SQL 튜닝 스터디 (2020.04.01)

작성인 :박재형

3.4 인덱스 설계

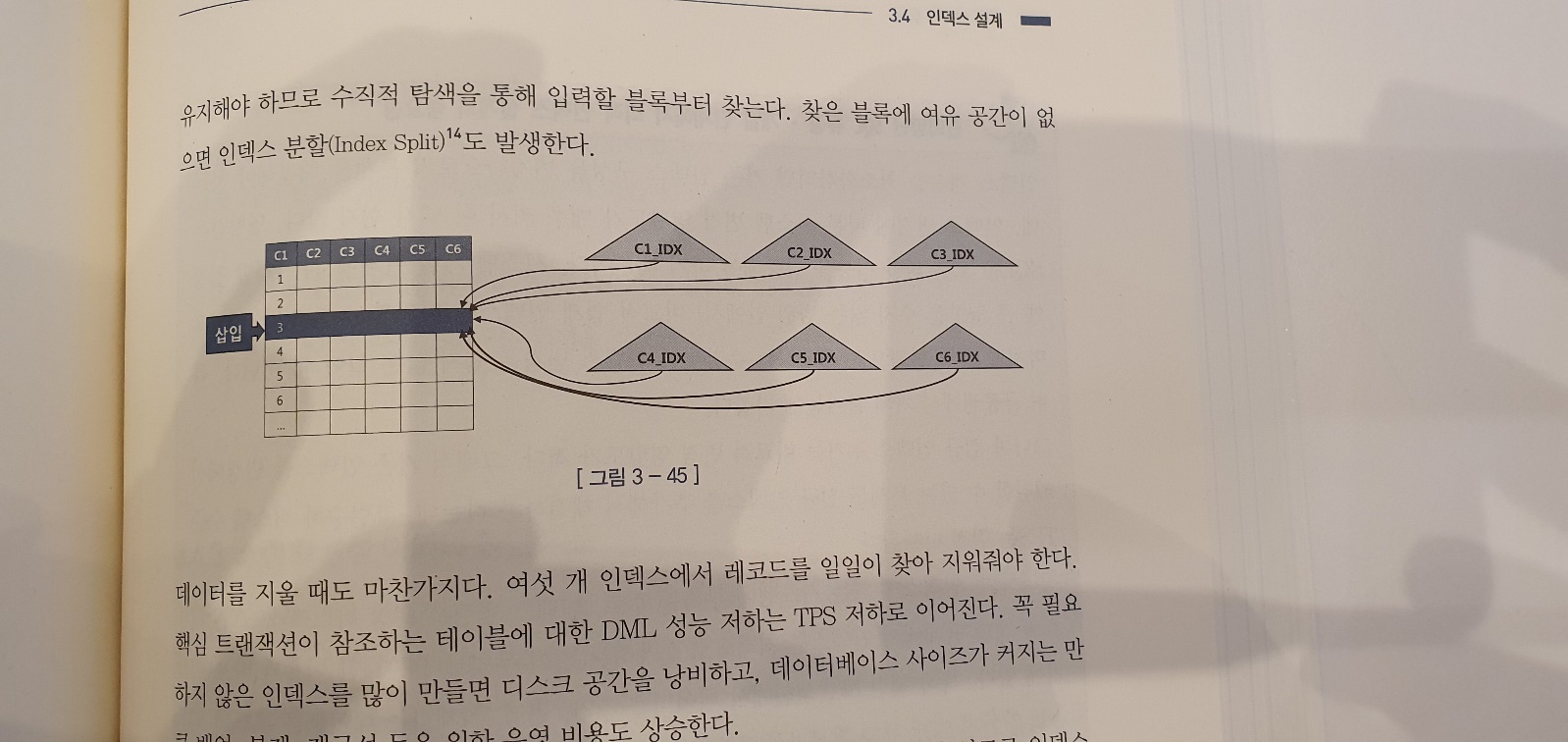
3.4.1 인덱스 설계가 어려운 이유?

\* 인덱스가 많으면 발생하는 문제

- DML 성능 저하(-> TPS저하)

\* 인덱스가 많으면 신규데이터를 입력할 때 각각의 인덱스에도 데이터 입력이 필요

\* 인덱스에 데이터 입력 시 정렬 상태를 유지해야 하므로 수직적 탐색 발생, 여유공간 부족 시 인덱스 분할 발생



\* 데이터 삭제 시 레코드를 일일이 삭제해줘야 함

- 데이터베이스 사이즈 증가(-> 디스크 공간 낭비)

- 데이터베이스 관리 및 운영 비용 상승

3.4.2 인덱스 선택 시 가장 중요한 선택 기준(인덱스 스캔 효율성 판단 기준)

