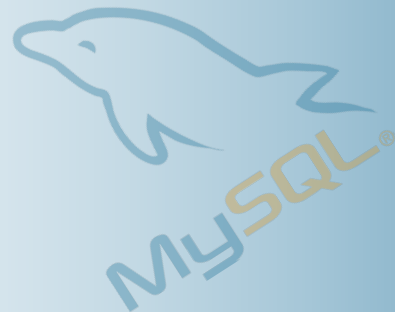
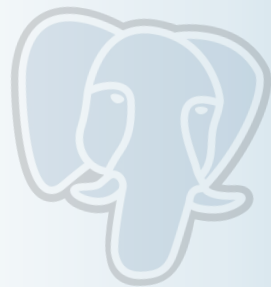


PostgreSQL

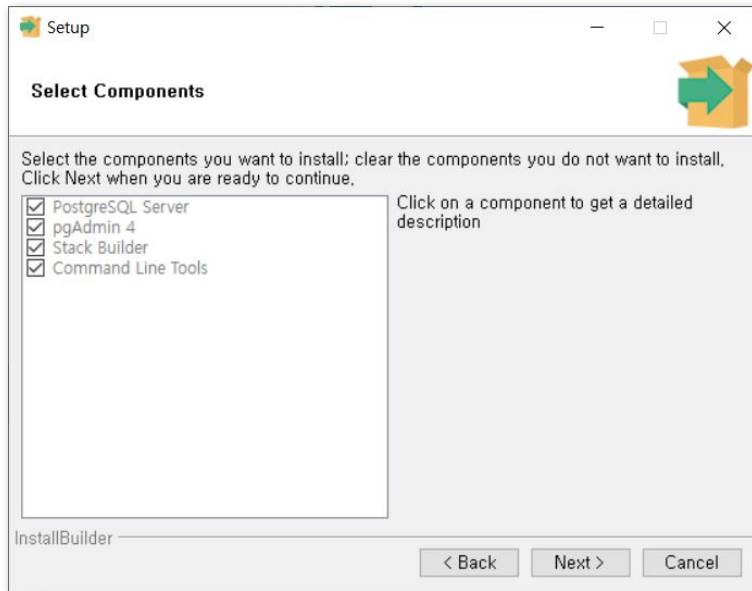
- 설치 및 사용 방법
- **MySQL**과 **PostgreSQL**의 차이
  - 쿼리 실행방법과 문법 차이
  - 다양한 자료형
  - Schema 구분
- **MVCC**구현방식의 차이와 **Vacuum**
  - MVCC란 무엇인가?
  - MySQL과 PostgreSQL의 MVCC 및 단점
    - Dead Tuple과 Vacuum
    - Transaction ID Wraparound와 Freeze
- **Java**와 **Node.js**에서 사용하는 방법
- 처리 속도 비교



# PostgreSQL 설치방법

<https://www.postgresql.org/download/>

1. 홈페이지에서 자신의 OS에 맞는 설치파일을 다운받아 실행한다
2. 컴포넌트를 고르는 단계에서 Server와 pgAdmin이 체크되어 있는지 확인한다
3. **root** 계정 암호를 잘 기억한다.
4. 설치가 끝난 후 Launch Stack Builder at exit? 체크를 해제한 뒤 끝낸다



# PostgreSQL 설치방법

설치가 완료되면 시작메뉴에 관련 프로그램이 표시되지 않는다

```
C:\Program Files\PostgreSQL\16\pgAdmin 4\runtime\pgAdmin4.exe
```

