test results. You can repeat this test at any later time by issuing the same command.

4. Installing The Files

   Note: If you are upgrading an existing system and are going to install the new files over the old ones, then you should have backed up your data and shut down the old server by now, as explained in Section 1.4 above.

   To install PostgreSQL enter

   ```
   gmake install
   ```

   This will install files into the directories that were specified in step 1. Make sure that you have appropriate permissions to write into that area. Normally you need to do this step as root. Alternatively, you could create the target directories in advance and arrange for appropriate permissions to be granted.

   If you built the Perl or Python interfaces and you were not the root user when you executed the above command then that part of the installation probably failed. In that case you should become the root user and then do

   ```
   gmake -C src/interfaces/perl5 install
   gmake -C src/interfaces/python install
   ```

If you do not have superuser access you are on your own: you can still take the required files and place them in other directories where Perl or Python can find them, but how to do that is left as an exercise.

The standard installation provides only the header files needed for client application development. If you plan to do any server-side program development (such as custom functions or data types written in C), then you may want to install the entire PostgreSQL include tree into your target include directory. To do that, enter

```
gmake install-all-headers
```

This adds a megabyte or two to the installation footprint, and is only useful if you don't plan to keep the whole source tree around for reference. (If you do, you can just use the source's include directory when building server-side software.)

Client-only installation: If you want to install only the client applications and interface libraries, then you can use these commands:

```
gmake -C src/bin install
gmake -C src/include install
gmake -C src/interfaces install
gmake -C doc install
```

To undo the installation use the command **gmake uninstall**. However, this will not remove any created directories.

After the installation you can make room by removing the built files from the source tree with the **gmake clean** command. This will preserve the files made by the configure program, so that you can rebuild everything with **gmake** later on. To reset the source tree to the state in which it was distributed, use **gmake distclean**. If you are going to build for several platforms from the same source tree you must do this and re-configure for each build.

If you perform a build and then discover that your configure options were wrong, or if you change anything that configure investigates (for example, you install GNU Readline), then it's a good idea to do **gmake distclean** before reconfiguring and rebuilding. Without this, your changes in configuration choices may not propagate everywhere they need to.

1.6. Post-Installation Setup

1.6.1. Shared Libraries

On some systems that have shared libraries (which most systems do) you need to tell your system how to find the newly installed shared libraries. The systems on which this is *not* necessary include BSD/OS, FreeBSD, HP-UX, IRIX, Linux, NetBSD, OpenBSD, Tru64 UNIX (formerly Digital UNIX), and Solaris.

The method to set the shared library search path varies between platforms, but the most widely usable method is to set the environment variable `LD_LIBRARY_PATH` like so: In Bourne shells (**sh**, **ksh**, **bash**, **zsh**)

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

or in **csh** or **tcsh**

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

Replace `/usr/local/pgsql/lib` with whatever you set `--libdir` to in step 1. You should put these commands into a shell start-up file such as `/etc/profile` or `~/.bash_profile`. Some good information about the caveats associated with this method can be found at http://www.visi.com/~barr/ldpath.html.

On some systems it might be preferable to set the environment variable `LD_RUN_PATH` *before* building.

If in doubt, refer to the manual pages of your system (perhaps **ld.so** or **rld**). If you later on get a message like

```
psql: error in loading shared libraries
libpq.so.2.1: cannot open shared object file: No such file or directory
```

then this step was necessary. Simply take care of it then.

If you are on BSD/OS, Linux, or SunOS 4 and you have root access you can run

```
/sbin/ldconfig /usr/local/pgsql/lib
```

(or equivalent directory) after installation to enable the run-time linker to find the shared libraries faster. Refer to the manual page of **ldconfig** for more information. On FreeBSD, NetBSD, and OpenBSD the command is

```
/sbin/ldconfig -m /usr/local/pgsql/lib
```

instead. Other systems are not known to have an equivalent command.

1.6.2. Environment Variables

If you installed into `/usr/local/pgsql` or some other location that is not searched for programs by default, you need to add `/usr/local/pgsql/bin` (or whatever you set `--bindir` to in step 1) into your `PATH`. To do this, add the following to your shell start-up file, such as `~/.bash_profile` (or `/etc/profile`, if you want it to affect every user):

```
PATH=/usr/local/pgsql/bin:$PATH
```

If you are using **csh** or **tcsh**, then use this command:

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

To enable your system to find the man documentation, you need to add a line like the following to a shell start-up file:

```
MANPATH=/usr/local/pgsql/man:$MANPATH
```

The environment variables `PGHOST` and `PGPORT` specify to client applications the host and port of the database server, overriding the compiled-in defaults. If you are going to run client applications remotely then it is convenient if every user that plans to use the database sets `PGHOST`. This is not required, however: the settings can be communicated via command line options to most client programs.

1.7. Supported Platforms

PostgreSQL has been verified by the developer community to work on the platforms listed below. A supported platform generally means that PostgreSQL builds and installs according to these instructions and that the regression tests pass.

Note: If you are having problems with the installation on a supported platform, please write to <pgsql-bugs@postgresql.org> or <pgsql-ports@postgresql.org>, not to the people listed here.

OS	Processor	Version	Reported	Remarks
AIX	RS6000	7.2	2001-12-19, Andreas Zeugswetter (<ZeugswetterA@spardat.at>), Tatsuo Ishii (<t-ishii@sra.co.jp>)	see also doc/FAQ_AIX
BeOS	x86	7.2	2001-11-29, Cyril Velter (<cyril.velter@libertysurf.fr>)	5.0.4
BSD/OS	x86	7.2	2001-11-27, Bruce Momjian (<pgman@candle.pha.pa.us>)	4.2
FreeBSD	Alpha	7.2	2001-12-18, Chris Kings-Lynne (<chriskl@familyhealth.com.au>)	
FreeBSD	x86	7.2	2001-11-14, Chris Kings-Lynne (<chriskl@familyhealth.com.au>)	
HP-UX	PA-RISC	7.2	2001-11-29, Joseph Conway (<Joseph.Conway@home.com>), Tom Lane (<tgl@sss.pgh.pa.us>)	11.00 and 10.20; see also doc/FAQ_HPUX
IRIX	MIPS	7.2	2001-11-28, Luis Amigo (<lamigo@atc.unican.es>)	6.5.13, MIPSPro 7.30
Linux	Alpha	7.2	2001-11-16, Tom Lane (<tgl@sss.pgh.pa.us>)	2.2.18; tested at SourceForge

OS	Processor	Version	Reported	Remarks
Linux	armv4l	7.2	2001-12-10, Mark Knox (<segfault@hardline.org>)	2.2.x
Linux	MIPS	7.2	2001-11-15, Hisao Shibuya (<shibuya@alpha.or.jp>)	2.0.x; Cobalt Qube2
Linux	PlayStation 2	7.2	2001-12-12, Permaine Cheung <pcheung@redhat.com>)	#undef HAS_TEST_AND_SET, lock_t
Linux	PPC74xx	7.2	2001-11-16, Tom Lane (<tgl@sss.pgh.pa.us>)	2.2.18; Apple G3
Linux	S/390	7.2	2001-12-12, Permaine Cheung <pcheung@redhat.com>)	
Linux	Sparc	7.2	2001-11-28, Doug McNaught (<doug@wireboard.com>)	2.2.19
Linux	x86	7.2	2001-11-15, Thomas Lockhart (<lockhart@fourpalms.org>)	2.0.x, 2.2.x, 2.4.x
MacOS X	PPC	7.2	2001-11-28, Gavin Sherry (<swm@linuxworld.com.au>)	10.1.x
NetBSD	Alpha	7.2	2001-11-20, Thomas Thai (<tom@minnesota.com>)	1.5W
NetBSD	arm32	7.1	2001-03-21, Patrick Welche (<prlw1@cam.ac.uk>)	1.5E
NetBSD	m68k	7.0	2000-04-10, Henry B. Hotz (<hotz@jpl.nasa.gov>)	Mac 8xx
NetBSD	PPC	7.2	2001-11-28, Bill Studenmund (<wrstuden@netbsd.org>)	1.5

OS	Processor	Version	Reported	Remarks
NetBSD	Sparc	7.2	2001-12-03, Matthew Green (<mrg@eterna.com.au>)	32- and 64-bit builds
NetBSD	VAX	7.1	2001-03-30, Tom I. Helbekkmo (<tih@kpnQwest.no>)	1.5
NetBSD	x86	7.2	2001-11-28, Bill Studenmund (<wrstuden@netbsd.org>)	1.5
OpenBSD	Sparc	7.2	2001-11-27, Brandon Palmer (<bpalmer@crimelabs.net>)	3.0
OpenBSD	x86	7.2	2001-11-26, Brandon Palmer (<bpalmer@crimelabs.net>)	3.0
Open UNIX	x86	7.2	2001-11-28, OU-8 Larry Rosenman (<ler@lerctr.org>), UW-7 Olivier Prenant (<ohp@pyrenet.fr>)	see also doc/FAQ_SCO
QNX 4 RTOS	x86	7.2	2001-12-10, Bernd Tegge (<tegge@repas-aeg.de>)	4.25; see also doc/FAQ_QNX4
Solaris	Sparc	7.2	2001-11-12, Andrew Sullivan (<andrew@libertyrms.com>)	2.6-8; see also doc/FAQ_Solaris
Solaris	x86	7.2	2001-11-28, Martin Renters (<martin@datafax.com>)	2.8; see also doc/FAQ_Solaris
SunOS 4	Sparc	7.2	2001-12-04, Tatsuo Ishii (<t-ishii@sra.co.jp>)	

OS	Processor	Version	Reported	Remarks
Tru64 UNIX	Alpha	7.2	2001-11-26, Alessio Bragadini (<alessio@albourne.com>), Bernd Tegge (<tegge@repas-aeg.de>)	5.0; 4.0g with cc and gcc
Windows	x86	7.2	2001-12-13, Dave Page (<dpage@vale-housing.co.uk>), Jason Tishler (<jason@tishler.net>)	with Cygwin; see doc/FAQ_MSWIN
Windows	x86	7.2	2001-12-10, Dave Page (<dpage@vale-housing.co.uk>)	native is client-side only; see Chapter 2

Unsupported Platforms: The following platforms are either known not to work, or they used to work in a previous release and we did not receive explicit confirmation of a successful test with version 7.2 at the time this list was compiled. We include these here to let you know that these platforms *could* be supported if given some attention.

OS	Processor	Version	Reported	Remarks
DG/UX 5.4R4.11	m88k	6.3	1998-03-01, Brian E Gallew (<geek+@cmu.edu>)	no recent reports
MkLinux DR1	PPC750	7.0	2001-04-03, Tatsuo Ishii (<t-ishii@sra.co.jp>)	7.1 needs OS update?
NeXTSTEP	x86	6.x	1998-03-01, David Wetzel (<dave@turbocat.de>)	bit rot suspected
QNX RTOS v6	x86	7.2	2001-11-20, Igor Kovalenko (<Igor.Kovalenko@motorola.com>)	patches available in archives, but too far for 7.2
SCO OpenServer 5	x86	6.5	1999-05-25, Andrew Merrill (<andrew@compclass.com>)	7.2 should work, but no reports; see also doc/FAQ_SCO

OS	Processor	Version	Reported	Remarks
System V R4	m88k	6.2.1	1998-03-01, Doug Winterburn (<dlw@seavme.xroads.com>)	needs new TAS spinlock code
System V R4	MIPS	6.4	1998-10-28, Frank Ridderbusch (<ridderbusch.pad@sni.de>)	no recent reports
Ultrix	MIPS	7.1	2001-03-26	TAS spinlock code not detected
Ultrix	VAX	6.x	1998-03-01	

Chapter 2. Installation on Windows

Build, installation, and use instructions for PostgreSQL client libraries on Windows

Although PostgreSQL is written for Unix-like operating systems, the C client library (libpq) and the interactive terminal (psql) can be compiled natively under Windows. The makefiles included in the source distribution are written for Microsoft Visual C++ and will probably not work with other systems. It should be possible to compile the libraries manually in other cases.

> **Tip:** If you are using Windows 98 or newer you can build and use all of PostgreSQL "the Unix way" if you install the Cygwin toolkit first. In that case see Chapter 1.

To build everything that you can on Windows, change into the `src` directory and type the command

```
nmake /f win32.mak
```

This assumes that you have Visual C++ in your path.

The following files will be built:

`interfaces\libpq\Release\libpq.dll`

The dynamically linkable frontend library

`interfaces\libpq\Release\libpqdll.lib`

Import library to link your program to `libpq.dll`

`interfaces\libpq\Release\libpq.lib`

Static library version of the frontend library

`bin\psql\Release\psql.exe`

The PostgreSQL interactive terminal

The only file that really needs to be installed is the `libpq.dll` library. This file should in most cases be placed in the `WINNT\SYSTEM32` directory (or in `WINDOWS\SYSTEM` on a Windows 95/98/ME system). If this file is installed using a setup program, it should be installed with version checking using the `VERSIONINFO` resource included in the file, to ensure that a newer version of the library is not overwritten.

If you plan to do development using libpq on this machine, you will have to add the `src\include` and `src\interfaces\libpq` subdirectories of the source tree to the include path in your compilers settings.

To use the libraries, you must add the `libpqdll.lib` file to your project. (In Visual C++, just right-click on the project and choose to add it.)

Chapter 3. Server Runtime Environment

This chapter discusses how to set up and run the database server and the interactions with the operating system.

3.1. The PostgreSQL user account

As with any other server daemon that is connected to the world at large, it is advisable to run PostgreSQL under a separate user account. This user account should only own the data itself that is being managed by the server, and should not be shared with other daemons. (Thus, using the user “nobody” is a bad idea.) It is not advisable to install the executables as owned by this user account because that runs the risk of user-defined functions gone astray or any other exploits compromising the executable programs.

To add a user account to your system, look for a command **useradd** or **adduser**. The user name postgres is often used but by no means required.

3.2. Creating a database cluster

Before you can do anything, you must initialize a database storage area on disk. We call this a *database cluster*. (SQL speaks of a catalog cluster instead.) A database cluster is a collection of databases that will be accessible through a single instance of a running database server. After initialization, a database cluster will contain one database named `template1`. As the name suggests, this will be used as a template for subsequently created databases; it should not be used for actual work.

In file system terms, a database cluster will be a single directory under which all data will be stored. We call this the *data directory* or *data area*. It is completely up to you where you choose to store your data. There is no default, although locations such as `/usr/local/pgsql/data` or `/var/lib/pgsql/data` are popular. To initialize a database cluster, use the command **initdb**, which is installed with PostgreSQL. The desired file system location of your database system is indicated by the `-D` option, for example

```
$ initdb -D /usr/local/pgsql/data
```

Note that you must execute this command while being logged into the PostgreSQL user account, which is described in the previous section.

> **Tip:** As an alternative to the `-D` option, you can set the environment variable `PGDATA`.

initdb will attempt to create the directory you specify if it does not already exist. It is likely that it won’t have the permission to do so (if you followed our advice and created an unprivileged account). In that case you should create the directory yourself (as root) and transfer ownership of it to the PostgreSQL user account. Here is how this might work:

```
root# mkdir /usr/local/pgsql/data
root# chown postgres /usr/local/pgsql/data
root# su postgres
postgres$ initdb -D /usr/local/pgsql/data
```

initdb will refuse to run if the data directory looks like it belongs to an already initialized installation.

Because the data directory contains all the data stored in the database, it is essential that it be well secured from unauthorized access. **initdb** therefore revokes access permissions from everyone but the PostgreSQL user account.

However, while the directory contents are secure, the default `pg_hba.conf` authentication method of `trust` allows any local user to connect to the database and even become the database superuser. If you don't trust other local users, we recommend you use **initdb**'s option `-W` or `--pwprompt` to assign a password to the database superuser. After **initdb**, modify `pg_hba.conf` to use `md5` or `password`, instead of `trust`, authentication *before* you start the server for the first time. (Other, possibly more convenient approaches include using `ident` authentication or file system permissions to restrict connections. See Chapter 4 for more information.)

One surprise you might encounter while running **initdb** is a notice similar to this one:

```
NOTICE:  Initializing database with en_US collation order.
        This locale setting will prevent use of index optimization for
        LIKE and regexp searches.  If you are concerned about speed of
        such queries, you may wish to set LC_COLLATE to "C" and
        re-initdb.  For more information see the Administrator's Guide.
```

This notice is intended to warn you that the currently selected locale will cause indexes to be sorted in an order that prevents them from being used for LIKE and regular-expression searches. If you need good performance of such searches, you should set your current locale to `C` and re-run **initdb**. On most systems, setting the current locale is done by changing the value of the environment variable `LC_ALL` or `LANG`. The sort order used within a particular database cluster is set by **initdb** and cannot be changed later, short of dumping all data, rerunning **initdb**, and reloading the data. So it's important to make this choice correctly now.

3.3. Starting the database server

Before anyone can access the database you must start the database server. The database server is called *postmaster*. The postmaster must know where to find the data it is supposed to work on. This is done with the `-D` option. Thus, the simplest way to start the server is, for example,

```
$ postmaster -D /usr/local/pgsql/data
```

which will leave the server running in the foreground. This must again be done while logged into the PostgreSQL user account. Without a `-D`, the server will try to use the data directory in the environment variable `PGDATA`; if neither of these works it will fail.

To start the postmaster in the background, use the usual shell syntax:

```
$ postmaster -D /usr/local/pgsql/data > logfile 2>&1 &
```

It is an extremely good idea to keep the server's stdout and stderr output around somewhere, as suggested here. It will help both for auditing purposes and to diagnose problems. (See Section 8.3 for a more thorough discussion of log file handling.)

The postmaster also takes a number of other command line options. For more information see the reference page and Section 3.4 below. In particular, in order for the server to accept TCP/IP connections (rather than just Unix domain socket ones), you must also specify the `-i` option.

This shell syntax can get tedious quickly. Therefore the shell script wrapper pg_ctl is provided that encapsulates some of the tasks. E.g.,

```
pg_ctl start -l logfile
```

will start the server in the background and put the output into the named log file. The `-D` option has the same meaning as when invoking postmaster directly. pg_ctl also implements a symmetric “stop” operation.

Normally, you will want to start the database server when the computer boots up. This is not required; the PostgreSQL server can be run successfully from non-privileged accounts without root intervention.

Different systems have different conventions for starting up daemons at boot time, so you are advised to familiarize yourself with them. Many systems have a file `/etc/rc.local` or `/etc/rc.d/rc.local` which is almost certainly no bad place to put such a command. Whatever you do, the server must be run by the PostgreSQL user account *and not by root* or any other user. Therefore you probably always want to form your command lines along the lines of `su -c '...' postgres`, for example:

```
su -c 'pg_ctl start -D /usr/local/pgsql/data -l serverlog' postgres
```

Here are a few more operating system specific suggestions. (Always replace the proper installation directory and the user name you chose.)

- For FreeBSD, take a look at the file `contrib/start-scripts/freebsd` in the PostgreSQL source distribution.
- On OpenBSD, add the following lines to the file `/etc/rc.local`:

  ```
  if [ -x /usr/local/pgsql/bin/pg_ctl -a -x /usr/local/pgsql/bin/postmaster ]; then
      su - -c '/usr/local/pgsql/bin/pg_ctl start -l /var/postgresql/log -s' postgres
      echo -n ' postgresql'
  fi
  ```

- On Linux systems either add

  ```
  /usr/local/pgsql/bin/pg_ctl start -l logfile -D /usr/local/pgsql/data
  ```

  to `/etc/rc.d/rc.local` or look into the file `contrib/start-scripts/linux` in the PostgreSQL source distribution to integrate the start and shutdown into the run level system.
- On NetBSD, either use the FreeBSD or Linux start scripts, depending on preference, as an example and place the file at `/usr/local/etc/rc.d/postgresql`.
- On Solaris, create a file called `/etc/init.d/postgresql` to contain the following single line:

  ```
  su - postgres -c "/usr/local/pgsql/bin/pg_ctl start -l logfile -D /usr/local/pgsql/dat
  ```

  Then, create a symbolic link to it in `/etc/rc3.d` as `S99postgresql`.

While the postmaster is running, its PID is in the file `postmaster.pid` in the data directory. This is used as an interlock against multiple postmasters running in the same data directory, and can also be used for shutting down the postmaster.

3.3.1. Server Start-up Failures

There are several common reasons for the postmaster to fail to start up. Check the postmaster’s log file, or start it by hand (without redirecting standard output or standard error) to see what complaint

messages appear. Some of the possible error messages are reasonably self-explanatory, but here are some that are not.

```
FATAL: StreamServerPort: bind() failed: Address already in use
        Is another postmaster already running on that port?
```

This usually means just what it suggests: you tried to start a second postmaster on the same port where one is already running. However, if the kernel error message is not `Address already in use` or some variant of that wording, there may be a different problem. For example, trying to start a postmaster on a reserved port number may draw something like

```
$ postmaster -i -p 666
FATAL: StreamServerPort: bind() failed: Permission denied
        Is another postmaster already running on that port?
```

A message like

```
IpcMemoryCreate: shmget(key=5440001, size=83918612, 01600) failed: Invalid ar-
gument
FATAL 1:  ShmemCreate: cannot create region
```

probably means that your kernel's limit on the size of shared memory areas is smaller than the buffer area that PostgreSQL is trying to create (83918612 bytes in this example). Or it could mean that you don't have System-V-style shared memory support configured into your kernel at all. As a temporary workaround, you can try starting the postmaster with a smaller-than-normal number of buffers (`-B` switch). You will eventually want to reconfigure your kernel to increase the allowed shared memory size, however. You may see this message when trying to start multiple postmasters on the same machine, if their total space requests exceed the kernel limit.

An error like

```
IpcSemaphoreCreate: semget(key=5440026, num=16, 01600) failed: No space left on de-
vice
```

does *not* mean that you've run out of disk space; it means that your kernel's limit on the number of System V semaphores is smaller than the number PostgreSQL wants to create. As above, you may be able to work around the problem by starting the postmaster with a reduced number of backend processes (`-N` switch), but you'll eventually want to increase the kernel limit.

If you get an "illegal system call" error, then it is likely that shared memory or semaphores are not supported at all in your kernel. In that case your only option is to re-configure the kernel to turn on these features.

Details about configuring System V IPC facilities are given in Section 3.5.1.

3.3.2. Client Connection Problems

Although the possible error conditions on the client side are both virtually infinite and application-dependent, a few of them might be directly related to how the server was started up. Conditions other than those shown below should be documented with the respective client application.

```
psql: could not connect to server: Connection refused
        Is the server running on host server.joe.com and accepting
        TCP/IP connections on port 5432?
```

This is the generic “I couldn’t find a server to talk to” failure. It looks like the above when TCP/IP communication is attempted. A common mistake is to forget the `-i` option to allow the postmaster to accept TCP/IP connections.

Alternatively, you’ll get this when attempting Unix-socket communication to a local postmaster:

```
psql: could not connect to server: Connection refused
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.5432"?
```

The last line is useful in verifying that the client is trying to connect where it is supposed to. If there is in fact no postmaster running there, the kernel error message will typically be either `Connection refused` or `No such file or directory`, as illustrated. (It is particularly important to realize that `Connection refused` in this context does *not* mean that the postmaster got your connection request and rejected it -- that case will produce a different message, as shown in Section 4.3.) Other error messages such as `Connection timed out` may indicate more fundamental problems, like lack of network connectivity.

3.4. Run-time configuration

There are a lot of configuration parameters that affect the behavior of the database system in some way or other. Here we describe how to set them and the following subsections will discuss each of them.

All parameter names are case-insensitive. Every parameter takes a value of one of the four types Boolean, integer, floating point, string as described below. Boolean values are `ON`, `OFF`, `TRUE`, `FALSE`, `YES`, `NO`, `1`, `0` (case-insensitive) or any non-ambiguous prefix of these.

One way to set these options is to edit the file `postgresql.conf` in the data directory. (A default file is installed there.) An example of what this file could look like is:

```
# This is a comment
log_connections = yes
syslog = 2
```

As you see, options are one per line. The equal sign between name and value is optional. Whitespace is insignificant, blank lines are ignored. Hash marks (“#”) introduce comments anywhere.

The configuration file is reread whenever the postmaster receives a SIGHUP signal (which is most easily sent by means of `pg_ctl reload`). The postmaster also propagates this signal to all already-running backend processes, so that existing sessions also get the new default. Alternatively, you can send the signal to only one backend process directly.

A second way to set these configuration parameters is to give them as a command line option to the postmaster, such as

```
postmaster -c log_connections=yes -c syslog=2
```

which would have the same effect as the previous example. Command-line options override any conflicting settings in `postgresql.conf`.

Occasionally it is also useful to give a command line option to one particular backend session only. The environment variable `PGOPTIONS` can be used for this purpose on the client side:

```
env PGOPTIONS='-c geqo=off' psql
```

(This works for any client application, not just psql.) Note that this won't work for options that are necessarily fixed once the server is started, such as the port number.

Finally, some options can be changed in individual SQL sessions with the **SET** command, for example

```
=> SET ENABLE_SEQSCAN TO OFF;
```

See the SQL command language reference for details on the syntax.

3.4.1. Planner and Optimizer Tuning

`CPU_INDEX_TUPLE_COST` (`floating point`)

Sets the query optimizer's estimate of the cost of processing each index tuple during an index scan. This is measured as a fraction of the cost of a sequential page fetch.

`CPU_OPERATOR_COST` (`floating point`)

Sets the optimizer's estimate of the cost of processing each operator in a WHERE clause. This is measured as a fraction of the cost of a sequential page fetch.

`CPU_TUPLE_COST` (`floating point`)

Sets the query optimizer's estimate of the cost of processing each tuple during a query. This is measured as a fraction of the cost of a sequential page fetch.

`EFFECTIVE_CACHE_SIZE` (`floating point`)

Sets the optimizer's assumption about the effective size of the disk cache (that is, the portion of the kernel's disk cache that will be used for PostgreSQL data files). This is measured in disk pages, which are normally 8 kB apiece.

`ENABLE_HASHJOIN` (`boolean`)

Enables or disables the query planner's use of hash-join plan types. The default is on. This is mostly useful to debug the query planner.

`ENABLE_INDEXSCAN` (`boolean`)

Enables or disables the query planner's use of index-scan plan types. The default is on. This is mostly useful to debug the query planner.

`ENABLE_MERGEJOIN` (`boolean`)

Enables or disables the query planner's use of merge-join plan types. The default is on. This is mostly useful to debug the query planner.

`ENABLE_NESTLOOP` (`boolean`)

Enables or disables the query planner's use of nested-loop join plans. It's not possible to suppress nested-loop joins entirely, but turning this variable off discourages the planner from using one if there is any other method available. The default is on. This is mostly useful to debug the query planner.

`ENABLE_SEQSCAN` (`boolean`)

Enables or disables the query planner's use of sequential scan plan types. It's not possible to suppress sequential scans entirely, but turning this variable off discourages the planner from using one if there is any other method available. The default is on. This is mostly useful to debug the query planner.

`ENABLE_SORT` (`boolean`)

Enables or disables the query planner's use of explicit sort steps. It's not possible to suppress explicit sorts entirely, but turning this variable off discourages the planner from using one if there is any other method available. The default is on. This is mostly useful to debug the query planner.

`ENABLE_TIDSCAN` (`boolean`)

Enables or disables the query planner's use of TID scan plan types. The default is on. This is mostly useful to debug the query planner.

`GEQO` (`boolean`)

Enables or disables genetic query optimization, which is an algorithm that attempts to do query planning without exhaustive search. This is on by default. See also the various other `GEQO_` settings.

`GEQO_EFFORT` (`integer`)
`GEQO_GENERATIONS` (`integer`)
`GEQO_POOL_SIZE` (`integer`)
`GEQO_RANDOM_SEED` (`integer`)
`GEQO_SELECTION_BIAS` (`floating point`)

Various tuning parameters for the genetic query optimization algorithm: The pool size is the number of individuals in one population. Valid values are between 128 and 1024. If it is set to 0 (the default) a pool size of 2^(QS+1), where QS is the number of FROM items in the query, is taken. The effort is used to calculate a default for generations. Valid values are between 1 and 80, 40 being the default. Generations specifies the number of iterations in the algorithm. The number must be a positive integer. If 0 is specified then `Effort * Log2(PoolSize)` is used. The run time of the algorithm is roughly proportional to the sum of pool size and generations. The selection bias is the selective pressure within the population. Values can be from 1.50 to 2.00; the latter is the default. The random seed can be set to get reproducible results from the algorithm. If it is set to -1 then the algorithm behaves non-deterministically.

`GEQO_THRESHOLD` (`integer`)

Use genetic query optimization to plan queries with at least this many `FROM` items involved. (Note that a `JOIN` construct counts as only one `FROM` item.) The default is 11. For simpler queries it is usually best to use the deterministic, exhaustive planner. This parameter also controls how hard the optimizer will try to merge subquery `FROM` clauses into the upper query.

`KSQO` (`boolean`)

The *Key Set Query Optimizer* (KSQO) causes the query planner to convert queries whose `WHERE` clause contains many OR'ed AND clauses (such as `WHERE (a=1 AND b=2) OR (a=2 AND b=3) ...`) into a union query. This method can be faster than the default implementation, but it doesn't necessarily give exactly the same results, since `UNION` implicitly adds a `SELECT DISTINCT` clause to eliminate identical output rows. KSQO is commonly used when working with products like Microsoft Access, which tend to generate queries of this form.

The KSQO algorithm used to be absolutely essential for queries with many OR'ed AND clauses, but in PostgreSQL 7.0 and later the standard planner handles these queries fairly successfully. Hence the default is off.

`RANDOM_PAGE_COST` (`floating point`)

Sets the query optimizer's estimate of the cost of a nonsequentially fetched disk page. This is measured as a multiple of the cost of a sequential page fetch.

Note: Unfortunately, there is no well-defined method of determining ideal values for the family of "COST" variables that were just described. You are encouraged to experiment and share your findings.

3.4.2. Logging and Debugging

`DEBUG_ASSERTIONS` (`boolean`)

Turns on various assertion checks. This is a debugging aid. If you are experiencing strange problems or crashes you might want to turn this on, as it might expose programming mistakes. To use this option, the macro `USE_ASSERT_CHECKING` must be defined when PostgreSQL is built (see the configure option `--enable-cassert`). Note that `DEBUG_ASSERTIONS` defaults to on if PostgreSQL has been built this way.

`DEBUG_LEVEL` (`integer`)

The higher this value is set, the more "debugging" output of various sorts is generated in the server log during operation. This option is 0 by default, which means no debugging output. Values up to about 4 currently make sense.

`DEBUG_PRINT_QUERY` (`boolean`)
`DEBUG_PRINT_PARSE` (`boolean`)
`DEBUG_PRINT_REWRITTEN` (`boolean`)
`DEBUG_PRINT_PLAN` (`boolean`)
`DEBUG_PRETTY_PRINT` (`boolean`)

These flags enable various debugging output to be sent to the server log. For each executed query, prints either the query text, the resulting parse tree, the query rewriter output, or the execution plan. `DEBUG_PRETTY_PRINT` indents these displays to produce a more readable but much longer output format. Setting `DEBUG_LEVEL` above zero implicitly turns on some of these flags.

`HOSTNAME_LOOKUP` (`boolean`)

By default, connection logs only show the IP address of the connecting host. If you want it to show the host name you can turn this on, but depending on your host name resolution setup it might impose a non-negligible performance penalty. This option can only be set at server start.

`LOG_CONNECTIONS` (`boolean`)

Prints a line informing about each successful connection in the server log. This is off by default, although it is probably very useful. This option can only be set at server start or in the `postgresql.conf` configuration file.

`LOG_PID` (`boolean`)

Prefixes each server log message with the process ID of the backend process. This is useful to sort out which messages pertain to which connection. The default is off.

`LOG_TIMESTAMP` (`boolean`)

Prefixes each server log message with a time stamp. The default is off.

`SHOW_QUERY_STATS` (`boolean`)
`SHOW_PARSER_STATS` (`boolean`)
`SHOW_PLANNER_STATS` (`boolean`)
`SHOW_EXECUTOR_STATS` (`boolean`)

For each query, write performance statistics of the respective module to the server log. This is a crude profiling instrument.

`SHOW_SOURCE_PORT` (`boolean`)

Shows the outgoing port number of the connecting host in the connection log messages. You could trace back the port number to find out what user initiated the connection. Other than that it's pretty useless and therefore off by default. This option can only be set at server start.

`STATS_COMMAND_STRING` (`boolean`)
`STATS_BLOCK_LEVEL` (`boolean`)
`STATS_ROW_LEVEL` (`boolean`)

These flags determine what information backends send to the statistics collector process: current commands, block-level activity statistics, or row-level activity statistics. All default to off. Enabling statistics collection costs a small amount of time per query, but is invaluable for debugging and performance tuning.

`STATS_RESET_ON_SERVER_START` (`boolean`)

If on, collected statistics are zeroed out whenever the server is restarted. If off, statistics are accumulated across server restarts. The default is on. This option can only be set at server start.

`STATS_START_COLLECTOR` (`boolean`)

Controls whether the server should start the statistics-collection subprocess. This is on by default, but may be turned off if you know you have no interest in collecting statistics. This option can only be set at server start.

`SYSLOG` (`integer`)

PostgreSQL allows the use of syslog for logging. If this option is set to 1, messages go both to syslog and the standard output. A setting of 2 sends output only to syslog. (Some messages will still go to the standard output/error.) The default is 0, which means syslog is off. This option must be set at server start.

To use syslog, the build of PostgreSQL must be configured with the `--enable-syslog` option.

`SYSLOG_FACILITY` (`string`)

This option determines the syslog "facility" to be used when syslog is enabled. You may choose from LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7; the default is LOCAL0. See also the documentation of your system's syslog.

`SYSLOG_IDENT` (`string`)

If logging to syslog is enabled, this option determines the program name used to identify PostgreSQL messages in syslog log messages. The default is `postgres`.

`TRACE_NOTIFY` (`boolean`)

Generates a great amount of debugging output for the **LISTEN** and **NOTIFY** commands.

3.4.3. General operation

`AUSTRALIAN_TIMEZONES` (`bool`)

If set to true, `CST`, `EST`, and `SAT` are interpreted as Australian time zones rather than as North American Central/Eastern time zones and Saturday. The default is false.

`AUTHENTICATION_TIMEOUT` (`integer`)

Maximum time to complete client authentication, in seconds. If a would-be client has not completed the authentication protocol in this much time, the server unceremoniously breaks the connection. This prevents hung clients from occupying a connection indefinitely. This option can only be set at server start or in the `postgresql.conf` file.

`DEADLOCK_TIMEOUT` (`integer`)

This is the amount of time, in milliseconds, to wait on a lock before checking to see if there is a deadlock condition or not. The check for deadlock is relatively slow, so we don't want to run it every time we wait for a lock. We (optimistically?) assume that deadlocks are not common in production applications, and just wait on the lock for awhile before starting to ask questions about whether it can ever get unlocked. Increasing this value reduces the amount of time wasted in needless deadlock checks, but slows down reporting of real deadlock errors. The default is 1000 (i.e., one second), which is probably about the smallest value you would want in practice. On a heavily loaded server you might want to raise it. Ideally the setting should exceed your typical transaction time, so as to improve the odds that the lock will be released before the waiter decides to check for deadlock. This option can only be set at server start.

`DEFAULT_TRANSACTION_ISOLATION` (`string`)

Each SQL transaction has an isolation level, which can be either "read committed" or "serializable". This parameter controls what the isolation level of each new transaction is set to. The default is read committed.

Consult the *PostgreSQL User's Guide* and the command **SET TRANSACTION** for more information.

`DYNAMIC_LIBRARY_PATH` (`string`)

If a dynamically loadable module needs to be opened and the specified name does not have a directory component (i.e., the name does not contain a slash), the system will search this path for the specified file. (The name that is used is the name specified in the **CREATE FUNCTION** or **LOAD** command.)

The value for dynamic_library_path has to be a colon-separated list of absolute directory names. If a directory name starts with the special value `$libdir`, the compiled-in PostgreSQL package library directory, which is where the modules provided by the PostgreSQL distribution are installed, is substituted. (Use `pg_config --pkglibdir` to print the name of this directory.) An example value:

```
dynamic_library_path = '/usr/local/lib/postgresql:/home/my_project/lib:$libdir'
```

The default value for this parameter is `$libdir`. If the value is set to the empty string, the automatic path search is turned off.

This parameter can be changed at run time by superusers, but note that a setting done that way will only persist till the end of the client connection, so this method should be reserved for

development purposes. The recommended way to set this parameter is in the `postgresql.conf` configuration file.

`FSYNC` (`boolean`)

If this option is on, the PostgreSQL backend will use the `fsync()` system call in several places to make sure that updates are physically written to disk and do not hang around in the kernel buffer cache. This increases the chance by a large amount that a database installation will still be usable after an operating system or hardware crash. (Crashes of the database server itself do *not* affect this consideration.)

However, this operation slows down PostgreSQL, because at all those points it has to block and wait for the operating system to flush the buffers. Without `fsync`, the operating system is allowed to do its best in buffering, sorting, and delaying writes, which can make for a considerable performance increase. However, if the system crashes, the results of the last few committed transactions may be lost in part or whole; in the worst case, unrecoverable data corruption may occur.

This option is the subject of an eternal debate in the PostgreSQL user and developer communities. Some always leave it off, some turn it off only for bulk loads, where there is a clear restart point if something goes wrong, some leave it on just to be on the safe side. Because it is the safe side, on is also the default. If you trust your operating system, your hardware, and your utility company (or better your UPS), you might want to disable `fsync`.

It should be noted that the performance penalty from doing `fsync`s is considerably less in PostgreSQL version 7.1 than it was in prior releases. If you previously suppressed `fsync`s because of performance problems, you may wish to reconsider your choice.

This option can only be set at server start or in the `postgresql.conf` file.

`KRB_SERVER_KEYFILE` (`string`)

Sets the location of the Kerberos server key file. See Section 4.2.3 for details.

`MAX_CONNECTIONS` (`integer`)

Determines how many concurrent connections the database server will allow. The default is 32 (unless altered while building the server). This parameter can only be set at server start.

