Beyond Promise

SQL 학습 소모임 1강

1. 개발자가 SQL 까지 잘해야 하는가?

- ✓ 개발자들이 SQL 에 대해서 좀 더 잘 알고 사용했으면 하는 심정에서……
- ✓ 많은 프로젝트에서 SQL 작성은 DBA 의 role 이 아니고 이미 개발자의 몫…
- ✓ 어플리케이션의 성능 문제 발생시 잘못된 SQL 이 원인인 경우가 생각보다 많다.

➢ DB 성능 저하 요인

DB Call, Network, I/O, Logging, Parsing, Lock …

가장 중요한 것은 SQL !!!

1. 개발자가 SQL 까지 잘해야 하는가?

가장 효율적인 접근 방법을 설명하자면…

➢ 데이터 모델 튜닝 > Application 튜닝 > Database 튜닝 > System 튜닝

- 데이터 모델 튜닝 : 모델링 관점에서 최적의 모델링 방안을 도출하는 과정
- Application 튜닝 : 일반적으로 SQL 튜닝 및 DB IO 개선 작업
- Database 튜닝 : DB 파라미터, 메모리 구성 등의 개선 작업
- System 튜닝 : OS 레벨에서의 튜닝

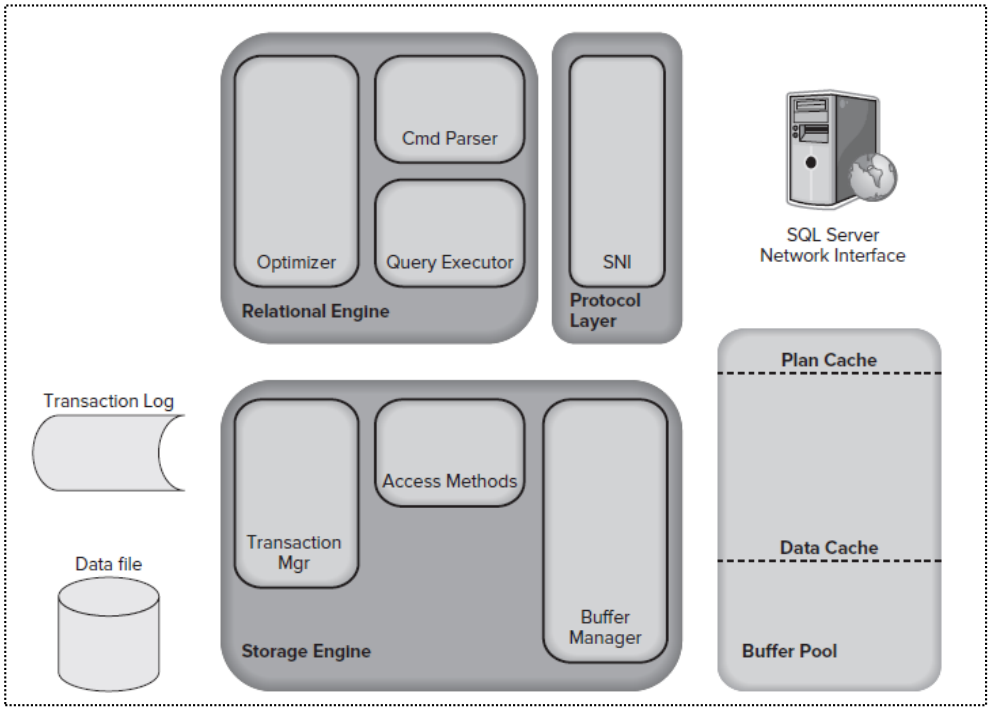
일반적으로 이미 분석 설계 작업이 끝나고 구축 과정에서 데이터 모델 변경 작업은 진행하기가 어려움.
결국 Application 튜닝에 많은 노력이 들어가게 됨.

1. 개발자가 SQL 까지 잘해야 하는가?

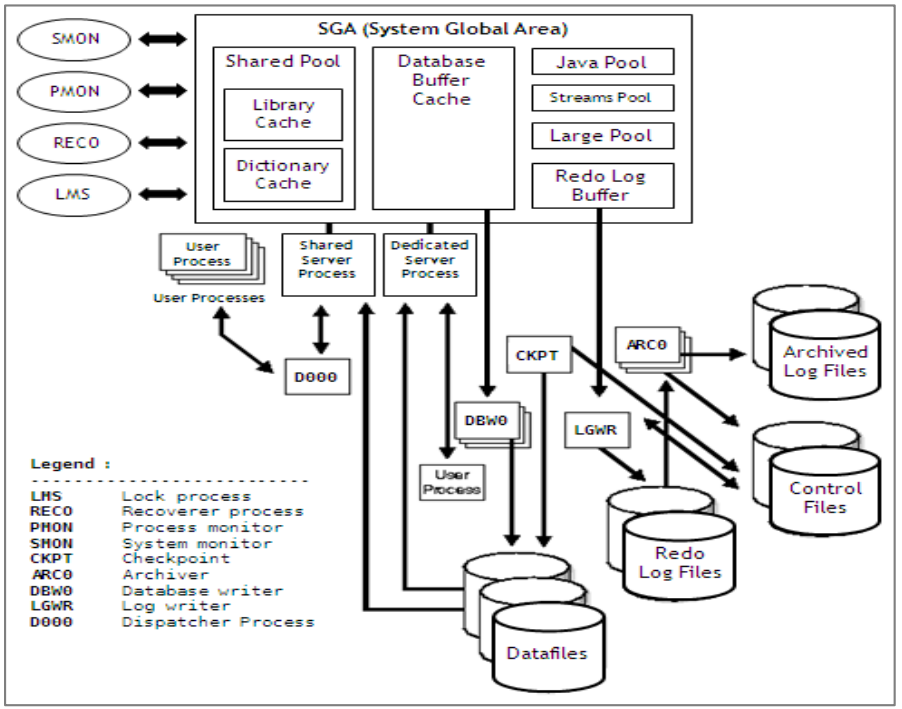
개발자는 어느 정도까지 알면 되는지...

- 간단한 DB 내부 구조 : 너무 깊이 들어가지 않아도 대충은 알고 가자.
- SQL 처리 절차 : 뭐..간단하니까... 알아둬야 함.
- 인덱스의 원리 : 아주 잘 알아야 됨
- 실행 계획 : 이것도 알아둬야 함
- Lock 의 원리 : 이것도 잘 알아야 됨
- 부분 범위 처리 : 이것도 알아둬야 함.
- 집합 개념의 SQL 작성법 : 이건 아주 아주 잘 알아야 됨.

2. 간단한 DB 내부 구조



SQL Server

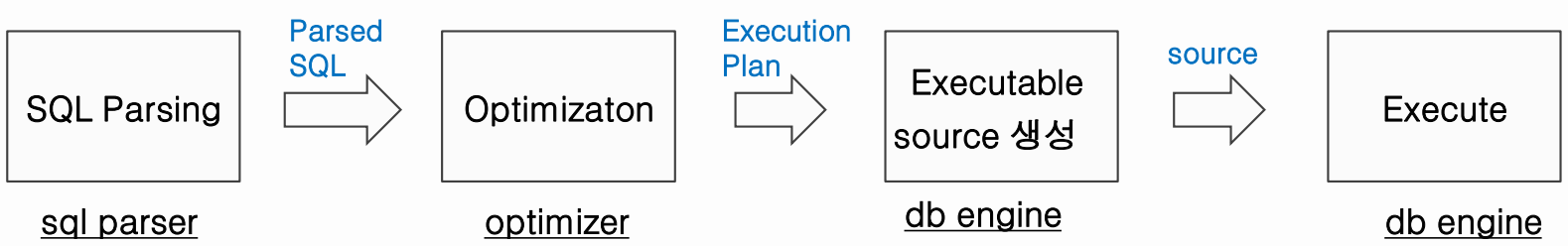


Oracle

- Buffer 영역 : 데이터 cache, sql 및 plan cache 역할
- 데이터 저장 영역 : 실제 데이터가 저장되는 영역
- DB 엔진 영역 : 옵티마이저, 쿼리 실행, 로그 관리, 트랜잭션 관리 등.

3. SQL 처리 절차

SQL 은 파싱과 최적화 과정을 거쳐 실행된다.



- soft parse : SQL 을 파싱하고 해당 SQL 에 대한 실행 계획이 cache 에 있을 때 기존 실행계획을 갖고 바로 실행하는 것.
- hard parse : SQL을 파싱하고 실행계획이 cache 에 없을 때 최적화 단계 이하 모든 단계를 걸치는 것.
  hard parse 는 시스템 자원을 상대적으로 많이 사용하고 수행시간이 오래 걸리는 원인.
  프로그램을 작성할 때 바인드 변수 사용하는 것을 권장. (나중에 좀 더 자세히 다룸)

```
select  name, job from tbl_member where id = :cust_id
```

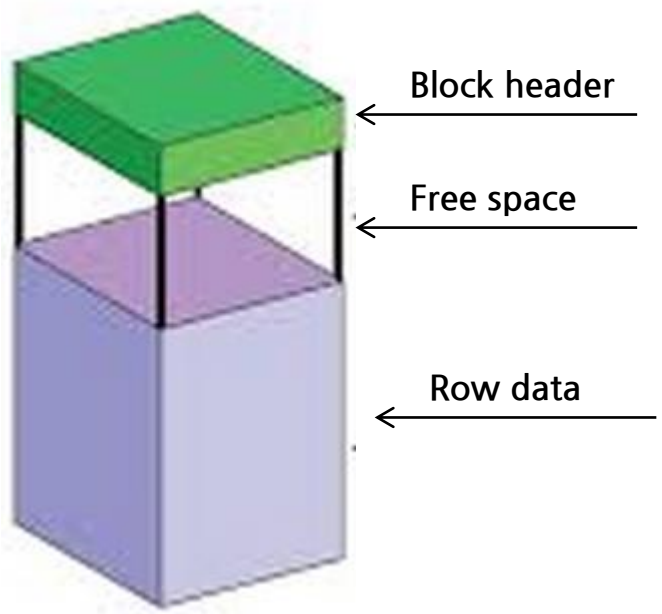
→ 1000 번 수행

Call	Count	Cpu	Elapsed	…
Parse	1000	0.01	0.03	…
Execute	1000	…	…	…
Fetch	3000	…	…	…
Total	5000	…	…	…

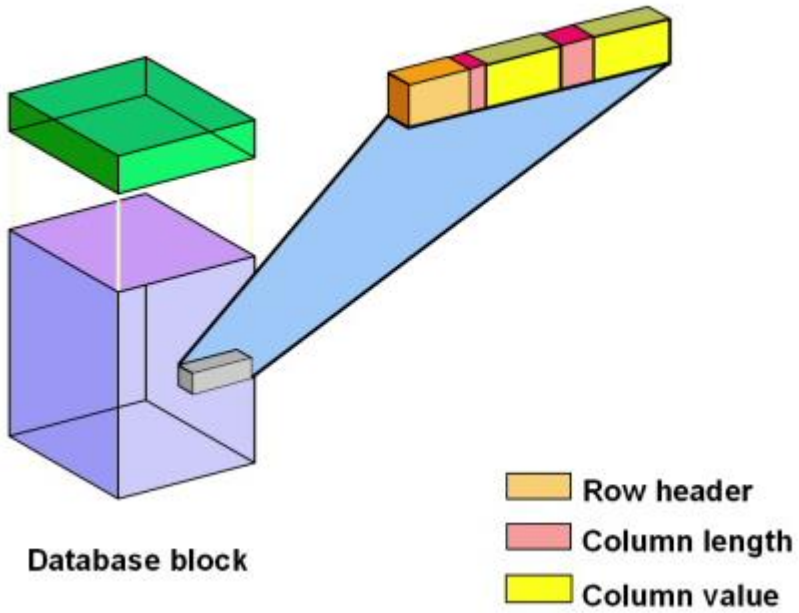
Misses in library chach during parse : 1

4. 데이터 저장 구조

▪ Block 구조



Row 구조



- Block (Page) : 데이터 row 가 저장되는 공간으로서 I/O 의 기본 단위
- Oracle 은 block, SQL Server 는 page 라고 호칭함.
- 디폴트 크기는 2k

5. 인덱스를 만들어 보자 (1) - Non-clustered index

인덱스를 만드는 과정을 간단하게 시뮬레이션 해보면서 특징을 이해한다.

