# 데이터웨어하우스를 위한 데이터베이스 기술 소개

성균관대학교 이상원

## 1. 서 론

1990년대 초반까지 조직의 전산화는 비즈니스 프로세스를 자동화하는 OLTP(On-Line Transaction Processing) 중심의 운영계 시스템 구축으로 대표된다. 90년대 중반 들어 어느 정도 운영계 시스템의 구축이 이루어지고 성숙 단계에 접어들면서, 운영계 시스템에서 생성되는 데이터를 온라인으로 분석하는 (OLAP: On-Line Analytical Processing) 방안을 모색하게 되어서, 데이터웨어하우스(Data Warehouse, 이하 DW)라는 개념이 태동하게 되었다[1]. 조직 내의 데이터를 분석해서 의사결정에 활용하려는 개념 자체는 이미 1970년대부터 경영정보 시스템(MIS) 등에서 추구했던 개념이다. 그런데, DW가 지난 1990년대부터 많은 관심을 끌고 활발히 도입된 데는 몇 가지 이유가 있는데, 기술적인 측면에서는 DW에서 요구하는 대용량 데이터의 저장과 복잡한 임의적 분석을 가능하게 하는 하드웨어와 소프트웨어, 예를 들어 고성능 병렬 컴퓨터, 저렴한 대용량 하드디스크 기술, 대용량 데이터베이스 기술, 그리고 다양한 OLAP 및 데이터마이닝 도구들 등의 기술적 뒷받침이 DW의 번성을 가능하게 했다.

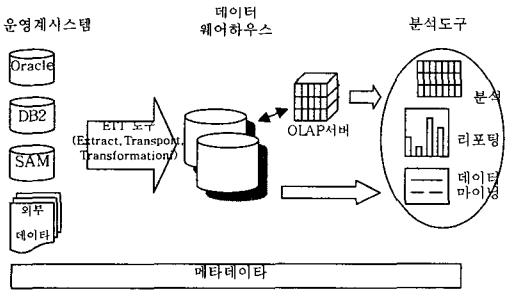


그림 1 데이터웨어하우스 아키텍처

그림 1은 DW 구축의 일반적인 아키텍처를 보여준다[2]. DW는 조직에서 분석에 필요로 하는 모든 데이터를 분석에 용이한 형태로 모델링해서 저장하고 있는데, 대용량의 데이터 처리에 적합한 관계형 DBMS가 주로 이용된다. 운영계 시스템에서 데이터를 추출(extraction), 변환(transformation)해서 DW로 적재(transportation)를 위한 도구들(이들을 통칭해서 ETT 도구라 하겠다.)이 사용된다. 최종 사용자가 별도의 프로그래밍 없이 다차원적으로 보고서 작성, 임의적인 분석, 데이터 마케팅, 데이터마이닝 등의 작업을 가능하게 해주는 다양한 최종 사용자 도구들이 포함된다. 최종 사용자 도구에서 제공하는 다차원의 OLAP 질의를 효과적으로 지원하기 위해 해당 도구와 DW를 저장하고 있는 관계형 데이터베이스 관리 시스템(database management system, 이하 DBMS)사이에 OLAP 서버가 존재할 수도 있다. 마지막으로, DW 아키텍처는 메타데이터(metadata) 관리를 위한 저장소(repository)를 포함하고 있다. 메타데이터는 의미적으로 "데이터에 관한 데이터"로, DW에서는 많은 종류의 메타데이터를 필요로 한다. 예를 들면, 운영계 시스템으로부터 데이터 추출, 변환 과정의 규칙, DW에서 모델링한 차원들과 각 차원의 계층구조 정보, 최종 사용자의 비즈니스 관점의 용어와 DW 내의 개념사이의 매핑 정보, 사용자 권한 정보 등을 들 수 있다.

그런데, 지난 10년간 하나의 분명한 기술적 추세는 그림 1에서 OLAP 도구나 ETT 도구들에서 수행하는 기능들이 점점 더 DBMS 안으로 흡수되고 있다는 점이다. 본 글에서는 이런 측면에서 최근에 DBMS에서 DW 분야를 위해 도입된 기술들을 소개하고자 하는데, 이 분야에서 가장 광범위하게 기술을 도입한 오라클(Oracle) DBMS를 중심으로 기술하겠다.

### 1.1 글의 구성

이 글은 최근 5년 사이에 오라클 DBMS에 도입된 기술들을 SQL의 OLAP 분석 기능, DW 성능 향상 기법, 그리고 ETT 관련 기능들을 중심으로 소개하고자 한다. 우선 2장에서는 기존의 OLAP 툴들이 제공하던 다차원 분석 기능들을 DBMS 내에서 지원하기 위한 SQL의 확장 내용을, 3장에서는 DW 질의 처리의 성능 향상을 위해 DBMS 내부에 도입된 기술들을 알아보고, 4장에서는 ETT 도구의 기능을 지원하기 위해 DBMS가 최근에 새로이 도입한 기능들을 소개한다. 마지막으로, 5장에서는 결론과 간단한 향후 기술 전망에 대해 알아본다.

## 2. 데이터 분석을 위한 SQL의 확장

현재의 관계형 데이터베이스의 표준 언어인 SQL의 모태는 관계형 DBMS의 프로토타입을 최초로 만든 IBM사의 Sequel이라는 언어에서 유래했다. 그리고, 이 Sequel이라는 언어는 이론적으로 Edgar F. Codd 박사가 1970년에 제안한 관계형 데이터 모델을 기반으로 일반 사용자들이 자연에 가까운 언어로 쉽게 사용하기 위해 만들어졌다. 그런데, 초기에 관계형 데이터 모델을 제안할 당시에는 데이터 분석보다는 단순한 데이터 저장, 검색, 조합에 초점이 맞추어졌었다. IBM에서 Sequel언어를 고안할 때(1975년 경) 목적은 관계형 연산자를 쉽게 표현하기 위해서 SELECT-FROM-WHERE 구조를 갖게 되었다. 다만, IBM의 Sequel 개발팀은 이 구조에다 group by와 aggregate 함수를 추가해서 간단한 데이터 통계치를 구할 수 있는 기능을 첨가했고, 또한 order by 구문을 통해서 결과의 순서를 제어할 수 있게 했다. 결과적으로 보통 사용자들이 익숙한 전형적인 SQL의 구조는 다음과 같다.

```
SELECT select-list
FROM table-list
WHERE conditions
GROUP BY grouping-column-list
HAVING having-condition
ORDER BY ordering-column-list;
```

