

11 February 2017
For PostgreSQL 9.6

PostgreSQL Internals (1)

Hewlett Packard Enterprise Japan Co, Ltd.
Noriyoshi Shinoda

Acknowledgment

Satoshi Nagayasu (Uptime Technologies, LLC.) and Ryota Watabe (CO-Sol Inc.) gave me reviews of this document before publishing its first edition. I received reviews from Akiko Takeshima, Tomoo Takahashi, and Takahiro Kitayama in Hewlett-Packard Enterprise Japan, Technology Consulting department. I really appreciated their support.

For updated edition, I received opinions from Satoshi, Ryota, Tomoo, Takahiro, and Akiko again. I received a review from Tomoo, Takahiro, and Akiko for this third edition. To create the English version, there was a devoted support from my family.

I would like to thank to everyone who develop open source software. I hope that this document will be useful for engineers who use PostgreSQL.

16 July 2014 1st Edition (for PostgreSQL 9.3)

16 March 2015 2nd Edition (for PostgreSQL 9.4)

11 February 2017 3rd Edition (for PostgreSQL 9.6)

Noriyoshi Shinoda

Author

Noriyoshi Shinoda

- Hewlett-Packard Enterprise Japan Co, Ltd
 - ✧ 1990-2000 Digital Equipment Corporation Japan / Compaq Computer Japan
 - ✧ 2000-Current Hewlett-Packard Enterprise Japan
- 10 years Software Development by C, C++, Java™, Java Script, Perl, Visual C++®, Visual Basic® and PHP on UNIX and Microsoft Windows environment
- 16 years consultant of data-store for PostgreSQL, Oracle Database, Microsoft SQL Server, Sybase ASE, JBoss Data Virtualization and HPE Vertica
- Wrote 15 books about Oracle Database administration and application development
- Oracle ACE
- Oracle Certified Master
- Mail: noriyoshi.shinoda@hpe.com

Index

Acknowledgment	2
Author.....	2
Index.....	3
Glossary	10
1. About This Document.....	12
1.1 Purpose	12
1.2 Audience	12
1.3 Scope	12
1.4 Software Version.....	12
1.5 Corresponding to the update request.....	12
1.6 Question, Comment, and Responsibility	12
1.7 Conversions.....	13
1.7.1 Conversion.....	13
1.7.2 Examples Notation	14
2. Process and Memory Architecture.....	15
2.1 Process Architecture.....	15
2.1.1 Process parent-child relationship.....	15
2.1.2 Process Names	16
2.1.3 Process and Signal.....	18
2.1.4 Starting and stopping the processes	28
2.1.5 Process configuration of the Microsoft Windows environment.	29
2.2 Memory Architecture	31
2.2.1 Shared buffer Overview	31
2.2.2 Implementation of the shared buffer	31
2.2.3 Huge Pages	32
2.2.4 Semaphore	34
2.2.5 Checkpoint.....	36
2.2.6 Ring Buffer	38
2.3 Operation at the time of instance startup / shutdown	39
2.3.1 Waiting for startup / shutdown completion.....	39
2.3.2 Instance parameter setting	41
2.3.3 Loading the external library.....	46
2.3.4 Behavior during instance stopping failure.....	47
2.3.5 Load library of instance startup.....	47



2.3.6 Major input and output files.....	48
2.3.7 Behavior at the time of Windows Service stop.	50
3. Storage Architecture	51
3.1 Structure of the Filesystem	51
3.1.1 Directory Structure	51
3.1.2 Database directory internals.....	52
3.1.3 TOAST feature	55
3.1.4 Relationship between TRUNCATE statement and file	57
3.1.5 FILLFACTOR attribute.....	59
3.2 Tablespace.....	61
3.2.1 What is tablespace?.....	61
3.2.2 Relationship between the database object and the file	62
3.3 File system and behavior	67
3.3.1 Protection mode of the database cluster	67
3.3.2 Update File	69
3.3.3 Visibility Map and Free Space Map.....	71
3.3.4 VACUUM behavior	73
3.3.5 Opened Files	83
3.3.6 Behavior of process (Writing WAL data)	85
3.3.7 Behavior of process (Writing by checkpoint)	88
3.3.8 Behavior of process (Writing by writer).....	89
3.3.9 Behavior of process (archiver)	89
3.4 Online Backup.....	91
3.4.1 Behavior of online backup.....	91
3.4.2 Backup Label File.....	93
3.4.3 Online Backup with Replication Environment	95
3.4.4 Instance shutdown with online backup	95
3.5 File Format.....	97
3.5.1 Postmaster.pid	97
3.5.2 Postmaster.opts.....	98
3.5.3 PG_VERSION	98
3.5.4 Pg_control	99
3.5.5 pg_filenode.map.....	103
3.6 Block format.....	106
3.6.1 Block and Page.....	106
3.6.2 Tuple	107

3.7 Wraparound problem of transaction ID	110
3.7.1 Transaction ID.....	110
3.7.2 The parameters for the FREEZE processing.....	112
3.8 Locale specification	113
3.8.1 Specifying the locale and encoding	113
3.8.2 The use of the index by LIKE	115
3.8.3 Using the index by <, > operators.....	116
3.8.4 Locale and encoding of the specified location	118
3.9 Data Checksum.....	119
3.9.1 Specifying the checksum.....	119
3.9.2 Checksum location	119
3.9.3 Checksum Error	120
3.9.4 Check the existence of checksum.....	121
3.10 Log File.....	122
3.10.1 Output of the log file.....	122
3.10.2 Log file name	122
3.10.3 Log Rotation	124
3.10.4 Contents of the log.....	125
3.10.5 Encoding of log file	126
4. Trouble Shooting.....	129
4.1 File deletion before startup instance.....	129
4.1.1 Deleted pg_control	129
4.1.2 Deleted WAL file.....	129
4.1.3 Behavior on data file deletion (Normal instance stop).....	130
4.1.4 Behavior of data file deletion (Instance Crash / No data changed)	131
4.1.5 Behavior of data file deletion (Instance Crash / Updated).....	133
4.1.6 Other files	134
4.2 Delete files in the instance running.....	135
4.2.1 Delete pg_control	135
4.2.2 Delete WAL	135
4.3 Process Failure	137
4.3.1 Behavior of the process abnormal termination.....	137
4.3.2 Behavior of the transaction at the process abnormal termination.	138
4.4 Other failure	139
4.4.1 Crash recovery	139
4.4.2 Instance abnormal termination during online backup	139

4.4.3 Failure of the archiving	140
5. Performance Related Information	144
5.1 Automatic statistical information collection.....	144
5.1.1 Timing	144
5.1.2 Conditions	144
5.1.3 The number of sample tuples	144
5.1.4 Information collected as statistics.....	147
5.1.5 Destination of the statistics	150
5.2 Automatic VACUUM	151
5.2.1 Interval	151
5.2.2 Conditions	151
5.2.3 Autovacuum worker process startup	151
5.2.4 Amount of usable memory	152
5.3 Execution Plan.....	153
5.3.1 EXPLAIN statement	153
5.3.2 Costs	154
5.3.3 Execution plan.....	155
5.3.4 Execution time	159
5.3.5 Cost estimate of the empty table	159
5.3.6 Disk sort	160
5.3.7 Table sequential scan and index scan	163
5.3.8 BUFFERS parameter	165
5.4 Configuration Parameters	167
5.4.1 Parameters related to performance	167
5.4.2 Effective_cache_size parameter	167
5.4.3 Effective_io_concurrency parameter	167
5.5 System Catalog	169
5.5.1 Entity of the system catalog.....	169
6. Specification of SQL statements	170
6.1 Lock	170
6.1.1 Lock type.....	170
6.1.2 Acquisition of lock.....	170
6.2 Partition Table	172
6.2.1 Partition Table Overview.....	172
6.2.2 Partition Table Implementation	172
6.2.3 Verify the execution plan.....	174

6.2.4 Constraint	178
6.2.5 Record move between partitions	178
6.2.6 Partition table and statistics	179
6.2.7 Partition table with External table	179
6.3 Sequence Object	182
6.3.1 Using the SEQUENCE object.....	182
6.3.2 Cache	183
6.3.3 Transaction	185
6.4 Bind variables and PREPARE statement	186
6.5 INSERT ON CONFLICT statement.....	188
6.5.1 Basic syntax of INSERT ON CONFLICT statement.....	188
6.5.2 Relation between ON CONFLICT Clause and Trigger.....	190
6.5.3 ON CONFLICT clause and Execution Plan	191
6.5.4 ON CONFLICT clause and the partition table.....	193
6.6 TABLESAMPLE	195
6.6.1 Overview	195
6.6.2 SYSTEM と BERNOULLI	195
6.6.3 Execution Plan	197
6.7 Changing a table attribute.	199
6.7.1 ALTER TABLE SET UNLOGGED	199
6.7.2 ALTER TABLE SET WITH OIDS.....	201
6.7.3 ALTER TABLE MODIFY COLUMN TYPE	202
6.8 ECPG.....	204
6.8.1 Format of the host variable	204
6.8.2 Behavior at the time of out-of-space.....	205
6.9 Parallel Query.....	207
6.9.1 Overview	207
6.9.2 Execution plan.....	209
6.9.3 Parallel processing and functions	210
6.9.4 Calculation of the degree of parallelism.....	214
7. Privileges and object creation	216
7.1 Object Privileges.....	216
7.1.1 The owner of the tablespace	216
7.1.2 The owner of the database	216
7.2 Row Level Security.....	217
7.2.1 What's Row Level Security	217

7.2.2 Preparation.....	217
7.2.3 Create POLICY object	218
7.2.4 Parameter Settings	221
8. Utilities.....	223
8.1 Utility usage	223
8.1.1 Pg_basebackup command	223
8.1.2 Pg_archivecleanup command	226
8.1.3 Psql command.....	227
8.1.4 Pg_resetxlog command	229
8.1.5 Pg_rewind command	230
8.1.6 Vacuumdb command.....	233
8.2 Exit status of Server/Client Applications	235
8.2.1 Pg_ctl command.....	235
8.2.2 Psql command.....	235
8.2.3 Pg_basebackup command	237
8.2.4 Pg_archivecleanup command	237
8.2.5 Initdb command	237
8.2.6 Pg_isready command	237
8.2.7 Pg_receivexlog command	238
9. System Configuration	239
9.1 Default Value of Parameters.....	239
9.1.1 Parameters derived at initdb command execution	239
9.2 Recommended Setting.....	240
9.2.1 Locale setting	240
9.2.2 Recommended parameter values	240
10. Streaming Replication.....	242
10.1 Mechanism of streaming replication	242
10.1.1 The streaming replication	242
10.1.2 Configuration of streaming replication	242
10.2 Construction of the replication environment	244
10.2.1 Replication Slot	244
10.2.2 Synchronous and asynchronous	247
10.2.3 Parameters	249
10.2.4 Recovery.conf file.....	250
10.3 Failover and Switchover	252
10.3.1 Procedure of switchover	252



10.3.2 Pg_ctl promote command	252
10.3.3 Promoted to the master by the trigger file	253
10.3.4 Log on a failure	253
11. Source code Tree	255
11.1 Directory Structure	255
11.1.1 Top directory	255
11.1.2 "src" directory	255
11.2 Build Environment	256
11.2.1 Configure command parameters	256
11.2.2 Make command parameters	256
12. Linux Operating System Configuration	257
12.1 Kernel Parameters	257
12.1.1 Memory Overcommit	257
12.1.2 I/O Scheduler	257
12.1.3 SWAP	257
12.1.4 Huge Pages	258
12.1.5 Semaphore	258
12.2 Filesystem Settings	258
12.2.1 When using the ext4 filesystem	258
12.2.2 When using the XFS filesystem	258
12.3 Core File Settings	259
12.3.1 CORE file output settings	259
12.3.2 Core administration with ABRT	259
12.4 User limits	261
12.5 systemd support	261
12.5.1 Service registration	261
12.5.2 Service start and stop	263
12.6 Others	265
12.6.1 SSH	265
12.6.2 Firewall	265
12.6.3 SE-Linux	265
12.6.4 systemd	265
Appendix. Bibliography	266
Appendix.1 Books	266
Appendix 2. URL	267
Modification History	268

Glossary

Table 1 Glossary

Terminology	Description
ACID	Set of properties that database transactions should held (Atomicity, Consistency, Isolation, Durability).
Contrib module	Library that extends the PostgreSQL functions. A list of Contrib modules, which can be used in a standard, is listed in the manual "Appendix F. Additional Supplied Modules ¹ ".
ECPG	Preprocessor for embedded SQL package for PostgreSQL.
EnterpriseDB	Company, which develops and sells Postgres Plus.
GUC	Memory area where PostgreSQL parameters are stored (Global Unified Configuration).
OID (Object ID)	ID number that identifies the object created in the internal database. Unsigned 32-bit value.
PL/pgSQL	One of the stored procedure language of PostgreSQL. It has a compatibility to some extent with PL/SQL in Oracle Database.
Postgres Plus	Commercial database products based on PostgreSQL.
PostgreSQL	Open source RDBMS product.
psql	Utility to execute SQL statements included in PostgreSQL.
TID (Tuple ID)	ID number that uniquely indicate a tuple in the table. It shows the physical location of the tuple.
WAL	PostgreSQL transaction log file (Write Ahead Logging)
XID (Transaction ID)	ID number that uniquely identifies the transaction; unsigned 32-bit value to distinguish the old and new tuple.
archive log	Copy of WAL used for recovery
system catalog	Table that contains the meta-information of the entire PostgreSQL database.
tuple	It indicates the records in the table.

¹ <https://www.postgresql.org/docs/9.6/static/contrib.html>

Table 1 (Cont.) Glossary

Terminology	Description
database cluster	Directory where the management information of the entire PostgreSQL database is stored.
relation	Same as table.
tablespace	Directory on the file system in which the object is stored.

1. About This Document

1.1 Purpose

This document is written for PostgreSQL engineers. This document is intended to provide knowledge about the "internal structure" and "operations that are not described in the manual" of PostgreSQL.

1.2 Audience

This document is written for engineers who have a knowledge of PostgreSQL such as installation, basic management, etc.

1.3 Scope

The main scope of this document are the internal structure of storage, which PostgreSQL uses, and internal operations not described in the manual. As this document is a material, which the author summarizes the results of research for self-study, there are variations in the technical level and completeness.

1.4 Software Version

This document is intended for the following versions of the software generally.

Table 2 Software Version

Software	Version	Note
PostgreSQL	PostgreSQL 9.6	9.6.2
Operating System	Red Hat Enterprise Linux 7 Update 1 (x86-64)	3.10.0-229
	Microsoft Windows Server 2008 R2	Some part

1.5 Corresponding to the update request

This document will be updated if there is a request, but the time and contents have not been determined.

1.6 Question, Comment, and Responsibility

The contents of this document is not an official opinion of the Hewlett-Packard Enterprise Corporation. The author and affiliation company do not take any responsibility about the problem caused by the mistake of contents. If you have any comments for this document, please contact to Noriyoshi Shinoda (noriyoshi.shinoda@hpe.com).

1.7 Conversions

1.7.1 Conversion

A port surrounded by curly braces ({}) indicates that it is converted into some kind of string. This document uses the following notation.

Table 3 Conversion

Conversion	Meaning	Example
{999999}	An arbitrary number string	16495
{9}	One-digit number	1
{ARCHIVEDFILE}	archived WAL file name	0000000100000000000000A8
{ARCHIVEDIR}	archived file output directory	/usr/local/pgsql/archive
{BACKUPLABEL}	Label string that is specified in the online backup	pg_basebackup base backup
{BGWORKER}	Custom Worker process name	custom_worker
{DATE}	Date and time string	2017-02-11_122532
{HOME}	Home directory	/home/postgres
{INSTALL}	PostgreSQL software install directory	/usr/local/pgsql
{MODULENAME}	Name of Contrib module	auto_explain
{OID}	Any of the OID number	12993
{PARAMETER}	Parameter name	log_checkpoints
{PASSWORD}	Password that does not appear	secret
{PGDATA}	Directory for the database cluster	/usr/local/pgsql/data
{PGDATABASE}	Database name	datadb1
{PGUSER}	Database user name	user1
{PID}	Process ID	3468
{PORT}	Connection port number	5432
{RELFILENODE}	File name that corresponds to the table, heck the relfilenode column of pg_class catalog	16531
{SERVICENAME}	Red Hat Linux service name	postgresql-9.6.2.service
{SLOT}	Replication slot name	slot_1
{SOURCE}	Parameter settings source of macro	
{SQL}	Arbitrary of the SQL statement	SELECT * FROM table1
{TABLE}	Any of the table name	table1

Table 3 Conversion (Cont.)

Conversion	Meaning	Example
{TABLESPACEDIR}	directory name for tablespace	/usr/local/pgsql/ts1
{TCP/IP (PORT)}	Pair of TCP / IP address and port number of client	192.168.1.100(65327)
{VERSION}	Version number	9.6
{WALFILE}	WAL filename	0000000100000000000000B0
{WALOFFSET}	WAL offset string	5225832
{YYYYMMDDN}	Format string	201608131
\${string}	Environment variable	\${PGDATA}

1.7.2 Examples Notation

This document contains examples of the execution of the command or SQL statement. Examples have been described using the following notation:

Table 4 Examples notation

Notation	Description
#	shell prompt for Linux root user
\$	shell prompt for Linux general user
Boldface	user input string
postgres=#	psql prompt for PostgreSQL administrator
postgres=>	psql prompt for PostgreSQL general user
backend>	prompt for PostgreSQL standalone mode
<i>Italic</i>	comment in the examples

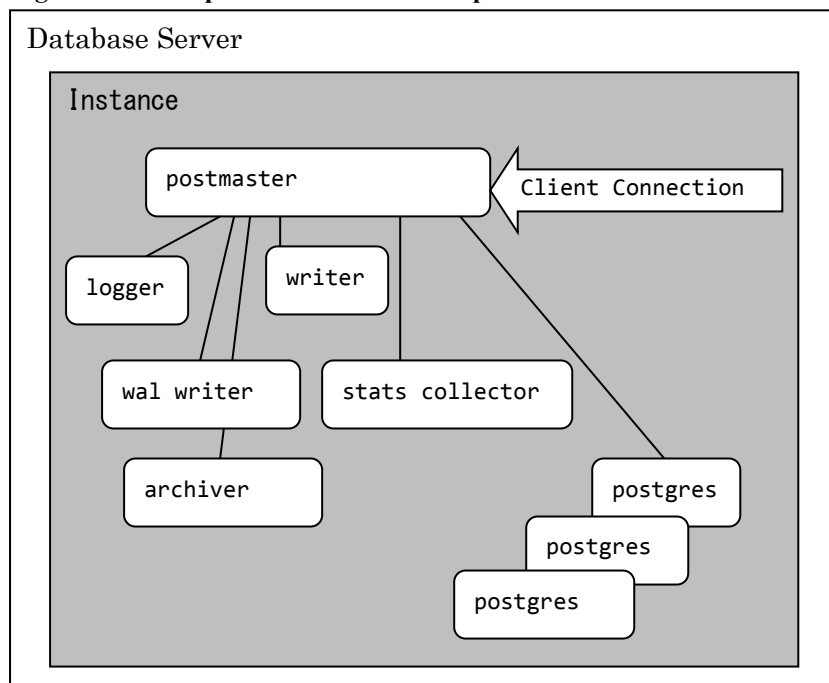
2. Process and Memory Architecture

2.1 Process Architecture

2.1.1 Process parent-child relationship

Process architecture of PostgreSQL is made up from multiple back-end processes whose parent process is postmaster². The process ID of the postmaster process is logged in the {PGDATA}/postmaster.pid file. This file is deleted when the instance is successfully shutdown. The database client makes a connection to the port on which the postmaster process is listening.

Figure 1 Process parent-child relationship



In the example below, process ID 2680 is the postmaster process. It is clear that all other processes are child processes of the postmaster. Postmaster process receives a connection from the client and authenticate it. Then postmaster start the postgres process as a child process to execute the SQL statement.

² The postgres process that becomes the parent of all processes is called "postmaster" by the historical reasons.

Example 1 Process parent-child relationship

```
$ ps -ef | grep postgres | grep -v grep
postgres  2680      1  0 10:25 ?          00:00:00 /usr/local/pgsql/bin/postgres -D
/usr/local/pgsql/data
postgres  2681    2680  0 10:25 ?          00:00:00 postgres: logger process
postgres  2683    2680  0 10:25 ?          00:00:00 postgres: checkpointer process
postgres  2684    2680  0 10:25 ?          00:00:00 postgres: writer process
postgres  2685    2680  0 10:25 ?          00:00:00 postgres: wal writer process
postgres  2686    2680  0 10:25 ?          00:00:00 postgres: autovacuum launcher process
postgres  2687    2680  0 10:25 ?          00:00:00 postgres: stats collector process
```

2.1.2 Process Names

PostgreSQL instance is configured multiple processes as described above. Part of the process name changes by specifying the parameter `update_process_title` to "on" (default: "on"). The names of each process that are referenced in the `ps` command are shown in the table below.

Table 5 Process names

Process	Process Name
postmaster	{INSTALL}/bin/postgres -D {PGDATA}
logger	postgres: logger process
checkpointer	postgres: checkpointer process
writer	postgres: writer process
wal writer	postgres: wal writer process
autovacuum launcher	postgres: autovacuum launcher process
autovacuum worker	postgres: autovacuum worker process {PGDATABASE}
archiver	postgres: archiver process last was {ARCHIVEDFILE}
stats collector	postgres: stats collector process
postgres (local)	postgres: {PGUSER} {PGDATABASE} [local] {SQL}
postgres (remote)	postgres: {PGUSER} {PGDATABASE} {TCP/IP (PORT)} {SQL}
wal sender	postgres: wal sender process {PGUSER} {TCP/IP (PORT)} streaming {WALFILE}
	postgres: wal sender process {PGUSER} {TCP/IP (PORT)} sending backup "{BACKUP_LABEL}"
wal receiver	postgres: wal receiver process streaming {WALFILE}
startup process	postgres: startup process recovering {WALFILE}
bgworker	postgres: bgworker: {BGWORKER}
parallel worker	postgres: bgworker: parallel worker for PID {PID}

If you specify the parameter `cluster_name`, which is a new feature of PostgreSQL 9.5, a string that has been specified as part of the process name is output. Below is an example when the "cluster1" is specified to parameter `cluster_name`.

Example 2 cluster_name parameter

```
$ ps -ef | grep postgres
postgres 12364      1  0 06:14 pts/0    00:00:00 /usr/local/pgsql/bin/postgres -D data
postgres 12365 12364  0 06:14 ?        00:00:00 postgres: cluster1: logger process
postgres 12367 12364  0 06:14 ?        00:00:00 postgres: cluster1: checkpointer process
postgres 12368 12364  0 06:14 ?        00:00:00 postgres: cluster1: writer process
postgres 12369 12364  0 06:14 ?        00:00:00 postgres: cluster1: wal writer process
postgres 12370 12364  0 06:14 ?        00:00:00 postgres: cluster1: autovacuum launcher
process
```

As you can see from this example, cluster name is not output in the process name of the postmaster process. Characters that can be specified in the parameter `cluster_name` is limited to ASCII string (0x20 ~ 0x7E). The other codes are output after being converted into a question mark (?).

2.1.3 Process and Signal

You can execute an action by sending a specific signal to the back-end process, which configure the instance. Here the actions respond to the several signals are verified.

□ SIGKILL signal

The entire process, including the child processes is aborted when the postmaster process receives a KILL signal. The `postmaster.pid` file is not deleted on this occasion. Although the following log is recorded after instance reboot, instance starts normally.

Example 3 Restart log after an abnormal termination

```
LOG:  database system was interrupted; last known up at 2017-02-11 11:12:03 JST
LOG:  database system was not properly shut down; automatic recovery in progress
LOG:  redo starts at 0/155E118
FATAL: the database system is starting up
FATAL: the database system is starting up
LOG:  invalid record length at 0/5A5C050: wanted 24, got 0
LOG:  redo done at 0/5A5C018
LOG:  last completed transaction was at log time 2017-02-11 12:25:15.443492+09
LOG:  MultiXact member wraparound protections are now enabled
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
```

When the postgres process terminate abnormally (including receive KILL signal), as well as the appropriate process, it will be reset all the sessions that are connected from the client. All transactions running on the instance is rolled back, become all SQL statements immediately after the signal reception error. To safely stop the postgres process, to execute the `pg_cancel_backend` function (sending a SIGINT signal), or `pg_terminate_backend` function (sending a SIGTERM signal).

Example 4 Log after receiving the KILL signal

```
LOG:  server process (PID 3416) was terminated by signal 9: Killed
LOG:  terminating any other active server processes
LOG:  archiver process (PID 3404) exited with exit code 1
WARNING: terminating connection because of crash of another server process
DETAIL:  The postmaster has commanded this server process to roll back the
current transaction and exit, because another server process exited abnormally
and possibly corrupted shared memory.
HINT:  In a moment you should be able to reconnect to the database and repeat
your command.
LOG:  all server processes terminated; reinitializing
```

Behavior when each back-end process receives a signal is as follows. SIG_IGN means signal ignored, and SIG_DFL shows the default behavior of the Linux process.

□ Behavior at the signal receiving of postgres process

Operations at signal reception of the postgres process are as follows.

Table 6 Behavior of postgres process

Signal	Signal Handler function	Behavior
SIGHUP	SigHupHandler	Reload configuration file
SIGINT	StatementCancelHandler	Destruction of the running transactions (Processing of pg_cancel_backend function)
SIGTERM	die	Destruction of the running transactions and exit process (Processing of pg_terminate_backend function)
SIGQUIT	quickdie or die	Forced termination
SIGALRM	handle_sig_alarm	Timeout occurrence notification
SIGPIPE	SIG_IGN	
SIGUSR1	procsignal_sigusr1_handler	Database recovery
SIGUSR2	SIG_IGN	
SIGFPE	FloatExceptionHandler	Output ERROR log
SIGCHLD	SIG_DFL	

- Behavior at the signal receiving of autovacuum launcher process

Operations at signal reception of the autovacuum launcher process are as follows.

Table 7 Behavior of autovacuum launcher process

Signal	Signal Handler function	Behavior
SIGHUP	avl_sighup_handler	Reload configuration file
SIGINT	StatementCancelHandler	Detraction of the running transaction
SIGTERM	avl_sigterm_handler	Normal exit
SIGQUIT	quickdie	Output log and force termination
SIGALRM	handle_sig_alarm	Timeout occurrence notification
SIGPIPE	SIG_IGN	
SIGUSR1	procsignal_sigusr1_handler	Execute recovery
SIGUSR2	avl_sigusr2_handler	Exit operation for autovacuum worker
SIGFPE	FloatExceptionHandler	Output ERROR log
SIGCHLD	SIG_DFL	

- Behavior at the signal receiving of bgworker process

Operations at signal reception of the bgworker launcher process are as follows.

Table 8 Behavior of bgworker process

Signal	Signal Handler function	Behavior
SIGHUP	SIG_IGN	
SIGINT	StatementCancelHandler	Detraction of the running transaction
	SIG_IGN	
SIGTERM	bgworker_die	Output FATAL error log
SIGQUIT	bgworker_quickdie	Forced termination
SIGALRM	handle_sig_alarm	Timeout occurrence notification
SIGPIPE	SIG_IGN	
SIGUSR1	procsignal_sigusr1_handler	Execute recovery
	bgworker_sigusr1_handler	Call the latch_sigusr1_handler function
SIGUSR2	SIG_IGN	
SIGFPE	FloatExceptionHandler	Output ERROR log
	SIG_IGN	
SIGCHLD	SIG_DFL	

- Behavior at the signal receiving of writer process

Operations at signal reception of the writer launcher process are as follows.

Table 9 Behavior of writer process

Signal	Signal Handler function	Behavior
SIGHUP	BgSigHupHandler	Reload configuration file
SIGINT	SIG_IGN	
SIGTERM	ReqShutdownHandler	Normal exit
SIGQUIT	bg_quickdie	Forced exit
SIGALRM	SIG_IGN	
SIGPIPE	SIG_IGN	
SIGUSR1	bgwriter_sigusr1_handler	Call the latch_sigusr1_handler function
SIGUSR2	SIG_IGN	
SIGCHLD	SIG_DFL	
SIGTTIN	SIG_DFL	
SIGTTOU	SIG_DFL	
SIGCONT	SIG_DFL	
SIGWINCH	SIG_DFL	

- Behavior at the signal receiving of checkpoint process

Operations at signal reception of the checkpoint launcher process are as follows.

Table 10 Behavior of checkpointer process

Signal	Signal Handler function	Behavior
SIGHUP	ChkptSigHupHandler	Reload configuration file
SIGINT	ReqCheckpointHandler	Request of the checkpoint execution
SIGTERM	SIG_IGN	
SIGQUIT	chkpt_quickdie	Forced exit
SIGALRM	SIG_IGN	
SIGPIPE	SIG_IGN	
SIGUSR1	chkpt_sigusr1_handler	Call the latch_sigusr1_handler function
SIGUSR2	ReqShutdownHandler	Close WAL file and exit normally
SIGCHLD	SIG_DFL	
SIGTTIN	SIG_DFL	
SIGTTOU	SIG_DFL	
SIGCONT	SIG_DFL	
SIGWINCH	SIG_DFL	

When you send a SIGINT signal to checkpointer process, the checkpoint will be executed. However, in case of using this way, log will not be output the parameter log_checkpoints is specified as "on". Pg_stat_bgwriter catalog will be updated.

□ Behavior at the signal receiving of stats collector process

Operations at signal reception of the stats collector process are as follows.



Table 11 Behavior of stats collector process

Signal	Signal Handler function	Behavior
SIGHUP	pgstat_sighup_handler	Reload configuration file.
SIGINT	SIG_IGN	
SIGTERM	SIG_IGN	
SIGQUIT	pgstat_exit	Normal exit
SIGALRM	SIG_IGN	
SIGPIPE	SIG_IGN	
SIGUSR1	SIG_IGN	
SIGUSR2	SIG_IGN	
SIGCHLD	SIG_DFL	
SIGTTIN	SIG_DFL	
SIGTTOU	SIG_DFL	
SIGCONT	SIG_DFL	
SIGWINCH	SIG_DFL	

□ Behavior at the signal receiving of postmaster process

Operations at signal reception of the postmaster process are as follows.

Table 12 Behavior of postmaster process

Signal	Signal Handler function	Behavior
SIGHUP	SIGHUP_handler	Reload configuration file Send SIGHUP signal to the child process
SIGINT	pmdie	FAST shutdown
SIGTERM	pmdie	SMART shutdown
SIGQUIT	pmdie	IMMEDIATE shutdown
SIGALRM	SIG_IGN	
SIGPIPE	SIG_IGN	
SIGUSR1	sigusr1_handler	Signal reception processing from the child process
SIGUSR2	dummy_handler	Does nothing
SIGCHLD	reaper	Processing at the end of a child process Restart the back-end process
SIGTTIN	SIG_IGN	
SIGTTOU	SIG_IGN	
SIGXFSZ	SIG_IGN	

When you send SIGHUP signal to the postmaster process, postgresql.conf file (postgresql.auto.conf, and pg_*.conf file, as well) is reloaded. This is the same behavior as the execution of pg_ctl reload command. The following log is output.

Example 5 Log output of re-read configuration file

```
LOG:  received SIGHUP, reloading configuration files
```

□ Behavior at the signal receiving of startup process startup

Operations at signal reception of the startup process are as follows.

Table 13 Behavior of startup process

Signal	Signal Handler function	Behavior
SIGHUP	StartupProcSigHupHandler	Reload configuration file
SIGINT	SIG_IGN	
SIGTERM	StartupProcShutdownHandler	Exit process
SIGQUIT	startupproc_quickdie	Exit process abnormally
SIGALRM	handle_sig_alarm	Timeout occurrence notification
SIGPIPE	SIG_IGN	
SIGUSR1	StartupProcSigUsr1Handler	Call latch_sigusr1_handler function
SIGUSR2	StartupProcTriggerHandler	Finish recovery operation, promote to master instance
SIGCHLD	SIG_DFL	
SIGTTIN	SIG_DFL	
SIGTTOU	SIG_DFL	
SIGCONT	SIG_DFL	
SIGWINCH	SIG_DFL	

- Behavior at the signal receiving of logger process startup

Operations at signal reception of the logger process are as follows.

Table 14 Behavior of logger process

Signal	Signal Handler function	Behavior
SIGHUP	sigHupHandler	Reload configuration file Reconfirm the log settings and create directory for log
SIGINT	SIG_IGN	
SIGTERM	SIG_IGN	
SIGQUIT	SIG_IGN	
SIGALRM	SIG_IGN	
SIGPIPE	SIG_IGN	
SIGUSR1	sigUsr1Handler	Execute log rotation
SIGUSR2	SIG_IGN	
SIGCHLD	SIG_DFL	
SIGTTIN	SIG_DFL	
SIGTTOU	SIG_DFL	
SIGCONT	SIG_DFL	
SIGWINCH	SIG_DFL	

□ Behavior at the signal receiving of "wal writer" process startup

Operations at signal reception of the wal writer process are as follows.

Table 15 Behavior of wal writer process

Signal	Signal Handler function	Behavior
SIGHUP	WalSigHupHandler	Reload configuration file
SIGINT	WalShutdownHandler	Exit process normally
SIGTERM	WalShutdownHandler	Exit process normally
SIGQUIT	wal_quickdie	Exit process immediately
SIGALRM	SIG_IGN	
SIGPIPE	SIG_IGN	
SIGUSR1	walwriter_sigusr1_handler	Call latch_sigusr1_handler function
SIGUSR2	SIG_IGN	
SIGCHLD	SIG_DFL	
SIGTTIN	SIG_DFL	
SIGTTOU	SIG_DFL	
SIGCONT	SIG_DFL	
SIGWINCH	SIG_DFL	

□ Behavior at the signal receiving of archiver process startup

Operations at signal reception of the archiver process are as follows.

Table 16 Behavior of archiver process

Signal	Signal Handler function	Behavior
SIGHUP	ArchSigHupHandler	Reload configuration file
SIGINT	SIG_IGN	
SIGTERM	ArchSigTermHandler	Exit process normally
SIGQUIT	pgarch_exit	Exit process immediately
SIGALRM	SIG_IGN	
SIGPIPE	SIG_IGN	
SIGUSR1	pgarch_waken	Execute archive the wal file
SIGUSR2	pgarch_waken_stop	Stop archiving
SIGCHLD	SIG_DFL	
SIGTTIN	SIG_DFL	
SIGTTOU	SIG_DFL	
SIGCONT	SIG_DFL	
SIGWINCH	SIG_DFL	

2.1.4 Starting and stopping the processes

Checkpoint, writer, and stats collector processes are always started. Start / stop timing of other processes is as follows. The child processes of postmaster check regularly for the presence of their parent process; postmaster and they terminate their own process when the stoppage of the postmaster process is detected.

Table 17 Start-up and stopping the processes

Process	Start / stop timing
logger	Start-up in the case where parameter logging_collector to "on" (default: "off")
autovacuum launcher	Start-up in the case where parameter autovacuum to "on" (default: "on")
autovacuum worker	Autovacuum launcher process is started with the interval specified by the parameter autovacuum_naptime (default: "1 min"); it stops after completing the work
archiver	Start-up in the case of stand-alone or in replication environment of the master instance, parameter archive_mode to "on" or "always" (default: "off"). In slave instance of replication environment, start-up only if the archive_mode to "always"
postgres (local)	Start with the local connection of the client, and stop on disconnect.
postgres (remote)	Start with the remote connection of the client, and stop on disconnect.
wal sender	<ul style="list-style-type: none"> It starts in the master instance of streaming replication environment. Starts when slave instances come to connect, and stops when slave instances disconnect. Starts during the backup by pg_basebackup command, and stop with the completion of the backup.
wal receiver	Starts in the slave instance of streaming replication environment. When the master instance is stopped, it automatically stops. Restarts when the master instance is restarted.
startup process	Always start in the slave instance of streaming replication environment.
wal writer	It does not start in the slave instance of the replication environment. Except for this situation, it always starts.
bgworker	Behavior is changed according to the specifications of the custom process.
parallel worker	Starting at the time of Parallel Query run, stop at the time of SQL execution is completed.

□ Number of autovacuum worker process

As you can see from its process name, autovacuum worker process is started for each database. The maximum number of the started process is determined by the parameters `autovacuum_max_workers` (default: 3). Each worker process performs the processing on a table-by-table basis.

□ Number of postgres process

Postgres process starts automatically when the client connects. The maximum number of postgres process is limited to the parameter `max_connections` (default: 100).

Number of the connection, which general users that do not have a SUPERUSER privileges can connect, is the results of the calculation of "`max_connections - superuser_reserved_connections` (default: 3)". Following log is output with the connection request with exceed this limit.

Example 6 Excess of the connection number from a general user

```
FATAL:  remaining connection slots are reserved for non-replication
superuser connections
```

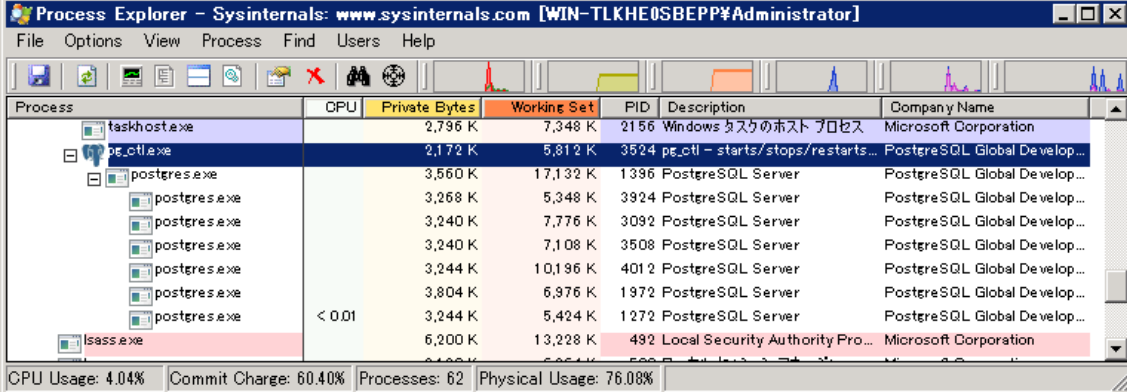
Example 7 Excess of the connection number specified in the parameter `max_connections`

```
FATAL:  sorry, too many clients already
```

2.1.5 Process configuration of the Microsoft Windows environment.

PostgreSQL running on Microsoft Windows will be run as a Windows Service. Execution image, which is registered as a Windows Service is `pg_ctl.exe`. Postgres.exe is executed as a child process of `pg_ctl.exe`, it runs as postmaster. The following figure shows the parent-child relationship of the process in the Windows environment.

Figure 2 Process configuration of the Microsoft Windows environment.



The screenshot shows the Process Explorer window from Sysinternals. The title bar reads 'Process Explorer - Sysinternals: www.sysinternals.com [WIN-TLKHE0SBEPF\Administrator]'. The menu bar includes File, Options, View, Process, Find, Users, and Help. The toolbar contains various icons for process management. The main window displays a list of processes with columns for Process, CPU, Private Bytes, Working Set, PID, Description, and Company Name. The processes listed are taskhost.exe, pg_ctl.exe, and several instances of postgres.exe. The status bar at the bottom shows CPU Usage: 4.04%, Commit Charge: 60.40%, Processes: 62, and Physical Usage: 76.08%.

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
taskhost.exe		2,796 K	7,348 K	2156	Windows タスクのホスト プロセス	Microsoft Corporation
pg_ctl.exe		2,172 K	5,812 K	3524	pg_ctl - starts/stops/restarts...	PostgreSQL Global Develop...
postgres.exe		3,560 K	17,132 K	1396	PostgreSQL Server	PostgreSQL Global Develop...
postgres.exe		3,268 K	5,348 K	3924	PostgreSQL Server	PostgreSQL Global Develop...
postgres.exe		3,240 K	7,776 K	3092	PostgreSQL Server	PostgreSQL Global Develop...
postgres.exe		3,240 K	7,108 K	3508	PostgreSQL Server	PostgreSQL Global Develop...
postgres.exe		3,244 K	10,196 K	4012	PostgreSQL Server	PostgreSQL Global Develop...
postgres.exe		3,804 K	6,976 K	1972	PostgreSQL Server	PostgreSQL Global Develop...
postgres.exe	< 0.01	3,244 K	5,424 K	1272	PostgreSQL Server	PostgreSQL Global Develop...
lsass.exe		6,200 K	13,228 K	492	Local Security Authority Pro...	Microsoft Corporation

CPU Usage: 4.04% Commit Charge: 60.40% Processes: 62 Physical Usage: 76.08%

2.2 Memory Architecture

2.2.1 Shared buffer Overview

PostgreSQL saves the cache blocks in the memory area called "shared buffer", and they are shared among multiple back-end processes. Shared buffer that PostgreSQL instance uses is configured by System V Shared Memory (shmget system call) and the memory-mapped file (mmap system call). For the locking process to cooperation between each process, System V Semaphores are utilized. The number of the semaphore set is not changed even if the number of the connecting client increases.

Example 8 Status of the shared memory

```
$ ipcs -a
```

----- Shared Memory Segments -----						
key	shmid	owner	perms	bytes	nattch	status
0x00530201	2621440	postgres	600	56	5	

----- Semaphore Arrays -----				
key	semid	owner	perms	nsems
0x00530201	19038210	postgres	600	17
0x00530202	19070979	postgres	600	17
0x00530203	19103748	postgres	600	17
0x00530204	19136517	postgres	600	17
0x00530205	19169286	postgres	600	17
0x00530206	19202055	postgres	600	17
0x00530207	19234824	postgres	600	17
0x00530208	19267593	postgres	600	17

----- Message Queues -----					
key	msqid	owner	perms	used-bytes	messages

When an instance is aborted, shared memory and semaphores remain occasionally, though, restart of the instance is done successfully.

2.2.2 Implementation of the shared buffer

System V Shared Memory in Linux environment is created using the shmget system call. For the creation of the System V Shared Memory, a unique key number and size on the host must be specified. The key number is generated using the following formula. If the key is already in use, PostgreSQL

search a free number while incrementing the value. This process is done in the PGSharedMemoryCreate function in the source code (src/backend/port/sysv_shmem.c).

Formula 1 Shared Memory Key

Shared Memory Key = parameter "port" * 1000 + 1

Since the standard port number waiting for a connection (parameter port) is 5,432, key of the shared memory is 5,432,001 (= 0x52e2c1). In PostgreSQL 9.3 or later, the memory volume, which is created as a System V Shared Memory, is the size of the structure PGShmemHeader (src/include/storage/pg_shmem.h). Most of the shared buffer used for tables and indexes are created in the memory-mapped file (mmap system call). The size of the memory area that is created by mmap is the sum of 100KB and calculated value from various parameters. In the Windows environment, the shared memory is configured by CreateFileMapping system call (src/backend/port/win32_shmem.c).

2.2.3 Huge Pages

In Linux environment equipped with large-scale memory Huge Pages feature can be utilized to reduce the memory management load. Adaptation to Huge Pages is a new feature of PostgreSQL 9.4, and it is determined by the parameters huge_pages. Page size when using Huge Pages is 2 MB (2 × 1,024 × 1,024 bytes). If you use Huge Pages, the size of the shared memory to be reserved is adjusted to a multiple of 2 MB based on the calculated value, and MAP_HUGETLB macro is specified to mmap system call.

□ Parameter setting for Huge Pages features

In order to use the Huge Pages features as the shared memory used by PostgreSQL, parameters huge_pages is set.

Table 18 Value that can be specified in the parameter huge_pages

Value	Description	Note
on	Forced to use a Huge Pages.	
off	Do not use a Huge Pages.	
try	Try the use of Huge Pages, use it if possible	Default value

If you specify a "try", the default values, it attempts to create a shared memory by specifying the MAP_HUGETLB macro to mmap system call. If it fails, shared memory is re-created by deleting the

MAP_HUGETLB macro. When "on" is specified to this parameter, Huge Pages is used forcibly. If the platform does not support Huge Pages, pg_ctl command cannot start instance and outputs the following error message.

Example 9 Error message

```
FATAL:  huge pages not supported on this platform
```

□ How to setup the Huge Pages environment

To enable Huge Pages in a Linux environment, specify the maximum number of pages counted using the 2 MB unit to the kernel parameters vm.nr_hugepages. The default value of this parameter is "0". The information of Huge Pages in use can be referred in /proc/meminfo file.

Example 10 Setup Huge Pages environment on Linux

```
# sysctl -a | grep nr_hugepages
vm.nr_hugepages = 0
vm.nr_hugepages_mempolicy = 0
# sysctl -w vm.nr_hugepages = 1000
vm.nr_hugepages = 1000
# grep ^Huge /proc/meminfo
HugePages_Total:      1000
HugePages_Free:       1000
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:         2048 kB
#
```

If the pages required on instance startup cannot be reserved while the parameter huge_pages to "on" is specified, following error occurs and the instance cannot start.

Example 11 Huge Pages page shortage error

```
$ pg_ctl -D data start -w
server starting
FATAL:  could not map anonymous shared memory: Cannot allocate memory
HINT:  This error usually means that PostgreSQL's request for a shared
memory segment exceeded available memory, swap space or huge pages. To
reduce the request size (currently 148324352 bytes), reduce PostgreSQL's
shared memory usage, perhaps by reducing shared_buffers or
max_connections.
```

Notice

In Red Hat Enterprise Linux 6.4, because due to the lack of MAP_HUGETLB macro in the header file, Huge Pages non-compliant binary is created when you build from the source code. At the time of binary creation, please check if following line is in the /usr/include/bits/mman.h.

```
# define MAP_HUGETLB    0x40000          /* Create huge page mapping. */
```

□ Calculation of the required memory area as Huge Pages

Volume of the shared memory, which PostgreSQL instances use, is calculated from the value of some parameters. A about 10 to 50 MB is added to the volume of parameter `shared_buffers` and parameter `wal_buffers`. This additional amount of memory is calculated from parameters `max_worker_processes`, `autovacuum_max_workers`, etc. For the kernel parameter `vm.nr_hugepages`, specify this value with rounding off 2 MB unit.

