Agens SQL 1.0 문서

Agens SQL 1.0 문서

차례

I. Administrator v

1. 서버 설정 및 운용 1
2. 서버 환경 설정 23
3. 클라이언트 인증 79
4. 데이터베이스 role 95
5. 데이터베이스 관리 100
6. 로컬라이제이션 106
7. 정기적인 데이터베이스 관리 작업들 118
8. 백업과 복원 128
9. 고가용성, 부하 분산, 복제 145
10. 복구 환경설정 168
11. 데이터베이스 성능 모니터링 172
12. 디스크 사용량 모니터링 200
13. 안정성 및 Write-Ahead 로그 202

표 목록

1-1. System V IPC 매개변수 6
1-2. SSL 서버 파일 사용 19
2-1. 메시지 심각도 레벨 54
2-2. 단축 옵션 키 77
6-1. Agens SQL 문자 집합 111
6-2. 클라이언트/서버 문자 집합 변환 115
9-1. 고가용성, 부하 분산, 복제 기능 도표 147
11-1. 표준 통계 정보 뷰 174
11-2. pg_stat_activity View 176
11-3. pg_stat_archiver View 178
11-4. pg_stat_bgwriter 뷰 179
11-5. pg_stat_database View 180
11-6. pg_stat_all_tables View 181
11-7. pg_stat_all_indexes View 182
11-8. pg_statio_all_tables View 183
11-9. pg_statio_all_indexes View 183
11-10. pg_statio_all_sequences View 184
11-11. pg_stat_user_functions View 184
11-12. pg_stat_replication View 184
11-13. pg_stat_database_conflicts View 185
11-14. 부가 통계 함수들 186
11-15. 백엔드 단위 통계 함수들 187
11-16. 내장된 DTrace 프로브 189
11-17. 프로브 매개변수에 사용된 정의된 자료형 196

I. Administrator

관리자 안내서는 Agens SQL 데이터베이스 관리자에게 필요한 주제를 다룬다. 소프트웨어 설치 및 셋업, 서버 환경 설정, 사용자와 데이터베이스 관리, 정기적인 관리에 대한 내용이 들어 있다. Agens SQL 서버를 개인 용도로 쓰거나 특히 운영상 사용해야 하는 경우, 이 주제들에 익숙해야 한다. 관리자 안내서는 처음 Agens SQL 사용시 읽으면 좋은 순서로 정리해 놓았다. 각 장의 내용은 독립적이기 때문에 개인의 선호에 따라 읽으면 된다. 관리자 안내서 내용은 주제 단위로 기술 되어 있으며, Agens SQL 데이터베이스 시스템 기본 사용법에 익숙한 독자들이 읽을 수 있다.

1장. 서버 설정 및 운용

이 장에서는 데이터베이스 서버를 설정하고 실행하는 방법과 운영 체제와 상호 작용하는 방법에 대해 다룬다.

1.1. Agens SQL 사용자 계정

외부에서 액세스 가능한 서버 데몬과 마찬가지로 Agens SQL도 별도의 사용자 계정으로 실행하는 것이 좋다. 이 사용자 계정은 서버에서 관리되는 데이터만 소유해야 하며, 다른 데몬과 공유해서는 안 된다. (예를 들면, 사용자 `nobody`를 사용하는 것은 바람직하지 않음.) 그럴 경우 손상된 시스템이 자체 바이너리를 변경할 수 있으므로 이 사용자가 소유한 실행 파일을 실행하는 것은 권장하지 않는다.

Unix 사용자 계정을 시스템에 추가하려면 **useradd** 또는 **adduser** 명령을 찾아본다. 이 설명서에 빈번하게 등장하는 사용자 이름 agens는 설정된 것으로, 원하는 다른 이름을 대신 사용할 수 있다.

1.2. 데이터베이스 클러스터 생성

작업을 하기 전에 디스크의 데이터베이스 저장소 영역을 초기화해야 한다. 이를 *데이터베이스 클러스터*라고 한다. (SQL에서는 카탈로그 클러스터라고 함.) 데이터베이스 클러스터는 실행 중인 데이터베이스 서버의 단일 인스턴스가 관리하는 데이터베이스 컬렉션이다. 초기화 후 데이터베이스 클러스터에는 일명 `agens`라는 데이터베이스가 포함되는데, 이것은 유틸리티 및 사용자, 타사 애플리케이션이 사용하는 기본 데이터베이스이다. 데이터베이스 서버 자체는 `agens` 데이터베이스가 불필요하지만, 다수의 외부 유틸리티 프로그램은 이 데이터베이스가 존재한다는 것을 전제로 한다. 초기화 중에 각 클러스터 내에 생성되는 또 다른 데이터베이스는 `template1`이라고 한다. 이름에서 알 수 있듯이, 이것은 이후에 생성된 데이터베이스의 템플릿으로 사용되며 실제 작업에 사용해서는 안 된다. (클러스터 내에서 데이터베이스로 생성하는 방법은 5장 참조.)

파일 시스템의 관점에서, 데이터베이스 클러스터는 모든 데이터가 저장되는 단일 디렉토리이다. 이것을 *데이터 디렉토리* 또는 *데이터 영역*이라고 한다. 데이터를 어디에 저장할 것인지는 전적으로 사용자의 선택에 달려 있다. `/usr/local/pgsql/data` 또는 `/var/lib/pgsql/data`가 일반적이지만, 필수는 아니다. 데이터베이스 클러스터를 초기화하려면 Agens SQL과 함께 설치된 명령을 사용한다. 데이터베이스 클러스터의 원하는 파일 시스템 위치는 `-D` 옵션으로 나타낼 수 있다. 예를 들면,

```
$ initdb -D /usr/local/pgsql/data
```

앞에서 설명한 대로 Agens SQL 사용자 계정으로 로그인한 상태에서 이 명령을 실행해야 한다.

> 작은 정보: `-D` 옵션 대신 환경 변수 PGDATA를 설정할 수 있다.

또는 다음과 같이 프로그램으로 **initdb**를 실행할 수 있다.

```
$ pg_ctl -D /usr/local/pgsql/data initdb
```

서버를 시작하고 중지할 때 좀 더 직관적인 방법으로써 **pg_ctl**을 사용하면(1.3절참조), **pg_ctl** 하나로 데이터베이스 서버 인스턴스를 관리할 수 있게 된다.

initdb는 지정한 디렉토리가 존재하지 않는 경우에 디렉토리를 생성한다. 디렉토리를 생성하는 권한은 없을 가능성이 높다(당사 조언을 따랐고 권한이 없는 계정을 만든 경우). 그런 경우 root 권한으로 디렉토리를 직접 생성하고 소유자를 Agens SQL 사용자로 변경해야 한다. 다음과 같이 하면 된다.

```
root# mkdir /usr/local/pgsql/data
root# chown agens /usr/local/pgsql/data
root# su agens
agens$ initdb -D /usr/local/pgsql/data
```

initdb는 데이터 디렉토리가 초기화돼 있으면 실행을 거부한다.

데이터 디렉토리에는 데이터베이스의 모든 데이터가 저장되어 있으므로, 무단 접근으로부터 데이터 디렉토리를 보호하는 것이 중요하기 때문에 **initdb**는 Agens SQL 사용자를 제외한 모든 사용자로부터 접근 권한을 해지한다.

단, 디렉토리 내용이 보호 중인 경우 기본 클라이언트 인증 설정은 로컬 사용자의 데이터베이스 연결을 허용하고 로컬 사용자가 데이터베이스 수퍼유저가 되는 것을 허용하기도 한다. 다른 로컬 사용자를 신뢰하지 않을 경우에는 **initdb**의 `-W` 또는 `--pwprompt`, `--pwfile` 옵션 중 하나를 사용하여 데이터베이스 수퍼유저에게 패스워드를 할당하는 것이 좋다. 또한 `-A md5` 또는 `-A password`를 지정하여 기본 `trust` 인증 모드를 사용하지 않거나, 서버를 처음으로 시작하기 전에 **initdb**를 실행한 후 생성된 `pg_hba.conf` 파일을 수정해야 한다. (기타 합리적 접근법에는 `peer` 인증 또는 파일 시스템 권한을 사용하여 연결을 제한하는 방법이 있다. 자세한 내용은 3장 참조.)

initdb에서 데이터베이스 클러스터의 기본 로케일(locale)을 초기화할 수도 있다. 일반적으로는 환경 로케일(locale) 설정을 가져와서, 이를 초기화된 데이터베이스에 적용한다. 데이터베이스에 서로 다른 로케일(locale)을 지정할 수 있다. 자세한 내용은 6.1절에 나와 있다. 특정 데이터베이스 클러스터 내에서 사용되는 기본 정렬 순서는 **initdb**로 설정되며, 서로 다른 정렬 순서로 새로운 데이터베이스를 생성하는 경우 initdb로 생성된 템플릿 데이터베이스에 사용되는 정렬 순서는 삭제 및 재생성하지 않고는 변경할 수 없다. `C` 또는 `POSIX` 이외의 로케일(locale)을 사용하는 경우 성능에도 영향을 미칠 수 있다. 따라서 처음부터 잘 선택 하는 것이 중요하다.

initdb로 데이터베이스 클러스터용 기본 문자 집합 인코딩도 설정한다. 일반적으로, 이것은 로케일(locale) 설정과 동일하게 선택해야 한다. 자세한 내용은 6.3절 참조.

1.2.1. 네트워크 파일 시스템

네트워크 파일 시스템에 데이터베이스 클러스터가 만들 때 여러 가지 설치가 필요한데, NFS를 통해 직접 생성되거나 내부적으로 NFS를 사용하는 NAS(Network Attached Storage) 장치로 생성되기도 한다. Agens SQL이 NFS파일 시스템에 특별한 무언가를 하는 것은 아니며, NFS가 로컬로 연결된 드라이브 (DAS, Direct Attached Storage)처럼 동작한다. 클라이언트 및 서버 NFS가 비표준 의미 체계로 구현될 경우 안정성에 문제가 된다. (http://www.time-travellers.org/shane/papers/NFS_considered_harmful.html 참조) 특히, NFS서버에 대한 쓰기 지연(비동기)은 안정성 문제의 원인이 되므로 이러한 문제를 피하려면 가능한 NFS 파일 시스템을 캐시 없이 동기적으로 마운트해야 한다. 또한 NFS의 소프트 마운트는 권장하지 않는다.(SAN(Storage Area Networks)은 NFS가 아닌 저수준 통신 프로토콜을 사용한다.)

1.3. 데이터베이스 서버 시작

데이터베이스에 액세스하기 전에 데이터베이스 서버를 시작해야 한다. 데이터베이스 서버 프로그램을 **agens**라고 한다. **agens**프로그램은 사용하려는 데이터 위치를 알고 있어야 한다. 이는 `-D` 옵션으로 가능하다. 따라서 서버를 시작하는 가장 손쉬운 방법은 다음과 같이 하는 것이다.

```
$ agens -D /usr/local/pgsql/data
```

이렇게 하면 서버가 포그라운드에서 실행된다. 이는 Agens SQL 사용자 계정으로 로그인한 상태에서 해야 한다. `-D`가 없으면 이름이 환경 변수 PGDATA인 데이터 디렉토리를 서버가 사용하려고 한다. 해당 변수가 제공되지 않으면 실패하게 된다.

보통은 백그라운드에서 **agens**를 시작하는 것이 좋다. 이것의 경우 일반적인 Unix 쉘 구문을 사용한다.

```
$ agens -D /usr/local/pgsql/data >logfile 2>&1 &
```

위와 같이 서버의 stdout 및 stderr 출력을 어딘가에 저장해 놓는 것이 중요하다. 그러면 감사 및 문제 진단 시 도움이 된다. (로그 파일 처리에 대한 자세한 내용은 7.3절 참조)

agens 프로그램에는 다른 커맨드라인 옵션도 많이 있다. 자세한 내용은 agens 참조 페이지 및 아래의 runtime-config 을 참조 바란다.

쉘 구문은 지루하고 따분하다. 따라서 일부 작업을 단순화할 수 있는 래퍼 프로그램 이 제공된다. 예:

```
pg_ctl start -l logfile
```

이것은 서버를 백그라운드에서 시작하고 출력을 지명된 로그 파일로 출력한다. `-D` 옵션은 **agens**에서 사용된 것과 의미가 동일하다. **pg_ctl**으로도 서버를 중지할 수 있다.

보통은, 컴퓨터 부팅 시 데이터베이스 서버도 시작하는 것이 일반적이다. 자동 시작 스크립트는 운영 체제마다 다르다. Agens SQL에서 배포되는 몇 가지 스크립트가 `contrib/start-scripts` 디렉토리에 있다. 하나를 설치하려면 루트 권한이 필요하다.

시스템이 다르면 부팅 시 데몬을 시작하기 위한 규칙(convention)도 달라진다. 다수의 시스템에 `/etc/rc.local` 또는 `/etc/rc.d/rc.local` 파일이 있다. 그 외에는 `init.d` 또는 `rc.d` 디렉토리를 사용한다. 서버는 루트 또는 다른 사용자가 아닌 Agens SQL 사용자 계정으로 실행해야 한다. 그러므로 `su agens -c '...'` 류의 명령을 사용해야 한다. 예를 들면,

```
su agens -c 'pg_ctl start -D /usr/local/pgsql/data -l serverlog'
```

운영 체제별로 특수한 몇 가지 예시는 다음과 같다. (각각의 경우 적절한 설치 디렉토리와 사용자 이름을 사용해야 하며, 여기서는 일반적인 값을 사용한다.)

- FreeBSD의 경우 Agens SQL 소스 배포에서 `contrib/start-scripts/freebsd` 파일을 검토해야 한다.
- OpenBSD에서 다음 라인을 `/etc/rc.local` 파일에 추가해야 한다.

```
if [ -x /usr/local/pgsql/bin/pg_ctl -a -x /usr/local/pgsql/bin/agens ]; then
    su -l agens -c '/usr/local/pgsql/bin/pg_ctl start -s -l /var/Agens SQL/log -D /usr/
    echo -n ' Agens SQL'
fi
```

- Linux 시스템에서는

```
/usr/local/pgsql/bin/pg_ctl start -l logfile -D /usr/local/pgsql/data
```
를 /etc/rc.d/rc.local 또는 /etc/rc.local에 추가하거나 Agens SQL 소스 배포에서 contrib/start-scripts/linux 파일을 검토해야 한다.

- NetBSD에서는 기본 설정에 따라 FreeBSD 또는 Linux 시작 스크립트를 사용해야 한다.
- Solaris에서는 다음 라인이 포함된 /etc/init.d/postgresql 파일을 생성해야 한다.

```
su - agens -c "/usr/local/pgsql/bin/pg_ctl start -l logfile -D /usr/local/pgsql/data"
```
그런 다음, S99Agens SQL로써 /etc/rc3.d에 심볼릭 링크를 생성해야 한다.

서버 실행 중에 PID는 데이터 디렉토리의 postmaster.pid 파일에 저장된다. 이것은 동일한 데이터 디렉토리에서 실행되는 다중 서버 인스턴스를 방지하는 데 사용되고, 서버를 셧다운 하는 데에도 사용될 수 있다.

1.3.1. 서버 시작 실패

서버 시작 실패에는 몇 가지 공통된 원인이 있다. 서버의 로그 파일을 확인하거나 직접 시작해서(표준 출력 또는 표준 에러를 리다이렉트하지 않고) 에러 메시지를 확인해야 한다. 아래는 몇 가지 공통된 에러 메시지를 자세히 설명한다.

```
LOG:  could not bind IPv4 socket: Address already in use
HINT:  Is another postmaster already running on port 5432? If not, wait a few seconds and retry.
FATAL:  could not create TCP/IP listen socket
```

일반적으로 이것은 서버가 이미 실행되고 있는 포트에서 사용자가 다른 서버를 시작하려고 했음을 의미한다. 단, 커널 에러 메시지가 Address already in use가 아니면 다른 문제일 가능성이 있다. 예를 들면, 예약된 포트 번호에서 서버를 시작하려고 하면 다음과 같이 할 것이다.

```
$ agens -p 666
LOG:  could not bind IPv4 socket: Permission denied
HINT:  Is another postmaster already running on port 666? If not, wait a few seconds and retry.
FATAL:  could not create TCP/IP listen socket
```

메시지는 다음과 같이 나타난다.

```
FATAL:  could not create shared memory segment: Invalid argument
DETAIL:  Failed system call was shmget(key=5440001, size=4011376640, 03600).
```

이것은 공유 메모리 크기에 대한 커널 제한이 Agens SQL이 생성하려고 하는 작업 영역보다 작다는 것을 의미하는 것일 수 있다. (이 예시에서 4011376640바이트) 또는 커널에 환경 설정된 System-V-style 공유 메모리가 일절 지원되지 않는다는 것을 의미할 수도 있다. 임시 해결책으로 버퍼 수를 정상보다 작게 해서 서버 시작을 시도해볼 수 있다(shared_buffers). 결국에는 허용된 공유 메모리 크기를 늘리기 위해 사용자는 다시 커널 환경 설정을 할 수도 있다. 또한 동일한 머신에서 다중 서버를 시작하려는 경우 총 요청 공간이 커널 제한을 초과하면 이 메시지가 나타날 수도 있다.

에러가 다음과 같을 수 있다.

```
FATAL:  could not create semaphores: No space left on device
DETAIL:  Failed system call was semget(5440126, 17, 03600).
```

이것은 사용자의 디스크 공간이 소진되었음을 의미하지 않는다. 이것은 System V 세마포어에 대한 커널 수 제한이 생성하려는 Agens SQL의 수보다 작다는 것을 의미한다. 위와 마찬가지로, 허용된 연결 수(max_connections)를 줄여서 서버를 시작함으로써 문제를 해결할 수 있지만, 결국에는 커널 제한을 늘리는 것이 좋다.

"illegal system call" 에러가 나타난 경우 공유 메모리 또는 세마포어가 사용자의 커널에서 일절 지원되지 않는 것일 수 있다. 이런 경우 유일한 방법은 커널을 다시 환경 설정하는 것뿐이다.

System V IPC 기능 환경 설정에 대한 자세한 내용은 1.4.1절에 나와 있다.

1.3.2. 클라이언트 연결 문제

클라이언트 측에서 발생 가능한 에러 조건은 다양하고 애플리케이션에 의존적이지만, 그 중 몇 가지는 서버가 시작되는 것과 직접적인 관련이 있다. 아래 조건 이외는 각 클라이언트 애플리케이션에 맞춰서 문서화되어야 한다.

```
psql: could not connect to server: Connection refused
        Is the server running on host "server.joe.com" and accepting
        TCP/IP connections on port 5432?
```

이것은 일반적인 "I couldn't find a server to talk to" 실패이다. TCP/IP 통신을 시도하면 위와 같이 보인다. TCP/IP 연결이 가능하게 서버를 환경 설정하는 것을 잊어버렸을 때 발생한다.

또, 로컬 서버에 대한 Unix 도메인 소켓 통신을 시도할 때 발생할 수 있다.

```
psql: could not connect to server: No such file or directory
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.5432"?
```

마지막 라인은 클라이언트가 올바른 곳으로 연결을 시도하고 있는지 확인하는 메시지이다. 사실상 실행 중인 서버가 없는 경우 커널 에러 메시지는 보통 `Connection refused` 또는 `No such file or directory` 중 하나이다. (`Connection refused`는 서버가 사용자의 연결 요청을 접수했는데 거부했다는 뜻이 아니다. 해당 사례는 3.4절에 표시된 대로 다른 메시지가 나타난다.) `Connection timed out` 같은 에러 메시지는 네트워크 연결성 부족 문제를 뜻하는 것일 수 있다.

1.4. 커널 리소스 관리

Agens SQL은 특히 서버의 사본들을 동일한 시스템에서 실행 하는 경우 또는 대규모 설치 하는 경우, 운영 체제 자원들을 한계치까지 쓰기도 한다. 이 절에서는 Agens SQL가 사용하는 커널 자원 및 커널 자원 소비와 관련된 문제 해결 단계를 다룬다.

1.4.1. 공유 메모리 및 세마포어

공유 메모리 및 세마포어는 통칭 "System V IPC"라고 한다. (Agens SQL과 무관한 메시지 큐와 함께) Windows 외에 Agens SQL이 이러한 기능을 자체적으로 제공하는 경우 Agens SQL을 실행하려면 이 기능이 요구된다.

이 기능이 없으면 서버 시작 시 잘못된 시스템 호출 에러가 발생한다. 이 경우 커널을 다시 환경 설정하는 것 외에는 대안이 없다. Agens SQL은 커널 없이 작동되지 않는다. 이 상황은 최신 운영 체제에서는 거의 일어나지 않는다.

Agens SQL이 하드 IPC제한을 초과한 경우 서버는 시작을 거부하고 문제와 조치를 설명하는 에러 메시지를 남긴다. (1.3.1절 참조) 관련 커널 매개변수의 이름은 각종 시스템 간에 동일하며, 표 1-1에 대략적인 내용이 나와 있다. 단, 매개변수 설정 방법은 다를 수 있다. 일부 플랫폼별 설정방법은 아래에 나와 있다.

> 참고**:** Agens SQL 9.3 이전에는 서버 시작 시 훨씬 더 많은 System V 공유 메모리가 필요했다. 오래된 버전에서 서버를 실행할 경우 문서에서 서버 버전을 참고하기 바란다.

표 **1-1. System V IPC** 매개변수

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 최대 수	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_workers + 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)

Agens SQL은 서버 사본별로 System V 공유 메모리 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.

Agens SQL의 경우, 공유 메모리 세그먼트의 최소 크기(SHMMIN)는 많아야 약 32바이트에 불

과하기 때문에(대개 1) 문제의 원인이 될 가능성은 낮다. 시스템 차원(system-wide)의 세그먼트 최대 수(SHMMNI) 또는 프로세스당 최대 수(SHMSEG)는 시스템이 영(0)으로 설정해 놓지 않는 한 문제의 원인이 될 가능성은 낮다.

Agens SQL은 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 Agens SQL의 경우 최소 17이어야 한다.

SEMMNU 및 SEMUME 같은 "semaphore undo"와 관련된 기타 설정은 Agens SQL에 영향을 미치지 않는다.

AIX

모든 메모리가 공유 메모리로 사용되도록 설정되므로, 적어도 5.1 버전에서는 SHMMAX같은 매개변수를 설정할 필요가 없다. SHMMAX는 DB/2 같은 다른 데이터베이스에서 일반적으로 사용되는 설정의 한 종류이다.

그러나, 파일 크기(`fsize`)와 파일 수(`nofiles`)의 기본 하드 제한이 너무 낮으므로 전역 **ulimit** 정보를 `/etc/security/limits`에서 변경해야 될 수도 있다.

FreeBSD

기본 설정은 **sysctl** 또는 **loader** 인터페이스를 사용하여 변경할 수 있다. 아래 매개변수들은 **sysctl**을 사용하여 설정할 수 있다.

```
# sysctl kern.ipc.shmall=32768
# sysctl kern.ipc.shmmax=134217728
```

재부팅 시에 이 설정을 유지하려면 `/etc/sysctl.conf`를 수정해야 한다.

이러한 세마포어 관련 설정은 **sysctl**을 사용한 경우, 읽기 전용이지만 `/boot/loader.conf`에서 다르게 설정할 수 있다.

```
kern.ipc.semmni=256
kern.ipc.semmns=512
kern.ipc.semmnu=256
```

수정된 설정이 적용되려면 재부팅이 필요하다. (참고: FreeBSD는 SEMMAP를 사용하지 않는다. 오래된 버전은 SEMMAP를 적용하지만 `kern.ipc.semmap`설정은 무시하고, 새 버전은 둘 다 무시한다.)

사용자는 커널 환경 설정을 하여 공유 메모리를 RAM에서 잠그고 스왑되지 않게 할 수 있다. 이것은 `kern.ipc.shm_use_phys`를 설정해서 **sysctl**를 사용하면 된다.

sysctl의 `security.jail.sysvipc_allowed`를 활성화하여 FreeBSD jail에서 실행할 시, 서로 다른 jail에서 실행 중인 postmaster들은 각기 다른 시스템 사용자로부터 실행되야 한다. 루트 사용자가 아닌 경우 서로 다른 jail에서 사용자가 공유 메모리 또는 세마포어를

간섭하지 못하게 하고, PostgreSQL IPC 클린업 코드가 제대로 작동되어 보안에 이점이 있다. (FreeBSD 6.0 이상에서 IPC 클린업 코드는 서로 다른 jail에서 동일한 포트로 postmaster를 실행하는 것을 막기 때문에 다른 jail의 프로세스를 인식하지 못한다.)

FreeBSD 4.0 이전 버전은 OpenBSD처럼 작동된다(아래 참조).

NetBSD

NetBSD5.0 이상에서, IPC 매개변수는 **sysctl**을 사용하여 조절 가능하다. 예를 들면 다음과 같다.

```
$ sysctl -w kern.ipc.shmmax=16777216
```

재부팅 시에도 설정을 유지하려면 `/etc/sysctl.conf`를 수정해야 한다.

사용자는 커널 환경 설정을 하여 공유 메모리를 RAM에서 잠그고 스왑되지 않게 할 수 있다. 이것은 `kern.ipc.shm_use_phys`를 설정해서 **sysctl**을 사용하면 가능하다.

NetBSD 5.0 이전 버전에서 `option`이 아니라 `options`란 키워드로 설정해야 하는 매개변수 외에는 OpenBSD (아래 참조)처럼 작동된다.

OpenBSD

`SYSVSHM` 옵션 및 `SYSVSEM`은 커널이 컴파일된 경우에 활성화되어야 한다(기본으로 설정돼 있음). 공유 메모리의 최대 크기는 옵션 `SHMMAXPGS` (페이지 단위)에 의해 결정된다. 다음 예시는 다양한 매개변수 설정 방법을 보여준다.

```
option        SYSVSHM
option        SHMMAXPGS=4096
option        SHMSEG=256

option        SYSVSEM
option        SEMMNI=256
option        SEMMNS=512
option        SEMMNU=256
option        SEMMAP=256
```

사용자는 커널 환경 설정을 하여 공유 메모리를 RAM에서 잠그고 스왑되지 않게 할 수 있다. 이것은 **sysctl** 설정 `kern.ipc.shm_use_phys`를 사용하면 가능하다.

HP-UX

기본 설정만으로도 정상적으로 설치 가능하다. HP-UX 10에서 `SEMMNS`의 출고시 기본 설정은 128인데, 거대 데이터베이스 사이트에는 너무 작을 수 있다.

IPC 매개변수는 커널 환경 설정→환경 설정 변수 아래의 시스템 관리 매니저 (SAM)에서 설정할 수 있다. 완료 시 새 커널 생성하기를 선택해야 한다.

리눅스

최대 세그먼트 크기 기본값은 32 MB이며, 최대 총 크기 기본값은 2097152 페이지이다. “huge pages”를 이용한 특수한 커널 환경 설정일 때 외에 페이지는 거의 항상 4096바이트이다(확인하려면 `getconf PAGE_SIZE` 사용).

공유 메모리 크기 설정은 **sysctl** 인터페이스를 통해 변경 가능하다. 예를 들어, 16GB로 설정하려면 아래처럼 입력한다.

```
$ sysctl -w kernel.shmmax=17179869184
$ sysctl -w kernel.shmall=4194304
```

또, `/etc/sysctl.conf` 파일로 재부팅 시에도 이 설정을 보존할 수 있다. 이는 매우 바람직한 설정 방법이다.

오래된 버전에는 **sysctl** 프로그램이 없을 수도 있지만 /proc 파일 시스템을 처리하여 동일하게 변경할 수 있다.

```
$ echo 17179869184 >/proc/sys/kernel/shmmax
$ echo 4194304 >/proc/sys/kernel/shmall
```

남은 기본 설정 값은 아주 넉넉한 크기로 지정돼 있어, 굳이 변경할 필요가 없다.

OS X

OS X에서 공유 메모리를 환경 설정하는 방법은 다음과 같은 변수 및 값이 포함된 /etc/sysctl.conf 파일을 생성하는 것이다.

```
kern.sysv.shmmax=4194304
kern.sysv.shmmin=1
kern.sysv.shmmni=32
kern.sysv.shmseg=8
kern.sysv.shmall=1024
```

일부 OS X 버전에서는 전체 **5**개의 공유 메모리 매개변수를 /etc/sysctl.conf에 설정해야 한다. 그렇게 하지 않으면 설정이 무시된다.

OS X 최근 버전에서는 SHMMAX 설정 값이 정확히 4096의 배수가 아니면 무시된다.

SHMALL은 이 플랫폼에서 4 kB로 적용된다.

OS X 구 버전에서는 공유 메모리 매개변수에 대한 변경 내용이 적용되려면 재부팅 해야 한다. 10.5는 현재, sysctl을 사용하여 SHMMNI를 제외한 모든 변수가 변경 가능하다. 그러나 재부팅 시 값이 유지될 수 있도록 /etc/sysctl.conf를 통해 원하는 값을 설정하는 것이 최선의 방법이다.

/etc/sysctl.conf 파일은 OS X 10.3.9 버전 이상에서만 유효하다. 이전 10.3.x 버전을 실행 중인 경우, /etc/rc 파일을 편집하여 다음 명령으로 값을 변경해야 한다.

```
sysctl -w kern.sysv.shmmax
sysctl -w kern.sysv.shmmin
sysctl -w kern.sysv.shmmni
sysctl -w kern.sysv.shmseg
sysctl -w kern.sysv.shmall
```

/etc/rc는 일반적으로 OS X 시스템 업데이트 시 덮어쓰기 되므로 업데이트가 있을 때마다 사용자는 위 편집을 반복해야 한다.

OS X 10.2 이전 버전에서는 /System/Library/StartupItems/SystemTuning/SystemTuning 파일에서 이 명령을 편집해야 한다.

SCO 오픈서버

기본 환경 설정에서 세그먼트당 512 kB의 공유 메모리만 허용된다. 설정을 변경하려면 먼저 /etc/conf/cf.d디렉토리로 이동해야 한다. SHMMAX의 현재 값을 표시하려면 다음을 실행한다.

```
./configure -y SHMMAX
```

SHMMAX에 새 값을 설정하려면 다음을 실행한다.

```
./configure SHMMAX=value
```

여기서 *value*는 사용하려는 새 값이다(바이트 단위). SHMMAX를 설정한 후에는 커널을 리빌드한다.

```
./link_unix
```

그런 다음 재부팅한다.

Solaris 2.6 에서 2.9 까지(Solaris 6 ~ Solaris 9)

해당 설정은 `/etc/system`에서 변경 가능하다. 예를 들면 다음과 같다.

```
set shmsys:shminfo_shmmax=0x2000000
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=256
set shmsys:shminfo_shmseg=256

set semsys:seminfo_semmap=256
set semsys:seminfo_semmni=512
set semsys:seminfo_semmns=512
set semsys:seminfo_semmsl=32
```

변경 내용을 적용하려면 재부팅이 필요하다. 이전 Solaris 버전에서 공유 메모리에 대한 내용은 http://sunsite.uakom.sk/sunworldonline/swol-09-1997/swol-09-insidesolaris.html 을 참조 바란다.

Solaris 2.10 (Solaris 10) 이상
OpenSolaris

Solaris 10 이상 및 OpenSolaris에서 기본 공유 메모리 및 세마포어 설정은 대부분의 Agens SQL 애플리케이션에서는 충분하다. 이제 Solaris는 시스템 RAM의 1/4을 `SHMMAX` 기본값으로 설정한다. 이 설정을 조정하려면 `agens` 사용자에 대한 프로젝트 설정을 사용해야 한다. 예를 들면, `root`로 다음을 실행한다.

```
projadd -c "Agens SQL DB User" -K "project.max-shm-memory=(privileged,8GB,deny)" -U a
```

위 명령은 `user.agens` 프로젝트를 추가하고 `agens` 사용자에 대한 공유 메모리 최대값을 8GB로 설정한다. 다음 사용자 로그인 시 적용되거나 Agens SQL 재시작 시 적용된다(리로드 아님). 위 명령은 Agens SQL이 `agens` 그룹의 `agens` 사용자로 실행되는 것으로 간주한다. (서버는 재부팅할 필요 없다.)

데이터베이스 서버가 다수 연결된 경우, 커널을 다음과 같이 설정할 것을 권한다.

```
project.max-shm-ids=(priv,32768,deny)
project.max-sem-ids=(priv,4096,deny)
project.max-msg-ids=(priv,4096,deny)
```

또, zone 내에서 Agens SQL을 실행할 경우, zone의 자원 사용 제한도 올려야 할 필요가 있다. `projects` 및 **prctl**에 대한 자세한 내용은 *시스템 관리자 가이드*의 "2장: 프로젝트와 태스크"를 참조 바란다.

UnixWare

UnixWare 7에서 공유 메모리 세그먼트의 최대 크기는 기본적으로 512 kB이다. `SHMMAX`의 현재 값을 보고 싶으면 다음을 입력하라.

```
/etc/conf/bin/idtune -g SHMMAX
```

이것은 현재값 및 기본값, 최소값, 최대값을 보여준다. `SHMMAX`에 새로운 값을 설정하려면 다음을 실행한다.

```
/etc/conf/bin/idtune SHMMAX 새로운 값
```

여기서 *새로운 값*은 사용하고자 하는 바이트 단위의 값이다. `SHMMAX`를 설정한 후에는 커널을 리빌드 하라.

```
/etc/conf/bin/idbuild -B
```

그리고 재부팅하라.

1.4.2. 자원 제한

Unix 계열의 운영 체제는 사용자의 Agens SQL 서버에도 영향을 미칠 수 있는 자원 제한 형태가 다양하다. 그 중 사용자별 프로세스 수, 프로세스당 개방 파일 수, 각 프로세스에서 사용 가능한 메모리 양에 대한 제한이 특히 중요하다. 이러한 제한들은 각각 "하드" 및 "소프트" 제한이 있다. 소프트 제한이 실제로 적용되는 것이고, 사용자가 하드 제한까지 증가 시킬 수 있다. 하드 제한은 root 사용자만 변경할 수 있다. 시스템 호출 `setrlimit`는 이 매개변수의 설정을 담당한다. 쉘에 내장된 명령어 **ulimit**(Bourne 쉘) 또는 **limit** (csh)는 커맨드 라인에서 자원 제한을 제어하는 데 사용된다. BSD 계열 시스템에서 로그인 시 `/etc/login.conf` 파일은 다양한 자원 제한 설정을 제어한다. 자세한 내용은 운영 체제 문서를 참조 바란다. 관련 매개변수는 `maxproc` 및 `openfiles`, `datasize` 이 있다. 예제는 아래와 같다.

```
default:\
...
        :datasize-cur=256M:\
        :maxproc-cur=256:\
        :openfiles-cur=256:\
...
```

(`-cur`는 소프트 제한이다. 하드 제한을 설정하려면 `-max`를 덧붙인다.)

커널은 일부 자원에 대해 시스템 차원(system-wide)의 제한을 가질 수 있다.

- Linux의 `/proc/sys/fs/file-max`는 커널이 지원하는 오픈 파일의 최대 수를 결정한다. 다른 수를 적거나 `/etc/sysctl.conf`에 값을 추가하면 변경된다. 프로세스당 최대 제한 파일 개수는 커널이 컴파일되는 시점에 적용된다. 자세한 내용은 `/usr/src/linux/Documentation/proc.txt`를 참조 바란다.

Agens SQL 서버는 연결당 프로세스 1개를 사용하므로 최소한 연결된 프로세스 개수 이상 지정해야 한다. 이것은 머신 1대에 여러 개의 서버를 실행하는 경우에 중요하다.

오픈 파일에 대한 기본적인 제한은 시스템 자원를 부적절하게 분할하지 않기 위해서, 여러 사용자가 머신에 공존할 수 있도록 "대체로 용인되는" 값으로 설정된다. 필요에 따라 머신 1대에서 여러 개의 서버를 실행할 수 있는데, 이 중 특정 서버의 제한만 변경할 수도 있다.

일부 시스템에서는 프로세스들이 여러 개의 파일들을 열 수 있게 한다. 그러면 몇 개의 프로세스만 실행해도 시스템 차원(system-wide)의 제한이 쉽게 초과된다. 이러한 상황이 발생할 경우, 시스템 차원(system-wide)의 제한을 변경하고 싶지 않으면 Agens SQL의 max_files_per_process 환경 설정 매개변수를 설정하여 오픈 파일 개수를 제한할 수 있다.

1.4.3. 리눅스 메모리 오버커밋

리눅스 2.4 이상에서의 기본적인 가상 메모리 동작은 Agens SQL의 경우 최적화 되어 있지 않다. 커널이 메모리 오버커밋을 이행하는 방식 때문에, Agens SQL 또는 다른 프로세스의 메모리 수요가 시스템의 가상 메모리 소진의 원인이 되는 경우, 커널은 Agens SQL postmaster(마스터 서버 프로세스)를 종료해야 한다.

이 경우가 발생하면 다음과 같은 커널 메시지가 나타난다(해당 메시지를 찾아 보려면 시스템 문서 및 환경 설정 참조).

```
Out of Memory: Killed process 12345 (agens).
```

이것은 agens 프로세스가 메모리 압박 때문에 종료되었음을 뜻한다. 기존 데이터베이스 연결이 정상 작동되더라도 새로운 연결은 수락되지 않는다. 복구하려면 Agens SQL을 재시작해야 한다.

이 문제를 방지하는 방법 중 하나는 다른 프로세스 때문에 머신의 메모리가 소진되지 않을 것이 확실한 머신에서 Agens SQL을 실행하는 것이다. 실제 메모리와 스왑 공간이 소진된 경우에만 메모리 부족(OOM) 킬러가 호출되기 때문에 메모리에 여유가 없는 경우에는 운영 체제의 스왑 공간을 늘리면 문제를 방지하는 데 도움이 된다.

Agens SQL 자체가 메모리 부족의 원인인 경우 환경 설정을 변경하면 문제를 방지할 수 있다. 경우에 따라 메모리 관련 환경 설정 매개변수, 특히 shared_buffers 및 work_mem을 줄이는 것이 도움이 된다. 그 외에는 데이터베이스 서버 자체로의 연결을 너무 많이 허용하는 것이 문제의 원인일 수 있다. 대체로, max_connections를 줄이는 대신 외부 연결 풀링 소프트웨어를 이용하는 것이 좋다.

리눅스 2.6 이상에서 커널의 동작을 수정해서 메모리 "오버커밋"을 방지할 수 있다. 이 설정으로 OOM killer (http://lwn.net/Articles/104179/)의 호출이 전적으로 방지되는 않지만 가능성은 확연히 줄어든다. **sysctl**을 통해 엄격한 오버커밋 모드를 선택하면 된다.

```
sysctl -w vm.overcommit_memory=2
```

위 동작을 /etc/sysctl.conf에 엔트리로 입력해도 된다. vm.overcommit_ratio을 수정하고 싶으면, 커널 문서 파일 Documentation/vm/overcommit-accounting을 참조 바란다.

vm.overcommit_memory를 사용하는 또다른 방법은 postmaster 프로세스의 프로세스 특정 oom_score_adj 값을 -1000으로 설정하는 것이다. 그렇게 함으로써 OOM 킬러의 타겟이 되는 것을 면할 수 있다. 이렇게 하는 가장 간단한 방법은 아래와 같다.

```
echo -1000 > /proc/self/oom_score_adj
```

postmaster를 호출하기 전에 상기 명령을 postmaster의 시작 스크립트에서 실행하는 것이다. 이 액션은 root로 실행하지 않으면 아무런 효력이 없으므로 root권한 소유의 시작 스크립트를 이용하는 것이 가장 간단하다. CPPFLAGS에 추가된 -DLINUX_OOM_SCORE_ADJ=0로 Agens SQL을 빌드할 수도 있다. 이는 postmaster 자식 프로세스를 일반 oom_score_adj 값인 0으로 실행해서 OOM 킬러가 필요시 사용될 수 있다.

이전 리눅스 커널은 /proc/self/oom_score_adj의 이전 버전인 /proc/self/oom_adj을 같은 기능으로 제공한다. 비활성 시 설정하는 값이 -1000이 아니라 -17인 것 외에는 동일하게 작동한다. Agens SQL의 빌드 플래그는 -DLINUX_OOM_ADJ=0이다.

> 참고: 일부 공급업체의 Linux 2.4 커널은 2.6 오버커밋 **sysctl** 매개변수의 초기 소스 코드를 가지고 있는 것으로 알려져 있다. 이전 소스 코드로 2.4 커널에서 vm.overcommit_memory를 2로 설정하는 것은 안 좋다. 실제 커널 소스 코드를 확인하여(mm/mmap.c 파일에서 vm_enough_memory 참조) 2.4 설치 시 사용자 커널에서 어떤 소스를 사용하는지 확인하는 것이 좋다. overcommit-accounting 문서 파일이 존재한다고 해서 오버커밋이 지원된다고 생각해서는 안 된다. 의심스러울 경우는 커널 전문가 또는 커널 공급업체에게 문의 하라.

1.4.4. 리눅스 huge pages

huge pages는 Agens SQL 처럼 서로 인접해 있는 거대한 메모리를 사용 시 오버헤드를 줄여준다. Agens SQL에서 이 huge pages을 활성화하려면 CONFIG_HUGETLB_PAGE=y 및 CONFIG_HUGETLBFS=y를 사용하는 커널이 필요하다. 시스템 설정 vm.nr_hugepages도 튜닝

해야 한다. 필요한 huge pages 수를 추정하려면 huge pages를 활성화하지 않고 Agens SQL을 시작해서 proc 파일 시스템에서 VmPeak 값을 확인해야 한다.

```
$ head -1 /path/to/data/directory/postmaster.pid
4170
$ grep ^VmPeak /proc/4170/status
VmPeak:  6490428 kB
```

6490428 / 2048(이 경우 PAGE_SIZE는 2MB)는 대략 3169.154 이므로 huge pages는 최소 3170가 필요하다.

```
$ sysctl -w vm.nr_hugepages=3170
```

가끔씩 커널이 huge pages를 할당하지 못하는 경우, 해당 명령을 반복하거나 재부팅을 해야 한다. 재부팅 시 이 설정을 유지하려면 /etc/sysctl.conf에 엔트리를 추가해야 한다.

Agens SQL는 기본적으로 huge pages를 가능하면 사용하는 것이고, 실패 시 정상 페이지로 폴백(fallback)하는 것이다. huge pages를 강제로 사용하려면 huge_pages를 on으로 설정하면 된다. huge pages가 충분하지 않으면 Agens SQL을 시작하지 못할 수도 있다.

리눅스 huge pages 기능에 대한 자세한 설명은
https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt를 읽어 보기 바란다.

1.5. 서버 셧다운

데이터베이스 서버를 셧다운 하는 방법에는 몇 가지가 있다. 사용자는 마스터 **postgres** 프로세스에 서로 다른 신호를 전송하여 셧다운 유형을 제어할 수 있다.

SIGTERM

이것은 *스마트 셧다운* 모드이다. SIGTERM을 수신한 후, 서버는 새로운 연결을 허용하지 않지만, 기존 세션은 정상적인 종료를 허용한다. 모든 세션이 종료된 후에만 셧다운된다. 서버가 온라인 백업 모드인 경우 온라인 백업 모드가 더 이상 작동하지 않을 때까지 조금 더 대기한다. 백업 모드가 작동되는 중 새로운 연결은 슈퍼유저에게만 계속 허용된다(슈퍼유저가 온라인 백업 모드를 종료할 수 있게). 서버가 복구 중인 상태에서 스마트 셧다운이 요청되면, 모든 정규 세션이 종료된 후에만 복구 및 스트리밍 리플리케이션이 중단된다.

SIGINT

이것은 *빠른 셧다운* 모드이다. 서버는 새로운 연결을 허용하지 않고 모든 기존 서버 프로세스에 SIGTERM을 전송하는데, 이로 인해 현재 트랜잭션이 중단되고 즉시 종료된다. 그런 다음, 모든 서버 프로세스가 종료될 때까지 기다렸다가 최종적으로 셧다운된다. 서버가 온라인 백업 모드인 경우에는 백업 모드가 종료된다.

SIGQUIT

이것은 *즉시 셧다운* 모드이다. 서버는 SIGQUIT를 모든 자식 프로세스에 전송하고 자식 프로세스가 종료될 때까지 대기한다. 5초 이내에 종료되지 않는 프로세스에는 마스터 **postgres** 프로세스가 SIGKILL을 전송하여 더 이상 대기하지 않고 종료된다. 이것은 다음 시작 시 복구로 이어진다(WAL 로그 리플레이에 의해). 이것은 비상 시에만 권장한다.

pg_ctl 프로그램은 이러한 서버 셧다운 신호를 전송하기 위한 편리한 인터페이스를 제공한다. Windows가 아닌 시스템에서 **kill**을 사용하여 직접 신호를 전송할 수도 있다. **postgres** 프로세스의 PID는 **ps** 프로그램을 사용하여 찾거나 데이터 디렉토리 내 `postmaster.pid` 파일에서 찾을 수 있다. 예를 들어, 빠른 셧다운을 하려면 아래와 같은 명령어로 가능하다.

```
$ kill -INT `head -1 /usr/local/pgsql/data/postmaster.pid`
```

> 중요: 서버 셧다운 시 SIGKILL은 사용하지 않는 것이 최선이다. 이를 사용할 경우 서버가 공유 메모리와 세마포어를 해제하지 않아서, 새 서버를 시작하기 전에 수동으로 해제를 해야 할 수 있다. 또한 SIGKILL은 종속 프로세스로 신호를 전달하지 않고 **postgres**를 kill하므로 각 종속 프로세스를 직접 kill해야 한다.

한 세션을 종료하려면 `pg_terminate_backend()`를 사용하거나 세션의 자식 프로세스에 SIGTERM 신호를 전송해야 한다.

1.6. Agens SQL 클러스터 업그레이드

이 절에서는 Agens SQL 배포판의 데이터베이스 데이터를 새롭게 업그레이드하는 방법을 다룬다.

Agens SQL 메이저 버전은 버전 번호의 첫 두 자리인데, 예를 들면 8.4이다. Agens SQL 마이너 버전은 버전 번호의 세 번째 자릿수부터이다. 예를 들면, 8.4.2는 8.4의 두 번째 부 배포판이다. 마이너 배포판은 내부 스토리지 형식을 절대 변경하지 않으며 메이저 버전 번호가 동일하면 이전 및 이후 마이너 배포판와 항상 호환된다. 예를 들면, 8.4.2는 8.4, 8.4.1 및 8.4.6과 호환된다. 호환 버전 간에 업데이트를 하려면 서버를 종료한 상태에서 실행 파일을 교체하고 서버를 시작하면 된다. 데이터 디렉토리는 변경되지 않고 유지된다.

Agens SQL의 메이저 배포판의 경우, 내부 데이터 스토리지 형식이 변경되므로 업그레이드가 복잡하다. 데이터를 새로운 메이저 버전으로 옮기는 방법은 데이터베이스를 덤프하고 다시 불러오는 것인데 느릴 수도 있다. 더 빠른 방법은 pgupgrade이다. 복제 방법도 아래에 언급된 바와 같이 사용 가능하다.

새 메이저 버전에서 사용자에게 가시적으로도 호환되지 않는 기능이 있기 때문에, 애플리케이션 프로그래밍을 변경해야 한다. 이 비호환 기능에 대한 내용은 릴리즈 노트에 나와 있다. "마이그레이션" 절을 특히 주의 깊게 보아야 한다. 몇 가지의 메이저 버전에 걸쳐 업그레이드하는 경우 각 중간 버전에 대한 릴리즈 노트를 읽어 보아야 한다.

세심한 사용자라면 새 버전으로 완전히 넘어가기 전에 새 버전에서 클라이언트 애플리케이션을 테스트해보고 싶을 것이다. 그러므로, 이전 버전과 새 버전의 동시 설치를 설정하는 것을 권장한다. Agens SQL 메이저 업그레이드를 테스트할 때는 다음과 같이 변경 가능성이 있는 카테고리를 고려해야 한다.

관리

> 서버를 모니터링 및 관리하기 위해 관리자가 사용할 수 있는 기능이 메이저 배포판에서 주로 변경 및 개선된다.

SQL

> 일반적으로 새 SQL 명령이 여기에 포함되고, 특별한 언급이 릴리즈 노트에 있지 않는 한 동작하지 않는 변경은 없다.

라이브러리 API

릴리즈 노트에 특별한 언급이 없으면, libpq 같은 전형적인 라이브러리만 새로운 기능을 추가한다.

시스템 카탈로그

시스템 카탈로그를 변경하면 보통 데이터베이스 관리 도구에만 영향을 미친다.

서버 C 언어 API

이것은 C 프로그래밍 언어로 작성된 백엔드 함수 API 변경과 관련이 있다. 이 변경은 서버 내 백엔드 함수를 참조하는 코드에 영향을 미친다.

1.6.1. pg_dumpall을 통한 데이터 업그레이드

업그레이드 방법 중 하나는 Agens SQL의 메이저 버전에서 데이터를 덤프하고 다른 버전에서 다시 불러오는 것이다. 이렇게 하려면 pg_dumpall 같은 논리적 백업 툴을 사용해야 한다. 파일 시스템 레벨 백업 방법은 작동되지 않는다. (호환되지 않는 Agens SQL 버전에서는 데이터 디렉토리를 사용하지 못하도록 하는 검사가 있기 때문에, 데이터 디렉토리에서 잘못된 서버 버전을 시작하려고 해도 큰 위험은 없다.)

이 프로그램에서 개선된 기능을 활용하려면 새 버전인 Agens SQL에서 pg_dump와 pg_dumpall 프로그램을 사용하는 것이 바람직하다. 현재 덤프 프로그램 버전은 과거 모든 서버 버전부터 7.0까지의 데이터를 읽을 수 있다.

이 프로그램을 사용하려면 `/usr/local/pgsql`에 Agens SQL이 설치돼 있고, 데이터 영역이 `/usr/local/pgsql/data` 안에 있어야 하므로, 적절히 경로를 바꾸어야 한다.

1. 백업 시 데이터베이스가 업데이트하고 있지 않은지 확인해야 한다. 이는 백업의 무결성에는 영향을 미치지 않지만 변경된 데이터는 당연히 백업에 포함되지 않는다. 필요 시 `/usr/local/pgsql/data/pg_hba.conf` 파일(또는 동등한 파일)에서 권한을 편집하여 사용자 본인을 제외한 모든 사람의 액세스를 차단해야 한다. 액세스 제어에 대한 자세한 내용은 3장을 참조 바란다.

   데이터베이스 설치를 백업하려면 다음을 입력한다.

   ```
   pg_dumpall > outputfile
   ```

   백업을 하기 위해 현재 실행 중인 버전에서 pg_dumpall 명령을 선택할 수 있다. 자세한 내용은 8.1.2절을 참조 바란다. 최선의 방법은 버그 수정 기능이 있고 이전 버전보다 개선된 Agens SQL 9.4.1에서 pg_dumpall 명령을 실행하는 것이다. 새 버전을 이전 버전과 병행 설치할 생각이면 이 방법이 좋다. 설치를 정상적으로 완료한 후에 데이터를 전송할 수 있는데, 이렇게 하면 다운타임도 줄어든다.

2. 이전 서버 셧다운하기

   ```
   pg_ctl stop
   ```

   부팅 시에 Agens SQL이 시작되는 시스템에서 동일한 작업을 수행하는 파일이 있을 수 있다. 예를 들면, Red Hat Linux 시스템에서도 이러한 동작을 찾아볼 수 있다.

   ```
   /etc/rc.d/init.d/Agens SQL stop
   ```

   서버 시작 및 중단에 대한 내용은 1장을 참조 바란다.

3. 백업으로부터 복구하는 경우 버전이 명시된 것이 아니라면, 이전 설치 디렉토리의 이름을 변경하거나 디렉토리를 삭제해야 한다. 문제가 발생해서 되돌아 가야 할 때를 대비해서 디렉토리를 삭제하는 것보다는 이름을 변경하는 것이 낫다. 디렉토리는 디스크 공간의 상당량을 차지한다. 디렉토리 이름을 변경하려면 다음과 같은 명령을 사용하라.

   ```
   mv /usr/local/pgsql /usr/local/pgsql.old
   ```

(디렉토리를 단일 유닛으로 이동시켜 관련 경로가 바뀌지 않게 해야 한다.)

4. Agens SQL 새 버전은 install-procedure.에 요약된 대로 설치한다.
5. 필요 시 데이터베이스 클러스터를 새로 생성한다. 특수한 데이터베이스 사용자 계정으로(업그레이드 중이면 기존 계정으로) 로그인해서 이 명령을 실행해야 한다.

   /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
6. 이전 `pg_hba.conf` 및 모든 `Agens SQL.conf` 수정 내용을 복원한다.
7. 특수 데이터베이스 사용자 계정으로 데이터베이스 서버를 다시 시작한다.

   /usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data
8. 마지막으로 다음 명령으로 데이터를 복원한다.

   /usr/local/pgsql/bin/psql -d postgres -f *outputfile*
   이때 새 psql을 사용한다.

새 서버를 다른 디렉토리에 설치하고 이전 및 새 서버를 서로 다른 포트에서 병렬 실행하면 다운타임을 최소화할 수 있다. 그런 다음, 사용자는 데이터를 전송하기 위해 다음과 같은 명령어를 사용할 수 있다.

```
pg_dumpall -p 5432 | psql -d postgres -p 5433
```

이 명령으로 데이터가 전송된다.

1.6.2. pg_upgrade를 통한 데이터 업그레이드

pgupgrade 모듈은 메이저 Agens SQL 버전에서 다른 버전으로 현재 위치에 마이그레이션되는 설치를 허용한다. 특별히 `--link` 모드를 사용하면 몇 분 이내에 업그레이드가 가능하다. 이것은 위의 pg_dumpall과 유사한 단계가 필요하다 (예: 서버 시작/중지, initdb 실행). pg_upgrade문서에는 필수 단계가 간략하게 나와 있다.

1.6.3. 복제를 통한 데이터 업그레이드

Slony 같은 특정한 복제 방법을 사용하여 Agens SQL 업그레이드 버전의 대기 서버를 생성할 수도 있다. 이것은 Slony가 메이저 Agens SQL 버전 간 복제를 지원하기 때문이다. 대기 서버는 동일한 컴퓨터 또는 다른 컴퓨터에 있는 서버이다. (Agens SQL 구 버전 실행 시)일단 마스터 서버와 동기화되면, 대기 서버를 마스터 서버로 전환한 상태에서 이전 데이터베이스 인스턴스를 종료시킬 수 있다. 업그레이드 다운타임은 몇 초 이내로 걸린다.

1.7. 서버 스푸핑 방지

서버 실행 중에 다른 사용자가 악의적으로 정상적인 데이터베이스 서버를 사용하는 것은 불가능하다. 로컬 사용자가 서버 종료 상태에서 다시 서버를 실행하고, 정상적인 서버를 스푸핑하는 것은 가능하다. 스푸핑 서버는 클라이언트가 보낸 패스워드와 쿼리를 읽을 수는 있지만, 디렉토리 권한 때문에 `PGDATA` 디렉토리가 보호되므로 데이터를 리턴할 수 없다. 어떤 사용자든 데이터베이스 서버를 시작할 수 있으므로 스푸핑이 가능하다. 특별히 환경 설정하지 않는 한 클라이언트가 잘못된 서버를 식별해낼 수는 없기 때문이다.

로컬 연결 시 스푸핑을 방지하는 가장 간단한 방법은 신뢰하는 로컬 사용자에 대해서만 쓰기 권한이 있는 Unix 도메인 소켓 디렉토리(unix_socket_directories)를 사용하는 것이다. 이로써

악의적 사용자가 자체 소켓 파일을 해당 디렉토리에 생성하는 것이 방지된다. 일부 애플리케이션이 소켓 파일로 /tmp를 계속 참조하는 것이 우려되면, 운영 체제 시작 시, 위치가 다른 소켓 파일을 가리키도록 심볼릭 링크 /tmp/.s.PGSQL.5432를 생성해야 한다. 또한 사용자의 /tmp 클린업 스크립트를 수정해서 심볼릭 링크가 삭제되지 않게 해야 한다.

TCP 연결 스푸핑을 방지하는 최고의 방법은 SSL 인증서를 사용하여 클라이언트가 서버 인증서를 확인하게 하는 것이다. 따라서 서버가 hostssl 연결만 수락하고(3.1절) SSL 키와 인증서 파일(1.9절)을 갖도록 설정해야 한다. TCP 클라이언트는 sslmode=verify-ca 또는 verify-full을 사용하여 연결해야 하며, 적절한 루트 인증서 파일이 설치되어야 한다.

1.8. 암호화 옵션

Agens SQL은 암호화 단계를 제공하고, 데이터베이스 서버 도난 및 비양심적인 관리자, 불안정한 네트워크로 인한 데이터 유출을 막는다. 암호화는 의료 기록 또는 금융 트랜잭션 같은 중요 데이터의 보호를 위해서도 필요하다.

패스워드 스토리지 암호화

기본적으로 데이터베이스 사용자 패스워드는 MD5 해시로 저장되므로 사용자에게 할당된 실제 패스워드를 관리자가 판단할 수 없다. 클라이언트 인증에 MD5 암호화를 사용하는 경우, 네트워크를 통해 전송하기 전에 클라이언트가 MD5 암호화를 실행하므로 서버에서 잠시라도 암호화되지 않은 패스워드는 없다.

특정 컬럼에 대한 암호화

pgcrypto 모듈은 특정 필드를 암호화해서 저장하는 것을 허용한다. 일부 데이터만 중요한 경우에 이것이 유용하다. 클라이언트가 암호 해독 키를 제공하면 서버에서 암호를 해독한 후, 클라이언트에 전송한다.

암호 해독된 데이터 및 암호 해독 키는 암호가 해독되고 클라이언트와 서버 간에 통신이 일어나는 짧은 시간 동안 서버에 제공된다. 여기에는 시스템 관리자처럼, 전체 데이터베이스 서버에 접근 권한이 있는 사용자가 데이터와 키를 가로챌 수 있는 짧은 순간이 존재한다.

데이터 파티션 암호화

스토리지 암호화는 파일 시스템 레벨 또는 블록 레벨에서 수행된다. 리눅스 파일 시스템 암호화 옵션에는 eCryptfs 및 EncFS가 포함되는 반면, FreeBSD는 PEFS를 사용한다. 블록 레벨 또는 풀 디스크 암호화 옵션은 리눅스에서는 dm-crypt + LUKS를 사용하고 FreeBSD에서는 GEOM 모듈 geli 및 gbde를 사용한다. Windows를 비롯한 여러 운영 체제에서 이 기능을 지원한다.

이 메커니즘은 드라이브 또는 전체 컴퓨터가 도난 당한 경우, 드라이브가 암호화되지 않은 데이터를 읽을 수 없게 한다. 파일 시스템이 마운트되는 경우, 운영 체제가 암호화되지 않은 데이터를 보여주므로 데이터를 보호하지 못한다. 그러나 암호화 키를 운영 체제에 전달하는 방법으로써 마운트 하는 호스트의 어떤 위치에 키를 저장하여 데이터를 보호할 수도 있다.

네트워크에서 패스워드 암호화

MD5 인증 방법은 서버로 전송하기 전에 클라이언트에서 패스워드를 이중으로 암호화한다. 데이터베이스가 연결되면 이것은 먼저 사용자 이름을 기준으로 MD5 암호화한 다음, 서버에 의해 전송되는 랜덤 salt를 기준으로 암호화된다. 이것이 네트워크를 통해 서버에 전송되는 이중 암호화된 값이다. 이중 암호화는 패스워드 검색을 방지하는 것 외에도,

MD5로 동일하게 암호화된 패스워드를 이용해 데이터베이스 서버에 연결하는 것도 방지한다.

네트워크에서 데이터 암호화

SSL 연결은 네트워크로 전송된 모든 데이터(패스워드 및 쿼리, 리턴 데이터)를 암호화한다. `pg_hba.conf` 파일은 어떤 호스트가 암호화되지 않은 연결을 할 수 있는지(`host`)와 어떤 호스트가 SSL-암호화된 연결을 하는지(`hostssl`)를 관리자가 지정할 수 있게 한다. 또, 클라이언트는 SSL을 통해서만 서버에 연결하도록 지정 가능하다. Stunnel 또는 SSH는 전송하는 데이터를 암호화할 때에도 사용할 수 있다.

SSL 호스트 인증

이것은 클라이언트 및 서버 양쪽에서 서로 간에 SSL 인증서를 제공하는 것이 가능하다. 양쪽에서 추가적인 설정이 일부 필요하지만, 이로써 단순히 패스워드만 사용하는 것보다 훨씬 강력한 ID 검증이 가능하다. 이것은 클라이언트가 보낸 패스워드를 읽는 동안 다른 컴퓨터가 서버인 척하는 것을 방지한다. 또한 클라이언트와 서버 사이에 있는 컴퓨터가 서버인 척 해서 클라이언트와 서버 사이의 모든 데이터를 읽고 전달하는 "중간자(man in the middle)" 공격을 방지하는 데에도 효과가 있다.

클라이언트 측 암호화

서버 머신의 시스템 관리자를 신뢰할 수 없는 경우, 클라이언트가 데이터를 암호화해야 한다. 암호화되지 않은 데이터는 데이터베이스 서버에 절대 나타나지 않는다. 데이터는 서버로 전송되기 전에 클라이언트에서 암호화되고, 데이터베이스 결과는 사용 되기 전에 클라이언트에서 해독되어야 한다.

1.9. SSL을 사용한 TCP/IP 연결 보호

Agens SQL은 SSL 연결을 이용해서 기본적으로 클라이언트/서버 통신을 암호화하는 보안 기능을 지원한다. 이것은 클라이언트와 서버 시스템에 OpenSSL을 설치해야 하고, 빌드시 Agens SQL에서 활성화 된다.

컴파일된 SSL을 사용함으로써 `postgresql.conf`에서 ssl을 `on`으로 설정하면 SSL이 활성화된 상태로 Agens SQL 서버를 시작할 수 있다. 서버는 동일한 TCP 포트에서 일반적인 혹은 SSL 연결을 listen하므로, 클라이언트를 SSL로 연결할지 결정한다. 기본적으로 이것은 클라이언트의 옵션이다. 일부 또는 모든 연결에 대해 SSL의 사용을 요구하도록 서버를 설정하는 방법은 3.1절을 참조 바란다.

Agens SQL은 시스템 차원(system-wide)의 OpenSSL 환경 설정 파일을 읽는다. 기본적으로 이 파일의 이름은 `openssl.cnf`이고 `openssl version -d`으로 확인되는 디렉토리에 위치한다. 이러한 기본값은 환경 변수 OPENSSL_CONF를 원하는 환경 설정 파일에 설정하여 덮어 쓸 수 있다.

OpenSSL은 다양한 암호화 및 다양한 레벨의 인증 알고리즘을 지원한다. 암호 목록을 OpenSSL 환경 설정 파일에 지정할 수 있는 반면, `postgresql.conf`에서 ssl_ciphers를 데이터베이스 서버에서 특별히 사용하기 위한 암호로 수정할 수 있다.

> 참고: `NULL-SHA` 또는 `NULL-MD5` 암호를 사용하여 암호화 오버헤드 없이 인증을 하는 것이 가능하다. 단, 중간자(man-in-the-middle)는 클라이언트와 서버 사이간 통신을 읽고 빠져나갈 수 있다. 암호화 오버헤드는 인증 오버헤드에 비해 아주 적으므로, NULL 암호는 권장하지 않는다.

SSL 모드에서 시작하려면 서버 인증서가 포함된 파일과 개인 키가 존재해야 한다. 기본적으로, 각 파일 이름은 데이터 디렉토리에서 `server.crt` 및 `server.key`로 되어 있지만, 환경 설정 매개변수 ssl_cert_file 및 ssl_key_file을 사용하여 이름을 변경하거나 위치를 지정할 수 있다. Unix 시스템에서 `server.key`에 대한 권한은 월드 또는 그룹에 대한 접근을 차단해야 한다. 이것은 **chmod 0600 server.key** 명령으로 수행된다. 개인 키가 암호로 보호되는 경우 서버는 암호를 묻는 메시지를 띄우고, 암호가 입력되기 전에는 서버가 시작되지 않는다.

경우에 따라 서버 인증서가 클라이언트에게 직접적으로 신뢰받는 기관이 아닌 "중간" 인증 기관에서 서명 받을 수도 있다. 이러한 인증서를 사용하려면 `server.crt` 파일에 서명 기관의 인증서를 첨부한 다음, 해당 상급 기관의 인증서를 첨부하고, 클라이언트가 신뢰한 인증 기관 또는 "root", "중간" 인증 기관까지 첨부한다 (예를 들면 클라이언트의 `root.crt` 파일에서 인증서로 서명된).

1.9.1. 클라이언트 인증서 사용

신뢰된 인증서를 클라이언트에게 요구하려면 신뢰하는 인증 기관(CA)의 인증서를 데이터 디렉토리의 `root.crt` 파일에 삽입하고, `postgresql.conf`의 ssl_ca_file를 `root.crt`로 설정하고 `pg_hba.conf`의 적절한 `hostssl` 라인에서 `clientcert`를 1로 설정해야 한다. 그러면, SSL 연결 시작 중에 인증서가 클라이언트로부터 요청된다. 서버는 클라이언트 인증서가 신뢰된 인증 기관 중 한 곳에서 서명된 것인지 검증한다. 중간 CA가 `root.crt`에 있어야 하고, 루트 CA도 포함되어야 한다. ssl_crl_file가 설정된 경우 Certificate Revocation List (CRL) 엔트리도 있는지 확인된다. (SSL 인증서 사용을 보여주는 다이어그램은 http://h71000.www7.hp.com/DOC/83final/BA554_90007/ch04s02.html 참조)

`pg_hba.conf`의 `clientcert` 옵션은 모든 인증 방법에서 사용할 수 있지만 `hostssl`로 지정된 행에만 해당된다. `clientcert`가 지정되지 않았거나 0으로 설정되면, 환경 설정된 것이 있을 경우 서버가 CA 목록에서 클라이언트 인증서가 있는지 계속 검증하지만, 클라이언트 인증서가 꼭 있어야 하는 것은 아니다.

서버의 `root.crt`에는 클라이언트 인증서 서명을 위해 신뢰된 최상위 CA가 나열되어 있다. 대부분의 경우 클라이언트 인증서에 대해서는 CA를 신뢰하더라도, 원칙적으로 서버의 인증서를 서명한 CA 목록은 필요 없다.

사용자가 클라이언트 인증서를 설정하는 경우 `cert` 인증 방법을 사용하면 인증서로 사용자 인증을 제어하고 연결 보안도 제공할 수 있다. 자세한 내용은 3.3.9절을 참조 바란다.

1.9.2. SSL 서버 파일 사용

표 1-2는 서버에서 SSL 설정과 관련된 파일들을 요약한 것이다. (표시된 파일 이름은 기본값 또는 일반적인 이름이다. 로컬에서 환경 설정된 이름은 다를 수 있다.)

표 **1-2. SSL** 서버 파일 사용

파일	내용	효과
ssl_cert_file (`$PGDATA/server.crt`)	서버 인증서	클라이언트로 전송되어 서버 ID 표시
ssl_key_file (`$PGDATA/server.key`)	서버 개인 키	소유자가 보낸 서버 인증서 검증. 인증서 소유자가 믿을만하다는 뜻은 아님
ssl_ca_file (`$PGDATA/root.crt`)	신뢰된 인증서 기관	클라이언트 인증서가 신뢰된 인증 기관에 의해 서명되었는지 확인

파일	내용	효과
ssl_crl_file ($PGDATA/root.crl)	인증 기관에서 취소된 인증서	클라이언트가 인증서가 이 목록에 있으면 안 됨

server.key, server.crt, root.crt 및 root.crl 파일(또는 환경 설정된 다른 이름)은 서버 시작 중에만 검사되므로 변경 내용을 적용하려면 서버를 재시작해야 한다.

1.9.3. 자체 서명된 인증서 생성

서버용 자체 서명된 인증서를 빠르게 생성하려면 다음과 같은 OpenSSL 명령을 사용해야 한다.

```
openssl req -new -text -out server.req
```

openssl이 요청하는 정보를 입력하고, 로컬 호스트 이름을 "Common Name"으로 입력했는지 확인한다. 챌린지 패스워드는 비워둘 수 있다. 프로그램은 패스프레이즈로 보호된 키를 생성한다. 4글자 미만의 패스프레이즈는 수락되지 않는다. 패스프레이즈를 삭제하려면(서버의 자동 시작을 원하는 경우처럼) 다음 명령을 실행한다.

```
openssl rsa -in privkey.pem -out server.key
rm privkey.pem
```

이전 패스프레이즈를 입력하여 기존 키를 해제하기 위해 아래와 같이 입력한다.

```
openssl req -x509 -in server.req -text -key server.key -out server.crt
```

그러면 인증서가 자체 서명된 인증서로 전환되고, 서버가 찾는 위치로 키와 인증서가 복사된다. 마지막으로, 다음과 같이 입력한다.

```
chmod og-rwx server.key
```

이렇게 하는 이유는 권한이 이것보다 좀 더 자유로운 경우에 서버가 파일을 거부하기 때문이다. 서버 개인 키와 인증서를 생성하는 방법에 대한 자세한 내용은 OpenSSL 문서를 참조 바란다.

자체 서명된 인증서를 테스트용으로 사용할 수 있지만 클라이언트가 서버 ID를 검증할 수 있도록 인증 기관(CA) (전역 CA 또는 로컬 CA 중 하나)이 서명한 인증서는 실제 운영 중에 사용되어야 한다. 모든 클라이언트가 로컬인 경우 로컬 CA를 사용하는 것을 권장한다.

1.10. SSH 터널을 사용하여 TCP/IP 연결 보호

클라이언트와 Agens SQL 서버 간 네트워크 연결을 암호화하기 위해 SSH를 사용할 수 있다. 이것은 SSL이 불가능한 클라이언트에 대해서도 적절한 네트워크 연결 보호를 제공한다.

먼저 SSH 서버가 Agens SQL 서버와 동일한 머신에서 실행 중인지 확인하고 **ssh**를 사용하여 사용자로 로그인할 수 있는지 확인한다. 그런 다음, 클라이언트 머신에서 아래와 같은 명령을 사용하여 안전한 터널을 생성할 수 있다.

```
ssh -L 63333:localhost:5432 joe@foo.com
```

-L 의 첫 번째 인자 63333은 터널 끝의 포트 번호이며, 미사용된 포트는 무엇이든 쓸 수 있다. (IANA는 개인용으로 포트 49152 ~ 65535를 제공한다.) 두 번째 숫자 5432는 터널의 원거리 끝

으로, 서버가 사용 중인 포트 번호이다. 포트 번호 간의 이름 또는 IP 주소는 연결하려는 데이터베이스 서버가 있는 호스트이며, 이 예시에서는 foo.com이 로그인되어 있는 호스트이다. 이 터널로 데이터베이스 서버에 연결하기 위해 사용자는 로컬 머신의 63333 포트에 연결한다.

```
psql -h localhost -p 63333 postgres
```

위 명령은 localhost에 연결하는 foo.com 호스트에서 사용자가 정말로 데이터베이스 서버에 대한 joe 사용자인 것처럼 보이게 하고, 이 사용자와 호스트와의 연결에 대해 환경 설정된 대로 인증 절차를 사용한다. 사실, SSH 서버와 Agens SQL 서버 간 연결이 암호화되지 않으므로 서버는 연결이 SSL로 암호화되어 있지 않다고 생각하게 된다. 동일한 머신에 있는 것이 아니라면 어떠한 보안 위험도 생기지 않을 것이다.

터널 설정을 적용하려면 사용자가 **ssh**를 사용하여 터미널 세션을 생성하려고 한 것처럼 joe@foo.com로 **ssh**를 통한 연결을 할 수 있어야 한다.

사용자는 다음과 같이 포트 포워딩을 설정할 수도 있다.

```
ssh -L 63333:foo.com:5432 joe@foo.com
```

그러나 데이터베이스 서버의 foo.com 인터페이스에 연결이 들어오는데, 이 인터페이스는 기본 설정인 listen_addresses = 'localhost'으로 열리지 않아, 사용자로 하여금 불편하다.

어떤 로그인 호스트를 통해 데이터베이스 서버에 "hop"해야 한다면 가능한 설정 중 하나는 다음과 같을 것이다.

```
ssh -L 63333:db.foo.com:5432 joe@shell.foo.com
```

shell.foo.com에서 db.foo.com으로의 연결은 SSH 터널에서 암호화되지 않는다는 점에 유의해야 한다. SSH는 네트워크가 제한되어 있는 여러 가지 상황에서 환경 설정 가능하다. 자세한 내용은 SSH 문서를 참조 바란다.

> 작은 정보: 방금 설명한 개념과 유사한 절차를 사용하여 보안 터널을 제공할 수 있는 애플리케이션이 일부 존재한다.

1.11. Windows에 이벤트 로그 등록

Windows 이벤트 로그 라이브러리를 운영 체제에 등록하려면 다음 명령을 실행해야 한다.

```
regsvr32 pgsql_library_directory/pgevent.dll
```

이것은 Agens SQL이라는 기본 이벤트 소스에 이벤트 뷰어가 사용하는 레지스트리 엔트리를 입력한다.

이벤트 소스 이름을 다르게 지정하려면(event_source 참조), /n 및 /i 옵션을 사용해야 한다.

```
regsvr32 /n /i:event_source_name pgsql_library_directory/pgevent.dll
```

운영 체제에서 이벤트 로그 라이브러리 등록을 취소하려면 아래 명령을 실행해야 한다.

```
regsvr32 /u [/i:event_source_name] pgsql_library_directory/pgevent.dll
```

참고: 데이터베이스 서버에서 이벤트 로깅을 활성화하려면 log_destination을 수정하여 `postgresql.conf`에 `eventlog`를 포함시켜야 한다.

2장. 서버 환경 설정

데이터베이스 시스템의 동작에 영향을 주는 환경 설정 매개변수는 여러 가지가 있다. 본문의 첫 번째 절에서는 환경 설정 매개변수와 인터랙션하는 방법을 설명하고 그 다음 절부터 각 매개변수를 자세하게 다룬다.

2.1. 매개변수 설정

2.1.1. 매개변수 이름 및 값

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.*unit*에서 확인할 수 있다. 편의상, 설정은 명시적으로 지정된 단위를 지정할 수 있다. 예를 들면, 시간 값이 '`120 ms`'인 경우, 매개변수의 실제 단위가 무엇이든 변환 된다. 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.*enumvals*를 참고할 수 있다. Enum 매개변수 값은 대소문자를 구분하지 않는다.

2.1.2. 환경 설정 파일을 통한 매개변수 인터랙션

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 reload`를 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 Agens SQL 데이터 디렉토리에는 `postgresql.auto.conf` 파일이 포함되어 있으며, `postgresql.conf`와 형식은 동일하지만 직접 편집해서는 안 된다. 이 파일에는 ALTER SYSTEM 명령을 통해 제공되는 설정이 포함되어 있다. `postgresql.conf`가 존재할 때마다 이 파일을 자동으로 읽어오고, 해당 설정이 동일하게 적용된다. `postgresql.auto.conf`의 설정은 `postgresql.conf`의 설정을 오버라이드한다.

