

SQLD 특강

29 ~ 31 SQL 최적화 기본 원리

강사 강태우

해당 창작물의 허가되지 않은 재배포, 무단 복사 및 무단 게시를 엄격히 금지합니다.
반드시 저작자에게 문의한 후 허가를 받아 사용해주세요.

SQLD 특강

30. 인덱스 기본

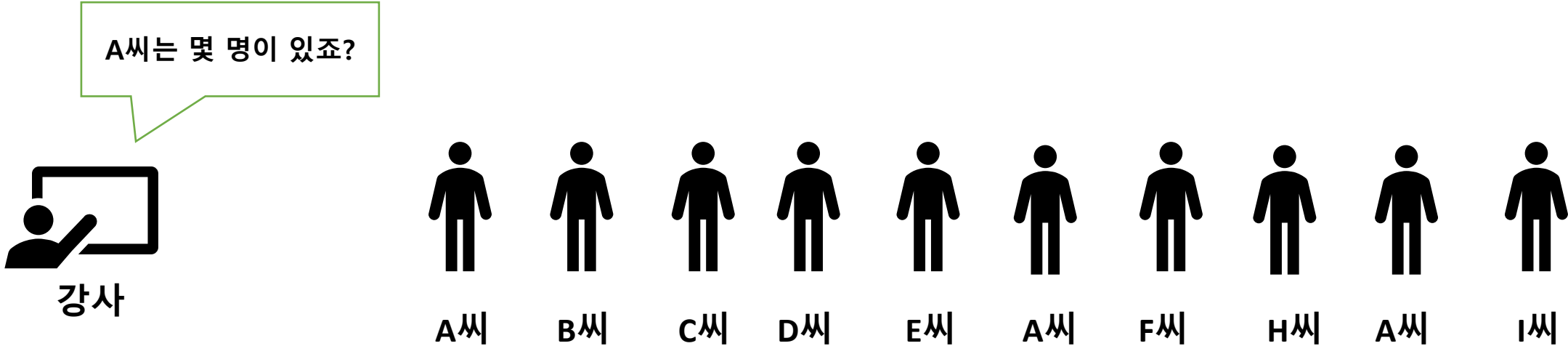
29. 옵티마이저와 실행계획

31. 조인 수행 원리

30. 인덱스(INDEX) 기본

인덱스(INDEX) 란?

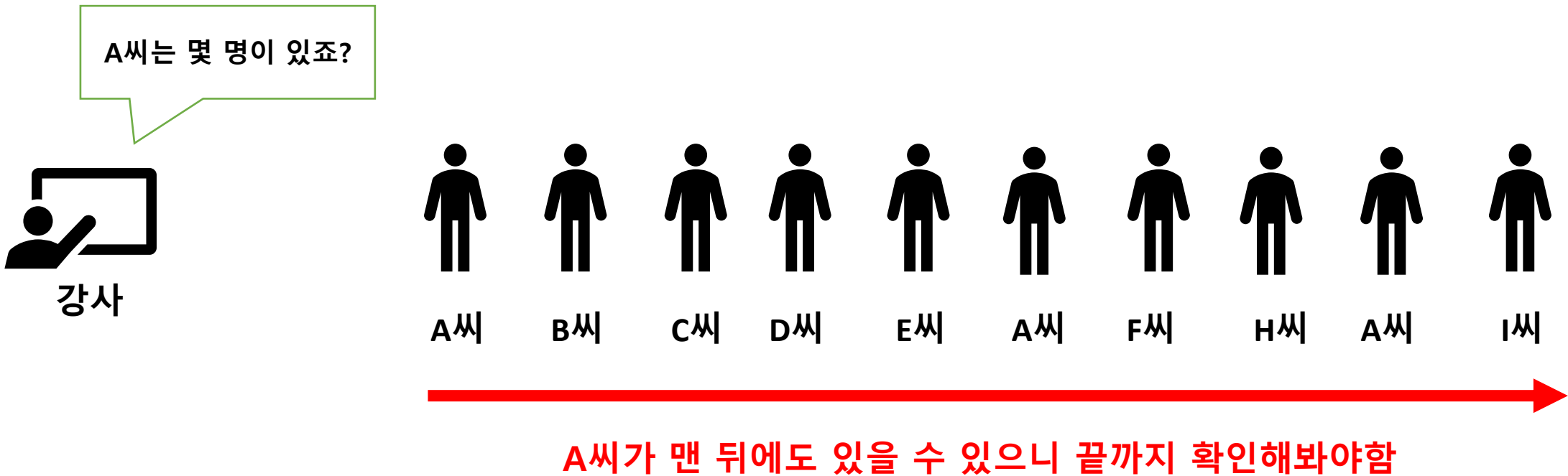
책에 있는 색인 혹은 목차처럼 특정 데이터를 쉽게 찾을 수 있게 **특정 기준으로 정리해놓은 객체**



현재 무작위로 자리에 착석한 상황

인덱스(INDEX) 란?

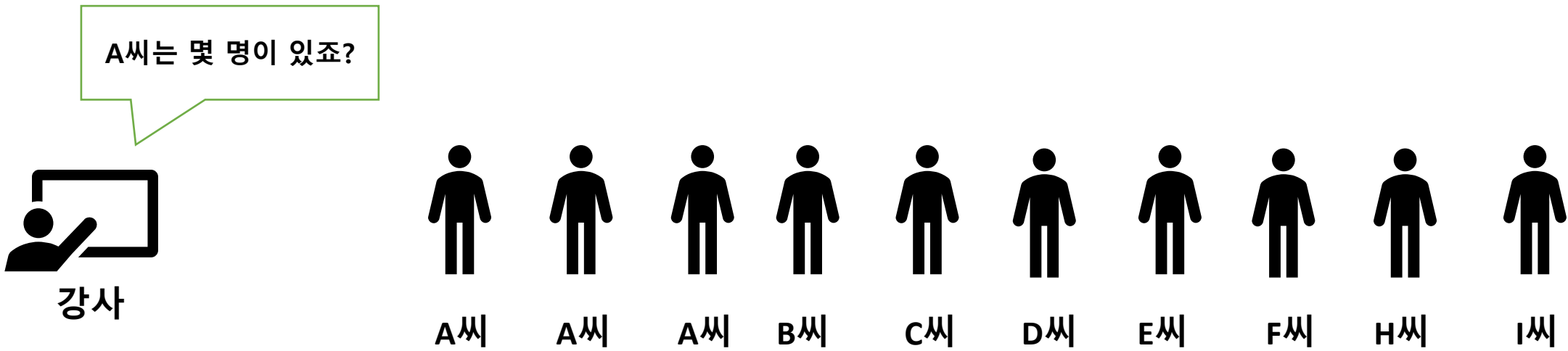
책에 있는 색인 혹은 목차처럼 특정 데이터를 쉽게 찾을 수 있게 **특정 기준으로 정리해놓은 객체**



A씨가 맨 뒤에도 있을 수 있으니 끝까지 확인해봐야함

인덱스(INDEX) 란?

