

DB

11 인덱스(Index)

주요 질문

11.

인덱스

- ✓ DBMS Index 에 대해서 설명해 보세요.
- ✓ DBMS Index 에 사용되는 알고리즘 유형에 대해서 설명해 보세요.
- ✓ DBMS Index 만들었지만, Index Scan 을 못 타는 경우는 어떤 경우가 있을까요?

인덱스(Index)의 개념

- 데이터베이스 테이블에 대한 검색 성능의 속도를 높여주는 자료구조
- 특정 컬럼에 인덱스를 생성하면, 해당 컬럼의 데이터들을 정렬하여 별도의 공간에 데이터의 물리적 주소와 함께 저장

- 목차 -

- Ch01_01 강사소개및기술면접주요팁
- Ch02_01 DB분야 주요 질문 사항
- Ch02_02 데이터 독립성
- Ch02_03 DBMS 정의
- Ch02_04 DBMS 질의어
- Ch02_05 트랜잭션 정의
- Ch02_06 조인의개념 및 종류
- Ch02_07 NoSql
- Ch02_08 데이터무결성
- Ch02_09 정규화
- Ch02_10 반정규화
- Ch02_11 인덱스

인덱스(Index)의 개념

- 데이터베이스 테이블에 대한 검색 성능의 속도를 높여주는 자료구조
- 특정 컬럼에 인덱스를 생성하면, 해당 컬럼의 데이터들을 정렬하여 별도의 공간에 데이터의 물리적 주소와 함께 저장



Query

Index(목차)

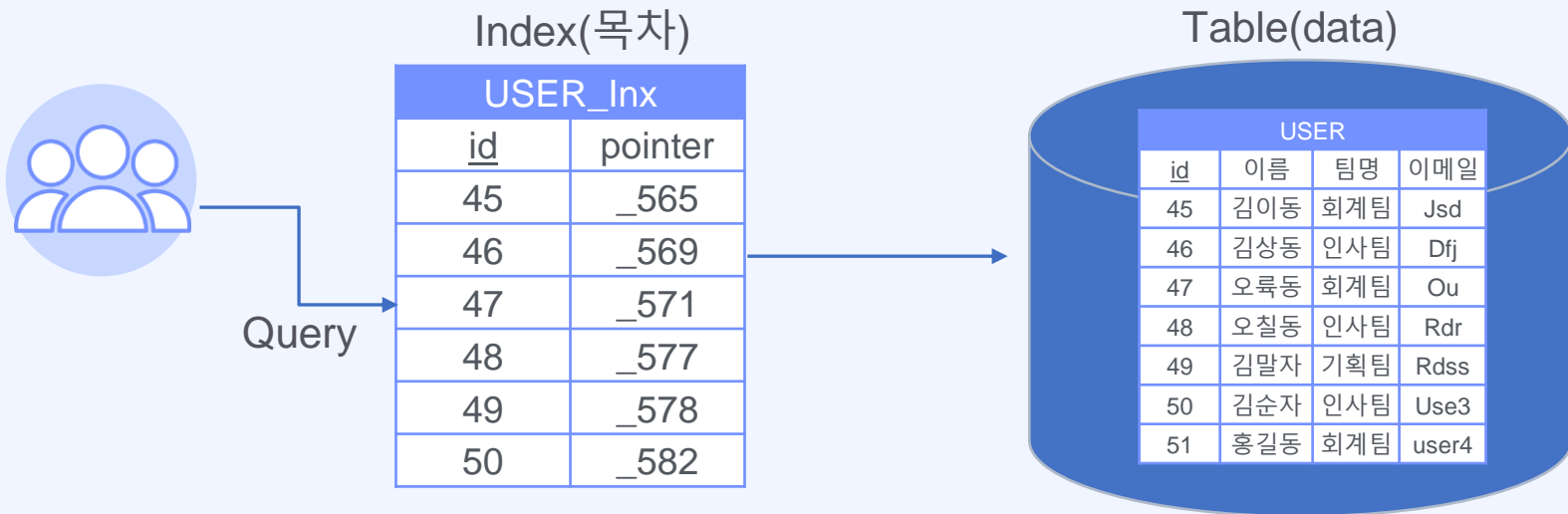
USER_Inx	
id	pointer
45	_565
46	_569
47	_571
48	_577
49	_578
50	_582

Table(data)