2.1.3. SQL을 통한 매개변수 인터랙션

Agens SQL는 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.conf`를 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 ROLE**로 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 Agens SQL 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 ALL**과 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';
  ```

2.1.4. 쉘을 통한 매개변수 인터랙션

전역 기본값을 설정하거나 데이터베이스 또는 role 레벨에서 오버라이드 하는 것 외에도, 쉘을 통해 Agens SQL로 설정을 전달할 수 있다. 서버와 libpq 클라이언트 라이브러리 모두 쉘을 통해 매개변수를 전달 받는다.

- 서버 시작 도중에 `-c` 커맨드 라인 매개변수를 사용하여 매개변수 설정을 **postgres** 명령에 전달할 수 있다. 예를 들면,

  ```
  postgres -c log_connections=yes -c log_destination='syslog'
  ```
  이런 방법으로 제공된 설정은 `postgresql.conf` 또는 **ALTER SYSTEM**을 통해 해당 설정을 오버라이드 하므로 서버를 재시작하지 않고는 전역적으로 설정을 변경할 수 없다.
- libpq를 통해 클라이언트 세션을 시작하면 PGOPTIONS 환경 변수를 사용하여 매개변수 설정이 될 수 있다. 이 설정값이 세션의 생명주기 동안 기본값이 되지만 다른 세션에는 영향을 주지 않는다. PGOPTIONS은 **postgres** 명령과 비슷한 형식으로, `-c` 플래그를 지정해야 한다. 예를 들면,

  ```
  env PGOPTIONS="-c geqo=off -c statement_timeout=5min" psql
  ```

  기타 클라이언트 및 라이브러리는 쉘로 자체 메커니즘을 제공하거나 SQL 명령을 직접적으로 사용하지 않고, 사용자가 세션 설정을 변경할 수 있게 한다.

2.1.5. 환경 설정 파일 내용 관리

Agens SQL는 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 지시어*가 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. `include`는 이것을 에러 조건으로 간주하지만, `include_if_exists`는 단순히 메시지를 로깅하여 참조 환경 설정 파일을 계속 처리 한다.

`postgresql.conf` 파일에는 `include_dir`도 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. 로깅 같이 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 커스텀화 된 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 (데이터베이스 환경 설정 파일을 버전 관리에 저장하는 것도 생각해 볼만하다.).

2.2. 파일 위치

앞에서 언급한 postgresql.conf 파일 외에도 Agens SQL는 수동으로 편집되는 환경 설정 파일 2개를 사용하는데, 이 파일은 클라이언트 인증을 관리하는 파일이다(이 파일은 3장에서 다룬다). 기본적으로 3개의 환경 설정 파일 모두 데이터베이스 클러스터의 데이터 디렉토리

에 저장된다. 이 절에서 설명하는 매개변수는 환경 설정 파일을 다른 곳에 배치할 수 있다(그렇게 하면 관리가 편하다. 특히, 환경 설정 파일이 별도로 관리되는 경우에는 환경 설정 파일을 백업하기가 훨씬 쉽다.).

`data_directory` (string)

데이터 저장소로 사용되는 디렉토리를 지정한다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

`config_file` (string)

메인 서버 환경 설정 파일(통상적으로 `postgresql.conf`라고 함)을 지정한다. 이 매개변수는 **postgres** 커맨드 라인에서만 설정 가능하다.

`hba_file` (string)

호스트 기반 인증(통상적으로 `pg_hba.conf`라고 함)용 환경 설정 파일을 지정한다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

`ident_file` (string)

3.2절 사용자 이름 맵핑(통상적으로 `pg_ident.conf`라고 함)용 환경 설정 파일을 지정한다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

`external_pid_file` (string)

서버 관리 프로그램에서 사용하기 위해 서버가 생성해야 하는 추가적인 프로세스 ID(PID) 파일의 이름을 정한다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

기본적으로 위 매개변수들은 설정되어 있지 않다. 데이터 디렉토리는 `-D` 커맨드 라인 옵션 또는 PGDATA 환경 변수로 지정되고, 모든 환경 설정 파일은 데이터 디렉토리에 위치한다.

데이터 디렉토리가 아닌 다른 곳에 환경 설정 파일을 저장하고 싶으면 **postgres** `-D` 커맨드 라인 옵션 또는 PGDATA 환경 변수가 환경 설정 파일이 있는 곳의 위치를 가리켜야 하고, 데이터 디렉토리가 실제로 어디에 있는지를 보여주는 `data_directory` 매개변수는 `postgresql.conf`(또는 커맨드 라인에서)에 설정되어야 한다. `data_directory`는 환경 설정 파일의 위치가 아니라 데이터 디렉토리의 위치에 대해 `-D` 및 PGDATA를 오버라이드한다.

`config_file`, `hba_file` 및/또는 `ident_file` 매개변수를 사용하여 환경 설정 파일의 이름과 위치를 지정할 수 있다. `config_file`은 **postgres** 커맨드 라인에서만 지정 가능하고, 기타 파일은 주 환경 설정 파일 내에서 설정할 수 있다. 이 3가지 매개변수와 `data_directory`를 설정하면, `-D`와 PGDATA를 지정할 필요 없다.

이 3가지 매개변수는 설정 시 **postgres**가 시작된 디렉토리의 상대 경로로 해석된다.

2.3. 연결 및 인증

2.3.1. 연결 설정

`listen_addresses` (string)

서버가 클라이언트 애플리케이션에서 들어온 연결을 listen하는 TCP/IP 주소를 지정한다. 값은 호스트 이름 및/또는 숫자 IP 주소가 쉼표로 구분된 목록의 형태이다. `*`는 사용 가능한 IP 인터페이스 모두를 뜻한다. `0.0.0.0`은 모든 IPv4 주소에 대해 listen하며, `::`는 모든 IPv6 주소에 대해 listen한다는 뜻이다. 목록이 빈칸이면 서버가 IP 인터페이스를 일

절 listen하지 않으며, 이런 경우 Unix 도메인 소켓만 사용해서 연결할 수 있다는 뜻이다. 기본값은 localhost이며, 로컬 TCP/IP "루프백" 연결을 구성한다. 클라이언트 인증(3장)으로 서버 접근 권한을 세분화할 수 있는 반면, listen_addresses는 연결 시도를 수락하는 인터페이스를 제어하여 안전하지 않은 네트워크 인터페이스 상에서 악의적 연결 요청이 반복되는 것을 방지할 수 있다. 이 매개변수는 서버 시작 시 설정된다.

포트 (integer)

기본적으로 서버가 listen하는 TCP 포트는 5432이다. 서버가 listen하는 모든 IP 주소에 동일한 포트 번호가 사용된다. 이 매개변수는 서버 시작 시 설정된다.

max_connections (integer)

데이터베이스 서버 동시 접속 최대 수를 결정한다. 기본값은 일반적으로 100이지만, 커널이 100을 지원하지 않으면 이보다 낮을 수도 있다(initdb 중에 결정됨). 이 매개변수는 서버 시작 시 설정된다.

대기 서버 실행 중에 사용자는 max_connections를 마스터 서버의 값보다 크거나 같게 설정해야 한다. 그렇게 하지 않으면 대기 서버가 쿼리를 허용하지 않는다.

superuser_reserved_connections (integer)

Agens SQL 슈퍼유저의 연결용으로 예약된 연결 "슬롯" 수를 결정한다. 최대 max_connections 까지 결정된다. 동시 연결 수가 max_connections에서 superuser_reserved_connections를 뺀 값보다 크면 슈퍼유저만 더 연결되고 복제 연결은 되지 않는다.

기본값은 3이고, max_connections 미만이어야 한다. 이 매개변수는 서버 시작 시 설정된다.

unix_socket_directories (string)

서버가 클라이언트 애플리케이션 연결을 listen하는 도메인 소켓의 디렉토리이다. 디렉토리들을 쉼표로 구분하여 나열하면 소켓들을 생성할 수 있다. 항목 간 공백은 무시된다. 이름에 공백이나 쉼표를 넣어야 하는 경우 디렉토리 이름 앞뒤에 큰따옴표를 사용한다. 값을 빈칸으로 두면 Unix 도메인 소켓에서 일절 listen하지 않는다. 이때 TCP/IP 소켓만 서버에 연결하는 데 사용될 수 있다. 기본값은 통상 /tmp이지만, 빌드 시에는 변경 가능하다. 이 매개변수는 서버 시작 시 설정된다.

소켓 외에, 파일 자체의 이름도 .s.PGSQL.*nnnn*이다. 여기서 *nnnn*은 서버의 포트 번호이며, 이름이 .s.PGSQL.*nnnn*.lock인 일반 파일이 각 unix_socket_directories 디렉토리에 생성된다. 어떤 파일이든 수동으로 삭제하면 절대 안 된다.

Unix 도메인 소켓이 없는 Windows와 이 매개변수는 무관하다.

unix_socket_group (string)

Unix 도메인 소켓의 소유자 그룹을 설정한다. (소켓을 소유한 사용자는 항상 서버를 시작하는 사용자이다.) unix_socket_permissions 매개변수와 함께 unix_socket_group을 Unix 도메인 연결 시 접근 제어 메커니즘으로 사용할 수 있다. 기본적으로 unix_socket_group은 비어 있는 string이며, 서버 사용자의 기본 그룹을 사용한다. 이 매개변수는 서버 시작 시 설정된다.

Unix 도메인 소켓이 없는 Windows와 이 매개변수는 무관하다.

unix_socket_permissions (integer)

Unix 도메인 소켓의 액세스 권한을 설정한다. Unix 도메인 소켓은 일반적인 Unix 파일 시스템 권한 세트를 사용한다. 매개변수 값은 chmod 및 umask 시스템 쉘에서 수용되는 숫자 형식을 따른다 (8진수 형식을 사용하려면 0(영)으로 시작되는 숫자여야 한다.).

기본 권한은 누구나 연결 가능한 0777이다. 합리적인 다른 대안은 0770(사용자와 그룹만. unix_socket_group 참조) 및 0700(사용자만)이다. (Unix 도메인 소켓의 경우 쓰기 권한에만 해당되는 문제이므로, 읽기 설정이나 취소 또는 실행 권한과는 무관하다.)

이러한 접근 제어 메커니즘은 3장에 설명된 것과는 별개이다.

이 매개변수는 서버 시작 시 설정된다.

이 매개변수는 특히 현재 Solaris 10인 Solaris 시스템과 무관하다. Solaris는 소켓 권한을 완전히 무시한다. unix_socket_directories를 원하는 대상에 한정된 검색 권한을 갖고 있는 디렉토리로 하여 유사한 효과를 낼 수 있다. Unix 도메인 소켓이 없는 Windows에서 이 매개변수는 사용할 수 없다.

bonjour (boolean)

Bonjour를 통해 서버의 존재를 알린다. 기본값은 off이다. 이 매개변수는 서버 시작 시 설정된다.

bonjour_name (string)

Bonjour 서비스 이름을 지정한다. 이 매개변수가 비어 있는 string ″으로 설정된 경우 컴퓨터 이름이 사용된다(기본값). 서버가 Bonjour 지원으로 컴파일되지 않은 경우 이 매개변수가 무시된다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

tcp_keepalives_idle (integer)

TCP가 keepalive 메시지를 클라이언트에 전송하기 전에 비활성화 상태로 대기하는 시간(초). 0을 지정하면 시스템 기본값을 사용한다. TCP_KEEPIDLE 또는 TCP_KEEPALIVE 심볼을 지원하는 시스템과 Windows에서만 지원된다. 다른 시스템에서는 항상 0이어야 한다. Unix 도메인 소켓을 통해 연결된 세션에서 이 매개변수는 무시되고 항상 0으로 읽힌다.

> 참고: Windows는 시스템 기본값을 읽을 수 있는 방법이 없으므로 Windows에서 0은 2시간으로 설정된다.

tcp_keepalives_interval (integer)

클라이언트에 의해 승인되지 않은 TCP keepalive 메시지를 재전송하기 전에 대기하는 시간(초). 0은 기본값을 사용한다. TCP_KEEPINTVL 심볼을 지원하는 시스템과 Windows에서만 지원된다. 다른 시스템에서는 항상 0이어야 한다. Unix 도메인 소켓을 통해 연결된 세션에서는 항상 0으로 읽힌다.

> 참고: Windows는 시스템 기본값을 읽을 수 있는 방법이 없으므로 Windows에서 0은 1초로 설정된다.

tcp_keepalives_count (integer)

몇 개의 TCP keepalive를 분실해야 클라이언트와 서버의 연결 상태를 dead로 판단하는 기준이 되는 지를 정한다. 0은 시스템 기본값을 사용한다. 이 매개변수는 TCP_KEEPCNT 심볼을 지원하는 시스템과 기타 시스템에서만 지원된다. 다른 시스템에서는 항상 0이어야 한다. Unix 도메인 소켓을 통해 연결된 세션에서는 항상 0으로 읽힌다.

참고: 이 매개변수는 Windows에서 지원되지 않으며, 0이어야 한다.

2.3.2. 보안 및 인증

`authentication_timeout` (integer)

클라이언트 인증이 완료되는 최대 시간. 초 단위. 클라이언트가 이 시간 내에 인증 프로토콜이 완료되지 않은 경우 서버는 연결을 닫는다. 이로써 응답이 없는 클라이언트가 연결을 무기한 점유하는 것을 방지한다. 기본값은 1분이다(`1m`). 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

`ssl` (boolean)

SSL 연결을 설정한다. 이것을 사용하기 전에 1.9절을 읽어보기 바란다. 기본값은 `off`이다. 이 매개변수는 서버 시작 시 설정된다. SSL 통신은 유일하게 TCP/IP 연결만 가능하다.

`ssl_ca_file` (string)

SSL 서버 인증 기관(CA)이 포함된 파일 이름을 지정한다. 기본값은 로드된 CA 파일이 없고 클라이언트 인증서 검증이 수행되지 않음을 뜻하는 빈칸이다. (Agens SQL의 이전 릴리스에서 이 파일의 이름은 `root.crt`로 하드 코딩되었다.) 상대 경로는 데이터 디렉토리에 상대적이다. 이 매개변수는 서버 시작 시 설정된다.

`ssl_cert_file` (string)

SSL 서버 인증서가 포함된 파일 이름을 지정한다. 기본값은 `server.crt`이다. 상대 경로는 데이터 디렉토리에 상대적이다. 이 매개변수는 서버 시작 시 설정된다.

`ssl_crl_file` (string)

SSL 서버 인증서 해지 목록(CRL)이 포함된 파일 이름을 지정한다. 기본값은 로드된 CRL 파일이 없음을 뜻하는 빈칸이다. (Agens SQL의 이전 릴리스에서 이 파일의 이름은 `root.crl`로 하드 코딩되었다.) 상대 경로는 데이터 디렉토리에 상대적이다. 이 매개변수는 서버 시작 시 설정된다.

`ssl_key_file` (string)

SSL 서버 개인 키가 포함된 파일 이름을 지정한다. 기본값은 `server.key`이다. 상대 경로는 데이터 디렉토리에 상대적이다. 이 매개변수는 서버 시작 시 설정된다.

`ssl_renegotiation_limit` (integer)

세션 키 renegotiation 일어나기 전에 암호화된 SSL 연결로 흐를 수 있는 데이터 양. renegotiation은, 대규모 트래픽을 검사할 수 있는 경우에 공격자의 암호 해독 가능성을 줄이지만 큰 성능 저하도 초래한다. 전송 및 수신된 트래픽 합계는 제한을 확인하는 데 사용된다. 이 매개변수가 0으로 설정되면 renegotiation이 실행되지 않는다. 기본값은 `512MB`이다.

참고: 2009년 11월 이전의 SSL 라이브러리는 SSL 프로토콜의 취약성 때문에 SSL renegotiation 사용 시 보안상 허점이 있다. 임시 수정 대책으로 일부 벤더는 renegotiation이 불가능한 SSL 라이브러리를 출시했다. 클라이언트 또는 서버에서 이 라이브러리를 사용하고 있으면 SSL renegotiation이 실행되지 않는다.

`ssl_ciphers` (string)

보안 연결에 사용할 수 있는 SSL cipher 스위트(suite) 목록을 지정한다. 이 설정 구문 및 지원되는 값 목록은 OpenSSL 패키지의 ciphers 설명서를 참조 바란다. 기본값은 `HIGH:MEDIUM:+3DES:!aNULL`이다. 이것은 특별한 보안 요구사항이 없을 경우에 일반적으로 합당하다.

기본값 설명:

`HIGH`

`HIGH` 그룹에서 cipher를 사용하는 Cipher 스위트(예: AES, Camellia, 3DES)

`MEDIUM`

`MEDIUM` 그룹에서 cipher를 사용하는 Cipher 스위트(예: RC4, SEED)

`+3DES`

`HIGH`에 대한 OpenSSL 기본 순서는 3DES 서열이 AES128보다 높기 때문에 문제가 된다. 3DES는 AES128보다 보안 수준이 떨어지고 느리기까지 하므로 이것은 잘못된 방법이다. `+3DES`는 다른 모든 `HIGH` 및 `MEDIUM`을 해독한 후, 재배치한다.

`!aNULL`

인증이 없는 익명의 cipher 스위트를 실행하지 않는다. 해당 cipher 스위트는 중간자(man-in-the-middle) 공격에 취약하므로 사용해서는 안 된다.

사용 가능한 cipher 스위트 상세 내역은 OpenSSL 버전에 따라 달라진다. 현재 설치된 OpenSSL 버전에 대한 실제 상세 내역은 `openssl ciphers -v 'HIGH:MEDIUM:+3DES:!aNULL'` 명령을 사용해야 한다. 이 목록은 서버 키 유형에 따라 런타임 시 필터링된다는 점에 유의해야 한다.

`ssl_prefer_server_ciphers` (bool)

SSL cipher 기본 설정을 서버 것으로 사용할 것인지, 클라이언트 것으로 사용할 것인지 지정한다. 기본값은 true이다.

다른 Agens SQL 버전은 이 설정이 없으며, 항상 클라이언트 기본 설정을 사용한다. 이 설정은 주로 해당 버전의 이전 버전과의 호환성에 대한 것이다. 서버가 적절하게 환경 설정되어 있을 가능성이 높으므로 보통은 서버의 기본 설정을 사용하는 것이 더 낫다.

`ssl_ecdh_curve` (string)

ECDH 키 교환에서 사용할 curve 이름을 지정한다. 연결하는 모든 클라이언트에서 지원되어야 한다. 서버의 Elliptic Curve 키에서 사용되는 것과 동일한 curve일 필요는 없다. 기본값은 `prime256v1`이다.

가장 일반적인 curve의 OpenSSL 이름: `prime256v1` (NIST P-256), `secp384r1` (NIST P-384), `secp521r1` (NIST P-521).

사용 가능한 curve의 전체 목록은 **openssl ecparam -list_curves** 명령을 사용하면 표시할 수 있다. 그렇더라도 모든 curve를 TLS에서 사용할 수 있는 것은 아니다.

password_encryption (boolean)

ENCRYPTED 또는 UNENCRYPTED를 쓰지 않고 패스워드를 CREATE USER 또는 ALTER ROLE에 지정한 경우, 패스워드를 암호화할 것인지 결정한다. 기본값은 on이다(패스워드 암호화).

krb_server_keyfile (string)

Kerberos 서버 키 파일의 위치를 설정한다. 자세한 내용은 3.3.3절을 참조 바란다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

krb_caseins_users (boolean)

GSSAPI 사용자 이름의 대소문자를 구분할 것인지 설정한다. 기본값은 off이다(대소문자 구분). postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

db_user_namespace (boolean)

데이터베이스별 사용자 이름 사용 여부이다. 기본값은 off이다. postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

on일 경우 username@dbname처럼 사용자를 생성해야 한다. 연결된 클라이언트가 username이 전달하면, @ 및 데이터베이스 이름이 사용자 이름에 추가되고 해당 데이터베이스 특정 사용자 이름이 서버에서 조회된다. SQL 환경에서 사용자 이름에 @를 넣어 사용자를 생성하면 사용자 이름에 따옴표를 사용해야 한다는 점에 유의해야 한다.

일반 전역 사용자를 생성할 수 있다. 예를 들면 joe@처럼 사용자 이름에 간단히 @를 추가하면 된다. @는 사용자 이름이 서버에 의해 조회되기 전에 제거된다.

db_user_namespace는 클라이언트와 서버의 사용자 이름이 다르게 표시되게 한다. 인증 검사는 항상 서버의 사용자 이름이 이용되므로 인증 방법은 클라이언트가 아니라 서버의 사용자 이름으로 설정되어야 한다. md5는 클라이언트와 서버에서 사용자 이름을 솔트로 사용하므로, md5를 db_user_namespace와 함께 사용할 수 없다.

> 참고: 이 기능을 완벽한 솔루션을 찾을 때까지의 임시 방편용이다. 때가 되면 이 옵션은 없어질 것이다.

2.4. 리소스 소비

2.4.1. Memory

shared_buffers (integer)

데이터베이스 서버가 공유 메모리 버퍼용으로 사용하는 메모리 양을 설정한다. 기본값은 일반적으로 128메가바이트(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, Agens SQL 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)

거대(huge) 메모리 페이지를 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. 자세한 내용은 1.4.4절을 참조 바란다.

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_buffers 증가분당, 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.

대기 서버 실행 시 마스터 서버 값보다 크거나 같게 설정해야 한다. 그렇게 하지 않으면 대기 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 **VACUUM**에 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). 표현식 평가(expression evaluation) 같이 서버의 모든 루틴이 아니라 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. Agens SQL이 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 운영 체제가 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.

2.4.2. 디스크

temp_file_limit (integer)

정렬 및 해시 임시 파일 같은 임시 파일 또는 보류된 커서용 저장소 파일에 세션이 사용할 수 있는 디스크 공간의 최대 크기를 지정한다. 이 제한을 초과하는 트랜잭션은 취소된다. 값은 킬로바이트 단위로 지정되며 -1(기본값)은 무제한을 의미한다. 슈퍼유저만 이 설정을 변경할 수 있다.

이 설정은 주어진 Agens SQL세션이 동시에 사용하는 모든 임시 파일의 총 공간을 제한한다. 쿼리 실행 시 조용히 사용되는 임시 파일과 달리, 명시적 임시 테이블용으로 사용되는 디스크 공간은 이 제한에 합산되지 않는다는 점에 유의해야 한다.

2.4.3. 커널 리소스 사용량

`max_files_per_process` (integer)

서버 하위 프로세스별로 허용된 동시 오픈 파일의 최대 수를 설정한다. 기본값은 1000이다. 커널이 프로세스별 안전 한계를 강제하는 경우 이 설정은 신경 쓸 필요가 없다. 그러나 일부 플랫폼(특히, 대부분의 BSD 시스템)에서는 여러 개의 프로세스가 모두 여러 파일을 열려고 할 때 커널은 시스템이 실제로 지원하는 파일 수보다 많아도 여는 것을 시도한다. “Too many open files” 실패가 나타난 경우 이 설정을 줄여야 한다. 이 매개변수는 서버 시작 시 설정된다.

2.4.4. 비용기반 **Vacuum** 지연

VACUUM 및 ANAYLYZE 명령 실행 시 다양한 I/O 명령을 수행하는 데 드는 예상 비용을 추적하는 내부 카운터가 시스템에 있다. 누적 비용이 제한값(`vacuum_cost_limit`으로 지정)에 도달하면 `vacuum_cost_delay`에서 지정된 값만큼 명령을 수행하는 프로세스가 잠시 슬립 상태가 된다. 그런 다음, 카운터가 리셋되고 실행이 계속된다.

이 기능으로 관리자는 데이터베이스 동시 작업 시 이러한 명령들이 I/O에 주는 부담을 완화시킬 수 있다. **VACUUM** 및 **ANALYZE** 같은 유지 보수 명령이 빨리 마무리되는 것이 중요하지 않을 때도 있다. 그러나, 일반적으로는 이 명령 때문에 다른 데이터베이스 명령을 수행 중인 시스템 능력이 저해되지 않게 하는 것이 중요하다. 비용 기반 vacuum 지연은 관리자가 이것을 수행하는 방법을 제공한다.

VACUUM 명령을 직접 실행하면 기본적으로 실행되지 않는다. 사용하는 것으로 설정하려면 `vacuum_cost_delay` 변수를 0 이외의 값으로 설정해야 한다.

`vacuum_cost_delay` (integer)

비용 제한을 초과한 경우 프로세스가 슬립하는 초 단위의 시간 길이. 기본값은, 비용 기반 vacuum 지연 기능을 사용하지 않는 0이다. 양의 값은 비용 기반 vacuuming을 사용하는 것으로 설정된다. 다수의 시스템에서 슬립 지연의 효율적인 설정은 10밀리초이다. `vacuum_cost_delay`를 10의 배수가 아닌 다른 값으로 설정하면 10의 배수로 값을 올림하여 설정한 것과 결과가 동일하다.

비용 기반 vacuuming을 사용하는 경우 `vacuum_cost_delay`는 일반적으로 매우 작으며, 보통 10 ~ 20밀리초이다. vacuum의 자원 소비를 조절하려면 다른 vacuum 비용 매개변수를 변경하는 것이 가장 좋다.

`vacuum_cost_page_hit` (integer)

공유 버퍼 캐시에 있는 버퍼 vacuuming의 예상 비용. 이것은 버퍼 풀을 잠그고, 공유 해시 테이블을 조회하고 페이지 내용을 스캔 하는 비용을 나타낸다. 기본값은 1이다.

`vacuum_cost_page_miss` (integer)

디스크에서 읽어온 버퍼를 vacuuming하는 데 드는 예상 비용. 이것은 버퍼 풀을 잠그고, 공유 해시 테이블을 조회하고, 디스크에어 원하는 블록을 읽고, 내용을 스캔하는 비용을 나타낸다. 기본값은 10이다.

vacuum_cost_page_dirty (integer)

이전에 클린한 블록을 vacuum이 수정하는 경우 예상 비용. 이것은 dirty 블록을 디스크에 다시 쓰는 데 필요한 추가 I/O를 나타낸다. 기본값은 20이다.

vacuum_cost_limit (integer)

vacuuming 프로세스를 슬립 시키는 누적 비용. 기본값은 200이다.

참고: 일부 명령은 critical locks을 갖고 있으며 따라서, 이러한 명령은 가능한 한 신속하게 완료해야 한다. 해당 명령이 수행되는 중에는 비용 기반 vacuum 지연이 발생하지 않는다. 따라서, 지정된 제한보다 비용이 훨씬 더 많아져 누적될 수도 있다. 이런 경우 쓸데없이 긴 지연을 방지하기 위해 실제 지연은 최대값이 vacuum_cost_delay * 4인 vacuum_cost_delay * accumulated_balance / vacuum_cost_limit으로 계산된다.

2.4.5. 백그라운드 Writer

*백그라운드 writer*라는 별도의 서버 프로세스가 있는데, 이 기능은 "더티(dirty)"(신규 또는 수정) 공유 버퍼에 쓰기 작업을 실행하는 것이다. 이것은 공유 버퍼에 쓰는 것이므로, 서버는 사용자 쿼리를 거의 처리하지 않거나 쓰기가 시작될 때까지 기다릴 필요도 없다. 그러나, 반복적으로 더티 페이지가 체크포인트 간격당 한 번만 기록될 수 있는 반면, 동일한 간격으로 더티 페이지에 백그라운드 writer가 여러 번 쓸 수 있으므로 백그라운드 writer는 최종적으로 I/O 로드의 전반적인 증가를 초래한다. 이 절에서 논의된 매개변수는 로컬로 필요한 동작을 조절하는 데 이용할 수 있다.

bgwriter_delay (integer)

백그라운드 writer의 작업 라운드 사이의 지연을 지정한다. 각 라운드에서 writer는 몇 개의 dirty 버퍼(다음 매개변수로 조절 가능)에 대해 쓰기를 실행한다. 그리고 bgwriter_delay 밀리초 동안 슬립한 다음, 반복한다. 버퍼 풀에 더티 버퍼가 없으면, bgwriter_delay와 무관하게 장기 슬립으로 들어간다. 기본값은 200밀리초이다(200ms). 다수의 시스템에서 효율적인 슬립 지연 설정은 10밀리초이다. bgwriter_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.

bgwriter_lru_maxpages (integer)

각 라운드에서 이것보다 큰 버퍼 수는 백그라운드 writer가 쓰지 않는다. 이것을 0으로 설정하면 백그라운드 쓰기가 사용되지 않는다. (별도의 전용 보조 프로세스에 의해 관리되는 해당 체크포인트들은 영향을 받지 않는다.) 기본값은 100개 버퍼이다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

bgwriter_lru_multiplier (floating point)

각 라운드에 쓰기 된 더티 버퍼의 수는 최근 라운드 중에 서버 프로세스가 필요로 했던 새 버퍼 수를 근거로 한다. 최근 평균 요구량은 다음 라운드에서 필요한 버퍼 추정치에 도달하도록 bgwriter_lru_multiplier를 곱한다. 깨끗하고 재사용 가능한 버퍼 수가 지정된 개수가 될 때까지 더티 버퍼는 쓰여진다. (그렇더라도 라운드당 bgwriter_lru_maxpages 이상의 버퍼는 쓰여지지 않는다.) 따라서, 1.0 설정은 정확히 필요한 버퍼 수를 기록하는 "just in time" 정책을 나타낸다. 더 큰 값을 설정하면 수요 급증에 대비할 수 있는 반면, 더 작은 값은 쓰기를 서버 프로세스가 처리하도록 일부러 미

처리 상태로 방치한다. 기본값은 2.0이다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

bgwriter_lru_maxpages 및 bgwriter_lru_multiplier 값을 더 작게 설정하면 백그라운드 writer로 인한 I/O 로드는 줄어들지만, 서버 프로세스가 자체적으로 쓰기를 실행해야 하므로 쿼리 인터랙션이 지연될 가능성이 높다.

2.4.6. 비동기 동작

effective_io_concurrency (integer)

Agens SQL이 동시에 실행 가능할 것으로 예상하는 동시 디스크 I/O 실행 수를 설정한다. 이 값을 올리면 각 Agens SQL 세션이 병렬 초기화를 시도하는 I/O 실행 수가 늘어난다. 허용 범위는 1 ~ 1000, 또는 비동기 I/O 요청의 실행을 비활성화하는 0이다. 현재, 이 설정은 비트맵 힙 스캔에만 영향을 미친다.

이 설정을 위한 권장 시작점은 데이터베이스에서 사용되는 RAID 0 스트라이프 또는 RAID 1 미러를 환경 설정하는 개별 드라이브 수이다. (RAID 5의 경우 패리티 드라이브 수는 계산하지 않는다.) 그러나 동시 세션에서 여러 개의 쿼리가 실행된 상태에서 데이터베이스가 매우 바쁠 경우, 디스크 배열을 바쁜 상태로 유지하기 위해서는 적은 값도 충분하다. 디스크를 바쁜 상태로 만들기 위해 필요 이상으로 큰 값을 설정하면 CPU 오버헤드만 가중된다.

버스 대역폭에 의해 제한되는 메모리 기반 저장소 또는 RAID 배열 같은 외부 시스템의 경우, 가용한 I/O 경로 수를 설정해야 한다. 최상의 값을 찾아내려면 몇 가지 시험이 필요할 수도 있다.

비동기 I/O는 효과적인 posix_fadvise 함수에 따라 달라지며, 이것이 운영 체제에 없을 수도 있다. 이 함수가 없을 경우 이 매개변수를 0 이외의 다른 것으로 설정하면 에러가 발생한다. 일부 운영 체제(예: Solaris)에는 함수가 존재하지만 실제로는 아무것도 하지 않는다.

max_worker_processes (integer)

시스템이 지원할 수 있는 백그라운드 프로세스의 최대 수를 설정한다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

대기 서버 실행 중에는 이 매개변수를 마스터 서버 값보다 크거나 같게 설정해야 한다. 그렇게 하지 않으면 대기 서버가 쿼리를 허용하지 않는다.

2.5. Write Ahead 로그

이 설정의 조정에 대한 자세한 내용은 13.4절을 참조 바란다.

2.5.1. 설정

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 대기 서버에서 읽기 전용 쿼리에 필요한

정보를 좀 더 추가한다. logical은 논리적 디코딩을 지원하는 데 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.
CREATE TABLE AS
CREATE INDEX
CLUSTER
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. 대기 서버에서 읽기 전용 쿼리를 사용하려면, 운영 서버에서 wal_level을 hot_standby 이상으로 설정하고 hot_standby는 대기 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 Agens SQL 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 충돌의 경우에 데이터 손상이 복구 불가능할 수 있다. 따라서 외부 데이터로 전체 데이터베이스를 손쉽게 재생성할 수 있는 경우에만 fsync를 해제하는 것이 바람직하다.

fsync를 해제하는 안전한 방법은 데이터베이스를 폐기 및 재생성한 후의 데이터 일괄처리하거나, 빈번하게 재생성되고 장애처리(failover)용으로 사용되지 않는 읽기 전용 데이터베이스 클론인 경우에 데이터베이스를 사용하면서 백업 파일로부터 새 데이터베이스 클러스터를 초기 로딩하는 것이다. 고성능 하드웨어 단독으로는 fsync를 해제하는 합당한 이유가 될 수 없다.

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

여러 가지 상황에서 중요하지 않은 트랜잭션에 대해 synchronous_commit를 해제하면 데이터 충돌 위험 없이 fsync를 해제함으로써 잠재적인 성능상 장점을 다수 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 사

이에 지연이 생길 수 있다. (최대 지연은 wal_writer_delay의 3배이다.) fsync와 달리, 이 매개변수를 off로 설정하면 데이터베이스 불일치 위험이 발생하지 않는다. 운영 체제 또는 데이터베이스 충돌은 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. 자세한 내용은 13.3절을 참조 바란다.

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

동기 복제를 사용 중인 경우 일반적으로 디스크에 로컬로 쓰기 되도록 기다리거나, WAL 레코드의 복제를 기다리거나, 트랜잭션이 비동기적으로 커밋되게 하는 것이 합리적이다. 그러나, local 설정은 디스크에 로컬로 쓰기 되도록 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 로컬로 디스크에 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 다중 문 트랜잭션 커밋 하나를 비동기적으로 만들려면 트랜잭션 내에서 **SET LOCAL synchronous_commit TO OFF**를 실행해야 한다.

wal_sync_method (enum)

디스크에 WAL을 강제로 업데이트할 때 사용되는 메서드. fsync가 off인 경우 WAL 파일을 일절 강제로 업데이트하지 않기 때문에 이 설정은 무관하다. 가능한 값은 다음과 같다.

- open_datasync (open() 옵션 O_DSYNC를 사용하여 WAL 파일 쓰기)
- fdatasync (커밋마다 fdatasync() 호출)
- fsync (커밋마다 fsync() 호출)
- fsync_writethrough (커밋마다 fsync() 호출, 모든 디스크 쓰기 캐시에서 write-through 강제)
- 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 메서드이다. 단, Linux에서는 fdatasync가 기본값이다. 기본값이 반드시 이상적인 것은 아니다. 충돌로부터 안전한 환경 설정을 만들거나 성능을 최적화하려면 값을 변경하거나 시스템 환경 설정의 다른 측면을 변경하는 것이 필요할 수도 있다. 이러한 측면은 13.1절에서 다룬다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

full_page_writes (boolean)

이 매개변수가 on이면, Agens SQL 서버는 checkpoint 이후의 각 디스크 페이지를 처음 수정하는 도중에 해당 페이지의 전체 내용을 WAL에 기록한다. 이것은, 운영 체제 충돌 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 checkpoint에서 시작되므로 checkpoint 이후의 페이지별 첫 번째 변경 중에 해도 충분하다. 그러므로 전체 페이지 쓰기 비용을 줄이는 한 가지 방법은 checkpoint 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 `fsync`을 해제했을 때와 유사하며, 해당 매개변수에 대해 권장되는 것과 환경이 동일할 때만 해제해야 한다.

이 매개변수를 해제하는 것은 point-in-time recovery(PITR)용 WAL 아카이빙의 사용에는 영향을 미치지 않는다(8.3절 참조).

이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 기본값은 `on`이다.

`wal_log_hints` (boolean)

이 매개변수가 `on`이면, Agens SQL 서버는 checkpoint 이후의 각 디스크 페이지를 처음 수정하는 도중에, 소위 힌트 비트(hint bits)의 중요하지 않은 수정에 대해서도 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 시에만 설정 가능하다. 기본값은 `off`이다.

`wal_buffers` (integer)

WAL 데이터에 사용되고 아직 디스크에 기록되지 않은 공유 메모리의 합계. 기본 설정 -1은 shared_buffers의 1/32번째(약 3%)와 동일하게 선택한다. `64kB` 이상, WAL 세그먼트 1개 크기 이하여야 하며, 일반적으로 `16MB`이다. 이 값은 자동 선택이 너무 크거나 작은 경우에 직접 선택할 수 있으며, `32kB` 미만의 양의 값은 `32kB`로 처리된다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

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 -1에 의해 선택된 자동 튜닝은 대부분의 경우 합당한 결과를 주어야 한다.

`wal_writer_delay` (integer)

WAL writer의 작업 라운드 사이의 지연을 지정한다. 각 라운드에서 writer는 WAL을 디스크에 기록한다. 그런 다음, `wal_writer_delay`밀리초 동안 슬립한 다음, 반복한다. 기본값은 200밀리초이다(`200ms`). 다수의 시스템에서 슬립 지연의 효율적인 설정은 10밀리초이다. `wal_writer_delay`를 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_delay`는 WAL 쓰기를 초기화하기 전에 측정된 시간 지연을 마이크로초 단위로 추가한다. 이것은 시스템 로드가 충분히 커서 주어진 간격 내에 트랜잭션을 추가로 커밋할 준비가 된 경우 단일 WAL 쓰기를 통해 대량의 트랜잭션이 커밋되게 함으로써 그룹 커밋 처리량을 개선할 수 있다. 그러나 이것은 WAL 쓰기별로 대기 시간을 최대 `commit_delay` 마이크로초까지 늘리기도 한다. 커밋할 준비가 된 트랜잭션이 없을 경우

지연은 낭비되는 시간이므로 최소한 commit_siblings인 경우만 지연이 수행된다. 쓰기가 곧 시작되는 경우 다른 트랜잭션이 작동된다. 또한 fsync가 비활성화되면 지연이 수행되지 않는다. 기본 commit_delay는 0이다(지연 없음). 수퍼유저만 이 설정을 변경할 수 있다.

9.3 이전의 Agens SQL 릴리스에서 commit_delay는 동작이 다르고 효과도 떨어진다. 이것은 모든 WAL 쓰기가 아닌 커밋에만 영향을 주었고 WAL 쓰기가 곧 완료된 경우에도 환경 설정된 지연 시간 동안 대기했다. Agens SQL 9.3 초반에, 쓸 준비가 된 첫 번째 프로세스는 환경 설정된 시간 간격을 기다리고, 후속 프로세스는 선행 프로세스의 쓰기 연산이 끝날 때까지 대기한다.

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.

2.5.2. Checkpoints

checkpoint_segments (integer)

자동 WAL checkpoints 간 로그 파일 세그먼트의 최대 수(각 세그먼트는 일반적으로 16메가바이트). 기본값은 3개 세그먼트이다. 이 매개변수를 늘리면 충돌 복구에 필요한 시간을 늘릴 수 있다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

checkpoint_timeout (integer)

자동 WAL checkpoints 간의 최대 시간. 초 단위. 기본값은 5분이다(5min). 이 매개변수를 늘리면 충돌 복구에 필요한 시간을 늘릴 수 있다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

checkpoint_completion_target (floating point)

checkpoints 간 총 시간 분할로써, checkpoints 완료 목표를 지정한다. 기본값은 0.5이다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

checkpoint_warning (integer)

checkpoint 세그먼트 파일을 채움으로써 checkpoints가 여기에 지정된 초 수보다 근접해서 발생한 경우 서버 로그에 메시지를 기록한다(checkpoint_segments를 증가시키는 것이 권장됨). 기본값은 30초이다(30s). 0은 경고를 비활성화한다. checkpoint_timeout가 checkpoint_warning 미만이면 경고가 발생하지 않는다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

2.5.3. Archiving

archive_mode (boolean)

archive_mode을 사용하는 것으로 설정하면 완료된 WAL 세그먼트가 archive_command 설정에 의해 아카이브 저장소로 전달된다. archive_mode 및 archive_command는 별개의 변수이므로 아카이빙 모드를 해지하지 않고도 archive_command를 변경할 수 있다.

이 매개변수는 서버 시작 시에만 설정 가능하다. wal_level이 minimal로 설정된 경우 archive_mode를 사용으로 설정할 수 없다.

archive_command (string)

완료된 WAL 파일 세그먼트를 아카이브하기 위해 실행하는 로컬 쉘 명령. string에서 %p는 아카이브할 파일의 경로명으로 대체되고 %f는 파일명으로만 대체된다. (경로명은 서버(예: 클러스터의 데이터 디렉토리)의 작업 디렉토리에 상대적이다.) % 문자를 명령에 포함하려면 %%를 사용해야 한다. 성공한 경우에만 명령이 0 종료(zero exit) 상태를 리턴하는 것이 중요하다. 자세한 내용은 8.3.1절을 참조 바란다.

이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다. archive_mode가 서버 시작 시에 활성화되지 않은 경우 무시된다. archive_mode가 사용으로 설정된 상태에서 archive_command의 string이 비어 있는 경우(기본값) WAL 아카이빙이 일시적으로 비활성화되지만 서버는 명령이 곧 제시될 것이라는 기대를 갖고 WAL 세그먼트 파일을 계속 누적한다. archive_command가 true만 리턴하는 명령으로 설정하면(예: /bin/true)(Windows에서 REM), 아카이빙이 효율적으로 비활성화되지만, 아카이브 복구에 필요한 WAL 파일의 체인이 끊어지므로 특이한 환경에서만 사용되어야 한다.

archive_timeout (integer)

archive_command는 완료된 WAL 세그먼트를 호출만 한다. 그러므로, 서버에서는 WAL 트래픽이 발생되지 않아서(따라서 여유 시간이 있음) 트랜잭션의 완료 및 아카이브 저장소에서 안전한 기록 사이에 긴 지연이 발생할 수 있다. 데이터가 아카이브되지 않은 채로 방치되지 않게 하기 위해 서버가 새 WAL 세그먼트 파일로 주기적으로 전환되도록 archive_timeout을 설정할 수 있다. 이 매개변수가 0보다 큰 경우 마지막 세그먼트 파일로 전환한 이후로 여기서 지정된 초 시간을 경과할 때마다, 그리고 단일 checkpoint를 비롯한 데이터베이스 작업이 있을 때마다 서버는 새 세그먼트 파일로 전환한다. (checkpoint_timeout을 늘리면 유휴 시스템에서 불필요한 checkpoints가 줄어든다.) 강제 전환 때문에 일찌감치 폐쇄된 아카이브된 파일의 길이는 완전한 전체 파일과 동일하다는 점에 유의해야 한다. 따라서, archive_timeout를 매우 짧게 하는 것은 아카이브 저장소를 부풀게 하므로 현명하지 못하다. archive_timeout을 1분 정도로 설정하는 것이 일반적으로 합당하다. 데이터를 마스터 서버로 빠르게 복사하려면 아카이빙 대신 streaming replication의 사용을 고려해야 한다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

2.6. 복제

이 설정은 내장 *streaming replication* 기능의 동작을 제어한다(9.2.5절 참조). 서버는 마스터 서버거나 대기 서버다. 마스터 서버는 데이터를 전송할 수 있고, 대기 서버는 언제나 복제된 데이터의 수신자이다. cascading replication(9.2.7절 참조)을 사용하는 경우 대기 서버는 수신자 겸 전송자일 수 있다. 매개변수는 주로 전송 및 대기 서버에 대한 것이며, 일부 매개변수는 마스터 서버에서만 의미가 있다. 클러스터 간에 설정은 필요 시 별 문제 없이 다르게 할 수 있다.

2.6.1. 전송 서버(들)

이 매개변수는 복제 데이터를 하나 이상의 대기 서버로 전송하는 서버에 설정할 수 있다. 마스터는 항상 전송 서버 이므로 이 매개변수를 언제나 마스터에 설정해야 한다. 이 매개변수의 역할 및 의미는 대기 서버가 마스터로 된 이후에 변경되지 않는다.

max_wal_senders (integer)

대기 서버 또는 스트리밍 베이스 백업 클라이언트로부터의 동시 연결 최대 수를 지정한다(예: 동시에 실행 중인 WAL 전송자 프로세스의 최대 수). 기본값은, 복제를 비활성화하는 0이다. WAL 전송자 프로세스는 총 연결 수에 포함되므로 매개변수를 max_connections보다 큰 값으로 설정할 수 없다. 스트리밍 클라이언트의 연결이 갑작스럽게 끊어지면 타임아웃이 될 때까지 고아 연결 슬롯이 생기므로, 예상되는 클라이언트의 최대 수보다 이 매개변수를 약간 더 크게 설정하여 연결이 끊어진 클라이언트가 즉시 재연결될 수 있도록 해야 한다. 이 매개변수는 서버 시작 시에만 설정 가능하다. 대기 서버로부터의 연결이 가능하도록 wal_level은 archive와 같거나 크게 설정해야 한다.

max_replication_slots (integer)

서버가 지원할 수 있는 복제 슬롯의 최대 수를 지정한다(9.2.6절 참조). 기본값은 0이다. 이 매개변수는 서버 시작 시에만 설정 가능하다. 복제 슬롯의 사용이 가능하도록 wal_level은 archive와 같거나 크게 설정해야 한다. 현재 존재하는 복제 슬롯 수보다 작은 값으로 설정하면 서버가 시작되지 않는다.

wal_keep_segments (integer)

대기 서버가 streaming replication을 위해 과거 로그 파일을 가져와야 하는 경우 pg_xlog 디렉토리에 저장되는 과거 로그 파일 세그먼트의 최소 수를 지정한다. 각 세그먼트는 보통 16메가바이트이다. 전송 서버에 연결된 대기 서버가 wal_keep_segments 세그먼트에 훨씬 못 미치면 전송 서버는 대기 서버에 의해 필요한 만큼 WAL 세그먼트를 삭제하고, 이때 복제 연결이 중단된다. 다운스트림 연결도 결과적으로 실패한다. (단, 대기 서버는 WAL 아카이빙이 사용 중인 경우 아카이브에서 세그먼트를 가져와서 복구할 수 있다.)

이것은, pg_xlog에 저장되는 최소 세그먼트 수만 지정한다. 시스템은 WAL 아카이브용으로 또는 checkpoint에서 복구용으로 세그먼트를 좀 더 보유해야 할 수 있다. wal_keep_segments가 0(기본값)인 경우 시스템은 대기 서버를 위한 추가 세그먼트를 유지하지 않으므로, 대기 서버에서 사용 가능한 예전 WAL 세그먼트의 수는 이전 checkpoint의 위치 및 WAL 아카이빙의 상태 함수이다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

wal_sender_timeout (integer)

지정된 밀리초 이상 작동되지 않은 복제 연결이 중단된다. 이것은 전송 서버가 대기 서버 충돌 또는 네트워크 중단을 검출할 때 유용하다. 0 값은 시스템 타임아웃 메커니즘을 비활성화한다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 기본값은 60초이다.

2.6.2. 마스터 서버

이 매개변수는 복제 데이터를 하나 이상의 대기 서버로 전송하는 마스터/운영 서버에 설정할 수 있다. 이 매개변수 외에, wal_level은 마스터 서버에 적절하게 설정되어야 하고 옵션으로 WAL 아카이빙도 활성화될 수 있다(2.5.3절 참조). 사용자가 대기 서버가 마스터가 될 가능성에 대비하여 이 매개변수를 설정하고 싶어 하더라도 대기 서버의 이 매개변수 값은 아무 상관이 없다.

synchronous_standby_names (string)

9.2.8절에 설명된 대로 *동기 복제*를 지원할 수 있는 대기 서버 이름을 쉼표로 구분된 목록으로 지정한다. 작동 중인 동기 대기 서버는 한 번에 많아야 하나이다. 커밋 대기 중인 트랜잭션은 이 대기 서버가 데이터 수신을 확인한 후에 진행이 허용된다. 동기 대기 서버는 이 목록 중에서 현재 연결되어 있고 실시간으로 데이터를 스트리밍하는 첫 번째 대기 서

버다(pg_stat_replication 뷰에서 streaming 상태로 표시됨). 이 목록에서 나중에 나타난 다른 대기 서버는 잠재적 동기 대기 서버를 나타낸다. 현재 동기 대기 서버가 어떤 이유로든 연결이 끊어진 경우 우선 순위가 그 다음으로 높은 대기 서버로 즉시 대체된다. 대기 서버 이름을 2개 이상 지정하면 매우 높은 고가용성을 달성할 수 있다.

대기 서버의 WAL 수신자의 primary_conninfo에 설정된 대로 이런 용도의 대기 서버 이름은 대기 서버의 application_name 설정이다. 고유성을 적용하는 메커니즘은 없다. 복제의 경우, 일치하는 대기 서버 중 하나는 정확히 어떤 것인지 가늠하기 어렵지만 동기 대기 서버가 되도록 선택된다. 특수 항목 *는 walreceiver의 기본 애플리케이션 이름을 비롯한 모든 application_name과 일치한다.

동기 대기 서버 이름이 여기서 지정되지 않으면 동기 복제는 활성화되지 않고 트랜잭션 커밋은 복제를 기다리지 않는다. 이것은 기본 환경 설정이다. 동기 복제가 활성화되더라도 개별 트랜잭션은 synchronous_commit 매개변수를 local 또는 off로 설정함으로써 복제를 기다리지 않도록 환경 설정할 수 있다.

이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

vacuum_defer_cleanup_age (integer)

VACUUM 및 HOT 업데이트가 데드 로우(dead row) 버전의 클린업을 연기하기 되는 트랜잭션 수를 지정한다. 기본값은 0개 트랜잭션이다. 이것은 데드 로우(dead row) 버전이 열린 트랜잭션에서 더 이상 보이지 않는 즉시, 가능한 한 빨리 제거될 수 있다는 것을 의미한다. 사용자는 9.5절에 설명된 대로 핫 스탠바이가 지원하는 운영 서버에서 이 값을 0이 아닌 다른 값으로 설정하고 싶을 수도 있다. 이것은 행의 조기 클린업에 의한 충돌 없이 대기 서버에서 쿼리가 완료되는 시간적 여유를 허용한다. 그러나, 운영 서버에서 발생한 쓰기 트랜잭션 수에 관해서 값이 평가되므로, 대기 서버 쿼리에 대해 얼마만큼의 유예 시간이 가능한지를 예측하기는 어렵다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

대체 서버로서 대기 서버의 hot_standby_feedback을 이 매개변수의 사용으로 설정하는 것도 고려해야 한다.

2.6.3. 대기 서버

이 설정은 복제 데이터를 수신하는 대기 서버의 동작을 제어한다. 마스터 서버의 값은 무관하다.

hot_standby (boolean)

9.5절에 설명된 대로 복구 중에 사용자가 쿼리를 연결하고 실행할 수 있는지를 지정한다. 기본값은 off이다. 이 매개변수는 서버 시작 시에만 설정 가능하다. 복구 중 또는 대기 서버 모드에서만 효과가 있다.

max_standby_archive_delay (integer)

핫 스탠바이가 작동 중이면 9.5.2절에서 설명된 대로 이 매개변수는 적용 직전의 WAL 항목과 충돌하는 대기 서버 쿼리를 취소하기 전에 대기 서버가 기다려야 하는 시간을 결정한다. WAL 데이터를 WAL 아카이브에서 읽어오는 경우(따라서 현재가 아닌 경우) max_standby_archive_delay가 적용된다. 기본값은 30초이다. 지정되지 않으면 단위는 밀리초이다. -1 값은 쿼리 충돌이 완료될 때까지 대기 서버가 무한정 대기하도록 허용한다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

취소 전 쿼리를 실행할 수 있는 시간 길이와 max_standby_archive_delay는 동일하지 않는 점에 유의해야 한다. 오히려 이것은 WAL 세그먼트의 데이터를 적용하는 것이 허용

된 최대 총 시간이다. 따라서 WAL 세그먼트 초반에 어떤 쿼리 때문에 상당한 지연이 발생한 경우 후속 충돌 쿼리는 유예 시간이 훨씬 짧아진다.

`max_standby_streaming_delay` (integer)

핫 스탠바이가 작동 중이면 9.5.2절에서 설명된 대로 이 매개변수는 적용 직전의 WAL 항목과 충돌하는 대기 서버 쿼리를 취소하기 전에 대기 서버가 기다려야 하는 시간을 결정한다. WAL 데이터를 스트리밍 복제를 통해 수신하는 경우 `max_standby_streaming_delay`가 적용된다. 기본값은 30초이다. 지정되지 않으면 단위는 밀리초이다. -1 값은 쿼리 충돌이 완료될 때까지 대기 서버가 무한정 대기하도록 허용한다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

취소 전 쿼리를 실행할 수 있는 시간 길이와 `max_standby_streaming_delay`는 동일하지 않은 점에 유의해야 한다. 오히려 이것은 운영 서버로부터 수신했던 WAL데이터를 적용하는 것이 허용된 최대 총 시간이다. 따라서 어떤 쿼리 때문에 상당한 지연이 발생한 경우 후속 충돌 쿼리는 대기 서버가 다시 따라잡을 때까지 유예 시간이 훨씬 짧아진다.

`wal_receiver_status_interval` (integer)

대기 서버의 WAL 수신자 프로세스가 복제 프로세스에 대한 정보를 운영 서버 또는 업스트림 스탠바이로 전송하는 최소 빈도를 지정한다. 이것은 `pg_stat_replication` 뷰를 사용하여 볼 수 있다. 대기 서버는 작성된 마지막 트랜잭션 로그 위치, 디스크에 기록한 마지막 위치 및 적용된 마지막 위치를 알려 준다. 이 매개변수의 값은 리포트 지점간 초 단위의 최대 간격이다. 업데이트는 쓰기 또는 플러시(flush) 위치가 변경될 때마다 전송되거나 최소한 이 매개변수가 지정한 빈도로 전송된다. 따라서 적용 위치는 실제 위치보다 약간 뒤처질 수 있다. 이 매개변수를 0으로 설정하면 상태 업데이트가 완전히 비활성화된다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 기본값은 10초이다.

`hot_standby_feedback` (boolean)

핫 스탠바이가 대기 서버에서 현재 실행 중인 쿼리에 대해 운영 서버 또는 업스트림 스탠바이로 피드백을 전송할 것인지를 지정한다. 이 매개변수는 클린업 레코드에 의해 야기된 쿼리 취소를 없애는 데 사용할 수 있지만 일부 작업 부하의 경우 운영 서버에서 데이터베이스 팽창을 초래할 수 있다. 피드백 메시지는 `wal_receiver_status_interval`당 한 번 이상 전송되지 않는다. 기본값은 `off`이다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

케스케이드형 복제를 사용 중인 경우 운영 서버에 도달할 때까지 피드백이 상류로 전달된다. 상류 전달 외에, 대기 서버는 수신하는 피드백을 다른 용도로 사용하지 않는다.

`wal_receiver_timeout` (integer)

지정된 밀리초 이상 작동되지 않은 복제 연결이 중단된다. 이것은 대기 서버가 프라이머리 노드 충돌 또는 네트워크 중단을 검출할 때 유용하다. 0 값은 시스템 타임아웃 메커니즘을 비활성화한다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 기본값은 60초이다.

2.7. 쿼리 플랜

2.7.1. 플래너 방법 환경 설정

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 솔루션이 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(2.7.2절 참조), ANALYZE를 수동으로 실행하고, default_statistics_target 환경 설정 매개변수 늘리고, **ALTER TABLE SET STATISTICS**를 사용하여 특정 칼럼에 대해 수집된 통계량을 늘리는 것이다.

`enable_bitmapscan` (boolean)

쿼리 플래너의 bitmap-scan plan types 사용을 활성화 또는 비활성화한다. 기본값은 `on`이다.

`enable_hashagg` (boolean)

쿼리 플래너의 hashed aggregation plan types 사용을 활성화 또는 비활성화한다. 기본값은 `on`이다.

`enable_hashjoin` (boolean)

쿼리 플래너의 hash-join plan types 사용을 활성화 또는 비활성화한다. 기본값은 `on`이다.

`enable_indexscan` (boolean)

쿼리 플래너의 index-scan plan types 사용을 활성화 또는 비활성화한다. 기본값은 `on`이다.

`enable_indexonlyscan` (boolean)

쿼리 플래너의 index-only-scan plan types 사용을 활성화 또는 비활성화한다. 기본값은 `on`이다.

`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 `on`이다.

`enable_mergejoin` (boolean)

쿼리 플래너의 merge-join plan types 사용을 활성화 또는 비활성화한다. 기본값은 `on`이다.

`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 하나를 사용하는 것이 방지된다. 기본값은 `on`이다.

`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 하나를 사용하는 것이 방지된다. 기본값은 `on`이다.

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 하나를 사용하는 것이 방지된다. 기본값은 on이다.

enable_tidscan (boolean)

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

2.7.2. 플래너 비용 상수

이 절에서 설명하는 *cost* 변수는 임의의 규모로 계산된다. 동일한 계수로 상향 또는 하향되는 상대적인 값만 플래너의 선택으로 바뀌지 않는다. 기본적으로, 이러한 비용 변수는 순차적 페이지 가져오기 비용을 근거로 한다. 즉, seq_page_cost는 인습적으로 1.0으로 설정되며, 다른 비용 변수는 그것을 기준으로 설정된다. 그러나 사용자가 원한다면 특정 머신에서 밀리초 단위의 실제 실행 시간 같이 다른 스케일을 사용할 수도 있다.

> 참고: 아쉽게도 비용 변수에 대한 이상적인 값을 결정하는 제대로 정의된(well-defined) 방법은 없다. 특정한 설치가 수신하는 전체 쿼리 믹스에 대한 평균으로 처리하는 것이 최선이다. 이것은 몇 가지 경험에 비추어 값을 변경하는 것은 매우 위험할 수 있음을 의미한다.

seq_page_cost (floating point)

플래너가 예상한, 순차 가져오기 시리즈의 일부분인 디스크 페이지 가져오기 비용을 설정한다. 기본값은 1.0이다. 이 값은 동일한 이름의 테이블스페이스 매개변수 설정에 의해 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.

기계적 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). 랜덤 읽기의 90%는 캐싱되는 것으로 예상되는 반면, 기본값은 순차보다 모델링 랜덤 액세스가 40배 느린 것으로 생각될 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. 반도체 드라이브 같이 랜덤 읽기 비용이 시퀀스에 비해 상대적으로 낮은 저장소는 더 낮은 random_page_cost 값으로 모델링이 더 잘 될 수도 있다.

> 작은 정보: random_page_cost를 seq_page_cost 미만으로 설정하는 것이 시스템에서 허용되더라도 실제로는 그렇게 하는 것이 합리적이지 않다. 단, 데이터베이스 전체가 RAM에 캐치되는 경우에는 시퀀스 밖 페이지를 손대는 것에 대한 패널티가 없으므로 동일하게 설정하는 것은 괜찮다. 또한 과도하게 캐시되는 데이터베이스에서 RAM에 이미 있는 페이지를 가

져오는 비용이 일반적인 상태의 것보다 훨씬 적으므로 사용자는 CPU 매개변수에 비례하여 양쪽 값을 줄여야 한다.

`cpu_tuple_cost` (floating point)

플래너가 예상한 쿼리 도중 각 행의 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)

단일 쿼리에 사용할 수 있는 디스크 캐시의 효율적인 크기에 대한 플래너의 가정을 설정한다. 이것은 인덱스를 사용하는 비용 추정에 반영된다. 값이 클수록 인덱스 스캔이 사용될 가능성이 높다. 값이 작을수록 순차 스캔이 사용될 가능성이 높다. 이 매개변수를 설정하는 경우 Agens SQL의 공유 버퍼와, Agens SQL 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 Agens SQL에 의해 할당된 공유 메모리 크기에는 효과가 없으며, 커널 디스크 캐시도 예약하지 않는다. 추정용으로만 사용된다. 또한 시스템은 디스크 캐시에 쿼리 간 데이터가 잔류할 것이라고 가정하지 않는다. 기본값은 4기가바이트이다(`4GB`).

2.7.3. 제네릭 쿼리 옵티마이저

제네릭 쿼리 옵티마이저(GEQO)는 휴리스틱 검색을 사용하는 쿼리 플래닝을 하는 알고리즘이다. 이것은 검색 비용이 많이 드는 일반 알고리즘보다 적은 플랜 생성 비용으로, 복잡한 쿼리(다수의 관계 조인)의 플래닝 시간을 줄인다.

`geqo` (boolean)

전체 쿼리 최적화를 활성화 또는 비활성화한다. 기본값은 on이다. 보통은 운영 중 해제하지 않는 것이 최선이며 `geqo_threshold` 변수는 좀 더 세분화된 GEQO 제어를 제공한다.

`geqo_threshold` (integer)

최소한 `FROM` 항목에 관련된 수만큼 쿼리를 플랜하는 전체 쿼리 최적화을 사용한다. (`FULL OUTER JOIN` 구문은 하나의 `FROM` 항목으로 계산된다.) 기본값은 12이다. 단순 쿼리의 경우는 일반적으로 정규, 소모성 검색 플래너를 사용하는 것이 낫지만, 테이블이 다수 있는 쿼리의 경우 소모성 검색은 너무 오래 걸리며, 차선의 플랜을 실행하는 것보다 더 오래 걸릴 수도 있다. 따라서 쿼리 크기에 대한 임계값은 GEQO 사용 관리에 편리한 방법이다.

`geqo_effort` (integer)

GEQO에서 플래닝 시간과 쿼리 플랜의 수준 간 트레이드 오프를 제어한다. 이 변수는 1 ~ 10 사이의 integer여야 한다. 기본값은 5이다. 값이 클수록 쿼리 플래닝에 소요되는 시간이 늘어나지만, 효율적인 쿼리 플랜이 선택될 가능성도 높아진다.

`geqo_effort`가 직접 아는 일은 실제로 없다. GEQO 동작에 영향을 미치는 다른 변수에 대한 기본 값을 계산하는 데에만 이용된다(아래에 설명). 원한다면 그 대신 다른 매개변수를 직접 설정할 수 있다.

`geqo_pool_size` (integer)

GEQO에서 사용되는 풀 크기를 제어한다. 풀 크기는 제네틱 채우기(genetic population)에서의 개체 수를 뜻한다. 이것은 최소 2 이상의 유용한 값이어야 하며, 일반적으로 100 ~ 1000이다. 0으로 설정되면(기본 설정) 적정값이 `geqo_effort` 및 쿼리의 테이블 수에 따라 선택된다.

`geqo_generations` (integer)

알고리즘 반복 숫자이자, GEQO에서 사용되는 생성 수를 제어한다. 이것은 최소 1 이상이어야 하며, 유용한 값은 풀 크기와 동일한 범위 내이다. 0으로 설정되면(기본 설정) 적정값이 `geqo_pool_size`에 따라 선택된다.

`geqo_selection_bias` (floating point)

GEQO에 의해 사용된 selection bias를 제어한다. selection bias는 채우기 내부의 선택적 압력이다. 값은 1.50 ~ 2.00일 수 있으며, 후자가 기본값이다.

`geqo_seed` (floating point)

조인 순서 검색 공간의 랜덤 경로를 선택하기 위해 GEQO에 의한 난수 발생기의 초기값을 제어한다. 값은 0(기본값) ~ 1일 수 있다. 값을 변경하면 탐색할 조인 경로 집합이 바뀌고 결과적으로 발견된 최상의 경로가 좋을 수도 있고 나쁠 수도 있다.

2.7.4. Other Planner Options

`default_statistics_target` (integer)

ALTER TABLE SET STATISTICS를 통해 설정된 칼럼 특정 타겟 없이, 테이블 칼럼에 대한 기본 통계 타겟을 설정한다. 큰 값을 설정하면 **ANALYZE**를 수행하는 데 필요한 시간이 늘어나지만 플래너 평가 수준을 높일 수 있다. 기본값은 100이다.

`constraint_exclusion` (enum)

쿼리 최적화를 위해 쿼리 플래너의 테이블 제약 조건을 제어한다. `constraint_exclusion`의 허용 값은 `on`(모든 테이블에 대해 제약 조건 검사), `off`(제약 조건 검사 안함) 및 `partition`(상속 자식 테이블 및 `UNION ALL` 하위 쿼리에 대해서만 제약 조건 검사)이다. `partition`은 기본 설정이다.

이 매개변수가 이것을 특정 테이블에 대해 허용하면 플래너가 쿼리 조건을 테이블의 `CHECK` 제약 조건과 비교하고 제약 조건에 위배되는 테이블을 검색하는 것은 생략한다. 예를 들면:

```
CREATE TABLE parent(key integer, ...);
CREATE TABLE child1000(check (key between 1000 and 1999)) INHERITS(parent);
CREATE TABLE child2000(check (key between 2000 and 2999)) INHERITS(parent);
...
SELECT * FROM parent WHERE key = 2400;
```

제약 조건 배제가 활성화되면 이 **SELECT**는 성능 개선을 위해 child1000을 일절 스캔하지 않는다.

현재, 제약 조건 배제는 테이블 파티션에 주로 사용되는 경우에만 기본값으로 활성화된다. 모든 테이블에 대해 활성화되며, 간단한 쿼리에도 눈에 띄게 플래닝 오버헤드가 가중되어 간단한 쿼리의 장점이 상쇄된다. 파티션된 테이블이 없을 경우 전적으로 해제하는 것이 좋다.

`cursor_tuple_fraction` (floating point)

검색할 커서 행의 분할에 대한 플래너의 추정치를 설정한다. 기본값은 0.1이다. 값이 작을수록 플래너가 커서에 대해 "fast start" 플랜을 사용하도록 유도되고, 그러면 전체 행을 가져오느라 시간이 오래 걸리는 와중에 처음 몇 개의 행만 빠르게 검색된다. 값이 클수록 총 예상 시간이 중요해진다. 최대 설정 1.0에서는 커서가 정확히 일반 쿼리처럼 플랜되어 총 예상 시간만 고려되고, 첫 번째 행을 얼마나 빨리 가져오는지는 고려하지 않는다.

`from_collapse_limit` (integer)

결과로 나온 `FROM` 목록에 이 숫자만큼의 항목이 없을 경우 플래너가 하위 쿼리를 상위 쿼리에 병합한다. 값이 작을수록 플래닝 시간은 짧아지지만 하위 쿼리 플랜이 나올 수 있다. 기본값은 8이다.

이 값을 geqo_threshold 이상으로 설정하면 GEQO 플래너의 사용이 트리거되어 결과적으로 최적화되지 않은 플랜이 된다. 2.7.3절을 참조 바란다.

`join_collapse_limit` (integer)

결과 목록이 이 항목에 미치지 못할 경우 플래너는 명시적 `JOIN` 구문(`FULL JOIN` 제외)을 `FROM` 항목으로 재작성한다. 값이 작을수록 플래닝 시간은 짧아지지만 하위 쿼리 플랜이 나올 수 있다.

기본적으로 이 변수는 `from_collapse_limit`와 동일하게 설정되며, 대부분의 사용에 적합하다. 1로 설정하면 명시적 `JOIN`의 재정렬이 방지된다. 따라서 이 쿼리에서 지정된 명시적 조인 순서는 관계가 조인되는 실제 순서가 된다. 쿼리 플래너가 항상 최적의 조인 순서를 선택하는 것은 아니므로 고급 사용자는 이 변수를 임시로 1로 선택한 다음, 원하는 조인 순서를 명시적으로 지정할 수 있다.

이 값을 geqo_threshold 이상으로 설정하면 GEQO 플래너의 사용이 트리거되어 결과적으로 최적화되지 않은 플랜이 된다. 2.7.3절을 참조 바란다.

2.8. 에러 리포팅 및 로깅

2.8.1. Where To 로그

`log_destination` (string)

Agens SQL은 stderr 및 csvlog, syslog를 비롯한 서버 메시지를 로깅하는 몇 가지 메서드를 지원한다. Windows의 경우, eventlog로 지원한다. 원하는 로그 대상 목록을 쉼표로 구분하여 이 매개변수를 설정한다. 기본값은 stderr로만 로깅하는 것이다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

csvlog가 `log_destination`에 포함된 경우 로그 항목은 프로그램으로 로그를 로딩하기 편리한 "comma separated value"(CSV) 형식으로 출력된다. 자세한 내용은 2.8.4절을 참조

바란다. CSV 형식 로그 출력을 사용으로 설정하려면 logging_collector를 사용으로 설정해야 한다.

참고: 대부분의 Unix 시스템에서 log_destination 옵션의 syslog를 사용하려면 syslog 데몬의 환경 설정을 변경해야 한다. Agens SQL은 syslog 기능 LOCAL0 ~ LOCAL7(syslog_facility 참조)로 로깅할 수 있지만 대부분의 플랫폼에서 기본값 syslog 환경 설정은 모든 해당 메시지를 취소한다. 이것이 작동되게 하려면 다음과 같은 문장을,

```
local0.*/var/log/postgresql
```

syslog 데몬의 환경 설정 파일에 추가해야 할 수 있다.

Windows에서 log_destination의 eventlog 옵션을 사용하는 경우 이벤트 소스와 라이브러리를 운영 체제에 등록해서 Windows 이벤트 뷰어가 이벤트 로그 메시지를 명확하게 표시하도록 해야 한다. 자세한 내용은 1.11절을 참조 바란다.

logging_collector (boolean)

이 매개변수는 stderr로 전송된 로그 메시지를 캡처하여 로그 파일로 리다이렉트하는 *logging collector* 백그라운드 프로세스를 활성화한다. 일부 메시지 유형은 syslog 출력에 나타나지 않을 수 있으므로 syslog에 로깅하는 것보다 이 방법은 대체로 유용하다. (공통된 예시 중 한 가지는 동적 링커 실패 메시지이고, 또 다른 예시는 archive_command 같은 스크립트에서 생성된 에러 메시지이다.) 이 매개변수는 서버 시작 시에만 설정 가능하다.

참고: logging collector를 사용하지 않고 stderr에 로깅하는 것이 가능하다. 서버의 stderr가 다이렉트된 곳이면 어디든 로그 메시지가 출력된다. 그러나, 해당 메서드는 로그 파일을 로테이션하는 편리한 방법을 제공하지 않으므로 로그 양이 적을 때만 적당하다. 또한 logging collector를 사용하지 않는 일부 플랫폼은 복수 프로세스가 동일한 로그 파일에 동시에 쓰기 때문에 서로가 덮어쓰기 되므로 결과적으로 로그 출력이 왜곡되거나 분실된다.

참고: logging collector는 메시지 분실을 방지하는 용도로 고안되었다. 이것은 부하가 매우 심한 경우에, 컬렉터가 뒤처졌을 경우 서버 프로세스가 추가 로그 메시지의 전송을 시도하면서 차단이 일어날 수 있다. 반대로, 기록할 수 없을 때는 syslog가 메시지를 드롭하는데, 이것은 이러한 상황에서 일부 메시지를 로깅하는 데는 실패했지만 시스템의 나머지는 블로킹하지 않음을 의미한다.

log_directory (string)

logging_collector를 사용으로 설정하면 이 매개변수는 로그 파일이 생성되는 디렉토리를 결정한다. 절대 경로 또는 클러스터 데이터 디렉토리에 대한 상대 경로로 설정할 수 있다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 기본값은 pg_log이다.

log_filename (string)

logging_collector를 사용으로 설정하면 이 매개변수는 생성된 로그 파일의 파일 이름을 설정한다. 값은 strftime 패턴으로 처리되므로, % 이스케이프를 사용하여 시간에 따라 바뀌는 파일 이름을 지정할 수 있다. (시간대 의존적 % 이스케이프가 있을 경우 log_timezone에서 지정된 시간대로 계산된다.) 지원되는 % 이스케이프는 Open Group의 strftime (http://pubs.opengroup.org/onlinepubs/009695399/functions/strftime.html) 규격에 등

재된 것과 유사하다. 시스템의 strftime이 직접 사용되지는 않으므로 플랫폼 특정(비표준) 확장자가 효력이 없다. 기본값은 `postgresql-%Y-%m-%d_%H%M%S.log`이다.

이스케이프 없이 파일 이름을 지정하면 로그 로테이션 유틸리티를 사용하여 결국에는 전체 파일이 채워지는 것을 방지하는 계획을 세워야 한다. 8.4 이전 릴리스에서, `%` 이스케이프가 사용되지 않으면 Agens SQL은 새 로그 파일 생성 시간 epoch를 추가했었는데, 이 기능은 사라졌다.

`log_destination`에서 CSV 형식 출력을 사용으로 설정한 경우 타임스탬프 로그 파일 이름 뒤에 `.csv`가 추가되어 CSV 형식 출력 파일 이름이 만들어진다. (`log_filename`이 `.log`로 끝나는 경우 접미사가 대신 사용된다.)

이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

`log_file_mode` (integer)

Unix 시스템에서, 이 매개변수는 `logging_collector`가 사용으로 설정된 경우 로그 파일에 대한 권한을 설정한다. (Microsoft Windows에서 이 매개변수는 무시된다.) 매개변수 값은, `chmod` 및 `umask` 시스템 쉘에서 수용되는 형식으로 지정된 숫자 형식이어야 한다. (관례적인 8진수 형식을 사용하려면 `0`(영)으로 시작되는 숫자여야 한다.)

기본 권한은, 서버 소유자만 로그 파일을 읽거나 쓸 수 있는 `0600`이다. 일반적으로 유용한 다른 설정은, 소유자 그룹의 멤버가 파일을 읽을 수 있는 `0640`이다. 그러나, 해당 설정을 사용하려면 클러스터 데이터 디렉토리 바깥에서도 파일을 저장하도록 log_directory를 변경해야 한다. 로그 파일에 중요한 데이터가 포함되어 있을 수도 있으므로 어떤 경우든 로그 파일을 누구나 읽을 수 있게 하는 것은 현명하지 못하다.

이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

`log_rotation_age` (integer)

`logging_collector`를 사용으로 설정하면 이 매개변수는 개별 로그 파일의 최대 수명을 결정한다. 여기서 지정된 분 시간이 경과된 후 새로운 로그 파일이 생성된다. 시간을 기준으로 새 로그 파일을 생성하지 않으려면 0으로 설정한다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

`log_rotation_size` (integer)

`logging_collector`를 사용으로 설정하면 이 매개변수는 개별 로그 파일의 최대 크기를 결정한다. 여기서 지정된 킬로바이트가 로그 파일에 방출된 후 새로운 로그 파일이 생성된다. 크기를 기준으로 새 로그 파일을 생성하지 않으려면 0으로 설정한다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

`log_truncate_on_rotation` (boolean)

`logging_collector`가 사용으로 설정된 경우 이 매개변수에 의해 Agens SQL가 이름이 동일한 기존 로그 파일에 추가하는 것이 아니라 파일을 비운다(덮어쓰기). 단, 비우기는 서버 시작 시 또는 크기 기준 로테이션이 아니라 시간 기준 로테이션에 의해 새 파일이 열린 경우에만 실행된다. off인 경우에는 모든 경우에 기존 파일이 추가된다. 예를 들면, `postgresql-%H.log` 같은 `log_filename`과 함께 이 설정을 사용하면 24시간마다 로그 파일을 생성하고 주기적으로 덮어쓰기 된다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

예: 7일간 로그를 유지하고, 1일 1로그 파일의 이름을 `server_log.Mon`, `server_log.Tue` 등으로 명명하고, 마지막 주의 로그를 이 주의 로그로 자동 덮어쓰기 하려면 `log_filename`은 `server_log.%a`로 설정하고, `log_truncate_on_rotation`은 `on`으로 설정하고, `log_rotation_age`는 `1440`으로 설정해야 한다.

예: 24시간 로그를 유지하고, 1시간당 1개 로그 파일을 생성하되, 로그 파일 크기가 1GB를 초과하면 곧장 로테이션되게 하려면 log_filename은 server_log.%H%M으로 설정하고, log_truncate_on_rotation은 on으로 설정하고, log_rotation_age는 60으로 설정하고, log_rotation_size는 1000000으로 설정해야 한다. log_filename 파일에서 %M을 포함하면 크기 구동 로테이션으로 시간의 초기 파일 이름과는 다른 파일 이름이 선택되도록 할 수 있다.

syslog_facility (enum)

syslog에 로깅하도록 설정된 경우 이 매개변수는 사용할 syslog "facility"를 결정한다. LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7 중에서 선택할 수 있으며, 기본값은 LOCAL0이다. 시스템의 syslog 데몬에 관한 문서를 참조하기 바란다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

syslog_ident (string)

syslog에 로깅하도록 설정된 경우 이 매개변수는 syslog 로그에서 Agens SQL 메시지를 식별하기 위해 사용되는 프로그램 이름을 결정한다. 기본값은 postgres이다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

event_source (string)

event log에 로깅하도록 설정된 경우 이 매개변수는 syslog 로그에서 Agens SQL 메시지를 식별하기 위해 사용되는 프로그램 이름을 결정한다. 기본값은 Agens SQL이다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

2.8.2. When To 로그

client_min_messages (enum)

클라이언트로 전송할 메시지 레벨을 제어한다. 유효 값은 DEBUG5 및 DEBUG4, DEBUG3, DEBUG2, DEBUG1, LOG, NOTICE, WARNING, ERROR, FATAL, PANIC이다. 각 레벨에는 후속되는 모든 레벨이 포함된다. 후속 레벨일수록 메시지가 적게 전송된다. 기본값은 NOTICE이다. LOG는 여기서 log_min_messages와는 다른 랭크를 갖는다.

log_min_messages (enum)

서버 로그에 기록할 메시지 레벨을 제어한다. 유효 값은 DEBUG5 및 DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL, and PANIC이다. 각 레벨에는 후속되는 모든 레벨이 포함된다. 후속 레벨일수록 메시지가 로그에 적게 전송된다. 기본값은 WARNING이다. LOG는 여기서 client_min_messages와는 다른 랭크를 갖는다. 수퍼유저만 이 설정을 변경할 수 있다.

log_min_error_statement (enum)

에러 상태를 유발한 SQL 문이 서버 로그에 기록되는 것을 제어한다. 메시지가 지정된 심각도 이상일 경우 현재 SQL 문이 로그 항목에 포함된다. 유효 값은 DEBUG5 및 DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL, PANIC이다. 기본값은, 에러, 로그 메시지, 심각한 에러 또는 패닉을 유발한 문(statement)을 로깅하는 ERROR이다. 실패 문의 로깅을 효율적으로 해제하려면 이 매개변수를 PANIC으로 설정해야 한다. 수퍼유저만 이 설정을 변경할 수 있다.

log_min_duration_statement (integer)

최소한 지정된 밀리 초 동안 문이 실행된 경우 완료된 문별로 지속 시간이 로깅되게 한다. 이것을 0으로 설정하면 모든 문의 기간이 인쇄된다. -1(기본값)은 로깅 문 기간을 비

활성화한다. 예를 들어, `250ms`로 설정하면 250ms 이상 실행된 모든 SQL 문이 로깅된다. 이 매개변수를 활성화하면 애플리케이션에서 최적화되지 않은 쿼리를 찾아내는 데 도움이 된다. 수퍼유저만 이 설정을 변경할 수 있다.

확장 쿼리 프로토콜을 사용하는 클라이언트의 경우 Parse, Bind 및 Execute 단계의 지속 시간이 각각 로깅된다.

> 참고**:** 이 옵션을 log_statement와 함께 사용하면 로그 메시지 지속 시간에 `log_statement`가 반복되지 않으므로 문의 텍스트가 로깅되지 않는다. syslog를 사용하지 않는 경우 프로세스 ID 또는 세션 ID를 사용하여 문 메시지를 나중의 지속 시간 메시지에 연결할 수 있는 log_line_prefix를 사용하여 PID 또는 세션 ID를 로깅하는 것이 좋다.

표 2-1은 Agens SQL에서 사용되는 메시지 심각도 레벨을 설명한다. 로깅 출력이 syslog 또는 Windows의 eventlog에 전송되는 경우 심각도 레벨은 표에 나타난 대로 해석된다.

표 **2-1.** 메시지 심각도 레벨

심각도	용도	**syslog**	**eventlog**
`DEBUG1..DEBUG5`	개발자를 위한 상세 정보를 제공한다.	`DEBUG`	`INFORMATION`
`INFO`	**VACUUM VERBOSE**로부터 출력 같은 사용자가 암시적으로 요청한 정보를 제공한다.	`INFO`	`INFORMATION`
`NOTICE`	긴 식별자 잘라내기에 대한 공지 같이 사용자에게 유익한 정보가 제공된다.	`NOTICE`	`INFORMATION`
`WARNING`	트랜잭션 블록 외부로 **COMMIT** 같은 문제의 가능성이 있는 경고를 제공한다.	`NOTICE`	`WARNING`
`ERROR`	현재 명령이 중단된 원인이 되는 에러를 알려준다.	`WARNING`	`ERROR`
`LOG`	checkpoint 작업 같이 관리자가 관심 있어 할 정보를 알려준다.	`INFO`	`INFORMATION`
`FATAL`	현재 세션이 중단된 원인이 되는 에러를 알려준다.	`ERR`	`ERROR`
`PANIC`	모든 데이터베이스 세션이 중단된 원인이 되는 에러를 알려준다.	`CRIT`	`ERROR`

2.8.3. What To 로그

`application_name` (string)

`application_name`은 NAMEDATALEN 글자 수(표준 빌드에서 64자) 이내의 string일 수 있다. 이것은 일반적으로 서버 연결 시 애플리케이션에 의해 설정된다. 이름은 pg_stat_activity 뷰에 표시되고 CSV 로그 항목에 포함된다. log_line_prefix 매개변수를 통해 일반 로그 항목에 포함될 수도 있다. 인쇄 가능한 ASCII 문자만 `application_name` 값으로 사용된다. 다른 문자는 물음표(`?`)로 대체된다.

`debug_print_parse` (boolean)
`debug_print_rewritten` (boolean)
`debug_print_plan` (boolean)

이 매개변수는 다양한 디버깅 출력을 활성화한다. 설정된 경우 결과로 나온 파싱 트리, 쿼리 재작성 출력 또는 실행된 각 쿼리별로 실행 플랜이 인쇄된다. 이 메시지는 `LOG` 메시지 수준으로 출력되므로 기본적으로 서버 로그에 나타나지만 클라이언트로 전송되지는 않는다. client_min_messages 및/또는 log_min_messages를 조절하여 변경할 수 있다. 이 매개변수의 기본값은 off이다.

`debug_pretty_print` (boolean)

설정된 경우, `debug_pretty_print`는 `debug_print_parse`, `debug_print_rewritten` 또는 `debug_print_plan`에 의해 생성된 메시지를 들여쓰기 한다. 따라서 가독성이 증가하는 대신, off로 설정된 경우의 “compact” 형식보다 출력이 길어진다. 기본값은 on이다.

`log_checkpoints` (boolean)

checkpoints 및 restartpoints가 서버 로그에 로깅되게 한다. 일부 통계는 작성된 버퍼 수 및 작성할 때 소요된 시간을 비롯한 로그 메시지에 포함된다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 기본값은 off이다.

`log_connections` (boolean)

서버로의 각 연결 시도 및 성공한 클라이언트 인증 완료가 로깅되게 한다. 이 매개변수는 세션 시작 후에는 변경할 수 없다. 기본값은 off이다.

> 참고: psql 같은 일부 클라이언트 프로그램은 패스워드가 필수인지 판단하면서 2번 연결을 시도하므로“connection received” 메시지가 중복되어 나타나도 문제를 뜻하지는 않는다.

`log_disconnections` (boolean)

이것은 세션을 중단할 때 외에는 `log_connections`와 유사하게 서버 로그에 한 줄을 출력하고 세션의 지속 시간을 포함한다. 기본값은 off이다. 이 매개변수는 세션 시작 후에는 변경할 수 없다.

`log_duration` (boolean)

완료된 모든 문의 지속 시간이 로깅 되게 한다. 기본값은 `off`이다. 수퍼유저만 이 설정을 변경할 수 있다.

확장 쿼리 프로토콜을 사용하는 클라이언트의 경우 Parse, Bind 및 Execute 단계의 지속 시간이 각각 로깅된다.

참고: 이 옵션과 log_min_duration_statement를 0으로 설정하는 것의 차이는 log_min_duration_statement를 초과하면 쿼리 텍스트가 강제로 로깅되지만, 이 옵션은 그렇지 않다는 것이다. 따라서 log_duration이 on으로 설정되고 log_min_duration_statement가 양의 값을 갖는 경우 모든 지속 시간이 로깅되지만 쿼리 텍스트는 임계값을 초과하는 문인 경우에만 포함된다. 이러한 동작은 고부하 설치에서 통계를 수집할 때 유용하다.

log_error_verbosity (enum)

로깅된 각 메시지에 대해 서버 로그에 작성되는 상세 내역을 제어한다. 유효 값은, 각각 메시지에 표시되는 필드를 나타내는 TERSE 및 DEFAULT, VERBOSE이다. TERSE는 DETAIL 및 HINT, QUERY, CONTEXT 에러 정보의 로깅을 제외한다. VERBOSE 출력은 SQLSTATE 에러 코드 및 소스 코드 파일 이름, 함수 이름 및 에러 발생 줄 번호를 포함한다. 수퍼유저만 이 설정을 변경할 수 있다.

log_hostname (boolean)

기본적으로, 연결 로그 메시지는 연결 호스트의 IP 주소만 표시한다. 이 매개변수를 활성화하면 호스트 이름도 로깅된다. 호스트 이름 설정에 따라 이것이 상당한 성능 패널티를 부과할 수도 있다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

log_line_prefix (string)

이것은 각 로그 줄의 처음에 출력되는 printf 스타일 string이다. % 문자는 "이스케이프 시퀀스"로 시작되며 아래 요약된 상태 정보로 대체된다. 미인식 이스케이프는 무시된다. 다른 문자는 로그 줄에 직접 복사된다. 일부 이스케이프는 세션 프로세스에 의해 인식만 된고, 메인 서버 프로세스 같은 백그라운드 프로세스에 의해 빈 것으로 처리된다. 상태 정보는 % 뒤, 옵션 앞에 숫자 리터럴을 지정함으로써 왼쪽 또는 오른쪽에 정렬될 수 있다. 음의 값은 최소 너비를 갖도록 상태 정보를 오른쪽에서 공백으로 채우고, 양의 값은 왼쪽에서 공백으로 채운다. 패딩(padding)은 로그 파일의 가독성을 늘릴 때 유용하다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 기본값은 비어 있는 string이다.

Escape	Effect	Session only
%a	어플리케이션 이름	yes
%u	사용자 이름	yes
%d	데이터베이스 이름	yes
%r	원격 호스트 이름 또는 IP 주소 및 원격 포트	yes
%h	원격 호스트 이름 또는 IP 주소	yes
%p	프로세스 ID	no
%t	밀리초 없는 타임스탬프	no
%m	밀리초 있는 타임스탬프	no
%i	명령 태그: 세션의 현재 명령 유형	yes
%e	SQLSTATE 에러 코드	no
%c	세션 ID: 아래 참조	no

Escape	Effect	Session only
%l	1부터 시작하는 각 세션 또는 프로세스의 로그 줄 번호	no
%s	프로세스 시작 타임스탬프	no
%v	가상 트랜잭션 ID(backendID/localXID)	no
%x	트랜잭션 ID(아무것도 할당되지 않은 경우 0)	no
%q	출력은 하지 않지만 이 시점에서 중단을 위한 비 세션 프로세스를 표시하며, 세션 프로세스에 의해 무시된다.	no
%%	리터럴 %	no

%c 이스케이프는 점으로 구분된 4바이트 16진수(선행 0 없음) 2개로 환경 설정되는 의사 고유(quasi-unique) 세션을 인쇄한다. 숫자는 프로세스 시작 시간 및 프로세스 ID이므로 해당 항목의 인쇄 공간 절약 방법으로 %c를 사용할 수도 있다. 예를 들면, pg_stat_activity의 세션 식별자를 생성하려면 아래 쿼리를 사용한다.