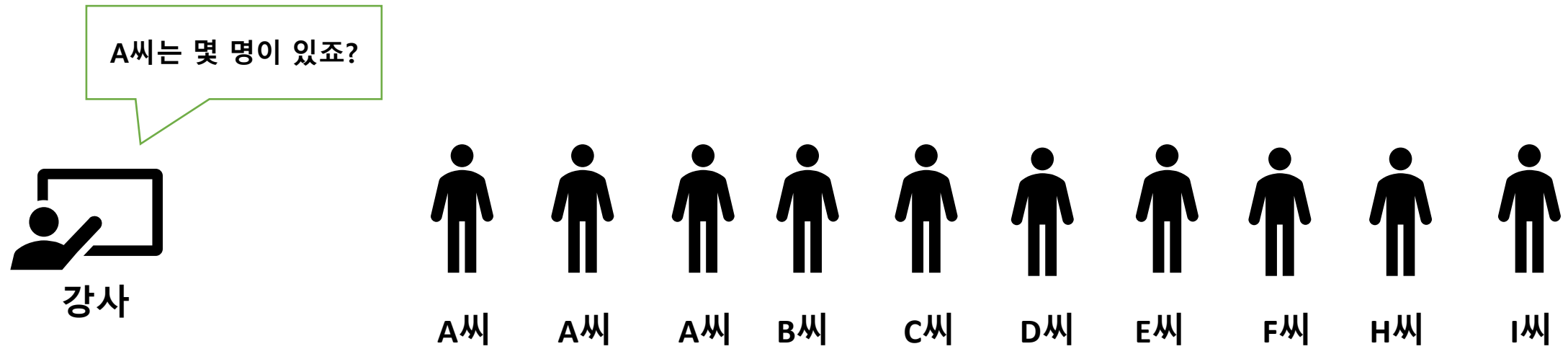
책에 있는 색인 혹은 목차처럼 특정 데이터를 쉽게 찾을 수 있게 **특정 기준으로 정리해놓은 객체**



이름 순으로 오름차순하여 자리에 앉은 상황

인덱스(INDEX) 란?

책에 있는 색인 혹은 목차처럼 특정 데이터를 쉽게 찾을 수 있게 **특정 기준으로 정리해놓은 객체**



이름 순으로 정렬되었기 때문에 A씨를 바로 모두 찾을 수 있음
B씨를 찾아본 이유는 ONE-PLUS SCAN 개념 때문

인덱스(INDEX) 가 없다면 ?

```
SELECT 직원ID , 이름 , 나이 , 입사일시 FROM 직원 WHERE 이름 = '이현정';
```

직원ID	이름	나이	입사일시
A0014	공손한	50	2023/01/24 12:41:26
A0001	김철수	25	2022/03/21 00:00:00
A0002	강홍수	28	2021/09/12 00:00:00
A0003	이현정	(null)	2022/11/06 00:00:00
A0004	김선미	(null)	2020/03/11 00:00:00
A0005	문현철	34	(null)
A0006	송대주	44	2015/07/16 00:00:00
A0007	메이슨	40	2016/08/19 00:00:00
A0008	송진아	47	2015/07/16 00:00:00
A0009	이서연	50	2013/11/23 00:00:00
A0010	김홍민	52	2013/11/23 00:00:00

여기서 이름이 “이현정”인 사람을
전부 찾아볼까요?
(단, 현재 인덱스가 없습니다)



인덱스(INDEX) 가 없다면?

```
SELECT 직원ID , 이름 , 나이 , 입사일시 FROM 직원 WHERE 이름 = '이현정';
```

직원ID	이름	나이	입사일시
A0014	공손한	50	2023/01/24 12:41:26
A0001	김철수	25	2022/03/21 00:00:00
A0002	강홍수	28	2021/09/12 00:00:00
A0003	이현정	(null)	2022/11/06 00:00:00
A0004	김선미	(null)	2020/03/11 00:00:00
A0005	문현철	34 (null)	
A0006	송대주	44	2015/07/16 00:00:00
A0007	메이슨	40	2016/08/19 00:00:00
A0008	송진아	47	2015/07/16 00:00:00
A0009	이서연	50	2013/11/23 00:00:00
A0010	김홍민	52	2013/11/23 00:00:00

실행 계획 설명 x

SQL | 0.043초

계획설명 보는법 : 쿼리에 커서 대고 F10 클릭 (SQL DEVELOPER 기준)

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	직원	FULL
Filter Predicates		
이름='이현정'		

실행 계획을 보면 **TABLE FULL SCAN** 이 떠있습니다.

이는 왼쪽 테이블 내용을 전부 뒤져서 “이현정” 을 찾은 걸 의미합니다.

인덱스(INDEX) 를 만들어봅시다.

```
CREATE INDEX 직원_이름_INDEX ON 직원 (이름 ASC )
```

인덱스를 생성합니다.

생성할 인덱스이름

직원 테이블에서 이름 컬럼 오름차 혹은 내림차 순으로 생성
(기본값은 오름차순)

직원ID	이름	나이	입사일시
A0014	공손한	50	2023/01/24 12:41:26
A0001	김철수	25	2022/03/21 00:00:00
A0002	강홍수	28	2021/09/12 00:00:00
A0003	이현정	(null)	2022/11/06 00:00:00
A0004	김선미	(null)	2020/03/11 00:00:00
A0005	문현철	34 (null)	
A0006	송대주	44	2015/07/16 00:00:00
A0007	메이슨	40	2016/08/19 00:00:00
A0008	송진아	47	2015/07/16 00:00:00
A0009	이서연	50	2013/11/23 00:00:00
A0010	김홍민	52	2013/11/23 00:00:00

< 직원 테이블 >

이름
강홍수
공손한
김선미
김철수
김홍민
메이슨
문현철
송대주
송진아
이서연
이현정

< 직원_이름_INDEX >

이름 컬럼을 기준으로
오름차순 정렬되어
있는 인덱스를 생성!



인덱스(INDEX) 를 이용한 경우

```
SELECT 직원ID , 이름 , 나이 , 입사일시 FROM 직원 WHERE 이름 = '이현정';
```

WHERE 조건에 인덱스가 있는 컬럼이 쓰이면 인덱스 사용 가능

직원ID	이름	나이	입사일시
A0014	공손한	50	2023/01/24 12:41:26
A0001	김철수	25	2022/03/21 00:00:00
A0002	강홍수	28	2021/09/12 00:00:00
A0003	이현정	(null)	2022/11/06 00:00:00
A0004	김선미	(null)	2020/03/11 00:00:00
A0005	문현철	34 (null)	
A0006	송대주	44	2015/07/16 00:00:00
A0007	메이슨	40	2016/08/19 00:00:00
A0008	송진아	47	2015/07/16 00:00:00
A0009	이서연	50	2013/11/23 00:00:00
A0010	김홍민	52	2013/11/23 00:00:00

스크립트 출력 x | 실행 결과 x | 계획 설명 x

SQL | 0.018초

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	직원	BY INDEX ROWID BATCHED
INDEX	직원_이름_INDEX	RANGE SCAN
Access Predicates		
	이름='이현정'	

