

# 목 차

**01** 분산 데이터베이스 개요

**02** 파티셔닝 (Partitioning)

**03** 샤딩 (Sharding)

**04** 레플리케이션 (Replication)

**05** 구현 예시

**06** 비교 분석

**07** Q&A

# 1. 분산 데이터베이스 개요

## 1. 분산 데이터베이스 개요

### 분산 데이터베이스란?

분산 데이터베이스(Distributed Database)란 하나의 데이터베이스를 여러개의 노드(서버, 인스턴스 등)에 나누어 저장하고 관리하는 시스템이다. 이를 통해 **확장성, 가용성, 성능 등을 높일 수 있으며** 사용목적에 따라 데이터 저장 방식(파티셔닝, 샤딩, 레플리케이션 등)을 적용하여 **데이터 처리의 부하를 분산하고 장애 상황에 대응할 수 있다.**

### 분산 데이터베이스를 사용하는 이유

단일 데이터베이스는 초기 시스템이나 소규모 데이터 처리에서는 간단하고 유지보수가 쉽다는 장점이 있지만 데이터가 많아지고 트래픽이 증가할수록 **DB 다운, 성능, 확장성, 저장 공간 등 여러 문제들이 발생한다.** 이러한 문제를 해결하기 위해 분산 데이터베이스가 필요하다.

## 1. 분산 데이터베이스 개요

### CAP 이론

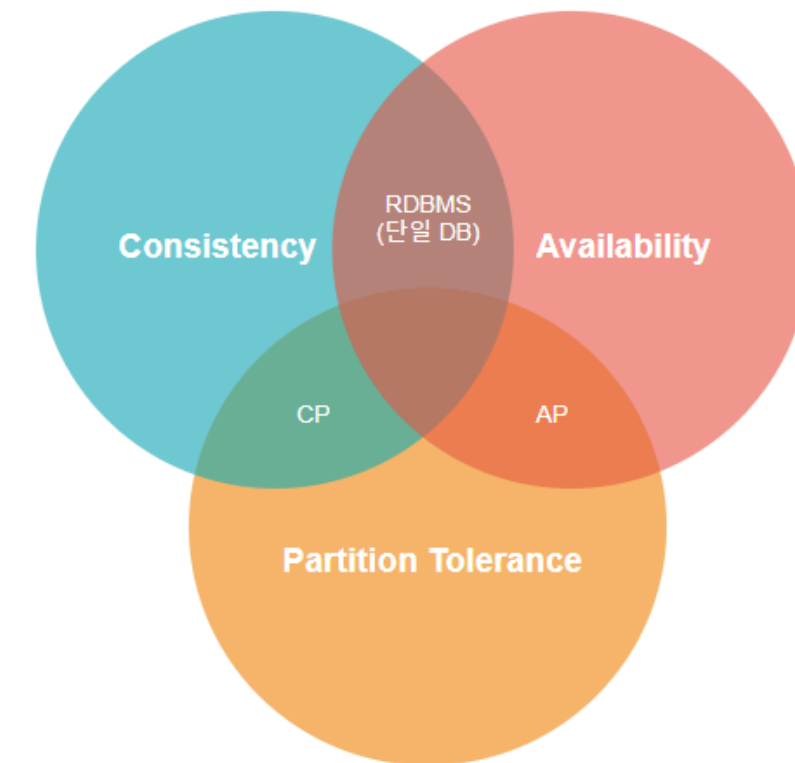
CAP 이론이란 분산 시스템에서 **일관성(Consistency)**, **가용성(Availability)**, **분할 허용성(Partition Tolerance)** 중 3가지를 동시에 만족할 수 없다는 이론이다. CAP 이론은 분산 데이터베이스의 핵심 원칙이며 뒤에 소개할 파티셔닝, 샤딩, 레플리케이션은 이 CAP 이론의 \*트레이드오프를 조정하기 위한 방법들로 사용된다.

분산 데이터베이스는 이름부터 분산인 만큼 여러 개의 노드로 구성되기 때문에 물리적으로 네트워크를 통해 데이터를 주고 받는다.

따라서 분산 환경에서는 네트워크 장애가 발생할 가능성이 항상 존재하기 때문에 **분할 허용성(Partition Tolerance)**은 반드시 허용해야 한다.

즉, CAP 이론을 기반으로 시스템 요구사항에 맞게 적절한 데이터베이스 분산 방식을 조합하여 설계할 수 있다.

\*트레이드오프 (Trade-off) : 두 가지 이상의 요소가 서로 충돌하여 하나를 얻으면 다른 하나는 어느정도 포기해야 하는 상황



## 2. 파티셔닝 (Partitioning)

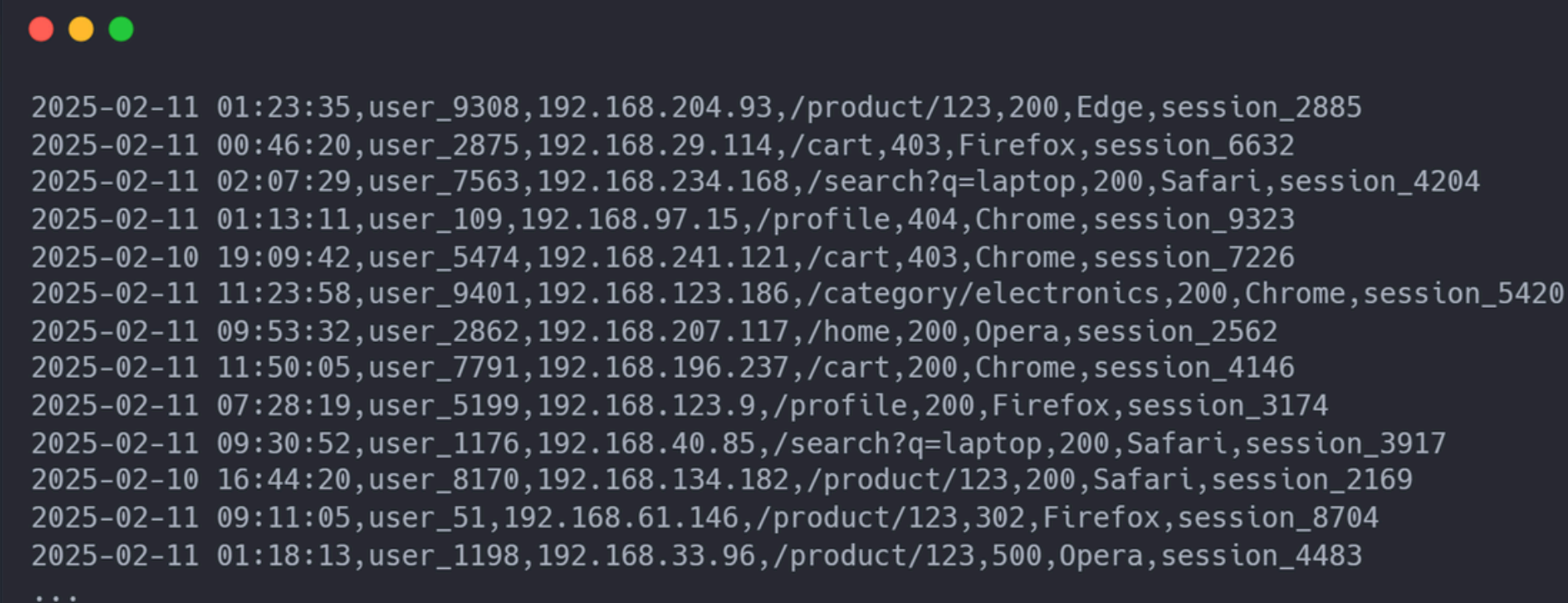
## 2-1. 파티셔닝 (Partitioning)이란?

