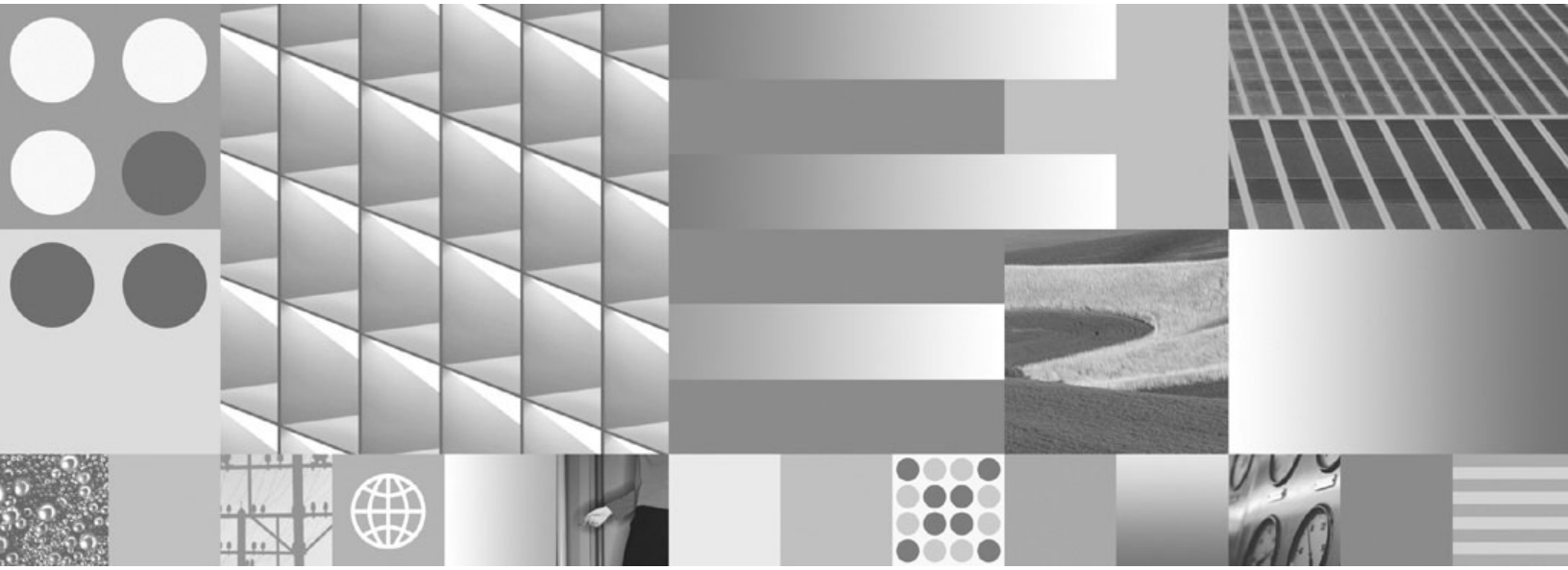
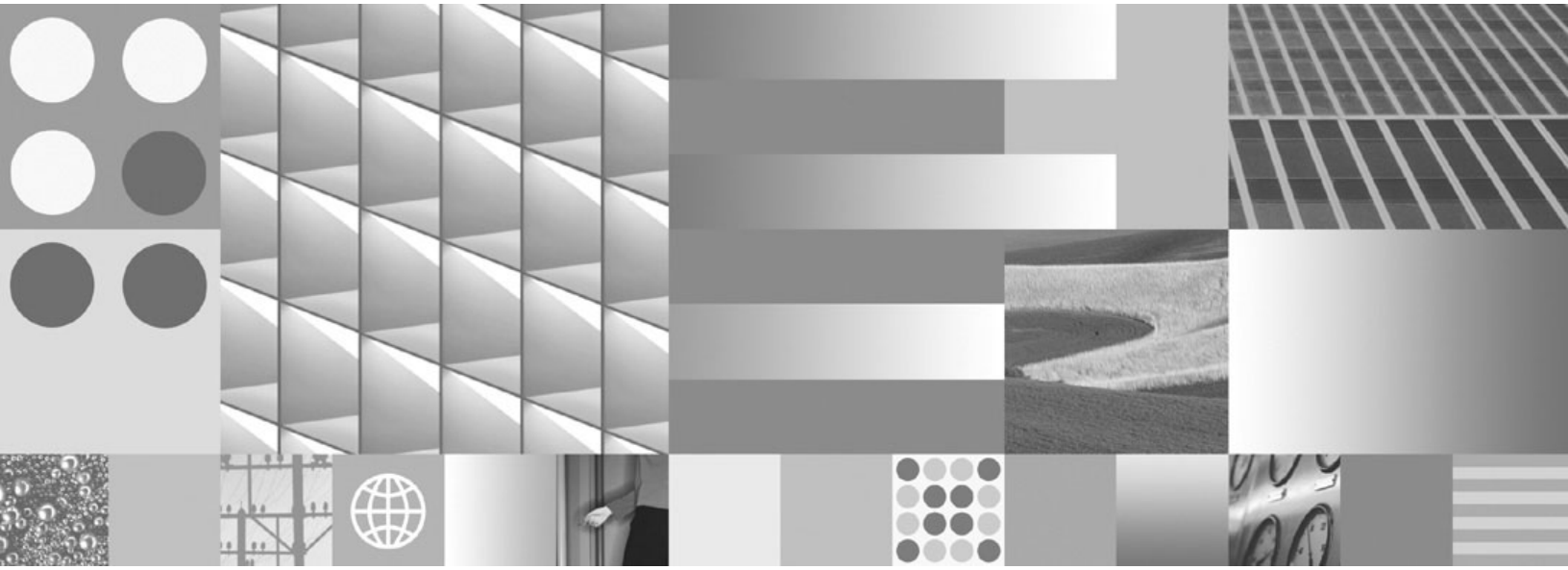
Linux, UNIX 및 Windows용
IBM DB2 9.7



데이터베이스 관리 개념 및 구성 참조서

SA30-3951-00

Linux, UNIX 및 Windows용
IBM DB2 9.7



데이터베이스 관리 개념 및 구성 참조서

SA30-3951-00

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 주의사항의 일반 정보를 읽으십시오.

개정판 주의사항

이 문서에는 IBM에서 소유하고 있는 정보가 있습니다. 이는 라이센스 계약에 따라 제공한 것이며 저작권의 보호를 받습니다. 이 책의 정보에는 제품 보증이 포함되지 않으며, 이 매뉴얼에서 제공된 어떠한 문장도 이와 같이 해석할 수 없습니다.

온라인으로 IBM 서적을 주문하거나 로컬 IBM 담당자를 통해 서적을 주문할 수 있습니다.

- 온라인으로 서적을 주문하려면 IBM Publications Center(www.ibm.com/shop/publications/order)로 이동하십시오.
- 로컬 IBM 담당자를 찾으려면 IBM Directory of Worldwide Contacts(www.ibm.com/planetwide)로 이동하십시오.

미국 또는 캐나다의 DB2 Marketing and Sales에서 DB2 서적을 주문하려면 1-800-IBM-4YOU (426-4968)로 전화하십시오.

IBM은 귀하가 IBM으로 보낸 정보를 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 사용하거나 배포할 수 있습니다.

목차

이 책에 대한 정보

*데이터베이스 관리 개념 및 구성 참조서*에는 데이터베이스 계획 및 설계에 대한 정보와 데이터베이스 오브젝트의 구현 및 관리에 대한 정보가 있습니다. 이 책에는 데이터베이스 구성 및 조정에 대한 참조 정보도 있습니다.

이 책의 사용자

이 책은 기본적으로 로컬 또는 원격 클라이언트가 액세스할 수 있는 데이터베이스를 설계, 구현 및 유지보수해야 하는 데이터베이스 및 시스템 관리자를 대상으로 합니다. DB2® 관계형 데이터베이스 관리 시스템의 관리 및 조작에 대해 알아야 하는 다른 사용자 및 프로그래머가 사용할 수도 있습니다.

