김도근 delri@ldcc.co.kr | 롯데정보통신 IS사업팀 DBA로 근무하고 있으며 올해 아지아 지역 최초의 OTN ACE로 선정됐다. RDBMS에 있어서 주된 관심사는 성능과 튜닝, 데이터 웨어하우징, 병렬 처리 쪽이다. 취미는 오라클 관련 책 수집과 여행, 축구 보기. 특히 축구라면 사족을 못 쓰는데 요즘은 프리미어 리그에 빠져있다고. 현재는 OCM을 준비 중이다.

오라클은 올 4월 오라클 데이터베이스 10g의 TPC-C 결과를 발표하며 32개 프로세스 부분에서 1분에 160만 트랜잭션을 처리해 세계 신기록을 기록했다고 밝힌 바 있다. 이러한 수치가 우리에게 의미하는 바는 무엇이며 오라클 10g가 다른 DBMS 제품과 차별화되는 점은 무엇인지 성능 측면에서 살펴본다.

성능 항목별 DBMS 3종 비교 분석

database

많은 DBMS 업체들이 자사 제품 성능의 우수함을 주장하기 위해 공신력 있는 기관의 벤치마크 결과를 인용한다. 이때 많이 근거로 제시되는 것이 바로 가장 객관적이라고 알려진 TPC(Transaction Processing Performance Council)이다. TPC는 TPC-C와 TPC-D라는 기준을 가지고 지속적으로 벤치마크 결과를 발표하는데, 전자는 TPC에 의해 기획된 OLTP(OnLine Transaction Processing, 실시간 데이터 처리) 벤치마크 테스트로 여기서 세계 신기록을 수립했다는 것은 32-프로세스 등 해당 부분에서 가장 좋은 성능을 가진 DBMS임을 의미한다. 마치 수영이나 육상 등의 시합에서 100m를 얼마만에 주파했는지 시간 기록이 있듯이 TPC-C도 처리 속도에 분당(tpmC) 혹은 비용($/tpmC) 대비 DBMS의 기록인 셈이다.

〈표 1〉은 각 데이터 볼륨 사이즈별 TPC-H의 결과치와 시간당 처리 쿼리(QphH) 순으로 상위 3위를 정리한 것이다. TPC-H는 의사결정 벤치마크 결과로, 복잡한 질의를 실행하고 중대한 비즈니스 질문에 대한 답변을 주는 의사결정 지원 시스템을 선택하는 기준으로 활용되곤 한다. 〈표 1〉에서 볼 수 있듯이 100GB에서 QphH의 수치는 MS SQL 서버 2005가 가장 뛰어난 성능을 보인다. 그러나 300GB 이상의 데이터 볼륨의 경우 오라클 데이터베이스 10g의 성능이 가장 뛰어난 처리능력을 보여주는 것을 알 수 있다. 즉 오라클은 중대형으로 올라가면 올라갈수록 진가를 발휘하는 셈이다. 실제로 시장에서는 MS SQL은 중소형 DBMS, 오라클은 대형 DBMS라는 인식이 형성돼 있다.

그렇다면 왜 오라클은 중대형 서버에서 제 성능을 발휘하는 것일까. 지금부터 실제 주요 DBMS와 오라클을 직접 비교해보자. 기능 비교는 DBMS의 성능을 결정하는 핵심 요인을 중심으로 이루어지며 구체적으로는 동시성 모델과 인덱싱, 파티셔닝, 병렬 실행, 클러스터링 등을 살펴볼 예정이다.

1 동시성 모델 : 데이터베이스의 Locking 메커니즘을 비교하기 위한 것으로, 얼마나 많은 사용자가 동시에 같은 데이터 읽기 일관성을 유지하는가가 관건이다.

2 파티셔닝 : 대량의 데이터를 처리하는데 있어서 파티셔닝, 즉 특정 값을 기준으로 데이터를 분할하는 방법

3 병렬 실행 : 멀티 CPU 환경에서 CPU 별로 균등하게 작업을 분배해 전체 처리 성능을 높인다.

4 클러스터링 : 복수의 노드를 단일 노드처럼 처리하여 성능과 고가용성을 높이는 요인이 된다.

오라클 데이터베이스 10g vs. SQL 서버 2005

먼저 오라클의 가장 최신 버전인 오라클 데이터베이스 10g 릴리즈

2와 마이크로소프트(이하 MS)의 SQL 서버 최신 제품인 SQL 서버 2005의 기술적인 차이점을 성능과 확장성 관점에서 비교해보자.

동시성 모델

동시성 모델(Concurrency Model)은 멀티-유저 환경에서 특정 사용자에 의해 수행된 데이터 업데이트가 다른 사용자에게 영향을 미치는지 여부를 알 수 있는 매우 중요한 지표다. 오라클 데이터베이스 10g와 SQL 서버 2005는 동시성 모델의 구현 방법에서 차이를 보이는데 주요 차이점은 〈표 2〉와 같다.

오라클 데이터베이스에서 구현되는 멀티-버전 읽기 일관성(multi-version read consistency)은 예를 들면, 트랜잭션에 의해 업데이트가 발생한 경우 기존 데이터 값은 데이터베이스의 언두(undo) 레코드에 기록이 되기 때문에 트랜잭션이 커밋되기 전까지 언두 레코드에 저장된 이전 버전의 정보를 사용자에게 반환하고 따라서 데이터의 읽기 일관성을 보장한다.

반면 SQL 서버 2005가 기본적으로 제공하는 격리(isolation) 모델은 읽기 작업에 대해 공유 읽기 잠금(shared read lock)을 사용한다. 즉 공유 잠금이 적용된 경우 현재 읽기 작업이 수행되고 있는 데이터에 대한 업데이트가 불가능하다. 이러한 모델은 읽기/쓰기 작업이 동시에 발생하는 환경에서 동시 요청을 처리하는데 성능상 불리할 수밖에 없다. 또한 애플리케이션이 점유하는 잠금의 수가 점차 증가함에 따라 잠금 에스컬레이션(lock escalation, 잠금의 확대, 예를 들어 row 레벨 락에서 테이블 락으로 확대되는 현상)이 발생해 동시성이 한층 더 제약되고 데드락(두 세션이 각각 상대방에

〈표 1〉 볼륨 사이즈별 TPC 상위 3위

100GB 결과

기업	시스템	vQphH	가격/QphH	System Availability	데이터베이스	운영체제	Date Submitted	클러스터
hp	HP ProLiant DL585 G1 4P	12,600	9.43$	11/07/05	MS SQL 서버 2005 엔터프라이즈 x64 에디션	MS 윈도우 서버 2003 엔터프라이즈 x64 에디션	11/04/05	N
IBM	IBM e서버 325	12,216	70.68$	11/08/03	IBM DB2 UDB 8.1	수세 리눅스 엔터프라이즈 서버 8	07/29/03	Y
썬	SunFire V890	10,487	46.29$	08/15/05	썬 사이베이스 IQ 12.6 싱글	썬 솔라리스 10		

300GB 결과

기업	시스템	vQphH	가격/QphH	System Availability	데이터베이스	운영체제	Date Submitted	클러스터
hp	HP BladeSystem ProLiant BL25p Cluster 8P DC	18,725	27.97$	11/11/05	오라클 10g 엔터프라이즈 에디션 R2 w/ Partitioning	레드햇 엔터프라이즈 리눅스 4 ES	11/11/05	Y
hp	HP BladeSystem ProLiant BL25p Cluster 8P	13,284	34.20$	10/31/05	오라클 데이터베이스 10g 릴리즈 2 엔터프라이즈 에디션	레드햇 엔터프라이즈 리눅스 4 ES	09/16/05	Y
IBM	IBM e서버 325	13,194	65.44$	11/08/03	IBM DB2 UDB 8.1	수세 리눅스 07/29/03 엔터프라이즈 서버 8	07/29/03	Y

1000GB 결과

기업	시스템	vQphH	가격/QphH	System Availability	데이터베이스	운영체제	Date Submitted	클러스터
hp	HP Integrity Superdome 엔터프라이즈 서버	68,100	59.00$	01/18/06	오라클 데이터베이스 10g R2 엔터프라이즈 에디션 w/Partitioning	HP UX 11.i V2 64비트	08/08/05	N
IBM	IBM e서버 xSeries 346	53,451	32.80$	02/14/05	IBM DB2 UDB 8.2	수세 리눅스 엔터프라이즈 서버 9	02/14/05	Y
hp	HP ProLiant DL585 Cluster 48P	35,141	59.93$	10/21/04	오라클 10g RAC with Partitioning	레드햇 엔터프라이즈 Linux AS 3	10/22/04	Y

10000GB 결과

기업	시스템	vQphH	가격/QphH	System Availability	데이터베이스	운영체제	Date Submitted	클러스터
썬	Sun Fire E25K server	108,099	53.80$	01/23/06	오라클 10g 엔터프라이즈 에디션 R2 w/ Partitioning	썬 솔라리스 10	11/29/05	N
IBM	IBM e서버 p5 575	104,100	61.17$	08/15/05	IBM DB2 UDB 8.2	IBM AIX 5L V5.3	05/20/05	Y
hp	HP Integrity Superdome 엔터프라이즈 서버	86,282	161.24$	04/06/05	오라클 데이터베이스 10g 엔터프라이즈 에디션	HP UX 11.i V2 64비트	10/07/04	Y

출처 | TPC, www.tpc.org/tpch/results/tpch_results.asp

대해서 lock을 잡고 있는 상태)으로 연결될 가능성도 있다. 이러한 문제 때문에 MS SQL 서버 2005에서는 이러한 문제에 대응하기 위해 구문 레벨 읽기 일관성(read committed with snapshots), 트랜잭션 레벨 읽기 일관성(snapshot isolation) 등 두 가지 격리 수준을 추가했다.