1. 일반적인 데이터 블록

Data block (90)	
Id	Name
2	LMNO
1	ABC
6	BCED

Data block (91)	
Id	Name
5	JKLM
7	UACT
20	CDEF

Data block (92)	
Id	Name
4	HKIK
27	PFOU
21	ZAKA

Data block (93)	
Id	Name
3	RSTE
12	QABC
23	QZAB

Data block (94)	
Id	Name
19	YHUO
13	BDEDF
22	OPQR

Data block (95)	
Id	Name
8	SABA
18	DEFG
10	FCDQ

Data block (96)	
Id	Name
11	CABD
9	DACA
24	CCAD

Data block (97)	
Id	Name
16	EABC
25	EGZD
14	FACD

Data block (98)	
Id	Name
15	DBAZ
26	KABC
17	LABC

2. 인덱스를 만들기 위하여 인덱스를 구성하는 컬럼으로 데이터 정렬

```
create index idx_01 on tableA (Name)
```

ABC
BCED
BDEDF
CABD
CCAD
CDEF
DACA
DBAZ
DEFG

EABC
EGZD
FACD
FCDQ
JKLM
KABC
LABC
LMNO
OPQR
QABC

QZAB
RSTE
SABA
UACT
YHUO

* non clustered index 는 주로 SQL Server 에서 사용하는 용어이며 일반 DB 의 index 와 비슷하다.

* SQL Server 는 이외에 clustered index 를 제공하며 이에 대해서는 뒤에 소개한다.

5. 인덱스를 만들어 보자 (2) - Non-clustered index

1. 정렬된 인덱스 키값을 기반으로 인덱스 leaf block 을 채워간다.
2. leaf block 을 채우다가 일정 부분 채워지면 새로운 leaf block 이 만들어지고 거기에 계속 인덱스 키값을 채워간다.

ABC
BCED
BDEDF
CABD
CCAD
CDEF
DACA
DBAZ
DEFG

3. 이러한 과정을 반복하여 모든 인덱스 키값을 인덱스 leaf block 에 채운다.
4. leaf block 의 인덱스는 인덱스 키 값과 테이블 row 에 대한 row id 를 갖고 있다.

97번 블록의 첫번째 row 를 찾아가라.

Leaf block (100)		
90	2	ABC
90	3	BCED
94	2	BDEDF

↔

Leaf block (101)		
96	1	CABD
96	3	CCAD
91	3	CDEF

↔

Leaf block (102)		
96	2	DACA
98	1	DBAZ
95	2	DEFG

↔

Leaf block (103)		
97	1	EABC
97	2	EGZD
97	3	FACD

.....

Data block (90)	
Id	Name
2	LMNO
1	ABC
6	BCED

Data block (91)	
Id	Name
5	JKLM
7	UACT
20	CDEF

Data block (92)	
Id	Name
4	HKIK
27	PFOU
21	ZAKA

Data block (93)	
Id	Name
3	RSTE
12	QABC
23	QZAB

Data block (94)	
Id	Name
19	YHUO
13	BDEDF
22	OPQR

Data block (95)	
Id	Name
8	SABA
18	DEFG
10	FCDQ

Data block (96)	
Id	Name
11	CABD
9	DACA
24	CCAD

Data block (97)	
Id	Name
16	EABC
25	EGZD
14	FACD

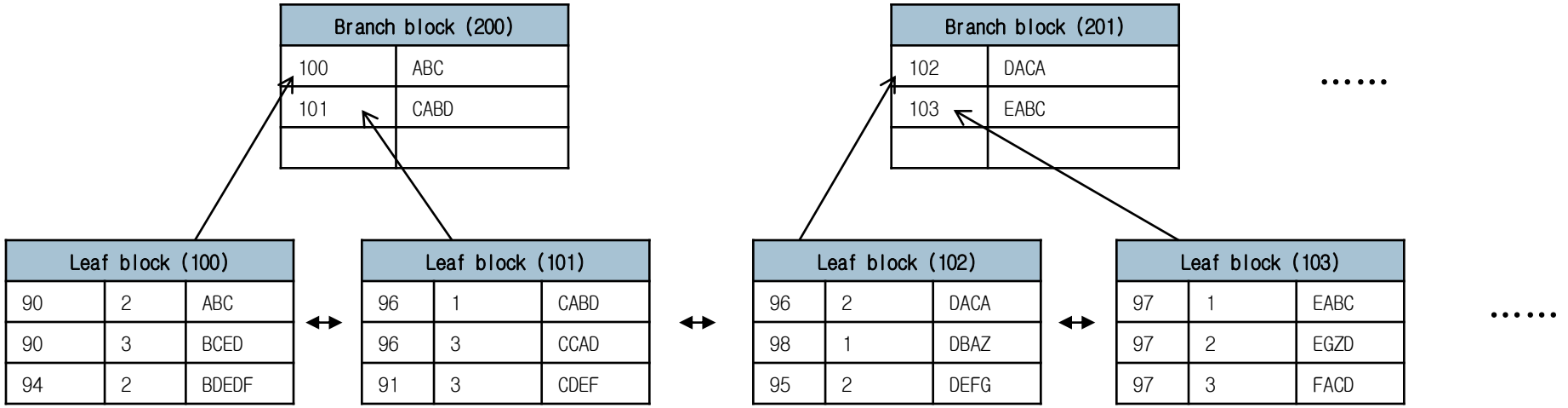
Data block (98)	
Id	Name
15	DBAZ
26	KABC
17	LABC

일반적인 방법을 설명한 것이고 DBMS 마다 약간의 차이는 존재한다.

5. 인덱스를 만들어 보자 (3) - Non-clustered index

1. leaf block 의 첫번째 row 값들로 branch block 을 채워나간다.
2. branch block 이 다 채워지면 다시 그 상위레벨에서 branch block 을 만들어 준다.
3. 즉, branch block 은 n 개의 계층구조를 갖는다.

!! oracle 에서는 약간 다른 개념으로 만들지만 원리는 비슷하다.



Data block (90)

Id	Name
2	LMNO
1	ABC
6	BCED

Data block (91)

Id	Name
5	JKLM
7	UACT
20	CDEF

Data block (92)

Id	Name
4	HKIK
27	PFOU
21	ZAKA

Data block (93)

Id	Name
3	RSTE
12	QABC
23	QZAB

Data block (94)

Id	Name
19	YHUO
13	BDEDF
22	OPQR

Data block (95)

Id	Name
8	SABA
18	DEFG
10	FCDQ

Data block (96)

Id	Name
11	CABD
9	DACA
24	CCAD

Data block (97)

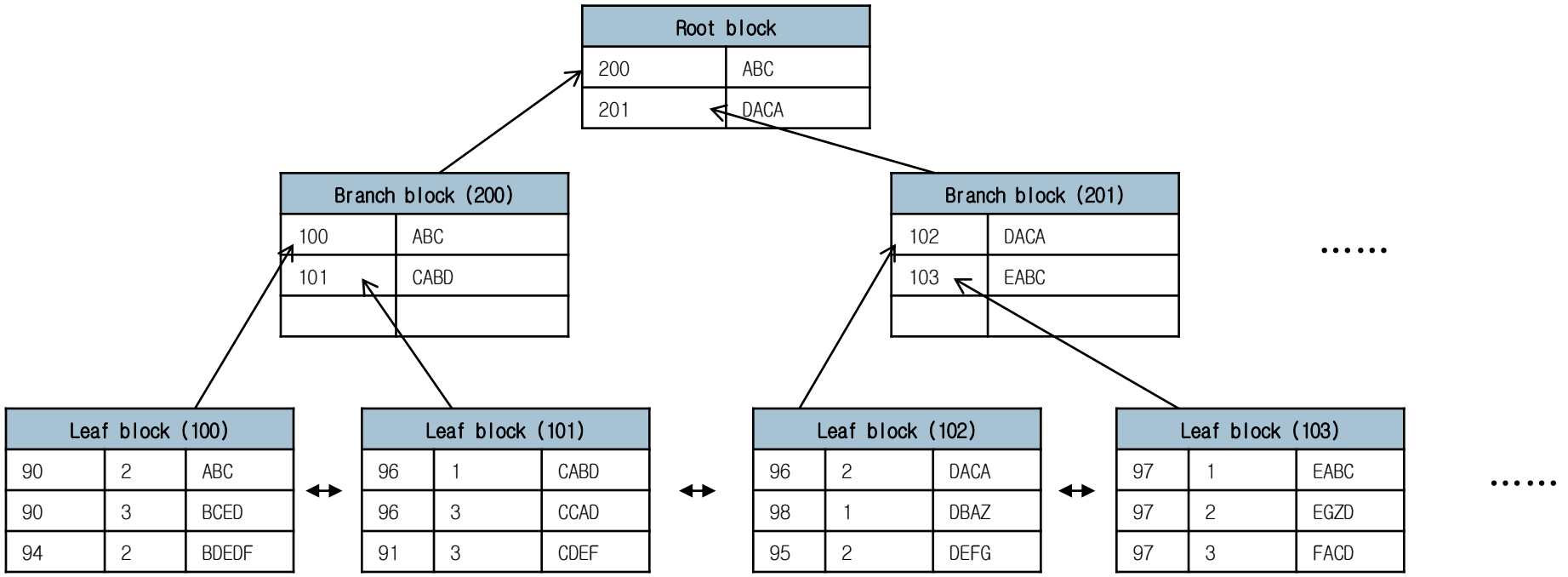
Id	Name
16	EABC
25	EGZD
14	FACD

Data block (98)

Id	Name
15	DBAZ
26	KABC
17	LABC

5. 인덱스를 만들어 보자 (4) - Non-clustered index

1. 이러한 과정을 반복하다 보면 최종적으로 마지막에 최상단에 하나의 block 이 만들어 지며 이를 root block 이라 한다.



Data block (90)

Id	Name
2	LMNO
1	ABC
6	BCED

Data block (91)

Id	Name
5	JKLM
7	UACT
20	CDEF

Data block (92)

Id	Name
4	HKIK
27	PFOU
21	ZAKA

Data block (93)

Id	Name
3	RSTE
12	QABC
23	QZAB

Data block (94)

Id	Name
19	YHUO
13	BDEDF
22	OPQR

Data block (95)

Id	Name
8	SABA
18	DEFG
10	FCDQ

Data block (96)

Id	Name
11	CABD
9	DACA
24	CCAD

Data block (97)

Id	Name
16	EABC
25	EGZD
14	FACD

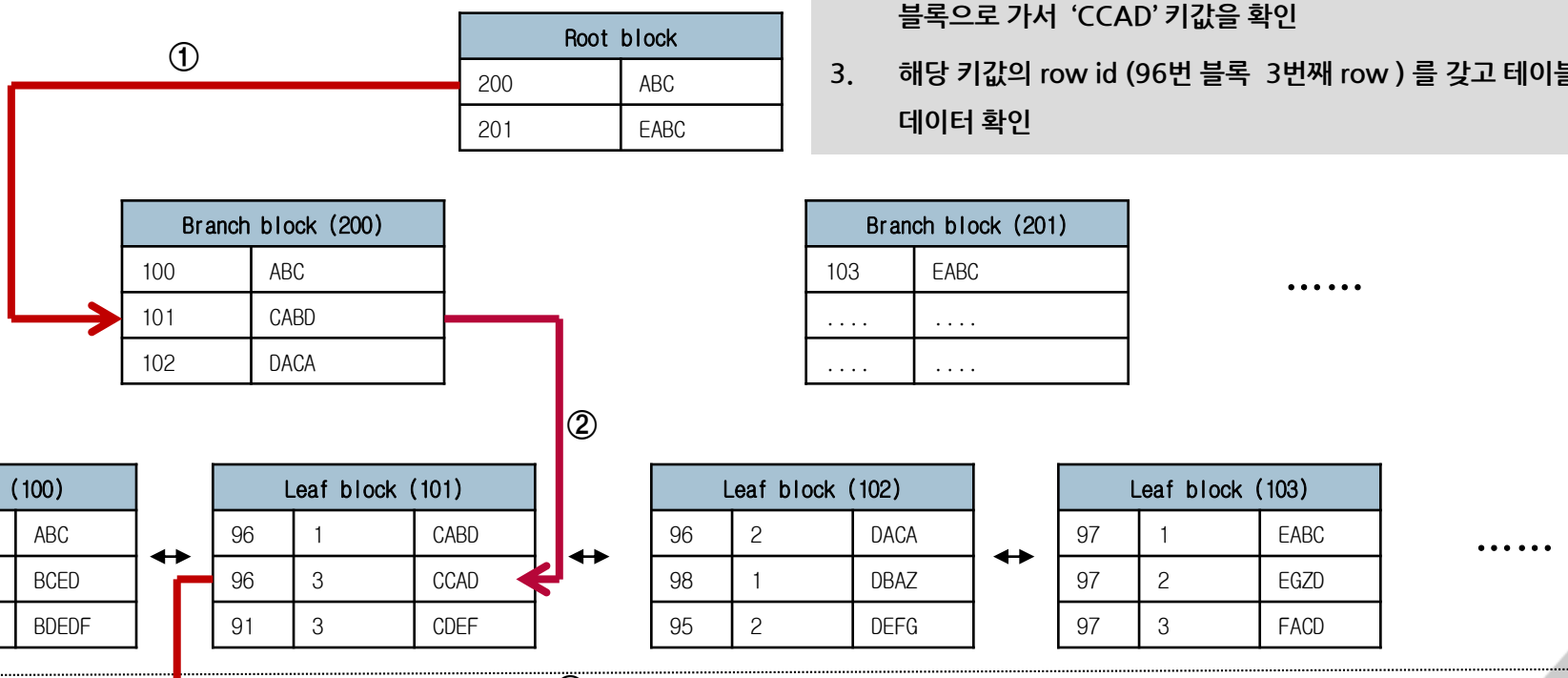
Data block (98)

