**3. 파티션을 활용한 DML 튜닝**

**3.1** 테이블 파티션

- 파티셔닝은 테이블 또는 인덱스 데이터를 특정 컬럼 값에 따라 별도 세그먼트에 나눠서 저장하는

것

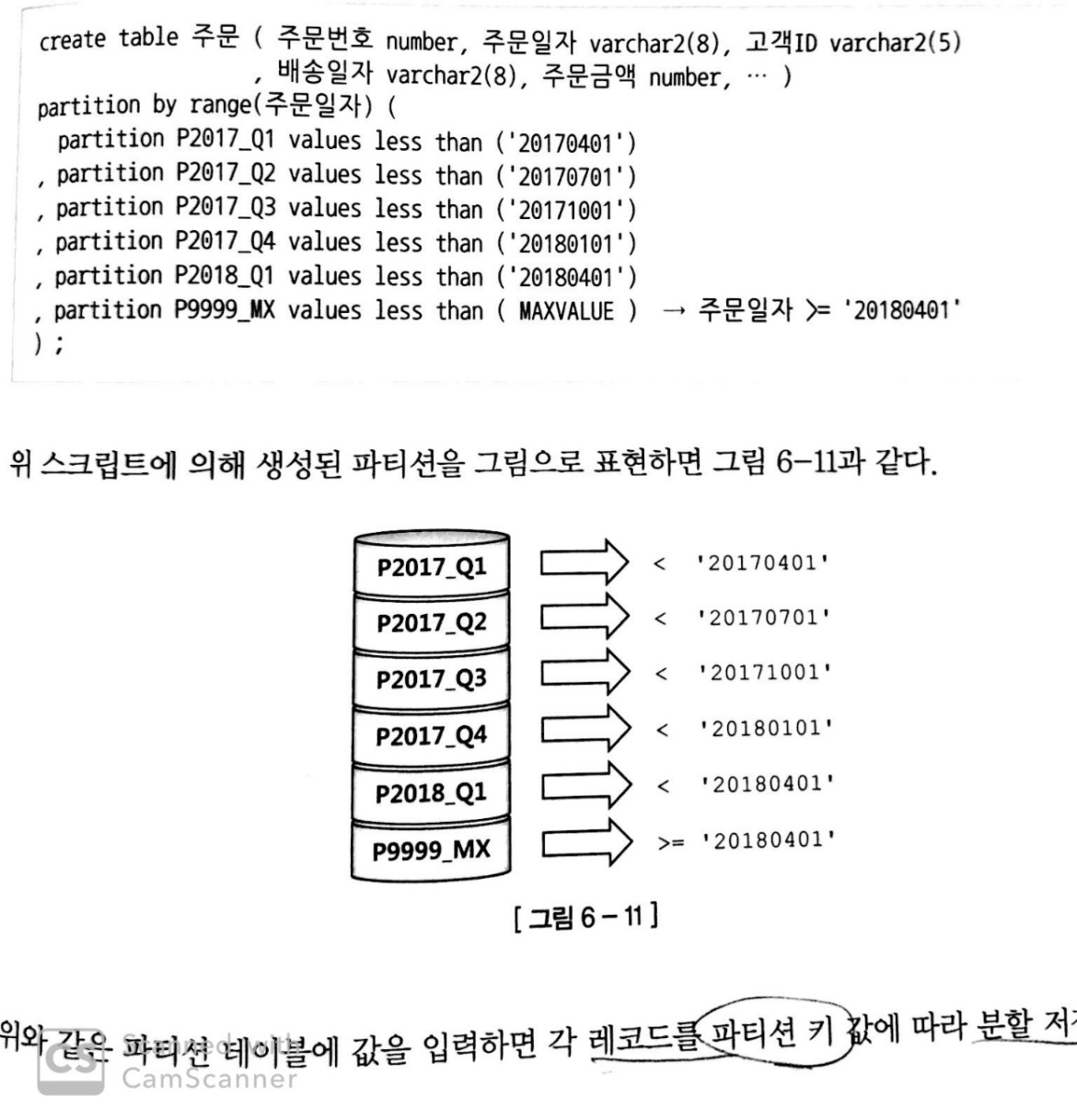
- 파티션이 필요한 이유

관리적 측면 : 파티션 단위 백업, 추가, 삭제, 변경 => 가용성 향상

성능적 측면 : 파티션 단위 조회 및 DML, 경합 또는 부하 분산

3.1.1 Range 파티션

- 가장 기초적인 방식으로 주로 날짜 컬럼을 기준으로 파티셔닝

 ex) 주문 테이블을 주문일자 기준으로 분기별로 Range 파티셔닝

-> 테이블에 값을 입력하면 파티션 키(주문일자) 값에 따라 분할 저장한다.

- 파티션 테이블에 대한 SQL 성능 향상 원리는 파티션 Pruning에 있다.

- 파티션 Pruning은 SQL 하드파싱이나 실행 지점에 조건절을 분석해서 읽지 않아도 되는 파티션

세그먼트를 액세스 대상에서 제외하는 기능이다.

