Tech Notes(vol.10)

성능 향상을 위한 Informix Dynamic Server 모니터링

소개

Informix Dynamic Server(IDS)와 그 아키텍처 및 성능 조정에 대해 자세히 설명한 서적과 자료는 많지만, 그 중에서 성능 모니터링에 대해 자세하게 다루고 있는 것은 별로 없는 것 같습니다. 그러나 효과적인 모니터링은 IDS 관리에서 매우 중요합니다. 이것을 통해 귀중한 통계를 모으고 데이터베이스 문제점을 조기에 확인함으로써, 성능 조정을 위한 수단을 미리 강구할 수 있습니다. 일단 Informix 데이터베이스가 정상적으로 구축되면, 데이터베이스 관리자가 최우선으로 해야 할 일은 성능을 모니터하는 것입니다.

이 글에서는 Informix Dynamic Server 성능을 모든 레벨에서 효과적으로 모니터하는 방법에 대해 자세히 설명하고, Informix 데이터베이스 시스템의 문제 해결 및 성능 조정에 대한 일반적인 팁을 제공합니다. 특히 다음과 같은 내용을 다룹니다.

- 모니터링 툴(Monitoring Tool)
- 인스턴스 동작(Instance Activity) 모니터링
- 데이터베이스 동작(Database Activity) 모니터링
- 세션 동작(Session Activity) 모니터링

모니터링 툴(Monitoring Tool)

Checkpoint 간격은 다음과 같은 매개변수로 결정됩니다.

Informix에서는 시스템 성능 모니터링을 위해 두 가지 주요 툴을 제공하는데, **onstat** 유틸리티와 다수의 시스템 모니터링 인터페이스(SMI;System Monitoring Interface)) 테이블이 그것입니다. **onstat** 유틸리티와 SMI 테이블 모두 IDS의 공유 메모리 동작을 조사함으로써 IDS 성능을 모니터하지만, 이 두 가지 사이에는 몇몇 중요한 차이점이 있습니다. **onstat** 유틸리티는 실제로 IDS 공유 메모리를 읽고 그 통계를 출력합니다. 이것은 사용하기는 쉽지만, 수집하는 통계가 IDS를 시작할 때부터 누적되는 경우가 종종 있으므로 실제로 필요한 것보다 더 많은 내용을 포함하는 고정 형식으로 나타날 수 있습니다. 반면 SMI 테이블은 IDS를 초기화할 때 자동적으로 생성되는 sysmaster 데이터베이스에 저장됩니다. SMI 테이블은 실제 테이블이 아니라, 동적

뷰이거나 IDS 공유 메모리 구조에 대한 포인터 역할을 하는 "의사 테이블(pseudo-tables)"입니다. 따라서 SMI 테이블에 있는 데이터는 IDS의 동작을 더 정확하게 반영합니다. 또한 SMI 테이블은 일반적인 SQL을 사용하여 질의할 수 있으며, 좀 더 의미 있고 알아보기 쉬운 형식으로 데이터를 나타낼 수 있습니다.

IDS 동작은 인스턴스 동작(instance activity), 데이터베이스 동작(database activity), 세션 동작(session activity)이라는 세 가지 범주로 분류할 수 있으며, Informix가 제공하는 onstat이나 SMI 테이블을 사용하면 각각의 동작을 효과적으로 모니터할 수 있습니다.

인스턴스 동작(Instance Activity) 모니터링

IDS 인스턴스란 Informix 공유 메모리, Informix 데이터베이스 및 Informix에 할당된 디스크 등의 물리적 장치를 가리킵니다. 모니터해야 할 가장 중요한 몇몇 인스턴스 동작을 소개하면 다음과 같습니다.

작동 모드(Operating Mode)

무엇보다도 가장 중요한 인스턴스 동작은 물론 IDS 작동 모드입니다. IDS가 제대로 실행되는지, 무슨 문제점은 없는지, 다운 상태는 아닌지 등을 확인하는 것입니다. **onstat -p** 명령을 사용하면 다음과 같이 출력됩니다.

```
Informix OnLine Version 7.24.UC5 -- On-Line -- Up 1 days 14:59:11 --302528 Kbytes
```

26289	0	60959453	0	0	0	5960	27077	
ixda-RA 54105	idx-RA 392	da-RA 50824	RA- pgsused 79198	lchwaits 1424				

첫번째 줄은 현재 IDS 작동 모드를 나타냅니다. 여기에서는 데이터베이스가 "온라인(On-Line)" 상태입니다. 작동 모드에는 여섯 가지가 있습니다. 이 여섯 가지 중에서 세 가지, 즉 오프라인(Off-Line), 대기(Quiescent), 온라인(On-Line)이 특히 중요합니다. 오프라인 모드는 IDS가 실행되지 않고 있다는 것을 나타냅니다. 대기 모드는 DBA만이 유지 관리 작업을 수행할 수 있는 단일 사용자 모드에서 IDS가 실행되고 있다는 것을 나타냅니다. 온라인 모드는 IDS가 정상적으로 실행되고 있고 모든 사용자들이 데이터베이스 서버에 연결하여 모든 종류의 작업을 할 수 있다는 것을 나타냅니다. 대부분의 경우 IDS는 온라인 모드여야 합니다. 어떤 이유에서건 IDS가 종료되었거나 오프라인 모드에 있게 되면, 이 명령을 실행할 때 다음 메시지가 나타납니다.

```
shared memory not initialized for InformixSERVER 'cassprod_shm'
```

이런 경우에는 문제의 근본 원인을 알기 위해 온라인 메시지 로그를 점검해야 합니다(아래의 "메시지 로그" 부분을 참조하십시오).

위와 같은 출력 내용은 현재 작동 모드 이외에도 시스템 성능에 대한 중요한 통계를 제공합니다. 두 개의 %cached 필드는 IDS에서 캐시를 얼마나 효과적으로 사용하고 있는가를 나타냅니다. 첫 번째 %cached 필드는 읽기 캐시 비율(read cache rate)의 백분율이고, 두 번째 필드는 쓰기 캐시 비율(write cache rate)의 백분율입니다. 읽기 캐시 비율과 쓰기 캐시 비율은 응용 프로그램과 이것에서 작동 중인 데이터의 형식 및 종류에 따라 아주 다양합니다. 그러나 일반적으로 읽기 캐시 비율과 쓰기 캐시 비율은 80-90퍼센트 범위 내에 있어야 합니다. 읽기 캐시 비율과 쓰기 캐시 비율 모두 80퍼센트보다 낮은 경우, 이 비율을 높이기 위해 Informix 구성 파일에서 BUFFERS 매개변수 값을 올리는 것을 고려할 수 있습니다. 이 비율이 낮다는 것은

IDS가 필요 이상으로 디스크 읽기와 쓰기를 많이 하고 있다는 뜻이며, 이렇게 함으 로써 성능이 대폭 저하됩니다.

출력에서 seqscans 필드는 데이터베이스가 생성된 이후에 수행된 순차적 스캔의 횟수를 표시합니다. 이 숫자가 크고(예를 들면 100,000 이상) 계속 증가한다면 일부 성능 문제(특히, OLTP 환경에서)가 있을 수 있으므로 좀 더 추가적인 조사가 필요합니다. 나중에 이 문제에 대해 더 자세히 설명할 것입니다.

ovlock 필드는 IDS가 동시 사용 가능한 최대의 로크(lock) 수를 초과하려고 시도한 횟수를 가리킵니다. 이 숫자가 0이 아니면, ONCONFIG 파일에서 LOCKS 매개변수 값을 높여야 할 것입니다. ovbuf 필드는 IDS가 동시에 사용 가능한 최대 버퍼의 수를 초과하려고 시도한 횟수를 가리킵니다. 로크와 마찬가지로 이 숫자가 0이 아니면, ONCONFIG 파일에서 BUFFERS 매개변수 값을 높여야 할 것입니다. bufwaits 필드는 사용자 스레드가 버퍼를 기다려야 하는 횟수를 가리킵니다. 이 숫자가 크면(예를 들면 100,000 이상) BUFFERS 매개변수 값을 높여서 사용자가 디스크의 데이터에 액세스할 때 버퍼를 기다릴 필요가 없도록 해야 합니다. 이는 응답 시간(response time)을 줄이므로 성능이 향상되는 결과를 가져옵니다.

다른 중요한 필드 그룹에는 ixda-RA, idx-RA, da-RA, RA-pgsused가 포함됩니다. 이 필드들은 IDS에서 미리 읽기 매커니즘(read-ahead mechanism)을 얼마나 효과적으로 사용하고 있는가를 나타냅니다. 미리 읽기는 순차적 스캔 또는 색인 읽기를 하는 동안 디스크에서 메모리로 일정한 수의 데이터 페이지를 미리 읽는 작업입니다. 가장 바람직한 것은 미리 읽기 페이지 수(ixda-RA, idx-RA, da-RA의 합)가 순차적 스캔 또는 색인 읽기를 하는 동안 사용된 페이지 수(RA-pgsused)와 같은 것입니다. 이것은 미리 읽기 페이지가 순차적 스캔 또는 색인 읽기에 100퍼센트 사용되고 있음을 나타냅니다. 이 두 숫자 사이에 큰 차이가 있다면 (>10,000), 이 차이가 양수건 음수건 관계 없이 IDS에서 미리 읽기를 효율적으로 사용하고 있지 않다는 것이므로, 더 나은 성능을 위해 미리 읽기 매개변수(예: RA_PAGES와 RA_THRESHOLD)를 조정할 수 있습니다. 이러한 매개변수를 조정하는 방법에 대해서는 Administrator's Guide 를 참고하십시오.

메시지 로그

메시지 로그는 온라인 로그라고도 합니다. 여기에는 검사점(checkpoints)의 시간과 지속 시간, 시작(startup)과 종료(shutdown), 백업과 복구 상태, 논리적 로그 백업 상태, 주요 구성 매개변수에 대한 변경 등과 같은 매우 중요한 인스턴스 동작에 대한 다양한 정보가 포함되어 있습니다. 또한 디스크 I/O 오류, 다운 청크, 데이터 무결성 오류 등의 중요한 오류들도 포함됩니다. Informix에서는 이러한 것들을 "가정 오류(assertion failure)"라고 합니다. 가정 오류가 발생하면, 메시지 로그는 보통 상대적 가정 오류("af.xxx") 파일을 알려주는데, 이 파일에는 데이터베이스가 종료된 시간에 대한 자세한 정보가 기록되어 있으며, 더러 문제 해결 방법에 대한 제안이 제공되기도 합니다. 다음은 메시지 로그에서 발췌한 것입니다.

```
15:09:37 Assert Failed: Unexpected virtual processor termination, pid = 587, exit = 0x9
15:09:37 Who: Session(7, Informix@, 0, 0) Thread(12, sqlexec, 0, 1)
```

Informix OnLine Version 7.24.UC5 -- On-Line -- Up 1 days 15:10:40 --302528 Kbytes

Dbspaces

address	number	flags	fchunk	nchunks	flags	owner	name
1a036100	1	1	1	1	N	Informix	rootdbs
1a05b6f8	2	1	2	3	N	Informix	llogs
1a05b768	3	2001	3	2	NT	Informix	temp_dbspace1

3 active, 2047 maximum

Chunks

address	chk/dbs	offset	size	free	bpages	flags	pathname
1a036170	1	1	0	50000	39923	PO-	/usr/Informix/dblink
1a0365c8	2	2	0	50000	1447	PO-	/usr/Informix/dblogs
1a0366a0	3	3	50000	100000	99947	PO-	/usr/Informix/dblink

3 active, 2047 maximum

출력이 두 부분으로 되어 있는 것을 알 수 있습니다. 첫 번째 부분에는 모든 DB영역이 나열되어 있고, 두 번째 부분에는 모든 청크가 나열되어 있습니다. Chunks 부분에서 flags 필드를 특히 주목할 필요가 있습니다. 이 필드의 첫 번째 문자는 청크가 기본("P") 청크인지 미러("M") 청크인지를 나타냅니다. flags 필드의 두 번째 문자는 청크의 현재 상태가 온라인("O")인지 오프라인("D")인지를 나타냅니다. "O"와 "D"는 매우 비슷하게 보일 수 있기 때문에(특히 사용자가 급하게 서두를 때), 어떤 다운 청크도 놓치지 않으려면 결과를 "grep PD" **(onstat -d |grep PD)**로 파이프할 수 있습니다. 다운 기본 청크(down primary chunk)가 있는 경우에는 데이터 무결성을 위해 즉시 백업 테이프로부터 cold restore 또는 warm restore를 실행하여 복구해야 합니다.

검사점 지속 시간(checkpoint duration time)

검사점(checkpoint)은 디스크의 페이지를 공유 메모리 버퍼 풀의 페이지와 동기화하는 프로세스입니다. 검사점이 진행되는 동안 IDS는 사용자 스레드가 중요한 세션으로 들어가는 것을 방지하고 모든 트랜잭션 동작을 차단합니다. 따라서 검사점 지속 시간이 길면 사용자의 시스템이 정지해 버릴 수 있습니다. 이것은 수천 개의 트랜잭션이 있고 응답 시간이 가장 중요한 OLTP 환경에서 특히 그렇습니다. 위에서 말한 것처럼, 메시지 로그를 보고 검사점 지속 시간을 모니터할 수 있지만, onstat -m 명령을 사용하는 것이 더 빠르고 효율적입니다.

다음은 가장 최근에 진행된 검사점의 지속 시간을 나타냅니다.

```
03:03:35 Checkpoint Completed: duration was 0 seconds.
03:13:35 Checkpoint Completed: duration was 0 seconds.
03:33:35 Checkpoint Completed: duration was 0 seconds.
03:53:35 Checkpoint Completed: duration was 0 seconds.
04:03:35 Checkpoint Completed: duration was 0 seconds.
04:13:35 Checkpoint Completed: duration was 0 seconds.
04:33:35 Checkpoint Completed: duration was 0 seconds.
04:53:35 Checkpoint Completed: duration was 0 seconds.
05:03:35 Checkpoint Completed: duration was 0 seconds.
```

검사점 지속 시간이 10초보다 길면 LUR_MIN_DIRTY 구성 매개변수와 LUR_MAX_DIRTY 구성 매개변수의 값을 줄여서 검사점 지속 시간을 더 짧게 할 수 있습니다. 또한 onstat -F 명령을 사용했을 때 청크 읽기가 매우 높고(>10,000) 이 숫자가 계속 증가한다면, 이것은 다음 두 가지 문제 중 하나를 가리킵니다. 우선 한 가지는 검사점 간격이 너무 짧고, 클리너(cleaners)가 검사점 사이에서 수정된 버퍼를 디스크에 기록할 시간이 충분하지 않다는 것이고, 다른 하나는 AIO VP가 너무 적어서 검사점 진행 도중에 과도한 디스크 쓰기를

공유할 수 없다는 것입니다. 이럴 경우에는 CKPINTVL, LRU, CLEANERS, NUMAIOVPS 구성 매개변수를 다시 조사하고 값들을 적절히 높여야 합니다. 이러한 매개변수 조정에 대한 자세한 내용은 Informix OnLine Dynamic Server Administration Guide를 참고하십시오.

DB영역 사용

각 DB영역에 충분한 공간을 확보하는 것은 매우 중요합니다. 만약 스크립트를 실행하여 각 DB영역이 할당된 크기, 사용된 양, 사용 가능한 영역의 백분율 등을 확인할 수 있습니다.

prod_idx2_dbs	75000	60101	20
dev_dbs	75000	60541	19
dev2_dbs	75000	60541	19

이 실행 결과를 보면 공간이 부족한 DB영역을 확인할 수 있습니다. 미리 대비를 하려면, DB영역의 디스크 사용율이 90퍼센트에 이를 때 디스크를 추가로 할당하는 것을 고려하십시오.