이 두 가지 격리 수준은 각각 오라클에서 예전부터 지원해 온 READ COMMITTED와 SERIALIZABLE 격리 수준에 대응된다. 이 두 가지 격리 수준에서는 특정 읽기 작업이 동일한 데이터에 접근하는 다른 읽기/쓰기 작업을 블로킹하지 않으며 쓰기 작업 역시 읽기 작업을 블로킹하지 않는 것이 특징이다. 먼저 트랜잭션 레벨 읽기 일관성은 로우 버저닝(row versioning)을 기반으로 하고 있다. 이는 커밋된 데이터 로우를 포함하고 있는 여러 버전들의 링크드 체인(linked chain), 즉 원래의 데이터 위치를 추적함으로써 읽기 일관성을 보장하는 방법이다. 링크드 체인은 tempdb라는 임시 스토어드 프로시저(temp stored procedure)와 기타 임시 작업을 위한 데이터베이스 저장공간에 위치한 별도의 버전 저장소(version store)에 저장된다.

이러한 SQL 서버 2005의 트랜잭션 레벨 읽기 일관성은 혁신적인 기술로 보기 힘들다. SQL 서버 2000에서는 지원되지 않았기 때문에 이전 버전보다 개선된 것은 사실이나 오라클은 이미 오래 전부터 멀티-버전 읽기 일관성을 기본으로 지원해왔기 때문이다. 이 외에도 SQL 서버 2005는 동시성 모델에 있어서 다음과 같은 한계들을 갖는다.

1 관리자는 데이터베이스 레벨에서 명시적으로 설정한 경우에만 read-committed with snapshot 또는 snaption isolation이 활성화된다(기본적으로는 성능상의 이유 때문에 'disable' 되어 있다).

2 기존에 운영 중이던 SQL 서버 애플리케이션의 경우 이 모드를 구현하려면 SQL 서버 애플리케이션을 오라클 환경으로 이전하는 것과 동등한 수준의 수정 작업이 요구된다. 왜냐하면 SQL 서버 2000에서는 이러한 기능이 지원되지 않았기 때문이다. 읽기 잠금을 이용하는 애플리케이션에 멀티-버전 읽기 일관성을 적용하기 위해서는 일정 수준의 재설계와 재개발 작업이 불가피하다.

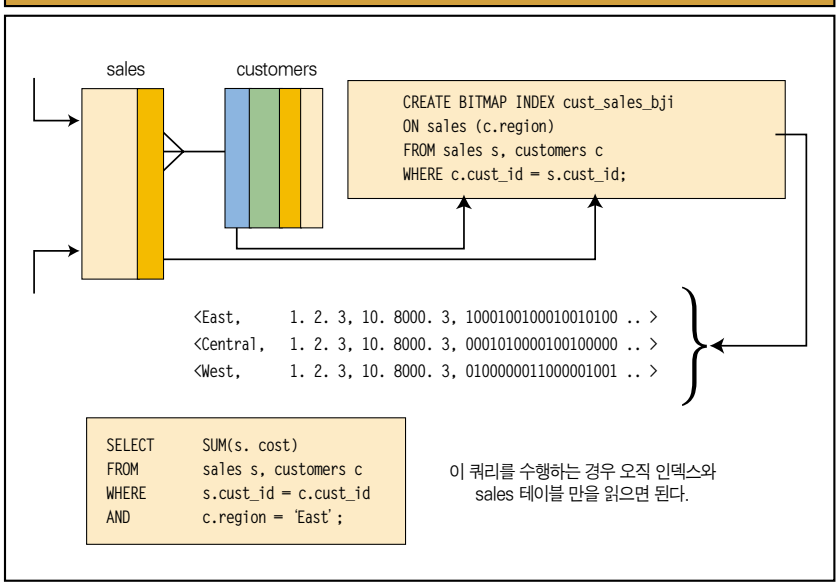
<표 2> 오라클 10g와 MS SQL 서버 2005의 동시성 모델 비교

	오라클 데이터베이스 10g	SQL 서버 2005
멀티 버전 읽기 일관성 (Multi-version read Consistency)	항상 가능	디폴트 아님. 기능의 사용을 위해 활성화해야 함
로우 레벨 잠금의 에스컬레이션 잠금 (Non-escalating row-level locking)	지원	지원 안됨 (Locks escalate)

<표 3> 인덱스 유형 비교

인덱스 유형	오라클 데이터베이스 10g	SQL 서버 2005
B-트리 인덱스	지원	지원
B-트리 클러스터 인덱스	지원	지원하지 않음
해시 클러스터 인덱스	지원	지원하지 않음
리버스 키 인덱스	지원	지원하지 않음
비트맵 인덱스	지원	지원하지 않음
비트맵 조인 인덱스	지원	지원하지 않음
기능 기반 인덱스	지원	지원하지 않음. 계산된 컬럼(computed column)에 대해서도 인덱스를 생성할 수 있지만 해당 컬럼이 테이블 내에 실제로 존재하고 있어야 함.
도메인 인덱스	지원	지원하지 않음
IOT	지원	지원(clustered index)

<그림 1> 비트맵 조인 인덱스



인덱싱

인덱스가 데이터에 대한 신속한 접근을 제공하기 위해 생성되는 중요한 기능 중에 하나라는 것은 데이터베이스를 사용하는 사람들에게는 상식적인 이야기다. 인덱스를 이용하면 디스크 I/O 작업을 크게 줄이고 데이터 인출 성능을 개선할 수 있으며 성능 향상을 기대할 수 있다. 그렇다면 오라클 데이터베이스 10g와 MS SQL 서버 2005의 인덱싱은 어떠한 차이가 있을까. <표 3>은 두 제품이 지원하는 인덱싱 메커니즘의 차이를 요약한 것이다.

오라클과 SQL 서버 2005는 모두 고전적인 B-트리 인덱스 구조를 지원한다. B-트리 인덱스는 순차적으로 정렬된 키 값을, 실제 값이 저

장된 테이블 로우의 저장 위치와 연계한 형태로 구성된다. B-트리 인덱스는 별도의 인덱스 영역에 키 값을 기준으로 정렬되어 있고 이 인덱스 영역은 실제 데이터의 위치 정보(RowID)를 가지고 있다. 또한 두 제품 모두 IOT(Index-Organized Table)을 지원한다(MS는 clustered index라는 용어를 사용한다). IOT는 테이블 로우를 프라이머리 키 인덱스의 리프 노드에 저장하고 있기 때문에 프라이머리 키를 기준으로 한 조건 및 영역 검색에서 뛰어난 성능을 보여준다. IOT의 대표적인 성격은 모든 테이블의 데이터를 인덱스처럼 저장하는 것이다. 즉 인덱스 입력 항목의 두 번째 요소로 행의 RowID를 가지지 않고 실제 데이터 행이 B-트리 인덱스에 저장된다.

게다가 오라클은 스태틱 비트맵 인덱스(static bitmap index)와 비트맵 조인 인덱스(bitmap join index)를 추가로 지원한다. 이 두 가지 인덱스는 데이터 웨어하우징 환경의 로드/쿼리 작업에서 좋은 성능 효과를 보여준다. 비트맵 인덱스는 RowID와 값에 대해 BIT 값으로 저장을 함으로써 나이, 성별, 지역처럼 전체 레코드 건수에 비해 카디널리티(선택도)가 낮은 속성들과 OR, AND 연산시에 효과적으로 사용할 수 있는 인덱스 구조이다. 또한 오라클 9i부터 지원되는 비트맵 조인 인덱스는 두 개 이상의 테이블에 조인 인덱스를 생성함으로써 질의 처리를 위한 조인에서 오는 부하를 피하고 그만큼 성능 향상을 가져올 수 있다.

