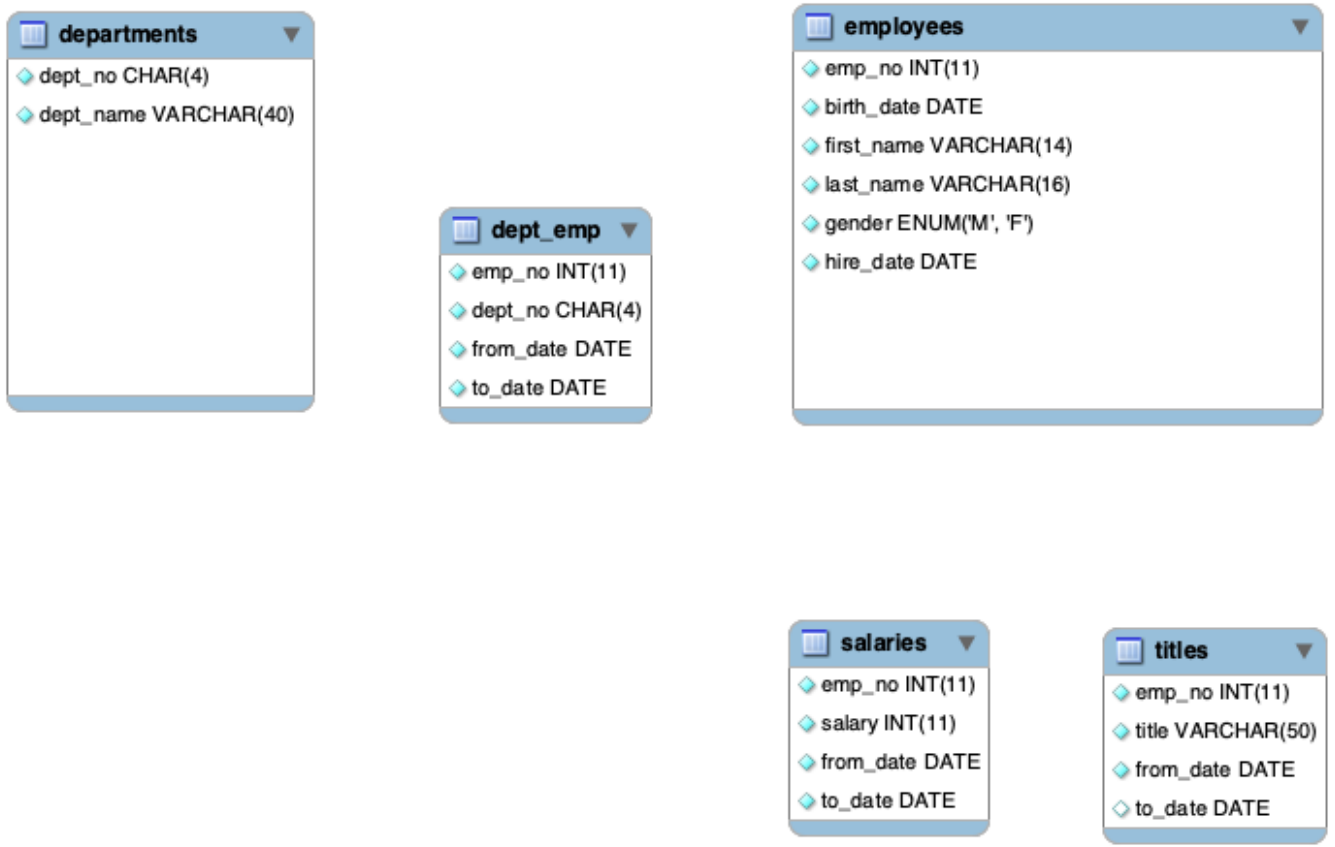
관계형 데이터베이스 성능개선

이상화



Database 설명

위는 아무런 index 가 만들어져 있지 않은 가상의 회사 DB 이다.

Departments 는 부서의 고유 번호와 부서의 이름의 데이터를 가지고 있는 테이블이다. 총 9 개의 row 를 가지고 있다.

Employees 는 직원의 고유 번호와 생년 월일, 이름, 성별 그리고 고용된 날짜를 가지고 있는 테이블이다. 총 300,024 개의 row 를 가지고 있다.

Dept_emp 는 부서와 직원을 매칭시킨 테이블이다. 언제부터 언제까지 그 부서에 있었는지에 대한 정보를 가지고있다. 총 331,603 개의 row 를 가지고 있다.

Salaries 는 직원의 봉급에 대한 테이블로, 언제부터 언제까지 그 봉급을 받았는지에 대한 정보를 가지고있다. 총 2,844,047 개의 row 를 가지고 있다.

Titles 는 직원의 직책에 대한 테이블로, 언제부터 언제까지 그 직책이었는지에 대한 정보를 가지고있다. 총 443,308 개의 row 를 가지고 있다.

테스트 환경

프로세서: 2.4GHz Quad - Core Intel Core i5

메모리: 16GB LPDDR3

디스크: APPLE SSD AP0256M

엔진: InnoDB

MySQL: Ver 8.0.18 for osx10.15 on x86_64

조회 쿼리 사용시 SQL_NO_CACHE 를 사용하여 cache 를 사용하지않도록 함.

