Oracle Database 10g의 SQL Model 구문

오라클 기술 백서
2003년 8월

ORACLE®

Oracle Database 10g의 SQL Model 구문

요약 3

서론 3

개념 4

기술 상세 6

기본 Syntax 6

샘플 데이타 6

첫 번째 Model 구문 예제 7

셀과 값의 참조 8

Positional Cell Reference – 단일 셀에 대한 접근 및 업데이트 8

Symbolic Cell Reference: 다수의 셀에 대한 접근 및 업데이트 9

단일 쿼리를 위한 Positional/Symbolic Cell Reference 10

포뮬러의 우측 항에 Multi-Cell Reference 적용하기 11

CV() 함수: 우측 항 계산을 위해 좌측 항의 값 이용하기 12

로우간 계산(inter-row calculation)을 위한 CV()의 활용 13

"ANY" 키워드를 사용한 와일드 카드 조건 적용 14

FOR 루프의 활용 14

FOR 루프를 이용한 value sequence의 생성 16

그 밖의 기능 16

포뮬러 처리 순서의 지정 17

NULL Measure와 Missing Cell 17

REFERENCE MODEL 17

ITERATIVE MODEL 17

결론 17

APPENDIX: MODEL 구문의 EXPLAIN PLAN 18

ORDERED 프로세싱을 이용한 Explain Plan 18

ACYCLIC FAST 프로세싱을 이용한 Explain Plan 19

Oracle Database 10*g*의 SQL Model 구문

요약

SQL 구문에서 복잡한 계산을 구현하는 것은 쉽지 않은 작업입니다. 특히 로우(row) 단위의 참조가 필요한 SQL 구문에서는 그 어려움이 더욱 배가됩니다. 매우 정교한 수준의 SQL Join과 Union 구문이 필요할 수도 있으며, 이렇게 작성된 코드는 개발, 유지보수, 실행 면에서 효율성을 보장하기 어렵습니다. Oracle Database 10*g*는 복잡한 SQL 계산 작업을 단순화하기 위한 혁신적인 접근 방법으로 SQL Model 구문을 제공합니다.

Model 구문을 이용하여 SQL의 Select 문을 확장함으로써, 관계형 데이타를 다차원 어레이 형태로 처리하고 어레이에 대한 포뮬러(formula)를 정의할 수 있습니다. 이렇게 작성된 코드는 문법적으로 간결하고 읽기 쉬울 뿐 아니라, 어려운 계산 작업을 효과적으로 처리하는 데 도움을 줍니다. 또 Model 구문을 사용해서 포뮬러의 종속성 문제를 해결하고, 상호연결된 포뮬러를 대량으로 생성하고 관리할 수 있습니다. 그 밖에도 보다 진보된 형태의 최적화 기술과 데이타 구조를 적용하고 성능을 향상시킬 수 있는 기능이 제공됩니다. Model 구문의 강력한 표현능력과 사용편의성을 활용하여 오라클의 확장성 및 관리성을 향상하고 데이타베이스 애플리케이션의 수준을 한 차원 높게 끌어올릴 수 있습니다.

서론

비즈니스 또는 테크놀로지 영역에서 데이타베이스 애플리케이션을 구현하다 보면, 종종 SQL로는 구현하기 어려운 계산 문제에 맞닥뜨리게 됩니다. 예를 들어 제품별, 지역별 시장 점유율을 계산하거나 커스터마이즈된 aggregation을 구현하려면 복잡한 self-join 또는 union 구문을 사용해야 할 수 있습니다. 또 시간 단위 분석, 반복 연산을 수행하기 위해 데이타베이스 외부의 계산용 엔진(calculation engine)을 활용해야 할 수도 있습니다. 예산 및 매출 예측과 같은 재무 모델링을 위해 논리적인 관점에서 상호독립적인 포뮬러를 생성하는 작업은 결코 간단하게 해결할 수 있는 문제가 아닙니다.

이처럼 SQL의 내재적인 한계가 애플리케이션 개발 및 유지보수 과정에서 걸림돌이 되는 경우도 있지만, 이보다 더 심각한 문제는 따로 있습니다. 데이타를 데이타베이스로 추출하는 작업을 수행해야 하는 경우, 관리자의 업무 부담은 증가하는 반면 관리편의성은 그만큼 저하됩니다. 추출된 데이타를 PC 스프레드시트 상에서 분석하는 시장 예측 애플리케이션의 경우를 예로 들어봅시다. 많은 수의 사용자가 각자의 스프레드시트에 개별적으로 데이타를 관리하고 있다면, 변경사항이 발생할 때마다

모든 PC의 데이타를 동시에 업데이트해 주어야 합니다. 그 중 하나라도 업데이트 시기를 놓쳐버린다면 정확한 분석 결과를 보장할 수 없게 됩니다. 포뮬러 자체의 일관성이 보장되는 환경이라 하더라도 추출된 데이타가 통합적으로 관리되고, 시기 적절하게 각각의 PC에 연결되고 업데이트되어야만 제대로 된 결과를 보장할 수 있습니다. 또 랩톱 컴퓨터를 사용하는 사용자들이 수시로 네트워크 연결을 해제하는 경우라면 문제가 더욱 어려워집니다. 이러한 점에서 볼 때, 데이타베이스 내부에서 복잡한 계산 작업을 중앙집중적으로 구현할 수 있다면, 비즈니스적으로 큰 이점을 갖게 될 것입니다.

