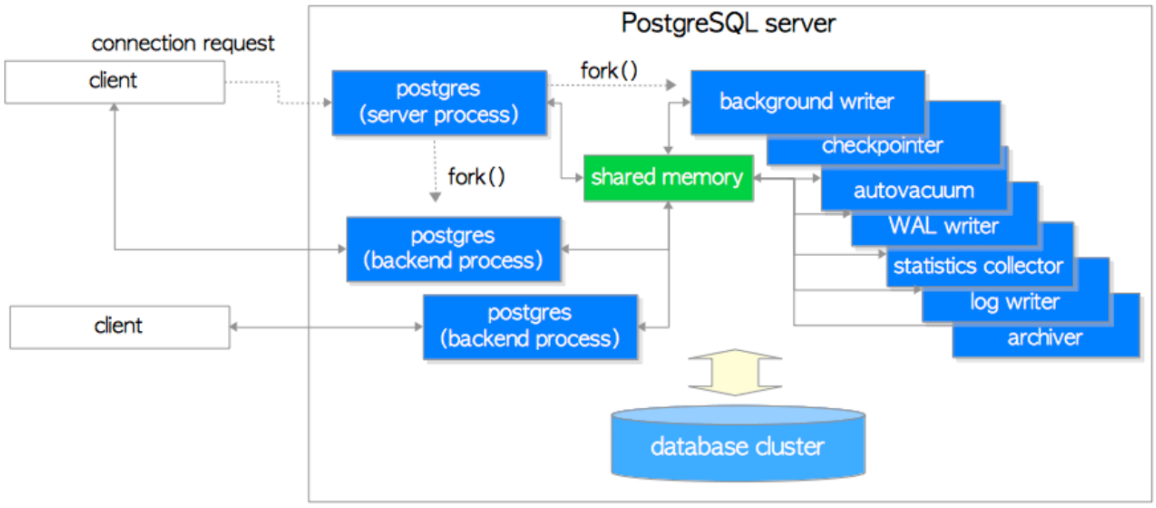
PostgreSQL

1. PostgreSQL 이란?

> 오픈 소스 객체-관계형 데이터베이스 시스템(ORDBMS)으로, Enterprise급 DBMS의 기능과 차세대 DBMS에서나 볼 수 있을 법한 기능들을 제공
>
> 약 20여년의 오랜 역사를 갖는 PostgreSQL은 다른 관계형 데이터베이스 시스템과 달리 연산자, 복합 자료형, 집계 함수, 자료형 변환자, 확장 기능 등 다양한 데이터베이스 객체를 사용자가 임의로 만들 수 있는 기능을 제공함으로써 마치 새로운 하나의 프로그래밍 언어처럼 무한한 기능을 손쉽게 구현

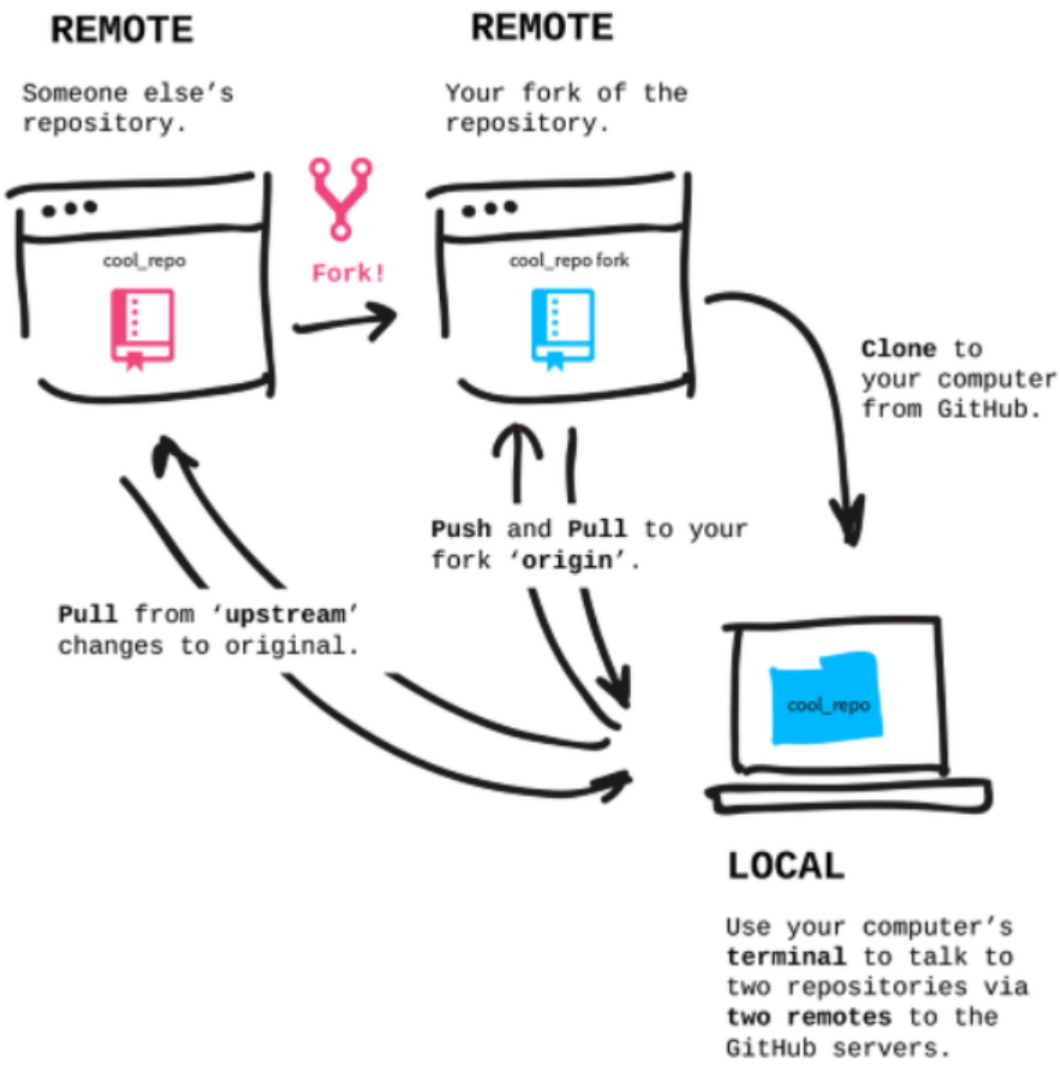
2. PostgreSQL의 구조

- PostgreSQL은 **클라이언트/서버 모델**을 사용
- 서버는 데이터베이스 파일들을 관리하며, 클라이언트 애플리케이션으로부터 들어오는 연결을 수용하고, 클라이언트를 대신하여 데이터베이스 액션을 수행
- 서버는 다중 클라이언트 연결을 처리할 수 있는데, 서버는 클라이언트의 연결 요청이 오면 각 커넥션에 대해 새로운 프로세스를 fork
- 그리고 클라이언트는 기존 서버와의 간섭 없이 새로 생성된 서버 프로세스와 통신하게 됨



※ fork란?