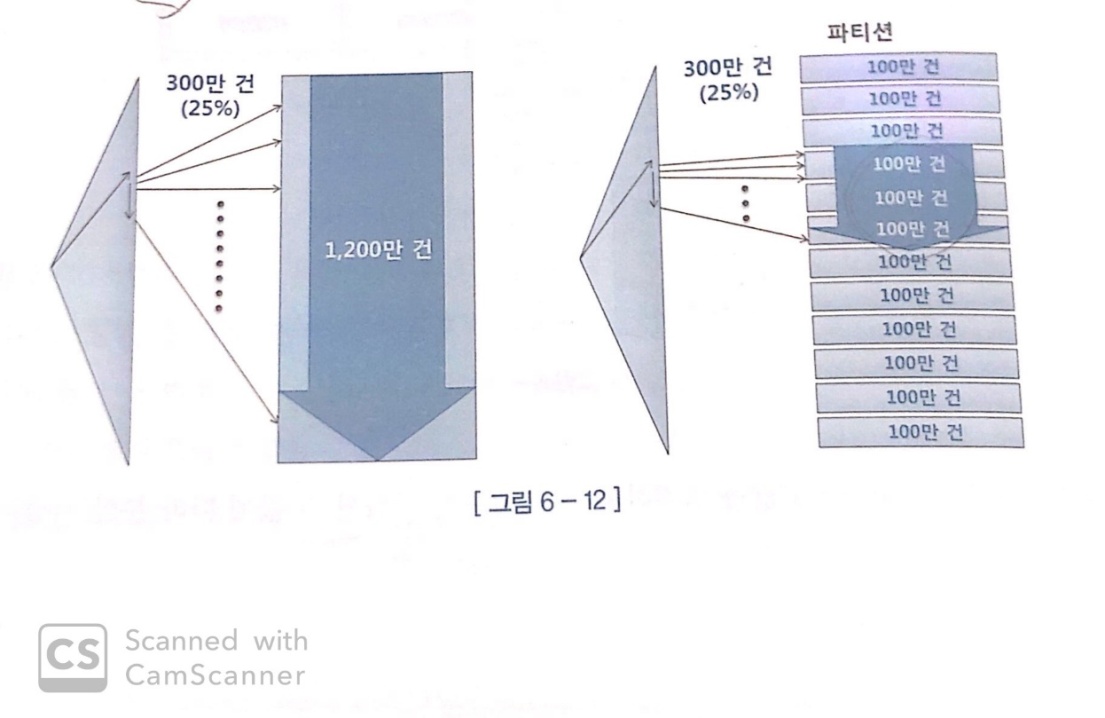
ex) 아래 쿼리의 조건절을 만족하는 데이터는 1200만 건 중 25%에 해당한다. 이 데이터를 인덱스

로 건건이 랜덤 액세스하면 테이블 전체를 스캔하는 것보다 오히려 성능이 더 느리다. 이 때,

**select \* from 주문**

**where 주문일자 >= ‘20120401’ and 주문일자 <= ‘20120630’**

파티션 되어 있는 테이블에서 일부만 읽고 멈출 수 있어 성능을 크게 향상시킨다.



- 파티션도 클러스터, IOT와 마찬가지로 관련 있는 데이터가 흩어지지 않고 물리적으로 인접하도록

저장하는 클러스터링 기술에 속한다. **파티션**은 세그먼트 단위로 모아서 저장, **클러스터**는 데이터

를 블록 단위로 모아서 저장, **IOT**는 데이터를 정렬된 순서로 저장하는 구조

3.1.2 해시 파티션

- 파티션 키 값을 해시 함수에 입력해서 반환받은 값이 같은 데이터를 같은 세그먼트에 저장하는

방식

- 파티션 개수는 사용자가 결정. 데이터를 분사하는 알고리즘은 오라클 내부 해시함수가 결정.

