12강 시퀀스와 인덱스

시퀀스의 개념

- 오라클에서 필수로 개체 무결성을 위해서 중복된 값을 가지면 안되는 컬럼이 존재한다. –기본키
- 그러나 데이터를 입력하는 사용자로부터 기본키를 입력받으면 데이터 중복 가능성이 생기게 된다. 그러면 오류의 발생 확률이 높아지므로 적절하지 않다.

- 오라클에서는 기본키가 유일한 값을 가지도록 자체적으로 중복 없는 값을 생성해 내서 오류를 최소화 할수 있는데 이런 기능을 시퀀스 라고 한다.

시퀀스의 개념

- 시퀀스의 기본 특징은 숫자로 만들어진다는 것이다.
  - 초기값이 있고 증가값이 존재해서 데이터 생성시 증가값만큼 증가하며 값을 생성해 낸다.

  - 예제 : 시작값이 10이고 10씩 증가하는 시퀀스 (seq_sample)을 만들어 보자

```
CREATE SEQUENCE seq__sample
START WITH 10
INCREMENT BY 10;
```

시퀀스의 개념

- 시퀀스를 작성하기 위한 옵션
  - start with : 시퀀스 시작을 지정
  - increment by : 연속적인 시퀀스 번호의 증가치를 지정할 때 사용
  - maxvalue : 시퀀스의 최대값을 지정(nomaxvalue 지정시
    오름차순: 10^{27} 내림차순: -1)
  - minvalue : 시퀀스의 최소값을 지정 (nominvalue 지정시
    내림차순: 10^{27} 오름차순: -1)
  - cycle : 시퀀스의 값이 최대까지 증가하면 처음부터 다시 시작한다.
    nocycle 지정시 시퀀스의 값이 최대 이후 증가를 하면 에러 발생

- 앞서 만든 시퀀스의 정보를 사전에서 알아보자

```
SELECT sequence_name, min_value, max_value, increment_by, cycle_flag
FROM user_sequences
WHERE sequence_name = 'SEQ__SAMPLE';
```

시퀀스의 개념

- 시퀀스의 현재 값을 알아보기 : currval
- 시퀀스의 다음 값을 알아보기 : nextval

- 시퀀스가 처음 시작되면 currval에는 아무런 값이 없다.

첫 currval에 값을 할당하기 위해서는 nextval을 사용해야 하는데
nextval로 새로운 값을 생성한 후 이 값으로 currval에 대체를 함

=> 한번도 사용하지 않은 시퀀스에 currval을 사용하면 에러 발생

```
SELECT seq__sample.currval from dual;
```

시퀀스의 개념

- currval과 nextval을 사용할 수 있는 경우
  - 서브 쿼리가 아닌 SELECT문
  - INSERT문의 SELECT 절
  - INSERT문의 VALUE 절
  - UPDATE문의 SET절

- currval과 nextval을 사용할 수 없는 경우
  - VIEW의 SELECT절
  - DISTINCT 키워드가 있는 SELECT문
  - GROUP BY, HAVING, ORDER BY 절이 있는 SELECT문
  - SELECT, DELETE, UPDATE의 서브 쿼리
  - CREATE TABLE, ALTER TABLE 명령의 DEFAULT 값

시퀀스의 실습

- 실습 : NAXTVAL로 새로운 값 생성

```
SELECT seq__sample.nextval from dual;
```

- 실습 : CURRVAL로 현재 값 알아보기

```
SELECT seq__sample.currval from dual;
```

시퀀스를 기본키에 접목

- 실습 : 부서번호를 만들 시퀀스를 만들어 본다(시작값10, 증가값 10)

```
CREATE SEQUENCE seq_dno
START WITH 10
INCREMENT BY 10;
```

- 실습 : 부서정보를 저장할 빈 테이블 만들기(구조복사)

```
CREATE TABLE dept_SEQ
AS
SELECT * FROM department where 0=1;
```

- 실습 : 시퀀스를 이용해서 데이터를 삽입해 본다.

dname	loc
ACCOUNTING	NEW YORK
RESEARCH	DALLAS
SALES	BOSTON

```
INSERT INTO dept_SEQ
VALUES (seq_dno.nextval, 'ACCOUNTING','NEW YORK');
INSERT INTO dept_SEQ
VALUES (seq_dno.nextval, 'RESEARCH','DALLAS');
INSERT INTO dept_SEQ
VALUES (seq_dno.nextval, 'SALES','BOSTON');
```

시퀀스의 수정 및 제거

- 시퀀스로 오라클의 다른 개체와 마찬가지로 수정 삭제가 가능하다.
- ALTER SEQUENCE 시퀀스이름 ~~

- 단 생성과 차이점이 있다면 START WITH를 사용할 수 없다는 점이다.