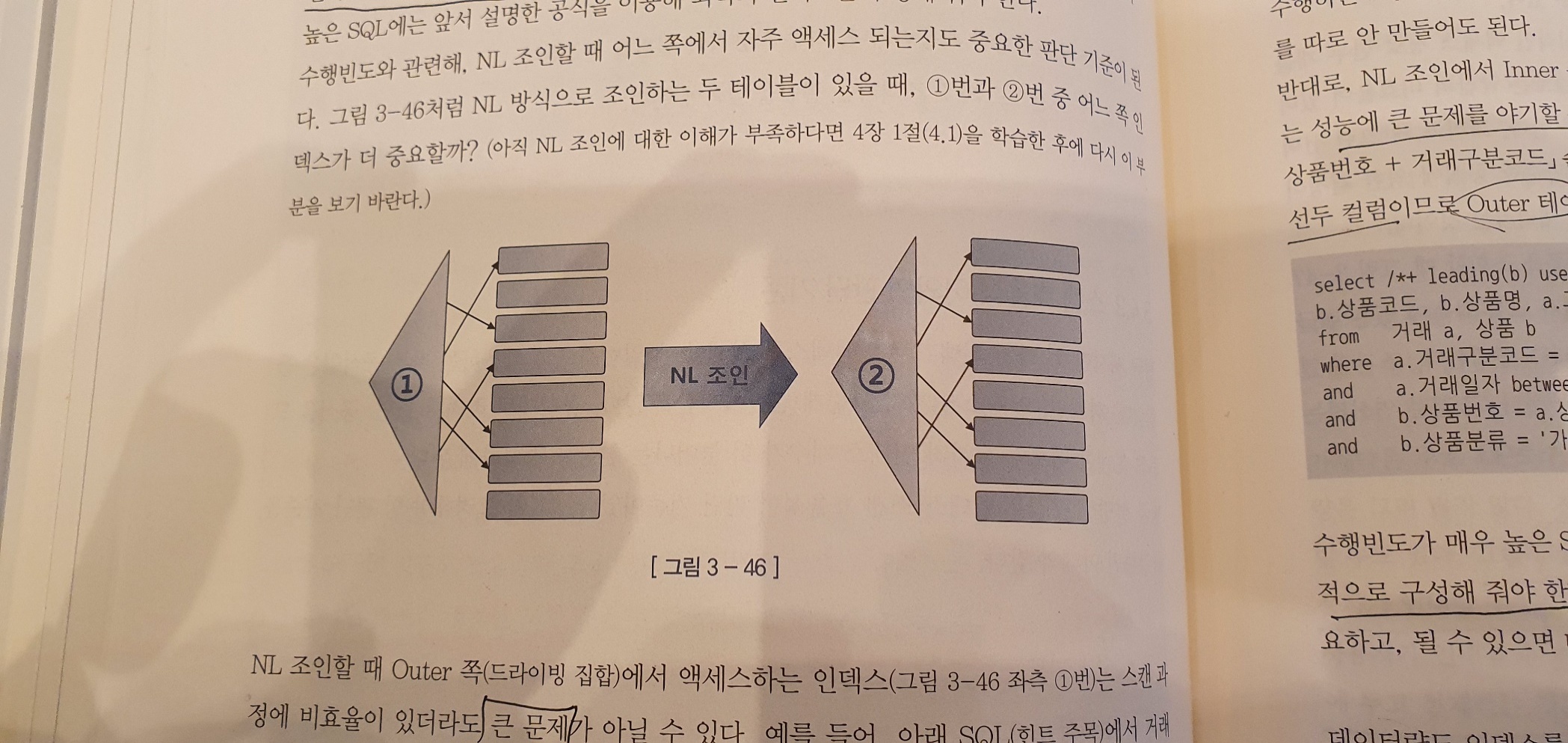
- 조건 절에 항상 사용하거나 자주 사용하는 컬럼을 선정해야 한다.

- '=' 조건으로 자주 조회하는 컬럼을 앞쪽에 둔다.

3.4.3 스캔 효율성 이외의 판단 기준

- 수행 빈도(판단 기준 내에서 가장 중요하다고 보면 됨) -> 수행빈도가 높을수록 그 먄큼 최적의 인덱스를 구성해줘야 하므로

-> NL조인 시 어느 쪽에서 자주 액세스 되는지도 중요한 판단 기준이 된다.(4.1절)



- 업무상 중요도

- 클러스터링 펙터(데이터가 모여있는 정도)

- 데이터 양

-> 데이터 양이 적다면(라고 판단이 된다면) 굳이 인덱스를 많이 만들 필요가 없다. Full scan으로도 충분히 빠르기 때문에

-> 테이블이 작으면 인덱스를 많이 만들어도 사실 저장공간이나 트랜잭션 부하 측면에서 그다지 문제 될 건 없음

-> 다만 초대용량일 때는 인덱스 설계 할 때 인덱스를 늘리거나 줄일 시 시스템에 미치는 영향력은 큼

- DML부하(= 기존 인덱스 개수. 초당 DML 발생량, 자주 갱신하는 컬럼 포함 여부 등)