Oracle Database 10*g*이 제공하는 SQL Model 구문을 이용하여 이러한 문제를 해결할 수 있습니다. Model 구문을 이용하여 SQL SELECT 문을 확장함으로써 다양한 애플리케이션의 계산 작업을 단순화하고 중앙집중화할 수 있습니다. Model 구문을 이용한 쿼리 결과는 다차원 어레이의 형태로 제공되며, 포뮬러를 이용해 로우 또는 어레이 간을 참조하고 복잡한 계산을 수행하는 것이 가능합니다. Model 구문을 이용하여 구현된 애플리케이션은 기존의 PC 스프레드시트와 계산 엔진을 대체할 뿐 아니라, 보다 안정적이고 효율적인 솔루션을 제공합니다. 복잡한 계산 작업을 데이타베이스에 통합함으로써 성능, 확장성, 관리성을 향상할 수 있습니다:

- **성능** – Model 구문을 이용하면 SQL join 및 union 연산의 적용을 최소화할 수 있습니다. Model 구문은 보다 향상된 알고리즘과 데이타 구조를 이용하므로 성능 향상이 가능합니다. 특히 Model 구문을 사용함으로써 외부 프로세싱을 위한 데이타 전달 및 반환 과정(데이타를 별도 애플리케이션에 복제하고, 데이타를 처리한 후 그 결과를 데이타베이스에 로드하는 과정)을 생략할 수 있다는 점을 중요한 성능 개선 포인트로 들 수 있습니다.
- **확장성** – 오라클의 병렬 쿼리 기능은 써드 파티 계산 엔진(calculation engine)에 비해 훨씬 뛰어난 수준의 확장성을 제공합니다. Model 구문은 오라클의 병렬 알고리즘(parallelism)을 통해 시스템 자원을 효과적으로 활용합니다.
- **관리성** – 데이타의 계산 작업을 중앙집중적으로 통합함으로써, 분산된 형태로 존재하는 데이타의 일관성 및 보안 문제를 해결할 수 있습니다. 또 여러 애플리케이션이 관계형 데이타베이스를 공유하도록 함으로써 통합 작업을 단순화하고, 여러 종류의 계산 엔진을 사용할 때 발생할 수 있는 데이타 구조의 호환성 문제를 방지할 수 있습니다.

개념

Model 구문을 이용하여 쿼리의 컬럼을 세 가지 유형(partitioning, dimension, measure column)으로 매핑함으로써 다차원 어레이를 정의할 수 있습니다. 각각의 컬럼 유형이 담당하는 역할이 아래와 같습니다:

- **파티션(partition)** – (오라클의 Data Warehousing Guide에 기술된) 분석 함수의 파티션과 유사한 방법으로 결과 집합(result set)의 논리적 블록을 정의합니다. 포뮬러는 각각의 파티션을 독립적인 어레이로 취급합니다.
- **디멘션(dimension)** – 파티션 내의 개별적인 measure cell을 정의합니다. 각각의 컬럼은 날짜, 지역, 제품명과 같은 속성을 정의합니다.

- **Measure** – 스타 스키마(star schema)의 fact table과 유사한 특성을 갖습니다. Measure는 일반적으로 매출 또는 비용과 같은 숫자 데이타를 저장하는데 사용됩니다. 파티션 내부의 개별적인 셀(cell)은 디멘션의 조합을 통해 액세스할 수 있습니다.

디멘션 값을 이용한 연산 룰(computation rule)을 정의하고, 이를 이용하여 다차원 어레이에서 포뮬러를 생성할 수도 있습니다. Rule은 와일드 카드 문자와 FOR 루프를 지원합니다. 새로운 Model 구문은 고전적인 MODEL 계산 기능이 제공하는 분석 함수를 데이타베이스에 통합한 형태로 구현되며, 개선된 심볼 참조 기능과 향상된 가용성, 확장성, 관리성을 제공합니다.

아래 그림은 SALES 테이블에 Model 구문을 적용하는 개념적인 사례를 설명하고 있습니다. SALES 테이블은 country, product, year, sales amount 등의 컬럼을 갖습니다. 아래 그림은 세 부분으로 나뉘어져 있습니다. 제일 윗부분은 테이블을 partitioning, dimension, measure 컬럼으로 구분한 결과를 보여주고 있습니다. 중간 부분은 2002년에 대한 Prod1과 Prod2 값의 계산을 위한 두 가지 rule을 나타내고 있습니다. 그리고 마지막 부분은 SALES 테이블에 규칙을 적용하여 쿼리를 수행한 결과를 보여주고 있습니다. 그림에서 음영으로 표시되지 않은 부분은 데이타베이스로부터 직접 가져온 데이타를, 음영으로 표시된 부분은 rule로부터 계산된 결과를 의미합니다. 아래 예에서는 rule이 각각의 파티션 별로 적용되었음을 참고하시기 바랍니다.

Mapping of columns to model entities:

Country	Product	Year	Sales
Partition	Dimension	Dimension	Measure

Rules:

Sales(Prod1, 2002) = Sales(Prod1,2000)+Sales(Prod1,2001)
Sales(Prod2, 2002) = Sales(Prod2,2000)+Sales(Prod2,2001)

Output of model clause:

Country	Product	Year	Sales	
Partition	Dimension	Dimension	Measure	
A	Prod1	2000	10	Original Data
A	Prod1	2001	15	
A	Prod2	2000	12	
A	Prod2	2001	16	
B	Prod1	2000	21	
B	Prod1	2001	23	
B	Prod2	2000	28	
B	Prod2	2001	29	
A	Prod1	2002	25	Rule Results
A	Prod2	2002	28	
B	Prod1	2002	44	
B	Prod2	2002	57	

Model 구문이 테이블의 데이타를 직접 업데이트하거나 insert하지 않는다는 점도 참고할 필요가 있습니다. 테이블의 값을 변경하려면 Model 구문의 수행 결과를 이용해 INSET, UPDATE, MERGE 구문을 실행해야 합니다.

기술 상세

지금까지 SQL Model 구문의 개념을 간략하게 살펴보았습니다. 기능을 분명하게 이해하려면 좀더 구체적인 사례가 필요할 것입니다. 이 문서의 나머지 부분에서는 예제를 통해 Model의 개념과 키워드를 단계별로 설명합니다. 예제에서 Model에 포함된 모든 기능을 설명할 수는 없겠지만, 중요한 개념은 한 번씩 짚고 넘어가기로 하겠습니다. 이해를 돕기 위해, 모든 예제는 Oracle Database 10*g*와 함께 제공되는 샘플 데이타를 기반으로 구현됩니다. 여기에서 사용되는 Sales History(SH) 샘플 스키마는 비즈니스 인텔리전스 애플리케이션에서 자주 사용되는 스타 스키마(star schema)입니다. SH의 팩트 테이블(fact table)에는 전세계 시장을 통해 가전제품을 판매하는 한 제조업체의 세일즈 히스토리 데이타가 저장되어 있습니다.