그렇다면 이러한 구조의 SQL은 데이터 분석에 적합한가? 결론부터 말하자면, 초창기의 SQL은 데이터 분석을 위주로 설계되지 않았고, 결국은 데이터 분석 측면에서 아주 초보적인 간단한 질문의 경우에도 이를 SQL로 쉽게 표현하고 DBMS가 내부적으로 효과적으로 처리하기가 힘들다. 이와 같은 SQL의 결점은 DW 분야, 특히 OLAP용 분석 질의에서 극명하게 드러나기 시작했다. 이 OLAP 분야를 효과적으로 지원하는데 필요한 SQL 기능은, 단순한 레코드의 삽입/삭제/변경, 그리고 단순한 조건에 의한 레코드 검색 위주의 OLTP(Online Transaction Processing)에서 필요로 하는 SQL 기능과는 전혀 다르다.

SQL이 데이터 분석 측면에서 기능이 부족하면 어떤 문제가 발생하게 되나? 앞서 주지한 바와 같이, 비즈니스 측면에서 아주 간단한 질의에 대해서, SQL에 꽤 익숙한 개발자라 하더라도, 해당 질의를 SQL로 표현하기 힘들거나 어떤 경우에는 SQL 문으로는 해결하기 힘든 경우도 있다. 이는 결국 DB 프로그램 개발과 유지관리 업무의 생산성을 저하시키게 된다. 또한, 일종의 편법(trick)으로 SQL을 작성한다 하더라도 DBMS에서 이를 효과적으로 구하기 위한 실행 계획을 찾아내기 힘들기 때문에, 해당 질의의 수행 속도가 느리게 된다. 또한, Relational OLAP(ROLAP) 도구에서는 데이터 소스인 DBMS의 분석 기능이 약하기 때문에 자연적으로 도구에서 처리해야 할 부담이 커지고, DBMS와 많은 양의 데이터를 자주 주고받게 되는 단점이 생긴다. 이와 같은 데이터 분석 측면의 SQL의 문제점을 해결하기 위해서, 미국의 산업계/학계/연구소를 중심으로 다음과 같은 단계를 거치면서, 급격한 SQL의 발전이 이루어진다.

- 1단계 - 산업체에서의 요구 사항(1995년 경): Ralph Kimball이라는 DW 분야의 대가가 그 당시의 SQL의 문제점을 제시하고, 이의 해결을 촉구했다[3].
- 2단계 - 연구소/학계에서 해결 방안 제시(1996년 경): 1단계에서 제시된 문제 해결을 위해 SQL에 추가될 본질적인 기능을 MS 연구소의 Jim Gray[4], 컬럼비아 대학교의 Chatziantoniou와 Ken Ross[5], 그리고 RedBrick DBMS에서 제안했다.
- 3단계 - 상용 DBMS와 SQL3 표준에 반영(1998~2000년 경): 오라클과 IBM에서 해당 기능들을 제공하고, 이를 SQL3의 표준으로 반영했다[6].

그럼 DW 초창기의 분석 기능 측면에서 SQL의 본

질적인 문제점은 무엇인가? 크게 다음 세 가지 분석 기능, 즉 다중 집합화(multiple aggregations), 리포팅(reporting), 그리고 비교(comparisons) 기능 측면에서 문제점을 보인다.

여기서는 각각의 문제가 무엇인지 그리고, 왜 기존 SQL이 문제인지, 그리고 이 문제를 새로운 SQL로 어떻게 표현하는지 살펴보자. 이 절에서 사용하는 SQL의 예는 다음 sales 테이블을 대상으로 설명하겠다. 이 테이블에서 다차원 모델링[7] 방식으로 표현된 스키마를 갖는데, prod, cust_id, year은 일종의 차원(dimension)으로써 각각 상품, 고객, 년도를 의미하고, qty는 판매량을 의미한다.

| prod | cust_id | year | qty |
|---|---|---|---|

그림 2 sales 테이블

## 2.1 다중 집합화(Multiple Aggregations)와 큐브(Cube) 연산자

분석 측면에서 SQL의 본질적인 문제점 중의 하나는 group by가 단일 레벨의 그룹만을 정의한다는 것이다. 예를 들어, sales 테이블을 대상으로 다음과 같은 형태의 보고서를 원한다고 가정하자.

| | T1 | T2 | 부분합 |
|---|---|---|---|
| P1 | 10 | 20 | 30 |
| P2 | 30 | 40 | 70 |
| 부분합 | 40 | 60 | ***100*** |

그림 3 cross tabulation 보고서

이 보고서에서 검은색 수치는 상품/시간별, 상품별, 시간별, 그리고 전체 판매 수량을 나타내는 보고서이다. OLAP과 엑셀과 같은 툴에서는 cross- tabulation 기능이라 해서 데이터 분석시에 자주 사용되는 기능이다. 그렇다면, SQL로서는 이 값들은 어떻게 구할 수 있을까? 독자들이 쉽게 생각할 수 있듯이, sales 테이블을 대상으로 다음과 같이 4개의 별도의 SQL 문장을 이용하면 값을 구할 수 있다. 즉, 4개의 서로 다른 집합화 레벨(multiple aggregations)을 대상으로 group by를 이용한 별도의 SQL 문을 작성해야 한다는 것이다.

```
SELECT prod, year, sum(qty)
FROM sales
GROUP BY prod, year;

SELECT prod, sum(qty)
FROM sales
GROUP BY prod;

SELECT year, sum(qty)
FROM sales
GROUP BY year;

SELECT sum(qty)
FROM sales;
```

그런데 다중 집단화를 필요로 하는 분석 기능을 이와 같은 방식으로 처리하게 되면 다음과 같은 문제가 있다. 첫째, 4개의 SQL 문이 별도로 수행되기 때문에, 각 SQL 수행시마다 sales 테이블을 모두 스캔해야 한다. 이 예제에서는 sales 테이블이 작아서 별 문제가 안되는 것처럼 여겨질 수도 있으나, 테이블의 크기가 몇 백 메가 또는 몇 십 기가가 된다고 가정해 보라. 그리고 이 보고서는 단순히 prod, year 두 칼럼에 대한 cross-tabulation을 요구했지만, n개의 칼럼에 대한 보고서를 구한다고 하면, n! 개의 SQL을 작성해야 한다. 예를 들어, 3개의 칼럼에 대해 생각해 보면 3차원 큐브(cube) 모양이 떠오를 것이다. 둘째, SQL을 작성한 사용자는 이 질의가 어떤 용도로 작성되었는지를 쉽게 알 수 있지만, 제 3자가 이 SQL을 보고 그 목적을 쉽게 이해하기가 어렵다.