=> 이미 생성되서 사용중인 시퀀스를 시작값을 변경할 수 없다

- 제거는 DROP명령을 사용한다.
- DROP SEQUENCE 시퀀스 이름

시퀀스의 실습

- 실습 : 부서 번호 시퀀스의 최대값을 확인하기

```
SELECT sequence_name, min_value, max_value, increment_by, cycle_flag
FROM user_sequences
WHERE sequence_name = 'SEQ_DNO';
```

- 실습 : 부서 번호 시퀀스의 최대값(50)을 변경하기 -> 다시 확인

```
ALTER SEQUENCE seq_dno
MAXVALUE 50;
```

- 실습 : 부서 번호 시퀀스 삭제

```
DROP SEQUENCE seq_dno;
```

인덱스의 개념

- 인덱스란 검색 속도를 향상시키기 위해서 사용하는 도구이다.
- 필요에 의해 직접 생성할 수도 있지만 무결성 확보를 위해 수시로 데이터를 검색하는 용도로 사용하는 기본키나 유니크 키는 자동으로 인덱스가 생성된다.

- 실습: EMPLOYEE와 DEPARTMENT에 생성된 인덱스를 살펴보자

```
SELECT index_name, table_name, column_name
FROM user_ind_columns
WHERE table_name IN('EMPLOYEE', 'DEPARTMENT');
```

- 실습 : ename 컬럼에 인덱스를 생성하고 확인해 보자

```
CREATE INDEX idx_employee_ename
ON employee(ename);
```

- 실습 : 위에서 생성한 인덱스를 삭제해 보자

```
DROP INDEX idx_employee_ename;
```

인덱스의 장단점

인덱스의 장점	인덱스의 단점
검색 속도가 빨라질 수 있다(항상 그런 것은 아니다.	처음 인덱스를 생성하는데 많은 시간이 소요될 수 있다
검색 속도 향상으로 인해 쿼리의 부하가 줄어들어 전체 시스템 성능을 향상된다.	인덱스도 데이터베이스 공간을 차지하므로 추가적인 공간이 필요해진다. (대략 10% 의 추가 공간이 필요하다)
	데이터 변경 작업이 많을 수록 오히려 성능이 나빠질 수 있다

인덱스를 사용해야 하는 경우	인덱스를 사용하지 말아야 하는 경우
테이블에 행의 수가 많을 때	테이블의 행의 수가 적을 때
WHERE 문에 해당 칼럼이 많이 사용될 때	WHERE 문에 해당 칼럼이 자주 사용되지 않을 때
검색 결과가 전체 데이터의 2%~4%정도 일 때	검색 결과가 전체 데이터의 10%~15%이상일 때
JOIN에 자주 사용되는 칼럼이나 NULL을 포함하는 칼럼이 많을 때	테이블에 DML 작업이 많은 경우 즉, 입력/수정/삭제 등이 자주 일어 날때

인덱스 – Btree(균형 트리)

