Chapter 2

Process and Memory Architecture

In this chapter, the process architecture and memory architecture in PostgreSQL are summarized to help to read the subsequent chapters. If you are already familiar with them, you may skip over this chapter.

2.1. Process Architecture

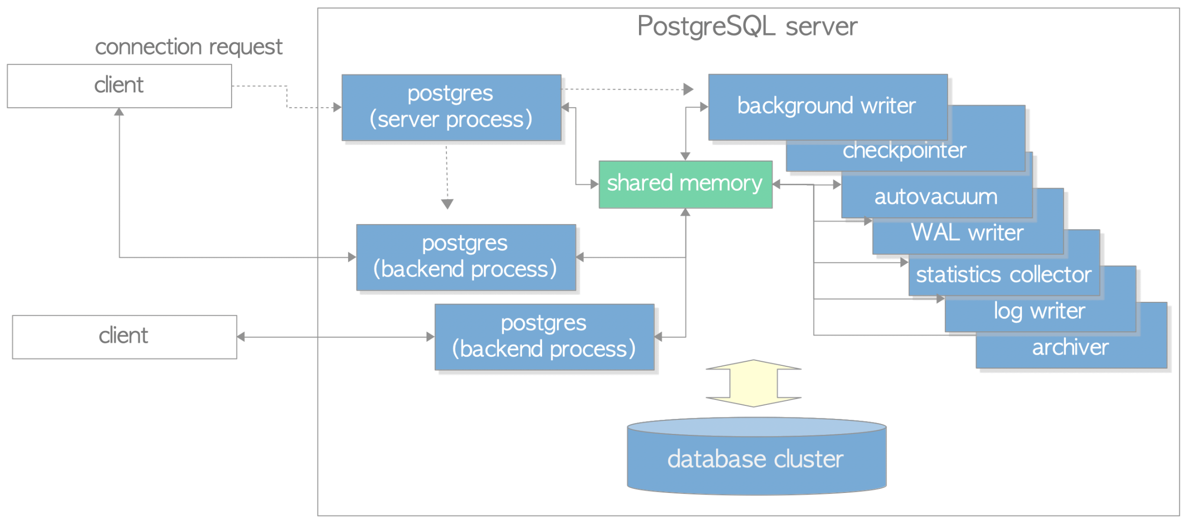
PostgreSQL is a client/server type relational database management system with a multi-process architecture that runs on a single host.

A collection of multiple processes that cooperatively manage a database cluster is usually referred to as a *'PostgreSQL server'*. It contains the following types of processes:

- The **postgres server process** is the parent of all processes related to database cluster management.
- Each **backend process** handles all queries and statements issued by a connected client.
- Various **background processes** perform processes perform tasks for database management, such as VACUUM and CHECKPOINT processing.
- **Peplication-associated processes** perform streaming replication. More details are described in Chapter 11.
- **Background worker processes** supported from version 9.3, it can perform any processing implemented by users. For more information, refer to the official document.

In the following subsections describe the details of the first three types of processes.

Fig. 2.1. An example of the process architecture in PostgreSQL.



This figure shows processes of a PostgreSQL server: a postgres server process, two backend processes, seven background processes, and two client processes. The database cluster, the shared memory, and two client processes are also illustrated.

2.1.1. Postgres Server Process

As already described above, a *postgres server process* is a parent of all processes in a PostgreSQL server. In the earlier versions, it was called 'postmaster'.

When you execute the pg_ctl utility with *start* option, a postgres server process starts up. It then allocates a shared memory area in memory, starts various background processes, starts replication-associated processes and background worker processes if necessary, and waits for connection requests from clients. Whenever it receives a connection request from a client, it starts a backend process. (The started backend process then handles all queries issued by the connected client.)

A postgres server process listens to one network port, the default port is 5432. Although more than one PostgreSQL server can be run on the same host, each server must be set to listen to different port number, such as 5432, 5433, and so on.

2.1.2. Backend Processes

A backend process, also called a *postgres* process, is started by the postgres server process and handles all queries issued by one connected client. It communicates with the client using a single TCP connection and terminates when the client disconnects.

Since a backend process is only allowed to operate on one database, you must explicitly specify the database you want to use when connecting to a PostgreSQL server.

PostgreSQL allows multiple clients to connect simultaneously. the configuration parameter *max_connections* controls the maximum number of the clients (default is 100).

