

**RDBMS**

# Index

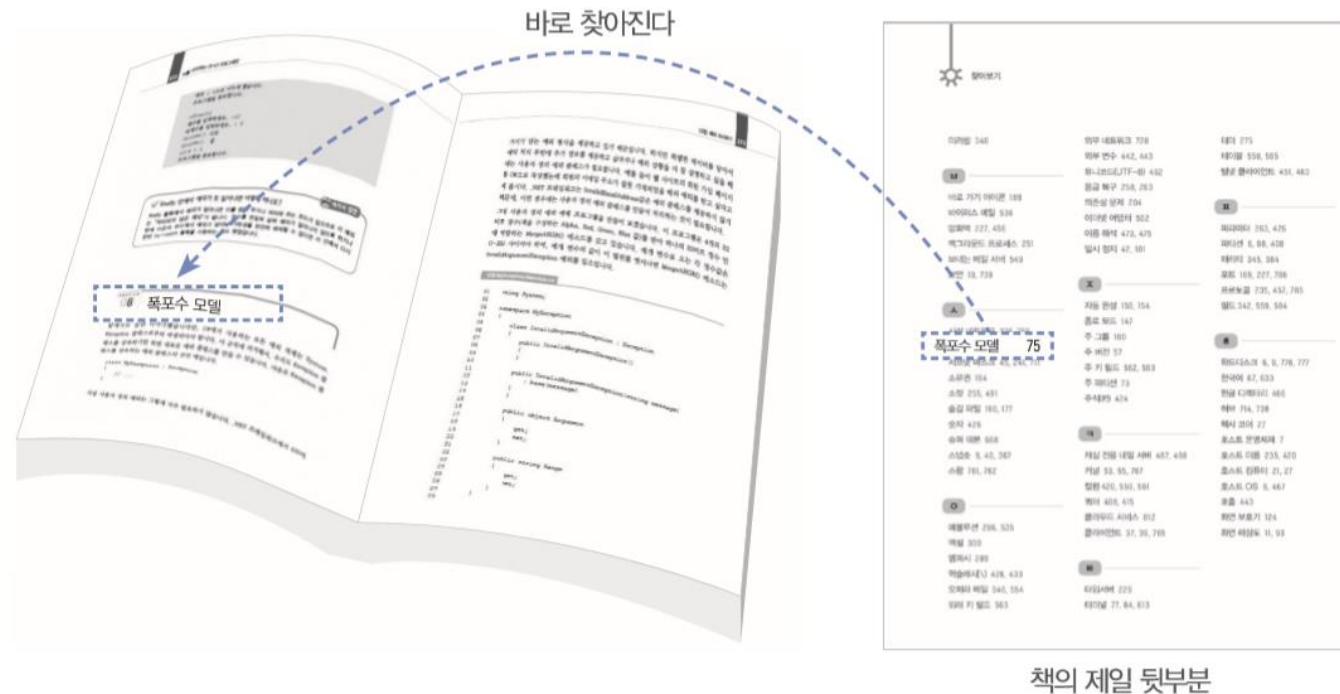
강사 양석환



# Index 개요



- 책의 “찾아보기”의 개념과 비슷함
- 데이터를 좀 더 빠르게 찾을 수 있도록 해 주는 도구



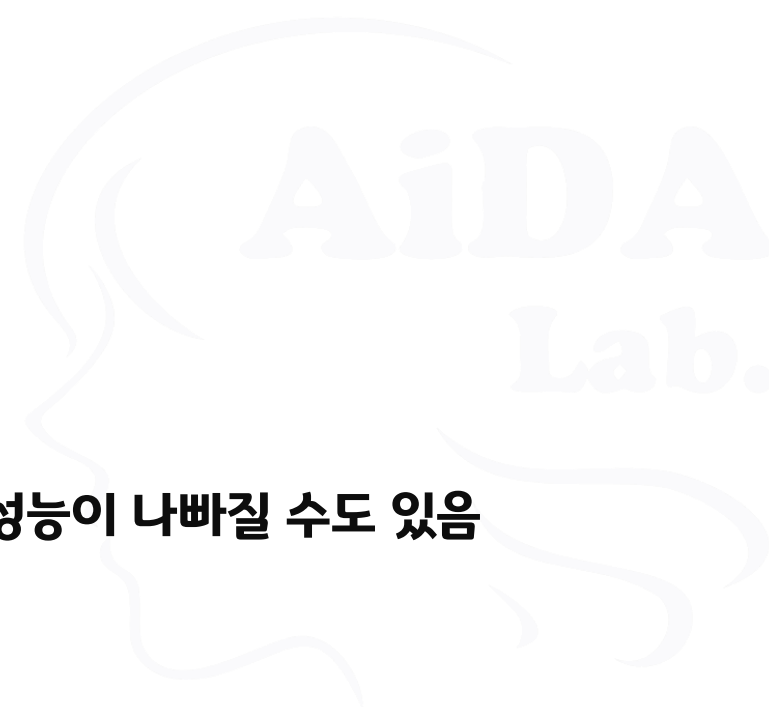
[그림 9-1] 책의 ‘찾아보기’ 개념

## • 장점

- 검색 속도가 무척 빨라질 수 있음 (항상 그런 것은 아님)
- 쿼리의 부하가 줄어들어 시스템 전체의 성능 향상

## • 단점

- 인덱스가 데이터베이스 공간을 차지해서 추가적인 공간 필요
- 대략 데이터베이스 크기의 10% 정도의 추가 공간 필요
- 처음 인덱스 생성하는데 시간 소요
- 데이터의 변경 (Insert, Update, Delete)이 자주 일어나는 경우 성능이 나빠질 수도 있음



- 클러스터형 인덱스 (Clustered Index)
  - ‘영어 사전’과 같은 책
  - 테이블 당 한 개만 지정 가능
  - 행 데이터를 인덱스로 지정한 열에 맞춰 자동 정렬
- 보조 인덱스 (Secondary Index)
  - 책 뒤에 <찾아보기>가 있는 일반 책
  - 테이블당 여러 개도 생성 가능



# Index의 생성과 삭제



- sqlDB에서 작업 → 테이블 생성

```
USE sqldb;  
CREATE TABLE tbl1  
(  
    a    INT    PRIMARY KEY,  
    b    INT,  
    c    INT  
);
```

- 인덱스 상태 확인

```
SHOW INDEX FROM tbl1;
```

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
▶	tbl1	0	PRIMARY	1	a	A	0	NULL	NULL		BTREE		

