

What is PostgreSQL?

PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux and Windows. It includes most SQL92 and SQL99 data types. It also supports storage of binary large objects, including pictures, sounds, or video. It has native programming interfaces and exceptional documentation.



PostgreSQL
www.postgresql.org



PostgreSQL and the PostgreSQL logo are trademarks of The PostgreSQL Global Development Group

About PostgreSQL

An enterprise class database, PostgreSQL boasts sophisticated features such as Multi-Version Concurrency Control (MVCC), point in time recovery, tablespaces, asynchronous replication, nested transactions (savepoints), online/hot backups, a sophisticated query planner/optimizer, and write ahead logging for fault tolerance. It supports international character sets, multibyte character encodings, Unicode, and it is locale-aware for sorting, case-sensitivity, and formatting. It is highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate. There are active PostgreSQL systems in production environments that manage in excess of 4 terabytes of data.

Fultus™



Published by Fultus Corporation
www.fultus.com



PostgreSQL
www.postgresql.org



PostgreSQL 8.4

Internals and Appendixes

Volume V



By The PostgreSQL Global Development Group

PostgreSQL 8.4



Internals and Appendixes

5



Linbrary™ - Linux Documentation Library
www.linbrary.com



PostgreSQL 8.4
Official Documentation

Internals and Appendixes

Volume V



Fultus™ Books

PostgreSQL



PostgreSQL 8.4 Official Documentation

Internals and Appendixes

Volume V

ISBN 1-59682-162-0

Copyright © 1996-2009 The PostgreSQL Global Development Group

Cover design and book layout by Fultus Corporation



Published by Fultus Corporation

Publisher Web: *www.fultus.com*

Linbrary - Linux Library: *www.linbrary.com*

Online Bookstore: *store.fultus.com*

email: *production@fultus.com*



This material may only be distributed subject to the terms and conditions set forth in the BSD License (presently available at <http://www.postgresql.org/about/licence>).

PostgreSQL and PostgreSQL logo are trademarks or registered trademarks of The PostgreSQL Global Development Group, in the U.S. and other countries. All product names and services identified throughout this manual are trademarks or registered trademarks of their respective companies.

The author and publisher have made every effort in the preparation of this book to ensure the accuracy of the information. However, the information contained in this book is offered without warranty, either express or implied. Neither the author nor the publisher nor any dealer or distributor will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Table of Contents

List of Tables.....	15
License.....	19
Abstract.....	20
Part VII Internals.....	21
Chapter 43. Overview of PostgreSQL Internals	22
43.1. The Path of a Query.....	22
43.2. How Connections are Established.....	23
43.3. The Parser Stage.....	23
43.3.1. Parser.....	24
43.3.2. Transformation Process.....	24
43.4. The PostgreSQL Rule System.....	25
43.5. Planner/Optimizer.....	25
43.5.1. Generating Possible Plans.....	26
43.6. Executor.....	27
Chapter 44. System Catalogs	29
44.1. Overview.....	29
44.2. pg_aggregate.....	31
44.3. pg_am.....	31
44.4. pg_amop.....	32
44.5. pg_amproc.....	33
44.6. pg_attrdef.....	33
44.7. pg_attribute.....	34
44.8. pg_authid.....	35
44.9. pg_auth_members.....	36
44.10. pg_cast.....	37
44.11. pg_class.....	38
44.12. pg_constraint.....	40
44.13. pg_conversion.....	41
44.14. pg_database.....	41
44.15. pg_depend.....	42
44.16. pg_description.....	44
44.17. pg_enum.....	44

Volume V

44.18. pg_foreign_data_wrapper	45
44.19. pg_foreign_server.....	45
44.20. pg_index	46
44.21. pg_inherits	47
44.22. pg_language.....	47
44.23. pg_largeobject.....	48
44.24. pg_listener.....	48
44.25. pg_namespace.....	49
44.26. pg_opclass.....	49
44.27. pg_operator.....	50
44.28. pg_opfamily.....	50
44.29. pg_pltemplate	51
44.30. pg_proc.....	52
44.31. pg_rewrite.....	54
44.32. pg_shdepend.....	54
44.33. pg_shdescription.....	55
44.34. pg_statistic.....	56
44.35. pg_tablespace	57
44.36. pg_trigger.....	58
44.37. pg_ts_config.....	58
44.38. pg_ts_config_map.....	59
44.39. pg_ts_dict.....	59
44.40. pg_ts_parser.....	60
44.41. pg_ts_template	60
44.42. pg_type.....	61
44.43. pg_user_mapping.....	64
44.44. System Views.....	65
44.45. pg_cursors.....	66
44.46. pg_group	67
44.47. pg_indexes.....	67
44.48. pg_locks	67
44.49. pg_prepared_statements.....	70
44.50. pg_prepared_xacts.....	70
44.51. pg_roles	71
44.52. pg_rules	72
44.53. pg_settings.....	72
44.54. pg_shadow.....	73
44.55. pg_stats	74
44.56. pg_tables.....	75

44.57. pg_timezone_abbrevs	75
44.58. pg_timezone_names	76
44.59. pg_user	76
44.60. pg_user_mappings	77
44.61. pg_views.....	77
Chapter 45. Frontend/Backend Protocol.....	78
45.1. Overview.....	78
45.1.1. Messaging Overview	79
45.1.2. Extended Query Overview	79
45.1.3. Formats and Format Codes.....	80
45.2. Message Flow	80
45.2.1. Start-Up.....	81
45.2.2. Simple Query	83
45.2.3. Extended Query.....	85
45.2.4. Function Call.....	89
45.2.5. COPY Operations	90
45.2.6. Asynchronous Operations	91
45.2.7. Cancelling Requests in Progress	92
45.2.8. Termination.....	93
45.2.9. SSL Session Encryption	93
45.3. Message Data Types	94
45.4. Message Formats.....	95
45.5. Error and Notice Message Fields.....	111
45.6. Summary of Changes since Protocol 2.0.....	113
Chapter 46. PostgreSQL Coding Conventions.....	115
46.1. Formatting.....	115
46.2. Reporting Errors Within the Server	116
46.3. Error Message Style Guide	118
46.3.1. What goes where	119
46.3.2. Formatting	119
46.3.3. Quotation marks.....	119
46.3.4. Use of quotes.....	120
46.3.5. Grammar and punctuation	120
46.3.6. Upper case vs. lower case.....	120
46.3.7. Avoid passive voice	121
46.3.8. Present vs past tense	121
46.3.9. Type of the object.....	121
46.3.10. Brackets	121
46.3.11. Assembling error messages.....	122