> 외부 원격 저장소에 업로드 되어있는 오픈소스를 내가 수정하고 싶을때 외부 원격저장소(Repository) 자체에 push할수 없으므로 외부 원격저장소의 히스토리를 나의 원격저장소로 복사해 오는것을 의미



3. PostgreSQL 기능

> 관계형 DBMS의 기본적인 기능인 트랜잭션과 ACID(Atomicity, Consistency, Isolation, Durability)를 지원

- ANSI:2008 구격을 상당 부분 만족시키고 있으며, 전부 지원하는 것을 목표로 계속 기능을 추가
    - ANSI(미국 국가표준학회) : 표준들을 계발하지 않지만, 학회는 표준들을 개발하는 조직들의 절차를 공인함으로써 표준들의 개발과 사용을 감독

제한

항목	제한 사항
최대 DB 크기(Database Size)	무제한
최대 테이블 크기(Table Size)	32TB
최대 레코드 크기(Row Size)	1.6TB
최대 컬럼 크기(Field Size)	1 GB
테이블당 최대 레코드 개수(Rows per Table)	무제한
테이블당 최대 컬럼 개수(Columns per Table)	250~1600개
테이블당 최대 인덱스 개수(Indexes per Table)	무제한

제공하는 기능

- Nested transactions (savepoints)
- Point in time recovery
- Online/hot backups, Parallel restore
- Rules system (query rewrite system)
- B-tree, R-tree, hash, GiST method indexes
- Multi-Version Concurrency Control (MVCC)
- Tablespaces
- Procedural Language
- Information Schema
- I18N, L10N
- Database & Column level collation
- Array, XML, UUID type
- Auto-increment (sequences),
- Asynchronous replication
- LIMIT/OFFSET
- Full text search
- SSL, IPv6
- Key/Value storage
- Table inheritance

PostgreSQL의 특징

- Portable

  > ANSI C로 개발되었으며, 지원하는 플랫폼의 종류로는 Windows, Linux, MAC OS/X, Unix 등 다양한 플랫폼을 지원

- Reliable

  > 트랜잭션 속성은 ACID에 대한 구현 및 MVCC
  >
  > 로우 레벨 라킹 등이 구현

- Scalable

> PostgreSQL의 멀티 버전에 대해 사용이 가능
> 대용량 데이터 처리를 위한 Table Partitioning과 Tables Space 기능 구현 가능

- Recovery & Availability

> Streaming Replication을 기본으로, 동기식/비동기식 Hot Standby 서버를 구축 가능
> WAL Log 아카이빙 및 Hot Back up 을 통해 Point in time recovery 가능

- Advanced

> pg_upgrade를 통해 업그레이드 가능
> 웹 또는 C/S 기반의 GUI 관리 도구를 제공하여 모니터링 및 관리는 물론 튜닝까지 가능
> 사용자 정의 Procedural로 Perl, Java, Php 등의 스크립트 언어 지원이 가능

Vacuum

Vacuum은 PostgreSQL에만 존재하는 고유 명령어로, 오래된 영역을 재사용 하거나 정리해주는 명령어이다. PostgreSQL에서는 MVCC(Multi-Version Concurrency Control, 다중 버전 동시성 제어) 기법을 활용하기 때문에 특정 Row를 추가 또는 업데이트 할 경우, 디스크 상의 해당 Row를 물리적으로 업데이트하여 사용하지 않고, 새로운 영역을 할당해서 사용한다. 예를 들어 전체 테이블을 Update하는 경우에는 자료의 수만큼 자료 공간이 늘어나게 된다. 그러므로 Update, Delete, Insert가 자주 일어나는 Database의 경우는 물리적인 저장 공간이 삭제되지 않고 남아있게 되므로, vacuum을 주기적으로 해주는 것이 좋다. Vacuum을 사용하면 어느 곳에서도 참조되지 않고, 안전하게 재사용할 수 있는 행을 찾아 FSM(Free Space Map)이라는 메모리 공간에 그 위치와 크기를 기록한다. 그리고 Insert 및 Update 등 새로운 행을 추가하는 경우, FSM에서 새로운 데이터를 저장할 수 있는 적당한 크기의 행을 찾아 사용한다.

✔MVCC

> 동시접근을 허용하는 데이터베이스에서 동시성을 제어하기 위해 사용하는 방법. 즉, MVCC 모델에서 데이터에 접근하는 사용자는 접근한 시점에서의 데이터베이스의 Snapshot을 읽는데, 이 snapshot 데이터에 대한 변경이 완료될 때(트랜잭션이 commit 될 때)까지 만들어진 변경사항은 다른 데이터베이스 사용자가 볼 수 없다. 이러한 개념에 의해 사용자가 데이터를 업데이트하면 이전의 데이터를 덮어 씌우는게 아니라 새로운 버전의 데이터를 생성한다. 대신 이전 버전의 데이터와 비교해서 변경된 내용을 기록한다. 이렇게해서 하나의 데이터에 대해 여러 버전의 데이터가 존재하게 되고, 사용자는 마지막 버전의 데이터를 읽게 된다.

기존 테이블	
no	name
1	홍길동
2	영이
3	철이
4	김태균

변경 테이블	
no	name
1	홍길동
2	영이
3	철이
4	김태균
2	영이/수정

2번째 행의 **"영이"**를 **"영이/수정"**으로 변경하면 기존 행은 유지되고 새로운 행이 추가됩니다.

4전째 행의 **"김태균"**을 삭제했지만 디스크에서는 지워지지 않고 유지됩니다.

변경 테이블	
no	name
1	홍길동
2	영이
3	철이
6	길동/추가
2	영이/수정

기존 테이블	
no	name
1	홍길동
2	영이
3	철이
4	김태균
2	영이/수정

새로운 데이터를 추가하면 **FSM**을 확인 후 빈 곳에 데이터를 추가합니다.

기존 테이블	
no	name
1	홍길동
2	영이
3	철이
6	길동/추가
2	영이/수정

변경 테이블	
no	name
1	홍길동
3	철이
6	길동/추가
2	영이/수정

VACUUM를 실행하면 불필요한 정보가 **삭제**됩니다.

Vacuum Command

vacuumdb를 활용하여 주기적으로 정리할 수 있는데, 관련 옵션들은 아래와 같다. full 옵션 없이 vacuumdb를 실행할 경우는 단순히 사용가능한 공간만을 반환한다. 하지만 full옵션을 추가하는 경우에는 빈 영역에 tuple을 옮기는 등 디스크 최적화 작업을 하게 된다. 디스크 최적화를 위해 table에는 LOCK이 걸리게 되고, 시간이 오래 걸리게 되므로 사용 시 주의해야 한다.

```
사용법:
 ``vacuumdb [옵션]... [DB이름]

 -- DB 전체 풀 실행
vacuum full analyze; // 데이터베이스가 대청소 후 잠금처리 되므로 운영중인 db에서는 사용 x

-- DB 전체 간단하게 실행
vacuum verbose analyze;

-- 해당 테이블만 간단하게 실행
vacuum analyse [테이블 명];

-- 특정 테이블만 풀 실행
vacuum full [테이블명];
```

```
옵션들:
  -a, --all                        모든 데이터베이스 청소
  -d, --dbname=DBNAME              DBNAME 데이터베이스 청소
  -e, --echo                       서버로 보내는 명령들을 보여줌
  -f, --full                       대청소 // 데이터베이스가 대청소 후 잠금처리 되므로
운영중인 db에서는 사용 x
  -F, --freeze                     행 트랜잭션 정보 동결
  -q, --quiet                      어떠한 메시지도 보여주지 않음
  -t, --table='TABLE[(COLUMNS)]'   지정한 특정 테이블만 청소
  -v, --verbose                    작업내역의 자세한 출력
  -V, --version                    output version information, then exit
  -z, --analyze                    update optimizer statistics
  -Z, --analyze-only               only update optimizer statistics
  -?, --help                       show this help, then exit
```

```
연결 옵션들:
  -h, --host=HOSTNAME              데이터베이스 서버 호스트 또는 소켓 디렉터리
  -p, --port=PORT                  데이터베이스 서버 포트
  -U, --username=USERNAME          접속할 사용자이름
  -w, --no-password                암호 프롬프트 표시 안 함
  -W, --password                   암호 프롬프트 표시함
  --maintenance-db=DBNAME          alternate maintenance database
```