```
SELECT to_hex(EXTRACT(EPOCH FROM backend_start)::integer) || '.' ||
 to_hex(pid)
FROM pg_stat_activity;
```

작은 정보: log_line_prefix에 비어 있지 않은 값을 설정하면 보통은 마지막 문자가 공백이 되도록 해서 로그 줄의 나머지와 육안상 구분이 되게 해야 한다. 문장 부호를 사용할 수도 있다.

작은 정보: Syslog는 자체 타임스탬프와 프로세스 ID 정보를 생성하므로 syslog에 로깅하는 경우 사용자는 이러한 이스케이프를 포함하는 것을 원하지 않을 수도 있다.

log_lock_waits (boolean)

잠금 획득을 위해 세션이 deadlock_timeout 이상 대기한 경우 로그 메시지를 생성할 것인지를 제어한다. 잠금 대기가 성능 저하의 원인이 되는지를 판단할 때 유용하다. 기본값은 off이다.

log_statement (enum)

로깅할 SQL 문을 제어한다. 유효 값은 none(off), ddl, mod, all(모든 문)이다. ddl은 **CREATE** 및 **ALTER**, **DROP** 문 같은 모든 데이터 정의 문을 로깅한다. mod는 모든 ddl 문과 **INSERT** 및 **UPDATE**, **DELETE**, **TRUNCATE**, **COPY FROM**같은 데이터 수정 문을 로깅한다. **PREPARE** 및 **EXECUTE**, **EXPLAIN ANALYZE**문도 포함된 명령이 적절한 타입인 경우 로깅된다. 확장 쿼리 프로토콜을 사용하는 클라이언트의 경우 Execute 메시지를 수신하면 로깅이 발생되고, Bind 매개변수의 값이 포함된다(작은따옴표를 겹쳐서 사용).

기본값은 none이다. 수퍼유저만 이 설정을 변경할 수 있다. The default is none. Only superusers can change this setting.

참고: 기본 파싱이 완료되어 문 타입이 결정된 후에만 로그 메시지가 발생되므로 log_statement = all 설정에 의해서도 간단한 구문 에러가 포함된 문은 로깅되지 않는다. 확장 쿼리 프로토콜의 경우 이 설정은 Execute 단계 전에 실패한 문을 로깅하지 않는다(예: 파싱 분석 또는 플래닝 도중). 해당 문을 로깅하려면 log_min_error_statement를 ERROR(또는 그 이상)로 설정해야 한다.

log_temp_files (integer)

임시 파일 이름과 크기의 로깅을 제어한다. 정렬, 해시 및 임시 쿼리 결과를 위해 임시 파일을 생성할 수 있다. 로그는 각 임시 파일이 삭제된 경우 입력된다. 0 값은 모든 임시 파일 정보를 로깅하고, 양의 값은 크기가 지정된 킬로바이트 이상일 때만 로깅된다. 기본값은, 해당 로깅이 비활성화되는 -1이다. 수퍼유저만 이 설정을 변경할 수 있다.

log_timezone (string)

서버 로그에 작성되는 타임스탬프에 사용할 시간대를 설정한다. TimeZone과 달리, 이 값은 클러스터 차원(cluster-wide)의 값이므로 모든 세션이 일관되게 타임스탬프를 알려준다. 내장 기본값은 GMT이지만, 일반적으로 postgresql.conf에 오버라이드되고 initdb는 시스템 환경에 해당되는 곳에 설정을 설치한다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

2.8.4. CSV 형식 로그 출력 사용

log_destination 목록에서 csvlog를 포함하면 로그 파일을 데이터베이스 테이블에 편리하게 가져올 수 있다. 이 옵션은 쉼표로 구분된 값(CSV) 형식으로 로그 줄을 출력하며, 밀리초의 타임스탬프, 사용자 이름, 데이터베이스 이름, 프로세스 ID, 클라이언트 호스트:포트 번호, 세션 ID, 세션별 줄 번호, 명령 태그, 세션 시작 시간, 가상 트랜잭션 ID, 일반 트랜잭션 ID, 에러 심각도, SQLSTATE 코드, 에러 메시지, 에러 메시지 상세, 힌트, 에러 유발 내부 쿼리(있을 경우), 에러 위치의 문자 카운트, 에러 문맥, 에어 유발 사용자 쿼리(있을 경우 및 log_min_error_statement에서 활성화된 경우), 에러 위치의 문자 카운트, Agens SQL 소스 코드에서 에러의 위치(log_error_verbosity가 verbose로 설정된 경우) 및 애플리케이션 이름의 칼럼으로 환경 설정되어 있다. CSV 형식 로그 파일 출력을 저장하기 위한 샘플 테이블 정의는 다음과 같다.

```
CREATE TABLE postgres_log
(
log_time timestamp(3) with time zone,
user_name text,
database_name text,
process_id integer,
connection_from text,
session_id text,
session_line_num bigint,
command_tag text,
session_start_time timestamp with time zone,
virtual_transaction_id text,
transaction_id bigint,
error_severity text,
sql_state_code text,
message text,
detail text,
hint text,
```

```
internal_query text,
internal_query_pos integer,
context text,
query text,
query_pos integer,
location text,
application_name text,
PRIMARY KEY (session_id, session_line_num)
);
```

로그 파일을 이 테이블로 가져오려면 **COPY FROM** 명령을 사용해야 한다.

```
COPY postgres_log FROM '/full/path/to/logfile.csv' WITH csv;
```

CSV 로그 파일 가져오기를 단순화하려면 몇 가지 작업이 필요하다.

1. 로그 파일에 대한 일관되고, 예측 가능한 네이밍 체계를 제공하려면 `log_filename` 및 `log_rotation_age`를 설정해야 한다. 이로써 사용자는 파일 이름과 개별 로그 파일이 완료되는 시점을 예상할 수 있으므로 가져오기에 대한 대비를 할 수 있다.
2. `log_rotation_size`를 0으로 설정하면 로그 파일 이름을 예상하기 어려워지므로 크기 기반 로그 파일 로테이션이 비활성화된다.
3. 오래된 로그 데이터가 새 데이터와 동일한 파일에 혼재되지 않게 하려면 `log_truncate_on_rotation`을 `on`으로 설정해야 한다.
4. 위의 테이블 정의에는 프라이머리 키 규격이 포함되어 있다. 이것은 동일한 정보를 두 번 가져오는 실수를 방지하는 데 유용하다. **COPY** 명령은 한 번에 가져올 모든 데이터를 커밋하므로 에러 발생 시 가져오기 전체가 실패하게 된다. 로그 파일을 일부만 가져오고, 나중에 완료 시 다시 파일을 가져오는 경우 프라이머리 키 위반 때문에 가져오기가 실패할 수 있다. 가져오기 전에 로그가 완료되고 닫힐 때까지 기다려야 한다. 이 절차는 또한 기록이 아직 완료되지 않은 일부 라인을 가져오는 실수를 함으로써 **COPY**가 실패하게 되는 사태를 방지한다.

2.9. 실시간 통계

2.9.1. 쿼리 및 인덱스 통계 콜렉터

이 매개변수는 서버 차원(server-wide)의 통계 수집 기능을 제어한다. 통계 수집이 활성화되면 생성된 데이터는 pg_stat 및 pg_statio 계열 시스템 뷰를 통해 액세스할 수 있다. 자세한 내용은 11장을 참조 바란다.

`track_activities` (boolean)

각 세션에서 현재 실행 중인 명령의 실행이 시작될 때 해당 명령에 대한 정보 수집을 활성화한다. 이 매개변수의 기본값은 on이다. 활성화된 경우에도 이 정보가 모든 사용자에게 보이는 것은 아니며, 수퍼유저 및 리포트되는 세션의 소유자에게만 표시되므로 보안 위험을 나타내서는 안 된다. 수퍼유저만 이 설정을 변경할 수 있다.

`track_activity_query_size` (integer)

각각의 활성 세션에 대해 현재 실행 중인 명령을 추적하기 위해 pg_stat_activity.*query* 필드에 예약된 바이트 수를 지정한다. 기본값은 1024이다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

`track_counts` (boolean)

데이터베이스 작업에 대한 통계 수집을 활성화한다. autovacuum은 수집된 정보를 필요로 하므로 이 매개변수의 기본값은 on이다. 수퍼유저만 이 설정을 변경할 수 있다.

`track_io_timing` (boolean)

데이터베이스 I/O 호출의 타이밍을 활성화한다. 운영 체제에 현재 시간을 반복해서 쿼리함으로써 일부 플랫폼에서는 상당한 오버헤드가 발생되므로 이 매개변수는 기본적으로 off이다. 사용자 시스템에서 타이밍 오버헤드를 측정하기 위해 pg_test_timing 도구를 사용할 수 있다. I/O 타이밍 정보는 BUFFERS 옵션이 사용되는 경우 및 pg_stat_statements에 의해 EXPLAIN 출력에서 pg_stat_database에 표시된다. 수퍼유저만 이 설정을 변경할 수 있다.

`track_functions` (enum)

함수 호출 횟수 및 사용된 시간의 추적을 활성화한다. 프로시저 언어 함수 `all`만 추적하기 위해 `pl`를 지정하면 SQL 및 C 언어 함수도 추적한다. 기본값은, 함수 통계 추적을 비활성화하는 `none`이다. 수퍼유저만 이 설정을 변경할 수 있다.

> 참고: 호출 쿼리에 간단하게 "inlined"되는 SQL 언어 함수는 이 설정과 무관하게 추적되지 않는다.

`update_process_title` (boolean)

새 SQL 명령이 서버에서 수신될 때마다 프로세스 제목 업데이트를 활성화한다. 프로세스 제목은 일반적으로 **ps** 명령으로 보거나 Process Explorer를 사용하여 Windows에서 볼 수 있다. 수퍼유저만 이 설정을 변경할 수 있다.

`stats_temp_directory` (string)

임시 통계 데이터를 저장할 디렉토리를 설정한다. 디렉토리에 대한 상대 경로이거나 절대 경로일 수 있다. 기본값은 `pg_stat_tmp`이다. RAM 기본 파일 시스템에서 이것을 지정하면 물리적 I/O 요구 사항이 줄어들고 성능 개선으로 이어질 수 있다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

2.9.2. 통계 모니터링

`log_statement_stats` (boolean)
`log_parser_stats` (boolean)
`log_planner_stats` (boolean)
`log_executor_stats` (boolean)

각 쿼리에 대해 각 모듈의 성능 통계를 서버 로그에 출력한다. 이것은 Unix `getrusage()` 운영 체제 기능과 유사한 대략적인 프로파일링 방법이다. `log_statement_stats`는 총 문 통계를 리포트하고, 그 외의 것은 모듈별 통계를 알려준다. `log_statement_stats`는

모듈별 옵션과 함께 활성화될 수 없다. 이 옵션은 모두 기본적으로 비활성화된다. 수퍼유저만 이 설정을 변경할 수 있다.

2.10. 자동 Vacuuming

이 설정은 *autovacuum* 기능의 동작을 제어한다. 자세한 내용은 7.1.6절을 참조 바란다.

`autovacuum` (boolean)

서버가 autovacuum 런처 데몬을 실행해야 하는지를 제어한다. 기본값은 on이지만, autovacuum이 작동되게 하려면 track_counts도 활성화해야 한다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

이 매개변수가 비활성화된 경우 필요시 시스템이 autovacuum 프로세스를 실행하여 트랜잭션 ID 랩어라운드를 방지한다. 자세한 내용은 7.1.5절을 참조 바란다.

`log_autovacuum_min_duration` (integer)

최소한 지정된 밀리초 동안 autovacuum에 의해 실행된 각각의 액션이 로깅되게 한다. 0으로 설정하면 모든 autovacuum 액션이 로깅된다. -1(기본값)은 autovacuum 액션을 비활성화한다. 예를 들면, 이것을 `250ms`로 설정한 경우 250ms 이상 지속되는 모든 자동 vacuums 및 분석이 로깅된다. 또한 이 매개변수가 `-1` 이외의 다른 값으로 설정된 경우 잠금 충돌이 존재하여 autovacuum 액션을 건너뛰면 메시지가 로깅된다. 이 매개변수를 활성화하면 autovacuum 작업을 추적하는 데 도움이 된다. 이 설정은 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정할 수 있다.

`autovacuum_max_workers` (integer)

한 번에 실행할 수 있는 autovacuum 프로세스(autovacuum 제외)의 최대 수를 지정한다. 기본값은 3이다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

`autovacuum_naptime` (integer)

주어진 데이터베이스에서 autovacuum 실행 사이의 최소 지연을 지정한다. 각 라운드에서 데몬은 데이터베이스를 검사하고 필요 시 해당 데이터베이스 테이블에 대해 **VACUUM** 및 **ANALYZE** 명령을 실행한다. 지연은 초 단위로 측정되며 기본값은 1분이다(`1min`). 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

`autovacuum_vacuum_threshold` (integer)

임의의 테이블에서 **VACUUM**을 트리거하는 데 필요한 업데이트 또는 삭제된 튜플의 최소 수를 지정한다. 기본값은 50튜플이다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 이 설정은 저장소 매개변수를 변경함으로써 개별 테이블에 오버라이드할 수 있다.

`autovacuum_analyze_threshold` (integer)

임의의 테이블에서 **ANALYZE**을 트리거하는 데 필요한 삽입, 업데이트 또는 삭제된 튜플의 최소 수를 지정한다. 기본값은 50튜플이다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 이 설정은 저장소 매개변수를 변경함으로써 개별 테이블에 오버라이드할 수 있다.

autovacuum_vacuum_scale_factor (floating point)

VACUUM의 트리거 여부를 결정할 때 autovacuum_vacuum_threshold에 추가할 테이블 크기의 부분을 지정한다. 기본값은 0.2이다(테이블 크기의 20%). 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 이 설정은 저장소 매개변수를 변경함으로써 개별 테이블에 오버라이드할 수 있다.

autovacuum_analyze_scale_factor (floating point)

ANALYZE의 트리거 여부를 결정할 때 autovacuum_analyze_threshold에 추가할 테이블 크기의 부분을 지정한다. 기본값은 0.1이다(테이블 크기의 10%). 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 이 설정은 저장소 매개변수를 변경함으로써 개별 테이블에 오버라이드할 수 있다.

autovacuum_freeze_max_age (integer)

테이블 내 트랜잭션 ID 랩어라운드를 방지하기 위해 **VACUUM** 명령을 강제로 실행하기 전에 테이블의 pg_class.*relfrozenxid* 필드가 도달할 수 있는 연령(트랜잭션에서)을 지정한다. autovacuum이 달리 비활성화된 경우에도 시스템은 랩어라운드를 방지하기 위해 autovacuum 프로세스를 실행한다는 점에 유의해야 한다.

Vacuum은 pg_clog 서브 디렉토리에서 오래된 파일도 제거하는데, 이는 기본값이 2억 트랜잭션으로 상대적으로 낮기 때문이다. 이 매개변수는 서버 시작 시에만 설정할 수 있지만 저장소 매개변수를 변경함으로써 개별 테이블에 대한 설정을 줄일 수 있다. 자세한 내용은 23.1.5절을 참조 바란다. Vacuum also allows removal of old files from the pg_clog subdirectory, which is why the default is a relatively low 200 million transactions. This parameter can only be set at server start, but the setting can be reduced for individual tables by changing storage parameters. For more information see 7.1.5 절.

autovacuum_multixact_freeze_max_age (integer)

테이블 내 multixact ID 랩어라운드를 방지하기 위해 **VACUUM** 명령을 강제하기 전에 테이블의 pg_class.*relminmxid* 필드가 도달할 수 있는 연령(multixacts에서)을 지정한다. autovacuum이 달리 비활성화된 경우에도 시스템은 랩어라운드를 방지하기 위해 autovacuum 프로세스를 실행한다는 점에 유의해야 한다.

Vacuuming multixacts는 pg_multixact/members 및 pg_multixact/offsets 서브 디렉토리에서 오래된 파일도 제거하는데, 이는 기본값이 4억 multixacts로 상대적으로 낮기 때문이다. 이 매개변수는 서버 시작 시에만 설정할 수 있지만 저장소 매개변수를 변경함으로써 개별 테이블에 대한 설정을 줄일 수 있다. 자세한 내용은 7.1.5.1절을 참조 바란다.

autovacuum_vacuum_cost_delay (integer)

자동 **VACUUM** 명령에 사용되는 비용 지연 값을 지정한다. -1을 지정하면 일반 vacuum_cost_delay 값이 사용된다. 기본값은 20밀리초이다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 이 설정은 저장소 매개변수를 변경함으로써 개별 테이블에 오버라이드할 수 있다.

autovacuum_vacuum_cost_limit (integer)

자동 **VACUUM** 명령에 사용되는 비용 제한 값을 지정한다. -1을 지정하면(기본값) 일반 vacuum_cost_limit 값이 사용된다. 실행 중인 autovacuum workers가 하나 이상 있을 경우 각 worker 제한의 합계가 이 변수의 제한값을 초과하지 않도록 값이 비례 분배된다. 이 매개변수는 postgresql.conf 파일 또는 서버 커맨드 라인에서만 설정 가능하다. 이 설정은 저장소 매개변수를 변경함으로써 개별 테이블에 오버라이드할 수 있다.

2.11. 클라이언트 연결 기본값

2.11.1. 문 동작

search_path (string)

이 변수는 개체(테이블, 데이터 타입, 함수 등)를 스키마가 지정되지 않은 간단한 이름으로 참조하는 경우 스키마가 검색되는 순서를 지정한다. 서로 다른 스키마에 이름이 동일한 개체가 있는 경우 사용된 검색 경로에 처음 발견된 것이 사용된다. 검색 경로에서 어떤 스키마에도 없는 개체는 스키마를 구체화(점을 이용해서) 한 이름으로 지정하여 참조 가능하다.

search_path에 대한 값은 스키마 이름을 쉼표로 구분한 목록이다. 스키마에 존재하지 않는 이름이거나, 사용자가 USAGE 권한이 없는 스키마 이름은 조용히 무시된다.

목록 항목 중 하나가 특수한 이름 $user인 경우, 해당 스키마 및 사용자에게 USAGE 권한이 있으면 SESSION_USER가 리턴한 이름의 스키마는 대체된다. (그 외에는 $user가 무시된다.)

시스템 카탈로그 스키마 pg_catalog는 경로에 있든 없든 항상 검색된다. 경로에 있을 경우 지정된 순서대로 검색된다. pg_catalog가 경로에 없으면 경로 항목을 검색하기 전에 검색된다.

마찬가지로, 현재 세션의 임시 테이블 스키마 pg_temp_*nnn*은 존재할 경우 항상 검색된다. pg_temp 별칭을 사용함으로써 경로에 명시적으로 나열할 수 있다. 경로에 나열하지 않으면 검색부터 된다(pg_catalog를 검색하기도 전에). 그러나, 관계(테이블, 뷰, 시퀀스 등) 및 데이터 타입 이름에 대한 임시 스키마만 검색된다. 함수 또는 연산자 이름으로는 절대 검색되지 않는다.

특별한 타겟 스키마를 지정하지 않고 개체를 생성한 경우 search_path에 명명된 첫 번째 유효 스키마에 배치된다. 검색 경로가 비어 있으면 에러가 리포트된다.

이 매개변수의 기본값은 "$user", public이다. 이 설정은 데이터베이스의 공유 사용(사용자에게 개인 스키마가 없고 모두 public의 공유 사용인 경우), 개인 사용자별 스키마 및 이러한 것들의 조합을 지원한다. 전역적 또는 사용자별 기본 검색 경로 설정을 전환함으로써 다른 효과를 얻을 수도 있다.

검색 경로에서 현재 효과적인 값은 SQL 함수 current_schemas를 통해서 검사할 수 있다. current_schemas는 search_path에 표시되는 항목이 해결되는 방법을 표시하므로 이것은 search_path 값을 검사하는 것과 다르다.

default_tablespace (string)

이 변수는 **CREATE** 명령이 테이블스페이스를 명시적으로 지정하지 않는 경우 개체(테이블 및 인덱스)가 생성되는 기본 테이블스페이스를 지정한다.

값은 테이블스페이스의 이름 또는 현재 데이터베이스의 기본 테이블스페이스를 사용하여 지정하기 위한 비어 있는 string이다. 기존 테이블스페이스의 이름과 값이 일치하지 않으면 Agens SQL이 자동으로 현재 데이터베이스의 기본 테이블스페이스를 사용한다. 기본값이 아닌 테이블스페이스가 지정되면 사용자는 CREATE 권한이 있어야 하며, 그렇지 않으면 생성 시도에 실패한다.

이 변수를 임시 테이블에는 사용되지 않으며, 대신 temp_tablespaces를 사용해야 한다.

이 변수는 데이터베이스 생성 시 사용되지 않는다. 기본적으로 새 데이터베이스는 복사되었던 템플릿 데이터베이스로부터 테이블스페이스 설정을 상속받는다.

테이블스페이스에 대한 자세한 내용은 5.6절을 참조 바란다.

temp_tablespaces (string)

이 변수는 **CREATE** 명령이 테이블스페이스를 명시적으로 지정하지 않는 경우 임시 개체(임시 테이블 및 임시 테이블의 인덱스)가 생성되는 테이블스페이스를 지정한다. 대형 데이터집합 정렬 같은 용도의 임시 파일도 이 테이블스페이스에서 생성된다.

값은 테이블스페이스의 이름 목록이다. 목록에서 이름이 2개 이상 있는 경우, 트랜잭션 내에서 연속 생성된 임시 개체가 목록에서 연속 테이블스페이스에 매치되는 것 외에는 Agens SQL는 임시 개체를 생성할 때마다 목록에서 멤버를 임의로 선택한다. 목록에서 선택된 요소가 비어 있는 string인 경우 Agens SQL이 현재 데이터베이스의 기본 테이블스페이스를 자동으로 대신 사용한다.

사용자에게 CREATE 권한이 없는 테이블스페이스를 지정하는 것이므로, temp_tablespaces가 인터랙티브하게 설정된 경우 존재하지 않는 테이블스페이스를 지정하는 것은 에러이다. 단, 사용자에게 CREATE 권한이 없는 테이블스페이스이므로, 이전에 설정된 값을 사용하는 경우 존재하지 않는 테이블스페이스는 무시된다. 특히, 이 규칙은 postgresql.conf에 설정된 값을 사용하는 경우에 적용된다.

기본 값은, 현재 데이터베이스의 기본 테이블스페이스에서 모든 임시 개체가 생성되는 비어 있는 string이다. default_tablespace도 참조 바란다.

See also default_tablespace.

check_function_bodies (boolean)

이 매개변수는 일반적으로 on이다. off로 설정되면 CREATE FUNCTION 중에 함수 본문 string의 검증이 비활성화된다. 검증을 비활성화하면 검증 프로세스의 부작용이 예방되고 전방 참조 같은 문제로 인한 거짓 긍정이 방지된다. 다른 사용자를 위해 함수를 로딩하기 전에 이 매개변수를 off로 설정해야 한다. pg_dump는 자동으로 off로 설정한다.

default_transaction_isolation (enum)

각 SQL 트랜잭션은 “read uncommitted” 또는 “read committed”, “repeatable read”, “serializable”의 격리 레벨을 갖고 있다. 이 매개변수는 새 트랜잭션마다 기본 격리 레벨을 제어한다. 기본값은 “read committed”이다.

default_transaction_read_only (boolean)

읽기 전용 SQL 트랜잭션은 비 임시 테이블은 변경할 수 없다. 이 매개변수는 새 트랜잭션마다 기본 읽기 전용 상태를 제어한다. 기본값은 off이다(읽기/쓰기).

default_transaction_deferrable (boolean)

serializable 격리 수준에서 실행 중인 경우 진행을 허용하기 전에 유예 가능한 읽기 전용 SQL 트랜잭션을 지연시킬 수 있다. 그러나 실행이 시작되면, 오버헤드가 발생하지 않고 직렬화를 보장하기 때문에 직렬화 코드는 동시 업데이트 때문에 강제 중단을 할 이유가 없어진다. 장기 실행되는 읽기 전용 트랜잭션에 적합하다.

이 매개변수는 새 트랜잭션마다 유예 가능한 기본 상태를 제어한다. 이것은 현재 읽기 전용 트랜잭션에 아무런 효과가 없고, serializable보다 낮은 격리 수준에서 작동되는 트랜잭션에는 효과가 없다. 기본값은 off이다.

session_replication_role (enum)

복제 관련 트리거 시작 및 현재 세션의 규칙을 제어한다. 이 변수를 설정하려면 수퍼유저 권한이 필요하고, 따라서 이전에 캐시된 쿼리 플랜이 삭제된다. 가능한 값은 origin(기본값), replica 및 local이다.

statement_timeout (integer)

명령이 클라이언트에서 서버로 도착한 때부터 걸린 시간이 지정된 밀리초를 초과한 문을 중단한다. log_min_error_statement가 ERROR 이하로 설정되면 타임아웃된 문도 로깅된다. 0 값(기본값)은 이것을 해제한다.

모든 세션에 영향을 줄 수 있으므로 postgresql.conf에서 statement_timeout을 설정하는 것은 권장하지 않는다.

lock_timeout (integer)

테이블, 인덱스, 행 또는 기타 데이터베이스 개체의 잠금을 획득하기 위해 대기한 시간이 지정된 밀리초를 초과하면 문을 중단한다. 획득 시도별로 시간 제한이 개별적으로 적용된다. 이 제한은 명시적 잠금 요청(예: **LOCK TABLE** 또는 NOWAIT하지 않은 **SELECT FOR UPDATE**) 및 암시적으로 획득한 잠금에 모두 적용된다. log_min_error_statement가 ERROR 이하로 설정되면 타임아웃된 문이 로깅된다. 0 값(기본값)은 이것을 해제한다.

statement_timeout과 달리, 이 타임아웃은 잠금 대기 시에만 발생한다. statement_timeout이 0이 아닌 경우, 문 타임아웃이 항상 먼저 트리거되므로 lock_timeout을 동일한 값이나 큰 값으로 설정하는 것은 무의미하다.

모든 세션에 영향을 줄 수 있으므로 postgresql.conf에서 lock_timeout을 설정하는 것은 권장되지 않는다.

vacuum_freeze_table_age (integer)

VACUUM은 테이블의 pg_class.*relfrozenxid* 필드가 이 설정에서 지정된 연령에 도달한 경우 전체 테이블 스캔을 수행한다. 기본값은 1억 5천만 트랜잭션이다. 사용자는 이 값을 0 ~ 20억 중 아무거나 설정할 수 있지만 **VACUUM**은 유효 값을 autovacuum_freeze_max_age의 95%로 슬며시 제한하므로 랩어라운드 방지 autovacuum이 테이블에서 실행되기 주기적으로 직접 **VACUUM**할 수 있다. 자세한 내용은 7.1.5절을 참조 바란다.

vacuum_freeze_min_age (integer)

테이블 스캔 중 행 버전 동결 여부를 결정할 때 **VACUUM**이 사용해야 하는 컷오프 연령(트랜잭션에서)을 지정한다. 기본값은 5천만 트랜잭션이다. 사용자는 이 값을 0 ~ 10억 중 아무거나 설정할 수 있지만 **VACUUM**은 유효 값을 autovacuum_freeze_max_age의 절반으로 슬며시 제한하므로 강제 autovacuum들 사이에 시간이 불합리하게 짧은 경우는 없다. 자세한 내용은 7.1.5절을 참조 바란다.

vacuum_multixact_freeze_table_age (integer)

VACUUM은 테이블의 pg_class.*relminmxid* 필드가 이 설정에서 지정된 연령에 도달한 경우 전체 테이블 스캔을 수행한다. 기본값은 1억 5천만 multixact이다. 사용자는 이 값을 0 ~ 20억 중 아무거나 설정할 수 있지만 **VACUUM**은 유효 값을 autovacuum_multixact_freeze_max_age의 95%로 슬며시 제한하므로 랩어라운드 방지가 테이블에서 실행되기 전에 주기적으로 직접 **VACUUM**할 수 있다. 자세한 내용은 7.1.5.1절을 참조 바란다.

vacuum_multixact_freeze_min_age (integer)

테이블 스캔 중 행 multixact ID를 후속 트랜잭션 ID 또는 multixact ID로 교체 여부를 결정할 때 **VACUUM**이 사용해야 하는 컷오프 연령(multixact에서)을 지정한다. 기본값은 5백만 multixact이다. 사용자는 이 값을 0 ~ 10억 중 아무거나 설정할 수 있지만 **VACUUM**은 유효 값을 autovacuum_multixact_freeze_max_age의 절반으로 슬며시 제한하므로 강제

autovacuum들 사이에 시간이 불합리하게 짧은 경우는 없다. 자세한 내용은 7.1.5.1절을 참조 바란다.

`bytea_output` (enum)

타입 bytea 타입의 값에 대한 출력 형식을 설정한다. 유효 값은 `hex`(기본값) 및 `escape`(전형적인 Agens SQL 형식)이다. bytea 타입은 이 설정과 무관하게 입력에서 항상 두양쪽 형식을 수용한다.

`xmlbinary` (enum)

바이너리 값이 XML로 인코딩되는 방법을 설정한다. 이것은 예를 들면 bytea 값이 `xmlelement` 또는 `xmlforest` 함수에 의해 XML로 변환되는 경우에 적용된다. 가능한 값은, XML 스키마 표준에서 정의된 `base64` 및 `hex`이다. 기본값은 `base64`이다.

여기서 선택한 것을 실제로 선호도의 문제로, 클라이언트 애플리케이션에서 가능한 제약에 의해서만 제한된다. 양쪽 방법은 hex 인코딩이 base64 인코딩보다 다소 크더라도 가능한 모든 값을 지원한다.

`xmloption` (enum)

XML과 문자 string 값 사이의 변환 시 `DOCUMENT` 또는 `CONTENT`가 암시적인지를 설정한다. 유효 값은 `DOCUMENT` 및 `CONTENT`이다. 기본값은 `CONTENT`이다.

SQL 표준에 따라 이 옵션을 설정하는 명령은 다음과 같다.

```
SET XML OPTION { DOCUMENT | CONTENT };
```

이 구문은 Agens SQL에서도 사용할 수 있다.

2.11.2. 로케일(locale) 및 형식 지정

`DateStyle` (string)

날짜 및 시간 값의 표시 형식과, 애매한 날짜 입력 값을 해석하는 규칙을 설정한다. 역사적인 이유로, 이 변수는 두 가지 독립적인 요소인 출력 형식 명세(`ISO` 또는 `Postgres`, `SQL`, `German`)와 연/월/일 순서의 입력/출력 명시(`DMY` 또는 `MDY`, `YMD`)로 환경 설정된다. 이것은 별개로 설정하거나 함께 설정할 수 있다. 키워드 `Euro` 및 `European`은 `DMY`의 동의어이고, 키워드 `US`, `NonEuro` 및 `NonEuropean`은 `MDY`의 동의어이다. 내장된 기본값은 `ISO`, `MDY`이지만 initdb는 선택된 `lc_time` 로케일(locale)의 동작에 해당되는 설정으로 환경 설정 파일을 초기화한다.

`IntervalStyle` (enum)

간격 값에 대한 표시 형식을 설정한다. `sql_standard`는 SQL 표준 간격 리터럴과 일치하는 출력을 생성한다. `postgres` 값(기본값)은 DateStyle 매개변수가 `ISO`로 설정된 경우 Agens SQL 8.4 이전 릴리스와 일치하는 출력을 생성한다. `postgres_verbose` 값은 `DateStyle` 매개변수가 비 `ISO` 출력으로 설정된 경우 Agens SQL 8.4 이전 릴리스와 일치하는 출력을 생성한다. `iso_8601` 값은 ISO 8601의 section 4.4.3.2에 정의된 “format with designators” 시간 간격과 일치하는 출력을 생성한다.

`IntervalStyle` 파라미터매개변수 또한 애매한 간격 입력을 해석하는 데 영향을 준다.

`TimeZone` (string)

타임스탬프를 표시 및 해석하기 위한 시간대를 설정한다. 내장 기본값은 `GMT`이지만, 일반적으로 `postgresql.conf`에 오버라이드되고 initdb는 시스템 환경에 해당되는 곳에 설정을 설치한다.

`timezone_abbreviations` (string)

서버에서 datetime 입력으로 수용되는 시간대 약어 컬렉션을 설정한다. 기본값은, 거의 전세계적으로 적용되는 컬렉션인 `'Default'`이며, 특정 설치용으로 정의할 수 있는 `'Australia'`와 `'India'` 및 기타 컬렉션도 있다.

`extra_float_digits` (integer)

이 매개변수는 float4, float8 및 기하학적(geometric) 데이터 타입을 비롯한 floating-point 값에 대해 표시할 수 있는 자릿수를 조절한다. 매개변수 값은 표준 자릿수에 추가된다(`FLT_DIG` 또는 `DBL_DIG`를 적절하게). 부분적 유효 숫자를 포함하여 값은 최대 3까지 설정할 수 있다. 이것은 정확한 복원이 필요한 부동 데이터를 덤프할 때 특히 유용하다. 또는 불필요한 숫자가 나타나지 않도록 음수로 설정할 수도 있다.

`client_encoding` (string)

클라이언트 측 인코딩(문자 집합)을 설정한다. 기본값은 데이터베이스 인코딩을 사용하는 것이다. Agens SQL 서버에서 지원하는 문자 집합은 6.3.1절에 나와 있다.

`lc_messages` (string)

메시지가 표시되는 언어를 설정한다. 허용되는 값은 시스템에 따라 다르다. 자세한 내용은 6.1절을 참조 바란다. 변수가 비어 있는 string으로 설정된 경우(기본값), 값은 시스템 의존적인 방법으로 서버의 실행 환경으로부터 상속된다.

일부 시스템에서 이 로케일(locale) 카테고리는 존재하지 않는다. 이 변수의 설정은 계속 유효하지만 아무런 효과는 없다. 또한 원하는 언어로 번역된 메시지가 존재하지 않을 수도 있다. 이런 경우 영어 메시지를 확인할 수 있다.

서버 로그 및 클라이언트로 전송되는 메시지에 영향을 줄 수 있고 부적절한 값이 서버 로그의 가독성을 해칠 수 있으므로 수퍼유저만 이 설정을 변경할 수 있다.

`lc_monetary` (string)

통화 형식 지정에 사용되는 로케일(locale)을 지정한다. 예를 들면, 함수의 `to_char` 계열을 사용한다. 허용되는 값은 시스템에 따라 다르다. 자세한 내용은 6.1절을 참조 바란다. 변수가 비어 있는 string으로 설정된 경우(기본값), 값은 시스템 의존적인 방법으로 서버의 실행 환경으로부터 상속된다.

`lc_numeric` (string)

숫자 형식 지정에 사용되는 로케일(locale)을 지정한다. 예를 들면, 함수의 `to_char` 계열을 사용한다. 허용되는 값은 시스템에 따라 다르다. 자세한 내용은 6.1절을 참조 바란다. 변수가 비어 있는 string으로 설정된 경우(기본값), 값은 시스템 의존적인 방법으로 서버의 실행 환경으로부터 상속된다.

`lc_time` (string)

날짜 및 시간 형식 지정에 사용되는 로케일(locale)을 지정한다. 예를 들면, 함수의 `to_char` 계열을 사용한다. 허용되는 값은 시스템에 따라 다르다. 자세한 내용은 6.1절을 참조 바란다.

`default_text_search_config` (string)

환경 설정을 지정하는 명시적 인수 없이, 텍스트 검색 함수의 변형에서 사용되는 텍스트 검색 환경 설정을 선택한다. 내장된 기본값은 `pg_catalog.simple`이지만 로케일(locale) 일치 환경 설정을 식별할 수 있는 경우 initdb는 선택된 `lc_ctype` 로케일(locale)에 해당되는 설정으로 환경 설정 파일을 초기화한다.

2.11.3. 공유 라이브러리 사전 로드

추가 기능을 로드하거나 성능상 이점을 위해 몇 가지 설정을 서버로의 공유 라이브러리 사전 로드에 사용할 수 있다. 예를 들면, '$libdir/mylib' 설정은 mylib.so(또는 일부 플랫폼에서, mylib.sl)가 설치 표준 라이브러리 디렉토리로부터 사전 로드되게 한다. 설정 간 차이라면 효과가 나타나는 시간과, 변경 시 필요한 권한이다.

Agens SQL 프로시저 언어 라이브러리는 이와 같은 방식으로 사전 로드할 수 있으며, 일반적으로 '$libdir/plXXX' 구문이 사용된다. 여기서 XXX는 pgsql, perl, tcl 또는 python이다.

각 매개변수별로 2개 이상의 라이브러리를 로드하는 경우 이름은 쉼표로 구분해야 한다. 모든 라이브러리 이름은 큰따옴표를 사용하지 않는 한 소문자로 변환된다.

특별히 Agens SQL과 함께 사용하려는 공유 라이브러리만 이와 같은 방식으로 로드할 수 있다. 모든 Agens SQL 지원 라이브러리는 호환성 보장을 검사하는 "magic block"이 있다. 이러한 이유로 비 Agens SQL 라이브러리는 이 방식으로 로드할 수 없다. 이를 위해 LD_PRELOAD 같은 운영 체제 기능을 사용할 수는 있다.

일반적으로 해당 모듈을 로드하는 권장 방법은 특정 모듈에 대한 문서를 참조 바란다.

local_preload_libraries (string)

이 변수는 연결 시작 시 사전 로드되는 공유 라이브러리를 하나 이상 지정한다. 특정 세션 시작 후에는 이 매개변수를 변경할 수 없다. 지정된 라이브러리를 찾지 못하면 연결 시도가 실패한다.

이 옵션은 사용자가 설정할 수 있다. 따라서 로드된 라이브러리는 설치의 표준 라이브러리 디렉토리의 plugins 서브 디렉토리에 나타나는 것으로 한정된다. ("안전한" 라이브러리만 설치되게 하는 것은 데이터베이스 관리자의 책임이다.) 예를 들면, $libdir/plugins/mylib 같이 local_preload_libraries의 항목은 이 디렉토리를 명시적으로 지정할 수 있으며, 또는 라이브러리 이름 mylib는 $libdir/plugins/mylib와 효과가 동일할 수 있다.

수퍼유저 이외에 사용자가 이러한 방식으로 사용할 수 있도록 모듈이 특수하게 디자인되어 있지 않는 한 이것은 일반적으로 올바른 설정 방법이 아니다. 대신 session_preload_libraries를 찾아 보기 바란다.

session_preload_libraries (string)

이 변수는 연결 시작 시 사전 로드되는 공유 라이브러리를 하나 이상 지정한다. 수퍼유저만 이 설정을 변경할 수 있다. 매개변수 값은 연결 시작 시에만 효과가 있다. 후속 변경은 무효하다. 지정된 라이브러리를 찾지 못하면 연결 시도가 실패한다.

이 기능은 명시적 **LOAD** 명령 없이 디버깅 또는 성능 평가 라이브러리가 특정 세션에 로드되도록 한다. 예를 들면, **ALTER ROLE SET**를 사용하여 이 매개변수를 설정하면 주어진 사용자 이름 하의 모든 세션에 대해 auto_explain의 활성화가 가능하다. 또한 이 매개변수는 서버 재시작 없이 변경 가능하므로(단, 세션을 새로 시작하는 경우에만 효과가 있다), 새 모듈을 모든 세션에 적용해야 하더라도 이렇게 추가하는 것이 쉽다.

shared_preload_libraries와 달리, 세션을 먼저 사용한 경우보다 세션 시작 시 라이브러리를 로딩하는 것이 큰 장점은 없다. 그래도, 연결 풀링을 사용하는 경우에는 장점이 약간 있기는 하다.

shared_preload_libraries (string)

이 변수는 서버 시작 시에 사전 로드할 하나 이상의 공유 라이브러리를 쉼표를 사용하여 지정한다. 이 매개변수는 서버 시작 시에만 설정 가능하다. 지정된 라이브러리를 찾지 못하면 서버 시작이 실패한다.

일부 라이브러리는 공유 메모리 할당, 경량 잠금 예약 또는 백그라운드 worker 시작 같은 postmaster 시작 시에만 일어날 수 있는 특정한 명령을 수행해야 한다. 해당 라이브러리는 이 매개변수를 통해 서버 시작 시에만 로드해야 한다. 자세한 내용은 각 라이브러리의 문서를 참조 바란다.

다른 라이브러리는 사전 로드할 수도 있다. 공유 라이브러리를 사전 로드해서 라이브러리를 먼저 사용하는 경우 라이브러리 시작 시간은 라이브러리가 처음 사용될 때 바뀐다. 단, 해당 프로세스가 라이브러리를 사용하지 않더라도 각각의 새로운 서버 프로세스 시작 시간이 약간 늘어날 수 있다. 따라서 이 매개변수는 대부분의 세션에서 사용되는 라이브러리인 경우에만 권장된다. 또한 이 매개변수를 변경하면 서버를 재시작해야 하므로 단기간 디버깅 시에는 설정이 바람직하지 않다. 대신 session_preload_libraries를 사용해야 한다.

> 참고: Windows 호스트에서 서버 시작 시 라이브러리를 사전 로드하는 것은 각각의 새 서버 프로세스를 시작하는 데 필요한 시간을 줄이지 않는다. 각 서버 프로세스는 모든 사전 로드 라이브러리를 리로드한다. 그러나, postmaster 시작 시에 명령을 수행해야 하는 라이브러리의 경우 `shared_preload_libraries`는 Windows 호스트에 여전히 유용하다.

2.11.4. 그 외 기본값

`dynamic_library_path` (string)

동적 로드 가능한 모듈을 열어야 하고 **CREATE FUNCTION** 또는 **LOAD** 명령에서 지정된 파일 이름에 디렉토리 성분이 없는 경우(예: 이름에 슬래시가 없음) 시스템은 필요한 파일에 대한 이 경로를 검색한다.

`dynamic_library_path`에 대한 값은 콜론(또는 Windows에서 세미콜론)으로 구분된 절대 디렉토리 경로 목록이어야 한다. 목록 요소가 특수 문자 `$libdir`로 시작되는 경우 컴파일된 Agens SQL 패키지 라이브러리 디렉토리가 $libdir에 대체된다. 이것은 표준 Agens SQL 배포에 의해 제공된 모듈이 설치되는 경우에 해당된다. (이 디렉토리의 이름을 찾으려면 `pg_config --pkglibdir`을 사용해야 한다.) 예를 들면:

```
dynamic_library_path = '/usr/local/lib/postgresql:/home/my_project/lib:$libdir'
```

또는 Windows 환경에서: or, in a Windows environment:

```
dynamic_library_path = 'C:\tools\postgresql;H:\my_project\lib;$libdir'
```

이 매개변수의 기본값은 `'$libdir'`이다. 값이 비어 있는 string으로 설정되면, 자동 경로 검색이 해제된다.

이 매개변수는 런타임 시 변경될 수 있지만 해당 설정은 클라이언트 연결이 끝날 때까지만 유지되므로 이 방법은 개발 단계에서 따로 준비해야 한다. 권장 방법은 `postgresql.conf` 환경 설정 파일에서 이 매개변수를 설정하는 것이다.

`gin_fuzzy_search_limit` (integer)

GIN 인덱스 스캔에 의해 리턴된 설정 크기에 대한 상한.

2.12. 잠금(lock) 관리

deadlock_timeout (integer)

이것은 데드록 상황인지 검사하기 전에 잠금을 대기하는 밀리초 단위의 시간이다. 데드록에 대한 검사는 상대적으로 고비용이므로 잠금을 기다릴 때마다 서버가 검사를 실행하지는 않는다. 실제 운영 중인 애플리케이션에서 데드록이 일반적이지는 않으며, 데드록 검사 전에 잠시 잠금을 기다리는 것이라고 긍정적으로 가정한다. 이 값을 늘리면 불필요한 데드록 검사를 위해 대기하는 시간이 줄어들지만 실제 데드록 에러 리포팅이 느려진다. 기본값은, 실제로 사용자가 희망하는 최소값일 가능성이 높은 1초이다(1s). 로드가 과도한 서버에서는 이 값을 올리고 싶을 것이다. 이상적으로, waiter가 데드록 검사를 결정하기 전에 잠금이 해제되는 이상한 상황을 개선하려면 이 설정은 사용자의 일반적인 트랜잭션 시간을 초과해야 한다. 수퍼유저만 이 설정을 변경할 수 있다.

log_lock_waits가 설정된 경우 이 매개변수는 잠금 대기에 대한 로그 메시지가 발생되기 전에 기다려야 하는 시간도 결정한다. 잠금 지연을 조사하려는 사용자라면 일반적인 deadlock_timeout보다 짧게 설정할 수 있다.

max_locks_per_transaction (integer)

공유 잠금 테이블은 max_locks_per_transaction * (max_connections + max_prepared_transactions) 개체(예: 테이블)에 대한 잠금을 추적하므로 이 개체 수 이하를 언제든 잠글 수 있다. 이 매개변수는 트랜잭션별로 할당된 평균 개체 잠금 수를 제어한다. 개별 트랜잭션은 모든 트랜잭션의 잠금이 잠금 테이블에 적합한 경우에 개체를 추가로 잠글 수 있다. 이것은 잠글 수 있는 행 수는 아니다. 해당 값은 무제한이다. 기본값으로 64가 충분한 것으로 입증되었지만, 쿼리가 단일 트랜잭션으로 서로 다른 여러 가지 테이블에 액세스하는 경우라면 이 값을 늘려야 할 수도 있다(예: 자식이 다수 있는 부모 테이블에 대한 쿼리). 이 매개변수는 서버 시작 시에만 설정 가능하다.

대기 서버 실행 중에 사용자는 이 매개변수를 마스터 서버 값보다 크거나 같게 설정해야 한다. 그렇지 않으면 대기 서버에서 쿼리가 허용되지 않는다.

max_pred_locks_per_transaction (integer)

공유 예측 잠금 테이블은 max_pred_locks_per_transaction * (max_connections + max_prepared_transactions) 개체(예: 테이블)에 대한 잠금을 추적하므로 이 개체 수 이하를 언제든 잠글 수 있다. 이 매개변수는 트랜잭션별로 할당된 평균 개체 잠금 수를 제어한다. 개별 트랜잭션은 모든 트랜잭션의 잠금이 잠금 테이블에 적합한 경우에 개체를 추가로 잠글 수 있다. 이것은 잠글 수 있는 행의 수는 아니다. 해당 값은 무제한이다. 기본값으로 64가 테스트 시 충분한 것으로 입증되었지만, 클라이언트가 직렬화 가능한 트랜잭션으로 서로 다른 여러 가지 테이블에 액세스하는 경우라면 이 값을 늘려야 할 수도 있다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

2.13. 버전 및 플랫폼 호환성

2.13.1. 이전 Agens SQL 버전

array_nulls (boolean)

이것은 따옴표 없는 NULL을 null 배열 요소로 지정하는 것으로 배열 입력 파서에 인식시킬 것인지를 제어한다. 기본적으로 이것은, null 값이 포함된 배열값의 입력을 허용하는 on이다. 단, 8.2 이전 버전의 Agens SQL은 배열에서 null 값을 지원하지 않으며, 따라서

string 값 "NULL"을 사용하는 일반적인 배열 요소를 지정하는 것으로 NULL을 처리한다. 예전 동작을 필요로 하는 어플리케이션 이전 버전과의 호환성 때문에 이 변수를 off로 설정할 수 있다.

이 변수가 off인 경우에도 null 값을 포함한 배열 변수를 생성하는 것도 가능하다.

backslash_quote (enum)

이것은 string 리터럴에서 따옴표를 \'로 표기할 수 있는지를 제어한다. 따옴표를 SQL 표준 방식으로 표기하는 방법은 이중으로 사용하는 것('')이지만 Agens SQL은 오래 전부터 \'도 허용해 왔다. 그러나, \'를 사용하면 일부 클라이언트 문자 집합 인코딩에서 마지막 바이트가 수치상 ASCII \와 동일한 멀티바이트 문자가 있기 때문에 보안상 위험하다. 클라이언트 측 코드가 잘못 이스케이프할 경우 SQL 인젝션 공격이 가능하다. 이러한 위험은, 역슬래시에 의해 이스케이프되는 따옴표가 있는 경우에 서버가 쿼리를 거부하도록 함으로써 예방할 수 있다. backslash_quote의 허용 값은 on(항상 \' 허용), off(항상 거부) 및 safe_encoding(클라이언트 인코딩이 멀티바이트 문자 내에서 ASCII \를 수락하지 않는 경우에만 허용)이다. 기본 설정은 safe_encoding이다.

표준 준수 string 리터럴에서, \는 \라는 점에 유의해야 한다. 이 매개변수는 이스케이프 string 구문(E'...')을 비롯한 비 표준 준수 리터럴의 처리에만 영향을 준다.

default_with_oids (boolean)

이것은 WITH OIDS 또는 WITHOUT OIDS가 지정되지 않은 경우 **CREATE TABLE** 및 **CREATE TABLE AS**가 새로 생성된 테이블에서 OID 칼럼을 포함할 것인지를 제어한다. 또한 OID를 **SELECT INTO**에 의해 생성된 테이블에 포함할 것인지를 결정한다. 이 매개변수는 기본적으로 off이며, Agens SQL 8.0 이전 버전에서는 기본적으로 on이다.

사용자 테이블에서 OID의 사용은 사용 가치가 없는 것(deprecated)으로 간주되므로 대부분의 설치는 이 변수를 비활성화해야 한다. 특별한 테이블에 OID가 필요한 애플리케이션은 테이블 생성 시 WITH OIDS를 지정해야 한다. 이 변수는 이 동작을 따르지 않는 예전 애플리케이션과의 호환성을 위해 활성화할 수 있다.

escape_string_warning (boolean)

on인 경우 역슬래시(\)가 일반 string 리터럴('...' 구문)에 있고 standard_conforming_strings가 off인 경우 경고가 나타난다. 기본값은 on이다.

SQL 표준에 따라 일반 string의 기본 동작은 역슬래시를 일반 문자로 처리하기 때문에 역슬래시를 이스케이프로 사용하려는 애플리케이션은 이스케이프 string 구문(E'...')을 수정해야 한다. 이 변수를 활성화하면 수정이 필요한 코드를 찾는 데 도움이 된다.

lo_compat_privileges (boolean)

9.0 이전의 Agens SQL 릴리스에서 거대(large) 개체는 액세스 권한이 없었으며, 항상 모든 사용자에 의해 읽기 및 쓰기가 가능했었다. 이 변수를 on으로 설정하면 이전 릴리스와의 호환성에 대한 새 권한 검사가 비활성화된다. 기본값은 off이다. 수퍼유저만 이 설정을 변경할 수 있다.

이 변수를 설정한다고 해서 거대(large) 개체와 관련된 모든 보안 검사가 비활성화되는 것은 아니다. 기본 동작에 대한 것만 Agens SQL 9.0에서 변경되었다. 예를 들면, lo_import() 및 lo_export()는 이 설정과 무관하게 수퍼유저 권한이 필요하다.

quote_all_identifiers (boolean)

데이터베이스가 SQL을 생성하는 경우 (현재) 키워드가 없더라도 모든 식별자에서 따옴표를 사용해야 한다. 이것은 **EXPLAIN** 출력과 pg_get_viewdef 같은 함수의 결과에 영향을 준다. pg_dump 및 pg_dumpall의 --quote-all-identifiers 옵션도 참조 바란다.

sql_inheritance (boolean)

이 설정은 장식이 없는 테이블 참조가 상속 자식 테이블을 포함하는 것으로 간주할 것인지를 제어한다. 기본값은, 자식 테이블이 포함됨을 의미하는 on이다(따라서 * 접미사가 기본적으로 추정된다). off으로 설정되면 자식 테이블이 포함되지 않는다(따라서 ONLY 접두사가 추정된다). SQL 표준은 포함할 자식 테이블이 필요하므로, off 설정은 규격에 맞지 않지만 7.1 이전 Agens SQL 릴리스와의 호환성을 위해 제공된다.

sql_inheritance를 off로 설정하는 것은 동작이 오류가 발생하기 쉽고 SQL 표준에도 반하기 때문에 의미가 없다(deprecated). 이 설명서 다른 곳에서 논의된 상속 동작은 보통 on인 것으로 가정한다.

standard_conforming_strings (boolean)

이것은 대개 string 리터럴('...')이 역슬래시를 SQL 표준에 지정된 대로 문자 그대로 처리할 것인지 제어한다. Agens SQL 9.1 초반에는 기본값이 on이다(이전 릴리스에서는 기본값이 off). 애플리케이션은 string 리터럴 처리하는 방법을 결정하는 이 매개변수를 검사한다. 이 매개변수가 존재하면 이스케이프 string 구문(E'...')이 지원되는 것으로 볼 수 있다. 애플리케이션이 역슬래시를 이스케이프 문자로 처리하기를 원할 경우에는 이 스케이프 string 구문을 사용해야 한다.

synchronize_seqscans (boolean)

이것은 거대(large) 테이블의 순차 스캔을 허용하여 서로를 동기화하므로 동시 스캔은 동일한 시간에 동일한 블록을 읽고 I/O 작업 부하를 공유한다. 이것이 활성화되면 스캔이 테이블 중간에서 시작되며, 이미 진행 중인 스캔 활동과 동기화되도록 모든 행을 끝까지 "랩어라운드"한다. 이것은 결과적으로 ORDER BY 절이 없는 쿼리에 의해 리턴된 행 정렬에서 예측 불가능한 변경으로 이어진다. 이 매개변수를 off로 설정하면 순차 스캔이 항상 테이블 초반에 시작되는 8.3 이전의 동작이 보장된다. 기본값은 on이다.

2.13.2. 플랫폼 및 클라이언트 호환성

transform_null_equals (boolean)

on으로 설정되면 양식의 표현식 *expr* = NULL(또는 NULL = *expr*)은 *expr* IS NULL로 처리된다. 즉, *expr*가 null 값으로 평가되면 true가 리턴되고, 그 외에는 false가 리턴된다. *expr* = NULL의 올바른 SQL 규격 호환 동작은 항상 null을 리턴한다(알 수 없음). 그러므로 이 매개변수의 기본값은 off이다.

그러나, Microsoft Access에서 필터링된 양식은 *expr* = NULL을 사용하여 null 값을 테스트하기 위한 쿼리를 생성하므로 해당 인터페이스를 사용하여 데이터베이스에 액세스하는 경우 사용자는 이 옵션을 on으로 설정하고자 할 수 있다. 표현식 *expr* = NULL은 항상 null 값을 리턴하므로(SQL 표준 해석 사용), 그다지 유용하지 않고, 일반적인 애플리케이션에 거의 나타나지 않으므로 실제로 이 옵션은 해가 되지 않는다. 하지만 새 사용자는 null 값이 관련된 표현식을 헷갈려 하므로 이 옵션은 기본적으로 off로 설정되어 있다.

이 옵션은 정확한 양식 = NULL에만 영향을 미치며, 다른 비교 연산자 또는 동일한 연산자(예: IN)와 관련된 몇몇 표현식과 계산상 동일한 다른 표현식에는 영향을 미치지는 않는다. 따라서 이 옵션이 나쁜 프로그래밍에 대한 해결책은 아니다.

2.14. 에러 처리

`exit_on_error` (boolean)

true인 경우 어떤 에러가 발생했든 현재 세션이 중단된다. 기본적으로 이것은 false로 설정되고 FATAL 에러 시 세션이 중단된다.

`restart_after_crash` (boolean)

기본값인 true로 설정되면 백엔드 충돌 후 Agens SQL이 자동으로 재초기화된다. 이 값을 계속 true로 설정하는 것은 일반적으로 데이터베이스의 가용성을 최대화하는 최고의 방법이다. 그러나, 클러스터웨어에서 Agens SQL를 호출하는 경우처럼 경우에 따라 재시작을 비활성화해서 클러스터웨어가 제어를 획득하고 적절하다고 생각되는 조치를 취할 수 있게 하는 점이 유용하다.

2.15. 프리셋 옵션

다음 "매개변수"는 읽기 전용이며, Agens SQL이 컴파일 되거나 설치된 경우에만 결정된다. 따라서 `postgresql.conf` 샘플 파일에서는 제외되었다. 이 옵션은 특정 애플리케이션, 특히 관리 프론트 엔드에 도움이 되는 Agens SQL 동작의 다양한 측면을 보여준다.

`block_size` (integer)

디스크 블록의 크기를 알려준다. 이것은 서버 빌드 시 `BLCKSZ` 값에 의해 결정된다. 기본값은 8192바이트이다. 일부 환경 설정 변수(예: shared_buffers)의 의미는 `block_size`의 영향을 받는다. 자세한 내용은 2.4절을 참조 바란다.

`data_checksums` (boolean)

데이터 체크섬이 이 클러스터에 대해 활성화되었는지를 알려준다. 자세한 내용은 데이터 체크섬을 참조 바란다.

`integer_datetimes` (boolean)

Agens SQL이 64비트 integer 날짜 및 시간으로 빌드되었는지를 알려준다. Agens SQL 빌드 시 `--disable-integer-datetimes`로 환경 설정하면 이것을 비활성화할 수 있다. 기본값은 `on`이다.

`lc_collate` (string)

텍스트 데이터의 정렬 로케일(locale)을 알려준다. 자세한 내용은 6.1절을 참조 바란다. 이 값은 데이터베이스를 생성할 때 결정된다.

`lc_ctype` (string)

문자 분류를 결정하는 로케일(locale)을 알려준다. 자세한 내용은 6.1절을 참조 바란다. 이 값은 데이터베이스를 생성할 때 결정된다. 대개는 `lc_collate`와 동일하지만 특수한 애플리케이션의 경우 다르게 설정될 수 있다.

`max_function_args` (integer)

함수 인수의 최대 수를 알려준다. 이것은 서버 빌드 시 `FUNC_MAX_ARGS` 값에 의해 결정된다. 기본값은 100개 인수이다.

max_identifier_length (integer)

최대 식별자 길이를 알려준다. 이것은 서버 빌드 시 NAMEDATALEN 값보다 하나 작게 결정된다. NAMEDATALEN의 기본값이 64이므로 max_identifier_length 기본값은 63바이트이며, 이것은 멀티바이트 인코딩 시 63자 미만일 수 있다.

max_index_keys (integer)

인덱스 키의 최대 수를 알려준다. 이것은 서버 빌드 시 INDEX_MAX_KEYS 값에 의해 결정된다. 기본값은 32개 키이다.

segment_size (integer)

파일 세그먼트 내에서 저장할 수 있는 블록(페이지)의 수를 알려준다. 이것은 서버 빌드 시 RELSEG_SIZE 값에 의해 결정된다. 세그먼트 파일의 최대 크기(바이트 단위)는 block_size를 곱한 segment_size와 같으며, 기본값은 1GB이다.

server_encoding (string)

데이터베이스 인코딩(문자 집합)을 알려준다. 데이터베이스를 생성할 때 결정된다. 대개, 클라이언트는 client_encoding 값만 사용해서 연결해야 한다.

server_version (string)

서버의 버전 번호를 알려준다. 이것은 서버 빌드 시 PG_VERSION 값에 의해 결정된다.

server_version_num (integer)

서버의 버전 번호를 integer로 알려준다. 이것은 서버 빌드 시 PG_VERSION_NUM 값에 의해 결정된다.

wal_block_size (integer)

WAL 디스크 블록의 크기를 알려준다. 이것은 서버 빌드 시 XLOG_BLCKSZ 값에 의해 결정된다. 기본값은 8192바이트이다.

wal_segment_size (integer)

WAL 세그먼트 파일 내에서 블록(페이지)의 수를 알려준다. WAL 세그먼트 파일의 총 크기(바이트 단위)는 wal_block_size를 곱한 wal_segment_size와 같으며, 기본값은 16GB이다. 자세한 내용은 13.4절을 참조 바란다.

2.16. 커스텀 옵션

이 기능은 일반적으로 Agens SQL이 알지 못하는 매개변수를 추가 모듈로 추가할 수 있게 한다(예: 프로시저 언어). 이렇게 하면 표준 방식으로 확장 모듈을 환경 설정할 수 있다.

커스텀 옵션은 확장명 다음에 점, 다음에 적절한 매개변수 이름의 두 부분으로 환경 설정되며, SQL의 정규화된 이름과 아주 유사하다. 예를 들면, plpgsql.variable_conflict와 같다.

커스텀 옵션은 관련 확장 모듈이 로드되지 않은 프로세스에서 설정되어야 하므로, Agens SQL은 두 부분의 매개변수 이름에 대한 설정을 허용한다. 해당 변수는 플레이스 홀더로 처리되며 변수를 정의하는 모듈이 로드되기 전까지는 함수를 갖지 않는다. 확장 모듈이 로드된 경우 변수 정의가 추가되고, 해당 변수에 따라 플레이스 홀더 값이 변환되고, 확장명으로 시작되는데 인식되지 않는 플레이스 홀더에 대해 경고를 보낸다.

2.17. 개발자 옵션

다음 매개변수는 Agens SQL 소스 코드에서 사용할 수 있으며 경우에 따라 심각한 데이터베이스 손상을 복구하는 데에도 도움이 된다. 실제 운영 중인 데이터베이스에서 이 매개변수를 사용할 이유는 없다. 따라서 `postgresql.conf` 샘플 파일에서는 제외되었다. 이러한 매개변수 다수는 어쨌든 제대로 작동하려면 특수한 소스 컴파일 플래그가 필요하다.

`allow_system_table_mods` (boolean)

시스템 테이블의 구조 수정을 허용한다. 이것은 **initdb**에서 사용된다. 이 매개변수는 서버 시작 시에만 설정 가능하다.

`debug_assertions` (boolean)

다양한 assertion 검사를 사용으로 설정한다. 이것은 디버깅 시 도움이 된다. 이상한 문제가 발생하거나 충돌이 있는 경우 프로그래밍 실수에 의한 것일 수 있으므로 사용자는 이것을 사용 설정하려고 할 수 있다. 이 매개변수를 사용하려면 Agens SQL을 빌드할 때 매크로 USE_ASSERT_CHECKING을 정의해야 한다(**configure** 옵션 `--enable-cassert`로 가능). assertion이 활성화된 상태에서 Agens SQL을 빌드하면 `debug_assertions`의 기본값은 `on`이 된다.

`ignore_system_indexes` (boolean)

시스템 테이블을 읽을 때 시스템 인덱스를 무시한다(그러나, 테이블을 수정하면 인덱스는 계속 수정된다). 이것은 손상된 시스템 인덱스를 복구할 때 유용하다. 이 매개변수는 세션 시작 후에는 변경할 수 없다.

`post_auth_delay` (integer)

0이 아닌 경우 인증 절차를 수행한 후 새 서버 프로세스가 시작되면 이 초 단위 수만큼의 지연이 발생한다. 이것은 개발자가 디버거를 사용하여 서버 프로세스에 접속할 수 있는 기회를 제공하기 위한 것이다. 이 매개변수는 세션 시작 후에는 변경할 수 없다.

`pre_auth_delay` (integer)

0이 아닌 경우 인증 절차를 수행하기 전 새 서버 프로세스가 시작된 직후 이 초 단위 수만큼의 지연이 발생한다. 이것은 개발자가 디버거를 사용하여 인증 오류를 추적할 수 있도록 서버 프로세스에 접속할 수 있는 기회를 제공하기 위한 것이다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

`trace_notify` (boolean)

LISTEN 및 **NOTIFY** 명령에 대한 대량의 디버깅 출력을 생성한다. 이 출력을 클라이언트 또는 서버 로그에 각각 전송하려면 client_min_messages 또는 log_min_messages는 `DEBUG1`이어야 한다.

`trace_recovery_messages` (enum)

복구 관련 디버깅 출력의 로깅을 활성화한다. 그 외에는 로깅되지 않는다. 이 매개변수는 사용자가 log_min_messages의 일반 설정을 오버라이드할 수 있지만, 특정 메시지에만 해당된다. 이것은 핫 스탠바이에서 사용된다. 유효 값은 `DEBUG5` 및 `DEBUG4`, `DEBUG3`, `DEBUG2`, `DEBUG1`, `LOG`이다. 기본적으로, `LOG`는 로깅 결정에 전혀 영향을 주지 않는다. 다른 값들은 `LOG` 우선 순위가 있어도 해당 우선 순위보다 높은 복구 관련 디버그 메시지를 발생시킨다. `log_min_messages`의 공통 설정의 경우 이것은 서버 로그로 무조건 메시지를 전송한다. 이 매개변수는 `postgresql.conf` 파일 또는 서버 커맨드 라인에서만 설정 가능하다.

`trace_sort` (boolean)

on으로 설정된 경우 정렬 명령 중에 리소스 사용량에 대한 정보를 출력한다. 이것은 Agens SQL이 컴파일된 경우 TRACE_SORT 매크로가 정의된 경우에만 사용할 수 있다. (단, TRACE_SORT는 현재 기본적으로 정의된다.)

`trace_locks` (boolean)

on으로 설정된 경우 정렬 명령 중에 잠금 사용에 대한 정보를 출력한다. 덤프된 정보에는 잠금 명령, 잠금 유형, 잠금 또는 잠금 해제된 개체의 고유 식별자가 포함된다. 이 개체에 이미 부여된 잠금 유형 및 이 개체를 기다리는 잠금 유형에 대한 비트 마스크도 포함된다. 각 잠금 유형의 경우 부여된 잠금 및 대기 중인 잠금 수에 대한 카운트 및 총계도 덤프된다. 로그 파일 출력에 대한 예제는 다음과 같다.

```
LOG:LockAcquire: new: lock(0xb7acd844) id(24688,24696,0,0,0,1)
grantMask(0) req(0,0,0,0,0,0,0)=0 grant(0,0,0,0,0,0,0)=0
wait(0) type(AccessShareLock)
LOG:GrantLock: lock(0xb7acd844) id(24688,24696,0,0,0,1)
grantMask(2) req(1,0,0,0,0,0,0)=1 grant(1,0,0,0,0,0,0)=1
wait(0) type(AccessShareLock)
LOG:UnGrantLock: updated: lock(0xb7acd844) id(24688,24696,0,0,0,1)
grantMask(0) req(0,0,0,0,0,0,0)=0 grant(0,0,0,0,0,0,0)=0
wait(0) type(AccessShareLock)
LOG:CleanUpLock: deleting: lock(0xb7acd844) id(24688,24696,0,0,0,1)
grantMask(0) req(0,0,0,0,0,0,0)=0 grant(0,0,0,0,0,0,0)=0
wait(0) type(INVALID)
```

덤프되는 구조에 대한 자세한 내용은 `src/include/storage/lock.h`에서 찾을 수 있다.

이것은 Agens SQL이 컴파일된 경우 LOCK_DEBUG 매크로가 정의된 경우에만 사용할 수 있다.

`trace_lwlocks` (boolean)

on으로 설정된 경우 정렬 명령 중에 가벼운 잠금(lightweight lock) 사용에 대한 정보를 출력한다. 가벼운 잠금(lightweight lock)은 주로 공유 메모리 데이터 구조에 대한 상호 배제 액세스를 제공하기 위함이다.

이것은 Agens SQL이 컴파일된 경우 LOCK_DEBUG 매크로가 정의된 경우에만 사용할 수 있다.

`trace_userlocks` (boolean)

on으로 설정된 경우 정렬 명령 중에 사용자 잠금(user lock) 사용에 대한 정보를 출력한다. 보조 잠금(advisory locks)인 경우에만 출력이 trace_locks와 동일하다.

이것은 Agens SQL이 컴파일된 경우 LOCK_DEBUG 매크로가 정의된 경우에만 사용할 수 있다.

`trace_lock_oidmin` (integer)

설정된 경우 이 OID 아래 테이블에 대해 잠금을 추적하지 않는다. (시스템 테이블에 출력을 방지하려고 사용)

이것은 Agens SQL이 컴파일된 경우 LOCK_DEBUG 매크로가 정의된 경우에만 사용할 수 있다.

`trace_lock_table` (integer)

이 테이블(OID)에 대한 잠금을 무조건 추적한다.

이것은 Agens SQL이 컴파일된 경우 LOCK_DEBUG 매크로가 정의된 경우에만 사용할 수 있다.

`debug_deadlocks` (boolean)

설정된 경우 데드락 타임아웃 발생 시 현재 모든 잠금에 대한 정보를 덤프한다.

이것은 Agens SQL이 컴파일된 경우 LOCK_DEBUG 매크로가 정의된 경우에만 사용할 수 있다.

`log_btree_build_stats` (boolean)

설정된 경우 다양한 B-트리 명령에서 시스템 리소스 사용량 통계(메모리 및 CPU)를 로깅한다.

이것은 Agens SQL이 컴파일된 경우 BTREE_BUILD_STATS 매크로가 정의된 경우에만 사용할 수 있다.

`wal_debug` (boolean)

설정된 경우 WAL 관련 디버깅 출력을 내보낸다. 이것은 Agens SQL이 컴파일된 경우 WAL_DEBUG 매크로가 정의된 경우에만 사용할 수 있다.

`ignore_checksum_failure` (boolean)

data checksums가 활성화된 경우에만 효과가 있다.

읽기 중 체크섬 실패를 감지하면 Agens SQL이 에러를 보고하고, 현재 트랜잭션을 중단한다. `ignore_checksum_failure`를 on으로 설정하면 시스템이 실패(그래도 경고는 보고된다)를 무시하고 프로세싱을 계속한다. 이러한 동작은 충돌 또는 충돌 전파, 숨김, 기타 심각한 문제를 초래한다. 그러나, 블록 헤더가 온전한 경우에는 에러를 무시하고 테이블에 잔존해 있을 수 있는 미손상 튜플을 검색할 수 있다. 블록 헤더가 손상된 경우 이 옵션이 활성화돼도 에러가 보고된다. 기본값은 `off`이고 수퍼유저에 의해서만 변경 가능하다.

`zero_damaged_pages` (boolean)

손상된 페이지 헤더가 감지되면 Agens SQL이 에러를 보고하고 현재 트랜잭션을 중단한다. `zero_damaged_pages`를 on으로 설정하면 시스템이 대신 경고를 보고하고, 메모리에서 손상된 페이지를 0으로 처리하고, 프로세싱을 계속한다. 이러한 작업은 손상된 페이지의 모든 행, 즉 데이터를 소멸시킨다. 그러나 이것을 이용하면 사용자는 에러를 무시하고 테이블에 남아 있을 수 있는 미손상 페이지에서 행을 검색할 수 있다. 이것은 하드웨어 또는 소프트웨어 에러에 의한 손상이 발생된 경우 데이터를 복구할 때 유용하다. 테이블의 손상된 페이지에서 데이터를 복구하지 않기로 결정하기 전에는 이것을 on으로 설정하면 안 된다. 0으로 처리된 페이지는 디스크에 강제로 쓰여지지 않으므로 이 매개변수를 off로 다시 설정하기 전에 테이블 또는 인덱스를 재생하는 것이 좋다. 기본값은 `off`이고 수퍼유저에 의해서만 변경 가능하다.

2.18. 단축 옵션

편의상 일부 매개변수에서는 1글자 커맨드 라인 옵션 스위치를 사용할 수 있다. 표 2-2에 나와 있다. 이 옵션 중 일부는 그 존재 이유에 나름의 내력이 있으며, 1글자 옵션이라고 해서 많이 사용해도 되는 것은 아니다.

표 **2-2.** 단축 옵션 키

단축 옵션	동등
-A *x*	debug_assertions = *x*
-B *x*	shared_buffers = *x*
-d *x*	log_min_messages = DEBUG*x*
-e	datestyle = euro
-fb, -fh, -fi, -fm, -fn, -fo, -fs, -ft	enable_bitmapscan = off, enable_hashjoin = off, enable_indexscan = off, enable_mergejoin = off, enable_nestloop = off, enable_indexonlyscan = off, enable_seqscan = off, enable_tidscan = off
-F	fsync = off
-h *x*	listen_addresses = *x*
-i	listen_addresses = '*'
-k *x*	unix_socket_directories = *x*
-l	ssl = on
-N *x*	max_connections = *x*
-O	allow_system_table_mods = on
-p *x*	port = *x*
-P	ignore_system_indexes = on
-s	log_statement_stats = on
-S *x*	work_mem = *x*
-tpa, -tpl, -te	log_parser_stats = on, log_planner_stats = on, log_executor_stats = on
-W *x*	post_auth_delay = *x*

3장. 클라이언트 인증

클라이언트 애플리케이션이 데이터베이스 서버에 연결하는 경우 어떤 Agens SQL 데이터베이스 사용자 이름으로 연결할 것인지 지정하는데, 이것은 특정 사용자로 Unix 컴퓨터에 로그인하는 것과 매우 유사하다. SQL 환경 내에서 사용 중인 데이터베이스 사용자 이름은 데이터베이스 개체에 대한 액세스 권한을 결정한다. 자세한 내용은 4장을 참고 바란다. 따라서 연결 가능한 데이터베이스 사용자를 제한하는 것이 중요하다.

> 참고: 4장에서 설명한 대로 Agens SQL은 실제로 "role" 면에서 권한을 관리한다. 이 장에서는 *데이터베이스 사용자*가 "`LOGIN` 권한이 있는 role"이라는 의미로 쓰인다.

*인증*은 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.

Agens SQL는 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.

Agens SQL 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.

3.1. The `pg_hba.conf` 파일

클라이언트 인증은 전통적으로 이름이 `pg_hba.conf`이고 데이터베이스 클러스터의 데이터 디렉토리에 저장되는 환경 설정 파일로 제어된다. (HBA는 호스트 기반 인증(host-based authentication)의 약어이다.) 기본 `pg_hba.conf` 파일은 데이터 디렉토리가 **initdb**로 초기화될 때 설치된다. 인증 환경 설정 파일을 다른 곳에 배치하는 것도 가능하다. hba_file 환경 설정 파일을 참조 바란다.

`pg_hba.conf` 파일의 일반 형식은 한 줄당 하나씩 있는 레코드 세트이다. 빈 줄은 무시된다. `#` 주석 문자 뒤의 텍스트도 무시된다. 레코드는 줄을 바꿔서 이어질 수 없다. 레코드는 여러 개의 필드로 구성되며, 공백 및/또는 탭으로 구분된다. 필드 값에 큰 따옴표를 사용하면 필드에 공백을 포함할 수 있다. 데이터베이스 또는 사용자, 주소 필드의 키워드에 따옴표를 사용하면(예: `all` 또는 `replication`) 단어는 자체의 특수한 의미를 상실하고 해당 이름의 데이터베이스, 사용자 또는 호스트와 일치하게 된다.

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 databaseuserauth-method[auth-options]
host  databaseuseraddressauth-method[auth-options]
hostssldatabaseuseraddressauth-method[auth-options]
```

```
hostnossldatabaseuseraddressauth-method[auth-options]
host databaseuserIP-addressIP-maskauth-method[auth-options]
hostssldatabaseuserIP-addressIP-maskauth-method[auth-options]
hostnossldatabaseuserIP-addressIP-maskauth-method[auth-options]
```

필드의 의미는 다음과 같다.

`local`

이 레코드는 Unix 도메인 소켓을 사용한 연결 시도와 일치한다. 이러한 유형의 레코드 없이 Unix 도메인 소켓 연결은 불가능 하다.

`host`

이 레코드는 TCP/IP를 사용한 연결 시도와 일치한다. `host` 레코드는 SSL 연결 시도 혹은 비 SSL 연결 시도와 일치한다.

> 참고**:** 기본 동작이 로컬 루프백 주소인 `localhost`에 대해서만 TCP/IP 연결을 listen하는 것이므로 서버가 적절한 listen_addresses 값으로 시작되지 않으면 원격 TCP/IP 연결이 불가능하다.

`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(자세한 내용은 1.9절 참조).

`hostnossl`

이 레코드 유형은 `hostssl`과는 반대로 동작한다. SSL을 사용하지 않는 TCP/IP 상의 연결 시도에 대해서만 일치한다.

`database`

이 레코드가 일치하는 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. (`samegroup`은 폐지되었지만 `samerole`은 계속 쓸 수 있다.) 수퍼유저는 직접 혹은 간접적으로 role의 명시적인 멤버가 아닐 경우, 단지 수퍼유저라는 이유로 `samerole`에 대한 role의 멤버로 간주되지 않는다. `replication` 값은 복제 연결이 요청되는 경우 레코드가 일치하도록 지정한다(복제 연결은 특정 데이터베이스를 지정하지는 않는다). 이 경우가 아니라면 특정 Agens SQL 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. (Agens SQL에서는 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, 수퍼유저는 단지 수퍼유저라는 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 `10.6.0.0/16`일 수 있다. `0.0.0.0/0`은 모든 IPv4 주소를 나타내며 `::/0`은 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.

IPv4 형식의 IP 주소는 해당 주소의 IPv6 연결과 일치한다. 예를 들면, `127.0.0.1`은 IPv6 주소 `::ffff:127.0.0.1`과 일치하게 된다. IPv6 형식의 항목은 표시된 주소가 IPv4-in-IPv6 범위 내이더라도 IPv6 연결만 일치하게 된다. IPv6 형식의 항목은 시스템의 C 라이브러리가 IPv6 주소를 지원하지 않는 경우 거부된다.

사용자는 아무 IP 주소나 일치하도록 `all`을 쓸 수도 있고, 서버의 자체 IP 주소 아무거나 일치하도록 `samehost`를 쓸 수도 있고, 서버가 직접 연결되는 서브넷의 아무 주소나 일치하도록 `samenet`을 쓸 수도 있다.

호스트 이름이 지정된 경우(IP 주소가 아니거나 특수 키워드가 호스트 이름으로 처리되는 모든 것) 해당 이름은 클라이언트 IP 주소의 역방향 이름 분석 결과와 비교된다(예: DNS가 사용되는 경우 역방향 DNS 조회). 호스트 이름 비교는 대소문자를 구분하지 않는다. 일치하는 호스트 이름이 있는 경우, 호스트 이름을 순방향 이름 분석(예: 순방향 DNS 조회)해서 클라이언트의 IP 주소와 동일한지 검사한다. 양방향으로 일치할 경우 항목이 일치하는 것으로 간주된다. (`pg_hba.conf`에서 사용되는 호스트 이름은 클라이언트 IP 주소의 주소-이름 분석(address-to-name resolution)이 리턴한 것이어야 하며, 리턴된 값이 아니면 일치할 수 없다.) 일부 호스트 이름 데이터베이스는 IP 주소를 호스트 이름 여러 개와 연결하는 것을 허용하지만, IP 주소를 분석하도록 요청된 경우 운영 체제는 호스트 이름을 하나만 리턴한다.

점(`.`)으로 시작되는 호스트 이름 규격은 실제 호스트 이름의 접미사와 일치한다. 따라서, `.example.com`은 `foo.example.com`과 일치하게 된다(`example.com`만으로는 일치하지 않음).

호스트 이름이 `pg_hba.conf`에 지정된 경우 이름 분석 속도가 빠른지 확인해야 한다. **nscd** 같은 로컬 이름 분석 캐시를 설정하는 것이 유리할 수 있다. 또한 사용자는 환경 설정 매개 변수 `log_hostname`을 활성화하여 로그의 IP 주소 대신 클라이언트의 호스트 이름을 볼 수 있다.

이 필드는 `host`, `hostssl` 및 `hostnossl` 레코드에 적용된다.

> 클라이언트 IP 주소의 역방향 조회를 비롯한 두 이름 분석 방법이 이렇게 복잡한 방식으로 호스트 이름을 처리하는 이유를 궁금해 하는 사용자도 있다. 클라이언트의 역방향 DNS 항목이 설정되지 않았거나 올바르지 않은 호스트 이름을 넘겨주는 경우에 사용법이 복잡해진다. 이것은 기본적으로 효율을 위한 것이다. 이와 같은 연결 시도는 기껏해야 두 가지 리졸버(resolver) 조회(역방향 하나 및 순방향 하나)를 시도한다. 일부 주소에 리졸버(resolver) 문제가 있는 경우 이것은 해당 클라이언트만의 문제이다. 순방향 조회만 수행하면 `pg_hba.conf`에 나오는 모든 호스트 이름을 연결 시도할 때마다 분석해야 한다. 이름이 많을 경우 속도가 매우 느려진다. 그리고, 호스트 이름 중 하나라도 리졸버(resolver) 문제가 있는 경우 이것은 전체의 문제가 된다.
>
> 또한, 패턴 일치를 위해서는 실제 클라이언트 호스트 이름을 알고 있어야 하므로 역방향 조회는 접미사 일치 기능을 구현해야 한다.
>
> 이러한 동작은 Apache HTTP 서버 및 TCP 래퍼 같은 다른 유명한 호스트 이름 기반의 액세스 제어 구현과 같다.

