7주차

13. 파티션

파티션 기능은 테이블을 논리적으로는 하나의 테이블이지만 물리적으로 여러 개의 테이블로 분리해 관리할 수 있도록 해준다. 파티션은 주로 대용량 테이블에 사용하지만 어떤 쿼리를 사용하느냐에 따라 성능이 오히려 나빠질 수 있다.

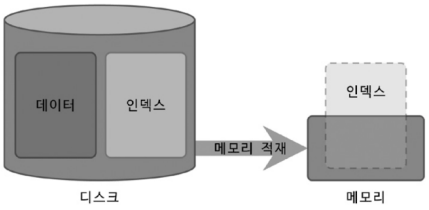
▼ 1. 개요

1.1 파티션을 사용하는 이유

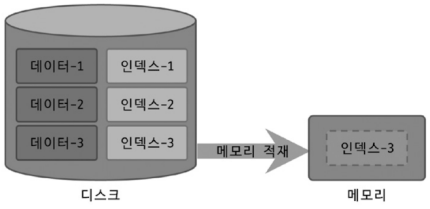
테이블이 너무 커져 인덱스 크기가 물리적 메모리보다 크거나 데이터 특성상 주기적인 삭제 작업이 필요한 경우 등이 파티션이 필요한 대표적인 예시이다.

1. 단일 INSERT와 단일 또는 범위 SELECT의 빠른 처리

인덱스는 일반적으로 `SELECT`를 위한 것이지만 `UPDATE`, `DELETE` 쿼리를 위해 필요할 때도 많다. 하지만 인덱스가 커질수록 쿼리의 작업이 느려질 것이다. 특히 인덱스의 **워킹 셋(Working Set)**의 크기가 물리적으로 MySQL이 사용 가능한 메모리 공간보다 크다면 더 심각할 것이다.



파티션되지 않은 테이블의 메모리 적재



파티션된 테이블의 메모리 적재

위 그림을 보면 파티션은 데이터와 인덱스를 조각화해서 물리적 메모리를 효율적으로 사용할 수 있게 해준다.

> 일반적으로 테이블의 전체 데이터에서 20 ~ 30%의 데이터만 활발히 사용되는데, 이 데이터를 워킹 셋(Working Set)이라고 표현한다. 테이블 데이터를 활발하게 사용되는 워킹 셋과 그렇지 않은 부분으로 나누어 파티션 한다면 효과적으로 성능을 개선할 수 있을 것이다.

2. 데이터의 물리적 저장소 분리

MySQL에서는 테이블의 파티션 단위로 인덱스를 생성하거나 파티션별로 다른 인덱스를 가지는 형태는 지원하지 않는다

3. 이력 데이터의 효율적 관리

로그 데이터의 경우 단기간에 대량으로 누적되는 동시에 일정 기간이 지나면 삭제하는 것이 일반적이다. 이때 파티션을 이용하면 효율적으로 관리할 수 있다.

1.2 MySQL 파티션의 내부 처리

```
CREATE TABLE tb_article
(
    article_id INT         NOT NULL,
    reg_userid VARCHAR(10) NOT NULL,
    reg_date   DATETIME    NOT NULL,
    PRIMARY KEY (article_id, reg_date)
) PARTITION BY RANGE (YEAR(reg_date))(
    PARTITION p2009 VALUES LESS THAN (2010),
    PARTITION p2010 VALUES LESS THAN (2011),
    PARTITION p2011 VALUES LESS THAN (2012),
    PARTITION p9999 VALUES LESS THAN MAXVALUE
    );
```

`reg_date`에서 연도 부분을 파티션 키로 지정

1. 파티션 테이블의 레코드 INSERT

`INSERT` 쿼리가 실행되면 파티션 키인 `reg_date` 컬럼값을 이용해 파티션 표현식을 평가하고 그 결과로 레코드가 저장될 파티션을 결정한다. 파티션이 결정되면 나머지 과정은 일반 테이블과 동일하게 처리된다.

2. 파티션 테이블의 레코드 UPDATE

`UPDATE` 쿼리의 `WHERE` 조건에 파티션 키 컬럼이 조건으로 존재하면 그 값을 이용해 레코드가 저장된 파티션에서 빠르게 대상 레코드를 검색할 수 있다. 그렇지 않으면 모든 파티션을 검색해야 한다.