베큠이 청소하는 기준

사람마다 청소하는 방범이 있듯 PostgreSQL도 청소 기준을 가지고 있습니다.
그 기준을 FSM(Free Space Map)이라고하는데, 더 이상 필요하지 않는 행의 정보를 보유하고 있습니다.
이 정보는 실제로 사용되지는 않지만 용량을 차지하고 있고, 새로운 행이 삽입될때 DBMS는 FSM의 여유 공간을 확인하여 해당 행을 사용하게 됩니다.
그리고 FSM 공간은 용량이 제한되어 있기때문에 주기적으로 청소하는게 좋습니다.

autovacuum 활용

PostgreSQL 서버 실행 시에 참고하는 `postgresql.conf` 파일 안의 AUTOVACUUM PARAMETERS를 지정하여 활성화할 수 있다. 9.0 이상의 버전에서 부터는 해당 파라미터들이 주석처리(#)되어 있어도, default로 실행이 되게 되어 있다.

PostgreSQL 실습

외부접속 설정

http://www.gurubee.net/lecture/2917

PSQL

1. \db: 테이블스페이스(tablespace)를 보여준다.

※테이블스페이스란?

- Mysql 과 MariaDB에서는 없는 개념이며, Oracle과 PostgreSQL에서만 존재하는 개념.

- 일반적으로 데이터베이스는 지정된 장소에만 저장할 수 있으나, 테이블스페이스를 이용하면 아무 경로로 지정하여 데이터베이스를 저장할 수 있다. 따라서, DB를 확장하거나 HDD에서 쓰이든 DB를 SSD로 옮기게 함으로써 성능 향상을 꾀할 수 있다.

데이터베이스를 만들 때, 테이블스페이스를 지정하지 않으면 기본 설정된 곳에 데이터베이스가 만들어진다.

tablespace의 생성 예: CREATE TABLESPACE gibeom LOCATION '/dev/ssd/db2/data';

-> /dev/ssd/db2/data 경로에 gibeom 이라는 테이블 스페이스가 만들어졌다.

https://www.postgresql.org/docs/10/static/sql-createtablespace.html

---

2. \l: 모든 데이터베이스를 출력

---

3. \c 데이터베이스이름: 해당 데이터베이스로 이동

---

4. \dt: 선택된 데이터베이스의 테이블 목록 출력

---

5. \dn: 해당 유저(Role)의 스키마 목록을 출력

※ Postgresql에서 데이터베이스와 스키마의 차이?

- 간단하게, 스키마는 실질적인 데이터베이스를 프로젝트 사용자(Role)가 관리하기 쉽도록 지정한 논리적 구조로 유연성이 있다.(같은 테이블을 다른 스키마 내에서 사용할 수 있음)

더 쉽게 말하자면, 데이터베이스를 관리해주는 일부분이다. 참고로 Mysql에선 데이터베이스와 스키마는 차이가 없다.

https://code.i-harness.com/ko/q/b147e5

---

6. \d: 테이블이름: 해당 테이블의 구조를 출력

---

7. \du: 유저(Role)들의 목록을 출력

---

8. \q : 나가기

쿼리문

1. 데이터베이스 만들기

TableSpace가 없을 때

> **CREATE DATABASE 데이터베이스이름;**

TableSpace가 있을 때

> **CREATE DATABASE 데이터베이스이름 TABLESPACE 테이블스페이스이름;**

---

2. 데이터베이스 지우기

> **DROP DATABASE 데이터베이스이름;**

---

3. 스키마 만들기

> **CREATE SCHEMA 스키마이름;**

---

4. 스키마 지우기

스키마 지우기

> **DROP SCHEMA 스키마이름;**

스키마 및 해당 스키마와 관련된 모든 테이블들까지 지우기(주의!!)

> **DROP SCHEMA 스키마이름 CASCADE;**

---

5. 특정 데이터베이스 접속

'\connect 데이터베이스이름' 입력

5-1. 특정 스키마에 만들지 않고 데이터베이스에 바로 만들 때(public 스키마에 저장 됨)

> **CRAETE TABLE 테이블이름(컬럼1 데이터타입1 PRIMARY KEY, 컬럼2 데이터타입2, ... 컬럼n 데이터타입n);**

예) 아이디, 비밀번호, 이름을 정보로 회원가입한 유저들의 테이블을 만든다고 가정한 경우.

CREATE TABLE REG_USERS (

U_NO serial PRIMARY KEY NOT NULL,

U_ID VARCHAR(25) NOT NULL,

U_PASSWORD text NOT NULL,

U_NAME VARCHAR(20) NOT NULL);

※주의: serial은 Mysql에서 쓰인 auto increment와 같다.

※PRIMARY KEY는 앞에 넣든 뒤에 넣든 상관없다.

5-2. 특정 스키마에 테이블을 만들었을 때

> **CREATE TABLE 스키마이름.테이블이름 (컬럼1 데이터타입1 .... 컬럼n 데이터타입n);**

---

6. 테이블 지우기

> **DROP TABLE 테이블이름;**

---

7. INSERT 쿼리문 - 테이블에 데이터를 삽입할 때

모든 컬럼에 빠짐없이 INSERT 할 때

> **INSERT INTO 테이블이름 VALUES (값1, 값2, 값3 ... 값n);**

특정 컬럼에 INSERT 할 때,

> **INSERT INTO 테이블이름 (컬럼1, 컬럼3, 컬럼5 ... 컬럼2n-1) VALUES (값1, 값3, 값5 ... 값2n-1);**

예) 아이디, 비밀번호, 이름을 테이블에 입력하는 경우.

