postgresql数据库是由：连接管理系统（系统控制器）、编译执行系统、存储管理系统、事务系统、系统表 五大部分组成。

①：连接管理系统：接收外部操作对系统的请求，对操作请求进行预处理和分发，起系统逻辑控制作用。

②：编译执行系统：由查询编译器、查询执行器组成，完成操作请求在数据库中的分析处理和转化工作，最终实现物理存储介质中数据的操作。

③：存储管理系统：由索引管理器、内存管理器、外存管理器组成，负责存储和管理物理数据，提供对编译查询系统的支持；

④：事务系统：是由事务管理器、日志管理器、并发控制、锁管理器组成，日志管理器和事务管理器完成对操作请求处理的事务一致性支持，锁管理器 和并发控制提供对并发访问数据的一致性支持；

⑤：系统表：是postgresql数据库的元信息管理中心，包括数据库对象信息和数据库管理控制信息。系统表管理元数据信息，将postgresql数据库的各个模块有机地连接在一起，形成一个高效的数据管理系统。

1、系统表：

数据字典是关系数据库系统管理控制信息的核心，在postgresql数据库系统中，系统表扮演着数据字典的角色。

系统表是postgresql数据库存放结构元数据的地方，它在postgresql中表现为存放有系统信息的普通表或者视图。用户可以删除然后重建这些表、增加列、插入和更新数值，然而由用户去修改系统会导致系统信息的不一致性，进而导致系统控制紊乱。正常情况下不应该由用户手工修改系统表，而是由sql命令关联的系统表操作自动维护系统表信息。

postgresql的每一个数据库中都有自己的一套系统表，其中大多数系统表都是在数据库创建时从模板数据库中拷贝过来的，因此这些系统表里的数据都是与所属数据库相关的。只有少数系统表是所有数据库共享的（比如pg_database），这些系统表里的数据是关于所有数据库的。

由于系统表保存了数据库的所有元数据，所以系统运行时对系统表的访问是非常频繁的。为了提高系统性能，在内存中建立了共享的系统表cache，使用hash函数和hash表提高查询效率。

主要系统表功能：

①：pg_namespace：

pg_namespace用于存储命名空间。命名空间是sql92模式下层的结构：每个名字空间有独立的关系、类型等集合，但并不会相互冲突。postgresql的名字空间层次是：数据库、模式、表、属性。

②：pg_tablespace：

pg_tablespace存储表空间信息，将表放置在不同的表空间有助于实施磁盘文件布局。pg_tablespace在整个数据簇里只有一份，也就是说同一个数据集簇内的所有数据库共享一个pg_tablespace表，而不是每个数据库都有自己的pg_tablespace表。

③：pg_database：

pg_database中存放了当前数据集簇中数据库的信息，它也是一个在整个集簇范围内共享的系统表。该表中每一个元祖就表示集簇中的一个数据库，每一个数据库都被分配一个OID作为唯一标识，并且存储在对应元祖的隐藏属性中。

④：pg_class：

pg_class存储表及表类似结构的数据库对象信息，包含索引、序列、视图、复合数据类型、toast表等。每一个对象都在pg_class中表示为一个元祖，并且每一个对象都会被分配一个OID作为唯一标识，该OID作为该元祖的一个隐藏属性存储。

⑤：pg_type：

pg_type存储数据类型信息。基本数据类型和枚举类型由create type创建，域类型由create domain创建，复合数据类型在表创建时自动创建。

⑥：pg_attribute：

pg_attribute存储表的属性信息，对于数据库中表的每个属性都有一个元祖。

⑦：pg_index：

pg_index存储索引的具体信息。

系统视图如下：

这些系统视图提供了查询系统表和访问数据库内部状态的方法，如下大部分系统视图及其用途：

pg_cursors ---打开的游标

pg_group ---数据库用户的组

pg_indexes ---索引

pg_locks ---当前持有的锁

pg_prepared_statements --预备语句

2、数据集簇

postgresql安装完成后，必须先使用initdb程序初始化磁盘上的数据存储区，即数据集簇，由postgresql管理的用户数据库以及系统数据库总称为数据集簇。在postgresql的实现中，数据库就是磁盘上一些文件的集合，只不过这些文件有特定的文件名、存储位置等，并且有些文件之间会相互关联。默认情况下，postgresql的所有数据读存储在其数据目录里，这个数据目录通常会用环境变量pgdata来引用。