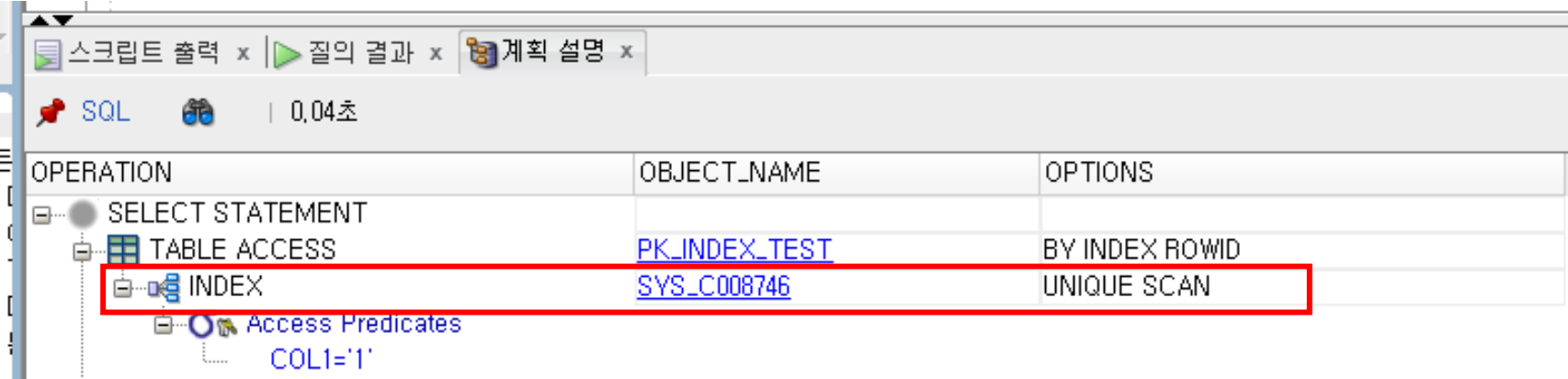
실행 계획을 보면 INDEX RANGE SCAN 이 떠있습니다.

이는 인덱스를 이용해서 특정 범위의 데이터만 조회한 걸 의미합니다.

PK 생성시 기본 인덱스가 **자동으로 생성**됩니다.

```
CREATE TABLE PK_IND_TEST (  
  COL1 VARCHAR2(10) PRIMARY KEY , -- PK설정된 컬럼에 자동으로 인덱스를 오름차순으로 생성  
  COL2 VARCHAR2(10) );  
  
INSERT INTO PK_IND_TEST VALUES (1,1);  
INSERT INTO PK_IND_TEST VALUES (2,2);  
INSERT INTO PK_IND_TEST VALUES (3,2);  
  
SELECT /*+ RULE */ * FROM PK_IND_TEST WHERE COL1 = '1' ;
```

실행계획 읽는법은 뒤에서 나옵니다.
여기서는 PK 로 설정한 컬럼에 대해
인덱스가 생성이 되었고 사용되었구나
정도로만 보시면 됩니다.



OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	PK_INDEX_TEST	BY INDEX ROWID
INDEX	SYS_C008746	UNIQUE SCAN
Access Predicates		COL1='1'



인덱스(INDEX) 원리 체크

```
SELECT 직원ID , 이름 , 나이 , 입사일시 FROM 직원 WHERE 이름 = '이현정';
```

인덱스는 이미 정렬이 되어있으므로
바로 “이” 부분을 찾아도 됩니다.



인덱스(INDEX) 의 목적은 조회성능 향상

```
SELECT 직원ID , 이름 , 나이 , 입사일시 FROM 직원 WHERE 이름 = '이현정';
```

직원ID	이름	나이	입사일시
A0014	공손한	50	2023/01/24 12:41:26
A0001	김철수	25	2022/03/21 00:00:00
A0002	강홍수	28	2021/09/12 00:00:00
A0003	이현정	(null)	2022/11/06 00:00:00
A0004	김선미	(null)	2020/03/11 00:00:00
A0005	문현철	34 (null)	
A0006	송대주	44	2015/07/16 00:00:00
A0007	메이슨	40	2016/08/19 00:00:00
A0008	송진아	47	2015/07/16 00:00:00
A0009	이서연	50	2013/11/23 00:00:00
A0010	김홍민	52	2013/11/23 00:00:00

인덱스 없을 때

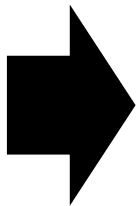
이름	직원ID	이름	나이	입사일시
강홍수	A0014	공손한	50	2023/01/24 12:41:26
공손한	A0001	김철수	25	2022/03/21 00:00:00
김선미	A0002	강홍수	28	2021/09/12 00:00:00
김철수	A0003	이현정	(null)	2022/11/06 00:00:00
김홍민	A0004	김선미	(null)	2020/03/11 00:00:00
메이슨	A0005	문현철	34 (null)	
문현철	A0006	송대주	44	2015/07/16 00:00:00
송대주	A0007	메이슨	40	2016/08/19 00:00:00
송진아	A0008	송진아	47	2015/07/16 00:00:00
이서연	A0009	이서연	50	2013/11/23 00:00:00
이현정	A0010	김홍민	52	2013/11/23 00:00:00

인덱스가 있을 때

대신 DML 성능이 감소합니다.(TRADE-OFF)

```
INSERT INTO 직원 (직원ID , 비밀번호 , 이름 , 주민등록번호 , 부서ID )  
VALUES ('A0015' , '123' , '새사람' , '941212-1231123' , 'D001' ) ;
```

	직원ID	이름	나이	입사일시
1	A0014	공손한	50	2023/01/24 12:41:26
2	A0001	김철수	25	2022/03/21 00:00:00
3	A0002	강홍수	28	2021/09/12 00:00:00
4	A0003	이현정	(null)	2022/11/06 00:00:00
5	A0004	김선미	(null)	2020/03/11 00:00:00
6	A0005	문현철	34	(null)
7	A0006	송대주	44	2015/07/16 00:00:00
8	A0007	메이슨	40	2016/08/19 00:00:00
9	A0008	송진아	47	2015/07/16 00:00:00
10	A0009	이서연	50	2013/11/23 00:00:00
11	A0010	김홍민	52	2013/11/23 00:00:00
12	A0015	새사람	(null)	(null)



	이름
1	강홍수
2	공손한
3	김선미
4	김철수
5	김홍민
6	메이슨
7	문현철
8	새사람
9	송대주
10	송진아
11	이서연
12	이현정

값이 입력/수정/삭제 가 되면
인덱스도 똑같이 반영됩니다.
(테이블 내용도 바뀌고 인덱스 내용도 바뀌고..)



INDEX를 이용해 조회하는게 무조건 좋을까? NO!

```
SELECT 직원ID , 이름 , 나이 , 입사일시
FROM 직원
WHERE 나이 BETWEEN 20 AND 50 ;
```

(현재 나이 컬럼에 대한 인덱스가 있다고 가정)

직원ID	이름	나이	입사일시
1 A0014	공손한	50	2023/01/24 12:41:26
2 A0001	김철수	25	2022/03/21 00:00:00
3 A0002	강홍수	28	2021/09/12 00:00:00
4 A0003	이현정	(null)	2022/11/06 00:00:00
5 A0004	김선미	(null)	2020/03/11 00:00:00
6 A0005	문현철	34	(null)
7 A0006	송대주	44	2015/07/16 00:00:00
8 A0007	메이슨	40	2016/08/19 00:00:00
9 A0008	송진아	47	2015/07/16 00:00:00
10 A0009	이서연	50	2013/11/23 00:00:00
11 A0010	김홍민	52	2013/11/23 00:00:00
12 A0015	새사람	(null)	(null)

인덱스가 없을 때

나이	직원ID	이름	나이	입사일시
25	1 A0014	공손한	50	2023/01/24 12:41:26
28	2 A0001	김철수	25	2022/03/21 00:00:00
34	3 A0002	강홍수	28	2021/09/12 00:00:00
40	4 A0003	이현정	(null)	2022/11/06 00:00:00
44	5 A0004	김선미	(null)	2020/03/11 00:00:00
47	6 A0005	문현철	34	(null)
50	7 A0006	송대주	44	2015/07/16 00:00:00
50	8 A0007	메이슨	40	2016/08/19 00:00:00
52	9 A0008	송진아	47	2015/07/16 00:00:00
(null)	10 A0009	이서연	50	2013/11/23 00:00:00
(null)	11 A0010	김홍민	52	2013/11/23 00:00:00
(null)	12 A0015	새사람	(null)	(null)