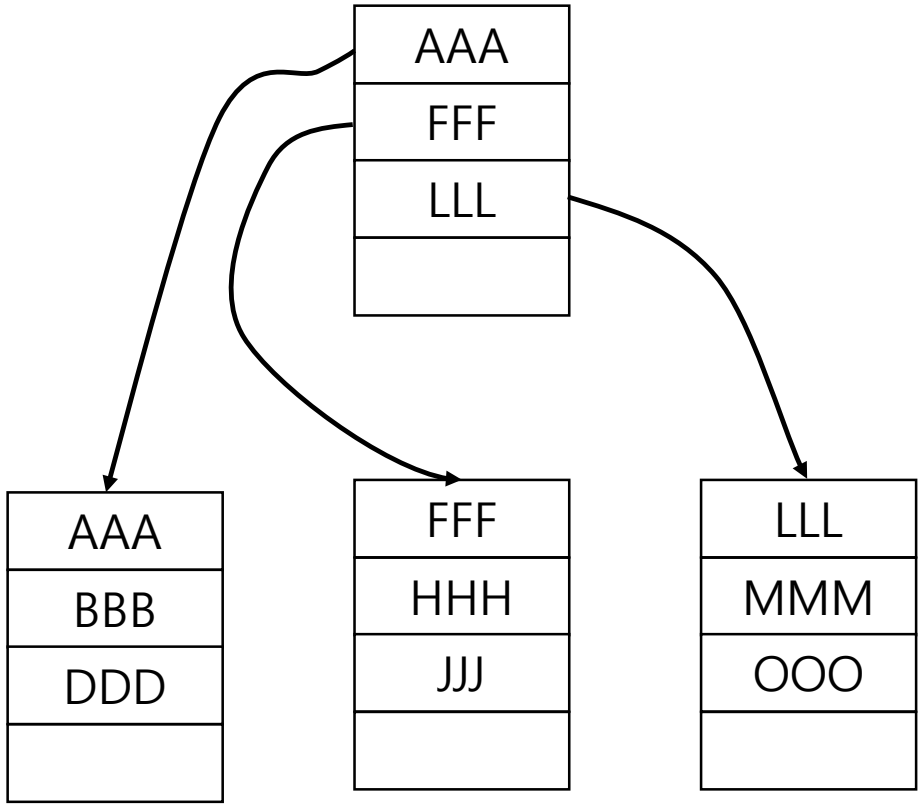
- B-Tree는 자료 구조에 나오는 범용적으로 사용되는 데이터 구조이다.
- 주로 인덱스를 표현하거나 그 외에도 많이 사용된다.

- 주요 용어
  - 노드 : 데이터가 존재하는 공간 => 갈라지는 '마디'
  - 루트 : 최 상위 노드를 의미한다.
  - 리프 노드 : 최 하위 노드를 의미한다.
  - 가지블록(중간노드) : 중간에 배치되는 노드를 의미한다.
  - 블록 : 노드와 같은 말로 사용되며 최소 8Kbyte로 구성되며 아무리 작은 데이터라도 8kByte는 사용됨을 의미한다.

- 예제: AAA, BBB, DDD, FFF, HHH, JJJ, LLL, MMM, OOO

위 데이터가 저장된 그림을 살펴보자

인덱스 – Btree(균형 트리)



- 일반 구조일 경우 MMM이라는 데이터를 찾기 위해서는 AAA부터 순서대로 찾아야 한다.(A,B,D,F,H,J,L,M 8회)

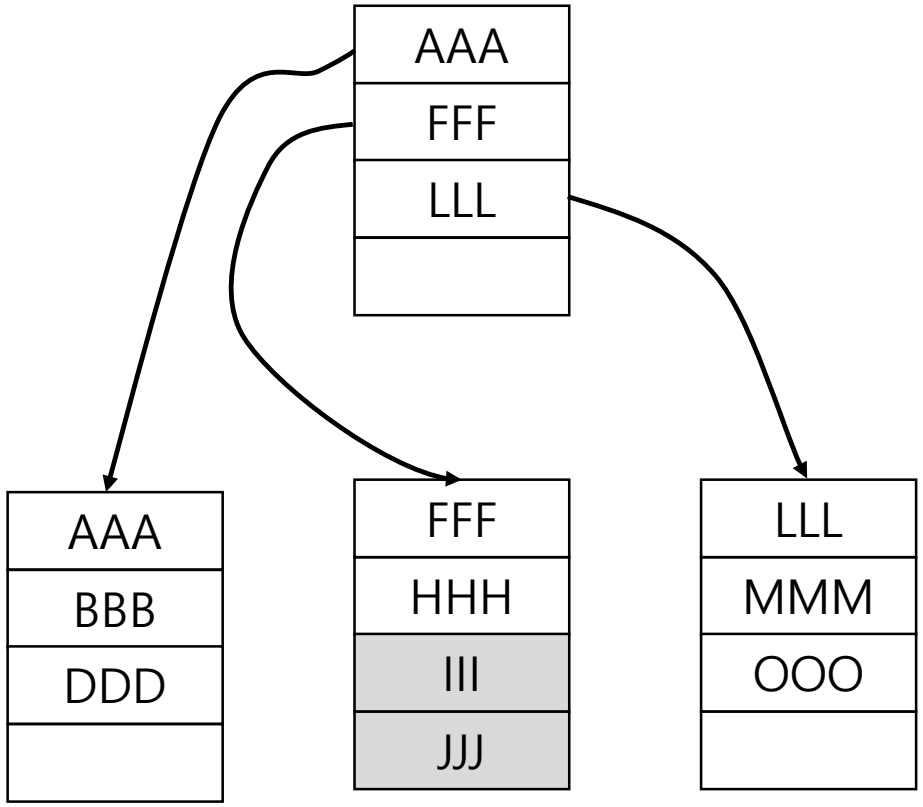
- 이런 검색 방법을 '전체 테이블 검색'이라고 한다.