비트맵 인덱스는 테이블 로우의 저장 위치 목록 대신 각 키 값에 대한 비트맵(또는 비트 벡터)을 사용한다. 비트맵의 각 비트는 테이블의 로우에 대응한다. 테이블의 로우가 키 값을 포함하고 있는 경우에 해당 비트가 설정된다. 로우의 저장 위치를 저장하는 방식과 비교했을 때 비트맵 표현 방식은 매우 많은 비용 절감 효과를 제공한다. B-트리 인덱스는 실제로 조건이 비교되는 컬럼 값에 대한 테이블의 원시 값과 Row의 물리적인 주소인 RowID를 인덱스 블럭에도 저장하므로 데이터의 중복 저장에 따른 공간낭비가 발생한다. 반면 비트맵 인덱스는 저장공간에 인덱스 컬럼 값이 아닌 1과 0의 비트값이 저장되고, 스캔에 의한 데이터 추출이 아닌 비트 연산에 의한 데이터 추출을 하기 때문에 성능을 높일 수 있다. 특히 선택도(cardinality)가 낮은 데이터가 사용되는 경우 효과적이다.

비트맵 인덱스는 AND, OR 등 고속의 불리언(Boolean) 연산을 통해 서로 다른 인덱스의 비트맵을 조합하는 형태로도 활용된다. 여러 개의 조건에 대한 연산을 수행하기 위해 각 조건에 대응되는 인덱스들을 WHERE절 내에서 효과적으로 조합한다. WHERE절 내의 모든 조건을 만족하지 않는 로우는 테이블에 대한 액세스가 수행되기 전에 필터링되며 상황에 따라 극적인 성능 개선도 기대할 수 있다.

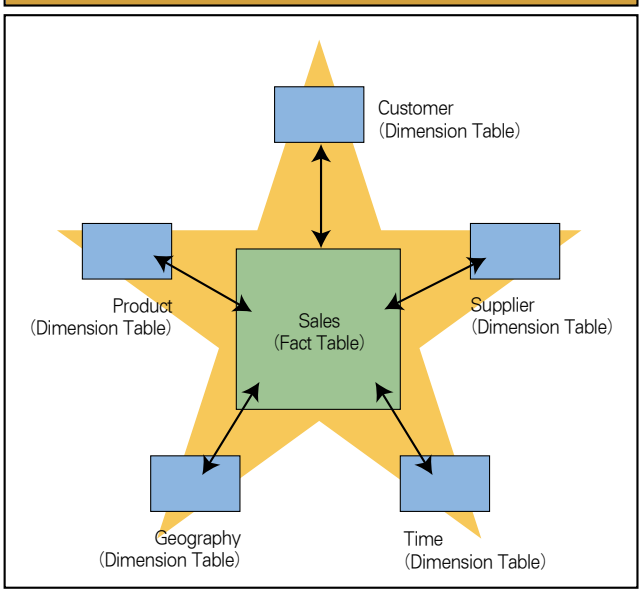
오라클 데이터베이스에서는 IOT에 대한 비트맵 인덱스를 생성하고, IOT를 데이터 웨어하우징 환경을 위한 팩트 테이블(fact table, 다차원 모델에서 중심이 되는 테이블)로 활용하는 것이 가능하다. 비트맵 조인 인덱스는 두 개 이상의 테이블을 조인(join)하기 위해 사용되는 비트맵 인덱스이다. 이를 이용하면 실제 테이블을 조인할 필요가 없으며 제약 조건을 미리 실행함으로써 실제로 조인되는 데이터의 양을 크게 줄일 수 있다. 또한 비트 단위 연산을 통해 Bit map join index를 이용하는 쿼리의 실행 속도도 개선할 수 있다.

〈그림 1〉은 비트맵 조인 인덱스의 대표적인 사례다. 조건절을 통해 Customer 테이블과 Sales 테이블 사이의 조인을 미리 계산하는(precomputation) 역할을 하게 되므로 두 개의 테이블 질의 시 비트맵 조인 인덱스를 통하게 되면 하나의 테이블에만 접근하면 된다. 게다가 비트맵 조인 인덱스는 다수의 디멘션 테이블을 포함하고 있으므로 비트 단위 연산이 필요치 않다(실제로 단일 테이블에 비트맵 인덱스의 형태로 스타 스키마를 적용한 경우에는 비트 단위 연산이 반드시 필요하다). 다양한 형태의 스타 스키마(Star Schema, 데이터 웨어하우징에서 복잡한 정보를 모델링하는 표준형 기술로, 중심이 되는 Fact 테이블을 중심으로 디멘전(dimension) 테이블이 붙어있는 형태이다)에 대한 쿼리 테스트를 수행해 보면 비트맵 조인 인덱스를 사용한 쿼리가 실제 성능 개선 효과를 제공한다는 사실을 확인할 수 있다.

파티셔닝

파티셔닝(Partitioning)은 테이블, 인덱스 등 대규모 데이터베이스 구조를 더 작고 관리하기 쉬운 단위로 분해하는 기능이다. 주로 관리성과 가용성을 개선하기 위해 활용되지만 성능 측면에서도 몇 가

〈그림 2〉 스타 스키마 모델



지 혜택을 제공한다.

파티셔닝은 애플리케이션 시나리오 별로 다양한 파티셔닝 테크닉을 고려할 수 있다. 예를 들어 레인지 파티셔닝(Range Partitioning)은 일정 영역의 컬럼 값을 이용해 로우를 파티션에 맵핑한다. 이 옵션은 히스토리(history) 데이터베이스 즉 이력 데이터에 특히 유용하게 활용되며 데이터 웨어하우스 환경의 롤링 윈도우(rolling window, 주기적으로 새로운 데이터가 추가되면서 오래된 데이터는 데이터 웨어하우스에서 삭제되는 것) 지원을 위한 이상적인 파티셔닝 방법으로 이용되기도 한다.

해시 파티셔닝(Hash Partitioning)은 파티션된 컬럼에 해시 함수를 적용해서 데이터를 분산시키는 방법으로 균일하게 분포된 데이터에 효과적이다. 즉 이력 데이터의 범위 분할 적용에서 나타나는 단점인, 각 범위(Bound)가 포함하는 데이터의 양이 일정하지 않아 분포도가 일정하지 않고 각 파티션의 크기가 다르게 나타나는 점을 개선한다. 이를 통해 일정한 분포를 가진 파티션으로 나누고 균등한 데이터 분포도를 이용한 병렬처리로 성능을 높인다.

리스트 파티셔닝(List Partitioning)은 로우를 파티션으로 맵핑하는 방법을 관리자가 명시적으로 설정할 수 있다. 관리자는 파티셔닝 컬럼을 위한 값의 리스트를 정의하는 방법으로 맵핑 방법을 설정한다. 컴포짓 파티셔닝(Composite partitioning)은 사용자가 다양한 파티셔닝 테크닉을 조합할 수 있도록 지원한다. 첫 번째 방법을 이용해서 테이블을 먼저 파티셔닝한 후 두 번째 방법을 통해 각 파티션을 다시 서브파티션으로 분할하는 것이다. 이때 인덱스는 크게 3가지 종류로 구분할 수 있다.

1 로컬 인덱스(Local Index) : 하부 파티션 테이블과 동일한 파티션 방법을 사용하여 파티셔닝된 테이블에 생성된 인덱스이다. 로컬 인덱스의 각 파티션은 하부 테이블의 특정 파티션에 맵핑된다.

2 글로벌 파티션드 인덱스(Global Partitioned Index) : 테이블의 서로 다른 파티셔닝-키를 이용해 파티션된 테이블이나 파티셔닝되지 않은 테이블에 생성된 인덱스를 가리킨다. 파티셔닝되지 않은 테이블의 인덱스와 동일한 형태로 구성되며 이때 인덱스 구조는 파티셔닝되지 않는다.

〈표 4〉는 오라클과 SQL 서버의 파티셔닝 옵션을 비교한 것이다. 오라클이 지원하는 다양한 파티셔닝 옵션을 확인할 수 있으며 실제 기능에 있어서도 〈표 5〉처럼 차이가 있음을 알 수 있다.