In order to know the accurate required amount of shared memory, start the instance by specifying `DEBUG3` to the parameter `log_min_messages`. The following message is output to the instance startup log (specified by `pg_ctl -l`).

Example 12 Shared memory required capacity

```
DEBUG:  invoking IpcMemoryCreate(size=148324352)
```

2.2.4 Semaphore

Semaphore has been used for lock control to prevent resource contention between the back-end processes. In PostgreSQL, semaphore sets, whose number is calculated from the following parameters,

are created at the instance startup.

Formula 2 Number of Semaphore Sets

```
Number of maximum backend processes =  
    max_connections + autovacuum_max_workers + 1 + max_worker_processes  
  
Number of semaphore sets = CEIL(# of maximum backend processes/17 + 1)
```

Each semaphore set contains 17 pieces of semaphore. In the case of Red Hat Enterprise Linux 6, the default value of the semaphore-related kernel parameters are ensured to include a sufficient amount for the database with the maximum number of sessions of about 1,000. If the semaphore-related kernel parameters are insufficient, following error occurs and the instance cannot start.

Example 13 Semaphore-related insufficient resources error

```
$ pg_ctl -D data start -w  
waiting for server to start....  
FATAL:  could not create semaphores: No space left on device  
DETAIL:  Failed system call was semget(5440029, 17, 03600).  
HINT:  This error does *not* mean that you have run out of disk space.  
It occurs when either the system limit for the maximum number of semaphore  
sets (SEMMNI), or the system wide maximum number of semaphores (SEMMNS),  
would be exceeded. You need to raise the respective kernel parameter.  
Alternatively, reduce PostgreSQL's consumption of semaphores by reducing  
its max_connections parameter.  
  
    The PostgreSQL documentation contains more information about  
configuring your system for PostgreSQL.  
  
.... stopped waiting  
pg_ctl: could not start server  
Examine the log output.
```

Key of the semaphore set is created using the same logic as the key of the shared memory (src/backend/port/sysv_sema.c). In the Microsoft Windows environment, a semaphore feature is created using CreateSemaphore of Windows API (src/backend/port/win32_sema.c).

2.2.5 Checkpoint

The point at which the memory and storage to guarantee the persistence by synchronized, called a checkpoint. The pages that have been modified on a shared memory in order to create a checkpoint writes to storage. Checkpoint will occur in several timing.

□ Occurrence trigger of checkpoint

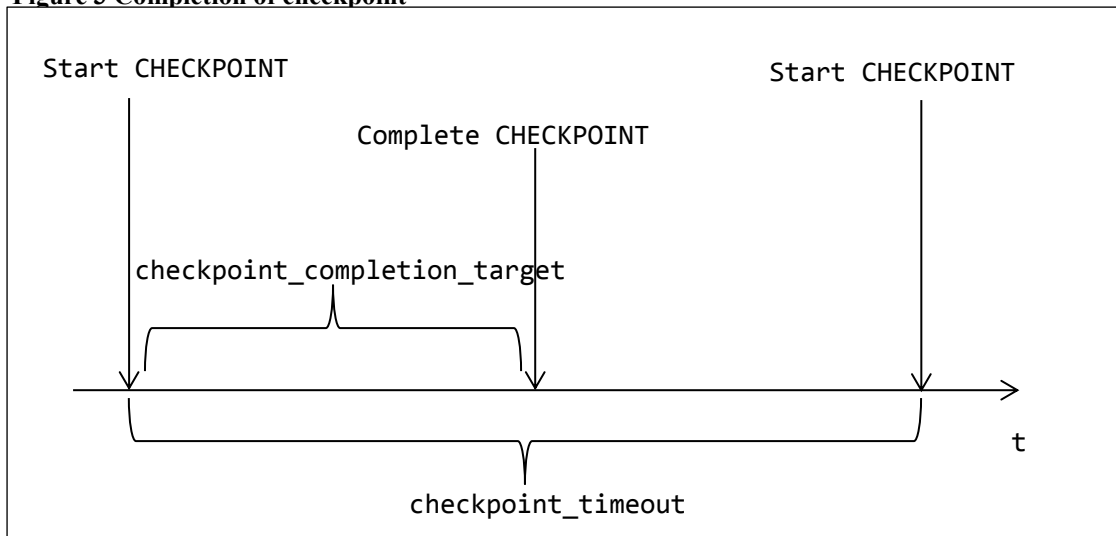
Checkpoint occurs in the following cases:

- Execution of CHECKPOINT statement
When the administrator runs the CHECKPOINT statement.
- With the interval specified in parameter checkpoint_timeout
By default, it runs at 300 seconds (5 minutes) interval.
- Amount of data written to the WAL has reached a parameter max_wal_size.
If the WAL data has been written to the amount specified in the parameter (default: 1GB).
- At the start of online backup
At the execution of pg_start_backup function
At the execution of pg_basebackup command
- At the shutdown of instance
Except for the pg_ctl stop -m immediate command execution
- At the time of database configuration
At the execution the CREATE DATABASE / DROP DATABASE statement

□ Completion of the process at checkpoint

There are two types of checkpoint. One is Regular Checkpoint generated at a certain time interval of the number of the WAL files, another is the Immediate Checkpoint at the time of instance stop or the issuance of the CHECKPOINT statement. For the process of Regular Checkpoint, a function not to write the dirty buffer at once, but to distribute processing to a certain period is provided. By setting the parameter checkpoint_completion_target, you can specify the percentage of time during which the process will be completed by the next checkpoint (specified in the parameter checkpoint_timeout). As the default value is 0.5, checkpoint will be completed within 50% of the time until the next checkpoint starts.

Figure 3 Completion of checkpoint



To check the progress, Autovacuum process compares the ratio of the number of the blocks write completed to that required to be written and the checkpoint interval (parameter `checkpoint_timeout`). If there is a margin for the amount to be written, Autovacuum process stops the processing for the 100 millisecond and resume it. This decision is done in `IsCheckpointOnSchedule` function (`src/backend/postmaster/checkpointer.c`).

□ The parameters for the checkpoint

Parameters for the checkpoint is as follows.

Table 19 The parameters for the checkpoint

Parameters	Description	Default Value
<code>checkpoint_timeout</code>	Checkpoint interval	5min
<code>bgwriter_delay</code>	writer process write interval	200ms
<code>bgwriter_lru_maxpages</code>	The number of write pages by writer process	100
<code>bgwriter_lru_multiplier</code>	Multiple value of the number of write pages of writer process	2.0
<code>checkpoint_completion_target</code>	Ratio to complete the checkpoint before the next checkpoint time	0.5
<code>log_checkpoints</code>	Write the checkpoint information to the log	off
<code>full_page_writes</code>	Write an entire page to WAL on update immediately after checkpoint	on

2.2.6 Ring Buffer

When Sequential scan for a table or bulk retrieval by COPY TO sentence is executed, the active page on the shared buffer might be deleted from the memory. Therefore, in the case such that a sequential scan is performed to the table which has the accessed table size more than 1/4 of the shared buffer, ring buffer which circulates a part of the shared buffer is utilized. The size of the created ring buffer cannot be changed because it is fixed on the source code.

Table 20 Size of ring buffer

Operation	Size	SQL
Bulk Read	256 KB	Seq Scan CREATE MATERIALIZED VIEW
Bulk Write	16 MB	CREATE TABLE AS COPY FROM
VACUUM	256 KB	VACUUM

The size of the ring buffer that is actually created is the smaller one between the 1/8 of the shared buffer and the size of the above table (src/backend/storage/buffer/freelist.c). Details of the ring buffer is described in the README (src/backend/storage/buffer/README) file.

2.3 Operation at the time of instance startup / shutdown

Startup/Shutdown behavior of instances, are summarizes below.

2.3.1 Waiting for startup / shutdown completion

Pg_ctl command is used to manage the instance. With pg_ctl command, you can specify the -w parameter to wait for completion of the process or -W parameter that do not wait for it. As described in the manual, -W parameter is the default at instance startup / restart, whereas -w parameter is default at instance shutdown (<https://www.postgresql.org/docs/9.6/static/app-pg-ctl.html>).

Table 21 Behavior during instance operation by pg_ctl command

Operation	Default behavior	Note
start	Asynchronous (-W)	
restart	Asynchronous (-W)	Stop process is synchronized
stop	Synchronous (-w)	

Time-out period in the case of performing the wait is specified by the -t parameter. Default value is 60 seconds. Status is checked every second, and it is repeated until the timeout.

□ Behavior of instance startup

Instance startup by "pg_ctl start" command does not wait for the completion of the startup unless -w parameter is specified. For the start of the postmaster process, only the return value of the "system" (3) function is checked (other than Windows). Further, under the Windows environment, though the Windows API CreateRestrictedProcess is running, the return value is not checked. For this reason, in the case the startup error occurs, the return value of the pg_ctl command becomes zero.

Example 14 Behavior of instance startup failure

```
$ pg_ctl -D data start
server starting
LOG:  redirecting log output to logging collector process
HINT:  Future log output will appear in directory "pg_log".
$ pg_ctl -D data start -- start 2 times for the same cluster (result in errors)
pg_ctl: another server might be running; trying to start server anyway
server starting
FATAL:  lock file "postmaster.pid" already exists
Is another postmaster (PID 3950) running in data directory
"/usr/local/pgsql/data"?
$ echo $? -- status of pg_ctl command is 0
0
```

□ A wait in the replication environment

If you specify the "-m smart" parameter³ when instance stops, it waits for the disconnection of the client until the time-out. However, since the connection by the slave instance in a replication environment is not considered as a client, instance can be stopped even if a connection from the slave is made.

Example 15 "-m smart" parameters of the replication environment

```
postgres=# SELECT state FROM pg_stat_replication ;
 state
-----
 streaming
(1 row)
postgres=# \q
$ pg_ctl stop -D data -m smart
waiting for server to shut down..... done
server stopped
```

³ The default value for the -m parameter has been changed to fast from smart in PostgreSQL 9.5.

When the slave instance that is not a hot-standby state (`hot_standby = off`) to start in waiting (`-w`) setting, does not command is completed until the timeout (60 seconds by default). This is because the `pg_ctl` command has confirmed the start of the instance using the PQping function.

Example 16 Start-up of the slave instance is not a hot-standby.

```
$ grep hot_standby data.stdbypostgresql.conf
hot_standby = off
$ pg_ctl -D data.stdbypostgresql start -w
waiting for server to start....
LOG:  redirecting log output to logging collector process
HINT:  Future log output will appear in directory "pg_log".
..... stopped waiting
server is still starting up
$ echo $?
0
```

2.3.2 Instance parameter setting

When an instance starts, `{PGDATA}/postgresql.conf` file is read, and parameters are set. After that `{PGDATA}/postgresql.auto.conf` file is read and parameters are overwritten.

To get a list of the parameters, search the `pg_settings` catalog or execute the "show all" command from `psql` utility. In the source column of the `pg_settings` catalog, information of the source of the parameter settings are provided. Column values below are "GucSource_Names" enum value defined within the source code `src/backend/utils/misc/guc.c`. In fact, it is accessed using the macros that are defined in the enum `GucSource` (`PGC_S_{SOURCE}`). Enum values are defined in the source code `src/include/utils/guc.h`.

Table 22 Source column values in pg_settings Catalog

Column Value	Description	Note
default	Default value	
environment variable	Derived from environment variable of the postmaster	
configuration file	Set in the postgresql.conf file	
command line	Postmaster process startup parameter	
global	Global	Details Unknown
database	Setting per database	
user	Setting per user	
database user	Setting per database and user	
client	Setting from the client	
override	Special case to force the use of default values	
interactive	Boundary for error reporting	
test	Test for each database or each user	
session	Changes by SET statement	

□ Dynamic change of the parameter file

In PostgreSQL 9.4 or later, settings of the parameter file can be permanence dynamically using ALTER SYSTEM statement. Only the user with a superuser privilege can execute ALTER SYSTEM statement.

Syntax 1 ALTER SYSTEM statement

```
ALTER SYSTEM SET parameter_name = value | DEFAULT
ALTER SYSTEM RESET parameter_name
```

The value of the parameter changed using ALTER SYSTEM statement is written into the "{PGDATA}/postgresql.auto.conf" file. Please note that this file should not be changed manually.

Example 17 Change the parameter by ALTER SYSTEM statement

```
postgres=# SHOW work_mem ;
work_mem
-----
4MB
(1 row)
postgres=# ALTER SYSTEM SET work mem = '8MB' ;
ALTER SYSTEM
postgres=# SHOW work_mem ;
work_mem
-----
4MB
(1 row)
postgres=# \q
$ cat data/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by ALTER SYSTEM command.
work mem = '8MB'
$
```

As can be seen in the above example, ALTER SYSTEM statement does not change the parameters of an instance, it rewrites only postgresql.auto.conf file. This file is parsed after the postgresql.conf file is loaded at the time of instance startup or pg_reload_conf function execution, and the values are applied.

Either Specifying the DEFAULT as a parameter value of the ALTER SYSTEM statement or executing the ALTER SYSTEM RESET statement, the specified parameter is removed from the postgresql.auto.conf file.

Example 18 Reset to default value by ALTER SYSTEM statement

```
postgres=# ALTER SYSTEM SET work mem = DEFAULT ;
ALTER SYSTEM
postgres=# \q
$ cat data/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by ALTER SYSTEM command.
$
```

□ Parameter file and the SET statement syntax

When writing a parameter with multiple values in the parameter file, separate the values with a comma (,), and enclose the whole in single quote ('). On the other hand, for changing the parameters of the session using SET statement, do not use a single quotation. The parameters of SET statement enclosed in single quotes will be recognized as a single value.

Example 19 The difference in the syntax between the file and the SET statement

```
$ grep temp_tablespace ${PGDATA}/postgresql.conf
temp_tablespace = 'pg_default,ts1'
$ psql
postgres=# SET temp_tablespace='ts2, ts3' ;
ERROR:  tablespace "ts2, ts3" does not exist
postgres=# SET temp_tablespace=ts2, ts3 ;
SET
postgres=#
```

□ Parameter description format and error

In addition to the PostgreSQL standard parameters, parameters to be used by the Contrib modules can be written in postgresql.conf file. Usually the format of the parameter name is '{MODULENAME}.{PARAMETER}'. On the instance startup, parameters that are written in this format is not checked the validity. Even if the parameters for the Contrib module is described incorrectly, the instance is started successfully. You can also get the information using the wrong parameter name in SHOW statement.

For the above reasons, when you set the parameters for Contrib module, you should check the result after setting.

Example 20 Description of the wrong parameter name

```
$ grep autoexplain postgresql.conf
autoexplain.loganalyze = true -- Correct name is "auto_explain.log_analyze"
$ pg_ctl -D data start -w
waiting for server to start....
LOG:  redirecting log output to logging collector process
HINT:  Future log output will appear in directory "pg_log".
done
server started -- Instance startup normally
$ psql
postgres=# SHOW autoexplain.loganalyze ; -- Can check as SHOW statement
autoexplain.loganalyze
-----
true
(1 row)
```

For ALTER SYSTEM statement, because the parameter names specified in the parameters of the Contrib modules are checked, the parameters with the wrong name could not be specified.

□ Confirmation of the parameter file

The contents of the parameter file (postgresql.conf, postgresql.auto.conf) can be confirmed from pg_file_settings catalog. Every time a search is done in this catalog, the file content is parsed, and we can refer to the information that has been described in the file.

Example 21 Confirm the file contents from the catalog

```
postgres=# ALTER SYSTEM SET port=5434 ;
ALTER SYSTEM
postgres=# SELECT sourcefile, name, setting FROM pg_file_settings
           WHERE name = 'port' ;
           sourcefile                                | name | setting
-----+-----+-----
 /usr/local/pglsq/data/postgresql.auto.conf | port | 5434
(1 row)
```

2.3.3 Loading the external library

PostgreSQL can be extended its functionality by dynamically loading external shared libraries.

□ Parameters for loading the library

The following parameters are defined in order to load the external library automatically. In each parameter, a list of the library (comma delimited) is specified.

Table 23 Parameters for loading the library

Parameter name	Description
shared_preload_libraries	Load at instance startup
session_preload_libraries	Load at the postgres process start; only superuser can change
local_preload_libraries	Load at the postgres process start; general user can change

On postgres process startup, first the library specified in the session_preload_libraries is loaded, libraries specified in the local_preload_libraries is loaded.

□ Parameter value of shared_preload_libraries

In the parameter shared_preload_libraries, a shared library name to be used in the modules such as Contrib module is set. When instance cannot find a shared library at startup, instance startup will result in an error.

The following example shows the error message occurred when the instance is started with setting the parameters under an environment where Contrib module pg_stat_statements has not installed.

Example 22 Error message by setting shared_preload_libraries parameter

```
$ pg_ctl -D data start -w
waiting for server to start....
FATAL:  could not access file "pg_stat_statements": No such file or directory
.... stopped waiting
pg_ctl: could not start server
Examine the log output.
```

Libraries specified in shared_preload_libraries parameter are retrieved from the path specified in parameter dynamic_library_path.

2.3.4 Behavior during instance stopping failure

"Pg_ctl stop -m smart" command waits for the disconnecting of the connection user, but pg_ctl command finishes in return value 1 when it elapses time-out (default 60 seconds). Even when the time-out occurs, the instance is still in a status which indicates "during shutdown". Therefore, a new client connection is not possible. When the existing sessions are all finished, instance shutdown automatically. The timeout setting is specified in the pg_ctl command parameters --timeout = number of seconds (or -t number of seconds).

Example 23 Instance termination time-out

```
$ pg_ctl -D data stop -m smart
waiting for server to shut
down..... failed
pg_ctl: server does not shut down
HINT: The "-m fast" option immediately disconnects sessions rather than
waiting for session-initiated disconnection.
$
$ psql -U user1
psql: FATAL: the database system is shutting down
-- The new session can not be accepted
$
$ pg_ctl stop -m immediate
waiting for server to shut down.... done
server stopped
$
```

2.3.5 Load library of instance startup

Shared libraries that are loaded at instance startup are shown below. The behavior of the instance startup was confirmed by tracing with the strace command.

Table 24 Read library of instance startup

Libraries	Directory
libpq.so.5	{INSTALL}/lib
libc.so.6	/lib64
libpthread.so.6	/lib64
libtinfo.so.5	/lib64
libdl.so.2	/lib64
librt.so.1	/lib64
libm.so.6	/lib64
libnss_files.so.2	/lib64
libselinux.so.1	/lib64
libacl.so.1	/lib64
libattr.so.1	/lib64

2.3.6 Major input and output files

Input and output files used on the instance startup are listed here. It is assumed that the instance was stopped successfully. Default values are specified for the parameters.



Table 25 Input and output files

Filenames	Directory	Note
postgresql.conf	{PGDATA}	
postgresql.auto.conf	{PGDATA}	
PG_VERSION	{PGDATA}	
postmaster.pid	{PGDATA}	
Japan	{INSTALL}/share/postgresql/timezone	
posixrules	{INSTALL}/share/postgresql/timezone	
Default	{INSTALL}/share/postgresql/timezonesets	
pg_control	{PGDATA}/global	
.s.PGSQL.5432.lock	/tmp	
.s.PGSQL.5432	/tmp	
0000	{PGDATA}/pg_notify	re-create
postmaster.opts	{PGDATA}	create
pg_log (directory)	{PGDATA}	create
postgresql-{DATE}.log	{PGDATA}/pg_log	
pgsql_tmp	{PGDATA}/base	
state	{PGDATA}/pg_replslot/{SLOTNAME}	Ver. 9.4-
pg_hba.conf	{PGDATA}	
pg_ident.conf	{PGDATA}	
pg_internal.init	{PGDATA}/global	
recovery.conf	{PGDATA}	
backup_label	{PGDATA}	
000000010...00001	{PGDATA}/pg_xlog	
0000	{PGDATA}/pg_multixact/offsets	
0000	{PGDATA}/pg_clog	
pg_filenode.map	{PGDATA}/global	
global.tmp	{PGDATA}/pg_stat_tmp	
db_{OID}.stat	{PGDATA}/pg_stat	
global.stat	{PGDATA}/pg_stat_tmp	
	{PGDATA}/pg_stat	
db_0.tmp	{PGDATA}/pg_stat_tmp	
archive_status	{PGDATA}/pg_xlog	

2.3.7 Behavior at the time of Windows Service stop.

PostgreSQL instance on Microsoft Windows will be able to operate as a Windows Service. Instance stop processing using the NET STOP command or "Service Manager" is performed in the fast mode (SIGINT signal). The following sources are part of the pgwin32_ServiceMain function in the "src/bin/pg_ctl/pg_ctl.c".

Example 24 Instance termination by NET STOP command

```
static void WINAPI
pgwin32_ServiceMain(DWORD argc, LPTSTR *argv)
{
    ...
    pgwin32_SetServiceStatus(SERVICE_STOP_PENDING);
    switch (ret)
    {
        case WAIT_OBJECT_0:          /* shutdown event */
        {
            /*
             * status.dwCheckPoint can be incremented by
             * test_postmaster_connection(), so it might not start from 0.
             */
            int maxShutdownCheckPoint = status.dwCheckPoint + 12;
            kill(postmasterPID, SIGINT);

            ...
        }
    }
}
```

3. Storage Architecture

3.1 Structure of the Filesystem

In this section, the information about the file system is described.

3.1.1 Directory Structure

The directory structure of the PostgreSQL database cluster is written here.

□ Database Cluster

In Database cluster, all the persisted information of a PostgreSQL database is stored. It is created by the `initdb` command with specifying a directory of the operating system. Database cluster is specified even in the `pg_ctl` command used at the time of instance start and stop, becomes the operational unit of the instance.

Example 25 File structure in the database cluster

```
$ ls -l ${PGDATA}
total 96
-rw----- 1 postgres postgres 4 Feb 11 12:45 PG_VERSION
drwx----- 6 postgres postgres 4096 Feb 11 13:00 base
drwx----- 2 postgres postgres 4096 Feb 11 15:52 global
drwx----- 2 postgres postgres 4096 Feb 11 12:45 pg_clog
-rw----- 1 postgres postgres 4222 Feb 11 12:45 pg_hba.conf
-rw----- 1 postgres postgres 1636 Feb 11 12:45 pg_ident.conf
drwxr-xr-x 2 postgres postgres 4096 Feb 11 15:52 pg_log
.....
drwx----- 2 postgres postgres 4096 Feb 11 15:54 pg_tblspc
drwx----- 2 postgres postgres 4096 Feb 11 12:45 pg_twophase
drwx----- 3 postgres postgres 4096 Feb 11 12:45 pg_xlog
-rw-r--r-- 1 postgres postgres 101 Feb 11 12:45 postgresql.auto.conf
-rw-r--r-- 1 postgres postgres 19598 Feb 11 12:45 postgresql.conf
-rw----- 1 postgres postgres 45 Feb 11 15:52 postmaster.opts
-rw----- 1 postgres postgres 73 Feb 11 15:52 postmaster.pid
$
```

Many directories and files are created in the directory specified as a database cluster. "base" directory is the standard directory where the persistent data is stored. Sub directories corresponding to the database are created in "base" directory.

3.1.2 Database directory internals

Under the directory corresponding to the database, objects stored in the database are created as a separate file. The following files are created automatically.

Table 26 Files that are created under database directory

Filename	Description
{999999}	Segment file
{999999}.{9}	Segment file (exceeding 1 GB)
{999999}_fsm	Free Space Map file
{999999}_vm	Visibility Map file
{999999}_init ⁴	File indicating the initialization fork of UNLOGGED TABLE
pg_filenode.map	The mapping of the OID and the physical file name of the part of the system catalog.
pg_internal.init	Cache file of the system information. It is re-created at instance startup. It is created under "{PGDATA}/global" directory and the directory where the database is saved.
PG_VERSION	Text file where version information is recorded. It is checked at the time of database use.

□ Component of the table and System catalog

The table of PostgreSQL is actually a collection of multiple objects. Internally, it is composed of the following elements.

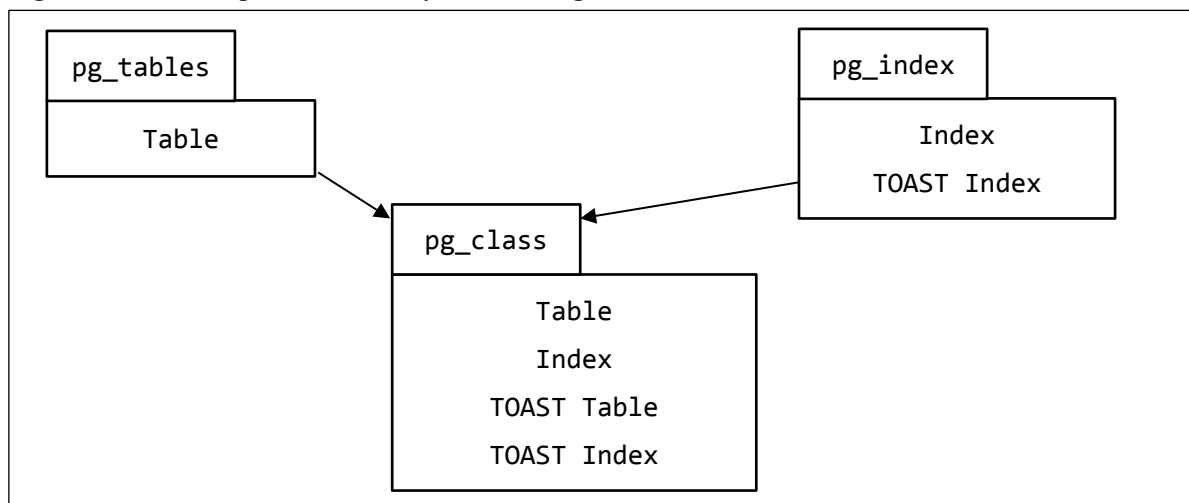
⁴ *_init file is created UNLOGGED TABLE, UNLOGGED TABLE of TOAST table, TOAST index of UNLOGGED TABLE, for an index that has been created for UNLOGGED TABLE.

Table 27 Component of the table

Object	Description	Note
Table	Area where the data is stored	
Index	Area to be created in the table for the rapid search	
TOAST Table	Area for storing large-scale data	Described later
TOAST Index	Index to speed up the search of the TOAST table	Described later

The `pg_class` catalog manages all of the above object elements. In the `pg_class` catalog table name (`relname`), and OID of TOAST table or TOAST index (`reltoastrelid`) are stored. `Pg_tables` catalog is a view, which extracts only tables from `pg_class` catalog. Catalogs, which associate tables and indexes, are `pg_index`. Information such as OID (`indexrelid`) of the table stored in `pg_class` catalog and OID (`indrelid`) of an index are stored in this catalog.

Figure 4 Relationship of Table and System Catalog



□ Relationship between tables and files

Tables, indexes, and file of the operating system correspond to the value of the `relfilenode` column of `pg_class` catalog.

Relationship between the object and the file can be confirmed also with the `oid2name` utility. Tablespace to be stored is confirmed by `reltablespace` column of `pg_class` catalog. If this column value is 0, it indicates that the object is used `pg_default` tablespace.

Example 26 Determination of file

```
$ oid2name -d datadb1
From database "datadb1":
  Filenode  Table Name
-----
      16437      data1

postgres=> SELECT relname, relfilenode, reltablespace FROM pg_class
           WHERE relname IN ('data2', 'data3') ;
relname | relfilenode | reltablespace
-----+-----+-----
data2   |      34115 |           0   -- pg_default tablespace
data3   |      34119 |      32778   -- tbl2 tablespace
```

□ Segment file

Segment file is a file in which the actual data of tables and indexes are stored. When the file size exceeds 1 GB ($RELSEG_SIZE \times BLCKSZ$), multiple files are created. In addition to the original file, files which has the file name added ". {9}" ({9} is a number starting with 1)" at the end are created.

Example 27 Segment file

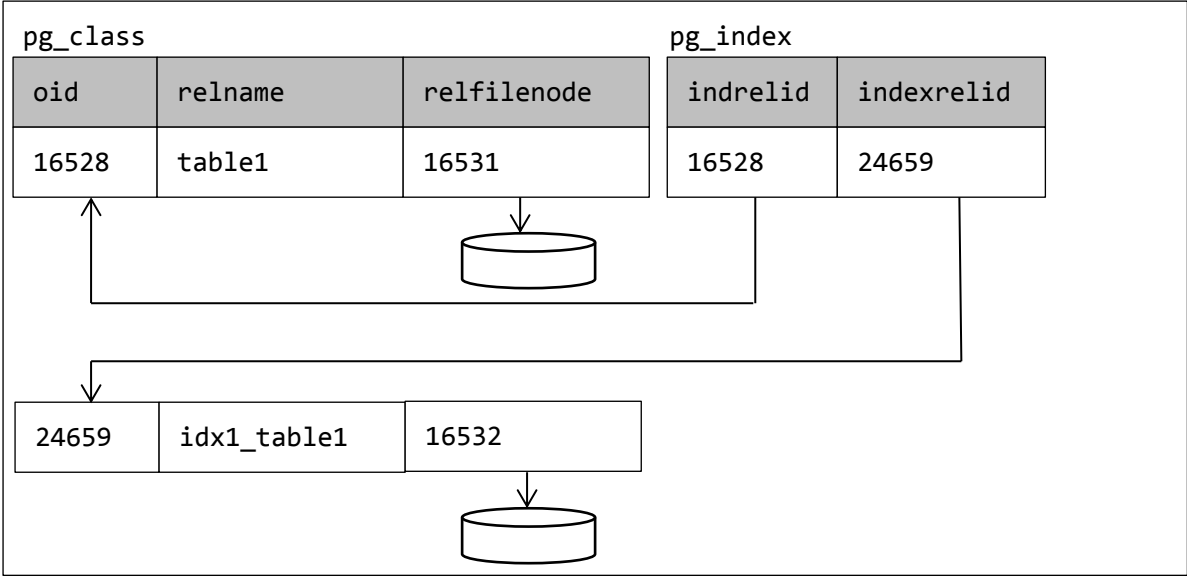
```
postgres=> SELECT oid, relname, relfilenode FROM pg_class WHERE
relname='large1' ;
oid | relname | relfilenode
-----+-----+-----
16468 | large1 |      16495
(1 row)

$ ls -l 16495*
-rw----- . 1 postgres postgres 1073741824 Feb 11 14:06 16495
-rw----- . 1 postgres postgres   96550912 Feb 11 14:06 16495.1
```

□ Index file

Similar to the table, index is also created as a separate file. File name of the index is stored in the relfilenode column of pg_class catalog as well. The catalog witch links the table and index is pg_index.

Figure 5 Relation between pg_class Catalog and Index



3.1.3 TOAST feature

Usually, PostgreSQL stores the tuple in the page of 8 KB unit. A tuple is not stored across the pages. Therefore, large-scale tuples cannot be stored in a page. To store a larger tuple, a feature called TOAST (The Oversized-Attribute Storage Technique) is supplied. TOAST data is created when the compressed column data exceeds the size, which is determined by TOAST_TUPLE_THRESHOLD (determined at compile time). It also stores the data into the TOAST table until it is reduced not more than TOAST_TUPLE_TARGET.

□ TOAST Table

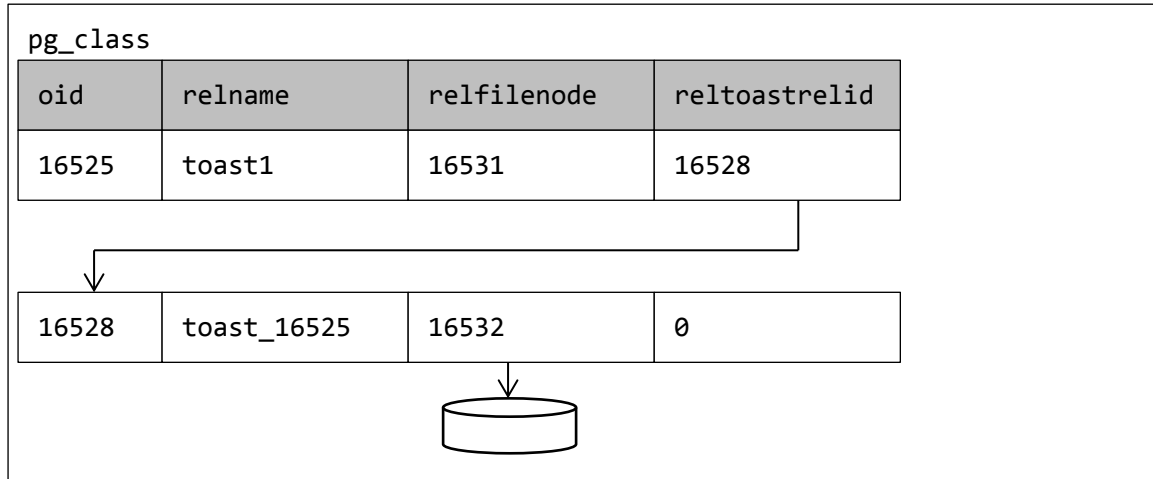
TOAST data are stored in a separate table (another file) with the file specified by the relfilenode column of pg_class catalog. In the reltoastrelid column of pg_class catalogs, oid of TOAST table is stored. By searching the file name of the TOAST table (relfilenode) from pg_class catalog, it is possible to identity the file. In order to speed up the search of the TOAST table, TOAST index is created with TOAST table. TOAST table does not appear in the pg_tables catalog.

Table 28 relname column of pg_class Catalog

Column Value	Description
{TABLENAME}	table name created in the CREATE TABLE statement
toast_{OID}	TOAST table corresponding to the table (OID is oid of the original table)
toast_{OID}_index	TOAST index for the TOAST table

The following figure shows the relationship between the table and the TOAST table. When you create a table toast1, TOAST table toast_16525 is automatically created and saved in the file 16532. TOAST index finds the tuple of indrelid column = 16528 from pg_index catalog.

Figure 6 Relationship between pg_class Catalog and TOAST table



□ Save of TOAST data

The save format can be specified with the TOAST data. Usually save format is determined automatically, but it can also be specified a column-by-column basis.

Table 29 TOAST data save format

Format	Description
PLAIN	Do not use TOAST
EXTENDED	Compression and TOAST table are used. Default value for the many data type that can use TOAST.
EXTERNAL	It does not compress, but uses TOAST table
MAIN	Compression is performed, but TOAST table is not used as a general rule

By executing "\d+ table_name" within psql command, you can confirm the save format of TOAST corresponding column. Following example shows that c1 column (varchar type) and c2 column (text type) of toast1 table are TOAST corresponding.

Example 28 Verify TOAST Column

```
postgres=> \d+ toast1
```

Table "public.toast1"					
Column	Type	Modifiers	Storage	Stats target	Description
c1	numeric		main		
c2	character varying(10)		<u>extended</u>		
c3	text		<u>extended</u>		

Has OIDs: no

To change the default save format, use SET STORAGE clause in ALTER TABLE statement.

Example 29 Change of TOAST storage format

```
postgres=> ALTER TABLE toast1 ALTER c2 SET STORAGE PLAIN ;
ALTER TABLE
postgres=> \d+ toast1
```

Table "public.toast1"					
Column	Type	Modifiers	Storage	Stats target	Description
c1	numeric		main		
c2	character varying(10)		<u>plain</u>		
c3	text		extended		

Has OIDs: no

3.1.4 Relationship between TRUNCATE statement and file

When a transaction, in which TRUNCATE statement is executed, is committed, the table and the corresponding file are truncated to the size of zero without waiting for the checkpoint. When the execution of the TRUNCATE statement is completed, a file is newly created to be INSERT next time, and relfilenode column of pg_class catalog is updated with the new file name. Old files used until executing of TRUNCATE is removed at the timing of the checkpoint.



Example 30 Relationship between TRUNCATE statement and file

```
postgres=> SELECT relfilenode FROM pg_class WHERE relname='tr1' ;
relfilenode
-----
      25782
(1 row)

$ ls -l 2578* -- check the file
-rw----- 1 postgres postgres 884736 Feb 11 11:23 25782
-rw----- 1 postgres postgres  24576 Feb 11 11:23 25782_fsm

postgres=> TRUNCATE TABLE tr1 ;      -- execute TRUNCATE statement
TRUNCATE TABLE
postgres=> SELECT relfilenode FROM pg_class WHERE relname='tr1' ;
relfilenode
-----
      25783  -- new file created
(1 row)

$ ls -l 2578*
-rw----- 1 postgres postgres 0 Feb 11 11:25 25782 -- old file
-rw----- 1 postgres postgres 0 Feb 11 11:25 25783 -- new file
$

postgres=# CHECKPOINT ;      -- execute CHECKPOINT statement
CHECKPOINT
postgres=# \q
$ ls -l 2578*
-rw----- 1 postgres postgres 0 Feb 11 11:25 25783 -- only new file
$
```

3.1.5 FILLFACTOR attribute

When INSERT statement is executed, the tuple is added to the page. If executor cannot store the tuple into the working page anymore, it searches the next free page. FILLFACTOR is an attribute to indicate the percentage of the size in the page where tuples can be stored. The default value for FILLFACTOR is 100 (%). For this reason, tuples are normally stored in the page without gaps. FILLFACTOR can be also specified to the index.

The advantage of the decrease of FILL FACTOR from 100% is the performance improvement of the access in page unit, because free spaces are used in case of update using UPDATE statement. On the other hand, since the number of the pages used by the table to expand, I/O increases in case of reading entire table. For the table or index on which update is frequently performed is recommended to lower the value of the FILLFACTOR from the default value.

□ FILLFACTOR verification at the time of CREATE TABLE statement is executed

To set the FILLFACTOR when creating tables, it should be written to the WITH clause of the CREATE TABLE statement. To verify, refer the reloptions column of pg_class catalog.

Example 31 Setting of FILLFACTOR attribute

```
postgres=> CREATE TABLE fill1(key1 NUMERIC, val1 TEXT) WITH (FILLFACTOR = 85) ;
CREATE TABLE
postgres=> SELECT relname,reloptions FROM pg_class WHERE relname='fill1' ;
 relname |      reloptions
-----+-----
 fill1   | {fillfactor=85}
(1 row)
```

□ Behavior during ALTER TABLE statement execution

To change the FILLFACTOR attribute for an existing table, specify it using SET clause in the ALTER TABLE statement. Even if FILLFACTOR attribute is changed, the tuples in the existing table are not changed.



Example 32 Impact on existing data due to the setting of the FILLFACTOR attribute.

```
postgres=> INSERT INTO fill1 VALUES (generate_series(1, 1000), 'data') ;
INSERT 0 1000
postgres=> SELECT MAX(lp) FROM heap_page_items(get_raw_page('fill1', 0)) ;
max
-----
 157
(1 row)
postgres=> ALTER TABLE fill1 SET (FILLFACTOR = 30) ;
ALTER TABLE
postgres=> SELECT MAX(lp) FROM heap_page_items(get_raw_page('fill1', 0)) ;
max
-----
 157
(1 row)
```

This example stores data in the table fill1. When checking the status of the page using heap_page_items⁵ function, it can be seen that 157 tuples are stored in the first page. This example executes ALTER TABLE statement to change the FILLFACTOR attributes, then it confirms the information on the page again, and it can be seen that the same number of tuples are stored.

⁵ This function is defined by Contrib modules pageinspect. Requires superuser privileges.

3.2 Tablespace

3.2.1 What is tablespace?

In PostgreSQL database, persistent objects such as DATABASE, TABLE, INDEX, and MATERIALIZED VIEW are stored in a tablespace. When database cluster is created, two tablespaces are created by default. Pg_default tablespace is used by the general user. Pg_global tablespace contains the system catalog to be shared by all database. If tablespace (TABLESPACE clause) is not specified on creating the database, pg_default tablespace is used.

□ Parameter default_tablespace

With the parameter default_tablespace, tablespace name is specified, which is used on objects such as TABLE, INDEX, MATERIALIZED VIEW without TABLESPACE clause. This setting of the parameter does not affect the destination of the database using CREATE DATABASE statement. The default value for this parameter is " (the empty string). To save the object, a tablespace where the database has been saved is used. This parameter can be changed not only as entire instance, but also per session.

Table 30 Object destination in case where is not specified tablespace

Object	Parameter default_tablespace value	
	Value is specified	No Value (empty string)
DATABASE	pg_default	pg_default
TABLE	Specified tablespace	Same tablespace as the database
INDEX	Specified tablespace	Same tablespace as the database
MATERIALIZED VIEW	Specified tablespace	Same tablespace as the database
SEQUENCE ⁶	Specified tablespace	Same tablespace as the database

If specified parameter default_tablespace per session using SET statement, it checks whether the table space with the specified name exists, but it does not actually check whether there is an object creation privilege. If specified with per-instance basis using the postgresql.conf file, the presence of the specified tablespace is not checked. In that case, tablespace at the destination database is used if TABLESPACE clause is omitted.

⁶ Though there is no TABLESPACE clause in the CREATE SEQUENCE statement syntax, it will be affected by default_tablespace parameter.

Example 33 Behavior when non-existent tablespace is specified

```
postgres=> SHOW default_tablespace ;
default_tablespace
-----
ts_bad          -- Specifies the table space name that does not exist
                  in the postgresql.conf
(1 row)
postgres=> CREATE TABLE data1 (c1 NUMERIC, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> \d data1
           Table "public.data1"
  Column |          Type          | Modifiers
-----+-----+-----
c1       | numeric                |
c2       | character varying(10)  |
           -- default table space name is used
postgres=> SET default_tablespace = ts_bad2 ;
SET default_tablespace = ts_bad2 ;
ERROR:  invalid value for parameter "default_tablespace": "ts_bad2"
DETAIL:  Tablespace "ts_bad2" does not exist.
-- When the parameters default_tablespace is changed by the SET
statement, check is performed.
```

□ Parameter temp_tablespaces

It specifies a list of tablespace names to create a temporary object. If more than one name is specified, the tablespace used will be chosen randomly.

3.2.2 Relationship between the database object and the file

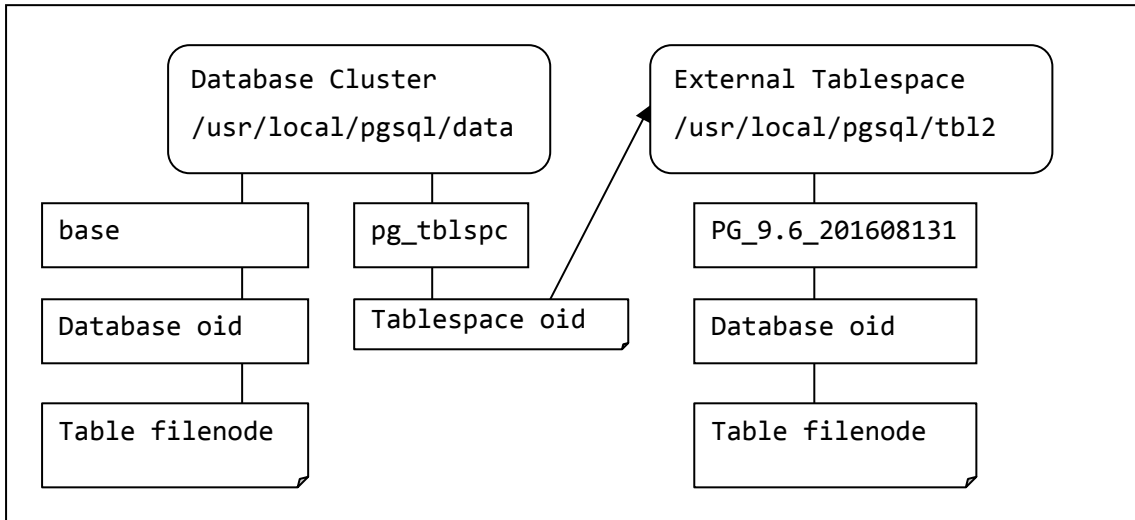
In PostgreSQL, objects such as databases and tables are correspond to the directories and files of the operating system.

□ Specific tablespace

Pg_default tablespace corresponds to the "base" directory in the database cluster. With an external tablespace is created a symbolic link is created in the {PGDATA}/pg_tblspc directory. File name of

the symbolic link corresponds to the oid column of pg_tablespace catalog.

Figure 7 Directory structure and tablespace



Example 34 Correspondence of tablespace

```

postgres=# CREATE TABLESPACE tbl2 LOCATION '/usr/local/pgsql/tbl2' ;
CREATE TABLESPACE
postgres=# SELECT oid, spcname FROM pg_tablespace ;
   oid | spcname
-----+-----
  1663 | pg_default
  1664 | pg_global
 32788 | tbl2
(3 rows)

$ ls -l /usr/local/pgsql/data/pg_tablespace
total 0
lrwxrwxrwx  1 postgres postgres 26 Feb 11 11:15 32788 ->
/usr/local/pgsql/tbl2
$

```

When a tablespace is created, sub-directory named "PG_{VERSION}_{YYYYMMDDN}" is created in the directory. "YYYYMMDDN" part is not tablespace creation date, but it seems to be the date for the format.



Example 35 The inside of the tablespace directory

```
$ ls -l /usr/local/pgsql/tbl2
total 4
drwx----- 2 postgres postgres 6 Feb 11 13:23 PG_9.6_201608131
$
```

□ Specific database

The database is granted an oid unique to the entire database cluster. This oid can be seen as oid pseudo column of pg_database catalog (or datid column of pg_stat_database catalog). Within the table space, directory with the same name as the oid of the database is created. You can also confirm the oid using utility oid2name.



Example 36 Identify database

```
postgres=> SELECT oid, datname FROM pg_database ;
  oid | datname
-----+-----
 13322 | postgres
 24577 | demodb
      1 | template1
 13321 | template0
(4 rows)

$ oid2name
All databases:
   Oid   Database Name   Tablespace
-----
 24577          demodb   pg_default
 13322          postgres pg_default
 13321      template0    pg_default
      1      template1    pg_default
      2

$ ls -l base
total 48
drwx----- 2 postgres postgres 8192 Feb 11 10:33 1
drwx----- 2 postgres postgres 8192 Feb 11 10:33 13321
drwx----- 2 postgres postgres 8192 Feb 11 12:25 13322
drwx----- 2 postgres postgres 8192 Feb 11 12:25 24577
$
```

□ Identify the file from the object name

In addition to searching for `pg_class` catalog, it is possible to identify the file name from the table name using `pg_relation_filepath` function. If specified the name of the TABLE, MATERIALIZED VIEW, and INDEX name to this function, it returns the relative path from the database cluster. If it uses a tablespace other than `pg_default`, it is displayed as stored below `pg_tblspc` directory, but it is actually a symbolic link destination file.



