AgensGraph Operations Manual

SKAI worldwide

September 30, 2022

Introduction

What is AgensGraph?

AgensGraph is a new generation multi-model graph database for the modern complex data environment. Based on PostgreSQL, AgensGraph supports both relational and graph data models simultaneously. This enables developers to integrate the legacy relational data model and the flexible graph data model in one database. AgensGraph supports ANSI-SQL and OpenCypher (https://www.opencypher.org). The SQL and Cypher queries can be integrated into a single query in AgensGraph.

AgensGraph is robust, fully-featured, and ready for enterprise use. It is optimized for handling complex connected graph data and provides plenty of powerful database features essential to the enterprise database environment including ACID transactions, multi-version concurrency control, stored procedure, triggers, constraints, sophisticated monitoring, and a flexible data model (JSON). Moreover, AgensGraph leverages the rich eco-systems of PostgreSQL and can be extended with many outstanding external modules, like PostGIS.

Deployment

System Requirements

AgensGraph requires at least 4GB of RAM and 2.5GB of disk space to run. It may run on any general Unix-compatible platforms, but the officially-certified platforms are Linux series (Centos, Ubuntu, RHEL) and Windows series (Windows Server 2008 64bits, Windows Server 2012 64bits, Windows 7 64bits).

Pre-install tasks

User account

Like other externally-accessible server daemons, it is recommended to run AgensGraph with a separate user account. If you are on Unix-compatible platforms, you can add a user account using `useradd` or `adduser` command.

```
$ useradd agens
```

Kernel resource management

Shared memory and Semaphores

Shared memory and semaphores are collectively referred to as “System V IPC”, which is required to run AgensGraph. In the case where the size requested by AgensGraph exceeds the IPC limit, the server fails to start.

Name	Description	Reasonable values
SHMMAX	Maximum size of shared memory segment in bytes	At least 1kB(or more if multiple copies of the server are running)
SHMMIN	Minimum size of shared memory segment in bytes	1
SHMALL	Total amount of shared memory available(bytes or pages)	Same as SHMMAX in bytes. Ceil (SHMMAX / PAGE_SIZE) in pages
SHMSEG	Maximum number of shared memory segments per process	Only 1 segment is needed, but the default is much higher
SHMMNI	The maximum number of system-wide shared memory segments	Like SHMSEG plus room for other applications
SEMMNI	Maximum number of semaphore identifiers (e.g. set)	At least ceil ((max_connections + autovacuum_max_workers + 4) / 16)
SEMMNS	System-wide semaphore	Maximum number of ceil ((max_connections + autovacuum_max_worker + 4) / 16) * 17 and excess of other applications
SEMMSL	Maximum number of semaphores per set	At least 17
SEMMAP	The number of items in a semaphore map	Refer to text
SEMVMX	Maximum semaphore value	At least 1000 (default is usually 32767; do not change it unless needed)

AgensGraph requires several bytes of System V shared memory per server copy (usually 48 bytes for 64-bit platforms). This amount is easily allocated in mordern operating systems. However, when running multiple copies of the server or when other applications also use System V shared memory, it may be necessary to increase SHMMAX (the maximum shared memory size in bytes) or SHMALL (the system-wide System V shared memory). Note that

SHMALL is handled on a page-by-page basis rather than a byte-by-byte basis on most systems.

In the case of AgensGraph, the minimum size of the shared memory segment (SHMMIN) is about 32 bytes at most (usually 1) and is less likely to cause a problem. The maximum number of segments system-wide (SHMMNI) or the maximum number per process (SHMSEG) is highly unlikely to cause problems unless the system is set to zero.

AgensGraph uses one semaphore per connection (max_connections) and one semaphore per autovacuum worker process (autovacuum_max_workers) among the 16 sets. Each set has a 17th semaphore with a "magic number" that detects collisions with a semaphore set of other applications. The maximum number of semaphores in the system is set by SEMMNS, and at least "max_connections + autovacuum_max_workers + 1 added to 16 allowed connections + worker" (see Table 1-1 formula). The parameter SEMMNI limits the number of semaphore sets that can exist concurrently on the system. It should be at least ceil ((max_connections + autovacuum_max_workers + 4) / 16). Reducing the number of acceptable connections can be a temporary solution in case of failure, but you may receive an ambiguous message from the semget function like "No space left on device."

In some cases, it may be necessary to increase SEMMAP so that it can be similar to SEMMNS at least. This parameter defines the size of the semaphore resource map, which contains the entries that adjacent blocks of the semaphore need. A set of released semaphores is added to the entry adjacent to the released block or is registered as a new entry. When the map is full, the released semaphore disappears (until it is rebooted). As the semaphore space is divided, the number of available semaphores becomes smaller.

SEMMSL that determines the number of semaphores to be included in a set should be at least 17 for AgensGraph.

Other settings related to "semaphore undo" such as SEMMNU and SEMUME do not affect AgensGraph.

> Note: If you encounter a "Warning: Out of Shared Memory" error while performing some queries, visit SKAI worldwide 's website(https://SKAI worldwide .net/contact) and request a technical support; the team will provide you with a guide on the memory settings suitable for your site environment.

Single instance install

There are two ways to install AgensGraph. You can download the binary from the official website and download the source code and build/install it. Either way, we recommend that you work with an account for managing AgensGraph.

Installing via binary download

Download the binary or installer suitable for your operating system from the S/W Download menu of the official AgensGraph website.

Linux

Linux binaries are compressed with tarball. After unzipping them to the desired location, perform the post-installation task to complete the installation.

Windows

Coming soon.

Installation by Source Code Build

1. Install the following essential libraries according to each OS By root account or user account with sudo.

```
# CENTOS
  $ sudo yum install gcc glibc glib-common readline readline-devel zlib
 zlib-devel
# Fedora
  $ sudo dnf install gcc glibc bison flex readline readline-devel zlib
zlib-devel
# Ubuntu
  $ sudo apt-get install build-essential libreadline-dev zlib1g-dev fle
x bison
# macOS (install Xcode)
  $ sudo xcode-select --install
```

2. make the User Account(Normally 'agens') to install the agensgraph, Connect to the github of AgensGraph and clone the source code.

```
$ sudo useradd agens
$ sudo passwd agens
$ su - agens

$ git clone https://github.com/SKAI worldwide -oss/agensgraph.git
```

3. Move to the clone location and run configure on the source tree. At this time, you can set the location where AgensGraph will be installed via –prefix.

```
### If the Installation Path is /home/agens/agensgraph-2.13.0
# Normal
$ ./configure --prefix=/home/agens/agensgraph-2.13.0
# If Use of XML
```

```
$ ./configure --prefix=/home/agens/agensgraph-2.13.0 --with-libxml
# If Use of Python
$ ./configure --prefix=/home/agens/agensgraph-2.13.0 --with-python
```

4. Perform a build.

```
$ make install
```

5. Install the extension module and binaries.

```
$ make install-world
```

Post-Installation Tasks (Linux)

In the case of Unix series and Linux, if you do not register an environment variable, you will not be able to read/activate the installed libraries properly, and you will have to enter the absolute path when making a call. You need to make the Data Cluster Directory(=Database Path) and add an environment variable to bash_profile of the user with AgensGraph installed as follows:

```
$ sudo mkdir -p /AGDATA
$ sudo chown -R agens:agens /AGDATA
$ sudo chmod -R 0700 /AGDATA
$ vi ~/.bash_profile

export AGHOME=/home/agens/agensgraph-2.13.0
export AGDATA=/AGDATA
export PATH=\$AGHOME/bin:\$PATH
export LD\_LIBRARY\_PATH=\$AGHOME/lib:$LD\_LIBRARY\_PATH

:wq

$ source ~/.bash_profile
```

Environment	Description
AGHOME	This is the directory where AgensGraph is installed.
AGDATA	Location of the data directory
PATH	Set $AGHOME/bin as the directory path to use AgensGraph.
LD_LIBRARY_PATH	This is the path where the shared library needed for using AgensGraph is located Set $AGHOME/lib.

Operations

initDB

To run AgensGraph, you need to initialize the database storage area using *initdb*. Such an area is called a database cluster, which is a collection of databases managed by a single instance of a running database server. A database cluster consists of a single directory where all the data is stored. From the file system perspective, it is considered a data directory or a data area. You can set where to store this data with the -D option.

```sql
# Normal
$ initdb -E UTF8 --locale=en_US.UTF-8 -U agens -D $AGDATA
# Change the WAL SIZE
$ initdb -E UTF8 --locale=en_US.UTF-8 -U agens -D $AGDATA --wal-segsize=SIZE
# Change the WAL Directory
$ initdb -E UTF8 --locale=en_US.UTF-8 -U agens -D $AGDATA --waldir=WALDIR
```

Or, you can initialize the database storage area using ag_ctl.

```
$ ag_ctl initdb -D $AGDATA
```

initdb creates a directory if the specified directory does not exist, and denies execution if it is already initialized.

Create a role

Roles can be considered a database user(s), or a group of database users, depending on how the database is set up. As a role can own a database object, it may assign permissions for the object to other roles to control who can access the object. As the membership of a role can be granted to other roles, you may make the member role use the permissions assigned to other roles. To create/delete a role, you should use the following SQL command.

```
db=# CREATE ROLE name;
db=# DROP ROLE name;
```

For your convenience, *createuser* and *dropuser* can be invoked from the shell command line (serving as the wrapper of SQL commands).

```
$ createuser name
$ dropuser name
```

createdb

To create/remove a database, start the AgensGraph server and create it using SQL commands *CREATE DATABASE* and *DROP DATABASE*.

```
db=# CREATE DATABASE name; db=# DROP DATABASE name;
```

The current role that has performed *CREATE DATABASE* automatically becomes the owner of the new database.

For your convenience, you may create/remove databases by calling *createdb* and *dropdb* from the shell command line. You may use the *-O* option to designate the owner of the new database that you are creating.

```
$ createdb dbname [-O rolename]
$ dropdb dbname
```

Startup

You should start the database server before accessing the database using *ag_ctl*. You need an initialized database repository to start, and have to specify the corresponding directory with -D option. If you set the database repository location via AGDATA, an environment variable, you may start the server without the -D option. Using -l option, you may set a logfile that will contain logs.

```
$ ag_ctl start [-D $AGDATA] [-l $AGDATA/logfile]
```

Shutdown

There are several ways to terminate AgensGraph. You may control the types of termination depending on which signal is sent to the master process.

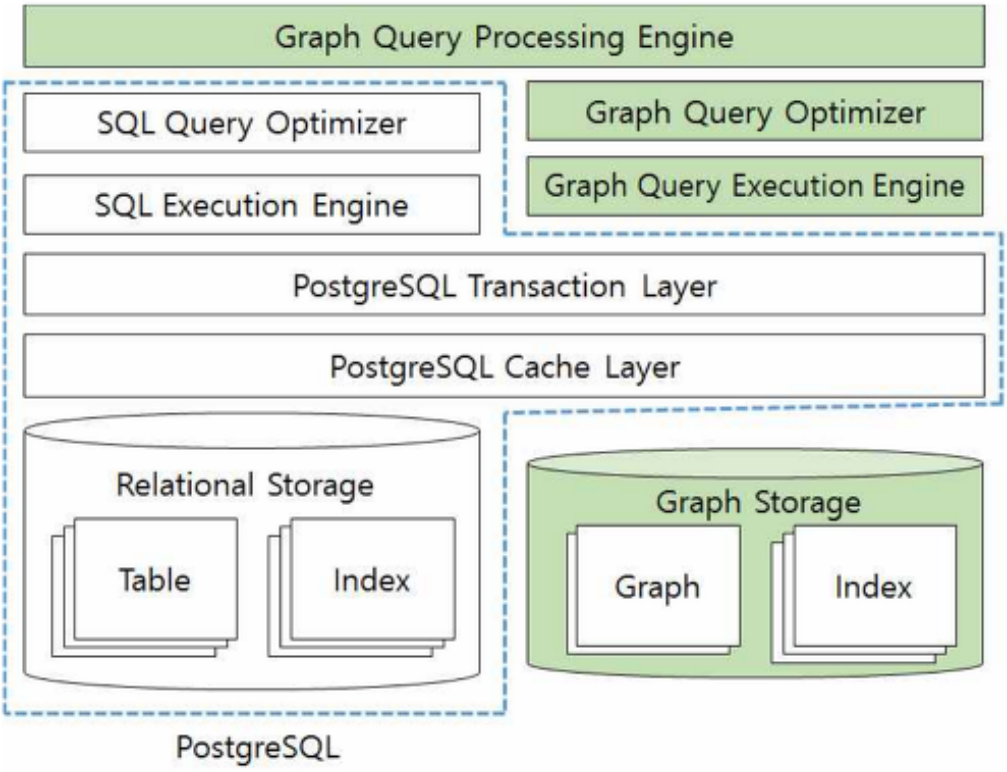
Mode	Description
SIGTERM	Smart shutdown mode. The server does not allow any new connections, but waits for the old session to terminate normally before shutdown.
SIGINT	Fast shutdown mode. The server does not allow any new connections, and aborts/terminates transactions for all existing processes.
SIGQUIT	Immediate shutdown mode. The server terminates all child processes. Child processes that do not terminate within 5 seconds are forced to terminate by sending SIGKILL, which leads to recovery at the next startup; therefore, this is recommended only in an emergency.

You can terminate *ag_ctl* with -m option as follows (unless -m is specified, the default is smart):

```
$ ag_ctl stop [-D DATADIR] -[m smart|fast|immediate]
```

Architecture

AgensGraph is a multi-model database developed on the basis of PostgreSQL. AgensGraph is designed to offer all the features provided by PostgreSQL and process graph queries using Cypher. Internally, they are largely divided into the PostgreSQL processor and the Graph processor, both of which jointly use cache and transaction areas. Users may take advantage of better performance and more convenient implementation by simultaneously performing SQL and Cypher queries on a single query.



Process structure

AgensGraph uses the same client/server model as PostgreSQL. A session (work) consists of the following interacting processes (programs):

- Server process
  - Manages database files, processes (accepts or rejects) the requests made by client applications to connect to the server, and prepares base work to enable clients to use the database. The name of this process is postgres.
- Client process

- Refers to user-side applications that need to access the database. There are a wide range of client applications, understandably: text-based programs, graphical applications, or, in some cases, web pages that can be shown through a web server. Some client programs are included in the distribution packages. Most of such applications are developed by users.

As is often the case with programs in a client/server environment, AgensGraph also allows the client and server to be different hosts. In this case, communications between them are made mostly under the TCP/IP network infrastructure- a point we should clearly understand; if the client and server are different, the client cannot directly access a database file it wants. In other words, a file that can be accessed by the client is a file on the host where the client is running, not a file on the host where the server is running.

For concurrent access by users, the AgensGraph server creates a new process for each connection. This means that when a client and a newly created server process communicate, the communication is done without intervention by postgres (master process). Simply put, postgres is always running on the server host to process the clients' connection requests and create new child server processes.

Security

Client authentication and database roles are important in security. Since AgensGraph was developed based on PostgreSQL, the PostgreSQL technical documents were quoted when introducing the features of the RDB (not GDB).

Client Authentication

When a client application connects to a database server, it is important to limit the number of database users who are allowed to access. Authentication is a process for building client IDs by the database server and for determining whether to allow connections of client applications when they try to access with the names of the database users (or users running the client applications).

AgensGraph provides several different client authentication methods. Such methods used to authenticate a particular client connection can be selected on the basis of (client) host address, database, and/or user.

The user names of the AgensGraph database are logically distinct from the user names of the operating system on which the server is running. Any user on a particular server can have an account on the server machine. It is reasonable, however, to assign a database user name that matches the operating system user name. The problem is that a server that accepts remote connections may have many database users without a local operating system account; in such cases, it is not necessary to associate database user names with OS user names.

pg_hba.conf File

Client authentication is traditionally controlled by pg_hba.conf, a configuration file, which is stored in the data directory of the database cluster.

The format of pg_hba.conf is a set of records, in which each line is a record. Empty lines are ignored, and the text that comes after "#" is ignored as well. A record, which cannot continue after a line break, consists of several fields with delimiters such as spaces and/or tabs. By using double quotes for field values, you may include spaces in the field. If you use quotation marks in keywords of the database or user/address fields, the words will lose their special meaning.

Each record specifies a connection type to be used for connection that matches these parameters, the range of client IP address (if applicable), database name, user name, and authentication method. The first record that matches the connection type, client address, the requested database, and user name is used during authentication. There is no "fall-through" or "backup." In the case where a record is selected but authentication fails, the next record will not be authenticated. Access is denied if there is no matching record.

A record is one of the following seven types:

```
local         database  user  auth-method [auth-options]
host          database  user  address     auth-method  [auth-options]
hostssl       database  user  address     auth-method  [auth-options]
hostnossl     database  user  address     auth-method  [auth-options]
hostgssenc    database  user  address     auth-method  [auth-options]
hostnogssenc  database  user  address     auth-method  [auth-options]
host          database  user  IP-address  IP-mask      auth-method  [auth-opt
ions]
hostssl       database  user  IP-address  IP-mask      auth-method  [auth-opt
ions]
hostnossl     database  user  IP-address  IP-mask      auth-method  [auth-opt
ions]
hostgssenc    database  user  IP-address  IP-mask      auth-method  [auth-opt
ions]
hostnogssenc  database  user  IP-address  IP-mask      auth-method  [auth-opt
ions]
```

The meaning of each field is as follows:

- **`local`**
  This record corresponds to a connection through a Unix domain socket. If there is no record of this type, the Unix domain socket connection is impossible.

- **`host`**
  This record corresponds to a connection through TCP/IP. The host record matches the SSL (or non-SSL) connection attempt.

- **`hostssl`**
  Even if this record matches the connection attempt through TCP/IP, it corresponds to a connection using SSL encryption only. To use this option, the server must have built-in SSL support. Set the ssl configuration parameter(s) to enable SSL when the server starts.

- **`hostnossl`**
  This type of record works in contrast to hostssl, matching a connection attempt on TCP/IP that does not use SSL.

- **`hostgssenc`**
  This record matches connection attempts made using TCP/IP, but only when the connection is made with GSSAPI encryption. To make use of this option the server must be built with GSSAPI support. Otherwise, the hostgssenc record is ignored except for logging a warning that it cannot match any connections.

- **`hostnogssenc`**
  This record type has the opposite behavior of hostgssenc; it only matches connection attempts made over TCP/IP that do not use GSSAPI encryption.

- **`database`**
  This record specifies the database name.

  - Specify `all` to match all databases.

  - Specify `sameuser` to match the records when the requested database has the same name as the requested user.

  - Specify `samerole` to indicate whether the requested user should be a member of the same "role" as the requested database. "superuser" shall not be regarded as a member of the role (in samerole) just because it is a superuser, if it is not directly or indirectly an explicit member of the role.
  - Specify `replication` to match the records if a replication connection is requested (replication connections do not specify a specific database). For other cases, it is the name of a specific AgensGraph database. You can write multiple database names separated by commas. A file containing the database name can be specified by prefixing the file name with @.
- **`user`**
  Specifies the database user name that matches this record. The value `all` specifies that it matches all users. Otherwise, this is either the name of a specific database user, or a group name preceded by +. (Recall that there is no real distinction between users and groups in PostgreSQL; a + mark really means "match any of the roles that are directly or indirectly members of this role", while a name without a + mark matches only that specific role.) For this purpose, a superuser is only considered to be a member of a role if they are explicitly a member of the role, directly or indirectly, and not just by virtue of being a superuser. Multiple user names can be supplied by separating them with commas. A separate file containing user names can be specified by preceding the file name with @.

  Specifies the database user name that matches this record. `all` is specified to match all users. In other cases, it is the name of a particular database user or a group name preceded by +. (In AgensGraph, there is no real difference between user name and group name: "" mark actually means "matching any role that is a direct or indirect member of this role" and the name without "+" mark matches a specific role.) For this reason, a superuser shall be considered a role member only when it is an explicit member of the role, directly or indirectly, not because it is merely a superuser. You can use multiple usernames separated by commas. A file containing a user name can be specified by prefixing the file name with @.

- **address**
  Specifies the client machine address that matches this record. This field may contain either a host name, an IP address range, or one of the special keywords described below.

  An IP address is specified in the dotted decimal standard notation of the CIDR mask length. The mask length indicates the number of upper bits of the client IP address to be matched. The bit to the right of this should be 0 at the given IP address. There should be no space between the IP address, "/", and CIDR mask length.

  Typical examples of such an IP address range: `172.20.143.89/32` for a single host, `172.20.143.0/24` for a small network, and `10.6.0.0/16` for a large network. `0.0.0.0/0` represents all IPv4 addresses, and `::/0` represents all IPv6 addresses. If you want to specify a single host, use CIDR mask 32 for IPv4 and 128 for IPv6. Do not omit 0 at the end of the network address.

  You can use "all" to match any IP address or `samehost` to match only the server's own IP address. You may also use `samenet` to match all addresses on the subnet to which the server is directly connected.

- **IP-address, IP-mask**
  This field can be used as an alternative to the `CIDR-address` notation. Instead of specifying the mask length, the actual mask is specified by separating it with commas. For example, `255.0.0.0` represents the IPv4 CIDR mask length 8, and `255.255.255.255` represents the CIDR mask length 32.

  This field applies to `host`, `hostssl`, and `hostnossl` records.

- **auth-method**
  Specifies an authentication method to use when the connection matches this record. Possible choices are:

  - **trust**
    Allows unconditional connections. This method allows any logged-in users to connect to the AgensGraph database server without requesting a password or other authentications.

  - **reject**
    Refuses unconditional connections. This is useful when you want to "filter" specific hosts in a group. For example, a `reject` line blocks connections from particular hosts, and subsequent lines allow connections with other hosts on a particular network.

- **md5**
  The client should provide a double-MD5-hashed password for authentication.

- **scram-sha-256**
  The method scram-sha-256 performs SCRAM-SHA-256 authentication, as described in RFC 7677. It is a challenge-response scheme that prevents password sniffing on untrusted connections and supports storing passwords on the server in a cryptographically hashed form that is thought to be secure.

This is the most secure of the currently provided methods, but it is not supported by older client libraries.

- **password**
  The client must provide an unencrypted password for authentication. As the password is transmitted in plaintext over the network, do not use it on untrusted networks.

- **gss**
  Authenticates users using GSSAPI. This can be used only on the TCP/IP connections.

- **sspi**
  Authenticates users using SSPI. This can be used only on Windows.

- **ident**
  Obtains the operating system user name of the client by contacting the client's ident server and verifies if it matches the requested database user name. Ident authentication can only be used on TCP/IP connections. For local connections, peer authentication is used instead.

- **peer**
  Obtains the operating system user name of the client from the operating system and verifies if it matches the requested database user name. This is only available for local connections.

- **ldap**
  Authentication using an LDAP server.

- **radius**
  Authentication using a RADIUS server.

- **cert**
  Authentication using SSL client authentication.

- **pam**
  Authenticates using Pluggable Authentication Modules (PAM) provided by the operating system.

- **bsd**
  Authenticate using the BSD Authentication service provided by the operating system.

- **auth-options**
  After the auth-method field, there can be a name=value field that specifies options for the authentication method.

Files containing @ shall be read as a list of names separated by spaces or commas. Annotations marked as #, like `pg_hba.conf`, and nested @ phrases are allowed. If the file name followed by @ is not an absolute path, it is treated as a relative path to the directory where the reference file is located.

The order of the records is important because the `pg_hba.conf` record is checked sequentially for each connection attempt. The connection (matching) parameters are meticulous and the authentication method is loose in the early records, while the connection matching parameters are loose and the authentication method is strong in the late records. For instance, you may use a trust authentication for a local TCP/IP connection while making a remote TCP/IP connection. In such a case, the record specifying the trust authentication for the connection from `127.0.0.1` appears before a record that supports password authentication for various allowed client IP addresses.

User Name Maps

If you are using an external authentication system such as Ident or GSSAPI, the name of the operating system user initiating the connection may be different from the name of the database user to connect. In such a case, you may use the user name map to map the operating system user name to the database user name. To use the user name mapping, you should specify `map=map-name` in the `pg_hba.conf` option field. This option is supported in all authentication methods that receive external user names. As different connections may require different mappings, the name of the map to use (for specifying maps per connection) can be designated in the `map-name` parameter of `pg_hba.conf`.

The username map is defined in the ident map file, and its name is `pg_ident.conf` (default) and stored in the data directory.

```
map-name system-username database-username
```

Comments and spaces are processed equally in pg_hba.conf. map-name is an arbitrary name used in pg_hba.conf to refer to the mapping. The other two fields specify the operating system user name and the database user name. You can specify multiple user mappings within a single map by using the same map-name several times.

There is no restriction as to how many database users correspond to a given operating system user (and vice versa). Therefore, with the entries in the map, you should consider that "this operating system user is allowed to connect as a database user", rather than that the users are identical. Connections are allowed if there is a map entry in which a user name obtained from the external authentication system pairs with a database user name used when the user made a connection request.

If the system-username field begins with a slash (/), the remainder of the field is processed as a regular expression. Regular expressions may contain single captures or bracket expressions, and can be referenced in the database-username field by \1 (backslash). It can map multiple user names in a single line, and is especially useful for simple syntax substitution. Examples:

```
mymap /^(.*)@mydomain\.com$\1
mymap /^(.*)@otherdomain\.com$ guest
```

This entry deletes the domain part for the user using the system user name ending in @mydomain.com and allows any user whose system name ends in @otherdomain.com to log in as guest.