병렬 실행과 클러스터링

SQL 작업은 병렬 실행을 통해 대량의 데이터가 수반되는 작업의 성능을 크게 개선할 수 있다. 특히 의사결정 시스템 또는 데이터 웨어하우스 등의 대규모 데이터베이스에서 데이터 집중적인 작업을 수행할 때 응답시간 개선에 도움이 된다. 예를 들어 오라클 사용자가 SQL문에 대한 병렬 처리를 수행하고자 한다면 오라클 서버는 사용자의 요청에 따라 가용 가능한 CPU 개수 만큼 병렬 처리를 수행한다. 4개의 CPU를 가진 서버에서의 병렬 처리를 수행한다면 3개의 CPU에서 실 SQL문에 대해 균등하게 작업을 할당하여 처리하고 나머지 1개의 CPU에서 이를 병합하는 작업을 수행한다. 오라클 데이터베이스는 파티셔닝된 데이터베이스 오브젝트 또는 파티셔닝되지 않은 데이터베이스 오브젝트에 액세스하는 과정에서 INSERT, UPDATE, DELETE, MERGE 등의 구문을 병렬적으로 실행한다. 반면 SQL 서버 2005의 INSERT, UPDATE, DELETE 구문은 순차적으로 실행된다.

클러스터는 사설 네트워크를 통해 연결된 다수의 독립적인 서버 또는 노드들이 마치 하나의 시스템인 것처럼 협력하여 동작하는 환경을 의미한다. 단일 노드 시스템이 갖는 확장성의 한계를 극복하고 대형 서버의 성능을 뛰어넘는 부하 처리를 가능케 한다. 오라클 RAC(Real Application Cluster)이 바로 이런 역할을 지원하는 솔루션으로 DBMS에 대한 요구사항이 증가함에 따라 단순히 노드를 추가함으로써 확장할 수 있는 것이 특징이다.

SQL 서버 2000은 제품문서에 명시된 것처럼 이러한 형태의 클러스터링을 지원하지 않는다. 이것은 SQL 서버 2005에서도 마찬가지인데 대신 'Federated Database Server'라는 새로운 방식을 지원한다. 두 접근법은 매우 큰 차이를 갖고 있어 애플리케이션의 성능과 확장성에 큰 영향을 미칠 수 있다.

Federated Database Server는 독립적인 데이터베이스들로 구성되며 공통 데이터 딕셔너리와 글로벌 인덱스를 지원하지 않는다. 이 때문에 성능과 확장성 면에서 많은 제약이 따른다. 또한 SQL 서버 2005의 접근법은 실제 애플리케이션 환경에서 적용이 매우 힘들다. 실제로 SAP, 피플소프트 등의 비즈니스 애플리케이션은 일반적으로 수천 개의 테이블로 구성되는데 SQL 서버 2005의 Federated Database Server를 구현하려면 모든 테이블을 파티셔닝하거

〈표 4〉 오라클과 SQL 서버의 파티셔닝 옵션 비교

파티셔닝 옵션	오라클 데이터베이스 10g 릴리즈 2	SQL 서버 2005
Range	지원	지원
Hash	지원	지원하지 않음
List	지원	지원
Composite	지원(Range-hashRange-list)	지원하지 않음
Local Index	지원	지원
Global Index	지원	지원

〈표 5〉 오라클 SQL 서버의 최대 파티션 수

	오라클 데이터베이스 10g 릴리즈 2	SQL 서버 2005
테이블 당 최대 파티션 수	1024K(100만 개 이상)	1000

나 각 노드로 복제해야 한다. 이처럼 거대한 애플리케이션을 포팅하는 것은 복잡할뿐만 아니라 많은 비용을 필요로 한다.

반면 오라클 RAC는 상대적으로 포괄적인 애플리케이션 호환성을 제공한다. 대표적인 기업용 애플리케이션들을 효과적으로 확장할 수 있으며 클러스터링 환경을 위한 커스터마이즈 작업도 필요치 않다. 즉 데이터 액세스 패턴이 데이터 블럭 핑을 감소 또는 어렵게 하더라도 애플리케이션을 분할할 필요가 없다. 단일 노드의 오라클 서버에서 확장성 있는 애플리케이션은 멀티 노드의 RAC 상에서도 확장성이 있다. 이 때문에 기존 애플리케이션을 재설계하거나 코드를 수정할 필요가 없으며 애플리케이션을 명시적으로 분할하거나 데이터를 파티셔닝할 필요도 없다.

또한 SQL 서버는 파티션을 실제로 소유한 노드만이 해당 파티션에 대한 읽기 작업을 수행할 수 있다. 프로세싱 파워는 테이블이 포함된 노드의 프로세싱 파워로 한정된다. 그러나 오라클 데이터베이스에서는 이러한 제약이 존재하지 않으며 심지어 전체 시스템의 프로세싱 파워, 다시 말해 모든 병렬 실행 서버의 리소스를 이용해 하나의 파티션에 대한 처리 작업을 수행하는 것도 가능하다.

오라클 데이터베이스 10g vs. IBM DB2 UDB

지금까지 최근에 새로운 버전을 발표한 MS SQL 서버 2005와 오라클 데이터베이스 10g에 대해서 비교해 보았다. 이번엔 오라클과 IBM의 DB2를 역시 성능 관점에서 비교해보자. 두 제품을 본격적으로 비교하기에 앞서 여기서 사용되는 DB2, DB2 UDB 등의 용어는 모두 DB2 UDB ESE(엔터프라이즈 서버 에디션) Version 8.2를 가리킨다. 또한 오라클, 오라클 데이터베이스, 오라클 데이터베이스 10g는 모두 오라클 데이터베이스의 최신 버전인 오라클 데이터베이스 10g 엔터프라이즈 에디션 릴리즈 2를 의미한다.

동시성 모델

비교 항목은 역시 앞서 진행했던 것과 동일하다. 먼저 동시성 모델을 보면 오라클 데이터베이스와 IBM DB2는 동시성 컨트롤의 구현 방식에서 〈표 6〉과 같은 차이를 보인다. 오라클의 경우 쿼리와 업데이트가 동시에 발생하는 혼합형 워크로드 환경을 지원하며 쓰기 작업이 읽기 작업을 차단하거나 읽기 작업이 쓰기 작업을 차단하는 상황이 발생하지 않는다. 반면 DB2는 사용자가 정확성(accuracy)과 동시성(concurrency)의 두 가지 중 하나를 양자택일할 수 밖에 없다. 즉 읽기 일관성을 보장하기 위해 쓰기 작업을 블로킹하거나 쓰기 작업을 차단하지 않는 대신 더티 리드(dirty read)로 인한 부정확한 결과를 감수해야 한다. 여기서 더티 리더란 언커밋 리더라고도 한다. 사용자가 변경시키고 있는 commit되지 않은 데이터를 다른 사용자가 읽는 현상을 말한다. 예를 들어서 A라는 사용자가 공유되어 있는 문서파일을 저장하지 않고 작성 중에 B 사용자가 이를 열어서 보는 현상을 들 수 있다.

오라클의 기본적인 아키텍처는 대용량 트랜잭션을 고려해 설계돼 있다. 이는 오라클이 특허를 보유한 논-에스컬레이팅 로우-레벨 락킹(non-escalating row-level locking) 기능(row에 대한 잠금을 가지는 lock이 이 잠금의 개수를 줄이기 위해서 상위 테이블 lock 등으로 확대시키지 않는 현상) 지원이 있기에 가능한 것인데, 애플리케이션에 연결되는 사용자의 수가 늘어나고 처리해야 하는 트랜잭션의 양이 증가해도 오라클 데이터베이스가 일관된 성능을 유지할 수 있는 것도 이 때문이다. Winter Corporation의 조사 결과 전 세계적으로 가장 규모가 큰 상위 10개 유닉스 데이터베이스가 모두 오라클 기반으로 운영되고 있는 것 역시 효율적인 동시성 모델에 기인한 바가 크다.

DB2의 경우 락 정보의 추적을 위해 사용되는 메모리 구조의 용량이 제한되어 있기 때문에 트랜잭션 규모가 증가할 경우 리소스 사용량을 줄이기 위한 방편으로 로우 락(row lock)을 테이블 락(table lock)으로 에스컬레이션한다. 따라서 불필요한 경합이 발생하고 처리 성능의 저하가 일어날 수 있다.