기본 Syntax

Model 구문의 기본적인 syntax가 아래와 같습니다. 아래에 기술된 내용 이외에도 추가적으로 적용할 수 있는 syntax 아이템이 있지만, 여기에서는 가장 핵심적인 부분만을 설명하는 것으로 합니다. Model 구문은 SELECT 문에 포함된 모든 구문(ORDER BY 제외)이 실행된 이후에 마지막으로 실행됩니다.

```
<prior clauses of SELECT statement>
MODEL [main]
        [reference models]
        [PARTITION BY (<cols>)]
         DIMENSION BY (<cols>)
         MEASURES (<cols>)
            [IGNORE NAV] | [KEEP NAV]
        [RULES
            [UPSERT | UPDATE]
            [AUTOMATIC ORDER | SEQUENTIAL ORDER]
            [ITERATE (n) [UNTIL <condition>] ]
            ( <cell_assignment> = <expression> ... )
```

샘플 데이타

이해를 돕기 위해, Oracle 10*g*의 Sales History(SH) 샘플 스키마를 대상으로 하는 뷰를 생성하여 예제를 더욱 단순화해 보겠습니다. sales_view 뷰는 각 국가별로 모든 채널의 product sales 값을 연 단위로 합산한 결과를 보여줍니다. sales_view 뷰는 1백만 개의 로우(row)를 갖는 팩트 테이블(fact table)로부터 생성되며, 아래와 같이 정의됩니다:

```
CREATE VIEW sales_view AS
SELECT country_name country, prod_name prod,
        calendar_year year,
        SUM(amount_sold) sale, COUNT(amount_sold) cnt
FROM sales, times, customers, countries, products
WHERE sales.time_id = times.time_id AND
        sales.prod_id = products.prod_id AND
        sales.cust_id = customers.cust_id AND
        customers.country_id = countries.country_id
GROUP BY country_name, prod_name, calendar_year;
```

위 구문의 materialized view를 생성하면 오라클 시스템 상에서 예제를 실행하는 데 소요되는 시간을 단축할 수 있습니다. 또 오라클의 summary management 기능을 이용하여 materialized view를 사용하도록 예제를 자동 재작성할 수 있습니다.

첫 번째 Model 구문 예제

Model의 첫 번째 사용 예로 다음과 같은 구문을 생각해 볼 수 있습니다. 아래 구문은 두 가지 제품의 매출을 계산하고, 계산된 결과를 기반으로 새로운 제품의 매출을 예측합니다.

```
SELECT SUBSTR(country,1,20) country, SUBSTR(prod,1,15) prod,
year, sales
FROM sales_view
WHERE country IN ('Italy','Japan')
MODEL RETURN UPDATED ROWS
        PARTITION BY (country)
        DIMENSION BY (prod, year)
        MEASURES (sale sales)
        RULES (
            sales['Bounce', 2002] = sales['Bounce', 2001] +
                sales['Bounce', 2000],
            sales['Y Box', 2002] = sales['Y Box', 2001],
            sales['2_Products', 2002] = sales['Bounce', 2002] +
        sales['Y Box', 2002])
        ORDER BY country, prod, year;
```

위 구문의 실행 결과가 아래와 같습니다:

COUNTRY	PROD	YEAR	SALES
Italy	2_Products	2002	92613.16
Italy	Bounce	2002	9299.08
Italy	Y Box	2002	83314.08
Japan	2_Products	2002	103816.6
Japan	Bounce	2002	11631.13
Japan	Y Box	2002	92185.47

위 실행결과를 통해 데이타가 국가를 기준으로 파티셔닝 되어 있으며, 각각의 국가를 대상으로 포뮬러가 적용되고 있음을 확인할 수 있습니다. 세일즈 팩트 데이타는 2001년을 마지막으로 하므로, 2002년 또는 그 이후에 대한 룰이 적용되는 경우 새

로운 셀(cell)이 삽입되게 됩니다. 첫 번째 룰은 "Bounce"라는 비디오 게임의 2002년 매출을 2000년 및 2001년 매출의 합계로 정의하고 있습니다. 두 번째 룰은 2002년 Y Box의 매출이 2001년과 동일한 것으로 정의하고 있습니다. 세 번째 룰은 "2_Products"라는 제품의 2002년 매출을 2002년 Bound와 Y Box 매출의 합계로 정의합니다. 2_Products의 매출 값이 앞의 두 가지 포뮬러로부터 파생되므로, 2_Product 룰을 적용하기 전에 먼저 Bounce 및 Y Box를 위한 룰을 실행해야 합니다.

위 예제에서 다음과 같은 점을 참고할 필요가 있습니다:

- MODEL 키워드 다음에 사용되는 "RETURN UPDATED ROWS" 구문을 이용하여, 이 쿼리에서 새로 생성되거나 업데이트된 로우(ROW)만을 반환하도록 제한할 수 있습니다. 이 구문은 새로 계산된 값만을 포함하는 결과가 필요한 경우 유용합니다. 아래 소개될 예제에서도 RETURN UPDATED ROWS 구문을 계속적으로 사용하게 될 것입니다.
- "RULES" 키워드는 반드시 필요하지 않으나, 가독성의 향상을 위해 포함시켰습니다.
- 위 예제에서는 ORDER BY 구문이 반드시 필요하지 않습니다. 하지만 사용자가 많은 수의 국가를 포함하는 형태로 구문을 변경하는 경우를 감안하여 포함시켰습니다.

셀과 값의 참조

다음으로 SQL Model에서 셀(cell)과 값(value)을 참조하는 테크닉에 대해 설명합니다. SQL Model 구문을 제대로 활용하려면 cell reference 방식에 대한 이해가 필수적으로 요구됩니다.

Positional Cell Reference – 단일 셀에 대한 접근 및 업데이트

Bounce 제품에 대한 2000년도의 Italy 지역 매출 값을 10으로 변경하고자 하는 경우를 가정해 봅시다. 아래와 같은 쿼리를 사용하여 셀의 값을 업데이트 할 수 있습니다:

```
SELECT SUBSTR(country,1,20) country, SUBSTR(prod,1,15) prod,
year, sales
FROM sales_view
WHERE country='Italy'
MODEL RETURN UPDATED ROWS
        PARTITION BY (country)
        DIMENSION BY (prod, year)
        MEASURES (sale sales)
        RULES (
            sales['Bounce', 2000] = 10 )
ORDER BY country, prod, year;
```

COUNTRY	PROD	YEAR	SALES
Italy	Bounce	2000	10

위 쿼리의 포뮬러에서는 "positional cell reference"를 사용하고 있습니다.