`MAX_EXPR_DEPTH` (`integer`)

Sets the maximum expression nesting depth that the parser will accept. The default value is high enough for any normal query, but you can raise it if you need to. (But if you raise it too high, you run the risk of backend crashes due to stack overflow.)

`MAX_FILES_PER_PROCESS` (`integer`)

Sets the maximum number of simultaneously open files in each server subprocess. The default is 1000. The limit actually used by the code is the smaller of this setting and the result of `sysconf(_SC_OPEN_MAX)`. Therefore, on systems where `sysconf` returns a reasonable limit, you don't need to worry about this setting. But on some platforms (notably, most BSD systems), `sysconf` returns a value that is much larger than the system can really support when a large number of processes all try to open that many files. If you find yourself seeing "Too many open files" failures, try reducing this setting. This option can only be set at server start or in the `postgresql.conf` configuration file; if changed in the configuration file, it only affects subsequently-started server subprocesses.

`MAX_FSM_RELATIONS` (`integer`)

Sets the maximum number of relations (tables) for which free space will be tracked in the shared free-space map. The default is 100. This option can only be set at server start.

`MAX_FSM_PAGES` (`integer`)

Sets the maximum number of disk pages for which free space will be tracked in the shared free-space map. The default is 10000. This option can only be set at server start.

`MAX_LOCKS_PER_TRANSACTION` (`integer`)

The shared lock table is sized on the assumption that at most `max_locks_per_transaction` * `max_connections` distinct objects will need to be locked at any one time. The default, 64, has historically proven sufficient, but you might need to raise this value if you have clients that touch many different tables in a single transaction. This option can only be set at server start.

`PASSWORD_ENCRYPTION` (`boolean`)

When a password is specified in **CREATE USER** or **ALTER USER** without writing either ENCRYPTED or UNENCRYPTED, this flag determines whether the password is to be encrypted. The default is off (do not encrypt the password), but this choice may change in a future release.

`PORT` (`integer`)

The TCP port the server listens on; 5432 by default. This option can only be set at server start.

`SHARED_BUFFERS` (`integer`)

Sets the number of shared memory buffers the database server will use. The default is 64. Each buffer is typically 8192 bytes. This option can only be set at server start.

`SILENT_MODE` (`bool`)

Runs postmaster silently. If this option is set, postmaster will automatically run in background and any controlling ttys are disassociated, thus no messages are written to standard output or standard error (same effect as postmaster's -S option). Unless some logging system such as syslog is enabled, using this option is discouraged since it makes it impossible to see error messages.

`SORT_MEM` (`integer`)

Specifies the amount of memory to be used by internal sorts and hashes before switching to temporary disk files. The value is specified in kilobytes, and defaults to 512 kilobytes. Note that for a complex query, several sorts and/or hashes might be running in parallel, and each one will be allowed to use as much memory as this value specifies before it starts to put data into temporary files. And don't forget that each running backend could be doing one or more sorts. So the total memory space needed could be many times the value of `SORT_MEM`.

`SQL_INHERITANCE` (`bool`)

This controls the inheritance semantics, in particular whether subtables are included into the consideration of various commands by default. This was not the case in versions prior to 7.1. If you need the old behavior you can set this variable to off, but in the long run you are encouraged to change your applications to use the `ONLY` keyword to exclude subtables. See the SQL language reference and the *User's Guide* for more information about inheritance.

`SSL` (`boolean`)

Enables SSL connections. Please read Section 3.7 before using this. The default is off.

`TCPIP_SOCKET` (`boolean`)

If this is true, then the server will accept TCP/IP connections. Otherwise only local Unix domain socket connections are accepted. It is off by default. This option can only be set at server start.

`TRANSFORM_NULL_EQUALS` (`boolean`)

When turned on, expressions of the form *`expr`* `= NULL` (or `NULL =` *`expr`*) are treated as *`expr`* `IS NULL`, that is, they return true if *`expr`* evaluates to the NULL value, and false otherwise. The

correct behavior of *expr* = NULL is to always return NULL (unknown). Therefore this option defaults to off.

However, filtered forms in Microsoft Access generate queries that appear to use *expr* = NULL to test for NULLs, so if you use that interface to access the database you might want to turn this option on. Since expressions of the form *expr* = NULL always return NULL (using the correct interpretation) they are not very useful and do not appear often in normal applications, so this option does little harm in practice. But new users are frequently confused about the semantics of expressions involving NULL, so we do not turn this option on by default.

Note that this option only affects the literal = operator, not other comparison operators or other expressions that are computationally equivalent to some expression involving the equals operator (such as IN). Thus, this option is not a general fix for bad programming.

Refer to the *User's Guide* for related information.

UNIX_SOCKET_DIRECTORY (string)

Specifies the directory of the Unix-domain socket on which the postmaster is to listen for connections from client applications. The default is normally /tmp, but can be changed at build time.

UNIX_SOCKET_GROUP (string)

Sets the group owner of the Unix domain socket. (The owning user of the socket is always the user that starts the postmaster.) In combination with the option UNIX_SOCKET_PERMISSIONS this can be used as an additional access control mechanism for this socket type. By default this is the empty string, which uses the default group for the current user. This option can only be set at server start.

UNIX_SOCKET_PERMISSIONS (integer)

Sets the access permissions of the Unix domain socket. Unix domain sockets use the usual Unix file system permission set. The option value is expected to be an numeric mode specification in the form accepted by the chmod and umask system calls. (To use the customary octal format the number must start with a 0 (zero).)

The default permissions are 0777, meaning anyone can connect. Reasonable alternatives would be 0770 (only user and group, see also under UNIX_SOCKET_GROUP) and 0700 (only user). (Note that actually for a Unix socket, only write permission matters and there is no point in setting or revoking read or execute permissions.)

This access control mechanism is independent from the one described in Chapter 4.

This option can only be set at server start.

VACUUM_MEM (integer)

Specifies the maximum amount of memory to be used by **VACUUM** to keep track of to-be-reclaimed tuples. The value is specified in kilobytes, and defaults to 8192 kilobytes. Larger settings may improve the speed of vacuuming large tables that have many deleted tuples.

VIRTUAL_HOST (string)

Specifies the TCP/IP host name or address on which the postmaster is to listen for connections from client applications. Defaults to listening on all configured addresses (including localhost).

3.4.4. WAL

See also Section 11.3 for details on WAL tuning.

`CHECKPOINT_SEGMENTS` (`integer`)

Maximum distance between automatic WAL checkpoints, in log file segments (each segment is normally 16 megabytes). This option can only be set at server start or in the `postgresql.conf` file.

`CHECKPOINT_TIMEOUT` (`integer`)

Maximum time between automatic WAL checkpoints, in seconds. This option can only be set at server start or in the `postgresql.conf` file.

`COMMIT_DELAY` (`integer`)

Time delay between writing a commit record to the WAL buffer and flushing the buffer out to disk, in microseconds. A nonzero delay allows multiple transactions to be committed with only one `fsync` system call, if system load is high enough that additional transactions become ready to commit within the given interval. But the delay is just wasted time if no other transactions become ready to commit. Therefore, the delay is only performed if at least COMMIT_SIBLINGS other transactions are active at the instant that a backend has written its commit record.

`COMMIT_SIBLINGS` (`integer`)

Minimum number of concurrent open transactions to require before performing the `COMMIT_DELAY` delay. A larger value makes it more probable that at least one other transaction will become ready to commit during the delay interval.

`WAL_BUFFERS` (`integer`)

Number of disk-page buffers in shared memory for WAL log. This option can only be set at server start.

`WAL_DEBUG` (`integer`)

If non-zero, turn on WAL-related debugging output on standard error.

`WAL_FILES` (`integer`)

Number of log files that are created in advance at checkpoint time. This option can only be set at server start or in the `postgresql.conf` file.

`WAL_SYNC_METHOD` (`string`)

Method used for forcing WAL updates out to disk. Possible values are `FSYNC` (call `fsync()` at each commit), `FDATASYNC` (call `fdatasync()` at each commit), `OPEN_SYNC` (write WAL files with `open()` option `O_SYNC`), or `OPEN_DATASYNC` (write WAL files with `open()` option `O_DSYNC`). Not all of these choices are available on all platforms. This option can only be set at server start or in the `postgresql.conf` file.

3.4.5. Short options

For convenience there are also single letter option switches available for many parameters. They are described in the following table.

Table 3-1. Short option key

Short option	Equivalent	Remark
-B *x*	shared_buffers = *x*	
-d *x*	debug_level = *x*	
-F	fsync = off	
-h *x*	virtual_host = *x*	
-i	tcpip_socket = on	
-k *x*	unix_socket_directory = *x*	
-l	ssl = on	
-N *x*	max_connections = *x*	
-p *x*	port = *x*	
-fi, -fh, -fm, -fn, -fs, -ft	enable_indexscan=off, enable_hashjoin=off, enable_mergejoin=off, enable_nestloop=off, enable_seqscan=off, enable_tidscan=off	*
-S *x*	sort_mem = *x*	*
-s	show_query_stats = on	*
-tpa, -tpl, -te	show_parser_stats=on, show_planner_stats=on, show_executor_stats=on	*

For historical reasons, options marked "*" must be passed to the individual backend process via the -o postmaster option, for example,

```
$ postmaster -o '-S 1024 -s'
```

or via PGOPTIONS from the client side, as explained above.

3.5. Managing Kernel Resources

A large PostgreSQL installation can quickly hit various operating system resource limits. (On some systems, the factory defaults are so low that you don't even need a really "large" installation.) If you have encountered this kind of problem then keep reading.

3.5.1. Shared Memory and Semaphores

Shared memory and semaphores are collectively referred to as "System V IPC" (together with message queues, which are not relevant for PostgreSQL). Almost all modern operating systems provide these features, but not all of them have them turned on or sufficiently sized by default, especially systems with BSD heritage. (For the QNX and BeOS ports, PostgreSQL provides its own replacement implementation of these facilities.)

The complete lack of these facilities is usually manifested by an Illegal system call error upon postmaster start. In that case there's nothing left to do but to reconfigure your kernel -- PostgreSQL won't work without them.

When PostgreSQL exceeds one of the various hard limits of the IPC resources then the postmaster will refuse to start up and should leave a marginally instructive error message about which problem was encountered and what needs to be done about it. (See also Section 3.3.1.) The relevant kernel parameters are named consistently across different systems; Table 3-2 gives an overview. The methods to set them, however, vary; suggestions for some platforms are given below. Be warned that it is often necessary to reboot your machine at least, possibly even recompile the kernel, to change these settings.

Table 3-2. System V IPC parameters

Name	Description	Reasonable values
SHMMAX	Maximum size of shared memory segment (bytes)	250kB + 8.2kB * shared_buffers + 14.2kB * max_connections or infinity
SHMMIN	Minimum size of shared memory segment (bytes)	1
SHMALL	Total amount of shared memory available (bytes or pages)	if bytes, same as SHMMAX; if pages, ceil(SHMMAX/PAGE_SIZE)
SHMSEG	Maximum number of shared memory segments per process	only 1 segment is needed, but the default is much higher
SHMMNI	Maximum number of shared memory segments system-wide	like SHMSEG plus room for other applications
SEMMNI	Maximum number of semaphore identifiers (i.e., sets)	>= ceil(max_connections / 16)
SEMMNS	Maximum number of semaphores system-wide	ceil(max_connections / 16) * 17 + room for other applications
SEMMSL	Maximum number of semaphores per set	>= 17
SEMMAP	Number of entries in semaphore map	see text
SEMVMX	Maximum value of semaphore	>= 255 (The default is often 32767, don't change unless asked to.)

The most important shared memory parameter is SHMMAX, the maximum size, in bytes, that a shared memory segment can have. If you get an error message from shmget along the lines of Invalid argument then it is possible that this limit has been exceeded. The size of the required shared memory segments varies both with the number of requested buffers (-B option) and the number of allowed connections (-N option), although the former is the dominant item. (You can therefore, as a temporary solution, lower these settings to get rid of the failures.) As a rough approximation you can estimate the required segment size as the number of buffers times the block size (8 kB by default) plus ample overhead (at least half a megabyte). Any error message you might get will contain the size of the failed allocation request.

Less likely to cause problems is the minimum size for shared memory segments (SHMMIN), which should be at most somewhere around 256 kB for PostgreSQL (it is usually just 1). The maximum number of segments system-wide (SHMMNI) or per-process (SHMSEG) should not cause a problem unless your system has them set to zero. Some systems also have a limit on the total amount of shared memory in the system; see the platform-specific instructions below.

PostgreSQL uses one semaphore per allowed connection (`-N` option), in sets of 16. Each such set will also contain a 17th semaphore which contains a "magic number", to detect collision with semaphore sets used by other applications. The maximum number of semaphores in the system is set by `SEMMNS`, which consequently must be at least as high as the connection setting plus one extra for each 16 allowed connections (see the formula in Table 3-2). The parameter `SEMMNI` determines the limit on the number of semaphore sets that can exist on the system at one time. Hence this parameter must be at least `ceil(max_connections / 16)`. Lowering the number of allowed connections is a temporary workaround for failures, which are usually confusingly worded "No space left on device", from the function `semget()`.

In some cases it might also turn out to be necessary to increase `SEMMAP` to be at least on the order of `SEMMNS`. This parameter defines the size of the semaphore resource map, in which each contiguous block of available semaphores needs an entry. When a semaphore set is freed it is either added to an existing entry that is adjacent to the freed block or it is registered under a new map entry. If the map is full, the freed semaphores get lost (until reboot). Fragmentation of the semaphore space could therefore over time lead to less available semaphores than there should be.

The `SEMMSL` parameter, which determines how many semaphores can be in a set, must be at least 17 for PostgreSQL.

Various other settings related to "semaphore undo", such as `SEMMNU` and `SEMUME`, are not of concern for PostgreSQL.

BSD/OS

Shared Memory. By default, only 4 MB of shared memory is supported. Keep in mind that shared memory is not pageable; it is locked in RAM. To increase the number of shared buffers supported by the postmaster, add the following to your kernel configuration file. A `SHMALL` value of 1024 represents 4MB of shared memory. The following increases the maximum shared memory area to 32 MB:

```
options "SHMALL=8192"
options "SHMMAX=\(SHMALL*PAGE_SIZE\)"
```

For those running 4.1 or later, just make the above changes, recompile the kernel, and reboot. For those running earlier releases, use bpatch to find the `sysptsize` value in the current kernel. This is computed dynamically at boot time.

```
$ bpatch -r sysptsize
0x9 = 9
```

Next, add `SYSPTSIZE` as a hard-coded value in the kernel configuration file. Increase the value you found using bpatch. Add 1 for every additional 4 MB of shared memory you desire.

```
options "SYSPTSIZE=16"
```

`sysptsize` cannot be changed by **sysctl**.

Semaphores. You may need to increase the number of semaphores. By default, PostgreSQL allocates 34 semaphores, which is over half the default system total of 60.

Set the values you want in your kernel configuration file, e.g.:

```
options "SEMMNI=40"
options "SEMMNS=240"
options "SEMUME=40"
options "SEMMNU=120"
```

FreeBSD
NetBSD
OpenBSD

The options `SYSVSHM` and `SYSVSEM` need to be enabled when the kernel is compiled. (They are by default.) The maximum size of shared memory is determined by the option `SHMMAXPGS` (in pages). The following shows an example of how to set the various parameters:

```
options         SYSVSHM
options         SHMMAXPGS=4096
options         SHMSEG=256

options         SYSVSEM
options         SEMMNI=256
options         SEMMNS=512
options         SEMMNU=256
options         SEMMAP=256
```

(On NetBSD and OpenBSD the key word is actually `option` singular.)

HP-UX

The default settings tend to suffice for normal installations. On HP-UX 10, the factory default for `SEMMNS` is 128, which might be too low for larger database sites.

IPC parameters can be set in the System Administration Manager (SAM) under Kernel Configuration⟶Configurable Parameters. Hit Create A New Kernel when you're done.

Linux

The default shared memory limit (both `SHMMAX` and `SHMALL`) is 32 MB in 2.2 kernels, but it can be changed in the `proc` file system (without reboot). For example, to allow 128 MB:

```
$ echo 134217728 >/proc/sys/kernel/shmall
$ echo 134217728 >/proc/sys/kernel/shmmax
```

You could put these commands into a script run at boot-time.

Alternatively, you can use sysctl, if available, to control these parameters. Look for a file called `/etc/sysctl.conf` and add lines like the following to it:

```
kernel.shmall = 134217728
kernel.shmmax = 134217728
```

This file is usually processed at boot time, but sysctl can also be called explicitly later.

Other parameters are sufficiently sized for any application. If you want to see for yourself look into `/usr/src/linux/include/asm-`*`xxx`*`/shmparam.h` and `/usr/src/linux/include/linux/sem.h`.

SCO OpenServer

In the default configuration, only 512 kB of shared memory per segment is allowed, which is about enough for `-B 24 -N 12`. To increase the setting, first change the directory to `/etc/conf/cf.d`. To display the current value of `SHMMAX`, in bytes, run

```
./configure -y SHMMAX
```

To set a new value for `SHMMAX`, run:

```
./configure SHMMAX=value
```

where *`value`* is the new value you want to use (in bytes). After setting `SHMMAX`, rebuild the kernel

```
./link_unix
```

and reboot.

Solaris

At least in version 2.6, the maximum size of a shared memory segment is set too low for PostgreSQL. The relevant settings can be changed in `/etc/system`, for example:

```
set shmsys:shminfo_shmmax=0x2000000
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=256
set shmsys:shminfo_shmseg=256

set semsys:seminfo_semmap=256
set semsys:seminfo_semmni=512
set semsys:seminfo_semmns=512
set semsys:seminfo_semmsl=32
```

You need to reboot to make the changes effective.

See also http://www.sunworld.com/swol-09-1997/swol-09-insidesolaris.html for information on shared memory under Solaris.

UnixWare

On UnixWare 7, the maximum size for shared memory segments is 512 kB in the default configuration. This is enough for about `-B 24 -N 12`. To display the current value of `SHMMAX`, run

```
/etc/conf/bin/idtune -g SHMMAX
```

which displays the current, default, minimum, and maximum values, in bytes. To set a new value for `SHMMAX`, run:

```
/etc/conf/bin/idtune SHMMAX value
```

where *`value`* is the new value you want to use (in bytes). After setting `SHMMAX`, rebuild the kernel

```
/etc/conf/bin/idbuild -B
```

and reboot.

3.5.2. Resource Limits

Unix-like operating systems enforce various kinds of resource limits that might interfere with the operation of your PostgreSQL server. Of importance are especially the limits on the number of processes per user, the number of open files per process, and the amount of memory available to a process. Each of these have a “hard” and a “soft” limit. The soft limit is what actually counts but it can be changed by the user up to the hard limit. The hard limit can only be changed by the root user. The system call `setrlimit` is responsible for setting these parameters. The shell’s built-in command **ulimit** (Bourne shells) or **limit** (csh) is used to control the resource limits from the command line. On BSD-derived systems the file `/etc/login.conf` controls what values the various resource limits are set to upon login. See login.conf for details. The relevant parameters are `maxproc`, `openfiles`, and `datasize`. For example:

```
default:\
...
        :datasize-cur=256M:\
        :maxproc-cur=256:\
        :openfiles-cur=256:\
```

```
...
```

(`-cur` is the soft limit. Append `-max` to set the hard limit.)

Kernels generally also have an implementation-dependent system-wide limit on some resources.

- On Linux `/proc/sys/fs/file-max` determines the maximum number of open files that the kernel will support. It can be changed by writing a different number into the file or by adding an assignment in `/etc/sysctl.conf`. The maximum limit of files per process is fixed at the time the kernel is compiled; see `/usr/src/linux/Documentation/proc.txt` for more information.

The PostgreSQL server uses one process per connection so you should provide for at least as many processes as allowed connections, in addition to what you need for the rest of your system. This is usually not a problem but if you run several servers on one machine things might get tight.

The factory default limit on open files is often set to "socially friendly" values that allow many users to coexist on a machine without using an inappropriate fraction of the system resources. If you run many servers on a machine this is perhaps what you want, but on dedicated servers you may want to raise this limit.

On the other side of the coin, some systems allow individual processes to open large numbers of files; if more than a few processes do so then the system-wide limit can easily be exceeded. If you find this happening, and don't want to alter the system-wide limit, you can set PostgreSQL's `max_files_per_process` configuration parameter to limit its consumption of open files.

3.6. Shutting down the server

Depending on your needs, there are several ways to shut down the database server when your work is done. The differentiation is done by what signal you send to the server process.

SIGTERM

After receiving SIGTERM, the postmaster disallows new connections, but lets existing backends end their work normally. It shuts down only after all of the backends terminate by client request. This is the *Smart Shutdown*.

SIGINT

The postmaster disallows new connections and sends all existing backends SIGTERM, which will cause them to abort their current transactions and exit promptly. It then waits for the backends to exit and finally shuts down the data base. This is the *Fast Shutdown*.

SIGQUIT

This is the *Immediate Shutdown* which will cause the postmaster to send a SIGQUIT to all backends and exit immediately (without properly shutting down the database system). The backends likewise exit immediately upon receiving SIGQUIT. This will lead to recovery (by replaying the WAL log) upon next start-up. This is recommended only in emergencies.

Important: It is best not to use SIGKILL to shut down the postmaster. This will prevent the postmaster from releasing shared memory and semaphores, which you may then have to do by hand.

The PID of the postmaster process can be found using the ps program, or from the file `postmaster.pid` in the data directory. So for example, to do a fast shutdown:

```
$ kill -INT `head -1 /usr/local/pgsql/data/postmaster.pid`
```

The program pg_ctl is a shell script that provides a more convenient interface for shutting down the postmaster.

3.7. Secure TCP/IP Connections with SSL

PostgreSQL has native support for connections over SSL to encrypt client/server communications for increased security. This requires OpenSSL to be installed on both client and server systems and support enabled at build time (see Chapter 1).

With SSL support compiled in, the PostgreSQL server can be started with the argument `-l` (ell) to enable SSL connections. When starting in SSL mode, the server will look for the files `server.key` and `server.crt` in the data directory. These files should contain the server private key and certificate respectively. These files must be set up correctly before an SSL-enabled server can start. If the private key is protected with a passphrase, the server will prompt for the passphrase and will not start until it has been entered.

The server will listen for both standard and SSL connections on the same TCP/IP port, and will negotiate with any connecting client whether or not to use SSL. See Chapter 4 about how to force on the server side the use of SSL for certain connections.

For details on how to create your server private key and certificate, refer to the OpenSSL documentation. A simple self-signed certificate can be used to get started for testing, but a certificate signed by a CA (either one of the global CAs or a local one) should be used in production so the client can verify the server's identity. To create a quick self-signed certificate, use the following OpenSSL command:

```
openssl req -new -text -out cert.req
```

Fill out the information that **openssl** asks for. Make sure that you enter the local host name as Common Name; the challenge password can be left blank. The script will generate a key that is passphrase protected; it will not accept a pass phrase that is less than four characters long. To remove the passphrase (as you must if you want automatic start-up of the server), run the commands

```
openssl rsa -in privkey.pem -out cert.pem
```

Enter the old passphrase to unlock the existing key. Now do

```
openssl req -x509 -in cert.req -text -key cert.pem -out cert.cert
cp cert.pem $PGDATA/server.key
cp cert.cert $PGDATA/server.crt
```

to turn the certificate into a self-signed certificate and to copy the key and certificate to where the server will look for them.

3.8. Secure TCP/IP Connections with SSH tunnels

Acknowledgement: Idea taken from an email by Gene Selkov, Jr. (<selkovjr@mcs.anl.gov>) written on 1999-09-08 in response to a question from Eric Marsden.

One can use ssh to encrypt the network connection between clients and a PostgreSQL server. Done properly, this should lead to an adequately secure network connection.

First make sure that an ssh server is running properly on the same machine as PostgreSQL and that you can log in using **ssh** as some user. Then you can establish a secure tunnel with a command like this from the client machine:

```
$ ssh -L 3333:foo.com:5432 joe@foo.com
```

The first number in the `-L` argument, 3333, is the port number of your end of the tunnel; it can be chosen freely. The second number, 5432, is the remote end of the tunnel -- the port number your server is using. The name or the address in between the port numbers is the host with the database server you are going to connect to. In order to connect to the database server using this tunnel, you connect to port 3333 on the local machine:

```
psql -h localhost -p 3333 template1
```

To the database server it will then look as though you are really user `joe@foo.com` and it will use whatever authentication procedure was set up for this user. In order for the tunnel setup to succeed you must be allowed to connect via **ssh** as joe@foo.com, just as if you had attempted to use **ssh** to set up a terminal session.

> **Tip:** Several other products exist that can provide secure tunnels using a procedure similar in concept to the one just described.

Chapter 4. Client Authentication

When a client application connects to the database server, it specifies which PostgreSQL user name it wants to connect as, much the same way one logs into a Unix computer as a particular user. Within the SQL environment the active database user name determines access privileges to database objects -- see Chapter 7 for more information about that. It is therefore obviously essential to restrict which database user name(s) a given client can connect as.

Authentication is the process by which the database server establishes the identity of the client, and by extension determines whether the client application (or the user who runs the client application) is permitted to connect with the user name that was requested.

PostgreSQL offers a number of different client authentication methods. The method to be used can be selected on the basis of (client) host and database; some authentication methods allow you to restrict by user name as well.

PostgreSQL database user names are logically separate from user names of the operating system in which the server runs. If all the users of a particular server also have accounts on the server's machine, it makes sense to assign database user names that match their operating system user names. However, a server that accepts remote connections may have many users who have no local account, and in such cases there need be no connection between database user names and OS user names.

4.1. The `pg_hba.conf` file

Client authentication is controlled by the file `pg_hba.conf` in the data directory, e.g., `/usr/local/pgsql/data/pg_hba.conf`. (HBA stands for host-based authentication.) A default `pg_hba.conf` file is installed when the data area is initialized by **initdb**.

The general format of the `pg_hba.conf` file is of a set of records, one per line. Blank lines and lines beginning with a hash character ("#") are ignored. A record is made up of a number of fields which are separated by spaces and/or tabs. Records cannot be continued across lines.

Each record specifies a connection type, a client IP address range (if relevant for the connection type), a database name or names, and the authentication method to be used for connections matching these parameters. The first record that matches the type, client address, and requested database name of a connection attempt is used to do the authentication step. There is no "fall-through" or "backup": if one record is chosen and the authentication fails, the following records are not considered. If no record matches, the access will be denied.

A record may have one of the three formats

```
local   database authentication-method [ authentication-option ]
host    database IP-address IP-mask authentication-method [ authentication-
option ]
hostssl database IP-address IP-mask authentication-method [ authentication-
option ]
```

The meaning of the fields is as follows:

`local`

This record pertains to connection attempts over Unix domain sockets.

`host`

This record pertains to connection attempts over TCP/IP networks. Note that TCP/IP connections are completely disabled unless the server is started with the `-i` switch or the equivalent configuration parameter is set.

`hostssl`

This record pertains to connection attempts with SSL over TCP/IP. To make use of this option the server must be built with SSL support enabled. Furthermore, SSL must be enabled with the `-l` option or equivalent configuration setting when the server is started. (Note: `host` records will match either SSL or non-SSL connection attempts, but `hostssl` records match only SSL connections.)

`database`

Specifies the database that this record applies to. The value `all` specifies that it applies to all databases, while the value `sameuser` identifies the database with the same name as the connecting user. Otherwise, this is the name of a specific PostgreSQL database.

`IP address`
`IP mask`

These two fields specify to which client machines a `host` or `hostssl` record applies, based on their IP address. (Of course IP addresses can be spoofed but this consideration is beyond the scope of PostgreSQL.) The precise logic is that

```
(actual-IP-address xor IP-address-field) and IP-mask-field
```

must be zero for the record to match.

`authentication method`

Specifies the method that users must use to authenticate themselves when connecting under the control of this authentication record. The possible choices are summarized here, details are in Section 4.2.

`trust`

The connection is allowed unconditionally. This method allows any user that has login access to the client host to connect as any PostgreSQL user whatsoever.

`reject`

The connection is rejected unconditionally. This is mostly useful to “filter out” certain hosts from a group.

`password`

The client is required to supply a password which is required to match the database password that was set up for the user.

An optional file name may be specified after the `password` keyword. This file is expected to contain a list of users who may connect using this record, and optionally alternative passwords for them.

The password is sent over the wire in clear text. For better protection, use the `md5` or `crypt` methods.

`md5`

Like the `password` method, but the password is sent over the wire encrypted using a simple challenge-response protocol. This protects against incidental wire-sniffing. This is now the recommended choice for password-based authentication.

The name of a file may follow the `md5` keyword. It contains a list of users who may connect using this record.

`crypt`

Like the `md5` method but uses older crypt encryption, which is needed for pre-7.2 clients. `md5` is preferred for 7.2 and later clients. The `crypt` method is not compatible with encrypting passwords in `pg_shadow`, and may fail if client and server machines have different implementations of the crypt() library routine.

`krb4`

Kerberos V4 is used to authenticate the user. This is only available for TCP/IP connections.

`krb5`

Kerberos V5 is used to authenticate the user. This is only available for TCP/IP connections.

`ident`

The identity of the user as determined on login to the operating system is used by PostgreSQL to determine whether the user is allowed to connect as the requested database user. For TCP/IP connections the user's identity is determined by contacting the *ident* server on the client host. (Note that this is only as reliable as the remote ident server; ident authentication should never be used for remote hosts whose administrators are not trustworthy.) On operating systems supporting `SO_PEERCRED` requests for Unix domain sockets, ident authentication is possible for local connections; the system is then asked for the connecting user's identity.

On systems without `SO_PEERCRED` requests, ident authentication is only available for TCP/IP connections. As a workaround, it is possible to specify the localhost address 127.0.0.1 and make connections to this address.

The *`authentication option`* following the `ident` keyword specifies the name of an *ident map* that specifies which operating system users equate with which database users. See below for details.

`pam`

This authentication type operates similarly to *password*, with the main difference that it will use PAM (Pluggable Authentication Modules) as the authentication mechanism. The *`authentication option`* following the `pam` keyword specifies the service name that will be passed to PAM. The default service name is `postgresql`. For more information about PAM, please read the Linux-PAM Page[1] and/or the Solaris PAM Page[2].

`authentication option`

This field is interpreted differently depending on the authentication method, as described above.

Since the `pg_hba.conf` records are examined sequentially for each connection attempt, the order of the records is very significant. Typically, earlier records will have tight connection match param-

1. http://www.kernel.org/pub/linux/libs/pam/
2. http://www.sun.com/software/solaris/pam/

eters and weaker authentication methods, while later records will have looser match parameters and stronger authentication methods. For example, one might wish to use `trust` authentication for local TCP connections but require a password for remote TCP connections. In this case a record specifying `trust` authentication for connections from 127.0.0.1 would appear before a record specifying password authentication for a wider range of allowed client IP addresses.

The `pg_hba.conf` file is read on startup and when the postmaster receives a SIGHUP signal. If you edit the file on an active system, you will need to signal the postmaster (using `pg_ctl reload` or `kill -HUP`) to make it re-read the file.

An example of a `pg_hba.conf` file is shown in Example 4-1. See below for details on the different authentication methods.

Example 4-1. An example `pg_hba.conf` file

```
# TYPE       DATABASE    IP_ADDRESS    MASK               AUTHTYPE  MAP

# Allow any user on the local system to connect to any
# database under any username, but only via an IP connection:

host         all         127.0.0.1     255.255.255.255    trust

# The same, over Unix-socket connections:

local        all                                          trust

# Allow any user from any host with IP address 192.168.93.x to
# connect to database "template1" as the same username that ident on that
# host identifies him as (typically his Unix username):

host         template1   192.168.93.0  255.255.255.0      ident     sameuser

# Allow a user from host 192.168.12.10 to connect to database "template1"
# if the user's password in pg_shadow is correctly supplied:

host         template1   192.168.12.10 255.255.255.255    md5

# In the absence of preceding "host" lines, these two lines will reject
# all connection attempts from 192.168.54.1 (since that entry will be
# matched first), but allow Kerberos V5-validated connections from anywhere
# else on the Internet. The zero mask means that no bits of the host IP
# address are considered, so it matches any host:

host         all        192.168.54.1   255.255.255.255    reject
host         all        0.0.0.0        0.0.0.0            krb5

# Allow users from 192.168.x.x hosts to connect to any database, if they
# pass the ident check.  If, for example, ident says the user is "bryanh"
# and he requests to connect as PostgreSQL user "guest1", the connection
# is allowed if there is an entry in pg_ident.conf for map "omicron" that
# says "bryanh" is allowed to connect as "guest1":

host         all        192.168.0.0    255.255.0.0        ident     omicron

# If these are the only two lines for local connections, they will allow
# local users to connect only to their own databases (database named the
# same as the user name), except for administrators who may connect to
```

```
# all databases.  The file $PGDATA/admins lists the user names who are
# permitted to connect to all databases.  Passwords are required in all
# cases.  (If you prefer to use ident authorization, an ident map can
# serve a parallel purpose to the password list file used here.)

local        sameuser                                    md5
local        all                                         md5  admins
```

4.2. Authentication methods

The following describes the authentication methods in more detail.

4.2.1. Trust authentication

When `trust` authentication is specified, PostgreSQL assumes that anyone who can connect to the postmaster is authorized to access the database as whatever database user he specifies (including the database superuser). This method should only be used when there is adequate system-level protection on connections to the postmaster port.

`trust` authentication is appropriate and very convenient for local connections on a single-user workstation. It is usually *not* appropriate by itself on a multiuser machine. However, you may be able to use `trust` even on a multiuser machine, if you restrict access to the postmaster's socket file using file-system permissions. To do this, set the parameter `unix_socket_permissions` (and possibly `unix_socket_group`) in `postgresql.conf`, as described in Section 3.4.3. Or you could set `unix_socket_directory` to place the socket file in a suitably restricted directory.

Setting file-system permissions only helps for Unix-socket connections. Local TCP connections are not restricted by it; therefore, if you want to use permissions for local security, remove the `host ... 127.0.0.1 ...` line from `pg_hba.conf`, or change it to a non-`trust` authentication method.

`trust` authentication is only suitable for TCP connections if you trust every user on every machine that is allowed to connect to the postmaster by the `pg_hba.conf` lines that specify `trust`. It is seldom reasonable to use `trust` for any TCP connections other than those from localhost (127.0.0.1).

4.2.2. Password authentication

Password-based authentication methods include `md5`, `crypt`, and `password`. These methods operate similarly except for the way that the password is sent across the connection. If you are at all concerned about password "sniffing" attacks then `md5` is preferred, with `crypt` a second choice if you must support obsolete clients. Plain `password` should especially be avoided for connections over the open Internet (unless you use SSL, SSH, or other communications security wrappers around the connection).

PostgreSQL database passwords are separate from operating system user passwords. Ordinarily, the password for each database user is stored in the pg_shadow system catalog table. Passwords can be managed with the query language commands **CREATE USER** and **ALTER USER**, e.g., **`CREATE USER foo WITH PASSWORD 'secret';`**. By default, that is, if no password has been set up, the stored password is `NULL` and password authentication will always fail for that user.

To restrict the set of users that are allowed to connect to certain databases, list the set of users in a separate file (one user name per line) in the same directory that `pg_hba.conf` is in, and mention the (base) name of the file after the `password`, `md5`, or `crypt` keyword, respectively, in `pg_hba.conf`.

If you do not use this feature, then any user that is known to the database system can connect to any database (so long as he supplies the correct password, of course).

These files can also be used to apply a different set of passwords to a particular database or set thereof. In that case, the files have a format similar to the standard Unix password file `/etc/passwd`, that is,

username:*password*

Any extra colon-separated fields following the password are ignored. The password is expected to be encrypted using the system's `crypt()` function. The utility program pg_passwd that is installed with PostgreSQL can be used to manage these password files.

Lines with and without passwords can be mixed in secondary password files. Lines without password indicate use of the main password in `pg_shadow` that is managed by **CREATE USER** and **ALTER USER**. Lines with passwords will cause that password to be used. A password entry of "+" also means using the pg_shadow password.

Alternative passwords cannot be used when using the `md5` or `crypt` methods. The file will be read as usual, but the password field will simply be ignored and the `pg_shadow` password will always be used.

Note that using alternative passwords like this means that one can no longer use **ALTER USER** to change one's password. It will appear to work but the password one is changing is not the password that the system will end up using.

4.2.3. Kerberos authentication

Kerberos is an industry-standard secure authentication system suitable for distributed computing over a public network. A description of the Kerberos system is far beyond the scope of this document; in all generality it can be quite complex (yet powerful). The Kerberos FAQ[3] or MIT Project Athena[4] can be a good starting point for exploration. Several sources for Kerberos distributions exist.

In order to use Kerberos, support for it must be enabled at build time. Both Kerberos 4 and 5 are supported (`./configure --with-krb4` or `./configure --with-krb5` respectively), although only one version can be supported in any one build.

PostgreSQL operates like a normal Kerberos service. The name of the service principal is *`servicename/hostname@realm`*, where *`servicename`* is `postgres` (unless a different service name was selected at configure time with `./configure --with-krb-srvnam=whatever`). *`hostname`* is the fully qualified domain name of the server machine. The service principal's realm is the preferred realm of the server machine.

Client principals must have their PostgreSQL user name as their first component, for example *`pgusername/otherstuff@realm`*. At present the realm of the client is not checked by PostgreSQL; so if you have cross-realm authentication enabled, then any principal in any realm that can communicate with yours will be accepted.

Make sure that your server key file is readable (and preferably only readable) by the PostgreSQL server account (see Section 3.1). The location of the key file is specified with the `krb_server_keyfile` run time configuration parameter. (See also Section 3.4.) The default is `/etc/srvtab` if you are using Kerberos 4 and `FILE:/usr/local/pgsql/etc/krb5.keytab` (or whichever directory was specified as `sysconfdir` at build time) with Kerberos 5.