이와 같은 group by의 문제점을 해결하기 위해 - 즉 한번의 group by를 통해서 다중 집단화를 처리할 수 있도록 - Jim Gray[4]가 group by에 cube와 rollup 및 기타 관련 구문을 도입했다. 이를 이용하면 앞의 보고서의 결과는 다음 SQL 문장 하나로 구할 수 있다.

```
SELECT prod, year, sum(qty) total_qty
FROM sales
GROUP BY CUBE(prod, year);
```

위에서 GROUP BY CUBE(prod, year)는 2 칼럼의 4가지 조합 모두에 대해 group by 값을 구하라는 의미이다. 이와 같이 cube 기능을 이용한 SQL을 앞

의 4개의 SQL과 비교해 보면 얼마나 단순한가! 그리고 sales 테이블을 한번만 스캔(scan)하면서 원하는 결과를 구할 수 있다. 속도 면에서, 4개의 칼럼에 대해 생각해 보면, 대략 4! = 24배 만큼의 속도가 빨라지게 된다.

이 기능을 DBMS에서 지원하게 됨으로써 어떤 효과가 있나? 이전에는 ROLAP 툴 등에서는 분석가들의 사고에 편리한 다차원 분석을 위해서는 2차원 테이블에서 여러 번의 SQL 수행을 통해서 DBMS에서 필요한 정보를 전송 받았다. 하지만, 이 기능을 DBMS에서 지원함으로써 한번의 SQL 수행으로 큐브 생성에 필요한 데이터를 생성할 수 있게 되었다.

## 2.2 리포팅(Reporting), 비교(comparison) 등을 위한 분석 함수(analytic function)

다음과 같은 개념적으로 아주 간단한 질문을 어떻게 SQL로 표현할 것인지 생각해 보자. "sales 테이블에서 각 상품의 판매량의 전체 매출 대비 비율을 구하라." 이를 위해서는 논리적으로 SQL 사용자는 다음과 같이, 1) 우선 상품별 판매량의 합계를 구하는 뷰 v1를 정의하고, 2) 전체 상품의 판매량을 구하는 v2를 구하고, 3) 이 두 뷰를 조인해서 전체 판매량 대비 해당 상품의 판매량을 구하면 된다.

```
CREATE VIEW v1
AS SELECT prod, SUM(qty) qty_by_prod
FROM sales
GROUP BY prod;

CREATE VIEW v2
AS SELECT SUM(qty) total_qty
FROM sales;

SELECT v1.prod, v1.qty_by_prod/v2.total_qty ratio
FROM v1, v2;
```

위와 같은 방식으로 작성한 SQL의 문제점은, 결과를 구하기 위해, 논리적으로 두 단계 즉, 1) 상품별 판매량의 합계를 구하고, 2) 이 정보와 전체 판매량을 비교하는 단계로 생각을 해야 한다는 점이다. 이와 같이 작성하게 됨으로써 sales 테이블의 조인을 피할 수 없다는 점이다. sales 테이블의 크기가 아주 작을 때는 별 문제가 없지만, 테이블의 크기가 클 때, 관계형 DBMS의 조인(join)은 아주 비용이 많이 드는 연산이기 때문에, 가급적이면 피하는 게 좋다. 그러나 기존 SQL 방법으로는 이와 같은 분석 질의를 위해서 셀프조인(self-join)이 불가피하다.

이와 같은 SQL의 문제점의 본질적인 이유 역시 기존 group by 메커니즘에 있다. 앞의 질의는 개별 상품의 판매량과 전체 상품의 판매량을 비교하는 것인데, 하나의 group by는 개별 상품들의 판매량 합계 또는 전체 판매량만 구할 수 있다. 즉, 집단화 레벨(aggregation level)이 다른 개별 상품들의 판매량과 전체 판매량을 하나의 SQL 문에서 비교하는 메커니즘을 제공하지 않고 있다.

이와 같은 group by의 문제점을 해결하기 위해 - 즉 한 그룹의 집단화 값을 구하면서 소속 튜플들의 정보도 같이 병행해서 보여주기 위해서 - [5]에서 제안된 방법에 기반해서 분석 함수(analytic function)가 표준화된 방법으로 제안되었다[6]. 이를 이용하면 앞의 보고서의 결과는 다음과 같은 간단한 SQL 문장 하나로 구할 수 있다. 이 기능을 이용함으로써 1) 원하는 질의를 간단하게 표현할 수 있고, 2) 이 질의의 수행은 sales 테이블에 대한 셀프조인이 필요없기 때문에 수행속도가 훨씬 빠르게 된다.

```
SELECT year, ename, qty, sum(qty) over(partition by
year),
        ratio_to_report(qty) over(partition by year),
FROM sales;
```

데이터 분석에서 자주 사용되는 또 다른 핵심 개념은 비교(comparison)이다. 예를 들어, 비즈니스적으로 아주 간단한 다음 질문을 어떻게 SQL로 표현할 것인지 생각해 보자. "지난달 대비 제품별 판매량 변화를 구하라." 이 질의를 위해 sales 테이블을 대상으로 다음과 같은 SQL 문장으로 구할 수 있다.

```
SELECT s1.prod, s1.year, s1.qty  s2.qty
FROM sales s1, sales s2
WHERE s1.year = s2.year +1 AND s1.prod = s2.prod;
```

이 질의는 비교적 간단하게 SQL로 표현할 수 있다. 하지만, 이 질의 역시 결과를 구하기 위해서 셀프조인을 필요로 한다는 점이다. 이 문제 역시 하나의 테이블 내의 특정 튜플들끼리 비교하는 기능이 SQL

에서 제공되지 않기 때문에 발생하는 문제이다.