이 책의 구성

이 책은 네 부분으로 구성되어 있습니다. 1부에서 3부까지는 DB2 제품에 대한 개념적인 개요로서 데이터 서버에 대한 일반 개념부터 DB2 데이터베이스를 공통적으로 포함하는 오브젝트에 대한 설명으로 구성되어 있습니다. 4부에는 참조 정보가 있습니다.

1부. 데이터 서버

이 절에서는 DB2 Data Server에 대해 간략하게 설명합니다. 여기에는 데이터 서버의 용량 관리와 AIX®의 64비트 환경에서의 대형 페이지 지원도 포함됩니다. 또한 한 대의 컴퓨터에 여러 개의 DB2 사본 실행에 대한 정보, 데이터베이스 시스템 관리에 유용한 자동 기능에 대한 정보, 인스턴스 설계, 작성 및 작업에 대한 정보 그리고 LDAP(Lighweight Directory Access Protocol) 서버 구성에 대한 선택적 정보도 제공합니다.

2부. 데이터베이스

이 절에서는 데이터베이스, 버퍼 풀, 테이블 스페이스 및 스키마의 설계, 작성 및 유지보수에 대해 설명합니다. 데이터베이스 파티션에 대한 자세한 정보는 *파티셔닝 및 클러스터링 안내서*의 내용을 참조하십시오.

3부. 데이터베이스 오브젝트

이 절에서는 테이블, 제한조건, 인덱스, 트리거, 시퀀스 및 뷰와 같은 데이터베이스 오브젝트의 설계, 작성 및 유지보수에 대해 설명합니다.

4부. 참조

이 절에는 환경 및 레지스트리 변수, 구성 매개변수로 데이터베이스 시스템을 구성 및 조정하기 위한 참조 정보가 있습니다. 또한 다양한 이름 작성 규칙과 SQL 및 XML 한계에 대한 설명도 나열되어 있습니다.

제 1 부 데이터 서버

제 1 장 DB2 Data Server

데이터 서버는 구조화된 정보를 안전하고 효율적으로 관리하는 데 필요한 소프트웨어 서비스를 제공합니다. DB2는 하이브리드 관계형 및 XML 데이터 서버입니다.

데이터 서버는 DB2 데이터베이스 엔진이 설치되어 있는 컴퓨터를 가리킵니다. DB2 엔진은 완전한 기능을 갖춘 강력한 데이터베이스 관리 시스템으로 실제 데이터베이스 사용을 기반으로 한 최적화된 SQL 지원 및 데이터 관리를 보조하는 도구가 포함되어 있습니다.

IBM에서는 모든 데이터 서버에 액세스할 수 있는 Data Server Client를 포함하여 다양한 데이터 서버 제품을 제공합니다. DB2 Data Server 제품의 전체 목록, 사용 가능한 기능 및 자세한 설명과 스펙은 http://www-306.ibm.com/software/data/db2/9/를 참조하십시오.

데이터 서버 용량 관리

데이터 서버 용량이 현재 또는 추후의 요구사항을 충족시키지 않는 경우 디스크 스페이스를 추가하고 추가 컨테이너를 작성하거나, 메모리를 추가하여 용량을 늘릴 수 있습니다. 이러한 간단한 방법으로 필요한 용량을 추가하지 못하는 경우에는 프로세서 또는 실제 파티션을 추가하십시오. 환경을 변경하여 시스템을 확장하는 경우에는 이러한 변경이 데이터 로딩 또는 백업 및 데이터베이스 리스토어와 같은 데이터베이스 프로시저에 미칠 수 있는 영향에 유의해야 합니다.

프로세서 추가

단일 프로세서를 가진 단일 파티션 데이터베이스 구성을 최대 용량까지 사용하는 경우 프로세서를 추가하거나 논리적 파티션을 추가하십시오. 프로세서를 추가하면 처리 능력이 향상되는 장점이 있습니다. SMP 시스템에서 프로세서는 메모리 및 스토리지 시스템 자원을 공유합니다. 모든 프로세서가 하나의 시스템에 있으므로 시스템 간의 통신 및 시스템 간 태스크 조정과 같은 추가적인 오버헤드 고려사항이 없습니다. 로드, 백업 및 리스토어와 같은 유틸리티에서 추가적인 프로세서를 사용할 수 있습니다.

주: Solaris 운영 체제와 같은 일부 운영 체제에서는 동적으로 프로세서를 온라인 및 오프라인으로 설정할 수 있습니다.

프로세서를 추가하는 경우 사용되는 프로세서 수를 판별하는 몇 가지 데이터베이스 구성 매개변수를 검토하고 수정하십시오. 다음 데이터베이스 구성 매개변수가 사용되는 프로세서 수를 판별하며, 매개변수를 갱신해야 합니다.

- 디폴트 등급(dft_degree)

- 최대 병렬 처리 수준(max_querydegree)
- 파티션 내 병렬 처리 사용(intra_parallel)

응용프로그램에서 병렬 처리를 수행하는 방법을 판별하는 매개변수도 평가해야 합니다.

통신에 TCP/IP를 사용하는 환경에서는 DB2TCPCONNMGRS 레지스트리 변수의 값을 검토하십시오.

추가적인 컴퓨터 추가

기존 파티션된 데이터베이스 환경이 있는 경우에는 추가적인 컴퓨터(단일 프로세서 또는 다중 프로세서) 및 스토리지 자원을 환경에 추가하여 처리 능력과 데이터 스토리지 용량을 늘릴 수 있습니다. 컴퓨터 간에 메모리와 스토리지 자원이 공유되지 않습니다. 이 방법을 선택하면 스토리지 및 컴퓨터 전반에서 데이터와 사용자 액세스 간의 균형을 유지할 수 있는 장점이 있습니다.

새 컴퓨터와 스토리지를 추가한 후에 START DATABASE MANAGER 명령을 사용하여 새 컴퓨터에 새 데이터베이스 파티션 서버를 추가할 수 있습니다. 추가한 각각의 새 데이터베이스 파티션 서버에 있는 인스턴스의 데이터베이스마다 새 데이터베이스 파티션이 작성 및 구성됩니다. 대부분의 경우 새 데이터베이스 파티션 서버 추가 후 인스턴스를 재시작할 필요가 없습니다.

64비트 환경에서 대형 페이지 지원을 사용 가능하도록 설정(AIX)

IBM® eServer™ pSeries® 시스템의 POWER4™ 프로세서(이상)에서는 일반적인 페이지 *크기*인 4KB 이외에도 16MB의 페이지 *크기*를 지원합니다. 집중 메모리 액세스가 필요하고 가상 메모리의 많은 부분을 사용하는 응용프로그램(예: AIX 64비트 Edition용 IBM DB2 버전 9.1)은 대형 페이지를 사용하여 성능을 향상시킬 수 있습니다.

주: 대형 페이지를 사용하면 자체 성능 조정 메모리 관리자가 전체 데이터베이스 메모리 사용을 자동으로 조정하지 않습니다. 따라서 상대적으로 정적 데이터베이스 메모리 요구사항이 포함된 잘 정의된 워크로드만 고려하면 됩니다.

1. vmo 명령 실행에 대한 자세한 지시사항은 AIX 매뉴얼을 참조하십시오.
2. 메모리 고정 및 대형 페이지 지원에 적합하도록 시스템을 구성하는 경우 매우 주의해야 합니다. 너무 많은 메모리를 고정하면 고정되지 않은 메모리 페이지에 대해 과도한 페이지 활동이 발생합니다. 대형 페이지에 너무 많은 실제 메모리를 할당하면 4KB의 페이지를 지원하기에 메모리가 부족한 경우 시스템 성능이 떨어집니다.
3. DB2_LARGE_PAGE_MEM 레지스트리 변수 설정은 메모리가 고정됨을 의미합니다.

