Tech Notes(vol.3)

Microsoft Windows NT 환경에서 Informix 제품사용

소개

Microsoft Windows NT(이하 Windows NT라 칭함) 환경은 UNIX 환경과 상당한 차이점이 있습니다. Informix 제품을 설치하기에 앞서 Windows NT 환경에 익숙해지고, Informix OnLine Dynamic Server, Informix OnLine Workgroup Server, Informix ESQL/C 등을 설치하는데 따른 모든 문제에 대하여 살펴보는 것이 중요합니다. 여기서는 Windows NT 환경과 UNIX 환경의 차이점에 대해 알아보고, Informix 제품을 설치할 때에 주의할 점들을 살펴보고자 합니다.

관리자

Windows NT 설치 프로그램은 설치시 Administrator, Guest, Initial User의 세 가지 기본 사용자 계정을 만들어 각 계정에 대하여 적절한 권한을 부여합니다. UNIX와는 달리 NT에는 단일 수퍼사용자 계정이 없으며 그룹을 통하여 권한이 부여됩니다. Windows NT에는 기본적으로 Administrators, Power Users, Guests, Everyone 그룹이 포함되어 있습니다.

Windows NT의 관리자 계정은 이름을 변경할 수 있습니다. Informix OnLine Dynamic Server 또는 Informix OnLine Workgroup Server 제품의 설치는 반드시 관리자 권한에서 실행해야 합니다.

참고

NT의 관리자 권한은 UNIX의 root 계정에 해당합니다. Informix 서버 제품들을 설치하려면 관리자 권한이 있어야 합니다.

Windows NT 파일시스템

Windows NT가 지원하는 파일 시스템 형식은 FAT, HPFS, NTFS 세 가지입니다.

- FAT 파일 시스템: FAT 파일 시스템은 원래 MS-DOS 파일 시스템입니다. 이 파일 시스템을 사용해도 Windows NT에서 긴 파일 이름을 사용할 수 있습니다. FAT 파티션의 주요 특징은 다음과 같

습니다:

1. 파일 시스템 오버헤드가 가장 적습니다.
2. 200 MB 이하 크기의 파티션에서 최적의 성능을 발휘합니다.
3. Windows NT의 파일 시스템 보호 기능인 지역 파일, 디렉토리 보안 기능을 사용할 수 없습니다.
4. DOS와 Windows NT 다중 부팅을 위해서는 필수적입니다.

단일 기계에서 DOS와 NT를 동시에 사용하려면 최소한 하나 이상의 FAT 파티션이 필요하다는 것을 기억하십시오.

- HPFS: HPFS는 OS/2가 지원하는 파일 시스템과 동일한 파일 시스템입니다. Windows NT 3.51 버전에서는 HPFS 파티션을 생성할 수 없습니다.

- NTFS: NTFS는 Windows NT에 가장 적합한 파일 시스템입니다. NTFS에는 다음과 같은 특징이 있습니다.
    1. 복구 가능한 파일 시스템 - NTFS에는 모든 디렉토리와 파일 갱신사항을 자동으로 로그하는 트랜잭션 로깅 기능이 있습니다.
    2. 파일 단위, 디렉토리 단위 압축 기능이 있습니다.
    3. POSIX 규격을 지원합니다.
    4. Windows NT 보안 기능을 이용할 수 있습니다.

NTFS 파티션의 최소 권장 크기는 50 MB입니다. Microsoft에 따르면, NTFS는 200 MB 이하의 크기에서는 비효율적이고, 400 MB 이상의 볼륨에서 최적의 효율을 발휘한다고 합니다.

Informix 서버 제품들은 반드시 NTFS 파일 시스템에 설치되어야 한다는 것을 기억하십시오.

Informix 서버는 NTFS를 필요로 하므로, Informix 서버를 사용하기 위해서는 기존의 FAT 파티션을 NTFS로 변환해야 할 필요가 있을 것입니다. 이런 경우, Windows NT의 CONVERT 유틸리티를 사용하십시오. 이 변환 유틸리티를 사용해도 파티션 내용이 파괴되지는 않지만, 변환 후에 다시 역변환은 불가능합니다. CONVERT 유틸리티의 명령행 구문은 다음과 같습니다.

convert 드라이브명: /fs:ntfs

CONVERT.EXE는 변환 디스크에 대해 배타적인 접근을 할 수 없는 경우(변환 디스크가 시스템 디스크(C:)인 경우) 오류 메시지를 내고, 시스템 재시동 후 자동으로 다시 시도하는 기능을 옵션으로 가지고 있습니다. 일단 FAT 또는 HPFS를 NTFS로 변환하면, 역변환은 불가능하다는 것에 주의하십시오.

Windows NT 루트 디렉토리

여기서는 Windows NT 루트 디렉토리의 서브디렉토리에 위치하는 시스템 파일에 대해 언급하고 있습니다. Windows NT 하나만 설치된 경우, 루트 디렉토리는 ₩WINNT35가 됩니다. Windows NT와 Microsoft Windows 3.x이 설치된 경우, 루트 디렉토리는 ₩WINDOWS가 됩니다.

Windows NT 레지스트리

Windows NT 레지스트리는 autoexec.bat, config.sys와 같은 DOS 시스템 파일과 system.ini, win.ini, reg.dat 등과 같은 Microsoft Windows 3.1 버전의 공유파일을 대체합니다. 레지스트리의 각 키는 .INI 파일의 괄호로 둘러싼 표제어와 유사한 것입니다.

Windows NT의 각 구성요소는 레지스트리를 이용하여 구성 데이터 저장, 시동 정보 획득, 장치 제어기 구성 매개변수 전달 및 획득, 신규 구성 데이터 저장을 수행합니다.

NT 레지스트리는 계층적인 구조로 이루어져 있습니다. 레지스트리에는 HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_USER, HKEY_CLASSES_ROOT 등 네 개의 하위 구조가 있습니다.

HKEY_LOCAL_MACHINE에는 모든 구성 정보가 저장됩니다. Informix 서버는 HKEY_LOCAL_MACHINE 부분에 구성 데이터, sqlhosts 정보, InformixDIR 및 InformixSERVER 등의 정보를 저장합니다.

HKEY_USERS에는 기본 시스템 설정과 현재 로그한 사용자의 구성 설정을 저장하는 두 개의 하위 키가 있습니다. HKEY_CURRENT_USER에 현재 로그한 사용자의 정보가 저장됩니다.

HKEY_CURRENT_USER 키의 정보와 HKEY_LOCAL_MACHINE의 정보가 중복될 경우, HKEY_CURRENT_USER에 우선권이 있습니다. Informix 클라이언트는 UNIX 상의 sqlhosts, InformixDIR, InformixSERVER, 기타 다른 환경변수들과 동등한 정보를 HKEY_CURRENT_USER에 저장합니다.

HKEY_CLASSES_ROOT에는 소프트웨어 구성 정보가 저장됩니다. 이 영역은 OLE와 파일 연관정보를 저장하는 곳입니다. HKEY_CLASSES_ROOT의 목적은 Microsoft Windows 3.1 버전의 등록정보 데이터베이스와 호환성을 유지하는 것입니다.

참고 레지스트리를 함부로 변경하면 시스템에 심각한 오류가 발생할 수 있기 때문에 아이콘으로 제공되지 않습니다. 그러나 프로그램 그룹에 포함시킬 수는 있습니다. 이 유틸리티는 Windows NT 루트 디렉토리 아래의 다음 디렉토리에 있습니다.

₩₩SYSTEM32₩REGEDT32.EXE

시스템 구성은 일반적으로 제어판 또는 Windows NT 설치 유틸리티를 통해서 변경되어야 한다는 것을 주의하십시오.

Informix 서버

Informix 서버는 서비스로서 설치되고, Informix라는 사용자로 시동됩니다. 만약 Informix 서버를 시작하는데 문제가 있다면, 다음 사항을 확인하십시오.

- Informix 서버 서비스는 Informix 사용자로 기동시켜야 합니다.
- Windows NT에 Informix로 로그인할 수 있어야 합니다.

Informix 암호가 변경되거나 유효기간이 만료되었다면 Informix 서버 서비스를 시작할 수 없습니다. Informix 서버를 시동하려면, 서비스 창에서 Informix 서버를 선택하여 시작시킵니다. 시스템 시작시 서비스가 시작되도록 설정할 수도 있습니다.

Informix 서버 서비스는 반드시 Informix 사용자만이 시동시킬 수 있다는 것에 주의하십시오.

다른 서버 문제에 대해서는 Informix 홈 디렉토리에 위치하는 ONLINE.LOG 파일을 참고하십시오. 만약 ONLINE.LOG 파일이 생성되지 않았다면, 관리자 도구 창의 이벤트 로그에서 Informix 서비스가 시동되지 않는 원인에 대한 정보를 얻을 수 있을 것입니다.

Informix 서버는 클라이언트와 TCP/IP를 통해 정보를 교환합니다. 기본 TCP/IP 서비스는 포트 번호 1526의 turbo입니다. 모든 서비스에 대한 정의는 다음 파일에 기록되어 있습니다.

₩₩system32₩drivers₩etc₩services

'oninit initialization failure: access violation': 접근 위반이라는 오류가 발생할 수 있습니다. 이 오류가 발생하는 이유는 NT 기계의 명칭이 DNS에서 인식되는 명칭과 다르기 때문입니다. 이 문제를 해결하려면, NT 기계의 명칭을 바꿔야 합니다. 그런 다음 Informix 서버를 재설치하거나, Informix 레지스트리가 참조하는 명칭을 새로운 이름으로 변경해야 합니다.

Informix ESQL/C의 경우 오류가 발생할 수 있으므로, 호스트 명에 하이픈 ("-")을 사용하지 않는 것이 좋습니다. 호스트 명에 하이픈을 사용하면, 서버가 정상적으로 설치되고 dbaccess가 정상적으로 작동하더라도 서버 연결에 실패할 수 있습니다.

Informix ESQL/C, 버전 7.10, 클라이언트

Informix 서버에 접근하려면, 최소한 Informix ESQL/C의 실행 프로그램만이라도 설치해야 합니다. Informix ESQL/C는 Informix 서버 소프트웨어 디렉토리가 아닌 다른 디렉토리에 설치하십시오. Informix ESQL/C 설치 프로그램은 기본적으로 ₩WIN32APP₩Informix 디렉토리에 프로그램을 설치합니다.

Informix ESQL/C는 레지스트리와 환경변수를 모두 사용합니다. 설치과정 중, Informix ESQL/C 실행에 필수적인 DLL 파일이 위치한 디렉토리가 사용자 환경변수인 PATH에 포함되도록, PATH 변수를 수정합니다. 환경변수는 시스템의 제어판에 있습니다.

Informix 서버와 통신할 때 필요한 정보는 레지스트리의 HKEY_CURRENT_USER 영역에 있습니다. 이것은 현재 사용자, 또는 현재 사용자가 실행한 프로세스만이 이 정보에 접근할 수 있다는 것을 의미입니다. 다른 사용자 또는 의사 SYSTEM 사용자로서 실행한 프로그램은 Informix 환경에 대한 정보를 얻을 수 없습니다. 그 프로그램은 Informix 파일에 올바로 접근하지 못하고, 오류를 발생시킬 것입니다. 다음은 전형적인 오류 메시지입니다.

-461 File open error, cannot open sgl.iem

UNIX의 InformixDIR와 동일한 환경변수가 응용프로그램 사용자를 위해 설정되지 않았기 때문에 파일 개방에 오류가 발생합니다. 이 값은 현재 사용자 레지스트리 하위 트리에 있습니다. 이런 문제는 Informix ESQL/C 클라이언트가 서비스로서 실행되는 환경에서 발생합니다. 기본으로, 이 서비스는 SYSTEM 유저로 시동됩니다.

레지스트리에는 InformixDIR, InformixSERVER, 그리고 DBNLS와 LANG과 같은 NLS 정보, UNIX의 sqlhosts 파일 내용과 유사한 서버 정보 등, Informix 환경에 대한 정보가 저장됩니다. 이 Informix 환경 정보를 설정하거나 갱신하려면, Informix에서 제공하는 SetNet 유틸리티를 사용하십시오. SetNet은 SetNet을 실행한 사용자만이 접근할 수 있는 레지스트리의 HKEY_CURRENT_USER 하위 트리를 갱신합니다.

만약 Informix ESQL/C 클라이언트가 Informix 서버가 실행되는 기계가 아닌 다른 기계에서 실행된다면, 그 시스템의 ₩₩SYSTEM32₩DRIVERS₩ETC₩SERVICES 내용을 갱신해야 합니다.

SetNet 유틸리티에는 프로토콜에 대한 항목이 있습니다. 보통, 드롭다운 콤비네이션 상자에 나타나는 첫 번째 옵션을 선택하면 됩니다. 만약 여기에 아무 내용이 없다면, onsoctcp를 입력하십시오.

서비스로서의 Informix ESQL/C 클라이언트

특히 클라이언트가 서비스로서 실행되는 경우에는 SetNet 프로그램의 범위를 이해하는 것이 중요합니다. 이

와 유사한 가장 잘 알려진 경우가 LiveWire PRO를 실행하는 Netscape Server입니다.

기본적으로, Windows NT 서비스는 의사 SYSTEM 사용자로 실행됩니다. SetNet을 USER1로서 실행하려면, 서비스의 시동 매개변수를 수정해 서비스가 USER1로 시작되도록 해야 합니다. 클라이언트 서비스를 시작할 때는 적절한 암호를 입력해야 하며, 암호가 변경되거나 유효기간이 만료된 경우에는 서비스를 시작할 수 없습니다.

성능조정

Microsoft Windows NT에서는 성능 조정에 대한 사용자의 역할이 많이 줄어들었습니다. 하지만 주로 디스크 I/O 분야에 사용자가 설정할 항목이 몇 가지 있습니다. Windows NT는 가상 운영 스킴을 사용합니다. 그림 1을 참고하십시오.

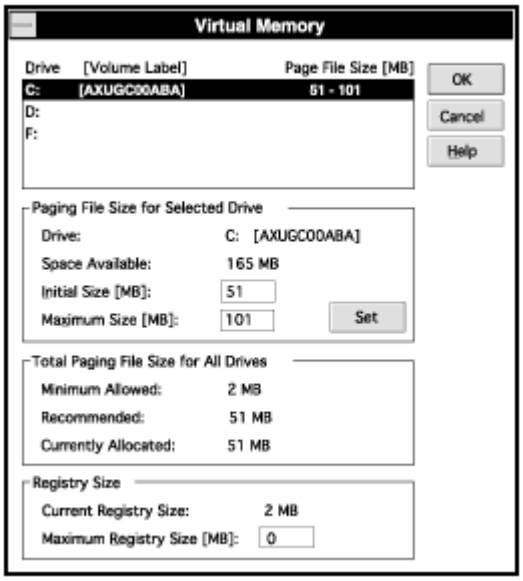


그림 1: 가상 메모리 페이징 파일 정보

Windows NT는 사용 가능한 디스크 공간이 가장 많은 파티션에 PAGEFILE.SYS라는 이름의 가상 메모리 페이징 파일을 생성합니다. 보통 이 파일의 크기는 10 MB 이상입니다.

만약 시스템에 복수의 하드 디스크가 장착되어 있다면, 각각의 하드 디스크에 페이징 파일을 만들 수 있습니다. 하드 디스크 콘트롤러가 동시에 두 하드 디스크에 읽고 쓸 수 있는 기능을 가지고 있다면, 복수의 페이징 파일에 정보를 분산시켜 I/O 성능을 향상시킬 수 있습니다.

페이징 파일을 Windows NT 디렉토리가 위치한 드라이브가 아닌 다른 드라이브로 옮기면 성능이 향상됩니다. 하지만, 페이징 파일을 시동 파티션이 아닌 다른 장소에 위치시키면 복구가 까다로와집니다.

시스템 반응 속도를 높이는 또 다른 방법은 페이징 파일의 초기 크기를 시스템이 필요로 하는 최적의 크기로 만드는 것입니다. 이렇게 함으로서 시스템이 페이징 파일을 최적 크기로 증가시키는데 사용하는 시간을 없앨 수 있습니다. 최적의 페이징 파일 크기는 꼭 정해진 것이 아니며, 상당 기간 동안 시스템을 감시함으로써 구할 수 있을 것입니다.

Windows NT 파일 시스템인 NTFS에는 파일 압축 기능이 있습니다. 단일 파일, 여러 개의 파일, 디렉토리, 서브디렉토리를 압축 및 압축 해제할 수 있습니다. 이것은 CPU 처리 싸이클을 희생해서 I/O 성능을 높이는 것입니다. 압축해제는 파일 전체에 대해 수행되므로 데이터베이스 파일에 압축 기능을 사용하는 것은 좋지 않습니다.

시스템 모니터

Windows NT에는 성능 모니터 도구가 있습니다. 이 도구는 일정 기간 동안의 성능 변화를 보여주고, 병목 현상 발생지점을 알려주며, 시스템 구성 변경이 미치는 영향을 감시하고, 시스템 처리능력을 판단합니다.

성능 모니터에서 시스템 구성요소는 개체로 표시됩니다. 성능 모니터는 개체의 특정 기능을 선택해서 그 성능을 지속적으로 추적할 수 있습니다. 이 기능을 카운터라고 합니다.

시동시에 커널에서 운영하는 일부 카운터는 꺼집니다. 예를 들어, 커널은 기본으로 디스크 성능 카운터를 운영하지 않습니다. 이런 카운터들을 사용하면 시스템 성능이 대략 2% 정도 떨어지게 됩니다.

디스크 성능 카운터를 작동시키려면, Windows NT 명령 프롬프트에서 다음 명령을 사용하십시오.

diskperf -y

같은 방법으로, 디스크 성능 카운터를 중지하려면, Windows NT 명령 프롬프트에서 다음 명령을 사용하십시오.

diskperf -n

성능 모니터는 대략 25개의 서로 다른 개체 형을 추적합니다. 이 개체의 형 목록은 다음 표에 나와 있습니다.

Monitor Object	Number of Counters

Server	26
Server Work	17
Queues System	20
TCP	9
Thread	12
UDP	5

표 1: 성능 모니터 목록

이 개체는 UNIX의 sar 명령과 유사한 성능 감시 기능을 제공합니다. 또한, 프로세스 객체에서 특정 프로세스를 감시하도록 설정할 수도 있습니다. ONINIT 프로세스를 선택하거나, 하나 또는 여러 개의 ONINIT 스레드를 선택하면 데이터베이스 서버에 어떤 일이 발생하는지 자세히 알 수 있습니다. ONINIT 프로세스 내의 특정 스레드 감시도 가능합니다.