Id	Name
15	DBAZ
26	KABC
17	LABC

6. 인덱스 동작 원리 (1) - Non-clustered index

```
select id  from tableA where name = 'CCAD'
```

1. root block에서 'CCAD' 는 'EABC' 보다 작으므로 200 번 블록으로 이동.
2. 200번 블록에서 'CCAD' 는 'DACA' 보다 작으므로 101번 블록으로 가서 'CCAD' 키값을 확인
3. 해당 키값의 row id (96번 블록 3번째 row) 를 갖고 테이블에서 데이터 확인



Data block (90)

Id	Name
2	LMNO
1	ABC
6	BCED

Data block (91)

Id	Name
5	JKLM
7	UACT
20	CDEF

Data block (92)

Id	Name
4	HKIK
27	PFOU
21	ZAKA

Data block (93)

Id	Name
3	RSTE
12	QABC
23	QZAB

Data block (94)

Id	Name
19	YHUO
13	BDEDF
22	OPQR

Data block (95)

Id	Name
8	SABA
18	DEFG
10	FCDQ

Data block (96)

Id	Name
1	CABD
9	DACA
24	CCAD

Data block (97)

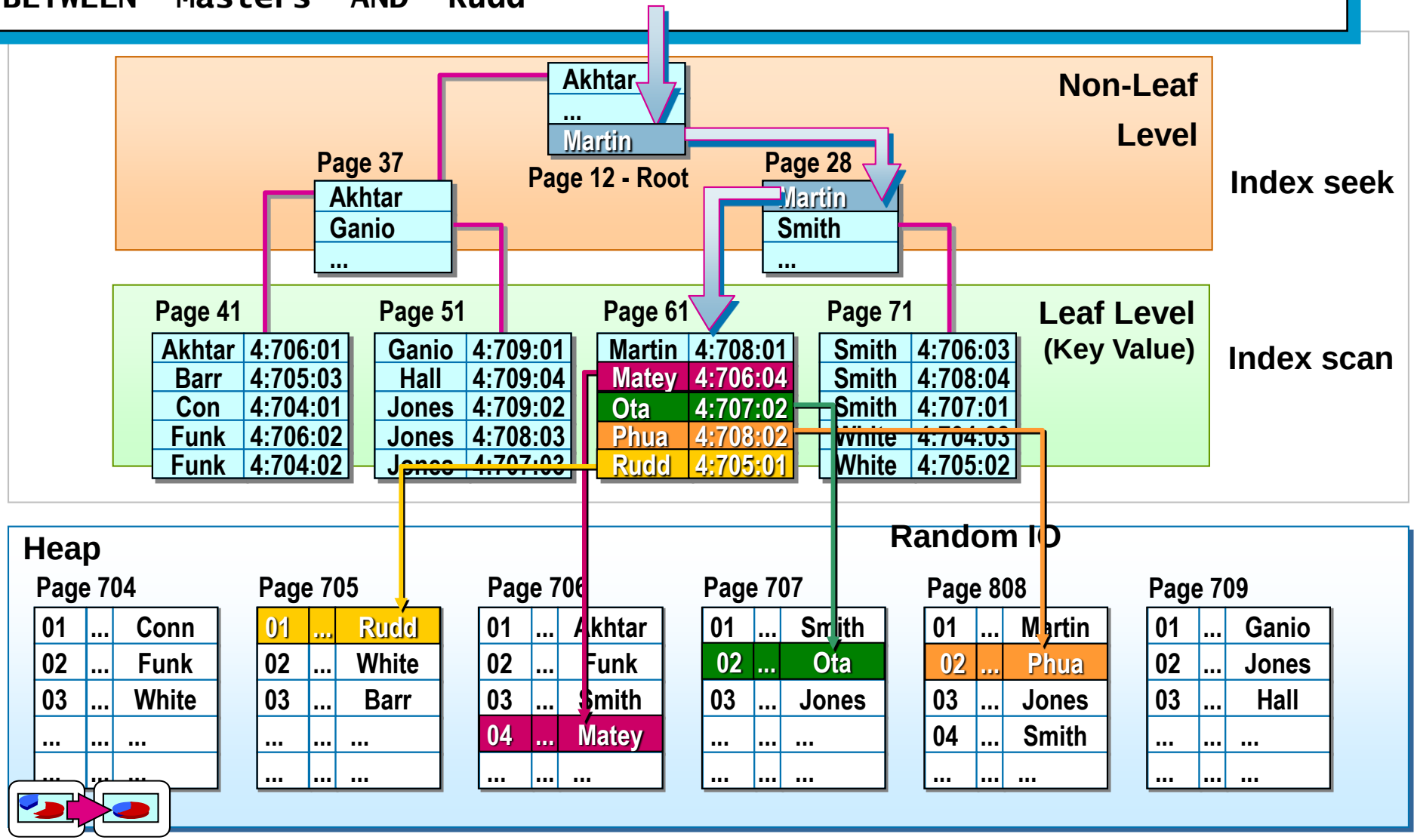
Id	Name
16	EABC
25	EGZD
14	FACD

Data block (98)

Id	Name
15	DBAZ
26	KABC
17	LABC

6. 인덱스 동작 원리 (2) - Non-clustered index

```
SELECT lastname, firstname FROM member WHERE lastname
BETWEEN 'Masters' AND 'Rudd'
```



7. 데이터 삽입시 인덱스 변화 (before) -Non clustered index

300

1	200
8	201
21	202

200

1	102	2
2	103	1
3	102	3
4	100	3
5	103	2
7	101	2

201

8	101	4
9	102	1
10	101	1
14	102	4
18	103	3
19	100	4

202

21	100	2
22	101	3
23	100	1

100

id	name	part
23	김경남	DEPT1
21	박재연	DEPT2
4	이상우	DEPT4
19	양건호	DEPT5

101

id	name	part
10	양재영	DEPT3
7	박동규	DEPT6
22	김지균	DEPT3
8	지태창	DEPT4

102

id	name	part
9	김순호	DEPT2
1	송수환	DEPT1
3	김선희	DEPT5
14	조영구	DEPT4

103

id	name	part
2	최규황	DEPT3
5	현종철	DEPT4
18	강수진	DEPT2

7. 데이터 삽입시 인덱스 변화 (After) - Non clustered index

데이터 삽입시 Index의 변화

1. 데이터 블록에서는 마지막 페이지에 append
2. 인덱스 leaf 페이지는 정렬되어 있으므로 중간에 값이 들어가야 되는 경우 split 발생
3. 순차적으로 branch 부터 root 까지 변경됨.

300

1	200
8	201
14	102
21	202

200

1	102	2
2	103	1
3	102	3
4	100	3
5	103	2
7	101	2

201

8	101	4
9	102	1
10	101	1
14	102	4

203

14	102	4
16	103	4
18	103	3
19	100	4

202

21	100	2
22	101	3
23	100	1

100

id	name	part
23	김경남	DEPT1
21	박재연	DEPT2
4	이상우	DEPT4
19	양건호	DEPT5

101

id	name	part
10	양재영	DEPT3
7	박동규	DEPT6
22	김지균	DEPT3
8	지태창	DEPT4

102

id	name	part
9	김순호	DEPT2
1	송수환	DEPT1
3	김선희	DEPT5
14	조영구	DEPT4

103

id	name	part
2	최규황	DEPT3
5	현종철	DEPT4
18	강수진	DEPT2
16	오유선	DEPT3

8. 인덱스를 만들어 보자 (1) - Clustered index

clustered index 는 SQL Server, MySQL 등에서 사용되는 인덱스 개념이다.

1. 일반적인 데이터 블록

Data block (90)	
Id	Name
2	LMNO
1	ABC
6	BCED

Data block (91)	
Id	Name
5	JKLM
7	UACT
20	CDEF

Data block (92)	
Id	Name
4	HKIK
27	PFOU
21	ZAKA

Data block (93)	
Id	Name
3	RSTE
12	QABC
23	QZAB

Data block (94)	
Id	Name
19	YHUO
13	BDEDF
22	OPQR

Data block (95)	
Id	Name
8	SABA
18	DEFG
10	FCDQ

Data block (96)	
Id	Name
11	CABD
9	DACA
24	CCAD

Data block (97)	
Id	Name
16	EABC
25	EGZD
14	FACD

Data block (98)	
Id	Name
15	DBAZ
26	KABC
17	LABC

```
create clustered  index idx_01 on tableA (Name)
```

2. 인덱스를 만들기 위하여 인덱스를 구성하는 컬럼으로 테이블 정렬

Data block (90)	
Id	Name
1	ABC
6	BCED
13	BDEDF

Data block (91)	
Id	Name
11	CABD
24	CCAD
20	CDEF

Data block (92)	
Id	Name
9	DACA
15	DBAZ
18	DEFG

Data block (93)	
Id	Name
16	EABC
25	EGZD
14	FACD

Data block (94)	
Id	Name
10	FCDQ
4	HKIK
5	JKLM

Data block (95)	
Id	Name
26	KABC
17	LABC
2	LMNO

Data block (96)	
Id	Name
22	OPQR
27	PFOU
12	QABC

Data block (97)	
Id	Name
23	QZAB
3	RSTE
19	SABA

Data block (98)	
Id	Name
7	UACT
19	YUHO
17	ZAKA

* non clustered index 에서는 인덱스 컬럼으로 데이터 정렬 후 인덱스 leaf 를 구성하였으나 clustered index 에서는 테이블 자체를 인덱스 컬럼 기준으로 정렬한다.

