

SQL 쿼리 튜닝

튜닝! 왜 배우야 하는가?



1초 이내 수행되던 서비스 쿼리

```
SELECT A
FROM emp e
      ,dept d
WHERE e.dept_id = d.id
      AND e.status = 'Y'
```

갑자기 10분 이상 수행



장애

조인없이 서비스DB에서 쿼리 수행

```
SELECT A
FROM emp e
      ,dept d
WHERE e.dept_id = d.id
      AND e.status = 'Y'
```

결과가 안나옴



장애

**이번 교육 목표는
쿼리의 이상유무를 확인할 수 있으며,
튜닝을 통해 성능을 향상 시키자!!!**

목차

1. 튜닝을 위한 기본지식 쌓기
2. SQL 분석 및 성능관련 해석
3. 실사례를 통한 SQL 성능향상 살펴보기
4. MySQL과 Oracle 개발상 차이 및 유의사항

1. 튜닝을 위한 기본지식 쌓기

- **DB 구조**

- SQL 이해
- 옵티마이저(Optimizer) 이해
- 힌트(Hint)
- 인덱스(Index)
- 조인(Join)

2. SQL 분석 및 성능관련 해석

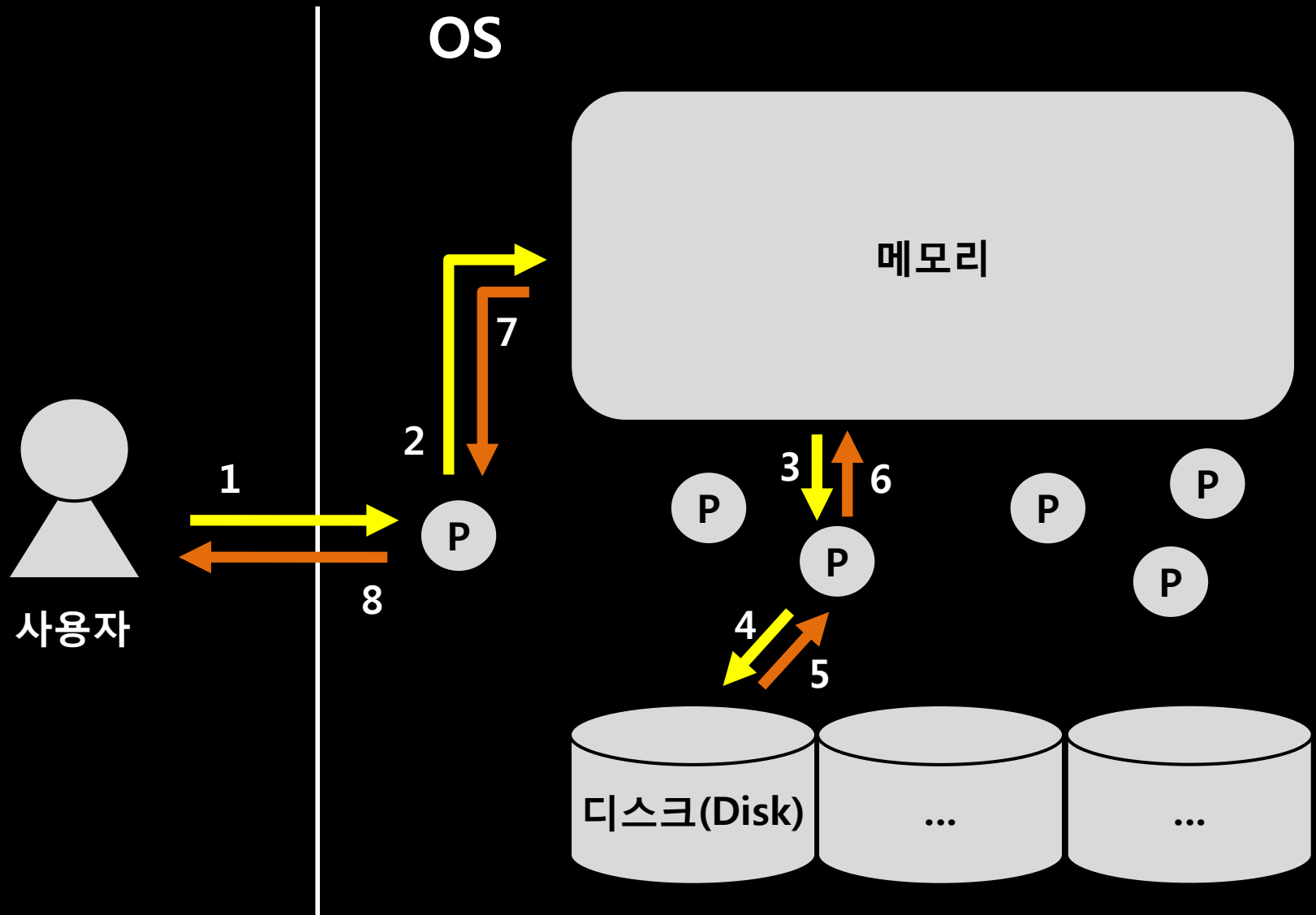
3. 실 사례를 통한 SQL성능향상 살펴보기

4. MySQL과 Oracle 개발 차이점 및 유의사항

DB구조



DB구조



목차

1. 튜닝을 위한 기본지식 쌓기

- DB구조
- **SQL 이해**
- 옵티마이저(Optimizer)
- 힌트(Hint)
- 인덱스(Index)
- 조인(Join)

2. SQL 분석 및 성능관련 해석

3. 실 사례를 통한 SQL성능향상 살펴보기

4. MySQL과 Oracle 개발 차이점 및 유의사항

SQL 이해

- SQL 실행 순서
 - 가장 기초지만, 가장 중요합니다.

	COL1	COL2	COL3
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7
8	8	8	8
9	9	9	9
10	10	10	10

```
SELECT *  
FROM TEST01  
WHERE ROWNUM <= 2  
ORDER BY COL3 DESC;
```



COL1	COL2	COL3
2	2	2
1	1	1

```
SELECT *  
FROM TEST01  
ORDER BY COL3 DESC  
LIMIT 2;
```



COL1	COL2	COL3
9	9	9
8	8	8

SQL 이해

SELECT

5

FROM

1

WHERE

2

GROUP BY

3

HAVING

4

ORDER BY

6

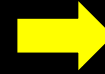
LIMIT

7

SQL 이해

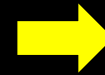
	COL1	COL2	COL3
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7
8	8	8	8
9	9	9	9
10	10	10	10

```
SELECT *  
FROM TEST01  
WHERE ROWNUM<=2  
ORDER BY COL3 DESC;
```



COL1	COL2	COL3
2	2	2
1	1	1

```
SELECT *  
FROM TEST01  
ORDER BY COL3 DESC  
LIMIT 2;
```



COL1	COL2	COL3
9	9	9
8	8	8

목차

1. 튜닝을 위한 기본지식 쌓기

- DB구조
- SQL 이해
- **옵티마이저(Optimizer)**
- 힌트(Hint)
- 인덱스(Index)
- 조인(Join)

2. SQL 분석 및 성능관련 해석

3. 실 사례를 통한 SQL성능향상 살펴보기

4. MySQL과 Oracle 개발상 차이 및 유의사항

옵티마이저(Optimizer)

- SQL을 가장 빠르고 효율적(최저비용)으로 수행할 수 있게 도와주는 것
- 옵티마이저 종류
 - 규칙기반 옵티마이저(Rule-Based Optimizer, RBO)
 - 비용기반 옵티마이저(Cost-Based Optimizer, CBO)
- 그럼 어떻게 도와주나요?
 - 인덱스 사용할 조건 있나?
 - 인덱스 사용하면 비용이 줄어드나?
 - 어떤 테이블을 먼저 읽을까?
 - 순차적으로 읽으면서 조인할까?
 - 쿼리를 변경할 필요는 없을까?

이런 정보들은 어디서 가져올까???

옵티마이저(Optimizer)

- 통계정보란?
 - 옵티마이저가 실행계획을 수립 시 참조하는 정보
- 테이블, 인덱스, 컬럼, 시스템 통계정보를 가지고 있음
 - 테이블 통계 : 크기, row 수, 한행당 평균 크기 등
 - 인덱스 통계 : 블럭수, 인덱스 형태 등
 - 컬럼 통계 : 컬럼 상태, 값의 수, 최저 값, 최고 값 등
 - 시스템 통계 : CPU 속도, 디스크 I/O 속도 등

**이런 정보들을 이용해서
좋은 실행계획을 찾는다.
하지만 현실은...**

옵티마이저(Optimizer)

- 옵티마이저의 한계
 - 사람이 만든 소프트웨어 엔진에 불과하며 결코 완벽할 수 없음
 - 부정확한 통계정보
 - 효과적이지 못한 인덱스 구조

어떻게 할까요?

방법은 있습니다.

그래서 교육을 받는 거죠.

목차

1. 튜닝을 위한 기본지식 쌓기

- DB구조
- SQL 이해
- 옵티마이저(Optimizer)
- **힌트(Hint)**
- 조인(Join)
- 인덱스(Index)

2. SQL 분석 및 성능관련 해석

3. 실 사례를 통한 SQL성능향상 살펴보기

4. MySQL과 Oracle 개발상 차이 및 유의사항

힌트(Hint)

- 더 좋은 실행계획으로 유도하는 방법
- 힌트 종류와 구체적인 사용법은 DBMS마다 다르다.
- 무시되는 경우도 발생함
 - 문법적으로 의미적으로 안 맞게 힌트를 기술
 - 의미적으로 안 맞게 힌트를 기술
 - 잘 못된 참고
 - 논리적으로 불가능한 액세스 경로
 - 버그

**어떤 종류가 있으며
어떻게 사용할까요?**

힌트(Hint)

- MySQL

SQL_CACHE	Query 결과를 캐쉬
SQL_NO_CACHE	Query 결과를 캐쉬 안함
STRAIGHT_JOIN	FROM에 기술된 테이블 순서로 join 해라
USE INDEX (index_name, ...)	지정한 Index 사용
IGNORE INDEX (index_name, ...)	지정한 Index 무시
FORCE INDEX (index_name, ...)	USE INDEX와 별차이는 없음 옵티마이저에게 미치는 영향이 더 강함.

힌트(Hint)

- MySQL

```
select SQL_CACHE ...
```

```
select SQL_NO_CACHE ...
```

```
select STRAIGHT_JOIN ...
```

```
select ... from emp e USE INDEX (idx_name)
```

```
IGNORE INDEX (idx_name)
```

```
FORCE INDEX (idx_name)
```

힌트(Hint)

- Oracle

RESULT_CACHE	Query 결과 저장 (11NF)
NO_RESULT_CACHE	Query 결과 저장 안함 (11NF)
ORDERED	FROM에 기술된 테이블 순서로 join 해라
USE_HASH (table [table])	HASH JOIN으로 유도
USE_NL (table [table])	NL JOIN으로 유도
USE_MERGE (table [table])	SORT-MERGE JOIN으로 유도
INDEX(table index)	지정된 Index를 사용하도록 지정
INDEX_ASC(table index)	지정된 Index를 사용하되, ascending 으로 읽기
INDEX_DESC(table index)	지정된 Index를 사용하되, descending 으로 읽기

힌트(Hint)

- Oracle

```
select /*+ RESULT_CACHE */ ...
```

```
select /*+ ORDERED */ ...
```

```
select /*+ USE_HASH (e d) */ from emp e, dept d ...
```

```
select /*+ INDEX(e idx_name) */ from emp e, dept d ...
```

```
select /*+ INDEX_ASC(e idx_name) */ from emp e ...
```

```
select /*+ ORDERED
```

```
        USE_NL(e d) INDEX_ASC(e idx_name) */
```

```
    from emp e, dept d
```

```
Where e.dept_id = d.id
```

목차

1. 튜닝을 위한 기본지식 쌓기

- DB구조
- SQL 이해
- 옵티마이저(Optimizer)
- 힌트(Hint)
- **인덱스(Index)**
- 조인(Join)

2. SQL 분석 및 성능관련 해석

3. 실 사례를 통한 SQL성능향상 살펴보기

4. MySQL과 Oracle 개발상 차이 및 유의사항

인덱스(Index)

- SQL 수행 시 주어진 쿼리를 해석하는 데 사용

- DB

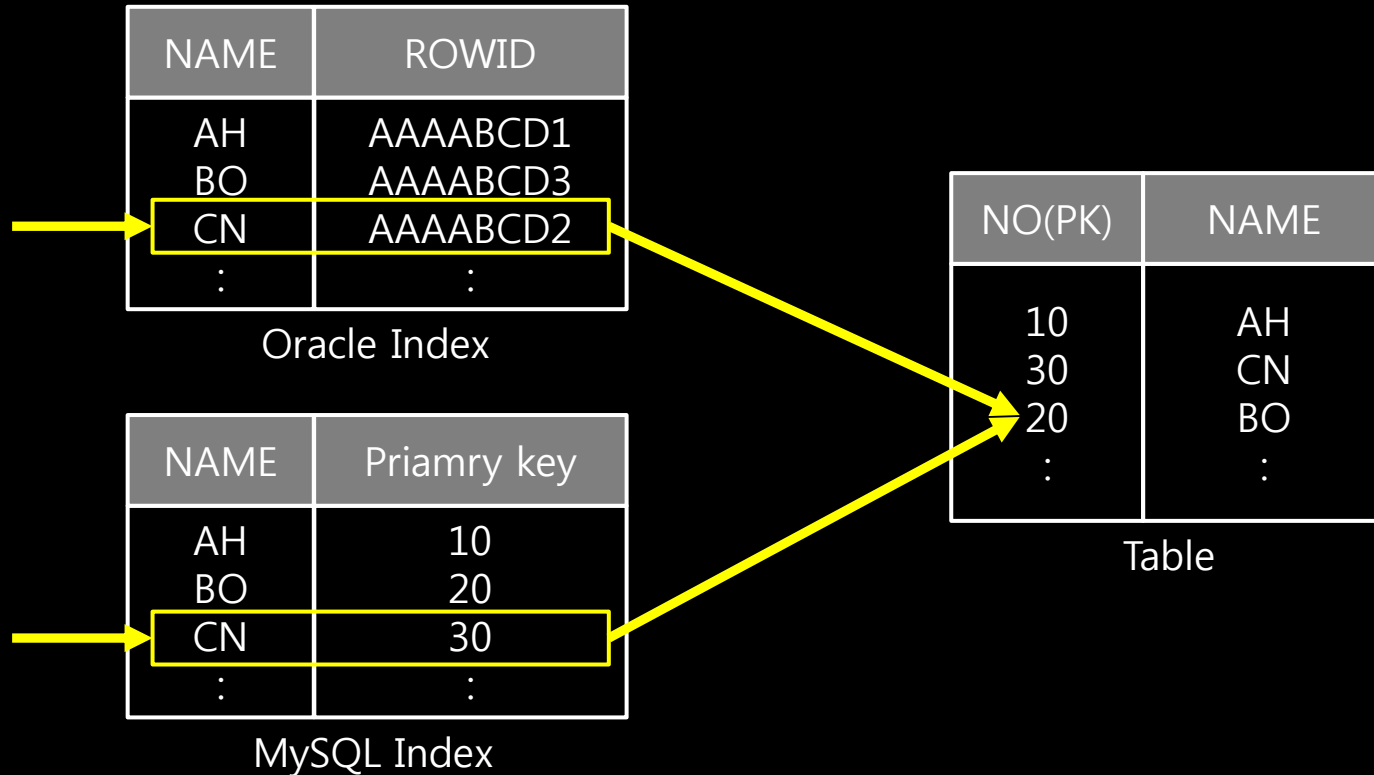
- 질의

- 종류

INDEX		
다음과 같이 정렬된 인덱스		
A~B		
absolute method	188	chage
Active mode	921	chattr
Anonymous	900	chgrp
APM 연동	1040	chkconfig
arp	681	chown
at	309	clear
atd	305	Cluster Suite
AWStats	1027	cmp
	306	continue
	308	cp
	568	cpio
		cron
		DAS
		Data Block