- Primary Key와 함께 Unique 제약 조건 생성

```
CREATE TABLE tbl2  
(  
    a    INT PRIMARY KEY,  
    b    INT UNIQUE,  
    c    INT UNIQUE,  
    d    INT );
```

- 인덱스 상태 확인 :

```
SHOW INDEX FROM tbl2;
```

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
▶	tbl2	0	PRIMARY	1	a	A	0	NULL	NULL		BTREE		
	tbl2	0	b	1	b	A	0	NULL	NULL	YES	BTREE		
	tbl2	0	c	1	c	A	0	NULL	NULL	YES	BTREE		



- Primary Key 없이 Unique Key만 지정

```
CREATE TABLE tbl3  
(  
    a INT UNIQUE,  
    b INT UNIQUE,  
    c INT UNIQUE,  
    d INT );
```

- 인덱스 상태 확인 :

```
SHOW INDEX FROM tbl3;
```

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
▶	tbl3	0	a	1	a	A	0	HULL	HULL	YES	BTREE		
	tbl3	0	b	1	b	A	0	HULL	HULL	YES	BTREE		
	tbl3	0	c	1	c	A	0	HULL	HULL	YES	BTREE		

- UNIQUE에 클러스터형 인덱스 지정, UNIQUE에 NOT NULL이 포함되면 클러스터형 인덱스로 지정됨

```
CREATE TABLE tbl4  
(  
    a    INT    UNIQUE NOT NULL,  
    b    INT    UNIQUE,  
    c    INT    UNIQUE,  
    d    INT    );
```

