Tech Info(vol.6)

Y2K 문제 대응

소개

"2000년" 문제는 "1997"을 "97"로 또는 "2001"을 "01"로 표시하는 것처럼, 두 자리수만을 사용하여 연도를 입력하거나 저장하는 응용프로그램과 내장 시스템에 의해 야기되는 문제입니다. 이러한 방법은 해당 세기가 20세기인지 21세기인지에 대한 혼란을 일으킬 수 있습니다. 그러므로 이와 같은 날짜를 기반으로 하는 수학적 연산은 부정확한 결과를 나타낼 수 있으며, 그 혼란은 저장과 입력의 두 곳에서 일어날 수 있습니다.

다수의 오래된 응용프로그램들은 연도를 두 자리수로만 저장하기 때문에 1901년과 2001년을 구별할 수 없습니다. 이 점이 수정하기에 가장 어려운 문제인데, 그 이유는 데이터 입력을 받아들이는 저장 메커니즘과 응용프로그램을 둘 다 수정해야 하기 때문입니다.

일부 응용프로그램들은 네 자리수를 모두 사용하여 연도를 저장하기도 하지만, 대부분의 응용 프로그램에서는 사용자에게 단지 두 자리수만을 입력하도록 하고, 처음 두 숫자가 "19"라고 추정하거나 컴퓨터의 시스템 시계와 동일한 세기를 사용합니다. 타임스탬프를 사용하는 다른 소프트웨어 프로그램들도 악영향을 받을 수 있습니다.

또 다른 문제는 윤년에 관한 문제입니다. 일부 컴퓨터 소프트웨어는 2000년이 윤년이라는 사실을 인식하지 못하여 "02/29/2000"을 유효한 날짜로 인정하지 않거나 02/29/2000까지의 시간 간격 길이를 잘못 연산할지도 모릅니다.

이 외에도 2000년과 관련된 또 다른 문제들이 발생할 수 있습니다(예를 들어, 일부 시스템은 "9/9/99"를 날짜 입력의 마지막 플래그(end-of-input flag)로 사용하여, 입력이 이러한 값에 이르게 되면 파일의 나머지 부분을 무시해 버릴지도 모릅니다).

주의

이 글에서는 MM/DD/YY 또는 MM/DD/YYYY 형식의 날짜 표기를 하였습니다. 따라서, 01/02/1999는 2월 1일이 아니라 1월 2일입니다.

배경

5 메가바이트의 디스크 드라이브 가격이 1000 달러 이상일 때에는 저장 공간을 절약하는 것이 매우 중요하였기 때문에 일부 소프트웨어들은 날짜를 "12/31/98"과 같이 연도의 마지막 두 자리수만으로 저장하였습니다. 이러한 소프트웨

/98"과 같이 연도의 마지막 두 자리수만으로 저장하였습니다. 이러한 소프트웨어는 일반적으로 연도의 처음 두 숫자를 "19"라고 추정하거나, 처음 두 숫자가 현재 연도의 처음 두 숫자와 동일하다고 추정하였습니다. 예를 들어, 현재 연도가 2000 년에서 2099 년 사이라면 "12/31/98"을 "12/31/2098"로 인식합니다. 그러나 연도가 1999 년에서 2000 년으로 넘어갈 때 일부 시스템은 "1/1/00"을 "1/1/
2000" 대신 "1/1/1900"으로 처리할 것입니다.

프로그램 디자이너와 프로그래머들은 마침내 날짜에 대한 해석이 애매하다는 사실을 인식하게 되었으나, 1960 년대와 1970 년대 그리고 1980 년대 중반까지 작성된 소프트웨어들은 2000 년이 되기 전에 대체될 것이라고 기대하였습니다. 그러나 불행하게도, 컴퓨터 하드웨어는 훨씬 빠르게 발전하고 가격도 내려갔지만, 컴퓨터 소프트웨어는 그 속도만큼 빠르게 진보하지 못하였습니다. 그러한 이유와 함께 이전부터 사용해 오던 소프트웨어들을 대체하려면 너무 많은 비용이 들 것이라는 이유로, 몇 십년 전에 만들어진 많은 프로그램들을 오늘날의 컴퓨터에서도 여전히 실행하고 있습니다. 또 다른 경우로, 서브루틴 라이브러리들은 서브루틴이 처음 작성된 이래로 몇 십년 동안 여전히 사용되어 오고 있습니다.

일반적이지는 않지만 또 다른 문제는, 윤년을 고려하는 일부 컴퓨터 프로그램도 2000 년이 윤년이라는 사실은 인식하지 못한다는 점입니다. 윤년에 대한 법칙은 "4 로 나누어 떨어지는 해는 윤년이고, 그 중에서 100 으로 나누어 떨어지면 윤년이 아니지만 다만 400 으로 나누어 떨어지는 해는 윤년"입니다. 이러한 법칙에 의하여, 1700 년과 1800 년 그리고 1900 년은 윤년이 아니지만 2000 년은 윤년인 것입니다. 그러나 일부 소프트웨어는 100 으로 나누어 떨어지는 모든 연도를 거부하도록 작성되었기 때문에 일부 컴퓨터 프로그램은 "02/29/2000" 같은 날짜를 인식하지 못할 수도 있습니다.

문제의 심각성

이러한 "오류"(대부분 우연이 아니라 신중하지 못한 데서 오는)는 인류 역사상 가장 값비싼 대가를 치르게 될 실수일 것입니다.
이 문제를 고치는 것은 간단할지 몰라도, 변경해야 할 프로그램과 데이터 파일들의 모든 부분을 프로그래머가 확인해야 하기 때문에 엄청난 시간이 소요될 것입니다. 어떤 경우에는 프로그램의 소스 코드가 더 이상 존재하지 않을지도 모르며, 소스 코드가 있더라도 그 소스 코드에 적당한 컴파일러나 다른 툴들이 더 이상 존재하지 않아서 사실상 프로그램을 수정하거나 다시 컴파일할 수 없을지도 모릅니다. 실제로 얼마나 많은 프로그램이 영향을 받을지 그리고 이를 고치는 데 얼마나 많은 시간과 비용이 필요할지 아무도 모르지만, 어림잡아 수천억 달러로 추정되며 2000 년이 되기까지 남은 시간은 모든 문제를 고치기에

너무도 부족합니다. (어떤 해학가가 지적했듯이, 이것은 조정될 수 없는 마감 시간인 것입니다.)