Authentication Methods

This section details authentication methods.

Trust Authentication

If trust authentication is specified, AgensGraph assumes that anyone who can connect to the server using the specified database user name is authenticated for database access (including the superuser name). The limitations of the database and user columns still apply as well. This method should only be used if adequate operating system level protection is provided for server connections.

trust authentication is appropriate for local connections to a single user workstation and is very convenient. It is not appropriate, however, for multiuser machines in general. trust authentication is suitable for TCP/IP connections only when all the users on all machines that are allowed to connect to the server are trusted in pg_hba.conf, which specifies trust. It is unreasonable to use trust for TCP/IP connections except for localhost (127.0.0.1).

Password Authentication

The password-based authentication methods are md5, scram-sha-256, and password. Both methods work similarly except that the password is sent in MD5 hash and plaintext respectively.

While md5 has been widely used in previous versions, scram-sha-256 is the most secure method in this version. Using scram-sha-256 or md5 to guard against password "sniffing" attacks is preferable; normal password should be avoided if possible. However, md5 cannot be used with the db_user_namespace feature. You can safely use a password if the connection is protected by SSL encryption.

The AgensGraph database password is distinguished from the operating system user password. Passwords of each database user are stored in the `pg_authid` system catalog. Such passwords can be managed by SQL commands CREATE USER and ALTER ROLE (e.g. CREATE USER foo WITH PASSWORD 'secret'). If the password is not set for a user, the stored password is null and password authentication always fails for the user.

GSSAPI Authentication

GSSAPI is an industry-standard protocol for security authentication as defined in RFC 2743, and provides single sign-on for GSSAPI-supporting systems. Authentication itself is secure, but if you do not use SSL, the data sent over the database connection is transmitted unencrypted.

When you connect to the database, you should check if there is a ticket for the security rule that matches the name of the requested database user. For example, if the database name is `fred`, the security rule `fred@EXAMPLE.COM` is connectable.

The following configuration options are supported for GSSAPI.

- **`include_realm`**
  If set to 1, the realm name of the authenticated user security rule is included in the system user name transmitted through the user name mapping. This is useful when processing users in multiple realms.

- **`map`**
  Allows mapping between the system and database user names.

- **`krb_realm`**
  Sets a realm that matches the user security rule name. When this parameter is set, only users in the corresponding realm are allowed. If not set, users in all realms can connect; the realm depends on whether the username mapping is complete or not.

SSPI Authentication

SSPI is a Windows technology for single sign-on (SSO) security authentication. AgensGraph uses SSPI in `negotiate` mode. It uses Kerberos if available, and in other cases, automatically falls back to NTLM. SSPI authentication works only when both the server and client use Windows; if GSSAPI is available, it works with non-Windows as well.

While using Kerberos authentication, SSPI works the same way as GSSAPI.

The following configuration options are supported for SSPI.

- **`include_realm`**
  If set to 1, the realm name of the authenticated user security rule is included in the system user name transmitted through the user name mapping. This is useful when processing users in multiple realms.

- **`map`**
  Allows mapping between the system and database user names.

- **`krb_realm`**
  Sets a realm that matches the user security rule name. When this parameter is set, only users in the corresponding realm are allowed. If not set, users in all realms can make a connection; the realm depends on whether the username mapping is complete or not.

Ident Authentication

The ident authentication method works by acquiring the client's operating system user name from the ident server and using it as an allowed database user name (optional user name mapping is applied). This authentication is supported only for TCP/IP connections.

The following configuration option is supported for ident:

- **`map`**
  Allows mapping between the system and database user names.

Some ident servers have a nonstandard option that allows the returned user name to be encrypted using a key known only to the administrator of the original machine. As AgensGraph does not have a way to decrypt the returned string to determine the actual user name, you should not use this option if you are using AgensGraph on the ident server.

Peer authentication

The peer authentication method works by acquiring the client's operating system user name from the kernel and using it as an allowed database user name (optional user name mapping is applied). This authentication is supported only for local connections.

The following configuration option is supported for peer:

- **`map`**
  Allows mapping between the system and database user names.

Peer authentication can only be used on operating systems such as Linux that provides `getpeereid()`, `SO_PEERCRED` (socket parameter), and/or similar mechanisms.

LDAP Authentication

This authentication works like a `password` except when you use LDAP as a password verification method. LDAP is used only when verifying username/password pairs. This means the user must exist in the database before applying LDAP for authentication.

LDAP authentication can be performed in two modes. The first mode (simple binding mode) is to bind the server to a unique name consisting of `prefix username suffix`. Typically, the `prefix` parameter is used to specify `cn=` or `DOMAIN\` in an active directory environment; `suffix` is used to specify the rest of the active directory environment.

In the second mode (search+binding mode), the server first binds to the LDAP directory using the specified and fixed username and password via `ldapbinddn` and `ldapbindpasswd`, and then searches for users who are trying to log in to the database. If no user and password are set, anonymous binding to the directory is attempted. The search is performed in the subtree of `ldapbasedn` and what matches `ldapsearchattribute` exactly are retrieved. If a user is found in this search, the server disconnects the connection and re-binds to the directory as the user using the password specified on the client to verify if the login is correct. This mode is the same as the one used in the LDAP authentication schema of other software, such as Apache `mod_authnz_ldap` and `pam_ldap`. While LDAP authentication works more flexibly when the user objects are in the directory, it disconnects the two LDAP servers.

The following configuration options are used in both modes.

- **`ldapserver`**
  The name or IP address of the LDAP server to connect to. You may specify multiple servers separated by spaces.

- **`ldapport`**
  The port number of the LDAP server to connect to. If no port is specified, the default port setting of the LDAP library will be used.

- **`ldaptls`**
  If set to 1, AgensGraph is connected to the LDAP server using TLS encryption. This encrypts only traffic to the LDAP server. The connection to the client will remain unencrypted unless you use SSL.

The following options are used only in the simple binding mode.

- **`ldapprefix`**
  A string to append before a user name in the DN binding when performing simple binding authentication.

- **`ldapsuffix`**
  A string to append after a user name in the DN binding when performing simple binding authentication.

The following options are used only in the search+bindingmode.

- **`ldapbasedn`**
  Root DN at which a user search is initiated when performing search+binding authentication.

- **`ldapbinddn`**
  User DN that binds to the directory to perform a search during search+binding authentication.

- **`ldapbindpasswd`**
  User's password that binds to the directory to perform a search during search+binding authentication.

- **`ldapsearchattribute`**
  Attribute that matches the user name in the search when performing search+binding authentication. If no attribute is specified, the uid attribute is used.

- **`ldapurl`**
  RFC 4516 LDAP URL. This is another way to write some of the other LDAP options in a more compact standard format. The default is:

```
ldap://host[:port]/basedn[?[attribute][?[scope]]]
```

  `scope` should be specified as `base`, `one`, or `sub` (usually the latter). Only one attribute is used, and other settings for standard LDAP URLs such as filters and extensions are not supported. For non-anonymous bindings, `ldapbinddn` and `ldapbindpasswd` should be specified as separate options.

  In order to use an encrypted LDAP connection, the option `ldaptls` should be used in addition to `ldapurl`. `ldaps` URL schema (direct SSL connection) is not supported.

  LDAP URLs are currently supported only by OpenLDAP, not Windows.

It is an error to mix the configuration options of simple bindings with the options of search+bindings.

An example of a simple binding LDAP configuration:

```
host ... ldap ldapserver=ldap.example.net ldapprefix="cn=" ldapsuffix=", dc=example, dc=net"
```

If the database user `someuser` is asked to connect to the database server, AgensGraph tries to bind to the LDAP server using DN `cn=someuser` and `dc=example`, `dc=net`, and the password provided by the client. If the connection is successful, database access is allowed.

An example of a search+binding configuration is as follows:

```
host ... ldap ldapserver=ldap.example.net ldapbasedn="dc=example, dc=net" ldapsearchattribute=uid
```

If the database user `someuser` is asked to connect to the database server, AgensGraph tries to bind to the LDAP server anonymously (because `ldapbinddn` is not specified) and performs a search for (uid=someuser) under the specified base DN. If an entry is found, it attempts to bind using the discovered information and the password provided by the client. If the secondary connection is successful, database access is allowed.

The same search+binding configuration with URL is as follows:

```
host ... ldap lapurl="ldap://ldap.example.net/dc=example,dc=net?uid?sub"
```

As other software supporting authentication for LDAP uses the same URL format, it gets easier to share settings.

RADIUS Authentication

This authentication works like a `password` except when you use LDAP as a password verification method. RADIUS is used only when verifying username/password pairs. This means the user must exist in the database before applying RADIUS for authentication.

If you are using RADIUS authentication, an Access Request message is sent to the configured RADIUS server. This request is an `Authenticate Only` type and contains parameters for `user name`, `password` (encrypted), and `NAS Identifier`. The request is encrypted using secret that is shared with the server. The RADIUS server sends back either `Access Accept` or `Access Reject`. The RADIUS accounts are not supported.

The following configuration options are supported for RADIUS.

- **radiusserver**
  The name or IP address of the RADIUS server to connect to. This parameter is required.

- **radiussecret**
  Shared secret used when communicating with a RADIUS server while maintaining security. Its values on AgensGraph and RADIUS servers should be identical. A string of at least 16 characters is recommended. This parameter is required.

- **radiusport**
  The port number of the RADIUS server to connect to. If no port is specified, the default port “1812” is used.

- **radiusidentifier**
  A string is used as a NAS identifier in RADIUS requests. For example, this parameter can be used as the second parameter in the RADIUS server by identifying the database user to be authenticated. If no identifier is specified, the default postgresql is used.

Certificate Authentication

As this method performs authentication using an SSL client certificate, it is available only for SSL connections. When you apply this authentication method, the server requires the client to provide a valid certificate. The password prompt is not sent to the client. The `cn` (common name) attribute of the certificate is compared to the requested database user name; if they match, the login is allowed. You may use the user name mapping to set `cn` different from the database user name.

The following configuration option is supported for SSL certificate authentication.

- **map**
  Allows mapping between the system and database user names.

PAM Authentication

This authentication works like a `password` except when you use Pluggable Authentication Modules (PAM) as an authentication mechanism. The default PAM service name is `postgresql`. PAM is used only when verifying username/password pairs. This means the user must exist in the database before applying PAM for authentication.

The following configuration option is supported for PAM.

- **pamservice**
  PAM service name.

Authentication Problems

Authentication failures and relevant issues are typically revealed through error messages as follows:

```
FATAL:no pg_hba.conf entry for host "123.123.123.123", user "andym", database
 "testdb"
```

This means connection with the user is possible. As indicated in the message, the server refused the connection request because it failed to find a match in the `pg_hba.conf` configuration file.

```
FATAL:password authentication failed for user "andym"
```

This message means that the user has contacted the server and connection with the user is possible, but the user should pass the authentication method specified in the `pg_hba.conf` file. The password entered by the user should be examined, or if there is a problem with this authentication type, the user's Kerberos or ident software should be checked.

```
FATAL:user "andym" does not exist
```

It failed to find the database user name shown.

```
FATAL:database "testdb" does not exist
```

The database that you want to connect to does not exist. If you do not specify a database name, the database user name is assumed to be the database name.

Role

This chapter describes how to create and manage roles. Since AgensGraph was developed based on PostgreSQL, the PostgreSQL technical documents were quoted when introducing the features of the RDB (not GDB).

AgensGraph uses the concept of "role" to manage database access permissions. Roles can be considered as a database user(s) or a group of database users, depending on how the database is set up. A role may own a database object (e.g. table) and control who can access the object by assigning permissions for the object to other roles. As the membership of a role can be granted to other roles, you may make the member role use the permissions assigned to other roles.

Database Role

The concept of database roles is completely different from operating system users. Maintaining a correspondence relationship between them may be convenient but is not necessary. Database roles are global between database cluster installations (not database-specific). To create a role, you must use CREATE ROLE SQL command.

```
db=# CREATE ROLE name;
```

`name` complies with the rules of SQL identifiers (no special characters or double quotation marks are allowed). To delete an existing role, use DROP ROLE command.

```
db=# DROP ROLE name;
```

For convenience, the programs createuser and dropuser, which can be invoked from the shell command line, are provided as the wrapper for SQL commands.

```
$ createuser name
$ dropuser name
```

To determine existing role sets, examine the pg_roles system catalog. For example:

```
db=# SELECT rolname FROM pg_roles;
```

The psql program's `\du` meta command is also useful for listing existing roles.

Role Attributes

A database role has many attributes that define permissions and interacts with the client authentication system.

- **Login Privilege**
  You may only use roles with the `LOGIN` attribute as the initial role name for database

connections. A role with the `LOGIN` attribute can be considered the same as a "database user." To create a role with a login privilege, use one of the followings:

```
CREATE ROLE name LOGIN;
CREATE USER name;
```

(`CREATE USER` differs from `CREATE ROLE` in that it gives a `LOGIN` privilege by default.)

- **Superuser Status**
  The database superuser skips all permission checks except for a login privilege. As the superuser authority can be risky, it should not be used carelessly. It is recommended that most of the tasks are processed by roles other than superuser. To create a new database superuser, use `CREATE ROLE name SUPERUSER`; a role of superuser should do this job.

- **Database Creation**
  To create a database, the required privilege should be explicitly given to the role (except for superuser, which skips all privilege checks). To create this role, use `CREATE ROLE name CREATEDB`.

- **Role Creation**
  To create an additional role, the required privilege should be explicitly given to the role (except for superuser, which skips all privilege checks). To create this role, use `CREATE ROLE name CREATEROLE`. A role with the `CREATEROLE` privilege can change and delete other roles, and grant or revoke membership. However, superuser status is required if you want to create, alter, delete, or change the membership of the superuser role; `CREATEROLE` cannot do this job.

- **Initiating Replication**
  To initiate streaming replication, the corresponding privilege should be explicitly given to the role (except for superuser, which skips all privilege checks). The role used for streaming replication must have the LOGIN privilege always. To create this role, use the `CREATE ROLE name REPLICATION LOGIN`.

- **password**
  Passwords are important only in the case of a client authentication method that requires the user to enter a password when connecting to the database. `password` and `md5` authentication methods use passwords. Database passwords are distinct from the operating system user passwords. When creating a role, a password can be specified using `CREATE ROLE name PASSWORD 'string'`.

Role Membership

Users may be grouped for convenience of authority management. This allows you to grant or revoke privileges on a group-by-group basis. In AgensGraph, this is done by creating a role that represents a group, then assigning membership of the group role to individual user roles.

To set up a group role, you should first create a role.

```
CREATE ROLE name;
```

In general, roles used as groups do not have the `LOGIN` attribute; you may set it if necessary.

If a group role exists, you can add and delete members using the GRANT and REVOKE commands.

```
GRANT group_role TO role1, ... ;
REVOKE group_role FROM role1, ... ;
```

You can also grant membership to other group roles (there is no actual distinction between group and non-group roles). The database does not allow you to set a circular membership loop and/or to grant role membership to `PUBLIC`.

Members of a group role can use the permissions of the role in two ways. First, all members of the group perform SET ROLE explicitly to "become" a group role temporarily. In this state, the database session has access to the group role (not the original login role), and the created database object is considered owned by the group role, not the login role. Second, the member role with `INHERIT` attribute automatically has the permissions of the role as a member, including all permissions inherited from that role. For example, suppose you have executed the following:

```
CREATE ROLE joe LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;
CREATE ROLE wheel NOINHERIT;
GRANT admin TO joe;
GRANT wheel TO admin;
```

Immediately after the database session connects to `joe` role, `joe` "inherits" the permissions of `admin` (in addition to those granted directly to joe) and is able to use them as well. `joe` is, indirectly, a member of `wheel`. As this membership, however, is given via admin with NOINHERIT attribute, the permissions granted to wheel cannot be used.

```
SET ROLE admin;
```

When you execute the above command, the session uses only those permissions granted to admin and does not use the permissions granted to `joe`.

```
SET ROLE wheel;
```

When you execute the command above, the session uses only the permissions granted to `wheel`, not those granted to `joe` or `admin`. The original privilege state is restored using one of the following:

```
SET ROLE joe;
SET ROLE NONE;
RESET ROLE;
```

`LOGIN` (role attribute), `SUPERUSER`, `CREATEDB`, and `CREATEROLE` can be thought of as special permissions, but they are not inherited as routine permissions of a database object. In order to use them, you should actually `SET ROLE` to a specific role holding one of these attributes. In the example above, it is also possible to grant `CREATEDB` and `CREATEROLE` to the `admin` role. However, a session connecting to joe role cannot have these permissions immediately as they will be granted only after executing `SET ROLE admin`.

You should use DROP ROLE to drop group roles.

```
DROP ROLE name;
```

Membership of the group role is automatically revoked (the member role is not affected though). Note that the objects owned by the group role should first be deleted or reassigned to another owner, and the permissions granted to the group role must be revoked.

Backup & Recovery

Like in other database systems, backup is critical in AgensGraph as well. This is because most of the important data for services in general are stored in databases. The backup and restore process is relatively simple, and we will outline the technical and conceptual aspects of the process for a better understanding. Since AgensGraph was developed based on PostgreSQL, the PostgreSQL technical documents were quoted when introducing the features of the RDB side (not GDB).

This chapter describes file system-based backups and archive-mode backups.

Backup

File system level backup

This is a way of backup at the file system level by directly copying the entire data storage space. It is not much different from the file system backup done by general operating systems. For instance:

```
tar -cf backup.tar /usr/local/pgsql/data
```

Backup with a command above is possible. However, as there are two limitations as follows to this approach, this is less useful in practical terms than using pg_dump command.

1. You must suspend the database server for normal backup operations. Blocking all client connections and copying the file system does not guarantee a safe backup. (This is because neither tar command nor any other similar tools provide a consistent snapshot of the file system state and can accurately reflect information about the internal buffering used by the server). Likewise, when restoring, the server must be in a stopped state.

2. If you know the locations and intended uses of various files in the database well, you may think that you will be able to copy/use only the files that correspond to individual databases or tables. Even if it is true, normal operation is not possible without commit log files. pg_clog/* files store the commit status of all transactions. Since these log files are processed as database clusters (not as tables), partial copying is impossible; even if you copy physical files and log files of individual tables, they may be related to other tables.

Another thing to consider in the file system-based backups is that if the file system provides a “consistent (and reliable) snapshot” feature, you should be careful when using the feature. A typical backup method using the file system snapshot feature: i) create a “frozen snapshot”; ii) copy all the data cluster files (as mentioned above, there is no point in back up partial files); iii) then unlock the locked snapshot. If the file system provides this

feature, backup while the database server is running is possible. At this time, as backup is done while in operation, the database files are saved without properly closing the database server. When you start the database server with backup data, it will recognize the old server as a conflict and replay the WAL log. However, if you run the CHECKPOINT command before starting and keep the WAL logs generated in the course of the backup operation separately, the execution will be done without a big issue. (You may, of course, ignore the WAL logs.)

The problem is that the database server uses multiple file systems. For example, if you create a tablespace that uses different partitions, you should be able to create a "frozen snapshot" of all the file systems used by the database server at the same time. To generate backups in this way, please read through the filesystem documentation carefully and conduct backups after you weigh the benefits against the risks involved.

If it is not possible to create multiple file system snapshots simultaneously, you must stop the database server while creating the snapshot.

Another way is to use a rsync application. The work proceeds in two steps. When the server is running, it synchronizes all the data first. Then, it stops the server and synchronizes data once again. This will minimize server downtime because only the changed files will be synchronized while the server is stopped.

File system backups typically require more backup space than SQL dumps. (This is because dump files created by pg_dump do not have indexes, physically, and only contain commands to make the indexes.) For this reason, the restore operation can be done much faster in file system backups.

Archive mode backup

Creating an archive of Write Ahead Logging (WAL)

Internally, the AgensGraph system constantly creates WAL records in manipulating the database. Those records are divided into WAL segment files so they can be stored in physical disk space. A single WAL file is basically a 16MB file (this size is determined when you compile the server). The filename uses the corresponding number in the WAL order. If you set it not to create a WAL archive file, only a few of these files will be created, and you will need to find and "reuse" any log files that you no longer use. Internally, it finds the status information of WAL records, reuses it by changing the checkpoint operation to a no-longer-in-use state, and recording a new WAL record in its place.

If you create a WAL archive file, you archive existing WAL record information elsewhere (same host, NFS mount point, or even tape) before reusing a certain WAL segment file. The archiving method is specified directly by the administrator. Let the administrator decide even in the case where there is already a file with the same name. However, when reusing the WAL segment file, the AgensGraph server only performs operations according to the "method of archiving" set by the administrator. You may simply use the cp command for the archive, use a more complex custom shell script, or use a backup solution command. This part is entirely up to the administrator.

To create a WAL archive file, specify archive (or hot_standby) with the value of the configuration parameter wal level. Next, set the environment configuration parameter archive mode to on, and the environment configuration parameter archive command to an appropriate system command. All of these settings are configured in the postgresql.conf file. You can use %p (absolute path to the WAL log file) and %f (the name of the log file to be archived) reserved arguments for the shell command to specify as the archive_command value. If you need to type % literally, type %%. In general, this setting is used in the following format.

```
# Unix
  archive_command = 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server
/archivedir/%f'
# Windows
  archive_command = 'copy "%p" "C:\\server\\archivedir\\%f"'
```

The above configuration simply copies the WAL segment file to the /mnt/server/archivedir directory (with the same file name). The above is only an example for Unix and Windows operating systems. Actual settings should be changed to match the operating system. %p %f reserved arguments are changed as follows for the actual execution of the command.

```
test ! -f /mnt/server/archivedir/000000010000000A900000065 &&
cp pg_xlog/000000010000000A900000065\ /mnt/server/archivedir/000000010000000A90
0000065
```

This will always make the WAL log file fresh at a certain location.

This archive command is executed by the privileges of the system user who ran the AgensGraph server. So, just like processing WAL segment files, you should consider security policies when processing archive files. It should be noted that a fatal security incident can occur if it can be read, overwritten, and deleted by anyone.

Another thing to note is that the archive command's shell execution return value should be 0 (zero) if the command is successful, or a different value if the command is not successful. The server will determine if the log file has been successfully archived or failed with this return value, and if successful, will either erase or reuse the original log file; if unsuccessful, it will retry until it succeeds.

You should not overwrite files that already exist with this archive command. Overwriting may cause unintended errors in the restore operation. (This file may already have been created or used by another database.) It is safest for the administrator to manually process the log file because constant errors may occur if it already exists.

It is recommended to check if the file already exists before you archive a log file. If it exists, set the command to return a non-zero value. On Unix, a test command is provided for this task. On some Unix platforms, the -i option is provided in the cp command to avoid overwriting files that already exist. You must check the return value when using this command. (The GNU cp command returns "0" if there is a log file, which is not desirable.)

When you save a WAL file separately, operational jobs may need to be aborted manually, or saving operation may fail repeatedly because of insufficient storage space. For example, if you want to archive a WAL segment file to a tape storage device and the device does not automatically exchange tapes and there is no free space on the tape; then, the user continues to run the file write operation until an error occurs and may repeat the process. In such a case, the `pg_xlog/` directory will not delete or reuse the WAL segment file because the data has not been safely copied to the tape storage, and the new WAL segment files created later will continue to accumulate and eventually stop as there will be no more free space in the `pg_xlog/` directory and a PANIC error will occur in the server.

You also have to worry about the data recording speed of the WAL file when it is stored separately. Even if the work is properly done, the same problem may occur if the speed at which the WAL file is created is faster than the speed at which the file is kept separately. If the `pg_xlog/` directory has enough free space, it will not be a problem, of course. Before using this feature, you should consider this kind of problem in this section sufficiently; the administrator should monitor whether this function works as intended as well.

The path to the save file should be up to 64 ASCII characters. The file name must use the reserved word %f (i.e. only directories can be changed). The original WAL segment file must use the reserved word %p. Note that changes to the `postgresql.conf`, `pg_hba.conf`, and `pg_ident.conf` files are not reflected in the database to be restored by this backup

method because the WAL file contains only transactional operation information. You should back up these configuration files according to your system's normal file backup policy.

The archive command is executed for files that are all reflected on the server (rollbacked or committed and then checkpointed) among the WAL files. That is, for a database with very small workloads, the interval at which archive commands are executed is very long. If there is a database failure at this interval, data loss occurs. Thus, you must forcefully use the archive command to save this segment file separately after a certain time before all the content of the WAL segment file is processed. If the setting value is too short, the disk space can be wasted (a disadvantage). Typically, the setting in minutes is appropriate. You may also force the user to change the segment file using the `pg_switch_xlog` function. This function is typically used when bulk data entry, modification, or deletion occurs and immediate backups are required.

If you set the `wal_level` value to minimal, you will not be able to save this WAL log and restore it based on it. This is because, if you use this setting, you do not keep the restore-related information in the WAL file. Therefore, when you change the `wal_level` setting, you must restart the server. In the case of archive_command setting, however, reloading the configuration files is sufficient. Stopping this operation during operation may be needed. To do that, specify the archive_command setting as an empty string (''). The archive_command setting will remain in the `pg_xlog/` directory until you copy the WAL file again.