- 인덱스 상태 확인 :

```
SHOW INDEX FROM tbl4;
```

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
▶	tbl4	0	a	1	a	A	0	NULL	NULL		BTREE		
	tbl4	0	b	1	b	A	0	NULL	NULL	YES	BTREE		
	tbl4	0	c	1	c	A	0	NULL	NULL	YES	BTREE		

- UNIQUE에 NOT NULL과 PRIMARY KEY를 모두 지정

```
CREATE TABLE tbl5  
(  
    a    INT    UNIQUE NOT NULL,  
    b    INT    UNIQUE,  
    c    INT    UNIQUE,  
    d    INT    PRIMARY KEY );
```

- 인덱스 상태 확인 :

```
SHOW INDEX FROM tbl5;
```

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
▶	tbl5	0	PRIMARY	1	d	A	0	NULL	NULL		BTREE		
	tbl5	0	a	1	a	A	0	NULL	NULL		BTREE		
	tbl5	0	b	1	b	A	0	NULL	NULL	YES	BTREE		
	tbl5	0	c	1	c	A	0	NULL	NULL	YES	BTREE		

- 예제

- 회원 테이블의 열만 정의

```
CREATE DATABASE IF NOT EXISTS testdb;  
USE testdb;  
DROP TABLE IF EXISTS usertbl;  
CREATE TABLE usertbl  
( userID  char(8) NOT NULL PRIMARY KEY,  
  name    varchar(10) NOT NULL,  
  birthYear  int NOT NULL,  
  addr      nchar(2) NOT NULL  
);
```



## • 데이터 입력 및 확인

```
INSERT INTO usertbl VALUES('LSG', '이승기', 1987, '서울');  
INSERT INTO usertbl VALUES('KBS', '김범수', 1979, '경남');  
INSERT INTO usertbl VALUES('KKH', '김경호', 1971, '전남');  
INSERT INTO usertbl VALUES('JYP', '조용필', 1950, '경기');  
INSERT INTO usertbl VALUES('SSK', '성시경', 1979, '서울');  
SELECT * FROM usertbl;
```

	userID	name	birthYear	addr
▶	JYP	조용필	1950	경기
	KBS	김범수	1979	경남
	KKH	김경호	1971	전남
	LSG	이승기	1987	서울
	SSK	성시경	1979	서울

- userID열의 Primary Key를 제거하고, name열을 Primary Key로 지정

```
ALTER TABLE usertbl DROP PRIMARY KEY ;  
ALTER TABLE usertbl  
    ADD CONSTRAINT pk_name PRIMARY KEY(name);  
SELECT * FROM usertbl;
```

	userID	name	birthYear	addr
▶	KKH	김경호	1971	전남
	KBS	김범수	1979	경남
	SSK	성시경	1979	서울
	LSG	이승기	1987	서울
	JYP	조용필	1950	경기

## • 자동 생성 인덱스의 특징

- PRIMARY KEY로 지정한 열은 클러스터형 인덱스가 생성
- UNIQUE NOT NULL로 지정한 열은 클러스터형 인덱스 생성
- UNIQUE(또는 UNIQUE NULL)로 지정한 열은 보조 인덱스 생성
- PRIMARY KEY와 UNIQUE NOT NULL이 존재
- PRIMARY KEY와 UNIQUE NOT NULL이 있으면 PRIMARY KEY에 지정한 열에 우선 클러스터형 인덱스 생성
- PRIMARY KEY로 지정한 열로 데이터가 오름차순 정렬

## • 인덱스 생성 형식

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name  
    [index_type]  
    ON tbl_name (key_part,...)  
    [index_option]  
    [algorithm_option | lock_option] ...
```

```
key_part:  
    {col_name [(length)] | (expr)} [ASC | DESC]
```

```
index_option:  
    KEY_BLOCK_SIZE [=] value  
    | index_type  
    | WITH PARSEr parser_name  
    | COMMENT 'string'  
    | {VISIBLE | INVISIBLE}
```

```
index_type:  
    USING {BTREE | HASH}
```

```
algorithm_option:  
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}
```

```
lock_option:  
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```



## • 인덱스 삭제 형식

```
DROP INDEX index_name ON tbl_name  
    [algorithm_option | lock_option] ...  
  
algorithm_option:  
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}  
  
lock_option:  
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