"positional cell reference "에서는 표현식(expression)에서의 포지션(position)를 기준으로 셀이 참조하는 값과 디멘션을 대응시킵니다. 모델의 DIMENSION BY 구문은 각 디멘션에 할당된 포지션을 정의합니다. 위 쿼리의 경우 첫 번째 포지션은 "prod" 디멘션에, 두 번째 포지션은 "year" 디멘션에 대응됩니다.

이번에는 Bounce 제품의 2005년도 Italy 지역의 매출을 20으로 설정하고자 하는 경우를 생각해 봅시다. 작성된 쿼리가 아래와 같습니다:

```
SELECT SUBSTR(country,1,20) country, SUBSTR(prod,1,15) prod,
year, sales
FROM sales_view
WHERE country='Italy'
MODEL RETURN UPDATED ROWS
PARTITION BY (country)
DIMENSION BY (prod, year)
MEASURES (sale sales)
RULES (
sales['Bounce', 2005] = 20 )
ORDER BY country, prod, year;
```

COUNTRY	PROD	YEAR	SALES
Italy	Bounce	2005	20

위 쿼리에서 사용된 포뮬러는 2005년도의 값을 설정하기 위해 어레이에 새로운 셀을 생성합니다.

참고: (미래의 매출 예측을 위해) 새로운 셀을 생성하는 경우, positional reference 또는 FOR 루프(FOR 루프에 대해서는 뒷부분에서 설명합니다)를 사용해야 합니다. positional reference는 어레이에 대한 update와 insert를 동시에 허용합니다. 이러한 과정을 "upsert" 프로세스라고도 부릅니다.

Symbolic Cell Reference: 다수의 셀에 대한 접근 및 업데이트

이번에는 대해 1999년 이후 Bounce 제품의 매출 데이타를 업데이트하는 쿼리를 작성해 봅시다. 이번 예제에서도 Italy 지역의 값을 10으로 변경하도록 합니다. 작성된 쿼리가 아래와 같습니다:

```
SELECT SUBSTR(country,1,20) country, SUBSTR(prod,1,15) prod,
        year, sales
FROM sales_view
WHERE country='Italy'
MODEL RETURN UPDATED ROWS
        PARTITION BY (country)
        DIMENSION BY (prod, year)
        MEASURES (sale sales)
        RULES (
            sales[prod='Bounce', year>1999] = 10 )
ORDER BY country, prod, year;
```

COUNTRY	PROD	YEAR	SALES
Italy	Bounce	2000	10
Italy	Bounce	2001	10

위의 쿼리는 "symbolic cell reference "를 사용하고 있습니다. Symbolic cell reference에서는 표준 SQL 조건문을 이용하여 포뮬러를 적용할 셀을 선택합니다. 이때 <, >, IN, BETWEEN과 같은 조건을 사용하는 것이 가능합니다. 위 쿼리에서는 Bounce를 제품명으로 하고 연도가 1999년 이후인 셀에 대하여 포뮬러를 적용하고 있습니다. 이와 같은 방법으로 단일 포뮬러를 사용하여 여러 개의 셀을 동시에 액세스할 수 있습니다.

참고: Symbolic reference는 매우 강력하지만 기존 셀을 업데이트하는 용도로만 사용됩니다. Symbolic reference 를 이용하여 새로운 셀을 생성하는 것은 불가능합니다 (예: 향후 매출의 예측을 위한 셀의 생성). cell reference가 디멘션에서 symbolic notation을 사용하는 경우, 포뮬러는 업데이트 작업만을 수행합니다. FOR 루프를 포함하는 단일 포뮬러를 사용하여 여러 개의 셀을 생성하는 방법에 대해서는 뒷부분에서 설명하도록 하겠습니다.

단일 쿼리를 위한 Positional/Symbolic Cell Reference

이번에는 여러 해에 걸쳐 여러 개의 국가, 여러 개의 제품에 대한 매출 데이타를 업데이트하고, 그 결과를 새로운 셀로 삽입하는 쿼리를 작성해 봅시다. 단일 포뮬러를 포함하는 쿼리를 여러 차례 수행하는 것보다 하나의 쿼리에서 모든 작업을 동시에 수행하는 것이 훨씬 효율적입니다. 이와 같이 함으로써 데이타 접근의 횟수를 최소화할 수 있을 뿐 아니라 보다 간략한 SQL 문을 구현하고 개발자의 생산성을 향상시킬 수 있습니다. 두 가지 기존 제품에 대한 값을 업데이트하고 새로운 제품을 insert 하는 구문이 아래와 같습니다. 쿼리의 주석을 참고하시기 바랍니다:

```
SELECT SUBSTR(country,1,20) country, SUBSTR(prod,1,15) prod,
       year, sales
FROM sales_view WHERE country IN ('Italy','Japan')
MODEL RETURN UPDATED ROWS
       PARTITION BY (country)
       DIMENSION BY (prod, year)
       MEASURES (sale sales)
       RULES (
            sales['Bounce', 2002] = sales['Bounce', year = 2001] ,
                 --positional notation: can insert new cell
            sales['Y Box', year>2000] = sales['Y Box', 1999],
                 --symbolic notation: can update existing cell
            sales['2_Products', 2005] = sales['Bounce', 2001] +
                 sales['Y Box', 2000] )
                 --positional notation: permits creation of new cell
                 --for new product
       ORDER BY country, prod, year;
```

COUNTRY	PROD	YEAR	SALES
Italy	2_Products	2005	34579.63
Italy	Bounce	2002	4928.65
Italy	Y Box	2001	15177.7
Japan	2_Products	2005	52563.55
Japan	Bounce	2002	6443.77
Japan	Y Box	2001	22297.76

2001년 이후의 값은 샘플 데이타에 존재하지 않기 때문에, 2002년 또는 그 이후에 룰을 적용하는 경우 새로운 셀이 insert 됩니다. 새로운 제품명이 정의된 경우에도 마찬가지로 새로운 셀이 insert 됩니다. 위 쿼리의 세 번째 포뮬러에서 '2_Products'라는 새로운 제품이 2005년에 대해 정의되었으므로 이를 위해 새로운 셀이 insert 될 것입니다. 2002년 Bounce 제품을 대상으로 하는 첫 번째 룰의 경우 positional notation이 사용되었으며 새로운 셀이 insert됩니다. Y Box를 대상으로 하는 두 번째 룰의 경우에는 symbolic notation이 사용되었으며, 2001년 'Y Box'에 대한 값이 이미 존재하므로 해당 값에 대하여 업데이트 작업만을 수행합니다. 2005년 '2_Products'를 대상으로 하는 세 번째 룰은 positional notation이 사용되었으므로 새로운 셀이 insert됩니다.

