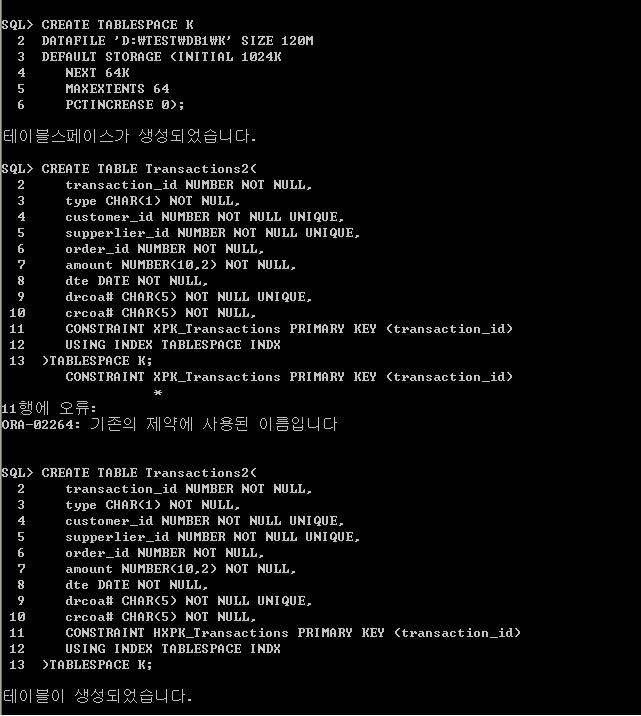
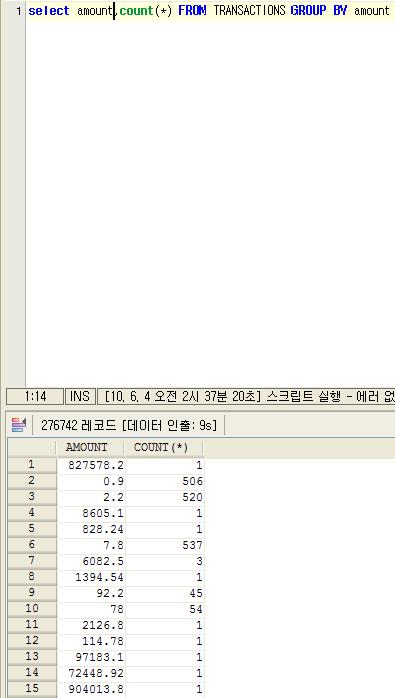
1) 실험 Table: 실험을 위해 사용한 Table에 대한 스키마 정의 및 데이터 생성