JOIN: MySQL Optimizer 가 Driving Table 과 Driven Table 을 알아서 고르기 때문에 STRAIGHT_JOIN 을 통해 강제화

Employees 개선

	기준 Query	개선 전 Query Cost / duration	개선 전 query plan	문제점
SELECT	SELECT SQL_NO_CACHE * FROM employees WHERE emp_no = 440433;	30203.15 / 0.103 sec	FULL TABLE SCAN	고유한 emp_no 가 440433 인 행을 검색할 때, 실제 데이터들을 모두 검색해야한다.
INSERT	INSERT INTO employees(emp_no, birth_date, first_name, last_name, gender, hire_date) VALUES (440433, '1960-11-20', 'Kolar', 'Decaestecker', 'F', '1995-05-14');	Null / 0.0010 sec		실제 데이터의 맨 끝에 삽입되면 되므로 문제가 되지않는다.
UPDATE	UPDATE employees SET first_name = 'TestLastName' WHERE emp_no = 440433;	Null / 0.138 sec		고유한 emp_no 가 440433 인 행을 검색할 때, 실제 데이터들을 모두 검색해야한다.
DELETE	DELETE FROM employees WHERE emp_no = 440433;	Null / 0.135 sec		고유한 emp_no 가 440433 인 행을 검색할 때, 실제 데이터들을 모두 검색해야한다.

해결방안	개선 후 Query Cost / duration	개선 후 query plan
실제 데이터가 emp_no 를 기준으로 정렬되어 있다면 모든 데이터를 둘러보지않아도 된다.	1.00 / 0.00040 sec	Single Row (Constant) Primary
하지만 Clustered index 가 만들어진 이후의 삽입은 실제 데이터에서 재정렬이 이뤄져야한다.	Null / 0.0012 sec	
실제 데이터가 emp_no 를 기준으로 정렬되어 있다면 모든 데이터를 둘러보지않아도 된다. 하지만 primary key 가 변경되면 재정렬이 이루어져야 하는 문제가 생긴다.	Null / 0.00042 sec	
실제 데이터가 emp_no 를 기준으로 정렬되어 있다면 모든 데이터를 둘러보지않아도 된다.	Null / 0.00071 sec	

Employees 테이블에 emp_no 가 primary key 로 유일한 인덱스 일 때,

Development 부서의 직원 각각의 지금까지의 급여의 평균을 계산하여라

```
SELECT dept_emp.emp_no, AVG(salaries.salary)
FROM departments
STRAIGHT_JOIN dept_emp
ON departments.dept_name = 'development' AND dept_emp.dept_no = departments.dept_no
STRAIGHT_JOIN salaries
ON salaries.emp_no = dept_emp.emp_no
GROUP BY dept_emp.emp_no;
```

Duration: 2.008 sec, Query cost: 9,391,541,041.24

문제점 1: departments 테이블에서 dept_name 은 unique 하지만 그에 맞는 키가 만들어져있지 않다.

해결방안 1: departments 테이블에서 dept_name 을 unique index 로 설정

문제점 2: departments 테이블과 dept_emp 테이블 간의 JOIN 이 이루어질 때 필요한 dept_no 가 index 로 만들어져있지 않다.

해결방안 2: departments 의 dept_no 를 primary key 로 설정하고, dept_emp 의 dept_no 를 index 로 설정한다.

문제점 3: dept_emp 테이블과 salaries 테이블이 JOIN 이루어질 때 필요한 emp_no 가 index 로 만들어져있지 않다

해결방안 3: dept_emp 테이블의 emp_no 를 index 로 설정하고, salaries 테이블의 emp_no 를 index 로 설정한다.

```
ALTER TABLE departments ADD UNIQUE INDEX(dept_name);
ALTER TABLE departments ADD PRIMARY KEY(dept_no);
ALTER TABLE dept_emp ADD INDEX(dept_no);
ALTER TABLE dept_emp ADD INDEX(emp_no);
ALTER TABLE salaries ADD INDEX(emp_no);
```

이후, Query cost 는 1,461,616.31 로 매우 줄었지만 Duration 이 15.370 sec 로, 개선이 되었다고 보기에는 힘들었다.

중간 평가: Non-clustered Index 보다 Clustered Index 를 통해 성능을 높여보자.

1. dept_emp 에는 Primary Key 가 없으므로 index 로 설정되어야했던 (dept_no, emp_no)를 Primary Key 로 설정해보자.

2. salaries 에는 Primary Key 가 없으므로 index 로 설정되어야했던 emp_no 를 포함하여 (emp_no, salary, from_date) 를 Primary Key 로 설정한다.

```
ALTER TABLE dept_emp ADD PRIMARY KEY(dept_no, emp_no);
ALTER TABLE salaries ADD PRIMARY KEY(emp_no, salary, from_date);
```

이후, Query cost 는 343,352.83 로 줄었고 Duration 이 0.0060 sec 로 매우 개선된 것을 확인할 수 있었다. 하지만 Fetch Time 이 0.3 sec 정도 소요되어 LIMIT 조건을 걸어주어 이 시간을 줄여야 하는 불편함이 있었다.