감시하는 카운터는 프로세스 수준과 스레드 수준 모두 가능합니다. 이 카운터에는 프로세서 시간 %, 사용자 시간 %, 초당 페이지 오류, 초당 문맥 전환, 스레드 상태, 스레드 대기 이유가 포함됩니다.

결론

Informix 서버에 있어 Windows NT는 새로운 환경입니다. Windows NT의 그래픽 사용자 인터페이스 (GUI)가 새로운 환경에 대한 복잡성을 감추었지만, 또다른 복잡성을 추가하기도 하였습니다. Informix 제품을 성공적으로 설치하고 운용하려면 Windows NT 구성에 대한 지식을 많이 습득하는 것이 좋습니다.

참고문헌

Microsoft Windows NT환경에 대해 보다 자세히 알고 싶다면 다음 서적들을 참고하십시오.

Supporting Microsoft Windows NT 3.51 Student Workbook, part number 5946B, release 07/95.

Informix OnLine Dynamic Server, Administrator's Supplement for Windows NT, 07/95.

Informix ESQL/C, Programmer's Supplement for Win32 Version 7.1, part number 000-7711, 06/95.

Informix Termcap 소개

서론

이 글은 독자들이 Informix에서 termcap(터미널 성능)을 사용하는 방법과 개념 및 용어에 익숙해지도록 하기 위한 것이며, 모든 Informix 제품에 적용됩니다.

Termcap이란?

termcap은 원래 BSD UNIX(TM)를 위해서 개발된 것으로, 터미널 성능을 내장하여 특별한 터미널 제어기가 없어도 다양한 ASCII 터미널을 사용할 수 있도록 한 것입니다. 기록되는 각각의 터미널 성능은 터미널의 특성 혹은 기능을 나타냅니다. termcap의 항목은 일련의 터미널 키를 응용프로그램 형식으로 변환합니다. 또한, 이 항목들에는 응용프로그램 출력을 터미널에 따른 화면 출력 정보로 변환하기 위한 템플릿이 있습니다.

대다수의 UNIX 시스템들은 termcap 파일을 제공하며, 이 파일은 보통 /etc 디렉토리에 위치합니다. UNIX System V에서는 termcap과 terminfo 데이터베이스 두 가지 모두 제공됩니다.

Informix의 모든 제품에는 termcap 파일이 포함되어 있습니다. 이 파일의 터미널 항목 대부분은 터미널 제조업체가 터미널과 함께 제공한 정보에 기초한 내용입니다. 새로운 터미널이 계속 나오고 있기 때문에, 모든 터미널에 대한 정보를 가지고 있는 termcap 파일을 제공한다는 것은 힘든 일입니다. 만약 사용하는 터미널에 대한 항목을 Informix termcap 파일에서 찾지 못했다면, 시스템 termcap 파일에서 필요한 항목을 찾을 수 있을 것입니다. 시스템 termcap에서도 찾을 수 없다면, 터미널 제조업체에 termcap 내용을 요청할 수 있을 것입니다.

Unix 환경에서의 Informix Termcap

Informix 제품은 기본으로 시스템 termcap 파일을 사용합니다. 기본 설정이 아닌, Informix termcap 또는 다른 termcap을 사용하려면, 환경변수 TERMCAP을 해당 termcap의 위치로 설정하면 됩니다.

참고: 비 Informix 항목의 경우 일부 터미널 성능이 포함되거나 또는 제외되는데, 이것은 Informix 제품 동작에 영향을 미칠 수 있습니다.

Informix termcap 파일을 사용하려면, 우선 사용하는 터미널에 대한 항목이 존재하는지 확인해야 합니다. 다음 명령 중 하나를 이용하면, 현재 사용중인 터미널이 시스템에 어떻게 인식되어 있는지 알 수 있습니다.

$ echo $TERM

set (출력 내용에서 TERM 변수를 찾음.)

env (출력 내용에서 TERM 변수를 찾음.)

Informix termcap은 ASCII 파일이며, $InformixDIR/etc (InformixDIR는 환경변수로서, 사용 시스템의 Informix 제품 설치 위치를 가리킵니다.) 디렉토리에 있습니다. ASCII 파일이므로, 아무 편집기나 UNIX 유틸리티를 이용해 필요한 termcap 항목을 찾을 수 있을 것입니다. termcap 파일에서 환경변수 TERM이 가리키는 터미널 형식을 찾습니다.

Informix 제품은 초기화될 때, 항상 termcap 파일에서 환경변수 TERM에 지정된 항목과 일치하는 항목을 검색합니다. 자주 사용되는 termcap 항목을 termcap 파일 앞 부분에 위치시키는 것이 좋습니다. 이렇게하면, 초기화 과정에서 termcap 항목을 찾는데 소요되는 시간을 줄일 수 있습니다.

termcap을 읽으려면 몇가지 기본적인 구문 규칙을 이해해야 합니다. 그림 1은 DEC VT100 터미널의 termcap 항목 중 일부를 발췌한 것입니다.

Informix TERMCAP ENTRY FOR DEC VT100 EMULATION

FOR USE ON UNIX SYSTEMS

v1|vt100|vt-100|DEC vt100:₩

:bs:am:xn:xv:pt:cd=₩E[J:ce=2₩E[K:cl=45₩E[H₩E[J:₩

:cs=%i₩E[%d;%dr:cm=%i₩E[%d;%dH:co#80:li#24:₩

:Hi=₩E=:Hf=₩E> :HI=^ |:Po=₩EI:Pe=₩E2:₩

:bc=₩E[D:do=₩E[B:nd=₩E[C:sr=₩EM:up=₩E[A:₩

:so=₩E[7m:se=₩E[m:₩

:ku=₩E[A:kd=₩E[B:kr=₩E[C:kl=₩E[D:₩

:kh=₩E[H:us=₩E[4m:ue=₩E[Om:₩

.

.

.

그림 1: VT100 termcap 항목

termcap 항목이 나오기 전에 각 항목에 대한 설명은 파운드 기호(#)로 시작되는 행에 기록할 수 있습니다. 위 예에서 제시된 VT100 항목에서는 처음 두행이 설명 행입니다.

Informix TERMCAP ENTRY FOR DEC VT1OO EMULATION # FOR USE ON UNIX SYSTEMS

termcap 항목의 본문 내용 중 첫번째 행은 터미널의 별칭으로 수직 막대 문자로 구분합니다. 이 별칭 중에 하나를 선택하여 환경변수 TERM에 지정하면 됩니다. Informix 응용프로그램은 초기화 과정에서 이 별칭과 환경변수 TERM의 내용을 비교하고, 일치하는 터미널 항목을 사용합니다.

v1|vt100|vt-100|DEC vt100:₩

행 끝의 콜론 문자는 별칭 목록이 끝나고, 다음에 성능 목록이 이어진다는 것을 표시합니다. termcap 항목

은 단 한 줄로 정의되어야 하기 때문에, 개행 문자가 기록되지 않도록 역슬래시 문자를 사용합니다. 그 뒤의 행들이 실제 터미널 성능 목록입니다. 각 행은 탭 문자로 들여쓰며, 각 터미널 성능은 콜론(:) 문자로 분리됩니다.

:bs:am:xn:xv:pt:cd=₩E[J:ce=2₩E[K:c1=45₩E[H₩E[J:₩

앞에서 설명한 바와 같이, 시스템 termcap 파일을 사용하지 않으려면 환경변수 TERMCAP을 다른 파일로 지정하면 됩니다. TERMCAP 변수가 Informix 제품에서 제공하는 termcap 파일을 가리키도록 설정할 수 있습니다. 본(Bourne) 쉘의 경우, 다음 명령을 사용하면 됩니다:

TERMCAP=$InformixDIR/etc/termcap export TERMCAP

C 셸이라면, 다음 명령을 사용하십시오.

setenv TERMCAP $InformixDIR/etc/termcap

Dos 환경에서의 Informix Termcap

Informix 제품은 비디오 디스플레이 RAM에 직접 기록하는 DOS 환경에서는 termcap 사용을 지원하지 않습니다. 따라서 DOS 환경에서는 Informix 제품에 사용 터미널 형식을 알리기 위해 환경변수를 설정할 필요가 없습니다.

결론

이 글에서는 Informix 제품에서 termcap을 사용하는 방법과, 기존의 termcap을 사용하도록 하는 환경설정 방법에 대해 개략적으로 알아보았습니다. termcap 항목에 대해 자세히 알고 싶으면 다음 문서를 참고하십시오.

Informix 4GL Reference Manual, Volume Two, Appendix I

Informix 제품을 위한 Termcap 항목작성

서론

이 글은 일반적인 termcap 성능에 대해 정의하고, Informix 제품이 필요로하는 성능에 대해 설명합니다.

일반적인 Termcap 성능

termcap 항목은 부울, 숫자, 문자열의 세 가지 종류의 성능으로 구성되어 있습니다. termcap에 기록된 부울 성능 항목은 터미널의 해당 성능 보유 여부를 나타냅니다. 어떤 성능에 대한 부울 항목이 termcap에 존재하지 않는다면, 그 값은 거짓으로 가정됩니다. 예를 들어, am이 termcap에 기록되었다면, 이 값은 해당 터미널에 자동 줄바꿈 기능이 있는지 여부를 나타냅니다. 숫자 성능 항목은 항목 이름 다음에 파운드 기호 및 숫자로 구성됩니다. 이 숫자는 해당 성능의 값을 나타냅니다. 예를 들어, co#80은 터미널이 80 컬럼이라는 것을 나타냅니다.

문자열 성능 항목은 성능 이름 다음에 등호와 문자열로 구성됩니다. 이 문자열은 터미널이 출력할 수 없는 문자를 받았을때 사용하는 코드화된 제어 문자 또는 이스케이프 시퀀스이거나, 응용프로그램이 터미널에 작업을 지시하기 위한 템플릿으로 사용하는 명령 및 제어 문자를 조합한 것입니다.

예를 들어, up=^ K 항목은 터미널에 ^ K가 커서를 한 줄위로 옮기는 명령이라는 것을 알립니다.

이스케이프 문자와 제어 문자

코드화된 이스케이프 문자 혹은 제어 문자는 마치 암호문처럼 보입니다. 가장 일반적인 형태는 이스케이프 문자는 ₩E이고, 제어 문자는 ^ 입니다. 그림 1의 도표는 문자열 성능 항목에 주로 사용되는 문자를 정의해 놓은 것입니다.

코드	설명	비고
₩E	이스케이프 문자	

^x	ctrl-x	여기서 x 는 임의의 문자
₩n	개행	
r	리턴	
₩t	탭	
₩b	백스페이스	
₩f	폼피드	
₩xxx	8 진수 값 xxx	반드시 3 개의 문자여야 함
₩041	느낌표(!)	C-셸 히스토리 기능에서 사용
₩072	콜론(:)	termcap 에서 구분자로 사용
₩200	널	₩000 는 널로 작용하지 않음

그림 1: 제어 문자 정의

termcap 매개변수

문자열 성능 중에는 응용프로그램에서 사용되는 문자열에 포함되는 코드화된 명령이 있습니다. 이 명령에는 실행시간에 실제 값이 지정되는 매개변수가 있습니다. 응용프로그램은 이 성능을 읽어들이고, 코드화된 명령에 따라 매개변수를 실제 값으로 대체한 다음, 그것을 ASCII 문자열로 변환해 터미널로 보냅니다. 그림 2는

Informix 제품에 사용할 수 있는 명령을 정리한 것입니다.

코드	설명
%d	주어진 값을 십진수 숫자로 보냅니다.
%2	주어진 값을 2 자리 십진수 숫자로 보냅니다.
%3	주어진 값을 3 자리 십진수 숫자로 보냅니다.
%.	주어진 숫자에 해당하는 ASCII 문자를 보냅니다.
%+x	보내기 전에 주어진 값에 ASCII 값 x 를 더합니다.
%>xy	주어진 값이 x 보다 크다면, y 를 더하고, 보내지는 않습니다.
%r	인수 순서를 원래 순서의 역으로 만듭니다.
%i	주어진 값에 1 을 더합니다.
%%	퍼센트 기호를 표시합니다.
%B	주어진 값을 이진코드화 십진수로 변환합니다.
%D	이어지는 값이 예약된 값임을 표시합니다.

그림 2: Informix 제품에서 사용할 수 있는 코드화된 인수의 명령

다음 예는 성능 항목인 cm을 사용하여 매개변수로 대체하는 것을 보인 것입니다. cm 성능은 Informix SQL 등의 응용프로그램에서 사용되는 x,y 좌표 기반의 스크린에서 지정된 위치로 커서를 이동시킵니다. 아래 예는 가상 터미널에 대한 것이며, 특정 터미널을 고려한 것은 아닙니다. 사용중인 터미널에 적합한 제어 문자에 대해서는 터미널 제조업체에서 제공하는 프로그래머를 위한 참조 설명서를 참고하십시오. 각각의 예에서는 대체되는 값은 5와 10입니다. "Informix 응용프로그램이 터미널로 보내는 제어 문자"라는 제목 아래 보이는 내용은 의사 코드입니다. Informix 응용프로그램이 실제로 보내는 제어 문자는 각각의 문자에 해당하는 ASCII 값으로서 esc 또는 ₩E인 경우 ASCII 27입니다.

Termcap 항목	Informix 제품이 터미널로 보내는 제어 문자	비고
cm=₩E&a%3c%2Y	esc&a005c10Y	x,y 순서의 좌표를 요구하는 터미널인 경우
cm=₩E&a%r%2c%3Y	esc&a10c005Y	y,x 순서의 좌표를 요구하는 터미널인 경우
cm=5₩E[%i%d;%d	5esc[6;11	x+1, y+1 의 좌표를 요구하는 터미널인 경우

Informix 제품에 필요한 termcap 성능

Informix 제품이 정상적으로 동작하기 위해서는 termcap에 다음과 같은 성능이 반드시 포함되어야 합니다.

일반적인 성능

cl

화면 전체를 지움. Informix 4GL의 CLEAR SCREEN 문과 같습니다.

ce

행 끝까지 지움. 현재 행에서 커서 위치 이후의 나머지를 지울 때 사용됩니다. 행 끝까지 공백인 메시지, 또는 기초 윈도우를 다시 그리는 경우에 이 성능을 사용합니다.

ti

커서 이동 기능을 사용하는 터미널을 초기화합니다. 이 성능은 Informix 응용프로그램에서 셸 프로세스를 시작할 때 또는 셸 프로세스를 마치고 돌아올 때 사용됩니다(Informix 4GL의 RUN 명령이나 Informix SQL의 ! 명령의 경우).

te

ti 설정에서 다시 원래 설정으로 되돌립니다. 이 성능은 각 Informix 프로세스가 종료되기 전에 사용됩니다.
커서 이동 기능

cm

커서를 화면 상의 지정된 위치로 이동시킵니다.

bt

커서를 이전 하드웨어 탭 위치로 이동시킵니다.

do

커서를 한 줄 아래로 이동시킵니다.

ho

커서를 초기 위치인 화면 좌상단으로 이동시킵니다.

ll

커서를 화면 좌하단으로 이동시킵니다.

nd

커서를 오른쪽으로 한 위치만큼 이동시킵니다.

ta

커서를 다음 하드웨어 탭 위치로 이동시킵니다.

up

커서를 한줄 위로 이동시킵니다.

다음 항목들은 사용중인 터미널에 해당하는 경우에만 termcap에 포함시켜야 합니다.

am

터미널에 자동 줄바꿈 기능이 있음.

bc

^ H가 아닌 터미널 고유의 다른 백스페이스 제어 문자

ms

블록 또는 밑줄 모드에서 커서가 안전하게 이동됨.

xs

블록 모드에서 겹쳐쓰기가 자동으로 처리됨.

다음의 termcap 성능 항목은 선택적입니다.

ks

키패드 on. 이 성능은 매번 Informix 응용프로그램이 시작될 때 마다 키패드를 숫자 모드로 전환합니다.

ke

키패드 off. 이 성능은 Informix 응용프로그램이 종료되기 전에 키패드를 응용프로그램 모드로 전환합니다.

ki

행 삽입. 4GL 배열에서 사용

kj

행 삭제. 4GL 배열에서 사용

kf

다음 페이지. 4GL 배열에서 사용

kg

이전 페이지. 4GL 배열에서 사용

so

블록 모드 시작. 역상 속성이 지정될 때 사용합니다.

se

블록 모드 종료

sg

블록 모드로 변경하는데 소요되는 공간 수

ue

밑줄 모드 종료

us

밑줄 모드 시작

ug

밑줄 모드로 변경하는데 필요한 공간 수

uc

현재 문자에 밑줄표시를 하는 제어 문자

k0~k9

F1에서 F10까지의 기능 키에 해당

kA~kZ

F11에서 F36까지의 기능 키에 해당. 예를 들어 k0는 1번 기능키와 대응됩니다.

참고 : 여기서는 Informix 제품이 사용하는 성능만을 설명합니다. vi 같은 다른 프로그램의 실행을 위해서는 이외의 다른 성능도 필요할 것입니다.

중요사항

Informix 응용프로그램은 특정 termcap 성능의 존재 여부에 따라 다음과 같은 몇 가지 사항을 조정합니다.

1. ue, us, ug 성능은 termcap에 so 성능 항목이 없을 경우에만 사용됩니다. so 항목이 없으면 ue, us, ug는 각각 se, so, sg를 대신합니다. 역상, 블록 등 모든 속성은 정상적으로 밑줄과 함께 표시될 것입니다.

면 형식에 문제가 발생할 것입니다.

3. Informix 제품은 is, if, rs, rf 속성을 사용하지 않습니다.
4. 만약 ki, kj, kf, kg 항목이 있다면, 4GL의 옵션 구문으로 이 항목의 설정을 바꿀 수 없습니다. 이 항목과 입력 배열 작업에서 쓰이는 기능 키 F1~F4를 혼동하지 않도록 하십시오.

결론

이 글에서는 termcap 성능 항목이 일반적으로 어떻게 정의되는지 설명하고, 특히 Informix 제품의 정상적 작동에 필수적인 termcap 성능에는 어떤 것들이 있는지 설명했습니다. 역상, 색상 항목, Informix 제품의 terminfo 사용 방식은 다음의 문헌을 참고하십시오.

참고문헌

Informix 4GL Reference Manual, Volume Two, Appendix I

동적관리 명령문

서론

이 글은 SQL을 사용하는 Informix ESQL/C, Informix ESQL/COBOL, Informix 4GL 등 모든 버전에 적용됩니다.

동적 관리 명령문을 제대로 사용하면, 효과적인 프로그램을 작성할 수 있습니다. 다음과 같은 경우에 동적 관리 명령문의 사용을 고려해야 합니다.