위 경로에 있는 pgAdmin4.exe를 실행하면

**PostgreSQL**의 GUI 관리 프로그램이 실행된다

# MySQL과

# PostgreSQL의 차이

- 쿼리 실행 방법
- 문법 차이
- 자료형
- Schema

# 쿼리 실행 방법

## MySQL

현재 커서가 있는 쿼리를

Ctrl + Enter로 실행

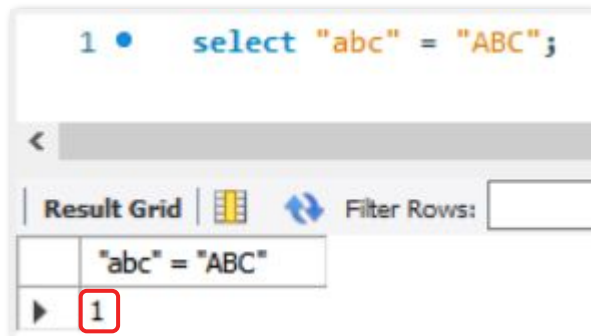
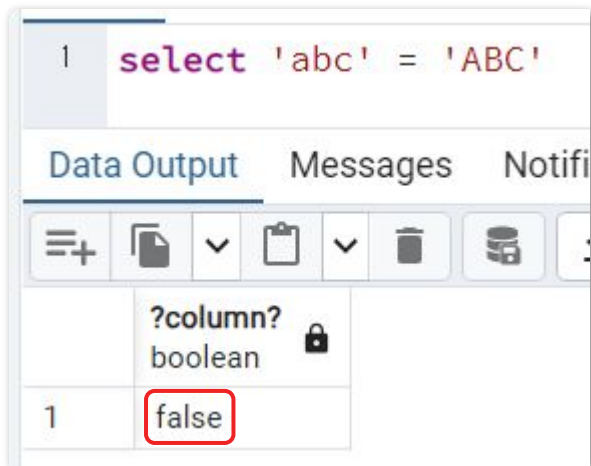
## PostgreSQL

드래그해서 블록잡은 쿼리를

F5로 실행

# 문법 차이

- 대소문자 구분



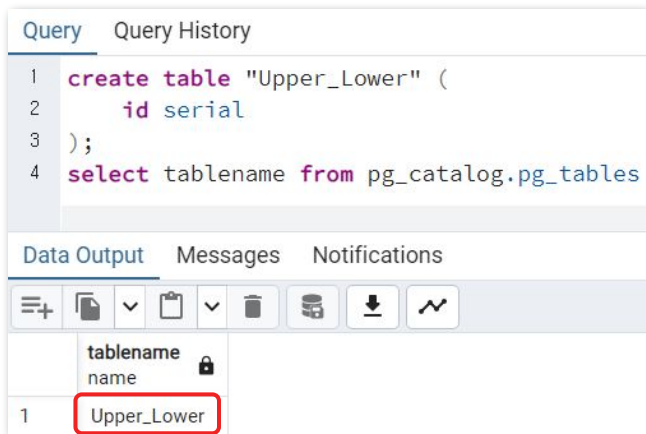
**PostgreSQL**은 대소문자를 구분한다

**MySQL**은 대소문자를 구분하지 않는다

# 문법 차이

- 쌍따옴표

**PostgreSQL**과 **MySQL**은 여러 요소 생성시 무조건 소문자로 적용되지만 **PostgreSQL**의 경우 쌍따옴표(")로 감싸면 대문자가 적용된다



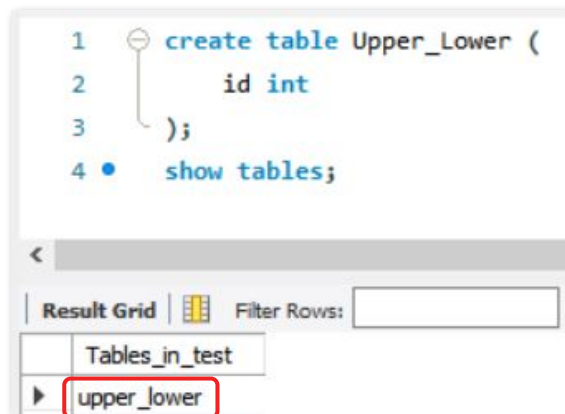
The screenshot shows a PostgreSQL query editor with the following SQL code:

```
1 create table "Upper_Lower" (  
2     id serial  
3 );  
4 select tablename from pg_catalog.pg_tables
```

Below the query editor, the 'Data Output' tab is active, showing a table with two columns: 'tablename' and 'name'. The first row of the result set is highlighted, showing 'Upper\_Lower' in the 'name' column, which is circled in red.

tablename	name
	Upper_Lower

**PostgreSQL**



The screenshot shows a MySQL query editor with the following SQL code:

```
1 create table Upper_Lower (  
2     id int  
3 );  
4 show tables;
```

Below the query editor, the 'Result Grid' tab is active, showing a table with one column: 'Tables\_in\_test'. The first row of the result set is highlighted, showing 'upper\_lower' in the column, which is circled in red.