- 저장 공간

- 인덱스 관리 비용 등

\*NL조인 : 선행테이블의 조건을 만족하는 행을 추출하여 후행 테이블을 읽으면서 조인을 수행하는 것

작업순서 : 선행테이블의 주어진 조건을 만족하는 행 추출 -> 선행 테이블의 조인 키를 가지고 후행테이블에서 조인 수행 -> 선행테이블의 조건을 만족하는 모든 행에 대해 1번 작업 반복 수행

3.4.4 공식을 초월한 전략적 설계

- 3.4.2 항에서 제시한 두 가지 공식만 알면 누구나 쉽게 설계가 가능하다(조건 절에 항상 사용하거나 자주 사용하는 컬럼에 인덱스를 선택, '=' 조건으로 자주 조회하는 컬럼을 앞쪽에 둠)

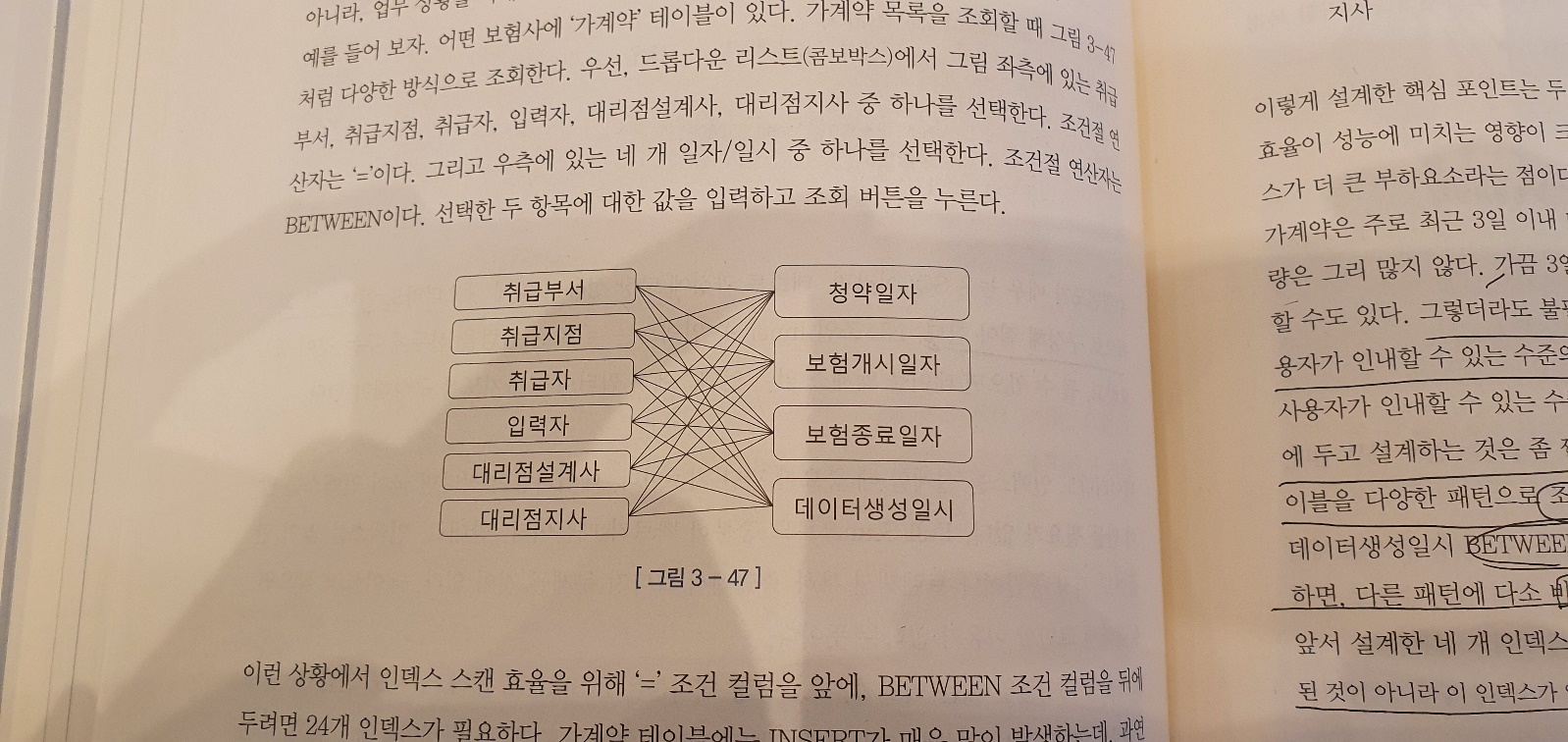
- 조건 절 패턴이 열 개 있을 때 인덱스를 하나씩 만들 수는 없다. SQL 튜닝 전문가라면 열 개 중 최적을 달성해야 할 핵심적인 액세스 경로 한두 개를 전략적으로 선택하여 최적 인덱스를 설계, 나머지 액세스 경로는 약간의 비효율이 있더라도 목표 성능을 만족하는 수준으로 인덱스를 구성해야 한다.

