ORACLE

15. 인덱스

목차

1. 인덱스 개념

- 인덱스(INDEX)
  - SQL 명령문의 처리 속도를 향상시키기 위해서 컬럼에 대해서 생성하는 오라클 객체다.

- 인덱스 사용 이유
  - 테이블 생성 방법을 책에서 찾으려고 할 때 어떻게 할까? 책 첫 페이지부터 한 장씩 넘겨가면서 테이블 생성 방법이 기술되어 있는지 일일이 살펴보는 사람은 드물 것이다.
  - 일반적으로 책 맨 뒤(앞)에 있는 색인(인덱스, 찾아보기)에서 해당 단어(테이블)를 찾아 그 페이지로 이동한다.
  - 이렇게 원하는 단어를 쉽게 찾는 방법으로 색인, 인덱스가 사용되는 것처럼 오라클의 인덱스 역시 원하는 데이터를 빨리 찾기 위해서 사용된다.

1.1 인덱스 주의점

- 오라클에서의 인덱스의 내부 구조는 B* 트리 형식으로 구성되어 있다.
- 트리란 나무의 뿌리 모양을 생각해 보시면 쉽게 이해할 수 있다.
- 뿌리(루트)를 근거로 아래로 나무뿌리 들이 뻗어 있는 모양을 하고 있다.
- 컬럼에 인덱스를 설정하면 이를 위한 B* 트리도 생성되어야 하기 때문에 인덱스를 생성하기 위한 시간도 필요하고 인덱스를 위한 추가적인 공간이 필요하게 된다.
- 인덱스가 생성된 후에 새로운 행을 추가하거나 삭제할 경우 인덱스로 사용된 컬럼 값도 함께 변경되는 경우가 발생한다.
- 인덱스로 사용된 컬럼 값이 변경될 때 이를 위한 내부 구조(B* 트리) 역시 함께 수정 되어야 한다.
- 이 작업은 오라클 서버에 의해 자동으로 일어나는데 그렇기 때문에 인덱스가 없는 경우 보다 인덱스가 있는 경우에 DML 작업이 훨씬 무거워지게 된다.

1.2 인덱스 장/단점

- 인덱스 장점
  - 검색 속도가 빨라진다.
  - 시스템에 걸리는 부하를 줄여서 시스템 전체 성능을 향상시킨다.

- 인덱스 단점
  - 인덱스를 위한 추가적인 공간이 필요하다.
  - 인덱스를 생성하는데 시간이 걸린다.
  - 데이터의 변경 작업(INSERT/UPDATE/DELETE)이 자주 일어날 경우에는 오히려 성능이 저하된다.

1.3 인덱스 데이터 딕셔너리

- USER_IND_COLUMNS 데이터 딕셔너리 뷰로 인덱스의 생성 유무를 확인 해보자.

예

```
SELECT INDEX_NAME, TABLE_NAME , COLUMN_NAME
FROM USER_IND_COLUMNS
WHERE TABLE_NAME IN('EMP', 'DEPT');
```

INDEX_NAME	TABLE_NAME	COLUMN_NAME
PK_DEPT	DEPT	DEPTNO
PK_EMP	EMP	EMPNO

1.4 인덱스 정보 조회

- 위 쿼리문의 결과 화면을 통해서 사용자가 인덱스를 생성하지 않았어도 오라클에서 기본 키나 유일 키에 대해서 자동으로 인덱스를 생성한다는 것을 확인할 수 있다.

- 인덱스 역시 테이블이나 뷰, 시퀀스와 같이 오라클 객체의 일종이고 모든 객체들은 이름이 있어야한다.

- 기본 키나 유일 키에 대한 인덱스는 오라클이 생성한 것이기에 인덱스의 이름 역시 오라클에서 자동 부여해준다.

- 자동으로 생성되는 인덱스 이름은 제약 조건(CONSTRAINT)명을 사용함을 확인할 수 있다.

1.5 사원 테이블 복사(1)

- 다음은 인덱스로 인해 검색시간이 현저하게 줄어드는 것을 증명하기 위한 실습을 위해서 사원 테이블을 복사해서 새로운 테이블을 생성해보자.

- 1. 사원(emp) 테이블을 복사해서 새로운 테이블을 생성한다.