Volume V

46.3.12. Reasons for errors.....	122
46.3.13. Function names.....	122
46.3.14. Tricky words to avoid.....	122
46.3.15. Proper spelling.....	123
46.3.16. Localization.....	123
Chapter 47. Native Language Support.....	124
47.1. For the Translator.....	124
47.1.1. Requirements.....	124
47.1.2. Concepts.....	124
47.1.3. Creating and maintaining message catalogs.....	126
47.1.4. Editing the PO files.....	126
47.2. For the Programmer.....	127
47.2.1. Mechanics.....	127
47.2.2. Message-writing guidelines.....	129
Chapter 48. Writing A Procedural Language Handler.....	131
Chapter 49. Genetic Query Optimizer.....	134
49.1. Query Handling as a Complex Optimization Problem.....	134
49.2. Genetic Algorithms.....	134
49.3. Genetic Query Optimization (GEQO) in PostgreSQL.....	135
49.3.1. Generating Possible Plans with GEQO.....	136
49.3.2. Future Implementation Tasks for PostgreSQL GEQO.....	137
49.4. Further Reading.....	137
Chapter 50. Index Access Method Interface Definition.....	138
50.1. Catalog Entries for Indexes.....	138
50.2. Index Access Method Functions.....	140
50.3. Index Scanning.....	144
50.4. Index Locking Considerations.....	145
50.5. Index Uniqueness Checks.....	147
50.6. Index Cost Estimation Functions.....	148
Chapter 51. GiST Indexes.....	151
51.1. Introduction.....	151
51.2. Extensibility.....	151
51.3. Implementation.....	152
51.4. Examples.....	159
51.5. Crash Recovery.....	159
Chapter 52. GIN Indexes.....	161
52.1. Introduction.....	161
52.2. Extensibility.....	161
52.3. Implementation.....	163

52.3.1. GIN fast update technique	163
52.3.2. Partial match algorithm	164
52.4. GIN tips and tricks	164
52.5. Limitations	165
52.6. Examples	166
Chapter 53. Database Physical Storage	167
53.1. Database File Layout	167
53.2. TOAST	169
53.3. Free Space Map	171
53.4. Visibility Map	171
53.5. Database Page Layout	172
Chapter 54. BKI Backend Interface	176
54.1. BKI File Format	176
54.2. BKI Commands	176
54.3. Structure of the Bootstrap BKI File	178
54.4. Example	178
Chapter 55. How the Planner Uses Statistics	179
55.1. Row Estimation Examples	179
Part VIII Appendixes	186
Appendix A. PostgreSQL Error Codes	187
Appendix B. Date/Time Support	196
B.1. Date/Time Input Interpretation	196
B.2. Date/Time Key Words	197
B.3. Date/Time Configuration Files	198
B.4. History of Units	200
Appendix C. SQL Key Words	202
Appendix D. SQL Conformance	231
D.1. Supported Features	232
D.2. Unsupported Features	243
Appendix E. Release Notes	254
E.1. Release 8.4	254
E.1.1. Overview	254
E.1.2. Migration to Version 8.4	255
E.1.2.1. General	255
E.1.2.2. Server Settings	255
E.1.2.3. Queries	256
E.1.2.4. Functions and Operators	257
E.1.3. Changes	258

Volume V

E.1.3.1. Performance.....	258
E.1.3.2. Server.....	259
E.1.3.3. Queries.....	262
E.1.3.4. Object Manipulation.....	264
E.1.3.5. Utility Operations.....	265
E.1.3.6. Data Types.....	267
E.1.3.7. Functions.....	269
E.1.3.8. Client Applications.....	271
E.1.3.9. Programming Tools.....	274
E.1.3.10. Build Options.....	275
E.1.3.11. Source Code.....	276
E.1.3.12. Contrib.....	277
Appendix F. Additional Supplied Modules.....	280
F.1. adminpack.....	281
F.1.1. Functions implemented.....	281
F.2. auto_explain.....	281
F.2.1. Configuration parameters.....	282
F.2.2. Example.....	283
F.2.3. Author.....	283
F.3. btree_gin.....	283
F.3.1. Example usage.....	283
F.3.2. Authors.....	284
F.4. btree_gist.....	284
F.4.1. Example usage.....	284
F.4.2. Authors.....	284
F.5. chkpass.....	284
F.5.1. Author.....	285
F.6. citext.....	285
F.6.1. Rationale.....	286
F.6.2. How to Use It.....	286
F.6.3. String Comparison Behavior.....	286
F.6.4. Limitations.....	287
F.6.5. Author.....	288
F.7. cube.....	288
F.7.1. Syntax.....	288
F.7.2. Precision.....	288
F.7.3. Usage.....	288
F.7.4. Defaults.....	290
F.7.5. Notes.....	291

F.7.6. Credits	291
F.8. dblink	291
dblink_connect	291
dblink_connect_u	294
dblink_disconnect	294
dblink	295
dblink_exec	298
dblink_open	300
dblink_fetch	301
dblink_close	303
dblink_get_connections.....	304
dblink_error_message	305
dblink_send_query	305
dblink_is_busy.....	306
dblink_get_result.....	307
dblink_cancel_query.....	309
dblink_get_pkey.....	309
dblink_build_sql_insert.....	310
dblink_build_sql_delete	311
dblink_build_sql_update	312
F.9. dict_int	314
F.9.1. Configuration	314
F.9.2. Usage	314
F.10. dict_xsyn	314
F.10.1. Configuration.....	315
F.10.2. Usage.....	315
F.11. earthdistance.....	315
F.11.1. Cube-based earth distances.....	316
F.11.2. Point-based earth distances.....	317
F.12. fuzzystmatch.....	317
F.12.1. Soundex	317
F.12.2. Levenshtein	318
F.12.3. Metaphone.....	319
F.12.4. Double Metaphone.....	319
F.13. hstore	319
F.13.1. hstore External Representation	320
F.13.2. hstore Operators and Functions.....	320
F.13.3. Indexes	321
F.13.4. Examples.....	321

Volume V

F.13.5. Statistics	321
F.13.6. Authors	322
F.14. intagg	322
F.14.1. Functions	322
F.14.2. Sample Uses	323
F.15. intarray	324
F.15.1. intarray Functions and Operators	324
F.15.2. Index Support	325
F.15.3. Example	326
F.15.4. Benchmark	326
F.15.5. Authors	326
F.16. isn	326
F.16.1. Data types	327
F.16.2. Casts	327
F.16.3. Functions and Operators	328
F.16.4. Examples	329
F.16.5. Bibliography	329
F.16.6. Author	330
F.17. lo	330
F.17.1. Rationale	330
F.17.2. How to Use It	331
F.17.3. Limitations	331
F.17.4. Author	331
F.18. ltree	331
F.18.1. Definitions	331
F.18.2. Operators and Functions	333
F.18.3. Indexes	335
F.18.4. Example	335
F.18.5. Authors	338
F.19. oid2name	338
F.19.1. Overview	338
F.19.2. Examples	339
F.19.3. Limitations	341
F.19.4. Author	341
F.20. pageinspect	341
F.20.1. Functions	341
F.21. pgbench	343
F.21.1. Overview	344
F.21.2. What is the "transaction" actually performed in pgbench?	346