인덱스가 있을 때

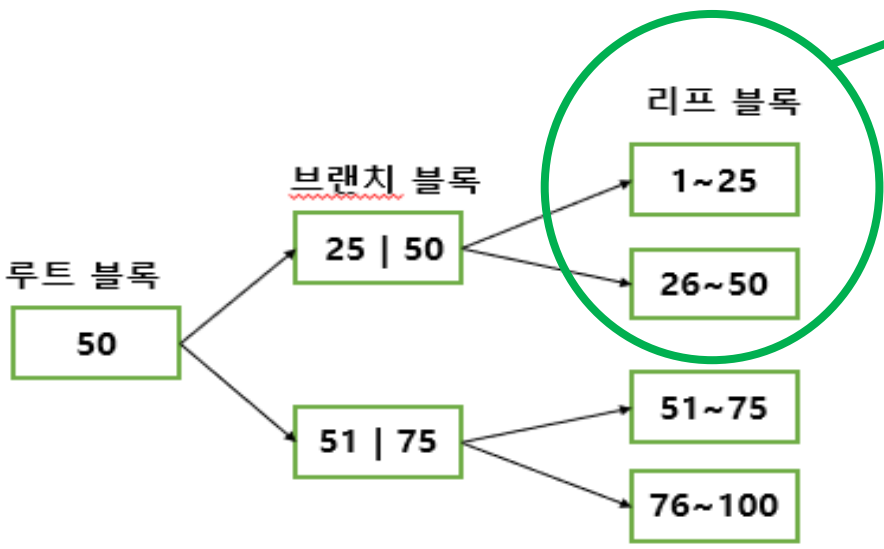
* 이유는 뒤에 옵티마이저와 실행계획에서 다룹니다

B-tree 인덱스 구조 알아보기

가장 일반적인 형태의 인덱스 구조

```
SELECT 직원ID , 이름 , 나이 , 입사일시
FROM 직원
WHERE 나이 BETWEEN 20 AND 28 ;
```

저장 값	데이터를 가리키는 고유주소 (ROWID)
25	AAABBBCCCDDD
28	AAABBBCCCDDE
34	AAABBBCCCDDF
40	AAABBBCCCDDG
...	...



나이	ROWID	직원ID	이름	나이	입사일시
25	2	A0001	김철수	25	2022/03/21 00:00:00
28	3	A0002	강홍수	28	2021/09/12 00:00:00
34	5	A0004	김선미	(null)	2020/03/11 00:00:00
40	7	A0006	송대주	44	2015/07/16 00:00:00
44	8	A0007	메이슨	40	2016/08/19 00:00:00
47	9	A0008	송진아	47	2015/07/16 00:00:00
50	10	A0009	이서연	50	2013/11/23 00:00:00
50	11	A0010	김홍민	52	2013/11/23 00:00:00
52	12	A0015	새사람	(null)	(null)
(null)					
(null)					
(null)					

▶ 30. INDEX(인덱스) 기본

<EMP 테이블>

EMP_ID	REGIST_DATE	DEPTNO
0001	2022.01.01	D001
0002	2022.01.02	D001
0003	2022.01.03	D001
0004	2022.01.04	D001
0005	2022.01.05	D001
0006	2022.01.06	D001
0007	2022.01.07	D001
0008	2022.01.08	D001
0009	2022.01.09	D001
0010	2022.01.10	D001
0011	2022.01.01	D002
0012	2022.01.02	D002
0013	2022.01.03	D002
0014	2022.01.04	D002
0015	2022.01.05	D002
0016	2022.01.06	D002
0017	2022.01.07	D002
0018	2022.01.08	D002
0019	2022.01.09	D002
0020	2022.01.10	D002
0021	2022.01.01	D003
0022	2022.01.02	D003
0023	2022.01.03	D003
0024	2022.01.04	D003
0025	2022.01.05	D003
0026	2022.01.06	D003
0027	2022.01.07	D003
0028	2022.01.08	D003
0029	2022.01.09	D003
0030	2022.01.10	D003

<인덱스 : [REGIST_DATE + DEPTNO] >

REGIST_DATE	DEPTNO
2022.01.01	D001
2022.01.01	D002
2022.01.01	D003
2022.01.02	D001
2022.01.02	D002
2022.01.02	D003
2022.01.03	D001
2022.01.03	D002
2022.01.03	D003
2022.01.04	D001
2022.01.04	D002
2022.01.04	D003
2022.01.05	D001
2022.01.05	D002
2022.01.05	D003
2022.01.06	D001
2022.01.06	D002
2022.01.06	D003
2022.01.07	D001
2022.01.07	D002
2022.01.07	D003
2022.01.08	D001
2022.01.08	D002
2022.01.08	D003
2022.01.09	D001
2022.01.09	D002
2022.01.09	D003
2022.01.10	D001
2022.01.10	D002
2022.01.10	D003

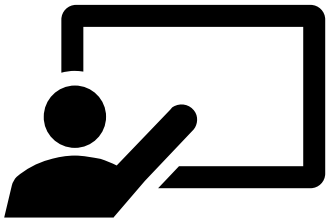
```
SELECT *  
FROM EMP  
WHERE DEPTNO = 'D001'  
AND REGIST_DATE BETWEEN '2022.01.01' AND '2022.01.03' ;
```

- (1) 인덱스 순서를 DEPTNO + REGIST_DATE 로 바꾸면 효율이 좋아지는가 ?
- (2) 이 인덱스로 DEPTNO = 'D001' 조건을 검색하면 효율적인 탐색이 가능한가?
- (3) REGIST_DATE에 대한 조건을 BETWEEN 이 아닌 = 로 바꾸면 인덱스가 효율적인가?
- (4) 이 인덱스는 대량 데이터를 탐색할 때 매우 효율적인가?

29. 옵티마이저와 실행계획

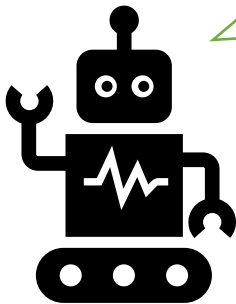
옵티마이저(=Optimizer) 란?

우리가 작성한 SQL에 대해 최적(=Optimize)의 실행 방법을 결정해주는 장치



사용자

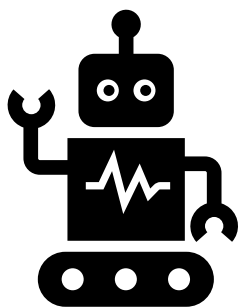
```
SELECT 직원ID , 이름 , 나이 , 입사일시  
FROM 직원  
WHERE 이름 = '이현정';
```



옵티마이저

이 쿼리를 실행할 방법이 2개가 있습니다.

- 1. 직원 테이블을 모두 조회해서 A0001을 찾는다
- 2. 인덱스를 활용해서 A0001인 데이터만 찾는다.