8. 인덱스를 만들어 보자 (2) - Clustered index

root block (300)	
300	ABC
301	FCDQ
302	UACT

branch block (300)	
200	ABC
201	DACA

branch block (301)	
202	FCDQ
203	DACA

branch block (301)	
204	UACT

branch block (200)	
90	ABC
91	DACA

branch block (201)	
92	DACA
93	EABC

branch block (202)	
94	FCDQ
95	KABC

branch block (203)	
96	QPQR
97	0ZAB

branch block (204)	
98	UACT

Data block (90)	
Id	Name
1	ABC
6	BCED
13	BDEDF

Data block (91)	
Id	Name
11	CABD
24	CCAD
20	CDEF

Data block (92)	
Id	Name
9	DACA
15	DBAZ
18	DEFG

Data block (93)	
Id	Name
16	EABC
25	EGZD
14	FACD

Data block (94)	
Id	Name
10	FCDQ
4	HKIK
5	JKLM

Data block (95)	
Id	Name
26	KABC
17	LABC
2	LMNO

Data block (96)	
Id	Name
22	0PQR
27	PF0U
12	QABC

Data block (97)	
Id	Name
23	QZAB
3	RSTE
19	SABA

Data block (98)	
Id	Name
7	UACT
19	YUH0
17	ZAKA

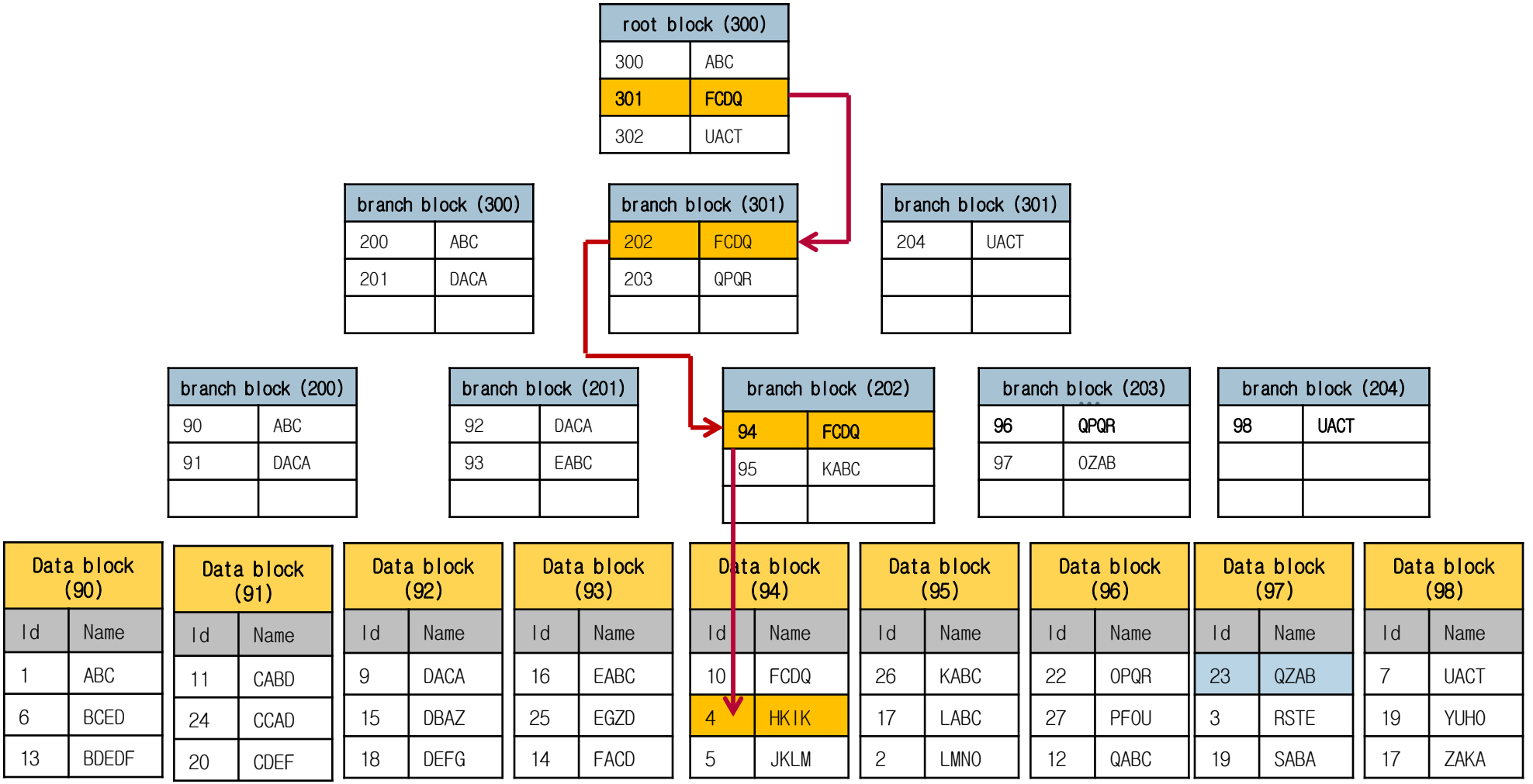
** leaf block 이면서 데이터 블록

1. 인덱스 컬럼으로 테이블 정렬
2. 결국 clustered index 의 leaf block 은 테이블 block
3. leaf block(table block 자체) 의
4. 첫번째 row 값들로 branch block 을 채워나간다.

이 과정을 반복하여 root block 까지 생성한다.

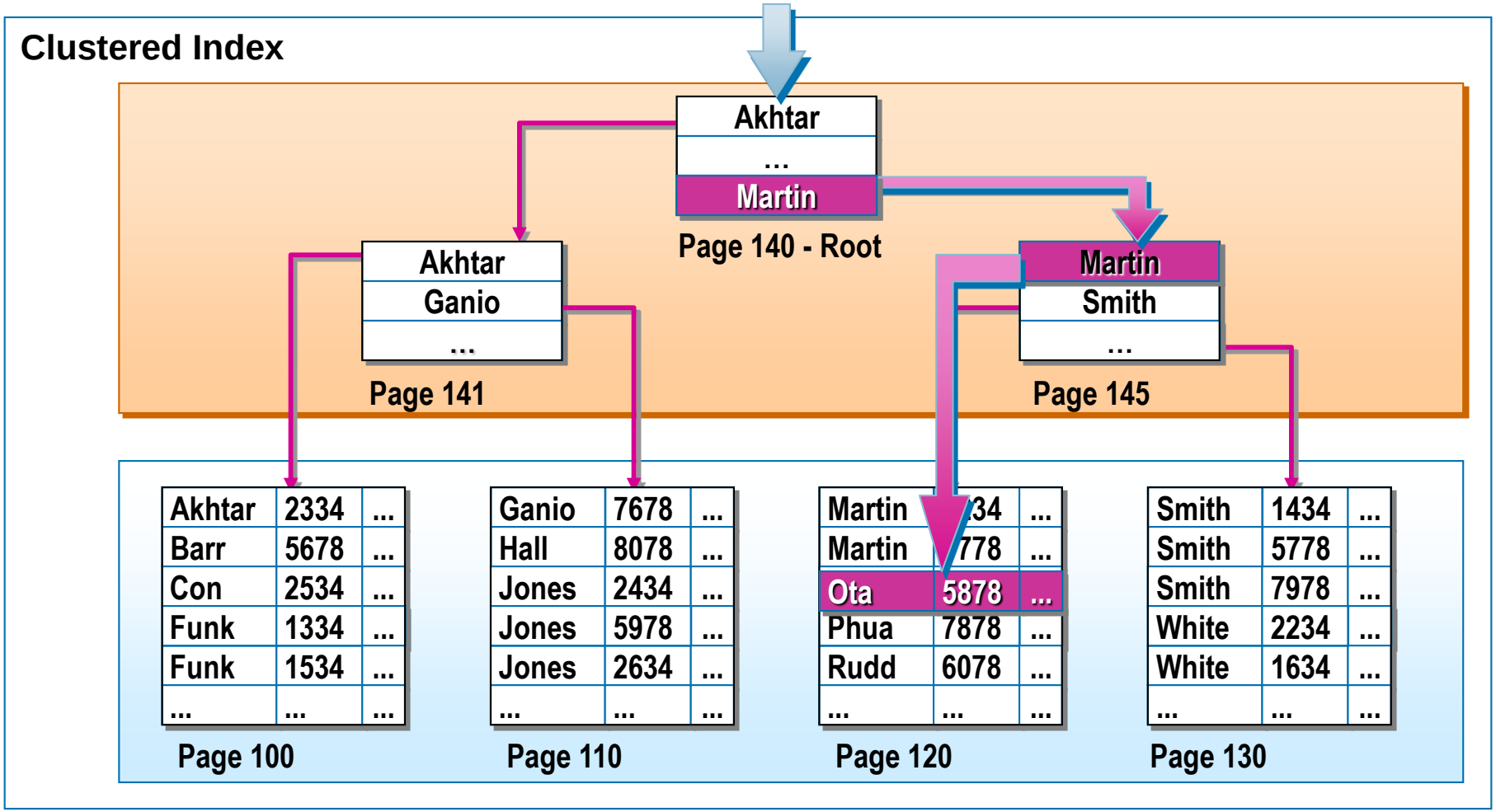
8. 인덱스를 만들어 보자 (2) - Clustered index

```
SELECT id, name from tbl where name = 'HKIK'
```



9. 인덱스 동작 원리 - Clustered index

```
SELECT lastname, firstname FROM member WHERE lastname = 'Ota'
```



클러스터 인덱스는 테이블에 몇 개까지 만들수 있을까요?

10. 잠깐만...

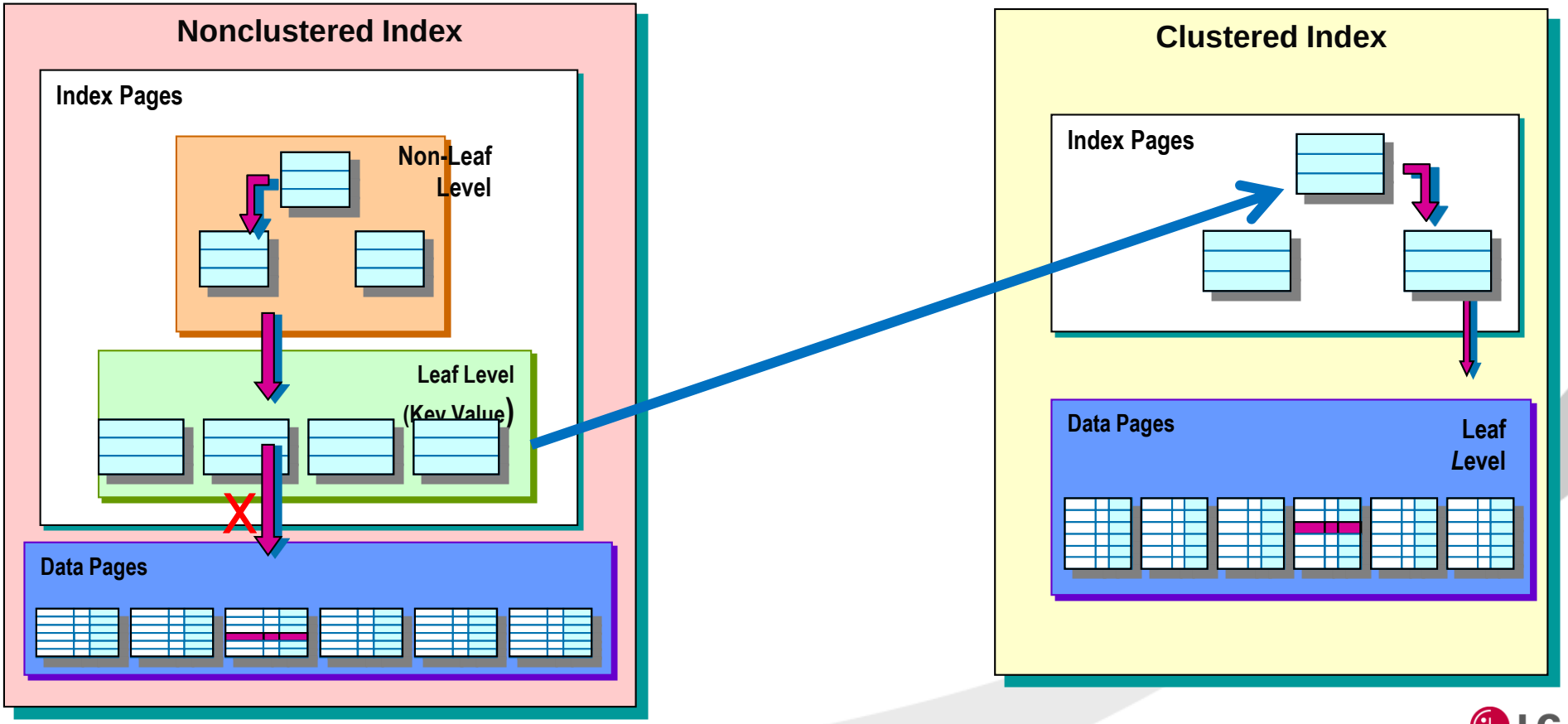
- **테이블에 1개 만 존재 가능 (data page chain 의 정렬 순서가 한 방향으로 만들어 지므로.)**
- **Data 자체가 물리적으로 Disk 드라이브에 정렬**
  - (주의) 물리적 저장소를 디스크 자체로 생각하지 말것. 클러스터 된 인덱스가 특정 순서로 디스크에 데이터를 유지한다면 데이터 변경이 무척 힘들것. 페이지가 꽉 차서 ,spilt 이 발생할 경우 이 페이지 이후의 모든 데이터들이 이동해야 함. 클러스터된 인덱스의 정렬 순서는 단순히 data page chain (double-linked list 구조) 이 이 순서로 되어 있다는 것을 의미함. SQL Server는 page chain을 따라가면서 데이터를 액세스하는것. 새 페이지가 추가될 때는 이 page chain에 있는 연결을 조절.

- **Clustered Index의 leaf노드는 실제 data page , 따라서 Pointer jump가 필요없고 디스크상에서 대량 범위 처리시(64KB이상) sequential I/O 작업으로 처리**
- **대량 범위 처리에 강점**

11. Clustered index 와 NonClustered index 의 혼용

- ✓ Clustered index 가 존재할 때 Non Clustered index 를 생성하면 Non Clustered index 에서는 rowid 대신에 Clustered index key 값을 갖게 됨.
- ✓ Select 문의 성능은 약간 저하되지만 DML 발생시 성능이 향상됨 … why? 뒤에서 설명.

