AgensGraph Operations Manual

Copyright Notice

Restricted Rights Legend

Trademarks

Open Source Software Notice

기술문서 정보
제목 : AgensGraph Operations Manual
발행일 : 2018년 3월 22일
소프트웨어 버전 : AgensGraph v1.3
기술문서 버젼 : v1.0

차 례

기술문서에 대하여

기술문서의 대상

본 기술문서는 AgensGraph®(이하 AgensGraph)를 사용하여 데이터베이스를 생성하고 원할한 AgensGraph의 동작을 보장하려는 데이터베이스 관리자(Database Administrator, 이하 DBA)를 대상으로 기술한다.

기술문서의 전제 조건

본 기술문서를 원할히 이해하기 위해서는 다음과 같은 사항을 미리 알고 있어야 한다.

- 그래프 데이터베이스의 이해
- 관계형 데이터베이스의 이해
- 운영체제 및 시스템 환경의 이해
- UNIX 계열(Linux 포함)의 기본지식

기술문서의 제한 조건

본 안내서는 AgensGraph를 실무에 적용하거나 운용하는 데 필요한 모든 사항을 포함하고 있지 않으므로 설치, 환경설정 등 운용 및 관리에 대해서는 각 기술서를 참고한다.

기술문서 규약

표기	**의미**
<AaBbCc123>	프로그램 소스 코드의 파일명, 디렉터리
[**Button**]	GUI의 버튼 또는 메뉴 이름
진하게	강조
“ ”(따옴표)	다른 관련 안내서 또는 안내서 내의 다른 장 및 절 언급
‘입력항목’	화면 UI에서 입력 항목에 대한 설명
하이퍼링크	메일계정, 웹 사이트
>	메뉴의 진행 순서
+—	하위 디렉터리 또는 파일 있음
|—	하위 디렉터리 또는 파일 없음
참고	참고 또는 주의사항
[**Figure 1.1**]	그림 이름
[**Table 1.1**]	표 이름
AaBbCc123	명령어, 명령어 수행 후 화면에 출력된 결과물, 예제코드
{}	필수 인수 값
[]	옵션 인수 값
|	선택 인수 값

연락처

Korea

Bitnine Inc.

A1201 GangSeo Hangang Xi Tower 401,

Yangcheon-ro, Gangseo-gu, Seoul,

South Korea

Tel : +82-70-4800-3517

Fax : +82-70-8677-2552

Email : agens@bitnine.net

Web : bitnine.net

USA

Bitnine Global Inc.

3945 Freedom Cir., Suite 260,

Santa Clara, CA 95054

U.S.A

Tel : +1 (408) 352-5165

Email : agens@bitnine.net

Web : bitnine.net

1 Introduction

1.1 What is AgensGraph?

AgensGraph는 복잡한 현대 데이터 환경에 최적화된 차세대 다중 모델 그래프 데이터베이스이다. AgensGraph는 PostgreSQL RDBMS 기반 다중 모델 데이터베이스로서 관계형 데이터 모델과 그래프 데이터 모델을 동시에 지원한다. 따라서 기존의 관계 데이터 모델 및 유연성 있는 그래프데이터 모델을 하나의 데이터베이스에서 통합 사용이 가능하다. AgensGraph는 Ansi-SQL과 Open Cypher (http://www.opencypher.org) 를 모두 지원한다. 즉, 하나의 쿼리내에서 SQL쿼리와 Cypher 쿼리를 함께 사용할 수 있다.

AgensGraph는 안정성과 신뢰성을 갖춘 기업용 데이터베이스이다. AgensGraph는 고도로 복잡하게 연결된 그래프 데이터를 다루는 데 최적화되어 있으며 ACID 트랜잭션 특성, 다중 버전 동시성 제어, 내장 함수, 트리거, 제약 조건, 고도화된 모니터링 기능, 유연한 데이터 모델 (JSON) 등을 포함해 엔터프라이즈 데이터베이스 환경에 필수적인 다양하고 강력한 데이터베이스 기능을 제공한다. 그뿐만 아니라, AgensGraph는 다양한 기능을 제공하는 PostgreSQL 에코시스템을 토대로 PostGIS와 같은 뛰어난 외부 모듈 확장이 용이하다.

2 Deployment

2.1 System Requirements

AgensGraph가 동작하기 위해서는 4GB 이상의 RAM과 2.5GB 이상의 disk 공간이 필요하다. 일반적인 Unix-호환 플랫폼에서 동작 가능하지만, 정식으로 지원되는 플랫폼은 Linux 계열 (Centos, Ubuntu, RHEL) 과 Windows 계열 (Windows Server 2008 64bits, Windows Server 2012 64bits, Windows 7 64bits) 이다.

2.2 Pre-install tasks

2.2.1 사용자 계정

외부에서 액세스 가능한 서버 데몬과 마찬가지로, AgensGraph도 별도의 사용자 계정으로 실행하는 것이 좋다. Unix 계열이라면 useradd 또는 adduser 명령을 통해 사용자 계정을 추가할 수 있다.

```
$ useradd agens
```

2.2.2 커널 리소스 관리

공유메모리 및 세마포어

공유 메모리 및 세마포어는 통칭 ``System V IPC'' 라고 한다. AgensGraph를 구동하기 위해선, 'System V IPC' 의 공유메모리와 세마포어가 요구된다. AgensGraph가 요구한 크기가 IPC 제한을 초과한 경우 서버는 시작에 실패한다.

이름	설명	적절한 값
SHMMAX	공유 메모리 세그먼트의 최대 크기 (바이트)	최소 1kB (서버 사본이 다수 실행되는 경우 그 이상)
SHMMIN	공유 메모리 세그먼트의 최소 크기 (바이트)	1
SHMALL	사용 가능한 공유 메모리의 총 양 (바이트 또는 페이지)	바이트인 경우 SHMMAX와 동일. 페이지인 경우 ceil (SHMMAX/PAGE_SIZE)
SHMSEG	프로세스당 공유 메모리 세그먼트의 최대수	1개 세그먼트만 필요하지만 기본값이 훨씬 큼
SHMMNI	시스템 차원 (system-wide) 의 공유 메모리 세그먼트의 최대 수	SHMSEG와 동량 외 다른 애플리케이션의 여유분
SEMMNI	세마포어 식별자의 최대 수 (예 : 세트)	최소한 ceil((max_connections + autovacuum_max_workers + 4) / 16)
SEMMNS	시스템 차원 (system-wide) 의 세마포어	최대 수 ceil((max_connections + autovacuum_max_worker + 4) / 16) * 17 외 다른 애플리케이션의 여유분
SEMMSL	세트별 세마포어 최대 수	최소한 17
SEMMAP	세마포어 맵에서 항목 수	텍스트 참조
SEMVMX	세마포어 최대 값	최소한 1000(기본값은 대체로 32767; 필요한 경우 외에는 변경하지 말 것)

AgensGraph은 서버 사본별로 System V 공유 메모리 수 바이트가 필요하다 (64비트 플랫폼의 경우 보통 48바이트). 최신 운영 체제에서 이 정도 양은 손쉽게 할당 가능하다. 그러나, 여러 서버 사본을 실행하거나 다른 애플리케이션도 System V 공유 메모리를 사용할 때는 바이트 단위의 공유 메모리 최대 크기인 SHMMAX를 늘리거나 시스템 차원 (system-wide) 의 System V 공유 메모리인 SHMALL를 늘려야 할 수도 있다. SHMALL는 대부분의 시스템에서 바이트 단위가 아닌 페이지 단위로 처리된다는 점에 유의하라.

AgensGraph의 경우, 공유 메모리 세그먼트의 최소 크기 (SHMMIN) 는 많아야 약 32바이트에 불과하기 때문에 (대개 1) 문제의 원인이 될 가능성은 낮다. 시스템 차원 (system-wide) 의 세그먼트 최대 수 (SHMMNI) 또는 프로세

스당 최대 수(SHMSEG)는 시스템이 영(0)으로 설정해 놓지 않는 한 문제의 원인이 될 가능성은 낮다.
AgensGraph은 16개의 세트 중에서 연결당 1개의 세마포어 (max_connections)와 autovacuum worker 프로세스당 (autovacuum_max_workers) 1개의 세마포어를 사용한다. 각 세트는 타 애플리케이션의 세마포어 세트와의 충돌을 감지하는 ``매직 넘버''가 포함된 17번째 세마포어를 갖고 있다. 시스템에서 세마포어 최대 수는 SEMMNS에 의해 설정되며, 최소한 max_connections + autovacuum_max_workers + 각각 허용된 16개 연결에 1 추가 + worker 이어야 한다(표 1-1 공식 참조). 매개변수 SEMMNI는 시스템 상 동시에 존재할 수 있는 세마포어 세트의 개수를 제한한다. 최소한 ceil((max_connections + autovacuum_max_workers + 4) / 16) 이어야 한다. 수용하는 연결 수를 줄이면 실패 시 임시 방편으로 해결할 수 있지만, semget 함수로부터 ``No space left on device''라는 애매한 메시지도 받게 된다.
경우에 따라서는 SEMMAP를 적어도 SEMMNS와 유사하게 늘릴 필요가 있을 수 있다. 이 매개변수는 세마포어 자원 맵의 크기를 정하며, 이 맵에는 세마포어의 서로 인접한 블록들이 각기 필요로 하는 엔트리가 들어 있다. 해제된 세마포어 세트는 해제된 블록에 인접하고 있는 엔트리에 추가되거나 새로운 엔트리에 등록된다. 맵이 꽉 차면 해제된 세마포어는 사라진다(재부팅될 때까지). 세마포어 공간이 쪼개질수록 가용한 세마포어가 점점 적어진다.
세트에 포함될 수 있는 세마포어 수를 결정하는 SEMMSL은 AgensGraph의 경우 최소 17이어야 한다.
SEMMNU 및 SEMUME 같은 ``semaphore undo''와 관련된 기타 설정은 AgensGraph에 영향을 미치지 않는다.

> note : 일부 쿼리 수행시 `경고 : 공유메모리 부족' 오류 발생가 발생하는 경우 Bitnine 홈페이지에서 기술 지원 요청을 통해 사이트 환경에 맞는 Memory 설정을 가이드 받도록 한다.

2.3 Single instance install

AgensGraph를 설치하는 방법은 두 가지가 있다. 공식 홈페이지에서 바이너리를 다운로드 받아 설치하는 방법과 소스 코드를 직접 다운받아 빌드해 설치하는 방법이 있다. 어느 방법이든, AgensGraph 관리를 위한 계정에서 작업할 것을 권장한다.

2.3.1 바이너리 다운을 통한 설치

AgensGraph 공식 홈페이지의 S/W Download에서 OS에 맞는 바이너리 또는 인스톨러를 다운로드 받는다

Linux

리눅스 바이너리는 tarball로 압축되어 있다. 이를 원하는 위치에 압축을 푼 뒤, 아래 있는 설치 후 작업을 수행하면 설치가 완료된다.

Windows

게시 예정

2.3.2 소스코드 빌드를 통한 설치

1. AgensGraph의 github에 접속하여 소스코드를 clone 받는다.

   $ git clone https://github.com/bitnine-oss/agensgraph.git

2. clone받은 위치로 이동하여 소스 트리에 configure를 수행한다. 이때 --prefix=/path/to/install를 통해 AgensGraph가 설치 될 위치를 설정할 수 있다.

```
$ ./configure
```

3. 빌드를 수행한다.

```
$ make install
```

4. 확장 모듈 및 바이너리를 설치한다.

```
$ make install-world
```

2.3.3 설치 후 작업 (Linux)

Unix계열과 Linux의 경우, 환경변수를 등록하지 않으면 설치된 라이브러리를 제대로 읽지 못해 작동하지 않고, 호출 시 절대경로를 입력해야하는 불편함이 있다. AgensGraph가 설치된 user의 bash_profile에 환경변수를 아래와 같이 추가하여 위와같은 문제를 방지할 수 있다.

export AGHOME=/path/to/AgensGraph

export AGDATA=/path/to/AgensGraph /data

export PATH=$AGHOME/bin:$PATH

export LD_LIBRARY_PATH=$AGHOME/lib:$LD_LIBRARY_PATH

Environment	Description
AGHOME	AgensGraph가 설치되는 디렉터리이다.
AGDATA	데이터 디렉토리 위치이다.
PATH	AgensGraph를 사용하기 위한 디렉터리 경로로 $AGHOME/bin을 설정한다.
LD_LIBRARY_PATH	AgensGraph 사용시 필요한 공유 라이브러리가 위치한 경로이다. $AGHOME/lib을 설정한다.

2.4 Operations

2.4.1 initDB

AgensGraph를 구동하기 위해서는 *initdb*를 통해 데이터베이스 저장소 영역을 초기화해야 한다. 이를 데이터베이스 클러스터라고 하며, 실행중인 데이터베이스 서버의 단일 인스턴스가 관리하는 데이터베이스 컬렉션이다. 데이터베이스 클러스터는 모든 데이터가 저장되는 단일 디렉토리로, 파일 시스템 관점에서는 데이터 디렉토리 또는 데이터 영역이라고 한다. 이 데이터를 어디에 저장할 것인지는 -D 옵션을 통해 나타낼 수 있다.

```
$ initdb -D /home/agensgraph/data
```

또는 ag_ctl을 이용해 데이터베이스 저장소 영역을 초기화할 수 있다.

```
$ ag_ctl initdb -D /home/agensgraph/data
```

initdb는 지정한 디렉토리가 존재하지 않는 경우 디렉토리를 생성하고, 이미 초기화돼 있는 디렉토리라면 실행을 거부한다.

2.4.2 role 생성

role은 데이터베이스가 설정된 방법에 따라 데이터베이스 사용자 또는 데이터베이스 사용자 그룹으로 생각할 수 있다. role은 데이터베이스 개체를 소유할 수 있으며 해당 개체에 대한 권한을 다른 role에 할당하여 해당 개체에 액세스할 수 있는 사용자를 제어할 수 있다. 또한, role의 멤버십을 다른 role에 부여할 수 있으므로 멤버 role이 다른 role에 할당된 권한을 사용하도록 할 수 있다.
role을 생성/삭제하려면 아래와 같은 SQL 명령을 사용해야 한다.

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

편의상, *createuser* 및 *dropuser*을 쉘 명령줄에서 호출할 수 있다.(SQL 명령의 래퍼(wrapper)로 동작한다.)

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

2.4.3 createdb

데이터베이스를 생성/제거하려면 AgensGraph 서버를 시작한 다음 SQL 명령 *CREATE DATABASE*, *DROP DATABASE*을 통해 생성한다.

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

*CREATE DATABASE*를 수행한 현재 role은 자동으로 새 데이터베이스의 소유자가 된다.
편의상, *createdb*와 *dropdb*를 쉘 명령줄에서 호출하여 데이터베이스를 생성/제거할 수 있다. 생성과 동시에 새 데이터베이스의 소유자를 지정하기 위해서 *-O* 옵션을 사용할 수 있다.

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

2.4.4 Startup

데이터베이스에 액세스하기 전 데이터베이스 서버를 시작해야 한다. 데이터베이스 서버를 시작하기 위해서 *ag_ctl*을 사용하며, 시작하기 위해서는 초기화된 데이터베이스 저장소가 필요하며, -D 옵션으로 해당 디렉토리를 지정해줘야 한다. 만약, 환경변수 AGDATA를 통해 데이터베이스 저장소 위치를 설정했다면 -D 옵션이 없이도 서버를 시작할 수 있다. -l 옵션을 통해 log를 남길 logfile을 설정할 수 있다.

```
$ ag_ctl start [-D /home/agensgraph/data] [-l /home/agensgraph/data/logfile]
```

2.4.5 Shutdown

AgensGraph를 종료하는 방법에는 몇 가지 방법이 있는데, 마스터 프로세스에 어떤 신호를 전송하느냐에 따라 종료 유형을 제어할 수 있다.

신호	설명
SIGTERM	스마트 셧다운 모드로, 서버는 새로운 연결은 허용하지 않지만, 기존 세션이 정상 종료될 때까지 기다린 후 종료된다.
SIGINT	빠른 셧다운 모드로, 서버는 새로운 연결을 허용하지 않고, 기존의 모든 프로세스의 트랜잭션을 중단하고 종료한다.
SIGQUIT	즉시 셧당다운 모드로, 서버는 모든 자식 프로세스를 종료시킨다. 5초 이내에 종료되지 않는 자식 프로세스는 SIGKILL을 전송해 강제 종료시키는데, 이로인해 다음 시작시 복구로 이어지므로 비상시에만 권장된다.

아래와 같이 *ag_ctl*에 -m 옵션을 이용해 종료할 수 있다(-m을 지정하지 않을 경우, 기본값은 smart 이다).

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

3 Architecture

AgensGraph는 PostgreSQL을 기반으로 개발된, Multi-model 데이터베이스이다. PostgreSQL이 제공하는 모든 기능뿐만 아니라, 그래프 질의어인 Cypher를 사용해 그래프 질의도 처리할 수 있도록 설계되어 있다. 내부적으로는 크게, PostgreSQL 처리기와 Graph 처리기로 나눌 수 있으며, 이들은 캐시와 트랜잭션 영역을 공통으로 사용한다. 사용자는 한 query에 SQL과 Cypher 질의를 동시에 수행해 성능상 이점이나 구현의 편리를 취할 수 있다.

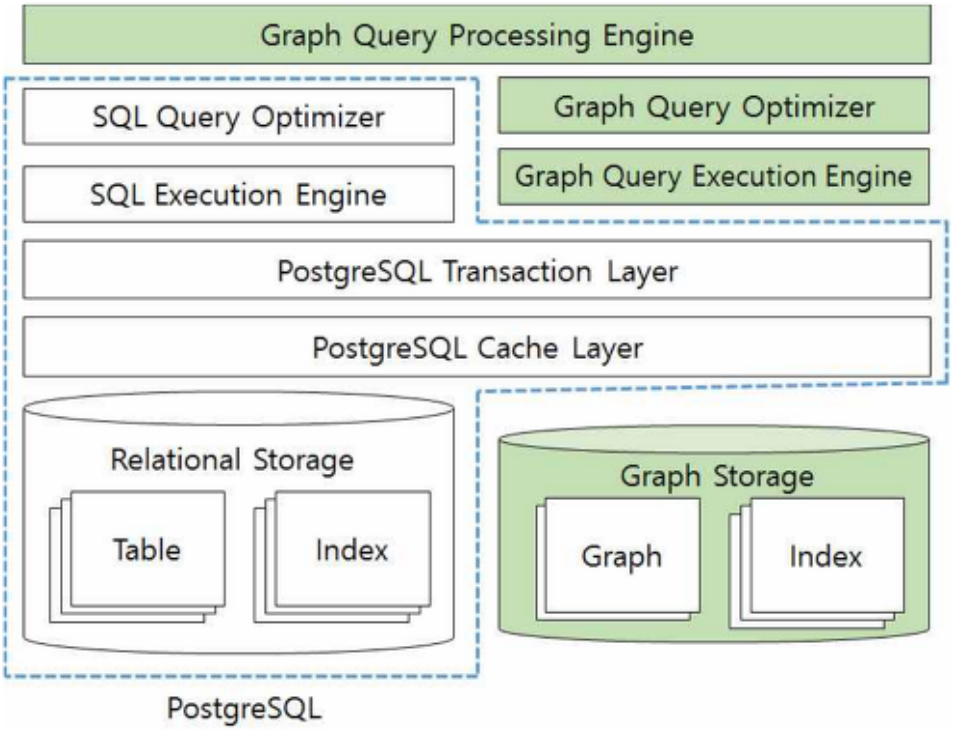

그림 1.1: AgensGraph Architecture

3.1 Process structure

AgensGraph는 PostgreSQL과 동일한 클라이언트/서버 모델을 사용하고 있다. 하나의 세션(작업)은 다음과 같은 프로세스들(프로그램들)의 상호 작동으로 구성된다.

- 서버 프로세스
  - 이것은 데이터베이스 파일을 관리하고, 클라이언트 응용 프로그램들이 서버에 연결을 요청할 때, 그 요청들을 처리(수락하거나 거부하는 일)하고, 클라이언트들이 데이터베이스를 사용할 수 있도록 기반 작업들을 준비한다. 이 프로세스의 이름은 postgres이다.
- 클라이언트 프로세스

- 이것은 데이터베이스를 사용하려는 사용자 측 응용 프로그램을 말한다. 클라이언트 응용 프로그램은 자연적으로 매우 다양한 형태를 띄고 있다. 어떤 것은 텍스트 기반의 프로그램이기도 하고, 어떤 것은 그래픽 응용 프로그램이기도 하고, 어떤 것은 웹서버를 통해서 웹페이지로 보여지기도 한다. 몇몇의 클라이언트 프로그램들은 이미 패키지 안에 포함되어서 배포되기도 한다. 이것들은 대부분 사용자들이 직접 개발한 것들이다.

클라이언트/서버 환경의 프로그램들은 대부분 그렇듯이, AgensGraph에서도 클라이언트와 서버가 서로 다른 호스트일 수 있다. 이런 경우에는 서로 간의 통신이 TCP/IP 네트워크 기반 아래서 이뤄진다. 이 부분은 아주 중요한 부분을 시사하고 있다. 클라이언트와 서버가 서로 다른 경우에는 클라이언트에서 접근하고자 하는 데이터베이스 파일에 대해서 직접적으로 접근할 수 없음을 의미한다. 즉, 클라이언트에서 접근 할 수 있는 파일은 그 클라이언트가 실행되고 있는 호스트의 파일이지 서버가 가동 중인 호스트의 파일이 아님을 알고 있어야한다. AgensGraph 서버는 사용자의 동시 접속을 위해서 각 접속에 대한 새로운 프로세스를 생성한다. 이 방법은 클라이언트와 새로 만들어진 서버 프로세스가 통신할 때, 마스터 프로세스인 postgres 프로세스의 간섭 없이 이루어짐을 의미한다. 종합하면, postgres 프로세스는 서버 호스트에서 항상 실행되고 있으면서, 클라이언트의 접속 요청을 처리해서 새로운 하위 서버 프로세스를 만드는 역할을 한다.

4 Security

이 장에서는 보안과 관련된 내용을 다룬다. 보안에 있어서 클라이언트 인증과 데이터베이스 role은 중요한 역할을 한다.

여기서 소개하고 있는 내용은 AgensGraph가 PostgreSQL 기반으로 개발되었기 때문에, GDB side가 아닌 RDB side의 기능소개는 PostgreSQL의 기술문서를 인용하였다.

4.1 Client Authentication

클라이언트 애플리케이션이 데이터베이스 서버에 연결하는 경우, 연결 가능한 데이터베이스 사용자를 제한하는 것이 중요하다.

인증은 데이터베이스 서버가 클라이언트 ID를 구축하는 프로세스이며, 더 나아가 요청된 데이터베이스 사용자 이름으로 클라이언트 애플리케이션(또는 클라이언트 애플리케이션을 실행하는 사용자)의 연결을 허용할 것인지 결정하는 프로세스이다.

AgensGraph는 서로 다른 여러 가지 클라이언트 인증 방법을 제공한다. 특정 클라이언트 연결을 인증하는 데 사용되는 방법은 (클라이언트) 호스트 주소 및 데이터베이스, 사용자를 기준으로 선택할 수 있다.

AgensGraph 데이터베이스 사용자 이름은 서버가 실행되는 운영 체제의 사용자 이름과 논리적으로 별개이다. 특정 서버의 모든 사용자도 서버 머신에 계정을 가질 수 있지만 운영 체제 사용자 이름과 일치하는 데이터베이스 사용자 이름을 할당하는 것이 합당하다. 그러나, 원격 연결을 수용하는 서버에는 로컬 운영 체제 계정이 없는 데이터베이스 사용자가 다수일 수 있으며, 이런 경우 데이터베이스 사용자 이름과 OS 사용자 이름을 연결 짓는 것은 불필요하다.

4.1.1 pg_hba.conf 파일

클라이언트 인증은 전통적으로 데이터베이스 클러스터의 데이터 디렉토리에 저장되는 환경 설정 파일인 `pg_hba.conf`로 제어된다.

`pg_hba.conf` 파일의 형식은 한 줄이 하나의 레코드로 이루어진 레코드들의 집합이다. 빈 줄은 무시된다. # 주석 문자 뒤의 텍스트도 무시된다. 레코드는 줄을 바꿔서 이어질 수 없다. 레코드는 여러 개의 필드로 구성되며, 구분자는 공백 및 / 또는 탭으로 구분된다. 필드 값에 큰 따옴표를 사용하면 필드에 공백을 포함할 수 있다. 데이터베이스 또는 사용자, 주소 필드의 키워드에 따옴표를 사용하면 단어는 자체의 특수한 의미를 상실한다.

각 레코드는 이러한 매개 변수와 일치하는 연결에 사용되는 연결 유형 및 클라이언트 IP 주소 범위(연결 유형에 해당하는 경우), 데이터베이스 이름, 사용자 이름, 인증 방법을 지정한다. 연결 타입 및 클라이언트 주소, 요청된 데이터베이스, 사용자 이름이 일치하는 첫 번째 레코드는 인증을 수행할 때 사용된다. ``제어 이동(fall-through)'' 또는 ``백업''은 없다. 레코드 하나가 선택되고 인증이 실패한 경우 다음 레코드는 인증되지 않는다. 일치하는 레코드가 없으면 액세스가 거부된다.

레코드는 다음 7가지 형식 중 하나이다.

```
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]
host      database user IP-address IP-mask auth-method [auth-options]
hostssl   database user IP-address IP-mask auth-method [auth-options]
hostnossl database user IP-address IP-mask auth-method [auth-options]
```

필드의 의미는 다음과 같다.

- **local**
  이 레코드는 Unix 도메인 소켓을 사용한 연결에 해당한다. 유형의 레코드가 없으면 Unix 도메인 소켓 연결은 불가능하다.

- **host**
  이 레코드는 TCP/IP를 사용한 연결에 해당한다. host 레코드는 SSL 연결 시도 혹은 비 SSL 연결 시도와 일치한다.

- **hostssl**
  이 레코드는 TCP/IP를 사용한 연결 시도와 일치하지만, SSL 암호화를 사용한 연결에만 해당된다. 이 옵션을 사용하려면 서버는 SSL 지원이 내장되어 있어야 한다. 또한 SSL은 ssl 환경 설정 매개 변수를 설정함으로써 서버 시작 시에 활성화되어야 한다.

- **hostnossl**
  이 레코드 유형은 hostssl과는 반대로 동작한다. SSL을 사용하지 않는 TCP/IP 상의 연결 시도에 대해서만 일치한다.

- **database**
  이 레코드는 데이터베이스 이름을 지정한다.

  - all 값은 모든 데이터베이스와 일치하도록 지정한다.
  - sameuser 값은 요청된 데이터베이스가 요청된 사용자와 이름이 동일한 경우에 레코드가 일치하도록 지정한다.
  - samerole 값은 요청된 사용자가 요청된 데이터베이스와 이름이 동일한 role의 멤버여야 하는지 지정한다. superuser는 직접 혹은 간접적으로 role의 명시적인 멤버가 아닐 경우, 단지 superuser라는 이유로 samerole에 대한 role의 멤버로 간주되지 않는다.
  - replication 값은 복제 연결이 요청되는 경우 레코드가 일치하도록 지정한다(복제 연결은 특정 데이터베이스를 지정하지는 않는다). 이 경우가 아니라면 특정 AgensGraph 데이터베이스의 이름으로

사용된다. 쉼표로 구분해서 데이터베이스 이름을 여러 개 쓸 수 있다. 데이터베이스 이름이 포함된 파일은 파일 이름 앞에 @를 붙여서 지정 가능하다.

- **user**

  이 레코드와 일치하는 데이터베이스 사용자 이름을 지정한다. all 값은 모든 사용자와 일치하도록 지정한다. 이 외에는, 특정한 데이터베이스 사용자의 이름이거나 앞에 +를 붙인 그룹 이름이다. (AgensGraph에서는 사용자와 그룹 이름 간에 실제로 차이는 없다. + 마크는 실제로 ``이 role의 직접 또는 간접 멤버인 아무 role과 일치함''을 의미하며, + 마크가 없는 이름은 유일하게 특정 role과 일치한다.) 이러한 이유로, superuser는 단지 superuser라는 이유 때문이 아니라, 직접 혹은 간접적으로 role의 명시적 멤버인 경우에만 role 멤버로 간주된다. 쉼표로 구분해서 사용자 이름을 여러 개 쓸 수 있다. 사용자 이름이 포함된 파일은 파일 이름 앞에 @를 붙여서 지정 가능하다.

- **address**

  이 레코드와 일치하는 클라이언트 머신 주소를 지정한다. 이 필드는 호스트 이름, IP 주소 범위 또는 아래 설명된 특수 키워드 중 하나를 포함할 수 있다.

  IP 주소는 CIDR 마스크 길이의, 점으로 구분된 십진수 (dotted decimal) 표준 표기법으로 지정된다. 마스크 길이는 일치해야 하는 클라이언트 IP 주소의 상위 비트 수를 나타낸다. 이것의 오른쪽에 있는 비트는 주어진 IP 주소에서 0이어야 한다. IP 주소 및 /, CIDR 마스크 길이 사이에 공백이 있으면 안 된다.

  이러한 방법으로 지정된 IP 주소 범위의 전형적인 예시는 단일 호스트의 경우 172.20.143.89/32, 소규모 네트워크의 경우 172.20.143.0/24, 대규모 네트워크의 경우 10.6.0.0/16일 수 있다. 0.0.0.0/0은 모든 IPv4 주소를 나타내며 ::/0은 모든 IPv6 주소를 나타낸다. 단일 호스트를 지정하려면 IPv4의 경우 CIDR 마스크 32를 사용하고 IPv6의 경우 128을 사용해야 한다. 네트워크 주소 끝의 0을 빠트리면 안 된다.

  사용자는 아무 IP 주소나 일치하도록 all을 쓸 수도 있고, 서버의 자체 IP 주소만을 일치하도록 samehost를 쓸 수도 있고, 서버가 직접 연결되는 서브넷의 모든 주소와 일치하도록 samenet을 쓸 수도 있다.

- **IP-address, IP-mask**

  이 필드는 CIDR-address 표기의 대안으로 사용될 수 있다. 마스크 길이를 지정하는 대신 실제 마스크가 쉼표로 구분하여 지정된다. 예를 들면, 255.0.0.0은 IPv4 CIDR 마스크 길이 8을 나타내고, 255.255.255.255는 CIDR 마스크 길이 32를 나타낸다.

  이 필드는 host 및 hostssl, hostnossl 레코드에 적용된다.

- **auth-method**

  연결이 이 레코드와 일치할 때 사용하는 인증 방법을 지정한다. 가능한 선택안은 다음과 같다.

  - **trust**

    무조건 연결을 허용한다. 이 방법은 패스워드나 다른 인증 없이 임의의 AgensGraph 데이터베이스 사용자로 로그인하여 누구나 AgensGraph 데이터베이스 서버에 연결할 수 있다.

- **reject**

  무조건 연결을 거부한다. 이것은 그룹에서 특정 호스트를 ``필터링''할 때 유용하다. 예를 들면, reject 줄은 특정 호스트의 연결을 차단하고, 그 이후의 줄은 특정 네트워크의 남은 호스트들과의 연결을 허용한다.

- **md5**

  클라이언트가 인증을 위해 double-MD5-hashed 패스워드를 제공해야 한다.

- **password**

  클라이언트가 인증을 위해 암호화되지 않은 패스워드를 제공해야 한다. 패스워드는 네트워크 상에서 일반 텍스트로 전송되므로 신뢰하지 않는 네트워크에서 이것을 사용하면 안 된다.

- **gss**

  GSSAPI를 사용하여 사용자를 인증한다. 이것은 TCP/IP 연결에서만 사용할 수 있다.

- **sspi**

  SSPI를 사용하여 사용자를 인증한다. 이것은 Windows에서만 사용할 수 있다.

- **ident**

  클라이언트의 ident 서버에 접촉함으로써 클라이언트의 운영 체제 사용자 이름을 획득하고, 요청된 데이터베이스 사용자 이름과 일치하는지 확인한다. Ident 인증은 TCP/IP 연결에서만 사용할 수 있다. 로컬 연결에 대해 지정하는 경우 피어 (peer) 인증이 대신 사용된다.

- **peer**

  클라이언트의 운영 체제 사용자 이름을 운영 체제에서 획득하고, 요청된 데이터베이스 사용자 이름과 일치하는지 확인한다. 이것은 로컬 연결에서만 사용할 수 있다.

- **ldap**

  LDAP 서버를 사용하여 인증한다.

- **radius**

  RADIUS 서버를 사용하여 인증한다.

- **cert**

  SSL 클라이언트 인증을 사용하여 인증한다.

- **pam**

  운영 체제에서 제공하는 PAM(Pluggable Authentication Modules)을 사용하여 인증한다.

- **auth-options**

  auth-method 필드 이후에 인증 방법에 대한 옵션을 지정하는 name=value 형식의 필드가 있을 수 있다.

@ 구문이 포함된 파일은, 공백 또는 쉼표로 구분된 이름 목록으로 읽는다. pg_hba.conf처럼 #으로 표시된 주석 및 중첩된 @ 구문이 허용된다. 파일 이름 뒤에 @가 나오는 것이 절대 경로가 아니면 참조 파일이 있는 디렉토리의 상대 경로로 취급된다.

pg_hba.conf 레코드는 각 연결 시도에 대해 순차적으로 검사되므로 레코드의 순서는 중요하다. 일반적으로 초기 레코드는 연결 일치 매개 변수는 치밀하고, 인증 방법은 느슨한 반면, 후기 레코드는 일치 매개 변수는 느슨하고 인증 방법은 강력하다. 예를 들면, 로컬 TCP/IP 연결에 대한 trust 인증을 사용하려고 하면서 원격 TCP/IP 연결을 할 수도 있다. 이런 경우 127.0.0.1로부터 연결을 위한 trust 인증을 지정한 레코드는 다양한 허용 클라이언트 IP 주소에 대해 패스워드 인증을 지원하는 레코드 이전에 나타난다.

4.1.2 사용자 이름 맵

Ident 또는 GSSAPI 같은 외부 인증 시스템을 사용하는 경우, 연결을 시작하는 운영 체제 사용자의 이름은 연결해야 하는 데이터베이스 사용자 이름과 다를 수 있다. 이런 경우 사용자 이름 맵을 사용하여 운영 체제 사용자 이름과 데이터베이스 사용자 이름을 맵핑할 수 있다. 사용자 이름 맵핑을 사용하려면 pg_hba.conf 옵션 필드에서 map=map-name을 지정해야 한다. 이 옵션은 외부 사용자 이름을 수신하는 모든 인증 방법에서 지원된다. 서로 다른 연결에 서로 다른 맵핑이 필요할 수 있으므로 연결별로 사용할 맵을 지정하기 위해 사용할 맵의 이름은 pg_hba.conf의 map-name 매개 변수에서 지정된다.
사용자 이름 맵은 ident 맵 파일에서 정의되며, 기본적으로 이름은 pg_ident.conf이며 데이터 디렉토리에 저장된다.

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

주석 및 공백은 pg_hba.conf에서와 동일하게 처리된다. map-name은 pg_hba.conf에서 이 맵핑을 참고하기 위해 사용되는 임의의 이름이다. 나머지 2개의 필드는 운영 체제 사용자 이름과 데이터베이스 사용자 이름을 지정한다. 동일한 map-name을 여러 번 사용해서 단일 맵 내에서 여러 사용자 맵핑을 지정할 수 있다.
주어진 한 명의 운영 체제 사용자가 몇 명의 데이터베이스 사용자에 대응하는지에 대해서는 아무런 제한이 없다(그 반대도 마찬가지). 따라서, 맵의 항목은 사용자가 동일함을 의미한다기보다 ``이 운영 체제 사용자는 이 데이터베이스 사용자로서 연결이 허용된다''로 생각되어야 한다. 사용자가 연결 요청을 한 데이터베이스 사용자 이름을 사용하여 외부 인증 시스템에서 획득한 사용자 이름과 쌍을 이루는 맵 항목이 있을 경우 연결이 허용된다.
system-username 필드가 슬래시(/)로 시작되는 경우 필드의 나머지는 정규식으로 처리된다. 정규식은 단일 캡처 또는 괄호 표현식을 포함할 수 있으며, \1 (역슬래시)로 database-username 필드에서 참조가 가능하다. 이것은 한 줄로 된 여러 사용자 이름을 맵핑할 수 있으며, 단순 구문 대체 시 특히 유용하다. 예를 들면,

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

이 항목은 @mydomain.com로 끝나는 시스템 사용자 이름을 사용하여 사용자에 대한 도메인 부분을 삭제하고, 시스템 이름이 @otherdomain.com로 끝나는 모든 사용자가 guest로 로그인하는 것을 허용한다.