테이블이 논리적으로 하나지만, 물리적으로 여러 개로 분리하여 관리하는 데이터베이스

특징

데이터가 많이 쌓여서 비대해진 테이블을 작은 단위의 테이블로 쪼개어 성능을 높임

## 2-1. 파티셔닝 (Partitioning)

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays a list of log entries, each on a new line. The entries follow a consistent format: a date and time, followed by a user ID, an IP address, a URL path, a number, a browser name, and a session ID. The logs span from February 10 to February 11, 2025. The last line is followed by an ellipsis (...).

```
2025-02-11 01:23:35,user_9308,192.168.204.93,/product/123,200,Edge,session_2885
2025-02-11 00:46:20,user_2875,192.168.29.114,/cart,403,Firefox,session_6632
2025-02-11 02:07:29,user_7563,192.168.234.168,/search?q=laptop,200,Safari,session_4204
2025-02-11 01:13:11,user_109,192.168.97.15,/profile,404,Chrome,session_9323
2025-02-10 19:09:42,user_5474,192.168.241.121,/cart,403,Chrome,session_7226
2025-02-11 11:23:58,user_9401,192.168.123.186,/category/electronics,200,Chrome,session_5420
2025-02-11 09:53:32,user_2862,192.168.207.117,/home,200,Opera,session_2562
2025-02-11 11:50:05,user_7791,192.168.196.237,/cart,200,Chrome,session_4146
2025-02-11 07:28:19,user_5199,192.168.123.9,/profile,200,Firefox,session_3174
2025-02-11 09:30:52,user_1176,192.168.40.85,/search?q=laptop,200,Safari,session_3917
2025-02-10 16:44:20,user_8170,192.168.134.182,/product/123,200,Safari,session_2169
2025-02-11 09:11:05,user_51,192.168.61.146,/product/123,302,Firefox,session_8704
2025-02-11 01:18:13,user_1198,192.168.33.96,/product/123,500,Opera,session_4483
...
```

사용  
이유

인덱스는 검색 성능 향상에 주로 사용하지만, 파티셔닝은 **파티션 프루닝**을 적용시켜서 INSERT, UPDATE, DELETE 트랜잭션의 성능을 높이는 데에도 도움이 됨

## 2-2. 파티셔닝 (Partitioning) 종류

Post			
	post_id	post_title	post_content
row1			
row2			
row3			

### 수직 분할 파티셔닝

많은 컬럼을 가지고 있는 테이블을  
관심사와 역할에 따라서  
작은 단위의 테이블로 나눈다.

예시. 게시글 테이블에서 용량이 큰  
content 컬럼을 분리하면  
게시글 테이블을 가볍게 만들 수 있다.



## 2-2. 파티셔닝 (Partitioning) 종류

Log

	log_id	log_status	log_detail
row1			
row2			
row3			

### 수평 분할 파티셔닝

테이블 중 하나의 컬럼을 정해서,  
**Partition key**로 활용한다.  
해당 컬럼의 값을 기준으로 파티셔닝한다.

값의 범위(Range)나  
특정 값(List, Hash)을 기준으로  
파티셔닝을 수행한다.

## 2-3. 파티셔닝을 사용하면 좋은 경우

### 장점

성능 향상

데이터 관리 효율

### 단점

일부 데이터베이스에서 기능 제한

Join 이 자주 발생하면, 악영향

파티셔닝된 테이블 중, 소수의 테이블만 사용할 경우 옵티마이저가 접근하는 테이블이 적어져서 성능이 좋아진다.

