psql 사용 팁

pgday.seoul 2023, 김상기

시작하기 앞서

- 인터페이스 - 사람과 데이터베이스 인스턴스를 이어주는 그 무엇, '얼굴 사이' :)
- cli - 명령행 인터페이스 - `sh`, `cmd.exe` ……
- gui - 그래픽 사용자 인터페이스 - Macintosh, X Window, Microsoft Windows,
- tui - 텍스트 기반 사용자 인터페이스 - mdir, `top`, `nano`

- 왜 아직도 저 구닥다리 같은 cli는 쓰일까?

첫번째

`psql --help`

붙임표 두개

- PostgreSQL 공식 배포판에 포함되는 모든 명령어의 도움말은 `--help`
- psql에서 자주 사용 되는 옵션들
  - 인스턴스 접속 옵션: -h (host), -p (port), -d (database), -U (user, -u 아님)
  - 쿼리 실행 옵션: -c (sql string), -f (sql file)
  - 출력 양식 바꾸는 옵션: -A (align), --csv, -t (tuples-only), -x
  - 디버그, 조사 옵션: -e (echo), -E (echo hidden), -s (step by step)

두번째

OS 환경 변수

- `psql` 실행에 앞서 psql 내부에서 사용하는 OS 환경 변수들을 미리 지정해 놓으면 psql 실행 방법이 편해진다.
- `PGDATABASE`, `PGHOST`, `PGPORT`, `PGUSER`, `PSQL_EDITOR`, `EDITOR`, `VISUAL`, `PSQL_EDITOR_LINENUMBER_ARG`, `PSQL_PAGER`, `PAGER`, `PSQL_WATCH_PAGER`, `PSQLRC`, `PSQL_HISTORY`, 그외 (16버전 기준)
- 여러 인스턴스를 함께 관리해야 하는 이들을 위한 팁
  - 각 인스턴스에 맞는 환경 변수들을 각 파일로 저장하고, 필요에 따라 `psql`을 실행하기전 그 환경 변수 설정 파일을 반영한다.
    `. 16.env` 또는 `source salesdb.env` 또는 `db-1.bat`
- 쉘 스크립트 안에서 `psql` 사용을 편하게 할 때도 유용하다.

세 번 째

파이프

- `pg_dump -h source -t table1 | psql -h target`
- ```
  psql -h source -c 'copy table1 to stdout' \
    | psql -h target -c 'copy table1 from stdin'
  ```
- 이 파이프는 `psql` 안에서도 쓸 수 있다.
  - `\g |` 쉘명령
  - `\o |` 쉘명령
- 이 밖에도 쓰임새가 많다. 여기서 이루 다 말할 수 없을 만큼, 그지없이 많다.

(그래서 몇가지 활용이 있는데?)
  ```

네 번 째

\g*

- psql 프롬프트 상태에서 **psql** 내장 명령어는 \ 문자로 시작한다.
- 관련 도움말은 **\?**
- 입력한 SQL 구문을 접속한 인스턴스로 보내서 그 응답을 받는 명령이 **\g**
  - 정확하게는 **psql** 쿼리 버퍼에 있는 내용을 (인스턴스로) '가라 go' 한다.
  - 이 버퍼가 비어있다면, (**\r** 명령으로 비움) 마지막 실행했던 쿼리를 다시 보낸다.
  - **psql** 내장 명령 가운데, 유일하게, \ 문자로 시작하지 않는 **;** 명령과 비교해서 그 차이점을 알아 두면 여러모로 편하다.
- **\gset**: 쿼리 결과가 단일건이면 칼럼별로 그 결과값을 내장 변수로 지정
  - **psql** 스크립트를 만들 때 쓰임새가 많다.
- **\gexec**: 쿼리 결과를 다시 쿼리, 사용에 각별히 주의를 기울여야 한다.

다섯번째

\d*

- 제일 많이 사용하는 각종 정보를 보는 명령어
- S 문자를 추가하면 보이지 않았던 시스템 카탈로그 정보도
- + 문자를 추가하면 해당 대상 관련 좀 더 자세한 정보를
- 대부분 명령에는 정규식 문자열을 인자로 쓸 수 있고, 이때는 해당 패턴만 출력
  - 예: `\dt+ *.*20*_bk` (모든 스키마 안에, 이름이 *20*_bk 인 테이블들만 보자, 더불어 그 크기도)
- DBA가 기억해두면 여러모로 편한 것들
  - **`\dconfig`**, **`\dP`**

여섯번째

\set, \gset, \echo, \prompt, \if

- 대화식 쿼리 질의 도구를 넘어 프로그래밍의 차원으로
- 순환구문을 아직 지원하지 않는 것이 아쉬움
- **\if \elif \else \endif** 는 구문 블럭이 완성되어야 하고, \if \elif 뒤에는 불리언형 변수만 올 수 있음