pgdata中还保存有数据集簇的配置文件和其他子目录，各个目录及用途说明如下：

pg_version ： 一个包含postgresql主版本号的文件。

base目录：包含每个数据库目录，数据库目录以数据库的OID编号命名，其中名为1的目录对应模板数据库template1

global目录：包含整个集簇共享的全局表，比如pg_database

pg_clog目录：包含事务提交状态数据的子目录

pg_multixact目录：包含多重事务状态数据的子目录（用于共享的行锁）

pg_stat_tmp目录：包含统计子系统所需临时文件的子目录。

pg_subtrans目录：包含子事务状态数据的子目录。

pg_tblspc目录：包含指向表空间的符合链接的子目录。

pg_twophase目录：包含用于预备事务的状态文件的子目录

pg_xlog目录：包含WAL（预写日志）文件的子目录。

postmaster.opts文件：记录服务器上一次启动时使用的命令行参数。

postmaster.pid文件：一个锁文件，记录着当前的守护进程postmaster的进程号和共享内存段ID，在服务器关闭之后此文件将被删除。

postgresql.conf文件：主要配置文件，除基于主机的访问控制和用户名映射之外的其他用户可设置参数都保存在这个文件中。

pg_hba.conf文件：基于主机的访问控制文件，保存对客户端认证方式的设置信息。

pg_indent.conf文件：用户名映射文件，定义了操作系统系统用户名和postgresql用户名之间的对应关系，这些对应关系会被pg_hba.conf用到。

initdb的主要参数介绍：

-A method 指定本地连接的默认用户认证方式。

-D datadir 数据目录的路径，必须是对当前用户可读写的空目录，也可以使用环境变量pgdata来指定

-E encodinc 指定默认数据库编码方式

-U name 指定数据库超级用户名

-W 指示超级用户设置口令

-d 以调试模式运行，可以打印出很多调试信息

-L 指定输入文件（比如postgres.bki）位置。

3、系统数据库：

在初始化完成之后，会默认创建3个系统数据库：template1，template0和postgres。其中template0和postgres都是在初始化过程中从template1拷贝过来的。

template1和template0 数据库用于创建数据库。postgresql中采用从模板数据库复制的方式来创建新的数据库，在创建数据库的命令中可以用-T选项来指定以哪个数据库为模板来创建新数据库。

postgres用于给初始化用户提供了一个可连接的数据库，就像Linux系统中一个用户的主目录一样。

注意：

这些系统数据库都是可以删除的，但是两个模板数据库在删除之前必须将其在pg_database中元组的datistemplate属性改为false，否则删除时会提示“不能删除一个模板数据库”。

4、postgresql进程结构：

postgresql使用一种专用服务器进程体系结构，其中，最主要的两个进程就是守护进程postmaster和服务进程postgres。从本质上来说，postmaster和postgres都是通过载入postgres程序而形成的进程，只是在运行时所处的分支不同而已。守护进程postmaster负责整个系统的启动和关闭。它监听并接受客户端的连接请求，为其分配服务进程postgres。服务进程postgres接受并执行客户端发送的命令。它在底层模块（如存储、事务管理、索引等）之上调用各个主要功能模块（如编译器、优化器、执行器等），完成客户端的各种数据库操作，并返回执行结果。

4.1、守护进程postmaster

它是一个运行在服务器上的总控进程，负责整个系统的启动和关闭，并且在服务进程出现错误时完成系统的恢复。它管理数据库文件、监听并接受来自客户端的连接请求，并且为客户端连接请求fork一个postgres服务进程，来代表客户端在数据库上执行各种命令。同时postmaster还管理与数据库运行相关的辅助进程。用户可以使用postmaster、postgres或者pg_ctl命令启动postmaster。

postmaster就像一个处理客户端请求的调度中心。当客户端程序需要对数据库进行操作时，首先会发出一个起始消息给postmaster进行请求。postmaster将根据这个起始消息中的信息对客户端的身份进行验证，如果身份验证通过，postmaster就为该客户端新建一个服务进程postgres。随后postmaster将于客户端的交互工作转交给postgres服务进程，由postgres来完成客户端所需要的数据库操作。

postmaster也负责整个系统范围的操作，例如：中断等操作，postmaster本身不进行这些操作，它只是指派一个子进程在适当的时间去处理它们。同时它要在数据库崩溃的时候重启系统。postmaster进程在起始时会建立共享内存和信号库，postmaster及其子进程的通信就通过共享内存和信号来实现。这种多进程设计使得整个系统的稳定性更好，即使某个后台进程崩溃也不会影响系统中其他进程的工作，postmaster只需要重置共享内存即可从单个后台进程的崩溃中恢复。