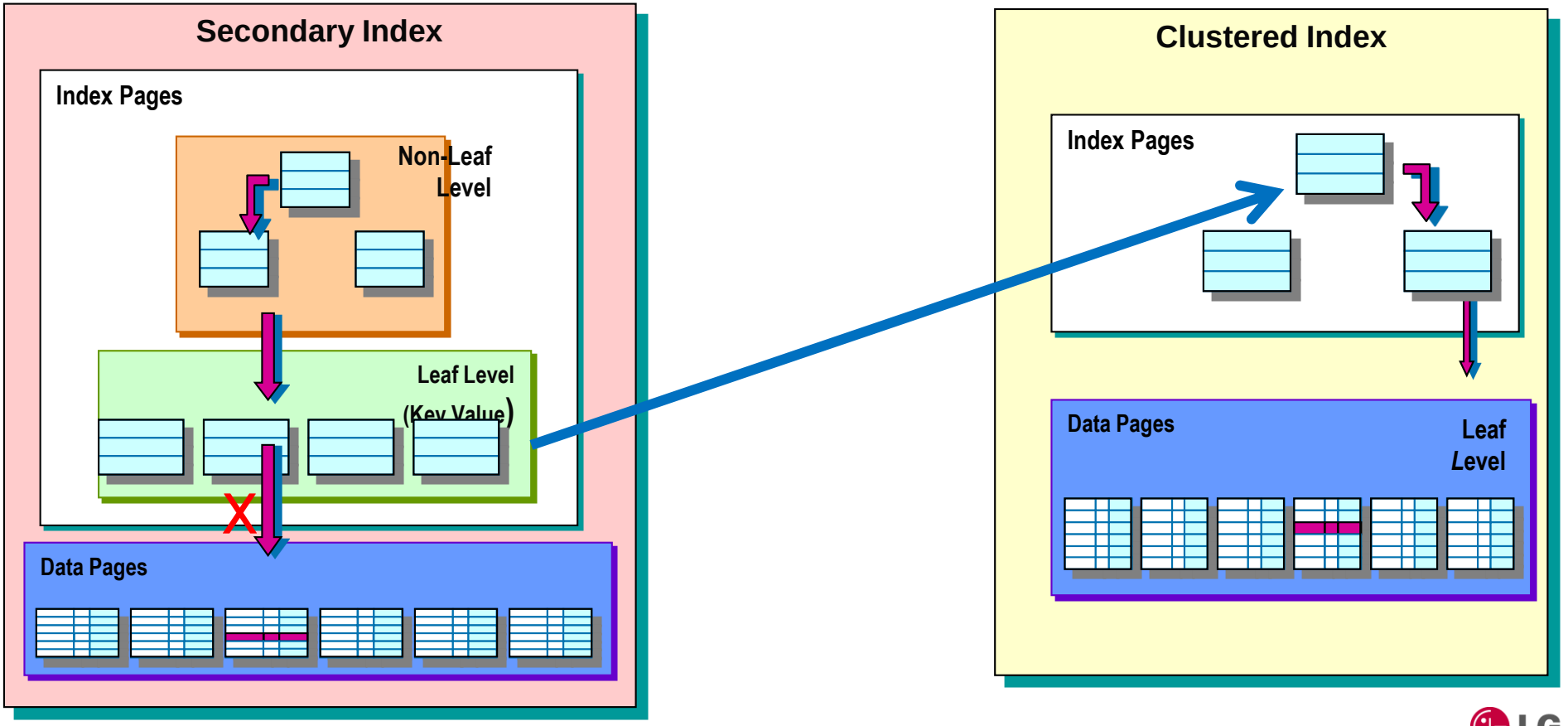
-



11. Clustered index 와 NonClustered index 의 혼용

- ✓ Clustered index 가 존재할 때 Non Clustered index 를 생성하면 Non Clustered index 에서는 rowid 대신에 Clustered index key 값을 갖게 됨.
- ✓ Select 문의 성능은 약간 저하되지만 DML 발생시 성능이 향상됨 ... why? 뒤에서 설명.

-



12. Clustered index 와 NonClustered index simulation

raw data 및 인덱스

100

id	name	part
23	김경남	DEPT1
21	박재연	DEPT2
4	이상우	DEPT4
19	양건호	DEPT5

101

id	name	part
10	양재영	DEPT3
7	박동규	DEPT6
22	김지균	DEPT3
8	지태창	DEPT4

102

id	name	part
9	김순호	DEPT2
1	송수환	DEPT1
3	김선희	DEPT5
14	조영구	DEPT4

103

id	name	part
2	최규황	DEPT3
5	현종철	DEPT4
18	강수진	DEPT2

- create clustered index idx_part on table(part)
  - part 테이블이 중복된 값이 있으면 내부적으로 4byte 값을 붙여 uniqui 하게 만듬 (SQL Server)
  - 타 DB 에서는 unique 해야만 만들 수 있는 경우도 있음.

- create nonclustered index ncidx_id on table (id)
- index 생성시 : Clustered Index → Non Clustered index
- index 삭제시 : Non Clustered Index → Clustered Index

12. Clustered index 와 NonClustered index simulation

CI & NCI

300

1	200
8	201
21	202

Non Clustered index root

200

1	DEPT1a
2	DEPT3b
3	DEPT5a
4	DEPT4
5	DEPT4c
7	DEPT6

201

8	DEPT4a
9	DEPT2b
10	DEPT3
14	DEPT4b
18	DEPT2c
19	DEPT5

202

21	DEPT2
22	DEPT3a
23	DEPT1

Non Clustered index leaf

400

DEPT1	100
DEPT2C	101
DEPT4	102
DEPT5	103

Clustered index root

100

id	name	part
23	김경남	DEPT1
1	송수환	DEPT1a
21	박재연	DEPT2
9	김순호	DEPT2b

101

id	name	part
18	강수진	DEPT2c
10	양재영	DEPT3
22	김지균	DEPT3a
2	최규황	DEPT3b

102

id	name	part
4	이상우	DEPT4
8	지태창	DEPT4a
14	조영구	DEPT4b
5	현종철	DEPT4c

103

id	name	part
19	양건호	DEPT5
3	김선희	DEPT5a
7	박동규	DEPT6

12. Clustered index 와 NonClustered index simulation

데이터 삽입시 NC Index의 변화

300

1	200
8	201
14	203
21	202

Non Clustered index root

200

1	DEPT1a	
2	DEPT3b	
3	DEPT5a	
4	DEPT4	
5	DEPT4c	
7	DEPT6	

201

8	DEPT4a	
9	DEPT2b	
10	DEPT3	

203

14	DEPT4b	
16	DEPT3c	
18	DEPT2c	
19	DEPT5	

202

21	DEPT2	
22	DEPT3a	
23	DEPT1	

Non Clustered index leaf

400

DEPT1	100
DEPT2c	101
DEPT3a	104
DEPT4	102
DEPT5	103

Clustered index root

id	name	part
23	김경남	DEPT1
1	송수환	DEPT1a
21	박재연	DEPT2
9	김순호	DEPT2b

id	name	part
18	강수진	DEPT2c
10	양재영	DEPT3

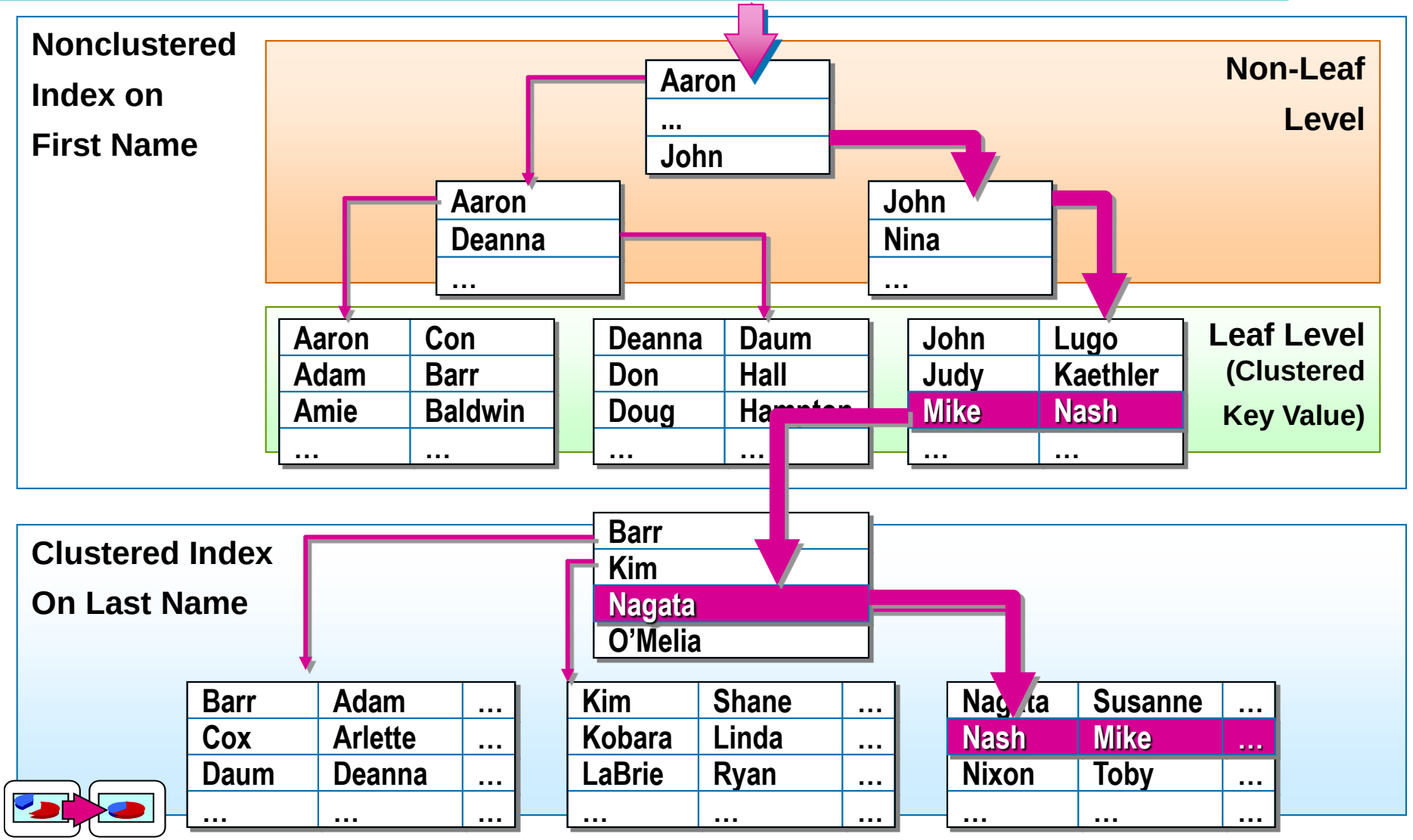
id	name	part
22	김지균	DEPT3a
2	최규황	DEPT3b
16	오유선	DEPT3c