- 동적 관리 명령문을 사용하여 컴파일시에는 알 수 없는 값이나 알 수 없는 명령 구조 처리하기
- 동적 SELECT 명령문과 비-SELECT 명령문 실행의 차이점
- 루프 밖에 SQL 명령문을 배치하여 프로그램 효율 증가시키기
- DESCRIBE 명령문을 사용하여 명령문에 대한 정보 얻기 (Informix ESQL/C에서만 사용 가능)
- 동적 접근 요구 생성에 있어서 Informix ESQL/C, Informix ESQL/COBOL, Informix 4GL 제품의 차이점

동적관리 명령문

동적 관리 명령문이란, 명령문이 작성될 때에는 해당 값을 알 수 없는 하나 이상의 호스트 변수를 참조하는 SQL 명령문을 말합니다. 컴파일시 구조를 알 수 없는 SQL 명령문도 동적 관리 명령문이 될 수 있습니다.

동적 관리 명령문은 모두 동적 키워드로 시작합니다. 그림 1은 각 동적 키워드에 대하여 설명한 것입니다.

SQL 동적 관리 명령문	
명령문	작업
PREPARE	문자열을 SQL 명령문으로 해석합니다
PREPARE	준비된 명령문에 대한 정보를 얻습니다
EXECUTE	준비된 명령문을 실행합니다
DECLARE	준비된 명령문을 위한 커서를 설정합니다

그림 1: 동적 키워드

여러 가지 유형의 고급 프로그래밍 응용프로그램에서 컴파일시에 알 수 없는 것을 처리해야 할 경우에 동적 관리 명령문을 사용할 수 있습니다. 동적 관리 명령문을 사용하면 보통 SQL 명령문보다 나은 유연성을 추가로 얻을 수 있습니다.

다음과 같은 유형의 응용프로그램에서는 대부분 동적 관리 명령문을 사용해야 합니다.

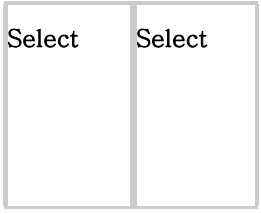
- 서로 다른 데이터베이스로 작업하도록 설계된 프로그램
- 컴파일시에 데이터 형식을 알 수 없는 프로그램 (예를 들어, 나중에 형식이 지정되는 호스트 변수가 WHERE 절에 포함되어 있을 경우)
- 컴파일시에 컬럼이나 매개변수의 수를 알 수 없는 프로그램 (예를 들어, 컬럼의 수가 나중에 지정되는 SELECT 명령문의 경우)
- 사용자가 (질의, 삽입, 갱신 등의) 데이터베이스 접근 요구를 입력하는 대화형 프로그램

동적 관리 명령문을 사용해야 하는 경우 외에, PREPARE 명령문을 사용하여 프로그램 효율을 증가시킬 수 있는 경우가 있습니다. 이에 대해서는 "효율적인 루프 사용과 최적화" 절에서 다룹니다.

SELECT 명령문과 비-SELECT 명령문 사이의 차이점

동적 SELECT 명령문과 동적 비-SELECT 명령문은 서로 다르게 사용됩니다. (이 글에서 사용하는 SELECT 명령문이란 말은 INTO TEMP 절이 포함되지 않은 SELECT 명령문을 말합니다.) 동적 비-SELECT 명령문이나 INTO TEMP 절이 포함된 SELECT 명령문을 실행하려면, PREPARE 명령문과 EXECUTE 명령문을 사용해야 합니다. 동적 SELECT 명령문을 실행하려면 PREPARE, DESCRIBE(필요할 경우), DECLARE, OPEN, FETCH, CLOSE 등의 명령문을 사용해야 합니다.

동적 SELECT 명령문을 실행할 것인가, 아니면 비-SELECT 명령문을 실행할 것인가에 따라 사용해야 하는 명령문의 순서를 그림 2에 나타내었습니다.



FETCH CLOSE FREE	

그림 2: 필요한 명령문 순서

이 그림에는 다음과 같은 내용이 들어 있습니다.

- (SELECT 명령문에 INTO TEMP 절이 포함되지 않으면) SELECT 명령문과 함께 EXECUTE 명령문을 사용할 수는 없습니다. 커서 관리 명령문을 사용해 SELECT 명령문을 실행하십시오.
- 동적 SELECT 명령문을 실행하려면, PREPARE 명령문, 그리고 필요할 경우에 DESCRIBE 명령문을 사용해야 합니다. 준비된 SELECT 명령문을 위한 커서를 선언한 다음, 그 커서를 OPEN하고, FETCH하고, CLOSE시켜야 합니다. 동적으로 정의된 모든 SELECT 명령문에는 이미 DECLARE된 커서로 사용되어야 합니다.
- PREPARE된 SELECT 명령문을 제외한 모든 동적 명령문에는 EXECUTE 명령문을 사용해야 합니다. 이 글에 있는 예들을 보면 이러한 SQL 명령문의 사용법을 알 수 있습니다.

동적관리 명령문에서 수행되는 작업

컴파일시에는 알 수 없는 값을 처리하려고 하거나 프로그램 효율을 증가시키려고 할 때는 동적 명령문을 실행하는 동안 어떤 일이 발생하는가에 대해 미리 알아두는 것이 좋습니다. 이 절에서는 동적 관리 명령문을 실행할 때 수행되는 과정에 대해 간단하게 다룹니다.

그림 3은 Informix 제품의 Front End과 Back End 구조를 구성하는 여러 가지 구성 요소들을 보여줍니다.

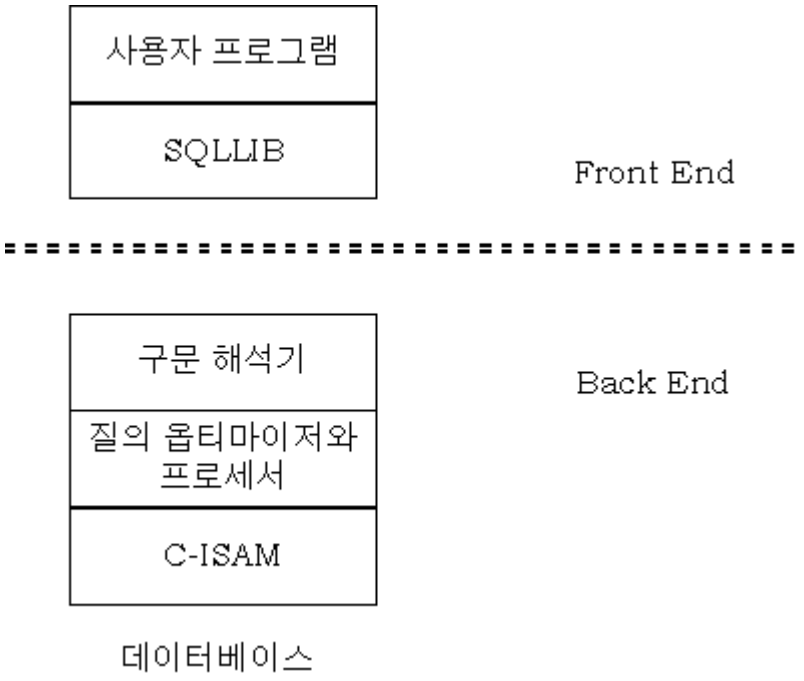


그림 3: Front End와 Back End 구조

아래 단락에서는 실행중에 수행되는 작업을 개략적으로 설명합니다.

PREPARE

명령문이 PREPARE되면, Front End는 문자열을 Back End에 메시지로 송신하여, 그 문자열의 구문을 해석하라고 요구합니다. Back End은 그 문자열의 구문을 해석하고 그 명령문을 처리할 공간을 할당합니다.

DESCRIBE (ESQL/C에서만 사용 가능)

명령문의 종류나 프로그램 변수의 수를 알지 못하는 상황이 아니면, DESCRIBE 명령문은 선택적입니다.

DESCRIBE 작업을 하는 동안, Front End는 Back End에게 SQL 명령문의 형식에 대한 정보를 요구합니다.

Back End는 명령문 형식을 Front End로 반환합니다.

DESCRIBE 되고 있는 명령문이 행을 반환하는 SELECT 명령문이면, DESCRIBE문은 그 SELECT 명령문이 실행될 때 반환될 행에 대한 설명을 반환합니다.

반환되는 값을 위해 적당량의 메모리를 할당해야 합니다. 또, 지시자 변수 (indicator variable; 반환되고 있는 값이 절삭되거나 NULL이라는 것을 나타내는 변수)를 위한 메모리도 할당해야 합니다.

EXECUTE

Back End가 EXECUTE 명령을 받으면, 그 명령문을 최적화하여 실행합니다. 실행되는 명령문에 대한 최적화는 EXECUTE 명령문을 처리하는 동안에만 발생합니다. 최적화를 하는 동안, Back End는 모든 정보를 종합해서 가장 효율적으로 행을 처리하는 방법을 결정합니다.

DECLARE

명령문이 SELECT 명령문이면, 반환되는 행을 관리하기 위해 커서를 선언해야 합니다. DECLARE문 다음에는 OPEN, FETCH, CLOSE문이 차례로 쓰입니다. (INTO TEMP 절이 있는 SELECT 명령문은 EXECUTE 명령문을 사용해 실행됩니다.)
이미 PREPARE된 명령문으로 커서를 선언할 경우에는 DECLARE 명령문이 그 명령문의 구문을 다시 분석하지 않을 것입니다. 그러나 PREPARE에 쓰이는 명령문이 이전에 PREPARE되지 않는다. 여기서 구문 분석을 실행하게 됩니다.

OPEN

OPEN 명령문은 파일을 열고 파일 포인터를 초기화합니다. 처음으로 커서가 OPEN되는 것이라면, OPEN 명령문이 실행되는 동안 Back End가 SELECT 명령문을 최적화합니다. 커서가 이전에 이미 OPEN된 적이 있다면, Back End는 파일 포인터를 재설정하기만 합니다.

FETCH

FETCH 명령문은 커서가 가리키는 다음 행의 내용을 변수로 가져옵니다.

CLOSE

CLOSE 명령문은 OPEN된 커서를 닫습니다.

FREE

FREE 명령문은 PREPARE된 오브젝트나 CLOSE된 커서에 할당되었던 엔진 자원을 선택적으로 해제합니다. 주: 비-SELECT 명령문의 경우, 또는 행을 반환하지 않거나 한 행을 반환하는 SELECT 명령문의 경우는 구문 분석, 최적화, 실행이 동시에 수행됩니다.

PREPARE명령문

PREPARE 명령문은 값을 알 수 없는 변수가 포함된 명령문의 실행을 돕는 동적 관리 명령문입니다. 루프에서 PREPARE 명령문을 적절히 배치하면, 오버헤드를 크게 줄일 수 있습니다. 이 내용은 "효율적인 루프 사용과 최적화" 절에서 다루기도 하고, 이 절에서는 PREPARE 명령문에 대한 개요입니다.

PREPARE 명령문은 나중에 실행될 때를 위해 Back End가 SQL 명령문이나 다중 SQL 명령문을 미리 처리하도록 합니다. 다음과 같은 경우에 PREPARE 명령문을 사용합니다.

- 컴파일시 SQL 명령문의 구조를 알 수 없는 경우
- 알 수 없는 자료 값이나 자료 형태를 포함한 동적 명령문을 실행하고자 하는 경우
- 자주 사용되는 비-SELECT 명령문의 실행 시간을 줄이고자 하는 경우

다음의 예는 PREPARE 명령문과 의문 부호(?) 사용법을 보여줍니다. DELETE 문이 완전히 이루어진 것이 아니며, 따라서 컴파일시에 DELETE 명령문의 구조를 알 수 없기 때문에 PREPARE 명령문이 필요합니다. 이 예에서 사용자는 WHERE 조건에 사용될 칼럼명을 column_name이라는 변수로 입력받을 수 있고, 또 그 칼럼명에 사용될 데이터 값도 column_value라는 변수로 입력받아 사용할 수 있도록 한 것입니다.

```
PROMPT "Enter column name: " FOR column_name

LET sql_str = "DELETE FROM customer WHERE", " ",
```

```
PREPARE x FROM sql_str

PROMPT "Enter search value: " FOR column_value

EXECUTE x USING column_value
```

컴파일시에는 column_name이란 컬럼에 사용할 값을 알 수 없기 때문에, 프로그램이 값을 Back End에 송신할 수 없습니다. 소스 코드에 *?*를 넣으면 이 문제를 해결할 수 있습니다. *?*는 컴파일시가 아니라 프로그램이 실행될 때에 비로소 EXECUTE (또는 OPEN) 명령문을 실행할 수 있도록 필요한 값들이 주어질 것이라고 Back End에 알리는 것입니다.

PREPARE 명령문을 실행하는 동안, Front End는 Back End에 문자열을 송신함으로써, 그 문자열의 구문을 분석하고 구문 분석 후에는 해당 명령문에 대하여 명령문 지시어(statement identifier)를 기억하도록 요청합니다. 위의 예에서 x는 명령문 지시어입니다.

효율적인 루프사용과 최적화

루프에 PREPARE 명령문을 사용하면, 중복 수행으로 생기는 오버헤드를 없앨 수 있습니다. 이렇게 하려면 루프의 적절한 위치에 PREPARE 명령문을 배치해야 합니다. "동적 관리 문에서 수행되는 작업" 절에서 설명했듯이, 명령문의 구문 분석에는 Back End 오버헤드가 크게 작용합니다. 예 1A에서는 루프가 실행될 때마다, WHILE 루프 안에 있는 UPDATE 명령문의 구문이 해석됩니다. 반복되는 구문 해석으로 추가되는 오버헤드 때문에 실행 시간이 길어집니다. UPDATE 명령문을 루프 밖에 배치하면, 예 1A에서 나오는 오버헤드 중 상당량을 없앨 수 있습니다.

예 1A (Informix 4GL):

```
WHILE TRUE

        PROMPT "enter customer number" FOR eno

        IF eno = 0 THEN

        EXIT WHILE

              END IF

        UPDATE customer

            SET discount = .1

            WHERE cust_num = eno

END WHILE
```

UPDATE 명령문을 루프 밖에 두고 루프 내부를 실행하면, 구문 분석은 단 한 번만 실행됩니다. 따라서 Back End는 같은 명령문이 실행될 때마다 구문 분석을 해야 하는 오버헤드를 발생시키지 않을 것입니다.

이렇게 실행하기 위해서 우리는 실행시에 값만 바꿔주면 됩니다. 위에서 보여준 프로그램 예 1A보다 실행 시간이 몇 배는 빨라질 수 있습니다. 루프가 실행되는 횟수가 많아질수록 효율은 높아집니다. 예 1B는 더 효율적인 프로그램이 될 것입니다.

예 1B (Informix 4GL):

```
PREPARE x FROM

        "update customer set discount = .1 where cust_num = ?"

WHILE TRUE

        PROMPT "enter customer number" FOR eno

        IF eno = 0 THEN EXIT WHILE

        END IF



                EXECUTE x USING eno

END WHILE
```

다음 두 프로그램(예 2A와 2B)은 할인율이 10%인 고객들의 범위를 갱신하는 것입니다.

예 2A에서는 UPDATE 명령문이 WHILE 루프 내에 있으므로 구문 분석 역시 WHILE 루프로 처리될 것입니다. 고객을 갖고 있는 데이터베이스라면 명령문의 구문 분석을 반복하는데 많은 시간이 필요합니다.

예 2A (Informix 4GL):

```
PREPARE sel_stmt from
```

```
"select * from customer where cust_num between ? and ? for update"

DECLARE x CURSOR FOR sel_stmt

OPEN x USING low, high



WHILE TRUE

        FETCH x INTO r_cust.*

        IF STATUS = NOTFOUND THEN

        EXIT WHILE

        END IF

        LET ptext = "Update ", r_cust.fname CLIPPED, " ",

            r_cust.lname CLIPPED, "?"

        PROMPT ptext FOR CHAR yn



        IF yn = "y" THEN
```

```
            UPDATE customer

            SET discount = .1

            WHERE CURRENT OF x

        END IF

END WHILE
```

예 2B에서는 UPDATE 명령문을 WHILE 루프 밖으로 처리했습니다. 따라서 구문 분석은 단 한 번만 이루어집니다. 이렇게 하면 많은 오버헤드를 줄일 수 있을 것입니다.

예 2B (Informix 4GL):

```
PREPARE sel_stmt FROM

"select * from customer where cust_num between ? and ? for update"

DECLARE x CURSOR FOR sel_stmt

OPEN x USING low, high

PREPARE u FROM

            "update customer set discount = .1 where current of x"
```

```
WHILE TRUE

        FETCH x INTO r_cust.*

        IF STATUS = NOTFOUND THEN

        EXIT WHILE

        END IF

        LET ptext = "Update ", r_cust.fname CLIPPED, " ",

                r_cust.lname CLIPPED, "?"

        PROMPT ptext FOR CHAR yn

        IF yn = "y" THEN

        EXECUTE u

        END IF

END WHILE
```

예 3A와 3B는 SELECT 명령문과 UPDATE 명령문을 사용하여 레코드와 갱신 차액을 검색하는 것을 보여 줍니다.

예 3A에서는 SELECT 명령문과 UPDATE 명령문이 모두 FOR 루프 내에서 해석됩니다. 3000 행이 있는 데이터베이스에서 예 3A 형식의 프로그램을 실행하면 6000번의 구문 분석이 이루어집니다.

예 3A (Informix ESQL/C):

```
for (ndx=l;ndx<=cnt;ndx++)

        {

      $SELECT balance INTO $bal FROM account

           WHERE accnum = $ndx;

      bal += 100;

      $UPDATE account SET balance = $bal

           WHERE accnum = $ndx;

      }
```

예 3B에서는 UPDATE 명령문이 FOR 루프 밖에 있습니다. SELECT 문을 실행하기 위해 커서를 선언하고, OPEN하고, FETCH하고, CLOSE합니다. UPDATE 명령문을 실행합니다. (동적 관리 명령문의 올바른 키워드 순서는 "SELECT 명령문과 비-SELECT 명령문의 차이점") 절에서 다룹니다.

예 3B는 프로그램에서는 구문이 분석이 단 두 번 이루어집니다. (다른 코딩 방법을 사용해 같은 결과가 나

오도록 할 수도 있습니다.)