- 왼쪽과 같은 B-tree 구조라면 MMM을 찾기 위해서 우선 루트 노드부터 검색

- AAA,FFF,LLL ->세번째 리프 노드로 이동 -> LLL,MMM 5회만에 검색

=> 5건의 데이터 검색, 2개의 블록

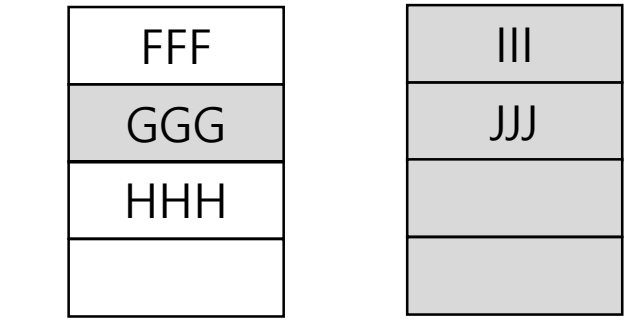
인덱스 – Btree(균형 트리)



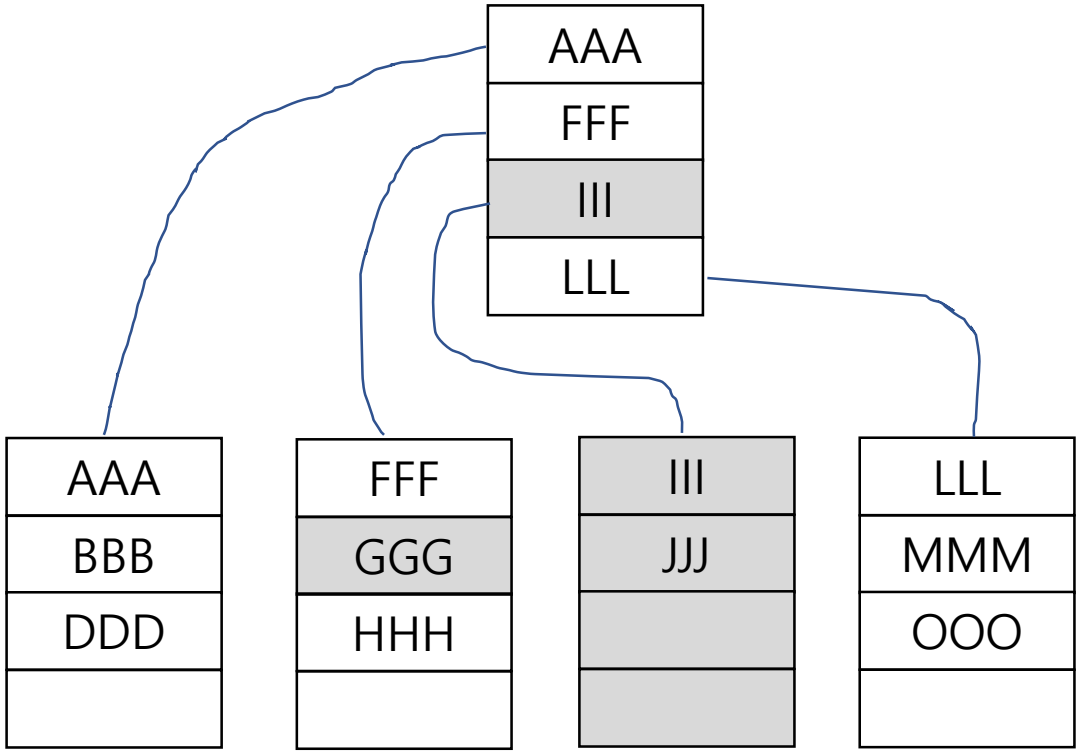
- 검색에서 B-Tree가 효율적임을 알았지만
- 데이터 변경(insert,update,delete)등에는 성능이 나빠질 수 있다.
- 특히 insert과정에서 인덱스 분할로 인해 성능이 자주 느려지는 데 그 이유을 알아보자

- 예: III 데이터가 삽입된다면 HHH가 속한 노드에서 HHH와 JJJ사이에 III가 추가된다.
- 이 때에는 속도의 저하가 없다.