```
select :SERVER_VERSION_NUM > 110000 as is_gt11 \gset
\if :is_gt11
\echo
\else
\echo 너무 오래된 인스턴스를 사용하고 있네요.
\end if
```

일곱번째

\pset

- psql 출력의 모든 것
- \pset pager
- \pset numericlocale
- \pset expanded auto
- \pset linestyle unicode
- \pset format csv
- psql -c 옵션 인자로 먼저 필요에 따라 선언하고, 마지막 -c 옵션에 원하는 쿼리를 지정하는 형태로 사용해서 출력 또는 파이프로 연결할 때 편하다.
  - psql -Xt --csv -P pager -c "....." == psql -X -c '\pset tuple_only' -c '\pset pager off' -c '\pset format csv' -c "....."

여덟 번째

\timing 과 \watch

- **\timing** 명령으로 보이는 시간은 psql 처리시간(클라이언트 작업시간)까지 포함하는 시간이다. 서버 시간만 확인 하려면, `explain analyze` SQL 명령으로
- \watch 인자로 쓴 반복 주기값은 소수점도 사용할 수 있다. `\watch 0.01`
  - 반복 작업을 하되, 중간에 잠깐 쉬는 시간을 가져야 하는 작업에서 유용
  - 16 버전에서는 반복 회수도 지정할 수 있다. `\watch 0.01 c=100`
- \watch의 출력 결과는 다음 장표에서 소개할 \pset 설정에 의존적이다.
- \watch 명령은 아쉽게도 명령행에서 사용할 수 없다.

아홉번째

~/.psqlrc 와 -X

- ~/.bash_profile 과 같은 것 - 잘 설정하면 많이 편하다.
- PSQLRC OS 환경 변수로 사용자 정의 파일이름을 지정할 수 있어 다양한 인스턴스 환경에 맞게 사용할 수 있다.
- 쉘 스크립트에서 아무 생각 없이 psql을 사용하면 기본적으로 이 ~/.psqlrc 파일이 호출된다. 원인 파악이 힘들었던 예:

> ~/.psqlrc 파일에 \set AUTOCOMMIT off 가 있고,
> 쉘 스크립트는 for i in $(seq 100) ; do psql -c "insert into t values ($i)"; done
> 스크립트를 아무리 돌려도 t 테이블에는 늘 자료가 없다.

- 윗 문제를 피하기 위해 쉘 스크립트 안에서는 psql 을 호출할 때는 **-X** 쓰는 것이 좋겠다.

열개 채우기 위한 마지막

pspg

- psql 페이지 단위 보기에서 사용할 수 있는 끝판왕
- 전형적인 텍스트 기반 사용자 인터페이스(tui)다.
- 사용법은

  ```
  export PSQL_PAGER=pspg
  ```
- pspg도 사용자 정의 환경 설정을 반영하기 위한 ~/.pspgconf 파일을 사용한다.

  자동생성은 pspg 화면 안에서 Options -> Save Setup
- pspg도 한글화 되었으면!

참고 자료

- psql 설명서: https://postgresql.org/docs/current/app-psql.html
- pspg: https://github.com/okbob/pspg