Recovery

Recovering Using a Continuous Archive Backup

Describes how to recover a database server using backups archived in the archive mode backup method when a problem occurs. The order of operation is as follows:

1. Stop the server first if it is running.
2. If the system has enough disk space, copy the entire database cluster directory, any associated custom tablespace files, and any required WAL segment files to a temporary location. To run this on a system that was already running, you would need at least twice as much disk space. If you do not have such free space, at least save the pg_xlog files and server configuration files in the cluster directory of your existing database server. If you save the WAL segment files that have been and have not been backed up, they can be restored to the state just before the server was stopped.

3. Delete both the data cluster directory for the existing server and the directory for the existing tablespace.

4. Copy the backup file to its original location. The user at this time must be a system user running the database server. (If you are working as root, you must change it to the owner!) Thus, you need to set the file access permissions and owner to match the system user. When using the next user-defined tablespace, copy the source directory and any files in it that are symbolic links in the pg_tblspc/ directory.

5. Leave the `pg_xlog/` directory empty. If there are any files in this directory, they should be removed. The necessary files are automatically created in this directory during the recovery process. If you have excluded the `pg_xlog/` directory from backup, create a directory or a symbolic link to make the directory available to the database server. Again, you need to make sure that the directory permissions and owner are only available to the system user running the database server. If it is a symbolic link, you must make sure that it is the same as the source path used in the previous database so that you can overwrite the previous data.

6. If you do not have a backup copy of the WAL segment file, copy the pg_xlog/ files you copied in step 2 to the location you want to restore. (copying, not relocating, is recommended. Copying is safe because recovery is not complete yet.)

7. Create recovery.conf in the database cluster directory. It is also a good idea to temporarily modify pg_hba.conf so that it is not accessible from the outside during the recovery process.

8. Run the server. The server runs in recovery mode and finds the required WAL files and begins to process the unrecovered transactions in batches. If the server hangs due to external impact during the recovery operation, simply resume the server and continue the recovery operation. At the end of the recovery process, the server changes the name of recovery.conf into recovery.done and waits for the client to connect to it in the normal run state; this is to prevent it from being rerun in recovery mode.

9. Now connect to the database to see if the data is normal; if not, review the server logs and go back to step 1. If all the data and database status are normal, modify the content of pg_hba.conf and allow external access.

The key to this recovery method is how to apply the backup WAL segment file to the database and until what point the restore should be made. This is specified in recovery.conf. A simple way to create this file is to first copy recovery.conf.sample into the share/ directory of the database distros and modify only the necessary parts. The essential part of recovery.conf is restore_command. This setting is a definition of how to apply the backup

WAL segment file in AgensGraph. In short, you can specify a command that is the opposite of archive_command in the server environment settings. The reserved words used here are the same as when specifying the archive_command setting value. The %f value is the file name in the backup archive directory, and the %p value is replaced with a transaction log file. (If relative paths are used, they are processed relative to the current directory in which the server is running.) Reserved word %% is treated as % characters. The following are typical settings.

```
restore_command = 'cp /mnt/server/archivedir/%f %p'
```

The above configuration is used to restore the WAL segment files that have been previously backed up in the /mnt/server/archivedir directory. Of course, this command can be much more complicated depending on the backup device you are using. You can also create and use a shell script that includes some commands directly (e.g. mounting a tape and moving it to the desired location like a tape backup device). The important point here is that if the operation specified here fails, the return value of the operation must be non-zero. This operation also requests files that are not in the archive backup directory. The result of this operation must return a non-zero return value (this is not an error). If there is no command in the script, or if the server is shut down normally and a signal other than SIGTERM signal (as part of the recovery process) stops the operation, that is considered an error. In such a situation, the recovery operation is stopped, and the server stops operating as well.

The server is able to find a file ending with .backup or .history. If there is no file the server is searching for, it should tell the server that the file does not exist (by returning a non-zero value). The %f filename and the %p filename copied to the server are not always the same. For this reason, you should be careful not to make mistakes while processing this part when you create and use your own shell script.

If you cannot find the WAL segment file in the archive backup directory, look in the pg_xlog/ directory inside the data cluster. If there are more WAL segment files to be processed, the files will be processed even if they are not backed up. However, if the file with the same filename already exists in the archive backup directory, that file will be applied, the file in pg_xlog/ will be ignored, and the new WAL segment name will reflect that log. For this reason, you must clean up the WAL segment files before the recovery process.

Typically, the recovery job reflects the last log file in the archive backup directory. As it finds until there is no next file, the last of the recovery log is printed as a “file not found” message. You can also find the file of 00000001.history when you start the recovery

process. All of these are logs that occur during a normal recovery process.

The recovery operation can also be stopped at any point by specifying a recovery stop point in recovery.conf. This point is called the “recovery target.” This feature is useful for an inexperienced database administrator to help recover from operational errors, such as accidentally deleting critical tables. The “recovery target” can be specified at a specific time, or can be any string specified by the administrator. It can also be specified with a specific transaction ID. If you cannot use a tool to determine at which transaction ID an accident occurred, simply specify a specific time or a predefined string specified by the administrator for recovery.

> The point at which recovery is stopped should be after the base backup time. It should be specified to be later than the time when pg_stop_backup command was executed. You cannot use the base backup to back up the data cluster files to the file system level and restore them to the time they were backed up. If you need to do this, you need the old base backup data and the corresponding WAL backup files.

If there is a problem with the WAL file itself, it only applies to transactions that have been processed normally; both the recovery operation and server stop. In this case, it is necessary to identify the time when the problem occurs and restore it from the beginning to the point of “recovery target.” If the operation is stopped due to external influences during the recovery operation, the cause of the problem must be solved. Then, simply rerun the server and continue the recovery process. The recovery operation is re-executed together with a checkpoint operation in the normal execution environment. Internally, the pg_control file is periodically updated and already-reflected logs are no longer reworked. This means there is no need to worry about duplicate processing.

Configuration

This chapter describes the meanings of environment variables that affect operations of database systems and how to set them.

Configuration Settings Reference

Setting parameters

All parameter names are case-sensitive. Each parameter value is one of five types: Boolean (or string), integer, floating point, and enumerated (enum). The data type sets the syntax for setting parameters.

- **Boolean**: You can set the value to one of "on, off, true, false, yes, no, 1, 0 (case insensitive)" or one of "t, f, y, n".
- **String**: In general, single quotation marks are placed at both ends; if a value itself contains a single quotation mark, add one more single quotation mark. Quotation marks can be omitted if the value is a common, simple number or identifier.
- **Numeric** (integer and floating point): The decimal point is allowed only when it is a floating-point parameter. Do not use thousands of separators (e.g. ',' in 1,000,000). Quotes are unnecessary.
- **Numeric with Unit**: Some numeric parameters have implicit units because they are used to describe memory or time. Units can be in kilobytes, blocks (usually 8 kilobytes), milliseconds, seconds, and minutes. Among these settings, numeric values without units use the basic unit of setting, which can be found in pg_settings.unit. For your convenience, the setting can specify explicitly-specified units. For instance, if the time value is "120 ms", the actual unit of the parameter will be converted to "ms." Note that the value must be written in string (including quotes) to use this feature. Unit names are case-sensitive, and there can be a space between the numeric value and the unit.
  - Valid memory units are kB (kilobytes) and MB (megabytes), GB (gigabytes), and TB (terabytes). The memory multiplier is 1024 (not 1000).
  - Valid time units are ms (milliseconds), s (seconds), min (minutes), h (hour), and d (days).
- **Enumerated**: Parameters of "enumerated" type are created in the same way as string parameters, but are limited to a single set of values. The allowed values of these parameters can be found in pg_settings.enumvals. Enum parameter values are not case-sensitive.

Parameter Interaction via the Configuration File

The most basic way to set these parameters is to edit postgresql.conf, which is usually in the data directory. If the database cluster directory is initialized, the default copy is installed. An example similar to this file is as follows:

```
# This is a comment
log_connections = yes
log_destination = 'syslog'
search_path = '"$user", public'
shared_buffers = 128MB
```

A parameter per line is specified. The equals sign between name and value is optional. Blank spaces are not important (except for the quoted parameters) and blank lines are ignored. The hash mark (#) means that the rest of the line is a comment. Simple identifiers or non-numeric parameter values should use single quotes. To include a single quotation mark in the parameter value, you must add one more single quotation mark, or use backslashes and quotation marks.

The parameters configured like this are provided in the cluster by default. The settings visible in the active session are these values unless you override them. The following section describes how an administrator or user overrides these defaults.

Each time the main server process receives a SIGHUP signal, the configuration file is read again. Executing `pg_ctl` reload on the command line or calling SQL function `pg_reload_conf ()` sends SIGHUP. The main server process also spreads this signal to all server processes that are currently running, allowing the new value to be applied to the existing sessions (applied after the currently-running client command is complete). Alternatively, the user may send signals directly to a single server process. Some parameters can only be configured when the server starts. Changing the entries of the configuration file will be ignored until the server restarts. Likewise, incorrect parameter settings in the configuration file are also ignored during the SIGHUP process (but recorded in the log).

In addition to postgresql.conf, the AgesnGraph data directory ($AGDATA) contains postgresql.auto.conf that has the same format as postgresql.conf; postgresql.auto.conf should not be edited directly. This file contains the settings provided by the ALTER SYSTEM command. Every time postgresql.conf is present, this file is automatically read and the corresponding settings are applied in the same manner. The settings in postgresql.auto.conf take precedence over those in postgresql.conf.

Parameter Interaction via SQL

AgensGraph provides three SQL commands for setting configuration defaults. The ALTER SYSTEM command mentioned above provides a way to change global defaults using a SQL syntax, which is equivalent to editing postgresql.conf in function. In addition, there are two commands that enable default settings for each database or role.

- ALTER DATABASE: Overrides global settings by the database.
- ALTER ROLE: Overrides both global and database-specific settings with custom values.

The values set using ALTER DATABASE and ALTER ROLE shall apply only when starting a new database session. This overrides the value obtained from the configuration file or server command line and applies the default for the rest of the session. As some settings cannot be changed after starting the server, configuration using this command (or one of those listed below) is not possible.

When a client connects to the database, it provides two additional SQL commands (or equivalent functions) that can interact with the AgensGraph session-local configuration settings.

- You can check the current values of all parameters with the SHOW command. The corresponding function is current_setting (setting_name text).
- SET command allows you to modify the current value of the parameter, which can be set locally in the session. It does not affect other sessions. The corresponding function is set_config (setting_name, new_value, is_local).

The system view pg_settings can also be used to check and change session-local values.

- View query is similar to SHOW ALL, but shows more detailed results. It is also more flexible as it enables you to specify filter conditions or join with other relations.
- In this view, using UPDATE to update the setting column is equivalent to executing the SET command.
  For example:

```
SET configuration_parameter TO DEFAULT;
```

The above syntax is identical with the one below:

```
UPDATE pg_settings SET setting = reset_val WHERE name = 'configuration_parame
ter';
```

Managing Configuration File Contents

AgensGraph provides features to subdivide the complicated postgresql.conf into smaller files. Though the configuration methods of the features are not identical, they are useful especially for managing related servers.

In addition to setting individual parameters, postgresql.conf has “include, a directive. It specifies another file to read and process as it as if the file is inserted into the configuration file. This feature physically divides the configuration file. Here is a simple example of include:

```
include 'filename'
```

If the filename is not absolute, it is processed as a relative path to the configuration file directory it references. “include” can be nested.

There is also a directive “include_if_exists”, which works the same as the include directive except when the reference file does not exist or the file cannot be read. “include” treats this as an error condition, but include_if_exists simply logs the message and continues to process the referenced configuration file.

postgresql.conf may also include include_dir, which specifies the path to the configuration file to include as follows:

```
include_dir 'path'
```

If it is not an absolute path, it is processed as a relative path to the referenced configuration file directory. Files that are not directories within the specified directory are included only when their names end with .conf. As such files may be hidden on some platforms, file names that start with `.` are also ignored to prevent mistakes. Files in the include directory are processed in the file name order (according to the C locale conventions; for example, numeric-alphabetical order and uppercase-lowercase order).

Rather than using a single postgresql.conf file, “include” files or directories can be used to logically separate database configuration. Imagine a company that runs two database servers with different memory capacities. In the case of logging, there are likely to be configuration elements shared by two databases. However, the two servers are likely to have different/customized memory-related parameters. The way to manage such a situation is to split the changed customized configuration content into three files. Users may include each file by adding the following code to the end of the postgresql.conf file:

```
include 'shared.conf'
include 'memory.conf'
include 'server.conf'
```

The shared.conf file on all systems is identical. Each server with a different memory size can share the same memory.conf. Users are able to manage both 8GB RAM and 16GB RAM servers as a single file. Server-specific configuration information is included in server.conf.

You may also create a configuration file directory and add this information into the file. For example, the conf.d directory can be referenced as the last entry in postgresql.conf.

```
include_dir 'conf.d'
```

Then, specify the filename of the conf.d directory as follows:

```
00shared.conf
01memory.conf
02server.conf
```

This naming convention clarifies the order of file loading. When the server reads the configuration file, only the last parameter setting is applied. In this example, the values are set in conf.d/02server.conf override the values set in conf.d/01memory.conf.

You can use this method for a descriptive naming of files.

```
00shared.conf
01memory-8GB.conf
02server-foo.conf
```

This placement order assigns a unique name for each configuration file change. By doing so, reduces ambiguity when several server configurations are stored in a single location (e.g. the version control repository).

File Locations

By default, the three configuration files (`postgresql.conf`, `pg_hba.conf`, and `pg_ident.conf`) are stored in the data directory of the database cluster, and the configuration file can be placed in a different location using the following parameters: By keeping configuration files separate, management and correct backup of configuration files can be done in an easier manner.

- **data_directory** (string)
  Specifies a directory to store the data. This parameter can be set only at server startup.

- **hba_file** (string)
  Specifies a configuration file for host-based authentication (pg_hba.conf). This parameter can be set only at server startup.

- **ident_file** (string)
  Specifies a configuration file for user name mapping (pg_ident.conf). This parameter can be set only at server startup.

- **external_pid_file** (string)
  Specifies the name of an additional process ID (PID) file that must be created by the server for use by a server management program. This parameter can be set only at server startup.

The above parameters are not explicitly set in the default installation. The data directory is specified as the `AGDATA` environment variable, and all configuration files can be found in the data directory.

To keep a configuration file somewhere other than the data directory, the `AGDATA` environment variable must contain the configuration file and `data_directory` parameters should be set in `postgresql.conf` so that the data directory is physically located. `data_directory` redefines the data directory for the locations of data directories (not the locations of configuration files).

If desired, the name/location of the configuration file can be specified individually using the `config_file`, `hba_file`, and `ident_file`, and can be set within the default configuration file (`postgresql.conf`). It is not necessary to specify `AGDATA` if three parameters and `data_directory` are explicitly set.

When setting these parameters, the relative path is interpreted relative to the directory where Agens starts.

Resource Consumption

Memory

- **shared_buffers** (integer)

  - Set the amount of memory used by the database server for shared memory buffers. The default value is typically 128 megabytes (128MB), but this value may not be reached if it is not supported by the kernel configuration (to be determined during initdb). This setting should be at least 128 kilobytes. (The BLCKSZ value (not default) changes the minimum value.) Settings much larger than the minimum value are used when good performance is needed.

  - If you are using a dedicated database server with more than 1GB of RAM, the appropriate starting value for shared_buffers is 25% of the system memory. Workloads are more effective with larger settings of shared_buffers. However, as AgensGraph also depends on the operating system cache,

allocating more than 40% of RAM to shared_buffers is not recommended for system efficiency. If you set a larger shared_buffers to run a large amount of new or changed data write processes over a long period of time, you should also increase the setting accordingly in checkpoint_segments.

  - If the system RAM is less than 1GB, you need adequate space for the operating system. Thus, making the RAM ratio smaller is the right decision. Increasing the shared_buffers value on Windows is not effective. You may get better results by making this setting smaller, and by making the cache relatively large for the operating system. The useful range for shared_buffers on Windows systems is 64MB to 512MB.

- **huge_pages** (enum)
  - Sets the Huge page to Active/Inactive. Valid values are tried (default) and on, off.
  - Currently, this feature is only supported on Linux. If it is set to try, it is ignored in other systems.
  - Using huge pages results in higher performance as it consumes smaller page tables and less CPU for memory management.
  - If you set huge_pages to try, the server will use huge pages. However, if it fails, it will fall back to using normal allocation. If set to on, the server will not start if it fails to use huge pages. If off, it does not use huge pages.
- **temp_buffers** (integer)
  - Sets the maximum number of temporary buffers used by each database session. There is a session-local buffer that is used only to access temporary tables. The default value is 8 megabytes (8 MB). Settings can be changed within an individual session, but only before the first use of the temporary tables in a session. Subsequent changes to the value will not take effect in the session.
  - The session allocates temporary buffers up to the limit set in temp_buffers. In practice, the cost for setting a large value in a session that does not require a lot of temporary buffers amounts to a buffer descriptor or only about 64 bytes each time temp_buffers increases. However, if the buffer is actually used, it requires an additional 8192 bytes (or typically BLCKSZ bytes).
- **max_prepared_transactions** (integer)

  - The maximum number of transactions that can be in a “ready” state at the same time.

  - If you do not plan to use prepared transactions, this parameter should be set to “0” to prevent mistakes in creating prepared transactions. If you set max_prepared_transactions to at least max_connections when using prepared transactions, you may let the session hold the prepared transaction.

  - When the standby server is running, it should be set to be greater than or equal to the master server value. Otherwise, the standby server will not allow queries.
- **work_mem** (integer)
  - Specifies the amount of memory to be used by the internal Align command and hash table before writing the temporary disk file. The default value is 4 megabytes (4MB). For complex queries, several Align or hash commands can be executed in parallel. Each command can use the amount of memory specified by this value before writing the data to the temporary file. Sessions that are running may execute this command at the same time as well. The total memory used is a multiple of work_mem. The Align command is used for ORDER BY, DISTINCT, and merge joins. Hash tables are used for hash joins, hash-based aggregations, and hash-based processing of IN subqueries.
- **hash_mem_multiplier** (floating point)
  - Used to compute the maximum amount of memory that hash-based operations can use. The final limit is determined by multiplying work_mem by hash_mem_multiplier. The default value is 1.0, which makes hash-based operations subject to the same simple work_mem maximum as sort-based operations. Consider increasing hash_mem_multiplier in environments where spilling by query operations is a regular occurrence, especially when simply increasing work_mem results in memory pressure (memory pressure typically takes the form of intermittent out of memory errors). A setting of 1.5 or 2.0 may be effective with mixed workloads. Higher settings in the range of 2.0 - 8.0 or more may be effective in environments where work_mem has already been increased to 40MB or more.
- **maintenance_work_mem** (integer)
  - Specifies the maximum amount of memory used in maintenance commands such as VACUUM, CREATE INDEX, and ALTER TABLE ADD FOREIGN KEY. The default is 64 megabytes (64 MB). This command may be executed one at a time in a database session, and a normal installation cannot have multiple concurrent commands. It is safe to set this value to a much larger value than

work_mem. Large settings can improve vacuuming and the database dump recovery performance.

  - If autovacuum is running, it can be allocated in multiples of autovacuum_max_workers in this memory. Thus, do not set the default too high. It may be useful to set/manage autovacuum_work_mem separately.

- **autovacuum_work_mem** (integer)

  - Specifies the maximum amount of memory used by each autovacuum worker process. The default value is -1, which means that maintenance_work_mem should be used instead. This setting does not affect VACUUM when running in a different context.
- **logical_decoding_work_mem** (integer)

  - Specifies the maximum amount of memory to be used by logical decoding, before some of the decoded changes are written to local disk. This limits the amount of memory used by logical streaming replication connections. It defaults to 64 megabytes (64MB). Since each replication connection only uses a single buffer of this size, and an installation normally doesn't have many such connections concurrently (as limited by max_wal_senders), it's safe to set this value significantly higher than work_mem, reducing the amount of decoded changes written to disk.
- **max_stack_depth** (integer)

  - Specifies the maximum safe depth of the server execution stack. The ideal configuration is to set it slightly short of the safety margin specified forcibly by the kernel (as set by ulimit -s or to be equivalent to local). The safety margin is required since only important parts of the potential recursive routines (e.g. expression evaluation), not all routines on the server, are checked for stack depth. The default setting is basically small, 2 megabytes (2MB), with a low probability of collision. However, if the setting is too small, it may be difficult to execute composite functions. Only the superuser may change the setting.

  - If max_stack_depth is set to a larger value than the actual kernel limit, the runaway recursive function may conflict with the backend process. On a platform where AgensGraph is allowed to determine the kernel limit, the server does not allow this variable to be set to an unstable value. However, as not all platforms provide information, you should be careful when choosing values.

- **shared_memory_type** (enum)

- Specifies the shared memory implementation that the server should use for the main shared memory region that holds PostgreSQL's shared buffers and other shared data. Possible values are mmap (for anonymous shared memory allocated using mmap), sysv (for System V shared memory allocated via shmget) and windows (for Windows shared memory). Not all values are supported on all platforms; the first supported option is the default for that platform. The use of the sysv option, which is not the default on any platform, is generally discouraged because it typically requires non-default kernel settings to allow for large allocations.

- **dynamic_shared_memory_type** (enum)
  - Specifies the dynamic shared memory implementation to be used by the server. Possible values include posix (POSIX shared memory allocated through shm_open) and sysv (System V shared memory allocated through shmget), windows (Windows shared memory), mmap (shared memory simulation that uses memory map stored in the data directory), and none (disabled function). Some of these are not supported on certain platforms. The first support option is the default settings of the given platform. Use of non-default mmap options on platforms is not recommended generally. This is because the operating system repeatedly rewrites the modified page to disk, increasing system I/O load. However, in the case you save the pg_dynshmem directory on a RAM disk or disable other shared memory features, debugging can be useful.

Write Ahead Log

Settings

- **wal_level** (enum)
  - wal_level determines the amount of information to be recorded in the WAL. The default is minimal, which records only the information required to recover from a collision or an immediate shutdown. Archive adds only the logging needed for the WAL archive. hot_standby adds more information to the standby server for read-only queries. logical adds necessary information to support logical decoding. Each level includes information logged at a low level. This parameter is set when the server starts.

  - At the minimal level, some bulk-running WAL logging can safely be skipped. This will speed up execution. Executions, where this optimization can be applied, include:

    1. CREATE TABLE AS

2. CREATE INDEX
3. CLUSTER
4. COPY into tables that were created or truncated in the same transaction.

The above executions apply to the tables created in the same transaction or the tables in which records were deleted.

- However, as the minimal WAL does not get enough information to reconstruct the data from the base backup and WAL logs, you must use the archive or higher to perform WAL archiving (archive_mode) and streaming replication.

- Information identical with the archive at the hot_standby level and information needed to reconfigure the state of the running transaction is logged from WAL. To use read-only queries on the standby server, set wal_level to hot_standby or higher on the operating server and enable hot_standby on the standby server. We consider that there is no measurable performance difference between the hot_standby and archive levels. If you find any noticeable change in operations, please feel free to give us your feedback.

- Information that is considered identical with using hot_standby at the logical level and information needed to use a logical changeset are logged from WAL. Using the logical level increases the WAL volume. This is true especially when you set multiple tables to REPLICA IDENTITY FULL and run multiple UPDATE and DELETE statements.

- **fsync** (boolean)

  - If this parameter is on, the AgensGraph server will attempt to verify whether the update has been physically written to disk or not using the fsync () system call or a corresponding method (see wal_sync_method). Accordingly, after a conflict with an operating system or hardware, it is possible to restore the database cluster to a certain state.

  - Turning off fsync has a performance advantage. Data corruption, however, may not be recoverable in the event of a power outage or system failure. It is therefore desirable to turn off fsync only if you can easily rebuild the entire database with external data.

- When turning off fsync is considered safe and when you plan to load a new database cluster from a backup file during initial loading, you can batch-process data from the database to be rebuilt, or can rebuild a read-only database that is used only for failover. It is not desirable for only high-performance hardware to turn off fsync.

- In the case where you disable and re-enable fsync, it is necessary to forcibly move all the changed buffers of the kernel to a durable storage area for recovery reliability. This can be done by executing initdb –sync-only or sync, unmounting the file system, or rebooting the server.

- If you disable synchronous_commit for non-critical transactions, you can have potential performance benefits as it disables fsync without risking data collisions.

- fsync can only be set in the postgresql.conf file or on the server command line. If you disable this parameter, you must consider disabling full_page_writes as well.

- **synchronous_commit** (enum)

  - Specifies whether the transaction commit should be waiting until the WAL record is written to disk before the command returns a “success” indication to the client. Valid values are on, remote_write, local, and off. The default and safety settings are on. If off, there may be a delay between the time when the “success” indication is delivered to the client and the time when it is guaranteed that the transaction is truly secure without a server crash. (The maximum delay is three times wal_writer_delay.) Unlike fsync, setting this parameter to off does not cause database consistency problems. It is true that operating systems or database failures result in partial loss of so-called recently-committed transactions, but the database state appears to be like a clear shutdown of the transaction. Therefore, disabling synchronous_commit can be a useful alternative if performance is more important than certainty for transactional durability.