To generate the keytab file, use for example (with version 5)

3. http://www.nrl.navy.mil/CCS/people/kenh/kerberos-faq.html
4. ftp://athena-dist.mit.edu

```
kadmin% ank -randkey postgres/server.my.domain.org
kadmin% ktadd -k krb5.keytab postgres/server.my.domain.org
```

Read the Kerberos documentation for details.

When connecting to the database make sure you have a ticket for a principal matching the requested database user name. An example: For database user name `fred`, both principal `fred@EXAMPLE.COM` and `fred/users.example.com@EXAMPLE.COM` can be used to authenticate to the database server.

If you use mod_auth_krb and mod_perl on your Apache web server, you can use `AuthType KerberosV5SaveCredentials` with a mod_perl script. This gives secure database access over the web, no extra passwords required.

4.2.4. Ident-based authentication

The "Identification Protocol" is described in *RFC 1413*. Virtually every Unix-like operating system ships with an ident server that listens on TCP port 113 by default. The basic functionality of an ident server is to answer questions like "What user initiated the connection that goes out of your port *X* and connects to my port *Y*?". Since PostgreSQL knows both *X* and *Y* when a physical connection is established, it can interrogate the ident server on the host of the connecting client and could theoretically determine the operating system user for any given connection this way.

The drawback of this procedure is that it depends on the integrity of the client: if the client machine is untrusted or compromised an attacker could run just about any program on port 113 and return any user name he chooses. This authentication method is therefore only appropriate for closed networks where each client machine is under tight control and where the database and system administrators operate in close contact. In other words, you must trust the machine running the ident server. Heed the warning:

> RFC 1413
>
> The Identification Protocol is not intended as an authorization or access control protocol.

On systems supporting `SO_PEERCRED` requests for Unix-domain sockets, ident authentication can also be applied to local connections. In this case, no security risk is added by using ident authentication; indeed it is a preferable choice for local connections on such a system.

When using ident-based authentication, after having determined the name of the operating system user that initiated the connection, PostgreSQL checks whether that user is allowed to connect as the database user he is requesting to connect as. This is controlled by the ident map argument that follows the `ident` keyword in the `pg_hba.conf` file. There is a predefined ident map `sameuser`, which allows any operating system user to connect as the database user of the same name (if the latter exists). Other maps must be created manually.

Ident maps other than `sameuser` are defined in the file `pg_ident.conf` in the data directory, which contains lines of the general form:

```
map-name ident-username database-username
```

Comments and whitespace are handled in the usual way. The *`map-name`* is an arbitrary name that will be used to refer to this mapping in `pg_hba.conf`. The other two fields specify which operating system user is allowed to connect as which database user. The same *`map-name`* can be used repeatedly to specify more user-mappings within a single map. There is no restriction regarding how many database users a given operating system user may correspond to and vice versa.

The `pg_ident.conf` file is read on startup and when the postmaster receives a SIGHUP signal. If you edit the file on an active system, you will need to signal the postmaster (using `pg_ctl reload` or `kill -HUP`) to make it re-read the file.

A `pg_ident.conf` file that could be used in conjunction with the `pg_hba.conf` file in Example 4-1 is shown in Example 4-2. In this example setup, anyone logged in to a machine on the 192.168 network that does not have the Unix user name bryanh, ann, or robert would not be granted access. Unix user robert would only be allowed access when he tries to connect as PostgreSQL user bob, not as robert or anyone else. ann would only be allowed to connect as ann. User bryanh would be allowed to connect as either bryanh himself or as guest1.

Example 4-2. An example `pg_ident.conf` file

```
#MAP            IDENT-NAME    POSTGRESQL-NAME

omicron         bryanh        bryanh
omicron         ann           ann
# bob has username robert on these machines
omicron         robert        bob
# bryanh can also connect as guest1
omicron         bryanh        guest1
```

4.3. Authentication problems

Genuine authentication failures and related problems generally manifest themselves through error messages like the following.

```
No pg_hba.conf entry for host 123.123.123.123, user joeblow, database testdb
```

This is what you are most likely to get if you succeed in contacting the server, but it does not want to talk to you. As the message suggests, the server refused the connection request because it found no authorizing entry in its `pg_hba.conf` configuration file.

```
Password authentication failed for user 'joeblow'
```

Messages like this indicate that you contacted the server, and it is willing to talk to you, but not until you pass the authorization method specified in the `pg_hba.conf` file. Check the password you are providing, or check your Kerberos or ident software if the complaint mentions one of those authentication types.

```
FATAL 1:  user "joeblow" does not exist
```

The indicated user name was not found.

```
FATAL 1:  Database "testdb" does not exist in the system catalog.
```

The database you are trying to connect to does not exist. Note that if you do not specify a database name, it defaults to the database user name, which may or may not be the right thing.

Note that the server log may contain more information about an authentication failure than is reported to the client. If you are confused about the reason for a failure, check the log.

Chapter 5. Localization

Describes the available localization features from the point of view of the administrator.

PostgreSQL supports localization with three approaches:

- Using the locale features of the operating system to provide locale-specific collation order, number formatting, translated messages, and other aspects.
- Using explicit multiple-byte character sets defined in the PostgreSQL server to support languages that require more characters than will fit into a single byte, and to provide character set recoding between client and server. The number of supported character sets is fixed at the time the server is compiled, and internal operations such as string comparisons require expansion of each character into a 32-bit word.
- Single byte character recoding provides a more light-weight solution for users of multiple, yet single-byte character sets.

5.1. Locale Support

Locale support refers to an application respecting cultural preferences regarding alphabets, sorting, number formatting, etc. PostgreSQL uses the standard ISO C and POSIX-like locale facilities provided by the server operating system. For additional information refer to the documentation of your system.

5.1.1. Overview

Locale support is not built into PostgreSQL by default; to enable it, supply the `--enable-locale` option to the `configure` script:

```
$ ./configure --enable-locale
```

Locale support only affects the server; all clients are compatible with servers with or without locale support.

To enable messages translated to the user's preferred language, the `--enable-nls` option must be used. This option is independent of the other locale support.

The information about which particular cultural rules to use is determined by standard environment variables. If you are getting localized behavior from other programs you probably have them set up already. The simplest way to set the localization information is the `LANG` variable, for example:

```
export LANG=sv_SE
```

This sets the locale to Swedish (`sv`) as spoken in Sweden (`SE`). Other possibilities might be `en_US` (U.S. English) and `fr_CA` (Canada, French). If more than one character set can be useful for a locale then the specifications look like this: `cs_CZ.ISO8859-2`. What locales are available under what names on your system depends on what was provided by the operating system vendor and what was installed.

Occasionally it is useful to mix rules from several locales, e.g., use U.S. collation rules but Spanish messages. To do that a set of environment variables exist that override the default of LANG for a particular category:

LC_COLLATE	String sort order
LC_CTYPE	Character classification (What is a letter? The upper-case equivalent?)
LC_MESSAGES	Language of messages
LC_MONETARY	Formatting of currency amounts
LC_NUMERIC	Formatting of numbers
LC_TIME	Formatting of dates and times

Additionally, all of these specific variables and the LANG variable can be overridden with the LC_ALL environment variable.

> **Note:** Some message localization libraries also look at the environment variable LANGUAGE which overrides all other locale settings for the purpose of setting the language of messages. If in doubt, please refer to the documentation of your operating system, in particular the gettext manual page, for more information.

If you want the system to behave as if it had no locale support, use the special locale C or POSIX, or simply unset all locale-related variables.

Note that the locale behavior of the server is determined by the environment variables seen by the server, not by the environment of any client. Therefore, be careful to set these variables before starting the server. A consequence of this is that if client and server are set up to different locales, messages may appear in different languages depending on where they originated.

The LC_COLLATE and LC_CTYPE variables affect the sort order of indexes. Therefore, these values must be kept fixed for any particular database cluster, or indexes on text columns will become corrupt. PostgreSQL enforces this by recording the values of LC_COLLATE and LC_CTYPE that are seen by initdb. The server automatically adopts those two values when it is started; only the other LC_ categories can be set from the environment at server startup. In short, only one collation order can be used in a database cluster, and it is chosen at initdb time.

5.1.2. Benefits

Locale support influences in particular the following features:

- Sort order in **ORDER BY** queries.
- The to_char family of functions
- The LIKE and ~ operators for pattern matching

The only severe drawback of using the locale support in PostgreSQL is its speed. So use locale only if you actually need it. It should be noted in particular that selecting a non-C locale disables index optimizations for LIKE and ~ operators, which can make a huge difference in the speed of searches that use those operators.

5.1.3. Problems

If locale support doesn't work in spite of the explanation above, check that the locale support in your operating system is correctly configured. To check whether a given locale is installed and functional you can use Perl, for example. Perl has also support for locales and if a locale is broken **perl -v** will complain something like this:

```
$ export LC_CTYPE='not_exist'
$ 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").
```

Check that your locale files are in the right location. Possible locations include: `/usr/lib/locale` (Linux, Solaris), `/usr/share/locale` (Linux), `/usr/lib/nls/loc` (DUX 4.0). Check the locale man page of your system if you are not sure.

Check that PostgreSQL is actually using the locale that you think it is. `LC_COLLATE` and `LC_CTYPE` settings are determined at initdb time and cannot be changed without repeating initdb. Other locale settings including `LC_MESSAGES` and `LC_MONETARY` are determined by the environment the postmaster is started in, and can be changed with a simple postmaster restart. You can check the `LC_COLLATE` and `LC_CTYPE` settings of a database with the `contrib/pg_controldata` utility program.

The directory `src/test/locale` contains a test suite for PostgreSQL's locale support.

Client applications that handle server-side errors by parsing the text of the error message will obviously have problems when the server's messages are in a different language. If you create such an application you need to devise a plan to cope with this situation. The embedded SQL interface (ecpg) is also affected by this problem. It is currently recommended that servers interfacing with ecpg applications be configured to send messages in English.

Maintaining catalogs of message translations requires the on-going efforts of many volunteers that want to see PostgreSQL speak their preferred language well. If messages in your language is currently not available or fully translated, your assistance would be appreciated. If you want to help, refer to the *Developer's Guide* or write to the developers' mailing list.

5.2. Multibyte Support

Author: Tatsuo Ishii (<ishii@postgresql.org>), last updated 2000-03-22. Check Tatsuo's web site[1] for more information.

Multibyte (MB) support is intended to allow PostgreSQL to handle multiple-byte character sets such as EUC (Extended Unix Code), Unicode, and Mule internal code. With MB enabled you can use multibyte character sets in regular expressions (regexp), LIKE, and some other functions. The default encoding system is selected while initializing your PostgreSQL installation using initdb. Note that

1. http://www.sra.co.jp/people/t-ishii/PostgreSQL/

this can be overridden when you create a database using createdb or by using the SQL command **CREATE DATABASE**. So you can have multiple databases each with a different encoding system.

5.2.1. Enabling Multibyte Support

Run configure with the multibyte option:

```
./configure --enable-multibyte[=encoding_system]
```

where *encoding_system* can be one of the values in the following table:

Table 5-1. Character Set Encodings

Encoding	Description
SQL_ASCII	ASCII
EUC_JP	Japanese EUC
EUC_CN	Chinese EUC
EUC_KR	Korean EUC
EUC_TW	Taiwan EUC
UNICODE	Unicode (UTF-8)
MULE_INTERNAL	Mule internal code
LATIN1	ISO 8859-1 ECMA-94 Latin Alphabet No.1
LATIN2	ISO 8859-2 ECMA-94 Latin Alphabet No.2
LATIN3	ISO 8859-3 ECMA-94 Latin Alphabet No.3
LATIN4	ISO 8859-4 ECMA-94 Latin Alphabet No.4
LATIN5	ISO 8859-9 ECMA-128 Latin Alphabet No.5
LATIN6	ISO 8859-10 ECMA-144 Latin Alphabet No.6
LATIN7	ISO 8859-13 Latin Alphabet No.7
LATIN8	ISO 8859-14 Latin Alphabet No.8
LATIN9	ISO 8859-15 Latin Alphabet No.9
LATIN10	ISO 8859-16 ASRO SR 14111 Latin Alphabet No.10
ISO-8859-5	ECMA-113 Latin/Cyrillic
ISO-8859-6	ECMA-114 Latin/Arabic
ISO-8859-7	ECMA-118 Latin/Greek
ISO-8859-8	ECMA-121 Latin/Hebrew
KOI8	KOI8-R(U)
WIN	Windows CP1251
ALT	Windows CP866

Important: Before PostgreSQL7.2, LATIN5 mistakenly meant ISO 8859-5. From 7.2 on, LATIN5 means ISO 8859-9. If you have a LATIN5 database created on 7.1 or earlier and want to migrate to 7.2 (or later), you should be very careful about this change.

> **Important:** Not all APIs supports all the encodings listed above. For example, the PostgreSQL JDBC driver does not support `MULE_INTERNAL`, `LATIN6`, `LATIN8`, and `LATIN10`.

Here is an example of configuring PostgreSQL to use a Japanese encoding by default:

```
$ ./configure --enable-multibyte=EUC_JP
```

If the encoding system is omitted (`./configure --enable-multibyte`), `SQL_ASCII` is assumed.

5.2.2. Setting the Encoding

initdb defines the default encoding for a PostgreSQL installation. For example:

```
$ initdb -E EUC_JP
```

sets the default encoding to `EUC_JP` (Extended Unix Code for Japanese). Note that you can use `--encoding` instead of `-E` if you prefer to type longer option strings. If no `-E` or `--encoding` option is given, the encoding specified at configure time is used.

You can create a database with a different encoding:

```
$ createdb -E EUC_KR korean
```

will create a database named korean with `EUC_KR` encoding. Another way to accomplish this is to use a SQL command:

```
CREATE DATABASE korean WITH ENCODING = 'EUC_KR';
```

The encoding for a database is represented as an *encoding column* in the `pg_database` system catalog. You can see that by using the `-l` option or the **\l** command of **psql**.

```
$ psql -l
            List of databases
   Database    |  Owner  |   Encoding
---------------+---------+--------------
 euc_cn        | t-ishii | EUC_CN
 euc_jp        | t-ishii | EUC_JP
 euc_kr        | t-ishii | EUC_KR
 euc_tw        | t-ishii | EUC_TW
 mule_internal | t-ishii | MULE_INTERNAL
 regression    | t-ishii | SQL_ASCII
 template1     | t-ishii | EUC_JP
 test          | t-ishii | EUC_JP
 unicode       | t-ishii | UNICODE
(9 rows)
```

5.2.3. Automatic encoding translation between server and client

PostgreSQL supports an automatic encoding translation between server and client for some encodings. The available combinations are listed in Table 5-2.

Table 5-2. Client/Server Character Set Encodings

Server Encoding	Available Client Encodings
SQL_ASCII	SQL_ASCII, UNICODE, MULE_INTERNAL
EUC_JP	EUC_JP, SJIS, UNICODE, MULE_INTERNAL
EUC_TW	EUC_TW, BIG5, UNICODE, MULE_INTERNAL
LATIN1	LATIN1, UNICODE MULE_INTERNAL
LATIN2	LATIN2, WIN1250, UNICODE, MULE_INTERNAL
LATIN3	LATIN3, UNICODE MULE_INTERNAL
LATIN4	LATIN4, UNICODE MULE_INTERNAL
LATIN5	LATIN5, UNICODE MULE_INTERNAL
LATIN6	LATIN6, UNICODE MULE_INTERNAL
LATIN7	LATIN7, UNICODE MULE_INTERNAL
LATIN8	LATIN8, UNICODE MULE_INTERNAL
LATIN9	LATIN9, UNICODE MULE_INTERNAL
LATIN10	LATIN10, UNICODE MULE_INTERNAL
ISO_8859_5	ISO_8859_5, UNICODE
ISO_8859_6	ISO_8859_6, UNICODE
ISO_8859_7	ISO_8859_7, UNICODE
ISO_8859_8	ISO_8859_8, UNICODE
ISO_8859_9	ISO_8859_9, WIN, ALT, KOI8R, UNICODE, MULE_INTERNAL
UNICODE	EUC_JP, SJIS, EUC_KR, EUC_CN, EUC_TW, BIG5, LATIN1 to LATIN10, ISO_8859_5, ISO_8859_6, ISO_8859_7, ISO_8859_8, WIN, ALT, KOI8
MULE_INTERNAL	EUC_JP, SJIS, EUC_KR, EUC_CN, EUC_TW, BIG5, LATIN1 to LATIN5, WIN, ALT, WIN1250
KOI8	ISO_8859_9, WIN, ALT, KOI8, UNICODE, MULE_INTERNAL
WIN	ISO_8859_9, WIN, ALT, KOI8, UNICODE, MULE_INTERNAL
ALT	ISO_8859_9, WIN, ALT, KOI8, UNICODE, MULE_INTERNAL

To enable the automatic encoding translation, you have to tell PostgreSQL the encoding you would like to use in the client. There are several ways to accomplish this.

- Using the **\encoding** command in psql. **\encoding** allows you to change client encoding on the fly. For example, to change the encoding to SJIS, type:

  ```
  \encoding SJIS
  ```

- Using libpq functions. **\encoding** actually calls `PQsetClientEncoding()` for its purpose.

  ```
  int PQsetClientEncoding(PGconn *conn, const char *encoding)
  ```

  where *conn* is a connection to the server, and *encoding* is an encoding you want to use. If it

successfully sets the encoding, it returns 0, otherwise -1. The current encoding for this connection can be shown by using:

```
int PQclientEncoding(const PGconn *conn)
```

Note that it returns the encoding ID, not a symbolic string such as EUC_JP. To convert an encoding ID to an encoding name, you can use:

```
char *pg_encoding_to_char(int encoding_id)
```

- Using **SET CLIENT_ENCODING TO**. Setting the client encoding can be done with this SQL command:

  ```
  SET CLIENT_ENCODING TO 'encoding';
  ```

  Also you can use the SQL92 syntax SET NAMES for this purpose:

  ```
  SET NAMES 'encoding';
  ```

  To query the current client encoding:

  ```
  SHOW CLIENT_ENCODING;
  ```

  To return to the default encoding:

  ```
  RESET CLIENT_ENCODING;
  ```

- Using PGCLIENTENCODING. If environment variable PGCLIENTENCODING is defined in the client's environment, that client encoding is automatically selected when a connection to the server is made. (This can subsequently be overridden using any of the other methods mentioned above.)

5.2.4. About Unicode

An automatic encoding translation between Unicode and other encodings has been supported since PostgreSQL 7.1. For 7.1 it was not enabled by default. To enable this feature, run configure with the --enable-unicode-conversion option. Note that this requires the --enable-multibyte option also.

For 7.2, --enable-unicode-conversion is not necessary. The Unicode conversion functionality is automatically enabled if --enable-multibyte is specified.

5.2.5. What happens if the translation is not possible?

Suppose you choose EUC_JP for the server and LATIN1 for the client, then some Japanese characters cannot be translated into LATIN1. In this case, a letter that cannot be represented in the LATIN1 character set would be transformed as:

```
(HEXA DECIMAL)
```

5.2.6. References

These are good sources to start learning about various kinds of encoding systems.

ftp://ftp.ora.com/pub/examples/nutshell/ujip/doc/cjk.inf

Detailed explanations of EUC_JP, EUC_CN, EUC_KR, EUC_TW appear in section 3.2.

http://www.unicode.org/

The web site of the Unicode Consortium

RFC 2044

UTF-8 is defined here.

5.2.7. History

```
Dec 7, 2000
 * An automatic encoding translation between Unicode and other
   encodings are implemented
 * Changes above will appear in 7.1

May 20, 2000
 * SJIS UDC (NEC selection IBM kanji) support contributed
   by Eiji Tokuya
 * Changes above will appear in 7.0.1

Mar 22, 2000
 * Add new libpq functions PQsetClientEncoding, PQclientEncoding
 * ./configure --with-mb=EUC_JP
   now deprecated. use
   ./configure --enable-multibyte=EUC_JP
   instead
   * Add SQL_ASCII regression test case
 * Add SJIS User Defined Character (UDC) support
 * All of above will appear in 7.0

July 11, 1999
 * Add support for WIN1250 (Windows Czech) as a client encoding
   (contributed by Pavel Behal)
 * fix some compiler warnings (contributed by Tomoaki Nishiyama)

Mar 23, 1999
 * Add support for KOI8(KOI8-R), WIN(CP1251), ALT(CP866)
   (thanks Oleg Broytmann for testing)
 * Fix problem with MB and locale

Jan 26, 1999
 * Add support for Big5 for frontend encoding
   (you need to create a database with EUC_TW to use Big5)
 * Add regression test case for EUC_TW
   (contributed by Jonah Kuo <jonahkuo@mail.ttn.com.tw>)

Dec 15, 1998
 * Bugs related to SQL_ASCII support fixed

Nov 5, 1998
 * 6.4 release. In this version, pg_database has "encoding"
   column that represents the database encoding

Jul 22, 1998
 * determine encoding at initdb/createdb rather than compile time
 * support for PGCLIENTENCODING when issuing COPY command
```

```
 * support for SQL92 syntax "SET NAMES"
 * support for LATIN2-5
 * add UNICODE regression test case
 * new test suite for MB
 * clean up source files

Jun 5, 1998
 * add support for the encoding translation between the backend
   and the frontend
 * new command SET CLIENT_ENCODING etc. added
 * add support for LATIN1 character set
 * enhance 8-bit cleanliness

April 21, 1998 some enhancements/fixes
 * character_length(), position(), substring() are now aware of
   multi-byte characters
 * add octet_length()
 * add --with-mb option to configure
 * new regression tests for EUC_KR
     (contributed by Soonmyung Hong)
 * add some test cases to the EUC_JP regression test
 * fix problem in regress/regress.sh in case of System V
 * fix toupper(), tolower() to handle 8bit chars

Mar 25, 1998 MB PL2 is incorporated into PostgreSQL 6.3.1

Mar 10, 1998 PL2 released
 * add regression test for EUC_JP, EUC_CN and MULE_INTERNAL
 * add an English document (this file)
 * fix problems concerning 8-bit single byte characters

Mar 1, 1998 PL1 released
```

5.2.8. WIN1250 on Windows/ODBC

The WIN1250 character set on Windows client platforms can be used with PostgreSQL with locale support enabled.

The following should be kept in mind:

- Success depends on proper system locales. This has been tested with Red Hat 6.0 and Slackware 3.6, with the `cs_CZ.iso8859-2` locale.
- Never try to set the server's database encoding to WIN1250. Always use LATIN2 instead since there is no WIN1250 locale in Unix.
- The WIN1250 encoding is usable only for Windows ODBC clients. The characters are recoded on the fly, to be displayed and stored back properly.

WIN1250 on Windows/ODBC

1. Compile PostgreSQL with locale enabled and the server-side encoding set to `LATIN2`.
2. Set up your installation. Do not forget to create locale variables in your environment. For example (this may not be correct for *your* environment):

   ```
   LC_ALL=cs_CZ.ISO8859-2
   ```

3. You have to start the server with locales set!
4. Try it with the Czech language, and have it sort on a query.
5. Install ODBC driver for PostgreSQL on your Windows machine.
6. Set up your data source properly. Include this line in your ODBC configuration dialog in the field Connect Settings:

   ```
   SET CLIENT_ENCODING = 'WIN1250';
   ```

7. Now try it again, but in Windows with ODBC.

5.3. Single-byte character set recoding

You can set up this feature with the `--enable-recode` option to `configure`. This option was formerly described as "Cyrillic recode support" which doesn't express all its power. It can be used for *any* single-byte character set recoding.

This method uses a file `charset.conf` file located in the database directory (`PGDATA`). It's a typical configuration text file where spaces and newlines separate items and records and # specifies comments. Three keywords with the following syntax are recognized here:

```
BaseCharset      server_charset
RecodeTable      from_charset to_charset file_name
HostCharset      host_spec    host_charset
```

BaseCharset defines the encoding of the database server. All character set names are only used for mapping inside of `charset.conf` so you can freely use typing-friendly names.

RecodeTable records specify translation tables between server and client. The file name is relative to the `PGDATA` directory. The table file format is very simple. There are no keywords and characters are represented by a pair of decimal or hexadecimal (0x prefixed) values on single lines:

```
char_value   translated_char_value
```

HostCharset records define the client character set by IP address. You can use a single IP address, an IP mask range starting from the given address or an IP interval (e.g., 127.0.0.1, 192.168.1.100/24, 192.168.1.20-192.168.1.40).

The `charset.conf` file is always processed up to the end, so you can easily specify exceptions from the previous rules. In the `src/data/` directory you will find an example `charset.conf` and a few recoding tables.

As this solution is based on the client's IP address and character set mapping there are obviously some restrictions as well. You cannot use different encodings on the same host at the same time. It is also inconvenient when you boot your client hosts into multiple operating systems. Nevertheless, when these restrictions are not limiting and you do not need multibyte characters then it is a simple and effective solution.

Chapter 6. Managing Databases

A database is a named collection of SQL objects ("database objects"). Generally, every database object (tables, functions, etc.) belongs to one and only one database. (But there are a few system catalogs, for example `pg_database`, that belong to a whole installation and are accessible from each database within the installation.) An application that connects to the database server specifies in its connection request the name of the database it wants to connect to. It is not possible to access more than one database per connection. (But an application is not restricted in the number of connections it opens to the same or other databases.)

> **Note:** SQL calls databases "catalogs", but there is no difference in practice.

In order to create or drop databases, the PostgreSQL postmaster must be up and running (see Section 3.3).

6.1. Creating a Database

Databases are created with the query language command **CREATE DATABASE**:

```
CREATE DATABASE name
```

where *name* follows the usual rules for SQL identifiers. The current user automatically becomes the owner of the new database. It is the privilege of the owner of a database to remove it later on (which also removes all the objects in it, even if they have a different owner).

The creation of databases is a restricted operation. See Section 7.1.1 for how to grant permission.

Bootstrapping: Since you need to be connected to the database server in order to execute the **CREATE DATABASE** command, the question remains how the *first* database at any given site can be created. The first database is always created by the **initdb** command when the data storage area is initialized. (See Section 3.2.) By convention this database is called `template1`. So to create the first "real" database you can connect to `template1`.

The name "template1" is no accident: When a new database is created, the template database is essentially cloned. This means that any changes you make in `template1` are propagated to all subsequently created databases. This implies that you should not use the template database for real work, but when used judiciously this feature can be convenient. More details appear below.

As an extra convenience, there is also a program that you can execute from the shell to create new databases, `createdb`.

```
createdb dbname
```

createdb does no magic. It connects to the template1 database and issues the **CREATE DATABASE** command, exactly as described above. It uses the psql program internally. The reference page on **createdb** contains the invocation details. Note that **createdb** without any arguments will create a database with the current user name, which may or may not be what you want.

6.1.1. Template Databases

CREATE DATABASE actually works by copying an existing database. By default, it copies the standard system database named `template1`. Thus that database is the "template" from which new databases are made. If you add objects to `template1`, these objects will be copied into subsequently

created user databases. This behavior allows site-local modifications to the standard set of objects in databases. For example, if you install the procedural language `plpgsql` in `template1`, it will automatically be available in user databases without any extra action being taken when those databases are made.

There is a second standard system database named `template0`. This database contains the same data as the initial contents of `template1`, that is, only the standard objects predefined by your version of PostgreSQL. `template0` should never be changed after `initdb`. By instructing **CREATE DATABASE** to copy `template0` instead of `template1`, you can create a "virgin" user database that contains none of the site-local additions in `template1`. This is particularly handy when restoring a `pg_dump` dump: the dump script should be restored in a virgin database to ensure that one recreates the correct contents of the dumped database, without any conflicts with additions that may now be present in `template1`.

It is possible to create additional template databases, and indeed one might copy any database in an installation by specifying its name as the template for **CREATE DATABASE**. It is important to understand, however, that this is not (yet) intended as a general-purpose "COPY DATABASE" facility. In particular, it is essential that the source database be idle (no data-altering transactions in progress) for the duration of the copying operation. **CREATE DATABASE** will check that no backend processes (other than itself) are connected to the source database at the start of the operation, but this does not guarantee that changes cannot be made while the copy proceeds, which would result in an inconsistent copied database. Therefore, we recommend that databases used as templates be treated as read-only.

Two useful flags exist in `pg_database` for each database: `datistemplate` and `datallowconn`. `datistemplate` may be set to indicate that a database is intended as a template for **CREATE DATABASE**. If this flag is set, the database may be cloned by any user with `CREATEDB` privileges; if it is not set, only superusers and the owner of the database may clone it. If `datallowconn` is false, then no new connections to that database will be allowed (but existing sessions are not killed simply by setting the flag false). The `template0` database is normally marked `datallowconn` = `false` to prevent modification of it. Both `template0` and `template1` should always be marked with `datistemplate` = `true`.

After preparing a template database, or making any changes to one, it is a good idea to perform **VACUUM FREEZE** or **VACUUM FULL FREEZE** in that database. If this is done when there are no other open transactions in the same database, then it is guaranteed that all tuples in the database are "frozen" and will not be subject to transaction ID wraparound problems. This is particularly important for a database that will have `datallowconn` set to false, since it will be impossible to do routine maintenance **VACUUM**s on such a database. See Section 8.2.3 for more information.

> **Note:** `template1` and `template0` do not have any special status beyond the fact that the name `template1` is the default source database name for **CREATE DATABASE** and the default database-to-connect-to for various scripts such as `createdb`. For example, one could drop `template1` and recreate it from `template0` without any ill effects. This course of action might be advisable if one has carelessly added a bunch of junk in `template1`.

6.1.2. Alternative Locations

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

Alternative database locations are referenced by an environment variable which gives the absolute path to the intended storage location. This environment variable must be present in the server's environment, so it must have been defined before the server was started. (Thus, the set of available alternative locations is under the site administrator's control; ordinary users can't change it.) Any valid environment variable name may be used to reference an alternative location, although using variable names with a prefix of `PGDATA` is recommended to avoid confusion and conflict with other variables.

To create the variable in the environment of the server process you must first shut down the server, define the variable, initialize the data area, and finally restart the server. (See Section 3.6 and Section 3.3.) To set an environment variable, type

```
PGDATA2=/home/postgres/data
export PGDATA2
```

in Bourne shells, or

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

in csh or tcsh. You have to make sure that this environment variable is always defined in the server environment, otherwise you won't be able to access that database. Therefore you probably want to set it in some sort of shell start-up file or server start-up script.

To create a data storage area in `PGDATA2`, ensure that the containing directory (here, `/home/postgres`) already exists and is writable by the user account that runs the server (see Section 3.1). Then from the command line, type

```
initlocation PGDATA2
```

Then you can restart the server.

To create a database within the new location, use the command

```
CREATE DATABASE name WITH LOCATION = 'location'
```

where *`location`* is the environment variable you used, `PGDATA2` in this example. The **createdb** command has the option `-D` for this purpose.

Databases created in alternative locations can be accessed and dropped like any other database.

> **Note:** It can also be possible to specify absolute paths directly to the **CREATE DATABASE** command without defining environment variables. This is disallowed by default because it is a security risk. To allow it, you must compile PostgreSQL with the C preprocessor macro `ALLOW_ABSOLUTE_DBPATHS` defined. One way to do this is to run the compilation step like this:
>
> ```
> gmake CPPFLAGS=-DALLOW_ABSOLUTE_DBPATHS all
> ```

6.2. Destroying a Database

Databases are destroyed with the command **DROP DATABASE**:

```
DROP DATABASE name
```

Only the owner of the database (i.e., the user that created it), or a superuser, can drop a database. Dropping a database removes all objects that were contained within the database. The destruction of a database cannot be undone.

You cannot execute the **DROP DATABASE** command while connected to the victim database. You can, however, be connected to any other database, including the `template1` database, which would be the only option for dropping the last user database of a given cluster.

For convenience, there is also a shell program to drop databases:

```
dropdb dbname
```

(Unlike **createdb**, it is not the default action to drop the database with the current user name.)

Chapter 7. Database Users and Permissions

Managing database users and their privileges is in concept similar to managing users of a Unix operating system, but the details are not identical.

7.1. Database Users

Database users are conceptually completely separate from operating system users. In practice it might be convenient to maintain a correspondence, but this is not required. Database user names are global across a database cluster installation (and not per individual database). To create a user use the **CREATE USER** SQL command:

```
CREATE USER name
```

name follows the rules for SQL identifiers: either unadorned without special characters, or double-quoted. To remove an existing user, use the analogous **DROP USER** command.

For convenience, the shell scripts `createuser` and `dropuser` are provided as wrappers around these SQL commands.

In order to bootstrap the database system, a freshly initialized system always contains one predefined user. This user will have the fixed id 1, and by default (unless altered when running **initdb**) it will have the same name as the operating system user that initialized the area (and is presumably being used as the user that runs the server). Customarily, this user will be named postgres. In order to create more users you first have to connect as this initial user.

The user name to use for a particular database connection is indicated by the client that is initiating the connection request in an application-specific fashion. For example, the **psql** program uses the `-U` command line option to indicate the user to connect as. The set of database users a given client connection may connect as is determined by the client authentication setup, as explained in Chapter 4. (Thus, a client is not necessarily limited to connect as the user with the same name as its operating system user, in the same way a person is not constrained in its login name by her real name.)

7.1.1. User attributes

A database user may have a number of attributes that define its privileges and interact with the client authentication system.

superuser

A database superuser bypasses all permission checks. Also, only a superuser can create new users. To create a database superuser, use `CREATE USER name CREATEUSER`.

database creation

A user must be explicitly given permission to create databases (except for superusers, since those bypass all permission checks). To create such a user, use `CREATE USER name CREATEDB`.

password

A password is only significant if password authentication is used for client authentication. Database passwords are separate from operating system passwords. Specify a password upon user creation with `CREATE USER name PASSWORD 'string'`.

A user's attributes can be modified after creation with **ALTER USER**. See the reference pages for **CREATE USER** and **ALTER USER** for details.

7.2. Groups

As in Unix, groups are a way of logically grouping users to ease management of permissions: permissions can be granted to, or revoked from, a group as a whole. To create a group, use

```
CREATE GROUP name
```

To add users to or remove users from a group, use

```
ALTER GROUP name ADD USER uname1, ...
ALTER GROUP name DROP USER uname1, ...
```

7.3. Privileges

When a database object is created, it is assigned an owner. The owner is the user that executed the creation statement. There is currently no polished interface for changing the owner of a database object. By default, only an owner (or a superuser) can do anything with the object. In order to allow other users to use it, *privileges* must be granted.

There are several different privileges: `SELECT` (read), `INSERT` (append), `UPDATE` (write), `DELETE`, `RULE`, `REFERENCES` (foreign key), and `TRIGGER`. (See the **GRANT** manual page for more detailed information.) The right to modify or destroy an object is always the privilege of the owner only. To assign privileges, the **GRANT** command is used. So, if `joe` is an existing user, and `accounts` is an existing table, write access can be granted with

```
GRANT UPDATE ON accounts TO joe;
```

The user executing this command must be the owner of the table. To grant a privilege to a group, use

```
GRANT SELECT ON accounts TO GROUP staff;
```

The special "user" name `PUBLIC` can be used to grant a privilege to every user on the system. Writing `ALL` in place of a specific privilege specifies that all privileges will be granted.

To revoke a privilege, use the fittingly named **REVOKE** command:

```
REVOKE ALL ON accounts FROM PUBLIC;
```

The special privileges of the table owner (i.e., the right to do **DROP**, **GRANT**, **REVOKE**, etc) are always implicit in being the owner, and cannot be granted or revoked. But the table owner can choose to revoke his own ordinary privileges, for example to make a table read-only for himself as well as others.

7.4. Functions and Triggers

Functions and triggers allow users to insert code into the backend server that other users may execute without knowing it. Hence, both mechanisms permit users to *Trojan horse* others with relative impunity. The only real protection is tight control over who can define functions (e.g., write to relations with SQL fields) and triggers. Audit trails and alerters on the system catalogs `pg_class`, `pg_shadow` and `pg_group` are also possible.

Functions written in any language except SQL run inside the backend server process with the operating systems permissions of the database server daemon process. It is possible to change the server's

internal data structures from inside of trusted functions. Hence, among many other things, such functions can circumvent any system access controls. This is an inherent problem with user-defined C functions.

Chapter 8. Routine Database Maintenance Tasks

8.1. General Discussion

There are a few routine maintenance chores that must be performed on a regular basis to keep a PostgreSQL installation running smoothly. The tasks discussed here are repetitive in nature and can easily be automated using standard Unix tools such as `cron` scripts. But it is the database administrator's responsibility to set up appropriate scripts, and to check that they execute successfully.

One obvious maintenance task is creation of backup copies of the data on a regular schedule. Without a recent backup, you have no chance of recovery after a catastrophe (disk failure, fire, mistakenly dropping a critical table, etc). The backup and recovery mechanisms available in PostgreSQL are discussed at length in Chapter 9.

The other main category of maintenance task is periodic "vacuuming" of the database. This activity is discussed in Section 8.2.

Something else that might need periodic attention is log file management. This is discussed in Section 8.3.

PostgreSQL is low-maintenance compared to some other database products. Nonetheless, appropriate attention to these tasks will go far towards ensuring a pleasant and productive experience with the system.

8.2. Routine Vacuuming

PostgreSQL's **VACUUM** command must be run on a regular basis for several reasons:

1. To recover disk space occupied by updated or deleted rows.
2. To update data statistics used by the PostgreSQL query planner.
3. To protect against loss of very old data due to *transaction ID wraparound*.

The frequency and scope of **VACUUM**s performed for each of these reasons will vary depending on the needs of each installation. Therefore, database administrators must understand these issues and develop an appropriate maintenance strategy. This section concentrates on explaining the high-level issues; for details about command syntax and so on, see the **VACUUM** command reference page.

Beginning in PostgreSQL 7.2, the standard form of **VACUUM** can run in parallel with normal database operations (selects, inserts, updates, deletes, but not changes to table schemas). Routine vacuuming is therefore not nearly as intrusive as it was in prior releases, and it's not as critical to try to schedule it at low-usage times of day.