이와 같은 비교 기능의 문제점을 해결하기 위해, 앞서 언급한 분석 함수가 표준화된 방법으로 제안되었다. 이를 이용하면 앞의 질의는 다음과 같이 셀프조인을 필요로 하지 않는 SQL 문장으로 표현 가능하다.

```
SELECT prod, year, qty  lag(qty,1) over (partition by
prod order by year)
FROM sales
GROUP BY prod, year;
```

앞의 리포팅의 경우와 마찬가지로, 이 기능을 이용함으로써 1) 원하는 질의를 간단하게 표현할 수 있고, 2) 이 질의의 수행 역시 sales 테이블에 대한 셀프조인(self-join)이 필요가 없기 때문에 수행 속도가 훨씬 빠르다.

이외에도 분석 함수는 랭킹(ranking), 이동 평균(moving average), 퍼센트(percentile) 등의 아주 복잡하고 다양한 분석 기능을 SQL의 프레임워크에서 가능하게 해준다. 분석 함수의 종류와 시맨틱에 관해서는 [6]을 참고하기 바란다. 분석 함수의 의미는 이들 분석 기능은 원래 DBMS의 영역이 아니라 OLAP 도구의 영역에서 수행되던 작업을 DBMS 엔진 안으로 흡수하게 했다는 점이다. 앞으로 이 분석 함수에 기반해서 점점 더 많은 분석 기능들이 SQL에서 지원 가능하게 될 것이다. 따라서, 점점 더 OLAP 도구와 같은 제품들은 사용자 인터페이스 측면에서 치중하게 되고, 관계형 DBMS가 분석 플랫폼의 기능을 수행할 것이다.

### 2.3 스프레드쉬트와 SQL

독자들도 주지하다시피, 스프레드쉬트는 많은 사람들이 사용하는 가장 성공적인 데이터 분석 도구이다. 행(row)과 열(column)로 구성된 2차원 배열에 데이터를 입력하고 다음과 같은 다양한 조작을 수행할 수 있다.

- 2차원 데이터를 대상으로 분석에 필요한 공식(formula)을 정의
- 재귀적 모델(recursive model)을 지원하는 수식 지원
- 선택된 셀 또는 셀의 범위에 대해 분석 함수와 집단 함수(aggregate function) 제공

반면, 엑셀 이상으로 성공적인 관계형 DBMS의 대표 언어 SQL은 분석적인 기능 측면에서, 큐브(cube)와 분석 함수(analytic functions)의 도입에도 불구하고, 스프레드쉬트 도구에서 제공하는 다양한 분석 기능에 비교하면 부족한 부분이 있다. 즉, 2차원 배열을 대상으로 한 복잡한 계산을 표현하기가 (불가능하지는 않지만) 힘들고, 이를 대량의 데이터에 대상으로 수행하게 되면 비효율적으로 처리된다는 점이다.

그렇지만, 최근에는 오라클에서는 엑셀로 대표되는 스프레드쉬트의 분석 모델도 SQL에 흡수하기에 이르렀다[8]. 이를 위한 SQL의 확장과 처리 방안에 관한 자세한 내용은 대해서는 [8]을 참고하기 바란다. 다만, SQL에서 스프레드쉬트의 분석 모델을 지원함으로써(즉, SQL에 엑셀 기능을 흡수했을 때), 스프레드쉬트 기반의 데이터 분석 아키텍처에 어떤 변화가 생기게 되나? 현재의 경우 스프레드쉬트가 DBMS 밖에서 별도로 관리되는 파일이지만, 미래에는 그림 4에서 보여지는 것처럼, 스프레드쉬트는 일반 테이블처럼 관계형 데이터베이스 내에 존재하게 된다. 해당 스프레드쉬트의 데이터는 특수한 릴레이션 행태로 데이터베이스에 저장되고, 스프레드쉬트의 정의는 관계형 데이터베이스의 메타데이터 딕셔너리에 저장된다. 엑셀 등의 도구는 데이터 저장 및 분석을 위한 계산 기능을 DBMS에게 넘겨주고, 자신은 거의 사용자 인터페이스만 담당하는 역할을 하게 될 것이다.

## 3. 데이터베이스 성능을 위한 기술들

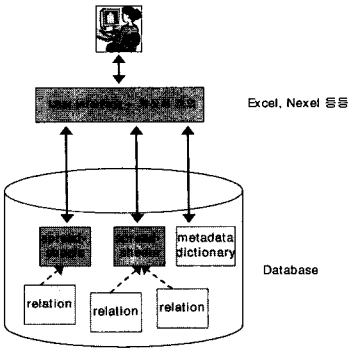


그림 4 엑셀과 관계형 DBMS 연동: 미래

지금까지는 DW의 분석 기능 측면에서 SQL에 도입된 획기적인 기술들을 살펴보았다. 이 절에서는 DW용으로 사용되는 관계형 DBMS에서 성능을 획기적으로 향상하기 위해 도입된 기술들을 오라클을 중심으로 간략히 알아보겠다. 이 절에서 다룰 주요한 기술들은 다음과 같다.

- 실체화 뷰(materialized view)
- 분할(partitioning)
- 비트맵 및 비트맵 조인 인덱스(bitmap and bitmap join index)
- 테이블 압축(table compression)
- Star 질의 최적화(star query optimization)
- 동적 SQL 메모리 관리(dynamic SQL memory management)

## 3.1 실체화 뷰(Materialized Views)

실체화 뷰는 DW 분야의 성능 향상을 위한 대표적인 기술로써[9], 일종의 인덱스와 유사한 역할을 수행한다고 보면 된다. 일반 뷰는 사용자로 하여금 질의를 간단히 작성할 수 있도록 해주는 역할을 할 뿐, 실제 질의 처리는 해당 뷰의 기본 테이블(base table)에 대한 질의로 재작성된 후 수행된다. 반면, 실체화 뷰는 해당 뷰의 정의에 해당하는 실제 내용이 디스크에 저장되어서 기본 테이블을 대상으로 한 질의에 대해 가능한 경우에 빠른 질의 처리를 위해 실체화 뷰로 재작성 해서 질의 처리를 수행한다.