Example 37 Identify file from the object name

```
postgres=> CREATE TABLE data1(c1 NUMERIC, c2 CHAR(10)) ;
CREATE TABLE
postgres=> SELECT pg_relation_filepath('public.data1') ;
pg_relation_filepath
-----
base/16394/16447
(1 row)
postgres=> CREATE TABLE data2 (c1 NUMERIC, c2 CHAR(10)) TABLESPACE ts1 ;
CREATE TABLE
postgres=> SELECT pg_relation_filepath('public.data2') ;
pg_relation_filepath
-----
pg_tblspc/32985/PG_9.6_201608131/16385/32986
(1 row)
```

To retrieve only the file name, use the `pg_relation_filenode` function.

3.3 File system and behavior

3.3.1 Protection mode of the database cluster

The directory specified in the database cluster should be the mode 0700 to which only the administrator user can access. If the permissions are set for the group or external users, it is impossible to start the instance.

Example 38 Protection mode and instance startup

```
$ chmod g+r data
$ pg_ctl -D data start -w
server starting
FATAL: data directory "/usr/local/pgsql/data" has group or world access
DETAIL: Permissions should be u=rwx (0700).
$ echo $?
1
```

When initdb command is executed on the empty directory and the database cluster is created, the protection mode of the directory will be changed automatically. The protection mode of the directory where tablespace has been created will also be changed in the same way.



Example 39 Changes in protection mode

```
$ mkdir data1
$ ls -ld data1
drwxrwxr-x 2 postgres postgres Feb 11 12:59 10:27 data1
$ initdb data1
The files belonging to this database system will be owned by user "postgres".
.....
    pg_ctl -D data1 -l logfile start
$ ls -ld data1
drwx----- 14 postgres postgres 4096 Feb 11 12:59 data1
$
$ mkdir ts1
$ ls -ld ts1
drwxr-xr-x. 2 postgres postgres 4096 Feb 11 12:59 ts1
$ psql
postgres=# CREATE TABLESPACE ts1 LOCATION '/usr/local/pgsql/ts1' ;
CREATE TABLESPACE
postgres=# \q
$ ls -ld ts1
drwx-----. 3 postgres postgres 4096 Feb 11 12:59 ts1
```

The check on the protection mode of instance startup is not executed in a tablespace that has been created outside the database cluster. For this reason, modification of the protection mode does not result in an error.



Example 40 Change of protection mode for tablespace directory

```
postgres=# CREATE TABLESPACE ts1 LOCATION '/usr/local/pgsql/ts1' ;
CREATE TABLESPACE
postgres=# \q
$ ls -ld ts1
drwx-----. 3 postgres postgres 4096 Feb 11 12:59 ts1
$ chmod a+r ts1
$ ls -ld ts1
drwxr--r--. 3 postgres postgres 4096 Feb 11 12:59 ts1
$ pg_ctl -D data restart -m fast
waiting for server to shut down.... done
server stopped
server starting
```

3.3.2 Update File

In PostgreSQL, database objects for example TABLE, INDEX, and MATERIALIZED VIEW are created as separate files. Following is the I/O status for the file.

- Immediately after table creation

When a table is created, the corresponding file is created. The mapping of the tables and files can be found in relfilenode column of oid2name command or pg_class catalog.

Example 41 Table creation and file

```
postgres=> CREATE TABLE data1(c1 VARCHAR(10), c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT relfilenode FROM pg_class WHERE relname='data1' ;
relfilenode
-----
16446
(1 row)

$ cd data/base/16424/
$ ls -l 16446
-rw----- 1 postgres postgres 0 Feb 11 16:48 16446
```

At the time of table creation, it is an empty file of size 0. A tuple will be stored in this table. This behavior is also applied if the table is truncated with TRUNCATE statement.

Example 42 File before the checkpoint

```
postgres=> INSERT INTO data1 VALUES('ABC', '123') ;
INSERT 0 1
postgres=> \q
$ ls -l 16446
-rw----- 1 postgres postgres 0 Feb 11 16:53 16446
```

Since the checkpoint has not occurred, the size will not be expanded. When a forced checkpoint is performed, data is written to the file (write by writer process might be performed).

Example 43 File after the checkpoint

```
postgres=# CHECKPOINT ;
CHECKPOINT
postgres=# \q

$ ls -l 16446
-rw----- 1 postgres postgres 8192 Feb 11 16:54 16446
```

The file is written in 8 KB unit block size.

- Store and update the data

As PostgreSQL is a RDBMS of write-once, when it updates the tuple, old tuple is not changed and the tuple after the change will be added to the page.

Example 44 Execution of UPDATE statement and condition of the file

```
postgres=> UPDATE data1 SET c1='DEF', c2='456' ;
UPDATE 1
postgres=# CHECKPOINT ;
CHECKPOINT

$ od -a 16446
0000000 nul nul nul nul  P  bs stx dc2 soh nul nul nul  sp nul  @  us
0000020 nul  sp eot  sp  ff dc4 etx nul  `  us  @ nul  @  us  @ nul
0000040 nul nul nul nul nul nul nul nul nul nul nul nul nul nul nul
*
0017700  ff dc4 etx nul nul nul nul nul nul nul nul nul nul nul nul nul
0017720 stx nul stx nul stx  ( can nul  ht  D E F ht  4 5 6
0017740 vt dc4 etx nul  ff dc4 etx nul nul nul nul nul nul nul nul nul
0017760 stx nul stx  @ stx soh can nul  ht  A B C ht  1 2 3
0020000
```

From the command execution example, it is found that header is written and the first tuple is stored at the end of the block. It is found also that the updated tuple is added. From this verification, we can see that updating of tuples in the one block.

□ Multiple update

If the tuple is updated several times, the block becomes full. If there is reusable space in the same block, unnecessary space in the block is deleted, and the same page is used as far as possible (HHeap Only Tuples / HOT feature).

3.3.3 Visibility Map and Free Space Map

Tables and indexes are managed as a single file (or multiple files depending on the size). Visibility Map and Free Space Map are the files created with each object except for the files to store the data. Visibility Map and Free Space Map file are created only for TABLE, UNLOGGED TABLE and MATERIALIZED VIEW. On other persistent object (TOAST table, TOAST index, index, sequence etc.), they are not created.



□ Visibility Map

Visibility Map is a file that records the page where garbage tuple exists. It manages each page included in the table of the file as two bits. The name of the Visibility Map file is "{RELFILENODE}_vm". By referring to this file, PostgreSQL skips the pages, which have no garbage tuple on execution of VACUUM, and as a result, it can reduce the I/O load of VACUUM process. The initial size is 8 KB. After table creation, it is created at the first checkpoint or VACUUM time. In fact, the data, it is skipped only if the page does not exist unnecessary tuples (Visible) is continuous 32 or more. This value is fixed at SKIP_PAGES_THRESHOLD macro to the source code (src/backend/commands/vacuumlazy.c).

In order to view the contents of the Visibility Map, you can use the pg_visibility of Contrib module (from PostgreSQL 9.6).

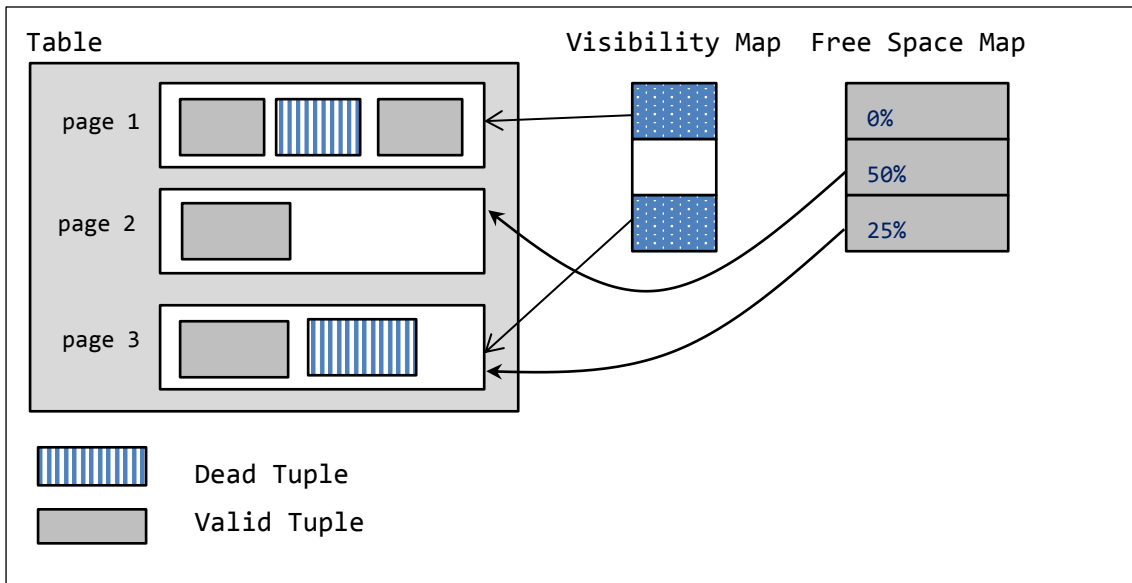
Example 45 Reference Visibility Map

```
postgres=# CREATE EXTENSION pg_visibility ;
CREATE EXTENSION
postgres=# SELECT pg_visibility_map('data1') ;
pg_visibility_map
-----
(0,t,f)
(1,t,f)
(2,t,f)
(3,t,f)
<<以下省略>>
```

□ Free Space Map

Free Space Map (FSM) is a file to manage the volume of the free space in each page of the table file. It manages each page included in the file of the table at a single byte. The name of the FSM file is "{RELFILENODE}_fsm". By referring to this file, it will be able to find the storage location of a tuple at a high speed. The initial size of this file is 24 KB. After table creation, it is created at the time of the first VACUUM execution. In addition, it will be updated every VACUUM execution. VACUUM performs processing while referring to the Visibility Map, and updates the Free Space Map.

Figure 8 Visibility Map and Free Space Map



Example 46 Visibility Map and Free Space Map

```
postgres=> SELECT relname, relfilenode FROM pg_class
            WHERE relname='data1' ;
relname | relfilenode
-----+-----
data1   |      16409
(1 row)

$ cd data/base/16385
$ ls 16409*
-rw----- 1 postgres postgres 8192 Feb 11 16:46 16409 -- Table Segment
-rw----- 1 postgres postgres 24576 Feb 11 16:46 16409_fsm -- Free Space Map
-rw----- 1 postgres postgres 8192 Feb 11 16:46 16409_vm -- Visibility Map
$
```

3.3.4 VACUUM behavior

Here we make sure how the contents of the file changes by VACUUM process. Confirmation of the operation is done with disabling automatic VACUUM (parameter autovacuum to "off").

□ VACUUM CONCURRENT

VACUUM CONCURRENT process marks the pre-update information to reusable state. It is verified

how the information block changes before and after the process.

Example 47 Insert data (12 tuples)

```
postgres=> CREATE TABLE data1 (c1 CHAR(500) NOT NULL, c2 CHAR(500) NOT NULL) ;
CREATE TABLE
postgres=> INSERT INTO data1 VALUES ('AAA', '111') ;
postgres=> INSERT INTO data1 VALUES ('BBB', '222') ;
postgres=> INSERT INTO data1 VALUES ('CCC', '333') ;
postgres=> INSERT INTO data1 VALUES ('DDD', '444') ;
postgres=> INSERT INTO data1 VALUES ('EEE', '555') ;
postgres=> INSERT INTO data1 VALUES ('FFF', '666') ;
postgres=> INSERT INTO data1 VALUES ('GGG', '777') ;
postgres=> INSERT INTO data1 VALUES ('HHH', '888') ;
postgres=> INSERT INTO data1 VALUES ('III', '999') ;
postgres=> INSERT INTO data1 VALUES ('JJJ', '000') ;
postgres=> INSERT INTO data1 VALUES ('AAA', 'aaa') ;
postgres=> INSERT INTO data1 VALUES ('AAA', 'bbb') ;
INSERT 0 1
postgres=# CHECKPOINT ;
CHECKPOINT
```

Create a table with 1KB tuple size, and stores the 12 tuples in it. As a result, two blocks table is created.

Example 48 Prepare data (Identify file)

```
$ oid2name -d datadb1
From database "datadb1":
  Filenode  Table Name
-----
      16470      data1

$ cd /usr/local/pgsql/data/base/16424
$ ls -l 16470
-rw----- 1 postgres postgres 16384 Feb 11 10:56 16470
```



Example 49 Initial state (First block)

```
0000000 nul nul nul nul dle U stx nak soh nul nul nul 4 nul H etx
* ...
0001740 ` bel nul nul G G G sp sp sp sp sp sp sp sp
* ...
0002720 sp sp sp sp sp sp sp sp ` bel nul nul 7 7 7 sp
* ...
0003740 ack nul stx nul stx bs can nul ` bel nul nul F F F sp
* ...
0004740 ` bel nul nul 6 6 6 sp sp sp sp sp sp sp sp
* ...
0005760 ` bel nul nul E E E sp sp sp sp sp sp sp sp
* ...
0006740 sp sp sp sp sp sp sp sp ` bel nul nul 5 5 5 sp
* ...
0007760 eot nul stx nul stx bs can nul ` bel nul nul D D D sp
* ...
0010760 ` bel nul nul 4 4 4 sp sp sp sp sp sp sp sp
* ...
0012000 ` bel nul nul C C C sp sp sp sp sp sp sp sp
* ...
0012760 sp sp sp sp sp sp sp sp ` bel nul nul 3 3 3 sp
* ...
0014000 stx nul stx nul stx bs can nul ` bel nul nul B B B sp
* ...
0015000 ` bel nul nul 2 2 2 sp sp sp sp sp sp sp sp
* ...
0016020 ` bel nul nul A A A sp sp sp sp sp sp sp sp
* ...
0017000 sp sp sp sp sp sp sp sp ` bel nul nul 1 1 1 sp
*
```

Example 50 Initial state (2nd block)

```

0020000 nul nul nul nul nul k stx nak soh nul nul nul , nul X vt
0020020 nul sp eot sp nul nul nul nul x esc dle bs p etb dle bs
0020040 h dc3 dle bs ` si dle bs X vt dle bs nul nul nul nul
0020060 nul nul nul nul nul nul nul nul nul nul nul nul nul nul
*
0025720 nul nul nul nul nul nul nul nul 6 dc4 etx nul nul nul nul
* ...
0025760 ` bel nul nul L L L sp sp sp sp sp sp sp sp
* ...
0026740 sp sp sp sp sp sp sp sp ` bel nul nul b b b sp
* ...
0027760 eot nul stx nul stx bs can nul ` bel nul nul K K K sp
* ...
0030760 ` bel nul nul a a a sp sp sp sp sp sp sp sp
* ...
0032000 ` bel nul nul J J J sp sp sp sp sp sp sp sp
* ...
0032760 sp sp sp sp sp sp sp sp ` bel nul nul 0 0 0 sp
* ...
0034000 stx nul stx nul stx bs can nul ` bel nul nul I I I sp
* ...
0035000 ` bel nul nul 9 9 9 sp sp sp sp sp sp sp sp
* ...
0036020 ` bel nul nul H H H sp sp sp sp sp sp sp sp
* ...
0037000 sp sp sp sp sp sp sp sp ` bel nul nul 8 8 8 sp
0037020 sp sp sp sp sp sp sp sp sp sp sp sp sp sp sp
*
0040000

```

Delete the middle tuple of each block, and then run the VACUUM process.



Example 51 Delete tuples and execute VACUUM

```
postgres=> DELETE FROM data1 WHERE c1 IN ('CCC', 'JJJ') ;  
DELETE 2  
postgres=# CHECKPOINT ;  
CHECKPOINT  
postgres=> VACUUM data1 ;  
VACUUM
```

By this operation, we can see how the block contents are changed. The following two cases show the block state after VACUUM. In the block, valid tuple is moved to the bottom of the block, and free space is created between the header and the bottom of the block. By this operation, it is verified that a contiguous free space is created. However, part of the tuple (001740 C1 = 'GGG', the tuple of C2 = '777', 003740 C1 = 'LLL', C2 = tuple of 'bbb' in the following example) is stored in duplicate.

Notice

Tuple organization in the block has been implemented in a function of HOT (Heat On Tuples). If there is no sufficient free space in the page, a work correspond to VACUUM is executed in the block. This behavior is described in the following pages (in Japanese).

http://lets.postgresql.jp/documents/tutorial/hot_2/hot2_2



Example 52 After VACUUM operation (First block)

```

0000000 nul nul nul nul  sp  7 stx nak soh nul soh nul  4 nul  P bel
*...
0001740  ` bel nul nul  G  G  G sp  sp  sp  sp  sp  sp  sp  sp
*...
0002720  sp  sp  sp  sp  sp  sp  sp  sp  ` bel nul nul  7  7  7 sp
*...
0003740 bel nul stx nul stx  ht can nul  ` bel nul nul  G  G  G sp
*...
0004740  ` bel nul nul  7  7  7 sp  sp  sp  sp  sp  sp  sp  sp
*...
0005760  ` bel nul nul  F  F  F sp  sp  sp  sp  sp  sp  sp  sp
*...
0006740  sp  sp  sp  sp  sp  sp  sp  sp  ` bel nul nul  6  6  6 sp
*...
0007760 enq nul stx nul stx  ht can nul  ` bel nul nul  E  E  E sp
*...
0010760  ` bel nul nul  5  5  5 sp  sp  sp  sp  sp  sp  sp  sp
*...
0012000  ` bel nul nul  D  D  D sp  sp  sp  sp  sp  sp  sp  sp
*...
0012760  sp  sp  sp  sp  sp  sp  sp  sp  ` bel nul nul  4  4  4 sp
*...
0014000 stx nul stx nul stx  ht can nul  ` bel nul nul  B  B  B sp
*...
0015000  ` bel nul nul  2  2  2 sp  sp  sp  sp  sp  sp  sp  sp
*...
0016020  ` bel nul nul  A  A  A sp  sp  sp  sp  sp  sp  sp  sp
*...
0017000  sp  sp  sp  sp  sp  sp  sp  sp  ` bel nul nul  1  1  1 sp
*

```



Example 53 After VACUUM operation (2nd block)

```

0020000 nul nul nul nul dle  H stx nak soh nul soh nul  , nul  ` si
*...
0025760 ` bel nul nul  L  L  L sp sp sp sp sp sp sp sp sp
*...
0026740 sp sp sp sp sp sp sp sp ` bel nul nul  b  b  b sp
*...
0027760 enq nul stx nul stx  ht can nul  ` bel nul nul  L  L  L sp
*...
0030760 ` bel nul nul  b  b  b sp sp sp sp sp sp sp sp sp
*...
0032000 ` bel nul nul  K  K  K sp sp sp sp sp sp sp sp sp
*...
0032760 sp sp sp sp sp sp sp sp ` bel nul nul  a  a  a sp
*...
0034000 stx nul stx nul stx  ht can nul  ` bel nul nul  I  I  I sp
*...
0035000 ` bel nul nul  9  9  9 sp sp sp sp sp sp sp sp sp
*...
0036020 ` bel nul nul  H  H  H sp sp sp sp sp sp sp sp sp
*...
0037000 sp sp sp sp sp sp sp sp ` bel nul nul  8  8  8 sp
*
0040000

```

□ Space usage after VACUUM

As free space is created by VACUUM process, PostgreSQL stores the new data.

Example 54 Insert New Data

```

postgres=> INSERT INTO data1 VALUES ('MMM', 'ccc') ;
INSERT 0 1
postgres=# CHECKPOINT ;
CHECKPOINT

```

Example 55 After INSERT (First block)

```

0000000 nul nul nul nul ` t stx nak soh nul soh nul 4 nul H etx
* ...
0001740 ` bel nul nul M M M sp sp sp sp sp sp sp sp sp
* ...
0002720 sp sp sp sp sp sp sp sp ` bel nul nul c c c sp
* ...
0003740 bel nul stx nul stx ht can nul ` bel nul nul G G G sp
* ...
0004740 ` bel nul nul 7 7 7 sp sp sp sp sp sp sp sp sp
* ...

```

As described above, it is found that 0,001,740 part, that has been stored redundantly, was overwritten. This behavior might be different by FILLFACTOR attributes of the table.

□ VACUUM FULL

VACUUM FULL performs, not only the re-usage of the updated tuple, but also reduction of the file. Here, we can see how the actual file changes.

When VACUUM FULL statement is executed, the i-node of the file and the file name are changed; this means that the new file is created. By this behavior, it is shown that VACUUM FULL reduces the file by reading the existing file and creating a new file with organizing tuples.

Example 56 VACUUM operation (Execute VACUUM FULL)

```

$ ls -li 16470 -- File before VACUUM FULL
558969 -rw----- 1 postgres postgres 16384 Feb 11 11:39 16470

$ oid2name -d datadb1
From database "datadb1":
  Filenode  Table Name
-----
    16476      data1      -- Changed Filenode due to VACUUM FULL

$ ls -li 16476 -- Changed file name and i-node
558974 -rw----- 1 postgres postgres 16384 Feb 11 11:47 16476

```




When executing VACUUM FULL on the database that contains multiple tables, the VACUUM process executed at the same time is only one table (CONCURRENT VACUUM as well). When VACUUM FULL is executed to the table consists of multiple segments, all of the files that compose the table are maintained until the completion of the VACUUM FULL to all files. Therefore, in case of the large-scale table (consists of many segments), there is a possibility that the storage capacity of the table becomes twice at the maximum.

□ Bulk update and partial update

Considering the table contains 1,000 tuples, the number of the unnecessary tuples processed by VACUUM is different according to the way of UPDATE execution: execute UPDATE statement to all 1,000 tuples collectively, or execute UPDATE 1,000 times to each tuple. The reason is that the copies of all the tuples are created in case of bulk update, whereas the unnecessary tuples in the block are re-used before executing VACUUM in case of single tuple update.

Example 57 The difference in the number of blocks by the update method

```

postgres=> CREATE TABLE data1(c1 NUMERIC, c2 VARCHAR(100), c3 VARCHAR(100)) ;
CREATE TABLE
-- insert 1000 tuples
postgres=> SELECT relpages, reltuples FROM pg_class WHERE relname='data1' ;
relpages | reltuples
-----+-----
      8 |      1000
(1 row)
postgres=> UPDATE data1 SET c1=c1+1 ; -- Bulk Update
UPDATE 1000
postgres=> SELECT relpages, reltuples FROM pg_class where relname='data1' ;
relpages | reltuples
-----+-----
     15 |      1000 -- Increase blocks
(1 row)
postgres=> TRUNCATE TABLE data1 ;
-- insert 1000 tuples
postgres=> UPDATE data1 SET C2='TEST' WHERE c1=100 ; -- 1,000 times Update
UPDATE 1
postgres=> SELECT relpages, reltuples FROM pg_class WHERE relname='data1' ;
relpages | reltuples
-----+-----
      8 |      1000 -- Remain block number
(1 row)

```

3.3.5 Opened Files

The file, which is opened by PostgreSQL in the instance, is investigated. Using lsof command of the Linux is the relation between a process and the open file is checked.

□ Immediately after the instance start

Immediately after instance startup, logger process opens a log file, and autovacuum launcher process opens files, which correspond to pg_database catalog.

Table 31 Opened Files

Process	Object / File
postmaster	/tmp/.s.PGSQL.{PORT}
logger process ⁷	{PGDATA}/pg_log/postgresql-{DATE}.log
autovacuum launcher	pg_database

□ Just after user connection

When a client connects, the backend process (postgres) opens pg_am catalog. It is unclear whether the user's connection leads to, but a checkpoint process opens a current WAL file.

⁷ When the parameters logging_collector set to 'on'

Table 32 Additional open files

Process	Object / File
postgres	pg_authid
postgres	pg_class
postgres	pg_attribute
postgres	pg_index
postgres	pg_am
postgres	pg_opclass
postgres	pg_amproc
postgres	pg_opclass_oid_index
postgres	pg_amproc_fam_proc_index
postgres	pg_class_oid_index
postgres	pg_attribute_relid_attnum_index
postgres	pg_index_indexrelid_index
postgres	pg_database_oid_index
postgres	pg_db_role_setting_databaseid_rol_index

- Just after the update transaction

When the update transaction occurs, the backend process opens not only objects to be updated, but also the current of WAL.

Table 33 Additional open files

Process	Object / File
postgres	pg_xlog/{WALFILE}
postgres	Updated object file

- Just after the user disconnect

When the user disconnects, because of the backend process stop, the opened file returns to original.

Table 34 Opened files

Process	Object / File
autovacuum launcher	pg_database
logger process	{PGDATA}/pg_log/postgresql- {DATE}.log

An above-mentioned experiment shows that PostgreSQL closes many files in the time when they become unnecessary. A log file and a WAL file are also opened at the time when it is necessary, and closed when it becomes unnecessary.

3.3.6 Behavior of process (Writing WAL data)

When a transaction commits, renewal information is written to a WAL file. WAL information is wrote by wal writer process or a postgres process. In some typical documents, it is said that only wal writer process writes WAL, but actually, postgres process also WAL. The selection method of the process which writes WAL, etc., validated adequately.

- In case of parameter `synchronous_commit` set to "on"

The following example, outputs the system call of postgres process when it issues the INSERT statement after creating a table for instance with parameter `synchronous_commit` set to "on" (default value). The postgres process writes of WAL.

Example 58 System call postgres process to be executed

```

1: recvfrom(10, "Q\\0\\0\\0,INSERT INTO data1 values (1"... , 8192, 0, NULL, NULL) = 45
2: open("base/16394/91338_fsm", O_RDWR)      = -1
3: open("base/16394/91338", O_RDWR)          = 16
4: lseek(16, 0, SEEK_END)                     = 0
5: lseek(16, 0, SEEK_END)                     = 0
6: kill(7487, SIGUSR1)                        = 0
7: write(16, "\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0..." , 8192) = 8192
8: open("pg_xlog/000000010000000D000000DA", O_RDWR) = 17
9: lseek(17, 6381568, SEEK_SET)               = 6381568
10: write(17, "u\\32\\5\\0\\1\\0\\0\\0\\0`a\\332\\r\\0\\0005\\4\\0\\0\\0\\0\\0\\0..." , 8192) = 8192
11: fdatasync(17)                             = 0
12: sendto(9, "\\2\\0\\0\\0\\230\\2\\0\\0\\n@\\0\\0\\6\\0\\0\\0\\0\\0\\0\\0..." , 664, 0, NULL, 0) = 664
13: sendto(10, "C\\0\\0\\0\\17INSERT 0 1\\0Z\\0\\0\\0\\5I", 22, 0, NULL, 0) = 22

```



Table 35 Executed System call

Line#	Operation
1	Receive INSERT statement from remote host
2	Access file for table (check fsm file)
3	Access file for table (open data file)
4	Scan file for table
5	Scan file for table
6	Send signal to process #7487 (writer process)
7	Initialize file for table (initialize page)
8	Open WAL file
9	Seek WAL file
10	Write WAL file
11	Sync WAL file
12	Send result to UDP network
13	Send result to TCP session

□ In case of parameter `synchronous_commit` set to "off"

When parameter `synchronous_commit` was set to "off", wal writer process started to write WAL. A postgres process does not access to WAL file. After reading a data file, postgres process sends SIGUSR1 signal to wal writer process (process ID 7635).



Example 59 Major system calls by postgres process executed

```
recvfrom(10, "Q\0\0\0.insert into data1 values (1"... , 8192, 0, NULL, NULL) = 47
open("base/16499/16519_fsm", O_RDWR)      = 34
lseek(34, 0, SEEK_END)                     = 40960
lseek(34, 0, SEEK_SET)                     = 0
read(34, "\0\0\001!\312\0\0\0\0\30\0\0 \0 \4 \0\0\0\0\350\350\0\350"... , 8192) =
8192
open("base/16499/16519", O_RDWR)          = 35
lseek(35, 44269568, SEEK_SET)              = 44269568
read(35, "\1\0\0\0\260774\2P\f\0 \4 \0\0\0\0\330\237D\0\260\237D\0"... , 8192) = 8192
kill(7635, SIGUSR1)                        = 0
sendto(9, "\2\0\0\0\320\3\0\0s@\0\0\t\0\0\0\0\0\0\0\0\0\0"... , 976, 0, NULL, 0) =
976
sendto(9, "\2\0\0\0\320\3\0\0s@\0\0\0\0\0\0\0\0\0\0\0\0"... , 976, 0, NULL, 0) =
976
sendto(9, "\2\0\0\0000\2\0\0s@\0\0\5\0\0\0\0\0\0\0\0\0\0\0"... , 560, 0, NULL, 0) =
560
sendto(10, "C\0\0\0\17INSERT 0 1\0Z\0\0\0\5I", 22, 0, NULL, 0) = 22
```

Example 60 Major system calls by wal writer process executed

```
--- SIGUSR1 (User defined signal 1) @ 0 (0) ---
write(4, "\0", 1)                          = 1
rt_sigreturn(0x4)                          = -1 EINTR (Interrupted system call)
read(3, "\0", 16)                          = 1
open("pg_xlog/000000010000000100000017", O_RDWR) = 5
write(5, "} \320\6\0\1\0\27\1\0\0\0\0\0\0\0\0\0\0\0L<\302x\322cuS"... , 8192) = 8192
fdatsync(5)
```

Wal writer process receives a SIGUSR1 signal, performs processing of the pipe, and writes to the WAL files.

□ Large-scale update transaction

When renewal data cannot be stored in a WAL buffer, the WAL file is updated before transaction is fixed. In this case, postgres process and wal writer process both update WAL file with communicating each other.

3.3.7 Behavior of process (Writing by checkpoint)

A checkpointer process is literally the process, which executes a checkpoint. When a checkpoint occurs, checkpointer process coincide the contents between a storage and shared buffer due to write a dirty buffer to a data file. The following example is a trace of the system calls, which are executed by checkpointer process when CHECKPOINT statement is executed.

Example 61 The writing process by checkpointer (part)

```

open("pg_clog/0000", O_RDWR|O_CREAT, 0600) = 5
lseek(5, 0, SEEK_SET) = 0
write(5, "@UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU...", 8192) = 8192
fsync(5) = 0
close(5) = 0
open("base/16499/24980_vm", O_RDWR) = 5
write(5, "\\0\\0\\0\\0\\210\\253h\\262\\0\\0\\0\\0\\0\\0\\0\\0\\200\\365?\\0"... , 8192) = 8192
open("base/16499/24980", O_RDWR) = 7 -- open data file
lseek(7, 442368, SEEK_SET) = 442368
write(7, "\\0\\0\\0\\3r\\262\\0\\0\\1\\0\\f\\4\\200\\16\\0 \\4 -\\10\\0:\\0\\1\\0"... , 8192) = 8192
--- Repeat lseek / write ---
open("base/16499/12725", O_RDWR) = 10
write(10, "\\0\\0\\0\\0\\30\\1\\0\\220\\0\\270\\17\\0 \\4 \\0\\0\\0\\00\\231|\\3"... , 8192) = 8192
fsync(7) = 0
fsync(8) = 0
fsync(5) = 0
fsync(10) = 0
lseek(14, 7528448, SEEK_SET) = 7528448
write(14, "}\\320\\5\\0\\1\\0\\0\\0\\0\\340r\\262\\0\\0\\0\\0&\\0\\0\\0\\0\\0\\0"... , 8192) = 8192
fdatasync(14) = 0
open("global/pg_control", O_RDWR) = 11
write(11, 0x\\322cuS\\251\\3\\0\\0\\2672\\1\\f\\6\\0\\0\\311\\237S\\0\\0\\0\\0"... , 240) = 240
fsync(11) = 0
close(11) = 0

```

Checkpointeer process writes the checkpoint information in the `pg_clog` directory, and then updates the `vm` (Visibility Map) file. After that, data file is updated block by block, and finally information of the checkpoint completion is written to `pg_control` file.

3.3.8 Behavior of process (Writing by writer)

While checkpoint process is writing with a relatively long period, writer process is writing changed pages (dirty buffers) little by little in a short period. It can prevent the peak of the I/O due to the occurrence of checkpoint by writing of writer process. Writing interval of the writer process is determined by the parameter `bgwriter_delay` (default: 200ms). Latency time is waited in the Linux / UNIX platforms by "select (2)" system call, and in the Windows environment by Windows API `WaitForMultipleObjects` (`WaitLatch` function in `src/backend/port/win32_latch.c` or `src/backend/port/pg_latch.c`).

The maximum value of the number of write block is determined by the parameter `bgwriter_lru_maxpages`. The default value is 100. The actual number of write blocks is calculated by multiplying the number of blocks that have been recently requested and the value of the parameter `bgwriter_lru_multiplier`. Even in that case, this number is not exceeded to the parameter `bgwriter_lru_maxpages`. The actual writing is done only when the number of the necessary pages, estimated by parameter `bgwriter_lru_maxpages` and the average number of the pages recently needed, is larger than the number of the reusable pages.

Table 36 Parameters related to the writer process

Parameter	Description	Default Value
<code>bgwriter_delay</code>	Writing interval of writer process	200ms
<code>bgwriter_lru_maxpages</code>	Maximum number of writing pages; in case of 0, the write is not executed	100
<code>bgwriter_lru_multiplier</code>	Value to be applied to the average requested pages	2.0

3.3.9 Behavior of process (archiver)

Archiver process performs the archive of WAL files that has been completely written. It does the archiving process with the following timing:

- Regularly at 60-second intervals
- Receive `SIGUSR1` signal

In fact, it performs the following processing.

- Search the directory `{PGDATA}/pg_xlog/archive_status`
- Find `{WALFILE}.ready` file
- Run the specified command in the parameter `archive_command` using the "system" function
- Check the "system" function status
- Rename the file from `{PGDATA}/pg_xlog/archive_status/{WALFILE}.ready` to `{PGDATA}/pg_xlog/archive_status/{WALFILE}.done`

Example 62 Major system call archiver process to be executed

```

--- SIGUSR1 (User defined signal 1) @ 0 (0) ---
rt_sigreturn(0x4)          = -1 EINTR (Interrupted system call)
open("pg_xlog/archive_status", O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = 5
clone(child_stack=0,                               flags=CLONE_PARENT_SETTID|SIGCHLD,
parent_tidptr=0x7fffcc9877d8) = 5119
wait4(5119, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 5119
--- SIGCHLD (Child exited) @ 0 (0) ---
rename("pg_xlog/archive_status/000000010000000000000003B.ready",
"pg_xlog/archive_status/000000010000000000000003B.done") = 0
sendto(9, "\v\0\0\0@\0\0\0\000000000100000000000003"..., 64, 0, NULL, 0) = 64

```

In order to get the execution parameters of parameter `archive_command`, specify `DEBUG3` on the parameter `log_min_messages`. The following log is outputted.

Example 63 Log for command with `archive_command` parameter

```

DEBUG:          executing          archive          command          "/bin/cp
pg_xlog/0000000100000000000000071 /arch/0000000100000000000000071"

```

In order to output a log at the time when the archive target of WAL has been determined, specify `DEBUG1` on the parameter `log_min_messages`. The following log is outputted.

Example 64 Log for archived WAL file

```

DEBUG: archived transaction log file "0000000100000000000000071"

```

3.4 Online Backup

3.4.1 Behavior of online backup

Online backup is a way to get a backup under instance startup condition. To do the online backup, execute `pg_start_backup` function for the instance, and back up all the files of the database cluster. When the backup is complete, execute `pg_stop_backup` function. These operations can be done automatically by `pg_basebackup` command. Online backup must get the entire backup of the database cluster (and the external table space). For the per-database backup, the logical backup such as `pg_dump` command must be used.

□ `Pg_start_backup` function

This function declares the start of online backup. When this function is executed, WAL offset value at the time of backup start appears. The label file "`{PGDATA}/backup_label`" is also created. In the label file, start time and WAL of information backup are listed.

Example 65 The start of the online backup

```
postgres=# SELECT pg_start_backup(now()::text) ;
pg_start_backup
-----
0/59000028
(1 row)
postgres=#
```

Example 66 Backup_label file at runtime of `pg_start_backup` function

```
$ cat data/backup_label
START WAL LOCATION: 0/6000028 (file 00000001000000000000000006)
CHECKPOINT LOCATION: 0/6000060
BACKUP METHOD: pg_start_backup
BACKUP FROM: master
START TIME: 2017-02-11 12:47:42 JST
LABEL: 2017-02-11 12:47:42.2466+09
```

□ `Pg_stop_backup` function

This function declares the completion of the online backup. When this function is executed, WAL offset value at the time of the end of the backup will be displayed. Label file

"{PGDATA}/backup_label" created during the execution of pg_start_backup function is deleted, and the new label file is created in an archiving log directory.

Example 67 End of online backup

```
postgres=# SELECT pg_stop_backup() ;
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
 pg_stop_backup
-----
 0/5B0000B8
(1 row)
```

When online backup is processed with parameter archive_mode set to "off", following warning message will display on the execution of pg_stop_backup function.

Example 68 Finish on line backup (archive_mode="off")

```
postgres=# SELECT pg_stop_backup() ;
NOTICE:  WAL archiving is not enabled; you must ensure that all required
WAL segments are copied through other means to complete the backup
 pg_stop_backup
-----
 0/590000B8
(1 row)
```

☐ Non-Exclusive mode

The pg_start_backup function and the pg_stop_backup function, parameters for the exclusive control "exclusive" (boolean) has been added in PostgreSQL 9.6. The default value is "true", the behavior is the same as previous versions. It does not create the backup_label files and tablespace_map file if you specify the "exclusive" parameter to "false". Pg_stop_backup function you must specify the same mode as the pg_start_backup function. When you run the pg_stop_backup function with Non-Exclusive mode, the output result will change with the Exclusive mode.

Example 69 Online backup by the Non-Exclusive mode

```
postgres=# SELECT pg_start_backup(now()::text, false, false) ;
pg_start_backup
-----
0/8000028
(1 row)

postgres=# SELECT pg_stop_backup() ;
ERROR:  non-exclusive backup in progress
HINT:  Did you mean to use pg_stop_backup('f')?
postgres=#
postgres=# SELECT pg_stop_backup(false) ;
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
pg_stop_backup
-----
(0/8000130,"START WAL LOCATION: 0/8000028 (file 000000010000000000000008)+
CHECKPOINT LOCATION: 0/8000060                                     +
BACKUP METHOD: streamed                                           +
BACKUP FROM: master                                              +
START TIME: 2017-02-11 12:50:16 JST                               +
LABEL: 2017-02-11 12:50:15.900273+09                             +
","16384 /home/postgres/ts1                                       +
")
(1 row)
```

3.4.2 Backup Label File

Backup label file is a text file in which information of online backup is stored. Executing `pg_start_backup` function, it will be created as a "{PGDATA}/backup_label" file. Running `pg_stop_backup` function, backup_label file is deleted after re-read the contents, and it will be created in the following filename format in the same directory as the archive log.

Syntax 2 Filename format of Backup Label File

```
{WALFILE}. {WALOFFSET}.backup
```

If there remains backup_label file at instance stop, it will be renamed to backup_label.old file in the same directory. This process will also be executed if "pg_ctl stop -m immediate" command is executed. Even if backup_label file still remains at instance startup, the file is renamed to backup_label.old and then instance is started. Nothing is outputted to the log. Backup Label File is in text format, and information about online backup are listed in it.

Table 37 Contents of Backup Label File

Line#	Output contents	Description	Note
1	START WAL LOCATION:	WAL location of start backup	
2	STOP WAL LOCATION:	WAL location of finish backup	
3	CHECKPOINT LOCATION:	Checkpoint information	
4	BACKUP METHOD:	Backup method	
5	BACKUP FROM:	Backup source	
6	START TIME:	Backup start date/time	
7	LABEL:	Backup label	Specify by pg_start_backup function
8	STOP TIME:	Stop time	Added by pg_stop_backup function

STOP TIME, in the last line, will be output only when pg_stop_backup function is executed.

Example 70 Backup Label File

```
START WAL LOCATION: 0/8000028 (file 000000010000000000000008)
STOP WAL LOCATION: 0/8000130 (file 000000010000000000000008)
CHECKPOINT LOCATION: 0/8000060
BACKUP METHOD: streamed
BACKUP FROM: master
START TIME: 2017-02-11 12:50:16 JST
LABEL: 2017-02-11 12:50:15.900273+09
STOP TIME: 2017-02-11 12:50:36 JST
```

3.4.3 Online Backup with Replication Environment

Online backup function can be executed only at the master instance. When pg_start_backup function is executed at a slave instance, "ERROR: recovery is in progress" error will occur.

Example 71 Start online backup at the slave instance

```
postgres=# SELECT pg_start_backup(now()::text) ;
ERROR:  recovery is in progress
HINT:  WAL control functions cannot be executed during recovery.
postgres=#
```

When pg_is_in_backup function is executed at a slave instance while online backup is running on the master instance, "f" (not in online backup) is returned.

Example 72 Check online backup status at the slave instance

```
postgres=# SELECT pg_is_in_backup() ;
 pg_is_in_backup
-----
_f
(1 row)
```

3.4.4 Instance shutdown with online backup

During the online backup, instance stop specified the smart mode would fail. Even if the stop operation fails, general users cannot connect for instance because the instance becomes the status in shutdown. To force to stop, the instance during online backup, specify the fast mode to pg_ctl command.



Example 73 Shutdown during online backup

```
postgres=# SELECT pg_start_backup(now()::text) ;
pg_start_backup
-----
0/A5000028
(1 row)
postgres=# \q
$
$ pg_ctl -D data stop -m smart
WARNING: online backup mode is active
Shutdown will not complete until pg_stop_backup() is called.

waiting          for          server          to          shut
down..... failed
pg_ctl: server does not shut down
HINT: The "-m fast" option immediately disconnects sessions rather than
waiting for session-initiated disconnection.
$ echo $?
1
$ pg_ctl -D data stop -m fast
waiting for server to shut down....
done
server stopped
$
```


3.5 File Format

3.5.1 Postmaster.pid

Postmaster.pid file is a text file that is created within the database cluster. It is created at instance startup, and removed at the time of instance normal shutdown. The manual says that pg_ctl command confirms the operation of the instance by the presence of the file, but in fact using the process ID written in a first line of the file. The information written to the postmaster.pid has been defined as row number of "LOCK_FILE_LINE_*" macro in the source code (src/include/miscadmin.h).

Table 38 The contents of postmaster.pid file

Line#	Description	Note
1	postmaster process ID	decimal
2	The path of the database cluster directory	
3	Instance start time	
4	IPv4 connection waiting port number	
5	Socket creation directory for local connections	
6	IPv4 connection waiting address	
7	The key of System V shared memory, ID information	decimal

If the process with an ID written in the first line of the file does not exist, pg_ctl status command or pg_ctl stop command does not work.

Example 74 Contents of postmaster.pid file

```
$ cat data/postmaster.pid
15141
/home/postgres/data
1475205932
5432
/tmp
*
5432001    196608
$
```



□ Parameter `external_pid_file`

If you specify the file name (or the path including the directory name) to the parameter `external_pid_file`, a file is created to write the ID of the postmaster process in it, in addition to the `postmaster.pid` file. However, the information output to the file specified in the `external_pid_file` is only the process ID of the postmaster. The `pg_ctl` command does not refer to the `external_pid_file`. While `postmaster.pid` file is created with protected mode `"-rw-----"`, the file specified in the `external_pid_file` is created with the protection mode `"-rw-r--"`, so it can be referred by the user other than PostgreSQL administrator.

The startup process will be done successfully even if the file specified in the parameter `external_pid_file` cannot be created at instance startup.

3.5.2 Postmaster.opts

`Postmaster.opts` file holds the parameters at the time of start-up postmaster process, and it will be created at instance startup. This file is not deleted even when the instance stops. If there is no write access privilege to this file at instance startup, instance startup will result in an error. In this file, the parameters that were specified in the `pg_ctl` start command are output.

Example 75 Contents of `postmaster.opts` file

```
$ pg_ctl start -D data
$ cat data/postmaster.opts
/usr/local/pgsql/bin/postgres "-D" "data"
$ pg_ctl stop
$ pg_ctl start
$ cat data/postmaster.opts
/usr/local/pgsql/bin/postgres
$
```

3.5.3 PG_VERSION

`PG_VERSION` files are text files that are created automatically in the database cluster and database oid directory in the tablespace. Basically the major version of PostgreSQL is listed. This file is a simple text file, but if the file in the database cluster is lost, then instance cannot startup, and a client cannot connect to the database if file in the database oid directory is lost.



Example 76 Contents of PG_VERSION file

```
$ cd /usr/local/pgsql/data
$ cat PG_VERSION
9.6
$
```

3.5.4 Pg_control

Pg_control file is a small binary file that is stored in the {PGDATA}/global directory. The size of this file is 8 KB (defined as PG_CONTROL_SIZE in src/include/catalog/within_pg_control.h). The data actually written is defined in the structure ControlFileData (src/include/catalog/pg_control.h).

□ Contents of pg_control file

The main contents of pg_control file can be confirmed by pg_controldata command or dedicated functions.

Example 77 pg_controldata command execution

```
$ pg_controldata data
pg_control version number:          960
Catalog version number:            201608131
Database system identifier:         6335932445819631823
Database cluster state:             in production
pg_control last modified:           Fri Feb 11 12:55:16 2017
Latest checkpoint location:         0/9000098
Prior checkpoint location:          0/8000060
Latest checkpoint's REDO location:  0/9000060
Latest checkpoint's REDO WAL file:  00000001000000000000000009
Latest checkpoint's TimeLineID:     1
.....
Latest checkpoint's newestCommitTsXid:0
Time of latest checkpoint:          Fri Feb 11 12:55:16 2017
Fake LSN counter for unlogged rels: 0/1
Minimum recovery ending location:    0/0
Min recovery ending loc's timeline:  0
Backup start location:               0/0
Backup end location:                 0/0
End-of-backup record required:       no
.....
Blocks per segment of large relation: 131072
WAL block size:                     8192
Bytes per WAL segment:               16777216
Maximum length of identifiers:       64
Maximum columns in an index:         32
Maximum size of a TOAST chunk:       1996
Size of a large-object chunk:        2048
Date/time type storage:               64-bit integers
Float4 argument passing:              by value
Float8 argument passing:              by value
Data page checksum version:          0
```

By the execution of this command, fixed information such as version information and the database ID, the last update time, the state of the instance, checkpoint information, and the information at the

compilation are output. As can be seen from the output result of `pg_controldata` command, `pg_control` information is updated at the time of the checkpoint and the instance status change. Information of `pg_control` file can also be acquired by using the following functions. These functions have been added in PostgreSQL 9.6.