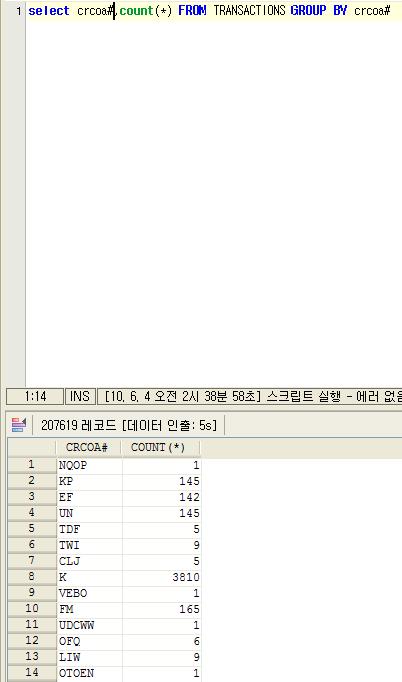
① Table의 생성문 정의

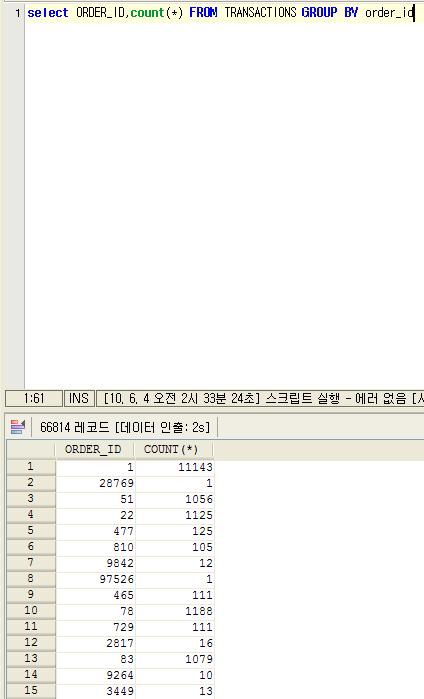


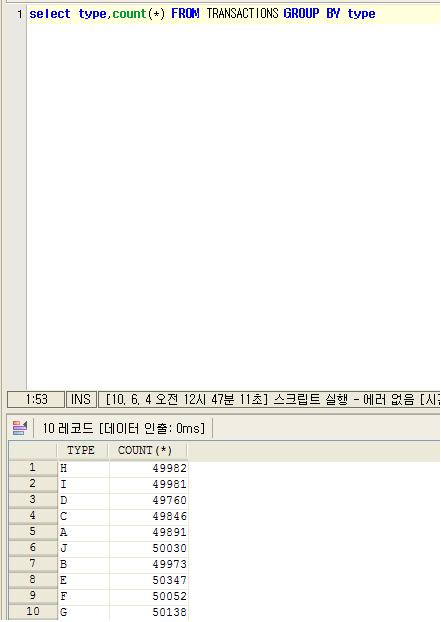
② Table data의 구성: tuple의 구성을 count를 통해 출력 하여 종류별 tuple 개수를 파악 할 수 있게 함



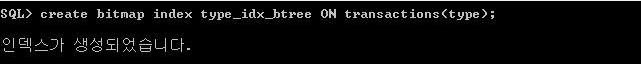








2) Index 생성 : 실험별로 필요한 Index 생성



3) 실험 Query 지정 : 각 실험에서 사용할 Query들의 정의와 설명.

Size 50% Size를 위한 Table 생성

CREATE TABLE transactions50 AS

SELECT \* FROM transactions

SAMPLE(50);

Size 25% Size를 위한 Table 생성

CREATE TABLE transactions25 AS

SELECT \* FROM transactions

SAMPLE(25);

Size 10% Size를 위한 Table 생성

CREATE TABLE transactions10 AS

SELECT \* FROM transactions

SAMPLE(10);

기본 100% Size의 성능을 알기 위한 Query문

Select count(\*) From Transactions;

50%의 Size의 성능을 알기 위한 Query문

Select count(\*) From Transactions;

25%의 Size의 성능을 알기 위한 Query 문

Select count(\*) From Transactions;

10%의 Size의 성능을 알기 위한 Query 문

Select count(\*) From Transactions;

B+Tree의 성능을 알기 위한 Query문

Select \* From Transactions Where Transaction\_id = 500000;

보통의 Index를 걸지 않는 다중의 값의 성능을 알기 위한 Query문

Select \* From Transactions Where Type=’J’;

Bitmap Index를 걸은 다중의 값의 성능을 알기 위한 Query문

Create Bitmap Index type\_idx\_btree ON transactions(type);

Select \* From Transactions Where Type=’J’;

1) 실험 과정을 상세하게 설명



1.위에 그림과 같이 Query에 따른 시간을 분석 할 수 있는 Setting을 해 놓는다.

1)Size에 따른 성능의 차이

1.Size에 따른 성능 분석을 하기 위해서 기본 테이블의 Query의 값을 날려서 성능에 값을 알아낸다.

2.Size가 50%에 따른 성능을 분석하기 위해서 Sample(50)을 이용한 Table생성을 하여서 성능에 따른 값을 알아낸다.

3.Size를 25% 10%에 따라서 위와 같은 실험을 반복하여 성능에 따른 값을 알아낸다.

4.값을 표와 그래프를 통해서 수치화 하여서 분석한다.

5.예측한 값과 실험 값의 결과를 비교한다.

2)Index에 따른 성능의 차이

1.Index에 따른 성능을 분석하기 위해서 FullScan을 위한 값을 위에 있는 Size에 따른 성능의 차이의 값을 가지고 온다.

2.처음으로 B-Tree에 Index를 걸려있는 값을 분석하기 위해서 UNIQUE한 제약사항이 걸려져 있는 값의 Query를 날려서 결과값을 알아낸다.

3.그 다음으로 BITMAP Index와 비교를 하기 위해서 Type에 따른 비교를 할 수 있는 아무것도 걸지 않는 Query를 날려서 결과값을 알아낸다.