예

```
CREATE TABLE EMP01
AS
SELECT * FROM EMP;
```

- 2. EMP와 EMP01 테이블에 인덱스가 설정되어 있는지 확인해보자.

예

```
SELECT TABLE_NAME, INDEX_NAME, COLUMN_NAME
FROM USER_IND_COLUMNS
WHERE TABLE_NAME IN('EMP', 'EMP01');
```

1.5 사원 테이블 복사(2)

- 결과 화면의 USER_IND_COLUMNS 를 살펴보면 EMP 테이블은 EMPNO 컬럼에 인덱스가 존재하지만 EMP를 서브 쿼리로 복사한 EMP01 테이블에 대해서는 어떠한 인덱스도 존재하지 않음을 확인할 수 있습니다. 서브 쿼리문으로 복사한 테이블은 구조와 내용만 복사될 뿐 제약 조건은 복사되지 않기 때문이다.

1.6 인덱스가 아닌 컬럼으로 검색하기(1)

- EMP01 테이블은 인덱스 설정이 되어 있지 않기에 검색하는데 시간이 걸린다. 이를 증명하기 위해서 EMP01 테이블에 수많은 데이터가 저장되어 있어야 한다. 서브 쿼리문으로 INSERT 문을 여러 번 반복해서 EMP01 테이블의 데이터를 늘린 후에 사원이름으로 특정 사원을 찾아보도록 하자. 속도의 차이가 현저하게 난다는 것을 느낄 수 있다.
- 1. 서브 쿼리문으로 INSERT 문을 여러 번 반복한다.

예

```
INSERT INTO EMP01 SELECT * FROM EMP01;
...
...
INSERT INTO EMP01 SELECT * FROM EMP01;
```

- 테이블 자체 복사를 여러 번 반복해서 상당히 많은 양의 행을 생성했다.

1.6 인덱스가 아닌 컬럼으로 검색하기(2)

- 2. 이제 검색용으로 사용할 행을 새롭게 하나 추가한다.

예

```
INSERT INTO EMP01(EMPNO, ENAME)
VALUES(1111, '홍길동');
```

- 3. 시간을 체크하기 위해서 다음과 같은 명령을 입력한다.

예

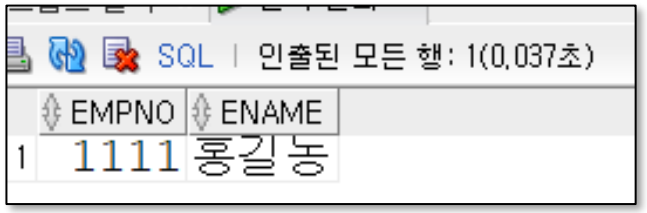
```
SET TIMING ON
```

1.6 인덱스가 아닌 컬럼으로 검색하기(3)

- 4. 사원 이름이 '홍길동'인 행을 검색해 보자.

예

```
SELECT DISTINCT EMPNO, ENAME
FROM EMP01
WHERE ENAME= ' 홍길동';
```



- 컴퓨터의 성능에 따라 검색하는데 소요되는 시간이 다르겠지만, 어느 정도의 시간은 소요됨을 확인할 수 있다.

2. 인덱스 생성(1)

- 제약 조건에 의해 자동으로 생성되는 인덱스 외에 CREATE INDEX 명령어로 직접 인덱스를 생성할 수도 있다.

형식	**CREATE INDEX *index_name*** **ON *table_name (column_name);***

- CREATE INDEX 다음에 인덱스 객체 이름을 지정한다. 어떤 테이블의 어떤 컬럼에 인덱스를 설정할 것인지를 결정하기위해서 ON 절 다음에 테이블 이름과 컬럼 이름을 기술한다.

2. 인덱스 생성(2)

- 인덱스가 지정하지 않은 컬럼인 ENAME 으로 조회하여 어느 정도의 시간은 소요됨을 확인하였으므로 이번에는 ENAME 컬럼으로 인덱스를 지정하여 조회 시간이 단축됨을 확인해보자.
- 1. 이번에는 테이블 EMP01의 컬럼 중에서 이름(ENAME)에 대해서 인덱스를 생성해보자.