F.21.3. Custom Scripts	346
F.21.4. Per-transaction logging.....	348
F.21.5. Good Practices	348
F.22. pg_buffercache	349
F.22.1. The pg_buffercache view	349
F.22.2. Sample output.....	350
F.22.3. Authors	350
F.23. pgcrypto	350
F.23.1. General hashing functions	351
F.23.1.1. digest ()	351
F.23.1.2. hmac ()	351
F.23.2. Password hashing functions	351
F.23.2.1. crypt ()	352
F.23.2.2. gen_salt ()	352
F.23.3. PGP encryption functions	354
F.23.3.1. pgp_sym_encrypt ()	354
F.23.3.2. pgp_sym_decrypt ()	355
F.23.3.3. pgp_pub_encrypt ()	355
F.23.3.4. pgp_pub_decrypt ()	355
F.23.3.5. pgp_key_id ()	355
F.23.3.6. armor (), dearmor ()	356
F.23.3.7. Options for PGP functions.....	356
F.23.3.8. Generating PGP keys with GnuPG	358
F.23.3.9. Limitations of PGP code	359
F.23.4. Raw encryption functions	359
F.23.5. Random-data functions	360
F.23.6. Notes	360
F.23.6.1. Configuration	360
F.23.6.2. NULL handling.....	361
F.23.6.3. Security limitations.....	361
F.23.6.4. Useful reading	361
F.23.6.5. Technical references.....	361
F.23.7. Author	362
F.24. pg_freespacemap	362
F.24.1. Functions	363
F.24.2. Sample output.....	363
F.24.3. Author	364
F.25. pgrowlocks.....	364
F.25.1. Overview	364

Volume V

F.25.2. Sample output	365
F.25.3. Author.....	365
F.26. pg_standby	365
F.26.1. Usage.....	365
F.26.2. Examples	367
F.26.3. Supported server versions	368
F.26.4. Author.....	368
F.27. pg_stat_statements	368
F.27.1. The pg_stat_statements view	368
F.27.2. Functions	369
F.27.3. Configuration parameters.....	369
F.27.4. Sample output	370
F.27.5. Author.....	371
F.28. pgstattuple	371
F.28.1. Functions	371
F.28.2. Authors	373
F.29. pg_trgm.....	373
F.29.1. Trigram (or Trigraph) Concepts.....	373
F.29.2. Functions and Operators.....	373
F.29.3. Index Support	374
F.29.4. Text Search Integration.....	375
F.29.5. References.....	375
F.29.6. Authors	375
F.30. seg	375
F.30.1. Rationale.....	376
F.30.2. Syntax.....	376
F.30.3. Precision	377
F.30.4. Usage.....	377
F.30.5. Notes	378
F.30.6. Credits.....	379
F.31. spi.....	379
F.31.1. refint.c - functions for implementing referential integrity	379
F.31.2. timetravel.c - functions for implementing time travel.....	380
F.31.3. autoinc.c - functions for autoincrementing fields.....	381
F.31.4. insert_username.c - functions for tracking who changed a table.....	381
F.31.5. moddatettime.c - functions for tracking last modification time	381
F.32. sslinfo.....	382
F.32.1. Functions Provided.....	382
F.32.2. Author.....	384

F.33. tablefunc	384
F.33.1. Functions Provided	384
F.33.1.1. normal_rand	384
F.33.1.2. crosstab(text)	385
F.33.1.3. crosstabN(text)	387
F.33.1.4. crosstab(text, text)	388
F.33.1.5. connectby	391
F.33.2. Author	394
F.34. test_parser	394
F.34.1. Usage	394
F.35. tsearch2	395
F.35.1. Portability Issues	395
F.35.2. Converting a pre-8.3 Installation	396
F.36. uuid-osp	397
F.36.1. uuid-osp Functions	397
F.36.2. Author	398
F.37. vacuumlo	398
F.37.1. Usage	399
F.37.2. Method	399
F.37.3. Author	400
F.38. xml2	400
F.38.1. Deprecation notice	400
F.38.2. Description of functions	400
F.38.3. xpath_table	401
F.38.3.1. Multivalued results	402
F.38.4. XSLT functions	403
F.38.4.1. xslt_process	403
F.38.5. Author	404
Appendix G. External Projects	405
G.1. Client Interfaces	405
G.2. Procedural Languages	406
G.3. Extensions	406
Appendix H. The CVS Repository	408
H.1. Getting The Source Via Anonymous CVS	408
H.2. CVS Tree Organization	409
H.3. Getting The Source Via rsync	411
H.4. Getting The Source Via CVSup	411
H.4.1. Preparing A CVSup Client System	411
H.4.2. Running a CVSup Client	412

Volume V

Appendix I. Documentation	415
I.1. DocBook	415
I.2. Tool Sets	416
I.2.1. Linux RPM Installation.....	417
I.2.2. FreeBSD Installation.....	417
I.2.3. Debian Packages.....	418
I.2.4. Manual Installation from Source.....	418
I.2.4.1. Installing OpenJade.....	418
I.2.4.2. Installing the DocBook DTD Kit.....	419
I.2.4.3. Installing the DocBook DSSSL Style Sheets	419
I.2.4.4. Installing JadeTeX.....	420
I.2.5. Detection by <code>configure</code>	420
I.3. Building The Documentation.....	421
I.3.1. HTML.....	421
I.3.2. Manpages.....	421
I.3.3. Print Output via JadeTeX.....	422
I.3.4. Print Output via RTF	422
I.3.5. Plain Text Files.....	424
I.3.6. Syntax Check.....	424
I.4. Documentation Authoring	425
I.4.1. Emacs/PSGML	425
I.4.2. Other Emacs modes	426
I.5. Style Guide.....	426
I.5.1. Reference Pages	426
Appendix J. Acronyms.....	429
Bibliography	437
List of Volumes	439
Index.....	442