오라클과 DB2 데이터베이스의 구현 방식은 멀티유저 환경에서 일반적으로 발생하는 다음과 같은 문제들을 방지하는 메커니즘에서도 큰 차이를 보인다. 참고로 여기서 non-repeatable read는 해당 트랜잭션 중 바로 전에 읽은 데이터가 다시 읽고 난 후 변경된 상태로, 첫 읽기 후 해당 데이터가 다른 트랜잭션에 의해 커밋된 상태를 의미한다. 또한 Phantom Read는 해당 트랜잭션 중 조건을 만족하는 튜플들을 리턴하는 쿼리를 재실행한 후 변경된 튜플들이 리턴될 때를 가리킨다.

◆ 트랜잭션이 커밋되지 않은 변경사항을 읽는 시점에 더티 리드(dirty read) 또는 언커밋트 리드(uncommited read)가 발생한다.

◆ 트랜잭션이 방금 전에 읽어 들인 데이터를 다시 읽는 과정에서 해당 데이터

〈표 6〉 오라클과 DB2의 동시성 모델 기능 차이

오라클 데이터베이스 10g	DB2 UDB
멀티-버전 읽기 일관성 (multi-version read consistency)	지원되지 않음
리드 락이 사용되지 않음	더티 리드를 방지하려면 리드 락이 필요
더티 리드를 사용하지 않음	리드 락을 사용하지 않는 경우 더티 리드 발생
로우-레벨 락(low-level locking)이 에스컬레이션 되지 않음	락의 에스컬레이션 발생
읽기 작업은 쓰기 작업을 블로킹하지 않음	읽기 작업이 쓰기 작업을 블로킹
쓰기 작업은 읽기 작업을 블로킹하지 않음	쓰기 작업이 읽기 작업을 블로킹
높은 부하에서 데드락이 전혀 발생하지 않음	높은 부하에서 데드락으로 인한 심각한 문제가 발생할 수 있음

가 다른 커밋된 트랜잭션에 의해 수정되거나 삭제됐음을 확인했을 때 non-repeatable read가 발생한다.

◆ 트랜잭션이 검색 조건을 만족하는 일련의 로우를 반환하는 쿼리를 2차례 반복 실행하고 다른 애플리케이션에 의한 INSERT 작업으로 인해 두 번째 쿼리에서 (첫 번째 쿼리에서는 반환되지 않은) 추가적인 로우가 반환되었을 때 phantom read가 발생한다.

오라클은 트랜잭션에 업데이트가 발생할 경우 기존 데이터는 데이터베이스의 언두 레코드에 저장된다. 데이터베이스가 읽기 작업을 수행하는 동안 데이터 변경을 방지하기 위해 또는 쿼리가 커밋되지 않은 변경 데이터를 읽는 것을 방지하기 위해 오라클은 락을 사용하는 대신 언두 레코드에 저장된 기존 정보를 이용하여 테이블 데이터에 대한 읽기 일관성을 확보한다. 반면 DB2는 멀티-버전 읽기 일관성을 제공하지 않는다. 대신 다양한 레벨의 격리 모델을 통해 읽기 잠금(read lock)을 사용하거나 더티 리드를 허용하는 방법을 사용한다. 읽기 잠금은 동시 수행 중인 트랜잭션에 의해 변경 중인 데이터를 읽을 수 없도록 차단하기 때문에 다수의 읽기/쓰기 작업이 동시에 발생하는 환경에서 서비스 동시 요청을 처리하는 능력이 제한될 수밖에 없다.

오라클이 지원하는 로우-레벨 락은 정교한 수준의 락 관리 방식으로 높은 데이터 동시성을 제공한다. 로우-레벨 락은 테이블의 특정 로우에 대한 업데이트 과정에서 해당 로우만을 잠금 처리하며 다른 모든 로우는 동시 작업이 가능하다. 오라클은 디폴트 동시성 모델로 로우-레벨 락을 사용하며 락 정보를 실제 로우 내부에 저장하고 이를 통해 데이터베이스의 로우 또는 인덱스 엔트리 숫자만큼 로우-레벨 락을 관리할 수 있게 해 데이터 동시성을 높였다.

DB2 역시 로우-레벨 락을 디폴트 동시성 모델로 지원한다. 그러나 DB2의 이전 버전에서는 로우-레벨 락이 기본 잠금 모드가 아니었고 후에 로우-레벨 락을 추가적으로 지원하는 과정에서 '락 리스트(lock list)' 라는 별도의 메모리 구조가 필요하게 됐다. 이 메모리는 제한된 용량을 가지고 있으며 이 때문에 데이터베이스에서 지원할 수 있는 최대 락의 숫자 또한 제약된다. 이 때문에 애플리케이션과 트랜잭션 볼륨에 접근하는 사용자의 수가 증가하면 DB2는 메모리 절약을 위해 로우-레벨 락을 테이블 락(table lock)으로 에스컬레이션한다. 이는 결국 데이터에 동시 접근할 수 있는 사용자의 수가 줄어들게 됨을 의미하는데 그만큼 대기 시간이 길어질 가능성이 있다.

실제로 DB2 매거진의 한 기사(www.db2mag.com/db_area/archives/1999/q2/99sp_yevich.shtml)는 '락 에스컬레이션은 ERP 환경에서 가장 심각한 성능 저하 요인의 하나로 꼽힌다' 고 지적하고 락 에스컬레이션을 비활성화할 것을 권고한 바 있다(그러나 이러한 작업은 OS/390 플랫폼의 DB2에서만 가능하며 유닉스와 윈도우 기반 DB2에서는 비활성화가 불가능하다).

<표 7> 오라클과 DB2의 인덱싱 기능 비교

기능	오라클	DB2
Stored Compressed Bitmap Indexes	지원	-
비트맵 조인 인덱스	지원	-
다이나믹 비트맵 인덱스	지원	지원
IOT	지원	-
리버스 키 인덱스	지원	-
기능 기반 인덱스	지원	부분적으로 지원

인덱싱

오라클과 DB2는 모두 고전적인 B-트리 인덱싱 메커니즘을 지원한다. 이미 살펴본 것처럼 오라클은 이 밖에도 스태틱 비트맵 인덱스와 비트맵 조인 인덱스를 지원할 뿐만 아니라 여러 개의 파티션에 대한 글로벌 인덱스를 지원해 OLTP 환경의 파티셔닝된 테이블에서 유용하다. 반면 DB2는 B-트리 인덱스와 다이내믹 비트맵 인덱스 만을 지원한다. 두 제품의 인덱싱 기능 차이는 〈표 7〉과 같다.

오라클의 경우 인덱스는 대상 테이블의 하나 또는 그 이상의 컬럼에 대한 함수로 생성될 수 있다. 함수 기반 인덱스(function-based index)는 함수 또는 표현식의 결과를 미리 계산해 인덱스에 저장하며 B-트리 인덱스 또는 비트맵 인덱스로 생성할 수 있다. DB2의 generated column 기능의 경우 표현식을 기반으로 생성된 컬럼의 값을 유도한 결과가 인덱스에 저장된다. 그러나 유도된 값을 테이블 형태로 저장한다는 점에서 오라클의 함수 기반 인덱스만큼 효율적이지 못하다.

IOT는 테이블 로우를 프라이머리 키 인덱스에 저장하며 프라이머리 키에 대한 조건과 영역 검색을 수반하는 쿼리에서 높은 성능을 나타낸다. IOT를 이용하는 경우 중요 컬럼이 테이블과 프라이머리 키 인덱스에 이중으로 저장되지 않으므로 공간을 절약할 수 있고 일반적인 테이블에서 로우의 주소를 저장하고 인덱스 값과 로우 데이터에 대한 링크를 제공하는 용도로 사용되는 RowID를 위해 추가적인 공간을 할당할 필요도 없다. IOT는 기본키 인덱스 구조로 모든 데이터를 저장하므로 기본키 인덱스 스캔 만으로 모든 작업을 종료할 수 있다. 일반 테이블은 기본키를 사용하여 인덱스 스캔하여 해당 테이블로 랜덤 액세스를 수행하므로 IOT보다 성능 저하가 발생할 수 있다. 따라서 빠른 조회를 요구하는 OLTP 업무에서 IOT는 클러스터 테이블과 더불어 그 성능을 발휘한다.

IOT는 RowID pseudo-column, LOB, 2차 인덱스, range/hash 파티셔닝, 오브젝트 지원, 병렬 쿼리 등 일반적인 테이블에서 지원되는 모든 기능을 지원한다. IOT에 비트맵 인덱스를 생성하고 데이터 웨어하우징 환경의 팩트 테이블로 활용하는 것도 가능한데 이러

한 기능은 오라클 데이터베이스 10g에서만 제공되는 기능이다.