4.1.1、辅助进程启动：

在postgresql中，守护进程postmaster负责整个系统的启动和关闭。在一个数据库管理系统实例中，除了守护进程postmaster和服务进程postgres外，还包括一些其他后台进程，用于专门负责某项具体的工作。在这些辅助进程出现问题时，postmaster进程重新产生出现问题的辅助进程。在postmaster的创建过程中会首先启动syslogger日志进程，并完成pgstat进程、autovacuum进程的初始化工作，而在postmaster的监听循环中检测辅助进程的状态，并新建或者重新创建这些辅助进程。

①：syslogger辅助进程

postmaster进程调用syslogger_start函数启动syslogger子进程。syslogger是8.0后新加的特性，它通过一个管道从postmaster、所有后台进程及其他的子进程那里收集所有的stderr输出，并将这些输出写入到日志文件中。在postgresql.conf配置文件中设置了日志文件的大小和存在时间，若当前的日志文件达到这些限制时，就会被关闭，并且一个新的日志文件会被创建。

②：辅助进程初始化

syslogger辅助进程启动完成后，postmaster开始对辅助进程pgstat进程、autovacuum进程进行初始化操作，为进程分配必要的资源。

在pgstat进程的初始化过程中，主要完成用于发送和接收统计消息的UDP端口创建和测试，UDP端口的创建过程与前面描述的TCP端口创建流程相同，但是socket端口类型改为sock_dgram。

4.2、辅助进程：

postgresql的各个辅助进程完成特定的功能，支撑postgresql系统运行和管理工作。在postmaster进程中，为每个辅助进程设置了一个全局变量来标识该进程号，分别为sysloggerPID、多个writerPID、walwriterPID、autovacPID、pgarchPID、pgstatPID。当这些变量值为0时，标识相应的进程尚未启动。

4.2.1、syslogger系统日志进程：

日志信息是数据库管理员获取数据库系统运行状态的有效手段。在syslogger的配置选项中可以设置日志文件的大小，syslogger会在日志文件达到指定大小时关闭当前日志文件，产生新的日志文件。在postgresql.conf配置文件可以配置日志操作的相关参数如下：

log_destination：配置日志输出目标，根据不同的运行平台会设置不同的值，Linux下默认为stderr。

logging_collector：是否开启日志收集器，当设置on时启动日志功能，否则系统将不产生系统日志辅助进程。

log_directory：配置日志输出文件夹。

log_filename：配置日志文件名称命名规则。

log_rotation_size：配置日志文件大小，当前日志文件达到这个大小时会被关闭，然后创建一个新的文件来作为当前日志文件。

（当然，postgresql.conf中还提供了其他配置参数，可以根据需要进行设置。）

4.2.2、后台写进程bgwriter

BgWriter进程是把共享内存中的脏页写到磁盘上的进程。它的作用有两个：一是定期把脏数据从内存缓冲区刷出到磁盘中，减少查询时的阻塞；二是PG在定期作检查点时需要把所有脏页写出到磁盘，通过BgWriter预先写出一些脏页，可以减少设置检查点（CheckPoint，数据库恢复技术的一

种）时要进行的IO操作，使系统的IO负载趋向平稳。BgWriter是PostgreSQL 8.0以后新加的特性，它的机制可以通过postgresql.conf文件中以"bgwriter_"开头配置参数来控制：

bgwriter_delay:

backgroud writer进程连续两次flush数据之间的时间的间隔。默认值是200，单位是毫秒。

bgwriter_lru_maxpages：

backgroud writer进程每次写的最多数据量，默认值是100，单位buffers。如果脏数据量小于该数值时，写操作全部由backgroud writer进程完成；反之，大于该值时，大于的部分将有server process进程完成。设置该值为0时表示禁用backgroud writer写进程，完全有server process来完成；配置为-1时表示所有脏数据都由backgroud writer来完成。(这里不包括checkpoint操作)

bgwriter_lru_multiplier：

这个参数表示每次往磁盘写数据块的数量，当然该值必须小于bgwriter_lru_maxpages。设置太小时需要写入的脏数据量大于每次写入的数据量，这样剩余需要写入磁盘的工作需要server process进程来完成，将会降低性能；值配置太大说明写入的脏数据量多于当时所需buffer的数量，方便了后面再次申请buffer工作，同时可能出现IO的浪费。该参数的默认值是2.0。

bgwriter的最大数据量计算方式：

1000/bgwriter_delay*bgwriter_lru_maxpages*8K=最大数据量

bgwriter_flush_after：

数据页大小达到bgwriter_flush_after时触发BgWriter，默认是512KB。

4.2.3、PgArch（归档）进程

类似于Oracle数据库的ARCH归档进程，不同的是ARCH是吧redo log进行归档，PgArch是把WAL日志进行归档。再深入点，WAL日志会被循环使用，也就是说，过去的WAL日志会被新产生的日志覆盖，PgArch进程就是为了在覆盖前把WAL日志备份出来。归档日志的作用是为了数据库能够使用全量备份和备份后产生的归档日志，从而让数据库回到过去的任一时间点。PG从8.X版本开始提供的PITR（Point-In-Time-Recovery）技术，就是运用的归档日志。