List of Tables

Table 44-1. System Catalogs	30
Table 44-2. pg_aggregate Columns.....	31
Table 44-3. pg_am Columns	32
Table 44-4. pg_amop Columns	33
Table 44-5. pg_amproc Columns.....	33
Table 44-6. pg_attrdef Columns	34
Table 44-7. pg_attribute Columns.....	35
Table 44-8. pg_authid Columns.....	36
Table 44-9. pg_auth_members Columns	37
Table 44-10. pg_cast Columns.....	37
Table 44-11. pg_class Columns	39
Table 44-12. pg_constraint Columns	41
Table 44-13. pg_conversion Columns	41
Table 44-14. pg_database Columns.....	42
Table 44-15. pg_depend Columns.....	43
Table 44-16. pg_description Columns.....	44
Table 44-17. pg_enum Columns.....	44
Table 44-18. pg_foreign_data_wrapper Columns	45
Table 44-19. pg_foreign_server Columns.....	45
Table 44-20. pg_index Columns.....	47
Table 44-21. pg_inherits Columns.....	47
Table 44-22. pg_language Columns.....	48
Table 44-23. pg_largeobject Columns.....	48
Table 44-24. pg_listener Columns.....	49
Table 44-25. pg_namespace Columns.....	49
Table 44-26. pg_opclass Columns	50
Table 44-27. pg_operator Columns.....	50
Table 44-28. pg_opfamily Columns.....	51
Table 44-29. pg_pltemplate Columns	51
Table 44-30. pg_proc Columns.....	53

Volume V

Table 44-31. pg_rewrite Columns	54
Table 44-32. pg_shdepend Columns.....	55
Table 44-33. pg_shdescription Columns.....	56
Table 44-34. pg_statistic Columns	57
Table 44-35. pg_tablespace Columns.....	57
Table 44-36. pg_trigger Columns	58
Table 44-37. pg_ts_config Columns	59
Table 44-38. pg_ts_config_map Columns.....	59
Table 44-39. pg_ts_dict Columns	60
Table 44-40. pg_ts_parser Columns	60
Table 44-41. pg_ts_template Columns.....	61
Table 44-42. pg_type Columns.....	64
Table 44-43. typcategory Codes.....	64
Table 44-44. pg_user_mapping Columns	65
Table 44-45. System Views	66
Table 44-46. pg_cursors Columns	66
Table 44-47. pg_group Columns.....	67
Table 44-48. pg_indexes Columns	67
Table 44-49. pg_locks Columns.....	68
Table 44-50. pg_prepared_statements Columns	70
Table 44-51. pg_prepared_xacts Columns	71
Table 44-52. pg_roles Columns.....	72
Table 44-53. pg_rules Columns.....	72
Table 44-54. pg_settings Columns.....	73
Table 44-55. pg_shadow Columns	73
Table 44-56. pg_stats Columns.....	75
Table 44-57. pg_tables Columns	75
Table 44-58. pg_timezone_abbrevs Columns.....	76
Table 44-59. pg_timezone_names Columns	76
Table 44-60. pg_user Columns	76
Table 53-1. Contents of PGDATA	167
Table 53-2. Overall Page Layout.....	172
Table 53-3. PageHeaderData Layout.....	173
Table 53-4. HeapTupleHeaderData Layout	174
Table A-1. PostgreSQL Error Codes.....	195
Table B-1. Month Names	198

Table B-2. Day of the Week Names	198
Table B-3. Date/Time Field Modifiers	198
Table C-1. SQL Key Words	230
Table D-1. SQL Conformance - Supported Features.....	243
Table D-2. SQL Conformance - Unsupported Features.....	253
Table F-1. Cube external representations	288
Table F-2. Cube functions	290
Table F-3. Cube-based earthdistance functions	317
Table F-4. Point-based earthdistance operators.....	317
Table F-5. hstore Operators.....	320
Table F-6. hstore Functions.....	321
Table F-7. intarray Functions.....	324
Table F-8. intarray Operators	325
Table F-9. isn data types.....	327
Table F-10. isn functions	328
Table F-11. ltree Operators	334
Table F-12. ltree Functions	335
Table F-13. oid2name switches.....	338
Table F-14. pgbench initialization options	345
Table F-15. pgbench benchmarking options.....	346
Table F-16. pgbench common options.....	346
Table F-17. pg_buffercache Columns.....	349
Table F-18. Supported algorithms for crypt ()	352
Table F-19. Iteration counts for crypt ()	352
Table F-20. Hash algorithm speeds	353
Table F-21. Summary of functionality with and without OpenSSL.....	360
Table F-22. pgrowlocks output columns	364
Table F- 23. pg_standby options.....	367
Table F-24. pg_stat_statements columns.....	369
Table F-25. pgstattuple output columns.....	372
Table F-26. pgstatindex output columns.....	373
Table F-27. pg_trgm functions.....	374
Table F-28. pg_trgm operators.....	374
Table F-29. seg external representations.....	377
Table F-30. Examples of valid seg input.....	377
Table F-31. tablefunc functions	384

Volume V

Table F-32. <code>connectby</code> parameters	392
Table F-33. Functions for UUID Generation	398
Table F-34. Functions Returning UUID Constants	398
Table F-35. <code>xml2</code> Functions.....	401
Table F-36. <code>xpath_table</code> Parameters	401
Table G-1. Externally Maintained Client Interfaces.....	406
Table G-2. Externally Maintained Procedural Languages	406

License

PostgreSQL is released under the BSD license.

PostgreSQL Database Management System
(formerly known as Postgres, then as Postgres95)

Portions Copyright (c) 1996-2009, The PostgreSQL Global Development Group

Portions Copyright (c) 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Abstract

Welcome to the *PostgreSQL 8.4 Official Documentation*! After many years of development, PostgreSQL has become feature-complete in many areas. This release shows a targeted approach to adding features (e.g., authentication, monitoring, space reuse), and adds capabilities defined in the later SQL standards.

Part VII.

Internals

This part contains assorted information that might be of use to PostgreSQL developers.

Chapter 43.

Overview of PostgreSQL Internals

Author: This chapter originated as part of *Enhancement of the ANSI SQL Implementation of PostgreSQL* (see page 437), Stefan Simkovics' Master's Thesis prepared at Vienna University of Technology under the direction of O.Univ.Prof.Dr. Georg Gottlob and Univ.Ass. Mag. Katrin Seyr.

This chapter gives an overview of the internal structure of the backend of PostgreSQL. After having read the following sections you should have an idea of how a query is processed. This chapter does not aim to provide a detailed description of the internal operation of PostgreSQL, as such a document would be very extensive. Rather, this chapter is intended to help the reader understand the general sequence of operations that occur within the backend from the point at which a query is received, to the point at which the results are returned to the client.

43.1. The Path of a Query

Here we give a short overview of the stages a query has to pass in order to obtain a result.