  - If synchronous_standby_names is set, this parameter also controls whether the transaction commit should wait until the WAL record of the transaction is replicated to the standby server. If set to on, the commit waits until the transaction commit record is received from the standby server and a response from the current synchronous standby server indicates that it has been written to disk. This ensures that transactions are not lost in the case where there is no corruption of the database store on both the production

server and the standby server. When set to remote_write, the commit waits until it receives a response indicating that i) a commit record of the transaction is received and written to the standby server's operating system, but ii) it is uncertain whether the data has arrived in the stable storage of the standby server. For data retention, this setting is sufficient even if the standby server instance of AgensGraph fails. However, this is not the case if the standby server fails at the operating system level.

  - If you are using synchronous replication, it is generally reasonable to i) wait for it to be written locally to disk, ii) wait for replication of WAL records, or ii) allow transactions to be committed asynchronously. However, the local settings can be used for transactions waiting for local disk writes (rather than synchronous replication). If you do not set synchronous_standby_names, on, remote_write, and local settings will provide the same synchronization level, and transaction commit will only wait for local disk writes.

  - You can change this parameter at any time. The actions of all transactions are determined by the settings when they are actually committed. Thus, it is possible and useful to make some transaction commits synchronous and others asynchronous. For instance, when the default is reversed, in order to make a single transaction commit with multiple statements asynchronous, you should execute SET LOCAL synchronous_commit TO OFF within the transaction.

- **wal_sync_method** (enum)

  - This method is used to forcibly update WAL on the disk. If fsync is off, this setting is ignored because the WAL file is not forcibly updated. Possible values are:

    1. open_datasync (write WAL files with open() option O_DSYNC)

    2. fdatasync (call fdatasync() at each commit)

    3. fsync (call fsync() at each commit)

    4. fsync_writethrough (call fsync() at each commit, forcing write-through of any disk write cache)

    5. open_sync (write WAL files with open() option O_SYNC)

  - Use O_DIRECT if you also need the open_* option. The option may not be available on some platforms. The first type in the list above supported by the

platform is the default value. In Linux, however, fdatasync is the default. The default value is not necessarily ideal. To create a fail-safe configuration or to optimize performance, it may be necessary to change values or change other aspects of system configuration, which will be covered in Chapter 13.1. This parameter can only be set in the postgresql.conf file or on the server command line.

- **full_page_writes** (boolean)
  - If this parameter is on, the AgensGraph server records the whole content of each page on disk to WAL during the first modification of each disk page after a checkpoint. This is necessary because, in the event of an operating system failure, page writes in progress are only partially completed and old/new data may coexist in the pages on disk. In general, you cannot fully recover such pages only with row-level change data stored in WAL during crash recovery. Storing full-page images would ensure proper recovery of the pages; it would, however, definitely increase the size of data to be written to WAL. (As WAL replay always starts at the checkpoint, doing this job during the first change of each page after the checkpoint will be enough. One way to reduce the cost of writing the entire page to WAL is to increase the checkpoint interval parameter.
  - Turning off this parameter would improve normal operating speed, but in the event of a system failure, corrupted data may become unrecoverable or data corruption may not be revealed. These risks are small, but they are similar to when fsync is turned off. This means that this parameter may be disabled only when the environment is identical to the recommended conditions in fsync.
  - Disabling this parameter does not affect the use of WAL archiving for point-in-time recovery (PITR).
  - This parameter can only be set in the postgresql.conf file or on the server command line. The default value is on.
- **wal_log_hints** (boolean)
  - If this parameter is on, the AgensGraph server writes the whole content of the page to WAL during the first modification of each disk page after the checkpoint, including minor modification of the so-called hint bits.
  - If data checksum is enabled, hint bit updates are always WAL-logged and this parameter setting is ignored. If data checksum is enabled in the database, you

can use this parameter setting to test how much additional WAL logging occurs.

  - This parameter is set at server startup. The default is off.

- **wal_recycle** (boolean)

  - If set to on (the default), this option causes WAL files to be recycled by renaming them, avoiding the need to create new ones. On COW file systems, it may be faster to create new ones, so the option is given to disable this behavior.

- **wal_buffers** (integer)

  - Some of the shared memory that is used for the WAL data and has not been written to disk yet. The size of default setting “-1” is identical with to that of 1/32 (about 3%) of shared_buffers (64kB or more, less than one WAL segment, and usually 16MB). If the automatic setting is too large or too small, you can directly select it; if it is less than 32kB, it is processed as 32kB. This parameter is set at server startup.

  - As the content of the WAL buffers is written to disk per transaction commit, an extremely large value is likely to have little advantage. However, setting this value to at least a few megabytes will improve the write performance of busy servers by allowing multiple clients to commit at once. Autotuning with the default setting of -1 produces reasonable results in most cases.

- **wal_writer_delay** (integer)

  - Determines the delay between operations of the WAL writer. At each operation, the writer writes WAL to disk. After sleeping for milliseconds (wal_writer_delay), it repeats the process. The default is 200 milliseconds (200ms). In many systems, the efficient sleep delay setting is 10 milliseconds. If wal_writer_delay is set to a value other than a multiple of 10, the result will be identical with the value set by rounding a multiple of 10. This parameter can only be set in postgresql.conf or on the server command line.

- **wal_skip_threshold** (integer)

  - When wal_level is minimal and a transaction commits after creating or rewriting a permanent relation, this setting determines how to persist the new data. If the data is smaller than this setting, write it to the WAL log; otherwise, use an fsync of affected files. Depending on the properties of your storage, raising or lowering this value might help if such commits are slowing

concurrent transactions. If this value is specified without units, it is taken as kilobytes. The default is two megabytes (2MB).

- **commit_delay** (integer)
  - commit_delay adds the measured time delay in microseconds before initializing the WAL flush. This can improve the group commit throughput by allowing a large volume of transactions to be committed through a single WAL flush if the system load is large enough to be ready to commit additional transactions within a given interval. However, this also increases the waiting time up to commit_delay microseconds per WAL flush. In the case where there is no transaction ready to commit, then the delay is a waste of time. Thus, with the writing about to start, the delay is done when there is the required number of transactions (at least commit_siblings) being performed. In addition, if fsync is disabled, the delay is not performed. The default commit_delay is 0 (no delay). Only the superuser may change the setting.
- **commit_siblings** (integer)
  - The minimum number of concurrent open transactions is required for performing commit_delay. If the value is large, it is highly likely that at least one other transaction is ready to commit during the delay interval. The default is 5 transactions.

Query Planning

Planner Method Configuration

These configuration parameters provide an approximate way to affect the query plan selected by the query optimizer. If the default plan selected by the optimizer for a particular query is not optimal, you may use one of these configuration parameters as an ad hoc way to force the optimizer to choose another plan. A better way for the optimizer to improve the level of the selected plan is to adjust planner cost constants, run ANALYZE manually, increase the default_statistics_target configuration parameter, and use ALTER TABLE SET STATISTICS; this way, the statistics collected for a particular column can be increased.

- **enable_bitmapscan** (boolean)
  - Enable or disable query planner's use of bitmap-scan plan types. The default value is on.
- **enable_hashagg** (boolean)
  - Enable or disable query planner's use of hashed aggregation plan types. The default value is on.

- **enable_incremental_sort** (boolean)
  - Enables or disables the query planner's use of incremental sort steps. The default is on.
- **enable_hashjoin** (boolean)
  - Enable or disable query planner's use of hash-join plan types. The default value is on.
- **enable_indexscan** (boolean)
  - Enable or disable query planner's use of index-scan plan types. The default value is on.
- **enable_indexonlyscan** (boolean)
  - Enable or disable query planner's use of index-only-scan plan types. The default value is on.
- **enable_material** (boolean)
  - Enable or disable query planner's use of materialization. It is difficult to completely inhibit materialization, but turning off this variable will prevent the planner from inserting materialized nodes except when accuracy is required. The default value is on.
- **enable_mergejoin** (boolean)
  - Enable or disable query planner's use of merge-join plan types. The default value is on.
- **enable_nestloop** (boolean)
  - Enable or disable query planner's use of nested-loop join plans. It is difficult to completely inhibit nested-loop joins, but turning off this variable will prevent the planner from using it if there are other alternative methods available. The default value is on.
- **enable_seqscan** (boolean)
  - Enable or disable query planner's use of sequential scan plan types. It is difficult to completely inhibit sequential scans, but turning off this variable will prevent the planner from using it if there are other alternative methods available. The default value is on.
- **enable_sort** (boolean)
  - Enable or disable query planner's use of explicit sort steps. It is difficult to completely inhibit explicit sorts, but turning off this variable will prevent the

planner from using it if there are other alternative methods available. The default value is on.

- **enable_tidscan** (boolean)
  - Enable or disable query planner's use of TID scan plan types. The default value is on.
- **enable_parallel_append** (boolean)
  - Enables or disables the query planner's use of parallel-aware append plan types. The default is on.
- **enable_parallel_hash** (boolean)
  - Enables or disables the query planner's use of hash-join plan types with parallel hash. Has no effect if hash-join plans are not also enabled. The default is on.
- **enable_partition_pruning** (boolean)
  - Enables or disables the query planner's ability to eliminate a partitioned table's partitions from query plans. This also controls the planner's ability to generate query plans which allow the query executor to remove (ignore) partitions during query execution. The default is on.
- **enable_partitionwise_join** (boolean)
  - Enables or disables the query planner's use of partitionwise join, which allows a join between partitioned tables to be performed by joining the matching partitions. Partitionwise join currently applies only when the join conditions include all the partition keys, which must be of the same data type and have exactly matching sets of child partitions. Because partitionwise join planning can use significantly more CPU time and memory during planning, the default is **off**.
- **enable_partitionwise_aggregate** (boolean)
  - Enables or disables the query planner's use of partitionwise grouping or aggregation, which allows grouping or aggregation on a partitioned tables performed separately for each partition. If the GROUP BY clause does not include the partition keys, only partial aggregation can be performed on a per-partition basis, and finalization must be performed later. Because partitionwise grouping or aggregation can use significantly more CPU time and memory during planning, the default is **off**.

The following are the parameters added in AgensGraph.

- **enable_eager** (boolean)

  - Enables or disables the use of eager plans in the query planner. The default value is ON.
- **enable_gathermerge** (boolean)
  - Enables or disables the use of gather merge plans in the query planner. The default value is ON.
- **enable_multiple_update** (boolean)
  - Users may choose whether to enable/disable `enable_multiple_update` to allow/disallow multiple modifications of the graph elements. The default value is ON.

Planner Cost Constants

The cost variable described in this section is calculated at an arbitrary scale. As only the relative values are related, the query plan does not change if it is moved up or down with the same coefficient. Basically, these cost variables are based on the cost of fetching sequential pages. Since seq_page_cost is generally set to 1.0, other cost variables are configured based on seq_page_cost. You may use other costs, if you want, such as the number of milliseconds that you actually run on a particular machine.

> Note: Unfortunately, there is no suitable way to determine the ideal value for a cost variable. It is best to process this as an average of the total query mixes received by a particular installation. In light of some experience, we should say changing this value can be very dangerous.

- **seq_page_cost** (floating point)
  - Sets the disk page fetch cost, which is part of the sequential fetch method expected by the planner. The default value is 1.0. This value can override tables and indexes in a particular tablespace by setting the same-named tablespace parameters.
- **random_page_cost** (floating point)
  - Sets the processing cost of nonsequentially-fetched disk pages expected by the planner. Sets the disk page fetch cost, which is part of the sequential fetch method expected by the planner. The default value is 4.0. This value can override tables and indexes in a particular tablespace by setting the same-named tablespace parameters.
  - Reducing this value in proportion to seq_page_cost causes the system to be biased towards the index scan. Increasing this value makes index scans more expensive; both values can be increased or decreased together to change the

importance of disk I/O costs in proportion to the CPU cost. This feature is explained in the following parameters.

  - Random access to disk storage is typically four times more expensive than sequential access. However, as with indexed reads, most of the random access to disk occurs in cache; a small default is used (4.0). Even if random access appears to be 40 times slower than sequential access, we may expect 90% of the actual random reads to be cached.

  - If a 90% cache ratio is a false assumption in the user's workload, you can increase the random_page_cost to reflect the actual costs of random storage reads. Thus, it may be appropriate to reduce random_page_cost if the database is smaller than the total server memory and the data is fully cached. A storage with a relatively low random reading cost, such as an SSD, may be better modeled with a lower random_page_cost value.

  Tip: Although setting random_page_cost below seq_page_cost is allowed in the system, it is not reasonable to do so in practice. However, if the entire database is cached in RAM, no cost is required in touching out-of-sequence pages and the two parameters may be identically configured. Also, since the cost for fetching pages that are already in RAM in an excessively cached database is much lower than the cost for fetching pages in a normal state, the user must reduce both values in proportion to the CPU parameters.

- **cpu_tuple_cost** (floating point)

  - Sets the processing cost of each row during a query expected by the planner. The default value is 0.01.

- **cpu_index_tuple_cost** (floating point)

  - Sets the processing cost of each index entry during the index scan expected by the planner. The default value is 0.005.

- **cpu_operator_cost** (floating point)

  - Sets the processing cost of each operator or function during a query expected by the planner. The default value is 0.0025.

- **effective_cache_size** (integer)

  - Sets the effective size of disk cache that can be used for a single query to be estimated by the planner. This parameter is reflected in the cost of using the index. The larger the value, the more likely the index scan will be used. The smaller the value, the more likely a sequential scan will be used. If you set this parameter, you must consider both the shared buffers and disk cache of the kernel used by the AgensGraph data file. The number of concurrent

queries that are expected for different tables should be considered as well. This parameter has no effect on the amount of shared memory allocated by AgensGraph, nor does it preserve the kernel disk cache; it is used only for estimation purposes. The system also does not assume that data will remain in the disk cache between queries. The default is 4 gigabytes (4GB).

Error Reporting and Logging

Where To Log

- **log_destination** (string)
  - There are several ways to record server messages, including stderr, csvlog, and syslog. Separate these parameters with commas and add them to the desired list of log targets. The default is to write to stderr only. This parameter can only be set in `postgresql.conf` or on the server command line.
  - If `log_destination` contains csvlog, the log entries are output in a "comma separated values (CSV)" format for easier loading into the program.
- **logging_collector** (boolean)
  - This parameter is a background process that uses logging_collector to capture log messages sent to stderr and redirect them to a log file. This approach is more useful than logging to syslog because some types of messages do not appear in the syslog output. (Common examples include a dynamic link error message and an error message generated by a script such as `archive_command`). This parameter can be set only at server startup.
- **log_directory** (string)
  - If `logging_collector` is enabled, this parameter determines the directory where the log files will be created. It can be specified as an absolute path or can be compared to the cluster data directory. This parameter can only be set in `postgresql.conf` or on the server command line. The default setting is `pg_log`.
- **log_filename** (string)
  - If `logging_collector` is enabled, this parameter sets the filename of the generated log file. As this value is processed as a `strftime` pattern, it can be used to specify a `%`-escapes are file name. (If there is a timezone-dependent `%`-escapes, the operation will be performed based on the realm specified in `log_timezone`.) The supported `%`-escapes is similar to those listed in the Open Group's `strfectime` specification. Platform-specific (non-standard)

extensions do not work as the system's `strfectime` is not directly used. The default value is
`postgresql-%Y-%m-%d_%H%M%S.log`. If you specify a file name, reuse the log so that it does not fill the entire disk.

If you enable the CSV format output in `log_destination`, `.csv` is added as a timestamp log file name to generate a file name in CSV format (If you write `log_filename` as `.log`, `.log` is created).

This parameter can be set only in `postgresql.conf` or on the server command line.

- **log_file_mode** (integer)
  - On UNIX systems, this parameter sets the permissions on the log file when `logging_collector` is enabled (this parameter is ignored on Microsoft Windows). The parameter value is expected to be specified in an approved format mode by the `chmod` and `umask` calls (if you want to use a general octet format, the number must start with 0 (zero)).

    The default permissions are `0600`, and only the server owner can read or write the log files. Another commonly useful setting is `0640`, which allows a group of users to read the files. To use these settings, you need to change `log_directory` to store files outside the cluster data directory. It is not wise to set the log file world-readable because it may contain sensitive data in some cases.

    This parameter can be set only in `postgresql.conf` or on the server command line.

- **log_rotation_age** (integer)
  - If `logging_collectoris` enabled, this parameter determines the maximum lifetime of individual log files. After the maximum lifetime expires, a new log file is created. Set this parameter to 0 if you do not want to create new log files on a time basis. This parameter can be set only in `postgresql.conf` or on the server command line.
- **log_rotation_size** (integer)
  - When `logging_collector` is enabled, this parameter determines the maximum size of individual log files. After a KB file of the specified size is generated, a new log file is created. Set this parameter to 0 if you do not want to create new log files on a size basis. This parameter can be set only in `postgresql.conf` or on the server command line.

- **log_truncate_on_rotation** (boolean)
  - If `logging_collector` is enabled, this parameter will overwrite existing log files with the same names using AgensGraph rather than adding new ones. However, truncate occurs only when a new file is opened by time-based rotation rather than server restart or size-based rotation. When the parameter is off, existing files are added in all cases. For instance, if `log_filename` is set to something like `postgresql-%H.log`, you can create a 24-hour log file and then overwrite it periodically. This parameter can be set only in `postgresql.conf` or on the server command line.

    Example: To maintain a seven-day log, create a log file named as `server_log.Mon`, `server_log.Tue`, etc. every day. In order to automatically overwrite last week's log with this week's log, set `log_filename` to `server_log.%a`, and turn on `log_truncate_on_rotation`, and set `log_rotation_age` to `1440`.

    Example: A log file per hour is required to maintain a 24-hour log. As faster circular logging is possible when the file size exceeds 1GB, set `log_filename` to `server_log.%H%M`, turn on `log_truncate_on_rotation`, set `log_rotation_age` to `60`, and set `log_rotation_size` to `1000000`. If `log_filename` includes `%M`, you may allow size-driven rotation by choosing a filename that is different from the time-specified filename.
- **syslog_facility** (enum)
  - If you enable logging to syslog, this parameter determines which syslog "facility" to use. You may choose one from `LOCAL0`, `LOCAL1`, `LOCAL2`, `LOCAL3`, `LOCAL4`, `LOCAL5`, `LOCAL6`, and `LOCAL7`. The default value is `LOCAL0`. Refer to the documentation for the system syslog daemon. This parameter can be set only in `postgresql.conf` or on the server command line.
- **syslog_ident** (string)
  - If you enable logging to syslog, this parameter determines the name of the program used to identify the AgensGraph messages in the syslog log. The default value is `PostgreSQL`. This parameter can be set only in `postgresql.conf` or on the server command line.
- **syslog_sequence_numbers** (boolean)
  - When logged in syslog and the parameter is on (default), each message is prefixed with an increasing sequence number (e.g. `[2]`). This cycle means suppression of "— last message iteration frequency—" that requires many syslog implementations. A more recent syslog implementation might not

need it
(e.g. `$RepeatedMsgReduction` in rsyslog) because it can configure repetitive message suppression. You can also disable this option if you really want to block repetitive messages.

This parameter can be set only in `postgresql.conf` or on the server command line.

- **syslog_split_messages** (boolean)
  - If you enable logging to syslog, this parameter determines how messages are passed to syslog. When set to on (default), messages are split by line; long lines are split into 1024 bytes, which is the usual size limit in existing syslog implementations. When set to off, the AgensGraph server log messages are forwarded to the syslog service and are connected to the syslog service to handle potentially large messages.

    If syslog is ultimately logged in a text file, it is a good idea to leave the settings as they are as most syslog implementations cannot handle large messages or configure them in an unusual way. However, if syslog is ultimately used on other media, keeping the message logically can be more useful.

    This parameter can be set only in `postgresql.conf` or on the server command line.
- **event_source** (string)
  - When you enable logging in the event log, this parameter determines the program name used to identify AgensGraph messages in the log. The default value is `PostgreSQL`. This parameter can be set only in `postgresql.conf` or on the server command line.

Version and Platform Compatibility

Other Platforms and Clients

The following are the parameters added in AgensGraph.

- **allow_null_properties** (boolean)
  - You may allow insertion of null properties into vertices and edges. The default is OFF.
- **case_sensitive_ident** (boolean)

  - Use of case-sensitive identifiers can be set. If case_sensitive_ident is ON, all identifiers are case-sensitive. The default is OFF.
- **case_compat_type_func** (boolean)
  - When `case_sensitive_ident` is ON, case-insensitive identifiers can be used for type and function names. The default is OFF.

Tools

Client Tool

Agens

Agens is the terminal-based front end of AgensGraph. Enter the query interactively to make AgensGraph return the results. Files or command-line arguments may be typed as well. Agens also writes scripts and automates various tasks by providing a variety of meta commands and shell-like functions.

How to use:

```
$ agens [OPTION]... [DBNAME [USERNAME]]
```

Options

-a
–echo-all
All input lines are shown as standard outputs as read (not read interactively). This is equivalent to setting the ECHO variable to all.

-A
–no-align
Switches to non-aligned output mode (otherwise default output mode is aligned).

-b
–echo-errors
Shows failed SQL commands as standard error outputs. This is equivalent to setting the ECHO variable to error.

-c ***command***
–command=***command***
Specifies ***command*** to cause agens to execute the specified command string. This option can be repeated and used by combining commands with `-f` option. If `-c` or `-f` is specified, agens processes all `-c` and `-f` options without reading the command as standard input and exits.

command must be a command string that can be fully parsed by the server or a single backslash command. Therefore, you cannot mix SQL and agens meta-commands in `-c` option. You are allowed to use the repeated `-c` option or use strings with a pipe ('|'). Examples:

```
$ agens -c '\x'-c 'SELECT * FROM test;'
```

or

```
$ echo '\x \\SELECT * FROM test;' | agens
```

(\\ is a delimiter metacommand.)

Each SQL command string delivered to `-c` is sent to the server as a single query. For this reason, if no explicit BEGIN/COMMIT commands that can be split into multiple transactions are contained in a string, the server runs it in a single transaction even with multiple SQL commands in a string. Agens also outputs only the results of the last SQL command to the string. As standard inputs are transmitted separately here, this behavior differs from that when they are sent as agens' standard input or when the same strings are read from a file.

Placing more than one command in a single `-c` string often leads to unexpected results because of this behavior. It is better to use the `-c` command repeatedly, or use echo as in the example above, or use the here-document shell, as in the example below, to enter multiple commands into the agens standard input.

```
psql <<EOF
\x
SELECT * FROM test;
EOF
```

-d ***dbname***
–dbname=***dbname***
Specifies the name of the database to connect to. This is the same as specifying ***dbname*** as an argument, not the first option on the command-line.

If this parameter has an equal (=) sign or starts with a valid URI prefix (postgresql:// or postgres://), it is processed as a conninfo string.

-e
–echo-queries
Copy all SQL commands sent to the server to standard output. This is equivalent to setting the variable ECHO in the query.

-E
–echo-hidden
Equivalent to turning the ECHO_HIDDEN variable to on.

-f ***filename***
–file=***filename***
Read commands from a file with a ***filename***, rather than standard input. This option can be repeated and combined in order using the `-c` option. When specifying `-c` or `-f`, agens does not read commands as standard input. Instead, it processes all the `-c` and `-f` options in

order and exits. Except for this, this option is generally equivalent to the metacommand `\i`.

If the ***filename*** is -(hyphen), standard input reads EOF or up to a metacommand `\q`. This allows interaction between interactive input and input from the file. In this case, however, Readline is not used (as if -n is specified).

Using this option is slightly different from using agens < ***filename***. In general, both will work as expected, but using `-f` will allow you to use useful features such as error messages with line numbers. This option may reduce startup overhead. On the other hand, a variant using the shell's input changes guarantees (theoretically) the same output as the input it receives when everything is entered manually.

-F ***separator***
–field-separator=***separator***
Use ***separator*** as the field delimiter for unaligned output. This is equivalent to `\pset fieldsep` or `\f`.

-h ***hostname***
–host=***hostname***
Specifies the host name of the system on which the server is running. If the value begins with a slash, it is used as a directory for the Unix domain socket.

-H
–html
Turns on the HTML tabular output. This is equivalent to the `\pset` format html or `\H` command.

-l
–list
List all available databases and exit. Other non-connection options are ignored. Similar to meta-command `\list`.

-L ***filename***
–log-file=***filename***
Create all query logs in ***filename***

-n
–no-readline
Do not use extended command-line editing.

-o ***filename***
–output=***filename***
Save all query results to the ***filename*** file. It is the same as the `\o` command.

-p ***port***
–port=***port***
Specifies the TCP port or local Unix domain socket file extension on which the server listens for connections. The default value is the value of the PGPORT environment variable or, if not set, specified at compile time (default: 5432).

-P ***assignment***
–pset=***assignment***
Specifies output options in `\pset` style. The name and value must be separated by an equal sign instead of a space. For example, to set the output format to LaTeX, use -P format=latex.

-q
–quiet
Specifies this for quiet work of agens. It basically outputs the start message and various information. With this option enabled, nothing would occur. This is useful when used with the `-c` option. Equivalent to turning on the QUIET variable.

-R ***separator***
–record-separator=***separator***
Use ***separator*** as the record delimiter for unaligned output. This is the same as the `\pset recordsep` command.

-s
–single-step
Run in single-step mode. That is, a message is displayed to the user before each command is sent to the server, and an undo option is also displayed. Used to debug scripts.

-S
–single-line
Like a semicolon, a new line character is executed in single-line mode, which terminates the SQL command.