4.1.3 인증 방법

이번 절에서는 인증 방법을 자세하게 다룬다.

트러스트 인증

trust 인증이 지정된 경우 AgensGraph는 지정한 데이터베이스 사용자 이름을 사용하여 서버에 연결 가능한 모든 이가 데이터베이스 액세스에 대한 인증을 받는 것으로 간주한다 (superuser 이름 포함). 물론, database 및 user 칼럼의 제한도 계속 적용된다. 이 방법은 서버 연결에 대한 적절한 운영 체제 수준의 보호가 제공되는 경우에만 사용되어야 한다.

trust 인증은 단일 사용자 워크스테이션에 대한 로컬 연결 시 적절하며, 매우 편리하다. 다중 사용자 머신에서는 일반적으로 적절하지 않다. trust 인증은 trust를 지정하는 pg_hba.conf에서 서버와 연결이 허용된 모든 머신의 모든 사용자를 신뢰하는 경우에만 TCP/IP 연결에 적합하다. localhost(127.0.0.1) 외에 TCP/IP 연결 시 trust를 사용하는 것은 별로 합당하지 않다.

패스워드 인증

패스워드 기반 인증 방법은 md5 및 password이다. 패스워드가 전송될 때 각각 MD5 해시 및 일반 텍스트로 전송되는 점을 제외하고 두 방법은 유사하게 작동된다.

패스워드 ``스니핑'' 공격에 대비하기 위해 md5를 사용하는 것이 바람직하다. 일반 password는 가능하면 피해야 한다. 대신, md5는 db_user_namespace 기능과 함께 사용할 수 없다. 연결이 SSL 암호화로 보호되는 경우 password를 안전하게 사용할 수 있다.

AgensGraph 데이터베이스 패스워드는 운영 체제 사용자 패스워드와 구분된다. 각 데이터베이스 사용자에 대한 패스워드는 pg_authid 시스템 카탈로그에 저장된다. 패스워드는 SQL 명령 CREATE USER 및 ALTER ROLE으로 관리할 수 있으며, 예를 들면 CREATE USER foo WITH PASSWORD 'secret'와 같다. 패스워드가 사용자에 대해 설정되지 않은 경우 저장된 패스워드는 null이고 패스워드 인증은 해당 사용자에 대해 항상 실패한다.

GSSAPI 인증

GSSAPI는 RFC 2743에 정의된 보안 인증을 위한 산업 표준 프로토콜이다. GSSAPI는 이것을 지원하는 시스템에 대해 자동 인증 (single sign-on)을 제공한다. 인증 자체는 안전하지만, SSL을 사용하지 않을 경우 데이터베이스 연결을 통해 전송된 데이터는 암호화되지 않은 상태로 전송된다.

데이터베이스에 연결할 때 요청된 데이터베이스 사용자 이름과 일치하는 보안 규칙에 대한 티켓이 있는지 확인해야 한다. 예를 들면, 데이터베이스 이름 fred의 경우 보안 규칙 fred@EXAMPLE.COM은 연결이 가능하다.

다음 환경 설정 옵션이 GSSAPI에 대해 지원된다.

- **include_realm**

  1로 설정되면 인증된 사용자 보안 규칙의 영역 이름이 사용자 이름 맵핑을 통해 전달되는 시스템 사용자 이름에 포함된다. 이것은 복수의 영역 (realm)에서 사용자를 처리할 때 유용하다.

- **map**

시스템과 데이터베이스 사용자 이름 사이의 맵핑을 허용한다.

- `krb_realm`

  사용자 보안 규칙 이름과 일치하는 영역(realm)을 설정한다. 이 매개 변수가 설정된 경우 해당 영역(realm)의 사용자만 허용된다. 설정되지 않으면 모든 영역(realm)의 사용자가 연결할 수 있으며, 사용자 이름 맵핑 완료 여부에 달려 있다.

SSPI 인증

SSPI는 SSO(single sign-on) 보안 인증을 위한 Windows 기술이다. AgensGraph는 `negotiate`모드에서 SSPI를 사용한다. 이것은 가능한 경우 Kerberos를 사용하고, 그 외에는 NTLM으로 자동 폴백(fall back) 된다. SSPI 인증은 서버와 클라이언트가 모두 Windows를 사용하는 경우에만 작동되고, GSSAPI를 사용할 수 있는 경우에는 비 Windows에서 작동된다.
Kerberos 인증 사용 중에는 SSPI가 GSSAPI와 동일한 방식으로 작동된다.
다음 환경 설정 옵션이 SSPI에 대해 지원된다.

- `include_realm`

  1로 설정되면 인증된 사용자 보안 규칙의 영역(realm) 이름이 사용자 이름 맵핑을 통해 전달되는 시스템 사용자 이름에 포함된다. 이것은 복수의 영역(realm)에서 사용자를 처리할 때 유용하다.

- `map`

  시스템과 데이터베이스 사용자 이름 사이의 맵핑을 허용한다.

- `krb_realm`

  사용자 보안 규칙 이름과 일치하는 영역(realm)을 설정한다. 이 매개 변수가 설정된 경우 해당 영역(realm)의 사용자만 허용된다. 설정되지 않으면 모든 영역(realm)의 사용자가 연결할 수 있으며, 영역은 사용자 이름 맵핑 완료 여부에 달려 있다.

Ident 인증

ident 인증 방법은 클라이언트의 운영 체제 사용자 이름을 ident 서버로부터 획득하고, 허용된 데이터베이스 사용자 이름으로 사용함으로써 작동된다(선택적 사용자 이름 맵핑 사용). 이것은 TCP/IP 연결에서만 지원된다.
다음 구성 옵션이 ident에 대해 지원된다.

- `map`

  시스템과 데이터베이스 사용자 이름 사이의 맵핑을 허용한다.

일부 ident 서버는 원래 머신의 관리자만 알고 있는 키를 사용하여, 리턴된 사용자 이름을 암호화되도록 하는 비표준 옵션이 있다. AgensGraph는 실제 사용자 이름을 결정하기 위해 리턴된 string의 암호를 해제할 방법이 없으므로 ident 서버에서 AgensGraph를 사용하는 경우에는 이 옵션을 사용해서는 안 된다.

피어(peer) 인증

피어(peer) 인증 방법은 클라이언트의 운영 체제 사용자 이름을 커널로부터 획득하고, 허용된 데이터베이스 사용자 이름으로 사용함으로써 작동된다(선택적 사용자 이름 맵핑 사용). 이 방법은 로컬 연결에만 지원된다.
다음 구성 옵션이 피어(peer)에 대해 지원된다.

- `map`

  시스템과 데이터베이스 사용자 이름 사이의 맵핑을 허용한다.

피어(Peer) 인증은 `getpeereid()` 함수 또는 `SO_PEERCRED` 소켓 매개 변수, 유사 메커니즘이 제공되는 Linux와 같은 운영 체제에서만 사용할 수 있다.

LDAP 인증

이 인증 방법은 패스워드 검증 방법으로 LDAP를 사용할 때 외에는 `password`와 유사하게 작동된다. LDAP는 사용자 이름/패스워드 쌍을 검증할 때에만 사용된다. 따라서 LDAP를 인증에 사용하기 전에 사용자가 데이터베이스에 존재해야 한다.
LDAP 인증은 2가지 모드로 수행할 수 있다. 간단한 바인딩 모드라고 하는 첫 번째 방법은 서버가 `prefix username suffix`로 구성된 고유한 이름에 바인딩하는 것이다. 일반적으로 `prefix` 매개 변수는 활성 디렉토리(Active Directory) 환경에서 `cn=` 또는 `DOMAIN\`을 지정하는 데 사용된다. `suffix`는 비 활성 디렉토리(Active Directory) 환경의 나머지 부분을 지정할 때 사용된다.
검색+바인딩 모드라고 하는 두 번째 모드에서 서버는 `ldapbinddn` 및 `ldapbindpasswd`로 지정 및 고정된 사용자 이름과 패스워드를 사용하여 LDAP 디렉토리에 먼저 바인딩한 다음, 데이터베이스에 로그인하려는 사용자를 검색한다. 사용자 및 패스워드가 설정되지 않은 경우 디렉토리에 익명으로 바인딩이 시도된다. `ldapbasedn`의 서브 트리에서 검색이 수행되고 `ldapsearchattribute`와 정확히 일치하는 것을 찾는다. 이 검색에서 사용자를 찾으면, 서버는 연결을 끊고 로그인이 올바른지 검증하기 위해 클라이언트에서 지정된 패스워드를 사용하여 이 사용자로 디렉토리에 다시 바인딩한다. 이 모드는 Apache mod_authnz_ldap 및 pam_ldap 같은 다른 소프트웨어의 LDAP 인증 스키마에서 사용되는 것과 동일하다. 이 방법은 사용자 객체들이 디렉토리에 있을 경우 더 유연하게 작용하지만 두 LDAP 서버 연결을 분리시킨다.
다음 구성 옵션이 양쪽 모드에 사용된다.

- `ldapserver`

  연결할 LDAP 서버의 이름 또는 IP 주소. 공백으로 구분된 서버를 여러 개 지정할 수 있다.

- `ldapport`

  연결할 LDAP 서버의 포트 번호. 포트가 지정되지 않으면 LDAP 라이브러리의 기본 포트 설정이 사용된다.

- `ldaptls`

1로 설정하면 TLS 암호화를 사용하여 AgensGraph와 LDAP 서버가 연결된다. 이것은 LDAP 서버로의 트래픽만 암호화한다. 클라이언트에 대한 연결은 SSL을 사용하지 않는 한 암호화되지 않은 상태가 지속된다.

다음 옵션은 간단 바인딩 모드에만 사용된다.

- **ldapprefix**
  간단한 바인딩 인증 수행 시 DN 바인딩의 사용자 이름 앞에 추가하는 string.
- **ldapsuffix**
  간단한 바인딩 인증 수행 시 DN 바인딩의 사용자 이름 뒤에 추가하는 string.

다음 옵션은 검색+바인딩 모드에만 사용된다.

- **ldapbasedn**
  검색+바인딩 인증 수행 시 사용자 검색을 시작하는 루트 DN.
- **ldapbinddn**
  검색+바인딩 인증 수행 시 검색 수행하기 위해 디렉토리에 바인딩하는 사용자 DN.
- **ldapbindpasswd**
  검색+바인딩 인증 수행 시 검색 수행하기 위해 디렉토리에 바인딩하는 사용자의 패스워드.
- **ldapsearchattribute**
  검색+바인딩 인증 수행 시 검색에서 사용자 이름에 대해 일치하는 속성. 속성이 지정되지 않으면 uid 속성이 사용된다.
- **ldapurl**
  RFC 4516 LDAP URL. 이것은 다른 LDAP 옵션 중 일부를 좀 더 간결한 표준 형식으로 작성하는 다른 방법이다. 기본값은 다음과 같다.

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

  scope는 base, one, sub 중 하나여야 하며, 일반적으로 후자이다. 한 가지 속성만 사용되며, 필터 및 확장 같은 표준 LDAP URL의 다른 설정은 지원되지 않는다. 비 익명 바인딩의 경우 ldapbinddn 및 ldapbindpasswd는 별도의 옵션으로 지정되어야 한다.

  암호화된 LDAP 연결을 사용하려면 ldapurl 외에도 ldaptls 옵션을 사용해야 한다. ldaps URL 스키마(다이렉트 SSL 연결)는 지원되지 않는다.

  LDAP URL은 현재 Windows가 아니라 OpenLDAP에서만 지원된다.

간단한 바인딩의 구성 옵션과 검색+바인딩의 옵션을 혼용하는 것은 에러이다.
간단한 바인딩 LDAP 구성의 예시는 다음과 같다.

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

데이터베이스 사용자 `someuser`로 데이터베이스 서버에 연결이 요청된 경우 AgensGraph는 DN `cn=someuser` 및 `dc=example, dc=net`, 클라이언트에서 제공된 패스워드를 사용하여 LDAP 서버에 바인딩을 시도한다. 해당 연결이 성공하면 데이터베이스 액세스가 허용된다.
검색+바인딩 구성의 예시는 다음과 같다.

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

데이터베이스 사용자 `someuser`로 데이터베이스 서버에 연결이 요청된 경우 AgensGraph는 익명으로(`ldapbinddn`가 지정되지 않았으므로) LDAP 서버에 바인딩을 시도하고 지정된 베이스 DN 아래에서 `(uid=someuser)`에 대한 검색을 수행한다. 항목이 발견되면 발견된 정보와 클라이언트가 제공한 패스워드를 사용하여 바인딩을 시도한다. 해당 제2차 연결이 성공하면 데이터베이스 액세스가 허용된다.
URL로 작성한 동일한 검색+바인딩 구성은 다음과 같다.

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

LDAP에 대한 인증을 지원하는 일부 다른 소프트웨어는 동일한 URL 형식을 사용하므로 설정을 공유하기 쉬워진다.

RADIUS 인증

이 인증 방법은 패스워드 검증 방법으로 RADIUS를 사용할 때 외에는 `password`와 유사하게 작동된다. RADIUS는 사용자 이름/패스워드 쌍을 검증할 때에만 사용된다. 따라서 RADIUS를 인증에 사용하기 전에 사용자가 데이터베이스에 존재해야 한다.
RADIUS 인증을 사용 중인 경우 구성된 RADIUS 서버로 액세스 요청 (Access Request) 메시지가 전송된다. 이 요청은 `Authenticate Only` 유형이며, `user name` 및 `password`(암호화됨), `NAS Identifier`에 대한 매개 변수가 포함된다. 요청은 서버와 공유되는 시크릿을 사용하여 암호화된다. RADIUS 서버는 `Access Accept` 또는 `Access Reject`를 응답한다. RADIUS 계정에 대한 지원은 없다.
다음 구성 옵션이 RADIUS에 대해 지원된다.

- **`radiusserver`**

  연결할 RADIUS 서버의 이름 또는 IP 주소. 이 매개 변수는 필수이다.

- **`radiussecret`**

  보안을 유지하면서 RADIUS 서버와 통신할 때 사용되는 공유 시크릿. 이것은 AgensGraph 및 RADIUS 서버에서 값이 정확하게 동일해야 한다. 최소 16자의 string이 권장된다. 이 매개 변수는 필수이다.

- **`radiusport`**

  연결할 RADIUS 서버의 포트 번호. 포트가 지정되지 않으면 기본 포트 1812가 사용된다.

- `radiusidentifier`

  RADIUS 요청에서 NAS Identifier로 사용되는 string. 이 매개 변수는 예를 들면, 사용자가 인증하려는 데이터베이스 사용자를 식별하여 RADIUS 서버에서 제2의 매개 변수로 사용될 수 있다. 식별자가 지정되지 않으면 기본 postgresql이 사용된다.

인증서 인증

이 인증 방법은 SSL 클라이언트 인증서를 사용하여 인증을 수행한다. 따라서 SSL 연결에서만 사용 가능하다. 이 인증 방법을 사용하는 경우 서버는 클라이언트가 유효한 인증서를 제공할 것을 요구한다. 패스워드 프롬프트는 클라이언트로 전송되지 않는다. 인증서의 `cn`(공통 이름) 속성은 요청된 데이터베이스 사용자 이름과 비교되며, 일치하는 경우 로그인이 허용된다. 사용자 이름 맵핑을 사용하여 `cn`을 데이터베이스 사용자 이름과 다르게 할 수 있다.
다음 구성 옵션이 SSL 인증서 인증에 대해 지원된다.

- `map`

  시스템과 데이터베이스 사용자 이름 사이의 맵핑을 허용한다.

PAM 인증

이 인증 방법은 인증 메커니즘으로 PAM(Pluggable Authentication Modules)을 사용할 때 외에는 `password`와 유사하게 작동된다. 기본 PAM 서비스 이름은 `postgresql`이다. PAM은 사용자 이름/패스워드 쌍을 검증할 때에만 사용된다. 따라서 PAM을 인증에 사용하기 전에 사용자가 데이터베이스에 존재해야 한다.
다음 구성 옵션이 PAM에 대해 지원된다.

- `pamservice`

  PAM 서비스 이름.

4.1.4 인증 문제

인증 실패 및 관련 문제는 일반적으로 다음과 같은 에러 메시지를 통해 드러난다.

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

이것은 사용자와 연결할 수 없다는 뜻이다. 메시지에 나타난 대로, 서버는 `pg_hba.conf` 환경 설정 파일에서 일치하는 항목을 찾기 못해서 연결 요청을 거부했다.

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

이 메시지는 사용자가 서버에 접촉했으며, 사용자와의 연결은 가능하지만 `pg_hba.conf` 파일에 지정된 인증 방법을 통과해야 함을 의미한다. 사용자가 입력한 패스워드를 검사하거나, 이러한 인증 유형에 문제가 있는 경우에 사용자의 Kerberos 또는 ident 소프트웨어를 검사해야 한다.

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

표시된 데이터베이스 사용자 이름을 찾지 못했다.

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

연결하려는 데이터베이스가 존재하지 않는다. 데이터베이스 이름을 지정하지 않으면 데이터베이스 사용자 이름을 데이터베이스 이름으로 간주한다.

4.2 Role

이 장에서는 role을 생성 및 관리하는 방법을 설명한다. 여기서 소개하고 있는 내용은 AgensGraph가 PostgreSQL 기반으로 개발되었기 때문에, GDB side가 아닌 RDB side의 기능소개는 PostgreSQL의 기술문서를 인용하였다. AgensGraph는 role이라는 개념을 사용하여 데이터베이스 액세스 권한을 관리한다. role은 데이터베이스가 설정된 방법에 따라 데이터베이스 사용자 또는 데이터베이스 사용자 그룹으로 생각할 수 있다. role은 데이터베이스 개체(예 : 테이블)를 소유할 수 있으며 해당 개체에 대한 권한을 다른 role에 할당하여 해당 개체에 액세스할 수 있는 사용자를 제어할 수 있다. 또한, role의 멤버십을 다른 role에 부여할 수 있으므로 멤버 role이 다른 role에 할당된 권한을 사용하도록 할 수 있다.

4.2.1 데이터베이스 role

데이터베이스 role은 개념상 운영 체제 사용자와 완전히 다르다. 사실, 대응 관계를 유지하는 것이 편리할 수 있지만, 그럴 필요는 없다. 데이터베이스 role은 데이터베이스 클러스터 설치 간에 전역적이다(데이터베이스에 개별적이지 않음). role을 생성하려면 CREATE ROLE SQL 명령을 사용해야 한다.

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

name은 SQL 식별자에 대한 규칙(특수 문자 또는 큰따옴표 사용 안함)을 준수한다. 기존 role을 삭제하려면 유사한 DROP ROLE 명령을 사용해야 한다.

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

편의상, 쉘 명령줄에서 호출할 수 있는 프로그램 createuser 및 dropuser를 SQL 명령의 래퍼(wrapper)로 제공된다.

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

기존 role 집합을 판단하려면 pg_roles 시스템 카탈로그를 검사해야 한다. 예를 들면,

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

psql 프로그램의 \du 메타 명령도 기존 role을 나열할 때 유용하다.

4.2.2 role 속성

데이터베이스 role은 권한을 정의하는 속성이 다수 있으며, 클라이언트 인증 시스템과 인터랙션한다.

- **로그인 권한**

데이터베이스 연결을 위한 초기 role 이름으로 LOGIN 속성이 있는 role만 사용할 수 있다. LOGIN 속성이 있는 role은 ``데이터베이스 사용자'' 와 동일한 것으로 간주될 수 있다. 로그인 권한이 있는 role을 생성하려면 다음 중 하나를 사용해야 한다.

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

(CREATE USER는 디폴트로 LOGIN 권한을 준다는 점에서 CREATE ROLE과 다르다.)

- **Superuser 상태**

  데이터베이스 superuser는 로그인 권한을 제외한 모든 권한 검사를 건너 뛴다. 이 권한은 위험하며, 무심코 사용해서는 안 된다. 작업 대부분은 superuser 이외의 다른 role로 수행하는 것이 좋다. 새 데이터베이스 superuser를 생성하려면 CREATE ROLE name SUPERUSER를 사용해야 한다. superuser인 role로 이것을 수행해야 한다.

- **데이터베이스 생성**

  데이터베이스를 생성하려면 권한이 명시적으로 role에 주어져야 한다(모든 권한 검사를 건너 뛰는 superuser인 경우 제외). 이러한 role을 생성하려면 CREATE ROLE name CREATEDB를 사용해야 한다.

- **role 생성**

  role을 추가적으로 생성하려면 권한이 명시적으로 role에 주어져야 한다(모든 권한 검사를 건너 뛰는 superuser인 경우 제외). 이러한 role을 생성하려면 CREATE ROLE name CREATEROLE을 사용해야 한다. CREATEROLE 권한이 있는 role은 다른 role을 변경 및 삭제할 수 있으며, 멤버십을 부여 또는 취소할 수도 있다. 단, superuser role의 멤버십을 생성, 변경 (alter), 삭제 또는 변경 (change) 하려면 superuser 상태가 필요하다. CREATEROLE로는 부족하다.

- **복제 초기화**

  streaming replication을 초기화하려면 권한이 명시적으로 role에 주어져야 한다(모든 권한 검사를 건너 뛰는 superuser인 경우 제외). streaming replication에 사용되는 role은 항상 LOGIN 권한이 있어야 한다. 이러한 role을 생성하려면 CREATE ROLE name REPLICATION LOGIN을 사용해야 한다.

- **패스워드**

  패스워드는 데이터베이스에 연결할 때 사용자가 패스워드를 입력해야 하는 클라이언트 인증 방법인 경우에만 중요하다. password 및 md5 인증 방법은 패스워드를 이용한다. 데이터베이스 패스워드는 운영체제 사용자 패스워드와 구분된다. role 생성 시 패스워드 지정은 CREATE ROLE name PASSWORD 'string'을 사용해야 한다.

4.2.3 role 멤버십

권한 관리의 편의상 사용자를 그룹으로 묶는 것이 편리할 수 있다. 이렇게 하면 권한을 그룹 단위로 부여하거나 취소할 수 있다. AgensGraph에서 이것은 그룹을 나타내는 role을 생성한 다음, 그룹 role의 멤버십을 개별 사용자 role에 부여하면 된다.
그룹 role을 설정하려면 먼저 role을 생성해야 한다.

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

일반적으로 그룹으로 사용되는 role은 LOGIN 속성이 없으며, 원하면 설정은 할 수 있다.
그룹 role이 존재하는 경우 GRANT 및 REVOKE 명령을 사용하여 멤버를 추가 및 삭제할 수 있다.

```
db=# GRANT group_role TO role1, ... ;
db=# REVOKE group_role FROM role1, ... ;
```

다른 그룹 role에도 멤버십을 부여할 수 있다(그룹 role과 비 그룹 role 사이에 실제로는 구분이 없음). 데이터베이스는 순환식 멤버십 루프의 설정을 허용하지 않는다. 또한, role의 멤버십을 PUBLIC에 부여하는 것도 허용하지 않는다.
그룹 role의 멤버는 두 가지 방법으로 role의 권한을 사용할 수 있다. 첫째, 그룹의 모든 멤버는 명시적으로 SET ROLE을 수행하여 일시적으로 그룹 role이 ``된다''. 이 상태에서 데이터베이스 세션은 원래의 로그인 role이 아닌 그룹 role에 대한 액세스 권한을 가지며, 생성된 데이터베이스 개체는 로그인 role이 아닌 그룹 role이 소유하는 것으로 간주된다. 둘째, INHERIT 속성이 있는 멤버 role은 해당 role에서 상속된 모든 권한을 비롯하여 멤버로서 role의 권한을 자동으로 갖는다. 예를 들면, 다음을 실행했다고 가정하자.

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

joe role로 연결한 직후에 데이터베이스 세션은 joe에 직접 부여된 권한 외에도, joe는 admin의 권한을 ``상속 받기'' 때문에 admin에 부여된 권한도 사용한다. 그러나, joe가 간접적으로 wheel의 멤버지만 이 멤버십은 NOINHERIT 속성을 갖는 admin을 통한 것이므로 wheel에 부여된 권한은 사용할 수 없다.

```
db=# SET ROLE admin;
```

위 명령을 실행하면, 세션은 admin에 부여된 이러한 권한만 사용하고 joe에 부여된 권한은 사용하지 않는다.

```
db=# SET ROLE wheel;
```

위 명령을 실행하면, 세션은 wheel에 부여된 이러한 권한만 사용하고 joe 또는 admin에 부여된 권한은 사용하지 않는다. 원래의 권한 상태는 다음 중 하나를 사용하면 복원된다.

```
db=# SET ROLE joe;
db=# SET ROLE NONE;
db=# RESET ROLE;
```

role 속성 LOGIN 및 SUPERUSER, CREATEDB, CREATEROLE은 특수한 권한으로 생각될 수 있지만 데이터베이스 개체의 일상적인 권한으로 상속되지 않는다. 속성을 사용하려면 이러한 속성 중 하나를 보유한 특정 role에 실제로 SET ROLE을 해야 한다. 위의 예시에 이어서, CREATEDB 및 CREATEROLE을 admin role에 부여할 수도 있다. 그러면, joe role로 연결하는 세션은 이러한 권한을 즉각 갖지는 못하며, SET ROLE admin을 수행한 이후에만 권한이 부여된다.
그룹 role을 소멸하려면 DROP ROLE을 사용해야 한다.

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

그룹 role의 멤버십이 자동 취소된다(단, 멤버 role은 영향을 받지 않음). 그룹 role이 소유한 개체를 먼저 삭제하거나 다른 소유자에게 재할당해야 하며, 그룹 role에 부여된 권한은 취소해야 한다는 점을 유의해야 한다.

5 Backup & Recovery

모든 데이터베이스 시스템이 그렇듯이, AgensGraph 백업은 아주 중요한 사항이다. 대부분의 서비스는 그 중요한 자료들은 대부분 데이터베이스에 보관하고 있기 때문이다. 백업과 복원 (restore) 과정은 비교적 간단한 작업이지만, 이 과정에 기술적인 부분과 개념적인 부분에 대해서 분명하게 숙지해 둘 필요가 있다. 여기서 소개하고 있는 내용은 AgensGraph가 PostgreSQL 기반으로 개발되었기 때문에, GDB side가 아닌 RDB side의 기능소개는 PostgreSQL의 기술문서를 인용하였다.
이 장에서는 파일 시스템 기반 백업 및 archive 모드 백업을 설명한다.

5.1 Backup

5.1.1 File system 기반 backup

데이터 저장 공간 전체를 직접 파일 복사하는 식으로 파일 시스템 차원에서 처리하는 방식이 있다. 운영체제에서 파일 시스템 백업하는 것과 크게 다르지 않다. 예를 들면,

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

위와 같은 명령으로 백업할 수 있다. 하지만 이 방식에는 다음 두 가지 한계가 있어, 오히려 pg_dump 명령을 사용하는 것보다 실용적인 측면에서 별 도움이 안된다.

1. 정상적인 백업 작업을 하려면 반드시 데이터베이스 서버를 중지해야한다. 모든 클라이언트의 접속을 막고 파일 시스템을 복사하는 방식도 안전한 백업을 보장하지 않는다. (tar 명령어나 기타 이 비슷한 도구들 모두 파일 시스템 상태에 대한 일관된 스냅샷을 제공하기 어렵고, 서버에서 사용하는 내부적인 버퍼링에 대한 정보를 정확하게 반영할 수 없기 때문이다). 또한 마찬가지로 복원할 때는 반드시 서버는 중지된 상태여야 한다.

2. 만일 데이터베이스에서 사용하는 각종 파일들의 위치와 용도를 잘 알고 있어, 개별 데이터베이스나 테이블에 해당하는 파일들만 복사해서 사용할 수 있을 것이다는 생각을 할 수도 있으나, 그렇게 한다고 해도, 커밋 로그 파일 없이는 정상적으로 작동하지 않을 것이다. pg_clog/* 파일들이 모든 트랜잭션의 커밋 상태를 보관하고 있다. 이 로그 파일들은 테이블 단위가 아니라, 데이터베이스 클러스터 단위로 처리되기 때문에, 개별 테이블의 물리적인 파일과 로그 파일을 복사한다고 해도 다른 테이블과 관계될 가능성이 있어, 부분 백업은 불가능하다.

파일 시스템 기반 백업에서 또 하나 고려해야할 사항은 파일 시스템이 ``일관적인 (consistent) 스냅샷'' 기능을 제공한다면, (그리고, 이 기능이 신뢰성이 있다면) 이 기능을 사용할 때 주의 해야한다는 점이다. 이 파일 시스템 스냅샷 기능을 이용한 전형적인 백업 방법은 먼저 ``동결된 (frozen) 스냅샷'' 을 만들고, 백업 장치로 모든 데이터

클러스터 모든 파일(위에서 언급했듯이, 부분 파일을 백업하는 것은 의미가 없다)을 복사하고, 잠긴 스냅샷을 푸는 방식으로 진행된다. 파일 시스템이 이런 방식의 기능을 제공하면, 데이터베이스 서버가 운영 중일 때도 백업이 가능하다. 이 때 운영 중 상태에서 백업을 받았기 때문에, 데이터베이스 서버가 제대로 종료되지 않은 상태로 데이터베이스 파일이 저장된다. 백업 데이터로 데이터베이스 서버를 시작하면 이전 서버가 충돌된 것으로 인식하고 WAL 로그를 리플레이 할 것이다. 하지만, 백업 시작 전에 CHECKPOINT 명령을 실행했고, 백업 작업 중간에 생긴 WAL 로그들을 따로 보관하고 있으면, 별 문제 없이 실행할 수 있다. (물론 WAL 로그를 무시 할 수도 있다.) 문제는 데이터베이스 서버가 여러 파일 시스템을 쓰는 경우다. 가령 예를 들어서 테이블스페이스를 만들고, 그 테이블스페이스가 다른 파티션을 사용한다면 데이터베이스 서버가 사용하는 모든 파일 시스템에 대해서 동시에 ``동결된(frozen) 스냅샷''을 만들 수 있어야 한다. 이 방식으로 백업을 하려면 먼저 파일 시스템 관련 문서를 세심하게 읽고, 타당성을 따져 본 후 작업하길 바란다.

동시에 여러 파일 시스템 스냅샷을 만들 수 없으면, 스냅샷을 만드는 동안 데이터베이스 서버를 중지해야한다.

다른 한 방법은 rsync 응용 프로그램을 이용하는 방법이다. 작업은 두 단계로 진행 된다. 먼저 서버가 실행 중일 때, 자료를 모두 동기화 하고, 그 다음에 서버를 중지하고 한 번 더 동기화를 한다. 이렇게 하면, 서버가 중지되면서 변경된 파일만 동기화를 하기 때문에 서버 중지 시간을 최소화 할 수 있다.

파일 시스템 백업은 SQL 덤프 보다 일반적으로 많은 백업 공간이 필요하다. (pg_dump 명령으로 만들어진 덤프 파일에는 인덱스가 물리적으로 없고, 인덱스를 만드는 명령어만 들어있기 때문이다.) 이 때문에, 복원 작업에 걸리는 시간은 파일 시스템 백업이 훨씬 빠르다.

5.1.2 Archive mode backup

WAL(Write Ahead Logging)의 archive 만들기

내부적으로 AgensGraph 시스템은 데이터베이스를 조작하는데 끊임 없이 순차적으로 WAL 레코드를 만든다. 이것을 물리적인 디스크 공간에 저장하기 위해서 WAL 세그먼트 파일로 나눠서 저장한다. 이 하나의 WAL 파일은 기본적으로 16MB 크기의 파일이다(이 크기는 서버를 컴파일 할 때 결정된다). 이 파일명은 WAL 순서에 따른 해당 번호를 사용한다. WAL 아카이브 파일을 만들지 않도록 설정하면 이 파일은 몇 개만 만들어지며, 이것 가운데 더이상 사용하지 않는 로그 파일을 찾아서 ``재사용''한다. 내부적으로 WAL 레코드들의 상태 정보를 찾아서 이미 체크포인트 작업이 일어난 것에 대해서는 더 이상 사용하지 않는 상태로 바꾸고, 그 자리에 새로운 WAL 레코드를 기록하는 방식으로 재사용한다.

WAL 아카이브 파일을 만들면 어떤 WAL 세그먼트 파일을 재사용하기 전에 기존에 있던 WAL 레코드 정보를 다른 곳(같은 호스트 내일 수도 있고, NFS 마운트 위치일 수도 있고, 심지어 테이프 같은 곳일 수도 있다)에 따로 보관하는 작업을 한다. 보관하는 방법은 관리자가 직접 지정한다. 이미 똑같은 이름의 파일이 있는 경우에 대해서도 관리자가 결정하도록 한다. 다만 AgensGraph 서버에서는 WAL 세그먼트 파일을 재사용할 때 그 전에 관리자가 설정한 ``보관하는 방법''에 따른 작업만 수행한다. 이 아카이브 보관 방법으로 단순하게 cp 명령을 이용할 수도 있고, 이보다 훨씬 복잡한 사용자 정의 쉘 스크립트를 사용할 수도 있고, 백업 솔루션의 명령을 쓸 수도 있다. 이 부분은 전적으로 관리자 몫이다.

WAL 아카이브 파일을 만들려면, 환경설정 매개변수 wal level의 값으로 archive (또는 hot_standby)를 지정한다. 그 다음 환경설정 매개변수 archive mode 값에는 on, 환경설정 매개변수 archive command 값에는 적당한 시스템 명령어를 지정한다. 이 설정은 모두 postgresql.conf 파일 안에서 설정한다. archive_command 값으로 지정할 쉘 명령어에는 %p (WAL 로그파일 절대경로), %f (보관할 로그 파일 이름) 예약 인자를 사용할 수 있다. % 글자 그대로 써야할 경우면 %%를 입력한다. 일반적으로 이 설정 값은 다음과 같은 형태로 사용된다.

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

위 설정은 단순하게 WAL 세그먼트 파일을 /mnt/server/archivedir 디렉토리에 이름을 똑같게 해서 복사하는 설정이다. 위 예제는 Unix와 Windows 운영체제에 대한 한 예제일 뿐이다. 실제 설정은 해당하는 운영체제에 맞게 변경 되어야한다. %p %f 예약 인자들은 다음과 같이 변경되어 실제로 명령이 실행된다.

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

이렇게 실행 하면 WAL 로그파일을 항상 특정 위치에 새롭게 만들 것이다.

이 아카이브 명령은 AgensGraph 서버를 실행했던 시스템 사용자 권한으로 실행된다. 그래서 WAL 세그먼트 파일을 처리하는 것과 같이 이 파일의 아카이브 파일도 처리시 보안 정책을 고려해야한다. 누구나 읽고 덮어 쓰고 지울 수 있으면 치명적인 보안 사고가 발생할 수 있음을 주의해야 한다.

또 하나 주의할 사항은 아카이브 명령의 쉘 실행 리턴값은 그 명령이 성공 했을 경우 0(zero) 아니면 다른 값이 되도록 해야한다. 서버는 이 리턴값으로 로그 파일을 잘 보관했는지 실패했는지를 판단하고, 성공했으면 원본 로그 파일을 지우거나 재사용하고, 실패하면 성공할 때까지 재시도한다.

이 아카이브 명령으로 이미 있는 파일을 덮어쓰지 않도록 해야한다. 덮어 쓰면 복원 작업시 의도치 않은 오류가 발생할 수 있다. (해당 파일은 이미 다른 데이터베이스에서 생성했거나 사용할 수도 있기 때문이다.) 로그 파일이 이미 있으면 끊임 없이 오류를 발생시키기 때문에 관리자가 수동으로 처리하는 것이 가장 안전하다.

로그 파일을 보관 하는 작업을 하기 전에 먼저 그 파일이 이미 존재 하는지 확인하고, 존재하는 경우 해당 명령어가 0 아닌 값을 리턴하도록 설정하는 것이 좋다. Unix에서는 이 작업을 위해, test 명령을 제공하고 있다. 일부 Unix 플랫폼에서는 이미 있는 파일을 덮어쓰지 않기 위해 cp 명령에서 -i 옵션을 제공하는데 이 명령을 사용할 때는 반드시 리턴값을 확인해야 한다. (GNU cp 명령은 로그 파일이 존재하는 경우 0 값을 리턴하는데, 이는 바람직하지 않다.)

WAL 파일을 따로 저장하는 작업을 할 때, 운영상 작업을 수동으로 중지 하는 경우 혹은 저장 공간이 부족해서 저장 작업이 실패하는 경우가 반복될 수 있으므로 잘 생각해 봐야 한다. 예를 들어, WAL 세그먼트 파일을 테이프 저장장치로 보관하려고 하는데, 해당 장치가 자동으로 테이프 교환을 하지 않고 테이프에 여유 공간이 없으면 사용자가 테이프를 바꾸기 전까지 계속해서 파일 기록 작업을 실행했다가 오류를 내고 다시 실행할 것이다.