Tables_in_test
upper_lower

**MySQL**



# 문법 차이

- **PostgreSQL**에서 쌍따옴표(")는 대소문자를 구분하는데 사용되므로 문자열을 표시할 때는 따옴표(')만 사용 가능하다
- 그 외에도 SHOW, DESC 등 **MySQL**에서 되던 쿼리들이 **PostgreSQL**에선 안되는 경우가 존재한다

# 자료형

PostgreSQL은 MySQL처럼 여러 자료형을 지원한다

## Serial

int형에  
auto increment과  
not Null이  
기본적으로  
포함되어있는 자료형

## Character varying

MySQL의  
varchar와 같음

## Array

PostgreSQL의  
특징으로  
Json 뿐만 아니라  
각 자료형의 배열도  
지원함

# Schema

## Schema란?

데이터가 담겨있는 **Table**들을 논리적으로 구분하는 기준이다

> 같은 **Database**에 속한다면 다른 **Schema**의 **Table**이라도 참조가 가능

**MySQL**은 단일 **Database**를 사용하며, **Schema**로만 테이블을 구분한다

> 모든 **Database(Schema)**가 서로 참조 가능

# Schema

## PostgreSQL



test\_db/postgres@PostgreSQL 16

Query Query History

```
1 select * from test_db2.testttt;
```

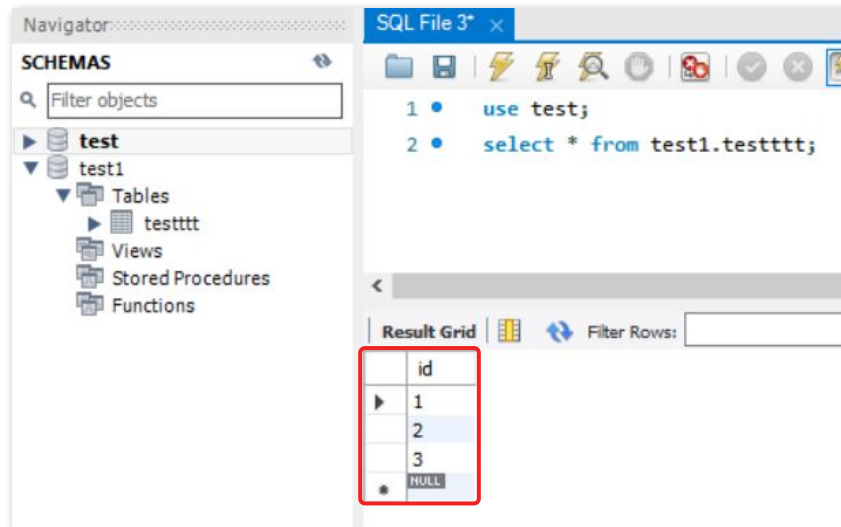
Data Output Messages Notifications

ERROR: "test\_db2.testttt" 이름의 릴레이션 (relation)이 없습니다  
LINE 1: select \* from test\_db2.testttt;  
                                  ^

오류: "test\_db2.testttt" 이름의 릴레이션 (relation)이 없습니다  
SQL state: 42P01  
Character: 15

다른 DB에 있는 테이블을  
참조할 수 없음

## MySQL



Navigator: SCHEMAS

Filter objects

test

test1

Tables

testttt

Views

Stored Procedures

Functions

SQL File 3\* x

```
1 use test;  
2 select * from test1.testttt;
```

Result Grid

	id
▶	1
	2
	3
*	HULL

다른 DB(Schema)에 있는 테이블을  
참조할 수 있음

# MVCC 구현 방식 차이와 Vacuum

- MVCC란 무엇이고 왜 필요한가?
- MySQL의 MVCC와 단점
- PostgreSQL의 MVCC와 발생하는 문제점
- PostgreSQL의 Vacuum기능