AIX 운영 체제 명령을 사용하려 작업하려면 루트 권한이 있어야 합니다.

1. 다음 플래그와 함께 vmo 명령을 발행하여 대형 페이지 지원에 적합하도록 AIX 서버를 구성하십시오.

   ```
   vmo -r -o lgpg_size=<LargePageSize> -o lgpg_regions=<LargePages>
   ```

   여기서 <LargePageSize>는 하드웨어에서 지원되는 대형 페이지 *크기*를 바이트로 지정하고 <LargePages>는 예약할 대형 페이지 수를 지정합니다. 예를 들어 대형 페이지 지원에 25GB를 할당해야 하는 경우 다음과 같이 명령을 실행하십시오.

   ```
   vmo -r -o lgpg_size=16777216 -o lgpg_regions=1600
   ```

2. 다음 시스템 시동 시 이전에 실행한 vmo 명령이 적용되도록 bosboot 명령을 실행하십시오.
3. 서버가 실행되면 고정된 메모리에 대해 사용 가능하도록 설정하십시오.
   - 다음 플래그와 함께 vmo 명령을 발행하십시오.

     ```
     vmo -o v_pinshm=1
     ```

   - db2set 명령을 사용하여 다음과 같이 DB2_LARGE_PAGE_MEM 레지스트리 변수를 『DB』로 설정한 다음 DB2를 시작하십시오.

     ```
     db2set DB2_LARGE_PAGE_MEM=DB
     db2start
     ```

제 2 장 여러 DB2 사본 개요

버전 9 이상을 사용하여 동일한 컴퓨터에서 여러 DB2 사본을 설치하여 실행할 수 있습니다. DB2 사본은 동일한 컴퓨터의 특정 위치에서 하나 이상의 DB2 데이터베이스 제품 설치를 참조합니다. 각 DB2 버전 9 사본은 동일하거나 다른 코드 레벨에 있을 수 있습니다.

이 경우의 이점은 다음과 같습니다.

- 동일한 컴퓨터에서 다른 DB2 버전이 필요한 응용프로그램을 동시에 실행할 수 있는 기능
- 각각 다른 기능에 필요한 DB2 제품의 독립 사본을 실행할 수 있는 기능
- 프로덕션 데이터베이스를 DB2 제품의 최신 버전으로 이동하기 전에 동일한 컴퓨터에서 테스트하는 기능
- 독립 소프트웨어 벤더의 경우 DB2 서버 제품을 제품에 임베드하고 사용자로부터 DB2 데이터베이스를 숨기는 기능. COM+ 응용프로그램의 경우 COM+ 응용프로그램에는 한 번에 하나의 *Data Server Runtime Client*만 사용할 수 있으므로 *Data Server Runtime Client* 대신 응용프로그램과 함께 *IBM Data Server Driver for ODBC and CLI*를 사용 및 분배합니다. *IBM Data Server Driver for ODBC and CLI*에는 이러한 제한사항이 없습니다.

표 1에는 각 범주의 관련 항목이 나열되어 있습니다.

표 1. 여러 DB2 사본 정보에 대한 개요

범주	관련 항목
일반 정보 및 제한사항	• 8 페이지의 『디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본』 • *DB2 Server 설치*의 『동일한 컴퓨터에 있는 DB2 사본(Linux® 및 UNIX®)』 • *DB2 Server 설치*의 『동일한 컴퓨터(Windows®)에 있는 여러 DB2 사본』
업그레이드	• *DB2 버전 9.7로 업그레이드*의 『여러 DB2 사본과 함께 DB2 서버 업그레이드』 • *DB2 버전 9.7로 업그레이드*의 『DB2 서버 업그레이드(Windows)』 • *DB2 버전 9.7로 업그레이드*의 『DB2 32비트 서버를 64비트 시스템으로 업그레이드(Windows)』
설치	• *DB2 Server 설치*의 『DB2 서버 설치(Linux 및 UNIX)』 • *DB2 Server 설치*의 『DB2 서버 설치(Windows)』

표 1. 여러 DB2 사본 정보에 대한 개요 (계속)

범주	관련 항목
구성	• 12 페이지의 『여러 DB2 사본을 실행하는 DAS 설정』 • 13 페이지의 『여러 DB2 사본 사용 시 디폴트 인스턴스 설정(Windows)』 • *DB2 Server 설치*의 『설치 후 디폴트 DB2 및 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본 변경(Windows)』 • *DB2 Server 설치*의 『여러 사본을 사용하여 IBM Data Server Client 연결』 • *Call Level Interface Guide and Reference, Volume 1*의 『Selecting a different DB2 copy for your Windows CLI application』 • *명령어 참조서*의 『dasupdt - Update DAS 명령』
관리	• 18 페이지의 『DB2 사본 갱신(Windows)』 • 16 페이지의 『DB2 사본 갱신(Linux 및 UNIX)』 • *DB2 Server 설치*의 『기존 DB2 사본으로 작업』 • *DB2 Server 설치*의 『시스템에 설치된 DB2 제품 나열(Linux 및 UNIX)』 • *DB2 Server 설치*의 『시스템에서 실행되는 DB2 서비스(Windows)』 • *DB2 Server 설치*의 『DB2 파일의 링크 작성』 • *명령어 참조서*의 『db2iupdt - 인스턴스 갱신 명령』 • *명령어 참조서*의 『db2swtch - 디폴트 DB2 사본 명령 전환』 • *관리 API 참조서*의 『db2SelectDB2Copy API - 응용프로그램에서 사용하는 DB2 사본 선택』
설치 제거	• *DB2 Server 설치*의 『DB2 사본 제거(Linux, UNIX 및 Windows)』 • *DB2 Server 설치*의 『db2_deinstall 또는 doce_deinstall 명령을 사용하여 DB2 제품 제거(Linux 및 UNIX)』

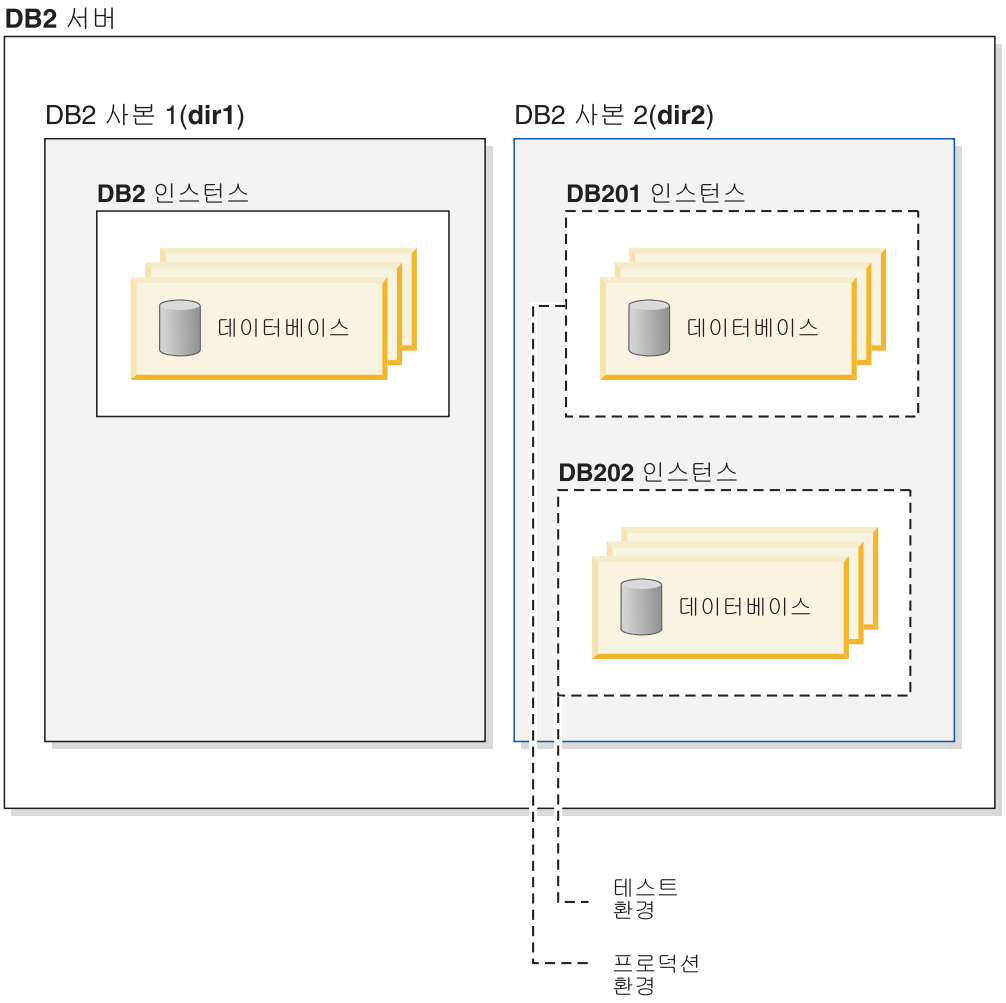
디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본