```
DROP INDEX 인덱스이름 ON 테이블이름;
```



## Index의 성능 비교



- 인덱스가 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도와 서버 부하 비교
  - 실습할 데이터베이스 만들기
    - CREATE DATABASE IF NOT EXISTS indexdb;
  - employees의 employees의 개수를 파악
    - USE indexdb;
    - SELECT COUNT(\*) FROM employees.employees;
  - 테이블 3개로 복사
    - CREATE TABLE emp SELECT \* FROM employees.employees ORDER BY RAND();
    - CREATE TABLE emp\_c SELECT \* FROM employees.employees ORDER BY RAND();
    - CREATE TABLE emp\_Se SELECT \* FROM employees.employees ORDER BY RAND();

## • 테이블 순서 확인

- SELECT \* FROM emp LIMIT 5;
- SELECT \* FROM emp\_c LIMIT 5;
- SELECT \* FROM emp\_Se LIMIT 5;

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	443622	1963-05-02	Elvis	Bage	M	1991-07-01
	293770	1959-08-23	Gianluca	Validov	M	1994-03-23
	463994	1962-01-11	Jouni	Borstler	F	1986-08-18
	448204	1953-05-31	Mani	Ebeling	M	1993-09-23
	269311	1953-09-09	Gino	Chepyzhov	F	1995-09-28

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	474999	1962-07-29	Leon	Staudhammer	M	1989-11-07
	495190	1958-04-07	Maia	Miara	F	1989-12-31
	264498	1955-12-07	Gaetan	Hertweck	M	1991-04-25
	406048	1956-02-03	Abdelghani	Alpay	F	1993-03-25
	259326	1955-06-30	Sachar	Perly	M	1988-08-25

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	459564	1960-07-06	Heping	Baek	M	1985-06-06
	61157	1954-02-25	Shim	McConalogue	F	1991-08-23
	71977	1964-05-08	Jordanka	Barbanera	M	1993-04-29
	497952	1953-01-31	Valeri	Plotkin	M	1997-05-16
	471926	1962-10-18	Hirochika	Chaudhury	F	1999-06-02

- SHOW TABLE STATUS문으로 테이블에 인덱스 있는지 확인
  - 세 테이블 모두 인덱스 없음

	Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length	Max_data_length	Index_length	Data_free	Auto_increment	Create_time