포뮬러의 우측 항에 Multi-Cell Reference 적용하기

위의 사례들은 모두 포뮬러의 좌측 항에 multi-cell reference를 사용한 경우를 설명하고 있습니다. 포뮬러의 우측 항에 multi-cell reference를 적용하려면, aggregate 함수를 이용하여 여러 셀의 값을 단일 값으로 변환하는 작업을 수행해야 합니다. 이때 OLAP aggregate(inverse distribution function, hypothetical rank, distribution function 등), statistical aggregate, 사용자 정의 aggregate 함수 등을 포함하는 모든 종류의 aggregate 함수를 사용할 수 있습니다.

Italy 지역의 Bounce 제품에 대한 2005년 매출 예측(sales forecast) 값을, 1999년부터 2001년까지의 기간 중의 최대 매출액에 100을 더한 값으로 설정하는 쿼리가 아래와 같습니다:

```
SELECT SUBSTR(country,1,20) country, SUBSTR(prod,1,15) prod,
       year, sales
FROM sales_view
WHERE country='Italy'
MODEL RETURN UPDATED ROWS
       PARTITION BY (country)
       DIMENSION BY (prod, year)
       MEASURES (sale sales)
       RULES (
            sales['Bounce', 2005] = 100 + MAX(sales)['Bounce',
                 year BETWEEN 1998 AND 2002] )
ORDER BY country, prod, year;
```

COUNTRY	PROD	YEAR	SALES
Italy	Bounce	2005	5028.65

위의 쿼리에서는 BETWEEN 조건을 사용하여 포뮬러 우측 항에 여러 개의 셀을 적용하고, MAX() 함수를 이용하여 단일 값을 생성하였습니다.

참고: aggregate 함수는 포뮬러의 우측 항에만 적용될 수 있습니다. aggregate 함수에 사용되는 인자(argument)로는 상수(constant), 바인드 변수(bind variable), MODEL 구문의 measure, 또는 이들을 포함하는 표현식(expression) 등이 사용가능합니다.

CV() 함수: 우측 항 계산을 위해 좌측 항의 값 이용하기

CV() 함수는 포뮬러의 좌측 항에 정의된 multi-cell reference를 우측 항으로 복사하는 기능을 제공하는 매우 강력한 툴입니다. CV() 함수를 이용하여 매우 간결하면서도 유연한 multi-cell 포뮬러를 구현할 수 있습니다. CV() 함수는 SQL join과 유사한 성격을 가지지만, 훨씬 간결하고 이해하기 쉽다는 부가적인 장점을 갖습니다.

Italy 지역의 Bounce 제품에 대한 다년 간의 매출 값을 업데이트해야 하는 경우를 생각해 봅시다. 여기서 연도별 매출은 해당 연도의 'Mouse Pad' 매출 합계와 해당 연도의 'Y Box' 매출의 20%를 더한 값으로 설정됩니다. 작성된 쿼리가 아래와 같습니다:

```
SELECT SUBSTR(country,1,20) country, SUBSTR(prod,1,15) prod,
year, sales
FROM sales_view
WHERE country='Italy'
MODEL RETURN UPDATED ROWS
        PARTITION BY (country)
        DIMENSION BY (prod, year)
        MEASURES (sale sales)
        RULES (
            sales['Bounce', year BETWEEN 1995 AND 2002] =
                sales['Mouse Pad', CV(year)] +
                0.2 * sales['Y Box', CV(year)]
        )
ORDER BY country, prod, year;
```

COUNTRY	PROD	YEAR	SALES
Italy	Bounce	1999	7681.51
Italy	Bounce	2000	9586.286
Italy	Bounce	2001	21587.916

포뮬러에서 사용된 두 개의 CV() 함수는 좌측 항에서 참조되고 있는 셀의 연도별 디멘션 값을 반환합니다. 포뮬러의 좌측 항이 'Bounce'와 1999년을 참조하는 경우, 우측 항의 표현식은 다음과 같게 됩니다:

```
sales['Mouse Pad', 1999] + 0.2 * sales['Y Box', 1999].
```

마찬가지로 포뮬러의 좌측 항이 'Bounce'와 2000년을 참조하는 경우, 우측 항의 표현식은 다음과 같습니다:

```
sales['Mouse Pad', 2000] + 0.2 * sales['Y Box', 2000].
```

CV() 함수는 디멘션 키(dimension key)를 인자(argument)로 사용합니다. 인자를

사용하지 않고 CV() 함수를 적용하는 경우에는 positional reference가 사용됩니다.

결과적으로 포뮬러는 다음과 같은 형태로 작성됩니다.

```
s ['Bounce', year BETWEEN 1995 AND 2002] =
        s['Mouse Pad', CV()] + 0.2 * s['Y Box', CV()]
```

위의 결과에서 (조건상으로는 1995년에서 2002년까지의 기간이 모두 참조되고 있는데 반해) 1999-2001년 간의 값만이 표시되고 있음을 주목하시기 바랍니다. 이는 테이블에 1999-2001년 기간의 데이타만이 존재하기 때문입니다. 위 쿼리에서는 포뮬러의 유연성을 설명하기 위한 목적에서 실제 데이타보다 넓은 범위의 조건을 적용하였습니다.

로우간 계산(inter-row calculation)을 위한 CV()의 활용

CV()를 이용하여 매우 유연한 표현식(expression)을 생성할 수 있습니다. 예를 들어 CV(year) 값에 대해 뺄셈 계산을 하고 그 결과를 이용하여 데이타 셋의 다른 로우(row)를 참조할 수 있습니다. 즉, 셀 참조를 위해 "CV(year)-2"라는 표현식을 생성하고 2년 전의 데이타를 액세스하는 것이 가능합니다.