- 해시 파티션은 고객ID처럼 변별력이 좋고 데이터 분포가 고른 컬럼을 파티션 기준으로 선정해야

효과적

**create table 고객 ( 고객ID varchar2(5), 고객명 varchar2(10), … )**

**partition by hash(고객ID) partitions 4;**

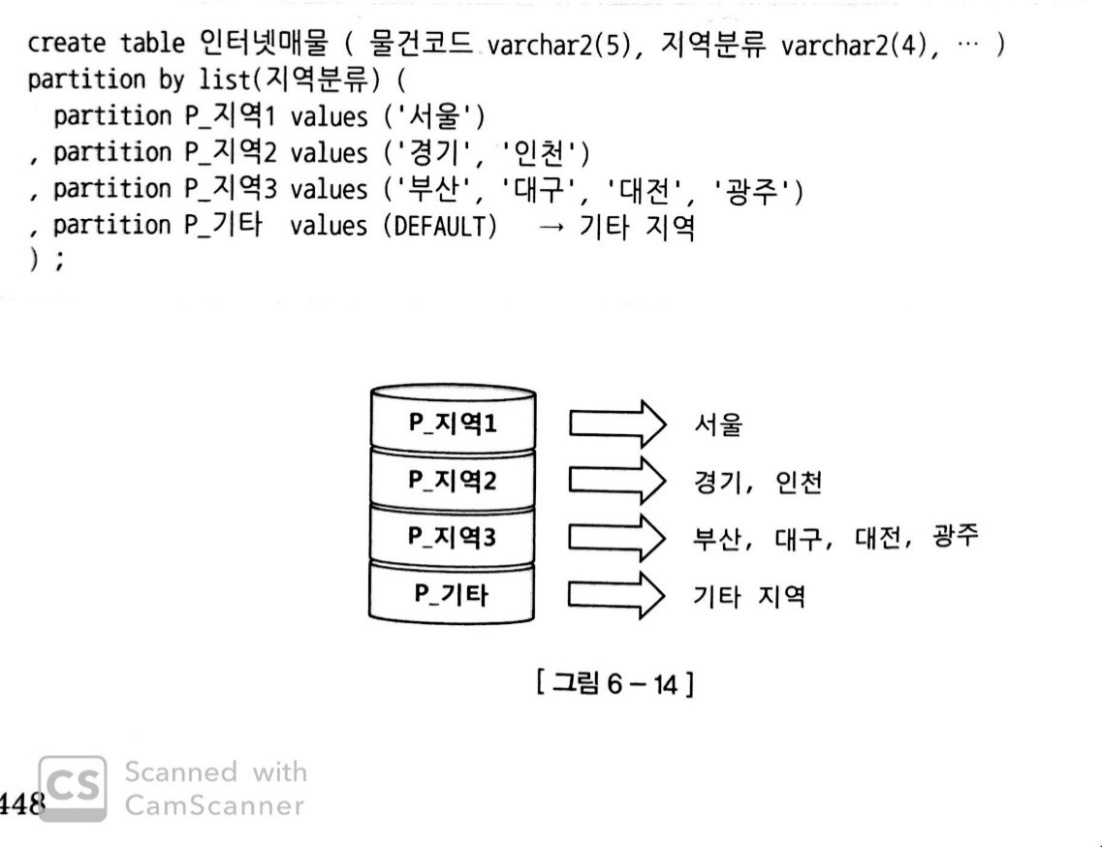
- 고객ID 기준으로 고객 테이블 해시 파티셔닝하는 방법

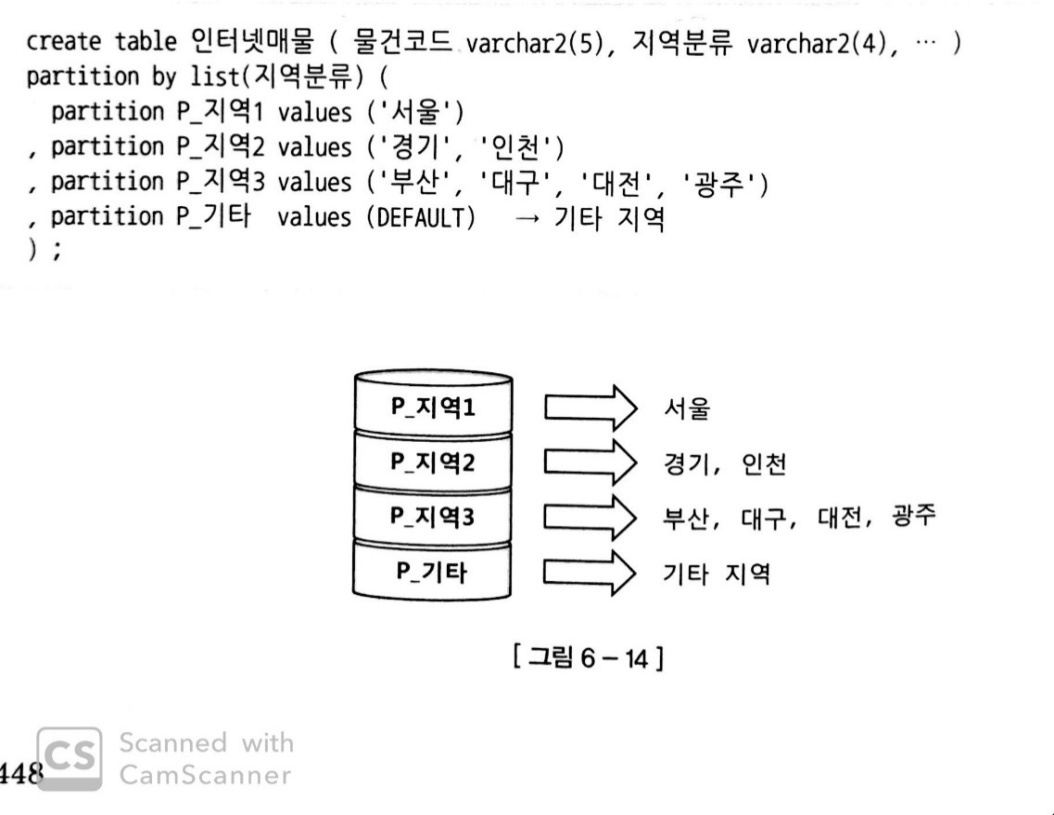
- 등치조건 또는 IN-List 조건으로 검색할 때만 파티셔닝 Pruning 작동

3.1.3 리스트 파티션

- 사용자가 정의한 그룹핑 기준에 따라 데이터를 분할 저장하는 방식

- 지역분류 기준으로 인터넷매물 테이블을 리스트 파티셔닝하는 방법





- Range 파티션에선 값의 순서에 따라 저장할 파티션이 결정

리스트 파티션에서는 순서와 상관없이 불연속적인 값의 목록에 의해 결정

- 해시 파티션은 오라클이 정한 해시 알고리즘에 따라 임의로 분할

리스트 파티션은 사용자가 정의한 논리적인 그룹에 따라 분할

**3.2** 인덱스 파티션

- 테이블은 파티션 테이블 / 비파티션 테이블로 나뉜다면

인덱스는 **로컬 파티션 인덱스 /** **글로벌 파티션 인덱스** / **비파티션 인덱스**로 나뉜다.

