**FILE LOCATIONS**

一般放置在安装时指定的数据目录中，如果想修改，可通过 postgresql.conf 配置文件指定

#data\_directory = 'ConfigDir'# 指定数据目录

#hba\_file = 'ConfigDir/pg\_hba.conf'#指定基于主机认证配置文件

#ident\_file = 'ConfigDir/pg\_ident.conf'#指定用于用户名称映射的配置文件

#external\_pid\_file = ''#指定可被服务器创建的用于管理程序的额外进程ID文件

**CONNECTIONS AND AUTHENTICATION**

**- Connection Settings -**

listen\_addresses = '\*'#配置服务器监听地址，0.0.0.0 允许监听所有 IPv4 地址， :: 允许监听所有 IPv6 地址。如果为空，则服务器不会侦听任何 IP 接口，只能使用 Unix 域套接字连接到它。默认值为 localhost，它只允许建立本地 TCP/IP “环回”连接。

port = 5432# 配置端口

max\_connections = 1000# 允许数据库的最大并发连接数，默认100

superuser\_reserved\_connections = 5# 预留给超级用户的连接池数，默认3

#unix\_socket\_directories = ''# 用于监听来自客户端连接的 Unix 域套接字目录，默认值通常是 /tmp

#unix\_socket\_group = ''# Unix域套接字所属组，默认为空字，表示它使用服务器用户的默认组

#unix\_socket\_permissions = 0777# Unix 域套接字的访问权限

#bonjour = off#通过Bonjour启用服务器存在的广告。 默认是关闭的。

# bonjour\_name = ''#声明Bonjour服务名称，默认值为空字符串， 表示使用本机名。

**- TCP settings -**

tcp\_keepalives\_idle = 60#表示在一个TCP连接空闲多长时间后会发送一个keepalive报文，默认值为0，表示使用操作系统设置的默认值(2小时)。

tcp\_keepalives\_interval = 10#两次发送TCP keepalive包的时间间隔， 默认值为0，表示使用操作系统设置的默认值(1秒)。

tcp\_keepalives\_count = 10#最多发送keepalive包的次数， 默认值为0，表示使用操作系统设置的默认值， Windows不支持此参数，所有必须设置为0。

#tcp\_user\_timeout = 0#指定传输的数据在TCP连接被强制关闭之前可以保持未确认状态的时间量。

#client\_connection\_check\_interval = 0# 两次检查的间隔时间

**- Authentication -**

#authentication\_timeout = 1min# 认证超时，取 1s-600s

#password\_encryption = scram-sha-256#密码加密算法：scram-sha-256 or md5

#db\_user\_namespace = off#启用针对每个数据库的用户名，如果这个参数为打开，应该把用户创建成username@dbname的形式。

#GSSAPI using Kerberos

#krb\_server\_keyfile = 'FILE:${sysconfdir}/krb5.keytab'#设置Kerberos服务器密钥文件的位置。

#krb\_caseins\_users = off#设置是否应该以大小写不敏感的方式对待GSSAPI用户名。

**- SSL -**

#ssl = off# 是否启用ssl 连接，默认关闭

#ssl\_ca\_file = ''#指定SSL服务器证书颁发机构（CA）的文件的名称

#ssl\_cert\_file = 'server.crt'#指定SSL服务器证书的文件的名称。

#ssl\_crl\_file = ''#指定SSL服务器证书吊销列表（CRL）的文件的名称。

#ssl\_key\_file = 'server.key'#指定服务器私钥文件名

#ssl\_ciphers = ‘HIGH:MEDIUM:+3DES:!aNULL’#指定可以使用的SSL加密算法。

HIGH ：使用来自HIGH组的密码的密码组（例如 AES, Camellia, 3DES）

MEDIUM：使用来自MEDIUM组的密码的密码组（例如 RC4, SEED）

+3DES： 使用DES密码组。

!aNULL：禁用不做认证的匿名密码组

#ssl\_prefer\_server\_ciphers = on#指定是否使用服务器的 SSL 密码首选项，而不是客户端的。

#ssl\_ecdh\_curve = 'prime256v1'# 指定用在ECDH密钥交换中的曲线名称，需要客户端支持。

#ssl\_min\_protocol\_version = ‘TLSv1.2‘#设置要使用的最小SSL/TLS版本

#ssl\_max\_protocol\_version = ‘‘#设置要使用的最大SSL/TLS版本

#ssl\_dh\_params\_file = ‘’#指定含有用于SSL密码的所谓临时DH家族的Diffie-Hellman参数的文件名。默认值为空。

#ssl\_passphrase\_command = ‘‘#设置当需要一个密码(私钥)来解密SSL文件时会调用的一个外部命令。参数为空，表示使用内建的提示机制。

#ssl\_passphrase\_command\_supports\_reload = off#这个参数决定在重载配置期间，如果一个密钥文件需要口令时，是否也调用ssl\_passphrase\_command设置的密码命令。

**RESOURCE USAGE (except WAL)**

**- Memory -**

shared\_buffers = 16GB# 设置共享内存缓存区大小，一般设置为总内存1/4。

huge\_pages = on# 是否为主共享内存区域请求大页，当为try时，未开启大页或分配大页内存太小，数据库仍能启动，但不使用大页；当为on时不能分配足够内存时，启动失败；当为off时，不请求大页。

#huge\_page\_size = 0#设置大页的大小。当设置为 0 时，将使用系统上默认的大页大小。

#temp\_buffers = 8MB# 设置会话的临时表的缓存大小，默认为8M，通常保持默认值就可以

#max\_prepared\_transactions = 0#设置可以同时处于“准备”状态的最大事务数，默认为0，即禁用准备事务功能。

work\_mem= 8MB#为每个进程单独分配的内存，主要用于排序、HASH等操作，若这部分内存小了，数据将写入临时磁盘文件。此外，可能会有多个会话正在同时执行此类操作。因此，使用的总内存可能是 work\_mem 值的许多倍；设置此值时需慎重。