# 3. 샤딩 (Sharding)

### 3-1. 샤딩 (Sharding) 개요

대량의 데이터를 여러 개의 데이터베이스 서버에 분산 저장하는 기법

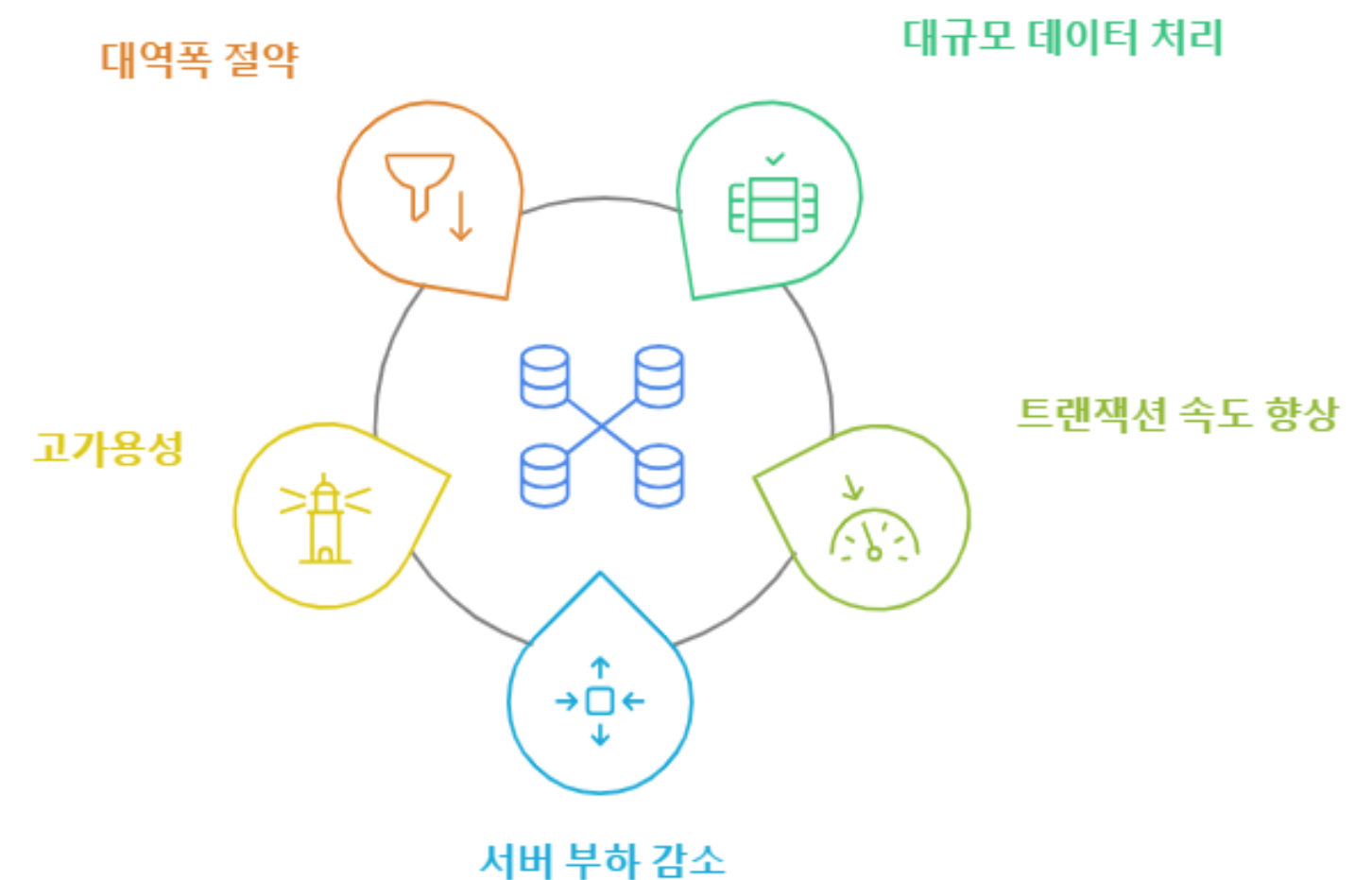
특징

한 서버에서 처리할 수 없는 대량의 데이터를 여러 샤드로 나누어 분산 처리

수억 명의 사용자가 있는 SNS(페이스북, 트위터 등)에서 사용자 데이터를 효율적으로 관리

## 3-2. 샤딩이 필요한 이유

- 대규모 데이터 처리
  - 한 서버에서 처리할 수 없는 대량의 데이터를 여러 샤드로 나누어 분산 처리
- 트랜잭션 처리 속도 향상
  - 하나의 데이터베이스에서 처리하는 대신 여러 샤드에서 동시 병렬 처리하여 성능을 극대화 가능
- 서버 부하 감소 및 확장성 향상
  - 특정 서버에 데이터가 집중되지 않도록 부하를 균등하게 분산
  - 새로운 샤드를 추가하는 방식으로 쉽게 확장 가능
- 고가용성 및 장애 격리
  - 특정 샤드에 장애가 발생해도 다른 샤드는 정상적으로 운영 가능
- 네트워크 대역폭 절약
  - 데이터를 지역 또는 사용 패턴에 따라 분할하여 네트워크 트래픽 감소



### 3-3. 샤딩 (Sharding) 종류

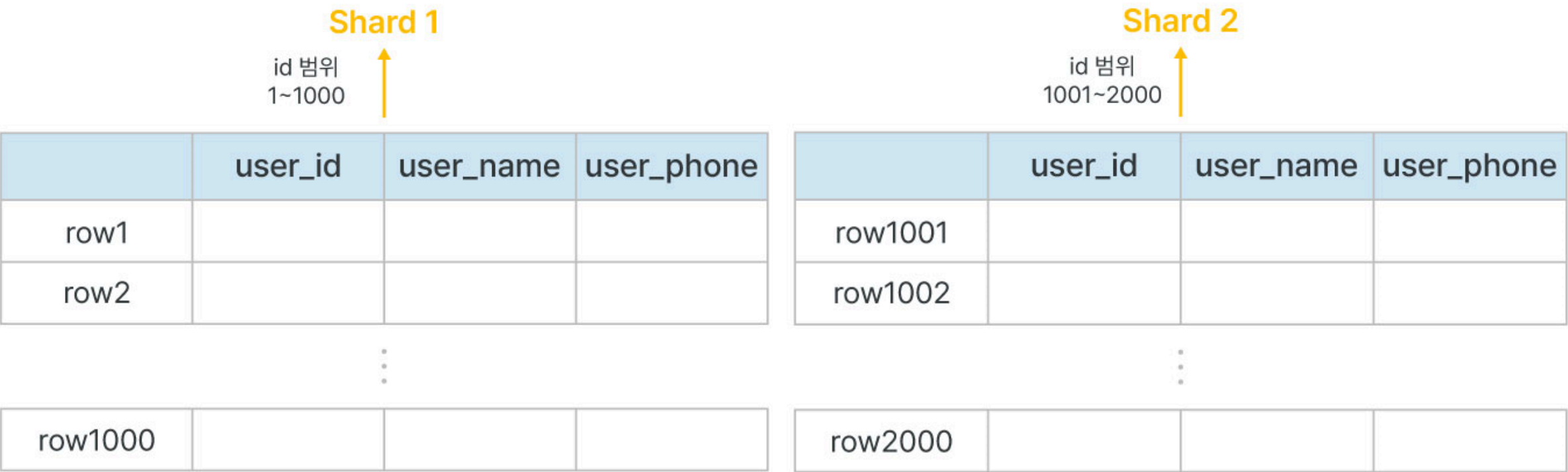
	Shard 1		Shard 2
	이름	이메일	구매 내역
row1			
row2			
row3			
			⋮
row n			