DB영역 I/O

DB영역 I/O는 디스크 읽기와 쓰기에 의해 측정됩니다. 일부 DB영역에서는 디스크 읽기와 쓰기가 과도하게 이루어지고 다른 DB영역에서는 거의 작업이 없다면, 시스템에서 디스크 I/O 병목 현상이 발생할 수 있습니다. DB영역 I/O의 균형을 잘 맞추면 시스템 디스크 I/O 로드가 수월하므로 전체적인 시스템 성능이 향상됩니다. 다음 스크립트는 각 DB영역에 대한 I/O 통계를 보여 줍니다.

```
select d.name, fname[15,25] path_name, sum(pagesread) diskreads,
sum(pageswritten) diskwrites
from syschkio c, syschunks k, sysdbspaces d
where d.dbsnum = k.dbsnum and k.chknum = c.chunknum
group by 1,2
order by 1;
```

실행 결과는 다음과 같습니다

name	path_name	diskreads	diskwrites

prod_dbs	/usr/Informix/link5	1	0
prod2_dbs	/usr/Informix/link1	237340	344
prod3_dbs	/usr/Informix/link	47415	8611
prod_idx_dbs	/usr/Informix/link4	1	0
dev_dbs	/usr/Informix/link1	3373	90

목표는 디스크 읽기와 쓰기가 모든 DB영역에서 균등하게 수행되도록 하는 것입니다. 그러나 완전히 균등하다는 것은 대부분의 경우 비현실적입니다. 위의 출력 내용을 통해 DB영역 I/O의 분배 방법에 대한 아이디어를 얻고, 디스크 읽기와 쓰기의 대부분이 몰리는 DB영역("hot" dbspaces)을 찾아낼 수 있습니다. 일부 DB영역에만 디스크 읽기와 쓰기가 너무 많이 몰리고 다른 DB영역에는 너무 적다면, 물리적 레이아웃 또는 디스크 레이아웃을 조정해야 합니다.

다음과 같이 sysmaster 데이터베이스에 있는 sysptprof 테이블을 질의하면, 대부분의 디스크 읽기와 쓰기가 수행되는 테이블을 알아낼 수 있습니다.

```
select dbsname, tabname, (isreads +  pagreads) diskreads, (iswrites +  pagwrites) diskwrites
from sysptprof
order by 3 desc, 4 desc;
```

이 질의에서 얻은 출력에 기초하여, DB영역 내에서 일부 테이블을 이동함으로써 더 나은 I/O 균형을 이룰 수 있습니다.

공유 메모리 세그먼트

가상 공유 메모리 세그먼트가 너무 많다는 것은(보통 셋 이상) 초기 가상 공유 메모리 세그먼트가 너무 적어 데이터베이스가 추가로 가상 공유 메모리 세그먼트를 계속 할당해야 한다는 것을 뜻합니다. 이것은 IDS 성능에 역효과를 내고 결국 시스템 종료를 초래할 것입니다. **onstat -g seg** 명령을 사용하면 현재 존재하는 공유 메모리 세그먼트를 확인할 수 있습니다.

Informix OnLine Version 7.24.UC5 -- On-Line -- Up 1 days 15:07:59 --302528 Kbytes

Segment Summary:

(resident segments are locked)

id	key	addr	size	ovhd	class	blkused	blkfree
0	1381386241	a000000	267845632	4920	R	32689	7
1	13813862421	a000000	41943040	1232	V	1384	3736
2	138138624231	d000000	614400	604	M	68	7
Total: - - 310304072 - - 34141 3750							

가상 공유 메모리 세그먼트가 네 개 이상 나타나면, 구성 파일에서 SHMVIRTSIZE 매개변수 값을 높여야 합니다. 이것은 IDS가 초기화 시점에 가상 공유 메모리 세그먼트를 충분히 할당하여 사용자가 시스템에 로그인하여 데이터베이스 작업을 수행할 때 가상 공유 메모리 세그먼트를 더 이상 할당하지 않도록 하고자 하는 것입니다. IDS 가상 공유 메모리 세그먼트의 크기를 계산하는 방법에 대한 자세한 내용은 Informix OnLine Dynamic Server Administration Guide를 참고하십시오.

전체적인 운영 체제 성능

IDS 성능을 정확하게 평가하려면 운영 체제 동작을 전반적으로 고려해야 합니다. 특히 데이터베이스가 전용 데이터베이스 서버에 있지 않은 경우에는 더더욱 그렇습니다. IDS가 RAM을 너무 많이 사용하면(예를 들어, 시스템에 512MB RAM이 있는데 IDS가 400MB 이상을 공유 메모리에 사용하는 경우), 사용자가 메모리를 많이 사용하는 작업을 수행할 때 운영 체제에 과도한 스와핑과 운영 중단 사태가 발생할 것입니다. 메모리가 충분하지 않으면 시스템은 메모리에 있는 일부 데이터 페이지를 디스크로 "스왑"하여 새로운 데이터를 위한

데이터베이스 동작(Database Activity) 모니터링

데이터베이스 동작을 모니터하는 목적은 모든 데이터베이스가 항상 최대 성능을 발휘하고 있는지 확인하는 데 있습니다. 이것은 잠재적인 성능 문제를 파악하고 그것의 근본 원인을 찾아내어 처음부터 제거해야 함을 의미합니다. 다음과 같은 사항들을 파악해야 합니다.

테이블 익스텐트

익스텐트(extent)는 물리적으로 연속된 페이지들의 모임입니다. 그러나 테이블에 익스텐트가 둘 이상 있는 경우, 이 익스텐트들이 연속되어 있다는 보장이 없습니다. 익스텐트는 테이블이 있는 전 DB영역에 걸쳐 분산되어 있을지도 모릅니다. 물리적 페이지들의 연속성은 성능에 중요한 영향을 미칩니다. 데이터 페이지들이 연속적이면 디스크에 있는 데이터에 액세스하는 시간이 최소화되고 데이터베이스는 데이터 행들을 순차적으로 읽을 수 있습니다. 테이블에 익스텐트가 너무 많으면, 익스텐트들이 디스크 내에서 여기저기 흩어져 있을 가능성이 높습니다. 이는 성능을 크게 떨어뜨립니다. 왜냐하면 어떤 테이블에서 데이터를 검색할 때, 디스크 헤드는 연속된 물리적 페이지들로 구성된 한 개의 대형 익스텐트 대신 이 테이블에 속하는 연속되지 않은 많은 익스텐트들을 찾아야 하기 때문입니다. 이것은 디스크 검색 속도를 상당히 떨어뜨립니다. 다음 스크립트는 다수의 익스텐트를 가지고 있는 데이터베이스 테이블을 찾아낼 수 있도록 각 테이블의 익스텐트 수를 디스플레이합니다.

```
select t.tabname, count(*) num_ext
from sysmaster:sysextents e, airgen_cm_db:systables t
where e.tabname = t.tabname
and dbsname = "cass_prod"
and t.tabname not like "sys%"
group by 1
having count(*) > 1
order by 2 desc;
```

결과는 다음과 같습니다.

tabname	num_ext

```
order by 2 desc;
```

위 스크립트의 결과는 다음과 같습니다

idxname	levels
cust_idx2	4
ord_idx2	4
price_idx2	4
prod_idx2	3
prod_idx3	3
com_foridx1	3
com_foridx2	3

4레벨 이상을 갖는 색인이 있으면, 성능 향상을 위해 색인을 삭제하고 더 적은 수의 레벨로 재구축하는 것을 고려해야 할 것입니다

색인 고유성

색인의 중복도가 높으면 업데이트와 삭제 성능에 심각한 영향을 미칠 수 있습니다. customer 테이블에서 customer_type 칼럼에 대한 색인을 가지고 있고, 여기에 다섯 개의 가능한 customer_type 코드가 있다고 가정해 봅시다. 테이블에 100만 행이 있다면 동일한 형식의 코드를 가진 행이 20만 개가 있을 수 있습니다. B-트리는 키 값을 저장할 것이고, 이어서 각 물리적 행에 대한 포인터 목록이 저장될 것입니다. 어떤 키 값을 삭제하거나 업데이트해야 할 때 문제가 발생합니다. IDS는 삭제하거나 업데이트할 정확한 키를 찾을 때까지 모든 중복 색인들을 검색해야 하기 때문입니다.

다음 스크립트를 사용하면 중복도가 높은 색인을 확인할 수 있습니다.

```
select tabname, idxname, nrows, nunique
from systables t, sysindexes i
where t.tabid = i.tabid
and t.tabid > 99
and nrows > 0
and nunique > 0;
```

결과는 다음과 같습니다.

tabname	idxname	nrows	nunique
customer	cust_idx1	193219	193219
product	prod_idx1	9519	9519
promotion	prom_idx1	58084	6990
item	items_idx1	6990	6990
splc_name	pname_idx1	58084	6990
tmp_cust	tmpcust1	1984	1984
networks	net_idx1	19301	1404

위의 내용에서 볼 때 가장 이상적인 것은 nunique 칼럼의 값이 nrows 칼럼의 값과 같아서, 색인의 모든 키가 유일 키인 경우입니다. 행 수(위의 nrows 칼럼)와 고유 키 수(위의 nunique 칼럼)에 기초하여, 각 색인이 중복되는 백분율을 다음과 같이 계산할 수 있습니다.

(nunique/nrows)*100

이 백분율이 높을수록, 색인이 더 고유합니다. 중복도가 높은 색인의 성능 병목 현상을 방지하려면, 원래 색인을 중복도가 높은 칼럼과 더 고유한 칼럼을 결합한 복합 색인으로 교체할 수 있습니다. 위의 예를 사용하면, 기본 키 칼럼 customer_id를 원래 색인에 추가하고, 이것을 복합 색인으로 만들 수 있습니다(예: "create index index_name on customer (customer_type, customer_id)").

순차적 스캔(Sequential Scan)

순차적 스캔은 데이터베이스가 명시된 테이블에서 원하는 행을 검색하기 위해 테이블의 모든 행을 한 번 씩 읽기 때문에, 테이블에 순차적으로 액세스하는 것은 때때로 성능에 악영향을 미칩니다. 테이블 크기가 작으면 완전히 메모리 내에 상주할 수 있기 때문에 반복적으로 행을 읽는다고 해도 해가 없습니다(특히 Informix의 라이트 스캔 메커니즘(Informix's light scan mechanism)을 사용하면, 인메모리 테이블의 순차적 스캔이 색인을 통해 동일한 테이블을 검색하는 것보다 훨씬 빠릅니다). 그러나 한 행의 크기에 따라 차이가 있지만 일반적으로 테이블의 행이 10만 개 이상 있는 경우, 반복된 순차적 스캔은 성능에 치명적입니다. 다음 스크립트를 사용하면 순차적 스캔이 많은 테이블을 확인할 수 있습니다.

tabname	tot_scans
customers	302927
orders	4943
prices	784
items	520
products	520
prices	468

위의 결과를 보면 customers 테이블에 순차적 스캔이 대단히 많다는 것을 알 수 있습니다. 이 테이블이 수천 또는 수백만 행을 갖는 큰 테이블이라면, 이 테이블에 색인을 추가하거나 프로그램 지시문(program directives)을 사용함으로써, 옵티마이저가 이 테이블의 해당 데이터를 검색하기 위해 순차적 스캔보다는 색인 검색을 하도록 고려해 볼 수 있습니다.

세션 동작(Session Activity) 모니터링

세션 동작(session activity)에 대한 통계는 잠재적인 성능 문제를 찾아내고 문제를 해결하는 데 매우 유용합니다. 앞에서 설명한 모니터링 툴을 사용하여 어떤 세션 동작에 대한 어떠한 정보를 얻을 수 있는지 알아보도록 하겠습니다.

일반적인 통계

sysmaster 데이터베이스에 있는 **syssessions** 테이블은 로그인명, 로그인 시간, 세션에 로그인한 호스트 컴퓨터, 운영 체제의 프로세스 ID 및 현재 상태 등과 같은 각 세션에 대한 일반적인 정보들을 저장합니다. 이 테이블을 질의하여 이러한 모든 정보를 얻을 수 있습니다.

```
select sid, username, hostname, connected loging_time hex(state)s_state
from syssessions;
```

결과는 다음과 같습니다.

sid	username	hostname	login_time	s_tate
3106	Informix	sys3502	945786951	0x00080021
966	omcadmin	sys3502	945731268	0x00080001
901	omcadmin	sys3502	945730394	0x00080001
91	omcadmin	mmi3502	945731245	0x00080001

s_state는 세션의 현재 동작을 나타내는 16진수입니다(이러한 각 동작에 대한 자세한 내용은 Informix OnLine Dynamic Server Administrator's Guide 의 38-27 페이지를 참고하십시오). login_time은 세션에 로그인한 시간을 나타내는 정수입니다.

sysmaster 데이터베이스에 있는 **syssesprof** 테이블은 각 세션에 대해 더 자세한 내용을 제공합니다. 다음 질의를 사용하면, 각 세션이 데이터베이스와 상호 작용하는 방법에 대해 더 잘 알 수 있습니다.

```
select sid, (isreads+ iswrites+ bufreads+ bufwrites+ pagreads+ pagwrites)
access, locksheld, seqscans, total_sorts, dsksorts
from syssesprof;
```

결과는 다음과 같습니다.

sid	access	locksheld	seqscans	total_sorts	dsksorts
31	57	2	590	0	0
35	3868	1	20	7	0
37	305614	59	56	39168	166028
11	223	1	0	0	88

access 필드는 세션이 데이터베이스에 얼마나 자주 액세스하지를 보여 줍니다. locksheld는 각 세션이 사용하는 로크 개수를 나타냅니다. seqscans는 데이터에 액세스하기 위해 각 세션이 순차적 스캔을 사용하는 빈도를 나타냅니다. 이 빈도가 너무 크면(>1000), 각 세션이 해당 데이터를 검색하기 위해 색인을 사용하는지를 알아 보고, 또한 최적화된 방법으로 데이터 검색을 실행하는지의 여부를 확인하기 위해 질의 실행 계획을 좀 더 자세히 조사할 수 있습니다. total_sorts와 dsksorts는 정렬 작업(sort operation)을 위해 각 세션이 메모리를 얼마나 효율적으로 사용하는가를 보여 줍니다. 다음 공식을 사용하여 각 세션이 정렬을 위해 메모리를 사용하는 백분율을 계산할 수 있습니다.

((total_sorts ?dsksorts) / total_sorts)* 100

이 백분율이 높을수록 정렬 작업이 더 효율적입니다. **syssession** 테이블과 **syssesprof** 테이블을 조인하여 각 세션의 사용자명과 호스트 컴퓨터를 더 잘 식별함으로써, 데이터베이스와 시스템의 잠재적인 문제가 어디에서 오는가를 알 수 있습니다. 다음 질의를 사용하면 이러한 모든 정보를 검색할 수 있습니다.

```
select username, hostname,
(isreads+ iswrites+ bufreads+ bufwrites+ pagreads+ pagwrites) access,
locksheld, seqscans, total_sorts, dsksorts
from syssessions s, syssesprof f where s.sid = f.sid;
```

syssesions 테이블을 **syslocks** 테이블과 조인하면, 사용자들 간의 로크 충돌을 식별하기 위해 어떤 데이터베이스의 어떤 테이블이 어떤 세션에 의해 로크가 걸려 있는지 등의 로크에 대한 더 자세한 정보를 알 수 있습니다.

```
select owner, username, hostname, dbsname, tabname, type
from syssessions s, syslocks 1
where sid = owner
and tabname not like "sys%";
```

위 질의를 실행하면 아래의 형식으로 출력될 것입니다.

owner	username	hostname	dbsname	tabname	type
5180	idsadmin	prod1	cassprod	promotion	S
5176	idsadmin	prod2	cassprod	customer	S
5329	idsadmin	dan	cassdev	commission	S
5223	idsadmin	cass	mydbs	tmp_orders	S
235416	idsadmin	charles	mydbs	tmp_shipping	X

로크 사용에 대한 충돌이 있는 경우(예를 들어, 한 사용자가 다른 사람이 로크를 건 테이블에 배타적으로 액세스하려는 경우) 그 로크의 소유자를 쉽게 식별할 수 있으며, 사용자 우선 순위에 기초하여 **onmode -z session-id** 명령을 실행함으로써 그 세션을 제거하고 로크를 해제할 수 있습니다(여기서 **session-id**는 위 출력에서의 owner 필드입니다). 이 명령은 "Informix" 사용자만이 실행할 수 있습니다.

질의 통계(Query Statistics)

질의 통계는 문제 해결과 질의 최적화에 매우 중요합니다. **onstat -g sql session-id** 명령은 현재 세션에 대한 질의 및 관련 통계를 파악하기 위해 사용합니다(여기서 **session-id**는 직접 수동으로 또는 UNIX 셸 프로그램에 의해 동적으로 플러그할 수 있는 세션 ID입니다). 예를 들어, 세션이 실행 중인 질의를 알아내려면 먼저 **onstat -g ses** 명령을 사용하여 세션 ID를 알아낸 후, 이 ID를 위의 명령(예를 들어, **onstat -g sql 28953**)의 형식으로 실행합니다. 실행 결과는 아래의 형식입니다.

```
Informix OnLine Version 7.24.UC5 -- On-Line -- Up 17 days 07:02:56 --144032 Kbytes
```

```
select unique ne . ne_name , ne . sys_tag
from ne , workspace , acg ,configset
where ne . dte_address = "2345" and ne_type = 0
and ne .workspace_id= acg . workspace_id
and ne . ne_inst = acg . acg_inst
and configset .config_set_version = ne . config_set_version
and configset . ne_dbmode= "R"
and workspace.workspace_id = ne. workspace_id
and workspace .state < 2
```

출력의 첫 번째 부분에서는 질의가 실행되고 있는 데이터베이스, 그 격리 레벨, 로크 모드와 같은, 실행 중인 질의에 대한 일반적인 정보를 보여 줍니다. 가장 흥미 있는 두 필드는 SQL Error와 ISAM Error입니다. 이 두 필드가 0이 아니면 이것은 질의에 어떤 문제가 있고 제대로 실행되고 있지 않다는 것을 나타냅니다. 이 오류 코드에 대한 자세한 내용은 **finderr** 유틸리티를 사용하여 확인할 수 있습니다. 이 문제의 원인을 파악하고 수정한 후 세션을 종료하고 질의를 다시 실행해야 합니다.