INSERT INTO REG_USERS (U_ID, U_PASSWORD, U_NAME) VALUES ('아이디', '비밀번호', '이름');

---

8. SELECT 쿼리문 - 테이블에 있는 데이터를 가져올 때

8-1. 조건(WHERE)이 없을 때

특정 컬럼만 가져올 때

> **SELECT 컬럼1, 컬럼2, ... 컬럼n FROM 테이블이름;**

모든 컬럼을 가져올 때

> **SELECT * FROM 테이블이름;**

8-2. 조건(WHERE)이 있을 때

> **SELECT 컬럼1, 컬럼2, ... 컬럼 n FROM 테이블이름 WHERE = 조건;**

예) 아이디가 roqkfwk인 유저의 이름을 출력하라

-> SELECT U_NAME FROM REG_USER WHERE U_ID = 'roqkfwk';

---

9. UPDATE 쿼리문 - 테이블에 있는 데이터를 수정할 때

> **UPDATE 테이블이름 SET 컬럼1 = 값1, 컬럼2 = 값2, ... 컬럼n=값n WHERE 조건;**

예) 아이디가 roqkfwk인 유저의 이름을 홍길동으로 바꾸어라

-> UPDATE REG_USER SET U_NAME = '홍길동' WHERE U_ID = 'roqkfwk';

---

10. DELETE 쿼리문 - 테이블에 있는 데이터를 삭제할 때

해당 테이블의 모든 행을 삭제하고 싶을 때(테이블을 빈값으로 만들고 싶을 때)

> **DELETE FROM 테이블이름**

조건에 해당하는 행만 삭제할 때

> **DELETE FROM 테이블이름 WHERE = 조건**

예) 아이디가 roqkfwkdls 유저의 행을 지워라.

-> DELETE FROM REG_USER WHERE = 'roqkfwk';

MySQL vs PostgreSQL

MySQL의 database == PostgreSQL의 schema

```
데이터베이스 구조
```

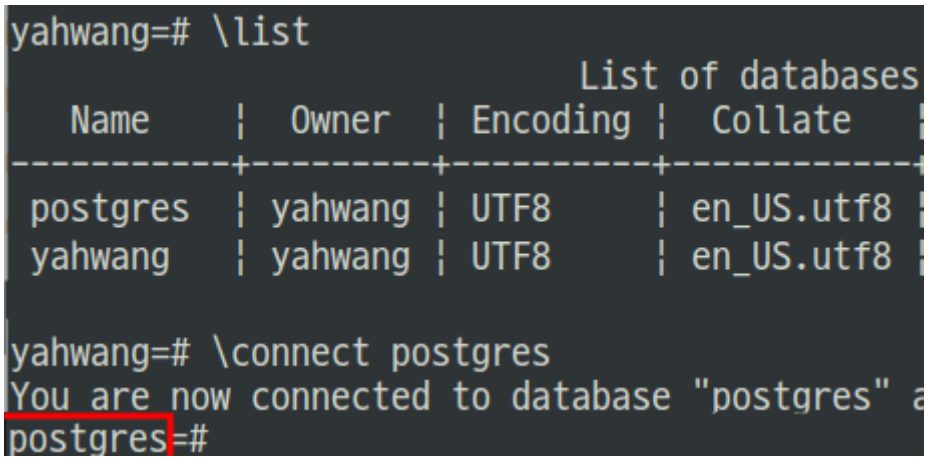
MySQL : **database** -> table

PostgreSQL : database -> **schema** -> table

테이블의 집합이라는 의미로 MySQL에서는 database, PostgreSQL에서는 schema가 사용된다.

`PostgreSQL의 database`

PostgreSQL은 하나의 데이터베이스를 기준으로 접속한다. 접속한 데이터베이스명을 표시한다.



`PostgreSQL의 schema`

PostgreSQL에서 schema를 지정하지 않으면 public을 기본으로 사용한다.

search_path는 schema 탐색 범위를 의미한다.

```
postgres=# show search_path;
   search_path
-----------------
 "$user", public
(1 row)

postgres=# create schema new_schema;
CREATE SCHEMA
postgres=# SET search_path TO public, new_schema;
SET
postgres=# show search_path;
    search_path
-------------------
 public, new_schema
(1 row)
```

기본 정보 확인 쿼리

쿼리 설명	MySQL	PostgreSQL
데이터베이스(schema) 확인	show databases;	\dn
테이블 확인	show tables;	\dt

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| yahwang            |
+--------------------+
```

```
yahwang=# \dn
   List of schemas
    Name    |  Owner
------------+---------
 new_schema | yahwang
 public     | yahwang
(2 rows)
```

```
yahwang=# show search_path;
   search_path
-----------------
 "$user", public
```

```
mysql> use mysql;
Database changed

mysql> show tables;
+-----------------------+
| Tables_in_mysql       |
+-----------------------+
| columns_priv          |
| component             |
```

```
yahwang=# \dt
             List of relations
 Schema |      Name       | Type  |
--------+-----------------+-------+
 public | amazon_reviews | table |
```

참고 : PostgreSQL에서 쿼리로 확인하려면 information_schema를 활용하면 된다.

```
-- \dt를 쿼리문으로 변환
SELECT table_name, table_schema, table_type
FROM information_schema.tables
WHERE table_schema IN ('public');
```

테이블 생성 시 증가하는 컬럼 설정 방법

MySQL은 **AUTO_INCREMENT** 속성 / PostgreSQL은 **serial** 타입으로 지정

참고 : [Using PostgreSQL SERIAL To Create Auto-increment Column]

```
-- MySQL
CREATE TABLE table_name(
   id INT NOT NULL AUTO_INCREMENT,
    ...
-- PostgreSQL
CREATE TABLE table_name(
   id SERIAL,
    ...
```

PostgreSQL 은 정수 / 정수를 정수로 계산한다.

타입 캐스트 또는 하나를 실수로 바꾸면 소수점 계산이 가능하다.

```
SELECT 100 / 3;

-- 해결방법
SELECT CAST(100 AS float) / 3;
SELECT 100.0 / 3;
```

문자열을 숫자 타입으로 변환하는 방법

MySQL은 정수(Integer)타입으로 변환 시 SIGNED(-포함) 혹은 UNSIGNED 타입을 요구한다.

실수 타입은 DECIMAL 타입으로 하고 자리 수 설정까지 해야 한다. PostgreSQL은 같은 용도로 Numeric 타입이 있다.

```
-- INTEGER
SELECT CAST('777' AS UNSIGNED);
SELECT CAST('-777' AS SIGNED);

-- FLOAT(DOUBLE)
-- DECIMAL(M, D) M은 총 자리 수 D는 소수 점 자리수를 의미한다.
SELECT CAST('77.77' as DECIMAL(4, 2));
-- INTEGER
SELECT CAST('777' AS INTEGER);
SELECT '-777'::INTEGER;
```

```
-- FLOAT(DOUBLE)
SELECT CAST('77.77' AS FLOAT);
SELECT CAST('77.77' AS DOUBLE PRECISION);
SELECT CAST('77.77' as NUMERIC(4, 2));
```

MySQL 은 문자열을 숫자 타입으로 변환 시 오류를 알려주지 않는다.

PostgreSQL은 오류를 바로 나타내는 반면, MySQL은 값을 0으로 바꿔버려서 주의가 필요하다.

참고 : Hive는 NULL로 표현한다고 한다.