#### 수직 샤딩

테이블을 컬럼(Column) 단위로  
분할하여 서로 다른 샤드에 저장

예시)  
사용자 정보(이름, 이메일) -> Shard 1  
주문정보(구매 내역) -> Shard 2

3-3. 샤딩 (Sharding) 종류

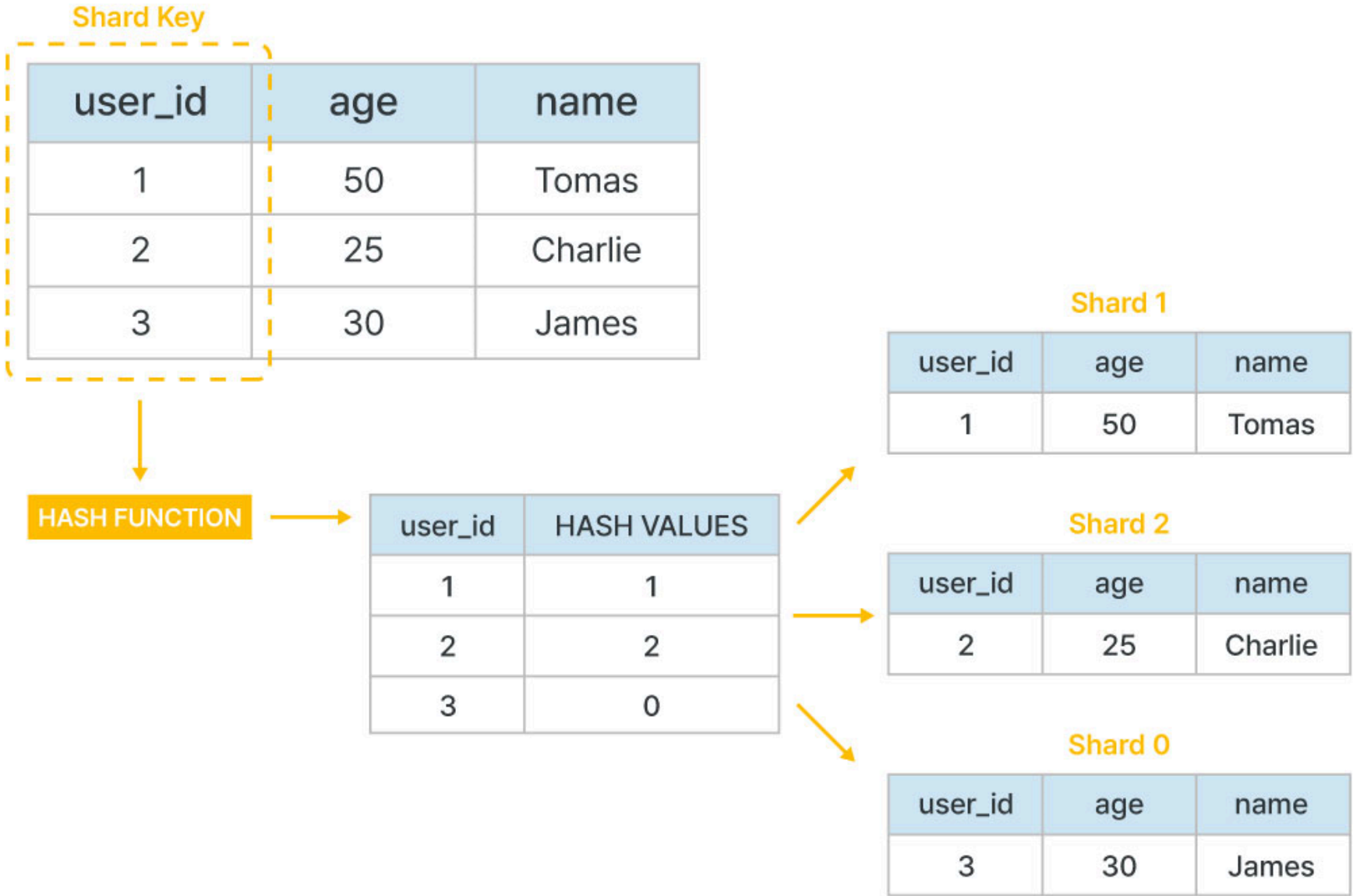


수평 샤딩

한 테이블의 데이터를 여러 샤드에  
행(Row) 단위로 분할

예시)  
user\_id 1~1000 → Shard 1  
user\_id 1001~2000 → Shard 2  
...

### 3-3. 샤딩 (Sharding) 종류



#### 해시 샤딩

수평 샤딩에 해당하는 기법 중 하나로  
특정 컬럼을 해시 함수를 적용해 데이  
터가 저장될 샤드를 결정

예시)

$\text{HASH}(\text{user\_id}) \% 3 = 0 \rightarrow \text{Shard 0}$

$\text{HASH}(\text{user\_id}) \% 3 = 1 \rightarrow \text{Shard 1}$

$\text{HASH}(\text{user\_id}) \% 3 = 2 \rightarrow \text{Shard 2}$



# 4. 레플리케이션 (Replication)

4-1. 레플리케이션 (Replication)이란?

여러 개의 DB를 권한에 따라 수직적인 구조(Master-Slave)로 구축하는 방식  
Master Node 는 쓰기 작업만 처리하며, Slave Node 는 읽기 작업만 처리

특징	데이터베이스 서버를 확장한 Clustering과 달리 서버와 스토리지 모두 확장
	여러 개의 DB를 권한에 따라 수직적인 구조(Master-Slave)로 구축하는 방식 Master : INSERT, UPDATE, DELETE Slave : SELECT

## 4-2. 레플리케이션 (Replication) 사용하는 이유



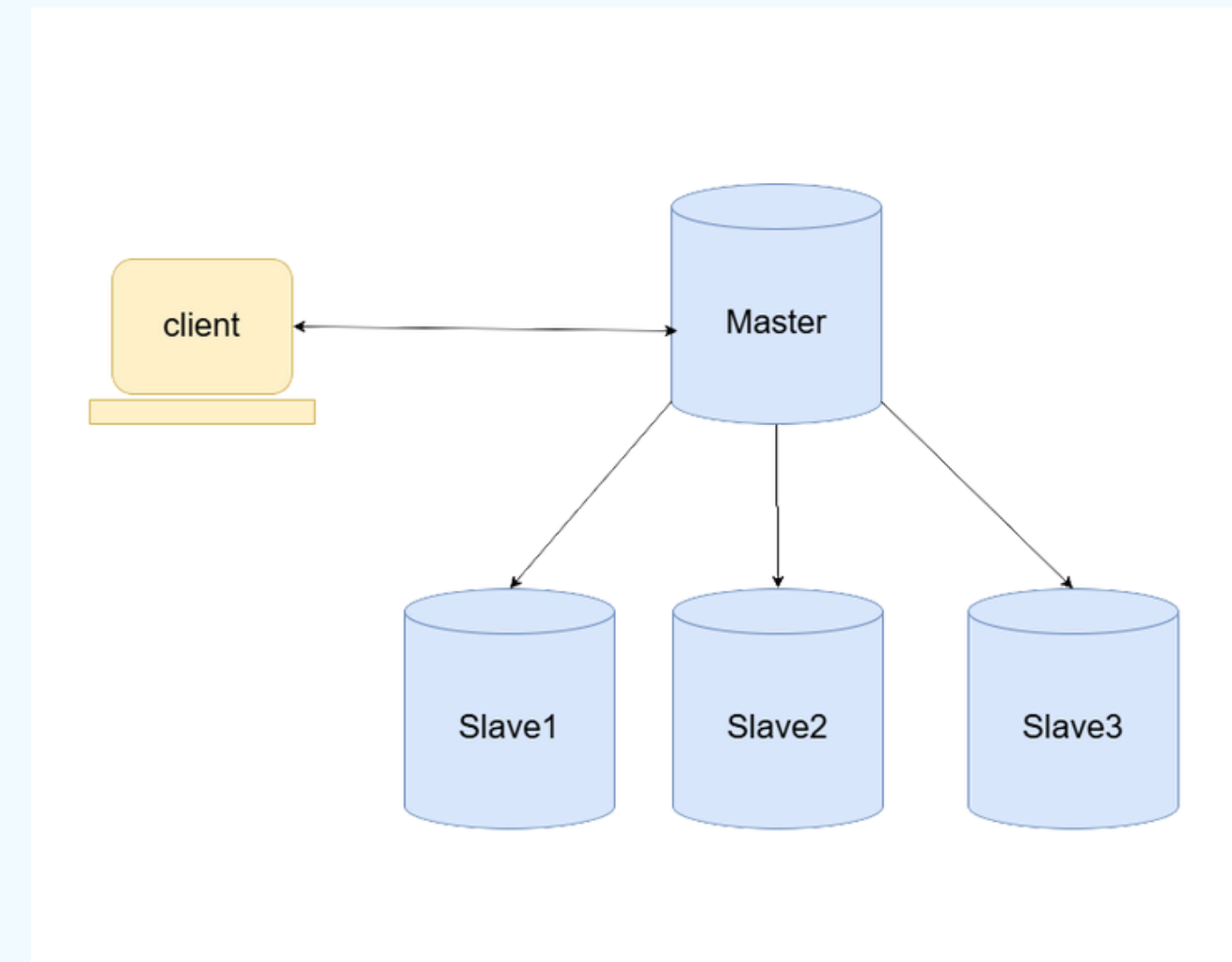
### 4-3. 레플리케이션 아키텍처 (Replication architectures)