이러한 문제의 충격은 아직 알 수 없지만, 아마도 매우 클 것입니다. 전문가들은 2000 년이 되기 전까지 모든 프로그램을 수정하기에는 프로그래머와 시간이 충분하지 않다는 사실에 동의합니다. 컴퓨터 시스템이 제대로 작동하지 않음으로써 발생할 수 있는 피해는 오류가 발생하는 시스템에 따라 사소한 불편에서부터 생존을 위협할 정도까지 다양합니다. 하늘에서 추락하는 사람도 있을 수 있고, 원자력 설비가 제어되지 않거나, 정부가 붕괴될지도 모릅니다. 그러나 그 충격이 어느 정도 심각할지는 아무도 모릅니다. 따라서 상당히 심각한 문제들에 대해서는 미리 대비하는 것이 신중한 방법이라 하겠습니다. 이러한 문제를 처리하려면 절차와 백업 시스템이 있어야 할 것입니다. 다행히 회사에 태풍이나 지진과 같은 위기 상황에 대처할 수 있는 유용한 재해 복구안이 있다면, Y2K 문제 처리에도 얼마간의 준비가 되어 있다고 할 수 있습니다.

피해 대상

대부분의 문제는 소프트웨어의 문제이지 하드웨어의 문제가 아니기 때문에, 소프트웨어와 날짜를 사용하는 거의 모든 유형의 시스템에서 문제가 발생할 수 있습니다. 예를 들면 비디오 카세트 레코더(날짜를 잘못 해석하기 때문에 좋아하는 프로그램을 녹화할 수 없게 될지도 모르는)와 연금 시스템(나이를 잘못 계산하여 퇴직금 지급이 거부되는)이 있습니다.

피해를 받을 수 있는 몇몇 시스템 유형은 다음과 같습니다.

- 내장 시스템(embedded systems)
- 하드웨어 또는 컴퓨터 내의 "BIOS"
- 운영 체제(예를 들어, UNIX, 마이크로소프트 윈도우 등)
- 데이터베이스 서버(다행히도, Informix OnLine 데이터베이스 소프트웨어는 발생 가능한 2000년 문제에 대해 폭 넓게 테스트되었고, 오류 또는 종속성(dependencies)과 같은 문제가 거의 없다고 알려졌습니다.)
- 데이터베이스 상의 "관련 소프트웨어" 트랜잭션을 제어 및 감시하는 소프트웨어 응용프로그램

Informix Y2K 준수

Informix 의 날짜(DATE) 데이터 형식은 1899 년 12 월 31 일부터 해당 날짜까지의 날짜 수를 나타내는 정수로 날짜를 저장합니다. 그러므로 날짜가 속한 세기와 관련된 혼동은 거의 없습니다. DATETIME 데이터 형식은 항상 네 자리수의 연도로 저장됩니다.

내부적으로 사용되는 몇몇 타임스탬프는 운영 시스템으로부터 가져와서 운영 시스템에 의해 보내진 모든 연도 정보와 함께 그대로 보관됩니다. 이러한 타임스탬프는 운영 시스템이 2000 년을 준수하는 한에는 2000 년에 의해 영향받지 않을 것입니다. 기타 내부 타임스탬프는 단순한 비교 타임스탬프로서, 상대적인 시간을 결정하는 비교를 위해 사용됩니다. 이러한 타임스탬프는 세기가 바뀌는 것에 영향받지 않을 것입니다.

연도가 문제시되는 또 하나의 상황은 두 자리수 연도 표시를 사용하는 데이터베이스 내로 데이터가 삽입될 때 발생합니다. 사용자가 두 자리수 연도 표시를 사용하여 날짜를 입력할 경우, 날짜는 1899 년 12 월 31 일부터의 날짜 수를 나타
내는 정수로 변환됩니다. 이전 버전의 Informix 제품들은 기본값에 의해 현재 날짜나 세기에 상관없이 두 자리수 연도들을 20 세기로 인식하였습니다. 예를 들어, 사용자가 "1/1/01"이라고 입력하면, 현재 날짜가 2001 년 1 월 1 일일 경우라도 1901 년 1 월 1 일로 될 것입니다. 그러나 프로그램에서 TODAY 키워드를 사용하거나, 사용자가 "1/1/2001"을 입력한다면, 화면에 두 자리수로만 연도가 표시되더라도 현재 날짜는 올바로 저장될 것입니다.

새로운 버전의 Informix 제품들은 "DBCENTURY"라는 환경 변수를 지원하는데, 이는 사용자가 날짜 영역에 두 자리수로만 연도를 입력할 경우, DATE/DATETIME 데이터 형식에 대해 기본 세기 값을 Informix 에서 어떻게 설정할지를 제어합니다. 만약 DBCENTURY 변수가 현재가 아니거나 설정되지 않았다면, Informix 는 기본 세기 값을 현재 세기로 설정하여 컴퓨터의 시스템 시계로부터 가져옵니다. DBCENTURY 특징에 대한 자세한 사항은 Informix SQL Reference 를 참고하십시오.

코드 내에서 문제 발생을 피하는 방법

새로운 프로그램을 작성하거나 이전 프로그램을 업그레이드할 때에는 두 자리수 연도 사용을 없애야 할 것입니다. (물론 이것이 때로는 쉽지 않은데, 예를 들어, 소프트웨어의 서로 다른 두 부분이 동일한 데이터 파일로부터 읽어온다면 데이터 파일은 물론 소프트웨어의 두 부분도 동시에 업데이트를 해야 할 것입니다.)