```
-- MySQL
SELECT CAST('DBDFD' AS SIGNED); -- => 0
```

MySQL 은 문자열 비교 시 case-insensitive하다.

case-insensitive는 대소문자를 구분하지 않는다는 의미이다. **LIKE**를 사용할 때도 MySQL과 PostgreSQL 방식이 다르다.

참고 : SQL에서 패턴을 찾아주는 LIKE 활용하기

```
SELECT 'Hello' =  'hello' -- => True

-- PostgreSQL => case-sensitive
SELECT 'Hello' =  'hello' -- => False
```

PostgreSQL 은 작은 따옴표 / 큰 따옴표 사용을 명확히 구분

작은 따옴표(single quote)는 string을 표현하고 큰 따옴표(double quotes)는 컬럼명과 테이블명 같은 identifier 네이밍에 활용된다.

```
-- MySQL
SELECT "HELLO"; -- => HELLO

-- PostgreSQL
SELECT "HELLO"; -- => Syntax Error
SELECT 'HELLO'; -- => HELLO
SELECT 'HELLO' AS "Postgres String";
```

문자열 안에 작은 따옴표가 들어가야 할 경우, **''** 형태로 사용하면 된다.

```
-- PostgreSQL
SELECT 'yahwang''s blog'; -- => yahwang's blog

-- MySQL - 큰 따옴표 활용 가능
SELECT "yahwang's blog";
SELECT 'yahwang''s blog';
```

PostgreSQL은 기본적으로 모든 identifier를 lower-case(소문자)로 인식한다.

테이블명이나 컬럼명에 대문자가 있다면 “first_Name”처럼 큰 따옴표(double quotes)를 사용해야 한다. (생성할 때도 포함)

PostgreSQL 은 오른쪽 공백이 들어간 문자를 다르게 인식한다.

참고 : MySQL에서 ‘a’ = ‘a ‘가 true로 평가된다? - 우아한형제 기술블로그

```
-- MySQL
SELECT 'hello' = 'hello   '; -- => True

-- PostgreSQL
SELECT 'hello' = 'hello   '; -- => False
```

MySQL 은 HAVING 절에 ALIAS를 허용한다.

표준 SQL에서는 SELECT보다 GROUP BY, HAVING 연산이 먼저 수행한다. (the logical order of processing)

그래서, ALIAS를 허용하지 않는 것이 원칙이다. 그러나, GROUP BY 절에서는 둘 다 ALIAS를 사용할 수 있다.

단, HAVING 절은 MySQL만 사용 가능하다. (MySQL에서는 HAVING 절의 ALIAS가 필수인 듯)

```
-- MySQL
SELECT EXTRACT(MONTH FROM date) as month, SUM(sales) as total_sum
FROM sample
GROUP BY EXTRACT(MONTH FROM date)
HAVING month > 2;
-- HAVING EXTRACT(MONTH FROM date) > 2; 은 오류 발생

-- PostgreSQL
SELECT EXTRACT(MONTH FROM date) as month, SUM(sales) as total_sum
FROM sample
GROUP BY EXTRACT(MONTH FROM date)
HAVING EXTRACT(MONTH FROM date) > 2;
```

주의할 점은 PostgreSQL의 HAVING 절을 MySQL에 사용하면 date 컬럼을 오히려 인식하지 못하는 오류가 생길 수 있다. (정확한 이유는 찾지 못함)

db-fiddle에서 확인 - MySQL 5

참고 : MySQL Handling of GROUP BY

MySQL 특정 함수

`IF 함수`

CASE WHEN 대신 SELECT 절에 활용 가능 (쿼리문이 간결해지는 효과)

```
-- MySQL에만 IF문이 존재
SELECT IF(5-3 > 0, 'TRUE', 'FALSE');

-- PostgreSQL
SELECT CASE WHEN 5-3 > 0 THEN 'TRUE' ELSE 'FALSE' END;
```

`IFNULL 함수`

IFNULL은 추가 인자로 한 개만 가능 / PostgreSQL에서는 COALESCE 함수로 사용

IFNULL은 첫번째 인자가 NULL이라면 다음 인자값을 리턴하는 의미

(COALESCE는 NULL이 아닌 값이 처음 나오는 값을 리턴하므로 같은 함수는 아니다.)

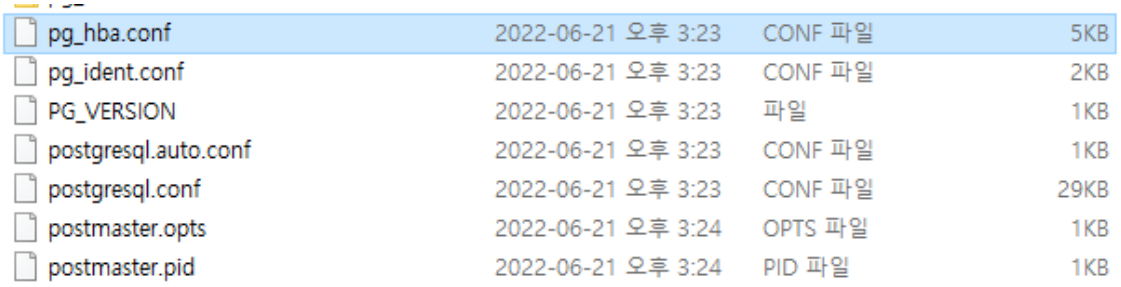
```
-- MySQL에만 IFNULL 문이 존재
SELECT IFNULL(NULL, 'IS NULL');

-- PostgreSQL
SELECT COALESCE(NULL, 'IS NULL');
```

사용자 옵션 목록

Option	Option
SUPERUSER | NOSUPERUSER	해당 USER를 SUPERUSER권한을 주는 것입니다. 따로 지정하지 않을 경우 DEFAULT값으로 NOSUPERUSER가 됩니다.
CREATEDB | NOCREATEDB	DATABASE를 생성하는 권한을 정의합니다. CREATEDB를 선택할 경우 USER는 DATABASE를 생성할 권한이 부여됩니다. NOCREATEDB를 선택할 경우 USER는 DATABASE를 생성할 권한이 거부됩니다. 따로 정의 되어있지 않을 경우 NOCREATEDB값이 default값으로 설정 되어 있습니다.
CREATEUSER | NOCREATEUSER	스스로 새로운 유저를 생성하는 권한을 부여하는 것을 정의합니다. CREATEUSER를 선택할 경우 USER를 생성할 수 있는 권한이 부여됩니다. NOCREATEUSER를 선택할 경우 USER를 생성할 권한이 거부됩니다.
INHERIT | NOINHERIT	DATABASE의 권한을 다른 구성원들에게 상속하는 역할을 합니다. 따로 정의되어있지 않을 경우 INHERIT 값이 default값으로 설정 되어 있습니다.
LOGIN | NOLIGIN	USER가 LOGIN을 하는 역할을 부여한다.
CONNECTION LIMIT connlimit	로그인 할 때 동시연결을 지원 하는 기능으로 default값으로 -1(제한없음)로 설정 되어 있습니다.
[ENCRYPTED | UNCRYPTED] PASSWORD 'password"	‘password’를 입력하고 인증이 필요 없는 경우 옵션을 생략이 가능합니다.

data 폴더 설정



pg_hba.conf	2022-06-21 오후 3:23	CONF 파일	5KB
pg_ident.conf	2022-06-21 오후 3:23	CONF 파일	2KB
PG_VERSION	2022-06-21 오후 3:23	파일	1KB
postgresql.auto.conf	2022-06-21 오후 3:23	CONF 파일	1KB
postgresql.conf	2022-06-21 오후 3:23	CONF 파일	29KB
postmaster.opts	2022-06-21 오후 3:24	OPTS 파일	1KB
postmaster.pid	2022-06-21 오후 3:24	PID 파일	1KB

pg_hba.conf	**ip대역별 접근권한을 설정한다. (Host Base Authentication)**
pg_ident.conf	사용자 계정별 접근권한을 설정한다. (pg_hba.conf method설정에 따라)
postgresql.conf	다음 포스팅에서 상세히 다뤄보기로 한다. 기본포트는 5432

pg_hba.conf

```
# 위쪽에 #하고 많이 뭔가 써져있는데 다 주석임

...