▶	emp	InnoDB	10	Dynamic	299088	57	17317888	0	0	4194304	NULL	2016-03-06 11:05:51
	emp_c	InnoDB	10	Dynamic	299841	57	17317888	0	0	4194304	NULL	2016-03-06 11:05:54
	emp_se	InnoDB	10	Dynamic	299389	57	17317888	0	0	4194304	NULL	2016-03-06 11:05:57

- emp\_c에는 클러스터형 인덱스(=Primary Key 인덱스)
- emp\_Se에는 보조 인덱스를 생성
  - ALTER TABLE emp\_c ADD PRIMARY KEY(emp\_no);
  - ALTER TABLE emp\_Se  
ADD INDEX idx\_emp\_no (emp\_no);

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	443622	1963-05-02	Elvis	Bage	M	1991-07-01
	293770	1959-08-23	Gianluca	Validov	M	1994-03-23
	463994	1962-01-11	Jouni	Borstler	F	1986-08-18
	448204	1953-05-31	Mani	Ebeling	M	1993-09-23
	269311	1953-09-09	Gino	Chepyzhov	F	1995-09-28

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	10001	1953-09-02	Georgi	Facello	M	1986-06-26
	10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
	10003	1959-12-03	Parto	Bamford	M	1986-08-28
	10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
	10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	459564	1960-07-06	Heping	Baek	M	1985-06-06
	61157	1954-02-25	Shim	McConalogue	F	1991-08-23
	71977	1964-05-08	Jordanka	Barbanera	M	1993-04-29
	497952	1953-01-31	Valeri	Plotkin	M	1997-05-16
	471926	1962-10-18	Hirochika	Chaudhury	F	1999-06-02

- 생성한 인덱스 적용

- ANALYZE문 사용
- ANALYZE TABLE emp, emp\_c, emp\_Se;

	Table	Op	Msg_type	Msg_text
▶	indexdb.emp	analyze	status	OK
	indexdb.emp_c	analyze	status	OK
	indexdb.emp_se	analyze	status	OK

## • 인덱스 생성 후 테이블 인덱스 확인

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
--	-------	------------	----------	--------------	-------------	-----------	-------------	----------	--------	------	------------

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