IP-address
IP-mask

이 필드는 *CIDR-address* 표기의 대안으로 사용될 수 있다. 마스크 길이를 지정하는 대신 실제 마스크가 쉼표로 구분하여 지정된다. 예를 들면, `255.0.0.0`은 IPv4 CIDR 마스크 길이 8을 나타내고, `255.255.255.255`는 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 선택안이 여기에 요약되어 있다. 자세한 내용은 3.3절을 참조 바란다.

`trust`

무조건 연결을 허용한다. 이 방법은 패스워드나 다른 인증 없이 임의의 Agens SQL 데이터베이스 사용자로 로그인하여 누구나 Agens SQL 데이터베이스 서버에 연결할 수 있다. 자세한 내용은 3.3.1절을 참조 바란다.

`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. 자세한 내용은 3.3.2절을 참조 바란다.

`password`

클라이언트가 인증을 위해 암호화되지 않은 패스워드를 제공해야 한다. 패스워드는 네트워크 상에서 일반 텍스트로 전송되므로 신뢰하지 않는 네트워크에서 이것을 사용하면 안 된다. 자세한 내용은 3.3.2절을 참조 바란다.

`gss`

GSSAPI를 사용하여 사용자를 인증한다. 이것은 TCP/IP 연결에서만 사용할 수 있다. 자세한 내용은 3.3.3절을 참조 바란다.

`sspi`

SSPI를 사용하여 사용자를 인증한다. 이것은 Windows에서만 사용할 수 있다. 자세한 내용은 3.3.4절을 참조 바란다.

`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. 자세한 내용은 3.3.5절을 참조 바란다.

`peer`

클라이언트의 운영 체제 사용자 이름을 운영 체제에서 획득하고, 요청된 데이터베이스 사용자 이름과 일치하는지 확인한다. 이것은 로컬 연결에서만 사용할 수 있다. 자세한 내용은 3.3.6절을 참조 바란다.

`ldap`

LDAP 서버를 사용하여 인증한다. 자세한 내용은 3.3.7절을 참조 바란다.

`radius`

RADIUS 서버를 사용하여 인증한다. 자세한 내용은 3.3.8절을 참조 바란다.

`cert`

SSL 클라이언트 인증을 사용하여 인증한다. 자세한 내용은 3.3.9절을 참조 바란다.

`pam`

운영 체제에서 제공하는 PAM(Pluggable Authentication Modules)을 사용하여 인증한다. 자세한 내용은 3.3.10절을 참조 바란다.

`auth-options`

`auth-method` 필드 이후에 인증 방법에 대한 옵션을 지정하는 *`name=value`* 형식의 필드가 있을 수 있다. 인증 방법에서 사용할 수 있는 옵션에 대한 자세한 내용은 아래에 나와 있다.

`@` 구문이 포함된 파일은, 공백 또는 쉼표로 구분된 이름 목록으로 읽는다. `pg_hba.conf`처럼 #로 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.

pg_hba.conf 파일은 시작 시 및 메인 서버 프로세스가 SIGHUP 신호를 수신하면 읽혀지게 된다. 활성화 된 시스템에서 파일을 편집하는 경우 파일을 다시 읽어오려면 postmaster에 신호를 전송해야 한다(pg_ctl reload 또는 kill -HUP 사용).

> 작은 정보: 특수한 데이터베이스에 연결하려면 pg_hba.conf 검사만 통과해서는 안 되며 데이터베이스에 대한 CONNECT 권한이 사용자에게 있어야 한다. 데이터베이스에 연결 가능한 사용자를 제한하고 싶으면 pg_hba.conf 항목에 규칙을 입력하는 것보다 CONNECT 권한을 부여/취소하는 것이 일반적으로 쉽다.

pg_hba.conf 항목에 대한 몇 가지 예시가 예 3-1에 나와 있다. 서로 다른 인증 방법에 대한 자세한 내용은 다음 절을 참조 바란다.

예 3-1. pg_hba.conf 항목 예시

```
# Allow any user on the local system to connect to any database with
# any database user name using Unix-domain sockets (the default for local
# connections).
#
# TYPEDATABASEUSERADDRESS METHOD
local all all trust

# The same using local loopback TCP/IP connections.
#
# TYPEDATABASEUSERADDRESS METHOD
hostall all 127.0.0.1/32trust

# The same as the previous line, but using a separate netmask column
#
# TYPEDATABASEUSERIP-ADDRESSIP-MASK METHOD
hostall all 127.0.0.1 255.255.255.255 trust

# The same over IPv6.
#
# TYPEDATABASEUSERADDRESS METHOD
hostall all ::1/128 trust

# The same using a host name (would typically cover both IPv4 and IPv6).
#
# TYPEDATABASEUSERADDRESS METHOD
hostall all localhost trust

# Allow any user from any host with IP address 192.168.93.x to connect
# to database "postgres" as the same user name that ident reports for
# the connection (typically the operating system user name).
#
# TYPEDATABASEUSERADDRESS METHOD
hostpostgresall 192.168.93.0/24 ident

# Allow any user from host 192.168.12.10 to connect to database
# "postgres" if the user's password is correctly supplied.
#
# TYPEDATABASEUSERADDRESS METHOD
hostpostgresall 192.168.12.10/32md5

# Allow any user from hosts in the example.com domain to connect to
```

```
# any database if the user's password is correctly supplied.
#
# TYPEDATABASEUSERADDRESS METHOD
hostall all .example.commd5

# In the absence of preceding "host" lines, these two lines will
# reject all connections from 192.168.54.1 (since that entry will be
# matched first), but allow GSSAPI connections from anywhere else
# on the Internet.The zero mask causes no bits of the host IP
# address to be considered, so it matches any host.
#
# TYPEDATABASEUSERADDRESS METHOD
hostall all 192.168.54.1/32 reject
hostall all 0.0.0.0/0 gss

# Allow users from 192.168.x.x hosts to connect to any database, if
# they pass the ident check.If, for example, ident says the user is
# "bryanh" and he requests to connect as Agens SQL user "guest1", the
# connection is allowed if there is an entry in pg_ident.conf for map
# "omicron" that says "bryanh" is allowed to connect as "guest1".
#
# TYPEDATABASEUSERADDRESS METHOD
hostall all 192.168.0.0/16ident map=omicron

# If these are the only three lines for local connections, they will
# allow local users to connect only to their own databases (databases
# with the same name as their database user name) except for administrators
# and members of role "support", who can connect to all databases.The file
# $PGDATA/admins contains a list of names of administrators.Passwords
# are required in all cases.
#
# TYPEDATABASEUSERADDRESS METHOD
local sameuserall md5
local all @admins md5
local all +supportmd5

# The last two lines above can be combined into a single line:
local all @admins,+supportmd5

# The database column can also use lists and file names:
local db1,db2,@demodbsall md5
```

3.2. 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-name`*을 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.conf`의 *`map-name`* 매개 변수에서 지정된다.

사용자 이름 맵은 ident 맵 파일에서 정의되며, 기본적으로 이름은 `pg_ident.conf`이며 클러스터의 데이터 디렉토리에 저장된다. (맵 파일을 다른 곳에 배치할 수도 있다. ident_file 환경 설정 매개 변수를 참조 바란다.)

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

주석 및 공백은 `pg_hba.conf`에서와 동일하게 처리된다. *map-name*은 `pg_hba.conf`에서 이 맵핑을 참고하기 위해 사용되는 임의의 이름이다. 나머지 2개의 필드는 운영 체제 사용자 이름 및 일치하는 데이터베이스 사용자 이름을 지정한다. 동일한 *map-name*을 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 1개)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.com`로 끝나는 시스템 사용자 이름을 사용하여 사용자에 대한 도메인 부분을 삭제하고, 시스템 이름이 `@otherdomain.com`로 끝나는 모든 사용자가 `guest`로 로그인하는 것을 허용한다.

> 작은 정보**:** 기본적으로 정규식은 string의 일부만 일치할 수 있다는 점에 유의해야 한다. 위의 예시처럼 전체 시스템 사용자 이름과 일치하도록 하려면 `^` 및 `$`를 사용하는 것이 좋다.

`pg_ident.conf` 파일은 시작 시 및 메인 서버 프로세스가 SIGHUP 신호를 수신했을 때 읽혀진다. 활성화 된 시스템에서 파일을 편집하는 경우 파일을 다시 읽어오려면 postmaster에 신호를 전송해야 한다(`pg_ctl reload` 또는 `kill -HUP` 사용).

예 3-1에서 `pg_ident.conf` 파일과 함께 사용할 수 있는 `pg_hba.conf` 파일이 예 3-2에 나와 있다. 이 예제에서 운영 체제 사용자 이름인 `bryanh` 또는 `ann`, `robert`가 없는 192.168 네트워크에서 로그인한 사용자는 액세스 권한을 부여 받지 못한다. Unix 사용자 `robert`는 `robert` 또는 다른 사람이 아니라 Agens SQL 사용자인 `bob`으로 연결을 시도하는 경우에만 액세스가 허용된다. `ann`은 `ann`으로 연결할 때만 허용된다. 사용자 `bryanh`은 `bryanh` 또는 `guest1`일 때만 연결이 허용된다.

예 **3-2. `pg_ident.conf`** 파일 예시

```
# MAPNAME SYSTEM-USERNAME PG-USERNAME

omicron bryanhbryanh
omicron ann ann
# bob has user name robert on these machines
omicron robertbob
# bryanh can also connect as guest1
omicron bryanhguest1
```

3.3. 인증 방법

아래 절에서는 인증 방법을 자세하게 다룬다.

3.3.1. 트러스트 인증

trust 인증이 지정된 경우 Agens SQL는 지정한 데이터베이스 사용자 이름을 사용하여 서버에 연결 가능한 모든 이가 데이터베이스 액세스에 대한 인증을 받는 것으로 간주한다(수퍼유저 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. 그러나, 파일 시스템 권한을 사용하여 서버의 Unix 도메인 소켓 파일에 대한 액세스를 제한하는 경우 다중 사용자 머신에서도 trust를 사용 가능할 수 있다. 이렇게 하려면 unix_socket_permissions(및 가능하면 unix_socket_group) 환경 설정 매개 변수를 2.3절에서 설명한 대로 설정해야 한다. 또는 unix_socket_directories 환경 설정 매개 변수를 설정하여 소켓 파일을 적절히 제한된 디렉토리에 배치할 수 있다.

파일 시스템 권한 설정은 Unix 소켓 연결 시에만 유용하다. 로컬 TCP/IP 연결은 파일 시스템 권한에 의해 제한되지 않는다. 따라서 로컬 보안을 위해 파일 시스템 권한을 사용하려면 host ... 127.0.0.1 ... 줄을 pg_hba.conf에서 삭제하거나, 비 trust 방법으로 변경해야 한다.

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.

3.3.2. 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" 공격을 주의하는 경우 md5가 바람직하다. 일반 password는 가능하면 피해야 한다. 대신, md5는 db_user_namespacedb_user_namespace 기능과 함께 사용할 수 없다. 연결이 SSL 암호화로 보호되는 경우 password를 안전하게 사용할 수 있다(SSL을 사용하는 경우 SSL 인증서 인증이 더 나을 수 있지만).

Agens SQL 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.

3.3.3. GSSAPI 인증

GSSAPI는 RFC 2743에 정의된 보안 인증을 위한 산업 표준 프로토콜이다. Agens SQL은 RFC 1964에 따라 Kerberos를 사용한 GSSAPI를 지원한다. GSSAPI은 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.

GSSAPI는 Agens SQL이 빌드된 경우 활성화되어야 한다.

GSSAPI가 Kerberos를 사용하는 경우 *servicename*/*hostname*@*realm* 형식으로 표준 규칙이 사용된다. Agens SQL 서버가 서버에서 사용되는 키탭에 포함된 보안 규칙을 수용하지만, krbsrvname 연결 매개 변수를 사용하여 클라이언트에서 연결을 할 때 올바른 보안 규칙 상세 정보를 지정할 때 특별히 주의해야 한다. 설치 기본값은 빌드 시에 ./configure --with-krb-srvnam=*whatever*를 사용하여 기본값 postgres에서 변경 가능하다. 대부분의 환경에서 이 매개 변수는 절대 변경할 필요가 없다. 일부 Kerberos 구현은 서비스 이름이 대문자여야 하는 Microsoft Active Directory처럼 서로 다른 서비스 이름을 요구할 수도 있다(POSTGRES).

*hostname*은 서버 머신의 정규화된 호스트 이름이다. 서비스 보안 규칙의 영역은 서버 머신의 기본 설정된 영역이다.

클라이언트 보안 규칙은 첫 번째 구성요소로 자체 Agens SQL 데이터베이스 사용자 이름을 갖는다(예:pgusername@realm). 또는, 보안 규칙 이름의 첫 번째 구성요소부터 데이터베이스 사용자 이름까지 맵핑을 위해 사용자 이름 맵핑을 사용할 수도 있다. 기본적으로 클라이언트의 영역(realm)은 Agens SQL에서 검사되지 않는다. 영역 간 인증(cross-realm authentication)을 활성화하고 영역(realm)을 검증해야 하는 경우 krb_realm 매개 변수를 사용하거나 include_realm을 활성화하고 사용자 이름 맵핑을 사용하여 영역(realm)을 검사해야 한다.

사용자는 서버 키탭 파일을 Agens SQL 서버 계정으로 판독 가능한지(기본적으로 판독만 가능한지) 확인해야 한다. S키 파일의 위치는 krb_server_keyfile 환경 설정 매개 변수에 의해 지정된다. 기본값은 /usr/local/pgsql/etc/krb5.keytab(또는 빌드 시 sysconfdir로 지정된 아무 디렉토리). 보안상의 이유로, 시스템 키탭 파일에 대한 권한을 여는 것보다는 Agens SQL 서버에 대해 별개의 키탭을 사용하는 것이 바람직하다.

키탭 파일은 Kerberos 소프트웨어에 의해 생성된다. 자세한 내용은 Kerberos 문서를 참조 바란다. 다음 예시는 MIT 호환 Kerberos 5 구현에 대한 것이다.

```
kadmin% ank -randkey postgres/server.my.domain.org
kadmin% ktadd -k krb5.keytab postgres/server.my.domain.org
```

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. 보안 규칙 fred/users.example.com@EXAMPLE.COM의 연결도 허용하려면 3.2절에 설명된 대로 사용자 이름 맵을 사용해야 한다.

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(3.2절). 이것은 복수의 영역(realm)에서 사용자를 처리할 때 유용하다. 기본값은 0이지만(시스템 사용자 이름의 영역에 포함되지 않음을 뜻함), 향후 Agens SQL 버전에서는 1로 바뀔 수도 있다. 사용자는 업그레이드할 때 이슈를 피하기 위해 0을 명시적으로 설정할 수 있다.

map

시스템과 데이터베이스 사용자 이름 사이의 맵핑을 허용한다. 자세한 내용은 3.2절을 참조 바란다. Kerberos 보안 규칙 username/hostbased@EXAMPLE.COM의 경우 맵핑에 사용된 사용자 이름은 include_realm이 비활성화됐을 시에 username/hostbased이고, include_realm이 활성화됐을 시에 username/hostbased@EXAMPLE.COM이다.

`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.

3.3.4. SSPI 인증

SSPI는 단일 사인온(sign-on)의 보안 인증을 위한 Windows 기술이다. Agens SQL은 `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. 자세한 내용은 3.3.3절을 참조 바란다.

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(3.2절). 이것은 복수의 영역(realm)에서 사용자를 처리할 때 유용하다. 기본값은 0이지만(시스템 사용자 이름의 영역에 포함되지 않음을 뜻함), 향후 Agens SQL 버전에서는 1로 바뀔 수도 있다. 사용자는 업그레이드할 때 이슈를 피하기 위해 0을 명시적으로 설정할 수 있다.

`map`

시스템과 데이터베이스 사용자 이름 사이의 맵핑을 허용한다. 자세한 내용은 3.2절을 참조 바란다.

`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.

3.3.5. 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.

> 참고**:** 로컬(비 TCP/IP) 연결에 대해 ident를 지정하는 경우 피어(peer) 인증이 대신 사용된다(3.3.6절 참조).

다음 구성 옵션이 ident에 대해 지원된다.

`map`

시스템과 데이터베이스 사용자 이름 사이의 맵핑을 허용한다. 자세한 내용은 3.2절을 참조 바란다.

"신분확인 프로토콜(Identification Protocol)"은 RFC 1413에 설명되어 있다. 실제로 모든 Unix 류의 운영 체제에는 기본적으로 TCP 포트 113에서 listen하는 ident 서버가 내장되어 있다. ident 서버의 기본적인 기능은, "당신의 포트 *X*에서 출력되어 내 포트 *Y*에 연결되는 연결을 초기화한 사용자는 누구인가?" 같은 질문에 응답하는 것이다. 실제 연결이 성립되면 Agens SQL은 *X*와 *Y*를 모두 알고 있으므로 연결 클라이언트의 호스트에 대한 정보를 ident 서버에서 얻을 수 있으며, 주어진 연결에서 운영 체제 사용자를 판단할 수 있다.

이 방법의 단점은 클라이언트의 무결성에 따라 달라진다. 클라이언트 머신을 신뢰할 수 없거나 손상된 경우 공격자(attacker)는 포트 113에서 프로그램을 실행하고, 선택한 사용자 이름으로 리턴할 수 있다. 따라서 인증 방법은 각 클라이언트 머신이 엄격하게 제어되고, 데이터베이스 및 시스템 관리자의 협력이 긴밀하게 이뤄지는 폐쇄된 네트워크의 경우에만 적합하다. 즉, 사용자는 ident 서버가 실행되는 머신을 신뢰해야 한다. 다음 경고에 유의해야 한다.

> 신분확인 프로토콜은 인증 또는 액세스 제어 프로토콜로 사용할 수 없다(The Identification Protocol is not intended as an authorization or access control protocol).
>
> —RFC 1413

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

3.3.6. 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. 자세한 내용은 3.2절을 참조 바란다.

피어(Peer) 인증은 `getpeereid()` 함수, SO_PEERCRED 소켓 매개 변수 또는 유사 메커니즘이 제공되는 운영 체제에서만 사용할 수 있다. 현재 Linux가 포함되며, OS X 및 Solaris를 비롯한 BSD가 가장 선호된다.

3.3.7. 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 Active Directory 환경에서 cn= 또는 *DOMAIN*\을 지정하는 데 사용된다. *suffix*는 비 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 Agens SQL과 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`*`]]]`
*`scope`*는 `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 `ldapbindpasswd`는 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=n
```

데이터베이스 사용자 `someuser`로 데이터베이스 서버에 연결이 요청된 경우 Agens SQL은 DN `cn=someuser, dc=example, dc=net` 및 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" ldapsearchattri
```

데이터베이스 사용자 `someuser`로 데이터베이스 서버에 연결이 요청된 경우 Agens SQL은 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.

작은 정보: LDAP는 주로 쉼표와 공백을 사용하여 DN의 서로 다른 부분을 구분하므로 예시에 표시된 대로 LDAP 옵션을 구성할 때 매개 변수를 큰따옴표로 둘러싸야 하는 경우가 있다.

3.3.8. 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 Identifier에 대한 매개 변수가 포함된다. 요청은 서버와 공유되는 시크릿을 사용하여 암호화된다. RADIUS 서버는 Access Accept 또는 Access Reject를 사용하여 이 서버에 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 Agens SQL 및 RADIUS 서버에서 값이 정확하게 동일해야 한다. 최소 16자의 string이 권장된다. 이 매개 변수는 필수이다.

참고: 사용되는 암호화 벡터는 Agens SQL가 OpenSSL을 지원하도록 빌드된 경우 강력한 방식으로 암호화되어야 한다. 그 외의 경우에, RADIUS 서버로의 전송은 보안이 되지 않은 애매한 것으로 간주해야 하며, 필요 시 외부 보안 대책을 적용해야 한다.

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.

3.3.9. 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 cn을 데이터베이스 사용자 이름과 다르게 할 수 있다.

다음 구성 옵션이 SSL 인증서 인증에 대해 지원된다.

map

시스템과 데이터베이스 사용자 이름 사이의 맵핑을 허용한다. 자세한 내용은 3.2절을 참조 바란다.

3.3.10. PAM 인증

이 인증 방법은 인증 메커니즘으로 PAM(Pluggable Authentication Modules)을 사용할 때 외에는 password와 유사하게 작동된다. 기본 PAM 서비스 이름은 postgresql이다. PAM은 사용자 이름/패스워드 쌍을 검증할 때에만 사용된다. 따라서 PAM을 인증에 사용하기 전에 사용자

가 데이터베이스에 존재해야 한다. PAM에 대한 자세한 내용은 Linux-PAM 페이지 (http://www.kernel.org/pub/linux/libs/pam/)를 참고 바란다.

다음 구성 옵션이 PAM에 대해 지원된다.

`pamservice`

PAM 서비스 이름.

참고: PAM이 `/etc/shadow`를 읽도록 설정된 경우 Agens SQL 서버는 root가 아닌 다른 사용자로 시작되므로 인증이 실패한다. 단, PAM이 LDAP 또는 다른 인증 방법을 사용하도록 설정된 경우 실행되지 않는다.

3.4. 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.

작은 정보: 서버 로그에 클라이언트에 보고된 것보다 더 자세한 인증 실패 정보가 나와 있다. 실패 이유가 명확하지 않은 경우 서버 로그를 확인해야 한다.

4장. 데이터베이스 role

Agens SQL은 *role*이라는 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 *멤버십*을 다른 role에 부여할 수 있으므로 멤버 role이 다른 role에 할당된 권한을 사용하도록 할 수 있다.

role의 개념은 "사용자" 및 "그룹"의 개념을 포함한다. Agens SQL 8.1 이전 버전에서 사용자와 그룹은 별개의 엔티티였지만 이제는 role만 있다. 모든 role은 사용자, 그룹 또는 양쪽 모두로 작용할 수 있다.

이 장에서는 role을 생성 및 관리하는 방법을 설명한다.

4.1. 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.

```
CREATE ROLE name;
```

*name*은 SQL 식별자에 대한 규칙(특수 문자 또는 큰따옴표 사용 안함)을 준수한다. (실제로는 LOGIN 같은 옵션을 명령에 추가하고 싶을 수도 있다. 자세한 내용이 아래에 나와 있다.) 기존 role을 삭제하려면 유사한 DROP ROLE 명령을 사용해야 한다.

```
DROP ROLE name;
```

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

```
createuser name
dropuser name
```

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

```
SELECT rolname FROM pg_roles;
```

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

데이터베이스 시스템을 부트스트랩하려면 갓 초기화된 시스템이 사전 정의된 role을 항상 포함해야 한다. 이 role은 항상 "수퍼유저"이며, 기본적으로(**initdb** 실행 중에 변경하지 않는 한) 데이터베이스 클러스터를 초기화했던 운영 체제와 이름이 동일하다. 통상적으로 이 role의 이름은 postgres이다. role을 좀 더 생성하려면 먼저 이 postgres로 연결해야 한다.

데이터베이스 서버에 대한 모든 연결은 일부 특정 role의 이름이 사용되며, 이러한 role은 해당 연결에서 실행된 명령에 대한 초기 액세스 권한을 결정한다. 특정 데이터베이스 연결에 사용할 role 이름은 애플리케이션 특정 방식의 연결 요청을 초기화하는 클라이언트에 의해 표시된다. 예를 들면, **psql** 프로그램은 -U 옵션을 사용하여 연결할 role을 표시한다. 다수의 애플리

케이션이 기본적으로 현재 운영 체제 시스템 사용자의 이름을 가정한다(**createuser** 및 **psql** 포함). 따라서 role 및 운영 체제 사용자 간에 네이밍 연관성을 유지하는 것이 편리하다.

주어진 클라이언트 연결을 할 수 있는 데이터베이스 role 집합은 3장에서 설명된 대로 클라이언트 인증 설정에 의해 결정된다. (따라서, 로그인 이름이 꼭 그 사람의 실명과 일치해야 할 필요가 없는 것처럼, 클라이언트를 운영 체제 사용자와 일치하는 role로 연결해야만 하는 것은 아니다.) role ID는 연결된 클라이언트에서 가능한 권한 집합을 결정하므로 다중 사용자 환경을 설정할 때 권한을 세심하게 구성하는 것이 중요하다.

4.2. 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.

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

(**CREATE USER**는 디폴트로 `LOGIN` 권한을 준다는 점에서 **CREATE ROLE**과 다르다.)

수퍼유저 상태

데이터베이스 수퍼유저는 로그인 권한을 제외한 모든 권한 검사를 건너 뛴다. 이 권한은 위험하며, 무심코 사용해서는 안 된다. 작업 대부분은 수퍼유저 이외의 다른 role로 수행하는 것이 좋다. 새 데이터베이스 수퍼유저를 생성하려면 `CREATE ROLE name SUPERUSER`를 사용해야 한다. 수퍼유저인 role로 이것을 수행해야 한다.

데이터베이스 생성

데이터베이스를 생성하려면 권한이 명시적으로 role에 주어져야 한다(모든 권한 검사를 건너 뛰는 수퍼유저인 경우 제외). 이러한 role을 생성하려면 `CREATE ROLE name CREATEDB`를 사용해야 한다.

role 생성

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

복제 초기화

streaming replication을 초기화하려면 권한이 명시적으로 role에 주어져야 한다(모든 권한 검사를 건너 뛰는 수퍼유저인 경우 제외). streaming replication에 사용되는 role은 항상 `LOGIN` 권한이 있어야 한다. 이러한 role을 생성하려면 `CREATE ROLE name REPLICATION LOGIN`을 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 운영 체제 사용자 패스워드와 구분된다. role 생성 시 패스워드 지정은 CREATE ROLE *name* PASSWORD '*string*'을 사용해야 한다.

role의 속성은 **ALTER ROLE**로 생성한 후 수정할 수 있다. 자세한 내용은 CREATE ROLE 및 ALTER ROLE 명령에 대한 참고 페이지를 참조 바란다.

작은 정보**:** 수퍼유저는 아니지만 CREATEDB 및 CREATEROLE 권한이 있는 role을 생성하고, 데이터베이스와 role의 모든 루틴 관리에 대해 이 role을 사용하는 것이 좋다. 이러한 방법으로 실제로 수퍼유저 권한이 불필요한 작업을 수퍼유저로 실행하는 위험이 방지된다.

role은 2장에 설명된 여러 가지 런타임 구성 설정에 대해 role별 기본값을 가질 수도 있다. 예를 들면, 연결할 때 인덱스 스캔을 비활성화하려면, 다음과 같이 할 수 있다.

```
ALTER ROLE myname SET enable_indexscan TO off;
```

이것은 설정을 저장한다(그러나 즉시 설정되지는 않음). SET enable_indexscan TO off가 세션 시작 직전에 실행되었더라도 추후 이 role이 연결할 때 이것이 나타난다. 세션 중에 이 설정을 변경할 수 있으며, 이것은 기본값에 불과하다. role별 기본 설정을 삭제하려면 ALTER ROLE *rolename* RESET *varname*을 사용해야 한다. LOGIN 권한 없이 role에 연결된 role별 기본값은 절대 호출되지 않으므로 아무 쓸모가 없다는 점에 유의해야 한다.

4.3. role 멤버십

권한 관리의 편의상 사용자를 그룹으로 묶는 것이 편리할 수 있다. 이렇게 하면 권한을 그룹 단위로 부여하거나 취소할 수 있다. Agens SQL에서 이것은 그룹을 나타내는 role을 생성한 다음, 그룹 role의 *멤버십*을 개별 사용자 role에 부여하면 된다.

그룹 role을 설정하려면 먼저 role을 생성해야 한다.

```
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.

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

```
CREATE ROLE joe LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;
CREATE ROLE wheel NOINHERIT;
GRANT admin TO joe;
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. 다음 명령을 실행한 후,

```
SET ROLE admin;
```

세션은 admin에 부여된 이러한 권한만 사용하고 joe에 부여된 권한은 사용하지 않는다. 다음 명령을 실행한 후,

```
SET ROLE wheel;
```

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

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

> 참고: **SET ROLE** 명령은 원래 로그인 role이 직간접적으로 멤버인 모든 role을 선택할 수 있도록 항상 허용한다. 따라서 위의 예시에서, wheel이 되기 전에 꼭 admin이어야 할 필요는 없다.

> 참고: SQL 표준에서 사용자와 role은 명확히 구분되며, role은 권한을 자동으로 상속 받지만, 사용자는 그렇지 않다. 이는 SQL 사용자로서 사용되는 role에게는 NOINHERIT 속성을 부여하고, SQL role로 사용되는 role에게는 INHERIT 속성을 부여한다. 그러나, 사용자에게 항상 멤버인 그룹에 부여된 권한이 있는 8.1 릴리스 이전 버전과의 호환성 때문에 Agens SQL은 기본적으로 모든 role에 INHERIT 속성을 부여한다.

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 ROLE**을 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 admin**을 수행한 이후에만 권한이 부여된다.

그룹 role을 소멸하려면 DROP ROLE을 사용해야 한다.

```
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.

4.4. 함수 및 트리거 보안

함수와 트리거를 사용하면 다른 사용자가 무심코 실행할 수 있는 코드를 백엔드 서버에 삽입할 수 있다. 따라서 두 메커니즘은 “트로이목마”를 상대적으로 쉽게 침투시킬 수 있다. 유일한 보호책은 함수를 정의할 수 있는 사용자를 엄격하게 제어하는 것이다.

함수는 데이터베이스 서버 데몬의 운영 체제 권한이 있는 백엔드 서버 프로세스 내에서 실행된다. 함수에 사용된 프로그래밍 언어가 미검사 메모리 액세스를 허용하는 경우 서버의 내부 데이터 구조를 변경하는 것이 가능하다. 따라서 해당 함수는 모든 시스템 액세스 제어를 피해갈 수 있다. 해당 액세스를 허용하는 함수 언어는 “신뢰되지 않음”으로 간주되고, Agens SQL는 수퍼유저만 해당 언어로 작성된 함수를 생성하도록 허용한다.

5장. 데이터베이스 관리

실행 중인 Agens SQL 서버의 모든 인스턴스는 하나 이상의 데이터베이스를 관리한다. 그러므로 데이터베이스는 SQL 객체("데이터베이스 객체")를 구성하기 위한 최상위 계층이다. 이 장에서는 데이터베이스의 특성과 데이터베이스를 생성 및 관리, 소멸하는 방법에 대해 다룬다.

5.1. 개요

데이터베이스는 명명된 SQL 객체 컬렉션이다("데이터베이스 객체"). 일반적으로 모든 데이터베이스 객체(테이블, 함수 등)은 데이터베이스 한 곳에만 속한다. (예를 들면, `pg_database`처럼 전체 클러스터에 속하고 클러스터 내의 각 데이터베이스에서 접근 가능한 시스템 카탈로그는 몇 개 있다.) 좀 더 정확하게 말하면 데이터베이스는 스키마의 컬렉션이며, 스키마에는 테이블, 함수 등이 포함된다. 따라서 전체 구조는 서버, 데이터베이스, 스키마, 테이블(또는 함수 같이 종류가 다른 일부 객체)로 구성된다.

데이터베이스 서버에 연결하는 경우 클라이언트는 연결하려는 데이터베이스 이름을 연결을 요청할 때 지정해야 한다. 연결당 데이터베이스를 하나 이상 접근할 수도 있다. 단, 애플리케이션은 동일한 데이터베이스 또는 다른 데이터베이스에 대해 열리는 연결 수에 제한되지 않는다. 데이터베이스는 물리적으로 구분되며 접근 제어는 연결 수준에서 관리된다. 한 Agens SQL 서버 인스턴스가 서로 인식하지 못하는 프로젝트 또는 사용자를 수용하는 경우, 두 가지를 별도의 데이터베이스에 두는 것이 좋다. 프로젝트 또는 사용자가 서로 연관되어 있고 서로의 자원을 사용해야 하는 경우에는 두 가지를 동일한 데이터베이스에 두고 스키마는 분리해야 한다. 스키마는 순전히 논리적 구조이며 권한 시스템이 관리하는 대상에 접근할 수 있는 사용자이다.

데이터베이스는 **CREATE DATABASE** 명령(5.2절 참조)으로 생성되고 DROP DATABASE 명령(5.5절 참조)으로 소멸된다. 기존 데이터베이스를 판단하려면 pg_database 시스템 카탈로그를 검사해야 한다. 예를 들면,

```
SELECT datname FROM pg_database;
```

psql 프로그램의 `\l` 메타 명령 및 `-l` 커맨드라인 옵션도 기존 데이터베이스를 나열하는 데 유용하다.

> 참고: SQL 표준은 데이터베이스 "카탈로그"를 호출하지만 실제로는 차이가 없다.

5.2. 데이터베이스 생성

데이터베이스를 생성하려면 Agens SQL 서버를 시작한 다음에 실행해야 한다(1.3절 참조).

데이터베이스는 SQL 명령 CREATE DATABASE으로 생성된다.

```
CREATE DATABASE name;
```

여기서, *name*은 SQL 식별자에 대한 일반 규칙을 따른다. 현재 role은 자동으로 새 데이터베이스의 소유자가 된다. 나중에 데이터베이스를 삭제하는 것은 데이터베이스 소유자의 권한이다(소유자가 달라도 데이터베이스의 모든 객체가 삭제됨).

데이터베이스 생성은 제한적이다. 권한 부여 방법은 4.2절을 참조 바란다.

CREATE DATABASE**CREATE DATABASE** 명령을 실행하려면 데이터베이스 서버에 연결해야 하는데, 주어진 사이트에서 첫 번째 데이터베이스를 어떻게 만들 것인가가 문제가 된다. 첫 번째 데이터베이스는 데이터 저장소 영역을 초기화할 때 **initdb** 명령으로 항상 생성된다. (1.2절 참조.) 이 데이터베이스를 `postgres`라고 한다. 따라서 첫 번째 "일반" 데이터베이스를 생성하기 위해 `postgres`에 연결할 수 있다.

두 번째 데이터베이스 `template1`도 데이터베이스 클러스터 초기화 중에 생성된다. 클러스터 내에서 새 데이터베이스를 생성할 때마다 `template1`이 복제된다. 이것은 `template1`에서 변경된 내용이 추후 생성된 데이터베이스로 전파된다는 것을 의미한다. 따라서 새로 생성된 모든 데이터베이스로 전파되는 것을 원하지 않으면 `template1`에서 객체 생성을 하지 말아야 한다. 자세한 내용은 5.3절에 나와 있다.

createdb는 편의상 새 데이터베이스를 생성하기 위해 쉘에서 실행할 수 있는 프로그램이다.

```
createdb dbname
```

createdb는 마법이 아니다. 이것은 `postgres` 데이터베이스에 연결하고 위에서 설명한 대로 정확하게 **CREATE DATABASE** 명령을 실행한다. 인자가 없는 **createdb**는 현재 사용자 이름으로 데이터베이스를 생성한다.

> 참고: 주어진 데이터베이스에 연결할 수 있는 사용자를 제한하는 방법에 대한 내용은 3장에 나온다.

다른 사용자를 위한 데이터베이스를 생성하고, 그 사용자를 새 데이터베이스의 소유자로 만들면, 해당 사용자가 스스로 새 데이터베이스를 환경 설정 및 관리할 수 있다. 이렇게 하려면 SQL 환경에서 다음 명령 중 하나를 사용해야 한다.

```
CREATE DATABASE dbname OWNER rolename;
```

또는 쉘에서 다음을 사용해야 한다.

```
createdb -O rolename dbname
```

다른 사용자를 위한 데이터베이스는 수퍼유저만 생성할 수 있다(즉, 멤버가 아닌 role의 경우).

5.3. 템플릿 데이터베이스

CREATE DATABASE는 실제로 기존 데이터베이스를 복사한다. 기본적으로 `template1`이라는 표준 시스템 데이터베이스를 복사한다. 따라서 해당 데이터베이스는 새 데이터베이스를 만드는 "템플릿"이다. 객체를 `template1`에 추가하면, 이 객체는 나중에 생성된 사용자 데이터베이스로 복사된다. 이 작업은 데이터베이스의 표준 객체 집합에 대한 사이트-로컬 수정을 가능하게 한다. 예를 들면, `template1`에서 프로시저 언어 PL/Perl을 설치하는 경우 해당 데이터베이스를 생성할 때 추가적인 작업 없이 사용자 데이터베이스가 자동으로 사용 가능하게 된다.

`template0`이라는 2차 표준 시스템 데이터베이스가 있다. 이 데이터베이스에는 `template1`의 초기 내용과 동일한 데이터가 포함되어 있다. 즉, Agens SQL 버전에 의해 사전 정의된 표준 객체만 포함되어 있다. 데이터베이스 클러스터를 초기화한 후에는 `template0`을 절대 변경하면 안 된다. `template1` 대신 `template0`을 복사하도록 **CREATE DATABASE**를 실행하면 `template1`에서 사이트-로컬 추가가 없는 "처녀" 사용자 데이터베이스를 생성할 수 있다. 이것은 `pg_dump` 덤프를 복원할 때 특히 유용하다. 나중에 `template1`에 추가되었을 수

도 있는 객체와 충돌 없이 덤프된 데이터베이스를 올바르게 재생성 하도록 처녀 데이터베이스에서 덤프 스크립트를 복원해야 한다.

template1 대신 template0을 복사하는 일반적인 다른 이유는 template1의 복사는 동일한 설정을 사용해야 하지만 template0을 복사하는 경우에는 새 인코딩 및 로케일(locale) 설정을 지정할 수 있기 때문이다. template1은 인코딩 또는 로케일(locale)에 관한 데이터를 포함하지만 template0은 그렇지 않다.

template0을 복사하여 데이터베이스를 생성하려면

```
CREATE DATABASE dbname TEMPLATE template0;
```

SQL 환경에서 다음을 사용해야 한다.

```
createdb -T template0 dbname
```

또는 쉘에서 다음을 사용해야 한다.

템플릿 데이터베이스를 추가적으로 생성하는 것이 가능하며, **CREATE DATABASE**의 경우 템플릿 이름을 지정하면 클러스터의 데이터베이스를 복사할 수도 있다. 그러나, 아직까지는 "**COPY DATABASE**"가 범용으로 사용되는 기능이 아니라는 점을 이해해야 한다. 이러한 제한은 복사 중인 소스 데이터베이스에 다른 세션을 연결할 수 없다는 것을 의미한다. 시작 시 다른 연결이 존재하면 **CREATE DATABASE**가 실패하며, 복사 명령 중에 소스 데이터베이스에 대한 새롭게 연결되는 것을 방지한다.

데이터베이스별로 pg_database에는 datistemplate 칼럼 및 datallowconn 칼럼의 두 가지 유용한 플래그가 존재한다. datistemplate를 설정하여 데이터베이스가 **CREATE DATABASE**의 템플릿임을 나타낼 수 있다. 이 플래그가 설정되면 CREATEDB 권한이 있는 사용자가 데이터베이스를 복제할 수 있다. 설정되지 않으면 수퍼유저와 데이터베이스 소유자만 복제할 수 있다. datallowconn이 false인 경우 해당 데이터베이스에 대한 새로운 연결이 허용되지 않는다(단, 플래그를 false로 설정해도 기존 세션은 중단되지 않는다). template0 데이터베이스는 수정 방지를 위해 일반적으로 datallowconn = false로 표시된다. 양쪽 template0 및 template1은 datistemplate = true로 항상 표시되어야 한다.

> 참고: template1 및 template0은 이름 template1이 **CREATE DATABASE**의 기본 소스 데이터베이스 이름이라는 사실 외에는 특별한 상태를 나타내지 않는다. 예를 들면, template1를 삭제하고 부작용 없이 template0으로 재생성 가능하다. 이러한 작업 과정은 template1에 잘못 추가한 것이 많을 경우에 좋다. (template1을 삭제하려면 pg_database.datistemplate = false여야 한다.)
>
> 데이터베이스 클러스터가 초기화될 때 postgres 데이터베이스도 생성된다. 이 데이터베이스는 연결하는 사용자 및 애플리케이션의 기본 데이터베이스임을 의미한다. 이것은 단순히 template1의 사본이며, 필요시 삭제 및 재생성이 가능하다.

5.4. 데이터베이스 환경 설정

2장에서 설명한 대로 Agens SQL 서버는 여러 가지 런타임 설정 변수를 제공한다. 이러한 여러 가지 설정에 대해 데이터베이스별 기본값을 설정할 수 있다.

예를 들면, 주어진 데이터베이스에 대해 GEQO 옵티마이저를 비활성화하려는 경우 대개는 모든 데이터베이스에 대해 비활성화해야 하거나 모든 연결 클라이언트가 신중하게 SET geqo TO off를 실행하는지 확인해야 한다. 특정 데이터베이스 내에서 이 설정을 기본값으로 설정하려면 다음 명령을 실행해야 한다.

```
ALTER DATABASE mydb SET geqo TO off;
```

이것은 설정을 저장한다(그러나 즉시 설정되지는 않음). `SET geqo TO off;`;가 세션 시작 직전에 실행되었더라도 이 데이터베이스를 추후 연결했을 때 이것이 적용된다. 세션 중에 사용자가 이 설정을 변경할 수 있으며, 이는 기본값에 불과하다. 설정을 실행 취소하려면 `ALTER DATABASE dbname RESET varname`을 사용해야 한다.

5.5. 데이터베이스 소멸

데이터베이스는 DROP DATABASE 명령으로 소멸된다.

```
DROP DATABASE name;
```

데이터베이스 소유자 또는 수퍼유저만 데이터베이스를 삭제할 수 있다. 데이터베이스를 삭제하면 데이터베이스에 포함된 모든 객체가 삭제된다. 데이터베이스 소멸은 실행 취소가 불가능하다.

삭제 대상 데이터베이스에 연결된 상태에서는 **DROP DATABASE** 명령을 실행할 수 없다. 그러나 `template1` 데이터베이스를 비롯한 다른 데이터베이스에 연결하는 것은 가능하다. 주어진 클러스트의 마지막 사용자 데이터베이스를 삭제할 때 `template1`은 유일한 옵션이 된다.

편의상 데이터베이스를 삭제할 수 있는 쉘 프로그램 dropdb도 있다.

```
dropdb dbname
```

(**createdb**와 달리, 이것은 현재 사용자 이름으로 데이터베이스를 삭제하는 기본 동작은 아니다.)

5.6. Tablespaces

Agens SQL의 테이블스페이스는 데이터베이스 관리자가 데이터베이스 객체를 나타내는 파일을 저장할 수 있는 파일 시스템의 위치를 정의할 수 있게 한다. 생성된 경우 테이블스페이스는 데이터베이스 객체 생성 시 이름으로 참조가 가능하다.

테이블스페이스를 사용함으로써 관리자는 Agens SQL 설치의 디스크 레이아웃을 제어할 수 있다. 이것은 최소 2가지 방법으로 활용할 수 있다. 첫째, 클러스터가 초기화된 파티션 또는 볼륨 공간이 소진되고 확장이 불가능한 경우, 다른 파티션에 테이블스페이스를 생성하고 시스템이 재인식될 때까지 사용할 수 있다.

두 번째, 테이블스페이스는 관리자가 성능 최적화를 위해 데이터베이스 객체의 사용 패턴에 대한 지식을 사용할 수 있게 한다. 예를 들면, 사용이 빈번한 인덱스는 고가의 SSD 같은 고속, 고가용성 디스크에 배치할 수 있다. 또한, 거의 사용되지 않거나 성능이 중요하지 않은 아카이브 데이터가 저장된 테이블은 저속, 저가의 디스크 시스템에 저장할 수 있다.

> **주의**
>
> Agens SQL 주 데이터 디렉토리 외부에 배치했더라도 테이블스페이스는 데이터베이스 클러스터 내부에 속해 있기 때문에 데이터 파일 자동 수집으로 처리되어서는 안 된다. 테이블스페이스는 주 데이터 디렉토리에 포함된 메타데이터에 종속적이므로 서로 다른 데이터베이스 클러스터에 연결되거나 개별적으로 백업할 수 없다. 마찬가지로, 테이블스페이스를 잃어버린 경우(파일 삭제, 디스크 오류 등), 데이터베이스 클러스터를 읽지 못하게 되거나 시작하지 못하게 될 수 있다. 램디스크 같은 임시 파일 시스템에 테이블 스페이스를 배치하면 전체 클러스터의 신뢰도가 위험할 수 있다.

테이블스페이스를 정의하려면 CREATE TABLESPACE 명령을 사용해야 한다. 예를 들면,

```
CREATE TABLESPACE fastspace LOCATION '/ssd1/postgresql/data';
```

위치는 빈 디렉토리로 존재해야 하며, Agens SQL 운영 체제 사용자가 소유한 것이어야 한다. 이후에 테이블스페이스 내부에서 생성된 모든 객체는 이 디렉토리 아래의 파일에 저장된다. 테이블스페이스가 누락 또는 분실된 경우 클러스터 작동이 실패할 수 있으므로 위치가 이동식 또는 임시 저장소이면 안 된다.

> 참고**:** 논리적 파일 시스템 내에서 개별 파일의 위치를 사용자가 제어할 수 없기 때문에 논리적 파일 시스템당 테이블스페이스를 2개 이상 만들 수 있는 지점이 일반적으로 많지 않다. 그러나 Agens SQL은 강제로 제한하지 않으므로, 실제로는 시스템에서 파일 시스템 경계를 직접적으로 인식하지 못한다. 단지 사용자가 사용할 것으로 지정한 디렉토리에 파일을 저장하기만 한다.

테이블스페이스 자체는 데이터베이스 수퍼유저로 생성해야 하지만, 생성한 후에는 일반 데이터베이스 사용자가 이 데이터베이스를 사용하도록 할 수 있다. 이렇게 하려면 CREATE 권한을 부여해야 한다.

테이블 및 인덱스, 전체 데이터베이스는 특정 테이블스페이스에 할당될 수 있다. 이를 위해, 주어진 테이블스페이스에 대한 CREATE 권한이 있는 사용자는 테이블스페이스 이름을 해당 명령에 대한 매개 변수로 전달해야 한다. 예를 들면, 다음은 테이블스페이스 space1에서 테이블을 생성한다.

```
CREATE TABLE foo(i int) TABLESPACE space1;
```

또는 default_tablespace 매개 변수를 사용한다.

```
SET default_tablespace = space1;
CREATE TABLE foo(i int);
```

default_tablespace가 비어 있는 string 이외의 것으로 설정되면 이것은 명시적이지 않은 **CREATE TABLE** 및 **CREATE INDEX** 명령에 대한 암시적 TABLESPACE 절을 제공한다.

임시 테이블과 인덱스, 거대(large) 데이터 세트 정렬 같은 목적으로 사용되는 임시 파일의 배치를 결정하는 temp_tablespaces 매개 변수도 있다. 이것은 하나 이상의 테이블스페이스명 목록일 수 있으므로 임시 객체와 관련된 부하가 다수의 테이블스페이스로 퍼질 수 있다. 목록의 임의의 멤버는 임시 객체가 생성될 때마다 선택된다.

데이터베이스와 관련된 테이블스페이스는 해당 데이터베이스의 시스템 카탈로그를 저장하는 데 사용된다. 또한 TABLESPACE 절이 없거나 default_tablespace 또는 temp_tablespaces에 의해 지정된 다른 선택이 없는 경우, 이것은 데이터베이스 내에서 생성된 테이블 및 인덱스, 임시 파일에 사용되는 기본 테이블스페이스이다. 테이블스페이스를 지

정하지 않고 데이터베이스를 생성하면, 복사했던 템플릿 데이터베이스와 동일한 테이블스페이스를 사용한다.

데이터베이스 클러스터가 초기화되면 테이블스페이스 2개가 자동으로 생성된다. `pg_global` 테이블스페이스는 공유 시스템 카탈로그에 사용된다. `pg_default` 테이블스페이스는 `template1` 및 `template0` 데이터베이스의 기본 테이블스페이스이다(따라서, **CREATE DATABASE**의 `TABLESPACE` 절로 덮어쓰지 않으면 다른 데이터베이스의 기본 테이블스페이스가 된다).

테이블스페이스가 생성되면 요청한 사용자에게 권한이 충분한 경우에는 모든 데이터베이스에서 사용할 수 있다. 이것은 테이블스페이스를 사용하는 모든 데이터베이스에서 모든 객체를 삭제하지 않으면 테이블스페이스를 삭제할 수 없다는 것을 의미한다.

비어 있는 테이블스페이스를 삭제하려면 DROP TABLESPACE 명령을 사용해야 한다.

기존 테이블스페이스를 확인하려면 pg_tablespace 시스템 카탈로그를 검사해야 한다. 예를 들면,

```
SELECT spcname FROM pg_tablespace;
```

psql 프로그램의 `\db` 메타 명령도 기존 테이블스페이스를 나열할 때 유용하다.

Agens SQL은 심볼릭 링크를 사용하여 테이블스페이스의 구현을 간략화한다. 이것은 심볼릭 링크가 지원되는 시스템에서만 테이블스페이스를 사용할 수 있다는 것을 의미한다.

디렉토리 `$PGDATA/pg_tblspc`에는 클러스터에 저장된 비 내장 테이블스페이스 각각을 가리키는 심볼릭 링크가 포함되어 있다. 권장 사항은 아니지만, 이 링크를 재정의하여 테이블스페이스의 레이아웃을 조정할 수도 있다. 이 작업은 어떤 경우에서든 서버 실행 중에 해야 한다. Agens SQL 9.1 이전 버전에서는 pg_tablespace 카탈로그를 새 위치로 업그레이드해야 한다. (그렇게 하지 않으면 `pg_dump`가 계속해서 이전 테이블스페이스 위치를 출력한다.)

6장. 로컬라이제이션

이 장에서는 관리자의 관점에서 사용 가능한 로컬라이제이션 기능을 설명한다. Agens SQL은 두 가지 로컬라이제이션 기능을 지원한다.

- 운영 체제의 로케일(locale) 기능을 사용하여 로케일별(locale-specific) 콜레이션 순서, 숫자 형식 설정, 메시지 번역 및 기타 이것은 6.1절 및 6.2절에서 다룬다.
- 텍스트를 모든 종류의 언어로 저장하는 것을 지원하는 각종 문자 집합 및 클라이언트와 서버 간 문자 집합 번역. 이것은 6.3절에서 다룬다.

6.1. 로케일(Locale) 지원

로케일(*Locale*) 지원이란 영문자, 정렬, 숫자 형식 등 문화적인 기본 설정과 관련된 지원을 의미한다. Agens SQL는 서버 운영 체제에서 제공되는 표준 ISO C 및 POSIX 로케일(locale) 기능을 사용한다. 부가적인 정보에 대해서는 시스템 문서를 참조 바란다.

6.1.1. 개요

로케일(locale) 지원은 **initdb**를 사용하여 데이터베이스 클러스터가 생성되면 자동으로 초기화된다. **initdb**은 기본적으로 실행 환경의 로케일(locale) 설정을 사용하여 데이터베이스 클러스터를 초기화하므로, 데이터베이스 클러스터에서 시스템이 이미 원하는 로케일(locale)을 사용하는 것으로 설정된 경우 사용자가 특별히 할 일은 없다. 다른 로케일(locale)을 사용하려면(또는 시스템에 어떤 로케일(locale)이 설정되었는지 모를 경우) `--locale` 옵션으로 **initdb** 명령을 실행하여 어떤 로케일(locale)이 사용되는지 정확히 알 수 있다. 예를 들면:

```
initdb --locale=sv_SE
```

Unix 시스템에 대한 이 예시에서, 스웨덴(`SE`)에서 사용되는 스웨덴어(`sv`)가 로케일로 설정된다. 다른 것으로는, `en_US`(U.S. English) 및 `fr_CA` (French Canadian)가 있을 수 있다. 로케일(locale)에 두 가지 이상의 문자 세트를 사용하는 경우 *`language_territory.codeset`* 규격의 양식을 따를 수 있다. 예를 들면, `fr_BE.UTF-8`는 UTF-8 문자 집합 인코딩을 사용하여 벨기에(BE)의 프랑스어(fr)를 나타낸다.

시스템에서 사용할 수 있는 로케일(locale)의 이름은 운영 체제 벤더에 따라, 설치된 것에 따라 다르다. 대부분의 Unix 시스템에서 `locale -a` 명령은 사용 가능한 로케일(locale) 목록을 보여준다. Windows는 `German_Germany` 또는 `Swedish_Sweden.1252`같이 로케일(locale) 이름을 더 자세하게 사용하지만 원리는 동일하다.

경우에 따라 몇 가지 로케일(locale) 규칙을 혼용하는 것이 유용하다. 예를 들면, English 콜레이션 규칙은 사용하되, Spanish 메시지는 사용하지 않을 수 있다. 이것을 지원하기 위해 로컬라이제이션 규칙의 특정 측면만 제어하는 로케일(locale) 보조 카테고리 집합이 존재한다.

LC_COLLATE	String 정렬 순서
LC_CTYPE	문자 분류(어떤 글자인지, 대문자도 동일한지)
LC_MESSAGES	메시지 언어

LC_MONETARY	통화 형식
LC_NUMERIC	숫자 형식
LC_TIME	날짜 및 시간 형식

카테고리 이름은 **initdb** 옵션 이름으로 번역되어 특정 카테고리의 로케일(locale) 선택을 오버라이드한다. 예를 들면, 로케일(locale)을 Canadian으로 설정하되, 통화 형식은 U.S. 규칙을 사용하려면 `initdb --locale=fr_CA --lc-monetary=en_US`를 사용해야 한다.

시스템에 로케일(locale) 지원이 안 되는 것처럼 하고 싶으면 특수한 로케일(locale) 이름 `C` 또는 동등하게 `POSIX`를 사용해야 한다.

일부 로케일(locale) 카테고리는 데이터베이스가 생성될 때 고정된 값이어야 한다. 서로 다른 데이터베이스에 대해 서로 다른 설정을 사용할 수 있지만 데이터베이스가 생성된 다음에는 해당 데이터베이스에 대한 설정을 변경할 수 없다. `LC_COLLATE` 및 `LC_CTYPE`이 이러한 카테고리이다. 이것은 인덱스 정렬 순서에 영향을 미치므로 고정된 상태로 유지되어야 하며, 그렇지 않을 경우 텍스트 칼럼의 인덱스가 손상을 입게 된다. (그러나 6.2절에 나오는 콜레이션을 사용하면 이 제한을 완화할 수 있다.) 이 카테고리의 기본값은 **initdb**가 실행 중에 결정되고 이 값은 **CREATE DATABASE** 명령에서 지정되지 않았을 경우 새 데이터베이스를 생성할 때 사용된다.

다른 로케일 카테고리는 로케일(locale) 카테고리와 이름이 동일한 서버 환경 설정 매개 변수를 설정함으로써 필요할 때마다 변경 가능하다(자세한 내용은 2.11.2절 참조). **initdb**에서 선택된 값은 실제로 환경 설정 파일 `postgresql.conf`에 작성되어 서버 시작 시에 기본값으로 사용된다. 이 값을 `postgresql.conf`에서 제거하면 서버가 실행 환경에서 설정을 상속 받는다.

서버의 로케일(locale) 동작은 클라이언트 환경이 아니라 서버에서 표시되는 환경 변수에서 결정된다는 점에 유의해야 한다. 따라서 서버를 시작하기 전에 올바른 로케일(locale) 설정을 환경 설정하는 것에 주의해야 한다. 이것의 결과는, 클라이언트와 서버가 서로 다른 로케일(locale)로 설정된 경우 출처에 따라 메시지가 다른 언어로 나타날 수 있다.

> 참고: 실행 환경에서 로케일(locale) 상속에 대해 언급할 때 이것은 대부분의 운영 체제에서 다음을 의미한다: 콜레이션이라는 지정된 로케일(locale) 카테고리의 경우 설정할 것을 찾을 때까지 LC_ALL, LC_COLLATE(또는 각 카테고리에 해당되는 변수), LANG 환경 변수가 이 순서로 검색된다. 이 환경 변수 중 어느 것도 설정되지 않은 경우 로케일(locale) 기본값은 `C`이다.
>
> 메시지 언어 설정을 위해 일부 메시지 로컬라이제이션 라이브러리도 다른 모든 로케일(locale) 설정을 오버라이드하는 환경 변수 LANGUAGE를 살펴본다. 더 자세한 건 운영 체제 문서, 특히 gettext에 대한 문서를 참조하기 바란다.

사용자가 원하는 언어로 메시지가 번역되게 하려면 빌드 시에 NLS를 선택해야 한다(`configure --enable-nls`). 다른 모든 로케일(locale) 지원은 자동으로 내장된다.

6.1.2. 동작

로케일(locale) 설정은 다음과 같은 SQL 기능에 영향을 준다.

- `ORDER BY`를 사용한 쿼리에서 정렬 순서 또는 텍스트 데이터에서 표준 비교 연산자
- `upper` 및 `lower`, `initcap` 함수
- 패턴 일치 연산자(`LIKE`, `SIMILAR TO` 및 POSIX 스타일 정규식). 대소문자 비 구분 일치 및 문자 클래스 정규식에 의한 문자 분류에 모두 영향을 미치는 로케일(locale)

- `to_char` 계열 함수
- `LIKE` 절을 사용한 인덱스 사용 능력

Agens SQL에서 `C` 또는 `POSIX`이 아닌 다른 로케일(locale)을 사용할 때의 단점은 성능이다. 문자 처리가 느려지고 `LIKE`에서 사용되는 일반 인덱스를 사용하지 못한다. 이러한 이유로, 실제로 필요한 경우에만 로케일(locale)을 사용해야 한다.

C가 아닌 로케일(locale) 하에서 `LIKE` 절을 사용한 인덱스를 Agens SQL이 이용하려면 몇 가지 커스텀 연산자 클래스가 존재해야 한다. 이것은 로케일(locale) 비교 규칙은 무시하면서 엄격한 문자별 비교를 수행하는 인덱스의 생성을 허용한다. 다른 방법은 6.2절에 설명된 대로 `C` 콜레이션을 사용하여 인덱스를 생성하는 것이다.

6.1.3. 문제

위의 설명대로 로케일(locale)이 지원되지 않으면 운영 체제의 로케일(locale) 지원이 바르게 환경 설정되었는지 확인해야 한다. 운영 체제에서 제공되는 경우 로케일(locale)이 시스템에 설치되었는지 확인하기 위해 `locale -a` 명령을 사용할 수 있다.

사용자가 생각하는 로케일(locale)을 Agens SQL이 실제로 사용 중인지 확인하라. LC_COLLATE 및 LC_CTYPE 설정은 데이터베이스가 생성될 때 결정되고, 새 데이터베이스를 생성할 때 외에는 변경할 수 없다. LC_MESSAGES 및 LC_MONETARY를 비롯한 다른 로케일(locale) 설정은 서버가 시작된 환경에 의해 처음 결정되고, 상황에 따라 바뀔 수 있다. **SHOW** 명령을 사용하면 활성화된 로케일(locale) 설정을 확인할 수 있다.

소스의 `src/test/locale` 디렉토리에는 Agens SQL의 로케일(locale)을 지원하는 테스트 세트가 포함되어 있다.

에러 메시지 텍스트를 파싱하여 서버 측 에러를 처리하는 클라이언트 애플리케이션은 서버의 메시지 언어가 다르면 명백하게 문제가 된다. 해당 애플리케이션의 작성자는 에러 코드 스키마를 대신 활용하는 것이 좋다.

메시지 번역 카탈로그를 유지하려면 Agens SQL이 선호된 언어로 표시되도록 개발하는 자원 봉사자들의 노력이 필요하다. 원하는 언어로 된 메시지가 현재 없거나 번역이 완전하지 않을 수 있기 때문에 사용자들의 개발 협조를 기다리고 있다.

6.2. 콜레이션 지원

콜레이션 기능으로 칼럼별 및 연산별 데이터의 정렬 순서 및 문자 분류 동작을 지정할 수 있다. 이것은 데이터베이스 생성 후 LC_COLLATE 및 LC_CTYPE 설정을 변경하지 못하도록 하는 제한을 풀어 준다.

6.2.1. 개념

개념적으로 콜레이션 가능한 데이터 타입의 모든 표현식은 콜레이션을 갖고 있다. (콜레이션 가능한 내장된 데이터 타입은 text 및 varchar, char이다. 사용자 정의된 베이스 타입은 콜레이션 가능으로 표시될 수 있으며, 콜레이션 가능한 데이터 타입의 도메인도 콜레이션이 가능하다.) 표현식이 칼럼 참조인 경우 표현식의 콜레이션은 칼럼이 정의한 콜레이션이다. 표현식이 상수인 경우 콜레이션은 상수 데이터 타입의 기본 콜레이션이다. 좀 더 복잡한 표현식의 콜레이션은 아래 설명대로 입력 콜레이션으로부터 결정된다.

표현식의 콜레이션은 데이터베이스에 대해 정의된 로케일(locale) 설정을 의미하는 "기본" 콜레이션이 될 수 있다. 표현식의 콜레이션을 결정되지 않은 상태로 하는 것도 가능하다. 이런 경우 콜레이션을 알아야 하는 정렬 명령 및 기타 명령은 실패한다.

데이터베이스 시스템이 정렬 또는 문자 분류를 수행해야 하는 경우 입력 표현식의 콜레이션이 사용된다. 이것은, 예를 들면 `ORDER BY` 절 및 함수 또는 < 같은 연산자 호출을 사용할 때 결정된다. `ORDER BY` 절에 적용되는 콜레이션은 단순히 정렬 키의 콜레이션이다. 함수 또는 연산자 호출에 적용되는 콜레이션은 아래 설명된 인자로부터 결정된다. 비교 연산자 외에, 콜레이션은 `lower` 및 `upper`, `initcap` 같이 대소문자 사이를 변환하는 함수와 그리고 패턴 일치 및 `to_char`와 관련 함수를 사용할 때 결정된다.

함수 또는 연산자 호출의 경우 인자 콜레이션 검사로 결정된 콜레이션이 작업 성능을 위해 런타임에 사용된다. 함수나 연산자 호출의 결과가 콜레이션 가능한 데이터 타입인 경우, 콜레이션에 대한 정보가 필요한 주변 표현식이 있을 때 콜레이션도 파싱 할 때의 함수 또는 연산자 표현식의 정의된 콜레이션으로 사용된다.

표현식의 *콜레이션 결정*은 암시적 또는 명시적일 수 있다. 이 차이는 서로 다른 콜레이션이 표현식에 나타나는 경우, 콜레이션이 결합되는 방식에 영향을 준다. `COLLATE` 절을 사용하면 명시적 콜레이션이 결정되고, 그 외 모든 콜레이션은 암시적으로 결정된다. 예를 들면, 함수 호출에서 여러 콜레이션을 결합해야 하는 경우 다음과 같은 규칙이 사용된다.

1. 입력 표현식이 명시적 콜레이션을 결정하는 경우 입력 표현식에서 명시적으로 결정된 모든 콜레이션은 동일해야 한다. 그렇지 않으면 에러가 발생한다. 명시적으로 결정된 콜레이션은 콜레이션 결합의 결과물이다.
2. 그 외에는, 모든 입력 표현식은 동일한 암시적 콜레이션을 결정하거나 기본 콜레이션을 사용해야 한다. 기본값이 아닌 콜레이션은 콜레이션 결합의 결과물이다. 그 외에는 기본 콜레이션이다.
3. 입력 표현식 사이에 기본이 아닌 암시적 콜레이션의 충돌이 있을 경우 결합은 불확정적인 콜레이션을 쓰는 것으로 간주된다. 호출하려는 특정 함수가 적용해야 할 콜레이션을 알아야 하는 경우가 아니면 이것은 에러 조건이 아니다. 그 외에는 런타임 시 에러가 발생된다.

예를 들어, 다음과 같은 테이블 정의를 생각해 보자.

```
CREATE TABLE test1 (
    a text COLLATE "de_DE",
    b text COLLATE "es_ES",
    ...
);
```

그러면,

```
SELECT a < 'foo' FROM test1;
```

이 구문에서 < 비교는, 표현식이 명시적으로 결정된 콜레이션을 기본 콜레이션과 결합하기 때문에 `de_DE` 규칙에 따라 수행된다. 그러나,

```
SELECT a < ('foo' COLLATE "fr_FR") FROM test1;
```

이 구문에서는 명시적 콜레이션이 암시적 콜레이션을 오버라이드하므로 `fr_FR` 규칙으로 비교가 수행된다. 게다가,

```
SELECT a < b FROM test1;
```

이 구문에서는 *a* 및 *b* 칼럼의 암시적 콜레이션이 충돌하므로 어떤 콜레이션을 적용할 것인지 파서로 결정할 수 없다. < 연산자는 어떤 콜레이션을 사용할 것인지 알 필요가 없으므로, 결과

적으로 에러가 발생한다. 암시적 콜레이션 지정자(specifier)를 입력 표현식에 추가하면 이 에러가 해결된다. 따라서,

```
SELECT a < b COLLATE "de_DE" FROM test1;
```

또는 동일하게, 다음과 같이 할 수 있다.

```
SELECT a COLLATE "de_DE" < b FROM test1;
```

즉, 구조상 동일한,

```
SELECT a || b FROM test1;
```

이 구문은 || 연산자가 콜레이션에 대해 무심하므로 결과적으로 에러가 발생한다. 이 결과는 콜레이션과 무관하게 동일하다.

또한 함수 또는 연산자의 결합된 입력 표현식에 할당된 콜레이션은 함수 또는 연산자가 콜레이션 가능한 데이터 타입의 결과를 전달하는 경우, 함수 또는 연산자의 결과에 적용되는 것으로 간주된다. 따라서,

```
SELECT * FROM test1 ORDER BY a || 'foo';
```

이 구문은 de_DE 규칙에 따라 정렬된다. 그러나 이 쿼리는,

```
SELECT * FROM test1 ORDER BY a || b;
```

|| 연산자는 콜레이션에 대해 알고 있을 필요가 없지만 ORDER BY 절은 그렇게 하므로 결과적으로 에러가 발생한다. 이전과 같이, 명시적 콜레이션 지정자를 사용하면 충돌을 해결할 수 있다.

```
SELECT * FROM test1 ORDER BY a || b COLLATE "fr_FR";
```

6.2.2. 콜레이션 관리

콜레이션은 SQL 이름을 운영 체제 로케일(locale)에 맵핑하는 SQL 스키마 개체이다. 특히, LC_COLLATE 및 LC_CTYPE 조합에 맵핑한다. (이름이 암시하듯 콜레이션의 주요 목적은 정렬 순서를 제어하는 LC_COLLATE를 설정하는 것이다. 그러나, 실제로는 LC_COLLATE와 다른 LC_CTYPE를 갖는 것은 거의 불필요하므로 표현식별로 LC_CTYPE 설정에 대한 다른 인프라를 생성하는 대신 한 가지 개념 하에서 이것을 수집하는 것이 더 편리하다.) 또한 콜레이션은 문자 집합 인코딩에 연결된다(6.3절 참조). 다른 인코딩으로 동일한 콜레이션 이름이 존재할 수도 있다.

모든 플랫폼에서 이름이 default 및 C, POSIX인 콜레이션을 사용할 수 있다. 추가적인 콜레이션은 운영 체제 지원에 따라 사용할 수 있다. default 콜레이션은 데이터베이스 생성 시에 지정된 LC_COLLATE 및 LC_CTYPE 값을 선택한다. C 및 POSIX 콜레이션 둘 다 ASCII 문자 "A" ~ "Z"만 글자로 처리되고 문자 코드 바이트 값으로 엄격하게 정렬이 되는 "전형적인 C" 동작을 지정한다.

운영 체제가 단일 프로그램(newlocale 및 관련 함수) 내에서 복수의 로케일(locale)을 사용하는 지원을 제공하는 경우 데이터베이스 클러스터가 초기화되면, **initdb**는 당시에 운영 체제에서 찾은 모든 로케일(locale)에 따라 콜레이션으로 시스템 카탈로그 pg_collation을 채운다. 예를 들면, 운영 체제는 de_DE.utf8이라는 이름의 로케일(locale)을 제공할 수도 있다. 그러면 **initdb**는 LC_COLLATE 및 LC_CTYPE가 둘 다 de_DE.utf8로 설정된 UTF8 인코딩에 대해 de_DE.utf8이라는 콜레이션을 생성한다. 또한 이것은 이름 일부만 추려낸 .utf8 태그를

사용하여 콜레이션을 생성한다. 따라서 작성이 간편하고 인코딩에 대한 이름 의존도가 낮은 de_DE라는 이름으로 콜레이션을 사용할 수도 있다. 그래도 콜레이션 이름의 초기 설정은 플랫폼에 따라 달라진다.

LC_COLLATE 및 LC_CTYPE에 대한 값이 서로 다른 콜레이션이 필요한 경우 CREATE COLLATION 명령을 사용하여 새로운 콜레이션을 생성할 수도 있다. 또한 해당 명령으로 기존 콜레이션에서 새 콜레이션을 생성할 수도 있으며, 이것은 애플리케이션에서 운영 체제로부터 독립된 콜레이션 이름을 사용할 때 유용하다.

특정한 데이터베이스 내에서 데이터베이스의 인코딩을 사용하는 콜레이션만 중요하다. pg_collation의 다른 항목은 무시된다. 따라서 de_DE 같이 일부만 추려낸 콜레이션 이름은 전역적으로는 고유하지 않더라도 지정된 데이터베이스 내에서는 고유한 것으로 간주될 수 있다. 데이터베이스 인코딩을 다른 것으로 변경하기로 결정한 경우 변경 사항이 적은 추려낸 콜레이션 이름을 사용하는 것이 권장한다. 그러나 default 및 C, POSIX 콜레이션은 데이터베이스 인코딩과 무관하게 사용할 수 있다.

Agens SQL은 속성이 동일한 경우에도 별개의 콜레이션 개체는 호환되지 않는 것으로 간주한다. 따라서 예를 들면,

```
SELECT a COLLATE "C" < b COLLATE "POSIX" FROM test1;
```

이 구문은 C 및 POSIX 콜레이션의 동작이 동일하더라도 에러가 발생된다. 따라서 추려낸 콜레이션 이름과 추려내지 않은 콜레이션 이름의 혼용은 권장하지 않는다.

6.3. 문자 집합 지원

Agens SQL에서 지원하는 문자 집합을 사용하면 ISO 8859 같은 싱글바이트 문자 집합 및 EUC(Extended Unix Code), UTF-8, 뮬 내부코드 같은 멀티바이트 문자 집합을 비롯한 다양한 문자 집합으로 텍스트를 저장할 수 있다. 지원되는 모든 문자 집합은 클라이언트에서 사용 가능한 것이 확실하지만, 몇 가지는 서버 내에서의 사용이 지원되지 않는다(즉, 서버 측 인코딩). 기본 문자 집합은 **initdb**를 사용하여 Agens SQL 데이터베이스 클러스터를 초기화하면서 선택된다. 데이터베이스를 생성할 때 오버라이딩돼서, 데이터베이스들마다 각각 다른 문자 집합을 사용할 수 있다.

하지만, 각 데이터베이스 문자 집합은 데이터베이스의 LC_CTYPE(문자 분류) 및 LC_COLLATE(string 정렬 순서) 로케일(locale) 설정과 호환되어야 한다는 제약 사항이 있다. C 또는 POSIX 로케일(locale)의 경우 임의의 문자 집합이 허용되지만, 다른 로케일(locale)의 경우 올바르게 작동되는 문자 집합은 하나밖에 없다. (그러나 Windows의 경우 UTF-8 인코딩은 모든 로케일(locale)에서 사용할 수 있다.)

6.3.1. 지원되는 문자 집합

표 6-1은 Agens SQL에서 사용할 수 있는 문자 집합을 보여준다.

표 **6-1. Agens SQL** 문자 집합

이름	설명	언어	서버?	바이트/문자	별칭
BIG5	Big Five	중국어 번체	아니요	1-2	WIN950, Windows950
EUC_CN	Extended UNIX Code-CN	중국어 간체	예	1-3	

이름	설명	언어	서버?	바이트/문자	별칭
EUC_JP	Extended UNIX Code-JP	일본어	예	1-3	
EUC_JIS_2004	Extended UNIX Code-JP, JIS X 0213	일본어	예	1-3	
EUC_KR	Extended UNIX Code-KR	한국어	예	1-3	
EUC_TW	Extended UNIX Code-TW	중국어 번체, 대만	예	1-3	
GB18030	National Standard	중국어	아니요	1-2	
GBK	Extended National Standard	중국어 간체	아니요	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	라틴어/키릴어	예	1	
ISO_8859_6	ISO 8859-6, ECMA 114	라틴어/아랍어	예	1	
ISO_8859_7	ISO 8859-7, ECMA 118	라틴어/그리스어	예	1	
ISO_8859_8	ISO 8859-8, ECMA 121	라틴어/히브리어	예	1	
JOHAB	JOHAB	한국어 (한글)	아니요	1-3	
KOI8R	KOI8-R	키릴어 (러시아어)	예	1	KOI8
KOI8U	KOI8-U	키릴어 (우크라이나어)	예	1	
LATIN1	ISO 8859-1, ECMA 94	서유럽어	예	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	중유럽어	예	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	남유럽어	예	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	북유럽어	예	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	터키어	예	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	스칸디나비아어	예	1	ISO885910
LATIN7	ISO 8859-13	발트어	예	1	ISO885913
LATIN8	ISO 8859-14	켈트어	예	1	ISO885914
LATIN9	ISO 8859-15	유로 및 액센트 사용 LATIN1	예	1	ISO885915

이름	설명	언어	서버?	바이트/문자	별칭
LATIN10	ISO 8859-16, ASRO SR 14111	루마니아어	예	1	ISO885916
MULE_INTERNAL	Mule 내부 코드	Multilingual Emacs	예	1-4	
SJIS	Shift JIS	일본어	아니요	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	일본어	아니요	1-2	
SQL_ASCII	미지정 (텍스트 참조)	아무거나	예	1	
UHC	Unified Hangul Code	한국어	아니요	1-2	WIN949, Windows949
UTF8	유니코드, 8비트	모두	예	1-4	Unicode
WIN866	Windows CP866	키릴어	예	1	ALT
WIN874	Windows CP874	태국어	예	1	
WIN1250	Windows CP1250	중유럽어	예	1	
WIN1251	Windows CP1251	키릴어	예	1	WIN
WIN1252	Windows CP1252	서유럽어	예	1	
WIN1253	Windows CP1253	그리스어	예	1	
WIN1254	Windows CP1254	터키어	예	1	
WIN1255	Windows CP1255	히브리어	예	1	
WIN1256	Windows CP1256	아랍어	예	1	
WIN1257	Windows CP1257	발트어	예	1	
WIN1258	Windows CP1258	베트남어	예	1	ABC, TCVN, TCVN5712, VSCII

모든 클라이언트 API가 나열된 모든 문자 집합을 지원하는 것은 아니다. 예를 들면, Agens SQL JDBC 드라이버는 MULE_INTERNAL 및 LATIN6, LATIN8, LATIN10을 지원하지 않는다.

SQL_ASCII 설정은 다른 설정과 상당히 다르게 동작한다. 서버 문자 집합이 SQL_ASCII인 경우 서버는 ASCII 표준에 따라 바이트 값 0-127을 해석하고, 바이트 값 128-255은 해석 불가 문자로 처리한다. 설정이 SQL_ASCII인 경우 인코딩 변환이 완료되지 않는다. 따라서 이 설정은 사용 중인 특정 인코딩에 대한 선언이라기 보다는 인코딩에 대한 무시 선언이다. 대부분의 경

우에, ASCII가 아닌 데이터를 사용 시 Agens SQL은 비 ASCII 문자를 변환하거나 검증할 수 없기 때문에 SQL_ASCII 설정을 사용하는 것은 현명하지 않다.

6.3.2. 문자 집합 설정

initdb는 Agens SQL 클러스터에 대한 기본 문자 집합(인코딩)을 정의한다. 예를 들면,

```
initdb -E EUC_JP
```

이것은 기본 문자 집합을 EUC_JP (Extended Unix Code for Japanese)로 설정한다. 옵션 string이 긴 것을 선호하는 경우 --encoding을 -E 대신 사용할 수 있다. -E 또는 --encoding 옵션을 지정하지 않으면 **initdb**는 지정되었거나 기본 로케일(locale)을 기반으로 사용할 적정 인코딩을 결정하려고 한다.

인코딩이 선택한 로케일(locale)과 호환되는 경우 사용자는 기본값이 아닌 인코딩을 데이터베이스 생성 시에 지정할 수 있다.

```
createdb -E EUC_KR -T template0 --lc-collate=ko_KR.euckr --lc-ctype=ko_KR.euckr korean
```

이것은 문자 집합 EUC_KR 및 로케일(locale) ko_KR을 사용하는 korean이라는 데이터베이스를 생성한다. 다른 방법으로 아래와 같은 SQL 명령을 사용할 수 있다.

```
CREATE DATABASE korean WITH ENCODING 'EUC_KR' LC_COLLATE='ko_KR.euckr' LC_CTYPE='ko_KR.eu
```

위의 명령은 template0 데이터베이스를 복사한다는 점에 유의하라. 다른 데이터베이스를 복사할 때 데이터 손상의 우려 때문에 원본 데이터베이스의 인코딩과 로케일(locale) 설정은 변경할 수 없다. 자세한 내용은 5.3절을 참조 바란다.

데이터베이스의 인코딩은 시스템 카탈로그 pg_database에 저장되어 있다. **psql** -l 옵션 또는 **\l** 명령을 사용하면 이것을 확인할 수 있다.

```
$ psql -l
                                        List of databases
   Name    |  Owner   | Encoding  |  Collation  |    Ctype    |          Access Privilege
-----------+----------+-----------+-------------+-------------+---------------------------
 clocaledb | hlinnaka | SQL_ASCII | C           | C           |
 englishdb | hlinnaka | UTF8      | en_GB.UTF8  | en_GB.UTF8  |
 japanese  | hlinnaka | UTF8      | ja_JP.UTF8  | ja_JP.UTF8  |
 korean    | hlinnaka | EUC_KR    | ko_KR.euckr | ko_KR.euckr |
 postgres  | hlinnaka | UTF8      | fi_FI.UTF8  | fi_FI.UTF8  |
 template0 | hlinnaka | UTF8      | fi_FI.UTF8  | fi_FI.UTF8  | {=c/hlinnaka,hlinnaka=CT
 template1 | hlinnaka | UTF8      | fi_FI.UTF8  | fi_FI.UTF8  | {=c/hlinnaka,hlinnaka=CT
(7 rows)
```

> 중요: 대부분의 최신 운영 체제에서 Agens SQL은 LC_CTYPE 설정으로 어떤 문자 집합이 암시되는지를 판단하며, 일치하는 데이터베이스 인코딩만 강제로 사용되도록 한다. 예전 시스템에서는 선택한 로케일(locale)에서 예상되는 올바른 인코딩을 사용하는 것이 사용자의 책임이었다. 여기서 실수를 했을 경우 정렬(sorting) 같이 로케일(locale)에 의존적인 작업은 이상하게 동작한다.
>
> Agens SQL은 LC_CTYPE이 C또는 POSIX가 아니더라도 수퍼유저가 SQL_ASCII 인코딩을 사용하여 데이터베이스를 생성하는 것을 가능하게 한다. 위에서 명시한 대로, SQL_ASCII는 데이터베이스에 저장된 데이터가 특정한 인코딩을 강제로 갖게 하지 않으므로 SQL_ASCII을 선택하면 로케일(locale)에 의존적인 동작은 잘못될 가능성이 있다. 이러한 설정 조합은 사용할 수 없으며 언젠가는 전면적으로 금지될 수 있다.

6.3.3. 서버와 클라이언트 간 자동 문자 집합 변환

Agens SQL은 서버와 클라이언트 간 특정 문자 집합 조합에 대해 자동 문자 집합 변환을 지원한다. 변환 정보는 pg_conversion 시스템 카탈로그에 저장된다. Agens SQL은 표 6-2에 나오는 사전 정의된 변환 몇 가지를 제공한다. SQL 명령 **CREATE CONVERSION**을 사용하면 새로운 변환을 생성할 수 있다.

표 **6-2.** 클라이언트/서버 문자 집합 변환

서버 문자 집합	사용 가능한 클라이언트 문자 집합
BIG5	서버 인코딩으로 지원되지 않음
EUC_CN	**EUC_CN**, MULE_INTERNAL, UTF8
EUC_JP	**EUC_JP**, MULE_INTERNAL, SJIS, UTF8
EUC_KR	**EUC_KR**, MULE_INTERNAL, UTF8
EUC_TW	**EUC_TW**, BIG5, MULE_INTERNAL, UTF8
GB18030	서버 인코딩으로 지원되지 않음
GBK	서버 인코딩으로 지원되지 않음
ISO_8859_5	**ISO_8859_5**, KOI8R, MULE_INTERNAL, UTF8, WIN866, WIN1251
ISO_8859_6	**ISO_8859_6**, UTF8
ISO_8859_7	**ISO_8859_7**, UTF8
ISO_8859_8	**ISO_8859_8**, UTF8
JOHAB	**JOHAB**, UTF8
KOI8R	**KOI8R**, ISO_8859_5, MULE_INTERNAL, UTF8, WIN866, WIN1251
KOI8U	**KOI8U**, UTF8
LATIN1	**LATIN1**, MULE_INTERNAL, UTF8
LATIN2	**LATIN2**, MULE_INTERNAL, UTF8, WIN1250
LATIN3	**LATIN3**, MULE_INTERNAL, UTF8
LATIN4	**LATIN4**, MULE_INTERNAL, UTF8
LATIN5	**LATIN5**, UTF8
LATIN6	**LATIN6**, UTF8
LATIN7	**LATIN7**, UTF8
LATIN8	**LATIN8**, UTF8
LATIN9	**LATIN9**, UTF8
LATIN10	**LATIN10**, UTF8
MULE_INTERNAL	**MULE_INTERNAL**, BIG5, EUC_CN, EUC_JP, EUC_KR, EUC_TW, ISO_8859_5, KOI8R, LATIN1 to LATIN4, SJIS, WIN866, WIN1250, WIN1251
SJIS	서버 인코딩으로 지원되지 않음
SQL_ASCII	아무거나(변환이 수행되지 않음)
UHC	서버 인코딩으로 지원되지 않음

서버 문자 집합	사용 가능한 클라이언트 문자 집합
UTF8	인코딩 모두 지원
WIN866	**WIN866**, ISO_8859_5, KOI8R, MULE_INTERNAL, UTF8, WIN1251
WIN874	**WIN874**, UTF8
WIN1250	**WIN1250**, LATIN2, MULE_INTERNAL, UTF8
WIN1251	**WIN1251**, ISO_8859_5, KOI8R, MULE_INTERNAL, UTF8, WIN866
WIN1252	**WIN1252**, UTF8
WIN1253	**WIN1253**, UTF8
WIN1254	**WIN1254**, UTF8
WIN1255	**WIN1255**, UTF8
WIN1256	**WIN1256**, UTF8
WIN1257	**WIN1257**, UTF8
WIN1258	**WIN1258**, UTF8

자동 문자 집합 변환을 활성화하려면 클라이언트에서 사용하려는 문자 집합(인코딩)을 Agens SQL에 알려 주어야 한다. 이렇게 하는 방법에는 몇 가지가 있다.

- psql에서 **\encoding** 명령 사용. **\encoding**을 사용하면 클라이언트 인코딩을 그때그때 변경할 수 있다. 예를 들면, 인코딩을 SJIS로 변경하려면 다음을 입력한다.