제가 테이블 , 인덱스 등의 통계정보를 이용해서 CPU 사용량 등을 계산해
봤는데 **최소 비용(COST)** 으로 **효율적으로 실행**하려면
2번 인덱스를 이용하는 방법이 더 좋겠습니다

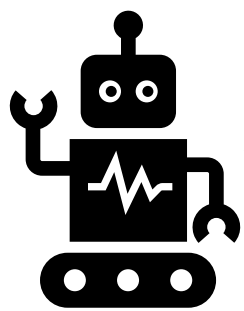
옵티마이저 (비용 기반)

직원ID	이름	나이	입사일시
A0014	공손한	50	2023/01/24 12:41:26
A0001	김철수	25	2022/03/21 00:00:00
A0002	강홍수	28	2021/09/12 00:00:00
A0003	이현정	(null)	2022/11/06 00:00:00
A0004	김선미	(null)	2020/03/11 00:00:00
A0005	문현철	34 (null)	
A0006	송대주	44	2015/07/16 00:00:00
A0007	메이슨	40	2016/08/19 00:00:00
A0008	송진아	47	2015/07/16 00:00:00
A0009	이서연	50	2013/11/23 00:00:00
A0010	김홍민	52	2013/11/23 00:00:00

1. 직원 테이블을 모두 조회해서 A0001을 찾는다

이름	직원ID	이름	나이	입사일시
강홍수	A0014	공손한	50	2023/01/24 12:41:26
공손한	A0001	김철수	25	2022/03/21 00:00:00
김선미	A0002	강홍수	28	2021/09/12 00:00:00
김철수	A0003	이현정	(null)	2022/11/06 00:00:00
김홍민	A0004	김선미	(null)	2020/03/11 00:00:00
메이슨	A0005	문현철	34 (null)	
문현철	A0006	송대주	44	2015/07/16 00:00:00
송대주	A0007	메이슨	40	2016/08/19 00:00:00
송진아	A0008	송진아	47	2015/07/16 00:00:00
이서연	A0009	이서연	50	2013/11/23 00:00:00
이현정	A0010	김홍민	52	2013/11/23 00:00:00

2. 인덱스를 활용해서 A0001인 데이터만 찾는다.



아래 실행계획(Execution plan) 대로 쿼리를 실행할까 해요

옵티마이저

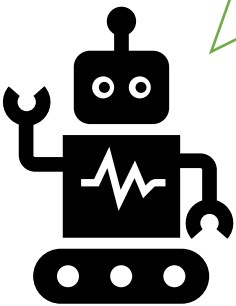
스크립트 출력 x 질의 결과 x 계획 설명 x		
SQL 0.03초		
OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	직원	BY INDEX ROWID BATCHED
INDEX	직원_이름_INDEX	RANGE SCAN
Access Predicates 이름='이현정'		

< 위 실행 계획대로 접근을 해서 데이터를 추출하겠다는 의미 >

이렇게 정하는 근거는 관련 다양한 통계정보!

옵티마이저는 두 종류가 있습니다

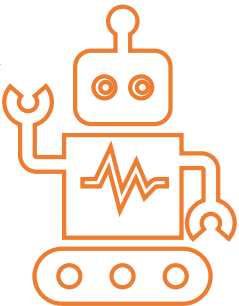
정확한 통계와 계산에
근거해서 최적의 실행
계획을 뽑아야 합니다



비용기반 옵티마이저

(Cost Based Optimizer , CBO) 128번 문제

인덱스가 있으면 무조건
인덱스를 사용한다는 등의
정해진 규칙에 맞춰서
실행계획을 뽑아야 합니다



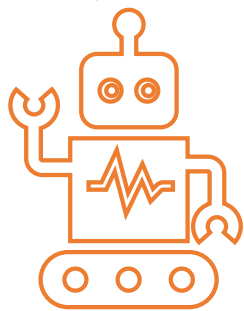
규칙기반 옵티마이저

(Rule Based Optimizer , RBO)

VS

RBO는 요즘 거의 안 씁니다. (시험은 비교용으로 나옵니다)

인덱스가 있으면 무.조.건
인덱스 사용



규칙기반 옵티마이저
(Rule Based Optimizer , RBO)

```
SELECT 직원ID , 이름 , 나이 , 입사일시
FROM 직원
WHERE 나이 BETWEEN 20 AND 50 ;
```

(현재 나이 컬럼에 대한 인덱스가 있다고 가정)

직원ID	이름	나이	입사일시
A0014	공손한	50	2023/01/24 12:41:26
A0001	김철수	25	2022/03/21 00:00:00
A0002	강홍수	28	2021/09/12 00:00:00
A0003	이현정	(null)	2022/11/06 00:00:00
A0004	김선미	(null)	2020/03/11 00:00:00
A0005	문현철	34	(null)
A0006	송대주	44	2015/07/16 00:00:00
A0007	메이슨	40	2016/08/19 00:00:00
A0008	송진아	47	2015/07/16 00:00:00
A0009	이서연	50	2013/11/23 00:00:00
A0010	김홍민	52	2013/11/23 00:00:00
A0015	새사령	(null)	(null)

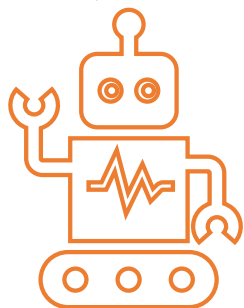
인덱스가 없을 때

나이	직원ID	이름	나이	입사일시
25	A0014	공손한	50	2023/01/24 12:41:26
28	A0001	김철수	25	2022/03/21 00:00:00
34	A0002	강홍수	28	2021/09/12 00:00:00
40	A0003	이현정	(null)	2022/11/06 00:00:00
44	A0004	김선미	(null)	2020/03/11 00:00:00
47	A0005	문현철	34	(null)
50	A0006	송대주	44	2015/07/16 00:00:00
50	A0007	메이슨	40	2016/08/19 00:00:00
52	A0008	송진아	47	2015/07/16 00:00:00
(null)	A0009	이서연	50	2013/11/23 00:00:00
(null)	A0010	김홍민	52	2013/11/23 00:00:00
(null)	A0015	새사령	(null)	(null)

인덱스가 있을 때

RBO 규칙 우선순위 15개 (지만 4개만 기억합시다)

이 규칙 우선순위에 맞춰
실행계획을 세웁니다



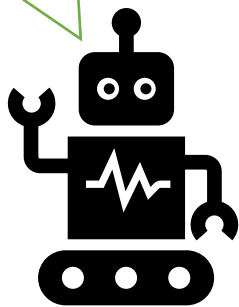
규칙기반 옵티마이저

(Rule Based Optimizer , RBO)

순위	접근 계획
1	ROWID 로 한 행(튜플)을 직접 접근하는 방법 (rowid)
...	...
4	고유키 나 primary key 로 직접 접근하는 방법 (index scan)
...	...
9	하나의 컬럼을 가진 인덱스로 접근하는 방법 (index scan)
...	...
15	테이블을 전부 뒤지기 (table full scan)

CBO는 거의 모든 DB에서 사용합니다.