"Current SQL statement" 부분은 실행 중인 질의 구문을 표시합니다. 이것은 문제 질의와 질의 최적화를 진단하는 데 매우 유용합니다. 응답 시간이 너무 길거나 시스템 CPU와 메모리를 너무 많이 사용한다는 등의 질의와 관련된 문제점을 발견하면, 차후 조사와 분석을 위해 표시된 것과 동일한 질의 사본을 만들 수 있습니다. 그런 다음 Informix dbaccess 유틸리티에 대해 동일한 질의를 실행하고, 문제의 근본 원인을 찾아내기 위해 질의 실행 계획과 조인 방법과 같은, 질의 성능에 대한 더 자세한 통계를 얻을 수 있습니다. 조사 결과 수집된 통계에 기초하여, 더 나은 성능을 위해 질의를 좀더 최적화할 수 있습니다. "Last parsed SQL statement" 부분은 메모리에서 분석된 질의를 표시합니다. 메모리에서 분석된 질의는 현재 실행 중인 질의이기 때문에, 이 두 부분은 정확하게 일치합니다.

그 밖의 팁

위에서 살펴 본 내용 외에, 좀 더 적극적으로 모니터하려면, IDS 설치 중에 Informix에서 제공하는 경보 프

로그램(alarm program)을 수정할 수 있습니다. 경보 프로그램은 오류가 발생할 때 IDS에서 자동으로 호출하는 UNIX 셸 스크립트입니다. IDS는 모든 인스턴스 오류를 다섯 가지 레벨(severity levels)로 분류하는데, 여기서 1은 가장 낮은 레벨이고 5는 가장 높은 레벨입니다. 인스턴스 가정 오류(instance assertion failure)가 발생할 때 DBA에게 전자 우편을 보내거나 호출기에 메시지를 보내도록 경보 프로그램을 설정할 수 있습니다. 경보 프로그램과 샘플 경보 프로그램을 수정하는 방법에 대한 자세한 내용은 Administrator's Guide와 Informix OnLine Dynamic Server Performance Guide를 참고하십시오.

그 밖에도 데이터베이스가 정상적으로 실행되고 있는지 확인하려면 몇몇 필수적인 유지 관리 업무를 수행해야 합니다. 기본적인 업무로는 데이터와 색인의 무결성 확인 및 복구(oncheck -cDI DB명:Table명), 질의 옵티마이저에 대한 내부 통계의 지속적인 업데이트(update statistics), 사용하지 않은 공유 메모리 세그먼트의 지속적인 재사용(onmode -F) 등이 있습니다.

맺음말

Informix Dynamic Server 성능 모니터링은 계속해서 수행해야 하는 작업입니다. 이 작업의 가치는 통계를 수집하는 것 보다는 시스템 및 데이터베이스의 잠재적인 문제점을 확인하고 그 문제를 해결하는 데 있습니다.

여기에서 설명한 대로 모니터링을 효과적으로 수행하면 시스템 및 데이터베이스 문제점들을 가장 빠른 단계에서 성공적으로 확인하여 성능 조정에 미리 대비할 수 있을 것입니다.

객체 관계형 데이터베이스 응용 프로그램 개발 (제1부)

소개

ORDBMS(Object-Relational Database Management System: 객체 관계형 데이터베이스 관리 시스템)를 사용하는 개발자는 데이터베이스 분석 및 설계에 있어 관계형 DBMS 기술을 사용할 때의 전형적인 방법보다 훨씬 더 "총체적인" 접근법을 취할 때 이 기술을 최대한 활용할 수 있습니다.

전통적으로 관계형 DBMS는 비지니스 데이터에 대한 효율적이고 신뢰할 수 있으며 상대적으로 안정적인 데

이터 저장 영역으로 여겨졌습니다. SQL-92는 흥미로운 현실 상황에 대한 사실(fact)을 저장하기 위한 융통성 있는 프레임워크와 이런 사실을 기록하는 데이터를 다루는 메커니즘을 제공합니다. 그러나 SQL의 단순성 때문에 개발자가 성취하기 어려운 것이 많이 있습니다. 예를 들어 SQL-92를 사용하여 다음과 같은 간단한 질문을 수행한다고 가정해 봅시다. "이번 주에 생일이 있는 직원은 누구입니까?"

'질의가 어렵습니까?'라고 반문하실 것입니다. 우리가 실제로 사용하는 달력이 SQL-92 DATE 데이터 형식처럼 단순하다면 이 질문은 쉬울 것입니다. 그러나 이 질문은 2월 29일에 태어난 사람은 윤년을 제외하면 매년 2월 28일에 생일을 지낸다는 사실 때문에 복잡해집니다. 그 결과 다음과 같은 SQL-92 질의를 작성해야 할 것입니다.

```
FROM Employees E1,
     Employees E2
WHERE E1.Name ! = E2.Name
  AND (
       (
        ((MONTH(E1.DOB)=2(MONTH(E2.DOB)=2))
         AND
        ((DAY(E1.DOB) IN (28,29))
         AND(DAY(E2.DOB)IN(28.29)))
        )
        OR
        (
          (MONTH(E1.DOB) = MONTH(E2.DOB)) AND
      ( DAY(E1.DOB) = DAY(E2.DOB)
    )
 )
 ORDER BY 1,3;
```

그림 1: "오늘이 생일인 사람은 누구인가"라는 SQL-92 질의에 대한 불완전한 해결책

ORDBMS 기술의 기본 목적은 이 같은 복잡성을 피하고 보다 기능이 우수한 데이터베이스를 생성할 수 있도록 지원하는 것입니다. 이 장 뒷부분에서는 바로 이 문제에 대한 ORDBMS 솔루션에 대해 살펴볼 것입니다. 여기에서는 우선 ORDBMS를 사용하는 개발자를 위해 심도 있는 개발 방법론을 다룰 것입니다. 이 방법론은 관계형 및 객체 지향적 분석과 설계의 측면을 모두 결합한 것입니다. 그러는 과정에서 무엇이 가장 바람직하고 무엇이 가장 바람직하지 못한가에 대한 실질적인 조언을 제공할 것입니다.

데이터베이스용 객체 설계

ORDBMS를 이해하는 한 가지 좋은 방법은 이것을 일종의 소프트웨어 "기본 틀"로, 즉 응용 프로그램과 관련된 소프트웨어 모듈(객체 클래스)을 내장(embed)시킬 수 있는 프레임워크로 생각하는 것입니다. ORDBMS

가 순수한 객체 지향적인 DBMS, 응용 프로그램 서버 또는 TP-모니터 미들웨어와 같은 보다 전통적인 소프트웨어 프레임워크와 구분되는 점은 내장된 객체 클래스가 추상적인 또는 논리적인 데이터 모델 내에서 구현된다는 것입니다. 그 결과 ORDBMS 개발 팀은 두 레벨에서 작업을 해야 합니다. 한 팀은 C나 Java와 같은 언어를 사용하여 데이터베이스의 객체를 구현하고, 다른 팀은 이런 객체들을 결합하여 응용 프로그램 문제 영역의 수준 높은 요구 사항을 해결하는 것입니다.

사용자의 요구 충족이라는 개발 프로젝트의 최우선 목표와 함께, 모든 MIS 개발 프로젝트의 출발점은 시스템에서 지원해야 할 사용자 집단입니다. 여기서는 모든 사용자 요구 사항을 가리켜 문제 영역(problem domain)이라고 부릅니다. 이런 사용자 요구 사항을 제대로 해결하려면 다음과 같은 특성을 갖춘 정보 시스템이 필요합니다

- 전체성

  시스템은 문제 영역 관련된 모든 정보를 포함해야 합니다. 다시 말해서 데이터베이스를 공유할 수 있어야 합니다. 이는 데이터베이스에 사용자가 제어할 수 없지만 결정을 내릴 때 의존하는 "참조" 데이터(맵, 문서, 메타데이터 등)가 포함되어 있어야 함을 의미하는 경우도 있습니다.

- 정확성

  시스템은 문제 영역을 정확하게 나타낼 수 있어야 합니다. 모든 일이 그렇듯이 이것도 시간에 따라 변하기 때문에 시스템에서 그런 변화를 수용할 수 있어야 합니다.

- 일관성

  서로 상반되는 두 정보가 표시되는 것과 같이, 정보 시스템의 모호성으로 인해 시스템 사용자 사이에 혼동이나 오류가 생길 수 있습니다. 정보 시스템을 설계할 때에는 이런 문제 위험을 최소화하도록 해야 합니다. 이렇게 하기 위한 한 가지 좋은 방법은 모든 사실(fact)이 단일 장소에 저장되도록 데이터베이스 스키마를 설계하는 것입니다.

- 융통성

  RDBMS 기술 성공에서 얻은 가장 중요한 교훈은 실행 중에 질문에 답할 수 있는 기능이 대단히 중요한 가치를 지닌다는 것입니다. 원래 이러한 기능은 설계 시에 기대하지 않았던 것입니다. 인간은 원래 컴퓨터 시스템보다 융통성이 있으며 항상 새로운 질문과 요구 사항에 직면하게 됩니다. 실제

로 데이터베이스에서 이런 활동을 얼마나 잘 지원하느냐가 전략적 가치의 척도입니다.

- 효율성

  시간은 돈입니다. 시스템의 효율성은 운영 성능(최종 사용자 응답 시간), 개발 기간(개발의 용이성) 및 관리 비용 등 몇 가지 기준에 따라 측정해야 합니다.

이런 특성은 구축에 사용된 기술과 상관없이 모든 정보 시스템이 추구하는 목적입니다. 어떤 목적에 비중을 두느냐에 따라 프로젝트가 달라집니다. 주로 다른 소프트웨어 시스템에서 액세스하는 내장된 응용 프로그램이라면 효율성과 일관성에 가장 큰 비중을 두겠지만 특별히 융통성이 있을 필요는 없을 것입니다. 반면 관리 담당 의사 결정권자를 지원하는 시스템이라면 정확성과 융통성이 무엇보다 우선시 될 것입니다. 새로운 시스템을 사용할 사용자 집단의 목표와 목적을 제대로 파악하면 어떤 설계를 선택할 것인지 결정하는데 있어 큰 도움이 됩니다.

데이터베이스 분석 및 설계

ANSI의 3-Tier 데이터베이스 모델(Three-Tier Database Model)은 20년이 되었지만 예전과 마찬가지로 오늘날에도 널리 사용됩니다. 3-Tier 모델이란 다음에 해당하는 세 개의 "추상화(abstraction) 레벨"을 나타냅니다.

- 최종 사용자의 개념적 프레임워크 및 문제 영역에 대해 추정하는 방식. 이를 가리켜 개념적(Conceptual) 레벨 또는 외부(External) 레벨이라고 합니다.
- 정보를 저장하고 다른 프로그램에서 이 정보를 검색하고 다룰 수 있도록 기능을 제공하는 소프트웨어 서비스. 이것은 DBMS 소프트웨어에서 제공하는 추상화 레벨에 해당합니다. 이를 가리켜 논리적(Logical) 레벨 또는 내부(Internal) 레벨이라고 합니다.
- 데이터 저장 영역 구성에 사용되는 물리적 구조와 데이터 검색 및 처리에 사용되는 알고리즘. 이 물리적(Physical) 레벨의 세부 사항을 감추는 것은 관계형 DBMS 또는 객체 관계형 DBMS의 중요한 부수적 효과입니다.

모든 DBMS 분석 및 설계 방법론에서는 서로 다른 추상화 레벨에서의 문제 영역을 나타내는 기술과 계층

사이에서 서로 다른 모델을 변형하는 알고리즘을 다룹니다. 다음의 그림 2는 세 가지 모델의 배치 및 상호 작용에 대한 이상적인 구성도를 보여 줍니다.

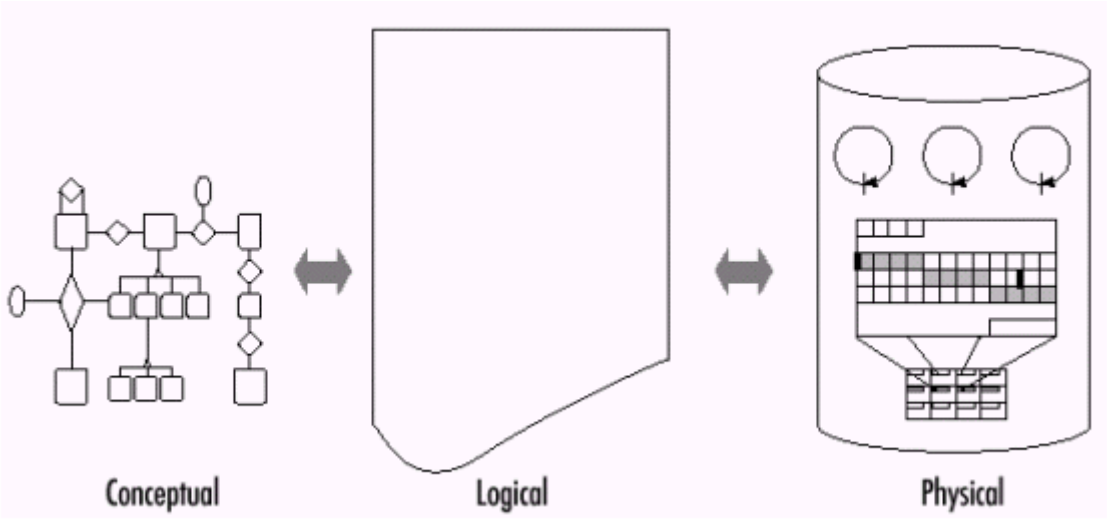

그림 2: ANSI 데이터베이스 모델의 세 계층

이 글에서 설명하는 방법론은 위 그림처럼 왼쪽에서 오른쪽으로 진행되는 매우 전통적인 것입니다. 원래 이 방법론은 액세스하는 정보에 대해 시스템 사용자가 어떻게 생각하는가를 완전히 설명하는 데 중점을 둡니다. 이런 개념적 뷰는 대상 객체 사용자와 이런 객체를 사용하여 설명하는 사실로 구성됩니다. 대부분의 웹 사이트에서 "사용자"는 이상화(idealize)됩니다. 즉 사이트 총 설계자는 사용자가 봐야 할 내용, 사용자가 봐야 할 위치 및 사용자의 행위 등을 미리 결정합니다.

개념적 모델은 객체 관계형 데이터베이스 스키마로 구현되는 논리적 데이터베이스 설계의 원동력입니다. ORDBMS에서 스키마 개념은 테이블, 뷰, 공통적인 비지니스 프로세스와 더불어 데이터 형식과 행동까지 포함하도록 확장됩니다. 마지막으로 ORDBMS는 개발자와 관리자가 논리적 스키마를 정밀하게 조정하여 시스템 성능을 최적화할 수 있도록 충분한 융통성을 제공합니다. 예를 들면, 물리적 계층에서 작업하고, 색인을 만들고, 데이터 파티션을 만들고, 적절한 성능을 얻기 위해 내장된 로직을 재구축하는 작업이 있습니다.

개념적 데이터 모델링

사용자에 따라 문제 영역에 대해 서로 다른 견해를 갖고 있습니다. 개발 팀의 첫번째 과제는 이런 내용을

모두 분석하여 문서화하는 것입니다. 개념적 모델에 의해 변경되는 확실한 지식을 바탕으로 이 작업을 시작하는 것이 매우 중요합니다. 전통적으로 사용자 인터페이스를 수정하는 것은 어려운 일이었습니다. 클라이언트 측 소프트웨어의 업그레이드는 대규모의 단계적 릴리즈를 통해 수행되었습니다. 그러나 HTML과 스크립트 언어 덕분에 웹 사이트에서의 최종 사용자가 경험할 내용을 수정하는 것이 매우 쉬워졌고, "웹 시간"은 가속화 프레임으로 참조되기 때문에 오늘날의 사용자 인터페이스(UI) 개발자는 백-엔드(back-end) 데이터베이스를 가능한 한 빨리 진전시키기 위해 심혈을 기울입니다.