이렇게 되면, pg_xlog/ 디렉토리에서는 테이프 저장장치로 자료가 안전하게 복사되지 않았기 때문에 해당 WAL 세그먼트 파일을 삭제하거나 재사용하지 않을 것이고, 이후 만들어지는 새로운 WAL 세그먼트 파일들은 계속 쌓여서 결국 pg_xlog/ 디렉토리의 여유 공간이 없어지고 서버가 PANIC 오류를 발생시키면서 중지하게 된다.

또한 WAL 파일을 따로 저장할 때의 그 저장 장치의 자료 기록 속도도 함께 고민해야 한다. 정상적으로 작업이 진행된다 하더라도 WAL 파일을 만들어내는 속도가 그 파일을 따로 보관하는 속도보다 빠르면, 똑같은 문제가 발생할 수 있다. 물론 따로 보관하는 속도가 다소 늦더라도 정상적으로 이루어지고 pg_xlog/ 디렉토리에 여유공간이 충분히 있으면 별 문제 되지는 않지만, 그렇지 않을 경우 위 문제가 발생할 여지가 있다. 이 기능을 사용하기 전에 충분히 이 부분에 대한 문제를 고민해 보아야하며, 관리자는 이 기능이 의도된 대로 잘 작동하는지 감시해야한다.

저장 파일의 경로는 최대 64 개의 ASCII 글자면 되고, 파일 이름은 %f 예약어를 사용해야하며, (즉, 디렉토리까지만 바꿀 수 있다.) 원본 WAL 세그먼트 파일은 %p 예약어를 사용해야한다.

WAL 파일에는 트랜잭션 작업 정보만 담겨있기 때문에 postgresql.conf, pg_hba.conf, pg_ident.conf 파일의 변경 사항이 이 백업 방법으로는 복원 되는 데이터베이스에 반영되지 않는 점을 주의해야 한다. 이 환경설정 파일 백업은 시스템의 일반 파일 백업 정책에 따라 백업하길 바란다.

아카이브 명령은 WAL 파일 가운데 서버에 모두 반영된 (롤백되거나 커밋 되어 체크포인트 작업이 끝난) 파일에 대해서 실행된다. 즉, 작업량이 아주 적은 데이터베이스의 경우라면 아카이브 명령이 실행되는 간격이 아주 길어진다. 이 간격에서 데이터베이스 장애가 생기면 자료 손실이 그 간격 동안 생기게 되므로 WAL 세그먼트 파일의 모든 내용이 처리되기 전에 특정 시간이 지나면 강제로 아카이브 명령을 사용해서 이 세그먼트 파일을 따로 저장해야 하는데, 이는 archive_timeout 값을 짧게 지정하면 된다. 이 설정값을 너무 짧게 잡으면 디스크 공간을 낭비하는 단점이 생긴다. 일반적으로 이 설정값은 분 단위가 적당하다.

또한 사용자가 강제로 세그먼트 파일을 pg_switch_xlog 함수를 이용해서 바꿀 수 있다. 일반적으로 대량 자료 입력, 변경, 삭제 작업이 일어나고 이것에 대한 즉시 백업이 필요한 경우에 이 함수를 사용한다.

wal_level 값을 minimal 로 설정하게 되면 이 WAL 로그를 따로 보관해서 그것을 기반으로 복원 하는 기능을 사용할 수 없다. 이 설정을 사용하면 WAL 파일에 복원 관련 정보를 보관하지 않기 때문이다. 그렇기 때문에 wal_level 설정값을 변경하면 서버를 재실행해야한다. 하지만 archive_command 설정값은 환경 설정 파일을 다시 로드하는 것으로도 충분하다. 필요에 따라서 운영 중에 이 작업이 중지되어야할 필요도 있기 때문이다. 그렇게 하려면 archive_command 설정값으로 빈문자열 ('') 로 지정하면 된다. 그러면 archive_command 설정값이 다시 WAL 파일을 복사하는 작업을 하기 전까지 pg_xlog/ 디렉토리에 계속 남아있게 된다.

5.2 Recovery

5.2.1 Archive 모드 백업을 이용한 복구

문제가 발생했을 때 아카이브 모드 백업 방식으로 보관해둔 백업을 가지고 데이터베이스 서버를 복구하는 방법에 대해서 설명한다. 작업 순서는 다음과 같다.

1. 서버가 실행 중이면, 먼저 서버를 중지한다.

2. 만일 시스템의 디스크 공간이 넉넉하면, 데이터베이스 클러스터 디렉토리 전체 및 관련된 사용자 정의 테이블스페이스의 파일들과 필요한 WAL 세그먼트 파일들을 모두 임시 장소에 복사해야 한다. 이 작업을 기존에 운영 중이던 시스템에서 하려면, 적어도 두 배 이상의 디스크 공간이 필요하게 된다. 이런 여유 공간이 없으면 적어도 기존 데이터베이스 서버의 클러스터 디렉토리 안에 있는 pg_xlog 파일들과 서버 환경 설정 파일들 만이라도 저장한다. 백업한 WAL 세그먼트 파일들과 미처 백업 되지 못한 WAL 세그먼트 파일들을 모두 저장하면 서버가 중지되기 직전의 상태로 복구될 수 있다.

3. 기존 서버용 데이터 클러스터 디렉토리와 기존 테이블스페이스용 디렉토리를 모두 지운다.

4. 백업 파일을 원래의 위치에 그대로 복사한다. 이 때 작업하는 사용자는 데이터베이스 서버를 실행하는 시스템 사용자여야한다. (root로 작업하면, 반드시 해당 소유주로 변경해야한다!) 그래서 파일 접근 권한과 소유주를 시스템 사용자와 같게 맞추어야한다. 다음 사용자 정의 테이블스페이스를 사용할 때 pg_tblspc/ 디렉토리 내에 심볼릭 링크로 되어 있는 원본 디렉토리와 그 안에 있는 모든 파일을 복사해 둔다.

5. pg_xlog/ 디렉토리는 비워둔다. 만일 이 디렉토리 안에 파일이 있으면 지워야한다. 복구 작업에서 이 디렉토리 안에 필요한 파일들은 자동으로 만들어진다. 만일 pg_xlog/ 디렉토리를 백업 대상에서 제외했으면 디렉토리를 만들거나 심볼릭 링크를 만들어 데이터베이스 서버가 해당 디렉토리를 사용할 수 있도록 한다. 이 때도 이 디렉토리의 접근권한과 소유주가 데이터베이스 서버를 실행하는 시스템 사용자만 사용할 수 있는지 확인해야한다. 만일 심볼릭 링크면 혹시 이전 데이터베이스에서 사용하는 원본 경로와 같아서 이전 자료들을 덮어쓸 수도 있는지 꼭 확인해야한다.

6. 만일 WAL 세그먼트 파일에 대한 백업본이 없으면 2단계에서 복사해 둔 pg_xlog/ 내의 파일들을 복구하려는 곳으로 복사한다. (옮기지 말고 복사하길 권한다. 아직 복구가 완벽하게 이루어지지 않았기 때문에, 복사가 안전하다.)

7. 데이터베이스 클러스터 디렉토리 안에 recovery.conf 파일을 만든다. 또한 복구 과정 중에 외부에서 접근할 수 없도록 임시로 pg_hba.conf 파일을 수정하는 것도 좋은 방법이다.

8. 서버를 실행한다. 서버는 복구 모드로 가동되면서 필요한 WAL 파일들을 찾아서 반영되지 않았던 트랜잭션들을 일괄 처리하기 시작한다. 복구 작업 도중 외부 영향으로 서버가 중지 되면, 단순히 서버를 재실행해서 복구 작업을 계속 진행한다. 복구 작업이 끝나면 서버는 복구 모드로 재실행 되는 것을 막기 위해 recovery.conf 파일을 recovery.done 이름으로 바꾸고 정상 실행 상태로 클라이언트의 접속을 기다린다.

9. 이제 데이터베이스로 접속해서 자료가 정상인지 살펴보고 만일 원하는 대로 복구 되지 않았으면 서버 로그를 살펴보면서 1단계부터 다시 진행한다. 자료가 모두 정상이고, 데이터베이스 상태도 정상이면 pg_hba.conf 파일의 내용을 원래대로 수정해서 외부 접속도 허용한다.

이 복구 방법의 핵심은 어떻게 백업 WAL 세그먼트 파일을 데이터베이스에 적용시키냐는 것과 그 복구를 어느 시점까지 할 것이냐이다. 이것을 recovery.conf 파일에서 지정한다. 이 파일을 만드는 간단한 방법은 먼저 데이터베이스 배포판의 share/ 디렉토리 안에 recovery.conf.sample 파일을 복사해서 필요한 부분만 수정해서

사용하는 것이다. recovery.conf 파일에서 꼭 필요한 부분은 restore_command 부분이다. 이 설정은 AgensGraph 에서 백업본 WAL 세그먼트 파일을 어떻게 적용시킬 것인가에 대한 정의다. 간단하게 설명하면 서버 환경 설정 가운데 archive_command 설정과 반대되는 명령을 지정하면 된다. 여기서 사용되는 예약어는 archive_command 설정값을 지정할 때와 같다. %f 값은 백업 아카이브 디렉토리 안에 있는 파일이름이 되고, %p 값은 트랜잭션 로그 파일로 대체된다. (상대 경로를 사용하면, 서버를 실행하는 현재 디렉토리를 기준으로 처리된다.) %% 예약어는 % 문자로 처리된다. 다음은 일반적인 이 설정값이다.

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

위 설정은 /mnt/server/archivedir 디렉토리 안에 있는 미리 백업 해둔 WAL 세그먼트 파일들을 복구 작업을 진행할 때 사용한다. 물론 이 명령은 사용하는 백업 장치에 따라서 훨씬 복잡할 수도 있으며, 테이프 백업 장치처럼 해당 테이프를 마운트하고 원하는 위치로 이동하는 등 일부 명령어들을 포함하여 직접 쉘 스크립트를 만들어서 사용할 수도 있다.

여기서 중요한 점은 여기서 지정한 작업이 실패 했을 경우 그 작업의 리턴값이 0이 아닌 (non-zero) 것이어야 한다는 점이다. 이 작업은 아카이브 백업 디렉토리 안에 없는 파일도 요청한다. 이 때 이 작업의 결과는 반드시 0 이 아닌 리턴값을 리턴해야한다. 이 상황은 오류가 아니다. 해당 스크립트에서 명령어가 없다거나 서버가 정상 종료되면서 복구작업을 하는 프로세스로 SIGTERM 시그널을 보내는 경우가 아닌 기타 다른 시그널로 그 작업이 중지 되는 경우가 오류 상황이다. 이런 상황에는 복구 작업이 중지 되고, 서버는 정상 작동을 하지 않고 멈춘다.

서버는 .backup 또는 .history 이름으로 끝나는 파일을 찾을 수 있는데, 이 때 그 파일이 없으면 그 파일이 없다고 (0 아닌 값을 리턴함으로써) 서버 쪽에 알려주어야한다. 또한, %f 파일 이름과 서버 쪽으로 복사되는 %p 파일 이름이 항상 같지는 않다. 그렇기 때문에, 쉘 스크립트를 직접 만들어 사용하면 이 부분에 대한 처리에 실수가 없도록 주의해야한다.

아카이브 백업 디렉토리에서 WAL 세그먼트 파일을 못 찾으면 데이터 클러스터 안에 있는 pg_xlog/ 디렉토리 안에서 찾는다. 혹시 처리해야할 WAL 세그먼트 파일이 더 있는데 미처 백업 되지 않은 것이 있으면 그것까지도 처리해 준다. 하지만 똑같은 파일이름으로 아카이브 백업 디렉토리 안에 이미 그 파일이 있으면 그 파일이 적용되고, pg_xlog/ 안에 있던 파일은 무시되고 새로운 WAL 세그먼트 이름으로 그 로그를 반영한다. 그렇기 때문에, 복구 작업 전에 반드시 WAL 세그먼트 파일들을 잘 정리해야한다.

일반적으로 복구 작업은 아카이브 백업 디렉토리 안에 가장 마지막 로그 파일을 반영하고 그 다음 파일이 없을 때까지 찾기 때문에, 복구 로그의 마지막은 어떤 파일을 찾을 수 없다는 ``file not found'' 메시지가 출력된다. 또한 복구 작업을 시작하면서 00000001.history 형태의 파일을 찾기도한다. 이 모든 경우는 정상적인 복구 과정 속에서 생기는 로그이다.

복구 작업은 recovery.conf 파일에서 복구 중지 지점을 지정해서 원하는 지점에서 복구 작업을 중지할 수도 있다. 이 지점을 ``복구 타겟 (recovery target)''이라고 한다. 이 기능은 미숙한 데이터베이스 관리자가 실수로 중요 테이블을 삭제하는 것 같은 운영상 실수에 대한 복구를 지원하는데 유용하다. ``복구 타겟''은 특정 시각으로 지정할 수도 있고 관리자가 미리 지정한 어떤 문자열일 수도 있고 특정 트랜잭션 ID로 지정할 수도 있다. 사고가 어느 트랜잭션 ID에서 발생했는지 알 수 있는 툴을 사용할 수 없으면 단순히 특정 시각 또는 관리자가 미리 지정한

어떤 문자열을 지정하는 것이 복구 작업을 하기에

> 복구 작업을 멈출 시점은 베이스 백업 시간 이후여야 한다. pg_stop_backup 명령이 실행된 시간보다 늦은 시점으로 지정해야 한다. 베이스 백업을 사용해서 데이터 클러스터 파일들을 파일 시스템 차원으로 백업하고 있었던 시간대로 복구할 수는 없다. 이 작업이 필요하면 이전 베이스 백업 자료와 해당 WAL 백업 파일들이 필요하다.

만일 WAL 파일 자체에 문제가 있으면 정상적으로 처리한 트랜잭션까지만 적용되고 복구 작업을 멈추고, 서버도 중지된다. 이런 경우에는 문제가 발생한 시점을 파악해서 처음부터 다시 그 시점 전까지만 ``복구 타겟'' 으로 지정해서 복구한다. 복구 작업 도중 외부적인 영향으로 작업이 중지 되면 문제의 원인을 해결해야 한다. 그리고나서 단순히 서버를 재실행하면 복구 작업을 이어서 계속 진행한다. 복구 작업은 일반 실행환경에서의 체크 포인트 작업과 같이 재실행된다. 내부적으로 pg_control 파일을 주기적으로 갱신해서 이미 반영된 로그들에 대해서는 더이상 재작업을 하지 않도록 하기 때문에 중복처리에 대한 염려는 안해도 된다.

6 Configuration

데이터베이스 시스템의 동작에 영향을 주는 환경 변수들에 대해서, 설정하는 법과 각 변수의 의미를 설명한다.

6.1 Configuration Settings Reference

6.1.1 Setting parameters

모든 매개변수 이름은 대소문자를 구분한다. 각 매개변수 값은 boolean 또는 string, integer, floating point, enumerated(enum)의 5가지 타입 중 하나이다. 데이터 타입은 매개변수 설정을 위한 구문을 설정한다.

- **Boolean**: on, off, true, false, yes, no, 1, 0 (대소문자 구문 안함) 또는 t, f, y, n 중 하나로 값을 설정할 수 있다.
- **String**: 일반적으로 앞뒤에 작은따옴표가 표시되며, 값 내의 작은따옴표에는 작은따옴표를 하나 더 붙여준다. 값이 보통 단순한 숫자 또는 식별자일 경우에는 따옴표를 생략할 수 있다.
- **Numeric** (integer 와 floating point): 소수점은 floating-point 매개변수일 때만 허용된다. 천 단위 구분자를 사용하면 안 된다(예: 1,000,000에서 `,'). 따옴표는 불필요하다.
- **단위가 있는 Numeric**: 일부 숫자 매개변수는 메모리 또는 시간을 설명하므로 암시적 단위를 갖고 있다. 단위는 킬로바이트, 블록(보통 8킬로바이트), 밀리초, 초, 분일 수 있다. 이러한 설정들 중 단위가 없는 숫자 값은 설정의 기본 단위를 사용하는데, pg_settings.unit에서 확인할 수 있다. 편의상, 설정은 명시적으로 지정된 단위를 지정할 수 있다. 예를 들면, 시간 값이 '120 ms'인 경우, 매개변수의 실제 단위가 무엇이든 변환된다. 이 기능을 사용하려면 값을 string(따옴표 포함)으로 작성해야 한다는 점에 유의하라. 단위 이름은 대소문자를 구분하며, 숫자 값과 단위 사이에 공백이 올 수 있다.
  - 유효 메모리 단위는 kB(킬로바이트) 및 MB(메가바이트), GB(기가바이트), TB(테라바이트)이다. 메모리 승수는 1024이다(1000이 아니고).
  - 유효 시간 단위는 ms(밀리초), s(초), min(분), h(시) 및 d(일)이다.
- **Enumerated**: Enumerated 타입의 매개변수는 string 매개변수와 작성 방식이 동일하지만 값 집합이 하나로 제한된다. 이 매개변수에서 허용되는 값은 pg_settings.enumvals를 참고할 수 있다. Enum 매개변수 값은 대소문자를 구분하지 않는다.

Parameter Interaction via the Configuration File

이러한 매개변수를 설정하는 가장 기본적인 방법은 일반적으로 데이터 디렉토리에 있는 postgresql.conf 파일을 편집하는 것이다. 데이터베이스 클러스터 디렉토리가 초기화된 경우 기본 사본이 설치된다. 이 파일과 유사한 예시는 다음과 같다.

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

라인당 매개변수 하나가 지정되어 있다. 이름과 값 사이의 등호는 옵션이다. 공백은 중요하지 않으며 (따옴표로 둘러싼 매개변수 제외), 빈 라인은 무시된다. 해시 마크 (#) 는 라인의 나머지가 주석임을 의미한다. 단순 식별자 또는 숫자가 아닌 매개변수 값은 작은따옴표를 사용해야 한다. 작은따옴표를 매개변수 값에 포함하려면 작은따옴표를 하나 더 붙이거나, 역슬래시와 따옴표를 사용해야 한다.

이렇게 설정된 매개변수는 클러스터에 기본값으로 제공된다. 값을 오버라이드하지 하지 않는 이상 활성 세션에서 보이는 설정은 이 값들이다. 다음 절에서는 관리자 또는 사용자가 이러한 기본값을 오버라이드하는 방법을 설명한다.

메인 서버 프로세스가 SIGHUP 신호를 수신할 때마다 환경 설정 파일이 다시 읽히게 된다. 커맨드 라인에서 pg_ctl reload를 실행하거나 SQL 함수 pg_reload_conf() 를 호출하면 SIGHUP가 전송된다. 또한 메인 서버 프로세스는 현재 실행 중인 모든 서버 프로세스에 이 신호를 퍼트려서 기존 세션에도 새 값이 적용되게 한다 (현재 실행 중인 클라이언트 명령이 완료된 후에 진행됨). 또는 사용자가 단일 서버 프로세스에 직접 신호를 전송할 수도 있다. 일부 매개변수는 서버 시작 시에만 설정 가능하다. 환경 설정 파일의 엔트리를 변경하면 서버가 재시작되기 전까지 무시된다. 마찬가지로, 환경 설정 파일에서 잘못된 매개변수 설정도 SIGHUP 처리 중에 무시된다 (단, 로그에는 기록된다).

postgresql.conf 외에 AgesnGraph 데이터 디렉토리 ($AGDATA) 에는 postgresql.auto.conf 파일이 포함되어 있으며, postgresql.conf와 형식은 동일하지만 직접 편집해서는 안 된다. 이 파일에는 ALTER SYSTEM 명령을 통해 제공되는 설정이 포함되어 있다. postgresql.conf가 존재할 때마다 이 파일을 자동으로 읽어오고, 해당 설정이 동일하게 적용된다. postgresql.auto.conf의 설정은 postgresql.conf의 설정보다 우선시 된다.

Parameter Interaction via SQL

AgensGraph는 환경 설정 기본값을 설정하기 위한 3가지 SQL 명령을 제공한다. 앞에서 언급한 ALTER SYSTEM 명령은 SQL 구문으로 전역 기본값을 변경할 수 있는 방법을 제공하는데, 기능상 postgresql.conf를 편집하는 것과 동일하다. 또, 데이터베이스별로 또는 role별로 기본값 설정이 가능한 명령이 2가지 있다.

- ALTER DATABASE 명령은 전역 설정을 데이터베이스별로 오버라이드한다.
- ALTER ROLE 명령은 전역 및 데이터베이스별 설정을 모두 사용자 지정 값으로 오버라이드한다.

ALTER DATABASE 및 ALTER ROLE로 설정된 값은 데이터베이스 세션을 새로 시작하는 경우에만 적용된다. 이것은 환경 설정 파일 또는 서버 커맨드 라인에서 구한 값을 오버라이드하고 나머지 세션에 대해 기본값을 적용한다. 서버 시작 후에 일부 설정은 변경이 불가하므로 이 명령 (또는 아래 나열된 것 중 하나) 으로 설정할 수 없다는 점에 유의해야 한다.

클라이언트가 데이터베이스에 연결되면 AgensGraph 세션-로컬 환경 설정 설정과 인터랙션이 가능한 SQL 명령 (또는 동등한 함수) 2개를 추가 제공한다.

- SHOW 명령으로 모든 매개변수의 현재 값을 확인할 수 있다. 해당 함수는 current_setting(setting_name text) 이다.
- SET 명령으로는 세션에 로컬로 설정할 수 있는 이 매개변수의 현재 값을 수정할 수 있다. 다른 세션에는 영향을 미치지 않는다. 해당 함수는 set_config(setting_name, new_value, is_local) 이다.

또한, 시스템 뷰 pg_settings는 세션-로컬 값을 확인하고 변경하는 데 사용할 수 있다.

- 뷰 쿼리는 SHOW ALL과 유사하지만, 좀 더 상세한 결과를 보여준다. 또한 필터 조건을 지정하거나 다른 릴레이션과 조인할 수 있어서 좀 더 유연하다.
- 이 뷰에서, setting 칼럼을 업데이트하기 위해 UPDATE를 사용하는 것은 SET 명령을 실행하는 것과 동일하다. 예를 들면,

```
SET configuration_parameter TO DEFAULT;
```

위의 구문은 아래와 동일하다.

```
UPDATE pg_settings SET setting = reset_val WHERE name = 'configuration_parameter';
```

Managing Configuration File Contents

AgensGraph는 복잡한 postgresql.conf 파일을 작은 파일로 세분화하는 기능들을 제공한다. 이 기능들의 환경 설정 방식이 동일하지는 않지만, 관련 있는 서버들을 관리할 때 특히 유용하다.
개별적인 매개변수 설정 외에, postgresql.conf 파일에는 include 지시어가 있다. 읽어올 다른 파일을 지정하여, 환경 설정 파일에 파일이 삽입된 것 같이 처리된다. 이 기능은 환경 설정 파일을 물리적으로 분할 한다. Include 지시어는 간략하게 다음과 같다.

```
include 'filename'
```

파일 이름이 절대 경로가 아니면 참조하는 환경 설정 파일 디렉토리의 상대 경로로 취급된다. include는 중첩이 가능하다.
include_if_exists 지시어도 있는데, 이것은 참조 파일이 존재하지 않거나 파일을 읽을 수 없는 경우 외에는 include 지시어와 동일하게 작동된다. include는 이것을 에러 조건으로 간주하지만, include_if_exists는 단순히 메시지를 로깅하여 참조 환경 설정 파일을 계속 처리 한다.
postgresql.conf 파일에는 include_dir도 포함될 수 있는데, 다음과 같이 포함할 환경 설정 파일의 경로를 지정한다.

```
include_dir '경로'
```

절대 경로가 아니면 참조하는 환경 설정 파일 디렉토리의 상대 경로로 취급된다. 지정된 디렉토리 내에서 디렉토리가 아닌 파일은 이름이 .conf로 끝나는 경우에만 포함된다. 해당 파일이 일부 플랫폼에서 숨겨질 수 있으므로 실수 예방 차원에서 문자 . 로 시작되는 파일 이름도 무시된다. include 디렉토리 내의 파일들은 파일 이름 순으로 처리된다(C 로케일(locale) 규칙에 따라, 예를 들면, 숫자-문자 순 및 대문자-소문자 순).
Include 파일 또는 디렉토리는 postgresql.conf 파일 하나만 쓰지 않고, 데이터베이스 환경 설정을 논리적으로 분리하는 데 사용될 수 있다. 메모리 용량이 각각 다른 데이터베이스 서버 2대를 운용하는 회사를 생각해보자. 로깅과 같은 경우는 데이터베이스 2개가 공유하는 환경 설정 요소가 있을 가능성이 높다. 그러나 서버의 메모리 관련 매개변수는 서로 상이할 것이고, 서버마다 커스텀화 했을 것이다. 이러한 상황을 관리하는 방법은 커스텀화된 환경 설정 변경 내용을 3개의 파일로 분할하는 것이다. 사용자는 아래 코드를 postgresql.conf 파일의 끝에 추가하여 각 파일을 포함하면 된다.

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

모든 시스템의 shared.conf 파일은 동일하다. 메모리 크기가 다른 각 서버는 동일한 memory.conf를 공유할 수 있다. 사용자는 RAM이 8GB인 서버와 16GB인 서버를 모두 한 파일로 관리할 수 있다. 그리고 server.conf에는 서버별 환경 설정 정보가 포함된다.
환경 설정 파일 디렉토리를 생성하고 이 정보를 그 파일에 넣는 방법도 있다. 예를 들면, conf.d 디렉토리는 postgresql.conf의 마지막 엔트리로 참조할 수 있다.

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

그런 다음, conf.d 디렉토리의 파일 이름을 다음과 같이 지정한다.

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

이러한 명명 규칙으로 파일이 로드되는 순서가 명확해진다. 서버가 환경 설정 파일을 읽을 때, 매개변수의 마지막 설정만 적용된다. 이 예시에서, conf.d/02server.conf에서 설정된 값들은 conf.d/01memory.conf에서 설정된 값을 오버라이드한다.
이 방법을 대신 사용하여 파일을 서술적으로 명명할 수 있다.

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

이러한 배치 순서는 각각의 환경 설정 파일 변화에 대해 고유한 이름을 부여한다. 이로써 버전 관리 저장소처럼 몇 개의 서버 환경 설정이 한곳에 저장되는 경우 모호함이 줄어든다.

6.1.2 Memory Consumption

shared_buffers (integer)

- 데이터베이스 서버가 공유 메모리 버퍼용으로 사용하는 메모리 양을 설정한다. 기본값은 일반적으로 128 메가바이트 (128MB) 이지만 커널 설정에서 지원하지 않는 경우 여기에 미치지 못할 수 있다 (initdb 중에 결정됨). 이 설정은 최소 128킬로바이트여야 한다. (기본값이 아닌 BLCKSZ 값은 최소값을 변경한다.) 단, 최소값보다 훨씬 큰 설정은 일반적으로 우수한 성능이 필요할 때 사용된다.
- RAM이 1GB 이상인 전용 데이터베이스 서버를 사용하는 경우 shared_buffers의 적절한 시작 값은 시스템 메모리의 25%이다. 작업 부하는 shared_buffers에 대한 설정이 클수록 효과적이지만, AgensGraph 역시 운영 체제 캐시에 의존적이므로 시스템 효율을 위해 40% 이상의 RAM을 shared_buffers에 할당하는 것은 좋지 않다. 장시간에 걸쳐 대량의 새 데이터 또는 변경된 데이터 쓰기 프로세스를 실행하기 위해 shared_buffers를 더 크게 설정하면 checkpoint_segments에서도 그에 맞게 설정을 증가시켜야 한다.
- 시스템 RAM이 1GB 미만인 경우에는 운영 체제를 위한 적정 공간이 필요하므로 RAM 비율을 더 작게 하는 것이 맞다. 또한 Windows에서 shared_buffers 값을 크게 하는 것은 효과적이지 않다. 설정을 작게 하고, 운영 체제는 캐시는 상대적으로 크게 함으로써 더 나은 결과가 나올 수도 있다. Windows 시스템의 shared_buffers에 대한 유용한 범위는 64MB ~ 512MB이다.

huge_pages (enum)

- Huge page를 활성/비활성으로 설정한다. 유효 값은 try(기본값) 및 on, off이다.
- 현재 이 기능은 Linux에서만 지원된다. try로 설정되면 다른 시스템에서는 무시된다.
- huge pages 페이지를 사용하면 결과적으로 메모리 관리에 더 작은 페이지 테이블과 더 짧은 CPU 시간을 사용하여 성능이 높아진다.
- huge_pages를 try로 설정하면 서버가 huge pages의 사용하지만, 실패 시 일반적인 할당을 사용하는 쪽으로 폴백한다. on의 경우 huge pages 사용에 실패하면 서버가 시작되지 않는다. off의 경우 huge pages를 사용하지 않는다.

temp_buffers (integer)

- 각 데이터베이스 세션이 사용하는 임시 버퍼의 최대 수를 설정한다. 임시 테이블에 액세스하는 용도로만 사용되는 세션-로컬 버퍼가 있다. 기본값은 8메가바이트 (8MB) 이다. 설정은 개별 세션 내에서 변경할 수 있지만, 세션 내 임시 테이블을 처음 사용하기 전에만 가능하다. 이후에 값을 변경하면 해당 세션에서 효과가 없다.
- 세션은 temp_buffers에 설정된 한계까지 필요한 임시 버퍼를 할당한다. 실제로는 임시 버퍼가 많이 필요 없는 세션에서 큰 값을 설정하는 데 드는 비용은 temp_buffers가 증가할 때마다, 버퍼 디스크립터 혹은 약

64바이트에 불과하다. 그러나 버퍼가 실제로 사용되는 경우에는 8192바이트가 추가적으로 필요하다(또는 일반적으로 BLCKSZ 바이트).

max_prepared_transactions (integer)

- 동시에 ``준비된'' 상태일 수 있는 트랜잭션의 최대 수.
- 준비된 트랜잭션을 사용할 계획이 없으면 이 매개변수는 0으로 설정하여 준비된 트랜잭션을 생성하는 실수를 방지해야 한다. 준비된 트랜잭션을 사용하는 경우 max_prepared_transactions가 최소한 max_connections 이상이 되도록 설정하면 세션이 준비된 트랜잭션을 보류시킬 수 있다.
- 스탠바이 서버 실행 시 마스터 서버 값보다 크거나 같게 설정해야 한다. 그렇게 하지 않으면 스탠바이 서버가 쿼리를 허용하지 않는다.

work_mem (integer)

- 임시 디스크 파일을 쓰기 전에 내부 정렬 명령 및 해시 테이블에서 사용되는 메모리 양을 지정한다. 기본값은 4메가바이트이다(4MB). 복잡한 쿼리의 경우 몇 가지 정렬 또는 해시 명령이 병렬로 실행될 수 있다. 각 명령은 데이터를 임시 파일에 쓰기 전에 이 값에 지정된 크기만큼 메모리를 사용할 수 있다. 실행 중인 세션들은 해당 명령을 동시에 실행할 수도 있다. 사용된 총 메모리는 work_mem의 배수가 된다. 정렬 명령은 ORDER BY 및 DISTINCT, 머지 조인에 사용된다. 해시 테이블은 해시 조인, 해시 기반 집계 (aggregation), IN 서브쿼리의 해시 기반 처리에 사용된다.

maintenance_work_mem (integer)

- VACUUM, CREATE INDEX 및 ALTER TABLE ADD FOREIGN KEY 같은 유지보수 명령에서 사용되는 최대 메모리 양을 지정한다. 기본값은 64메가바이트이다(64MB). 이 명령은 데이터베이스 세션에서 한 번에 하나만 실행할 수 있으며, 정상 설치에는 동시 실행되는 명령이 여러 개 있을 수 없다. 이 값은 work_mem보다 훨씬 큰 값으로 설정하는 것이 안전하다. 설정값이 큰 경우에는 vacuuming 및 데이터베이스 덤프 복구 성능이 개선될 수 있다.
- autovacuum 실행 시 이 메모리에서 autovacuum_max_workers의 배수로 할당할 수 있으므로 기본값을 너무 높게 설정하지 않도록 해야 한다. autovacuum_work_mem을 별도로 설정하여 이것을 관리하는 것이 유용할 수 있다.

autovacuum_work_mem (integer)

- 각 autovacuum worker 프로세스에서 사용되는 최대 메모리 양을 지정한다. 기본값은 maintenance_work_mem 값을 대신 사용해야 함을 나타내는 -1이다. 다른 컨텍스트에서 실행하는 경우 이 설정은 VACUUM에 영향을 끼치지 않는다.

max_stack_depth (integer)

- 서버 실행 스택의 최대 안전 깊이를 지정한다. 이상적인 설정은 커널이 강제로 지정한 안전 마진 (safety margin)에서 약간 부족하게 설정하는 것이다(ulimit -s에 의해 설정된 대로 하거나 로컬과 동등하게). 서버의 모든 루틴이 아니라 표현식 평가(expression evaluation) 같이 잠재적 재귀 루틴 중 중요한 것만 스택 깊이가 검사되기 때문에, 안전 마진(safety margin)이 필요하다. 기본 설정은 기본적으로 작고, 충돌 가능성이 낮은 2메가바이트이다(2MB). 그러나, 설정값이 너무 작으면 복합 함수의 실행이 어려울 수 있다. 슈퍼유저만 이 설정을 변경할 수 있다.

- 실제 커널 제한보다 max_stack_depth를 큰 값으로 설정하면 런어웨이 재귀 함수가 백엔드 프로세스와 충돌할 수 있다. AgensGraph가 커널 제한을 결정할 수 있는 플랫폼에서 서버는 이 변수가 불안정한 값으로 설정되는 것을 허용하지 않는다. 그러나 모든 플랫폼이 정보를 제공하지는 않으므로 값 선택 시 신중을 기해야 한다.

dynamic_shared_memory_type (enum)

- 서버가 사용해야 하는 동적 공유 메모리 구현을 지정한다. 가능한 값은 posix(shm_open을 사용하여 할당된 POSIX 공유 메모리의 경우) 및 sysv(shmget을 통해 할당된 System V 공유 메모리의 경우), windows(Windows 공유 메모리의 경우), mmap(데이터 디렉토리에 저장된 메모리 맵 파일을 사용하는 공유 메모리 시뮬레이션). none(이 기능 비활성)이다. 일부 플랫폼에서는 몇 개가 지원되지 않는다. 첫 번째 지원 옵션은 해당 플랫폼의 기본값이다. 플랫폼에서 기본값이 아닌 mmap 옵션의 사용은 일반적으로 권장하지 않는다. 이유는 운영체제가 수정된 페이지를 디스크에 반복해서 다시 쓰면서 시스템 I/O 로드가 늘어나기 때문이다. 그러나, pg_dynshmem 디렉토리를 RAM 디스크에 저장하거나 다른 공유 메모리 기능을 사용할 수 없는 경우에는 디버깅용으로 유용하다.