예 3B (Informix ESQL/C):

```
$DECLARE selcurs CURSOR FOR

        SELECT balance FROM account WHERE accnum = $ndx;




$PREPARE updstmt FROM

        "UPDATE account SET balance = ? WHERE accnum = ?";

for (ndx=l;ndx<=cnt;ndx++)

            {

            $OPEN selcurs USING $ndx;

            $FETCH selcurs INTO $bal;

            bal += 100;

            $EXECUTE updstmt USING $bal,$ndx;

            $CLOSE selcurs
```

```
}
```

DESCRIBE 명령문

PREPARE된 명령문은 컴파일할 때에는 알 수 없는 정보가 있습니다. (Informix ESQL/C에서만 사용할 수 있는) DESCRIBE 명령문은 이러한 PREPARE된 명령문의 알 수 없는 정보를 얻는 동적 관리 명령문입니다. 일례로, 여러분은 SELECT 명령문에 의해 반환되는 데이터를 저장하기 위해 얼마나 많은 프로그램 변수가 필요한지 알 수 없습니다. DESCRIBE 명령문은 이러한 정보를 얻음으로써, 알 수 없는 양의 변수를 포함하는 프로그램을 작성하거나 알 수 없는 많은 명령문 형식을 제대로 처리할 수 있는 코드를 작성할 수 있도록 유연성을 제공합니다.

DESCRIBE 명령문을 사용하면 다음과 같은 정보를 얻을 수 있습니다.

- SQL 명령문의 종류
- SELECT 명령문의 경우, 반환되는 값을 유지하기 위해 필요한 프로그램 변수의 수와 자료 형식

명령문의 종류와 프로그램 변수의 수를 알게 되면, 여러분의 프로그램이 하나의 행을 유지하기 위해 필요한 메모리를 할당할 수 있습니다. 그리고 이 경우에만 Back End 명령문을 실행할 수 있습니다.

DESCRIBE 명령문을 사용하면 PREPARE된 명령문에 대해 이러한 정보를 얻을 수 있습니다. DESCRIBE 명령문은 반환되고 있는 컬럼을 설명하는 SQLDA 구조 (Informix ESQL/C에서 미리 정의된 구조)의 집합을 초기화합니다.

DESCRIBE 명령문에 대한 정보가 더 필요하면 Informix ESQL/C 제품 문서를 참조하십시오.

Informix ESQL/C, Informix ESQL/COBOL, Informix 4GL 사이의 차이점

동적 질의와 접근 요구 생성은 Informix ESQL/C, Informix 4GL, Informix ESQL/COBOL에서 서로 다르게 수행됩니다. 다음 목록은 이런 면에서 세 제품 사이의 주된 차이점을 개략적으로 설명합니다.

- Informix ESQL/C에서 DESCRIBE 명령문을 사용하면 실행될 명령문에 대한 정보를 얻을 수 있습니다. Informix 4GL이나 Informix ESQL/COBOL에는 DESCRIBE 명령문이 없기 때문에, 컴파일시에 명령문의 형식과 반환되는 값의 수와 형식을 알아야 합니다.
- Informix 4GL에서는 CONSTRUCT 명령문이 동적 질의를 처리하는 기본적인 방법입니다. CONSTRUCT 명령문을 사용하면, 실행시에 화면에 생성되는 Query by Example을 사용하여 사용자가 질의를 생성할 수 있습니다. Informix ESQL/C나 Informix ESQL/COBOL에서는 CONSTRUCT 명령문을 사용할 수 없기 때문에, 수동으로 WHERE 절을 생성해야 합니다. CONSTRUCT 명령문에 대한 정보가 더 필요하면, Informix 4GL 참고 설명서를 보십시오.

Informix Online Dynamic Server의 다중 쓰레드

Informix OnLine Dynamic Server 릴리즈는 데이터베이스 서버 내에 다중 스레드 기술을 통합한 것입니다. 이 기술은 이전 Informix 이중 프로세스 모델의 구조적 변경을 나타냄과 동시에, 다중 스레드에 의해 가능해진 OLTP 프로세싱에 대한 새로운 접근법을 나타내기도 합니다.

주: 이제부터 두 제품을 구분하기 위해, Informix OnLine Dynamic Server는 "OnLine Dynamic Server"라고 하고, Informix OnLine은 간단하게 "OnLine"이라고 하겠습니다.

이 글에서는 동적 확장 구조(DSA)의 설계와 기본 개념을 알아보도록 하겠습니다. 가변값, 유틸리티, 그리고 그 옵션들을 망라한 설명을 하고자 하는 것은 아닙니다. 다만, 이 글이 다중 스레드 DSA 엔진에 대한 한정되면서도 추상적인 개념을 바꾸는데 도움이 되었으면 합니다.

새로운 다중 스레드 엔진 설계의 상대적인 중요성을 더 잘 이해하기 위해서는, 우선 현재의 이중 프로세스 모델을 검토하고, 다중 스레드가 피하려고 하는 이 모델 고유의 한계를 몇 가지 검사하도록 하겠습니다.

전통적인 이중 프로세스모델

OnLine 처리 모델을 간단히 설명하자면, 데이터베이스 요구를 생성하는 Front End 또는 클라이언트 프로세스와 데이터베이스 요구를 처리하는 Back End 또는 서버 프로세스로 구성됩니다. 이 모델에서는 작업을 완료하기 위해 클라이언트/서버 통신중에 다중 프로세스를 사용합니다. Front End(예를 들어, Informix ESQL/C 프로그램)는 클라이언트 시스템 측에 상주하고, 로컬 Back End 프로세스는 네트워크를 통해 서버 시스템과 통신을 합니다. 또, 서버 시스템 상의 Back End 프로세스는 실제로 데이터베이스 요구를 수행하고, 서버 시스템 상에는 별도의 데몬(sqlexecd)이 실행되어 서버 Back End 프로세스를 분기시킵니다. 간단히 말해, OnLine에서는 추가 프로세스, 문맥 전환, 통신 계층 등이 없이 클라이언트가 직접 서버와 대화를 할 수는 없습니다.

클라이언트/서버든, 로컬 통신이든, 이중 (또는 그 이상) 프로세스 모델의 한계는 분명합니다. 특히 오버헤드 비용을 생각해 보면 명확해집니다. 프로세스의 수가 클수록 커널 테이블에 많은 양의 메모리가 할당되고 더 많은 명령이 스케쥴러에 의한 실행을 요구하기 때문에, 유닉스 커널에 과중한 로드가 있게 됩니다. 이러한 것들은 운영체제 관리 작업의 일부에 지나지 않습니다.

이중 프로세스 모델에서 가장 비용이 많이 드는 기능은 모델 자체가 가지고 있는 제한입니다. OnLine 구조는 각 엔진 프로세스가 독립적으로 유지되도록 되어 있습니다. 각 엔진 프로세스는 자체에만 의존해서 사용자 요구에 응답하고, 디스크 입출력을 수행하고, 데이터베이스 접근 로직을 실행합니다. 예를 들어, 테이블에 행을 삽입하려면 다음과 같은 과정의 작업이 필요할 것입니다.

1. 요구 처리하기
2. 물리적 로그 파일에 기록하기
3. 논리적 로그 파일에 기록하기
4. 수정된 버퍼 캐시를 디스크에 옮기기

OnLine의 구조에서는 이러한 모든 작업들이 도움을 받지 않고 하나의 독립적인 엔진 프로세스에 의해 수행되기 때문에, 부득이하게 동기로 수행되어야 합니다. 게다가, 대칭 다중처리 (SMP) 하드웨어 환경에서 실행될 때는 CPU에 바인드하기 위한 세부사항이 프로세스 수준에서 수행되기 때문에, 항상 하나의 CPU만을 사용해 처리합니다.

새로운 다중 스레드 구조는 앞서 말한 제한을 받지 않으면서도 나중에 설명할 고급 RDBMS 기술을 지원할 수 있는 플랫폼을 생성합니다.

다중쓰레드구조

한 프로세스 내의 다중 스레드는 여러 면에서 UNIX 운영체제 내의 다중 프로세싱과 비슷합니다. 스레드는 다중 프로세싱 운영체제에서 일반적으로 프로세스에 대해 수행되는 커널같은 작업처럼, 생성되거나 분기되고 계획되어야 하고, 프로세스의 내외부로 전환되어야 하고, 파괴되어야 합니다. 따라서 다중 스레드 프로세스를 이해하려면, 우선 단일 스레드 프로세스가 (즉, 전통적인 UNIX 프로세스) 작동하는 방법에 익숙해져야 합니다. 이것을 완전히 알고 나면, OnLine Dynamic Server 시스템의 DBA와 사용자는 새로운 DSA 구조를 완벽하게 사용할 수 있게 됩니다.

다중 스레드는 서로 다른 사용자들이 한 프로세스의 여러 인스턴스를 운영체제 수준에서 분기시킬 필요없이 한 프로세스를 여러 번 반복적으로 실행하는 방법입니다. 대신 여러 "스레드"가 프로세스 수준에서 분기됩니다. 다중 스레드는 여러 클라이언트 프로세스의 요구를 받고 이에 응답하는 하나의 "서버" 프로세스로 실행되는 단순한 프로그램이 아닙니다. 따라서 프로세스는 한 사용자만이 아닌 여러 사용자를 위해 실행될 수 있습니다. 이러한 처리는 다른 운영체제 루틴을 호출하지 않고 전적으로 사용자의 프로세스 내에서 수행됩니다. UNIX에서는 여전히 다른 것과 마찬가지로 단일 프로세스를 실행하고 있는 것으로 인식합니다.

비슷한 예를 한 가지 더 들어보자면, UNIX가 다중 스레드를 수행하는 방법을 생각해 보십시오. 1000개의 프로세스를 실행하면, 주어진 시간에 단 하나의 프로세스만이 UNIX에 의해 실행됩니다. 각 프로세스가 지정된 시간동안 실행되고 나면, 커널이 점유하게 되고 다음으로 계획된 프로세스가 실행됩니다. 실행중인 프로세스를 점유할 때는 나중에 그 프로세스를 다시 시작할 수 있도록 프로세스에 대한 정보를 충분히 저장해야 합니다. 이 정보를 프로세스의 "context"라 하고 "context switching"이란 말도 여기서 나온 것입니다.

한 프로세스의 context는 user level context, register context, system level context 등으로 구성됩니다.

User Level Context는 다음과 같이 구성됩니다.

TEXT 세그먼트:

프로세스에서 어드레스를 지정하여 사용할 수 있는 영역으로, 해당 프로그램을 위한 시스템 명령(machine instruction)이 들어 있습니다.

DATA 세그먼트:

프로그램의 전역 변수와 정적 변수를 저장하는 영역이며, 프로세스는 어드레스로 지정할 수 있다.

STACK 세그먼트:

프로그램의 각 함수가 사용하는 지역 변수를 저장하기 위한 영역이며, 프로세스에서 주소 지정이 가능합니다.

Register Context는 다음과 같이 구성됩니다.

프로그램 카운터(Program Counter)

실행할 다음 명령의 주소를 지정합니다.

스택 포인터(Stack Pointer)

스택 세그먼트에 있는 엔트리의 현재 주소를 담고 있습니다.

범용 레지스터(General Purpose Register)

프로세스가 실행되는 동안 생성되는 데이터를 저장합니다.

System Level Context는 다중 스레드의 개념을 이해하는데 중요하지 않으므로, 여기서는 간단하게 설명하도록 하겠습니다. System Level Context는 프로세스 테이블 엔트리(Process table entry), U 영역, 지역과 페이지 테이블 엔트리(Region and page table entry), 커널 모드 스택 프레임(Kernel mode stack frame), 그리고 예외와 인터럽트를 처리하는 동적 시스템 수준 문맥 계층(Dynamic system level context layer)으로 구성됩니다.

이제, Context Switching의 원리를 알아보도록 하겠습니다. 커널은 현재 실행중인 프로세스의 context를 저장하고 다음으로 계획된 프로세스의 문맥을 로드합니다. 문맥을 로드하는 데는 다음과 같은 과정이 포함됩니다.

1. 프로세스가 최종적으로 점유되었을 때, 프로그램 카운터, 스택 포인터, 그리고 범용 레지스터 (즉, register context) 등을 커널에 의해 저장된 값들로 복원하기
2. 시스템 수준의 지역과 페이지 테이블 엔트리를 통해 물리적 메모리에 있는 텍스트, 데이터, 스택 세그먼트 (즉, user context)를 재대응시키기
3. 나머지 system level context (커널 자체 관리를 위해) 복원하기.

이제 새로운 프로세스를 실행할 준비가 되었습니다.

CPU가 프로세스의 메모리 내용 (사용자 세그먼트)과 변위에 대한 모든 포인터 (스택 포인터, 레지스터 오프셋 등) 가 복원되었다는 것을 인식하고, 프로그램 카운터를 읽어서 텍스트 세그먼트에서 실행될 다음 명령의 주소를 받고, 명령을 로드한 다음 그것을 실행합니다. 커널이 프로세스의 context를 저장하고 복원하는 방법에는 특이한 것이 없습니다. context를 저장할 때는, 레지스터 내용이나, 프로그램 카운터 등이 또 다른 프로세스의 context를 유지하기 위한 영역으로 작용하도록, 미리 할당된 메모리의 데이터 구조로 이것을 복사하는 시스템 명령을 실행합니다. context를 복원할 때는 이 작업이 반대로 실행됩니다.

스레드란 무엇인가?

이제 이러한 개념들을 스레드로 옮겨가겠습니다. 프로그램이 자신의 프로세스의 레지스터 context를 자기자신의 주소 공간 내의 데이터 구조로 복사하고 나서, 실행을 계속하기 위해 이미 저장되어 있던 다른 레지스터 context를 로드한다고 합시다. 위에서 커널 전환에 대해 설명했던 것과 같은 결과가 나오겠지만, 실제로는 훨씬 더 간단합니다. 즉, 같은 프로세스가 계속해서 실행되는 것입니다. 따라서, 사용자 세그멘트와 system level segment는 메모리에 대응되어 유지됩니다. 단지 Register Context만이 변경됩니다. 결국, 프로그램 카운터는 텍스트 세그먼트에 있는 새로운 명령을 가리키고, 스택 포인터는 다른 메모리 영역을 가리키고, 범용 레지스터는 이 문맥을 위해 전에 저장되었던 값들로 복원됩니다. 실제로 실행중인 같은 프로세스 내에서 다른 제어 순서를 실행하는 것은 UNIX를 놀리는 것과 같습니다. 이와 같이 실행중인 프로세스 내의 연속적인 제어 흐름을 "스레드(thread)"라고 합니다.

최근의 UNIX 버전은 대부분 "비중이 적은(lightweight)" 스레드에 대한 기능을 제공합니다. 따라서, 운영체제에서 사용할 수 있는 라이브러리 루틴만을 사용해도 프로세스를 다중 스레드로 설계하는 것이 가능합니다. 그리고 이것으로 인해 프로그래머는 앞 절에서 설명한 다중 스레드 로직을 실행할 수 있습니다.

OnLine Dynamic Server에서 다중 스레드를 구현하기 위해, 접근 방법 (RSAM) 계층뿐 아니라, 기존의 관계형 로직 (SQL) 계층의 하부에 이러한 루틴을 포함하는 새로운 다중 스레드 (MT) 계층이 도입되었습니다. 이 새로운 MT 계층은 MT 호출을 운영체제 호출로 쉽게 대응시킬 수 있도록 해주는 Posix 스레드 표준을 기반으로 하고 있습니다.(기본적으로 운영체제가 Posix 표준을 따르는 비중이 적은 스레드 기능을 제공할 경우)

가상 프로세서(VP)란 무엇인가?

다중 스레드는 OnLine을 구조적으로 크게 변경시켰습니다. 모든 서버 활동을 처리하는 가상 프로세서는 많지 않습니다. 아래 그

림은 VP와 시스템 리소스의 구성을 보여줍니다.

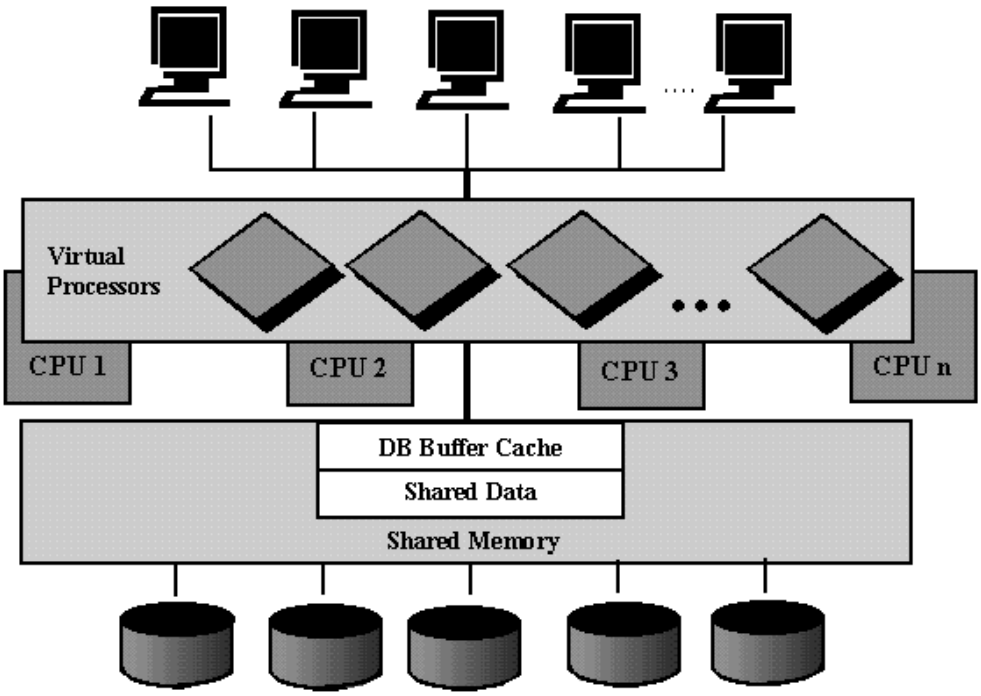


그림 1: OnLine Dynamic Server는 모든 클라이언트 요구에 응답할 수 있는 데이터베이스 서버 프로세스의 결합, 즉 가상 프로세서(Virtual Processor)라는 것으로 구성됩니다.

VP는 독립된 UNIX 프로세스로 ('ps' 목록에는 'oninit'로 나타남), 각각이 여러 스레드를 처리할 수 있습니다. 아래 그림에서 보여주듯이, VP는 클래스들로 분류되어 각 클래스가 서로 다른 엔진 프로세싱 영역을 나타냅니다.