1. A connection from an application program to the PostgreSQL server has to be established. The application program transmits a query to the server and waits to receive the results sent back by the server.
2. The *parser stage* checks the query transmitted by the application program for correct syntax and creates a *query tree*.
3. The *rewrite system* takes the query tree created by the parser stage and looks for any *rules* (stored in the *system catalogs*) to apply to the query tree. It performs the transformations given in the *rule bodies*.

One application of the rewrite system is in the realization of *views*. Whenever a query against a view (i.e., a *virtual table*) is made, the rewrite system rewrites the user's query to a query that accesses the *base tables* given in the *view definition* instead.

4. The *planner/optimizer* takes the (rewritten) query tree and creates a *query plan* that will be the input to the *executor*.

It does so by first creating all possible *paths* leading to the same result. For example if there is an index on a relation to be scanned, there are two paths for the scan. One possibility is a simple sequential scan and the other possibility is to use the index. Next the cost for the execution of each path is estimated and the cheapest path is chosen. The cheapest path is expanded into a complete plan that the executor can use.

5. The executor recursively steps through the *plan tree* and retrieves rows in the way represented by the plan. The executor makes use of the *storage system* while scanning relations, performs *sorts* and *joins*, evaluates *qualifications* and finally hands back the rows derived.

In the following sections we will cover each of the above listed items in more detail to give a better understanding of PostgreSQL's internal control and data structures.

43.2. How Connections are Established

PostgreSQL is implemented using a simple "process per user" client/server model. In this model there is one *client process* connected to exactly one *server process*. As we do not know ahead of time how many connections will be made, we have to use a *master process* that spawns a new server process every time a connection is requested. This master process is called `postgres` and listens at a specified TCP/IP port for incoming connections. Whenever a request for a connection is detected the `postgres` process spawns a new server process. The server tasks communicate with each other using *semaphores* and *shared memory* to ensure data integrity throughout concurrent data access.

The client process can be any program that understands the PostgreSQL protocol described in *Chapter 45* (page 78). Many clients are based on the C-language library `libpq`, but several independent implementations of the protocol exist, such as the Java JDBC driver.

Once a connection is established the client process can send a query to the *backend* (server). The query is transmitted using plain text, i.e., there is no parsing done in the *frontend* (client). The server parses the query, creates an *execution plan*, executes the plan and returns the retrieved rows to the client by transmitting them over the established connection.

43.3. The Parser Stage

The *parser stage* consists of two parts:

- The *parser* defined in `gram.y` and `scan.l` is built using the Unix tools `yacc` and `lex`.
- The *transformation process* does modifications and augmentations to the data structures returned by the parser.

43.3.1. Parser

The parser has to check the query string (which arrives as plain ASCII text) for valid syntax. If the syntax is correct a *parse tree* is built up and handed back; otherwise an error is returned. The parser and lexer are implemented using the well-known Unix tools yacc and lex.

The *lexer* is defined in the file `scan.l` and is responsible for recognizing *identifiers*, the *SQL key words* etc. For every key word or identifier that is found, a *token* is generated and handed to the parser.

The parser is defined in the file `gram.y` and consists of a set of *grammar rules* and *actions* that are executed whenever a rule is fired. The code of the actions (which is actually C code) is used to build up the parse tree.

The file `scan.l` is transformed to the C source file `scan.c` using the program `lex` and `gram.y` is transformed to `gram.c` using `yacc`. After these transformations have taken place a normal C compiler can be used to create the parser. Never make any changes to the generated C files as they will be overwritten the next time `lex` or `yacc` is called.



Note

The mentioned transformations and compilations are normally done automatically using the *makefiles* shipped with the PostgreSQL source distribution.

A detailed description of yacc or the grammar rules given in `gram.y` would be beyond the scope of this paper. There are many books and documents dealing with `lex` and `yacc`. You should be familiar with yacc before you start to study the grammar given in `gram.y` otherwise you won't understand what happens there.

43.3.2. Transformation Process

The parser stage creates a parse tree using only fixed rules about the syntactic structure of SQL. It does not make any lookups in the system catalogs, so there is no possibility to understand the detailed semantics of the requested operations. After the parser completes, the *transformation process* takes the tree handed back by the parser as input and does the semantic interpretation needed to understand which tables, functions, and operators are referenced by the query. The data structure that is built to represent this information is called the *query tree*.

The reason for separating raw parsing from semantic analysis is that system catalog lookups can only be done within a transaction, and we do not wish to start a transaction immediately upon receiving a query string. The raw parsing stage is sufficient to identify the transaction control commands (`BEGIN`, `ROLLBACK`, etc), and these can then be correctly executed without any further analysis. Once we know that we are dealing with an actual

query (such as `SELECT` or `UPDATE`), it is okay to start a transaction if we're not already in one. Only then can the transformation process be invoked.

The query tree created by the transformation process is structurally similar to the raw parse tree in most places, but it has many differences in detail. For example, a `FuncCall` node in the parse tree represents something that looks syntactically like a function call. This might be transformed to either a `FuncExpr` or `Aggref` node depending on whether the referenced name turns out to be an ordinary function or an aggregate function. Also, information about the actual data types of columns and expression results is added to the query tree.

43.4. The PostgreSQL Rule System

PostgreSQL supports a powerful *rule system* for the specification of *views* and ambiguous *view updates*. Originally the PostgreSQL rule system consisted of two implementations:

- The first one worked using *row level* processing and was implemented deep in the *executor*. The rule system was called whenever an individual row had been accessed. This implementation was removed in 1995 when the last official release of the Berkeley Postgres project was transformed into Postgres95.
- The second implementation of the rule system is a technique called *query rewriting*. The *rewrite system* is a module that exists between the *parser stage* and the *planner/optimizer*. This technique is still implemented.

The query rewriter is discussed in some detail in *Chapter 36* (Vol.III page 94), so there is no need to cover it here. We will only point out that both the input and the output of the rewriter are query trees, that is, there is no change in the representation or level of semantic detail in the trees. Rewriting can be thought of as a form of macro expansion.

43.5. Planner/Optimizer

The task of the *planner/optimizer* is to create an optimal execution plan. A given SQL query (and hence, a query tree) can be actually executed in a wide variety of different ways, each of which will produce the same set of results. If it is computationally feasible, the query optimizer will examine each of these possible execution plans, ultimately selecting the execution plan that is expected to run the fastest.



Note

In some situations, examining each possible way in which a query can be executed would take an excessive amount of time and memory space. In particular, this occurs when executing queries involving large numbers of join operations. In order to determine a reasonable (not necessarily optimal) query plan in a reasonable amount of time, PostgreSQL uses a Genetic Query Optimizer (page 134) when the number of joins exceeds a threshold (see `geqo_threshold` - Vol.II page 97).