다행히도, Informix 데이터베이스 서버와 같은 현대식 고성능 데이터베이스 소프트웨어는 네 자리수 연도로 날짜를 저장합니다. 날짜를 DATE 또는

DATETIME 형식의 칼럼에 저장할 경우, 데이터베이스에서는 연도의 네 자리수를 모두 저장합니다. 일반적으로 DATE 필드가 아닌 필드들(예를 들어, CHAR(6))을 사용하여 연도의 두 자리수만으로 데이터를 저장할 경우에만 두 자리수 연도와 관련된 문제가 발생할 것입니다. 그러므로 DATE 또는 DATETIME 필드를 사용하여 날짜 데이터를 저장할 것을 권장합니다.

데이터베이스의 데이터를 새로 고치는 것만으로는 충분하지 않습니다. 데이터를 처리하는 모든 루틴도 새로 고쳐야 합니다. 예를 들어, 날짜들 사이의 차이를 계산할 경우에는(예를 들어, 오늘 날짜로부터 출생한 날짜를 뺌으로써 나이를 계산하고자 할 때), 네 자리수 연도를 사용하여 작업이 이루어져야 할 것입니다. Informix 는 INTERVAL 값(INTERVAL 데이터 형식에 대한 자세한 내용은 Informix Guide To SQL: Reference 참고)을 연산할 때 네 자리수 연도를 사용하므로, Informix 고객들이 이러한 데이터 형식을 사용할 경우에는 아무런 문제도 없을 것입니다. 날짜 계산을 위한 서브루틴을 작성할 경우에는, 네 자리수 연도를 사용하고 있는지 검토하고 100 년 이상의 시간 간격(interval)을 처리할 수 있는지 확인해야 합니다.

ESQL/C 와 같은 프로그래밍 언어로 내장 SQL 을 사용하는 프로그램을 작성할 때에는 연도의 네 자리수를 모두 저장하는 방식으로 호스트 변수들을 선언해야 합니다. 최근의 Informix 엔진과 클라이언트 소프트웨어 개발 킷(Client Software Development Kits: SDKs)을 사용하고 있다면, 호스트 변수들을 DATE 또는 DATETIME 형식으로 선언하는 데에 아무 문제도 없을 것입니다. INTERVAL 형식의 호스트 변수는 충분한 작업 범위를 갖도록 해야 합니다. 예를 들어, 연금을 받게 될 나이를 계산할 때에는 100 년 이상을 처리할 수 있는 INTERVAL 을 사용해야 하며, 호스트 변수에 interval year(2)와 같이 선언하는 일이 없도록 해야 합니다.

데이터 입력 화면에서도 완전한 네 자리수 연도 사용이 허용되어야 하는데, 연도의 마지막 두 자리수만을 입력해도 되거나 두 자리수만 입력하도록 되어 있는 경우가 많습니다. 그러나 이것은 위험한 관행입니다.
또 다른 문제 발행 요인은(다행히 현대식 데이터베이스 소프트웨어 사용 시 매우 드물기는 하지만) 어떤 값을 "플래그(flags)"로 사용하는 것입니다. 예를 들어, 일부 COBOL 시스템은 "end of input" 표시자로 "9/9/99" 날짜를 사용합니다.

결국, 방어적으로 코드를 작성할 것을 권장합니다. 복잡한 길에서 차를 운전할 때 잠재적인 위험에 대해 알고 있어야 하듯이, 프로그래밍을 할 때도 잠재적인 위험에 대해 알고 있어야 할 것입니다. 문자열의 형식으로(예를 들어, "12/31/1998") 날짜를 전달하는 서브루틴을 작성할 경우에는, 날짜가 유효한지를 검사하고 유효하지 않을 때에 알맞은 오류 메시지를 나타내도록 하는 것이 현명한

방법일 것입니다.

Informix 의 날짜 관련 함수를 호출할 때에는, 함수로부터의 리턴 코드를 검사해야 합니다. 예를 들어, ESQL/C 프로그램에서는 코드가 다음과 유사해야 합니다.

```
strcpy(output_format, "yyyy-mm-dd");

error_code=rfmtdate(date_value, output_format, output_buffer);

if (error_code != 0

{

printf("ERROR: program <name> function <name> detected \n");

printf("an error when calling the rfmtdate() function \n");

printf("with the following parameter values ...\n");

printf("format = %s, ...", output_format, ...);

}
```

실패를 경험하지 않기 위한 것이라 해도 추가 작업량이 너무 많은 것이 아니냐는 생각이 들 수도 있지만, "잘못될 수 있는 것은 결국 잘못될 것이다"라는 "머피의 법칙"을 기억하는 것이 중요합니다. 프로그램은 "예기치 않은 상황을 예측"해야 하고, 잘못되었을 때 적어도 경고를 해 주어야 합니다.

소프트웨어의 날짜 관련 부분에 대한 실제의 수정 작업이 일반적으로 쉽다고 해도, 다음의 두 가지 문제에 직면하게 될 것입니다:

1. 변경해야 하는 모든 위치를 확인하는 문제
2. 그 변경 사항들을 상호 조정하는 문제. 예를 들어, 서로 다른 두 프로그램이 같은 파일로부터 읽어올 경우에는 그 파일과 두 프로그램을 동시에 업데이트해야 할 것입니다. 프로그램 하나와 데이터 파일만을 업데이트하고, 다른 하나의 프로그램을 업데이트하지 않는다면, 두 번째 프로그램은 거의 확실히 실패할 것입니다. 소프트웨어를 업데이트할 때에는, 아마도 소프트웨어에서 두 자리수 연도나 네 자리수 연도 중 하나를 처리할 수 있도록 만들고자 할 것입니다. 그렇게 해서 프로그램들을 한

번에 하나씩 변경하고(동시라기 보다), 마지막에 데이터 파일을 변경할 수 있습니다.