6.1.3 Write Ahead Log

wal_level (enum)

- wal_level은 WAL에 기록되는 정보의 양을 결정한다. 기본값은 충돌 또는 즉시 셧다운으로부터 복구하기 위해 필요한 정보만 기록하는 minimal이다. archive는 WAL 아카이브에 필요한 로깅만 추가한다. hot_standby는 스탠바이 서버에서 읽기 전용 쿼리에 필요한 정보를 좀 더 추가한다. logical은 논리적 디코딩을 지원하는데 필요한 정보를 추가한다. 각 레벨에는 모두 저수준에서 로깅된 정보가 포함된다. 이 매개변수는 서버 시작 시 설정된다.
- minimal 레벨에서, 일부 벌크 실행 WAL 로깅은 안전하게 건너뛸 수 있다. 그러면 실행이 빨라진다. 이러한 최적화를 적용할 수 있는 실행에는 다음이 포함된다.
  1. CREATE TABLE AS
  2. CREATE INDEX
  3. CLUSTER
  4. COPY 상기는 동일 트랜잭션에서 생성되었거나 또는 레코드가 지워진 테이블에 적용된다.
- 그러나 최소 WAL에는 베이스 백업 및 WAL 로그로부터 데이터를 재구성하는 데 필요한 정보가 충분하지 않으므로 WAL 아카이빙 (archive_mode) 및 스트리밍 복제를 하려면 archive 이상을 사용해야 한다.
- hot_standby 레벨에서 archive와 동일한 정보 및 WAL로부터 실행 트랜잭션의 상태 재구성에 필요한 정보가 로깅된다. 스탠바이 서버에서 읽기 전용 쿼리를 사용하려면, 운영 서버에서 wal_level을 hot_standby 이상으로 설정하고 hot_standby는 스탠바이 서버에서 활성화해야 한다. hot_standby와 archive 레벨 사용시 측정 가능한 성능 차이는 거의 없는 것으로 생각되므로 운영상 눈에 띄는 변화가 있을 경우 피드백을 주기 바란다.
- 논리적 수준에서 hot_standby를 사용하는 것과 동일한 정보 및 WAL로부터 논리적 변경 세트를 사용하는 데 필요한 정보가 로깅된다. 논리적 수준을 사용하면 WAL 볼륨이 증가한다. 특히 여러 개의 테이블을 REPLICA IDENTITY FULL로 환경 설정하고 UPDATE 및 DELETE 문을 여러 개 실행하는 경우 그렇다.

fsync (boolean)

- 이 매개변수가 on인 경우 AgensGraph 서버는 업데이트가 물리적으로 디스크에 기록되었는지를 fsync() 시스템 호출 또는 상응하는 다양한 메서드 (wal_sync_method 참조)를 사용하여 확인하려고 한다. 이로써 운영 체제 또는 하드웨어 충돌 후에 데이터베이스 클러스터를 일정한 상태로 복구할 수 있다.
- fsync를 해제하는 것은 성능상 장점이 있지만, 결과적으로는 정전 또는 시스템 장애의 경우에 데이터 손상이 복구 불가능할 수 있다. 따라서 외부 데이터로 전체 데이터베이스를 손쉽게 재생성할 수 있는 경우에만 fsync를 해제하는 것이 바람직하다.

- fsync를 해제를 고려할 수 있는 안전한 경우는 백업 파일로부터 새 데이터베이스 클러스터를 초기 로딩하는 경우, 재생성할 데이터베이스의 데이터를 일괄 처리하거나, 장애처리 (failover) 로만 사용되는 읽기 전용 데이터베이스를 자주 복제를 통해 재생성하는 경우가 있다. 고성능 하드웨어만을 위해 fsync를 해제하는 것은 바람직하지 않다.

- fsync를 해제했다가 다시 설정하는 경우 복구 신뢰도를 위해 커널의 모든 변경된 버퍼를 내구성이 좋은 저장소로 강제 이동하는 것이 필요하다. 이것은 클러스터가 셧다운 중이거나 fsync가 on일 때 initdb --sync-only를 실행하거나, sync를 실행하거나, 파일 시스템의 마운트를 해제하거나, 서버를 리부팅함으로써 가능하다.

- 중요하지 않은 트랜잭션에 대해 synchronous_commit를 해제하면 데이터 충돌 위험 없이 fsync를 해제함으로써 잠재적인 성능상 장점을 얻을 수 있다.

- fsync는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정할 수 있다. 이 매개변수를 해제할 경우 full_page_writes의 해제도 고려해야 한다.

synchronous_commit (enum)

- 명령이 ``success'' 표시를 클라이언트에 리턴하기 전에 WAL 레코드가 디스크에 기록될 때까지 트랜잭션 커밋이 기다릴지 여부를 지정한다. 유효 값은 on 및 remote_write, local, off이다. 기본값 및 안전 설정은 on이다. off인 경우 success표시가 클라이언트에 전달되는 시간과 서버 충돌 없이 트랜잭션이 정말로 안전하다는 것이 보장되는 시간 사이에 지연이 생길 수 있다. (최대 지연은 wal_writer_delay의 3배이다.) fsync와 달리, 이 매개변수를 off로 설정하면 데이터베이스 일관성 문제는 발생하지 않는다. 운영 체제 또는 데이터베이스 장애는 이른바 최근에 커밋된 트랜잭션이 일부 분실되는 결과가 발생하지만 데이터베이스 상태는 해당 트랜잭션이 깔끔하게 중단된 것과 같다. 따라서, 트랜잭션 영속성에 대해 정확한 확실성보다는 성능이 더 중요한 경우에 synchronous_commit를 해제하는 것이 유용한 대안일 수 있다.

- synchronous_standby_names가 설정되면, 트랜잭션의 WAL 레코드가 스탠바이 서버로 복제될 때까지 트랜잭션 커밋이 기다릴지의 여부를 이 매개변수로도 제어한다. on으로 설정되면, 스탠바이 서버로부터 트랜잭션의 커밋 레코드를 수신했고 디스크에 쓰기 되었다는 응답이 현재의 동기 스탠바이 서버로부터 올 때까지 커밋이 대기한다. 이것은 운영 서버 및 스탠바이 서버 양쪽에서 데이터베이스 저장소의 손상이 없는 경우에 트랜잭션이 분실되지 않았음을 보장한다. remote_write로 설정되면, 트랜잭션의 커밋 레코드를 수신했고 스탠바이 서버의 운영 체제에 쓰기 되었지만, 스탠바이 서버의 안정된 저장소에 데이터가 도착했는지는 확실하지 않다는 응답이 현재의 동기 스탠바이 서버로부터 올 때까지 커밋이 대기한다. 데이터 보존을 위해서는 AgensGraph의 스탠바이 서버 인스턴스에 장애가 발생한 경우에도 이 설정으로 충분하지만 스탠바이 서버가 운영 체제 수준에서 장애가 발생한 경우는 그렇지 않다.

- 동기 복제를 사용 중인 경우 일반적으로 로컬에서 디스크로 쓰기 되도록 기다리거나, WAL 레코드의 복제를 기다리거나, 트랜잭션이 비동기적으로 커밋되게 하는 것이 합리적이다. 그러나, local 설정은 로컬에서 디스크로 써지는 것은 기다리지만 동기 복제는 기다리지 않는 트랜잭션에 사용할 수 있다.

synchronous_standby_names를 설정하지 않으면 on 및 remote_write, local 설정 모두 동일한 동기화 레벨을 제공하며 트랜잭션 커밋은 로컬에서 디스크로 쓰기만을 기다린다.

- 이 매개변수는 언제든 변경할 수 있다. 모든 트랜잭션의 동작은 사실상 커밋되었을 때의 설정에 따라 결정된다. 따라서 일부 트랜잭션 커밋은 동기적으로, 그 외에는 비동기적으로 만드는 것이 가능하고 유용하다. 예를 들면, 기본값이 반대인 경우 구문이 여러 개인 트랜잭션 커밋 하나를 비동기적으로 만들려면 트랜잭션 내에서 SET LOCAL synchronous_commit TO OFF를 실행해야 한다.

wal_sync_method (enum)

- 디스크에 WAL을 강제로 업데이트할 때 사용하는 방법으로, fsync가 off인 경우는 WAL 파일을 강제로 업데이트하지 않기 때문에 이 설정은 무시된다. 가능한 값은 다음과 같다.

  1. open_datasync (open() 옵션 O_DSYNC를 사용하여 WAL 파일 쓰기)
  2. fdatasync (커밋마다 fdatasync() 호출)
  3. fsync (커밋마다 fsync() 호출)
  4. fsync_writethrough (커밋마다 fsync() 호출, 모든 디스크 쓰기 캐시에서 write-through 강제로 수행)
  5. open_sync (open() 옵션 O_SYNC를 사용하여 WAL 파일 쓰기)

- open_* 옵션도 필요 시 O_DIRECT를 사용한다. 이와 같은 선택이 항상 모든 플랫폼에서 가능한 것은 아니다. 기본값은 플랫폼에서 지원되는 위의 목록에서 첫 번째 방식이다. 단, Linux에서는 fdatasync가 기본값이다. 기본값이 반드시 이상적인 것은 아니다. 장애로부터 안전한 환경 설정을 만들거나 성능을 최적화하려면 값을 변경하거나 시스템 환경 설정의 다른 측면을 변경하는 것이 필요할 수도 있다. 이러한 측면은 13.1 절에서 다룬다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

full_page_writes (boolean)

- 이 매개변수가 on이면, AgensGraph 서버는 체크포인트 이후의 각 디스크 페이지를 처음 수정할 때, 해당 페이지의 전체 내용을 WAL에 기록한다. 이것은 운영 체제 장애 시 진행 중인 페이지 쓰기가 부분적으로만 완료되어 디스크 상의 페이지에 옛날 데이터와 새 데이터가 공존할 수 있기 때문에 필요하다. 일반적으로 WAL에 저장되는 행 수준(row-level) 변경 데이터는 충돌 후 복구 중에 그러한 페이지를 완전히 복구하는 데 충분하지 않다. 전체 페이지 이미지를 저장하면 페이지의 올바른 복구가 보장되지만 WAL에 기록해야 하는 데이터량의 증가를 감수해야 한다. (WAL 리플레이는 항상 체크포인트에서 시작되므로 체크포인트 이후의 페이지별 첫 번째 변경 중에만 해도 충분하다. 전체 페이지를 WAL에 쓰는 비용을 줄이는 한 가지 방법은 체크포인트 간격 매개변수를 늘리는 것이다.)

- 이 매개변수를 해제하면 정상적인 운영 속도가 빨라지지만 시스템 장애 발생 시 손상된 데이터가 복구 불가능하게 되거나 데이터 손상이 드러나지 않을 수 있다. 이러한 위험은 규모는 작지만 fsync을 해제했을 때와 유사하며, fsync에 권장되는 것과 환경이 동일할 때만 해제해야 한다.

- 이 매개변수를 해제하는 것은 point-in-time recovery(PITR) 용 WAL 아카이빙의 사용에는 영향을 미치지 않는다.
- 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 기본값은 on이다.

wal_log_hints (boolean)

- 이 매개변수가 on이면, AgensGraph 서버는 체크포인트 이후의 각 디스크 페이지를 처음 수정하는 도중에는 소위 힌트 비트(hint bits)의 중요하지 않은 수정도 포함하여 해당 페이지의 전체 내용을 WAL에 기록한다.
- 데이터 체크섬이 사용으로 설정되면 힌트 비트(hint bit) 업데이트가 항상 WAL 로깅되고 이 설정은 무시된다. 데이터베이스에서 데이터 체크섬이 사용으로 설정된 경우 이 설정을 사용하여 WAL 로깅이 추가로 얼마나 발생하는지 테스트할 수 있다.
- 이 매개변수는 서버 시작 시 설정된다. 기본값은 off이다.

wal_buffers (integer)

- WAL 데이터에 사용되고 아직 디스크에 기록되지 않은 공유 메모리의 합계. 기본 설정 -1은 shared_buffers의 1/32번째(약 3%)와 동일한 크기로써 64kB 이상, WAL 세그먼트 1개 크기 이하이고, 일반적으로 16MB이다. 자동 설정이 너무 크거나 작은 경우에 직접 선택할 수 있으며, 32kB 미만은 32kB로 처리된다. 이 매개변수는 서버 시작 시 설정 된다.
- WAL 버퍼의 내용은 모든 트랜잭션 커밋마다 디스크에 쓰기 되므로 극단적으로 큰 값은 별다른 장점이 없을 가능성이 높다. 그러나, 이 값을 최소한 몇 메가바이트로 설정하면 여러 클라이언트가 한꺼번에 커밋함으로써 busy한 서버의 쓰기 성능이 개선된다. 기본 설정 -1인 자동 튜닝은 대부분 합리적인 결과를 만든다.

wal_writer_delay (integer)

- WAL writer의 작업 간 지연을 정한다. 각 작업 시 writer는 WAL을 디스크에 기록한다. 그런 다음, wal_writer_delay 밀리초 동안 슬립한 다음, 반복한다. 기본값은 200밀리초이다(200ms). 다수의 시스템에서 효율적인 슬립 지연 설정은 10밀리초이다. wal_writer_delay를 10의 배수가 아닌 다른 값으로 설정하면 10의 배수로 값을 올림하여 설정한 것과 결과가 동일하다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

commit_delay (integer)

- commit_delay는 WAL 플러쉬(flush)를 초기화하기 전에 측정된 시간 지연을 마이크로초 단위로 추가한다. 이것은 시스템 로드가 충분히 커서 주어진 간격 내에 트랜잭션을 추가로 커밋할 준비가 된 경우 단일 WAL 플러쉬를 통해 대량의 트랜잭션이 커밋되게 함으로써 그룹 커밋 처리량을 개선할 수 있다. 그러나 이것은 WAL 플러쉬별로 대기 시간을 최대 commit_delay 마이크로초까지 늘리기도 한다. 커밋할 준비가 된 트랜잭션이 없을 경우 지연은 낭비되는 시간이므로 쓰기가 곧 시작되는 경우 최소한 commit_siblings만큼의

다른 트랜잭션이 수행될 때 지연이 수행된다. 또한 fsync가 비활성화되면 지연이 수행되지 않는다. 기본 commit_delay는 0이다(지연 없음). 수퍼유저만 이 설정을 변경할 수 있다.

commit_siblings (integer)

- commit_delay 지연을 수행하기 전에 필요한 동시 개방 트랜잭션의 최소 수. 값이 크면, 지연 간격 중에 커밋 준비가 된 다른 트랜잭션이 최소한 하나 이상일 확률이 높다. 기본값은 5개 트랜잭션이다.

6.1.4 Query Planning

Planner Method Configuration

이 환경 설정 매개변수는 쿼리 옵티마이저에 의해 선택된 쿼리 플랜에 영향을 주는 대략적인 방법을 제공한다. 특정 쿼리에 대한 옵티마이저에 의해 선택된 기본 플랜이 최적이 아닌 경우 임시 방편으로 이 환경 설정 매개변수 중 하나를 사용하여 옵티마이저가 다른 플랜을 선택하게 강제할 수 있다. 옵티마이저가 선택한 플랜의 수준을 개선하는 더 나은 방법은 플래너 비용 상수를 조절하고, ANALYZE를 수동으로 실행하고, default_statistics_target 환경 설정 매개변수 늘리고, ALTER TABLE SET STATISTICS를 사용하여 특정 칼럼에 대해 수집된 통계량을 늘리는 것이다.

enable_bitmapscan (boolean)

- 쿼리 플래너의 bitmap-scan plan types 사용을 활성화 또는 비활성화한다. 기본값은 on이다.

enable_hashagg (boolean)

- 쿼리 플래너의 hashed aggregation plan types 사용을 활성화 또는 비활성화한다. 기본값은 on이다.

enable_hashjoin (boolean)

- 쿼리 플래너의 hash-join plan types 사용을 활성화 또는 비활성화한다. 기본값은 on이다.

enable_indexscan (boolean)

- 쿼리 플래너의 index-scan plan types 사용을 활성화 또는 비활성화한다. 기본값은 on이다.

enable_indexonlyscan (boolean)

- 쿼리 플래너의 index-only-scan plan types 사용을 활성화 또는 비활성화한다. 기본값은 on이다.

enable_material (boolean)

- 쿼리 플래너의 materialization의 사용을 활성화 또는 비활성화한다. materialization을 완전히 억제하는 것은 어렵지만 이 변수를 해제하면 정확도가 요구되는 경우 외에는 플래너의 materialize 노드 삽입이 방지된다. 기본값은 on이다.

enable_mergejoin (boolean)

- 쿼리 플래너의 merge-join plan types 사용을 활성화 또는 비활성화한다. 기본값은 on이다.

enable_nestloop (boolean)

- 쿼리 플래너의 nested-loop join plans 사용을 활성화 또는 비활성화한다. nested-loop joins를 완전히 억제하는 것은 어렵지만 이 변수를 해제하면 사용 가능한 다른 방법이 있는 경우 플래너가 이것을 사용하는 것이 방지된다. 기본값은 on이다.

enable_seqscan (boolean)

- 쿼리 플래너의 sequential scan plan types 사용을 활성화 또는 비활성화한다. sequential scans를 완전히 억제하는 것은 어렵지만 이 변수를 해제하면 사용 가능한 다른 방법이 있는 경우 플래너가 이것을 사용하는 것이 방지된다. 기본값은 on이다.

enable_sort (boolean)

- 쿼리 플래너의 explicit sort steps 사용을 활성화 또는 비활성화한다. explicit sorts를 완전히 억제하는 것은 어렵지만 이 변수를 해제하면 사용 가능한 다른 방법이 있는 경우 플래너가 이것을 사용하는 것이 방지된다. 기본값은 on이다.

enable_tidscan (boolean)

- 쿼리 플래너의 TID scan plan types 사용을 활성화 또는 비활성화한다. 기본값은 on이다.

Planner Cost Constants

이 절에서 설명하는 cost 변수는 임의의 규모로 계산된다. 상대적인 값만 관련 있기 때문에, 동일한 계수로 상향 또는 하향되면 쿼리 플랜은 바뀌지 않는다. 기본적으로, 이러한 비용 변수는 순차적 페이지를 가져오는 비용을 근거로 한다. seq_page_cost는 일반적으로 1.0으로 설정되어 있으므로 다른 비용 변수는 이를 기준으로 설정된다. 그러나 사용자가 원한다면 특정 머신에서 실제 실행되는 밀리초 단위의 시간처럼 다른 비용을 사용할 수도 있다.

> 참고 : 아쉽게도 비용 변수에 대한 이상적인 값을 결정하는 데 적당한 방법은 없다. 특정한 설치가 수신하는 전체 쿼리 믹스에 대한 평균으로 처리하는 것이 최선이다. 몇 가지 경험에 비추어서, 이 값을 변경하는 것은 매우 위험할 수 있다.

seq_page_cost (floating point)

- 플래너가 예상하는 순차 가져오기 방법의 일부인 디스크 페이지 가져오기 비용을 설정한다. 기본값은 1.0이다. 이 값은 동일한 이름의 테이블스페이스 매개변수 설정이 특수한 테이블스페이스의 테이블과 인덱스를 오버라이드할 수 있다.

random_page_cost (floating point)

- 플래너가 예상한, 비순차적으로 가져온 디스크 페이지의 처리 비용을 설정한다. 기본값은 4.0이다. 이 값은 동일한 이름의 테이블스페이스 매개변수 설정에 의해 특수한 테이블스페이스의 테이블과 인덱스를 오버라이드할 수 있다.
- 이 값을 seq_page_cost에 비례하여 줄이면 시스템이 인덱스 스캔 쪽으로 치우치게 된다. 이 값을 늘리면 인덱스 스캔이 좀 더 비싸진다. 양쪽 값을 함께 늘리거나 줄여서 CPU 비용에 비례하여 디스크 I/O 비용의 중요도를 변경할 수 있다. 이것은 이후의 매개변수에서 설명된다.
- 디스크 저장소에 대한 랜덤 액세스는 일반적으로 순차 액세스보다 4배 이상 비싸다. 그러나 인덱싱된 읽기 같이 디스크에 대한 랜덤 액세스 대부분은 캐시에서 일어나므로 작은 기본값이 사용된다(4.0). 순차보다 랜덤 액세스가 40배정도 느린 것으로 보이지만, 실제 랜덤 읽기의 90%는 캐싱되는 것을 예상할 수 있다.
- 사용자의 작업 부하에서 90%의 캐시율이 잘못된 가정인 경우 random_page_cost를 늘려서 랜덤 저장소 읽기의 실제 비용이 반영되도록 할 수 있다. 그에 따라, 총 서버 메모리보다 데이터베이스가 작아서 데이터가 완전히 캐시되는 경우 random_page_cost를 줄이는 것이 적절할 수 있다. SSD 같이 랜덤 읽기 비용이 시퀀스에 비해 상대적으로 낮은 저장소는 더 낮은 random_page_cost 값으로 모델링이 더 잘 될 수도 있다.

  Tip: random_page_cost를 seq_page_cost 미만으로 설정하는 것이 시스템에서 허용되더라도 실제로는 그렇게 하는 것이 합리적이지 않다. 단, 데이터베이스 전체가 RAM에 캐싱되는 경우에는 시퀀스 밖 페이지를 건드리는 비용이 없으므로 동일하게 설정하는 것은 괜찮다. 또한 과도하게 캐시되는 데이터베이스에서 RAM에 이미 있는 페이지를 가져오는 비용이 일반적인 상태의 것보다 훨씬 적으므로 사용자는 CPU 매개변수에 비례하여 양쪽 값을 줄여야 한다.

cpu_tuple_cost (floating point)

- 플래너가 예상한 쿼리 도중 각 로우의 처리 비용을 설정한다. 기본값은 0.01이다.

cpu_index_tuple_cost (floating point)

- 플래너가 예상한 인덱스 스캔 도중 각 인덱스 항목의 처리 비용을 설정한다. 기본값은 0.005이다.

cpu_operator_cost (floating point)

- 플래너가 예상한, 쿼리 도중 실행된 각 연산자 또는 함수의 처리 비용을 설정한다. 기본값은 0.0025이다.

effective_cache_size (integer)

- 플래너가 추정할 단일 쿼리에 사용할 수 있는 디스크 캐시의 효율적인 크기를 설정한다. 이것은 인덱스를 사용하는 비용에 반영된다. 값이 클수록 인덱스 스캔이 사용될 가능성이 높다. 값이 작을수록 순차 스캔이 사용될 가능성이 높다. 이 매개변수를 설정하는 경우 AgensGraph의 공유 버퍼와, AgensGraph 데이터 파일에

사용되는 커널의 디스크 캐시 부분을 모두 고려해야 한다. 또한 사용 가능한 공간을 공유해야 하므로 서로 다른 테이블에 대해 예상되는 동시 쿼리 수도 고려해야 한다. 이 매개변수는 AgensGraph에 의해 할당된 공유 메모리 크기에는 효과가 없으며, 커널 디스크 캐시도 보존하지 않는다. 추정용으로만 사용된다. 또한 시스템은 디스크 캐시에 쿼리 간에 데이터가 남아 있을 것이라고 가정하지 않는다. 기본값은 4기가바이트이다(4GB).

6.2 File Locations

6.2.1 Config File setting

기본적으로 세 가지 구성 파일(`postgresql.conf`, `pg_hba.conf`, `pg_ident.conf`)은 데이터베이스 클러스터의 데이터 디렉토리에 저장되며 아래의 매개 변수를 사용하여 구성 파일을 다른 위치에 배치 할 수 있다. 구성파일을 분리하여 보관하면 관리가 쉬워지고 구성파일을 올바르게 백업하는 것이 더 쉽다.

- data_directory (string)
  데이터를 저장할 디렉토리를 지정한다. 이 매개 변수는 서버를 시작할 때만 설정할 수 있다.

- hba_file (string)
  호스트 기반 인증(`pg_hba.conf`)을 위한 구성 파일을 지정한다. 이 매개 변수는 서버를 시작할 때만 설정할 수 있다.

- ident_file (string)
  사용자 이름 매핑(`pg_ident.conf`)을 위한 구성 파일을 지정한다. 이 매개 변수는 서버를 시작할 때만 설정할 수 있다.

- external_pid_file (string)
  서버 관리 프로그램에서 사용하기 위해 서버가 작성해야하는 추가 프로세스 ID(PID) 파일의 이름을 지정한다. 이 매개 변수는 서버를 시작할 때만 설정할 수 있다.

기본 설치에서는 위의 매개 변수가 명시적으로 설정되지 않는다. 데이터 디렉토리는 `AGDATA` 환경 변수로 지정되며 구성 파일은 모두 데이터 디렉토리에서 찾을 수 있다.
구성 파일을 데이터 디레렉토리가 아닌 다른 곳에 보관하려면 `AGDATA` 환경 변수가 구성 파일을 포함하고 있어야 하며, `data_directory` 매개 변수는 데이터 디렉토리가 실제로 위치하도록 `postgresql.conf`를 설정해야 한다. `data_directory`는 데이터 디렉토리의 위치에 대해 데이터 디렉토리를 재정의하지만 구성 파일의 위치는 아니다. 원하는 경우 `config_file`, `hba_file`, `ident_file` 매개 변수를 사용하여 구성 파일 이름과 위치를 개별적으로 지정할 수 있으며 기본 구성 파일(`postgresql.conf`) 내에서 설정할 수 있다. 3개의 파라미터와 `data_directory`가 명시적으로 설정되어 있는 경우에는 `AGDATA`를 지정할 필요가 없다.
이러한 매개 변수를 설정할 때, 상대적 경로는 `Agens`가 시작되는 디렉토리와 관련된 것으로 해석된다.

6.2.2 Log File Setting

- log_destination (string)

  stderr, csvlog, syslog를 포함하여 서버 메시지를 기록하는 몇 가지 방법을 지원한다. 이 매개 변수를 쉼표로 구분하여 원하는 로그 대상 목록으로 설정한다. 기본값은 stderr에만 기록하는 것이다. 이 매개 변수는 `postgresql.conf` 파일 또는 서버 명령 행에서만 설정 할 수 있다.

  `log_destination`에 csvlog가 포함된 경우 로그 항목은 프로그램에 로드하는 데 편리한 ``CSV(comma separated values)'' 형식으로 출력된다.

- logging_collector (boolean)

  이 매개 변수는 *logging collector*를 사용하여 stderr로 전송된 로그 메시지를 캡처하여 로그 파일로 리디렉션하는 백그라운드 프로세스이다. 일부 유형의 메시지가 syslog출력에 나타나지 않기 때문에 이러한 접근 방식은 syslog에 로깅을 사용하는 것보다 더 유용하다. (일반적인 예는 동적 링크 오류 메시지이다. 다른 하나는 `archive_command`와 같은 스크립트에서 생성된 오류 메시지이다). 이 매개 변수는 서버를 시작할 때만 설정할 수 있다.

- log_directory (string)

  `logging_collector`를 사용하도록 설정한 경우 이 매개 변수는 로그 파일이 생성될 디렉토리를 결정한다. 이는 절대 경로로 지정하거나 클러스터 데이터 디렉터리와 비교할 수 있다. 이 매개 변수는 `postgresql.conf` 파일 또는 서버 명령줄에서만 설정할 수 있다. 기본설정은 `pg_log`이다.

- log_filename (string)

  `logging_collector`를 사용하도록 설정한 경우 이 매개 변수는 생성된 로그 파일의 파일 이름을 설정한다. 이 값은 `strftime`패턴으로 처리되므로 %-escapes 파일 이름을 지정하는 데 사용할 수 있다 (timezone에 의존적인 %-escapes가 있는 경우에는 `log_timezone`에서 지정한 영역 기준으로 연산이 수행된다). 지원되는 %-escapes은 Open Group의 `strftime` 사양에 나열된 것과 유사하다. 시스템의 `strfectime`이 직접적으로 사용되지 않으므로 플랫폼별 (비표준) 확장이 작동하지 않는다. 기본 값은 `postgresql-%Y-%m-%d_%H%M%S.log`이다.

  파일 이름을 지정하면 전체 Disk를 채우지 않도록 로그를 재사용한다.

  `log_destination`에서 CSV형식 출력을 활성화한 경우 `.csv`는 CSV형식 출력을 대한 파일 이름을 생성하기 위해서 타임 스탬프 로그 파일 이름으로 추가된다 (`log_filename`을 `.log`로 작성하면 `.log`로 생성된다).

  이 매개 변수는 `postgresql.conf` 파일 또는 서버 명령줄에서만 설정 할 수 있다.

- log_file_mode (integer)

  UNIX시스템에서 이 매개 변수는 `logging_collector`를 사용하도록 설정할 때 로그 파일에 대한 권한을 설정한다 (마이크로 소프트 윈도우즈에서는 이 매개 변수가 무시된다). 매개 변수 값은 `chmod`와 `umask`호출에

의해 승인된 형식 모드로 지정될 것으로 예상된다(통상적인 8진수 형식을 사용하려면 숫자가 0(zero)으로 시작해야 한다).

기본 사용 권한은 0600이며 서버 소유자만 로그 파일을 읽거나 쓸 수 있다. 다른 일반적으로 유용한 설정은 0640이며, 사용자 그룹이 파일을 읽을 수 있도록 허용한다. 이러한 설정을 사용하려면 클러스터 데이터 디렉토리 외부에 파일을 저장하기 위해 log_directory를 변경해야 한다. 어떠한 경우 민감한 데이터를 포함할 수 있기 때문에 로그 파일을 world-readable 파일로 만드는 것은 현명하지 못하다.

이 매개 변수는 postgresql.conf 파일 또는 서버 명령줄에서만 설정 할 수 있다.

- log_rotation_age (integer)

  logging_collector를 사용하도록 설정하면 이 매개 변수는 개별 로그 파일의 최대 수명을 결정한다. 이 시간이 경과하면 새 로그 파일이 생성된다. 새 로그 파일을 시간 기반으로 생성하지 않도록 설정하려면 0으로 설정한다. 이 매개 변수는 postgresql.conf파일 또는 서버 명령줄에서만 설정할 수 있다.

- log_rotation_size (integer)

  logging_collector를 사용하도록 설정하면 이 매개 변수는 개별 로그 파일의 최대 크기를 결정한다. 지정한 사이즈의 KB파일이 생성된 이후 새로운 로그 파일이 생성된다. 새 로그 파일을 사이즈 기반으로 생성하지 않도록 설정하려면 0으로 설정한다. 이 매개 변수는 postgresql.conf파일 또는 서버 명령줄에서만 설정할 수 있다.

- log_truncate_on_rotation (boolean)

  logging_collector를 사용하도록 설정하면 이 매개 변수는 AgensGraph를 사용하여 동일한 이름의 기존 로그 파일을 새로 추가하지 않고 덮어쓴다. 단, 서버 재기동 또는 size-based rotation이 아닌 time-based rotation에 의해 새로운 파일이 열리는 경우에만 truncate가 발생한다. 매개 변수가 off 일때, 기존에 있던 파일은 모든 경우에 추가된다. 예를 들어, log_filename을 postgresql-%H.log와 같은 설정으로 사용하면 24시간 로그 파일을 생성한 다음 주기적으로 덮어쓸 수 있다. 이 매개 변수는 postgresql.conf파일 또는 서버 명령줄에서만 설정할 수 있다.

  Example: 7일간의 로그를 유지하려면 server_log.Mon, server_log.Tue 등의 이름으로 하루에 하나의 로그 파일을 작성하고 지난주의 로그를 이번주 로그로 자동으로 덮어 쓰려면 log_filename을 server_log.%a, log_truncate_on_rotation을 on으로 설정하고 log_rotation_age를 1440로 설정한다.

  Example: 24시간 분량의 로그를 유지하려면 시간당 하나의 로그 파일이 필요하지만 파일 크기가 1GB를 초과할 경우 더 빠르게 순환될 수 있으므로 log_filename은 server_log.%H%M으로 log_truncate_on_rotation을 on, log_rotation_age를 60, log_rotation_size를 1000000으로 설정한다. log_filename에 %M을 포함하면 시간이 지정된 파일 이름과 다른 파일 이름을 선택하여 size-driven rotation을 허용할 수 있다.

- syslog_facility (enum)

  syslog에 로깅을 사용하도록 설정하면 이 매개 변수는 사용할 syslog ``facility'' 를 결정한다. LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7 중에 고를 수 있으며 기본값은 LOCAL0이다. 시스템

syslog 데몬의 설명서를 참조한다. 이 매개 변수는 postgresql.conf파일 또는 서버 명령줄에서만 설정할 수 있다.

- syslog_ident (string)

  syslog에 로깅을 사용하도록 설정하면 이 매개 변수는 syslog로그의 AgensGraph 메시지를 식별하는데 사용되는 프로그램 이름을 결정한다. 기본 값은 PostgreSQL이다. 이 매개 변수는 postgresql.conf파일 또는 서버 명령줄에서만 설정할 수 있다.

- syslog_sequence_numbers (boolean)

  syslog에 로깅되고 매개 변수가 on(기본값) 일 때, 각 메시지는 증가하는 시퀀스 번호 (예 : [2]) 로 접두사가 붙는다. 이 순환은 많은 syslog구현이 기본적으로 수행되는 ``---마지막 메시지 반복 횟수---'' 억제를 의미한다. 보다 최신의 syslog구현에서는 반복적인 메시지 억제를 구성할 수 있으므로 필요하지 않을 수 있다 (예 :rsyslog의 $RepeatedMsgReduction). 또한 실제로 반복되는 메시지를 차단하려는 경우에도 이 옵션을 해제할 수 있다.

  이 매개 변수는 postgresql.conf파일 또는 서버 명령줄에서만 설정할 수 있다.

- syslog_split_messages (boolean)

  syslog에 로깅을 사용하도록 설정하면 이 매개 변수는 메시지가 syslog로 전달되는 방식을 결정한다. on(기본값) 으로 설정시 메시지는 행별로 구분되고 긴 행은 기존 syslog구현의 일반적인 크기 제한인 1024 bytes에 맞춰 분할 된다. off로 설정시 AgensGraph 서버 로그 메시지는 syslog서비스로 전달되며, 잠재적으로 대용량 메시지에 대처하기 위해 syslog서비스에 연결된다.

  syslog가 텍스트 파일에 궁극적으로 로깅 되는 경우에는 대부분의 syslog구현이 큰 메시지를 처리할 수 없거나 특수하게 구성하도록 구성하므로 설정을 그대로 유지하는 것이 좋다. 그러나 syslog가 다른 매체에 궁극적으로 쓰이는 경우에는 메시지를 논리적으로 유지하는 것이 더 유용할 수 있다.

  이 매개 변수는 postgresql.conf파일 또는 서버 명령줄에서만 설정할 수 있다.

- event_source (string)

  이벤트 로그에 로깅을 사용하도록 설정하면 이 매개 변수는 로그에서 AgensGraph 메시지를 식별하는데 사용되는 프로그램 이름을 결정한다. 기본 값은 PostgreSQL이다. 이 매개 변수는 postgresql.conf파일 또는 서버 명령줄에서만 설정할 수 있다.

7 Tools

7.1 Client Tool

7.1.1 Agens

agens는 AgensGraph의 터미널 기반 프론트 엔드이다. 대화식으로 쿼리를 입력하고 이를 AgensGraph로부터 결과를 반환받는다. 파일이나 명령행 인수로 입력할 수도 있다. 또한 agens는 여러 가지 메타 명령 및 쉘과 유사한 기능을 제공하여 스크립트를 작성하고 다양한 작업을 자동화한다.

사용법은 아래와 같다.

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

Options

-a

--echo-all

모든 입력 라인을 판독된 대로 표준 출력으로 보여준다(대화형으로 판독되지 않는다). 이것은 ECHO 변수를 all로 설정하는 것과 같다.

-A

--no-align

정렬되지 않은 출력 모드로 전환한다(그렇지 않으면 기본 출력 모드가 정렬된다).

-b

--echo-errors

실패한 SQL 명령을 표준 오류 출력으로 보여준다. 이것은 ECHO 변수를 오류로 설정하는 것과 같다.

-c ***command***

--command=***command***

agens가 지정된 명령 문자열을 실행하도록 ***command***를 지정한다. 이 옵션은 반복될 수 있고 `-f` 옵션으로 명령을 조합하여 사용할 수 있다. 대신에 `-c` 또는 `-f`가 지정된 경우, agens는 표준 입력으로 명령을 읽지 않고 `-c` 및 `-f` 옵션을 모두 처리한 후 종료한다.

command는 서버에서 완전히 분석할 수 있는 명령 문자열이거나 단일 백슬래시 명령이 어어야 한다. 따라서 `-c` 옵션에서 SQL과 agens meta-commands를 혼용하여 사용할 수 없다. 이를 위해 반복된 `-c` 옵션을 사용하거나 문자열을 agens에 파이프('|')를 이용하여 사용할 수 있다. 사용 예는 아래와 같다.

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

또는

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

(\\ 는 분리 기호 메타 명령이다.)

-c에 전달된 각 SQL 명령 문자열은 단일 쿼리로 서버에 전송된다. 이 때문에 서버는 문자열에 여러 개의 트랜잭션으로 나눌 수 있는 명시적인 BEGIN/COMMIT 명령이 없으면 문자열에 여러 SQL 명령이 포함되어 있어도 단일 트랜잭션으로 이를 실행한다. 또한 agens는 마지막 SQL 명령의 결과만 문자열에 출력한다. 이것은 표준 입력이 개별적으로 전송되기 때문에 agens의 표준 입력으로 보내지거나 파일에서 동일한 문자열을 읽었을 때의 동작과 다르다.

이 동작으로 인해 단일 -c 문자열에 둘 이상의 명령을 배치하면 종종 예기치 않은 결과가 발생한다. -c 명령을 반복하여 사용하거나 위의 예제와 같이 echo를 사용, 또는 아래의 예제와 같이 here-document 쉘을 사용하여 agens의 표준 입력에 여러 명령을 입력하는 것이 더 좋다.

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

-d ***dbname***

--dbname=***dbname***

연결할 데이터베이스의 이름을 지정한다. 이것은 ***dbname***을 명령행에서 첫 번째 옵션이 아닌 인수로 지정하는 것과 같다.

이 매개 변수에 = 기호가 있거나 올바른 URI 접두어 (postgresql:// 또는 postgres://) 로 시작하는 경우 conninfo 문자열로 처리된다.