Table 39 Functions for `pg_control` information

Function name	Description
<code>pg_control_checkpoint</code>	Execution status of the checkpoint
<code>pg_control_init</code>	Information of the determined various limits at compile time
<code>pg_control_recovery</code>	Backup / recovery information
<code>pg_control_system</code>	Version information, system ID information

Example 78 Execute `pg_control_system` function

```
postgres=> \x
Expanded display is on.
postgres=> SELECT * FROM pg_control_system() ;
-[ RECORD 1 ]-----+-----
pg_control_version      | 960
catalog_version_no      | 201608131
system_identifier       | 6335932445819631823
pg_control_last_modified | 2017-02-11 13:00:16+09
```

□ Database cluster state

In "Database cluster state", which is execution result of `pg_controldata` command, the state of the database cluster recognized by the current `pg_control` file is output.

Table 40 Database cluster state

File's value	Output message	Description
0	starting up	Instance is starting up
1	shut down	Instance was shut down normally
2	shut down in recovery	Instance is shut down in recovery
3	shutting down	Instance is on shutting down
4	in crash recovery	Instance is in crash recovery state
5	in archive recovery	Instance is in archive recovery state
6	in production	Instance started normally
-	unrecognized status code	Unknown status (Broken pg_control?)

□ Database system identifier

In "Database system identifier" item, ID number that uniquely identifies each database cluster (unsigned 64-bit integer) is output. This number is determined at the creation of the database cluster, and it is never changed. Streaming replication will be performed between the database clusters, which have the same ID. In addition, this number is also recorded in the first block of the WAL file (XLogLongPageHeaderData structure). This avoids that to be used for recover by mistake WAL files of different databases.

Database system identifier generates a unique number by the following code (in BootStrapXLOG function).

Example 79 BootStrapXLOG function (src/backend/access/transam/xlog.c)

```
uint64          sysidentifier;

gettimeofday(&tv, NULL);
sysidentifier = ((uint64) tv.tv_sec) << 32;
sysidentifier |= ((uint64) tv.tv_usec) << 12;
sysidentifier |= getpid() & 0xFFF;
```

Example 80 Header structure of WAL File (src/include/access/xlog_internal.h)

```

/*
 * When the XLP_LONG_HEADER flag is set, we store additional fields in the
 * page header. (This is ordinarily done just in the first page of an
 * XLOG file.) The additional fields serve to identify the file accurately.
 */
typedef struct XLogLongPageHeaderData
{
    XLogPageHeaderData std;          /* standard header fields */
    uint64      xlp_sysid;          /* system identifier from pg_control */
    uint32      xlp_seg_size; /* just as a cross-check */
    uint32      xlp_xlog_blksize; /* just as a cross-check */
} XLogLongPageHeaderData;

```

3.5.5 pg_filenode.map

Pg_filenode.map file will be saved in the "{PGDATA}/global directory" and "directory for the database", it is a small binary file. The size of the file is 512 bytes. This file contains the information to correspond to the actual file name and OID of the part of the system catalog. Storage format is defined in RelMapFile structure.

Example 81 Stored format (src/backend/utils/cache/relmapper.c)

```

typedef struct RelMapping
{
    Oid          mapoid;          /* OID of a catalog */
    Oid          mapfilenode;     /* its filenode number */
} RelMapping;

typedef struct RelMapFile
{
    int32      magic;          /* always RELMAPPER_FILEMAGIC */
    int32      num_mappings; /* number of valid RelMapping entries */
    RelMapping mappings[MAX_MAPPINGS];
    pg_crc32c  crc;          /* CRC of all above */
    int32      pad;          /* to make the struct size be 512 exactly */
} RelMapFile;

```

The first four bytes (magic) is a fixed value 0x592717 (5842711). The number of mapping to the next 4 bytes is stored. The example below is the content of the following file specific database directory.

Example 82 Contents of pg_filnode.map file

```
$ od -t u4 pg_filnode.map
0000000      5842711          15          1259          1259 -- Include 15 entry
0000020           1249          1249          1255          1255 -- OID and file name set
0000040           1247          1247          2836          2836
0000060           2837          2837          2658          2658
0000100           2659          2659          2662          2662
0000120           2663          2663          3455          3455
0000140           2690          2690          2691          2691
0000160           2703          2703          2704          2704
0000200              0              0              0              0
*
0000760              0              0 983931237              0 -- CRC and Padding data
0001000
```

The pg_filnode.map in directory \${PGDATA}/global contains the following table.

- pg_pltemplate
- pg_pltemplate_name_index
- pg_tablespace
- pg_shdepend
- pg_shdepend_depender_index
- pg_shdepend_reference_index
- pg_authid
- pg_auth_members
- pg_database
- pg_shdescription (with TOAST table and TOAST index)
- pg_shdescription_o_c_index
- pg_database_datname_index
- pg_database_oid_index
- pg_authid_rolname_index
- pg_authid_oid_index



- pg_auth_members_role_member_index
- pg_auth_members_member_role_index
- pg_tablespace_oid_index
- pg_tablespace_spcname_index
- pg_db_role_setting (with TOAST table and TOAST index)
- pg_db_role_setting_databaseid_rol_index
- pg_shseclabel (with TOAST table and TOAST index)
- pg_shseclabel_object_index
- pg_replication_origin
- pg_replication_origin_roident_index
- pg_replication_origin_roname_index

The pg_filenode.map in directory for database contains the following table.

- pg_class
- pg_attribute
- pg_proc (with TOAST table and TOAST index)
- pg_type
- pg_attribute_relid_attnam_index
- pg_attribute_relid_attnum_index
- pg_class_oid_index
- pg_class_relname_nsp_index
- pg_class_tblspc_relfilenode_index
- pg_proc_oid_index
- pg_proc_proname_args_nsp_index
- pg_type_oid_index
- pg_type_typname_nsp_index

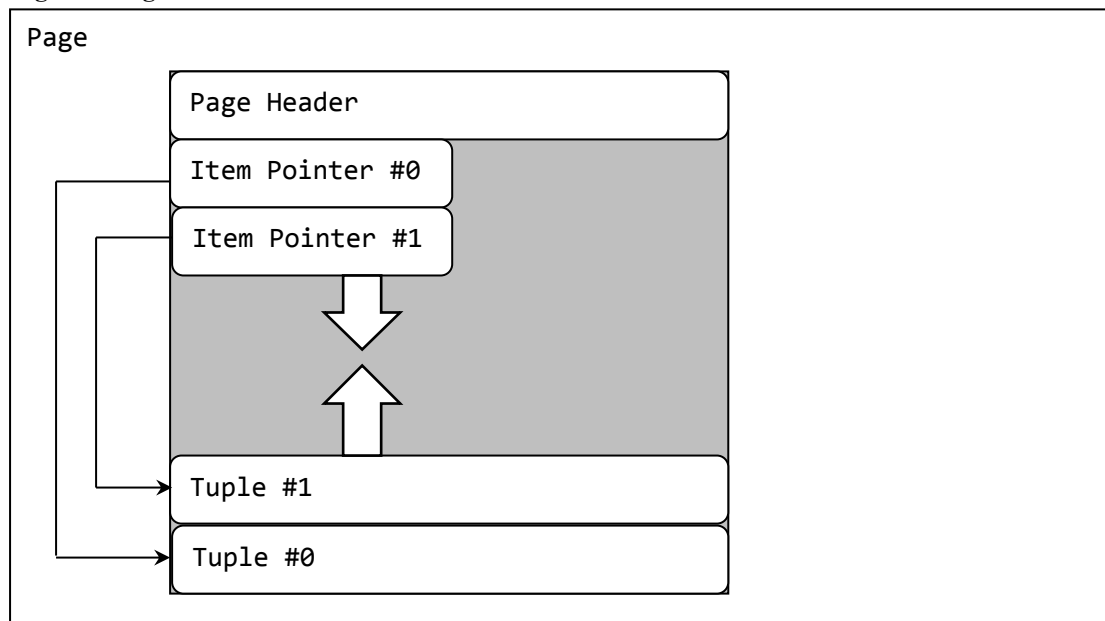
3.6 Block format

3.6.1 Block and Page

In PostgreSQL, I / O is done on a block basis. The size of the block is 8 KB by default. In order to change the value it is necessary to change the "#define BLOCKSZ 8192" in the "src/include/pg_config.h" and recompile it. In current PostgreSQL, a block contains one page, therefore a block and a page can be considered the same thing. 131,072 blocks (pages) can be stored within one file. When the data file exceeds this size ($8 \text{ KB} \times 131,072 = 1 \text{ GB}$), a new file is created. This value is determined by the RELSEG_SIZE macro of pg_config.h file.

In each page, a pointer to each tuple in the page and to the page header is stored. Item pointer indicates the location of the tuples in the page. Tuple is stored towards the top of the page from the end of the page. On the other hand, item pointer will be added towards the end of the page immediately after the page header.

Figure 9 Page Structure



Page header is defined in the structure PageHeaderData in the "src/include/storage/bufpage.h".

Example 83 PageHeaderData structure

```
typedef struct PageHeaderData
{
    /* XXX LSN is member of *any* block, not only page-organized ones */
    PageXLogRecPtr pd_lsn;          /* LSN: next byte after last byte of xlog
                                     * record for last change to this page */

    uint16         pd_checksum;     /* checksum */
    uint16         pd_flags;        /* flag bits, see below */
    LocationIndex  pd_lower;        /* offset to start of free space */
    LocationIndex  pd_upper;        /* offset to end of free space */
    LocationIndex  pd_special;      /* offset to start of special space */
    uint16         pd_pagesize_version;
    TransactionId  pd_prune_xid; /* oldest prunable XID, or zero if none */
    ItemIdData     pd_linp[1];     /* beginning of line pointer array */
} PageHeaderData;
```

Table 41 PageHeaderData internal

Variables	Description	Note
pd_lsn	Log sequence number	
pd_checksum	Checksum value	Changed at 9.3
pd_flags	Flags	
pd_lower	Starting position of the free space in the page	
pd_upper	End position of the free space in the page	
pd_special	End region of the special space in the page	
pd_pagesize_version	Page size and version information	
pd_prune_xid	XMAX value, which is the oldest in the page and is not truncated. If it does not exist, this value is zero	
pd_linp[]	Item pointer	

3.6.2 Tuple

The structure of the tuple (record) consists of "Tuple Header", "NULL bitmap", and "data". Tuple header is defined in the "src/include/access/htup_details.h". In T_heap, defined tuple header top, the transaction-related information is recorded. NULL bitmaps are stored in t_bits field.



Example 84 HeapTupleHeaderData structure

```
struct HeapTupleHeaderData
{
    union
    {
        HeapTupleFields t_heap;
        DatumTupleFields t_datum;
    } t_choice;
    ItemPointerData t_ctid; /* current TID of this or newer tuple */
    /* Fields below here must match MinimalTupleData! */
    uint16 t_infomask2; /* number of attributes + various flags */
    uint16 t_infomask; /* various flag bits, see below */
    uint8 t_hoff; /* sizeof header incl. bitmap, padding */
    /* ^ - 23 bytes - ^ */
    bits8 t_bits[1]; /* bitmap of NULLs -- VARIABLE LENGTH */
    /* MORE DATA FOLLOWS AT END OF STRUCT */
};
```

Example 85 HeapTupleFields (t_heap)

```
typedef struct HeapTupleFields
{
    TransactionId t_xmin; /* inserting xact ID */
    TransactionId t_xmax; /* deleting or locking xact ID */

    union
    {
        CommandId t_cid; /* inserting or deleting command ID, or both */
        TransactionId t_xvac; /* old-style VACUUM FULL xact ID */
    } t_field3;
} HeapTupleFields;
```

Table 42 HeapTupleFields internal

Variable	Description	Note
t_xmin	Inserted XID	
t_xmax	Deleted XID	Usually 0, updated even if ROLLBACK occurs
t_cid	Deleted command ID	
t_xvac	Version that has been moved by VACUUM	

T_xmin and t_max fields of tuple correspond to the virtual column XMIN and XMAX.

3.7 Wraparound problem of transaction ID

3.7.1 Transaction ID

When a transaction starts in PostgreSQL, a unique transaction ID is issued (obtained in `txid_current` function). The size of the transaction ID is an unsigned 32-bit (as defined in `src/include/c.h`). When the tuple of the table is updated, updated transaction ID is stored to each tuple header. By this feature, PostgreSQL can maintain referential integrity on search.

□ Confirmation of the transaction ID

Transaction ID corresponding to the tuples on the table can be checked by specifying the XMAX and XMIN pseudo column. XMIN column shows the transaction ID where tuple is added (by INSERT or UPDATE). XMAX column is the transaction ID of the deleted tuple. For this reason, valid tuple's XMAX value is zero. In the example below, the XMAX value is non-zero in the column of C1 = 300, this indicates that the update transaction is rolled back.

Example 86 Transaction ID of the tuples

```
postgres=> SELECT XMAX, XMIN, c1 FROM data1 ;
```

xmin	xmax	c1
507751	0	100
507752	0	200
507754	0	400
507755	0	500
507756	<u>507757</u>	300

(5 rows)

□ Transaction ID wraparound

Transaction ID has a size of unsigned 32-bit, and it will increase monotonically. In large systems, utilizing a large number of transactions may runs out 32 bits. Therefore, a mechanism is provided to avoid the problem when the transaction ID of PostgreSQL wraparound. This mechanism replaces the old transaction ID to a special value (FrozenXID = 2) regularly.

This replacement process is referred to as the FREEZE process, and usually done in the VACUUM processing. When the number of the transactions for the table exceeds the number calculated in the parameter "autovacuum_freeze_max_age - vacuum_freeze_min_age" FREEZE process occurs even if the automatic VACUUM is invalid.

Automatic VACUUM discover the processing target block using the Visibility Map. When the FREEZE process is completed, automatic VACUUM process update the Visibility Map. When the number of transactions that have been specified in the parameter `vacuum_freeze_table_age` is executed, it performs the search for a block that has not been Freeze from Visibility Map.

Transaction ID, which is a freeze target of each table, is defined by `relfrozenxid` column of `pg_class` catalog. This column is the minimum value of XMIN virtual column in the table (except FrozenXID). The minimum value of `relfrozenxid` of `pg_class` catalog can be found in the `datfrozenxid` column of `pg_database` catalog.

□ In case of automatic VACUUM process failure

If FREEZE process is not performed for some reason, when the remaining transaction number to the wraparound falls below 10 million, following message is outputted.

Example 87 Warning message for transaction wraparound

```
WARNING:  database "postgres" must be vacuumed within 9999999 transactions
HINT:    To avoid a database shutdown, execute a database-wide VACUUM in "postgres".
```

Moreover, when the remaining transaction number falls below 1 million, the following message is output, and the system will stop.

Example 88 Error message for transaction wraparound

```
ERROR:  database is not accepting commands to avoid wraparound data loss in
database "postgres"
HINT:   Stop the postmaster and use a standalone backend to VACUUM in "postgres".
```

Database fallen into such a state starts in stand-alone mode, and executes VACUUM statement. In the example below VACUUM processing is performed to postgres database.

Example 89 Start by stand-alone mode and VACUUM

```
$ postgres --single -D data postgres

PostgreSQL stand-alone backend 9.6.2
backend> VACUUM
backend>
```

3.7.2 The parameters for the FREEZE processing

Parameters for FREEZE operation of the transaction ID is as follows:

□ Autovacuum_freeze_max_age

Specifies the maximum age that `pg_class.relFrozenxid` can reach in order to prevent the circulation of the transaction ID. Even if automatic VACUUM is disabled, it will launch VACUUM worker process. The default value is 200 million (200,000,000). The minimum value is 100 million (100,000,000), the maximum 2 billion (2,000,000,000).

□ Vacuum_freeze_min_age

Specifies the cut-off age to replace the transaction ID to FrozenXID at the time of table scan by VACUUM. The default value is 50 million (50,000,000). The value from 0 to 50% of `autovacuum_freeze_max_age` can be specified.

□ Vacuum_freeze_table_age

When `pg_class.relFrozenxid` of the table reaches to the time specified in this value, VACUUM scans the table aggressively. The default value is 150 million (150,000,000). The value can be specified from 0 to 1 billion (1,000,000,000) or up to 95% of the `autovacuum_freeze_max_age`.

PostgreSQL 9.3.3 later, the following parameters have been added. There are multi transaction ID (MultiXactId) and the parameters for the freeze, but details are unconfirmed.

- `autovacuum_multixact_freeze_max_age`
- `vacuum_multixact_freeze_min_age`
- `vacuum_multixact_freeze_table_age`

3.8 Locale specification

3.8.1 Specifying the locale and encoding

The default locale setting is specified in `--locale` parameter of the `initdb` command. When encoding is also specified in `--locale` parameter, it becomes the defaults to the encoding.

Example 90 Specify the locale name and encoding as locale

```
$ export LANG=C
$ initdb --locale=ja_JP.utf8 data
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "ja_JP.utf8".
The default database encoding has accordingly been set to "UTF8".
initdb: could not find suitable text search configuration for locale "ja_JP.utf8"
The default text search configuration will be set to "simple".
.....
```

When only the locale name is specified in the locale, the default encoding of the locale is used. In the case that `ja_JP` is specified as locale, Japanese EUC (EUC_JP) is specified as the encoding.

Example 91 Specified only locale name as the locale

```
$ export LANG=en_US.utf8
$ initdb --locale=ja_JP data
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "ja_JP".
The default database encoding has accordingly been set to "EUC_JP".
initdb: could not find suitable text search configuration for locale "ja_JP"
The default text search configuration will be set to "simple".
.....
```

When only the encoding is specified without using the locale, encoding sequence of `pg_database` catalog is specified, but in the locale-related column (`datcollate`, etc.) "C" is output.

Example 92 Specify encoding without using the locale

```
$ initdb --no-locale --encoding=utf8 data
```

The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "C".
The default text search configuration will be set to "english".

.....

```
postgres=> SELECT datname, pg_encoding_to_char(encoding), datcollate FROM
           pg_database ;
```

datname	pg_encoding_to_char	datcollate
template1	UTF8	C
template0	UTF8	C
postgres	UTF8	C

(3 rows)

In the table below, the relationship between the specification method of the initdb command and the locale of the database cluster under the conditional where ja_JP.utf8 is specified in the environment variable LANG.

Table 43 initdb command and locale

initdb command parameter	lc_collate	lc_ctype	encoding
not specified	ja_JP.utf8	ja_JP.utf8	UTF8
--locale=ja_JP	ja_JP	ja_JP	EUC_JP
--locale=ja_JP --encoding=utf8	initdb command error		
--locale=ja_JP.utf8	ja_JP.utf8	ja_JP.utf8	UTF8
--locale=ja_JP.EUC_JP --encoding=utf8	initdb command error		
--no-locale	C	C	SQL_ASCII
--no-locale --encoding=utf8	C	C	UTF8

- The function of the ja_JP locale setting
Sorry, Japanese version document only.

3.8.2 The use of the index by LIKE

In locale enabled database, there is a specification that index of the corresponding column is not used in the front match by LIKE clause.

Example 93 LIKE search with locale use

```
postgres=> CREATE TABLE locale1(c1 varchar(10), c2 varchar(10)) ;
CREATE TABLE
postgres=> CREATE INDEX idx1_locale1 ON locale1(c1) ;
CREATE INDEX
postgres=> INSERT INTO locale1 VALUES ('ABC', 'DEF') ;
INSERT 0 1
.....
postgres=> ANALYZE locale1 ;
ANALYZE
postgres=> EXPLAIN SELECT C1 FROM locale1 WHERE C1 LIKE 'A%' ;
               QUERY PLAN
-----
Seq Scan on locale1  (cost=0.00..1790.01 rows=10 width=5)
  Filter: ((c1)::text ~~ 'A% '::text)
Planning time: 1.742 ms
(3 rows)
```

Execution plan becomes Seq Scan, and you can confirm that all tuples search is performed. In order to avoid such situation, specify the option when CREATE INDEX statement is executed to perform the index search by binary.

Syntax 3 Option specification of the CREATE INDEX statement

```
CREATE INDEX index_name ON table_name (column_name optional)
```

Depending on the data type of the column, the following options can be specified.

Table 44 Binary comparison options

Column data type	Option
varchar	varchar_pattern_ops
char	bpchar_pattern_ops
text	text_pattern_ops
name	name_pattern_ops

Example 94 LIKE search when option specified

```
postgres=> CREATE INDEX idx2_locale1 ON locale1(c2 varchar_pattern_ops) ;
CREATE INDEX
postgres=> \d locale1
           Table "public.locale1"
  Column |          Type          | Modifiers
-----+-----+-----
 c1      | character varying(10) |
 c2      | character varying(10) |
Indexes:
    "idx1_locale1" btree (c1)
    "idx2_locale1" btree (c2 varchar_pattern_ops)

postgres=> EXPLAIN SELECT C2 FROM locale1 WHERE C2 LIKE 'A%' ;
               QUERY PLAN
-----
Index Only Scan using idx2_locale1 on locale1 (cost=0.42..8.44 rows=10 width=5)
  Index Cond: ((c2 ~>=~ 'A'::text) AND (c2 ~<~ 'B'::text))
  Filter: ((c2)::text ~~ 'A%'::text)
Planning time: 0.541 ms
(4 rows)
```

3.8.3 Using the index by <, > operators

In the LIKE operator, option specification is necessary. However, if users want to use an index with operators "<" and ">" to compare the value, the options described in the previous section cannot be used.

Example 95 Specified at the time of the index option

```

postgres=> \d locale1

          Table "public.locale1"
   Column |          Type          | Modifiers
-----+-----+-----
  c1      | character varying(10) | 
  c2      | character varying(10) | 
Indexes:
    "idx1_locale1" btree (c1)
    "idx2_locale1" btree (c2 varchar_pattern_ops)

postgres=> EXPLAIN SELECT c1 FROM locale1 WHERE c1 < '10' ;

              QUERY PLAN
-----
Index Only Scan using idx1_locale1 on locale1  (cost=0.42..334.75 rows=306
width=5)
    Index Cond: (c1 < '10'::text)
    Planning time: 0.210 ms
    (3 rows)

postgres=> EXPLAIN SELECT c2 FROM locale1 WHERE c2 < '10' ;

              QUERY PLAN
-----
Seq Scan on locale1  (cost=0.00..1790.01 rows=10 width=5)
    Filter: ((c2)::text < '10'::text)
    Planning time: 0.140 ms
    (3 rows)

```

3.8.4 Locale and encoding of the specified location

The locale and encoding can be specified not only on the creation of the database cluster, but also on the database creation. If the locale / encoding different from the default is specified, it is necessary to specify template0 as a template and specify ENCODING, LC_COLLATE, and LC_CTYPE clause.

Example 96 Specifying the locale and encoding

```
postgres=# CREATE DATABASE eucdb1 WITH TEMPLATE=template0 ENCODING='EUC JP'
        LC_COLLATE='C' LC_CTYPE = 'C' ;
CREATE DATABASE
postgres=# \l
```

List of databases

Name	Owner	Encoding	Collate	Ctype	Access privileges
eucdb1	postgres	EUC JP	C	C	
postgres	postgres	UTF8	ja_JP.utf8	ja_JP.utf8	
datadb1	user1	UTF8	ja_JP.utf8	ja_JP.utf8	
template0	postgres	UTF8	ja_JP.utf8	ja_JP.utf8	=c/postgres +
					postgres=Ctc/postgres
template1	postgres	UTF8	ja_JP.utf8	ja_JP.utf8	=c/postgres +
					postgres=Ctc/postgres

(5 rows)

The above example specifies a "C" to LC_COLLATE and LC_CTYPE, but in order to create a database in a different locale, specify the locale name and encoding name to both parameters. If you specify Japanese EUC to encoding, "eucjp" or "EUC_JP" can also be specified.

3.9 Data Checksum

Block checksum function has been added from PostgreSQL 9.3. For each block, the checksum is given during the update, and check is performed on reading.

3.9.1 Specifying the checksum

Checksum feature is disabled by default, but you can create a database cluster that have enabled the checksum by specifying the `-k8` option to `initdb` command.

Example 97 Enable checksum features

```
$ initdb -k data
```

The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.UTF-8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are enabled. -- enabled checksum features

.....

3.9.2 Checksum location

The checksum is stored as a 16-bit area behind `pd_lsn` field in the page header. This place is the part, where timeline ID (`pd_tli`) was stored up to PostgreSQL 9.2. Since the checksum header size has not still changed by adding checksum, there is no change in I/O amount in comparison with the previous version. CPU resources for calculations and check of checksum are expected to increase. The structure of the page header (`PageHeaderData`) is defined in the header file `src/include/storage/bufpage.h`. Additional checksum will be conducted at the time of writing of the page. Actual checksum is created in `pg_checksum_page` function in a header file `include/storage/checksum_impl.h`.

⁸ Or `--data-checksums` option

Example 98 Page Header Structure

```
typedef struct PageHeaderData
{
    /* XXX LSN is member of *any* block, not only page-organized ones */
    PageXLogRecPtr pd_lsn;          /* LSN: next byte after last byte of xlog
                                     * record for last change to this page */

    uint16         pd_checksum;    /* checksum */
    uint16         pd_flags;       /* flag bits, see below */
    LocationIndex pd_lower;        /* offset to start of free space */
    LocationIndex pd_upper;        /* offset to end of free space */
    LocationIndex pd_special;      /* offset to start of special space */
    uint16         pd_pagesize_version;
    TransactionId pd_prune_xid;    /* oldest prunable XID, or zero if none */
    ItemIdData     pd_linp[1];     /* beginning of line pointer array */
} PageHeaderData;
```

3.9.3 Checksum Error

□ Verification of checksum

Checksum verification is done at the time when the read of the block into the shared buffer has been completed. If a checksum error is detected, it is logged by ReadBuffer_common function of source code src/backend/storage/buffer/bufmgr.c. Actual verification is done in PageIsVerified function of source code src/backend/storage/page/bufpage.c. The following errors will occur in case of incorrect checksum detection.

Example 99 Incorrect checksum error

```
WARNING: page verification failed, calculated checksum 2773 but expected 29162
ERROR:   invalid page in block 0 of relation base/12896/16385
```

For the tables where checksum error occurs, the subsequent execution of the DML cause all the same error.

□ Ignore checksum

The parameter ignore_checksum_failure is set to "on", PostgreSQL ignores the error of checksum (default: "off").



3.9.4 Check the existence of checksum

In order to confirm the checksum specification for a database cluster, verify the execution result of `pg_controldata` utility, parameters `data_checksums` or `pg_control_init` function. Parameter `data_checksums` is available from PostgreSQL 9.3.4.

Example 100 Checking the checksum

```
$ pg_controldata ${PGDATA} | grep checksum
Data page checksum version:          1      -- Checksum enabled
$
$ psql
postgres=> SHOW data_checksums ;
 data_checksums
-----
 on              -- Checksum enabled
(1 row)
postgres=> SELECT data_page_checksum_version FROM pg_control_init() ;
 data_page_checksum_version
-----
                          1      -- Checksum enabled
(1 row)
```

3.10 Log File

This section describes the log files that PostgreSQL outputs.

3.10.1 Output of the log file

PostgreSQL outputs two types of logs; the log for outputs of the activity during an instance start, and the log for pg_ctl commands.

□ Instance activity log

Log output destination during the instance running is determined by parameters log_destination (default: "stderr"). It will transfer to SYSLOG when the value of this parameter is set to "syslog" (specified in "eventlog" in the Windows environment).

When the parameter log_destination is set to "stderr" or "csv", and parameter logging_collector specified to "on" (default: "off"), the log is written to a file.

Table 45 Output destination of the log file (parameter log_destination)

Parameter values	Description	Note
stderr	Standard error output	
csvlog	CSV file	Require logging_collector=on
syslog	SYSLOG transfer	
eventlog	Windows event	Only in the Windows environment

□ Start / stop instance log

In order to output the execution result of pg_ctl command to a log file, specify the file name with the -l option. If not specified, it is output to the standard output. If the log file cannot be created, start of the instance (pg_ctl start) will fail and it cannot be started, but stop Instance (pg_ctl stop) is successfully. The log will be appended if an existing file name is specified in the -l option.

□ Output method of log file

Log file is opened by "fopen" function, and written by "fwrite" function. Since "fflush" function is not performed for each write, in case of OS crash or using storage replication, log might not be written partly.

3.10.2 Log file name

This section describes the elements that determine the log file name.

□ Parameters

Output destination and file name of the log file are determined by the following parameters.

Table 46 Determination of the log file

Parameter	Description	Default Value
log_directory	Log file output directory path	pg_log
log_filename	Log file name	postgresql-%Y-%m-%d_%H%M%S.log
log_file_mode	Log file access mode	0600

In the parameter log_directory, you can specify a relative path from the database cluster or an absolute path. It is automatically created during the instance start if the specified directory does not exist. Instance startup will fail if the directory cannot be created. Following example is an instance startup error when value "/var" is specified to the parameter log_directory.

Example 101 Directory creation error

```
$ pg_ctl -D data start -w
waiting for server to start....FATAL:  could not open log file
"/var/postgresql-2017-02-11_131642.log": Permission denied
stopped waiting
pg_ctl: could not start server
Examine the log output.
```

When "%A" is included in parameter log_filename, a day of the week name in English is output without depending on a locale of a database cluster.

□ Output CSV file

When "csvlog" is designated in parameter log_destination, the output form of the log file becomes CSV, i.e., with comma (,) separation.

Example 102 CSV File format

```
2017-02-11 13:18:55.935 JST,,,15302,,57ede7af.3bc6,2,,2017-02-11 13:18:55
JST,,0,LOG,00000,"database system is ready to accept connections",,,,,,,,,,""
2017-02-11 13:18:55.937 JST,,,15308,,57ede7af.3bcc,1,,2017-02-11 13:18:55
JST,,0,LOG,00000,"autovacuum launcher started",,,,,,,,,,""
```

If the file extension that is specified in the parameter `log_filename` is ".log", the log file whose extension is changed to ".csv" is created. The file with the extension of ".log" will also be created at the same, but only the part of the contents are output.

□ Output to SYSLOG

Specifying "syslog" to a parameter `log_destination`, log data will be forwarded to syslog of the local host. However, a log file in which a few lines are recorded is also created.

Example 103 Information that has been transferred to the SYSLOG

```
Feb 11 13:21:04 rel71-2 postgres[15328]: [4-1] LOG:  MultiXact member wraparound
protections are now enabled
Feb 11 13:21:04 rel71-2 postgres[15325]: [3-1] LOG:  database system is ready
to accept connections
Feb 11 13:21:04 rel71-2 postgres[15332]: [3-1] LOG:  autovacuum launcher started
```

Table 47 Parameters related to the SYSLOG output

Parameter	Description	Default value
<code>log_destination</code>	Enable SYSLOG transfer by setting to syslog	stderr
<code>syslog_facility</code>	SYSLOG facility	LOCAL0
<code>syslog_ident</code>	Application name that is output to the log	postgres
<code>syslog_sequence_numbers</code>	Add a sequence number to the SYSLOG message	on
<code>syslog_split_messages</code>	Split long SYSLOG messages per 900 bytes	on

In the output log of SYSLOG, process ID of the log output process is written after the name specified in the parameter `syslog_ident`.

3.10.3 Log Rotation

When `pg_rotate_logfile` function is executed, or SIGUSR1 signal is sent to logger process, log rotation is performed. However, if a new file cannot be created, for example parameter `log_filename` does not contain time information, log rotation will not be performed.

□ Specification of `pg_rotate_logfile` function

Only users with superuser privileges can perform this function. When a general user executes it, following message will be output.

Example 104 Error Message

```
ERROR: must be superuser to rotate log files
```

This function returns true when logging_collector parameter is set to "on" (= logger process has been started). If logging_collector parameter is "off", the execution of the function succeeds, but the following message is output and the function returns false.

Example 105 Error Message

```
WARNING: rotation not possible because log collection not active
```

□ Log file deletion during output

When the log currently in use is deleted, a new log file will not be automatically re-created. A new log file can be created by forcing a rotation.

3.10.4 Contents of the log

In the default configuration log, the string that indicates the category of the log is output at the beginning, followed by the recording log contents. Strings shown in the category are as follows.

Table 48 Categories of error log

Category	Contents
DEBUG:	Message for developers
INFO:	Explicit detail information by the user (VACUUM VERBOSE, etc.)
NOTICE:	Auxiliary information to the user, such as truncation of long identifiers
WARNING:	Warning to the user, such as COMMIT execution outside the transaction
ERROR:	Execution error of command, etc.
LOG:	The message for administrators, such as activities of the checkpoint
FATAL:	Error with the end of the session etc.
PANIC:	Stop instances error or the end of all sessions error, etc.
???	Unknown message. It is not output.

In the standard configuration log, log output time, user name, database name, etc., are not output at all. It is insufficient to use as a purpose of the audit. In order to output these information to log, specify the parameter log_line_prefix. You can specify the following character.

Table 49 String that can be specified as a parameter log_line_prefix

String	Description
%a	Application name (set by SET application_name)
%u	Connection user name
%d	Connection database name
%r	Remote host name and port number (on local connection, [local])
%h	Remote host name
%p	Process ID
%t	Time stamp, excluding the millisecond
%m	Time stamp of including milliseconds
%i	Command name (INSERT, SELECT, etc.)
%e	SQLSTATE error code
%c	Session ID
%l	Session line number
%s	Session start time
%v	Virtual transaction ID
%x	Transaction ID
%q	Stop the output after this scope in the non-session processes
%%	% character
%n	Timestamp that have been expressed as a decimal number.

3.10.5 Encoding of log file

To output an error message in the character code of non-standard, you need to specify the "--enable-nls=yes" to the configure command before the PostgreSQL installation. This parameter is not specified by default, rpm file of the community edition will contain the binary, which is specify this parameter. PostgreSQL settings currently in use can be found in pg_config command.

Example 106 Display of configure settings using a pg_config command

```
$ pg_config | grep CONFIGURE
CONFIGURE = '--enable-nls=yes'
$
```

Character code string that is output to the log file is different in the three parts of the following. The following example is acquired by PostgreSQL you specify the "--enable-nls=yes".

- Log on at instance startup / shutdown

Log of pg_ctl command is output in the OS locale specified when the command is executed (the LANG environment variable, etc.).

Example 107 Instance startup log (depends on the environment variable LANG)

```
LOG: データベースシステムは 2015-11-11 12:30:26 JST にシャットダウンしました
LOG: MultiXact member wraparound protections are now enabled
LOG: データベースシステムの接続受付準備が整いました。
LOG: 自動バキュームランチャプロセス
```

- Error message

Error messages are output in the encoding specified in the parameter lc_message in locale enabled database.

Example 108 Error message (depends on the parameter lc_message)

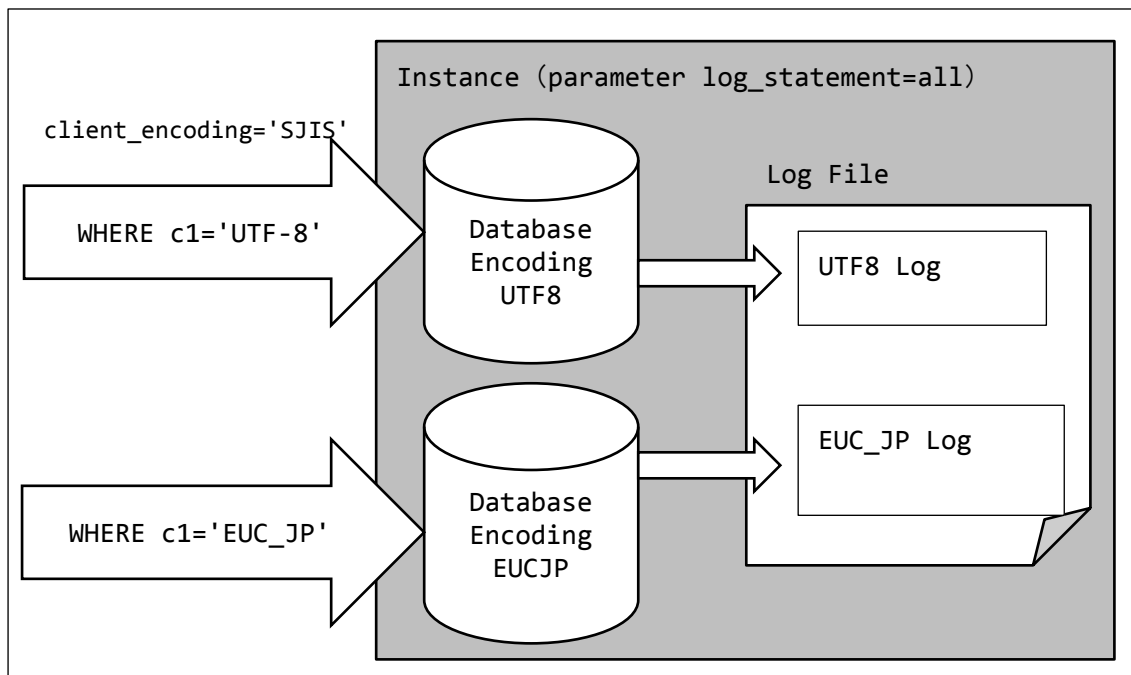
```
ERROR: リレーション"data1"は存在しません(文字位置 15)
ステートメント: select * from data1;
ERROR: リレーション"data1"は存在しません
行 1: select * from data1;
```

- SQL statement

Encoding of the SQL statements that are output by specifying the parameter log_statement does not depend on the client-side configuration, it is determined by the encoding of the destination database. For this reason, in an environment where multiple database with a different encoding are created, the character code of the SQL statement output to the log will be mixed.

In the following figure, A SQL statement is executed to the Japanese EUC (EUC_JP) database and UTF-8 database. Parameter client_encoding is set to "SJIS", so SQL statement is sent in Shift_JIS. As the parameters log_statement is specified to "all", executed SQL statement is recorded in the log. Recorded SQL statements are converted into UTF8 or Japanese EUC respectively, and they are mixed in the interior of the log file.

Figure 10 Encoding the log



4. Trouble Shooting

4.1 File deletion before startup instance

4.1.1 Deleted pg_control

An instance cannot start an instance in the case that pg_control file does not exist.

Example 109 Error log instance startup

```
postgres: could not find the database system
Expected to find it in the directory "/usr/local/pgsql/data",
but could not open file "/usr/local/pgsql/data/global/pg_control": No
such file or directory
```

In order to recover the pg_control file, restore the pg_control file from a backup, and then re-create the WAL file with specifying the -x option to pg_resetxlog command. If the instance was terminated abnormally, specify -f option also at the same time. Instance cannot start in case that pg_control file is only restored.

Example 110 Instance startup error when pg_control restore only

```
LOG:  invalid primary checkpoint record
LOG:  invalid secondary checkpoint record
PANIC: could not locate a valid checkpoint record
LOG:  startup process (PID 12947) was terminated by signal 6: Aborted
LOG:  aborting startup due to startup process failure
```

About the transaction ID specified in the -x option of pg_resetxlog command, please refer to the manual.

4.1.2 Deleted WAL file

This section describes the operation when the WAL file is deleted.

- When all WAL file has been deleted

If an instance is successful, regardless of the case of abnormal termination, if the entire WAL file has been deleted, Instance cannot start under the condition. Execute pg_resetxlog command to re-create the WAL file. In case of immediately after an abnormal instance termination, specify -f option to pg_resetxlog command to create WAL files forcibly.

Example 111 Startup failure log due to missing WAL

```
LOG:  could not open file "pg_xlog/000000010000000000000002" (log file
0, segment 2): No such file or directory
LOG:  invalid primary checkpoint record
LOG:  could not open file "pg_xlog/000000010000000000000002" (log file
0, segment 2): No such file or directory
LOG:  invalid secondary checkpoint record
PANIC: could not locate a valid checkpoint record
LOG:  startup process (PID 27972) was terminated by signal 6: Aborted
LOG:  aborting startup due to startup process failure
```

- After instance abnormal termination, when deleting latest WAL file

If the instance is abnormally terminated, and up-to-date of the WAL file has been deleted, the log file will be an error, but the instance will start normally. Crash recovery is not only run halfway, but nothing is output to the log.

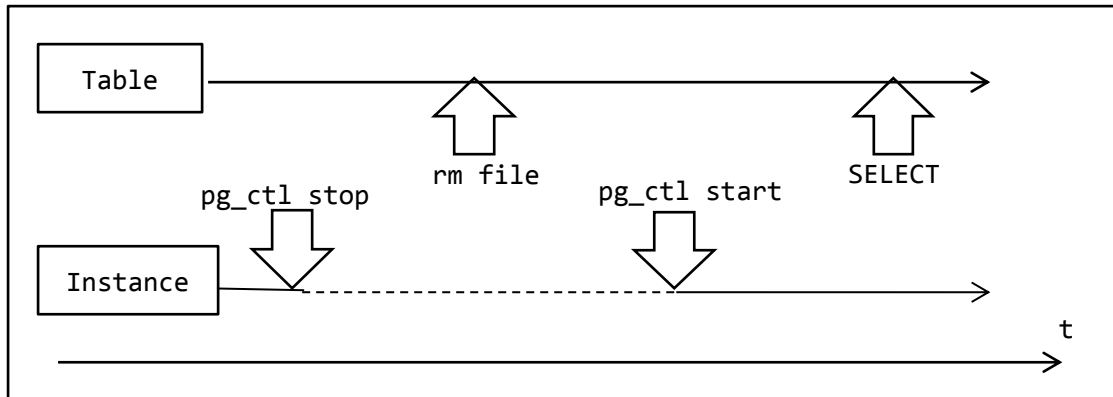
Example 112 Startup log of when the latest WAL file has been deleted

```
LOG:  database system was interrupted; last known up at 2017-02-11 12:07:16 JST
LOG:  database system was not properly shut down; automatic recovery in progress
LOG:  redo starts at 0/2000B98
FATAL: the database system is starting up
FATAL: the database system is starting up
LOG:  redo done at 0/4FFFF78
LOG:  last completed transaction was at log time 2017-02-11 12:08:22.592264+09
FATAL: the database system is starting up
LOG:  MultiXact member wraparound protections are now enabled
LOG:  database system is ready to accept connections
LOG:  autovacuum launcher started
```

4.1.3 Behavior on data file deletion (Normal instance stop)

The data file of which a table is composed was eliminated after a normal shutdown of an instance. Behavior after an instance restart was verified.

Figure 11 Operation of the data file deletion



□ Execution results

Instance started successfully. Error (ERROR category) occurred on the deleted table when accessing with the SELECT statement. Instance or session are not affected.

□ Output log

Logs are not output at the time of instance startup. At the time of SELECT statement execution, following log has been output.

Example 113 Log output on accessing the deleted table

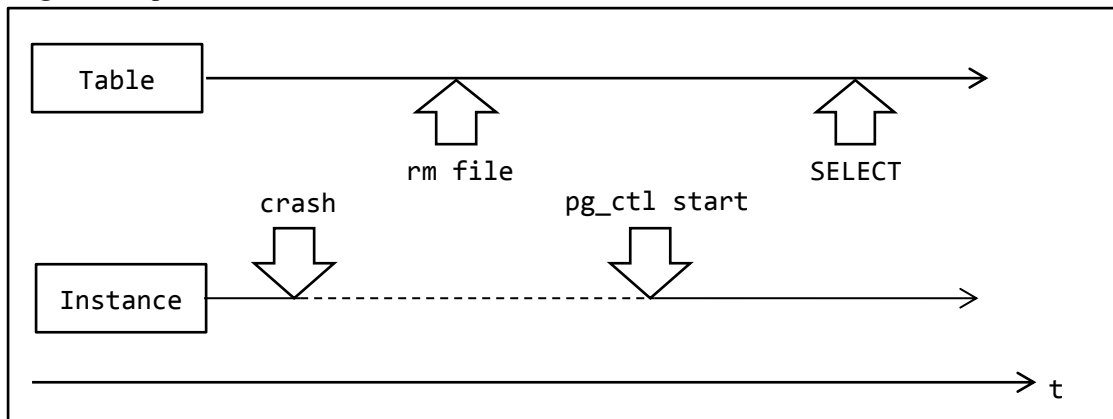
```

LOG:  database system is ready to accept connections
ERROR:  could not open file "base/16385/16392": No such file or directory
STATEMENT:  SELECT COUNT(*) FROM backup1 ;
    
```

4.1.4 Behavior of data file deletion (Instance Crash / No data changed)

The behavior in case of instance crash during running was verified. This is the behavior in case that the data file of the table, which is not modified from the last checkpoint, is deleted. This is the supposition that the database server is terminated abnormally due to OS panic and the file is deleted by fsck command.

Figure 12 Operation of the data file deletion



□ Execution results

Instance started successfully. Error (ERROR category) occurred on the deleted table when accessing with the SELECT statement. Instance of session Instance or sessions are not affected.

□ Outputted log

Crash recovery is logged at instance startup. At the time of SELECT statement execution, following log has been output.