- 그 후 왜 그런 선택을 했는지, 전략적 판단 근거가 무엇인지 답을 할 수 있어야 함

- 단순한 공식에 따라 결정하기보단, 업무사항을 이해하고 나름의 판단 기준을 가지고 결정을 내리라는 것

예) 가계약 테이블이 있다는 가정.

- 가계약 목록 조회 시 그림 3-47처럼 다양한 방식으로 조회



- 먼저 드롭 다운 리스트에서 그림 좌측에 있는 취급 부서, 취급 지점, 취급자, 입력자, 대리점설계사, 대리점 지사 중 하나를 선택

- 이때 조건 절 연산자는 '='

- 그리고 우측에 있는 네 개 일자/일시 중 하나를 선택

- 조건 절 연산자는 BETWEEN

- 선택한 두 항목에 대한 값을 입력 후 조회버튼 클릭

좌측 항목에서 우측 항목까지 다 가능케 하려면 인덱스를 24개가 필요하다. 근데 24개를 다 만들기에는 가계약 테이블에는 INSERT가 많이 발생하기 때문에 부담이 있다.

따라서 일자나 일시 조건을 선두에 놓고 자주 사용하는 필더조건을 모두 뒤쪽으로 추가해서 다음과 같은 셜계가 완료된다.

설계 결과

- X01 : 청약일자 + 취급부서 + 취급지점 + 취급자 + 입력자 + 대리점 설계사 + 대리점 지사

- X02 : 보험개시일자 + 취급부서 + 취급지점 + 취급자 + 입력자 + 대리점 설계사 + 대리점 지사

- X03 : 보험종료일자 + 취급부서 + 취급지점 + 취급자 + 입력자 + 대리점 설계사 + 대리점 지사

- X04 : 데이터생성일시 + 취급부서 + 취급지점 + 취급자 + 입력자 + 대리점 설계사 + 대리점 지사

핵심 포인트는

1. 일자 조회구간이 길지 않으면 인덱스 스캔 비효율에 영향을 미치지 않는다는 점

2. 인덱스 스캔 효율보다 테이블 액세스가 더 큰 부하요소라는 것

위의 4개의 인덱스를 설계한 것은

- 가계약은 3일 이내의 데이터를 조회, 대개 전일자로 조회하기 때문에 인덱스 스캔량은 그리 많지는 않음

- 가끔 3일 초과한 기간으로 조회할 수 있고, 어쩌다 한달 치를 조회할 수도 있다.

- 그렇더라도 불필요한 테이블 엑세스는 전혀 발생하지 않도록 설계를 했으므로 사용자가 안내할 수 있는 수준의 성능이 나옴

--> 이때 조금 생각을 해보면 일자 조회이기 때문에 BETWEEN 조건절을 사용하는데 있어서 조금 찜찜하다. 그런데도 이 결정을 한 것은 가계약 테이블의 다양한 패턴으로 조회를 하지만, 그 중 많이 사용하는 것은 입력자'='와 테이블 생성 일시 BETWEEN조건이므로 다른 패턴에 비해 비효율이 있어 보여도 업무에는 크게 지장이 없게 설계를 한 것

+X05 :입력자 + 데이터 생성 일시

24개의 인덱스가 나와야 하지만 업무 상황을 고려한 전략적 판단으로 5개로 줄였다. 인덱스 개수를 최소화하면 사용 빈도가 높거나 중요한 엑세스 경로가 새로 도출이 되었을 때 최적의 인덱스를 추가할 여유도 생김

3.4.5 소트 연산을 생략하기 위한 컬럼 추가

- 인덱스는 항상 정렬 상태를 유지하므로 ORDER BY, GROUP BY를 위한 소트 연산을 생략할 수 있게 해줌.

- 조건 절에 사용하지 않는 컬럼 이더라도 소트 연산을 생략할 목적으로 인덱스 구성에 포함시킴으로써 성능 개선을 도모할 수 있음

- 아래 쿼리에 ORDER BY 절이 있음에도 불구하고 소트 연산이 발생하지 않도록 인덱스를 구성

SELECT 계약ID, 청약일자, 입력자ID, 계약상태코드, 보험시작일자, 보험종료일자

FROM 계약

WHERE 취급지점ID = :trt\_brch\_id

AND 청약일자 between :sbcp\_dt1 and :sbcp\_dt2

AND 입력일자 >= trunc(sysdate - 3)

and 계약상태코드 in (:ctr\_stat\_cd1, :ctr\_stat\_cd2, :ctr\_stat\_cd3 )