이번에는 Italy 지역의 'Y Box', 'Bounce', 'Mouse Pad' 제품에 대한 연 단위 매출 성장률을 계산해 봅시다. 작성된 쿼리와 실행 결과가 아래와 같습니다:

```
SELECT SUBSTR(country,1,10) country, SUBSTR(prod,1,10) prod,
year, sales, growth_pct
FROM sales_view
WHERE country='Italy'
MODEL RETURN UPDATED ROWS
        PARTITION BY (country)
        DIMENSION BY (prod, year)
        MEASURES (sale sales, 0 growth_pct)
        RULES (
            growth_pct[prod IN ('Bounce','Y Box','Mouse Pad'),
                    year BETWEEN 1998 and 2001] =
                100* (sales[CV(prod), CV(year)] -
                    sales[CV(prod), CV(year) -1] ) /
                    sales[CV(prod), CV(year) -1] )
ORDER BY country, prod, year;
```

COUNTRY	PROD	YEAR	SALES	GROWTH_PCT
Italy	Bounce	1999	2,474.78	
Italy	Bounce	2000	4,333.69	75.11
Italy	Bounce	2001	4,846.30	11.83
Italy	Mouse Pad	1998	3,055.69	
Italy	Mouse Pad	1999	4,663.24	52.61
Italy	Mouse Pad	2000	3,662.83	-21.45
Italy	Mouse Pad	2001	4,747.90	29.62
Italy	Y Box	1999	15,215.16	
Italy	Y Box	2000	29,322.89	92.72
Italy	Y Box	2001	81,207.55	176.94

위 결과에서 빈 칸으로 표시된 부분에는 NULL 값이 반환되었음을 참고하십시오.

제품에 대한 2년 전의 값이 존재하지 않는 경우 포뮬러는 null을 반환합니다. 1998년에는 세 가지 제품 모두 존재하지 않았으므로, 1999년의 GROWTH_PCT 계산 결과는 모두 NULL로 표시되었습니다.

"ANY" 키워드를 사용한 와일드 카드 조건 적용

와일드 카드 연산자를 Model 구문의 셀 참조에 적용하기 위해 "ANY" 키워드를 사용할 수 있습니다. 앞의 쿼리에서 "year between 1998 and 2001"의 조건을 아래와 같이 대치하는 것이 가능합니다:

```
SELECT SUBSTR(country,1,10) country, SUBSTR(prod,1,10) prod,
year, sales, growth_pct
FROM sales_view
WHERE country='Italy'
MODEL RETURN UPDATED ROWS
PARTITION BY (country)
DIMENSION BY (prod, year)
MEASURES (sale sales, 0 growth_pct)
RULES (
growth_pct[prod IN ('Bounce','Y Box','Mouse Pad'),
ANY] =
100* (sales[CV(prod), CV(year)] -
sales[CV(prod), CV(year) -1] ) /
sales[CV(prod), CV(year) -1] )
ORDER BY country, prod, year;
```

COUNTRY	PROD	YEAR	SALES	GROWTH_PCT
Italy	Bounce	1999	2,474.78	
Italy	Bounce	2000	4,333.69	75.11
Italy	Bounce	2001	4,846.30	11.83
Italy	Mouse Pad	1998	3,055.69	
Italy	Mouse Pad	1999	4,663.24	52.61
Italy	Mouse Pad	2000	3,662.83	-21.45
Italy	Mouse Pad	2001	4,747.90	29.62
Italy	Y Box	1999	15,215.16	
Italy	Y Box	2000	29,322.89	92.72
Italy	Y Box	2001	81,207.55	176.94

이처럼 ANY 키워드를 사용하여 쿼리 조건을 변경하더라도 (데이타 셋의 범위가 1998년에서 2001년으로 동일하게 한정되므로) 동일한 결과가 반환됨을 확인할 수 있습니다.

NULL을 포함하는 모든 디멘션 값을 셀 참조에 포함하도록 하기 위해 ANY를 사용할 수도 있습니다. symbolic reference 에서 "IS ANY" 구문을 사용하면 됩니다. ANY 키워드를 사용하면 positional/symbolic notation 두 가지 경우 모두 새로운 셀의 insert가 불가능하다는 사실에 주의하시기 바랍니다.

FOR 루프의 활용

FOR 루프를 이용하면 단일 포뮬러를 사용하여 다수의 새로운 셀을 insert 할 수 있습니다. (FOR 구문은 포뮬러의 좌측 항에만 적용 가능합니다.) 아래 포뮬러는 3가

지 제품의 2005년 매출 예측 값으로 2001년 매출 값보다 30% 높은 수치를 적용하고 있습니다.

```
RULES
( sales['Mouse Pad', 2005] = 1.3 * sales['Mouse Pad', 2001],
  sales['Bounce', 2005] = 1.3 * sales['Bounce', 2001],
  sales['Y Box', 2005] = 1.3 * sales['Y Box', 2001] )
```

포뮬러의 좌측 항에 positional notation을 사용함으로써, 2005년에 해당하는 제품별 데이타가 insert되도록 할 수 있습니다. 하지만 이러한 방법을 사용하는 경우, 제품 종류의 수만큼 포뮬러를 작성해야 하므로 구현이 복잡해집니다. 제품의 종류가 십여 가지에 달한다면 이러한 방법은 고려하기 어려울 것입니다. 이때 아래와 같이 FOR 루프를 사용하여 간결한 형태로 계산 작업을 구현할 수 있습니다.

```
SELECT SUBSTR(country,1,20) country, SUBSTR(prod,1,15) prod,
year, sales
FROM sales_view
WHERE country='Italy'
MODEL RETURN UPDATED ROWS
        PARTITION BY (country)
        DIMENSION BY (prod, year)
        MEASURES (sale sales)
        RULES (
            sales[FOR prod IN ('Mouse Pad', 'Bounce', 'Y Box'),
                2005] = 1.3 * sales[CV(prod), 2001] )
ORDER BY country, prod, year;
```

```
COUNTRY         PROD            YEAR              S
-------------------------------------------------------------------------------
Italy           Bounce          2005         6407.245
Italy           Mouse Pad       2005          6402.63
Italy           Y Box           2005       108308.304
```

위의 쿼리에서 FOR 키워드를 삭제하면 기존에 존재하는 셀의 값만이 업데이트되며, 새로운 셀은 insert되지 않습니다. 따라서 반환되는 로우(row)는 없게 됩니다. 수정된 쿼리가 아래와 같습니다:

```
SELECT SUBSTR(country,1,20) country, SUBSTR(prod,1,15) prod,
year, sales
FROM sales_view
WHERE country='Italy'
MODEL RETURN UPDATED ROWS
        PARTITION BY (country)
        DIMENSION BY (prod, year)
        MEASURES (sale sales)
        RULES (
            sales[prod IN ('Mouse Pad', 'Bounce', 'Y Box'), 2005] =
                1.3 * sales[CV(prod), 2001] )
ORDER BY country, prod, year;
no rows selected
```

이처럼 FOR 루프는 단일 포뮬러와 positional reference를 이용해 여러 개의 포뮬러

를 작성하고 새로운 셀을 생성하는 (UPSERT 동작을 구현하는) 툴로써 활용됩니다.