- 파티션 키 이외의 컬럼만 변경될 때는 파티션이 적용되지 않은 일반 테이블과 마찬가지로 컬럼 값만 변경한다.
- 파티션 키 컬럼이 변경될 때는 기존 레코드가 저장된 파티션에서 해당 레코드를 제거하고 새로운 파티션을 결정하여 레코드를 새로 저장한다.

3. 파티션 테이블의 검색

파티션 테이블 검색 성능에 영향을 미치는 조건

- `WHERE` 절의 조건으로 검색해야 할 파티션을 선택할 수 있는가?
- `WHERE` 절의 조건이 인덱스를 효율적으로 사용(인덱스 레인지 스캔)할 수 있는가?

파티션 선택은 가능하지만 효율적인 인덱스 사용이 불가한 경우, 또는 파티션 선택과 효율적 인덱스 사용 모두 불가능한 경우는 최대한 피하자.

4. 파티션 테이블의 인덱스 스캔과 정렬

MySQL의 파티션 테이블에서 인덱스는 전부 로컬 인덱스에 해당한다. 즉 모든 인덱스는 파티션 단위로 생성(개별 파티션별로 서로 다른 인덱스를 생성할 수 있다는 의미가 아님!)되며 테이블 전체 단위로 글로벌하게 통합된 인덱스는 지원하지 않는다.

MySQL 서버는 여러 파티션에 대해 인덱스 스캔을 수행할 때 각 파티션으로부터 조건에 일치하는 레코드를 정렬된 순서대로 읽으며 우선순위 큐(Priority Queue)에 임시로 저장한다. 그리고 우선순위 큐에서 다시 필요한 순서(인덱스 정렬 순서)대로 데이터를 가져간다.

5. 파티션 프루닝

최적화 단계에서 필요한 파티션만 골라내고 불필요한 것들은 실행 계획에서 배제하는 것을 파티션 프루닝(Partition pruning)이라고 한다. `EXPLAIN` 명령어 결과의 `partitions` 컬럼에 어떤 파티션만 조회하였는지 확인할 수 있다.

▼ 2. 주의사항

2.1 파티션의 제약 사항

- 스토어드 루틴이나 UDF, 사용자 변수 등을 파티션 표현식에 사용할 수 없다.
- MySQL 내장 함수 중 일부는 파티션 표현식으로 사용할 때 파티션 프루닝을 지원하지 않을 수 있다.
- PK를 포함한 테이블의 모든 유니크 인덱스는 파티션 키 컬럼을 포함해야 한다.
- 파티션된 테이블의 인덱스는 모두 로컬 인덱스이며, 동일 테이블에 소속된 모든 파티션은 같은 구조의 인덱스만 가질 수 있다. 또한 파티션 개별로 인덱스를 변경하거나 추가할 수 없다.
- 동일 테이블에 속한 모든 파티션은 동일 스토리지 엔진만 가질 수 있다.
- 최대 8192개의 파티션을 가질 수 있다.(서브 파티션 포함)
- 파티션 생성 이후 `sql_mode` 시스템 변수 변경은 데이터 파티션의 일관성을 깨뜨릴 수 있다.
- 파티션 테이블에서는 외래키를 사용할 수 없다.
- 파티션 테이블은 전문 검색 인덱스 생성이나 전문 검색 쿼리를 사용할 수 없다.
- 공간 데이터 컬럼 타입은 사용할 수 없다.
- 임시 테이블(Temporary table)은 파티션 기능을 사용할 수 없다.

> MySQL :: MySQL 8.0 Reference Manual :: 26.6.3 Partitioning Limitations Relating to Functions
> This section discusses limitations in MySQL Partitioning relating specifically to functions used in partitioning expressions.
> https://dev.mysql.com/doc/refman/8.0/en/partitioning-limitations-functions.html

표현식에 사용 가능한 내장 함수

2.2 파티션 사용 시 주의사항

1. 파티션과 유니크 키(PK 포함)