예

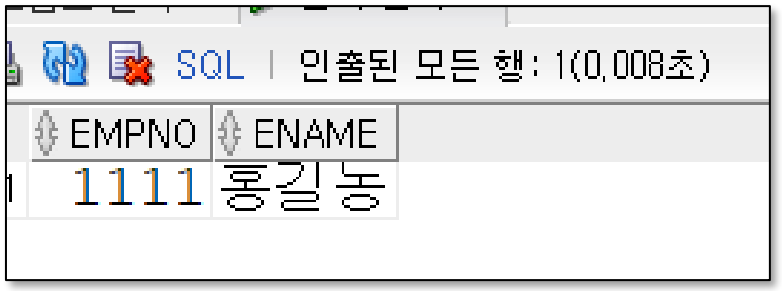
```
CREATE INDEX IDX_EMP01_ENAME
ON EMP01(ENAME);
```

- 2. 사원 이름이 '홍길동'인 행을 검색해보자

예

```
SELECT DISTINCT EMPNO, ENAME
FROM EMP01
WHERE ENAME='홍길동';
```

2. 인덱스 생성(3)



- 인덱스를 생성한 후에 사원 이름이 '홍길동'인 행을 다시 검색하면 수행속도가 매우 감소함을 알 수 있다.

문제

1. EMP01 테이블의 직급 컬럼을 인덱스로 설정하되 인덱스 이름을 IDX_EMP01_JOB로 주고 데이터 딕셔너리를 이용하여 설정된 화면을 출력하시오.

TABLE_NAME	INDEX_NAME	COLUMN_NAME
EMP	PK_EMP	EMPNO
EMP01	IDX_EMP01_ENAME	ENAME
EMP01	IDX_EMP01_JOB	JOB

3. 인덱스 제거

- 인덱스가 검색 속도를 현저하게 줄이는 것을 확인하기 위해서 위와 같은 예제를 실습해 보았다.
- 이번에는 인덱스를 삭제해보자.
- 이를 위해서 오라클은 DROP INDEX 명령어를 제공한다.

형식 **DROP INDEX *index_name;***

- ex) EMP01 테이블의 IDX_EMP01_ENAME만 사용자가 인덱스를 생성했다. 이를 제거보자. 생성된 인덱스 객체를 제거하기 위해서는 DROP INDEX 문을 사용한다.

형식 **DROP INDEX IDX_EMP01_ENAME;**

4. 인덱스 사용 경우 판단하기(1)

- 인덱스가 검색을 위한 처리 속도만 향상시킨다고 했다.
- 하지만, 무조건 인덱스를 사용한다고 검색 속도가 향상되는 것은 아니다.
- 계획성 없이 너무 많은 인덱스를 지정하면 오히려 성능을 저하시킬 수도 있다.

인덱스를 사용해야 하는 경우	인덱스를 사용하지 말아야 하는 경우
테이블에 행의 수가 많을 때	테이블에 행의 수가 적을 때
WHERE 문에 해당 컬럼이 많이 사용될 때	WHERE 문에 해당 컬럼이 자주 사용되지 않을 때
검색 결과가 전체 데이터의 2%~4% 정도 일 때	검색 결과가 전체 데이터의 10%~15% 이상 일 때
JOIN에 자주 사용되는 컬럼이나 NULL을 포함하는 컬럼이 많은 경우	테이블에 DML 작업이 많은 경우 즉, 입력 수정 삭제 등이 자주 일어 날 때

4. 인덱스 사용 경우 판단하기(2)

- 다음과 같은 조건에서 사원 테이블의 부서 번호에 인덱스를 거는 것이 좋을까?
  - 테이블에 전체 행의 수는 10000건이다.
  - 위의 쿼리문을 전체 쿼리문들 중에서 95% 사용된다.
  - 쿼리문의 결과로 구해지는 행은 10건 정도이다.
- 조건을 위 표를 비추어보고 판단해 보면 DEPTNO 컬럼을 인덱스로 사용하기에 알맞다는 결론이 난다.
- 위 결론에 따라 사원 테이블의 부서 번호(DEPTNO)를 인덱스로 지정한다.