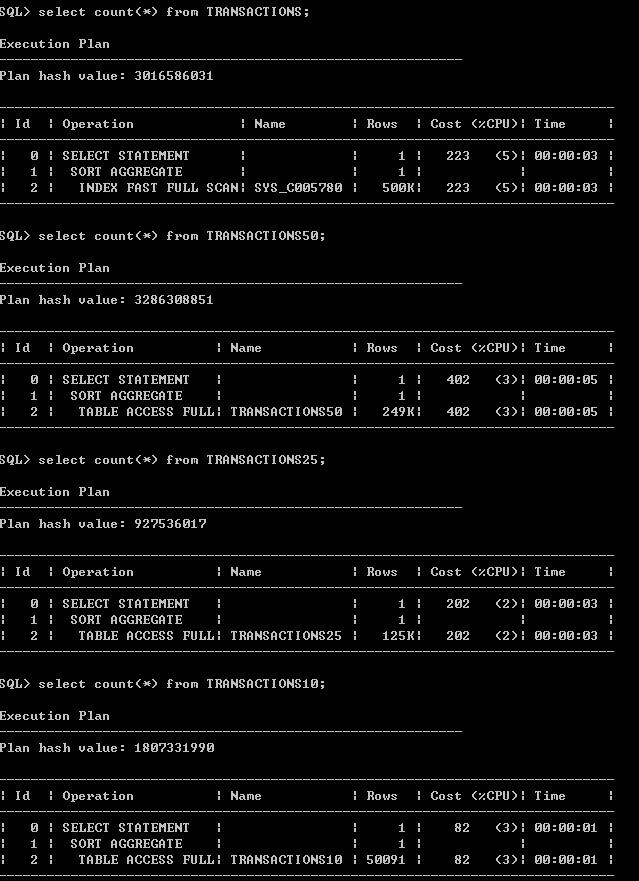
4.BITMAP INDEX를 걸은 후에 type에 따른 3.와 동일한 Query를 날려서 결과 값을 알아낸다.

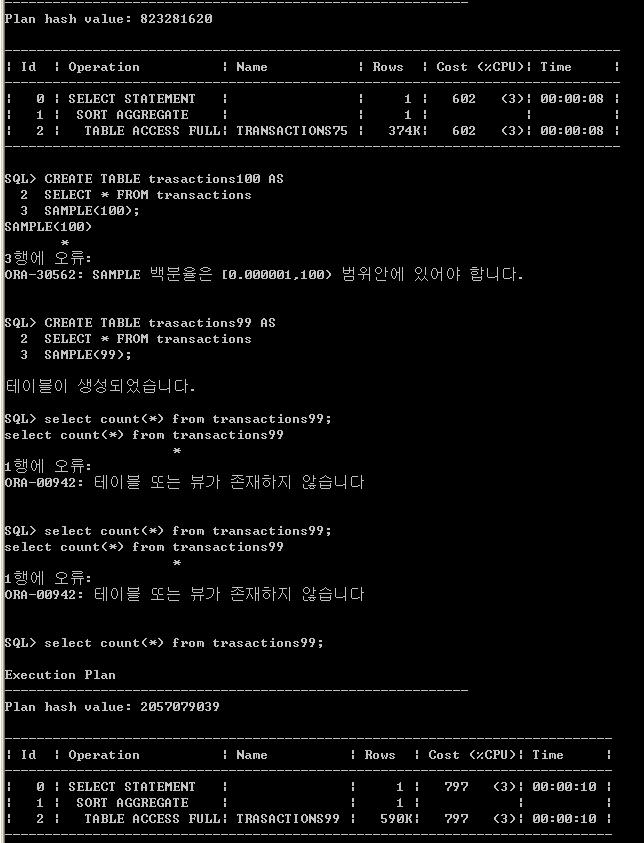
5.값을 표와 그래프로 수치화 하여서 분석한다.

6.예측한 값과 실험 값의 결과를 비교한다.

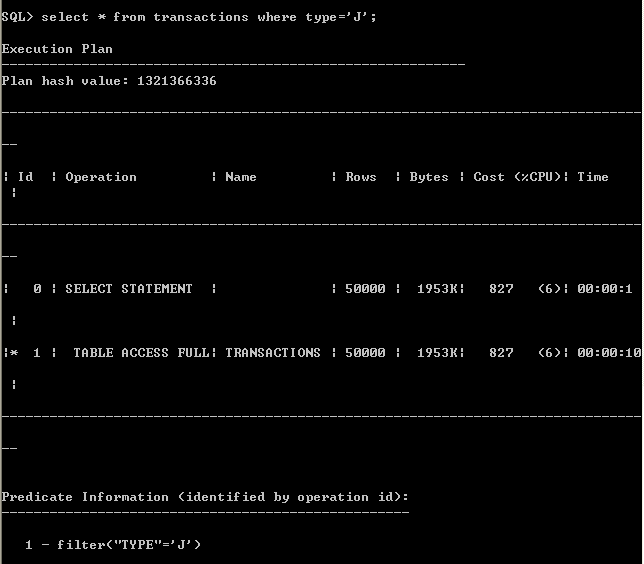
2) 다양한 실험 환경 변화에 따른 실험

① Table size의 변화에 따른 실험(500,000 tuples, 250,000 tuples, 125,000 tuples, 50,000 tuples)