하나의 컴퓨터에 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본 이외에 여러 DB2 사본이 있을 수 있습니다. 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본은 클라이언트 응용프로그램이 디폴트로 데이터베이스와 상호 작용하는 데 필요한 ODBC 드라이버, CLI 드라이버 및 .NET Data Provider 코드를 갖는 방식입니다.

버전 9.1 이상에서 IBM 데이터베이스 클라이언트 인터페이스 사본의 코드는 DB2 사본과 함께 포함되어 있습니다. 버전 9.5 이상을 사용할 경우 새 제품을 선택하여 설치할 수 있으며 이 제품에는 클라이언트 응용프로그램이 데이터베이스와 상호 작용하도록 허용하는 데 필요한 코드가 있습니다. 이 제품은 IBM Data Server Driver Package(DSDRIVER)입니다. 버전 9.5 이상을 사용할 경우 DB2 사본 설치와 별도로 IBM Data Server Driver 사본에 DSDRIVER를 설치할 수 있습니다.

버전 9.1 이후로 컴퓨터에 다중 DB2 사본을 설치할 수 있습니다. 버전 9.5 이후로는 컴퓨터에 다중 IBM 데이터베이스 클라이언트 인터페이스 사본 및 다중 DB2 사본을 설치할 수 있습니다. 새 DB2 사본 또는 새 IBM Data Server Driver 사본을 설치하는 동안 디폴트 DB2 사본 및 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본을 변경할 수 있는 기회가 있습니다.

다음 다이어그램에는 DB2 서버에 설치된 다중 DB2 사본이 표시되어 있습니다. 어떠한 조합의 DB2 데이터베이스 제품이든 여러 DB2 사본이 될 수 있습니다.



버전 8과 버전 9 이상의 사본은 동일한 컴퓨터에 공존할 수 있지만 버전 8이 디폴트 DB2 및 IBM 데이터베이스 클라이언트 인터페이스 사본이어야 합니다. 먼저 버전 9 이상으로 업그레이드하거나 버전 8 사본을 설치 제거하지 않는 한, 설치 중에 디폴트 DB2 사본 또는 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본으로 버전 8 사본을 버전 9 이상 사본으로 변경할 수 없으며 나중에 디폴트 사본 전환 명령 db2swtch를 실행할 수도 없습니다. 시스템에 버전 8이 있을 때 db2swtch 명령을 실행하면 시스템에서 버전 8을 찾았기 때문에 디폴트 사본을 변경할 수 없음을 나타내는 오류 메시지가 수신됩니다.

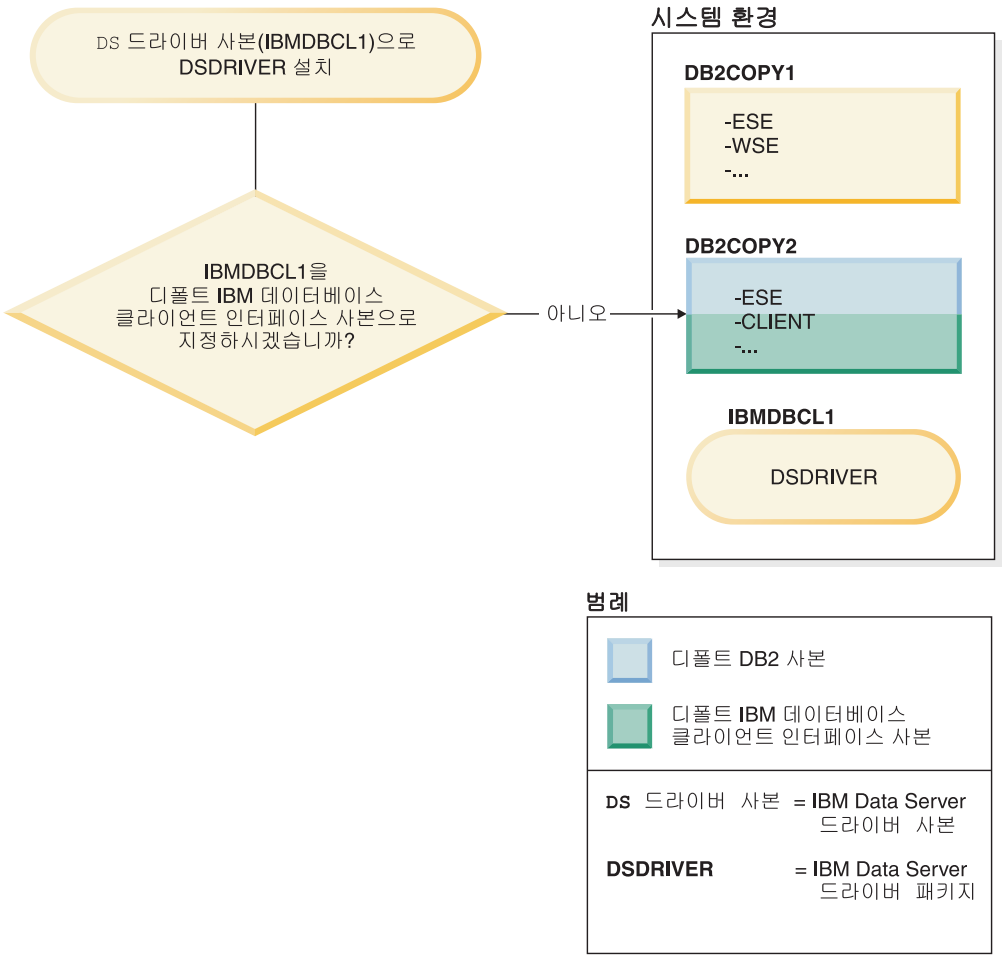
때때로 다중 DB2 사본 또는 다중 IBM Data Server Driver 사본 설치 후 디폴트 DB2 사본이나 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본을 변경하고자 할 수 있습니다. 버전 8이 설치되어 있는 경우에는 디폴트 DB2 사본을 변경하거나 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본을 변경하기 전에 제품을 설치 제거하거나 버전 9 이상으로 업그레이드해야 합니다.

클라이언트 응용프로그램은 항상 DSDRIVER가 설치된 디렉토리인 데이터 서버 드라이버 위치로 바로 이동하도록 선택할 수 있습니다.

디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본이었던 IBM Data Server Driver 사본 또는 DB2 사본을 설치 제거하는 경우 디폴트값이 관리됩니다. 선택한 디폴트 사본이 제거되고 새 디폴트값이 선택됩니다. 시스템에서 최종 DB2 사본이 아닌 디폴트 DB2 사본을 설치 제거하는 경우 먼저 디폴트를 다른 DB2 사본으로 전환하도록 사용자에게 요청합니다.