id	name	part
4	이상우	DEPT4
8	지태창	DEPT4a
14	조영구	DEPT4b
5	현종철	DEPT4c

id	name	part
19	양건호	DEPT5
3	김선희	DEPT5a
7	박동규	DEPT6

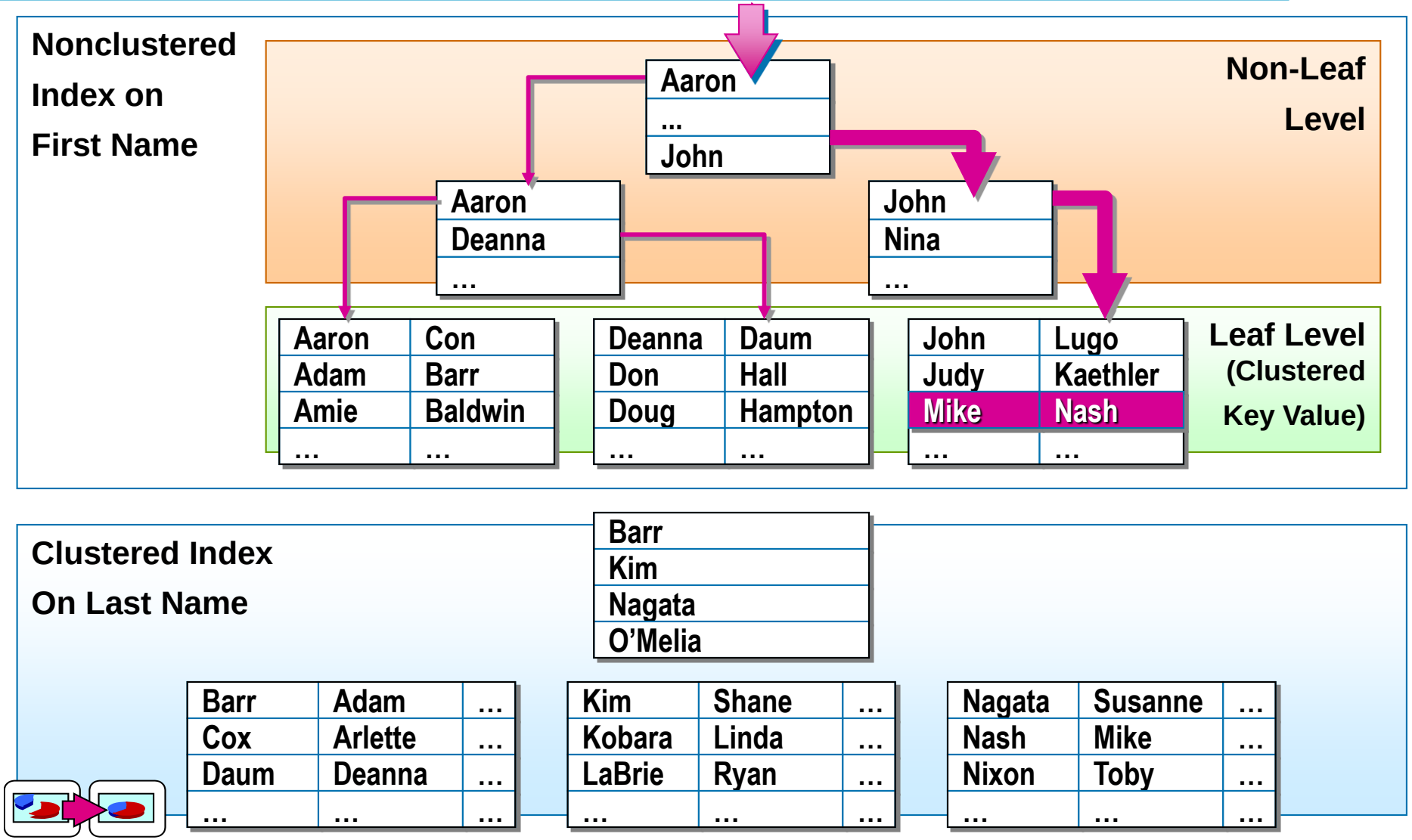
12. Clustered index 와 NonClustered index simulation

```
SELECT lastname, firstname, phone FROM member WHERE firstname = 'Mike'
```



12. Clustered index 와 NonClustered index simulation

```
SELECT lastname, firstname FROM member WHERE firstname = 'Mike'
```



12. Clustered index 와 NonClustered index simulation

	Clustered index	NC index
색인 만든 후 크기 증가	테이블의 1~5%	10~20%
하나 값 주고 찾을 때	상당히 빠르다	클러스터보다 아주 약간 느리다
범위 주고 찾을 때	여전히 빠르다	table scan 보다 느린 경우도 있음
데이터의 수정이 발생할 때	부하가 많이 걸린다.	클러스터보다 부하는 적다.
		Covered index 쓰면 상당히 빨라짐

선택도란, 대상 집합에서 해당 조건을 만족하는 row 가 차지하는 비율

High selectivity

member_no	last_name	first_name
1	Randall	Joshua
2	Flood	Kathie
.		
.		
.		
10000	Anderson	Bill

```
SELECT *
FROM member
WHERE member_no > 8999
```

$$\frac{\text{Number of rows meeting criteria}}{\text{Total number of rows in table}} = \frac{1000}{10000} = 10\%$$

Low selectivity

member_no	last_name	first_name
1	Randall	Joshua
2	Flood	Kathie
.		
.		
.		
10000	Anderson	Bill

```
SELECT *
FROM member
WHERE member_no < 9001
```

$$\frac{\text{Number of rows meeting criteria}}{\text{Total number of rows in table}} = \frac{9000}{10000} = 90\%$$

Cadinality

카디널러티란 판정대상이 가진 결과 건수 혹은 다음 단계로 들어가는 중간 결과수를 의미.

→ selectivity * 전체 row 수

◆ 선택도가 있음에도 불구하고 왜 cardinality 가 필요한가?
- 선택도는 단지 비율에 지나지 않음
- 백만건의 1% 와 천건의 1% 는 다르기 때문.
- 조인의 순서나 방향을 결정하기 위해서 cardinality 가 필요.

결합 인덱스는 여러 개의 컬럼으로 구성된 인덱스이다.

```
CREATE INDEX IDX_1 ON tableA (col1, col2)
```

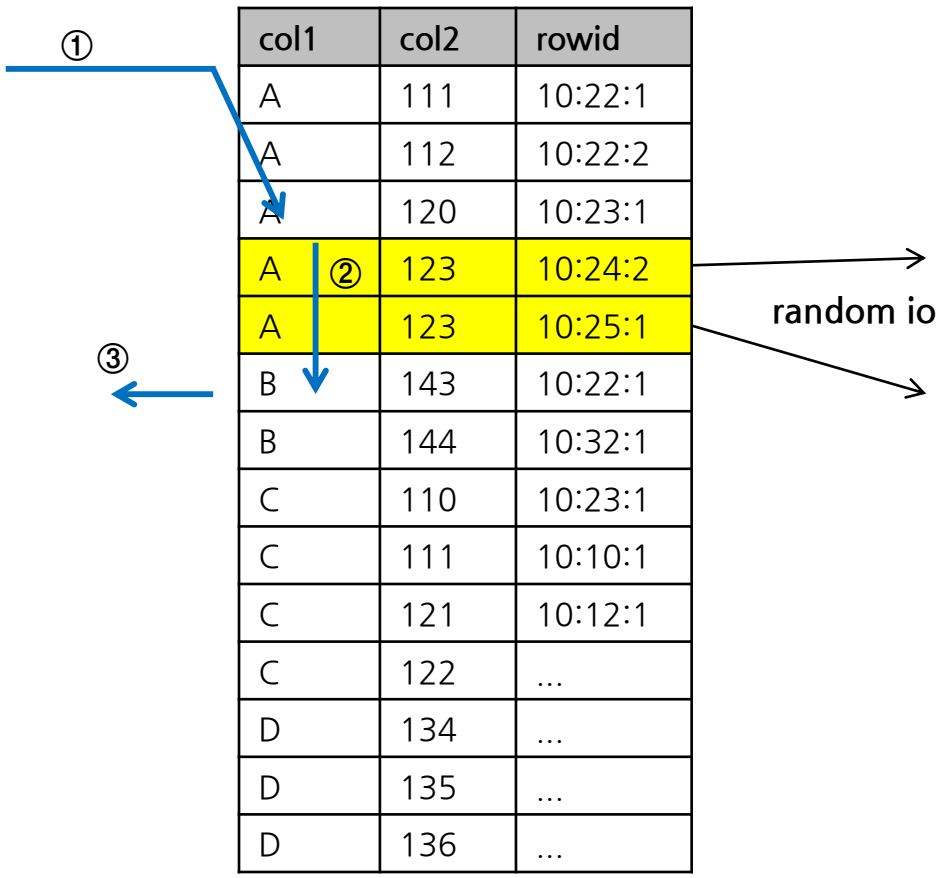
Leaf block (100)			
Block num	Row num	Col1	Col2
90	3	A	110
94	2	A	111
95	1	A	112
96	2	A	113
882	1	A	114
87	5	B	100
88	54	B	101
89	41	B	105
90	2	C	100
90	3	C	102
90	4	C	103
90	5	C	108

논리적인 row id (동일한 인덱스 값 내에서는 row id 로 정렬되어 진다.)
(DB 마다 구성은 틀림)

✓ 사용되는 SQL 문들의 분석을 통하여 여러 개의 컬럼으로 구성된 인덱스를 설계하여야 하는 경우가 존재한다.

✓ 인덱스를 구성하는 컬럼에서 col2 는 col1 을 기준으로 다시 내부에서 정렬되어 진다.

```
select col3, col4, col1 from tablaA where col1='A' and col2 = '123'
```



col1	col2	rowid
A	111	10:22:1
A	112	10:22:2
A	120	10:23:1
A	123	10:24:2
A	123	10:25:1
B	143	10:22:1
B	144	10:32:1
C	110	10:23:1
C	111	10:10:1
C	121	10:12:1
C	122	…
D	134	…
D	135	…
D	136	…

Index

Table

① index 의 root block 으로부터 branch block 을 지나 leaf block 찾아 조건에 부합하는 첫번째 index row 를 바로 찾는다.

② rowid 를 이용하여 테이블의 로우를 access 한다.

③ 다음 index row 를 스캔하고 조건에 만족하면 다시 table row 에 access하고 아니면 index scan 을 종료한다.

selectivity 가 좋은 컬럼이 앞에 오는 것이 최우선 조건인가?

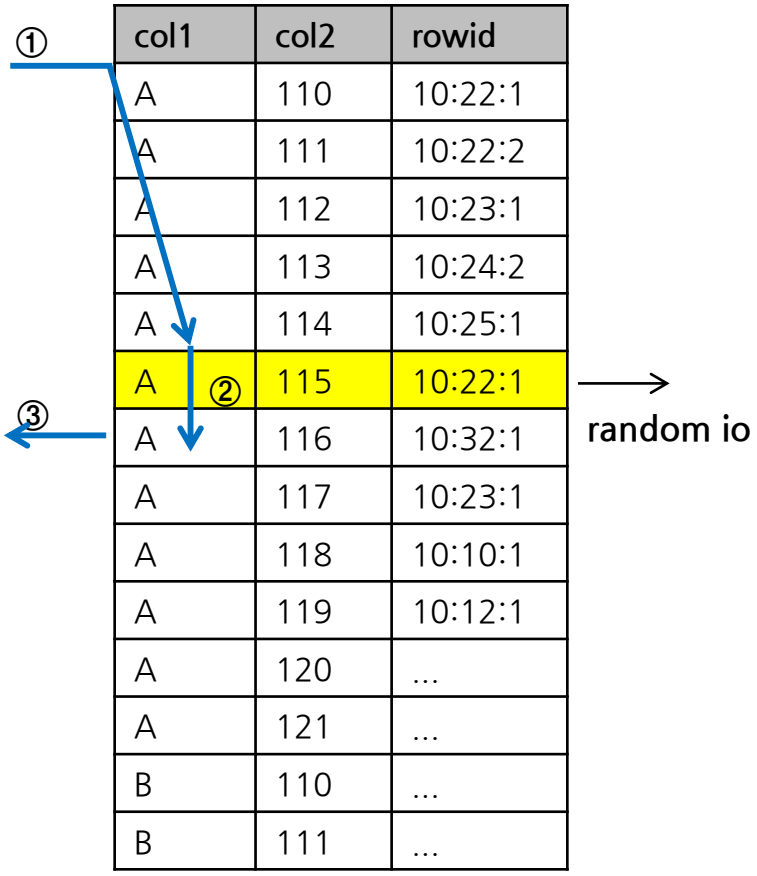
```
select col3, col4, col1 from tablaA where col1='A' and col2  = 123
```

col1	col2	rowid
A	110	10:22:1
A	111	10:22:2
A	112	10:23:1
A	113	10:24:2
A	114	10:25:1
A	115	10:22:1
A	116	10:32:1
A	117	10:23:1
A	118	10:10:1
A	119	10:12:1
A	120	...
A	121	...
B	110	...
B	111	...

col2	col1	rowid
113	A	10:23:1
113	B	10:22:1
113	C	10:10:1
113	D	10:22:2
114	A	10:23:1
114	B	...
114	C	...
114	D	...
115	A	...
115	B	...
115	C	...
115	D	...
116	A	...
116	B	...