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
# IPv6 local connections:
host    all             all             ::1/128                 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
```

> **Host Type**

접근자의 접근위치와 통신의 암호화 관련 설정이다.
local / host / hostssl / hostnossl 중 한가지를 사용한다.

local Unix Domain Socket접속용이므로 사용하지 않아도 된다.
host 일반접속과 ssl(보안서버)접속을 모두 허용한다.
hostssl ssl인증서를 통한 암호화 통신만 지원한다.
hostnossl ssl접속을 지원하지 않는다.

> **Database Name (데이터베이스 명)**

all 모든 데이터베이스에 영향을 주는 규칙을 만든다
db명을 기입하면 해당 데이터베이스만 영향을 주는 규칙을 만든다.
여러개의 db명을 콤마(,)로 분리하여 입력이 가능하다.

> **User Name (계정명)**

all 모든 계정에 영향을 주는 규칙을 만든다.
계정을 기입하면 해당 계정만 영향을 주는 규칙을 만든다.
데이터베이스와 마찬가지로 콤마(,)로 분리하여 입력이 가능하다.

> **CIDR-ADDRESS**

IP 규칙을 만든다.
IPv4 C-Class를 처리를 할 경우는 : xxx.xxx.xxx.0/24
해당 IP에 대한 처리를 할 경우는 : xxx.xxx.xxx.xxx/32

> **Authentication Method**

패스워드의 전송방식을 설정한다.
trust / reject / md5 / password / 기타등등의 패스워드 전달방식을 설정하지만 md5를 설정하면된다.
단 유의할것은 PG 과거 버전에서 설치후 trust가 기본적으로 설정된다.
trust는 비밀번호가 없이도 바로 접근이 가능하므로 반드시 md5등의 암호화 방법으로 바꾸어서 운영하자.

trust 패스워드 없이 접근 가능
reject 거부
md5 md5암호화 전송
password text로 전송

postgresql.conf

postgresql.conf 파일은 해당 데이터베이스의 환경설정 파일이다.

오라클을 접해본 사람이라면 sqlnet.ora 파일과 유사하다고 이해하면 된다.

pg_hba.conf 와 마찬가지로 '#'은 주석문이다.

자주 바꾸는 설정들

listen_addresses = 'localhost'

서버를 초기화하면 기본적으로 'localhost'로 설정된다. 이 상황에서는 로컬pc에서만 DB 접근이 가능하다.'*' 로 바꾸어야 모든 접근이 가능하다.
서버 의 네트워크 카드가 다중이고 특정 IP로의 접근만을 허용하고자 한다면 해당 어댑터에 설정한 IP를 기입하여 제한적으로 허용이 가능하다.

port = 5432

PostgreSQL의 기본 Port는 5432 이다.

원하는 포트로 바꾸어 운영할 수 있다.

max_connections = 100

최대 동시접속자의 수를 설정하는 옵션이다. 상황에 따라 원하는 동시접속자 수를 설정한다.

superuser_reserved_connections = 3

Superuser 의 동시접속 수를 설정하는 옵션이다.

authentication_timeout = 1min

인증대기 오류 Timeout이며 기본 60초이다.

상황에 따라 짧게 설정 하여 운영할수 있다.

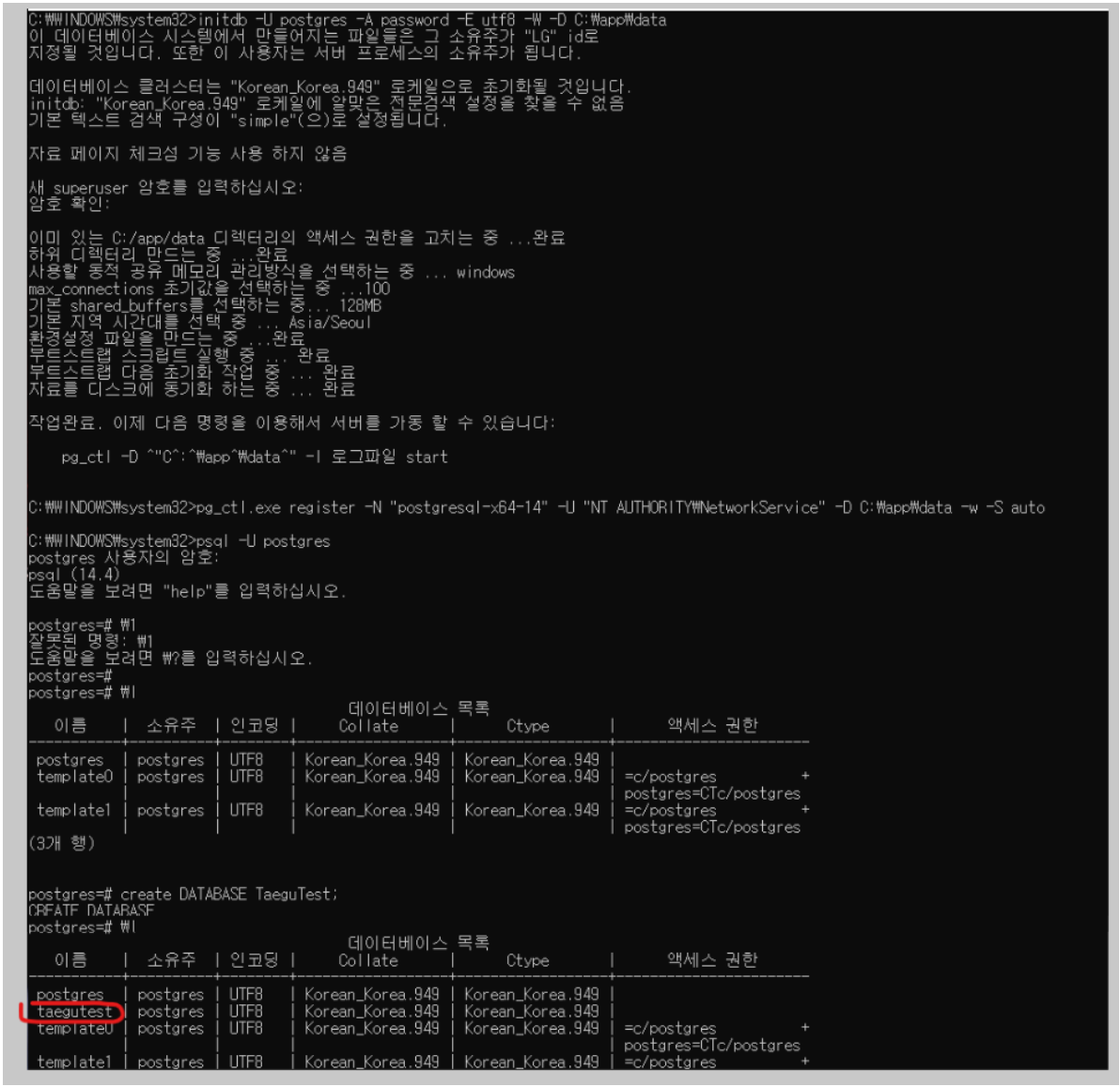
```
authentication_timeout = 30    #30초
authentication_timeout = 60    #1분
authentication_timeout = 1min  #30초
```

ssl = false

SSL인증을 허용한다. (인증서 필요)
pg_hba.conf 설정과 DB접속시 ConnectionString에 SSL부분을 추가해야한다.

archive_mode = off

아카이브 모드를 사용한다. (off / on / always)