문제가 발생하자마자 감지하는 방법

SQL 소프트웨어에서는, 제한 조건(constraints)을 사용하여 날짜 관련 오류에 대한 유형을 감지할 수 있을 것입니다. 예를 들어, 1985 에 사용하기 시작한 회계 시스템이 있고, 그 안에는 1985 년 이전의 데이터가 전혀 없다고 가정해 봅시다. 또한, 소프트웨어가 현재 문을 닫은 회사의 제품이며 소스 코드의 사본을 가지고
있지 않다고 가정해 봅시다. 그렇다면 이 소프트웨어가 네 자리수의 날짜를 사용하는지 두 자리수의 날짜를 사용하는지 알 수 없지만, 적어도 일부에서는 두 자리수의 날짜를 사용하는 것으로 의심이 될 수 있을 것입니다(아마도 입력 화면에 기반하여 볼 때). 다음 방법을 통해 두 자리수 날짜 감지도를 시험해 볼 수 있습니다.

- 별도의 컴퓨터에 Informix 데이터베이스 소프트웨어와 회계 시스템 소프트웨어 사본 및 데이터의 사본을 설치합니다(데이터 전체를 로드할 필요는 없으며, 데이터의 샘플 정도로도 충분할 것입니다).
- 이 컴퓨터에서 날짜를 2000년으로 설정합니다(다른 소프트웨어에서 문제가 발생하지 않도록 하려면 현재 다른 사람이 이 컴퓨터를 사용하지

다. 만약 이 응용프로그램이 데이터베이스 상에서 실행되고 있다면, 잘못된 값을 거부하는 제한 조건을 추가할 수 있습니다(예를 들어, 1901 년 이전의 값을 거부하도록 하여 "00"이 "1900"으로 해석되는 것을 감지할 제한 조건). 응용프로그램을 수정할 수는 없을지라도, 데이터베이스를 수정하여 유효하지 않은 날짜를 지닌 레코드를 거부하거나 여과시킬 수 있는 제한 조건을 만들 수 있습니다.

이렇게 하면 발생 가능한 문제에 대해 즉각적인 경고를 얻을 수 있습니다. 그러나 이것이 실제로 문제를 해결하는 것은 아닙니다.

이 방법은 특히 Informix 엔진 상에서 응용프로그램을 실행하고 있으며, 응용프로그램 자체를 수정할 방법이나 2000 년 문제에 치명적인지 알아보기 위해 소스 코드를 시험해 볼 방법을 갖고 있지 않은 고객에게 유용합니다.

고객들은 응용프로그램을 다른 컴퓨터에서 실행하고 날짜를 일부러 2000 년으로 앞당겨 설정하여 문제가 발생하는지의 여부를 살펴봄으로써 발생 가능한 문제를 미리 파악할 수 있습니다.

2000년 문제 해결을 위한 대비책

현재 지원되는 Informix 데이터베이스 서버는 2000 년과 관련된 어떠한 문제도 야기할 것 같지 않지만, 다른 소프트웨어는 그렇지 않을 수도 있습니다. 잠재적인 2000 년 문제를 처리하기 위해 간단하게나마 2000 년 문제 해결을 위한 대비책
을 마련하여야 할 것입니다. 재해가 발생한다면 그것이 자연 재해이건 인간이 만든 재해이건, 사내 직원 및 고객들에 대한 서비스의 중단을 가능한 한 최소로 해야 할 것입니다.

2000 년 문제 해결을 위한 대비책으로 다음과 같은 사항들을 고려할 수 있을 것입니다.

- 데이터 백업
- 소프트웨어 백업
- 소스 코드 백업
- 툴 백업
- 지속적인 전력 공급

데이터 백업

이것은 반드시 해야하는 일입니다. 현재 하고 있지 않다면 지금 당장 시작하십시오. 백업은 계속 교체해야 합니다(백업 테이프를 한 세트만 가지고 있는데, 이전 백업 위에 오늘 다시 백업하는 동안 시스템이 고장난다면, 원본 데이터 또

는 완전한 백업이 사라져 버릴 것입니다).

몇몇 백업은 다른 장소에, 가능하면 멀리 떨어진 곳에 보관해야 합니다. 이렇게 함으로써 지진이나 화재와 같은 재해에도 대비할 수 있을 것입니다.

소프트웨어 백업

데이터 백업의 경우와 마찬가지로 다른 두 장소를 사용하는 규칙을 따르십시오. 그리고 소스 코드와 컴파일된/실행 가능 코드를 모두 저장하도록 하십시오.

툴 백업

만약 구식 컴파일러로만 소스 코드를 컴파일할 수 있다면(사람이 읽을 수 있는 소스 코드에서 컴퓨터에서 실행 가능한 "기계 코드"로 번역할 수 있다면), 그 컴파일러의 사본도 안전한 곳에 보관해야 할 것입니다. 툴(또는 소프트웨어)을 처음 제작한 회사들이 오래 전에 사라졌거나 그러한 구식 버전의 제품을 지원하지 않을지도 모르기 때문에 이 일은 중요합니다. 대부분의 소프트웨어 라이센스는 소프트웨어의 "보관" 사본을 만드는 것을 허용합니다.

지속적인 전력 공급

12/31/1999 부터 01/01/2000 에 걸쳐 컴퓨터를 실행 상태로 둘 계획이라면, 계속
적으로 전력이 공급되는 설비(UPS)에 컴퓨터를 연결해 두어야 합니다. 이렇게 하면 전력이 중단되더라도 안전하게 소프트웨어를 종료하거나 하드웨어를 끌 수 있는 몇 분의 시간을 확보할 수 있을 것입니다. 일반적으로 대부분의 UPS 는 배터리를 완전히 소모하기 전 몇 분 동안만 전력을 제공하므로 이러한 방법을 사용하려면 야간에도 사무실에 직원이 있어야 할 것입니다.

경우 1: 문제가 있음을 알고 있을 경우

Informix 서버 상에서 응용프로그램을 실행하고 있는데 그 응용프로그램이 2000 년 규정을 완전히 준수하지 않는다면, DBCENTURY 환경 변수를 사용하여 문제를 해결할 수 있을 것입니다. DBCENTURY 환경 변수는 두 자리수 연도를 지닌 날짜를 해석하는 방법을 서버에게 알려줍니다. 예를 들어, 응용프로그램이 서버로 "01/01/50"이란 날짜를 보낼 경우, 날짜가 "01/01/1950"으로 해석되도록 하려면, 서버에서 그렇게 하도록 DBCENTURY 를 설정할 수 있습니다.