#hash\_mem\_multiplier = 1.0#用于计算基于哈希的操作可以使用的最大内存量。它是 work\_mem 乘以 hash\_mem\_multiplier。

maintenance\_work\_mem = 1GB#指定维护操作使用的最大内存量，例如 VACUUM、CREATE INDEX 和 ALTER TABLE ADD FOREIGN KEY。较大的设置可能会提高清理和还原数据库转储的性能。

#autovacuum\_work\_mem = -1# 指定每个自动清理工作进程使用的最大内存量，默认-1 表示使用maintenance\_work\_mem 的值

#logical\_decoding\_work\_mem = 64MB#指定在将某些已解码更改写入本地磁盘之前，逻辑解码要使用的最大内存量。

#max\_stack\_depth = 2MB#指定服务器执行堆栈的最大安全深度，执行函数时使用。设置的太小而无法执行复杂的功能，设置的太大容易使单个后端进程崩溃。

#shared\_memory\_type = mmap#指定服务器用于保存 PostgreSQL 的共享缓冲区和其他共享数据的主共享内存区域的共享内存实现的方法

# mmap 是一种内存的映射文件的方法，好处是不再需要配置内核参数"kernel.shmmax"和"kernel.shmall"。

# sysv是用于通过shmget分配的System V共享内存

# windows是用于windows共享内存

dynamic\_shared\_memory\_type = posix#指定服务器使用的动态共享内存的实现方式

# posix(使用shm\_open分配的posix共享内存)

# sysv(通过shmget分配的System V共享内存)

# windows(用于windows共享内存)

# mmap(用于使用存储在数据目录中的内存映射文件模拟共享内存)，因为操作系统可能会反复将修改后的页面写回磁盘，从而增加系统 I/O 负载，通常不鼓励使用。

#min\_dynamic\_shared\_memory = 0MB#指定在服务器启动时分配的以供并行查询使用的内存量。默认值为 0（无）。

**- Disk -**

#temp\_file\_limit = -1#指定进程可用于临时文件（例如排序和散列临时文件）的存储文件的最大磁盘空间量。超过限制的操作将被取消。 -1（默认）表示没有限制。

**- Kernel Resources -**

#max\_files\_per\_process = 1000#设置每个服务器子进程允许同时打开的最大文件数。

**- Cost-Based Vacuum Delay -**

#vacuum\_cost\_delay = 0#为防止Vacuum对系统产生太大冲击， 每完成一定的工作量， 就休眠多少毫秒后再重新开始工作。默认值为0，表示不延迟清理。

#vacuum\_cost\_page\_hit = 1# VACUUM访问的数据块在共享内存中的代价值。

#vacuum\_cost\_page\_miss = 2# VACUUM访问的数据块不在共享内存中的代价值

#vacuum\_cost\_page\_dirty = 20# VACUUM改变一个非脏数据块为脏数据块的代价值，如果页面被清理弄脏了，则计数为20。因为把脏块刷出到磁盘需要I/O。

#vacuum\_cost\_limit = 200#限制一次完成vacuum的累计成本，每次清理工作达到200后，它就会休眠多少毫秒

**- Background Writer -**

#bgwriter\_delay = 200ms#表示每次执行bgwriter的间隔时间

#bgwriter\_lru\_maxpages = 100#每次执行bgwriter，刷新缓存的最大页面数量。

#bgwriter\_lru\_multiplier = 2.0#预估缓存数目的因子，采用计算的方式来决定多少缓存应该被bgwriter刷到磁盘。该值必须小于bgwriter\_lru\_maxpages。写“脏”缓存区到磁盘的数目取决于服务器最近几次使用的缓存区数目。最近的buffers数目的平均值乘以bgwriter\_lru\_multiplier以评估下次服务器进程需要的buffers数目。

#bgwriter\_flush\_after = 0#只要后台写入的数据超过这个数量，尝试强制 OS 把这些写发送到底层存储上。 0表示禁用这个功能， Linux 上的默认值是512kB。

**- Asynchronous Behavior -**

#backend\_flush\_after = 0#只要一个后端写入的数据量超过这个数量时，就会尝试强制 OS 把这些写发送到底层存储。

effective\_io\_concurrency = 2|200 #设置PostgreSQL可以同时被执行的并发磁盘 I/O 操作的数量。调高这个值，可以增加任何单个PostgreSQL会话试图并行发起的 I/O 操作的数目。允许的范围是 1 到 1000，或 0 表示禁用异步 I/O 请求。当前这个设置仅影响位图堆扫描。

#maintenance\_io\_concurrency = 10#与effective\_io\_concurrency相似，但用于支持许多客户端会话完成的维护工作。

max\_worker\_processes = 16#设置整个数据库实例层面允许支持的最大后台工作进程数。在备库上，必须把这个参数设置为等于或者高于主库上的值。否则， 备库可能不会允许查询。

max\_parallel\_workers\_per\_gather = 4#设置某个并行操作允许并行度。如果设置为0表示关闭并行查询。此参数比max\_parallel\_workers高则无效。 一般设置为max\_worker\_processes>=max\_parallel\_workers>=max\_parallel\_workers\_per\_gather| max\_parallel\_maintenance\_workers 。

max\_parallel\_maintenance\_workers = 4#设置某个维护操作能够运行的最大并行度。

max\_parallel\_workers = 16#设置整个数据库实例层面允许用做并行的后台工作进程数。若此参数设置为比max\_worker\_processes高则无效。