비용을 계산할 때
TABLE FULL SCAN 이
더 유리하네



비용기반 옵티마이저
(Cost Based Optimizer , CBO)

```
SELECT 직원ID , 이름 , 나이 , 입사일시
FROM 직원
WHERE 나이 BETWEEN 20 AND 50 ;
```

(현재 나이 컬럼에 대한 인덱스가 있다고 가정)

인덱스가 없을 때

직원ID	이름	나이	입사일시
A0014	공손한	50	2023/01/24 12:41:26
A0001	김철수	25	2022/03/21 00:00:00
A0002	강홍수	28	2021/09/12 00:00:00
A0003	이현정	(null)	2022/11/06 00:00:00
A0004	김선미	(null)	2020/03/11 00:00:00
A0005	문현철	34	(null)
A0006	송대주	44	2015/07/16 00:00:00
A0007	메이슨	40	2016/08/19 00:00:00
A0008	송진아	47	2015/07/16 00:00:00
A0009	이서연	50	2013/11/23 00:00:00
A0010	김홍민	52	2013/11/23 00:00:00
A0015	새사령	(null)	(null)

인덱스가 있을 때

나이	직원ID	이름	나이	입사일시
25	A0014	공손한	50	2023/01/24 12:41:26
28	A0001	김철수	25	2022/03/21 00:00:00
34	A0002	강홍수	28	2021/09/12 00:00:00
40	A0003	이현정	(null)	2022/11/06 00:00:00
44	A0004	김선미	(null)	2020/03/11 00:00:00
47	A0005	문현철	34	(null)
50	A0006	송대주	44	2015/07/16 00:00:00
50	A0007	메이슨	40	2016/08/19 00:00:00
52	A0008	송진아	47	2015/07/16 00:00:00
(null)	A0009	이서연	50	2013/11/23 00:00:00
(null)	A0010	김홍민	52	2013/11/23 00:00:00
(null)	A0015	새사령	(null)	(null)

실행계획 읽는 방법 (SQL을 어떤 순서로 실행할 것인가?)

같은 계층은 위에서 아래로 , 다른 계층은 안쪽에서부터

```
SELECT A.직원ID
      , A.이름
      , B.근무지
FROM   직원 A
      , 부서 B
WHERE  A.부서ID = B.부서ID
      AND A.이름 = '이현정';
```

▶ 실행 결과 x

▶ 계획 설명 x

SQL | 0.03초

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT 7		
NESTED LOOPS 6		
NESTED LOOPS 4		
TABLE ACCESS 2	직원	BY INDEX ROWID
INDEX 1	직원_이름_INDEX	RANGE SCAN
INDEX 3	PK_부서	UNIQUE SCAN
TABLE ACCESS 5	부서	BY INDEX ROWID

실행계획으로 알 수 있는 내용

질의 결과 x

계획 설명 x

SQL 0,022초

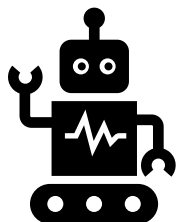
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			12	6
HASH JOIN			12	6
Access Predicates				
TABLE ACCESS	직원	FULL	12	3
TABLE ACCESS	부서	FULL	6	3

조인수행방식

엑세스(접근)방식

단계별 예상출력건수

단계별 예상 처리 비용



데이터에 접근하는 방식으로는 TABLE FULL SCAN 방식으로 접근하고 조인방식으로는 해시조인을 사용할 겁니다.
전체 예상 출력건수는 12건이며 예상 처리 비용(cost)는 6 정도 입니다.

▶ 29. 옵티마이저와 실행계획

```
SELECT A.직원ID
      , A.이름
      , B.근무지
FROM 직원 A
      , 부서 B
WHERE A.부서ID = B.부서ID
      AND A.이름 = '이현정';
```

질의 결과 x

계획 설명 x

SQL | 0.03초

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT 7		
NESTED LOOPS 6		
NESTED LOOPS 4		
TABLE ACCESS 2	직원	BY INDEX ROWID
INDEX 1	직원_이름_INDEX	RANGE SCAN
INDEX 3	PK_부서	UNIQUE SCAN
TABLE ACCESS 5	부서	BY INDEX ROWID

< 직원_이름_INDEX >

< 직원 테이블 >

< PK_부서 >

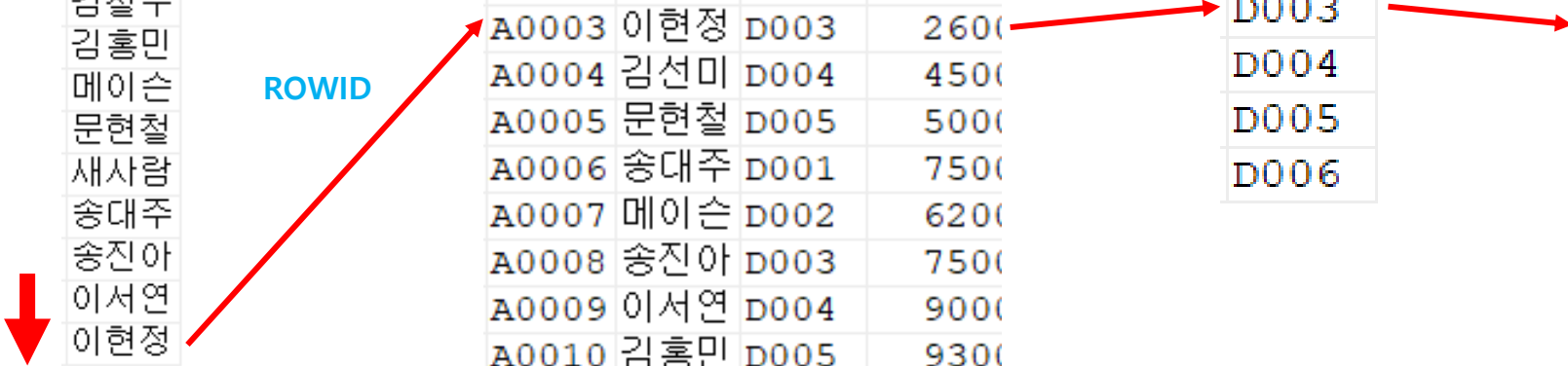
< 부서 테이블 >

이름
강홍수
공손한
김선미
김철수
김홍민
메이슨
문현철
새사람
송대주
송진아
이서연
이현정

직원ID	이름	부서ID	연봉
A0014	공손한	D002	7000
A0001	김철수	D001	2800
A0002	강홍수	D002	3000
A0003	이현정	D003	2600
A0004	김선미	D004	4500
A0005	문현철	D005	5000
A0006	송대주	D001	7500
A0007	메이슨	D002	6200
A0008	송진아	D003	7500
A0009	이서연	D004	9000
A0010	김홍민	D005	9300
A0015	새사람	D001	(null)

부서ID
D001
D002
D003
D004
D005
D006

부서ID	부서명	근무지
D001	인사부	서울
D002	급여부	서울
D003	전략기획부	경기
D004	SI사업부	경기
D005	사업부	제주
D006	인프라서비스부	서울



31. 조인 수행 원리

조인이란 여러 테이블을 한번에 조회하기 위한 결합 방식입니다. (두 테이블 씩 조인)

```
SELECT A.직원ID
      , A.이름
      , B.근무지
FROM   직원 A
      , 부서 B
WHERE  A.부서ID = B.부서ID
      AND A.이름 = '이현정';
```



직원 테이블과 부서 테이블을 조인해서
한번에 데이터를 조회해오려고 함

두 개 이상의 테이블을 조인할 때 수행 원리는
3가지가 있습니다.