-e

--echo-queries

서버로 보낸 모든 SQL 명령을 표준 출력으로 복사한다. 이는 쿼리에 변수 ECHO를 설정하는 것과 동일하다.

-E

--echo-hidden

ECHO_HIDDEN 변수를 on으로 설정하는 것과 같다.

-f ***filename***

--file=***filename***

표준 입력보다는 ***filename***의 파일로부터 명령을 읽는다. 이 옵션은 -c 옵션을 사용하여 순서대로 반복하고 결합할 수 있다. -c 또는 -f를 지정할 때 agens는 표준 입력으로 명령을 읽지 않는다. 대신 모든 -c, -f 옵션을 순서대로 처리한 후 종료한다. 이를 제외하고 이 옵션은 메타 명령 \i와 대체로 동일하다.

filename이 -(hyphen) 이라면 표준 입력은 EOF 표시 또는 메타 명령인 \q까지 읽는다. 이를 통해 대화형 입력을 파일로부터의 입력과 상호 작용할 수 있다. 그러나 이 경우 Readline은 사용되지 않는다(-n이 지정된 것처럼).
이 옵션을 사용하는 것은 agens < ***filename***을 쓰는 것과 약간 다르다. 일반적으로 두 가지 모두 예상대로 작동하지만 -f를 사용하면 줄 번호가 있는 오류 메시지와 같은 유용한 기능을 사용할 수 있다. 또한 이 옵션을 사용하면 시작 오버 헤드가 줄어들 수도 있다. 반면 쉘의 입력 변경을 사용하는 변형은 (이론적으로) 모든 것을 수동으로 입력했을 때 받은 출력과 똑같은 출력을 보장한다.

-F ***separator***

--field-separator=***separator***

정렬되지 않은 출력의 필드 구분 기호로 ***separator***를 사용한다. 이것은 \pset fieldsep 또는 \f와 동일하다.

-h ***hostname***

--host=***hostname***

서버가 실행 중인 시스템의 호스트 이름을 지정한다. 값이 슬래시로 시작하면 Unix 도메인 소켓의 디렉터리로 사용된다.

-H

--html

HTML 표 형식 출력을 켠다. 이것은 \pset format html 또는 \H 명령과 동일하다.

-l

--list

사용 가능한 모든 데이터베이스를 나열하고 종료한다. 다른 non-connection 옵션은 무시되며 meta-command \list와 유사하다.

-L ***filename***

--log-file=***filename***

filename에 모든 쿼리 log를 작성한다.

-n

--no-readline

확장된 명령행 편집 기능을 사용하지 않는다.

-o ***filename***

--output=***filename***

filename 파일에 모든 질의 결과를 저장한다. \o 명령과 동일하다.

-p ***port***

--port=***port***

서버가 연결을 청취하는 TCP 포트 또는 로컬 Unix 도메인 소켓 파일 확장자를 지정한다. 기본값은 PGPORT 환경 변수 값이거나 설정하지 않았다면 컴파일 타임에 지정된 (기본값:5432) 를 사용한다.

-P ***assignment***

--pset=***assignment***

\pset 스타일로 출력 옵션을 지정한다. 여기서 이름과 값은 공백 대신 등호로 구분해야 한다. 예를 들어 출력 형식을 LaTeX로 설정하려면 -P format=latex로 사용한다.

-q

--quiet

agens가 조용하게 작업하도록 지정한다. 기본적으로 시작 메시지와 다양한 정보를 출력한다. 이 옵션을 사용하면 아무 일도 일어나지 않는다. 이것은 -c 옵션과 사용할 때 유용하다. 이것은 QUIET 변수를 on으로 설정하는 것과 같다.

-R ***separator***

--record-separator=***separator***

정렬되지 않은 출력의 레코드 구분 기호로 ***separator***를 사용한다. 이것은 \pset recordsep 명령과 동일하다.

-s

--single-step

단일 단계 모드로 실행한다. 즉, 각 명령이 서버로 전송되기 전에 사용자에게 메시지가 표시되고 실행 취소 옵션도 표시된다. 스크립트를 디버깅할 때 사용한다.

-S

--single-line

세미콜론처럼 개행 문자가 SQL 명령을 종료하는 단일 행 모드로 실행된다.

> Notice: 이 모드는 사용을 원하는 사람들을 위해 제공되는 것으로 반드시 사용하도록 권장하지는 않는다. 특히 SQL과 메타 명령을 한 줄에 섞어 놓으면 처음 접하는 사용자에게는 실행 순서가 명확하지 않을 수도 있다.

-t

--tuples-only

컬럼명과 결과 행 출력을 끈다. 이것은 \t 명령과 동일하다.

-T ***table_options***

--table-attr=***table_options***

HTML table 태그 내에 배치할 옵션을 지정한다. 자세한 내용은 \pset을 참조한다.

-U ***username***

--username=***username***

기본값 대신 ***username***으로 데이터베이스에 연결한다(권한이 있어야 한다).

-v ***assignment***

--set=***assignment***

--variable=***assignment***

\set 메타 명령과 같은 변수 지정을 수행한다. 이름과 값이 있는 경우 명령 줄의 등호로 구분해야 한다. 변수 설정을 해제하려면 등호를 그대로 두면 된다. 변수를 빈 값으로 설정하려면 등호를 사용하고 값을 그대로 둔다. 시작 초기 단계에서 할당되어 수행되나 내부 용으로 예약된 변수는 나중에 덮어쓸 수 있다.

-V

--version

agens 버전 정보를 보여주고 종료한다.

--revision

agens revision 정보를 보여주고 종료한다.

-w

--no-password

암호를 입력하지 않는다. 서버가 암호 인증을 요구하고 .pgpass파일과 같은 다른 방법으로 암호를 요구하는 경우는 연결 시도가 실패한다. 이 옵션은 암호를 입력할 사용자가 없는 배치 업무나 스크립트에서 유용하다.

이 옵션은 전체 세션에 대해 설정되어 있으므로 초기 연결 시도뿐만 아니라 메타 명령 \connect의 사용에도 영향을 준다.

-W

--password

데이터베이스에 연결하기 전에 비밀 번호를 입력하도록 agens를 강제 설정한다.

서버가 패스워드 인증을 요구하면, agens는 자동적으로 패스워드를 요구하기 때문에 이 옵션은 필수적인 것은 아니다. 그러나 agens는 서버가 암호를 원한다는 것을 알기 위해 연결 시도를 낭비 할 것이다. 경우에 따라 추가 연결 시도를 피하기 위해 -W 를 입력하는 것이 좋다.

이 옵션은 전체 세션에 대해 설정되어 있으므로 초기 연결 시도뿐만 아니라 메타 명령 \connect의 사용에도 영향을 준다.

-x

--expanded

확장된 표현식 지정 모드를 켠다. 이것은 \w 명령과 동일하다.

-X

--no-psqlrc

시작 파일 (시스템의 psqlrc 파일 또는 사용자의 .psqlrc 파일 모두) 을 읽지 않는다.

-z

--field-separator-zero

정렬되지 않은 출력의 필드 분리자를 0 바이트로 설정한다.

-0

--record-separator-zero

정렬되지 않은 출력에 대한 레코드 분리 기호를 0 바이트로 설정한다. 예를 들어 xargs -0과 같은 인터페이스에 유용하다.

-1

--single-transaction

이 옵션은 하나 이상의 -c 및 / 또는 -f 옵션과 함께 사용해야 한다.

이 옵션은 첫 번째로 BEGIN 명령을 발행하고, 마지막으로 COMMIT 명령을 발행하여 모든 명령을 단일 트랜잭션으로 래핑 합니다. 이렇게 하면 모든 명령이 성공적으로 완료되거나 변경 사항이 적용되지 않는다.
명령 자체에 BEGIN, COMMIT, ROLLBACK을 포함하면 이 옵션은 원하는 효과를 가질 수 없다. 또한 트랜잭션 블록 내부에서 개별 명령을 실행할 수 없는 경우 이 옵션을 지정하면 전체 트랜잭션이 실패한다.

-?

--help[=***options***]

agens에 대한 도움말을 표시하고 종료한다. 선택 항목인 ***options*** 매개 변수 (옵션 기본값) 는 설명된 agens의 부분을 선택한다. 명령은 agens의 백슬래시 명령을 설명하고 options는 agens에 전달할 수 있는 명령행 옵션을 설명한다. agens 구성 변수에 대한 도움말을 보여준다.

Exit Status

agens는 쉘이 정상적으로 완료되면 0을 반환하고, 치명적인 자체 오류 (예 : 메모리가 부족하고 파일을 찾을 수 없음) 가 발생하면 1을 반환한다. 서버에 연결이 잘못되어 세션이 대화 형이 아닌 경우 2를 반환하고, 스크립트에서 오류가 발생하고 변수 ON_ERROR_STOP가 설정되어 있다면 3을 반환한다.

Usage

Connecting to a Database

agens는 일반적인 AgensGraph 클라이언트 응용 프로그램이다. 데이터베이스에 연결하려면 대상 데이터베이스의 이름, 서버의 호스트 이름과 포트 번호, 연결할 사용자 이름을 알아야 한다. agens는 -d , -h , -p 및 -U 와 같은 명령행 옵션을 통해 매개 변수에 대해 각각 알릴 수 있다. 옵션에 속하지 않는 인수가 발견되면 데이터베이스 이름 (또는 데이터베이스 이름이 이미 지정된 경우 사용자 이름) 으로 해석된다. 이러한 모든 옵션이 필요하지는 않으며, 유용한 기본값이 있다. 호스트 명을 생략하면 agens는 Unix 도메인 소켓을 통해 로컬 호스트의 서버에 연결하거나 TCP/IP를 통해 Unix 도메인 소켓이 없는 시스템의 localhost에 연결한다. 기본 포트 번호는 컴파일 시 결정된다. 데이터베이스 서버는 동일한 기본값을 사용하므로 대부분의 경우 포트를 지정할 필요가 없다. 기본 사용자 이름은 운영 체제 사용자 이름이며 기본 데이터베이스 이름이다. 사용자 이름으로 모든 데이터베이스에 연결할 수는 없다. 데이터베이스 관리자는 액세스 권한을 사용자에게 알려야 한다.
기본값이 적절하지 않을 때 PGDATABASE, PGHOST, PGPORT 및 / 또는 PGUSER 환경 변수를 적절한 값으로 설정하여 사용한다. 암호를 정기적으로 입력하지 않으려면 ~/.pgpass 파일을 사용하는 것이 편리하다.

Entering SQL Commands

일반적으로 agens는 agens가 현재 연결되어 있는 데이터베이스의 이름과 슈퍼유저의 경우 ``=#'', 일반 유저의 경우 ``=#'' 라는 문자열의 이름을 제공한다. 예는 다음과 같다.

```
$ agens testdb
agens (AgensGraph 1.3.0, based on PostgreSQL 9.6.2)
Type "help" for help.


testdb=#
```

프롬프트에서 사용자는 SQL 명령을 입력할 수 있다. 일반적으로 명령 줄 끝의 세미콜론에 도달하면 입력 줄이 서버로 보내진다. 행의 끝은 명령을 종료하지 않는다. 따라서 명료하게 하기 위해 명령을 여러 행으로 분산시킬 수 있다. 명령을 보내고 오류 없이 실행하면 명령 결과가 화면에 표시된다.
명령이 실행될 때마다 agens는 LISTEN과 NOTIFY에 의해 생성된 비동기 알림 이벤트를 폴링 한다.
C 스타일 블록 주석은 처리 및 제거를 위해 서버로 전달되지만 SQL 표준 주석은 agens에 의해 제거된다.

Meta-Commands

agens에 입력한 따옴표로 묶지 않은 백슬래시로 시작하는 것은 agens 자체에서 처리하는 agens 메타 명령이다. 이 명령은 agens가 관리 또는 스크립팅에 보다 유용하도록 만든다. 메타 명령은 대개 슬래시 또는 백슬래시 명령이라고 한다.
agens 커멘드의 형식은 백슬래시 뒤에 명령 동사가 오고, 그 =다음에 인수 순으로 작성한다. 인수는 명령 동사와 서로 다른 여러 개의 공백 문자로 구분된다.
인수에 공백 문자를 포함하려면 작은따옴표를 사용한다. 인수에 작은따옴표를 포함 시키려면 작은따옴표 안에 두 개의 작은따옴표를 써야 한다. 단일 인용문에 포함된 것은 `\n`(새 라인), `\t`(탭), `\B`(백스페이스), `\r`(캐리지 리턴), `\f`(형식 피드), ***`\digits`***(8진수) 및 ***`\xdigits`***(16진수)에 적용된다. 작은따옴표로 묶인 텍스트 안에 있는 다른 문자의 앞에 있는 백슬래시는 그 문자가 무엇이든지 그 문자를 인용한다.
인수 내에서 역 따옴표 (`)로 묶인 텍스트는 쉘에 전달되는 명령행으로 간주된다. 명령의 출력(뒤따르는 줄 바꿈이 제거됨)은 역 따옴표 안의 텍스트를 대체한다.
따옴표가 없는 콜론(:) 뒤에 agens 변수 이름이 나타나는 경우 이는 SQL Interpolation에 설명된 대로 변수의 값으로 대체된다.
일부 명령은 인수로 SQL ID (예:테이블 이름)를 사용한다. 이 인수는 SQL 구문 규칙을 따른다. 따옴표로 묶지 않은 문자는 소문자로, 큰따옴표 (``)는 대/소문자 변환을 방지하고 식별자에 공백을 포함시킨다. 큰따옴표 안에는 쌍 따옴표가 하나의 큰따옴표로 축소된다. 예를 들어, FOO"BAR``BAZ는 fooBARbaz로 해석되고"A weird``" name``은 A weird" name이 된다.
인수에 대한 구문 분석은 줄 끝에서 또는 인용 부호가 없는 다른 백슬래시가 발견되면 중지한다. 인용 부호가 없는 백슬래시는 새로운 메타 명령의 시작으로 간주된다. 특수 시퀀스 \\ (두 개의 백 슬래시)는 인수의 끝을 표시하고 SQL 명령이 있는 경우 구문 분석을 계속한다. 그렇게 하면 SQL과 agens 명령을 한 줄에 자유롭게 섞을 수 있다. 그러나 어떤 경우이든, 메타 명령의 주장은 그 행의 끝을 넘어 계속될 수 없다.
다음과 같은 메타 데이터가 정의된다.

\a

현재 테이블 출력 형식이 정렬되지 않으면 정렬로 전환된다. 정렬되지 않은 경우 정렬되지 않음으로 설정된다.

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

AgensGraph 서버에 새로운 연결을 설정한다. 사용할 연결 매개 변수는 위치 구문을 사용하거나 ***conninfo*** 연결 문자열을 사용하여 지정할 수 있다.

명령이 데이터베이스 이름, 사용자, 호스트 또는 포트를 생략하는 경우, 새 연결은 이전 연결의 값을 재사용 할 수 있다. 기본적으로 이전 연결의 값은 ***conninfo*** 문자열을 처리할 때를 제외하고 재사용 된다. -reuse-previous=on 또는 -reuse-previous=off의 첫 번째 인수를 전달하면 기본값을 대체한다. 명령이 특정 매개 변수를 지정하거나 재사용하지 않으면 libpq 기본값이 사용된다. ***dbname***, ***username***, ***host***, ***port***의 임의 지정은 파라미터를 생략하는 것과 동일하다.

새 연결이 성공적으로 이루어지면 이전 연결이 닫힌다. 접속 시도가 실패한 경우 (잘못된 사용자 이름, 액세스 거부 등) agens가 대화식 모드인 경우 이전 연결만 유지된다. 비 대화식 스크립트를 실행할 때 오류가 발생하면 즉시 처리가 중지된다. 이 구분은 오타에 대한 사용자 편의 및 스크립트가 실수로 다른 데이터베이스에서 실수로 작동하지 않도록 하는 안전 메커니즘으로 선택되었다.

예제는 다음과 같다.

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

\C [***title***]

쿼리의 결과로 출력되는 임의의 테이블의 타이틀을 설정 또는 해제한다. 이 명령은 \pset title title과 동일하다 (이 명령의 이름은 이전에 HTML 표에서 캡션을 설정하는데 사용되었기 때문에 ``캡션''에서 파생되었다).

\cd [***directory***]

현재 작업 디렉터리를 ***directory***로 변경한다. 인수가 없으면 현재 사용자의 홈 디렉터리가 변경된다.

> Tip : 현재 작업 디렉터리는 \! pwd로 확인한다.

\conninfo

현재 데이터베이스 연결에 대한 정보를 보여준다.

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

프론트 엔드 (client) 복사를 수행한다. 이 작업은 SQL COPY 명령을 실행하는 작업이지만 서버가 지정한 파일을 읽거나 쓰는 대신 서버와 로컬 파일 시스템 간의 데이터를 읽거나 쓰기를 수행한다. 파일 액세스 및 권한은 서버가 아닌 로컬 사용자의 액세스 및 권한이며 SQL Superuser 권한은 필요하지 않다.

프로그램이 지정된 경우, 명령은 agens로 실행되며, 명령은 서버와 클라이언트 사이에서 전달되거나 명령어 간에 라우팅 된다. 다시 실행 권한은 서버가 아닌 로컬 사용자의 실행 권한이며 SQL 슈퍼유저 권한은 필요하지 않다.

\copy ... from stdin의 경우, 데이터 행은 명령을 실행한 동일한 소스에서 읽은 다음 \. 또는 EOF까지 읽는다. 이 옵션은 SQL 스크립트 파일에서 in-line 테이블을 채우는데 유용하다. \copy ... to stdout의 경우, 출력이 agens 명령 출력과 동일한 위치로 전송되고 COPY 카운트 명령 상태는 출력되지 않는다 (데이터 행과 혼동될 수 있기 때문). 현재의 명령 소스 또는 \o에 관계없이 agens의 표준 입력 또는 출력을 읽거나 쓰려면 pstdin 또는 pstdout을 쓴다.

이 명령의 구문은 SQL COPY 명령과 유사하다. 데이터 source/destination 이외의 모든 옵션은 COPY에 지정된 것과 같다. 이러한 이유로 특수 구문 분석 규칙은 \copy명령에 적용된다. 특히, agens의 변수 대체 규칙과 백슬래시 이스케이프는 적용되지 않는다.

> Tip : 이 작업은 모든 데이터가 클라이언트/서버 연결을 통과해야 하기 때문에 SQL COPY 명령만큼 효율적이지 않다. 대량의 데이터의 경우 SQL 명령이 더 좋을 수 있다.

\copyright

AgensGraph의 저작권 및 배포 조건을 표시한다.

\crosstabview [***colV*** [***colH*** [***colD*** [***sortcolH***]]]]

현재 쿼리 버퍼 (예 :\g) 를 실행하고 크로스 바 그리드에 결과를 표시한다. 쿼리는 최소한 세 개의 열을 반환해야 한다. ***colV***로 식별되는 출력 열은 수직 헤더가 되고 ***colH***로 식별되는 출력 열은 수평 헤더가 된다. ***colD***는 그리드 내에 표시할 출력 열을 식별한다. ***sortcolH***는 수평 헤더의 선택적 정렬 컬럼을 식별한다.

각 열 지정은 열 번호 (1부터 시작) 또는 열 이름일 수 있다. 일반적인 SQL 케이스 접기 및 인용 규칙이 열 이름에 적용된다. 생략된 경우, ***colV***는 컬럼 1로, ***colH***가 컬럼 2가 된다. ***colH***와 ***colV***는 달라야 한다. ***colD***가 지정되지 않은 경우 쿼리 결과에 정확히 세 개의 열이 있어야 하며 ***colV*** 또는 ***colH***도 ***colD***로 표시된다.

왼쪽 맨 위에 표시된 세로 방향 헤더는 조회 결과와 동일한 순서대로 칼럼에서 발견된 값을 포함하고 있으나 중복은 제거된다.

첫 번째 행으로 표시된 수평 머리글에는 칼럼이 중복 제거된 ***colH***열에 있는 값이 포함되어 있다. 기본적으로 이러한 순서는 조회 결과와 동일한 순서로 나타난다. 그러나 선택적 ***sortcolH*** 인수가 주어진 경우 값이 정수 번호여야 하며 ***colH*** 값은 해당 ***sortcolH*** 값에 따라 정렬된 수평 헤더에 값이 표시된다.

크로스 탭 그리드 내부에 각각 ***colH***의 x 값과 ***colV***의 y 값이 명확할 경우, 교차점 (x, y) 에 위치한 셀은 ***colH***의 값이 x이고 ***colV***의 값이 y인 쿼리 결과 행에 ***colD*** column의 값을 포함한다. 그러한 열이 없으면 빈셀을 반환하며 행이 여러 개 있으면 오류가 발생한다.

\d[S+] [***patterns***]

각 관계 (table, view, materialized view, index, sequence, foreign table) 또는 ***patterns*** 일치 유형의 경우, 모든 열에 해당하는 모든 열과 유형, 기본 값 (기본 값이 아닌 경우) 및 NULL이 표시된다. 연관된 인덱스, 제약 조건, 규칙 및 트리거도 표시된다. foreign table의 경우 연관된 foreign 서버도 표시된다.(``패턴 매칭'' 은 아래 패턴에 정의되어 있다.)

일부 관계 유형의 경우 \d는 각 열에 대해 sequences의 컬럼값, 인덱스에 대한 인덱스 표현식, foreign tables에 대한 foreign data wrapper 옵션 등의 추가 정보를 보여준다.

명령 양식 \d+는 더 많은 정보가 표시된다는 점을 제외하고는 동일하다. 테이블의 열과 관련된 모든 주석이 표시되며, 테이블에 있는 OID가 표시되고, 관계가 없는 경우에는 기본적인 복제본 ID 설정이 표시된다.
기본적으로 사용자 생성 개체만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다.

> Note: \d가 패턴 인수 없이 사용되는 경우 \dtvmsE와 동일하다. tables, views, materialized views, sequences, foreign tables의 목록이 표시된다.

\da[S] [***pattern***]
집계 함수를 리턴 유형 및 조작할 데이터 유형과 함께 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 집계만 보여준다. 기본적으로 사용자 생성 개체만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다.

\dA[+] [***pattern***]
액세스 방법을 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 액세스 방법만 보여준다. +가 명령 이름에 추가되면 각 테이블 영역은 디스크 크기, 사용 권한 및 설명과 관련된 옵션과 함께 나열된다.

\db[+] [***pattern***]
tablespace를 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 tablesapce만 보여준다. +가 명령 이름에 추가되면 각 테이블 영역은 디스크 크기, 사용 권한 및 설명과 관련된 옵션과 함께 나열된다.

\dc[S+] [***pattern***]
문자 집합 인코딩 간의 변환을 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 전환 패턴만 나열된다. 기본적으로 사용자 생성 개체만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다. +가 명령 이름에 추가되면 각각의 오브젝트는 그 연관된 설명과 함께 나열된다.

\dC[+] [***pattern***]
cast의 타입을 나열한다. ***pattern***이 지정된 경우 원본 또는 대상 유형의 패턴과 일치하는 cast만 나열된다. +가 명령 이름에 추가되면 각각의 오브젝트는 그 연관된 설명과 함께 나열된다.

\dd[S] [***pattern***]
유형 제약 조건, 연산자 클래스, 연산자 패밀리, 규칙 및 트리거 의 객체에 대한 설명이 나열된다. 다른 모든 주석은 해당 오브젝트 유형에 대한 각각의 백슬래시 명령으로 볼 수 있다.
\dd는 ***pattern***과 일치하는 오브젝트 또는 인수가 제공되지 않는 경우 적절한 유형의 가시 오브젝트에 대한 설명을 보여준다. 그러나 두 경우 모두 설명이 있는 개체만 나열된다. 기본적으로 사용자 생성 개체만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다.
오브젝트에 대한 설명은 COMMENT SQL 명령을 사용하여 작성할 수 있다.

\ddp [***pattern***]
기본 액세스 권한 설정을 나열한다. 기본 권한 설정이 기본 제공된 기본값에서 변경된 각 역할(적용 가능한 경우 스키마)에 대한 항목이 나열된다. ***patterns***이 지정된 경우 역할 이름 또는 스키마 이름이 패턴과 일치하는 항목만 나열된다.
ALTER DEFAULT의 권한 명령은 기본 액세스 권한을 설정하는데 사용된다. 권한 표시의 의미는 GRANT에 설명되어 있다.

\dD[S+] [***pattern***]

도메인을 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 도메인만 보여준다. 기본적으로 사용자 생성 개체만 표시됩니다. 기본적으로 사용자 생성 개체만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다. +가 명령 이름에 추가되면 각각의 오브젝트는 그 연관된 설명과 함께 나열된다.

\dE[S+] [***pattern***]

\di[S+] [***pattern***]

\dm[S+] [***pattern***]

\ds[S+] [***pattern***]

\dt[S+] [***pattern***]

\dv[S+] [***pattern***]

이 명령 그룹에서 문자 E, i, m, s, t 및 v는 각각 foreign table, index, materialized view, sequence, table, view를 나타낸다. 이러한 문자 중 일부 또는 전부를 임의의 순서로 지정하여 이러한 유형의 객체 목록을 얻을 수 있다. 예를 들어, \dit는 인덱스와 테이블을 나열한다. +가 명령 이름에 추가되면 각 오브젝트 영역은 디스크 크기, 사용 권한 및 설명과 관련된 옵션과 함께 나열된다. ***patterns***이 지정된 경우 패턴과 일치하는 이름의 오브젝트만 나열된다. 기본적으로 사용자 생성 개체만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다.

\des[+] [***pattern***]

외부 서버를 나열한다 (mnemonic: ``external servers''). ***pattern***이 지정된 경우 일치하는 이름의 서버만 나열된다. \des+ 형식을 사용하면 서버의 ACL, 유형, 버전, 옵션 및 설명을 포함하여 각 서버의 전체 설명이 표시된다.

\det[+] [***pattern***]

외부 테이블을 나열한다 (mnemonic: ``external tables''). ***pattern***이 지정된 경우 테이블 이름 또는 스키마 이름이 패턴과 일치하는 항목만 나열된다. \det+ 형식을 사용하면 일반 옵션과 외부 테이블 설명도 표시된다.

\deu[+] [***pattern***]

사용자 매핑을 나열한다 (mnemonic: ``external users''). ***pattern***이 지정된 경우 패턴과 일치하는 사용자 이름의 매핑만 나열한다. \deu+ 형식을 사용하면 각 매핑에 대한 추가 정보가 나열된다.

> Caution : \deu+는 원격 사용자의 사용자 이름과 암호도 표시할 수 있으므로 공개하지 않도록 주의해야 한다.

\dew[+] [***pattern***]

외부 데이터 래퍼를 나열한다 (mnemonic: ``external wrappers''). ***pattern***이 지정된 경우 패턴과 일치하는 외부 데이터 래퍼만 나열된다. \dew+ 형식을 사용하면 외부 데이터 래퍼의 ACL, 옵션 및 설명도 표시된다.

\df[antwS+] [***pattern***]

함수를 ``agg'' (집계), ``normal'' , ``trigger'' 또는 ``window'' 로 분류된 결과 데이터 유형, 인수 데이터 유형 및 함수 유형과 함께 나열한다. 특정 유형의 기능만 표시하려면 해당 문자 a, n, t 또는 w를 명령에 추가한다. ***pattern***이 지정된 경우 패턴과 이름이 일치하는 함수만 보여준다. 기본적으로 사용자 생성 개체만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다. \df+ 형식을 사용하면 변동성, 병렬 안전성, 소유자, 보안 분류, 액세스 권한, 언어, 소스 코드 및 설명을 포함하여 각 기능에 대한 추가 정보가 표시된다.

Tip: 인수를 취하거나 특정 데이터 형식의 값을 반환하는 함수를 찾으려면 호출기의 검색 기능을 사용하여 \df 출력을 스크롤 한다.

\dF[+] [***pattern***]

텍스트 검색 설정을 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 설정만 보여준다. \dF+ 형식을 사용하면 기본 텍스트 검색 구문 분석기와 각 구문별 토큰 유형에 대한 사전 목록을 포함한 각 설정의 전체 설명을 보여준다.

\dFd[+] [***pattern***]

텍스트 검색 사전을 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 사전만 보여준다. \dFd+ 형식을 사용하면 기본 텍스트 검색 템플릿과 옵션 값을 포함한 각각의 선택된 사전에 대한 추가 정보를 보여준다.

\dFp[+] [***pattern***]

텍스트 검색 구문 분석기를 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 파서만 보여준다. \dFp+ 형식을 사용하면 기본 기능 및 인식된 토큰 유형 목록을 포함한 각 파서의 전체 설명을 보여준다.

\dFt[+] [***pattern***]

텍스트 검색 템플릿을 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 템플릿만 보여준다. \dFt+ 형식을 사용하면 기본 기능 이름을 포함한 각각의 템플릿에 대한 추가 정보를 보여준다.

\dg[S+] [***pattern***]

데이터베이스 역할을 나열한다(``사용자''와 ``그룹''의 개념이 ``역할''로 통합되었으므로 이 명령은 이제 \du 와 같다). 기본적으로 사용자 생성 역할만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 역할만 나열된다. \dg+ 형식을 사용하면 각 역할에 대한 추가 정보를 보여준다. 현재 이것은 각 역할에 대한 설명을 추가한다.

\dG[+] [***pattern***]

그래프의 목록을 보여준다.

\dGe[+] [***pattern***]

그래프의 edge labels의 목록을 보여준다.

\dGl[+] [***pattern***]

그래프의 labels의 목록을 보여준다.

\dGv[+] [***pattern***]

그래프의 vertex labels의 목록을 보여준다.

\dGi[+] [***pattern***]

그래프의 property indexes의 목록을 보여준다.

\di[S+] [***pattern***]

indexes의 목록을 보여준다.

\dl

이것은 \lo_list에 대한 별명이며 큰 오브젝트의 목록을 보여준다.

\dL[S+] [***pattern***]

procedural 언어를 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 언어만 나열된다. 기본적으로 사용자 생성 언어만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다. +가 명령 이름에 추가되면 각 언어는 해당 호출 핸들러, 유효성 검사기, 접근 권한 및 시스템 개체 인지 여부와 함께 나열된다.

\dn[S+] [***pattern***]

스키마(namespaces)를 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 스키마만 보여준다. 기본적으로 사용자 생성 언어만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다. +가 명령 이름에 추가되면 각 객체는 연관된 권한 및 설명(있는 경우)과 함께 나열된다.

\do[S+] [***pattern***]

연산자를 피연산자 및 결과 유형과 함께 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 연산자만 보여준다. 기본적으로 사용자 생성 개체만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다. +를 명령 이름에 추가하면 각 연산자에 대한 추가 정보가 표시되며, 현재 기본 기능의 이름만 표시된다.

\dO[S+] [***pattern***]

데이터 정렬을 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 데이터 정렬만 나열된다. 기본적으로 사용자 생성 언어만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다. +를 명령 이름에 추가하면 각각의 조합은 그와 관련된 설명과 함께 나열된다. 현재 데이터베이스의 인코딩과 함께 사용할 수 있는 데이터 정렬만 표시되므로 동일한 설치의 다른 데이터베이스에서 결과가 다를 수 있다.

\dp [***pattern***]

테이블, 뷰 및 시퀀스와 연관된 액세스 권한을 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 테이블, 뷰, 시퀀스만 나열된다.

GRANT와 REVOKE 명령은 접근 권한을 설정하는데 사용된다. 권한 표시의 의미는 GRANT에 설명되어 있다.

\drds [***role-pattern*** [***database-pattern***]]

정의된 구성 설정을 나열한다. 이 설정은 role-specific, database-specific 또는 둘 모두 일 수 있다. ***role-pattern***과 ***database-pattern***은 각각 특정 역할 및 데이터베이스를 선택하는데 사용된다. 생략되거나 +가 지정된 경우 지정되지 않은 각각의 role-specific 또는 database-specific을 포함한 모든 설정이 나열된다.

ALTER ROLE과 ALTER DATABASE 명령은 역할 및 데이터베이스 별 구성 설정을 정의하는 데 사용된다.

\dT[S+] [***pattern***]

데이터 타입을 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 타입만 나열된다. +가 명령 이름에 추가되면 각 유형은 내부 이름과 크기, 즉 enum type과 연관된 사용 권한이 있는 값과 함께 나열된다. 기본적으로 사용자 생성 개체만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다.

\du[S+] [***pattern***]

데이터베이스 역할을 나열한다. (``사용자''와 ``그룹''의 개념이 ``역할''로 통합되었으므로 이 명령은 \dg와 같다). 기본적으로 사용자 생성 개체만 표시되며 시스템 개체를 포함하도록 패턴 또는 S 한정자를 제공한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 역할만 보여준다. \du+ 형식을 사용하면 각 역할의 추가 설명을 보여준다. 현재 이것은 각 역할에 대한 설명을 추가한다.

\dx[+] [***pattern***]

설치된 확장 기능을 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 확장만 나열된다. \dx+ 형식을 사용하면 각 일치하는 확장자에 속하는 모든 개체가 나열된다.

\dy[+] [***pattern***]

이벤트 트리거에 대해 나열한다. ***pattern***이 지정된 경우 패턴과 일치하는 이름의 이벤트 트리거만 나열된다. +를 명령 이름에 추가하면 각각의 오브젝트는 그 연관된 설명과 함께 나열된다.

\e or \edit [***filename***] [***line_number***]

filename이 지정되어 있으면 파일이 편집되고 편집기가 종료된 후 해당 콘텐츠가 조회 버퍼로 다시 복사된다. ***filename***이 지정되지 않은 경우 현재 쿼리 버퍼가 동일한 방식으로 편집된 임시 파일로 복사된다.

agens의 일반적인 규칙에 따라 새 쿼리 버퍼가 다시 구문 분석된다. 여기서 전체 버퍼는 단일 행으로 처리된다 (방법으로 스크립트를 만들 수 없으므로 \i를 사용한다). 즉 세미콜론으로 끝나는 쿼리는 즉시 실행된다. 그렇지 않으면 쿼리 버퍼에서 대기하게 된다. 세미콜론을 입력하거나 \g 또는 \r을 눌러 취소한다.

행 번호를 지정하는 경우, agens는 파일 또는 쿼리 버퍼의 지정된 행에 커서를 배치한다. 단 하나의 모든 자릿수 인수가 주어지면 agens는 파일 이름이 아니라 행 번호로 가정한다.

> Tip: 에디터 구성 및 사용자 정의 방법은 아래의 Environment를 참조한다.

\echo ***text*** [...]

표준 출력에 인수를 하나의 공백으로 분리 한 다음 개행 문자를 따라 출력한다. 이것은 스크립트 출력에 정보를 삽입하는데 유용할 수 있다. 예제는 다음과 같다.

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

첫 번째 인수가 인용되지 않은 -n 이면 후행 줄 바꿈이 작성되지 않는다.

> Tip: \o 명령을 사용하여 쿼리 출력을 다시 전송하는 경우 이 명령 대신 \qecho를 사용할 수 있다.

\ef [***function_description*** [***line_number***]]

이 명령은 CREATE OR REPLACE FUNCTION 명령의 형식으로 명명된 함수의 정의를 패치하고 편집한다. 편집은 \edit와 같은 방식으로 수행된다. 편집기가 종료된 후 업데이트된 명령은 쿼리 버퍼에서 대기한다. 세미콜론을 입력하거나 \g을 보내거나 \r을 눌러 취소한다.

대상 함수는 이름만으로 또는 이름과 인수(예:`foo(integer, text)`)로 지정할 수 있다. 동일한 이름의 함수가 두 개 이상 있는 경우 인수 유형을 지정해야 한다.

함수를 지정하지 않으면 편집을 위해 빈 CREATE FUNCTION 템플리트가 표시된다.

행 번호가 지정되고 있는 경우, agens는 함수 본체의 지정된 행에 커서를 배치한다. (함수 본문은 일반적으로 파일의 첫 번째 줄에서 시작하지 않는다.)

> Tip: 에디터 구성 및 사용자 정의 방법은 Environment 아래를 참조한다.

\encoding [***encoding***]

클라이언트 문자 세트 인코딩을 설정한다. 인수가 없으면 이 명령은 현재 인코딩을 표시한다.

\ev [***view_name*** [***line_number***]]

이 명령은 CREATE OR REPLACE VIEW 명령의 형식으로 명명된 뷰의 정의를 패치하고 편집한다. 편집은 \edit과 같은 방식으로 수행된다. 편집기가 종료된 후 업데이트된 명령은 쿼리 버퍼에서 대기한다. 세미콜론을 입력하거나 \g 또는 \r을 눌러 취소한다.

뷰를 지정하지 않으면 편집을 위해 빈 CREATE VIEW 템플릿이 표시된다.