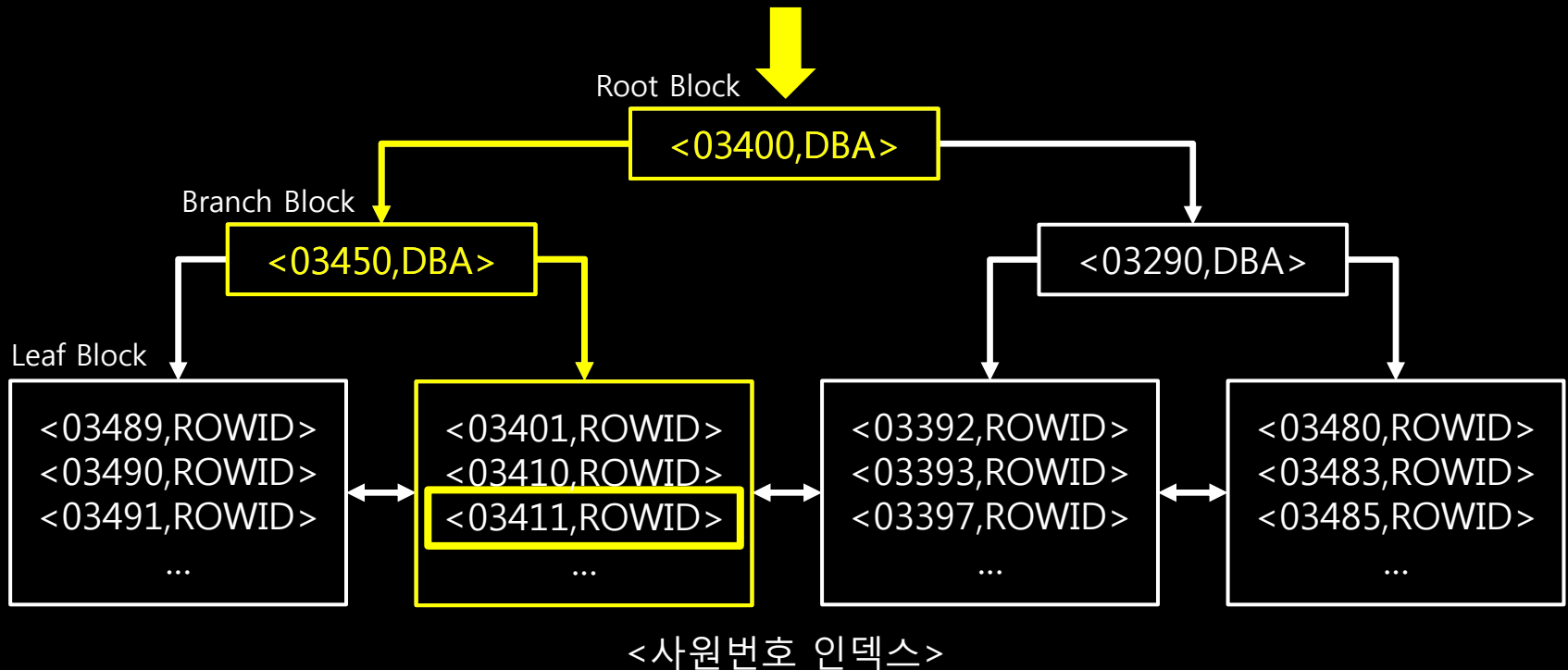
인덱스(Index)

- 인덱스의 데이터가 테이블을 찾아 가는 방법은?
 - Oracle : ROWID
 - MySQL : Primary key (Unique Key, 자동 생성된 기본키)



인덱스(Index) – B*Tree Index

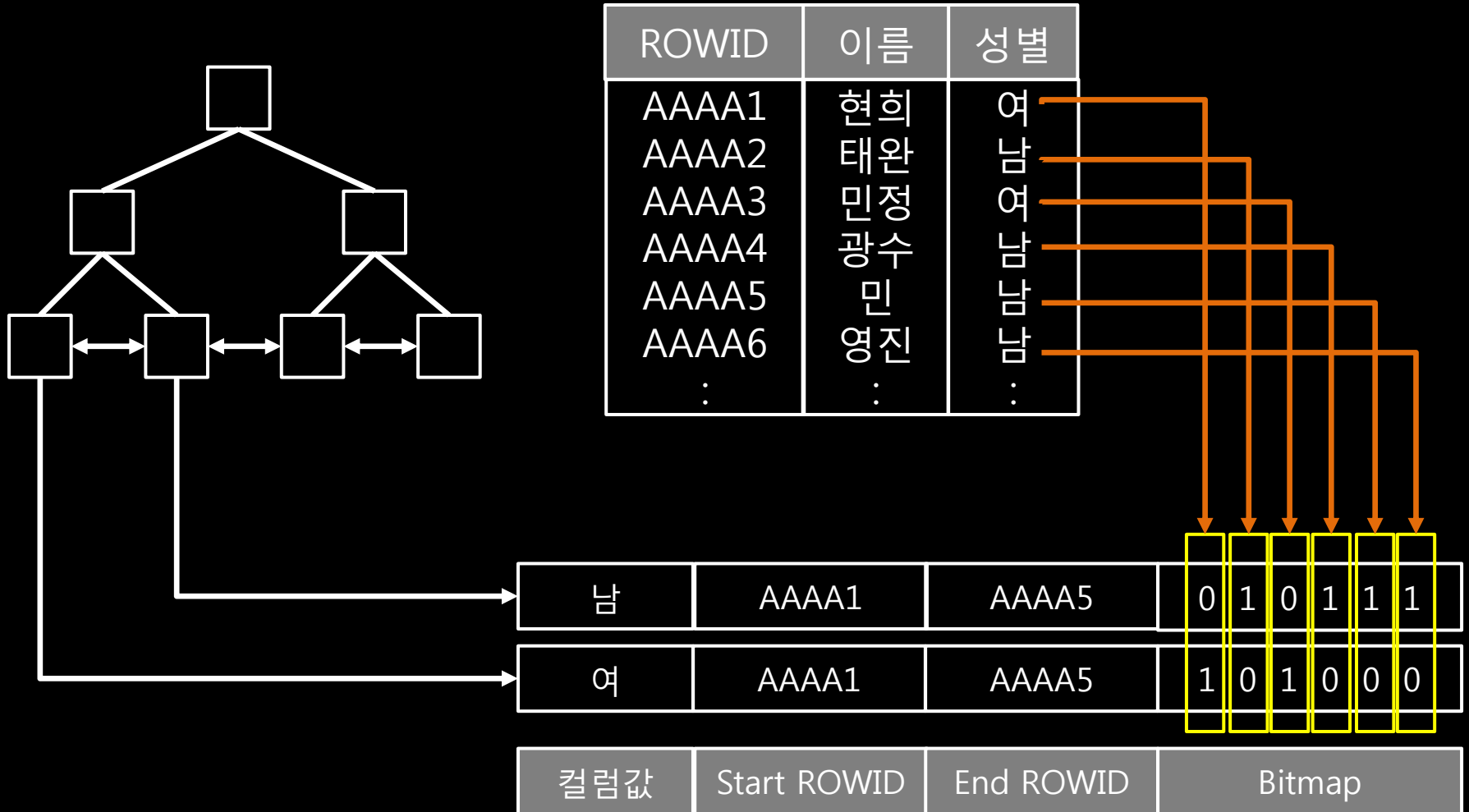
- B*Tree Index (Balanced Tree Index)



SELECT * FROM 사원 WHERE 사원번호 = '03411'

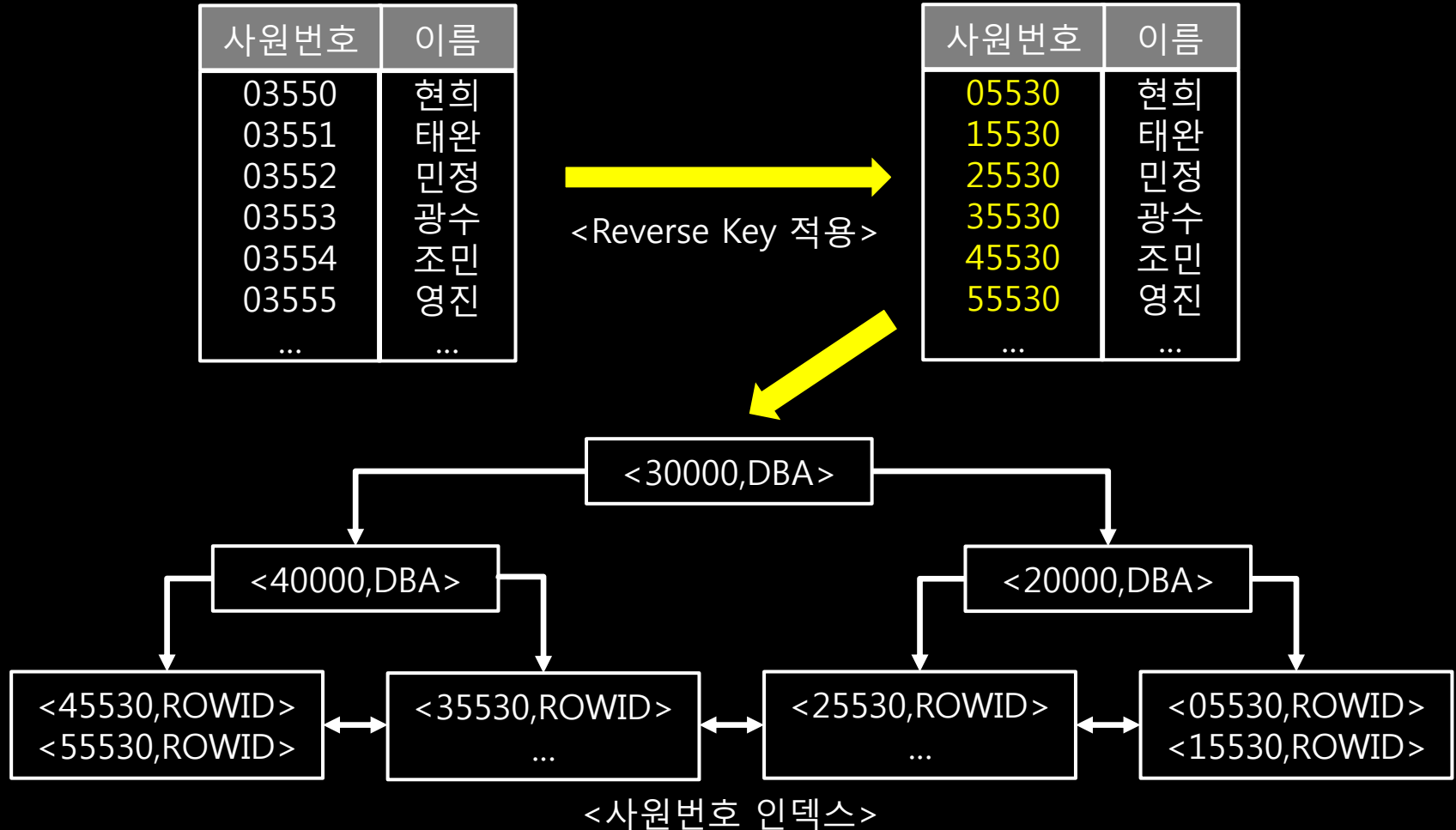
인덱스(Index) - Bitmap Index

- Bitmap Index



인덱스(Index) - Reverse Key Index

- Reverse Key Index



인덱스(Index) - Function Based Index

- Function Based Index(FBI)

INDEX : 입사일자

SELECT * FROM 사원 WHERE **substr(입사일자, 0, 6)** = '201311'

ROWID	이름	입사일자
AAAA1	현희	20131021
AAAA2	태완	20130911
AAAA3	민정	20120119
AAAA4	광수	20131030
AAAA5	민	20131001
AAAA6	영진	20110727
:	:	:

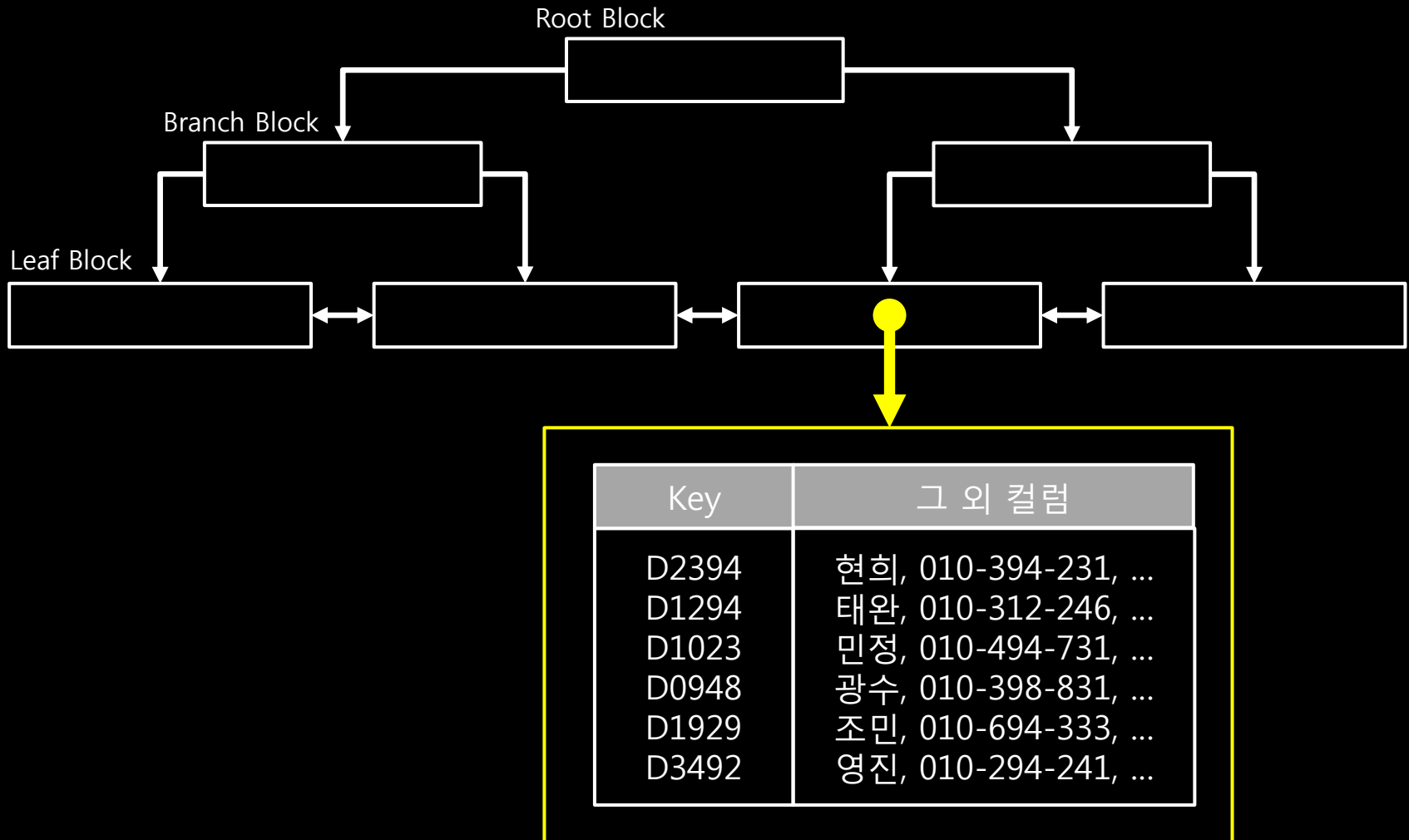
<사원 테이블>

Create index ...
(substr(입사일자,0,6))

ROWID	입사일자
AAAA1	201310 21
AAAA2	201309 11
AAAA3	201201 19
AAAA4	201310 30
AAAA5	201310 01
AAAA6	201107 27
:	:

인덱스(Index) - IOT(Index Organized Table)

- IOT(Index Organized Table)



인덱스(Index)

- Index scan시 Table Row의 3~5%이상을 사용하면 성능이 저하됨
- Index scan을 많이하면 왜 성능저하가 될까?
 - Random Access가 많이 발생하여 성능저하가 발생
- Random Access란?
 - Index scan을 통한 Table Access시
Single Block I/O를 발생시키는 Access를 말함.
- Random Access 감소방법
 - 최적의 Index 선정
 - Cluster Factor 최적화

인덱스(Index)

- Random Access

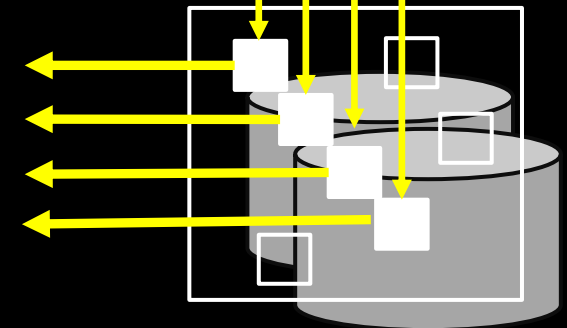
```
SELECT 고객명, 가입일자  
FROM 고객거래  
WHERE 고객번호 = '10'
```

고객번호	가입일자	ROWID
10	20130911	AAAA1
10	20130921	AAAA2
10	20141019	AAAA3
10	20131030	AAAA4
11	20130901	AAAA5
12	20111227	AAAA6
⋮	⋮	⋮

<고객 Index>

Random Access

해당 DB Block 만 Access

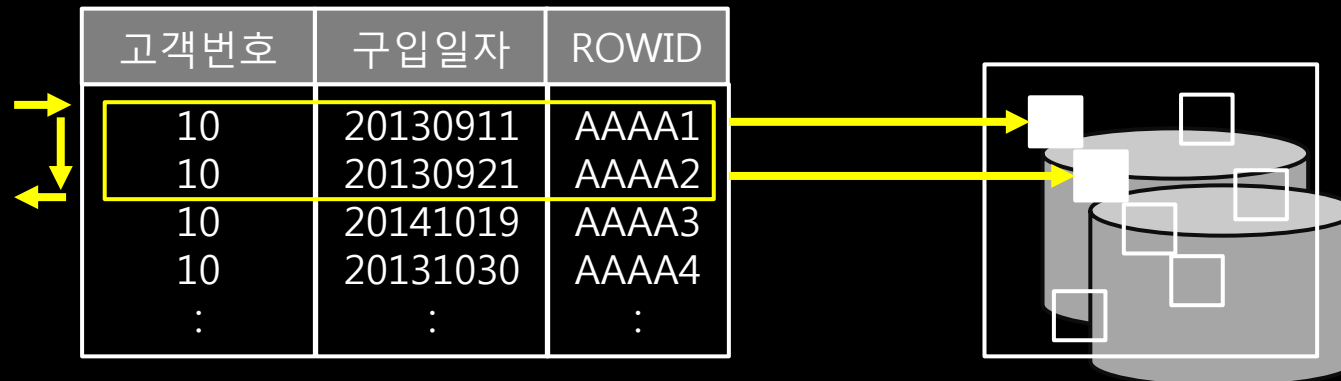
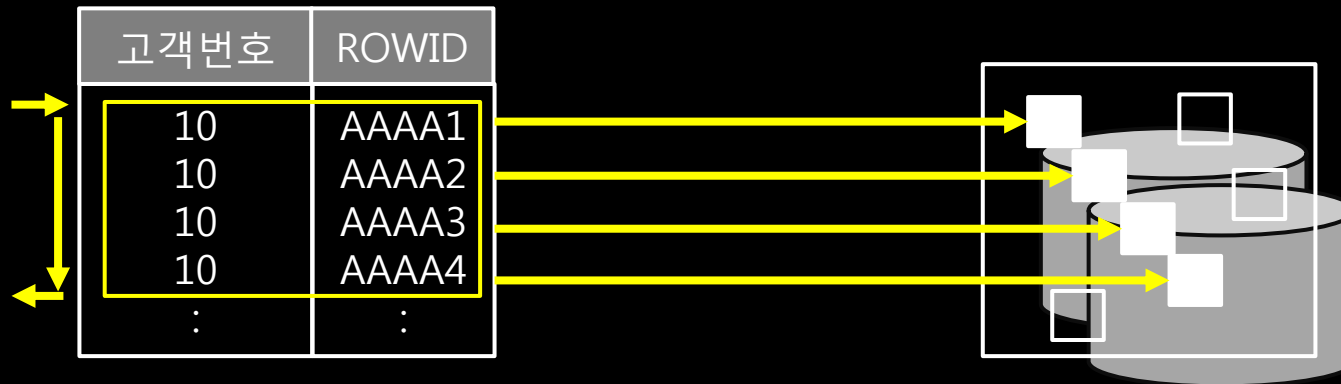


<디스크>

인덱스(Index)

- 최적의 Index 선정

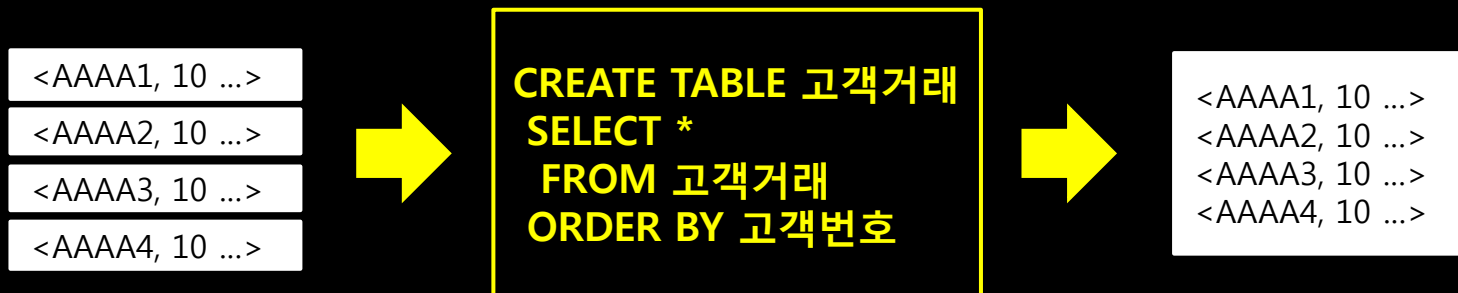
```
SELECT 고객명, 가입일자  
FROM 고객거래  
WHERE 고객번호 = '10'  
AND 가입일자 between '20130911' AND '20130921'
```



인덱스(Index)

- Cluster Factor 최적화
 - 주기적인 TABLE의 재구성 (Order By 이용)
 - Partition Table
 - Cluster Table
 - Index-Organized Table(IOT)

```
SELECT 고객명, 가입일자  
FROM 고객거래  
WHERE 고객번호 = '10'  
AND 가입일자 between '20130911' AND '20130921'
```



인덱스(Index)

- 인덱스 대상 컬럼 기준은?
 - 특정범위/순서의 데이터 조회가 필요한 경우
 - 수행속도에 영향을 미칠 것으로 예상되는 컬럼
 - Join시 연결고리가 되는 컬럼
 - Access 유형에 자주 등장하는 컬럼

인덱스(Index)

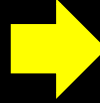
- 조건별 인덱스 사용유무 확인

	Index Key (ID+REGDATE)
WHERE ID = x	O
WHERE ID = x AND REGDATE = y	O
WHERE ID = x ORDER BY REGDATE	O
WHERE ... ORDER BY ID, REGDATE	O
SELECT REGDATE WHERE ID = x	O
WHERE REGDATE = x	X
WHERE ID LIKE 'x%';	O
WHERE ID LIKE '%x';	X
WHERE function(ID) = x	X

인덱스(Index)

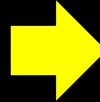
- 조건별 인덱스 사용유무 확인

Index : job
SELECT dept, ename
FROM emp
WHERE SUBSTR(job, 1, 4)='SALE'



Index : job
SELECT dept, ename
FROM emp
WHERE **job LIKE 'SALE%'**

Index : sal
SELECT dept, ename
FROM emp
WHERE sal*12 = 100000



Index : sal
SELECT dept, ename
FROM emp
WHERE **sal = 100000/12**

Index : register_date
SELECT dept, ename
FROM emp
WHERE to_cahr(register_date,'yyyymmdd') = '20100425'



Index : register_date
SELECT dept, ename
FROM emp
WHERE **register_date >= to_date(200100425,'yyyymmdd')**
AND register_date < to_date(200100425,'yyyymmdd') +1

인덱스(Index)

- 조건별 인덱스 사용유무 확인

Index : job
SELECT dept, ename
FROM emp
WHERE SUBSTR(job, 1, 4)='SALE'



Index : job
SELECT dept, ename
FROM emp
WHERE **job LIKE 'SALE%'**

Index : sal
SELECT dept, ename
FROM emp
WHERE sal*12 = 100000



Index : sal
SELECT dept, ename
FROM emp
WHERE **sal = 100000/12**

Index : register_date
SELECT dept, ename
FROM emp
WHERE to_cahr(register_date,'yyyymmdd') = '20100425'



Index : register_date
SELECT dept, ename
FROM emp
WHERE **register_date >= to_date(200100425,'yyyymmdd')**
AND register_date < to_date(200100425,'yyyymmdd') +1

목차

1. 튜닝을 위한 기본지식 쌓기

- DB구조
- SQL 이해
- 옵티마이저(Optimizer)
- 힌트(Hint)
- 인덱스(Index)
- **조인(Join)**

2. SQL 분석 및 성능관련 해석

3. 실 사례를 통한 SQL성능향상 살펴보기

4. MySQL과 Oracle 개발 차이점 및 유의사항

조인(Join)

- 조인(Join)이란?
 - 집합의 공통 부분이나 특정부분을 가지고 새로운 집합 만들기
- 조인 방법(join Method)
 - Nested-Loop Join
 - Hash Join
 - Sort Merge Join

조인(Join) - Nested-Loop Join

- Nested-Loop Join 이란?
 - 처리 범위만큼 순차적으로 Loop를 돌면서 조인하는 방식
- OLTP와 같은 적은 데이터 처리시 유리함.
- 조인시 Index에 의해 성능을 결정
 - Index scan이 아닐때 심각한 성능 저하가 발생함.
- MySQL은 오직 Nested-Loop join만을 지원함.