The planner's search procedure actually works with data structures called *paths*, which are simply cut-down representations of plans containing only as much information as the planner needs to make its decisions. After the cheapest path is determined, a full-fledged *plan tree* is built to pass to the executor. This represents the desired execution plan in sufficient detail for the executor to run it. In the rest of this section we'll ignore the distinction between paths and plans.

43.5.1. Generating Possible Plans

The planner/optimizer starts by generating plans for scanning each individual relation (table) used in the query. The possible plans are determined by the available indexes on each relation. There is always the possibility of performing a sequential scan on a relation, so a sequential scan plan is always created. Assume an index is defined on a relation (for example a B-tree index) and a query contains the restriction `relation.attribute OPR constant`. If `relation.attribute` happens to match the key of the B-tree index and `OPR` is one of the operators listed in the index's *operator class*, another plan is created using the B-tree index to scan the relation. If there are further indexes present and the restrictions in the query happen to match a key of an index, further plans will be considered. Index scan plans are also generated for indexes that have a sort ordering that can match the query's `ORDER BY` clause (if any), or a sort ordering that might be useful for merge joining (see below).

If the query requires joining two or more relations, plans for joining relations are considered after all feasible plans have been found for scanning single relations. The three available join strategies are:

- *nested loop join*: The right relation is scanned once for every row found in the left relation. This strategy is easy to implement but can be very time consuming. (However, if the right relation can be scanned with an index scan, this can be a good strategy. It is possible to use values from the current row of the left relation as keys for the index scan of the right.)
- *merge join*: Each relation is sorted on the join attributes before the join starts. Then the two relations are scanned in parallel, and matching rows are combined to form join rows. This kind of join is more attractive because each relation has to be scanned only once. The required sorting might be achieved either by an explicit sort step, or by scanning the relation in the proper order using an index on the join key.
- *hash join*: the right relation is first scanned and loaded into a hash table, using its join attributes as hash keys. Next the left relation is scanned and the appropriate values of every row found are used as hash keys to locate the matching rows in the table.

When the query involves more than two relations, the final result must be built up by a tree of join steps, each with two inputs. The planner examines different possible join sequences to find the cheapest one.

If the query uses fewer than `geqo_threshold` (Vol.II page 97) relations, a near-exhaustive search is conducted to find the best join sequence. The planner preferentially considers joins between any two relations for which there exist a corresponding join clause in the `WHERE` qualification (i.e., for which a restriction like `where rel1.attr1=rel2.attr2` exists). Join pairs with no join clause are considered only when there is no other choice, that is, a particular relation has no available join clauses to any other relation. All possible plans are generated for every join pair considered by the planner, and the one that is (estimated to be) the cheapest is chosen.

When `geqo_threshold` is exceeded, the join sequences considered are determined by heuristics, as described in *Chapter 49* (page 134). Otherwise the process is the same.

The finished plan tree consists of sequential or index scans of the base relations, plus nested-loop, merge, or hash join nodes as needed, plus any auxiliary steps needed, such as sort nodes or aggregate-function calculation nodes. Most of these plan node types have the additional ability to do *selection* (discarding rows that do not meet a specified boolean condition) and *projection* (computation of a derived column set based on given column values, that is, evaluation of scalar expressions where needed). One of the responsibilities of the planner is to attach selection conditions from the `WHERE` clause and computation of required output expressions to the most appropriate nodes of the plan tree.

43.6. Executor

The *executor* takes the plan handed back by the planner/optimizer and recursively processes it to extract the required set of rows. This is essentially a demand-pull pipeline mechanism. Each time a plan node is called, it must deliver one more row, or report that it is done delivering rows.

To provide a concrete example, assume that the top node is a `MergeJoin` node. Before any merge can be done two rows have to be fetched (one from each subplan). So the executor recursively calls itself to process the subplans (it starts with the subplan attached to `lefttree`). The new top node (the top node of the left subplan) is, let's say, a `Sort` node and again recursion is needed to obtain an input row. The child node of the `Sort` might be a `SeqScan` node, representing actual reading of a table. Execution of this node causes the executor to fetch a row from the table and return it up to the calling node. The `Sort` node will repeatedly call its child to obtain all the rows to be sorted. When the input is exhausted (as indicated by the child node returning a `NULL` instead of a row), the `Sort` code performs the sort, and finally is able to return its first output row, namely the first one in sorted order. It keeps the remaining rows stored so that it can deliver them in sorted order in response to later demands.

The `MergeJoin` node similarly demands the first row from its right subplan. Then it compares the two rows to see if they can be joined; if so, it returns a join row to its caller. On

the next call, or immediately if it cannot join the current pair of inputs, it advances to the next row of one table or the other (depending on how the comparison came out), and again checks for a match. Eventually, one subplan or the other is exhausted, and the `MergeJoin` node returns `NULL` to indicate that no more join rows can be formed.

Complex queries can involve many levels of plan nodes, but the general approach is the same: each node computes and returns its next output row each time it is called. Each node is also responsible for applying any selection or projection expressions that were assigned to it by the planner.

The executor mechanism is used to evaluate all four basic SQL query types: `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. For `SELECT`, the top-level executor code only needs to send each row returned by the query plan tree off to the client. For `INSERT`, each returned row is inserted into the target table specified for the `INSERT`. (A simple `INSERT ... VALUES` command creates a trivial plan tree consisting of a single `Result` node, which computes just one result row. But `INSERT ... SELECT` can demand the full power of the executor mechanism.) For `UPDATE`, the planner arranges that each computed row includes all the updated column values, plus the *TID* (tuple ID, or row ID) of the original target row; the executor top level uses this information to create a new updated row and mark the old row deleted. For `DELETE`, the only column that is actually returned by the plan is the *TID*, and the executor top level simply uses the *TID* to visit each target row and mark it deleted.

Part VIII.

Appendixes

Appendix J.

Acronyms

This is a list of acronyms commonly used in the PostgreSQL documentation and in discussions about PostgreSQL.