PgArch进程通过postgresql.conf文件中的如下参数进行配置：

```
# - Archiving -
#archive_mode = off          # enables archiving; off, on, or always
```

```
                                        # (change requires restart)
#archive_command = ''          # command to use to archive a logfile segment
                                        # placeholders: %p = path of file to archive
                                        #               %f = file name only
                                        # e.g. 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server
#archive_timeout = 0           # force a logfile segment switch after this
                                        # number of seconds; 0 disables

archive_mode：表示是否进行归档操作，可选择为off（关闭）、on（启动）和always（总是开启），默认值
archive_command：由管理员设置的用于归档WAL日志的命令。在用于归档的命令中，预定义变量"%p"用来指代
archive_timeout：表示归档周期，在超过该参数设定的时间时强制切换WAL段，默认值为0（表示禁用该功能
```

4.2.4、PgStat（统计数据收集）进程：

PgStat进程是PostgreSQL数据库的统计信息收集器，用来收集数据库运行期间的统计信息，如表的增删改次数，数据块的个数，索引的变化等等。收集统计信息主要是为了让优化器做出正确的判断，选择最佳的执行计划。postgresql.conf文件中与PgStat进程相关的参数，如下：

```
#------------------------------------------------------------------------------
# RUNTIME STATISTICS
#------------------------------------------------------------------------------
# - Query/Index Statistics Collector -
#track_activities = on
#track_counts = on
#track_io_timing = off
#track_functions = none                 # none, pl, all
#track_activity_query_size = 1024       # (change requires restart)
#stats_temp_directory = 'pg_stat_tmp'
track_activities：表示是否对会话中当前执行的命令开启统计信息收集功能，该参数只对超级用户和会话所
track_counts：表示是否对数据库活动开启统计信息收集功能，由于在AutoVacuum自动清理进程中选择清理
track_io_timing：定时调用数据块I/O，默认是off，因为设置为开启状态会反复的调用数据库时间，这给数
track_functions：表示是否开启函数的调用次数和调用耗时统计。
track_activity_query_size：设置用于跟踪每一个活动会话的当前执行命令的字节数，默认值为1024，只能
stats_temp_directory：统计信息的临时存储路径。路径可以是相对路径或者绝对路径，参数默认为pg_stat
```

4.2.5、AutoVacuum（自动清理）进程

在PG数据库中，对数据进行UPDATE或者DELETE操作后，数据库不会立即删除旧版本的数据，而是标记为删除状态。这是因为PG数据库具有多版本的机制，如果这些旧版本的数据正在被另外的事务打开，那么暂时保留他们是很有必要的。当事务提交后，旧版本的数据已经没有价值了，数据库需要清理垃圾数据腾出空间，而清理工作就是AutoVacuum进程进行的。postgresql.conf文件中与AutoVacuum进程相关的参数有：

```
#------------------------------------------------------------------------------
# AUTOVACUUM PARAMETERS
#------------------------------------------------------------------------------
#autovacuum = on                        # Enable autovacuum subprocess?  'on'
                                        # requires track_counts to also be on.
#log_autovacuum_min_duration = -1       # -1 disables, 0 logs all actions and
                                        # their durations, > 0 logs only
                                        # actions running at least this number
                                        # of milliseconds.
#autovacuum_max_workers = 3             # max number of autovacuum subprocesses
                                        # (change requires restart)
#autovacuum_naptime = 1min              # time between autovacuum runs
#autovacuum_vacuum_threshold = 50       # min number of row updates before
                                        # vacuum
#autovacuum_analyze_threshold = 50      # min number of row updates before
                                        # analyze
#autovacuum_vacuum_scale_factor = 0.2   # fraction of table size before vacuum
#autovacuum_analyze_scale_factor = 0.1  # fraction of table size before analyze
#autovacuum_freeze_max_age = 200000000  # maximum XID age before forced vacuum
                                        # (change requires restart)
#autovacuum_multixact_freeze_max_age = 400000000        # maximum multixact age
                                        # before forced vacuum
                                        # (change requires restart)
#autovacuum_vacuum_cost_delay = 20ms    # default vacuum cost delay for
                                        # autovacuum, in milliseconds;
                                        # -1 means use vacuum_cost_delay
#autovacuum_vacuum_cost_limit = -1      # default vacuum cost limit for
                                        # autovacuum, -1 means use
                                        # vacuum_cost_limit

autovacuum：是否启动系统自动清理功能，默认值为on。
log_autovacuum_min_duration：这个参数用来记录 autovacuum 的执行时间，当 autovaccum 的执行时间
autovacuum_max_workers：设置系统自动清理工作进程的最大数量。
autovacuum_naptime：设置两次系统自动清理操作之间的间隔时间。
autovacuum_vacuum_threshold和autovacuum_analyze_threshold：设置当表上被更新的元组数的阈值超
autovacuum_vacuum_scale_factor和autovacuum_analyze_scale_factor：设置表大小的缩放系数。
autovacuum_freeze_max_age：设置需要强制对数据库进行清理的XID上限值。
```

```
autovacuum_vacuum_cost_delay：当autovacuum进程即将执行时，对 vacuum 执行 cost 进行评估，如果
autovacuum_vacuum_cost_limit：这个值为 autovacuum 进程的评估阀值，默认为 -1，表示使用 "vacuu
```