파티셔닝

이미 살펴본 것처럼 파티셔닝은 대규모 데이터베이스를 관리하기 쉬운 단위로 분할하기 위해 사용되며 파티션 프루닝(partition pruning)이라 불리는 테크닉을 활용하는 경우 성능의 개선을 기대할 수 있다. 파티션 프루닝은 필요한 데이터가 존재하는 파티션에 대해서만 작업이 실행되도록 제한하는 기능을 말한다. 작업 과정에서 필요한 데이터를 포함하지 않은 파티션들은 검색 과정에서 제외된다. 이를 통해 디스크로부터 인출되는 데이터의 양과 프로세싱 시간을 크게 줄이고 쿼리 성능과 리소스 사용률을 개선할 수 있다.

파티셔닝 환경에서 partition-wise join 테크닉을 사용해 멀티-테이블 조인 작업의 성능을 개선할 수도 있다. 이것은 두 개의 테이블이 함께 조인되고 조인 키(join key)를 기준으로 두 테이블이 파티셔닝된 경우에 적용되는데 대규모 조인 작업을 각 파티션 별로 작은 크기의 조인 작업으로 분할하고 전체 조인 작업에 소요되는 시간을 단축하는 효과가 나타난다. 따라서 순차/병렬 작업 환경에서 성능 개선 효과를 기대할 수 있다. 마지막으로 파티셔닝 환경에서 DML 작업의 병렬 실행 기능을 활성화함으로써 데이터 집중적인 작업이 수반되는 대규모 의사결정 시스템이나 데이터 웨어하우스 환경의 응답시간을 단축할 수 있다.

이미 오라클에서 제공하는 파티셔닝은 살펴보았으므로 여기서는 DB2 UDB의 파티셔닝을 중점적으로 살펴보자. 〈표 8〉은 두 제품의 파티셔닝 옵션을 비교한 것이다. DB2는 해시 파티셔닝만을 지원(ftp.software.ibm.com/ps/products/db2/info/vr8/pdf/letter/db2s2e80.pdf)하기 때문에 오라클과 차이가 있음을 알 수 있다. 레인지 파티셔닝 또는 리스트 파티셔닝과 달리 해시 파티셔닝은 일부 쿼리에 대해 파티션 프루닝을 지원하지 않는다. 따라서 데이터 웨어하우스를 최신 상태로 유지하려면 새로운 데이터를 로드하고 오래된 데이터를 삭제하는 작업을 지속적으로 반복해야 하는 번거로움이 있다. 해시 파티셔닝이 적용된 DB2 환경에서는 전체 파티션에 대한 재분배 작업이 불가피하며 결과적으로 새로운 데이터를 로드하는데 더 많은 시간이 소요되고 데이터 재분배 과정의 테이블 잠금으로 인해 가용성이 저하될 가능성이 있다.

또한 DB2는 테이블과 인덱스 간의 'equi-partitioning'(인덱스가 같은 칼럼에 대해 같은 값으로 파티션되어 있는 것)을 요구하며 따라서 글로벌 인덱스의 생성이 불가능하다. 이러한 제약은 개별 레코드에 대한 효율적인 액세스를 위해 글로벌 인덱스를 빈번하게 활용해야 하는 OLTP 환경에서 심각한 문제를 야기할 가능성이 있다. 이처럼 DB2 기반의 애플리케이션 설계 과정에서는 파티셔닝 환경의 유연한 인덱스 구성이 어렵다(www-128.ibm.com/ developerworks/db2/library/techarticle/dm-0405wilkins/ index.html).

클러스터

RAC은 오라클 데이터베이스 10g에 포함된 하드웨어 클러스터 지원 옵션이다. 이는 공유 디스크(shared disk) 방식을 채택하고 있는데 공유 디스크 아키텍처에서 데이터베이스 파일은 다수의 노드에 의해 논리적으로 공유되며 각 시스템의 인스턴스는 모든 데이터에 대한 접근이 허용된다. RAC 역시 오라클이 특허를 보유한 캐시 퓨전(Cache Fusion) 아키텍처를 기반으로 하고 있다. 캐시 퓨전은 상호 연결된 캐시를 이용해 OLTP, DSS, 패키지 애플리케이션 등 다양한 애플리케이션에 대한 데이터베이스 클러스터 기능을 지원한다. 사용자의 쿼리는 로컬 캐시 또는 다른 노드의 원격 캐시를 통해서도 처리할 수 있으며 업데이트 작업 과정에서 로컬 노드는 다른 클러스터 노드의 데이터베이스 캐시로부터 필요한 블럭을 직접 가져오므로 동기화를 위한 별도의 읽기/쓰기 작업을 수행할 필요가 없는 점도 특징이다.

반면 DB2는 Shared-Nothing 접근 방식을 사용한다. 이 아키텍처에서는 데이터베이스 파일이 파티셔닝을 통해 클러스터를 구성하는 각 노드의 인스턴스에 분산된 형태로 존재한다. 각 인스턴스 또는 노드는 일정 범위의 데이터만을 보유하며 해당 데이터를 배타적으로 점유하고 있다. 즉 Shared-Nothing 시스템은 파티셔닝을 통해 워크로드를 다수의 노드에 분산하는 효과를 제공하며 이것은 노드의 데이터 소유권이 자주 변경되지 않는 경우에 효과적이다(단 데이터베이스 재편성, 노드 장애시 데이터 소유권이 변경될 수 있다).

표면적으로는 Shared-Nothing 시스템이 분산형 데이터베이스와 유사하게 보인다. 그러나 Shared-Nothing 데이터베이스는 하나의 데이터 딕셔너리를 가진 하나의 물리적 데이터베이스라는 점에서 분산형 데이터베이스와는 근본적인 차이가 있다.

이미 살펴본 것처럼 오라클 데이터베이스 10g RAC은 패키지 애플리케이션을 별도의 수정 과정없이 단일 시스템에서 클러스터 구성으로 마이그레이션할 수 있다. 반면 DB2 데이터베이스를 DB2 UDB EEE로 마이그레이션하려면 데이터 파티셔닝 작업과 추가적인 개발 작업이 불가피하다. 〈표 9〉는 두 제품의 아키텍처가 갖는

〈표 8〉 파티셔닝 옵션 비교

기능	오라클	DB2
레인지 파티셔닝	지원	-
리스트 파티셔닝	지원	-
해시 파티셔닝	지원	지원
컴포짓 파티셔닝	지원	-
로컬 인덱스	지원	지원
글로벌 파티션드 인덱스	지원	-
그로벌 넌 파티션드 인덱스	지원	-

특집 2부

[오라클 데이터베이스 10g 최신 성능 관리 팁]

성능 향상의 열쇠는 테이블 스캔 방법

10g Segment Shrink 기능을 이용한 HWM 낮추기

오라클 데이터베이스에서 모든 세그먼트는 세그먼트 내에 데이터를 포함하거나 데이터를 쓴 적이 있는 상위 경계선인 HWM(High Water Mark)을 가진다. HWM은 일반적으로 Full Table Scan(전체 테이블 검색) 시에 HWM까지 읽음으로써 빈번한 삭제가 일어나서 실 데이터에 대한 영역이 소수 존재하는 테이블에 대한 테이블 스캔 시 필요 이상의 블럭을 스캔하게 된다. 기존 버전에서는 이 HWM을 낮추기 위해 TRUNCATE나 DROP 후 CREATE와 같은 테이블을 재구성해야 했지만 10g 버전부터 SEGMENT SHRINK 기능을 이용해 테이블을 재구성하지 않고 운영 중에 동적으로 HWM을 낮추는 것이 가능해졌다. 적용되는 Object는 Normal Table, Index, lob, IOT, Mview 등이다.