조인(Join) - Nested-Loop Join

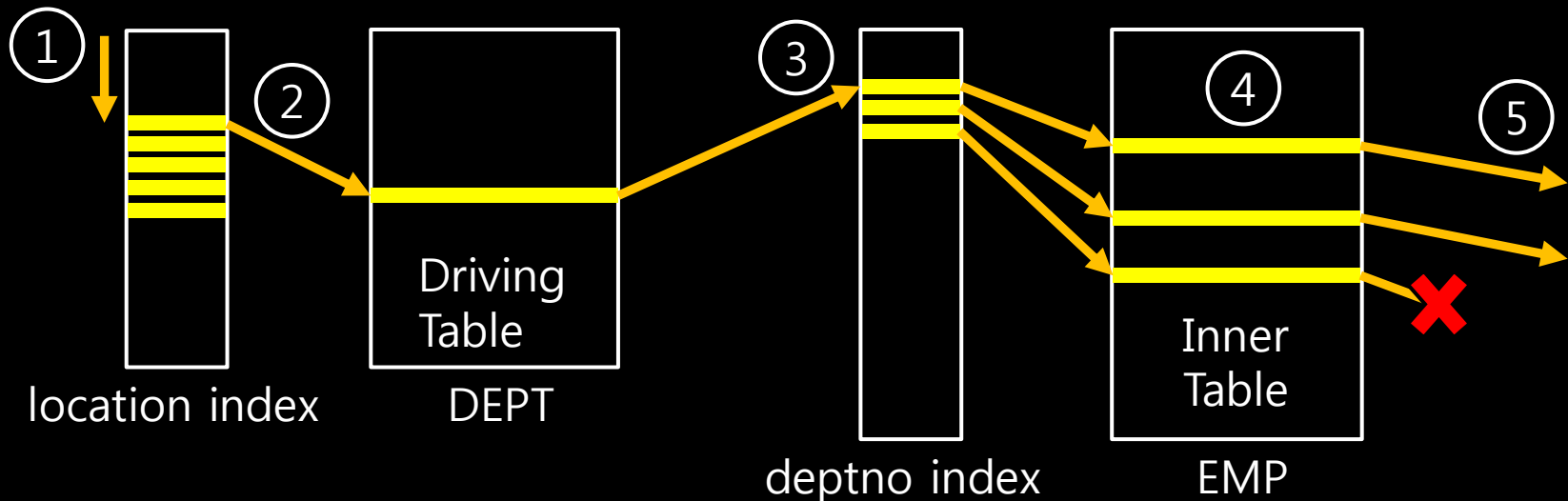
- 용어 이해
 - Driving(Outer) Table
Join 대상 Table 중에서 가장 먼저 Scan 되는 테이블
 - Inner Table
Join 대상 Table 중에서 나중에 Scan 되는 테이블

조인(Join) - Nested-Loop Join

- 수행방식

```
SELECT a.dname, b.ename, b.sal  
FROM DEPT a, EMP b  
WHERE a.location = 'jeju'  
      AND b.sal > 200  
      AND a.deptno = b.deptno
```

DEPT Index : location
EMP Index : deptno

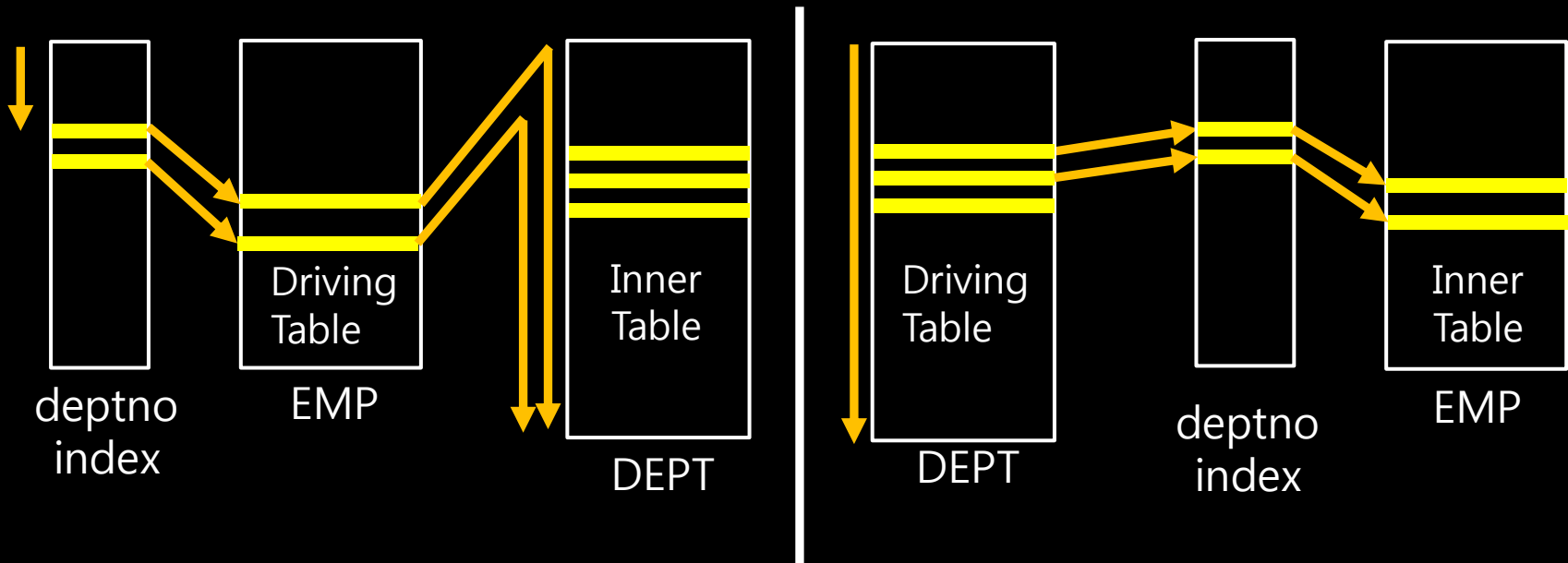


조인(Join) - Nested-Loop Join

- 조인시 DEPT 테이블에 인덱스가 없을 경우?

```
SELECT a.dname, b.ename, b.sal  
FROM DEPT a, EMP b  
WHERE a.location = 'jeju'  
AND b.sal > 200  
AND a.deptno = b.deptno
```

DEPT Index : ~~location~~
EMP Index : deptno



조인(Join) - Nested-Loop Join

- 고려사항

조인(Join) - Hash Join

- Hash Join이란?
 - Hash Function을 이용한 조인 방식
- Hash Join은 Sort Merge Join을 Upgrade한 방식임
- Batch 작업 등과 같이 대용량 데이터 처리시 사용
- 빠른 성능을 보장함

조인(Join) - Hash Join

- 용어 이해

- **Build Table**

- .해시 테이블 생성시 조인에 사용하는 두개의 테이블 중 집합이 작다고 판단되는 테이블.
 - .선행되는 테이블을 말함.

- **Hash Table**

- .Hash Bucket으로 구성된 Hash Map.

- **Hash Bucket**

- .Build Table의 데이터를 해쉬 함수를 통하여 리턴받은 해시값이 같은 데이터끼리 모아 놓은 곳
 - .같은 데이터는 체인(연결 리스트)로 연결됨.

- **Probe Table**

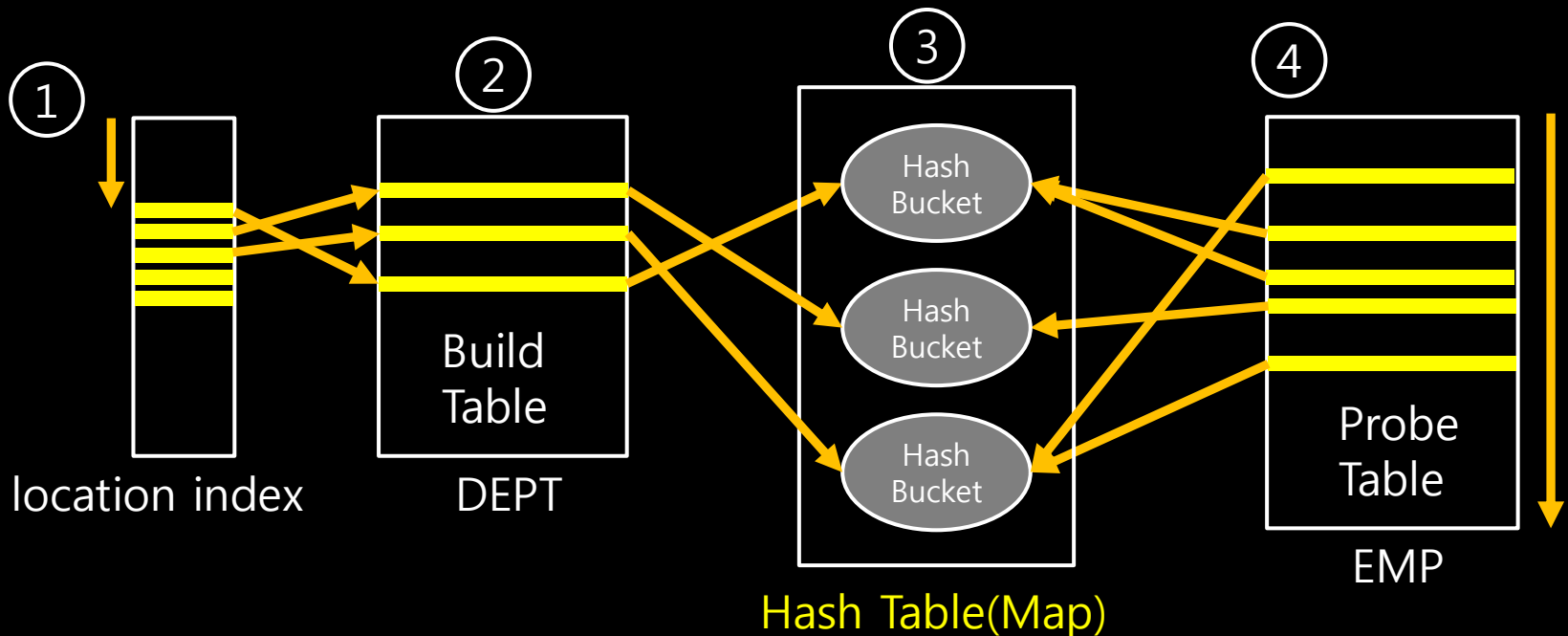
- .나중에 Access하여 Join을 수행하기 위한 후행 집합.
 - .조인시 해쉬 함수를 이용하며, 해쉬 함수에서 리턴받은 값을 이용하여 해쉬 버킷 주소로 찾아가 체인을 스캔하면서 데이터를 검색.

조인(Join) - Hash Join

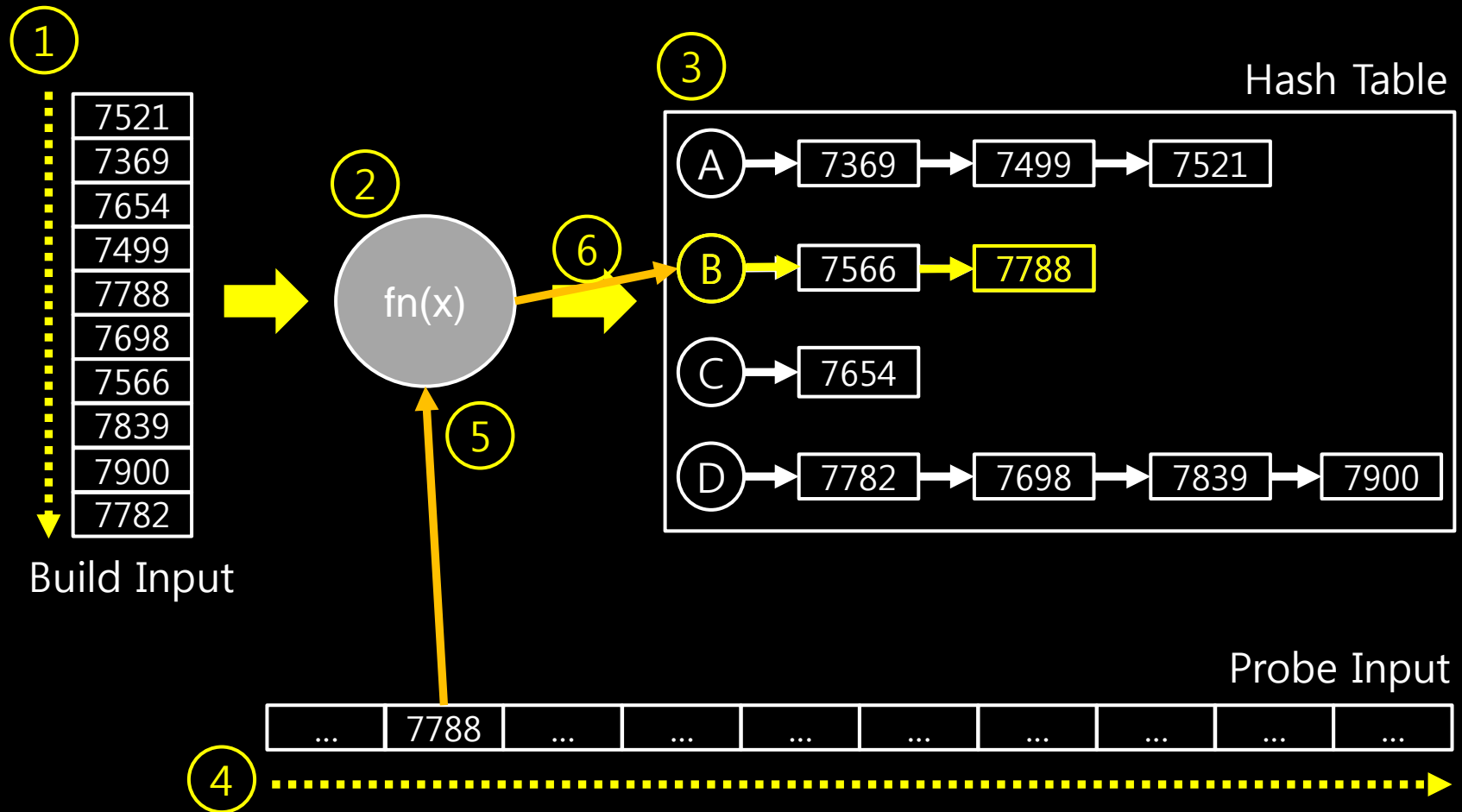
- 수행방식

```
SELECT a.dname, b.ename, b.sal  
FROM DEPT a, EMP b  
WHERE a.location = 'jeju'  
AND b.sal > 200  
AND a.deptno = b.deptno
```