그러나 변하지 않는 한 가지 사실은 개념적 모델을 문서화하는 최선의 방법은 다어어그램을 이용하는 것이라는 점입니다. 그래픽 사용자 인터페이스에서는 복잡한 그림을 쉽게 그릴 수 있습니다. 또한 서로 다른 그림의 의미를 형상화한 다수의 의미론적(semantic) 데이터 모델이 있기 때문에 개발 팀이 사용자의 생각을 정확히 알 수 있습니다. 이런 의미론적 모델에는 다음과 같은 것이 포함되어 있습니다.

- EER(Extended Entity-Relationship Modeling)

  확장 엔티티 관계 모델링)은 상속(inheritance) 및 추상적 데이터 형식(abstract datatypes)(객체의 영역 또는 클래스)과 같은 개념으로 전통적 엔티티 관계 모델링을 확장한 것입니다. EER 분석에 관한 추천할 만한 서적으로는 Toby Theorey의 Database Modeling and Design(Morgan Kaufmann)과 Fleming과 von Halle의 Handbook of Relational Database Design(Addison Wesley)이 있습니다.

- 최근에는 더 발전된 관계 모델링 기술이 개발되었습니다.

  이런 기술 가운데 가장 널리 알려진 것이 ORM(Object-Role Modeling)입니다. Terry Halpin의 Conceptual Schema and Relational Database Design(Prentice Hall)은 이런 기술과 개념 분석 전반에 대해 소개하는 훌륭한 서적입니다.

- 객체 지향적인 분석 및 설계를 바탕으로 매우 유용한 몇 가지 다이어그래밍 언어(diagramming language)가 개발되었습니다.

  가장 널리 알려진 것이 UML(Universal Modeling Language)입니다. 객체 지향적인 프로그래밍 언어를 사용하는 개발자를 위해 만들어진 UML은 ORDBMS로 수행하는 작업에도 쉽게 적용할 수 있습니다. 많은 설명서에서 UML을 설명하고 있으며 많은 툴에서 사용됩니다.

개발자들이 개념적 모델링을 자동화하는 데 도움이 되는 다수의 CASE(Computer Assisted Software Engineering) 프로그램이 있습니다. 불행하게도 대부분 이런 툴에서는 현재 ORDBMS의 모든 성능을 반영하지 못합니다. 예를 들면, 데이터베이스 테이블 내에서 결합된 데이터 형식의 팔레트 또는 이런 객체의 행동을 구현하는 기능에 대한 정보를 포착하지 못합니다. 하지만 시간이 지나면 이런 툴들의 기능이 향상될 것입니다.

EER 모델링

명확성과 간편함을 위해 여기에서는 예제에 EER 다이어그래밍 모델을 사용하였습니다. 모든 모델링 상황에서 EER을 사용할 수는 없지만, EER은 단순하고 알기 쉬우며 대부분의 경우 탁월하게 적용됩니다.

ORDBMS 방법론에서 EER을 사용하는 것이 전통적인 접근법과 다른 것은 각 엔티티의 내용을 다소 다르게 분석하는 것입니다. EER을 사용하는 사용자의 개념적 모델에서 높은 레벨의 개요에 이르렀을 경우, 객체 지향적인 기술을 사용하여 엔티티들을 객체 클래스의 최소 집합으로 나눌 수 있습니다. 전체 엔티티를 하나의 객체 클래스로 처리하는 경우도 있습니다. 그러나 엔티티를 구성하는 요소(각 요소의 종류(kind)와 엔티티에서 사용될 형식의 역할(role))를 데이터베이스 내에서 나눌 수 없는 의미의 단위로서 효과적으로 모델링하는 것이 보다 일반적입니다(객체 지향적인 분석의 기술을 사용하여).

예를 들어 고전적인 Employee/Department/Product/Customer 데이터베이스 스키마를 고려해 봅시다. 당연히 여기에서 살펴보는 예제는 어느 정도 업데이트 되었으며 다소 복잡해서 전통적인 RDBMS로는 처리할 수 없는 전형적인 현실적 세부 사항이 포함되어 있습니다. 그림 3은 이 예제에 대한 높은 수준의 개념적 모델을 나타내는 EER 다이어그램입니다.

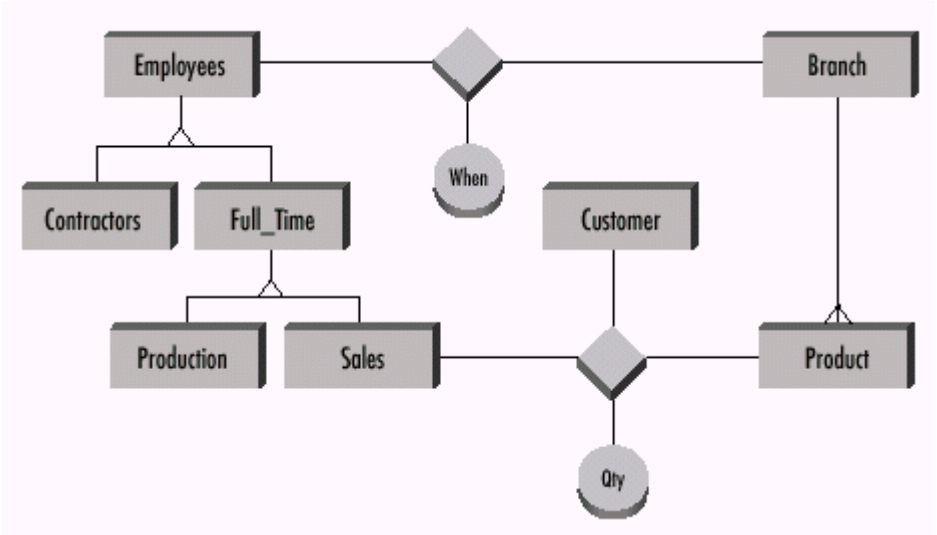

그림 3: "Boxes-R-Us Inc"의 EER 모델

다음은 이 응용 프로그램에 대해 간략하게 설명한 것입니다.

"Boxes-R-Us Inc.(회사)에서는 색상과 크기가 다양한 광범위한 판지 상자 Products를 제조하여 Customer에게 판매합니다. 이 회사 제품은 일련의 Branch를 통해 배포됩니다. 이 회사의 Employees는 각 Branch에서 일하지만 시간이 지남에 따라 Branch 간에 순환 근무를 할 수도 있습니다. Employees는 서로 다른 하위 형식, 즉 Contractor와 두 유형의 Full_Time 직원(Production 근로자와 Sales 직원)으로 나뉩니다. Employees는 직원 유형에 따라 서로 다른 일정으로 임금을 지급 받기 때문에 이런 분류가 생긴 것입니다. Sales 직원은 Customer에게 일정한 수량의 Product를 판매하는 일을 담당합니다."

ORDBMS를 사용하여 작업할 때에는 EER 모델의 특징을 이루는 모든 개념은 그대로 유지됩니다. 전통적인 EER 모델과 마찬가지로 다이어그램에 다음을 문서화하는 것이 좋습니다.

- 엔티티 간의 has_a 관계의 패리티를 추정합니다.

  이 예제에서는 "crows feet" 기술을 사용하여, 각 Branch에서 다수의 Product를 만들지만 어떠한 Product도 여러 Branch에서 제작되지는 않음을 표시하고 있습니다. 또한 의무(compulsory) 관계(약한 엔티티의 인스턴스 존재가 관련 강한 엔티티의 존재에 의존하는 것)와 비지니스 규칙과 같은 기타 개념도 도움이 됩니다.

- 모든 키(key)를 설명합니다.

키 개념은 두 가지 이유 때문에 중요합니다. 첫째, 키는 데이터베이스 설계자가 스키마의 각 사실(fact)이 단 한번만 표시되도록 보장할 수 있는 방법인 일반화 프로세스(normalization process)를 구현합니다. 둘째, 키는 저장되는 데이터의 정확성을 보장하기 위해 강화해야 할 데이터에 대한 중요한 규칙을 나타낼 경우가 많습니다.

- 앞의 그림에는 엔티티 간의 간단한 has_a 관계보다는 다소 복잡한 관계 쌍이 포함되어 있습니다. 첫 번째 것은 Employees 계층 구조에 반영됩니다. 그와 같은 상속(inheritance) 계층 구조는 RDBMS보다 ORDBMS에서 더 직접적으로 나타낼 수 있습니다. 두 번째 종류의 관계는 자격 부여(qualified)입니다. 예를 들면, Employees는 일정 기간 동안 Branches에 근무합니다.

이런 엔티티 몇몇의 내부 구조에 대해 자세히 살펴보겠습니다. 비록 여기에서 상당히 표준적인 Entity 다이어그램을 사용하지만 보다 자세한 설명을 위해 UML 스타일의 Class Diagram을 사용하는 것이 더 좋을 수도 있습니다. RDBMS와 ORDBMS로 EER 모델링을 하는 데 있어서 중요한 차이점은 ORDBMS 모델의 경우 스트롱 타이핑(strong typing)이 지원된다는 것입니다. 즉 종류(관계 이론에서는 영역(domain)이라고 함)가 같은 모든 것은 같은 것으로 구별되어야 합니다. 이런 모든 영역을 EER 모델의 일부로 분류하려면 실제로 각 엔티티의 요소를 구성하는 데이터의 종류에 특히 주의를 기울여야 합니다. 예를 들면 다음과 같습니다.

Employee	
Attribute_Name	**Attribute_Kind**
Id	Employee_Num
Name	Person_Name
Address	Mail_Address
Resume	Document
Lives_At	Geo_Point
Date_of_Birth	Date

Customer	
Attribute_Name	**Attribute_Kind**
Id	Customer_Num
Contact	Person_Name
Address	Deliver_Address
Bill_Address	Mail_Address

그림 4: EER 모델의 두 엔티티에 대한 상세 구조

이런 엔티티에 대해서 다음 사항을 지적할 수 있습니다.

우선 서로 다른 데이터 종류가 얼마나 되는지, 어떻게 스트롱 타이핑(strong typing)이 개념적 모델에서 보다 많은 의미 정보를 포착하는지를 주목하십시오. RDBMS 기술을 사용할 경우에는, 각 속성을 최소 구성 요

소로 나누고 각각에 SQL-92 형식을 할당하는 것이 일반적입니다. 예를 들면 여기에서는 Employee_Num과 Customer_Num 대신 INTEGER를 사용할 수 있을 것입니다. 그럴 경우 정보 종류를 구분하기 위해서 속성의 이름을 사용해야 하는 불행한 결과가 생깁니다. 그러나 ORDBMS를 사용할 경우에는, 각 데이터 객체 종류에 상응하는 별도의 형식을 생성하고 엔티티에 그 역할에 대한 속성의 이름을 예약해 두는 일을 쉽게 할 수 있습니다.

스트롱 타이핑은 프로그래밍에서 매우 좋은 습관입니다. 스트롱 타이핑을 인식하는 스키마에서는 시스템 카탈로그를 사용하여 서로 다른 스키마 부분들이 함께 어울리는 방식에 대한 모든 종류의 정보를 알아낼 수 있습니다. 예를 들어 데이터베이스에 포함되어 있는 Employees에 대한 모든 것을 알아야 할 필요가 있을 수도 있습니다. 여기의 스키마에서 Employees는 Employee Numbers (직원 엔티티의 기본 키는 Id라는 이름의 속성으로, Employee_Num이 되도록 지정됨)에 의해 식별됩니다. 따라서 비록 일반적인 외부 키 제한이 전혀 정의되어 있지 않더라도, 이 형식의 칼럼이 있는 스키마의 모든 "테이블"은 Employees와 관계가 있다고 말할 수 있습니다.

스트롱 타이핑은 특히 의사 결정 지원 데이터베이스에 효과적입니다. 데이터 웨어하우징이 목적이라면, 개발자는 여러 작동 시스템에서 추출한 데이터를 포함하는 여러 테이블을 생성하여 단일 데이터베이스로 로드할 수도 있을 것입니다. 전형적으로 Employee_Num과 같은 데이터 객체는 서로 다른 시스템 간의 레코드를 상호 확인하는 데 사용된 값입니다. 일단 이 데이터가 모두 단일 데이터베이스로 로드되면, 어떤 데이터가 있는지 알지 못할 수도 있는 개발자는 스트롱 타이핑을 사용하여 "직원에 대해 어떤 정보를 가지고 있는가?"와 같은 중요한 메타데이터 질문에 대답할 수 있습니다.

이 레벨에서 분석할 때에는 작업하고 있는 데이터의 종류에 대해 너무 자세히 들어가지 않는 것이 좋습니다. RDBMS를 사용할 경우에는 두 개의 Address 속성을 First_Line, Second_Line, City, State 및 Zip으로 나눌 수 있을 것입니다. ORDBMS를 최대한 활용하려면 다소 다른 접근법을 사용해야 합니다. 일단 특정 유형의 데이터가 사용된 문제 영역 내에서 모든 장소를 확인하고 나면, 모두가 공통적으로 갖고 있는 것에 대해 보다 쉽게 이해할 수 있을 것입니다. 그런 다음 일단 객체의 구조와 행동을 개발하여 스키마 전체에서 다시 사용할 수 있습니다.

직원의 이력서와 직원이 살고 있는 지리적 위치를 포함하는 속성들은 RDBMS 개발자와 어울리지 않아 보입니다. 그러나 다수의 객체 관계형 데이터베이스에서 이런 종류의 데이터를 관리합니다. 이런 환경에서 개발자는 InformixR DataBladeR 모듈이라는 확장 번들을 구매하여 이런 종류의 데이터를 관리할 수 있습니다.

Products	
Attribute_Name	**Attribute_Kind**
Id	Product_Number
Name	String
Dimensions	Physical_Size
Capacity	Mass
Available_In	SET(Color)
Price	Currency

그림 5: Products 엔티티의 내부 구조

Products 엔티티를 구성하는 요소에 대해서 다음 두 가지 사항을 지적할 수 있습니다.

ORDBMS 데이터 모델은 1차 정규화 형식이 아닌(non 1NF) 속성을 지원합니다. 여기에서 특정 상자에 사용할 수 있는 색상 범위를 저장하는 속성 사용 예제를 볼 수 있습니다. 이 단계에서는 이런 종류의 구조가 물리적으로 어떻게 나타날 것인지에 대해 너무 큰 걱정을 할 필요가 없습니다. ORDBMS 테이블에서 단일 칼럼에 데이터 값들을 저장할 수 있다고 해도, 이것은 최선의 방법일 수도 있고 아닐 수도 있습니다.

당사에서 제조하는 각각의 상자에는 물리적 사양이 있습니다. 그 가운데 가장 중요한 것은 상자의 크기(길이, 너비, 높이)와 재질의 강도(찢어지지 않고 견딜 수 있는 자재의 양)입니다. Boxes-R-U에 대한 사업적 효율성은 얼마나 쉽게 고객이 주문할 수 있도록 되어 있는가와 결부되어 있습니다. 다시 말해서 데이터베이스에서 물리적 자재와, "12인치 x 10인치 x 8인치"인 물건이 "31센티미터 x 31센티미터 x 31센티미터"인 상자 안에 들어갈지 여부와 같은 것에 대해 직접적으로 판단할 수 있도록 허용할 경우 이상적일 것입니다. ORDBMS 확장성 덕분에 이 같은 객체를 직접 질의 언어에 포함할 수 있으며, 이 점은 비지니스 응용 프로그램을 구축할 때 이 기술에서 얻을 수 있는 최대의 장점입니다.

그러면 이런 활동이 유용한 이유는 무엇일까요? 그것은 응용 프로그램에서 서로 다른 종류의 데이터에 대해 최소의 카탈로그를 만들 수 있도록 해주기 때문입니다. 앞으로 살펴 보겠지만, ORDBMS의 사용자 정의 형식(user-defined type)과 사용자 정의 함수(user-defined function) 기능을 사용하여 이런 다양한 종류의 데

이터를 구현하는 것이 개발 프로세스의 일부로 포함됩니다.

객체 지향적인 분석 및 설계

객체 지향적인 분석 및 설계의 핵심은 객체의 인터페이스(객체의 처리 수단)가 그것의 구현(내부의 데이터 구조와 논리)에 대한 상세 내용(detail)과 분리되어야 한다는 생각입니다. ORDBMS 개발에 있어 역할 분석은 확장 형식 시스템으로 작업하기 위한 개념적인 틀입니다. 새로운 객체는 사용자 정의 형식 메카니즘을 사용하여 구현합니다.