# 다중 버전 동시성 제어(MVCC)

- 동시성 제어 - Concurrency Control

여러 사용자가 동시에 DB에 접근해 읽기, 쓰기를 할 경우  
데이터의 일관성을 보장하는것을 말한다

- 다중 버전 동시성 제어 - Multiversion Concurrency Control

동시성 제어의 방법중 하나이다

하나의 데이터에 여러 버전이 존재하고,  
**잠금을 사용하지 않는다**

# MySQL의 MVCC

## Undo Segment (Rollback Segment)

Transaction에서 변경된 내용은 모두 **Undo Log**라는곳에 저장되고,  
Undo Log에 저장된 항목은 자신 앞의 변경내용을 포인터로 가리킨다

새 Transaction이 수행되면 이 Undo Log를 쫓 검색해서  
자신이 읽을 수 있는 시점의 데이터를 불러온다

Transaction이 길어질수록 Undo Log가 계속 길어지게 된다

> 읽기 성능에 영향을 줌

# PostgreSQL의 MVCC

Transaction에서 변경된 데이터의 전, 후 내용(Tuple)을 Page에 같이 저장한다

저장할 때 변경된 시점을 xmin, xmax라는 Metadata Field에 같이 저장한다

이 시점을 비교하여 읽을 수 있는 데이터를 불러오게 된다



# PostgreSQL의 MVCC

## Dead Tuple

Transaction이 Commit 되거나 Rollback 되면  
더이상 사용되지 않는 **Tuple**이 생기게 된다  
이것을 **Dead Tuple**이라고 부른다