3.2.1 로컬 파티션 인덱스

**create index 주문\_x01 on 주문( 주문일자, 주문금액 ) LOCAL;**

- 각 테이블 파티션과 인덱스 파티션이 서로 1:1 대응 관계가 되도록 오라클이 자동으로 관리하는

파티션 인덱스

- 인덱스 파티션은 테이블 파티션 속성을 그대로 상속 받는다.

- 테이블 파티션 키가 주문일자면 인덱스 파티션 키도 주문일자!!

- 테이블 파티션 구성을 변경하더라고 인덱스를 재생성할 필요 X -> 관리 편의성

3.2.2 글로벌 파티션 인덱스

**create index 주문\_x03 on 주문( 주문금액, 주문일자 ) GLOBAL**

**partition by range(주문금액) (**

**partition P\_01 values less than ( 100000 )**

**partition P\_MX values less than ( MAXVALUE ) );**

- 로컬이 아닌 모든 파티션 인덱스를 뜻하며 테이블 파티션과 독립적인 구성을 갖는다.

- 파티션 유형이 다르거나, 파티션 키가 다르거나, 파티션 기준값 정의가 다른 경우

- 비파티션 테이블이어도 인덱스는 파티셔닝 가능!

- 테이블 파티션 구성을 변경하는 순간 Unusable 상태로 바뀌므로 곧바로 인덱스를 재생성해 줘야

한다.

3.2.3 비파티션 인덱스

- 파티셔닝하지 않은 인덱스

- 만드는 방법은 일반 CREATE INDEX 문!

- 비파티션 인덱스는 여러 테이블 파티션을 가리킨다 => **‘글로벌 비파티션 인덱스’**

3.2.4 Prefixed vs Nonprefixed

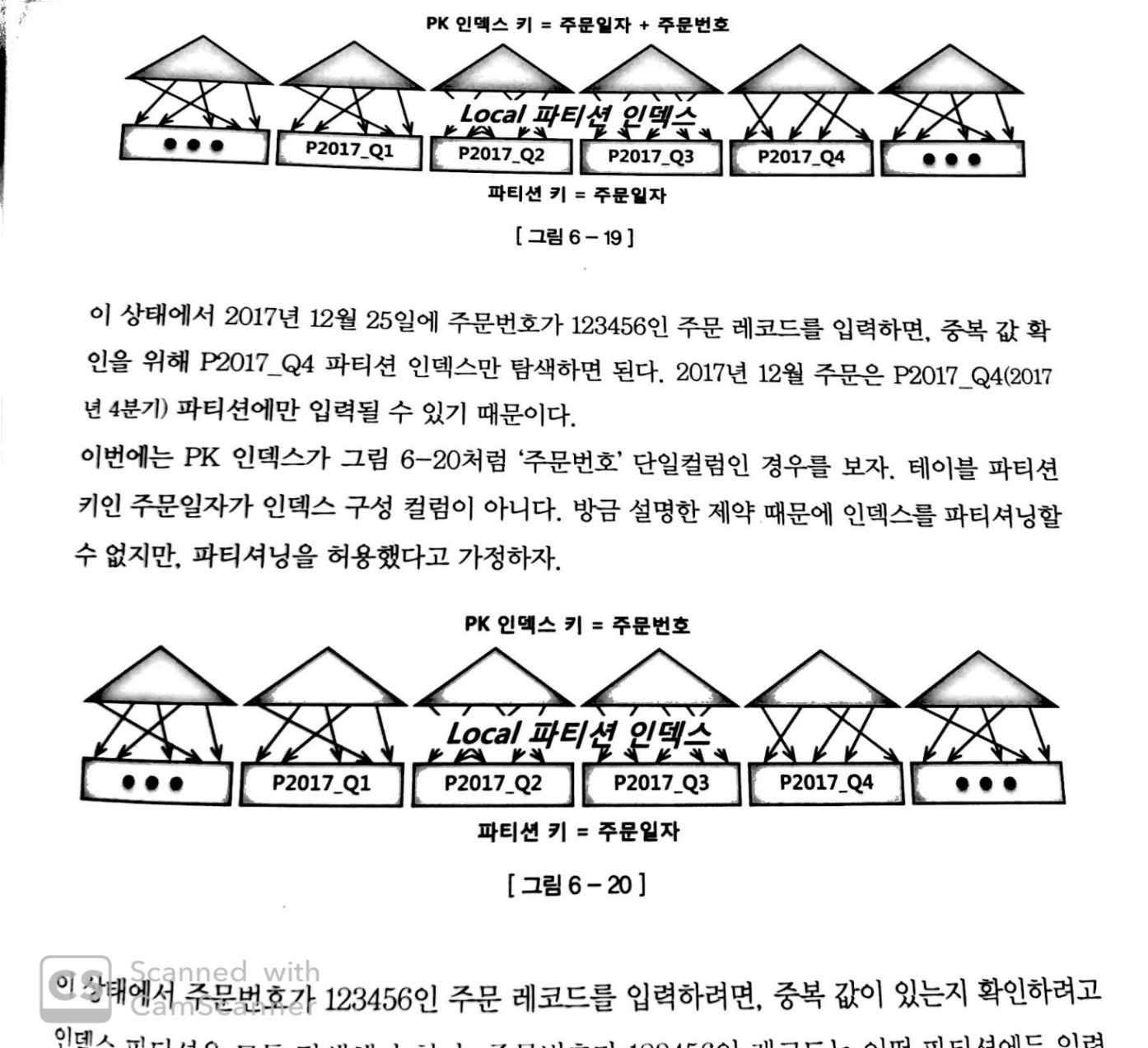
- 파티션 인덱스를 Prefixed와 Nonprefixed로 나눌 수 있음

- Prefixed : 인덱스 파티션 키 컬럼이 인덱스 키 컬럼 왼쪽 선두에 위치

- Nonprefixed : 인덱스 파티션 키 컬럼이 인덱스 키 컬럼 왼쪽 선두에 위치하지 않는다.