이런 형식의 행위는 사용자 정의 함수 내에서 구현됩니다.

첫번째 작업은 EER 모델에서 확인할 수 있는 모든 다양한 데이터 종류의 목록을 생성하는 것입니다. 여기에는 엔티티와 이런 엔티티의 속성을 구성하는 다양한 종류의 데이터가 모두 포함됩니다. 다음은 이 글의 예제에서 확인할 수 있는 다양한 종류의 데이터에 대한 부분적인 목록을 구성하는 표입니다(특별한 순서 없음).

Products_Number	String	Physical_Size	Mass
Color	Employee_Num	Person_Name	Mail_Address
Geo_Point	Document	Customer_Num	Deliver_Address
Currency	Bonus_Plan	Period	Quantity
Products_Number	Employees	Full_Time_Emp	Sales

그림 6: Boxes-R-Us EER 스키마의 영역 목록

이 접근법의 중요한 장점은 공유 데이터 객체를 한번만 개발하면 된다는 것입니다. 이러한 객체는 스키마의 여러 장소에서 사용될 수도 있습니다. 이렇게 하면 분석의 초기 단계에서 오류나 실수를 발견할 경우 비교적 쉽게 오류를 고칠 수 있습니다. 하지만 이런 객체에 대한 정의가 스키마의 여러 장소에 걸쳐 분산되어 있다면 문제를 해결하는 것이 정말로 큰 작업이 될 것입니다.

흔히 같은 것(동의어)에 대해 서로 다른 용어를 사용하거나 다른 것(반의어)에 대해 같은 용어를 사용하는

경우가 많습니다. 객체 지향적인 기술을 사용하면 이런 모호성을 없애는 데 도움이 됩니다. 객체 지향적인 소프트웨어 엔지니어링 기술에서는 "객체"의 개념을 상태(state)와 행위(behavior)를 한데 모으는 의미의 최소 단위로서 인식합니다. 비록 이름에 동의할 사람이 없을지라도, 두 객체가 처리되는 방식이 충분히 유사할 경우 두 객체는 같은 것일 가능성이 큽니다. 마찬가지로 이름은 같지만 정의가 완전히 다른 두 종류의 데이터는 서로 다른 객체일 것입니다.

예를 들어 Mail_Address와 Delivery_Address 영역은 동의어가 될 수 있는 후보입니다. 다음 그림은 객체 인터페이스를 제공하기 위해 UML 표준을 사용하는 객체 각각에 대한 인터페이스를 나타냅니다. UML 클래스 다이어그램에는 개발자가 형식을 사용하여 구조의 요소를 직접 다룰 수 있는지 여부를 비롯하여 객체의 정적 구조가 표시됩니다. 아래 그림에서 분명한 것은 이 두 종류의 데이터가 매우 밀접하게 연관되어 있지만, Delivery_Address의 인스턴스에는 세부적인 배달(delivery) 지시 사항을 제공하는 문서를 포함할 수 있는 점이 다릅니다.

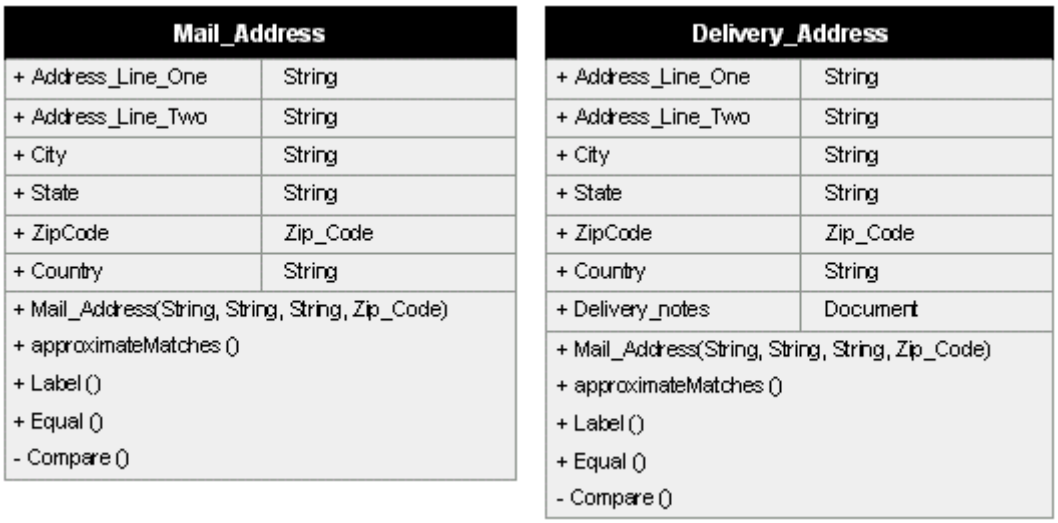

그림 7: 두 개의 비슷한 객체에 대한 ULM 클래스 다이어그램

이러한 분석을 구현 계획으로 전환할 때 개발자에게 두 가지 선택권이 있습니다. 하나는 이런 각각의 것을 전혀 관계가 없는 형식으로 개발하는 것입니다. 다른 방법은 형식 상속(type inheritance)을 사용하여 Delivery_Address에서 Mail_Address 형식의 구현 방식을 다시 사용하는 것입니다. 형식 상속을 사용하면 개발자가 작성해야 할 코드의 총 분량이 줄어듭니다. Mail_Address에서 주소 라벨 서식을 지정하기 위해 사용한 코드는 Delivery_Address에 다시 사용할 수 있습니다. 물론 필요할 경우 언제라도 행위를 오버로드할 수 있는 옵션도 있습니다.

일부 데이터 유형은 패턴 예제입니다. 패턴은 다수의 형식에 사용되는 반복적인 구조입니다. 패턴을 사용하면 몇 가지 특성을 바탕으로 전체 설계를 완성할 수 있기 때문에 유용합니다.

열거된(enumerated) 객체는 아주 간단한 패턴 예제입니다. 이 데이터베이스에서는 상자를 만드는 데 사용할 수 있는 Color의 범위가 매우 적다는 것(검은색, 흰색 및 갈색 정도)이 그 예가 될 것입니다. 데이터의 일관성을 최대한 유지하기 위해 개발자는 열거된 형식(enumerated type)의 구현 방법을 사용하여 이런 규칙을 적용할 수 있습니다. 내부적으로 Color 객체의 인스턴스는 단순히 2바이트 정수로 저장될 것입니다. 그러나 클라이언트 프로그램에 값을 리턴하고 테이블에 값을 삽입하는 코드에서 개발자는 유효하지 않은 인스턴스를 불가능하게 만드는 논리를 작성할 수 있습니다.

이를 달리 설명하면, 특정 방식으로 사용되기 위해 새 객체는 반드시 특정 행동을 구현해야 한다는 것입니다. 예를 들어, 고객이나 판매 직원은 제품 카탈로그를 검색할 때 특정 기준에 따라 검색할 것입니다. 예를 들어 3파운드 무게의 물건을 담을 수 있는 상자만을 원할 수도 있습니다. 그들은 효과적인 검색을 위해 질의를 통해 일치하는 상자를 가용 용량의 오름차순으로 리턴하도록 할 수 있습니다.

질의는 다음과 같을 것입니다.

"3파운드 이상의 물건을 담을 수 있는 모든 상자를 파운드 순으로 보여주십시오."

```
SELECT P.Id,
  P.Capacity
  FROM products P
  WHERE P.Capacity > '3 POUNDS'
  ORDER BY P.Capacity ASC
```

그림 8: 워크로드 질의의 예제

이런 종류의 질의가 중요하다는 것을 알게 됨으로써 사용자는 이 객체의 인스턴스를 다른 것과 비교하고 정렬하며 색인화하고자 하리라는 것을 추론할 수 있습니다. 이런 종류의 기능을 지원하려면 비교 연산자 함수(LessThan, LessThanOrEqual, Equal, GreaterThanOrEqual, GreaterThan, NotEqual)와 B-트리 지원 함수

(Compare())를 구현하기 위한 새로운 형식이 필요합니다.

다음 그림에서는 Mass 데이터 형식에 대한 UML 다이어그램을 볼 수 있습니다. 이 다이어그램에서 각 객체의 행동에 대한 완전한 정의를 사용한 방식에 주목하시기 바랍니다. 이를 통해 객체들을 서로 구분하고 여러 형식에 공통되는 "예약된" 행위를 식별할 수 있습니다.

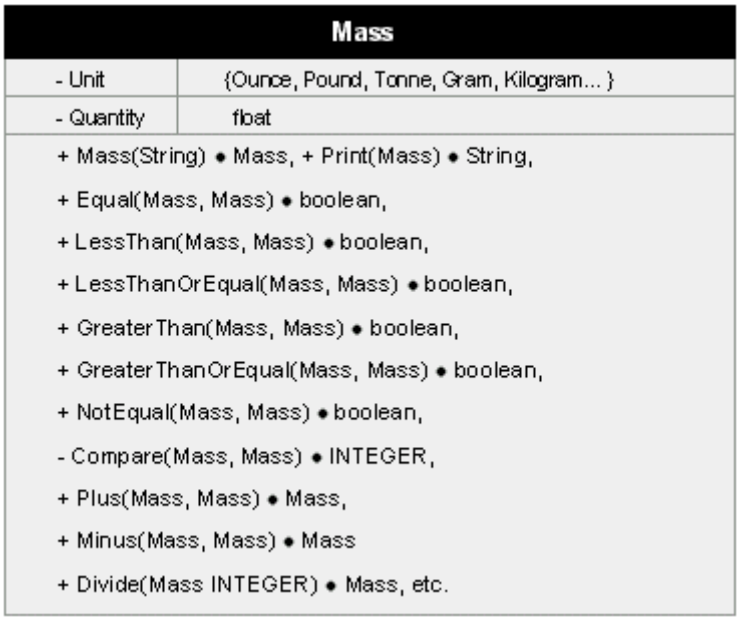

그림 9: 비교 연산이 지원되는 UML 클래스 다이어그램의 예

다른 색인화를 사용하는 다른 형식에는 다른 지원 함수가 필요합니다. 예를 들어 특정 직원이 지점에서 근무한 기간을 저장하는 데 사용되는 Period 데이터 형식을 생각해 봅시다. Period는 시작일과 종료일 또는 날짜 및 시간이 포함된 객체입니다. 오버랩되는 기간과 같은 개념을 효과적으로 지원하려면, 형식에 R-트리 액세스 방식에서 사용할 수 있는 행위가 포함되어 있어야 합니다.

직원 계층 구조를 보다 자세히 살펴봅시다. Employee를 하위 범주로 나누는 주된 이유는 Boxes-R-Us에서 수행하는 일의 종류에 따라 서로 다른 유형의 직원이 서로 다른 임금을 받기 때문입니다(보수가 서로 다르게 계산됨). 다음은 Production 근로자와 Sales 직원의 차이를 나타내는 그림입니다.

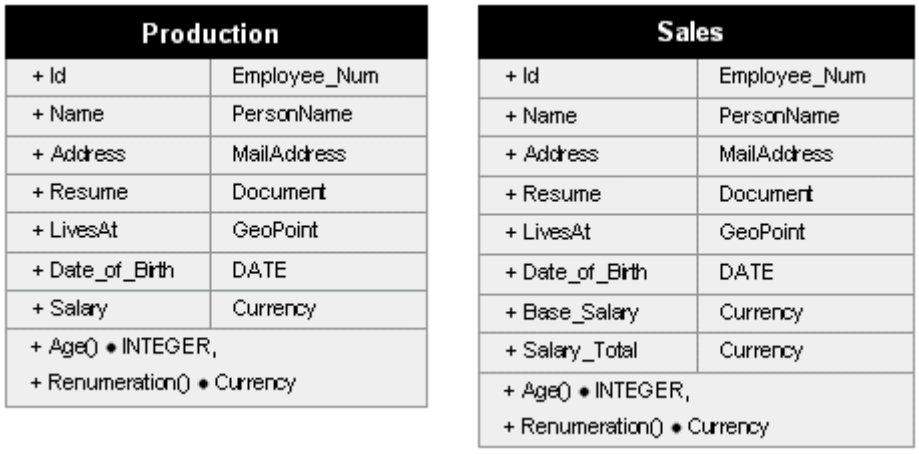

그림 10: 엔티티 형식에 대한 UML 클래스 다이어그램

이 정의에서 오버랩의 정도에 주목하십시오. 서로 다른 유형의 직원에 대한 임금 지급 방식과 관련된 속성을 제외하면 이 두 엔티티는 동일합니다. 또한 이 두 엔티티가 공통적인 행위를 공유하는 것을 알 수 있습니다. 서로 다른 직원 유형을 관찰하면 비슷한 패턴을 발견할 수 있었습니다. 그러나 보수를 계산하는 데 사용되는 알고리즘은 직원 범주에 따라 다릅니다.

이 단계에서는 행위를 형식화(formalize)하는 것이 단연 중요한 과제입니다. 지금까지 소개한 모든 클래스에는 일종의 행위가 포함되어 있습니다. 몇몇 경우에는 행위를 표현하는 방식이 분명하지 않습니다. 예를 들어 Mass 클래스에는 미터법과 영국 표준 단위가 모두 필요할 수도 있으며, 그럴 경우 변환 처리도 필요할 것입니다. 때로는 행위가 바뀔 수도 있음을 알게 될 것입니다. Sales 직원의 연봉을 계산하는 데 사용되는 정확한 알고리즘은 경영층에서 영업팀에게 지시하는 목표에 따라 매년 다릅니다.

여기에서는 방법론을 소개하면서 하향식 접근법을 사용했습니다. 문제를 최소의 부분들로 나누기 전에 먼저 보다 큰 구조를 확인하고 분석하는 것입니다. 이것 만큼 유용한 또 다른 접근법은 문제의 최소 구성 요소부터 분석하기 시작하여 어떻게 결합되어 있는가를 확인하는 것입니다. 상향식 분석이라고 하는 이 접근법은 보다 상위 레벨의 구조(예를 들어 테이블과 테이블 간의 관계)를 더 빠르게 변경하고 더 간단하게 수정할 수 있다는 장점이 있습니다. 데이터베이스 내에서 객체 형식의 광범위한 팔레트를 결합하는 데 먼저 초점을 맞추면 보다 상위 레벨 구조에 대한 질문에 더 간단하게 답할 수 있을 것입니다. 프로젝트에 어떤 것이 적합한가를 결정하는 것은 취향과 경험의 문제입니다.

데이터 흐름, 워크로드 및 질의

이 단계에서 전체 응용 프로그램에 대해 고려해야 할 또 다른 중요한 측면은, 정보 시스템이 지원하는 공통적인 질의 및 비지니스 프로세스의 집합인 워크로드입니다. 워크로드가 정리되면 다음 몇 가지 측면에서 유용합니다.

첫째, 개발자가 논리 구현 비지니스 운영을 ORDBMS에 내장하여 클라이언트 프로그램에서 직접 호출할 수 있습니다. 관계형 DBMS에서는 이 같은 데이터베이스 프로시저를 한동안 지원했습니다. 그러나 이 기술이 개선된 성능을 제공하기는 하지만 해당 DBMS에 국한된 프로시저 언어를 사용해서 코드를 개발해야 하기 때문에 대다수 개발자가 선뜻 이 기술을 사용하지 못했습니다. ORDBMS에서는 Java와 JDBC 같은 표준 API를 사용하는 개방형 언어를 사용하여 이런 데이터베이스 프로시저를 구현할 수 있습니다.

둘째, 초기에 ORDBMS 기술을 채택한 사람들이 겪은 경험 중 하나는 전통적으로 외부 프로그램에 연결되었던 논리를 ORDBMS에 도입함으로써 시스템 전반에 걸쳐 성능과 융통성을 향상시킬 수 있었다는 점입니다. 예를 들어, Boxes-R-Us에서 각 직원에게 생일날 카드나 메시지를 보내고자 할 수도 있습니다. 앞에서 보았듯이 일치하는 생일을 찾을 때 윤년의 2월 29일에 태어난 사람을 고려해야 하기 때문에 SQL-92에서는 "오늘 생일을 맞이한 사람은 누구인가"와 같은 너무도 간단한 작업을 지원하는 것이 불가능합니다. 윤년이 아닐 경우 이런 사람은 2월 28일에 생일 축하를 할 것입니다. 이상적이라면 다음 질의를 실행하여 문제의 복잡성을 처리하도록 해야 할 것입니다.