If many clients, such as WEB applications, frequently connect and disconnect from a PostgreSQL server, it can increase the cost of establishing connections and creating backend processes, as PostgreSQL does not have a native connection pooling feature. This can have a negative impact on the performance of the database server. To deal with such a case, a connection pooling middleware such as pgbouncer or pgpool-II) is usually used.

2.1.3. Background Processes

Table 2.1 shows a list of background processes. In contrast to the postgres server process and the backend process, it is impossible to explain each of the functions simply. This is because these functions depend on the individual specific features and PostgreSQL internals. Therefore, in this chapter, only introductions are made. Details will be described in the following chapters.

Table 2.1: background processes.

process	description	reference
background writer	This process writes dirty pages on the shared buffer pool to a persistent storage (e.g., HDD, SSD) on a regular basis gradually.	Section 8.6

	(In versions 9.1 or earlier, it was also responsible for the checkpoint process.)	
checkpointer	This process performs the checkpoint process in versions 9.2 or later.	Section 8.6, Section 9.7
autovacuum launcher	This process periodically invokes the autovacuum-worker processes for the vacuum process. (More precisely, it requests the postgres server to create the autovacuum workers.)	Section 6.5
WAL writer	This process writes and flushes the WAL data on the WAL buffer to persistent storage periodically.	Section 9.9
statistics collector	This process collects statistics information such as for pg_stat_activity and pg_stat_database, etc.	
logging collector (logger)	This process writes error messages into log files.	
archiver	This process executes archiving logging.	Section 9.10

The actual processes of a PostgreSQL server are shown below. In the following example, there is one postgres server process (pid is 9687), two backend processes (pids are 9697 and 9717), and several background processes listed in Table 2.1. See also Figure 2.1.

```
postgres> pstree -p 9687
-+= 00001 root /sbin/launchd
 \-+- 09687 postgres /usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data
   |--= 09688 postgres postgres: logger process
   |--= 09690 postgres postgres: checkpointer process
   |--= 09691 postgres postgres: writer process
   |--= 09692 postgres postgres: wal writer process
   |--= 09693 postgres postgres: autovacuum launcher process
   |--= 09694 postgres postgres: archiver process
   |--= 09695 postgres postgres: stats collector process
   |--= 09697 postgres postgres: postgres sampledb 192.168.1.100(54924) idle
   \--= 09717 postgres postgres: postgres sampledb 192.168.1.100(54964) idle in transaction
```