DBCENTURY 환경 변수에 대한 내용은 Informix Guide To SQL: Reference 를 참고하십시오.

환경 변수는 다음 네 가지 값들 중 하나로 설정할 수 있습니다.

DBCENTURY 의미

---------- --------------------------

C Closest: 오늘과 가장 근접한 날짜를 사용

F Future: 미래에서 처음으로 일치되는 날짜를 사용

P Past: 과거에서 가장 최근의 일치되는 날짜를 사용

R cuRrent/pResent: 현재 세기를 적용

몇몇 예를 살펴보도록 하겠습니다.

관공서에서 출생 증명서를 발급하는데, 출생일을 기입하는 응용프로그램이 두 자리수 연도만을 입력할 수 있도록 한다고 가정해 봅시다. 이미 출생한 사람의 출생 증명서만을 발급할 수 있으므로 입력한 날짜는 과거일 것입니다. DBCENTURY 를 "P"로 설정하여 컴퓨터로 하여금 두 숫자의 각 연도를 과거에서 가장 최근의 날짜 즉, 오늘보다 앞선 가장 최근의 날짜로 해석하도록 할 수 있습니다. 오늘이 2000 년 1 월 3 일일 경우, 직원이 "12/29/99"라고 입력하였다면, 서버는 형식을 일치시킬 가장 근접한 이전 날짜가 "12/29/1999"이므로 이와 같이 해석할 것입니다.

주의

DBCENTURY=P 는 컴퓨터가 날짜를 "오늘보다 앞서거나 같은(prior to or equal to)" 가장 가까운 날짜가 아니라 "오늘보다 앞선(PRIOR TO)" 가장 가까운 날짜로 해석하도록 하는 것입니다. 만약 오늘이 01/03/2000 이고 출생 등록을 하는 관공서 직원이 출생일로 "01/03/00"이라고 입력하면 서버에서는 이것을 "01/03/2000"이 아니라 "01/03/1900"으로 해석할 것입니다.

마찬가지로 DBCENTURY=F(미래)에 대한 규칙은 엔진이 두 자리수의 날짜를 미래의 가장 가까운 날짜, 즉 오늘 이후(AFTER)의 가장 가까운 날짜로 해석하도록 하는 것입니다. 이것은 미래의 유지 보수 일정을 잡는 것과 같은 일정 관리용 응용프로그램에 유용할 것입니다. DBCENTURY=P 에서와 마찬가지로 "F"도 가장 가깝게 일치하는 날짜가 오늘이 될 수는 없습니다. 만일 오늘이 01/03/2000 이고, DBCENTURY 가 "F"로 설정되어 있을 때 01/03/00 이라고 입력하면 컴퓨터는 날짜를 01/03/2000 이 아니라 01/03/2100 으로 처리할 것입니다.

DBCENTURY=C 로 설정하면, 엔진은 두 자리수의 연도를 오늘과 가장 가까운 날짜로서 해석할 것입니다. 예를 들어, 오늘이 01/03/2000 이고 DBCENTURY 가 "C"일 경우 날짜를 07/04/49 로 입력하면, 엔진은 이 날짜를 07/04/2049 로 해석할 것입니다. 그리고 날짜를 07/04/51 이라고 입력하면 그 날짜는 07/04/1951 로 해석될 것입니다.

주의

이 엔진의 일부 버전에서는(1998 년 9 월), 엔진이 "근사값(closest)"을 연산할 때 전체 날짜가 아니라 연도만을 봅니다. 예를 들어, 오늘이 07/01/2000 이라고 가정해 봅시다. 엔진이 가장 가까운 날짜를 정확히 채택한다면 07/02/50 은 07/02/1950 으로 해석될 것이고, 반면에 06/30/50 은 06/30/2050 으로 해석될 것입니다. 그러나 불행하게도, "근사값(closest)"을 연산할 때 현재 버전의 엔진은 월이나 일은 검사하지 않고 연도만을 검사합니다. 그러므로 앞의 두 날짜들은 2050 또는 1950 중에서 동일한 연도에 일어난 것으로 간주될 것입니다. DBCENTURY=C 를 사용하고 거의 정확히 미래의 50 년 또는 과거의 50 년의 날짜를 입력해야 한다면, DBCENTURY 에 의지하지 말고 연도의 모든 네 자리수를 입력하는 것이 가장 좋은 방법입니다. 이전의 49 년이나 앞으로의 49 년 내의 날짜에 대해서는 DBCENTURY=C 가 유용할 것입니다.

DBCENTURY=R 로 설정한다면, 엔진은 현재 날짜의 세기를 사용하게 됩니다. 예를 들어, 01/03/99 와 같은 날짜는 1999 년에는 01/03/1999 로 해석되는 반면에, 2000 년에는 동일한 날짜가 01/03/2099 로 해석될 것입니다.

DBCENTURY에 대한 마지막 주의 사항

DBCENTURY 는 두 자리수의 날짜 뿐만 아니라 한 자리수와 세 자리수의 날짜에도 적용됩니다. 이해를 돕기 위해서, 여기서는 대부분 두 자리수의 날짜에 관해서만 설명하였습니다. 만약 고고학적 데이터나 기타 데이터를 입력하면서 A.D. 1 년과 A.D. 999 년 사이의 연도를 입력한다면, 정확한 날짜를 입력하도록 주의를 기울여야만 하며 DBCENTURY 를 설정하지 않아야 합니다. 다행히, A.D. 1-999 연대의 고고학적 데이터나 기타 데이터를 기록하도록 설계된 응용프로그램 소프트웨어는 연도의 두 자리수만을 입력하도록 제한되어 있지 않을 것입니다.