  ```
  \encoding SJIS
  ```

- libpq에는 클라이언트 인코딩을 제어하는 기능이 있다.
- **SET client_encoding TO** 사용.

  ```
  SET CLIENT_ENCODING TO 'value';
  ```
  또는 표준 SQL 구문 SET NAMES을 사용할 수도 있다.

  ```
  SET NAMES 'value';
  ```
  현재 클라이언트 인코딩을 확인하려면,

  ```
  SHOW client_encoding;
  ```
  기본 인코딩을 리턴하려면,

  ```
  RESET client_encoding;
  ```

- PGCLIENTENCODING 사용. 환경 변수 PGCLIENTENCODING이 클라이언트의 환경에서 정의된 경우 해당 클라이언트 인코딩은 서버 연결 시 자동으로 선택된다. (위에 언급된 다른 방법을 사용하면 나중에 이것을 무시할 수 있다.)
- 환경 설정 변수 client_encoding 사용. client_encoding 변수가 설정된 경우 해당 클라이언트 인코딩은 서버 연결 시 자동으로 선택된다. (위에 언급된 다른 방법을 사용하면 나중에 이것을 무시할 수 있다.)

특정한 문자 변환이 불가능한 경우 사용자가 서버에 대해서는 EUC_JP를 선택하고 클라이언트에 대해서는 LATIN1을 선택한 것으로 추정되며, LATIN1 표현이 없는 일본어 문자가 리턴된다. 에러가 발생한다.

클라이언트 문자 집합이 SQL_ASCII로 설정된 경우 서버의 문자 집합과 무관하게 인코딩 변환이 비활성화된다. 서버의 경우처럼 모든 ASCII 데이터를 사용하지 않을 때 SQL_ASCII는 바람직하지 않다.

6.3.4. 추가 자료

이 소스는 다양한 인코딩 시스템을 배우는 데 도움이 된다.

CJKV Information Processing: Chinese, Japanese, Korean & Vietnamese Computing

EUC_JP, EUC_CN, EUC_KR, EUC_TW에 대한 자세한 설명이 나와 있다.

http://www.unicode.org/

유니코드 컨소시엄 웹사이트.

RFC 3629

UTF-8 (8-bit UCS/Unicode Transformation Format)가 여기에 정의되어 있다.

7장. 정기적인 데이터베이스 관리 작업들

다른 데이터베이스 소프트웨어와 마찬가지로, Agens SQL에서도 최적의 성능을 유지하지 위해서는 주기적인 관리 작업이 필요하다. 여기서 관리 작업이란 꼭 필요한 작업이지만 이 작업들은 대부분 항상 같은 형태로 반복 되는 작업이기 때문에, cron이나 윈도우즈의 작업 스케줄러 같은 프로그램을 이용해서 그 작업을 등록해서 주기적으로 실행하면 된다. 물론, 이런 작업 스크립트를 작성 하는 일과 그 작업이 정상적으로 실행 되었는지를 확인하는 일은 데이터베이스 관리자의 몫이다.

주기적인 작업으로 가장 으뜸은 당연히 백업이다. 장애(하드웨어 측면의 장애이거나 운영상 사용자 실수에 의한 장애이거나)가 발생했을 때, 마지막 보루는 가장 최근에 백업한 자료로 데이터베이스를 복구하는 것이다. 백업과 복구에 대한 자세한 이야기는 8장에서 소개하고 있다. (데이터베이스 관리자들 사이 흔한 농담으로 테이블을 날려 먹는 관리자는 용서 할 수 있어도, 백업 관리를 하지 않는 관리자는 용서할 수 없다고 한다.)

또 다른 중요한 정기적인 관리 작업은 데이터베이스 "vacuum 하기"이다. 이 부분에 대한 자세한 이야기는 7.1절에서 다룬다. 이어서 쿼리 실행 계획기가 바른 실행 계획을 짜도록 테이블의 자료 통계 정보를 갱신 하는 것에 대한 부분은 7.1.3절에서 다룬다.

또 다른 작업으로는 서버 로그 파일 관리 작업인데, 이 부분에 대해서는 7.3절에서 다룬다.

check_postgres (http://bucardo.org/wiki/Check_postgres) 프로그램은 데이터베이스 상태를 지켜보고, 비정상적인 상태에 대해서 보고하는 일을 한다. Nagios나 MRTG에서 check_postgres 프로그램을 이용하면 유용하게 쓰일 것이다. 물론 이 프로그램 단독으로 사용해도 충분히 그 역할을 한다.

Agens SQL은 다른 데이터베이스 관리 시스템에 비해서 비교적 적은 관리 비용이 든다. 그럼에도 불구하고, 이런 정기적인 관리 작업에 대한 어느 정도의 관심은 보다 쾌적한 데이터베이스 서비스 품질을 제공할 수 있을 것이다.

7.1. 정기적인 Vacuum 작업

Agens SQL 데이터베이스에서는 *vacuum* (배큠이라고 읽는다)이라는 주기적인 관리 작업이 필요하다. 이 작업은 대부분의 서버 환경에서는 *autovacuum 데몬*이 담당해서 자동으로 처리되기 때문에, 특별히 신경쓸 필요는 없다. 이 부분은 7.1.6절에서 자세히 소개한다. autovacuum 관련 환경 설정값을 바꾸어서 그 데몬의 동작 상태를 조절 할 수 있다. 또한 일부 데이터베이스 관리자는 **VACUUM** 명령을 직접 실행하는 것이 데이터베이스를 효율적으로 사용할 수 있다고 판단해서, cron 이나 작업 스케줄러 같은 프로그램을 이용해서 이런 주기적인 정리 작업을 한다. 이런 작업을 잘 하기 위해서는 여기서 설명하고 있는 내용을 잘 이해하고 있어야한다. autovacuum 기능을 이용하는 일반적인 환경에서 특별히 문제가 발생하지 않는다면 이 부분은 대충 읽어도 좋다.

7.1.1. Vacuum 기초

Agens SQL에서 VACUUM 명령은 다음과 같은 여러 가지 이유로 정기적으로 각 테이블 단위로 실행되어야 한다.

1. 변경 또는 삭제된 자료들이 차지 하고 있는 디스크 공간을 다시 사용하기 위한 디스크 공간 확보 작업이 필요하다.
2. Agens SQL 쿼리 실행 계획기가 사용할 자료 통계 정보를 갱신할 필요가 있다.

3. 인덱스 전용 검색 성능을 향상하는데 이용하는 실자료 지도(visibility map, vm) 정보를 갱신하는 작업이 필요하다.
4. *트랜잭션 ID 겹침*이나 *다중 트랙잭션 ID 겹침* 상황으로 오래된 자료가 손실될 가능성을 방지할 필요가 있다.

이런 이유로 **VACUUM** 작업은 그 작업 이유에 맞게 다양한 주기로 다양한 대상으로 진행된다. 이 부분에 대한 자세한 설명은 이 글의 하위 항목에서 각각 설명한다.

VACUUM 작업은 두 가지 종류가 있다. 표준 **VACUUM**과 **VACUUM FULL**이다. **VACUUM FULL** 작업은 물리적인 디스크 여유 공간을 확보할 수 있으나 그 작업 속도가 매우 느리다. 하지만 표준 **VACUUM** 작업은 운영 환경에서도 사용할 수 있도록 여러 다른 작업들(**SELECT**, **INSERT**, **UPDATE**, **DELETE** 같은 명령어로 수행되는 작업들)이 실행 되고 있어도 동시에 사용할 수 있다. (하지만, **ALTER TABLE** 명령과 같은 명령은 VACUUM 작업이 실행되고 있는 상황에서는 사용할 수 없다.) **VACUUM FULL** 명령은 해당 테이블에 대한 배타적 잠금(exclusive lock)을 지정하기 때문에 어떤 작업도 할 수 없게 된다. 그러므로 일반적 상황에서는 관리자는 **VACUUM FULL** 작업을 되도록이면 피하고 표준 **VACUUM** 작업을 하도록 해야 한다.

VACUUM 작업은 추가적으로 디스크 입출력 부하를 만든다. 이 때문에 동시에 작업하고 있는 다른 세션의 성능을 떨어뜨린다. 이 부분은 2.4.4절 에서 소개하고 있는 VACUUM 작업의 비용 조절 관련 환경 설정을 변경해서 어느 정도는 조절이 가능하다.

7.1.2. 디스크 여유 공간 확보

Agens SQL에서는 **UPDATE**나 **DELETE** 작업 대상이 된 해당 자료의 옛 버전을 작업 완료 후 바로 버리지 않는다. 이 작업은 다중 버전 동시성 제어(multiversion concurrency control, MVCC) 기법을 구현하는데 이점이 있기 때문이다. 삭제된 자료를 다른 트랜잭션에서 사용하고 있을 때 그 자료를 삭제하면 안되기 때문이다. 하지만 다른 트랜잭션이 더 이상 그 옛 버전 자료에 대한 접근이 필요 없으면 옛 버전 자료는 쓸모 없는 자료가 된다. 이 상태로 계속 운영되면 디스크에는 쓸모 없는 자료들이 넘쳐나게 될 것이다. 더 이상 사용할 수 없고 사용해서는 안될 자료들을 정리해서 그 자료가 있었던 공간을 빈 공간으로 바꾸는 작업을 **VACUUM** 명령이 담당한다.

VACUUM 기본 작업은 테이블과 인덱스에서 삭제된 자료 (old version row, dead row 라고 한다) 를 정리하고, 그 자리를 다른 자료가 저장 될 수 있도록 빈공간으로 표시하는 것이다. 하지만 이 작업은 운영체제 입장에서의 디스크 여유 공간을 확보하는 것을 의미하지는 않는다. 물론 한 테이블의 자료가 모두 지워졌고, 하나 또는 소수의 페이지만 없애면 되는데 이 작업을 위해 테이블 전체의 배타적 잠금도 쉽게 할 수 있는 상황과 같이 특별한 경우는 해당 페이지를 삭제 해서 운영체제 입장의 디스크 여유 공간을 확보할 수도 있다. 이와 반대로, **VACUUM FULL** 작업은 해당 테이블의 사용할 수 있는 자료들만을 따로 모아 아예 새 파일에 저장하는 방식을 이용하기 때문에 운영체제 입장에서 디스크 여유 공간을 확보할 수 있다. 작업 결과로 해당 테이블에 대해서 최적의 물리적 크기로 테이블이 만들어진다. 하지만 그 작업은 일반 VACUUM 작업에 비해 시간이 꽤 걸린다. 또한 이 작업이 완료되기 전까지 이 작업을 할 수 있는 여유 공간이 있어야 작업을 할 수 있다.

일반적인 vacuum 전략은 주기적인 표준 **VACUUM** 작업을 해서 꾸준히 빈 공간을 확보하고, 디스크가 어느 정도 커지면 더 이상 커지지 않게 하는 것이다. 이는 최대한 **VACUUM FULL** 작업을 해야하는 상황을 방지하기 위한 전략이다. autovacuum 데몬이 이런 전략으로 작업을 한다. 즉, autovacuum 기능을 사용한다면, **VACUUM FULL** 작업을 하지 않는 것을 기본 정책으로 하면 된다. 이런 전략은 각 테이블이 최소의 디스크 공간을 사용한다는 것을 의미하는 것이 아니라 최적의 디스크 공간을 사용함을 의미한다. 각 테이블은 실제 자료가 저장되어 있는 공간과 함께 vacuum 작업으로 처리된 빈공간을 함께 사용한다. 반면 어떤 테이블은 더이상 변경/삭제 작업이 없어 최소의 디스크 공간만 사용하면 된다고 판단되면 **VACUUM FULL** 명

령을 이용할 수도 있다. 이렇게 해서 쓸모 없는 공간을 운영체제 쪽으로 반환할 수도 있다. 대용량 테이블을 관리하는 입장에서 보면 비정기적인 **VACUUM FULL** 작업보다 정기적인 표준 **VACUUM** 작업이 운영상 더 낫다.

어떤 관리자는 vacuum 작업을 사용량이 적은 밤 시간에 주기적으로 작업 하도록 직접 관리하려고 한다. 이렇게 특정 시간에 vacuum 작업을 할 때는 갑자기 자료 변경이 많은 작업이 생겨 디스크 공간을 많이 쓰게 되는 경우도 함께 고려해야 한다. 최악의 경우는 디스크 공간이 모자라 **VACUUM FULL** 작업을 선택해야 할 경우도 생기기 때문이다. autovacuum 데몬을 이용하면 이런 예상치 못한 상황에 대해서도 자동으로 vacuum 작업이 진행되어 위와 같은 문제들을 피해갈 수 있다. 정확한 데이터베이스 사용량을 파악하지 않고 그냥 autovacuum 기능을 끄는 것은 현명하지 못한 선택이다. 한 가지 대안은 이 데몬의 실행 환경 설정값을 변경해서 예상치 못한 대량 변경 작업에 대해서만 vacuum 작업이 자동으로 실행 되게 하는 것이다. 동시에 주기적인 관리자 정의 **VACUUM** 작업도 하면서 예외 상황에 대해서 자동으로 대처하도록 하는 것이 좋다.

autovacuum 기능을 사용하지 않을 때 주의해야할 점은 해당 데이터베이스 서버에서 사용하고 있는 모든 데이터베이스에 대해서 **VACUUM** 작업을 해야한다는 것이다. 일반적으로 하루에 한 번 밤시간에 지정하는 것이 일반적이며 자료가 빈번하게 변경되는 테이블에 대해서는 더 자주 vacuum 작업 하도록 설정한다. (아주 빈번한 테이블에 대해서 몇 분에 한 번씩 작업 하도록 설정해야할 필요도 있을 것이다.) 각 데이터베이스별 vacuum 작업을 할 때 vacuumdb 응용 프로그램을 이용하면 도움이 될 것이다.

> 작은 정보**:** 표준 **VACUUM** 작업은 해당 테이블 전체를 대상으로 하는 변경 작업이나 삭제 작업과 같은 대량의 작업에 대해서는 만족할 만한 결과를 제공하지는 않는다. 이 경우, 디스크 여유 공간을 확보해야 할 필요성이 있으면 **VACUUM FULL** 명령이나 CLUSTER 또는 ALTER TABLE 명령(CLUSTER ON 옵션)을 이용해서 테이블을 아예 새롭게 만드는 방법을 선택할 수 있다. 이들 명령을 사용할 때 반드시 주의해야할 사항은 이들 명령은 테이블을 대상으로 배타적 잠금을 한다는 점이다. 즉, 이 작업이 완료되기 전까지 그 테이블을 대상으로 하는 다른 모든 작업들을 다른 세션에서는 할 수 없음을 염두해 두어야 하며 이 작업은 작업 도중 원본과 다른 새로운 복사본 자료를 만들기 때문에 그만큼의 디스크 공간이 필요하다는 것도 기억하고 있어야 한다.

> 작은 정보**:** 테이블의 모든 자료를 아예 지워버리고자 하면 **DELETE** 명령보다는 TRUNCATE 명령을 사용하는 것이 낫다. 이 명령은 작업 뒤에 **VACUUM** 이나 **VACUUM FULL** 명령을 사용할 필요가 없다. 단, 이 명령을 사용하게 되면 MVCC 대상에서 제외되기 때문에 다중 세션 다중 트랜잭션 환경에서는 위험하다.

7.1.3. 실행계획 통계 정보 갱신

Agens SQL 쿼리 실행 계획기는 쿼리의 좋은 실행 계획을 짜기 위해서 각 테이블에 저장된 자료를 바탕으로 수집된 통계 정보를 이용한다. 이 통계 정보는 ANALYZE 명령을 이용해서 만든다. 또한 **VACUUM** 명령을 수행하면서 옵션으로 이 작업을 할 수 있다. 이 통계 정보 갱신 작업이 제대로 되지 않으면 의도 하지 않은 쿼리 실행 계획이 짜여질 것이고 이것은 전체적으로 데이터베이스 성능을 떨어뜨리는 결과를 초래하기 때문에 바른 통계 정보 갱신 작업을 주기적으로 하는 것은 중요하다.

autovacuum 기능을 이용하면 통계 정보를 갱신해야할 필요성이 있는 테이블들에 대해서 주기적으로 **ANALYZE** 명령을 자동 수행한다. 반면, 자료 변경 작업이 어떤 칼럼에서 일어나고 그 변경 작업 때문에 통계 정보를 갱신해서 실행 계획이 잘 짜여지는 것과 관련 없다고 명확하게 판단되는 경우, 관리자는 이 통계 정보 갱신 작업을 직접 수행하거나 아예 안 할 수도 있

다. autovacuum 데몬은 이런 세세한 경우까지는 고려하지 않고 테이블의 자료가 새로 추가 되었거나 변경/삭제 된 경우, 무조건 통계 정보 갱신 작업 대상으로 판단한다.

통계 정보 갱신 작업도 디스크 여유 공간 확보를 위한 vacuum 작업과 마찬가지로 테이블의 자료 변화량이 많은 경우는 보다 빈번하게, 그 반대의 경우는 좀 더 드물게 진행하는 것이 좋다. 물론, 테이블의 자료가 빈번하게 변경된다고 하더라도 그 변경 내용이 수집할 통계 정보와 관련 없는 것이면 당연히 통계 정보 갱신 작업이 필요 없다. 통계 정보를 갱신 할 빈도를 추측하는 가장 간단한 방법은 그 칼럼의 최소값과 최대값이 얼마나 자주 바뀌느냐를 살펴보는 것이다. 예를 들어 웹서비스에서 각 페이지의 마지막 접근 정보를 기록하는 테이블에서 마지막 접근 시각을 기록하는 timestamp 자료형의 칼럼은 URL을 담고 있는 칼럼보다 훨씬 빈번하게 변경될 것이다. 따라서 URL 정보의 통계 정보 갱신 작업보다 마지막 접근 시각의 통계 정보 갱신이 더 빈번하게 일어나야 보다 정확한 실행 계획이 짜여질 것이다.

ANALYZE 명령을 사용자가 직접 실행 할 때 한 테이블의 특정 칼럼 정보에 대해서만 통계 정보를 갱신하도록 할 수 있다. 이렇게 하면 위에서 언급한 것처럼 칼럼별 통계 정보 갱신 작업 빈도를 칼럼별로 조절할 수 있다. 하지만 운영 환경에서는 일반적으로 데이터베이스 전체를 대상으로 이 작업을 한다. 왜냐하면 기본적으로 자료 통계 정보 갱신 작업은 전체 자료를 대상으로 하지 않고 임의의 샘플 자료만을 대상으로 하기 때문에 꽤 빨리 작업이 끝난다.

> 작은 정보**:** 칼럼 단위로 **ANALYZE** 작업을 하는 것은 운영 환경에서 사용하기에는 번거로운 일이긴 하지만, 보다 정확한 통계 정보를 싼 비용으로 갱신 할 수 있다는 것이 장점이다. 예를 들어 한 칼럼의 자료 분포가 아주 넓고 그 칼럼이 `WHERE` 절의 조건 검색으로 자주 사용되면, 이 칼럼의 통계 정보는 다른 칼럼보다 더 꼼꼼하게 관리되는 것이 좋을 것이다. 이 경우, **ALTER TABLE SET STATISTICS** 명령을 통해 해당 테이블의 개별 통계 수집 설정값을 바꾸어서 꼼꼼하게 관리 할 수 있다. 또한 서버 환경 설정인 defaults-statistics-target 값을 조정해서 데이터베이스 전체를 대상으로 조절할 수도 있다.
>
> 또, 함수 사용에 대한 통계 정보는 기본적으로 제한된 정보만 제공 한다. 하지만 함수 기반 인덱스를 만들면 이 부분에 대해서는 통계 정보 갱신 작업이 함수 반환값을 대상으로 이루어 지기 때문에 검색 조건으로 인덱스를 만들 때처럼 해당 칼럼에 해당 함수를 사용하다면 쿼리 실행 계획기는 바르게 실행 계획을 짤 것이다.

> 작은 정보**:** autovacuum 데몬은 외부 테이블(foreign table)에 대해서는 **ANALYZE** 작업을 하지 않는다. 외부 테이블 사용시 그것의 통계 정보가 갱신 되야 할 때는 직접 **ANALYZE** 명령을 주기적으로 실행해야 할 것이다.

7.1.4. 실자료 지도 갱신

vacuum 작업은 실자료 지도를 갱신하는 작업을 한다. 실자료 지도(visibility map, vm)란 현재 작업 중인 트랜잭션들(또는 그 자료들이 변경 되기 전까지 이용할 미래의 모든 트랜잭션들)이 실제로 사용할 자료들에 대한 각 테이블별 지도다. 이 작업은 두가지 목적이 있다. 하나는 vacuum 작업은 이미 지도 정리 작업이 끝난 것에 대해서는 더 이상 그 작업을 하지 않는 것이다.

다른 하나는 이 지도 정보는 인덱스 전용 쿼리들(더 이상 실제 테이블 자료를 검사 하지 않는 쿼리들)에 대해서 빠른 응답을 제공하는데 사용하는 것이다. Agens SQL의 인덱스에는 실자료들에 대해서만 따로 모아서 그 정보를 제공하지 않는다. 즉, 어떤 자료를 해당 세션에게 보여 주어야 할지를 결정 하는 정보는 그 자료의 테이블 페이지까지 살펴 보아야 알 수 있다. *인덱스 전용 검색*인 경우는 테이블 페이지를 검색하지 않고 먼저 이 실자료 지도를 검색해서 이곳에 해당 자료가 있으면 사용한다. 그만큼 테이블 페이지 읽기 작업을 줄일 수는 있다. 특히

테이블 크기가 큰 경우에는 디스크 읽기 작업을 상당히 줄이는 효과를 볼 수 있다. 왜냐하면, 실제 테이블 페이지 보다 이 실자료 지도의 크기가 훨씬 작기 때문이다.

7.1.5. 트랜잭션 ID 겹침 오류 방지

Agens SQL에서는 트랜잭션 자료에 대한 MVCC 기법은 트랜잭션 ID (XID)를 숫자로 처리하고 그것을 비교하는 방식이다. 한 로우의 자료 입력 XID 값이 현재 트랜잭션 XID 보다 더 크면 "앞으로 생길" (다른 트랙잭션에서 입력된) 자료이며 현재 트랜잭션에서는 보이지 말아야 할 자료임을 뜻한다. 트랜잭션 ID는 32bit 정수형 크기이고, 이 값은 하나의 클러스터 기준으로 관리되기 때문에 서버가 오랫 동안 운영 되었으면 (트랜잭션을 40억 개 넘게 사용했으면) *트랙잭션 ID 겹침* 오류를 발생할 수 있다. 트랙잭션 ID 계산기가 40억을 넘어 다시 0부터 시작하려고 하면 보관 되어 있는 모든 자료의 XID 값이 0보다 크기 때문에 모든 자료는 보이지 말아야할 자료로 처리할 것이다. 단순하게 말하면, 자료가 엄청나게 꼬여 버릴 것이다. (실제로는 유효한 자료임에도 불구하고, 그 자료를 볼 수 없는 황당한 사태가 발생할 것이다.) 이런 문제를 방지하기 위해서 모든 데이터베이스의 모든 테이블에 대해 20억 rodml 트랜잭션을 사용하기 전에 vacuum 작업이 필요하다.

주기적인 vacuum 작업이 이 문제를 해결 할 수 있는 이유는 Agens SQL은 `FrozenXID` 라는 특별 XID를 미리 예약 해 두었기 때문이다. 이 XID는 일반적인 XID 비교 대상에서 항상 제외되어 항상 보여지는(언제나 구 버전) XID이다. 일반 XID 비교 방법은 2^{32} 나머지 연산을 이용한다. 이 말은 20억 개의 "옛" XID와, 20억 개의 "새" XID 로 나누고, 이 XID 값은 계속 순환 하며 사용한다는 뜻이다. 그래서 한 XID로 저장 되었으면 그 이후 20억 개의 트랜잭션이 생기기 전까지는 그 자료는 "옛" XID로 처리되지만, 그 이상의 트랜잭션이 생기면, 그 현재 트랜잭션 기준으로 봤을 때 그 옛 XID는 앞으로 저장될 XID로 간주해 버린다. 이 문제를 피하는 방법은 20억 트랜잭션이 생기기 전에 그 옛 XID 자료의 XID 값을 `FrozenXID`로 바꾸는 것이다. 이렇게 "영구 보관용" 자료로 바꿔 놓으면 트랜잭션 XID 비교 작업에서 항상 제외 되기 때문에, XID 겹침 오류를 피해갈 수 있게 된다. 이 XID 변경 작업을 바로 **VACUUM** 명령으로 한다. (이 작업을 자료 프리징(data freezing)이라고 한다.)

vacuum-freeze-min-age 환경 변수는 `FrozenXID`로 바꾸기 전 얼마나 옛 XID를 남길 것인가를 지정한다. 이 값이 크면 그 만큼 트랜잭션 정보를 많이 보관할 것이고, 이 값을 줄이면 그 만큼 해당 테이블에 많은 트랜잭션을 vacuum 작업 없이 저장할 수 있게 된다.

표준 **VACUUM** 작업은 한 자료 페이지에 모든 자료가 UPDATE나 DELETE 명령 없이 오직 INSERT 명령으로 자료가 입력된 것만 있으면 그 페이지는 작업 대상에서 제외 한다. 그렇지만 그 자료 페이지에도 아직 `FrozenXID`로 바뀌지 않은 XID들이 있을 것이다. 이 XID들도 당연히 `FrozenXID`로 바뀌어야 XID 겹침 오류를 피해갈 수 있다. 이 작업을 위해서 vacuum-freeze-table-age 환경 변수 값을 조정 해서 **VACUUM** 작업에서 모든 자료 페이지를 대상으로 이 XID 변경 작업 할 것인지를 결정 할 수 있다. 해당 테이블의 나이가 `vacuum_freeze_table_age` - `vacuum_freeze_min_age` 값 보다 크면 **VACUUM** 작업은 모든 자료 페이지를 검사해서 변경 작업을 진행한다. `vacuum_freeze_table_age` 값을 0으로 지정하면 효율적인 작업을 위해 실자료 지도를 참조하지 않고 항상 모든 페이지를 검사한다.

한 테이블이 vacuum 작업 없이 계속 트랜잭션 작업을 할 수 있는 간격은 그 테이블의 마지막 vacuum 이후부터 20억 - `vacuum_freeze_min_age` 값만큼의 트랜잭션이다. 즉, 이 이상 트랜잭션이 발생했고 vacuum 작업이 없었으면 자료를 잃게 된다. 물론 현실적으로 이런 사태는 일어나지 않는다. autovacuum 기능을 사용하지 않아도 autovacuum-freeze-max-age 환경 설정 값으로 지정한 간격이 생기면 강제로 서버는 자체 vacuum 작업을 진행하기 때문이다.

한 테이블에 대해 vacuum 작업을 한 번도 하지 않았어도 `autovacuum_freeze_max_age` - `vacuum_freeze_min_age` 값 만큼의 트랜잭션이 발생하면, autovacuum 작업이 자동으로 진행된다. 자료 변경, 삭제 작업이 빈번한 테이블들의 경우 여유 공간 확보 작업과 동시에 진행되기 때문에 트랜잭션 ID 겹침 방지 작업은 별로 중요하지 않지만, 자료 추가만 계속 되는 테이블의 경우는 이 간격 만큼 주기적으로 vacuum 작업이 진행 된다. 그래서 이 반복 주기를 크

게 하려면 autovacuum_freeze_max_age 값을 크게 설정하거나 vacuum_freeze_min_age 값을 작게 설정한다.

vacuum_freeze_table_age 설정에 대한 최대값은 autovacuum_freeze_max_age * 0.95 이다. 이 보다 더 큰 값이 지정되어도 무시하고, 최대값이 사용된다. 95%로 지정한 이유는 이 값이 autovacuum_freeze_max_age 값보다 큰 경우는 어차피 autovacuum 작업 계획에 따라 무조건 vacuum 작업이 일어나기 때문에 의미가 없고 이 정도의 여지를 둔것은 그 사이 사용자가 직접 **VACUUM** 작업을 할 것인지를 판단할 수 있도록 하기 위함이다. 대략 **vacuum_freeze_table_age** 값은 autovacuum_freeze_max_age 값보다 작은 값으로 지정해서 그 차이값 만큼의 충분한 간격을 마련해 놓는 것이 좋다. 이렇게 해서 사용자가 직접 실행하는 주기적인 **VACUUM** 작업이나 자료 변경, 삭제 작업 때문에 자동으로 실행되는 autovacuum 작업들이 원활히 진행 되도록 한다. autovacuum_freeze_max_age - vacuum_freeze_table_age이 너무 작으면 최근에 여유 공간 확보 작업을 위해 vacuum 작업을 했음에도 불구하고 트랜잭션 ID 겹침 오류 방지 작업을 위해 또 vacuum 작업을 하는 빈도가 늘어날 것이다. 반대로 너무 크면 vacuum_freeze_table_age 작아서 잦은 테이블 자료 페이지 들을 전수 조사하는 빈도가 늘어날 것이다.

autovacuum_freeze_max_age 값(vacuum_freeze_table_age 값도 포함해서)을 크게 설정 할 때의 유일한 단점은 데이터베이스 클러스터 디렉토리의 하위 디렉토리인 pg_clog 디렉토리의 디스크 사용량이 커진다는 점이다. 왜냐하면 그 만큼의 트랜잭션 커밋 정보를 보관하고 있어야 하기 때문이다. 트랜잭션 커밋 정보는 2비트를 사용하므로 autovacuum_freeze_max_age 값을 최대치인 20억으로 지정 했으면, 그 디렉토리는 0.5GB의 공간이 필요하다. 이 정도의 크기가 데이터 클러스터 전체 크기에 비해서 별로 신경 쓸 크기가 아니면, autovacuum_freeze_max_age 값으로 최대값을 지정하는 것이 좋을 것이다. 그렇지 않은 경우, 이 값을 적당히 조절 해서 pg_clog 디렉토리의 저장 공간을 조절할 필요가 있다. (기본값인 2억 트랜잭션이라면 pg_clog 디렉토리는 최대 50MB 공간을 사용한다.)

vacuum_freeze_min_age 환경 설정 값을 줄이는 것의 단점은 빈번한 FrozenXID 변환 작업을 통해 전체적으로 vacuum 작업 시간이 많이 걸린다는 것이다. (굳이 필요 없는 작업을 더 하는 꼴이 된다.) 따라서 이 값은 각 자료가 더 이상 변경 되지 않아 영구 보관용으로 남기는 것이 좋겠다싶은 자료들을 대상으로 충분히 큰 값을 지정하는 것이 좋다. 이 값을 줄이는 것의 또 다른 단점은 해당 테이블의 많은 자료들이 FrozenXID로 변경 되어 버리면 데이터베이스 장애 시 원인분석을 위해 참조할 정보가 그 만큼 줄어든다는 것이다. 그래서 정적 테이블이 아니면 이 값을 되도록이면 줄이지 않을 것을 권장한다.

한 데이터베이스에서 가장 오래된 XID 값을 조사하는 방법은 pg_class와 pg_database 테이블을 살펴보는 것이다. **VACUUM** 작업 후 XID 정보를 이 두 테이블에 보관하고 있기 때문이다. pg_class 테이블의 *relfrozenxid* 칼럼값은 **VACUUM** 작업으로 그 테이블 전체에 대해서 XID 정리 작업을 했던 트랜잭션 ID 값이다. 이 값은 FrozenXID로 변경 하는 작업 시, 이 값보다 오래된 트랜잭션을 정리 대상으로 식별할 때 이용된다. 이와 비슷하게, pg_database 테이블의 *datfrozenxid* 칼럼은 해당 데이터베이스 전체를 대상으로 각 테이블의 *relfrozenxid* 값들 가운데 가장 오래된 값이다. 이 정보를 살펴볼 방법은 다음과 같은 쿼리를 실행해 보면 된다:

```
SELECT c.oid::regclass as table_name,
       greatest(age(c.relfrozenxid),age(t.relfrozenxid)) as age
FROM pg_class c
LEFT JOIN pg_class t ON c.reltoastrelid = t.oid
WHERE c.relkind IN ('r', 'm');

SELECT datname, age(datfrozenxid) FROM pg_database;
```

age 칼럼 값은 현재 시점 기준으로 마지막 vacuum 작업으로 정리된 가장 오래된 XID의 나이를 말한다.

표준 **VACUUM** 작업은 마지막 vacuum 작업이 있은 뒤부터 변경된 데이터 페이지들에 대해서만 작업 대상으로 삼는다. 하지만, 다음 세가지 경우에는 테이블의 모든 페이지를 조사한다. 첫번째, 이 *relfrozenxid* 값을 조사해서, 이 값의 나이가 vacuum_freeze_table_age 값보다 크다면 해당 테이블의 모든 페이지를 조사한다. 두번째, **VACUUM** 명령어에서 FREEZE 옵션을 사용할 때도 모든 페이지를 조사한다. 마지막으로 더 이상 사용되지 않는 자료를 정리하는 작업이 모든 페이지에 걸쳐 있어야 하는 경우에도 같은 작업을 한다. 보통은 이렇게 **VACUUM** 작업을 통해 모든 페이지가 조사 되었으면, 그 작업이 끝난 뒤 해당 테이블의 age(relfrozenxid) 값은 vacuum_freeze_min_age보다 약간 큰 값으로 보여진다. (**VACUUM** 작업이 시작되고, 확인 할 때까지의 트랜잭션 수 만큼 증가한다) 만일, **VACUUM** 작업을 계속 주기적으로 했으나 항상 변경된 페이지들만 조사하는 작업만 했으면 age(relfrozenxid) 값이 autovacuum_freeze_max_age 값에 이르렀을 때 autovacuum 데몬에 의해 강제로 테이블의 모든 페이지를 조사해서 트랜잭션 ID 겹침 오류를 방지한다.

만일 어떤 알 수 없는 오류로, autovacuum 작업이 제대로 진행 되지 못하는 경우를 대비 해서 그 데이터베이스의 가장 오래된 트랜잭션이 위험 수위에 다다르면 (통상 1천만 트랜잭션 정도 밖에 처리 할 수 없는 상황이면) 다음과 같은 서버 경고 메시지를 보여준다:

```
WARNING:  database "mydb" must be vacuumed within 177009986 transactions
HINT:  To avoid a database shutdown, execute a database-wide VACUUM in "mydb".
```

(서버 힌트처럼 이런 경우는 관리자가 직접 수동으로 **VACUUM** 작업을 해야한다. 이 작업은 데이터베이스의 *datfrozenxid* 값까지 변경해야 하므로 반드시 데이터베이스 관리자 권한으로 진행되어야 한다.) 이 경고를 무시하면 트랜잭션 ID 겹침 오류가 발생되기까지 1백만 트랜잭션 정도 남았을 때 데이터베이스 서버는 다음 오류 메시지를 남기고 이후 모든 트랜잭션 작업을 하지 않는다.

```
ERROR:  database is not accepting commands to avoid wraparound data loss in database "myd
HINT:  Stop the postmaster and use a standalone backend to VACUUM in "mydb".
```

1백만 트랜잭션을 남겨 놓은 이유는 관리자가 작업(수동 **VACUUM** 작업) 할 트랜잭션을 보장하기 위해서다. 이 메시지가 발생하고 있는 상황에서는 어떠한 작업도 할 수 없기 때문에 관리자는 서버를 중지하고 관리자 단독 모드로 서버를 실행해서 **VACUUM** 작업을 진행해야 한다. 관리자 단독 모드로 서버를 실행 하는 방법에 대해서는 Agens 설명서를 참조 하라.

7.1.5.1. 다중 트랜잭션과 겹침

다중 트랜잭션 ID 는 다중 트랜잭션 수행시 로우를 잠금 하는데 쓰인다. 잠금 정보를 저장할 수 있는 튜플 헤더 공간은 제한되어 있기 때문에 여러 트랜잭션이 한 로우를 잠금하면"다중 트랜잭션 ID"(또는 multixact ID)로 암호화 하여 저장된다. 다중 트랜잭션 ID에 포함된 트랜잭션 ID 정보는 pg_multixact 하위 디렉토리에 따로 저장되고, 다중 트랜잭션 ID만 튜플 헤더의 *xmax* 필드로 나타난다. 트랜잭션 ID처럼 다중 트랜잭션 ID도 32-bit 카운터와 스토리지로 실행되고 수명관리, 스토리지 청소, 겹침 처리가 필요하다.

VACUUM 테이블 스캔을 부분 혹은 전체 실행하면 vacuum multixact freeze min age 보다 큰 다중 트랜잭션 ID는 0 또는 트랜잭션 ID, 새로운 다중 트랜잭션 ID로 바뀐다. 각 테이블의 pg_class.*relminmxid* 는 전체 튜플에서 가장 오래된 다중 트랜잭션 ID를 저장한다. pg_class.relminmxid이 vacuum multixact freeze table age보다 크면 전체 테이블 스캔이 강제로 실행된다. 전체 **VACUUM** 스캔이 실행되면 무조건 해당 테이블의 vacuum_multixact_freeze_table_age가 바뀐다. 결국 전체 데이터베이스의 전체 테이블이 스캔되고 각 테이블의 pg_class.relminmxid는 바뀌고 오래된 다중 트랜잭션의 디스크 스토리지는 삭제된다.

multixact-age가 autovacuum-multixact-freeze-max-age 보다 큰 테이블에는 안전 장치 역할로 VACUUM 테이블을 전체 스캔 한다. AUTOVACUUM 기능이 꺼져 있어도 실행된다.

7.1.6. Autovacuum 데몬

Agens SQL에서는 추가적인 기능이기는 하지만 **VACUUM** 명령과 **ANALYZE** 명령을 주기적으로 자동 실행하는 *autovacuum*이라는 기능을 기본적으로 사용하길 권장한다. 이 기능을 켜두면 테이블의 자료가 많이 변경 되었을 때(추가, 변경, 삭제 작업이 많이 있었을 때) 자동으로 해당 테이블에 대해서 자동으로 윗 명령들을 실행한다. 이런 자동 실행이 가능 하려면 먼저 track counts 환경 설정 값이 `true`로 지정되어 데이터베이스 작업에 대한 통계 정보를 자동으로 수집해야 한다. 이런 설정은 모두 기본 설정이며, 이와 관련된 다른 설정들도 모두 적당하게 이미 설정 되어 있다.

"autovacuum 데몬"은 내부적으로 여러개의 프로세스로 구성된다. *autovacuum launcher*라는 이름의 프로세스가 항상 실행 되어 있으면서 그 프로세스가 *autovacuum worker*라는 이름의 하위 프로세스를 실행 해서 모든 데이터베이스에 대한 vacuum 작업을 하는 방식으로 운영된다. launcher 프로세스는 autovacuum-naptime 값으로 지정한 초 간격으로 한 번에 하나의 데이터베이스를 작업할 수 있도록 worker 프로세스의 실행 시간을 관리한다. (즉, *N*개의 데이터베이스가 있으면, `autovacuum_naptime`/*N* 초 간격으로 worker 프로세스는 자신이 작업 해야할 데이터베이스를 대상으로 작업을 시작한다.) 동시에 실행 될 수 있는 worker 프로세스의 최대 개수는 autovacuum-max-workers 환경 변수 설정값으로 지정할 수 있다. 만일 이 개수 보다 많은 데이터베이스가 있다면, 가장 먼저 실행한 worker 프로세스가 종료되는 즉시 남은 데이터베이스 가운데 하나를 대상으로 작업한다. worker가 하는 작업은 먼저 각 테이블의 상태를 조사해서, 필요하면 **VACUUM** 또는 **ANALYZE** 명령을 수행한다. `log_autovacuum_min_duration` 환경 변수 설정 값을 지정해서 autovacuum 상태를 지켜볼 수 있다.

만일 worker 프로세스가 각각 아주 큰 테이블의 vacuum 작업을 하게 되면 동시에 모든 worker 프로세스가 아주 오랜 시간 작업을 하게 될 것이다. 그런 동안에는 다른 테이블들에 대해서는 autovacuum 프로세스가 vacuum 작업을 할 수 없게 된다. 하나의 데이터베이스에 대해서 동시에 실행 될 수 있는 worker 프로세스의 수는 제한이 없다. 각 프로세스들은 이미 다른 프로세스가 작업 중인 테이블은 통과하고 다른 테이블을 조사해서 필요한 작업을 한다. 실행되는 worker 프로세스 개수는 max-connections 설정값에 포함되지 않으며, superuser-reserved-connections 설정값에도 포함되지 않음을 알고 있어야 한다.

테이블의 나이(*relfrozenxid* 칼럼 값을 age() 함수로 조사한 값)가 autovacuum-freeze-max-age 설정으로 지정한 트랜잭션 수 보다 많으면 그 테이블은 무조건 vacuum 작업을 한다. (또한 각 테이블 별로 저장 환경 설정 - 아래 참조 - 인 freeze 최대 나이값을 지정했으면, 그것을 참조해서 작업한다.) 또한, 테이블의 자료가 변경 되어 "vacuum 임계치"를 초과했으면 vacuum 작업을 진행 한다. 이 임계치 계산은 다음 식으로 산정한다.

```
vacuum 임계치 = vacuum 초기 임계치 + vacuum 배율값 * 로우수
```

vacuum 초기 임계치는 autovacuum-vacuum-threshold에서, vacuum 배율값은 autovacuum-vacuum-scale-factor에서, 로우수는 pg_class.*reltuples*에서 참조 한다. 더 이상 사용하지 않는 쓸모 없는 로우 정보는 통계 수집기가 수집한 정보를 사용한다. 이 정보는 **UPDATE** 명령이나 **DELETE** 명령에 의해서 갱신된 비교적 정확한 수치다. (비교적 정확하다고 한 이유는 서버 과부하시 이 값이 정확하게 수집되지 않을 수도 있기 때문이다.) 테이블의 *relfrozenxid* 나이값이 `vacuum_freeze_table_age` 설정값보다 크면, 테이블의 모든 로우를 조사해서 영구 보관용 XID로 바꾸고 *relfrozenxid* 값을 변경 한다. 그렇지 않은 경우는 마지막 vacuum 작업 뒤 변경된 자료 페이지만을 대상으로 작업한다.

analyze 작업도 위에서 설명한 방식과 비슷하게 진행 된다. 이 임계치 계산식은 다음과 같다:

```
analyze 임계치 = analyze 초기 임계치 + analyze 배율값 * 로우수
```

이렇게 계산된 임계치와 마지막 **ANALYZE** 작업이 있은 뒤 발생한 INSERT, UPDATE, DELETE 작업의 대상인 모든 로우수와 비교해서 작업을 진행한다.

임시 테이블은 autovacuum 대상에서 제외된다. vacuum 작업과, analyze 작업이 필요하면, 해당 세션에서 SQL 명령으로 사용자가 직접 작업 해야 한다.

임계치와 배율값은 기본적으로 `postgresql.conf` 파일에서 지정한다. 하지만 각 테이블 별로도 개별 지정이 가능하다. 이 부분에 대한 것은 Storage Parameters에서 자세히 다룬다. 이 값들이 각 테이블 별로 지정 되면, 그 테이블에 대해서는 서버 전역으로 설정된 값이 무시된다. autovacuum에 대한 환경 설정 매개변수들의 자세한 설명은 2.10절에서 다룬다.

위에서 설명 한 것 외에 테이블 단위 저장 옵션 환경 설정 매개변수는 여섯 개가 더 있다. 하나는 `autovacuum_enabled` 설정으로 이 값을 `false`로 지정하면, autovacuum 데몬은 트랜잭션 ID 겹침 방지에 대한 검사만 하고, 나머지 모든 작업을 하지 않는다. 다른 두 개는 `autovacuum_vacuum_cost_delay`, `autovacuum_vacuum_cost_limit` 인데, 테이블 단위 vacuum 작업 지연 기능에 관계된다. (2.4.4절 참조) 나머지 `autovacuum_freeze_min_age`, `autovacuum_freeze_max_age`, `autovacuum_freeze_table_age` 설정은 각각 vacuum-freeze-min-age autovacuum-freeze-max-age vacuum-freeze-table-age 설정과 같은 역할을 한다.

worker 프로세스가 여러 개인 경우, 위에서 지정한 비용 처리는 "균등배분" 방식이다. 즉, 실행 되는 프로세스가 많다고 해도 시스템 전체적인 입장에서 그 비용은 동일하다.

7.2. 정기적인 인덱스 다시 만들기

REINDEX 명령을 이용하거나, 이 작업과 상응하는 일련의 작업을 통해 주기적으로 인덱스를 다시 만드는 작업은 몇몇 상황에서는 가치 있는 작업이다.

완전히 빈 B-트리 인덱스 페이지는 재사용 하지만, 해당 페이지에 몇몇 자료를 남기고 대부분 자료가 지워지면 그 인덱스는 전체적으로 과도한 디스크 영역을 사용하게 된다. (이것을 인덱스 팽창(index bloating)이라고 한다.) 이런 경우 이 인덱스는 검색에서 디스크 읽기 비용이 최적화 된 인덱스보다 당연히 더 든다. (vacuum 작업은 한 페이지의 사용하지 않는 인덱스 로우들을 정리만 하지, 그 자료들의 재정렬 작업은 하지 않는다.) 이런 이유로 주기적인 인덱스 재사용 작업은 서버의 성능을 높일 수 있는 요소가 된다.

B-트리 인덱스가 아닌 경우는 팽창 정도가 얼마나 되는지 구체적으로 알 수 없다. 이 부분은 그 인덱스의 물리적 크기가 어느 정도 커지는지 주기적으로 직접 살펴봐야 한다.

또한 B-트리 인덱스에서는 새로 만든 경우가 여러번 변경 작업을 한 인덱스보다 순차적인 인덱스 검색 속도가 빠르다. 여러번의 변경 작업이 진행되다보면 해당 리프 페이지의 논리적인 순서와 물리적인 순서가 달라지기 때문이다. (이것을 pgstattuple 확장 모듈에서는 단편화라는 용어를 사용해서 그 빈도를 보여준다. B-트리 인덱스가 아닌 인덱스에 대해서는 이 현상이 발생하지 않는다.) 그럼에도 불구하고 주기적으로 인덱스를 새롭게 만드는 방법이 검색 속도를 높이는데 도움을 준다.

인덱스를 다시 새롭게 만드는 가장 손쉬운 방법은 REINDEX 명령을 이용하는 것이다. 하지만 이 작업은 해당 테이블에 대해서 배타적 잠금을 지정하기 때문에 새로운 인덱스를 만들고, 옛 인덱스로 바꾸는 여러 작업으로 나누어 진행하는 것이 일반적이다. CREATE INDEX 명령을 사용할 때, 해당 인덱스가 `CONCURRENTLY` 옵션을 사용 할 수 있으면 이 옵션을 이용하는 것도 한 방법이다. 새 인덱스가 정상적으로 만들어졌으면, 옛 인덱스를 DROP INDEX 명령으로 지우고, 새 인덱스의 이름을 ALTER INDEX 명령으로 바꾸면 된다. 한편, 인덱스가 유니크나 기본키 제약조건에서 사용되면 새로 만들어진 새 인덱스를 ALTER TABLE 명령을 통해서 제약조건 변경 작업을 하고, 옛 인덱스를 지우는 식으로 진행한다. 여기서 소개한 인덱스 다시 만들기 작업은 반드시 각 작업이 정상적으로 진행되지 못했을 경우 복구할 수 있는 방안에 대해서 꼼꼼히 준비해서 작업해야할 것이다.

7.3. 로그 파일 관리

데이터베이스 서버 로그를 `/dev/null` 쪽으로 보내서 그냥 버리기 보다 그것을 어떤 장소에 보관하는 것이 좋다. 왜냐하면 장애 원인 분석을 할 때 서버 로그는 중요한 단서가 되기 때문이다. 하지만 원인 분석을 위해서 서버 로그를 보다 자세히 남기도록 설정을 했거나 서버가 바빠서 로그가 많이 쌓이는 경우, 그 로그 파일이 분석하기 힘들 정도로 크기가 커지기도 한다. 이러 이유로 서버 로그 파일에 대한 주기적인 관리 작업이 필요하다. 대표적인 작업이 로그를 크기나 날짜 단위로 쪼개서 로그 파일의 크기를 분석하기 편하도록 하는 일과 아주 오래된 로그 파일들을 지워 로그 파일이 쌓이는 디렉토리의 여유 공간을 일정하게 유지 하도록 하는 일 등이다.

서버 로그를 파일로 저장하는 가장 간단한 방법은 **agens**의 stderr 출력을 파일로 저장하는 것이다. 하지만, 이렇게 로그 파일을 만들면 로그 파일을 바꾸려고 할 경우 서버를 중지 하거나 재실행 해야한다. 이런 운영 방식은 개발 환경에서는 수용할 수 있지만, 운영 환경 에서는 수용하기 힘들다.

stderr 출력을 파일로 저장 할 때는 그 내용을 특정 로그 파일로 쉽게 바꾸어 저장할 수 있는 프로그램을 사용한다. 이 방법 말고 서버 내부적으로 이런 로그 파일을 바꾸는 기능을 이용하는 방법도 있다. 이 기능을 사용하려면, `postgresql.conf`에서 `logging_collector` 환경 설정값을 `true`로 지정하고, 서버를 재실행 한다. 이 기능의 작동 방식을 제어하는 환경 설정 매개 변수들이 몇가지 있는데, 이들에 대한 자세한 설명은 2.8.1 절에서 한다. 또한, 서버 로그를 컴퓨터가 쉽게 분석할 수 있도록, CSV(반점 구분 양식, comma-separated values) 형태로 로그를 저장 할 수도 있다.

앞에서 잠깐 언급한 stderr 출력을 그대로 사용해서 로그 파일를 쉽게 바꾸는 방법 가운데 하나는 Apache 배포판에서 제공하는 rotatelogs 프로그램을 파이프를 통해 사용하는 것이다. **pg_ctl** 명령으로 서버를 기동하면, 이 프로그램은 모든 stderr 출력을 stdout 쪽으로 보내기 때문에 단순하게 다음과 같은 명령으로 사용할 수 있다.

```
pg_ctl start | rotatelogs /var/log/pgsql_log 86400
```

로그 파일을 다루는 또 다른 한가지 방법은 그 출력을 syslog 쪽으로 보내는 것이다. 이렇게 하면 syslog 프로그램이 로그 파일 관리에 대한 부분을 모두 담당한다. 이렇게 설정하려면 `postgresql.conf` 파일에서 `log_destination` 환경 설정 값을 `syslog`로 지정한다. syslog 프로세스에 `SIGHUP` 시그널을 보내서 로그 파일을 바꿀 수 있을 것이다. 물론 syslog 프로세스가 다루는 여느 로그 파일과 같이 logrotate 을 이용해서 그 파일을 특정 기준으로 바꿀 수도 있다.

하지만, 많은 시스템에서 이 방법은 잘 사용되지 않는다. syslog 특성상 기록해야할 내용이 너무 많으면 그 내용을 자르거나 시스템이 너무 바쁘면 로그 기록을 간혹 안하기 때문이다. 또한 Linux 환경에서는 기본적으로 로그 내용 하나를 파일에 기록할 때마다 OS의 디스크 I/O 버퍼에 있는 내용을 디스크와 동기화 하기 때문에 성능 저하를 초래한다. (syslog 환경 설정 파일에 지정하는 로그 파일 이름 앞에 "-" 문자를 더 적어주면 이런 디스크 동기화 작업을 생략할 있다.)

로그 파일을 관리하는 작업 가운데 또 하나 고려 해야할 사항은 아주 오래된 로그 파일 처리에 대한 정책을 세우는 일이다. 일반적으로 일정 기간이 지난 로그 파일은 주기적으로 삭제하는 정책을 만든다. 그리고 그 작업을 자동화 하는 방식을 택한다. 또 다른 방법은 로그 파일 이름을 순환 하는 형태로 정하고 해당 로그를 해당 파일에 기록하는 방식을 사용해서 로그 파일 개수가 일정하게 유지 되도록 한다.

아울러 이 로그 파일을 사용하는 몇몇 도구를 소개하면 pgBadger (http://dalibo.github.io/pgbadger/) 라는 서버가 남긴 로그 파일을 정교하게 분석해서 보여주는 외부 프로젝트가 있다. check_postgres (http://bucardo.org/wiki/Check_postgres) 스크립트는 Nagios 경고 모듈로 사용할 수 있다. 이것으로 로그 파일에 기록 된 내용들 가운데 경고가 필요한 내용을 정교하게 지정해서 찾을 수도 있다.

8장. 백업과 복원

모든 데이터베이스 시스템이 그렇듯이, Agens SQL 데이터베이스 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.

이 장에서는 Agens SQL 자료를 백업하는 세 가지 서로 다른 방법을 소개하고 있다.

- SQL 덤프
- 파일 시스템 기반 백업
- 아카이브 모드 백업

위 방법들은 서로 장단점이 있다. 이것들에 대한 자세한 설명은 다음 각 절에서 설명하고 있다.

8.1. SQL 덤프

이 방법은 백업 프로그램이 백업 대상에 대해서 SQL 구문의 파일을 만들고, 복원을 할 때는 서버에 이 SQL 구문을 실행해서 백업 할 때의 상태로 만드는 기법이다. 이 방법을 구현한 Agens SQL 유틸리티 프로그램이 pg dump 이다. 기본적인 사용법은 다음과 같다:

```
pg_dump 데이터베이스이름 > 출력파일
```

위와 같이, pg_dump 명령의 결과는 표준 출력(stdout)으로 출력된다. 다음 글에서 이 프로그램이 얼마나 유용하게 쓰이는 지 알 수 있을 것이다. 위 명령은 텍스트 파일을 만들지만 pg_dump 명령에 부가 옵션들을 사용하면, 복원 작업 시 보다 섬세한 처리나 병렬 처리를 할 수 있도록 텍스트가 아닌 다른 양식으로도 만들 수 있다.

pg_dump 프로그램은 여느 Agens SQL 클라이언트 응용 프로그램과 같다(특히 잘 만들어진). 이 말은 원격 데이터베이스 서버에 대해서 그 접속 권한이 있으면, 그 접속 가능한 다른 호스트에서 이 명령을 실행 할 수 있음을 의미한다. 다만, 덤프 작업을 하기 때문에 필요에 따라서 스키마의 접근 권한이 있어야 하고 테이블의 읽기 권한이 있어야 하는 등 그 적절한 접근 권한이 있어야한다. 대부분 실무에서는 이 작업은 데이터베이스 슈퍼유저로 실행된다. (물론 해당 데이터베이스 전체를 백업할 수 없는 일반 유저 권한이라도 `-n schema` 또는 `-t table` 옵션을 사용해서 자신이 접근 할 수 있는 객체만 백업할 수 있다.)

pg_dump 프로그램을 실행해서 원격 데이터베이스 서버를 지정하기 위해서는 `-h 호스트이름`, `-p 포트` 옵션을 사용한다. `-h` 옵션을 사용하지 않으면 먼저 PGHOST 시스템 환경 변수에서 지정된 호스트로 접속을 시도하며, 이 값이 지정되어 있지 않으면 로컬 호스트로 접속한다. `-p` 옵션을 사용하지 않으면 먼저 PGPORT 시스템 환경 변수에서 지정된 포트로 접속을 시도하며, 이 값이 지정되어 있지 않으면 이 프로그램이 컴파일 될 때 지정한 포트로 접속한다.

다른 Agens SQL 클라이언트 응용 프로그램과 마찬가지로 pg_dump 프로그램도 데이터베이스 사용자 계정을 특별히 지정하지 않으면 현재의 운영체제 시스템 사용자 계정의 이름을 사용한다. 데이터베이스 사용자 계정을 지정하려면 `-U` 옵션을 사용하며 PGUSER 시스템 환경 변수로도 이 데이터베이스 사용자 계정을 지정할 수 있으며, 이 값의 사용 우선 순위도 위의 다른 시스템 환경 변수들과 같다. 위에서 언급한 대로 pg_dump 프로그램의 데이터베이스 접속은 클라이언트 인증 메카니즘을 사용한다. 이에 대한 보다 자세항 사항은 3장에서 다룬다.

pg_dump 프로그램을 이용한 데이터베이스 백업 방법은 파일 시스템 기반 백업과 아카이브 모드 백업 방법에 비교해서 다음과 같은 잇점이 있다. 먼저 복원 서버의 버전이 백업 서버의

버전보다 더 최신인 것도 복원이 가능하다. 32-bit, 64-bit와 같은 머신 아키텍처가 서로 달라도 가능하다.

pg_dump 프로그램이 실행되면, 그 실행되는 시점의 스냅샷 상태로 작업을 진행한다. pg_dump 프로그램이 실행되는 동안 다른 세션에서 **ALTER TABLE** 명령과 같이 exclusive lock이 발생하지 않는 상태이면 대부분의 작업은 특별한 잠김 없이 작업을 진행한다.

8.1.1. 덤프 파일 복원하기

pg_dump 프로그램으로 만든 텍스트 덤프 파일은 psql 프로그램에서 바로 사용할 수 있다. 일반적인 사용법은 다음과 같다.

```
psql 데이터베이스이름 < 입력파일
```

입력파일 자리에 pg_dump 명령으로 만든 덤프 파일을 지정한다. *dbname* 자리에는 psql 클라이언트가 접속 해서 덤프 파일의 명령을 실행할 데이터베이스 이름을 지정한다. 주의할 점은 위 명령으로 해당 데이터베이스가 만들어지지는 않는다는 점이다. psql 명령을 실행하기 전에 먼저 `createdb -T template0 데이터베이스이름` 명령과 같이 먼저 접속할 데이터베이스를 만들어 놓아야한다. psql 프로그램도 pg_dump 프로그램과 같은 데이터베이스 접속과 관련된 접속 인증 방식으로 환경변수와 커맨드라인 옵션을 사용하면 된다. psql 프로그램에 대한 자세한 설명은 psql 설명서에서 다룬다. 일반 텍스트 덤프 파일이 아닌 백업 파일을 복원할 경우는 pg restore 명령을 이용한다.

SQL 덤프 파일 안에는 각 객체의 접근 권한을 지정하는 명령도 포함되어 있기 때문에 그 덤프 파일 안에서 사용하는 모든 사용자들도 새 데이터베이스에 미리 만들어야한다. 해당 사용자가 없으면 이 접근 권한을 설정하는 명령어들은 모두 실패할 것이다. (가끔 이것을 의도해서 이렇게 처리하기도 하지만 일반적인 방법은 아니다.)

기본적으로 psql 스크립트는 해당 작업 도중 오류가 발생하면 멈추지 않고 다음 작업을 진행한다. 만일 이런 기능을 원치 않으면 psql 프로그램을 실행 할 때, `ON_ERROR_STOP` 변수를 지정한다. 이렇게 하면, 스크립트 실행 도중 오류가 발생했을 때 바로 스크립트는 중지되고 psql 실행 리턴 코드 3으로 종료된다. 다음은 그 사용 방법이다.

```
psql --set ON_ERROR_STOP=on dbname < infile
```

이 방법은 스크립트에서 정상적으로 실행된 작업들에 대해서는 해당 데이터베이스에 반영되었다는 것을 의미한다. 트랜잭션 처리 방식 처럼 모든 스크립트가 정상 처리되어야만 전체가 반영되게 하려면 `-1` 또는 `--single-transaction` 옵션을 사용해서 스크립트 전체를 하나의 트랜잭션으로 처리하도록 한다. 단점은 많은 작업을 해야하는데 작은 오류 하나로 전체가 롤백 되면서 많은 시간을 소비할 수 있다는 점이다. 대신에 오류가 발생하면 수동으로 일일이 복잡한 데이터베이스 정리 작업을 하지 않아도 된다.

pg_dump, psql 두 프로그램 모두 표준 입출력을 사용할 수 있기 때문에 파이프를 사용하면 덤프 하면서 바로 다른 데이터베이스 서버에 복원할 수도 있다. 사용법은 다음과 같다.

```
pg_dump -h 호스트1 데이터베이스이름 | psql -h 호스트2 데이터베이스이름
```

> 중요: pg_dump 명령으로 만들어지는 결과물은 `template0` 데이터베이스 기반으로 변경된 모든 작업이 포함된다. 그래서 프로시져 언어, 확장 모듈과 관련된 함수들이 모두 포함되어있다. 만일 사용의 편의를 위해서 이런 기반 작업들을 `template1` 데이터베이스에 해 두면 일반적인 방법으로 데이터베이스를 만들 때 `template1` 데이터베이스를 참조해서 만들기 때문에 복원 작업에서 충돌이 발생할 수 있다. 이것을 방지하기 위해서는 위에서 언급한 대로 `template0` 데이터베이스를 참조해서 새 데이터베이스를 만들어서 사용하는 것이 바람직하다.

백업 파일의 복원 작업이 끝난 후에는 쿼리 최적화기가 올바른 통계 정보를 사용할 수 있도록 ANALYZE 명령을 실행 한다.(7.1.3절과 7.1.6절 참조.) 대용량 데이터베이스 작업에서, 보다 나은 성능을 고려하면 7.1.3절을 참조해서 덤프, 복원 작업을 하기 전에 먼저 서버 환경을 변경하고 작업을 진행하는 것도 좋은 방법이다.

8.1.2. pg_dumpall 명령어 이용하기

pg_dump 명령어는 지정한 하나의 데이터베이스만 한 번에 백업한다. 또한 데이터베이스 객체에 속하지 않는 role(데이터베이스 사용자), 테이블스페이스 정보는 백업되지 않는다. 데이터베이스 클러스터 기준 모든 정보를 백업받으려면, pg dumpall 프로그램을 이용한다. 일반적인 사용법은 다음과 같다.

```
pg_dumpall > outfile
```

이 덤프 파일은 psql 프로그램에서 다음과 같이 사용한다:

```
psql -f 덤프파일 postgres
```

(정확하게는, 데이터베이스 서버에 접속하기 위해서 `postgres` 데이터베이스 이름을 지정한 것 뿐이지, 실제로는 접속이 끝나면 바로 이 데이터베이스에서 작업하기 전에 먼저 데이터베이스 클러스터 전역 정보부터 복원 작업을 시작한다.) pg_dumpall 명령을 사용할 때는 접속할 해당 데이터베이스의 슈퍼유저로 접속해야 한다. 그래야 role, 테이블스페이스 정보들을 덤프할 수 있다. 또한 이 덤프된 자료를 복원할 때도 마찬가지다. 주의할 점은 만일 기본 테이블스페이스 외에 다른 테이블스페이스를 사용하면 그 테이블스페이스의 실제 디렉토리도 덤프하는 서버에서 사용하는 경로와 똑같이 만들어 놓아야한다.

pg_dumpall 명령은 먼저 데이터베이스 사용자를 위한 role 정보, 다음 테이블스페이스 정보, 다음 기본 템플릿 데이터베이스에 대한 덤프 작업을 한 다음 각 데이터베이스를 덤프하기 위해 pg_dump 프로그램을 호출 한다. 이 말은 각 데이터베이스가 각기 다른 시간에 백업 작업을 진행 하기 때문에 데이터베이스간 똑같은 시간대의 스냅샷 형태의 자료 덤프는 불가능하다는 의미다.

데이터베이스 클러스터의 서버 전역 객체들만 백업하려면 pg_dumpall `--globals-only` 옵션을 이용한다. 이 경우는 pg_dump 명령을 이용해서 각각의 데이터베이스를 따로 백업해서 복원하려고 할 때 필요하다.

8.1.3. 대용량 데이터베이스 다루기

몇 운영체제는 만들 수 있는 최대 파일 크기가 제한 되어 있어, pg_dump 명령으로 만들어지는 파일이 이 최대 파일 크기를 초과하는 경우 문제가 될 수 있다. 다행이, pg_dump 명령의 출력 내용을 표준 출력(stdout)으로 보낼 수 있기 때문에 표준 유닉스 명령들을 이용하면 이런 잠재적인 문제를 피해갈 수 있다. 그 방법은 다음과 같이 여러가지다.

덤프 내용을 압축하라. gzip과 같은 압축 프로그램을 사용하는 방법:

```
pg_dump dbname | gzip > filename.gz
```

복원 할 때는,

```
gunzip -c filename.gz | psql dbname
```

또는,

```
cat filename.gz | gunzip | psql dbname
```

split 명령을 이용하라**. split** 명령은 표준 입력(stdin)으로 들어오는 내용을 지정한 크기로 나누어서 파일로 저장하는 기능을 제공한다. 다음은 1MB 크기로 나눠서 덤프 파일을 생성하는 예제다:

```
pg_dump dbname | split -b 1m - filename
```

복원 할 때는,

```
cat filename* | psql dbname
```

pg_dump의 사용자 정의 포멧 기능을 이용하라**.** 만일 Agens SQL 배포판을 빌드할 때, zlib 압축 라이브러리를 사용할 수 있도록 했으면 덤프 파일을 만들 때 그 내용을 압축해서 만든다. 이 때 최종 덤프 파일은 **gzip** 프로그램으로 원본 덤프 파일을 압축 했을 때의 크기만큼 줄일 수 있다. 또한 이 기능을 이용해서 백업을 하면 테이블을 선택적으로 지정해서 그 테이블만 복원할 수도 있다. 다음은 사용자 정의 포멧을 지정해서 덤프 하는 예제다.

```
pg_dump -Fc dbname > filename
```

이렇게 만들어진 덤프 파일은 psql에서 사용할 수 있는 스크립트 파일이 아니다. 그러므로 다음과 같이 pg_restore 명령을 사용해서 복원 해야한다.

```
pg_restore -d dbname filename
```

자세한 사용법은 pg dump, pg restore 사용설명서를 참조하라.

엄청나게 큰 데이터베이스면 **split** 명령어와 함께 기타 다른 명령어들을 함께 연결해서 사용할 필요성도 있다.

pg_dump의 병렬적 덤프 기능을 이용하라**..** 대용량 데이터베이스의 덤프 속도를 높이기 위해서는 pg_dump의 병렬 모드를 사용하면 된다. 그러면 테이블들을 동시에 덤프한다. 병렬 처리 수준은 **-j** 매개변수로 조절할 수 있다. 병렬적 덤프는 "디렉토리" 아카이브 형식에만 지원된다.

```
pg_dump -j num -F d -f out.dir dbname
```

pg_restore -j를 이용해서 병렬적 덤프를 복구할 수 있다. 이 방식은 "사용자 정의" 혹은 "디렉터리" 아카이브 모드의 아카이브에서 가능하며 **pg_dump -j**로 만들어지지 않았어도 상관없다.

8.2. 파일 시스템 기반 백업

백업 하는 다른 방법으로 Agens SQL에서 사용하는 데이터 저장 공간 전체를 직접 파일 복사하는 식으로 파일 시스템 차원에서 처리하는 방식이 있다. 해당 파일의 위치는 1.2절에서 자세히 설명하고 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 시스템의 단위적(atomic) 스냅샷을 지원하지 않으며, 서버에서 사용하는 내부적인 버퍼링에 대한 정보를 정확하게 반영할 수 없기 때문이다.) 서버를 중지하는 방법은 1.5절에서 다룬다. 또한 마찬가지로 복원할 때는 반드시 서버는 중지된 상태여야한다.
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 "완전한 스냅샷(consistent snapshot)" 기능을 제공한다면, (그리고, 이 기능이 신뢰성이 있다면) 이 기능을 사용할 때 주의 해야한다는 점이다. 이 파일 시스템 스냅샷 기능을 이용한 전형적인 백업 방법은 먼저 "현 시점 스냅샷 frozen snapshot"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 무시할 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 "현 시점 스냅샷 frozen snapshot"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. 또 다른 방법으로 아카이브 모드 백업(8.3.2절)과 그 복구(8.3.4절) 방식을 이용해야한다. 자세한 이야기는 다음 절과, 앞 사용 설명서를 참조하라.

다른 한 방법은 rsync 응용 프로그램을 이용하는 방법이다. 작업은 두 단계로 진행 된다. 먼저 서버가 실행 중 일 때, 자료를 모두 동기화 하고, 그 다음에 서버를 중지하고 한 번 더 동기화를 한다. 이렇게 하면, 서버가 중지 되면서 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.

8.3. 아카이브 모드 백업(Continuous Archiving)과, 특정시점 복구(Point-in-Time Recovery, PITR)

Agens SQL에서는 미리 쓰기 기록 (선행 기입 로그, *write ahead log*, WAL)을 데이터베이스 클러스터 디렉토리 안 `pg_xlog/` 디렉토리에서 관리하고 있다. 이 로그는 데이터베이스의 데이

터 파일에 대한 모든 조작 기록을 보관하고 있다. 이 로그를 만드는 가장 첫번째 이유는 서버가 갑자기 중지되었을 경우 미처 데이터 파일에 적용하지 못한 작업(checkpoint 작업이 안된)을 이 로그에서 읽어서 그대로 "다시 실행해서" 서버를 안전하게 복구하기 위해서이다. 이 방법을 그대로 응용하면 이 로그를 준비된 다른 서버로 보내서 이 로그의 내용을 그대로 재실행해서 원본 서버와 똑같은 상태를 만들 수 있다. (물론 특정 시점까지만 실행하면 데이터베이스를 특정시점으로 되돌릴 수도 있다.) 여기서 소개하고 있는 백업과 복원 방법은 지금까지 이야기한 것보다 훨씬 복잡하다. 하지만 다음과 같은 장점이 있다.

- 파일 시스템을 백업할 때, 파일 시스템 상태에 대해서 신경 쓸 필요가 없다. 복원할 때 WAL 파일의 내용을 재실행 하면서 마치 데이터베이스가 갑자기 중지되었을 때 복구하는 기법과 같이 데이터베이스를 안전하게 복원한다. 즉, 파일 시스템의 스냅샷 기능을 고려할 필요 없이 tar와 같은 간단한 프로그램으로 백업 작업을 할 수 있다.
- 복원 작업에 필요한 WAL 파일의 수량에 제한이 없기 때문에 백업을 시작하는 시점 뒤부터 생길 WAL 로그만 정확하게 보관하면 백업 기간이 아무리 길어도 상관 없다. 이 점은 자주 전체 백업을 할 수 없는 대용량 데이터베이스 백업에서 유용하다.
- 복원을 할 때 WAL 파일의 내용을 끝까지 재실행할 필요가 없다. 이 말은 특정 시점에서 이 작업을 멈출 수 있다는 것이다. 이렇게 하면 특정 시점의 데이터베이스 상태로 복원이 가능하다. 이것을 특정시점 복구 *point-in-time recovery* 기능이라고 한다.
- 다른 호스트에 베이스 백업을 복원해 놓고 운영 서버에서 만들어지는 WAL 파일들을 주기적으로 서버에 반영 하면, 운영 서버 장애가 발생했을 때 빠르게 대응할 수 있다. 이렇게 운영하는 방법을 *warm standby* 시스템 구축 기법이라고 한다.

> 참고: pg_dump 명령과 pg_dumpall 명령을 이용해서 만든 백업 파일은 데이터베이스의 논리적 백업이기 때문에 실제 데이터베이스에서 일어난 모든 기록을 보관하고 있지 않다. 그래서 파일 시스템 기반 백업이나 아카이브 모드 백업에서는 사용할 수 없다.

파일 시스템 기반 백업에서 이야기한 것과 같이 이 백업/복원 작업도 데이터베이스의 특정 부분을 대상으로 할 수 없고 데이터베이스 클러스터 전체를 대상으로 한다. 또한 아카이브 로그를 다루는 여유 공간도 상당히 필요하다. 베이스 백업량도 많고 업무량도 많아서 WAL 파일을 많이 만들면 베이스 백업과 WAL 파일을 처리하기 위한 비용도 상당히 많이 든다. 하지만 고가용성 데이터베이스 시스템으로 운영해야하는 환경에서는 여전히 이 방법이 가장 추천할 만한 방법이다.

이 아카이브 모드 백업(다른 데이터베이스에서는 흔히 "온라인 백업"이라고 함)을 이용한 복원 작업을 무사히 마치려면, 적어도 베이스 백업을 시작하기 직전의 WAL 파일부터 베이스 백업이 끝나는 시점까지 새로 생긴 WAL 파일들이 필요하다. 그래서 이 백업 방식을 사용하려면 먼저 아카이브 로그 백업에 대한 정책을 수립하고, 베이스 백업을 진행해야 한다.

8.3.1. WAL의 아카이브 파일 만들기

내부적으로 Agens SQL 시스템은 데이터베이스를 조작하는데 끊임 없이 순차적으로 WAL 레코드를 만든다. 이것을 물리적인 디스크 공간에 저장하기 위해서 WAL *세그먼트 파일*로 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 이

미 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 Agens SQL 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, wal level 환경설정 매개변수의 값으로 `archive` (또는 `hot_standby`)를 지정한다. 그 다음 archive mode 환경설정 매개변수 값에는 `on`, archive command 환경설정 매개변수 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.

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

위 설정은 단순하게 WAL 세그먼트 파일을 `/mnt/server/archivedir` 디렉토리로 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/00000001000000A900000065 && cp pg_xlog/00000001000000A9
```

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

이 아카이브 명령은 Agens SQL 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. Agens SQL 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. 이 환경설정 파일 백업을 위해서 해당 파일의 위치를 바꾸려면, 2.2절 항목을 참조하라.

아카이브 명령은 WAL 파일 가운데 서버에 모두 반영된(rollback되거나 commit 되어 checkpoint 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 환경설정 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 환경설정값을 minimal로 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.

8.3.2. 베이스 백업 만들기

베이스 백업을 만드는 가장 손쉬운 방법은 pg basebackup 툴을 사용하는 것이다. 이 툴을 이용하면 일반적인 파일이나 tar 묶음 파일로 베이스 백업 파일을 만들 수 있다. pg basebackup 툴을 이용하는 방법보다 유연하게 백업을 하고 싶으면, 저수준 API를 이용해서 사용자 정의 백업 방식을 구현할 수 있다. (8.3.3절 참조)

베이스 백업을 만드는 데 걸리는 시간은 우려할 필요가 없다. 하지만 운영 서버가 full_page_writes 비활성화 상태로 운영 되고 있으면, 베이스 백업을 하는 동안 자동으로 full_page_writes 설정이 활성화 되어 서버의 성능이 약간 떨어질 것이다.

이 베이스 백업을 이용해서 복구 작업을 하면, 베이스 백업 작업에서 파일 시스템 복사를 시작하는 시점부터 복사가 끝나는 시점까지 만든 모든 WAL 파일을 따로 보관하고 있어야 한다. 어떤 파일부터 따로 보관해야 하는지는 베이스 백업을 시작하는 시점부터 백업이 끝나는 시점까지 생성된 WAL 파일을 *백업 내역 파일*에 기록해 둔다. 이 파일의 이름은 복구에 필요한 WAL 파일의 첫번째 파일 이름을 사용한다. 예를 들어서, 필요한 파일이 `000000010000123400000055CD` 이라면, 백업 내역 파일은 `000000010000123400000055CD.007C9330.backup` 파일이다. 이 파일 또한 pg_xlog 디렉토리 안에 있으면, 베이스 백업이 끝나 이 파일이 만들어졌을 때 아카이빙 작업에 의해 WAL 세그먼트 파일과 함께 따로 보관하는 장소에 복사된다. (WAL 세그먼트 파일 이름 뒤에 있는 정보는 파일의 정확한 시작 위치인데 이 부분은 무시해도 좋다.) 베이스 백업이 끝나면 백업 내역 파일의 이름을 보고 알파벳 순으로 그 이름보다 이전이면 복구를 하는데 사용되지 않는다. 그러므로 다중 백업 환경이 아니면 오래된 WAL 세그먼트 파일들은 파일 시스템에서 삭제되어도 무방하다. 다중 백업 환경이면 각 백업 복구에 필요한 모든 WAL 세그먼트 파일들을 잘 정리해 두어야한다.

백업 내역 파일은 크기가 작은 단순 텍스트 파일이다. 이 파일에는 pg basebackup 툴에서 자동으로 지정한 백업 라벨과 백업 복구에 필요한 WAL 세그먼트 파일 이름 정보와 백업 시간이 기록되어있다. 백업본을 구분하기 위해 라벨을 사용할 때, 이 백업 내역 파일을 이용한다.

베이스 백업 주기는 전체 백업량과 그 백업 간격 사이 만들어지는 WAL 세그먼트 파일의 양, 백업 파일을 보관하는 속도, 여유 공간, 복구할 때 소요되는 시간까지 여러 부분을 고려해야 한다. — WAL 세그먼트 파일이 많으면 베이스 백업을 준비해도 그 파일을 모두 반영하는데 시간이 필요하기 때문이다.

8.3.3. 저수준 API를 이용한 베이스 백업 만들기

저수준 API를 이용해서 베이스 백업을 만드는 방법은 pg basebackup 툴을 사용할 때보다 몇 단계의 작업이 추가 되기는 하지만 상대적으로 간단하다. 이렇게 작업할 때 가장 주의해야 할 점은 다음 설명할 각 단계가 반드시 정상적으로 수행된 경우에만 다음 단계 작업이 되야한다는 점이다.

1. 서버가 아카이브 모드로 설정되어 있고 WAL 세그먼트 파일들을 따로 보관하는 작업이 실행되는 상태여야 한다.
2. 해당 데이터베이스에 슈퍼유저로 접속해서 다음 명령을 실행한다:

   ```
   SELECT pg_start_backup('라벨');
   ```

   라벨 자리에는 어떤 문자열이 와도 상관 없다. 관리자가 해당 백업 작업에 대한 식별자로 사용되는 문자열이면 된다. (백업본을 저장하려고 하는 디렉토리의 전체 경로를 이 이름으로 정하는 것도 좋은 방법이다.) `pg_start_backup` 함수는 먼저 데이터 클러스터 디렉토리에 `backup_label` 이라는 파일을 만들고, 그 안에 지정한 라벨 문자열과 백업을 시작하는 시간을 기록한다. 이 파일은 복구할 때 꼭 필요한 파일이기 때문에 백업 결과물에 이 파일이 없거나 손상되었으면 백업 작업을 처음부터 다시 해야한다.

   접속한 데이터베이스의 클러스터 디렉토리의 위치는 몰라도 된다. `pg_start_backup` 함수의 실행 결과가 오류를 발생시키면 그에 적합한 조치를 취해야겠지만 오류를 내지 않으면 이 작업의 내부 작업에 대해서는 신경쓰지 않아도 된다.

   기본적으로, `pg_start_backup` 함수 실행을 끝내는데, 꽤 긴 시간이 소요되기도 한다. 백업 시작 전에 이 함수로 체크포인트 작업을 하는데, 체크포인트에 필요한 I/O가 기본적으로 내부 체크포인트 간격만큼 지속된다. (환경 설정 매개변수 checkpoint_completion_target를 참조) 이는 쿼리 처리 작업에 끼치는 영향을 최소로 줄여주기 때문에 유용하다. 만일 이렇게 긴 기다림의 시간 없이 바로 백업 준비를 위한 작업을 곧바로 하려면 다음과 같은 명령을 사용한다.

```
SELECT pg_start_backup('label', true);
```
위 명령은 최대한 빨리 체크포인트 작업을 진행시킨다.

3. 데이터 클러스터 디렉토리를 시스템 차원에서 백업한다. 이 백업 방법은 파일 시스템 백업하는 어떤 방법으로든지 가능하다. 예를 들면, (pg_dump, pg_dumpall 명령어를 사용하는 것이 아니라) tar, cpio같은 명령어를 사용하는 것을 말한다. 이 백업 작업을 하는 동안 데이터베이스를 중지 하면 안된다.