#parallel\_leader\_participation = on# PG并行计算框架中，并行任务由计算进程执行，计算进程执行的结果，由GATHER（leader process）收集再转交下一个节点。这个参数表示是否允许leader process参与计算任务，而不是空等所有计算进程。

换句话说，就像领导和小弟一起干一样的活，所有人活都干完后，领导依旧需要把结果汇总进行下一个环节。

1、parallel\_leader\_participation设置为ON，表示领导和小弟一起干活，再将结果汇总进行下一步。

2、parallel\_leader\_participation设置为OFF，表示领导不干小弟的活，而是等所有小弟干完，再将结果汇总进行下一步。

#old\_snapshot\_threshold = -1#设置在使用快照时，一个快照可以被使用而没有发生“snapshot too old” 错误的风险的最小时间。超过此阈值时间的dead tuple将允许被清除。 -1表示禁用此功能，实际上将快照的时限设置为无穷大。

**WRITE-AHEAD LOG**

**- Settings -**

wal\_level = replica#设置wal类型，取值范围：minimal、replica、logical，此配置项决定了多少信息写入WAL日志中。

minimal：只写入在数据库崩溃或突然关机后进行恢复时所需要的信息。

replica：会添加一些备库只读查询时需要的信息

logical：逻辑同步

fsync = on#是否使用fsync()系统调用把文件系统中的脏页刷新到物理磁盘，确保数据库能在操作系统或者硬件崩溃的情况下恢复到一致的状态。

synchronous\_commit = on# 提交一个事务是否需要等待把WAL日志写入磁盘后才返回，取值范围：off, local, remote\_write, remote\_apply, or on。

wal\_sync\_method = fdatasync#设置将wal buffer强制更新到WAL日志文件的方法，有以下几种：

open\_datasync： 使用O\_DSYNC选项的open()函数打开WAL日志。

fdatasync： 每次提交时调用fdatasync函数。

fsync\_writethrough： 每次提交时调用fsync()函数， 同时把所有Cache都刷新到物理硬盘中。

fsync： 每次提交时调用fsync()函数。

open\_sync： 使用O\_SYNC选项的open()函数来打开WAL日志。

可以用 pg\_test\_fsync 来测试哪种方式更快。

full\_page\_writes = on#当PostgreSQL在检查点之后第一次修改一个页面时，整个页面都被写入wal，为了防止块折断，从而保证数据的安全性。

wal\_log\_hints = off#设置这个参数使得能够记录特定提示位（hint-bit）的变化。一些特定的工具(pg\_rewind)会要求设置这个参数来正常的运行，所以将它设置为ON就好。

#wal\_compression = on #是否启用允许压缩

#wal\_init\_zero = on #新创建的 WAL 文件被零填充

#wal\_recycle = on #重用(重命名) WAL 文件，从而避免创建新文件。

#wal\_buffers = -1# 设置wal buffer大小，-1 表示等于shared\_buffers的 1/32 的尺寸（大约3%），但是不小于64kB也不大于 WAL 段的尺寸（通常为16MB）。

#wal\_writer\_delay = 200ms#指定wal writer process把WAL日志写入磁盘的周期。 在每个周期中会把缓存中的WAL日志刷新到磁盘上， 休眠wal\_writer\_delay时间， 然后重复上述过程。当synchronous\_commit为off时有效。

#wal\_writer\_flush\_after = 1MB# wal 写入器刷写wal 的大小。wal\_writer 的字节数超过该阈值时触发fsync。Wal\_writer 的字节数小于该阈值，则WAL将只被写入到操作系统的缓存而不是被刷到磁盘。

#wal\_skip\_threshold = 2MB#当wal\_level设置为minimal，在创建表或向表中插入大量数据时，如果超过wal\_skip\_threshold的设置(默认是2M),则跳过写入WAL日志。这样可以防止事务处理缓慢，而达到无法忍受的程度。

#commit\_delay = 0#指定向WAL缓冲区写入记录和将缓冲区刷新到磁盘上之间的时间延迟， 非零的设置值允许多个事务共用一个fsync()系统调用刷新数据。但是如果没有其他事务准备提交， 那么该延迟就是在浪费时间。 因此， 该延迟只在一个服务器进程写其提交日志时， 且至少commit\_siblings个其他事务处于活跃状态的情况下执行。 默认是“0”（ 无延迟） 。

#commit\_siblings = 5#在执行commit\_delay延迟时要求同时打开的最小并发事务数。 默认是“5”。

**- Checkpoints -**

checkpoint\_timeout = 10min#设置Checkpoint发生的间隔时间

checkpoint\_completion\_target = 0.9# 设置完成checkpoint的时间，0.9表示PostgreSQL在两个检查点间隔时间的0.9倍时间内完成所有脏页的刷新。

#checkpoint\_flush\_after = 0#当执行检查点时，超过这个数量会强制写到底层存储，否则，这些页面可能会保存在操作系统的页面缓存中，0表示禁用这个功能。

#checkpoint\_warning = 30s#设置这个对检查点参数进行简单的检查。 如果检查点发生的时间间隔接近checkpoint\_warning秒， 就会在服务器日志中输出一条消息， 这样， 通过监控日志信息， 就可以在检查点发生得过于频繁时通知用户减小该频率。

max\_wal\_size = 8GB #指定WAL日志的空间上限。 保证WAL不会占用太多的空间。

min\_wal\_size = 2GB #指定WAL日志空间下限。该值通常不要设置得太小， 容易导致Standby失效。

**- Archiving -**

archive\_mode = on# 是否开启归档，当 wal\_level 设置为 minimum 时，无法启用 archive\_mode。

#archive\_command = '/usr/bin/true'#执行归档命令，这里虽然设置了，但设置的是一个无用的命令， 主要是因为修改archive\_mode需要重启数据库服务器， 而修改archive\_command不需要重启数据库服务器。

archive\_command = 'cp %p /pgdata/archivedir/%f'