8.2.1. Recovering disk space

In normal PostgreSQL operation, an **UPDATE** or **DELETE** of a row does not immediately remove the old *tuple* (version of the row). This approach is necessary to gain the benefits of multiversion concurrency control (see the *User's Guide*): the tuple must not be deleted while it is still potentially visible to other transactions. But eventually, an outdated or deleted tuple is no longer of interest to any

transaction. The space it occupies must be reclaimed for reuse by new tuples, to avoid infinite growth of disk space requirements. This is done by running **VACUUM**.

Clearly, a table that receives frequent updates or deletes will need to be vacuumed more often than tables that are seldom updated. It may be useful to set up periodic cron tasks that vacuum only selected tables, skipping tables that are known not to change often. This is only likely to be helpful if you have both large heavily-updated tables and large seldom-updated tables --- the extra cost of vacuuming a small table isn't enough to be worth worrying about.

The standard form of **VACUUM** is best used with the goal of maintaining a fairly level steady-state usage of disk space. The standard form finds old tuples and makes their space available for re-use within the table, but it does not try very hard to shorten the table file and return disk space to the operating system. If you need to return disk space to the operating system you can use **VACUUM FULL** --- but what's the point of releasing disk space that will only have to be allocated again soon? Moderately frequent standard **VACUUM**s are a better approach than infrequent **VACUUM FULL**s for maintaining heavily-updated tables.

Recommended practice for most sites is to schedule a database-wide **VACUUM** once a day at a low-usage time of day, supplemented by more frequent vacuuming of heavily-updated tables if necessary. (If you have multiple databases in an installation, don't forget to vacuum each one; the `vacuumdb` script may be helpful.) Use plain **VACUUM**, not **VACUUM FULL**, for routine vacuuming for space recovery.

VACUUM FULL is recommended for cases where you know you have deleted the majority of tuples in a table, so that the steady-state size of the table can be shrunk substantially with **VACUUM FULL**'s more aggressive approach.

If you have a table whose contents are deleted completely every so often, consider doing it with **TRUNCATE** rather than using **DELETE** followed by **VACUUM**.

8.2.2. Updating planner statistics

The PostgreSQL query planner relies on statistical information about the contents of tables in order to generate good plans for queries. These statistics are gathered by the **ANALYZE** command, which can be invoked by itself or as an optional step in **VACUUM**. It is important to have reasonably accurate statistics, otherwise poor choices of plans may degrade database performance.

As with vacuuming for space recovery, frequent updates of statistics are more useful for heavily-updated tables than for seldom-updated ones. But even for a heavily-updated table, there may be no need for statistics updates if the statistical distribution of the data is not changing much. A simple rule of thumb is to think about how much the minimum and maximum values of the columns in the table change. For example, a `timestamp` column that contains the time of row update will have a constantly-increasing maximum value as rows are added and updated; such a column will probably need more frequent statistics updates than, say, a column containing URLs for pages accessed on a website. The URL column may receive changes just as often, but the statistical distribution of its values probably changes relatively slowly.

It is possible to run **ANALYZE** on specific tables and even just specific columns of a table, so the flexibility exists to update some statistics more frequently than others if your application requires it. In practice, however, the usefulness of this feature is doubtful. Beginning in PostgreSQL 7.2, **ANALYZE** is a fairly fast operation even on large tables, because it uses a statistical random sampling of the rows of a table rather than reading every single row. So it's probably much simpler to just run it over the whole database every so often.

> **Tip:** Although per-column tweaking of **ANALYZE** frequency may not be very productive, you may well find it worthwhile to do per-column adjustment of the level of detail of the statistics collected by **ANALYZE**. Columns that are heavily used in WHERE clauses and have highly irregular data distributions may require a finer-grain data histogram than other columns. See **ALTER TABLE SET STATISTICS**.

Recommended practice for most sites is to schedule a database-wide **ANALYZE** once a day at a low-usage time of day; this can usefully be combined with a nightly **VACUUM**. However, sites with relatively slowly changing table statistics may find that this is overkill, and that less-frequent **ANALYZE** runs are sufficient.

8.2.3. Preventing transaction ID wraparound failures

PostgreSQL's MVCC transaction semantics depend on being able to compare transaction ID (*XID*) numbers: a tuple with an insertion XID newer than the current transaction's XID is "in the future" and should not be visible to the current transaction. But since transaction IDs have limited size (32 bits at this writing) an installation that runs for a long time (more than 4 billion transactions) will suffer *transaction ID wraparound*: the XID counter wraps around to zero, and all of a sudden transactions that were in the past appear to be in the future --- which means their outputs become invisible. In short, catastrophic data loss. (Actually the data is still there, but that's cold comfort if you can't get at it.)

Prior to PostgreSQL 7.2, the only defense against XID wraparound was to re-**initdb** at least every 4 billion transactions. This of course was not very satisfactory for high-traffic sites, so a better solution has been devised. The new approach allows an installation to remain up indefinitely, without **initdb** or any sort of restart. The price is this maintenance requirement: *every table in the database must be vacuumed at least once every billion transactions*.

In practice this isn't an onerous requirement, but since the consequences of failing to meet it can be complete data loss (not just wasted disk space or slow performance), some special provisions have been made to help database administrators keep track of the time since the last **VACUUM**. The remainder of this section gives the details.

The new approach to XID comparison distinguishes two special XIDs, numbers 1 and 2 (`BootstrapXID` and `FrozenXID`). These two XIDs are always considered older than every normal XID. Normal XIDs (those greater than 2) are compared using modulo-2^{31} arithmetic. This means that for every normal XID, there are two billion XIDs that are "older" and two billion that are "newer"; another way to say it is that the normal XID space is circular with no endpoint. Therefore, once a tuple has been created with a particular normal XID, the tuple will appear to be "in the past" for the next two billion transactions, no matter which normal XID we are talking about. If the tuple still exists after more than two billion transactions, it will suddenly appear to be in the future. To prevent data loss, old tuples must be reassigned the XID `FrozenXID` sometime before they reach the two-billion-transactions-old mark. Once they are assigned this special XID, they will appear to be "in the past" to all normal transactions regardless of wraparound issues, and so such tuples will be good until deleted, no matter how long that is. This reassignment of XID is handled by **VACUUM**.

VACUUM's normal policy is to reassign `FrozenXID` to any tuple with a normal XID more than one billion transactions in the past. This policy preserves the original insertion XID until it is not likely to be of interest anymore (in fact, most tuples will probably live and die without ever being "frozen"). With this policy, the maximum safe interval between **VACUUM**s of any table is exactly one billion transactions: if you wait longer, it's possible that a tuple that was not quite old enough to be reassigned last time is now more than two billion transactions old and has wrapped around into the future --- ie, is lost to you. (Of course, it'll reappear after another two billion transactions, but that's no help.)

Since periodic **VACUUM**s are needed anyway for the reasons described earlier, it's unlikely that any table would not be vacuumed for as long as a billion transactions. But to help administrators ensure this constraint is met, **VACUUM** stores transaction ID statistics in the system table `pg_database`. In particular, the `datfrozenxid` field of a database's `pg_database` row is updated at the completion of any database-wide vacuum operation (ie, **VACUUM** that does not name a specific table). The value stored in this field is the freeze cutoff XID that was used by that **VACUUM** command. All normal XIDs older than this cutoff XID are guaranteed to have been replaced by `FrozenXID` within that database. A convenient way to examine this information is to execute the query

```
SELECT datname, age(datfrozenxid) FROM pg_database;
```

The `age` column measures the number of transactions from the cutoff XID to the current transaction's XID.

With the standard freezing policy, the `age` column will start at one billion for a freshly-vacuumed database. When the `age` approaches two billion, the database must be vacuumed again to avoid risk of wraparound failures. Recommended practice is to vacuum each database at least once every half-a-billion (500 million) transactions, so as to provide plenty of safety margin. To help meet this rule, each database-wide **VACUUM** automatically delivers a warning if there are any `pg_database` entries showing an `age` of more than 1.5 billion transactions, for example:

```
play=# vacuum;
NOTICE:  Some databases have not been vacuumed in 1613770184 transactions.
        Better vacuum them within 533713463 transactions,
        or you may have a wraparound failure.
VACUUM
```

VACUUM with the **FREEZE** option uses a more aggressive freezing policy: tuples are frozen if they are old enough to be considered good by all open transactions. In particular, if a **VACUUM FREEZE** is performed in an otherwise-idle database, it is guaranteed that *all* tuples in that database will be frozen. Hence, as long as the database is not modified in any way, it will not need subsequent vacuuming to avoid transaction ID wraparound problems. This technique is used by `initdb` to prepare the `template0` database. It should also be used to prepare any user-created databases that are to be marked `datallowconn = false` in `pg_database`, since there isn't any convenient way to vacuum a database that you can't connect to. Note that **VACUUM**'s automatic warning message about unvacuumed databases will ignore `pg_database` entries with `datallowconn = false`, so as to avoid giving false warnings about these databases; therefore it's up to you to ensure that such databases are frozen correctly.

8.3. Log File Maintenance

It's a good idea to save the database server's log output somewhere, rather than just routing it to `/dev/null`. The log output is invaluable when it comes time to diagnose problems. However, the log output tends to be voluminous (especially at higher debug levels) and you won't want to save it indefinitely. You need to "rotate" the log files so that new log files are started and old ones thrown away every so often.

If you simply direct the postmaster's stderr into a file, the only way to truncate the log file is to stop and restart the postmaster. This may be OK for development setups but you won't want to run a production server that way.

The simplest production-grade approach to managing log output is to send it all to syslog and let syslog deal with file rotation. To do this, make sure PostgreSQL was built with the `--enable-syslog` configure option, and set `syslog` to 2 (log to syslog only) in `postgresql.conf`. Then you can send a `SIGHUP` signal to the syslog daemon whenever you want to force it to start writing a new log file.

On many systems, however, syslog is not very reliable, particularly with large log messages; it may truncate or drop messages just when you need them the most. You may find it more useful to pipe the postmaster's stderr to some type of log rotation script. If you start the postmaster with pg_ctl, then the postmaster's stderr is already redirected to stdout, so you just need a pipe command:

```
pg_ctl start | logrotate
```

The PostgreSQL distribution doesn't include a suitable log rotation program, but there are many available on the net; one is included in the Apache distribution, for example.

Chapter 9. Backup and Restore

As everything that contains valuable data, PostgreSQL databases should be backed up regularly. While the procedure is essentially simple, it is important to have a basic understanding of the underlying techniques and assumptions.

There are two fundamentally different approaches to backing up PostgreSQL data:

- SQL dump
- File system level backup

9.1. SQL Dump

The idea behind the SQL-dump method is to generate a text file with SQL commands that, when fed back to the server, will recreate the database in the same state as it was at the time of the dump. PostgreSQL provides the utility program pg_dump for this purpose. The basic usage of this command is:

```
pg_dump dbname > outfile
```

As you see, pg_dump writes its results to the standard output. We will see below how this can be useful.

pg_dump is a regular PostgreSQL client application (albeit a particularly clever one). This means that you can do this backup procedure from any remote host that has access to the database. But remember that pg_dump does not operate with special permissions. In particular, you must have read access to all tables that you want to back up, so in practice you almost always have to be a database superuser.

To specify which database server pg_dump should contact, use the command line options `-h host` and `-p port`. The default host is the local host or whatever your `PGHOST` environment variable specifies. Similarly, the default port is indicated by the `PGPORT` environment variable or, failing that, by the compiled-in default. (Conveniently, the server will normally have the same compiled-in default.)

As any other PostgreSQL client application, pg_dump will by default connect with the database user name that is equal to the current Unix user name. To override this, either specify the `-U` option or set the environment variable `PGUSER`. Remember that pg_dump connections are subject to the normal client authentication mechanisms (which are described in Chapter 4).

Dumps created by pg_dump are internally consistent, that is, updates to the database while pg_dump is running will not be in the dump. pg_dump does not block other operations on the database while it is working. (Exceptions are those operations that need to operate with an exclusive lock, such as **VACUUM FULL**.)

> **Important:** When your database schema relies on OIDs (for instance as foreign keys) you must instruct pg_dump to dump the OIDs as well. To do this, use the `-o` command line option. “Large objects” are not dumped by default, either. See pg_dump’s command reference page if you use large objects.

9.1.1. Restoring the dump

The text files created by pg_dump are intended to be read in by the psql program. The general command form to restore a dump is

```
psql dbname < infile
```

where *infile* is what you used as *outfile* for the pg_dump command. The database *dbname* will not be created by this command, you must create it yourself from template0 before executing psql (e.g., with `createdb -T template0 dbname`). psql supports similar options to pg_dump for controlling the database server location and the user names. See its reference page for more information.

If the objects in the original database were owned by different users, then the dump will instruct psql to connect as each affected user in turn and then create the relevant objects. This way the original ownership is preserved. This also means, however, that all these users must already exist, and furthermore that you must be allowed to connect as each of them. It might therefore be necessary to temporarily relax the client authentication settings.

The ability of pg_dump and psql to write to or read from pipes makes it possible to dump a database directly from one server to another, for example

```
pg_dump -h host1 dbname | psql -h host2 dbname
```

> **Important:** The dumps produced by pg_dump are relative to template0. This means that any languages, procedures, etc. added to template1 will also be dumped by pg_dump. As a result, when restoring, if you are using a customized template1, you must create the empty database from template0, as in the example above.

9.1.2. Using pg_dumpall

The above mechanism is cumbersome and inappropriate when backing up an entire database cluster. For this reason the pg_dumpall program is provided. pg_dumpall backs up each database in a given cluster and also makes sure that the state of global data such as users and groups is preserved. The call sequence for pg_dumpall is simply

```
pg_dumpall > outfile
```

The resulting dumps can be restored with psql as described above. But in this case it is definitely necessary that you have database superuser access, as that is required to restore the user and group information.

9.1.3. Large Databases

> **Acknowledgement:** Originally written by Hannu Krosing (<hannu@trust.ee>) on 1999-06-19

Since PostgreSQL allows tables larger than the maximum file size on your system, it can be problematic to dump the table to a file, since the resulting file will likely be larger than the maximum size

allowed by your system. As pg_dump writes to the standard output, you can just use standard *nix tools to work around this possible problem.

Use compressed dumps. Use your favorite compression program, for example gzip.

```
pg_dump dbname | gzip > filename.gz
```

Reload with

```
createdb dbname
gunzip -c filename.gz | psql dbname
```

or

```
cat filename.gz | gunzip | psql dbname
```

Use split. This allows you to split the output into pieces that are acceptable in size to the underlying file system. For example, to make chunks of 1 megabyte:

```
pg_dump dbname | split -b 1m - filename
```

Reload with

```
createdb dbname
cat filename* | psql dbname
```

Use the custom dump format. If PostgreSQL was built on a system with the zlib compression library installed, the custom dump format will compress data as it writes it to the output file. For large databases, this will produce similar dump sizes to using **gzip**, but has the added advantage that the tables can be restored selectively. The following command dumps a database using the custom dump format:

```
pg_dump -Fc dbname > filename
```

See the pg_dump and pg_restore reference pages for details.

9.1.4. Caveats

pg_dump (and by implication pg_dumpall) has a few limitations which stem from the difficulty to reconstruct certain information from the system catalogs.

Specifically, the order in which pg_dump writes the objects is not very sophisticated. This can lead to problems for example when functions are used as column default values. The only answer is to manually reorder the dump. If you created circular dependencies in your schema then you will have more work to do.

For reasons of backward compatibility, pg_dump does not dump large objects by default. To dump large objects you must use either the custom or the TAR output format, and use the -b option in pg_dump. See the reference pages for details. The directory `contrib/pg_dumplo` of the PostgreSQL source tree also contains a program that can dump large objects.

Please familiarize yourself with the pg_dump reference page.

9.2. File system level backup

An alternative backup strategy is to directly copy the files that PostgreSQL uses to store the data in the database. In Section 3.2 it is explained where these files are located, but you have probably found them already if you are interested in this method. You can use whatever method you prefer for doing usual file system backups, for example

```
tar -cf backup.tar /usr/local/pgsql/data
```

There are two restrictions, however, which make this method impractical, or at least inferior to the pg_dump method:

1. The database server *must* be shut down in order to get a usable backup. Half-way measures such as disallowing all connections will not work as there is always some buffering going on. For this reason it is also not advisable to trust file systems that claim to support "consistent snapshots". Information about stopping the server can be found in Section 3.6.

   Needless to say that you also need to shut down the server before restoring the data.

2. If you have dug into the details of the file system layout you may be tempted to try to back up or restore only certain individual tables or databases from their respective files or directories. This will *not* work because the information contained in these files contains only half the truth. The other half is in the commit log files `pg_clog/*`, which contain the commit status of all transactions. A table file is only usable with this information. Of course it is also impossible to restore only a table and the associated `pg_clog` data because that will render all other tables in the database cluster useless.

Also note that the file system backup will not necessarily be smaller than an SQL dump. On the contrary, it will most likely be larger. (pg_dump does not need to dump the contents of indexes for example, just the commands to recreate them.)

9.3. Migration between releases

As a general rule, the internal data storage format is subject to change between releases of PostgreSQL. This does not apply to different "patch levels", these always have compatible storage formats. For example, releases 7.0.1, 7.1.2, and 7.2 are not compatible, whereas 7.1.1 and 7.1.2 are. When you update between compatible versions, then you can simply reuse the data area in disk by the new executables. Otherwise you need to "back up" your data and "restore" it on the new server, using pg_dump. (There are checks in place that prevent you from doing the wrong thing, so no harm can be done by confusing these things.) The precise installation procedure is not subject of this section, these details are in Chapter 1.

The least downtime can be achieved by installing the new server in a different directory and running both the old and the new servers in parallel, on different ports. Then you can use something like

```
pg_dumpall -p 5432 | psql -d template1 -p 6543
```

to transfer your data, or use an intermediate file if you want. Then you can shut down the old server and start the new server at the port the old one was running at. You should make sure that the database

is not updated after you run pg_dumpall, otherwise you will obviously lose that data. See Chapter 4 for information on how to prohibit access. In practice you probably want to test your client applications on the new setup before switching over.

If you cannot or do not want to run two servers in parallel you can do the back up step before installing the new version, bring down the server, move the old version out of the way, install the new version, start the new server, restore the data. For example:

```
pg_dumpall > backup
pg_ctl stop
mv /usr/local/pgsql /usr/local/pgsql.old
cd /usr/src/postgresql-7.2
gmake install
initdb -D /usr/local/pgsql/data
postmaster -D /usr/local/pgsql/data
psql < backup
```

See Chapter 3 about ways to start and stop the server and other details. The installation instructions will advise you of strategic places to perform these steps.

> **Note:** When you “move the old installation out of the way” it is no longer perfectly usable. Some parts of the installation contain information about where the other parts are located. This is usually not a big problem but if you plan on using two installations in parallel for a while you should assign them different installation directories at build time.

Chapter 10. Monitoring Database Activity

A database administrator frequently wonders "what is the system doing right now?" This chapter discusses how to find that out.

Several tools are available for monitoring database activity and analyzing performance. Most of this chapter is devoted to describing PostgreSQL's *statistics collector*, but one should not neglect regular Unix monitoring programs such as **ps** and **top**. Also, once one has identified a poorly-performing query, further investigation may be needed using PostgreSQL's **EXPLAIN** command. The *User's Guide* discusses **EXPLAIN** and other methods for understanding the behavior of an individual query.

10.1. Standard Unix Tools

On most platforms, PostgreSQL modifies its command title as reported by **ps**, so that individual server processes can readily be identified. A sample display is

```
$ ps auxww | grep ^postgres
postgres   960  0.0  1.1  6104 1480 pts/1    SN   13:17   0:00 postmaster -
i
postgres   963  0.0  1.1  7084 1472 pts/1    SN   13:17   0:00 postgres: stats buffer pr
cess
postgres   965  0.0  1.1  6152 1512 pts/1    SN   13:17   0:00 postgres: stats col-
lector process
postgres   998  0.0  2.3  6532 2992 pts/1    SN   13:18   0:00 postgres: tgl run-
bug 127.0.0.1 idle
postgres  1003  0.0  2.4  6532 3128 pts/1    SN   13:19   0:00 postgres: tgl re-
gression [local] SELECT waiting
postgres  1016  0.1  2.4  6532 3080 pts/1    SN   13:19   0:00 postgres: tgl re-
gression [local] idle in transaction
```

(The appropriate invocation of **ps** varies across different platforms, as do the details of what is shown. This example is from a recent Linux system.) The first process listed here is the *postmaster*, the master server process. The command arguments shown for it are the same ones given when it was launched. The next two processes implement the statistics collector, which will be described in detail in the next section. (These will not be present if you have set the system not to start the statistics collector.) Each of the remaining processes is a server process handling one client connection. Each such process sets its command line display in the form

`postgres: `*`user database host activity`*

The user, database, and connection source host items remain the same for the life of the client connection, but the activity indicator changes. The activity may be `idle` (ie, waiting for a client command), `idle in transaction` (waiting for client inside a BEGIN block), or a command type name such as `SELECT`. Also, `waiting` is attached if the server is presently waiting on a lock held by another server process. In the above example we can infer that process 1003 is waiting for process 1016 to complete its transaction and thereby release some lock or other.

> **Tip:** Solaris requires special handling. You must use **/usr/ucb/ps**, rather than **/bin/ps**. You also must use two **w** flags, not just one. In addition, your original invocation of the postmaster must have a shorter **ps** status display than that provided by each backend. If you fail to do all three things, the **ps** output for each backend will be the original postmaster command line.

10.2. Statistics Collector

PostgreSQL's *statistics collector* is a subsystem that supports collection and reporting of information about server activity. Presently, the collector can count accesses to tables and indexes in both disk-block and individual-row terms. It also supports determining the exact query currently being executed by other server processes.

10.2.1. Statistics Collection Configuration

Since collection of statistics adds some overhead to query execution, the system can be configured to collect or not collect information. This is controlled by configuration variables that are normally set in `postgresql.conf` (see Section 3.4 for details about setting configuration variables).

The variable `STATS_START_COLLECTOR` must be set to `true` for the statistics collector to be launched at all. This is the default and recommended setting, but it may be turned off if you have no interest in statistics and want to squeeze out every last drop of overhead. (The savings is likely to be small, however.) Note that this option cannot be changed while the server is running.

The variables `STATS_COMMAND_STRING`, `STATS_BLOCK_LEVEL`, and `STATS_ROW_LEVEL` control how much information is actually sent to the collector, and thus determine how much runtime overhead occurs. These respectively determine whether a server process sends its current command string, disk-block-level access statistics, and row-level access statistics to the collector. Normally these variables are set in `postgresql.conf` so that they apply to all server processes, but it is possible to turn them on or off in individual server processes using the **SET** command. (To prevent ordinary users from hiding their activity from the administrator, only superusers are allowed to change these variables with **SET**.)

> **Important:** Since the variables `STATS_COMMAND_STRING`, `STATS_BLOCK_LEVEL`, and `STATS_ROW_LEVEL` default to `false`, no statistics are actually collected in the default configuration! You must turn one or more of them on before you will get useful results from the statistical display functions.

10.2.2. Viewing Collected Statistics

Several predefined views are available to show the results of statistics collection. Alternatively, one can build custom views using the underlying statistics functions.

When using the statistics to monitor current activity, it is important to realize that the information does not update instantaneously. Each individual server process transmits new access counts to the collector just before waiting for another client command; so a query still in progress does not affect the displayed totals. Also, the collector itself emits new totals at most once per PGSTAT_STAT_INTERVAL (500 milliseconds by default). So the displayed totals lag behind actual activity.

Another important point is that when a server process is asked to display any of these statistics, it first fetches the most recent totals emitted by the collector process. It then continues to use this snapshot for all statistical views and functions until the end of its current transaction. So the statistics will appear not to change as long as you continue the current transaction. This is a feature, not a bug, because it allows you to perform several queries on the statistics and correlate the results without worrying that the numbers are changing underneath you. But if you want to see new results with each query, be sure to do the queries outside any transaction block.

Table 10-1. Standard Statistics Views

View Name	Description
`pg_stat_activity`	One row per server process, showing process PID, database, user, and current query. The current query column is only available to superusers; for others it reads as NULL. (Note that because of the collector's reporting delay, current query will only be up-to-date for long-running queries.)
`pg_stat_database`	One row per database, showing number of active backends, total transactions committed and total rolled back in that database, total disk blocks read, and total number of buffer hits (ie, block read requests avoided by finding the block already in buffer cache).
`pg_stat_all_tables`	For each table in the current database, total numbers of sequential and index scans, total numbers of tuples returned by each type of scan, and totals of tuple insertions, updates, and deletes.
`pg_stat_sys_tables`	Same as pg_stat_all_tables, except that only system tables are shown.
`pg_stat_user_tables`	Same as pg_stat_all_tables, except that only user tables are shown.
`pg_stat_all_indexes`	For each index in the current database, the total number of index scans that have used that index, the number of index tuples read, and the number of successfully fetched heap tuples (this may be less when there are index entries pointing to expired heap tuples).
`pg_stat_sys_indexes`	Same as pg_stat_all_indexes, except that only indexes on system tables are shown.
`pg_stat_user_indexes`	Same as pg_stat_all_indexes, except that only indexes on user tables are shown.
`pg_statio_all_tables`	For each table in the current database, the total number of disk blocks read from that table, the number of buffer hits, the numbers of disk blocks read and buffer hits in all the indexes of that table, the numbers of disk blocks read and buffer hits from the table's auxiliary TOAST table (if any), and the numbers of disk blocks read and buffer hits for the TOAST table's index.
`pg_statio_sys_tables`	Same as pg_statio_all_tables, except that only system tables are shown.
`pg_statio_user_tables`	Same as pg_statio_all_tables, except that only user tables are shown.

View Name	Description
pg_statio_all_indexes	For each index in the current database, the numbers of disk blocks read and buffer hits in that index.
pg_statio_sys_indexes	Same as pg_statio_all_indexes, except that only indexes on system tables are shown.
pg_statio_user_indexes	Same as pg_statio_all_indexes, except that only indexes on user tables are shown.
pg_statio_all_sequences	For each sequence object in the current database, the numbers of disk blocks read and buffer hits in that sequence.
pg_statio_sys_sequences	Same as pg_statio_all_sequences, except that only system sequences are shown. (Presently, no system sequences are defined, so this view is always empty.)
pg_statio_user_sequences	Same as pg_statio_all_sequences, except that only user sequences are shown.

The per-index statistics are particularly useful to determine which indexes are being used and how effective they are.

The pg_statio_ views are primarily useful to determine the effectiveness of the buffer cache. When the number of actual disk reads is much smaller than the number of buffer hits, then the cache is satisfying most read requests without invoking a kernel call.

Other ways of looking at the statistics can be set up by writing queries that use the same underlying statistics access functions as these standard views do. The per-database access functions accept a database OID to identify which database to report on. The per-table and per-index functions accept a table or index OID (note that only tables and indexes in the current database can be seen with these functions). The per-backend access functions accept a backend ID number, which ranges from one to the number of currently active backends.

Table 10-2. Statistics Access Functions

Function	Return Type	Description
pg_stat_get_db_numbackends(oid)	integer	Number of active backends in database
pg_stat_get_db_xact_commit(oid)	bigint	Transactions committed in database
pg_stat_get_db_xact_rollback(oid)	bigint	Transactions rolled back in database
pg_stat_get_db_blocks_fetched(oid)	bigint	Number of disk block fetch requests for database
pg_stat_get_db_blocks_hit(oid)	bigint	Number of disk block requests found in cache for database
pg_stat_get_numscans(oid)	bigint	Number of sequential scans done when argument is a table, or number of index scans done when argument is an index

Function	Return Type	Description
pg_stat_get_tuples_returned(oid)	bigint	Number of tuples read by sequential scans when argument is a table, or number of index tuples read when argument is an index
pg_stat_get_tuples_fetched(oid)	bigint	Number of valid (unexpired) table tuples fetched by sequential scans when argument is a table, or fetched by index scans using this index when argument is an index
pg_stat_get_tuples_inserted(oid)	bigint	Number of tuples inserted into table
pg_stat_get_tuples_updated(oid)	bigint	Number of tuples updated in table
pg_stat_get_tuples_deleted(oid)	bigint	Number of tuples deleted from table
pg_stat_get_blocks_fetched(oid)	bigint	Number of disk block fetch requests for table or index
pg_stat_get_blocks_hit(oid)	bigint	Number of disk block requests found in cache for table or index
pg_stat_get_backend_idset()	set of integer	Set of currently active backend IDs (from 1 to N where N is the number of active backends). See usage example below.
pg_stat_get_backend_pid(integer)	integer	PID of backend process
pg_stat_get_backend_dbid(integer)	oid	Database ID of backend process
pg_stat_get_backend_userid(integer)	oid	User ID of backend process
pg_stat_get_backend_activity(integer)	text	Current query of backend process (NULL if caller is not superuser)

Note: blocks_fetched minus blocks_hit gives the number of kernel read() calls issued for the table, index, or database; but the actual number of physical reads is usually lower due to kernel-level buffering.

The function pg_stat_get_backend_idset provides a convenient way to generate one row for each active backend. For example, to show the PIDs and current queries of all backends:

```
SELECT pg_stat_get_backend_pid(S.backendid) AS procpid,
       pg_stat_get_backend_activity(S.backendid) AS current_query
FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS S;
```

Chapter 11. Write-Ahead Logging (WAL)

Author: Vadim Mikheev and Oliver Elphick

11.1. General Description

Write Ahead Logging (WAL) is a standard approach to transaction logging. Its detailed description may be found in most (if not all) books about transaction processing. Briefly, WAL's central concept is that changes to data files (where tables and indexes reside) must be written only after those changes have been logged - that is, when log records have been flushed to permanent storage. When we follow this procedure, we do not need to flush data pages to disk on every transaction commit, because we know that in the event of a crash we will be able to recover the database using the log: any changes that have not been applied to the data pages will first be redone from the log records (this is roll-forward recovery, also known as REDO) and then changes made by uncommitted transactions will be removed from the data pages (roll-backward recovery - UNDO).

11.1.1. Immediate Benefits of WAL

The first obvious benefit of using WAL is a significantly reduced number of disk writes, since only the log file needs to be flushed to disk at the time of transaction commit; in multiuser environments, commits of many transactions may be accomplished with a single `fsync()` of the log file. Furthermore, the log file is written sequentially, and so the cost of syncing the log is much less than the cost of flushing the data pages.

The next benefit is consistency of the data pages. The truth is that, before WAL, PostgreSQL was never able to guarantee consistency in the case of a crash. Before WAL, any crash during writing could result in:

1. index tuples pointing to nonexistent table rows
2. index tuples lost in split operations
3. totally corrupted table or index page content, because of partially written data pages

Problems with indexes (problems 1 and 2) could possibly have been fixed by additional `fsync()` calls, but it is not obvious how to handle the last case without WAL; WAL saves the entire data page content in the log if that is required to ensure page consistency for after-crash recovery.

11.1.2. Future Benefits

In this first release of WAL, UNDO operation is not implemented, because of lack of time. This means that changes made by aborted transactions will still occupy disk space and that we still need a permanent `pg_clog` file to hold the status of transactions, since we are not able to re-use transaction identifiers. Once UNDO is implemented, `pg_clog` will no longer be required to be permanent; it will be possible to remove `pg_clog` at shutdown. (However, the urgency of this concern has decreased greatly with the adoption of a segmented storage method for `pg_clog` --- it is no longer necessary to keep old `pg_clog` entries around forever.)

With UNDO, it will also be possible to implement *savepoints* to allow partial rollback of invalid transaction operations (parser errors caused by mistyping commands, insertion of duplicate primary/unique

keys and so on) with the ability to continue or commit valid operations made by the transaction before the error. At present, any error will invalidate the whole transaction and require a transaction abort.

WAL offers the opportunity for a new method for database on-line backup and restore (BAR). To use this method, one would have to make periodic saves of data files to another disk, a tape or another host and also archive the WAL log files. The database file copy and the archived log files could be used to restore just as if one were restoring after a crash. Each time a new database file copy was made the old log files could be removed. Implementing this facility will require the logging of data file and index creation and deletion; it will also require development of a method for copying the data files (operating system copy commands are not suitable).

A difficulty standing in the way of realizing these benefits is that they require saving WAL entries for considerable periods of time (eg, as long as the longest possible transaction if transaction UNDO is wanted). The present WAL format is extremely bulky since it includes many disk page snapshots. This is not a serious concern at present, since the entries only need to be kept for one or two checkpoint intervals; but to achieve these future benefits some sort of compressed WAL format will be needed.

11.2. Implementation

WAL is automatically enabled from release 7.1 onwards. No action is required from the administrator with the exception of ensuring that the additional disk-space requirements of the WAL logs are met, and that any necessary tuning is done (see Section 11.3).

WAL logs are stored in the directory *$PGDATA*/pg_xlog, as a set of segment files, each 16 MB in size. Each segment is divided into 8 kB pages. The log record headers are described in access/xlog.h; record content is dependent on the type of event that is being logged. Segment files are given ever-increasing numbers as names, starting at 0000000000000000. The numbers do not wrap, at present, but it should take a very long time to exhaust the available stock of numbers.

The WAL buffers and control structure are in shared memory, and are handled by the backends; they are protected by lightweight locks. The demand on shared memory is dependent on the number of buffers. The default size of the WAL buffers is 8 buffers of 8 kB each, or 64 kB total.

It is of advantage if the log is located on another disk than the main database files. This may be achieved by moving the directory, pg_xlog, to another location (while the postmaster is shut down, of course) and creating a symbolic link from the original location in *$PGDATA* to the new location.

The aim of WAL, to ensure that the log is written before database records are altered, may be subverted by disk drives that falsely report a successful write to the kernel, when, in fact, they have only cached the data and not yet stored it on the disk. A power failure in such a situation may still lead to irrecoverable data corruption. Administrators should try to ensure that disks holding PostgreSQL's log files do not make such false reports.

11.2.1. Database Recovery with WAL

After a checkpoint has been made and the log flushed, the checkpoint's position is saved in the file pg_control. Therefore, when recovery is to be done, the backend first reads pg_control and then the checkpoint record; then it performs the REDO operation by scanning forward from the log position indicated in the checkpoint record. Because the entire content of data pages is saved in the log on the first page modification after a checkpoint, all pages changed since the checkpoint will be restored to a consistent state.

Using pg_control to get the checkpoint position speeds up the recovery process, but to handle possible corruption of pg_control, we should actually implement the reading of existing log segments in

reverse order -- newest to oldest -- in order to find the last checkpoint. This has not been implemented, yet.

11.3. WAL Configuration

There are several WAL-related parameters that affect database performance. This section explains their use. Consult Section 3.4 for details about setting configuration parameters.

There are two commonly used WAL functions: `LogInsert` and `LogFlush`. `LogInsert` is used to place a new record into the WAL buffers in shared memory. If there is no space for the new record, `LogInsert` will have to write (move to kernel cache) a few filled WAL buffers. This is undesirable because `LogInsert` is used on every database low level modification (for example, tuple insertion) at a time when an exclusive lock is held on affected data pages, so the operation needs to be as fast as possible. What is worse, writing WAL buffers may also force the creation of a new log segment, which takes even more time. Normally, WAL buffers should be written and flushed by a `LogFlush` request, which is made, for the most part, at transaction commit time to ensure that transaction records are flushed to permanent storage. On systems with high log output, `LogFlush` requests may not occur often enough to prevent WAL buffers being written by `LogInsert`. On such systems one should increase the number of WAL buffers by modifying the `WAL_BUFFERS` parameter. The default number of WAL buffers is 8. Increasing this value will correspondingly increase shared memory usage.

Checkpoints are points in the sequence of transactions at which it is guaranteed that the data files have been updated with all information logged before the checkpoint. At checkpoint time, all dirty data pages are flushed to disk and a special checkpoint record is written to the log file. As result, in the event of a crash, the recoverer knows from what record in the log (known as the redo record) it should start the REDO operation, since any changes made to data files before that record are already on disk. After a checkpoint has been made, any log segments written before the undo records are no longer needed and can be recycled or removed. (When WAL-based BAR is implemented, the log segments would be archived before being recycled or removed.)

The checkpoint maker is also able to create a few log segments for future use, so as to avoid the need for `LogInsert` or `LogFlush` to spend time in creating them. (If that happens, the entire database system will be delayed by the creation operation, so it's better if the files can be created in the checkpoint maker, which is not on anyone's critical path.) By default a new 16MB segment file is created only if more than 75% of the current segment has been used. This is inadequate if the system generates more than 4MB of log output between checkpoints. One can instruct the server to pre-create up to 64 log segments at checkpoint time by modifying the `WAL_FILES` configuration parameter.

The postmaster spawns a special backend process every so often to create the next checkpoint. A checkpoint is created every `CHECKPOINT_SEGMENTS` log segments, or every `CHECKPOINT_TIMEOUT` seconds, whichever comes first. The default settings are 3 segments and 300 seconds respectively. It is also possible to force a checkpoint by using the SQL command **CHECKPOINT**.

Reducing `CHECKPOINT_SEGMENTS` and/or `CHECKPOINT_TIMEOUT` causes checkpoints to be done more often. This allows faster after-crash recovery (since less work will need to be redone). However, one must balance this against the increased cost of flushing dirty data pages more often. In addition, to ensure data page consistency, the first modification of a data page after each checkpoint results in logging the entire page content. Thus a smaller checkpoint interval increases the volume of output to the log, partially negating the goal of using a smaller interval, and in any case causing more disk I/O.

The number of 16MB segment files will always be at least `WAL_FILES` + 1, and will normally not exceed `WAL_FILES` + 2 * `CHECKPOINT_SEGMENTS` + 1. This may be used to estimate space requirements for WAL. Ordinarily, when an old log segment file is no longer needed, it is recycled (renamed to become the next sequential future segment). If, due to a short-term peak of log output rate, there

are more than `WAL_FILES` + 2 * `CHECKPOINT_SEGMENTS` + 1 segment files, then unneeded segment files will be deleted instead of recycled until the system gets back under this limit. (If this happens on a regular basis, `WAL_FILES` should be increased to avoid it. Deleting log segments that will only have to be created again later is expensive and pointless.)