FOR 루프를 이용한 value sequence의 생성

일정한 간격을 갖는 수열(sequence) 형태의 디멘션 값(dimension value)을 생성하는 용도로도 FOR 루프를 활용할 수 있습니다.

```
FOR dimension FROM <value1> TO <value2>
      [INCREMENT | DECREMENT] <value3>
```

위 구문은 value1과 value2을 범위로 value3만큼 증가(감소)하는 수열을 사용하여 디멘션 값을 생성합니다. 위와 같은 방법을 사용하여 다수의 셀을 업데이트하거나 insert하는 작업을 극적으로 단순화할 수 있습니다.

그 밖의 기능

포뮬러 처리 순서의 지정

원칙적으로 포뮬러는 Model 구문 내에서 정의된 순서대로 처리(evaluate)됩니다. "SEQUENTIAL ORDER" 키워드를 사용하면 포뮬러의 처리 순서를 명시적으로 정의할 수 있습니다. 또 "AUTOMATIC ORDER" 키워드를 사용하면 포뮬러의 종속성에 따라 올바른 순서로 자동 실행되도록 설정하는 것이 가능합니다. Model에 많은 포뮬러가 포함되어 있는 경우라면, 수작업으로 포뮬러의 처리 순서를 지정하는 것보다 AUTOMATIC ORDER 옵션을 사용하는 것이 더 효율적입니다. 이처럼 AUTOMATIC ORDER 옵션을 사용하여 개발 및 유지보수 과정의 생산성을 향상시킬 수 있습니다.

NULL Measure와 Missing Cell

SQL Model을 사용하는 애플리케이션을 구현하는 경우, 셀의 참조에 관련한 두 가지 예외적인 상황을 고려해야 합니다. 먼저 어레이 내부에 존재하지만 NULL 값이 할당된 셀, 그리고 어레이 내부에 아예 존재하지 않는 셀이 있을 수 있습니다. cell reference에 의해 참조되고 있으나 어레이 내부에 존재하지 않는 셀을 "missing cell"이라 부릅니다. Model 구문은 기본적으로 NULL 값에 대해서 별도의 예외 처리를 수행하지 않으며, 따라서 missing cell은 NULL로써 취급됩니다. 또는 모델 전체, 또는 개별 룰에 대해 "IGNORE NAV" 키워드 (NAV는 "non-available value"를 의미합니다)를 적용할 수도 있습니다. IGNORE NAV 옵션은 산술 계산의 경우 NULL과 missing cell을 숫자 '0'으로, 문자열 처리의 경우 빈 문자열(empty string)로 처리하게 합니다.

REFERENCE MODEL

기존 셀에 대한 update 작업, 신규 셀에 대한 insert 작업을 수행하는 다차원 어레이를 Main SQL Model이라 부릅니다. Model 구문은 Main Model 이외에도 읽기 전용으로 활용되는 다차원 어레이를 제공하며 이를 Reference Model이라 부릅니다. Reference Model은 일종의 look-up table로서 기능합니다. Reference Model의 포뮬러는 서로 다른 차원(dimensionality)의 어레이들을 동시에 참조할 수 있습니다.

예를 들어, 수익 예측(profit projection)을 위한 Model에서 tax reference array와 cost reference array를 사용하고, tax는 국가를 기준으로 한 디멘션으로, cost는 제품을 기준으로 한 디멘션으로 참조하는 것이 가능합니다. 또는 환율 계산 과정에서 Reference Model을 이용해 변환 규칙을 구현할 수 있습니다. 이처럼 reference model, "ANY" 와일드 카드, CV() 함수 등을 이용하면 매우 강력하고 유연한 포뮬러를 생성할 수 있습니다.

ITERATIVE MODEL

MODEL 구문의 ITERATE 옵션을 사용하면, 포뮬러를 일정한 횟수만큼 반복 처리(evaluate)하는 것이 가능합니다. 특히 여러 포뮬러가 서로 상호의존적인 형태로 사용되는 경우(다시 말해, 여러 개의 포뮬러를 동시에 수행해야 하는 경우)에 iterative model이 활용됩니다. INTERATE 구문의 인자(argument)를 통해 반복 횟수를 지정할 수 있습니다.

또 포뮬러가 지정된 횟수만큼 반복 처리되기 전에 상황에 따라 종료되도록 하는 조건부 종료 구문을 포함시킬 수 있습니다. 종료 조건은 UNTIL 키워드를 사용하여 정의됩니다. 경우에 따라서는 iteration 과정에서의 셀 값의 변화를 기준으로 조건을 정의해야 할 수도 있습니다. 오라클은 UNTIL 조건에서 iteration 이전의 값과 이후 값을 참조하기 위한 메커니즘을 제공합니다. 또는 현재의 iteration number를 조회할 수도 있습니다.

결론

복잡한 형태의 로우 단위 계산 작업은 데이타베이스 외부(별도 언어 또는 계산 엔진을 사용한 프로그램)에 구현하는 것이 지금까지의 일반적인 관례였습니다. Oracle Database 10*g*는 SQL Model 구문을 이용해 이러한 문제를 해결하고 있습니다. SELECT 문에 포함된 형태로 활용되는 Model 구문은 관계형 데이타베이스를 다차원 어레이로써 취급하고, 매우 정교하고 유연한 notation을 이용하여 각각의 셀을 참조할 수 있게 합니다. 이와 같이 함으로써, 복잡한 SQL join 또는 union 구문의 사용을 억제하고 처리 성능을 최적화할 수 있습니다. 또 포뮬러 간의 논리적인 종속 관계를 자동으로 처리함으로써 계산 작업의 개발 및 유지보수를 단순화하는 것이 가능합니다. 오라클이 Model 구문을 통해 제공하는 병렬 쿼리 처리 성능을 활용하여 엔터프라이즈 수준의 확장성을 확보할 수 있습니다.