%p表示在线WAL日志文件的全路径名， “%f”表示不包括路径的WAL日志文件名。

archive\_timeout = 15min#指定多少秒数内强制数据库切换一个WAL日志文件。如果主库在某段时间内比较闲， 可能会很长时间才产生WAL日志文件， 这会导致主库和Standby库之间有较大的延迟， 这时可以配置此参数。

**- Archive Recovery -**

#restore\_command = ‘’#指定如何获得WAL日志文件， 通常是配置一个拷贝命令， 从备份目录中把WAL日志文件拷贝过来。

#archive\_cleanup\_command = ''#清理不需要的WAL日志文件。

#recovery\_end\_command = ''#恢复完成后， 可以执行一个命令。

**# - Recovery Target -**

#recovery\_target = ‘’#目前此参数只能配置为空或“immediate”， 配置为“immediate”， 则数据库恢复到一个一致性的点时就立即停止恢复。

#recovery\_target\_name = ‘’#在备份之前可以创建一个恢复点， 在恢复的时候可以恢复到这个恢复点， 此参数用来指定该恢复点的名称。

#recovery\_target\_time = ‘’#用于指定恢复到哪个时间点。 恢复到设定时间点之前最近的一致点还是该时间点之后最近的一致点是由后面的参数“recovery\_target\_inclusive”来指定的。

#recovery\_target\_xid = ‘’#指定恢复到哪个指定的事务。

#recovery\_target\_lsn = ‘’#指定恢复到哪个指定的预写日志位置的LSN。

#recovery\_target\_inclusive = on#指定恢复到恢复目标（recovery target） 之后还是之前。 默认为恢复目标之后， 即值为“true”。

#recovery\_target\_timeline = ‘latest’#指定恢复的时间线。 默认只恢复到当前的时间线， 而不会切换到新的时间线。

#recovery\_target\_action = ‘pause’#指定达到恢复目标后，服务器应采取的操作。

**REPLICATION**

**# - Sending Servers -**

#max\_wal\_senders = 10#指定来自备用服务器或流式基本备份客户端的最大并发连接数（同时运行的WAL发送器进程的最大数） 。

#max\_replication\_slots = 10#指定服务器可以支持的最大复制插槽数。 #wal\_keep\_size = 0 #指定保存在 pg\_wal 目录中的过去日志文件段的最小大小，以放在wal日志被清理。

#max\_slot\_wal\_keep\_size = -1#指定复制槽允许保留在 pg\_wal 目录中的 WAL 文件的最大大小。 -1（默认值），表示复制槽可以保留无限量的 WAL 文件。

#wal\_sender\_timeout = 60s#不活动的复制连接的时间超过这个参数指定的毫秒数， 就会被终止掉。

#track\_commit\_timestamp = off#记录事务的提交时间。

**# - Primary Server -**

#synchronous\_standby\_names = ‘’#指定同步复制Standby的名称，各个名称用逗号分隔，而Standby名称是在Standby的连接参数“application\_name”指定的。 ‘\*’ = all

#vacuum\_defer\_cleanup\_age = 0#让主库延迟清理。0表示可以尽快删除死行版本

**- Standby Servers -**

#primary\_conninfo = ''#在流复制中， 指定如何连接主库， 是一个标准的libpq连接串。

#primary\_slot\_name = ''#指定复制槽（Replication Slot） 。

#promote\_trigger\_file = ‘’#指定激活Standby的触发文件。 Standby数据库发现存在此文件时， 就会把Standby激活为主库。

#hot\_standby = on# on， 说明备库是“Hot Standby”，可以查询； off， 说明备库是“Warm Standby”， psql是无法连接这个备库的。

#max\_standby\_archive\_delay = 30s#备库从WAL归档中读取时的最大延迟。 默认为30秒，如果设置为-1， 则会一直等待。

#max\_standby\_streaming\_delay = 30s#备库从流复制中读取WAL时的最大延迟。 默认为30秒， 如果设置为-1， 则会一直等待。

#wal\_receiver\_create\_temp\_slot = off#指定当没有配置要使用的永久复制槽时（使用 primary\_slot\_name），WAL 接收器进程是否应在远程实例上创建临时复制槽。

#wal\_receiver\_status\_interval = 10s#指定在备库上的 WAL接收者进程向主库发送有关复制进度的信息的最小周期；

#hot\_standby\_feedback = off#指定将在备库上执行的查询的反馈发送到主库。这样主库上的AutoVacuum就不会清理掉这些数据， 就能大大减少查询冲突的发生。

#wal\_receiver\_timeout = 60s#终止超过此时间不活动的复制连接。

#wal\_retrieve\_retry\_interval = 5s#指定当 WAL 数据不能从任何源（流复制、本地 pg\_wal 或 WAL 归档）获得时，备用服务器应等待多长时间，然后再尝试检索 WAL 数据。

#recovery\_min\_apply\_delay = 0 #指定让Standby落后主库一段时间。

**- Subscribers -**

#max\_logical\_replication\_workers = 4#设置逻辑复制的最大进程数，它是从 max\_worker\_processes 定义的池中获取的。 #max\_sync\_workers\_per\_subscription = 2#每个订阅的最大同步线程数。是从 max\_logical\_replication\_workers 定义的池中获取的。