#### Replication architectures

**마스터-슬레이브(Master-slave replication) :**

읽기 작업 분산으로 성능 향상과 마스터 부하 감소에 효과적.

마스터-슬레이브 모델에서는 한 개의 데이터베이스(**마스터**)가 주요 데이터의 원천 역할을 하고,  
**다른 데이터베이스(슬레이브)**들이 마스터로부터 데이터를 복제



## 4-4. 레플리케이션 아키텍처 (Replication architectures)

### 클러스터링 (Clustering)

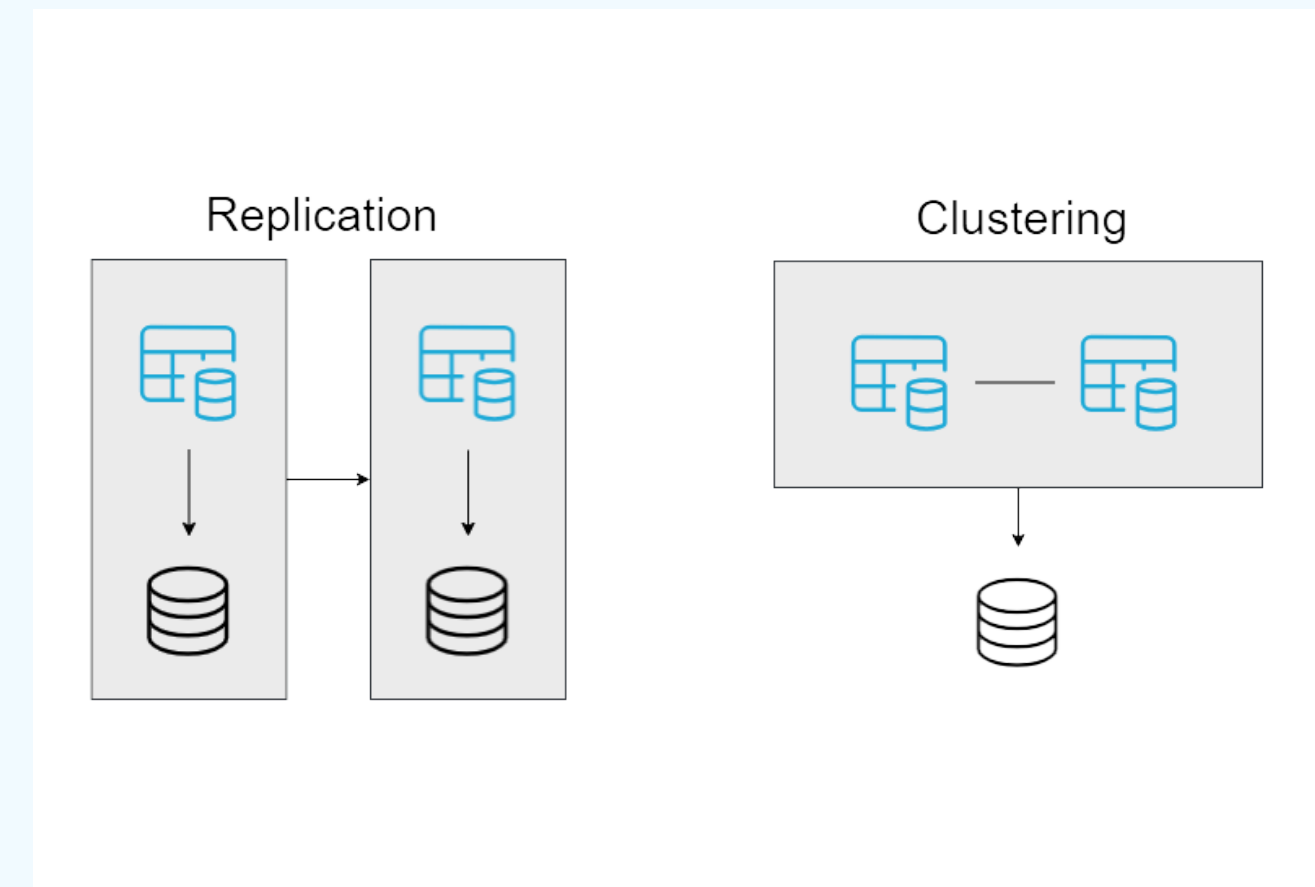
여러 개의 DB를 **수평적인 구조**로 구축하는 방식  
분산 환경을 구성하여 Single point of failure와 같은 문제를  
해결할 수 있는 Fail Over 시스템을 구축하기 위해서 사용  
**동기 방식**으로 노드들 간의 데이터를 동기화

\*single point of failure(단일 장애점,SPOF)