**Dead Tuple**은 용량과 자리를 그대로 차지하기 때문에  
쌓이게 되면 읽기 성능에 영향을 준다

# PostgreSQL의 MVCC

## Vacuum

Dead Tuple을 정리하기위해서

**PostgreSQL**에는 **Vacuum**이라는 기능이 존재한다

**Vacuum**을 사용하면 **Dead Tuple**을 정리하여 처리속도를 최적화 할 수 있다

내부 알고리즘에 따라 임계치를 넘어가게 되면

자동으로 **AutoVacuum**이라는 것이 실행되어 **Dead Tuple**이 정리된다

# PostgreSQL의 MVCC

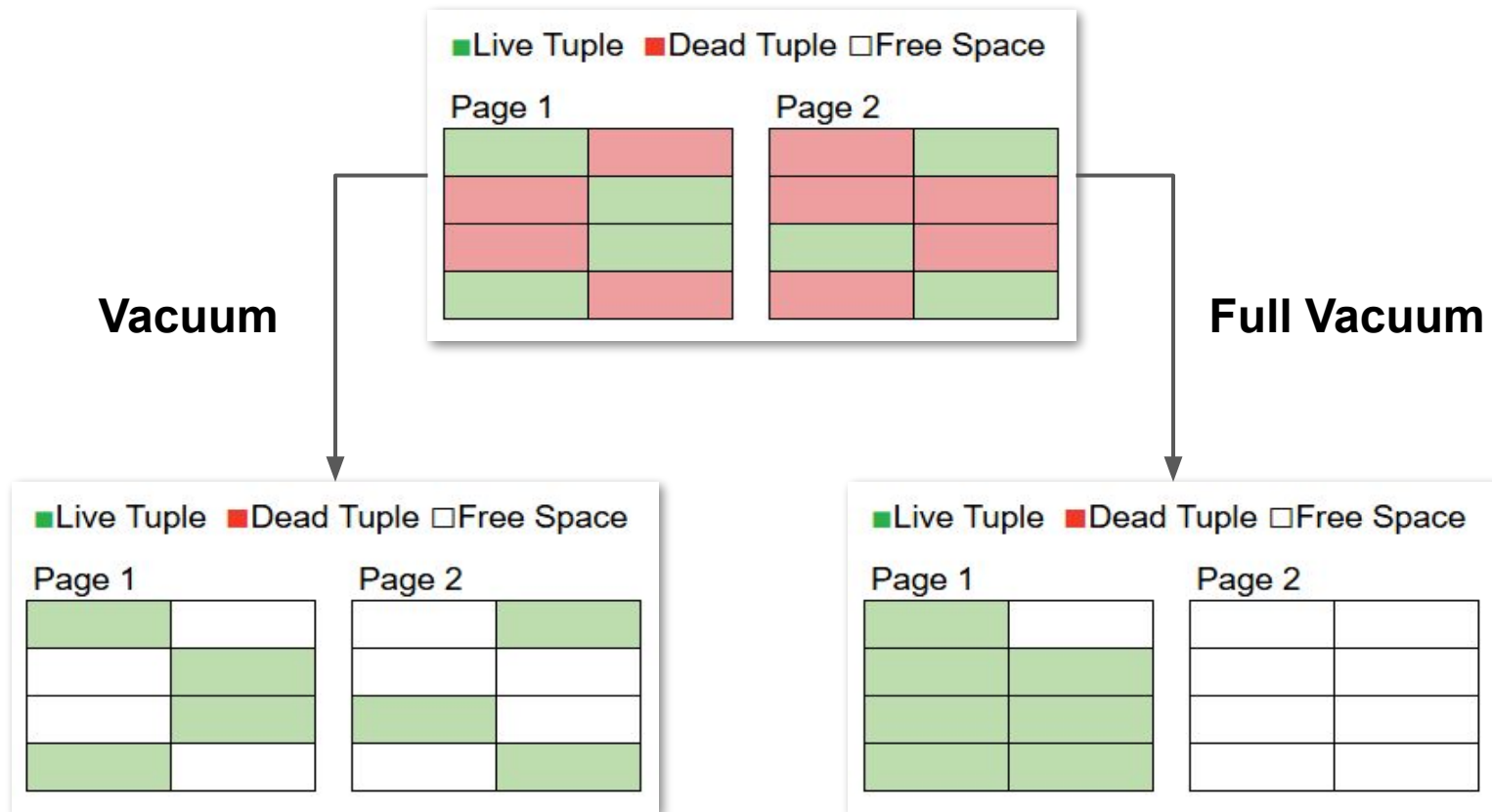
## Vacuum Full

Vacuum을 사용하면 Dead Tuple을 지워 재사용 가능하게 하지만 공간까지 최적화 하지는 못해 Table이 차지하는 공간은 변하지 않는다

Vacuum Full을 사용하면 Dead Tuple과 비어있는 공간을 정리할 수 있다

하지만 대상 Table을 복사하여 처리하기 때문에 용량에 여유가 필요하고 Select를 포함한 모든 작업에 잠금이 걸리기 때문에 신중하게 실시해야 한다

