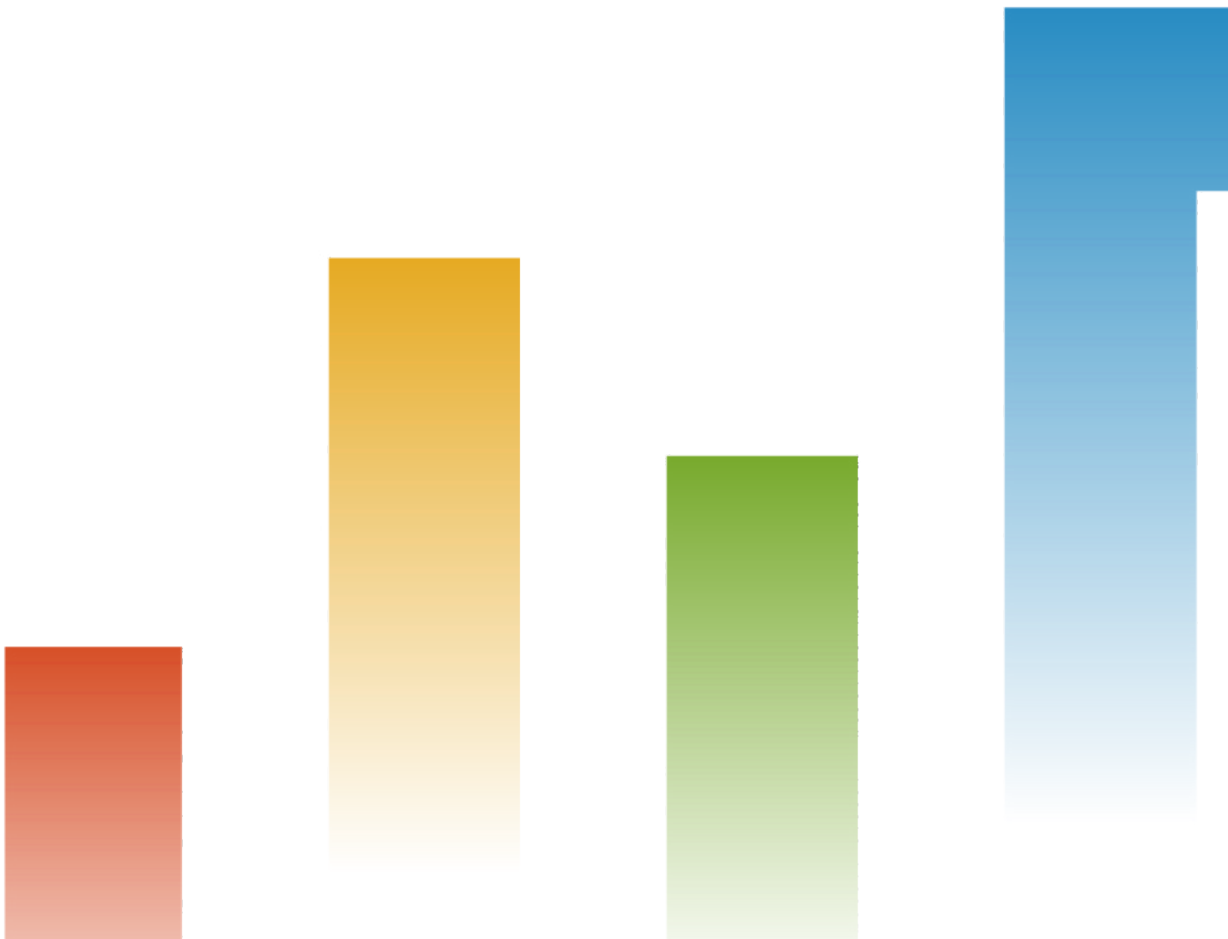
WhaTap

OpenMetrics

release date. 2026. 09. 10.



OpenMetrics 탐색기

이 문서는 **OpenMetrics 탐색기** 메뉴에서 PromQL 쿼리로 메트릭을 조회하고 차트·테이블로 시각화하는 방법을 안내합니다.
처음 PromQL을 쓰는 사용자도 단계적으로 따라 할 수 있도록 기본 → 함수 → 실전 예제 순서로 정리했습니다.

주요 기능

- PromQL 쿼리를 통한 메트릭 조회
- Graph/Table 뷰로 데이터 시각화
- 시간 범위 선택 시 자동으로 PromQL에 시간 입력
- 실시간 메트릭 모니터링

화면 구성

- 상단: 시간 범위 선택기, 쿼리 입력창
- 중앙: Graph/Table/Stacked Bar 뷰 전환 버튼
- 하단: 조회 결과 표시 (그래프 또는 테이블)

왜 PromQL이 필요한가요?

OpenMetrics는 Prometheus 메트릭 포맷을 기반으로 다양한 시스템과 애플리케이션의 시계열 지표를 수집합니다.
Prometheus는 오픈소스 모니터링 도구로, CPU 사용률, 메모리 사용량, HTTP 요청 수, 에러율 등과 같은 수치 기반의 시간 흐름 데이터를 메트릭 지표로 저장합니다.

이러한 지표들은 대부분 수초 단위로 지속적으로 수집되며, 시간에 따라 끊임없이 변화하는 '시계열 데이터(time-series data)'의 형태를 가집니다.

하지만 단순히 데이터를 수집하는 것만으로 "지금 서비스가 정상인가?", "에러 비율이 평소보다 높아졌는가?", "어떤 API가 느려졌는가?" 같은 의미 있는 분석이 어렵습니다.

그래서 **OpenMetrics 탐색기**에서 수집된 메트릭을 분석할 수 있는 쿼리 언어가 필요하고, 그 역할을 하는 것이 바로

PromQL(Prometheus Query Language)입니다.

PromQL을 사용하면 다음과 같은 분석이 가능합니다.

- **실시간 모니터링**: 현재 시스템 상태를 즉시 확인 (현재 CPU 사용률, 활성 사용자 수 등)
- **추세 분석**: 시간에 따른 변화 패턴 파악 (최근 1시간 동안 초당 요청 수 증가율)
- **비교 분석**: 여러 서버나 서비스 간의 지표 비교 (서버별 에러율, 지역별 응답 시간)
- **집계 및 계산**: 복잡한 수학 연산과 통계 계산 (전체 에러율, 평균 응답 시간, 상위 10개 느린 API)
- **알림 조건 설정**: 특정 임계값 초과 시 자동 알림 (에러율 5% 이상, 메모리 사용률 90% 이상)

PromQL 사용자 가이드

쿼리 실행 방식

표 | PromQL 쿼리 실행 방식

쿼리	설명
Instant Query (즉시 쿼리)	특정 시점의 데이터를 조회함. Table 탭에서 사용됨
Range Query (범위 쿼리)	시작 시간과 종료 시간 사이의 데이터를 일정한 간격으로 조회함

데이터 타입

PromQL 표현식은 4가지 타입으로 평가됩니다.

표 | PromQL 데이터 타입

타입	설명
Instant Vector	각 시계열에서 단일 샘플을 포함하는 시계열 집합 (모두 동일한 타임스탬프)
Range Vector	각 시계열에서 시간 범위의 데이터 포인트를 포함하는 시계열 집합

타입	설명
Scalar	단순한 숫자 값
String	단순한 문자열 값

기본 사용법

1. 메트릭 선택(Instant Vector Selectors)

기본 메트릭 조회

```
nginx_http_response_count_total
```

레이블 필터링

```
nginx_http_response_count_total{status="200"}
```

여러 레이블 조건

```
nginx_http_response_count_total{method="GET", status="200"}
```

2. 레이블 매칭 연산자

표 | 레이블 매칭 연산자

연산자	설명
=	레이블 값이 정확히 일치
!=	레이블 값이 일치하지 않음
=~	레이블 값이 정규표현식과 일치
!~	레이블 값이 정규표현식과 일치하지 않음

정규표현식 예제

```
# staging, testing, development 환경만 조회
nginx_http_response_count_total{environment=~"staging|testing|development"}

# GET 메소드를 제외한 모든 메소드
nginx_http_response_count_total{method!="GET"}

# rep로 시작하는 replica 조회
nginx_http_response_count_total{replica=~"rep.*"}
```

3. 시간 범위 선택(Range Vector Selectors)

현재 시점부터 과거 특정 기간의 데이터를 선택합니다.

```
# 최근 5분간의 데이터
nginx_http_response_count_total[5m]

# 최근 1시간의 데이터
nginx_http_response_count_total{status="200"}[1h]
```

4. 시간 오프셋(Offset Modifier)

과거 특정 시점의 데이터를 조회합니다.

```
# 5분 전의 값
nginx_http_response_count_total offset 5m

# 1주일 전의 5분간 비율
rate(nginx_http_response_count_total[5m] offset 1w)

# 미래 비교 (음수 offset)
rate(nginx_http_response_count_total[5m] offset -1w)
```

실용 예제

예제 1: 상태 코드별 요청 수 조회

```
nginx_http_response_count_total{instance="http://192.168.100.122:4040/metrics"}
```

예제 2: 특정 상태 코드만 조회

```
# 200 응답만
nginx_http_response_count_total{status="200"}

# 4xx 에러만(정규표현식)
nginx_http_response_count_total{status=~"4.."}

# 5xx 에러만
nginx_http_response_count_total{status=~"5.."}
```

예제 3. 메소드와 상태 코드 조합

```
# GET 요청 중 200 응답
nginx_http_response_count_total{method="GET", status="200"}

# POST, PUT, DELETE 요청 중 404 응답
nginx_http_response_count_total{method=~"POST|PUT|DELETE", status="404"}
```

예제 4. 메트릭 이름으로 검색

```
# "job:"로 시작하는 모든 메트릭
{__name__=~"job:.*"}

# nginx로 시작하는 모든 메트릭
{__name__=~"nginx.*"}
```

예제 5. 최근 시간 범위 데이터

```
# 최근 5분간의 nginx 응답
nginx_http_response_count_total{status="200"}[5m]

# 최근 1시간의 404 에러
nginx_http_response_count_total{status="404"}[1h]
```

예제 6. 시간대 비교

```
# 현재 값과 1시간 전 값 비교
nginx_http_response_count_total{status="200"}
nginx_http_response_count_total{status="200"} offset 1h

# 어제 같은 시간의 데이터
nginx_http_response_count_total offset 1d
```

집계 함수

집계 함수(Aggregation Functions)는 여러 시계열 데이터를 하나로 결합하여 계산합니다.

sum()

모든 시계열의 값을 더합니다.

```
# 모든 인스턴스의 요청 수 합계
sum(nginx_http_response_count_total)

# 상태 코드별로 그룹화하여 합계
sum by (status) (nginx_http_response_count_total)

# 인스턴스별로 그룹화하여 합계
sum by (instance) (nginx_http_response_count_total)
```

avg()

모든 시계열의 **평균**값을 계산합니다.

```
# 모든 인스턴스의 평균 응답 수
avg(nginx_http_response_count_total)

# 상태 코드별 평균
avg by (status) (nginx_http_response_count_total)
```

count()

시계열의 **개수**를 셉니다.

```
# 전체 시계열 개수
count(nginx_http_response_count_total)

# 상태 코드별 시계열 개수
count by (status) (nginx_http_response_count_total)
```

max() / min()

시계열의 **최대값과 최소값**을 계산합니다.

```
# 최대값
max(nginx_http_response_count_total)

# 최소값
min(nginx_http_response_count_total)

# 상태 코드별 최대값
max by (status) (nginx_http_response_count_total)
```

topk() / bottomk()

지정한 수(K)만큼 **상위** 또는 **하위** 시계열을 반환합니다.

```
# 상위 5개 시계열
topk(5, nginx_http_response_count_total)

# 하위 3개 시계열
bottomk(3, nginx_http_response_count_total)

# 상태 코드별 상위 3개
topk(3, sum by (status) (nginx_http_response_count_total))
```

시계열 함수

rate()

가장 많이 사용되는 함수로, Counter 메트릭의 **초당 평균 증가율**을 계산합니다.

```
# 최근 5분간 초당 평균 요청 수
rate(nginx_http_response_count_total[5m])

# 상태 코드 200의 초당 평균 요청 수
rate(nginx_http_response_count_total{status="200"}[5m])

# 모든 인스턴스의 초당 총 요청 수
sum(rate(nginx_http_response_count_total[5m]))
```