Example 114 Log at the time of the instance startup

```
LOG: database system was interrupted; last known up at 2017-02-11 14:59:05 JST
LOG: database system was not properly shut down; automatic recovery in progress
LOG: redo starts at 3/A000098
LOG: invalid record length at 3/CA7AC28
LOG: redo done at 3/CA79988
FATAL: the database system is starting up
LOG: MultiXact member wraparound protections are now enabled
LOG: database system is ready to accept connections
```

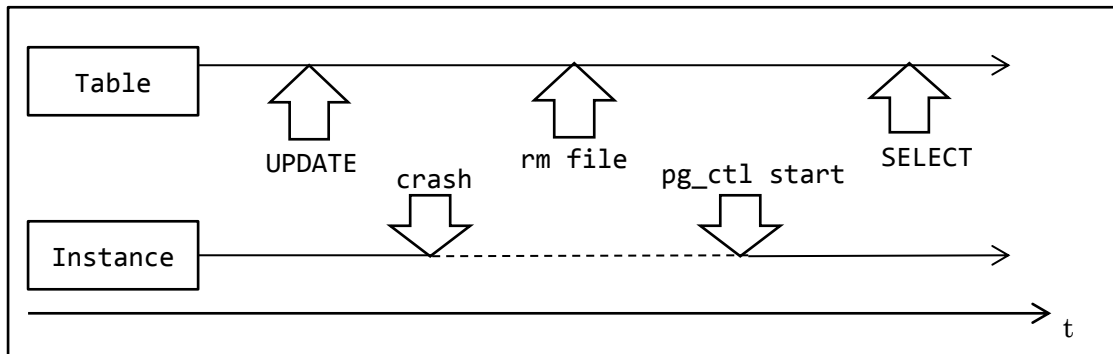
Example 115 Log at the time of the SELECT statement

```
ERROR: could not open file "base/16385/16601": No such file or directory
STATEMENT: SELECT * FROM data1 ;
```

4.1.5 Behavior of data file deletion (Instance Crash / Updated)

The behavior in case of instance crash during running was verified. This is the behavior in case that the data file of the table, which was updated after last checkpoint, and whose transaction information was recorded in WAL, is deleted. This is the supposition that the database server is terminated abnormally due to OS panic, and the file is deleted by fsck command.

Figure 13 Operation of the data file deletion



□ Execute result

Instance started successfully. Error did not occur when accessing in the SELECT statement. However, the information other than the updated block is lost. Instance or sessions are not affected

□ Outputted log

Crash recovery is logged at instance startup. However, the log regarding a loss of the data is not output.

Example 116 Startup log

```
LOG: database system was interrupted; last known up at 2017-02-11 15:03:32 JST
LOG: database system was not properly shut down; automatic recovery in progress
LOG: redo starts at 3/CA7AC98
LOG: invalid record length at 3/CD51DB8
LOG: redo done at 3/CD51D90
LOG: last completed transaction was at log time 2017-02-11 11:35:11.928603+09
LOG: MultiXact member wraparound protections are now enabled
LOG: database system is ready to accept connections
```

4.1.6 Other files

The behaviors in case of the deletion of other files are verified.

□ Behavior of Visibility Map (VM) / Free Space Map (FSM) file deletion

Error does not occur on VM files, FSM file deletion, and SQL statement for the target table succeeds.

These files are re-created on the next VACUUM.

□ Behavior at the time of pg_filenode.map file deletion

When pg_filenode.map file is deleted, it becomes impossible to mapping of the system catalog and the file name, and then cannot use the database.

Example 117 Log on pg_filenode.map deletion

```
FATAL:  could not open relation mapping file
"base/16385/pg_filenode.map": No such file or directory
```

□ Behavior at the time of PG_VERSION file deletion

When PG_VERSION file is deleted, the directory cannot be recognized as the directory for PostgreSQL database.

Example 118 Log on PG_VERSION deletion

```
FATAL:  "base/16385" is not a valid data directory
DETAIL:  File "base/16385/PG_VERSION" is missing.
```

4.2 Delete files in the instance running

I verified the behavior when a file is deleted during instance running

4.2.1 Delete pg_control

If the instance cannot access to pg_control during operation, PANIC occurs and instance stops. Detection is done at a checkpoint occurrence.

Example 119 PANIC log on checkpoint occurrence

```
PANIC: could not open control file "global/pg_control": Permission denied
LOG: checkpointer process (PID 3806) was terminated by signal 6: Aborted
LOG: terminating any other active server processes
WARNING: terminating connection because of crash of another server process
DETAIL: The postmaster has commanded this server process to roll back the
current transaction and exit, because another server process exited abnormally
and possibly corrupted shared memory.
HINT: In a moment you should be able to reconnect to the database and repeat
your command.
```

4.2.2 Delete WAL

□ Delete of WAL during instance running (re-creation available)

When it is detected that the WAL file has been deleted, WAL file is automatically re-created. It was verified by deleting the WAL files in the instance startup state.

□ Delete of WAL during instance running (re-creation not available)

When it is detected that the WAL file has been deleted, and found not to be recreated, PANIC occurs and instance stops. It was verified by removing the WAL files and setting the pg_xlog directory in write-protected state.



Example 120 Log from the running state instance to the time of WAL inaccessible

```
PANIC:  could not open file "pg_xlog/000000010000000300000026": Permission
denied
LOG:  WAL writer process (PID 2518) was terminated by signal 6: Aborted
LOG:  terminating any other active server processes
WARNING:  terminating connection because of crash of another server process
DETAIL:  The postmaster has commanded this server process to roll back the
current transaction and exit, because another server process exited abnormally
and possibly corrupted shared memory.
HINT:  In a moment you should be able to reconnect to the database and repeat
your command.
LOG:  all server processes terminated; reinitializing
LOG:  database system was interrupted; last known up at 2017-02-11 10:43:58 JST
LOG:  creating missing WAL directory "pg_xlog/archive_status"
FATAL:  could not create missing directory "pg_xlog/archive_status": Permission
denied
LOG:  startup process (PID 2624) exited with exit code 1
LOG:  aborting startup due to startup process failure
```


4.3 Process Failure

4.3.1 Behavior of the process abnormal termination

If the backend process other than postmaster terminated abnormally, postmaster process detect it and restart them. Some of the process might restart at the same time. The postgres process that handles the SQL statement from the client will not be restarted because the sessions between client and instance are closed with the abnormal termination.

Table 50 Behavior of process on abnormal termination

Process	Behavior
postmaster ⁹	The entire instance is terminated abnormally; shared memory is not deleted
logger	It will be restarted by the postmaster
writer	It is restarted by postmaster simultaneously with wal writer and autovacuum launcher. All of the postgres process halts
wal writer	It is restarted by postmaster simultaneously with writer and autovacuum launcher. All of the postgres process halts
autovacuum launcher	It's restarted by postmaster simultaneously with writer and wal writer
stats collector	It will be restarted by the postmaster
archiver	It will be restarted by the postmaster
checkpointer	It will be restarted by the postmaster. All of the postgres process halts
postgres	It will not be restarted
wal sender	It will be restarted by the postmaster. Wal receiver process of slave instance will also be restarted. All of the postgres process halts
wal receiver	It will be restarted by the postmaster. Wal sender process of master instance will also be restarted. Postgres process of slave instance is shut down
startup process	The entire slave instance is terminated

The above behavior is the operation when the value of the parameter `restart_after_crash` is set to "on" the default value. If this parameter is set to "off", the entire instance will abort when the backend process stops.

⁹ If you stop the postmaster by sending a KILL signal, shared memory and semaphores are not deleted.



4.3.2 Behavior of the transaction at the process abnormal termination.

When the "stats collector", "logger" and "archiver" other than the process is abnormal termination, all of the postgres processes that were connected to the client will stop. Therefore, transaction in progress will be discarded. In the replication environment, process failure of the slave instance, does not affect the operation of the master instance.

4.4 Other failure

4.4.1 Crash recovery

When an instance is abnormally terminated, the information of the update transaction is stored only WAL because the checkpoint has not been completed. When instance restarts, crash recovery processing which eliminates the inconsistency between data files and WAL is automatically executed.

Crash recovery will begin from reading the `pg_control` file. If the status of the instance is `DB_SHUTDOWNED` (1), crash recovery will not be done because the instance shutdown successfully. Other status means that instance terminated abnormally therefore crash recovery is necessary.

Below I will write the treatment of crash recovery. Following is brief explanation of the crash recovery process.

1. Check the location of the checkpoint. WAL up to the checkpoint are guaranteed to be written to the data file, therefore recovering is not necessary
2. Read the transaction information that occurred after the last checkpoint from WAL
3. Since the first update after the checkpoint wrote entire block to WAL, recover the block (parameter `full_page_writes`)
4. Apply the information of the update transaction from WAL
5. Re-run the update transaction up to the latest WAL

4.4.2 Instance abnormal termination during online backup

PostgreSQL online backups will be conducted in the following procedure.

1. Start an instance in the archive log mode
2. Execute `pg_start_backup` function
3. Copy the database cluster files (by the OS command)
4. Execute `pg_stop_backup` function

Among the procedures above, the behavior in case that an instance terminates abnormally during procedure 3 was inspected. Instance termination was done by sending KILL signal to postmaster.



Example 121 Instance abnormal termination in online backup

```
postgres=# SELECT pg_start_backup('label') ;
pg_start_backup
-----
0/5000020
(1 row)
$ ps -ef | grep postgres
postgres 6016      1  0 12:46 pts/1    00:00:00 /usr/local/pgsql/bin/postgres
$ kill -KILL 6016
$ pg_ctl -D data start -w
pg_ctl: another server might be running; trying to start server anyway
server starting

postgres=# SELECT pg_stop_backup() ;
ERROR:  a backup is not in progress
```

As postmaster.pid file has not been deleted, warnings have been issued, but the restart was successful. From the fact that an error "a backup is not in progress" returns after executing the function pg_stop_backup, we can see that the backup mode is cleared by the restart of the instance.

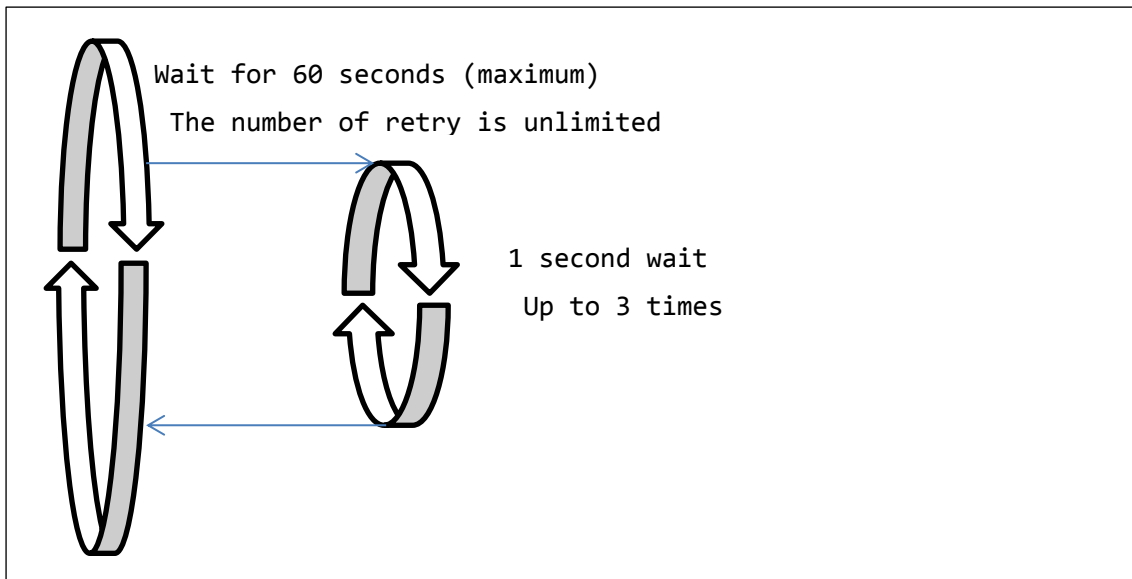
4.4.3 Failure of the archiving

When the WAL file is entirely written, the command specified in the parameter archive_command will be performed using the "system (3)" function (Linux / Windows both) (pgarch_archiveXlog function in src/backend/postmaster/pgarch.c).

□ Re-run of the archiving process

When the "system" function returns a value other than 0, it is considered as the archive process failure. If all three retries fail, archiver process waits a maximum of 60 seconds, and archiver process retry three time.

Figure 14 Re-run of the archiving process



The number of times of the archive processing failure can be checked in `pg_stat_archiver` catalog (from PostgreSQL 9.4).

□ Failure log of the archiving process

When the archive process fails, the log is output. The example below shows an error in case that "cp %p /arch/%f" is specified in the parameter `archive_command` and there is no writer access permission on the archive log output directory. Cp command stopped with status 1. LOG level error reports three times, and finally WARNING level error occurs.

Example 122 Archive processing failure log

```
cp: accessing `/arch/0000000100000000000000070': Permission denied
LOG:  archive command failed with exit code 1
DETAIL:      The      failed      archive      command      was:      cp
pg_xlog/0000000100000000000000070 /arch/0000000100000000000000070
cp: accessing `/arch/0000000100000000000000070': Permission denied
LOG:  archive command failed with exit code 1
DETAIL:      The      failed      archive      command      was:      cp
pg_xlog/0000000100000000000000070 /arch/0000000100000000000000070
cp: accessing `/arch/0000000100000000000000070': Permission denied
LOG:  archive command failed with exit code 1
DETAIL:      The      failed      archive      command      was:      cp
pg_xlog/0000000100000000000000070 /arch/0000000100000000000000070
WARNING:  archiving transaction log file "0000000100000000000000070"
failed too many times, will try again later
```

Error messages during the archive process failure are as follows.

Table 51 Archive processing failure log

Error Level	Error Message	Description
WARNING	archive_mode enabled, yet archive_command is not set	On archive log mode, but the configuration parameter archive_command is not specified.
WARNING	archiving transaction log file \"%s\" failed too many times, will try again later	For all three retries fail, it will temporarily wait.
FATAL LOG	archive command failed with exit code %d	Archive command exits with a failure status
FATAL	archive command was terminated by exception	Archive command received an exception (Windows), and failed
FATAL	archive command was terminated by signal %d	Archive command received an exception (other than Windows), and failed
FATAL LOG	archive command exited with unrecognized status %d	Archive command exits with unknown error



As for error level selection (FATAL or LOG), when WIFSIGNALED macro exceeds 128 or WEXITSTATUS macro becomes true, the level becomes FATAL. Otherwise, the error level is LOG.

□ WAL with archive failure

When the archive process fails, WAL file for which the process fails is not reused, and new WAL file will be added. For this reason, when the archive process continues to fail, a large number of WAL files leave below pg_xlog directory.

5. Performance Related Information

5.1 Automatic statistical information collection

5.1.1 Timing

Collection of statistics will be executed at the same time as automatic VACUUM. From autovacuum launcher process, autovacuum worker process that performs the actual processing at the interval specified in the parameter `autovacuum_naptime` (default 1 min) is launched.

5.1.2 Conditions

Collection of statistical information is determined by comparing the value acquired by the following equation and the number of tuples that have been updated from the previous acquisition of the statistics. The number of updated tuples is the sum of the tuples that were affected by the UPDATE / DELETE / INSERT statement. Check of the condition is performed by `relation_needs_vacanalyze` function in the source code (`src/backend/postmaster/autovacuum.c`).

Formula 3 Autovacuum Threshold

$$\text{Threshold} = \text{autovacuum_analyze_threshold} + \text{autovacuum_analyze_scale_factor} * \text{reltuples}$$

The meaning of the above equation is as follows:

- `autovacuum_analyze_threshold`
The value of the parameter `autovacuum_analyze_threshold` (default value is 50), or the value of the `autovacuum_analyze_threshold` attribute of the table
- `autovacuum_analyze_scale_factor`
The value of the parameter `autovacuum_analyze_scale_factor` (default value 0.1 = 10%), or the value of the `autovacuum_analyze_scale_factor` attributes of the table
- `reltuples`
This is the number of valid tuples of the table when the statistics information acquired last time.

5.1.3 The number of sample tuples

Collection of statistical information is handled by sampling. Number of the tuples of sampling does not depend on the number of the tuples or the blocks of the table. Corresponding to the setting of the target table for which (or within automatic VACUUM) ANALYZE statement will be executed, it is

determined by the following formula.

Formula 4 Number of sampling tuples

Number of sampling tuples = MAX(column STATISTICS value) * 300

For the default value for STATISTICS of each column, parameter default_statistics_target is used. When the number of active tuples in the table is less or equal than this value all the active tuples in the table become the target of the sampling. This formula is defined in std_typanalyze function of source code (src/backend/commands/analyze.c). The following is a comment about the details to determine this formula.

Example 123 Comment in the source code:

The following choice of minrows is based on the paper "Random sampling for histogram construction: how much is enough?" by Surajit Chaudhuri, Rajeev Motwani and Vivek Narasayya, in Proceedings of ACM SIGMOD International Conference on Management of Data, 1998, Pages 436-447. Their Corollary 1 to Theorem 5 says that for table size n , histogram size k , maximum relative error in bin size f , and error probability γ , the minimum random sample size is

$$r = 4 * k * \ln(2*n/\gamma) / f^2$$

Taking $f = 0.5$, $\gamma = 0.01$, $n = 10^6$ rows, we obtain

$$r = 305.82 * k$$

Note that because of the log function, the dependence on n is quite weak; even at $n = 10^{12}$, a $300*k$ sample gives ≤ 0.66 bin size error with probability 0.99. So there's no real need to scale for n , which is a good thing because we don't necessarily know it at this point.

Table 52 Data of calculation formula

Item	Description
r	The number of sample tuples necessary to acquire
k	The size of the histogram (Parameter default_statistics_target or column STATISTICS)
n	Number of tuples (constant fixed to 1,000,000)
gamma	Error probability (constant fixed to 0.01)
f	The maximum relative error (constant fixed to 0.5)

To output the number of sample tuples that are acquired by the ANALYZE statement, either execute ANALYZE VERBOSE statement or set the parameters log_min_messages to DEBUG2. The following example is a log of executing "ANALYZE VERBOSE data1" statement. To the table, which stores about 100,000 tuples, 30,000 tuples have been sampled.

Example 124 Server log of statistics collection

```
DEBUG:  analyzing "public.data1"  
DEBUG:  "data1": scanned 542 of 542 pages, containing 100100 live rows  
and 0 dead rows; 30000 rows in sample, 100000 estimated total rows
```

Example 125 The output of the ANALYZE VERBOSE statement

```
postgres=> ANALYZE VERBOSE stat1 ;  
INFO:  analyzing "public.stat1"  
INFO:  "stat1": scanned 542 of 542 pages, containing 100100 live rows  
and 0 dead rows; 30000 rows in sample, 100100 estimated total rows  
ANALYZE  
postgres=>
```

When ANALYZE VERBOSE statement is executed, not only the number of sampling tuples, but also the number of the pages and the number of active tuples, the number of the dead tuples, the number of tuples of the entire expected table will be output.

□ Change of STATISTICS value of the column

Column STATISTICS settings will be executed in the ALTER TABLE ALTER COLUMN statement. The upper limit of the STATISTICS setting is 10,000.

Example 126 Column STATISTICS setting and confirmation

```
postgres=> ALTER TABLE data1 ALTER COLUMN col1 SET STATISTICS 200 ;
ALTER TABLE
postgres=> \d+ data1
```

Column	Type	Modifiers	Storage	Stats target	Desc
c1	numeric		main	<u>200</u>	
c2	character varying(100)		extended		

Has OIDs: no

5.1.4 Information collected as statistics

Statistics that are collected by the execution of ANALYZE statement or automatic VACUUM, are stored in the pg_statistic catalog and pg_class catalog. For pg_statistic catalog's format is difficult to use information is normally searched from pg_stats catalog. The tuples in a table, where ANALYZE statement (including automatic VACUUM) is not executed, are not included in this catalog.

Table 53 pg_stats catalog

Attribute	Description
schemaname	schema name
tablename	table name
attname	attribute name
inherited	If true, including the information of the inheritance table
null_frac	Percentage of NULL row
avg_width	Average byte length of the column
n_distinct	Estimated unique tuple number (positive or negative)
most_common_vals	Most frequent value
most_common_freqs	Rate of most frequent value
histogram_bounds	Percentage of the histogram appearance
correlation	Correlation between physical layout and logical placement
most_common_elems	Most value is large column value (non-scalar value)
most_common_elem_freqs	The proportion of most value often column value (non-scalar value)
elem_count_histogram	Percentage of the histogram appearance (non-scalar value)

Table 54 Statistics of pg_class catalog

Attribute	Description
reltuples	The number of tuples included in the table
relpages	Number of pages of the table

□ N_distinct statistics

N_distinct column of pg_stats catalog shows the number of the unique values in the table. This value may become a negative value.

- As a result of sampling, if the column value is determined to be unique, n_distinct value is specified -1.0.
- If the number of the unique values is considered to be more than 10% of the total number of tuples, the following calculations are specified.

Formula 5 Number of Unique Value

$$-1 * (\text{Estimate of the unique value} / \text{Total number of tuples})$$

- Otherwise, the calculation formula in the comment below is used as an estimated value.

Example 127 comments indicating the n_distinct formula (src/backend/commands/analyze.c)

Estimate the number of distinct values using the estimator proposed by Haas and Stokes in IBM Research Report RJ 10025:

$$n*d / (n - f1 + f1*n/N)$$

where f1 is the number of distinct values that occurred exactly once in our sample of n rows (from a total of N), and d is the total number of distinct values in the sample. This is their Duj1 estimator; the other estimators they recommend are considerably more complex, and are numerically very unstable when n is much smaller than N.

Overwidth values are assumed to have been distinct.

The value of n_distinct statistics can be overwritten by the ALTER TABLE table_name ALTER column_name SET statement, but the value on the pg_stats catalogs does not change until the next ANALYZE statement is executed.

For the attributes of column, `n_distinct` and `n_distinct_inherited` can be specified.

`N_distinct_inherited` is the column information of inherited table, but you can change it even if you are not using the inheritance table. Column attributes that have been changed in the `ALTER TABLE` statement can be confirmed by `pg_attribute` catalog.

□ `Most_common_vals` statistics

`Most_common_vals` Statistics (MCV) is an array of the column value which appears the most frequently. Maximum number of elements in the array is `default_statistics_target` parameter values (default 100), or `STATISTICS` value of the column, if specified. Thus, when the parameter `default_statistics_target` is expanded, the number of the buckets in histogram, the number of the sampling tuples, and the number of the elements in the MCV are also expanded.

5.1.5 Destination of the statistics

In addition to the per-object statistics, various statistical information is automatically collected in PostgreSQL. They can be checked by `pg_stat_*` catalogs¹⁰ and `pg_statio_*` catalogs.

Table 55 Statistics Information Catalog

Catalog name	Contents
<code>pg_stat_activity</code>	Current activity of that process
<code>pg_stat_{all sys user}_indexes</code>	Statistics for the index
<code>pg_stat_{all sys user}_tables</code>	Statistics for the table
<code>pg_stat_archiver</code>	Statistical information about the archive log
<code>pg_stat_bgwriter</code>	Statistical information about the writer process
<code>pg_stat_database</code>	Statistics for each database
<code>pg_stat_database_conflicts</code>	Competitive information with the standby database
<code>pg_statio_{all sys user}_sequences</code>	I/O statistics on sequence object
<code>pg_statio_{all sys user}_tables</code>	I/O statistics on the table
<code>pg_statio_{all sys user}_indexes</code>	I/O statistics on the index
<code>pg_statistic</code>	I/O statistics on the all objects
<code>pg_stat_replication</code>	Statistical information about the replication
<code>pg_stats</code>	Formatting catalog of <code>pg_statistic</code>
<code>pg_stat_ssl</code>	SSL usage of client
<code>pg_stat_user_functions</code>	Statistics for the functions
<code>pg_stat_xact_{all sys user}_tables</code>	Updated statistical information for the table
<code>pg_stat_xact_user_functions</code>	Updated statistics for function
<code>pg_stat_wal_receiver</code>	Slave information of the replication environment
<code>pg_stat_progress_vacuum</code>	VACUUM execution status

The real content of the statistics consists of `global.stat` file where statistics of whole instance is stored, and `db_{OID}.stat` file created for each database. At the instance startup, these files are moved from `{PGDATA}/pg_stat` directory to the directory specified by the parameter `stats_temp_directory` (default: `pg_stat_tmp`). It is returned to the `pg_stat` directory at the instance shutdown.

If the read of the statistics is bottleneck, we can make it faster by specifying the parameter `stats_temp_directory` to a directory on a fast storage.

¹⁰ Online manual is <https://www.postgresql.org/docs/9.6/static/monitoring-stats.html>

5.2 Automatic VACUUM

5.2.1 Interval

Automatic VACUUM is performed by the autovacuum worker process which is activated by autovacuum launcher process. The activation intervals of autovacuum worker is decided by the parameter `autovacuum_naptime` (default: 1min).

5.2.2 Conditions

Automatic VACUUM is determined by comparing the value acquired by the following equation, and the number of the tuples that are updated and become unnecessary. Though the number of the unnecessary tuples essentially matches the number of the tuples updated by the UPDATE / DELETE statement, it may not be counted in case of multiple UPDATE for the same tuple. This is assumed to be caused by re-use using HOT. Condition check is done in `relation_needs_vacanalyze` function in the source code (`src/backend/postmaster/autovacuum.c`).

Formula 6 Autovacuum Threshold

$$\text{Threshold} = \text{autovacuum_vacuum_threshold} + \text{autovacuum_vacuum_scale_factor} * \text{reltuples}$$

The meaning of the above equation is as follows:

- `autovacuum_vacuum_threshold`
The value of the parameter `autovacuum_vacuum_threshold` (default value is 50), or the value of the `autovacuum_vacuum_threshold` attribute of the table.
- `autovacuum_vacuum_scale_factor`
The value of the parameter `autovacuum_analyze_scale_factor` (default: 0.2 = 20%), or the value of the `autovacuum_analyze_scale_factor` attributes of the table.
- `reltuples`
This is the number of the valid tuples when the statistics were acquired last time of the time. It can be found in `pg_class` catalog `reltuples` column.

5.2.3 Autovacuum worker process startup

Actual VACUUM processing is executed by the autovacuum worker process that is regularly started. Autovacuum worker process is a child process of the postmaster process. The tables which need VACUUM are checked at the intervals of "`autovacuum_naptime` parameter / number of the data changed database", and when the table which contains the unnecessary tuples beyond the threshold

described above is found, autovacuum worker process is activated. Autovacuum worker process stops when it completes the VACUUM processing in the target database.

If the tables to be performed VACUUM exists in multiple databases, autovacuum worker processes start for each database. If there still remains VACUUM target table at the next check interval, VACUUM process is performed by starting a new autovacuum worker process.

The maximum number of autovacuum worker process is limited by the parameter `autovacuum_max_workers` (default: 3). The log is not output even if it reaches the maximum value.

Execution status of automatic VCAUUM can be found in the `pg_stat_progress_vacuum` catalog. However, you can search the records only during VACUUM processing execution.

Example 128 Retrieval of `pg_stat_progress_vacuum` catalog

```
postgres=# SELECT * FROM pg_stat_progress_vacuum ;
-[ RECORD 1 ]-----+-----
pid           | 3184
datid         | 16385
datname       | demodb
relid         | 16398
phase         | scanning heap
heap_blks_total | 10052
heap_blks_scanned | 2670
heap_blks_vacuumed | 2669
index_vacuum_count | 0
max_dead_tuples | 291
num_dead_tuples | 185
```

5.2.4 Amount of usable memory

The parameter that controls the size calculation of the memory area to be used for the automatic VACUUM processing is `autovacuum_work_mem`. The value specified by this parameter is not always used, but the calculated value based on a parameter value is used.

The default value for this parameter is -1. In case of default value, the value of the parameter `maintenance_work_mem` is used. It is not documented in the manual, but the lower limit value for this parameter is 1024 (= 1MB). If the value other than -1 and less than 1MB is specified, the parameter value is set to 1MB.

5.3 Execution Plan

5.3.1 EXPLAIN statement

In order to view the execution plan of PostgreSQL, use the EXPLAIN statement. By specifying ANALYZE clause to EXPLAIN statement, the estimate statistics calculated at runtime planning and the result statistics of the SQL statements are displayed side by side.

Example 129 Execution of EXPLAIN statement

```
postgres=> EXPLAIN ANALYZE SELECT * FROM data1 WHERE c1 BETWEEN 1000 AND 2000 ;
               QUERY PLAN
-----
Index Scan using pk_data1 on data1 (cost=0.29..8.31 rows=1 width=11)
    (actual time=0.033..0.033 rows=0 loops=1)
    Index Cond: ((c1 >= '1'::numeric) AND (c1 <= '1000'::numeric))
    Planning time: 9.849 ms
    Execution time: 1.691 ms
(4 rows)
```

Table 56 Example of the output result of the EXPLAIN statement

Output	Description	Note
Index Scan using	The object name of the execution plan and the target	
cost	The calculated cost (details will be described later)	
rows	Estimated number of tuples	
width	Estimated typical output number of bytes (width of 1 tuple)	
actual time	The actual execution time (details will be described later)	
rows	The number of output tuples	
loops	Number of repetitions of the process	
Index Cond:	Indicate that it has performed the partial search of the index	
Planning time:	Expected time (ms)	Added 9.4
Execution time:	Total execution time (ms)	

The EXPLAIN statement does not display which planner, the dynamic programming (DP) or the genetic optimization (GEQO), was used to perform planning. And the execution plan of SQL statements that are executed within stored procedures written in PL/pgSQL, etc. is not displayed.

5.3.2 Costs

Cost part displayed with the EXPLAIN statement is the cost required to execute the SQL statement. Cost is a relative estimate value when the cost to read one page (8 KB) in sequential I/O is defined as 1.0.

Example 130 Cost outputted by EXPLAIN statement

```
cost=0.00..142.10
```

The first number is the start-up cost, and the second number is the total cost. Start-up cost is used in case of using some of the operators. The total cost should be noted in the tuning.

Table 57 The major execution plan and the default cost

Parameter	Description	Default Value
seq_page_cost	Sequential page read	1.0
cpu_index_tuple_cost	Processing of index once	0.005
cpu_operator_cost	Calculation once	0.0025
cpu_tuple_cost	Operation of one tuple	0.01
random_page_cost	Random page read	4.0
parallel_setup_cost	Parallel query initial cost	1000
parallel_tuple_cost	Tuple cost of parallel query	0.1

Example 131 Display of cost

```
postgres=> EXPLAIN SELECT * FROM data1 ;
               QUERY PLAN
-----
Seq Scan on data1 (cost=0.00..2133.98 rows=89998 width=41)
(1 row)

postgres=> SELECT reltuples, relpages FROM pg_class WHERE relname='data1' ;
 reltuples | relpages
-----+-----
    89998 |     1234
(1 row)
```

Calculated value of the cost in the above example is the relpages (1,234) × seq_page_cost (1.0) + reltuples (89,998) × cpu_tuple_cost (0.01) = 2,133.98.

5.3.3 Execution plan

The execution plan that is output with the EXPLAIN command outputs the following query operators. This list is searched from the source code of PostgreSQL 9.6.2 (src/backend/commands/explain.c).

Table 58 Execution Plan Operator

Query Operator	Behavior	Startup Cost
Result	Non-table query	No
Insert	Execution of INSERT statement	
Delete	Execution of DELETE statement	
Update	Execution of UPDATE statement	
Append	Additional processing of data	No
Merge Append	Merge processing	
Recursive Union	Recursive Union	
BitmapAnd	Bit map search AND	
BitmapOr	Bit map search OR	
Nested Loop	Nested Loops	No
Merge Join	Merge Join	Yes
Hash Join	Hash Join	Yes
Seq Scan	All cases search	No
Sample Scan	Sample Scan	
Index Scan	Index range search	No
Index Only Scan	Range search of the index only	
Bitmap Index Scan	Bitmap scan of index	
Bitmap Heap Scan	Bitmap scan of the heap	Yes
Tid Scan	TID scan plan	No
Subquery Scan	Subquery Search	No
Function Scan	Function scan	No
Values Scan	Value scan	
CTE Scan	CTE scan using WITH clause	
WorkTable Scan	Temporary table search	
Foreign Scan	External table search	
Materialize	Subquery	Yes
Custom Sacn	Custom Scan	
Sort	Sorting	Yes

Table 58 Execution Plan (Cont.)

Query Operator	Behavior	Startup Cost
Group	Processing of GROUP BY clause	Yes
Aggregate	Use of aggregate processing	Yes
GroupAggregate	Grouping	
HashAggregate	Use of hash aggregation processing	
WindowAgg	Window aggregation processing	
Unique	processing of DISTINCT / UNION clause	Yes
SetOp	processing of INTERCEPT / EXCEPT clause	Yes
HashSetOp	Hashing	
LockRows	Row locking	
Limit	Processing of LIMIT clause	Yes (OFFSET > 0)
Hash	Hashing	Yes
Parallel Seq Scan	Parallel sequential scan	
Finalize Aggregate	Final consolidation of parallel processing	
Gather	Aggregation of parallel workers	
Partial Aggregate	Parallel processing of the aggregate	
Partial HashAggregate		
Partial GroupAggregate		
Single Copy	Run in a single process	

Below is the description of the typical query operator.

□ Sort

In addition to the explicit specification by ORDER BY clause, it can be performed in the implicit sort due to Merge Join processing or Unique processing.

□ Index Scan

Find the table from the index. It refers to the full scan of the index if the "Index Cond" execution plan does not appear.

□ Index Only Scan

To get all the necessary information from the index, it does not to search for the table. However, a result of referring to the visibility map, it may executor to access the table.



□ Bitmap Scan

Use the BitmapOr and BitmapAnd to create a bit map of the entire relationship in memory.

□ Result

This message is displayed when returning the results without accessing the table.

□ Unique

It will eliminate duplicate values. It is used in the processing of DISTINCT clause or UNION clause.

□ Limit

It is used when the LIMIT clause is specified with the ORDER BY clause.

□ Aggregate

It is used in the GROUP BY clause or aggregate functions. GroupAggregate or HashAggregate might be used.

□ Append

It is used in the data append processing by UNION (ALL).

□ Nested Loop

It is used in the INNER JOIN and LEFT OUTER JOIN. It scans the external table, and search the tuples that match the internal table.

□ Merge Join

There are Merge Right Join and Merge In Join. It binds a sorted record set.

□ Hash, Hash Join

Create a hash table, and then compare the two tables. The initial cost for the creation of the hash table is required.

□ Tid Scan

It is used in the search that specifies Tuple Id (ctid).

□ Function

It is used when a function generates a tuple (SELECT * FROM func (), etc.).

□ SetOp

It is used in the processing of EXCEPT ALL clause, INTERSECT clause, INTERSECT ALL clause, and EXCEPT clause.

Parameters that control the operator planner to create an execution plan to select is as follows. Setting value of these parameters is turned "on" by default. It is set to "off" these parameters, the operator specified does not mean that it is completely prohibited. If you specify in the parameter "off", the execution plan to the startup cost 1.0e10 (10,000,000,000) has been added are compared.

Table 59 Parameters that control the execution plan

Parameter name	Description	Default Value
enable_bitmapscan	Use of bitmap scan	on
enable_indexscan	Use of index scan	on
enable_tidscan	Use of TID scan	on
enable_seqscan	Use of sequential scan	on
enable_hashjoin	Use of hash join	on
enable_mergejoin	Use of merge join	on
enable_nestloop	Use of nested loop join	on
enable_hashagg	Use of hash aggregate	on
enable_material	Use of materialization	on
enable_sort	Use of sort	on

In the following example, in an environment where only sequential scan is performed, to set the value of the parameter enable_seqscan to "off". The initial cost will be larger, but you can see that the sequential scan (Seq Scan) has been selected.



Example 132 Change the parameters and execution plan

```
postgres=> SHOW enable_seqscan ;
enable_seqscan
-----
on
(1 row)
postgres=> EXPLAIN SELECT * FROM data1 ;
               QUERY PLAN
-----
Seq Scan on data1  (cost=0.00..7284.27 rows=466327 width=11)
(1 row)

postgres=> SET enable_seqscan = off ;
SET
postgres=> EXPLAIN SELECT * FROM data1 ;
               QUERY PLAN
-----
Seq Scan on data1  (cost=10000000000.00..10000007284.27 rows=466327 width=11)
(1 row)
```

5.3.4 Execution time

If a ANALYZE clause is specified to EXPLAIN statement, the actual execution time is output. In the "actual" part, there are two numbers which indicate time. The first number is the time when the first tuple is output, and the second number is the total execution time.

Example 133 Display of execution time

```
actual time=0.044..0.578
```

5.3.5 Cost estimate of the empty table

When executing the EXPLAIN statement to conduct a search on a table with no tuple the cost item and rows item show the numbers different from the actual number. Empty table is calculated as 10 blocks, and the system calculates the cost by estimating the maximum number of tuples that can be stored in 10 blocks.

Example 134 Display of rows and cost for empty table

```
postgres=> CREATE TABLE data1 (c1 NUMERIC, c2 NUMERIC) ;
CREATE TABLE
postgres=> ANALYZE data1 ;
ANALYZE
postgres=> SELECT reltuples, relpages FROM pg_class WHERE relname='data1' ;
 reltuples | relpages
-----+-----
          0 |          0
(1 row)
postgres=> EXPLAIN SELECT * FROM data1 ;
              QUERY PLAN
-----
Seq Scan on data1  (cost=0.00..18.60 rows=860 width=64)
(1 row)
postgres=>
```

5.3.6 Disk sort

Sorting process is performed in memory that has been specified by the parameter `work_mem`. However, if tuples cannot be stored in working memory, sorting is performed on the storage. Temporary data for the disk sorting, is created in the `pgsql_tmp` directory of tablespace where the database, to which the table belongs, is stored. If the destination of the database is tablespace `pg_default`, `{PGDATA}/base/pgsql_tmp` directory is used; otherwise `{TABLESPACEDIR}/PG_9.6_201608131/pgsql_tmp` directory is used. File name that is used for sorting is "pgsql_tmp{PID}.{9}". {PID} is the backend process ID, and {9} is a unique number that starts from 0.

Example 135 File for disk sorting

```
$ pwd
/usr/local/pgsql/data/base/pgsql_tmp
$ ls -l
total 34120
-rw----- 1 postgres postgres 34897920 Sep 30 17:02 pgsql_tmp6409.0
```

Following is the confirmation method of the disk sorting.

□ Execution plan

When the execution plan is obtained by EXPLAIN ANALYZE statement, "Sort Method: external merge" indicating the disk sort or "Sort Method: external sort", and the amount of disk space that is used is output.

Example 136 Execution plan of disk sort

```
postgres=> EXPLAIN ANALYZE SELECT * FROM data1 ORDER BY 1, 2 ;
               QUERY PLAN
-----
Sort  (cost=763806.52..767806.84 rows=1600128 width=138)
      (actual time=7600.693..9756.909 rows=1600128 loops=1)
    Sort Key: c1, c2
    Sort Method: external merge  Disk: 34080kB
    -> Seq Scan on data1  (cost=0.00..24635.28 rows=1600128 width=138)
      (actual time=1.239..501.092 rows=1600128 loops=1)
Total runtime: 9853.630 ms
(5 rows)
```

By the number of tuples for sorting, the method of disk sort is selected Replacement Selection or Quicksort. When the number of tuples sort object is less than or equal to the specified parameters replacement_sort_tuples (default value 150000), Replacement Selection will be selected.

□ Parameter trace_sort

Specifying a parameters trace_sort to "on", sort-related event is logged in the log (default: "off"). This parameter causes log output, even during execution of memory sort, therefore it should not be set in a commercial environment.



Example 137 Output log with trace_sort=on (sorted by Replacement Selection)

```
LOG: statement: SELECT * FROM data1 ORDER BY 1;
LOG: begin tuple sort: nkeys = 1, workMem = 4096, randomAccess = f
LOG: numeric_abbrev: cardinality 10049.436974 after 10240 values (10240 rows)
LOG: switching to external sort with 15 tapes: CPU 0.03s/0.01u sec elapsed 0.04
sec
LOG: replacement selection will sort 58253 first run tuples
LOG: performsort starting: CPU 0.99s/0.51u sec elapsed 1.51 sec
LOG: finished incrementally writing only run 1 to tape 0: CPU 1.01s/0.54u sec
elapsed 1.55 sec
LOG: performsort done: CPU 1.01s/0.54u sec elapsed 1.56 sec
LOG: external sort ended, 2687 disk blocks used: CPU 1.50s/0.95u sec elapsed
2.48 sec
```

Example 138 Output log with trace_sort=on (Sorted by Quicksort)

```
LOG: begin tuple sort: nkeys = 1, workMem = 4096, randomAccess = f
LOG: numeric_abbrev: cardinality 10049.436974 after 10240 values (10240 rows)
LOG: switching to external sort with 15 tapes: CPU 0.01s/0.01u sec elapsed 0.02
sec
LOG: starting quicksort of run 1: CPU 0.01s/0.01u sec elapsed 0.02 sec
LOG: finished quicksort of run 1: CPU 0.01s/0.01u sec elapsed 0.03 sec
LOG: finished writing run 1 to tape 0: CPU 0.02s/0.01u sec elapsed 0.04 sec
LOG: performsort starting: CPU 0.31s/0.25u sec elapsed 0.56 sec
LOG: starting quicksort of run 20: CPU 0.31s/0.25u sec elapsed 0.56 sec
LOG: finished quicksort of run 20: CPU 0.31s/0.25u sec elapsed 0.56 sec
LOG: finished writing run 20 to tape 5: CPU 0.31s/0.25u sec elapsed 0.57 sec
LOG: finished 7-way merge step: CPU 0.45s/0.36u sec elapsed 0.82 sec
LOG: grew memtuples 1.29x from 58253 (1366 KB) to 74896 (1756 KB) for final
merge
LOG: tape 0 initially used 144 KB of 144 KB batch (1.000) and 4581 out of 5348
slots (0.857)
LOG: performsort done (except 14-way final merge): CPU 0.45s/0.37u sec elapsed
0.82 sec
LOG: external sort ended, 2679 disk blocks used: CPU 2.07s/0.40u sec elapsed
2.48 sec
```

□ `pg_stat_database` catalog

In `pg_stat_database` catalog temporary file information is stored. These values are not only disk sort by `ORDER BY` clause, but also disk sorting of indexing by `CREATE INDEX` statement.

Table 60 Temporary files related data in `pg_stat_database` catalog

Column name	Description
<code>datname</code>	database name
<code>temp_files</code>	Number of the created temporary files
<code>temp_bytes</code>	Total size of the created temporary files

5.3.7 Table sequential scan and index scan

Sequential scan, which access the entire table, and index scan are compared using the system call issued by postgres process. This is verified immediately after instance startup and no data exists in the shared buffer. In the following example, the system calls are traced by performing `SELECT * FROM data1` statement for the sequential scan, and `SELECT * FROM data1 BETWEEN c1 10000 AND 2000` statement for the index scan. The file of the table `data1` is "base/16499/16519", and the file of index `idx1_data1` is "base/16499/16535".

□ Sequential Scan

In the sequential scan, postgres process read the tuples from the beginning of the table one block by one, and after reading a few blocks, postgres process sends the tuple to the client. Postgres process does not read multi-block at once. The reason for reading the beginning of the index, even though it does not use an index, is assumed for execution planning decision.



Example 139 System calls of sequential scan

```

open("base/16499/16519", O_RDWR)      = 27      -- open file for table
lseek(27, 0, SEEK_END)                  = 88563712

open("base/16499/16525", O_RDWR)      = 29      -- open index file
lseek(29, 0, SEEK_END)                  = 67477504
lseek(29, 0, SEEK_SET)                  = 0
read(29, "\0\0\0\0h\342\251e\0\0\60\37\4 \0\0\0\0b1\5\0\2\0\0\0"... , 8192) = 8192

lseek(27, 0, SEEK_END)                  = 88563712
lseek(27, 0, SEEK_SET)                  = 0      -- move to first block of table
read(27, "\1\0\0\0\020\374\2\30\3\0 \4 \0\0\0\0\330\0\260\237D\0"... , 8192) = 8192
read(27, "\1\0\0\0\200S\270\50\3\0 \4 \0\0\0\0\330\237D\0\237D\0"... , 8192) = 8192 -
-- read 1 block
read(27, "\1\0\0\0\360s\270\5\0\0\5\0 \4 \0\0\0\0\330\230\237D\0"... , 8192) = 8192 -
-- read 1 block
sendto(10, "T\0\0\0000\0\2c1\0\0\0@\207\0\1\0\0\67\0"... , 8192, 0, NULL, 0) = 8192 -
-- send block to client
read(27, "\1\0\0\0\` \224\20\0\74\2\30\3\0 \4 \0\0\0\0\3260\237D\0"... , 8192) = 8192 -
-- read 1 block
read(27, "\1\0\264\270\5\0\0\4\2\30\3\0 \4 \0\0\0\0\330\23237D\0"... , 8192) = 8192 -
-- read 1 block
sendto(10, "\0\25\0\2\0\0\0\003556\0\01D\0\0\0\00355"... , 8192, 0, NULL, 0) = 8192 -
-- sent block to client

```

□ Index Scan

In the index scan, postgres process repeats "reading of the index" and "reading a table". For this reason lseek system call and read system call are repeated.



Example 140 System calls of index scan

```
open("base/16499/16519", O_RDWR)      = 36  -- open file for table
lseek(36, 0, SEEK_END)                  = 88563712

open("base/16499/16525", O_RDWR)      = 38  -- open file for index
lseek(38, 0, SEEK_END)                  = 67477504
lseek(38, 0, SEEK_SET)                  = 0
read(38, "\0\0\0h\342\251e\0\0\360\37\360\37\4 \05\0\2\0\0\0"... , 8192) = 8192

lseek(38, 2375680, SEEK_SET)            = 2375680 -- move in index file and read
read(38, "\0\0\033\304\0\260\230\35\360\37\4 0\0\350\20H\237 \0"... , 8192) = 8192
lseek(38, 24576, SEEK_SET)              = 24576  -- move in index file and read
read(38, "\0\0\0\0\210if\0L\3(\23\360\37\4 \340\237 \0\337\20\0"... , 8192) = 8192
lseek(38, 237568, SEEK_SET)            = 237568 -- move in index file and read
read(38, "\1\0\0\0\330\2302\v200\26\360\37\4 340\237 \0\0\237 \0"... , 8192) = 8192

lseek(36, 434176, SEEK_SET)            = 434176 -- move in table file and read
read(36, "\1\0X\352\276\374\2\30\3\0 \4 \330\237D\0\260\237D\0"... , 8192) = 8192

sendto(10, "T\0\0\0\33\02\0\0\4\23\377\16\0\0D\0\0"... , 8192, 0, NULL, 0) = 8192
-- send data to client
```

5.3.8 BUFFERS parameter

Specifying BUFFERS option to EXPLAIN statement in conjunction with the ANALYZE option, the buffer information acquired at the time of execution is output. Information to be output are buffer I/O information for each category; shared buffer (shared), local buffer (local), and temporary segment (temp).