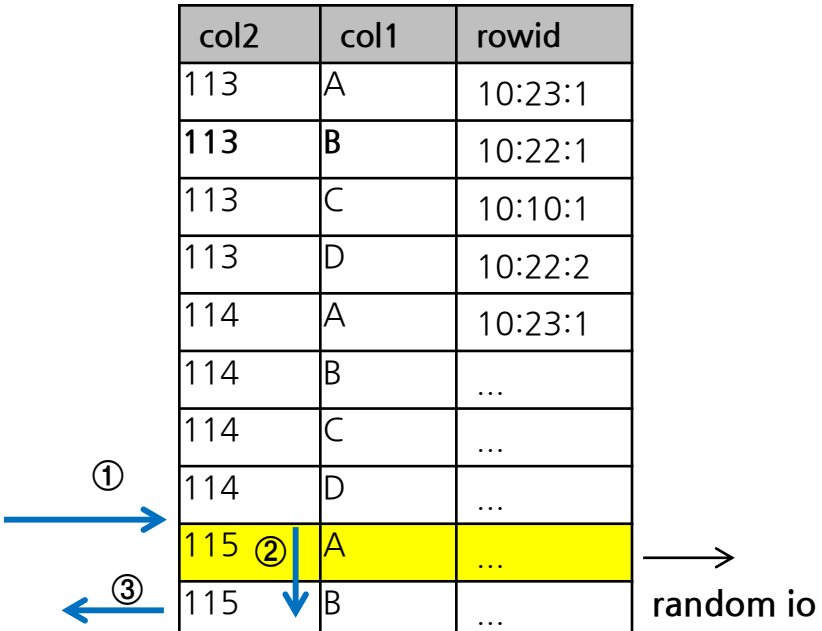
모두 equal 로 비교될 때는 selectivity 는 중요하지 않다.

```
select col3, col4, col1 from tablaA where col1='A' and col2 = 115
```



col1	col2	rowid
A	110	10:22:1
A	111	10:22:2
A	112	10:23:1
A	113	10:24:2
A	114	10:25:1
A	115	10:22:1
A	116	10:32:1
A	117	10:23:1
A	118	10:10:1
A	119	10:12:1
A	120	…
A	121	…
B	110	…
B	111	…

index row scan : 2 , random io : 1



col2	col1	rowid
113	A	10:23:1
113	B	10:22:1
113	C	10:10:1
113	D	10:22:2
114	A	10:23:1
114	B	…
114	C	…
114	D	…
115	A	…
115	B	…
115	C	…
115	D	…
116	A	…
116	B	…

index row scan : 2 , random io : 1

그럼 equal 비교가 아니고 범위 비교와 섞여 있다면 ?

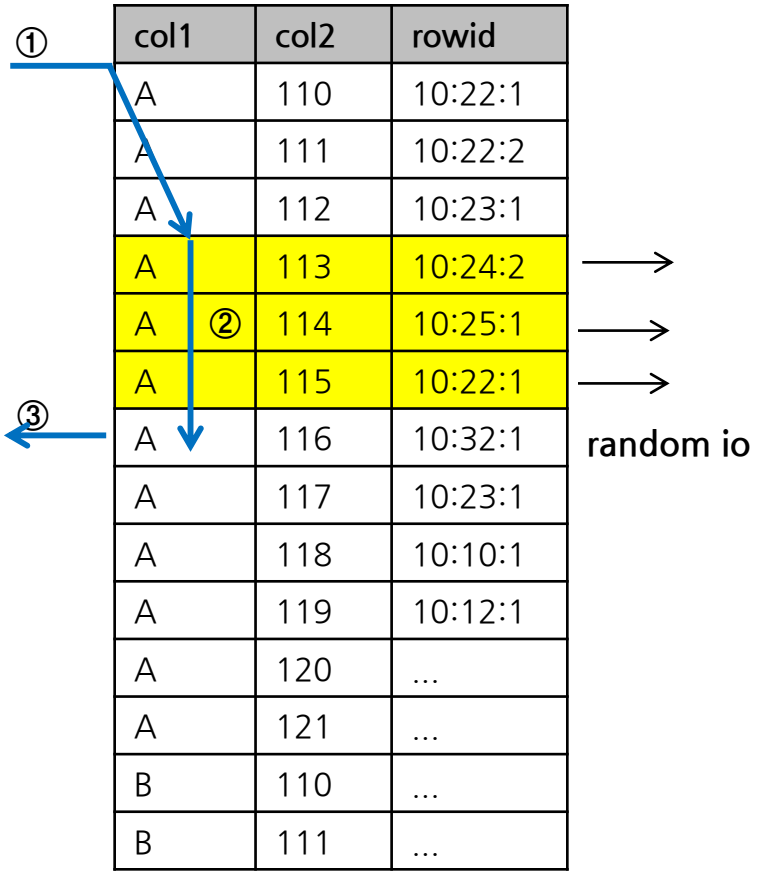
```
select col3, col4, col1 from tablaA where col1='A' and col2 betweeen 113 and 115
```

col1	col2	rowid
A	110	10:22:1
A	111	10:22:2
A	112	10:23:1
A	113	10:24:2
A	114	10:25:1
A	115	10:22:1
A	116	10:32:1
A	117	10:23:1
A	118	10:10:1
A	119	10:12:1
A	120	...
A	121	...
B	110	...
B	111	...

col2	col1	rowid
113	A	10:23:1
113	B	10:22:1
113	C	10:10:1
113	D	10:22:2
114	A	10:23:1
114	B	...
114	C	...
114	D	...
115	A	...
115	B	...
115	C	...
115	D	...
116	A	...
116	B	...

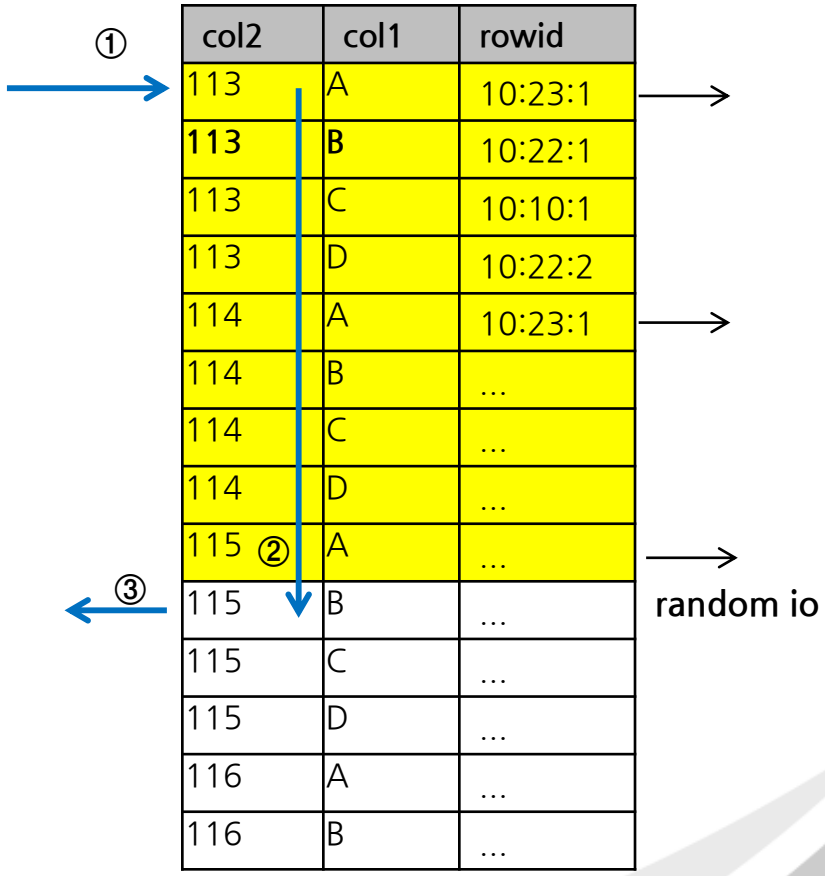
index 의 선두 컬럼이 between 조건으로 검색되면 처리 범위가 증가한다.

```
select col3, col4, col1 from tablaA where col1='A' and col2 betweeen 113 and 115
```



col1	col2	rowid
A	110	10:22:1
A	111	10:22:2
A	112	10:23:1
A	113	10:24:2
A	114	10:25:1
A	115	10:22:1
A	116	10:32:1
A	117	10:23:1
A	118	10:10:1
A	119	10:12:1
A	120	…
A	121	…
B	110	…
B	111	…

index row scan : 4 , random io : 3

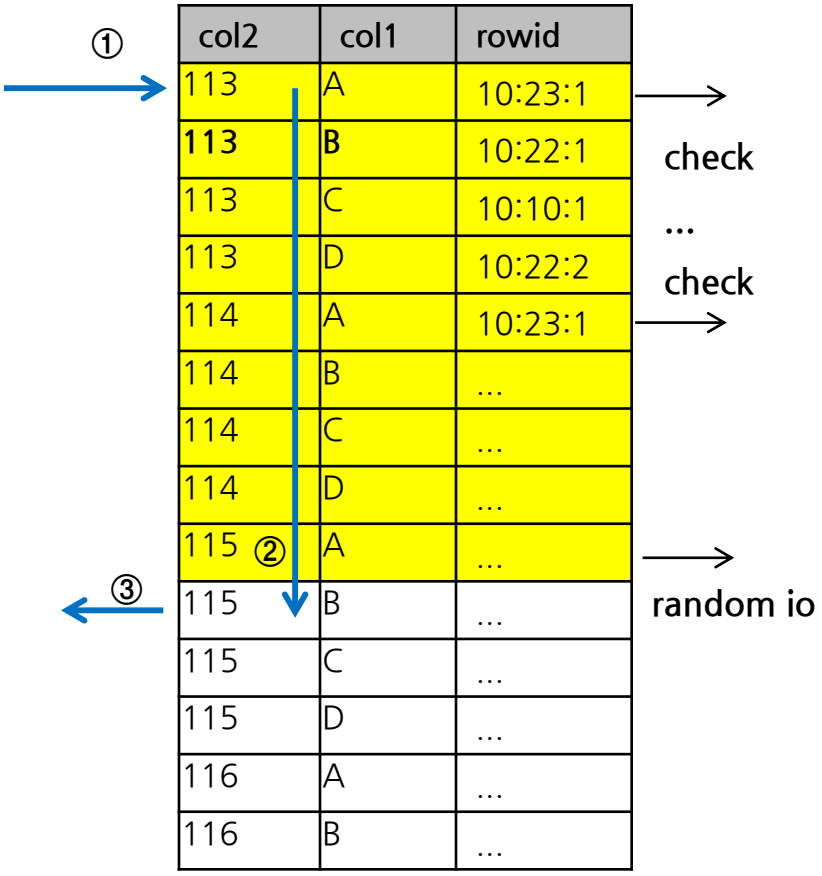


col2	col1	rowid
113	A	10:23:1
113	B	10:22:1
113	C	10:10:1
113	D	10:22:2
114	A	10:23:1
114	B	…
114	C	…
114	D	…
115	A	…
115	B	…
115	C	…
115	D	…
116	A	…
116	B	…

index row scan : 10 , random io : 3

selectivity 보다는 검색 조건이 결합 인덱스의 컬럼 순서를 결정하는데 결정적 역할을 한다.

```
select col3, col4, col1 from tablaA where col1='A' and col2 betweeen 113 and 115
```