> **주의사항**
>
> - Counter 메트릭에만 사용
> - 시간 범위는 스크랩 간격의 최소 2배 이상 권장 (예: 스크랩 간격 30초 → [1m] 이상)
> - 카운터 리셋(서버 재시작 등)을 자동으로 처리

increase()

지정한 시간 범위 동안의 **총 증가량**을 계산합니다.

```
# 최근 5분간 총 요청 수
increase(nginx_http_response_count_total[5m])

# 최근 1시간 동안의 에러 수
increase(nginx_http_response_count_total{status=~"5.."}[1h])
```

rate()와 increase()의 관계

`increase(v[5m]) = rate(v[5m]) × 300초`

- rate(): 초당 비율
- increase(): 절대 증가량

irate()

마지막 두 데이터 포인트를 기반으로 **초당 순간 증가율**을 계산합니다.

```
# 즉시 증가율
irate(nginx_http_response_count_total[5m])
```

irate() vs rate()

- irate(): 빠르게 변하는 카운터에 적합, 짧은 스파이크 감지
- rate(): 알림 및 느리게 변하는 카운터에 적합, 평균값 제공

시간 범위 함수

특정 시간 범위 내에서 값을 집계합니다.

avg_over_time()

시간 범위 평균 함수입니다.

```
# 최근 10분간 평균값
avg_over_time(nginx_http_response_count_total[10m])
```

max_over_time() / min_over_time()

지정한 시간의 **최대/최소값**을 계산하는 시간 범위 함수입니다.

```
# 최근 1시간 동안 최대값
max_over_time(nginx_http_response_count_total[1h])

# 최근 1시간 동안 최소값
min_over_time(nginx_http_response_count_total[1h])
```

sum_over_time()

지정한 기간의 값을 더한 **시간 범위 합계** 함수입니다.

```
# 최근 5분간 모든 값의 합
sum_over_time(nginx_http_response_count_total[5m])
```

중요 규칙: Rate then Sum

이 규칙은 Counter 메트릭에 rate(), increase(), irate() 같은 시계열 함수를 사용할 때 모든 집계 함수(sum, avg, max, min, count 등)에 적용됩니다.

> ⚠ **항상 rate() 먼저, 그 다음 sum()**
>
> Counter 메트릭을 집계할 때 반드시 이 순서를 지켜야 합니다.

🔵 올바른 방법

```
# rate()를 먼저 적용한 후 sum()
sum(rate(nginx_http_response_count_total[5m]))
```

❌ 잘못된 방법

```
# sum()을 먼저 하면 카운터 리셋 시 문제 발생
rate(sum(nginx_http_response_count_total[5m]))
```

Counter 메트릭을 sum()으로 먼저 합치면, 개별 서버의 카운터 리셋(재시작)을 감지할 수 없게 됩니다. 서버 한 대가 재시작되면 전체 합계가 갑자기 감소하고, rate()는 이를 음수 증가율이나 비정상적인 스파이크로 잘못 계산합니다. 반면 rate()를 먼저 적용하면 각 서버별로 카운터 리셋을 정확히 감지하고 처리할 수 있습니다.

실전 예제

예제 1. 초당 요청 수 (RPS) 계산

```
# 전체 RPS
sum(rate(nginx_http_response_count_total[5m]))

# 상태 코드별 RPS
sum by (status) (rate(nginx_http_response_count_total[5m]))

# 메소드별 RPS
sum by (method) (rate(nginx_http_response_count_total[5m]))
```

전체 Requests Per Second(RPS) PromQL에 대한 설명

sum(rate(nginx_http_response_count_total[5m]))이 왜 RPS일까?

1. nginx_http_response_count_total은??
   - Counter 메트릭: 서버 시작 이후 누적 응답 수
   - 예: 1000 → 1050 → 1120 → 1180 → 1250...
2. rate(...[5m])은 무엇일까?
   - 최근 5분간 데이터를 보고 초당 평균 증가율 계산
   - 예시: 5분 전: 1000 현재: 1300 차이: 300개 (5분 = 300초) rate = 300 / 300초 = 1.0 requests/second

   → rate()의 결과 = 초당 요청 수 (RPS)
3. sum()은 왜 필요한가?
   - 여러 서버/레이블의 rate 값을 모두 더함
   - 예: server1, status=200 → 10 req/s server1, status=404 → 2 req/s server2, status=200 → 15 req/s server2, status=404 → 3 req/s 합계 = 30 req/s

결론: sum(rate(nginx_http_response_count_total[5m])) = 모든 서버의 초당 총 요청 수 (RPS)

참고. 5분간 총 요청 수를 원하면 increase() 사용 sum(increase(nginx_http_response_count_total[5m])) = 5분간 총 요청 수

예제 2. 에러율 계산

```
# 전체 요청 대비 5xx 에러 비율 (%)
sum(rate(nginx_http_response_count_total{status=~"5.."}[5m])) / sum(rate(nginx_http_response_count_total[5m])) * 100

# 전체 요청 대비 4xx 에러 비율
sum(rate(nginx_http_response_count_total{status=~"4.."}[5m])) / sum(rate(nginx_http_response_count_total[5m]))
```

예제 3. 상위 N개 인스턴스 찾기

```
# 요청이 가장 많은 상위 5개 인스턴스
topk(5, sum by (instance) (rate(nginx_http_response_count_total[5m])))
```

```
# 에러가 가장 많은 상위 3개 인스턴스
topk(3, sum by (instance) (rate(nginx_http_response_count_total{status=~"5.."}[5m])))
```

예제 4. 시간대별 비교

```
# 현재 RPS
sum(rate(nginx_http_response_count_total[5m]))

# 1시간 전 RPS
sum(rate(nginx_http_response_count_total[5m] offset 1h))

# 어제 같은 시간 RPS
sum(rate(nginx_http_response_count_total[5m] offset 1d))
```

예제 5. 평균 응답 수

```
# 인스턴스별 평균 응답 수
avg by (instance) (rate(nginx_http_response_count_total[5m]))

# 상태 코드별 평균
avg by (status) (rate(nginx_http_response_count_total[5m]))
```

수학 연산자

산술 연산

```
# 더하기
sum(rate(nginx_http_response_count_total{status="200"}[5m]))
+
sum(rate(nginx_http_response_count_total{status="201"}[5m]))

# 빼기
sum(rate(nginx_http_response_count_total[5m])) - sum(rate(nginx_http_response_count_total{status=~"5.."}[5m]))

# 곱하기 (바이트를 MB로 변환)
nginx_http_response_size_bytes / 1024 / 1024
```

```
# 나누기 (비율 계산)
sum(rate(nginx_http_response_count_total{status=~"5.."}[5m])) / sum(rate(nginx_http_response_count_total[5m]))
```

비교 연산

```
# 100보다 큰 값만
nginx_http_response_count_total > 100

# 특정 값과 같은 경우
nginx_http_response_count_total == 200

# 범위 지정
nginx_http_response_count_total > 100 and nginx_http_response_count_total < 1000
```

실전 팁

1. 스크랩(수집 주기)별 시간 범위 선택

표 | 스크랩 간격별 권장 시간 범위

스크랩 간격	최소 시간 범위	권장 시간 범위	이유
15초	[30s]	[1m] 이상	2~4개 데이터 포인트 확보
30초	[1m]	[2m] 이상	2~4개 데이터 포인트 확보
1분	[2m]	[5m] 이상	2~4개 데이터 포인트 확보

2. 그룹화 레이블 선택

```
# 너무 세분화 (너무 많은 시계열)
sum by (instance, method, status, path) (rate(nginx_http_response_count_total[5m]))

# 적절한 그룹화
sum by (status) (rate(nginx_http_response_count_total[5m]))
```

3. 알림 규칙 작성 시

```
# 5분간 에러율이 5% 이상인 경우
sum(rate(nginx_http_response_count_total{status=~"5.."}[5m])) / sum(rate(nginx_http_response_count_total[5m]))
> 0.05
```

4. 함수 조합 순서

```
# 1. rate() 먼저
# 2. 집계 함수 (sum, avg 등)
# 3. 수학 연산

# 올바른 순서
sum(rate(nginx_http_response_count_total[5m])) * 100
```

기타 함수

abs()

값의 **절대값**을 반환하는 함수입니다.

```
abs(rate(nginx_http_response_count_total[5m]) - 100)
```

round()

값을 가장 가까운 정수 또는 지정한 단위로 **반올림**합니다.

```
# 소수점 반올림
round(sum(rate(nginx_http_response_count_total[5m])))

# 10 단위로 반올림
round(sum(rate(nginx_http_response_count_total[5m])), 10)
```

clamp_max() / clamp_min()

값이 지정한 **최대값이나 최소값을 넘지 않도록 제한**합니다.

```
# 최대값 1000으로 제한
clamp_max(nginx_http_response_count_total, 1000)

# 최소값 0으로 제한
clamp_min(nginx_http_response_count_total, 0)
```

주의사항

빈 레이블 값 매칭

빈 레이블 값을 매칭하면 해당 레이블이 설정되지 않은 모든 시계열도 선택됩니다.

```
# environment 레이블이 없거나 빈 값인 모든 시계열
nginx_http_response_count_total{environment=""}
```

필수 선택자

벡터 선택자는 반드시 메트릭 이름이나 빈 문자열과 일치하지 않는 레이블 매처를 하나 이상 지정해야 합니다.

```
# ❌ 잘못된 예
{job=~".*"}

# ✅ 올바른 예
{job=~".+"}
{job=~".*", method="get"}
```

성능 고려사항

효율적인 쿼리를 위한 팁

메트릭 이름만 사용하면 수천 개의 시계열이 선택될 수 있습니다.

- 항상 적절한 레이블 필터를 사용하여 결과를 제한하세요.
- 처음에는 Table 뷰에서 결과를 확인하고, 적절한 수준으로 필터링한 후 차트 뷰로 전환하세요.
- 수백 개 이하의 시계열이 이상적입니다.

Staleness (오래된 데이터)

시계열이 더 이상 수집되지 않으면 "stale"로 표시됩니다.

- Stale로 표시된 후에는 쿼리 결과에 포함되지 않습니다.
- 기본적으로 5분 동안 데이터가 수집되지 않으면 stale로 간주됩니다.

정규표현식

Prometheus는 RE2 문법을 사용합니다.

- 모든 정규표현식은 완전히 앵커링됩니다. (자동으로 `^`와 `$`가 추가됨)
- `env=~"foo"`는 `env=~"^foo$"`와 동일합니다.

주석

```
# 이것은 주석입니다
nginx_http_response_count_total{status="200"}  # 라인 끝 주석도 가능
```

추천 학습 순서

- 기초: rate(), sum(), avg()
- 중급: increase(), irate(), topk()
- 고급: 복잡한 비율 계산, 여러 함수 조합

참고 자료

- PromQL 기본 문서
- PromQL 함수 문서
- PromQL 사용 예제

MXQL

> MXQL을 알아보기 전에 메트릭스의 개념에 대해 먼저 알아보길 권장합니다. 메트릭스에 대한 자세한 내용은 다음 문서를 참조하세요.

MXQL이란?

MXQL은 와탭의 성능 데이터(메트릭스)를 유연하게 조회하기 위한 쿼리 언어입니다. 하나의 프로젝트에 포함된 여러 에이전트에서 수집한 메트릭스들을 종합적으로 조회하고 활용하기 위해서 사용합니다.

MXQL과 SQL의 차이

많이 알려진 SQL과 비교를 통해 MXQL의 개념을 알아봅니다.

용어

우선 SQL에서 사용하는 용어를 살펴봅니다.

SQL의 데이터 저장 구조

위와 같이 whatap의 데이터베이스(Database)에 product 테이블(Table)이 포함되어 있습니다. product 테이블(Table)은 id, description, 두 개의 컬럼(Column)이 포함합니다. SQL의 Database, Table, Column에 대응하는 MXQL의 용어는 각각 database, category, field입니다.

저장방식	MXQL	SQL
대분류	Database	Database
중분류	Category	Table
소분류	Field	Column

쿼리(Query)

MXQL과 SQL의 샘플 쿼리입니다. 각 라인의 오른쪽에 주석 내용을 참조하세요.