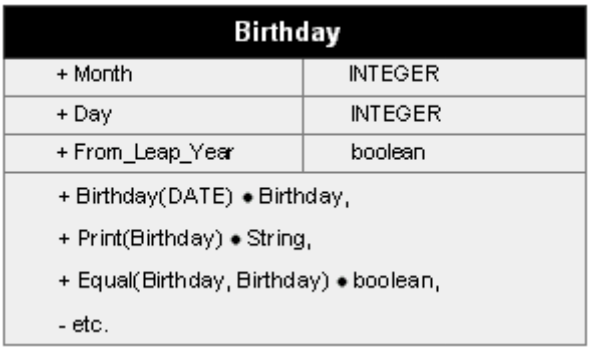

그림 12: Birthday에 대한 UML 클래스 다이어그램

ORDBMS 데이터 모델의 중요한 원칙은 이 같은 형식을 테이블의 구조나 칼럼을 정의하는 데 사용할 필요가 없다는 것입니다. 그림 11과 같이 이 객체의 행위를 사용하여 다른 전통적 데이터에 대해 복잡한 질의 작업을 구현할 수 있습니다. 또한 대부분의 객체 지향적인 방법론에서는 실제로 필요한 것이 객체가 아니라 단순히 함수나 프로시저인 경우가 있다는 점을 인식하고 있습니다. 객체 지향적인 모든 것이 쓸데없는 소리로만 들리는 독자라면, 순전히 기능(functional) 또는 모듈(modular) 모드에서도 쉽게 ORDBMS 데이터베이스 설계에 대해 생각할 수 있으며 전적으로 가능하다는 사실에 만족을 얻을 수 있을 것입니다.

세째, 워크로드 카탈로그는 성능과 확장성 테스트 환경에 대한 기준을 제공합니다. 새로운 소프트웨어 시스템에서 기술적으로 실패하는 가장 공통적인 원인 중 하나는 개발자가 구현할 내용을 생산 환경과 같은 실제 환경에서 테스트하지 않는다는 것입니다.

맺음말

이 글에서는 객체 지향적인 데이터베이스 응용 프로그램에 대한 초기 개발 방법론에 대해 살펴보았습니다. ORDBMS를 최대한 활용하려면 개발자는 응용 프로그램의 문제 영역(problem domain)에 대해 통상적으로 관계형 DBMS 개발에서 했던 것 보다 많은 정보를 수집해야 할 필요가 있습니다. ORDBMS에서는 소프트웨어의 모듈을 내장하여 문제 영역에서 객체의 상태와 행위를 나타낼 수 있기 때문에, 개발자는 전통적으로 외부 프로그램, 미들웨어 또는 클라이언트 인터페이스의 작업이었던 응용 프로그램 측면에 주의를 기울여야 합니다.

EER 모델링과 UML 접근법을 함께 사용하여 클래스 정의를 시각적으로 나타냄으로써 다음을 얻을 수 있었

1. 문제 영역에 관련된 객체에 대한 최소한의, 그러나 전체적인 목록

2. 기록하려는 사실(fact)을 정확하고 일관성 있게 표현하기 위해 이러한 객체들을 구성하는 방법

본 TechNotes의 다음 호(Vol 11)에서 연재될 객체 관계형 데이터베이스 응용 프로그램 개발 제2부에서는 이 글에서 설명한 의미상의 모델(semantic model)을 ORDBMS 데이터 모델의 다양한 기능을 사용하여 나타낼 수 있는 절차에 대해 설명할 것입니다. 이 세부적인 설계 프로세스의 일부로서, 서로 다른 메커니즘 사이에서 성능과 융통성의 득실에 대해 살펴보겠습니다.

주요 ERP 응용 프로그램을 위한 Informix Dynamic Server 최적화

소개

ERP(Enterprise Resource Planning) 소프트웨어 패키지는 1990년대 들어 세계 시장에서 보다 효율적이고 효과적이며 경쟁적인 지위를 차지하려는 기업들 사이에서 최고의 화두로 등장했습니다. IS(Information System) 관리자는 ERP 소프트웨어 패키지를 이용해 Year 2000 호환성 문제를 해결할 수 있습니다. 이러한 소프트웨어 패키지 덕분에 비지니스 프로세스 표준 수립이 훨씬 더 용이해졌으며 이를 통하여 기업체 내에서 전통적인 조직적 "연결 구조"의 통합이 가능해져 효율성을 개선할 수 있게 되었습니다. 이제는 기업체가 단일 ERP 시스템 상에서 운영되기 때문에 생산직 사원과 회계 담당 직원 간의 의사 소통이 가능해졌으며, 기업 내에서 국제적으로 서로 다른 지역에 있는 사무실에도 동일한 생산 표준이나 회계 표준을 비롯한 모든 것을 공유할 수 있게 되었습니다.

ERP 업체는 막대한 연구 및 개발 자원을 투자하여 응용 프로그램의 기능을 향상시켜 왔습니다. 비지니스 논리가 더 복잡해지고 저장된 데이터의 중요성이 더 높아짐에 따라 이러한 응용 프로그램들은 기반이 되는 관계형 데이터베이스에 큰 부담을 갖게 됩니다. 적절한 튜닝과 관리를 하지 않을 경우 이런 시스템은 IS 관리자에게도 조만간 부담이 될 것입니다.

이 글에서는 현재 전세계에서 가장 널리 사용되는 4대 ERP 응용 프로그램의 사용을 최적화하기 위해

Informix Dynamic Server(IDS)를 튜닝하는 방법에 대해 다룹니다. 4대 ERP 응용 프로그램에는 SAP, PeopleSoft, Baan, Lawson이 있습니다. 각 응용 프로그램마다 사용자가 데이터를 얻는 방법과 SQL 작성 방식이 매우 다릅니다. 각 업체마다의 튜닝 팁을 보면 응용 프로그램 환경의 일반적인 구조에 대한 설명으로 시작됩니다. 이런 내용을 이해하면 왜 특정 튜닝 매개변수를 깊이 살펴볼 필요가 있는지 알게 됩니다. 여기에서는 먼저 각 응용 프로그램에 대한 튜닝 방법을 설명하고 네 응용 프로그램에 모두 적용되는 일반적인 튜닝 팁을 소개합니다.

특히 이 글에서는 다음과 같은 내용을 다룹니다.

- Informix Dynamic Server
- SAP 튜닝
- PeopleSoft 튜닝
- Baan 튜닝
- Lawson 튜닝
- 모든 ERP 응용 프로그램에 대한 일반 튜닝

현재 사용하고 있는 프로그램이 위의 ERP 제품이 아니더라도, 여기에서 소개하는 튜닝 팁은 다른 OLTP 환경에도 적용할 수 있습니다.

Informix Dynamic Server

Informix Dynamic Server 버전 7이 출시된 지 5년이 되었습니다. 특별히 추가 프로그래밍 작업 없이도 특정한 IDS 기능들을 활용할 수 있는 ERP 업체의 기술 덕분에 IDS 아키텍처는 고객에게 매우 편리한 아키텍처가 되었습니다. 전반적으로 모든 ERP 업체는 특정 데이터베이스를 위해 응용 프로그램에 추가 프로그래밍 작업을 하지 않습니다. IDS를 사용하면 그럴 필요가 전혀 없으며, 이 글을 통하여 그 이유를 알게 될 것입니다.

다음과 같은 기술을 사용하여 ERP 응용 프로그램에 대해 IDS를 튜닝할 수 있습니다.

- **$ONCONFIG 파일**

  - 구성 매개변수 튜닝

- **분할화(fragmentation)로 데이터 및 색인 분할(partitioning)**

  - I/O 최소화

- **Informix의 병렬 기능(PDQPRIORITY > 0)으로 병렬 색인 생성**

  - 색인 생성 시간 최소화

- **UPDATE STATISTICS**

  - 비용 기반 옵티마이저가 시스템 정보를 더 많이 확보하여 SQL 분석 능력 향상

- **ERP에서 만드는 도구로 DBA에게 도움이 되는 작업**

  - 업체에서 제작

위의 각 항목은 각자의 환경 내에서 효율성을 개선하기 위해 검토하고 튜닝해야 할 잠재적 영역입니다.

Informix는 ERP 업체와의 협력을 통해 성능을 개선한 IDS 버전 7.3을 출시했습니다. 7.3에는 응용 프로그램을 재컴파일하거나 전혀 변경하지 않고 OLTP 성능을 향상시키는 데 사용할 수 있는 50가지 이상의 새로운 기능이 포함되어 있습니다. 여기에는 새로운 옵티마이저 기능과 기타 향상된 기능에서 백업 및 복원 기

예를 들어 PeopleSoft 내에서 고객은 생산 환경을 전혀 변경하지 않고도 단지 이전 IDS 버전을 7.3으로 업그레이드함으로써 평균 20%의 성능 향상 효과를 볼 수 있었습니다. 추가된 기능의 예는 그림 1에 있으며 보다 자세한 내용은 $InformixDIR/release/en_us/0333/IDS_7.3?파일 안에 있는 7.30 릴리즈 정보를 참조하십시오.

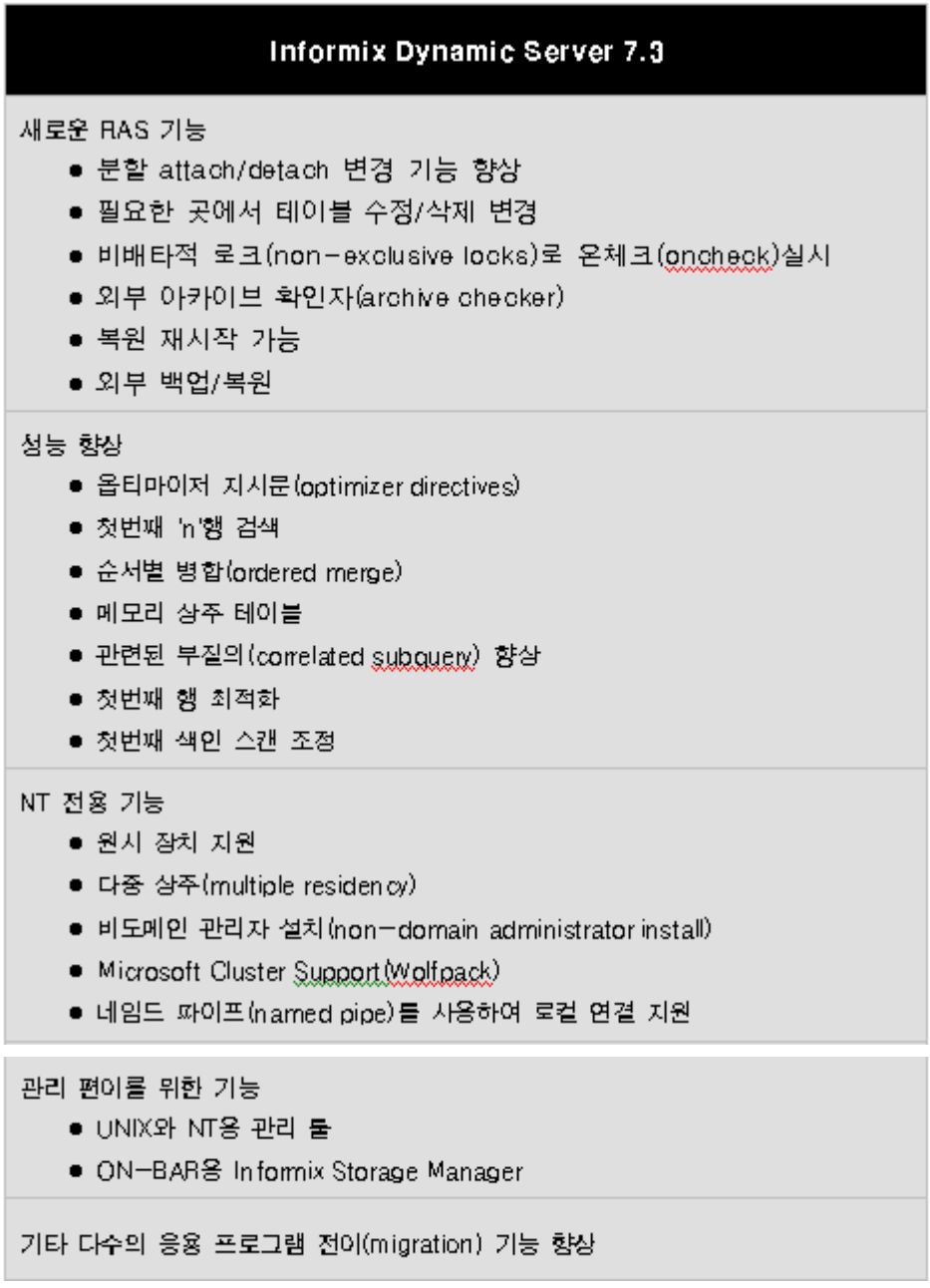

그림 1: Informix Dynamic Server 7.3의 새로운 기능

SAP 튜닝

SAP의 아키텍처에는 "BASIS 커널"이라는 중간 계층(middle tier)이 있습니다. 이 BASIS 커널은 SAP의 ABAP 코드로 작성된 사용자 요청을 취해서 이를 Informix 제품에서 실행할 수 있는 SQL로 번역합니다. 일반적으로 BASIS 커널은 Informix보다 5-6배 정도 많은 CPU 및 메모리 용량을 필요로 합니다. BASIS는 서로 다른 유형의 작업을 수행하여 사용자에게 필요한 결과를 제공하기 위한 일단의 "큐"와 버퍼링, 로크를?자체적으로보유하고 있습니다.

Informix에서 작업하는 것은 방대한 분량의 데이터 세트에 대한 다수의 조인(join), 집계(aggregation) 및 정렬(sort)과 관련되는 매우 복잡한 질의를 작성하는 일입니다. 사용되는 데이터베이스의 80% 이상을 차지하는 **psapbtab** 및 **psapstab**와 함께 SAP을 구성하는 표준 DB영역 세트가 있습니다. 200GB 이상의 데이터베이스를 가지고 있는 SAP 고객과 14,000개 이상의 테이블을 가지고 있는 시스템은 흔히 볼 수 있습니다.

SAP을 사용하지 않을 경우에는 다음의 내용에서 OSS에 대한 참조 사항을 보실 수 있습니다. 이는 SAP 고객이 환경 관리를 위해 참조할 수 있도록 정리한 내부 "라이브러리"입니다.

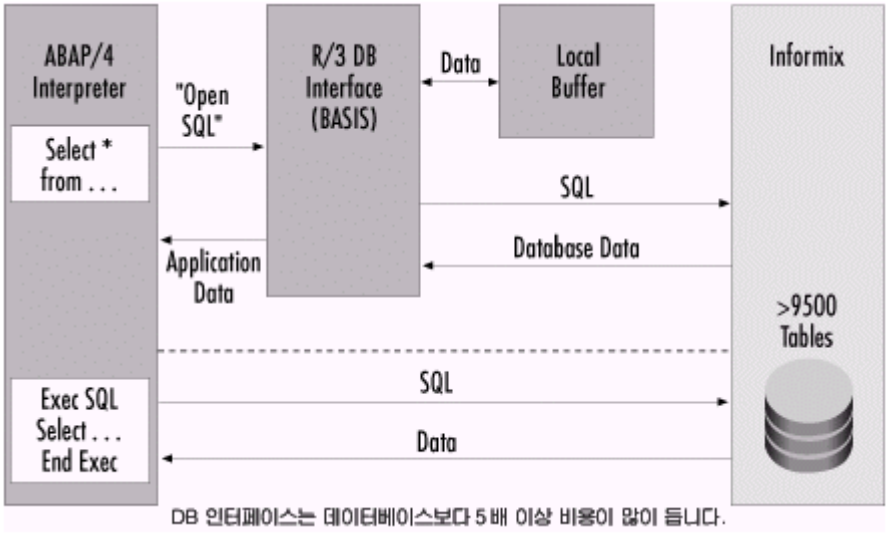

그림 2: SAP 아키텍처에 대한 소개

이러한 SAP 환경을 위해 $ONCONFIG 매개변수를 다음과 같이 설정할 것을 권장합니다.

- LRU_MAX_DIRTY <= 20
-

LRU 페이지 중 수정된 페이지를 백분율로 표현한 값으로, 수정된 페이지(MLRU)가 이 퍼센티지에 도달하면 수정된 페이지를 디스크로 반영하고 해당 페이지 버퍼를 "free(FLRU)" 상태로 되돌립니다. 즉, 수정된 페이지를 정리하는 시기를 결정하는 값입니다.

- LRU_MIN_DIRTY <= 10
-

역시 LRU 페이지 중 수정된 페이지를 백분율로 표현한 값으로, 수정된 페이지를 디스크로 반영하여 정리하는 작업을 종료하는 시기를 결정하는 값입니다. 즉, 수정된 페이지의 비율이 LRU_MAX_DIRTY에 이르면 수정된 페이지들을 디스크로 반영하여 정리하기 시작하며, 이렇게 정리하다가 수정된 페이지의 비율이 LRU_MIN_DIRTY에 이르면 페이지 정리를 더 이상 강제로 수행하지는 않는다는 의미입니다.