경우 2: 문제가 있음을 너무 늦게(12/31/1999 이후) 알게 된 경우

따랐다고 가정하며, 특히 1999 년에서 2000 년으로 변하기 바로 전에 모든 데이터를 백업하였다고 가정합니다. 또한 컴퓨터 하드웨어와 운영 시스템 등이 모두 올바로 작동하고 있다고 가정합니다.

주의

아래의 설명은 Informix 소프트웨어 뿐만 아니라 대부분의 소프트웨어에도 적용됩니다. 이해를 돕기 위해, 발견된 문제는 Informix 소프트웨어에 관련된 것이고, 서버 버전을 업그레이드하고 백업으로부터 복구한 다음 계속 진행해야 한다고 가정하겠습니다. 이러한 접근법을 Informix 소프트웨어에 대해 사용하건 아니면 다른 소프트웨어에 대해 사용하건, 이 단원의 마지막 부분에 설명할 주의 사항에 유의하시기 바랍니다.

잘못된 것을 복구하려면 다음과 같이 하십시오:

1. 소프트웨어를 종료합니다. (이 단원의 마지막 부분에서 설명하고 있는 주의 사항을 참고하십시오.)
2. 컴퓨터의 날짜를 재설정합니다.
3. 문제를 바로잡습니다. 예를 들어, 문제가 2000년 규정을 준수하지 않는 이전 버전의 소프트웨어에서 실행되었기 때문이라면, 2000년 규정을 준수하는 소프트웨어의 새로운 버전을 설치하도록 합니다.
4. 백업 테이프로부터 이전 데이터를 복구합니다.
5. 백업을 만든 이후 행했던 작업을 반복합니다. 예를 들면, 최근 며칠의 신용 카드 사용 내역을 다시 입력합니다. 이 작업 방식에 따라 컴퓨터의 날짜/시간 정보를 조정해야 할 것입니다. 예를 들어, 12월 30일 6시간과 12월 31일 6시간의 모든 입력 작업을 반복할 수 있다면, 다음과 같은 일을 하게 될 것입니다.
6. 컴퓨터의 시계를 1999년 12월 30일 8:00 AM으로 설정합니다.
7. 1999년 12월 30일의 거래 내역을 입력합니다.
8. 소프트웨어를 종료합니다.
9. 컴퓨터의 시계를 1999년 12월 31일 8:00 AM으로 설정합니다.
10. 소프트웨어를 다시 시작합니다.
11. 1999년 12월 31일의 거래 내역을 입력합니다.
12. 소프트웨어를 종료합니다.
13. 컴퓨터의 시계를 2000년 1월 1일 8:00 AM으로 설정합니다.
14. 소프트웨어를 다시 시작합니다.
15. 2000년 1월 1일의 거래 내역을 입력합니다.
16. 소프트웨어를 종료합니다.
17. 등등

이렇게 해서 과거의 거래 내역을 모두 입력하십시오.

주의

응용프로그램이 야간이나 새벽에 일종의 일괄처리를 하도록 설정되어 있다면, 컴퓨터의 시계를 일괄처리가 일어나기 바로 전의 날짜/시간으로 설정한 다음, 컴퓨터에서 일괄 처리가 진행되도록 해야 할 것입니다.

1. 데이터가 정확한지 확인합니다.
2. 시계를 실제의 현재 시간으로 재설정합니다(예를 들어, 01/03/2000 10:00:00).
3. 하루나 이틀간 정상적으로 작업한 후에, 예기치 않은 문제가 없는지 데이터를 다시 확인해 봅니다.

주의 사항!!

컴퓨터에서 단 하나의 프로그램만을(예를 들면, Informix 서버) 실행하고 있다면, 컴퓨터를 종료하고 시계를 01/01/2000 이전의 날짜/시간으로 재설정한 다음, 이후의 시간으로 진행하는 것에 별 어려움이 없을 것입니다. 그러나 컴퓨터에서 다른 소프트웨어를 사용하고 있다면 시계를 재설정하고 이후의 시간으로 진행하는 것이 다른 소프트웨어에 대해 문제를 일으킬지도 모릅니다. 예를 들어, 달(또는 해)이 바뀌자마자 이전 달(또는 해)에 대한 계산서를 자동으로 생성하는 소프트웨어가 있다면 계산서의 복사본을 만들어야 할 것입니다. 계산서를 두 번 인쇄하기 위해 종이를 낭비하는 것 이외에도, 청구액이 계산서에 자동으로 추가되거나 이자가 두 번 추가될 경우에 고객에게 과도하게 청구하는 일이 발생할 수 있을 것입니다.

미래의 문제

않은 것과 같은 변명입니다.

물론 2038 년에는 32 비트 컴퓨터가 많이 남아 있지 않을 것입니다. 고성능 컴퓨터들은 64 비트나 128 비트 시스템(또는 아마도 "보다 큰" 시스템)일 것입니다. 그러나 오늘날 64 비트 시스템을 생산하는 회사들조차도 32 비트의 날짜/시간 카운터를 여전히 사용하고 있으며, 2038 년에는 실패를 경험하게 될 것입니다.

2000 년 버그를 수정하는 동안 2038 문제를 최소화할 수 있는 좋은 프로그래밍 기술을 사용하는 것이 현명할 것입니다.

또 다른 문제

윤년은 왜 존재하며 그 법칙은 왜 그렇게 복잡할까요?

1 년의 길이는 365.2422 일이며 이는 지구가 태양의 주위를 한 바퀴 도는 데 걸리는 시간으로 측정됩니다(천문학자들은 매우 멀리 떨어져 있고 태양과 지구에 비해 상대적으로 거의 움직임이 없는 별들의 위치를 확인하여 시간을 매우 정확하게 계산할 수 있습니다).

1 년의 길이는 365 일보다 약 하루의 4 분의 1 이 더 길기 때문에 4 년마다 하루를 추가하지 않으면 계절이 조금씩 변화될 것입니다. 약 720 년 후에, 북반구에서는 12 월이 여름 날씨가 될 수도 있을 것입니다. 그러나 1 년은 정확히 365.25 일이 아니라 대략 365.2422 일이기 때문에, 4 년마다 윤년을 두어도 정확히 일치되지 않으므로, 400 년마다 3 번의 윤년을 건너뛰게 되는 것입니다.