▶	emp_c	0	PRIMARY	1	emp_no	A	299468	NULL	NULL		BTREE

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
▶	emp_se	1	idx_emp_no	1	emp_no	A	299339	NULL	NULL		BTREE

	Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length	Max_data_length	Index_length	Data_free
▶	emp	InnoDB	10	Dynamic	299238	57	17317888	0	0	4194304
	emp_c	InnoDB	10	Dynamic	299468	57	17317888	0	0	2097152
	emp_se	InnoDB	10	Dynamic	299339	57	17317888	0	5783552	2097152

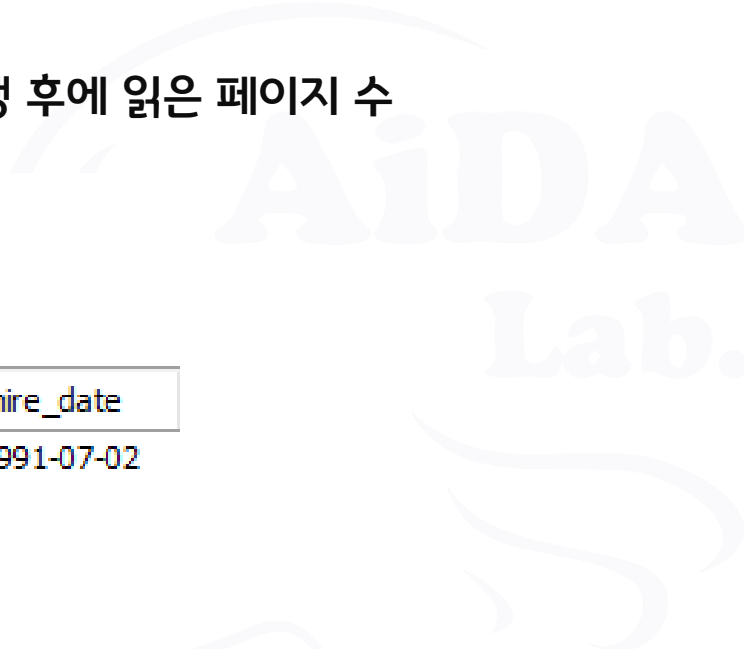


- **인덱스 생성 후 테이블 인덱스 확인**

- emp 테이블은 인덱스 없음
- emp\_c는 클러스터형(PRIMARY) 인덱스 생성되어 Data\_free 영역이 줄어듦
- emp\_Se는 보조 인덱스 생성되어 데이터 변화 없음, 인덱스 페이지만 추가 생성됨



- MySQL 전체의 시스템 상태 초기화
- 인덱스 없는 emp 테이블 조회
  - SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 전의 읽은 페이지 수
  - SELECT \* FROM emp WHERE emp\_no = 100000;
  - SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 후에 읽은 페이지 수



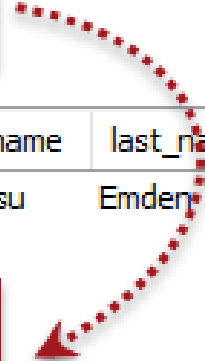
	Variable_name	Value
▶	Innodb_pages_read	1032

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	100000	1956-01-11	Hiroyasu	Emden	M	1991-07-02

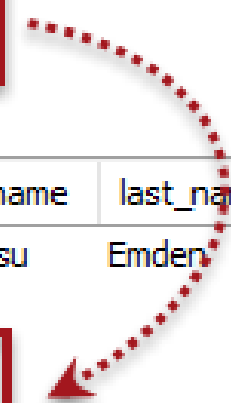
  

	Variable_name	Value
▶	Innodb_pages_read	2090



- 클러스터형 인덱스가 있는 테이블 조회

- SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 전의 읽은 페이지 수
- SELECT \* FROM emp\_c WHERE emp\_no = 100000;
- SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 후의 읽은 페이지



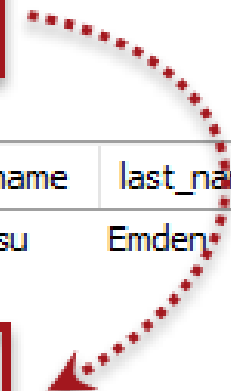
Variable_name	Value
Innodb_pages_read	3159

emp_no	birth_date	first_name	last_name	gender	hire_date
100000	1956-01-11	Hiroyasu	Emden	M	1991-07-02

Variable_name	Value
Innodb_pages_read	3162

- 보조 인덱스가 있는 테이블 조회

- SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 전의 읽은 페이지 수