인덱스 – Btree(균형 트리)



두번째 리프 노드 신규 노드

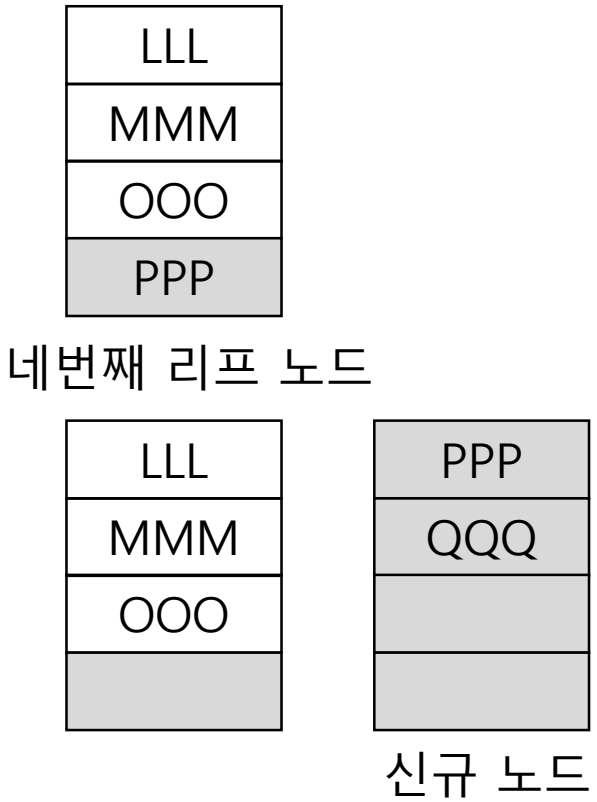


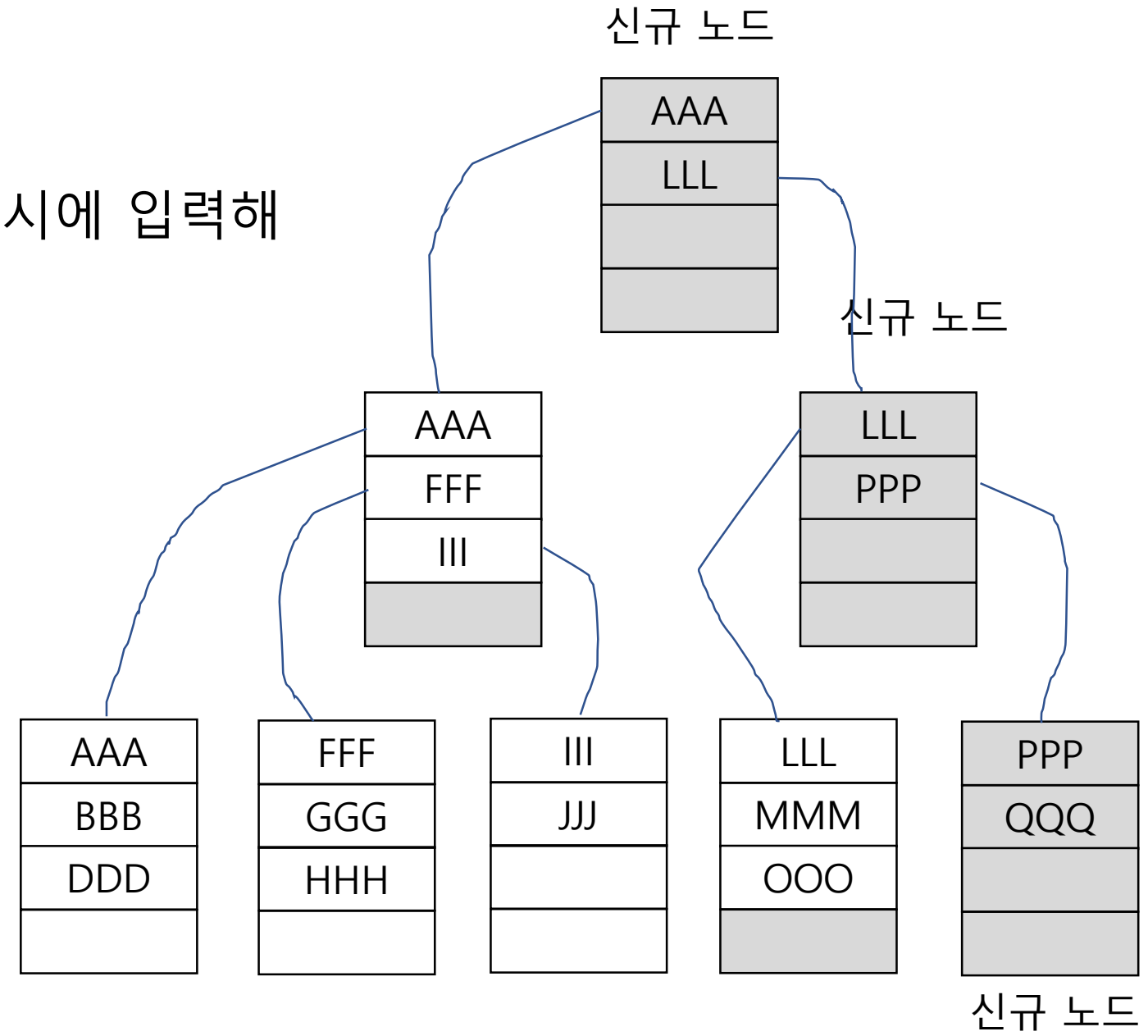
- 그러나 새롭게 GGG데이터를 넣어보자.
- 순서상 FFF와 HHH사이에 들어가야 하는데 두번째 리프 노드에는 빈 공간이 없다.
- 이럴때 인덱스 분할 작업이 일어난다.
  1) 신규 노드 생성
  2) 두번째 리프노드를 적절하게 분할해서 신규노드를 채운다.
  3) 신규 노드를 가르킬 데이터를 루트노드에 등록한다.