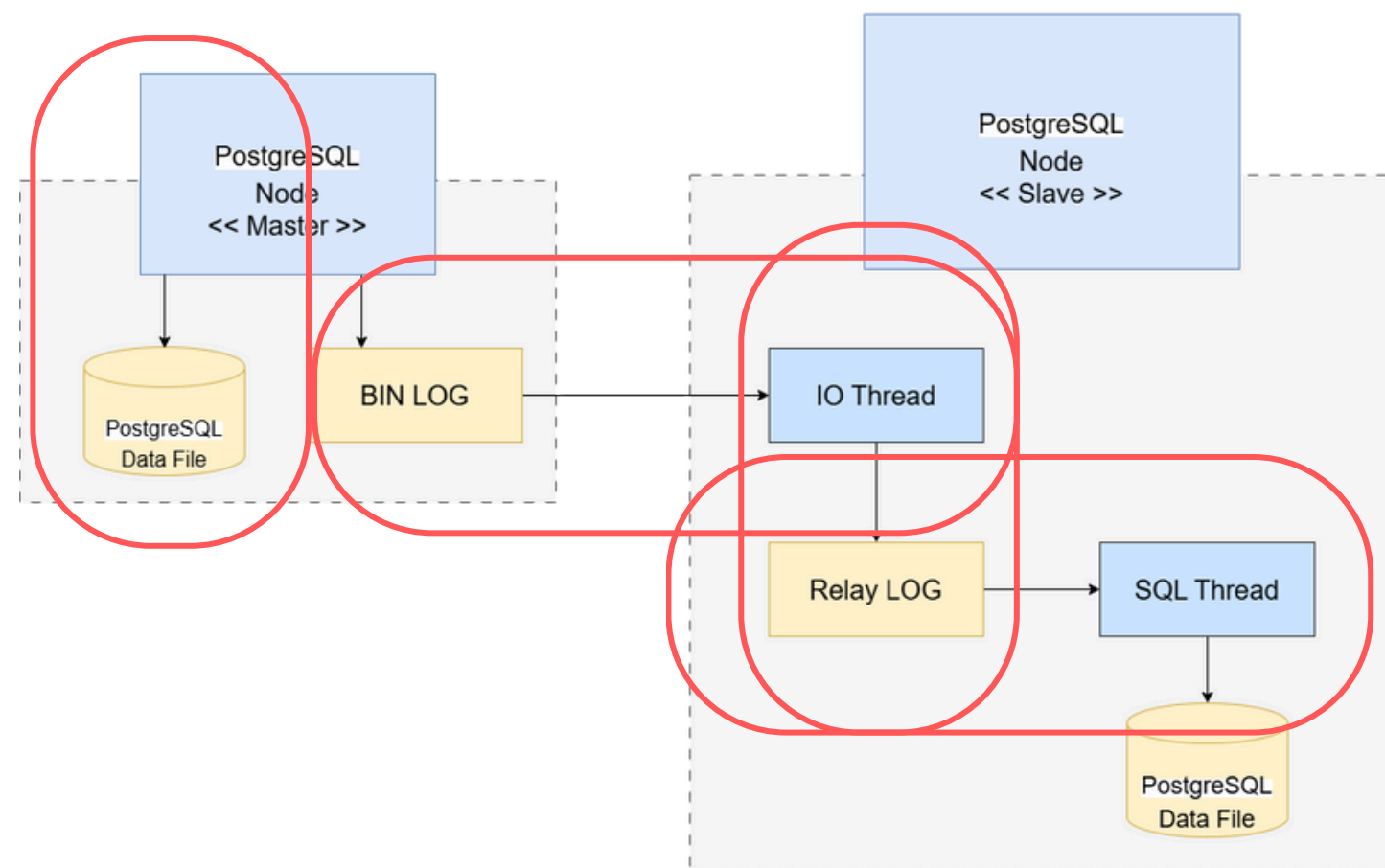
시스템 구성 요소 중에서, 동작하지 않으면 전체 시스템이 중단되는 요소가 이중화가 되어 있지 않다면 SPOF일 가능성 높음

\*Fail over

실 운영환경(컴퓨터 서버, 시스템, 네트워크) 등에서 이상이 생겼을 때, 대체 작동 또는 장애 극복(조치)을 위해  
예비 운영환경으로 자동전환되는 기능



## 4-5. 레플리케이션 (Replication) 처리방식



### Replication 처리방식

1. Master 노드에 쓰기 트랜잭션이 수행
2. Master 노드는 데이터를 저장하고 트랜잭션에 대한 로그를 파일에 기록(BIN LOG)
3. Slave 노드의 IO Thread는 Master 노드의 로그 파일(BIN LOG)를 파일(Replay Log)에 복사
4. Slave 노드의 SQL Thread는 파일(Replay Log)를 한 줄씩 읽으며 데이터를 저장

## 4-6. 레플리케이션 (Replication) 장점



### 장점

#### 성능 향상

DB 요청의 60~80% 정도가 **읽기** 작업이기 때문에 Replication만으로도 충분히 성능을 높일 수 있다.

#### 지연 시간 감소

**비동기** 방식으로 운영되어 지연 시간이 거의 없다.

#### 장애 허용성

일부 노드에 장애가 발생하거나 사용할 수 없게 되더라도 데이터나 서비스에 **접근을 보장**한다.

# 5. 구현



## 5-1. 구현 - 파티셔닝

🔥 **구현 목표** 라이프 사이클의 주기가 짧은 로그 데이터를 조회할 때 I/O 부하를 줄이기 위해 파티셔닝 진행

```
-- 테이블 하나 생성하기
create table user_logs_partition (
  time_stamp varchar(50),
  user_id varchar(50),
  ip_address varchar(50),
  page_url varchar(50),
  http_response int4,
  browser varchar(50),
  session_id varchar(50)
) PARTITION BY RANGE (time_stamp);

CREATE TABLE user_logs_20250210 PARTITION OF user_logs_partition
FOR VALUES FROM ('2025-02-10') TO ('2025-02-11');

CREATE TABLE user_logs_20250211 PARTITION OF user_logs_partition
FOR VALUES FROM ('2025-02-11') TO ('2025-02-12');
```

### 원본 테이블

Startup-Cos	0.00
Total-Cost	20195.00
Plan-Rows	2083
Plan-Width	78

### 파티셔닝 테이블

Startup-Cost	0.00
Total-Cost	10090.72
Plan-Rows	1041
Plan-Width	78

데이터가 줄어드는 만큼 Cost가 줄어드는 것을 볼 수 있다.