> Notice: This mode is provided for those who want to use it and is not basically recommended for use. In particular, if you mix SQL and meta-commands on a single line, the order of execution may not be obvious to the first-time user.

-t
–tuples-only
Turn off column name and result row output. This is the same as the `\t` command.

-T ***table_options***
–table-attr=***table_options***
Specifies the options to place within the HTML table tag. For more information, refer to `\pset`.

-U ***username***
–username=***username***
Connect to the database with ***username*** instead of the default (you must have necessary permissions).

-v ***assignment***
–set=***assignment***
–variable=***assignment***
Performs the same variable assignment as the meta-command `\set`. If the name and value are present, they must be separated by an equal sign on the command-line. To unset the variable, leave the equal sign. To set a variable to an empty value, use the equal sign and leave the value unchanged. Variables that are allocated/executed in the initial phase of startup but are reserved for internal use can be overwritten later on.

-V
–version
Shows agens version information and exits.

–revision
Shows agens revision information and exits.

-w
–no-password
Do not enter a password. If the server requires password authentication and a password in some other ways, such as `.pgpass` file, the connection attempt will fail. This option is useful for deployment jobs or scripts where there is no user to enter a password.

As this option is set for the entire session, it affects use of a metacommand `\connect` as well as the initial connection attempt.

-W
–password
Forcibly sets agens to request a password before connecting to the database.

This option is not needed when the server requires password authentication; agens automatically requires a password as well. However, agens will waste connection attempts to know whether the server wants a password or not. In some cases it is better to type -W to avoid further connection attempts.

As this option is set for the entire session, it affects use of a metacommand `\connect` as well as the initial connection attempt.

-x
–expanded
Turns on extended expression assignment mode. This is the same as the `\w` command.

-X
–no-psqlrc
Do not read the startup file (either the system's `psqlrc` file or your `.psqlrc` file).

-z
–field-separator-zero
Sets the field separator for unaligned output to 0 bytes.

-0
–record-separator-zero
Sets the record separator for unaligned output to 0 bytes. For example, it is useful for interfaces like `xargs -0`.

-1
–single-transaction
This option should be used with one or more `-c` and `/` or `-f` options.

This option wraps all commands in a single transaction by issuing BEGIN first, and COMMIT in the end. This ensures that all commands are completed successfully or that the changes do not take effect.

If you include `BEGIN`, `COMMIT`, or `ROLLBACK` in the command itself, this option cannot have the desired effect. In addition, if you cannot execute individual commands within a transaction block, specifying this option will cause the entire transaction to fail.

-?
–help[=***options***]
Displays help for agens and exits. The optional ***options*** parameter (optional default) selects the part of agens described. The command describes agens' backslash command, and options describes the command-line options that can be passed to agens. Shows help for agens configuration variables.

Exit Status

Agens will return 0 if the shell is done successfully, or 1 if there is a fatal internal error (e.g. memory shortage and file cannot be found). Agens will return 2 if the session is not interactive because of incorrect connection to the server, or return 3 if the script has an error and the variable `ON_ERROR_STOP` is set.

Usage

Connecting to a Database

Agens is a generic AgensGraph client application. In order to connect to the database, you need to know the name of the target database, hostname and port number of the server, and user name to connect to. Agens can advertise each parameter via command-line options such as `-d`, `-h`, `-p` and `-U`. If an argument that does not belong to the options is found, it is interpreted as the database name (or the user name if the database name is already specified). Not all of these options are required, and there are useful defaults. If you omit the hostname, agens will connect to the server on the local host via a Unix domain socket or to localhost on a system that does not have a Unix domain socket via TCP/IP. The default port number is determined at compile time. As the database server uses the same defaults, you do not need to specify the port (number) in most cases. The default user name is the operating system user name and the default database name. You cannot connect to any database by user name. The database administrator should inform the user of the access rights.

When the default values are not appropriate, use the `PGDATABASE`, `PGHOST`, `PGPORT` and `/` or `PGUSER` environment variables by setting them with appropriate values. If you do not want to enter your password regularly, it is convenient to use the `~/.pgpass` file.

Entering SQL Commands

In general, agens provides the name of the database to which agens is currently connected; additionally, it gives “=#” to superuser and a string named “=#” to end users. Here is an example:

```
$ agens postgres
agens (11.11)
Type "help" for help.
postgres=#
```

At the prompt, the user may enter SQL commands. Typically, when the semicolon at the end of the command-line is reached, the input line is sent to the server. The end of the line does not terminate the command. Thus, commands can be distributed across multiple lines for clarity. If a command is sent and executed without error, the result of the command is displayed on the screen.

Each time the command is executed, agens polls asynchronous notification events generated by LISTEN and NOTIFY.

C-style block comments are passed to the server for processing and removal, but the SQL standard comments are removed by agens.

Meta-Commands

Starting with a backslash that is not enclosed in quotes (entered in agens) is an agens meta-command processed by agens itself. This command makes agens more useful for administration or scripting. Meta-commands are usually called slash- or backslash-commands.

The format of agens commands: A backslash followed by a command verb and arguments. The arguments are separated from the command verb by a number of different whitespace characters.

In order to include a space character in the argument, use single quotes. To include single quotes in an argument, you must put two single quotes inside the single quotes. A single citation may include `\n` (newline), `\t` (tab), `\B` (backspace), `\r` (carriage return), `\f` (form feed), ***`\digits`*** (octal) and ***`\xdigits`*** (hexadecimal). A backslash in front of other characters in single-quoted text cites whatever the character is.

The text enclosed in backticks (`) within an argument is considered to be a command-line passed to shell. The output of the command (with the following linefeed removed) replaces the text inside the backticks.

If an agens variable name appears after a colon (:) without quotes, it is replaced with the value of the variable as described in SQL Interpolation.

Some commands use an SQL ID (e.g. table name) as an argument. This argument complies with the SQL syntax rules. Unquoted characters are processed as lowercase letters and double quotation marks (“) prevent case toggling and include spaces in identifiers. A double quote within double quotes is reduced to a single double quotation mark. For example, FOO”BAR”BAZ is interpreted as fooBARbaz and “A weird”” name” becomes A weird” name.

Parsing arguments stops at the end of a line or when another backslash is encountered without quotes. A backslash without quotes is considered start of a new metacommand. The special sequence `\\` (two backslashes) marks the end of the argument; if there is an SQL command, it continues parsing. This allows you to mix SQL and agens commands on a single line. In any case, however, the meta-command cannot continue beyond the end of the line.

The following metadata is defined.

`\a`
If the current table output format is unaligned, it is switched to aligned. If it is not unaligned, it is set to unaligned. This command is kept for backward compatibility.

`\c` or `\connect` [-reuse-previous=***on|off***] [***dbname*** [***username***] [***host***] [***port***] | ***conninfo***]

Sets up a new connection to the AgensGraph server. The connection parameters to be used can be specified using the location syntax or ***conninfo*** connection string.

If the command omits the database name, user, host, or port, the new connection can reuse the values of the previous connection. By default, the values of the previous connection are reused except when processing the ***conninfo*** string. Passing the first argument of `-reuse-previous=on` or `-reuse-previous=off` the default value. If the command specifies or does not reuse certain parameters, the libpq default value is used. Randomly specifying ***dbname***, ***username***, ***host***, and ***port*** are equivalent to omitting the parameters.

If the new connection is successful, the previous connection is closed. If the connection attempt fails (invalid username, access denied, etc.) with agens in the interactive mode, only the previous connection is maintained. If an error occurs when running a non-interactive script, processing stops immediately. This feature was chosen for user convenience in case of typos and as a safety mechanism to prevent scripts from accidentally working in other databases.

Here is an example:

```
db=# \c mydb myuser SKAI worldwide  5555
db=# \c service=mydb
db=# \c "host=localhost port=5555 dbname=mydb connect_timeout=10 sslmode=disa
ble"
db=# \c postgresql://myuser@localhost/mydb?application_name=myapp
```

`\C` [***title***]
Sets or clears the title of any table displayed as a result of a query. This command is equivalent to `\pset title title` (the name of this command was derived from “caption” because it was previously used to set caption in the HTML table).

`\cd` [***directory***]
Changes the current working directory to ***directory***. Without arguments, the current user’s home directory is changed.

> Tip : The current working directory can be checked by typing `\! pwd`.

`\conninfo`
Shows information on the current database connection.

`\copy` { ***table*** [(***column_list***)] | (***query***) } { from | to } { ***'filename'*** | program ***'command'*** | stdin | stdout | pstdin | pstdout } [[with] (***option*** [, ...])]

Performs a frontend (client) copy. This is the job of executing the SQL COPY command. However, instead of reading or writing the specified file, it reads or writes the data between the server and the local file system. File access and permissions are the access and permissions of the local user (not the server), and SQL superuser privileges are not required.

If a program is specified, the command is executed as agens and is passed between the server and the client, or routed between commands. The redo permission is an execute permission of the local user, not the server, and SQL superuser privileges are not required.

In the case of `\copy ... from stdin`, it reads data rows from the source where the command was executed and reads until it encounters `\.` or EOF. This option is useful for filling in-line tables in SQL script files. In the case of `\copy ... to stdout`, the output is sent to the location where the agens command output is and the COPY count command state is not displayed (because it can be confused with data rows). Use pstdin or pstdout to read or write agens' standard input or output, regardless of the current command source or `\o`.

The syntax of this command is similar to the SQL COPY command. All options except data source/destination are the same as those specified in COPY. For this reason, special parsing rules apply to the `\copy` command; agens' substitution rules and backslash escapes do not apply.

> Tip : This is not as efficient as the SQL COPY command because all data must pass through a client/server connection. SQL commands can be more efficient and better for a large volume of data.

`\copyright`
Displays the copyright and distribution terms of AgensGraph.

`\crosstabview` [***colV*** [***colH*** [***colD*** [***sortcolH***]]]]
Executes the current query buffer (e.g. \g) and displays the result in the crossbar grid. The query must return at least three columns. The output columns identified by ***colV*** are vertical headers and the output columns identified by ***colH*** are horizontal headers. ***colD*** identifies the output column to display in the grid. ***sortcolH*** identifies optional sorting columns of horizontal headers.

Each column specification can be a column number (starting from 1) or a column name. Common SQL case collapse and quotation rules apply to column names. If omitted, ***colV*** becomes column 1 and ***colH*** becomes column 2. ***colH*** and ***colV*** must be different. If c ***colD*** is

not specified, there must be exactly three columns in the query result; ***colV*** or colH is also displayed as ***colD***.

The vertical header shown at the top left contains the values found in the column in the same order as the query results, but the duplicates are removed.

The horizontal header shown in the first row contains the value in the ***colH*** column where the column is de-duplicated. By default, these appear in the same order as the query results. However, if the optional ***sortcolH*** argument is given, the value must be an integer number; the ***colH*** value is displayed in the horizontal header sorted according to the ***sortcolH*** value.

If the x values of ***colH*** and the y values of ***colV*** in the crosstab grid are clear, then the cell at the intersection (x, y) contains a query result row (***colH*** column value= x and ***colV*** value= y) and the value of ***colD*** column. If there is no such column, it returns a blank cell. If there are multiple rows, an error occurs.

`\d[S+]` [***patterns***]
For each relation type (table, view, materialized view, index, sequence, and foreign table) or ***patterns*** match type, the columns, types, default values (if they are not defaults), and NULL corresponding to all columns are displayed,. Associated indexes, constraints, rules, and triggers are displayed as well. In the case of foreign tables, associated foreign servers are also displayed ("pattern matching" is defined in the pattern below).

For some relationship types, `\d` displays additional information of each column, such as the column values of the sequences, index expressions for indexes, and foreign data wrapper options for foreign tables.

The command form `\d+` is identical except that more information is displayed. All comments related to the columns in the table are displayed, including the OIDs in the table. If not related, the default replica ID setting is displayed.

By default, only user-generated objects are displayed, and patterns or S qualifiers are provided to include system objects.

> Note: If `\d` is used without a pattern argument, it is equivalent to `\dtvmsE`. A list of tables, views, materialized views, sequences, and foreign tables are displayed.

`\da[S]` [***pattern***]
Lists aggregate functions with the return type and data type to be manipulated. If the ***pattern*** is specified, only aggregates of names that match the pattern are displayed. By default, only user-generated objects are displayed, and patterns or S qualifiers are provided to include system objects.

`\dA[+]` [***pattern***]
Lists access methods. If the ***pattern*** is specified, only access methods with names matching

the pattern are shown. When + is added to the command name, each table realm is listed with options related to disk size, permissions, and description.

`\db[+]` [***pattern***]
Lists tablespaces. If the ***pattern*** is specified, only tablespace names matching the pattern are displayed. When + is added to the command name, each table realm is listed with options related to disk size, permissions, and description.

`\dc[S+]` [***pattern***]
Lists the conversions between character set encodings. If the ***pattern*** is specified, only the conversion patterns with names matching the pattern are listed. By default, only user-generated objects are displayed, and patterns or S qualifiers are provided to include system objects. When + is added to the command name, objects with an associated description are listed.

`\dC[+]` [***pattern***]
Lists the types of cast. If the ***pattern*** is specified, only the casts matching the pattern of the source or destination type are listed. When + is added to the command name, objects with an associated description are listed.

`\dd[S]` [***pattern***]
Type constraints, operator classes, operator families, rules, and descriptions of trigger objects are listed. All other comments can be seen by a backslash command of each corresponding object type.

`\dd` shows a description of the appropriate type of visible object if an object or argument that matches the ***pattern*** is not provided. In both cases, however, only objects with descriptions are listed. By default, only user-generated objects are displayed, and patterns or S qualifiers are provided to include system objects.

A description of the object can be created using the COMMENT SQL command.

`\ddp` [***pattern***]
Lists the default access permission settings. The entries for each role (schema, if applicable) that has changed from the built-in defaults are listed. If ***patterns*** are specified only those entries whose role names or schema names match the pattern are listed.

The permission command of ALTER DEFAULT is used to set default access permissions. The meaning of the displayed permissions is described in GRANT.

`\dD[S+]` [***pattern***]
Lists domains. If the ***pattern*** is specified, only domains with names matching the pattern are displayed. By default, only user-generated objects are displayed, and patterns or S

qualifiers are provided to include system objects. When + is added to the command name, objects with an associated description are listed.

`\dE[S+]` [***pattern***]
`\di[S+]` [***pattern***]
`\dm[S+]` [***pattern***]
`\ds[S+]` [***pattern***]
`\dt[S+]` [***pattern***]
`\dv[S+]` [***pattern***]
In this command group, the characters E, i, m, s, t, and v represent a foreign table, index, materialized view, sequence, table, and view, respectively. You can get a list of these types of objects by specifying some or all of these characters in any order. For example, `\dit` lists indexes and tables. When + is added to the command name, each object realm is listed with options related to disk size, permissions, and description. If ***patterns*** are specified, only objects with names matching the pattern are listed. By default, only user-generated objects are displayed, and patterns or S qualifiers are provided to include system objects.

`\des[+]` [***pattern***]
Lists external servers (mnemonic: “external servers”). If the ***pattern*** is specified, only servers with matching names are listed. If you use the format `\des`, you will also see a full description of each server, including the server’s ACL, type, version, options, and description.

`\det[+]` [***pattern***]
List external tables (mnemonic: “external tables”). If the ***pattern*** is specified, only those entries whose table name or schema name matches the pattern are listed. If you use the format `\det+`, you will also see general options and a description of external tables.

`\deu[+]` [***pattern***]
Lists user mappings (mnemonic: “external users”). If the ***pattern*** is specified, only mappings of user names that match the pattern are listed. If you use the format `\deu+`, you will also see a list of additional information about each mapping.

> Caution : `\deu+` may also display the remote users’ user names and passwords. Be careful not to disclose them.

`\dew[+]` [***pattern***]
Lists external data wrappers (mnemonic: “external wrappers”). If the ***pattern*** is specified, only external data wrappers matching the pattern are listed. If you use the format `\dew+`, you will also see ACLs, options, and descriptions of external data wrappers.

`\df[antwS+]` [***pattern***]
The function is listed with the result data type, argument data type, and function type classified as “agg” (aggregate), “normal”, “trigger”, or “window”. To display only certain

types of functions, add the corresponding letter a, n, t, or w to the command. If the ***pattern*** is specified, only functions whose names match the pattern are displayed. By default, only user-generated objects are displayed, and patterns or S qualifiers are provided to include system objects. If you use the format `\df+`, you will see additional information on each feature, including variability, parallel safety, owner, security classification, access permissions, language, source code, and description.

> Tip: To find a function that takes an argument or returns a value of a certain data format, use the pager's search function to scroll through the `\df` output.

`\dF[+]` [***pattern***]
Lists text search settings. If the ***pattern*** is specified, only the settings for the names that match the pattern are displayed. If you use the format `\dF+`, you will see a full description of each setting, including a default text search parser and a dictionary list of token types for each syntax.

`\dFd[+]` [***pattern***]
Lists text search dictionaries. If the ***pattern*** is specified, only the dictionary with the name matching the pattern is displayed. If you use the format `\dFd+`, you will see additional information on each selected dictionary, including the default text search template and option values.

`\dFp[+]` [***pattern***]
Lists text search parsers. If the ***pattern*** is specified, only the parser whose name matches the pattern is displayed. If you use the format `\dFp+`, you will see a full description of each parser, including a list of basic features and the recognized token types.

`\dFt[+]` [***pattern***]
Lists text search templates. If the ***pattern*** is specified, only templates with names matching the pattern are displayed. If you use the format `\dFt+`, you will see additional information for each template, including the default feature names.

`\dg[S+]` [***pattern***]
List the database roles. (This is now equivalent to `\du` as the concepts of "user" and "group" have been integrated into "role"). By default, only user-created roles are displayed, and patterns or S qualifiers are provided to include system objects. If the ***pattern*** is specified, only roles with names matching the pattern are listed. If you use the format `\dg+`, you will see additional information on each role (currently, this adds a description of each role).

`\dG[+]` [***pattern***]
Displays a list of graphs.

`\dGe[+]` [***pattern***]
Displays a list of edge labels in the graph.

`\dGl[+]` [***pattern***]
Displays a list of labels in the graph.

`\dGv[+]` [***pattern***]
Displays a list of vertex labels in the graph.

`\dGi[+]` [***pattern***]
Displays a list of property indexes in the graph.

`\di[S+]` [***pattern***]
Displays a list of indexes.

`\dl`
This is an alias for `\lo_list` that shows a list of large objects.

`\dL[S+]` [***pattern***]
Lists the procedural languages. If the ***pattern*** is specified, only the languages whose names match the pattern are listed. By default, only user-generated languages are displayed, and patterns or S qualifiers are provided to include system objects. When + is added to the command name, languages are listed with their respective call handler, validator, access rights, and whether it is a system object.

`\dn[S+]` [***pattern***]
Lists the schemas (namespaces). If the ***pattern*** is specified, only the schemas with names matching the pattern are displayed. By default, only user-generated languages are displayed, and patterns or S qualifiers are provided to include system objects. When + is added to the command name, each object is listed with its associated permissions and description (if any).

`\do[S+]` [***pattern***]
Lists the operators, together with their operands and result types. If the ***pattern*** is specified, only the operators whose names match the pattern are displayed. By default, only user-generated objects are displayed, and patterns or S qualifiers are provided to include system objects. When + is appended to the command name, additional information on each operator is displayed (only the names of current basic features are displayed).

`\dO[S+]` [***pattern***]
Lists the collations. If the ***pattern*** is specified, only the collations of names matching the pattern are listed. By default, only user-generated languages are displayed, and patterns or S qualifiers are provided to include system objects. When + is added to the command name, combinations are listed with their respective description. As only the collations that can be used with an encoding of the current database are displayed, the result may be different from that you get from other databases of identical installation.

`\dp` [***pattern***]
Lists access rights associated with tables, views, and sequences. If the ***pattern*** is specified, only tables, views, and sequences with names that match the pattern are listed.

The GRANT and REVOKE commands are used to set access permissions. The meanings of the displayed permissions are described in GRANT.

`\drds` [***role-pattern*** [***database-pattern***]]
Lists the defined configuration settings. These settings can be role-specific, database-specific, or both. The ***role-pattern*** and ***database-pattern*** are used to select specific roles and databases, respectively. If this option is omitted or + is specified, all settings are listed, including each role-specific or database-specific configuration that is not specified.

The ALTER ROLE and ALTER DATABASE commands are used to define the role and database-specific configuration settings.

`\dT[S+]` [***pattern***]
Lists the data types. If the ***pattern*** is specified, only types with names matching the pattern are listed. When + is added to the command name, all types are listed with their respective internal name and size (i.e. values with the permissions associated with the enum type). By default, only user-generated objects are displayed, and patterns or S qualifiers are provided to include system objects.

`\du[S+]` [***pattern***]
Lists the database roles. (This command is equivalent to `\dg` since the concepts of “user” and “group” were integrated into “role”). By default, only user-generated objects are displayed, and patterns or S qualifiers are provided to include system objects. If the ***pattern*** is specified, only roles with names matching the pattern are shown. If you use the format `\du+`, you will see additional information on each role (currently, this adds a description of each role).

`\dx[+]` [***pattern***]
Lists installed extensions. If the ***pattern*** is specified, only extensions whose names match the pattern are listed. If you use the format `\dx+`, you will see all objects belonging to each matching extension.

`\dy[+]` [***pattern***]
Lists the event triggers. If the ***pattern*** is specified, only event triggers with names matching the pattern are listed. When + is added to the command name, objects are listed with their associated descriptions.

`\e` or `\edit` [***filename***] [***line_number***]
If ***filename*** is specified, the file is edited, the editor closes, and the content is copied back to

the lookup buffer. If the ***filename*** is not specified, the current query buffer is copied to the temporary file edited in the same manner.

The new query buffer is re-parsed according to the general rules of agens. Here, the entire buffer is processed as a single line (as you cannot create a script in this way, use `\i`). That is, queries that end with a semicolon are executed immediately. Otherwise, it will wait in the query buffer. Type a semicolon or press `\g` or `\r` to cancel.

If you specify a line number, agens places the cursor on the specified line in the file or query buffer. If only one digit argument is given, agens regards it as a line number, not as a file name.

> Tip: See the Environment below for editor configuration and custom instructions.

`\echo` ***text*** [...]
Separates an argument with a single space (standard output) and prints along with a newline character. This can be useful for inserting information into script output. Here is an example:

```
db=# \echo `date`
Tue Oct 26 21:40:57 CEST 1999
```

If the first argument is an unquoted `-n`, no trailing linefeed is done.

> Tip: If you use the `\o` command to resubmit a query output, you can use `\qecho` instead of this command.

`\ef` [***function_description*** [***line_number***]]
This command patches and edits definitions of the named functions in the form of the CREATE OR REPLACE FUNCTION command. Editing is done in the same way as `\edit`. After the editor terminates, the updated command waits in the query buffer. Type a semicolon or press `\g` or `\r` to cancel.

The target function can be specified by name alone or by name and arguments (e.g. `foo(integer, text)`). If you have more than one function of the same name, you must specify the argument type.

If no function is specified, an empty CREATE FUNCTION template will be displayed for editing.

If a line number is specified, agens places the cursor on the specified line in the function body. (The function body does not usually start on the first line of the file.)

> Tip: For more information on configuring/customizing the editor, see Environment below.

\encoding [***encoding***]
Sets the client character set encoding. If there is no argument, this command displays the current encoding.

\ev [***view_name*** [***line_number***]]
This command patches and edits definitions of the named views in the form of the CREATE OR REPLACE VIEW command. Editing is done in the same way as \edit. After the editor terminates, the updated command waits in the query buffer. Type a semicolon or press \g or \r to cancel.

If you do not specify a view, an empty CREATE VIEW template will be displayed for editing.

If a line number is specified, agens places the cursor on the specified line in the view definition.

\f [***string***]
Sets the field delimiter for unordered query output. The default is the vertical bar (|). Regarding the usual way to set output options, refer to \pset.

\g [***filename***]
\g [|***command***]
Sends the current query input buffer to the server, saves the query result to ***filename***, or outputs the shell command ***command*** of the pipe (|). A file or command is written only if the query returns zero or more tuples successfully, rather than a SQL command that fails or does not return data.

\g is essentially equal to a semicolon. \g with an argument is a “one-shot” alternative to the \o command.

\gexec Sends the current query input buffer to the server and then processes each column of each row of the query output as an SQL statement to execute. For example, to create an index on each column of my_table, you should do the following:

```
dbdb=# SELECT format('create index on my_table(%I)', attname)
-# FROM pg_attribute
-# WHERE attrelid = 'my_table'::regclass AND attnum > 0
-# ORDER BY attnum
-# \gexec
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
```

The generated query is executed in the order that the rows are returned, and if there is more than one column, it runs from left to right within each row. NULL fields are ignored.

As the generated query is sent to the server for literal processing, it cannot contain an agens meta-command or an agens variable reference. If individual queries fail and `ON_ERROR_STOP` is not set, the remaining queries will continue to run. Execution of each query is subject to `ECHO` processing. (If you use `\gexec`, it is better to set `ECHO` to `all` or `queries`.) Query logging, single-stage mode, timing, and other query execution features also apply to each query generated.