Example 141 BUFFERS option

```
postgres=> EXPLAIN (ANALYZE true, BUFFERS true) SELECT * FROM stat1 s1
           ORDER BY c1, c2 ;

           QUERY PLAN

-----
Sort  (cost=1041755.43..1057768.57 rows=6405258 width=10)
      (actual time=20067.420..25192.887 rows=6406400 loops=1)
      Sort Key: c1, c2
      Sort Method: external merge  Disk: 130288kB
      Buffers: shared hit=16153 read=18477, temp read=33846 written=33846
-> Seq Scan on stat1 s1  (cost=0.00..98682.58 rows=6405258 width=10)
      (actual time=0.290..751.019 rows=6406400 loops=1)
      Buffers: shared hit=16153 read=18477

Planning time: 0.079 ms
Execution time: 25535.583 ms
(8 rows)
```

The items output after "Buffers:" are as follows. All of the printed numbers are number of blocks (8 KB units).

Table 61 Buffers: output items

Category	Item	Description
shared	hit	Shared buffer cache hit
	read	Shared buffer cache miss
	dirtied	Reading from dirty buffers
	written	Shared buffer cache write
local	hit	Local buffer cache hit
	read	Local buffer cache miss
	dirtied	Reading from dirty buffers
	written	Local buffer cache write
temp	read	Reading of temporary segments
	written	Writing of temporary segments

5.4 Configuration Parameters

5.4.1 Parameters related to performance

Major parameters related to PostgreSQL server performance is as follows.

Table 62 Parameters related to performance

Parameter name	Description	Note
autovacuum_work_mem	Memory size of automatic Vacuum	9.4 or later
effective_cache_size	The effective capacity of the disk cache that is available to a single query	
effective_io_concurrency	The number of concurrent disk I / O operation	
huge_pages	Use the Huge Pages or not	9.4 or later
maintenance_work_mem	The memory for the maintenance work such as VACUUM, CREATE INDEX, etc.	
shared_buffers	Shared buffer size	
temp_buffers	Memory size for temporary table	
wal_buffers	Shared memory where WAL information is stored	
work_mem	Temporary memory for hash, and sort	
max_wal_size	WAL write amount which becomes a generation opportunity of checkpoint	9.5 or later
replacement_sort_tuples	The maximum number of tuples to perform Replacement Selection in external sort	9.6

5.4.2 Effective_cache_size parameter

This parameter specifies the total size of a shared buffer and the cache used by the OS. It is mainly used as a parameter to calculate the cost of the index scan at the decision of the execution planning. It will be used in gistInitBuffering function (src/backend/access/gist/gistbuild.c) and index_pages_fetched function (src/backend/optimizer/path/costsize.c).

5.4.3 Effective_io_concurrency parameter

The value specified for this parameter is copied to the GUC target_prefetch_pages. The specified value is used to calculate the number of prefetch during the processing of the execution plan bitmap heap scan. Description of this parameter in the manual is obscure, and the effect of the change has not been verified. As the parameter increases, prefetch number becomes bigger. It is not used in other than



the bitmap heap scan. The manual of PostgreSQL 9.6, the following description has been added.

“SSDs and other memory-based storage can often process many concurrent requests, so the best value might be in the hundreds.”

5.5 System Catalog

5.5.1 Entity of the system catalog

System catalog views whose name begins with "pg_" is explained in manual as "PostgreSQL's system catalogs are regular tables", but the catalogs, the name of which begins with "pg_stat", and get the execution statistics, provides information from different data sources. From the execution plan, the entities are verified.

Table 63 Entity of the system catalog

System Catalog	Retrieve information from:
pg_statio_*_indexes	pg_namespace, pg_class, pg_index, pg_stat_*
pg_statio_*_sequences	pg_namespace, pg_class, pg_stat_*
pg_statio_*_tables	pg_namespace, pg_index, pg_class, pg_stat_*
pg_stat_activity	pg_database, pg_authid, pg_stat_get_activity()
pg_stat_archiver	pg_database, pg_stat_get_db_*
pg_stat_bgwriter	pg_stat_get_*
pg_stat_database	pg_database, pg_stat_get_*
pg_stat_database_conflicts	pg_database, pg_authid, pg_stat_get_*
pg_stat_replication	pg_authid, pg_stat_get_activity(), pg_stat_get_wal_senders()
pg_stat_*_indexes	pg_namespace, pg_class, pg_index, pg_stat_get_*
pg_stat_*_tables	pg_namespace, pg_class, pg_index, pg_stat_get_*
pg_stat_*_functions	pg_namespace, pg_proc, pg_stat_get_function_*
pg_stat_xact_*_tables	pg_namespace, pg_class, pg_index, pg_stat_get_xact_*
pg_stat_xact_*_functions	pg_namespace, pg_proc, pg_index, pg_stat_get_xact_*
pg_stat_xact_*_tables	pg_namespace, pg_index, pg_stat_get_xact_*
pg_statistic	table
pg_stats	pg_namespace, pg_class, pg_statistic, pg_attribute, has_column_privilege()
pg_stat_progress_vacuum	pg_stat_get_progress_info(), pg_database
pg_stat_wal_receiver	pg_stat_get_wal_receiver()

6. Specification of SQL statements

6.1 Lock

6.1.1 Lock type

PostgreSQL gets the lock automatically to keep the integrity of the tuple on the table. Sometimes applications make an explicit lock by executing LOCK TABLE statement or SELECT FOR UPDATE statement. Usually locks are held until the transaction is finalized (COMMIT or ROLLBACK). More information about the lock is described in the manual¹¹.

Table 64 Lock type

Lock name	Description
ACCESS SHARE	On running SELECT statement, it will be acquired for the target table. It becomes the weakest lock.
ROW SHARE	On running SELECT FOR UPDATE / SELECT FOR SHARE statement it will be acquired for the table.
ROW EXCLUSIVE	This is a lock which UPDATE, DELETE, and INSERT, statements get. ACCESS SHARE is also acquired for the referenced table.
SHARE UPDATE EXCLUSIVE	It will be acquired by VACUUM, ANALYZE, CREATE INDEX CONCURRENTLY and ALTER TABLE statement.
SHARE	It will be acquired by CREATE INDEX statement which has no CONCURRENTLY clause.
SHARE ROW EXCLUSIVE	It is not acquired automatically.
EXCLUSIVE	It is not acquired automatically.
ACCESS EXCLUSIVE	It will be acquired by ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER and VACUUM FULL statement.

6.1.2 Acquisition of lock

Unlike other database such as Oracle Database, PostgreSQL acquires the ACCESS SHARE lock even which a simple SELECT statement. If IN clause is not specified in the LOCK TABLE statement, the ACCESS EXCLUSIVE lock is acquired. For this reason, when LOCK TABLE statement is executed to a table, no access can be performed to the table including SELECT statement. The current lock

¹¹ Online Manual <http://www.postgresql.org/docs/9.6/static/explicit-locking.html>

situation can be confirmed from pg_locks catalog.

Example 142 LOCK TABLE statement and lock status

```
postgres=> BEGIN ;
BEGIN
postgres=> LOCK TABLE data1 ;
LOCK TABLE
postgres=>
-- from another session
postgres=> SELECT locktype, relation, mode FROM pg_locks ;
  locktype  | relation |      mode
-----+-----+-----
virtualxid |          | ExclusiveLock
relation   |    16519 | AccessExclusiveLock
```

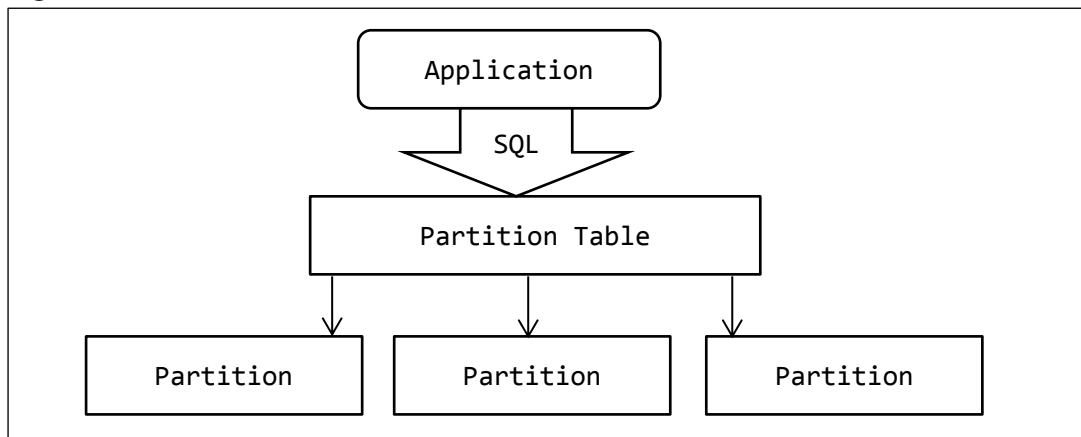
ACCESS EXCLUSIVE LOCK conflicts with all other locks. For this reason, the table, which is performing the process acquiring this lock, e.g., VACUUM FULL, cannot be accessed even if processing the search.

6.2 Partition Table

6.2.1 Partition Table Overview

Partition table is a function which divides a large-scale table into multiple partition and reduce the I/O range in order to improve the performance and maintainability. Since partition is hidden from applications, it looks like a single table.

Figure 15 Partition Table



In general, the tuple in the partition table is automatically determined to which partition it should be stored, by the scope and fixed value of the column values.

6.2.2 Partition Table Implementation

Unlike the Oracle Database and Microsoft SQL Server, etc., PostgreSQL does not have the function of the partition table that can be used for native. It implements the function of the partition table by combining the inheritance of the table, CHECK constraints and triggers.

Partition table in PostgreSQL will be created in the following procedure.

- Create a parent table (table that the application access via SQL)
- Creating an inheritance table from parent that contains a check constraint (partition)
- Create a function for trigger
- Create an INSERT trigger to the parent table and register a trigger for the function
- Execute SQL statements for the parent table (application issue SQL)

Following example divides the main1table into three partitions (main1_part100, main1_part200, main1_part300) by the range of key1 row.



Example 143 Create parent table

```
postgres=> CREATE TABLE main1 (key1 NUMERIC, val1 VARCHAR(10), val2  
VARCHAR(10)) ;  
CREATE TABLE
```

It creates an inheritance table that becomes the partition. Specify the parent table in INHERITS clause to specify a CHECK constraint on the tuple included in the partition.

Example 144 Create inherit tables (Partition)

```
postgres=> CREATE TABLE main1_part100 (  
CHECK(key1 < 100)  
) INHERITS (main1) ;  
CREATE TABLE  
postgres=> CREATE TABLE main1_part200 (  
CHECK(key1 >= 100 AND key1 < 200)  
) INHERITS (main1) ;  
CREATE TABLE  
postgres=> CREATE TABLE main1_part300 (  
CHECK(key1 >= 200 AND key1 < 300)  
) INHERITS (main1) ;  
CREATE TABLE
```

Next example creates a function to be used for the trigger.

Example 145 Create the FUNCTION for the TRIGGER

```
postgres=> CREATE OR REPLACE FUNCTION func_main1_insert()  
          RETURNS TRIGGER AS $$  
          BEGIN  
            IF      (NEW.key1 < 100) THEN  
                INSERT INTO main1_part100 VALUES (NEW.*) ;  
            ELSIF (NEW.key1 >= 100 AND NEW.key1 < 200) THEN  
                INSERT INTO main1_part200 VALUES (NEW.*) ;  
            ELSIF (NEW.key1 < 300) THEN  
                INSERT INTO main1_part300 VALUES (NEW.*) ;  
            ELSE  
                RAISE EXCEPTION 'ERROR! key1 out of range.' ;  
            END IF ;  
            RETURN NULL ;  
          END ;  
          $$  
          LANGUAGE plpgsql ;  
CREATE FUNCTION
```

The last examples registers the function created by CREATE FUNCTION statement into the trigger.

Example 146 Create the TRIGGER

```
postgres=> CREATE TRIGGER trg_main1_insert  
          BEFORE INSERT ON main1  
          FOR EACH ROW EXECUTE PROCEDURE func_main1_insert() ;  
CREATE TRIGGER
```

6.2.3 Verify the execution plan

The execution plans for the partition table are verified here.

- The partition selection by the SELECT statement

If there is a syntax that can identify the partition in WHERE clause, it accesses automatically inherits only the table. However, if the calculation (left-hand side is calculated value, etc.) are necessary to specify the partitioning column, all the partition tables are accessed.

Example 147 Execution plan of a SELECT statement

```
postgres=> EXPLAIN SELECT * FROM main1 WHERE key1 = 10 ;
               QUERY PLAN
-----
Append  (cost=0.00..8.17 rows=2 width=108)
   -> Seq Scan on main1  (cost=0.00..0.00 rows=1 width=108)
       Filter: (key1 = 10::numeric)
   -> Index Scan using pk_main1_part100 on main1_part100  (cost=0.15..8.17 rows=1
width=108)
       Index Cond: (key1 = 10::numeric)  -- Access to only one table
Planning time: 0.553 ms
(6 rows)

postgres=> EXPLAIN SELECT * FROM main1 WHERE key1 + 1 = 11 ;
               QUERY PLAN
-----
Append  (cost=0.00..20.88 rows=6 width=108)
   -> Seq Scan on main1  (cost=0.00..0.00 rows=1 width=108)
       Filter: ((key1 + 1::numeric) = 11::numeric)
   -> Seq Scan on main1_part100  (cost=0.00..18.85 rows=3 width=108)
       Filter: ((key1 + 1::numeric) = 11::numeric)
   -> Seq Scan on main1_part200  (cost=0.00..1.01 rows=1 width=108)
       Filter: ((key1 + 1::numeric) = 11::numeric)
   -> Seq Scan on main1_part300  (cost=0.00..1.01 rows=1 width=108)
       Filter: ((key1 + 1::numeric) = 11::numeric)
Planning time: 0.167 ms  -- Access to all tables
(10 rows)
```

□ Execution of INSERT statement

INSERT statement is distributed to the partition tables by the trigger. On the execution plan it is output that the trigger is activated.

Example 148 Execution plan of the INSERT statement

```
postgres=> EXPLAIN ANALYZE VERBOSE INSERT INTO main1 VALUES (101, 'val1', 'val2') ;
                                         QUERY PLAN
-----
Insert on public.main1  (cost=0.00..0.01 rows=1 width=0) (actual time=0.647..0.647
rows=0 loops=1)
  -> Result  (cost=0.00..0.01 rows=1 width=0) (actual time=0.001..0.001 rows=1
loops=1)
        Output:  101::numeric, 'val1'::character varying(10), 'val2'::character
varying(10)
Planning time: 0.046 ms
Trigger trg_main1_insert: time=0.635 calls=1
Execution time: 0.675 ms
(6 rows)
```

□ Execution of DELETE statement

DELETE statement accesses only to a particular partition if it is possible to specify a partition by WHERE clause. The condition to specify a partition is the same as SELECT statement.

Example 149 Execution plan of the DELETE statement

```
postgres=> EXPLAIN DELETE FROM main1 WHERE key1 = 100 ;
                                         QUERY PLAN
-----
Delete on main1  (cost=0.00..8.17 rows=2 width=6)
  -> Seq Scan on main1  (cost=0.00..0.00 rows=1 width=6)
        Filter: (key1 = 100::numeric)
  -> Index Scan using pk_main1_part200 on main1_part200  (cost=0.15..8.17 rows=1
width=6)
        Index Cond: (key1 = 100::numeric)
Planning time: 0.973 ms
```

□ Execution of UPDATE statement

UPDATE statement accesses only to a particular partition if it is possible to specify a partition by WHERE clause. The condition to specify a partition is the same as the SELECT statement.

Example 150 Execution plan of the UPDATE statement

```
postgres=> EXPLAIN UPDATE main2 SET val1='upd' WHERE key1 = 100 ;
QUERY PLAN
-----
Update on main2 (cost=0.00..8.30 rows=2 width=46)
  -> Seq Scan on main2 (cost=0.00..0.00 rows=1 width=76)
        Filter: (key1 = 100::numeric)
  -> Index Scan using pk_main2_part1 on main2_part1 (cost=0.29..8.30 rows=1 width=15)
        Index Cond: (key1 = 100::numeric)
Planning time: 1.329 ms
(6 rows)
```

□ JDBC PreparedStatement

The execution plan in case that the partition key columns are formed to bind variable using a PreparedStatement object was confirmed by Java application. As JDBC Driver, postgresql-9.3-1101.jdbc41.jar; and as JRE; 1.7.0_09-icedtea was used. It is confirmed that the automatic selection function works even if bind variables are used for the partition key.

Example 151 Part of Java application source

```
PreparedStatement st = cn.prepareStatement("SELECT * FROM main1 WHERE key1=?") ;
st.setInt(1, 200) ;
ResultSet rs = st.executeQuery() ;
```

Example 152 Execution plan

```
Append (cost=0.00..188.99 rows=2 width=62) (actual time=0.018..2.947 rows=1 loops=1)
  -> Seq Scan on main1 (cost=0.00..0.00 rows=1 width=108) (actual time=0.003..0.003
rows=0 loops=1)
        Filter: (key1 = 200::numeric)
  -> Seq Scan on main1_part200 (cost=0.00..188.99 rows=1 width=16)
        (actual time=0.014..2.943 rows=1 loops=1)
        Filter: (key1 = 200::numeric)
        Rows Removed by Filter: 9998
Planning time: 1.086 ms
Execution time: 3.005 ms
```

6.2.4 Constraint

The constraint created to the parent table of the partition table does not functions. Constraints must be specified to the inherit table. In the following example, a primary key constraint is specified on the parent table, but the primary key violation record is stored.

Example 153 Store a record of a primary key constraint violation

```
postgres=> ALTER TABLE main1 ADD CONSTRAINT pk_main1 PRIMARY KEY (key1) ;
ALTER TABLE
postgres=> INSERT INTO main1 VALUES (100, 'val1', 'val2') ;
INSERT 0 0
postgres=> INSERT INTO main1 VALUES (100, 'val1', 'val2') ;
INSERT 0 0
```

6.2.5 Record move between partitions

The UPDATE statement, which attempts to change the value of the partition key column to the value to be stored in another partition, cannot be executed.

Example 154 Update of the partition key column

```
postgres=> \d+ main1_part1
                                Table "public.main1_part1"
  Column |          Type          | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
  key1   | numeric                |           | main    |              |
  val1   | character varying(10)  |           | extended |              |
  val2   | character varying(10)  |           | extended |              |
Check constraints:
    "main1_part1_key1_check" CHECK (key1 < 10000::numeric)
Inherits: main1

postgres=> UPDATE main1 SET key1 = 15000 WHERE key1 = 100 ;
ERROR:  new row for relation "main1 part1" violates check constraint
"main1 part1 key1 check"
DETAIL:  Failing row contains (15000, val1, val2).
postgres=>
```

6.2.6 Partition table and statistics

Statistical information in the partition table is stored independently for each parent table and inheritance table. The value of the reltuples column and relpages column of pg_class catalog of the parent table is 0. Statistics of pg_stats catalog of the parent table is the sum of all inheritance table.

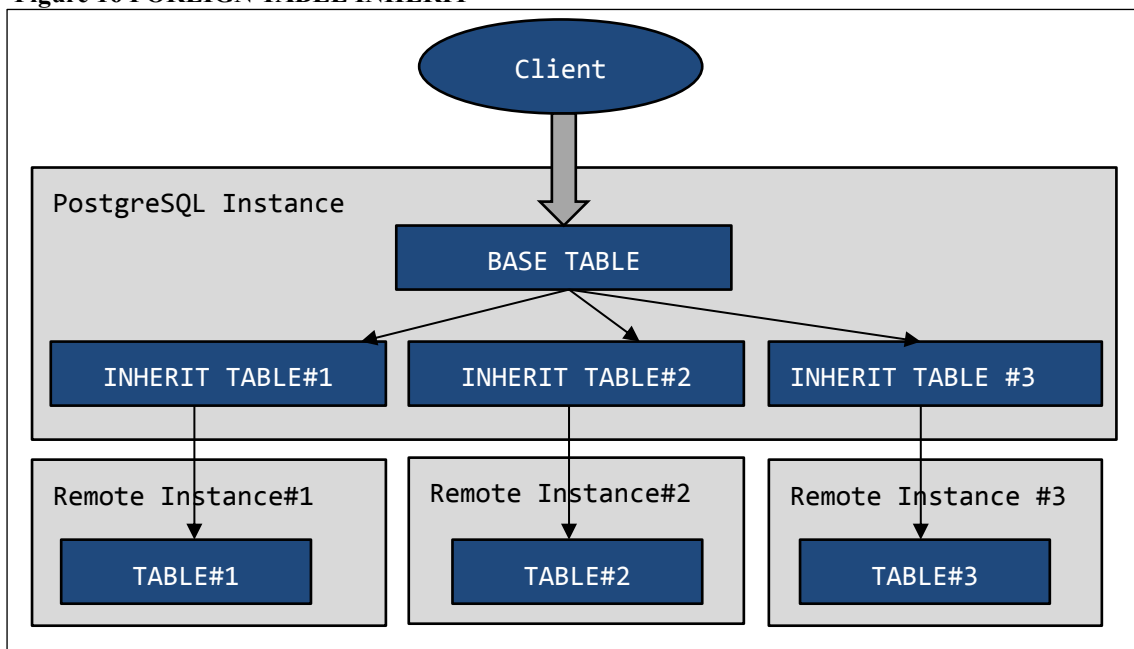
Statistics of inheritance table in the ANALYZE statement for the parent table is not updated. In the case of update the statistics of inheritance table, execute ANALYZE statement for each inheritance table.

The update of the parent table by automatic VACUUM is done only when the update DML for the parent table is executed. Case of performing a direct update DML to inheritance table, ANALYZE for the parent table is not executed. As a result, the sum of the statistical information of statistical information the inheritance table and of the parent table might be different.

6.2.7 Partition table with External table

In PostgreSQL 9.5, as inheritance table of an existing table, you can create an external table (FOREIGN TABLE). This feature makes it possible to distribute the processing to the multiple hosts with a combination of partitioning and external table.

Figure 16 FOREIGN TABLE INHERIT



Syntax 4 CREATE FOREIGN TABLE INHERITS

```
CREATE FOREIGN TABLE table_name (check_constraints ...)
INHERITS (parent_table)
SERVER server_name
OPTIONS (option = 'value' ...)
```

INHERITS clause is new feature of PostgreSQL 9.5. Parent table is specified in INHERITS clause. Example of the implementation and the verification of the execution plan are shown below.

Example 155 Create parent table

```
postgres=> CREATE TABLE parent1(key NUMERIC, val TEXT) ;
CREATE TABLE
```

Create tables (inherit1, inherit2, inherit3) on the remote instance.

Example 156 Create child table on the remote instance

```
postgres=> CREATE TABLE inherit1(key NUMERIC, val TEXT) ;
CREATE TABLE
```

Execute CREATE FOREIGN TABLE statement to create an external table corresponds to the table (inherit1, inherit2, inherit3) on each remote instance.

Example 157 Create External Tables

```
postgres=# CREATE EXTENSION postgres_fdw ;
CREATE EXTENSION
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
           OPTIONS (host 'remsvr1', dbname 'demodb', port '5432') ;
CREATE SERVER
postgres=# CREATE USER MAPPING FOR public SERVER remsvr1
           OPTIONS (user 'demo', password 'secret') ;
CREATE USER MAPPING
postgres=# GRANT ALL ON FOREIGN SERVER remsvr1 TO public ;
GRANT
postgres=> CREATE FOREIGN TABLE inherit1(CHECK(key < 1000))
           INHERITS (parent1) SERVER remsvr1 ;
CREATE FOREIGN TABLE
```

The changes from PostgreSQL 9.4 are: specifying CHECK constraint rather than column definition in CREATE FOREIGN TABLE statement, and specifying the original table in INHERITS clause. Although the above example shows only one instance, in fact, execute CREATE SERVER statement, CREATE USER MAPPING statement, and CREATE FOREIGN TABLE statement should be executed for multiple instances.

Example 158 Check FOREIGN TABLE Definition with INHERITS Clause

```
postgres=> \d+ inherit2

                                Foreign table "public.inherit2"
 Column | Type   | Modifiers | FDW Options | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----+-----
 key    | numeric |           |             | main    |              |
 val    | text    |           |             | extended |              |
Check constraints:
    "inherit2_key_check" CHECK (key >= 1000::numeric AND key < 2000::numeric)
Server: remsvr2
Inherits: parent1
```

When you check the execution plan, you can see that SQL accesses only to a specific instance due to CHECK constraints.

Example 159 Check the Execution Plan

```
postgres=> EXPLAIN SELECT * FROM parent1 WHERE key = 1500 ;

                                QUERY PLAN
-----
Append  (cost=0.00..121.72 rows=6 width=64)
  -> Seq Scan on parent1 (cost=0.00..0.00 rows=1 width=64)
      Filter: (key = '1500'::numeric)
  -> Foreign Scan on inherit2 (cost=100.00..121.72 rows=5 width=64)
(4 rows)
```

6.3 Sequence Object

6.3.1 Using the SEQUENCE object

SEQUENCE is an object that automatically generates a unique numeric value, and it is created using the CREATE SEQUENCE statement. For detailed usage, please refer to the manual¹². To get the value using the SEQUENCE, specify the name of a sequence to nextval function. In order to get the current value, specify the currval function. If the currval function is executed without running the nextval function in the session, it will result in an error.

Example 160 Using the SEQUENCE object

```
postgres=> CREATE SEQUENCE seq01 ;
CREATE SEQUENCE
postgres=> SELECT currval('seq01') ;
ERROR:  currval of sequence "seq01" is not yet defined in this session
postgres=> SELECT nextval('seq01') ;
nextval
-----
         1
(1 row)

postgres=> SELECT currval('seq01') ;
currval
-----
         1
(1 row)
```

Sequence list refers \ds command of psql utility or the information_schema.sequences view. In addition, to get the individual information of the sequence, execute a SELECT statement by specifying the sequence name in the table name.

¹² Online Manual <https://www.postgresql.org/docs/9.6/static/sql-createsequence.html>

Example 161 Information obtainment of SEQUENCE

```
postgres=> \ds+
                                List of relations
 Schema | Name  | Type   | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
 public | seq01 | sequence | data1 | 8192 bytes |
(1 row)
postgres=> SELECT sequence_schema, sequence_name, start_value FROM
            information_schema.sequences ;
 sequence_schema | sequence_name | start_value
-----+-----+-----
 public          | seq01         | 1
(1 row)
postgres=> SELECT sequence_name, start_value, cache_value FROM seq01 ;
 sequence_name | start_value | cache_value
-----+-----+-----
 seq01         | 1           | 1
(1 row)
```

6.3.2 Cache

CACHE clause can be specified in the CREATE SEQUENCE statement. For this attribute, the number of the cached sequence value is specified. The default value for the CACHE clause is 1, and in this case cache is generated. The manual says, "The optional clause CACHE cache specifies how many sequence numbers are to be pre-allocated and stored in memory for faster access. The minimum value is 1 (only one value can be generated at a time, ie, no cache), and this is also the default. ", but it does not describe the exact memory area which "memory" means.

The memory area where cache actually takes place is a virtual memory area of the back-end process postgres. If more than one session get the SEQUENCE value from one sequence caches corresponding to the each session are generated. The same value is never obtained, but the relation of the sequence value size and the chronological sequence value does not match.



Example 162 Cache and chronological sequence value

```
(SESSION#1) postgres=> CREATE SEQUENCE seq01 CACHE 10 ;
CREATE SEQUENCE
(SESSION#1) postgres=> SELECT nextval('seq01') ;
nextval
-----
      1
(1 row)
(SESSION#2) postgres=> SELECT nextval('seq01') ;
nextval
-----
     11
(1 row)

(SESSION#1) postgres=> SELECT nextval('seq01') ;
nextval
-----
     2
(1 row)

(SESSION#2) postgres=> SELECT nextval('seq01') ;
nextval
-----
     12
(1 row)
```

In the above example, a SEQUENCE is created by specifying CACHE 10, and the value is obtained using nextval function. At this point in the SESSION#1 session the value 1 to 10 is created as a cache. Next, as the nextval function is executed in SESSION#2 session, another cache of value 11 to 20 is created, and the nextval function returns 11. DISCARD SEQUENCES statement becomes available on PostgreSQL 9.4 to remove a sequence cache.



6.3.3 Transaction

Sequence values are independent of the transaction. Sequence value obtained during a transaction cannot be rolled back.

Example 163 Sequence and Transaction

```
postgres=> BEGIN ;
BEGIN
postgres=> SELECT nextval('seq01') ;
nextval
-----
      3
(1 row)
postgres=> ROLLBACK ;
ROLLBACK
postgres=> SELECT nextval('seq01') ;
nextval
-----
      4
(1 row)
postgres=>
```

6.4 Bind variables and PREPARE statement

Executing PREPARE statement can create a prepared statement object on the session. By reusing a prepared statement, it can be used to reduce the load of the SQL parsing and rewriting. I confirmed that execution plan is created when EXECUTE statement is executed using the auto_explain Contrib module. When not in use bind variables, the execution plan of the SQL statement that was created by the PREPARE statement using the table and the SQL statements that change the execution plan by the column values has been confirmed or not to change.

In the following example, after creating a bind1test object in the PREPARE statement, running twice SELECT statement with different parameters. To get the most of the data in the table in the first run, in the second run and has obtained part in the table. Index has been given to the column c2.

Example 164 Creating and executing PREPARE object

```
postgres=> PREPARE bind1test(VARCHAR) AS SELECT COUNT(*) FROM bind1
WHERE c2=$1 ;
PREPARE
postgres=> EXECUTE bind1test('maj') ;
count
-----
10000000
(1 row)

postgres=> EXECUTE bind1test('min') ;
count
-----
101
(1 row)
```

I've output the execution plan by auto_explain Contrib module. Seq Scan has been adopted in the first execution.



Example 165 Execution plan of the first execution

```
LOG:  duration: 1583.140 ms  plan:
      Query Text: PREPARE bind1test(VARCHAR) AS SELECT COUNT(*) FROM bind1
WHERE c2=$1 ;
      Aggregate  (cost=204053.52..204053.53 rows=1 width=0)
      ->  Seq Scan on bind1  (cost=0.00..179055.85 rows=9999068 width=0)
          Filter: ((c2)::text = 'maj'::text)
```

In the second execution, Index Only Scan has been adopted.

Example 166 Execution plan of the second execution

```
LOG:  duration: 0.046 ms  plan:
      Query Text: PREPARE bind1test(VARCHAR) AS SELECT COUNT(*) FROM
bind1 WHERE c2=$1;
      Aggregate  (cost=41.44..41.45 rows=1 width=0)
      ->  Index Only Scan using idx1_bind1 on bind1  (cost=0.43..38.94
rows=1000 width=0)
          Index Cond: (c2 = 'min'::text)
```

As a result, you will find that the execution plan has been changed. Because of this execution plan even when the run the PREPARE statement was confirmed that is created each time. This behavior has been adopted from PostgreSQL 9.2. Until the execution plan as Oracle Database and Microsoft SQL Server does not mean to cache.

6.5 *INSERT ON CONFLICT* statement

6.5.1 Basic syntax of INSERT ON CONFLICT statement

When INSERT statement is executed under condition that it becomes a constraint violation, automatic switch to the UPDATE statement is now available on PostgreSQL 9.5 (called UPSERT statement). To use this feature, specify the ON CONFLICT clause in the INSERT statement.

Syntax 5 INSERT ON CONFLICT

```
INSERT INTO ...  
ON CONFLICT [ { (column_name, ...) | ON CONSTRAINT constraint_name }]  
{ DO NOTHING | DO UPDATE SET column_name = value }  
[ WHERE ... ]
```

In the ON CONFLICT clause, specify where the constraint violation will occur.

- Specify a list of column names, or a constraint name in the syntax of the "ON CONSTRAINT *constraint_name*".
- In case of specifying constraints consisting of multiple columns, it is necessary to specify all of the column names that are included in the constraint.
- If you omit the par after ON CONFLICT, all of the constraint violation is checked. This omission is available only in case of using DO NOTHING clause.
- When the constraint violation occurs for other than the specified columns or constraints in the ON CONFLICT clause, INSERT statement will result in an error.

After the ON CONFLICT clause, the behavior when constraint violation has occurred is described. If you specify a DO NOTHING clause, nothing is done for constraint violations occurred (constraint violation does not occur). If you specify a DO UPDATE clause, UPDATE to the specific columns is executed. Following is the execution examples.

Example 167 Prepare table and data

```
postgres=> CREATE TABLE upsert1 (key NUMERIC, val VARCHAR(10)) ;
CREATE TABLE
postgres=> ALTER TABLE upsert1 ADD CONSTRAINT pk_upsert1 PRIMARY KEY (key) ;
ALTER TABLE
postgres=> INSERT INTO upsert1 VALUES (100, 'Val 1') ;
INSERT 0 1
postgres=> INSERT INTO upsert1 VALUES (200, 'Val 2') ;
INSERT 0 1
postgres=> INSERT INTO upsert1 VALUES (300, 'Val 3') ;
INSERT 0 1
```

The following is a description example of the ON CONFLICT clause. Since DO NOTHING is specified in the handling part, nothing is done even if constraint violation occurs.

Example 168 ON CONFLICT clause

```
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
          ON CONFLICT DO NOTHING ; -- omit constraint name or column
INSERT 0 0
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
          ON CONFLICT(key) DO NOTHING ; -- set column name
INSERT 0 0
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
          ON CONFLICT(val) DO NOTHING ; -- error when no constraint column set
ERROR:  there is no unique or exclusion constraint matching the ON CONFLICT
specification
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
          ON CONFLICT ON CONSTRAINT pk_upsert1 DO NOTHING ; -- set constraint
INSERT 0 0
```

In the DO UPDATE clause, you describes the update process. This is basically the same as the part subsequent the SET clause in the UPDATE statement. If you use an alias EXCLUDED, you can access the records that could not be stored when INSERT statement is executed.

Example 169 DO UPDATE clause samples

```
postgres=> INSERT INTO upsert1 VALUES (400, 'Upd4')
           ON CONFLICT DO UPDATE SET val = EXCLUDED.val ; -- No constraint error
ERROR:  ON CONFLICT DO UPDATE requires inference specification or constraint name
LINE 2: ON CONFLICT DO UPDATE SET val = EXCLUDED.val;
           ^
HINT:  For example, ON CONFLICT ON CONFLICT (<column>).
postgres=> INSERT INTO upsert1 VALUES (300, 'Upd3')
           ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val ; -- Use EXCLUDED alias
INSERT 0 1
postgres=> INSERT INTO upsert1 VALUES (300, 'Upd3')
           ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val WHERE upsert1.key = 100 ;
INSERT 0 0 -- Can determin UPDATE conditions when specify the WHERE clause
```

6.5.2 Relation between ON CONFLICT Clause and Trigger

How the trigger works during the execution of INSERT ON CONFLICT statement is verified. BEFORE INSERT trigger has always been executed. If the record is updated by DO UPDATE statement, BEFORE INSERT trigger and BEFORE / AFTER UPDATE trigger worked. Only BEFORE INSERT trigger was executed if the UPDATE is not performed by the WHERE clause.

Table 65 Executed triggers (ON EACH ROW trigger; the numbers are the order of execution)

Trigger	Success INSERT	DO NOTHING	DO UPDATE (Updated)	DO UPDATE (NO Updated)
BEFORE INSERT	1, Execute	1, Execute	1, Execute	1, Execute
AFTER INSERT	2, Execute	-	-	-
BEFORE UPDATE	-	-	2, Execute	-
AFTER UPDATE	-	-	3, Execute	-

Table 66 Executed triggers (ON EACH STATEMENT; the numbers are the order of execution)

Trigger	Success INSERT	DO NOTHING	DO UPDATE (Updated)	DO UPDATE (NO Updated)
BEFORE INSERT	1,Execute	1,Execute	1,Execute	1,Execute
AFTER INSERT	4,Execute	2,Execute	4,Execute	4,Execute
BEFORE UPDATE	2,Execute	-	2,Execute	2,Execute
AFTER UPDATE	3,Execute	-	3,Execute	3,Execute

6.5.3 ON CONFLICT clause and Execution Plan

By executing ON CONFLICT clause, the execution plan will change. When you run the EXPLAIN statement, Conflict Resolution, Conflict Arbiter Indexes, and Conflict Filter appear in the execution plan. Actual output is shown in the following example.

Example 170 ON CONFLICT Clause and Execution Plan

```

postgres=> EXPLAIN INSERT INTO upsert1 VALUES (200, 'Update 1')
           ON CONFLICT(key) DO NOTHING ;
           QUERY PLAN

-----
Insert on upsert1 (cost=0.00..0.01 rows=1 width=0)
  Conflict Resolution: NOTHING
  Conflict Arbiter Indexes: pk upsert1
  -> Result (cost=0.00..0.01 rows=1 width=0)
(4 rows)

postgres=> EXPLAIN INSERT INTO upsert1 VALUES (400, 'Upd4')
           ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val ;
           QUERY PLAN

-----
Insert on upsert1 (cost=0.00..0.01 rows=1 width=0)
  Conflict Resolution: UPDATE
  Conflict Arbiter Indexes: pk upsert1
  -> Result (cost=0.00..0.01 rows=1 width=0)
(4 rows)

postgres=> EXPLAIN INSERT INTO upsert1 VALUES (400, 'Upd4')
           ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val WHERE upsert1.key = 100 ;
           QUERY PLAN

-----
Insert on upsert1 (cost=0.00..0.01 rows=1 width=0)
  Conflict Resolution: UPDATE
  Conflict Arbiter Indexes: pk upsert1
  Conflict Filter: (upsert1.key = '100'::numeric)
  -> Result (cost=0.00..0.01 rows=1 width=0)
(5 rows)

```

In current version does not support ON CONFLICT DO UPDATE statement to the remote instance using postgres_fdw module.

6.5.4 ON CONFLICT clause and the partition table

ON CONFLICT clause for partition table using the INSERT trigger it will be ignored.

Example 171 INSERT ON CONFLICT statement for the partition table#1

```
postgres=> CREATE TABLE main1 (key1 NUMERIC, val1 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE TABLE main1_part100 (CHECK(key1 < 100)) INHERITS (main1) ;
CREATE TABLE
postgres=> CREATE TABLE main1_part200 (CHECK(key1 >= 100 AND key1 < 200))
        INHERITS (main1) ;
CREATE TABLE
postgres=> ALTER TABLE main1_part100 ADD CONSTRAINT pk_main1_part100
        PRIMARY KEY (key1);
ALTER TABLE
postgres=> ALTER TABLE main1_part200 ADD CONSTRAINT pk_main1_part200
        PRIMARY KEY (key1);
ALTER TABLE
postgres=> CREATE OR REPLACE FUNCTION func_main1_insert()
        RETURNS TRIGGER AS $$
        BEGIN
            IF      (NEW.key1 < 100) THEN
                INSERT INTO main1_part100 VALUES (NEW.*) ;
            ELSIF (NEW.key1 >= 100 AND NEW.key1 < 200) THEN
                INSERT INTO main1_part200 VALUES (NEW.*) ;
            ELSE
                RAISE EXCEPTION 'ERROR! key1 out of range.' ;
            END IF ;
            RETURN NULL ;
        END ;
        $$ LANGUAGE 'plpgsql';
CREATE FUNCTION
```



Example 172 INSERT ON CONFLICT statement for the partition table#2

```
postgres=> CREATE TRIGGER trg_main1_insert BEFORE INSERT ON main1
            FOR EACH ROW EXECUTE PROCEDURE func_main1_insert() ;
CREATE TRIGGER
postgres=> INSERT INTO main1 VALUES (100, 'DATA100') ;
INSERT 0 0
postgres=> INSERT INTO main1 VALUES (100, 'DATA100') ;
ERROR:  duplicate key value violates unique constraint "pk_main1_part200"
DETAIL:  Key (key1)=(100) already exists.
CONTEXT:  SQL statement "INSERT INTO main1_part200 VALUES (NEW.*)"
PL/pgSQL function func_main1_insert() line 6 at SQL statement
postgres=> INSERT INTO main1 VALUES (100, 'DATA100')
            ON CONFLICT DO NOTHING ;
ERROR:  duplicate key value violates unique constraint "pk_main1_part200"
DETAIL:  Key (key1)=(100) already exists.
CONTEXT:  SQL statement "INSERT INTO main1_part200 VALUES (NEW.*)"
PL/pgSQL function func_main1_insert() line 6 at SQL statement
```

6.6 TABLESAMPLE

6.6.1 Overview

TABLESAMPLE clause is the syntax for sampling a certain percentage of the record from the table. This syntax is available from PostgreSQL 9.5.

Syntax 6 TABLESAMPLE Clause

```
SELECT ... FROM table_name ...  
TABLESAMPLE {SYSTEM | BERNOULLI} (percent)  
[ REPEATABLE (seed) ]
```

You can choose SYSTEM or BERNOULLI as sampling method. SYSTEM uses the tuple entire block sampled. BERNOULLI choose a more constant rate of tuples from sampled block. In 'percent' you specify the sampling percentage (1-100). When you specify the value other than 1 - 100 SELECT statement results in an error. REPEATABLE clause is optional the seed of sampling can be specified.

6.6.2 SYSTEM と BERNOULLI

SYSTEM clause uses a tuple of the entire block that was sampled. BERNOULLI will select a certain percentage of the tuple from within the block. In order to perform the sampling, it must first estimate the number of records in the entire table. Scanning method of the table, how to check the visible record is changed by the sampling method and the sampling rate.

□ SYSTEM

If the sampling rate is more than 1%, bulk load (Bulk Read) is performed. Check visualization is done in page mode. The following is the source code of the relevant part.



Example 173 system_beginsamplescan function (src/backend/access/tablesample/system.c)

```
/*
 * Bulkread buffer access strategy probably makes sense unless we're
 * scanning a very small fraction of the table. The 1% cutoff here is a
 * guess. We should use pagemode visibility checking, since we scan all
 * tuples on each selected page.
 */
node->use_bulkread = (percent >= 1);
node->use_pagemode = true;
```

□ **BERNOULLI**

Always will be performed bulk load (Bulk Read). Check visualization is done in page mode when the sampling rate is more than 25 percent. The following is the source code of the relevant part.

Example 174 bernoulli_beginsamplescan function (src/backend/access/tablesample/bernoulli.c)

```
/*
 * Use bulkread, since we're scanning all pages. But pagemode visibility
 * checking is a win only at larger sampling fractions. The 25% cutoff
 * here is based on very limited experimentation.
 */
node->use_bulkread = true;
node->use_pagemode = (percent >= 25);
```

In the following example, to create a table of the same structure, it has confirmed the number of read block. Both table size is 5,000 block.

Example 175 Confirmation of the number of read block

```
postgres=> SELECT COUNT(*) FROM samplesys TABLESAMPLE SYSTEM (10) ;
postgres=> SELECT COUNT(*) FROM sampleber TABLESAMPLE BERNOULLI (10) ;
postgres=> SELECT relname, heap_blks_read FROM pg_statio_user_tables ;
  relname  | heap_blks_read
-----+-----
 samplesys |             533
 sampleber |            4759
(2 rows)
```

6.6.3 Execution Plan

Execution plan in the case of performing the sampling will be as follows. If you specify a BERNOULLI, you can see that the cost increases.

Example 176 Execution plan for TABLESAMPLE SYSTEM

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 TABLESAMPLE SYSTEM (10) ;
                                QUERY PLAN
-----
Aggregate  (cost=341.00..341.01 rows=1 width=0) (actual time=4.914..4.915 rows=1
loops=1)
  -> Sample Scan (system) on data1  (cost=0.00..316.00 rows=10000 width=0)
      (actualtime=0.019..3.205 rows=10090 loops=1)
Planning time: 0.106 ms
Execution time: 4.977 ms
(4 rows)
```



Example 177 Execution plan for TABLESAMPLE BERNOULLI

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 TABLESAMPLE
            BERNOULLI (10) ;
            QUERY PLAN
-----
Aggregate  (cost=666.00..666.01 rows=1 width=0) (actual time=13.654..13.655
rows=1 loops=1)
  -> Sample Scan (bernoulli) on data1  (cost=0.00..641.00 rows=10000 width=0)
      (actual time=0.013..12.121 rows=10003 loops=1)
Planning time: 0.195 ms
Execution time: 13.730 ms
```

6.7 Changing a table attribute.

Table definition can be changed in the ALTER TABLE statement. Here I describe the impact of the ALTER TABLE statement.

6.7.1 ALTER TABLE SET UNLOGGED

By executing the ALTER TABLE SET LOGGED statement or ALTER TABLE SET UNLOGGED statement, you can switch between the normal table and the unlogged table.

Example 178 Switching to UNLOGGED TABLE

```
postgres=> CREATE TABLE logtbl1 (c1 NUMERIC, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> \d+ logtbl1
```

Column	Type	Modifiers	Storage	Stats target	Description
c1	numeric		main		
c2	character varying(10)		extended		

```
postgres=> ALTER TABLE logtbl1 SET UNLOGGED ;
ALTER TABLE
postgres=> \d+ logtbl1
```

Column	Type	Modifiers	Storage	Stats target	Description
c1	numeric		main		
c2	character varying(10)		extended		

The \d+ command of psql command, you can see that the normal table has been changed to UNLOGGED table. When setting change, internally to create a new UNLOGGED TABLE (or TABLE) with the same structure, and then copy the data. The relfilenode columns and relpersistence column of the pg_class catalog will be changed.