USER			
id	이름	팀명	이메일
48	김이동	회계팀	Jsd
50	김상동	인사팀	Dfj
47	오륙동	회계팀	Ou
46	오칠동	인사팀	Rdr
49	김말자	기획팀	Rdss
55	김순자	인사팀	Use3
51	홍길동	회계팀	user4

인덱스(Index)의 개념

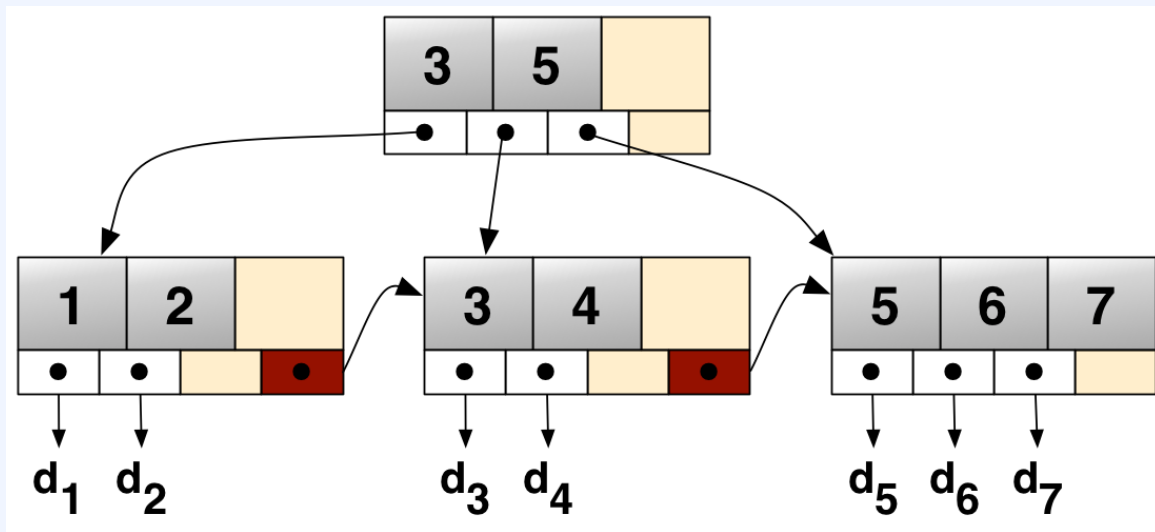
- 데이터베이스 테이블에 대한 검색 성능의 속도를 높여주는 자료구조
- 특정 컬럼에 인덱스를 생성하면, 해당 컬럼의 데이터들을 정렬하여 별도의 공간에 데이터의 물리적 주소와 함께 저장



인덱스(Index)의 알고리즘

11. 인덱스

- B+트리, B*트리 알고리즘으로 구현



인덱스(Index)를 사용하면 좋은 경우

- 데이터 규모가 큰 테이블
- 삽입(INSERT), 수정(UPDATE), 삭제(DELETE) 작업이 자주 발생하지 않는 컬럼
- WHERE나 ORDER BY, JOIN 등이 자주 사용되는 컬럼
- 데이터의 중복도가 낮은 컬럼 (분포도가 좋음)

인덱스(Index)를 사용하면 안 좋은 경우

- 데이터 규모가 작은 테이블
- 삽입(INSERT), 수정(UPDATE), 삭제(DELETE) 작업이 자주 발생하는 컬럼
- 데이터의 중복도가 높은 컬럼 (분포도가 나쁨. Eg. 성/별)
- 추가적인 데이터 저장소 필요 (Cost 발생)

인덱스 구조 방식

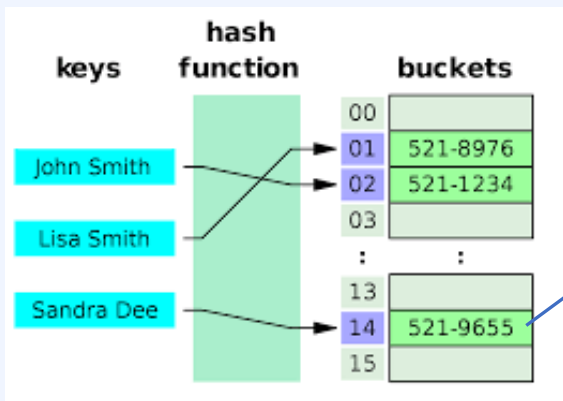
11.