종류와 관계없이 테이블에 유니크 인덱스(PK 포함)가 있으면 파티션 키는 모든 유니크 인덱스의 일부 또는 모든 컬럼을 포함해야 한다. 각 유니크 키에 대해 값이 주어졌을 때 해당 레코드가 어느 파티션에 저장되어 있는지 계산할 수 있어야 한다.

```
CREATE TABLE tb_partition // 1
(
    fd1 INT NOT NULL,
    fd2 INT NOT NULL,
    fd3 INT NOT NULL,
    UNIQUE KEY (fd1, fd2)
) PARTITION BY HASH ( fd3 )
    PARTITIONS 4;

CREATE TABLE tb_partition // 2
(
    fd1 INT NOT NULL,
    fd2 INT NOT NULL,
    fd3 INT NOT NULL,
    UNIQUE KEY (fd1),
    UNIQUE KEY (fd2)
) PARTITION BY HASH ( fd1 + fd2 )
    PARTITIONS 4;

CREATE TABLE tb_partition // 3
(
    fd1 INT NOT NULL,
    fd2 INT NOT NULL,
    fd3 INT NOT NULL,
    PRIMARY KEY (fd1),
    UNIQUE KEY (fd2, fd3)
) PARTITION BY HASH ( fd1 + fd2 )
    PARTITIONS 4;
```

1. 유니크 키와 파티션 키가 전혀 연관이 없다.
2. 첫 번째 유니크 키 컬럼인 `fd1` 만으로 파티션 결정이 되지 않는다.(`fd2` 컬럼도 같이 있어야 파티션의 위치를 판단할 수 있다.) 두 번째 유니크 키 또한 첫 번째와 같은 이유로 불가능하다.
3. PK 컬럼인 `fd1` 값만으로 파티션 판단이 되지 않으며, 유니크 키인 `fd2`와 `fd3`으로도 파티션 위치를 결정할 수 없다.

```
CREATE TABLE tb_partition
(
    fd1 INT NOT NULL,
```

```
    fd2 INT NOT NULL,
    fd3 INT NOT NULL,
    UNIQUE KEY (fd1, fd2, fd3)
) PARTITION BY HASH ( fd1 )
    PARTITIONS 4;

CREATE TABLE tb_partition
(
    fd1 INT NOT NULL,
    fd2 INT NOT NULL,
    fd3 INT NOT NULL,
    UNIQUE KEY (fd1, fd2)
) PARTITION BY HASH ( fd1 + fd2)
    PARTITIONS 4;

CREATE TABLE tb_partition
(
    fd1 INT NOT NULL,
    fd2 INT NOT NULL,
    fd3 INT NOT NULL,
    UNIQUE KEY (fd1, fd2, fd3),
    UNIQUE KEY (fd3)
) PARTITION BY HASH ( fd3 )
    PARTITIONS 4;
```

2. 파티션과 open_files_limit 시스템 변수

`open_files_limit` 시스템 변수는 MySQL 서버에서 동시에 오픈할 수 있는 파일 개수를 설정할 수 있다. 파티션되지 않은 일반 테이블은 보통 1개당 오픈된 파일의 개수가 2~3개 수준이지만 파티션 테이블에서는 `파티션 개수 * 2~3`개가 된다.

파티션 프루닝으로 최적화된다고 해도 일단 동시에 모든 파티션의 데이터 파일을 오픈해야 한다. 그래서 파티션을 많이 사용하는 경우에는 `open_files_limit` 시스템 변수 값을 적절히 높은 값으로 설정할 필요가 있다.

▼ 3. MySQL 파티션 종류

- 레인지 파티션
- 리스트 파티션
- 해시 파티션
- 키 파티션

해시와 키 파티션에 대해서는 리니어(Linear) 파티션과 같은 추가적인 기법도 제공한다.

3.1 레인지 파티션

파티션 키의 연속된 범위로 파티션을 정의하는 방법

가장 자주 사용되는 파티션 방법 중 하나로, 다른 파티션 방법과 달리 `MAXVALUE` 키워드를 이용해 명시되지 않은 범위의 키 값이 담긴 레코드를 저장하는 파티션을 정의할 수 있다.

1. 레인지 파티션의 용도

- 날짜를 기반으로 데이터가 누적되고 년, 월, 일 단위로 분석하고 삭제해야 하는 경우
- 범위 기반으로 데이터를 여러 파티션에 균등하게 나눌 수 있을 때
- 파티션 키 위주로 검색이 자주 실행될 때

데이터베이스에서 파티션의 장점

- 큰 테이블을 작은 크기의 파티션으로 분리
- 필요한 파티션만 접근(쓰기•읽기 모두)

두 가지 장점을 모두 취하는 것은 어렵지만 이력을 저장하는 테이블에서 레인지 파티션은 두 가지 장점을 모두 취할 수 있다.