DEPT Index : location



조인(Join) - Hash Join



조인(Join) - Hash Join

- 고려사항

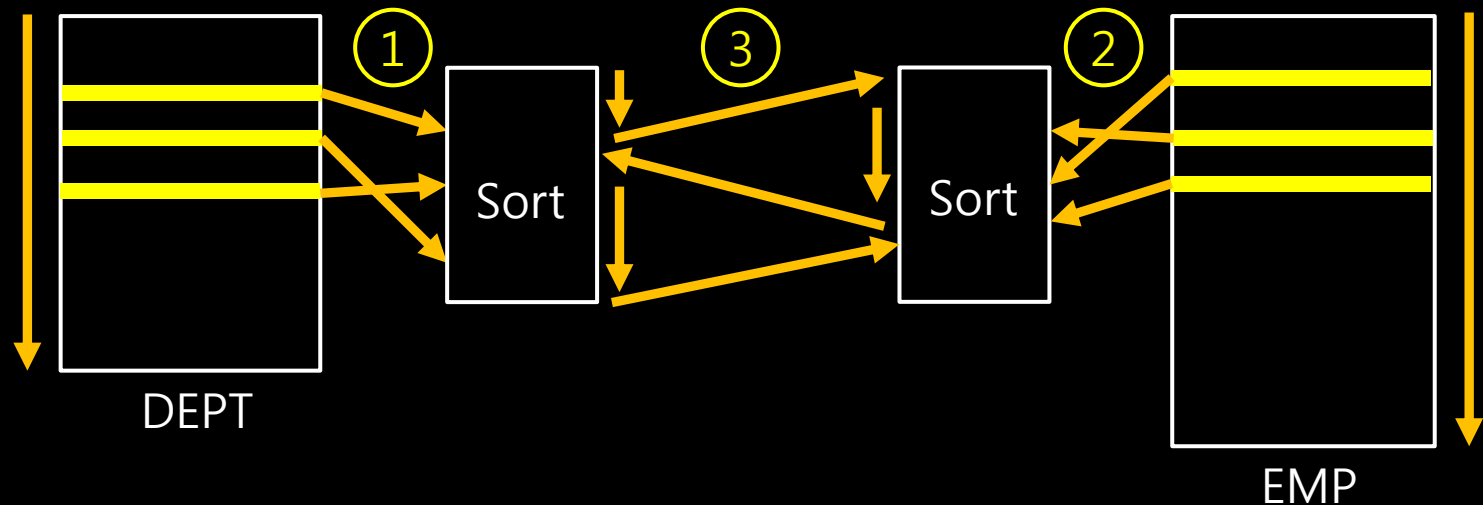
조인(Join) - Sort Merge Join

- Sort Merge Join 이란?
 - 조인 대상 Table을 각각 Sort한 후 조인을 수행
 - Hash Join의 이전 버전
 - 대용량 처리 대상 집합에 유리하지만 Hash 조인이 더 효율 적임
 - SQL 튜닝을 유도하는 경우는 거의 없음.

조인(join) - Sort Merge Join

- 수행방식

```
SELECT a.dname, b.ename, b.sal  
FROM DEPT a, EMP b  
WHERE a.location = 'jeju'  
      AND b.sal > 200  
      AND a.deptno = b.deptno
```



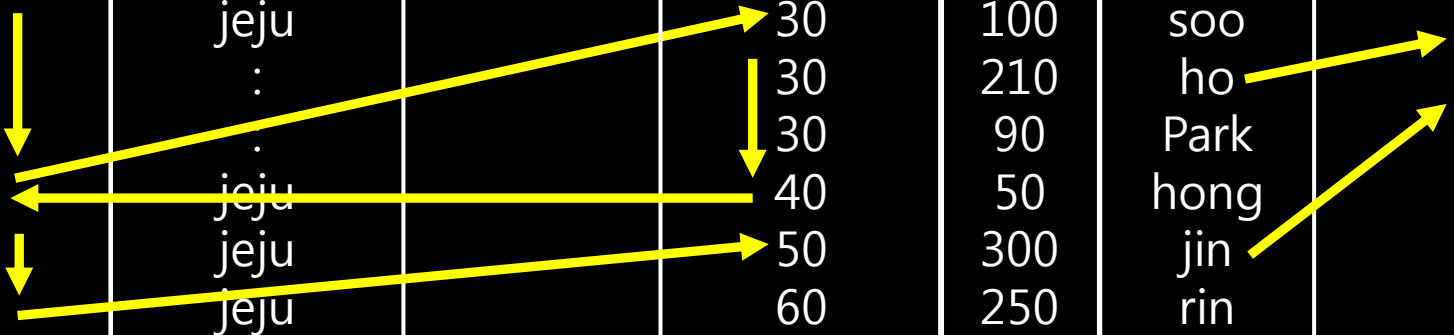
조인(Join) - Sort Merge Join

DEPTNO	LOCATION
:	:
20	jeju
:	:
:	:
40	jeju
50	jeju
50	jeju
:	:

DEPT Table의 Sort된 결과

DEPTNO	SAL	ENAME
:	:	:
30	100	soo
30	210	ho
30	90	Park
40	50	hong
50	300	jin
60	250	rin
:	:	:

EMP Table의 Sort된 결과



조인(Join) - Sort Merge Join

- 고려사항

요약

- DB구조는?
 - => 메모리, 프로세스, Disk
- SQL 처리 순서는?
 - => FROM -> WHERE -> GROUP BY -> HAVING
 - > SELECT -> ORDER BY -> LIMIT
- 옵티마이저(Optimizer)란?
 - => SQL을 가장 빠르고 효율적으로 수행할 수 있게 도와줌
- 힌트(Hint)란?
 - => 더 좋은 실행계획으로 유도하는 방법
- 인덱스(Index)의 종류?
 - => B*tree, Bitmap, Reverse Key,, Function Based, IOT
- 조인(Join)의 종류?
 - => Nested-Loop, Hash-Join, Sort-Merge-Join

목차

1. 튜닝을 위한 기본지식 쌓기

2. SQL 분석 및 성능관련 해석

- 실행 계획 개념

- 실행 계획 생성 및 해석
- 실행 계획 종류

3. 실사례를 통한 SQL 성능향상 살펴보기

4. MySQL과 Oracle 개발 차이점 및 유의사항

실행 계획이란?

- SQL을 어떻게, 어떠한 방법으로 수행할지 계획을 보여주는 것
- 옵티마이저에 의해 만들어짐.
- 실행 계획에 따라 성능을 좌우함.
- 잘 못된 실행 계획 생성시 힌트를 이용하여 변경할 수 있음.

**이번 수업에는
Oracle 과 MySQL에 대해서
알아 보도록 하겠습니다.**

목차

1. 튜닝을 위한 기본지식 쌓기

2. SQL 분석 및 성능관련 해석

- 실행 계획 개념
- 실행 계획 생성 및 해석
- 실행 계획 종류

3. 실사례를 통한 SQL 성능향상 살펴보기

4. MySQL과 Oracle 개발 차이점 및 유의사항

Oracle

실행 계획 생성 및 해석 - Oracle

- Oracle 생성 방법

```
SQL> EXPLAIN PLAN FOR  
      Select ...
```

```
SQL> EXPLAIN PLAN FOR select * From emp where empno = 12314;  
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	

0	SELECT STATEMENT		1	146	3 (0)	00:00:01	
1	TABLE ACCESS BY INDEX ROWID	EMP	1	146	3 (0)	00:00:01	
*	INDEX UNIQUE SCAN	EMP_PK	1		2 (0)	00:00:01	

Predicate Information (identified by operation id):

2 - access("EMPNO"=12314)

실행 계획 생성 및 해석 - Oracle

- Oracle 해석 방법

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		500K	19M	1258 (2)	00:00:16
1	NESTED LOOPS		500K	19M	1258 (2)	00:00:16
2	TABLE ACCESS BY INDEX ROWID	DEPT	1	13	1 (0)	00:00:01
* 3	INDEX UNIQUE SCAN	PK_DEPT	1		0 (0)	00:00:01
* 4	TABLE ACCESS FULL	EMP	500K	13M	1257 (2)	00:00:16

↓ Level 1
↓ Level 2
↓ Level 3
↓ Level 4

Level이 가장 낮은 실행 계획을 먼저 해석

Level이 동일하면 위치상 위에있는 실행 계획을 먼저 해석

조인 방식이 있을 경우 조인 방식에 의한 해석

실행 계획 생성 및 해석 - Oracle

- Oracle 해석 방법

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		500K	19M	1258 (2)	00:00:16
1	NESTED LOOPS		500K	19M	1258 (2)	00:00:16
2	TABLE ACCESS BY INDEX ROWID	DEPT	1	13	1 (0)	00:00:01
* 3	INDEX UNIQUE SCAN	PK_DEPT	1		0 (0)	00:00:01
* 4	TABLE ACCESS FULL	EMP	500K	13M	1257 (2)	00:00:16

Rows : 예측 Row 수 (통계기반 정보)

Bytes : Result Set 크기(byte)

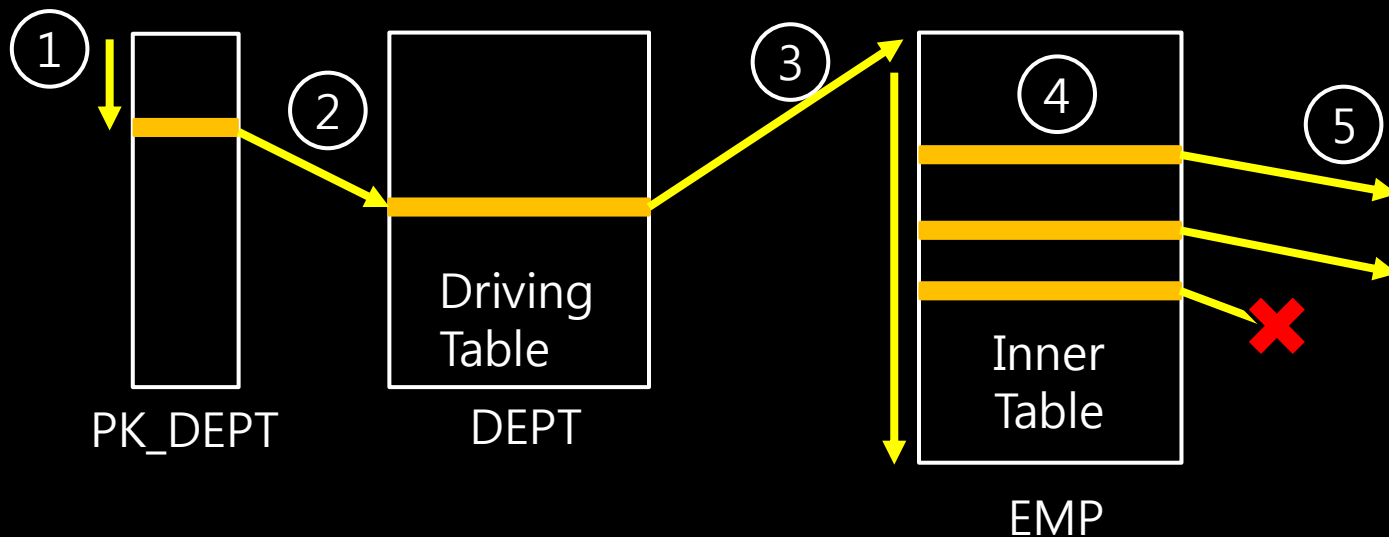
Cost : 통계정보기반 단계별 예상 비용

Time : 예상시간

실행 계획 생성 및 해석 - Oracle

- Oracle 해석 방법

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		500K	19M	1258 (2)	00:00:16
1	NESTED LOOPS		500K	19M	1258 (2)	00:00:16
2	TABLE ACCESS BY INDEX ROWID	DEPT	1	13	1 (0)	00:00:01
* 3	INDEX UNIQUE SCAN	PK_DEPT	1		0 (0)	00:00:01
* 4	TABLE ACCESS FULL	EMP	500K	13M	1257 (2)	00:00:16



실행 계획 생성 및 해석 - Oracle

- Oracle 해석 방법
 - ACCESS 조건 : 인덱스를 제대로 읽었다.

EMP table
- Primary key (empno)

```
SELECT *  
  FROM emp  
 WHERE empno < 10
```

empno	ename	...
1	TAM	...
2	BOB1	...
3	ADAM	...
4	BOB2	...
...
10

10건
PK Index
block access

Id	Operation	Name
0	SELECT STATEMENT	
* 1	TABLE ACCESS BY INDEX ROWID	EMP
* 2	INDEX RANGE SCAN	EMP_PK

Predicate Information

2 - access("EMPNO"<10)

10보다 작은
EMPNO
인덱스 ACCESS

실행 계획 생성 및 해석 - Oracle

- Oracle 해석 방법
 - FILTER 조건 : DATA를 읽은 후 원하지 않는 것을 제거(필터)

EMP table
- Primary key (empno)

```
SELECT *  
  FROM emp  
 WHERE empno < 10  
 AND ename like '%BOB%';
```

empno	ename	...
1	TAM	...
2	BOB1	...
3	ADA	...
4	BOB2	...
...
10	SAM	...
...

인덱스에서(access)
10건 access후
테이블에서
8건 제거 -> 2건 도출

Id	Operation	Name
0	SELECT STATEMENT	
* 1	TABLE ACCESS BY INDEX ROWID	EMP
* 2	INDEX RANGE SCAN	EMP_PK

Predicate Information

1 - filter("ENAME " like '%bob%')
2 - access("EMPNO"<10)

10건 중
BOB 포함
데이터 필터

실행 계획 생성 및 해석 - Oracle

- Oracle 해석 방법

```
EXPLAIN PLAN FOR
SELECT a.*, b.dname
  FROM emp a
       ,dept b
 WHERE a.empno < 10
       AND b.dname= 'OPERATIONS'
       AND a.deptno=b.deptno;

SELECT * FROM TABLE(dbms_xplan.display());
```

EMP table
- Primary key (empno)

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		5	205	6 (0)	00:00:01
* 1	TABLE ACCESS BY INDEX ROWID	EMP	5	140	3 (0)	00:00:01
2	NESTED LOOPS		5	205	6 (0)	00:00:01
* 3	TABLE ACCESS FULL	DEPT	1	13	3 (0)	00:00:01
* 4	INDEX RANGE SCAN	EMP_PK	9		2 (0)	00:00:01

Predicate Information (identified by operation id):

```
1 - filter("A"."DEPTNO"="B"."DEPTNO")
3 - filter("B"."DNAME"='OPERATIONS')
4 - access("A"."EMPNO"<10)
```

MySQL

실행 계획 생성 및 해석 - MySQL

- MySQL 생성 방법

```
SQL> EXPLAIN  
      Select ...
```

```
SQL> EXPLAIN select * From articlenews;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	articlenews	ALL	NULL	NULL	NULL	NULL	1	

실행 계획 생성 및 해석 - MySQL

- MySQL 해석 방법

```
EXPLAIN
SELECT e.*, ec.content, c.label categoryLabel
FROM (SELECT *
      FROM tt_Entries USE KEY (owpu)
      WHERE OWNER = 977
      ORDER BY published DESC, id DESC
      LIMIT 10) e
LEFT OUTER JOIN tt_Categories c      ON e.category = c.id
LEFT OUTER JOIN tt_EntryContents ec ON e.owner = ec.owner AND e.id = ec.id
```

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
①	1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	10	
③	1	PRIMARY	c	eq_ref	PRIMARY	PRIMARY	4	e.category	1	
④	1	PRIMARY	ec	eq_ref	PRIMARY	PRIMARY	8	e.owner,e.id	1	
②	2	DERIVED	tt_Entries	ref	owpu	owpu	4		17356	Using where

위쪽에서 아래쪽으로, derived로 분기!!!