**QUERY TUNING**

**- Planner Method Configuration –**

#enable\_async\_append = on #是否允许查询使用异步感知追加

#enable\_bitmapscan = on #是否允许查询使用位图扫描

#enable\_gathermerge = on #是否允许查询使用收集合并

#enable\_hashagg = on #是否允许查询使用hash聚合

#enable\_hashjoin = on #是否允许查询使用hash连接

#enable\_incremental\_sort = on #是否允许查询增量排序

#enable\_indexscan = on #是否允许查询使用索引扫描

#enable\_indexonlyscan = on #是否允许查询使用index-only扫描

#enable\_material = on #是否允许查询使用物化

#enable\_memoize = on #是否允许查询使用memoize来缓存嵌套循环连接内参数化扫描的结果。

#enable\_mergejoin = on #是否允许查询使用合并排序连接

#enable\_nestloop = on #是否允许查询使用嵌套循环

#enable\_parallel\_append = on #是否允许查询使用并行感知追加

#enable\_parallel\_hash = on #是否允许查询使用并行散列连接

#enable\_partition\_pruning = on #是否运行从查询计划中消除分区表的功能。

#enable\_partitionwise\_join = off #是否允许对分区进行联接

#enable\_partitionwise\_aggregate = off #是否允许对分区分组或聚合

#enable\_seqscan = on #是否允许查询使用顺序扫描

#enable\_sort = on #是否允许查询使用显示排序

#enable\_tidscan = on #是否允许查询使用TID 扫描

**- Planner Cost Constants -**

#seq\_page\_cost = 1.0 # 一次顺序访问一个数据块页面的开销

random\_page\_cost = 4.0|1.1 #一次随机访问一个数据块页面的开销

#cpu\_tuple\_cost = 0.01 #计算处理一条数据行的开销

#cpu\_index\_tuple\_cost = 0.005 #计算处理一条索引行的开销

#cpu\_operator\_cost = 0.0025 #执行一个操作符或函数的开销

#parallel\_setup\_cost = 1000.0 #启动并行进程的成本

#parallel\_tuple\_cost = 0.1 #通过并行进程处理一行数据的成本

#min\_parallel\_table\_scan\_size = 8MB #设置开启并行的条件之一，表占用空间小于此值将不会开启并行

#min\_parallel\_index\_scan\_size = 512kB #设置开启并行的条件之一，并行扫描索引数据块小于此值不会开启并行。

effective\_cache\_size = 48GB #在一次索引扫描中可用的磁盘缓冲区的有效大小。这个因素会被用到使用索引的成本考虑中：值越大，使用索引扫描的可能性就越大；值越小，使用顺序扫描的可能性就越大。

#jit\_above\_cost = 100000 #表示超过多少cost 的查询才会使用JIT 功能

#jit\_inline\_above\_cost = 500000 #表示超过多少cost 的查询使用JIT 的inline 功能。

#jit\_optimize\_above\_cost = 500000 #表示超过多少cost 的查询使用JIT 的optimization 功能。

**- Genetic Query Optimizer -**

#geqo = on #是否启用基因查询优化

#geqo\_threshold = 12 #只有当涉及的FROM关系数量至少有geqo\_threshold，才会使用基因查询优化

#geqo\_effort = 5 #控制geqo中规划时间和查询规划的有效性之间的平衡，较大的值会增加执行查询计划所花费的时间，但也会增加选择有效查询计划的可能性。

#geqo\_pool\_size = 0 #控制geqo使用池的大小，即遗传种群中的个体数量。

#geqo\_generations = 0 #控制geqo使用的代数，即算法的迭代次数。

#geqo\_selection\_bias = 2.0 #控制geqo使用的选择性偏好，选择偏好是群体内的选择压力。

#geqo\_seed = 0.0 #控制geqo使用的随机数产生器的初始值

**- Other Planner Options -**

#default\_statistics\_target = 100 #为没有通过 ALTER TABLE SET STATISTICS 设置特定列目标的表列设置默认统计目标。较大的值会增加执行 ANALYZE 所需的时间，但可能会提高规划者估计的质量。

#constraint\_exclusion = partition #用于控制排除约束，取值范围：on、off 或partition，默认为partition(on)，则表示启用排除约束；如果为off，则表示不启用。当启用了排除约束，服务器就会检查定义给每个分区的约束来决定分区是否能满足查询。

#cursor\_tuple\_fraction = 0.1 # 使用这个参数值来估算需要哪部分由查询返回的行。0.1，表示结果集中前 10% 的行将很快返回，而其余部分的行则需要更长时间才能返回。

#from\_collapse\_limit = 8 #如果生成的 FROM 列表不超过这么多项目，则计划器会将子查询合并到上层查询中。较小的值会减少计划时间，但可能会产生较差的查询计划。

#jit = on #是否启用JIT 编译

#join\_collapse\_limit = 8 #如果得出的列表中不超过配置值，那么规划器将把显式JOIN（除了FULL JOIN）结构重写到 FROM项列表中。较小的值可减少规划时间，但是可能会生成差些的查询计划；

#plan\_cache\_mode = auto #指定执行计划的缓存模式，允许的值为 auto（默认值）、force\_custom\_plan 和 force\_generic\_plan。

**REPORTING AND LOGGING**

**- Where to Log -**

log\_destination = 'csvlog' #设置记录消息的方法，包括 stderr、csvlog 、syslog和eventlog，默认为stderr

logging\_collector = on # 是否启用日志收集器，日志输出方式为stderr或csvlog时，需要启动这项，主进程会启动Logger辅助进程。

#log\_directory = ‘log‘ #设置日志目录，可以设置为绝对路径，也可以设置为相对路径(pgdata)