order by 청약일자, 입력자ID

-성능을 고려하지 않는다면 order by 절 순서대로 청약일자+입력자 id로 구성하면 됨

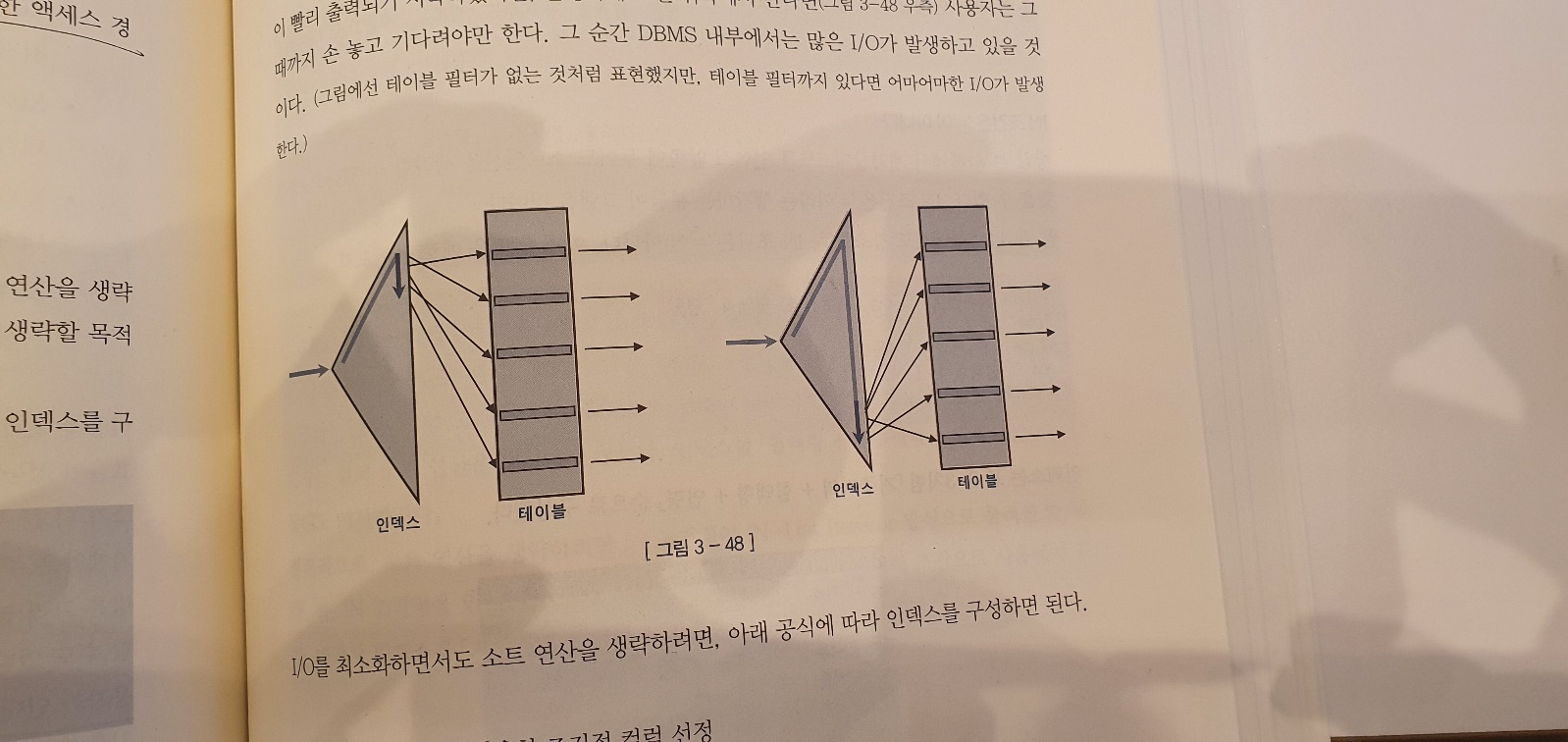
- '='조건절 컬럼은 order by 절에 없더라도 인덱스 구성에 포함 가능(위 SQL에서는 취급지점 ID가 “=”조건)

- 이를 포함하여 청약일자 + 취급지점ID + 입력자 ID 순으로 구성해도 소트 연산을 생략할 수 있다는 뜻(위치는 앞뒤 중간에 어디에 두어도 상관 없음)

- “=” 가 아닌 조건절 컬럼들은 반드시 ORDER BY 컬럼보다 뒤쪽에 두어야 소트 연산을 생략할 수 있다.

일단 소트는 생략은 했는데 성능이 문제, 조건을 만족하는 데이터가 앞쪽에 있다면 결과집합이 빨리 출력되기 시작하겠지만 뒤에 있으면 당연히 느려지기 마련

그 순간에 I/O는 계속 발생하고 있을 것인데, 결론적으로는 I/O를 최소화하면서 소트연산을 생략하려면 아래 공식처럼 인덱스를 구성하면 된다.



1. “=” 연산자로 사용한 조건 절 컬럼 선정

2. ORDER BY 절에 기술한 컬럼 추가

3. “=” 연산자가 아닌 조건 절 컬럼은 데이터 분포를 고려해 추가여부 결정

- 위 공식대로 구성하면 취급지점id + 청약일자 + 입력자 id순으로 구성한다.

- 입력일자와 계약상태코드는 뒤쪽에 붙여도 되고, 안 붙여도 된다.