[1] NL (Nested Loop) Join

[2] Sort Merge Join

[3] Hash Join

[1] NL (Nested Loop) Join

= 자바의 이중 for문과 비슷한 개념

```
for ( 직원 테이블 ) {  
  for ( 부서 테이블 ) {  
  
    if( 직원.부서id == 부서.부서id) {  
      --해당 튜플은 조건에 맞으니 출력하세요!  
    }  
  
  }  
}
```

< 직원_이름_INDEX >

이름
강홍수
공손한
김선미
김철수
김홍민
메이슨
문현철
새사람
송대주
송진아
이서연
이현정

< 직원 테이블 >

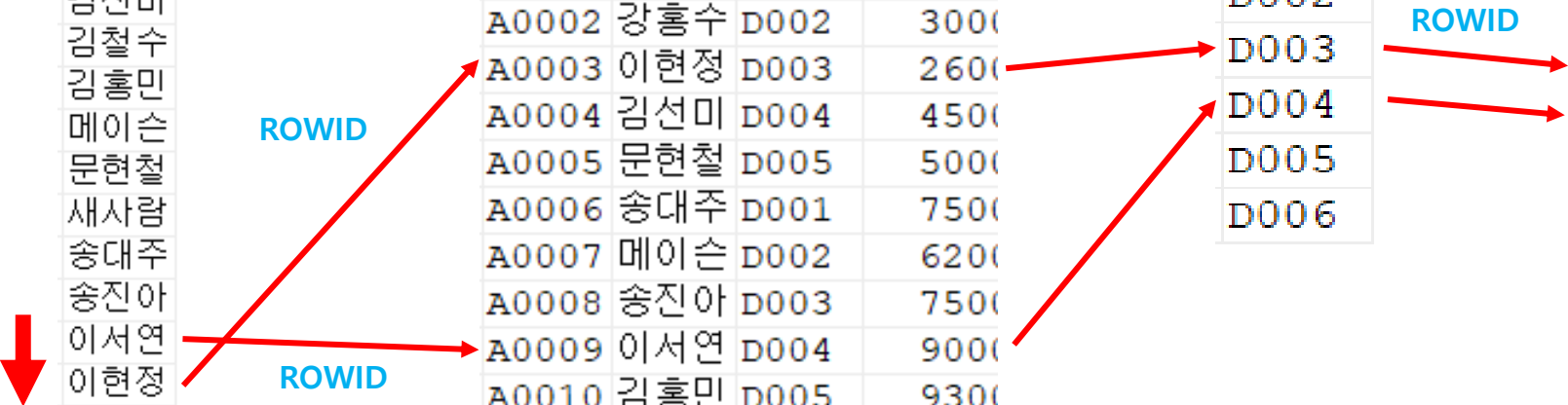
직원ID	이름	부서ID	연봉
A0014	공손한	D002	7000
A0001	김철수	D001	2800
A0002	강홍수	D002	3000
A0003	이현정	D003	2600
A0004	김선미	D004	4500
A0005	문현철	D005	5000
A0006	송대주	D001	7500
A0007	메이슨	D002	6200
A0008	송진아	D003	7500
A0009	이서연	D004	9000
A0010	김홍민	D005	9300
A0015	새사람	D001	(null)

< PK_부서 >

부서ID
D001
D002
D003
D004
D005
D006

< 부서 테이블 >

부서ID	부서명	근무지
D001	인사부	서울
D002	급여부	서울
D003	전략기획부	경기
D004	SI사업부	경기
D005	사업부	제주
D006	인프라서비스부	서울



[1] NL (Nested Loop) Join

= 자바의 이중 for문과 비슷한 개념

질의 결과 x

계획 설명 x

SQL 0.03초

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
NESTED LOOPS		
NESTED LOOPS		
TABLE ACCESS	직원	BY INDEX ROWID
INDEX	직원_이름_INDEX	RANGE SCAN
INDEX	PK_부서	UNIQUE SCAN
TABLE ACCESS	부서	BY INDEX ROWID

<실행계획에서 보이는 NL 조인 기법>

[1] NL (Nested Loop) Join 장단점

장점

인덱스를 활용, 랜덤 액세스를 통해 **소량의 데이터 추출** 가능

대부분의 웹사이트에서는 NL조인을 주로 활용하여 활용도가 높다

단점

인덱스가 있어야만 사용할 수 있다.

소량의 데이터가 아니라면 NL조인은 매우 비효율적이다.

[2] Sort Merge Join

= 조인할 두 테이블을 **조인 키 컬럼 기준으로 정렬해놓고** 조인

```
SELECT A.직원ID
      , A.이름
      , B.근무지
FROM   직원 A
      , 부서 B
WHERE  A.부서ID = B.부서ID
      AND A.이름 = '이현정';
```

(1) 먼저 두 테이블을
조인 컬럼인 부서ID 기준으로 정렬하자!

직원ID	이름	부서ID
A0001	김철수	D001
A0015	새사람	D001
A0006	송대주	D001
A0002	강홍수	D002
A0014	공손한	D002
A0007	메이슨	D002
A0008	송진아	D003
A0003	이현정	D003
A0009	이서연	D004
A0004	김선미	D004
A0010	김홍민	D005
A0005	문현철	D005

<정렬된 직원 테이블>

부서ID	부서명	근무지
D001	인사부	서울
D002	급여부	서울
D003	전략기획부	경기
D004	SI사업부	경기
D005	사업부	제주
D006	인프라서비스부	서울

<정렬된 부서 테이블>

[2] Sort Merge Join

= 조인할 두 테이블을 **조인 키 컬럼 기준으로 정렬해놓고** 조인

```
SELECT A.직원ID
      , A.이름
      , B.근무지
FROM  직원 A
      , 부서 B
WHERE A.부서ID = B.부서ID
      AND A.이름 = '이현정';
```

직원ID	이름	부서ID
A0001	김철수	D001
A0015	새사람	D001
A0006	송대주	D001
A0002	강홍수	D002
A0014	공손한	D002
A0007	메이슨	D002
A0008	송진아	D003
A0003	이현정	D003
A0009	이서연	D004
A0004	김선미	D004
A0010	김홍민	D005
A0005	문현철	D005

<정렬된 직원 테이블>

부서ID	부서명	근무지
D001	인사부	서울
D002	급여부	서울
D003	전략기획부	경기
D004	SI사업부	경기
D005	사업부	제주
D006	인프라서비스부	서울

<정렬된 부서 테이블>

[2] Sort Merge Join

= 조인할 두 테이블을 **조인 키 컬럼 기준으로 정렬해놓고** 조인

질의 결과 x

계획 설명 x

SQL | 0.025초

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
MERGE JOIN		
SORT		JOIN
TABLE ACCESS	직원	BY INDEX ROWID BATCHED
INDEX	직원_이름_INDEX	RANGE SCAN
SORT		JOIN
Access Predicates		
Filter Predicates		
TABLE ACCESS	부서	FULL

<실행계획에서 보이는 SORT MERGE 조인 기법>

[2] Sort Merge Join 장단점

장점

인덱스가 없어도 조인을 할 수 있다. (FULL TABLE SCAN)

대량의, 넓은 범위의 데이터도 빠르게 조인이 가능하다.

동등 조인과 비동등 조인 둘 다 사용할 수 있다. (NL도 가능)

만약 이미 정렬되어 있는 테이블이라면 다시 정렬할 필요가 없다. (속도 업)

단점

정렬해야하는 데이터가 너무 많으면 비효율적으로 변한다. (정렬 공간이..)