예를 들어, sales 테이블에 다음과 같이 정의된 실체화 뷰가 존재하는 경우를 가정하자.

```
CREATE MATERIALIZED VIEW total_by_cust_id_prod AS
SELECT cust_id, prod, sum(qty)
FROM sales
GROUP BY cust_id, prod
```

이때 다음 질의를 처리하기 위해서는 기본 테이블 sales을 접근하는 대신에 위에서 정의한 실체화 뷰를 이용해서 대답이 가능하다. 위에서 year 칼럼이 10년치의 값을 갖는다면, 대략 실체화 뷰의 크기가 1/10이 될 것이기 때문에 (기본 테이블과 실체화 뷰 모두 인덱스가 없는 경우) 수행 속도가 10배 정도는 빠를 수 있다.

```
SELECT cust_id, sum(qty)
FROM sales
GROUP BY cust_id
```

이 실체화 뷰를 지원하기 위해서는 세 가지의 기술적인 과제가 해결되어야 한다. 우선 기본 테이블의 내용이 바뀔때 해당 실체화 뷰도 일관성 있게 그리고 효과적으로 재구성(MV refreshment)을 할 수 있어야 하며, 둘째 질의 최적화기가 어떤 질의에 대해 특정 실체화 뷰를 사용해서 결과를 구할 수 있도록 "질의 재작성"(query rewrite using MV)을 할 수 있어야 하며, 마지막으로 제한된 디스크 공간을 이용해서 최적화된 성능을 내도록 "생성할 실체화 뷰를 선택하는 문제"(MV selection)를 해결해야 한다. 현재 주요 관계형 DBMS들은 이들 이슈에 대해 충분히 실무에 적용할 수 있을 만큼 성숙한 기술들을 제공하고 있다.

## 3.2 분할(Partitioning)

주요 관계형 DBMS들은 분할 기능을 제공하는데, 이는 하나의 논리적인 테이블, 인덱스 등을 여러 개의 물리적인 개체들로 분할해서 저장함으로써, 성능 측면에서 커다란 장점을 제공하는 기능이다. 예를 들어, sales 테이블을 다음과 같이 년도별로 분할해서 정의했다고 하자. 이 경우 논리적으로는 하나의 테이블 sales가 존재하지만, 물리적으로는 10개의 파티션이 생성된다.

```
CREATE TABLE sales (prod number, cust_id
number, year number, qty number, PARTITION BY
RANGE(year))
(PARTITION sales_1994 VALUES LESS THAN
(1995),
.....
PARTITION sales_2003 VALUES LESS THAN
(2004))
```

이 경우 사용자는 논리적으로 sales 테이블만을 대상으로 삽입/삭제/갱신 및 질의를 수행하는데, DBMS에서 해당 자동적으로 관련 파티션을 대상으로 적당한 연산을 수행하게 된다. 예를 들어, 년도가 2003년도에 관한 질의를 수행하게 되면 10년 동안의 모든 데이터를 검색할 필요없이 2003년에 해당하는

파티션만을 대상으로 질의를 처리하면 되므로 대략 10배 정도의 성능 향상을 가져올 수 있다. 1994년도의 데이터가 필요가 없어서 삭제하고자 하는 경우 해당 파티션만을 제거하면 되므로 관리 측면(manageability)에서 훨씬 용이하고, 특정한 파티션에 대해 다른 관리 작업을 수행하는 경우에 나머지 파티션들에 대해서는 접근이 가능하기 때문에 이용 가능성(availability) 측면에서도 장점을 제공한다. 오라클 질의 최적화기는 분할을 활용하는 특징(partition-aware optimization)을 제공하고 있는데, 예를 들면 분할된 테이블에 대해 분할 키에 대한 정렬 연산을 수행할 때 개별 파티션 별로 정렬해서 합병하는 것이 더 효율적이다.

사용자 입장에서는 분할을 하는 기준이 얼마나 다양한가에 따라 이 기능의 유용성이 결정되는데, 오라클 DBMS의 경우 범위(range) 기반의 분할 이외에 해시(hash) 기반 분할, 리스트(list) 기반 분할, 복합(composite) 분할 등 다양한 분할 기능을 제공하고 있다. 이들 서로 다른 분할 기능을 데이터의 특성에 따라 성능이 달라지기 때문에 사용자가 자신의 데이터와 작업부하의 특징에 따라 적절하게 골라야 한다.

### 3.3 비트맵 및 비트맵 조인 인덱스(Bitmap & Bitmap Join Index)

비트맵 인덱스는 다차원 모델링에서 주로 차원 역할을 하는 애트리뷰트들, 예를 들어 나이, 성별, 지역, 상품처럼 전체 레코드 건수에 비해 카디널리티가 적은 애트리뷰트들을 대상으로 생성하고, 이들 차원 애트리뷰트에 대해 AND와 OR 등을 이용한 복잡한 조건이 주어졌을 때 효과적으로 사용할 수 있는 인덱스 구조이다. 같은 크기의 테이블에 대해 B+ 트리 인덱스에 비해서 훨씬 적은 인덱스 공간을 차지하고, 비트맵에 대해 AND/OR 연산을 효과적으로 계산할 수 있기 때문에 DW 질의에서 빠른 성능을 보장한다.

오라클 9i에서는 이 비트맵 인덱스를 한 단계 더 발전시켜서 비트맵 조인 인덱스 기능을 제공하고 있다. 이는 테이블들 사이의 조인 결과에 비트맵 인덱스를 생성하는 것으로 질의 처리를 위한 조인을 피함으로써 월등한 성능 향상을 제공한다. 그림 5는 sales 테이블과 차원 테이블인 customer 테이블을 대상으로 비트맵 조인 인덱스를 생성하는 예를 보이고 있다.

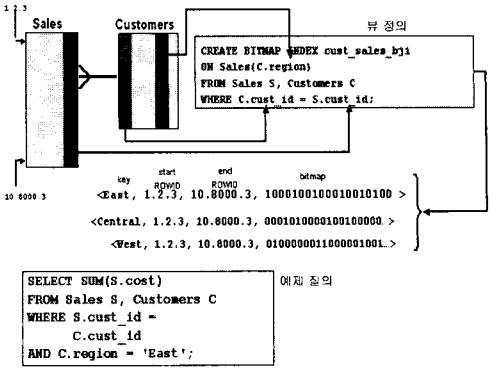


그림 5 비트맵 조인 인덱스 예

그림 5에서 CREATE BITMAP INDEX 문은 customer 테이블의 region 칼럼을 대상으로 sales 테이블에 대한 비트맵 인덱스를 생성하는 것이다. 예를 들어, "region = East"인 customer 튜플들과 조인하는 sales 튜플들에 대한 비트맵 인덱스를 생성하는 것이다. 이 비트맵 조인 인덱스는 CREATE BITMAP INDEX 문의 조건인 "C.cust_id = S.cust_id"를 통해서 미리 차원 테이블인 customer와 사실 테이블 sales 사이의 조인을 미리 계산하는(precomputation) 역할을 하게 된다. 이렇게 되면, 그림 5의 예제 질의의 경우 조인을 유발하지만, 실제로는 customer 테이블을 접근할 필요없고, 또한 sales 테이블과 customer 테이블의 조인도 필요없다. 단지, 비트맵 조인 인덱스를 통해서 sales 테이블을 접근하기만 하면 된다.