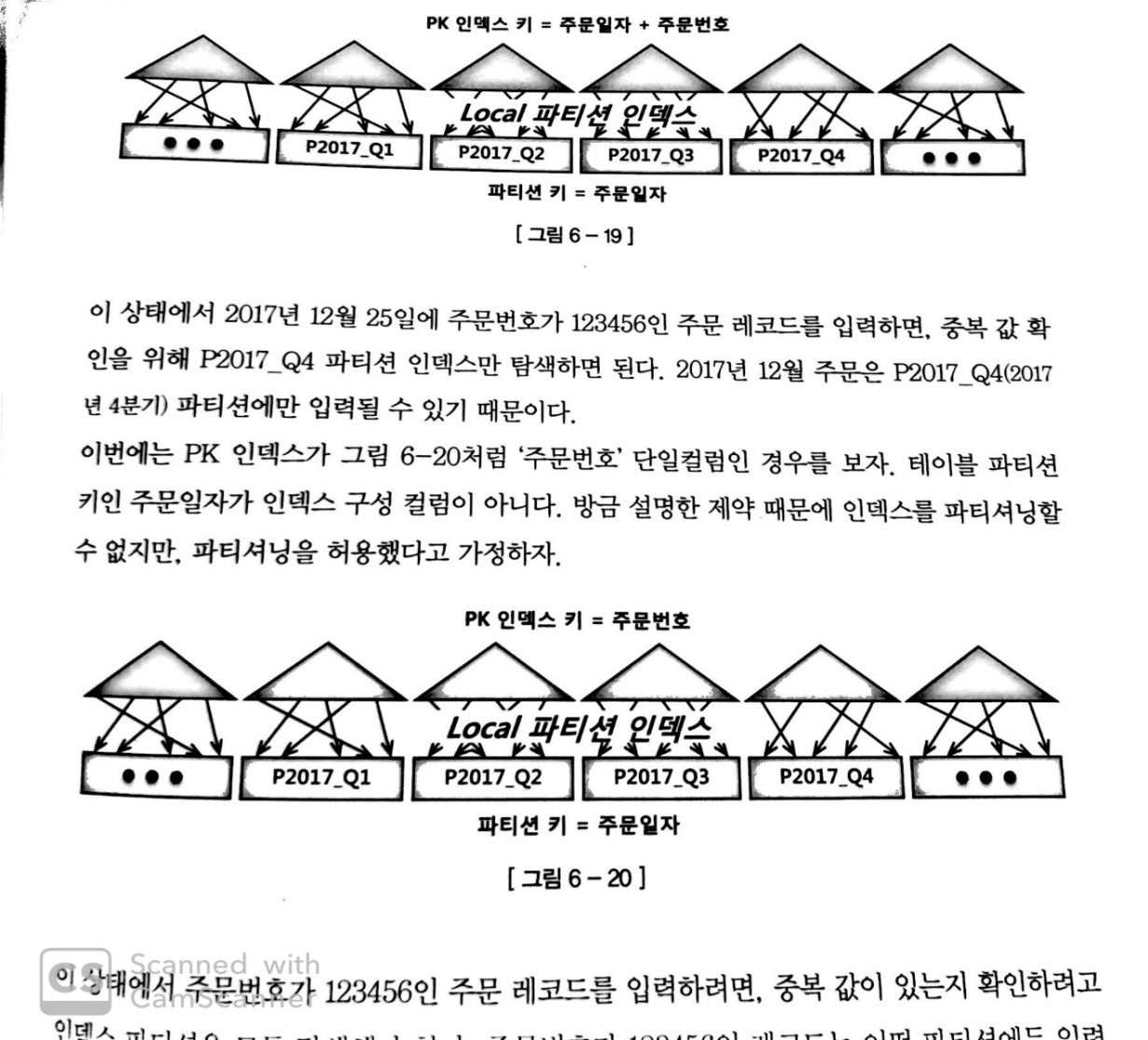
3.2.5 중요한 인덱스 파티션 제약

ex) PK 인덱스 키 = 주문일자 + 주문번호, 인덱스는 로컬 파티션



-> 여기서 2017년 12월 25일, 주문번호가 123456인 주문 레코드를 입력하면 중복 값 확인은 **P2017\_Q4** 파티션 인덱스만 탐색하면 된다.

ex) PK 인덱스 키 = 주문일자 ( 테이블 파티션 키인 주문일자가 인덱스 구성 컬럼 X )



-> 여기서 주문번호가 12345인 주문 레코드를 입력하면 중복 값을 확인하려면 인덱스 파티션을

모두 탐색해야 한다.