행 번호가 지정되었을 경우, agens는 뷰 정의의 지정된 행에 커서를 놓는다.

\f [***string***]

정렬되지 않은 쿼리 출력에 대한 필드 구분 기호를 설정한다. 기본값은 세로 막대 (|) 이다. 출력 옵션을 설정하는 일반적인 방법은 \pset을 참조한다.

\g [***filename***]

\g [|***command***]

현재 쿼리 입력 버퍼를 서버로 보내고, 쿼리의 결과를 ***filename***에 저장하거나 파이프 (|) 의 shell command ***command***에 대한 출력을 수행한다. 쿼리가 실패하거나 데이터를 반환하지 않는 SQL 명령이 아닌 쿼리가 0 개 이상의 튜플을 성공적으로 반환하는 경우에만 파일 또는 명령이 기록된다.

\g는 본질적으로 세미콜론과 같다. 인수가 있는 \g는 \o 명령의 ``one-shot'' 대안이다.

\gexec 현재 쿼리 입력 버퍼를 서버로 보낸 다음 쿼리 출력의 각 행의 각 열을 실행할 SQL 문으로 처리한다. 예를 들어, my_table의 각 열에 인덱스를 만들려면 다음과 같이 수행한다.

```
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
```

생성된 쿼리는 행이 반환되는 순서대로 실행되고 둘 이상의 열이 있는 경우 각 행 내에서 왼쪽에서 오른쪽으로 실행된다. NULL 필드는 무시된다. 생성된 쿼리는 문자 그대로 처리를 하기 위해 서버로 보내지므로 agens 메타 명령이나 agens 변수 참조를 포함할 수 없다. 개별 쿼리가 실패하면 ON_ERROR_STOP 이 설정되어 있지 않으면 나머지 쿼리가 계속 실행된다. 각 쿼리의 실행은 ECHO 처리의 대상이 된다. (\gexec를 사용하는 경우 ECHO를 all 또는 queries로 설정하는 것이 좋다.) 쿼리 로깅, 단일 단계 모드, 타이밍 및 기타 쿼리 실행 기능은 생성된 각 쿼리에도 적용된다.

\gset [***prefix***]

서버 현재 쿼리 입력 버퍼로 보내고 쿼리의 결과를 agens 변수에 저장한다(참고 Variables). 실행될 쿼리는 정확히 하나의 행을 반환해야 한다. 행의 각 열은 열과 동일한 별도의 변수에 저장된다. 예제는 다음과 같다.

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

prefix를 지정하면 해당 문자열이 쿼리의 열 이름 앞에 추가되어 사용할 변수 이름을 만든다.

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

열 결과가 NULL이면 해당 변수는 설정되지 않은 상태로 설정된다.

쿼리가 실패하거나 하나의 행을 반환하지 않으면 변수가 변경되지 않는다.

\h or \help [***command***]

지정된 SQL 명령에 대한 구문 도움말을 제공한다. ***command***가 지정되지 않은 경우 agens는 도움말 사용할 수 있는 모든 명령을 나열한다. ***command***가 별표(*)이면 모든 SQL 명령에 대한 구문 도움말을 보여준다.

> Note: 입력을 단순화하기 위해 여러 단어로 구성된 명령을 따옴표로 묶지 않아도 된다. \help alter table과 같이 입력하는 것이 좋다.

\H or \html

HTML 쿼리 출력 형식을 켠다. HTML 형식이 이미 켜져 있으면 기본 정렬 텍스트 형식으로 다시 전환된다. 이 명령은 호환성과 편의성을 위해 사용되지만 다른 출력 옵션 설정에 대해서는 \pset을 참조한다.

\i or \include ***filename***

파일 ***filename***에서 입력을 읽고 키보드에서 입력 한 것처럼 실행한다.

filename이 -(hyphen)이라면 표준 입력은 EOF 표시 또는 메타 명령인 \q까지 읽는다. 이를 통해 대화형 입력을 파일로부터의 입력과 상호 작용할 수 있다. Readline 동작은 가장 바깥쪽 레벨에서 활성화된 경우에만 사용된다.

> Note: 화면에서 읽은 행을 보려면 변수 ECHO를 all로 설정해야 한다.

\ir or \include_relative ***filename***

\l[+] or \list[+] [***pattern***]

서버의 데이터베이스를 나열하고 이름, 소유자, 문자 세트 인코딩 및 액세스 권한을 표시한다. ***pattern***이 지정된 경우 패턴과 일치하는 데이터베이스만 표시된다. +가 명령 이름에 추가되면 데이터베이스 크기, 기본 테이블 스페이스 및 설명도 표시된다(크기 정보는 현재 사용자가 연결할 수 있는 데이터베이스에서만 사용할 수 있다).

\lo_export ***loid filename***

데이터베이스의 OID ***loid***로 large object를 읽고 ***filename***에 기록한다. 데이터베이스 서버는 서버의 파일 시스템과 동일하게 실행되는 사용자의 권한에 따라 작동하는 서버 함수 lo_export와는 미묘한 차이가 있다는 점을 유의해야한다.

> Tip: \lo_list를 사용하여 large object의 OID 찾는다.

\lo_import ***filename*** [***comment***]

파일을 AgensGraph large object로 저장 한다. 필요한 경우 지정된 주석을 사용한다. 예제는 다음과 같다.

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

응답은 large object가 미래에 새로 작성된 large object에 액세스하는 데 사용될 수 있는 오브젝트 ID 152801을 수신했음을 나타낸다. 가독성을 위해 사람이 읽을 수 있는 설명을 모든 개체와 항상 연관시키는 것이 좋다. OID와 주석은 \lo_list 명령으로 볼 수 있다.

이 명령은 lo_import 서버의 사용자 및 파일 시스템 대신 로컬 파일 시스템에서 로컬 사용자로 작동하므로 서버 측과 약간 다르다.

\lo_list

현재 데이터베이스에 저장되어 있는 모든 AgensGraph의 large object 목록과 제공된 모든 주석을 보여준다.

\lo_unlink ***loid***

데이터베이스에서 OID ***loid***로 large object를 삭제한다.

> Tip: \lo_list를 사용하여 large object의 OID를 찾는다.

\o or \out [***filename***]

\o or \out [|***command***]

향후 질의 결과를 ***filename*** 파일에 저장하거나 향후 결과를 쉘 명령어 ***command***에 저장한다. 인수가 지정되지 않으면 조회 출력이 표준 출력으로 재설정 된다.

``쿼리 결과''에는 데이터베이스 서버에서 가져온 모든 테이블, 명령 응답 및 알림뿐만 아니라 데이터베이스를 쿼리하는 다양한 백슬래시 명령 (예 :\d)의 출력이 포함되지만 오류 메시지는 포함되지 않는다.

> Tip: 쿼리 결과 사이에 텍스트 출력을 삽입하려면 \qecho를 사용한다.

\p or \print 현재 쿼리 버퍼를 표준 출력으로 출력한다.

\password [***username***]

지정한 사용자의 암호를 변경한다(기본적으로 현재 사용자). 이 명령은 새 암호를 묻는 메시지를 표시하고 암호화한 다음 ALTER ROLE 명령으로 서버에 보낸다. 변경된 새로운 암호는 명령 내역, 서버로그 또는 다른 어디에도 기록되지 않는다.

\prompt [***text***] ***name***

사용자에게 변수 ***name***에 할당된 텍스트를 입력하라는 메시지를 표시한다. 선택적 프롬프트 문자열, ***text***를 지정할 수 있다(다중 작업 메시지의 경우 텍스트를 작은 따옴표로 감싼다).

기본적으로 \prompt는 입출력을 위해 터미널을 사용한다. 그러나 -f 명령 줄 스위치를 사용하려면 \prompt는 표준 입력과 표준 출력을 사용한다.

\pset [***option*** [***value***]]

이 명령은 조회 결과 표의 출력에 영향을 주는 옵션을 설정한다. ***option***은 설정할 옵션을 나타낸다. ***value***의 의미는 선택한 옵션에 따라 다르다. 일부 옵션의 경우 ***value***를 생략하면 특정 옵션에 설명 된대로 옵션이 지정되거나 설정이 해제된다. 동작이 지정되지 않은 경우 ***value***를 생략하면 현재 설정이 표시된다.

인수가 없는 \pset은 모든 출력 옵션의 현재 상태를 표시한다.

조정 가능한 인쇄 옵션은 다음과 같다.

- border

  value는 숫자여야 한다. 일반적으로 숫자가 클수록 더 많은 테두리와 선을 갖지만 세부 사항은 특정 형식에 따라 다르다. HTML 형식에서는 border= 속성으로 직접 전환된다. 대부분의 형식에서는 0(경계 없음), 1(내부 분할 선), 2(표 프레임) 만 의미가 있으며 2 이상의 값은 border=2와 동일하게 처리된다. latex와 latex-longtable은 별도로 3의 값이 데이터 행 사이의 구분선에 추가할 수 있다.

- columns

  wrapped 형식의 대상 너비와 확장된 자동 모드에서 호출기 또는 세로 표시 장치로 전환할 수 있도록 출력이 충분한지를 결정하기 위한 너비 제한을 설정한다. Zero(기본값) 는 COLUMNS 환경 변수에 의해 목표 너비가 제어되도록 하거나 COLUMNS가 설정되지 않은 경우 감지된 화면 너비를 제어한다. 또한 columns이 0 일 때 wrapped 된 형식은 화면 출력에만 영향을 준다. columns가 0 이면 파일 및 파이프 출력이 해당 너비에도 래핑 된다.

- expanded (or x)

  value를 지정하면 확장모드를 활성화 또는 비활성화하는 on/off 또는 auto로 설정해야 한다. ***value***를 생략하면 on/off 설정으로 전환된다. 확장 모드를 사용하면 쿼리 결과가 두 개의 열로 표시된다. 열 이름은 왼쪽에 있고 데이터는 오른쪽에 있다. 이 모드는 데이터가 정상 ``수평'' 상태에서 화면에 맞지 않을 때 유용하다. auto 설정에서 확장된 모드는 쿼리 출력에 둘 이상의 열이 있고 화면보다 더 넓을 때마다 사용된다. 그렇지 않으면 일반 모드가 사용된다. 자동 설정은 정렬 및 래핑 된 형식에서만 유효하다. 다른 형식에서는 항상 확장 모드가 꺼져있는 것처럼 작동한다.

- fieldsep

  정렬되지 않은 출력 형식으로 사용될 필드 구분 기호를 지정한다. 이렇게 하면 다른 프로그램이 선호하는 탭이나 쉼표로 구분된 출력을 생성할 수 있다. 필드 구분 기호로 탭을 설정하려면 \pset fieldsep '\t'를 입력한다. 기본 필드 구분 기호는 '|'이다.

- fieldsep_zero

  정렬되지 않은 출력 형식으로 사용할 필드 구분 기호를 0 바이트로 설정한다.

- footer

  value가 지정된 경우 테이블 바닥글 (***n*** rows) 합계를 활성화 또는 비활성화할 수 있는 on/off를 설정해야 한다. 값이 생략된 경우 바닥글 표시를 켜거나 끈다.

- format

  unaligned, aligned, wrapped, html, asciidoc, latex(uses tabular), latex-longtable, troff-ms 중 하나로 출력 형식을 설정한다. 고유 한 약어가 허용된다 (한 글자만으로도 충분하다). unaligned는 한 행에 있는 모든 열을 현재 활성 필드 구분 기호로 구분하여 사용한다. 이 기능은 다른 프로그램 (예 : 탭으로 구분되거나 쉼표로 구분 된 형식) 에서 읽을 수 있는 출력을 만들 때 유용하다. aligned 형식은 사람이 읽을 수 있는 표준 형식의 텍스트 출력이다. 이것이 기본값이다. wrapped 형식은 aligned 된 것처럼 보이지만 대상 열 너비에 출력이 맞도록 여러 줄에 걸쳐 넓은 데이터 값을 래핑한다. 목표 너비는 columns 옵션에서 설명한 대로 결정된다. agens에서 컬럼명은 래핑하려고 시도하지 않는다. 따라서 열 머리글에 필요한 전체 너비가 대상을 초과하는 경우 wrapped 형식이 aligned 된 것과 동일하게 작동한다. html, asciidoc, latex, latex-longtable, troff-ms 형식의 포맷은 각각의 mark-up를 사용하여 문서에 포함될 수 있는 표를 작성한다. 그것들은 완전한 문서가 아니다. HTML에 필요하지 않을 수도 있지만, LaTeX에서는 완전한 문서 wrapper가 있어야 한다. Latex-longtable은 LaTeX longtable과 booktabs 패키지도 필요하다.

- linestyle

  테두리 선 그리기 스타일을 ascii, old-ascii 또는 unicode 중 하나로 설정한다. 고유한 약어가 허용된다 (한 글자만으로도 충분하다). 기본 설정은 ascii 이다. 이 옵션은 정렬 및 래핑된 출력 형식에만 영향을 준다. ascii 스타일은 일반 ASCII 문자를 사용한다. 데이터의 개행은 오른쪽 여백에 + 기호를 사용하여 표시된다. 래핑 된 형식이 개행 문자없이 한 줄에서 다른 줄로 데이터를 래핑 할 때 첫 번째 줄의 오른쪽 여백과 점선의 왼쪽 여백에 도트 (.) 가 표시된다. old-ascii 스타일은 현재보다 이전의 버전에서 사용된 형식화 스타일을 사용한 일반 ASCII 문자를 사용한다. 데이터의 개행 문자는 왼쪽 열 구분 기호 대신에 : 기호를 사용하여 표시된다. 데이터가 개행 문자없이 한 줄에서 다른 줄로 줄 바꿈 될 때 ; 기호는 왼쪽 열 구분 기호 대신 사용된다. 유니 코드 스타일은 유니 코드 상자 드로잉 문자를 사용한다. 데이터의 개행은 오른쪽 여백에 캐리지 리턴 기호를 사용하여 표시된다. 데이터가 개행 문자없이 한 줄에서 다른 줄로 줄 바꿈 될 때, 줄 바꿈 기호는 첫 줄의 오른쪽 여백과 다시 다음 줄의 왼쪽 여백에 표시된다. 테두리 설정이 0보다 크면 linestyle 옵션도 테두리 선을 그리는 문자를 결정한다. 일반 ASCII 문자는 어디에서나 사용할 수 있지만 유니 코드 문자는 문자를 인식하는 디스플레이에서 더 보기가 좋다.

- null

  null값 대신 인쇄할 문자열을 설정한다. 기본 값은 아무것도 인쇄하지 않는 것이며, 이는 빈 문자열로 오인될 수 있다. 예를 들어 `\pset null '(null)'`을 선호한다.

- numericlocale

value가 지정되면 이것은 반드시 on 또는 off 여야하며, 이것은 로케일 맞춤 숫자 표기 기능을 활성 또는 비활성화 한다. 이 ***value***가 생략되면 표준 형식과 로케일 맞춤 숫자 출력으로 전환한다.

- pager

  쿼리 및 agens 도움말 출력에 대한 pager 프로그램의 사용을 제어한다. 환경변수에 PAGER가 설정되어 있다면 출력은 지정된 프로그램 형식으로 출력된다. 그렇지 않으면 플랫폼 종속 기본값(more 등)이 사용된다. pager 옵션이 꺼져 있으면 pager 프로그램이 사용되지 않는다. pager 옵션이 켜져 있을 때 pager는 출력이 종료되고 화면에 맞지 않을 때 적절하게 사용된다. 호출기 옵션을 항상 설정하여 화면에 맞는지 여부에 상관없이 모든 터미널에 사용할 호출기를 사용할 수도 있다. ***value***없이 `\pset pager`를 사용하면 pager 사용을 켜고 끈다.

- pager_min_lines

  `pager_min_lines`가 페이지 높이 보다 큰 숫자로 설정 되면, 적어도 이 출력 라인이 표시되지 않는 한 pager 프로그램은 호출되지 않는다. 기본 설정은 0이다.

- recordsep

  정렬되지 않은 출력 형식으로 사용할 레코드(줄) 구분 기호를 지정한다. 기본값은 개행 문자이다.

- recordsep_zero

  정렬되지 않은 출력 형식으로 사용할 레코드 구분 기호를 0 바이트로 설정한다.

- tableattr (or T)

  HTML 형식은 테이블 태그 내에 배치할 속성을 지정한다. `cellpadding` 또는 `bgcolor`를 예로 들수 있다. 이미 `\pset` 테두리가 이미 처리되어 있으므로 여기에 테두리를 지정하지 않는 것이 좋다. 값이 지정되지 않으면 테이블 속성이 설정되지 않는다. `latex-longtable` 형식에서는 왼쪽 정렬 데이터 유형이 포함된 각 열의 비례 폭을 제어한다. 이 값은 예를 들어 '0.2 0.2 0.6' 등의 값으로 구분된 값 목록으로 지정된다. 지정되지 않은 출력 열은 마지막으로 지정된 값을 사용한다.

- title (or C)

  이후에 인쇄되는 표의 제목을 설정한다. 출력 설명 태그를 제공하는 데 사용할 수 있다. 값을 지정 하지 않으면 제목이 설정되지 않는다.

- tuples_only (or t)

  value가 지정되면 튜플 전용 모드를 활성화 또는 비활성화하는 on 또는 off 중 하나 여야 한다. ***value***가 생략되면 표준 형식과 튜플 전용 모드 출력으로 전환한다. 일반 출력에는 열 머리글, 제목 및 다양한 바닥글과 같은 추가 정보가 포함된다. 튜플 전용 모드에서는 실제 테이블 데이터만 표시된다.

- unicode_border_linestyle

  유니 코드 선 스타일의 테두리 그리기 스타일을 `single` 또는 `double` 중 하나로 설정한다.

- unicode_column_linestyle

유니 코드 선 스타일의 열 그리기 스타일을 `single` 또는 `double` 중 하나로 설정한다.

- unicode_header_linestyle

  유니 코드 선 스타일의 머리글 그리기 스타일을 `single` 또는 `double` 중 하나로 설정한다.

  Tip: `\pset`에 대한 다양한 바로 가기 명령이 있다. 참조 `\`, `\C`, `\H`, `\t`, `\T`, `\X`.

`\q` or `\quit`

agens를 종료한다. 스크립트 파일에서는 해당 스크립트의 실행만 종료된다.

`\qecho` ***text*** […]

이 명령은 `\echo`와 동일하지만 출력은 `\o`에 설정된대로 쿼리 출력 채널에 기록된다.

`\r` or `\reset`

쿼리 버퍼를 다시 설정(해제) 한다.

`\s` [***filename***]

agens의 명령 행 기록을 ***filename***으로 인쇄한다. ***filename***이 생략된 경우 기록은 표준 출력에 기록된다(해당하는 경우 호출기 사용). 이 명령은 agens가 Readline지원 없이 빌드 된 경우에는 사용할 수 없다.

`\set` [***name*** [***value*** […]]]

agens 변수 ***name***을 ***value*** 또는 둘 이상의 값이 주어진 경우 모두 지정된 값으로 설정한다. 인자가 하나만 주어지면 변수는 빈 값으로 설정된다. 변수 설정을 해제하려면 `\unset` 명령을 사용한다.

인자가 없는 `\set`은 현재 agens에 설정된 모든 이름과 값을 보여준다.

유효한 변수 이름은 문자, 숫자 및 밑줄을 포함할 수 있다. Variables를 참고한다. 변수 이름은 대소문자를 구분한다.

임의의 변수를 원하는 대로 설정할 수 있지만 agens는 여러 변수를 특수한 것으로 취급한다. Variables에서 설명한다.

Note: 이 명령은 SQL 명령 SET과 관련이 없다.

`\setenv` ***name*** [***value***]

환경 변수 ***name***을 ***value***로 설정하거나 값이 제공되지 않으면 환경 변수 설정 해제한다. 예제는 다음과 같다.

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

`\sf[+]` ***function_description***

이 명령은 CREATE OR REPLACE FUNCTION 명령의 형식으로 명명된 함수의 정의를 패치하고 표시한다. 정의는 `\o`에서 설정한 대로 현재 쿼리 출력 채널에 출력된다.

대상 함수는 이름만으로 또는 이름과 인수로 지정할 수 있다(예:`foo(integer, text)`). 동일한 이름의 함수가 두 개 이상 있는 경우 인수 유형을 지정해야 한다.

+를 명령 이름에 추가하면 출력 라인이 번호가 매겨지고 함수 본문의 첫 번째 줄이 첫 번째 줄이 된다.

\sv[+] ***view_name***

이 명령은 CREATE OR REPLACE VIEW 명령의 양식으로 명명된 뷰의 정의를 반입 및 표시한다. 정의는 \o에 설정된 대로 현재 쿼리 출력 채널에 출력된다.

+를 명령 이름에 추가하면 출력 라인이 1부터 번호가 매겨진다.

\t

출력 칼럼 이름 머리글과 행 카운트 하단의 디스플레이를 전환한다. 이 명령은 \pset tuples_only에 해당하며 편의상 제공된다.

\T ***table_options***

테이블 태그 내에 있는 속성을 HTML 출력 형식으로 지정한다. 이 명령은 \pset tableattr ***table_options***에 해당한다.

\timing [***on*** | ***off***]

매개 변수가 없으면 각 SQL 문이 걸리는 시간을 밀리초 단위로 표시한다. 매개 변수를 사용하여 동일하게 설정한다.

\unset ***name***

agens 변수 ***name***을 해제 (삭제) 한다.

\w or \write ***filename***

\w or \write | ***command***

현재 쿼리 버퍼를 ***filename*** 파일로 출력하거나 쉘 명령 ***command***으로 파이프를 출력한다.

\watch [***seconds***]

쿼리가 중단되거나 쿼리가 실패할 때까지 현재 쿼리 버퍼 (\g와 같은) 를 반복적으로 실행한다. 실행 사이에 지정된 시간 (기본 값 2) 을 기다린다. 각 쿼리 결과는 \pset 제목 문자열 (있는 경우), 쿼리 시작 시간 및 지연 간격을 포함하는 헤더로 표시된다.

\x [***on*** | ***off*** | ***auto***]

확장된 테이블 형식 지정 모드를 설정하거나 전환한다. 따라서 \pset expanded와 동일하다.

\z [***pattern***]

테이블, 뷰 및 시퀀스와 연관된 액세스 권한을 나열한다. 패턴이 지정된 경우 패턴과 이름이 일치하는 테이블, 뷰 및 시퀀스만 나열된다.

이것은 \dp(``표시 권한'') 의 별명이다.

\! [***command***]

별개의 쉘로 전환되거나 쉘 명령 ***command***를 수행한다. 인수는 더 이상 해석되지 않으며 쉘은 현재 상태 그대로 본다. 특히, 가변 대체 규칙과 백슬래시 이탈은 적용되지 않는다.

\? [***topic***]

도움말 정보를 표시한다. 선택 항목인 ***topic*** 매개 변수 (기본값은 commands) 는 설명할 agens의 부분을 선택한다. 명령은 agens의 백슬래시 commands을 설명한다. options는 agens에 전달할 수 있는 명령행 옵션을 설명한다. 그리고variables은 agens의 구성 변수에 대한 도움말을 보여준다.

Patterns

다양한 \d 명령은 ***pattern*** 매개 변수를 사용하여 표시할 개체 이름을 지정한다. 가장 단순한 경우 패턴은 객체의 정확한 이름이다. 패턴 내의 문자는 일반적으로 SQL 이름과 마찬가지로 소문자로 변환된다. 예를 들어, \dt FOO는 foo라는 테이블을 표시한다. SQL 이름에서처럼 패턴 주위에 큰따옴표를 배치하면 소문자로 전환되지 않는다. 패턴에 실제 큰따옴표 문자를 포함해야 하는 경우 큰따옴표 안에 쌍 따옴표로 작성한다. 이것은 SQL 인용 식별자에 대한 규칙과 일치한다. 예를 들어 \dt "FOO""BAR"는 FOO"BAR로 명명된 테이블을 표시한다(foo"bar가 아닌). SQL 이름에 대한 일반 규칙과 달리 패턴의 일부분만 큰따옴표를 넣을 수 있다. 예를 들어 \dt FOO"FOO"BAR는 fooFOObar로 명명된 테이블을 보여준다.

pattern 매개 변수가 완전히 생략 될 때마다 \d 명령은 현재 스키마 검색 경로에 표시되는 모든 객체를 표시한다. 이는 *를 패턴으로 사용하는 것과 동일하다. (개체가 검색 경로에 포함되어 있고 동일한 종류의 개체가 검색 경로 이전에 나타나지 않는 경우 개체가 검색 경로에 표시된다. 이는 객체가 명시적인 스키마 자격 없이 이름을 기준으로 참조할 수 있다는 문구와 동일하다.) 가시성에 관계 없이 데이터베이스의 모든 개체를 보려면 *.*를 사용한다.

패턴 내에서 *는 임의의 문자 시퀀스(문자 없음 포함)이고, ? 모든 단일 문자와 일치한다(이 표기법은 Unix 쉘 파일 이름 패턴과 유사하다). 예를 들어, \dt int*는 이름이 int로 시작하는 모든 테이블을 표시한다. 그러나 큰따옴표 안에 *와 ?는 특별한 의미를 잃게되고 문자 그대로 일치되는 객체를 보여준다.

점(.)을 포함하는 패턴은 스키마 이름 패턴과 오브젝트 이름 패턴으로 해석된다. 예를 들어 \dt foo*.*bar*는 스키마 이름이 foo로 시작하는 테이블 이름에 bar를 포함하는 모든 테이블을 표시한다. 점이 나타나지 않으면 패턴은 현재 스키마 검색 경로에 표시되는 객체만 일치한다. 큰 따옴표 안에 있는 점은 특별한 의미를 잃어 버리고 문자 그대로 일치되는 객체를 보여준다.

고급 사용자는 문자 클래스와 같은 정규식 표기법을 사용할 수 있다(예:[0-9]). 모든 정규식 특수 문자는 링크에서 명시한 대로 작동한다. 이는 위에서 언급한 바와 같이 구분 기호로 번역된다. 즉, *는 regular-expression 표기법으로 번역된다. 이는 위에서 언급한 바와 같이 구분 기호로 번역된다. 즉, *는 regular-expression 표기법으로 번역된다. *, ?는 .로, $는 문자 그대로 번역된다. .은 ?, R*은 (R+|), R?은 (R|)로 작성하여 필요에 따라 이러한 패턴 문자를 모방 할 수 있다. 정규식의 일반적인 해석과는 달리 패턴이 전체 이름과 일치하지 않으므로 $는 정규식과 일치하지 않는다(즉, $는 패턴에 자동으로 추가된다). 패턴을 고정시키지 않으려면 시작 및/또는 종료 시 *를 쓴다. 큰 따옴표 안에는 모든 정규 표현식 특수 문자가 특별한 의미를 잃어 버리고 문자 그대로 일치한다는 점에 유의한다. 또한 정규식 특수 문자는 연산자 이름 패턴에서 문자 그대로 일치한다(예:\do의 인수).

Advanced Features

Variables

agens는 일반적인 유닉스 명령 쉘과 유사한 변수 대체 기능을 제공한다. 변수는 단순히 이름/값 쌍이며 값은 임의 길이의 문자열일 수 있다. 이름은 문자(라틴 문자가 아닌 문자 포함), 숫자 및 밑줄로 구성되어야 한다.

변수를 설정하려면 agens 메타 명령 \set을 사용한다. 예제는 다음과 같다.

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

변수 foo 값을 bar로 설정한다. 변수의 내용을 검색하려면 이름 앞에 콜론(:)을 입력한다. 예를 들면 다음과 같다.

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

이것은 일반 SQL 명령과 메타 명령 모두에서 작동한다. 아래의 SQL Interpolation에 자세한 내용이 있다.
두번째 인수없이 \set를 호출하면 변수는 빈 문자열이 값으로 설정된다. 변수를 설정 해제(삭제)하려면 \unset 명령을 사용한다. 모든 변수의 값을 표시하려면 인수없이 \set을 호출한다.
이러한 변수의 수는 agens에 의해 특별히 처리된다. 변수의 값을 변경하여 런타임에 변경할 수 있는 특정 옵션 설정을 나타내거나 경우에 따라 agens의 변경 가능한 상태를 나타낸다. 이러한 변수들을 다른 용도로 사용할 수는 있지만, 프로그램 동작이 정말로 빨리 진행될 수 있기 때문에 이것은 권장되지 않는다. 관습에 따라 모든 특수 처리 변수의 이름은 모두 대문자 ASCII 문자 (가능하면 숫자와 밑줄)로 구성된다. 차후에 최대한의 호환성을 보장하려면 사용자 자신의 용도로 이러한 변수 이름을 사용하지 않는다. 모든 특수 처리 된 변수의 목록이 다음과 같다.

AUTOCOMMIT

on(기본 값) 일 때, 각 SQL명령은 성공적으로 완료되면 자동으로 커밋 된다. 이 모드에서 commit을 연기하면 BEGIN 또는 START TRANSACTION SQL 명령을 입력해야 한다. off 또는 unset으로 설정하면 SQL명령은 COMMIT 또는 END를 명시적으로 호출할 때까지 커밋 되지 않는다.

> Note: In autocommit-off 모드에서는 ABORT 또는 ROLLBACK을 입력하여 실패한 트랜잭션을 명시 적으로 포기해야한다. 또한 커밋하지 않고 세션을 종료하면 작업이 손실된다.

> Note: autocommit-on 모드는 AgensGraph의 일반적인 동작이지만 autocommit-off는 SQL 사양에 더 가깝다. autocommit-off를 원할 경우 시스템 전반의 psqlrc 파일 또는 ~/.psqlrc 파일에서 설정할 수 있다.

COMP_KEYWORD_CASE

SQL 키워드를 완성 할 때 사용할 문자를 결정한다. lower 또는 upper로 설정 하면 완료된 단어는 각각 대문자 또는 소문자로 표시된다. preserve-lower 또는 preserve-upper(기본 값)로 설정되어 있다면, 완료된 단어는 단어가 이미 입력된 경우에 완료된 단어가 입력된다. 입력한 내용은 각각 소문자 또는 대문자로 표시된다.

DBNAME

현재 연결되어있는 데이터베이스의 이름이다. 이 설정은 데이터베이스에 연결할 때마다(프로그램 시작을 포함하여) 설정되지만 설정을 해제 할 수 있다.

ECHO

all로 설정하면 비어 있지 않은 모든 입력 행이 읽혀지면서 표준 출력에 인쇄된다(대화식으로 읽은 행에는 적용되지 않는다). 프로그램 시작시 이 동작을 선택하려면 스위치 -a를 사용한다. queries로 설정된 경우, agens는 서버로 전송될 때마다 각 쿼리를 표준 출력에 출력한다. 이 스위치는 -e이다. error로 설정된 경우 오류가 발생한 쿼리만 표준 오류 출력에 표시된다. 이 스위치는 -b이다. unset인 경우 또는 none으로 설정된 경우(또는 위의 값 이외의 값) 쿼리가 표시되지 않는다.

ECHO_HIDDEN

이 변수가 on으로 설정되고 백슬래시 명령이 데이터베이스를 조회하면 조회가 먼저 표시된다. 이 기능을 사용하면 AgensGraph 내부를 연구 하고 자신의 프로그램에서 유사한 기능을 제공 할 수 있다(프로그램 시작시 이 동작을 선택하려면 스위치 -E를 사용한다). 변수를 noexec값으로 설정하면 쿼리는 표시되지만 실제로 서버에 전송되어 실행되지는 않는다.

ENCODING

현재 클라이언트 문자 세트 인코딩이다.

FETCH_COUNT

이 변수가 정수 값> 0으로 설정되면, 표시되기 전에 전체 결과 세트를 수집하는 디폴트 동작보다는 SELECT 쿼리의 결과가 패치되어 많은 행의 그룹으로 표시된다. 따라서 결과 집합의 크기에 관계없이 제한된 양의 메모리만 사용된다. 이 기능을 활성화 할 때 일반적으로 100에서 1000까지의 설정이 사용된다. 이 기능을 사용할 때 일부 행이 이미 표시된 후 쿼리가 실패 할 수 있다.

> Tip: 이 기능으로 모든 출력 형식을 사용할 수 있지만 기본 정렬 형식은 각 그룹의 FETCH_COUNT 행이 별도로 형식화되어 행 그룹 전체에서 다양한 열 너비로 이어지기 때문에 좋지 않은 경향이 있다. 다른 출력 형식이 더 잘 작동한다.

HISTCONTROL

이 변수가 ignorespace로 설정된 경우, 공백으로 시작하는 행은 히스토리 목록에 입력되지 않는다. ignoredups 값으로 설정하면 이전 히스토리 행과 일치하는 행은 입력되지 않는다. ignoreboth 값은 두 옵션을 결합한다. 설정되지 않았거나 none(또는 위의 값 이외의 값)으로 설정된 경우 대화식 모드에서 읽은 모든 행이 내역 목록에 저장된다.

> Note: 이 기능은 bash에서 가져온 기능이다.

HISTFILE

기록 목록을 저장하는데 사용할 파일 이름이다. 기본값은 ~/.psql_history 이다. 입력의 예이다.

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

~/.psqlrc에서 agens는 각 데이터베이스에 대해 별도의 기록을 유지한다.

> Note: 이 기능은 bash에서 가져온 기능이다.

HISTSIZE

명령 히스토리에 저장할 명령의 수이다. 기본값은 500이다.

> Note: 이 기능은 bash에서 가져온 기능이다.

HOST

현재 연결되어있는 데이터베이스 서버 호스트이다. 이 설정은 데이터베이스에 연결할 때마다(프로그램 시작을 포함하여) 설정되지만 설정을 해제 할 수 있다.

IGNOREEOF

unset의 경우, EOF 문자(일반적으로 **Control + D**)를 agens의 대화형 세션에 보내면 응용 프로그램이 종료된다. 숫자 값으로 설정 하면 응용 프로그램이 종료되기 전에 많은 EOF 문자가 무시된다. 변수가 설정되었지만 숫자 값이 없으면 기본값은 10 이다.

> Note: 이 기능은 bash에서 가져온 기능이다.

LASTOID

INSERT 또는 `\lo_import` 명령에서 리턴된 최종 영향을 받은 OID 값이다. 이 변수는 다음 SQL 명령의 결과가 표시될 때까지만 유효하다.

ON_ERROR_ROLLBACK

`on`으로 설정된 경우 트랜잭션 블록의 명령문이 오류를 생성하면 오류가 무시되고 트랜잭션이 계속된다. 대화형 세션으로 설정할 때 이러한 오류는 스크립트 파일을 읽을 때만 대화형 세션에서만 무시된다. unset 또는 `off`로 설정하면 오류를 생성하는 트랜잭션 블록의 명령문이 전체 트랜잭션을 중단합니다. 오류 롤백 모드는 트랜잭션 블록에 있는 각 명령 바로 앞에 내재 된 `SAVEPOINT`를 발행 한 다음 명령이 실패 할 경우 저장시점으로 롤백하여 작동한다.

ON_ERROR_STOP

기본적으로 명령 처리는 오류 후 계속된다. 이 변수를 `on`으로 설정하면 즉시 처리가 중지된다. 대화형 모드에서 agens는 명령 프롬프트로 돌아간다. 그렇지 않으면 agens가 종료되고 오류 코드 3을 반환하며 오류 코드 1을 사용하여 보고된 치명적인 오류 조건과 이 사례를 구분한다. 두 경우 모두 현재 실행중인 모든 스크립트(최상위 스크립트(있는 경우)및 기타 스크립트가 호출될 수 있는 기타 스크립트)는 즉시 종료된다. 최상위 명령 문자열에 여러 SQL 명령이 포함되어 있으면 처리가 현재 명령으로 중지된다.

PORT

현재 연결되어 있는 데이터베이스 서버 포트이다. 이 설정은 데이터베이스에 연결할 때마다(프로그램 시작을 포함하여) 설정되지만 설정을 해제 할 수 있다.

PROMPT1

PROMPT2

PROMPT3

agens 결과를 prompt에 어떻게 보여줄지를 지정한다. 아래 Prompting을 참조한다.

QUIET

이 변수를 on으로 설정 하면 명령 행 옵션 -q와 같다. 이것은 대화형 모드에서는 그다지 유용하지 않다.

SHOW_CONTEXT

이 변수는 CONTEXT 필드가 서버의 메시지에 표시되는지 여부를 제어하기 위해 never, errors, always라는 값으로 설정할 수 있다. 기본값은 errors이다 (오류 메시지에 표시되지만 경고 또는 경고 메시지가 표시되지 않음). VERBOSITY가 terse로 설정되면 아무런 영향을 주지 않는다 (방금받은 오류에 대한 자세한 버전을 원할 때 사용하려면 \errverbose도 참조한다).