2. 레인지 파티션 테이블 생성

```
CREATE TABLE employees
(
    id         INT  NOT NULL,
    first_name VARCHAR(30),
    last_name  VARCHAR(30),
    hired      DATE NOT NULL DEFAULT '1970-01-01',
    age        INT  NOT NULL
) PARTITION BY RANGE (YEAR(hired)) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1996),
    PARTITION p2 VALUES LESS THAN (2001),
    PARTITION p3 VALUES LESS THAN MAXVALUE
    );
```

- `PARTITION BY RANGE` 키워드로 레인지 파티션 정의
- `PARTITION BY RANGE` 뒤에 컬럼 또는 내장 함수를 이용해 파티션 키를 명시
- `VALUES LESS THAN`으로 명시된 값보다 작은 값만 해당 파티션에 저장하게 설정(단 LESS THAN 절에 명시된 값은 그 파티션에 포함되지 않음)
- `VALUES LESS THAN MAXVALUE`로 명시되지 않은 레코드를 저장할 파티션을 지정(선택 사항)
- `VALUES LESS THAN MAXVALUE`가 정의되지 않으면 최대 값을 넘어가는 경우 INSERT 시에 에러가 발생하며 'Table has no partition for value xxx' 메시지 표시
- 테이블과 각 파티션은 같은 스토리지 엔진으로 정의 가능

3. 레인지 파티션의 분리와 병합

단순 파티션 추가

```
ALTER TABLE employees
    ADD PARTITION (PARTITION p4 VALUES LESS THAN (2011));
```

만약 `LESS THAN MAXVALUE` 파티션을 가지고 있다면 에러가 발생할 것이다. 이런 경우 `ALTER TABLE … REORGANIZE PARTITION` 명령을 사용한다.

```
ALTER TABLE employees
    ALGORITHM = INPLACE,
    LOCK = SHARED,
    REORGANIZE PARTITION p3 INTO (
        PARTITION p3 VALUES LESS THAN (2011),
        PARTITION p4 VALUES LESS THAN MAXVALUE
        );
```

해당 명령어는 p3 파티션의 레코드를 모두 새로운 두 개의 파티션으로 복사하는 작업을 필요로 하기 때문에 레코가 매우 많으면 시간이 오래 걸릴 수 있다.

파티션 삭제

`DROP PARTITION` 키워드 사용

레인지 파티션을 삭제하는 작업이나 리스트 파티션 테이블에서 특정 파티션을 삭제하는 작업은 아주 빠르게 처리

```
ALTER TABLE employees
    DROP PARTITION p0;
```

> 레인지 파티션을 삭제할 때 항상 가장 오래된 파티션 순서로만 삭제가 가능하다. 레인지 파티션들 중에 중간에 있는 파티션을 먼저 삭제할 수 없다. 레인지 파티션은 가장 마지막 파티션만 새로 추가할 수 있고, 가장 오래된 파티션만 삭제할 수 있다. ??? 테스트 해보니까 왜 삭제가 되죠...

기존 파티션의 분리

`REORGANIZE PARTITION` 명령어 사용 ('단순 파티션 추가' 챕터 확인)

파티션 재구성(`REORGANIZE PARTITION`) 명령은 `INPLACE` 알고리즘을 사용할 수 있지만 최소한 읽기 잠금(Shared Lock)이 필요하다. 즉, 파티션이 재구성되는 동안은 테이블의 쓰기가 불가능해지므로 파티션 재구성 작업은 서비스 점검 시간대나 쿼리 처리가 많지 않은 시간대에 진행하는 것이 좋다.

기존 파티션의 병합

`REORGANIZE PARTITION` 키워드 사용

```
ALTER TABLE employees
    ALGORITHM = INPLACE,
    LOCK = SHARED,
```

```
    REORGANIZE PARTITION p2, p3 INTO (
        PARTITION p23 VALUES LESS THAN (2011)
        );
```

3.2 리스트 파티션

레인지 파티션과 많은 부분에서 흡사하게 동작하지만 가장 큰 차이는 레인지 파티션은 파티션 키 값의 범위로 파티션을 구성할 수 있지만 리스트 파티션은 파티션 키 값 하나하나를 리스트로 나열해야 한다. 또한 `MAXVALUE` 파티션을 정의할 수 없다.