`\gset` [***prefix***]
The server sends the current query to the input buffer and stores the result of the query in the agens variable (see Variables). The query to be executed must return exactly one row. Each column of a row is stored in a separate variable that is identical with the column. Here is an example:

```
db=# SELECT 'hello' AS var1, 10 AS var2
db-# \gset
db=# \echo :var1 :var2
hello 10
```

If ***prefix*** is specified, the corresponding string is appended to the column name of the query to create a variable name to use.

```
db=# SELECT 'hello' AS var1, 10 AS var2
db-# \gset result_
db=# \echo :result_var1 :result_var2
hello 10
```

If the column result is NULL, the variable is set to "unset."

If the query fails or does not return a row, the variable is not changed.

`\h` or `\help` [***command***]
Provides syntax help for the specified SQL command. If ***command*** is not specified, agens lists all commands of which help information is available. If ***command*** is an asterisk(*), it shows syntax help for all SQL commands.

> Note: You do not need to enclose a command consisting of multiple words in quotes to simplify typing. We recommend you to enter the command with `\help alter table`.

`\H` or `\html`
Turns on the HTML query output format. If the HTML format is already on, it will switch back to the default alignment text format. This command is used for compatibility and convenience. Refer to `\pset` for other output option settings.

`\i` or `\include` ***filename***
Reads the input from the file ***filename*** and executes it as if the input has been typed at the keyboard.

If ***filename***is -(hyphen), standard input reads up to EOF or metacommand `\q`. This allows the interactive input to interact with the input from the file. Readline operation is used only when activated at the outermost level.

Note: You must set the variable `ECHO` to `all` to view the rows read from the screen.

`\ir` or `\include_relative` ***filename***
`\l[+]` or `\list[+]` [***pattern***]
Lists the database on the server and displays the name, owner, character set encoding, and access permissions. If the ***pattern*** is specified, only databases matching the pattern are displayed. When + is added to the command name, the database size, default tablespace, and description are also displayed (the size information is available only to the databases that can be connected by the current user).

`\lo_export` ***loid filename***
Reads a large object using the OID ***loid*** of the database and write it to ***filename***. Note that the database server has a subtle difference from the server function `lo_export`, which operates according to the privileges of the user running in the same way as the server's file system.

Tip: Use `\lo_list` to find OID of the large object.

`\lo_import` ***filename*** [***comment***]
Saves the file as an AgensGraph large object. Use the specified annotation if necessary. Here is an example:

```
db=# \lo_import '/home/peter/pictures/photo.xcf' 'a picture of me'
lo_import 152801
```

The response indicates that the large object has received object ID 152801, which can be used to access large objects created in the future. It is always good to associate descriptions readable by humans with all objects to ensure readability. OIDs and comments can be viewed with the `\lo_list` command.

This command is slightly different from the server side as it acts as a local user on the local file system, rather than on the user/file systems of the `lo_import` server.

`\lo_list`
Displays a list of all AgensGraph large objects currently stored in the database and all annotations provided.

`\lo_unlink` ***loid***
Deletes a large object from the database using OID ***loid***.

> Tip: Use `\lo_list` to find OID of the large object.

`\o` or `\out` [***filename***]
`\o` or `\out` [|***command***]
Saves future query results to the filename file or saves future results to the shell ***command***. If no argument is specified, the query output is reset to standard output.

"Query result" includes the output of various backslash commands (e.g. `\d`) querying the database as well as all tables, command responses, and notifications from the database server, except error messages.

> Tip: Use `\qecho` to insert text output between query results.

`\p` or `\print` prints out the current query buffer to standard output.

`\password` [***username***]
Changes the password of the specified user (default is current user). This command prompts for a new password, encrypts it, and sends it to the server with the ALTER ROLE command. The new password is not recorded in the command history, server log, or anywhere else.

`\prompt` [***text***] ***name***
The user is prompted to enter the text assigned to the variable ***name***. An optional prompt string and ***text*** can be specified (in the case of multiple operation messages, the text is enclosed in single quotes).

By default, `\prompt` uses the terminal for input/output. However, in order to use the `-f` command-line switch, `\prompt` uses standard input/output.

`\pset` [***option*** [***value***]]
This command sets the option that affect output of the query results table. ***option*** indicates the option to set. The meaning of ***value*** varies depending on the option selected. For some options, if you omit ***value***, the options are specified or unset as described in the specific options. If no action is specified and ***value*** is omitted, the current setting is displayed.

`\pset` without an argument displays the current status of all output options.

The adjustable print options are:

- border
  value must be a number. In general, the larger the number, the more borders and lines it has. However, its details vary depending on particular formats. In HTML format, it is directly converted to the `border=` property. In most formats, only 0 (no

border), 1 (inner split line), and 2 (table frame) are meaningful, and values more than 2 are processed the same as `border=2`. In `latex` and `latex-longtable`, however, the value of 3 means that a divider can be added between the rows of data.

- columns
  columns Sets a width limit to determine if the target width in `wrapped` format and output is sufficient to switch to a pager or portrait display in extended silent mode. Zero (default) controls the detected screen width if the target width is controlled by the `COLUMNS` environment variable or `COLUMNS` is not set. When `columns` is 0, the `wrapped` format affects only the display output. With `columns` 0, the file and pipe output is also wrapped in that width.

- expanded (or x)
  If ***value*** is specified, it must be set to `on`/`off` or `auto` to enable or disable the expanded mode. If ***value*** is omitted, it is switched to on/off setting. In enhanced mode, query results are displayed in two columns. The column name is on the left and the data is shown on the right. This mode is useful when the data does not fit the screen in a normal “horizontal” state. The expanded mode in auto setting is used whenever one or more columns exist in the query output and they are wider than the screen. In other cases, normal mode is used. Automatic settings are only valid in sorted and wrapped formats. Other formats always behave as if expanded mode is turned off.

- fieldsep
  fieldsep Specifies the field delimiter to be used as an unaligned output format. This allows other programs to generate their own tab-delimited or comma-separated output. To set the tab as a field delimiter, type `\pset fieldsep '\t'`. The default field delimiter is `'|'`.

- fieldsep_zero
  fieldsep_zero Sets the field delimiter to 0 bytes for an unaligned output format.

- footer
  If ***value*** is specified, you must set on/off to enable or disable the sum of the table footer (***n*** rows). Turns the footer display on or off if the value is omitted.

- format
  format Sets the output format to `unaligned`, `aligned`, `wrapped`, `html`, `asciidoc`, `latex`(uses `tabular`), `latex-longtable`,troff-ms`. Abbreviations specific to each format are allowed (one letter is enough).

`unaligned` uses all the columns in a row separated by the currently active field delimiter. This feature is useful for creating output that can be read by other programs (e.g. tab-separated or comma-separated format).
The `aligned` format is a human-readable standard format text output (default).
The `wrapped` format appears to be `aligned`, but it wraps wide data values over multiple lines to match the output to the target column width. The target width is determined as described in the `columns` option. In agens, the column name does not attempt to wrap. Thus, if the total width required for the column headers exceeds the target, the `wrapped` format behaves the same as the `aligned` one.
`html`, `asciidoc`, `latex`, `latex-longtable`, `troff-ms` formats use a respective markup to create a table that can be included in a document; these are not complete documents. It may not be necessary for HTML, but LaTeX requires a complete document wrapper. `Latex-longtable` also requires the LaTeX `longtable` and `booktabs` packages.

- linestyle
  Sets border line drawing style to one of `ascii`, `old-ascii`, or `unicode`. Abbreviations specific to each format are allowed (one letter is enough). The default setting is ascii. This option only affects aligned and wrapped output formats.
  The `ascii` style uses regular ASCII characters. The newline of the data is displayed using the + sign in the right margin. When a wrapped format wraps data from one line to another without a newline character, a dot (.) appears in the right margin of the first line and in the left margin of the dotted line.
  The `old-ascii` style uses regular ASCII characters with the formatting style used in earlier versions. The newline character of the data is displayed using the `:` symbol instead of the left column delimiter. When data is wrapped from one line to another line without a newline, `;` symbol is used instead of the left column delimiter. The Unicode style uses Unicode box drawing characters. The newline of the data is displayed using the carriage return symbol in the right margin. When data is wrapped from one line to another line without a newline character, a newline is displayed in the right margin of the first line and in the left margin of the next line. If the border setting is greater than 0, the linestyle option also determines the character that draws the border line. Normal ASCII characters can be used anywhere, but Unicode characters are better viewed on character-recognizing displays.

- null
  Sets the string to print instead of the null value. The default value is to print nothing, which can be mistaken for an empty string. For example, we prefer `\pset null '(null)'`.

- numericlocale
  If ***value*** is specified, this must be on or off, which enables or disables locale-specific numeric notation. If this ***value*** is omitted, it switches to standard format and locale-specific numeric output.

- pager
  Controls use of the pager program for query and agens help output. If `PAGER` is set in the environment variable, the output is displayed in the specified program format. Otherwise, platform-dependent defaults (e.g. more) are used. If the pager option is off, the pager program is not used. When the pager option is on, the pager output is terminated and can be used appropriately on a screen that it does not fit. You can also enable the pager option always so that pager can be used for all terminals regardless of their fit on the screen. Using `\pset pager` with no ***value*** turns pager use on and off.

- pager_min_lines
  If `pager_min_lines` is set to a number greater than the page height, the pager program will not be called unless at least this output line is displayed. The default setting is 0.

- recordsep
  Specifies the record (line) delimiter to use as an unaligned output format. The default is the newline character.

- recordsep_zero
  Sets the record delimiter to 0 bytes to use for the unaligned output format.

- tableattr (or T)
  The HTML format specifies attributes to place within a table tag (e.g. `cellpadding` or `bgcolor`). You are recommended not to specify a border here because the `\pset` border has already been processed. If no value is specified, the table attribute is not configured.
  The `latex-longtable` format controls the proportional width of each column containing left-aligned data types; it is specified as a list of values separated (e.g. “0.2 0.2 0.6”). Unspecified output columns use the last specified value.

- title (or C)
  Sets the title of the table to be printed later. Can be used to provide output description tags. Without a value specified, the title is not configured.

- tuples_only (or t)
  If ***value*** is specified, it must be either on or off to enable or disable tuple-only mode. If ***value*** is omitted, switch to standard format and tuple-only mode output. General

output includes additional information such as column headers, titles, and various footers. In tuple-only mode, only actual table data are displayed.

- unicode_border_linestyle
  Sets the border drawing style in a Unicode line style to either `single` or `double`.
- unicode_column_linestyle
  Sets the column drawing style in a Unicode line style to either `single` or `double`.
- unicode_header_linestyle
  Sets the header drawing style in a Unicode line style to either `single` or `double`.

Tip: There are various shortcut commands for `\pset`. Refer to `\`, `\C`, `\H`, `\t`, `\T`, `\X`.

`\q` or `\quit`
Exits agens. In the script file, only execution of the script is terminated.

`\qecho` ***text*** [...]
This command is equivalent to `\echo`, but the output is written to the query output channel as set in `\o`.

`\r` or `\reset`
The query buffer is reset (released).

`\s` [***filename***]
Print the command-line history of agens as ***filename***. If ***filename*** is omitted, the record is written to standard output (use pager if applicable). This command cannot be used if agens were built without Readline support.

`\set` [***name*** [***value*** [...]]]
Sets agens variable ***name*** to ***value*** or all if given more than one value. If only one argument is given, the variable is set to an empty value. To unset the variable, use the `\unset` command.

`\set` without arguments shows all the names and values currently set in agens.

Valid variable names may include letters, numbers, and underscores. Refer to Variables. Variable names are case-sensitive.

You can set arbitrary variables as you like, but agens treats them as special variables. This will be explained in Variables.

Note: This command has nothing to do with the SQL command SET.

`\setenv` ***name*** [***value***]
Sets the environment variable ***name*** to ***value*** or unset the environment variable if no value is provided. Here is an example:

```
db=# \setenv PAGER less
db=# \setenv LESS -imx4F
```

`\sf[+]` ***function_description***
This command fetches and displays the definition of the named function in the form of the CREATE OR REPLACE FUNCTION command. The definition is output to the current query output channel as set by `\o`.

The target function can be specified by name alone or by name and argument (e.g. `foo(integer, text)`). If you have more than one function with the same name, you must specify the argument type.

When + is added to the command name, the output lines are numbered and the first line of the function body becomes the first line.

`\sv[+]` ***view_name***
This command fetches and displays the definition of the named view in the form of the CREATE OR REPLACE VIEW command. The definition is output to the current query output channel as set by `\o`.

When + is added to the command name, the output lines are numbered from 1.

`\t`
Toggles the header of output column name and the display at the bottom of row count. This command corresponds to `\pset tuples_only` and is provided for user convenience.

`\T` ***table_options***
Specifies attributes in the table tag in HTML format. This command corresponds to `\pset tableattr` ***table_options***.

`\timing` [***on*** | ***off***]
If there are no parameters, the time each SQL statement takes is displayed in milliseconds. Sets in the same way using parameters.

`\unset` ***name***
Releases (deletes) the agens variable ***name***.

`\w` or `\write` ***filename***
`\w` or `\write` | ***command***
Prints the current query buffer to a ***filename*** file or output a pipe to a shell command ***command***.

`\watch` [***seconds***]
Repeatedly executes current query buffer (such as `\g`) until the query is aborted or the query fails. Waits for the specified time (default 2) between runs. Each query result is represented by a header that contains the `\pset` title string (if any), query start time, and delay interval.

`\x` [***on*** | ***off*** | ***auto***]
Sets or toggles the expanded table format mode. Equivalent to `\pset expanded`.

`\z` [***pattern***]
Lists access rights associated with tables, views, and sequences. If a pattern is specified, only tables, views, and sequences whose names match the pattern are listed.

This is an alias for `\dp`("visibility privileges").

`\!` [***command***]
Switches to a separate shell or run a shell ***command***. The argument is no longer interpreted and the shell sees it as it is. In particular, variable substitution rules and backslash escapes do not apply.

`\?` [***topic***]
Displays help information. The optional ***topic*** parameter (`commands` by default) selects the part of agens to be described. The command describes agens' backslash `commands`. `options` describes the command-line options that can be passed to agens. `variables` show help info for the agens' configuration variables.

Patterns

The various `\d` commands use the ***pattern*** parameter to specify the name of the object to be displayed. In the simplest case, a pattern is the exact name of the object. Characters in a pattern are usually converted to lowercase, as in SQL names. For example, `\dt FOO` displays a table called `foo`. Placing double quotes around a pattern, as in SQL names, does not convert it to lowercase. If you need to include an actual double-quote character in the pattern, type two double quote characters within double quotation marks. This is consistent with the rules for the SQL citation identifier. For example, `\dt "FOO""BAR"` displays a table named `FOO"BAR`(not `foo"bar`). S Unlike the general rules for SQL names, you may put double quotes around a certain portion of a pattern. For example, `\dt FOO"FOO"BAR` shows a table named `fooFOObar`

Whenever the ***pattern*** parameter is omitted altogether, the `\d` command displays all objects that appear in the current schema search path. This is equivalent to using *as a pattern. (If an object is included in the search path and objects of the same kind do not appear before the search path, the object will be displayed in the search path. This is

identical to the statement that an object can be referenced by name without explicit schema credentials.) Use `*.*` To see all objects in the database, regardless of visibility.

Within a pattern, * is an arbitrary sequence of characters (including no characters), and ? matches any single character (this notation is similar to the Unix shell filename pattern). For instance, `\dt int*` displays all tables whose names begin with `int`. However, * and ? in double quotes lose their special meanings and show objects that are matched literally.

A pattern containing a dot (.) is interpreted as a schema name pattern and an object name pattern. For example, `\dt foo*.*bar*` displays all tables that contain "bar" in the table name whose schema name begins with "foo." If no dots appear, the pattern matches only those objects that appear in the current schema search path. Dots in double quotes lose their special meaning and show objects that are matched literally.

Advanced users can use regular-expression notation, such as character classes (e.g. [0-9]). All regular-expression special characters work as specified in the link. It is translated as a delimiter as mentioned above. That is,* is translated into regular-expression notation. * and ? are literally translated into `.` and `$`; these pattern characters can be imitated as needed by writing `.` as `?`, `R*` as `(R+|)`, and `R?` as `(R|)`. As the pattern does not match the full name, unlike the general interpretation of regular expressions, `$` does not match the regular expression (i.e. `$` is added to the pattern automatically). If you do not want to lock the pattern, write * at the start and/or end. Note that within double quotes all regular-expression special characters lose their special meaning and are literally matched. Regular-expression special characters are also matched literally in the operator name pattern (e.g. argument in `\do`).

Advanced Features

Variables

Agens provides a variable substitution feature, which is similar to the normal Unix command shell. A variable is simply a name/value pair and can be a string of any length. The name must consist of letters (including non-Latin characters), numbers, and underscores.

To set a variable, use `\set`, an agens meta-command. Here is an example:

```
db=# \set foo bar
```

Set the variable "foo" to "bar." To search the content of a variable, enter a colon (:) before the name. For example:

```
db=# \echo :foo
bar
```

It works on both regular SQL commands and meta-commands. See SQL Interpolationbelow for more information.

If you call `\set` without a second argument, the variable is set to an empty string value. To unset (delete) a variable, use the `\unset` command. To display the values of all variables, call `\set` with no arguments.

The number of these variables is specially processed by agens. You can change the value of a variable to indicate either a specific option setting that can be changed at runtime or a changeable state of agens in some cases. Using these variables for other purposes is allowed but not recommended as the program operation can go too fast. By convention, all special-processing variables are named with uppercase ASCII characters (possibly numbers and underscores). To ensure maximum compatibility in the future, do not use these variable names for your own use. Here is a list of all specially processed variables.

AUTOCOMMIT
When this is `on` (default), each SQL command is committed automatically once it is complete successfully. If you defer committing in this mode, you should enter a `BEGIN` or `START TRANSACTION SQL` command. If set to `off` or unset, SQL commands are not committed until `COMMIT` or `END` is explicitly called.

> Note: In autocommit-off mode, you should explicitly abort the failed transaction by entering `ABORT` or `ROLLBACK`. Also, if you terminate the session without committing it, your work will be lost.
>
> Note: The autocommit-on mode is a generic behavior of AgensGraph, but autocommit-off is closer to the SQL specification. If you want autocommit-off, you can set it using a `psqlrc` file available in the system or `~/.psqlrc` file.

COMP_KEYWORD_CASE
Determines a character to use when completing a SQL keyword. When set to `lower` or `upper`, the completed word is shown in uppercase or lowercase, respectively. If it is set to `preserve-lower` or `preserve-upper` (default), the completed word is shown only when the word is already entered; the entered content is shown in lowercase or uppercase, respectively.

DBNAME
The name of the currently connected database. Although this is set every time you connect to the database (including program startup), you may turn it off.

ECHO
If set to `all`, all non-empty input lines are read and printed to standard output (not applied to interactively read lines). Use the switch `-a` to select this behavior at program startup. If set to `queries`, agens prints each query to standard output each time it is sent to the server

(switch `-e`). If set to `error`, only the queries that caused an error will be displayed in the standard error output (switch `-b`). If it is unset or set to `none` (or any value other than above), the query will not be displayed.

ECHO_HIDDEN
If this variable is set to `on` and the backslash command inquires the database, the query is displayed first. This feature allows you to study the AgensGraph internals and provide similar functionality in your own program (use switch `-E` to select this behavior at program startup). If you set the variable to a value of `noexec`, the query will be displayed but will not be actually sent to the server for execution.

ENCODING
Character set encoding of the current client.

FETCH_COUNT
If this variable is set to an integer value>0, what is fetched and displayed in a group of rows is the result of the `SELECT` query, rather than the default behavior of collecting the entire result set. Thus, only a limited amount of memory is used, regardless of the size of the result set. When activating this function, the setting from 100 to 1000 is generally used. When using this feature, the query may fail after some rows are already displayed.

> Tip: Although this feature allows all output formats, the default alignment format tends to be not great; this is because the `FETCH_COUNT` rows of each group are formatted separately, leading to various column widths across the row group. Other output formats work better.

HISTCONTROL
If this variable is set to `ignorespace`, lines beginning with a space is not entered in the history list. If set to `ignoredups`, lines that match the previous history lines are not entered. `ignoreboth` combines these two options. If not set or set to `none` (or a value other than above), all rows read in the interactive mode are stored in the history list.

> Note: This is a feature from bash.

HISTFILE
The name of the file to use in order to save the history list. The default value is `~/.psql_history`. This is an example of input.

```
\set HISTFILE ~/.psql_history- :DBNAME
```

In `~/.psqlrc`, agens maintains a separate record of each database.

> Note: This is a feature from bash.

HISTSIZE
The number of commands to store in the command history. The default value is 500.

Note: This is a feature from bash.

HOST
This is the database server host currently connected. Although this is set every time you connect to the database (including program startup), you may turn it off.

IGNOREEOF
For unset, sending an EOF character (typically **Control + D**) to the agens interactive session will terminate the application. If set to a numeric value, many EOF characters will be ignored before the application terminates. If the variable is set but has no numeric value, the default is 10.

Note: This is a feature from bash.

LASTOID
The last-affected OID value is returned by INSERT or `\lo_import` command. This variable is only valid until the result of the next SQL command is displayed.

ON_ERROR_ROLLBACK
If this is set to `on` and the statement in the transaction block generates an error, the error will be ignored and the transaction will continue. When setting up an interactive session, these errors will be ignored only in the interactive session while the script file is read. If set to unset or `off`, the statement in the transaction block that generates the error will abort the entire transaction. The error rollback mode works by issuing an implicit `SAVEPOINT` immediately before each command in the transaction block and then rolling back to the save point if the command fails.

ON_ERROR_STOP
By default, command processing continues after an error. Setting this variable to `on` will stop immediate processing. In the interactive mode, agens returns to the command prompt. Otherwise, agens terminates, returns an error code 3, and distinguishes this case from the fatal error condition reported using an error code 1. In both cases, any scripts that are currently running (top-level scripts (if any) and any other scripts that can call other scripts) are terminated immediately. If the top-level command string contains multiple SQL commands, the processing is aborted with the current command.

PORT
This is the database server port you are currently connected to. Although this is set every time you connect to the database (including program startup), you may turn it off.

PROMPT1
PROMPT2

PROMPT3
Specifies how the agens results are displayed in the prompt. Refer to [Prompting] below.

QUIET
Setting this variable to on is equivalent to the command-line option `-q`. This is not very useful in the interactive mode.

SHOW_CONTEXT
This variable can be set to `never`, or `errors`, `always` to control whether the CONTEXT field should be displayed in the server's message. The default is `errors` (displayed in error messages but not in warning or warning messages). No effect if VERBOSITY is set to `terse` (see `\errverbose` for a more detailed version of the error you just received).

SINGLELINE
Setting this variable to on is equivalent to the command-line option `-S`.

SINGLESTEP
Setting this variable to on is equivalent to the command-line option `-S`.

USER
Currently connected database user. Although this is set every time you connect to the database (including program startup), you may turn it off.

VERBOSITY
This variable can be set to `default`, `verbose` , or `terse` to control the details of the error report. See `\errverbose` for a more detailed version of the error you just received.

SQL Interpolation

The core function of the agens variables is to be able to "interpolate" ordinary SQL statements as well as meta-command arguments. In addition, agens provides functions that allow SQL literals and variable values used as identifiers to be quoted appropriately. The syntax for interpreting the value without any quotes is prepended to the colon variable name (:). For example:

```
db=# \set foo 'my_table'
db=# SELECT * FROM :foo;
```

Query `my_table` table. This may not be safe since the value of a variable is literally copied and unbalanced quotes or backslash instructions may be contained. Make sure you know where you put it.

Enclosing the value in quotes when used as an SQL literal or identifier is the safest way. To quote a variable's value as an SQL literal, put the variable name in single quotation marks after the colon. To quote a value as an SQL identifier, write the variable name after the

colon in double quotation marks. This structure correctly processes quotes and other special characters contained within variable values. Here's an example that made the previous example safer.

```
db=# \set foo 'my_table'
db=# SELECT * FROM :"foo";
```

Variable interpolation is not performed within quoted SQL literals and identifiers. Therefore, a structure like `:foo` will not produce a literal quote from the value of the variable (it may not be safe because it does not correctly process quotes included in the value).

One example of using this mechanism is to copy a file's content into a table column. First, load the file into a variable and then interpolate the variable's value into a quoted string.

```
db=# \set content `cat my_file.txt`
db=# INSERT INTO my_table VALUES (:'content');
```

(This function does not work if `my_file.txt` contains NUL bytes. agens does not support embedded NUL bytes in variable values.)

Since a colon can appear legitimately in an SQL command, obvious attempts to interpolate (i.e. :name, :'name' or :"name") are not replaced unless the named variable is currently set. In any case, you can escape the colon with a backslash to protect it from the replacement code.

The colon syntax for variables is a standard SQL for built-in query languages such as ECPG. The colon syntax for array slices and type casts is an AgensGraph extension and may conflict with the standard usage. The colon-quote syntax for escaping the value of a variable to an SQL literal or identifier is an extension of agens.

Prompting

Agens can be customized by the user as prompted. The three variables `PROMPT1`, `PROMPT2`, and `PROMPT3` have a special escape sequence and a string that describes the appearance of the prompt. PROMPT1 is a generic prompt issued when agens requests a new command. PROMPT2 is issued when the command does not end with a semicolon or quotation marks and, therefore, more input is needed during command input. PROMPT3 is issued when the `SQL COPY FROM STDIN` command is being executed and the row value must be entered into the terminal.
The value of the selected prompt variable is printed as is, except when there is a percentage sign (`%`). Specific text is replaced by the following characters. The defined alternatives include:

%M
For connections over a UNIX domain socket, the entire hostname (including the domain name) of the database server or [`local`] is set to [`local`: ***/dir/name***] if the Unix domain socket is not in the default location.