- 인덱스 분할작업은 노드블록이 약 80%가 채워지면 분할 작업에 들어간다.

인덱스 – Btree(균형 트리)

- 이번엔 PPP, QQQ 두개를 동시에 입력해 봅니다.





인덱스 – Btree(균형 트리)

- employee테이블을 복사해서 emp_idx 테이블을 만들어본다.

```
CREATE TABLE emp_idx
AS
SELECT eno,ename FROM employee;
```

- 인덱스없는 상태의 구조는 다음과 같이 저장된다(실제 하나의 블록에 전부 저장될 것이다.)

7876	ADAMS
7499	ALLEN
7698	BLAKE
7782	CLARK

7902	FORD
7900	JAMES
7566	JONES
7839	KING

7654	MARTIN
7934	MILLER
7788	SCOTT
7369	SMITH

7844	TURNER
7521	WARD

- ROWID를 포함해서 출력해보자
  - ROWID는 실제 물리주소로 보면 된다.(앞에서 부터 6자리는 데이터객체번호, 3자리는 파일번호, 다음 6자리는 블록 번호, 마지막 3자리는 행번호)

```
SELECT ROWID, eno, ename FROM emp_idx ORDER BY ENO ASC;
```

인덱스 – Btree(균형 트리)

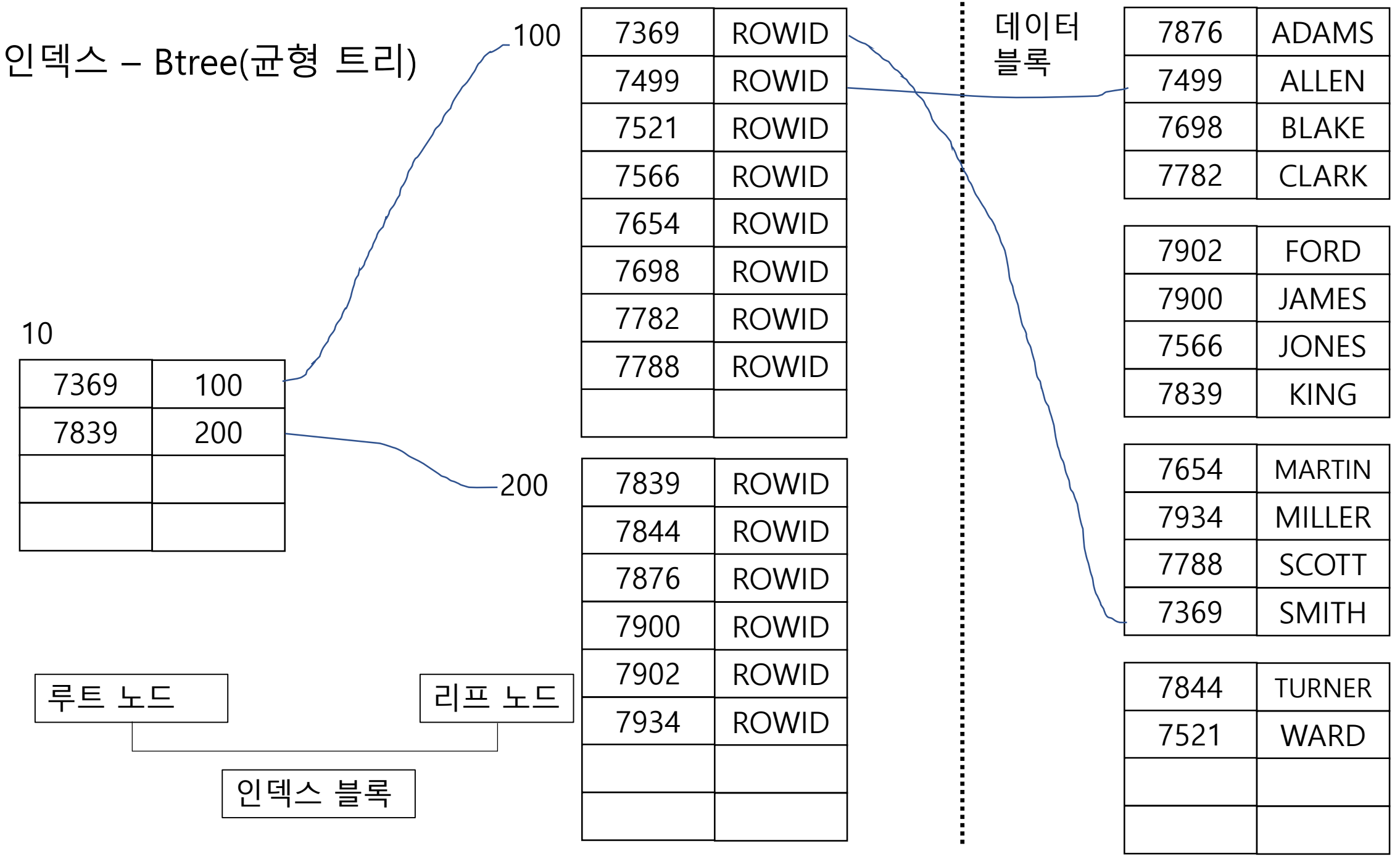
- 고유 인덱스를 구성해보자(기본키를 설정하면 자동으로 고유 인덱스가 구성된다.)

```
ALTER TABLE emp_idx
ADD CONSTRAINT emp_idx_eno_pk PRIMARY KEY (eno);
```

- 샘플 데이터가 매우 적으므로 데이터를 다시 확인해도 테이블 자체의 변화는 없을 것이다.

- 그러나 내부적으로 다음과 같은 구조를 가질 것이다.

인덱스 – Btree(균형 트리)



인덱스 – Btree(균형 트리)

- 데이터가 별로 없어서 인덱스 있는 것과 없는 것이 차이가 없지만 데이터 량이 많으면 매우 빠르게 검색할 수 있다.

- 인덱스 특징
  - 인덱스 생성시 기존 데이터 블록은 그냥 두고 인덱스 블록을 생성
  - 인덱스의 리프 블록은 데이터블록의 주소값을 가진다.
  - 데이터 수정,입력,삭제에는 인덱스가 없을 때보다 느리다.
  - 인덱스는 여러 개 생성 가능하다. 그러나 남용하는 경우 오히려 시스템 성능 저하를 가져오므로 반드시 필요한 열에만 작성한다.

  - 인덱스 검색을 위해서는 반드시 일차 조건(WHERE)에 인덱스가 적용된 열 이름이 나와야 한다. 물론 인덱스 생성한 열 이름이 나온다고 해서 반드시 인덱스를 활용하는 것은 아니다.