SQL query

```
SELECT time, pcode  -- Column 선택(time, pcode 컬럼만 조회하도록 설정합니다.)
FROM app_counter    -- Table 선택(app_counter 테이블에서 데이터를 조회합니다.)
WHERE tx_count = 1  -- 데이터 필터링(tx_count column의 값이 1인 데이터만 조회하도록 설정합니다.)
```

MXQL query

```
CATEGORY app_counter              -- app_counter 카테고리에서 데이터를 조회하도록 설정합니다.
TAGLOAD                           -- 데이터를 조회합니다.
SELECT [ time, pcode ]            -- 조회된 전체 컬럼 중에서 time, pcode 필드만 선택합니다.
FILTER { key : tx_count, value : 5}  -- tx_count 필드의 값이 5인 데이터만 남깁니다.
```

실행 결과

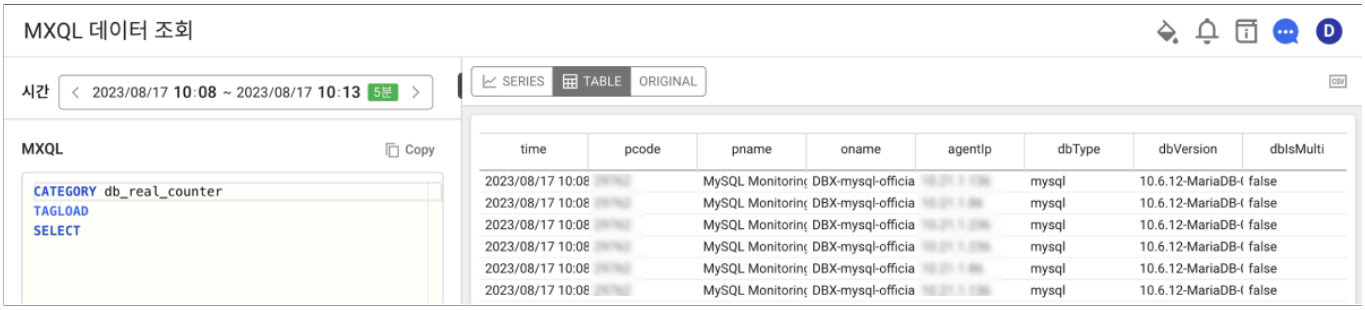
MXQL 쿼리를 수행하면 선택한 카테고리에서 선택한 필드의 메트릭스를 조회합니다.

`app_counter` 카테고리에서 `tx_count`, `tx_error` 지표를 조회하는 쿼리는 다음과 같습니다.

MXQL

```
CATEGORY app_counter              -- app_counter 카테고리에서 데이터를 조회하도록 설정합니다.
TAGLOAD                           -- 데이터를 조회합니다.
SELECT [time, oid, tx_count, tx_error] -- 조회하고 싶은 필드의 이름을 설정합니다.
```

쿼리를 수행하면 다음과 같이 메트릭스를 조회합니다.



메트릭스에는 항상 `time` , `oid` 값을 포함하기 때문에 MXQL 쿼리에서도 `time` , `oid` 필드를 항상 포함해 조회할 것을 권장합니다. 최종 조회한 데이터가 언제(`time`) 어떤 에이전트(`oid`)에서 수집한 메트릭스인지 확인할 수 있습니다.

MXQL 문법 가이드

형식

MXQL은 각 라인마다 **명령어**와 **오퍼랜드**로 구성되며 띄어쓰기로 구분합니다.

```
<명령어> <오퍼랜드>
```

`명령어` 는 한 단어의 예약어입니다. `명령어` 는 대문자로 입력하며 `오퍼랜드` 는 소문자로 입력합니다. `명령어` 마다 입력 가능한 `오퍼랜드` 의 형식은 정해져 있습니다. `오퍼랜드` 에는 4가지 타입의 값이 올 수 있습니다.

1. 오퍼랜드가 없는 경우

   ```
   TAGLOAD
   ```

2. 문자열(숫자 혹은 단어)

   ```
   CATEGORY app_counter
   ```

3. 문자열 배열

   ```
   SELECT [ time, pcode ]
   ```

4. JSON 문자열 타입

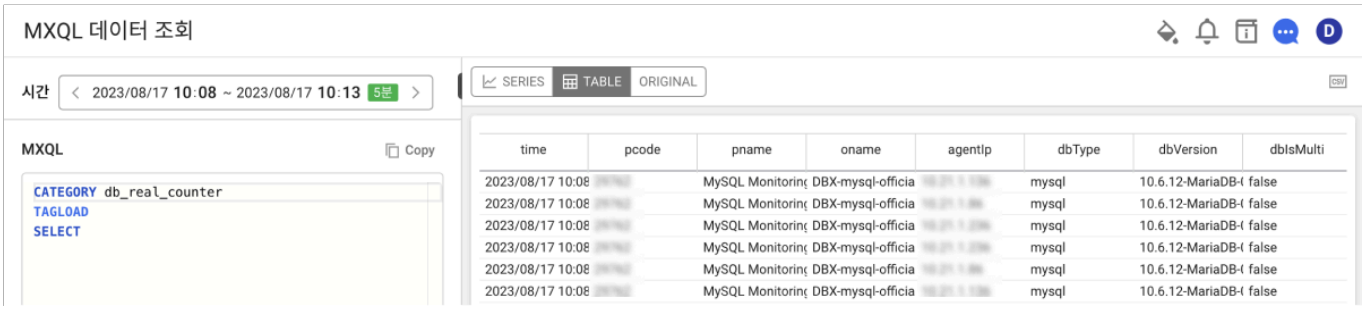
   ```
   FILTER { key : tx_count, value : 5}
   ```

Sample MXQL query

```
CATEGORY app_counter
TAGLOAD
SELECT [ time, pcode ]
FILTER { key : tx_count, value : 5}
```

테스트 환경

홈 화면 > 프로젝트 선택 > **사이트맵** > **실험실** > **MXQL 데이터 조회** 메뉴에서 MXQL 쿼리를 테스트할 수 있습니다.



> ⓘ 메트릭스에는 태그와 필드가 구분되어 있지만 **MXQL 데이터 조회** 메뉴에서는 태그와 필드의 구분 없이 표현합니다.

단계별 구성

MXQL은 단계별 구성을 가지고 있습니다. 각 단계별로 사용할 수 있는 **명령어**의 종류가 정해져 있으며 각 단계의 이름과 특징은 다음과 같습니다.

1. **메트릭스 선택**: 어떤 에이전트에서 수집한 어떤 메트릭스를 사용할지 선택하세요.
2. **메트릭스 로딩**: 이전 단계에서 설정한 값들을 사용해서 메트릭스를 불러옵니다. 대부분의 경우 `TAGLOAD` 를 이용하며, 특수한 경우에만 `FLEXLOAD` 를 이용하세요. `FLEXLOAD` 를 사용해야 하는 경우는 다음 문서를 참조하세요.
3. **메트릭스 가공**: 이전 단계에서 불러온 메트릭스에 대해서 단계별로 가공을 수행합니다.

Example

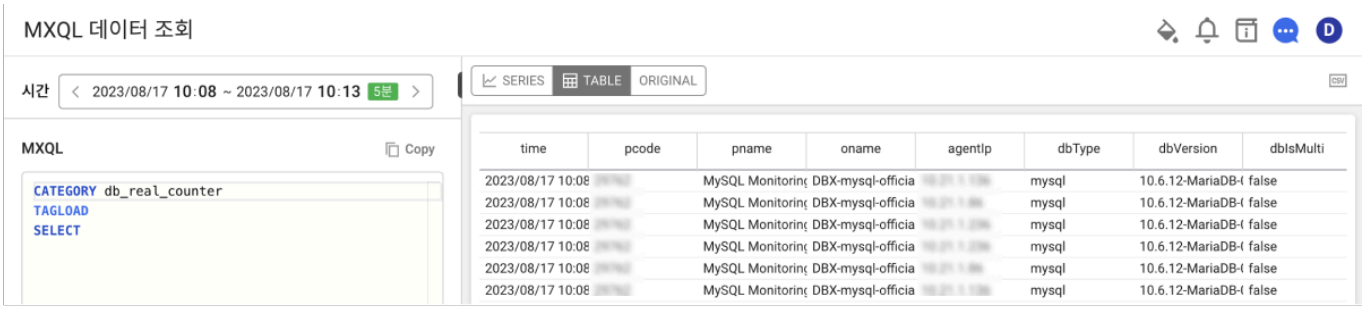
```
# 메트릭스 선택 단계
CATEGORY app_counter -- 카테고리 선택

# 메트릭스 로딩 단계
TAGLOAD -- 데이터 1000개 조회

# 메트릭스 가공 단계
```

```
SELECT [time, oid, active_tx_count, tx_count, tx_error] -- 1000개 데이터의 5개 필드만 다음 단계로 전달
FILTER {expr : "tx_count > 40"}       -- 데이터 1000개 중 100개만 통과
FILTER {expr : "active_tx_count > 10"} -- 데이터 100개 중 10개만 통과
FILTER {expr : "tx_error < 3"}        -- 데이터 10개 중 3개만 통과
```

다음은 메트릭스 가공 단계를 모두 통과한 메트릭스 예시입니다.



주석

"#" 또는 "--"으로 시작하는 문장은 무시됩니다.

Example

```
# 데이터 조회 설정
CATEGORY app_counter

# 데이터 조회
TAGLOAD

# 데이터 가공
SELECT [ time, pcode ]
FILTER { key : tx_count, value : 5}
```

MetricValue(복합값)

MetricValue(복합값)은 메트릭스 가공 단계에서 자주 사용하는 연산을 편리하게 지원하는 MXQL의 자료 구조입니다. 메트릭스 가공 단계의 GROUP, UPDATE 명령어는 메트릭스가 MetricValue 형태로 저장되어 있을 때만 사용할 수 있습니다.

예를 들어 다음과 같은 데이터가 있다고 가정해보겠습니다.

time	tx_count
2021/06/24 13:40:00	1
2021/06/24 13:40:10	2
2021/06/24 13:40:20	3
2021/06/24 13:40:30	4
2021/06/24 13:40:40	5
2021/06/24 13:40:50	6

위 데이터를 30초 간격으로 GROUP의 merge 옵션을 진행하면 다음과 같은 형태로 데이터를 변형할 수 있습니다.

time	tx_count
2021/06/24 13:40:00 ~ 2021/06/24 13:40:20	1,2,3에 대한 MetricValue
2021/06/24 13:40:30 ~ 2021/06/24 13:40:50	4,5,6에 대한 MetricValue

데이터를 MetricValue로 변환하면 총 6가지 옵션을 사용할 수 있습니다.

옵션	기능
sum	MetricValue에 포함된 값을 더합니다.
min	MetricValue에 포함된 값 중 최소값을 구합니다.
max	MetricValue에 포함된 값 중 최대값을 구합니다.
last	MetricValue에 포함된 값 중 마지막에 추가된 값을 구합니다.
avg	MetricValue에 포함된 값의 평균을 구합니다.

옵션	기능
cnt	MetricValue에 포함된 값의 갯수를 구합니다.

MetricValue 옵션은 UPDATE 명령어를 통해 이용할 수 있습니다.