# PostgreSQL의 MVCC



# PostgreSQL의 MVCC

## Transaction ID Wraparound

PostgreSQL은 데이터의 변경 시점을 xmin, xmax라는 Transaction ID(XID)로 저장한다

비교에 사용되는 xmin, xmax는 4 Byte 값으로  
약 43억( $2^{32}$ )개의 값을 표현할 수 있다

현재 XID의 절반은 과거, 나머지는 미래를 표현하는데 사용한다

하지만 이 XID는 무한하지 않아서 결국 끝이 존재한다

# PostgreSQL의 MVCC

## Transaction ID Wraparound

XID가  $2^{32}$ 을 넘어서게 되면 값에 Overflow가 일어나고

Transaction ID Wraparound가 발생하게 된다

이는 과거와 미래의 데이터가 서로 뒤섞이는 **매우 심각한** 문제이다

# PostgreSQL의 MVCC

## Transaction ID Wraparound

이러한 문제를 방지하기 위해서

현재 **XID** - 생성시점 **XID**가  $2^{32}$ 의 절반을 넘기 전에

모든 데이터를 정리해 xmin의 **XID**를 2(Frozen XID)로 변경한다

이 작업을 **Freeze**(or Anti Wraparound Vacuum)라고 부른다

# PostgreSQL의 MVCC

## Transaction ID Wraparound

**PostgreSQL 9.4**버전 이후부터는 Freeze 방식이 바뀌었다

새 버전부터는 xmin값은 그대로 유지하고,

1bit짜리 Flag값을 추가하여 Freeze된 데이터라는것을 표기한다

이렇게 하면 사고 원인을 분석할 때 더 유리해진다



# PostgreSQL의 MVCC

## Transaction ID Wraparound

Freeze를 할 대상을 구분하기 위해 **Age**라는 값이 도입되었다

Age는 Tuple이 생성되었을 때 1부터 시작되고

Transaction이 발생할 때 마다 1씩 증가한다

즉 Age는 생성지점 **XID**와 현재 **XID**의 차이를 의미한다

Age 값이 **AUTOVACUUM\_FREEZE\_MAX\_AGE**에 도달하면

AutoVacuum으로 Freeze가 자동으로 실행된다

# Java 및 Node.js에서 사용하는 방법

- Java에서 사용하는 방법
- Node.js에서 사용하는 방법

# Java에서 PostgreSQL 사용하기

## MySQL

<https://dev.mysql.com/downloads/connector/j/>

**Postgres** 홈페이지에서 **JDBC** 라이브러리 다운받아 추가하기

## PostgreSQL

<https://jdbc.postgresql.org/download/>

**MySQL** 홈페이지에서 **JDBC** 라이브러리 다운받아 추가하기

# Java에서 PostgreSQL 사용하기

## MySQL

```
final String url = "jdbc:mysql://192.168.80.39:3306/test";  
final String user = "test";  
final String passwd = "1234";  
Connection connect = DriverManager.getConnection(url, user, passwd);  
  
connect.close();
```

## PostgreSQL

```
final String url = "jdbc:postgresql://127.0.0.1:5432/test_db";  
final String user = "test";  
final String passwd = "1234";  
Connection connect = DriverManager.getConnection(url, user, passwd);  
  
connect.close();
```

# Java에서 PostgreSQL 사용하기

## MySQL

```
String sql = "select <Column> from <Table>";  
PreparedStatement ps = connect.prepareStatement(sql);  
ResultSet rs = ps.executeQuery(sql);  
while (rs.next()) {  
    // 처리 코드  
}
```

## PostgreSQL

```
String sql = "select <Column> from <Table>";  
Statement st = connect.createStatement();  
ResultSet rs = st.executeQuery(sql);  
while (rs.next()) {  
    // 처리 코드  
}
```

# Node.js에서 PostgreSQL 사용하기

```
PS C:\Users\Playdata\Documents\projects\temp> npm install pg  
  
added 16 packages, and audited 17 packages in 3s  
  
found 0 vulnerabilities
```

\$ npm install pg

Node.js 프로젝트에 pg 모듈 설치

정상적으로 모듈이 추가된 모습 ▶

```
▼ node_modules  
  > buffer-writer  
  > packet-reader  
  > pg  
  > pg-cloudflare  
  > pg-connection-string  
  > pg-int8  
  > pg-pool  
  > pg-protocol  
  > pg-types  
  > pgpass  
  > postgres-array  
  > postgres-bytea  
  > postgres-date  
  > postgres-interval  
  > split2  
  > xtend
```

# Node.js에서 PostgreSQL 사용하기

```
const { Client } = require("pg");
```

pg 모듈 불러오기

```
const client = new Client({  
  user: "postgres",  
  host: "127.0.0.1",  
  database: "test_db",  
  password: "1234",  
  port: 5432,  
});
```

DB 서버 연결정보

```
client.connect();
```

DB 서버와 연결

```
client.query("SELECT NOW()", (err, res) => {  
  console.log(err, res.rows[0].now);  
  client.end();  
});
```