Example 179 Change of relfilenode column

```
postgres=> SELECT relname, relfilenode, relpersistence FROM pg_class
           WHERE relname='logtbl1' ;
 relname | relfilenode | relpersistence
-----+-----+-----
 logtbl1 |      16483 | p
(1 row)
```

```
postgres=> ALTER TABLE logtbl1 SET UNLOGGED ;
ALTER TABLE
postgres=> SELECT relname, relfilenode, relpersistence FROM pg_class
           WHERE relname='logtbl1' ;
 relname | relfilenode | relpersistence
-----+-----+-----
 logtbl1 |      16489 | u
(1 row)
```

□ Switching in the replication environment

You cannot access the UNLOGGED table in slave instance of streaming replication environment. For this reason, the table has been changed from LOGGED TABLE to UNLOGGED TABLE is, it will no longer be accessible from the slave instance.

Example 180 Reference UNLOGGED TABLE in slave instance

```
MASTER
postgres=> ALTER TABLE logtest1 SET UNLOGGED ;
ALTER TABLE

SLAVE
postgres=> SELECT * FROM logtest1 ;
ERROR:  cannot access temporary or unlogged relations during recovery
```

On the other hand, in the change from the UNLOGGED TABLE to LOGGED TABLE, because the WAL is output, the contents of the table, which was converted into LOGGED TABLE is, will be accessible at the slave instance.



Example 181 Reference TABLE in slave instance

MASTER

```
postgres=> CREATE UNLOGGED TABLE logtest1(c1 NUMERIC) ;  
CREATE TABLE  
postgres=> INSERT INTO logtest1 VALUES (generate_series(1, 10000)) ;  
INSERT 0 10000  
postgres=> ALTER TABLE logtest1 SET LOGGED ;  
ALTER TABLE
```

SLAVE

```
postgres=> SELECT COUNT(*) FROM logtest1 ;  
count  
-----  
10000  
(1 row)
```

6.7.2 ALTER TABLE SET WITH OIDS

When you execute ALTER TABLE SET WITH OIDS statement or ALTER TABLE SET WITHOUT OIDS statement, the temporary table of different attributes are created, data is copied. For this reason, the file names constituting the table is changed. It will also be re-created also the index that have been granted to the table.



Example 182 Filename with ALTER TABLE SET WITH OIDS

```
postgres=> SELECT relname, relfilenode FROM pg_class WHERE relname = 'data1' ;
 relname  | relfilenode
-----+-----
 data1    |         16468
(1 row)

postgres=> ALTER TABLE data3 SET WITH OIDS ;
ALTER TABLE
postgres=> SELECT relname, relfilenode FROM pg_class WHERE relname = 'data1' ;
 relname  | relfilenode
-----+-----
 data1    |         17489
(1 row)
```

6.7.3 ALTER TABLE MODIFY COLUMN TYPE

If you run the ALTER TABLE ALTER COLUMN TYPE statement can change the data type of the column. Case of reduce the size of the column data by the execution of the ALTER TABLE statement it will be re-created table. For this reason the file name for which you want to configure the table is changed, it will also be re-created all of the index, which is granted to the table. The following examples set the parameters log_min_messages to debug5, it is a log of when the c2 column of the table data1 column was changed from NUMERIC type NUMERIC (10) type. You will find that index idx1_data1 has been re-created.

Example 183 Log for ALTER TABLE ALTER COLUMN TYPE statement

```
DEBUG: StartTransactionCommand
DEBUG: StartTransaction
DEBUG:  name: unnamed; blockState:          DEFAULT; state: INPROGR,
xid/subid/cid: 0/1/0, nestlvl: 1, children:
DEBUG: ProcessUtility
DEBUG: EventTriggerTableRewrite(16415)
DEBUG: building index "pg_toast_16447_index" on table "pg_toast_16447"
DEBUG: rewriting table "data1"
DEBUG: building index "idx1_data1" on table "data1"
DEBUG: drop auto-cascades to type pg_temp_16415
DEBUG: drop auto-cascades to type pg_temp_16415[]
DEBUG: drop auto-cascades to toast table pg_toast.pg_toast_16415
DEBUG: drop auto-cascades to index pg_toast.pg_toast_16415_index
DEBUG: drop auto-cascades to type pg_toast.pg_toast_16415
DEBUG: CommitTransactionCommand
DEBUG: CommitTransaction
```

6.8 ECPG

ECPG is a preprocessor to write SQL statements directly into the C language program. In this section I have examined the host variable to get the results of executing the SQL statement.

6.8.1 Format of the host variable

I examined the output format of the case that has acquired the character string data in the host variable.

□ CHAR type column values

The CHAR type column of the table writing to the array of char. Stored the "ABC" to a column of CHAR (5) column type and validate the format of the case that was stored in the C language variable char [7]. SP is space (0x20), NL is NULL (0x00), OR indicates that not been changed in the following examples.

Table 67 Column type CHAR (5) to host variable char [7]

Host array	char[0]	char[1]	char[2]	char[3]	char[4]	char[5]	char[6]
Stored data	A	B	C	SP	SP	NL	OR

Is a space after the string is given, the last contains the NULL.

□ VARCHAR type column values

The VARCHAR type column of the table writing to the array of char. Stored the "ABC" to a column of VARCHAR (5) column type and validate the format of the case that was stored in the C language variable char [7].

Table 68 Column type VARCHAR(5) to host variable char[7]

Host array	char[0]	char[1]	char[2]	char[3]	char[4]	char[5]	char[6]
Stored data	A	B	C	NL	OR	OR	OR

Is stored NULL after the string, it will value after the NULL can be seen that not been changed.

□ VARCHAR type host variable

It can be specified the VARCHAR type host variable. When this variable is passed through the ecpg, it will be converted into a structure with member int len and char arr [{99}].

Table 69 Column type VARCHAR(5) to host variable VARCHAR(7)

Host array	arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]
Stored data	A	B	C	NL	NL	NL	NL

All char arr[] array in which data is stored will see that it has been initialized with NULL (0x00).

6.8.2 Behavior at the time of out-of-space

When output the value of the string type column to the host variable, when the area of the host variable is insufficient result is truncated, positive value is written to the indicator variable. The manual is shown as follows, there is no description of the specific operation.

Example 184 Manual for Indicator

This second host variable is called the indicator and contains a flag that tells whether the datum is null, in which case the value of the real host variable is ignored.

□ At the time of out-of-space host variables #1

Writes a VARCHAR type column of the table to a char array, to verify the format when char type is running out of space. Stores the "ABCDE" to VARCHAR (5) type column, to verify the format of the case that is stored in the C language variable char [3].

Table 70 Column type VARCHAR(5) to host variable char[3]

Host array	char[0]	char[1]	char[2]
Stored data	A	B	C

It is stored to the part that can be stored as described above but NULL values are not granted. The indicator is stored five.

□ At the time of out-of-space host variables #2

The VARCHAR type column writes to char array but char type of area was to verify the format of the case that are missing only NULL values. Stores the "ABCDE" to VARCHAR (5) type column, to verify the format of the case that is stored in the C language variable char [5].



Table 71 Column type VARCHAR(5) to host variable char[5]

Host array	char[0]	char[1]	char[2]	char[3]	char[4]
Stored data	A	B	C	D	E

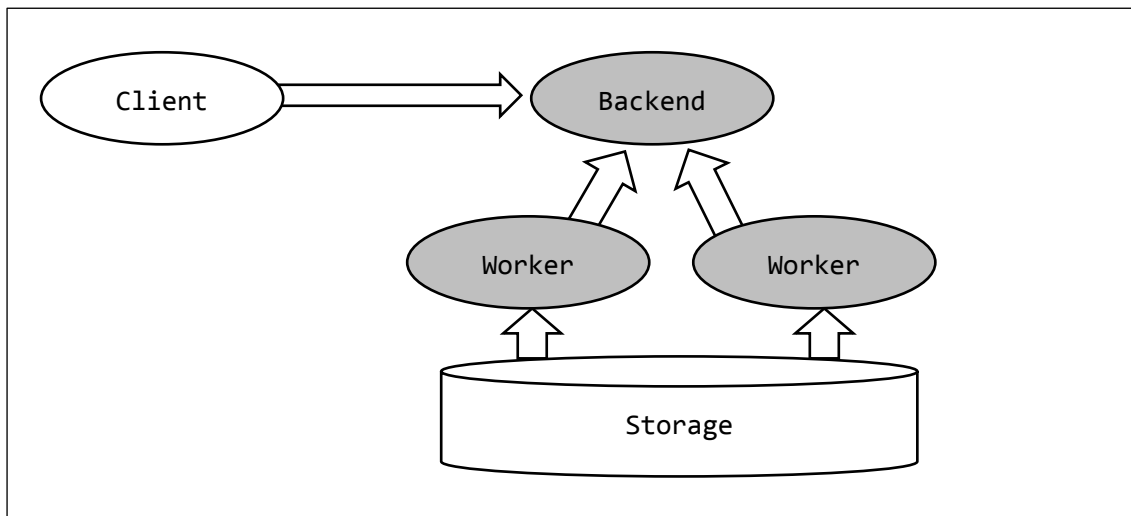
Database storage character as described above will be saved as it is, NULL values are not given. Zero is specified for the indicator. For this reason, space for the NULL cannot identify in the indicator also missing.

6.9 Parallel Query

6.9.1 Overview

In the conventional PostgreSQL, all of the SQL statements were executed only by the back-end process that accepts the connection. In PostgreSQL 9.6 it is possible to perform parallel processing by multiple worker processes.

Figure 17 Parallel Seq Scan / Parallel Aggregate



Parallel processing can be executed only for Seq Scan, Join and Aggregate. The degree of parallelism depends on the size of the table. Processes executing parallel processing use the mechanism of the Background Worker. The maximum value of the degree of parallelism is determined by the parameter `max_parallel_workers_per_gather` or `max_worker_processes`, whichever is smaller. Parameter `max_parallel_degree` can be changed by general users by per-session.

Table 72 Related parameters for parallel processing

Parameter Name	Description (context)	Default value
max_parallel_workers_per_gather	The maximum value of the degree of parallelism (user)	0
parallel_setup_cost	Start cost of parallel processing (user)	1000
parallel_tuple_cost	Tuple cost of parallel processing (user)	0.1
max_worker_processes	The maximum value of the worker process (postmaster)	8
force_parallel_mode	Force parallel processing (user)	off
min_parallel_relation_size	Minimum table size to consider the parallel processing	8MB

□ Parameter force_parallel_mode

Parallel processing is executed only when the cost is considered lower than the normal serial processing. By specifying the parameter force_parallel_mode to on, parallel processing is forced (Also value "regress" is for the regression test). However, the parallel processing is executed only when the parameter max_parallel_workers_per_gather is 1 or more.

□ Related table option

Table option parallel_workers determines the degree of parallelism for each table. When the value is set to 0, parallel processing is prohibited. If not set, the parameters max_parallel_workers_per_gather of the session will be the upper limit. If parallel_workers is set to greater than the max_parallel_workers_per_gather, the upper limit of the actual degree of parallelism cannot exceed the max_parallel_degree.

Example 185 Set the table option parallel_degree

```
postgres=> ALTER TABLE data1 SET (parallel_degree = 2) ;
ALTER TABLE
postgres=> \d+ data1
```

Table "public.data1"					
Column	Type	Modifiers	Storage	Stats target	Description
c1	numeric		main		
c2	character varying(10)		extended		

Options: parallel workers=5

6.9.2 Execution plan

The example below is the execution plan of the parallel processing SELECT statement. COUNT processing of large-scale table is processed in 3 parallelism.

Example 186 Execution plan of parallel processing

```
postgres=> SET max_parallel_degree = 10 ;
SET
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT COUNT(*) FROM data1 ;

               QUERY PLAN

-----
Finalize Aggregate  (cost=29314.09..29314.10 rows=1 width=8)
    (actual time=662.055..662.055 rows=1 loops=1)
    Output: pg_catalog.count(*)
    -> Gather  (cost=29313.77..29314.08 rows=3 width=8)
        (actual time=654.818..662.043 rows=4 loops=1)
        Output: (count(*))
        Workers Planned: 3
        Workers Launched: 3
        -> Partial Aggregate  (cost=28313.77..28313.78 rows=1 width=8)
            (actual time=640.330..640.331 rows=1 loops=4)
            Output: count(*)
            Worker 0: actual time=643.386..643.386 rows=1 loops=1
            Worker 1: actual time=645.587..645.588 rows=1 loops=1
            Worker 2: actual time=618.493..618.494 rows=1 loops=1
            -> Parallel Seq Scan on public.data1  (cost=0.00..25894.42
                rows=967742 width=0) (actual time=0.033..337.848 rows=750000 loops=4)
                Output: c1, c2
                Worker 0: actual time=0.037..295.732 rows=652865 loops=1
                Worker 1: actual time=0.026..415.235 rows=772230 loops=1
                Worker 2: actual time=0.042..359.622 rows=620305 loops=1

Planning time: 0.130 ms
Execution time: 706.955 ms
(18 rows)
```

Following execution plan component can be shown by the EXPLAIN statement about parallel processing.

Table 73 The output of the EXPLAIN statement

Plan component	Description	Explain Statement
Parallel Seq Scan	Parallel search processing	All
Partial Aggregate	Aggregation processing performed by the worker process	All
Partial HashAggregate		
Partial GroupAggregate		
Gather	Processing to aggregate the worker process	All
Finalize Aggregate	The final aggregation processing	All
Workers Planned:	The number of planned worker processes	All
Workers Launched:	The number of workers that are actually run	ANALYZE
Worker N (N=0,1,...)	Processing time of each worker, etc	ANALYZE, VERBOSE
Single Copy	Processing to be executed in a single process	All

6.9.3 Parallel processing and functions

There are usable functions and unusable functions in parallel processing. When functions which have "u" (PARALLEL UNSAFE) value for proparallel column in pg_proc catalog are user in SQL statement, parallel processing cannot be performed. The following table shows major standard PARALLEL UNSAFE functions.

Table 74 Major PARALLEL UNSAFE standard functions

Category	Function name examples
SEQUENCE object	nextval, curval, setval, lastval
Large Object	lo_*, loread, lowrite
Replication	pg_create_*_slot, pg_drop_*_slot, pg_logical_*, pg_replication_*
Other	make_interval, parse_ident, pg_extension_config_dump, pg_*_backup, set_config, ts_debug, txid_current, query_to_xml*

In the following example, two SQL statements that differ only conditional part of the WHERE clause are executed. SELECT statement with the literal in the WHERE clause will be performed parallel processing but, SELECT statement with the curval of sequence operation function is executed in serial.

Example 187 The difference of the execution plan by the use of PARALLEL UNSAFE function

```

postgres=> EXPLAIN SELECT COUNT(*) FROM data1 WHERE c1=10 ;

               QUERY PLAN

-----
Aggregate  (cost=29314.08..29314.09 rows=1 width=8)
  -> Gather  (cost=1000.00..29314.07 rows=3 width=0)
        Workers Planned: 3
        -> Parallel Seq Scan on data1  (cost=0.00..28313.78 rows=1 width=0)
              Filter: (c1 = '10'::numeric)
(5 rows)

postgres=> EXPLAIN SELECT COUNT(*) FROM data1 WHERE c1=currval('seq1') ;

               QUERY PLAN

-----
Aggregate  (cost=68717.01..68717.02 rows=1 width=8)
  -> Seq Scan on data1  (cost=0.00..68717.00 rows=3 width=0)
        Filter: (c1 = (currval('seq1'::regclass))::numeric)
(3 rows)

```

In pg_proc in the catalog, functions that are the proparallel column "r" can only be run on the leader process of parallel processing.

Table 75 Major RESTRICTED PARALLEL SAFE standard functions

Category	Function name examples
Date and Age	age, now
Random number	random, setseed
Upgrade	binary_upgrade*
Convert to XML	cursor_to_xml*, database_to_xml*, schema_to_xml*, table_to_xml*
Other	pg_start_backup, inet_client*, current_query, pg_backend_pid, pg_conf*, pg_cursor, pg_get_viewdef, pg_prepared_statement, etc

□ User-defined functions and PARALLEL SAFE

To indicate whether it is possible to perform parallel processing for user-defined functions, can be used PARALLEL SAFE clause or PARALLEL UNSAFE clause in the CREATE FUNCTION statement or ALTER FUNCTION statement. The default is PARALLEL UNSAFE.

Example 188 User-defined functions and PARALLEL SAFE

```

postgres=> CREATE FUNCTION add(integer, integer) RETURNS integer
postgres->   AS 'select $1 + $2;'
postgres->   LANGUAGE SQL IMMUTABLE RETURNS NULL ON NULL INPUT ;
CREATE FUNCTION
postgres=> SELECT proname, proparallel FROM pg_proc WHERE proname = 'add' ;
 proname | proparallel
-----+-----
 add     | u
(1 row)
postgres=> ALTER FUNCTION add(integer, integer) PARALLEL SAFE ;
ALTER FUNCTION
postgres=> SELECT proname, proparallel FROM pg_proc WHERE proname='add' ;
 proname | proparallel
-----+-----
 add     | s
(1 row)

```

Determine Parallel Safe or Parallel Unsafe of user-defined function, the function author must determine. Even if Parallel Unsafe function is called in the Parallel Safe specified in the user-defined function, the optimizer is likely to create an execution plan for a parallel query. Nextval function, etc., some standard functions, and it raises an error when detecting the parallel query is being performed.

In the example below, I create an unsafe1 a Parallel Unsafe function. In addition, it has created a Parallel Safe function safe1 to run the Parallel Unsafe function.



Example 189 Parallel Safe function to run the Parallel Unsafe function

```
postgres=> CREATE FUNCTION unsafe1() RETURNS numeric AS $$
postgres$> BEGIN RETURN 100; END;
postgres$> $$ PARALLEL UNSAFE LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=>
postgres=> CREATE FUNCTION safe1() RETURNS numeric AS $$
BEGIN RETURN unsafe1(); END;
$$ PARALLEL SAFE LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=>
postgres=> SELECT proname, proparallel FROM pg_proc WHERE proname like
'%safe%' ;
    proname      | proparallel
-----+-----
unsafe1          | u
unsafe1insafe    | s
(2 rows)
```

In the example below, the result of the execution of the SELECT statement, including the Parallel Safe function safe1, you can see that the parallel query is executed.

Example 190 Execution plan for Parallel safe function witch include Parallel Unsafe function

```
postgres=> EXPLAIN ANALYZE VERBOSE SELECT * FROM data1 WHERE c1 = safe1() ;

               QUERY PLAN
-----
Gather  (cost=1000.00..115781.10 rows=1 width=11)
    (actual time=1.354..2890.075 rows=1 loops=1)
    Output: c1, c2
    Workers Planned: 2
    Workers Launched: 2
    -> Parallel Seq Scan on public.data1 (cost=0.00..114781.00 rows=0 width=11)
    (actual time=1904.255..2866.980 rows=0 loops=3)
        Output: c1, c2
        Filter: (data1.c1 = unsafe1unsafe())
        Rows Removed by Filter: 333333
        Worker 0: actual time=2844.205..2844.205 rows=0 loops=1
        Worker 1: actual time=2867.581..2867.581 rows=0 loops=1
    Planning time: 0.083 ms
    Execution time: 2890.790 ms
(12 rows)
```

6.9.4 Calculation of the degree of parallelism

The degree of parallelism will be calculated based on the size of the search target of the table. The reference size of the table is the parameter `min_parallel_relation_size` table, will increase the degree of parallelism for each three times. The upper limit of the degree of parallelism is determined within a range not exceeding the parameter `max_parallel_workers_per_gather` or parameter `max_worker_processes`. The actual calculation process has been described in the `create_plain_partial_paths` function in the source code `src/backend/optimizer/path/allpaths.c`.

Example 191 A part of create_plain_partial_paths function

```
int                                parallel_threshold;

/*
 * If this relation is too small to be worth a parallel scan, just
 * return without doing anything ... unless it's an inheritance child.
 * In that case, we want to generate a parallel path here anyway. It
 * might not be worthwhile just for this relation, but when combined
 * with all of its inheritance siblings it may well pay off.
 */
if (rel->pages < (BlockNumber) min_parallel_relation_size &&
    rel->reloptkind == RELOPT_BASEREL)
    return;

/*
 * Select the number of workers based on the log of the size of the
 * relation. This probably needs to be a good deal more
 * sophisticated, but we need something here for now. Note that the
 * upper limit of the min_parallel_relation_size GUC is chosen to
 * prevent overflow here.
 */
parallel_workers = 1;
parallel_threshold = Max(min_parallel_relation_size, 1);
while (rel->pages >= (BlockNumber) (parallel_threshold * 3))
{
    parallel_workers++;
    parallel_threshold *= 3;
    if (parallel_threshold > INT_MAX / 3)
        break;                                /* avoid overflow */
}
```

7. Privileges and object creation

7.1 Object Privileges

Regarding the object privileges of PostgreSQL, the operations, executable not by the owner of the object, but by the general user (has login privilege only), are summarized here.

7.1.1 The owner of the tablespace

It was verified whether the object can be created if the owner of the connected users and table space is different. From the following results, we found that the object can be created when the database owned by a connection user is created in the tablespace.

Table 76 Connection user and the owner of the table are different

Operation	No Database	Exists Database
CREATE TABLE	Error	OK
CREATE INDEX	Error	OK

7.1.2 The owner of the database

It was verified whether the object can be created in a "public" schema if the database owner and connected user are different. As you see the following table, the major tables other than schmema can be created even if the owner of the database is different. If you want to prohibit access to the "public" schema, all the access rights should be stripped once from public role, and only the required permissions must be added to the appropriate user.

Table 77 Object creation for pg_default table space (postgres user-owned)

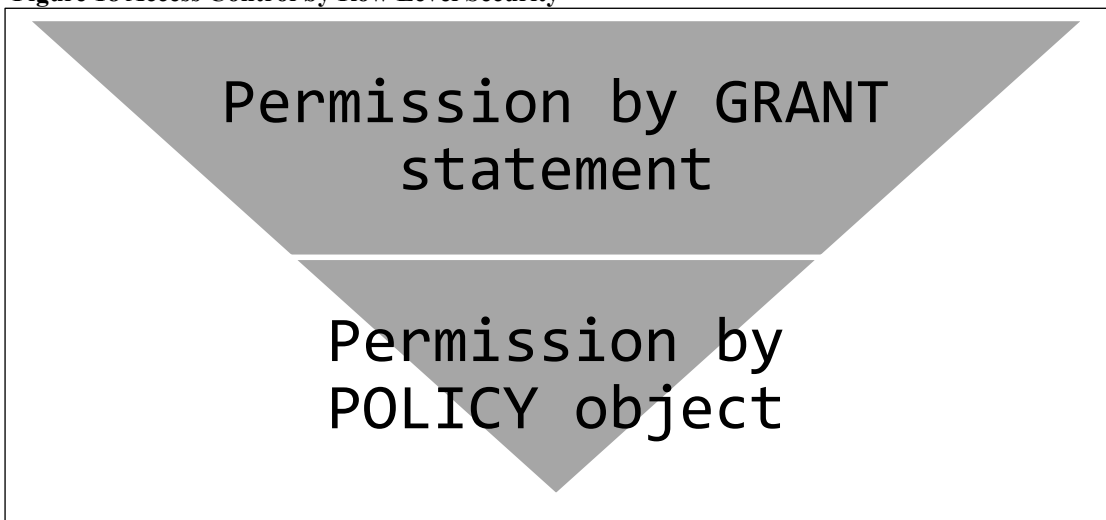
Operation	Executability
CREATE TABLE	OK
CREATE INDEX	OK
CREATE SEQUENCE	OK
CREATE SCHEMA	Error
CREATE FUNCTION	OK
CREATE TYPE	OK

7.2 Row Level Security

7.2.1 What's Row Level Security

In PostgreSQL, the privilege to access to tables and columns is specified by GRANT statement. This style is available even in PostgreSQL 9.5, and furthermore the function of Row Level Security is added. Row Level Security is a feature that allows you to limit the tuples (records) with tuple level, which were allowed in the GRANT statement. In order to use the access restriction by Row Level Security, create an object called POLICY.

Figure 18 Access Control by Row Level Security



7.2.2 Preparation

In order to use Row Level Security, execute the ALTER TABLE ENABLE ROW LEVEL SECURITY statement to the table to be restricted by policy. By default Row Level Security setting of the table is disabled. To disable the setting for the table, run the ALTER TABLE DISABLE ROW LEVEL SECURITY statement.

Example 192 Enable Row Level Security to the table

```
postgres=> ALTER TABLE poltbl1 ENABLE ROW LEVEL SECURITY ;
ALTER TABLE
postgres=> \d+ poltbl1
```

Table "public.poltbl2"					
Column	Type	Modifiers	Storage	Stats target	Description
c1	numeric		main		
c2	character varying(10)		extended		
uname	character varying(10)		extended		

Policies (Row Security Enabled): (None)

7.2.3 Create POLICY object

To specify access privileges for a table, create the POLICY object. POLICY is created using the CREATE POLICY statement. General users can create POLICY.

Syntax 7 CREATE POLICY

```
CREATE POLICY policy_name ON table_name
[ FOR { ALL | SELECT | INSERT | UPDATE | DELETE } ]
[ TO { roles | PUBLIC, ... } ]
USING (condition)
[ WITH CHECK (check_condition) ]
```

Table 78 Syntax for CREATE POLICY statement

Clause	Description
<i>policy_name</i>	Specify the policy name
ON	Specify the table name to which the policy is applied
FOR	Operation name to apply the policy or ALL
TO	Target role name to allow policy or PUBLIC
USING	Specify conditions for permission of the access to the tuple (Same syntax as WHERE clause). Only the tuples in which the condition specified by USING clause becomes TRUE is returned to the user
WITH CHECK	Specify the conditions for rows that can be updated by the UPDATE statement. You can not specify the CHECK clause in the policy for the SELECT statement

In the example below, the POLICY to the table poltbl1 is created. Since TO clause is omitted, the subject is all users (PUBLIC), the operation is executed to all the SQL (FOR ALL), and the performed tuple is the one whose "uname" column's value is the same as the current username (current_user function).

Example 193 CREATE POLICY statement

```
postgres=> CREATE POLICY pol1 ON poltbl1 FOR ALL USING (uname = current_user) ;
```

```
CREATE POLICY
```

```
postgres=> \d+ poltbl1
```

```

                                Table "public.poltbl2"
Column |          Type          | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
c1     | numeric                |           | main    |              |
c2     | character varying(10) |           | extended |              |
uname  | character varying(10) |           | extended |              |

```

```
Policies:
```

```
    POLICY "pol1" FOR ALL
```

```
    USING ((uname)::name = "current user"())
```

Information of the created policy can be checked from pg_policy catalog. The information of the table that has been set a policy can be checked from pg_policies catalog.

Following example verifies the effect of the policy:

- User "user1", who is the owner of the table poltbl1, stores 3 records (2-12 lines).
- Although table tblpol1 is searched with the privilege of user "user2", only one record whose "uname" column's value is "user2" is referenced (15-19 lines).
- Though the user was trying to change the value of "uname" column, UPDATE statement failed because of the deviation of the condition specified in USING clause of the CREATE POLICY statement (20-21 lines).

Example 194 Effect of the policy

```

1  $ psql -U user1
2  postgres=> INSERT INTO tblpol1 VALUES (100, 'Val100', 'user1') ;
3  INSERT 0 1
4  postgres=> INSERT INTO tblpol1 VALUES (200, 'Val200', 'user2') ;
5  INSERT 0 1
6  postgres=> INSERT INTO tblpol1 VALUES (300, 'Val300', 'user3') ;
7  INSERT 0 1
8  postgres=> SELECT COUNT(*) FROM tblpol1 ;
9    count
10  -----
11         3
12  (1 row)
13
14  $ psql -U user2
15  postgres=> SELECT * FROM poltbl1 ;
16    c1 |   c2   | uname
17  -----+-----+-----
18    200 | val200 | user2
19  (1 row)
20  postgres=> UPDATE poltbl1 SET uname='user3' ;
21  ERROR:  new row violates row level security policy for "poltbl1"
```

CREATE POLICY statement, which creates a policy, does not result in an error when executed to the table which is not specified ENABLE ROW LEVEL SECURITY clause. In this case, ROW LEVEL SECURITY feature is not enabled for the corresponding table; only the authorization by the GRANT statement is enabled.

To change or delete policy setting is performed in ALTER POLICY statement or DROP POLICY statement, respectively. When specified more than one policy for a table, the record that matches the logical OR is taken.

Example 195 Effect of multiple policy

```

1  $ psql -U user1
2  postgres=> CREATE TABLE pol1(c1 NUMERIC, c2 NUMERIC) ;
3  CREATE TABLE
4  postgres=> ALTER TABLE pol1 ENABLE ROW LEVEL SECURITY ;
5  ALTER TABLE
6  postgres=> CREATE POLICY p1_pol1 ON pol1 FOR ALL USING (c1 = 100) ;
7  CREATE POLICY
8  postgres=> CREATE POLICY p2_pol1 ON pol1 FOR ALL USING (c2 = 100) ;
9  CREATE POLICY
10 postgres=> GRANT SELECT ON pol1 TO user2 ;
11 GRANT
12
13 $ psql -U user2
14 postgres=> EXPLAIN SELECT * FROM pol1 ;
15              QUERY PLAN
16  -----
17   Seq Scan on pol1  (cost=0.00..23.20 rows=9 width=64)
18     Filter: ((c2 = '100'::numeric) OR (c1 = '100'::numeric))
19   (2 rows)

```

7.2.4 Parameter Settings

Feature of Row Level Security is controlled by the parameter row_security. It can take the following values.

Table 79 Choices for Parameter row_security

Value	Description
on	Enable the feature of Row Level Security. This is the default value.
off	Disable the feature of Row Level Security. Since the access control by POLICY disabled, only the permission by GRANT statement is enabled.
force	It will force the feature of Row Level Security. The permission of the POLICY is enforced to the table to which the POLICY is set. Therefore, even the owner of the table will not be able to access the data with policy violations.

□ User Privileges

Superuser with BYPASSRLS privilege will be able to bypass the policy setting. General user who has only BYPASSRLS privilege cannot ignore the POLICY. NOBYPASSRLS is specified as the default of CREATE USER statement.

8. Utilities

8.1 Utility usage

This chapter explains how to use the distinctive command.

8.1.1 Pg_basebackup command

Pg_basebackup¹³ command was developed in order to create a complete copy of the database cluster. It executes the same process as online backup internally. On using this command, note the following points.

- With the -x option, conduct a log switch after the backup completion.
- Table space directory other than the database cluster is stored in the same path. In order to change the path, old and new path must be specified in --tablespace-mapping parameters (it can be specified from PostgreSQL 9.4).
- Backup destination directory must be left empty.
- WAL writing directory is {PGDATA}/pg_xlog. If you want to specify a different directory, specify the --xlogdir parameters (can be specified from PostgreSQL 9.4).

Example 196 pg_basebackup command execution

```
$ pg_basebackup -D back -h hostsrc1 -p 5432 -x -v
Password: {PASSWORD}
transaction log start point: 0/7E000020
transaction log end point: 0/7E0000A8
pg_basebackup: base backup completed
$
```

□ Data transfer operation

Processing of pg_basebackup command is mostly executed within wal sender process on the destination instance. Wal sender process does the following processing.

- Execute pg_start_backup function
- Transfer of files in the database cluster
- External table space entity search and transfer of files
- Execute pg_stop_backup function
- In case of WAL transfer setting, transfer of WAL files

¹³ Online Manual <https://www.postgresql.org/docs/9.6/static/app-pgbasebackup.html>

Data of the file to be backed up is loaded into each 32 KB¹⁴, and sent to the client.

□ Copied file

Pg_basebackup command performs a copy of the database cluster, but it does not mean to copy all of the files. The following files (and directories) are different from the source of the database cluster.

Table 80 pg_basebackup command and difference of the source of the

Files/Directories	Difference	Note
backup_label	Create New	
backup_label.old	Create New	Former backup_label file
pg_replslot	Not copied	Replication slot
pg_stat_tmp	Not copied	
pg_xlog	Not copied	If -x is not specified
	Some are not copied	If -x is specified
postmaster.opts	Not copied	
postmaster.pid	Not copied	
PostgreSQL unmanaged files for the external table in the area	Not copied	

□ Restriction of transfer rate

If you specify the --max-rate parameter when executing the pg_basebackup command, you can limit the network transfer rate per unit time (PostgreSQL 9.4 new feature). Control of the transfer amount is executed within wal sender process. Each time the amount of data transfer exceeds 1/8 of the number of the byte specified in the parameter, waiting by latch timeout is performed to suppress the amount of data transferred within a certain period of time.

□ Recovery.conf file

If you specify the -R parameter to pg_basebackup command, recovery.conf file is automatically created. The created recovery.conf file contains only primary_conninfo parameters and standby_mode = 'on'. Even if it recovery.conf file already exists in the backup source of the database in the cluster (in case of getting the backup from the slave instance), specifying the -R parameter creates a new recovery.conf file. If -R parameter is not specified, existing recovery.conf files are copied to the backup destination directory.

¹⁴ Defined by TAR_SEND_SIZE in src/backend/replication/basebackup.c

□ Copy of Replication Slot

In `pg_basebackup` command, slot information that was created in `pg_create_physical_replication_slot` function is not copied. Therefore, when `pg_basebackup` command is completed, the copy destination `{PGDATA}/pg_replslot` directory will be empty.

□ User-created files and directories

The behavior when a directory and a file unrelated to PostgreSQL are created in the database cluster is verified.

Example 197 Copy of the user-created files (in database clusters)

```
$ touch data/test_file.txt
$ mkdir data/test_dir/
$ touch data/test_dir/test_file.txt
$ pg_basebackup -D back -p 5432 -v -x
transaction log start point: 0/2000028 on timeline 1
transaction log end point: 0/20000C0
pg_basebackup: base backup completed
$ ls back/test*
back/test file.txt

back/test dir:
test file.txt
```

It is confirmed that directories and files created in the database cluster and not managed by PostgreSQL are backed up by `pg_basebackup` command. Then confirmed whether the directory / file that was created on a tablespace that has been created in the `CREATE TABLESPACE` statement is copied or not.

Example 198 Copy of the user-created file (table space)

```
$ mkdir ts1
postgres=# CREATE TABLESPACE ts1 OWNER demo LOCATION '/usr/local/pgsql/ts1' ;
CREATE TABLESPACE
$ touch ts1/test_file.txt
$ mkdir ts1/test_dir/
$ touch ts1/test_dir/test_file.txt
$ pg_basebackup -D back -p 5432 -v -x
    --tablespace-mapping=/usr/local/pgsql/ts1=/usr/local/pgsql/back_ts1
transaction log start point: 0/2000028 on timeline 1
transaction log end point: 0/20000C0
pg_basebackup: base backup completed
$ ls back_ts1/
PG_9.6_201608131
```

As seen in the above example, a user-created file that is created within a table space, is confirmed to be not copied.

□ Behavior of --xlog-method parameter

The --xlog-method (or -X) parameter is used to specify the method of transfer WAL file. If this parameter specified, --xlog (or -x) parameter can not be specified. But WAL file is transferred to the backup destination from the source instance. If you specify a --xlog-method = streaming, it will launch two wal sender process at the same time in the PostgreSQL instance.

Table 81 Process to start by pg_basebackup command

Process name	Description
postgres: wal sender process ... sending backup ...	Process for file transfer
postgres: wal sender process ... streaming ...	Process for WAL transfer

8.1.2 Pg_archivecleanup command

Pg_archivecleanup¹⁵ command deletes archived log files that are no longer needed because of the backup completion.

Usually it is used as a parameter value of archive_cleanup_command in a recovery.conf file under

¹⁵ Online Manual <https://www.postgresql.org/docs/9.6/static/pgarchivecleanup.html>

the streaming at the replication environment. Specify the archive log output directory to the first parameter, and "%r" indicating the final WAL file to the second parameter to the second parameter.

Example 199 Specify in a recovery.conf file

```
archive_cleanup_command = 'pg_archivecleanup /usr/local/pgsql/archive %r'
```

Pg_archivecleanup command can also be used in a stand-alone environment. To the second parameter, specify the label file created in the online backup. By creating a program such as following example, the final label file can be obtained.

Example 200 pg_archivecleanup command execution script

```
#!/bin/sh

export PATH=/usr/local/pgsql/bin:${PATH}

ARCHDIR=/usr/local/pgsql/archive
LASTWALPATH=`/bin/ls $ARCHDIR/*.backup | /bin/sort -r | /usr/bin/head -1`

if [ $LASTWALPATH = '' ]; then
    echo 'NO label file found.'
    exit 1
fi

LASTWALFILE=`/bin/basename $LASTWALPATH`

pg_archivecleanup $ARCHDIR $LASTWALFILE
stat=$?

echo 'Archivelog cleanup complete'
exit $stat
```

8.1.3 Psql command

Psql command¹⁶ is a client tool to run the interactive SQL statements. Environment variables used

¹⁶ Online Manual <https://www.postgresql.org/docs/9.6/static/app-psql.html>

by psql command is as follows.

Table 82 Environment variables used by psql command

Environment Variable	Description	Default value
COLUMNS	New line width limits Default value of \pset columns	Calculated from the width of the terminal
PAGER	Pager command name	Under the Cygwin environment "less", otherwise "more"
PGCLIENTENCODING	Client encoding	auto
PGDATABASE	Default database name	OS username
PGHOST	Default host name	localhost
PGPORT ¹⁷	Default port number	5,432
PGUSER	Default database username	OS username
PSQL_EDITOR	Editor name to be used in the \e command. Search from the top of the list.	Linux/Unix: "vi"
EDITOR		Windows: "notepad.exe"
VISUAL		
PSQL_EDITOR_LINEN UMBER_ARG	Command for passing a line number to the editor	Linux/UNIX: "+", Windows: no setting
COMSPEC	Command shell for \! (Windows)	cmd.exe
SHELL	Command shell for \! (Linux/UNIX)	/bin/sh
PSQL_HISTORY	History save file	Linux/UNIX: {HOME}/.psql_history Windows: {HOME}\psql_history
PSQLRC	The path for the initialization command file	Linux/UNIX: {HOME}/.psqlrc Windows: {HOME}\psqlrc.conf
TMPDIR	Temporary directory for file editing	Linux/UNIX: /tmp Windows: result of GetTempPath API
PGPASSFILE	Password file	Linux: \$HOME/.pgpass Windows: %APPDATA%\postgr esql\pgpass.conf

¹⁷ It is also used as the default value for the instance startup waiting for a connection port number.

8.1.4 Pg_resetxlog command

Pg_resetxlog¹⁸ command performs a re-creation of the WAL file. This command can not be executed during instance startup. As in the manual, this command checks the presence of {PGDATA}/postmaster.pid file. It checks only the existence of the file, and it does not mean to check the activation of the instance.

Pg_resetxlog command executes the following processing.

1. Check the command option
2. Check the database cluster and move to directory
3. Presence check of postmaster.pid file
4. Read the pg_control file
 - a. If cannot read, exit program
 - b. Check the version number and CRC
5. Predict the correct value if there is a mismatch in the pg_control file
6. Search the final WAL files from pg_xlog directory
7. Check whether the previous instance is successfully completed (DB_SHUTDOWNED (1)); if it does not complete successfully, end unless the -f option is specified.
8. Delete and re-create the pg_control file
9. Delete WAL file
10. Delete file in the archive_status directory
11. Create a new WAL file

The following is a comparative example of the output result of pg_controldata command before and after the execution of pg_resetxlog command. Only the different parts are shown.

¹⁸ Online Manual <https://www.postgresql.org/docs/9.6/static/app-pgresetxlog.html>

Table 83 Comparison of pg_controldata command

Output item	Before pg_resetxlog command	After pg_resetxlog command
pg_control last modified	Fri Feb 11 15:22:13 2017	Fri Feb 11 16:02:40 2017
Latest checkpoint location	2/A4000028	2/AC000028
Prior checkpoint location	2/A3002D20	0/0
Latest checkpoint's REDO location	2/A4000028	2/AC000028
Latest checkpoint's REDO WAL file	0000000100000002000000A4	0000000100000002000000AC
Time of latest checkpoint	Fri Feb 11 15:22:01 2017	Fri Feb 11 16:02:26 2017
Backup start location	0/E1000028	0/0

8.1.5 Pg_rewind command

Pg_rewind¹⁹ command was added in PostgreSQL 9.5.

□ Overview

Pg_rewind command is a tool to build a replication environment. Unlike pg_basebackup command, it can perform a synchronization for existing database cluster. It is assumed the resynchronization between promoted slave instance and the old master instance.

□ Parameters

Following parameters can be specified to pg_rewind command.

¹⁹ Online Manual <https://www.postgresql.org/docs/9.6/static/app-pgrewind.html>

Table 84 Command parameters

Parameter name	Description
-D / --target-pgdata	Directory of the database cluster to be performed the update
--source-pgdata	Source directory of data acquisition
--source-server	Connection information of data acquisition source (remote instance)
-P / --progress	Output of the execution status
-n / --dry-run	Execute run simulation
--debug	Display debugging information
-V / --version	Display version information
-? / --help	Display how to use

□ Conditions

In order to execute the `pg_rewind` command, there are some conditions. `Pg_rewind` command checks the conditions to execute from the contents of the `pg_control` file of source and target.

First, it is necessary to set the PostgreSQL instance's parameter `wal_log_hints` to "on" (default value "off"), or enable the function of checksum, parameter `full_page_writes` should also be set to "on" (the default value "on").

Example 201 Error messages on the parameter settings

```
$ pg_rewind --source-server='host=remhost1 port=5432 user=postgres'
      --target-pgdata=data -P
connected to remote server

target server need to use either data checksums or "wal_log_hints = on"
Failure, exiting
```

The targeted instance of the database cluster must be stopped normally.

Example 202 The error message during target instance startup

```
$ pg_rewind --source-server='host=remhost1 port=5432 user=postgres'
      --target-pgdata=data -P
target server must be shut down cleanly
Failure, exiting
```

Copy processing of the data uses the connection to wal sender process similar to the `pg_basebackup` command. `Pg_hba.conf` file setting of the destination (data provider) instance, and setting of the

max_wal_senders parameter are necessary.

□ Execution procedure

In order to run pg_rewind, following procedure should be done. In the example below, pg_rewind connects to the slave instance of performing the promotion, and the old master instance is set to the new slave instance.

1. Parameter settings

Set parameters for the current master instance and pg_hba.conf file. Reload the file information, if necessary.

2. Stop target instance

Stop the synchronization instance (the old master).

3. Execute pg_rewind command

In the old master instance (which updates the data) run the pg_rewind command. First run with the -n parameter for test, and after the test, run without the -n parameter.

Example 203 Execute pg_rewind -n command

```
$ pg_rewind --source-server='host=remhost1 port=5432 user=postgres'
--target-pgdata=data -n
The servers diverged at WAL position 0/50000D0 on timeline 1.
Rewinding from last common checkpoint at 0/5000028 on timeline 1
Done!
```

Example 204 Execute pg_rewind command

```
$ pg_rewind --source-server='host=remhost1 port=5432 user=postgres'
--target-pgdata=data
The servers diverged at WAL position 0/50000D0 on timeline 1.
Rewinding from last common checkpoint at 0/5000028 on timeline 1
Done!
```

4. Edit recovery.conf file

Pg_rewind command does not create recovery.conf file in the update destination database cluster. For this reason, recovery.conf file is created for the new slave instance. Specify the replication slot name

created in the previous section.

5. Edit postgresql.conf file

By the execution of `pg_rewind` command, postgresql.conf file has been copied from the remote host and overwritten. Edit the parameters as needed.

6. Start new slave instance

Starup the new slave instance.

□ Command exit status

`Pg_rewind` command exits with 0 when the process is completed successfully, or exits with 1 if it fails.

8.1.6 Vacuumdb command

`Vacuumdb` command executes `VACUUM` processing forcibly. It can be specified the `--jobs` parameters in order to actively use the multiple cores. With the `--jobs` parameter, specify the number of jobs to be processed in parallel. Here, the detailed implementation of `--jobs` parameters is examined. The number of jobs is more than 1, less than "macro `FD_SETSIZE - 1`" (in Red Hat Enterprise Linux 7 1,023 or less).

Example 205 Upper and lower limits of the `--jobs` parameters

```
$ vacuumdb --jobs=-1
vacuumdb: number of parallel "jobs" must be at least 1
$ vacuumdb --jobs=1025
vacuumdb: too many parallel jobs requested (maximum: 1023)
```

□ `--jobs` parameter and the number of sessions

If a number is specified to the `--jobs` parameter, the same number of sessions as specified in the parameter created. If you perform a `VACUUM` for all database (`--all` specified) is, each database. If perform a `VACUUM` on a single database, do the processing in parallel to each table. The default value for this parameter is 1, and it is the same behavior as previous versions. In the example below, as `--jobs=10` is specified, 10 postgres processes started.

Example 206 Specifying the --jobs parameter and session

```
$ vacuumdb --jobs=10 -d demodb &
vacuumdb: vacuuming database "demodb"

$ ps -ef|grep postgres
postgres 14539      1  0 10:59 pts/2  00:00:00 /usr/local/pgsql/bin/postgres -D data
postgres 14540 14539  0 10:59 ?        00:00:00 postgres: logger process
postgres 14542 14539  0 10:59 ?        00:00:00 postgres: checkpointer process
postgres 14543 14539  0 10:59 ?        00:00:00 postgres: writer process
postgres 14544 14539  0 10:59 ?        00:00:00 postgres: wal writer process
postgres 14545 14539  0 10:59 ?        00:00:00 postgres: stats collector process
postgres 14569 14539  6 11:00 ?        00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14570 14539  0 11:00 ?        00:00:00 postgres: postgres demodb [local] idle
postgres 14571 14539  5 11:00 ?        00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14572 14539  7 11:00 ?        00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14573 14539  0 11:00 ?        00:00:00 postgres: postgres demodb [local] idle
postgres 14574 14539  0 11:00 ?        00:00:00 postgres: postgres demodb [local] idle
postgres 14575 14539  9 11:00 ?        00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14576 14539  5 11:00 ?        00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14577 14539  0 11:00 ?        00:00:00 postgres: postgres demodb [local] idle
postgres 14578 14539  1 11:00 ?        00:00:00 postgres: postgres demodb [local] idle
```

If the value specified in the --jobs parameter is above the number of --table parameters, the upper limit of the degree of parallelism become the number of tables. Since PostgreSQL max_connections parameter is not considered in the calculation of the number of sessions, the excess of the number of sessions is detected, "FATAL: sorry, too many clients already" error will occur.