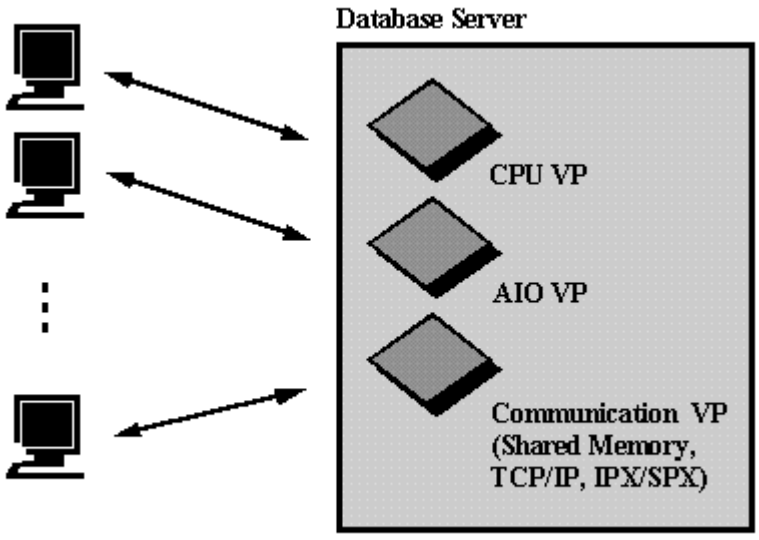


그림 2: 위의 프로세서들은 특정 기능을 위해 최적화된 클래스들로 분류됩니다. 이 그림은 세 가지 주된 클래스를 보여줍니다.

CPU VP는 모든 SQL과 RSAM 처리를 포함해 작업의 대부분을 수행합니다. 따라서, 일반적으로 많은 CPU를 갖고 있는 시스템에는 (NUMCPUVPS의 수가 시스템 상의 CPU 수보다 적어야 함에도 불구하고) 둘 이상의 NUMCPUVPS 프로세서가 있습니다. I/O VP는 우선 순위를 변경해 가면서 비동기로 디스크 I/O를 처리합니다. 운영체제 공급자가 커널 I/O (KIO) 를 제공할 때는 DSA도 비동기 I/O 실행시 KIO를 사용합니다.

클라이언트/서버 메시지 전송 시스템에는 TLI, 소켓(SOC), 그리고 공유 메모리 (SHM) VP가 사용됩니다. 이 메시지 전송 처리 방식중 하나를 사용할 수도 있고, 처리 방식들을 운영체제가 지원하면 세 가지 모두를 조합해서 사용할 수도 있습니다. 사용자가 네트워크 VP의 수와 네트워크 형식을 설정할 수 있습니다. 논리적 로그 I/O VP와 물리적 로그 I/O VP처럼, 사용자가 설정할 수 없는 VP 클래스들도 있습니다.

내장된 연결 기능(Built-in Connectivity)

둘 이상의 통신 처리 방식이 사용되면, 각 클라이언트 프로세스가 환경 변수를 사용해 연결 모드를 지정합니다. SHM 연결이 TLI와 소켓 연결보다 빠르지만, 로컬 연결에만 사용할 수 있으며, 불안전할 수 있습니다. 물론, TLI와 소켓 연결은 원격 접속에 사용할 수 있으며, 로컬 연결도 가능합니다. 이러한 Connectivity는 OnLine Dynamic Server에서 기본적으로 제공하기 때문에, Informix STAR 제품은 더 이상 필요하지 않습니다.

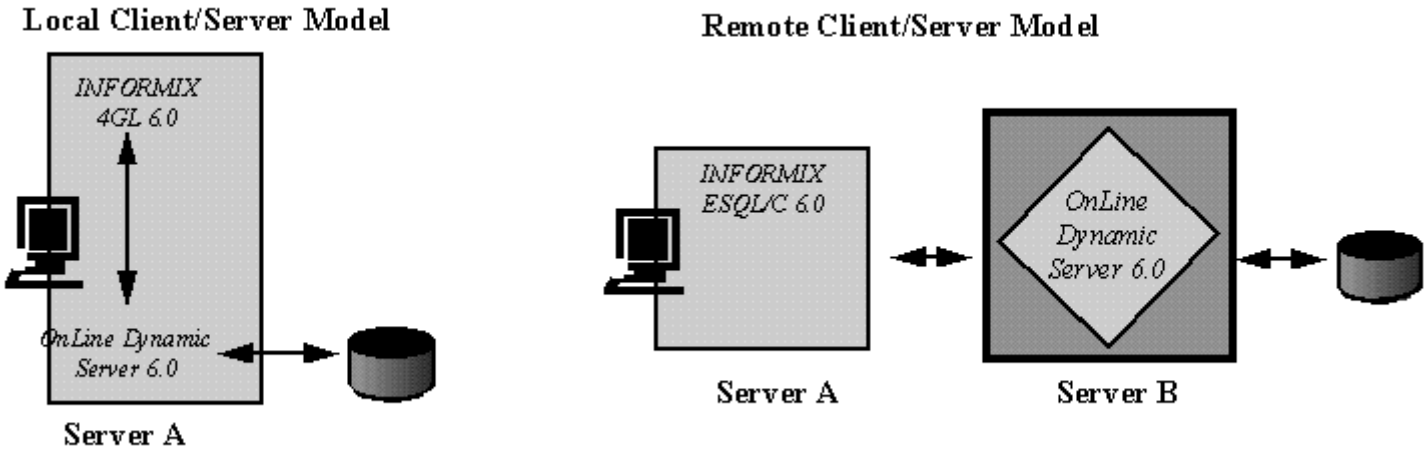


그림 3: OnLine Dynamic Server에는 연결 라이브러리가 내장되어 있습니다. 따라서, 시스템에서 직접 클라이언트/서버 환경을 사용할 수 있습니다.

OnLine Dynamic Server의 성능 이득

OnLine Dynamic Server의 주된 이점 중 하나는 같은 클래스로 된 다중 VP를 가질 수 있다는 것입니다. 이런 설계를 사용하게 되면, 다른 프로세서들은 수행하는 작업이 없이 한가하고 어느 한 프로세서에서는 실행되는 작업이 많아 이 VP에서만 사용자 요구가 병목되는 것을 방지할 수 있으며, 따라서 SMP 환경에서 성능을 크게 향상시킬 수 있습니다. 예를 들어, 아래 그림에 표시한 것처럼, 몇몇의 VP를 하나의 특정 물리적 프로세서로 결합할 수 있기 때문에 하나의 OnLine Dynamic Server 인스턴스가 세 개의 CPU VP를 실행할 수도 있습니다. 이런 환경에서는, CPU VP 중 어느 하나에 로드가 많을 경우, 이 CPU VP의 요구 방향을 이미 요구 처리를 완료한 나머지 두 CPU VP로 바꿔서 시스템의 CPU 처리에 균형을 유지합니다.

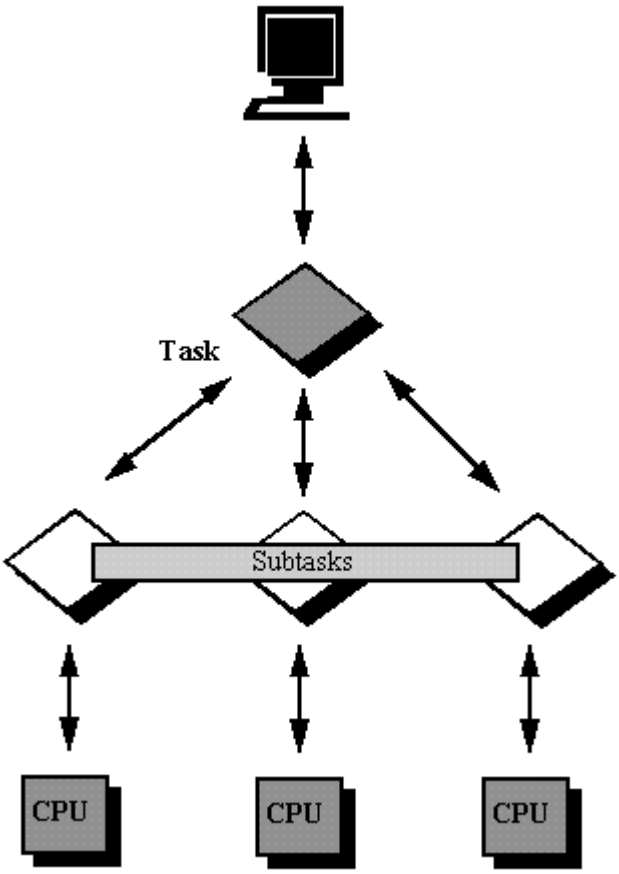


그림 4: OnLine Dynamic Server의 핵심 DSA 기술은 처리할 데이터베이스 요구가 있을 경우, 할당된 모든 프로세서를 일정하게 사용하고, 노는 CPU가 없도록 함으로써 시스템 리소스를 완전히 활용하는 것입니다.

다시 말해, 이러한 시스템 설계에서는 한 VP로부터 같은 클래스에 속하는 다른 VP로 스레드를 "이송(migrate)"할 수 있습니다. 앞에서 우리는 다중 스레드가 한 프로세스에 의해서 유지되고 실행되는 방법을 살펴보았습니다. 스레드 이송은 스레드 context의 중요한 부분(Register Context와 Stack)을 공유 메모리에 유지하여, 모든 VP가 이 정보에 접근할 수 있도록 함으로써 가능합니

다.

결론

OnLine Dynamic Server에는 이 글의 제한된 범위를 넘는 여러 가지 기능들이 있습니다. 이러한 기능으로 인해 OnLine Dynamic Server를 더 많은 분야에 활용할 수 있겠지만, DBA와 Informix 사용자들은 엔진 조정과 관리 기술을 익혀야 합니다.

이 글에서 기본적인 DSA 개념에 대해 설명하면서, 사용자들이 훌륭한 기술을 통합한 이 제품을 완벽하게 사용할 수 있기를 바랍니다.

용어해설

비동기 I/O (AIO: Asynchronous I/O)

비 장애 처리(non-blocking processing)를 요구하지 않는 I/O 호출

동적 확장 구조 (DSA: Dynamic Scalable Architecture)

OLTP 성능을 크게 향상시키면서 더 크고 복잡한 질의에 대한 요구를 처리하도록 설계된 고성능 다중 스레드 구조

동적 공유 메모리 (DSM: Dynamic Shared Memory)

필요한 만큼 동적으로 증가하는 공유 메모리

커널 I/O (KIO: Kernel I/O)

운영체제 커널에 의해 제공되는 비동기 I/O 기능

래치/뮤텍스 (Latches/Mutexes)

공유 리소스를 상호 배타적(mutual exclusion)으로 처리하는 세마포 메카니즘

다중 동시 스레드 (MCT: Multiple Concurrent Thread)

다중 세션 스레드

다중 스레드 (MT: Multithreading)

서로 다른 사용자들이 한 프로세스의 여러 인스턴스를 운영체제 레벨에서 분기시킬 필요없이 프로세스를 여러 번 반복해서 실행하는 방법

온라인 트랜잭션 처리 (OLTP: On-Line Transaction Processing)

현재 작업을 요구한 클라이언트가 어디에서 응답을 기다리고 있는가에 대한 데이터를 지속적으로 읽고 갱신하는 소프트웨어 응용프로그램 환경

병렬 처리 (Parallel Processing)

주어진 작업을 병렬로 계산함으로써 시스템상의 모든 리소스를 효과적으로 사용하도록 하는 기능

Posix 표준 (Posix Standard)

운영체제 공급자들이 통일된 방법을 따르도록 IEEE가 개발한 표준

소켓 (Socket)

네트워크를 통한 통신 기능이 추가된 프로세스간 통신 형식의 소프트웨어 처리 방식

대칭 다중처리 (SMP: Symmetric Multprocessing)

표준 SQL 명령문이 내부 공유 메모리 데이터 구조를 질의하도록 하는 처리 방식

스레드 (Thread)

데이터베이스 서버 프로세스 내에서 불연속적인 작업을 나타내는 하나의 연속된 제어 순서

가상 프로세서 (VP: Virtual Processor)

동적으로 구성할 수 있는 데이터베이스 서버 프로세스 집합

Informix 백업 및 복원 제품 개발 전략

서론

오늘날과 같은 경쟁 사회에서 각 회사들은 개방 시스템상에서 생성되고 지원되는 데이터베이스의 규모를 빠른 속도로 확장해 나가고 있습니다. 관계형 데이터베이스 시스템(Relational Database Management System : RDBMS)의 진보-RDBMS 엔진의 성능 향상, (높은) 가용성의 확장, 그리고 비표준적인 자료형 지원등-로 인해 초 대용량의 데이터베이스(Very Large Database: VLDB)의 지원은 더 이상 불가능한 일이 아닙니다.

그러나 데이터베이스의 크기가 수 백 GByte에서 TByte에 이르게 되면서 데이터를 백업하여 데이터 손상을 막아야 한다는 새로운 문제가 발생하게 되었습니다. 날마다 테이프에 데이터베이스를 백업하는 것만으로는

충분하지 않기 때문에 백업 문제에 있어서 빠르고 쉽고 신뢰성 있는 해결책이 필요합니다. 또한 백업 문제를 해결하는 데에는 자동 교환 장치, 정전 대비, 네트워크와 분산 작업, 파일 압축과 해제, 그리고 데이터 암호화와 암호 해독과 같은 편리하고 특수한 기능을 제공하는 다양한 저장 관리 솔루션을 유연하게 통합할 수 있어야 합니다.

백업 및 복구 유틸리티는 Informix가 데이터베이스 제품을 만들 때 항상 포함되는 부분입니다. Informix는 Informix OnLine Dynamic Server에 ON-Archive와 ON-Tape 등의 최적화된 백업 및 복구 유틸리티를 포함시켰습니다. 이 유틸리티를 사용하면 병렬, 온라인 및 파티션 수준의 백업과 저장, 시간 지정 복구, 그리고 점진적 백업 등의 작업을 할 수 있어 성능 및 가용성을 극대화할 수 있게 됩니다.

증가하는 대용량 데이터베이스를 지원하기 위해 Informix는 ON-Bar라고 하는 새로운 백업과 저장 유틸리티를 채택하였습니다. ON-Bar는 관리 기능, 신뢰성 및 유연성에 대한 확장된 기능을 포함하며 이것은 상당히 진보된 백업 및 저장 유틸리티로 점점 증가되는 회사의 요구를 충족시킬 수 있습니다.

데이터베이스 백업 및 복원이란?

데이터베이스 백업이란 데이터베이스를 안전한 저장 장소에 복사하여 비상 사태가 발생하였을 때 데이터를 복원하여 사용하기 위한 일련의 데이터 복사 절차를 말하며, 복원이란 백업되어 있는 매체로부터 데이터를 가져와 데이터베이스를 전에 있던 상태로 재생성하는 절차를 말합니다. 매일 On-Line 트랜잭션 처리(on-line transaction processing:OLTP)로 업무에 데이터를 사용하는 회사는 데이터베이스 백업이 대단히 중요합니다. 예기치 않은 오류나 데이터의 파손에 대비한 적절한 백업 과정이 없다면 커다란 손해를 입을 수도 있습니다.

데이터베이스 백업

데이터베이스 백업에는 물리적, 논리적 백업이 모두 포함됩니다. 물리적 백업은 디스크에 저장된 객체(자료)의 전부 또는 일부를 제 2 의 저장 장소에 복사하게 됩니다. 이러한 데이터베이스 객체는 단일 또는 다중 DB 영역 및 BLOB 영역에 존재할 수 있습니다.

물리적 백업으로 디스크에 저장된 데이터를 손상되지 않도록 보호할 수 있지만 수정되고 있는 일시적인 데

이터는 보호할 수 없습니다. 수정된 데이터는 디스크에 물리적으로 기록되기 전에 먼저 논리 로그 파일에 저장됩니다. 이 논리 로그 파일은 오류가 발생하면 종료되지 않은 트랜잭션을 전상 회복시키는 데 뿐만 아니라 종료된 모든 트랜잭션을 롤포워드시키는데 사용됩니다. 논리 로그 파일 내의 저장된 파일을 보호하기 위해서는 관리자는 논리적 백업을 수행하여 하나 또는 그 이상의 논리 로그 파일을 제2의 저장 장소에 복사하는 작업을 하게 됩니다.

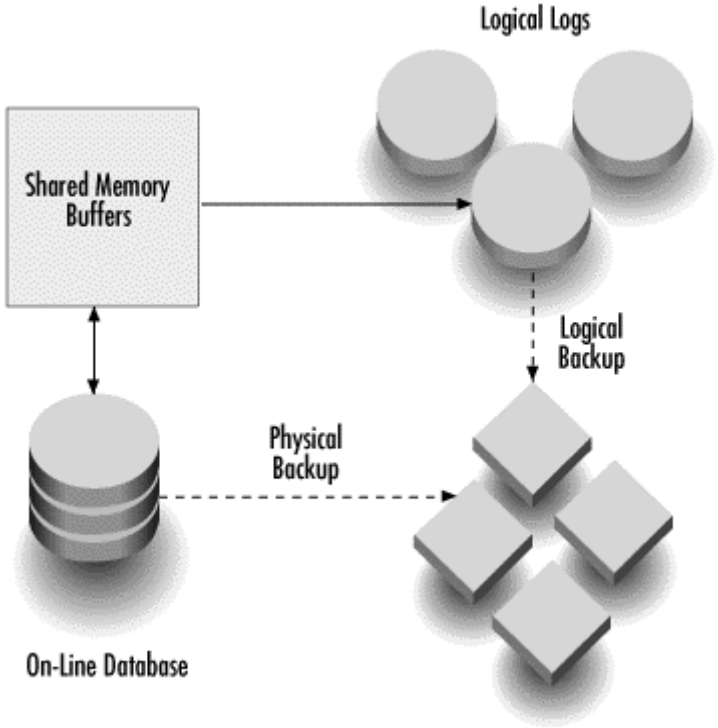


그림1: 데이터베이스 및 논리 로그 백업

오류에 대비하기 위해서 매일 매일 모든 DB 영역을 백업할 수도 있는데 이것을 전체 시스템 백업(full system backup)이라 부릅니다. 하지만 전체 시스템을 백업한다는 것은 데이터베이스의 크기 때문에 오랜 시간이 걸리게 됩니다. 예를 들면 매일 10GB의 데이터베이스를 백업하는 것은 가능할 수도 있지만 그 크기가 500GB에 이르면 시간이 너무 걸려 백업이 불가능하게 됩니다. 때문에 대용량의 데이터베이스를 사용하는 많은 회사들은 전체 시스템을 백업하는 일을 최소화하기 위해 노력하고 있습니다.

대신, 데이터의 손실을 막기 위해 회사들은 전체 시스템 백업의 사이사이에 발생하는 트랜잭션을 논리적 백업으로 복구합니다. 하지만 전체 시스템 백업의 사이의 시간 간격이 늘어나게 되면 이에 따라 대단히 많은 논리 로그 파일이 생기게 됩니다. 복구하는 동안 모든 논리 로그 파일을 적용하는 일은 대단히 많은 시간이

소요됩니다.

