**선형 vs 이진 탐색**

선형과 이진 탐색은 원하는 데이터를 찾는 가장 기본적인 두 가지 방법입니다.

**선형 탐색**

이렇게 하나의 숫자 배열이 있다고 합시다. 여기서 2를 찾고 싶습니다.



선형 탐색은 그냥 가장 첫 원소부터 시작해서, 가장 마지막 원소까지 순서대로 하나씩 확인해보면서 2를 찾는 탐색법입니다.

**이진 탐색**

이번에는 숫자 배열이 오름차순으로 정렬돼있다고 합시다.



여기서 3을 찾고 싶으면 어떻게 찾을 수 있을까요? 물론 선형 탐색을 사용해서 가장 앞부터 뒤까지 모든 원소를 확인해볼 수 있는데요. 배열이 정렬돼있기 때문에 조금 더 효율적인 방법을 사용할 수 있습니다.

배열의 중간 원소를 확인합니다.

5입니다.

이 원소가 5이라는 사실을 통해서, 우리는 한 가지 확신을 얻을 수 있습니다. 바로 이 배열이 원소를 오름차순으로 저장하고 있기 때문에, 이 5 뒤에 있는 원소들에는 **절대로 3**이 있을 수 없다는 점이죠. 그렇기 때문에 이 뒤에 부분은 볼 필요가 없습니다. 다음에는 3이 있을 수도 있는 부분에서 또 3을 찾아봅니다. 이번에도 남은 범위의 중간 원소를 확인합니다.

2입니다.

자 이번에는 같은 논리로 3은 절대 2 앞에 있지 않을 것이라는 확신을 할 수 있습니다. 그렇기 때문에 지금 표시된 이 부분에서 다시 3을 찾으면 됩니다. 또 3가 있을 수도 있는 범위의 중간 숫자를 확인합니다.

3입니다.

원하는 값을 찾았습니다

이렇게 데이터가 특정 순서대로 정렬돼 있을 때는, 계속 남은 범위의 중간 원소를 확인해봄으로써, 탐색 범위를 배번 반씩 줄여나갈 수 있습니다. 이 방법을 이진 탐색이라고 부르는데요. 이진 탐색은 선형 탐색보다 훨씬 더 효율적으로 주어진 조건의 데이터를 찾을 수 있습니다.

**선형 vs 이진 탐색 차이**

좀 전문적으로 얘기하면 선형 탐색은 데이터를 O(n)*O*(*n*)으로, 이진 탐색은 데이터를 O(lg(n))*O*(*lg*(*n*))으로 찾는다고 하는데요. 이 차이는 1024개의 정렬된 데이터가 있을 때 선형 탐색을 쓰면 최대 1024번 만에 데이터를 찾아낼 수 있는 반면, 이진 탐색은 한 번 탐색할 때마다 남은 데이터의 반을 배제할 수 있기 때문에 최대 11번 만에 원하는 데이터를 찾아낼 수 있죠.

**인덱스**

인덱스는 "데이터가 정렬돼있으면 원하는 데이터를 더 빠르게 찾을 수 있다"라는 특성을 데이터베이스에 적용한 개념입니다.

우리는 일상 생활에서도 인덱스를 접할 수 있는데요. 좀 어려운 책들을 보면, 가장 뒤에 색인이나 인덱스라고 불리는 부분이 있습니다. 이 색인이나 인덱스에는 특정 개념들과, 그 개념들이 소개된 책의 페이지가 개념을 기준으로 정렬된 상태로 표기돼있는데요. 그럼 저희는 원하는 개념을 이진 탐색을 통해 찾고, 그 개념이 소개된 페이지로 쉽게 넘어갈 수 있습니다.

데이터베이스 인덱스 개념도 똑같습니다.

원래는 특정 컬럼 값을 갖는 로우를 찾고 싶을 때는 선형 탐색과 같이 모든 로우를 순서대로 돌면서 원하는 값을 찾아야하는데요.

인덱스는 테이블의 특정 컬럼들을 정렬된 상태로 저장합니다. 각 컬럼 값에 해당하는 로우의 주소와 함께요.



그럼 특정 이메일을 갖는 유저 로우를 찾고 싶다면, 인덱스에서 이진 탐색을 통해 빠르게 해당 이메일을 찾은 후, 같이 저장된 주소로 그 이메일을 갖는 로우를 찾아낼 수 있는 거죠. 책의 인덱스와 다른 게 하나도 없죠.

**Clustered 인덱스**

Clustered 인덱스는 데이터베이스에 저장돼있는 데이터 자체를 특정 순서대로 저장하는 인덱스입니다. 예를 들어 이 user 테이블에서 이메일 column에 대한 clustered 인덱스를 만든다고 할게요. 그럼 데이터베이스 테이블 안의 내용이 실제로 email의 알파벳 순서대로 저장이 됩니다. a에서 시작해서 쭉 z까지 이메일들이 순서대로 저장이 됩니다.



이메일이 순서대로 저장돼있으니까 특정 이메일을 찾을 때는 이 순서를 이용해서 데이터를 찾을 수 있죠.

Clustered 인덱스는 일반적으로 조회 속도가 굉장히 빠르다는 장점이 있습니다. 하지만 데이터를 특정 순서대로 저장했기 때문에 Clustered 인덱스는 하나밖에 만들 수 없습니다. 지금은 데이터를 이미 이메일 순서대로 정렬해놨는데, 여기서 이메일 순서로도 정렬돼있으면서, 동시에 이름 순서대로 정렬할 수는 없죠.

책으로 비유를 하면, 책 가장 뒤에 따로 인덱스를 만들어놓는 게 아니라, 영한 사전 같이 애초부터 단어들을 알파벳 순서대로 저장하는 겁니다.

이렇게 하면 찾고 싶은 영어 단어는 빠르게 찾을 수 있겠지만, 이미 알파벳 순서대로 책을 만들어놨기 때문에, 사전 안에서 특정 한글 단어를 찾고 싶을 때는 선형 탐색을 사용해야 합니다.

**Non-clustered 인덱스**

Non-Clustered 인덱스는 로우들이 실제 저장된 순서 자체는 건들지 않고, 위에서 얘기했던 거처럼 아예 데이터베이스 다른 곳에 컬럼 값들의 순서를 저장하는 방법입니다.



이메일에 대한 non-clustered 인덱스를 만들면 데이터베이스의 다른 곳에 이런 식의 데이터를 저장하는 거죠. 각 이메일에 대해서 해당 row의 유저가 저장된 주소를 저장합니다. 그래서 원하는 이메일을 찾았을 때, 주소를 사용해서 바로 원하는 유저 row를 찾을 수 있죠.

Non-Clustered 인덱스는 실제 테이블과 무관하게 **순서**를 저장하기 때문에 개수의 제한이 없습니다. 그 어떤 column에 대해서도 non-clustered 인덱스를 만들 수 있습니다.

아무래도 실제 테이블과 다른 곳에 저장돼있기 때문에 데이터를 찾는 게 clustered 인덱스보다 조금 느리다는 단점을 갖고 있습니다.

Non-clustered 인덱스는 얘기했던 것과 같이 책의 색인, 또는 인덱스랑 비슷합니다. 책 내용은 그대로 유지하면서, 따로 개념들을 정렬해놔서, 언제든지 원하는 개념을 빠르게 찾을 수 있도록 하죠.