## 5-2. 구현 - 샤딩

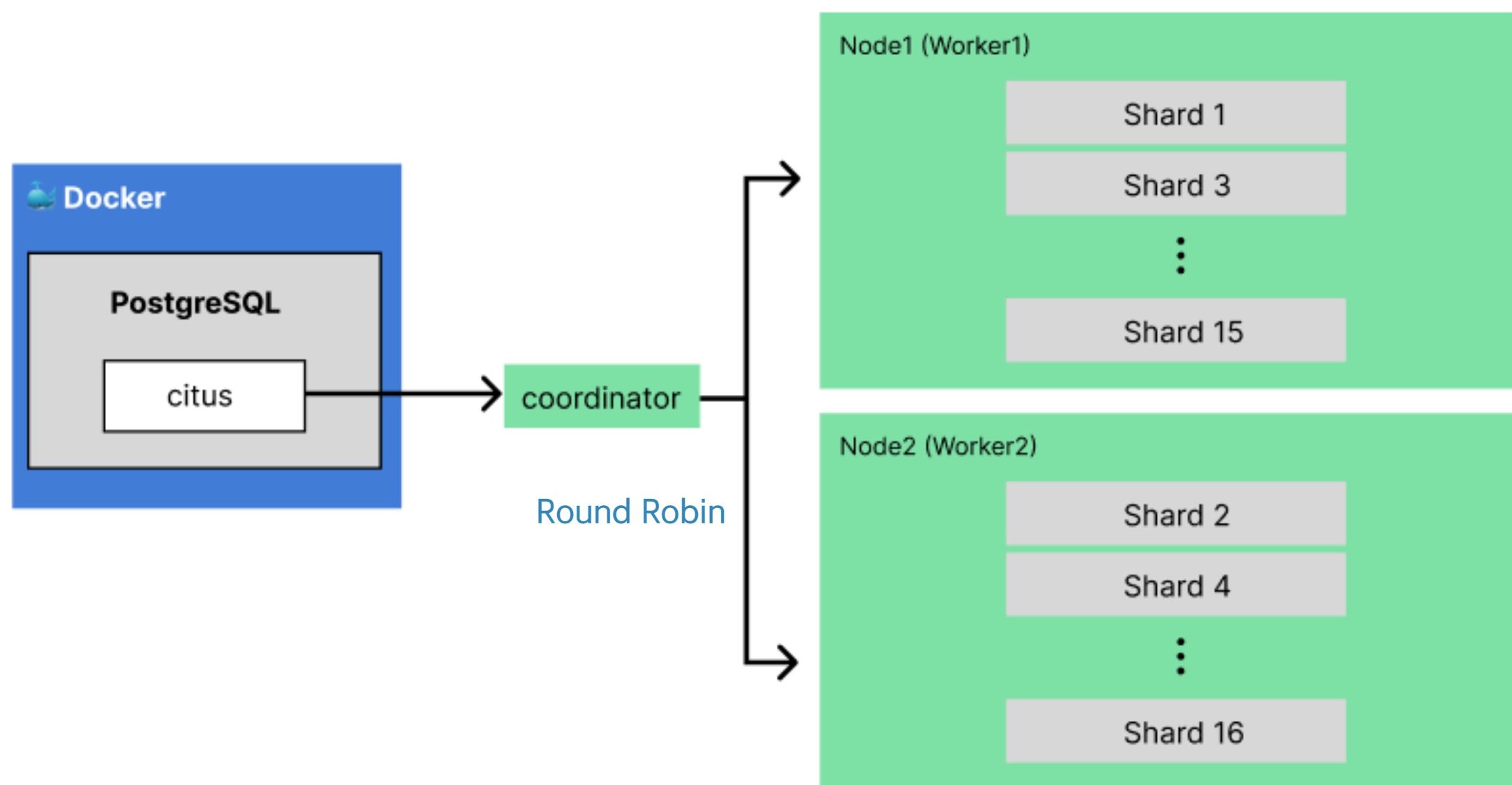
### 구현 목표

PostgreSQL과 Citus 기반 샤딩 환경에서 데이터 조회 성능을 비교하여, 샤딩이 특정 쿼리 성능에 미치는 영향을 분석한다. 이를 위해 샤딩된 테이블과 일반 테이블 간의 조건을 동등하게 하고 성능을 공정하게 비교한다.

### 테스트 환경

- Container 환경으로 구성 - Docker
- DB 서버 - PostgreSQL & Citus 클러스터
- Citus 설정에서 노드(Worker)는 2개로 설정
- Sharding 할 Shard는 32개로 설정
- 해시 샤딩 구현

## 🔥 아키텍처



### 👁 👁 구현 방향

- user\_data 테이블을 만들고 100만개의 더미 데이터를 INSERT
- user\_data 테이블에 [user\_id]값을 샤딩키로 샤딩을 적용 시켜 user\_data\_sharding 테이블을 생성
- 각 테이블에서 SELECT \* FROM TABLE WHERE user\_id = "; 을 실행 시켜 Cost, Execution Time 값을 비교

## 1. Citus extension을 활성화 하고 worker 노드 2개 추가

```
1 CREATE EXTENSION IF NOT EXISTS citus;  
2  
3  
4 SELECT * FROM master_add_node('worker1', 5432);  
5 SELECT * FROM master_add_node('worker2', 5432);
```

## 2. 정상적으로 노드(worker)가 추가 됐는지 확인

```
SELECT * FROM master_get_active_nodes();
```

	node_name	node_port
1	worker2	5,432
2	worker1	5,432

3. 테이블명, 샤딩키로 테이블 샤딩 후 정상적으로 샤딩이 됐는지 확인

```
SELECT create_distributed_table('user_data_sharding', 'user_id');  
  
SELECT shardid, shard_name, nodename FROM citus_shards;
```

	123 shardid	A-Z shard_name	A-Z nodename	123 count
1	102,008	user_data_sharding_102008	worker1	1
2	102,009	user_data_sharding_102009	worker2	32
3	102,010	user_data_sharding_102010	worker1	
4	102,011	user_data_sharding_102011	worker2	
5	102,012	user_data_sharding_102012	worker1	
6	102,013	user_data_sharding_102013	worker2	
7	102,014	user_data_sharding_102014	worker1	

> 100만개의 데이터가 32개의 샤드에 해싱을 통해 동등하게 분배 (약 31250개씩)

SELECT * FROM run_command on sha Enter a SQL expression to filter results (use Ctrl+Space)			
	123 shardid	success	A-Z result
1	102,008	[v]	31429

4. 일반 테이블, 샤딩된 테이블에서 각각 SELECT 쿼리 실행

[Query] EXPLAIN ANALYZE SELECT \* FROM TABLE WHERE user\_id = 'user\_123';

원본 테이블

```

1  Gather (cost=1000.00..16993.43 rows=1 width=49) (actual time=0.164..33.919 rows=1 loops=1)
2    Workers Planned: 2
3    Workers Launched: 2
4    -> Parallel Seq Scan on user_data (cost=0.00..15993.33 rows=1 width=49) (actual time=18.10
5          Filter: ((user_id)::text = 'izMUmj9Z'::text)
6          Rows Removed by Filter: 333333
7    Planning Time: 0.053 ms
8    Execution Time: 33.945 ms