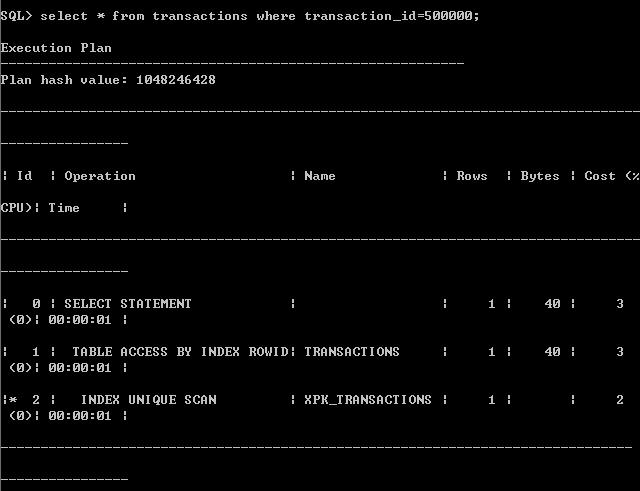




② Point Query(단일 Tulple 검색) 와 Mult-Point Query(검색 결과 복수 Tulple 검색)에 따른 실험







4. 위 3번의 다양한 실험을 통한 결과 분석 및 설명

1) 그래프 및 표를 사용하여 수치를 통한 결과 분석

|  |  |  |  |
| --- | --- | --- | --- |
| Percentage Value | %CPU | cost | Time |
| Transactions | 5 | 223 | 0:00:03 |
| Transactions99 | 3 | 797 | 0:00:10 |
| Transactions75 | 3 | 602 | 0:00:08 |
| Transactions50 | 3 | 402 | 0:00:05 |
| Transactions25 | 2 | 202 | 0:00:03 |
| Transactions10 | 1 | 82 | 0:00:01 |

|  |  |  |  |
| --- | --- | --- | --- |
| Percentage Value | Time | cost | %CPU |
| select \* from transactions where type='j'; | 0:00:10 | 827 | 6 |
| select \* from transactions where type='j';(Bitmap) | 0:00:01 | 1 | 0 |
| select \*from transactions where transaction\_id=500000;(B+tree) | 0:00:01 | 40 | 3 |

2) 실험결과에 대한 직관적 이유 및 분석결과 설명

1.Size에 따른 분석에 따른 성능의 분석

처음에 기존의 Table을 가지고 성능을 값을 분석하고, 50%, 25%,10%의 성능의 값을 분석을 하였다 하지만 기존의 테이블 보다 50%의 성능이 낮게 나오자 당황하게 되었다. 그래서 75%를 Sample하고 99%의 Sample을 가지고 테이블을 생성을 하여서 분석을 하였다. 기존의 테이블은 ACCESS FAST FULL스캔을 하는 반면에 Sample을 한 Table은 그냥 Full Scan을 하는 것을 그때에 발견을 할 수 가 있었다. 그래서 99%부터 10%값을 분석을 한 결과 Size가 작은 것이 성능도 좋게 나타난다는 것을 알 수가 있었다. 왜냐하면은 검색을 하는 범위가 적게 되면은 성능도 좋아지는 것이 당연하기 때문이다. 하지만 기존의 테이블을 통해서 얻을 수 있는 것은 검색하는 방법에 따라서도 성능이 Size와 비슷하게 할 정도로 성능이 나올 수도 있다는 것도 알게 되었다.

2.Point Query단일 tuple, 복수 tuple에 따른 실험

위에 있는 전체를 검색하는 것과 하나의 단일 tuple을 비교하고, 복수의 tuple을 비교할 때는 BITMAP INDEX의 우무를 통해서 분석을 하였다. 전체를 검색을 하는 것 보다는 단일 tuple을 검색하는 것이 빨랐다. 앞의 이유는 B-Tree를 통해서 빠르게 검색을 해 나갔기 때문에 가능한 것이라고 생각이 든다. 범위를 줄여나가면서 거맥의 범위를 100에서 50으로 절반씩 줄여나가기 때문에 검색이 빨라질 것이고 마지막 Leaf노드에 도달하고 인덱스 걸린 값을 얻어 낼 수가 있을 것이라고 생각이 든다.

다음으로 BITMAP인덱스를 걸었을 경우에는 걸지 않는 경우보다 걸은 경우가 10배의 시간의 이 줄은 것을 알 수가 있었다. BITMAP과 같은 경우에는 INDEX를 걸었을 때에 비트를 가지고 빠르게 AND연산을 통해서 Tuple을 검색하여서 Query에 대한 결과를 빠르게 나오게 되었던 것이 10배의 빠른 시간이라는 성능 향상을 가져온 것 같다.