UPDATE

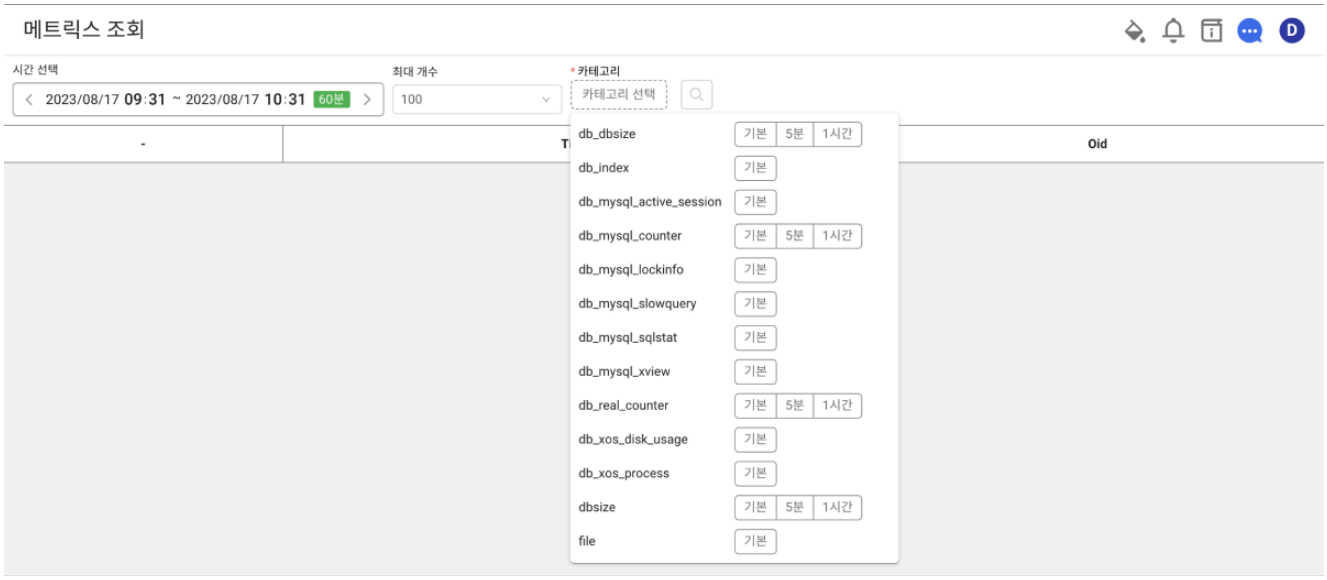
```
CATEGORY app_counter
TAGLOAD
SELECT [ time, okindName, okind, apdex_satisfied, apdex_tolerated, apdex_total]
-- GROUP 명령어의 merge 옵션을 통해 MetricValue로 변환할 field를 설정
GROUP { timeunit:5000, pk:okind, last:okindName, merge:[apdex_satisfied, apdex_tolerated, apdex_total] }
-- UPDATE 명령어를 통해 sum 옵션을 적용
UPDATE { key:[apdex_satisfied, apdex_tolerated, apdex_total], value:sum }
```

MetricValue 타입의 데이터 이용 방법

- GROUP 명령어의 merge 옵션에 필드를 설정합니다.

  ```
  CATEGORY app_counter
  TAGLOAD
  SELECT [ time, okindName, okind, apdex_satisfied, apdex_tolerated, apdex_total]
  -- GROUP 명령어의 merge 옵션을 통해 MetricValue로 변환할 field를 설정
  GROUP { timeunit:5000, pk:okind, last:okindName, merge:[apdex_satisfied, apdex_tolerated, apdex_total] }
  -- UPDATE 명령어를 통해 sum 옵션을 적용
  UPDATE { key:[apdex_satisfied, apdex_tolerated, apdex_total], value:sum }
  ```

- 수집서버에 데이터를 저장할 때부터 모든 필드에는 MetricValue 형식으로 저장된 카테고리가 있습니다. **사이트맵** > **분석** > **메트릭스 조회** > **카테고리** 옵션에는 **기본**, **5분**, **1시간** 단위로 선택할 수 있는 카테고리를 확인할 수 있습니다. 여기서 **5분** 또는 **1시간**을 선택할 수 있는 카테고리가 MetricValue 형식으로 저장된 카테고리입니다.



5분 또는 **1시간**을 선택할 수 있는 카테고리의 이름에 `{m5}` 또는 `{h1}`을 조합하면 `GROUP` 명령어의 merge 옵션을 적용하지 않고 바로 `UPDATE` 명령어의 `sum` 옵션을 적용할 수 있습니다.

```
CATEGORY app_counter{m5}
TAGLOAD
SELECT [time, pname, host_ip, pid, httpc_count]
-- GROUP 명령어를 적용하지 않아도 이미 데이터가 MetricValue 타입이기 때문에 UPDATE 명령어를 적용할 수 있습니다.
UPDATE { key : httpc_count, value : avg }
```

MetricValue 타입의 기본 연산 `avg`

MetricValue 형식 필드의 기본 출력 형태는 `avg` 입니다. 즉 MetricValue 형식의 필드는 별도의 옵션을 설정하지 않으면 `avg`를 적용합니다. 다음 두 쿼리는 같은 결과를 가집니다.

- **avg를 지정하지 않은 경우**

```
CATEGORY app_counter
TAGLOAD
SELECT [time, pcode,pname, tps]
GROUP  {timeunit:5000, pk:pcode, last: pname, merge:tps}
```

- **avg를 지정한 경우**

```
CATEGORY app_counter
TAGLOAD
SELECT [time, pcode,pname, tps]
GROUP  {timeunit:5000, pk:pcode, last: pname, merge:tps}
UPDATE {key:tps, value:avg}
```

사전 정의 MXQL 쿼리문

MXQL 쿼리를 직접 작성하지 않고 사전에 정의된 MXQL 쿼리파일의 경로를 설정해 MXQL을 수행할 수 있습니다. 예를 들어 '에이전트별 액티브TX 건수', '<구간별> 건수', '최근 15초'를 구하는 MXQL 쿼리는 다음과 같습니다.

```
HEADER {  act0$:I, act3$:I, act8$:I, act$:I}
TIME-RANGE {duration:15s, etime:$etime}
OIDSET {oid:$oid, okind:$okind, onode:$onode}
CATEGORY app_counter

TAGLOAD {backward:true}

SELECT [oid, oname, active_tx_0, active_tx_3, active_tx_8, active_tx_count, pcode]
FIRST-ONLY {key:oid}
RENAME {src:active_tx_0,  dst:act0}
RENAME {src:active_tx_3,  dst:act3}
RENAME {src:active_tx_8,  dst:act8}
RENAME {src:active_tx_count,  dst:act}
CREATE {key:_id_, from:oid}
CREATE {key:_name_, from:oname}

INJECT default
```

만약 위 쿼리가 수집서버에 등록되어 있다면 다음과 같이 입력하는 것만으로 같은 데이터를 조회할 수 있습니다. 설정한 경로에 저장한 서버의 파일 내용을 읽어서 호출해주는 방식입니다. 사전 정의한 파일의 경로를 입력하세요.

```
/app/act_tx/act_tx_oid
```

다음 문서에서 사용 가능한 쿼리를 확인할 수 있습니다.

참고자료

바인드 변수(파라미터)

MXQL에서는 바인드 변수를 사용할 수있습니다. 바인드 변수는 `$` 로 시작합니다. 또한 value에 해당하는 부분만 사용할 수 있습니다.

```
SKIP $skip_value
```

```
SKIP [$skip_value]
```

```
SKIP {value:$skip_value}
```

key로 바인드 변수를 전달할 수 없습니다.

```
SKIP {$option:10}
```

쿼리에서 바인드 변수를 사용했으면 MXQL을 실행할 때 입력할 값을 전달해야 합니다.

ⓘ 바인드 변수의 이름은 대소문자 알파벳만 가능합니다. 숫자 및 특수문자는 바인드 변수의 이름에 포함할 수 없습니다.

데이터 로딩 방식

MXQL에서 조회하는 데이터는 메트릭스의 구조에 따라 서로 다른 로딩 명령어를 사용합니다.

- **태그 기반 메트릭스**
  태그와 필드로 구분된 데이터는 `TAGLOAD` 명령어를 사용해 로드합니다.
- **플랫 구조 메트릭스**
  모든 데이터가 필드에 저장된 메트릭스는 `FLEXLOAD` 명령어를 사용해 로드합니다.
- **로그 원본 데이터**

로그 원본을 조회할 때는 `LogSink` 명령어를 사용합니다.

- **로그 카운트 집계 데이터**
  시간 단위로 로그 건수를 집계할 때는 `LogSinkCount` 명령어를 사용합니다.

사전 정의된 MXQL 쿼리 목록

MXQL 쿼리를 직접 작성하지 않고 경로(path)로 설정할 수 있는 기능의 주된 목적은 복잡한 쿼리를 간편하게 호출하기 위함이 아니라, 관리자가 직접 본인의 의도대로 쿼리를 작성하고 이를 사용하는데 있습니다. 따라서 이미 Yard에 어떤 쿼리들이 포함되어 있는지 확인하고 사용하기보다 직접 쿼리를 등록하는 방향으로 활용해야 합니다. `JOIN` 명령어를 사용해야하는 경우는 MXQL 쿼리를 사용하는 경우 중에서도 특별한 경우이기 때문에 관리자가 직접 쿼리를 등록하고 해당 파일의 경로(path)를 사용하도록 되어 있습니다.

> ⓘ yard.conf 파일의 `mxql_root` 에 설정한 경로 아래에 등록하고 싶은 쿼리를 파일로 저장할 수 있습니다. (default ./mxql)

에이전트별 액티브TX 건수, 건수, 최근15초

- 경로 : /app/act_tx/act_tx_oid
- 쿼리 :

```
HEADER {  act0$:I, act3$:I, act8$:I, act$:I}

TIME-RANGE {duration:15s, etime:$etime}

OIDSET {oid:$oid, okind:$okind, onode:$onode}

CATEGORY app_counter

TAGLOAD {backward:true}

SELECT [oid, oname, active_tx_0, active_tx_3, active_tx_8, active_tx_count]
FIRST-ONLY {key:oid}
RENAME {src:active_tx_0,  dst:act0}
RENAME {src:active_tx_3,  dst:act3}
```

```
RENAME {src:active_tx_8,  dst:act8}
RENAME {src:active_tx_count,  dst:act}

CREATE {key:_id_, from:oid}
CREATE {key:_name_, from:oname}
```

에이전트 별 상세정보 & 에이전트별 액티브TX 건수, 건수, 최근15초

- 경로 : /app/act_tx/agent_with_tx
- 쿼리 :

```
CATEGORY agent_list

OIDSET {oid:$oid, okind:$okind, onode:$onode}

FLEXLOAD

JOIN {query:'/app/act_tx/act_tx_oid', pk:oid, field:[act0,act3,act8, act] }

UPDATE {key:act0, notnull:0}
UPDATE {key:act3, notnull:0}
UPDATE {key:act8, notnull:0}
UPDATE {key:act, notnull:0}

RENAME {src:[act0, act3, act8, act],  dst:[normal, slow, verySlow, total]}

INJECT default
```

에이전트 별 상세정보 & 에이전트별 액티브TX 건수, 건수, 최근15초

메트릭스 선택

MXQL 문법을 이용해 메트릭스를 선택하는 명령어에 대해서 알아봅니다.

명령어	기능
CATEGORY	어떤 카테고리에서 데이터를 조회할지 선택합니다. 이 명령어와 오퍼랜드는 반드시 설정해야 합니다.
OKIND	특정 `OKIND`로부터 수집 데이터만 조회하도록 설정합니다.
OID	특정 `OID`로부터 수집 데이터만 조회하도록 설정합니다.
ONODE	특정 `ONODE`로부터 수집 데이터만 조회하도록 설정합니다.
HEADER	프론트 단에 전달하기 위한 값을 설정합니다.
TIME-RANGE	데이터를 조회할 시작 시간, 종료 시간을 설정합니다.

CATEGORY

어떤 카테고리에서 데이터를 조회할지 선택합니다. 이 **명령어**와 **오퍼랜드**는 반드시 설정해야 합니다.

오퍼랜드

- **문자열**: 지정한 카테고리의 데이터를 조회합니다. 조회시간에 상관없이 항상 `app_counter` 카테고리의 데이터를 로드합니다

  ```
  CATEGORY app_counter
  TAGLOAD
  ```

- **JSON**: 데이터 조회 시간의 길이에 따라서 카테고리를 선택하도록 설정할 수 있습니다.

  조회시간이 6시간, 3일, 그 이상인 경우에 대해 각각 `app_counter` , `app_counter{m5}` , `app_counter{h1}` 카테고리를 선택합니다.

```
CATEGORY {"app_counter":6h, "app_counter{m5}":3d, "app_counter{h1}":unlimit }
TAGLOAD
```

- `CATEGORY` 의 오퍼랜드로 하나의 값만 지정할 수 있습니다. 여러 `CATEGORY` 의 데이터를 한 번에 확인하고 싶은 경우 JOIN을 사용해야합니다.
- 로딩 방식(TAGLOAD 또는 FLEXLOAD)에 따라서 설정 가능한 카테고리가 달라집니다.
- `CATEGORY` 의 이름에 `{m5}` 또는 `{h1}` 가 붙어있는 경우 MetricValue를 참고해주세요.