ANSI

American National Standards Institute
(http://en.wikipedia.org/wiki/American_National_Standards_Institute)

API

Application Programming Interface
(<http://en.wikipedia.org/wiki/API>)

ASCII

American Standard Code for Information Interchange
(<http://en.wikipedia.org/wiki/Ascii>)

BKI

Backend Interface
(page 176)

CA

Certificate Authority
(http://en.wikipedia.org/wiki/Certificate_authority)

CIDR

Classless Inter-Domain Routing
(http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing)

CPAN

Comprehensive Perl Archive Network
(<http://www.cpan.org/>)

CRL

Certificate Revocation List
(http://en.wikipedia.org/wiki/Certificate_revocation_list)

CSV

Comma Separated Values

(http://en.wikipedia.org/wiki/Comma-separated_values)

CVE

Common Vulnerabilities and Exposures

(<http://cve.mitre.org/>)

CVS

Concurrent Versions System

(http://en.wikipedia.org/wiki/Concurrent_Versions_System)

DBA

Database Administrator

(http://en.wikipedia.org/wiki/Database_administrator)

DBI

Database Interface (Perl)

(<http://dbi.perl.org/>)

DBMS

Database Management System

(<http://en.wikipedia.org/wiki/Dbms>)

DDL

Data Definition Language

(http://en.wikipedia.org/wiki/Data_Definition_Language),

SQL commands such as CREATE TABLE, ALTER USER

DML

Data Manipulation Language

(http://en.wikipedia.org/wiki/Data_Manipulation_Language),

SQL commands such as INSERT, UPDATE, DELETE

DST

Daylight Saving Time

(http://en.wikipedia.org/wiki/Daylight_saving_time)

ECPG

Embedded C for PostgreSQL

(Vol.II page 329)

ESQL

Embedded SQL
(http://en.wikipedia.org/wiki/Embedded_SQL)

FAQ

Frequently Asked Questions
(<http://en.wikipedia.org/wiki/FAQ>)

FSM

Free Space Map
(page 171)

GEQO

Genetic Query Optimizer
(page 134)

GIN

Generalized Inverted Index
(page 161)

GiST

Generalized Search Tree
(page 151)

GMT

Greenwich Mean Time
(<http://en.wikipedia.org/wiki/GMT>)

GSSAPI

Generic Security Services Application Programming Interface
(http://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface)

GUC

Grand Unified Configuration
(Vol.II page 77), the PostgreSQL subsystem that handles server configuration

HBA

Host-Based Authentication
(Vol.II page 129)

HOT

Heap-Only Tuples
(<http://anoncvs.postgresql.org/cvsweb.cgi/pgsql/src/backend/access/heap/README.HOT>)

IEC

International Electrotechnical Commission

(http://en.wikipedia.org/wiki/International_Electrotechnical_Commission)

IEEE

Institute of Electrical and Electronics Engineers

(<http://standards.ieee.org/>)

IPC

Inter-Process Communication

(http://en.wikipedia.org/wiki/Inter-process_communication)

ISO

International Standards Organization

(<http://www.iso.org/iso/home.htm>)

ISSN

International Standard Serial Number

(<http://en.wikipedia.org/wiki/Issn>)

JDBC

Java Database Connectivity

(http://en.wikipedia.org/wiki/Java_Database_Connectivity)

LDAP

Lightweight Directory Access Protocol

(http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol)

MSVC

Microsoft Visual C

(http://en.wikipedia.org/wiki/Visual_C++)

MVCC

Multi-Version Concurrency Control

(Vol.I page 361)

NLS

National Language Support

(http://en.wikipedia.org/wiki/Internationalization_and_localization)

ODBC

Open Database Connectivity

(http://en.wikipedia.org/wiki/Open_Database_Connectivity)

OID

Object Identifier
(Vol.I page 187)

OLAP

Online Analytical Processing
(<http://en.wikipedia.org/wiki/Olap>)

OLTP

Online Transaction Processing
(<http://en.wikipedia.org/wiki/OLTP>)

ORDBMS

Object-Relational Database Management System
(<http://en.wikipedia.org/wiki/ORDBMS>)

PAM

Pluggable Authentication Modules
(http://en.wikipedia.org/wiki/Pluggable_Authentication_Modules)

PGSQL

PostgreSQL
(Vol.I page 3)

PGXS

PostgreSQL Extension System
(Vol.III page 46)

PID

Process Identifier
(http://en.wikipedia.org/wiki/Process_identifier)

PITR

Point-In-Time Recovery
(Vol.II page 184)
(Continuous Archiving)

PL

Programming Languages (server-side)
(Vol.III page 12)

POSIX

Portable Operating System Interface
(<http://en.wikipedia.org/wiki/POSIX>)

RDBMS

Relational Database Management System
(http://en.wikipedia.org/wiki/Relational_database_management_system)

RFC

Request For Comments
(http://en.wikipedia.org/wiki/Request_for_Comments)

SGML

Standard Generalized Markup Language
(<http://en.wikipedia.org/wiki/SGML>)

SPI

Server Programming Interface
(Vol.III page 210)

SQL

Structured Query Language
(<http://en.wikipedia.org/wiki/SQL>)

SRF

Set-Returning Function
(Vol.III page 51)

SSH

Secure Shell
(http://en.wikipedia.org/wiki/Secure_Shell)

SSL

Secure Sockets Layer
(http://en.wikipedia.org/wiki/Secure_Sockets_Layer)

SSPI

Security Support Provider Interface
(<http://msdn2.microsoft.com/en-us/library/aa380493.aspx>)

SYSV

Unix System V
(http://en.wikipedia.org/wiki/System_V)

TCP/IP

Transmission Control Protocol (TCP) / Internet Protocol (IP)
(http://en.wikipedia.org/wiki/Transmission_Control_Protocol)

TID

Tuple Identifier
(Vol.I page 187)

TOAST

The Oversized-Attribute Storage Technique
(page 169)

TPC

Transaction Processing Performance Council
(<http://www.tpc.org/>)

URL

Uniform Resource Locator
(<http://en.wikipedia.org/wiki/URL>)

UTC

Coordinated Universal Time
(http://en.wikipedia.org/wiki/Coordinated_Universal_Time)

UTF

Unicode Transformation Format
(<http://www.unicode.org/>)

UTF8

Eight-Bit Unicode Transformation Format
(<http://en.wikipedia.org/wiki/Utf8>)

UUID

Universally Unique Identifier
(Vol.I page 170)

WAL

Write-Ahead Log
(Vol.II page 230)

XID

Transaction Identifier
(Vol.I page 187)

XML

Extensible Markup Language
(<http://en.wikipedia.org/wiki/XML>)

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 SQL Variants*, Fourth Edition, Addison-Wesley Professional, ISBN 0-201-70309-2, 2001.

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*, Eighth Edition, Addison-Wesley, ISBN 0-321-19784-4, 2003.

Ramez Elmasri and Shamkant Navathe, *Fundamentals of Database Systems*, Fourth Edition, Addison-Wesley, ISBN 0-321-12226-7, 2003.

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.

¹ <http://s2k-ftp.cs.berkeley.edu:8000/postgres/papers/>

Zelaine Fong, The design and implementation of the POSTGRES query optimizer², University of California, Berkeley, Computer Science Department.