실행 계획 생성 및 해석 - MySQL

- MySQL 해석 방법

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
----	-------------	-------	------	---------------	-----	---------	-----	------	-------

id : Select 쿼리별로 부여되는 식별자

select_type : Select 쿼리가 어떤 타입인지 표시

table : 테이블 이름

type : 각 테이블의 Row를 어떠한 방식으로 읽었는지를 표시함

possible_keys : 테이블을 조인하기 위해 사용할 수 Index 정보

key : 조인할 때 실제로 사용하고 있는 Index 정보

NULL 은 키를 사용하고 있지 않다는 뜻

key_len : 사용된 키의 길이(컬럼수)를 표시

ref : row를 선택시 사용한 키(컬럼) 또는 상수값을 선택할 때 사용한 키 정보

rows : 조인을 실행하기 위해 테이블마다 읽어야 하는 row의 수

(통계 정보를 기준으로 표시함)

extra : 어떻게 조인을 실행하는가에 대한 정보

실행 계획 생성 및 해석 - MySQL

- MySQL 해석 방법

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	10	
1	PRIMARY	c	eq_ref	PRIMARY	PRIMARY	4	e.category	1	
1	PRIMARY	ec	eq_ref	PRIMARY	PRIMARY	8	e.owner,e.id	1	
2	DERIVED	tt_Entries	ref	owpu	owpu	4		17356	Using where

- select_type : Select 쿼리가 어떤 타입인지 표시
 - PRIMARY : 가장 바깥쪽(outer)에 있는 단위 쿼리를 표시함
 - SIMPLE : Union or Subquery를 사용하지 않는 단순한 SELECT 표시
 - DERIVED : FROM 절에서 인라인 뷰 사용시 표시됨
서브쿼리 사용시에도 표시됨.

실행 계획 생성 및 해석 - MySQL

- MySQL 해석 방법

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	10	
1	PRIMARY	c	eq_ref	PRIMARY	PRIMARY	4	e.category	1	
1	PRIMARY	ec	eq_ref	PRIMARY	PRIMARY	8	e.owner,e.id	1	
2	DERIVED	tt_Entries	ref	owpu	owpu	4		17356	Using where

- type : 각 테이블의 Row를 어떠한 방식으로 읽었는지를 확인함(엑세스 타입)
 - ALL : 테이블 전체를 읽을때
 - const : Primary key 또는 Unique key를 이용하여 반드시 1건을 반환하는 경우 표시
 - Unique Index scan 이라고 함
 - eq_ref : 조인시 Primary key 또는 Unique key를 이용하여 데이터를 하나만 가져올 경우 표시됨
 - ref : 인덱스의 종류와 관계없이 =(equal) 조건으로 데이터 추출
복합 인덱스중 앞부분의 일부 컬럼만 사용시 표시됨
 - Index : 인덱스 전체를 읽을때
 - range : 인덱스를 사용하여 특정 범위 내의 데이터만 읽을때

실행 계획 생성 및 해석 - MySQL

- MySQL 해석 방법

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	10	
1	PRIMARY	c	eq_ref	PRIMARY	PRIMARY	4	e.category	1	
1	PRIMARY	ec	eq_ref	PRIMARY	PRIMARY	8	e.owner,e.id	1	
2	DERIVED	tt_Entries	ref	owpu	owpu	4		17356	Using where

- key
 - 조인할 때 실제로 사용하고 있는 Index 정보
 - optimizer가 가장 적은 비용으로 query를 실행하기 위해 선택한 인덱스임

실행 계획 생성 및 해석 - MySQL

- MySQL 해석 방법

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	10	
1	PRIMARY	c	eq_ref	PRIMARY	PRIMARY	4	e.category	1	
1	PRIMARY	ec	eq_ref	PRIMARY	PRIMARY	8	e.owner,e.id	1	
2	DERIVED	tt_Entries	ref	owpu	owpu	4		17356	Using where

- Key_len

- 사용된 키의 길이(컬럼수)를 표시
- optimizer가 가장 적은 비용으로 query를 실행하기 위해 선택한 인덱스임



실행 계획 생성 및 해석 - MySQL

- MySQL 해석 방법

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	10	
1	PRIMARY	c	eq_ref	PRIMARY	PRIMARY	4	e.category	1	
1	PRIMARY	ec	eq_ref	PRIMARY	PRIMARY	8	e.owner,e.id	1	
2	DERIVED	tt_Entries	ref	owpu	owpu	4		17356	Using where

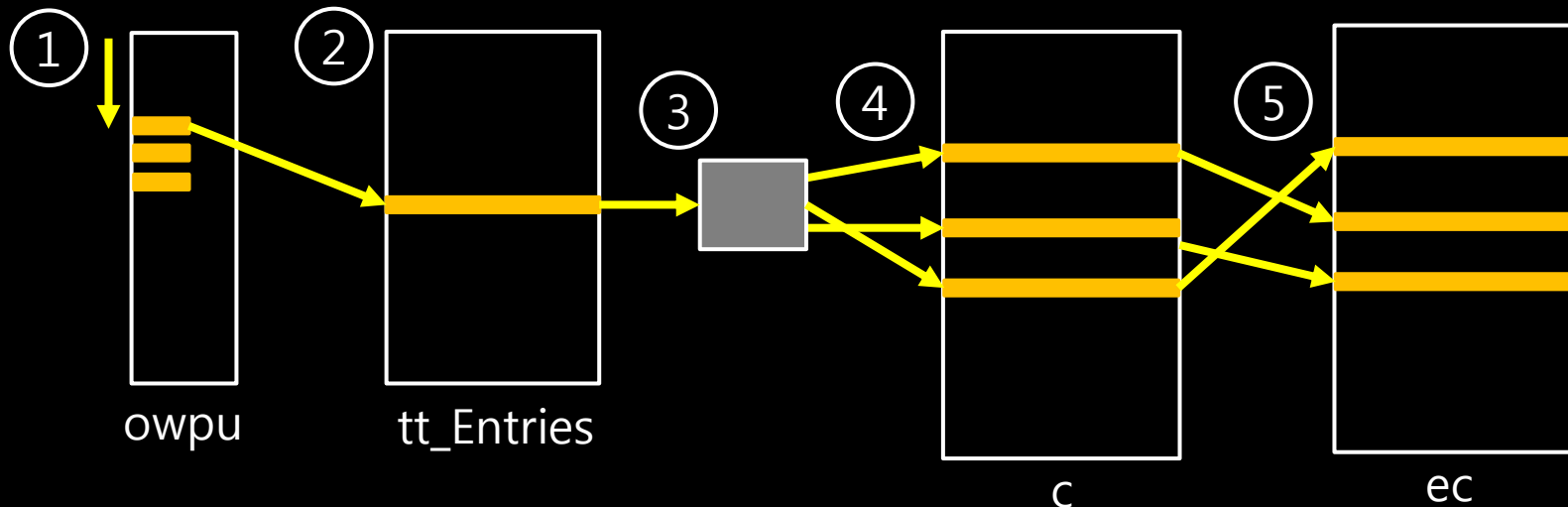
- Extra : 어떻게 조인을 실행하는가에 대한 정보

- Using index : 테이블 액세스 없이 인덱스만 액세스
- Using where : 테이블에서 row를 추출할 다음 조건 절에 의해 filtering
- Using temporary : Query 수행 중 중간 결과를 저장하기 위해서 temporary table 생성
group by, order by 같은 정렬 작업이 있을 경우 발생
- Using filesort : 정렬되지 않는 결과에 대해 external sort 작업을 할 경우 발생
정렬 작업은 메모리 또는 디스크에서 발생(확인 할 수 없음)

실행 계획 생성 및 해석 - MySQL

- MySQL 해석 방법

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
①1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	10	
③1	PRIMARY	c	eq_ref	PRIMARY	PRIMARY	4	e.category	1	
④1	PRIMARY	ec	eq_ref	PRIMARY	PRIMARY	8	e.owner,e.id	1	
②2	DERIVED	tt_Entries	ref	owpu	owpu	4		17356	Using where



목차

1. 튜닝을 위한 기본지식 쌓기

2. SQL 분석 및 성능관련 해석

- 실행 계획 개념
- 실행 계획 생성 및 해석
- 실행 계획 종류

3. 실사례를 통한 SQL 성능향상 살펴보기

4. MySQL과 Oracle 개발 차이점 및 유의사항

실행 계획 종류

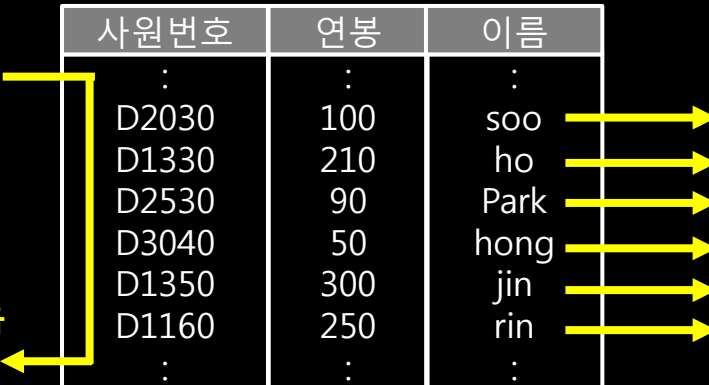
- 테이블 관련 실행계획
 - Full Table Scan
- 인덱스 관련 실행계획
 - Index Unique Scan
 - Index Range Scan
 - Index Full Scan

실행 계획 종류 - 테이블

- Full Table Scan
 - 테이블을 처음부터 끝까지 읽음

```
SELECT 사원번호, 이름  
FROM EMP
```

테이블을 처음부터 끝까지 읽음



사원번호	연봉	이름
:	:	:
D2030	100	soo
D1330	210	ho
D2530	90	Park
D3040	50	hong
D1350	300	jin
D1160	250	rin
:	:	:

<EMP Table>

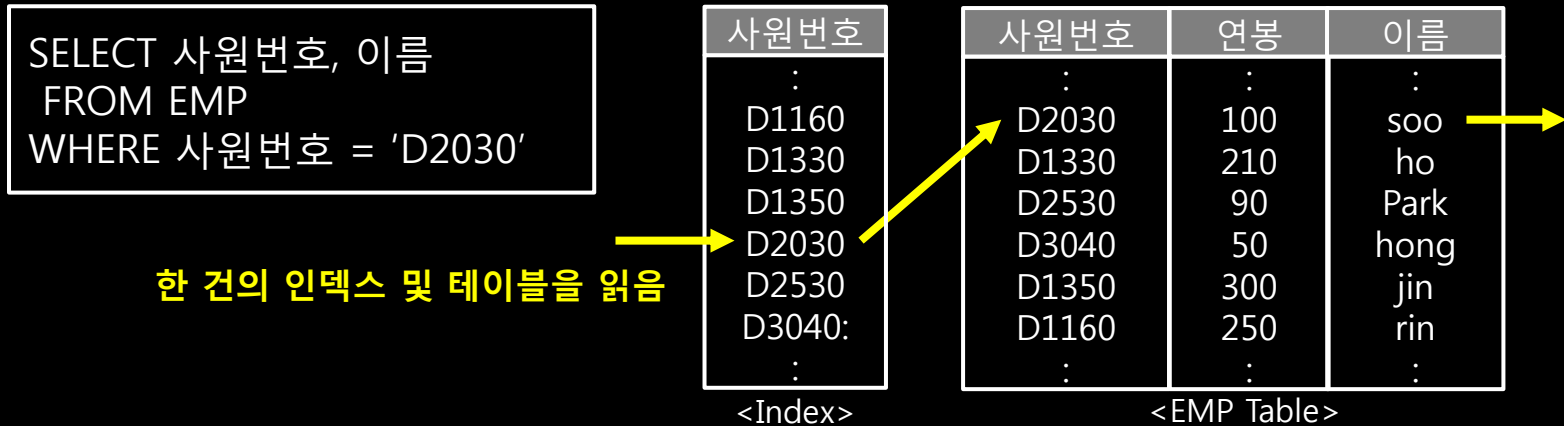
```
-- Oracle : Multi Block read  
SELECT STATEMENT  
  TABLE ACCESS (FULL) OF 'EMP' (TABLE)
```

```
-- MySQL : Innodb plugin 부터
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	EMP	ALL	NULL	NULL	NULL	NULL	1	

실행 계획 종류 - 인덱스

- Index Unique Scan
 - Unique Index로 된 컬럼을 Equal(=)로 조회하는 경우



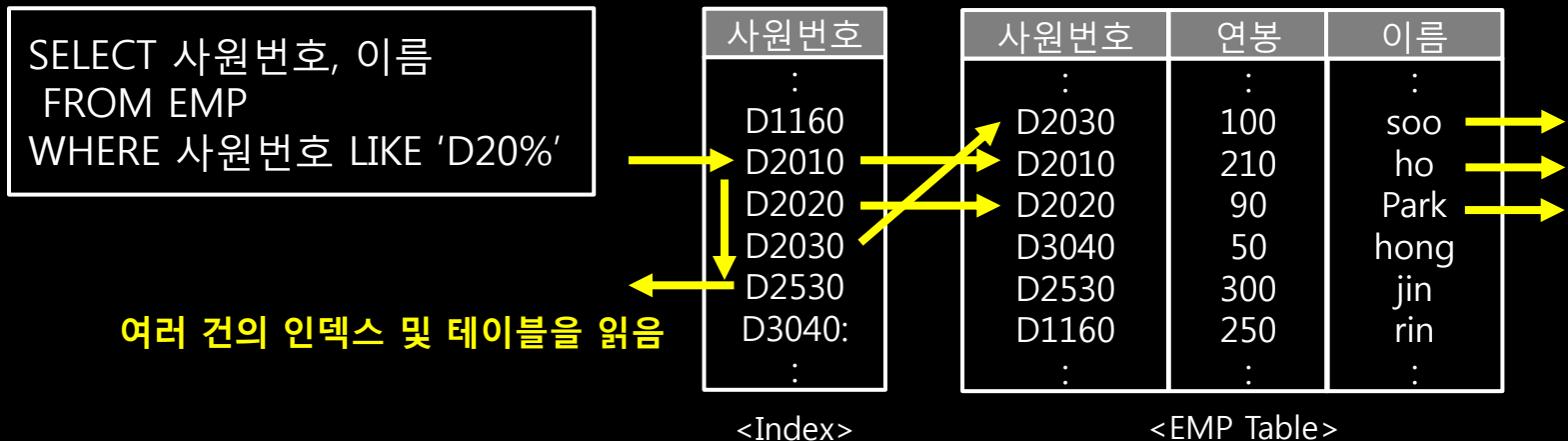
```
-- Oracle
SELECT STATEMENT
  TABLE ACCESS (BY INDEX ROWID) OF 'EMP' (TABLE)
    INDEX (UNIQUE SCAN) OF 'IDX_UNIQUE' (INDEX (UNIQUE))
```