4. 파일 시스템 차원의 데이터베이스 클러스터 디렉토리에 대한 백업이 끝나면 다시 해당 데이터베이스에 슈퍼유저로 접속해서 다음 명령을 실행한다.

   ```
   SELECT pg_stop_backup();
   ```
   이 작업은 데이터베이스 서버 쪽에 이제 백업이 끝났으니, 지금까지 사용했던 WAL 세그먼트 파일들은 아카이브 보관 작업 처리하는 쪽으로 넘기고 새 WAL 세그먼트 파일로 바꾸어서 다시 서버를 일상적인 상태로 실행한다.

5. 다음 이 마지막 로그 파일을 보관 하는 작업이 완료되면 베이스 백업은 끝난다.
   `pg_stop_backup` 명령이 끝나는 시점은 마지막 사용했던 WAL 파일이 `archive_command` 설정값으로 지정된 명령으로 무사히 처리되었을 때다. `pg_stop_backup` 명령은 `archive_mode` 설정값이 on 상태일 때만 실행되며, 이 명령이 실행되기 직전까지 사용하고 있던 WAL 파일은 반드시 정상적으로 보관처리가 되어야 끝난다. 일반적인 환경에서 이 명령은 짧은 시간 안에 끝나지만, 예상했던 것보다 이 명령 시간이 오래 걸리면, `archive_command` 설정값으로 지정된 작업에 문제가 있는지 살펴보고, 그 원인을 해결해야한다. 또한 `archive_command` 설정값이 비어 있어도 이 명령은 끝나지 않는다. `archive_command` 설정값으로 지정된 작업은 성공 할 때까지 반복 한다는 점도 주의 해야한다. 필요하면, `statement_timeout` 설정값을 지정해서 `pg_stop_backup` 작업이 무한 대기 상태로 빠지는 것을 막을 수도 있다.

베이스 백업을 할 때 위와 같은 순서로 직접 하나 하나 작업을 진행할 수도 있지만 pg basebackup 응용 프로그램을 이용할 수도 있다. 이 프로그램은 위 작업을 Agens SQL 서버로 연결해서 해당 쿼리를 실행하고, 리플리케이션 프로토콜을 이용해서 데이터 파일들을 복사하고, 마무리까지 (필요하다면, 백업 중간에 변경 된 xlog 파일의 적용까지) 한 번에 할 수 있다. 또한 이 프로그램은 `pg_start_backup()`/`pg_stop_backup()` 함수를 이용하는 파일 시스템 기반 베이스 백업을 하고 있는 중에도 사용할 수 있다.

몇몇 파일 시스템 백업 툴들은 파일 복사 도중에 그 파일이 변경되면 경고나 오류를 내는 경우도 있다. 운영 중인 데이터베이스를 대상으로 베이스 백업을 만들고 있는 상황에서 이 경고나 오류는 정상적인 것이다. 하지만 이 베이스 백업 작업 자체의 오류와 위 파일이 변경되어서 발생하는 오류를 구분할 필요가 있다. 예를 들어서, rsync 프로그램의 어떤 버전은 "사라져 버린 원본 파일"에 대한 처리를 할 때, 실행 결과 리턴값을 다르게 처리한다. 이것을 이용하면 리턴값을 조사해서 정상적인 처리와 오류 처리를 구분하는 백업 스크립트를 작성할 수도 있을 것이다. 또한, 어떤 tar 버전에서는 위와 같은 경우와 물리적인 묶는 작업 실패에 대한 구분을 하지 않고 같은 리턴 값으로 작업이 종료되기도 한다. 다행히, GNU tar 1.16 이후 버전부터는 백업 도중 원본 파일이 변경된 경우는 1, 그 외 오류인 경우는 2를 리턴하면서 종료한다.

`pg_start_backup` 명령이 실행되고 `pg_stop_backup` 명령을 실행하기 전까지 파일 시스템을 복사하는 작업 시간에 대해서는 신경 쓰지 않아도 된다. 데이터 클러스터 전체를 시스템 차원의 백업 시간과 같이(또는 이 보다 더 늦게) 처리 되어도 괜찮다. 단, 서버 환경 설정이 `full_page_writes` off 상태로 운영되고 있던 서버이면 `pg_start_backup` 명령을 실행할 때 이 값이 on 되기 때문에 `pg_stop_backup` 명령이 실행되기 전까지 그 만큼의 성능 저하가 발생 할 것이다. 이 작업에서 신경 써야 할 부분은 베이스 백업 작업이 길어지고 있을 때 또 새로운 백업을 시도할 경우 이 명령은 무시 되고 그 백업 스크립트 오류로 `pg_stop_backup` 명령이 의도되지 않게 실행되어 기존 베이스 백업 작업을 망치는 경우다. 이 베이스 백업을 할 때는 반드시 위에서 언급한 절차가 모두 순차적으로 끝나야 한다. 중간 작업이 실패하면, 그 백업본은 사용할 수 없다.

백업본을 만들 때는 반드시 데이터 클러스터 디렉토리(대개, 이 디렉토리의 이름은 /usr/local/pgsql/data 이다.) 안에 있는 모든 파일을 보관해야한다. 또한 기본 테이블스페이스 외에 다른 사용자 정의 테이블스페이스를 사용하면 그 파일들로 모두 보관해야한다. 그렇게 하지 않으면 복원 작업이 정상적으로 이루어지지 않는다.

한편, 데이터 클러스터 디렉토리에 대한 백업본을 만들 때 pg_xlog/ 디렉토리를 빼야한다. 이는 클러스터 디렉토리 안에 pg_xlog/ 디렉토리를 두지 않고 그냥 심볼릭 링크로 만들어 사용하면 손쉽게 처리할 수 있다. 또한 복원 작업을 손쉽게 하기 위해서 데이터베이스 서버가 실행 중일 때만 생기는 postmaster.pid, postmaster.opts 파일들의 백업본을 만들 때, 제외해두면 좋다. 이 파일들이 있으면, pg_ctl 명령으로 서버를 실행하려고 할 때 문제가 발생한다. 그래서 수동으로 정리작업을 해 주어야한다.

디렉터리 안에 덤프파일을 만들어서 리플리케이션 슬롯 대신 백업 일부로 사용할 수 있다. 그렇게 하지 않으면 대기 서버를 만들 때마다 마스터 서버에 연결된 리플리케이션 사용자가 계속 마스터 서버의 슬롯을 업데이트하고, hot standby feedback이 활성화된 상태에선 마스터가 폭발할 지도 모른다. 백업이 새 마스터를 만들 때만 사용된다 하더라도 리플리케이션 슬롯을 복사하는 것은 좋지 않다. 슬롯의 컨텐츠도 새 마스터가 연결된 때에는 이미 오래된 정보일 가능성이 높기 때문이다.

이 백업본을 정상적으로 사용하려면 파일 시스템 백업을 시작 한 뒤부터 생긴 WAL 세그먼트 파일들을 모두 보관하고 있어야 한다. 이 작업을 쉽게 할 수 있도록 pg_stop_backup 함수가 실행 되면 *백업 내역 파일*을 만들어 WAL 아카이브 영역 쪽에서 즉시 보관한다. 이 파일 이름은 백업을 시작하면서 첫 체크포인트 작업을 한 다음 세그먼트 번호로 결정된다. 예를 들어, WAL 세그먼트 파일의 백업이 필요한 파일 이름이 000000010000123400000055CD면, 백업 내역 파일의 이름은 000000010000123400000055CD.007C9330.backup으로 된다. (이 이름의 두번째 부분은 해당 세그먼트 내의 정확한 위치를 나타내는데, 크게 신경쓰지 않아도 된다.) 이렇게 해서 이 파일 이름의 세그먼트 파일과 파일 시스템 백업 과정에 생긴 WAL 세그먼트 파일들만 보관하면, 정상적인 복구가 가능하다. 즉, 이 백업 시작 전 세그먼트 파일들은 더 이상 필요 없다. (일반적으로 쓸모 없는 파일들은 16진수로, 백업 내역 파일의 이름보다 작다.) 하지만, 현재 베이스 백업으로 복원이 불가능한 경우를 대비해서 이전 WAL 세그먼트 파일들도 따로 보관해 둘 필요도 있다.

백업 내역 파일은 작은 텍스트 파일이다. 이 파일 안에는 pg_start_backup 함수를 실행할 때 지정한 라벨 문자열이 있고 파일 시스템 백업 시작, 종료 시간, 각 WAL 세그먼트 파일 이름이 기록되어 있다. 파일 시스템 백업과 관련된 라벨을 잘 이용하면 여러 개의 베이스 백업본을 독립적으로 보관할 수 있으며 필요에 따라 그 라벨을 보고 선택적으로 복원 작업을 할 수 있다.

정상적인 복원 작업이 이루어지게 하려면, 이처럼 마지막 베이스 백업 작업을 끝낸 뒤부터 다음 베이스 백업 작업을 할 때까지의 간격에 대해서 고민할 필요가 있다. 간격이 길면, 그 만큼 복원 작업에서 사용될 WAL 세그먼트 파일이 많아지기 때문이다. 또한 백업 저장 장치의 공간도 고려되어야 한다. 백업 저장 장치의 입출력 속도와 여유 공간과 데이터베이스 서버에서 만들어내는 WAL 세그먼트 파일들과 베이스 백업을 하는 동안 데이터베이스 서버의 서비스 영향도 모두 고려해서 가장 적당한 백업 주기를 선택하는 것이 좋다.

또한, 데이터베이스 클러스터 디렉토리 안에 있는 backup_label 파일도 신경써야한다. 이 파일은 pg_start_backup 함수가 실행될 때 만들어지며 pg_stop_backup 함수가 실행될 때, 해당 디렉토리에서는 삭제 되고 WAL 세그먼트 파일 백업 처리와 같은 방식으로 보관된다. 즉, 베이스 백업본에는 반드시 이 backup_label 파일이 있게 된다. 베이스 백업이 완료되면 실행 되는 서버 쪽에서는 이 파일이 없어야하고, 백업본에는 이 파일이 있는 것이 정상이다. 만일 그렇지 않다면, 해당 베이스 백업은 비정상적으로 처리된 것이다. 따라서 백업 도중 데이터베이스 서버가 중지 되어 버린 경우 이 점을 감안해서 베이스 백업본을 사용할 수 있는지를 판단해야한다. 또한, 이 backup_label 파일은 복구 작업에 반드시 필요한 정보들을 포함하고 있기 때문에, 이 파일은 임의로 삭제 해서는 안된다.

서버가 중지된 상태라면, 간단하게 데이터베이스 클러스터 디렉토리 전체를 백업하는 것으

로도 충분하다. 이 때는 당연히 pg_start_backup, pg_stop_backup 함수를 사용하지 않는다. 그러므로 서버가 정상적으로 중지된 것이 아니라면 서버가 복원될 때 이 WAL 파일을 사용할 수 있도록 수동으로 따로 보관해야할 필요가 있다.

8.3.4. 아카이브 모드 백업을 이용한 복구

이제 최악의 사고가 일어났을 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 파일들, 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 옛 데이터베이스에서 사용하는 원본 경로와 같아서 옛 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. (이 파일의 내용에 대한 자세한 설명은 10장 참조) 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 Agens SQL에서 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(nonzero) 것이어야 한다는 점이다. 이 작업은 아카이브 백업 디렉토리 안에 없는 파일도 요청한다. 이 때 이 작업의 결과는 반드시 0이 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. 자세한 이야기는 8.3.5절에서 설명하고 있다.

복구 작업은 `recovery.conf` 파일에서 복구 중지 지점을 지정해서 원하는 지점에서 복구 작업을 중지할 수도 있다. 이 지점을 "recovery target"이라고 한다. 이 기능은 미숙한 데이터베이스 관리자가 실수로 중요 테이블을 삭제하는 것 같은 운영상 실수에 대한 복구를 지원하는데 유용하다. "recovery target"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 손쉽다.

참고: 복구 작업을 멈출 시점은 베이스 백업 시간 이후여야 한다. `pg_stop_backup` 명령이 실행된 시간보다 늦은 시점으로 지정해야 한다. 베이스 백업 - 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 "recovery target" 으로 지정해서 복구한다. 복구 작업 도중 외부적인 영향으로 작업이 중지 되면 (시스템 리부팅, disk full, ... 이런 이유들) 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.

8.3.5. 타임라인

어떤 데이터베이스를 특정 시점으로 복원하는 것은 마치 공상과학 소설에서 나오는 시간여행이나 다원우주론 같이 복잡한 여러 문제점을 만든다. 예를 들어서, 그 데이터베이스의 최초의 변경 내역에서 화요일 오후 5시 15분에 중요한 테이블을 지웠고, 그것을 수요일 아침까지 몰랐다고 치자. 관리자는 침착하게 백업 자료를 가지고, 화요일 오후 5시 14분까지 복구를 하고, 서버를 운영했다. 이렇게 해서 두 번째 변경 내역이 진행되었고, 이 데이터베이스 변경 내역에서는 그 테이블이 삭제 되지 않고 계속 운영 되었다. 하지만, 좀 있다가 판단해 보니, 이 판단이 바람직한 것이 아닌 것 같아 다시 최초의 변화 내역 기준으로 수요일 아침 데이터베이스 상태로 (그 중요 테이블이 지원진 채로 운영된 상태) 다시 되돌아 가야할 필요가 생긴다. 단순하게 생각하면, 이미 한 번 과거로 되돌아 갔고, 거기서 이미 WAL 파일을 덮어써 버렸기 때문에, 이 작업이 불가능할 것 같다. 이 문제를 해결하기 위해서 백업 WAL 세그먼트 파일을 이용한 데이터베이스 복구 작업을 할 때, 서버가 각각의 복구 작업에 필요한 WAL 세그먼트 파일의 이름을 다르게 해서 사용한다.

이것을 Agens SQL에서는 *타임라인*이라는 개념으로 설명하고 있다. 복구 작업이 끝나고, 데이터베이스가 새롭게 실행되면 새로운 타임라인 ID로 WAL 레코드를 만든다. 이 타임라인 ID는 WAL 세그먼트 파일 이름의 한 부분으로 사용되어 같은 트랜잭션 번호에 대해서도 다른 WAL 파일을 만들 수 있도록 해서 예전 데이터베이스 변경 내역 정보를 덮어쓰지 않도록 한다. 이런 방식이면 얼마든지 다양한 타임라인이 존재할 수 있다. 필요 없는 기능 같아 보이기는 하지만 이 기능이 가끔 구원자 역할을 톡톡히 한다. 복구시점을 정확히 모르고 그 시점을 정확하게 찾을 때까지 계속 복구 작업을 해야 하는 상황에 아주 유용하게 사용된다. 타임라인 정보가 없으면 이런 작업은 거의 관리가 안될 정도로 복잡해저버린다. 타임라인을 사용하면 이미 분기 되어 더이상 사용하지 않는 어떠한 이전 데이터베이스 상태로도 되돌아갈 수 있다.

새로운 타임라인으로 운영될 때마다 Agens SQL에서는 "타임라인 내역" 파일을 만든다. 이 파일에는 언제, 어디서 분기되었는지에 대한 정보를 담고 있다. 이 내역 파일은 다중 타임라인이 있을 경우 사용자가 지정한 특정 타임라인을 사용할 때 어느 WAL 파일을 사용해야 하는지에 대한 정보를 담고 있어서 복원 작업에 사용된다. 즉, 이 파일 또한 새롭게 생성 되면 바로 WAL 세그먼트 파일 백업 처리할 때와 마찬가지로 archive_command 설정값에서 지정한 작업을 진행해서 따로 보관 된다. 이 내역 파일은 (큰 크기의 WAL 세그먼트 파일과 달리) 단순한 텍스트 파일이다. 그래서 충분히 많은 파일들을 보관할 수 있다. 또한 해당 내역 파일을 문서 편집기로 열어서 그 복구 내역에 대한 구체적인 내용을 기록해 두면 나중에 여러모로 쓸모가 있을 것이다.

복구 작업에서 기준이 되는 타임라인은 특별히 다른 타임라인을 지정하지 않으며, 베이스 백업 당시 사용되었던 타임라인을 사용한다. 다른 타임라인을 사용하려면, `recovery.conf`

파일에 원하는 타임라인 ID를 지정해주면 된다. 물론 이 타임라인은 반드시 베이스 백업 뒤에 사용한 하위 타임라인 가운데 하나여야 한다.

8.3.6. 팁과 예제

아카이브 모드를 사용하는 환경설정과 관련된 몇 가지 팁과 예제를 여기서 소개한다.

8.3.6.1. 독립된 핫백업

Agens SQL 백업 방식으로 독립된 핫백업을 구현할 수 있다. 이 방식으로 구현하면 특정시점 복구는 불가능하지만, 장애복구시 pg_dump 프로그램을 이용해서 만든 덤프 파일을 사용하는 복구보다 훨씬 빠른 시간안에 복구할 수 있다. (물론 pg_dump의 덤프 파일 크기가 데이터 클러스터 디렉토리 사이즈에 비해 아주 적은 경우 오히려 더 늦을 수도 있다.) 베이스 백업에서 언급했던 것처럼 독립된 핫백업 서버를 구축하는 가장 손쉬운 방법은 pg basebackup 툴을 이용해서 베이스 백업을 받는 것이다. pg basebackup을 호출할 때 `-x` 매개변수를 쓰면 사용되는 트랜잭션 로그 전체가 백업에 자동으로 쓰이고, 그 이상의 작업은 필요 없다.

백업 파일을 복사하는데 보다 유연한 방법이 필요한 경우, 보다 저수준의 작업으로 독립된 핫백업을 구현할 수 있다. 독립된 핫백업을 구축하려면, `wal_level` 값은 `archive`, `archive_mode` 값은 `on`, `archive_command` 값으로 스위치 파일이 있는 경우에 한해서 그 파일을 다른 곳에도 보관할 수 있도록 설정하면 된다. 다음은 이 설정의 예제다:

```
archive_command = 'test ! -f /var/lib/pgsql/backup_in_progress || (test ! -f /var/lib/pgs
```

위 설정은 `/var/lib/pgsql/backup_in_progress` 파일이 없으면 그냥 건너뛰고, 이 파일이 있으면 복사하려는 WAL 세그먼트 파일이 없는 경우 다른 곳에 복사하는 작업을 한다. (이렇게 해서, Agens SQL 서버가 더이상 쓸모가 없는 WAL 파일을 다시 사용할 수 있도록 한다.)

WAL 세그먼트 파일들의 백업 설정을 위와 같이 해 두고, 아래와 같이 베이스 백업 스크립트를 만든다:

```
touch /var/lib/pgsql/backup_in_progress
psql -c "select pg_start_backup('hot_backup');"
tar -cf /var/lib/pgsql/backup.tar /var/lib/pgsql/data/
psql -c "select pg_stop_backup();"
rm /var/lib/pgsql/backup_in_progress
tar -rf /var/lib/pgsql/backup.tar /var/lib/pgsql/archive/
```

`/var/lib/pgsql/backup_in_progress` 이름으로 스위치 파일을 먼저 만들어 베이스 백업을 준비하는 동안만 WAL 파일을 다른 장소에 따로 보관하고, 베이스 백업을 tar 파일로 만들고, 백업 작업이 끝났다고 서버쪽에 알리고, 스위치 파일을 지우고, tar 작업을 하는 동안 생긴 생긴 WAL 파일들을 같은 tar 파일에 추가하고 작업을 끝낸다. 이 때 이 스크립트의 오류를 확인해 예외처리를 해야할 필요도 있다.

저장 공간이 넉넉치 않으면, pg_compresslog http://pglesslog.projects.postgresql.org 프로그램을 이용해서, 필요없는 full page writes 정보와 실제 작업할 내용이 없는 WAL 파일의 영역을 정리해서 보관할 수도 있다. 그리고 pg_compresslog 프로그램으로 정리된 내용을 gzip 프로그램을 이용해서 더 압축할 수도 있다:

```
archive_command = 'pg_compresslog %p - | gzip > /var/lib/pgsql/archive/%f'
```

복구 할 때는 gunzip 프로그램과 pg_decompresslog 프로그램을 다음과 같이 사용한다:

```
restore_command = 'gunzip < /mnt/server/archivedir/%f | pg_decompresslog - %p'
```

8.3.6.2. archive_command 스크립트

많은 사람들이 archive_command 설정값으로 postgresql.conf 파일에 아래와 같이 스크립트를 사용해서 그 값을 단순화 한다.

```
archive_command = 'local_backup_script.sh "%p" "%f"'
```

아카이빙 작업을 단순한 시스템 명령들의 조합으로 설정하지 않고 하나의 스크립트로 지정해 놓으면 백업 관련 작업과 데이터베이스 관리 작업을 분리할 수 있어 얻는 이득이 많다. 필요시 그 스크립트만 수정하면 백업 정책을 언제든지 바꿀 수 있기 때문이다. 이렇게 하면 자주 쓰는 bash 또는 perl 스크립트로 작성할 수 있으며 보다 안전하고 꼼꼼한 작업을 할 수 있다.

> 작은 정보: archive_command 에서 지정한 스크립트의 로그를 서버 로그로 저장하고 싶으면 logging collector 환경 설정값을 on으로 켜두면 된다. 이렇게 하면 스크립트에서 stderr 쪽으로 출력하는 모든 내용은 서버 로그로 저장된다. 이렇게 해서 보다 손 쉽게 오류를 검사 할 수 있다.

스크립트를 실행하면서 생기는 모든 stderr 출력은 데이터베이스 서버 로그로 기록된다. 이렇게 해서, 백업 스크립트 오류를 쉽게 진단할 수 있다.

이 스크립트를 만들 때는 다음 항목들에 대한 고려도 함께 하면 좋다.

- 파일이 복사 되면서 발생하는 파일 누출 문제 - 보안 문제
- WAL 파일 복사(전송, 다른 곳에 따로 보관하는 것)의 가장 안전하고 확실한 방법 - 실패 했을 경우 재시도에 대한 정책, 복구를 빠르게 하기 위한 이중 백업 등
- 다른 백업, 복구 소프트웨어를 사용할 때의 인터페이스 문제
- 오류 모니터링 소프트웨어와의 인터페이스 문제

8.3.7. 위험부담

여기서는 아카이브 모드 백업 기술의 여러가지 한계점을 소개하고 있다. 이 한계들은 추후 배포판에서 수정될 예정이다.

- 해시 인덱스 작업에 대한 기록은 WAL 로그에 포함되지 않는다. 그래서 이런 인덱스들에 대해서는 대기(standby) 서버(복원 서버)에서 삽입, 갱신 작업을 하지 않는다. 다시 말하면, 이렇게 제대로 반영되지 않은 인덱스 때문에 원하는 결과가 출력되지 않을 수 있다. 만일 해시 인덱스를 사용하고 있으면 복원 작업이 끝난 뒤에 수동으로 REINDEX 작업을 해야한다.
- 베이스 백업이 시작 된 다음 CREATE DATABASE 명령으로 새 데이터베이스가 만들어지고, 그 데이터베이스를 만들 때 원본 템플릿 데이터베이스가 베이스 백업이 끝나기 전에 변경 되었으면 복원 작업을 할 때 원본과 같은 새 데이터베이스가 만들어지지 않을 가능성이 있다. 이 문제를 피하려면, 베이스 백업 중에는 템플릿 데이터베이스를 변경 하지 않는 것이 가장 좋은 방법이다.
- CREATE TABLESPACE 명령은 거기서 사용하는 실제 경로가 그대로 반영 된다. 이 말은 복원 하는 시점에 미리 그 실제 경로의 디렉토리가 있어야함을 의미한다(물론 처음 테이블스

페이스를 만들 때처럼 그 디렉토리는 비어있고, 소유주와 접근권한이 데이터베이스가 사용할 수 있는 상태여야한다). 이 점은 복원 작업에서 상당한 주의가 필요하다. 만일 같은 호스트에서 복원을 하는데 임시 복원 작업을 진행하면서 그 테이블스페이스에 있던 원본 파일을 손상 할 수도 있으며 다른 호스트에서 테이블스페이스의 실제 디렉토리가 없어서 원본 자료를 손실할 수도 있다. 이 문제를 피해갈 수 있는 가장 좋은 방법은 일단 테이블스페이스를 만들거나 삭제할 경우에는 반드시 그 다음에 베이스 백업본을 만들어 두는 것이다.

또 다른 단점은 기본 WAL 포맷은 디스크 페이지 스냅샷을 많이 포함하고 있어 그 크기가 상당히 크다는 점이다. 이 페이지 스냅샷 기법은 서버가 갑자기 중지될 경우와 해당 디스크 페이지가 부분적으로 기록되었을 경우 그 페이지를 안전하게 복구 하기 위해서 도입되었다. 물론 사용하고 있는 하드웨어와 소프트웨어에 따라 이런 부분적인 디스크 페이지 기록으로 인한 오류가 미비하거나 무시할 정도인 경우도 있다. 이럴 경우, full page writes 환경변수 값을 off로 설정하면, 이 로그 전체 크기를 많이 줄일 수 있다. 이 환경변수값을 off로 설정해도 PITR 기능을 사용할 수 있다. 또한 `full_page_writes` 값이 on 이더라도 아카이브 로그를 만들 때 필요없는 페이지 복사본들은 버리고 복원에 필요한 것만 담아서 만드는 방식도 도입되어야 할 것이다. (미래의 개발자들에게 이 일은 넘긴다.) 운영시 관리자가 체크 포인트 간격을 늘려서 이 WAL 로그를 적게 만드는 방법도 한 방법이다.

9장. 고가용성, 부하 분산, 복제

데이터베이스 서버는 하나의 운영 서버에서 장애가 생기면, 대기 서버가 빠르게 운영 서버로 운영될 수 있도록 구축할 수 있다. 이것을 고가용성 high availability 이라고 한다. 또한, 여러 대의 똑같은 자료를 사용하는 서버를 동시에 운영해서, 서버 부하를 분산해서 운영할 수도 있다. 이것을 부하 분산 load balancing 이라고 한다. 단순하게 이상적으로만 생각한다면 데이터베이스 서버도 별 문제 없이 여러 대의 같은 서버가 운영될 수 있을 것이다. 정적인 웹 페이지를 제공하는 웹 서버라면 여러 대의 웹 서버를 구축해서 여러 웹 블로우져 요청에 동시에 해당 페이지를 제공해서 부하를 분산하는 작업은 그리 어려운 일이 아니다. 하지만 데이터베이스가 읽기 전용 형태로 서비스 되는 경우는 극히 드물다. 또한 하나의 요청에 내부적으로 자료를 읽고 쓰는 작업이 복잡하게 얽혀 있어 쓰기 작업은 다른 서버에서 그 변경된 자료를 읽는 서비스까지 제공해야하기 때문에 자료의 일관성 - 자료 정합성이라고 한다 - 을 유지하는 작업이 꽤 난해한 작업이다.

이 동기화 문제가 바로 병렬 데이터베이스 서버를 구축하기 어려운 가장 근본적인 문제다. 이 문제를 해결 하는 여러 방법들이 있으며, 서로 다른 개선책을 제시하고 있다.

한 가지 해결책은 오직 하나의 데이터베이스만 읽기/쓰기를 허용하는 것이다. 이 때, 이 서버를 *마스터* 또는 *프라이머리* 서버라고 한다. 마스터 서버의 변경된 내용이 그대로 내부적으로 동기화 되는 서버를 *대기(스탠바이)* 또는 *슬래이브* 서버라고 한다. 마스터 서버의 장애로 운영 제어권을 대기 서버로 남기기 전까지 대기 서버를 사용할 없는 환경을 *warm standby* 서버 환경이라고 하고, 이와 달리 대기 서버를 읽기 전용으로 사용이 가능한 환경을 *hot standby* 서버 환경이라고 한다.

다른 한 가지 해결책은 동기화 문제의 안전성을 보장하는 것이다. 데이터 변경 트랜잭션이 발생 했을 때, 사용하는 모든 서버에서 그 트랜잭션이 커밋 되었을 때만 그 트랜잭션을 유효한 트랜잭션으로 처리하는 방식이다. 이렇게 구현되면, 부하를 분산할 수 있으며, 하나의 서버에서 장애가 발생되어도 자료의 정합성은 유지될 수 있다. 문제는 서버 자료 동기화 방식을 비동기식으로 처리한다면, 동기화 작업에 지연이 생긴 상황에서 한 서버에서 장애가 발생하면, 자료 손실이 있을 수 있다. 비동기방식을 사용하는 경우는 서버간 통신이 아주 느린 경우에 사용한다.

또 다른 한 가지 해결책은 자료 자체를 나눠서 동기화 하는 방법이다. 데이터베이스 단위이거나, 테이블 단위로 해당 부분만 동기화한다.

이런 여러 방식 가운데 어떤 방식을 선택할 것인가는 서버의 운영 특성과 자료 동기화 하는 방식의 성능에 따라 고려 해야한다. 일반적으로 기능과 성능간의 상관관계를 가진다. 예를 들어, 느린 네트워크 환경에서는 전체 동기화 방식을 사용한다면, 성능은 절반 이하로 떨어질 것이지만, 이런 경우에 비동기화 방식을 사용한다면, 서버 성능 저하를 최소화 할 수 있다.

다음은 장애처리(failover), 복제(replication), 부하 분산(load balancing) 기능을 제공하는 여러 기법을 대략적으로 소개하고 있다. 각 용어들의 자세한 설명은 용어설명 (http://www.postgres-r.org/documentation/terms) 페이지를 참조하라.

9.1. 여러 해결 기법 비교

공유 디스크를 이용한 장애처리(failover)

> 공유 디스크를 이용한 장애처리는 하나의 물리적인 데이터베이스 자료 파일만 사용하기 때문에, 자료 동기화에 대한 비용을 최소화 할 수 있다. 하나의 디스크 영역을 여러 서버가 공유하는 방식이다. 운영 서버에 장애가 생기면 대기 서버가 그 디스크 영역을 마운트 해서, 데이터베이스를 복구 모드로 실행한다. 이 방식은 자료 손실이 없이 빠르게 장애처리를 할 수 있다.

이 방식은 일반적으로 네트워크 저장 장치와 같이 하드웨어 기능을 이용해서 공유하는 방식이다. 네트워크 파일 시스템을 이용할 수도 있으나, 그 파일 시스템이 POSIX 모든 기능을 다 수용할 수 있어야한다. (자세한 이야기는 1.2.1절 참조) 이 방식의 한 가지 한계점은 운영서버가 그 디스크 영역을 쓰는 동안에는 대기서버가 전혀 그 디스크 영역을 사용할 수 없는 것이며, 만일 그 디스크 영역에 장애가 발생하면, 두 서버 모두 사용할 수 없다는 점이다.

파일 시스템 (블록-장치) 복제

하드웨어 기능을 이용한 또 다른 방식은 파일 시스템 자체를 복제하는 방식이다. 데이터베이스가 운영되어 데이터 파일의 변경이 있었다면, 그 변경 내용에 대해서 다른 서버에도 파일 시스템 차원에서 동기화한다. 제약 조건은 운영 서버의 변경 순서와 동일한 순서로 파일 시스템이 대기 서버에도 복제되어야한다. 이 방식을 이용한 리눅스 솔루션으로는 DRBD이다.

트랜잭션 로그 전달

운영 서버의 미리 쓰는 로그(WAL)의 내용을 대기 서버로 옮겨 그대로 다시 실행해서 운영 서버 상태와 같은 상태를 유지하도록 하는 방법이다. 운영 서버가 장애로 중지되면 즉시 대기 서버가 운영 서버로 운영 될 수 있다. WAL 내용을 전달하는 방식은 동기식 또는 비동기식으로 이루어지며, 데이터베이스 서버 전체를 대상으로 한다. 부분만 복제할 수 없다.

대기 서버를 구축하려면, WAL 내용을 대기 서버로 전달해야하 하는데, 그 전달하는 방법으로 파일 기반 로그 복사(log shipping)를 하는 방식(9.2절)과 스트리밍 리플리케이션(9.2.5절) 방식, 이 두 방법을 혼용해서 사용할 수 있다. hot standby 서버 구축에 대한 자세한 설명은 9.5절 에서 자세히 다루고 있다.

트리거 기반 운영-대기 복제

운영-대기 복제 환경의 기본 기능은 운영 서버에 일어난 자료 변경에 관계된 모든 쿼리를 대기 서버 쪽에도 똑같이 실행해서 운영 서버와 같은 환경을 만든다. 그리고 대기 서버에서는 읽기 전용 쿼리만 허용해서, 자료 통계 작업 같은 할 수 있도록 이용한다.

이 방식을 이용하는 제품으로는 Slony-I이 있다. 테이블 단위 복제가 가능하며, 여러 대의 대기 서버를 함께 운영 할 수 있다. 대기 서버의 자료 동기화 방식으로 비동기식이 사용되므로 장애 발생시 자료 손실이 있을 수 있다.

명령 구문 기반 복제 미들웨어

데이터베이스 서버로 보내는 모든 자료 변경 관련 쿼리를 미들웨어가 가로채서 관리되는 모든 데이터베이스 서버에 동시에 그 쿼리를 실행하는 방식이다. 각각의 서버들은 서로 의존적이지 않게 운영될 수 있다. 읽기 전용 작업이라면 그냥 한 서버 쪽으로만 작업하면 되고, 쓰기 작업이라면 모든 서버로 보낸다. 물론 변경 작업 뒤 변경이 반영된 자료를 조회 하고자 한다면, 조회 하는 서버에 변경 작업 완료가 보장되어야한다.

이 방식에서는 `random()` 함수나 `CURRENT_TIMESTAMP`, 시퀀스 값과 같이 개별 서버에 의존적인 쿼리들에 대해서는 의도치 않게 일관성을 유지하지 못 할 수 있다. 이런 쿼리들이 자료 변경 작업에 사용되면 전혀 엉뚱한 결과가 나올 수도 있다. 이런 문제에 대한 해결책을 미들웨어가 제시하지 못하면 응용 프로그램에서 이런 문제의 해결책을 반드시 찾고 사용해야한다. 한 방법은 한 서버에 먼저 자료를 반영해서 반영 결과를 가지고 다른 서버에 모두 반영하는 방법이 있고, 다른 방법으로 자료 변경에 대해서는 특정 시점 복구 방식을 이용한 운영-대기 서버 형태로 구축해서 자료 변경은 반드시 운영 서버에서만 일어나도록 미들웨어나 응용프로그램 사용하는 것이다. 또 한 가지 주의할 사항은 한 서버에서 트랜잭션이 시작되어 커밋되었을 때, 다른 모든 서버에서도 모두 커밋되어야 그 트랜잭션이 유효하다고 처리되어야한다. 특정 서버에서 롤백되면, 그 트랜잭션은 모든 서버에서 롤백되어야한다. 이것을 구현하기 위해서는 미들웨어가 2중 커밋 two-phase

commit 을 이용할 것이다 (PREPARE TRANSACTION, COMMIT PREPARED 명령). 이 방식을 이용하는 제품으로는 Pgpool-II와 Continuent Tungsten가 있다.

비동기식 다중 운영 서버 복제

PC와 노트북, PC와 리모트 서버간 연결 같이, 데이터베이스 서버간 연결이 항구적이지 않을 때, 그 자료의 정합성을 유지한다는 것은 꽤 고급 기술이다. 비동기식 다중 운영 서버 환경에서 복제 기능은 다음 문제점을 해결해야 한다. 각 서버는 독립적으로 운영 되어야하며, 주기적으로 트랜잭션 충돌을 피하기 위해 서버간에 통신 해야한다. 충돌이 발생하면 사용자가 해결 할 수 있는 방법을 제공하거나 충돌 해결 정책에 따라 자동으로 해결할 수 있는 기능이 있어야한다. 이 방식을 이용하는 제품으로는 Bucardo가 있다.

동기식 다중 운영 서버 복제

모든 자료 변경이 모든 서버에 바르게 정리 되어야만 다음 작업을 할 수 있도록 구축되기 때문에 자료 변경이 빈번한 환경이면 단독 운영 서버 구축 환경 보다 성능이 더 떨어질 수 있다. 읽기 전용 작업이라면 어느 서버에서도 작업할 수 있다. 서버간 통신 작업 비용을 줄이기 위해서, 몇 시스템은 공유 디스크를 사용하기도 한다. 자료 관리 입장에서 본다면 이 방식이 가장 이상적인 방식이다. 하나의 자료 결과에 대해서도 서버간 자료가 공유되기 때문에 `random()` 함수와 같은 것은 문제 없이 사용할 수 있다.

Agens SQL에서는 이 방식의 복제 기능은 제공하지 않는다. 하지만, Agens SQL 이중 커밋 two-phase commit (PREPARE TRANSACTION, COMMIT PREPARED) 명령이 이 방식을 구현 하는데 사용될 수 있다.

상용 솔루션

Agens SQL은 소스가 공개 되어 있고 쉽게 확장이 가능하기 때문에, 이것을 가지고 독자적인 기법으로 장애처리, 복제, 부하 분산 기능을 포함 시켜 소스를 공개하지 않은 채 상용으로 판매 할 수 있다. 이렇게 몇몇 솔루션들이 판매되고 있다.

다음 표 9-1 표는 앞에 언급한 다양한 기능들을 요약한 것이다.

표 **9-1.** 고가용성**,** 부하 분산**,** 복제 기능 도표

기능	공유 디스크 장애처리	파일 시스템 복제	트랜잭션 로그 전달	트리거 기반 운영-대기 복제	명령 구문 기반 미들웨어	비동기식 다중 운영 복제	동기식 다중 운영 복제
많이 알려진 제품	NAS	DRBD	스트리밍 리플리케이션	Slony	pgpool-II	Bucardo	
통신 방법	공유 디스크	디스크 블럭	WAL	테이블 로우	SQL	테이블 로우	테이블 로우와 로우 잠금
특별한 하드웨어 필요없음		•	•	•	•	•	•
다중 운영 서버 구축가능					•	•	•
운영 서버 측 복제 부하 없음	•		•		•		

기능	공유 디스크 장애처리	파일 시스템 복제	트랜잭션 로그 전달	트리거 기반 운영-대기 복제	명령 구문 기반 미들웨어	비동기식 다중 운영 복제	동기식 다중 운영 복제
다른 서버들의 지연 없음	•		비동기식일 때	•		•	
운영서버 장애시 자료 손실 없음	•	•	동기식일 때		•		•
대기 서버 쪽 읽기전용 허용			hot standby 일때	•	•	•	•
테이블 단위 복제가능				•		•	•
자료 충돌 해결 작업 필요없음	•	•	•	•			•

위에서 소개한 해별 방법 외에도 아래와 같은 방법도 있다:

자료 분산

자료 분산 방식은 하나의 테이블에 보관되는 자료를 여러 테이블로 나눠서 각 테이블을 각각의 데이터베이스 서버가 처리하는 방식이다. 예를 들어 런던 지사, 파리 지사 자료를 각각의 서버로 처리하는 방식이다. 만일 런던 지사 자료와 파리 지사 자료를 합쳐서 조회를 해야하면 그 일은 응용 프로그램에서 맡던지 아니면 읽기 전용 복제 방식을 이용해서 서로 자료를 공유하는 하는 방식을 이용한다.

다중 서버 병렬 쿼리 실행

위에서 언급한 방법들은 여러 서버 쪽으로 여러 쿼리를 사용하는 방법이지, 하나의 쿼리를 여러 서버에서 분산 처리하는 방식은 아니다. 이 방식으로 먼저 중앙 서버가 있고 한 쿼리는 중앙 서버에서 실행되고 중앙 서버는 각각의 하위 서버로 쿼리를 나눠서 보내고 결과를 중앙 서버가 취합해서 사용자에게 보내주는 방식이 사용된다. Pgpool-II 제품이 이런 기능을 사용할 수 있다. PL/Proxy 제품도 이런 방식을 사용할 수 있다.

9.2. 로그 전달(Log-Shipping) 대기 서버

운영 서버에서 만드는 트랜잭션 로그 조각을 정기적으로 대기 서버로 옮기고 그것을 적용시켜, 운영 서버가 장애로 멈추게 되면 대기 서버를 운영해서 가용성을 향상할 수 있다. 이 기능을 *warm standby* 또는 *log shipping* 기능이라고 한다.

이 복제 방식은 먼저, 운영 서버와 대기 서버가 모두 실행 중이어야한다. 하지만 두 서버 쌍방간의 연결 상태는 다른 복제 방식보다 나빠도 괜찮다. 운영 서버는 아카이브 모드로 운영되어야하며, 운영 중에 생기는 다 쓴 WAL 세그먼트 파일(스위칭된 트랜잭션 로그 파일)을 차례대로 대기 서버로 보내고, 대기 서버는 복구 모드 전용(복구가 끝나도 다음 복구 파일이 있으면 계속 복구 작업을 하는 상태)으로 실행된다. 이 방식을 이용하면 데이터베이스 테이블들

을 수정 해야할 필요가 없다. 또한 다른 복제 방식에 비해 관리 작업 비용도 적으며 복제 작업이 운영 서버 쪽으로 끼치는 영향도 다른 복제 방식보다 적다.

이 방식의 구현 방법은 간단하다. 운영 서버에서 다 쓴 WAL 파일을 다른 서버로 운송(shipping) 하는 것 뿐이다. Agens SQL에서 로그 옮기는 작업은 한 번에 하나의 로그 파일을 옮길 수 있도록 구현되어 있다. WAL 파일(16MB)을 옮기는 작업은 데이터베이스 서버 밖에서 관리자가 정의한 방식으로 진행 되기 때문에 같은 사이트 내로 옮겨도 되고 전혀 다른 시스템 쪽으로 보내도 되고 여러 시스템으로 한꺼번에 보내도 된다. 이 부분은 전적으로 관리자에게 맡긴다. 단지 고려해야할 사항은 파일이 전송될 때의 전송량 때문에 운영 서버에 영향을 줄 수도 있다. 이 부분이 염려되면, 전송 속도를 제한 할 수 있는 방법도 고려해야할 것이다. 아니면 레코드 기반 로그 전달 방식 (스트리밍 복제)을 고려할 수 도 있다. (9.2.5절 참조)

로그 전달 방식은 비동기식임을 기억해야 한다. 다시 말하면 전송하는 WAL 내용은 이미 커밋된 자료이기 때문에 그 자료가 대기 서버로 미처 전달 되기전에 운영 서버가 멈춰버리면, 그 자료는 손실 된다. 자료 손실량을 줄이는 방법으로 `archive_timeout` 환경설정값을 몇 초 정도로 짧게 지정해 자주 사용하는 WAL 파일을 바꾸고 그것을 전송하면 되겠지만 그 파일의 크기가 16MB이기 때문에 잦은 WAL 파일 전송 작업으로 네트워크 사용량이 증가할 것이다. 스트리밍 복제 방식(9.2.5절 참조)을 이용하면 이 손실 되는 자료량을 최소화할 수 있다.

(물론 위에서 언급한 한계점이 있기는 하지만) 대기 서버로 넘어 오는 WAL 파일이 제때에 잘 넘어 온다면, 운영 서버가 중지 되어 대기 서버가 그 역할을 맡기까지 서비스가 중지되는 시각은 극히 짧다. 이렇게 가용성을 향상 시킨다. 베이스 백업 자료를 준비하고, 지금까지 보관해둔 WAL 파일을 가지고 서버를 복구 하는 방법은 이 방식보다 꽤 많은 서비스 중지 시간이 필요할 것이다. 서버가 복구 되는 기술적인 방식은 동일하지만 가용성 입장에서는 차이가 난다. warm standby 방식으로 구현되면 대기 서버 쪽에서는 어떤 쿼리도 사용할 수 없다. 대기 서버 쪽에서 읽기 전용 쿼리를 사용하려면, hot standby 방식으로 구축해야한다. 이 부분은 9.5절에서 자세히 설명한다.

9.2.1. 계획

일반적으로 데이터베이스 시스템을 구축할 때, 운영 서버와 대기 서버를 함께 구축한다면 그 두 서버 환경은 최대한 비슷한 환경으로 구축하는 것이 제일 좋다. 물론 하드웨어 사양은 틀려도 크게 문제가 되지 않으나 테이블스페이스 문제를 고려했을 때만 보아도 운영 서버에서 CREATE TABLESPACE 명령으로 테이블스페이스를 조작할 일이 생겼을 때 대기 서버에서도 똑 같은 작업이 정상적으로 진행되려면 운영 서버와 같은 디렉토리 구조를 가지고 있어야 한다. 물론 운영체제 전체를 운영하는 입장에서 봤을 때 하드웨어와 운영체제도 똑같은 사양이면 보다 쉽게 운영할 수 있을 것이다. 하드웨어 머신 아키텍쳐는 같아야한다. 로그 전달 방식의 복제를 사용할 경우 운영 서버가 64bit이면 거기서 만든 로그 파일을 32bit 대기 서버에서는 사용할 수 없다.

일반적으로 로그 전달 방식 복제는 메이저 버전이 서로 다른 Agens SQL 서버 간은 구현이 불가능하다. 디스크 자료 저장 포맷이 바뀔 경우에, Agens SQL 개발 그룹에서는 메이저 버전을 바꾸어 출시하는 것이 정책이기 때문이다. 물론 마이너 버전의 서로 다른 경우는 복제 환경을 구축해도 잘 작동할 것이다. 이렇게 서로 마이너 버전이 다른 경우 해결 하기 힘든 예상치 못한 문제가 발생할 수도 있기 때문에, 가능하면 운영 서버와 대기 서버의 버전을 최대한 같은 것을 사용하는 것을 권장한다. 마이너 버전 업데이트가 필요하면 이 작업을 할 때 가장 좋은 순서는 먼저 대기 서버 쪽을 업데이트해서 옛 버전의 WAL 파일을 잘 적용할 수 있는지 부터 확인하고 운영 서버를 업데이트하는 것이다.

9.2.2. 대기 서버 동작방식

서버가 대기 모드로 실행되면 서버는 마스터 서버에서 받는 WAL 파일을 계속해서 자신의

서버에 반영하는 작업만 한다. 대기 서버는 운영 서버가 보내는 파일을 보관해 두는 디렉토리에 새 WAL 파일이 있는지 확인해서 새 WAL 파일을 반영하는 방식(warm standby)도 있고 (restore command 참조), TCP 연결 방식을 이용해서 운영 서버와 직접 연결하고 커밋된 트랜잭션을 즉시 대기 서버로 반영하는 방식(스트리밍 복제)도 있다. 또 내부적으로 보면 대기 서버의 pg_xlog 디렉토리에 있는 WAL 파일을 참조하는 경우도 있다. 이 경우는 스트리밍 복제 환경에서 운영 서버에서 보낸 트랜잭션을 미처 반영하기 전에 대기 서버가 중지 되었다가 다시 실행될 경우에 일어난다. 물론 관리자가 직접 pg_xlog 디렉토리에 직접 WAL 파일을 두고 서버를 실행할 수도 있다.

대기 서버가 실행되면, 제일 먼저 restore_command 설정값에 지정된 명령어를 진행한다. 적용 해야할 WAL 세그먼트 파일이 아카이브 디렉토리에 있는지 확인하고, 있으면 차례대로 적용한다. 이 작업이 끝나고 더 이상 적용할 파일이 없으면 restore_command 설정값에 지정된 명령은 실패로 끝나야한다. 복구 환경이 스트리밍 복제 방식이면 운영 서버 쪽으로 접속해서 운영 서버가 데이터베이스 연결을 통해 보내는 트랜잭션들을 자신의 pg_xlog 디렉토리 내 해당 WAL 로그 파일에 추가한다. 그 다음부터는 서버가 알아서 해당 작업을 수행하고 커밋 되었다고 기록하고, 체크포인트 작업을 하는 등 일련의 자료 변경 작업을 수행한다. 반면, 복구 환경이 warm standy 방식이면 restore_command 설정값에 지정된 명령이 실패로 끝난 경우 원하는 WAL 세그먼트 파일이 새롭게 생길 때까지 기다렸다가 생겼을 때 다시 작업 하고 또 더 이상 없으면 실패로 끝나면서 대기하는 작업을 반복한다. 이 반복 작업은 서버가 중지되거나, 대기 작업을 이제 그만 하고 스스로 독립된 운영 서버 역할을 하라고 알려주는 트리거 파일이 생기기 전까지 계속 반복된다.

대기 모드는 **pg_ctl promote** 명령을 실행하거나, 트리거 파일(trigger_file 설정값에 지정한 파일)이 생기면 끝난다. 물론 적용해야할 WAL 파일이 아카이브 디렉토리에 아직 있거나 pg_xlog 디렉토리 내에 있으면 이것들을 모두 적용하고 대기 모드를 끝낸다. 이때 스트리밍 복제를 위한 운영 서버와의 연결도 더 이상 재시도하지 않는다.

9.2.3. 대기 서버 구축을 위한 운영 서버 준비작업

첫번째 작업은 대기 서버로 보낼 WAL 세그먼트 파일을 그냥 버리지 않고 다르게 처리하는 설정을 해야한다. 이 방법에 대한 자세한 이야기는 8.3절에서 하고 있다. 그냥 버리지 않고 다르게 처리 한다는 것은 그 WAL 세그먼트 파일이 운영 서버가 중지되어도 대기 서버가 사용할 수 있는 위치로 옮겨 놓는다는 것을 의미한다. 이 방법에는 여러 방법이 있을 수 있다. 단순히 네트워크 드라이브에 복사를 하는 방법도 있을 수 있고, FTP를 이용해서 대기 서버의 원하는 디렉토리로 업로드 하는 방법도 있고, 백업용 테이프 드라이브 쪽으로 보내는 방법도 있을 것이다. 중요한 것은 운영 서버를 더 이상 사용할 수 없을 때도 대기 서버는 그 WAL 세그먼트 파일들을 사용할 수 있어야한다는 점이다.

스트리밍 복제 방식을 사용하려면, 대기 서버에서 요청하는 데이터베이스 접속을 허용하도록 설정하는 작업을 해야한다. 총 세가지 작업인데 첫번째는 접속 계정의 권한을 복제 기능을 사용할 수 있도록 해야 하고, 두번째는 pg_hba.conf 파일에 대기 서버가 운영되는 호스트가 등록 되어야 하고, 그곳에 데이터베이스 항목은 replication으로 지정 되어야하며, 세번째는 서버 환경 설정에서 max_wal_senders 값이 대기 서버의 연결수 만큼은 지정되어야한다.

대기 서버를 실행하려면, 먼저 운영서버의 베이스 백업을 준비해야한다. 이것에 대한 자세한 이야기는 8.3.2절에서 다룬다.

9.2.4. 대기 서버 구축하기

대기 서버를 구축하려면, 먼저 운영서버의 베이스 백업 자료를 복원해야한다(8.3.4절 참조). 다음 대기 서버의 데이터 클러스터 디렉토리 안에 recovery.conf 파일을 만들고, 그 파일 안에 standby_mode 값을 on으로 설정해서 restore_command 값으로 대기 서버로 보내진 WAL

세그먼트 파일을 대기 서버에서 사용할 수 있도록 가져오는 명령을 지정한다. 고가용성을 고려해서, 대기 서버가 여럿 있는 경우는 `recovery_target_timeline` 값으로 `latest`를 지정한다. 이렇게 해서, 대기 서버 가운데 몇 번 재실행 되었던 서버도 정상적으로 운영 서버의 작업 내용을 반영할 수 있도록 한다.

> 참고: 여기서 설명하는 구축 방법을 사용할 때는 pg_standby나 기타 다른 복제 툴들을 사용하면 안된다. `restore_command` 설정값으로 지정하는 명령은 작업할 파일이 없을 경우 즉시 종료될 수 있는 것을 사용해야한다. 서버 쪽에서 이 명령의 재실행 처리를 관리하기 때문이다. pg_standby 명령을 사용하려면, 9.4절을 참조하라.

스트리밍 복제 기능을 사용하려면, `primary_conninfo` 설정값을 지정해야한다. 이 값은 libpq 라이브러리에서 사용하는 데이터베이스 연결 문자열이다. 이 값에는 운영서버의 호스트 이름(또는 IP 주소)은 반드시 지정해야하며, 그 외 접속에 필요한 정보를 지정한다. 운영 서버에서 데이터베이스 접속을 계정 비밀번호를 입력해야 하면, 이곳에 그 비밀번호를 지정하거나 대기 서버를 실행하는 시스템 계정의 홈 디렉토리에 .pgpass 파일에 지정하면 된다.

만일 대기 서버가 고가용성을 고려해서 구축되는 경우 대기 서버의 서버 환경도 운영 서버와 똑같이 아카이브 모드로 운영 되어야하며 서버 접근 인증 설정도 운영 서버와 같아야한다. 왜냐하면, 운영 서버 장애가 발생하면 이 대기 서버가 운영 서버 역할을 해야하기 때문이다.

운영 서버에서 넘겨 받은 WAL 파일을 사용하면, archive cleanup command 설정으로 이미 대기 서버에 반영이 완료된 파일을 지워서, 디스크 공간을 확보 할 수도 있다. 이 설정값으로 pg_archivecleanup 명령을 이용하면, 보다 쉽게 이 작업을 할 수 있다. 이 명령에 대한 자세한 설명은 pg archivecleanup 명령 도움말을 참조하라. 반면, 대기 서버가 운영 서버의 백업 용도로 구성하면 운영 서버의 베이스 백업 이후로 생긴 모든 WAL 세그먼트 파일을 지우지 않는 것이 좋다. 설령 그 파일이 대기 서버로 모두 반영되었다 하더라도 어떤 방식으로 다시 복원 작업을 할지 모르기 때문에 모두 남겨둬라. 물론 베이스 백업이 바뀌었을 때 이 베이스 백업 이전 로그 파일들은 기록 보관용적인 자료가 아니라면 삭제해도 무방할 것이다.

위 내용을 참고해서 대기 서버로 구축하기 위한 서버의 `recovery.conf` 파일은 일반적으로 아래와 같은 내용으로 구성한다:

```
standby_mode = 'on'
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo password=foopass'
restore_command = 'cp /path/to/archive/%f %p'
archive_cleanup_command = 'pg_archivecleanup /path/to/archive %r'
```

대기 서버는 다중으로 구성할 수 있지만 스트리밍 복제 방식을 이용하면 운영 서버의 `max_wal_senders` 환경 설정값으로 이 대기 서버 모두가 동시에 접속 할 수 있도록 충분한 연결을 확보해야한다.

9.2.5. 스트리밍 복제

스트리밍 복제는 WAL 세그먼트 파일 전달 방식보다 운영 서버의 자료 상태를 거의 실시간으로 동기화 한다. 대기 서버는 운영 서버로 접속해서 WAL 레코드 단위로 그 정보를 가져오고 그것을 반영하기 때문이다. WAL 세그먼트 파일 전달 방식을 이용하면 운영 서버가 그 로그 파일을 다 채워서 다른 로그 파일을 쓰겠다고 하는 시점에 대기 서버로 옮겨지기 때문에 그 동안은 대기 서버와 운영 서버 사이의 자료 불일치가 일어난다.

스트리밍 복제 방식은 기본적으로 비동기식으로 이루어진다. (9.2.8절 참조) 그래서, 운영 서버와 대기 서버간의 자료 불일치가 약간 있을 수는 있다. 하지만 대기 서버와 운영 서버 사이의 네트워크 환경이 양호하고 대기 서버의 성능이 좋으면 파일 기반 로그 전달 방식과는 비교

도 안될 만큼 1초 미만의 지연시간이 생긴다. 운영 서버와 대기 서버가 같은 네트워크 대역대 안에 있으면 거의 실시간으로 동기화 된다. 스트리밍 복제 환경에서는 archive_timeout 설정값을 짧게 해서 자료 손실을 줄일 수 있다.

파일 기반 로그 전달 방식을 함께 사용하지 않고, 단지 스트리밍 복제 기능만 사용하면, 운영 서버에서 반드시 wal_keep_segments 환경설정값을 지정해서, 아직 대기 서버로 반영되지 못한 WAL 파일을 남겨두는 최대 개수를 지정해 주어야한다. 운영 서버에는 WAL 파일을 재활용하기 때문에 대기 서버의 중지 시간이 길어져 이 대기 서버로 반영되지 못한 WAL 파일이 계속 쌓이면 사용할 수 있는 디스크 공간이 점점 줄어들 것이다. 또한 대기 서버의 중지 시간이 아주 길어졌고 운영 서버쪽에서는 더 이상 로그를 보관할 수 없어 재활용 해 버렸으면 더 이상 동기화 할 자료를 찾지 못하기 때문에 다시 대기 서버를 구축 해야한다. 만일 운영 서버쪽에서 WAL 세그먼트 로그 파일을 대기 서버 쪽으로 보내는 설정을 해 두었고, 대기 서버 쪽에서 그 파일을 사용할 수 있는 환경이면, wal_keep_segments 설정은 굳이 필요가 없다. 대기 서버 쪽으로 처리되어야 할 WAL 로그 파일들은 로그 전달 설정을 통해서 이미 대기 서버 쪽으로 옮겨졌기 때문에 충분히 운영 서버의 자료와 동기화 할 수 있을 것이다.

이런 이유로, 스트리밍 복제 기능을 이용 할 때 먼저 대기 서버에서 파일 기반 로그 전달 기능도 함께 설정 하는 것이 좋다. 자세한 것은 9.2절에서 설명하고 있다. 파일 기반 로그 전달 방식에서 스트리밍 복제 방식으로 바꾸려면, recovery.conf 파일에서 primary_conninfo 설정값에 접속할 운영 서버의 정보를 지정하면 된다. 운영 서버에서는 대기 서버가 접속할 수 있도록 listen addresses 환경설정값이 바뀌어야하며, pg_hba.conf 파일에 대기 서버에서의 접속을 허용할 있도록 지정하며 그 때 사용할 데이터베이스 이름은 replication 으로 지정한다. 자세한 사항은 9.2.5.1절 을 참조하라.

또한 운영 서버 입장에서 클라이언트의 TCP 연결이 끊겼을 경우 빠르게 그 소켓을 정리해서, TCP 연결을 원할하게 하기 위해 tcp keepalives idle, tcp keepalives interval, tcp keepalives count 설정값을 조정할 수도 있다.

또한 대기 서버의 최대 동시 연결수도 지정해야한다. (max wal senders 환경설정값 항목을 참조).

대기서버는 먼저 운영 서버에서 넘겨 받은 WAL 세그먼트 파일을 모두 자신의 서버에 적용하면 primary_conninfo 설정값에 지정한 운영 서버로 접속한다. 이 접속이 성공하면 대기 서버는 walreceiver 프로세스를 만들어 운영 서버의 walsender 프로세스에서 보내는 트랜잭션 정보를 하나씩 받아서 자신의 서버에 적용한다.

9.2.5.1. 인증

WAL 스트리밍 정보는 보안상 슈퍼유저나 인증된 데이터베이스 사용자만 사용할 수 있어야 한다. 이 권한을 부여하려면, REPLICATION 권한을 부여한다. 그래서 대기 서버에서 운영 서버로 접속하는 데이터베이스 사용자는 LOGIN 권한과 REPLICATION 권한을 반드시 지정해야한다. 이 리플리케이션 전용 사용자는 운영 서버의 자료를 조작할 수 없도록 슈퍼유저가 아닌 별개의 사용자를 만들어 사용하는 것이 보안상 안전하다.

스트리밍 복제 기능을 이용하기 위한 운영 서버의 클라이언트 인증은 pg_hba.conf 파일에서 해당 데이터베이스 이름으로 replication 이라는 예약어를 지정해야한다. 예를 들어, 대기 서버의 IP가 192.168.1.100 이면, 운영 서버의 pg_hba.conf 파일은 다음과 같은 형식으로 지정한다:

```
# 아래 "foo" 는 스트리밍 복제를 위해 대기 서버가 사용할 운영 서버로 접속할
# 데이터베이스 사용자 이름이다. 인증 방식은 md5 비밀번호 인증을 사용한다.
# 그 대기 서버의 IP는 192.168.1.100 딱 하나다.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    replication     foo             192.168.1.100/32        md5
```

대기 서버 입장에서 접속할 운영 서버 접속 정보는 recovery.conf 파일에서 지정한다. 접속할 데이터베이스 사용자의 비밀번호는 이 파일에 지정해도 되고 대기 서버를 실행하는 시스템 사용자의 홈 디렉토리에 있는 ~/.pgpass 파일에서 지정해도 된다. (이 때 데이터베이스 이름으로 replication 예약어를 사용한다.) 예를 들어 운영 서버의 IP는 192.168.1.50, 포트는 5432, 데이터베이스 사용자는 foo, 그 사용자의 비밀번호는 foopass인 경우, 대기 서버에서 사용하는 recovery.conf 파일에 다음과 같이 지정한다:

```
# 이 대기 서버는 호스트 IP 192.168.1.50, 포트 5432 운영 서버로
# "foo" 사용자, "foopass" 비밀번호로 접속한다.
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo password=foopass'
```

9.2.5.2. 모니터링

복제 환경의 모니터링 가운데 가장 중요한 사항은 운영 서버에서 만들어내는 WAL 레코드들이 얼마나 잘 대기 서버로 반영 되고 있느냐를 살펴보는 것이다. 이 작업은 운영 서버의 현재 WAL 쓸 위치와 대기 서버의 마지막 처리된 WAL 쓸 위치 차이를 계산하는 방식을 이용하면 된다. 이 작업을 위해서 운영 서버에서는 pg_current_xlog_location, 대기 서버에서는 pg_last_xlog_receive_location 두 함수를 사용한다. 대기 서버에서 마지막 받은 WAL 위치는 OS의 프로세스를 확인하는 **ps** 명령으로 출력되는 WAL receiver 프로세스의 이름에서도 확인할 수 있다. (프로세스 모니터링에 대한 자세한 이야기는 11.1절을 참조)

또한 운영 서버에서 사용하는 WAL sender 프로세스의 정보는 뷰로 확인할 수 있다. 이 뷰에서 pg_current_xlog_location 값과 sent_location 값이 차이가 많이 나면 운영 서버가 많이 바쁜 상황이고, sent_location 값과 pg_last_xlog_receive_location 값이 차이가 많이 나면, 대기 서버가 많이 바쁘거나 네트워크 상황이 좋지 않은 경우다.

9.2.6. 리플리케이션 슬롯

리플리케이션 슬롯은 모든 대기 서버가 WAL 세그먼트를 받을 때까지 마스터 서버가 WAL 세그먼트를 삭제하지 않게 해주고, 대기 서버와의 연결이 끊어져도 마스터 서버가 로우를 보존하게 해서 복구 충돌이 발생하지 않게 해 주는 자동화 시스템이다.

리플리케이션 슬롯을 이용하는 대신 wal keep segments를 쓰거나, archive command를 사용할 때 아카이브에 세그먼트들을 저장해서 이전 WAL 세그먼트 삭제를 방지할 수 있다. 리플리케이션 슬롯은 필요한 세그먼트의 개수만큼만 보관하는 반면에 이 방법은 필요 이상의 WAL 세그먼트를 보관하는 경우가 종종 있다는 것이 단점이다. 장점은 pg_xlog에 필요한 공간을 남겨둔다는 것이다. 리플리케이션 슬롯에는 아직 이 기능이 없다.

유사한 방법으로, hot standby feedback 방식과 vacuum defer cleanup-age 방식은 vacuum이 삭제한 해당 로우를 보존하는데, hot standby feedback 방식은 대기 서버가 연결되어 있지 않을 때는 보존할 수 없고, vacuum defer cleanup age 방식은 큰 값으로 지정해서 보존한다. 리플리케이션 슬롯은 이 같은 제한이 없다.

9.2.6.1. 질의 수행과 리플리케이션 슬롯 처리

각 리플리케이션 슬롯에는 알파벳 소문자, 숫자, 언더스코어로 쓸 수 있는 이름이 있다.

리플리케이션 슬롯과 상태는 pg_replication_slots 뷰에서 확인할 수 있다.

슬롯은 스트리밍 리플리케이션 프로토콜이나 SQL 함수

9.2.6.2. 설정 예제

리플리케이션 슬롯은 다음과 같이 생성할 수 있다.

```
postgres=# SELECT * FROM pg_create_physical_replication_slot('node_a_slot');
  slot_name  | xlog_position
-------------+---------------
 node_a_slot |

postgres=# SELECT * FROM pg_replication_slots;
  slot_name  | slot_type | datoid | database | active | xmin | restart_lsn
-------------+-----------+--------+----------+--------+------+-------------
 node_a_slot | physical  |        |          | f      |      |
(1 row)
```

대기 서버가 슬롯을 쓰도록 설정할 때 대기 서버의 `recovery.conf` 파일에서 `primary_slot_name`을 설정해야 한다. 아래는 간단한 예제이다.

```
standby_mode = 'on'
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo password=foopass'
primary_slot_name = 'node_a_slot'
```

9.2.7. 순차적 복제

순차적 복제 cascading replication 기능은 대기 서버가 마치 이어 달리기처럼 다른 대기 서버의 스트리밍 복제를 할 수 있도록 하는 것이다. 이 기능은 스트리밍 복제에 필요한 운영 서버의 부하를 최소화할 수 있다.

대기 서버가 트랜잭션 로그를 받음과 동시에 또 다른 대기 서버로 보낼 수 있음을 의미한다. 이렇게 전달하는 방식을 순차적 복제라고 하며, 운영 서버에서 복제 정보를 받아오는 서버를 업스트림 서버라고 하고, 그 서버로부터 정보를 받아가는 서버를 다운스트림 서버라고 한다. 순차적 복제에서 다운스트림 서버수는 제한이 없다. 마치 기존 하나의 운영 서버에 여러 대의 대기 서버를 연결해서 복제를 구현 하는 것 같다.

순차적 복제에서는 운영서버에 만든 트랜잭션 로그 레코드를 전달하는 것뿐만 아니라, 아카이빙 모드로 보관된 로그들도 함께 전달한다. 그래서 업스트림 서버와의 연결이 끊기더라도, 자신의 트랜잭션 로그는 다운스트림 서버 쪽으로 모두 보낸다. (하위 노드 입장에서 보면 자신의 상위 노드가 대기 서버에서 운영 서버로 전환되어 새로운 트랜잭션 로그를 만들어도 여전히 복제를 할 수 있음을 뜻한다.)

순차적 복제 기능은 현재 비동기식으로 구현되었다. 동기식 복제 설정(9.2.8절 참조)을 하더라도 순차적 복제 기능을 이용하면, 그 설정은 무시된다.

대기 서버의 `hot_standby_feedback` 설정은 순차적 복제 환경에서도 그대로 작동 한다.

업스트림 대기 서버가 새로운 운영 서버가 된다면, 다운스트림 대기 서버의 `recovery_target_timeline` 설정값이 `'latest'`이면 새 운영 서버로부터 복제를 계속한다.

순차적 복제 기능을 사용하려면, 업스트림 서버 쪽에서는 기본적인 복제 기능을 위해 운영 서버 쪽에서 설정했던 것을 그대로 설정하며 (max wal senders,hot standby 설정값 조정하고, 호스트 기반 인증 설정을 한다), 다운스트림 대기 서버는 기존 대기서버 설정과 같이 하면서 `primary_conninfo` 설정에서 업스트림 대기 서버 정보를 지정하면 된다.

9.2.8. 동기식 복제

Agens SQL 스트리밍 복제 기능은 기본이 비동기식이다. 이 말은 운영 서버가 장애로 멈춰, 대기 서버가 운영 서버로 역할 전환 작업(failover, 장애처리)을 할 때, 운영 서버와 대기 서버 사이 자료 동기화에 지연이 있을 시 그 만큼의 자료를 잃어버린다는 뜻이다.

동기식 복제 방식은 하나의 트랜잭션을 대기 서버에 반영하고, 그 결과를 운영 서버가 확인하는 방식이다. 이 방식은 트랜잭션 커밋으로 제공하는 기본 자료 안정성을 더 확장한 것이다. 이것을 컴퓨터 과학 이론에서는 이중 복제라한다.

동기식 복제 기능을 이용하면, 운영 서버, 대기 서버 모두 트랜잭션이 트랜잭션 로그 파일에 기록되었을 경우에만 정상처리 되었다고 판단한다. 이렇게 하면, 운영 서버가 중지 되어도 대기 서버에서 자료 손실이 일어나지 않는다. 이렇게 해서, 자료의 안정성을 제공하지만, 대기 서버의 작업 완료 응답을 확인하는 작업까지 포함되어 운영 서버의 복제 기능을 이용하는 비용이 상대적으로 늘어난다. 직렬화 된 트랜잭션 전달 작업은 현재 전달할 트랜잭션의 응답을 받아야 다음 작업을 진행하기 때문에, 최소한 그 만큼의 지연시간이 생길 수 밖에 없다.

읽기 전용 트랜잭션과, 트랜잭션 롤백은 대기 서버의 응답을 받지 않는다. 최 상위 레벨의 트랜잭션에 대해서 응답을 확인하지, 그 하위 레벨 트랜잭션인 서브트랜잭션에 대해서는 응답을 확인하지 않는다. 대량 자료 등록이나, 인덱스 생성 작업 같이 시간이 많이 걸리는 트랜잭션에 대해서도 대기 서버의 마지막 커밋 메시지를 확인하지 않는다. 이중 커밋(2PC)에 대해서는 모든 작업에 대해서 확인한다.

9.2.8.1. 기본 환경설정

이미 복제 기능을 사용하고 있는데, 동기식 복제 방식으로 변경하려면, synchronous standby names 설정값을 비워두지 않으면 된다. `synchronous_commit` 설정값도 `on`이어야하지만, 이 값이 기본값이 때문에 실질적으로는 위 환경설정 파라미터만 지정하면 된다. (2.5.1절과 2.6.2절 참조) 동기식 복제 사용하는 대기 서버를 지정하면 그 대기 서버로 보내는 모든 커밋 작업은 대기 서버에서도 커밋 작업을 완료했다는 응답을 기다린다. 이 응답을 확인 해야 운영 서버의 다음 커밋 작업을 진행한다. `synchronous_standby_names` 설정값은 쉼표 구분 여러 개의 대기 서버 이름을 지정할 수 있으며, 여기 등록되지 않은 대기 서버는 동기식 복제를 하지 않는다. `synchronous_commit` 설정은 환경 설정 파일에서 변경할 수 있을 뿐만 아니라, 각 트랜잭션에 대한 각각의 내구성 보장 방법을 개별적으로 설정할 수 있도록 개별 사용자나 특정 데이터베이스나 특정 응용 프로그램에서 이 값을 필요할 때 변경할 수 있다.

커밋 레코드가 운영 서버 WAL 세그먼트 파일에 기록된 뒤에 그 레코드를 대기 서버로 보낸다. 대기 서버는 그 레코드를 자신의 WAL 세그먼트 파일에 기록한 뒤에 `wal_receiver_status_interval` 환경설정 값이 0으로 지정되어 있지 않으면 운영 서버로 작업을 완료했다고 메시지를 보낸다. 대기 서버 가운데 운영 서버에서 `synchronous_standby_names` 환경설정 값에 지정한 첫번째 대기 서버가 보내는 응답을 확인 해서 대기 서버에서 WAL 레코드 복제를 완벽하게 했다고 운영 서버는 판단한다. 여기서 언급한 환경설정 매개변수들이 대기 서버의 동기화 방식에 대한 설정에 관계된 것들이다. 동기식 복제 기능 설정을 할 때는 어느 환경 변수는 대기 서버에서 설정하고 어느 환경 변수는 운영 서버에서 설정하는지 잘 파악하고 있어야한다. 또한, 대기 서버로 등록되는 서버들은 반드시 운영 서버와 직접 연결된 서버들이여야한다. 대기 서버의 하위 대기 서버는 동기식 복제를 사용할 수 없다.

`synchronous_commit` 설정값을 `remote_write`로 하면, 대기 서버의 OS 차원에서 해당 자료가 디스크에 기록되었다는 것까지만 확인한다. 실제로 OS가 디스크 버퍼 내용을 물리적으로 디스크에 기록했다는 것에 대해서는 확인하지 않는다. 이 설정은 `on`으로 설정했을 때 보다는 자료를 안전하게 동기화 하지는 못한다. Agens SQL 서버 장애가 아니라, OS 장애가 생겼을 경우에는 대기서버 측 자료 손실이 발생할 수도 있기 때문이다. 하지만, 동기화 작업 시간은 그 만큼 줄어들기 때문에, 특수 환경에서는 유용하게 사용될 수도 있다. 시스템 전체적인

입장에서 본다면, 자료가 손실 되는 경우는 운영서버와, 대기 서버 모두 동시에 장애가 발생 될 경우에만 일어나기 때문에다.

동기식 복제 기능을 이용하는 경우 빠른 서버 중지 명령이 실행되면 운영 서버가 정상적인 빠른 서버 중지 작업을 진행하지만, 비동기식 복제 기능을 이용하는 경우는 대기 서버로 보내야할 WAL 레코드를 연결 되어 있는 모든 대기 서버로 모두 보내고 서버가 중지된다.

9.2.8.2. 성능을 고려한 계획

동기식 복제 기능을 이용할 때는 세심한 계획이 필요하다. 응용 프로그램들이 제 성능을 잘 낼 수 있도록 대기 서버들을 구축해야한다. 대기 서버 작업 완료 확인을 위한 지연 작업이 시스템 자원을 거의 사용하지는 않지만 그 만큼 트랜잭션의 잠금 현상이 일어난다. 결과적으로 세심한 계획 없이 무작정 구축한 동기식 복제 환경은 많은 클라이언트 접속이 있을 경우 심각하게 반응 시간이 느려지기도 한다.

Agens SQL에서는 응용 프로그램 개발자가 복제 작업에서의 커밋 트랜잭션 작업에 대한 운영 서버의 확인 수준을 결정할 수 있도록 제공한다. 이 수준 조정 작업은 시스템 전체를 대상으로 할 수도 있고 사용자 단위, 데이터베이스 연결 세션 단위, 심지어 개별 트랜잭션 단위로도 할 수 있다.

예를 들어 한 서비스에서 10% 정도는 고객의 중요한 세부 정보 변경 작업이고, 90%는 사용자가 채팅 정보와 같은 장애가 발생되어 자료 손실이 생겨도 치명적인 문제가 발생하지 않는 자료에 대한 작업으로 운영된다고 하자.

이 때, 이 10% 작업에 대해서만 동기식 방식을 이용하고 나머지는 비동기식으로 사용해서 운영 서버의 성능을 개선할 수 있다. 이것을 응용 프로그램측에서 지정해서 특정 업무 상태에서만 동기식 복제를 이용할 수 있다.

이와 함께 고려해야할 부분은 운영 서버와 대기 서버 사이 복제를 위해 사용되는 네트워크 사용량이다. 이 부분을 잘못 계산하면 운영 서버가 운영을 위해 사용해야 할 네트워크 자원을 복제 작업 비용으로 써버리는 경우가 생기고 이것은 곧바로 성능 저하를 발생시킨다.

9.2.8.3. 고가용성을 고려한 계획

운영 서버 환경 설정에서 `synchronous_commit` 값이 `on` 또는 `remote_write` 상태이면 운영 서버에서 일어나는 모든 커밋은 대기 서버의 응답이 있을 때까지 대기하는 과정을 거치게 된다. 운영 서버는 반드시 적어도 하나의 대기 서버에서는 커밋 완료 응답을 받아야 한다. 그렇지 않으면 운영 서버에서 일어나는 커밋은 무작정 기다리게 된다.

자료 손실을 막는 가장 좋은 방법은 적어도 하나 이상의 동기식 대기 서버가 장애로 멈추지 않게 만드는 것이다. 운영 서버의 환경설정에서 `synchronous_standby_names` 변수의 값으로 동기식 방식을 이용할 대기 서버들을 나열해서 이것을 구현할 수 있다. 이 변수에 나열된(쉼표로 구분) 첫번째 대기 서버가 응답이 없으면 다음 대기 서버의 응답을 기다린다.

대기 서버가 처음으로 실행되어 운영 서버로 접속하는 시점에는 대기 서버는 아직까지는 운영 서버와 동기화가 되지 않았을 것이다. 이 때 대기 서버는 운영 서버 동기화를 위해 더 이상 동기화를 할 필요가 없는 상태가 될 때까지 계속 스트리밍 방식으로 자료를 동기화 할 것이다. 이런 일련의 상태를 따라잡기(`catchup`) 상태라고 한다. 이 상태의 기간은 대기 서버와 운영 서버의 자료 동기화 간격(차이가 나는 WAL 레코드 수)을 좁혀서 최종적으로는 동기화 간격을 없게 만드는 시간이다. 달리 말하면 운영 서버에서는 반영되었으나 대기 서버에는 아직 반영되지 않은 WAL 레코드들을 모두 처리하는 시간이다. 이 따라잡기 시간은 대기 서버가 얼마동안 중지해 있었고, 그 사이 얼마나 많은 자료 변경 트랜잭션이 일어났냐에 따라 결정된다.

운영 서버에서는 커밋되었고, 운영 서버가 대기 서버의 커밋 완료 응답을 기다리고 있는 중에 운영 서버가 재실행되면 운영 서버가 재실행 복구 작업 시 그 트랜잭션은 대기 서버에서

커밋 완료 되었었다는 응답을 받았다고 간주하고 복구 작업을 진행한다. 이것은 모든 대기 서버가 그 커밋을 완료했는지 알 길이 없기 때문이다. 하지만 복구가 완료되어 운영서버가 정상적으로 실행되었고 어떤 대기 서버가 그 작업을 아직 하지 않았으면, 그 작업부터 진행할 것이기 때문에 자료 손실은 없을 것이다. 응용 프로그램 입장에서는 이런 상황이 발생했으면, - 커밋 명령이 성공적으로 끝나기 전에 데이터베이스 연결이 끊겼다면 - 재연결시 반드시 해당 작업으로 자료가 정상적으로 반영되었는지 확인해야할 필요가 있다.

`synchronous_standby_names` 환경 변수값에 등록된 모든 대기 서버들이 모두 커밋 완료 응답을 보내지 않아 운영 서버의 커밋 작업이 멈춰 있는 상태이면, 이 변수의 값을 빈문자열('')로 지정하고, 환경 설정을 다시 적용해서 운영 서버를 정상화 할 수 있다.

운영 서버와 대기 서버와의 연결이 끊겼고, 대기 서버를 운영 서버로 사용해야 할 상황이라면, 대기 서버들 가운데 운영 서버와 동기화가 잘 된 서버를 운영 서버로 선택하는 것은 관리자의 몫이다. (동기식 복제를 하는 대기 서버를 운영 서버로 채택하는 것이 제일 안전할 것이다.)

대기 서버의 트랜잭션 완료 응답을 기다리고 있는 중에 새로운 대기 서버를 만들어야할 상황이면, pg_start_backup(), pg_stop_backup() 함수를 사용해서 일반 베이스 백업을 만드는 방식으로 만들면 되지만, 이 때 이 작업을 하는 세션은 반드시 `synchronous_commit` = `off` 환경으로 작업을 진행해야한다. 그렇게 하지 않으면, 위 함수들 조차 대기 서버의 응답을 기다리게 되어 동기화 하고 있는 대기 서버의 문제가 풀릴 때까지 무한 대기 상태가 되어버린다.

9.3. 장애처리(failover)

운영 중인 서버가 장애로 멈췄다면, 대기 중인 서버를 운영 서버로 사용할 장애처리 작업을 해야한다. 이것을 장애처리(failover) 절체라고 한다.

대기 서버가 장애일 경우는 운영 서버에 영향을 주지 않기 때문에 이 작업이 필요 없다. 그냥 재실행 할 수 있으면 그렇게 하면 되고, 재구축이 필요하면, 대기 서버 구축 방법에 따라 진행하면 된다. 대기 서버는 실행 순서에 따라 자동으로 운영 서버와 동기화를 할 것이다.

장애처리 작업이 일어나서 새로운 운영 서버가 그 역할을 하기 시작했는데 장애가 생겼던 옛 운영 서버가 다시 실행되는 경우 클라이언트가 그 옛 운영 서버를 사용할 수 없도록 조치할 필요가 있다. 이것을 흔히 STONITH (Shoot The Other Node In The Head) 기능이라고 한다. 두 개의 운영 서버가 실행 되고 서로 자신이 운영 역할을 하기 때문에 심각한 경우엔 자료 손실이 일어난다. 이 문제를 어떻게 막을 것인지를 고려해야한다.

대부분의 장애처리 기반 시스템은 두 개의 시스템으로 구성된다. 이 시스템은 어떤 종류의 하트비트 메카니즘으로 주기적으로 장애처리를 해야하는지를 검사한다. 서로 하트비트 검사를 하는 방식 말고, 이 검사를 제 3의 서버(이것을 감시 서버라고 한다)를 맡아 보다 안전한 장애처리를 할 수 있도록 하기도 한다. 이 경우는 전자 방식보다 더 세밀한 테스트와 구축 비용이 더 많이 들것이다.

Agens SQL에서는 이 장애처리를 판단하고, 대기 서버에게 운영 서버 역할을 맡도록 알리는 시스템 프로그램을 제공하지는 않는다. 시스템 의존적인 작업이기 때문이기도 하고, 이미 이런 소프트웨어는 많이 있다.