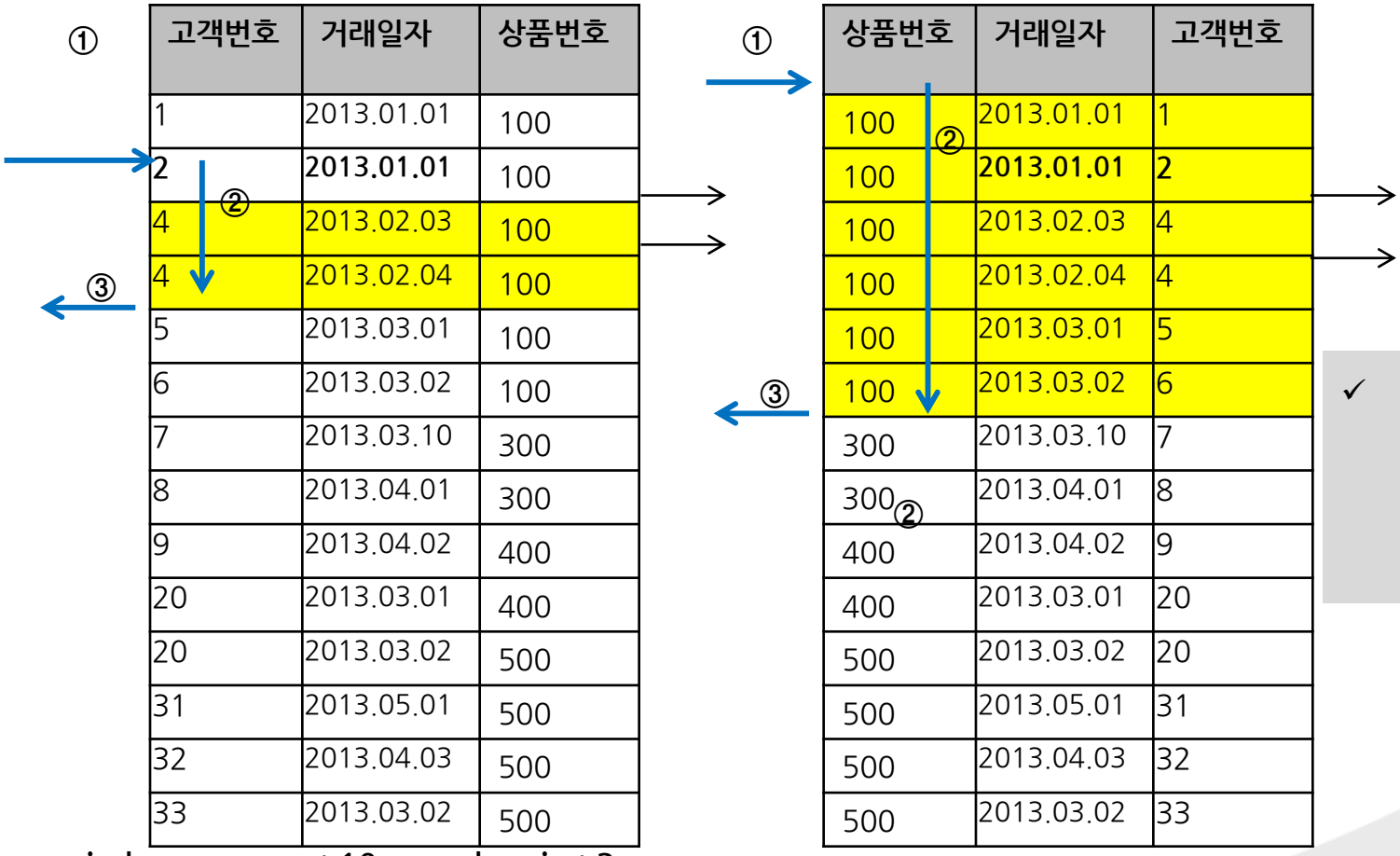


index row scan : 10 , random io : 3

- ✓ col2 가 범위 조건으로 검색되는 순간 col1 은 해당 범위 내에서 정렬이 보장되지 않느다.
- ✓ 어떤 위치에 있는 컬럼이더라도 자기보다 앞에 있는 컬럼이 '=' 로 사용되지 않으면 자신의 역할은 체크기능 으로 전락한다.
- ✓ 인덱스 컬럼 순서의 결정에는 selectivity 보다 '=' 이 우선적으로 감안되어야 한다.

selectivity 가 좋은 컬럼을 앞에 놓는 것이 좋은 경우는 ?

select * from table where 고객번호 = 4 and 상품번호 = 100 and 거래일자 between '2013-01-01' and '2013-05-31'



고객번호	거래일자	상품번호
1	2013.01.01	100
2	2013.01.01	100
4	2013.02.03	100
4	2013.02.04	100
5	2013.03.01	100
6	2013.03.02	100
7	2013.03.10	300
8	2013.04.01	300
9	2013.04.02	400
20	2013.03.01	400
20	2013.03.02	500
31	2013.05.01	500
32	2013.04.03	500
33	2013.03.02	500

상품번호	거래일자	고객번호
100	2013.01.01	1
100	2013.01.01	2
100	2013.02.03	4
100	2013.02.04	4
100	2013.03.01	5
100	2013.03.02	6
300	2013.03.10	7
300	2013.04.01	8
400	2013.04.02	9
400	2013.03.01	20
500	2013.03.02	20
500	2013.05.01	31
500	2013.04.03	32
500	2013.03.02	33

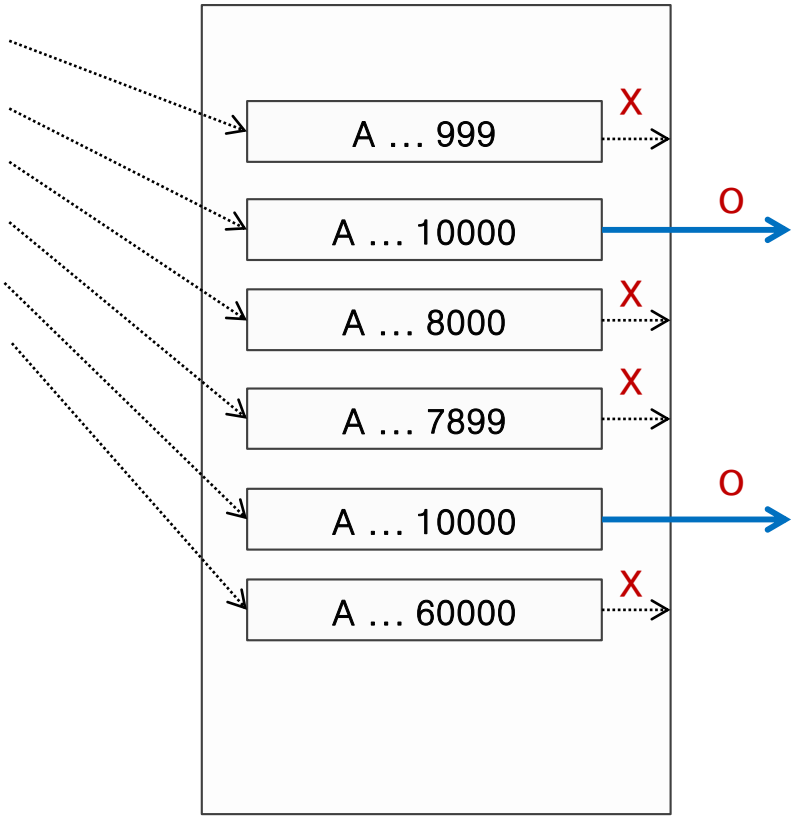
index row scan : 10 , random io : 3

✓ 범위 검색 조건을 사이에 둔 컬럼끼리는 선택도가 낮은 컬럼을 앞쪽에 두는 것이 유리하다.

결합 인덱스에 check 조건을 만족하기 위해서 컬럼을 추가할 수 있다.

col1	col2	rowid
A	110	10:22:1
A	111	10:22:2
A	112	10:23:1
A	113	10:24:2
A	114	10:25:1
A	115	10:22:1
B	116	10:32:1
B	117	10:23:1
B	118	10:10:1
B	119	10:12:1
B	120	…
B	121	…
C	110	…
C	111	…

인덱스



테이블

✓ select *
from tab
where col1 = 'A'
and col3 = 10000

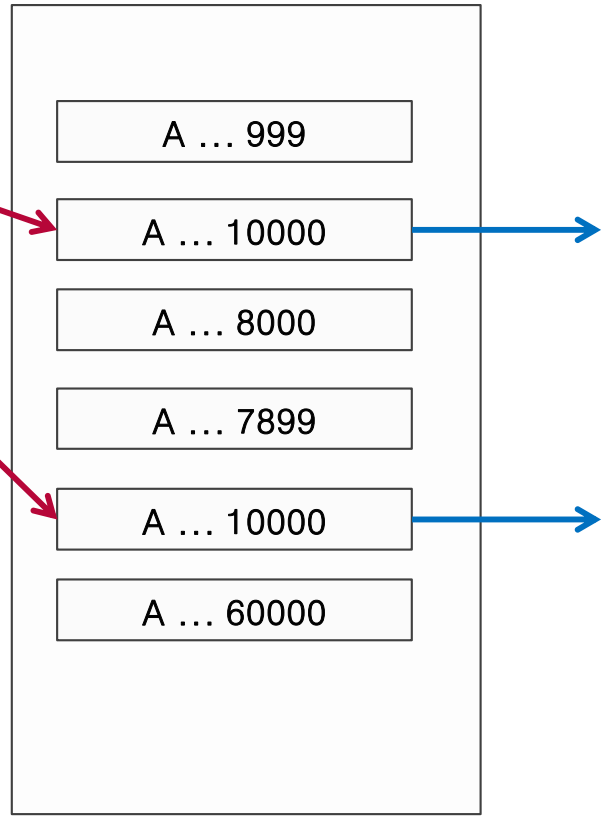
✓ index의 선행 컬럼이 '=' 조건으로 실행되고 있으나 전체 6번의 table access 가 발생하고 이중에 2건만 col3 조건을 만족하여 결과값으로 보내어 진다.

인덱스 컬럼의 추가

col3 를 추가로 결합 인덱스에 추가함으로써 table 로의 random io 를 줄일 수 있다.

col1	col2	col3	rowid
A	110	999	10:22:1
A	111	8000	10:22:2
A	112	10000	10:23:1
A	113	7899	10:24:2
A	114	10000	10:25:1
A	115	60000	10:22:1
B	116		10:32:1
B	117		10:23:1
B	118		10:10:1
B	119		10:12:1
B	120		…
B	121		…
C	110		…
C	111		…

인덱스



테이블

✓ select *
from tab
where col1 = 'A'
and col3 = 10000

✓ index 처리 범위는 동일하지만 'col3' 가 인덱스 내에 있기 때문에 index 를 scan 하면서 성공한 것만 테이블로 random io 를 수행한다.

인덱스를 선정하는 데는 다양한 사항을 고려해야 한다.

① 항상 사용하는가?

② 항상 '=' 로 사용하는가?

③ 어느 것이 더 좋은 selectivity 를 가지는가?

④ 자주 정렬되는 순서는 무엇인가?

⑤ 부가적으로 추가시킬 컬럼은 어떤 것으로 할것인가?

인덱스 선정에는 정답이 없다.

✓ 전제 조건

① 총 고객수는 100만, 상품은 10만개 미만

② 거래 일자 검색 범위는 유동적이다.

✓ 검색 조건

① select * from table where 고객번호 = 1
and 거래일자 between '20100101' and '20100331'

② select * from table where 상품번호 = 'A'
and 거래일자 between '20100101' and '20100331'

③ select * from table where 상품번호 = 'A'
and 고객번호 = 1
and 거래일자 between '20100101' and '20100331'

④ select * from table where 거래일자 between '20100101' and '20100331'

13. 결합 인덱스

예 제

	IDX1	IDX2	IDX3	IDX4

13. 결합 인덱스

예 제

	IDX1	IDX2	IDX3	IDX4

14. Do not!!!

인덱스 컬럼에 가공을 가하면 인덱스를 사용할 수 없다.

수정전	수정 방안
WHERE to_char(일시,'yyyymmdd') = :dt	Where 일시 >= to_date(:dt,'yyyymmdd') and 일시 < to_date(:dt,'yyyymmdd') + 1
WHERE price*12=100	WHERE price = 100/12
WHERE substring(au_lname,1,1)='S'	WHERE au_lname like 'S%'
WHERE 회원번호 || 지점번호 = :str	WHERE 회원번호 = substr(:str, 1,2) and 지점번호 = substr(:str,3,4)
WHERE au_lname like '%Sm'	WHERE au_lname like 'Sm%'
* WHERE 연월 = substr(일자, 1,6) -1	???

14. Do not!!!

인덱스 컬럼에 가공을 가하면 인덱스를 사용할 수 없다.

수정전	수정 방안
WHERE to_char(일시,'yyyymmdd') = :dt	Where 일시 >= to_date(:dt,'yyyymmdd') and 일시 < to_date(:dt,'yyyymmdd') + 1
WHERE price*12=100	WHERE price = 100/12
WHERE substring(au_lname,1,1)='S'	WHERE au_lname like 'S%'
WHERE 회원번호 || 지점번호 = :str	WHERE 회원번호 = substr(:str, 1,2) and 지점번호 = substr(:str,3,4)
WHERE au_lname like '%Sm'	WHERE au_lname like 'Sm%'
* WHERE 연월 = substr(일자, 1,6) -1	WHERE 연월 = to_char(add_months(to_date (일자,'yyyymmdd'),-1),'yyyymm')

묵시적 형변환의 예

: varchar 컬럼에 숫자를 더하거나 빼면 내부적으로 숫자형으로 형변환됨.

위의 예에서 substr(일자,1,6) -1 을 하는 순간 이 값은 숫자로 형변환.

'연월' 컬럼이 varchar 라고 가정 했을 때 쿼리 수행 단계에서 '연월' 컬럼도 숫자 타입으로 형변환됨. 결국 '연월' 컬럼이 가공되어 버려 인덱스를 사용 못함.