```
-- MySQL
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	EMP	const	idx_unique	idx_unique	32	const	1	

실행 계획 종류 - 인덱스

- Index Range Scan
 - Where 조건에 Like, Between, <, > 등을 사용할 경우 발생



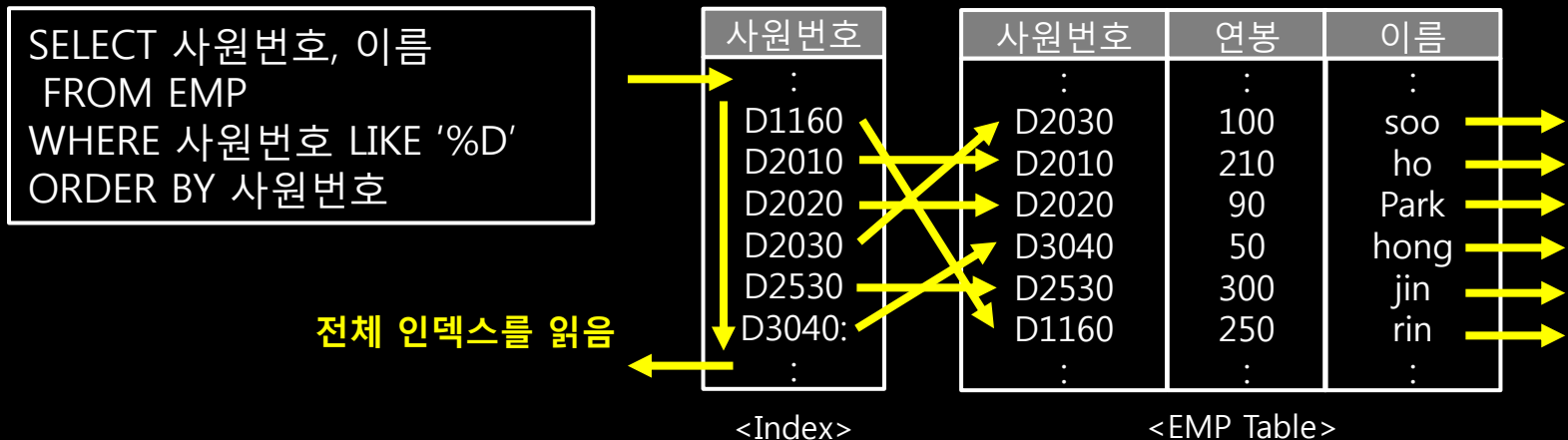
```
-- Oracle  
SELECT STATEMENT  
TABLE ACCESS (BY INDEX ROWID) OF 'EMP' (TABLE)  
INDEX (RANGE SCAN) OF 'IDX_UNIQUE' (INDEX (UNIQUE))
```

```
-- MySQL
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	EMP	range	idx_unique	idx_unique	32	NULL	1	Using where

실행 계획 종류 - 인덱스

- Index Full Scan
 - 인덱스의 첫번째 컬럼으로 정렬된 데이터 추출



```
-- Oracle
SELECT STATEMENT
  TABLE ACCESS (BY INDEX ROWID) OF 'EMP' (TABLE)
    INDEX (FULL SCAN) OF 'IDX_UNIQUE' (INDEX (UNIQUE))
```

-- MySQL

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	EMP	index	NULL	idx_unique	33	NULL	6	Using where

목차

1. 튜닝을 위한 기본지식 쌓기
2. SQL 분석 및 성능관련 해석

3.실사례를 통한 SQL 성능향상 살펴보기

4. MySQL과 Oracle 개발 차이점 및 유의사항

SQL 성능향상

- 어떻게 하면 SQL 성능 향상을 시킬 수 있을까?
 - 최적의 실행 계획
 - 스캔 범위를 좁혀라
 - 원하는 데이터만 읽자 (필요한 컬럼만)
 - ?????

SQL 성능향상 - 사례1

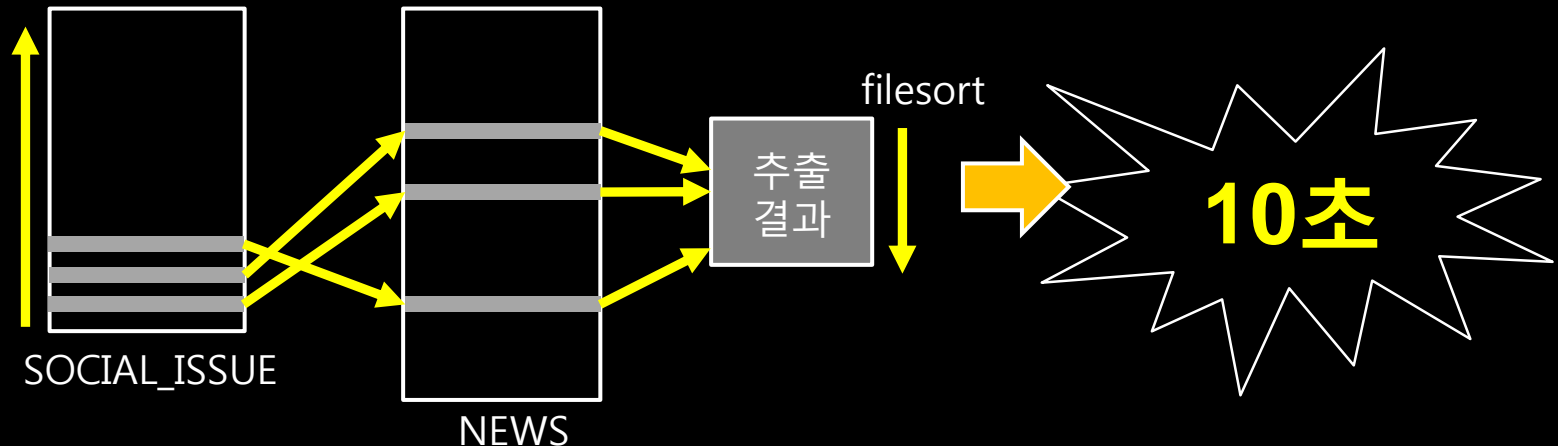
- 사례1 - 변경전

SOCIAL_ISSUE table
Primary key (news_id)

NEWS table
Primary key (id)

```
SELECT sif.*, n.*  
  FROM SOCIAL_ISSUE sif  
        ,NEWS n  
 WHERE n.id = sif.news_id  
 ORDER BY n.id desc  
 LIMIT 0, 10;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	sif	ALL	PRIMARY	NULL	NULL	NULL	1945	Using temporary; Using filesort
1	SIMPLE	n	eq_ref	PRIMARY	PRIMARY	4	bloggernews.sif.news_id	1	using where



SQL 성능향상 - 사례1

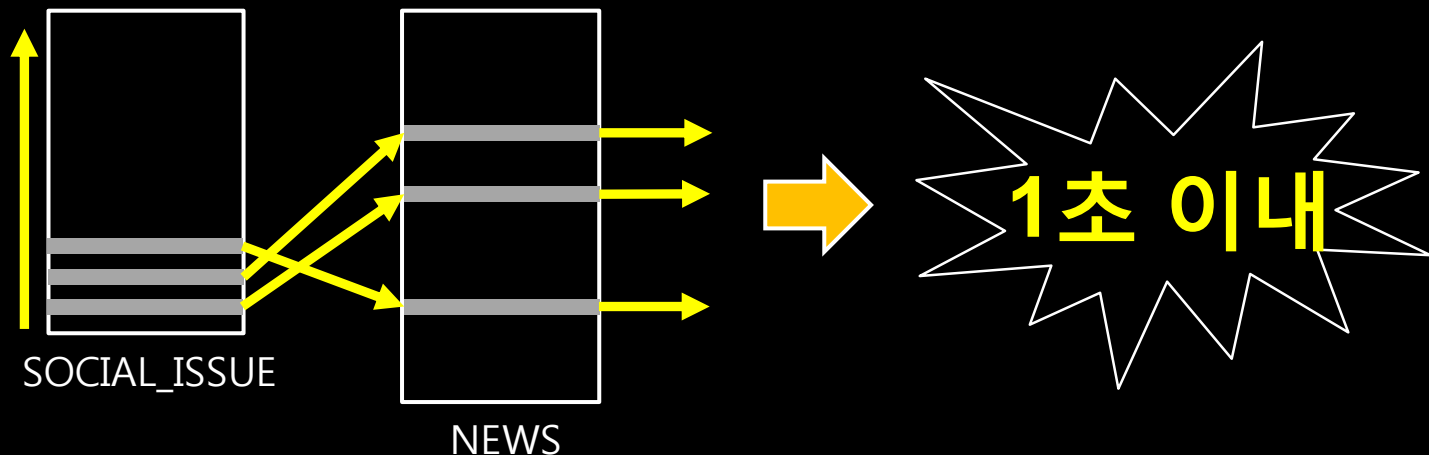
- 사례1 - 변경후

SOCIAL_ISSUE table
Primary key (news_id)

NEWS table
Primary key (id)

```
SELECT sif.*, n.*  
  FROM SOCIAL_ISSUE sif  
        ,NEWS n  
 WHERE n.id = sif.news_id  
    ORDER BY sif.news_id desc  
    LIMIT 0, 10;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	sif	index	PRIMARY	PRIMARY	4	NULL	10	
1	SIMPLE	n	eq_ref	PRIMARY	PRIMARY	4	bloggernews.sif.news_id	1	Using where



SQL 성능향상 - 사례2

- 사례2 - 변경전

CONTEST table
Primary key (CONT_ID)

CONTEST_RANK table
Primary key (ID)

CONTEST_RANK table
IDX_PAGE1 (ID + USERID)



2초

```
SELECT *
FROM
( SELECT
    /*+ ORDERED USE_NL(r, c)
      INDEX_DESC(r IDX_PAGE1) */
    c.userid,
    c.content AS,
    MIN(s.id) OVER() AS 글 최소ID,
    MAX(s.id) OVER() AS 글 최대ID,
    COUNT(*) OVER() AS 가져온 글 개수,
    ROWNUM AS rnum
  FROM   CONTEST_RANK r,
         CONTEST c,
  WHERE  1=1
        AND r.userid = 100
        AND r.id <= 350000
        AND r.cont_id = c.cont_id
        AND rownum <= 51
    )
WHERE  rnum BETWEEN 1 and 5
```


SQL 성능향상 - 사례2

- 사례2 - 변경전

Rows	Row Source Operation
------	----------------------

0	STATEMENT
5	VIEW
51	WINDOW BUFFER
51	COUNT STOPKEY
51	NESTED LOOPS
51	TABLE ACCESS BY INDEX ROWID CONTEST_RANK
*350000	INDEX RANGE SCAN DESCENDING IDX_PAGING1
51	TABLE ACCESS BY INDEX ROWID CONTEST
51	INDEX UNIQUE SCAN CONTEST_PK

Predicate Information (identified by operation id):

7 - access(R.USERID=100 AND R.ID<=350000)
filter(R.USERID=100) ---> 인덱스 필터

ID	USERID	
350000	100	filter
349999	90	
349998	50	
:	:	
:	:	
3	100	
2	100	
1	100	

IDX_PAGE1 index

350000 인덱스

ACCESS

SQL 성능향상 - 사례2

- 사례2 - 변경후

CONTEST table
Primary key (CONT_ID)

CONTEST_RANK table
Primary key (ID)

CONTEST_RANK table
IDX_PAGE1 (ID + USERID)



IDX_PAGE1 (USERID + ID)

0.01초

```
SELECT *
FROM
( SELECT
    /*+ ORDERED USE_NL(r, c)
      INDEX_DESC(r IDX_PAGE1) */
    c.userid,
    c.content AS,
    MIN(s.id) OVER() AS 글 최소ID,
    MAX(s.id) OVER() AS 글 최대ID,
    COUNT(*) OVER() AS 가져온 글 개수,
    ROWNUM AS rnum
  FROM   CONTEST_RANK r,
         CONTEST c,
  WHERE  1=1
        AND r.userid = 100
        AND r.id <= 350000
        AND r.cont_id = c.cont_id
        AND rownum <= 51
    )
WHERE  rnum BETWEEN 1 and 5
```

SQL 성능향상 - 사례2

- 사례2 - 변경후

Rows Row Source Operation

```
-----
 0 STATEMENT
 5 VIEW
51 WINDOW BUFFER
51 COUNT STOPKEY
51 NESTED LOOPS
51 TABLE ACCESS BY INDEX ROWID CONTEST RANK
* 51 INDEX RANGE SCAN DESCENDING IDX_PAGING1
51 TABLE ACCESS BY INDEX ROWID CONTEST
51 INDEX UNIQUE SCAN CONTEST_PK
```

Predicate Information