인덱스

인덱스 구조	설명
트리 기반 구조	<ul style="list-style-type: none"> OLTP 범위 검색 자주 사용 B+트리, Root, Leaf 노드 구조
해시 기반 구조	<ul style="list-style-type: none"> OLTP 키 검색 자주 사용 버킷, 해시함수, 해시 테이블
비트맵 기반 구조	<ul style="list-style-type: none"> DW, Mart 데이터 검색 주로 사용 비트맵 인덱스

해시 기반 구조 방식

11.
인덱스



USER			
id	이름	팀명	이메일
21	John Smith	회계팀	Jsd
50	김상동	인사팀	Dfj
47	오륙동	회계팀	Ou
46	오칠동	인사팀	Rdr
49	김말자	기획팀	Rdss
55	김순자	인사팀	Use3
51	홍길동	회계팀	user4

출처 : <http://wiki.hash.kr/index.php/%ED%95%B4%EC%8B%9C%ED%85%8C%EC%9D%B4%EB%B8%94>

비트맵 기반 구조 방식

11. 인덱스

부품			
부품번호	부품명	색상	크기
21	부품1	블루	S
50	부품2	레드	M
47	부품3	레드	L
46	부품4	블루	NULL
49	부품5	그린	NULL
55	부품6	레드	L

블루	1	0	0	1	0	0
레드	0	1	1	0	0	1
그린	0	0	0	0	1	0
NULL	0	0	0	0	0	0

- 첫 번째와 마지막 비트의 rowid만을 갖고 있다가 테이블 액세스가 필요할 때면 각 비트가 첫 번째 비트로부터 떨어져 있는 상대적인 거리를 이용해 rowid값을 환산한다.
- 데이터 블록 사이즈 단위 고정되어 있음
- BITMAP 압축해서 사용

KEY	START	END	BITMAP
블루	1.2	122.2	100100 .. 1000
레드	1.2	122.2	011001 .. 0001
그린	1.2	122.2	000010 .. 1100
NULL	1.2	122.2	000000 .. 1001

비트맵 기반 구조 연산

SELECT * FROM 부품 WHERE (크기='S' OR 크기=NULL)
AND 색상='GREEN'

크기

S	1	0	0	0	0	0
---	---	---	---	---	---	---

OR

크기

NULL	0	0	0	1	1	0
------	---	---	---	---	---	---

AND

색상

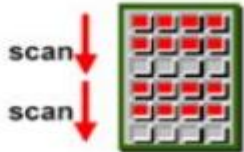
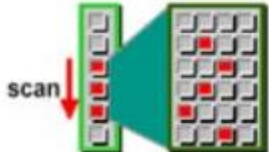
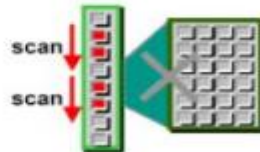
그린	0	0	0	0	1	0
----	---	---	---	---	---	---

Bitwise 연산 수행

인덱스 스캔 방식

11.

인덱스

구분	Full table scan	Index Scan	Fast full index scan
특징	1) 순차적 블록 액세스 2) 멀티 블록 I/O 및 병렬화 가능	1) 인덱스 블록 액세스 후 ROWID 를 통해 데이터 블록 획득 2) 비 순차적인 블록 액세스	1) 질의에 필요한 모든 컬럼이 인덱스에 포함된 경우 2) 멀티 블록 I/O 및 병렬화 가능
방식			
사용범위	- 저용량 데이터 조회 - 한 행의 15% 이상을 검색하는 경우	- 대용량 데이터의 조회 - 한행의 15% 이하의 검색을 하는 경우	

인덱스 스캔을 못 타는 경우

11. 인덱스

종류	예시
형변화	SELECT reg_date FROM table_name WHERE TO_CHAR(reg_date , 'YYYYMMDD') = '20130909';
NULL, NOT NULL	SELECT column_name FROM table_name WHERE column_name IS NULL
부정연산	SELECT column_name FROM table_name WHERE column_name != 30; (NOT EXISTS)
Like 연산	SELECT column_name FROM table_name WHERE column_name LIKE '%S%';
OR 조건	SELECT * FROM table1 t1, table t2 WHERE (t1. name1 = t2.name1 OR t1.name2 = t2.name2) and t1.code='103'