새 IBM 데이터베이스 클라이언트 인터페이스 사본 설치 시 디폴트 선택

버전 9.5 이후로는 두 개의 DB2 사본(DB2COPY1 및 DB2COPY2)을 설치한 시나리오를 검토합니다. DB2COPY2는 디폴트 DB2 사본이며 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본입니다.

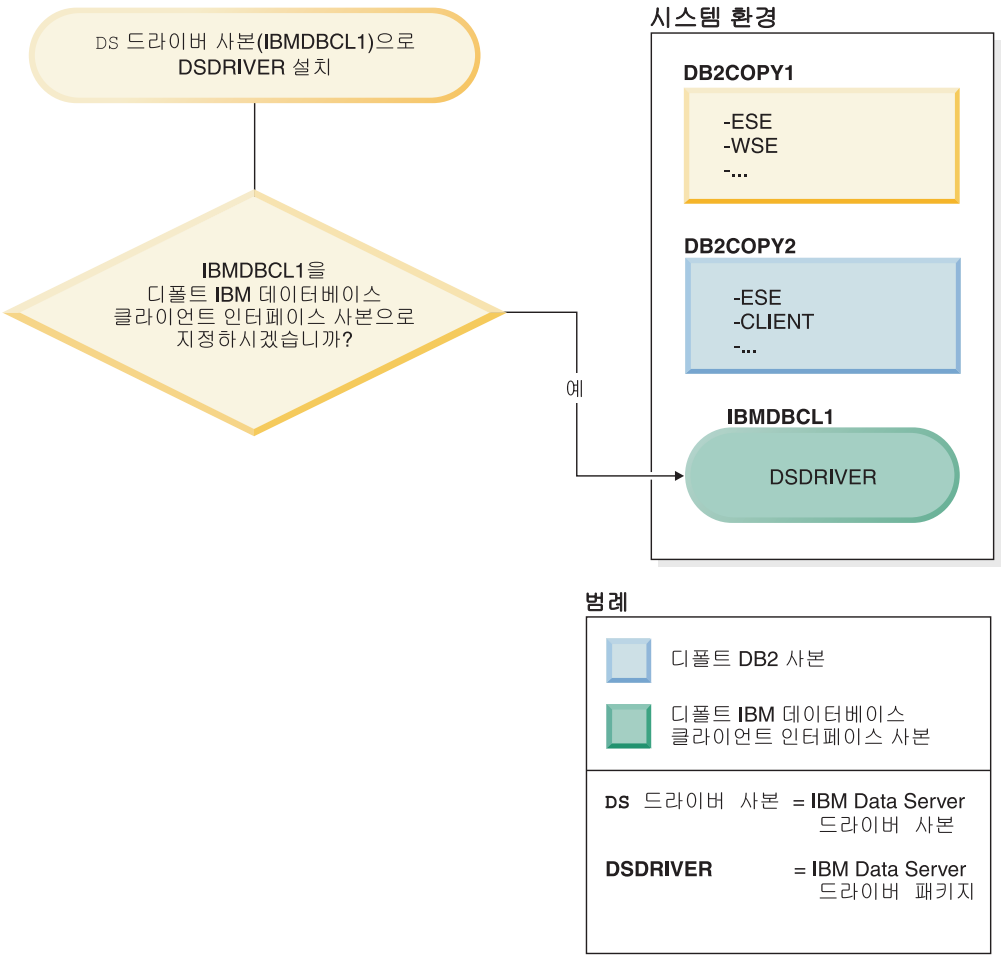


새 IBM Data Server Driver 사본에 IBM Data Server Driver Package(DSDRIVER)를 설치하십시오.

새 IBM Data Server Driver 사본(IBMDBCL1) 설치 시 새 IBM Data Server Driver 사본을 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본으로 설정할지 사용자에게 묻습니다.

『아니오』로 응답하면 DB2COPY2가 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본으로 유지됩니다. (그리고 디폴트 DB2 사본으로 유지됩니다.)

그러나 동일한 시나리오에서 새 IBM Data Server Driver 사본을 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본으로 사용하려면 질문에 『예』로 응답하십시오.



이 경우 IBMDBCL1이 디폴트 IBM 데이터베이스 클라이언트 인터페이스 사본이 됩니다. (DB2COPY2는 디폴트 DB2 사본으로 유지됩니다.)

여러 DB2 사본을 실행하는 DAS 설정

버전 9.1에서부터 동일한 컴퓨터에서 여러 DB2 사본을 실행할 수 있습니다. 이렇게 하면 DB2 Administration Server(DAS) 작동 방법에 영향을 줍니다. DAS는 동일한 컴퓨터에 많은 DB2 사본이 설치되어 있더라도 하나의 버전만 활성화하도록 제한된 데이터베이스 관리 프로그램 내의 고유한 구성요소입니다. 이러한 이유로 인해 다음 제한사항 및 기능 요구사항이 적용됩니다.

서버에는 하나의 DAS 버전만 있을 수 있으며 해당 DAS는 다음과 같이 인스턴스를 관리합니다.

- DAS가 버전 9.1 또는 버전 9.5에서 실행되는 경우 해당 DAS는 버전 8, 버전 9.1, 버전 9.5 인스턴스를 관리할 수 있습니다.
- DAS가 버전 8에서 실행되는 경우 해당 DAS는 버전 8 인스턴스만 관리할 수 있습니다. 버전 8 DAS를 업그레이드할 수 있습니다. 또는 해당 DAS를 삭제한 다음 새

로운 버전 9.5 DAS를 작성하여 버전 8 이상 인스턴스를 관리할 수 있습니다. 이는 제어 센터를 사용하여 인스턴스를 관리하는 경우에만 필요합니다.

동일한 컴퓨터에 설치된 DB2 사본 수에 관계없이 지정된 시간에 지정된 컴퓨터에서 하나의 DAS만 작성할 수 있습니다. 이러한 DAS는 동일한 컴퓨터에 있는 모든 DB2 사본에서 사용됩니다. 버전 9.1 이상에서 DAS는 현재 설치된 모든 DB2 사본에 속할 수 있습니다.

DAS가 버전 9.5 사본에서 실행 중인데 해당 DAS를 다른 버전 9.5 사본에서 실행하려고 하면 dasupdt 명령을 사용하십시오. DAS가 버전 8, 버전 9.1 또는 버전 9.5 사본에서 실행 중인데 해당 DAS를 버전 9.7 사본에서 실행하려는 경우 dasupdt를 사용할 수 없습니다. dasmigr 명령을 사용하여 DAS를 버전 9.7로 업그레이드해야 합니다.

또한 Windows 운영 체제에서 동일한 버전의 새로운 디폴트 DB2 사본에서 DAS를 실행해야 하는 경우 dasupdt 명령을 사용할 수 있습니다.

프로시저

DB2 사본 중 하나에서 DAS를 설정하려면 다음을 수행하십시오.

다음 조치 중 하나를 선택하십시오.

- DAS가 작성되지 않은 경우 DB2 사본 중 하나에서 DAS를 작성하십시오.
- DAS가 동일한 릴리스의 다른 DB2 사본에서도 실행되도록 dasupdt 명령을 사용하여 DAS를 갱신만 하십시오.
- dasmigr 명령을 사용하여 버전 8, 버전 9.1 또는 버전 9.5에서 버전 9.7 DAS로 업그레이드하십시오.

여러 DB2 사본 사용 시 디폴트 인스턴스 설정(Windows)

버전 9.1에서부터 DB2INSTANCE 환경이 사용자 환경이 사용하도록 현재 설정된 DB2 사본에 따라 설정됩니다. 현재 사본의 인스턴스로 명시적으로 설정하지 않으면 DB2INSTDEF 프로파일 레지스트리 변수를 사용하여 지정된 디폴트 인스턴스가 디폴트값으로 설정됩니다.