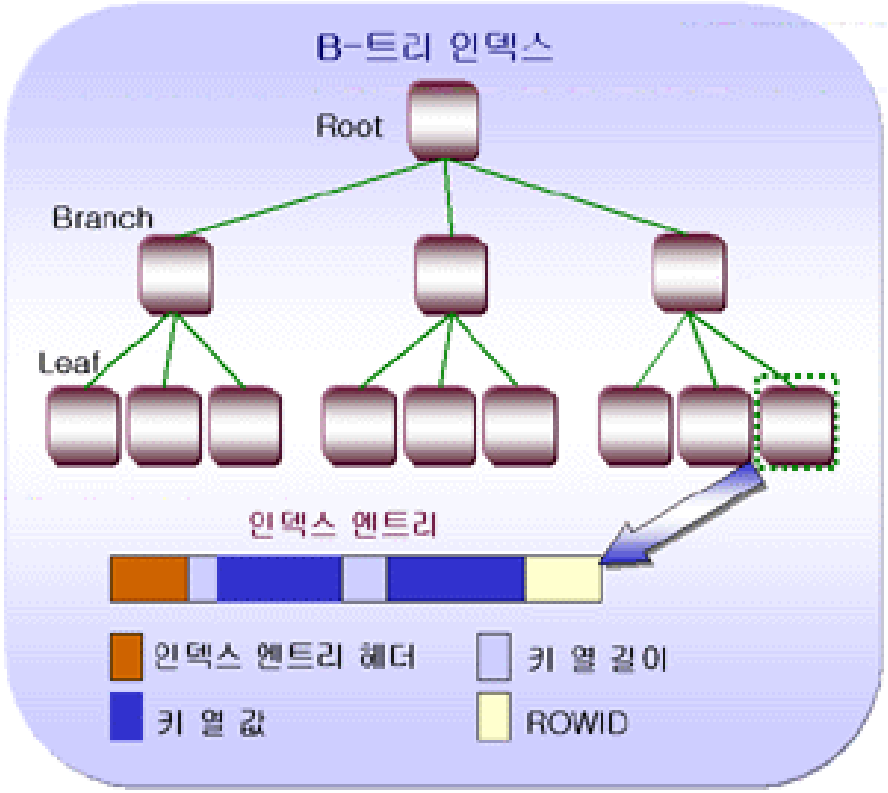
예

```
CREATE INDEX IDX_EMP01_DEPTNO
ON EMP01(DEPTNO);
```

```
Index IDX_EMP01_DEPTNO이(가) 생성되었습니다.
```

5. 인덱스의 물리적인 구조와 인덱스 재생성(1)

- 오라클에서의 인덱스의 내부 구조는 B-트리 형식으로 구성되어있다.
- 트리란 나무의 뿌리 모양을 생각해 보면 쉽게 이해할 수 있다.
- 뿌리(루트)를 근거로 아래로 나무뿌리들이 뻗어 있는 모양을 하고 있다.



5. 인덱스의 물리적인 구조와 인덱스 재생성(2)

- B-트리 형식의 인덱스를 B-트리 인덱스라고 한다.
- B-트리 인덱스를 보다 정확하게 말하자면 인덱스 키에 대해 각각의 인덱스 엔트리로 구성되어 있어 있으며 인덱스 엔트리에 로우 아이디를 저장하고 있고 있다.
- 인덱스가 생성된 후에 새로운 행이 추가되거나 삭제될 수도 있고 인덱스로 사용된 컬럼 값이 변경될 수도 있다.
- 이럴 경우는 본 테이블에서 추가, 삭제, 갱신 작업이 일어날 때 해당 테이블에 걸린 인덱스의 내용도 함께 수정 되어야 한다.
- 이 작업은 오라클 서버에 의해 자동으로 일어나는데 삭제, 갱신 작업이 일어날 경우에 해당 인덱스 엔트리가 바로 인덱스로부터 제거 될까? DELETE 문이 실행되면 논리적인 삭제 과정만 일어난다.

5. 인덱스의 물리적인 구조와 인덱스 재생성(3)

- DML 작업 특히 DELETE 명령을 수행한 경우에는 해당 인덱스 엔트리가 논리적으로만 제거 되고 실제 인덱스 엔트리는 그냥 남아 있게 된다.
- 인덱스에 제거된 엔트리가 많아질 경우에는 제거된 인덱스들이 필요 없는 공간을 차지하고 있기 때문에 종종 인덱스를 재생성시켜야 한다.
- 다음은 인덱스를 재생성할 때 사용하는 기본 형식이다.