OLTP 환경에서는 LRU에서 수행해야 하는 작업의 양을 최소화하여 검사점(checkpoint) 수행 시간(checkpoint duration time)을 길지 않도록 유지해야 합니다. 이는 MAX와 MIN을 각각 20과 10 이하로 설정하여 수행할 수 있습니다.

- LTXHWM = 70
-

Long Transaction High Water Mark의 의미로서, 전체 논리로그 파일 중에서 사용한 논리로그의 백분율입니다. 즉, 논리로그를 70%만큼 사용했다면 IDS는 긴 트랜잭션을 확인합니다. 긴 트랜잭션이란 논리로그 사용율이 LTXHWM에 도달했을 때 가장 먼저 오픈돼서 그 시점까지 끝나지 않는 트랜잭션으로, 이 값에 도달하면 확인된 긴 트랜잭션은 롤백하기 시작하며, 새로 시작되는 트랜잭션은 받아들여지지 않습니다.

- LTXEHWM = 80
-

Long Transaction Exclusive High Water Mark의 의미로서, 전체 논리로그 파일 중에서 사용한 논리로그의 백분율입니다. 논리 로그 파일의 사용율이 이 값에 도달하면 현재 롤백되는 긴 트랜잭션에 논리 로그에 대한 배타적 액세스가 제공됩니다. 즉, 긴 트랜잭션을 안전하게 롤백할 수 있도록 다른 트랜잭션에 대하여 논리로그 사용을 제한합니다.

긴 트랜잭션이 발생하지 않도록 하려면 이 값을 상대적으로 높은 레벨, 즉 70이나 80으로 유지할 수도 있습니다. 즉, 너무 작은 값으로 설정하면 긴 트랜잭션이 자주 발생할 수 있습니다.

- USEOSTIME = 0
-

시간을 OS 의 시스템 호출로 설정("1")할 것인지 아니면 Informix 의 내부 타이머로 설정("0")할 것인지의 여부를 결정합니다. 질의에 의해 시간이나 날짜에 관련된 데이터베이스에 대한 요청이 이루어질 경우(TODAY 함수) Informix에서 이런 정보를 얻을 수 있는 방법은 두 가지입니다. Informix 내부 타이머를 이용하거나 운영 체제 시계를 이용하는 것입니다. 이 값을 "0"으로 설정하면 검색시 비용이 보다 적은 Informix 시계가 사용됩니다.

- LOCKS = 500,000
-

각각의 로크에는 44바이트의 상주 메모리(64비트 OS의 경우 72바이트)가 필요하기 때문에 메모리 사용을 줄여야 할 경우 이 값이 적절합니다. 일반적으로 SAP 고객은 500,000에서 4,000,000 사이의 값을 설정합니다. 클라이언트 복사(client copies)에 요구되는 로크가 있을 경우 보다 큰 값으로 설정해야 합니다. 이 값을 확인하려면 onstat -k | grep total을 사용하여 특정 시점에 사용된 로크의 수를 확인할 수 있습니다.

Informix/Waldorf에서는 이제 SAPDBA의 일부가 된 INFCFGCHECK라는 툴을 개발했습니다. 이 프로그램을 통해 사용자에게 친근한 방법으로 다수의 구성과 성능 확인을 신속하게 실행할 수 있습니다. 이 프로그램은 $ONCONFIG 파일, 디스크 레이아웃, 권한, DB영역 내의 여유 공간 및 전체적인 Informix 환경을 구성하는 기타 측면을 검토합니다.

이 툴에 대한 자세한 내용은 SAP에서 제공되는 다음의 내용을 참조하십시오.

- OSS 62340-INFCFGCHECK: "다운로드" 및 "첫 단계"
- OSS 71776-자동 데이터베이스 확인("infcfgcheck")

- OSS 64001-INFCFGCHECK: 자세한 메시지와 개별 확인

SAP에서 UPDATE STATISTICS를 실행할 경우 반드시 SAPDBA 도구 내에서 실행해야 합니다. 절대 DBACCESS에서 실행하지 마십시오. SAP을 사용하여 UPDATE STATISTICS를 실행하는 최선의 방법을 찾기 위해 상당한 시간과 분석이 소요되었으며 그 해결책을 코드화한 결과가 SAPDBA 도구입니다. 실제로 **dbaccess** 내에서 UPDATE STATISTICS를 실행하면 성능에 부정적인 영향을 미칠 수 있습니다.

SAP에서는 Informix의 테이블 분할(table partitioning) 사용을 완전히 지원합니다. 같은 테이블의 서로 다른 부분을 여러 DB영역에 배치함으로써, 동일한 레코드 세트를 찾는 데 필요한 드라이브에 대한 I/O의 양을 줄이고 이런 테이블을 동시에 백업/복원할 수 있습니다. 이는 또한 드라이브가 다운되었을 때 불량 DB영역만을 복원하는 데에도 적용됩니다. 마지막으로 SAP의 IS-Retail 모듈 내에는 스스로 32GB 이상으로 쉽게 커질 수 있는 트랜잭션 테이블이 있습니다. 테이블을 여러 DB영역에 분할하면 테이블 크기에 제한을 받지 않습니다.

이 옵션에 대한 자세한 내용은 다음을 참조하십시오.

- OSS 83183-분할화(fragmentation) 기초
- OSS 79070-테이블 분할: 단계별 절차
- OSS 81642-색인 분할/분리: R3의 특별한 행동

DB영역 전체에 걸쳐 테이블을 분할한다는 개념을 따르는 것은 특정 색인의 색인 페이지를 얻어 각각의 DB영역에 배치하는 능력입니다. 이는 데이터 페이지를 전혀 변경하지 않기 때문에 일반적으로 훨씬 더 쉽게 구현할 수 있으며, DBA 입장에서 볼 때에도 위험이 훨씬 적습니다.

색인을 분리하면 DB영역 내에서 테이블이 차지하는 공간의 양이 줄어들고, 테이블을 재구성해야 할 가능성이 줄어들며, 발생하는 I/I 경쟁이 감소됩니다. 한 DB영역에서는 색인 요청을 실행하는 동안, 데이터 페이지가 포함된 다른 DB영역에서는 BASIS 레이어에 레코드를 제공할 수 있습니다. 자세한 내용은 앞서 살펴본

OSS 81642와 OSS 79191-"분리된 색인" 생성을 참조하십시오.

병렬 처리(parallelism)는 Informix의 고도로 통합된 기능이기 때문에 병렬 처리를 이용하기 위해 SQL을 변경할 필요는 없습니다. 예를 들어 SAP의 SAPDBA 도구에는 "색인 생성" 옵션이 있습니다. 색인을 분리하는 경우에는 최대한 빨리 재생성하고자 할 것입니다. 이를 위해서는 OSS 85067에 요약된 Informix의 병렬 측면을 활용하면 됩니다. ONCONFIG 파일을 약간 변경하고 PDQPRIORITY 변수를 사용하여 병렬 처리 기능을 가동할 수 있습니다.

다음은 SAP 환경 튜닝과 관련하여 참조할만한 기타 OSS입니다.

- OSS 38307-공유 메모리 튜닝과 SAP
- OSS 29413-논리 로그 크기를 조정하는 단계별 방법
- OSS 41360-성능/튜닝/신뢰성을 위해 전체 $ONCONFIG 파일 검토
- OSS 50157-시스템의 다른 어떤 것도 변경하지 않고 Informix를 업그레이드할 수 있음

PeopleSoft 튜닝

PeopleSoft에서는 응용 프로그램 서버가 기본적인 기능을 수행합니다. 즉 전통적인 3계층 설정(3-tier setup)에서 응용 프로그램을 실행할 수 있습니다. 대량의 캐시나 작업 큐가 없으며, 사용자 요청을 채우는 Tuxedo 응용 프로그램 서버의 수정된 버전만 있을 뿐입니다.

사용자가 직접 데이터베이스와 통신을 할 수도 있으며 때로는 이것이 필요합니다. 이것이 가능하도록 하려면 클라이언트 PC에서 Informix Connect(I-Connect)를 구성해야 합니다. nVision 및 SQR과 같은 도구를 실행할 경우 특히 그러합니다. 다음은 I-Connect의 자세한 구성 설정을 나타낸 것입니다.

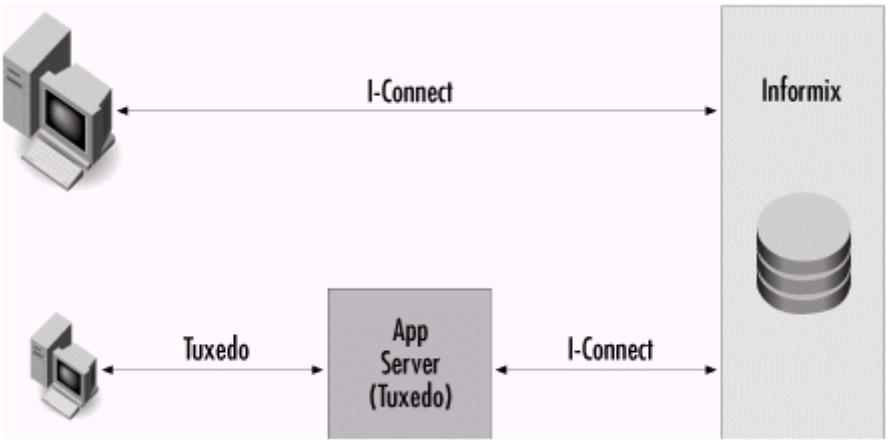

그림 3: PeopleSoft 아키텍처에 대한 소개

질의 그 자체는 SAP의 질의와 많이 유사하지만 질의 계획과 실행이 복잡합니다. 다수의 조인, 집계 및 정렬이 포함된 대규모의 질의입니다. Informix 7.30 이전에는 PeopleSoft에서 다수의 관련된 하위 질의(correlated subqueries)를 작성하여 성능 상의 문제를 일으키는 원인이 되었습니다. IDS 7.30부터는 이런 형식의 질의 내에서 질의 계획을 "단순화(flatten)"하기 위해 상당한 분량의 작업이 발생하여 처리 성능이 보다 빨라졌습니다.

데이터베이스의 분할과 관련하여 PeopleSoft가 설정한 표준은 없습니다. 따라서 자체적인 정밀 분석만이 분할 대상을 테스트하는 유일한 방법입니다. 그림 4는 참고할 만한 몇 가지 권장 사항입니다.

PeopleSoft 내에서 Informix 테이블 분할화 사용

- PeopleSoft는 분할화 및 Informix와 관련하여 특별한 표준이나 제한 사항이 없습니다.
- Informix 고객을 위한 필드에 사용된 예제

HR 테이블	분할 기준	이유
ps_earnings_bal	by period	12개월 롤링, 보고서 작성
ps_tax_balance	by period	12개월 롤링, 보고서 작성

ps_deduction_bal	by period	12 개월 롤링, 보고서 작성
ps_job	round robin	PDQ 로 구축한 색인
ps_earnings	round robin	병렬 백업 테이블
ps_deductions	round robin	병렬 백업 테이블
ps_tax	round robin	병렬 백업 테이블
ps_distribution	round robin	병렬 백업 테이블
Informix 전용: 결과: 성능/확장성		

그림 4: PeopleSoft 내에서 Informix 테이블 분할 사용

클라이언트 PC는 다른 어떤 ERP 응용 프로그램에서보다 PeopleSoft에서 데이터베이스와 훨씬 더 빈번하게 직접적으로 통신을 하기 때문에 클라이언트 PC(SetNet32)의 I-Connect 설정에 대한 구성을 검토할 필요가 있습니다.

다음은 검토해야 할 설정입니다.

- FET_BUF_SIZE

  페치(fetch) 버퍼가 클수록 네트워크를 통한 호출이 적습니다. 환경에 따라 FET_BUF_SIZE를 5000에서 20000 사이의 값으로 설정합니다.

- OPTOFC

  오버헤드 패킷 수를 줄입니다. 다수의 질의에 의해 소량의 데이터가 리턴될 때 더 큰 성능 향상 효과가 나타납니다. OPTOFC 환경 변수를 "1"로 설정하십시오.

- AUTOFREE

  SELECT 마지막 행이 페치(fetch)된 뒤 커서를 자동으로 닫아 네트워크 전체에서 호출이 감소됩니다. IFX_AUTOFREE 환경 변수를 "1"로 설정하십시오.

- OPTMSG

  BEGIN/COMMIT 작업 명령문을 번들 처리합니다. 서버에서 보내는 일부 COMMIT 메시지를 삭제합니다. OPTMSG 환경 변수를 "1"로 설정하십시오.

- DEFERRED PREPARE

  네트워크의 전체에서 호출을 감소시킵니다. IFX_DEFERRED_PREPARE 환경 변수를 "1"로 설정하십시오.

PeopleSoft에서 UPDATE STATISTICS를 실행하는 데 있어 PeopleSoft 자체에 의해 설정된 표준은 없습니다. 일단 다음을 사용하고 필요에 따라 수정하십시오.

- 각 테이블에 대해 UPDATE STATISTICS MEDIUM... DISTRIBUTIONS ONLY를 실행합니다.
- 각 색인의 첫번째 칼럼에 대해 UPDATE STATISTICS HIGH를 실행합니다.
- 복합 색인의 모든 칼럼에 대해 UPDATE STATISTICS LOW를 실행합니다.

  또한 필요할 경우 다음과 같이 하십시오.
- 색인이 뒤에 오지 않지만 등식 또는 부등식에 사용된 칼럼에 대해 UPDATE STATISTICS HIGH를 실행합니다.

이런 권장 사항은 Informix/PeopleSoft 고객 사이트와 Informix 내에서 찾은 "가장 실용적인 것"을 모아 놓은 것입니다. 국제 Informix 사용자 그룹 웹 사이트(http://www.iiug.org/)에 가면 이런 작업을 자동화할 수 있는 도구를 찾을 수 있습니다.

IDS 7.30 버전에서 중요하게 바뀐 것 중 하나는 옵티마이저에 있는 관련된 하위 질의(correlated subquery) 알고리즘입니다. 이것이 바뀜으로써 성능이 크게 개선되었습니다. 따라서 IDS 버전 7.24x 이하에서 실행할 경우 최신 7.3x 버전으로 업그레이드하는 것을 고려해 보십시오.

마지막으로 PeopleSoft에 영향을 주는 몇몇 $ONCONFIG 매개변수가 있습니다. 다음과 같은 것들이 포함됩니다.

- LRU_MAX_DIRTY <= 15
- LRU_MIN_DIRTY <= 10

- LOCKS = 500,000

SQR을 사용할 경우 값을 1,000,000 이상으로 높일 필요가 있습니다.

PeopleSoft의 보고용 도구인 SQR에서는 하나의 트랜잭션에 다수의 로크가 필요할 수 있습니다.

Baan 튜닝

Baan 환경에서 모든 사용자는 "Port Set" 또는 "DB Driver"라는 중간 계층(middle tier)을 거칩니다. 이 드라이버는 사용자 요청을 취하여 이를 완수하는 데 필요한 SQL을 생성합니다. Baan 환경에서 DBA의 가장 큰 어려움은 새로운 "부서" 또는 "회사"를 만들 때마다 약 6000개의 새 테이블이 생성되는 것입니다. Peoplesoft 또는 SAP 환경이라면 기존 테이블에 행이 추가되겠지만 Baan에서는 새 테이블이 생성됩니다. 그 결과 Baan 환경에서는 단일 데이터베이스 인스턴스에 40,000개 이상의 테이블이 있는 경우를 쉽게 볼 수 있습니다.

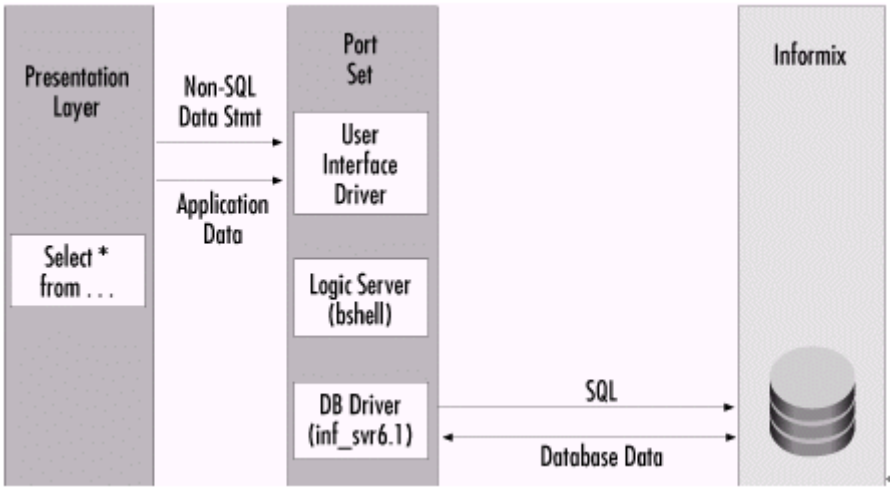

그림 5: Baan 아키텍처에 대한 소개

드라이버 세트는 Level I과 Level II 두 종류입니다. Level II 드라이버를 사용하려면 IDS 7.31을 실행해야 합니다.