이러한 수정안도 완벽하지 않으므로 지금부터 몇 천년 후에는 또 다른 조정안이 만들어져야 하며 그 법칙은 한층 더 기억하기 어려워질 것입니다.

응용프로그램과 Informix 제품들의 2000년 규정 준수

소개

21 세기가 가까워짐에 따라 시스템 관리자나 개발자들은 응용프로그램에서 날짜 특히 두 자리수 연도 처리와 관련된 문제의 심각성을 깨닫게 되었습니다. 이러한 문제에 보다 적극적으로 대처하기 위해 Informix 에서는 자사의 클라이언트와 서버 제품이 확실히 2000 년 규정을 준수하도록 작업해 왔습니다. 뿐만 아니라 Informix 에서는 응용프로그램의 2000 년 문제를 간단히 처리할 수 있는 DBCENTURY 환경 변수를 일부 제품과 함께 제공하고 있습니다.

있습니다. 2000 년 규정 준수에 대한 가장 최신의 목록은 Informix 의 웹 사이트 www.Informix.com 에서 확인할 수 있습니다.

이 글의 목적은 DBCENTURY 환경 변수로 해결된 2000 년 문제의 본질을 명확히 하고, DBCENTURY 의 한계와 이 변수로 해결할 수 없는 날짜 관련 문제들에 대해 논의하고, 상황을 해결하는 데 사용할 수 있는 대체 옵션들에 대해 소개하는 것입니다.

DBCENTURY로 해결된 날짜 관련 문제

DBCENTURY 는 사용자의 로그인 프로파일에서 설정되는 환경 변수입니다. 클라이언트와 서버 제품들은 응용프로그램에서 제공한 두 자리수 연도의 세기를 해석하기 위해 이 변수를 사용합니다.

예를 들어, 2003 년에 사용자가 두 개의 날짜 필드에 각각 '78'과 '08' 값을 입력하였다고 가정해봅시다. 만약 DBCENTURY 설정값이 발견되지 않으면, 시스템 시계의 현재 세기가 기본값이 됩니다. 그러나 DBCENTURY 가 'P'(과거)로 설정되어 있으면 Informix 서버는 연도를 '1978'과 '1908'로 해석하고, 'F'(미래)로 설정되어 있으면 '2078'과 '2008'로, 가장 일반적으로 사용되는 'C'(가장 가까운 때)로 설정되어 있으면 현재 연도와 가장 가까운 연도를 기본값으로 사용합니다. 예를 들어, '2003'년에는 위의 값이 '1978'과 '2008'로 해석되지만, 이 값들을 2040 년에 입력한다면 각각 '2078'과 '2008'로 해석됩니다.

따라서 응용프로그램의 모든 날짜 데이터들을(두 자리수 연도) 일정하게 해석할 수 있다면, 클라이언트 환경에 간단히 DBCENTURY 변수를 설정함으로써 다음 세기에 커다란 문제없이 응용프로그램을 사용할 수 있을 것입니다. 데이터베이스 서버가 세기를 정확히 해석하게 될 것이며, 데이터가 데이터베이스에 영구히 저장될 것이고, 응용프로그램에서 이루어지는 차후의 처리, 질의 및 보고에 대해 정확한 세기 값을 사용할 수 있을 것입니다.

DBCENTURY의 한계

DBCENTURY 는 클라이언트 환경에서 설정하는 환경 변수입니다. 그러므로 전체 구현 과정을 통해 클라이언트 환경 설정값들이 일관성을 갖는 것이 중요합니다. 그러나 다른 비헤이비어를 수용하기 위해 또 다른 데이터 필드를 필요로 할 경우에는(예를 들면, 어떤 필드는 'P' 설정값만을 필요로 하고 다른 필드는 'F' 설정값만을 필요로 하는 경우), DBCENTURY 에서 해결책을 찾을 수 없습니다.

만약 실행 시 해석되는 분할화 표현식(fragmentation expressions), 제한 조건, 기본값 등에서 데이터베이스가 두 자리수 연도를 사용한다면, 서로 다른 DBCENTURY 설정값들에 따라 서로 다른 값들을 얻게 될 것입니다. 다음의 분활화 표현식을 예로 들어 보겠습니다:

```
date_field < '01/01/00' IN DBSPACE1,

    : : : : : :

    REMAINDER IN DBSPACE
```

위의 경우에는, 서로 다른 DBCENTURY 설정값들에 대해 서로 다른 분할 영역(fragment)에 기록들이 삽입될 것입니다. DBCENTURY 설정값은 서로 다른데 똑같은 질의를 하는 사용자들은 특히 날짜와 관련된 필터에서 다른 결과를 보게 될 가능성이 있습니다. 예를 들어, 데이터를 로드하거나 언로드하며 DBDATE=DMY2 를 사용하는 모든 사용자 백업 프로그램은 업로드하는 과정에서 잘못된 세기 정보를 삽입할 수도 있을 것입니다. 그러한 모든 상황에서는 네 자리수 연도를 사용하는 것이 좋습니다.

DBCENTURY에서 지원되지 않는 날짜 관련 문제

만약 응용프로그램에서 날짜 관련 값을 저장하는 데 항상 'DATE' 데이터 형식을 사용하는 것이 아니라면 DBCENTURY 는 완전한 해결책이 아닙니다.
응용프로그램에서 C-ISAM 과 같이 'DATE' 데이터 형식을 지원하지 않는 Informix 제품을 사용한다면 DBCENTURY 는 부적합합니다. Informix 에 대해 다른 업체로부터 제공된 자체 드라이버를 시스템에서 사용할 경우에는, 해당 업체가 DBCENTURY 를 제품에 통합할 것인지를 확인하고 그 일정을 확실하게 보장받아 두는 것이 좋을 것입니다.