#log\_filename = 'postgresql-%Y-%m-%d\_%H%M%S.log‘ #日志文件命名方式

#log\_file\_mode = 0640 #日志文件的创建模式

#log\_rotation\_age = 1d #每天生成一条新的日志

#log\_rotation\_size = 10MB #日志写满10MB，则切换一个日志

#log\_truncate\_on\_rotation = off #日志是否覆盖，只有基于时间的切换的日志才会发生截断。

示例： 保留一个星期的日志， 进行循环覆盖。配置方法如下：

log\_filename = ‘postgresql-%u.log‘ #%u代表星期几，和 %a 表示相同

log\_truncate\_on\_rotation = on

log\_rotation\_age = 1d

#syslog\_facility = ‘LOCAL0‘ #当启用了向syslog记录时，这个参数决定要使用的syslog“设备”，可以选择LOCAL0、LOCAL1、LOCAL2、LOCAL3、LOCAL4、LOCAL5、LOCAL6、LOCAL7。

#syslog\_ident = 'postgres‘ #启用日志记录到 syslog 时，此参数确定用于标识 syslog 日志中的 PostgreSQL 消息的程序名称。

#syslog\_sequence\_numbers = on #设置日志前缀

#syslog\_split\_messages = on #设置日志分割

#event\_source = ‘PostgreSQL‘ #确定用于标识日志中的 PostgreSQL 消息的程序名称。

**- When to Log -**

#log\_min\_messages = warning #设置日志级别，每个级别包括其后的所有级别，级别越高，发送到日志的消息就越少。消息级别包括：debug5、debug4、debug3、debug2、debug1、info、notice、warning、error、log、fatal、panic

#log\_min\_error\_statement = error #设置错误SQL 级别

log\_min\_duration\_statement = 200ms # 设置记录慢查询的阈值，例如，如果设置为 200 毫秒，那么所有运行 200 毫秒或更长时间的 SQL 语句都将被记录。

#log\_min\_duration\_sample = -1 #执行时间超出 此参数值 的慢查询将被抽样到日志中，采样率由 log\_statement\_sample\_rate 控制。例如，如果设置为 100 毫秒，那么所有运行 100 毫秒或更长的 SQL 语句都将被考虑进行采样。 -1表示禁用采样功能。

#log\_statement\_sample\_rate = 1.0 # 设置采用因子，采样是随机的，例如 0.5 意味着有二分之一的机会记录任何给定的语句。 1.0，表示记录所有采样语句。

#log\_transaction\_sample\_rate = 0.0 #以事务为视角的采样率，它对于事务采样帮助比较大。

**- What to Log -**

#debug\_print\_parse = off #

#debug\_print\_rewritten = off

#debug\_print\_plan = off

#这些参数可以发出各种调试输出。设置后，它们会打印解析树、查询重写信息或执行计划。

#debug\_pretty\_print = on #格式化输出日志

log\_autovacuum\_min\_duration = 0#autovacuum 执行多久，则记录该操作。例如，设置为 200 毫秒，那么所有运行 200 毫秒或更长时间的autovacuum操作都会被记录下来。-1表示不记录。

log\_checkpoints = on #是否记录checkpoint操作

#log\_connections = off #是否记录用户的登陆信息

#log\_disconnections = off #是否记录用户的断开的登陆的信息

#log\_duration = off #是否记录每条sql的执行时间

#log\_error\_verbosity = default #控制每条要记录的日志的详细信息量。有效值为 TERSE、DEFAULT 和 VERBOSE，默认default。 TERSE 不包括 DETAIL、HINT、QUERY 和 CONTEXT 错误信息的记录。 VERBOSE 输出包括 SQLSTATE 错误代码以及生成错误的源代码文件名、函数名和行号。

#log\_hostname = off #默认记录信息中带有客户端的IP地址，不带有对方的机器名

#log\_line\_prefix = ‘%m [%p] ’ #对日志的打印的格式和信息的设置，默认值为 '%m [%p] '，它记录时间戳和进程 ID。

#log\_lock\_waits = off #当一个会话为获得一个锁等到超过deadlock\_timeout时，是否记录日志。

#log\_recovery\_conflict\_waits = off # 当备库恢复冲突等待的时间超过deadlock\_timeout时，是否记录日志。 #log\_parameter\_max\_length = -1 #确定非错误statement的绑定参数的可打印长度， -1允许完整记录绑定参数，0表示不记录。

#log\_parameter\_max\_length\_on\_error = 0 #确定错误statement的绑定参数的可打印长度。

log\_statement = 'ddl' #控制记录哪些 SQL 语句。有效值为 none (off)、ddl、mod（ 所有 ddl 语句和数据修改语句） 和 all（所有语句）。

#log\_replication\_commands = off #是否记录复制命令

#log\_temp\_files = -1 #控制临时文件名和大小的记录。如果通过此设置启用，则在删除每个临时文件时都会发出一个日志条目。0值记录所有临时文件信息，正值仅记录大小大于或等于指定数据量的文件。 -1，即禁用此类日志记录。

log\_timezone = 'Asia/Shanghai' #设置用于写入服务器日志的时间戳的时区。

**PROCESS TITLE**

#cluster\_name = ‘’ #为所有服务器进程设置进程标题

#update\_process\_title = off #每次服务器接收到新的 SQL 命令时启用更新进程标题。

**STATISTICS**

**- Query and Index Statistics Collector -**

#track\_activities = on #是否收集当前正在执行的SQL。

track\_activity\_query\_size = 1024 # 指定为每个活动会话跟踪当前正在执行的sql的字节数，用于pg\_stat\_activity.query字段。

#track\_counts = on #是否收集表和索引上的统计信息，

#track\_io\_timing = off #是否收集I/O的时间信息。

#track\_wal\_io\_timing = off #启用 WAL I/O 调用的计时。

#track\_functions = none #启用对函数调用计数和使用时间的跟踪。 pl 仅跟踪过程语言函数，all 跟踪 SQL 和 C 语言函数，none，即禁用函数统计跟踪。