- 이들 조건을 만족하는 데이터가 적으면, 인덱스에 추가하는게 좋다. 테이블 엑세스를 줄일 수 있기 때문

- 이들 조건을 만족하는 데이터가 많으면 굳이 인덱스에 추가하지 않아도 된다. 테이블에서 필터링 할 때와 큰 성능 차이가 없기 때문

- 단 몇 회라도 테이블 엑세스를 줄이면 조회 성능은 좋지만 반대 급부도 생각해야 한다.

IN 조건은 ”=”이 아니다.

-간단하게 IN조건을 사용하였을 때 UNION ALL 이 자동으로 생성되는 쿼리가 생성이 되고 마지막에 항상 ORDER BY를 수행하게 된다.인덱스를 거주지역+혈액형+연령으로 해 놨음에도 불구하고 최종적으로는 연령을 기준으로 소트가 진행되기 때문에 소트연산을 생략하고 싶다면 IN 조건절은 인덱스 엑세스 조건으로 사용하면 안된다는 것

|  |  |  |
| --- | --- | --- |
| 거주지역 | 혈액형 | 연령 |
| 서울 | A | 23 |
| 서울 | A | 35 |
| 서울 | A | 48 |
| 서울 | A | 62 |
| 서울 | O | 29 |
| 서울 | O | 32 |
| 서울 | O | 45 |
| 서울 | O | 57 |

3.4.6 결합 인덱스 선택도

* 인덱스 생성 여부를 결정할 때 선택도\*가 충분히 낮은지가 중요한 판단기준
* 인덱스 선택도는 인덱스 컬럼을 모두 =로 조회할 떄 평균적으로 선택되는 비율을 의미
* 선택도가 높은 인덱스는 생성해봐야 효용가치가 별로 없다.인덱스 엑세스가 자주 발생
* 인덱스를 생성할 때 반드시 선택도 카디널리티를 확인해야한다.

\*선택도 :전체 레코드 중에서 조건 절에 의해 선택되는 레코드 비율, 선택도에 총 레코드 수를 곱하여 카디널리티\*\*를 구한다.

\*\*카디널리티 : 데이터베이스테이블의 특정 열에 포함된 데이터 값의 고유성

SELECT count(\*) as NDV, max(cnt) as MX\_CARD, min(cnt) MN\_CARD, avg(cnt). as AVG\_CARD

FROM (

SELECT 계약ID, 취급지점ID, count(\*) as cnt

FROM 계약 조직

WHERE (계약 ID is not null or 취급지점ID is not null)

GROUP BY 계약ID, 취급지점ID

)