1. 리스트 파티션의 용도

- 파티션 키 값이 코드 값이나 카테고리와 같이 고정적일 때
- 키 값이 연속되지 않고 정렬 순서와 관계없이 파티션을 해야 할 때
- 파티션 키 값을 기준으로 레코드의 건수가 균일하고 검색 조건에 파티션 키가 자주 사용될 때

2. 리스트 파티션 테이블 생성

```
CREATE TABLE product
(
    id          INT NOT NULL,
    name        VARCHAR(30),
    category_id INT NOT NULL,
    price       INT NOT NULL
) PARTITION BY LIST (category_id) (
    PARTITION p_appliance VALUES IN (3),
    PARTITION p_computer VALUES IN (1, 9),
    PARTITION p_sports VALUES IN (2, 6, 7),
    PARTITION p_etc VALUES IN (4, 5, 8, NULL)
    );
```

- `PARTITION BY LIST` 키워드로 리스트 파티션 정의
- `PARTITION BY LIST` 뒤에 파티션 키를 정의
- `VALUES IN (…)`으로 파티션별 저장할 파티션 키 값 목록 나열
- 파티션 키 값으로 `NULL` 명시 가능
- 나머지 모든 값을 저장하는 `MAXVALUE` 파티션 정의 불가

정수 타입뿐 아니라 문자열 타입도 가능

3. 리스트 파티션의 분리와 병합

`VALUES LESS THAN` 대신 `VALUES IN`을 사용한다는 것 외에는 레인지 파티션의 추가 및 삭제, 병합 작업이 모두 동일하다.

4. 리스트 파티션 주의사항

- 명시되지 않은 나머지 값을 저장하는 `MAXVALUE` 파티션을 정의할 수 없다.
- 레인지 파티션과 달리 `NULL`을 저장하는 파티션을 별도로 생성할 수 있다.

3.3 해시 파티션

MySQL에서 정의한 해시 함수에 의해 레코드가 저장될 파티션을 결정하는 방법

MySQL에서 정의한 해시 함수는 복잡한 알고리즘이 아닌 파티션 표현식의 결과값을 파티션의 개수로 나눈 나머지로 저장될 파티션을 결정하는 방식이다. 해시 파티션의 키는 항상 정수 타입의 컬럼이나 정수를 반환하는 표현식만 사용 가능하다. 파티션의 개수는 레코드를 각 파티션에 할당하는 알고리즘과 연관되기 때문에 파티션을 추가하거나 삭제하는 작업에는 테이블 전체 레코드를 재분배하는 작업이 필요하다.

1. 해시 파티션 용도

- 레인지 파티션이나 리스트 파티션으로 데이터를 균등하게 나누는 것이 어려울 때
- 테이블의 모든 레코드가 비슷한 사용 빈도를 보이지만 테이블이 너무 커서 파티션을 적용해야 할 때

2. 해시 파티션 테이블 생성

```
CREATE TABLE employees
(
    id         INT  NOT NULL,
    first_name VARCHAR(30),
    last_name  VARCHAR(30),
    hired      DATE NOT NULL DEFAULT '1970-01-01',
    age        INT  NOT NULL
) PARTITION BY HASH ( id )
    PARTITIONS 4;

// ------------- 파티션 이름 별도 지정 -------------

CREATE TABLE employees
(
    id         INT  NOT NULL,
    first_name VARCHAR(30),
    last_name  VARCHAR(30),
    hired      DATE NOT NULL DEFAULT '1970-01-01',
    age        INT  NOT NULL
) PARTITION BY HASH ( id )
    PARTITIONS 4 (
    PARTITION p0 ENGINE = InnoDB,
    PARTITION p1 ENGINE = InnoDB,
    PARTITION p2 ENGINE = InnoDB,
```

```
    PARTITION p3 ENGINE = InnoDB
    );
```

- `PARTITION BY HASH` 키워드로 해시 파티션 정의
- `PARTITION BY HASH` 뒤에 키를 명시
- 해시 파티션의 파티션 키 또는 파티션 표현식은 반드시 정수 타입의 값을 반환
- `PARTITIONS n`으로 생성할 파티션 개수 지정
- 직접 파티션 이름을 명시할 수 있지만 해시나 키 파티션에서는 특정 파티션을 삭제하거나 병합하는 작업이 거의 불필요하기 때문에 크게 의미는 없다. 파티션 개수만 지정하면 파티션 이름은 기본적으로 'p0, p1, p2, ...'과 같은 규칙으로 생성

3. 해시 파티션 분리와 병합

파티션 개수가 변경되는 것은 해시 알고리즘을 변경하는 것이므로 전체 파티션이 영향을 받는다.