쿼리문 실행

# MySQL과 PostgreSQL의 처리 속도 비교

- 측정 방법
- 측정 조건
- 측정 결과
- RDB 비교



# 측정 방법

1. for문으로 Thread를 생성시키고  
실행된 Thread는 while(true)로 대기시킨다
2. Thread 실행을 완료되면 시간을 기록하고  
while(false)로 바꿔 일제히 실행시킨다
3. 각 Thread는 종료될 때 synchronized된 카운트 추가 Method를 호출하여  
Thread가 종료된 상황을 파악할 수 있게 한다
4. 모든 스레드가 종료되면 시간차를 계산해서 출력한다

# 측정 조건

- 각 스레드별로 커넥션을 생성해 유지한채로 작업하고, 작업이 종료되면 커넥션을 닫는다.
- 테스트할 테이블은 int형의 PK인 id 하나에  
MySQL은 varchar(255) 4열,  
PostgreSQL은 character varying 255 4열을 사용한다.
- 매 INSERT 작업마다 Transaction을 생성하고, commit한다.
- INSERT시 넣는 데이터: '123', '123', '123', '123'
- 스레드 완료 확인은 10ms마다 수행한다(오차  $\pm 10$  ms)

# 측정 코드 - Main

```
System.out.print("DB Table 초기화중... ");
c = DriverManager.getConnection(MySQLInit.url, MySQLInit.user, MySQLInit.passwd);
sql = "delete from speed where l=1";
ps = c.prepareStatement(sql);
rs = ps.executeUpdate();
c.close();
System.out.println("완료");
```

초기화

```
System.out.println("\nMySQL - 40 Thread, 100 INSERT");
System.out.print("Thread 생성중... ");
for (int i = 0; i < 40; i++) {
    MysqlThreadInsert mysql = new MysqlThreadInsert();
    mysql.start();
}
```

Thread 생성

```
System.out.println("완료");
time = Instant.now();
Count.mysqlInsertStart = false;
System.out.println("INSERT 시작...");
while (true) {
    try {
        Thread.sleep(10);
    } catch (InterruptedException e) {}
    if (Count.getMysqlInsert() > 39) break;
}
System.out.println("완료. 소모시간: " + (Duration.between(time, Instant.now()).toMillis()) + "ms");
```

실행 및 대기

# 측정 코드 - Thread

```
public void run() {  
    while (Count.mysqlInsertStart) {}  
    try {  
        Connection c = DriverManager.getConnection(MySQLInit.url, MySQLInit.user, MySQLInit.passwd);  
        for (int i = 0; i < 100; i++) {  
            c.setAutoCommit(false);  
  
            String sql = "insert into speed values (null, '123', '123', '123', '123')";  
            PreparedStatement ps = c.prepareStatement(sql);  
            int rs = ps.executeUpdate();  
            c.commit();  
        }  
        c.close();  
        Count.addMysqlInsert();  
    } catch (Exception e) {  
    }  
}
```

# 측정 결과

```
<terminated> Main (8) [Java Application] C:\Program Files\Eclipse
DB Table 초기화중... 완료

MySQL - 20 Thread, 100 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 943ms

MySQL - 50 Thread, 100 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 411ms

DB Table 초기화중... 완료

PostgreSQL - 20 Thread, 100 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 519ms

PostgreSQL - 50 Thread, 100 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 634ms
```

```
<terminated> Main (8) [Java Application] C:\Program Files\Eclipse
DB Table 초기화중... 완료

MySQL - 20 Thread, 100 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 837ms

MySQL - 50 Thread, 100 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 420ms

DB Table 초기화중... 완료

PostgreSQL - 20 Thread, 100 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 356ms

PostgreSQL - 50 Thread, 100 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 631ms
```

```
<terminated> Main (8) [Java Application] C:\Program Files\Eclipse
DB Table 초기화중... 완료

MySQL - 20 Thread, 100 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 937ms

MySQL - 50 Thread, 100 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 338ms

DB Table 초기화중... 완료

PostgreSQL - 20 Thread, 100 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 349ms

PostgreSQL - 50 Thread, 100 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 617ms
```