### 3.4 테이블 압축(Table Compression)

DW 분야는 다차원 데이터 모델링 특성상 3차 정규형에 위배되는 반정규형(denormalization)으로 데이터의 중복이 일반적으로 많이 발생한다. 따라서, 특정 칼럼에 대해 같은 값을 갖는 튜플들이 흔하다. 이는 결국 디스크 상에 같은 값을 갖는 내용이 중복적으로 저장되므로 인해 1) 디스크 공간의 낭비를 가져오고, 2) 이는 입출력 비용의 증가를 가져오고, 3) 제한된 크기의 버퍼 공간의 효용성을 저하시키는 결과가 된다.

이와 같은 데이터 중복의 문제점을 해결하기 위해서 오라클 DBMS에서는 최근에 관계형 데이터 특성을 활용해서 테이블과 실체화 뷰의 압축(compression) 기법을 제안했다[10]. 이 기법에 따르면, 압축의 단위는 블록이고, 같은 블록에 속하는 다른 튜플

들에 동일한 칼럼 값이 중복적으로 나타날 때 이를 심볼 테이블 형태로 구성해서 압축하는 방법을 취하고 있다. [10]에 따르면, 이 방법은 작게는 압축률(compression factor)이 2배에서 많게는 12배까지 보이고 있다.

이와 같이 압축된 데이터 프로세싱은 입출력 시스템을 통해 버퍼 캐시에도 압축된 형태로 올라오기 때문에 입출력의 비용을 줄이게 되고 버퍼 캐시에 더 많은 데이터를 보관할 수 있는 장점이 있다. 단, 개별 튜플들을 접근해야 하는 질의 처리기에서는 해당 블록내의 심볼 테이블을 참조하는 포인터를 통한 간접적인 데이터 접근의 오버헤드가 있다. 그렇지만, 질의 처리 성능은 입출력 비용의 획기적인 감소에 따라 TPC-H 벤치마크를 대상으로 한 실험에 따르면[10], 평균 15% 정도의 성능 향상을 보인다.

### 3.5 Star 질의 최적화(Star Query Optimization)

다차원 모델링의 경우, 주로 그림 6과 같은 스타 스키마(star schema) 형태를 갖는데 이는 하나의 사실 테이블(fact table, 이 경우 sales)과 이 사실 테이블이 참조하는 차원 테이블(dimension table, 이 경우 product, customer, year)들로 이루어진다.

이 스타 스키마를 대상으로 다음 형태의 질의가 주어졌다고 하자. 이때 sales 테이블의 각 차원 칼럼들을 대상으로 비트맵 인덱스가 정의되어 있다고 하자. 그리고, 질의에서 조건은 차원 테이블의 다른 칼럼을 대상으로 정의되어 있다. 이 경우 일반적인 질의 처리 방식으로는 비트맵 인덱스를 활용할 수 없고, 또한 중간 결과 테이블의 크기가 일반적으로 커지기 때문에 비효율적이다.

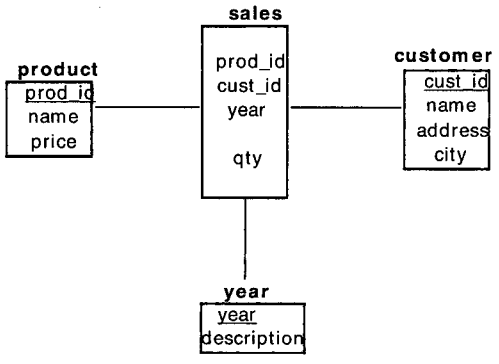


그림 6 스타 스키마 예

```
SELECT p.name, c.cust_address, y.description, SUM
(s.qty) sales_amount
  FROM sales s, year y, customer c, product p
  WHERE s.year = y.year
  AND   s.cust_id = c.cust_id
  AND   s.prod_id = p.prod_id
  AND   c.cust_address = 'CA'
  AND   p.name in ('A','B')
  AND   y.description IN ('THIS YEAR','LAST
  YEAR')
  GROUP BY p.name, c.cust_address, y.description;
```

그러나, 이 질의를 sales 테이블의 비트맵 인덱스들을 활용할 수 있는 질의 형태로 바꾸어서 수행하고, sales 테이블에 조건을 적용한 작은 크기의 결과를 관련된 차원 테이블과 조인을 함으로써, 1) 비트맵 인덱스의 성능과 2) 불필요한 조인의 수행 방지를 통해서 성능 향상을 달성할 수 있다. 자세한 예는 [11,12]를 참고하기 바란다.

### 3.6 SQL 실행 메모리 공간의 자동적인 관리

그림 7은 오라클의 SQL 실행 메모리 모델을 보여주고 있는데, 그림에서 SGA 영역은 주로 데이터베이스 전체에서 공유하는 영역(Shared Global Area)을 의미하고, 각 사용자 프로세스 별로 할당되는 PGA는 개별 세션 별로 사용하는 영역(Private Global Area)으로 이 PGA 부분 안에는 SQL 실행 메모리(SQL Execution Memory)가 포함된다. 이 메모리 모델은 IBM이나 MS SQL Server에서도 거의 동일하다고 보면 된다. 이 SQL 실행 메모리는 주로 SQL 실행시 필요한 정렬(sorting), 해시(hash), 비트맵 인덱스 관련 연산 등을 위한 공간이다. 그런데, 이전의 OLTP 성격이 강한 데이터베이스에서는 정렬이나 해시 데이터의 크기가 상대적으로 작았기 때문에 이 공간의 관리가 크게 중요하지 않았다. 하지만, 최근에 OLAP 성격의 대용량 데이터 분석용 애플리케이션에서는 정렬이나 해시 연산을 필요로 하는 SQL 연산이 많아지고, 각 SQL 문장별로 최적으로 수행하기 위해 필요한 PGA 영역 크기의 편차가 크기 때문에 이 SQL 실행 메모리의 효과적인 관리가 데이터베이스 전체 성능 관리에 큰 영향을 미치게 된다.