형식 **ALTER INDEX *index_name* REBUILD;**

- 예를 들어 IDX_EMP01_DEPTNO를 재생성해보자.

형식 **ALTER INDEX IDX_EMP01_DEPTNO REBUILD;**

6. 인덱스 종류

- 인덱스는 다음과 같이 구분할 수 있다.

1. 고유 인덱스(Unique Index)
2. 비 고유 인덱스(NonUnique Index)
3. 단일 인덱스(Single Index)
4. 결합 인덱스(Composite Index)
5. 함수 기반 인덱스(Function Based Index)

- 이들 인덱스에 대해서 예를 들어 하나씩 살펴보도록 하자.

6.1 고유 | 비 고유 인덱스(1)

- 비 고유 인덱스를 이해하려면 고유 인덱스와 비교해 보아야 한다.
- 고유 인덱스(유일 인덱스라고도 부름)는 기본키나 유일키처럼 유일한 값을 갖는 컬럼에 대해서 생성하는 인덱스이다.
- 반면 비고유 인덱스는 중복된 데이터를 갖는 컬럼에 대해서 인덱스를 생성하는 경우를 말한다.
- 우리가 지금까지 사용한 인덱스는 비고유 인덱스이다.
- 고유 인덱스를 설정하려면 UNIQUE 옵션을 추가해서 인덱스를 생성해야 한다.

형식

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name);
```

6.1 고유 | 비 고유 인덱스(2)

- 부서 테이블에 다음과 같은 데이터가 존재한다면 DEPTNO 컬럼과 LOC 컬럼에 대해 고유와 비고유 인덱스 중 어떤 것을 지정할 수 있는지 살펴보도록 하자.

예 **SELECT * FROM DEPT01;**

DEPTNO	DNAME	LOC
10	인사과	서울
20	총무과	대전
30	교육팀	대전

- LOC 컬럼에는 중복된 지역 명이 저장되어 있으므로 고유 인덱스로 설정할 수 없고 비 고유 인덱스만 설정할 수 있다.
- DEPTNO 컬럼에는 부서번호가 중복되어 저장되어 있지 않고 유일한 값만을 갖고 있으므로 비 고유 인덱스는 물론 고유 인덱스도 설정할 수 있다.

6.2 고유 | 비 고유 인덱스 정의하기(1)

- ex) 고유 인덱스와 비 고유 인덱스를 비교하기 위해서 중복된 데이터가 없는 컬럼(DEPTNO)과 있는 컬럼(LOC)으로 구성된 부서 테이블을 만든다.
- 1. 부서 테이블을 생성한다.

예

```
DROP TABLE DEPT01;
CREATE TABLE DEPT01
AS
SELECT * FROM DEPT WHERE 1=0;
```

- 2. 다음과 같은 데이터를 입력한다.

예

```
INSERT INTO DEPT01 VALUES(10, '인사과', '서울');
INSERT INTO DEPT01 VALUES(20, '총무과', '대전');
INSERT INTO DEPT01 VALUES(30, '교육팀', '대전');
```

6.2 고유 | 비 고유 인덱스 정의하기(2)

- 3. 고유 인덱스를 지정하려면 UNIQUE 옵션을 지정해야 한다. 다음은 부서 테이블의 DEPTNO 컬럼을 고유 인덱스로 지정하는 예이다.

예

```
CREATE UNIQUE INDEX IDX_DEPT01_DEPTNO
ON DEPT01(DEPTNO);
```

- DEPTNO 컬럼에는 부서번호가 중복되어 저장되어 있지 않고 유일한 값만을 갖고 있으므로 비 고유 인덱스는 물론 고유 인덱스도 설정할 수 있다.
- 4. UNIQUE 옵션을 지정했는데 중복된 데이터를 갖는 컬럼을 인덱스로 지정하면 오류가 발생한다.

예