2.4.6、WalWriter（预写式日志写）进程

预写式日志WAL（Write Ahead Log，也称为Xlog）的中心思想是对数据文件的修改必须是只能发生在这些修改已经记录到日志之后，也就是先写日志后写数据（日志先行）。使用这种机制可以避免数据频繁的写入磁盘，可以减少磁盘I/O。数据库在宕机重启后可以运用这些WAL日志来恢复数据库。postgresql.conf文件中与WalWriter进程相关的参数如下：

```
#------------------------------------------------------------------------------
# WRITE AHEAD LOG
#------------------------------------------------------------------------------
# - Settings -
#wal_level = minimal                    # minimal, replica, or logical
                                        # (change requires restart)
#fsync = on                             # flush data to disk for crash safety
                                        # (turning this off can cause
                                        # unrecoverable data corruption)
#synchronous_commit = on                # synchronization level;
                                        # off, local, remote_write, remote_apply, or on
#wal_sync_method = fsync                # the default is the first option
                                        # supported by the operating system:
                                        #   open_datasync
                                        #   fdatasync (default on Linux)
                                        #   fsync
                                        #   fsync_writethrough
                                        #   open_sync
#full_page_writes = on                  # recover from partial page writes
#wal_compression = off                  # enable compression of full-page writes
#wal_log_hints = off                    # also do full page writes of non-critical updates
                                        # (change requires restart)
#wal_buffers = -1                       # min 32kB, -1 sets based on shared_buffers
                                        # (change requires restart)
#wal_writer_delay = 200ms               # 1-10000 milliseconds
#wal_writer_flush_after = 1MB           # measured in pages, 0 disables
#commit_delay = 0                       # range 0-100000, in microseconds
#commit_siblings = 5                    # range 1-1000

wal_level：控制wal存储的级别，wal_level决定有多少信息被写入到WAL中，默认值是最小的（minimal），
logical 主要用于logical decoding 场景
```

```
fsync：该参数直接控制日志是否先写入磁盘。默认值是ON（先写入），表示更新数据写入磁盘时系统必须等待
synchronous_commit：参数配置是否等待WAL完成后才返回给用户事务的状态信息。默认值是ON，表明必须等
wal_sync_method：WAL写入磁盘的控制方式，默认值是fsync，可选用值包括open_datasync、fdatasync、
full_page_writes：表明是否将整个page写入WAL。
wal_buffers：用于存放WAL数据的内存空间大小，系统默认值是64K，该参数还受wal_writer_delay、commi
wal_writer_delay：WalWriter进程的写间隔时间，默认值是200毫秒，如果时间过长可能造成WAL缓冲区的内
wal_writer_flush_after：
commit_delay：表示一个已经提交的数据在WAL缓冲区中存放的时间，默认值是0毫秒，表示不用延迟；设置为
commit_siblings：表示当一个事务发出提交请求时，如果数据库中正在执行的事务数量大于commit_sibling
 wal_writer_flush_after：当脏数据超过阈值时，会被刷出到磁盘。
```

2.4.7、CheckPoint（检查点）进程

检查点是系统设置的事务序列点，设置检查点保证检查点前的日志信息刷到磁盘中。postgresql.conf文件中与之相关的参数有：

```
# - Checkpoints -
#checkpoint_timeout = 5min              # range 30s-1d
#max_wal_size = 1GB
#min_wal_size = 80MB
#checkpoint_completion_target = 0.5     # checkpoint target duration, 0.0 - 1.0
#checkpoint_flush_after = 256kB         # measured in pages, 0 disables
#checkpoint_warning = 30s               # 0 disables
```