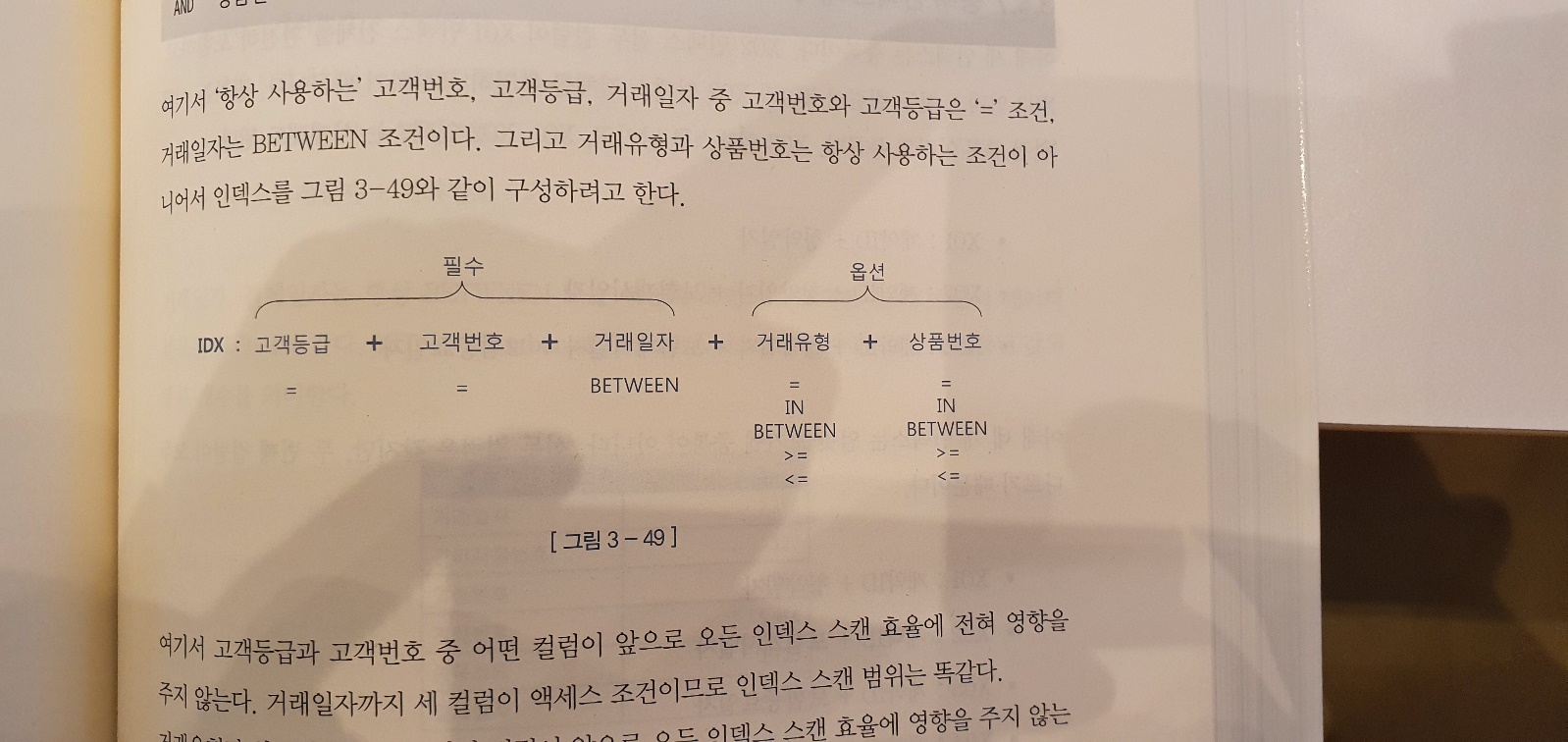
컬럼순서 결정시, 선택도 이슈

* 아래 쿼리에서 고객번호를 앞에 두는 것이 유리하다고 생각하겠지만,성별이나 고객번호 중 어떤 컬럼이 앞으로 오든 인덱스 스캔 효율에 전혀 차이가 없다.
* 둘 다 인덱스 엑세스 조건이므로 어떤 컬럼이 앞으로 오든 인덱스 스캔 범위는 똑같음

WHERE 성별= :GENDER

AND 고객번호 = :CUST\_NO

* 인덱스 설계 시 해야 할 일은 항상 사용하는 컬럼을 앞쪽에 두고 그 중 =조건을 앞쪽에 위치시키는 것이다.
* 선택도가 낮은 컬럼을 굳이 앞으로 두려는 노력은 의미가 없거나 오히려 손해일 수도 있다.



* 고객 등급과 고객 번호 중 어떤 컬럼이 앞에 오건 인덱스 스캔 효율에는 영향이 없다.
* 거래유형과 상품번호 간에도 어떤 컬럼이 앞으로 오든 인덱스 스캔 효율에 영향을 주지 않는다.
* 즉 필수조건(고객등급,고객번호,거래일자)는 앞에 두는 가정에 그들간의 순서는 상관이 없다는 것.
* Index skip scan이나 in-list활용,인덱스 압축 효율까지 고려한다면 방금 사례에선 고객 등급을 앞쪽에 두는 것이 유리하다.

결론 :인덱스 생성 여부를 결정할 때는 선택도도 중요하지만, 컬럼 간 순서를 결정할 때는 각 컬럼의 선택도보다 필수 조건 여부,연산자 형태가 더 중요한 판단 기준이라는 것.어느 컬럼을 앞에 두는지 결정 요소는 상황에 따라 판단할 일이다.

3.4.7 중복 인덱스 제거

예)아래 나와있는 세 인덱스는 중복이다. X02 인덱스 선두 컬럼이 X01 인덱스 전체를 포함하고, X03 인덱스 선두 컬럼이 X01과 X02 인덱스 전체를 완전히 포함하기 때문.이를 완전 중복이라 필자는 설명.이때 X01, X02는 남기고 X03만 살려도 사용이 가능