DB2INSTDEF은 사용 중인 현재 DB2 사본에 고유한 디폴트 인스턴스 변수입니다. 모든 DB2 사본에는 고유한 자체 DB2INSTDEF 프로파일 레지스트리 변수가 있습니다. 인스턴스 이름은 인스턴스가 작성된 시스템에서 고유해야 하며 데이터베이스 관리 프로그램은 기존 사본을 스캔하여 고유성을 보장합니다.

다음 지침을 수행하여 여러 DB2 사본 사용 시 디폴트 인스턴스를 설정하십시오.

- DB2INSTANCE가 특정 DB2 사본에 대해 설정되어 있지 않으면 DB2INSTDEF의 값이 해당 DB2 사본에 대해 사용됩니다. 이는 다음을 의미합니다.

  - DB2INSTANCE=ABC 및 DB2INSTDEF=XYZ, ABC는 사용되는 값입니다.
  - DB2INSTANCE가 *설정되어 있지 않으면* DB2INSTDEF=XYZ, XYZ가 사용됩니다.
  - DB2INSTANCE가 *설정되어 있지 않고* DB2INSTDEF가 *설정되어 있지 않으면* 유효한 DB2INSTANCE에 종속된 모든 응용프로그램 또는 명령이 작동하지 않습니다.
- db2envar.bat 명령 또는 db2SelectDB2Copy API를 사용하여 DB2 사본을 전환할 수 있습니다. 모든 환경 변수(예: PATH, INCLUDE, LIB 및 DB2INSTANCE)는 적절하게 설정되지만 제대로 설정되었는지 확인해야 합니다.

  주: db2envar.bat 명령을 사용해도 환경 변수 설정과 완전히 동일하지 않습니다. db2envar.bat 명령은 자신이 속한 DB2 사본을 판별하고 PATH 환경 변수 앞에 이 DB2 사본의 경로를 추가합니다.

  동일한 컴퓨터에 여러 DB2 사본이 있는 경우 PATH 환경 변수는 이러한 사본 중 하나인 DEFAULT COPY만 나타낼 수 있습니다. 예를 들어 DB2COPY1이 c:₩sqllib₩bin에 있고 디폴트 사본이면 DB2COPY2는 d:₩sqllib₩bin에 있습니다. 일반 명령 창에서 DB2COPY2를 사용하려면 해당 명령 창에서 d:₩sqllib₩bin₩db2envar.bat를 실행합니다. 이렇게 하면 d:₩sqllib₩bin에서 실행 파일을 선택하도록 해당 명령 창에 대해 PATH(및 일부 다른 환경 변수)가 조정됩니다.
- DB2INSTANCE는 사용 중인 DB2 사본 아래에서만 유효합니다. 그러나 db2envar.bat 명령을 사용하여 사본을 전환하는 경우 DB2INSTANCE는 처음에 전환한 DB2 사본에 대한 DB2INSTDEF 값으로 갱신됩니다.
- DB2INSTANCE는 해당 DB2 사본에서 실행 중인 응용프로그램에서 사용될 현재 DB2 인스턴스입니다. 사본 간에 전환하면 디폴트로 DB2INSTANCE가 해당 사본에 대한 DB2INSTDEF 값으로 변경됩니다. DB2INSTDEF는 모든 인스턴스가 현재 사본에 있으므로 하나의 사본 시스템에서는 크게 유용하지 않습니다. 그러나 다른 인스턴스가 설정되지 않은 경우 디폴트 인스턴스로 적용됩니다.
- SET VARIABLE=<variable_name>을 사용하여 전역 프로파일 레지스트리 변수를 지정하지 않으면 이러한 모든 변수는 DB2 사본에 고유합니다.

데이터베이스 관리 프로그램의 다중 인스턴스

단일 서버에서 데이터베이스 관리 프로그램의 다중 인스턴스를 작성할 수 있습니다. 이는 실제 컴퓨터에서 동일한 제품의 여러 인스턴스를 작성하고 이들 인스턴스를 동시에 실행할 수 있음을 의미합니다. 이를 통해 유연하게 환경을 설정할 수 있습니다.

주: 두 개의 서로 다른 DB2 사본에서 동일한 인스턴스 이름을 사용할 수 없습니다.

다중 인스턴스를 사용하여 다음과 같은 환경을 작성할 수 있습니다.

- 개발 환경을 프로덕션 환경에서 분리합니다.
- 환경이 적용될 특정 응용프로그램에 적합하게 개별적으로 각 환경을 조정합니다.
- 관리자로부터 중요한 정보를 보호합니다. 예를 들어, 다른 인스턴스 소유자가 급여 데이터를 볼 수 없도록 본인 소유의 인스턴스에서 급여 데이터베이스를 보호할 수 있습니다.

주:

- (UNIX 운영 체제에만 해당) 둘 이상의 인스턴스 간 환경 충돌이 일어나지 않게 하려면 각 인스턴스 홈 디렉토리가 로컬 파일 시스템에 있어야 합니다.
- (Windows 플랫폼에만 해당) 인스턴스는 노드 디렉토리에서 로컬 또는 리모트로 카탈로그됩니다. 사용자의 디폴트 인스턴스는 DB2INSTANCE 환경 변수를 통해 정의됩니다. 다른 인스턴스에 접속하여 인스턴스 레벨에서만 수행할 수 있는 유틸리티 태스크 및 유지보수를 수행할 수 있습니다(예: 데이터베이스 작성, 응용프로그램 강제 해제, 데이터베이스 모니터링 또는 데이터베이스 관리 프로그램 구성 갱신). 디폴트 인스턴스에 포함되지 않은 인스턴스에 접속하려고 하면 해당 인스턴스와 통신할 방법을 판별하기 위해 노드 디렉토리가 사용됩니다.
- (모든 플랫폼에 해당) DB2 데이터베이스 프로그램 파일은 실제로 한 위치에 저장되고, 작성되는 인스턴스마다 프로그램 파일이 중복되지 않도록 각각의 인스턴스는 해당 인스턴스가 속하는 사본을 가리킵니다. 여러 개의 관련 데이터베이스를 단일 인스턴스에 배치할 수 있습니다.

다중 인스턴스(Windows)

동일한 컴퓨터에서 데이터베이스 관리 프로그램의 다중 인스턴스를 실행할 수 있습니다. 데이터베이스 관리 프로그램의 각 인스턴스는 자체 데이터베이스를 유지보수하고 고유한 데이터베이스 관리 프로그램 구성 매개변수를 갖습니다.

주: 또한 인스턴스는 컴퓨터에서 서로 다른 데이터베이스 관리 프로그램 레벨에 있는 DB2 사본에 속할 수 있습니다.

데이터베이스 관리 프로그램의 인스턴스는 다음 항목으로 구성됩니다.

- 인스턴스를 나타내는 Windows 서비스. 서비스 이름은 인스턴스 이름과 동일합니다. 서비스 패널의 서비스 표시 이름은 인스턴스 이름이며 이름 앞에 『DB2 - 』 문자열이 붙습니다. 예를 들어, 『DB2』라는 이름의 인스턴스에는 표시 이름이 『DB2 - <DB2 사본 이름> - DB2』인 『DB2』라는 Windows 서비스가 있습니다.

  주: 클라이언트 인스턴스에는 Windows 서비스가 작성되지 않습니다.
- 인스턴스 디렉토리. 이 디렉토리에는 데이터베이스 관리 프로그램 구성 파일, 시스템 데이터베이스 디렉토리, 노드 디렉토리, 데이터베이스 연결 서비스(DCS) 디렉토리, 인스턴스와 연관된 모든 진단 로그 및 덤프 파일이 들어 있습니다. 인스턴스 디렉토리

는 Windows 계열 운영 체제의 개정판에 따라 다릅니다. Windows에서 디폴트 디렉토리를 검증하려면 db2set DB2INSTPROF 명령을 사용하여 **DB2INSTPROF** 환경 변수 설정을 점검하십시오. **DB2INSTPROF** 환경 변수를 변경하여 디폴트 인스턴스 디렉토리를 변경할 수도 있습니다. 예를 들어, 디렉토리를 c:₩DB2PROFS로 설정하려면 다음을 수행하십시오.