```

-> Execution Time : 33.945ms

샤딩 테이블

```

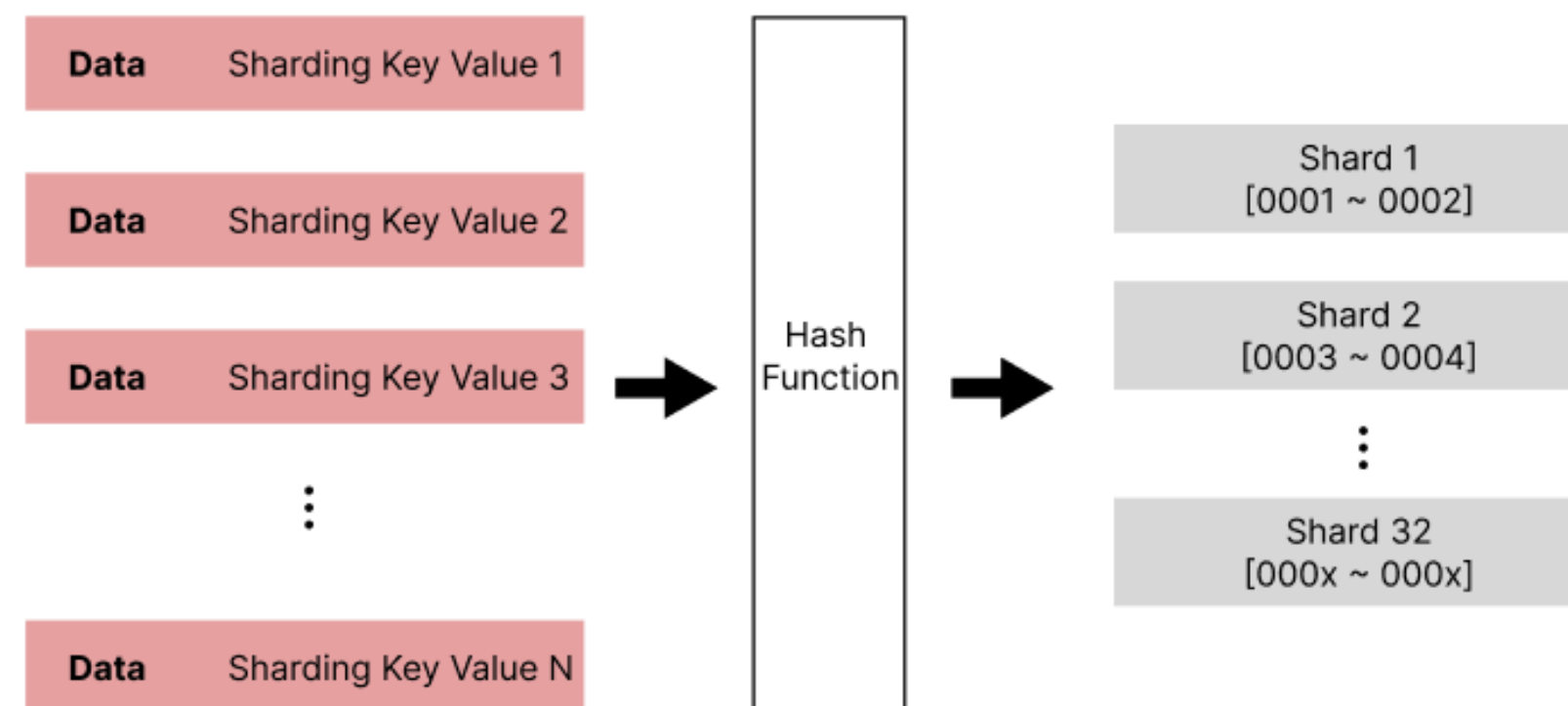
1  Custom Scan (Citius Adaptive) (cost=0.00..0.00 rows=0 width=0) (actual time=2.994..2.995 rows=1 loops=1)
2    Task Count: 1
3    Tuple data received from nodes: 39 bytes
4    Tasks Shown: All
5    -> Task
6          Tuple data received from node: 39 bytes
7          Node: host=worker2 port=5432 dbname=citusb
8    -> Seq Scan on user_data_sharding_102033 user_data_sharding (cost=0.00..733.10 rows=1 width=48) (actual time=0
9          Filter: ((user_id)::text = 'izMUmj9Z'::text)
10          Rows Removed by Filter: 31127
11          Planning Time: 0.042 ms
12          Execution Time: 1.752 ms
13    Planning Time: 0.072 ms
14    Execution Time: 3.008 ms

```

-> Execution Time : 3.008ms

### 🔍 탐구 | 해시 샤딩에서 각 샤드는 자신이 어떤 데이터를 갖고 있는지 어떻게 알까?

> 해시 샤딩은 샤딩 키 값을 해시 함수로 변환하여 나온 해시 값의 최소값, 최대값을 구하여 균등하게 범위를 지정해서 샤드에 부여하고 데이터를 범위에 해당하는 샤드에 할당하는 방식이다. 따라서 데이터는 해당 해시 값이 속하는 샤드에 자동으로 저장되고 샤드는 자신이 어떤 키 값에 대한 데이터를 갖고 있는지 알고있다.





# 6. 비교 분석

6. 비교분석 - 샤딩(Sharding) vs 파티셔닝(Partioning)

	샤딩	파티셔닝
방식	여러 개의 DB로 분할	하나의 DB 내에서 분할
확장성	높은 확장성 제공	상대적으로 낮음
운영 난이도	높은 운영 복잡도	상대적으로 쉬운 관리
적합 상황	주로 유니크한 값으로 데이터 분산하는 경우	데이터를 특정 그룹(카테고리)이나 시간 범위로 나눠야 하는 경우
사용 사례 예시	수억 명의 사용자가 존재하며, 각 사용자의 user_id 기반으로 데이터를 저장	최근 1개월 내 데이터를 주로 조회하지만, 오래된 데이터도 유지해야 함

6. 비교분석 - 샤딩(Sharding) vs 레플리케이션(Replication)

	샤딩	레플리케이션
정의	데이터를 분할하여 여러 노드에 분산 저장	동일한 데이터를 여러 노드에 복제하여 저장
주요 목적	데이터 분산을 통해 시스템 확장성(Scalability) 확보	데이터의 가용성(Availability)과 장애 복구(Recovery) 보장
데이터 분포 방식	데이터를 특정 기준(예: 키 값)에 따라 각 샤드에 나누어 저장	모든 복제본(replica)에 동일한 데이터 저장
가용성	샤딩만으로는 가용성 보장 어려움 (샤드 손실 시 해당 데이터 접근 불가)	복제를 통해 높은 가용성 보장 (노드 장애 시 다른 복제본 사용 가능)
사용 사례 예시	트래픽이 매우 높은 대규모 애플리케이션 (예: 소셜 네트워크, 로그 데이터 저장)	고가용성이 중요한 서비스 (예: 금융 시스템, 데이터베이스 백업)