운영, 대기 각각 하나의 서버로 구성했을 경우 대기 서버가 운영을 맡게 되면, 운영 서버를 위한 대기 서버가 없는 단독 운영 서버 환경이 된다. 원래 구축 하려고 했었던 가용성 높은 시스템 환경 구축이라는 목표에서 벗어나게 됨을 의미한다. 간단하게 생각하면 멈춘 옛 운영 서버를 대기 서버로 사용하면 된다 라고 하지만 실무에서는 대기 서버가 어떤 문제로 운영을 맡게 될지 아무도 모르기 때문에 이 생각은 위험하다. 만일 하드웨어 장애였으면 그것을 고치고, 대기 서버로 새로 구축해서 대기 서버로 실행해야하기 때문에, 꽤 많은 시간을 운영 서버

혼자 감당해야한다. 운영적인 입장에서는 이런 경우 어떻게 최소한의 비용으로 빠르게 대기 서버를 구축할 것인가도 준비해야한다. 비용이 넉넉하다면, 제 3의 대기 서버를 준비해 두었다가, 이런 상황에서 대기 서버로 작동할 수 있도록 한다.

결론적으로 대기 서버가 운영 역할을 맡는 작업은 비교적 짧은 시간 안에 일어나지만, 다시 원래 환경과 같은 환경을 만드는 작업에는 꽤 많은 시간이 걸린다. 또한 대기 서버가 그 역할을 제대로 할 수 있는지 주기적으로 확인하는 작업도 필요하다. 확인 작업은 장애처리에 필요한 모든 작업이 될 것이다. 운영 서버 장애를 감지할 수 있는지, 대기 서버는 잘 복제를 하고 있는지, 대기 서버가 운영 서버 역할을 맡고 별 이상이 없는지 등등이 될 것이다.

로그 전달 방식을 이용한 대기 서버환경에서 대기 서버가 운영 서버 역할을 하게 하려면, **pg_ctl promote** 명령을 실행하거나 `recovery.conf` 파일에 있는 `trigger_file` 설정값으로 지정한 파일을 만들어주면 된다. **pg_ctl promote** 명령을 사용할 경우에는 `trigger_file` 설정값이 지정되어 있지 않아도 된다. 고가용성을 고려한 구축이 아니라, 그냥 읽기 전용 쿼리를 사용하기 위해서 구축한 로그 전달 방식의 대기 서버인 경우 굳이 이 작업을 하지 않고, 그냥 운영 서버 장애를 직접 복구 하는 것이 나을 것이다.

9.4. 로그 전달 복제에 대한 다른 방법

운영 서버에서 사용했던 로그를 대기 서버로 보내서 복제 기능을 구현하는 방법은 `restore_command` 설정값으로 지정한 명령으로 전달 된 파일을 자기 서버에 적용하고, 적용할 파일이 없으면 그 명령을 계속 반복하는 식이다. 이 부분은 앞 장에서 구체적으로 설명했다. 여기서는 이 방법이 아닌 8.4 이하 버전에서도 사용할 수 있는 데이터베이스 복구 기능을 이용해서 독자적인 로그 전달 복제 기법을 설명한다. 이렇게 하려면 대기 모드로 실행하지 않고 단순 복구 모드로 실행하도록 `standby_mode` 설정값을 off로 지정해야한다. pg standby 모듈이 이 독자적인 복제 기능을 구현해 놓은 것이다.

이 방식에서 주의해야할 점은, 운영 서버에서 만들어진 WAL 파일은 그것들이 만들어지는 순서에 따라 그대로 대기 서버로 반영 되어야한다는 점이다. 또한 반영 하는 작업은 반드시 직렬화 되어서 한 번에 하나의 파일만 적용해야한다. 대기 서버에서 쿼리를 사용할 수 있게 설정 하면, (Hot Standby) WAL 파일 적용 간격에 차이 때문에 자료가 정확하게 운영 서버와 같지 않다는 점도 기억해야한다. 이 대기 서버로의 자료 반영 지연 현상을 줄이려면, `archive_timeout` 설정값을 짧게 하면 된다. 또한 이 방식으로 구현하면, 스트리밍 방식 복제를 사용할 수 없다는 점도 기억해야한다.

기본 작동 방식은 일반적인 로그 전달, 복구 방식과 같다. 트랜잭션 로그를 전달하는 방법은 WAL 레코드가 다 채워진 WAL 세그먼트 파일을 서로 전달하고, 받아서 사용하는 길 뿐이다. 운영 서버에서 만드는 순차적인 WAL 세그먼트 파일들을 대기 서버로 옮길 때 누락되는 일이 없도록, 그리고 나중에 만들어진 파일이 이전에 만들어진 파일보다 먼저 대기 서버에 도착하는 일이 없도록 해야한다. 또한 다른 운영 서버에서 만든 WAL 세그먼트 파일과 섞여 구분이 안되는 일이 없도록 해야한다. 대기 기능만 구현하고자 한다면, WAL 파일 전달량은 그렇게 많지는 않다.

복제 기능을 독자적으로 구현하는 열쇠는 `restore_command` 설정값에 있다. 이 값은 대기 서버가 실행할 때 사용하는 `recovery.conf` 파일 안에 `restore_command` 서버 환경 변수 매개변수 값으로 지정한다. 여기서 사용하는 명령은 서버가 처리해야하는 파일이 없으면 0 아닌 값을 리턴해서 복구 모드를 종료하고 일반적인 정상 운영 서버로 실행 되도록 한다. 하지만 복제용 대기 서버로 구축하려면 이 명령은 원하는 파일이 없을 때 그 파일이 생길 때까지(운영 서버가 대기 서버로 보내줄 때까지) 기다리고 있다가, 파일이 생겼을 때 다시 대기 서버가 복구 작업을 계속 할 수 있어야한다. 또한 `.backup`, `.history`이 운영 서버로부터 넘어오면 그 파일은 무시하고, 0 아닌 값을 리턴하면서 이 명령을 끝내야한다. 이렇게 설명한 모든 부분을 사용자가 직접 프로그래밍 해야한다. 이 프로그램은 아울러, 장애처리를 위한 운영 역할로 전환할 수 있는 기능도 있어야하며, OS 시그널 처리도 있어야 한다.

restore_command 설정값으로 사용될 프로그램의 의사코드는 다음과 같다:

```
triggered = false;
while (!NextWALFileReady() && !triggered)
{
    sleep(100000L);         /* wait for ~0.1 sec */
    if (CheckForExternalTrigger())
        triggered = true;
}
if (!triggered)
        CopyWALFileForRecovery();
```

이런 방식으로 구현된 실제 프로그램을 pg standby 모듈에서 제공한다. 이 모듈은 위에서 설명한 구현 요소들을 모두 구현했다. 이 모듈과 함께 사용자 정의 스크립트들을 사용해서 구축할 수도 있을 것이다.

어떻게 대기 서버가 운영 서버로 바뀔 것인가에 대한 설계는 꽤 중요한 부분이다. 이 또한 restore_command 설정값으로 지정하는 명령어에서 담당한다. 단순하게 이 명령어의 리턴값이 0 아닌 값으로 종료되면 된다. 0 아닌 값으로 종료되는 상황은 어떤 특정 파일이 생겼을 때, 또는 더 나아가 운영 서버가 응답이 없을 때 등등 발생할 수 있는 모든 상황을 꼼꼼히 살펴보고 그것을 구현하면 된다. 문제는 이 명령어 실행은 서버가 각 개별 WAL 파일에 대해서 한 번 처리하고 종료되고 다시 실행하는 식으로 작동되기 때문에 서버 데몬과 달리 시그널 처리를 하기 힘들다. 그렇기 때문에 어떤 특정 파일을 만들어 장애처리 신호로 사용할 때 그 파일은 다른 프로세스가 만드는 것이 타당하다. 한편, 운영 서버의 archive_timeout 값을 참조해서 원하는 WAL 파일이 넘어오지 않을 경우 다음 작업을 할 수 있는 시간 제한 기능을 둘 수도 있으나 이 때 네트워크 상태나 운영 서버의 부하 상태도 함께 고려해야한다. 그렇게 하지 않으면 의도치 않게 장애처리 기능이 작동해서 상황을 더 악화시킬 수도 있다.

9.4.1. 실행

대기 서버 설정은 간단한 절차를 통해서 할 수 있다. 절차들의 세부 내용은 각 참조하는 절을 참고하라.

1. 운영 서버와 대기 서버의 시스템을 최대한 똑같이 설정 하고, 각자 동일한 버전의 Agens SQL을 사용한다.
2. 운영 서버에서 대기 서버의 WAL 아카이브 디렉토리로 연속적인 아카이빙을 설정하라. archive mode, archive command, archive timeout 가 운영 서버에 적절하게 설정되도록 확인한다(8.3.1절을 참조).
3. 운영 서버의 기본 백업을 만들고(8.3.2절을 참조), 대기 서버에 데이터를 적재하라.
4. 이전에 설명한 것처럼(8.3.4절을 참조) restore_command가 명시된 recovery.conf로 지역 WAL 아카이브에서 대기 서버로의 복구 작업을 시작하라.

복구 작업은 WAL 아카이브를 읽기 전용으로 처리하여, WAL 파일이 대기 서버 시스템에 복사 되면 테이프로 WAL 파일이 복사되어 데이터베이스 대기 서버가 동시에 읽을 수 있다. 이와 같이 대기 서버는 재난 복구를 목적으로 파일을 장기간 저장하면서 동시에 고가용성을 구현할 수 있다.

테스트를 목적으로 운영 서버와 대기 서버를 같은 시스템에서 실행할 수 있다. 이 작업은 server robustness를 개선하지 않기 때문에 고가용성이라 하지 않는다.

9.4.2. 레코드 기반의 로그 전달

레코드 기반으로 로그를 전달하는 것도 가능한데, 사용자가 직접 개발을 해야 된다. 또, 전체 WAL 파일이 전달됐을 때에는 hot standby 쿼리에만 변경 사항이 보인다.

`pg_xlogfile_name_offset()` 함수 를 호출해서 파일명과 현재 마지막 WAL의 정확한 바이트 오프셋을 구할 수 있다. pg_xlogfile_name_offset 함수는 WAL 파일에 직접 접근할 수 있고 WAL 파일 마지막 데이터를 대기 서버에 복사할 수 있다. 이 방법을 쓰면 데이터 손실은 프로그램을 복사하는 폴링(polling) 순환 시기에 발생하는데, 손실은 매우 적고, 부분적으로 사용되는 세그먼트 파일을 강제로 아카이빙할 때 낭비하는 대역폭은 없다. 대기 서버의 `restore_command` 스크립트는 전체 WAL 파일을 다루므로, 점차적으로 복사된 데이터는 보통 대기 서버에서 사용할 수 없다. 이 데이터는 운영 서버가 죽었을 때만 사용되는데, 마지막 WAL 파일 부분이 대기 서버에 쓰이게 된다. 이 프로세스를 정확하게 실행하려면 데이터 복사 프로그램과 `restore_command` 스크립트의 조합이 필요하다.

Agens SQL 9.0 버전 이상은 스트리밍 리플리케이션(9.2.5절을 참조)으로 편리하게 할 수 있다.

9.5. 상시 대기

여기서 말하는 상시 대기 Hot Standby란 그 서버가 아카이브 파일로 복구 작업을 하고 있거나 대기 모드로 있을 때도 클라이언트가 그 서버로 접속할 수 있으며, 읽기 전용 쿼리를 실행할 수 있는 것을 말한다. 이 기능은 복제 기능을 구현하는 방법으로, 가장 최근 상태로 백업을 하는 방법으로 유용하다. 또한 상시 대기라는 용어는 그 대기 서버로 클라이언트들이 접속해 있는 상태에서 즉시 운영 서버로 변경해서 운영 서버에서 사용하던 쿼리를 사용할 수 있는 기능을 뜻한다.

상시 대기 모드에서 사용하는 쿼리도 크게 다르지는 않지만, 아래와 같이 몇가지 제약 사항이 있으며, 관리적인 측면에서 주의해야할 부분이 있다.

9.5.1. 사용자 측면 개요

대기 서버에서 hot standby 매개변수의 값이 true로 설정되면, 복구 작업으로 시스템이 일관성 있는 상태가 되었을 때 연결을 허용하게 된다. 모든 연결은 무조건 읽기 전용이다. 임시 테이블도 쓰기를 허용하지 않는다.

운영 서버에 있는 데이터는 대기 서버로 올 때까지 시간이 걸리므로 운영서버와 대기 서버 간 지연이 생긴다. 따라서 운영 서버와 대기 서버에 거의 동시에 같은 쿼리를 수행하는 것은 다른 결과를 가져올 수 있다. 대기 서버의 데이터가 운영 서버와 *결국 일치*할 것이라고 판단된다. 트랜잭션에 커밋된 레코드가 대기 서버에서 리플레이되면, 트랜잭션 변경 사항이 스냅샷에 보인다. 스냅샷은 각 쿼리 혹은 트랜잭션이 시작될 때 찍히는데, 현재 트랜잭션의 고립 수준에 따라 다르다.

hot standby에서 시작된 트랜잭션은 아래 명령어들을 수행할 수 있다.

- 쿼리 접근 - **SELECT**, **COPY TO**
- 커서 명령어 - **DECLARE**, **FETCH**, **CLOSE**
- 매개변수 - **SHOW**, **SET**, **RESET**
- 트랜잭션 관리 명령어
  - **BEGIN**, **END**, **ABORT**, **START TRANSACTION**

  - **SAVEPOINT**, **RELEASE**, **ROLLBACK TO SAVEPOINT**
  - **EXCEPTION** 블록과 그 외 내부적인 하위트랜잭션들
- **LOCK TABLE**(`ACCESS SHARE`, `ROW SHARE`, `ROW EXCLUSIVE` 중 한 모드에 있을 때)
- 계획과 리소스 - **PREPARE**, **EXECUTE**, **DEALLOCATE**, **DISCARD**
- 플러그인과 확장 모듈 - **LOAD**

hot standby에서 시작된 트랜잭션은 트랜잭션 ID를 할당 받지 못하고 시스템 WAL에 쓰기 작업을 할 수 없다. 아래를 수행하면 에러 메시지를 남길 것이다.

- 데이터 조작어 (DML) - **INSERT**, **UPDATE**, **DELETE**, **COPY FROM**, **TRUNCATE**. 복구하는 동안 트리거를 실행할 수는 없다. 임시 테이블도 마찬가지인데, 테이블 로우는 트랜잭션 ID 없이 읽거나 쓸 수 없기 때문이다. 이 기능은 아직 Hot Standby 환경에 구현되지 않았다.
- 데이터 정의 언어 (DDL) - **CREATE**, **DROP**, **ALTER**, **COMMENT**. 이 규칙은 임시 테이블에도 적용되는데, 이 작업들을 수행하려면 시스템 카탈로그 테이블도 수정해야 하기 때문이다.
- **SELECT ... FOR SHARE | UPDATE**, 로우 잠금을 하기 위해서 데이터 파일을 수정해야 할 때도 있기 때문이다.
- DML 명령어를 생성하는 **SELECT** 문에 대한 규칙
- `ROW EXCLUSIVE MODE`보다 높은 모드를 명시적으로 요청하는 **LOCK**.
- `ACCESS EXCLUSIVE MODE`를 요청하는 축약된 디폴트 형태의 **LOCK**.
- 읽기 전용이 아닌 상태를 명시적으로 설정한 트랜잭션 관리 명령어
  - **BEGIN READ WRITE**, **START TRANSACTION READ WRITE**
  - **SET TRANSACTION READ WRITE**, **SET SESSION CHARACTERISTICS AS TRANSACTION READ WRITE**
  - **SET transaction_read_only = off**
- 두 단계의 커밋 명령어 - **PREPARE TRANSACTION**, **COMMIT PREPARED**, **ROLLBACK PREPARED** 읽기전용의 트랜잭션도 준비 단계에서 WAL에 쓰기 작업을 해야 하기 때문이다(두 단계중 첫 번째 커밋 단계).
- 시퀀스 업데이트 - `nextval()`, `setval()`
- **LISTEN**, **UNLISTEN**, **NOTIFY**

보통은 "읽기 전용" 트랜잭션이 시퀀스를 수정하고 **LISTEN**, **UNLISTEN**, and **NOTIFY**를 사용하기 때문에 Hot Standby 세션들이 정상적인 읽기 전용 세션보다 조금 더 엄격하게 작업해야 한다. 추후 버전에선 이런 점이 완화될 수도 있다.

hot standby 시점에 `transaction_read_only` 매개변수는 항상 true로 지속될 수 있다. 데이터베이스를 수정하려는 시도가 없는 한 hot standby 연결은 다른 데이터베이스 연결과 별반 다르지 않을 것이다. 장애처리나 스위치오버가 발생하면 데이터베이스는 정상적인 프로세싱 모드로 전환된다. 세션은 서버 모드가 변경 돼도 연결된 상태를 유지한다. hot standby가 종료되면 읽기-쓰기 트랜잭션이 초기화될 수 있다(hot standby에서 시작된 세션조차도).

사용자는 **SHOW transaction_read_only**를 실행해서 읽기 전용 여부를 판별할 수 있다. 사용자는 함수 집합을 통해 대기 서버 정보에 접근할 수 있다. 함수들을 사용해서 현재 데이터베

이스와 연결된 프로그램을 쓸 수 있다. 복구 절차를 모니터링하거나 데이터베이스를 특정 상태로 복원하는 복잡한 프로그램을 쓸 수 있다.

9.5.2. 쿼리 충돌 처리하기

운영 서버와 대기 서버은 대부분 느슨한 연결 상태를 유지한다. 운영 서버에서 발생하는 트랜잭션 커밋은 대기서버가 그 트랜잭션을 정상적으로 커밋 했는지와 상관 없이 정상 처리된다. 이런 특성 때문에 이런 복제 기법에는 항상 예상치 않은 장애와 두 서버간 자료 충돌이 발생할 가능성을 내포하고 있다. 가장 대표적인 자료 충돌로는 성능 저하가 발생하는 것을 꼽을 수 있을 것이다: 예를 들어 운영 서버에서 많은 자료가 입력되고 있으면, 그로 인해 많은 트랜잭션 로그가 만들어 질 것이고, 이것을 대기 서버로 전달하기 위해 운영 서버는 단일 서버를 운영 할 때와 달리 부가적인 입출력 자원을 사용할 것이다. 또한 대기 서버에서도 실행 되고 있던 쿼리들이 이 입력 처리 때문에 발생하는 비용 때문에 영향을 받을 것이다.

다음은 대기 서버에서 발생할 수 있는 자료 충돌 종류들이다. 이런 충돌을 해결 하기 위해서는 어떤 경우는 실행 되고 있는 쿼리를 중지 하는 경우도 있고, 또는 세션을 중지해야 하는 경우도 발생할 것이다. 이런 의미에서 이런 충돌은 강한 충돌이라고 한다. 사용자는 이런 충돌을 잘 이해해서 각 상황에 맞는 조치를 취해야 한다:

- 운영 서버에서 **LOCK** 명령을 사용했거나, 다양한 DDL 구문에 의해서 발생하는 배타적 접근 잠금에 대해서, 대기 서버는 그것에 대한 정확한 상태를 유지하지 못한다. 즉, 운영 서버에서 해당 객체가 잠겨도 대기 서버에서는 접근이 가능하다. (이렇게 되면, 대기 서버에서 읽어서 운영 서버로 반영하는 자료들이 그 정합성을 잃을 수 있다.)
- 운영 서버에서 테이블 스페이스가 삭제 되고 있는 중에도 대기 서버는 그 테이블 스페이스에 속한 객체들을 접근할 수 있다.
- 대기 서버에 해당 데이터베이스를 사용하는 세션이 있음에도 불구하고, 운영 서버에서 해당 데이터베이스를 삭제 할 수 있다.
- 대기 서버에서 발생한 트랜잭션을 위해 보관 되어야 할 옛 버전 자료들이 운영 서버의 vacuum 작업으로 삭제 되어 버리는 일이 생길 수 있다.
- 위 경우와 반대로 보이지 말아야 할 옛 버전 자료들이 보여지는 경우도 발생할 수 있다.

단독으로 운영 되는 서버라면, 이런 쿼리 충돌이 일어나지 않는다. 왜냐하면, 한 작업이 마무리 되기 전까지 그 작업과 관련된 다른 세션들의 작업들은 대기 상태가 되기 때문이다. 하지만 복제 환경에서 대기 서버는 이런 대기를 제어 할 수 없다: 운영 서버에서 발생한 트랜잭션 로그를 대기 서버가 반영 하도록 전해 받았다면, 그저 그 작업을 진행할 뿐이기 때문이다. 이 작업을 진행 할 때, 운영 서버의 다른 세션 상태를 고려할 수 없기 때문이다. 물론 가장 완벽한 방법은 대기 서버까지도 모두 해당 트랜잭션 작업이 끝날 때까지 모든 세션(운영 서버의 세션과 대기 서버의 세션 모두)의 대기 상태를 유지 하는 것이지만, 현실적으로 대기 서버와 운영 서버간의 연결을 완벽하게 보장한다는 것은 거의 불가능하기 때문에, 이런 방식은 오히려 운영 서버의 안정성을 더 떨어뜨린다. 그래서, 이런 충돌이 발생했을 때 대기 서버는 충돌을 해결 하기 위한 별도의 처리 방식을 제공하고 있어야 하며, 이에 따라 몇가지 충돌 자동 해결 기능을 제공하고 있다.

그 한 예로 운영 서버에서 **DROP TABLE** 명령으로 한 테이블을 삭제 하면, 대기 서버에서 아직 삭제 되지 않은 그 테이블을 조회하는 쿼리는 운영 서버에서 전달 받은 이 트랜잭션이 감지 됐을 때 자동으로 취소 된다. 일반적으로 단독 서버 환경일 때, 조회 작업이 먼저 진행 중이고, 삭제 작업이 발생 하면, 조회 작업이 끝날 동안 삭제 작업은 기다린다. 하지만 대기 서버가 있는 상황에서는 운영 서버가 대기 서버에서 해당 테이블을 조회하는 작업이 있는지 알 수 없으므로 일단 삭제하고, 그 로그를 대기 서버로 보낸다. 이때 대기 서버의 가장 바람직한 선택

은 대기 서버에서 조회 하고 있던 쿼리를 자동으로 취소하고, 최대한 빨리 운영 서버에서 실행 했던 삭제 작업을 대기 서버에도 반영 하는 것이다. 이렇게 처리하지 않고 운영 서버처럼 조회 작업이 끝날 때까지 기다렸다가 삭제 작업을 진행한다면, 오히려 자료 동기화 (복제 작업) 의 근본 목적을 달성하는데 방해가 될 것이다.

달리 생각하면, 해당 충돌이 순식간에 끝나는 경우라면, 충돌을 피하기 위해 그냥 대기 서버도 운영 서버 방식을 취하는 것이 바람직할 것이다. 하지만, 대기 서버의 조회 작업이 오래 지속된다면, 트랜잭션 로그 동기화는 그만큼 지연 될 것이고, 이에 따른 다른 객체들까지 영향을 받을 것이다. 그래서, max-standby-archive-delay, max-standby-streaming-delay 환경 설정 매개변수로 이런 최대 지연 시간을 지정해서 이 지연 시간을 넘기면 대기 서버의 작업을 중지하도록 설정 한다. 이렇게 함으로써 최대한 대기 서버의 상태를 운영 서버와 같도록 한다. 이 두 환경 설정 매개변수는 아카이브 로그 파일 반영시 스트리밍 트랜잭션 로그 전달에서 최대 지연 시간을 지정하는 것이다. (이 설정값들의 초기값은 30초다. 이 값의 최적값을 찾는 것은 데이터베이스 관리자의 몫인 것 같다.)

고가용성이 주 목적인 대기 서버는 쿼리 지연으로 인해 운영 서버에 뒤쳐지지 않도록 지연 매개변수를 비교적 짧게 정하는 것이 가장 좋다. 그러나 대기 서버가 장시간 쿼리를 수행하게 된다면 지연 값을 크게 혹은 무한으로 설정하는 것이 좋다. 장시간 수행 쿼리가 WAL 레코드 적용을 지연시키면 다른 세션들이 운영 서버의 최근 변경 사항을 보지 못할 수도 있다.

`max_standby_archive_delay` 혹은 `max_standby_streaming_delay` 로 지정된 시간을 초과하면 충돌 쿼리는 취소된다. 그러면 취소 에러만 발생하는데, **DROP DATABASE**를 실행할 경우 전체 충돌 세션이 종료된다. 휴지 상태인 트랜잭션의 잠금이 충돌 원인일 때 충돌 세션은 종료된다(이 방법은 향후 바뀔 수 있음).

취소된 쿼리는 바로 재시도한다(새 트랜잭션을 시작한 이후에만). 취소된 쿼리는 리플레이된 WAL 레코드에 의존하므로, 취소된 쿼리가 다시 실행되면 성공할 확률이 높다.

지연 매개변수는 WAL 데이터를 대기 서버가 수신한 이후 경과 시간을 뜻한다는 것을 기억해 두자. 대기 서버에 있는 쿼리에 대한 유예 기간은 지연 매개변수를 초과할 수 없다. 이전 쿼리들이 완료되기를 기다리다가 혹은 업데이트 양이 너무 많아서 이미 뒤쳐진 상태이면 지연 매개변수보다 훨씬 작을 수도 있다.

대기 서버 쿼리와 WAL 리플레이 충돌의 원인으로 가장 잘 알려진 것은 "조기 청소early cleanup"이다. 보통 Agens SQL에서는 MVCC 규칙에 따라 데이터의 가시성을 정하기 위해, 트랜잭션이 보지 않는 이전 로우 버전들을 청소할 수 있다. 하지만 이 규칙은 마스터에서 수행하는 트랜잭션에만 해당된다. 그러므로 마스터에서 청소하면 대기서버의 트랜잭션이 보고 있는 로우 버전들이 삭제될 수도 있다.

숙련된 사용자는 로우 버전 청소와 로우 버전 동결(freezing)이 대기 서버 쿼리들과 충돌할 수 있다는 것을 안다. **VACUUM FREEZE**를 수동적으로 실행하면 로우가 업데이트되거나 삭제되지 않은 테이블에도 충돌을 일으킬 수 있다.

운영 서버에서 정기적으로 대량 업데이트 된 테이블은 대기 서버에서 장시간 수행되는 쿼리를 급하게 취소할 수 있다. 이런 경우에 `max_standby_archive_delay`나 `max_standby_streaming_delay`에 한정적인 값을 설정하면 `statement_timeout`을 설정하는 것과 비슷한 역할을 한다.

대기 서버 쿼리를 취소하는 횟수가 많아질 때를 대비하는 방법도 있다. 첫 번째 방법은 `hot_standby_feedback`매개변수를 설정하는 것인데, 최근 삭제된 로우들에 **VACUUM**을 수행하는 것을 막아서 청소 충돌을 방지하는 것이다. 운영 서버에서 삭제된 로우들에 대한 청소를 지연하여 테이블이 원치 않게 커질 수가 있다. 하지만 운영 서버에서 직접 대기 서버 쿼리를 실행하고 있을 때 대기 서버에서도 수행되는 경우보다는 나을 수도 있다. 이 경우, 지연된 WAL 파일에 대기 서버 쿼리와 충돌할 엔트리가 포함됐을 수 있으므로, `max_standby_archive_delay`가 항상 큰 값으로 유지되어야 한다.

또 다른 방법은 운영 서버에서 vacuum-defer-cleanup-age 를 증가시켜서 삭제된 로우가 평소보다 빨리 청소되는 것을 막는 것이다. `max_standby_streaming_delay`를 높게 설정하지 않

아도 대기 서버에서 쿼리들이 취소되기 전에 수행될 시간을 더 준다. 하지만 특정 수행 시간을 보장할 수는 없는데, 운영 서버에서 실행된 트랜잭션에 vacuum_defer_cleanup_age가 반영되기 때문이다.

쿼리 취소 횟수와 취소 원인은 대기 서버의 pg_stat_database_conflicts 시스템 뷰로 확인할 수 있다. pg_stat_database 시스템 뷰에는 요약 정보도 포함돼 있다.

9.5.3. 관리자 측면 개요

agens_sql.conf에서 hot_standby 상태가 on이고, recovery.conf 파일이 존재하면, 서버는 Hot Standby 모드로 실행될 것이다. 그러나 Hot Standby 접속이 허용되려면 시간이 꽤 걸릴 수도 있다. 서버가 일관적으로 어떤 쿼리를 실행하기 위해 복구를 끝낼 때까지 연결을 수락하지 않기 때문이다. 이 시간 동안에 연결을 시도하는 클라이언트는 에러 메시지가 뜨면서 거부될 것이다. 서버가 연결할 준비가 되었는지 확인하기 위해서는 애플리케이션 접속을 시도하는 루프를 돌거나 서버 로그에서 아래 메시지들을 찾아 볼 수 있다.

```
LOG:  entering standby mode

... then some time later ...

LOG:  consistent recovery state reached
LOG:  database system is ready to accept read only connections
```

일관성에 대한 정보는 운영 서버에 체크 포인트당 한번 기록된다. wal_level이 운영 서버의 hot_standby로 설정돼 있지 않을 때 쓰여진 WAL을 읽으면 hot standby를 켤 수 없다. 일관적인 상태는 아래 두 조건이 충족되면 지연될 수 있다.

- 64개 이상의 부트랜잭션을 갖는 쓰기 트랜잭션이 있을 때
- 매우 수명이 긴 트랜잭션이 있을 때

파일 기반의 로그 전달("warm standby")을 수행할 때 그 다음 WAL 파일이 도착할 때까지 기다려야 할 수도 있는데, 운영 서버에서 archive_timeout에 설정해 놓은 만큼 기다릴 수 있다.

대기 서버에서 설정한 일부 매개 변수들은 운영 서버에서 변경 되면 재설정되어야 한다. 이 매개변수들에 대한 대기 서버의 값은 운영 서버에서의 값보다 크거나 같다. 이 매개변수들이 높게 설정돼 있지 않으면 대기 서버에서 실행할 수 없다. 높게 설정되면 서버가 복구하기 위해 재시작한다. 매개변수들은 다음과 같다.

- max_connections
- max_prepared_transactions
- max_locks_per_transaction

관리자가 max-standby-archive-delay와 max-standby-streaming-delay에 적절한 설정을 해주는 것이 중요하다. 최적의 설정 방법은 비즈니스 우선순위에 따라 다르다. 예를 들어, 서버가 주로 고 가용성 서버로서의 임무를 맡으면 지연 값을 낮추는 것이 좋다. 극단적으로 0도 가능하다. 대기 서버가 decision support 쿼리를 위한 추가적인 서버일 경우, 최대 지연 값을 장시간 혹은 쿼리가 끝날 때까지를 의미하는 -1로 설정할 수 있다.

운영 서버에 쓰인 "hint bits" 트랜잭션 상태는 WAL로그에 기록되지 않으므로 대기 서버의 데이터가 힌트를 재사용할 수도 있다. 그러므로 대기 서버는 모든 사용자가 읽기 권한만 있어도 디스크 쓰기를 수행할 것이다. 데이터 값 자체에는 어떤 변화도 생기지 않는다. 사용자는

대용량의 정렬 임시 파일을 만들 수 있고, relcache 정보 파일을 재생성해서 hot standby mode일 때 데이터베이스의 어떤 부분도 읽기 전용이 아닌 상태로 만들 수 있다. dblink 모듈을 사용해서 원격 데이터 베이스에 쓰기작업을 하고 PL함수를 사용해서 데이터베이스 외부에서 다른 작업을 하는 것은 트랜잭션이 지역적으로 읽기 전용이라 하더라도 가능하다.

아래 관리 명령어 타입들은 복구 모드일 때는 허용되지 않는다.

- Data Definition Language (DDL) - e.g. **CREATE INDEX**
- Privilege and Ownership - **GRANT**, **REVOKE**, **REASSIGN**
- Maintenance commands - **ANALYZE**, **VACUUM**, **CLUSTER**, **REINDEX**

몇 개의 명령어들은 운영 서버의 읽기 전용인 트랜잭션에만 해당된다는 것을 알아 두자.

결론적으로 대기서버에 단독으로 존재하는 추가 인덱스들 혹은 통계치를 생성할 수는 없다. 위의 관리 명령어들이 필요할 때는 운영 서버에서 실행해야 하고, 변경 사항이 생기면 대기 서버에 전달 된다.

`pg_cancel_backend()`와 `pg_terminate_backend()`는 사용자 백엔드에서 작동하지만, 복구 작업을 하는 스타트업Startup 프로세스에서는 작동하지 않는다. pg_stat_activity 는 스타트업 프로세스에 대한 엔트리를 보여주지 않고 활성화된 복구 트랜잭션은 보여주지 않는다. 결론적으로, pg_prepared_xacts 는 복구 중에 항상 비어 있다. prepared transaction을 분석하고 싶으면, 운영 서버에 있는 `pg_prepared_xacts`을 보고 명령어를 실행해서 트랜잭션을 분석하자.

pg_locks는 백엔드가 사용하는 잠금을 보여준다. pg_locks also shows a virtual transaction managed by the Startup process that owns all `AccessExclusiveLocks` held by transactions being replayed by recovery. 스타트업 프로세스는 데이터베이스를 변경할 때 잠금을 사용하지 않으므로 `AccessExclusiveLocks` 외의 잠금은 pg_locks에 표시되지 않는다. 잠금이 존재한다고 추정될 뿐이다.

Nagios 플러그인 check_pgsql 은 간단한 정보를 확인하기 때문에 작동한다. check_postgres 모니터링 스크립트는 일부 수치가 예상과 다르거나 혼동되기도 하지만 작동은 한다. 예를 들어 vacuum은 대기 서버에서 작동하지 않으므로, 최근 vacuum 시간은 보존되지 않는다. 운영 서버에서 실행되는 vacuum은 대기 서버에 변경 사항을 전송한다.

WAL 파일을 조절하는 명령어는 복구 중에 작동하지 않는데, 예를 들면 `pg_start_backup`, `pg_switch_xlog` 등이 있다.

동적으로 적재 가능한 모듈은 pg_stat_statements를 포함해서 작동한다..

교착상태 감지와 같은 보조 잠금advisory lock은 복구 중에 정상적으로 작동한다. 보조 잠금은 WAL에 기록되지 않으므로, 운영 서버나 대기 서버에 있는 보조 잠금이 WAL 리플레이와 충돌할 수는 없다. 운영 서버에서 보조 잠금을 획득하고 대기 서버에서 비슷한 보조 잠금을 만드는 것도 불가능하다. 보조 잠금이 획득된 서버에서만 보조 잠금을 쓸 수 있다.

Slony나 Londiste, Bucardo같은 트리거 기반의 리플리케이션 시스템은 대기 서버에서 절대 실행될 수 없다. 대기 서버에 변경 사항이 전달되지 않는 한 운영 서버에서 잘 실행될 것이다. WAL 리플레이는 트리거 기반이 아니므로, 대기 서버에서 추가적인 데이터베이스 쓰기 작업을 요구하거나 트리거 쓰임에 의존하는 시스템에는 전달할 수 없다.

새로운 OID들은 할당되지 않으나, UUID 생성자generator는 데이터베이스에 새로운 상태를 쓰지 않는 한 작동한다.

현재 임시 테이블은 읽기 전용 트랜잭션 수행 중에 생성될 수 없으므로, 기존 스크립트들이 제대로 작동하지 않을 수도 있다. 이 제약은 향후 출시 버전에서 개선될 수 있다. 현재는 SQL 표준 준수와 기술적인 문제가 결합되어 있다.

DROP TABLESPACE는 테이블스페이스가 비어있을 때만 가능하다. 일부 대기 서버 사용자들은 `temp_tablespaces`로 테이블스페이스를 자주 사용할 수도 있다. 테이블스페이스에 임시 파일이 있으면, 임시 파일을 제거하기 위해 실행 중인 쿼리들이 모두 취소된다. 그러면 테이블스페이스가 제거되어 WAL 리플레이가 계속 진행될 수 있다.

운영 서버에서 **DROP DATABASE** 혹은 **ALTER DATABASE ... SET TABLESPACE**를 실행하면 대기 서버에서 이 데이터베이스에 연결된 모든 사용자들의 연결을 강제로 끊는 WAL 엔트리가 생성된다. 이는 `max_standby_streaming_delay`와 상관 없이 바로 발생한다. **ALTER DATABASE ... RENAME**은 사용자와의 연결을 끊지 않는다. 데이터베이스명에 의존하는 프로그램의 경우에는 혼동이 생길 수 있다.

정상(미복원) 모드에서 사용자가 연결된 상태에서, 그 사용자에게 로그인 기능이 있는 role에 대해 **DROP USER**나 **DROP ROLE**을 실행하면 그 사용자에게는 아무 일도 일어나지 않는다. 사용자의 연결 상태는 유지된다. 그러나 재연결은 불가능하다. 복구 시에도 마찬가지로, 운영 서버에서 **DROP USER**를 실행해도 대기 서버 사용자의 연결을 끊어지지 않는다.

statistics collector은 복구 중에 활성화 된다. 모든 스캔, 읽기, 블록, 인덱스 사용 등은 대기 서버에 정상적으로 기록된다. 리플레이 과정은 운영 서버에서 반복되지 않으므로 insert를 리플레이 해도 pg_stat_user_tables의 Inserts 컬럼을 증가시키지 않는다. stats 파일은 복구 초기에 삭제되므로 운영 서버와 대기 서버의 stats은 다르다. 이것은 기능일 뿐 버그가 아니다.

Autovacuum은 복구 중에 비활성화 된다. 복구가 끝나면 정상적으로 작동한다.

background writer는 복구 중에 활성화 되고, restartpoint를 수행 하고(운영 서버의 체크포인트와 비슷함), 정상적인 블록 청소 활동을 시작한다. 또 대기 서버에 저장된 hint bit 정보에 대한 업데이트도 한다. **CHECKPOINT** 명령어는 복구 중에 수행되는데, 새로운 체크포인트가 아닌 restartpoint를 수행한다.

9.5.4. 상시 대기 관련 환경 설정 매개변수 설명

9.5.2절과 9.5.3절에서 다양한 매개변수가 언급되었다.

운영 서버에서 wal-level과 vacuum-defer-cleanup-age를 사용할 수 있다. max-standby-archive-delay와 max-standby-streaming-delay는 운영 서버에서 설정돼도 아무 영향이 없다.

대기 서버에서 hot-standby와 max-standby-archive-delay, max-standby-streaming-delay를 사용할 수 있다. vacuum-defer-cleanup-age는 대기 서버 모드가 아닌 이상 아무 영향이 없고, 대기 서버가 운영 서버가 되면 영향을 미칠 수 있다.

9.5.5. 주의사항

Hot Standby에는 몇 가지 제약 사항이 있다. 이는 향후 출시 버전에서 개선될 수 있을 것이다.

- 해쉬 인덱스에 대한 작업은 WAL에 기록되지 않으므로 재실행하면 이 인덱스들은 업데이트되지 않는다.
- 트랜잭션을 실행하는 것에 대한 모든 정보는 스냅샷을 찍기 전에 필요하다. 부트랜잭션들을 많이 사용하는(현재는 64보다 많이 사용하는 경우를 가리킴) 트랜잭션은 가장 오래 걸리는 쓰기 트랜잭션이 끝날 때까지 읽기 전용 연결 시작을 늦출 것이다. 이런 상황이 발생할 때는 서버 로그에 메시지가 전달된다.

- 대기 서버 쿼리에서 유효한 starting point는 마스터 서버의 각 체크포인트에서 생성된다. 마스터 서버가 셧다운shutdown 상태일 때 대기 서버가 셧다운 되면, 운영 서버가 WAL의 starting point를 만들어야 하므로 Hot Standby에 재진입하는 것이 불가능 할 수도 있다. 이런 상황은 별로 문제 되지 않는다. 운영 서버가 셧다운되어 사용 불가능한 경우는 보통 대기 서버가 새로운 운영 서버로 바뀌는 데 실패했기 때문이다. 운영 서버를 의도적으로 끈 경우, 대기 서버를 자연스럽게 새로운 운영 서버로 만드는 것도 표준 절차이다.
- 복구 마지막 단계에서 prepared transaction이 사용한 `AccessExclusiveLocks`은 평소 잠금 테이블 엔트리 수의 두 배가 필요하다. `AccessExclusiveLocks`이 필요한 동시적 prepared transaction을 여러 개 수행하거나 여러 `AccessExclusiveLocks`이 필요한 transaction을 수행하면 `max_locks_per_transaction`를 크게 설정하는 것이 좋다. 운영 서버의 `max_locks_per_transaction`의 두 배도 좋다. `max_prepared_transactions`의 값이 0이면 이를 전혀 고려하지 않아도 된다.
- 직렬형의 트랜잭션 고립 수준(Serializable transaction isolation level)은 hot standby에서 아직 사용할 수 없다. 트랜잭션을 hot standby 모드에서 serializable isolation level로 설정하면 에러가 발생한다.

10장. 복구 환경설정

이 장에서는 recovery.conf 파일의 설정 방법에 대해서 설명한다. 여기서 설명하는 모든 것은 복구 작업에서만 적용되는 부분이다. 모든 설정은 복구 작업이 진행 된 뒤에는 변경해도 반영되지 않는다. 이 설정값들을 바꾸고 그것이 적용되길 원한다면, 다시 처음부터 복구 작업을 해야한다.

recovery.conf 파일에서 사용하는 구문은 설정항목 = '설정값' 형태로 한 줄에 하나씩 나열된다. 줄에 해시 기호(#)가 있으면, 그 뒤부터는 주석처리 된다. 설정값 안에 작은 따옴표가 필요하면, 작은 따옴표 두 개(")를 사용한다.

미리 제공되는 샘플 파일은 데이터베이스 서버가 설치된 디렉토리 안에 있는 share/recovery.conf.sample 파일이다.

10.1. 아카이브 복구 설정

restore_command (문자열)

복구 과정에서 사용할 WAL 세그먼트 파일들을 서버에 적용시킬 시스템 명령이다. 아카이브 모드 복구 작업을 하는 경우에 이 매개변수는 반드시 지정해야 한다. 스트리밍 리플리케이션 대기 서버를 구축하는 경우라면 이 설정이 꼭 필요한 것은 아니지만, 마스터 서버와의 통신이 끊겨서 수동으로 트랜잭션 로그를 반영해야하는 경우를 대비해서 설정해 두는 것이 좋다. %f 예약어는 적용 할 WAL 파일 이름으로, %p 예약어는 서버에서 사용할 WAL 파일 경로로 사용된다. (상대 경로를 사용하면, 서버를 실행하는 현재 디렉토리를 기준으로 처리된다.) %r 예약어는 마지막 실행 가능한 위치 정보를 나타내는 이름으로 처리된다. %r 예약어는 warm-standby 설정에서만 사용된다. (9.2절 참조). % 문자를 사용하려면, %% 를 사용한다.

여기서 중요한 사항은 여기서 지정한 작업이 성공했을 경우에는 반드시 그 명령의 리턴값이 0(zero)이어야 하며, 실패했을 경우에는 0이 아니여야한다. 서버가 이 명령을 통해 보관하고 있지 않는 파일을 사용하려고 할 때, 이 명령은 반드시 리턴 값으로 0 아닌 값으로 종료되어야한다. 다음은 일반적인 사례다:

```
restore_command = 'cp /mnt/server/archivedir/%f "%p"'
restore_command = 'copy "C:\\server\\archivedir\\%f" "%p"'  # Windows
```

archive_cleanup_command (문자열)

이 설정은 선택 항목이다. 여기서 지정한 명령은 warm-standby 서버가 중지 되었다가 다시 실행 될 때 사용된다. 이미 대기 서버에 반영 되어 더 이상 쓸모 없는 WAL 세그먼트 파일들을 찾아서 지우는 일을 할 수 있도록 지정한다. 이렇게 해서, 대기 서버가 빠르게 대기 상황이 될 수 있도록 한다. 이 때 주의할 점은 대기 서버가 여러 대가 있을 경우, 그 서버들이 같은 백업 WAL 파일을 사용할 경우, 이 명령이 실행되어 미처 반영되지 못한 파일이 지워지는 경우를 막아야한다. %r 예약어는 최종 반영된 위치로 처리된다. 즉, %r 이름보다 이전 파일들은 지워도 되지만, 그 이후 파일들은 지우지 말아야한다. 이 작업을 쉽게 하기 위해 이 설정값에는 일반적으로 pg archivecleanup 확장 모듈로 설치되는 명령을 사용한다. (이 명령은 파일을 지우는 작업을 하기 때문에 단독 대기 서버 환경일 경우에만 사용하는 것이 좋다.) 사용법은 다음과 같다:

```
archive_cleanup_command = 'pg_archivecleanup /mnt/server/archivedir %r'
```

다중 대기 서버 환경이라면, 이 WAL 정리 작업은 좀 더 복잡해지는데, 이 부분은 관리자에게 맡긴다. 이 설정값은 warm-standby 서버 구축 환경에서만 사용된다. (9.2절 참조). % 문자를 사용하려면, %%를 사용한다.

이 명령이 0 아닌 값을 리턴값으로 종료되면, 서버 로그에 WARNING으로 기록된다.

`recovery_end_command` (문자열)

복구 작업이 완료되면 딱 한 번 여기서 지정한 명령이 실행된다. 이 설정값도 선택적으로 지정할 수 있다. 여기서 지정한 명령은 아카이브 모드 백업을 이용한 운영 서버 복구 때나 대기 서버가 운영 서버로 바뀌는 시점이나 스트리밍 리플리케이션 서버에서 마스터 서버와의 스트리밍 통신이 끊겨 백업 파일을 다시 참조해서 모두 반영한 경우에 실행된다. `%r` 예약어는 마지막 반영된 위치이다. 이것은 archive-cleanup-command 명령에서 사용된다.

이 작업이 0 아닌 값으로 종료되면 데이터베이스 서버 로그에 WARNING 레벨로 오류 메시지가 기록되며, 서버는 가동된다. 하지만, 이 작업 도중 명령어가 시그널을 받아서 종료되는 경우는 데이터베이스 서버는 가동 되지 않는다.

10.2. 복구 시점 설정

`recovery_target = 'immediate'`

이 매개변수는 일관적인 상태가 되자마자 복구가 종료되도록 한다. 온라인 백업에서 복구하는 경우, 백업이 끝난 시점을 가리킨다.

실질적으로 이는 문자열 매개변수지만 `'immediate'` 가 쓸 수 있는 유일한 값이다.

`recovery_target_name` (문자열)

이 매개변수는 특정 복구 위치를 이름으로 지정한다. 그 이름은 `pg_create_restore_point()` 함수에서 지정한 이름이다. 대개 `recovery_target_name` 값이나 recovery target time, recovery target xid 값 가운데 하나를 지정한다. 이 값을 지정하지 않으면 복구할 WAL 파일의 마지막까지 복구한다.

`recovery_target_time` (timestamp)

이 매개변수는 특정 시점을 시간으로 지정한다. 대개 `recovery_target_time` 값이나 recovery-target-name, recovery-target-xid 값 가운데 하나를 지정한다. 이 값을 지정하지 않으면 복구할 WAL 파일의 마지막까지 복구한다. 또한 recovery-target-inclusive 값에 영향을 받아서 정확한 중지 위치가 결정되기도 한다.

`recovery_target_xid` (문자열)

이 매개변수는 특정 트랜잭션 ID로 지정한다. 지정한 이 트랜잭션 ID까지 복구 작업을 한다. 트랜잭션 ID는 복구 서버의 마지막부터 차례대로 진행되기 때문에 결과적으로 백업의 트랜잭션 ID와 복구 서버의 트랜잭션 ID가 같지 않을 수 있음을 기억 해야한다. 복구되는 트랜잭션은 지정한 트랜잭션 이전에 커밋된 모든 트랜잭션들이다. (선택적으로 지정한 트랜잭션도 포함될 수 있다.) 대개 `recovery_target_xid` 값이나 recovery target name, recovery target time 값 가운데 하나를 지정한다. 이 값을 지정하지 않으면 복구할 WAL 파일의 마지막 트랜잭션까지 복구한다. 또한 recovery target inclusive 값에 영향을 받아서 정확한 중지 위치가 결정되기도 한다.

`recovery_target_inclusive` (boolean)

위에서 지정한 지점을 포함해서 복구 할 것인지, 포함하지 않고 복구 할 것인지를 결정한다. 기본값은 `true`이다. `false`로 지정하면 특정 지점(트랜잭션 ID나, 시간) 전까지만 복구를 하고 복구 작업을 중지한다. 이 값은 recovery target time, recovery target xid 설정값의 작동에 영향을 준다.

recovery_target_timeline (문자열)

특정 타임라인 번호를 지정한다. 이 값을 지정하지 않으면 베이스 백업에서 사용한 타임라인의 다음 타임라인 번호가 사용된다. 이 번호는 백업된 WAL 세그먼트 파일을 모아둔 디렉토리 안에 타임라인 이력 파일(일반적으로 0000000n.history 형태의 파일이다. 이 앞부분 숫자가 타임라인 번호다.)들 가운데 하나의 번호를 선택한다. 타임라인은 복구를 하고 운영 되었다가 같은 베이스 백업 기반으로 다시 복구를 해서 여러 타임라인이 생겼을 경우 특정 타임라인으로 복구 할 때 유용하게 사용된다. 자세한 이야기는 8.3.5절을 참조하라.

pause_at_recovery_target (boolean)

리플리케이션 대기 서버 환경에서 마지막 복구를 한 상태가 되면 복구 작업을 멈추고 대기할 것인지를 지정한다. 기본값은 true이다. 이 설정은 복구 작업을 마지막까지 했을 때 그 위치가 원하는 위치인지 확인 할 수 있는 쿼리를 실행할 수 있는지를 결정한다. 임시 중지 상태라면 pg_xlog_replay_resume() 함수를 이용해서 복구 작업을 이어서 다시 할 수 있다. 복구 작업을 하다가 원하는 마지막 위치에서 멈출 수 없으면, 데이터베이스는 중지된다. 이 경우에는 복구 타켓을 마지막으로 바꾸고, 서버를 재실행 해서 복구 작업을 계속 해야한다.

이 설정은 hot-standby 활성 상태에서만 사용할 수 있으며, 복구 타켓이 지정되어 있어야만 적용된다.

10.3. 대기 서버 설정

standby_mode (boolean)

Agens SQL 서버를 대기 전용(standby) 서버로 운영 하고자 할 때 이 값을 on으로 설정한다. 이렇게 하면 복구할 WAL 파일을 모두 서버에 적용하고 나서 primary_conninfo 접속 정보로 운영 서버로 접속하면 운영 서버에서 반영된 트랜잭션을 대기 서버에도 그대로 반영한다. 운영 서버와의 접속이 끊기면 다시 restore_command 설정에 지정한 명령을 이용해서 다시 대기 서버 쪽에 있는 WAL 파일들을 반영한다. 이런 작업을 반복해서 운영 서버의 대기 서버로 운영할 수 있다.

primary_conninfo (문자열)

운영 서버(primary server, master server)의 접속 정보. 여기서 사용할 수 있는 접속 환경은 표준 접속 방식과 동일하므로 접속에 관련된 모든 환경 변수들을 그대로 사용할 수 있다. 사용 우선 순위도 동일하다. 환경 변수가 지정되어 있고 이 문자열에 지정되어 있지 않으면 환경 변수 값을 사용하고, 환경 변수도 지정돼 있지 않으면 기본값을 사용한다.

위 standby_mode 설정값을 on으로 설정했으면, 이 접속 정보 문자열에는 최소한 운영 서버의 호스트 이름(또는 주소)는 지정해야한다. 포트가 기본값이라면, 생략해도 된다. 접속하려는 사용자가 계정은 운영 서버에 있어야하며, 해당 계정이 운영 서버로 접속해서 복제 작업을 할 수 있는 권한이 있어야한다. (9.2.5.1절 참조) 비밀번호 인증이 필요하면, primary_conninfo 문자열에 지정하든가 대기 서버의 ~/.pgpass 파일에 데이터베이스 이름을 replication으로 지정하고, 비밀번호를 등록해서 사용한다.
primary_conninfo 설정값에는 접속할 데이터베이스 이름은 지정하지 않는다.

standby_mode 설정값이 off 상태면, 이 설정은 무시된다.

trigger_file (문자열)

대기 상태가 끝났음을 알리는 파일 이름을 지정한다. 이 파일이 있으면 대기 모드 상태를 끝내고 독립 운영 서버로 실행환경을 바꾼다. 이 파일 이름을 지정하지 않으면 **pg_ctl**

promote 명령으로 대기 모드를 끝낼 수 있다. `standby_mode` 설정값이 `off` 상태면 이 설정은 무시된다.

`recovery_min_apply_delay` (integer)

기본적으로 대기 서버는 운영서버의 WAL 레코드를 최대한 빠르게 복원한다. 지연된 데이터 사본을 이용해 데이터 손실 에러를 다양한 방식으로 해결할 수 있다. 이 매개 변수는 지정된 시간만큼 복구를 지연하게 하고, 시간 단위는 기본적으로 밀리 세컨드이다. 예를 들어 매개 변수를 `5min`으로 지정하면, 대기 서버는 시스템 시각이 적어도 마스터 서버에 보고된 커밋 시간보다 5분 지났을 때 각 트랜잭션 커밋을 리플레이 한다.

서버 간 리플리케이션 지연이 이 매개변수의 값을 초과할 수도 있다. 지연 시간은 마스터 서버에 쓰인 WAL timestamp와 현재 대기 서버의 시간 차이를 뜻한다. 네트워크나 지속적인 리플리케이션 설정으로 인한 전송 상의 지연 시간은 실제 대기 시간을 대폭 줄여준다. 마스터 서버와 대기 서버간 시스템 시간이 동기화되지 않으면 예상보다 일찍 레코드를 복구할 수 있다. 서버간 시간 편차를 이용하는 것보다 매개변수를 잘 설정하는 것이 더 유용하다. 마스터 서버와 대기 서버의 타임존은 다르게 설정해야 한다.

지연은 COMMIT과 복원 시점인 WAL 레코드에서만 발생한다. 다른 레코드들은 지정한 지연 시간보다 일찍 리플레이 되는데, 복구 충돌이 더 자주 발생할 가능성도 있지만 MVCC에서는 큰 문제가 되지 않는다.

지연은 대기 서버가 promote되거나 trigger될 때까지 발생한다. 그리고 나서 대기 서버는 대기하는 것을 멈추고 복구를 종료한다.

이 매개변수는 스트리밍 리플리케이션 배치 용도로 만들어졌지만, 모든 경우에 적용된다. 동기적인 리플리케이션은 영향을 받지 않는데, 트랜잭션 커밋을 동기적으로 요청하는 설정이 없기 때문이다. `hot_standby_feedback`은 이 기능을 사용하면 지연되어 마스터 서버에 과부하가 걸릴 수 있으므로 두 매개변수를 적절히 사용해야 한다.

11장. 데이터베이스 성능 모니터링

데이터베이스 관리자가 하는 일 가운데 하나는 "이 데이터베이스 서버가 정상적으로 운영되고 있는지" 를 지켜 보는 일이다. 여기서는 이에 대한 부분을 다룬다.

데이터베이스 운영 상태를 지켜 보거나, 성능을 분석하는 도구들은 여러가지가 있다. 이 장에서 다루는 대부분의 내용은 Agens SQL 통계수집기에 대한 것이지만, **ps**, **top**, **iostat**, **vmstat** 같은 유닉스에서 일반적으로 사용하는 모니터링 프로그램에 대한 설명도 함께 한다. 또한 EXPLAIN 명령어로 살펴보아야 하는 최적화 되지 못한 쿼리들에 대한 해결 방법도 잠깐 다룬다.

11.1. 표준 유닉스 도구들

대부분의 유닉스 환경에서는 Agens SQL 서버의 동작 상태를 **ps** 쉘 명령어만으로도 대략적으로 살펴 볼 수 있다. 왜냐하면 서버는 자신의 각 하위 프로세스들의 명령어 이름들을 동적으로 바꾸어서 운영체제가 그것을 볼 수 있도록 하기 때문이다. 그래서 다음과 같은 간단한 명령어로 서버 상태를 살펴 볼 수 있다:

```
$ ps auxww | grep ^postgres
postgres  15551  0.0  0.1  57536  7132 pts/0    S    18:02   0:00 postgres -i
postgres  15554  0.0  0.0  57536  1184 ?        Ss   18:02   0:00 postgres: writer proces
postgres  15555  0.0  0.0  57536   916 ?        Ss   18:02   0:00 postgres: checkpointer
postgres  15556  0.0  0.0  57536   916 ?        Ss   18:02   0:00 postgres: wal writer pr
postgres  15557  0.0  0.0  58504  2244 ?        Ss   18:02   0:00 postgres: autovacuum la
postgres  15558  0.0  0.0  17512  1068 ?        Ss   18:02   0:00 postgres: stats collect
postgres  15582  0.0  0.0  58772  3080 ?        Ss   18:04   0:00 postgres: joe runbug 12
postgres  15606  0.0  0.0  58772  3052 ?        Ss   18:07   0:00 postgres: tgl regressio
postgres  15610  0.0  0.0  58772  3056 ?        Ss   18:07   0:00 postgres: tgl regressio
```

(이 명령은 여러 유닉스 제각각이다. 윗 예제는 요즘 사용하는 리눅스 시스템에서 사용하는 명령어다. **ps** 명령의 옵션에 대한 자세한 설명은 사용하고 있는 유닉스 운영체제 설명서를 설펴보는 것이 좋다.) 윗 예제 출력 결과의 첫번째 줄은 데이터베이스 서버의 최상위 프로세스이며, 서버를 실행 할 때 사용했던 옵션들도 함께 보인다. (Agens SQL 서버는 쓰레드 방식이 아니라, 다중 프로세스 방식으로 운영된다. 다중 프로세스 방식이란 서버 관리자가 서버 실행 명령어를 OS 쉘에서 실행하면 서버 최상위 프로세스 실행되고 그 프로세스를 필요한 여러 프로세스를 실행하는 방식이다.) 다음 다섯 줄은 이 최상위 프로세스가 실행한 서버 운영에 필요한 하위 프로세스들이다. 이것을 서버 백그라운드 프로세스라고 한다. 이 프로세스들은 윗 예제와 꼭 같지는 않다. "autovacuum launcher" 프로세스는 autovacuum 기능을 사용하지 않으면 보이지 않을 것이며, 서버 로그를 stderr 쪽으로 보내지 않고 따로 보관하는 기능을 사용하면, "postgres: logger process" 같은 프로세스도 보일 것이다. 나머지는 클라이언트가 접속해서 서버측에서 만든 세션 프로세스들이다. 이 세션 프로세스들은 다음과 같은 양식으로 보여준다.

`postgres:` *사용자 데이터베이스 호스트 현재작업상태*

사용자, 데이터베이스, (클라이언트) 호스트 정보는 그 프로세스가 종료 될 때까지 항상 같지만 현재 작업 상태 정보는 그 세션의 작업 상태에 따라 바뀐다. 현재 작업 상태에 `idle`은 이 세션이 클라이언트의 명령을 대기하고 있음을 뜻한다. `idle in transaction`으로 표시되는 것은 그 세션이 현재 **BEGIN** 명령을 사용해서 트랜잭션 영역 안에 있지만 클라이언트 측에서 아무 작업도 안하고 있는 상태이다. `SELECT`와 같이 명령어 종류를 보여 주는 경우도 있고, 그 뒤에 `waiting`이 붙어 있는 경우도 있다. 이 경우는 다른 프로세스가 어떤 작업을 하고 있어 이 세션이 해당 작업을 대기하고 있는 상태를 나타낸다. 윗 예제를 보면 15610 프로세스 때문

에 15606 프로세스가 작업을 대기하고 있음을 알 수 있다. (윗 예제 화면에서는 다른 세션들이 없기 때문에 15610 프로세스가 잠금 문제를 일으킨 프로세스로 파악 할 수 있다. 어떤 잠금을 사용하고 있기에 다른 프로세스가 기다리고 있는지에 대한 자세한 정보는 pg_locks 뷰를 통해서 알 수 있다.)

update-process-title 환경 설정값을 off로 지정했을 때 프로세스 현재작업상태 자리에는 그 프로세스의 첫 작업에 대한 이름으로 지정되면 그 프로세스가 종료될 때까지 그대로 유지된다. 일부 OS에서는 이 프로세스 이름을 바꾸는 작업이 부하를 유발한다고 사용하지 않는 경우도 있고, 또 일부 OS에서는 이 작업에 대해서 전혀 신경 쓰지 않는 경우도 있다.

> 작은 정보**:** Solaris에서는 이 부분 처리가 독특하다. 먼저 **/bin/ps** 명령 대신에 **/usr/ucb/ps** 사용하고 `w` 옵션을 두 개 지정하고 서버 시작 최상위 프로세스의 이름이 각 세션 프로세스의 이름보다 짧게 지정하면 세션 프로세스들의 이름이 동적으로 바뀌는 것을 살펴볼 수 있다. 하지만 이 세가지 조건 중 하나라도 만족하지 않으면 모든 프로세스 이름은 서버 최상위 프로세스의 이름과 같게 보인다.

11.2. 통계 수집기

Agens SQL *통계 수집기*는 서버 운영 상태에 대한 정보를 수집하거나 보고하기 위한 작업을 하는 백그라운드 시스템이다. 현재 이 수집기는 테이블이나 인덱스의 디스크 블록 단위 또는 개별 로우 단위의 접근 횟수를 수집할 수 있다. 또한 각 테이블에 저장 되어있는 총 로우 수를 수집하며, 각 테이블에 대한 vacuum 작업과 analyze 작업에 관한 정보들도 수집한다. 또한 사용자 정의 함수들의 호출 횟수와 그것들의 각 총 수행 시간들도 수집한다.

또한 Agens SQL에서는 각기 다른 서버 프로세스들이 자기가 무슨 작업을 하고 있는지에 대한 현재 상태를 살펴볼 수 있다. 이 기능은 수집기 프로세스와 별개로 제공하고 있는 기능이다.

11.2.1. 통계 수집기 환경설정

통계 자료를 수집한다는 것은 궁극적으로는 그 만큼의 추가 비용을 서버가 사용한다는 것을 의미한다. 그래서 그 추가 비용을 얼마만큼 쓸 것인지에 대한 결정을 서버 환경변수로 제어할 수 있다. 다음은 `postgresql.conf` 환경설정 파일에서 사용할 수 있는 환경변수들이다. (이들에 대한 보다 자세한 부분은 2장을 참고하라.)

track-activities 설정값을 활성화하면 각각의 서버 프로세스들은 현재 자신이 하고 있는 작업을 프로세스 이름으로 보여준다.

track-counts 설정값을 활성화하면 테이블과 인덱스의 사용빈도를 통계 수집기가 수집한다.

track-functions 설정값을 활성화하면 사용자 정의 함수들의 사용빈도를 통계 수집기가 수집한다.

track-io-timing 설정값을 활성화면 블록 읽기 쓰기 회수를 통계 수집기가 수집한다.

일반적으로 이 환경설정 변수들은 `postgresql.conf`에서 그 값을 지정하면 이것은 모든 서버의 프로세스들을 대상으로 작동하게 된다. 하지만 SET 명령을 이용해서 개별 세션 단위로 이 설정값을 변경할 수도 있다. (이 작업은 일반 사용자가 악의적으로 사용될 가능성이 있기 때문에 슈퍼유저만 사용할 수 있다.)

통계 수집기가 만든 정보는 stats-temp-directory 설정값으로 지정한 디렉토리 (초기값은 `pg_stat_tmp`) 안에 있는 임시 파일로 저장되고, 그것을 여러 다른 Agens SQL 프로세스들이 이용한다. 보다 나은 성능을 위해서는 이 파일의 I/O 성능을 높이기 위해 이 디렉토리를 메모

리 기반 파일시스템에 두는 것도 좋은 방법이다. 이 통계 정보는 서버가 중지 될 때 pg_stat 디렉토리 쪽으로 복사하는 작업을 하기 때문에 서버 중지, 재시작에도 자료를 그대로 유지할 수 있다.

11.2.2. 수집된 통계 정보 보기

수집된 통계 정보는 아래 표 11-1에서 나열한 여러 뷰를 통해서 살펴 볼 수 있다. 다른 방법으로는 11.2.3절에서 제공하는 여러 통계 정보 보기 함수들을 이용해서 필요한 뷰를 직접 만들어서 살펴 볼 수도 있을 것이다.

여기서 중요한 점은 살펴 보고 있는 통계 정보가 현재 데이터베이스의 정확한 현재 상태가 아니라는 점이다. 각 개별 프로세스들이 수집한 통계 정보는 그 프로세스가 아무런 작업을 하고 있지 않을 때 수집기에게 전달한다. 즉 한 쿼리가 실행 되고 있다거나 트랜잭션 내에 있으면 그 과정에 생긴 통계 정보들은 반영되지 않고 있다는 것이다. 또한 통계 수집기의 작업은 PGSTAT_STAT_INTERVAL (기본값은 500ms 이다) 시간 간격으로 진행된다. 즉 이 만큼의 통계 정보 오차가 발생한다. 하지만 track_activities 관련 정보는 항상 즉시 반영 된다.

또 다른 중요한 점은 한 통계 정보는 해당 뷰나 함수로 출력할 때, 한 트랜잭션 내에서는 항상 같은 값을 출력한다. 즉, 트랜<51257>션이 끝날 때까지 그 값을 유지한다. 이와 비슷하게 모든 세션들의 현재 쿼리에 대한 정보도 하나의 트랜잭션 내에서는 같은 정보를 출력한다. 이것은 버그가 아니라 특성이다. 이것은 한 트랜잭션 내에서 그 통계 정보가 일정하게 유지 되도록해서 여러 다른 쿼리들에서 그 값을 일관성 있게 한다. 하지만 이것을 원치 않는다면 각 쿼리들을 트랜잭션 단위로 분리해서 작업하면 된다. 다른 방법으로 pg_stat_clear_snapshot() 함수를 사용해서 현재 스냅샷으로 지정된 통계 정보를 버리고 새 통계 정보를 사용할 수도 있다.

하나의 트랜잭션 내에서 그 자신의 통계 정보 (아직까지 수집기 쪽으로 보내지 않은 현재 작업 내역에 대한 정보)들은 다음과 같은 뷰에서 제공한다: pg_stat_xact_all_tables, pg_stat_xact_sys_tables, pg_stat_xact_user_tables, pg_stat_xact_user_functions. 이들의 통계 정보는 현재 트랜잭션 내에서도 자신의 세션 작업에 대한 통계치를 반영 해서 다른 세션에서 수집된 정보와 자신의 세션에서 수집된 정보와 차이가 생길 수도 있다.

표 **11-1.** 표준 통계 정보 뷰

뷰 이름	설명
pg_stat_activity	세션 프로세스들의 정보를 각각 하나의 로우로 보여준다. 이 프로세스들은 현재 서버를 사용하고는 있고, 클라이언트가 서버로 접속 해서 만들어진 하위 서버 프로세스들이다. 이들의 현재 상태 및 실행 중인 쿼리들을 살펴 볼 수 있다.
pg_stat_archiver	WAL 아카이버 프로세스 작동에 대한 통계 정보를 하나의 로우로 보여준다.
pg_stat_bgwriter	백그라운드 라이터 프로세스의 작업 통계 정보.
pg_stat_database	한 로우에 하나씩 각 데이터베이스 전역 통계 정보를 보여 준다.
pg_stat_all_tables	현재 접속한 데이터베이스에 속한 모든 테이블에 대해서 한 로우씩 그 테이블 사용에 대한 통계 정보를 보여 준다.
pg_stat_sys_tables	pg_stat_all_tables 내용과 같은데, 시스템 테이블에 대해서만 보여준다.

뷰 이름	설명
pg_stat_user_tables	pg_stat_all_tables 내용과 같은데, 시스템 테이블을 제외한 현재 사용자 접근 할 수 있는 테이블에 대한 정보만 보여준다.
pg_stat_xact_all_tables	pg_stat_all_tables 내용과 비슷하지만, 여기서 반영된 숫자들은 현재 트랜잭션 내에서 반영된 정보만 보여준다. (즉 pg_stat_all_tables 쪽으로 반영 되지 않은 트랜잭션 내의 정보다). 이 뷰에는 사용하는 실 로우 수, 사용하지 않는 로우(dead row) 수, vacuum과 analyze 작업에 관계된 정보는 제공하지 않는다.
pg_stat_xact_sys_tables	pg_stat_xact_all_tables 내용과 같은데, 시스템 테이블에 대해서만 보여준다.
pg_stat_xact_user_tables	pg_stat_xact_all_tables 내용과 같은데, 시스템 테이블을 제외한 현재 사용자가 접근 할 수 있는 테이블에 대해서만 보여준다.
pg_stat_all_indexes	현재 접속한 데이터베이스에 속한 모든 인덱스에 대해서 한 로우씩 그 인덱스 사용에 대한 통계 정보를 보여 준다.
pg_stat_sys_indexes	pg_stat_all_indexes 내용과 같은데, 시스템 인덱스에 대해서만 보여준다.
pg_stat_user_indexes	pg_stat_all_indexes 내용과 같은데, 시스템 인덱스를 제외한 현재 사용자가 접근 할 수 있는 인덱스에 대해서만 보여준다.
pg_statio_all_tables	현재 접속한 데이터베이스에 속한 모든 테이블에 대해서 한 로우씩 그 테이블에 대한 I/O 통계 정보를 보여 준다.
pg_statio_sys_tables	pg_statio_all_tables 내용과 같은데, 시스템 테이블에 대해서만 보여준다.
pg_statio_user_tables	pg_statio_all_tables 내용과 같은데, 시스템 테이블을 제외한 현재 사용자가 접근 할 수 있는 테이블에 대해서만 보여준다.
pg_statio_all_indexes	현재 접속한 데이터베이스에 속한 모든 인덱스에 대해서 한 로우씩 그 인덱스에 대한 I/O 통계 정보를 보여 준다.
pg_statio_sys_indexes	pg_statio_all_indexes 내용과 같은데, 시스템 인덱스에 대해서만 보여준다.
pg_statio_user_indexes	pg_statio_all_indexes 내용과 같은데, 시스템 인덱스를 제외한 현재 사용자가 접근 할 수 있는 인덱스에 대해서만 보여준다.
pg_statio_all_sequences	현재 접속한 데이터베이스에 속한 모든 시퀀스에 대해서 한 로우씩 그 시퀀스에 대한 I/O 통계 정보를 보여 준다.
pg_statio_sys_sequences	pg_statio_all_sequences 내용과 같은데, 시스템 시퀀스에 대해서만 보여준다. (현재 사용하고 있는 시스템 시퀀스가 없음으로 이 뷰는 항상 그 내용이 비어있을 것이다.)

뷰 이름	설명
pg_statio_user_sequences	pg_statio_all_sequences 내용과 같은데, 시스템 시퀀스를 제외한 현재 사용자가 접근 할 수 있는 시퀀스에 대해서만 보여준다.
pg_stat_user_functions	현재 데이터베이스에 만들어진 사용자 정의 함수들의 실행에 대한 통계 정보를 보여준다.
pg_stat_xact_user_functions	pg_stat_user_functions 내용과 비슷하지만, 여기서 반영된 숫자들은 현재 트랜잭션 내에서 반영된 정보만 보여준다, (즉 pg_stat_user_functions 쪽으로 반영 되지 않은 트랜잭션 내의 정보다).
pg_stat_replication	하나의 WAL 송신 프로세스에 대해서 하나의 로우로, 대기 서버 쪽으로 보내는 리플리케이션 작업에 대한 통계 정보를 보여준다.
pg_stat_database_conflicts	하나의 데이터베이스에서 그 전역에 걸쳐 발생한 대기 서버 복제 작업 충돌에 의한 쿼리 실행 실패 통계 정보를 각각 하나의 로우로 보여준다.

인덱스 별 통계 정보는 어느 인덱스가 많이 사용되며, 어떻게 영향을 미치는 지를 조사하는데 특히 유용하게 쓰인다.

pg_statio_ 뷰들은 공유 버퍼의 사용 빈도를 파악하는데 가장 기본이 되는 것들이다. 이상적인 서버 운영 모습은 디스크 읽기 빈도 보다 공유 버퍼를 사용하는(buffer hit) 빈도가 훨씬 높아서, 대부분의 작업이 커널 호출 작업 없이 이루어지도록 하는 것이다. 한편, Agens SQL 에서는 단지 이 서버 기준으로 디스크를 읽은 것과 서버의 공유 버퍼를 사용한 것에 대한 정보만 제공할 뿐이다. 즉, 커널의 I/O 캐시 사용 빈도에 대해서는 제공하지 않는다. 다시 말하면, 정말 디스크를 읽은 것인지, OS의 I/O 캐시를 사용한 것인지는 이 뷰를 통해서는 알 수 없다. 이런 세세한 부분까지 살펴 보려면 OS 관련 도구들을 이용해야 할 것이다.

표 **11-2. pg_stat_activity View**

칼럼	자료형	설명
datid	oid	이 백엔드가 연결된 데이터베이스의 OID
datname	name	이 백엔드가 연결된 데이터베이스의 이름
pid	integer	이 백엔드의 프로세스 ID
usesysid	oid	이 백엔드에 로그인 한 사용자의 OID
usename	name	이 백엔드에 로그인 한 사용자의 이름
application_name	text	이 백엔드에 연결된 애플리케이션의 이름

칼럼	자료형	설명
client_addr	inet	이 백엔드에 연결된 클라이언트의 IP 주소. 이 필드가 null이면 클라이언트가 서버 머신의 UNIX 소켓으로 연결이 되어 있다는 뜻이거나 autovacuum같은 내부 프로세스라는 뜻임
client_hostname	text	*client_addr*의 DNS를 역조회해서 찾은 클라이언트의 호스트명. 이 필드는 IP 연결했을 때와 log-hostname이 활성화됐을 때만 null이 아닌 값으로 저장됨
client_port	integer	이 백엔드와 교류하는 데 클라이언트가 쓰는 TCP 포트 넘버. -1은 UNIX 소켓이 사용됐을 때
backend_start	timestamp with time zone	이 프로세스가 시작된 시간. 예를 들어 클라이언트가 서버에 연결되었을 때
xact_start	timestamp with time zone	이 프로세스의 현재 트랜잭션이 시작된 시간이거나 어떤 트랜잭션도 수행되고 있지 않을 때 null로 표시함 현재 쿼리가 트랜잭션의 첫 번째 쿼리이면 이 칼럼은 *query_start* 칼럼과 같다.
query_start	timestamp with time zone	현재 수행 중인 쿼리가 시작된 시간이거나 *state*가 active가 아닐 때에는 마지막 쿼리가 시작된 시간을 뜻함
state_change	timestamp with time zone	*state*가 마지막으로 변경된 시간
waiting	boolean	이 백엔드가 잠금을 기다리고 있으면 true

칼럼	자료형	설명
state	text	현재 이 백엔드의 전체 상태. 가능한 값들은 아래와 같다. • active: 백엔드가 쿼리를 수행하고 있다는 뜻 • idle: 백엔드가 새로운 클라이언트 명령어를 기다리고 있음을 나타냄 • idle in transaction: 백엔드가 트랜잭션을 수행하고 있지만 쿼리를 수행하고 있지는 않은 상태 • idle in transaction (aborted): idle in transaction과 비슷하지만 트랜잭션의 구문들 중 하나가 에러를 발생시켰을 때 • fastpath function call: 백엔드가 fast-path함수를 수행할 때 • disabled: track-activities가 이 백엔드에서 비활성화됐을 때
query	text	이 백엔드의 가장 최근 쿼리 내 텍스트. *state*가 active이면 이 필드는 현재 수행 중인 쿼리를 보여 줌. 다른 state들에서는 마지막으로 수행된 쿼리를 보여 줌

pg_stat_activity 뷰는 서버 프로세스당 한 개의 로우를 가지며 이는 해당 프로세스의 현재 활동 관련 정보를 보여준다.

> 참고: *waiting*과 *state*칼럼은 독립적이다. 백엔드가 active 상태이면, waiting일 수도 있고 아닐 수도 있다. 상태가 active이고 *waiting*이 true면, 쿼리가 수행 중임을 뜻하지만 시스템 어딘가에서 잠금으로 블록 됐음을 뜻한다.

표 **11-3. pg_stat_archiver View**

칼럼	자료형	설명
archived_count	bigint	성공적으로 아카이브 된 WAL 파일의 개수

칼럼	자료형	설명
last_archived_wal	text	최근 성공적으로 아카이브 된 WAL 파일의 이름
last_archived_time	timestamp with time zone	최근 성공적으로 아카이브 작업을 한 시간
failed_count	bigint	WAL 파일 아카이빙 시도에 실패한 횟수
last_failed_wal	text	마지막으로 아카이빙 작업에 실패한 WAL 파일의 이름
last_failed_time	timestamp with time zone	마지막으로 아카이빙 작업에 실패한 시간
stats_reset	timestamp with time zone	이 통계치가 마지막으로 리셋된 시간

pg_stat_archiver 뷰는 클러스터의 아카이버 프로세스에 관한 데이터가 있는 하나의 로우만 갖고 있다.

표 **11-4. pg_stat_bgwriter** 뷰

칼럼	자료형	설명
checkpoints_timed	bigint	checkpoint_timeout 환경 설정값에 따른 체크 포인트 작업 회수
checkpoints_req	bigint	체크포인트 명령이 수행되어 진행한 작업 회수
checkpoint_write_time	double precision	체크포인트 작업으로 발생한 디스크 쓰기 작업의 밀리세컨드 총 시간.
checkpoint_sync_time	double precision	체크포인트 작업으로 발생할 디스크 쓰기에 대한 디스크 동기화 작업(fsync)에 소요된 밀리세컨드 총 시간.
buffers_checkpoint	bigint	checkpointer 프로세스가 기록한 총 버퍼 수
buffers_clean	bigint	writer 프로세스가 기록한 총 버퍼 수
maxwritten_clean	bigint	writer 프로세스가 기록해야 할 버퍼수가 bgwriter_lru_maxpages 환경 설정값보다 많아서 작업이 중지된 회수
buffers_backend	bigint	백엔드(세션) 프로세스가 직접 기록한 총 버퍼 수

칼럼	자료형	설명
buffers_backend_fsync	bigint	백엔드(세션) 프로세스가 직접 fsync 작업을 한 회수 (일반적으로 백엔드(세션) 프로세스가 공유 버퍼를 디스크로 내려 쓰더라도, 이 동기화 작업은 writer 프로세스가 담당한다. 즉 writer 프로세스의 fsync 작업 부하량을 파악하는데 도움이 됨)
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	이 통계값들이 초기화 된 시각

pg_stat_bgwriter 뷰는 항상 한 개의 로우만 보여준다. 이 자료는 데이터클러스터 전역 정보다.

표 **11-5. pg_stat_database View**

칼럼	자료형	설명
datid	oid	데이터베이스의 OID
datname	name	해당 데이터베이스의 이름
numbackends	integer	현재 연결된 백엔드의 수. 현재 상태를 가리키는 값을 리턴하는 유일한 칼럼. 다른 칼럼들은 마지막 리셋 시점 이후로 축적된 값들을 리턴함.
xact_commit	bigint	커밋된 트랜잭션의 수
xact_rollback	bigint	롤백한 트랜잭션의 수
blks_read	bigint	읽힌 디스크 블록의 수
blks_hit	bigint	버퍼 캐쉬에 이미 존재하는 디스크 블록을 감지하여 읽기가 불필요함을 알려줌. (Agens SQL 버퍼 캐쉬 내 히트 수만 계산하고, 운영체제 파일 시스템 캐쉬는 확인하지 않음)
tup_returned	bigint	쿼리가 리턴한 로우의 개수
tup_fetched	bigint	쿼리가 fetch한 로우의 개수
tup_inserted	bigint	쿼리로 삽입된 로우의 개수
tup_updated	bigint	쿼리로 업데이트된 로우의 개수
tup_deleted	bigint	쿼리로 삭제된 로우의 개수
conflicts	bigint	복구 충돌로 인해 취소된 쿼리의 개수 (대기 서버에서만 발생하는 충돌을 말함. pg-stat-database-conflicts-view 참조)

칼럼	자료형	설명
temp_files	bigint	쿼리로 생성된 임시 파일의 개수. 모든 임시 파일은 임시 파일이 생성된 이유와 log-temp-files 설정을 불문하고 카운트 됨
temp_bytes	bigint	쿼리로 임시 파일에 쓰여진 데이터의 총합. 모든 임시 파일은 임시 파일이 생성된 이유와 log-temp-files 설정을 불문하고 카운트 됨
deadlocks	bigint	감지된 교착상태의 개수
blk_read_time	double precision	백엔드가 데이터 파일 블록을 읽는 데 소요된 시간으로 밀리세컨드 단위를 사용
blk_write_time	double precision	백엔드가 데이터 파일 블록을 쓰는데 소요된 시간으로 밀리세컨드 단위를 사용
stats_reset	timestamp with time zone	통계치가 마지막으로 리셋된 시간

pg_stat_database 뷰는 클러스터 내 데이터베이스 당 한 개의 로우를 가지며 전체 데이터베이스 통계치를 보여준다.