X01 :계약ID + 청약일자

X02 :계약ID + 청약일자 + 보험개시일자

X03 :계약ID + 청약일자 + 보험개시일자 + 보험종료일자

- 아래 네 개 인덱스는 중복이 아니다.선두는 같이만 두번째가 모두 다름

X01 :계약 ID + 청약일자

X02 :계약 ID + 보험개시일자

X03 :계약 ID + 보험종료일자

X04 :계약 ID + 데이터생성일시

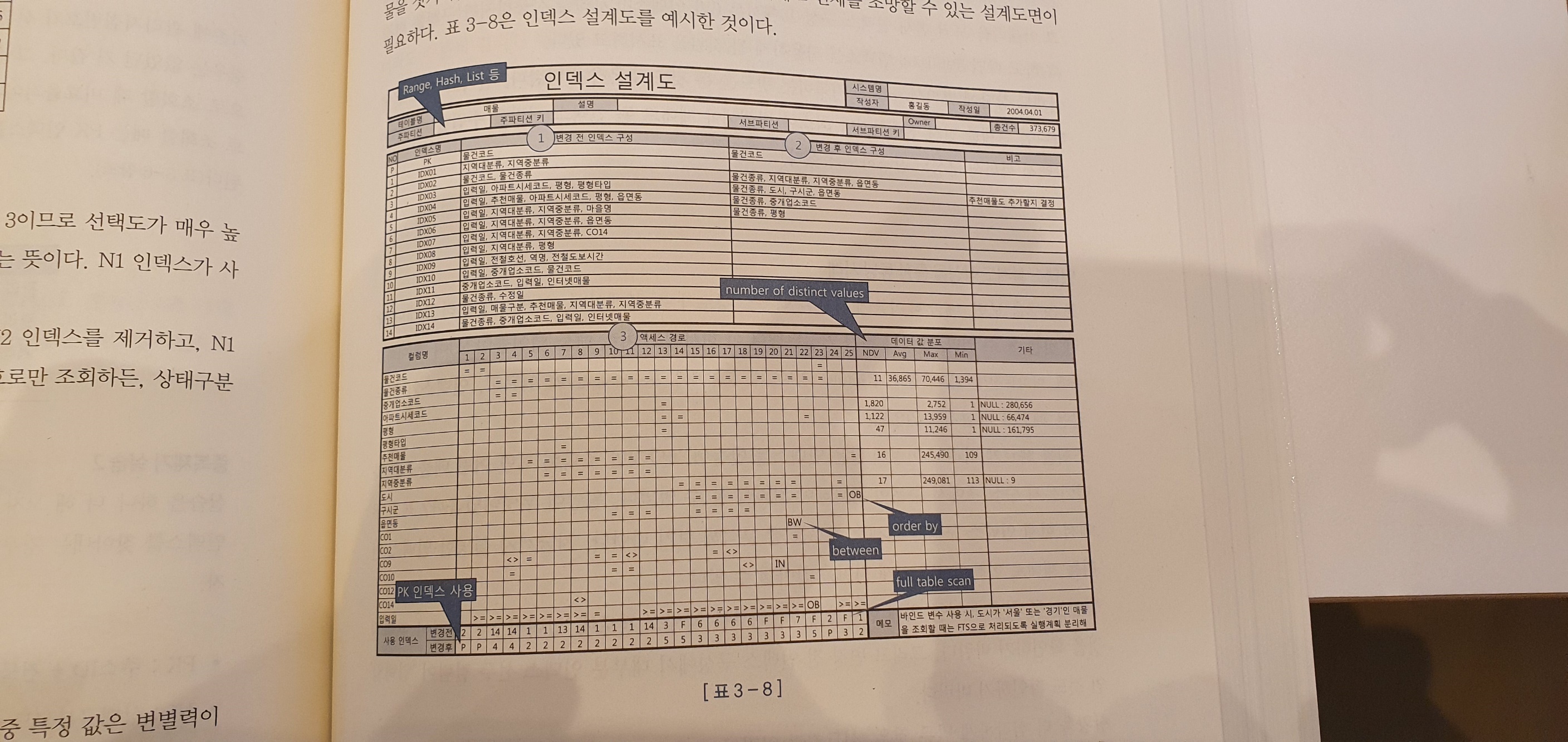
- 이때 계약ID의 평균 카디널리티가 낮다면 사실상 중복이다. 가령 계약ID 카디널리티가 5라 가정하면 계약ID를 ‘=’ 조건으로 조회하면 평균 다섯 건이 조회된다는 뜻이다.

- 즉 굳이 인덱스를 4개씩이나 만들 이유는 없다는 뜻. 아래처럼 하나만 만들면 그만. 이 떄를 완전중복과 대비하여 불완전 중복이라고 부른다고 함

X01 :계약ID + 청약일자 + 보험개시일자 + 보험종료일자 + 데이터 생성일시

3.4.8 인덱스 설계도 작성

* 인덱스 설계시 시스템 전체 효율을 따져야 한다고 앞에서 이야기를 했었다.조화를 이룬 건축물을 짓기 위해 설계도가 필수인 것 처럼 인덱스 설계에도 전체를 조망할 수 있다는 설계도가 필요하다.



* 인덱스 설계도에 보면 변경 전,변경 후가 둘로 나눠져 있다.
* 개별 SQL이 아니라 전체를 보면서 전략을 수립하려면,일단 테이블 별로 실제 발생하는 엑세스 유형을 모두 조사하는 과정이 필요하다.중간에 엑세스 경로가 그것이다.
* 17번 엑세스 경로를 예로 들면 조건절에 사용한 비교 연산자가 물건종류,도시,구시군,읍면동, CO9 ◇, 입력일 “>=” 인 경우를 표시한 것이다.
* 그리고 그 아래쪽을 보면 현재 인덱스 구성에서는 6번 인덱스를 사용하지만,구성을 변경하고 나면 3번 인덱스를 사용하게 될 것임을 표시하고 있다.
* 설계도 상단에 파티션 구성을 기록하는 필드를 뒀다. 인덱스 설계 전에 파티션 설계를 먼저 진행하거나 최소한 병행해야 제대로 된 인덱스 전락을 수립할 수 있다.