**2장. 인덱스 기본**

**Ⅰ.** **인덱스 구조 및 탐색**

**⑴** 인덱스 튜닝

**①** 튜닝의 핵심요소

- 인덱스 스캔 효율화 튜닝

-> 인덱스 스캔 과정에서 발생하는 비효율을 줄이는 것

- 랜덤 액세스 최소화 튜닝

-> 테이블 액세스 횟수를 줄이는 것, 성능에 미치는 영향이 더 큼.

**②** SQL 튜닝은 랜덤 I/O와의 전쟁

**⑵** 인덱스 구조

**①** 인덱스 : 대용량 테이블에서 필요한 데이터만 빠르게 효율적으로 액세스하기 위해 사용하는 오브젝트

**②** B\*Tree 인덱스

- DBMS는 일반적으로 B\*Tree 인덱스 사용

- B\*Tree 기본적인 구조

: 최상위에 하나의 ‘루트 블록’, 가장 하위인 ‘리프 블록’, 나머지인 ‘브랜치 블록’

- ‘루트 블록’과 ‘브랜치 블록’에 있는 레코드는 하위 블록에 대한 주소값

- ‘리프 블록’은 키 값으로 정렬되어 있음, 실제 데이터 레코드 가리키는 주소값(ROWID)

- ‘루트 블록’과 ‘브랜치 블록’에는 키 값을 **갖지 않는** 첫번째 레코드(LMC; Leftmost Child)가 가리키는 블록에는 키 값을 **가진** 첫번째 레코드 보다 작거나 같은 레코드 저장되어 있다.

**⑶** 인덱스 탐색

① 수직적 탐색 : 인덱스 스캔 시작지점을 찾는 과정

- ‘루트 블록’에서 시작해서 ‘리프 블록’까지 수직적 탐색

- 찾고자 하는 값보다 **‘크거나 같은 값’**을 만나면 직전 레코드가 가리키는 하위 블록으로 이동

(크거나 같은 값이 없다면 맨 마지막 레코드가 가리키는 하위 블록으로 이동)

② 수평적 탐색 : 스캔 시작점 찾았으면, 찾고자 하는 데이터 안 나타날 떄까지 인덱스 리프 블록을수평적 탐색. 데이터를 찾는 과정

- ‘리프 블록’끼리는 서로 앞뒤 블록에 대한 주소값을 갖는다.

- 수평적 탐색 이유 : 조건절을 만족하는 데이터를 모두 찾고, ROWID를 얻기 위함.

**⑷** 결합 인덱스 구조와 탐색

- 두 개 이상의 컬럼을 결합해서 인덱스를 만들 수 있음.

- (고객명+성별)과 (성별+고객명)처럼 인덱스 컬럼의 순서가 바뀌었을 경우 탐색하는 블록은 달라질 수 있지만 개수는 동일하다.