  – db2set.exe -g 명령을 사용하여 **DB2INSTPROF**를 c:₩DB2PROFS로 설정하십시오.

  – DB2ICRT.exe 명령을 실행하여 인스턴스를 작성하십시오.

- Windows 운영 체제에서 인스턴스를 작성하는 경우 인스턴스 디렉토리 및 db2cli.ini 파일과 같은 사용자 데이터 파일의 디폴트 위치는 다음 디렉토리입니다.

  – Windows XP 및 Windows 2003 운영 체제: Documents and Settings₩All Users₩Application Data₩IBM₩DB2₩Copy Name

  – Windows 2008 및 Windows Vista 이상의 운영 체제: ProgramData₩IBM₩DB2₩Copy Name

  여기서 Copy Name은 DB2 사본 이름을 나타냅니다.

  주: db2cli.ini 파일의 위치는 Microsoft® ODBC 드라이버 관리자 사용 여부, 사용되는 데이터 소스 이름(DSN)의 유형, 설치 중인 클라이언트 또는 드라이버의 유형, 레지스트리 변수 **DB2CLIINIPATH**가 설정되었는지 여부에 따라 변경됩니다. 자세한 정보는 *Call Level Interface Guide and Reference, Volume 1*의 『db2cli.ini initialization file』을 참조하십시오.

DB2 사본 갱신(Linux 및 UNIX)

DB2 사본을 갱신하면 기존 DB2 사본 및 해당 사본에서 실행 중인 모든 인스턴스가 갱신되거나 새 DB2 사본이 설치되고 설치 후 이러한 새 사본에서 실행되는 인스턴스가 선택적으로 갱신됩니다.

시작하기 전에

- 루트 액세스 권한이 있는지 확인하십시오.
- DB2 사본에 추가 제품이 설치되어 있는지 여부에 따라 범용 FixPack 또는 제품별 FixPack을 다운로드하여 압축을 해제하십시오. FixPack 및 갱신하려는 DB2 사본의 릴리스는 동일해야 합니다. 자세한 내용은 *DB2 Server 설치*의 『FixPack을 설치하기 전에』를 참조하십시오.

이 태스크에 대한 정보

DB2 사본 갱신은 DB2 사본 버전 9.1 이상의 경우 동일한 릴리스의 DB2 사본만 참조합니다. DB2 사본을 갱신하면 추가 기능을 설치할 수도 있습니다.

DB2 버전 8, 버전 9.1 또는 버전 9.5 사본이 있으면 이러한 사본을 이전 릴리스에서 DB2 버전 9.7로 갱신할 수 없습니다. 해당 사본을 업그레이드해야 합니다. *DB2 버전 9.7로 업그레이드*의 『DB2 서버 업그레이드(Linux 및 UNIX)』를 참조하십시오.

제한사항

비루트 설치 사본이 있는 경우 비루트 설치 사본을 갱신하는 방법에 대한 자세한 내용은 *DB2 Server 설치*의 『비루트 설치에 FixPack 적용』을 참조하십시오.

- 동시에 하나 이상의 DB2 사본을 갱신할 수 없습니다. 동일한 컴퓨터에 설치될 수 있는 기타 DB2 사본을 갱신하려면 설치를 다시 실행해야 합니다.
- 버전 7과 버전 9.1 이상은 함께 설치할 수 없습니다. 버전 9.1 이상의 새 DB2 사본을 설치하려면 버전 7을 설치 제거해야 합니다.

프로시저

DB2 사본을 갱신하려면 다음을 수행하십시오.

1. 루트로 로그온하십시오.
2. 모든 DB2 프로세스를 중지하십시오. 자세한 내용은 *DB2 Server 설치*의 『모든 DB2 프로세스 중지(Linux 및 UNIX)』를 참조하십시오.
3. 다음 선택사항 중 하나를 사용하여 각 DB2 사본을 갱신하십시오.
   - 기존 DB2 사본을 갱신하고 해당 DB2 사본에서 실행 중인 모든 인스턴스를 갱신하려면 installFixPack 명령을 발행하십시오. 이러한 명령을 사용하여 추가 기능을 설치할 수 없습니다. 설치 후 태스크에 대한 자세한 내용은 *DB2 Server 설치*의 『DB2 데이터베이스 제품을 갱신하도록 FixPack 설치(Linux 및 UNIX)』를 참조하십시오.
   - 새 DB2 사본을 설치하고 설치 후 기존 DB2 사본에서 실행 중인 인스턴스를 새 사본으로 선택적으로 갱신하려면 db2setup 명령을 발행한 다음 **제품 설치** 패널에서 **새로 설치**를 선택합니다. 새 사본을 설치하기 위해 응답 파일 설치를 수행하거나 db2_install 명령을 발행하여 새 위치를 설치 경로로 지정할 수도 있습니다. 이러한 옵션 중 하나를 사용하면 추가 기능을 설치할 수도 있습니다.
   - 기존 DB2 사본에 기능을 추가하려면 **제품 설치** 패널에서 **기존 제품으로 설치**를 선택합니다. 그런 다음 **새 기능 추가** 조치로 갱신하려는 DB2 사본을 선택합니다. 이 조치는 DB2 사본이 설치 이미지와 동일한 릴리스 레벨에 있는 경우에만 사용할 수 있습니다. 기능을 추가하기 위해 응답 파일 설치를 수행하거나 db2_install 명령을 발행할 수 있습니다.
4. 새 DB2 사본을 설치한 경우 db2iupdt 명령을 사용하여 새 사본 아래에서 인스턴스를 실행하려고 하는 동일한 릴리스의 다른 DB2 사본에서 실행 중인 인스턴스를 갱신하십시오. 다음 표는 인스턴스 갱신의 여러 가지 예를 보여줍니다.

인스턴스	**DB2** 사본	다른 사본으로 갱신하는 예
db2inst1	/opt/IBM/db2/V9.1/	cd /opt/IBM/db2/V9.1_FP3/instance ./db2iupdt db2inst1
db2inst2	/opt/IBM/db2/V9.5FP2/	cd /home/db2/myV9.5_FP1/instance ./db2iupdt -D db2inst2[a]
db2inst3	/opt/IBM/db2/V9.7/	cd /home/db2/myV9.7/instance ./db2iupdt -k db2inst3[b]

주:

a. 높은 릴리스 레벨 사본에서 낮은 릴리스 레벨 사본으로 인스턴스를 갱신하려면 -D 매개변수를 사용합니다.

b. 높은 레벨의 인스턴스 유형인 DB2 사본으로 갱신하는 중 현재 인스턴스 유형을 유지하려면 -k 매개변수를 사용합니다. WSE에서 ESE로 갱신한 경우 이러한 매개변수를 사용하지 않고 인스턴스를 갱신하면 인스턴스 유형 wse가 ese로 변환됩니다.

결과

DB2 사본을 설치 또는 갱신하면 db2iupdt 명령을 발행하여 새 DB2 사본에서 실행되도록 동일한 릴리스의 다른 DB2 사본에서 실행되는 인스턴스를 항상 갱신할 수 있습니다.

DB2 사본 갱신(Windows)

DB2 사본을 갱신하면 기존 DB2 사본 및 해당 사본에서 실행 중인 모든 인스턴스가 갱신되거나 새 DB2 사본이 설치되고 설치 후 이러한 새 사본에서 실행되는 인스턴스가 선택적으로 갱신됩니다.