- **“Unique 인덱스를 파티셔닝하려면, 파티션 키가 모두 인덱스 구성 컬럼이어야 한다!”**

=> 이 제약으로 인해 PK 인덱스를 로컬 파티셔닝하지 못하면 파티션 구조 변경 작업을 하는

순간 PK 인덱스가 Unusable 상태로 바뀌므로 인덱스를 재생성해야하고, 그동안 테이블을

사용하는 서비스를 중단해야 한다.

- 대량으로 데이터를 추가/변경/삭제하는 기준 컬럼을 PK에 포함하도록 노력해야 한다.

**3.3** 파티션을 활용한 대량 UPDATE 튜닝

- 대량 데이터를 입력/수정/삭제할 때는 인덱스를 Drop 하거나 Unusable 상태로 변경하고서 작업하는

방법을 많이 활용

- 입력/수정/삭제하는 데이터 비중이 5%를 넘으면, 인덱스를 그대로 둔 상태에서 작업하는 것 보다

인덱스 없이 작업한 후에 재생성 하는게 더 빠르다.

ex) 거래 테이블 총 10억 건 조건절에 만족하는 데이터 5000만 건(5%)라고 가정

**update 거래**

**set 상태코드 = ‘ZZZ’**

**where 상태코드 <> ‘ZZZ’ and 거래일자 < ‘20150101’**

-> 실무적으로 이정도 대용량 테이블이라면 인덱스 전체 재생성은 부담이 크기 때문에 인덱스를

그대로 둔 상태에서 작업한다.

- 이 때 테이블이 파티셔닝 되어있고 인덱스도 로컬 파티션이라면 수정된 값을 갖는 **임시 세그먼트를**

**만들어 원본 파티션과 바꾸는 방식**을 이용하여 대량 데이터 변경

① 임시 테이블 생성 (nologging 모드로 생성)

**create table 거래\_t nologging**

**as**

**select \* from 거래 where 1 = 2;**

② 거래 데이터를 읽어 임시 테이블에 입력하면서 상태코드 값 수정

**insert /\*+ append \*/ into 거래\_t**

**select 고객번호, 거래일자, 거래순번, … ,**

**(case when 상태코드 <> ‘ZZZ’ then ‘ZZZ’ else 상태코드 end) 상태코드**

**from 거래**

**where 거래일자 < ‘20150101’;**

③ 임시 테이블에 원본 테이블과 같은 구조로 인덱스 생성(nologging 모드로 생성)

**create unique index 거래\_t\_pk on 거래\_t (고객번호, 거래일자, 거래순번) nologging;**

**create index 거래\_t\_x1 on 거래\_t (거래일자, 고객번호) nologging;**

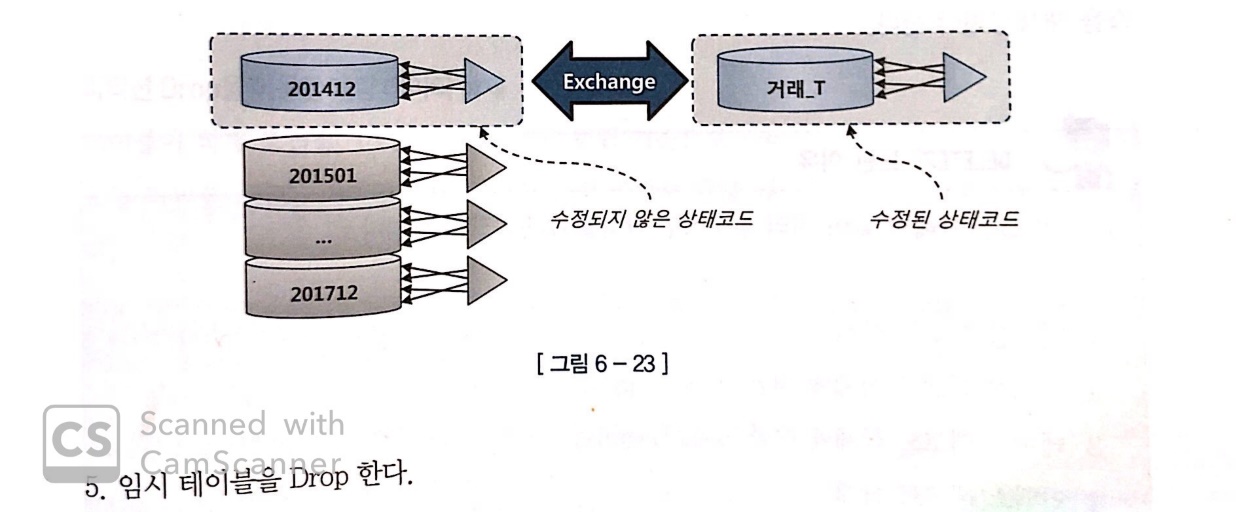
**create index 거래\_t\_x2 on 거래\_t (상태코드, 거래일자) nologging;**

**alter table 거래**

**exchange partition p201412 with table 거래\_t**

**including indexes without validation;**

④ 원본 파티션과 임시 테이블을 Exchange



⑤ 임시 테이블 Drop

**drop table 거래\_t;**

⑥ 파티션을 logging 모드로 전환

**alter table 거래 modify partition p201412 logging;**

**alter index 거래\_pk modify partition p201412 logging;**

**alter index 거래\_x1 modify partition p201412 logging;**

**alter index 거래\_x2 modify partition p201412 logging;**

**3.4** 파티션을 활용한 대량 DELETE 튜닝

**delete from 거래**

**where 거래일자 < ‘20150101’;**

- 수천만 건 데이터를 삭제할 때도, 대용량 테이블 인덱스를 모두 Drop 했다가 재생성하기에는 부담이

크다.