OID, OKIND, ONODE

특정 `OID` , `OKIND` , `ONODE` 로부터 추출한 데이터만 조회하도록 설정합니다. `OID` , `OKIND` , `ONODE` 중 한 가지 값만 설정할 수 있습니다.

오퍼랜드

문자열로 하나의 값을 설정하거나 문자열 배열로 여러개의 값을 설정할 수 있습니다.

Example 1

```
CATEGORY app_counter
OID 1388369480
TAGLOAD
```

Example 2

```
CATEGORY app_counter
OID [1388369480, 1388369481]
TAGLOAD
```

Example 3

```
CATEGORY app_active_stat
ONODE 1693789385
```

```
TAGLOAD
```

- `OID` , `OKIND` , `ONODE` 중 한 가지 값만 설정할 수 있습니다.
- 오퍼랜드를 입력하지 않은(또는 파라미터의 값이 전달되지 않은) 명령어는 무시합니다.
- `OID` , `OKIND` , `ONODE` 를 중복해서 사용하는 경우 가장 마지막에 입력한 명령어만 적용합니다.
- `OIDSET` 은 deprecated되어 `OID` , `OKIND` , `ONODE` 중 한 가지 명령어를 사용하는 것을 권장합니다.
- 데이터 로드 단계에서부터 필요한 데이터만을 조회한다는 점에서 데이터 가공 단계의 FILTER를 적용하는 것과 차이점이 있습니다.

`OKIND` , `ONODE` 명령어의 경우 `CATEGORY` 명령어에서 설정한 카테고리에 `okind` , `onode` 관련 필드(`okind` , `onode` , `okindName` , `onodeName`)를 포함하는 경우만 사용할 수 있습니다. **사이트맵** > **분석** > **메트릭스 조회** 메뉴에서 어떤 카테고리에 어떤 필드를 포함하고 있는지 확인할 수 있습니다.

HEADER

MXQL로 조회한 데이터는 Flex 보드의 위젯에서 사용됩니다. MXQL로 조회한 데이터 중 어떤 필드를 어떤 타입으로 사용해서 Flex 보드의 위젯을 표현하는지에 대한 정보를 설정할 수 있습니다.

오퍼랜드

JSON 문자열로만 설정할 수 있습니다.

Example

```
HEADER { apdex_satisfied$:I, apdex_tolerated$:I, apdex_total$:I,  apdex$:F, category: app_counter}
OID $oid
CATEGORY app_counter
TAGLOAD
```

Flex 보드의 위젯에서 사용하는 형식에 맞추어 전달해야 합니다.

TIME-RANGE

데이터 조회 시간 범위를 설정할 수 있습니다. 기본적으로 테스트 환경의 DatePicker를 사용해서 시간을 설정합니다. 만약 테스트 환경에서 본 명령어를 사용하는 경우 DatePicker를 설정한 값을 무시합니다.

- 최근 15초 동안의 데이터만 조회

```
TIME-RANGE { recent : 15s }
CATEGORY app_counter
TAGLOAD
SELECT
```

- `etime` 이전 15초 동안의 데이터만 조회 (파라미터로 `etime` 을 전달)

```
TIME-RANGE {duration:15s, etime:$etime}
CATEGORY app_counter
TAGLOAD
SELECT
```

- `stime` 부터 15초 동안의 데이터만 조회 (파라미터로 `stime` 을 전달)

```
TIME-RANGE {duration:15s, stime:$stime}
CATEGORY app_counter
TAGLOAD
SELECT
```

메트릭스 로딩

MXQL 문법을 이용해 메트릭스를 불러오는 명령어에 대해서 알아봅니다.

명령어	기능
TAGLOAD	수집한 데이터를 `tag-field` 형식으로 저장하는 카테고리의 정보를 조회할 때 사용합니다.
FLEXLOAD	수집한 데이터를 `field` 형식으로 저장하는 카테고리의 정보를 조회할 때 사용합니다.
LogSink	로그 원본 데이터를 조회할 때 사용합니다.
LogSinkCount	시간 단위로 로그 건수를 집계할 때 사용합니다.

TAGLOAD

수집한 데이터를 `tag-field` 형식으로 저장하는 카테고리의 정보를 조회할 때 사용합니다.

옵션	기능
`{backward : true}`	시간 역순으로 데이터를 로드합니다.
`{filter : {key:fieldName, value :fieldValue}}`	fieldName 필드의 값이 fieldValue과 같은 데이터만 추출합니다.
`{filter : {key:fieldName, exclude :fieldValue}}`	fieldName 필드의 값이 fieldValue과 같은 데이터를 제외하고 추출합니다.
`{filter : {key:fieldName, like :fieldValue}}`	fieldName 필드의 값이 fieldValue을 부분 문자열로 가지는 데이터만 추출합니다.
`{filter : {key:fieldName, notlike :fieldValue}}`	fieldName 필드의 값이 fieldValue을 부분 문자열로 데이터를 제외하고 추출합니다.

- 옵션을 설정하지 않은 경우

```
CATEGORY app_counter
TAGLOAD
```

- backward 옵션을 설정한 경우

```
CATEGORY app_counter
TAGLOAD {backward : true}
```

- value filter를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD {filter : {key:pid, value:905}}
```

- exclude filter를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD {filter : {key:pid, exclude:905}}
```

- like filter를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD {filter : {key:okindName, like:keeper}}
```

- notlike filter를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD {filter : {key:okindName, notlike:keeper}}
```

- 복수의 filter를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD { filter:[{key:'host_ip', exclude:'192.168.1.102'}, {key:'container', like:'gateway'}] }
```

> • `TAGLOAD` 와 `FLEXLOAD` 는 각각 설정할 수 있는 `CATEGORY` 의 값이 정해져있습니다.

- `fileter-like` 또는 `filter-notlike` 옵션을 사용할 때 값으로 숫자가 오는 경우 작은 따옴표('') 또는 큰 따옴표("")로 감싸주어야 동작합니다.

```
CATEGORY app_counter
TAGLOAD { filter:[{key:'host_ip', exclude:'192.168.1.102'},{key:okindName, like:"1"}] }
```

FLEXLOAD

수집된 데이터를 `field` 형식으로 저장하는 카테고리의 정보를 조회할 때 사용합니다.

옵션	기능
`{backward : true}`	시간 역순으로 데이터를 로드합니다.

데이터 검색조건 단계에서 설정한 정보에 따라 데이터를 로드합니다.

```
CATEGORY event_cache
FLEXLOAD {backward : true}
```

대부분의 카테고리는 `TAGLOAD` 를 사용합니다. 다음 문서에 포함한 카테고리의 데이터를 사용할 때에만 `FLEXLOAD` 를 사용합니다.

FLEXLOAD를 사용해야하는 카테고리 목록

- `agent_list` 카테고리

```
CATEGORY agent_list
FLEXLOAD
SELECT
```

- `db_agent_list` 카테고리

```
CATEGORY db_agent_list
FLEXLOAD
SELECT
```

- `agent_count` 카테고리

```
CATEGORY agent_count
FLEXLOAD
SELECT
```

- `event_cache` 카테고리

```
CATEGORY event_cache
FLEXLOAD
SELECT
```

LogSink

로그 원본 데이터를 조회할 때 사용합니다.

```
CATEGORY AppLog
LogSink
```

로그 필터링

특정 조건으로 로그를 필터링할 때 **LogTag**를 사용합니다.

```
CATEGORY AppLog
LogTag {key:city, value:"junju"}
LogTag {key:status, value:"200", exclude:true}
LogSink
```

LogTag 파라미터

- **key**: 검색할 인덱스 키
- **value**: 검색할 값 (와일드카드 사용 가능)
- **exclude**: 검색 제외 여부 (true/false)

중요사항

- LogTag는 LogSink나 LogSinkCount 앞에 기술되어야 합니다
- LogTag는 동일 키 혹은 다른 키에 대해서 다양하게 기술할 수 있습니다
- 검색 키워드 단위로 LogTag가 늘어나야 합니다

예제

특정 키워드가 포함된 로그의 건수를 조회하는 예제입니다.

```
CATEGORY AppLog
LogTag {key:content, value:"*DebugUtil*", exclude:false}
LogTag {key:content, value:"*HttpServletRequest*", exclude:true}
LogSink
SELECT
GROUP {timeunit:1d, pk:pcode, merge:[rows]}
RENAME {src:_rows_, dst:rows}
CREATE {key:_pk_, value:pcode}
UPDATE {key:rows, value:sum}
```

쿼리 설명

- 첫 번째 LogTag: content 필드에 "DebugUtil"이 포함된 로그 검색
- 두 번째 LogTag: content 필드에 "HttpServletRequest"가 포함된 로그 제외
- GROUP의 timeunit:1d로 일별 집계
- merge:[rows]로 로그 건수를 MetricValue 타입으로 변환
- UPDATE의 sum으로 일별 총 건수 계산

LogSinkCount

시간 단위로 로그 건수를 집계할 때 사용합니다.

기본 사용법

```
CATEGORY AppLog
LogSinkCount
```

시간대별로 로그 건수가 집계됩니다 (예: 1949, 1625, 1674... 건).

그룹별 집계

특정 필드를 기준으로 그룹화하여 로그를 집계할 수 있습니다.

단일 필드 그룹화

```
CATEGORY AppLog
LogSinkCount {group:oname}
```

여러 필드 그룹화

```
CATEGORY AppLog
LogSinkCount {group:"oname,oid"}
```

쉼표로 구분된 여러 개의 필드를 지정할 수 있습니다.

시간당 그룹별 집계

숫자 필드에 대한 집계 연산을 수행할 수 있습니다.

```
CATEGORY AppLog
LogSinkCount {group:oname, aggr:price.n}
```

> ⓘ Aggr에 저장할 수 있는 필드는 숫자(n) 필드에 한정됩니다.

여러 필드 집계

```
CATEGORY AppLog
LogSinkCount {group:oname, aggr:"price.n,age.n"}
```

aggr에는 쉼표로 구분된 여러 개의 필드를 입력할 수 있으나 계산은 각각 이루어집니다.

LogTag와 함께 사용

LogTag를 사용하여 특정 조건의 로그만 집계할 수 있습니다.

```
CATEGORY AppLog
LogTag {key:level, value:"ERROR"}
LogSinkCount {group:oname}
```

위 예제는 ERROR 레벨의 로그만 필터링하여 oname별로 건수를 집계합니다.

메트릭스 가공

MXQL 문법을 이용해 메트릭스를 가공하는 명령어에 대해서 알아봅니다. 메트릭스 가공 단계는 조회된 데이터들이 각 단계를 순차적으로 수행하는 구조입니다. 따라서 이 단계에 포함되는 **명령어**들의 입력 순서가 중요합니다.

명령어	기능
ROWNUM	줄번호 필드를 추가합니다.
SELECT	필드를 선택합니다. 선택되지 않은 필드은 조회되지 않습니다.
CREATE	필드를 추가합니다.
DELETE	필드를 삭제합니다.
RENAME	필드의 이름을 변경합니다.
GROUP	데이터를 그룹화합니다.
ORDER	데이터를 정렬합니다.
JOIN	다른 MQL에서 조회한 데이터를 본 데이터에 컬럼단위로 추가할때 사용합니다.
UPDATE	데이터를 가공, 정제합니다.
LIMIT	추출되는 데이터의 수를 제한합니다.
SKIP	해당 위치에서 조회되는 일부 데이터를 무시합니다.
FILTER-KEYS	특정 데이터를 가지고 있는 데이터만 추출합니다.
FIRST-ONLY	데이터 중에서 처음 데이터만을 통과 시키고 나머지는 버립니다.
TIME-FILTER	특정시간의 데이터를 SKIP할때 사용합니다.
INJECT	해당 위치에 MXQL 쿼리를 추가합니다.

명령어	기능
ADJUST	숫자 필드의 값을 변경하기 위해 사용합니다.
FILTER	특정 조건을 가지고 있는 데이터만 다음 단계로 전달합니다.

ROWNUM

줄번호 필드를 추가합니다.

Example

```
CATEGORY agent_list
FLEXLOAD
ROWNUM
```

SELECT

필드를 선택합니다. 선택하지 않은 필드는 다음 단계로 전달하지 않습니다.

옵션 이름	옵션 기능
default	`like`, `notlike`와 상관없이 조회하고 싶은 필드를 설정합니다.
like	설정한 값을 부분 문자열로 가지는 필드만 조회합니다.
notlike	설정한 값을 부분 문자열로 가지지 않는 필드만 조회합니다.

- 모든 필드를 선택하는 경우 1 (`SELECT` 명령어, 오퍼랜드를 모두 입력하지 않은 경우)

  ```
  CATEGORY app_counter
  TAGLOAD
  ```

- 모든 필드를 선택하는 경우 2 (`SELECT` 명령어의 오퍼랜드를 입력하지 않은 경우)

```
CATEGORY app_counter
TAGLOAD
SELECT
```

- 조회하고 싶은 필드 이름을 직접 설정하는 경우, 문자열배열 오퍼랜드를 사용합니다.