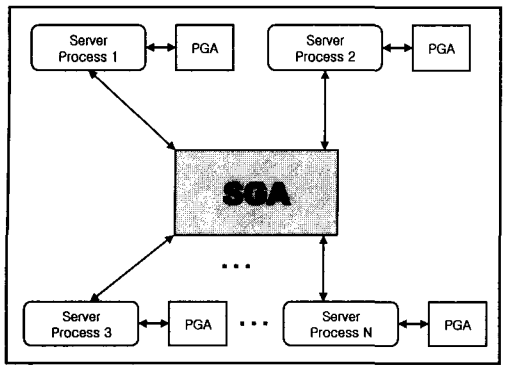


그림 7 SQL 실행 메모리 모델

단일 SQL을 수행할 때, 이용 가능한 메모리 량에 따른 정렬과 해시 연산의 수행 시간 사이의 관계는 어떤가? 먼저 정렬의 경우, 이용 가능한 메모리 공간이 정렬할 데이터 양보다 많을 경우, 최적의 수행 시간을 보이고, 메모리 공간이 조금 더 적을 경우에는 단일패스(1-pass) 내에 정렬이 가능하고, 더 적을 경우에는 메모리 양에 반비례해서 수행 시간이 점점 더 길어지게 된다. 또한, 해시 연산의 경우에도 수행 시간은 대체로 이용 가능한 메모리 양에 반비례하게 된다. 정렬이나 해시 연산에 대해 최적의 성능을 보장하기 위해 필요한 메모리 공간은 세션 별로, 그리고 동일한 세션 내에서도 수행되는 SQL 문에 따라 달라지게 된다. 그런데 지금까지 DBMS들은 일반적으로 이 SQL 작업 공간의 관리는 DBA가 특정한 값을 지정하게 되면 모든 세션에 대해 거의 고정적인 메모리를 할당하는 방식을 취했다. 그러므로, 어떤 세션은 필요 이상의 과다한 SQL 실행 메모리를 갖게 되고, 어떤 세션은 SQL 실행 메모리가 부족해서 최적의 성능을 보장할 수 없게 된다.

이런 단점을 극복하기 위해 최근에 오라클 9i에서는 자동적인 SQL 실행 메모리 관리 기술을 도입했다[13]. 이 기술에 따르면, 개별 세션 별로 고정된 크기의 메모리가 할당되는 것이 아니라, DBA가 PGA 영역에 대해 전체적으로 총량을 지정하게 되면, 개발 세션에서는 해당 세션 내에서 수행하는 SQL의 각 단계에서 각각의 수행 시점에 필요한 만큼의 SQL 실행 메모리만을 할당 받기 때문에, 제한된 메모리 공간을 최적으로 활용할 수 있게 된다. 즉, 특정 시점에 SQL 실행 메모리를 많이 필요로 하는 세션은 큰 메모리 공간을 할당 받고, 적은 SQL 실행 메모리를 필요로 하는 세션은 최소의 공간만을 사용함으로써 개별 SQL의 수행 시간(response time)도 단축하게 되고, 전체 시스템의 성능(throughput)도 향상될 수 있다[13]. 이와 같이 제한된 자원에 관한 자동적인 튜닝(self tuning)은 당분간 데이터베이스 기술의 중요한 흐름이 될 것이다.

## 4. ETT 지원을 위한 데이터베이스 기술들

OLAP 분석 기능과 DW 성능 향상 기술 이외에, 비교적 최근에 SQL과 데이터베이스 엔진에서 지원하기 시작한 중요한 기술들이 ETT 프로세스 지원을 위한 기능들이 있다[14,15]. 이를 통해서, 1절의 그림 1에서 기존의 ETT 도구들이 수행하던 역할의 상당 부분을 DBMS 엔진에서 흡수하게 된다. 이 절에서는 오라클을 중심으로 최근에 도입된 ETT 지원을 위한 다음 기능들을 간단히 알아보겠다. 참고로 이 절의 내용은 주로 [15]에 기반한다.

- 변경 데이터 인식(change data capture)
- 외부 테이블(external table)
- DML 연산의 확장
- 테이블 함수(table function)

### 4.1 변경 데이터 인식(change data capture)

DW 환경에서는 소스 데이터(source data)로부터 주기적으로(예를 들어, 1시간, 일주일 단위) 정보를 추출해서 DW로 적재를 해야 한다. 이때 매번 소스 데이터 전체를 대상으로 이 작업을 수행하는 것은 굉장히 비효율적이다. 따라서, 최근에 ETT를 수행한 이후에 소스에서 발생한 변경 내용만 추출해서 DW에 반영할 수 있어야 한다. 이를 변경 데이터 인식이라고 한다.

그런데 지금까지는 소스 시스템에서 변경 데이터 인식을 위한 특수한 애플리케이션을 별도로 작성해야 했다. Oracle 9i에서는 소스 시스템이 오라클 DB인 경우 이 변경 데이터 인식을 자동으로 수행할 수 있는 기능을 제공한다. 또한 소스 시스템이 오라클 DB가 아닌 경우에도 특수한 API를 제공하면 오라클 9i의 변경 데이터 인식 프레임워크와 연동할 수 있다.

### 4.2 외부 테이블(external table)

DW 환경에서는 ETT 과정, 특히 데이터 변형

(data transformation)과 적재(loading) 과정에서 데이터 무결성 검사 등의 이유로 외부 파일에 존재하는 데이터를 참조할 필요가 있다. 그런데, 이 외부 데이터를 변형과 적재 과정에서 SQL 언어에서 직접 사용하기 위해서는 이 데이터를 우선적으로 데이터베이스에 적재를 해야만 한다. 이 과정에서 적재를 위한 시간과 데이터베이스에서 공간을 차지하기 때문에 비효율적이다.

이를 위해 외부에 운영체제의 파일로 존재하는 데이터를 직접 데이터베이스에 적재하지 않고도, 마치 오라클 데이터베이스 내의 가상적인 테이블처럼 사용할 수 있는 기능이 외부 테이블이다. 이처럼 외부 파일을 외부 테이블로 데이터베이스에 등록만 하면, 마치 데이터베이스 내에 존재하는 일반 테이블과 마찬가지로 SQL에서 사용할 수 있다.