#stats\_temp\_directory = 'pg\_stat\_tmp' #设置存储临时统计数据的目录。

**- Monitoring -**

#compute\_query\_id = auto #开启或关闭query\_id的计算。 off（总是禁用）、on（总是启用）= auto

#log\_statement\_stats = off # 这个参数设置为on则会开启log\_parser\_stats，log\_planner\_stats，log\_executor\_stats这三个选项。

#log\_parser\_stats = off

#log\_planner\_stats = off

#log\_executor\_stats = off

**AUTOVACUUM**

autovacuum = on #是否开启 autovacuum 守护进程，还必须启用 track\_counts 才能使 autovacuum 工作。

#autovacuum\_max\_workers = 3 #指定自动清理的最大worker数量

#autovacuum\_naptime = 1min #设置两次系统autovacuum操作之间的间隔时间。

#autovacuum\_vacuum\_threshold = 50 #当表上发生变化的行数至少达到此参数值时， 才可能让AutoVacuum对其进行VACUUM，还有另一个参数“autovacuum\_vacuum\_scale\_factor”同时控制VACUUM的执行条件。触发autovacuum的计算公式：autovacuum\_vacuum\_scale\_factor×表上记录数+autovacuum\_vacuum\_threshold。

#autovacuum\_vacuum\_insert\_threshold = 1000 #当表上插入的行数至少达到此参数值时， 才可能让AutoVacuum对其进行VACUUM，还有另一个参数“autovacuum\_vacuum\_insert\_scale\_factor”同时控制VACUUM的执行条件。

#autovacuum\_analyze\_threshold = 50 #当表上发生变化的行数达到此参数值时， 才可能让AutoVacuum对其进行ANALYZE， 还有另一个参数“autovacuum\_analyze\_scale\_factor”同时控制VACUUM的执行条件。

#autovacuum\_vacuum\_scale\_factor = 0.2 #触发VACUUM的第二个阈值条件。

#autovacuum\_vacuum\_insert\_scale\_factor = 0.2 #触发VACUUM的第二个阈值条件。 #autovacuum\_analyze\_scale\_factor = 0.1 #触发ANALYZE的第二个阈值条件。

#autovacuum\_freeze\_max\_age = 200000000 #设置需要强制对数据库进行清理的XID上限值，以防止事务回卷。

#autovacuum\_multixact\_freeze\_max\_age = 400000000 #设置需要强制对数据库进行清理的多事务ID上限值

#autovacuum\_vacuum\_cost\_delay = 2ms #为防止AutoVacuum对系统产生太大冲击， AutoVacuum每完成一定的工作量， 就休眠2ms再重新开始工作。

#autovacuum\_vacuum\_cost\_limit = -1 #指定autovacuum操作的代价限制值， -1表示将使用常规的Vacuum\_cost\_limit 值。

**CLIENT CONNECTION DEFAULTS**

**- Statement Behavior -**

#client\_min\_messages = notice # 设置客户端的消息级别，包括：debug5、debug4、debug3、debug2、debug1、log、notice、warning、error

#search\_path = ‘“$user”, public’ #设置搜索对象的模式顺序

#row\_security = on #是否启用行安全策略

#default\_table\_access\_method = ‘heap’ #控制表的访问方式，即创建新表时默认要使用哪个存储引擎。通常我们是堆表。

#default\_tablespace = ‘’ #设置默认表空间

#default\_toast\_compression = 'pglz' # 设置默认的 TOAST 压缩方法。 'pglz' or 'lz4'

#temp\_tablespaces = ‘’ #指定要在其中创建临时对象（临时表和临时表上的索引）的表空间。空字符串表示在当前数据库的默认表空间中创建所有临时对象。

#check\_function\_bodies = on #是否开启对函数体字符串的验证

#default\_transaction\_isolation = ‘read committed’ #设置默认的事务隔离级别

#default\_transaction\_read\_only = off #控制每个新事务的默认只读状态。

#default\_transaction\_deferrable = off #控制每个新事务的默认可延迟状态。

#session\_replication\_role = 'origin' #控制在当前会话，是否将replication相关的trigger和rule失效。可能的值是 origin（默认值）、replica 和 local。

#statement\_timeout = 0 #设置sql执行超时时间

#lock\_timeout = 0 #设置锁超时时间

#idle\_in\_transaction\_session\_timeout = 0 #设置空闲事务超时时间

#idle\_session\_timeout = 0 #设置空闲会话超时时间

#vacuum\_freeze\_table\_age = 150000000 #如果表事务ID的年龄大于此设置指定的年龄，将触发freeze。

#vacuum\_freeze\_min\_age = 50000000 #指定触发vacuum freeze的事务id的最小年龄。

#vacuum\_failsafe\_age = 1600000000 #指定在 VACUUM 采取特殊措施以避免事务回卷之前，表的 pg\_class.relfrozenxid 字段可以达到的最大年龄（在事务中）。这是 VACUUM 的最后手段。

#vacuum\_multixact\_freeze\_table\_age = 150000000 #如果表的 pg\_class.relminmxid 字段已达到此设置指定的年龄，会执行VACUUM将触发FREEZE。

#vacuum\_multixact\_freeze\_min\_age = 5000000 #指定VACUUM在扫描表时用来决定是否把组合事务 ID 替换为一个更新的事务 ID或组合事务 ID 的切断年龄（以组合事务计）。

#vacuum\_multixact\_failsafe\_age = 1600000000 #指定表的 pg\_class.relminmxid 字段在 VACUUM 采取特殊措施以避免事务回卷之前可以达到的最大年龄。这是 VACUUM 的最后手段。