오라클이 제공하는 안정된 플랫폼을 기반으로 개선된 OLAP 기능과 Model 구문을 함께 활용할 수 있습니다. 복잡한 계산 기능을 SQL 구문에 구현함으로써 관리성과 데이타 통합성을 향상시키고 DBMS 내부적으로 데이타를 처리한 후 그 결과를 바로 데이타베이스에 반환할 수 있습니다. 새로운 Model 구문은 비즈니스 인텔리전스를 포함하는 다양한 종류의 데이타베이스 애플리케이션에서 사용이 가능합니다.

APPENDIX: MODEL 구문의 EXPLAIN PLAN

explain plan을 통해 오라클이 Model을 처리하는 알고리즘을 확인할 수 있습니다. Model에서 SEQUENTIAL ORDER 포뮬러를 사용하는 경우, "ORDERED"로 디스플레이 됩니다. AUTOMATIC ORDER Model 에서는 포뮬러의 종속 관계에 따라 "ACYCLIC" 또는 "CYCLIC"으로 표시됩니다. 또 ORDERED 또는 ACYCLIC 포뮬러의 좌측 항이 단일 셀을 참조하고, 포뮬러의 우측 항에 sum, count, avg 등의 단순 aggregate 함수가 사용된 경우 추가적으로 "FAST"라 표시됩니다 (이 경우 포뮬러의 처리가 매우 빠르고 효과적으로 수행된다는 의미에서 "FAST" 레이블이 추가됩니다).

ORDERED 프로세싱을 이용한 Explain Plan

```
EXPLAIN PLAN FOR
SELECT SUBSTR(country,1,10) country,
        SUBSTR(prod,1,10) product, year, sales
FROM sales_view
WHERE country IN ('Italy','Brazil')
MODEL RETURN UPDATED ROWS
        PARTITION BY (country) DIMENSION BY (prod, year)
        MEASURES (sale sales)
        RULES SEQUENTIAL ORDER
        (
        sales['Bounce', 2003] =
             AVG(sales)[ANY, 2001] * 1.24,
        sales[prod != 'Y Box', 2000] =
             sales['Y Box', 2000] * 1.25
        );
```

위 쿼리를 수행한 결과의 마지막 다섯 줄이 아래와 같습니다:

```
COUNTRY    PRODUCT      YEAR       SALES
---------- ---------- ---------- ----------
Italy      Bounce     2000       36653.6125
Italy      Mouse Pad  2000       36653.6125
Italy      Music CD-R 2000       36653.6125
Italy      Fly Fishin 2000       36653.6125
Italy      Deluxe Mou 2000       36653.6125
```

위 쿼리가 사용하는 explain plan의 시작 부분이 아래와 같습니다:

```
SELECT STATEMENT
        SQL MODEL ORDERED
```

두 번째 포뮬러의 좌측 항에서 multi-cell reference가 사용되고 있으므로, 오라클은 이 쿼리에 대해 FAST method를 적용하지 않습니다.

ACYCLIC FAST 프로세싱을 이용한 Explain Plan

```
EXPLAIN PLAN FOR
SELECT SUBSTR(country,1,10) country,
SUBSTR(prod,1,10) product, year, sales
FROM sales_view
WHERE country IN ('Italy','Brazil')
MODEL RETURN UPDATED ROWS
PARTITION BY (country) DIMENSION BY (prod, year)
MEASURES (sale sales)
RULES AUTOMATIC ORDER
(
sales['Bounce', 2004] =
sales['Y Box', 2001] * 0.25,
sales['Mouse Pad', 2004] =
sales['Mouse Pad', 2001] / SUM(sales)[ANY, 2001] *
2 * sales['All Products', 2004],
sales['All Products', 2004] = 200000
);
```

쿼리의 실행 결과가 아래와 같습니다:

```
COUNTRY    PRODUCT    YEAR    SALES
-------------------------------------------------------------------------------
Italy      All Produc 2004    200000
Italy      Bounce     2004     20301.8875
Italy      Mouse Pad  2004      1342.86407
Brazil     All Produc 2004    200000
Brazil     Bounce     2004
Brazil     Mouse Pad  2004      2344.7976
```

위 쿼리가 생성한 explain plan의 시작 부분이 아래와 같습니다:

```
SELECT STATEMENT
        SQL MODEL ACYCLIC FAST
```

이 모델은 cyclic한 형태의 종속성을 갖지 않으며 따라서 explain plan은 “ACYCLIC”으로 표시됩니다. 또 쿼리가 앞부분에서 설명한 두 가지 요구조건을 모두 만족하므로 FAST method가 사용되었음을 알 수 있습니다.

ORACLE®

한국오라클(주)
서울특별시 강남구 삼성동 144-17
삼화빌딩
대표전화 : 2194-8000
FAX : 2194-8001

한국오라클교육센타
서울특별시 영등포구 여의도동 28-1
전경련회관 5층, 7층
대표전화 : 3779-4242~4
FAX : 3779-4100~1

대전사무소
대전광역시 서구 둔산동 929번지
대전둔산사학연금회관 18층
대표전화 : (042)483-4131~2
FAX : (042)483-4133

대구사무소
대구광역시 동구 신천동 111번지
영남타워빌딩 9층
대표전화 : (053)741-4513~4
FAX : (053)741-4515

부산사무소
부산광역시 동구 초량동 1211~7
정암빌딩 8층
대표전화 : (051)465-9996
FAX : (051)465-9958

광주사무소
광주광역시 서구 양동 60-37
금호생명빌딩 8층
대표전화 : (062)350-0131
FAX : (062)350-0130

고객에게 완전하고 효과적인
정보관리 솔루션을 제공하기 위하여
오라클사는 전 세계 145개국에서
제품, 기술지원, 교육 및
컨설팅 서비스를
제공하고 있습니다.

제품 구입문의
수신자부담 전화번호 : 080-2194-114
수신자부담 팩스번호 : 080-2194-080
E-Mail 문의 : oracledirect_kr@oracle.com

http://www.oracle.com/
http://www.oracle.com/kr