다음은 이 두 드라이버의 차이입니다.

- Level I

  모든 조인과 정렬 작업이 드라이버에서 발생하기 때문에 데이터베이스로 보내지는 질의는 매우 간단합니다. 데이터베이스 테이블에는 "색인 스캔"만 수행하는 질의를 생성하기 위해 색인에 의해 사용되는 "해시" 칼럼이 있습니다.

- Level II

  데이터베이스로 보내지는 질의는 데이터베이스 내에서 조인 및 정렬 작업이 수행되도록 작성됩니다. "해시" 칼럼이 없으며 이런 칼럼에 의존하지 않고도 Informix에서 질의를 최적화할 수 있습니다.

다음은 Level I 드라이버일 경우의 $ONCONFIG 권장 사항 목록입니다

- BUFFERS = 시스템 메모리의 10%
- LOCKS = 50,000(사용자 수에 따라 다를 수 있음)
- LRU_MIN_DIRTY = 8
- LRU_MAX_DIRTY = 12
- SHMVIRTSIZE = 4MB * 사용자 수
- DD_HASHSIZE = 211 # 해시 테이블의 항목 수
- DD_HASHMAX = 45 # 항목 목록 하나의 최대 길이

onstat -g dic을 사용하여 이 해시 값들을 검토하십시오.

질의가 단순하기 때문에 LOCKS와 BUFFERS는 최소 값만 필요합니다.

Baan 환경의 또 다른 특성은 Baan에서 데이터베이스에 대한 세션을 생성하고 사용하는 방식으로 인해 사용자 한 사람이 6개 이상의 데이터베이스 세션을 만들 수 있다는 것입니다. 그 결과 예를 들어 100명의 활성 사용자가 데이터베이스에 대해 600개의 세션을 만들 수 있습니다. 폴링 오버헤드 때문에 생기는 이 문제를 해결하기 위해 IDS 7.23.UC4 버전부터 "Multiplexed Connections"가 추가되었습니다. Multiplexed Connections에서는 데이터베이스 서버와 클라이언트 간에 단일 네트워크 연결을 사용하여 클라이언트로부

터의 다중 데이터베이스 연결을 처리합니다.

Multiplexed Connections를 이행하려면 다음을 설정하십시오.

- NETTYPE에 대한 $ONCONFIG 값 = SQLMUX
- $Informix/etc/sqlhosts에 대해 데이터베이스 서버 연결에 m=1 값을 추가해야 합니다(Informix 7.3 Admin Guide, pp. 4-7).

다음은 Baan 환경 내에서 고려해야 할 기타 사항입니다.

- 큰 트랜잭션 테이블 분할
- 색인은 절대 분리하지 말 것
- 매주 UPDATE STATISTICS 실행
- 회사별로 하나씩 데이터베이스 생성
  - 각각 자체 DB영역에서 생성
  - 결과: 다량의 테이블 및 색인
- 시스템 카탈로그가 LARGER가 되도록 "Next Extent" 수정
  - 그렇지 않을 경우 sysmaster 데이터베이스가 분할될 것임
  - 테이블과 색인 수가 많으면 sysmaster가 커짐

마지막으로 Baan에서는 고객이 예상되는 수의 사용자를 처리하기 위해 필요한 하드웨어 플랫폼의 규모를 결정하는 데 도움이 되는 내부 하드웨어 크기 조정 벤치마킹 결과에 대한 요약문이 실린 "Baan Sizing Guide"를 발행하고 있는데, 이 안내서가 하드웨어 플랫폼 결정에 도움이 될 것입니다.

Lawson 튜닝

Lawson 환경은 Baan의 Level I 드라이버와 매우 유사합니다. 이 환경에는 COBOL과 C 프로그램을 함께 사용하여 정보에 대한 사용자 요청을 처리하는 중간 계층(middle tier)이 있습니다. 그 결과가 "ifxdb7"라는

이름으로 Informix와 통신하는 데이터베이스 인터페이스입니다. 이 "C" 프로그램은 Lawson과 Informix가 협력하여 데이터베이스 레벨의 향상을 통해 더 큰 효율성을 얻을 수 있도록 지속적으로 개선하고 있는 소프트웨어입니다.

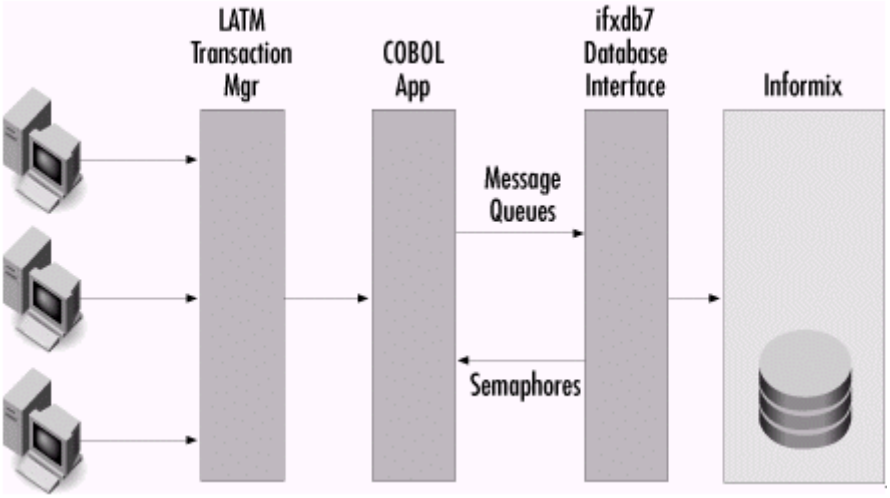

그림 6: Lawson 아키텍처

데이터베이스에 보내지는 질의는 매우 단순하며 항상 데이터베이스 내에서 생성된 색인을 활용하도록 작성됩니다. Lawson 시스템 내의 평균 테이블에는 5개 이상의 색인이 있으며, 기본 또는 외부 제한 조건이 없고, 저장된 프로시저나 트리거가 없으며, CHAR, SMALLINT, INTEGER, DECIMAL 및 DATE만 포함되는 데이터 형식이 있습니다. 마지막으로 모든 조인과 정렬은 "ifxdb7" 드라이버에서 처리합니다. 따라서 비용이 많이 드는 대부분의 프로세스는 데이터베이스가 아니라 모두 드라이버에서 처리하도록 만듭니다.

Lawson에서는 세 벤더들에 맞게 성능 관련 정보나 튜닝 정보를 특별히 제공하지는 않습니다. 다음은 Lawson의 권장 사항입니다.

- 항상 Lawson 웹 사이트에서 "ifxdb7" 드라이버의 최신 "버전"을 다운받으십시오.
- 정확한 색인이 있는지 확인하십시오. 성능 문제가 생길 수 있으므로 순차적 스캔은 절대 사용되지 않도록 하십시오.

  - Lawson에서 제공하는 "verifyifx7" 툴을 사용하여 색인 일관성을 확인할 수 있습니다.

- RAID 5를 사용하지 말고 RAID 0 또는 1만을 사용하십시오. RAID 5는 I/O 오버헤드가 지나치게 많습니다.

- 서로 다른 DB영역과 물리적 디스크에 대해 별도의 로그, 데이터 및 색인을 사용하십시오.

  - 모든 색인을 자체 DB영역으로 분리하십시오

- n sysmaster:sysptprof SMI 테이블을 사용하여 액세스 수가 많은 테이블을 찾아서 이 테이블을 별도의 DB영역으로 이동하십시오.

  - sysmaster:sysptprof에서 tabname, isreads, iswrites, bufreads 및 bufwrites를 선택하여 isreads 설명에 따라 순서를 정하십시오.

- 시스템 메모리의 1/3 정도를 Informix용으로 보유해 두십시오.

마지막으로 반드시 최신 데이터베이스 드라이버와 7.30.UC1 이상의 IDS에서 실행하도록 하십시오. 이전 버전을 사용할 경우 UPDATE STATISTICS를 실행하지 마십시오. 7.30에서부터 데이터베이스 드라이버 내에 모든 질의에서 색인을 사용할 수 있도록 지원하는 옵티마이저 명령이 제공됩니다. 이전 버전에서는 옵티마이저에서 가끔씩 순차적 스캔을 선택하였지만, 이는 Lawson 환경에서 최적이 아닙니다.

모든 ERP 응용 프로그램에 대한 일반 튜닝

모든 ERP 응용 프로그램의 튜닝에서 지켜야 할 일관된 고려 사항이 있습니다. $ONCONFIG 파일에서 다음의 값에 주의하십시오.

SHMVIRTSIZE - 최대한 크게 설정

- 이유:

  - 새 공유 메모리 세그먼트를 동적으로 할당하면 시간이 많이 소비됩니다.

  - 사용할 수 있는 공유 메모리 세그먼트에 대해 여러 운영 체제에서 제한이 있습니다.

  - 2개 이상의 공유 메모리 세그먼트를 사용할 경우 여러 운영 체제에서 성능이 크게 떨어집니다.

- onstat -g seg를 실행하여 둘 이상의 "V"(가상 메모리 공간)가 할당되었는지 여부를 확인합니다.

NUMAIOVPS-Informix 실행 Asynchronous I/O(AIO) 프로세스의 수

- 기본값은 NULL(청크 수x 2) 또는 6이며 보다 큰 수가 만들어집니다.
- Kernel AIO(KAIO)를 사용할 경우 1로 설정합니다. 운영 체제 로그 파일에 기록하기 위해 Informix 에는 여전히 그 AIO VP 하나가 필요합니다.
- Informix의 AIO를 사용할 경우 (2 * 하드 드라이브 수)로 설정합니다. 이 값은 드라이브 처리를 위한 "최적"의 I/O 스레드 수를 제공합니다.
- **onstat -g iof**로 KAIO가 작동 상태인지, **onstat -g iov**로 AIO가 사용 가능하게 설정되었는지 검토하십시오.

CLEANERS-페이지 클리너 스레드

- 하드 드라이브가 20개 이하일 경우 디스크 당 하나의 클리너
- 하드 드라이브가 20에서 100개 사이일 경우 드라이브 2개당 하나의 클리너
- 하드 드라이브가 100개 이상일 경우 드라이브 4개당 하나의 클리너
- onstat -F로 클리너를 검토하십시오.

  - Foreground >0이면 CLEANERS를 높이고 LRU_MAX_DIRTY를 줄이십시오.

OPTCOMPIND = 0

- Informix 기본값은 2입니다.
- "0"으로 설정하면 사용량이 많은 OLTP 환경에서 최적의 조인 알고리즘이 제공됩니다.

LRU

- NUMCPUVPS <= 4이면 LRU = 4입니다.
- NUMCPUVPS > 4이면 LRU = NUMCPUVPS입니다.

ERP 시스템을 설정할 때 일반적으로 범하기 쉬운 한 가지 실수는 논리 로그에 관련하여 기본 $ONCONFIG 파일에서 제공하는 디폴트 값을 사용한다는 것입니다. 그럴 경우 로그 백업이 빈번해지고 긴 트랜잭션이 발생할 가능성이 많습니다. 이를 피하려면 다음의 주의 사항을 참고하십시오.

- 기본값인 LOGFILES = 6과 LOGSIZE = 500을 사용하지 마십시오. 이 값을 사용하면 긴 트랜잭션이 발생하고 검사점이 빈번해지는 등의 문제가 생길 수 있습니다.
- 각각 >= 10,000 KB인 큰 로그를 여러 개 만드십시오. Sears/PeopleSoft 구현일 경우 1.9GB의 논리 로그 공간이 있습니다.
- 로그를 별도의 DB영역과 디스크에 두십시오.

이런 모든 응용 프로그램에서는 Informix용으로 원시 디스크(raw disk)를 권장하고 지원합니다. 다음은 초기 시스템을 설정하고 새 청크를 추가할 때 일반적으로 범하기 쉬운 세 가지 실수입니다.

- 원시 디바이스(raw device)를 사용할 때 오프셋을 주지 않은 경우

  이유:

  - 항상 원시 디바이스를 사용할 때는 일정한 오프셋을 두면 그 오프셋 만큼은 사용하지 않으므로, 디스크 시작 부분에 있을 수 있는 디스크 제어 정보 위에 덮어쓰는 것을 방지합니다.
  - SAP 설치 스크립트 내에서 모든 초기 오프셋의 표준은 Offset = 8입니다.

- 원시 디바이스에 대해 상징적 링크(symbolic link)를 사용하지 않은 경우

  이유:

  - 디스크 등의 하드웨어 문제가 생길 경우, 상징적 링크를 사용하면 Informix에서는 원시 장치명이 아니라 링크를 사용하여 복구할 수 있습니다.
  - 시스템 이중화가 필요할 경우 DB영역에 해당하는 디스크에 대한 일관성 있는 이름 지정 규칙이 있어야 합니다.

  방법:

  - 데이터 링크를 위한 디렉토리를 만듭니다.
  - 이 디렉토리의 /dev 아래에 있는 원시 디바이스에 대한 상징적 링크를 만듭니다.

- 원시 디바이스에 적합한 모든 Informix 권한(660 mode, Informix owner, Informix group)이 있는지 확인합니다.

- DB영역을 미러링하지 않은 경우

  이유:

  - 사용 가능 시간(uptime)을 최대화하는 통합 미러링
  - 읽기 분할

  다음에 대해 Informix 미러링을 사용하는 것이 좋습니다.

  - rootdbs
    * 가장 중요한 DB영역
    * 미러링으로 디스크 장애시 아카이브 테이프로부터 복구(cold restore)를 해야하는 경우를 최대한 방지해야 합니다.
  - 물리 로그/논리 로그 DB영역
    * 디스크 장애로 트랜잭션 완료를 못하는 상황을 미러링으로 피할 수 있습니다.

마지막으로 어떤 응용 프로그램에서건 가장 튜닝하기 편한 것은 가장 빈번히 사용되는 디스크/청크를 확인하고 I/O 작업을 확산하는 것입니다. onstat 출력을 검토함으로써 문제가 되는 청크를 찾아내고 문제가 되는 테이블을 여러 DB영역에 걸쳐 저장하여 보다 균형잡힌 I/O 계획을 얻을 수 있습니다.

맺음말

앞에서 우리는 Informix의 SAP, PeopleSoft, Lawson 및 Baan 고객에게 옵션을 변경하여 튜닝하는 방법과 주의 사항을 살펴보았습니다. 그러나 이 글은 위의 응용 프로그램 이외의 제품을 사용하는 분들에게도 도움이 될 수 있습니다. 위의 응용 프로그램들은 관계형 데이터베이스에서 기술적으로 지원하기에 가장 까다로운 환경을 예로 든 것입니다. 따라서 이런 ERP 업체에 도움이 된다면 다른 OLTP 환경에도 분명히 도움이 될 것입니다.

요약해 보면, 위에서 설명한 응용 프로그램을 최적화하고 튜닝하기 위해서 다음 내용을 살펴 보았습니다.

- $ONCONFIG 파일 설정
- Informix 분할(fragmentation)로 데이터와 색인 분할(partitioning)
- Informix 병렬 처리로 병렬 색인 생성
- UPDATE STATISTICS-올바른 또는 잘못된 실행 방법
- ERP 업체에서 만든 툴에 대한 소개

지면의 제한 때문에 다음 내용은 다루지 않았습니다.

- ON-Tape 및 ON-Bar를 이용한 백업/복구
- NT 전용 튜닝 매개변수
- 기타 작업의 병렬 처리(배치 프로그램, 큰 질의)
- 성능 문제의 원인/효과를 추적하는데 도움이 되는 SMI 질의

웹에서 이용할 수 있는 추가 리소스:

http://www.Informix.com/answers - Adobe 파일 형식으로 된 모든 Informix 문서를 무료로 이용할 수 있습니다.

http://www.iiug.org/ - 국제 Informix 사용자 그룹 웹 사이트입니다.

http://www.Informix.com/sap - Informix에서 SAP을 실행하는 것과 관련된 최신 정보만을 다루는 웹 사이트입니다.