%m
The hostname of the database server, truncated at the first dot, or [`local`] if the connection is over a Unix domain socket.

%>
The port number at which the database server is listening.

%n
The database session user name. (The expansion of this value might change during a database session as the result of the command `SET SESSION AUTHORIZATION`.)

%/
The name of the current database.

%~
Like %/, but the output is ~ (tilde) if the database is your default database.

%#
If the session user is a database superuser, then a #, otherwise a >. (The expansion of this value might change during a database session as the result of the command SET SESSION AUTHORIZATION.)

%p
The process ID of the backend is currently connected to.

%R
In prompt 1 normally =, but ^ if in single-line mode, or `!` if the session is disconnected from the database (which can happen if `\connect` fails).

In prompt 2, %R is replaced by a character that depends on why psql expects more input: - if the command simply wasn't terminated yet, but * if there is an unfinished `/* ... */` comment, a single quote if there is an unfinished quoted string, a double quote if there is an unfinished quoted identifier, a dollar sign if there is an unfinished dollar-quoted string, or (if there is an unmatched left parenthesis. In prompt 3, %R doesn't produce anything.

%l
The line number inside the current statement, starting from 1.

%[… %]
A prompt may include, for instance, terminal control characters that change the color,

background, or style of the prompt text, or change the title of the terminal window. In order for Readline's row editing to work properly, these nonprinting control characters must be enclosed in %[and %] to be invisible. Multiple pairs of options can occur within the prompt.

Here is an example:

```
db=# \set PROMPT1 '%[%033[1;33;40m%]%n@%/%R%[%033[0m%]%# '
```

A bold (1;), yellow-on-black (33;40) prompt appears on a VT100 compatible, color-capable terminal.

Use %% to insert a percentage sign at the prompt. The default prompt is %/%R%# for prompts 1 and 2 and >> for prompt 3.

> Note: This is a feature that comes from tcsh.

Command-Line Editing

Agens supports the Readline library for convenient line editing and searching. When you exit agens, the command history is automatically saved and reloaded when you (re)start agens. Tab completion is supported even if the completion logic does not request SQL parsing. Queries generated by tab-completion may conflict with other SQL commands (such as SET TRANSACTION ISOLATION LEVEL). If you do not like the tab completion, you may disable it by adding it to the home directory's `.inputrc` file.

```
$if agens
set disable-completion on
$endif
```

(This is a feature of Readline, not agens. Refer to the corresponding documentation for details.)

Environment

COLUMNS
If \pset columns are 0, you need to adjust the `wrapped` format and width or switch from the extended automatic mode to the vertical format, in order to determine whether the extensive output is required.

PAGER
If the query results do not fit on the screen, they are piped through this command. Typical values are `more` or `less`. The default value depends on the platform. You can disable `PAGER` by setting it to blank or using pager-related options of the `\pset` command.

PGDATABASE
PGHOST

PGPORT
PGUSER
Basic connection parameters (Reference).

PSQL_EDITOR
EDITOR
VISUAL
An editor is used for `\e`, `\ef`, and `\ev` commands. These variables are checked in the order listed and are used in the first set order.

The built-in default editor is `vi` on Unix systems and `notepad.exe` on Windows systems.

PSQL_EDITOR_LINENUMBER_ARG
When using `\e`, `\ef`, and `\ev` with a line number argument, these variables specify the command-line argument used to pass the start line number to the user's editor. For editors like Emacs or vi, this is a plus (+) sign. If there has to be a space between the option name and the line number, include the trailing space in the variable value. Here is an example:

```
PSQL_EDITOR_LINENUMBER_ARG='+'
PSQL_EDITOR_LINENUMBER_ARG='--line '
```

On Unix systems, the default is + (this corresponds to the default editor `vi` and is useful for many other common editors). However, there is no default on Windows systems.

PSQL_HISTORY
A tilde (~) expansion is performed as an alternative location to the command history file.

PSQLRC
A tilde (~) extension is performed as an alternative location to the user's `.psqlrc` file.

SHELL
Commands executed with `\!`.

TMPDIR
A directory to store temporary files (default is `/tmp`). This utility uses environment variables supported by libpq.

Files

`psqlrc` and `~/.psqlrc`
Agens reads and executes commands from the system-wide startup file (`psqlrc`) and your personal startup file (`~/.psqlrc`) before connecting to the database and accepting normal commands unless you specify the `-X` option. These files are typically used to set up clients and/or servers using the `\set` and `SET` commands.

The system-wide startup file is named `psqlrc` and can be found in the "System Configuration" directory of the installation. This directory is most reliably identified by running `pg_config --sysconfdir`. By default, this directory is `../etc/` for the directory containing the AgensGraph executable(s). The name of this directory can be explicitly set via the `PGSYSCONFDIR` environment variable.

The user's personal startup file name is `.psqlrc` and the calling user's home directory is searched. On Windows where such a concept is not available, the name of the personal startup file is `%APPDATA%\postgresql\psqlrc.conf`. The location of the user-startup file can be explicitly set via the `PSQLRC` environment variable.

`.psql_history`
command-line history is stored in `~/.psql_history`
or `%APPDATA%\postgresql\psql_history` file on Windows.

The location of the history file can be explicitly set via the PSQL_HISTORY environment variable.

Notes

Agens works best with major versions (old or new) of the server. The backslash command can fail, especially if the server is newer than agens itself. The generic ability to execute SQL commands and display query results should work on new major versions of the server, but cannot be guaranteed in every case. In the case where you want to use agens and try to connect an agens program to multiple servers of other major versions, you had better use the latest version. Alternatively, you should keep a copy of each major version of agens and use the version that matches the server.

Notes for Windows Users

Agens is built as a "console application". Be especially cautious when using 8-bit characters in agens because the Windows console window uses a different encoding from the rest of the system. When agens searches for a problematic console code page, a warning message is displayed at startup. To change the console code page, you need two things as follows:

- Enter and set `cmd.exe /c chcp 1252` code page (1252 is a code page suitable for German; enter an appropriate code page). If you are using Cygwin, you can put this command in /etc/profile.

- Since raster fonts do not work with ANSI code pages, set the console font to `Lucida Console`.

Examples

The first example (below) shows how to distribute commands over multiple lines of input.

```
agens=# CREATE TABLE my_table (
agens(#  first integer not null default 0,
agens(#  second text)
agens-# ;
CREATE TABLE
```

Check the definition of the table.

```
agens=# \d my_table
             Table "my_table"
 Attribute |  Type   |      Modifier
-----------+---------+--------------------
 first     | integer | not null default 0
 second    | text    |
```

You can change the prompt as follows:

```
agens=# \set PROMPT1 '%n@%m %~%R%# '
tester@[local] agens=#
```

Populate the table with data and query the table.

```
tester@[local] agens=#  SELECT * FROM my_table;
 first | second
-------+--------
     1 | one
     2 | two
     3 | three
     4 | four
(4 rows)
```

You can use the \pset command to display tables in different ways.

```
tester@[local] agens=# \pset border 2
Border style is 2.
tester@[local] agens=# SELECT * FROM my_table;
+-------+--------+
| first | second |
+-------+--------+
|     1 | one    |
|     2 | two    |
|     3 | three  |
|     4 | four   |
```

```
+-------+--------+
(4 rows)

tester@[local] agens=# \pset border 0
Border style is 0.
tester@[local] agens=# SELECT * FROM my_table;
first second
----- ------
    1 one
    2 two
    3 three
    4 four
(4 rows)

tester@[local] agens=# \pset border 1
Border style is 1.
tester@[local] agens=# \pset format unaligned
Output format is unaligned.
tester@[local] agens=# \pset fieldsep ,
Field separator is ",".
tester@[local] agens=# \pset tuples_only
Showing only tuples.
tester@[local] agens=# SELECT second, first FROM my_table;
one,1
two,2
three,3
four,4
```

Or you can use a short command.

```
tester@[local] agens=# \a \t \x
Output format is aligned.
Tuples only is off.
Expanded display is on.
tester@[local] agens=# SELECT * FROM my_table;
-[ RECORD 1 ]-
first  | 1
second | one
-[ RECORD 2 ]-
first  | 2
second | two
-[ RECORD 3 ]-
first  | 3
second | three
-[ RECORD 4 ]-
```

```
first  | 4
second | four
```

If appropriate, you can use the \crosstabview command to display the query results in a crosstab representation.

```
db=# SELECT first, second, first > 2 AS gt2 FROM my_table;
 first | second | gt2
-------+--------+-----
     1 | one    | f
     2 | two    | f
     3 | three  | t
     4 | four   | t
(4 rows)

db=# \crosstabview first second
 first | one | two | three | four
-------+-----+-----+-------+------
     1 | f   |     |       |
     2 |     | f   |       |
     3 |     |     | t     |
     4 |     |     |       | t
(4 rows)
```

This second example shows a multiplication table in which the numeric sequence is sorted in reverse order and the columns are in ascending numerical order.

```
agens=# SELECT t1.first as "A", t2.first+100 AS "B", t1.first*(t2.first+100)
as "AxB",
agens-# row_number() over(order by t2.first) AS ord
agens-# FROM my_table t1 CROSS JOIN my_table t2 ORDER BY 1 DESC
agens-# \crosstabview "A" "B" "AxB" ord
 A | 101 | 102 | 103 | 104
---+-----+-----+-----+-----
 4 | 404 | 408 | 412 | 416
 3 | 303 | 306 | 309 | 312
 2 | 202 | 204 | 206 | 208
 1 | 101 | 102 | 103 | 104
(4 rows)
```

pg_dump

pg_dump is a utility that backs up the AgensGraph database. Performs consistent backups even while the database is in use. pg_dump extracts it into a script file or other archive file.

Here is how to use:

```
$ pg_dump [option]... [dbname]
```

Options

dbname
Specifies the name of a database to dump. If not specified, the username specified in the connection will be used.

-a
–data-only
Dumps only data, not schema (data definition).

-b
–blobs
Includes a large object in the dump. This is the default behavior unless `--schema`, `--table`, or `--schema-only` is specified. Therefore, the `-b` switch is useful only if a particular schema or table adds a large object to the requested dump. The blob is regarded as data, so it is included when `--data-only` is used, but not included when it is not `--schema-only`.

-c
–clean
Prints a command to clean up (delete) database objects before issuing a command to create a database object. (If `--if-exists` is not specified and the object does not exist in the target database, a false error message may be generated.)

This option is meaningful only in plain text format. For archive formats, you can specify options when you call
`pg_restore`.

-C
–create
Starts the output with a command that creates a database itself and reconnects to the created database. (If you use this form of script, the database of the target installation you want to connect to is not important before you run the script.) If `--clean` is also specified, the script deletes the target database and then reconnects.

This option is meaningful only in plain text format. For archive formats, you can specify options when you call
`pg_restore`.

-E ***encoding***
–encoding=***encoding***
Creates a dump with the specified character set encoding. By default, dumps are written in

database encodings. (Another way to achieve the same result is to set the `PGCLIENTENCODING` environment variable to the desired dump encoding.)

-f ***file***
–file=***file***
Sends the output to the specified file. This parameter can be omitted in a file-based output format where standard output is used. However, it should be provided for a directory output format that specifies a target directory instead of a file. In such a case, a directory (that has not existed) is created by `pg_dump`.

-F ***format***
–format=***format(c|d|t|p)***
Selects an output format. the format can be one of the following:

- p
  plain
  Prints plain-text SQL script files (default).

- c
  custom
  Prints user-defined type archives suitable for `pg_restore` input. Along with the directory output format, this option is the most flexible output format that allows you to manually select the items kept during a restore and change their sequence. This format is compressed by default.

- d
  directory
  Prints user-defined type archives suitable for `pg_restore` input. This will create each table, a file that will dump blob, and a directory that contains a so-called table of contents file that describes the objects dumped in a machine-readable format that can be read by `pg_restore`. Directory-type archives can be manipulated with standard Unix tools. For instance, files in uncompressed archives can be compressed with gzip. This format is compressed by default and supports parallel dumps.

- t
  tar
  Prints a tar-format archive suitable for input to `pg_restore`. The `tar` format is compatible with the directory format. Extracting a tar format archive creates a valid directory format archive. However, the tar format does not support compression. With the tar format, the relative order of table data items cannot be changed during a restore.

-j ***njobs***
–jobs=***njobs***

Runs a parallel dump by dumping the ***njobs*** tables at the same time. This option reduces the dump time, but increases the load on the database server. As this option is the only output format that allows multiple processes to write data at the same time, it can be used only in the directory output format.

Since `pg_dump` will open an ***njobs***+1 connection to the database, the max_connections setting should be high enough to accommodate all connections.

If you request an exclusive lock on a database object while executing a parallel dump, the dump may fail. This is because the `pg_dump` master process requires the worker process to request a shared lock on the object to dump later so that no one deletes it while the dump is running. If another client requests an exclusive lock on the table, the lock is not granted, but it waits until the shared lock of the master process is released. As a result, no other access to the table are granted and they wait after the exclusive lock request. This includes the worker process that tries to dump the table. If there are no precautions, a classical deadlock occurs. To detect this conflict, the `pg_dump` worker process requests another shared lock using the NOWAIT option. If this shared lock is not granted to the worker process, someone else has to request an exclusive lock; as there is no way to continue the dump, `pg_dump` will have to stop dumping.

For consistent backup, the database server must support synchronized snapshots. With this feature, database clients can see the same dataset even if they use different connections. `pg_dump -j` uses multiple database connections. It connects to the database once in the master process and connects to each worker job again. Without the synchronized snapshot feature, inconsistent backups may occur because different worker jobs cannot see the same data on each connection.

-n ***schema***
–schema=***schema***
Dumps only ***schema*** that matches the schema. This selects both the schema itself and all objects contained in the schema. If this option is not specified, all non-system schemas in the target database are dumped. Multiple schemas can be selected by creating multiple `-n` switches. Schema parameters are also interpreted in the same pattern as the rules used by agens' `\d` command, and wildcard characters can be used to select multiple schemas. When using wildcards, use quotation marks to prevent the shell from extending wildcards (refer to Examples).

> Notice: If `-n` is specified, `pg_dump` will not dump any other database objects that may vary depending on the schema selected. Therefore, there is no guarantee that the result of a particular schema dump can be restored to a normal clean database by itself.
>
> Notice: Non-schema objects such as blob are not dumped if `-n` is specified. You can add blob back to the dump using the `--blobs` switch.

-N ***schema***
–exclude-schema=***schema***
Do not dump ***schema*** that matches the ***schema*** pattern. Patterns are interpreted according to the same rules as `-n`. `-N` can be used more than once to exclude schemas that match one of several patterns.

Given both `-n` and `-N`, this operation dumps schemas that match at least one `-n` switch, except for the `-N` switch. If `-N` appears without `-n`, schemas that match `-N` are excluded from the normal dump.

-o
–oids
Dumps the object identifier (OID) into part of the data in all tables. Use this option if your application refers to an OID column in some ways (e.g. foreign key constraint). Otherwise, this option is not available.

-O
–no-owner
Do not print commands that set ownership of an object to match the original database. By default, `pg_dump` issues an `ALTER OWNER` or `SET SESSION AUTHORIZATION` statement to set ownership of the created database object. This command will fail when the script is run unless the superuser (or the same user who owns all objects in the script) starts this command. Specify -O to create a script that grants ownership of all objects only to an applicable user, but can be restored by any user.

This option is meaningful only in plain text format. For archive formats, you can specify options when you call
`pg_restore`.

-s
–schema-only
Dumps object definitions (schemas) only.

This option is the opposite of `--data-only`. This is similar to `--section=pre-data --section=post-data`.

(Do not confuse this with the `--schema` option, which uses the word “schema” in other ways.)

To exclude table data only for a subset of the tables in the database, see `--exclude-table-data`.

-S ***username***
–superuser=***username***

Specifies the name of superuser to use when not using the trigger. This option is relevant only if `--disable-triggers` is used (generally it is better to skip this step; instead you should start the results with a supercomputer).

-t ***table***
–table=***table***
Dumps only a ***table*** whose name matches the table. To this end, “table” includes views, materialized views, sequences, and foreign tables. You may create multiple `-t` switches to select multiple tables. As the ***table*** parameter is interpreted in the same pattern as the agens `\d` command, you can select multiple tables using wildcard characters in the pattern. To prevent the shell from extending wildcards, use quotes (refer to Examples).

The `-n` and `-N` switches do not work when using `-t`, because the tables selected with `-t` are dumped regardless of the switches and non-table objects are not dumped.

> Notice: If you specify `-t`, pg_dump will not attempt to dump other database objects on which the selected table depends. Therefore, there is no guarantee that the results for a particular database can be successfully restored to its own database.

-T ***table***
–exclude-table=***table***
Do not dump tables that match the ***table*** pattern. This pattern is interpreted according to the same rules as `-t`. `-T` can be given more than once, except for tables that match one of the various patterns.

When both `-t` and `-T` are given, this action dumps tables that match at least one `-t` switch, except for the `-T` switch. If `-T` is used without `-t`, tables that match `-T` are excluded from normal dumps.

-v
–verbose
Specifies the verbose mode. This will cause pg_dump to print the detailed object description and start/stop times to a dump file and process the message as a standard error.

-V
–version
Displays the version of pg_dump and exits.

-x
–no-privileges
–no-acl
Prevents dumping of access privileges (grant/revoke command).

-Z ***0..9***
–compress=***0..9***
Specifies the compression level to use. “0” means no compression. For the user-defined archive format, this specifies compression of individual table data segments, the default is to compress at a medium level. For plain text output, setting a nonzero compression level compresses the entire output file as supplied by gzip (but, the default is not compressed). The tar archive format currently does not support compression at all.

–binary-upgrade
This option is used by the full upgrade utility. Using this option for other purposes is not recommended or supported. The behavior of this option may change in future releases without notice.

–column-inserts
Dumps data into an `INSERT` statement with an explicit column name (INSERT INTO ***table*** (***column***, …) VALUES …). This will greatly slow down the restoration. This option is primarily useful for creating dumps that can be loaded into a non-Agens database. This option generates a separate command for each row, so if an error occurs while reloading a row, only that row (not the entire table content) is lost.

–disable-dollar-quoting
This option disables the use of dollar quotes for the function body and allows the use of quotation marks using SQL standard string syntax.

–disable-triggers
This option is relevant only when creating a data-only dump. Instructs `pg_dump` to include a command to temporarily disable trigger on the target table while data is being reloaded. Use this option in case of referential integrity check in which you do not want a call while reloading data or in the case where there are other triggers on the table.

Currently, the commands generated for `--disable-triggers` must be run by the superuser. Therefore, be careful when you specify the superuser name with -S or start the resulting script as the superuser.

This option is meaningful only in plain text format. For archive formats, you can specify options when you call
`pg_restore`.

–enable-row-security
This option is relevant only when dumping the content of a table with row security. By default, `pg_dump` sets
`row_security` to off so that all data is dumped from the table. An error occurs if the user does not have sufficient privileges to bypass row security. This parameter allows `pg_dump` to dump some of the user-accessible table content by setting `row_security` to on.

–exclude-table-data=***table***
Do not dump data for a table that matches the ***table*** pattern. `--exclude-table-data` can be supplied more than once to exclude tables that match one of several patterns. This option is useful when you need to define a specific table, even if you do not need the data.

To exclude data for all tables in the database, refer to `--schema-only`.

–if-exists
Use conditional commands (such as the IF EXISTS clause) to clean up database objects. This option is not valid unless `--clean` is specified.

–inserts
Dumps data with `INSERT` command instead of `COPY`. This will greatly slow down the restoration speed. This option is primarily useful for creating dumps that can be loaded into a non-Agens database. This option generates a separate command for each row, so if an error occurs while reloading a row, only that row (not the entire table content) is lost. Rearranging the column order can cause the restore to fail. The `--column-insert` option is slow, but can be used safely when changing the column order.

–lock-wait-timeout=***timeout***
Does not wait forever to acquire the shared table lock at the beginning of the dump. It will fail if the table cannot be locked within the specified ***timeout***. The timeout can be specified in the format allowed by the `SET statement_timeout` (allowed values are integers in milliseconds).

–no-security-labels
Does not dump the security label.

–no-synchronized-snapshots
This option allows you to run `pg_dump -j`. Refer to the description of the `-j` parameter for details.

–no-tablespaces
Does not print command to select tablespaces. When this option is used, all objects are created in the table realm, which is the default value during restoration.

This option is meaningful only in plain text format. For archive formats, you can specify options when you call
`pg_restore`.

–no-unlogged-table-data
Does not dump the content of unlogged tables. This option does not affect whether the table definition (schema) is dumped or not. Limit only dumping of table data. Data in non-logged tables is always excluded when dumping from the standby server.

–quote-all-identifiers
Forces all identifiers to be quoted. This option is recommended when the main version of AgensGraph dumps the database on a different server from `pg_dump`, or when you load the output on another major version of the server. By default, `pg_dump` quotes only identifiers that are reserved words in its major version. This may cause a compatibility problem when processing different versions of the server with different versions of the reserved word set. You can use `--quote-all-identifier` to avoid this problem instead of using a hard-to-read dump script.

–section=***section***
Dumps only the named section. A ***section*** can be pre-data, data, or post-data. This option can be specified more than once to select multiple sections. The default is to dump all sections.

The data section includes actual table data, large-volume object content, and sequence values. The post-data entry contains the definitions of indexes, triggers, rules, and constraints (not validity checking constraints). The pre-data entry contains all other data definition entries.

–serializable-deferrable
Uses a `serializable` transaction in dumps to ensure that the used snapshot matches the state of the database at a later time. This option ensures there is no risk of a dump failing or another transaction being rolled back to serialization_failure by waiting for a point in the transaction stream for which no exception is present.

This option is not useful for disaster recovery-only dumps. It can be useful to load a database copy while the database is still being updated or to load a database copy for another read-only load sharing. Without it, the dump can eventually reflect a condition that is inconsistent with any subsequent execution of the committed transaction. For instance, you may use batch processing techniques to mark batches as closed without displaying all the items in the batch.

This option makes no difference if there is no active read/write transaction when `pg_dump` starts. If a read-write transaction is active (enabled), the start of a dump can be delayed for an indeterminate amount of time. Once executed, performance will be the same regardless of the use of switches.

–snapshot=***snapshot***
Uses a specified synchronized snapshot when dumping the database.

This option is useful when you need to synchronize a dump with a logical replication slot or a concurrent session.

For a parallel dump, the snapshot name defined by this option is used instead of creating a new snapshot.

–strict-names
Requires that each schema (`-n/--schema`) and table (`-t/--table`) qualifiers match at least one schema/table in the database to be dumped. Without a matching schema/table qualifier, pg_dump generates an error without `--strict-names`.

This option does not affect `-N/--exclude-schema`, `-T/--exclude-table`, `--exclude-table-data`. An exclusion pattern that does not match an object is not regarded as an error.

–use-set-session-authorization
Prints the SQL standard `SET SESSION AUTHORIZATION` command, instead of the `ALTER OWNER` command, to identify the object ownership. By doing so, the dump can be compliant with the standard, but may not be restored properly depending on the record of the object in the dump. In addition, dumps using `SET SESSION AUTHORIZATION` require that superuser privileges be restored correctly, but ALTER OWNER requires fewer privileges.

-?
–help
Displays help for pg_dump command-line arguments and exits.

The following command-line options control database connection parameters.

-d ***dbname***
–dbname=***dbname***
Specifies the name of the database to connect to. This is equivalent to specifying ***dbname***as an argument, not the first option on the command-line.

If this parameter has an equal (=) sign or starts with a valid URI prefix (postgresql:// or postgres://), it is processed as a conninfo string.

-h ***host***
–host=***host***
Specifies the hostname of the system on which the server is running. If the value starts with a slash, it is used as the directory for Unix domain sockets. The default value is taken from the PGHOST environment variable (if set). Otherwise, a Unix domain socket connection is attempted.

-p ***port***
–port=***port***
Specifies the TCP port or local Unix domain socket file extension on which the server listens for connections. The default value is the PGPORT environment variable. It should be set to the (compiled) default value.

-U ***username***
–username=***username***
The name of the user to connect to.

-w
–no-password
Does not prompt for a password. If the server requires password authentication and cannot use the password in any other way, such as in a `.pgpass` file, the connection attempt will fail. This option can be useful in batch jobs and scripts where there is no user who may enter a password.

-W
–password
Before connecting to the database, run `pg_dump` to enter the password.

As `pg_dump` automatically requires a password when the server requests password authentication, this option is never needed. However, `pg_dump` wastes connection attempts to find out if the server wants a password. In some cases, it is better to type -W to avoid extra connection attempts.

–role=***rolename***
Specifies the role name to be used to create a dump. With this option, `pg_dump` issues the SET ROLE ***rolename*** command after connecting to the database. This is useful when an authenticated user (specified by -U) lacks the privileges required by `pg_dump`, but has the authority to switch to the required privilege. If you have a policy that prevents you from logging in directly as the superuser, you may use this option to generate a dump without violating the policy.