The `COMMIT_DELAY` parameter defines for how many microseconds the backend will sleep after writing a commit record to the log with `LogInsert` but before performing a `LogFlush`. This delay allows other backends to add their commit records to the log so as to have all of them flushed with a single log sync. No sleep will occur if `fsync` is not enabled or if fewer than `COMMIT_SIBLINGS` other backends are not currently in active transactions; this avoids sleeping when it's unlikely that any other backend will commit soon. Note that on most platforms, the resolution of a sleep request is ten milliseconds, so that any nonzero `COMMIT_DELAY` setting between 1 and 10000 microseconds will have the same effect. Good values for these parameters are not yet clear; experimentation is encouraged.

The `WAL_SYNC_METHOD` parameter determines how PostgreSQL will ask the kernel to force WAL updates out to disk. All the options should be the same as far as reliability goes, but it's quite platform-specific which one will be the fastest. Note that this parameter is irrelevant if `FSYNC` has been turned off.

Setting the `WAL_DEBUG` parameter to any nonzero value will result in each `LogInsert` and `LogFlush` WAL call being logged to standard error. At present, it makes no difference what the nonzero value is. This option may be replaced by a more general mechanism in the future.

Chapter 12. Disk Storage

This section needs to be written. Some information is in the FAQ. Volunteers? - thomas 1998-01-11

Chapter 13. Database Failures

Database failures (or the possibility of such) must be assumed to be lurking, ready to strike at some time in the future. A prudent database administrator will plan for the inevitability of failures of all possible kinds, and will have appropriate plans and procedures in place *before* the failure occurs.

Database recovery is necessary in the event of hardware or software failure. There are several categories of failures; some of these require relatively minor adjustments to the database, while others may depend on the existence of previously prepared database dumps and other recovery data sets. It should be emphasized that if your data is important and/or difficult to regenerate, then you should have considered and prepared for various failure scenarios.

13.1. Disk Filled

A filled data disk may result in subsequent corruption of database indexes, but not of the fundamental data tables. If the WAL files are on the same disk (as is the case for a default configuration) then a filled disk during database initialization may result in corrupted or incomplete WAL files. This failure condition is detected and the database will refuse to start up. You must free up additional space on the disk (or move the WAL area to another disk; see Section 11.3) and then restart the postmaster to recover from this condition.

13.2. Disk Failed

Failure of any disk (or of a logical storage device such as a RAID subsystem) involved with an active database will require that the database be recovered from a previously prepared database dump. This dump must be prepared using pg_dumpall, and updates to the database occurring after the database installation was dumped will be lost.

Chapter 14. Regression Tests

14.1. Introduction

The regression tests are a comprehensive set of tests for the SQL implementation in PostgreSQL. They test standard SQL operations as well as the extended capabilities of PostgreSQL. The test suite was originally developed by Jolly Chen and Andrew Yu, and was extensively revised and repackaged by Marc Fournier and Thomas Lockhart. From PostgreSQL 6.1 onward the regression tests are current for every official release.

14.2. Running the Tests

The regression test can be run against an already installed and running server, or using a temporary installation within the build tree. Furthermore, there is a "parallel" and a "sequential" mode for running the tests. The sequential method runs each test script in turn, whereas the parallel method starts up multiple server processes to run groups of tests in parallel. Parallel testing gives confidence that interprocess communication and locking are working correctly. For historical reasons, the sequential test is usually run against an existing installation and the parallel method against a temporary installation, but there are no technical reasons for this.

To run the regression tests after building but before installation, type

```
$ gmake check
```

in the top-level directory. (Or you can change to `src/test/regress` and run the command there.) This will first build several auxiliary files, such as platform-dependent "expected" files and some sample user-defined trigger functions, and then run the test driver script. At the end you should see something like

```
=====================
 All 77 tests passed.
=====================
```

or otherwise a note about what tests failed. See Section 14.3 below for more.

> **Note:** Because this test method runs a temporary server, it will not work when you are the root user (the server will not start as root). If you already did the build as root, you do not have to start all over. Instead, make the regression test directory writable by some other user, log in as that user, and restart the tests. For example,
>
> ```
> root# chmod -R a+w src/test/regress
> root# chmod -R a+w contrib/spi
> root# su - joeuser
> joeuser$ cd top-level build directory
> joeuser$ gmake check
> ```
>
> (The only possible "security risk" here is that other users might be able to alter the regression test results behind your back. Use common sense when managing user permissions.)
>
> Alternatively, run the tests after installation.

Tip: The parallel regression test starts quite a few processes under your user ID. Presently, the maximum concurrency is twenty parallel test scripts, which means sixty processes --- there's a backend, a psql, and usually a shell parent process for the psql for each test script. So if your system enforces a per-user limit on the number of processes, make sure this limit is at least seventy-five or so, else you may get random-seeming failures in the parallel test. If you are not in a position to raise the limit, you can edit the file `src/test/regress/parallel_schedule` to split the larger concurrent test sets into more manageable groups.

Tip: On some systems, the default Bourne-compatible shell (`/bin/sh`) gets confused when it has to manage too many child processes in parallel. This may cause the parallel test run to lock up or fail. In such cases, specify a different Bourne-compatible shell on the command line, for example:

```
$ gmake SHELL=/bin/ksh check
```

If no non-broken shell is available, you can alter the parallel test schedule as suggested above.

To run the tests after installation (see Chapter 1), initialize a data area and start the server, as explained in Chapter 3, then type

```
$ gmake installcheck
```

The tests will expect to contact the server at the local host and the default port number, unless directed otherwise by `PGHOST` and `PGPORT` environment variables.

14.3. Test Evaluation

Some properly installed and fully functional PostgreSQL installations can "fail" some of these regression tests due to platform-specific artifacts such as varying floating-point representation and time zone support. The tests are currently evaluated using a simple diff comparison against the outputs generated on a reference system, so the results are sensitive to small system differences. When a test is reported as "failed", always examine the differences between expected and actual results; you may well find that the differences are not significant. Nonetheless, we still strive to maintain accurate reference files across all supported platforms, so it can be expected that all tests pass.

The actual outputs of the regression tests are in files in the `src/test/regress/results` directory. The test script uses diff to compare each output file against the reference outputs stored in the `src/test/regress/expected` directory. Any differences are saved for your inspection in `src/test/regress/regression.diffs`. (Or you can run diff yourself, if you prefer.)

14.3.1. Error message differences

Some of the regression tests involve intentional invalid input values. Error messages can come from either the PostgreSQL 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 that can be validated by inspection.

14.3.2. Locale differences

The tests expect to run in plain "C" locale. This should not cause any problems when you run the tests against a temporary installation, since the regression test driver takes care to start the server in C locale. However, if you run the tests against an already-installed server that is using non-C locale settings, you may see differences caused by varying rules for string sort order, formatting of numeric and monetary values, and so forth.

In some locales the resulting differences are small and easily checked by inspection. However, in a locale that changes the rules for formatting of numeric values (typically by swapping the usage of commas and decimal points), entry of some data values will fail, resulting in extensive differences later in the tests where the missing data values are supposed to be used.

14.3.3. Date and time differences

Some of the queries in the `timestamp` test will fail if you run the test on the day of a daylight-saving time changeover, or the day before or after one. These queries assume that the intervals between midnight yesterday, midnight today and midnight tomorrow are exactly twenty-four hours -- which is wrong if daylight-saving time went into or out of effect meanwhile.

Most of the date and time results are dependent on the time zone environment. The reference files are generated for time zone `PST8PDT` (Berkeley, California) and there will be apparent failures if the tests are not run with that time zone setting. The regression test driver sets environment variable `PGTZ` to `PST8PDT`, which normally ensures proper results. However, your system must provide library support for the `PST8PDT` time zone, or the time zone-dependent tests will fail. To verify that your machine does have this support, type the following:

```
$ env TZ=PST8PDT date
```

The 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:

```
PGTZ='PST8PDT7,M04.01.0,M10.05.03'; export PGTZ
```

There appear to be some systems that do not accept the recommended syntax for explicitly setting the local time zone rules; you may need to use a different `PGTZ` setting on such machines.

Some systems using older time zone libraries fail to apply daylight-saving corrections to dates before 1970, causing pre-1970 PDT times to be displayed in PST instead. This will result in localized differences in the test results.

14.3.4. Floating-point differences

Some of the tests involve computing 64-bit (`double precision`) numbers from table columns. Differences in results involving mathematical functions of `double precision` columns have been observed. The float8 and geometry tests are particularly prone to small differences across platforms, or even with different compiler optimization options. 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 PostgreSQL code.

14.3.5. Polygon differences

Several of the tests involve operations on geographic data about the Oakland/Berkeley, California street map. The map data is expressed as polygons whose vertices are represented as pairs of `double precision` numbers (decimal latitude and longitude). Initially, some tables are created and loaded with geographic data, then some views are created that 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 following:

```
SELECT * from street;
SELECT * from iexit;
```

14.3.6. Row ordering differences

You might see differences in which the same rows are output in a different order than what appears in the expected file. In most cases this is not, strictly speaking, a bug. Most of the regression test scripts are not so pedantic as to use an ORDER BY for every single SELECT, and so their result row orderings are not well-defined according to the letter of the SQL specification. In practice, since we are looking at the same queries being executed on the same data by the same software, we usually get the same result ordering on all platforms, and so the lack of ORDER BY isn't a problem. Some queries do exhibit cross-platform ordering differences, however. (Ordering differences can also be triggered by non-C locale settings.)

Therefore, if you see an ordering difference, it's not something to worry about, unless the query does have an ORDER BY that your result is violating. But please report it anyway, so that we can add an ORDER BY to that particular query and thereby eliminate the bogus "failure" in future releases.

You might wonder why we don't order all the regress test queries explicitly to get rid of this issue once and for all. The reason is that that would make the regression tests less useful, not more, since they'd tend to exercise query plan types that produce ordered results to the exclusion of those that don't.

14.3.7. The "random" test

There is at least one case in the "random" test script that is intended to produce random results. This causes random to fail the regression test once in a while (perhaps once in every five to ten trials). Typing

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

should produce only one or a few lines of differences. You need not worry unless the random test always fails in repeated attempts. (On the other hand, if the random test is *never* reported to fail even in many trials of the regression tests, you probably *should* worry.)

14.4. Platform-specific comparison files

Since some of the tests inherently produce platform-specific results, we have provided a way to supply platform-specific result comparison files. Frequently, the same variation applies to multiple platforms;

rather than supplying a separate comparison file for every platform, there is a mapping file that defines which comparison file to use. So, to eliminate bogus test "failures" for a particular platform, you must choose or make a variant result file, and then add a line to the mapping file, which is `resultmap`.

Each line in the mapping file is of the form

```
testname/platformpattern=comparisonfilename
```

The test name is just the name of the particular regression test module. The platform pattern is a pattern in the style of expr (that is, a regular expression with an implicit `^` anchor at the start). It is matched against the platform name as printed by `config.guess` followed by `:gcc` or `:cc`, depending on whether you use the GNU compiler or the system's native compiler (on systems where there is a difference). The comparison file name is the name of the substitute result comparison file.

For example: some systems using older time zone libraries fail to apply daylight-saving corrections to dates before 1970, causing pre-1970 PDT times to be displayed in PST instead. This causes a few differences in the `horology` regression test. Therefore, we provide a variant comparison file, `horology-no-DST-before-1970.out`, which includes the results to be expected on these systems. To silence the bogus "failure" message on HPPA platforms, `resultmap` includes

```
horology/hppa=horology-no-DST-before-1970
```

which will trigger on any machine for which the output of **config.guess** begins with "`hppa`". Other lines in `resultmap` select the variant comparison file for other platforms where it's appropriate.

Appendix A. Release Notes

A.1. Release 7.2

Release date: 2002-02-04

A.1.1. Overview

This release improves PostgreSQL for use in high-volume applications.

Major changes in this release:

VACUUM

Vacuuming no longer locks tables, thus allowing normal user access during the vacuum. A new **VACUUM FULL** command does old-style vacuum by locking the table and shrinking the on-disk copy of the table.

Transactions

There is no longer a problem with installations that exceed four billion transactions.

OID's

OID's are now optional. Users can now create tables without OID's for cases where OID usage is excessive.

Optimizer

The system now computes histogram column statistics during **ANALYZE**, allowing much better optimizer choices.

Security

A new MD5 encryption option allows more secure storage and transfer of passwords. A new Unix-domain socket authentication option is available on Linux and BSD systems.

Statistics

Administrators can use the new table access statistics module to get fine-grained information about table and index usage.

Internationalization

Program and library messages can now be displayed in several languages.

A.1.2. Migration to version 7.2

A dump/restore using **pg_dump** is required for those wishing to migrate data from any previous release.

Observe the following incompatibilities:

- The semantics of the **VACUUM** command have changed in this release. You may wish to update your maintenance procedures accordingly.

- In this release, comparisons using `= NULL` will always return false (or NULL, more precisely). Previous releases automatically transformed this syntax to `IS NULL`. The old behavior can be re-enabled using a `postgresql.conf` parameter.
- The `pg_hba.conf` and `pg_ident.conf` configuration is now only reloaded after receiving a SIGHUP signal, not with each connection.
- The function `octet_length()` now returns the uncompressed data length.
- The date/time value `'current'` is no longer available. You will need to rewrite your applications.
- The `timestamp()` function is no longer available. Use `timestamp 'string'` instead, or `CAST`.

The `SELECT ... LIMIT #,#` syntax will be removed in the next release. You should change your queries to use separate LIMIT and OFFSET clauses, e.g. `LIMIT 10 OFFSET 20`.

A.1.3. Changes

A.1.3.1. Server Operation

Create temporary files in a separate directory (Bruce)
Delete orphaned temporary files on postmaster startup (Bruce)
Added unique indexes to some system tables (Tom)
System table operator reorganization (Oleg Bartunov, Teodor Sigaev, Tom)
Renamed pg_log to pg_clog (Tom)
Enable SIGTERM, SIGQUIT to kill backends (Jan)
Removed compile-time limit on number of backends (Tom)
Better cleanup for semaphore resource failure (Tatsuo, Tom)
Allow safe transaction ID wraparound (Tom)
Removed OID's from some system tables (Tom)
Removed "triggered data change violation" error check (Tom)
SPI portal creation of prepared/saved plans (Jan)
Allow SPI column functions to work for system columns (Tom)
Long value compression improvement (Tom)
Statistics collector for table, index access (Jan)
Truncate extra-long sequence names to a reasonable value (Tom)
Measure transaction times in milliseconds (Thomas)
Fix TID sequential scans (Hiroshi)
Superuser ID now fixed at 1 (Peter E)
New pg_ctl "reload" option (Tom)

A.1.3.2. Performance

Optimizer improvements (Tom)
New histogram column statistics for optimizer (Tom)
Reuse write-ahead log files rather than discarding them (Tom)
Cache improvements (Tom)
IS NULL, IS NOT NULL optimizer improvement (Tom)
Improve lock manager to reduce lock contention (Tom)
Keep relcache entries for index access support functions (Tom)
Allow better selectivity with NaN and infinities in NUMERIC (Tom)

R-tree performance improvements (Kenneth Been)
B-tree splits more efficient (Tom)

A.1.3.3. Privileges

Change UPDATE, DELETE permissions to be distinct (Peter E)
New REFERENCES, TRIGGER privileges (Peter E)
Allow GRANT/REVOKE to/from more than one user at a time (Peter E)
New has_table_privilege() function (Joe Conway)
Allow non-superuser to vacuum database (Tom)
New SET SESSION AUTHORIZATION command (Peter E)
Fix bug in privilege modifications on newly created tables (Tom)
Disallow access to pg_statistic for non-superuser, add user-accessible views (Tom)

A.1.3.4. Client Authentication

Fork postmaster before doing authentication to prevent hangs (Peter E)
Add ident authentication over Unix domain sockets on Linux, *BSD (Helge Bahmann, Oliver Elphick, Teodor Sigaev, Br
Add a password authentication method that uses MD5 encryption (Bruce)
Allow encryption of stored passwords using MD5 (Bruce)
PAM authentication (Dominic J. Eidson)
Load pg_hba.conf and pg_ident.conf only on startup and SIGHUP (Bruce)

A.1.3.5. Server Configuration

Interpretation of some time zone abbreviations as Australian rather than North American now settable at run time (Bruce)
New parameter to set default transaction isolation level (Peter E)
New parameter to enable conversion of "expr = NULL" into "expr IS NULL", off by default (Peter E)
New parameter to control memory usage by VACUUM (Tom)
New parameter to set client authentication timeout (Tom)
New parameter to set maximum number of open files (Tom)

A.1.3.6. Queries

Statements added by INSERT rules now execute after the INSERT (Jan)
Prevent unadorned relation names in target list (Bruce)
NULLs now sort after all normal values in ORDER BY (Tom)
New IS UNKNOWN, IS NOT UNKNOWN Boolean tests (Tom)
New SHARE UPDATE EXCLUSIVE lock mode (Tom)
New EXPLAIN ANALYZE command that shows run times and row counts (Martijn van Oosterhout)
Fix problem with LIMIT and subqueries (Tom)
Fix for LIMIT, DISTINCT ON pushed into subqueries (Tom)
Fix nested EXCEPT/INTERSECT (Tom)

A.1.3.7. Schema Manipulation

Fix SERIAL in temporary tables (Bruce)
Allow temporary sequences (Bruce)
Sequences now use int8 internally (Tom)
New SERIAL8 creates int8 columns with sequences, default still SERIAL4 (Tom)
Make OIDs optional using WITHOUT OIDS (Tom)
Add %TYPE syntax to CREATE TYPE (Ian Lance Taylor)
Add ALTER TABLE / DROP CONSTRAINT for CHECK constraints (Christopher Kings-Lynne)
New CREATE OR REPLACE FUNCTION to alter existing function (preserving the function OID) (Gavin Sherry)
Add ALTER TABLE / ADD [UNIQUE | PRIMARY] (Christopher Kings-Lynne)
Allow column renaming in views
Make ALTER TABLE / RENAME COLUMN update column names of indexes (Brent Verner)
Fix for ALTER TABLE / ADD CONSTRAINT ... CHECK with inherited tables (Stephan Szabo)
ALTER TABLE RENAME update foreign-key trigger arguments correctly (Brent Verner)
DROP AGGREGATE and COMMENT ON AGGREGATE now accept an aggtype (Tom)
Add automatic return type data casting for SQL functions (Tom)
Allow GiST indexes to handle NULLs and multikey indexes (Oleg Bartunov, Teodor Sigaev, Tom)
Enable partial indexes (Martijn van Oosterhout)

A.1.3.8. Utility Commands

Add RESET ALL, SHOW ALL (Marko Kreen)
CREATE/ALTER USER/GROUP now allow options in any order (Vince)
Add LOCK A, B, C functionality (Neil Padgett)
New ENCRYPTED/UNENCRYPTED option to CREATE/ALTER USER (Bruce)
New light-weight VACUUM does not lock table; old semantics are available as VACUUM FULL (Tom)
Disable COPY TO/FROM on views (Bruce)
COPY DELIMITERS string must be exactly one character (Tom)
VACUUM warning about index tuples fewer than heap now only appears when appropriate (Martijn van Oosterhout)
Fix permission checks for CREATE INDEX (Tom)
Disallow inappropriate use of CREATE/DROP INDEX/TRIGGER/VIEW (Tom)

A.1.3.9. Data Types and Functions

SUM(), AVG(), COUNT() now uses int8 internally for speed (Tom)
Add convert(), convert2() (Tatsuo)
New function bit_length() (Peter E)
Make the "n" in CHAR(n)/VARCHAR(n) represents letters, not bytes (Tatsuo)
CHAR(), VARCHAR() now reject strings that are too long (Peter E)
BIT VARYING now rejects bit strings that are too long (Peter E)
BIT now rejects bit strings that do not match declared size (Peter E)
INET, CIDR text conversion functions (Alex Pilosov)
INET, CIDR operators << and <<= indexable (Alex Pilosov)
Bytea \### now requires valid three digit octal number
Bytea comparison improvements, now supports =, <>, >, >=, <, and <=
Bytea now supports B-tree indexes
Bytea now supports LIKE, LIKE...ESCAPE, NOT LIKE, NOT LIKE...ESCAPE
Bytea now supports concatenation
New bytea functions: position, substring, trim, btrim, and length

New encode() function mode, "escaped", converts minimally escaped bytea to/from text
Add pg_database_encoding_max_length() (Tatsuo)
Add pg_client_encoding() function (Tatsuo)
now() returns time with millisecond precision (Thomas)
New TIMESTAMP WITHOUT TIMEZONE data type (Thomas)
Add ISO date/time specification with "T", yyyy-mm-ddThh:mm:ss (Thomas)
New xid/int comparison functions (Hiroshi)
Add precision to TIME, TIMESTAMP, and INVERVAL data types (Thomas)
Modify type coercion logic to attempt binary-compatible functions first (Tom)
New encode() function installed by default (Marko Kreen)
Improved to_*() conversion functions (Karel Zak)
Optimize LIKE/ILIKE when using single-byte encodings (Tatsuo)
New functions in contrib/pgcrypto: crypt(), hmac(), encrypt(), gen_salt() (Marko Kreen)
Correct description of translate() function (Bruce)
Add INTERVAL argument for SET TIME ZONE (Thomas)
Add INTERVAL YEAR TO MONTH (etc.) syntax (Thomas)
Optimize length functions when using single-byte encodings (Tatsuo)
Fix path_inter, path_distance, path_length, dist_ppath to handle closed paths (Curtis Barrett, Tom)
octet_length(text) now returns non-compressed length (Tatsuo, Bruce)
Handle "July" full name in date/time literals (Greg Sabino Mullane)
Some datatype() function calls now evaluated differently
Add support for Julian and ISO time specifications (Thomas)

A.1.3.10. Internationalization

National language support in psql, pg_dump, libpq, and server (Peter E)
Message translations in Chinese (simplified, traditional), Czech, French, German, Hungarian, Russian, Swedish (Peter E, Serguei A. Mokhov, Karel Zak, Weiping He, Zhenbang Wei, Kovacs Zoltan)
Make trim, ltrim, rtrim, btrim, lpad, rpad, translate multibyte aware (Tatsuo)
Add LATIN5,6,7,8,9,10 support (Tatsuo)
Add ISO 8859-5,6,7,8 support (Tatsuo)
Correct LATIN5 to mean ISO-8859-9, not ISO-8859-5 (Tatsuo)
Make mic2ascii() non-ASCII aware (Tatsuo)
Reject invalid multibyte character sequences (Tatsuo)

A.1.3.11. PL/pgSQL

Now uses portals for SELECT loops, allowing huge result sets (Jan)
CURSOR and REFCURSOR support (Jan)
Can now return open cursors (Jan)
Add ELSEIF (Klaus Reger)
Improve PL/pgSQL error reporting, including location of error (Tom)
Allow IS or FOR key words in cursor declaration, for compatibility (Bruce)
Fix for SELECT ... FOR UPDATE (Tom)
Fix for PERFORM returning multiple rows (Tom)
Make PL/pgSQL use the server's type coercion code (Tom)
Memory leak fix (Jan, Tom)
Make trailing semicolon optional (Tom)

A.1.3.12. PL/Perl

New untrusted PL/Perl (Alex Pilosov)
PL/Perl is now built on some platforms even if libperl is not shared (Peter E)

A.1.3.13. PL/Tcl

Now reports errorInfo (Vsevolod Lobko)
Add spi_lastoid function (bob@redivi.com)

A.1.3.14. PL/Python

...is new (Andrew Bosma)

A.1.3.15. Psql

\d displays indexes in unique, primary groupings (Christopher Kings-Lynne)
Allow trailing semicolons in backslash commands (Greg Sabino Mullane)
Read password from /dev/tty if possible
Force new password prompt when changing user and database (Tatsuo, Tom)
Format the correct number of columns for Unicode (Patrice)

A.1.3.16. Libpq

New function PQescapeString() to escape quotes in command strings (Florian Weimer)
New function PQescapeBytea() escapes binary strings for use as SQL string literals

A.1.3.17. JDBC

Return OID of INSERT (Ken K)
Handle more data types (Ken K)
Handle single quotes and newlines in strings (Ken K)
Handle NULL variables (Ken K)
Fix for time zone handling (Barry Lind)
Improved Druid support
Allow eight-bit characters with non-multibyte server (Barry Lind)
Support BIT, BINARY types (Ned Wolpert)
Reduce memory usage (Michael Stephens, Dave Cramer)
Update DatabaseMetaData (Peter E)
Add DatabaseMetaData.getCatalogs() (Peter E)
Encoding fixes (Anders Bengtsson)
Get/setCatalog methods (Jason Davies)
DatabaseMetaData.getColumns() now returns column defaults (Jason Davies)
DatabaseMetaData.getColumns() performance improvement (Jeroen van Vianen)
Some JDBC1 and JDBC2 merging (Anders Bengtsson)
Transaction performance improvements (Barry Lind)

Array fixes (Greg Zoller)
Serialize addition
Fix batch processing (Rene Pijlman)
ExecSQL method reorganization (Anders Bengtsson)
GetColumn() fixes (Jeroen van Vianen)
Fix isWriteable() function (Rene Pijlman)
Improved passage of JDBC2 conformance tests (Rene Pijlman)
Add bytea type capability (Barry Lind)
Add isNullable() (Rene Pijlman)
JDBC date/time test suite fixes (Liam Stewart)
Fix for SELECT 'id' AS xxx FROM table (Dave Cramer)
Fix DatabaseMetaData to show precision properly (Mark Lillywhite)
New getImported/getExported keys (Jason Davies)
MD5 password encryption support (Jeremy Wohl)
Fix to actually use type cache (Ned Wolpert)

A.1.3.18. ODBC

Remove query size limit (Hiroshi)
Remove text field size limit (Hiroshi)
Fix for SQLPrimaryKeys in multibyte mode (Hiroshi)
Allow ODBC procedure calls (Hiroshi)
Improve boolean handing (Aidan Mountford)
Most configuration options on setable via DSN (Hiroshi)
Multibyte, performance fixes (Hiroshi)
Allow driver to be used with iODBC or unixODBC (Peter E)
MD5 password encryption support (Bruce)
Add more compatibility functions to odbc.sql (Peter E)

A.1.3.19. ECPG

EXECUTE ... INTO implemented (Christof Petig)
Multiple row descriptor support (e.g. CARDINALITY) (Christof Petig)
Fix for GRANT parameters (Lee Kindness)
Fix INITIALLY DEFERRED bug
Various bug fixes (Michael, Christof Petig)
Auto allocation for indicator variable arrays (int *ind_p=NULL)
Auto allocation for string arrays (char **foo_pp=NULL)
ECPGfree_auto_mem fixed
All function names with external linkage are now prefixed by ECPG
Fixes for arrays of structures (Michael)

A.1.3.20. Misc. Interfaces

Python fix fetchone() (Gerhard Haring)
Use UTF, Unicode in Tcl where appropriate (Vsevolod Lobko, Reinhard Max)
Add Tcl COPY TO/FROM (ljb)
Prevent output of default index op class in pg_dump (Tom)

Fix libpgeasy memory leak (Bruce)

A.1.3.21. Build and Install

Configure, dynamic loader, and shared library fixes (Peter E)
Fixes in QNX 4 port (Bernd Tegge)
Fixes in Cygwin and Win32 ports (Jason Tishler, Gerhard Haring, Dmitry Yurtaev, Darko Prenosil, Mikhail Terekhov)
Fix for Win32 socket communication failures (Magnus, Mikhail Terekhov)
Hurd compile fix (Oliver Elphick)
BeOS fixes (Cyril Velter)
Remove configure --enable-unicode-conversion, now enabled by multibyte (Tatsuo)
AIX fixes (Tatsuo, Andreas)
Fix parallel make (Peter E)
Install SQL language manual pages into OS-specific directories (Peter E)
Rename config.h to pg_config.h (Peter E)
Reorganize installation layout of header files (Peter E)

A.1.3.22. Source Code

Remove SEP_CHAR (Bruce)
New GUC hooks (Tom)
Merge GUC and command line handling (Marko Kreen)
Remove EXTEND INDEX (Martijn van Oosterhout, Tom)
New pgjindent utility to indent java code (Bruce)
Remove define of true/false when compiling under C++ (Leandro Fanzone, Tom)
pgindent fixes (Bruce, Tom)
Replace strcasecmp() with strcmp() where appropriate (Peter E)
Dynahash portability improvements (Tom)
Add ’volatile’ usage in spinlock structures
Improve signal handling logic (Tom)

A.1.3.23. Contrib

New contrib/rtree_gist (Oleg Bartunov, Teodor Sigaev)
New contrib/tsearch full-text indexing (Oleg, Teodor Sigaev)
Add contrib/dblink for remote database access (Joe Conway)
contrib/ora2pg Oracle conversion utility (Gilles Darold)
contrib/xml XML conversion utility (John Gray)
contrib/fulltextindex fixes (Christopher Kings-Lynne)
New contrib/fuzzystrmatch with levenshtein and metaphone, soundex merged (Joe Conway)
Add contrib/intarray boolean queries, binary search, fixes (Oleg Bartunov)
New pg_upgrade utility (Bruce)
Add new pg_resetxlog options (Bruce, Tom)

A.2. Release 7.1.3

Release date: 2001-08-15

A.2.1. Migration to version 7.1.3

A dump/restore is *not* required for those running 7.1.X.

A.2.2. Changes

```
Remove unused WAL segements of large transactions (Tom)
Multiaction rule fix (Tom)
PL/pgSQL memory allocation fix (Jan)
VACUUM buffer fix (Tom)
Regression test fixes (Tom)
pg_dump fixes for GRANT/REVOKE/comments on views, user-defined types (Tom)
Fix subselects with DISTINCT ON or LIMIT (Tom)
BeOS fix
Disable COPY TO/FROM a view (Tom)
Cygwin build (Jason Tishler)
```

A.3. Release 7.1.2

Release date: 2001-05-11

This has one fix from 7.1.1.

A.3.1. Migration to version 7.1.2

A dump/restore is *not* required for those running 7.1.X.

A.3.2. Changes

```
Fix PL/pgSQL SELECTs when returning no rows
Fix for psql backslash core dump
Referential integrity permission fix
Optimizer fixes
pg_dump cleanups
```

A.4. Release 7.1.1

Release date: 2001-05-05

This has a variety of fixes from 7.1.

A.4.1. Migration to version 7.1.1

A dump/restore is *not* required for those running 7.1.

A.4.2. Changes

```
Fix for numeric MODULO operator (Tom)
pg_dump fixes (Philip)
pg_dump can dump 7.0 databases (Philip)
readline 4.2 fixes (Peter E)
JOIN fixes (Tom)
AIX, MSWIN, VAX, N32K fixes (Tom)
Multibytes fixes (Tom)
Unicode fixes (Tatsuo)
Optimizer improvements (Tom)
Fix for whole tuples in functions (Tom)
Fix for pg_ctl and option strings with spaces (Peter E)
ODBC fixes (Hiroshi)
EXTRACT can now take string argument (Thomas)
Python fixes (Darcy)
```

A.5. Release 7.1

Release date: 2001-04-13

This release focuses on removing limitations that have existed in the PostgreSQL code for many years.

Major changes in this release:

Write-ahead Log (WAL)

> To maintain database consistency in case of an operating system crash, previous releases of PostgreSQL have forced all data modifications to disk before each transaction commit. With WAL, only one log file must be flushed to disk, greatly improving performance. If you have been using -F in previous releases to disable disk flushes, you may want to consider discontinuing its use.

TOAST

TOAST - Previous releases had a compiled-in row length limit, typically 8k - 32k. This limit made storage of long text fields difficult. With TOAST, long rows of any length can be stored with good performance.

Outer Joins

We now support outer joins. The UNION/NOT IN workaround for outer joins is no longer required. We use the SQL92 outer join syntax.

Function Manager

The previous C function manager did not handle NULLs properly, nor did it support 64-bit CPU's (Alpha). The new function manager does. You can continue using your old custom functions, but you may want to rewrite them in the future to use the new function manager call interface.

Complex Queries

A large number of complex queries that were unsupported in previous releases now work. Many combinations of views, aggregates, UNION, LIMIT, cursors, subqueries, and inherited tables now work properly. Inherited tables are now accessed by default. Subqueries in FROM are now supported.

A.5.1. Migration to version 7.1

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release.

A.5.2. Changes

```
Bug Fixes
---------
Many multi-byte/Unicode/locale fixes (Tatsuo and others)
More reliable ALTER TABLE RENAME (Tom)
Kerberos V fixes (David Wragg)
Fix for INSERT INTO...SELECT where targetlist has subqueries (Tom)
Prompt username/password on standard error (Bruce)
Large objects inv_read/inv_write fixes (Tom)
Fixes for to_char(), to_date(), to_ascii(), and to_timestamp() (Karel,
    Daniel Baldoni)
Prevent query expressions from leaking memory (Tom)
Allow UPDATE of arrays elements (Tom)
Wake up lock waiters during cancel (Hiroshi)
Fix rare cursor crash when using hash join (Tom)
Fix for DROP TABLE/INDEX in rolled-back transaction (Hiroshi)
Fix psql crash from \l+ if MULTIBYTE enabled (Peter E)
Fix truncation of rule names during CREATE VIEW (Ross Reedstrom)
Fix PL/perl (Alex Kapranoff)
Disallow LOCK on views (Mark Hollomon)
Disallow INSERT/UPDATE/DELETE on views (Mark Hollomon)
Disallow DROP RULE, CREATE INDEX, TRUNCATE on views (Mark Hollomon)
Allow PL/pgSQL accept non-ASCII identifiers (Tatsuo)
Allow views to proper handle GROUP BY, aggregates, DISTINCT (Tom)
Fix rare failure with TRUNCATE command (Tom)
Allow UNION/INTERSECT/EXCEPT to be used with ALL, subqueries, views,
```

```
    DISTINCT, ORDER BY, SELECT...INTO (Tom)
Fix parser failures during aborted transactions (Tom)
Allow temporary relations to properly clean up indexes (Bruce)
Fix VACUUM problem with moving rows in same page (Tom)
Modify pg_dump to better handle user-defined items in template1 (Philip)
Allow LIMIT in VIEW (Tom)
Require cursor FETCH to honor LIMIT (Tom)
Allow PRIMARY/FOREIGN Key definitions on inherited columns (Stephan)
Allow ORDER BY, LIMIT in sub-selects (Tom)
Allow UNION in CREATE RULE (Tom)
Make ALTER/DROP TABLE rollback-able (Vadim, Tom)
Store initdb collation in pg_control so collation cannot be changed (Tom)
Fix INSERT...SELECT with rules (Tom)
Fix FOR UPDATE inside views and subselects (Tom)
Fix OVERLAPS operators conform to SQL92 spec regarding NULLs (Tom)
Fix lpad() and rpad() to handle length less than input string (Tom)
Fix use of NOTIFY in some rules (Tom)
Overhaul btree code (Tom)
Fix NOT NULL use in Pl/pgSQL variables (Tom)
Overhaul GIST code (Oleg)
Fix CLUSTER to preserve constraints and column default (Tom)
Improved deadlock detection handling (Tom)
Allow multiple SERIAL columns in a table (Tom)
Prevent occasional index corruption (Vadim)

Enhancements
------------
Add OUTER JOINs (Tom)
Function manager overhaul (Tom)
Allow ALTER TABLE RENAME on indexes (Tom)
Improve CLUSTER (Tom)
Improve ps status display for more platforms (Peter E, Marc)
Improve CREATE FUNCTION failure message (Ross)
JDBC improvements (Peter, Travis Bauer, Christopher Cain, William Webber,
    Gunnar)
Grand Unified Configuration scheme/GUC.  Many options can now be set in
    data/postgresql.conf, postmaster/postgres flags, or SET commands (Peter E)
Improved handling of file descriptor cache (Tom)
New warning code about auto-created table alias entries (Bruce)
Overhaul initdb process (Tom, Peter E)
Overhaul of inherited tables; inherited tables now accessed by default;
   new ONLY keyword prevents it (Chris Bitmead, Tom)
ODBC cleanups/improvements (Nick Gorham, Stephan Szabo, Zoltan Kovacs,
    Michael Fork)
Allow renaming of temp tables (Tom)
Overhaul memory manager contexts (Tom)
pg_dumpall uses CREATE USER or CREATE GROUP rather using COPY (Peter E)
Overhaul pg_dump (Philip Warner)
Allow pg_hba.conf secondary password file to specify only username (Peter E)
Allow TEMPORARY or TEMP keyword when creating temporary tables (Bruce)
New memory leak checker (Karel)
New SET SESSION CHARACTERISTICS (Thomas)
Allow nested block comments (Thomas)
Add WITHOUT TIME ZONE type qualifier (Thomas)
New ALTER TABLE ADD CONSTRAINT (Stephan)
Use NUMERIC accumulators for INTEGER aggregates (Tom)
Overhaul aggregate code (Tom)
```

```
New VARIANCE and STDDEV() aggregates
Improve dependency ordering of pg_dump (Philip)
New pg_restore command (Philip)
New pg_dump tar output option (Philip)
New pg_dump of large objects  (Philip)
New ESCAPE option to LIKE (Thomas)
New case-insensitive LIKE - ILIKE (Thomas)
Allow functional indexes to use binary-compatible type (Tom)
Allow SQL functions to be used in more contexts (Tom)
New pg_config utility (Peter E)
New PL/pgSQL EXECUTE command which allows dynamic SQL and utility statements
    (Jan)
New PL/pgSQL GET DIAGNOSTICS statement for SPI value access (Jan)
New quote_identifiers() and quote_literal() functions (Jan)
New ALTER TABLE table OWNER TO user command (Mark Hollomon)
Allow subselects in FROM, i.e. FROM (SELECT ...) [AS] alias (Tom)
Update PyGreSQL to version 3.1 (D'Arcy)
Store tables as files named by OID (Vadim)
New SQL function setval(seq,val,bool) for use in pg_dump (Philip)
Require DROP VIEW to remove views, no DROP TABLE (Mark)
Allow DROP VIEW view1, view2 (Mark)
Allow multiple objects in DROP INDEX, DROP RULE, and DROP TYPE (Tom)
Allow automatic conversion to/from Unicode (Tatsuo, Eiji)
New /contrib/pgcrypto hashing functions (Marko Kreen)
New pg_dumpall --globals-only option (Peter E)
New CHECKPOINT command for WAL which creates new WAL log file (Vadim)
New AT TIME ZONE syntax (Thomas)
Allow location of Unix domain socket to be configurable (David J. MacKenzie)
Allow postmaster to listen on a specific IP address (David J. MacKenzie)
Allow socket path name to be specified in hostname by using leading slash
    (David J. MacKenzie)
Allow CREATE DATABASE to specify template database (Tom)
New utility to convert MySQL schema dumps to SQL92 and PostgreSQL (Thomas)
New /contrib/rserv replication toolkit (Vadim)
New file format for COPY BINARY (Tom)
New /contrib/oid2name to map numeric files to table names (B Palmer)
New "idle in transaction" ps status message (Marc)
Update to pgaccess 0.98.7 (Constantin Teodorescu)
pg_ctl now defaults to -w (wait) on shutdown, new -l (log) option
Add rudimentary dependency checking to pg_dump (Philip)

Types
-----
Fix INET/CIDR type ordering and add new functions (Tom)
Make OID behave as an unsigned type (Tom)
Allow BIGINT as synonym for INT8 (Peter E)
New int2 and int8 comparison operators (Tom)
New BIT and BIT VARYING types (Adriaan Joubert, Tom, Peter E)
CHAR() no longer faster than VARCHAR() because of TOAST (Tom)
New GIST seg/cube examples (Gene Selkov)
Improved round(numeric) handling (Tom)
Fix CIDR output formatting (Tom)
New CIDR abbrev() function (Tom)

Performance
-----------
Write-Ahead Log (WAL) to provide crash recovery with less performance
```

```
    overhead (Vadim)
ANALYZE stage of VACUUM no longer exclusively locks table (Bruce)
Reduced file seeks (Denis Perchine)
Improve BTREE code for duplicate keys (Tom)
Store all large objects in a single table (Denis Perchine, Tom)
Improve memory allocation performance (Karel, Tom)

Source Code
-----------
New function manager call conventions (Tom)
SGI portability fixes (David Kaelbling)
New configure --enable-syslog option (Peter E)
New BSDI README (Bruce)
configure script moved to top level, not /src (Peter E)
Makefile/configuration/compilation overhaul (Peter E)
New configure --with-python option (Peter E)
Solaris cleanups (Peter E)
Overhaul /contrib Makefiles (Karel)
New OpenSSL configuration option (Magnus, Peter E)
AIX fixes (Andreas)
QNX fixes (Maurizio)
New heap_open(), heap_openr() API (Tom)
Remove colon and semi-colon operators (Thomas)
New pg_class.relkind value for views (Mark Hollomon)
Rename ichar() to chr() (Karel)
New documentation for btrim(), ascii(), chr(), repeat() (Karel)
Fixes for NT/Cygwin (Pete Forman)
AIX port fixes (Andreas)
New BeOS port (David Reid, Cyril Velter)
Add proofreader's changes to docs (Addison-Wesley, Bruce)
New Alpha spinlock code (Adriaan Joubert, Compaq)
UnixWare port overhaul (Peter E)
New Darwin/MacOS X port (Peter Bierman, Bruce Hartzler)
New FreeBSD Alpha port (Alfred)
Overhaul shared memory segments (Tom)
Add IBM S/390 support (Neale Ferguson)
Moved macmanuf to /contrib (Larry Rosenman)
Syslog improvements (Larry Rosenman)
New template0 database that contains no user additions (Tom)
New /contrib/cube and /contrib/seg GIST sample code (Gene Selkov)
Allow NetBSD's libedit instead of readline (Peter)
Improved assembly language source code format (Bruce)
New contrib/pg_logger
New --template option to createdb
New contrib/pg_control utility (Oliver)
New FreeBSD tools ipc_check, start-scripts/freebsd
```

A.6. Release 7.0.3

Release date: 2000-11-11

This has a variety of fixes from 7.0.2.