표 **11-6. pg_stat_all_tables View**

칼럼	자료형	설명
relid	oid	테이블의 OID
schemaname	name	테이블을 포함하는 스키마 이름
relname	name	테이블명
seq_scan	bigint	풀스캔을 시도한 횟수
seq_tup_read	bigint	풀스캔으로 가져온 live row 개수
idx_scan	bigint	인덱스 스캔을 시도한 횟수
idx_tup_fetch	bigint	인덱스 스캔으로 가져온 live row 개수
n_tup_ins	bigint	삽입된 로우 개수
n_tup_upd	bigint	업데이트된 로우 개수
n_tup_del	bigint	삭제된 로우 개수
n_tup_hot_upd	bigint	HOT 업데이트 된 로우 개수 (예를 들어 따로 인덱스 업데이트 필요가 없는)
n_live_tup	bigint	추정되는 live row 개수
n_dead_tup	bigint	추정되는 dead row 개수

칼럼	자료형	설명
last_vacuum	timestamp with time zone	테이블이 수동적으로 vacuum된 마지막 시간 (**VACUUM FULL**은 해당되지 않음)
last_autovacuum	timestamp with time zone	autovacuum 데몬으로 테이블이 vacuum된 마지막 시간
last_analyze	timestamp with time zone	테이블이 수동적으로 분석된 마지막 시간
last_autoanalyze	timestamp with time zone	autovacuum 데몬으로 테이블이 분석된 마지막 시간
vacuum_count	bigint	수동적으로 vacuum된 횟수 (**VACUUM FULL**은 제외)
autovacuum_count	bigint	autovacuum 데몬으로 vacuum된 횟수
analyze_count	bigint	수동적으로 분석된 횟수
autoanalyze_count	bigint	autovacuum 데몬으로 분석된 횟수

pg_stat_all_tables 뷰는 현재 데이터베이스의 테이블(TOAST 테이블 포함)당 한 개의 로우를 가지며, 이는 해당 테이블 접근에 대한 통계치를 보여준다. pg_stat_user_tables와 pg_stat_sys_tables뷰는 같은 정보를 가지나 사용자와 시스템 테이블 각각을 보여주는 데만 쓰인다.

표 **11-7. pg_stat_all_indexes View**

칼럼	자료형	설명
relid	oid	이 인덱스를 갖는 테이블의 OID
indexrelid	oid	이 인덱스의 OID
schemaname	name	이 인덱스가 있는 스키마의 이름
relname	name	이 인덱스를 갖는 테이블명
indexrelname	name	인덱스명
idx_scan	bigint	이 인덱스로 스캔한 횟수
idx_tup_read	bigint	이 인덱스로 스캔해서 리턴된 인덱스 엔트리의 개수
idx_tup_fetch	bigint	이 인덱스로 인덱스 스캔해서 가져온 live 테이블 로우의 개수

pg_stat_all_indexes 뷰는 현재 데이터베이스에서 인덱스 당 한 개의 로우를 가지며 해당 인덱스 접근에 대한 통계치를 보여준다. pg_stat_user_indexes와 pg_stat_sys_indexes 뷰는 같은 정보를 갖지만 각각 사용자와 시스템 인덱스만 보여준다.

인덱스는 일반 인덱스 스캔 혹은 “bitmap” 인덱스 스캔을 사용할 수 있다. 비트맵 스캔에서 일부 인덱스 결과는 AND나 OR 규칙으로 통합될 수 있어서, 비트맵 스캔이 사용되면 각각의 heap 로우들을 특정 인덱스에 매칭하기 어렵다. 비트맵 스캔은 사용된 인덱스의 pg_stat_all_indexes.*idx_tup_read*를 증가시키고, 테이블의

pg_stat_all_tables.*idx_tup_fetch*를 증가시키지만 pg_stat_all_indexes.*idx_tup_fetch*에는 영향을 미치지 않는다.

> 참고: *idx_tup_read*와 *idx_tup_fetch*의 값은 비트맵 스캔을 하지 않아도 다를 수 있다. *idx_tup_read*는 인덱스에서 회수된 인덱스 엔트리 개수를 세지만, *idx_tup_fetch* 는 테이블에서 가져온 live row를 세기 때문이다. 후자는 죽었거나 아직 커밋되지 않은 로우가 fetch되어 인덱스를 사용할 경우나 index-only 스캔으로 heap fetch들을 감지하지 못한 경우 감소한다.

표 **11-8. pg_statio_all_tables View**

칼럼	자료형	설명
relid	oid	테이블의 OID
schemaname	name	이 테이블이 속한 스키마 이름
relname	name	이 테이블의 이름
heap_blks_read	bigint	이 테이블에서 읽힌 디스크 블록의 수
heap_blks_hit	bigint	이 테이블 내 버퍼 히트 수
idx_blks_read	bigint	테이블의 전체 인덱스에서 읽힌 디스크 블록의 수
idx_blks_hit	bigint	테이블 내 모든 인덱스에 대한 버퍼 히트의 수
toast_blks_read	bigint	TOAST 테이블(있으면)에서 읽힌 디스크 블록의 수
toast_blks_hit	bigint	TOAST 테이블(있으면) 내 버퍼 히트 수
tidx_blks_read	bigint	TOAST 테이블 인덱스(있으면)에서 읽힌 디스크 블록 수
tidx_blks_hit	bigint	TOAST 테이블 인덱스(있으면) 내 버퍼 히트 수

pg_statio_all_tables 뷰는 현재 데이터베이스에서 각 테이블당(TOAST 테이블을 포함해서) 한 개의 로우를 갖는데, 해당 테이블의 I/O에 대한 통계치를 보여준다. pg_statio_user_tables와 pg_statio_sys_tables 뷰도 같은 정보를 갖지만 각기 사용자와 시스템 테이블만 보여준다.

표 **11-9. pg_statio_all_indexes View**

칼럼	자료형	설명
relid	oid	인덱스가 포함된 테이블의 OID
indexrelid	oid	인덱스의 OID
schemaname	name	인덱스가 포함된 스키마의 이름
relname	name	인덱스가 포함된 테이블의 이름
indexrelname	name	인덱스명

칼럼	자료형	설명
idx_blks_read	bigint	인덱스로 읽힌 디스크 블록의 수
idx_blks_hit	bigint	인덱스 내 버퍼 히트 수

pg_statio_all_indexes뷰는 현재 데이터베이스의 각 인덱스당 하나의 로우를 갖는데, 해당 인덱스의 I/O에 대한 통계치를 보여준다. pg_statio_user_indexes와 pg_statio_sys_indexes 뷰도 같은 정보를 갖지만 사용자와 시스템 인덱스만 각각 보여준다.

표 **11-10. pg_statio_all_sequences View**

칼럼	자료형	설명
relid	oid	시퀀스의 OID
schemaname	name	시퀀스가 포함된 스키마명
relname	name	시퀀스명
blks_read	bigint	시퀀스에서 읽힌 디스크 블록의 수
blks_hit	bigint	시퀀스 내 버퍼 히트 수

pg_statio_all_sequences 뷰는 현재 데이터베이스의 각 시퀀스당 한 개의 로우를 갖고, 해당 시퀀스의 I/O에 대한 통계치를 보여준다.

표 **11-11. pg_stat_user_functions View**

칼럼	자료형	설명
funcid	oid	함수의 OID
schemaname	name	함수가 포함된 스키마명
funcname	name	함수명
calls	bigint	함수가 호출된 횟수
total_time	double precision	함수와 호출된 함수들에 소요된 밀리세컨드 총 시간
self_time	double precision	호출된 함수들을 제외하고 이 함수에 소요된 밀리세컨드 총 시간

pg_stat_user_functions 뷰는 추적된 각 함수당 한 개의 로우를 가지며, 함수 수행에 대한 통계치를 보여 준다. track-functions는 정확히 어떤 함수가 추적되어야 하는지 정한다.

표 **11-12. pg_stat_replication View**

칼럼	자료형	설명
pid	integer	WAL 송신 프로세스의 프로세스 ID
usesysid	oid	WAL 송신 프로세스에 로그인된 사용자의 OID
usename	name	WAL 송신 프로세스에 로그인된 사용자의 이름

칼럼	자료형	설명
application_name	text	WAL 송신 프로세스에 연결된 애플리케이션명
client_addr	inet	WAL 송신자에 연결된 클라이언트의 IP 주소. 이 필드가 null이면 클라이언트가 서버 머신의 UNIX 소켓으로 연결되었음을 의미함
client_hostname	text	*client_addr*의 DNS를 역조회해서 찾은 클라이언트의 호스트명. 이 필드는 IP로 연결했을 때와 log-hostname이 활성화됐을 때만 null이 아닌 값으로 저장됨
client_port	integer	클라이언트가 WAL 송신자와 커뮤니케이션 하기 위해 쓰는 TCP 포트 넘버. -1 이면 유닉스 소켓이 사용됐을 때
backend_start	timestamp with time zone	이 프로세스가 시작된 시간. 예를 들어 WAL 송신자와 클라이언트가 연결된 시간
state	text	현재 WAL 송신자 상태
sent_location	text	이 연결에 전송된 마지막 트랜잭션 로그 위치
write_location	text	대기 서버가 디스크에 쓴 마지막 트랜잭션 로그 위치
flush_location	text	대기 서버가 디스크로 내린 마지막 트랜잭션 로그 위치
replay_location	text	대기 서버에 있는 데이터베이스로 리플레이 된 마지막 트랜잭션 로그 위치
sync_priority	integer	동기적인 대기 서버로 선택된 대기 서버의 우선순위
sync_state	text	대기 서버의 동기적 상태

pg_stat_replication뷰는 WAL 송신자 프로세스당 하나의 로우를 가지며, 송신자가 연결된 대기 서버의 리플리케이션에 관한 통계치를 보여준다. 직접 연결된 대기 서버만 보여진다. 다운스트림 대기 서버에 관한 정보는 없다.

표 **11-13. pg_stat_database_conflicts View**

칼럼	자료형	설명
datid	oid	데이터베이스의 OID
datname	name	데이터베이스명
confl_tablespace	bigint	드롭된 테이블스페이스로 인해 취소된 쿼리의 수

칼럼	자료형	설명
confl_lock	bigint	잠금 시간초과로 취소된 쿼리의 수
confl_snapshot	bigint	이전 스냅샷으로 취소된 쿼리의 수
confl_bufferpin	bigint	고정된 버퍼로 취소된 쿼리의 수
confl_deadlock	bigint	교착상태로 취소된 쿼리의 수

pg_stat_database_conflicts 뷰는 데이터베이스당 하나의 로우를 가지며, 대기 서버의 복구 충돌로 인해 발생한 쿼리 취소에 대해 전체 데이터베이스 통계치를 보여준다. 마스터 서버에서 충돌이 발생하지 않으므로 이 뷰는 대기 서버의 정보만 갖고 있다.

11.2.3. 통계 함수들

위에서 소개한 뷰를 통해서 서버 통계 정보를 살펴보는 방법과 함께 또 다른 한 방법은 각각의 정보를 제공하는 함수를 사용하는 방법이다. 위에서 소개한 뷰들의 뷰 정의를 살펴보면, 각각의 뷰가 어떤 함수들을 사용하는지 알 수 있다. (예를 들면, psql 에서 `\d+ pg_stat_activity` 명령을 사용하면 된다.) 데이터베이스 단위로 통계 정보를 보여주는 함수들에 대해서는 함수의 입력 인자로 데이터베이스 OID를 입력해야 한다. 이처럼 테이블별, 인덱스별, 함수별 통계 정보를 살펴보려면 각 객체의 OID를 입력 인자로 사용하면 된다. 기억해야 할 점은 테이블, 인덱스, 함수 같은 각 데이터베이스 소속 객체들을 입력 인자로 쓰는 함수면 그 데이터베이스는 현재 접속한 데이터베이스가 된다.

부가적으로 제공하는 통계 관련 함수들은 표 11-14에서 소개한다.

표 **11-14.** 부가 통계 함수들

함수이름	리턴 자료형	설명
`pg_backend_pid()`	integer	현재 세션을 처리하는 서버 프로세스의 프로세스 ID
`pg_stat_get_activity(integer)`	setof record	특정 PID를 갖는 백엔드 정보를 담은 레코드를 리턴하거나 인자가 NULL일 경우에는 시스템에서 활성화된 각 백엔드의 레코드를 리턴함. 리턴된 필드는 pg_stat_activity 뷰의 부분집합
`pg_stat_clear_snapshot()`	void	현재 통계치 스냅샷을 버림
`pg_stat_reset()`	void	현재 데이터베이스의 통계치 카운터 전체를 0으로 만듦(슈퍼 유저 권한이 필요함)

함수이름	리턴 자료형	설명
pg_stat_reset_shared(text)	void	인자에 따라서 전체 클러스터 통계치 카운터를 0으로 만듦(슈퍼 유저 권한이 필요함) pg_stat_reset_shared('bgwriter')를 호출하면 pg_stat_bgwriter뷰에 보여진 전체 카운터들을 0으로 만듦
pg_stat_reset_single_table_counters(oid)	void	현재 데이터베이스의 한 테이블 혹은 인덱스에 대한 통계치를 0으로 만듦(슈퍼 유저 권한이 필요함)
pg_stat_reset_single_function_counters(oid)	void	현재 데이터베이스의 한 함수에 대한 통계치를 0으로 만듦(슈퍼 유저 권한이 필요함)

pg_stat_activity 뷰에서 사용되는 pg_stat_get_activity 함수는 지정한 한 세션의 현재 상태에 대한 모든 정보를 레코드 자료형으로 리턴한다. 이렇게 가끔은 뷰보다 함수를 직접 사용하는 것이 보다 유용할 때가 있다. 위에서 소개한 뷰들(표 11-15)은 함수를 호출하고 그것을 뷰로 보여주기 때문에 뷰를 내용으로 출력 되는 결과는 이미 지난 자료 집합이다 보다 정확한 현재 상태를 파악 하고자 할 때 이런 함수를 직접 사용하는 방법이 좋다. 한 예제를 소개하면, 다음은, 현재 접속해 있는 세션들의 PID와 현재 실행 중인 쿼리 내용을 보는 쿼리다:

```
SELECT pg_stat_get_backend_pid(s.backendid) AS pid,
       pg_stat_get_backend_activity(s.backendid) AS query
    FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS s;
```

표 **11-15.** 백엔드 단위 통계 함수들

함수이름	리턴 자료형	설명
pg_stat_get_backend_idset()	setof integer	현재 활성화된 백엔드 ID 넘버 집합(1부터 활성화된 백엔드 개수까지)
pg_stat_get_backend_activity(integer)	text	백엔드의 최근 쿼리의 텍스트
pg_stat_get_backend_activity_start(integer)	timestamp with time zone	최근 쿼리가 시작된 시간
pg_stat_get_backend_client_addr(integer)	inet	백엔드에 연결된 클라이언트의 IP 주소
pg_stat_get_backend_client_port(integer)	integer	클라이언트가 커뮤니케이션에 쓰고 있는 TCP 포트 넘버
pg_stat_get_backend_dbid(integer)	oid	백엔드가 연결된 데이터베이스의 OID
pg_stat_get_backend_pid(integer)	integer	백엔드의 프로세스 ID
pg_stat_get_backend_start(integer)	timestamp with time zone	프로세스가 시작된 시간

함수이름	리턴 자료형	설명
pg_stat_get_backend_userid(integer)	oid	백엔드에 로그인한 사용자의 OID
pg_stat_get_backend_waiting(integer)	boolean	백엔드가 현재 잠금을 기다리고 있는 경우엔 true
pg_stat_get_backend_xact_start(integer)	timestamp with time zone	현재 트랜잭션이 시작된 시간

11.3. 잠금 보기

데이터베이스 운영 상태를 모니터링 하는 또 다른 도구는 pg_locks 시스템 뷰다. 이 테이블은 잠금 관리자가 현재 처리하고 있는 여러 잠금들에 대한 정보를 제공한다. 이 테이블로 다음과 같은 정보들을 살펴 볼 수 있다:

- 모든 잠금 상황을 보거나 특정 데이터베이스에 속한 객체들의 잠금을 보거나 특정 객체와 관계된 잠금들 또는 Agens SQL 특정 세션에 관계된 잠금들을 볼 수 있다.
- 현재 데이터베이스에서 어떤 세션이 잠금이 필요한데, 다른 세션이 먼저 잠그고 있어 다른 세션들이 대기 상태로 있는지를 조사할 수 있다.
- 잠금 경합 빈도를 살펴 전반적인 서버 성능을 조사할 수 있다. 또한 이런 경합이 데이터베이스 사용량 변화에 어떤 영향을 주는지도 살펴 볼 수 있다.

잠금에 대한 설명은 Agens SQL 동시성 제어에 대한 이해가 선행되어야 쉽게 읽을 수 있다.

11.4. Dynamic Tracing

Agens SQL은 데이터 서버의 dynamic tracing을 지원하는 기능을 제공한다. 코드 특정 지점에서 호출된 외부 유틸리티로 dynamic tracing을 한다.

프로브probe 혹은 trace point는 소스 코드에 이미 삽입돼 있는 경우가 많다. 프로브는 데이터베이스 개발자와 관리자들이 사용하도록 되어 있다. 기본적으로 프로브는 Agens SQL로 컴파일 되지 않는다. 사용자는 설정 스크립트에 프로브를 컴파일 하도록 명시해야 한다.

현재 DTrace (https://en.wikipedia.org/wiki/DTrace) 유틸리티가 지원되고 있는데, 이 다큐멘테이션이 작성되는 시점에는 Solaris, mac OSX, FreeBSD, NetBSD, Oracle Linux에서 사용 가능하다. 리눅스의 SystemTap (http://sourceware.org/systemtap/) 프로젝트로 DTrace와 비슷한 유틸리티를 사용할 수 있다. 이 외의 dynamic tracing 유틸리티는 `src/include/utils/probes.h`에 있는 매크로 정의를 변경하여 사용할 수 있다.

11.4.1. Dynamic Tracing을 위한 컴파일링

기본적으로 프로브는 사용하지 않으므로, 설정 스크립트에 Agens SQL이 프로브를 사용하도록 명시해 주어야 한다. DTrace를 활성화 하기 위해서 `--enable-dtrace`를 명시해 주어야 한다.

11.4.2. Built-in Probes

대부분의 표준 프로브는 표 11-16처럼 소스 코드에서 제공된다. 표 11-17에서 프로브에 사용된 타입을 볼 수 있다. 프로브가 많을수록 Agens SQL의 가관측성을 높일 수 있다.

표 **11-16.** 내장된 **DTrace** 프로브

Name	**Parameters**	설명
transaction-start	(LocalTransactionId)	새로운 트랜잭션이 시작하는 지점에서 발생하는 프로브. 인자는 트랜잭션 ID
transaction-commit	(LocalTransactionId)	트랜잭션이 성공적으로 완료할 때 발생하는 프로브. 인자는 트랜잭션 ID
transaction-abort	(LocalTransactionId)	트랜잭션이 성공적으로 완료되지 않았을 때 발생하는 프로브. 인자는 트랜잭션 ID
query-start	(const char *)	쿼리 처리가 시작됐을 때 발생하는 프로브. 인자는 쿼리 문자열
query-done	(const char *)	쿼리 처리가 끝났을 때 발생하는 프로브. 인자는 쿼리 문자열
query-parse-start	(const char *)	쿼리 파싱이 시작됐을 때 발생하는 프로브. 인자는 쿼리 문자열
query-parse-done	(const char *)	쿼리 파싱이 끝났을 때 발생하는 프로브. 인자는 쿼리 문자열
query-rewrite-start	(const char *)	쿼리를 다시 쓰기 시작할 때 발생하는 프로브. 인자는 쿼리 문자열
query-rewrite-done	(const char *)	쿼리 다시 쓰기가 완료됐을 때 발생하는 프로브. 인자는 쿼리 문자열
query-plan-start	()	쿼리 플래닝이 시작됐을 때 발생하는 프로브
query-plan-done	()	쿼리 플래닝이 끝났을 때 발생하는 프로브
query-execute-start	()	쿼리 수행이 시작됐을 때 발생하는 프로브
query-execute-done	()	쿼리 수행이 끝났을 때 발생하는 프로브
statement-status	(const char *)	서버 프로세스가 pg_stat_activity.`status`를 업데이트할 때마다 발생하는 프로브. 인자는 새로운 상태 문자열

Name	Parameters	설명
checkpoint-start	(int)	체크포인트가 시작되면 발생하는 프로브. 인자는 shutdown나 immediate, force같은 체크포인트 타입을 표시하는 비트 플래그
checkpoint-done	(int, int, int, int, int)	체크포인트가 완료되면 발생하는 프로브(체크포인트 처리 도중에 그 다음 차례인 프로브가 연속적으로 발생함) 첫 번째 인자는 쓰여진 버퍼의 개수. 두 번째 인자는 버퍼의 총 개수. 세 번째, 네 번째, 다섯 번째 인자는 추가되고 삭제되고 재사용된 각 xlog 파일 개수
clog-checkpoint-start	(bool)	체크포인트의 CLOG 부분이 시작됐을 때 발생하는 프로브. 인자를 일반 체크 포인트의 경우 true로, shutdown 체크포인트의 경우 false로 전달
clog-checkpoint-done	(bool)	체크 포인트의 CLOG 부분이 완료되면 발생하는 프로브. 인자는 clog-checkpoint-start인자와 같은 뜻
subtrans-checkpoint-start	(bool)	체크포인트의 SUBTRANS 부분이 시작되면 발생하는 프로브. 인자는 일반 체크포인트의 경우 true로, shutdown 체크포인트의 경우 false로 전달
subtrans-checkpoint-done	(bool)	체크포인트의 SUBTRANS 부분이 끝나면 발생하는 프로브. 인자는 subtrans-checkpoint-start인자와 같은 뜻
multixact-checkpoint-start	(bool)	체크포인트의 MultiXact 부분이 시작되면 발생하는 프로브. 인자는 일반적인 체크포인트의 경우 true로, shutdown 체크포인트의 경우 false로 전달
multixact-checkpoint-done	(bool)	체크포인트의 MultiXact부분이 완료되면 발생하는 프로브. 인자는 multixact-checkpoint-start인자와 같은 뜻

Name	Parameters	설명
buffer-checkpoint-start	(int)	체크포인트의 버퍼 쓰기 부분이 시작되면 발생하는 프로브. 인자는 shutdown, immediate, force같은 체크포인트 타입을 구별하기 위한 비트 플래그
buffer-sync-start	(int, int)	체크포인트 중간에(어떤 버퍼가 쓰여야 하는지 식별한 후) dirty 버퍼를 쓰기 시작할 때 발생하는 프로브. 첫 번째 인자는 버퍼 총 개수를 뜻함. 두 번째 인자는 현재 dirty하고 쓰여져야 하는 버퍼의 개수를 뜻함
buffer-sync-written	(int)	체크포인트 중간에 버퍼가 쓰여질 때마다 발생하는 프로브. 인자는 버퍼의 ID 넘버
buffer-sync-done	(int, int, int)	모든 dirty 버퍼가 쓰여졌을 때 발생하는 프로브. 첫 번째 인자는 버퍼의 총 개수를 의미. 두 번째 인자는 체크포인트 프로세스가 실제로 쓴 버퍼의 개수. 세 번째 인자는 쓰기로 예상된 버퍼의 넘버(buffer-sync-start의 두 번째 인자).
buffer-checkpoint-sync-start	()	dirty 버퍼가 커널에 쓰여진 이후와 fsync 요청을 보내기 이전에 발생한 프로브
buffer-checkpoint-done	()	버퍼와 디스크 동기화가 완료될 때 발생하는 프로브
twophase-checkpoint-start	()	체크포인트 두 단계가 시작됐을 때 발생하는 프로브
twophase-checkpoint-done	()	체크포인트의 두 단계 부분이 완료됐을 때 발생하는 프로브

Name	Parameters	설명
buffer-read-start	(ForkNumber, BlockNumber, Oid, Oid, Oid, int, bool)	버퍼 읽기가 시작될 때 발생하는 프로브. 첫 번째 인자와 두 번째 인자는 페이지의 포크와 블록 넘버를 전달(두 번째 인자는 릴레이션 익스텐션relation extension 요청일 경우 -1). 세 번째, 네 번째, 다섯 번째 인자는 테이블스페이스와 데이터베이스, 릴레이션의 식별자인 릴레이션 OID를 뜻함. 여섯 번째 인자는 로컬 버퍼에 임시 릴레이션을 만든 백엔드의 ID 이거나 공유 버퍼의 InvalidBackendID (-1)을 뜻함. 일곱 번째 인자는 릴레이션 익스텐션을 요청할 때 true이고, 일반 읽기 작업에는 false를 전달함.
buffer-read-done	(ForkNumber, BlockNumber, Oid, Oid, Oid, int, bool, bool)	버퍼 읽기를 완료했을 때 발생하는 프로브. 첫 번째 및 두 번째 인자는 페이지의 포크와 블록 넘버를 전달(릴레이션 익스텐션 요청의 경우 두 번째 인자는 새로 추가된 블록의 블록 넘버를 가짐). 세 번째, 네 번째, 다섯 번째 인자는 테이블스페이스, 데이터베이스, 릴레이션을 식별하는 릴레이션 OID를 뜻함. 여섯 번째 인자는 로컬 버퍼에 임시 릴레이션을 만든 백엔드의 ID 이거나 공유 버퍼의 InvalidBackendID (-1)을 뜻함. 일곱 번째 인자는 릴레이션 익스텐션을 요청할 때 true이고, 일반 읽기 작업에는 false를 전달함. 여덟 번째 인자는 풀에서 버퍼를 발견하면 true이고 아니면 false를 전달함.
buffer-flush-start	(ForkNumber, BlockNumber, Oid, Oid, Oid)	공유 버퍼에 쓰기 요청을 하기 전에 발생하는 프로브. 첫 번째 및 두 번째 인자는 페이지의 포크 및 블록 넘버를 전달. 세 번째, 네 번째, 다섯 번째 인자는 테이블스페이스, 데이터베이스, 릴레이션을 식별하는 릴레이션 OID를 전달함

Name	Parameters	설명
buffer-flush-done	(ForkNumber, BlockNumber, Oid, Oid, Oid)	쓰기 요청이 완료됐을 때 발생하는 프로브. (데이터를 커널에 전달하는 시간만 알려주며, 디스크에 쓰여지지 않았을 때임) 인자들은 buffer-flush-start와 같음
buffer-write-dirty-start	(ForkNumber, BlockNumber, Oid, Oid, Oid)	서버 프로세스가 dirty 버퍼를 쓸 때 발생하는 프로브. (자주 발생하면 shared-buffers가 너무 작거나 bgwriter control 매개변수가 조정이 필요함을 뜻함) 첫 번째와 두 번째 인자는 페이지의 포크 및 블록 넘버를 전달함. 세 번째, 네 번째, 다섯 번째 인자는 테이블스페이스, 데이터베이스, 릴레이션 식별하는 릴레이션 OID를 전달함
buffer-write-dirty-done	(ForkNumber, BlockNumber, Oid, Oid, Oid)	dirty-buffer 쓰기가 완료됐을 때 발생하는 프로브. 인자는 buffer-write-dirty-start와 같음
wal-buffer-write-dirty-start	()	WAL 버퍼 공간이 남아 있지 않아서 서버 프로세스가 dirty WAL 버퍼에 쓰기 시작할 때 발생하는 프로브(자주 발생하면 wal-buffers가 너무 작다는 뜻임)
wal-buffer-write-dirty-done	()	dirty WAL 버퍼 쓰기가 완료됐을 때 발생하는 프로브
xlog-insert	(unsigned char, unsigned char)	WAL 레코드가 삽입됐을 때 발생하는 프로브. 첫 번째 인자는 해당 레코드의 리소스 관리자(rmid)임. 두 번째 인자는 info 플래그를 전달함
xlog-switch	()	WAL 세그먼트 스위치를 요청했을 때 발생하는 프로브

Name	Parameters	설명
smgr-md-read-start	(ForkNumber, BlockNumber, Oid, Oid, Oid, int)	릴레이션에서 블록을 읽기 시작할 때 발생하는 프로브. 첫 번째 및 두 번째 인자는 페이지의 포크 및 블록 넘버를 전달함. 세 번째, 네 번째, 다섯 번째 인자는 테이블스페이스, 데이터베이스, 릴레이션을 식별하는 릴레이션 OID를 전달. 여섯 번째 인자는 로컬 버퍼에 임시 릴레이션을 만든 백엔드의 ID 이거나 공유 버퍼의 InvalidBackendID (-1)을 뜻함.
smgr-md-read-done	(ForkNumber, BlockNumber, Oid, Oid, Oid, int, int, int)	블록 읽기가 완료됐을 때 발생하는 프로브. 첫 번째 및 두 번째 인자는 페이지의 포크 및 블록 넘버를 전달함. 세 번째, 네 번째, 다섯 번째 인자는 테이블스페이스, 데이터베이스, 릴레이션을 식별하는 릴레이션 OID를 전달. 여섯 번째 인자는 로컬 버퍼에 임시 릴레이션을 만든 백엔드의 ID 이거나 공유 버퍼의 InvalidBackendID (-1)을 뜻함. 일곱 번째 인자는 실제로 읽힌 바이트 수를 뜻하고, 여덟 번째 인자는 요청된 숫자를 뜻함(이 두 개가 다르면 문제가 발생)
smgr-md-write-start	(ForkNumber, BlockNumber, Oid, Oid, Oid, int)	릴레이션에 블록을 쓸 때 발생하는 프로브. 첫 번째 및 두 번째 인자는 페이지의 포크 및 블록 넘버를 전달함. 세 번째, 네 번째, 다섯 번째 인자는 테이블스페이스, 데이터베이스, 릴레이션을 식별하는 릴레이션 OID를 전달. 여섯 번째 인자는 로컬 버퍼에 임시 릴레이션을 만든 백엔드의 ID 이거나 공유 버퍼의 InvalidBackendID (-1)을 뜻함.

Name	Parameters	설명
smgr-md-write-done	(ForkNumber, BlockNumber, Oid, Oid, Oid, int, int, int)	블록 쓰기가 완료됐을 때 발생하는 프로브. 첫 번째 및 두 번째 인자는 페이지의 포크 및 블록 넘버를 전달함. 세 번째, 네 번째, 다섯 번째 인자는 테이블스페이스, 데이터베이스, 릴레이션을 식별하는 릴레이션 OID를 전달. 여섯 번째 인자는 로컬 버퍼에 임시 릴레이션을 만든 백엔드의 ID 이거나 공유 버퍼의 InvalidBackendID (-1)을 뜻함. 일곱 번째 인자는 실제로 쓰여진 바이트 수를 뜻하고 여덟 번째 인자는 요청된 숫자를 뜻함(이 두 개가 다르면 문제가 발생)
sort-start	(int, bool, int, int, bool)	정렬 작업이 시작됐을 때 발생하는 프로브. 첫 번째 인자는 heap, 인덱스, 데이터 정렬을 뜻함. 두 번째 인자는 unique-value를 시행할 경우 true를 전달. 세 번째 인자는 키 칼럼의 수이고, 네 번째 인자는 허용된 워크 메모리의 킬로바이트 수를 뜻하고, 다섯 번째 인자는 정렬 결과에 랜덤 액세스가 요청됐을 때 true를 전달함.
sort-done	(bool, long)	정렬이 완료됐을 때 발생하는 프로브. 첫 번째 인자는 외부 정렬일 경우 true이고, 내부 정렬일 경우에는 false임. 두 번째 인자는 외부 정렬에 사용된 디스크 블록의 수이거나 내부 정렬에 사용된 메모리의 킬로바이트 수를 뜻함
lwlock-acquire	(LWLockId, LWLockMode)	LWLock이 획득됐을 때 발생하는 프로브. 첫 번째 인자는 LWLock의 ID. 두 번째 인자는 요청된 잠금 모드로 독점적이거나 공유됨
lwlock-release	(LWLockId)	LWLock이 풀렸을 때 발생하는 프로브(waiter가 아직 깨어있지 않을 때). 인자는 LWLock의 ID임.

Name	Parameters	설명
lwlock-wait-start	(LWLockId, LWLockMode)	LWLock이 즉시 가용하지 않고, 서버 프로세스가 잠금이 가용해질 때까지 기다리기 시작했을 때 발생하는 프로브. 첫 번째 인자는 LWLock의 ID이고, 두 번째 인자는 요청된 잠금 모드로 독점적이거나 공유됨
lwlock-wait-done	(LWLockId, LWLockMode)	서버 프로세스가 LWLock 대기 상태에서 풀려났을 때(잠금을 아직 획득하지 않은 상태) 발생하는 프로브. 첫 번째 인자는 LWLock의 ID임. 두 번째 인자는 요청된 잠금 모드로 독점적이거나 공유됨
lwlock-condacquire	(LWLockId, LWLockMode)	호출자가 대기 하지 않겠다고 하고, LWLock을 획득했을 때 발생하는 프로브. 첫 번째 인자는 LWLock의 ID임. 두 번째 인자는 요청된 잠금 모드로 독점적이거나 공유됨
lwlock-condacquire-fail	(LWLockId, LWLockMode)	호출자가 대기 하지 않겠다고 했는데, LWLock을 획득하지 못했을 때 발생하는 프로브. 첫 번째 인자는 LWLock의 ID임. 두 번째 인자는 요청된 잠금 모드로 독점적이거나 공유됨
lock-wait-start	(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, LOCKMODE)	잠금이 가용하지 않아서 heavyweight 잠금(lmgr lock) 요청이 대기하기 시작했을 때 발생하는 프로브. 첫 번째부터 네 번째까지의 인자들은 오브젝트가 잠겼는지 식별하는 태그 필드들임. 다섯 번째 인자는 잠긴 오브젝트 타입을 가리키고, 여섯 번째 인자는 요청된 잠금 타입을 가리킴.
lock-wait-done	(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, LOCKMODE)	heavyweight 잠금(lmgr lock) 요청의 대기 상태가 끝났을 때(예를 들어 잠금을 획득한 경우) 발생하는 프로브. 인자들은 lock-wait-start의 인자와 같음
deadlock-found	()	교착상태 감지자가 교착상태를 발견했을 때 발생하는 프로브

표 **11-17.** 프로브 매개변수에 사용된 정의된 자료형

자료형	정의
LocalTransactionId	무부호 int
LWLockId	int
LWLockMode	int
LOCKMODE	int
BlockNumber	무부호 int
Oid	무부호 int
ForkNumber	int
bool	char

11.4.3. Using Probes

아래 예시는 트랜잭션 수를 분석하는 DTrace 스크립트로써 성능 테스트 이전과 이후에 pg_stat_database의 스냅샷을 찍는 역할을 한다.

```
#!/usr/sbin/dtrace -qs

postgresql$1:::transaction-start
{
      @start["Start"] = count();
      self->ts  = timestamp;
}

postgresql$1:::transaction-abort
{
      @abort["Abort"] = count();
}

postgresql$1:::transaction-commit
/self->ts/
{
      @commit["Commit"] = count();
      @time["Total time (ns)"] = sum(timestamp - self->ts);
      self->ts=0;
}
```

수행 하면 위 D 스크립트는 다음과 같이 출력한다.

```
# ./txn_count.d `pgrep -n postgres` or ./txn_count.d <PID>
^C

Start                                          71
Commit                                         70
Total time (ns)                        2312105013
```

> 참고: SystemTap은 trace point가 호환돼도 DTrace와 다른 trace 스크립트 기호를 사용한다. SystemTap 스크립트는 하이픈 기호 대신 이중 언더스코어 기호를 사용해서 프로브 이름을 레퍼런싱해야 한다. 이는 향후 SystemTap 버전에서 개선될 예정이다.

DTrace 스크립트는 쓰거나 디버깅할 때 주의해야 한다. 수집된 trace 정보의 의미가 사라질 수 있기 때문이다. 문제가 발생하는 대부분의 경우, 시스템 내부가 아니라 장치가 원인이다. Dynamic tracing을 사용해서 정보를 얻을 때, DTrace 스크립트도 체크하고 확인해야 한다.

더 많은 스크립트 예제는 PgFoundry dtrace project (http://pgfoundry.org/projects/dtrace/)에서 참조할 수 있다.

11.4.4. Defining New Probes

개발자가 원하는 위치 내에서 새로운 프로브를 정의할 수 있는데, 재컴파일이 필요하다. 아래는 새로운 프로브를 삽입하는 과정이다.

1. 1. 프로브 이름과 프로브로 가용한 데이터를 정하라.
2. 2. `src/backend/utils/probes.d`에 프로브 정의를 추가하라.
3. 3. 프로브 point가 있는 모듈에 `pg_trace.h`가 없으면 포함시키고, `TRACE_POSTGRESQL`프로브 매크로를 소스 코드의 원하는 위치에 넣는다.
4. 4. 다시 컴파일하고 새 프로브가 가용한지 확인한다.

Example: 트랜잭션 ID로 모든 새로운 트랜잭션을 추적하기 위해 프로브를 추가하는 예제이다.

1. 1. 이름은 `transaction-start` 이고 LocalTransactionId 타입인 매개변수를 갖는 프로브를 정하라.
2. 2. 프로브 정의를 `src/backend/utils/probes.d`에 추가하라.

   ```
   probe transaction__start(LocalTransactionId);
   ```

   프로브 이름 안의 이중 언더스코어가 있음을 유의하자. 프로브를 사용한 DTrace 스크립트에서 이중 밑줄은 하이픈으로 대체되므로 사용자 문서 이름은 `transaction-start`다.
3. 3. 컴파일 시점에 `transaction__start`는 `TRACE_POSTGRESQL_TRANSACTION_START`라 불리는 매크로로 바뀐다(언더스코어가 각각 한 개씩 쓰이는 것을 주의하자). 이 매크로는 `pg_trace.h`를 포함했을 때 가용하다. 매크로 호출 부분을 소스 코드의 적당한 위치에 넣는다. 예를 들면 다음과 같다.

   ```
   TRACE_POSTGRESQL_TRANSACTION_START(vxid.localTransactionId);
   ```
4. 4. 재컴파일링과 새 바이너리 파일을 실행하면, 아래 DTrace 명령어를 실행했을 때 새로 추가된 프로브가 가용하다는 것을 알 수 있다. 아래와 비슷한 출력을 확인할 수 있을 것이다.

   ```
   # dtrace -ln transaction-start
      ID    PROVIDER          MODULE           FUNCTION NAME
   18705 postgresql49878     postgres     StartTransactionCommand transaction-start
   18755 postgresql49877     postgres     StartTransactionCommand transaction-start
   18805 postgresql49876     postgres     StartTransactionCommand transaction-start
   18855 postgresql49875     postgres     StartTransactionCommand transaction-start
   18986 postgresql49873     postgres     StartTransactionCommand transaction-start
   ```

trace 매크로를 C코드에 추가할 때 몇 가지 주의할 점이 있다.

- 프로브의 매개변수로 명시된 데이터 타입이 매크로 변수의 데이터 타입과 일치해야 한다. 그렇지 않으면 컴파일 에러가 발생할 것이다.
- 대부분의 플랫폼의 경우, Agens SQL이 `--enable-dtrace`로 설치되면, trace 매크로 인자들을 컨트롤할 때마다 **trace**하지 않아도 검증된다. 몇 개의 로컬 변수 값을 확인하는 목적이라면 크게 신경 쓰지 않아도 된다. 하지만 인자에 고비용의 함수 호출을 넣는 것은 주의해야 한다. 그런 경우에는 trace가 실제로 활성화 되었는지 확인해서 매크로를 보호하는 것이 좋다.

  ```
  if (TRACE_POSTGRESQL_TRANSACTION_START_ENABLED())
      TRACE_POSTGRESQL_TRANSACTION_START(some_function(...));
  ```

  각 trace 매크로에는 `ENABLED` 매크로가 있다.

12장. 디스크 사용량 모니터링

이 장에서는 Agens SQL 데이터베이스 시스템의 디스크 사용량을 모니터링하는 방법을 다룬다.

12.1. 디스크 사용량 결정

각 테이블은 대부분의 데이터가 저장되는 1차 힙 디스크 파일을 갖고 있다. 테이블 칼럼의 값이 잠재적으로 큰 범위인 경우 메인 테이블에 잘 맞지 않는 값을 저장하는 데 사용되고 테이블에 연결된 TOAST 파일이 있을 수도 있다. 그런 경우에는, 유효 인덱스 1개가 TOAST 테이블에 존재한다. 또한 베이스 테이블에 연결된 인덱스도 있을 수 있다. 각 테이블 및 인덱스는 개별 디스크 파일에 저장된다. 파일 크기가 1기가바이트를 초과하는 경우에는 파일이 하나 이상일 수 있다.

디스크 공간은 3가지 방법으로 모니터링할 수 있는데, SQL 함수를 사용하거나, oid2name 모듈을 사용하거나, 시스템 카탈로그 수동 검사를 사용할 수 있다. SQL 함수는 사용법이 매우 쉬워 보편적으로 권장된다. 이 절의 나머지 부분에서는 시스템 카탈로그를 검사하는 방법에 대해 설명한다.

최근 vacuum되었거나 분석된 데이터베이스에서 psql을 사용하면 쿼리를 실행하여 임의의 테이블에서 디스크 사용량을 볼 수 있다.

```
SELECT pg_relation_filepath(oid), relpages FROM pg_class WHERE relname = 'customer';
```

```
 pg_relation_filepath | relpages
----------------------+----------
 base/16384/16806     |       60
(1 row)
```

각 페이지는 일반적으로 8킬로바이트이다. (*relpages*는 **VACUUM** 및 **ANALYZE**, **CREATE INDEX** 같은 몇 가지 DDL 명령으로만 업데이트된다.) 파일 경로명은 테이블의 디스크 파일을 직접 검사하려는 경우에 중요하다.

TOAST 테이블에서 사용하는 공간을 표시하려면 다음과 같은 쿼리를 사용한다.

```
SELECT relname, relpages
FROM pg_class,
     (SELECT reltoastrelid
      FROM pg_class
      WHERE relname = 'customer') AS ss
WHERE oid = ss.reltoastrelid OR
      oid = (SELECT indexrelid
             FROM pg_index
             WHERE indrelid = ss.reltoastrelid)
ORDER BY relname;
```

```
       relname        | relpages
----------------------+----------
 pg_toast_16806       |        0
 pg_toast_16806_index |        1
```

인덱스 크기를 손쉽게 표시할 수도 있다.

```
SELECT c2.relname, c2.relpages
FROM pg_class c, pg_class c2, pg_index i
WHERE c.relname = 'customer' AND
      c.oid = i.indrelid AND
      c2.oid = i.indexrelid
ORDER BY c2.relname;

       relname        | relpages
----------------------+----------
 customer_id_indexdex |       26
```

이 정보를 이용하면 최대 크기의 테이블과 인덱스를 찾을 때 용이하다.

```
SELECT relname, relpages
FROM pg_class
ORDER BY relpages DESC;

       relname        | relpages
----------------------+----------
 bigtable             |     3290
 customer             |     3144
```

12.2. 디스크 꽉 참(Full) 실패

데이터베이스 관리자의 가장 중요한 디스크 모니터링 작업은 디스크가 꽉 차지 않게 관리하는 것이다. 채워진 데이터 디스크가 데이터 손상으로 이어지지는 않지만 작동에 지장을 줄 수 있다. WAL 파일이 저장된 디스크가 꽉 차게 되면 데이터베이스 서버에 혼란이 생기고, 결과적으로 셧다운이 발생할 수 있다.

다른 데이터를 삭제함으로써 추가 공간을 확보하기 어려운 경우 테이블스페이스를 사용하여 데이터베이스 파일 일부를 다른 파일 시스템으로 옮길 수 있다. 자세한 내용은 5.6절을 참조 바란다.

> 작은 정보: 파일 시스템 중 일부는 용량이 꽉 차게 되면 작동에 문제가 생기므로 디스크가 꽉 찰 때까지 기다리면 안 된다.

사용자별 디스크 한도가 지원되는 시스템인 경우 서버를 실행하는 사용자에게 부여된 한도에 따라 데이터베이스가 자연스럽게 적용된다. 한도를 초과하면 디스크 공간을 완전히 초과했을 때와 동일한 효과가 나타난다.

13장. 안정성 및 Write-Ahead 로그

이 장에서는 효율적이고 안정된 운용을 위해 Write-Ahead 로그를 사용하는 방법에 대해 설명한다.

13.1. 안정성

안정성은 중요 데이터베이스 시스템에서 중요한 성질이며 Agens SQL은 안정된 실행을 보장하기 위해 할 수 있는 모든 것을 한다. 안정된 실행의 한 가지 측면은, 커밋된 트랜잭션에 의해 기록된 모든 데이터를 비휘발성 영역에 저장해서 정전, 운영 체제 실패 및 하드웨어 고장의 영향으로부터 안전하도록 보장하는 것이다 (비휘발성 영역 자체의 실패는 제외). 컴퓨터의 영구적 저장소(디스크 드라이브 등)에 데이터를 성공적으로 기록하면 일반적인 이러한 요구조건에 부합된다. 사실, 컴퓨터가 치명적으로 손상되었더라도 디스크 드라이브가 망가지지 않아서 유사 하드웨어의 다른 컴퓨터로 옮길 수만 있다면, 모든 커밋된 트랜잭션은 온전하게 유지할 수 있다.

데이터를 디스크 플래터에 강제로 쓰는 것은 간단한 명령처럼 보이지만 그렇지 않다. 디스크 드라이브가 메인 메모리와 CPU보다 훨씬 느려지므로 컴퓨터의 메인 메모리와 디스크 플래터 사이에 몇 개의 캐싱 레이어가 존재한다. 첫째, 요청 빈도가 높은 디스크 블록을 캐시하고 디스크 쓰기를 결합하는 운영 체제의 버퍼 캐시가 있다. 다행히도, 모든 운영 체제는 버퍼 캐시에서 디스크로 강제로 쓰는 애플리케이션을 제공하며, Agens SQL은 해당 기능을 사용한다. (이것의 조정 방법은 wal_sync_method 매개 변수를 참조 바란다.)

다음으로, 디스크 드라이브 컨트롤러에 캐시가 있을 수 있다. 이것은 특히 RAID 컨트롤러 카드에서 공통된 사항이다. 이러한 캐시 중 일부는, 쓰기가 도착하는 즉시 드라이브에 전송되는 *write-through*이다. 다른 캐시들은, 데이터가 시간차를 두고 드라이브에 전송되는 *write-back*이다. 디스크 컨트롤러 캐시의 메모리는 휘발성이라서 정전 시 내용이 삭제될 수 있으므로 해당 캐시는 안정성에 문제가 있을 수 있다. 더 뛰어난 컨트롤러 카드는, 정전 시 캐시 전력을 유지하기 위한 배터리가 포함된 카드를 의미하는 *battery-backup units*(BBU)를 갖고 있다. 전원이 복구된 후 데이터가 디스크 드라이브에 쓰기 된다.

그리고 마지막으로, 대부분의 디스크 드라이브는 캐시를 갖고 있다. 일부는 write-through이고, 또 일부는 write-back이며, write-back 드라이브 캐시에 대한 데이터 손실 우려는 디스크 컨트롤러 캐시와 동일하다. 소비자 수준의 IDE 및 SATA 드라이브의 캐시는 특히 정전 시 살아남지 못하는 write-back일 가능성이 있다. 다수의 solid-state drives(SSD) 역시 휘발성 write-back 캐시이다.

이러한 캐시는 일반적으로 비활성화된다. 단, 이렇게 하는 방법은 운영 체계 및 드라이브 유형에 따라 다르다.

- Linux에서 IDE 및 SATA 드라이브는 **hdparm -I**를 사용하여 쿼리할 수 있다. 쓰기 캐싱은 `Write cache` 다음에 `*`가 있는 경우에 활성화된다. **hdparm -W 0**은 쓰기 캐싱을 끌 때 사용할 수 있다. SCSI 드라이브는 sdparm (http://sg.danny.cz/sg/sdparm.html)을 사용하여 쿼리할 수 있다. 쓰기 캐시가 활성화되었는지, **sdparm --get=WCE**가 그것을 비활성화하는지를 확인하려면 **sdparm --clear=WCE**를 사용해야 한다.
- FreeBSD에서 IDE 드라이브는 **atacontrol**을 사용하여 쿼리할 수 있고, 쓰기 캐시는 `/boot/loader.conf`에서 `hw.ata.wc=0`을 사용하여 끌 수 있다. SCSI 드라이브는 **camcontrol identify**를 사용하여 쿼리할 수 있고, 사용 가능한 경우 **sdparm**을 사용하여 쓰기 캐시를 쿼리하고 변경할 수 있다.
- Solaris에서 디스크 쓰기 캐시는 **format -e**로 제어된다. (Solaris ZFS 파일 시스템은 자체적인 디스크 캐시 쓰기 명령을 실행하므로 디스크 쓰기 캐시가 활성화된 상태에서 안전하다.)

- Windows에서 `wal_sync_method`가 `open_datasync`(기본값)인 경우 `My Computer\Open\`*`disk drive`*`\Properties\Hardware\Properties\Policies\Enable write caching on the disk`를 선택 해제하면 쓰기 캐싱을 비활성화할 수 있다. 또는, `wal_sync_method`를 `fsync` 또는 `fsync_writethrough`로 설정해서 쓰기 캐싱을 금지해야 한다.
- OS X에서 `wal_sync_method`를 `fsync_writethrough`로 설정하면 쓰기 캐싱을 금지할 수 있다.

최신 SATA 드라이브(후속 ATAPI-6 이상)는 드라이브 캐시 쓰기 명령(**FLUSH CACHE EXT**)을 제공하며, SCSI 드라이브는 길지만 유사한 지원 명령 **SYNCHRONIZE CACHE**를 제공한다. 이러한 명령은 Agens SQL에 직접 액세스할 수 없지만 일부 파일 시스템(예: ZFS, ext4)은 이러한 명령을 사용하여 write-back 가능 드라이브의 플래터에 데이터를 쓸 수 있다. 배터리 백업 유닛(BBU) 디스크 컨트롤러가 결합된 경우에는 아쉽게도 해당 파일 시스템은 최적이 아닌 준최적으로 동작한다. 해당 설정에서 동기화 명령은 컨트롤러 캐시의 모든 데이터를 디스크에 강제로 쓰기 때문에 BBU의 여러 가지 장점이 상쇄된다. 영향이 있을 경우 pg_test_fsync 프로그램을 실행하여 확인해 볼 수 있다. 영향이 있을 경우 이것이 옵션이라면 파일 시스템의 쓰기 차단을 해제하거나 디스크 컨트롤러를 다시 환경 설정함으로써 BBU의 성능상의 장점은 다시 활용할 수 있다. 쓰기 차단을 해제한 경우 배터리가 작동되는 상태인지 확인해야 한다. 배터리 고장 시 데이터가 손실될 가능성이 있다. 다행히 파일 시스템 및 디스크 컨트롤러 설계자는 결국 이러한 준최적 동작을 해결하려고 할 것이다.

운영 체제가 쓰기 요청을 저장소 하드웨어로 전송한 경우 비 휘발성 저장소 영역에 데이터가 정말로 도착했는지 확인할 방법이 없다. 오히려, 모든 저장소 환경 설정요소가 데이터 및 파일 시스템 메타데이터 모두에 대한 무결성을 보장하는지를 확인하는 것은 관리자의 책임이다. 배터리 백업 쓰기 캐시가 없는 디스크 컨트롤러는 피해야 한다. 드라이브 수준에서, 셧다운 전에 데이터가 쓰기될 것이라는 보장을 드라이브가 할 수 없으면 write-back 캐시를 비활성화해야 한다. SSD를 사용하는 경우에는 다수가 기본적으로 캐시 쓰기 명령을 이행하지 않는다는 점에 유의해야 한다. I/O 서브 시스템의 동작을 믿을 수 있는지는 `diskchecker.pl` (http://brad.livejournal.com/2116715.html)을 사용하여 테스트할 수 있다.

데이터 손실의 또 다른 위험은 디스크 플래터 쓰기 명령 자체이다. 디스크 플래터는 일반적으로 각각 512바이트인 섹터로 분할된다. 모든 물리적 읽기 또는 쓰기 명령은 전체 섹터를 처리한다. 쓰기 요청이 드라이브에 도착하면 이것은 512바이트 배수에 대한 것일 수 있으며(Agens SQL 는 보통 한 번에 8192 바이트나 16섹터를 쓰기한다.), 쓰기 프로세스는 언제든 정전 때문에 실패할 수 있다. 이것은 512바이트 섹터의 일부는 쓰기가 되었고, 나머지는 쓰기가 되지 않았다는 것을 의미한다. 이러한 실패를 방지하기 위해, Agens SQL은 디스크에서 실제 페이지를 수정하기 전에 전체 페이지 이미지를 영구적 WAL 저장소에 주기적으로 기록한다. 이렇게 함으로써 충돌 복구 도중에 Agens SQL은 부분적으로 쓰기된 페이지를 WAL로부터 복구할 수 있다. 부분적 페이지 쓰기를 방지하는 파일 시스템 소프트웨어가 있는 경우(예: ZFS), full_page_writes 매개 변수를 해제하여 이러한 페이지 이미징을 해제할 수 있다. 배터리 백업 유닛(BBU) 디스크 컨트롤러는 전체(8kB) 페이지로 데이터를 BBU에 쓰도록 보장하지 않는 한 부분적 페이지 쓰기를 막지 않는다.

Agens SQL은 하드웨어 에러 또는 읽기/쓰기 가비지 데이터 같이 시간 경과에 따른 매체 실패 때문에 발생할 수 있는 저장 장치의 몇 가지 데이터 손상도 보호한다.

- WAL 파일의 개별 레코드는, 레코드 내용이 올바른지를 사용자에게 알려주는 CRC-32(32비트) 검사로 보호된다. CRC 값은 각 WAL 레코드를 쓰는 경우에 설정되고, 충돌 복구, 아카이브 복구 및 복제 중에 검사된다.
- WAL 레코드에 기록된 전체 페이지 이미지가 보호되더라도 데이터 페이지는 기본적으로 현재 체크섬되지 않는다.
- `pg_clog`, `pg_subtrans`, `pg_multixact`, `pg_serial`, `pg_notify`, `pg_stat`, `pg_snapshots` 같은 내부 데이터 구조는 직접적으로 체크섬되지 않거나, 전체 페이지 쓰기에 의해 보호되

는 페이지가 아니다. 그러나, 이러한 데이터 구조가 지속적인 경우, WAL 레코드는 최근의 변경 사항이 충돌 시에 정확하게 리빌드되도록 하고, 이러한 WAL 레코드는 위에서 언급된 대로 보호된다.

- `pg_twophase`에서 개별 상태 파일은 CRC-32로 보호된다.
- 거대(large) SQL 쿼리에서 정렬, 구체화 및 중간 결과를 위해 사용되는 임시 데이터 파일은 현재 체크섬되지 않으며, 이러한 파일의 변경 내용에 대해 WAL 레코드가 기록되지 않는다.

Agens SQL은 수정 가능한 메모리 에러를 보호하지 않으며, 산업 표준 Error Correcting Codes(ECC) 또는 더 나은 보호를 사용하는 RAM을 사용자가 활용할 것이라고 가정한다.

13.2. Write-Ahead 로깅(WAL)

Write-Ahead 로깅 (WAL)은 데이터 무결성을 보장하는 표준 방법이다. 자세한 설명은 트랜잭션 프로세싱에 대한 대부분의 책자에 나와 있다(전부는 아니지만). 짧게 말해서, WAL의 중심 개념은 변경을 로깅한 후에만, 즉 변경 내용을 설명하는 로그 레코드를 영구적 저장소에 먼저 기록한 후에 데이터 파일(테이블과 인덱스가 있는)의 변경 내용을 작성해야 한다는 것이다. 이 절차를 준수한 경우 충돌 발생 시 로그를 사용하여 데이터베이스를 복수할 수 있으므로 트랜잭션 커밋마다 데이터 페이지를 디스크에 쓸 필요가 없다. 데이터 페이지에 적용되지 않은 변경 내용은 로그 레코드에서 실행 취소가 가능하다. (이것은 롤포워드roll-forward 복구이며, REDO라고도 한다.)

> 작은 정보: WAL은 충돌 후 데이터베이스 파일 내용을 복구하므로 데이터 파일 또는 WAL 파일의 안정적인 저장소의 경우 저널링된 파일 시스템은 불필요하다. 사실, 저널링 오버헤드는 특히 저널링이 파일 시스템 데이터를 디스크에 쓰게 하는 경우, 성능에 역효과가 난다. 다행히도 저널링 중 데이터 쓰기는 파일 시스템 마운트 옵션(예: Linux ext3 파일 시스템의 경우 `data=writeback`)을 사용하여 빈번하게 비활성화된다.

트랜잭션에 의해 변경된 모든 데이터 파일보다는 트랜잭션이 커밋된 것을 보장하기 위해 로그 파일만 디스크에 써야 하기 때문에 WAL을 사용하면 디스크 쓰기 수가 상당히 줄어든다. 로그 파일은 순차적으로 작성되며, 따라서 로그 파일 동기화 비용은 데이터 페이지 쓰기 비용보다 훨씬 적다. 이것은 특히 서버가 데이터 스토어의 서로 다른 부분을 건드리는 소규모 트랜잭션을 다수 처리하는 경우에 그렇다. 또한, 서버가 소규모 동시 트랜잭션을 다수 처리하는 경우에 로그 파일의 `fsync` 하나로 여러 가지 트랜잭션을 충분히 커밋할 수 있다.

WAL은 또한 8.3절에 설명된 대로 온라인 백업 및 PIT(point-in-time) 복구를 지원을 가능하게 한다. WAL 데이터를 아카이빙함으로써, 가용한 WAL 데이터가 즉각 커버하는 시점으로 복귀를 지원할 수 있다. 간단히 데이터베이스의 실제 백업을 설치하고 원하는 시간만큼 WAL 로그를 리플레이하면 된다. 무엇보다, 실제 백업은 데이터베이스 상태의 즉각적인 스냅샷일 필요는 없다. 이것이 일정 기간에 걸쳐 이루어진 경우 해당 기간에 대한 WAL 로그를 리플레이하면 내부 불일치가 해결된다.

13.3. 비동기 커밋

*비동기 커밋*은 데이터베이스가 충돌한 경우 가장 최근 트랜잭션이 분실될 수도 있는 대신, 트랜잭션을 좀 더 빨리 완료할 수 있는 옵션이다. 다수의 애플리케이션에서 이것은 수용 가능한 트레이드오프이다.

앞 절에서 설명한 대로 트랜잭션 커밋은 일반적으로 *동기식*이며, 서버는 클라이언트로 성공 표시를 리턴하기 전에 트랜잭션의 WAL 레코드가 영구적 저장소에 쓰기될 때까지 기다린다. 따라서 클라이언트는 커밋 직후에 서버 충돌이 일어났더라도 커밋하려는 트랜잭션이 보존되었음을 확신할 수 있다. 그러나 짧은 트랜잭션의 경우 이러한 지연은 총 트랜잭션 시간의 주요 부분을 환경 설정한다. 비동기 커밋 모드를 선택하는 것은 WAL 레코드가 실제로 디스크에 기록되기 전에 트랜잭션이 논리적으로 완료되는 즉시 서버는 성공을 리턴한다는 것을 의미한다. 이것은 소규모 트랜잭션의 경우 처리량을 확 늘릴 수 있다.

비동기 커밋은 데이터 손실의 위험이 있다. 트랜잭션이 완료됨을 클라이언트에 알리는 리포트와 트랜잭션이 실제로 커밋된 때 사이에는 짧은 시간차가 있다(즉, 서버 충돌 시 무손실 보장). 따라서 클라이언트가 트랜잭션을 기억할 것이라는 전제 하에서 외부 액션을 취하는 경우에는 비동기 커밋을 사용해서는 안 된다. 예를 들면, 은행은 ATM의 현금 인출 트랜잭션 레코딩에 대한 비동기 커밋을 절대로 사용하지 않을 것이다. 그러나, 이벤트 로깅 같은 다수의 시나리오에서 이러한 유형을 강력하게 보장할 필요는 없다.

비동기 커밋을 사용함으로써 유발되는 위험은 데이터 손상이 아니라 데이터 손실이다. 데이터베이스가 충돌한 경우 쓰기 되었던 최신 레코드로 WAL를 리플레이함으로써 복구가 된다. 따라서 데이터베이스는 자기 모순이 없는 상태로 복원되지만 미처 디스크에 쓰기 되지 않는 트랜잭션은 해당 상태가 반영되지 못한다. 그러므로 순수 효과는 마지막 몇 개 트랜잭션의 손실이다. 트랜잭션은 커밋 명령에서 리플레이되기 때문에 불일치가 있을 수는 없다. 예를 들면, 트랜잭션 B가 이전 트랜잭션 A의 결과에 따라 변경을 하는 경우 B의 효과는 유지하면서 A의 효과는 소실되게 하는 것은 불가능하다.

사용자는 트랜잭션별로 커밋 모드를 선택할 수 있으므로 동시에 실행되는 동기 및 비동기 커밋 트랜잭션을 모두 갖는 것이 가능하다. 이것은 성능과 트랜잭션 영속성의 확실성 사이에 유연한 트레이드 오프가 가능하다. 커밋 모드는 사용자가 설정한 매개 변수 synchronous_commit로 제어되며, 환경 설정 매개 변수를 설정하는 방법대로 변경이 가능하다. 임의의 트랜잭션 하나에 대해 사용되는 모드는 트랜잭션 커밋이 시작된 경우 `synchronous_commit`의 값에 따라 달라진다.

예를 들면, **DROP TABLE**같은 특정 유틸리티 명령은 `synchronous_commit` 설정과 무관하게 강제로 동기식 커밋을 한다. 이것은 서버의 파일 시스템과 데이터베이스의 논리적 상태 간에 일관성을 유지하기 위한 것이다. **PREPARE TRANSACTION**같이 2단계 커밋을 지원하는 명령도 항상 동기식이다.

비동기 커밋 시점과 트랜잭션의 WAL 레코드 쓰기 시점 사이의 위험 시간대에 데이터베이스가 충돌할 경우 해당 트랜잭션 중에 만들어진 변경 사항은 손실될 것이다. 백그라운드 프로세스("WAL writer")는 wal_writer_delay 밀리초 단위로 쓰기 되지 않은 WAL 레코드를 디스크에 기록하기 때문에 위험 시간대의 지연 시간은 제한된다. WAL writer는 바쁜 기간 중에 전체 페이지를 한 번에 작성하도록 되어 있으므로 위험 시간대의 실제 최대 지연 시간은 `wal_writer_delay`의 세 배이다.

> **경고**
>
> 즉시 방식(immediate-mode) 셧다운은 서버 충돌과 동일하며, 따라서 미기록된 비동기 커밋의 손실이 야기된다.

비동기 커밋은 fsync = off 설정과는 다르게 동작한다. fsync는 모든 트랜잭션의 동작이 바뀌는 서버 차원(server-wide)의 설정이다. 이것은 데이터베이스의 서로 다른 부분에 쓰기를 동기화하는 Agens SQL 내의 모든 로직을 비활성화하므로 시스템 충돌(즉, Agens SQL 자체의 실패가 아니라 하드웨어 또는 운영 체제 충돌) 시 데이터베이스 상태가 제멋대로 망가지게 된다. 여러 가지 시나리오에서 비동기 커밋은 데이터 손상 위험 없이 `fsync`를 해제하여 성능을 최고로 개선해 준다.

commit_delay 역시 비동기 커밋과 매우 유사하지만 이것은 실제로 동기 커밋 방식이다(사실, commit_delay는 비동기 커밋 중에 무시된다). commit_delay는 해당 트랜잭션에 의해 실행된 쓰기가 다른 트랜잭션 커밋도 동시에 수행할 수 있도록 트랜잭션이 WAL을 디스크에 쓰기 직전에 지연을 야기한다. 이 설정으로 여러 트랜잭션에 쓰기 비용을 분할하기 위해 트랜잭션이 쓰기(flush) 하려는 그룹에 조인하는 시간을 늘릴 수 있다.

13.4. WAL 환경 설정

데이터베이스 성능에 영향을 미치는 WAL 관련 환경 설정 매개 변수가 몇 가지 있다. 이 절에서는 그것의 사용에 대해 설명한다. 서버 환경 설정 매개 변수의 설정에 대한 내용은 2장을 참조 바란다.

*Checkpoints*는 힙 및 인덱스 데이터 파일이 해당 checkpoint 전에 기록된 모든 정보로 업데이트되도록 보장하는 트랜잭션 시퀀스의 지점이다. checkpoint 시에, 모든 dirty 데이터 페이지는 디스크에 쓰기되고, 특수한 checkpoint 레코드는 로그 파일에 기록된다. (변경 레코드는 이전에 WAL 파일에 기록되었다.) 충돌 발생 시, 충돌 복구 프로시저는 REDO 명령을 시작해야 하는 로그의 지점을 판단하기 위해 최신 checkpoint 레코드를 찾아본다. 해당 지점 이전의 데이터 파일을 변경해도 디스크에는 남아 있다. 따라서, checkpoint 이후에 redo 레코드가 포함되기 이전의 로그 세그먼트는 더 이상 불필요하며, 재활용되거나 제거할 수 있다. (WAL 아카이빙이 완료되면 로그 세그먼트는 재활용 또는 제거되기 전에 아카이브되어야 한다.)

모든 dirty 데이터 페이지를 디스크에 쓰는 checkpoint 요구조건은 상당한 I/O 로드를 야기할 수 있다. 그러므로 checkpoint 시작 시 I/O가 시작되고, 다음 checkpoint가 시작되기 전에 완료되도록 checkpoint 활동이 조절된다. 이렇게 함으로써 checkpoint 시점에 성능 저하가 최소화된다.

서버의 checkpointer 프로세스는 매우 빈번하게 모든 checkpoint를 자동으로 수행한다. checkpoint는 모든 checkpoint_segments 로그 세그먼트 또는 모든 checkpoint_timeout 초마다 먼저 해당되는 것부터 시작된다. 기본 설정은 각각 3개의 세그먼트들 및 300초(5분)이다. 이전 checkpoint 이후로 기록된 WAL이 없으면 checkpoint_timeout을 초과했더라도 새 checkpoint를 건너뛴다. (WAL 아카이빙을 사용 중이고, 데이터 손실 가능성에 대한 제한을 두려고 파일 아카이빙 간격에 대한 하한을 설정하고 싶으면 checkpoint 매개 변수가 아니라 archive_timeout 매개 변수를 조절해야 한다.) SQL 명령 **CHECKPOINT**를 사용하여 checkpoint를 강제 적용하는 것도 가능하다.

checkpoint_segments 및/또는 checkpoint_timeout을 줄이면 checkpoint가 좀 더 빈번하게 발생한다. 이렇게 하면 redo에 필요한 작업이 줄어들므로 충돌 후 복구가 빨라진다. 그러나 dirty 데이터 페이지를 빈번하게 기록함으로써 늘어나는 비용 간에 균형을 맞출 필요가 있다. full_page_writes가 설정된 경우(기본값), 다른 요소를 고려해야 한다. 데이터 페이지의 일관성을 유지하려면 각 checkpoint 후 데이터 페이지의 첫 번째 수정은 결과적으로 전체 페이지 내용을 로깅하는 것으로 이어진다. 이런 경우 checkpoint 간격이 짧을수록 WAL 로그로의 출력 볼륨이 증가하여 짧은 간격으로 사용하는 목적이 부분적으로 무력화되고, 경우에 따라서는 디스크 I/O가 늘어나기도 한다.

checkpoint는, 첫째로 모든 현재의 dirty 버퍼를 기록해야 하고, 둘째로 위에서 설명한 대로 추후 WAL 트래픽이 추가 발생하기 때문에 매우 비싸다. 따라서, checkpoint가 너무 빈번하지 않도록 checkpointing 매개 변수를 최대한 크게 설정하는 것이 좋다. checkpointing 매개 변수의 간단한 정상 여부 검사로서 checkpoint_warning 매개 변수를 설정할 수 있다. checkpoint가 checkpoint_warning 초에서 설정된 것보다 간격이 짧은 경우 checkpoint_segments를 늘리라는 서버 로그 권고문이 메시지로 출력된다. 해당 메시지가 가끔씩 출현하면 경고가 발생하지 않지만 빈번하게 나타날 경우 checkpoint 제어 매개 변수를 증가해야 한다. checkpoint_segments를 충분히 크게 설정한 경우 거대(large) **COPY** 전송 같은 대량 작업으로 해당 경고가 다수 나타날 수 있다.

폭발적인 페이지 쓰기량으로 인한 I/O 시스템 폭주를 막기 위해 checkpoint 중 dirty 버퍼 쓰기는 일정 기간에 걸쳐 분산된다. 해당 기간은 checkpoint_completion_target에 의해 제어되며, 이것은 checkpoint 간격의 분획으로 지정된다. 지정된 `checkpoint_segments` WAL 세그먼트 분획이 checkpoint 시작 후에 소모되었거나, 지정된 `checkpoint_timeout` 초 분획을 경과한 경우, 둘 중에 빠른 것에 의해 checkpoint가 완료되도록 I/O 속도가 조정된다. 기본값이 0.5인 Agens SQL은 다음 checkpoint가 시작되기 전 시간의 약 절반이 지난 후 각 checkpoint를 완료하는 것으로 예상한다. 정상 실행 중에 최대 I/O 처리량에 매우 근접한 시스템에서는 I/O 로드를 checkpoint로부터 줄이고자 `checkpoint_completion_target`를 늘리려고 할 수 있다. 이것의 단점은, 복구 시에 사용할 수 있도록 WAL 세그먼트를 더 많이 확보해야 하기 때문에 연장된 checkpoint가 복구 시간에 영향을 준다는 것이다. `checkpoint_completion_target`을 1.0로 설정할 수는 있지만 checkpoint는 dirty 버퍼 쓰기 외에 다른 활동도 일부 포함하므로 그것보다는 낮게 유지하는 것이 좋다(최대 0.9). 1.0로 설정하면 checkpoint가 제시간에 완료되지 않을 가능성이 높으므로, 필요한 WAL 세그먼트 수의 예상치 못한 변동으로 성능 손실이 야기될 수 있다.

항상 하나 이상의 WAL 세그먼트 파일이 있으며, 일반적으로(2 + `checkpoint_completion_target`) * `checkpoint_segments` + 1 또는 `checkpoint_segments` + wal_keep_segments + 1개의 파일 이하이다. 각 세그먼트 파일은 일반적으로 16 MB이다(이 크기는 서버 빌드 시에 변경 가능). 이것을 사용하여 WAL에 필요한 공간을 측정할 수 있다. 보통은 이전 로그 세그먼트 파일이 더 이상 필요 없을 때 재활용된다(즉, 번호 순서에 따라 추후 세그먼트가 될 이름으로 변경). 로그 출력 속도가 단기간 최고치에 도달해, 3 * `checkpoint_segments` + 1 개의 세그먼트 파일보다 많아질 경우 시스템이 이 제한으로 내려올 때까지는 불필요한 세그먼트 파일이 재활용되지 않고 삭제된다.

아카이브 복구 또는 스탠바이 모드에서 서버는 주기적으로 *restartpoints*,를 수행하는데, 이것은 정상 실행된 checkpoints와 유사하다. 서버는 모든 상태를 디스크에 강제로 기록하고, `pg_control` 파일을 업데이트하여 이미 처리된 WAL 데이터를 다시 스캔할 필요가 없음을 표시하여 `pg_xlog` 디렉토리에 있는 예전 로그 세그먼트 파일을 재활용할 수 있게 한다. restartpoints는 checkpoint 레코드에서만 수행될 수 있으므로 restartpoints는 마스터에서의 수행 빈도가 checkpoints보다 적다. 마지막 restartpoint 이후에 최소한 `checkpoint_timeout` 초를 경과한 경우 checkpoint 레코드에 도달하면 restartpoint가 트리거된다. 스탠바이 모드에서, 최소한 `checkpoint_segments` 로그 세그먼트가 마지막 restartpoint 이후에 리플레이된 경우에도 restartpoint가 트리거된다.

일반적으로 사용되는 내부 WAL 함수는 `XLogInsert` 및 `XLogFlush`의 두 가지가 있다. `XLogInsert`는 공유 메모리에서 새 레코드를 WAL 버퍼에 배치할 때 사용된다. 새 레코드를 위한 공간이 없는 경우, `XLogInsert`는 몇 개의 채워진 WAL 버퍼를 기록해야 한다(커널 캐시로 이동). 영향을 받는 데이터 페이지에 배타적 잠금이 걸려 있어서 명령이 가능한 빨라야 하는 경우, `XLogInsert`가 모든 데이터베이스 저수준 변경(예를 들면, 행 삽입)에 사용되므로 이는 바람직하지 않다. 더 안 좋은 것은, WAL 버퍼 쓰기 작업 때문에 새로운 로그 세그먼트가 생성되어 시간이 더 늘어날 수도 있다는 것이다. 일반적으로, WAL 버퍼는 `XLogFlush` 요청에 의해 쓰기 되어야 하지만, 대부분의 경우 트랜잭션 레코드가 영구적인 저장소에 기록되도록 트랜잭션 커밋 시에 발생한다. 로그 출력이 많은 시스템에서 `XLogFlush` 요청은 `XLogInsert`가 쓰기를 금지할 만큼 빈번하지 않다. 해당 시스템에서는 wal_buffers 매개 변수를 변경하여 WAL 버퍼 수를 늘려야 한다. full_page_writes가 설정된 경우 및 시스템이 매우 바쁜 경우, `wal_buffers`를 큰 값으로 설정하면 각 checkpoint 바로 다음 기간 중에 순조로운 반응 시간을 유도할 수 있다.

commit_delay 매개 변수는 그룹 커밋 리더 프로세스가 XLogFlush`XLogFlush` 내에서 잠금을 획득한 후에 슬립하는 마이크로초 시간을 정의하며, 그룹 커밋 팔로워는 리더 뒤에서 대기한다. 이러한 지연은 다른 서버 프로세스가 자신의 커밋 레코드를 WAL 버퍼에 추가하는 것을 허용하므로 이들 모두는 리더의 최종 동기화 명령에 의해 쓰기 된다. fsync가 활성화되지 않으면 슬립이 발생하지 않으며, commit_siblings보다 적을 경우 다른 세션이 현재 활성 트랜잭션이 된다. 이렇게 하면 다른 세션이 곧 커밋하지 않을 경우에 슬립을 방지할 수 있다. 일부 플랫

폼에서 슬립 요청 시간은 10밀리초이므로 1에서 10000마이크로초 사이의 숫자에서 0이 아닌 값으로 commit_delay를 설정하면 동일한 효과를 갖는다. 일부 플랫폼에서 슬립 명령은 매개변수에 의해 요청된 것보다 약간 길 수 있다.

commit_delay의 목적은 각 쓰기 명령의 비용이 동시 커밋된 트랜잭션 간에 분할되도록 하는 것이므로(트랜잭션 대기 시간 비용) 설정 전에 비용을 적절하게 선택할 수 있도록 정량화할 필요가 있다. 비용이 클수록 트랜잭션 처리량을 증가시키는 데 commit_delay의 효율이 어느 정도까지 커진다. pg_test_fsync 프로그램을 사용하면 단일 WAL 쓰기 명령을 수행할 때의 평균 시간을 마이크로초 단위로 측정할 수 있다. 단일 8kB 쓰기 명령 후에 쓰기에 소요되는 것으로 프로그램이 리포트한 평균 시간의 절반 값은 commit_delay에 가장 효율적인 설정이므로 이 값은 특정 작업 부하를 최적화할 때 사용되는 시작점으로 권장된다. commit_delay 튜닝은 WAL 로그가 고비용의 대기 회전 디스크(high-latency rotating disk)에 저장된 경우에 특히 유용하며, solid-state drive 또는 배터리 백업 쓰기 캐시가 있는 RAID 배열 같이 동기화 시간이 매우 빠른 저장 매체에서도 장점을 발휘한다. 단, 이것은 대표적인 작업 부하에 대해 테스트해야 한다. 그런 경우 commit_siblings에 더 큰 값을 설정해야 하는데, commit_siblings 값을 작게 하면 대기 시간이 긴 매체에서 종종 유용하다. commit_delay 설정이 너무 크면 총 트랜잭션 처리에 걸리는 시간만큼 트랜잭션 대기 시간이 늘어날 수 있다.

commit_delay가 0으로 설정된 경우(기본값), 발생하는 그룹 커밋 형태로 여전히 가능하지만 각 그룹은 이전 쓰기 명령(있을 경우)이 발생한 시간에 커밋 레코드를 기록해야 하는 지점에 도달하는 세션만으로 환경 설정된다. 높은 클라이언트 카운트에서 "통로 효과(gangway effect)"가 발생하는 추세이면 commit_delay가 0일 때 그룹 커밋의 효과는 상당히 크며, 따라서 commit_delay의 명시적 설정은 무용지물이 될 가능성이 높다. commit_delay 설정은, (1) 일부 동시 커밋 트랜잭션이 있는 경우 및 (2) 처리량이 커밋 속도에 의해 일정 수준으로 제한되는 경우에만 도움이 된다. 그러나 높은 회전 대기 시간을 사용하는 경우 두 개의 클라이언트만큼의 트랜잭션 처리량 증가 시 이 설정이 효율적일 수 있다(즉, 형제 트랜잭션이 1개 있는 단일 커밋 클라이언트).

wal_sync_method 매개 변수는 Agens SQL이 디스크로의 WAL 업데이트를 커널에 요청하는 빈도를 결정한다. 다른 옵션은 그렇지 않지만, 디스크 캐시에 강제로 쓰기할 수 있는 fsync_writethrough를 제외하고는 안정성 측면에서 모든 옵션은 동일해야 한다. 그러나 어떤 것이 가장 빠른지는 플랫폼에 따라 다르다. pg_test_fsync 프로그램을 사용하면 서로 다른 옵션의 속도를 테스트해 볼 수 있다. fsync가 해제된 경우에는 이 매개 변수가 무효화된다.

wal_debug 환경 설정 매개 변수(Agens SQL이 지원을 사용하여 컴파일된 경우)를 활성화하면, 결과적으로 각 XLogInsert 및 XLogFlush WAL 호출이 서버 로그에 로깅된다. 이 옵션은 나중에 좀 더 일반적인 메커니즘으로 교체될 수 있다.

13.5. WAL 인터널

WAL은 자동으로 활성화된다. WAL 로그에 대한 디스크 공간 요구사항을 맞춰야 할 때 및 필수 튜닝이 완료되었을 때 외에는 관리자의 개입이 불필요하다(13.4절 참조).

WAL 로그는 각각의 크기가 보통 16 MB인 세그먼트 파일 집합으로, 데이터 디렉토리 하위인 pg_xlog 디렉토리에 저장된다(서버 빌드 시 --with-wal-segsize 환경 설정 옵션을 변경하면 크기를 바꿀 수 있다). 각 세그먼트는 일반적으로 각각 8 kB의 페이지로 분할된다(이 크기는 --with-wal-blocksize 환경 설정 옵션을 통해 변경할 수 있다). 로그 레코드 헤더는 access/xlog.h에 설명되어 있다. 레코드 내용은 로깅된 이벤트 유형에 따라 다르다. 세그먼트 파일 이름은 000000010000000000000000에서 시작되며 숫자가 계속 증가한다. 숫자는 랩(wrap)되지 않지만 가용 숫자 재고가 소진되려면 아주 오래 걸린다.

로그가 메인 데이터베이스 파일이 아닌 다른 디스크에 위치한 경우에 이것은 장점이 된다. 서버가 종료된 상태에서 pg_xlog 디렉토리를 다른 위치로 옮기고, 메인 데이터 디렉토리의 원래 위치로부터 새 위치로 심볼릭 링크를 생성하면 된다.

WAL의 목적은 데이터베이스 레코드가 변경되기 전에 로그가 기록되도록 하는 것이지만, 사실은 데이터를 캐싱만하고 디스크에 미처 저장하지 못한 경우 디스크 드라이브가 커널에 쓰기를 성공했단 내용을 잘못 리포트해 엉망이 될 수 있다. 이러한 상황에서 정전이 발생하면 복구 불가능한 데이터 손상이 발생하게 된다. 관리자는 Agens SQL의 WAL 로그 파일이 저장되는 디스크가 이런 잘못된 리포트를 하지 않도록 해야 한다. (13.1절 참조.)

checkpoint가 설정되고 로그가 쓰기된 후 checkpoint의 위치는 `pg_control` 파일에 저장된다. 그러므로, 복구 시작 시 서버는 먼저 `pg_control`을 읽은 다음, checkpoint 레코드를 읽는다. 그러고 나서 chekcpoint 레코드에 표시된 로그 위치에서 순방향으로 스캔하여 REDO 명령을 수행한다. 데이터 페이지의 전체 내용은 checkpoint 이후의 첫 번째 페이지 수정에 대한 로그에 저장되므로 (full_page_writes는 비활성화되었다고 가정) checkpoint 이후의 모든 페이지 변경은 일관된 상태로 복원된다.

`pg_control`가 손상된 경우를 처리하려면 최신 checkpoint를 찾기 위해 역순(가장 최신- 가장 과거 순)으로 기존 로그 세그먼트를 스캐닝 해봐야 한다. 이는 아직 구현되지 않았다. `pg_control`은 충분히 작아서(디스크 페이지 1개 미만) partial-write 문제에 해당되지 않으며, 현재까지, `pg_control` 자체가 읽지 못하는 것만으로 데이터베이스 오류가 난 적은 없었다. 따라서, 이론적으론 고쳐야 할 부분이지만 `pg_control`이 실제로 문제가 되는 것은 아니다.