Examples

To dump a database called `mydb` into a SQL script file:

```
$ pg_dump mydb > db.sql
```

To reload a script into a newly created database named `newdb`:

```
$ agens -d newdb -f db.sql
```

To dump a database to a user-defined type archive file:

```
$ pg_dump -Fc mydb > db.dump
```

To dump a database to a directory format archive:

```
$ pg_dump -Fd mydb -f dumpdir
```

To dump a database into a directory format archive in parallel with five worker jobs:

```
$ pg_dump -Fd mydb -j 5 -f dumpdir
```

To reload an archive file into a newly created database called `newdb`:

```
$ pg_restore -d newdb db.dump
```

To dump a single table called `mytab`:

```
$ pg_dump -t mytab mydb > db.sql
```

To dump all the tables in the `detroit` schema that start with the name `emp` except for the table named `employee_log`:

```
$ pg_dump -t 'detroit.emp*' -T detroit.employee_log mydb > db.sql
```

To dump all schemas whose names start with `east` or `west` and end with `gsm`, and exclude schemas whose names contain `test`:

```
$ pg_dump -n 'east*gsm' -n 'west*gsm' -N '*test*' mydb > db.sql
```

or

```
$ pg_dump -n '(east|west)*gsm' -N '*test*' mydb > db.sql
```

To dump all database objects except tables whose names start with `ts_`:

```
$ pg_dump -T 'ts_*' mydb > db.sql
```

You must enclose the name in double quotes to specify a name that mixes uppercase or mixed-case characters in the `-t` and related switches. Otherwise, it is converted to lowercase. However, as double quotes apply only to shells, you must enclose them in quotes. Thus, to dump a single table with a mixed case, you should do the following:

```
$ pg_dump -t "\"MixedCaseName\"" mydb > mytab.sql
```

pg_restore

`pg_restore` is a utility that restores the AgensGraph database to an archive created by `pg_dump`. Executes commands necessary to reconstruct the database at the time the database was saved. The archive file also allows `pg_restore` to change the order of items before selecting or restoring them. The archive file is designed to be applicable to the overall architecture.

`pg_restore` can operate in two modes. If a database name is specified, `pg_restore` connects to the specified database and restores the archived content directly to the database. Otherwise, a script containing the SQL commands required to rebuild the

database is written and saved (recorded) to a file or standard output. This script output is identical to the plain text output format of `pg_dump`. Thus, some of the options that control output are similar to the `pg_dump` option.

Of course, `pg_restore` cannot restore information that is not in the archive file. For example, if you created an archive using the "data dump with INSERT command" option, `pg_restore` cannot load data using the COPY statement.

Here is how to use:

```
$ pg_restore [OPTION]... [FILE]
```

Options

`pg_restore` accepts the following command-line arguments:

filename
Specifies the location of the archive file to be restored (or directory for directory-format archives). If not specified, standard input is used.

-a
–data-only
Restore only data that is not the schema (data definition). Table data, large objects, and sequence values are restored if they are in the archive.

This option is similar to specifying `--section=data`.

-c
–clean
Drops a database object before regenerating it. (If you do not use `--if-exists`, a harmless error message may be generated when there is no object in the target database.)

-C
–create
Creates a database before restoring. If specified with `-clean`, drop, and recreate the target database before connection.

With this option, the database named `-d` is used only to execute the initial `DROP DATABASE` and `CREATE DATABASE` commands. All data is restored to the database name that appears in the archive.

-d ***dbname***
–dbname=***dbname***
Connects to the database ***dbname*** and restores it directly to the database.

-e
–exit-on-error
If an error occurs while sending an SQL command to the database, it terminates. The default is to continue and display the number of errors at the end of the restore.

-f ***filename***
–file=***filename***
Specifies the output file of the generated script or, if used with `-l`, specifies the file to list. The default is standard output.

-F ***format***
–format=***format***
Specifies the archive format. Specifying this format is not required as `pg_restore` automatically determines it. If specified, it can be one of the following:

- c
  custom
  A user-defined format of `pg_dump`.
- d
  directory
  A directory archive.
- t
  tar
  A tar archive.

-I ***index***
–index=***index***
Only definitions of the named indexes are restored. Multiple indexes can be specified with multiple -I switches.

-j ***number-of-jobs***
–jobs=***number-of-jobs***
Executes the most time-consuming part of `pg_restore` as multiple concurrent tasks are used to load data, create indexes, or create constraints. This option greatly reduces the time to restore a large database to a server running on a multiprocessor system.

Each task is a process or thread, depending on the operating system, and uses a separate connection to the server.

The optimal value for this option varies depending on the hardware settings of the server, client, and network (e.g. the number of CPU cores, disk configuration). If you set a larger

value than the number of CPU cores in the server, you can shorten the restore time in most cases. A too high value may cause performance degradation due to memory thrashing.

Only the user-defined and directory archive formats are supported in this option. The input must be a regular file or directory. This option is ignored when exporting scripts without connecting directly to the database server. You cannot use multiple jobs with the `--single-transaction` option.

-l
–list
Lists the contents of the archive. The output of this operation can be used as input to the `-L` option. Use a filtering switch such as `-n` or `-t` with `-l` to restrict the listed items.

-L ***list-file***
–use-list=***list-file***
Restores only the archive elements in ***list-file*** in the order they are listed in the file. If you use a filtering switch such as `-n` or `-t` with `-L`, you further restrict what is restored.

list-file is usually created by editing the output of the previous `-l` operation. Lines can be moved or removed, and annotations can be used with a semicolon (;) at the beginning of the line.

-n ***namespace***
–schema=***schema***
Restores only the objects in the named schema. Multiple schemas can be specified with a multiple `-n` switch. You can combine this option with `-t` to restore only specific tables.

-O
–no-owner
Does not print commands that set ownership of the objects so that they match the original database. By default, `pg_restore` sets ownership of schema elements created through the `ALTER OWNER` or `SET SESSION AUTHORIZATION` statement. This statement fails if the superuser (or the same user who owns all objects in the script) does not make an initial connection to the database. Using -O allows all usernames to be used for initial connections; the user will own all the objects created.

-P ***function-name(argtype [, ...])***
–function=***function-name(argtype [, ...])***
Restores only named functions. Be careful with the spelling of the function name and arguments as shown in the table of contents of the dump file. Multiple functions can be specified using a multiple `-P` switch.

-s
–schema-only

Restores only the schema (data definition) to the extent of schema entry in the archive (not in the data).

This option is the opposite of `--data-only`. It is similar to specifying `--section=pre-data`,`--section=post-data`.

(Do not confuse this with the `--schema` option, which uses the word “schema” in other ways.)

-S ***username***
–superuser=***username***
Specifies the superuser name to use when disabling trigger. This is only relevant if `--disable-triggers` are used.

-t ***table***
–table=***table***
Restores the definition or data of a named table. For this purpose, “table” includes views, materialized views, sequences, and foreign tables. Multiple tables can be selected by writing multiple -t switches. This option can be combined with the `-n` option to specify a table in a particular schema.

> Note: If you specify `-t`, `pg_restore` does not attempt to restore any other database objects on which the selected table depends. Therefore, there is no guarantee that the result of a particular schema dump can be restored to a normal clean database by itself.
>
> Note: This flag does not work in the same way as the `-t` flag of `pg_dump`. There is currently no provision for wildcard matching in `pg_restore`, and you cannot include the schema name in `-t`.

-T ***trigger***
–trigger=***trigger***
Restores only named triggers. Multiple triggers can be specified as multiple `-T` switches.

-v
–verbose
Specifies the verbose mode.

-V
–version
Displays the version of `pg_restore` and exits.

-x
–no-privileges

–no-acl
Prevents restoration of access privileges (grant/revoke command).

-1
–single-transaction
Runs a restore in a single transaction (i.e. wrap the command released by BEGIN/COMMIT). By doing this, all commands are completed successfully or the changes do not take effect. This option means `--exit-on-error`.

–disable-triggers
This option is relevant only when performing a data-only restore. This instructs `pg_restore` to temporarily disable trigger on the target table while executing a command to reload the data. Use this option in case of referential integrity check in which you do not want a call while reloading data or in the case where there are other triggers on the table.

Currently the command for `--disable-trigger` must be run as superuser. Thus, specify the superuser name with `-S`, or run `pg_restore` as the AgensGraph superuser.

–enable-row-security
This option is relevant only when restoring the content of a table with row security. By default, `pg_restore` sets `row_security` to off so that all data is restored to the table. An error occurs if the user does not have sufficient privileges to bypass row security. This parameter allows `pg_restore` to restore the content of the table with row security by setting `row_security` to on. This operation may fail even if the user is not authorized to insert rows from the dump into the table.

As `COPY FROM` does not support row security, the current dump should be an `INSERT` type in this option.

–if-exists
Uses conditional commands (such as the IF EXISTS clause) to clean up database objects. This option is not valid unless –clean is specified.

–no-data-for-failed-tables
Basically, table data creation is restored even if the table creation command fails. If you use this option, data in those tables are skipped. This is useful if you already have content in the table you want in the target database. For example, auxiliary tables for PostgreSQL extensions, such as PostGIS, may already be loaded into the target database. Specifying this option prevents duplicate or obsolete data from loading.

This option is valid only when restoring directly to the database, not when generating a SQL script output.

–no-security-labels
The command to restore the security label does not print the command even if it is included in the archive.

–no-tablespaces
Does not print the command to select tablespaces. When this option is enabled, all objects are created in all tablespaces that are the default during a restore.

–section=***section***
Restores only the named sections. Section names can be pre-data, data, or post-data. This option can be specified more than once to select multiple sections. The default is to restore all sections.

The data section contains actual table data as well as large object definitions. A post-data item consists of the definitions of indexes, triggers, rules, and constraints (not validity check constraints). The Pre-data item consists of all other data definition entries.

–strict-names
Each schema (-n-schema-schema) and table (-T-table) qualifiers need to match one or more schemas/tables with the backup file.

–use-set-session-authorization
Prints the SQL standard `SET SESSION AUTHORIZATION` command, instead of the ALTER OWNER command, to identify object ownership. By doing this, the dump can be compliant with the standard, but may not be restored properly depending on the record of the object in the dump.

-?
–help
Displays help for `pg_restore` command-line arguments and exits.

`pg_restore` also accepts the following command-line arguments for connection parameters:

-h ***host***
–host=***host***
Specifies the hostname of the system on which the server is running. If the value starts with a slash, it is used as a directory for Unix domain sockets. The default value is taken from the PGHOST environment variable (if set). Otherwise, a Unix domain socket connection is attempted.

-p ***port***
–port=***port***
Specifies the TCP port or local Unix domain socket file extension on which the server listens for connections. The default value is the PGPORT environment variable and should be set to the (compiled) default.

-U ***username***
–username=***username***
The name of the user to connect to.

-w
–no-password
Does not prompt for a password. If the server requires password authentication and cannot use the password in any other way, such as in a `.pgpass` file, the connection attempt will fail. This option can be useful in batch jobs and scripts where there is no user who can enter the password.

-W
–password
Before connecting to the database, run `pg_restore` to enter the password.

As `pg_restore` automatically requires a password when the server requests password authentication, this option is never needed. However, `pg_restore` wastes connection attempts to find out if the server wants a password. In some cases, it is better to type `-W` to avoid extra connection attempts.

–role=***rolename***
Specifies the role name to be used to create a dump. With this option, `pg_restore` issues the SET ROLE ***rolename*** command after connecting to the database. This is useful when an authenticated user (specified by -U) lacks the privileges required by `pg_restore`, but has the authority to switch to the required privilege. If you have a policy that prevents you from logging in directly as the superuser, you may use this option to generate a dump without violating the policy.

Examples

Suppose you have dumped a database called `mydb` into a dump file in a user-defined format.

```
$ pg_dump -Fc mydb > db.dump
```

To delete the database and re-create the database in the dump:

```
$ dropdb mydb
$ pg_restore -C -d postgres db.dump
```

The database named in the `-d` switch can be any database in the cluster. `pg_restore` uses it to run the CREATE DATABASE command for `mydb`. With `-C`, the data is always restored to the database name that appears in the dump file.

To reload a dump into a new database called newdb:

```
$ createdb -T template0 newdb
$ pg_restore -d newdb db.dump
```

Connect directly to the database to be restored without using `-C`. Replicate the new database from `template0`, not `template1`, to make sure it is initially empty.

To rearrange database entries, you must first dump the table of contents of the archive.

```
$ pg_restore -l db.dump > db.list
```

The list file consists of a header and a line for each entry as follows (example):

```
;
; Archive created at Mon Sep 14 13:55:39 2009
;     dbname: DBDEMOS
;     TOC Entries: 81
;     Compression: 9
;     Dump Version: 1.10-0
;     Format: CUSTOM
;     Integer: 4 bytes
;     Offset: 8 bytes
;     Dumped from database version: 8.3.5
;     Dumped by pg_dump version: 8.3.8
;
;
; Selected TOC Entries:
;
3; 2615 2200 SCHEMA - public pasha
1861; 0 0 COMMENT - SCHEMA public pasha
1862; 0 0 ACL - public pasha
317; 1247 17715 TYPE public composite pasha
319; 1247 25899 DOMAIN public domain0 pasha
```

The semicolon marks the beginning of a comment, and the number at the beginning of the line indicates the internal archive ID assigned to each entry.

You can annotate, delete, and resort lines in a file. Here is an example:

```
10; 145433 TABLE map_resolutions postgres
;2; 145344 TABLE species postgres
;4; 145359 TABLE nt_header postgres
6; 145402 TABLE species_records postgres
;8; 145416 TABLE ss_old postgres
```

It can be used as input to `pg_restore` and only entries 10 and 6 are restored in order.

```
$ pg_restore -L db.list db.dump
```

Server Tool

ag_ctl

ag_ctl is a utility that initializes the AgensGraph cluster and starts, stops, or restarts the database server, or displays the status of the running server. You can start the server manually, but ag_ctl performs operations such as reconnecting the log output and properly detaching it from the terminal/process groups. It also provides convenient options for controlled termination.

Here is how to use:

```
ag_ctl init[db]               [-D DATADIR] [-s] [-o "OPTIONS"]
ag_ctl start   [-w] [-t SECS] [-D DATADIR] [-s] [-l FILENAME] [-o "OPTIONS"]
ag_ctl stop    [-W] [-t SECS] [-D DATADIR] [-s] [-m SHUTDOWN-MODE]
ag_ctl restart [-w] [-t SECS] [-D DATADIR] [-s] [-m SHUTDOWN-MODE]
               [-o "OPTIONS"]
ag_ctl reload  [-D DATADIR] [-s]
ag_ctl status  [-D DATADIR]
ag_ctl promote [-D DATADIR] [-s]
ag_ctl kill    SIGNALNAME PID
```

init or initdb mode creates a new AgensGraph database cluster. A database cluster is a collection of databases managed on a single server instance. This mode invokes the initdb command.

The start mode starts a new server. The server starts in the background and standard input is connected to /dev/null (nul on Windows). In Unix-like systems, the server's standard output and standard error are sent to the standard output (not standard error) of ag_ctl. You should then redirect the standard output of ag_ctl to a file or export it to another process, such as rotatelogs, a log rotation program. On Windows, the server's standard output and standard error are sent to the terminal by default. This default behavior can be changed by adding the server's output to a log file using -l. It is recommended to use either -l or output redirection.

The stop mode terminates the server running in the specified data directory. The -m option allows you to choose between three different shutdown methods. "Smart" mode waits until all active clients are disconnected and online backup is complete. If the server is in a hot standby state, recovery and streaming replication are terminated when all clients are disconnected. "Fast" mode (default) will end the ongoing online backup without waiting for the client to be disconnected. All active transactions will be rolled back and the server will shut down if the client is forced to disconnect. "Immediate" mode immediately aborts all server processes. This will cause crash-recovery to run upon restart.

The `restart` mode effectively executes stop and then executes start. This allows you to change the agens command-line option. The restart may fail if the relative path specified on the command-line is entered during server startup. The `reload` mode simply sends a SIGHUP signal to the agens process to read the configuration files (postgresql.conf, pg_hba.conf, etc.) again. Therefore, you may change configuration file options that do not require a full restart.

The `status` mode verifies that the server is running in the specified data directory. If so, the PID and command-line options used to invoke it will be displayed. If the server is not running, the process returns exit status 3. If an accessible data directory is not specified, the process returns exit status 4.

In the `promote` mode, it instructs the standby server running in the specified data directory to terminate recovery and start the read/write operations.

Use the `kill` mode to send a signal to the specified process. This is especially useful on Windows without the kill command. For a list of supported signal names, use –help.

Options

-c
–core-file
The server releases the soft resource limit stored in the core file to allow creation of the core files. This is useful for debugging or diagnosing problems by allowing the failed server process to obtain stack traces.

-D ***datadir***
–pgdata ***datadir***
Specifies the file system locations of the database configuration files. If omitted, the environment variable AGDATA will be used.

-l ***filename***
–log ***filename***
Adds the server log output to ***filename***. If the file does not exist, a new file with the filename is created. As umask is set to 077, other users are now allowed to access the log file by default.

-m ***mode***
–mode ***mode***
Specifies the shutdown mode. ***mode*** can be the first letter of one of `smart`, `fast`, and `immediate`. If not set, `fast` will be used.

-o ***options***
Specifies the options to pass to the `agens` command. Multiple option calls are added.

The options are usually enclosed in single or double quotes so they can be passed in groups.

-o ***initdb-options***
Specifies the options to pass directly to the `initdb` command.

The options are usually enclosed in single or double quotes so they can be passed in group.

-p ***path***
Specifies the location of the `agens` executable. By default, the `agens` executable runs in the same directory as `ag_ctl` or fails in the hard-wired installation directory. You do not need to use this option unless you use it in general or there is an error that the `agens` executable cannot be found.

In `init` mode, use this option to specify the location of the `initdb` executable.

-s
–silent
Only error messages can be printed.

-t
–timeout
The maximum time (seconds) to wait before startup or shutdown. The default value is the value of the PGCTLTIMEOUT environment variable, or 60 seconds (if not set).

-V
–version
Prints the `ag_ctl` version and exits.

–revision
Shows the agens revision information and exits.

-w
Waits until startup or shutdown completes. Standby mode is the default option for shutdown, but not for a startup. When waiting for the startup, `ag_ctl` repeatedly tries to connect to the server. When waiting for shutdown, it waits for the server to remove the PID file. This option allows you to enter an SSL password on startup. `ag_ctl` returns an exit code depending on the success of the startup or shutdown.

-W
Does not wait for startup or shutdown to complete. This is the default for start and restart modes.

-?
–help
Displays help for `ag_ctl` command-line arguments and exits.

Options for Windows

-N ***servicename***
The name of the system service to register. The name is used as a service name and a displayed name.

-P ***password***
Password that enables service startup by the user.

-S ***start-type***
Startup type of system service to be registered. The startup type can be `auto`, `demand`, or the first character of either. If it is omitted, an auto will be used.

-U ***username***
The name of the user to start the service. For domain users, use the `DOMAIN\username` format.

Environment

AGDATA Default AGDATA data directory location.

Examples

Starting the Server

To start the server, use:

```
$ ag_ctl start
```

To start the server, wait for the server to accept the connection.

```
$ ag_ctl -w start
```

To start the server using port 5432 and run without fsync, use:

```
$ ag_ctl -o "-F -p 5432" start
```

Stopping the Server

To stop the server, use:

```
$ ag_ctl stop
```

You can use the `-m` option to control how the server shuts down.

```
$ ag_ctl stop -m fast
```

Restarting the Server

Restarting the server is equivalent to stopping and restarting the server, except when saving/reusing the command-line options passed to the previously run instance by `ag_ctl`. To restart the server in its simplest form, use:

```
$ ag_ctl restart
```

Stops the server and waits for its restart for a server restart.

```
$ ag_ctl -w restart
```

To restart using port 5432, disable fsync on restart.

```
$ ag_ctl -o "-F -p 5432" restart
```

Showing the Server Status

The following is the sample status output of `ag_ctl`.

```
$ ag_ctl status
ag_ctl: server is running (PID: 2823)
/path/to/AgensGraph/bin/postgres
```

This is the command-line to be called in restart mode.

pg_upgrade

`pg_upgrade` allows you to upgrade the data stored in data files to the latest AgensGraph major version without any data dump/reload required for major version upgrades. `pg_upgrade` performs a quick upgrade by creating a new system table and reusing the old user data file.

Here is how to use:

```
$ pg_upgrade [OPTION]...
```

Options

pg_upgrade accepts the following command-line arguments:

-b ***bindir***
–old-bindir=***bindir***
Previous AgensGraph bin directory (environment variable: PGBINOLD)

-B ***bindir***
–new-bindir=***bindir***
New AgensGraph bin directory (environment variable: PGBINNEW)

-c
–check
Checks the cluster without changing the data.

-d ***datadir***
–old-datadir=***datadir***
Previous cluster data directory (environment variable: PGDATAOLD)

-D ***datadir***
–new-datadir=***datadir***
New cluster data directory (environment variable: PGDATANEW)

-j
–jobs
The number of processes or threads that can be used concurrently

-k
–link
Uses hard links instead of copying files to the new cluster.

-o ***options***
–old-options ***options***
Multiple option calls are added as an option passed directly to the previous agens command.

-O ***options***
–new-options ***options***
Multiple option calls are added as an option passed directly to the new agens command.

-p ***port***
–old-port=***port***
Old cluster port number (environment variable: PGPORTOLD)

-P ***port***
–new-port=***port***
New cluster port number (environment variable: PGPORTNEW)

-r
–retain
Maintains SQL and log files even after successful completion.

-U ***username***
–username=***username***
Installation user name for the cluster (environment variable: PGUSER)

-v
–verbose
Enables verbose internal logging

-V
–version
Displays version information and exits.

-?
–help
Displays help and exits.

Usage

Here are the steps to perform an upgrade using `pg_upgrade`:

1. Back up your existing database cluster.

```
$ mv /user/local/AgensGraph /usr/local/AgensGraph.old
```

2. Install a new AgensGraph binary.

3. Initialize the new AgensGraph cluster.

```
$ initdb
```

4. Terminate the services of the existing cluster and the new cluster.

```
$ ag_ctl -D /opt/AgensGraph/2.12/db_cluster stop
$ ag_ctl -D /opt/AgensGraph/2.13/db_cluster stop
```

5. Run pg_upgrade.

```
$ pg_upgrade -d oldCluster/data -D newCluster/data -b oldCluster/bin -B newCl
uster/bin
```

or

```
$ export PGDATAOLD=oldCluster/data
$ export PGDATANEW=newCluster/data
$ export PGBINOLD=oldCluster/bin
$ export PGBINNEW=newCluster/bin
$ pg_upgrade
```

6. Start a new AgensGraph.

```
$ ag_ctl start
```

7. After upgrading, perform the following tasks.

   - Statistics
     As optimizer statistics are not sent by `pg_upgrade`, you should run the command at the end of the upgrade to regenerate that information. You may need to set connection parameters to match the new cluster.

   - Deleting old clusters If you are satisfied with the upgrade, you can delete the old cluster's data directory by running the script mentioned upon completion of `pg_upgrade`.

Appendix

System Catalogs

This section describes the System Catalogs provided by AgensGraph. System Catalogs store schema metadata. The list of System Catalogs provided by AgensGraph is shown below; refer to the link for the catalogs of PostgreSQL.

Catalog Name	Purpose
ag_graph	A list of graphs created in the DB
ag_label	A list of labels created in the DB
ag_graphmeta	Meta information between labels

ag_graph

Shows a list of graphs created in the DB. One graph corresponds with one schema.

- ag_graph columns

Name	Type	Description
oid	oid	Graph oid(=oid in ag_label)
graphname	name	Graph name
nspid	oid	Oid of the associated schema (= namespace)

ag_label

Shows a list of labels created in the DB. One label corresponds with one table (relation) (one-to-one correspondence).

- ag_label columns

Name	Type	Description
labname	name	Label name
nspid	oid	oid of the graph belonging
labid	int32	Sequential id to distinguish labels in the same graph id of vertex table(integer part) matched id of edge table(integer part) matched
relid	relid	oid of linked table (relation)
labkind	char	This item has a value of either V or E. V: vertex label E: edge label

ag_graphmeta(Optional)

Stores meta information between labels and is used for identifying the graph model.

- ag_graphmeta columns

Name	Type	Description
graph	oid	Graph oid
edge	int16	labid of edge
start	int16	labid of start vertex
end	int16	labid of end vertex
edgecount	int16	The number of the same edges between start/end labels

System Views

Describes the System Views provided by AgensGraph. In addition to the System Catalogs, there are other views that provide intuitive information to users. The list of System Views provided by AgensGraph is shown below; refer to the link for the views of PostgreSQL.

Catalog Name	Purpose
ag_property_indexes	List of property indexes
ag_graphmeta_view	Provides graph_meta information as object name

ag_property_indexes

Displays a list of property indexes.

- ag_property_indexes columns

Name	Type	Description
graphname	name	Graph name
labelname	name	Label name
indexname	name	Property index name
tablespace	name	Tablespace name
unique	boolean	Whether duplicates are allowed or not
indexdef	text	The query statement used to create the index
owner	name	index owners
size	text	index size
description	text	index description (comment)

ag_graphmeta_view(Optional)

ag_graphmeta is a view that provides intuitive information.

- ag_graphmeta_view columns

Name	Type	Description
graphname	name	Graph name
start	name	start vlabel name
edge	name	edge name
end	name	end vlabel name
edgecount	bigint	The number of the same edges between start/end labels