```
C:₩WINDOWS₩system32>initdb -U postgres -A password -E utf8 -W -D C:₩app₩data
이 데이터베이스 시스템에서 만들어지는 파일들은 그 소유주가 "LG" id로
지정될 것입니다. 또한 이 사용자는 서버 프로세스의 소유주가 됩니다.

데이터베이스 클러스터는 "Korean_Korea.949" 로케일으로 초기화될 것입니다.
initdb: "Korean_Korea.949" 로케일에 알맞은 전문검색 설정을 찾을 수 없음
기본 텍스트 검색 구성이 "simple"(으)로 설정됩니다.

자료 페이지 체크섬 기능 사용 하지 않음

새 superuser 암호를 입력하십시오:
암호 확인:

이미 있는 C:/app/data 디렉터리의 액세스 권한을 고치는 중 ...완료
하위 디렉터리 만드는 중 ...완료
사용할 동적 공유 메모리 관리방식을 선택하는 중 ... windows
max_connections 초기값을 선택하는 중 ...100
기본 shared_buffers를 선택하는 중... 128MB
기본 지역 시간대를 선택 중 ... Asia/Seoul
환경설정 파일을 만드는 중 ...완료
부트스트랩 스크립트 실행 중 ... 완료
부트스트랩 다음 초기화 작업중 ... 완료
자료를 디스크에 동기화 하는 중 ... 완료

작업완료. 이제 다음 명령을 이용해서 서버를 가동 할 수 있습니다:

    pg_ctl -D ^"C^:^₩app^₩data^" -l 로그파일 start


C:₩WINDOWS₩system32>pg_ctl.exe register -N "postgresql-x64-14" -U "NT AUTHORITY₩NetworkService" -D C:₩app₩data -w -S auto

C:₩WINDOWS₩system32>psql -U postgres
postgres 사용자의 암호:
psql (14.4)
도움말을 보려면 "help"를 입력하십시오.

postgres=# ₩l
잘못된 명령: ₩l
도움말을 보려면 ₩?를 입력하십시오.
postgres=#
postgres=# ₩l
                                  데이터베이스 목록
   이름    |  소유주  | 인코딩 |     Collate     |     Ctype       |      액세스 권한
-----------+----------+--------+-----------------+-----------------+-----------------------
 postgres  | postgres | UTF8   | Korean_Korea.949 | Korean_Korea.949 |
 template0 | postgres | UTF8   | Korean_Korea.949 | Korean_Korea.949 | =c/postgres          +
           |          |        |                 |                 | postgres=CTc/postgres
 template1 | postgres | UTF8   | Korean_Korea.949 | Korean_Korea.949 | =c/postgres          +
           |          |        |                 |                 | postgres=CTc/postgres
(3개 행)