```
CREATE UNIQUE INDEX IDX_DEPT01_LOC
ON DEPT01(LOC);
```

```
명령의 1 행에서 시작하는 중 오류 발생 -
create UNIQUE INDEX IDX_DEPT01_LOC
ON DEPT01(LOC)
오류 보고 -
ORA-01452: cannot CREATE UNIQUE INDEX; duplicate keys found
01452. 00000 -  "cannot CREATE UNIQUE INDEX; duplicate keys found"
*Cause:
*Action:
```

6.2 고유 | 비 고유 인덱스 정의하기(3)

```
명령의 1 행에서 시작하는 중 오류 발생 -
create UNIQUE INDEX IDX_DEPT01_LOC
ON DEPT01(LOC)
오류 보고 -
ORA-01452: cannot CREATE UNIQUE INDEX; duplicate keys found
01452. 00000 -  "cannot CREATE UNIQUE INDEX; duplicate keys found"
*Cause:
*Action:
```

- 5. 중복된 데이터가 저장된 컬럼을 인덱스로 지정할 경우 비고유 인덱스로 저정해야 한다. 비고유 인덱스 는 UNIQUE 옵션을 생략한 채 인덱스를 생성하면 된다.

예

```
CREATE INDEX IDX_DEPT01_LOC
ON DEPT01(LOC);
```

6.3 결합 인덱스

- 지금까지 생성한 인덱스들처럼 한 개의 컬럼으로 구성한 인덱스는 단일 인덱스이다.

- 두 개 이상의 컬럼으로 인덱스를 구성하는 것을 결합 인덱스라고 한다.

예

```
CREATE INDEX IDX_DEPT01_LOC
ON DEPT01(LOC);
```

6.4 결합 인덱스 정의하기

- 부서 번호와 부서명을 결합하여 결합 인덱스를 설정해보자
- 1. 다음과 같이 부서 번호와 부서명을 결합하여 인덱스를 설정할 수 있는 데 이를 결합 인덱스라고 한다.

예

```
CREATE INDEX IDX_DEPT01_COM
ON DEPT01(DEPTNO, DNAME);
```

- 2. 데이터 딕셔너리인 USER_IND_COLUMNS 테이블에서 IDX_DEPT01_COM 인덱스는 DEPTNO와 DNAME 두 개의 컬럼이 결합된 것임을 확인할 수 있다.

예

```
SELECT INDEX_NAME, COLUMN_NAME
FROM USER_IND_COLUMNS
WHERE TABLE_NAME = 'DEPT01';
```

6.5 함수 기반 인덱스

- 검색조건으로 WHERE SAL = 300이 아니라 WHERE SAL*12 = 3600와 같이 SELECT 문 WHERE 절에 산술 표현 또는 함수를 사용하는 경우가 있다.

- 이 경우 만약 SAL 컬럼에 인덱스가 걸려 있다면 인덱스를 타서 빠르리라 생각 할 수도 있지만 실상은 SAL 컬럼에 인덱스가 있어도 SAL*12는 인덱스를 타지 못한다.

- 인덱스 걸린 컬럼이 수식으로 정의 되어 있거나 SUBSTR 등의 함수를 사용해서 변형이 일어난 경우는 인덱스를 타지 못하기 때문이다.

- 이러한 수식으로 검색하는 경우가 많다면 아예 수식이나 함수를 적용하여 인덱스를 만들 수 있다. SAL*12로 인덱스를 만들어 놓으면 SAL*12가 검색 조건으로 사용될 시 해당 인덱스를 타게 된다.

6.6 함수 기반 인덱스 정의하기

- 사원 테이블에서 급여 컬럼에 저장된 데이터로 연봉을 인덱스로 지정하기 위한 산술 표현을 인덱스로 지정해보자.
- 1. 함수 기반 인덱스를 생성한다.

예

```
CREATE INDEX IDX_EMP01_ANNSAL
ON EMP01(SAL*12);
```

- 2. 다음은 데이터 딕셔너리인 USER_IND_COLUMNS에 함수 기반 인덱스가 기록 되어 있는 것을 확인하기 위한 쿼리문이다.

예

```
SELECT INDEX_NAME, COLUMN_NAME
FROM USER_IND_COLUMNS
WHERE TABLE_NAME = 'EMP01'
```

INDEX_NAME	COLUMN_NAME
IDX_EMP01_JOB	JOB
IDX_EMP01_DEPTNO	DEPTNO
IDX_EMP01_ANNSAL	SYS_NC00009$