SINGLELINE

이 변수를 on으로 설정하면 명령 행 옵션 -S와 동일하다.

SINGLESTEP

이 변수를 on으로 설정하면 명령 행 옵션 -s와 동일하다.

USER

현재 연결되어 있는 데이터 베이스 사용자이다. 이 설정은 데이터베이스에 연결할 때마다 (프로그램 시작을 포함하여) 설정되지만 설정을 해제 할 수 있다.

VERBOSITY

이 변수는 오류 보고서의 자세한 정보를 제어하기 위해 default, verbose , terse 값으로 설정할 수 있다. 방금받은 오류에 대한 자세한 버전을 원할 때 사용하려면 \errverbose를 참조한다.

SQL Interpolation

agens 변수의 핵심 기능은 메타 명령의 인수뿐만 아니라 일반 SQL 문으로 대체 (``interpolate'') 할 수 있다는 것이다. 게다가, agens는 SQL 리터럴과 식별자로 사용된 변수 값이 적절히 인용되도록 하는 기능을 제공한다. 어떠한 인용없이 값을 보간하기 위한 구문은 콜론 변수 이름 앞에 추가한다 (:). 예를 들면 다음과 같다.

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

my_table 테이블을 질의한다. 이는 안전하지 않을 수 있다. 변수의 값은 글자 그대로 복사되므로 불균형한 따옴표 또는 백슬래시 명령을 포함 할 수 있다. 이것을 어디에 두었는지 반드시 확인해야 한다.
값이 SQL 리터럴 또는 식별자로 사용될 때 따옴표로 묶는 것이 가장 안전하다. 변수의 값을 SQL 리터럴로 인용하려면 콜론 뒤에 작은 따옴표로 변수 이름을 쓴다. 값을 SQL 식별자로 인용하려면 콜론 뒤에 변수 이름을 큰 따옴표로 작성한다. 이러한 구조는 따옴표 및 변수 값 내에 포함 된 다른 특수 문자를 올바르게 처리한다. 이전 예제를 더 안전하게 작성한 예제는 다음과 같다.

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

가변 보간은 인용된 SQL 리터럴 및 식별자 내에서 수행되지 않는다. 따라서 ':foo'와 같은 구조는 변수의 값에서 인용된 리터럴을 생성하지 못한다 (값에 포함된 따옴표를 올바르게 처리하지 못하기 때문에 안전하지 않을 수

있다).
이 메커니즘을 사용하는 한 가지 예로는 파일 내용을 테이블 열로 복사하는 것이다. 먼저 파일을 변수에 로드 한 다음 변수의 값을 인용 문자열로 보간한다.

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

(my_file.txt에 NUL 바이트가 포함되어 있는 경우에도 이 기능은 작동하지 않는다. agens는 변수 값에 내장 된 NUL 바이트를 지원하지 않는다.)
콜론은 SQL 명령에 합법적으로 나타날 수 있기 때문에 명명된 변수가 현재 설정되어 있지 않으면 보간 (즉, :name, :`name' 또는 :``name'')에 대한 명백한 시도가 대체되지 않는다. 어쨌든 콜론을 백슬래시로 이스케이프하여 대체 코드에서 보호 할 수 있다.
변수의 콜론 구문은 ECPG와 같은 내장된 쿼리 언어에 대한 표준 SQL 이다. 배열 슬라이스와 형식 캐스트에 대한 콜론 구문은 AgensGraph 확장이며 표준 사용법과 충돌 할 수 있다. 변수의 값을 SQL 리터럴 또는 식별자로 이스케이프하는 콜론 쿼트 구문은 agens의 확장이다.

Prompting

agens는 프롬프트를 원하는대로 사용자가 정의 할 수 있다. 세 변수 PROMPT1, PROMPT2, PROMPT3에는 프롬프트의 모양을 설명하는 문자열과 특수 이스케이프 시퀀스가 있다. PROMPT1은 agens가 새 명령을 요청할 때 발행되는 일반 프롬프트이다. 예를 들어 명령이 세미콜론 또는 따옴표로 끝나지 않아 명령 입력 중에 더 많은 입력이 필요할 때 PROMPT2가 발행된다. PROMPT3은 SQL COPY FROM STDIN 명령을 실행 중이고 행 값을 터미널에 입력해야 할 때 발행된다.
선택한 프롬프트 변수의 값은 백분율 기호(%)가 있는 경우를 제외하고는 그대로 출력된다. 다음 문자에 따라 특정 텍스트가 대체된다. 정의된 대체 항목은 다음과 같다.
%M
UNIX 도메인 소켓을 통한 연결인 경우 데이터베이스 서버의 전체 호스트 이름(도메인 이름 포함) 또는 [local]을 유닉스 도메인 소켓이 기본 위치에 있지 않은 경우 [local: ***/dir/name***]로 설정된다.
%m
UNIX 도메인 소켓을 통한 연결인 경우 첫번째 점에서 잘린 데이터베이스 서버의 호스트 이름 또는 [local]로 설정된다.
%>
데이터베이스 서버가 수신하는 포트 번호로 설정된다.
%n
데이터베이스 세션 사용자 이름으로 설정된다(이 값의 확장은 SET SESSION AUTHORIZATION 명령의 결과로 데이터베이스 세션 동안 변경 될 수 있다).
%/

현재 데이터베이스 이름으로 설정된다.

%~

%/와 같다. 그러나 데이터베이스가 기본 데이터베이스라면 출력은 ~ (물결)로 설정된다.

%#

세션 사용자가 데이터베이스 superuser인 경우 #, 그렇지 않으면 >로 설정된다(이 값의 확장은 SET SESSION AUTHORIZATION 명령의 결과로 데이터베이스 세션 동안 변경 될 수 있다).

%p

현재 연결된 백엔드의 프로세스 ID로 설정된다.

%R

prompt 1에서는 보통 = 이지만 단선 모드에서는 ^를 세션이 데이터베이스와 연결이 끊어진 경우(\connect가 실패 할 때 발생할 수 있음) !를 사용한다.

prompt 2에서 %R은 agens의 명령이 종료되지 않아 추가 입력을 필요로 할 때 다음의 경우에 따라 다른 문자로 대체된다(단순히 명령이 아직 끝나지 않은 경우 -, /* ... */ 주석이 끝나지 않은 경우 *, 인용된 문자열이 끝나지 않은 경우 ', 인용된 식별자가 끝나지 않은 경우 ", 달러 기호가 끝나지 않은 경우 $, 매치되지 않은 왼쪽 괄호가 있는 경우 (). prompt 3에서 %R은 아무것도 생성하지 않는다.

%l

1부터 시작하는 현재 문장 내 라인 번호로 설정된다.

%[… %]

프롬프트에는 예를 들어 프롬프트 텍스트의 색상, 배경 또는 스타일을 변경하거나 터미널 창의 제목을 변경하는 터미널 제어 문자가 포함될 수 있다. Readline의 행 편집 기능이 제대로 작동하려면 이러한 인쇄되지 않는 제어 문자를 %[와 %]로 묶어서 보이지 않는 것으로 지정해야 한다. 여러 옵션들의 쌍이 프롬프트 내에서 발생할 수 있다. 예제는 다음과 같다.

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

VT100 호환 가능, color-capable 터미널에서 굵은 글꼴(1;), yellow-on-black(33;40) 프롬프트로 나타난다.
프롬프트에 백분율 기호를 삽입하려면 %%를 사용한다. 기본 프롬프트는 프롬프트 1과 2의 경우 '%/%R%# ' 이고 프롬프트 3의 경우는 '>>' 이다.

> Note: 이 기능은 tcsh에서 가져온 기능이다.

Command-Line Editing

agens는 편리한 라인 편집과 검색을 위해 Readline 라이브러리를 지원한다. agens를 종료할 때 명령 히스토리는 자동 저장되고, agens를 시작할 때 다시 로드 된다. 완료 로직이 SQL구문 분석을 요청하지 않아도 탭 완성 기능이 지원된다. 탭 완성에 의해 생성 된 쿼리는 다른 SQL 명령 (예 : SET TRANSACTION ISOLATION LEVEL)과 충돌 할 수도 있다. 탭 완성이 마음에 들지 않는다면 홈 디렉터리의 .inputrc 파일에 넣어 해제 할 수 있다.

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

(이것은 agens가 아니라 Readline 기능이다. 자세한 내용은 해당 설명서를 참조한다.)

Environment

COLUMNS

`\pset columns`가 0일 경우, 광범위한 출력이 필요한지 여부를 결정하기 위해 `wrapped` 형식과 너비의 폭을 조절하거나 확장된 자동 모드에서 수직 형식으로 전환해야 한다.

PAGER

조회 결과가 화면에 맞지 않으면 이 명령을 통해 파이프된다. 일반적인 값은 `more` 또는 `less`이다. 기본값은 플랫폼에 따라 다르다. `PAGER`를 공백으로 설정 또는 `\pset` 명령의 호출기 관련 옵션을 사용하여 비활성화 할 수 있다.

PGDATABASE

PGHOST

PGPORT

PGUSER

기본 연결 매개 변수 (참조).

PSQL_EDITOR

EDITOR

VISUAL

`\e`, `\ef`, `\ev` 명령에 사용되는 편집기이다. 이러한 변수는 나열된 순서대로 검사되며, 처음 설정된 순서로 사용된다. 내장된 기본 편집기는 유닉스 시스템에서는 `vi`이고 windows 시스템에서는 `notepad.exe`이다.

PSQL_EDITOR_LINENUMBER_ARG

`\e`, `\ef`, `\ev`를 행 번호 인수와 함께 사용할 때, 이 변수는 사용자의 편집기로 시작 줄 번호를 전달하는데 사용되는 명령 줄 인수를 지정한다. Emacs 또는 vi와 같은 편집기의 경우 이것은 더하기 기호이다. 옵션 이름과 행 번호 사이에 공백이 있어야 할 경우 변수 값에 후행 공백을 포함한다. 예제는 다음과 같다.

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

Unix 시스템에서는 기본값이 + 이다 (기본 편집기 `vi`에 해당하며 다른 많은 공통 편집기에 유용하다). 그러나 Windows 시스템에는 기본값이 없다.

PSQL_HISTORY

명령 실행 기록 파일의 대체 위치로 물결표 (~) 확장이 수행된다.

PSQLRC

사용자의 .psqlrc 파일의 대체 위치로 물결표(~) 확장이 수행된다.

SHELL

\!으로 실행된 명령

TMPDIR

임시파일을 저장하기 위한 디렉터리로 기본값은 /tmp이다. 이 유틸리티는 libpq에서 지원하는 환경변수를 사용한다.

Files

psqlrc과 ~/.psqlrc

-X 옵션을 지정하지 않는 한 데이터베이스에 연결한 후 정상 명령을 수락하기 전에 agens는 시스템 전체 시작 파일(psqlrc)과 사용자의 개인 시작 파일(~/.psqlrc)에서 명령어를 읽고 실행한다. 이 파일들은 일반적으로 \set과 SET 명령을 사용하여 클라이언트 및/또는 서버를 설정하기 위해 사용될 수 있다.

시스템 전체의 시작 파일은 psqlrc로 명명 되며 설치의 ``시스템 구성'' 디렉터리에서 찾을 수 있다. 이 디렉터리는 pg_config --sysconfdir을 실행하여 가장 안정적으로 식별된다. 기본적으로 이 디렉터리는 AgensGraph 실행 파일을 포함하는 디렉터리에 대한 ../etc/ 이다. 이 디렉터리의 이름은 PGSYSCONFDIR 환경변수를 통해 명시적으로 설정할 수 있다.

사용자의 개인 시작 파일 이름은 .psqlrc이며 호출하는 사용자의 홈 디렉터리에서 찾는다. 그러한 개념이 없는 Windows에서 개인 시작 파일의 이름은 %APPDATA%\postgresql\psqlrc.conf이다. 사용자 시작 파일의 위치는 PSQLRC 환경변수를 통해 명시적으로 설정할 수 있다.

.psql_history

명령 줄 기록은 ~/.psql_history 또는 Windows의 %APPDATA%\postgresql\psql_history 파일에 저장된다.

히스토리 파일의 위치는 PSQL_HISTORY 환경변수를 통해 명시적으로 설정할 수 있다.

Notes

agens는 동일하거나 이전 버전의 주요 버전의 서버에서 가장 잘 작동한다. 백슬래시 명령은 서버가 agens 자체보다 새로운 버전인 경우 특히 실패할 가능성이 있다. SQL 명령을 실행하고 쿼리 결과를 표시하는 일반적인 기능은 새로운 주요 버전의 서버에서도 작동해야 하지만 모든 경우에서 이를 보장 할 수는 없다.

agens를 사용하려는 경우 agens 프로그램을 다른 주요 버전의 여러 서버에 연결하는데, 최신 버전을 사용하는 것이 좋다. 또는 각 주요버전의 agens 사본을 보관하고 해당 서버와 일치하는 버전을 사용해야한다.

Notes for Windows Users

agens는 ``console application''으로 빌드된다. Windows 콘솔 창은 나머지 시스템과 다른 인코딩을 사용하기 때문에 agens에서 8비트 문자를 사용할 때는 특히 주의해야 한다. agens가 문제가 있는 콘솔 코드 페이지를 검색하면 시작시 경고 메시지가 표시된다. 콘솔 코드 페이지를 변경하려면 다음 두 가지가 필요하다.

- `cmd.exe /c chcp 1252` 코드 페이지를 입력하여 설정한다(1252는 독일어에 적합한 코드 페이지로, 적절한 코드 페이지를 입력한다). Cygwin을 사용하는 경우, 이 명령을 /etc/profile에 넣을 수 있다.
- 래스터 글꼴은 ANSI 코드 페이지에서 작동하지 않으므로 콘솔 글꼴을 `Lucida Console`로 설정한다.

Examples

첫 번째 예제는 여러 줄의 입력에 명령을 분산시키는 방법을 보여준다.

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

테이블의 정의를 확인한다.

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

프롬프트를 아래와 같이 변경할 수 있다.

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

테이블에 데이터를 채우고 테이블을 조회한다.

```
tester@[local] agens=#  SELECT * FROM my_table;
 first | second
-------+--------
```

```
     1 | one
     2 | two
     3 | three
     4 | four
(4 rows)
```

\pset 명령으로 테이블을 다른 방법으로 표시할 수 있다.

```
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
```

또는 짧은 명령을 사용할 수 있다.

```
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
```

적합할 경우, \crosstabview 명령을 사용하여 쿼리 결과를 크로스 탭 표현으로 표시 할 수 있다.

```
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)
```

이 두번째 예제는 숫자 순서가 역순으로 정렬된 행과 오름차순 숫자 순서로 독립된 컬럼이 있는 곱셈 테이블을 보여준다.

```
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)
```

7.1.2 pg_dump

pg_dump는 AgensGraph 데이터베이스를 백업하는 유틸리티이다. 데이터베이스가 사용되는 경우에도 일관된 백업을 수행한다. pg_dump는 스크립트 파일이나 다른 아카이브 파일로 추출한다.

사용법은 아래와 같다.

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

Options

dbname

덤프 할 데이터베이스의 이름을 지정한다. 이것이 지정되지 않으면 연결에 지정된 사용자 이름이 사용된다.

-a

--data-only

스키마(데이터 정의)가 아닌 데이터만 덤프한다.

-b

--blobs

덤프에 large object를 포함한다. 이것은 `--schema`, `--table` 또는 `--schema-only`가 지정된 경우를 제외하고는 기본 동작이다. 따라서 `-b` 스위치는 특정 스키마 또는 테이블이 요청된 덤프에 대형 오브젝트를 추가하는 경우에만 유용하다. blob은 데이터로 간주되므로 `--data-only`가 사용될 때 포함되지만 `--schema-only`가 아닌 경우 포함되지 않는다.

-c

--clean

데이터베이스 객체를 생성하기 위한 명령을 출력하기 전에 데이터베이스 객체를 정리(삭제)하는 명령을 출력한다. (`--if-exists`가 지정되지 않고 대상 데이터베이스에 개체가 존재하지 않는 경우 잘못된 오류 메시지를 생성할 수 있다.)

이 옵션은 일반 텍스트 형식에서만 의미가 있다. 아카이브 형식의 경우 `pg_restore`를 호출할 때 옵션을 지정할 수 있다.

-C

--create

데이터베이스 자체를 작성하고 작성된 데이터베이스에 다시 연결하는 명령으로 출력을 시작한다(이 양식의 스크립트를 사용하면 스크립트를 실행하기 전에 연결할 대상 설치의 데이터베이스가 중요하지 않다). `--clean` 도 지정되면 스크립트는 대상 데이터베이스를 삭제 한 후 다시 연결한다.

이 옵션은 일반 텍스트 형식에서만 의미가 있다. 아카이브 형식의 경우 `pg_restore`를 호출할 때 옵션을 지정할 수 있다.

-E ***encoding***

--encoding=***encoding***

지정된 문자 세트 인코딩으로 덤프를 작성한다. 기본적으로 덤프는 데이터베이스 인코딩으로 작성된다(동일한 결과를 얻는 또 다른 방법은 `PGCLIENTENCODING` 환경 변수를 원하는 덤프 인코딩으로 설정하는 것이다).

-f ***file***

--file=***file***

출력을 지정된 파일로 전송한다. 이 파라미터는 파일 기반 출력 형식에 대해 생략할 수 있으며, 이 경우에는 표준 출력이 사용된다. 그러나 파일 대신 대상 디렉토리를 지정하는 디렉토리 출력 형식에 대해 제공되어야 한다. 이

경우 디렉토리는 pg_dump에 의해 만들어지며 기존에 존재하지 않아야한다.

-F ***format***

--format=***format(c|d|t|p)***

출력 형식을 선택한다. format은 다음 중 하나 일 수 있다.

- p

  plain

  일반 텍스트 SQL 스크립트 파일을 출력한다(기본값).

- c

  custom

  pg_restore 입력에 적합한 사용자 정의 형식 아카이브를 출력한다. 디렉터리 출력 형식과 함께 이 옵션은 복원 중에 보관된 항목을 수동으로 선택하고 순서를 변경할 수 있는 가장 유연한 출력 형식이다. 이 형식은 기본적으로 압축된다.

- d

  directory

  pg_restore 입력에 적합한 사용자 정의 형식 아카이브를 출력한다. 이렇게 하면 각 테이블과 blob을 덤프할 파일 하나와 pg_restore가 읽을 수 있는 기계 가독 형식으로 덤프된 오브젝트를 설명하는 소위 목차 파일이 있는 디렉터리가 생성된다. 디렉터리 형식 아카이브는 표준 Unix 도구로 조작할 수 있다. 예를 들어, 압축되지 않은 아카이브의 파일은 gzip 도구로 압축할 수 있다. 이 형식은 기본적으로 압축되어 있으며 병렬 덤프를 지원한다.

- t

  tar

  pg_restore 에 입력에 적합한 tar 형식 아카이브를 출력한다. tar 형식은 디렉토리 형식과 호환된다. tar 형식 아카이브를 추출하면 유효한 디렉토리 형식 아카이브가 작성된다. 그러나 tar 형식은 압축을 지원하지 않는다. 또한 tar 형식을 사용하면 테이블 데이터 항목의 상대 순서를 복원 중에 변경할 수 없다.

-j ***njobs***

--jobs=***njobs***

njobs 테이블을 동시에 덤프하여 덤프를 병렬로 실행한다. 이 옵션은 덤프 시간을 줄이지만 데이터베이스 서버의 로드도 증가시킨다. 이 옵션은 여러 프로세스가 동시에 데이터를 쓸 수 있는 유일한 출력 형식이기 때문에 디렉터리 출력 형식으로만 사용할 수 있다.

pg_dump는 데이터베이스에 대한 ***njobs***+1 연결을 열 것이므로 max_connections 설정이 모든 연결을 수용할 수 있을 만큼 충분히 높아야 한다.

병렬 덤프를 실행하는 동안 데이터베이스 오브젝트에 exclusive lock을 요청하면 덤프가 실패 할 수 있다. 그 이유는 pg_dump master 프로세스는 worker 프로세스가 나중에 덤프 할 오브젝트에 대한 shared lock을 요청하여

덤프가 실행되는 동안 아무도 이를 삭제하지 않도록 한다. 다른 클라이언트가 테이블에 대한 exclusive lock을 요청하면 해당 lock은 부여되지 않지만 master 프로세스의 shared lock이 해제될 때까지 대기한다. 결과적으로 테이블에 대한 다른 액세스는 부여되지 않고 exclusive lock 요청 이후에 대기한다. 여기에는 테이블을 덤프하려는 worker 프로세스가 포함된다. 주의 사항이 없으면 고전적인 deadlock 상태가 된다. 이 충돌을 감지하기 위해 pg_dump worker 프로세스는 NOWAIT 옵션을 사용하여 또 다른 shared lock을 요청한다. worker 프로세스에 이 shared lock이 부여되지 않은 경우 다른 누군가가 exclusive lock을 요청해야 하며 덤프를 계속할 방법이 없기 때문에 pg_dump는 덤프를 중단 할 수밖에 없다.

일관된 백업을 위해 데이터베이스 서버는 동기화된 스냅 샷을 지원해야 한다. 이 기능을 사용하면 데이터베이스 클라이언트는 서로 다른 연결을 사용하더라도 동일한 데이터 세트를 볼 수 있다. pg_dump -j는 복수의 데이터베이스 접속을 사용한다. master 프로세스로 한 번 데이터베이스에 연결하고 각 worker job에 대해 다시 한 번 연결한다. 동기화된 스냅 샷 기능이 없으면 다른 worker job이 각 연결에서 동일한 데이터를 볼 수 없으므로 일관성 없는 백업이 발생할 수 있다.

-n ***schema***

--schema=***schema***

스키마와 일치하는 ***schema***만 덤프한다. 이것은 스키마 자체와 스키마에 포함된 모든 객체를 모두 선택한다. 이 옵션을 지정하지 않으면 대상 데이터베이스의 모든 비 시스템 스키마가 덤프된다. -n 스위치를 여러개 작성하면 여러 스키마를 선택할 수 있다. 또한 스키마 매개 변수는 agens의 \d명령에 사용되는 규칙과 동일한 패턴으로 해석되며 패턴에 와일드 카드 문자를 쓰면 복수의 스키마도 선택할 수 있다. 와일드 카드를 사용할 때는 쉘이 와일드 카드를 확장하지 못하게 하려면 따옴표를 사용한다(Examples를 참조한다).

> Notice: -n이 지정된 경우 pg_dump는 선택한 스키마에 따라 달라질 수 있는 다른 데이터베이스 개체를 덤프하지 않는다. 따라서, 특정 스키마 덤프의 결과가 그 자체로 정상적으로 깨끗한 데이터베이스로 복원될 수 있다는 보장은 없다.
>
> Notice: blob과 같은 스키마가 아닌 오브젝트는 -n이 지정된 경우에는 덤프 되지 않는다. --blobs 스위치를 사용하여 blob을 덤프에 다시 추가 할 수 있다.

-N ***schema***

--exclude-schema=***schema***

schema 패턴과 일치하는 ***schema***를 덤프하지 않는다. 패턴은 -n과 동일한 규칙에 따라 해석된다. -N은 여러 패턴 중 하나와 일치하는 스키마를 제외하기 위해 두 번 이상 사용될 수 있다.

-n과 -N이 둘 다 주어지면 이 동작은 -N 스위치의 것을 제외하고 적어도 하나의 -n 스위치와 일치하는 스키마를 덤프한다. -n 없이 -N이 나타나는 경우, -N과 일치하는 스키마는 일반 덤프에서 제외한다.

-o

--oids

개체 식별자(OID)를 모든 테이블의 데이터 일부로 덤프한다. 응용 프로그램이 어떤 방식(예 : 외래키 제약 조건)으로 OID열을 참조하는 경우 이 옵션을 사용한다. 그렇지 않으면 이 옵션을 사용할 수 없다.

-O

--no-owner

원래 데이터베이스와 일치하도록 개체의 소유권을 설정하는 명령을 출력하지 않는다. 기본적으로 pg_dump는 생성된 데이터베이스 객체의 소유권을 설정하기 위해 ALTER OWNER 또는 SET SESSION AUTHORIZATION문을 발행한다. superuser(또는 스크립트의 모든 오브젝트를 소유하는 동일한 사용자)가 이 명령을 시작하지 않는 한 이 명령은 스크립트가 실행될 때 실패한다. 모든 사용자가 복원할 수 있지만 해당 사용자에게 모든 개체에 대한 소유권을 부여할 수 있는 스크립트를 만들려면 -O를 지정한다.

이 옵션은 일반 텍스트 형식에서만 의미가 있다. 아카이브 형식의 경우 pg_restore를 호출할 때 옵션을 지정할 수 있다.

-s

--schema-only

데이터가 아닌 객체 정의(스키마)만 덤프한다.

이 옵션은 --data-only의 반대이다. 이것은 --section=pre-data --section=post-data와 유사하다.

(``schema''라는 단어를 다른 의미로 사용하는 --schema 옵션과 혼동하지 않도록한다.)

데이터베이스에 있는 테이블의 서브 세트에 대해서만 테이블 데이터를 제외하려면 --exclude-table-data를 참조한다.

-S ***username***

--superuser=***username***

트리거를 사용하지 않을 때 사용할 superuser 이름을 지정한다. 이 옵션은 --disable-triggers가 사용 된 경우에만 관련이 있다(일반적으로 이 과정을 생략하는 것이 좋으며, 대신에 결과를 슈퍼 컴퓨터로 시작하는 것이 좋다).

-t ***table***

--table=***table***

테이블과 이름이 일치하는 ***table***만 덤프한다. 이를 위해 ``테이블''에는 views, materialized views, sequences, foreign tables이 포함된다. 여러 개의 -t 스위치를 작성하여 여러 테이블을 선택할 수 있다. 또한, ***table*** 매개 변수는 agens의 \d 명령과 같은 패턴으로 해석되므로 패턴에 와일드 카드 문자를 써서 여러 테이블을 선택할 수도 있다. 쉘이 와일드 카드를 확장하지 못하게 하려면 따옴표를 사용한다(Examples를 참조한다).

-t로 선택한 테이블은 스위치에 관계없이 덤프되고 테이블이 아닌 오브젝트들은 덤프되지 않기 때문에 -t를 사용할 경우 -n 및 -N 스위치는 작동하지 않는다.

> Notice: -t를 지정하면, pg_dump는 선택한 테이블이 의존하는 다른 데이터베이스 객체를 덤프하려고 하지 않는다. 따라서 특정 데이터베이스에 대한 결과를 자체 데이터베이스로 성공적으로 복원할 수 있다는 보장은 없다.

-T ***table***

--exclude-table=***table***

table 패턴과 일치하는 테이블을 덤프하지 않는다. 이 패턴은 -t와 동일한 규칙에 따라 해석된다. -T는 여러가지 패턴 중 하나와 일치하는 테이블을 제외하고 한번 이상 주어질 수 있다.

-t와 -T가 모두 주어졌을 때, 이 동작은 -T 스위치의 것을 제외하고 적어도 하나의 -t스위치와 일치 하는 테이블 덤프한다. -T가 -t 없이 사용되는 경우, -T와 일치하는 테이블은 일반 덤프에서 제외됩니다.

-v

--verbose

상세 모드를 지정한다. 이렇게 하면 pg_dump가 상세한 개체 설명 및 시작/중지 시간을 덤프 파일로 출력하고 메시지를 표준 오류로 진행한다.

-V

--version

pg_dump의 버전을 표시하고 종료한다.

-x

--no-privileges

--no-acl

액세스 권한의 덤프를 금지한다(grant/revoke 명령).

-Z ***0..9***

--compress=***0..9***

사용할 압축 수준을 지정한다. 0은 압축이 없음을 의미한다. 사용자 정의 아카이브 형식의 경우 개별 테이블 데이터 세그먼트의 압축을 지정하며 기본값은 중간 수준에서 압축하는 것이다. 일반 텍스트 출력의 경우, 0이 아닌 압축 레벨을 설정하면 gzip을 통해 공급된 것처럼 전체 출력 파일이 압축된다. 그러나 기본값은 압축되지 않는다. tar 아카이브 형식은 현재 압축을 전혀 지원하지 않는다.

--binary-upgrade

이 옵션은 전체 업그레이드 유틸리티에서 사용한다. 다른 용도로 사용하는 것은 권장되거나 지원되지 않는다. 옵션의 동작은 예고 없이 향후 릴리스에서 변경될 수 있다.

--column-inserts

명시적인 컬럼명이 있는 INSERT 명령문(INSERT INTO ***table*** (***column***, ⋯) VALUES ⋯)으로 데이터를 덤프한다. 이렇게 하면 복원 속도가 매우 느려진다. 비 Agens 데이터베이스에 로드 할 수 있는 덤프를 만드는데 주로 유용하다. 이 옵션은 각 행에 대해 별도의 명령을 생성하므로 행을 다시 로드하는 과정에서 오류가 발생하면 전체 테이블 내용이 아닌 해당 행 만 손실된다.

--disable-dollar-quoting

이 옵션은 함수 본문에 대한 달러 인용 부호 사용을 비활성화하고 SQL 표준 문자열 구문을 사용하여 인용 부호를 사용하도록 한다.

--disable-triggers

이 옵션은 데이터 전용 덤프를 생성할 때만 관련이 있다. pg_dump가 데이터가 다시 로드되는 동안 대상 테이블에 대한 트리거를 일시적으로 비활성화하는 명령을 포함 하도록 지시한다. 데이터를 다시 로드하는 동안 호출을 원하지 않는 참조 무결성 검사 또는 테이블 위에 다른 트리거가 있는 경우 이 옵션을 사용한다.

현재 --disable-triggers에 대해 생성된 명령은 superuser로 수행해야 한다. 따라서 -S로 superuser 이름을 지정

하거나 결과 스크립트를 superuser로 시작할 때 주의해야 한다.

이 옵션은 일반 텍스트 형식에서만 의미가 있다. 아카이브 형식의 경우 pg_restore를 호출할 때 옵션을 지정할 수 있다.

--enable-row-security

이 옵션은 행 보안이 있는 테이블의 내용을 덤프 할 때만 관련이 있다. 기본적으로 pg_dump는 row_security를 off로 설정하여 모든 데이터가 테이블에서 덤프 되도록 한다. 사용자가 행 보안을 생략할 수 있는 충분한 권한이 없는 경우 오류가 발생한다. 이 매개 변수는 pg_dump가 row_security를 대신 on으로 설정하도록 함으로써 사용자가 액세스할 수 있는 테이블 컨텐츠의 일부를 덤프 하도록 허용한다.

--exclude-table-data=***table***

table 패턴과 일치하는 테이블에 대한 데이터를 덤프하지 않는다. --exclude-table-data는 여러 패턴 중 하나와 일치하는 테이블을 제외하기 위해 한 번 이상 제공될 수 있다. 이 옵션은 데이터가 필요하지 않더라도 특정 테이블의 정의가 필요할 때 유용하다.

데이터베이스의 모든 테이블에 대한 데이터를 제외하려면 --schema-only를 참조한다.

--if-exists

데이터베이스 오브젝트를 정리할 때 조건부 명령 (예 : IF EXISTS 절 추가) 을 사용한다. 이 옵션은 --clean을 지정하지 않으면 유효하지 않다.

--inserts

COPY가 아닌 INSERT 명령으로 데이터를 덤프한다. 이렇게 하면 복원 속도가 매우 느려진다. 비 AgensGraph 데이터베이스에 로드 할 수 있는 덤프를 만드는 데 주로 유용하다. 그러나 이 옵션은 각 행에 대해 별도의 명령을 생성하므로 행을 다시 로드 할 때 오류가 발생하면 전체 테이블 내용이 아닌 해당행 만 손실된다. 열 순서를 재배열하면 복원에 실패할 수 있다. --column-insert 옵션은 속도가 느리지만, 열 순서 변경 시 안전하게 사용할 수 있다.

--lock-wait-timeout=***timeout***

덤프의 시작 부분에서 공유 테이블 lock을 획득하기 위해 영구적으로 기다리지 않는다. 지정된 ***timeout*** 내에 테이블을 lock 할 수 없는 경우에는 실패한다. 타임 아웃은 SET statement_timeout이 허용하는 형식으로 지정할 수 있다 (허용되는 값은 정수 (밀리초) 이다).

--no-security-labels

보안 레이블을 덤프하지 않는다.

--no-synchronized-snapshots

이 옵션을 사용하면 pg_dump -j를 실행할 수 있다. 자세한 내용은 -j 매개 변수의 설명을 참조한다.

--no-tablespaces

테이블 공간을 선택하는 명령을 출력하지 않는다. 이 옵션을 사용하면 복원하는 동안 기본 값이 되는 테이블 영역에 모든 개체가 생성된다.

이 옵션은 일반 텍스트 형식에서만 의미가 있다. 아카이브 형식의 경우 pg_restore를 호출 할 때 옵션을 지정할 수 있다.

--no-unlogged-table-data

로그 되지 않은 테이블의 내용을 덤프 하지 않는다. 이 옵션은 테이블 정의(스키마)가 덤프 되는지에 영향을 주지 않는다. 테이블 데이터를 덤프 하는 것만을 제한한다. 로그 되지 않은 테이블의 데이터는 대기 서버에서 덤프 할 때 항상 제외된다.

--quote-all-identifiers

모든 식별자를 강제 인용한다. 이 옵션은 AgensGraph 주 버전이 pg_dump와 다른 서버에서 데이터베이스를 덤프 할 때나 출력을 다른 주 버전의 서버에 로드 할 때 권장된다. 기본적으로, pg_dump는 자신의 메이저 버전에서 예약어인 식별자만을 인용한다. 이는 때때로 다른 버전의 예약된 단어 집합이 있는 다른 버전의 서버를 처리할 때 호환성 문제가 발생할 수 있다. --quote-all-identifier를 사용하면 읽기 어려운 덤프 스크립트를 사용하는 대신 이러한 문제가 발생하는 것을 방지할 수 있다.

--section=***section***

명명된 섹션만 덤프한다. ***section***은 pre-data, data, post-data가 될 수 있다. 이 옵션은 여러 섹션을 선택하기 위해 두 번 이상 지정 될 수 있다. 기본값은 모든 섹션을 덤프하는 것이다.

데이터 섹션에는 실제 테이블 데이터, 대용량 객체 컨텐츠 및 시퀀스 값이 포함된다. post-data 항목에는 유효성이 검사 제한이 아닌 indexes, triggers, rules 및 제약 조건의 정의가 포함된다. pre-data 항목에는 다른 모든 데이터 정의 항목이 포함된다.

--serializable-deferrable

덤프에 serializable 트랜잭션을 사용하여 사용된 스냅 샷이 나중의 데이터베이스 상태와 일치하는지 확인한다. 예외가 존재하지 않는 트랜잭션 스트림의 지점을 기다림으로써 덤프가 실패하거나 다른 트랜잭션이 serialization_failure로 롤백 될 위험이 없도록 이 작업을 수행한다.

이 옵션은 재해 복구 전용 덤프에는 유용하지 않다. 데이터베이스가 계속 업데이트되는 동안 데이터베이스 복사본을 로드하거나 다른 읽기 전용 로드 공유를 위해 데이터베이스 복사본을 로드하는 데 유용할 수 있다. 그것이 없다면, 덤프는 결국 커밋된 거래의 어떠한 연속적인 실행과 일치하지 않는 상태를 반영할 수 있다. 예를 들어 일괄 처리 기술을 사용하면 배치에 있는 모든 항목을 표시하지 않고 일괄 처리를 닫은 것으로 표시 할 수 있다.

이 옵션은 pg_dump가 시작될 때 활성화된 읽기/쓰기 트랜잭션이 없으면 아무런 차이가 없다. 읽기-쓰기 트랜잭션이 활성화된 경우, 덤프의 시작은 불확실한 시간 동안 지연될 수 있다. 일단 실행되면 스위치 사용 여부에 관계없이 성능은 동일하다.

--snapshot=***snapshot***

데이터베이스를 덤프할 때 지정된 동기화된 스냅 샷을 사용한다.

이 옵션은 덤프를 논리적 복제 슬롯 또는 동시 세션과 동기화해야 할 때 유용하다.

병렬 덤프의 경우 새 스냅 샷을 작성하는 대신 이 옵션으로 정의된 스냅 샷 이름이 사용된다.

--strict-names

각 스키마(-n/--schema) 및 테이블(-t/--table) 한정자가 덤프할 데이터베이스의 적어도 하나의 스키마/테이블과 일치하도록 요구한다. 일치하는 스키마/테이블 한정자가 없으면 pg_dump는 --strict-names 없이도 오류를 생성한다.