```
-----
 7 - access(R.USERID=100 AND R.ID <=350000)
```

USERID	ID
100	350000
100	3
100	2
⋮	⋮
⋮	⋮
99	1
90	349999
50	349998

filter

51건 인덱스

ACCESS

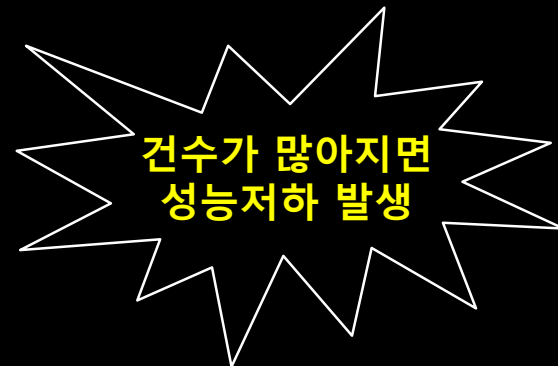
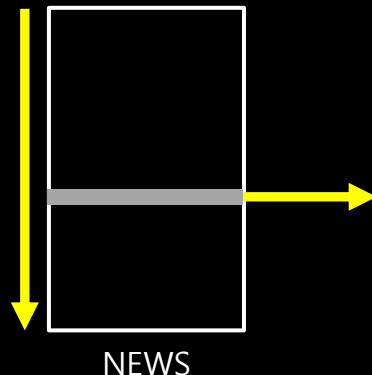
SQL 성능향상 - 사례3

- 사례3 - 변경전

```
CREATE TABLE NEWS (  
  news_no varchar(11) NOT NULL  
  ,title varchar(64)      DEFAULT NULL  
  ,reg_date datetime     DEFAULT NULL  
  ,status enum('normal','hidden')  
        NOT NULL DEFAULT 'normal'  
  ,PRIMARY KEY (`news_no`));
```

```
SELECT n.*  
FROM NEWS n  
WHERE n.news_no = 10030165  
      AND n.status IN ('normal', 'hidden')
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	n	ALL	PRIMARY	NULL	NULL	NULL	19674	Using where



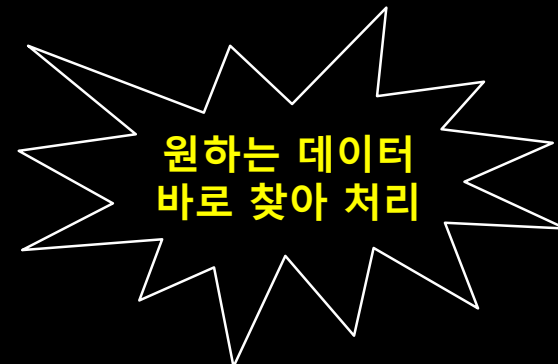
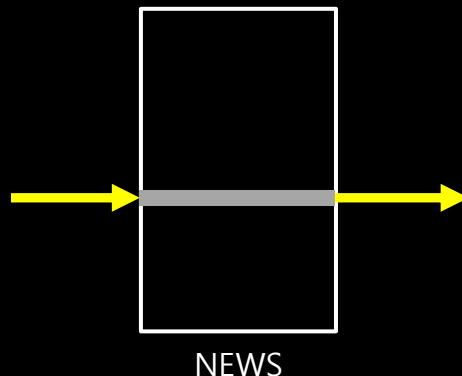
SQL 성능향상 - 사례3

- 사례3 - 변경후

```
CREATE TABLE NEWS (  
  news_no varchar(11) NOT NULL  
  ,title varchar(64)      DEFAULT NULL  
  ,reg_date datetime     DEFAULT NULL  
  ,status enum('normal','hidden')  
                        NOT NULL DEFAULT 'normal'  
  ,PRIMARY KEY (`news_no`));
```

```
SELECT n.*  
FROM NEWS n  
WHERE n.news_no = '10030165'  
      AND n.status IN ('normal', 'hidden')
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	n	const	PRIMARY	PRIMARY	35	const	1	



SQL 성능향상 - 사례3

- 사례3 - 변경후

- 숫자 컬럼과 문자열 컬럼 비교시 **문자열 컬럼은 숫자 타입으로 변경됨**
- 문자열 컬럼과 비교할때는 문자 타입
- 숫자 컬럼과 비교시에는 숫자 타입

SELECT * FROM 사원 WHERE 사원번호 = 1010



인덱스 정상적으로 이용

SELECT * FROM 사원 WHERE 사원이름 = '수지'



인덱스 정상적으로 이용

SELECT * FROM 사원 WHERE 사원번호 = '1010'



인덱스 정상적으로 이용

SELECT * FROM 사원 WHERE 사원번호 = 1010



인덱스 사용 못함

목차

1. 튜닝을 위한 기본지식 쌓기
2. SQL 분석 및 성능관련 해석
3. 실사례를 통한 SQL 성능향상 살펴보기

4.MySQL과 Oracle 개발 차이점 및 주의사항

차이점 및 주의사항

- MySQL은 **인덱스 자체 FILTER 기능이 없다**
- MySQL은 **SubQuery는 느리다.**
- Distinct VS Group by

차이점 및 주의사항

- MySQL은 **인덱스 자체 FILTER 기능이 없다**

TEST01

- PK (ID)
- IDX(COL1, COL2)

```
SELECT *  
FROM TEST01  
WHERE col1 between 1 and 3  
AND col2 like '%im%'
```

Storage Engine

B-Tree index

col1	col2	id
1	foo	row3
2	kim	row4
3	lim	row1
4	lee	row2
5:	ks	row5

Table Data

id	col1	col2	...
row1	3	foo	...
row2	4	kim	...
row3	1	lim	...
row4	2	lee	...
row5	5:	ks	...

col1 between 1 and 10



DB Engine

col2 like '%im%'

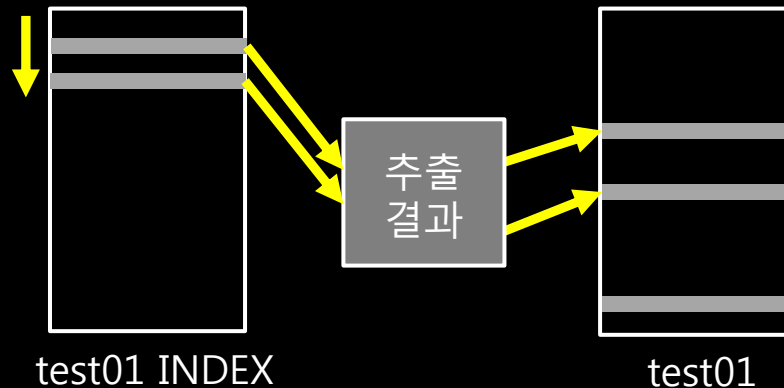
차이점 및 주의사항

- MySQL은 인덱스 자체 FILTER 기능이 없다 -> 어떻게 해결하지?
 - 인덱스로만 구성된 컬럼에서만 스캔

```
SELECT *  
FROM TEST01  
WHERE col1 between 1 and 3  
AND col2 like '%im%'
```



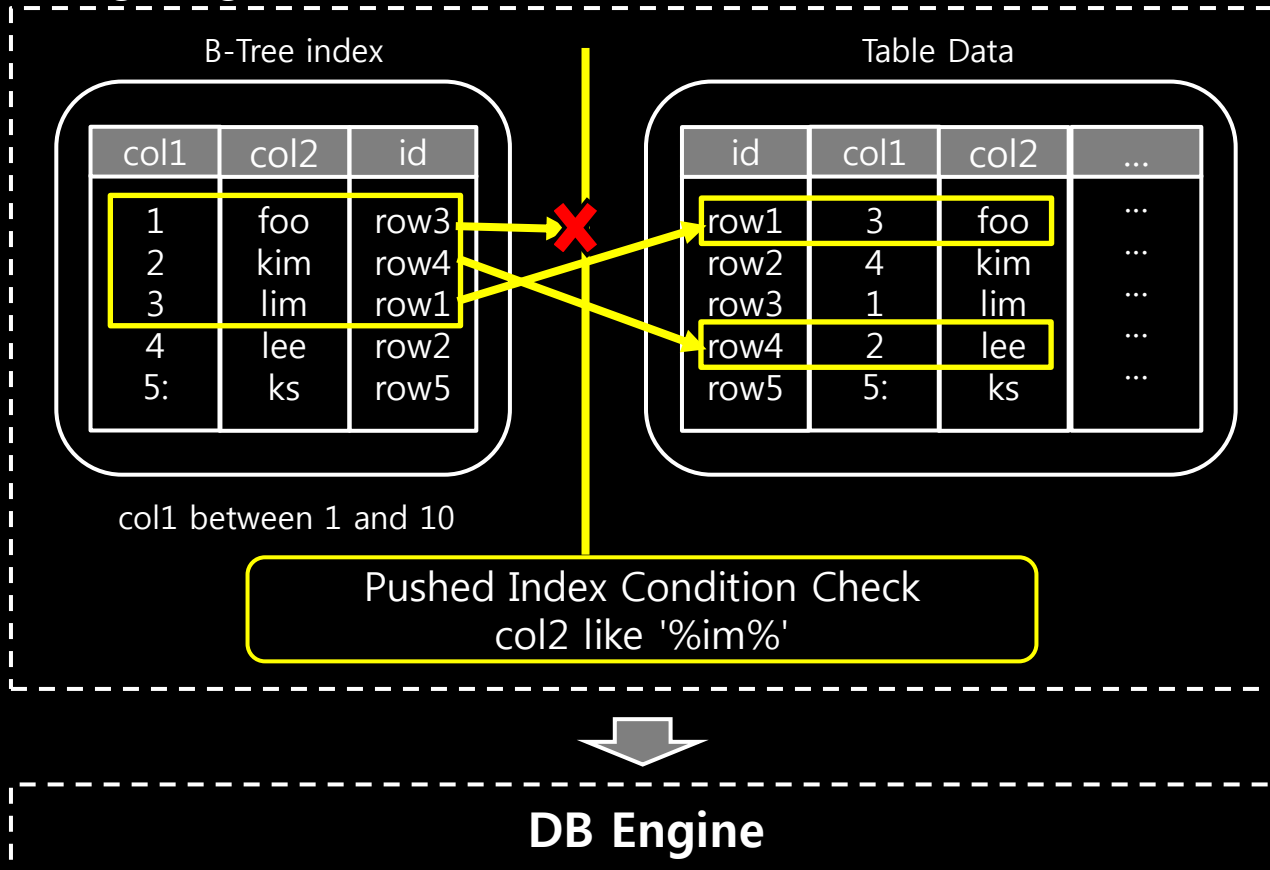
```
SELECT b.name, b.address  
FROM ( SELECT id  
        FROM test01  
        WHERE col1 between 1 and 3  
        AND col2 like '%im%'  
      ) a  
INNER JOIN test01 b ON a.id= b.id
```



차이점 및 주의사항

- MySQL은 인덱스 자체 FILTER 기능이 없다 -> **어떻게 해결하지?**
 - ICP(Pushed Index Condition Check) 기능을 제공하는 MySQL 5.6 사용

Storage Engine

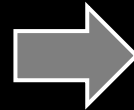


차이점 및 주의사항

- MySQL **SubQuery**는 느리다.

EMP table (사원 정보)
- Primary index : 사원번호
- 1억건의 데이터 존재

EMP_MVP table (MVP 사원 정보)
- Index : 평가년도
- 500 건의 데이터 존재



2010년도
Daum MVP 를 조회
(500명)

```
SELECT *  
FROM EMP  
WHERE  
  사원번호 IN (SELECT 사원번호  
                FROM emp_mvp  
                WHERE 평가년도 = '2010'  
                AND 등급 =1)
```

1. 2010년도 MVP를
평가년도로 RANGE 스캔

2. 추출한 MVP (500건)를
EMP 테이블의 사원번호 PK 스캔



차이점 및 주의사항

- MySQL **SubQuery**는 느리다.

1. EMP 1억 건 다 읽으면서

2. EMP_MVP 조인 비교할꺼야

```
SELECT *  
FROM EMP  
WHERE  
  사원번호 IN  
    (SELECT 사원번호  
     FROM emp_mvp  
     WHERE 평가년도 = '2010'  
     AND 등급 =1)
```

Exists
변경함



```
SELECT *  
FROM EMP a ①  
WHERE  
  EXISTS  
    (SELECT 사원번호  
     FROM emp_mvp b ②  
     WHERE 평가년도 = '2010'  
     AND 등급 =1  
     AND a.사원번호 = b.사원번호)
```

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	EMP	ALL	NULL	NULL	100000000	Using where
2	DEPENDENT SUBQUERY	emp_mvp	ref	gradeyear_idx	const	500	Using where

사원 1명에

mvp 500명 비교 * 1억명

소요시간 : ...
읽는 ROW : 500억 건

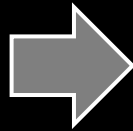
차이점 및 주의사항

- MySQL SubQuery는 느리다. -> 어떻게 해결하지?

서브쿼리를 조인으로 변경

EMP_MVP 조인시 INDEX로

DB 업그레이드



```
SELECT a.*  
FROM EMP a ,  
      INNER JOIN  
      (SELECT DISTINCT eno  
       FROM emp_mvp  
       WHERE 평가년도 = '2010'  
        AND 등급 =1) b  
ON a.empno = b.empno
```

EMP_MVP 인덱스 생성
- IDX2 (사원번호, 평가년도)

MYSQL 5.6

차이점 및 주의사항

- Distinct VS Group by
 - **성능 차이는 없음**
 - Distinct : 중복을 제거한 데이터를 조회시 사용
 - Group by : 데이터를 그룹핑해서 그 결과를 가져오는 경우 사용
- 사용가이드
 - 집계함수를 사용하여 특정 그룹으로 구분 할 때는 **GROUP BY**
 - 특정 그룹 구분없이 중복된 데이터를 제거할 경우 **DISTINCT**

```
SELECT COUNT(DISTINCT fd1) FROM tab;
```

```
SELECT fd1 ,MIN(fd2), MAX(fd2) FROM tab  
GROUP BY fd1;
```

차이점 및 주의사항

- Distinct VS Group by
 - updated 컬럼 Index 있음
 - 정렬작업 없이 정렬된 데이터 출력

Distinct

select SQL_NO_CACHE distinct updated from USER -> (0.25424)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	USER	range	NULL	a	8	NULL	100373	Using index for group-by

Group by

select SQL_NO_CACHE updated from USER group by updated -> (0 0.25119)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	USER	range	NULL	a	8	NULL	100373	Using index for group-by

차이점 및 주의사항

- Distinct VS Group by
 - updated 컬럼 Index 없음
 - 정렬 안된 데이터 출력 및 Temp 테이블 생성

Distinct

select SQL_NO_CACHE distinct updated from USER -> (0.34134)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	USER	ALL	NULL	NULL	NULL	NULL	100207	Using temporary

Group by

select SQL_NO_CACHE updated from USER group by updated -> (0.34903)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	USER	ALL	NULL	NULL	NULL	NULL	99768	Using temporary

"진정한 교육은 가르치는 것이 아니라,
함께 성장하기 위한 것이고
서로에게 도움을 주는 것이라고 생각합니다."



22