2.2. Memory Architecture

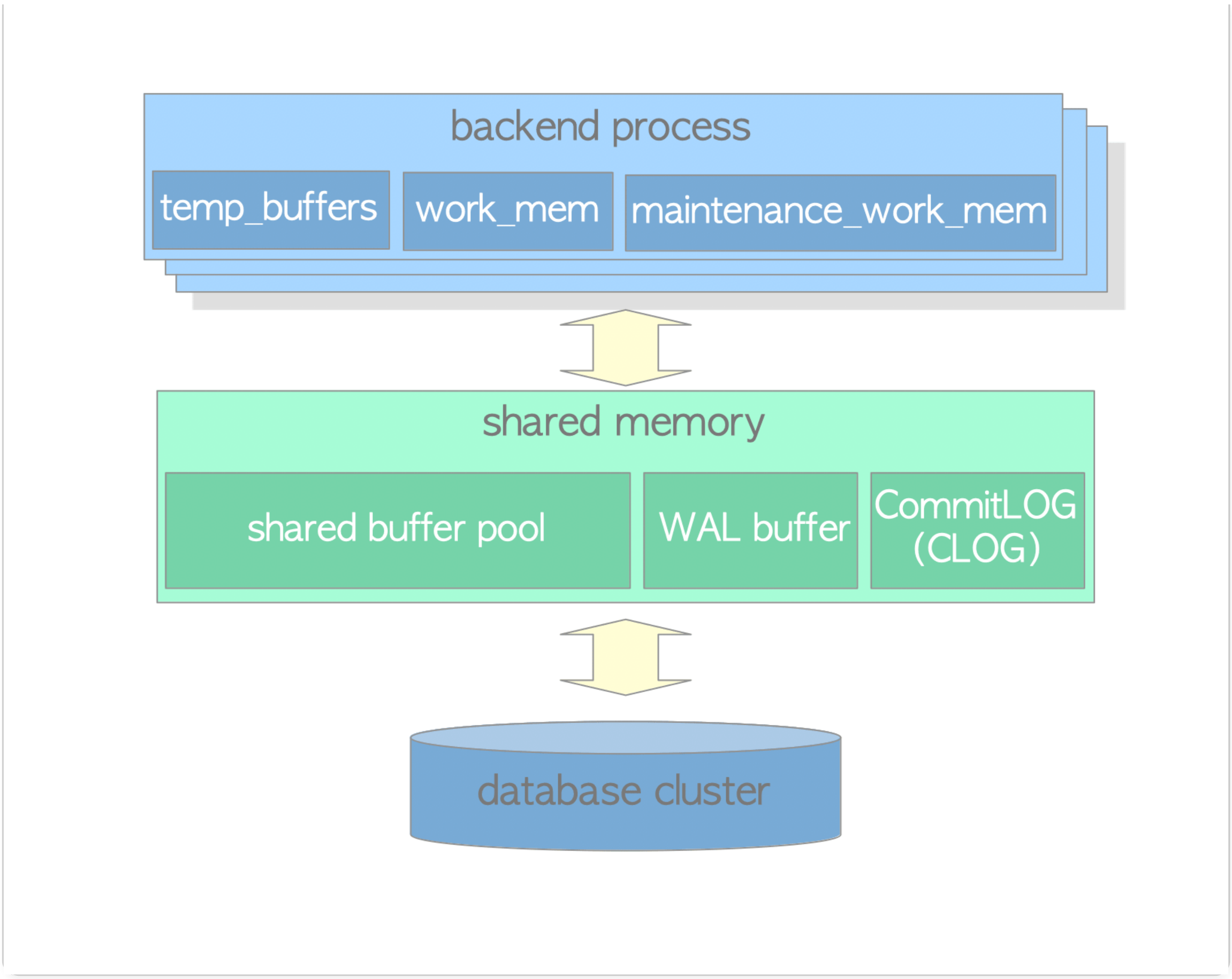
Memory architecture in PostgreSQL can be classified into two broad categories:

- Local memory area – allocated by each backend process for its own use.

  Shared memory area – used by all processes of a PostgreSQL server.

In the following subsections, those are briefly descibed.

Fig. 2.2. Memory architecture in PostgreSQL.



2.2.1. Local Memory Area

Each backend process allocates a local memory area for query processing. The area is divided into several sub-areas, whose sizes are either fixed or variable. Table 2.2 shows a list of the major sub-areas. The details of each sub-area will be described in the following chapters.

Table 2.2: Local memory area

sub-area	description	reference
work_mem	The executor uses this area for sorting tuples by ORDER BY and DISTINCT operations, and for joining tables by merge-join and hash-join operations.	Chapter 3
maintenance_work_mem	Some kinds of maintenance operations (e.g., VACUUM, REINDEX) use this area.	Section 6.1
temp_buffers	The executor uses this area for storing temporary tables.	

2.2.2. Shared Memory Area

A shared memory area is allocated by a PostgreSQL server when it starts up. This area is also divided into several fixed-sized sub-areas. Table 2.3 shows a list of the major sub-areas. The details will be described in the following chapters.

Table 2.3: Shared memory area

sub-area	description	reference
shared buffer pool	PostgreSQL loads pages within tables and indexes from a persistent storage to this area, and operates them directly.	Chapter 8
WAL buffer	To ensure that no data has been lost by server failures, PostgreSQL supports the WAL mechanism. WAL data (also referred to as XLOG records) are the transaction log in PostgreSQL. The WAL buffer is a buffering area of the WAL data before writing to a persistent storage.	Chapter 9
commit log	The commit log (CLOG) keeps the states of all transactions (e.g., in_progress, committed, aborted) for the concurrency control (CC) mechanism.	Section 5.4

In addition to the shared buffer pool, WAL buffer, and commit log, PostgreSQL allocates several other areas, as shown below:

- Sub-areas for the various access control mechanisms. (e.g., semaphores, lightweight locks, shared and exclusive locks, etc)
- Sub-areas for the various background processes, such as the checkpointer and autovacuum.
- Sub-areas for transaction processing, such as savepoints and two-phase commit.

and others.