postgres=# create DATABASE TaeguTest;
CRFATF DATABASF
postgres=# ₩l
                                  데이터베이스 목록
   이름    |  소유주  | 인코딩 |     Collate     |     Ctype       |      액세스 권한
-----------+----------+--------+-----------------+-----------------+-----------------------
 postgres  | postgres | UTF8   | Korean_Korea.949 | Korean_Korea.949 |
 taegutest | postgres | UTF8   | Korean_Korea.949 | Korean_Korea.949 |
 template0 | postgres | UTF8   | Korean_Korea.949 | Korean_Korea.949 | =c/postgres          +
           |          |        |                 |                 | postgres=CTc/postgres
 template1 | postgres | UTF8   | Korean_Korea.949 | Korean_Korea.949 | =c/postgres          +
```

현재

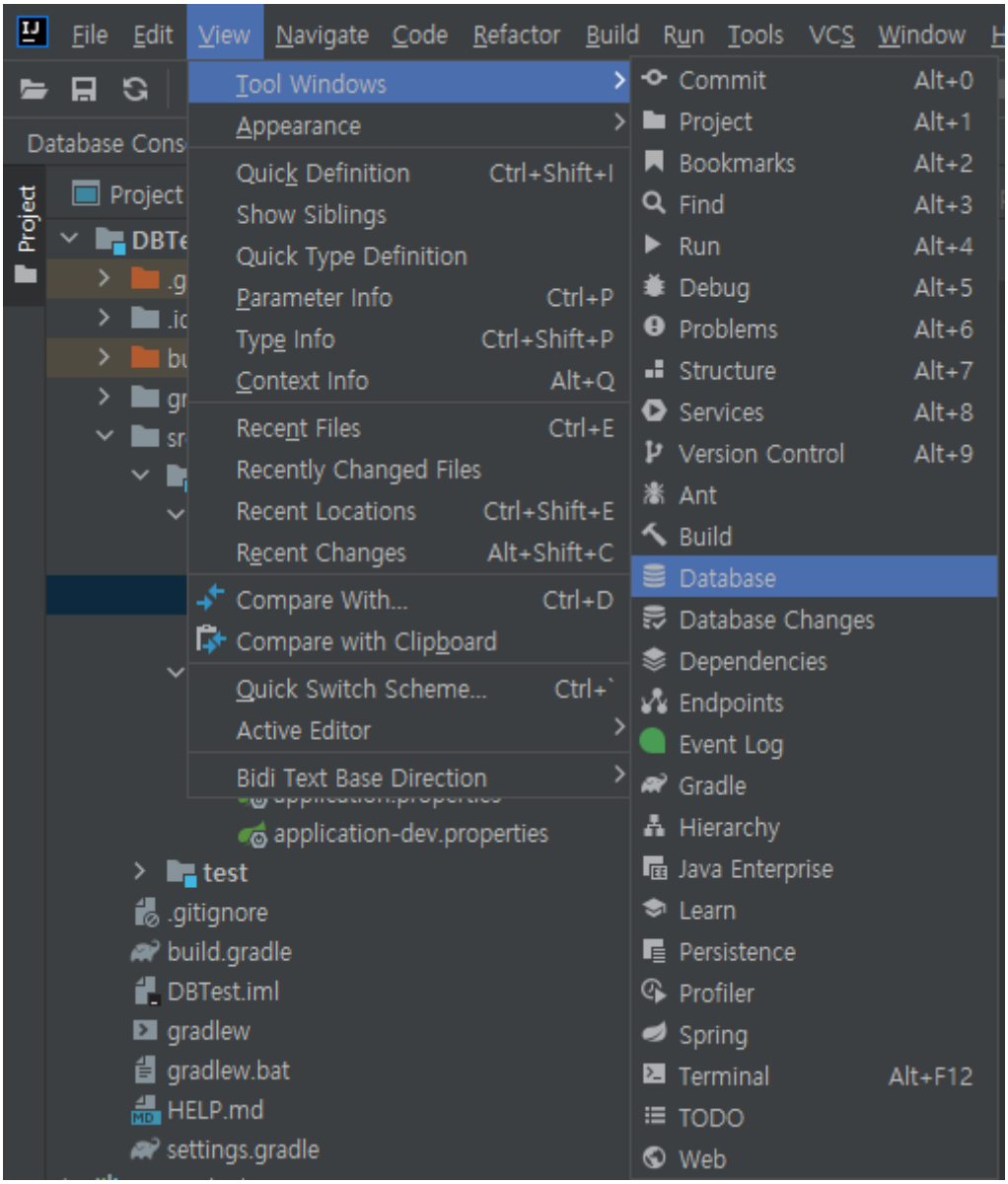
user : postgres

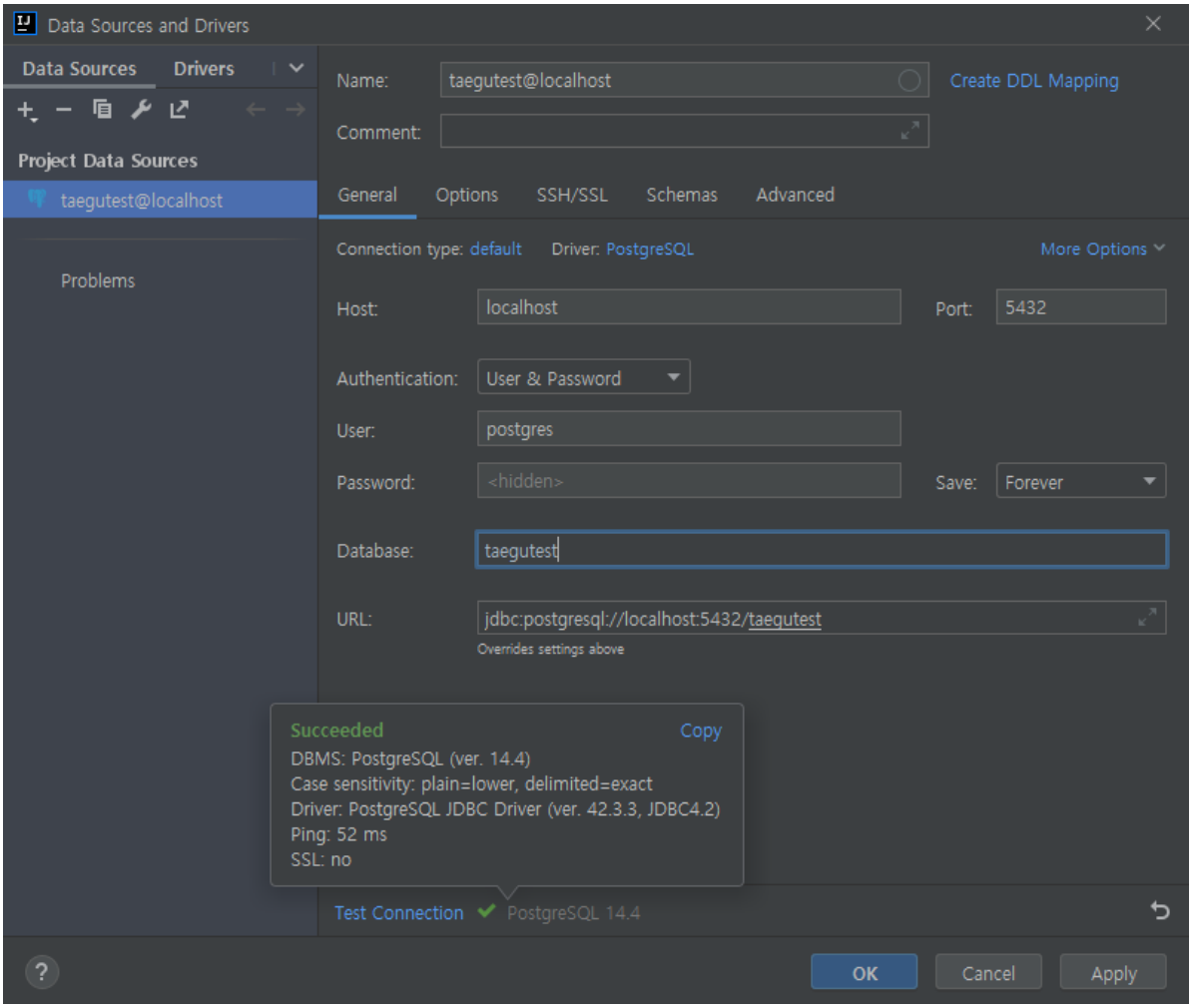
password : 0000

DATABASE : taegutest

psql -U postgres

0000


File Edit View Navigate Code Refactor Build Run Tools VCS Window
Tool Windows
Appearance
Quick Definition Ctrl+Shift+I
Show Siblings
Quick Type Definition
Parameter Info Ctrl+P
Type Info Ctrl+Shift+P
Context Info Alt+Q
Recent Files Ctrl+E
Recently Changed Files
Recent Locations Ctrl+Shift+E
Recent Changes Alt+Shift+C
Compare With... Ctrl+D
Compare with Clipboard
Quick Switch Scheme... Ctrl+`
Active Editor
Bidi Text Base Direction
Commit Alt+0
Project Alt+1
Bookmarks Alt+2
Find Alt+3
Run Alt+4
Debug Alt+5
Problems Alt+6
Structure Alt+7
Services Alt+8
Version Control Alt+9
Ant
Build
Database
Database Changes
Dependencies
Endpoints
Event Log
Gradle
Hierarchy
Java Enterprise
Learn
Persistence
Profiler
Spring
Terminal Alt+F12
TODO
Web
Database Cons
Project
application-dev.properties
test
.gitignore
build.gradle
DBTest.iml
gradlew
gradlew.bat
HELP.md
settings.gradle


Data Sources and Drivers
Data Sources
Drivers
Project Data Sources
taegutest@localhost
Problems
Name:
taegutest@localhost
Create DDL Mapping
Comment:
General
Options
SSH/SSL
Schemas
Advanced
Connection type: default
Driver: PostgreSQL
More Options
Host:
localhost
Port:
5432
Authentication:
User & Password
User:
postgres
Password:
<hidden>
Save:
Forever
Database:
taegutest
URL:
jdbc:postgresql://localhost:5432/taegutest
Overrides settings above
Succeeded
Copy
DBMS: PostgreSQL (ver. 14.4)
Case sensitivity: plain=lower, delimited=exact
Driver: PostgreSQL JDBC Driver (ver. 42.3.3, JDBC4.2)
Ping: 52 ms
SSL: no
Test Connection
PostgreSQL 14.4
OK
Cancel
Apply