만약 응용프로그램이 현재 날짜를 기준으로 50 년 이전이나 이후보다 더 확장되는 날짜 데이터를 다루고, 두 자리수 연도를 사용한다면, DBCENTURY 는 유효한 일부 설정값에 대해 세기를 정확하게 해석하지 못할 수도 있습니다. DBCENTURY 가 2001 년에 'C'로 설정되었을 때 사용자가 다음과 같은 값을 입력한다고 가정해봅시다:

Date of birth = 01/01/49

Date of next increment = 01/01/02

여기에서 출생년도는 '2049'로 부정확하게 해석될 것입니다.

응용프로그램이 내부 'DATE' 형식에서 두 자리수 연도의 텍스트나 정수 형식으로 데이터를 변환한다면, 그리고 이 값들을 사용하여 날짜를 계산하거나 비교한다면, 이 구문 이면에 있는 논리(logic)도 변경해야 할 것입니다.

응용프로그램이 위의 상황 중 한 가지에 해당된다면, DBCENTURY 는 2000 년 문제에 대한 완전한 해결책이 될 수 없습니다.

Informix 응용프로그램에 대해 사용 가능한 대체 옵션

응용프로그램이 DBCENTURY 기능을 효과적으로 사용하지 못한다면, 2000 년 규정 준수를 위해 응용프로그램을 변경해야 할 것입니다. 가장 간단한 해결 방법은 응용프로그램을 수정하여 항상 'DATE' 필드에서 네 자리수 연도를 다루도록 하는 것입니다. 그러면 Informix 의 클라이언트와 서버 제품은 세기를 해석하지 않아도 되고, 다른 데이터 형식과 마찬가지로 날짜 데이터에 대해서도 저장과 검색 관리만 하면 됩니다. 그러나 이것이 항상 비용이 가장 적게 드는 해결책은 아닙니다.

일반적인 2000 년 수정 프로젝트는 다음의 네 단계로 구성되어 있습니다:

- 평가
- 수정
- 테스트

- 구현

평가 단계는 응용프로그램이 2000 년 문제에 의해 실제로 영향을 받는지의 여부를 결정하는 데 초점을 맞추고 있습니다. 이 단계에서는 일반적으로 툴의 도움이 필요합니다. 대부분의 툴은 응용프로그램의 구성 요소들을 검색하여 영향을 받는 날짜 변수를 찾기 위해 '패턴 일치' 알고리즘을 적용합니다. Informix 응용프로그램들에 대해서는, 이 툴로 DB 스키마, 4GL, ESQL/C 및 기타 언어 소스 코드, 화면 레이아웃 파일 그리고 데이터베이스 자체의 데이터를 스캔해야 합니다. 다음은 권장 평가 항목들입니다:

- 2000년 문제 수정이 정말로 필요한 것인가
- DBCENTURY 환경 변수 사용으로 문제를 충분히 수정할 수 있는가
- 비용이 가장 적게 드는 수정 전략은 무엇인가
- 가장 믿을만한 테스트 전략은 무엇인가 등등

평가 단계의 최종 산물은 구체적인 프로젝트 계획입니다.

툴을 사용한 분석이 모두 100 퍼센트 정확하지는 않을 수도 있음을 염두에 두어야 합니다. 패턴 일치 알고리즘은 '거짓 부정(false negative)'과 '거짓 긍정(false positive)'의 두 가지 오류 유형으로 나타날 수 있습니다. 그러나 훌륭한 툴은 자체 성능을 파악하고 개선할 능력을 가지고 있을 것입니다. 만족할만한 정확도를 얻기 위해 때로는 분석을 반복할 필요가 있습니다.

수정 단계에서는 소스 코드, 데이터베이스 디자인 그리고 사용자 인터페이스 요소에 대해 필요한 수정 작업을 진행합니다. 응용프로그램의 아키텍처와 구조에 따라, 모든 날짜 데이터 요소가 네 자리수 연도 사용으로 변환되는 '날짜 확장(date expansion)' 전략이나 '날짜 윈도윙(date windowing)' 전략(특정한 실행 날짜에 대해 실행 시 논리적으로 세기가 결정됨) 또는 둘을 조합하여 사용할 수 있습니다. 이것은 일반적으로 자동화된 처리 과정이 아닙니다.

테스트 단계에서는 다음 사항을 확인합니다:

- 수정 작업에 의해 응용프로그램 기능이 변경되지 않았는가
- 응용프로그램이 2000년 규정을 준수하는가

복귀(regression) 테스트가 첫 번째 목표에 대한 자연스런 해답이 될 것입니다. 2000 년 문제의 부작용 중 긍정적인 것이 있다면 그것은 중요한 복귀 테스트 토대를 만들어 낸 자원을 정보 서비스(Information Services) 업체에게 제공했다는 것입니다. 2000 년 규정 준수의 데모를 보여주기 위해, 많은 '미래'의 날짜들을 시뮬레이트하는 동안 복귀 테스트가 실행됩니다. 20 세기 전체, 21 세기 전체 그리고 세기가 바뀌는 기간에 대해 트랜잭션을 테스트하는 이와 같은 반복적인 테스트

작업을 위해 특정한 데스트 사례를 구축해야 할 필요가 있습니다.

구현 단계는 수정하고 테스트를 거친 응용프로그램을 다시 제품으로 소개할 수 있도록 계획하는 과정입니다. 이와 같이 하려면 아직 2000 년 규정을 준수하지 않거나 IS 업체의 통제 능력을 벗어난 다른 응용프로그램으로부터 받거나 보내는 날짜 데이터를 여과시키는 임시 또는 영구적인 '연결 매체(bridge)'를 만들어야 할 것입니다.

맺음말

인포믹스 응용프로그램은 2000년 문제에 심각하게 영향을 받을 수도 있고 그렇지 않을 수도 있습니다. 일부 응용프로그램에서는 DBCENTURY를 사용하여 대부분의 문제를 해결할 수 있을 것입니다. 그러나 데이터 기반의 분석이 완료되고, 대부분의 적절한 권장 사항이 만들어져 구현됨을 확인할 수 있는 공식적인 응용프로그램 평가가 이루어져야 합니다. 2000년은 이제 그리 멀지 않았습니다.