8.2 Exit status of Server/Client Applications

The exit status of the various utilities that accompany with PostgreSQL is confirmed here.

8.2.1 Pg_ctl command

Pg_ctl command returns 0 if the operation was successful. If -w option is not specified at the instance startup (pg_ctl start) it exits with 0 at the time the background processing completes successfully by the system(3) function, therefore it is not mean the successful completion of the instance startup.

Table 85 Exit status of pg_ctl command

Status Code	Description	Note
0	Operation succeeds	
1	Operation fails Output a message to the standard error (if -l option is specified, to the log file).	
3	The instance is not running at the time of pg_ctl status command execution	
4	The database cluster does not exist when pg_ctl status command execution	9.4-

8.2.2 Psql command

Psql command returns 0 if the operation was successful. In case of specifying the SQL statement with the -c option, it returns 1 if the operation fails. However, in case of specifying the SQL statement with the -f option, it returns 0 if the operation fails. The behavior of "-f" option is changed by setting the ON_ERROR_STOP attribute to true (or 1). ON_ERROR_STOP attribute is set either by running \set command, or specifying the --set option of psql command.

Table 86 Exit status of psql command

Status Code	Description
0	Operation succeeds
1	Operation fails
2	Connection failure or flex-related errors
3	An error occurs after running the \set ON_ERROR_STOP true



Example 207 Error detection of psql command

```
$ psql -c 'SELECT * FROM notexists'
Password: {PASSWORD}
ERROR:  relation "notexist" does not exist
LINE 1: SELECT * FROM notexist
                        ^

$ echo $?
1

$ cat error.sql
SELECT * FROM notexists ;

$ psql -f error.sql
Password: {PASSWORD}
psql:error.sql:1: ERROR:  relation "notexists" does not exist
LINE 1: SELECT * FROM notexists ;
                        ^

$ echo $?
0

$ psql -f error.sql--set=ON_ERROR_STOP=true
Password: {PASSWORD}
psql:error.sql:1: ERROR:  relation "notexists" does not exist
LINE 1: SELECT * FROM notexists ;

$ echo $?
3

$ cat stop.sql
\set ON_ERROR_STOP true
SELECT * FROM notexists ;

$ psql -f error.sql
Password: {PASSWORD}
psql:error.sql:1: ERROR:  relation "notexists" does not exist
LINE 1: SELECT * FROM notexists ;
                        ^

$ echo $?
3
```

8.2.3 Pg_basebackup command

Pg_basebackup command returns 0 if the operation was successful. When processing failed, it outputs a message to the standard output, then exits with 1.

Table 87 Exit status of pg_basebackup command

Status Code	Description
0	Operation succeeds
1	Operation fails

8.2.4 Pg_archivecleanup command

Pg_archivecleanup command returns 0 if the operation was successful. When the operation failed, it outputs a message to the standard error, then exits with 2.

Table 88 Exit status of pg_archivecleanup command

Status Code	Description
0	Operation succeeds (or output help message)
2	Operation fails

8.2.5 Initdb command

Initdb command returns 0 if the operation was successful. When the operation failed, it outputs a message to the standard output, then exits with 1.

Table 89 Exit status of initdb command

Status Code	Description
0	Operation succeeds
1	Operation fails

8.2.6 Pg_isready command

Pg_isready command checks the parameters, and returns 3 if an invalid parameter is specified. Then it executes the PQpingParams function (src/interfaces/libpq/fe-connect.c) and ends with the return value of the function. The following values are returned.

Table 90 Exit status of pg_isready command

Status Code	Description	Remark
0	Instance is working and possible to accept connection	PQPING_OK
1	Instance is working but cannot accept the connection	PQPING_REJECT
2	Communication to the instance disabled	PQPING_NO_RESPONSE
3	Parameter or connection string incorrect	PQPING_NO_ATTEMPT

The macros of remarks are defined in the header (src/interfaces/libpq/fe-connect.h).

8.2.7 Pg_receivexlog command

Pg_receivexlog command returns 0 if the operation was successful. When the operation failed, it outputs a message to the standard output, then exits with 1.

Table 91 Exit status of pg_receivexlog command

Status Code	Description	Note
0	Operation succeeds	Status is zero in the following case: <ul style="list-style-type: none"> • Receive a SIGINT signal • Specified --help parameter • Specified --version parameter
1	Operation fails	

9. System Configuration

9.1 Default Value of Parameters

Parameters to be used in the PostgreSQL are described in the postgresql.conf file in the database cluster. The rows which start with '#' are treated as comment. After executing initdb command, the changed parameters are investigated.

9.1.1 Parameters derived at initdb command execution

Some of the parameters derive the value from the environment variable or setting status of host at the time of initdb command execution, and set it in the postgresql.conf file.

Table 92 Parameters set at the initdb command execution

Parameter	Setting	Default Value
max_connections	100	100
shared_buffers	128MB	8MB
dynamic_shared_memory_type	posix	posix
log_timezone	Derived from the environment variable	GMT
datestyle	Derived from the environment variable	ISO,MDY
timezone	Derived from the environment variable	GMT
lc_messages	Derived from the environment variable	-
lc_monetary	Derived from the environment variable	C
lc_numeric	Derived from the environment variable	C
lc_time	Derived from the environment variable	C
default_text_search_config	Derived from the environment variable	'pg_catalog.simple'

9.2 *Recommended Setting*

In PostgreSQL, many parameters and attributes are defined, and changed it if necessary. It is recommended to use the following values at first as the initial state.

9.2.1 Locale setting

Recommended parameters value of initdb command to be specified when creating the database cluster is as follows. It is not recommended to use it unless the locale-related function clearly needed. In addition, for encoding, UTF-8 is recommended because its character set is large.

Table 93 Recommended parameters of initdb command

Parameter	Recommended Value	Note
--encoding	UTF8	Or EUC_JIS_2004 (for Japanese)
--locale	Not specified	
--no-locale	Specified	
--username	postgres	
--data-checksums	Specified	Determined by the application requirements

9.2.2 Recommended parameter values

Recommended parameter settings in a typical system is as follows.

Table 94 Recommended parameters to be set in the postgresql.conf file

Parameter name	Recommended value	Note
archive_command	'test ! -f {ARCHIVEDIR}/%f && cp %p {ARCHIVEDIR}/%f'	
archive_mode	on	
autovacuum_max_workers	Greater than or equal to the number of database	
max_wal_size	2GB	
checkpoint_timeout	30min	
checkpoint_warning	30min	
client_encoding	utf8	
effective_cache_size	Amount of installed memory	
log_autovacuum_min_duration	60	
log_checkpoints	on	
log_line_prefix	'%t %u %d %r '	
log_min_duration_statement	30s	
log_temp_files	on	
logging_collector	on	
maintenance_work_mem	32MB	
max_connections	Expected number of connections or more	
max_wal_senders	Slave instance number +1 or more	
server_encoding	utf8	
shared_buffers	1/3 of the amount of installed memory	
tcp_keepalives_idle	60	
tcp_keepalives_interval	5	
tcp_keepalives_count	5	
temp_buffers	8MB	
timezone	Default	
wal_buffers	16MB	
work_mem	8MB	
wal_level	replica	
max_replication_slots	Slave instance number +1 or more	Replication

10. Streaming Replication

This chapter makes a brief description of streaming replication available from PostgreSQL 9.0.

10.1 Mechanism of streaming replication

PostgreSQL provides the replication feature that synchronize the instances and data running on the remote host. This section describes the streaming replication feature which is a standard PostgreSQL replication function.

10.1.1 The streaming replication

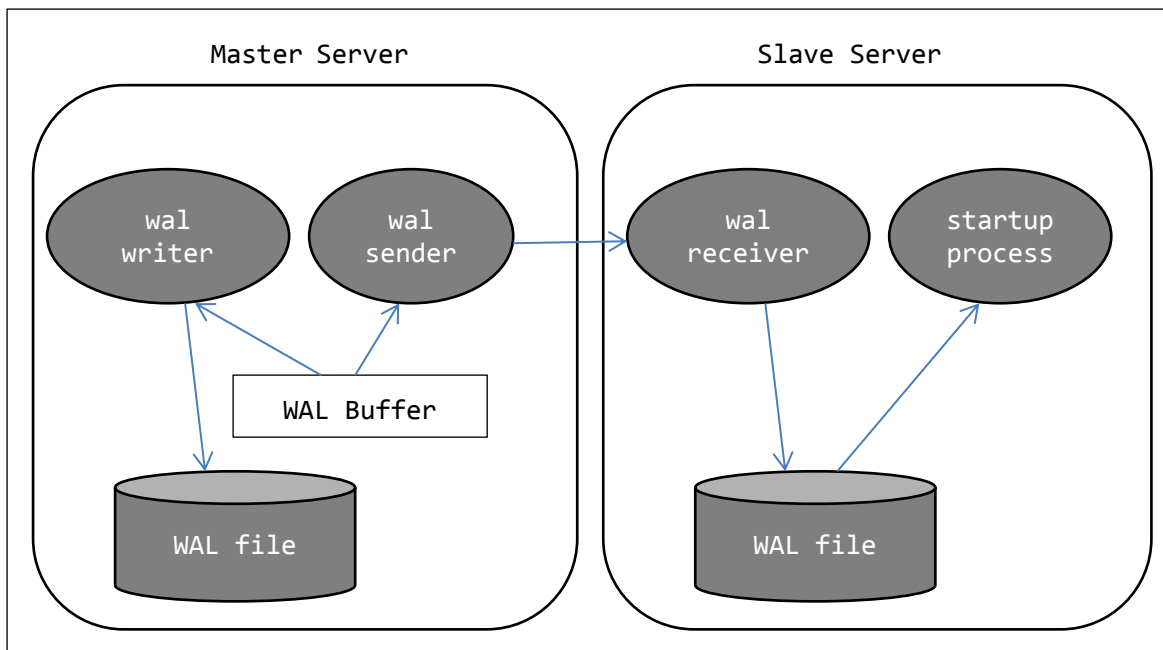
In previous version, i.e, older than PostgreSQL 9.0, Slony-I or pgpool-II, an independent tool of PostgreSQL, was used for the replication of the database. These tools are still effective today, but from PostgreSQL 9.0, streaming replication capabilities to perform the replication by transferring the transaction log (WAL) is provided in the standard.

In streaming replication environment, the update process is always run in only one instance. The updating instance is called the master instance. WAL information generated by the updates to the database are transferred to the slave instance. Slave instances ensure the uniqueness of multiple databases by applying the WAL information received from the database. In slave instance, the recovery of the database is carried out in real time. Slave instance can be started in read-only state, and it will be able to perform a search (SELECT) on the table. For this reason, it can be used for distributed search load of the replication environment.

10.1.2 Configuration of streaming replication

As the streaming replication are the feature to update a replica by transferring and applying WAL, it requires the underlying database for WAL application. PostgreSQL obtains a copy of the database cluster to be a master, and makes it the base of the replication. When the update transaction for the master instance occurs, the local WAL is updated. Then wal sender process transfers the WAL information to the standby instance. In standby instance, wal receiver process receives the WAL information, and write the WAL in storage. Written WAL is applied to the database cluster asynchronously, and slave instance is updated.

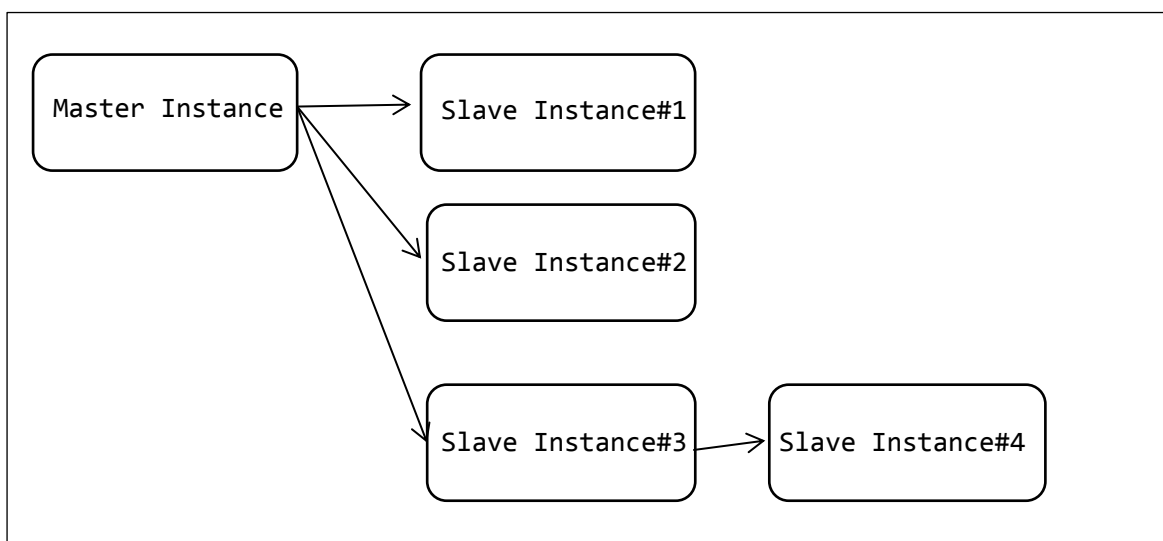
Figure 19 Streaming Replication



□ Cascading replication configuration

The simplest configuration of streaming replication is providing only one slave instance for the master instance. It is also possible to perform the replication for multiple slaves. In addition, a cascading replication configuration that regards a slave instance as a master instance of other instance is also available.

Figure 20 Cascading Replication configuration



10.2 Construction of the replication environment

This section describes how to build a replication environment.

10.2.1 Replication Slot

Streaming replication of PostgreSQL 9.4 or later creates an object called a "Replication Slot" in the master instance, and the slave instance manages the progress of replication by referring to the slot name. Replication before PostgreSQL 9.3 kept the WAL files, the number of which was equal to or less than the number specified in `wal_keep_segments` parameter, even if the slave instance stopped. By using the slots, WAL files necessary for the slave are automatically managed, and the feature has been changed that the master of the WAL is not deleted unless the slave receives. As the structure of the basic replication is not changed, the setting of parameters for wal sender process and `pg_hba.conf` file is still required.

□ Management of replication slot

Replication slots are managed by the following functions. Replication slot in use cannot be deleted. Streaming replication that is available from the past is called Physical Replication.

Syntax 8 Replication slot creation function

```
pg_create_physical_replication_slot('{SLOTNAME}')
```

```
pg_create_logical_replication_slot('{SLOTNAME}', '{PLUGINNAME}')
```

Syntax 9 Replication slot delete function

```
pg_drop_replication_slot('{SLOTNAME}')
```

The maximum number of replication slots that can be created in the database cluster is specified by the parameter `max_replication_slots`. As the default value for this parameter is 0, it must be changed in case of replicating. At instance startup, information related to the replication is expanded on the shared memory based on the value specified in the parameter `max_replication_slots`. Created replication slot information can be seen from `pg_replication_slots` catalog.

Example 208 Create and verify replication slot

```
postgres=# SELECT pg_create_physical_replication_slot('slot_1') ;
pg_create_physical_replication_slot
-----
(slot_1,)
(1 row)
postgres=# SELECT slot_name, active, active_pid FROM pg_replication_slots ;
 slot_name | active | active_pid 
-----+-----+-----
 slot_1    | f      | 
(1 row)
```

Replication slot created by the master instance, is referred from recovery.conf slave instance.

Example 209 Replication slot reference of recovery.conf file

```
primary_slot_name = 'slot_1'
primary_conninfo = 'host=hostmstr1 port=5433 application_name=prim5433'
standby_mode = on
```

When the replication succeeds, pg_stat_replication catalog and pg_replication_slots catalog are shown as follows:

Example 210 Checking the replication status

```
postgres=# SELECT pid, state, sync_state FROM pg_stat_replication ;
 pid | state | sync_state
-----+-----+-----
 12847 | streaming | async
(1 row)

postgres=# SELECT slot_name, slot_type, active, active_pid FROM
           pg_replication_slots ;
 slot_name | slot_type | active | active_pid
-----+-----+-----+-----
 slot_1    | physical | t      |      12847
(1 row)
```

In the `pg_stat_replication` catalog of PostgreSQL 9.4, `backend_xmin` column has been added. In the `pg_replication_slots` catalog of PostgreSQL 9.5, `active_pid` column has been added. In slave instance can be confirmed replication status in `pg_stat_wal_receiver` catalog.

□ Behavior caused by wrong setting

In the current implementation, replication will succeed even if the `primary_slot_name` parameter is not specified to the slave side. If the replication slot is not created on the master side despite the description of `primary_slot_name`, following error occurs.

Example 211 Errors in case of specifying a nonexistent replication slot name

```
FATAL: could not start WAL streaming: ERROR: replication slot "slot_1"
does not exist
```

If the replication slot is not found, replication cannot be executed, but an instance of the slave side starts. If a replication slot already used in parameter `primary_slot_name` is specified in case of starting multiple slave instance, the following error occurs. In this case, replication cannot be executed, although the slave instance starts.

Example 212 Errors if specifying a replication slot in use

```
FATAL: could not start WAL streaming: ERROR: replication slot "slot_1"
is already active
```

□ Entity of the replication slot

Entity of the slot is the directory and file with the same name as the slot name that has been created in the {PGDATA}/pg_replslot directory.

Example 213 Entity of the slot

```
$ ls -l data/pg_replslot/
total 4
drwx----- . 2 postgres postgres 4096 Feb 11 15:42 slot_1
$ ls -l data/pg_replslot/slot_1/
total 4
-rw----- . 1 postgres postgres 176 Feb 11 15:42 state
```

In cascaded the replication environment, create a slot in all instances to provide the WAL. Slave instance is read-only, but you can run the create function slot.

10.2.2 Synchronous and asynchronous

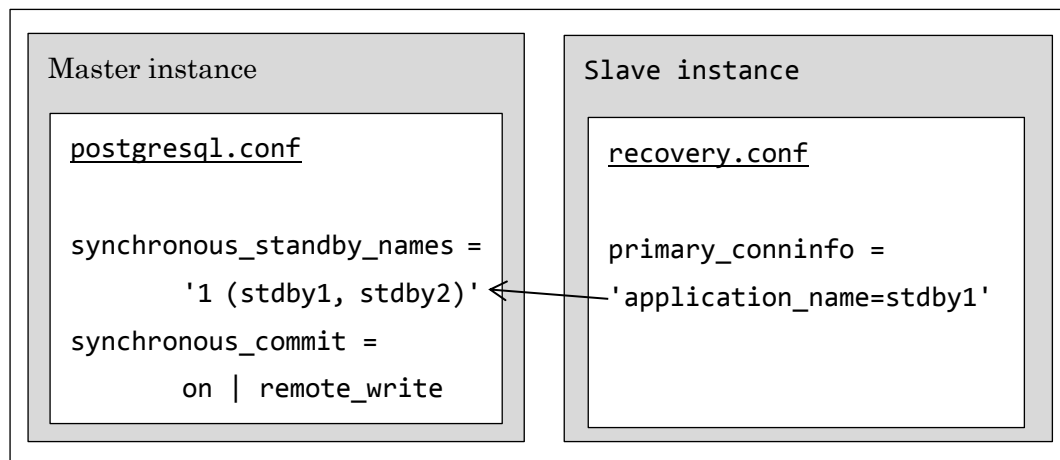
Streaming replication synchronizes with the slave instance by transferring the WAL information from the master instance. If the completion of the transaction is notified to the user after confirming that WALs of both master and slave instances are written, the reliability improves, though the performance declines significantly. On the other hand, if WAL to the slave instance is written asynchronously, the performance improves, however, the written transaction could be lost in case of the abnormal termination of the master instance before WAL arrival to the slave instance. For this reason, the streaming replication of PostgreSQL, offers five modes to select the balance of the reliability and performance. The choice of mode is determined by the parameter `synchronous_commit` of the master instance.

Table 95 Parameter `synchronous_commit` setting

Setting	Primary WAL	Slave WAL	Slave Apply	Note
on	Synchronous	Synchronous	Asynchronous	Synchronization until storage write
remote_write	Synchronous	Synchronous	Asynchronous	Synchronization until memory write
local	Synchronous	Asynchronous	Asynchronous	
off	Asynchronous	Asynchronous	Asynchronous	
remote_apply	Synchronous	Synchronous	Synchronous	PostgreSQL 9.6-

In order to execute synchronous replication, specify a list of any name (comma delimited) and number of synchronous replication instances to the parameter `synchronous_standby_names` of master instance. For slave instance, specify the same name in `application_name` items in `primary_conninfo` parameters of `recovery.conf` file.

Figure 21 Synchronous replication setting



Synchronous state of the replication can be found in the `sync_state` column of `pg_stat_replication` catalog. This column may have the following values.

Table 96 `sync_state` column value

Value	Description
sync	Synchronous replication
async	Asynchronous replication
potential	Asynchronous replication at present; it is changed to the synchronous replication if synchronous replication instance with higher priority stops.

Synchronization number of slave instances that replication is carried out is the description the value at the beginning of the parameter `synchronous_standby_names`. Priority to the high instance order synchronous replication is done. Priority is determined by the name specified in the parameter `synchronous_standby_names` of the master instance in the order from the left. Current priority can be found in the `sync_priority` column of `pg_stat_replication` catalog. In asynchronous replication, value of this column is always 0.

When the slave instance with a higher priority returns to the replication environment, instances for which synchronous replication is performed automatically switch.

□ Behavior at the time of the slave instance stop

In asynchronous replication environment, the master instance will continue to run even if the slave instance goes down. It is necessary to apply the stopped WAL for the slave instance in order to resume the replication and catch up the master instance. Under PostgreSQL 9.3, specify the number of WAL files held in `pg_xlog` directory in parameter `wal_keep_segments`. After PostgreSQL 9.4, WAL files not applied to slave are managed by the replication slot, and are now automatically maintained. In synchronous replication, when all slave instances stop, the master instance cannot execute the transaction and fall into a hung state.

10.2.3 Parameters

Parameters related to replication is as follows.

Table 97 Parameters for master instance

Parameter	Description	Setting value
<code>wal_level</code>	WAL output level	<code>hot_standby</code>
<code>archive_mode</code>	Archive output mode	<code>on</code> (or <code>always</code>)
<code>archive_command</code>	Archive output command	<code>cp</code> command, etc.
<code>max_wal_senders</code>	maximum number of wal sender process	Slave instance +1 or more
<code>max_replication_slots</code>	The maximum number of replication slots	Slave instance +1 or more
<code>synchronous_commit</code>	Synchronization commit	Application requirement
<code>synchronous_standby_names</code>	Synchronization commit name	Set in case of a synchronous replication

For executing SELECT statement to the slave instance, specify the following parameters. If default value is specified to the parameter `hot_standby`, it is not able to execute SELECT statement to the slave instance. This value does not affect even if specified to the master instance, but it can also be set in advance for the switch over.

Table 98 Parameters for slave instance

Parameter	Description	Setting
<code>hot_standby</code>	Set the slave to referable	on
<code>archive_mode</code>	Archive log output mode	off (or always)

10.2.4 Recovery.conf file

Slave instance receives the WAL information and make a recovery. For this reason, similar to the recovery of the database, it creates a `recovery.conf` file in the database cluster. The following parameters can be specified in `recovery.conf` file. Primary_slot_name parameters can be specified from PostgreSQL 9.4.

Table 99 Recovery.conf file for the slave instance

Parameter name	Description	Recommended Setting
<code>standby_mode</code>	Standby mode	on
<code>primary_slot_name</code>	Replication slot name	Replication slot name on the primary instance
<code>primary_conninfo</code>	Primary instance connection	Connection information
<code>restore_command</code>	Restore archive log command	Copy by scp from primary instance
<code>recovery_target_timeline</code>	Setting for timeline	Application Depend
<code>min_recovery_apply_delay</code>	Delay time for recovery	Setup time
<code>trigger_file</code>	Trigger file	file name

Example 214 Example of recovery.conf

```
standby_mode = 'on'
primary_slot_name = 'slot_1'
primary_conninfo = 'host=hostmstr1 port=5432 user=postgres password=secret
                    application_name=stdby1'
restore_command = 'scp hostmstr1:/usr/local/pgsql/archive/%f %p'
recovery_target_timeline='latest'
```

□ **Trigger_file** parameter

Trigger_file parameter is not necessary. If specified, startup process checks the file every 5 seconds, and if the file exists, slave instance is promoted to master.

□ **Primary_conninfo** parameter

In the primary_conninfo parameter, describe the information to connect to the master instance. Multiple description of the parameters is possible, separated by a space "parameter name = value". To "user" clause, specify a user name that has REPLICATION privileges.

Table 100 Parameters that can be specified in primary_conninfo

Parameter	Description	Note
service	Service name	
user	Connect user name	
password	Connect password	
connect_timeout	Connection timeout (seconds)	
dbname	Connect database name	
host	Connect hostname	Master instance host name
hostaddr	Connect host IP address	Master instance IP address
port	Connect port number	
client_encoding	Client encoding	
options	Options	
application_name	Application name	For Synchronous Replication

□ **Restore_command** parameter

Specify the command to get the archive log files required for recovery. If the slave instance has been stopped for a long time, the specified command is executed to get the archive log files needed to recover. If the slave instance is running on the remote host, files are copied using a command which does not require a password, such as scp command. %p parameter will be expanded to {PGDATA}/pg_xlog/RECOVERYXLOG. When the copy is complete, the RECOVERYXLOG file name is changed, then "{PGDATA}/pg_xlog/archive_status/{WALFILE}.done" file is created.

10.3 Failover and Switchover

Failover means the promotion of the slave instance to the master in case of the error in the master instance; switchover is the replacement the role of the master and the slave instances.

10.3.1 Procedure of switchover

In order to perform the switchover, perform the following steps.

1. Stop the master instance successfully
2. Stop the slave instance successfully
3. Modify the parameters of both instances, if necessary
4. Delete recovery.conf file of the old slave instance
5. Create a recovery.conf file of the old master instance
6. Startup both instances

10.3.2 Pg_ctl promote command

In order to promote the slave instance to the master instance, run the `pg_ctl promote` command to the slave instance. From immediately after execution of the command, the slave instance acts as the master instance. In the following example, `pg_ctl promote` command is executed on the master side, but it becomes an error because it is not in standby mode. In the execution of the command to the slave instance, "server promoting" message is output and it is found that the promotion is executed.

Example 215 pg_ctl promote command

```
$ pg_ctl -D data.master promote
pg_ctl: cannot promote server; server is not in standby mode
$ echo $?
1
$
$ pg_ctl -D data.slave promote
server promoting
```

Even after the slave instance was promoted, the former master instance keeps running.

□ Behavior of pg_ctl promote command

PostgreSQL failover is performed by the `pg_ctl promote` command to the slave instance. `Pg_ctl promote` command runs the following processing.

- Postmaster process ID acquisition and checking
 - Acquisition by performing the analysis of postmaster.pid file
- Check if Single-User Server or not
- Presence check of recovery.conf file
- Create {PGDATA}/promote file
- Send the SIGUSR1 signal to the postmaster process
- Output "server promoting" message

This operation is executed in the `do_promote` function of the `"src/bin/pg_ctl/pg_ctl.c"` file.

Postmaster process sends a SIGUSR2 signal to the startup process when it receives a SIGUSR1.

10.3.3 Promoted to the master by the trigger file

When the file that is specified in the `trigger_file` parameter of `recovery.conf` file is created, the slave instance is promoted to master. When promoted to the master instance is complete, the file is deleted. Promotion to master does not throw an error if it cannot delete the file. When the master promoted by the trigger file is output the following log.

Example 216 Log when promoted to master by the trigger file

```
LOG:  trigger file found: /tmp/trigger.txt
FATAL: terminating walreceiver process due to administrator command
LOG:  invalid record length at 0/6000060: wanted 24, got 0
LOG:  redo done at 0/6000028
LOG:  selected new timeline ID: 2
LOG:  archive recovery complete
LOG:  MultiXact member wraparound protections are now enabled
LOG:  database system is ready to accept connections
LOG:  autovacuum launcher started
```

10.3.4 Log on a failure

The log output in case of the master or slave instance stop is investigated under replication environment.

□ Stop of the slave instance

When the slave instance stops in smart or fast mode, nothing is output to the master instance log. In case of the abnormal termination of the slave instance following log is output.

Example 217 Abnormal termination log of the slave instance

```
LOG: unexpected EOF on standby connection
```

If the slave instance is restarted, following logs are output (in the case of synchronous replication only).

Example 218 Resuming log of synchronous replication

```
LOG: standby "stdby_1 " is now the synchronous standby with priority 1
```

□ Stop of the master instance

When the master instance is stopped, following log is output to the slave instance.

Example 219 Master instance stop log

```
FATAL: could not send data to WAL stream: server closed the connection  
unexpectedly
```

This probably means the server terminated abnormally
before or while processing the request.

When the master instance is restarted, the following log is output.

Example 220 Replication resume log

```
LOG: started streaming WAL from primary at 0/5000000 on timeline 1
```

11. Source code Tree

As PostgreSQL is an open source software, the source is public. Most of the source code of PostgreSQL written in C language. The source code can be downloaded from PostgreSQL Global Development Group website (<http://www.postgresql.org/ftp/source/>).

11.1 Directory Structure

11.1.1 Top directory

Expanding the downloaded source code, postgresql-`{VERSION}` directory is created. In the created directory, the following files and directories are also created.

Table 101 Top-level directories and files

File / Directory	Description
aclocal.m4	Part of the configure file
config	File directory for configure
config.log	Configure command execution log
config.status	Scripts that configure command generates
configure	Configure program
configure.in	Skeleton of configure program
contrib	Directory for Contrib module
doc	Document directory
src	Source code directory
COPYRIGHT	Copyright information
GNUmakefile	Top-level Makefile
GNUmakefile.in	Skeleton of Makefile
HISTORY	Describes a URL to view the release notes
INSTALL	Overview of installation
Makefile	Dummy Makefile
README	Overview of instructional materials

11.1.2 "src" directory

In the "src" directory, source code is stored in a hierarchical structure.

Table 102 Main directory in the "src" directory

Directory	Description
backend	Source code of the back-end process group
bin	Source code of the command such as pg_ctl
common	Commonly used source code
include	Header files
interfaces	Source code of libpq and ECPG library
makefiles	Makefile
pl	Source code of PL/Perl, PL/pgSQL, PL/Python and PL/tcl
port	Source code of libpgport library
template	Shell scripts for various OS
test	Build tests
timezone	Time-zone related information
tools	Build tools
tutorial	SQL Tutorials

11.2 Build Environment

11.2.1 Configure command parameters

In the case of outputting a Japanese message to the server log, the `--enable-nls` parameter of the "configure" command should be enabled. Only English messages are output at the default.

11.2.2 Make command parameters

After the configuration with the configure command, compile and install the binary using make command. Items that can be specified as the target of the make command are as follows.

Table 103 Main options for the make command

Target	Description
-	Build PostgreSQL binary
world	Build binary, Contrib module and HTML documents, etc.
check	Run regression tests
install	Install PostgreSQL binary
install-docs	Install HTML and man document
install-world	Install binary, Contrib module and HTML documents, etc.
clean	Remove binary

12. Linux Operating System Configuration

This chapter describes the settings that are recommended to change in the Linux environment in order to run PostgreSQL.

12.1 Kernel Parameters

12.1.1 Memory Overcommit

On Red Hat Enterprise Linux, memory overcommit feature is working by default. In order to prevent PostgreSQL instance from being killed because of its large memory usage in case of memory shortage, the functionality of the memory overcommit should be stopped.

Table 104 Memory over commit settings

Kernel Parameter	Default Value	Recommended Value
vm.overcommit_memory	0	2
vm.overcommit_ratio	50	99

12.1.2 I/O Scheduler

An example was presented that by changing I/O scheduler to "deadline", the irregularity of performance was eliminated. In the system, using SSD storage should also consider changing to "noop".

Example 221 Change the I/O scheduler to noop

```
# cat /sys/block/sda/queue/scheduler
noop [deadline] cfq
# grubby --update-kernel=ALL --args="elevator=noop"
```

As the default I / O scheduler in Red Hat Enterprise Linux 7 has been changed to "deadline", setting is generally no longer required.

12.1.3 SWAP

In order to maintain the process in memory as long as possible without swap out, the kernel parameters vm.swappiness is recommended to be set 5 or less.

Table 105 Swap setting

Kernel Parameter	Default value	Recommended value
vm.swappiness	30	0

12.1.4 Huge Pages

In a large-scale memory environment make the setting to use the Huge Pages. Huge Pages should be used if possible in the default configuration of PostgreSQL 9.4. For the kernel parameters `vm.nr_hugepages`, specify the size larger than the area to be used in shared memory (2 MB units). If the parameter `huge_pages` set to "on", when the required memory area is insufficient, an instance startup becomes an error.

Table 106 Huge Pages settings

Kernel Parameter	Default value	Recommended Value
vm.nr_hugepages	0	More than <code>shared_buffers</code> + <code>wal_buffers</code>

12.1.5 Semaphore

In systems where the number of simultaneous sessions is more than 1,000, there is a possibility of a shortage of semaphore set to be saved at instance startup. In the case of expanding the parameter `max_connections`, update the kernel parameters `kernel.sem`.

12.2 Filesystem Settings

PostgreSQL uses the file system as a storage, and it creates a many small files automatically. Therefore, the performance of the file system will greatly affect the performance of the system. In Linux environment, Ext4 or XFS (Standard at Red Hat Enterprise Linux 7) is recommended.

12.2.1 When using the ext4 filesystem

As mount options of the file system for the database cluster, specify `noatime` and `nodiratime`.

12.2.2 When using the XFS filesystem

As mount options of the file system for the database cluster, specify `nobarrier`, `noatime`, `noexec` and `nodiratime`.

12.3 Core File Settings

For the analysis of the trouble, core files that were generated at the time of the failure are useful. Here describes the setting to make PostgreSQL generate core files, using trouble analysis tool of Red Hat Enterprise Linux.

12.3.1 CORE file output settings

As the size limit of the core file is set to 0 by default, the restriction should be removed.

- Edit limits.conf file

Add the following entry to the /etc/security/limits.conf file. "postgres" is the PostgreSQL instance execution user name.

Example 222 core file limit

```
postgres - core unlimited
```

- Edit .bashrc file

In the "postgres" user's {HOME}/.bashrc file, add the following entry.

Example 223 user limit

```
ulimit -c unlimited
```

12.3.2 Core administration with ABRT

In the Red Hat Enterprise Linux 6 and later, Auto Bug Reporting Tool (ABRT) which automatically collect the necessary information for the bug report has been installed. ABRT is running automatically with standard installation.

- Kernel parameter settings

In ABRT installed environment, the kernel parameters kernel.core_pattern has been changed to the following settings.

Example 224 core_pattern kernel parameter value

```
 |/usr/libexec/abrt-hook-ccpp %s %c %p %u %g %t e
```

Therefore, when core file is generated, contents of the file is transferred to ABRT.

□ Directory creation and the output destination setting

Core file is output to the `/var/spool/abrt` directory by default. To change the directory, modify the `DumpLocation` parameters of `/etc/abrt/abrt.conf` file.

Example 225 Directory creation and the output destination setting

```
# mkdir -p /var/crash/abrt
# chown abrt:abrt /var/crash/abrt
# chmod 755 /var/crash/abrt
# cat /etc/abrt/abrt.conf
DumpLocation = /var/crash/abrt -- core output destination
MaxCrashReportsSize = 0          -- maximum core file size to unlimited
```

□ ABRT package settings

By default, the Core file for the program that has not been digitally signed will not be generated. In order to remove this restriction, set the `OpenPGPCheck` parameters of `/etc/abrt/abrt-action-save-package-data.conf` file to "no".

Example 226 Output the core for unsigned programs

```
# cat /etc/abrt/abrt-action-save-package-data.conf
OpenPGPCheck = no
```

□ Other settings

In the `/etc/abrt/plugins/CCpp.conf` file, specify generation rules and format of the Core file.

Example 227 Other core file settings

```
# cat /etc/abrt/plugins/CCpp.conf
MakeCompatCore = no
SaveBinaryImage = yes
```

Please refer to the following URL for ABRT details.

<https://access.redhat.com/documentation/en->

[US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/ch-abrt.html](https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/ch-abrt.html)

12.4 User limits

PostgreSQL instances on Linux works generally with the privileges of the Linux user postgres. On Red Hat Enterprise Linux 6 system where the number of concurrent connections is more than 1,000, extend the process limit of postgres user. The upper limit of the number of processes is written in /etc/security/limits.conf file.

Example 228 limits.conf file settings

postgres	soft	nproc	1024
postgres	hard	nproc	1024

In Red Hat Enterprise Linux 7, the default value of this limit is changed to 4096, therefore above deal is no longer required.

12.5 systemd support

In the Red Hat Enterprise Linux 7, systemd is used for service management of the operating system. In order to automatically start the PostgreSQL instance during the Linux boot, it is necessary to correspond to systemd.

12.5.1 Service registration

In order to correspond to systemd, create a script, and register to systemd daemon. In the following example, the service name is set to postgresql-9.6.2.service. It uses systemctl command to register the service.

Example 229 systemd registration

```
# vi /usr/lib/systemd/system/postgresql-9.6.2.service
#
# systemctl enable postgresql-9.6.2.service
ln -s /usr/lib/systemd/system/postgresql-9.6.2.service
   /etc/systemd/system/multi-user.target.wants/postgresql-9.6.2.service'
#
# systemctl --system daemon-reload
#
```

The following example is a sample script to be created in the `/usr/lib/systemd/system/` directory.

Example 230 systemd script

```
[Unit]
Description=PostgreSQL 9.6.2 Database Server
After=syslog.target network.target

[Service]
Type=forking
TimeoutSec=120

User=postgres

Environment=PGDATA=/usr/local/pgsql/data
PIDFile=/usr/local/pgsql/data/postmaster.pid

ExecStart=/usr/local/pgsql/bin/pg_ctl start -D "/usr/local/pgsql/data" -l
"/usr/local/pgsql/data/pg_log/startup.log" -w -t ${TimeoutSec}
ExecStop=/usr/local/pgsql/bin/pg_ctl stop -m fast -D "/usr/local/pgsql/data"
ExecReload=/usr/local/pgsql/bin/pg_ctl reload -D "/usr/local/pgsql/data"

[Install]
WantedBy=multi-user.target
```

12.5.2 Service start and stop

Starting and stopping services are performed by using the `systemctl` command. Please refer to the online manual of Red Hat Enterprise Linux for details of `systemctl` command.

Table 107 Control of services by `systemctl` command

Operation	Command
Start service	<code>systemctl start {SERVICENAME}</code>
Stop service	<code>systemctl stop {SERVICENAME}</code>
Service status check	<code>systemctl status {SERVICENAME}</code>
Restart service	<code>systemctl restart {SERVICENAME}</code>
Reload service	<code>systemctl reload {SERVICENAME}</code>

Example 231 Control of services by systemctl command

```
# systemctl start postgresql-9.6.2.service
# systemctl status postgresql-9.6.2.service
postgresql-9.6.2.service - PostgreSQL 9.6.2 Database Server
    Loaded: loaded (/usr/lib/systemd/system/postgresql-9.6.2.service; enabled)
    Active: active (running) since Tue 2017-02-11 12:02:00 JST; 5s ago
    Process: 12655 ExecStop=/usr/local/pgsql/bin/pg_ctl stop -m fast -w -D
/home/postgres/data (code=exited, status=1/FAILURE)
    Process: 12661 ExecStart=/usr/local/pgsql/bin/pg_ctl -w start -D
/home/postgres/data -l /home/postgres/data/pg_log/startup.log -w -t
${TimeoutSec} (code=exited, status=0/SUCCESS)
    Main PID: 12663 (postgres)
    CGroup: /system.slice/postgresql-9.6.2.service
            + /usr/local/pgsql/bin/postgres -D /home/postgres/data
            + postgres: logger process
            + postgres: checkpointer process
            + postgres: writer process
            + postgres: wal writer process
            + postgres: autovacuum launcher process
            + postgres: archiver process
            + postgres: stats collector process

Feb 11 12:02:00 rel71-2 systemd[1]: Starting PostgreSQL 9.6.2 Database Server...
Feb 11 12:02:00 rel71-2 systemd[1]: PID file
/home/postgres/data/postmaster.pid ...t.
Feb 11 12:02:00 rel71-2 pg_ctl[12661]: waiting for server to start... stopped
waiting
Feb 11 12:02:00 rel71-2 pg_ctl[12661]: server is still starting up
Feb 11 12:02:00 rel71-2 systemd[1]: Started PostgreSQL 9.6.2 Database Server.
Hint: Some lines were ellipsized, use -l to show in full.
```

Systemctl command manages the state of the service of the started process. If the instance started by systemctl command is stopped by pg_ctl command, systemd daemon determines that the service is terminated abnormally.

12.6 Others

12.6.1 SSH

In the replication environment without replication slot, use "restore_command" parameter of recovery.conf file in order to eliminate the gaps in the archived log. In this parameter, describe the command to copy the archive log files of the primary instance, and make the settings so that PostgreSQL can connect to the primary host using "scp" command without password.

12.6.2 Firewall

If you use a firewall, allow a connection to the local TCP port 5,432 (parameter "port"). The following example allows a connection to the service postgresql.

Example 232 firewalld setting

```
# firewall-cmd --permanent --add-service=postgresql
success
```

12.6.3 SE-Linux

At present, it seems that there is no clear guidance for the combination of SE-Linux and PostgreSQL. Typically, Permissive mode or Disabled mode is set.

12.6.4 systemd

In Red Hat Enterprise Linux 7 Update 2, when the user logs off the setting to delete a shared memory created by the user now the default. In the default state, in order to log off at the same time as PostgreSQL instance shared memory is deleted, return this setting to the same value as the old version. You should modify the /etc/systemd/logind.conf file as follows.

Example 233 logind.conf file setting

```
[login]
RemoveIPC=no           -- add this line
```

Appendix. Bibliography

Appendix.1 Books

Bellow is the information of the books that would be helpful for PostgreSQL.

Table 108 Books

Book Name	Author	Publisher
PostgreSQL Replication - Second Edition	Hans-Jurgen Schonig	PACKT
PostgreSQL 9 Administration Cookbook - Second Edition	Simon Riggs Gianni Ciolli	PACKT
Troubleshooting PostgreSQL	Hans-Jurgen Schonig	PACKT
PostgreSQL for Data Architects	Jayadevan Maymala	PACKT
PostgreSQL Server Programming - Second Edition	Usama Dar Hannu Krosing	PACKT
PostgreSQL Developer's Guide	Ahmed, Ibrar Fayyaz, Asif	PACKT
PostgreSQL Cookbook	Chitij Chauhan	PACKT
PostgreSQL: Up and Running	Regina O. Obe Leo S. Hsu	O'Reilly

Appendix 2. URL

Bellow is the information of the URL that would be helpful for PostgreSQL.

Table 109 URL

Name	URL
PostgreSQL Online Documents	http://www.postgresql.org/docs/
PostgreSQL JDBC Driver	http://jdbc.postgresql.org/
PostgreSQL GitHub	https://github.com/postgres/postgres
PostgreSQL Commitfests	https://commitfest.postgresql.org/
Michael Paquier - Open source developer based in Japan	http://michael.otacoo.com/
PostgreSQL 9.5 WAL format	https://wiki.postgresql.org/images/a/af/FOSDEM-2015-New-WAL-format.pdf
EnterpriseDB	http://www.enterprisedb.com/
PostgreSQL Internals (Japanese)	http://www.postgresqlinternals.org/index.php
PostgreSQL Deep Dive (Japanese)	http://pgsqldeepdive.blogspot.jp/
PostgreSQL Japan User Group	https://www.postgresql.jp/
Configuring and tuning HP ProLiant Servers for low-latency applications	http://h10032.www1.hp.com/ctg/Manual/c01804533.pdf

Modification History

Modification History

Version	Date	Author	Description
1.0	16-Jul-2014	Noriyoshi Shinoda	Create a first edition. PostgreSQL 9.4 Beta 1
1.0.1	04-Aug-2014	Noriyoshi Shinoda	Typo fixed
1.1	16-May-2015	Noriyoshi Shinoda	Create a second edition, supports PostgreSQL 9.4 official version 2.1.2 Information added 3.1.3 Fix error about TOAST function 3.3.8 Add archiver process behavior 3.9.1 Add locale function detail 3.10.3 Add log rotation 5.1.4 Add statistics information 5.2.3 Add autovacuum information 5.3.8 Add EXPLAIN detail 6.2.6 Add statistics information 6.3 Move postgres_fdw to (2) 6.4 Add PREPARE statement 12.1.6 Add Semaphore information 12.4 Add user restriction Add Appendix and URL information
1.1.1	18-May-2015	Noriyoshi Shinoda	Typo fixed
1.2	11-Feb-2017	Noriyoshi Shinoda	Create the third edition, support PostgreSQL 9.6 official version. Create English version. Unify the installation location to /usr/local/pgsql 2.1.5 Processes of Windows environment added 2.3.1/2.3.2/2.3.3 Some information added 2.3.7 Stop instance on Windows added 3.3.3 Visibility Map added 3.4.1/3.5.4 Some information added 3.5.5 pg_filenode.map added



Modification History (Cont.)

Version	Date	Author	Description
1.2	11-Feb-2017		3.5.4 pg_control detail added 3.10.5 Log file encoding information added 4.1.2 Some information added 4.3.2 Behavior of process termination added 5.2.4 amount of memory use added 5.3.4 Some information added 6.2.7 Partition table with External table added 6.5 INSERT ON CONFLICT added 6.6 TABLESAMPLE added 6.7 Changing table attribute added 6.9 Parallel Query added 7.2 Row Level Security added 8.1.1 Add information for pg_basebackup 8.1.5 pg_rewind added 8.1.6 vacuumdb added 12 Red Hat Enterprise Linux 7 support 12.1.6 Remove cstate setting Modify Appendix



Hewlett Packard
Enterprise



Hewlett Packard
Enterprise