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, Apr, 1990.

L. Rowe and M. Stonebraker, "*The POSTGRES data model*"³, Proc. VLDB Conference, Sept. 1987.

P. Seshadri and A. Swami, "Generalized Partial Indexes (*cached version*)"⁴, Proc. Eleventh International Conference on Data Engineering, 6-10 March 1995, IEEE Computer Society Press, Cat. No.95CH35724, 1995, 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, 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.

² <http://s2k-ftp.cs.berkeley.edu:8000/postgres/papers/UCB-MS-zfong.pdf>

³ <http://s2k-ftp.cs.berkeley.edu:8000/postgres/papers/ERL-M87-13.pdf>

⁴ <http://citeseer.ist.psu.edu/seshadri95generalized.html>

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>

List of Volumes

Volume I. The SQL Language

Preface

- What is PostgreSQL?
- A Brief History of PostgreSQL
- Conventions
- Further Information
- Bug Reporting Guidelines

Part I. Tutorial

1. Getting Started
2. The SQL Language
3. Advanced Features

Part II. The SQL Language

4. SQL Syntax
5. Data Definition
6. Data Manipulation
7. Queries
8. Data Types
9. Functions and Operators
10. Type Conversion
11. Indexes
12. Full Text Search
13. Concurrency Control
14. Performance Tips

Volume II. Server Administration

Part III. Server Administration

15. Installation from Source Code
16. Installation from Source Code on Windows
17. Server Setup and Operation

Volume V

18. Server Configuration
19. Client Authentication
20. Database Roles and Privileges
21. Managing Databases
22. Localization
23. Routine Database Maintenance Tasks
24. Backup and Restore
25. High Availability, Load Balancing, and Replication
26. Monitoring Database Activity
27. Monitoring Disk Usage
28. Reliability and the Write-Ahead Log
29. Regression Tests

Part IV. Client Interfaces

30. libpq - C Library
31. Large Objects
32. ECPG - Embedded SQL in C
33. The Information Schema

Volume III. Server Programming

Part V. Server Programming

34. Extending SQL
35. Triggers
36. The Rule System
37. Procedural Languages
38. PL/pgSQL - SQL Procedural Language
39. PL/Tcl - Tcl Procedural Language
40. PL/Perl - Perl Procedural Language
41. PL/Python - Python Procedural Language
42. Server Programming Interface

Volume IV. Reference

Part VI. Reference

- I. SQL Commands
- II. PostgreSQL Client Applications
- III. PostgreSQL Server Applications

Volume V. Internals and Appendixes

Part VII. Internals

- 43. Overview of PostgreSQL Internals
- 44. System Catalogs
- 45. Frontend/Backend Protocol
- 46. PostgreSQL Coding Conventions
- 47. Native Language Support
- 48. Writing A Procedural Language Handler
- 49. Genetic Query Optimizer
- 50. Index Access Method Interface Definition
- 51. GiST Indexes
- 52. GIN Indexes
- 53. Database Physical Storage
- 54. BKI Backend Interface
- 55. How the Planner Uses Statistics

Part VIII. Appendixes

- A. PostgreSQL Error Codes
- B. Date/Time Support
- C. SQL Key Words
- D. SQL Conformance
- E. Release Notes
- F. Additional Supplied Modules
- G. External Projects
- H. The CVS Repository
- I. Documentation
- J. Acronyms

Bibliography

Index

Index

dblink.....	291	pg_auth_members.....	36
dict_int.....	314	pg_authid.....	35
dict_xsyn.....	314	pg_buffercache.....	349
earthdistance.....	315	pg_cast.....	37
elog.....	116	pg_class.....	38
ereport.....	116	pg_constraint.....	40
error codes		pg_conversion.....	41
list of.....	187	pg_cursors.....	66
extensions.....	406	pg_database.....	41
Free Space Map.....	171	pg_depend.....	42
fuzzystrmatch.....	317	pg_description.....	44
hstore.....	319	pg_enum.....	44
index		pg_foreign_data_wrapper.....	45
GIN.....	161	pg_foreign_server.....	45
GIST.....	151	pg_freespacemap.....	362
intagg.....	322	pg_group.....	67
intarray.....	324	pg_index.....	46
interfaces		pg_indexes.....	67
externally maintained.....	405	pg_inherits.....	47
isn.....	326	pg_language.....	47
key word		pg_largeobject.....	48
list of.....	202	pg_listener.....	48
lo330		pg_locks.....	67
pageinspect.....	341	pg_namespace.....	49
pg_aggregate.....	31	pg_opclass.....	49
pg_am.....	31	pg_operator.....	50
pg_amop.....	32	pg_opfamily.....	50
pg_amproc.....	33	pg_pltemplate.....	51
pg_attrdef.....	33	pg_prepared_statements.....	70
pg_attribute.....	34	pg_prepared_xacts.....	70

pg_proc.....	52	procedural language	
pg_rewrite.....	54	externally maintained.....	405
pg_roles.....	71	handler for.....	131
pg_rules.....	72	protocol	
pg_settings.....	72	frontend-backend.....	78
pg_shadow.....	73	row estimation	
pg_shdepend.....	54	planner.....	179
pg_shdescription.....	55	seg.....	375
pg_standby.....	365	sliced bread.....	<i>See</i> TOAST
pg_stat_statements.....	368	SPI	
pg_statistic.....	56	examples.....	379
pg_stats.....	74	SQL/CLI.....	231
pg_tables.....	75	SQL/Foundation.....	231
pg_tablespace.....	57	SQL/Framework.....	231
pg_timezone_abbrevs.....	75	SQL/JRT.....	231
pg_timezone_names.....	76	SQL/MED.....	231
pg_trgm.....	373	SQL/OLB.....	231
pg_trigger.....	58	SQL/PSM.....	231
pg_ts_config.....	58	SQL/Schemata.....	231
pg_ts_config_map.....	59	SQL/XML.....	231
pg_ts_dict.....	59	sslinfo.....	382
pg_ts_parser.....	60	tablefunc.....	384
pg_ts_template.....	60	test_parser.....	394
pg_type.....	61	time zone	
pg_user.....	76	input abbreviations.....	198
pg_user_mapping.....	64	TOAST.....	169
pg_user_mappings.....	77	tsearch2.....	395
pg_views.....	77	uuid-osp.....	397
pgbench.....	343	vacuumlo.....	398
pgcrypto.....	350	Visibility Map.....	171
pgrowlocks.....	364	VM.....	<i>See</i> Visibility Map
pgstattuple.....	371	xml2.....	400