이 옵션은 -N/--exclude-schema, -T/--exclude-table, --exclude-table-data에 영향을 미치지 않는다. 오브젝트와 일치하지 않는 제외 패턴은 오류로 간주되지 않는다.

--use-set-session-authorization

오브젝트 소유권을 식별하기 위해 ALTER OWNER 명령 대신 SQL 표준 SET SESSION AUTHORIZATION명령을 출력한다. 이렇게 하면 덤프가 표준과 호환되지만 덤프에 있는 객체의 기록에 따라 제대로 복원되지 않을 수 있다. 또한 SET SESSION AUTHORIZATION을 사용하는 덤프는 superuser 권한이 올바르게 복원되어야 하지만 ALTER OWNER는 더 적은 권한을 요구한다.

-?

--help

pg_dump 명령 행 인수에 대한 도움말을 표시하고 종료한다.

다음 명령 줄 옵션은 데이터베이스 연결 매개 변수를 제어한다.

-d ***dbname***

--dbname=***dbname***

연결할 데이터베이스의 이름을 지정한다. 이것은 ***dbname***을 명령행에서 첫 번째 옵션이 아닌 인수로 지정하는 것과 같다.

이 매개 변수에 = 기호가 있거나 올바른 URI 접두사 (postgresql:// 또는 postgres://) 로 시작하는 경우 conninfo 문자열로 처리된다.

-h ***host***

--host=***host***

서버가 실행중인 시스템의 호스트 이름을 지정한다. 값이 슬래시로 시작하면 Unix 도메인 소켓의 디렉토리로 사용된다. 기본값은 PGHOST 환경 변수에서 가져온다 (설정되어 있는 경우). 그렇지 않으면 Unix 도메인 소켓 연결이 시도된다.

-p ***port***

--port=***port***

서버가 연결을 수신 대기하는 TCP 포트 또는 로컬 유닉스 도메인 소켓 파일 확장자를 지정한다. 기본 값은 PGPORT 환경 변수이며 설정된 경우 또는 컴파일된 기본 값으로 설정한다.

-U ***username***

--username=***username***

연결할 사용자 이름이다.

-w

--no-password

암호를 묻는 메시지를 표시하지 않는다. 서버가 암호 인증을 요구하고 .pgpass 파일과 같은 다른 방법으로 암호를 사용할 수 없는 경우 연결 시도가 실패한다. 이 옵션은 암호를 입력할 수 있는 사용자가 없는 배치 작업 및 스크립트에서 유용할 수 있다.

-W

--password

데이터베이스에 연결하기 전에 pg_dump를 실행하여 암호를 입력하도록 한다.

서버가 패스워드 인증을 요구하면, pg_dump는 자동적으로 패스워드를 요구하기 때문에, 이 옵션은 절대로 필요하지 않다. 그러나 pg_dump는 서버가 암호를 원한다는 사실을 알아내는데 연결 시도를 낭비한다. 경우에 따라서 여분의 연결 시도를 피하기 위해 -W를 입력하는 것이 좋다.

--role=***rolename***

덤프를 작성하는데 사용될 역할 이름을 지정한다. 이 옵션을 사용하면 데이터베이스에 연결한 후 pg_dump가 SET ROLE ***rolename*** 명령을 실행한다. 인증된 사용자(-U에 의해 지정됨)가 pg_dump에 필요한 권한이 부족하지만 필요한 권한으로 전환할 수 있는 권한이 있는 경우 유용하다. superuser로 직접 로그인하지 못하는 정책이 있을 경우 이 옵션을 사용하면 정책을 위반하지 않고 덤프를 생성할 수 있다.

Examples

mydb라는 데이터베이스를 SQL 스크립트 파일로 덤프하려면 다음을 수행한다.

```
$ pg_dump mydb > db.sql
```

newdb라는 이름의 새롭게 생성된 데이터베이스로 스크립트를 다시 로드하려면 다음을 수행한다.

```
$ agens -d newdb -f db.sql
```

데이터베이스를 사용자 정의 형식 아카이브 파일로 덤프하려면 다음을 수행한다.

```
$ pg_dump -Fc mydb > db.dump
```

데이터베이스를 디렉토리 형식 아카이브로 덤프하려면 다음을 수행한다.

```
$ pg_dump -Fd mydb -f dumpdir
```

5개의 worker job과 병렬로 데이터베이스를 디렉토리 형식 아카이브로 덤프하려면 다음을 수행한다.

```
$ pg_dump -Fd mydb -j 5 -f dumpdir
```

newdb라는 새롭게 생성된 데이터베이스로 아카이브 파일을 다시 로드하려면 다음을 수행한다.

```
$ pg_restore -d newdb db.dump
```

mytab라는 단일 테이블을 덤프하려면 다음을 수행한다.

```
$ pg_dump -t mytab mydb > db.sql
```

employee_log라는 테이블을 제외하고 detroit 스키마에서 이름이 emp로 시작하는 모든 테이블을 덤프하려면 다음을 수행한다.

```
$ pg_dump -t 'detroit.emp*' -T detroit.employee_log mydb > db.sql
```

이름이 east 또는 west로 시작하고 gsm으로 끝나는 모든 스키마를 덤프하고 이름에 test라는 단어가 포함 된 스키마를 제외하려면 다음을 수행한다.

```
$ pg_dump -n 'east*gsm' -n 'west*gsm' -N '*test*' mydb > db.sql
```

또는

```
$ pg_dump -n '(east|west)*gsm' -N '*test*' mydb > db.sql
```

이름이 ts_로 시작하는 테이블을 제외한 모든 데이터베이스 개체를 덤프하려면 다음을 수행한다.

```
$ pg_dump -T 'ts_*' mydb > db.sql
```

-t 및 관련 스위치에서 대문자 또는 대/소문자를 혼용하는 이름을 지정하려면 이름을 큰 따옴표로 묶어야한다. 그렇지 않으면 소문자로 변환된다. 그러나 큰 따옴표는 쉘에만 적용되므로 따옴표로 묶어야한다. 따라서 대/소문자가 혼합된 단일 테이블을 덤프하려면 다음과 같이 사용해야 한다.

```
$ pg_dump -t "\"MixedCaseName\"" mydb > mytab.sql
```

7.1.3 pg_restore

pg_restore는 pg_dump로 생성된 아카이브로 AgensGraph 데이터베이스를 복원하는 유틸리티이다. 데이터베이스를 저장한 시점의 상태로 데이터베이스를 재구성하는데 필요한 명령을 실행한다. 또한 아카이브 파일을 사용하면 pg_restore가 복원 대상을 선택하거나 복원하기 전에 항목 순서를 변경할 수 있다. 아카이브 파일은 전반적인 아키텍처에 적용 가능하도록 설계되어 있다.

pg_restore는 두 가지 모드로 작동 할 수 있다. 데이터베이스 이름이 지정되면 pg_restore는 해당 데이터베이스에 연결하고 아카이브 내용을 데이터베이스에 직접 복원한다. 그렇지 않으면, 데이터베이스를 재빌드하는데 필요한 SQL 명령을 포함하는 스크립트가 작성되어 파일이나 표준 출력에 기록된다. 이 스크립트 출력은 pg_dump의 일반 텍스트 출력 형식과 동일하다. 따라서 출력을 제어하는 옵션 중 일부는 pg_dump 옵션과 유사하다.

물론, pg_restore는 아카이브 파일에 없는 정보를 복원 할 수 없다. 예를 들어, ``INSERT 명령으로 데이터 덤프'' 옵션을 사용하여 아카이브를 만든 경우 pg_restore는 COPY 문을 사용하여 데이터를 로드 할 수 없다.

사용법은 아래와 같다.

```
$ pg_restore [OPTION]... [FILE]
```

Options

pg_restore는 다음 명령 행 인수를 허용한다.

filename

복원할 아카이브 파일(또는 디렉터리 형식 아카이브의 경우 디렉터리)의 위치를 지정한다. 지정하지 않으면 표준 입력이 사용된다.

-a

--data-only

스키마 (데이터 정의)가 아닌 데이터만 복원한다. 테이블 데이터, 대형 오브젝트 및 시퀀스 값은 아카이브에 있는 경우 복원된다.

이 옵션은 `--section=data`를 지정하는 것과 유사하다.

-c

--clean

데이터베이스 개체를 다시 생성하기 전에 삭제(drop) 한다(`--if-exists`를 사용 하지 않으면 대상 데이터베이스에 개체가 없는 경우 무해한 오류 메시지가 생성 될 수 있다).

-C

--create

복원하기 전에 데이터베이스를 생성한다. `-clean`과 함께 지정된 경우 연결하기 전에 대상 데이터베이스를 삭제하고 다시 생성한다.

이 옵션을 사용하면 `-d`로 명명된 데이터베이스가 초기 `DROP DATABASE` 및 `CREATE DATABASE` 명령을 실행하는 데에만 사용된다. 모든 데이터는 아카이브에 나타나는 데이터베이스 이름으로 복원된다.

-d ***dbname***

--dbname=***dbname***

데이터베이스 ***dbname***에 연결하고 데이터베이스에 직접 복원한다.

-e

--exit-on-error

SQL 명령을 데이터베이스로 보내는 중에 오류가 발생하면 종료한다. 기본값은 계속 진행하고 복원이 끝날 때 오류 수를 표시하는 것이다.

-f ***filename***

--file=***filename***

생성된 스크립트의 출력 파일을 지정하거나 `-l`과 함께 사용하는 경우 나열할 파일을 지정한다. 기본값은 표준 출력이다.

-F ***format***

--format=***format***

아카이브 형식을 지정한다. `pg_restore`가 자동적으로 형식을 결정하므로, 형식을 지정할 필요는 없다. 지정된 경우 다음 중 하나일 수 있다.

- c

  custom

아카이브는 pg_dump의 사용자 정의 형식이다.

- d

  directory

  아카이브는 디렉토리 아카이브이다.

- t

  tar

  아카이브는 tar 아카이브이다.

-I ***index***

--index=***index***

명명된 인덱스의 정의만을 복원한다. 다중 인덱스는 여러 개의 -I 스위치로 지정 될 수 있다.

-j ***number-of-jobs***

--jobs=***number-of-jobs***

pg_restore에서 가장 시간이 많이 걸리는 부분, 즉 여러 개의 동시 작업을 사용하여 데이터를 로드하거나 인덱스를 작성하거나, 제약 조건을 생성 하는 부분을 실행한다. 이 옵션은 대형 데이터베이스를 다중 프로세서 시스템에서 실행중인 서버로 복원하는 시간을 크게 줄인다.

각 작업은 운영 체제에 따라 하나의 프로세스 또는 하나의 스레드이며 서버에 별도의 연결을 사용한다.

이 옵션의 최적 값은 서버, 클라이언트 및 네트워크의 하드웨어 설정에 따라 다르다. CPU 코어 수 및 디스크 설정 등이 요인이다. 서버의 CPU 코어 수이지만, 이보다 큰 값을 지정하면 대부분의 경우 복원 시간이 단축될 수 있다. 너무 높은 값은 메모리를 올리고 내리는 작업 (thrashing) 때문에 성능이 저하될 수도 있다.

이 옵션에서는 사용자 정의 및 디렉터리 아카이브 형식만 지원된다. 입력은 일반 파일이나 디렉터리 여야 한다. 이 옵션은 데이터베이스 서버에 직접 연결하지 않고 스크립트를 내보낼 때 무시된다. 또한 여러 작업을 --single-transaction 옵션과 함께 사용할 수 없다.

-l

--list

아카이브의 내용을 나열한다. 이 작업의 출력은 -L옵션의 입력으로 사용할 수 있다. -n 또는 -t와 같은 필터링 스위치를 -l과 함께 사용하면 나열된 항목을 제한한다.

-L ***list-file***

--use-list=***list-file***

list-file에 나열된 아카이브 요소만 복원하고 파일에 표시된 순서대로 복원한다. -L과 함께 -n 또는 -t와 같은 필터링 스위치를 사용하는 경우 복원되는 항목을 추가로 제한한다.

list-file은 일반적으로 이전의 -l 조작의 출력을 편집하여 작성된다. 라인은 이동하거나 제거할 수 있으며, 라인 시작 부분에 세미콜론 (;) 을 붙여 주석을 사용할 수도 있다.

-n ***namespace***

--schema=***schema***

명명된 스키마에 있는 개체만 복원한다. 다중 스키마는 다중 -n 스위치로 지정 될 수 있다. 이 옵션을 -t 옵션과 결합하여 특정 테이블만 복원 할 수 있다.

-O

--no-owner

원래 데이터베이스와 일치하도록 개체의 소유권을 설정하는 명령을 출력하지 않는다. 기본적으로 pg_restore는 ALTER OWNER 또는 SET SESSION AUTHORIZATION문을 통해 생성된 스키마 요소의 소유권을 설정한다. superuser(또는 스크립트의 모든 객체를 소유하는 동일한 사용자)가 데이터베이스에 초기 연결을 하지 않으면 이 명령문은 실패한다. -O를 사용하면 모든 사용자 이름을 초기 연결에 사용할 수 있으며 이 사용자는 생성 된 모든 객체를 소유하게 된다.

-P ***function-name(argtype [, ⋯])***

--function=***function-name(argtype [, ⋯])***

명명된 기능만 복원한다. 덤프 파일의 목차에 표시된 대로 함수 이름과 인수의 철자를 주의한다. 다중 기능은 다중 -P 스위치로 지정될 수 있다.

-s

--schema-only

데이터가 아닌 아카이브에 있는 스키마 항목에 해당하는 범위까지의 스키마(데이터 정의)만 복원한다.
이 옵션은 --data-only 의 반대이다. --section=pre-data,--section=post-data`를 지정하는 것과 비슷하다.
(이것을 ``schema'' 라는 단어를 다른 의미로 사용하는 --schema 옵션과 혼동하지 않도록 한다.)

-S ***username***

--superuser=***username***

트리거를 비활성화할 때 사용할 superuser 이름을 지정한다. 이는 --disable-triggers가 사용되는 경우에만 관련이 있다.

-t ***table***

--table=***table***

명명된 테이블의 정의 또는 데이터를 복원한다. 이러한 목적으로, ``table'' 에는 views, materialized views, sequences, foreign tables가 포함된다. 여러 개의 -t 스위치를 기록하여 여러 개의 테이블을 선택할 수 있다. 이 옵션을 -n옵션과 결합하여 특정 스키마의 테이블을 지정할 수 있다.

> Note: -t를 지정 하면, pg_restore는 선택된 테이블이 의존하는 다른 데이터베이스 객체를 복원하려고 하지 않는다. 따라서 특정 테이블을 깨끗한 데이터베이스로 복원한다는 보장은 없다.
>
> Note: 이 플래그는 pg_dump의 -t 플래그와 동일하게 작동하지 않는다. 현재 pg_restore 에서 와일드 카드 매칭을 위한 조항이 없으며 -t 내에 스키마 이름을 포함할 수 없다.

-T ***trigger***

--trigger=***trigger***

이름이 지정된 트리거만 복원한다. 다중 트리거는 다중 -T 스위치로 지정될 수 있다.

-v

--verbose

상세 모드를 지정한다.

-V

--version

pg_restore의 버전을 표시하고 종료한다.

-x

--no-privileges

--no-acl

액세스 권한의 복원을 막는다(grant/revoke 명령).

-1

--single-transaction

복원을 단일 트랜잭션으로 실행한다(즉, BEGIN/COMMIT에서 방출된 명령을 랩핑한다). 이렇게 하면 모든 명령이 성공적으로 완료되거나 변경 사항이 적용되지 않는다. 이 옵션은 --exit-on-error를 의미한다.

--disable-triggers

이 옵션은 데이터 전용 복원을 수행할 때만 관련된다. pg_restore가 명령을 실행하여 데이터가 다시 로드되는 동안 대상 테이블에서 트리거를 일시적으로 사용하지 않도록 지시한다. 데이터를 다시 로드하는 동안 호출하지 않으려는 테이블에서 참조 무결성 검사 또는 기타 트리거를 사용하는 경우 이 옵션을 사용한다.

현재 --disable-trigger에 대한 명령은 superuser로 수행되어야 한다. 따라서 -S와 함께 superuser이름을 지정하거나, AgensGraph 슈퍼 사용자로 pg_restore를 실행한다.

--enable-row-security

이 옵션은 행 보안이 있는 테이블의 내용을 복원할 때만 관련된다. 기본적으로 pg_restore는 모든 데이터가 테이블에 복원되도록 row_security를 off로 설정한다. 사용자가 행 보안을 생략할 수 있는 충분한 권한이 없는 경우 오류가 발생한다. 이 매개 변수는 pg_restore가 row_security를 on으로 설정하여 행 보안이 활성화된 상태에서 테이블의 내용을 복원할 수 있게 한다. 사용자가 덤프에서 테이블로 행을 삽입할 권한이 없는 경우에도 이 작업은 실패할 수 있다.

COPY FROM은 행 보안을 지원하지 않으므로 이 옵션은 현재 덤프가 INSERT 형식이어야 한다.

--if-exists

데이터베이스 오브젝트를 정리할 때 조건부 명령(예: IF EXISTS절 추가)을 사용한다. 이 옵션은 --clean을 지정하지 않으면 유효하지 않다.

--no-data-for-failed-tables

기본적으로 테이블 데이터 작성은 테이블 작성 명령이 실패한 경우에도 복원된다. 이 옵션을 사용하면 이러한 테이블의 데이터를 건너뛴다. 이 동작은 대상 데이터베이스에 원하는 테이블 내용이 이미 있는 경우 유용하다. 예를 들어 PostGIS와 같은 PostgreSQL extensions을 위한 보조 테이블은 이미 대상 데이터베이스에 로드되어 있을 수 있다. 이 옵션을 지정하면 중복되거나 쓸모없는 데이터가 로드되지 않는다.

이 옵션은 SQL 스크립트 출력을 생성할 때가 아니라 데이터베이스에 직접 복원할 때만 유효하다.

--no-security-labels

보안 레이블을 복원하는 명령은 아카이브에 포함되어 있어도 명령을 출력하지 않는다.

--no-tablespaces

tablespaces을 선택하는 명령을 출력하지 않는다. 이 옵션을 사용하면 모든 오브젝트가 복원 중 기본값인 모든 tablespaces에 작성된다.

--section=***section***

명명된 섹션만 복원한다. 섹션 이름은 pre-data, data, post-data가 될 수 있다. 이 옵션은 여러 섹션을 선택하기 위해 두 번 이상 지정될 수 있다. 기본값은 모든 섹션을 복원하는 것이다.

데이터 섹션에는 대형 오브젝트 정의뿐만 아니라 실제 테이블 데이터가 들어 있다. Post-data 항목은 유효성이 검사 제한이 아닌 indexes, triggers, rules 및 constraints의 정의로 구성된다. Pre-data 항목은 다른 모든 데이터 정의 항목으로 구성된다.

--strict-names

각 스키마(-n-schema-schema) 및 테이블(-T-테이블) 한정자가 백업 파일에 하나 이상의 스키마/테이블을 일치시킬 필요가 있다.

--use-set-session-authorization

오브젝트 소유권을 식별하기 위해 `ALTER OWNER` 명령 대신 SQL 표준 `SET SESSION AUTHORIZATION` 명령을 출력한다. 이렇게 하면 덤프가 표준과 호환되지만 덤프에 있는 객체의 기록에 따라 제대로 복원되지 않을 수 있다.

-?

--help

`pg_restore` 명령행 인수에 대한 도움말을 표시하고 종료한다.

또한 `pg_restore`는 연결 매개 변수에 대해 다음 명령행 인수를 허용한다.

-h ***host***

--host=***host***

서버가 실행중인 시스템의 호스트 이름을 지정한다. 값이 슬래시로 시작하면 Unix 도메인 소켓의 디렉토리로 사용된다. 기본값은 PGHOST 환경 변수에서 가져온다(설정되어 있는 경우). 그렇지 않으면 Unix 도메인 소켓 연결이 시도된다.

-p ***port***

--port=***port***

서버가 연결을 수신 대기하는 TCP 포트 또는 로컬 유닉스 도메인 소켓 파일 확장자를 지정한다. 기본값은 PGPORT 환경 변수이며 설정된 경우 또는 컴파일된 기본값으로 설정한다.

-U ***username***

--username=***username***

연결할 사용자 이름이다.

-w

--no-password

암호를 묻는 메시지를 표시하지 않는다. 서버가 암호 인증을 요구하고 .pgpass 파일과 같은 다른 방법으로 암호를 사용할 수 없는 경우 연결 시도가 실패한다. 이 옵션은 암호를 입력할 수 있는 사용자가 없는 배치 작업 및 스크립트에서 유용할 수 있다.

-W

--password

데이터베이스에 연결하기 전에 pg_restore를 실행하여 암호를 입력하도록 한다.

서버가 패스워드 인증을 요구하면, pg_restore는 자동적으로 패스워드를 요구하기 때문에, 이 옵션은 절대로 필요하지 않다. 그러나 pg_restore는 서버가 암호를 원한다는 사실을 알아내는데 연결 시도를 낭비한다. 경우에 따라서 여분의 연결 시도를 피하기 위해 -W를 입력하는 것이 좋다.

--role=***rolename***

덤프를 작성하는데 사용될 역할 이름을 지정한다. 이 옵션을 사용하면 데이터베이스에 연결한 후 pg_restore가 SET ROLE ***rolename*** 명령을 실행한다. 인증 된 사용자 (-U에 의해 지정됨) 가 pg_restore에 필요한 권한이 부족하지만 필요한 권한으로 전환할 수 있는 권한이 있는 경우 유용하다. superuser로 직접 로그인하지 못하는 정책이 있을 경우 이 옵션을 사용하면 정책을 위반하지 않고 덤프를 생성할 수 있다.

Examples

mydb라는 데이터베이스를 사용자 정의 형식의 덤프 파일에 덤프 했다고 가정한다.

```
$ pg_dump -Fc mydb > db.dump
```

데이터베이스를 삭제하고 덤프에서 데이터베이스를 다시 작성하려면 다음을 수행한다.

```
$ dropdb mydb
$ pg_restore -C -d postgres db.dump
```

-d 스위치에 명명된 데이터베이스는 클러스터에 있는 모든 데이터베이스가 될 수 있다. pg_restore는 이를 사용하여 mydb에 대한 CREATE DATABASE 명령을 실행한다. -C를 사용하면 데이터는 항상 덤프 파일에 나타나는 데이터베이스 이름으로 복원된다.

덤프를 newdb라는 새 데이터베이스로 다시 로드하려면 다음을 수행한다.

```
$ createdb -T template0 newdb
$ pg_restore -d newdb db.dump
```

-C를 사용하지 않고 복원할 데이터베이스에 직접 연결한다. template1이 아닌 template0에서 새 데이터베이스를 복제하여 초기에 비어 있는지 확인한다.

데이터베이스 항목을 재정렬하려면 먼저 아카이브의 목차를 덤프해야한다.

```
$ pg_restore -l db.dump > db.list
```

목록 파일은 각 항목에 대해 다음과 같이 머리글과 한줄로 구성된다. 예제는 다음과 같다.

```
;
; 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
```

세미콜론은 주석의 시작을 나타내고 줄의 시작 부분에 있는 숫자는 각 항목에 할당된 내부 아카이브 ID를 나타낸다.
파일의 행을 주석 처리, 삭제 및 재정렬할 수 있다. 예제는 다음과 같다.

```
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
```

pg_restore에 대한 입력으로 사용할 수 있으며 항목 10과 6만 순서대로 복원한다.

```
$ pg_restore -L db.list db.dump
```

7.2 Server Tool

7.2.1 ag_ctl

ag_ctl은 AgensGraph 클러스터를 초기화하고 데이터베이스 서버를 시작, 중지 또는 재시작하거나 실행 중인 서버의 상태를 표시하는 유틸리티이다. 서버를 수동으로 시작할 수 있지만 ag_ctl은 로그 출력을 재연결하고 터미널 및 프로세스 그룹에서 적절하게 분리하는 등의 작업을 수행한다. 또한 제어된 종료를 위한 편리한 옵션도 제공한다.

사용법은 아래와 같다.

```
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 또는 initdb 모드는 새로운 AgensGraph 데이터베이스 클러스터를 생성한다. 데이터베이스 클러스터는 단일 서버 인스턴스에서 관리되는 데이터베이스의 모음이다. 이 모드는 initdb 명령을 호출한다.

start 모드는 새로운 서버가 시작된다. 서버는 백그라운드에서 시작되고 표준 입력은 /dev/null(Windows에서 nul)에 연결된다. Unix 계열 시스템에서는 기본적으로 서버의 표준 출력과 표준 오류가 ag_ctl의 표준 출력으로 보내진다(표준 오류는 아니다). 그런 다음 ag_ctl의 표준 출력을 파일로 리디렉션하거나 rotatelogs와 같은 로그 회전 프로그램과 같은 다른 프로세스로 내보내야 한다. Windows에서는 기본적으로 서버의 표준 출력과 표준 오류가 터미널에 전송된다. 이러한 기본 동작은 -l을 사용하여 서버의 출력을 로그 파일에 추가하여 변경할 수 있다. -l 또는 출력 리디렉션 중 하나를 사용하는 것이 좋다.

stop 모드는 지정된 데이터 디렉토리에 실행하는 서버가 종료된다. -m 옵션을 사용하면 세 가지 다른 종료 방법을 선택할 수 있다. ``Smart'' 모드는 모든 활성 클라이언트가 연결 해제되고 온라인 백업이 완료 될 때까지 대기한다. 서버가 핫 스탠바이 상태인 경우 모든 클라이언트의 연결이 해제되면 복구 및 스트리밍 복제가 종료된다. ``Fast'' 모드(기본값)는 클라이언트의 연결이 끊길 때까지 기다리지 않고 진행중인 온라인 백업을 종료한다. 모든 활성 트랜잭션이 롤백되고 클라이언트가 강제로 연결이 끊어지면 서버가 종료된다. ``Immediate'' 모드는 즉시 모든 서버 프로세스를 중단한다. 이렇게 하면 다시 시작할 때 crash-recovery가 실행된다.

restart 모드는 효과적으로 중지를 실행 한 다음 시작을 실행한다. 이렇게 하면 agens 명령 행 옵션을 변경할 수 있다. 서버 시작 중 명령 행에 지정된 상대 경로가 지정된 경우 restart가 실패 할 수 있다.

`reload` 모드는 단순히 agens 프로세스에 SIGHUP 신호를 보내서 구성 파일 (postgresql.conf, pg_hba.conf 등) 을 다시 읽게 한다. 따라서 전체 재시작이 필요하지 않은 구성 파일 옵션을 변경할 수 있다.

`status` 모드는 서버가 지정된 데이터 디렉토리에서 실행 중인지 확인한다. 그럴 경우 PID 및 이를 호출하는데 사용된 명령 행 옵션이 표시된다. 서버가 실행 중이 아니면 프로세스는 종료 상태 3을 리턴한다. 액세스 가능한 데이터 디렉토리가 지정되지 않으면 프로세스는 종료 상태 4를 리턴한다.

`promote` 모드에서는 지정된 데이터 디렉토리에서 실행중인 대기 서버에 복구를 종료하고 읽기/쓰기 작업을 시작하도록 명령한다.

`kill` 모드를 사용하면 지정된 프로세스로 신호를 보낼 수 있다. 이것은 kill 명령이 없는 Windows에서 특히 유용하다. 지원되는 신호 이름 목록을 보려면 --help 를 사용한다.

Options

-c

--core-file

서버가 코어 파일에 저장된 소프트 리소스 제한을 해제하여 가능한 한 코어 파일을 생성하여 코어 파일을 생성할 수 있도록 허용한다. 이는 오류가 발생한 서버 프로세스에서 스택 추적을 얻을 수 있도록 하여 문제를 디버깅 또는 진단하는 데 유용하다.

-D ***datadir***

--pgdata ***datadir***

데이터베이스 구성 파일의 파일 시스템 위치를 지정한다. 생략된 경우 환경 변수 AGDATA를 사용한다.

-l ***filename***

--log ***filename***

서버 로그 출력을 ***filename***에 추가한다. 파일이 존재하지 않으면 생성된다. umask가 077로 설정되어 있으므로 로그 파일에 대한 액세스 권한이 기본적으로 다른 사용자에게 허용되지 않는다.

-m ***mode***

--mode ***mode***

종료 모드를 지정한다. ***mode***는 `smart`, `fast`, `immediate`, 이 세 가지 중 하나의 첫 글자일 수 있다. 설정되지 않을 경우 `fast`가 사용된다.

-o ***options***

`agens` 명령으로 전달할 옵션을 지정한다. 여러 옵션 호출이 추가된다.

옵션은 대개 그룹으로 전달되도록 단일 또는 큰 따옴표로 묶어야한다.

-o ***initdb-options***

`initdb` 명령에 직접 전달할 옵션을 지정한다.

옵션은 대개 그룹으로 전달되도록 단일 또는 큰 따옴표로 묶어야한다.

-p ***path***

agens실행 파일의 위치를 지정한다. 기본적으로 agens 실행 파일은 ag_ctl과 동일한 디렉토리에서 실행되거나 hard-wired 설치 디렉토리에서 실패한다. 일반적으로 사용하지 않거나 agens 실행 파일을 찾을 수 없다는 오류가 발생하지 않는 한 이 옵션을 사용할 필요가 없다.

`init` 모드에서는 이 옵션을 사용하여 `initdb` 실행 파일의 위치를 지정한다.

-s

--silent

오류 메시지만 인쇄할 수 있다.

-t

--timeout

시작 또는 종료를 기다릴 때 대기할 최대 시간(초)이다. 기본 값은 PGCTLTIMEOUT 환경 변수의 값이거나, 설정되지 않은 경우 60초이다.

-V

--version

`ag_ctl` 버전을 인쇄하고 종료한다.

--revision

agens revision 정보를 보여주고 종료한다.

-w

시작 또는 종료가 완료될 때까지 기다린다. 대기 모드는 종료에 대한 기본 옵션이지만 시작에는 해당되지 않는다. 시작을 기다릴 때, `ag_ctl`은 반복적으로 서버에 연결을 시도한다. 종료를 기다릴 때는 서버가 해당 PID파일을 제거할 때까지 기다린다. 이 옵션을 사용하면 시작시 SSL암호를 입력할 수 있다. `ag_ctl`은 시작 또는 종료의 성공을 바탕으로 종료 코드를 반환한다.

-W

시작 또는 종료가 완료될 때까지 기다리지 않는다. 이는 시작 및 재시작 모드의 기본값이다.

-?

--help

`ag_ctl` 명령 행 인수에 대한 도움말을 표시하고 종료한다.

Options for Windows

-N ***servicename***

등록할 시스템 서비스의 이름이다. 이름은 서비스 이름과 표시된 이름으로 사용된다.

-P ***password***

사용자가 서비스를 시작할 수 있는 암호이다.

-S ***start-type***

등록할 시스템 서비스의 시작 유형이다. 시작 유형은 `auto`, `demand` 또는 이 둘중 하나의 첫번째 문자일 수 있다.

이것이 생략되면 auto 가 사용된다.

-U ***username***

서비스를 시작할 사용자 이름이다. 도메인 사용자의 경우 `DOMAIN\username`형식을 사용한다.

Environment

AGDATA 기본 데이터 디렉터리 위치이다.

Examples

Starting the Server

서버를 시작하려면 다음을 사용한다.

```
$ ag_ctl start
```

서버를 시작하려면 서버가 연결을 수락할 때까지 기다린다.

```
$ ag_ctl -w start
```

포트 5432을 사용하여 서버를 시작하고 fsync없이 실행하려면 다음을 사용한다.

```
$ ag_ctl -o "-F -p 5432" start
```

Stopping the Server

서버를 중지하려면 다음을 사용한다.

```
$ ag_ctl stop
```

`-m`옵션을 사용하면 서버가 종료되는 방법을 제어할 수 있다.

```
$ ag_ctl stop -m fast
```

Restarting the Server

서버를 재시작하는 것은 `ag_ctl`이 이전에 실행된 인스턴스로 전달된 명령줄 옵션을 저장하고 재사용하는 경우를 제외하고 서버를 중지하고 다시 시작하는 것과 동일하다. 가장 간단한 형태로 서버를 재시작하려면 다음과 같이 사용한다.

```
$ ag_ctl restart
```

서버를 재시작하기 위해 서버를 중지하고 재시작하는 것을 기다린다.

```
$ ag_ctl -w restart
```

포트 5432를 사용하여 재시작하려면 재시작 시 fsync를 해제한다.

```
$ ag_ctl -o "-F -p 5432" restart
```

Showing the Server Status

다음은 ag_ctl의 샘플 상태 출력이다.

```
$ ag_ctl status
ag_ctl: server is running (PID: 2823)
/path/to/AgensGraph/bin/postgres
```

이것은 재시작 모드에서 호출될 명령 행이다.

7.2.2 pg_upgrade

pg_upgrade는 데이터 파일에 저장된 데이터를 주요 버전 업그레이드에 필요한 데이터 덤프/재로드 없이 최신 AgensGraph 주 버전으로 업그레이드 할 수 있게 한다. pg_upgrade 새로운 시스템 테이블을 생성하고 이전 사용자 데이터 파일을 재사용함으로써 신속한 업그레이드를 수행한다.
사용법은 아래와 같다.

```
$ pg_upgrade [OPTION]...
```

Options

pg_upgrade 는 다음 명령행 인수를 허용한다.

-b ***bindir***

--old-bindir=***bindir***

이전 AgensGraph bin 디렉토터리 (환경변수 : PGBINOLD)

-B ***bindir***

--new-bindir=***bindir***

새로운 AgensGraph bin 디렉토리 (환경변수 : PGBINNEW)

-c

--check

클러스터만 확인하고 데이터를 변경하지 않는다.

-d ***datadir***

--old-datadir=***datadir***

이전 클러스터 데이터 디렉토리(환경변수: PGDATAOLD)

-D ***datadir***

--new-datadir=***datadir***

새로운 클러스터 데이터 디렉토리(환경변수: PGDATANEW)

-j

--jobs

동시에 사용할 수 있는 processes 또는 threads의 수

-k

--link

새 클러스터에 파일을 복사하는 대신 하드 링크를 사용한다.

-o ***options***

--old-options ***options***

이전 agens 명령에 직접 전달되는 옵션으로 여러 옵션 호출이 추가된다.

-O ***options***

--new-options ***options***

새로운 agens 명령에 직접 전달되는 옵션으로 여러 옵션 호출이 추가된다.

-p ***port***

--old-port=***port***

이전 클러스터 포트 번호(환경변수: PGPORTOLD)

-P ***port***

--new-port=***port***

새로운 클러스터 포트 번호(환경변수: PGPORTNEW)

-r

--retain

성공적으로 완료된 후에도 SQL 및 로그 파일 유지

-U ***username***

--username=***username***

클러스터의 설치 사용자 이름(환경변수: PGUSER)

-v

--verbose

자세한 내부 로깅 사용

-V

--version

버전 정보를 표시한 다음 종료한다.

-?

--help

도움말을 표시 한 다음 종료한다.

Usage

다음은 pg_upgrade를 사용하여 업그레이드를 수행하는 단계이다.

1. 기존의 database cluster를 백업한다.

   ```
   $ mv /user/local/AgensGraph /usr/local/AgensGraph.old
   ```

2. 새로운 AgensGraph 바이너리를 설치한다.

3. 새로운 AgensGraph 클러스터를 초기화 한다.

   ```
   $ initdb
   ```

4. 기존 클러스터 및 신규 클러스터의 서비스를 종료한다.

   ```
   $ ag_ctl -D /opt/AgensGraph/1.2/db_cluster stop
   $ ag_ctl -D /opt/AgensGraph/1.3/db_cluster stop
   ```

5. pg_upgrade를 실행한다.

   ```
   $ pg_upgrade -d oldCluster/data -D newCluster/data -b oldCluster/bin -B newCluster/bin
   ```

   또는

   ```
   $ export PGDATAOLD=oldCluster/data
   $ export PGDATANEW=newCluster/data
   $ export PGBINOLD=oldCluster/bin
   $ export PGBINNEW=newCluster/bin
   $ pg_upgrade
   ```

6. 새로운 AgensGraph를 시작한다.

   ```
   $ ag_ctl start
   ```

7. 업그레이드 후 다음과 작업을 행한다.

- 통계

  옵티마이저 통계는 pg_upgrade에 의해 전송되지 않으므로 업그레이드가 끝날 때 명령을 실행하여 해당 정보를 다시 생성해야 한다. 새 클러스터와 일치하도록 연결 매개 변수를 설정해야 할 수도 있다.

- 이전 클러스터 삭제

  업그레이드에 만족하면 pg_upgrade 완료 시 언급된 스크립트를 실행하여 이전 클러스터의 데이터 디렉토리를 삭제할 수 있다.