### 4.3 DML 연산의 확장

DW의 적재(loading) 과정에서는 발생하는 두 가지의 중요한 이슈는 1) 하나의 소스 데이터가 데이터 값에 따라 여러 개의 테이블로 나뉘어져 삽입되어야 하는 것과, 2) 특정 테이블에 삽입하는 과정에서 이미 어떤 키 값을 갖는 레코드가 존재하는 경우 삽입(insert)대신에 해당 레코드를 갱신(update)하는 방법이다. 기존의 SQL로는 각 이슈에 대해 한번의 SQL 수행으로 해결할 수가 없다. 따라서, 오라클 9i에서는 이 두 경우를 위해 다중 테이블 삽입(multi-table insert)과 갱신 삽입(upsert: update + insert) 기능을 지원하고 있다.

다음은 다중 테이블 삽입의 단순화된 예를 보이고 있는데, 하나의 소스 테이블을 한번만 스캔해서 개별 레코드의 조건에 따라 서로 다른 타겟 테이블에 삽입한다. 기존의 방법으로는 소스 테이블을 세 번 스캔해야만 가능하다.

```
INSERT
WHEN cond1 THEN
        INTO target_table1
ELSE cond2 THEN
        INTO target_table2;
ELSE
        INTO target_table3
SELECT * FROM source_table;
```

### 4.4 테이블 함수(table function)

일반적으로 DW의 ETT 과정에서 소스 데이터가 타겟 테이블에 적재되기 이전에 여러 단계의 복잡한 변환 과정을 거치게 된다. 그런데, 각 변환 과정의 중간 결과는 일반적으로 (메인 메모리에 크기보다는) 매우 크다. 그러면, 이 중간 결과를 디스크에 임시로 저장하고 다음 변환을 위해 또 읽어들이는 과정을 반복해야 한다. 따라서, 이 입출력 비용이 변환 과정을 굉장히 느리게 만든다. 이를 개선하기 위해 오라클에서 도입한 기능이 테이블 함수이다.

테이블 함수의 핵심 개념은 두 가지인데, 우선 각 중간 결과를 디스크에 쓰지 않고 다음 변환 과정을 위한 입력으로 파이프라인식으로 전달해 주고, 이렇게 다음 변환 단계를 병렬화할 수 있도록 이전 결과를 분할해서 다음 단계에 전달할 수 있다. 따라서, 중간 결과를 디스크에 기록하고 읽어들이는 비용을 없앨 수 있기 때문에 변환 과정을 훨씬 단축할 수 있다.

## 5. 결론 및 향후 전망

이상에서 우리는 지난 10년 정도에 걸쳐 DW 분야를 위해 오라클 DBMS에 도입된 기술들을 분석 기능, 성능 향상, 그리고 ETT 기능을 중심으로 살펴보았다. IBM DB2나 MS SQL Server 같은 주요 관계형 DBMS 벤더들도 이와 유사한 DW 관련 기술들을 꾸준히 발전시켜 왔다. 분명한 점은 이 기술들을 통해서 관계형 DBMS가 진정한 DW용 플랫폼으로 자리잡았다는 점이고, 이전에는 OLAP 도구들이나 ETT 도구에서 수행되던 여러 복잡하고 지능적인 역할을 DBMS 엔진 안으로 흡수하고 있다는 점이다.

이와 같은 DW 관련 데이터베이스 기술의 발전 연장선 상에서 지속적인 성능 개선 기술이 나올 것이며, 분석 기능을 넘어선 데이터마이닝(data mining) [16], 웹 서비스나 유비쿼터스 환경에서 개인화(personalization) 서비스를 위한 SQL의 확장[17]이 있을 것이다.

## 참고문헌

[1] Bill Inmon, Building the Data Warehouse. John Wiley, 1992

[2] Surajit Chaudri et al, "An Overview of Data Warehousing and OLAP Technology," ACM

SIGMOD Record, Vol. 27, No. 1, 1997
[3] R. Kimball, "Why Decision Support Fails and How to Fix it?," ACM SIGMOD Record, Vol. 25, No. 3, 1995
[4] Jim Gray et al, "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals," Proceedings of ICDE 1996
[5] D. Chatziantoniou, K. Ross, "Querying Multiple Features in Relational Databases," Proceedings of VLDB 1996
[6] Oracle Corp., "Analytic SQL Features in Oracle 9i," Oracle Technical White Paper 2002,
[7] Ralph Kimball, "The Data Warehouse Life cycle Toolkit," John Wiley, 1999
[8] Andrew Witkowski et al., "Spreadsheets in RDBMS for OLAP," Proceedings of SIGMOD 2003
[9] R. G. Bello et al., "Materialized Views in Oracle," Proceedings of VLDB 1998
[10] Meikel Pss and Dmitry Potapov, "Table Compression in Oracle," Proceedings of VLDB 2003
[11] Oracle Corp., Oracle9i Data Warehousing Guide Release 2 (9.2), http://otn.Oracle.com/docs/products/Oracle9i/doc_library/release2/appdev.920/a96595.pdf
[12] Oracle Corp. "Query Optimization in Oracle9i," Oracle Technical White Paper, 2002
[13] Benoit Dageville, Mohamed Zait, "SQL Memory Management in Oracle9i," VLDB2002
[14] J-C Freytag et al, "Exploring SQL for ETML Program Execution," Proceedings of ICDE 2001
[15] Oracle Corp., "ETL Processing within Oracle9i," Oracle Technical White Paper 2002
[16] Sunita Sarawagi et al., "Integrating Mining with Relational Database Systems: Alternatives and Implications," Proceedings of SIGMOD 1998
[17] Werner Kieling, "Foundations of Preferences in Database Systems," Proceedings of VLDB, 2002

이 상 원

1991 서울대학교 컴퓨터공학 학사
1994 서울대학교 컴퓨터공학 석사
1999 서울대학교 컴퓨터공학 박사
1999~2001 한국오라클
2001~2002 이화여자대학교 컴퓨터학과 BK21 계약교수
2002~현재 성균관대학교 정보통신공학부 컴퓨터전공 교수
관심분야 : 데이터웨어하우스, OLAP, 대용량 데이터베이스
E-mail : wonlee@ece.skku.ac.kr