[3] Hash join

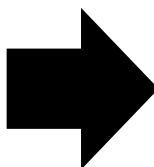
조인할 두 테이블을 **해싱 기법을 이용하여** 조인한다.

```
SELECT A.직원ID
      , A.이름
      , B.근무지
FROM 직원 A
      , 부서 B
WHERE A.부서ID = B.부서ID
      AND A.이름 = '이현정';
```

(1) 크기가 작은 테이블을 선택해 조인 키 컬럼으로 해시테이블을 만든다

직원ID	이름	부서ID	연봉
A0014	공손한	D002	7000
A0001	김철수	D001	2800
A0002	강홍수	D002	3000
A0003	이현정	D003	2600
A0004	김선미	D004	4500
A0005	문현철	D005	5000
A0006	송대주	D001	7500
A0007	메이슨	D002	6200
A0008	송진아	D003	7500
A0009	이서연	D004	9000
A0010	김홍민	D005	9300
A0015	새사람	D001	(null)

부서ID	부서명	근무지
D001	인사부	서울
D002	급여부	서울
D003	전략기획부	경기
D004	SI사업부	경기
D005	사업부	제주
D006	인프라서비스부	서울



해시(키)	결과
H (D001)	인사부 , 서울
H (D002)	급여부 , 서울
H (D003)	전략기획부 , 경기
H (D004)	SI사업부 , 경기
H (D005)	사업부 , 제주
H (D006)	인프라서비스부 , 서울

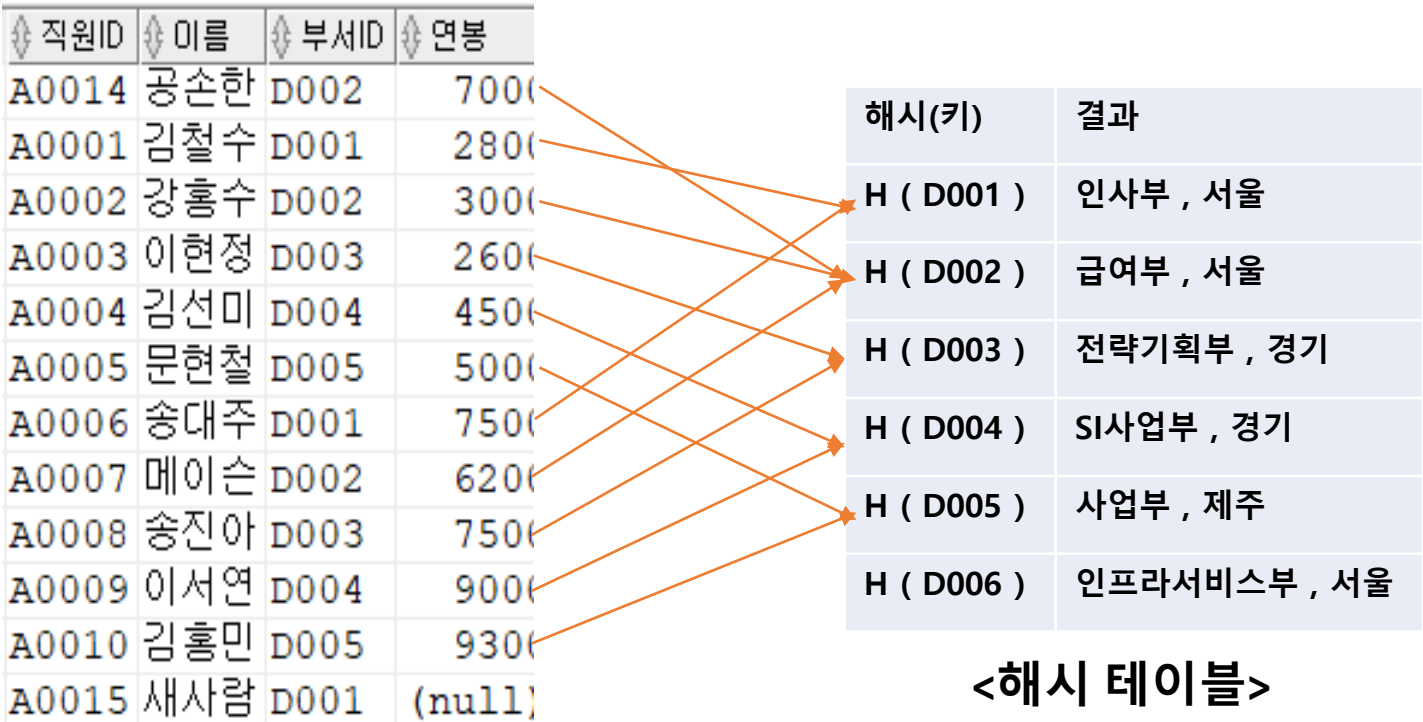
<해시 테이블>

[3] Hash join

조인할 두 테이블을 **해싱 기법을 이용**하여 조인한다.

(2) 해시테이블을 기준으로 다른 테이블과 값을 조인한다.

```
SELECT A.직원ID
      , A.이름
      , B.근무지
FROM 직원 A
      , 부서 B
WHERE A.부서ID = B.부서ID
      AND A.이름 = '이현정';
```



이때 동일한 키로 해시 테이블을 조회할 경우
이미 만들어 놓은 테이블 내용을 그대로 사용

(NL 조인은 똑같은 키 값이 나와도
똑같이 반복적으로 해당 값을 조회해야함
예시)

[3] Hash join

조인할 두 테이블을 **해싱 기법을 이용**하여 조인한다.

▶ 질의 결과 x

📄 계획 설명 x

📌 SQL

👁

| 0.022초

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
HASH JOIN		
Access Predicates		
TABLE ACCESS	직원	FULL
TABLE ACCESS	부서	FULL

<실행계획에서 보이는 hash 조인 기법>

[3] Hash join 장단점

장점

인덱스가 없어도 조인을 할 수 있다. (FULL TABLE SCAN)

대량의, 넓은 범위의 데이터도 빠르게 조인이 가능하다.

Sort merge 조인처럼 따로 정렬할 필요도 없다.

단점

동등 조인만 사용가능하다 (nl 과 sort merge 는 비동등도 가능함)

CPU 연산을 많이 잡아먹어서 자주 사용하는 쿼리에는 쓰기 어렵다 (보통 프로그램은 NL을 애용함)

실행 계획이 달라도 결과는 똑같습니다.

```
SELECT /*+ ORDERED use_nl(B) */ A.직원ID
      , A.이름
      , B.근무지
FROM   직원 A
      , 부서 B
WHERE  A.부서ID = B.부서ID

SELECT /*+ ORDERED use_merge(B) */ A.직원ID
      , A.이름
      , B.근무지
FROM   직원 A
      , 부서 B
WHERE  A.부서ID = B.부서ID

SELECT /*+ ORDERED use_hash(B) */ A.직원ID
      , A.이름
      , B.근무지
FROM   직원 A
      , 부서 B
WHERE  A.부서ID = B.부서ID
```

직원ID	이름	근무지
A0014	공손한	서울
A0001	김철수	서울
A0002	강홍수	서울
A0003	이현정	경기
A0004	김선미	경기
A0005	문현철	제주
A0006	송대주	서울
A0007	메이슨	서울
A0008	송진아	경기
A0009	이서연	경기
A0010	김홍민	제주
A0015	새사람	서울

<NL조인>

직원ID	이름	근무지
A0001	김철수	서울
A0015	새사람	서울
A0006	송대주	서울
A0002	강홍수	서울
A0014	공손한	서울
A0007	메이슨	서울
A0008	송진아	경기
A0003	이현정	경기
A0009	이서연	경기
A0004	김선미	경기
A0010	김홍민	제주
A0005	문현철	제주

<SORT MERGE 조인>

직원ID	이름	근무지
A0001	김철수	서울
A0006	송대주	서울
A0015	새사람	서울
A0014	공손한	서울
A0002	강홍수	서울
A0007	메이슨	서울
A0003	이현정	경기
A0008	송진아	경기
A0004	김선미	경기
A0009	이서연	경기
A0005	문현철	제주
A0010	김홍민	제주

<HASH 조인>

SQLD 특강

29 ~ 31 SQL 최적화 기본 원리 종료

강사 강태우