A.6.1. Migration to version 7.0.3

A dump/restore is *not* required for those running 7.0.*.

A.6.2. Changes

```
Jdbc fixes (Peter)
Large object fix (Tom)
Fix lean in COPY WITH OIDS leak (Tom)
Fix backwards-index-scan (Tom)
Fix SELECT ... FOR UPDATE so it checks for duplicate keys (Hiroshi)
Add --enable-syslog to configure (Marc)
Fix abort transaction at backend exit in rare cases (Tom)
Fix for psql \l+ when multi-byte enabled (Tatsuo)
Allow PL/pgSQL to accept non ascii identifiers (Tatsuo)
Make vacuum always flush buffers (Tom)
Fix to allow cancel while waiting for a lock (Hiroshi)
Fix for memory aloocation problem in user authentication code (Tom)
Remove bogus use of int4out() (Tom)
Fixes for multiple subqueries in COALESCE or BETWEEN (Tom)
Fix for failure of triggers on heap open in certain cases (Jeroen van
    Vianen)
Fix for erroneous selectivity of not-equals (Tom)
Fix for erroneous use of strcmp() (Tom)
Fix for bug where storage manager accesses items beyond end of file
    (Tom)
Fix to include kernel errno message in all smgr elog messages (Tom)
Fix for '.' not in PATH at build time (SL Baur)
Fix for out-of-file-descriptors error (Tom)
Fix to make pg_dump dump 'iscachable' flag for functions (Tom)
Fix for subselect in targetlist of Append node (Tom)
Fix for mergejoin plans (Tom)
Fix TRUNCATE failure on relations with indexes (Tom)
Avoid database-wide restart on write error (Hiroshi)
Fix nodeMaterial to honor chgParam by recomputing its output (Tom)
Fix VACUUM problem with moving chain of update tuples when source and
    destination of a tuple lie on the same page (Tom)
Fix user.c CommandCounterIncrement (Tom)
Fix for AM/PM boundary problem in to_char() (Karel Zak)
Fix TIME aggregate handling (Tom)
Fix to_char() to avoid coredump on NULL input (Tom)
Buffer fix (Tom)
Fix for inserting/copying longer multibyte strings into char() data
    types (Tatsuo)
Fix for crash of backend, on abort (Tom)
```

A.7. Release 7.0.2

Release date: 2000-06-05

This is a repackaging of 7.0.1 with added documentation.

A.7.1. Migration to version 7.0.2

A dump/restore is *not* required for those running 7.*.

A.7.2. Changes

```
Added documentation to tarball.
```

A.8. Release 7.0.1

Release date: 2000-06-01

This is a cleanup release for 7.0.

A.8.1. Migration to version 7.0.1

A dump/restore is *not* required for those running 7.0.

A.8.2. Changes

```
Fix many CLUSTER failures (Tom)
Allow ALTER TABLE RENAME works on indexes (Tom)
Fix plpgsql to handle datetime->timestamp and timespan->interval (Bruce)
New configure --with-setproctitle switch to use setproctitle() (Marc, Bruce)
Fix the off by one errors in ResultSet from 6.5.3, and more.
jdbc ResultSet fixes (Joseph Shraibman)
optimizer tunings (Tom)
Fix create user for pgaccess
Fix for UNLISTEN failure
IRIX fixes (David Kaelbling)
QNX fixes (Andreas Kardos)
Reduce COPY IN lock level (Tom)
```

```
Change libpqeasy to use PQconnectdb() style parameters (Bruce)
Fix pg_dump to handle OID indexes (Tom)
Fix small memory leak (Tom)
Solaris fix for createdb/dropdb (Tatsuo)
Fix for non-blocking connections (Alfred Perlstein)
Fix improper recovery after RENAME TABLE failures (Tom)
Copy pg_ident.conf.sample into /lib directory in install (Bruce)
Add SJIS UDC (NEC selection IBM kanji) support (Eiji Tokuya)
Fix too long syslog message (Tatsuo)
Fix problem with quoted indexes that are too long (Tom)
JDBC ResultSet.getTimestamp() fix (Gregory Krasnow & Floyd Marinescu)
ecpg changes (Michael)
```

A.9. Release 7.0

Release date: 2000-05-08

This release contains improvements in many areas, demonstrating the continued growth of PostgreSQL. There are more improvements and fixes in 7.0 than in any previous release. The developers have confidence that this is the best release yet; we do our best to put out only solid releases, and this one is no exception.

Major changes in this release:

Foreign Keys

Foreign keys are now implemented, with the exception of PARTIAL MATCH foreign keys. Many users have been asking for this feature, and we are pleased to offer it.

Optimizer Overhaul

Continuing on work started a year ago, the optimizer has been improved, allowing better query plan selection and faster performance with less memory usage.

Updated psql

psql, our interactive terminal monitor, has been updated with a variety of new features. See the psql manual page for details.

Join Syntax

SQL92 join syntax is now supported, though only as INNER JOINs for this release. JOIN, NATURAL JOIN, JOIN/USING, JOIN/ON are available, as are column correlation names.

A.9.1. Migration to version 7.0

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release of PostgreSQL. For those upgrading from 6.5.*, you may instead use pg_upgrade to upgrade to this release; however, a full dump/reload installation is always the most robust method for upgrades.

Interface and compatibility issues to consider for the new release include:

- The date/time types `datetime` and `timespan` have been superseded by the SQL92-defined types `timestamp` and `interval`. Although there has been some effort to ease the transition by allowing PostgreSQL to recognize the deprecated type names and translate them to the new type names, this mechanism may not be completely transparent to your existing application.
- The optimizer has been substantially improved in the area of query cost estimation. In some cases, this will result in decreased query times as the optimizer makes a better choice for the preferred plan. However, in a small number of cases, usually involving pathological distributions of data, your query times may go up. If you are dealing with large amounts of data, you may want to check your queries to verify performance.
- The JDBC and ODBC interfaces have been upgraded and extended.
- The string function `CHAR_LENGTH` is now a native function. Previous versions translated this into a call to `LENGTH`, which could result in ambiguity with other types implementing `LENGTH` such as the geometric types.

A.9.2. Changes

```
Bug Fixes
---------
Prevent function calls exceeding maximum number of arguments (Tom)
Improve CASE construct (Tom)
Fix SELECT coalesce(f1,0) FROM int4_tbl GROUP BY f1 (Tom)
Fix SELECT sentence.words[0] FROM sentence GROUP BY sentence.words[0] (Tom)
Fix GROUP BY scan bug (Tom)
Improvements in SQL grammar processing (Tom)
Fix for views involved in INSERT ... SELECT ... (Tom)
Fix for SELECT a/2, a/2 FROM test_missing_target GROUP BY a/2 (Tom)
Fix for subselects in INSERT ... SELECT (Tom)
Prevent INSERT ... SELECT ... ORDER BY (Tom)
Fixes for relations greater than 2GB, including vacuum
Improve propagating system table changes to other backends (Tom)
Improve propagating user table changes to other backends (Tom)
Fix handling of temp tables in complex situations (Bruce, Tom)
Allow table locking at table open, improving concurrent reliability (Tom)
Properly quote sequence names in pg_dump (Ross J. Reedstrom)
Prevent DROP DATABASE while others accessing
Prevent any rows from being returned by GROUP BY if no rows processed (Tom)
Fix SELECT COUNT(1) FROM table WHERE ...' if no rows matching WHERE (Tom)
Fix pg_upgrade so it works for MVCC (Tom)
Fix for SELECT ... WHERE x IN (SELECT ... HAVING SUM(x) > 1) (Tom)
Fix for "f1 datetime DEFAULT 'now'"  (Tom)
Fix problems with CURRENT_DATE used in DEFAULT (Tom)
Allow comment-only lines, and ;;; lines too. (Tom)
Improve recovery after failed disk writes, disk full (Hiroshi)
Fix cases where table is mentioned in FROM but not joined (Tom)
Allow HAVING clause without aggregate functions (Tom)
Fix for "--" comment and no trailing newline, as seen in perl interface
Improve pg_dump failure error reports (Bruce)
Allow sorts and hashes to exceed 2GB file sizes (Tom)
Fix for pg_dump dumping of inherited rules (Tom)
Fix for NULL handling comparisons (Tom)
Fix inconsistent state caused by failed CREATE/DROP commands (Hiroshi)
```

```
Fix for dbname with dash
Prevent DROP INDEX from interfering with other backends (Tom)
Fix file descriptor leak in verify_password()
Fix for "Unable to identify an operator =$" problem
Fix ODBC so no segfault if CommLog and Debug enabled (Dirk Niggemann)
Fix for recursive exit call (Massimo)
Fix for extra-long timezones (Jeroen van Vianen)
Make pg_dump preserve primary key information (Peter E)
Prevent databases with single quotes (Peter E)
Prevent DROP DATABASE inside  transaction (Peter E)
ecpg memory leak fixes (Stephen Birch)
Fix for SELECT null::text, SELECT int4fac(null) and SELECT 2 + (null) (Tom)
Y2K timestamp fix (Massimo)
Fix for VACUUM 'HEAP_MOVED_IN was not expected' errors (Tom)
Fix for views with tables/columns containing spaces  (Tom)
Prevent permissions on indexes (Peter E)
Fix for spinlock stuck problem when error is generated (Hiroshi)
Fix ipcclean on Linux
Fix handling of NULL constraint conditions (Tom)
Fix memory leak in odbc driver (Nick Gorham)
Fix for permission check on UNION tables (Tom)
Fix to allow SELECT 'a' LIKE 'a' (Tom)
Fix for SELECT 1 + NULL (Tom)
Fixes to CHAR
Fix log() on numeric type (Tom)
Deprecate ':' and ';' operators
Allow vacuum of temporary tables
Disallow inherited columns with the same name as new columns
Recover or force failure when disk space is exhausted (Hiroshi)
Fix INSERT INTO ... SELECT with AS columns matching result columns
Fix INSERT ... SELECT ... GROUP BY groups by target columns not source columns (Tom)
Fix CREATE TABLE test (a char(5) DEFAULT text '', b int4) with INSERT (Tom)
Fix UNION with LIMIT
Fix CREATE TABLE x AS SELECT 1 UNION SELECT 2
Fix CREATE TABLE test(col char(2) DEFAULT user)
Fix mismatched types in CREATE TABLE ... DEFAULT
Fix SELECT * FROM pg_class where oid in (0,-1)
Fix SELECT COUNT('asdf') FROM pg_class WHERE oid=12
Prevent user who can create databases can modifying pg_database table (Peter E)
Fix btree to give a useful elog when key > 1/2 (page - overhead) (Tom)
Fix INSERT of 0.0 into DECIMAL(4,4) field (Tom)

Enhancements
------------
New CLI interface include file sqlcli.h, based on SQL3/SQL98
Remove all limits on query length, row length limit still exists (Tom)
Update jdbc protocol to 2.0 (Jens Glaser <jens@jens.de>)
Add TRUNCATE command to quickly truncate relation (Mike Mascari)
Fix to give super user and createdb user proper update catalog rights (Peter E)
Allow ecpg bool variables to have NULL values (Christof)
Issue ecpg error if NULL value for variable with no NULL indicator (Christof)
Allow ^C to cancel COPY command (Massimo)
Add SET FSYNC and SHOW PG_OPTIONS commands(Massimo)
Function name overloading for dynamically-loaded C functions (Frankpitt)
Add CmdTuples() to libpq++(Vince)
New CREATE CONSTRAINT TRIGGER and SET CONSTRAINTS commands(Jan)
Allow CREATE FUNCTION/WITH clause to be used for all language types
```

configure --enable-debug adds -g (Peter E)
configure --disable-debug removes -g (Peter E)
Allow more complex default expressions (Tom)
First real FOREIGN KEY constraint trigger functionality (Jan)
Add FOREIGN KEY ... MATCH FULL ... ON DELETE CASCADE (Jan)
Add FOREIGN KEY ... MATCH <unspecified> referential actions (Don Baccus)
Allow WHERE restriction on ctid (physical heap location) (Hiroshi)
Move pginterface from contrib to interface directory, rename to pgeasy (Bruce)
Change pgeasy connectdb() parameter ordering (Bruce)
Require SELECT DISTINCT target list to have all ORDER BY columns (Tom)
Add Oracle's COMMENT ON command (Mike Mascari <mascarim@yahoo.com>)
libpq's PQsetNoticeProcessor function now returns previous hook(Peter E)
Prevent PQsetNoticeProcessor from being set to NULL (Peter E)
Make USING in COPY optional (Bruce)
Allow subselects in the target list (Tom)
Allow subselects on the left side of comparison operators (Tom)
New parallel regression test (Jan)
Change backend-side COPY to write files with permissions 644 not 666 (Tom)
Force permissions on PGDATA directory to be secure, even if it exists (Tom)
Added psql LASTOID variable to return last inserted oid (Peter E)
Allow concurrent vacuum and remove pg_vlock vacuum lock file (Tom)
Add permissions check for vacuum (Peter E)
New libpq functions to allow asynchronous connections: PQconnectStart(),
   PQconnectPoll(), PQresetStart(), PQresetPoll(), PQsetenvStart(),
   PQsetenvPoll(), PQsetenvAbort (Ewan Mellor)
New libpq PQsetenv() function (Ewan Mellor)
create/alter user extension (Peter E)
New postmaster.pid and postmaster.opts under $PGDATA (Tatsuo)
New scripts for create/drop user/db (Peter E)
Major psql overhaul (Peter E)
Add const to libpq interface (Peter E)
New libpq function PQoidValue (Peter E)
Show specific non-aggregate causing problem with GROUP BY (Tom)
Make changes to pg_shadow recreate pg_pwd file (Peter E)
Add aggregate(DISTINCT ...) (Tom)
Allow flag to control COPY input/output of NULLs (Peter E)
Make postgres user have a password by default (Peter E)
Add CREATE/ALTER/DROP GROUP (Peter E)
All administration scripts now support --long options (Peter E, Karel)
Vacuumdb script now supports --all option (Peter E)
ecpg new portable FETCH syntax
Add ecpg EXEC SQL IFDEF, EXEC SQL IFNDEF, EXEC SQL ELSE, EXEC SQL ELIF
 and EXEC SQL ENDIF directives
Add pg_ctl script to control backend start-up (Tatsuo)
Add postmaster.opts.default file to store start-up flags (Tatsuo)
Allow --with-mb=SQL_ASCII
Increase maximum number of index keys to 16 (Bruce)
Increase maximum number of function arguments to 16 (Bruce)
Allow configuration of maximum number of index keys and arguments (Bruce)
Allow unprivileged users to change their passwords (Peter E)
Password authentication enabled; required for new users (Peter E)
Disallow dropping a user who owns a database (Peter E)
Change initdb option --with-mb to --enable-multibyte
Add option for initdb to prompts for superuser password (Peter E)
Allow complex type casts like col::numeric(9,2) and col::int2::float8 (Tom)
Updated user interfaces on initdb, initlocation, pg_dump, ipcclean (Peter E)
New pg_char_to_encoding() and pg_encoding_to_char() functions (Tatsuo)

```
Libpq non-blocking mode (Alfred Perlstein)
Improve conversion of types in casts that don't specify a length
New plperl internal programming language (Mark Hollomon)
Allow COPY IN to read file that do not end with a newline (Tom)
Indicate when long identifiers are truncated (Tom)
Allow aggregates to use type equivalency (Peter E)
Add Oracle's to_char(), to_date(), to_datetime(), to_timestamp(), to_number()
 conversion functions (Karel Zak <zakkr@zf.jcu.cz>)
Add SELECT DISTINCT ON (expr [, expr ...]) targetlist ... (Tom)
Check to be sure ORDER BY is compatible with the DISTINCT operation (Tom)
Add NUMERIC and int8 types to ODBC
Improve EXPLAIN results for Append, Group, Agg, Unique (Tom)
Add ALTER TABLE ... ADD FOREIGN KEY (Stephan Szabo)
Allow SELECT .. FOR UPDATE in PL/pgSQL (Hiroshi)
Enable backward sequential scan even after reaching EOF (Hiroshi)
Add btree indexing of boolean values, >= and <= (Don Baccus)
Print current line number when COPY FROM fails (Massimo)
Recognize POSIX time zone e.g. "PST+8" and "GMT-8" (Thomas)
Add DEC as synonym for DECIMAL (Thomas)
Add SESSION_USER as SQL92 keyword, same as CURRENT_USER (Thomas)
Implement SQL92 column aliases (aka correlation names) (Thomas)
Implement SQL92 join syntax (Thomas)
Make INTERVAL reserved word allowed as a column identifier (Thomas)
Implement REINDEX command (Hiroshi)
Accept ALL in aggregate function SUM(ALL col) (Tom)
Prevent GROUP BY from using column aliases (Tom)
New psql \encoding option (Tatsuo)
Allow PQrequestCancel() to terminate when in waiting-for-lock state (Hiroshi)
Allow negation of a negative number in all cases
Add ecpg descriptors (Christof, Michael)
Allow CREATE VIEW v AS SELECT f1::char(8) FROM tbl
Allow casts with length, like foo::char(8)
New libpq functions PQsetClientEncoding(), PQclientEncoding() (Tatsuo)
Add support for SJIS user defined characters (Tatsuo)
Larger views/rules supported
Make libpq's PQconndefaults() thread-safe (Tom)
Disable // as comment to be ANSI conforming, should use -- (Tom)
Allow column aliases on views CREATE VIEW name (collist)
Fixes for views with subqueries (Tom)
Allow UPDATE table SET fld = (SELECT ...) (Tom)
SET command options no longer require quotes
Update pgaccess to 0.98.6
New SET SEED command
New pg_options.sample file
New SET FSYNC command (Massimo)
Allow pg_descriptions when creating tables
Allow pg_descriptions when creating types, columns, and functions
Allow psql \copy to allow delimiters (Peter E)
Allow psql to print nulls as distinct from "" [null] (Peter E)

Types
-----
Many array fixes (Tom)
Allow bare column names to be subscripted as arrays (Tom)
Improve type casting of int and float constants (Tom)
Cleanups for int8 inputs, range checking, and type conversion (Tom)
Fix for SELECT timespan('21:11:26'::time) (Tom)
```

```
netmask('x.x.x.x/0') is 255.255.255.255 instead of 0.0.0.0 (Oleg Sharoiko)
Add btree index on NUMERIC (Jan)
Perl fix for large objects containing NUL characters (Douglas Thomson)
ODBC fix for for large objects (free)
Fix indexing of cidr data type
Fix for Ethernet MAC addresses (macaddr type) comparisons
Fix for date/time types when overflows happened in computations (Tom)
Allow array on int8 (Peter E)
Fix for rounding/overflow of NUMERIC type, like NUMERIC(4,4) (Tom)
Allow NUMERIC arrays
Fix bugs in NUMERIC ceil() and floor() functions (Tom)
Make char_length()/octet_length including trailing blanks (Tom)
Made abstime/reltime use int4 instead of time_t (Peter E)
New lztext data type for compressed text fields
Revise code to handle coercion of int and float constants (Tom)
Start at new code to implement a BIT and BIT VARYING type (Adriaan Joubert)
NUMERIC now accepts scientific notation (Tom)
NUMERIC to int4 rounds (Tom)
Convert float4/8 to NUMERIC properly (Tom)
Allow type conversion with NUMERIC (Thomas)
Make ISO date style (2000-02-16 09:33) the default (Thomas)
Add NATIONAL CHAR [ VARYING ] (Thomas)
Allow NUMERIC round and trunc to accept negative scales (Tom)
New TIME WITH TIME ZONE type (Thomas)
Add MAX()/MIN() on time type (Thomas)
Add abs(), mod(), fac() for int8 (Thomas)
Rename functions to round(), sqrt(), cbrt(), pow() for float8 (Thomas)
Add transcendental math functions (e.g. sin(), acos()) for float8 (Thomas)
Add exp() and ln() for NUMERIC type
Rename NUMERIC power() to pow() (Thomas)
Improved TRANSLATE() function (Edwin Ramirez, Tom)
Allow X=-Y operators  (Tom)
Allow SELECT float8(COUNT(*))/(SELECT COUNT(*) FROM t) FROM t GROUP BY f1; (Tom)
Allow LOCALE to use indexes in regular expression searches (Tom)
Allow creation of functional indexes to use default types

Performance
-----------
Prevent exponential space consumption with many AND's and OR's (Tom)
Collect attribute selectivity values for system columns (Tom)
Reduce memory usage of aggregates (Tom)
Fix for LIKE optimization to use indexes with multibyte encodings (Tom)
Fix r-tree index optimizer selectivity (Thomas)
Improve optimizer selectivity computations and functions (Tom)
Optimize btree searching for cases where many equal keys exist (Tom)
Enable fast LIKE index processing only if index present (Tom)
Re-use free space on index pages with duplicates (Tom)
Improve hash join processing (Tom)
Prevent descending sort if result is already sorted(Hiroshi)
Allow commuting of index scan query qualifications (Tom)
Prefer index scans in cases where ORDER BY/GROUP BY is required (Tom)
Allocate large memory requests in fix-sized chunks for performance (Tom)
Fix vacuum's performance by reducing memory allocation requests (Tom)
Implement constant-expression simplification (Bernard Frankpitt, Tom)
Use secondary columns to be used to determine start of index scan (Hiroshi)
Prevent quadruple use of disk space when doing internal sorting (Tom)
Faster sorting by calling fewer functions (Tom)
```

Create system indexes to match all system caches (Bruce, Hiroshi)
Make system caches use system indexes (Bruce)
Make all system indexes unique (Bruce)
Improve pg_statistics management for VACUUM speed improvement (Tom)
Flush backend cache less frequently (Tom, Hiroshi)
COPY now reuses previous memory allocation, improving performance (Tom)
Improve optimization cost estimation (Tom)
Improve optimizer estimate of range queries x > lowbound AND x < highbound (Tom)
Use DNF instead of CNF where appropriate (Tom, Taral)
Further cleanup for OR-of-AND WHERE-clauses (Tom)
Make use of index in OR clauses (x = 1 AND y = 2) OR (x = 2 AND y = 4) (Tom)
Smarter optimizer computations for random index page access (Tom)
New SET variable to control optimizer costs (Tom)
Optimizer queries based on LIMIT, OFFSET, and EXISTS qualifications (Tom)
Reduce optimizer internal housekeeping of join paths for speedup (Tom)
Major subquery speedup (Tom)
Fewer fsync writes when fsync is not disabled (Tom)
Improved LIKE optimizer estimates (Tom)
Prevent fsync in SELECT-only queries (Vadim)
Make index creation use psort code, because it is now faster (Tom)
Allow creation of sort temp tables > 1 Gig

Source Tree Changes

Fix for linux PPC compile
New generic expression-tree-walker subroutine (Tom)
Change form() to varargform() to prevent portability problems
Improved range checking for large integers on Alphas
Clean up #include in /include directory (Bruce)
Add scripts for checking includes (Bruce)
Remove un-needed #include's from *.c files (Bruce)
Change #include's to use <> and "" as appropriate (Bruce)
Enable WIN32 compilation of libpq
Alpha spinlock fix from Uncle George <gatgul@voicenet.com>
Overhaul of optimizer data structures (Tom)
Fix to cygipc library (Yutaka Tanida)
Allow pgsql to work on newer Cygwin snapshots (Dan)
New catalog version number (Tom)
Add Linux ARM
Rename heap_replace to heap_update
Update for QNX (Dr. Andreas Kardos)
New platform-specific regression handling (Tom)
Rename oid8 -> oidvector and int28 -> int2vector (Bruce)
Included all yacc and lex files into the distribution (Peter E.)
Remove lextest, no longer needed (Peter E)
Fix for libpq and psql on Win32 (Magnus)
Internally change datetime and timespan into timestamp and interval (Thomas)
Fix for plpgsql on BSD/OS
Add SQL_ASCII test case to the regression test (Tatsuo)
configure --with-mb now deprecated (Tatsuo)
NT fixes
NetBSD fixes (Johnny C. Lam <lamj@stat.cmu.edu>)
Fixes for Alpha compiles
New multibyte encodings

A.10. Release 6.5.3

Release date: 1999-10-13

This is basically a cleanup release for 6.5.2. We have added a new PgAccess that was missing in 6.5.2, and installed an NT-specific fix.

A.10.1. Migration to version 6.5.3

A dump/restore is *not* required for those running 6.5.*.

A.10.2. Changes

```
Updated version of pgaccess 0.98
NT-specific patch
Fix dumping rules on inherited tables
```

A.11. Release 6.5.2

Release date: 1999-09-15

This is basically a cleanup release for 6.5.1. We have fixed a variety of problems reported by 6.5.1 users.

A.11.1. Migration to version 6.5.2

A dump/restore is *not* required for those running 6.5.*.

A.11.2. Changes

```
subselect+CASE fixes(Tom)
Add SHLIB_LINK setting for solaris_i386 and solaris_sparc ports(Daren Sefcik)
Fixes for CASE in WHERE join clauses(Tom)
Fix BTScan abort(Tom)
Repair the check for redundant UNIQUE and PRIMARY KEY indexes(Thomas)
Improve it so that it checks for multi-column constraints(Thomas)
Fix for Win32 making problem with MB enabled(Hiroki Kataoka)
Allow BSD yacc and bison to compile pl code(Bruce)
Fix SET NAMES working
int8 fixes(Thomas)
Fix vacuum's memory consumption(Hiroshi,Tatsuo)
Reduce the total memory consumption of vacuum(Tom)
Fix for timestamp(datetime)
```

```
Rule deparsing bugfixes(Tom)
Fix quoting problems in mkMakefile.tcldefs.sh.in and mkMakefile.tkdefs.sh.in(Tom)
This is to re-use space on index pages freed by vacuum(Vadim)
document -x for pg_dump(Bruce)
Fix for unary operators in rule deparser(Tom)
Comment out FileUnlink of excess segments during mdtruncate()(Tom)
IRIX linking fix from Yu Cao >yucao@falcon.kla-tencor.com<
Repair logic error in LIKE: should not return LIKE_ABORT
   when reach end of pattern before end of text(Tom)
Repair incorrect cleanup of heap memory allocation during transaction abort(Tom)
Updated version of pgaccess 0.98
```

A.12. Release 6.5.1

Release date: 1999-07-15

This is basically a cleanup release for 6.5. We have fixed a variety of problems reported by 6.5 users.

A.12.1. Migration to version 6.5.1

A dump/restore is *not* required for those running 6.5.

A.12.2. Changes

```
Add NT README file
Portability fixes for linux_ppc, IRIX, linux_alpha, OpenBSD, alpha
Remove QUERY_LIMIT, use SELECT...LIMIT
Fix for EXPLAIN on inheritance(Tom)
Patch to allow vacuum on multi-segment tables(Hiroshi)
R-Tree optimizer selectivity fix(Tom)
ACL file descriptor leak fix(Atsushi Ogawa)
New expresssion subtree code(Tom)
Avoid disk writes for read-only transactions(Vadim)
Fix for removal of temp tables if last transaction was aborted(Bruce)
Fix to prevent too large tuple from being created(Bruce)
plpgsql fixes
Allow port numbers 32k - 64k(Bruce)
Add ^ precidence(Bruce)
Rename sort files called pg_temp to pg_sorttemp(Bruce)
Fix for microseconds in time values(Tom)
Tutorial source cleanup
New linux_m68k port
Fix for sorting of NULL's in some cases(Tom)
Shared library dependencies fixed (Tom)
Fixed glitches affecting GROUP BY in subselects(Tom)
Fix some compiler warnings (Tomoaki Nishiyama)
Add Win1250 (Czech) support (Pavel Behal)
```

A.13. Release 6.5

Release date: 1999-06-09

This release marks a major step in the development team's mastery of the source code we inherited from Berkeley. You will see we are now easily adding major features, thanks to the increasing size and experience of our world-wide development team.

Here is a brief summary of the more notable changes:

Multi-version concurrency control(MVCC)

This removes our old table-level locking, and replaces it with a locking system that is superior to most commercial database systems. In a traditional system, each row that is modified is locked until committed, preventing reads by other users. MVCC uses the natural multi-version nature of PostgreSQL to allow readers to continue reading consistent data during writer activity. Writers continue to use the compact pg_log transaction system. This is all performed without having to allocate a lock for every row like traditional database systems. So, basically, we no longer are restricted by simple table-level locking; we have something better than row-level locking.

Hot backups from pg_dump

pg_dump takes advantage of the new MVCC features to give a consistent database dump/backup while the database stays online and available for queries.

Numeric data type

We now have a true numeric data type, with user-specified precision.

Temporary tables

Temporary tables are guaranteed to have unique names within a database session, and are destroyed on session exit.

New SQL features

We now have CASE, INTERSECT, and EXCEPT statement support. We have new LIMIT/OFFSET, SET TRANSACTION ISOLATION LEVEL, SELECT ... FOR UPDATE, and an improved LOCK TABLE command.

Speedups

We continue to speed up PostgreSQL, thanks to the variety of talents within our team. We have sped up memory allocation, optimization, table joins, and row transfer routines.

Ports

We continue to expand our port list, this time including Windows NT/ix86 and NetBSD/arm32.

Interfaces

Most interfaces have new versions, and existing functionality has been improved.

Documentation

New and updated material is present throughout the documentation. New FAQs have been contributed for SGI and AIX platforms. The *Tutorial* has introductory information on SQL from Stefan Simkovics. For the *User's Guide*, there are reference pages covering the postmaster and more utility programs, and a new appendix contains details on date/time behavior. The *Administrator's Guide* has a new chapter on troubleshooting from Tom Lane. And the *Programmer's Guide* has a description of query processing, also from Stefan, and details on obtaining the PostgreSQL source tree via anonymous CVS and CVSup.

A.13.1. Migration to version 6.5

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release of PostgreSQL. pg_upgrade can *not* be used to upgrade to this release because the on-disk structure of the tables has changed compared to previous releases.

The new Multi-Version Concurrency Control (MVCC) features can give somewhat different behaviors in multi-user environments. *Read and understand the following section to ensure that your existing applications will give you the behavior you need.*

A.13.1.1. Multi-Version Concurrency Control

Because readers in 6.5 don't lock data, regardless of transaction isolation level, data read by one transaction can be overwritten by another. In other words, if a row is returned by **SELECT** it doesn't mean that this row really exists at the time it is returned (i.e. sometime after the statement or transaction began) nor that the row is protected from being deleted or updated by concurrent transactions before the current transaction does a commit or rollback.

To ensure the actual existence of a row and protect it against concurrent updates one must use **SELECT FOR UPDATE** or an appropriate **LOCK TABLE** statement. This should be taken into account when porting applications from previous releases of PostgreSQL and other environments.

Keep the above in mind if you are using `contrib/refint.*` triggers for referential integrity. Additional techniques are required now. One way is to use **LOCK parent_table IN SHARE ROW EXCLUSIVE MODE** command if a transaction is going to update/delete a primary key and use **LOCK parent_table IN SHARE MODE** command if a transaction is going to update/insert a foreign key.

> **Note:** Note that if you run a transaction in SERIALIZABLE mode then you must execute the **LOCK** commands above before execution of any DML statement (**SELECT/INSERT/DELETE/UPDATE/FETCH/COPY_TO**) in the transaction.

These inconveniences will disappear in the future when the ability to read dirty (uncommitted) data (regardless of isolation level) and true referential integrity will be implemented.