해시 파티션 추가

해시 파티션을 추가할 때는 별도의 영역이나 범위는 명시하지 않고 몇 개의 파티션을 더 추가할 것인지만 지정하면 된다.

```
ALTER TABLE employees
    ALGORITHM = INPLACE,
    LOCK = SHARED,
    ADD PARTITION (PARTITION p5 ENGINE = InnoDB);

ALTER TABLE employees
    ALGORITHM = INPLACE,
    LOCK = SHARED,
    ADD PARTITION PARTITIONS 3;
```

`INPLACE` 알고리즘으로 실행된다고 해도 레코드 리빌드 작업이 필요하며 테이블에 대한 읽기 잠금이 필요하다.

해시 파티션 삭제

해시나 키 파티션은 파티션 단위로 레코드를 삭제하는 방법이 없다.

해시 파티션 분할

해시나 키 파티션은 분할이 불가능하다. 단지 파티션 개수를 늘리는 것만 가능하다.

해시 파티션 병합

해시나 키 파티션은 병합이 불가능하다. 단지 파티션 개수를 줄이는 것만 가능하다.

`COALESCE PARTITION` 명령을 사용해 개수를 줄일 수 있다.

```
ALTER TABLE employees
    ALGORITHM = INPLACE,
```

```
    LOCK = SHARED,
    COALESCE PARTITION 10; // 줄이고자 하는 파티션 개수 (16개 - 10개 => 6개)
```

해시 파티션 주의사항

- 특정 파티션만 삭제(`DROP PARTITION`) 불가능
- 새로운 파티션을 추가하는 작업은 기존 모든 데이터의 재배치 작업이 필요
- 레인지나 리스트 파티션과는 상당히 다른 방식으로 관리되기 때문에 해시 파티션이 용도에 적합한 해결책인지 확인이 필요
- 일반적으로 사용자들에게 익숙한 파티션의 조작이나 특성은 대부분 레인지나 리스트 파티션에만 해당하는 것이 많음

3.4 키 파티션

해시 파티션과 사용법과 특성이 거의 같지만, 해시 파티션과 다르게 해시 값의 계산도 MySQL 서버가 수생한다. 키 파티션에서는 정수 타입이나 정수값을 반환하는 표현식뿐만 아니라 대부분의 데이터 타입에 대해 파티션 키를 적용할 수 있다. 선정된 파티션 키의 값을 `MD5()` 함수를 이용해 해시값을 계산하고, 그 값을 `MOD` 연산하여 데이터를 파티션에 분배한다.

1. 키 파티션 생성

```
CREATE TABLE k1
(
    id   INT NOT NULL,
    name VARCHAR(20),
    PRIMARY KEY (id)
) PARTITION BY KEY ()
    PARTITIONS 2;

// -------------------------

CREATE TABLE k1
(
    id   INT NOT NULL,
    name VARCHAR(20),
    PRIMARY KEY (id, name)
) PARTITION BY KEY (id)
    PARTITIONS 2;
```

- PK가 있는 경우 자동으로 PK가 파티션 키로 설정
- PK가 없는 경우 유니크 키가 존재한다면 유니크 키가 파티션 키로 설정
- PK나 유니크 키의 컬럼 일부를 파티션 키로 명시 설정(PK나 유니크 키를 구성하는 컬럼 중 일부만 선택해 파티션 키로 설정 가능)

파티션 생성

- `PARTITION BY KEY` 키워드로 키 파티션 정의
- `PARTITION BY KEY` 뒤에 파티션 키 컬럼을 명시
- 파티션 키를 아무 컬럼도 명시하지 않으면 자동으로 PK나 유니크 키의 모든 컬럼을 파티션 키로 선택(일부만 파티션 키로 명시하는 것도 가능)
- `PARTITIONS n`으로 생성할 파티션 개수 지정

2. 키 파티션 주의사항 및 특이사항

- 키 파티션은 MySQL 서버가 내부적으로 `MD5()` 함수를 이용해 파티션하기 때문에 파티션 키가 반드시 정수 타입이 아니어도 된다. 해시 파티션으로 파티션이 어려우면 키 파티션 적용 고려
- PK나 유니크 키를 구성하는 컬럼 중 일부만으로도 파티션 가능
- 유니크 키를 파티션 키로 사용할 때 해당 유니크 키는 반드시 `NOT NULL`이어야 한다.
- 해시 파티션에 비해 파티션 간의 레코드를 더 균등하게 분할할 수 있어 키 파티션이 더 효율적이다.