전체 시스템 백업을 보다 자주하는 것이 논리 로그 복구 작업을 편하게 할 수 있는 한 가지 방법이 될 수도 있겠지만 Informix는 마지막으로 백업한 이후에 변경된 데이터만을 점진적으로 백업하여 사용할 수 있는 혁신적인 기능을 제공합니다. 점진적 백업(incremental backup)이란 관리자가 데이터베이스를 백업하는데 소비하는 시간을 절약할 수 있게 하는 기능이며 이는 전체 시스템 백업과 같은 수준으로 데이터를 보호할 수 있습니다.

백업될 백업 매체는 오프라인, 즉 Informix OnLine 시스템에서 분리되어 있어야만 덮여써지거나 또는 데이터가 손상되는 일이 없이 보관할 수 있습니다.

데이터베이스 복원

복원이란 데이터를 백업 매체로부터 작업할 수 있는 상태로 재생성하는 일련의 절차를 말합니다. 백업 매체로부터 데이터를 복사해 옴으로써 데이터베이스가 재생성됩니다. 데이터베이스 복원은 일반적으로 두 가지의 형태로 분류됩니다. 첫째는 마지막 물리적 백업으로 생성된 제2의 저장 매체로부터 데이터를 복사해옴으로써 손실되거나 변경된 데이터베이스 객체를 복원하는 물리적 복원이고, 둘째는 논리적 백업 저장 매체를 사용하여 마지막 백업 이후 데이터베이스에 적용된 데이터베이스 트랜잭션을 재생성하는 논리적 복원입니다.

물리적 및 논리적 복원은 오류가 발생한 시간과 비슷한 시간에 데이터베이스의 복사본을 만들게 됩니다. 그렇지만 가장 최근에 발생한 데이터베이스 트랜잭션을 논리 로그 파일에 반영하지는 않으며 백업되지도 않습니다. 그렇기 때문에 좀더 최근의 작업 내용으로 데이터베이스를 만들려면 논리 로그 파일에 저장된 트랜잭션도 데이터베이스에 적용되어야 합니다. 예를 들어 일요일 오후 10시에 전체 시스템 백업을 하고, 매일 오후 10시에는 논리 로그 백업이 이루어 지는데, 수요일 아침 11:00에 시스템에 데이터베이스 오류가 발생하였다고 가정하면, 이 데이터베이스를 복원하기 위해, 먼저 일요일 밤에 만들어 놓은 전체 시스템 백업으로 저장한 매체를 이용하여 백업받은 일요일까지의 데이터베이스를 복원합니다. 그리고 나서 월요일과 화요일에 만들어진 논리 로그 백업을 화요일 10시까지 발생한 트랜잭션을 롤포워드시키기 위해 사용하고, 마지막으로 아직 디스크에 남아 있는 논리 로그 파일을 사용하여 화요일 오후 10시부터 수요일 아침 10시 사이에

발생한 트랜잭션을 롤포워드시킵니다.

Informix 백업 및 복원 : ON-Bar

Informix는 차세대 백업 및 복원 유틸리티인 ON-Bar를 현대의 데이터베이스의 요구에 맞춰 특별하게 개발해 왔습니다. ON-Bar는 Informix의 첨단 데이터베이스 아키텍처, 즉 Dynamic Scalable Architecture TM(동적 확장 구조:DSA)의 기능을 이용하여 메인프레임 급의 백업과 복원 능력을 개방 시스템에도 사용할 수 있도록 해줍니다. 어떠한 크기의 데이터베이스도 이상적으로 처리할 수 있는 ON-Bar를 사용하면 다음과 같은 중요한 이점을 얻을 수 있습니다.

- 고성능
- 높은 가용성
- 용이한 관리
- 신뢰성
- 유연성

고성능

시스템 관리자가 백업과 복원을 자주 수행한다는 것을 최대한 고려한 ON-Bar에는 DSA가 제공한 병렬 처리 고유의 모든 이점을 취하여 가능한 최고의 성능을 갖게 하였습니다. 병렬 백업과 복원은 이용 가능한 모든 CPU와 장치에게 해야 하는 작업을 나누어 분산시킴으로써 밀접하게 연관된 아키텍처와 느슨하게 연관된 아키텍처의 모든 처리 능력을 활용합니다. 이렇게 ON-Bar는 데이터의 읽기와 쓰기를 효과적으로 병렬 처리하여 백업과 복원의 처리 속도를 현저히 높입니다.

보다 확장된 기능으로는 시간 지정 복원 및 점진적 백업이 있습니다. 시간 지정 복원 기능은 관리자가 특정 날짜와 시간에 데이터베이스를 복원할 수 있도록 하며, 점진적 백업은 마지막으로 백업된 이후에 수정된 데이터만을 백업할 수 있도록 합니다. 이러한 기능들을 결합하여 ON-Bar는 다른 데이터베이스와는 비교할 수 없는 뛰어난 성능과 확장성을 나타낼 수 있습니다.

높은 가용성

ON-Bar에는 데이터베이스를 직접 가져와서 다루지 않고도 백업과 복원을 할 수 있는 여러가지 기능이 있습니다. 이러한 기능으로는 온라인 백업과 복원, 그리고 파티션 수준 백업과 복원이 있습니다. 이러한 기능을 결합함으로써 일상적인 또는 갑작스런 시스템의 정지 시간을 현저히 최소화할 수 있습니다.

현재의 많은 산업은 24x7 기능을 요구하기 때문에 데이터베이스 백업과 같은 관리 기능은 데이터베이스를 직접 가져오지 않고도 작업을 수행할 수 있어야 합니다. ON-Bar를 사용하면 온라인 백업을 할 수 있는데 이 온라인 백업으로 데이터베이스의 실행이 계속되고 있을 동안에도 전체, 또는 일부의 데이터베이스를 백업할 수 있습니다. ON-Bar를 사용하면 온라인 복원 또한 가능합니다. 데이터베이스 서버가 온라인으로 연결되어 있으면 손상을 받지 않은 데이터베이스 객체는 완전히 복구될 수 있습니다. 온라인 복원을 이용하면 데이터베이스의 섹션이 복구되는 동안에도 데이터베이스를 계속 사용할 수 있습니다. ON-Bar는 또한 파티션 수준 백업과 복원을 지원합니다. 파티션 작업은 테이블을 여러 디스크에 분산시켜 마치 작은 테이블을 모아 놓은 것처럼 보이게 합니다. 테이블을 파티션하면 한 번에 한 파티션에 대해 백업과 복원 작업이 이루어지므로 테이블의 다른 부분은 사용자가 계속 이용할 수 있습니다.

신뢰성

데이터베이스 백업의 중요한 목적은 데이터가 손상되는 것을 막는데 있습니다. 데이터를 복사한 백업본이 데이터베이스 시스템을 완전히 복원하는데 사용될 수 없다면 백업 작업은 아무런 소용이 없는 일입니다. 그래서 신뢰성은 아마도 백업 및 복원 유틸리티를 선택할 때 가장 중요한 요소가 될 것입니다.

ON-Bar는 데이터를 신뢰성 있게 백업하고 복원하는 데 사용할 수 있는 간단한 명령들을 제공합니다. 이 명령들은 사용하고 이해하기 쉬우며 백업 및 복원 기능을 실행하는데 관련한 모든 혼란을 없애주게 될 것입니다. 예를 들면, 다음의 명령은 데이터베이스 인스턴스와 관계된 모든 DB 영역과 BLOB 영역에 대한 백업 작업을 수행합니다.

```
onbar -b
```

아래의 명령은 DB 영역인 yyy에 대해 물리적 복원을 수행합니다.

onbar -r -p yyy

각각의 작업이 끝나면 관리자가 작업의 상황을 알 수 있도록 메시지 로그에 엔트리를 기록합니다.

ON-Bar는 ON-Bar 프로그램 구성요소들 사이의 호환성을 유지하기 위해서 뿐만 아니라 데이터베이스 인스턴스와 백업 객체 등과 같은 여러가지 백업 정보를 지속적으로 유지하기 위한 카탈로그 테이블을 제공합니다. 또한 ON-Bar는 중요한 DB 영역과 논리 로그를 포함한 비상용 부트 파일을 준비하여 그 안에 카탈로그의 요약 정보를 저장합니다. 이 비상용 부트 파일은 데이터베이스를 온라인 상으로 가져오는데 필요한 사항을 포함하고 있기 때문에 오류가 발생하였을 때 모든 데이터베이스 객체를 복원하고자 할 때에 이용할 수 있습니다.

ON-Bar는 자동으로 계속해서 논리적 로그를 백업하기 위한 환경을 설정할 수 있으므로 가장 최근에 데이터베이스에 가해진 변화를 확실하게 백업할 수 있습니다. 논리 로그 파일이 채워지는 즉시 ON-Bar는 자동으로 로그를 백업하며 채워진 다음 로그 파일에 대해서도 계속 이러한 작업을 반복합니다. 따라서, 오류가 발생해도 데이터베이스의 손실을 최소로 하여 복구할 수 있습니다.

용이한 관리

ON-Bar는 설치하기가 쉽고 일련의 기본값으로 사전에 환경 설정이 되어 있습니다. 특별한 백업이나 복원을 하게 되는 업무에서는 환경 설정을 다시 맞추어 사용할 수 있습니다. ON-Bar는 또한 처음의 시도가 실패했을 경우 백업 기능을 자동으로 다시 실행하는 것 등의 여러가지의 오류 처리 능력이 있습니다.

ON-Bar에는 백업 및 복원 세션을 시행하는 여러가지 방법이 있습니다. 세션은 명령행이나 Informix Enterprise Command Center(IECC)와 같은 다양한 Informix 관리 그래픽 사용자 환경(graphical user interfaces:GUI)을 통하여 실행될 수 있으며, 또는 다른 회사에서 제작한 저장 관리자(Storage Manager)를 통해 실행될 수도 있습니다. 가장 익숙한 인터페이스를 선택하여 사용한다면 관리하기 쉬워지고 그 유용성도 현저히 증가될 것입니다.

일반적인 환경에서 수행되는 백업을 확실하게 하기 위해 운영 체제와 시스템 관리 환경에 의해 제공된 여러 가지 스케쥴러를 사용하여 ON-Bar 백업 작업의 일정을 조정할 수 있으며 사람의 개입을 최소화 하거나 또는 전혀 없이도 가능합니다. 일단 백업이 완료되면, 다양한 저장 관리자가 제공하는 자동 테이프 라벨링 기능을 사용하여 백업 매체가 정확히 보관될 수 있도록 합니다.

유연성

백업과 복원할 데이터의 양이 많아지면서 저장 관리자의 시장 규모가 커지고 있습니다. 이들 회사의 저장 관리자는 보안성이 좋고, 저장 공간을 줄이고, 작업자의 개입을 최소화한 한발 앞선 기능은 물론 성능을 향상시키기 위한 복잡한 저장 장치도 지원합니다.

하지만 이러한 타 회사의 저장 관리자를 사용하여 데이터베이스를 저장하려면 데이터를 데이터베이스에서 운영 체제 파일에 복사한 후 저장 관리자를 사용하여 파일을 백업해야 합니다. 이 방법은 시간과 메모리를 낭비하며 복잡하고 자동화하기도 쉽지 않습니다.

ON-Bar는 타 회사의 저장 관리자를 통합한 산업 표준 애플리케이션 프로그래밍 인터페이스(application programming interface:API)를 준수하여 이러한 문제를 해결합니다. 이 API는 데이터베이스가 저장 관리자에 직접 접근하도록 하여 운영 체제 파일을 백업하는 것과 같은 방법으로 데이터베이스를 백업합니다. ON-Bar를 사용하면 사용자는 선택할 수 있는 범위가 넓어져 자신의 기업에서 필요로 하는 백업 체제에 가장 잘 맞는 저장 관리자를 유연하게 선택할 수 있습니다.

성능은 뛰어나지만 복잡한 다른 저장 관리자를 사용할 필요가 없는 회사를 위해 ON-Bar에는 자체적인 저장 관리자가 포함되어 있습니다. Informix 저장 관리자를 사용하면 하드웨어 제조업체에서 제공하는 원래의 테이프나 디스크 드라이브를 사용하여 지역 데이터베이스를 백업할 수 있습니다.

On-Bar 아키텍처

ON-Bar 백업 및 복원 시스템은 다음 3개의 주요 구성요소로 구성되어 있습니다

- onbar 프로그램

- X/BSA 인터페이스
- 저장 관리자

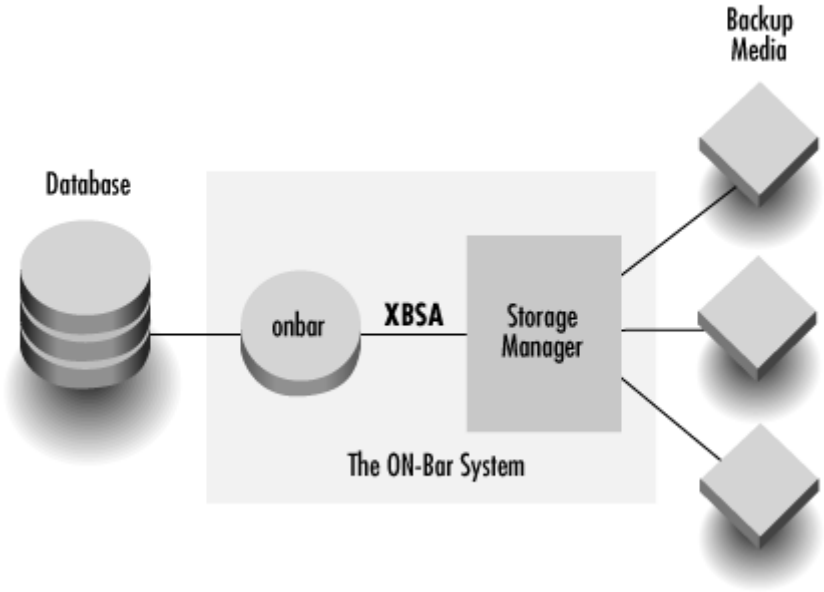


그림 2: ON-Bar의 백업 및 복원 시스템

Informix에서는 ON-Bar 백업 및 복원 시스템에 세 개의 구성요소를 제공하지만, 다른 회사의 저장 관리자와 그와 연관된 X/Open Backup Services API (XBSA) 인터페이스를 구하여 사용할 수도 있습니다. 이에 대해서는 이 절의 뒷부분에 자세히 설명되어 있습니다.

또 다른 두 구성요소인 카탈로그 테이블과 비상용 부트 파일도 설명됩니다.

onbar 프로그램

onbar 프로그램의 역할은 백업 및 복원을 실행하는 것입니다. 백업 및 복원을 실행 요청이 들어오면 onbar 프로그램은 OnLine 데이터베이스와 저장 관리자를 모두 사용하여 세션을 시작합니다. 백업의 경우, onbar 프로그램은 데이터베이스 서버에서 DB 영역, BLOB 영역 및 논리 로그 파일을 검색하여 저장 관리자로 보냅니다. 복원의 경우, onbar 프로그램은 저장 관리자로부터 DB 영역, BLOB 영역 및 논리 로그 파일을 요청해 OnLine 데이터베이스 서버로 보냅니다.

Onbar 프로그램은 UNIX 명령행, 저장 관리자에서 또는 Informix Enterprise Command Center 같은 통합

시스템 관리 환경 등 다양한 소스로부터 백업 및 복원을 요청할 수 있습니다.

X/Open Backup Service API(XBAS)

onbar 프로그램은 XBSA를 사용하여 저장 관리자와 정보를 교환합니다. XBSA는 데이터베이스와 저장 관리자간의 백업 및 복원에 대한 정보를 교환하기 위해 개발된 산업 표준 API입니다. Oracle등과 같은 다른 데이터베이스 개발사는 백업 유틸리티에 자신만의 독자적인 독점 인터페이스를 만들어 다른 저장 관리자를 통합합니다. 이러한 통합 형태는 표준화 되지 않았을 뿐 아니라 실행 시간이 많이 들게 됩니다.

onbar 프로그램은 XBSA 규정을 준수하므로 XBSA 인터페이스를 제공하는 수 많은 저장 관리자와 함께 사용될 수 있습니다. 이들 저장 관리자는 ON-Bar 백업 및 복원 시스템에 번들로 들어있는 Informix 저장 관리자일 수도 있고 다른 저장 관리자 개발사(SMV)의 제품일 수도 있습니다. XBSA 표준을 충실히 따르기 때문에 Informix의 고객은 빠르며, 믿을 수 있고, "plug-and-play" 백업이 가능한 첨단 저장 관리자의 추가된 기능과 성능을 이용할 수 있습니다.

저장 관리자(Storage Manager)

저장 관리자란 저장 장치와 백업 및 복원에 사용되는 매체를 관리하는 응용프로그램을 말합니다. 백업이 이루어 지는 동안에 저장 관리자는 데이터베이스 객체를 XBSA 인터페이스를 통하여 받으며 저장 장치에 이 객체를 기록합니다. 복원하는 경우, 저장 관리자는 복원이 요구된 데이터베이스 객체를 검색하여 이를 XBSA 인터페이스를 통해 onbar 프로그램으로 보냅니다.

IBM의 ADSM, Hewlett-Packard사의 OpenView OmniBack II, 또는 Legato의 NetWorker와 같은 다른 회사의 제품을 저장 관리자로 사용할 수도 있습니다. 다른 회사의 제품을 사용하면 많은 장점이 있습니다. 간단한 테이프 및 디스크에서 stacker, 로보트, jukebox와 같은 자동 교환기에 이르기까지 다양한 저장 매체를 제공하며 대부분 스케쥴러가 포함되어 있어 백업 작업을 쉽게 자동화할 수 있으므로 작업자의 개입이 줄어들게 됩니다.

높은 수준의 기능이 필요하지 않은 사용자라면 Informix 저장 관리자(ISM)를 사용하여 일반적인 디스크와 테이프 장치에 데이터를 백업할 수 있습니다. 업무 환경과는 무관하게 ON-Bar를 사용하면 고객의 필요에

가장 잘 맞는 저장 관리자를 유연하게 선택할 수 있습니다.

ON-Bar 카탈로그 테이블

sysutil 데이터베이스는 OnLine 데이터베이스 서버가 실행되는 과정을 기록하는 Informix 카탈로그 데이터베이스입니다. sysutil 데이터베이스 안의 ON-Bar에 관련된 테이블에는 데이터베이스 서버, 데이터베이스 객체 및 백업, 복원이 이루어지는 동안 취해진 행동이 기록됩니다. 각 요소들의 버전간 호환성을 확보하기 위해 onbar 프로그램, XBSA 및 저장 관리자의 버전이 테이블에 기록됩니다.