```
CATEGORY app_counter
TAGLOAD
SELECT [time, pcode]
```

- `default`로 설정할 필드의 값이 하나인 경우와 `like` 옵션을 사용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT {default:time, like:_m}
```

- `default`로 설정할 필드의 값이 여러 개인 경우와 `like` 옵션을 사용하는 경우, `default`로 설정할 필드의 값이 여러 개인 경우와 `like` 옵션을 사용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT {default:[time,name], like:_m}
```

- `like`와 `notlike`를 모두 사용하고 싶은 경우 `SELECT` 명령어를 두 번에 나누어서 입력해야 합니다.

```
CATEGORY app_counter
TAGLOAD
SELECT {default:[time,name], like:name}
SELECT {notlike:pname}
```

> - 만약 전체 필드를 선택할때는 오퍼랜드를 입력하지 않습니다.
> - `like`와 `notlike`는 한 번에 설정할 수 없습니다. 여러 번의 `SELECT`에서 나누어서 설정해야 합니다.

> `SELECT` 명령어는 출력되는 필드 순서를 변경할 때도 사용합니다.

CREATE

필드를 추가합니다.

옵션 이름	옵션 기능
value	특정 값을 가지는 필드를 생성합니다.
from	설정한 필드의 값을 가지는 필드를 생성합니다.
expr	입력한 수식의 결과를 값으로 가지는 필드를 생성합니다. 수식에는 필드의 이름이 사용될 수 있습니다.
oname	`oid` 컬럼의 이름을 설정하여 `oid` 값에 대응되는 `oname`의 값을 가지는 컬럼을 생성합니다.
okind	`okind` 컬럼의 이름을 설정하여 `okind` 값에 대응되는 `okind name`의 값을 가지는 컬럼을 생성합니다.
onode	`onode` 컬럼의 이름을 설정하여 `onode` 값에 대응되는 `onode name`의 값을 가지는 컬럼을 생성합니다.

- `value` 속성을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
CREATE {key:active$, value:'#'}
```

- `from` 속성을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
CREATE {key:_id_, from:okind }
```

- `expr` 속성을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
CREATE { key:apdex, expr:" (apdex_satisfied(apdex_tolerated*0.5))/apdex_total " }
```

- `okind` 속성을 설정하는 경우

```
CATEGORY agent_list
FLEXLOAD
CREATE { key : my_okind_name, okind : okind}
SELECT [ time, okind, okindName, my_okind_name]
```

DELETE

필드를 삭제합니다.

Example

```
CATEGORY app_counter
TAGLOAD
DELETE [pcode]
```

> 반드시 문자열 배열로 입력해야합니다. `DELETE` `pcode` 는 동작하지 않습니다. (2021-06-23 기준)

RENAME

필드의 이름을 변경합니다.

- `pcode` 필드의 이름 `my_pcode` 로 변경합니다.

```
CATEGORY app_counter
TAGLOAD
RENAME { src : pcode, dst : my_pcode }
```

> `time` 은 `ORDER` 에서 최우선으로 사용하는 정렬 기준으로 `time` 의 이름을 임의로 변경하면 ORDER가 동작하지 않을 수 있습니다.

GROUP

데이터를 그룹화합니다.

옵션 이름	옵션 기능
timeunit	그룹을 나눌 시간 기준을 설정합니다.
pk or primaryKey	그룹의 `primaryKey`를 설정합니다.
last	설정한 컬럼(column)의 데이터 중 마지막 값만 저장하고 싶을 때 설정합니다. `oname`과 같이 반복해서 같은 값이 사용될 때 사용합니다. 내부적으로 설정한 값을 key로 가지는 값에 계속 덮어써집니다.
listup	설정한 컬럼(column)의 데이터를 모두 메모리에 저장하고 싶을 때 설정합니다. 내부적으로는 설정한 값을 key로 가지는 List에 계속 값이 추가됩니다.
user	실시간 사용자를 계산하기 위한 옵션입니다. Blob 타입의 데이터를 저장한 컬럼(column)만 설정할 수 있습니다.(`app_user` 카테고리의 `logbits` 등)
merge	설정한 컬럼(column)의 데이터를 MetricValue(복합값)으로 저장하고 싶을 때 설정합니다. 내부적으로는 설정한 값을 key로 가지는 MetricValue값에 추가됩니다.
rows	하나의 그룹에 최대로 저장될 수 있는 데이터의 수를 설정합니다. 기본값은 `10000`입니다.

- 설정한 필드에 대해 `merge`로 설정하여 MetricValue로 설정한 뒤 sum 연산을 수행합니다.

```
CATEGORY app_counter
TAGLOAD
SELECT [ time, okindName, okind, apdex_satisfied, apdex_tolerated, apdex_total]
GROUP { timeunit:5000, pk:okind, last:okindName, merge:[apdex_satisfied, apdex_tolerated, apdex_total] }
UPDATE { key:[apdex_satisfied, apdex_tolerated, apdex_total], value:sum }
```

ⓘ 원칙적으로 `merge` 필드는 별도로 설정해야 합니다. 하지만 `last`, `merge`, `listup` 세 가지 속성을 모두 명시하지

ⓘ 않은 경우에는 모든 `number` 필드는 `merge` 필드로, `number` 필드가 아닌 필드는 `last` 필드로 자동 선택 됩니다.

⚠ 만약 레코드에 `time` 필드가 없으면 전체를 그룹핑합니다.

- `GROUP` 명령어가 수행되기 전에 `SELECT` 명령어로 `time` 필드를 설정하지 않은 경우
- `RENAME` 명령어로 `time` 필드의 이름을 변경한 경우
- `DELETE` 명령어로 `time` 필드를 삭제한 경우

UPDATE

필드의 데이터를 수정합니다. MetricValue 상태인 필드에 대해서 연산을 선택할 수 있습니다.

옵션	기능
sum	MetricValue에 포함된 값을 더합니다.
min	MetricValue에 포함된 값 중 최소값을 구합니다.
max	MetricValue에 포함된 값 중 최대값을 구합니다.
last	MetricValue에 포함된 값 중 마지막에 추가된 값을 구합니다.
avg	MetricValue에 포함된 값의 평균을 구합니다.
cnt	MetricValue에 포함된 값의 갯수를 구합니다.
datetime	시간 데이터의 형식을 변경합니다.
timezone	시간 데이터의 기준을 설정합니다.
notnull	설정한 컬럼의 값이 `null`인 경우 적용할 `default` 값을 설정합니다.
pct	`GROUP` 명령 수행 시 `percentile`을 위해 필드의 모든 값을 listup 했을 경우 `percentile` 값을 필드값으로 변경할

옵션	기능
	수있습니다.
decimal	필드의 숫자 데이터를 포멧팅 할 수 있습니다.

다음의 옵션을 설정해 데이터의 값을 수정할 수 있습니다.

- `value` 옵션을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT [time, pcode,pname, tps]
GROUP  {timeunit:5000, pk:pcode, last: pname, merge:tps}
UPDATE {key:tps, value:sum}
```

- `datetime` , `timezone` 옵션을 설정하는 경우 `CREATE {key:localtime, from:time}` 의 경우 `time` 필드의 값 `long` 타입의 값으로 복사합니다.

```
CATEGORY app_user
TAGLOAD
SELECT [time, pcode, pname, logbits]
CREATE {key:localtime, from:time}
UPDATE {key:localtime, datetime:'yyyyMMdd HH:mm:ss', timezone: GMT9}
```

- `notnull` 옵션을 설정하는 경우

```
UPDATE {key:tps, notnull:0}
```

- `pct` 를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT [ time, pcode, tx_count ]
GROUP { key : pcode, listup : tx_count}
UPDATE { key : tx_count, pct : 90}
```

- `decimal` 옵션을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT [ time, oname, apdex_satisfied, apdex_tolerated, apdex_total]
GROUP  { timeunit:5m, pk:oname}
UPDATE { key:[apdex_satisfied, apdex_tolerated, apdex_total], value:sum }
CREATE { key:apdex, expr:" (apdex_satisfied(apdex_tolerated/2.0))/apdex_total " }
UPDATE { key:time, datetime:'yyyyMMdd HH:mm:ss', timezone:'GMT9'}
UPDATE { key:apdex, decimal:'0.000'}
ROWNUM
```

- `{datetime:'yyyyMMdd HH:mm:ss'}` 의 경우, 문자 쌍점(:)을 포함하고 있어, 반드시 작은 따옴표('') 또는 큰 따옴표("")로 감싸주어야 합니다.
- `pct: 90` 은 90%번째의 값을 선택한다는 의미입니다. 단, 해당 필드는 `GROUP` 명령을 수행할 때 `listup` 필드로 설정돼 있어야 합니다.
- format의 형식은 Java, Decimal Format을 사용합니다.

ORDER

데이터를 정렬합니다.

옵션	기능
key	정렬할 필드를 선택합니다.
sort	정렬한 direction을 설정합니다. (asc 또는 desc)
rows	같은 시간 값을 가지는 데이터를 최대 몇 개 남길지 설정합니다. default `10000`

- `key` , `sort` , `rows` 를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
```

```
SELECT [time, pname, host_ip, pid, httpc_count]
ORDER {key: [pid, host_ip, httpc_count] , sort: [desc, desc, desc], rows:2}
```

- 두 번에 나누어서 정렬하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT [time, pname, host_ip, pid, httpc_count]
ORDER {key: [pid, host_ip, httpc_count] , sort: [desc, desc, desc], rows:1000}
ORDER  {key:tps, sort:desc}
```

> 데이터에 `time` 필드이 포함되는 경우에는 `ORDER` 의 key에 `time` 이 포함되어 있지 않아도 `time` 이 정렬의 최우선 기준이 됩니다.

JOIN

`JOIN` 명령어에 대한 설명에 앞서 join의 개념을 알아 보겠습니다. join은 두 쿼리문의 결과를 합쳐서 확인하기 위해서 이용합니다. 이 때 두 쿼리의 결과 중 어떤 필드를 기준으로 합칠지에 대한 정보를 전달해야하며 이 필드를 `pk` 또는 `primaryKey` 라고 합니다.

Time	Oid	Fields			
2021-06-30 15:30:00	2031382584	field_name_1	field_name_2	field_name_3	field_name_4
		sampleData	123	2.543	testData

표1. JOIN 명령어 샘플 데이터 - 첫 번째 쿼리 결과(pk = field_name_4)

Time	Oid	Fields			
2021-06-30 15:30:00	2031382584	field_name_4	field_name_5	field_name_6	field_name_7
		testData	myData	testData	myData

표2. JOIN 명령어 샘플 데이터 - 두 번째 쿼리 결과(pk = field_name_4)

Time	Oid	Fields						
2021-06-30 15:30:00	2031382584	field_name_1	field_name_2	field_name_3	field_name_4	field_name_5	field_name_6	field_name_4
		sampleData	123	2.543	testData	myData	testData	myData

표3. JOIN 명령어 샘플 데이터 - 두 쿼리 결과를 pk를 기준으로 join한 결과

표1과 **표2**는 쿼리의 결과를 나타내며, `pk`로 설정한 `file_name_4` 필드는 파란색으로 표시되어 있습니다. **표3**은 `pk`로 설정한 `file_name_4`를 기준으로 두 쿼리의 결과가 합쳐진 모습입니다.

두 MXQL에서 조회한 데이터를 합쳐서 볼 수 있습니다. `RENAME` 명령어와 `INJECT` 명령어는 `JOIN` 명령어를 수행한 결과를 가공하는 과정으로 join의 동작에는 영향을 미치지 않습니다.

- 첫 번째 쿼리 : CATEGORY agent_list FLEXLOAD
- 두 번째 쿼리 : /app/act_tx/act_tx_oid