먼저 다음과 같이 테이블을 생성해보자. 해당 테이블이 생성되는 테이블 스페이스는 AUTO Segment Space Managed 테이블 스페이스여야 한다.

```
CREATE TABLE SHRINK_TEST
AS
SELECT * FROM ALL_OBJECTS;
```

이제 Analyzing 후에 해당 블럭의 상태를 확인한다.

```
ANALYZE TABLE SHRINK_TEST COMPUTE STATISTICS;

SELECT blocks, empty_blocks, num_rows
FROM   user_tables
WHERE table_name = 'SHRINK_TEST';

BLOCKS      EMPTY_BLOCKS            NUM_ROWS
------------------------------------------
563         77                      38956
```

여기서 블럭의 개수는 세그먼트에 의해 사용된 적이 있는 블럭의 수, 즉 HWM의 수치를 나타낸다. 또한 EMPTY_BLOCKS 수는 HWM 위의 블럭을 의미한다. Delete시킨 다음 Analyzing 후 다음과 같이 해당 자료사전을 조회해 보자.

```
DELETE FROM SHRINK_TEST WHERE ROWNUM < 30000;
COMMIT;
ANALYZE TABLE SHRINK_TEST COMPUTE STATISTICS;

SELECT blocks, empty_blocks, num_rows
FROM   user_tables
WHERE table_name = 'SHRINK_TEST';

BLOCKS      EMPTY_BLOCKS            NUM_ROWS
------------------------------------------
563         77                      8957
```

BLOCKS과 EMPTY_BLOCKS의 수치가 그대로인 것을 확인할 수 있다. DELETE를 함으로써 HWM의 위치를 낮추지는 않는다. 그러므로 HWM 이후에 존재하는 EMPTY_BLOCKS의 개수 또한 변함이 없다. 여기에 Segment Shrink 기능을 적용해보자. 먼저 다음과 같이 테이블의 row movement 기능을 활성화시킨다.

```
SQL>ALTER TABLE SHRINK_TEST ENABLE ROW MOVEMENT;
```

이제 테이블과 HWM을 shrink시킨다.

```
SQL> ALTER TABLE SHRINK_TEST SHRINK SPACE;
```

이처럼 SQL>ALTER TABLE 테이블명 SHRINK SPACE 〈[OPTION], 〈COMPACT, CASCADE〉; 형태를 띠는데 여기서 각 옵션은 다음과 같다.

1 **COMPACT** : 테이블을 SHRINK시키고 HWM는 그대로 둔다

2 **CASCADE** : 테이블 및 관련된 인덱스를 모두 shrink시킨다. MView 형태의 테이블을 shrink시키려면 'SQL〉 ALTER TABLE 〈table name〉 SHRINK SPACE;' 와 같이 사용하고 인덱스만 shrink시킬 때는 'SQL〉 ALTER INDEX 〈index name〉 SHRINK SPACE;' 처럼 사용한다.

이제 SHRINK SPACE 명령으로 해당 테이블을 재구성한 뒤 Analyzing 후 결과를 확인해보자.

```
ANALYZE TABLE SHRINK_TEST COMPUTE STATISTICS;

SELECT blocks, empty_blocks, num_rows
FROM   user_tables
WHERE table_name = 'SHRINK_TEST';

BLOCKS      EMPTY_BLOCKS            NUM_ROWS
------------------------------------
112         16                      8957
```

이제 해당 블럭의 수가 112로 줄어든 것을 확인할 수 있다.

10g의 BFT의 생성과 관리

10g에서는 새로운 테이블 스페이스 타입인 BFT(Big File Tablespace, 대용량 파일 테이블 스페이스)를 새롭게 선보였다. BFT의 크기는 블럭 크기에 따라 8TB부터 128TB까지 지원한다. BFT와 구별하기 위해 10g 이전 버전에서 존재했던 테이블 스페이스를 '스몰 테이블 스페이스(Small Tablespace)'라고 부른다. 이 BFT는 반드시 Locally-Managed Tablespace만 지원하고 오직 하나의 테이터 파일로만 구성이 된다. 따라서 BFT는 테이블 스페이스와 테이터 파일이 1:1로 대

응됨으로써 파일 관리가 용이해졌고 기존의 저장 공간을 극대화할 수 있는 장점이 있다. BFT를 생성하려면 다음과 같이 BIGFILE 키워드를 이용하면 된다.

```
SQL> CREATE BIGFILE TABLESPACE BIG_TABLESPACE DATAFILE
'C:\oracle\product\ 10.1.0\oradata\orcl\bin.dbf' SIZE 10M;
```

일반 테이블 스페이스와 테이블 스페이스와 마찬가지로 BTF 파일의 변경은 ALTER 명령어로써 RESIZING할 수 있다(ALTER TABLESPACE BIG_TABLESPACE RESIZE 100M;). 또한 BFT 관련된 정보로는 DATABASE_PROPERTIES, V$TABLESPACE, DBA_TABLESPACES에 BIGFILE에 대한 정보가 추가됐다.

```
SQL>SELECT TABLESPACE_NAME,BIGFILE FROM DBA_TABLESPACE;

TABLESPACE_NAME          BIGFILE
------------------------------
SYSTEM                   NO
UNDOTBS1     NO
SYSAUX                   NO
TEMP                     NO
BIG_TABLESPACE           YES
```

또한 BFT는 기존의 RowID 체계와 달리 상대파일 번호(Realtive File Number)를 제외하고 이 자리에 BLOCK NUMBER를 기록하는 자리로 사용하고 있다. BFT에서는 상대파일 번호가 항상 1024로 고정되어 있기 때문에 이에 대한 정보가 필요치 않다. 따라서 기존의 RowID 정보를 이용한 애플리케이션이나 PL/SQL에서 BFT을 사용하려면 이에 대한 영향을 미리 검토해야 한다.

```
RowID 체계
- Smallfile Tablespace
  000000   FFF   BBBBBB   RRR
- Bigfile  Tablespace
  000000   LLL   LLLLLL   RRR
- 000000 : data object number
  FFF : relative file number(상대 파일 번호)
  BBBBBB : data block number
  RRR : row number
  LLL LLLLLL : encoded block number (BFT에서의 BLOCK NUMBER를 처리하기
               위한 자리)
```

성능과 확장성 면에서의 차이를 비교한 것이다.

오라클 데이터베이스 10g RAC은 트랜잭션을 실행 중인 노드에 로그를 기록하는 작업이 완료되는 즉시 커밋을 수행할 수 있다. 트랜잭션이 클러스터의 다른 노드에 의해 수정된 데이터를 접근해야 하는 경우에도 추가적인 디스크 I/O를 수반하지 않고 고속 연결을 통해 블럭을 전송한다. 로그의 쓰기 작업이 완료되지 않은 상태에서도 블럭을 전송할 수 있어 SAP SD 벤치마크처럼 집중적인 INSERT 작업이 수반되는 벤치마크 환경에서도 로그 쓰기 작업으로 인해 전송이 지연되는 경우가 5% 이하인 것으로 나타났다.

반면 DB2 시스템은 하나의 트랜잭션을 통해 두 개 이상의 파티션의 데이터가 변경된 경우 트랜잭션의 정합성을 보장하기 위해 two-phase 커밋 프로토콜(커밋 시점의 두 단계 커밋의 첫 번째 시점에 준비 레코드를 기록해야하며, 첫번째 단계를 완료해야 두 번째 단계를 진행하는 것)이 반드시 수행돼야 한다. DB2 트랜잭션은 커밋 시점에 쓰기 작업을 수행할 레코드를 미리 준비한 후 two-phase commit의 첫번째 단계를 완료한 이후에 두 번째 단계를 수행하며 이는 OLTP 애플리케이션의 응답시간을 저하시키는 결과를 초래할 수 있다.

RAC은 GCS(global cache service, 데이터가 필요하고 캐시에 여유 공간이 있는 RAC이 수정된 데이터를 독립적으로 캐시할 수 있게 해주는 서비스. 이 데이터에 대한 추가 액세스는 메인 메모리 속도로 수행할 수 있다)를 사용해 캐시 일관성을 보장한다. GCS는 RAC가 간헐적으로 변경되는 데이터를 여러 노드의 캐시에 동시에 저장하고 캐시를 위한 공간을 확보하기 때문에 이후 데이터에 대한 접근이 발생하는 경우 메인 메모리의 전송 속도에 준하는 응답시간을 나타낸다.

반면 DB2는 마지막 액세스가 발생한 이후 데이터가 변경되지 않은 경우에도 노드 간의 통신을 통해 다른 파티션의 데이터에 대한 접근을 처리한다. DB2는 인덱스와 테이블을 동일하게 파티셔닝하기 때문에 쿼리를 수행하는 과정에서 다수의 파티션에 대한 검색 작업이 불가피하다. 예를 들어 직원 테이블이 직원 번호를 기준으로 파티셔닝돼 있고 직원 이름을 기준으로 한 인덱스가 생성되어 있다면 직원 이름을 조회하는 쿼리를 수행하려면 모든 파티션을 동시에 검색해야 한다. 직원의 이름을 기준으로 한 조회 작업은 파티션의 수가 많으면 많을수록 높은 부하를 수반하게 된다.

또한 DB2 시스템은 특정 노드에 대한 부하 집중의 위험도가 높아 데이터가 전체 파티션에 균등하게 분산되어 있지 않을 수 있다. 예를 들어 금융계의 최근 거래내역의 빈번한 조회라든지 특정 데이터 영역대의 과도한 조회 업무에 따라 특정 파티션의 데이터가 집중적으로 조회될 가능성이 있다.