# 측정 결과

```
<terminated> Main (8) [Java Application] C:\Program Files\Eclipse
DB Table 초기화중... 완료

MySQL - 40 Thread, 100 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 1269ms

MySQL - 100 Thread, 100 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 581ms

DB Table 초기화중... 완료

PostgreSQL - 40 Thread, 100 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 611ms

PostgreSQL - 100 Thread, 100 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 1678ms
```

```
<terminated> Main (8) [Java Application] C:\Program Files\Eclipse
DB Table 초기화중... 완료

MySQL - 40 Thread, 100 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 3586ms

MySQL - 100 Thread, 100 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 851ms

DB Table 초기화중... 완료

PostgreSQL - 40 Thread, 100 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 714ms

PostgreSQL - 100 Thread, 100 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 1938ms
```

# 측정 결과

```
<terminated> Main (8) [Java Application] C:\Program Files\Eclipse
DB Table 초기화중... 완료

MySQL - 1 Thread, 3000 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 11611ms

MySQL - 1 Thread, 5000 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 740ms

DB Table 초기화중... 완료

PostgreSQL - 1 Thread, 3000 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 612ms

PostgreSQL - 1 Thread, 5000 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 433ms
```

```
<terminated> Main (8) [Java Application] C:\Program Files\Eclipse
DB Table 초기화중... 완료

MySQL - 1 Thread, 3000 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 10725ms

MySQL - 1 Thread, 5000 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 589ms

DB Table 초기화중... 완료

PostgreSQL - 1 Thread, 3000 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 548ms

PostgreSQL - 1 Thread, 5000 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 406ms
```

```
<terminated> Main (8) [Java Application] C:\Program Files\Eclipse
DB Table 초기화중... 완료

MySQL - 1 Thread, 3000 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 11710ms

MySQL - 1 Thread, 5000 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 604ms

DB Table 초기화중... 완료

PostgreSQL - 1 Thread, 3000 INSERT
Thread 생성중... 완료
INSERT 시작...
완료. 소모시간: 530ms

PostgreSQL - 1 Thread, 5000 SELECT
Thread 생성중... 완료
SELECT 시작...
완료. 소모시간: 392ms
```

# 측정 결과 평균

	20 Thread 100 Insert	1 Thread 3000 Insert	50 Thread 100 Select	1 Thread 5000 Select
MySQL	905.67 ms	11348.67 ms	389.67 ms	644.33 ms
PostgreSQL	408.00 ms	563.33 ms	627.33 ms	410.33 ms



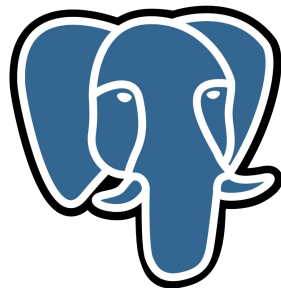
# RDB 비교



MySQL

무료

반복 **Insert**가 매우 느리다  
상대적으로 다루기 쉽다



PostgreSQL

무료

동시 **Select**가 조금 느리다  
반복처리가 매우 빠르다

ORACLE®

ORACLE

유료

비싸다  
지원이 빵빵하다

# PostgreSQL을 사용하는 기업

The logo for Coupang, featuring the word "coupang" in a lowercase, sans-serif font. The letters are colored: 'c' is brown, 'o' is red, 'u' is orange, 'p' is green, 'a' is blue, and 'ng' is dark blue.The logo for Zigbang, featuring an orange icon of a stylized 'Z' or a network node, followed by the word "zigbang" in a lowercase, sans-serif font.The logo for SK, featuring a stylized orange and red icon of a person or a flame, followed by the letters "SK" in a bold, red, sans-serif font.The logo for Uihyehyehye, featuring the Korean text "우아한형제들" in a bold, black, stylized font with a white outline.The logo for Dangn Market, featuring an orange icon of a carrot with a green leaf, followed by the Korean text "당근마켓" in a bold, orange, sans-serif font.The logo for Kakao, featuring the word "kakao" in a bold, yellow, sans-serif font.