```
CATEGORY agent_list
FLEXLOAD
JOIN {query:'/app/act_tx/act_tx_oid', pk:oid, field:[act0,act3,act8, act] }
RENAME {src:[act0, act3, act8, act], dst:[normal, slow, verySlow, total]}
INJECT default
```

샘플 쿼리의 결과 예시는 다음과 같습니다.

Time	Oid	Fields							
2021-06-30 15:30:00	2031382584	pcode	pname	...	type	act0	act3	act8	act
		sampleData	123	...	testData	0	1	0	1

표4. 샘플 쿼리 결과

```
CATEGORY agent_list
FLEXLOAD
JOIN {query:'/app/act_tx/act_tx_oid', pk:oid, field:[act0,act3,act8,act] }
```

ⓘ Yard에 저장하는 모든 데이터는 `time`, `oid`의 값을 가지고 있습니다. 언제(`time`) 어떤 에이전트(`oid`)로부터 수집한 정보인지를 나타내기 위함입니다. 이 필드들도 `pk`로 사용될 수 있습니다.

> - `JOIN` 명령어에 사용하는 첫 번째 쿼리는 직접 작성한 MXQL 쿼리, 두 번째 쿼리는 path로 설정할 수 있는 쿼리만 가능합니다.
> - `JOIN` 명령어를 사용한 MXQL 쿼리 전체를 Yard에 파일로 등록해서 사용하면 3개 이상의 카테고리도 `JOIN` 할 수 있습니다.

LIMIT

추출하는 데이터의 수를 제한합니다. 앞에서부터 설정한 수만큼의 데이터를 다음 단계로 전달합니다.

가장 먼저 추출된 데이터 3개를 출력합니다.

```
CATEGORY app_counter
TAGLOAD
LIMIT 3
```

SKIP

이전 단계로부터 전달받은 데이터 중 일부 데이터를 무시합니다.

1~5번째 데이터는 제외되고 6번째부터 10개의 데이터를 보여줍니다.

```
CATEGORY app_counter
TAGLOAD
SKIP 5
LIMIT 10
```

FILTER-KEYS

특정 데이터를 가지고 있는 데이터만 추출합니다.

```
CATEGORY app_counter
TAGLOAD
```

```
FILTERKEYS {keys : [oid], values : [497765289]}
```

> ! key , value 가 아닌 keys , values 입니다. 복수형 s에 주의하세요.

FIRST-ONLY

특정 데이터(쌍)을 가진 첫 데이터만 다음 단계로 전달합니다.

```
CATEGORY app_counter
TAGLOAD
FIRST-ONLY {key:oid}
```

```
CATEGORY app_counter
TAGLOAD
FIRST-ONLY {key: [httpc_count, type]}
SELECT [httpc_count, type]
```

```
CATEGORY app_counter
TAGLOAD
FIRST-ONLY [httpc_count, type]
SELECT [httpc_count, type]
```

> ! 데이터 로드 단계에서 {backward : true} 를 사용했다면 본 명령어의 결과가 달라질 수 있습니다.

TIME-FILTER

특정시간의 데이터를 SKIP할때 사용합니다.

옵션	기능
time	yyyy/MM/dd HH:mm:ss 로 설정합니다. 설정한 시간 기준 duration:1000 을 설정합니다. (설정한 시간 기준

옵션	기능
	1000ms동안의 데이터 제외)
date	`yyyy/MM/dd`로 설정해야합니다. 설정한 시간 기준 duration:d1과 같이 설정합니다. (설정한 시간 기준 하루동안의 데이터 제외)
duration 또는 dur	필터링 범위를 설정합니다.(`d1` : 1일, `h1` : 1시간 , `m1` , `m5` , `m10` : 1분, 5분, 10분 , `number` : millisec)
timezone	데이터 타임존을 설정합니다. (예시, '`GMT9`')
gmt	데이터 타임존을 설정합니다. (예시, `9` 또는 `-9`)

```
CATEGORY app_counter
TAGLOAD
TIME-FILTER { date:'2020/07/28' , timezone:'GMT9'}
```

```
CATEGORY app_counter
TAGLOAD
TIME-FILTER {time:'2021/06/22 00:00:00', gmt:9 }
```

INJECT

해당 위치에 MXQL 쿼리를 추가합니다.

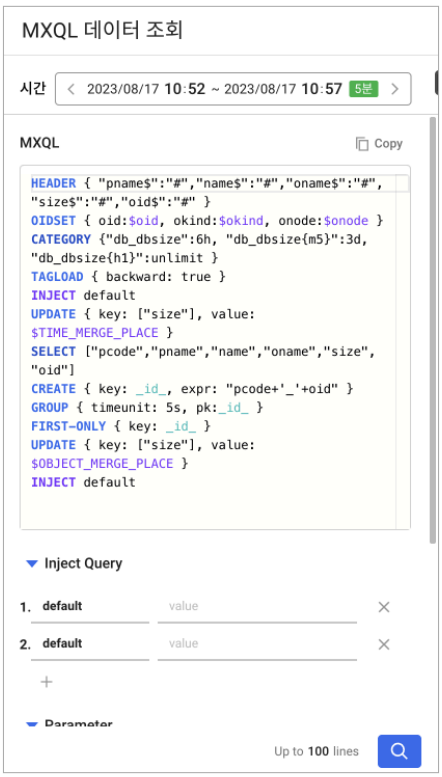
`default` 위치에 inject될 MXQL 쿼리를 전달해야 합니다.

```
CATEGORY app_counter
TAGLOAD
SELECT
INJECT default
ROWNUM
```

> ⚠ 프론트 단에서 `INJECT` 명령어의 오퍼랜드에 맵핑될 정보를 전달해주어야 합니다. 다음 예제는 키가 `default` 로

설정된 값을 추가합니다.

사이트 맵 > **MXQL 데이터 조회**'에서 INJECT 값 전달하는 예시



ADJUST

숫자 필드의 값을 변경하기 위해 사용합니다. (`time` 값은 변경할 수 없습니다.)

옵션 이름	옵션 기능
add	모든 숫자 데이터에 값을 더합니다.
sub	모든 숫자 데이터에 값을 뺍니다.

옵션 이름	옵션 기능
mul	모든 숫자 데이터에 값을 곱합니다.
div	모든 숫자 데이터에 값을 나눕니다.
over	모든 숫자 데이터에 최소값을 설정합니다.
under	모든 숫자 데이터의 최대값을 설정합니다.

- `mul` 을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
ADJUST {mul : 100}
```

- `over` 를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
ADJUST { key:[rate], over:30}
```

- `under` 를 설정하는 경우

```
ADJUST { key:[rate], under:30}
```

FILTER

특정 조건을 가지고 있는 데이터만 다음 단계로 전달합니다.

옵션 이름	옵션 기능
expr	조건을 수식으로 입력합니다.
value	특정 값을 가지고 있는 데이터를 찾습니다.
exist	값이 있는 데이터를 찾습니다.
notexist	값이 없는 데이터를 찾습니다.
over	특정 값보다 같거나 큰 데이터를 찾습니다.(greater 또는 equal to)
under	특정 값보다 같거나 작은 데이터를 찾습니다.(less 또는 equal to)

- `expr` 옵션을 적용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
FILTER {expr : "tx_count != 0"}
```

- `value` 옵션을 적용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
FILTER { key : tx_count, value : 5}
```

- `exist` 옵션을 적용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
FILTER { key : tx_count, exist : true}
```

- `notexist` 옵션을 적용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
FILTER { key : tx_count, notexist : true}
```

- `under` 옵션을 적용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
FILTER { key : tx_count, under : 6}
```

> - 데이터가 0인 경우도 데이터가 존재하는 경우입니다. `{exist: true}` 에 적용됩니다.
> - `{exist : false}` 와 `{notexist : false}` 는 불가능합니다. `{notexist : true}` 와 `{exist : true}` 를 사용해주세요.

용어 사전

용어 사전은 지속적으로 업데이트됩니다.

기본 용어

모니터링

모니터링(Monitoring)이란 어떤 대상을 감시, 관찰한다는 뜻입니다. IT 서비스 분야에서는 한정된 비용과 리소스를 가지고 서비스를 제공합니다. 때문에 모니터링을 통해 예기치 못한 상황과 오류를 대비하고 극복하는 것이 매우 중요합니다.

SaaS (Software as a Service)

SaaS는 고객을 대신하여 소프트웨어와 데이터를 제공하고 관리합니다. SaaS는 개별 컴퓨터에 응용 프로그램을 다운로드하고 설치할 필요가 없습니다. 고객은 유지 보수 및 자원을 간소화하면서 비즈니스에 집중할 수 있습니다.

SaaS를 통해 서비스를 공급하는 업체는 고객을 대신해 데이터, 미들웨어, 서버 및 스토리지와 같은 모든 잠재적인 기술적 문제를 관리합니다.

애플리케이션 (Application Performance Management)

'애플리케이션 성능 관리'를 의미합니다. 보통은 APM 서비스로 불립니다.

- A는 Application을 의미합니다.
- P는 Performance, 애플리케이션의 성능을 의미합니다. 그리고 애플리케이션의 성능은 웹서비스의 응답속도를 통해 측정하게 됩니다. 웹 서비스의 응답속도를 구하기 위해 APM 서비스는 트랜잭션을 추적하고 분석하는 일을 합니다.
- M은 Management 또는 Monitoring이 사용됩니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 애플리케이션 모니터링이란?

에이전트

와탭에서 에이전트란 모니터링 대상으로부터 수집한 데이터를 와탭 수집 서버로 전달하는 애플리케이션입니다. 에이전트는 웹 서버에 설치됩니다.

에이전트는 모니터링 대상과 1 대 1 대응됩니다. 모니터링 대상이 애플리케이션이라면, 애플리케이션이 실행될 때 에이전트도 함께 실행됩니다. 모니터링 대상이 서버라면, 서버에 에이전트를 설치하고 실행합니다.

> ⓘ 에이전트 설치에 관한 자세한 내용은 다음 문서를 참조하세요.
>
> - Java 에이전트 설치
> - Linux 에이전트 설치
> - MySQL 에이전트 설치

처리량

성능에 있어서 처리량은 '얼마나 많은 요청을 완료할 수 있는가'를 의미합니다. 이 말은 '시스템이 얼마나 많은 트랜잭션을 처리하는가'를 나타냅니다.

처리량은 초 단위 또는 분 단위로 측정합니다. 처리량은 요청량과 다릅니다. 처리량은 마무리된 요청의 양입니다. 초당 요청량(RPS)이 100이지만 초당 처리량(TPS)이 10이라면 90건 요청은 아직도 처리되지 못한 상태입니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 성능 테스트 필수 항목
> - 성능 - 처리량

클라우드 컴퓨팅

클라우드 컴퓨팅은 인터넷으로 가상화된 IT 리소스를 서비스로 제공하는 것을 의미합니다. 그리고 클라우드 컴퓨팅에서 가상화하여 서비스로 제공하는 대상은 서버, 플랫폼, 소프트웨어입니다. 사용자는 사용한 만큼만 비용을 지불하면 됩니다.

클라우드 서비스의 종류는 크게 3가지가 있습니다.

- Infrastructure as a Service(IaaS, 아이아스, 이에스)
- Platform as a Service(PaaS, 파스)
- Software as a Service(SaaS, 사스)

튜닝

시스템의 성능을 개선하는 것을 의미합니다. 튜닝에는 크게 처리량 튜닝과 안정성 튜닝이 있습니다.

- 처리량 튜닝은 최대 처리량을 늘려주는 튜닝입니다.
- 안정성 튜닝은 과부하 상태에서도 시스템이 다운되지 않도록 유지하는 튜닝입니다.