A.13.2. Changes

```
Bug Fixes
---------
Fix text<->float8 and text<->float4 conversion functions(Thomas)
```

```
Fix for creating tables with mixed-case constraints(Billy)
Change exp()/pow() behavior to generate error on underflow/overflow(Jan)
Fix bug in pg_dump -z
Memory overrun cleanups(Tatsuo)
Fix for lo_import crash(Tatsuo)
Adjust handling of data type names to suppress double quotes(Thomas)
Use type coercion for matching columns and DEFAULT(Thomas)
Fix deadlock so it only checks once after one second of sleep(Bruce)
Fixes for aggregates and PL/pgsql(Hiroshi)
Fix for subquery crash(Vadim)
Fix for libpq function PQfnumber and case-insensitive names(Bahman Rafatjoo)
Fix for large object write-in-middle, no extra block, memory consumption(Tatsuo)
Fix for pg_dump -d or -D and  quote special characters in INSERT
Repair serious problems with dynahash(Tom)
Fix INET/CIDR portability problems
Fix problem with selectivity error in ALTER TABLE ADD COLUMN(Bruce)
Fix executor so mergejoin of different column types works(Tom)
Fix for Alpha OR selectivity bug
Fix OR index selectivity problem(Bruce)
Fix so \d shows proper length for char()/varchar()(Ryan)
Fix tutorial code(Clark)
Improve destroyuser checking(Oliver)
Fix for Kerberos(Rodney McDuff)
Fix for dropping database while dirty buffers(Bruce)
Fix so sequence nextval() can be case-sensitive(Bruce)
Fix !!= operator
Drop buffers before destroying database files(Bruce)
Fix case where executor evaluates functions twice(Tatsuo)
Allow sequence nextval actions to be case-sensitive(Bruce)
Fix optimizer indexing not working for negative numbers(Bruce)
Fix for memory leak in executor with fjIsNull
Fix for aggregate memory leaks(Erik Riedel)
Allow username containing a dash GRANT permissions
Cleanup of NULL in inet types
Clean up system table bugs(Tom)
Fix problems of PAGER and \? command(Masaaki Sakaida)
Reduce default multi-segment file size limit to 1GB(Peter)
Fix for dumping of CREATE OPERATOR(Tom)
Fix for backward scanning of cursors(Hiroshi Inoue)
Fix for COPY FROM STDIN when using \i(Tom)
Fix for subselect is compared inside an expression(Jan)
Fix handling of error reporting while returning rows(Tom)
Fix problems with reference to array types(Tom,Jan)
Prevent UPDATE SET oid(Jan)
Fix pg_dump so -t option can handle case-sensitive tablenames
Fixes for GROUP BY in special cases(Tom, Jan)
Fix for memory leak in failed queries(Tom)
DEFAULT now supports mixed-case identifiers(Tom)
Fix for multi-segment uses of DROP/RENAME table, indexes(Ole Gjerde)
Disable use of pg_dump with both -o and -d options(Bruce)
Allow pg_dump to properly dump GROUP permissions(Bruce)
Fix GROUP BY in INSERT INTO table SELECT * FROM table2(Jan)
Fix for computations in views(Jan)
Fix for aggregates on array indexes(Tom)
Fix for DEFAULT handles single quotes in value requiring too many quotes
Fix security problem with non-super users importing/exporting large objects(Tom)
Rollback of transaction that creates table cleaned up properly(Tom)
```

```
Fix to allow long table and column names to generate proper serial names(Tom)

Enhancements
------------
Add "vacuumdb" utility
Speed up libpq by allocating memory better(Tom)
EXPLAIN all indexes used(Tom)
Implement CASE, COALESCE, NULLIF  expression(Thomas)
New pg_dump table output format(Constantin)
Add string min()/max() functions(Thomas)
Extend new type coercion techniques to aggregates(Thomas)
New moddatetime contrib(Terry)
Update to pgaccess 0.96(Constantin)
Add routines for single-byte "char" type(Thomas)
Improved substr() function(Thomas)
Improved multibyte handling(Tatsuo)
Multi-version concurrency control/MVCC(Vadim)
New Serialized mode(Vadim)
Fix for tables over 2gigs(Peter)
New SET TRANSACTION ISOLATION LEVEL(Vadim)
New LOCK TABLE IN ... MODE(Vadim)
Update ODBC driver(Byron)
New NUMERIC data type(Jan)
New SELECT FOR UPDATE(Vadim)
Handle "NaN" and "Infinity" for input values(Jan)
Improved date/year handling(Thomas)
Improved handling of backend connections(Magnus)
New options ELOG_TIMESTAMPS and USE_SYSLOG options for log files(Massimo)
New TCL_ARRAYS option(Massimo)
New INTERSECT and EXCEPT(Stefan)
New pg_index.indisprimary for primary key tracking(D'Arcy)
New pg_dump option to allow dropping of tables before creation(Brook)
Speedup of row output routines(Tom)
New READ COMMITTED isolation level(Vadim)
New TEMP tables/indexes(Bruce)
Prevent sorting if result is already sorted(Jan)
New memory allocation optimization(Jan)
Allow psql to do \p\g(Bruce)
Allow multiple rule actions(Jan)
Added LIMIT/OFFSET functionality(Jan)
Improve optimizer when joining a large number of tables(Bruce)
New intro to SQL from S. Simkovics' Master's Thesis (Stefan, Thomas)
New intro to backend processing from S. Simkovics' Master's Thesis (Stefan)
Improved int8 support(Ryan Bradetich, Thomas, Tom)
New routines to convert between int8 and text/varchar types(Thomas)
New bushy plans, where meta-tables are joined(Bruce)
Enable right-hand queries by default(Bruce)
Allow reliable maximum number of backends to be set at configure time
      (--with-maxbackends and postmaster switch (-N backends))(Tom)
GEQO default now 10 tables because of optimizer speedups(Tom)
Allow NULL=Var for MS-SQL portability(Michael, Bruce)
Modify contrib check_primary_key() so either "automatic" or "dependent"(Anand)
Allow psql \d on a view show query(Ryan)
Speedup for LIKE(Bruce)
Ecpg fixes/features, see src/interfaces/ecpg/ChangeLog file(Michael)
JDBC fixes/features, see src/interfaces/jdbc/CHANGELOG(Peter)
Make % operator have precedence like /(Bruce)
```

```
Add new postgres -O option to allow system table structure changes(Bruce)
Update contrib/pginterface/findoidjoins script(Tom)
Major speedup in vacuum of deleted rows with indexes(Vadim)
Allow non-SQL functions to run different versions based on arguments(Tom)
Add -E option that shows actual queries sent by \dt and friends(Masaaki Sakaida)
Add version number in start-up banners for psql(Masaaki Sakaida)
New contrib/vacuumlo removes large objects not referenced(Peter)
New initialization for table sizes so non-vacuumed tables perform better(Tom)
Improve error messages when a connection is rejected(Tom)
Support for arrays of char() and varchar() fields(Massimo)
Overhaul of hash code to increase reliability and performance(Tom)
Update to PyGreSQL 2.4(D'Arcy)
Changed debug options so -d4 and -d5 produce different node displays(Jan)
New pg_options: pretty_plan, pretty_parse, pretty_rewritten(Jan)
Better optimization statistics for system table access(Tom)
Better handling of non-default block sizes(Massimo)
Improve GEQO optimizer memory consumption(Tom)
UNION now suppports ORDER BY of columns not in target list(Jan)
Major libpq++ improvements(Vince Vielhaber)
pg_dump now uses -z(ACL's) as default(Bruce)
backend cache, memory speedups(Tom)
have pg_dump do everything in one snapshot transaction(Vadim)
fix for large object memory leakage, fix for pg_dumping(Tom)
INET type now respects netmask for comparisons
Make VACUUM ANALYZE only use a readlock(Vadim)
Allow VIEWs on UNIONS(Jan)
pg_dump now can generate consistent snapshots on active databases(Vadim)

Source Tree Changes
-------------------
Improve port matching(Tom)
Portability fixes for SunOS
Add NT/Win32 backend port and enable dynamic loading(Magnus and Daniel Horak)
New port to Cobalt Qube(Mips) running Linux(Tatsuo)
Port to NetBSD/m68k(Mr. Mutsuki Nakajima)
Port to NetBSD/sun3(Mr. Mutsuki Nakajima)
Port to NetBSD/macppc(Toshimi Aoki)
Fix for tcl/tk configuration(Vince)
Removed CURRENT keyword for rule queries(Jan)
NT dynamic loading now works(Daniel Horak)
Add ARM32 support(Andrew McMurry)
Better support for HP-UX 11 and UnixWare
Improve file handling to be more uniform, prevent file descriptor leak(Tom)
New install commands for plpgsql(Jan)
```

A.14. Release 6.4.2

Release date: 1998-12-20

The 6.4.1 release was improperly packaged. This also has one additional bug fix.

A.14.1. Migration to version 6.4.2

A dump/restore is *not* required for those running 6.4.*.

A.14.2. Changes

```
Fix for datetime constant problem on some platforms(Thomas)
```

A.15. Release 6.4.1

Release date: 1998-12-18

This is basically a cleanup release for 6.4. We have fixed a variety of problems reported by 6.4 users.

A.15.1. Migration to version 6.4.1

A dump/restore is *not* required for those running 6.4.

A.15.2. Changes

```
Add pg_dump -N flag to force double quotes around identifiers.  This is
 the default(Thomas)
Fix for NOT in where clause causing crash(Bruce)
EXPLAIN VERBOSE coredump fix(Vadim)
Fix shared-library problems on Linux
Fix test for table existance to allow mixed-case and whitespace in
 the table name(Thomas)
Fix a couple of pg_dump bugs
Configure matches template/.similar entries better(Tom)
Change builtin function names from SPI_* to spi_*
OR WHERE clause fix(Vadim)
Fixes for mixed-case table names(Billy)
contrib/linux/postgres.init.csh/sh fix(Thomas)
libpq memory overrun fix
SunOS fixes(Tom)
Change exp() behavior to generate error on underflow(Thomas)
pg_dump fixes for memory leak, inheritance constraints, layout change
update pgaccess to 0.93
Fix prototype for 64-bit platforms
Multibyte fixes(Tatsuo)
New ecpg man page
Fix memory overruns(Tatsuo)
Fix for lo_import() crash(Bruce)
Better search for install program(Tom)
Timezone fixes(Tom)
HP-UX fixes(Tom)
Use implicit type coercion for matching DEFAULT values(Thomas)
```

```
Add routines to help with single-byte (internal) character type(Thomas)
Compilation of libpq for Win32 fixes(Magnus)
Upgrade to PyGreSQL 2.2(D'Arcy)
```

A.16. Release 6.4

Release date: 1998-10-30

There are *many* new features and improvements in this release. Thanks to our developers and maintainers, nearly every aspect of the system has received some attention since the previous release. Here is a brief, incomplete summary:

- Views and rules are now functional thanks to extensive new code in the rewrite rules system from Jan Wieck. He also wrote a chapter on it for the *Programmer's Guide*.
- Jan also contributed a second procedural language, PL/pgSQL, to go with the original PL/pgTCL procedural language he contributed last release.
- We have optional multiple-byte character set support from Tatsuo Iishi to complement our existing locale support.
- Client/server communications has been cleaned up, with better support for asynchronous messages and interrupts thanks to Tom Lane.
- The parser will now perform automatic type coercion to match arguments to available operators and functions, and to match columns and expressions with target columns. This uses a generic mechanism which supports the type extensibility features of PostgreSQL. There is a new chapter in the *User's Guide* which covers this topic.
- Three new data types have been added. Two types, `inet` and `cidr`, support various forms of IP network, subnet, and machine addressing. There is now an 8-byte integer type available on some platforms. See the chapter on data types in the *User's Guide* for details. A fourth type, `serial`, is now supported by the parser as an amalgam of the `int4` type, a sequence, and a unique index.
- Several more SQL92-compatible syntax features have been added, including **INSERT DEFAULT VALUES**
- The automatic configuration and installation system has received some attention, and should be more robust for more platforms than it has ever been.

A.16.1. Migration to version 6.4

A dump/restore using pg_dump or pg_dumpall is required for those wishing to migrate data from any previous release of PostgreSQL.

A.16.2. Changes

```
Bug Fixes
---------
Fix for a tiny memory leak in PQsetdb/PQfinish(Bryan)
Remove char2-16 data types, use char/varchar(Darren)
Pqfn not handles a NOTICE message(Anders)
Reduced busywaiting overhead for spinlocks with many backends (dg)
Stuck spinlock detection (dg)
Fix up "ISO-style" timespan decoding and encoding(Thomas)
Fix problem with table drop after rollback of transaction(Vadim)
Change error message and remove non-functional update message(Vadim)
Fix for COPY array checking
Fix for SELECT 1 UNION SELECT NULL
Fix for buffer leaks in large object calls(Pascal)
Change owner from oid to int4 type(Bruce)
Fix a bug in the oracle compatibility functions btrim() ltrim() and rtrim()
Fix for shared invalidation cache overflow(Massimo)
Prevent file descriptor leaks in failed COPY's(Bruce)
Fix memory leak in libpgtcl's pg_select(Constantin)
Fix problems with username/passwords over 8 characters(Tom)
Fix problems with handling of asynchronous NOTIFY in backend(Tom)
Fix of many bad system table entries(Tom)

Enhancements
------------
Upgrade ecpg and ecpglib,see src/interfaces/ecpc/ChangeLog(Michael)
Show the index used in an EXPLAIN(Zeugswetter)
EXPLAIN  invokes  rule system and shows plan(s) for rewritten queries(Jan)
Multibyte awareness of many data types and functions, via configure(Tatsuo)
New configure --with-mb option(Tatsuo)
New initdb --pgencoding option(Tatsuo)
New createdb -E multibyte option(Tatsuo)
Select version(); now returns PostgreSQL version(Jeroen)
Libpq now allows asynchronous clients(Tom)
Allow cancel from client of backend query(Tom)
Psql now cancels query with Control-C(Tom)
Libpq users need not issue dummy queries to get NOTIFY messages(Tom)
NOTIFY now sends sender's PID, so you can tell whether it was your own(Tom)
PGresult struct now includes associated error message, if any(Tom)
Define "tz_hour" and "tz_minute" arguments to date_part()(Thomas)
Add routines to convert between varchar and bpchar(Thomas)
Add routines to allow sizing of varchar and bpchar into target columns(Thomas)
Add bit flags to support timezonehour and minute in data retrieval(Thomas)
Allow more variations on valid floating point numbers (e.g. ".1", "1e6")(Thomas)
Fixes for unary minus parsing with leading spaces(Thomas)
Implement TIMEZONE_HOUR, TIMEZONE_MINUTE per SQL92 specs(Thomas)
Check for and properly ignore FOREIGN KEY column constraints(Thomas)
Define USER as synonym for CURRENT_USER per SQL92 specs(Thomas)
Enable HAVING clause but no fixes elsewhere yet.
Make "char" type a synonym for "char(1)" (actually implemented as bpchar)(Thomas)
Save string type if specified for DEFAULT clause handling(Thomas)
Coerce operations involving different data types(Thomas)
Allow some index use for columns of different types(Thomas)
Add capabilities for automatic type conversion(Thomas)
Cleanups for large objects, so file is truncated on open(Peter)
Readline cleanups(Tom)
```

Allow psql \f \ to make spaces as delimiter(Bruce)
Pass pg_attribute.atttypmod to the frontend for column field lengths(Tom,Bruce)
Msql compatibility library in /contrib(Aldrin)
Remove the requirement that ORDER/GROUP BY clause identifiers be
included in the target list(David)
Convert columns to match columns in UNION clauses(Thomas)
Remove fork()/exec() and only do fork()(Bruce)
Jdbc cleanups(Peter)
Show backend status on ps command line(only works on some platforms)(Bruce)
Pg_hba.conf now has a sameuser option in the database field
Make lo_unlink take oid param, not int4
New DISABLE_COMPLEX_MACRO for compilers that can't handle our macros(Bruce)
Libpgtcl now handles NOTIFY as a Tcl event, need not send dummy queries(Tom)
libpgtcl cleanups(Tom)
Add -error option to libpgtcl's pg_result command(Tom)
New locale patch, see docs/README/locale(Oleg)
Fix for pg_dump so CONSTRAINT and CHECK syntax is correct(ccb)
New contrib/lo code for large object orphan removal(Peter)
New psql command "SET CLIENT_ENCODING TO 'encoding'" for multibytes
feature, see /doc/README.mb(Tatsuo)
/contrib/noupdate code to revoke update permission on a column
Libpq can now be compiled on win32(Magnus)
Add PQsetdbLogin() in libpq
New 8-byte integer type, checked by configure for OS support(Thomas)
Better support for quoted table/column names(Thomas)
Surround table and column names with double-quotes in pg_dump(Thomas)
PQreset() now works with passwords(Tom)
Handle case of GROUP BY target list column number out of range(David)
Allow UNION in subselects
Add auto-size to screen to \d? commands(Bruce)
Use UNION to show all \d? results in one query(Bruce)
Add \d? field search feature(Bruce)
Pg_dump issues fewer \connect requests(Tom)
Make pg_dump -z flag work better, document it in manual page(Tom)
Add HAVING clause with full support for subselects and unions(Stephan)
Full text indexing routines in contrib/fulltextindex(Maarten)
Transaction ids now stored in shared memory(Vadim)
New PGCLIENTENCODING when issuing COPY command(Tatsuo)
Support for SQL92 syntax "SET NAMES"(Tatsuo)
Support for LATIN2-5(Tatsuo)
Add UNICODE regression test case(Tatsuo)
Lock manager cleanup, new locking modes for LLL(Vadim)
Allow index use with OR clauses(Bruce)
Allows "SELECT NULL ORDER BY 1;"
Explain VERBOSE prints the plan, and now pretty-prints the plan to
the postmaster log file(Bruce)
Add indexes display to \d command(Bruce)
Allow GROUP BY on functions(David)
New pg_class.relkind for large objects(Bruce)
New way to send libpq NOTICE messages to a different location(Tom)
New \w write command to psql(Bruce)
New /contrib/findoidjoins scans oid columns to find join relationships(Bruce)
Allow binary-compatible indexes to be considered when checking for valid
Indexes for restriction clauses containing a constant(Thomas)
New ISBN/ISSN code in /contrib/isbn_issn
Allow NOT LIKE, IN, NOT IN, BETWEEN, and NOT BETWEEN constraint(Thomas)
New rewrite system fixes many problems with rules and views(Jan)

```
 * Rules on relations work
 * Event qualifications on insert/update/delete work
 * New OLD variable to reference CURRENT, CURRENT will be remove in future
 * Update rules can reference NEW and OLD in rule qualifications/actions
 * Insert/update/delete rules on views work
 * Multiple rule actions are now supported, surrounded by parentheses
 * Regular users can create views/rules on tables they have RULE permits
 * Rules and views inherit the permissions on the creator
 * No rules at the column level
 * No UPDATE NEW/OLD rules
 * New pg_tables, pg_indexes, pg_rules and pg_views system views
 * Only a single action on SELECT rules
 * Total rewrite overhaul, perhaps for 6.5
 * handle subselects
 * handle aggregates on views
 * handle insert into select from view works
System indexes are now multi-key(Bruce)
Oidint2, oidint4, and oidname types are removed(Bruce)
Use system cache for more system table lookups(Bruce)
New backend programming language PL/pgSQL in backend/pl(Jan)
New SERIAL data type, auto-creates sequence/index(Thomas)
Enable assert checking without a recompile(Massimo)
User lock enhancements(Massimo)
New setval() command to set sequence value(Massimo)
Auto-remove unix socket file on start-up if no postmaster running(Massimo)
Conditional trace package(Massimo)
New UNLISTEN command(Massimo)
Psql and libpq now compile under win32 using win32.mak(Magnus)
Lo_read no longer stores trailing NULL(Bruce)
Identifiers are now truncated to 31 characters internally(Bruce)
Createuser options now availble on the command line
Code for 64-bit integer supported added, configure tested, int8 type(Thomas)
Prevent file descriptor leaf from failed COPY(Bruce)
New pg_upgrade command(Bruce)
Updated /contrib directories(Massimo)
New CREATE TABLE DEFAULT VALUES statement available(Thomas)
New INSERT INTO TABLE DEFAULT VALUES statement available(Thomas)
New DECLARE and FETCH feature(Thomas)
libpq's internal structures now not exported(Tom)
Allow up to 8 key indexes(Bruce)
Remove ARCHIVE keyword, that is no longer used(Thomas)
pg_dump -n flag to supress quotes around indentifiers
disable system columns for views(Jan)
new INET and CIDR types for network addresses(TomH, Paul)
no more double quotes in psql output
pg_dump now dumps views(Terry)
new SET QUERY_LIMIT(Tatsuo,Jan)

Source Tree Changes
-------------------
/contrib cleanup(Jun)
Inline some small functions called for every row(Bruce)
Alpha/linux fixes
HP-UX cleanups(Tom)
Multibyte regression tests(Soonmyung.)
Remove --disabled options from configure
Define PGDOC to use POSTGRESDIR by default
```

```
Make regression optional
Remove extra braces code to pgindent(Bruce)
Add bsdi shared library support(Bruce)
New --without-CXX support configure option(Brook)
New FAQ_CVS
Update backend flowchart in tools/backend(Bruce)
Change atttypmod from int16 to int32(Bruce, Tom)
Getrusage() fix for platforms that do not have it(Tom)
Add PQconnectdb, PGUSER, PGPASSWORD to libpq man page
NS32K platform fixes(Phil Nelson, John Buller)
SCO 7/UnixWare 2.x fixes(Billy,others)
Sparc/Solaris 2.5 fixes(Ryan)
Pgbuiltin.3 is obsolete, move to doc files(Thomas)
Even more documention(Thomas)
Nextstep support(Jacek)
Aix support(David)
pginterface manual page(Bruce)
shared libraries all have version numbers
merged all OS-specific shared library defines into one file
smarter TCL/TK configuration checking(Billy)
smarter perl configuration(Brook)
configure uses supplied install-sh if no install script found(Tom)
new Makefile.shlib for shared library configuration(Tom)
```

A.17. Release 6.3.2

Release date: 1998-04-07

This is a bug-fix release for 6.3.x. Refer to the release notes for version 6.3 for a more complete summary of new features.

Summary:

- Repairs automatic configuration support for some platforms, including Linux, from breakage inadvertently introduced in version 6.3.1.
- Correctly handles function calls on the left side of BETWEEN and LIKE clauses.

A dump/restore is NOT required for those running 6.3 or 6.3.1. A `make distclean`, `make`, and `make install` is all that is required. This last step should be performed while the postmaster is not running. You should re-link any custom applications that use PostgreSQL libraries.

For upgrades from pre-6.3 installations, refer to the installation and migration instructions for version 6.3.

A.17.1. Changes

```
Configure detection improvements for tcl/tk(Brook Milligan, Alvin)
Manual page improvements(Bruce)
BETWEEN and LIKE fix(Thomas)
fix for psql \connect used by pg_dump(Oliver Elphick)
New odbc driver
pgaccess, version 0.86
qsort removed, now uses libc version, cleanups(Jeroen)
fix for buffer over-runs detected(Maurice Gittens)
fix for buffer overrun in libpgtcl(Randy Kunkee)
fix for UNION with DISTINCT or ORDER BY(Bruce)
gettimeofday configure check(Doug Winterburn)
Fix "indexes not used" bug(Vadim)
docs additions(Thomas)
Fix for backend memory leak(Bruce)
libreadline cleanup(Erwan MAS)
Remove DISTDIR(Bruce)
Makefile dependency cleanup(Jeroen van Vianen)
ASSERT fixes(Bruce)
```

A.18. Release 6.3.1

Release date: 1998-03-23

Summary:

- Additional support for multibyte character sets.
- Repair byte ordering for mixed-endian clients and servers.
- Minor updates to allowed SQL syntax.
- Improvements to the configuration autodetection for installation.

A dump/restore is NOT required for those running 6.3. A `make distclean`, `make`, and `make install` is all that is required. This last step should be performed while the postmaster is not running. You should re-link any custom applications that use PostgreSQL libraries.

For upgrades from pre-6.3 installations, refer to the installation and migration instructions for version 6.3.

A.18.1. Changes

```
ecpg cleanup/fixes, now version 1.1(Michael Meskes)
pg_user cleanup(Bruce)
large object fix for pg_dump and tclsh (alvin)
```

```
LIKE fix for multiple adjacent underscores
fix for redefining builtin functions(Thomas)
ultrix4 cleanup
upgrade to pg_access 0.83
updated CLUSTER manual page
multibyte character set support, see doc/README.mb(Tatsuo)
configure --with-pgport fix
pg_ident fix
big-endian fix for backend communications(Kataoka)
SUBSTR() and substring() fix(Jan)
several jdbc fixes(Peter)
libpgtcl improvements, see libptcl/README(Randy Kunkee)
Fix for "Datasize = 0" error(Vadim)
Prevent \do from wrapping(Bruce)
Remove duplicate Russian character set entries
Sunos4 cleanup
Allow optional TABLE keyword in LOCK and SELECT INTO(Thomas)
CREATE SEQUENCE options to allow a negative integer(Thomas)
Add "PASSWORD" as an allowed column identifier(Thomas)
Add checks for UNION target fields(Bruce)
Fix Alpha port(Dwayne Bailey)
Fix for text arrays containing quotes(Doug Gibson)
Solaris compile fix(Albert Chin-A-Young)
Better identify tcl and tk libs and includes(Bruce)
```

A.19. Release 6.3

Release date: 1998-03-01

There are *many* new features and improvements in this release. Here is a brief, incomplete summary:

- Many new SQL features, including full SQL92 subselect capability (everything is here but target-list subselects).
- Support for client-side environment variables to specify time zone and date style.
- Socket interface for client/server connection. This is the default now so you may need to start postmaster with the `-i` flag.
- Better password authorization mechanisms. Default table permissions have changed.
- Old-style *time travel* has been removed. Performance has been improved.

Note: Bruce Momjian wrote the following notes to introduce the new release.

There are some general 6.3 issues that I want to mention. These are only the big items that can not be described in one sentence. A review of the detailed changes list is still needed.

First, we now have subselects. Now that we have them, I would like to mention that without subselects, SQL is a very limited language. Subselects are a major feature, and you should review your code for places where subselects provide a better solution for your queries. I think you will find that there are more uses for subselects than you may think. Vadim has put us on the big SQL map with subselects, and fully functional ones too. The only thing you can't do with subselects is to use them in the target list.

Second, 6.3 uses Unix domain sockets rather than TCP/IP by default. To enable connections from other machines, you have to use the new postmaster -i option, and of course edit `pg_hba.conf`. Also, for this reason, the format of `pg_hba.conf` has changed.

Third, `char()` fields will now allow faster access than `varchar()` or `text`. Specifically, the `text` and `varchar()` have a penalty for access to any columns after the first column of this type. `char()` used to also have this access penalty, but it no longer does. This may suggest that you redesign some of your tables, especially if you have short character columns that you have defined as `varchar()` or `text`. This and other changes make 6.3 even faster than earlier releases.

We now have passwords definable independent of any Unix file. There are new SQL USER commands. See the *Administrator's Guide* for more information. There is a new table, pg_shadow, which is used to store user information and user passwords, and it by default only SELECT-able by the postgres super-user. pg_user is now a view of pg_shadow, and is SELECT-able by PUBLIC. You should keep using pg_user in your application without changes.

User-created tables now no longer have SELECT permission to PUBLIC by default. This was done because the ANSI standard requires it. You can of course GRANT any permissions you want after the table is created. System tables continue to be SELECT-able by PUBLIC.

We also have real deadlock detection code. No more sixty-second timeouts. And the new locking code implements a FIFO better, so there should be less resource starvation during heavy use.

Many complaints have been made about inadequate documentation in previous releases. Thomas has put much effort into many new manuals for this release. Check out the doc/ directory.

For performance reasons, time travel is gone, but can be implemented using triggers (see `pgsql/contrib/spi/README`). Please check out the new \d command for types, operators, etc. Also, views have their own permissions now, not based on the underlying tables, so permissions on them have to be set separately. Check `/pgsql/interfaces` for some new ways to talk to PostgreSQL.

This is the first release that really required an explanation for existing users. In many ways, this was necessary because the new release removes many limitations, and the work-arounds people were using are no longer needed.

A.19.1. Migration to version 6.3

A dump/restore using pg_dump or pg_dumpall is required for those wishing to migrate data from any previous release of PostgreSQL.

A.19.2. Changes

```
Bug Fixes
---------
Fix binary cursors broken by MOVE implementation(Vadim)
```

```
Fix for tcl library crash(Jan)
Fix for array handling, from Gerhard Hintermayer
Fix acl error, and remove duplicate pqtrace(Bruce)
Fix psql \e for empty file(Bruce)
Fix for textcat on varchar() fields(Bruce)
Fix for DBT Sendproc (Zeugswetter Andres)
Fix vacuum analyze syntax problem(Bruce)
Fix for international identifiers(Tatsuo)
Fix aggregates on inherited tables(Bruce)
Fix substr() for out-of-bounds data
Fix for select 1=1 or 2=2, select 1=1 and 2=2, and select sum(2+2)(Bruce)
Fix notty output to show status result.  -q option still turns it off(Bruce)
Fix for count(*), aggs with views and multiple tables and sum(3)(Bruce)
Fix cluster(Bruce)
Fix for PQtrace start/stop several times(Bruce)
Fix a variety of locking problems like newer lock waiters getting
 lock before older waiters, and having readlock people not share
 locks if a writer is waiting for a lock, and waiting writers not
 getting priority over waiting readers(Bruce)
Fix crashes in psql when executing queries from external files(James)
Fix problem with multiple order by columns, with the first one having
 NULL values(Jeroen)
Use correct hash table support functions for float8 and int4(Thomas)
Re-enable JOIN= option in CREATE OPERATOR statement (Thomas)
Change precedence for boolean operators to match expected behavior(Thomas)
Generate elog(ERROR) on over-large integer(Bruce)
Allow multiple-argument functions in constraint clauses(Thomas)
Check boolean input literals for 'true','false','yes','no','1','0'
 and throw elog(ERROR) if unrecognized(Thomas)
Major large objects fix
Fix for GROUP BY showing duplicates(Vadim)
Fix for index scans in MergeJion(Vadim)

Enhancements
------------
Subselects with EXISTS, IN, ALL, ANY keywords (Vadim, Bruce, Thomas)
New User Manual(Thomas, others)
Speedup by inlining some frequently-called functions
Real deadlock detection, no more timeouts(Bruce)
Add SQL92 "constants" CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP,
 CURRENT_USER(Thomas)
Modify constraint syntax to be SQL92-compliant(Thomas)
Implement SQL92 PRIMARY KEY and UNIQUE clauses using indexes(Thomas)
Recognize SQL92 syntax for FOREIGN KEY. Throw elog notice(Thomas)
Allow NOT NULL UNIQUE constraint clause (each allowed separately before)(Thomas)
Allow PostgreSQL-style casting ("::") of non-constants(Thomas)
Add support for SQL3 TRUE and FALSE boolean constants(Thomas)
Support SQL92 syntax for IS TRUE/IS FALSE/IS NOT TRUE/IS NOT FALSE(Thomas)
Allow shorter strings for boolean literals (e.g. "t", "tr", "tru")(Thomas)
Allow SQL92 delimited identifiers(Thomas)
Implement SQL92 binary and hexadecimal string decoding (b'10' and x'1F')(Thomas)
Support SQL92 syntax for type coercion of literal strings
 (e.g. "DATETIME 'now'")(Thomas)
Add conversions for int2, int4, and OID types to and from text(Thomas)
Use shared lock when building indexes(Vadim)
Free memory allocated for an user query inside transaction block after
 this query is done, was turned off in <= 6.2.1(Vadim)
```

```
New SQL statement CREATE PROCEDURAL LANGUAGE(Jan)
New PostgreSQL Procedural Language (PL) backend interface(Jan)
Rename pg_dump -H option to -h(Bruce)
Add Java support for passwords, European dates(Peter)
Use indexes for LIKE and ~, !~ operations(Bruce)
Add hash functions for datetime and timespan(Thomas)
Time Travel removed(Vadim, Bruce)
Add paging for \d and \z, and fix \i(Bruce)
Add Unix domain socket support to backend and to frontend library(Goran)
Implement CREATE DATABASE/WITH LOCATION and initlocation utility(Thomas)
Allow more SQL92 and/or PostgreSQL reserved words as column identifiers(Thomas)
Augment support for SQL92 SET TIME ZONE...(Thomas)
SET/SHOW/RESET TIME ZONE uses TZ backend environment variable(Thomas)
Implement SET keyword = DEFAULT and SET TIME ZONE DEFAULT(Thomas)
Enable SET TIME ZONE using TZ environment variable(Thomas)
Add PGDATESTYLE environment variable to frontend and backend initialization(Thomas)
Add PGTZ, PGCOSTHEAP, PGCOSTINDEX, PGRPLANS, PGGEQO
 frontend library initialization environment variables(Thomas)
Regression tests time zone automatically set with "setenv PGTZ PST8PDT"(Thomas)
Add pg_description table for info on tables, columns, operators, types, and
 aggregates(Bruce)
Increase 16 char limit on system table/index names to 32 characters(Bruce)
Rename system indexes(Bruce)
Add 'GERMAN' option to SET DATESTYLE(Thomas)
Define an "ISO-style" timespan output format with "hh:mm:ss" fields(Thomas)
Allow fractional values for delta times (e.g. '2.5 days')(Thomas)
Validate numeric input more carefully for delta times(Thomas)
Implement day of year as possible input to date_part()(Thomas)
Define timespan_finite() and text_timespan() functions(Thomas)
Remove archive stuff(Bruce)
Allow for a pg_password authentication database that is separate from
 the system password file(Todd)
Dump ACLs, GRANT, REVOKE permissions(Matt)
Define text, varchar, and bpchar string length functions(Thomas)
Fix Query handling for inheritance, and cost computations(Bruce)
Implement CREATE TABLE/AS SELECT (alternative to SELECT/INTO)(Thomas)
Allow NOT, IS NULL, IS NOT NULL in constraints(Thomas)
Implement UNIONs for SELECT(Bruce)
Add UNION, GROUP, DISTINCT to INSERT(Bruce)
varchar() stores only necessary bytes on disk(Bruce)
Fix for BLOBs(Peter)
Mega-Patch for JDBC...see README_6.3 for list of changes(Peter)
Remove unused "option" from PQconnectdb()
New LOCK command and lock manual page describing deadlocks(Bruce)
Add new psql \da, \dd, \df, \do, \dS, and \dT commands(Bruce)
Enhance psql \z to show sequences(Bruce)
Show NOT NULL and DEFAULT in psql \d table(Bruce)
New psql .psqlrc file start-up(Andrew)
Modify sample start-up script in contrib/linux to show syslog(Thomas)
New types for IP and MAC addresses in contrib/ip_and_mac(TomH)
Unix system time conversions with date/time types in contrib/unixdate(Thomas)
Update of contrib stuff(Massimo)
Add Unix socket support to DBD::Pg(Goran)
New python interface (PyGreSQL 2.0)(D'Arcy)
New frontend/backend protocol has a version number, network byte order(Phil)
Security features in pg_hba.conf enhanced and documented, many cleanups(Phil)
CHAR() now faster access than VARCHAR() or TEXT
```

```
ecpg embedded SQL preprocessor
Reduce system column overhead(Vadmin)
Remove pg_time table(Vadim)
Add pg_type attribute to identify types that need length (bpchar, varchar)
Add report of offending line when COPY command fails
Allow VIEW permissions to be set separately from the underlying tables.
 For security, use GRANT/REVOKE on views as appropriate(Jan)
Tables now have no default GRANT SELECT TO PUBLIC.  You must
 explicitly grant such permissions.
Clean up tutorial examples(Darren)

Source Tree Changes
-------------------
Add new html development tools, and flow chart in /tools/backend
Fix for SCO compiles
Stratus computer port Robert Gillies
Added support for shlib for BSD44_derived & i386_solaris
Make configure more automated(Brook)
Add script to check regression test results
Break parser functions into smaller files, group together(Bruce)
Rename heap_create to heap_create_and_catalog, rename heap_creatr
 to heap_create()(Bruce)
Sparc/Linux patch for locking(TomS)
Remove PORTNAME and reorganize port-specific stuff(Marc)
Add optimizer README file(Bruce)
Remove some recursion in optimizer and clean up some code there(Bruce)
Fix for NetBSD locking(Henry)
Fix for libptcl make(Tatsuo)
AIX patch(Darren)
Change IS TRUE, IS FALSE, ... to expressions using "=" rather than
 function calls to istrue() or isfalse() to allow optimization(Thomas)
Various fixes NetBSD/Sparc related(TomH)
Alpha linux locking(Travis,Ryan)
Change elog(WARN) to elog(ERROR)(Bruce)
FAQ for FreeBSD(Marc)
Bring in the PostODBC source tree as part of our standard distribution(Marc)
A minor patch for HP/UX 10 vs 9(Stan)
New pg_attribute.atttypmod for type-specific info like varchar length(Bruce)
UnixWare patches(Billy)
New i386 'lock' for spin lock asm(Billy)
Support for multiplexed backends is removed
Start an OpenBSD port
Start an AUX port
Start a Cygnus port
Add string functions to regression suite(Thomas)
Expand a few function names formerly truncated to 16 characters(Thomas)
Remove un-needed malloc() calls and replace with palloc()(Bruce)
```

A.20. Release 6.2.1

Release date: 1997-10-17

6.2.1 is a bug-fix and usability release on 6.2.

Summary:

- Allow strings to span lines, per SQL92.
- Include example trigger function for inserting user names on table updates.

This is a minor bug-fix release on 6.2. For upgrades from pre-6.2 systems, a full dump/reload is required. Refer to the 6.2 release notes for instructions.

A.20.1. Migration from version 6.2 to version 6.2.1

This is a minor bug-fix release. A dump/reload is not required from version 6.2, but is required from any release prior to 6.2.

In upgrading from version 6.2, if you choose to dump/reload you will find that avg(money) is now calculated correctly. All other bug fixes take effect upon updating the executables.

Another way to avoid dump/reload is to use the following SQL command from **psql** to update the existing system table:

```
update pg_aggregate set aggfinalfn = 'cash_div_flt8'
 where aggname = 'avg' and aggbasetype = 790;
```

This will need to be done to every existing database, including template1.

A.20.2. Changes

```
Allow TIME and TYPE column names(Thomas)
Allow larger range of true/false as boolean values(Thomas)
Support output of "now" and "current"(Thomas)
Handle DEFAULT with INSERT of NULL properly(Vadim)
Fix for relation reference counts problem in buffer manager(Vadim)
Allow strings to span lines, like ANSI(Thomas)
Fix for backward cursor with ORDER BY(Vadim)
Fix avg(cash) computation(Thomas)
Fix for specifying a column twice in ORDER/GROUP BY(Vadim)
Documented new libpq function to return affected rows, PQcmdTuples(Bruce)
Trigger function for inserting user names for INSERT/UPDATE(Brook Milligan)
```

A.21. Release 6.2

Release date: 1997-10-02

A dump/restore is required for those wishing to migrate data from previous releases of PostgreSQL.

A.21.1. Migration from version 6.1 to version 6.2

This migration requires a complete dump of the 6.1 database and a restore of the database in 6.2.

Note that the **pg_dump** and **pg_dumpall** utility from 6.2 should be used to dump the 6.1 database.

A.21.2. Migration from version 1.*x* to version 6.2

Those migrating from earlier 1.* releases should first upgrade to 1.09 because the COPY output format was improved from the 1.02 release.