- DELETE가 느린 이유

① 테이블 레코드 삭제

② 테이블 레코드 삭제에 대한 Undo Logging

③ 테이블 레코드 삭제에 대한 Redo Logging

④ 인덱스 레코드 삭제

⑤ 인덱스 레코드 삭제에 대한 Undo Logging

⑥ 인덱스 레코드 삭제에 대한 Redo Logging

⑦ Undo(2번 5번)에 대한 Redo Logging

3.4.1 파티션 Drop을 이용한 대량 데이터 삭제

- 테이블이 조건절 컬럼 기준으로 파티셔닝 되어 있고 인덱스도 로컬 파티션이라면 대량 데이터

순식간에 삭제 가능

**alter table 거래 drop partition p201412;**

3.4.2 파티션 Truncate를 이용한 대량 데이터 삭제

- 조건을 만족하는 데이터가 소수이면 아래 DELETE 문 그대로 사용

**delete from 거래**

**where 거래일자 < ‘20150101’**

**and (상태코드 <> ‘ZZZ’ or 상태코드 is null);**

- 조건을 만족하는 데이터가 대다수이면 **남길 데이터만 백업했다가 재입력하는 방식**을 사용

① 임시 테이블을 생성하고, 남길 데이터만 복제

**create table 거래\_t**

**as**

**select \* from 거래**

**where 거래일자 < ‘20150101’ and 상태코드 = ‘ZZZ’;**

② 삭제 대상 테이블 파티션을 Truncate

**alter table 거래 truncate partition p201412;**

③ 임시 테이블에 복제해 둔 데이터를 원본 테이블에 입력

**insert into 거래**

**select \* from 거래\_t;**

④ 임시 테이블 Drop

**drop table 거래\_t;**

- 서비스 중단없이 파티션을 Drop 또는 Truncate 하려면 아래 조건 모두 만족

① 파티션 키와 커팅 기준 컬럼이 일치해야 함

② 파티션 단위와 커팅 주기가 일치해야 함

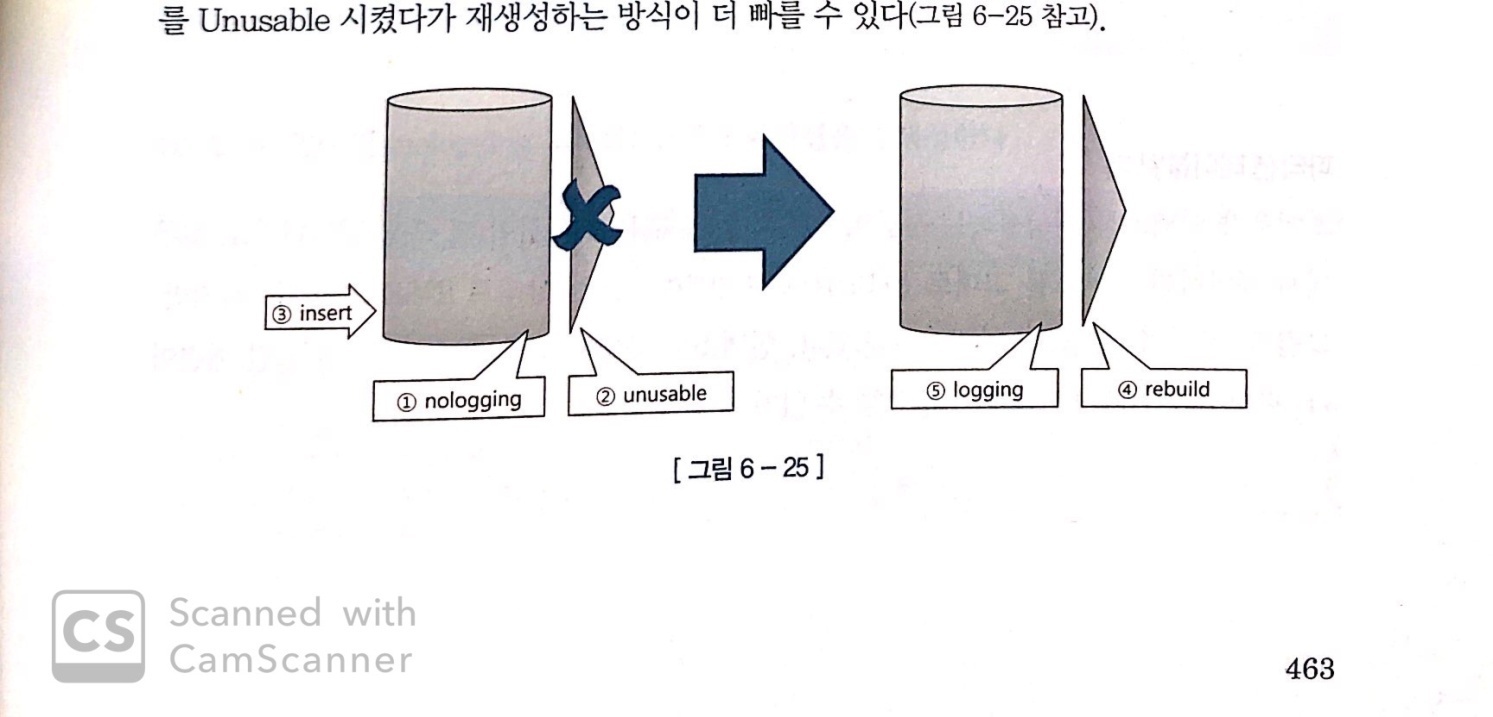
③ 모든 인덱스가 로컬 파티션 인덱스이어야 함

**3.5** 파티션을 활용한 대량 INSERT 튜닝

3.5.1 비파티션 테이블일 때