반면 RAC 환경에서는 개별 노드가 데이터를 점유하지 않으며

모든 노드가 동일한 데이터에 접근하므로 부하 분산의 불균형이 발생하지 않는다. 트랜잭션을 클러스터의 특정 노드군으로 라우팅함으로써 RAC의 성능을 더 높일 수 있으며 이를 통해 데이터 친화도(data affinity, 다량의 서로 다른 데이터에서 서로의 유사한 패턴)를 높이고 노드 간의 통신을 줄일 수 있다. 라우팅은 오라클 넷의 서비스 네임을 통해 간단하게 설정할 수 있다. 반면 DB2의 경우 트랜잭션에 의해 접근되는 데이터의 위치 정보가 별도로 필요하므로 트랜잭션의 라우팅이 훨씬 까다롭다. 또 데이터의 재분배 작업을 수행하지 않은 상태에서 다수의 논리적 노드에 트랜잭션을 수행해야 하므로 성능 저하 현상이 발생할 수 있고 부하의 변화에 유연하게 대처하지 못할 가능성이 높다.

RAC은 애플리케이션의 바인드 값(bind value)를 기반으로 미들웨어가 요청을 라우팅하도록 구성되기도 한다. 예를 들어 사용자의 로그인 정보를 기반으로 메일 서버가 이메일 연결을 라우팅하도록 설정하는 식이다. 최적의 성능을 위해서는 레인지나 리스트 파티셔닝을 이용해 바인드 값을 기준으로 한 파티셔닝을 수행하는 것이다. 반면 DB2는 데이터의 위치를 사용자가 직접 결정할 수 없으므로 이와 같은 방식을 구현하기 힘들다.

<화면 1> ADDM을 통한 자가 튜닝 보고서

<화면 2> 튜닝 어드바이스

<표 9> 성능과 확장성 측면에서 오라클과 DB2 비교

오라클 데이터베이스 10g RAC	DB2 EEE
two-phase 커밋 불필요	two-phase 커밋 필요
데이터는 다수 노드의 캐시에 저장됨	다른 파티션에 접근하려는 경우 IPC 필요
데이터를 단 한 차례만 조회	다수의 파티션에 대해 데이터 조회
균등한 부하 분배	부하가 특정 노드에 집중될 가능성 높음

<표 10> 성능 관리 관련 기능 비교

	오라클 데이터베이스 10g	DB2
성능 관련 관리 기능	- Automatic Workload Repository	유사한 기능이
	- Automatic Database Diagnostic Monitor	존재하지
	- Automatic SQL Tuning	않음

셀프 튜닝과 성능 관련 기능

마지막으로 오라클과 DB2는 진단 및 셀프-튜닝 기능 측면에서도 차이가 있다. 오라클 데이터베이스 10g는 성능 모니터링 작업을 단순화하고 성능 문제의 진단과 해결을 자동화하기 위한 다양한 툴을 기본으로 지원해 이를 통해 시스템 리소스의 사용 상황에 따라 데이터 매개변수를 자동으로 조정한다. 관리자가 만일 어떠한 원인으로 일어날 수 있는지에 대한 시나리오를 시뮬레이션할 수 있는 인텔리전트 어드바이스 기능도 제공하는데 index advisory, summary advisory, memory advisory, MTTR advisory, table/index usage advisory 등이 대표적이다.

DB2 역시 일부 셀프-튜닝 기능과 어드바이스 기능을 제공하고 있지만 여전히 관리자에게 상당한 수준의 데이터베이스 지식을 요구한다. 예를 들어 DB2의 Control Center는 실시간 모니터링에 필요한 다양한 성능지표를 제공하지만 시스템의 전반적인 상태를 확인하기 위해 어떤 성능지표를 참고해야 하는지에 대한 정보는 알려주지 않는다. 알 수 없는 이유로 시스템의 성능이 저하된 경우 DB2 관리자는 전적으로 자신의 개인적인 지식에 의존해서 문제 해결 작업을 수행해야 하는 것이다. 반면 오라클은 어드바이스 기능을 이용하여 관리자에 대한 가이드를 제공하고, 도움말과 드릴다운을 통해 문제의 근본원인을 분석할 수 있도록 지원한다.

〈표 10〉은 오라클이 데이터베이스 튜닝 관련 정보를 제공하고 튜닝 프로세스 자동화를 위해 제공하는 기능을 요약한 것이다. AWR(Automatic Workload Repository)은 데이터베이스 작업에 관련한 성능 데이터와 통계를 저장하기 위해 활용되는 공간이다. 오라클 데이터베이스는 중요한 통계 정보와 워크로드 정보의 스냅 샷을 일정 주기로 생성하고 이를 AWR에 저장한다. 수집/처리된 통계정보는 오라클 데이터베이스 10g에 의해 사전 예방적/사후 대응적 모니터링을 위한 진단 데이터로 활용된다. 그러나 DB2는 이와 유사한 기능을 제공하지 않는다.

ADDM(Automatic Database Diagnostic Monitor)은 시스템 상태를 확인하기 위해 AWR에 캡처된 데이터를 분석하는 데이터베이스 자가진단 엔진이다. ADDM은 시스템의 어느 부분이 가장 많은 'DB time'을 사용하는지 분석하고 해결 방안을 제안하거나 SQL Access Advisor와 같은 다른 솔루션을 제안함으로써 DB time을 최소화하는 것을 기본 목적으로 하고 있다. ADDM은 표면

적인 현상에 초점을 맞추는 대신 드릴다운을 통해 문제의 근본 원인을 확인하고 문제로 인한 시스템의 전반적인 영향에 대해 리포트를 제공한다. 또한 제시된 해결방안이 제공하는 기대효과를 정량화하고 성능에 문제가 없는 또는 튜닝이 불필요한 시스템 영역에 대한 보고서를 〈화면 1〉처럼 제공한다.

오라클 데이터베이스 10g는 SQL 구문의 튜닝 과정을 상당 부분 자동화했다. Automatic SQL Tuning은 Automatic Tuning Optimizer를 기반으로 구현된 기능으로, Oracle Query Optimizer는 자동 튜닝 모드에서 튜닝 프로세스에 필요한 조사와 검증 작업에 더 많은 시간을 할애한다. 이와 같은 추가적인 시간을 통해 다이내믹 샘플링, 부분 실행(partial execution) 등 일반 운영 모드에서는 시간적인 제약으로 인해 적용될 수 없었던 테크닉이 사용되며 비용, 선택성(selectivity)과 확률에 대한 검증 작업을 수행하는 것이 가능하다.

Automatic Tuning Optimizer에 의해 얻어진 결론은 SQL Tuning Advisor를 통해 튜닝 어드바이스의 형태로 사용자에게 전달된다. 어드바이스는 하나 또는 그 이상의 권고사항으로 구성되며 각 권고사항 별로 근거와 예상 효과가 명시된다. 어드바이스에는 새로운 인덱스의 추가, SQL 구문의 재작성, 또는 SQL 프로파일의 구현과 같은 내용이 포함될 수 있으며 사용자는 어드바이스의 이행 여부를 단순히 선택해 SQL 구문의 튜닝 과정을 완료할 수 있다. 반면 DB2는 SQL 관련 문제를 진단하기 위한 쉽고 편리한 방법을 제공하지 않으며 (오직 트레이스 기능만 제공) SQL 구문의 재작성을 통해 튜닝을 수행하는 툴 또한 제공하지 않는다.

오라클은 어렵다?

오라클 데이터베이스는 다양한 업계 표준/ISV 벤치마크를 통해 그 성능을 인정받고 있으며 이것은 가장 최근에 출시된 오라클 데이터베이스 10g 역시 예외는 아니다. 필자 역시 기존 버전 제품에서의 변화보다 더 많은 변화를 몸소 느끼고 있다.

이번 글에서는 다른 DBMS와 차이점을 중심으로 살펴보았지만 오라클 역시 타 DBMS에서 좋은 점들을 벤치마킹하기 위해 노력하고 있고 실례로 기존 버전에서 쉽게 손댈 수 없는 SQL 튜닝, 메모리 튜닝 등 자동화된 관리 기능은 초보자들도 쉽게 다룰 수 있도록 배려한 것으로 볼 수 있다. 즉 '오라클은 어렵다'는 등식도 점점 깨져가고 있는 것이다. 현재 국내에서 가장 널리 사용되고 있는 RDBMS인 오라클의 진화를 필자는 흥미진진한 마음으로 지켜보고 있다.

정리 | 박상훈 | nanugi@imaso.co.kr