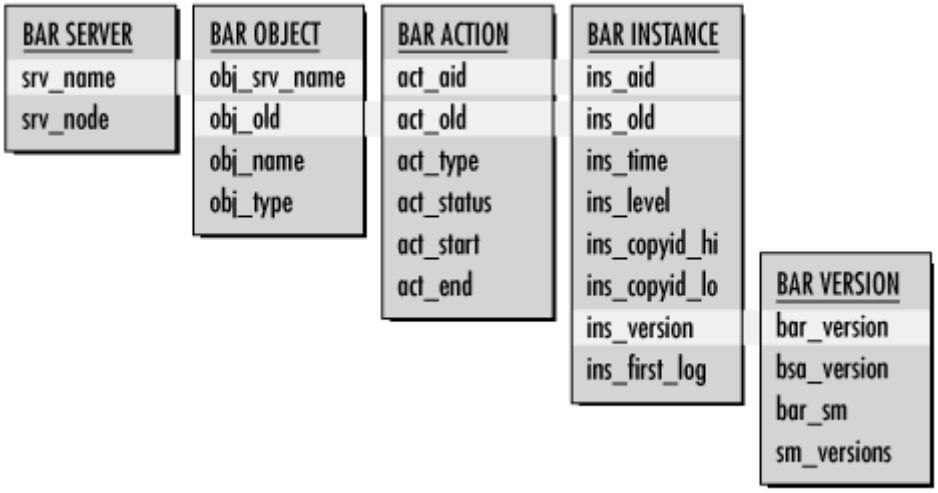


그림 3: ON-Bar 카탈로그 테이블

비상용 부트 파일

앞에서 설명된 카탈로그 테이블은 ON-Bar 세션이 호출될 때 접근되어 백업 활동 내역에 대한 관련 정보를 제공합니다. 그렇지만 오프라인으로 복원해야 하는 경우에는 시스템이 오프라인 상태이기 때문에 sysutil 테이블을 사용할 수 없습니다. 따라서 오프라인 복원시에는 sysutil 테이블의 대용으로 부트 파일이 제공됩니다.

비상용 부트 파일에는 루트 DB 영역과 같은 중요한 DB 영역을 포함하여 데이터베이스에 있는 모든 데이터베이스 객체에 대한 정보가 들어 있습니다. 전체 복원을 하는 동안 데이터베이스를 온라인 모드로 만들기 위해, ON-Bar는 비상용 시동 파일 정보를 사용하여 저장 관리자로부터 올바른 백업 객체를 요청합니다. 그런 다음 남아있는 DB 영역을 온라인으로 만들기 위해 비상용 부트 파일이 사용됩니다.

ON-Bar의 동작 방법

앞에서 설명한 바와 같이 ON-Bar 시스템의 백업과 복원 처리는 두 단계로 나뉘어 있습니다. onbar 프로그램에 의해 수행되는 첫번째 단계는 OnLine 서버와 연결되어 데이터를 추출하거나 복원하는 작업과 관련된 것입니다. 저장 관리자에 의해 수행되는 두번째 단계는 저장 매체와 연결되어 데이터를 가져오거나 기록하는 작업과 관련된 것입니다. onbar 프로그램과 저장 관리자는 XBSA 인터페이스를 통해 정보를 교환합니다.

ON-Bar는 호출될 때 포함될 객체의 수를 결정합니다. 복수의 객체가 있다면 각각의 객체에 대해 ON-Bar 인스턴스가 만들어 집니다. 만들어진 인스턴스는 병렬 처리를 위해 시스템의 사용가능한 모든 프로세서들에 골고루 나누어 집니다.

각각의 ON-Bar 인스턴스는 새로운 XBSA 세션을 만들어 저장 관리자와 데이터를 교환하는데 사용됩니다. ON-Bar 인스턴스와 XBSA 세션 사이의 데이터는 공유 메모리를 통해 교환됩니다.

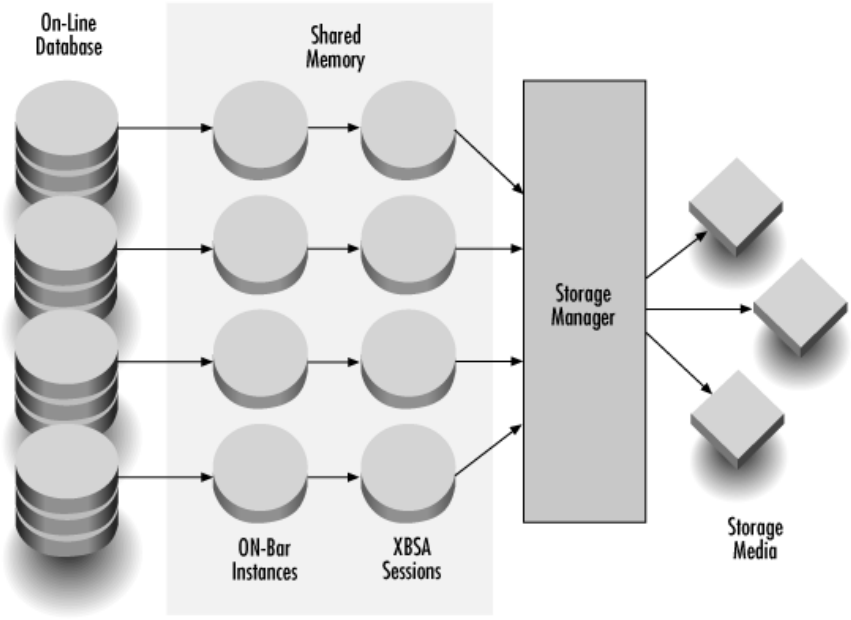


그림 4: 병렬 백업

백업의 경우 저장 관리자는 데이터를 XBSA 세션에 의해 보내진 데이터를 저장 장치에 기록합니다. 복원의 경우는 저장 관리자가 저장 장치에서 데이터를 검색하여 검색된 데이터를 XBSA 세션에 보냅니다. 일부 다

른 회사의 저장 관리자는 동시성과 공간 효율을 향상시키기 위해 여러 장치에 데이터를 지능적으로 분산시키는 기능을 제공하는데, 이러한 기능은 백업 및 복원 작업 성능을 훨씬 향상시켜 줍니다.

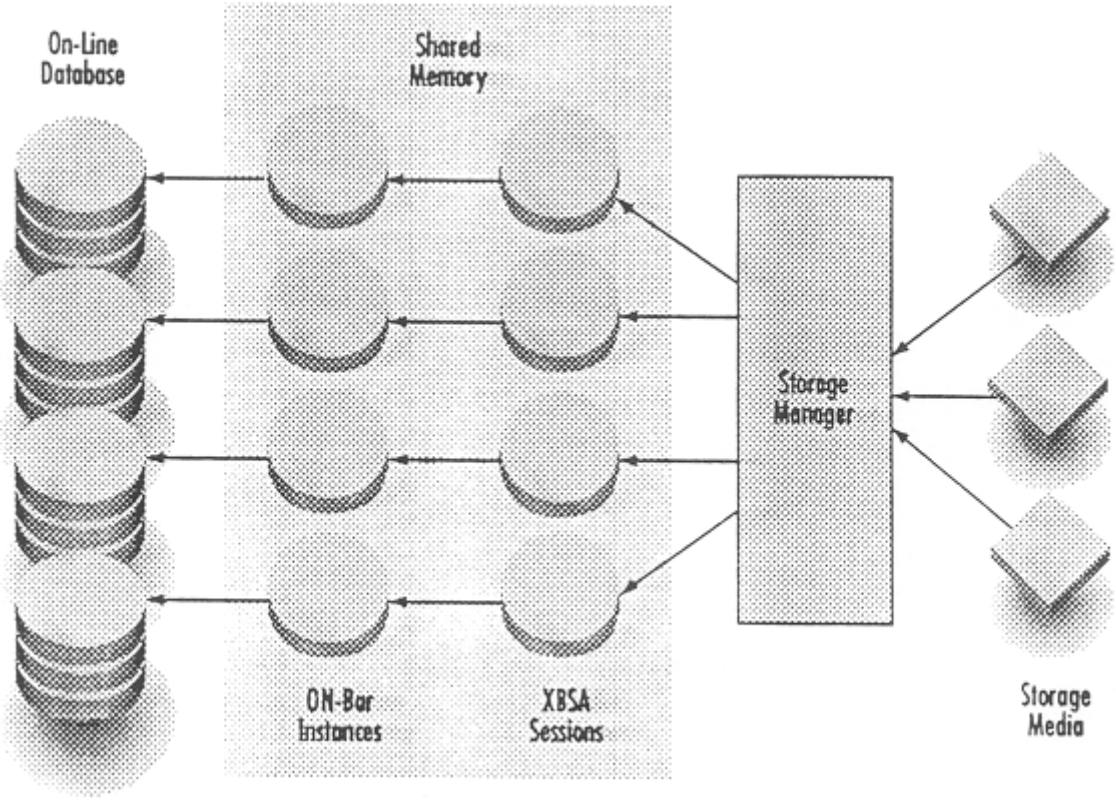


그림 5: 병렬 복원

ON-Bar : 기술상의 특징

- ON-Bar의 주요 기능
- ON-Bar의 실행

ON-Bar의 주요 기능

데이터베이스 산업이 급격히 성장함에 따라 병렬 처리 작업의 수행 능력이 백업 및 복원 유틸리티의 중요한 관건이 되었습니다. 앞 절에서 설명된 바와 같이 ON-Bar는 DSA의 내부 데이터베이스 병렬 처리를 이용하여 진정한 의미의 병렬 백업 및 병렬 복원 능력을 가지고 있습니다.

ON-Bar는 병렬 백업과 복원 뿐 아니라 업무상의 모든 요구를 처리하기 위해 다음과 같이 폭넓은 기능을

갖고 있습니다.

- 온라인 백업 및 복원
- 파티션 수준 백업 및 복원
- 점진적인 백업
- 시간 지정 복구

온라인 백업 및 복원

ON-Bar를 사용하면 데이터베이스를 가져오지 않고도 데이터베이스를 백업하거나 복원할 수 있습니다. 온라인 백업 명령을 실행하는 데에는 단지 백업될 데이터베이스 객체를 지정하고 백업 명령을 실행하기만 하면 됩니다. 예를 들어, 데이터베이스가 온라인 상태인 동안에도 다음 명령을 실행할 수 있습니다.

```
onbar -b dbspace1 blobspace1
```

데이터베이스 복원은 온라인, 오프라인 또는 이 둘이 혼합된 상태에서도 가능합니다. 선택 복원이라고도 하는 온라인 복원을 사용하면 데이터베이스가 온라인 상태인 경우에도 복원을 할 수 있어, 비교적 덜 중요한 DB 영역(시스템 정보가 아닌 일반 데이터만 들어 있는)이나 BLOB 영역을 복원할 때 사용됩니다. 전체 복원이라고도 하는 오프라인 복원은 복원될 데이터베이스가 오프라인 상태에 있어야 합니다. 오프라인 복원은 전체 시스템을 복원하거나 또는 루트 DB 영역과 같은 중요한 DB 영역을 복원하여야 할 때 사용됩니다.

혼합 복원은 오프라인 복원을 한 후에 온라인 복원합니다. 먼저 중요한 DB 영역(루트 영역이나 로그가 들어 있는 영역)을 오프라인으로 복원하여 데이터베이스를 온라인으로 가져올 수 있도록 합니다. 일단 데이터베이스가 온라인 상태로 되면, 즉시 데이터베이스에 로그온하여 데이터베이스의 물리적 및 논리적으로 확실히 복원된 부분에 접근할 수 있습니다. 남아있는 비교적 덜 중요한 DB 영역 및 BLOB 영역은 데이터베이스가 온라인 상태에 있는 동안에 복원할 수 있습니다.

파티션 수준 백업 및 복원

데이터 파티션은 지능적으로 대형 테이블을 작은 부분으로 나누어 여러 개의 디스크에 분산시켜 데이터베이

스 병렬 처리를 가능하게 하는 중요한 요소 중의 하나입니다. 대형 테이블을 더 잘게 나누어 여러 개의 디스크에 분산시킴으로써 데이터베이스의 조작을 여러 개의 CPU와 디스크에서 병렬로 나누어 처리할 수 있기 때문에 성능과 가용성이 더욱 증가됩니다.

데이터베이스 백업과 복원은 파티션 수준에서 수행될 수 있습니다. 테이블이 적당히 분할되면 다중 파티션을 병렬로 백업하고 복원할 수 있으므로 작업의 속도가 빨라집니다. 그림 4에 이러한 접근 방법을 설명하였습니다.

ON-Bar가 온라인 백업을 지원하지만 데이터베이스 관리자는 더 나은 성능을 위해 오프라인 백업을 선택할 수도 있습니다. 그러나 오프라인 백업은 빠르기는 하지만 자주 실행할 수가 없습니다. 24x7 환경에서는 더욱 실행할 수가 없습니다. 파티션 수준 백업은 데이터베이스의 한 부분은 백업을 위해 오프라인 상태로 유지하고 나머지 부분은, 트랜잭션 처리를 위해 온라인 상태로 유지하면서도 두 상태 모두 최상으로 처리하므로 높은 가용성과 성능을 함께 제공합니다.

그림 6은 파티션의 일부를 사용자가 이용할 수 있도록 남겨놓고 백업하기 위해 테이블의 파티션을 오프라인으로 가져오는 방법을 나타낸 것입니다.

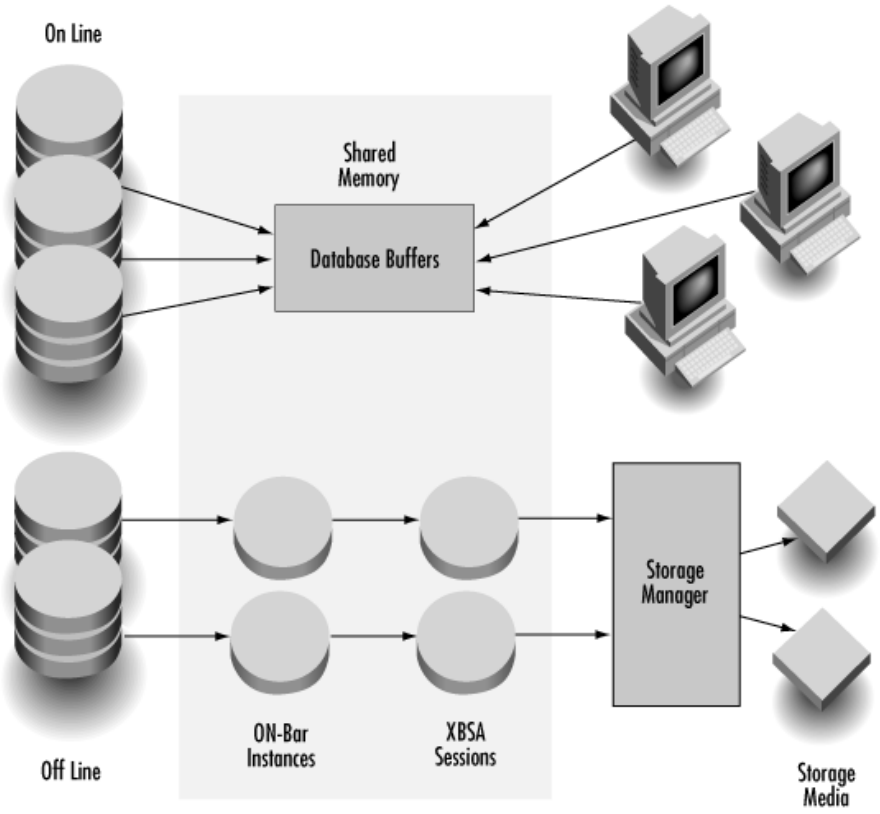


그림 6: 파티션 수준 백업 이용으로 확장된 가용성

이와 유사하게 파티션 수준 복원에 있어서도 데이터베이스를 가능한 잘게 나누어 복원함으로써 가용성을 현저히 향상시킬 수 있습니다. 예를 들어, 데이터베이스가 5개의 디스크에 나뉘어져 있고 4번 디스크에 오류가 발생했다면 디스크 전체를 백업할 필요는 없을 것입니다. 단지 4번 디스크의 데이터만을 복원하면 되는 것입니다.

또한 파티션 수준 복원은 전체 시스템 복원의 가용성도 향상시킵니다. 예를 들어, 서버의 고장으로 데이터베이스가 손상되었다면 백업 매체로부터 복원해야 합니다. 이러한 경우에 관리자는 루트 DB 영역과 같은 모든 중요한 DB 영역을 오프라인으로 복구할 것입니다. 그 다음에는 남아 있는 테이블을 복구하기 위해 데이터베이스를 온라인 상태로 만듭니다. 만일 테이블이 여러 개의 저장 장치에 나뉘어 있다면 병렬로 복원하여 처리 속도를 높일 수 있습니다. 일단 파티션이 물리적 및 논리적으로 확실히 복원되면 남아있는 파티션이 복구되는 동안 분할된 데이터는 트랜잭션 처리에 즉시 사용될 수 있습니다.

점진적 백업(Incremental Backup)

각각의 데이터베이스 페이지에는 date/time 스템프가 있습니다. 페이지가 변경된 시간이 기록되어 있는 이 date/time 스템프를 사용하면 관리자는 파티션이나 테이블 전체를 백업하는 일 없이 변경된 페이지만을 백업할 수 있습니다. 이를 점진적 백업이라고 합니다. 비교적 내용 변경이 적은 대용량 테이블인 경우에는 점진적 백업을 이용하면 시간이 현저히 단축됩니다.

백업 세션동안 기록된 날짜는 ON-Bar가 각각의 페이지를 검사하는데 사용합니다. 만일 페이지에 포함되어 있는 시간이 마지막 백업된 시간보다 크면 이 페이지는 저장 매체에 기록됩니다. 그렇지 않다면 이 페이지를 건너 뛰어 다음 페이지에 대한 작업을 계속합니다. 예를 들어, 그림 7에서는 마지막 백업한 날짜가 11/14/96입니다. 11/20/96에 백업을 하면 ON-Bar는 date 스템프를 보고 11/14/96보다 큰 날짜(예를 들면 11/17/96이나 11/19/96)인 페이지는 복사하며 date 스템프가 11/14/96보다 작거나 같은 날짜(예를 들어, 11/09/96)인 페이지는 건너 뜁니다.