#bytea\_output = 'hex‘ # hex, escape #设置 bytea 类型值的输出格式。

#xmlbinary = 'base64' #设置二进制值在 XML 中的编码方式。

#xmloption = 'content' #设置在 XML 和字符串值之间转换时 DOCUMENT 或 CONTENT 是否是隐式的。

#gin\_pending\_list\_limit = 4MB #设置当GIN索引启用fastupdate时，pending list容量的最大值。当pending list的容量大于设置值时，会把pending list中数据批量移动到GIN索引数据结构中以进行清理。

**- Locale and Formatting -**

datestyle = 'iso, ymd' #设置日期的显示格式

#intervalstyle = 'postgres' #设置间隔值的显示格式。

timezone = 'Asia/Shanghai' #设置时区。

#timezone\_abbreviations = ‘Default’ #设置服务器将接受的用于日期时间输入的时区缩写集合，默认为“Default”，这是一个适用于世界大部分地区的集合。还有“Australia”和“India”，可以为特定装置定义其他系列。

#extra\_float\_digits = 1 #调整用于浮点值文本输出的位数

#client\_encoding = sql\_ascii #设置客户端编码（字符集）。默认是使用数据库编码。

lc\_messages = 'Chinese (Simplified)\_China.936' #设置显示消息的语言。

lc\_monetary = 'Chinese (Simplified)\_China.936' #设置用于格式化货币金额的语言环境

lc\_numeric = 'Chinese (Simplified)\_China.936' #设置用于格式化数字的语言环境

lc\_time = 'Chinese (Simplified)\_China.936' # 设置用于格式化日期和时间的语言环境

default\_text\_search\_config = ‘pg\_catalog.simple’ #设置默认文本搜索配置

**- Shared Library Preloading -**

#local\_preload\_libraries = '' #指定一个或者多个要在连接开始时预加载的共享库。

#session\_preload\_libraries = '' #指定一个或者多个要在连接开始时预载入的共享库，只有超级用户可以更改这个设置。

shared\_preload\_libraries = 'pg\_stat\_statements' #指定在服务器启动时预加载一个或多个shared libraries。此参数只能在服务器启动时设置，如果找不到指定的库，服务器将无法启动。

#jit\_provider = ‘llvmjit’ #指定 jit provider library 的名称

**- Other Defaults -**

#dynamic\_library\_path = ‘$libdir’ #指定动态共享库地址

#gin\_fuzzy\_search\_limit = 0 # GIN 索引扫描返回结果行数的上限。 0表示没有限制。

**LOCK MANAGEMENT**

#deadlock\_timeout = 1s #这是进行死锁检测之前在一个锁上等待的总时间。

#max\_locks\_per\_transaction = 64 #控制每个事务能够得到的平均的对象锁的数量。

#max\_pred\_locks\_per\_transaction = 64 #控制每个事务能够得到的平均的predicate-locked的数量。

#max\_pred\_locks\_per\_relation = -2 #控制在将锁提升为覆盖整个 relation 之前，单个 relation 中能有多少 pages 或者 tuples 能被 predicate-locked 。 负值表示：(max\_pred\_locks\_per\_transaction / -max\_pred\_locks\_per\_relation) - 1

#max\_pred\_locks\_per\_page = 2 #控制在将锁提升为覆盖整个page之前，单个页面上的多少tuples能被predicate-locked。

**VERSION AND PLATFORM COMPATIBILITY**

**- Previous PostgreSQL Versions -**

#array\_nulls = on #控制数组输入解析器是否将未引用的 NULL 识别为指定空数组元素。默认是打开的，允许输入包含空值的数组值。

#backslash\_quote = safe\_encoding #是否允许使用反斜线\转义单引号? \’ 。on 允许, off 不允许, safe\_encoding（仅当client\_encoding不允许反斜线\出现在多字节字符中时（大多数字符集\都是单字节表示），那么才允许使用'转义单引号.）

#escape\_string\_warning = on #同时在''中的字符串中包含反斜线时，是否输出警告。

#lo\_compat\_privileges = off #控制是否启动对大对象权限检查的向后兼容模式。 on表示当读取或修改大对象时禁用权限检查。

#quote\_all\_identifiers = off #当数据库生成SQL时，强制引用所有的标识符（包括非关键字）。

#standard\_conforming\_strings = on #控制普通字符串文本（'...'）中是否按照SQL标准把反斜扛当普通文本。

#synchronize\_seqscans = on #允许大型表的顺序扫描彼此同步，在大约相同的时间内并行扫描读取相同的数据块，从而共享 I/O 工作负载。

**- Other Platforms and Clients -**

#transform\_null\_equals = off #控制表达式expr = NULL（或NULL = expr）当做expr IS NULL处理。如果expr得出NULL值则返回真，否则返回假。

**ERROR HANDLING**

#exit\_on\_error = off # 是否遇到任务问题都终止会话。

#restart\_after\_crash = on # 后端crash后是否重新初始化

#data\_sync\_retry = off #控制当fsync到磁盘失败后是否继续运行数据库。由于在某些操作系统的场景下，fsync失败后重试阶段即使再次fsync失败也不会报错，从而导致数据丢失。

#recovery\_init\_sync\_method = fsync #当设置为默认的 fsync 时，PostgreSQL 将在崩溃恢复开始之前同步数据和 WAL 目录中的所有文件。这允许在具有许多数据库文件的系统上进行更快的崩溃恢复。

**CONFIG FILE INCLUDES**

#include\_dir = ‘...’ #加载其他配置文件的目录

#include\_if\_exists = '...' #如果文件存在，加载配置文件

#include = '...' #加载配置文件

参数调优网站：https://pgtune.leopard.in.ua/#/