- 비파티션 테이블에 손익분기점을 넘는 대량 데이터를 INSERT 하려면, 인덱스를 Unusable 시켰다

가 재생성하는 방식이 더 빠를 수 있다.



① 테이블을 nologging 모드로 전환

**alter table target\_t nologging ;**

② 인덱스를 Unusable 상태로 전환

**alter index target\_t\_x01 unusable ;**

③ 대량 데이터 입력

**insert /\*+ append \*/ into target\_t**

**select \* from source\_t ;**

④ 인덱스 재생성

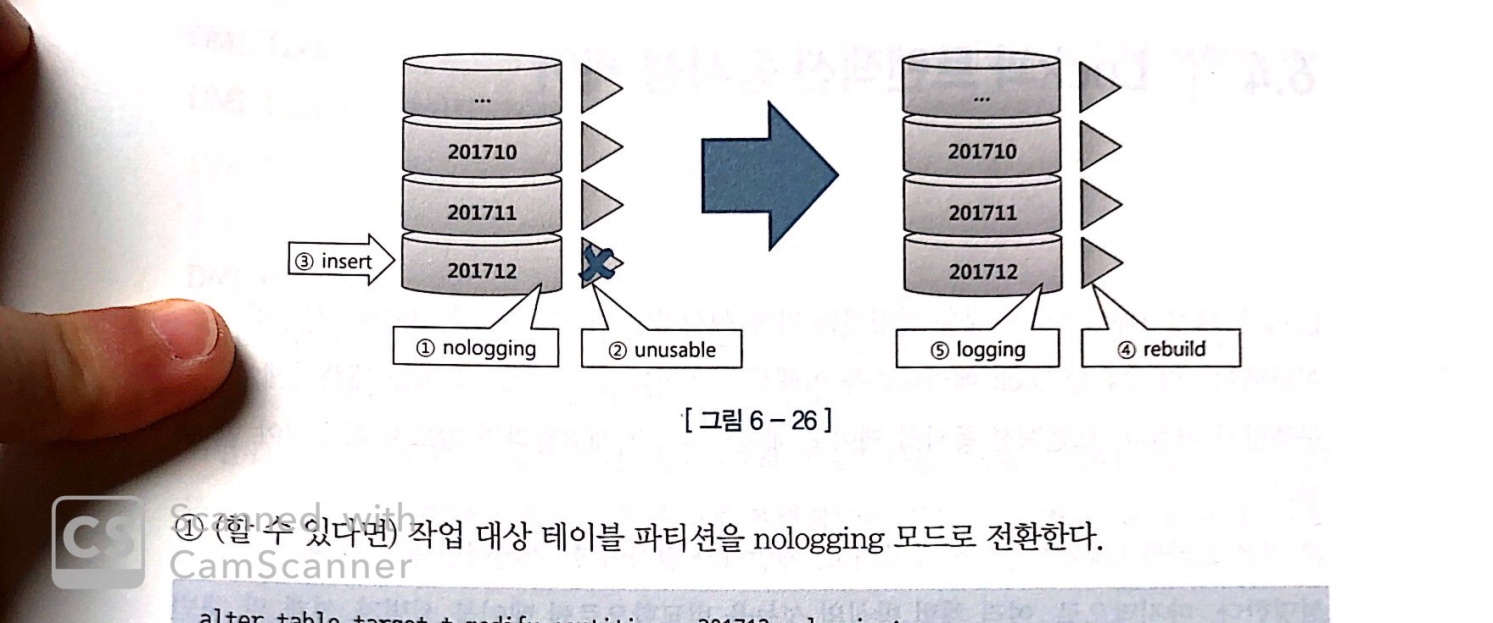
**alter index target\_t\_x01 rebuild nologging ;**

⑤ logging 모드로 전환

**alter index target\_t\_x01 rebuild nologging ;**

3.5.2 파티션 테이블일 때

- 파티셔닝 테이블이고, 로컬 파티션 인덱스이면 파티션 단위로 인덱스 재생성



① 테이블을 nologging 모드로 전환

**alter table target\_t modify partition p\_201712 nologging ;**

② 대상 테이블 파티션과 매칭되는 인덱스 파티션은 Unusable 상태로 전환

**alter index target\_t\_x01 modify partition p\_201712 unusable ;**

③ 대량 데이터를 입력

**insert /\*+ append \*/ into target\_t**

**select \* from source\_t where dt between ‘20171201’ and ‘20171231’ ;**

④ 인덱스 파티션을 재생성

**alter index target\_t\_x01 rebuild partition p\_201712 nologging ;**

⑤ 작업 파티션을 nologging 모드로 전환

**alter index target\_t modify partition p\_201712 logging ;**

**alter index target\_t\_x01 modify partition p\_201712 logging ;**