평균 응답 시간

애플리케이션 서버가 사용자에게 요청 결과를 반환하는 데 걸리는 시간입니다. 와탭의 서비스는 5초 간격으로 트랜잭션의 평균 응답 시간을 계산합니다. 평균 응답 시간은 튜닝 지표로서 의미를 가집니다.

대시보드

대시보드

대시보드를 이용해 시스템 전체 현황을 실시간으로 파악할 수 있습니다. 대시보드에서는 진행 중 트랜잭션 현황, 종료 트랜잭션의 응답시간 분포, 사용자 수, CPU, 메모리 추이 등의 주요 지표를 보여줍니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 애플리케이션 대시보드
> - 서버 리소스 보드
> - 서버 컴파운드 아이

동시 접속 사용자 (Realtime User)

실시간 브라우저 사용자 수를 보여줍니다. 10초마다 최근 5분 이내에 트랜잭션을 일으킨 사용자를 카운팅 하여 보여줍니다.

사용자는 브라우저의 IP를 기반으로 카운팅 합니다. 에이전트 설정에서 사용자를 구분하기 위해 IP를 사용하거나 쿠키를 사용할 수 있습니다.

DISK I/O

Disk I/O(%) 지표는 디스크 사용률을 보여줍니다. Disk I/O(%)가 80%를 넘으면 시스템 성능에 영향을 줄 수 있습니다.

기본 경고 값은 90%입니다. Disk I/O(%)가 100%라면 디스크가 쉬지 않고 일하고 있다는 의미입니다.

DISK IOPS (Input/Output Operations Per Second)

초당 입력과 출력 작업을 나타내는 측정 단위입니다. 작업은 KiB 단위로 측정됩니다.

기본 드라이브 기술에 따라 볼륨 유형이 단일 I/O로 계산하는 최대 데이터 용량이 결정됩니다. 일반적으로 HDD의 IOPS 범위는 55-180입니다. SSD의 IOPS는 3,000 ~ 40,000입니다.

리소스 보드 (Resource Board)

리소스 보드에서 하나의 프로젝트에 등록된 모든 서버를 모니터링할 수 있습니다. 프로젝트 내 모든 서버의 요약 정보와 실시간 자원 사용량의 변화를 확인할 수 있는 CPU Resource Map 맵을 제공합니다.

전체 자원의 규모, CPU, 메모리, 프로세스의 사용률 Top 5를 보여줍니다. 리소스 보드를 통해 장애 상황을 즉시 인지하고 대응할 수 있습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

리소스 이퀄라이저

CPU, Memory, Disk I/O, Disk IOPS의 항목에 대해 상위 5개의 서버 목록을 실시간으로 보여줍니다.

MSA 분석 (Microservices Architecture)

URL을 기준으로 각각의 서비스 간의 호출 관계를 비율로 보여줍니다.

메트릭스 차트 (Metrics Chart)

수집된 데이터를 시계열 차트로 보여줍니다. 시간대를 선택한 후 원하는 측정 항목을 고르면 결과를 보여줍니다.

CPU 리소스 맵 (CPU Resource Map)

전체 서버들의 CPU 사용량을 나타내는 분포도입니다. 10분 동안의 데이터를 보여주고 10초 주기로 갱신됩니다.

Apdex (Application Performance Index)

Apdex는 애플리케이션 성능 지표를 의미합니다. Apdex는 응답 시간에 기반하며 전체 요청 중 만족과 허용건 비율로 수치화합니다. Apdex는 사용자 만족도에 대한 지표로 활용할 수 있으며, 0~1 사이의 값을 갖습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 성능 카운터
> - 애플리케이션 성능 지표

성능 추이

지정한 시간에 해당하는 성능에 영향을 끼치는 정보들을 조회할 수 있습니다. 또한 전체, 개별 애플리케이션 서버 별로 그래프를 확인할 수 있습니다. 성능에 많은 영향을 끼치는 애플리케이션 서버가 무엇인지 파악 가능합니다.

성능 추이에서 조회할 수 있는 정보는 다음과 같습니다.

- 실시간 사용자
- 트랜잭션/초(합계)
- 응답시간
- CPU
- 힙 메모리
- 액티브 TX
- 많이 처리된 트랜잭션 Top 10
- HTTP Call Top 10
- SQL Top 10

애플리케이션 토폴로지

프로젝트 범위에서 포함된 모든 애플리케이션의 관계 정보를 표현합니다.

액티브 스테이터스 (Active Status)

액티브 트랜잭션들을 각 상태별로 갯수를 보여줍니다.

- METHOD : 메소스 수행중인 상태
- SQL : SQL을 수행중인 상태
- HTTPC : 외부 API 호출 상태
- DBC : 트랜잭션이 Connection Pool로 부터 새로운 Connection을 획득(get)하려는 상태
- SOCKET : 외부로 TCP Socket을 연결 중인 상태

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

컴파운드 아이

와탭 에이전트가 설치된 각각의 서버를 하나의 눈(Eye)으로 표현합니다. 컴파운드 아이는 서버들을 한곳에 모아서 보여줍니다.

총 5가지의 정보를 제공합니다.

- CPU 사용량
- Memory 사용량
- Disk 사용량
- 네트워크의 Rx (수신량)
- 네트워크의 Tx (송신량)

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 컴파운드 아이

큐브

5분 단위로 만든 성능 통계를 큐브라고 부릅니다. 큐브 분석은 큐브에 저장된 5분 단위 성능 데이터를 활용한 분석 기능입니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 큐브
> - 큐브 데이터 저장소

트랜잭션 맵

종료된 개별 트랜잭션의 응답 시간 분포도입니다. 히트맵과 동일하게 분포 패턴에 따른 문제점을 발견하고 분석할 수 있습니다.

히트맵은 5분 시간 단위로 트랜잭션을 그룹화해서 보여주지만, 트랜잭션 맵은 트랜잭션을 개별로 표시합니다.

플렉스 보드

사용자가 원하는 방식으로 커스터마이즈 할 수 있는 대시보드입니다. 애플리케이션, 서버, 데이터베이스, 컨테이너 등 와탭 프로젝트의 데이터를 화면에 자유자재로 배치할 수 있습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

히트맵

응답 시간 분포 차트입니다. 특정 기간의 응답 시간 분포를 한눈에 파악할 수 있습니다.

트랜잭션 발생 위치에 따라 느린 트랜잭션을 쉽게 찾아낼 수 있습니다. 색상을 통하여 에러 발생 여부에 대해서도 빠르게 찾아낼 수 있습니다.
X축은 트랜잭션의 종료 시간, Y축은 응답 시간을 의미합니다. 히트맵 상의 특정 영역을 드래그하면 트랜잭션 목록을 보여주는 화면으로 이동합니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

힙 메모리 (Heap Memory)

JVM(자바 가상머신)은 프로그램을 실행하기 위해 메모리에 데이터 저장 공간을 할당합니다.

메모리 공간은 크게 3가지 영역으로 분류됩니다. Static, Stack, Heap 영역입니다. 객체(인스턴스), 배열 등 주요 데이터는 Heap 메모리 영역에 저장됩니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 대시보드 위젯 힙 메모리
> - 힙 메모리

로그

로그(Log)

로그는 애플리케이션 실행 중 발생하는 이벤트와 메시지 등을 기록한 파일입니다.

애플리케이션 활동과 발생한 이슈의 원인을 이해하려면 반드시 로그 파일을 들여다봐야 합니다.

로그 모니터링 (Log Monitoring)

와탭의 로그 모니터링을 통해서 실시간으로 로그를 조회할 수 있습니다. 혹은 특정 시간, 카테고리, 태그 또는 필터를 적용하여 원하는 로그만 선택적으로 볼 수도 있습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

서버

수집 서버 (Repository Server)

프로젝트 생성시 모니터링 데이터를 수집하는 서버를 의미합니다. SaaS형 서비스 사용시 와탭의 수집 서버를 사용하므로 별도로 수집 서버를 구축할 필요가 없습니다.

Redis 서버

인-메모리 방식의 데이터 저장소입니다. 와탭에선 HTTP 세션을 담는 세션 저장소로 쓰고 있습니다.

스택

스레드(Thread)

프로세스 내에서의 실행 단위입니다. 모든 프로세스에는 한 개 이상의 스레드가 존재하여 작업을 수행합니다.

그리고 두 개 이상의 스레드를 가지는 프로세스를 멀티스레드 프로세스(multi-thread process)라고 합니다. 각 스레드는 자신의 스택과 레지스터를 가지고 있습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

탑 스택 (Top Stack)

탑 스택은 트랜잭션의 스택 정보를 수집하여 실행중인 메소드의 사용량 통계를 제공합니다.

스택의 최상단의 사용량 통계를 통해 서비스에 가장 많은 영향을 주는 메소드를 확인합니다. 메소드의 호출 빈도를 파악하면 CPU 또는 메모리에 부하가 걸리는 원인을 분석할 수 있습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

유니크 스택 (Unique Stack)

실행된 메소드의 집합이 동일한 경우에 대한 통계 정보입니다. 유니크 스택을 통해 많이 사용되는 스택의 정보를 알 수 있습니다.

유니크 스택에 많이 노출되었다면 통계적으로 빈번한 호출 또는 오랜 시간 동작하는 메소드입니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

액티브 스택 (Active Stack)

실행 중인 트랜잭션의 스택 정보를 수집하는 기능입니다. 스택 정보는 10초마다 수집됩니다. 수집된 데이터는 통계로 확인할 수 있습니다.

통계 정보는 긴 시간 실행되는 메소드, 짧은 시간 수행되지만 잦은 빈도로 실행되는 메소드 모두 비율을 통해 식별할 수 있습니다. 트랜잭션이 진행 중 메소드 레벨에서 어느 부분이 지연이 되었는지 확인 가능합니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

알림

알림 (Alarm)

모니터링 프로젝트에서 문제 상황이 발생하면 다양한 채널을 통해 사용자에게 실시간으로 알림이 갑니다.

알림 서비스는 모니터링하려는 IT 시스템 특성에 맞게 사전 정의된 알림 규칙을 제공합니다. 필요 시 사용자는 구체적인 알림 조건을 설정할 수 있습니다.

> ⓘ • 경고 알림 설정하기

트랜잭션

트랜잭션 (Transaction)

애플리케이션 모니터링에 있어 트랜잭션이란, 애플리케이션에 유입된 단일 요청(request)이 애플리케이션의 처리 과정을 거쳐 응답(response)이 반환되기까지의 과정을 일컫습니다.

TPS (Transactions Per Second)

초당 처리된 트랜잭션 건수를 의미합니다. 서비스 성능지표 중 기준이 됩니다. 와탭은 프로젝트 전체 TPS를 실시간으로 보여줍니다.

트랜잭션 트레이스

단일 트랜잭션의 수행과정에서의 일련의 처리 과정을 의미합니다.

액티브 트랜잭션 (Active Transaction)

진행 중인 트랜잭션을 의미합니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 액티브 트랜잭션
> - 액티브 트랜잭션이란
> - 장애를 가장 빠르게 알아내는 액티브 트랜잭션

멀티 트랜잭션

MSA와 같이 여러 애플리케이션의 호출 관계를 추적해야할 경우에 어느 부분에서 문제가 발생했고, 개선이 필요한지 식별할 수 있는 기능입니다.

Caller와 Callee

- **Caller**: 서비스를 호출한 트랜잭션(호출자)을 의미합니다.
- **Callee**: 서비스를 호출 당한 트랜잭션(피호출자)을 의미합니다.

기타

Okind

에이전트가 어떤 용도로 쓰이는지 종류를 구분하는데 쓰입니다. 예를 들어 다음과 같은 경우 해당 에이전트가 mobile_ui 용이라는 의미입니다.

```
-Dwhatap.okind=mobile_ui
```

CPU Steal Time

CPU Steal Time은 하이퍼바이저가 다른 가상 프로세서를 서비스하는 동안 가상 CPU가 실제 CPU를 기다리는 시간을 백분율로 표시한 값입니다.

가상 환경에서 동작하는 VM(Virtual Machine)은 단일 호스트에 있는 다른 인스턴스와 리소스를 공유합니다.

CPU Steal Time을 통해 VM에서 동작하는 CPU가 물리 머신으로부터 자원을 할당받기 위해 얼마나 대기하고 있는지 알 수 있습니다.