A.21.3. Changes

```
Bug Fixes
---------
Fix problems with pg_dump for inheritance, sequences, archive tables(Bruce)
Fix compile errors on overflow due to shifts, unsigned, and bad prototypes
  from Solaris(Diab Jerius)
Fix bugs in geometric line arithmetic (bad intersection calculations)(Thomas)
Check for geometric intersections at endpoints to avoid rounding ugliness(Thomas)
Catch non-functional delete attempts(Vadim)
Change time function names to be more consistent(Michael Reifenberg)
Check for zero divides(Michael Reifenberg)
Fix very old bug which made tuples changed/inserted by a commnd
 visible to the command itself (so we had multiple update of
 updated tuples, etc)(Vadim)
Fix for SELECT null, 'fail' FROM pg_am (Patrick)
SELECT NULL as EMPTY_FIELD now allowed(Patrick)
Remove un-needed signal stuff from contrib/pginterface
Fix OR (where x != 1 or x isnull didn't return tuples with x NULL) (Vadim)
Fix time_cmp function (Vadim)
Fix handling of functions with non-attribute first argument in
 WHERE clauses (Vadim)
Fix GROUP BY when order of entries is different from order
 in target list (Vadim)
Fix pg_dump for aggregates without sfunc1 (Vadim)

Enhancements
------------
Default genetic optimizer GEQO parameter is now 8(Bruce)
Allow use parameters in target list having aggregates in functions(Vadim)
Added JDBC driver as an interface(Adrian & Peter)
pg_password utility
Return number of tuples inserted/affected by INSERT/UPDATE/DELETE etc.(Vadim)
Triggers implemented with CREATE TRIGGER (SQL3)(Vadim)
SPI (Server Programming Interface) allows execution of queries inside
```

```
 C-functions (Vadim)
NOT NULL implemented (SQL92)(Robson Paniago de Miranda)
Include reserved words for string handling, outer joins, and unions(Thomas)
Implement extended comments ("/* ... */") using exclusive states(Thomas)
Add "//" single-line comments(Bruce)
Remove some restrictions on characters in operator names(Thomas)
DEFAULT and CONSTRAINT for tables implemented (SQL92)(Vadim & Thomas)
Add text concatenation operator and function (SQL92)(Thomas)
Support WITH TIME ZONE syntax (SQL92)(Thomas)
Support INTERVAL unit TO unit syntax (SQL92)(Thomas)
Define types DOUBLE PRECISION, INTERVAL, CHARACTER,
 and CHARACTER VARYING (SQL92)(Thomas)
Define type FLOAT(p) and rudimentary DECIMAL(p,s), NUMERIC(p,s) (SQL92)(Thomas)
Define EXTRACT(), POSITION(), SUBSTRING(), and TRIM() (SQL92)(Thomas)
Define CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP (SQL92)(Thomas)
Add syntax and warnings for UNION, HAVING, INNER and OUTER JOIN (SQL92)(Thomas)
Add more reserved words, mostly for SQL92 compliance(Thomas)
Allow hh:mm:ss time entry for timespan/reltime types(Thomas)
Add center() routines for lseg, path, polygon(Thomas)
Add distance() routines for circle-polygon, polygon-polygon(Thomas)
Check explicitly for points and polygons contained within polygons
 using an axis-crossing algorithm(Thomas)
Add routine to convert circle-box(Thomas)
Merge conflicting operators for different geometric data types(Thomas)
Replace distance operator "<===>" with "<->"(Thomas)
Replace "above" operator "!^" with ">^" and "below" operator "!|" with "<^"(Thomas)
Add routines for text trimming on both ends, substring, and string position(Thomas)
Added conversion routines circle(box) and poly(circle)(Thomas)
Allow internal sorts to be stored in memory rather than in files(Bruce & Vadim)
Allow functions and operators on internally-identical types to succeed(Bruce)
Speed up backend start-up after profiling analysis(Bruce)
Inline frequently called functions for performance(Bruce)
Reduce open() calls(Bruce)
psql:  Add PAGER for \h and \?,\C fix
Fix for psql pager when no tty(Bruce)
New entab utility(Bruce)
General trigger functions for referential integrity (Vadim)
General trigger functions for time travel (Vadim)
General trigger functions for AUTOINCREMENT/IDENTITY feature (Vadim)
MOVE implementation (Vadim)

Source Tree Changes
-------------------
HP-UX 10 patches (Vladimir Turin)
Added SCO support, (Daniel Harris)
MkLinux patches (Tatsuo Ishii)
Change geometric box terminology from "length" to "width"(Thomas)
Deprecate temporary unstored slope fields in geometric code(Thomas)
Remove restart instructions from INSTALL(Bruce)
Look in /usr/ucb first for install(Bruce)
Fix c++ copy example code(Thomas)
Add -o to psql manual page(Bruce)
Prevent relname unallocated string length from being copied into database(Bruce)
Cleanup for NAMEDATALEN use(Bruce)
Fix pg_proc names over 15 chars in output(Bruce)
Add strNcpy() function(Bruce)
remove some (void) casts that are unnecessary(Bruce)
```

```
new interfaces directory(Marc)
Replace fopen() calls with calls to fd.c functions(Bruce)
Make functions static where possible(Bruce)
enclose unused functions in #ifdef NOT_USED(Bruce)
Remove call to difftime() in timestamp support to fix SunOS(Bruce & Thomas)
Changes for Digital Unix
Portability fix for pg_dumpall(Bruce)
Rename pg_attribute.attnvals to attdispersion(Bruce)
"intro/unix" manual page now "pgintro"(Bruce)
"built-in" manual page now "pgbuiltin"(Bruce)
"drop" manual page now "drop_table"(Bruce)
Add "create_trigger", "drop_trigger" manual pages(Thomas)
Add constraints regression test(Vadim & Thomas)
Add comments syntax regression test(Thomas)
Add PGINDENT and support program(Bruce)
Massive commit to run PGINDENT on all *.c and *.h files(Bruce)
Files moved to /src/tools directory(Bruce)
SPI and Trigger programming guides (Vadim & D’Arcy)
```

A.22. Release 6.1.1

Release date: 1997-07-22

A.22.1. Migration from version 6.1 to version 6.1.1

This is a minor bug-fix release. A dump/reload is not required from version 6.1, but is required from any release prior to 6.1. Refer to the release notes for 6.1 for more details.

A.22.2. Changes

```
fix for SET with options (Thomas)
allow pg_dump/pg_dumpall to preserve ownership of all tables/objects(Bruce)
new psql \connect option allows changing usernames without changing databases
fix for initdb --debug option(Yoshihiko Ichikawa))
lextest cleanup(Bruce)
hash fixes(Vadim)
fix date/time month boundary arithmetic(Thomas)
fix timezone daylight handling for some ports(Thomas, Bruce, Tatsuo)
timestamp overhauled to use standard functions(Thomas)
other code cleanup in date/time routines(Thomas)
psql’s \d now case-insensitive(Bruce)
psql’s backslash commands can now have trailing semicolon(Bruce)
fix memory leak in psql when using \g(Bruce)
major fix for endian handling of communication to server(Thomas, Tatsuo)
Fix for Solaris assembler and include files(Yoshihiko Ichikawa)
allow underscores in usernames(Bruce)
pg_dumpall now returns proper status, portability fix(Bruce)
```

A.23. Release 6.1

Release date: 1997-06-08

The regression tests have been adapted and extensively modified for the 6.1 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 consistent across the data types. The polygon output in misc.out has only been spot-checked for correctness relative to the original regression output.

PostgreSQL 6.1 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).

A.23.1. Migration to version 6.1

This migration requires a complete dump of the 6.0 database and a restore of the database in 6.1.

Those migrating from earlier 1.* releases should first upgrade to 1.09 because the COPY output format was improved from the 1.02 release.

A.23.2. Changes

```
Bug Fixes
---------
packet length checking in library routines
lock manager priority patch
check for under/over flow of float8(Bruce)
multi-table join fix(Vadim)
SIGPIPE crash fix(Darren)
large object fixes(Sven)
allow btree indexes to handle NULLs(Vadim)
timezone fixes(D'Arcy)
```

```
select SUM(x) can return NULL on no rows(Thomas)
internal optimizer, executor bug fixes(Vadim)
fix problem where inner loop in < or <= has no rows(Vadim)
prevent re-commuting join index clauses(Vadim)
fix join clauses for multiple tables(Vadim)
fix hash, hashjoin for arrays(Vadim)
fix btree for abstime type(Vadim)
large object fixes(Raymond)
fix buffer leak in hash indexes (Vadim)
fix rtree for use in inner scan (Vadim)
fix gist for use in inner scan, cleanups (Vadim, Andrea)
avoid unnecessary local buffers allocation (Vadim, Massimo)
fix local buffers leak in transaction aborts (Vadim)
fix file manager memmory leaks, cleanups (Vadim, Massimo)
fix storage manager memmory leaks (Vadim)
fix btree duplicates handling (Vadim)
fix deleted tuples re-incarnation caused by vacuum (Vadim)
fix SELECT varchar()/char() INTO TABLE made zero-length fields(Bruce)
many psql, pg_dump, and libpq memory leaks fixed using Purify (Igor)

Enhancements
------------
attribute optimization statistics(Bruce)
much faster new btree bulk load code(Paul)
BTREE UNIQUE added to bulk load code(Vadim)
new lock debug code(Massimo)
massive changes to libpg++(Leo)
new GEQO optimizer speeds table multi-table optimization(Martin)
new WARN message for non-unique insert into unique key(Marc)
update x=-3, no spaces, now valid(Bruce)
remove case-sensitive identifier handling(Bruce,Thomas,Dan)
debug backend now pretty-prints tree(Darren)
new Oracle character functions(Edmund)
new plaintext password functions(Dan)
no such class or insufficient privilege changed to distinct messages(Dan)
new ANSI timestamp function(Dan)
new ANSI Time and Date types (Thomas)
move large chunks of data in backend(Martin)
multi-column btree indexes(Vadim)
new SET var TO value command(Martin)
update transaction status on reads(Dan)
new locale settings for character types(Oleg)
new SEQUENCE serial number generator(Vadim)
GROUP BY function now possible(Vadim)
re-organize regression test(Thomas,Marc)
new optimizer operation weights(Vadim)
new psql \z grant/permit option(Marc)
new MONEY data type(D'Arcy,Thomas)
tcp socket communication speed improved(Vadim)
new VACUUM option for attribute statistics, and for certain columns (Vadim)
many geometric type improvements(Thomas,Keith)
additional regression tests(Thomas)
new datestyle variable(Thomas,Vadim,Martin)
more comparison operators for sorting types(Thomas)
new conversion functions(Thomas)
new more compact btree format(Vadim)
allow pg_dumpall to preserve database ownership(Bruce)
```

```
new SET GEQO=# and R_PLANS variable(Vadim)
old (!GEQO) optimizer can use right-sided plans (Vadim)
typechecking improvement in SQL parser(Bruce)
new SET, SHOW, RESET commands(Thomas,Vadim)
new \connect database USER option
new destroydb -i option (Igor)
new \dt and \di psql commands (Darren)
SELECT "\n" now escapes newline (A. Duursma)
new geometry conversion functions from old format (Thomas)

Source tree changes
-------------------
new configuration script(Marc)
readline configuration option added(Marc)
OS-specific configuration options removed(Marc)
new OS-specific template files(Marc)
no more need to edit Makefile.global(Marc)
re-arrange include files(Marc)
nextstep patches (Gregor Hoffleit)
removed WIN32-specific code(Bruce)
removed postmaster -e option, now only postgres -e option (Bruce)
merge duplicate library code in front/backends(Martin)
now works with eBones, international Kerberos(Jun)
more shared library support
c++ include file cleanup(Bruce)
warn about buggy flex(Bruce)
DG/UX, Ultrix, IRIX, AIX portability fixes
```

A.24. Release 6.0

Release date: 1997-01-29

A dump/restore is required for those wishing to migrate data from previous releases of PostgreSQL.

A.24.1. Migration from version 1.09 to version 6.0

This migration requires a complete dump of the 1.09 database and a restore of the database in 6.0.

A.24.2. Migration from pre-1.09 to version 6.0

Those migrating from earlier 1.* releases should first upgrade to 1.09 because the COPY output format was improved from the 1.02 release.

A.24.3. Changes

```
Bug Fixes
---------
ALTER TABLE bug - running postgress process needs to re-read table definition
Allow vacuum to be run on one table or entire database(Bruce)
Array fixes
Fix array over-runs of memory writes(Kurt)
Fix elusive btree range/non-range bug(Dan)
Fix for hash indexes on some types like time and date
Fix for pg_log size explosion
Fix permissions on lo_export()(Bruce)
Fix unitialized reads of memory(Kurt)
Fixed ALTER TABLE ... char(3) bug(Bruce)
Fixed a few small memory leaks
Fixed EXPLAIN handling of options and changed full_path option name
Fixed output of group acl permissions
Memory leaks (hunt and destroy with tools like Purify(Kurt)
Minor improvements to rules system
NOTIFY fixes
New asserts for run-checking
Overhauled parser/analyze code to properly report errors and increase speed
Pg_dump -d now handles NULL's properly(Bruce)
Prevent SELECT NULL from crashing server (Bruce)
Properly report errors when INSERT ... SELECT columns did not match
Properly report errors when insert column names were not correct
Psql \g filename now works(Bruce)
Psql fixed problem with multiple statements on one line with multiple outputs
Removed duplicate system oid's
SELECT * INTO TABLE . GROUP/ORDER BY gives unlink error if table exists(Bruce)
Several fixes for queries that crashed the backend
Starting quote in insert string errors(Bruce)
Submitting an empty query now returns empty status, not just " " query(Bruce)

Enhancements
------------
Add EXPLAIN manual page(Bruce)
Add UNIQUE index capability(Dan)
Add hostname/user level access control rather than just hostname and user
Add synonym of != for <>(Bruce)
Allow "select oid,* from table"
Allow BY,ORDER BY to specify columns by number, or by non-alias table.column(Bruce)
Allow COPY from the frontend(Bryan)
Allow GROUP BY to use alias column name(Bruce)
Allow actual compression, not just reuse on the same page(Vadim)
Allow installation-configuration option to auto-add all local users(Bryan)
Allow libpq to distinguish between text value " and null(Bruce)
Allow non-postgres users with createdb privs to destroydb's
Allow restriction on who can create C functions(Bryan)
Allow restriction on who can do backend COPY(Bryan)
Can shrink tables, pg_time and pg_log(Vadim & Erich)
Change debug level 2 to print queries only, changed debug heading layout(Bruce)
Change default decimal constant representation from float4 to float8(Bruce)
European date format now set when postmaster is started
Execute lowercase function names if not found with exact case
Fixes for aggregate/GROUP processing, allow 'select sum(func(x),sum(x+y) from z'
Gist now included in the distrubution(Marc)
```

```
Idend authentication of local users(Bryan)
Implement BETWEEN qualifier(Bruce)
Implement IN qualifier(Bruce)
Libpq has PQgetisnull()(Bruce)
Libpq++ improvements
New options to initdb(Bryan)
Pg_dump allow dump of oid's(Bruce)
Pg_dump create indexes after tables are loaded for speed(Bruce)
Pg_dumpall dumps all databases, and the user table
Pginterface additions for NULL values(Bruce)
Prevent postmaster from being run as root
Psql \h and \? is now readable(Bruce)
Psql allow backslashed, semicolons anywhere on the line(Bruce)
Psql changed command prompt for lines in query or in quotes(Bruce)
Psql char(3) now displays as (bp)char in \d output(Bruce)
Psql return code now more accurate(Bryan?)
Psql updated help syntax(Bruce)
Re-visit and fix vacuum(Vadim)
Reduce size of regression diffs, remove timezone name difference(Bruce)
Remove compile-time parameters to enable binary distributions(Bryan)
Reverse meaning of HBA masks(Bryan)
Secure Authentication of local users(Bryan)
Speed up vacuum(Vadim)
Vacuum now had VERBOSE option(Bruce)

Source tree changes
-------------------
All functions now have prototypes that are compared against the calls
Allow asserts to be disabled easly from Makefile.global(Bruce)
Change oid constants used in code to #define names
Decoupled sparc and solaris defines(Kurt)
Gcc -Wall compiles cleanly with warnings only from unfixable constructs
Major include file reorganization/reduction(Marc)
Make now stops on compile failure(Bryan)
Makefile restructuring(Bryan, Marc)
Merge bsdi_2_1 to bsdi(Bruce)
Monitor program removed
Name change from Postgres95 to PostgreSQL
New config.h file(Marc, Bryan)
PG_VERSION now set to 6.0 and used by postmaster
Portability additions, including Ultrix, DG/UX, AIX, and Solaris
Reduced the number of #define's, centeralized #define's
Remove duplicate OIDS in system tables(Dan)
Remove duplicate system catalog info or report mismatches(Dan)
Removed many os-specific #define's
Restructured object file generation/location(Bryan, Marc)
Restructured port-specific file locations(Bryan, Marc)
Unused/uninialized variables corrected
```

A.25. Release 1.09

Release date: 1996-11-04

Sorry, we didn't keep track of changes from 1.02 to 1.09. Some of the changes listed in 6.0 were actually included in the 1.02.1 to 1.09 releases.

A.26. Release 1.02

Release date: 1996-08-01

A.26.1. Migration from version 1.02 to version 1.02.1

Here is a new migration file for 1.02.1. It includes the 'copy' change and a script to convert old ASCII files.

> **Note:** The following notes are for the benefit of users who want to migrate databases from Postgres95 1.01 and 1.02 to Postgres95 1.02.1.
>
> If you are starting afresh with Postgres95 1.02.1 and do not need to migrate old databases, you do not need to read any further.

In order to upgrade older Postgres95 version 1.01 or 1.02 databases to version 1.02.1, the following steps are required:

1. Start up a new 1.02.1 postmaster
2. Add the new built-in functions and operators of 1.02.1 to 1.01 or 1.02 databases. This is done by running the new 1.02.1 server against your own 1.01 or 1.02 database and applying the queries attached at the end of the file. This can be done easily through **psql**. If your 1.01 or 1.02 database is named `testdb` and you have cut the commands from the end of this file and saved them in `addfunc.sql`:

   ```
   % psql testdb -f addfunc.sql
   ```

   Those upgrading 1.02 databases will get a warning when executing the last two statements in the file because they are already present in 1.02. This is not a cause for concern.

A.26.2. Dump/Reload Procedure

If you are trying to reload a pg_dump or text-mode, `copy tablename to stdout` generated with a previous version, you will need to run the attached **sed** script on the ASCII file before loading it into the database. The old format used '.' as end-of-data, while '\.' is now the end-of-data marker. Also, empty strings are now loaded in as '' rather than NULL. See the copy manual page for full details.

```
sed 's/^\.$/\\./g' <in_file >out_file
```

If you are loading an older binary copy or non-stdout copy, there is no end-of-data character, and hence no conversion necessary.

```
-- following lines added by agc to reflect the case-insensitive
-- regexp searching for varchar (in 1.02), and bpchar (in 1.02.1)
create operator ~* (leftarg = bpchar, rightarg = text, procedure = texticregexeq);
create operator !~* (leftarg = bpchar, rightarg = text, procedure = texticregexne);
create operator ~* (leftarg = varchar, rightarg = text, procedure = texticregexeq);
create operator !~* (leftarg = varchar, rightarg = text, procedure = texti-
cregexne);
```

A.26.3. Changes

```
Source code maintenance and development
 * worldwide team of volunteers
 * the source tree now in CVS at ftp.ki.net

Enhancements
 * psql (and underlying libpq library) now has many more options for
   formatting output, including HTML
 * pg_dump now output the schema and/or the data, with many fixes to
   enhance completeness.
 * psql used in place of monitor in administration shell scripts.
   monitor to be depreciated in next release.
 * date/time functions enhanced
 * NULL insert/update/comparison fixed/enhanced
 * TCL/TK lib and shell fixed to work with both tck7.4/tk4.0 and tcl7.5/tk4.1

Bug Fixes (almost too numerous to mention)
 * indexes
 * storage management
 * check for NULL pointer before dereferencing
 * Makefile fixes

New Ports
 * added SolarisX86 port
 * added BSD/OS 2.1 port
 * added DG/UX port
```

A.27. Release 1.01

Release date: 1996-02-23

A.27.1. Migration from version 1.0 to version 1.01

The following notes are for the benefit of users who want to migrate databases from Postgres95 1.0 to Postgres95 1.01.

If you are starting afresh with Postgres95 1.01 and do not need to migrate old databases, you do not need to read any further.

In order to Postgres95 version 1.01 with databases created with Postgres95 version 1.0, the following steps are required:

1. Set the definition of `NAMEDATALEN` in `src/Makefile.global` to 16 and `OIDNAMELEN` to 20.
2. Decide whether you want to use Host based authentication.

    a. If you do, you must create a file name `pg_hba` in your top-level data directory (typically the value of your `$PGDATA`). `src/libpq/pg_hba` shows an example syntax.

    b. If you do not want host-based authentication, you can comment out the line

    ```
    HBA = 1
    ```

    in `src/Makefile.global`

    Note that host-based authentication is turned on by default, and if you do not take steps A or B above, the out-of-the-box 1.01 will not allow you to connect to 1.0 databases.

3. Compile and install 1.01, but DO NOT do the **initdb** step.
4. Before doing anything else, terminate your 1.0 postmaster, and backup your existing `$PGDATA` directory.
5. Set your `PGDATA` environment variable to your 1.0 databases, but set up path up so that 1.01 binaries are being used.
6. Modify the file `$PGDATA/PG_VERSION` from 5.0 to 5.1
7. Start up a new 1.01 postmaster
8. Add the new built-in functions and operators of 1.01 to 1.0 databases. This is done by running the new 1.01 server against your own 1.0 database and applying the queries attached and saving in the file 1.0_to_1.01.sql. This can be done easily through **psql**. If your 1.0 database is name `testdb`:

    ```
    % psql testdb -f 1.0_to_1.01.sql
    ```

    and then execute the following commands (cut and paste from here):

    ```
    -- add builtin functions that are new to 1.01

    create function int4eqoid (int4, oid) returns bool as 'foo'
    language 'internal';
    create function oideqint4 (oid, int4) returns bool as 'foo'
    language 'internal';
    create function char2icregexeq (char2, text) returns bool as 'foo'
    language 'internal';
    create function char2icregexne (char2, text) returns bool as 'foo'
    language 'internal';
    create function char4icregexeq (char4, text) returns bool as 'foo'
    language 'internal';
    create function char4icregexne (char4, text) returns bool as 'foo'
    language 'internal';
    create function char8icregexeq (char8, text) returns bool as 'foo'
    ```

```
   language 'internal';
   create function char8icregexne (char8, text) returns bool as 'foo'
   language 'internal';
   create function char16icregexeq (char16, text) returns bool as 'foo'
   language 'internal';
   create function char16icregexne (char16, text) returns bool as 'foo'
   language 'internal';
   create function texticregexeq (text, text) returns bool as 'foo'
   language 'internal';
   create function texticregexne (text, text) returns bool as 'foo'
   language 'internal';

   -- add builtin functions that are new to 1.01

   create operator = (leftarg = int4, rightarg = oid, procedure = int4eqoid);
   create operator = (leftarg = oid, rightarg = int4, procedure = oideqint4);
   create operator ~* (leftarg = char2, rightarg = text, procedure = char2icregexeq);
   create operator !~* (leftarg = char2, rightarg = text, procedure = char2icregexne);
   create operator ~* (leftarg = char4, rightarg = text, procedure = char4icregexeq);
   create operator !~* (leftarg = char4, rightarg = text, procedure = char4icregexne);
   create operator ~* (leftarg = char8, rightarg = text, procedure = char8icregexeq);
   create operator !~* (leftarg = char8, rightarg = text, procedure = char8icregexne);
   create operator ~* (leftarg = char16, rightarg = text, procedure = char16icregexeq);
   create operator !~* (leftarg = char16, rightarg = text, procedure = char16icregexne);
   create operator ~* (leftarg = text, rightarg = text, procedure = texticregexeq);
   create operator !~* (leftarg = text, rightarg = text, procedure = texti-
   cregexne);
```

A.27.2. Changes

```
Incompatibilities:
 * 1.01 is backwards compatible with 1.0 database provided the user
   follow the steps outlined in the MIGRATION_from_1.0_to_1.01 file.
   If those steps are not taken, 1.01 is not compatible with 1.0 database.

Enhancements:
 * added PQdisplayTuples() to libpq and changed monitor and psql to use it
 * added NeXT port (requires SysVIPC implementation)
 * added CAST .. AS ... syntax
 * added ASC and DESC keywords
 * added 'internal' as a possible language for CREATE FUNCTION
   internal functions are C functions which have been statically linked
   into the postgres backend.
 * a new type "name" has been added for system identifiers (table names,
   attribute names, etc.)  This replaces the old char16 type.   The
   of name is set by the NAMEDATALEN #define in src/Makefile.global
 * a readable reference manual that describes the query language.
 * added host-based access control.  A configuration file ($PGDATA/pg_hba)
   is used to hold the configuration data.  If host-based access control
   is not desired, comment out HBA=1 in src/Makefile.global.
 * changed regex handling to be uniform use of Henry Spencer's regex code
   regardless of platform.  The regex code is included in the distribution
 * added functions and operators for case-insensitive regular expressions.
```

```
   The operators are ~* and !~*.
 * pg_dump uses COPY instead of SELECT loop for better performance

Bug fixes:
 * fixed an optimizer bug that was causing core dumps when
   functions calls were used in comparisons in the WHERE clause
 * changed all uses of getuid to geteuid so that effective uids are used
 * psql now returns non-zero status on errors when using -c
 * applied public patches 1-14
```

A.28. Release 1.0

Release date: 1995-09-05

A.28.1. Changes

```
Copyright change:
 * The copyright of Postgres 1.0 has been loosened to be freely modifiable
   and modifiable for any purpose.  Please read the COPYRIGHT file.
   Thanks to Professor Michael Stonebraker for making this possible.

Incompatibilities:
 *  date formats have to be MM-DD-YYYY (or DD-MM-YYYY if you're using
   EUROPEAN STYLE).  This follows SQL-92 specs.
 *  "delimiters" is now a keyword

Enhancements:
 *  sql LIKE syntax has been added
 *  copy command now takes an optional USING DELIMITER specification.
   delimiters can be any single-character string.
 *  IRIX 5.3 port has been added.
   Thanks to Paul Walmsley and others.
 *  updated pg_dump to work with new libpq
 *  \d has been added psql
   Thanks to Keith Parks
 *  regexp performance for architectures that use POSIX regex has been
   improved due to caching of precompiled patterns.
   Thanks to Alistair Crooks
 *  a new version of libpq++
   Thanks to William Wanders

Bug fixes:
 *  arbitrary userids can be specified in the createuser script
 *  \c to connect to other databases in psql now works.
 *  bad pg_proc entry for float4inc() is fixed
 *  users with usecreatedb field set can now create databases without
   having to be usesuper
 *  remove access control entries when the entry no longer has any
   permissions
```

```
 *  fixed non-portable datetimes implementation
 *  added kerberos flags to the src/backend/Makefile
 *  libpq now works with kerberos
 *  typographic errors in the user manual have been corrected.
 *  btrees with multiple index never worked, now we tell you they don't
   work when you try to use them
```

A.29. Postgres95 Release 0.03

Release date: 1995-07-21

A.29.1. Changes

```
Incompatible changes:
 * BETA-0.3 IS INCOMPATIBLE WITH DATABASES CREATED WITH PREVIOUS VERSIONS
   (due to system catalog changes and indexing structure changes).
 * double-quote (") is deprecated as a quoting character for string literals;
   you need to convert them to single quotes (').
 * name of aggregates (eg. int4sum) are renamed in accordance with the
   SQL standard (eg. sum).
 * CHANGE ACL syntax is replaced by GRANT/REVOKE syntax.
 * float literals (eg. 3.14) are now of type float4 (instead of float8 in
   previous releases); you might have to do typecasting if you depend on it
   being of type float8.  If you neglect to do the typecasting and you assign
   a float literal to a field of type float8, you may get incorrect values
   stored!
 * LIBPQ has been totally revamped so that frontend applications
   can connect to multiple backends
 * the usesysid field in pg_user has been changed from int2 to int4 to
   allow wider range of Unix user ids.
 * the netbsd/freebsd/bsd o/s ports have been consolidated into a
   single BSD44_derived port.  (thanks to Alistair Crooks)

SQL standard-compliance (the following details changes that makes postgres95
more compliant to the SQL-92 standard):
 * the following SQL types are now built-in: smallint, int(eger), float, real,
   char(N), varchar(N), date and time.

   The following are aliases to existing postgres types:
  smallint -> int2
  integer, int -> int4
  float, real  -> float4
   char(N) and varchar(N) are implemented as truncated text types. In
   addition, char(N) does blank-padding.
 * single-quote (') is used for quoting string literals; " (in addition to
   \') is supported as means of inserting a single quote in a string
 * SQL standard aggregate names (MAX, MIN, AVG, SUM, COUNT) are used
   (Also, aggregates can now be overloaded, i.e. you can define your
   own MAX aggregate to take in a user-defined type.)
```

```
 * CHANGE ACL removed. GRANT/REVOKE syntax added.
   - Privileges can be given to a group using the "GROUP" keyword.
 For example:
  GRANT SELECT ON foobar TO GROUP my_group;
 The keyword 'PUBLIC' is also supported to mean all users.

 Privileges can only be granted or revoked to one user or group
 at a time.

 "WITH GRANT OPTION" is not supported.  Only class owners can change
 access control
   - The default access control is to to grant users readonly access.
     You must explicitly grant insert/update access to users.  To change
     this, modify the line in
  src/backend/utils/acl.h
     that defines ACL_WORLD_DEFAULT

Bug fixes:
 * the bug where aggregates of empty tables were not run has been fixed. Now,
   aggregates run on empty tables will return the initial conditions of the
   aggregates. Thus, COUNT of an empty table will now properly return 0.
   MAX/MIN of an empty table will return a tuple of value NULL.
 * allow the use of \; inside the monitor
 * the LISTEN/NOTIFY asynchronous notification mechanism now work
 * NOTIFY in rule action bodies now work
 * hash indexes work, and access methods in general should perform better.
   creation of large btree indexes should be much faster.  (thanks to Paul
   Aoki)

Other changes and enhancements:
 * addition of an EXPLAIN statement used for explaining the query execution
   plan (eg. "EXPLAIN SELECT * FROM EMP" prints out the execution plan for
   the query).
 * WARN and NOTICE messages no longer have timestamps on them. To turn on
   timestamps of error messages, uncomment the line in
   src/backend/utils/elog.h:
 /* define ELOG_TIMESTAMPS */
 * On an access control violation, the message
 "Either no such class or insufficient privilege"
   will be given.  This is the same message that is returned when
   a class is not found.  This dissuades non-privileged users from
   guessing the existence of privileged classes.
 * some additional system catalog changes have been made that are not
   visible to the user.

libpgtcl changes:
 * The -oid option has been added to the "pg_result" tcl command.
   pg_result -oid returns oid of the last tuple inserted.   If the
   last command was not an INSERT, then pg_result -oid returns "".
 * the large object interface is available as pg_lo* tcl commands:
   pg_lo_open, pg_lo_close, pg_lo_creat, etc.

Portability enhancements and New Ports:
 * flex/lex problems have been cleared up.  Now, you should be able to use
   flex instead of lex on any platforms.  We no longer make assumptions of
   what lexer you use based on the platform you use.
 * The Linux-ELF port is now supported.  Various configuration have been
```

```
   tested:  The following configuration is known to work:
 kernel 1.2.10, gcc 2.6.3, libc 4.7.2, flex 2.5.2, bison 1.24
   with everything in ELF format,

New utilities:
 * ipcclean added to the distribution
   ipcclean usually does not need to be run, but if your backend crashes
   and leaves shared memory segments hanging around, ipcclean will
   clean them up for you.

New documentation:
 * the user manual has been revised and libpq documentation added.
```

A.30. Postgres95 Release 0.02

Release date: 1995-05-25

A.30.1. Changes

```
Incompatible changes:
 * The SQL statement for creating a database is 'CREATE DATABASE' instead
   of 'CREATEDB'. Similarly, dropping a database is 'DROP DATABASE' instead
   of 'DESTROYDB'. However, the names of the executables 'createdb' and
   'destroydb' remain the same.

New tools:
 * pgperl - a Perl (4.036) interface to Postgres95
 * pg_dump - a utility for dumping out a postgres database into a
 script file containing query commands. The script files are in a ASCII
 format and can be used to reconstruct the database, even on other
 machines and other architectures. (Also good for converting
 a Postgres 4.2 database to Postgres95 database.)

The following ports have been incorporated into postgres95-beta-0.02:
 * the NetBSD port by Alistair Crooks
 * the AIX port by Mike Tung
 * the Windows NT port by Jon Forrest (more stuff but not done yet)
 * the Linux ELF port by Brian Gallew

The following bugs have been fixed in postgres95-beta-0.02:
 * new lines not escaped in COPY OUT and problem with COPY OUT when first
   attribute is a '.'
 * cannot type return to use the default user id in createuser
 * SELECT DISTINCT on big tables crashes
 * Linux installation problems
 * monitor doesn't allow use of 'localhost' as PGHOST
 * psql core dumps when doing \c or \l
 * the "pgtclsh" target missing from src/bin/pgtclsh/Makefile
 * libpgtcl has a hard-wired default port number
```

```
* SELECT DISTINCT INTO TABLE hangs
* CREATE TYPE doesn't accept 'variable' as the internallength
* wrong result using more than 1 aggregate in a SELECT
```

A.31. Postgres95 Release 0.01

Release date: 1995-05-01

Initial release.

Bibliography

Selected references and readings for SQL and PostgreSQL.

Some white papers and technical reports from the original POSTGRES development team are available at the University of California, Berkeley, Computer Science Department web site[1]

SQL Reference Books

Judith Bowman, Sandra Emerson, and Marcy Darnovsky, *The Practical SQL Handbook: Using Structured Query Language*, Third Edition, Addison-Wesley, ISBN 0-201-44787-8, 1996.

C. J. Date and Hugh Darwen, *A Guide to the SQL Standard: A user's guide to the standard database language SQL*, Fourth Edition, Addison-Wesley, ISBN 0-201-96426-0, 1997.

C. J. Date, *An Introduction to Database Systems*, Volume 1, Sixth Edition, Addison-Wesley, 1994.

Ramez Elmasri and Shamkant Navathe, *Fundamentals of Database Systems*, 3rd Edition, Addison-Wesley, ISBN 0-805-31755-4, August 1999.

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

Jeffrey D. Ullman, *Principles of Database and Knowledge: Base Systems*, Volume 1, Computer Science Press, 1988.

PostgreSQL-Specific Documentation

Stefan Simkovics, *Enhancement of the ANSI SQL Implementation of PostgreSQL*, Department of Information Systems, Vienna University of Technology, November 29, 1998.

Discusses SQL history and syntax, and describes the addition of `INTERSECT` and `EXCEPT` constructs into PostgreSQL. Prepared as a Master's Thesis with the support of O. Univ. Prof. Dr. Georg Gottlob and Univ. Ass. Mag. Katrin Seyr at Vienna University of Technology.

A. Yu and J. Chen, The POSTGRES Group, *The Postgres95 User Manual*, University of California, Sept. 5, 1995.

Zelaine Fong, *The design and implementation of the POSTGRES query optimizer*[2], University of California, Berkeley, Computer Science Department.

1. http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/papers/
2. http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/papers/UCB-MS-zfong.pdf

Proceedings and Articles

Nels Olson, *Partial indexing in POSTGRES: research project*, University of California, UCB Engin T7.49.1993 O676, 1993.

L. Ong and J. Goh, "A Unified Framework for Version Modeling Using Production Rules in a Database System", *ERL Technical Memorandum M90/33*, University of California, April, 1990.

L. Rowe and M. Stonebraker, "The POSTGRES data model[3]", Proc. VLDB Conference, Sept. 1987.

P. Seshadri and A. Swami, "Generalized Partial Indexes[4] ", Proc. Eleventh International Conference on Data Engineering, 6-10 March 1995, IEEE Computer Society Press, Cat. No.95CH35724, 1995, p. 420-7.

M. Stonebraker and L. Rowe, "The design of POSTGRES[5]", Proc. ACM-SIGMOD Conference on Management of Data, May 1986.

M. Stonebraker, E. Hanson, and C. H. Hong, "The design of the POSTGRES rules system", Proc. IEEE Conference on Data Engineering, Feb. 1987.

M. Stonebraker, "The design of the POSTGRES storage system[6]", Proc. VLDB Conference, Sept. 1987.

M. Stonebraker, M. Hearst, and S. Potamianos, "A commentary on the POSTGRES rules system[7]", *SIGMOD Record 18(3)*, Sept. 1989.

M. Stonebraker, "The case for partial indexes[8]", *SIGMOD Record 18(4)*, Dec. 1989, p. 4-11.

M. Stonebraker, L. A. Rowe, and M. Hirohama, "The implementation of POSTGRES[9]", *Transactions on Knowledge and Data Engineering 2(1)*, IEEE, March 1990.

M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos, "On Rules, Procedures, Caching and Views in Database Systems[10]", Proc. ACM-SIGMOD Conference on Management of Data, June 1990.

3. http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/papers/ERL-M87-13.pdf
4. http://simon.cs.cornell.edu/home/praveen/papers/partindex.de95.ps.Z
5. http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/papers/ERL-M85-95.pdf
6. http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/papers/ERL-M87-06.pdf
7. http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/papers/ERL-M89-82.pdf
8. http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/papers/ERL-M89-17.pdf
9. http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/papers/ERL-M90-34.pdf
10. http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/papers/ERL-M90-36.pdf

Index

A

Australian time zones, 26

B

backup, 70
bison, 1
BSD/OS, 33

C

client authentication, 39
configuration
  server, 21
configure, 3

D

data area
  (See database cluster)
database, 58
database cluster, 17
deadlock
  timeout, 26
disk space, 65
dynamic loading, 26
dynamic_library_path, 26

F

flex, 1
FreeBSD, 19, 34
fsync, 27

G

genetic query optimization, 23
GEQO
  (See genetic query optimization)

H

HP-UX, 34

I

ident, 45
index scan, 22
inheritance, 28
initlocation, 60
installation, 1
  on Windows, 1, 16
IS NULL, 28

K

Kerberos, 44

L

LC_COLLATE, 18
ldconfig, 10
Linux, 19, 34
locale, 47
log files, 68

M

make, 1
MANPATH, 10
  (See Also man pages)
MD5, 43
multibyte, 49

N

NetBSD, 19, 34

O

OpenBSD, 19, 34
OpenSSL, 7
  (See Also SSL)
ORDER BY, 48

P

password, 43
PATH, 10
PGDATA, 17
pg_ctl, 18
pg_dumpall, 2
pg_hba.conf, 39
pg_ident.conf, 45
port, 28
postgres user, 17
postmaster, 18
ps
  to monitor activity, 75

R

readline, 1
regression test, 8

S

SCO OpenServer, 34
semaphores, 31
sequential scan, 22
shared libraries, 9
shared memory, 31
SHMMAX, 32
SIGHUP, 21, 42, 45
Solaris, 19, 35
ssh, 37
SSL, 28, 37
statistics, 75

T

TCP/IP, 18
timeout
  authentication, 26
  deadlock, 26
transaction ID
  wraparound, 67
transaction isolation level, 26

U

Unicode, 53
UnixWare, 35
upgrading, 2, 73

V

vacuum, 65

Y

yacc, 1