3.5 리니어 해시 파티션/리니어 키 파티션

Power-fo-two(2의 승수) 알고리즘을 사용해 새로운 파티션을 추가하거나 파티션을 통합할 때 전체 파티션 레코드 재분배 작업이 발생하는 해시 파티션이나 키 파티션의 단점을 최소화하는 파티션

파티션 추가•통합 시 다른 파티션에 미치는 영향을 최소화

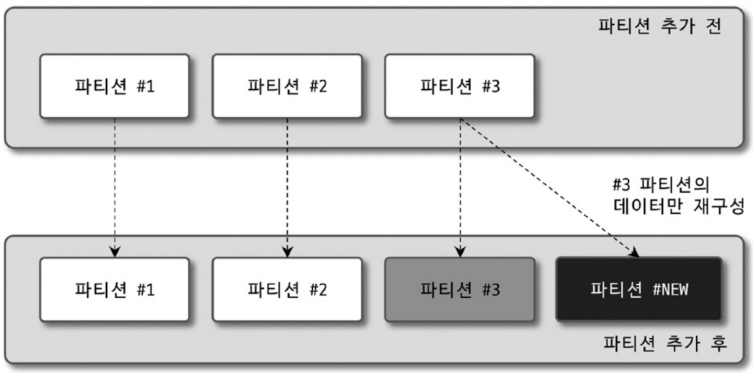
1. 리니어 해시 파티션/리니어 키 파티션 추가 및 통합

나머지 연산으로 레코드가 저장될 파티션을 결정하는 것이 아닌 Power-of-two 분배 방식을 사용하기 때문에 파티션의 추가나 통합 시 특정 파티션의 데이터에 대해서만 이동 작업을 하면 된다.

파티션 추가

추가 명령어는 일반 해시 파티션이나 키 파티션과 동일하다.

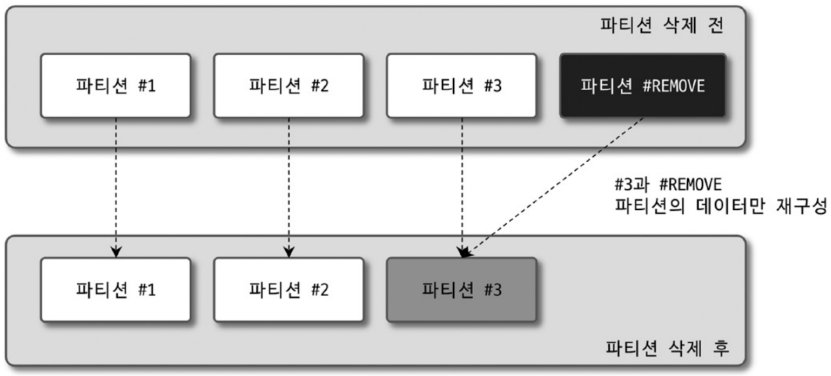
새로운 파티션을 추가할 때도 특정 파티션의 레코드만 재분배하기 때문에 해시 파티션이나 키 파티션 추가보다 매우 빠르게 처리할 수 있다.



리니어 해시 파티션/리니어 키 파티션 추가

파티션 통합

파티션 추가와 마찬가지로 일부 파티션에 대해서만 레코드 통합 작업만 하면 된다.



리니어 해시 파티션/리니어 키 파티션 통합

리니어 해시 파티션/리니어 키 파티션 주의사항

파티션 추가•통합 시 작업의 범위를 최소화하는 대신 각 파티션이 가지는 레코드의 건수는 일반 해시 파티션이나 키 파티션에 비해 덜 균등할 수 있다. 새로운 파티션을 추가하거나 삭제해야 할 요건이 많다면 리니어 해시 파티션 또는 리니어 키 파티션을 적용하는 것이 좋다.

3.6 파티션 테이블의 쿼리 성능

파티션 테이블에 쿼리가 실행될 때 테이블의 모든 파티션을 읽을지 아니면 일부 파티션만 읽을지는 성능에 아주 큰 영향을 미친다. 쿼리 성능은 테이블에서 얼마나 많은 파티션을 프루닝할 수 있는지가 관건이다.

파티션을 사용할 때는 반드시 파티션 프루닝이 얼마나 도움이 될지를 먼저 예측해보고 응용 프로그램에 적용하자.