- SELECT \* FROM emp\_Se WHERE emp\_no = 100000;
- SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 후의 읽은 페이지 수

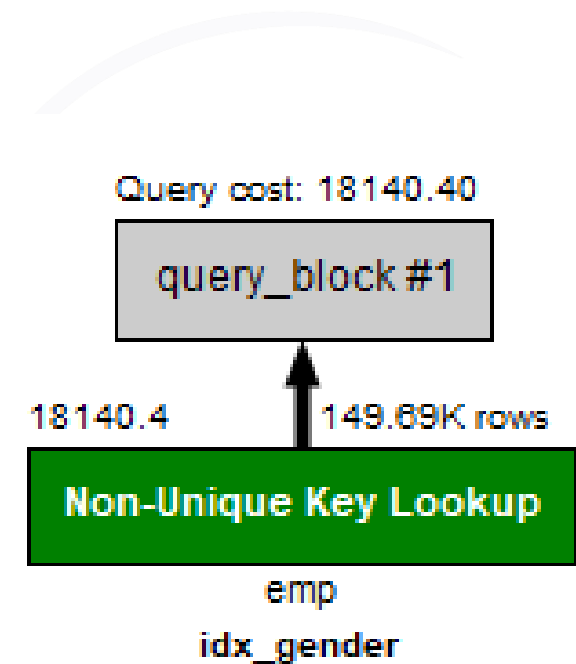


Variable_name	Value
Innodb_pages_read	3162

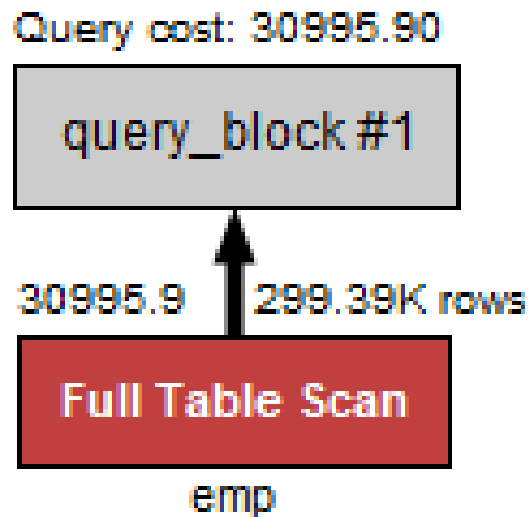
emp_no	birth_date	first_name	last_name	gender	hire_date
100000	1956-01-11	Hiroyasu	Emden	M	1991-07-02

Variable_name	Value
Innodb_pages_read	3167

- 데이터 중복도에 따른 인덱스의 효용
- 인덱스를 만들지 않은 emp테이블의 gender(성별)열에 인덱스 생성
  - ALTER TABLE emp ADD INDEX idx\_gender (gender);
  - ANALYZE TABLE Emp; -- 생성한 인덱스를 통계에 적용시킴
  - SHOW INDEX FROM Emp;
  - SELECT \* FROM emp WHERE gender = 'M' LIMIT 500000;
- 쿼리 비용 약 2만 정도 나옴
- Data Read 19MB 정도 읽음



- 강제로 인덱스를 사용하지 못하게 한 후에 실행 계획 비교
  - `SELECT * FROM emp IGNORE INDEX (idx_gender) WHERE gender = 'M' LIMIT 500000;`
- 쿼리 비용 약 3만 정도 나옴



## • 비교 실습 결과

- 데이터의 중복도가 높은 경우에, 인덱스 사용하는 것이 효율이 있음
- 하지만 인덱스의 관리 비용과 INSERT 등의 구문에서는 오히려 성능이 저하될 수 있다는 점 등을 고려하면 인덱스가 반드시 바람직하다고 보기는 어려움

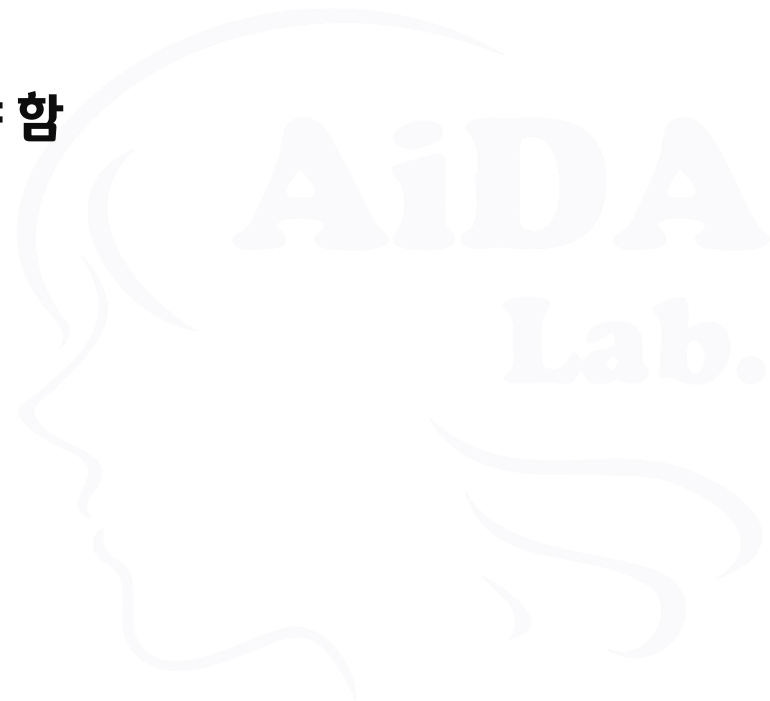


- **비교 실습에 따른 결론**

- **인덱스는 열 단위에 생성**
  - 두 개 이상의 열을 조합해서 인덱스 생성 가능
- **WHERE절에서 사용되는 열에 인덱스를 만들어야 함**
  - 테이블 조회 시 WHERE절의 조건에 해당 열이 나오는 경우에만 인덱스 주로 사용
- **WHERE절에 사용되더라도 자주 사용해야 가치가 있음**
  - SELECT문이 자주 사용 되어야 효과적
  - INSERT문이 자주 사용되고 생성된 인덱스가 클러스터형이면 효율 감소



- 데이터의 중복도가 높은 열은 인덱스 만들어도 효과 없음
  - 인덱스의 관리 비용 때문에 인덱스가 없는 편이 나은 경우도 있음
- 외래 키 지정한 열에는 자동으로 외래 키 인덱스가 생성
- JOIN에 자주 사용되는 열에는 인덱스를 생성해 주는 것이 좋음
- INSERT/UPDATE/DELETE가 얼마나 자주 일어나는지 고려해야 함
  - 인덱스는 단지 읽기에서만 성능 향상
  - 데이터의 변경에서는 오히려 부담



- 클러스터형 인덱스는 테이블당 하나만 생성 가능
  - 클러스터형 인덱스를 생성할 열은 범위(BETWEEN, >, < 등의 조건)로 사용하거나 집계 함수를 사용하는 경우 아주 적절하게 사용
- 클러스터형 인덱스가 테이블에 아예 없는 것이 좋은 경우도 있음
- 사용하지 않는 인덱스는 제거
  - 공간 확보 및 데이터의 입력 시에 발생하는 부하 줄임



**THANK  
YOU**