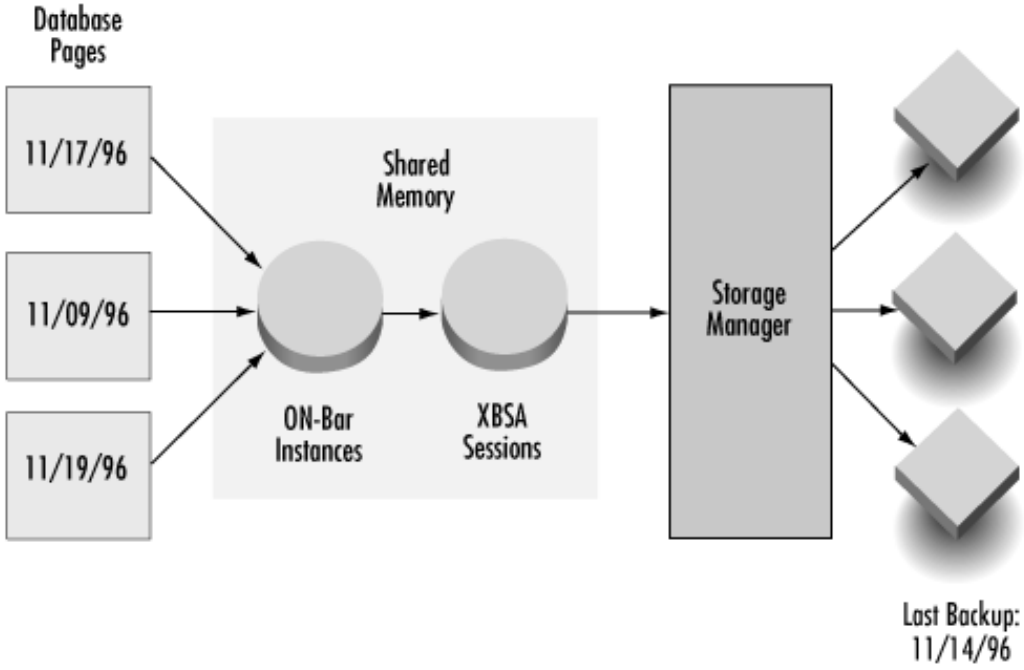


그림 7: 점진적 백업

시간 지정 복구

ON-Bar에는 전체 OnLine 설치를 특정 시간에 복원할 수 있도록 하는 기능이 있습니다. 이 기능을 시간 지정(point-in-time:PIT) 복구라고 합니다.

PIT 복구는 응용프로그램의 논리 오류로 인해 OnLine 인스턴스 내에 오류가 발생 하였을 때 특히 유용합니다. PIT 복구를 사용하면 응용프로그램 에러가 발생하기 바로 직전에 데이터를 안전한 상태로 가져 올 수 있습니다.

PIT 복구를 호출하기 위해 관리자는 복원을 하는 동안 날짜와 시간을 기록합니다. ON-Bar는 이 매개변수를 사용하여 최초의 트랜잭션을 찾을 때(경계 레코드가 특정 날짜와 시간보다 클 때)까지 레코드를 처리합니다. 위치를 찾으면 ON-Bar는 레코드 처리를 중지하고 급속 복구를 시작합니다. 급속 복구는 사용자가 지정한 날짜와 시간 이전의 완료되지 않은 레코드를 모두 전상 회복시킵니다.

예를 들어, 가장 최근의 물리적 백업이 11/10/96에 있었고 물리적 백업 이후 매일 매일 논리적 백업을 하였으며, 11/15/96에 응용프로그램의 오류가 데이터베이스 인스턴스에 손상을 입혔으면 관리자는 이 오류가 11/14/96에서 11/15/96 사이에 발생하였다고 계산할 수 있습니다. 그렇다면 다음의 명령으로 11/13/96까지의 데이터베이스를 복원할 수 있습니다.

```
onbar -r -w -t "13-Nov-1996 00:00:00"
```

위 명령은 물리적 백업 저장 매체를 사용하여 11/10/96까지의 데이터베이스를 복원하고 논리적 백업 저장 매체를 사용하여 11/13/96 이전에 발생한 완료되지 않은 트랜잭션을 모두 롤포워드합니다.

ON-Bar의 실행

ON-Bar는 여러 다양한 인터페이스를 통해 실행될 수 있습니다. 가장 일반적으로 사용되는 인터페이스는 다음과 같습니다.

- 명령행 인터페이스
- Informix Enterprise Command Center
- 그밖의 저장 관리자 GUI

명령행 인터페이스

운영 체제의 명령행에서 직접 ON-Bar를 실행할 수 있습니다. ON-Bar에는 DB 영역, BLOB 영역 및 논리 로그의 백업과 데이터의 복원을 위해 간단히 사용할 수 있는 명령이 있습니다. 다음은 명령행 인터페이스에서 ON-Bar를 실행하는 몇 가지 예를 나타낸 것입니다.

Command Line Example	Functionality
`onbar -b dbs_central_tbls dbs_lookup`	Invokes ON-Bar to back up dbspaces.
`onbar -l`	Invokes ON-Bar to back up all full logical log files.
`onbar -r dbspace`	Invokes ON-Bar for restoring a dbspace.

표 1: 명령행에서 ON-Bar를 사용하는 예

Informix Enterprise Command Center(IECC)

Informix는 데이터베이스 서버를 관리하고 모니터할 수 있도록 Informix Enterprise Command Center (IECC)라는 관리 환경을 제공합니다. 이 GUI 환경은 관리자가 분산된 데이터베이스를 중앙 관리 콘솔에서 관리할 수 있어 분산된 데이터베이스를 관리하기가 대단히 편합니다.

ON-Bar를 포함한 Informix 관리 유틸리티는 IECC에 통합되어 분산 관리를 가능하게 합니다. IECC는 모든 데이터베이스 서버, 객체 및 현재 사용하고 있는 관리 응용프로그램을 검색하여 찾아내고 관리 콘솔에 아이콘의 형태로 나타냅니다. 이렇게 함으로써 데이터베이스 관리자가 데이터베이스 서버, 객체 및 도구를 사용하기 위해 선택하는 관리 작업을 쉽게 수행할 수 있습니다.

예를 들어, 여러 개의 데이터베이스 서버 중에서 데이터베이스 서버 5의 DB 영역 1을 백업하려면 관리자는 단순히 관리 콘솔에서 데이터베이스 서버 5를 지정하기만 하면 됩니다. 일단 선택이 되면 IECC는 서버에 접속하여 관리자가 DB 영역 1을 선택할 수 있는 서버 5에 대해 구성된 모든 데이터베이스 객체를 식별해 냅니다. 그러면 관리자는 관리 콘솔을 통하여 데이터베이스 서버 5에 대하여 ON-Bar 유틸리티를 호출해 DB 영역 1을 백업할 수 있습니다.

또한 관리자는 서버 5에는 로그온하기만 하고 명령행 인터페이스를 통하여 ON-Bar를 호출할 수도 있습니다. ON-Bar가 호출되는 방법과 관계없이 먼저 이 유틸리티가 작업이 수행될 노드에 설치되어 적절히 구성

되어 있어야 합니다.

다른 회사의 저장 관리자 GUI

다른 회사의 저장 관리자 GUI를 통하여 ON-Bar를 호출할 수도 있습니다. 저장 관리자 GUI에서 백업이나 복원 세션이 실행되는 경우, 저장 관리자는 자동적으로 ON-Bar를 호출하여 매개변수와 실행 요구를 onbar 프로그램에 전달합니다. 실행 요구를 하는 관리자는 저장 관리자가 onbar 프로그램을 호출하는 과정을 명확하게 알 수 있습니다.

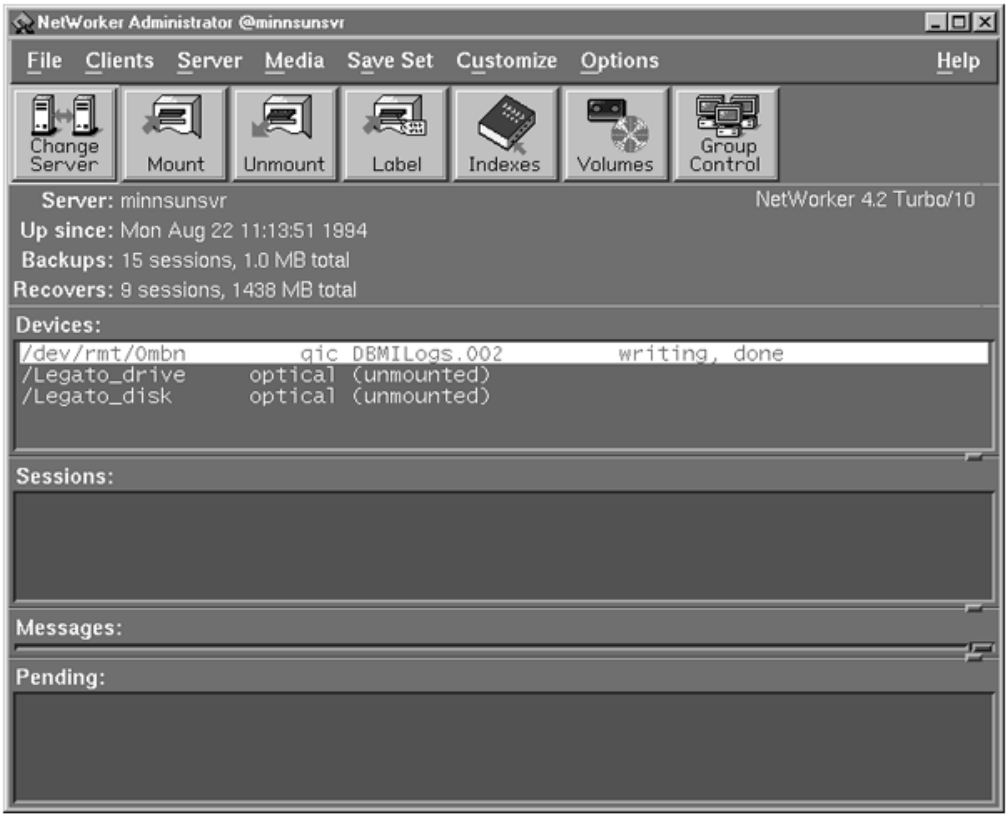


그림 8: Legato NetWorker GUI를 통한 ON-Bar 호출

그림 8은 Legato NetWorker GUI를 사용하여 백업하는 방법을 설명한 것입니다. 제품마다 GUI는 다르지만 각 GUI를 통하여 ON-Bar 작업을 실행하는 기본 개념은 크게 다르지 않습니다.

저장관리자 (Storage Manager)

저장 관리자는 저장 관리자 공급자가 제공한 다른 회사의 제품일 수도 있고 Informix 저장 관리자일 수도 있습니다.

Informix 저장 관리자

다른 회사의 저장 관리자가 제공하는 고급 기능이 필요하지 않은 사용자를 위해 Informix에서는 Informix 저장 관리자(Informix Storage Manager: ISM)라고 하는 간단한 장치 관리자를 제공합니다. ON-Bar의 번들로 제공되는 ISM을 사용하여 관리자는 OnLine 데이터베이스를 시스템 하드웨어 공급자가 제공하는 테이프나 디스크 드라이브와 같은 간단한 장치에 백업하고 복원할 수 있습니다.

ISM는 다음과 같은 작업에 사용될 수 있습니다.

- 장치 생성, 수정 및 삭제
- 논리적 컬렉션으로 장치 그룹화
- 볼륨, 저장 객체 및 장치가 로크된 상태인지 사용중인지 결정
- 로크된 저장 객체 볼륨, 저장 객체 및 장치 해제
- 볼륨 이름과 라벨 확인

신뢰성을 보장하기 위해 ISM은 모든 백업 및 복원 요청, 그리고 처리를 이벤트 로그에 기록합니다. 관리자는 이 이벤트 로그를 사용하여 작업의 성공 여부 등, 수행된 작업에 대한 중요한 정보를 알아낼 수 있습니다.

다른 회사의 저장 관리자

데이터베이스를 간단한 저장 장치에 백업하는 사용자들에게는 ISM도 훌륭한 해결책이기는 하지만 기능에 한계가 있습니다. 예를 들어, 동시 스트림의 수나 지원되는 장치의 수가 제한됩니다. 이러한 제한 때문에 ISM은 자동 교환 장치 기능이나 많은 수의 동시 스트림 지원과 같이, 복잡한 저장 관리가 주요 요소로 필요한 대용량의 분산 데이터베이스 환경은 지원하지 못합니다.

대용량의 분산 환경에 필요한 모든 기능을 제공하도록 ISM을 향상시키는 대신에 Informix는 산업 표준 XBSA 인터페이스를 준수하여 첨단 저장 관리자와 쉽게 통합될 수 있는 방법을 채택하였습니다. 이러한 전략은 고객의 필요에 맞게 최상의 저장 관리자를 선택할 수 있도록 하면서 Informix는 고객에게 가능한 최고의 데이터베이스 시스템을 제공하는데 중점을 두기 위한 것입니다.

현재, Informix는 20여 개의 저장 관리 공급자들과 XBSA 통합에 협력하고 있습니다. 이 절에서는 이미 XBSA를 통하여 ON-Bar와 통합된 4개의 저장 관리자에 대하여 간단히 설명합니다. 설명될 저장 관리자는 IBM ADSTAR Distributed Storage Manager, Hewlett-Packard OpenView OmniBack II, Legato NetWorker, 및 EMC Data Manager입니다.

IBM ADSTAR Distributed Storage Manager

IBM ADSTAR Distributed Storage Manager (ADSM)는 데이터 백업, 복원 및 보관을 지원할 뿐만 아니라 재난 복구와 계층적 저장 관리(HSM)와 같은 고급 요구도 만족하는 이해하기 쉬운 저장 관리자입니다. IBM ADSM의 주요 기능으로는 실행되는 플랫폼의 보고 느끼는 특징을 살리도록 설계된 직관적이고 사용하기 쉬운 GUI, 권한이 부여되지 않은 사용자의 접근을 막는 클라이언트/서버 인증, 저장 매체의 효율적 사용, 그리고 광범위한 장치 지원 등을 들 수 있습니다.

IBM ADSM은 OS/2, VM, AIX, VSE/ESA, OS/400, 및 MVS를 포함한 모든 IBM 운영 환경을 지원합니다. IBM 운영체제가 아닌 것 중에서는 HP-UX와 Sun Solaris를 지원합니다. 백업 서버는 IBM AIX나 Sun Solaris를 사용해야 합니다. IBM은 1997년 내에 HP-UX도 백업 서버로 사용할 수 있도록 할 예정입니다.

Hewlett-Packard (HP) OpenView OmniBack II

HP OpenView OmniBack II는 다중 공급자 분산 컴퓨팅 환경에서 기업 규모의 백업과 복원을 목적으로 한 저장 관리자입니다. 주요 기능으로는 중앙 백업 및 복원 관리, 자동화된 백업 및 복원, 분산 관리 작업 배분, 사용하기 쉬운 GUI, 광범위한 중앙 및 분산 백업 장치 지원, 그리고 UNIX와 PC 환경에서의 모든 공통 데이터 형식 지원 등이 있습니다.

HP OpenView OmniBack II는 많이 사용되는 UNIX와 PC 운영 환경을 포함하여 광범위한 플랫폼에서 데이터를 백업하고 복원하는데 사용될 수 있습니다. HP나 Windows NT 서버를 백업 서버로 사용할 수 있습니다. 백업 장치는 중앙 시스템에 연결된 것이나 멀리 떨어진 장치를 모두 사용할 수 있으며, 어떤 플랫폼에서도 중앙에서 제어할 수 있습니다.

수 테라바이트 크기의 대용량 데이터를 처리하는 경우, HP는 HP OpenView OmniStorage라고 하는 계층적 저장 관리(Hierachical Storage Management: HSM) 솔루션을 제공합니다. OmniStorage와 OmniBack II는 OminBack II가 HSM 파일을 직접 효율적으로 백업할 수 있도록 통합되었습니다.

Legato NetWorker

Legato NetWorker는 현대의 대용량, 복합, 분산 컴퓨팅 환경을 지원할 수 있도록 설계된 기업 저장 관리자입니다. 네트워크 백업, 복구, 보관 및 기업 전체에 걸쳐 저장된 모든 데이터와 응용프로그램 파일의 이주를 위해 기본 제품의 환경을 구성할 수 있습니다. 지원되는 백업 서버 운영 체제로는 Microsoft NT, Novell NetWare, Sun OS, Sun Solaris, IBM AIX, HP-UX 및 SCO UNIX 등이 있습니다. 주요 기능으로는 클라이언트 병렬 처리를 위한 독점적 분산 "smart" 아키텍처, 자동화된 자원 관리, 그리고 지역, 중앙, 원격 관리를 위한 일관된 전역 메뉴 등이 있습니다.

Legato에는 NetWorker로 쉽게 통합될 수 있는 고급 관리 옵션인 다양한 모듈러가 있습니다. 이러한 옵션에는 계층적 저장 관리(HSM) 지원, 자동 교환 장치 지원, 메인프레임 확장, 그리고 SNMP 프레임워크 통합 등이 있습니다.

그 외에도, NetWorker는 온라인 데이터베이스와 응용프로그램 보호용 Legato's BusinessSuite를 사용하여 기능을 강화시킬 수 있습니다. 대용량 데이터 센터와 데이터 웨어하우스를 위해 Legato는 시간당 400 GB를 넘는 초고속 데이터 보호를 위한 PowerEdition을 제공합니다.

EMC Data Manager

EMC Data Manager (EDM)는 네트워크 및 분산 환경에 이상적인 하드웨어와 소프트웨어 백업 및 복원 솔루션입니다. EDM은 성능과 확장성을 높이기 위해 설계되었기 때문에 시간당 70 GB의 속도로 10 TB까지 저장할 수 있으며 수백 개의 클라이언트와 방대한 데이터베이스를 처리할 수 있습니다. EDM의 다른 기능으로는 Sun, IBM, HP, Digital, Sequent, Pyramid, AT&T, NCR, SGI, DG clients 등의 다중 플랫폼 지원, 중앙 관리를 강화하는 사용하기 쉬운 GUI, 그리고 가용성이 높은 다양한 기능 등이 있습니다.

또한 EDM에는 무인 자동화 백업 처리를 위한 많은 기능이 있습니다. 자동 볼륨 관리 능력과 완전한 로봇식의 라이브러리 유닛이 제공됩니다. 게다가, EDM은 매체를 효율적으로 사용하기 위해 백업 순환 스케쥴링을 이용하여 매체 관리를 빈틈없이 통합합니다.

결론

Informix ON-Bar 백업 및 복원 시스템은 고성능, 높은 가용성, 관리성, 신뢰성 및 유연성을 가지고 있어 사용자에게 빠르고 쉬우며 안정적인 완전한 데이터베이스 백업 및 복원 솔루션을 제공합니다.

ON-Bar는 산업 표준 XBSA 인터페이스를 통하여 다양한 첨단 저장 관리자를 사용할 수 있도록 합니다. 산업 표준 솔루션을 준수함으로써, Informix 사용자들은 저장 관리 목적에 가장 부합되는 저장 관리자를 다양하게 선택할 수 있습니다. 또한 ON-Bar는 표준을 준수하는 솔루션이기 때문에 회사의 기업 백업과 복원 전략에 쉽게 통합될 수 있습니다.