시작하기 전에

- 로컬 관리자 권한이 있는지 확인하십시오.
- DB2 사본에 추가 제품이 설치되어 있는지 여부에 따라 범용 FixPack 또는 제품별 FixPack을 다운로드하여 압축을 해제하십시오. FixPack 및 갱신하려는 DB2 사본의 릴리스는 동일해야 합니다. 자세한 내용은 *DB2 Server 설치*의 『FixPack을 설치하기 전에』를 참조하십시오.

이 태스크에 대한 정보

DB2 사본 갱신은 DB2 사본 버전 9.1 이상의 경우 동일한 릴리스의 DB2 사본만 참조합니다. DB2 사본을 갱신하면 추가 기능을 설치할 수도 있습니다.

기존 DB2 사본을 갱신하도록 선택하면 버전 8, 버전 9.1 또는 버전 9.5 사본에 **갱신** 조치 이외에도 **업그레이드** 조치를 사용할 수 있습니다. 이들 사본을 이전 릴리스에서

DB2 버전 9.7로 갱신할 수 없으며, 업그레이드해야 합니다. 『*DB2 버전 9.7로 업그레이드*의 DB2 서버 업그레이드(Windows)』를 참조하십시오.

제한사항

- 동일한 릴리스의 인스턴스는 낮은 릴리스 레벨 사본에서 높은 릴리스 레벨 사본으로만 갱신할 수 있습니다. 높은 릴리스 레벨 사본에서 낮은 릴리스 레벨 사본으로 인스턴스를 갱신할 수 없습니다.
- 동시에 하나 이상의 DB2 사본을 갱신할 수 없습니다. 동일한 컴퓨터에 설치될 수 있는 기타 DB2 사본을 갱신하려면 설치를 다시 실행해야 합니다.
- 버전 7과 버전 9.1 이상은 함께 설치할 수 없습니다. 버전 9.1 이상의 새 DB2 사본을 설치하려면 버전 7을 설치 제거해야 합니다.
- 동일한 Windows x64 컴퓨터에 32비트 DB2 Data Server와 64비트 DB2 Data Server를 함께 설치할 수 없습니다. 버전 8의 32비트 x64 DB2 설치에서 버전 9.7의 64비트 설치로 바로 업그레이드할 수 없습니다. 자세한 내용은 『*DB2 버전 9.7로 업그레이드*의 DB2 32비트 서버에서 64비트 시스템으로 업그레이드(Windows)』를 참조하십시오.

프로시저

DB2 사본을 갱신하려면 다음을 수행하십시오.

1. 로컬 관리자 권한을 사용하여 사용자로 로그온하십시오.
2. 모든 DB2 인스턴스, 서비스 및 응용프로그램을 중지하십시오. 자세한 내용은 *DB2 Server 설치*의 『모든 DB2 인스턴스, 서비스 및 응용프로그램 중지(Windows)』를 참조하십시오.
3. setup.exe를 실행하여 DB2 마법사를 시작하여 DB2 사본을 설치하십시오. 다음 중에서 선택할 수 있습니다.
   - 기존 DB2 사본을 갱신하고 해당 DB2 사본에서 실행 중인 모든 인스턴스를 갱신하면 **제품 설치** 패널에서 **기존 제품으로 설치**를 선택합니다. 그런 다음 **갱신** 조치를 통해 갱신할 DB2 사본을 선택합니다. 이러한 조치를 통해 추가 기능을 설치할 수 없습니다. 설치 후 태스크에 대한 자세한 내용은 *DB2 Server 설치*의 『FixPack의 설치 후 태스크(Windows)』를 참조하십시오.
   - 새 DB2 사본을 설치하고 설치 후 기존 DB2 사본에서 실행 중인 인스턴스를 새 사본으로 선택적으로 갱신하려면 **제품 설치** 패널에서 **새로 설치**를 선택합니다. 이 옵션을 사용하면 추가 기능을 설치할 수 있습니다.
   - 기존 DB2 사본에 기능을 추가하려면 **제품 설치** 패널에서 **기존 제품으로 설치**를 선택합니다. 그런 다음 **새 기능 추가** 조치로 갱신하려는 DB2 사본을 선택합니다. 이 조치는 DB2 사본이 설치 이미지와 동일한 릴리스 레벨에 있는 경우에만 사용할 수 있습니다.

4. 새 DB2 사본을 설치한 경우 db2iupdt 명령을 사용하여 새 사본 아래에서 인스턴스를 실행하려고 하는 동일한 릴리스의 다른 DB2 사본에서 실행 중인 인스턴스를 갱신하십시오. 다음 표는 인스턴스 갱신의 여러 가지 예를 보여줍니다.

인스턴스	**DB2** 사본	다른 사본으로 갱신하는 예
db2inst1	C:¥Program FilesWIBMWSQLLIB_91WBIN	cd D:WProgram FilesWIBMWSQLLIB_91_FP5WBIN db2iupdt db2inst1 /u: *user-name*,*password*
db2inst2	C:¥Program FilesWIBMWSQLLIB_97WBIN	cd D:WProgram FilesWIBMWSQLLIB_97WBIN db2iupdt db2inst2 /u: *user-name*,*password*

결과

DB2 사본을 설치 또는 갱신하면 db2iupdt 명령을 발행하여 새 DB2 사본에서 실행되도록 동일한 릴리스의 다른 DB2 사본에서 실행되는 인스턴스를 항상 갱신할 수 있습니다.

여러 인스턴스를 동시에 실행(Windows)

동일한 DB2 사본 또는 다른 DB2 사본에서 여러 인스턴스를 동시에 실행할 수 있습니다.

동일한 DB2 사본에서 여러 인스턴스를 동시에 실행하려면 명령행을 사용하여 다음과 같이 입력하십시오.

1. 다음과 같이 입력하여 시작하려는 다른 인스턴스의 이름으로 DB2INSTANCE 변수를 설정합니다.

   ```
   set db2instance=<another_instName>
   ```

2. db2start 명령을 입력하여 인스턴스를 시작합니다.

다른 DB2 사본에서 여러 인스턴스를 동시에 실행하려면 다음 방법 중 하나를 사용하십시오.

- 시작 → 프로그램 → IBM DB2 → *<DB2 사본 이름>* → 명령행 도구 → DB2 명령 창에서 DB2 명령 창을 사용합니다. 명령 창은 이미 선택한 특정 DB2 사본의 올바른 환경 변수로 설정되어 있습니다.
- 다음과 같이 명령 창에서 db2envar.bat를 사용합니다.
  1. 명령 창을 엽니다.
  2. 응용프로그램에서 사용하도록 하려는 DB2 사본의 완전한 경로를 사용하여 db2envar.bat 파일을 실행합니다.

     ```
     <DB2 Copy install dir>¥bin¥db2envar.bat
     ```

특정 DB2 사본으로 전환한 후 위의 "동일한 DB2 사본에서 여러 인스턴스를 동시에 실행" 절에서 지정한 방법을 사용하여 인스턴스를 시작합니다.

동일한 또는 다른 DB2 사본의 인스턴스로 작업

DB2 사본 또는 다른 DB2 사본에서 동시에 여러 인스턴스를 실행할 수 있습니다.

동일한 DB2 사본의 인스턴스에 대해 작업하려면 다음을 수행해야 합니다.

1. 인스턴스를 작성하거나 모든 인스턴스를 동일한 DB2 사본으로 업그레이드합니다.
2. 해당 인스턴스에 대해 명령을 발행하기 전에 작업하는 데 사용한 인스턴스의 이름으로 DB2INSTANCE 환경 변수를 설정합니다.

하나의 인스턴스가 다른 인스턴스의 데이터베이스에 액세스하도록 하지 않도록 인스턴스의 데이터베이스 파일이 해당 인스턴스 이름과 이름이 동일한 디렉토리 아래에 작성됩니다. 예를 들어 인스턴스 『DB2』를 위해 드라이브 C:에 데이터베이스를 작성하면 C:₩DB2 디렉토리 내에 데이터베이스 파일이 작성됩니다. 유사하게 인스턴스 TEST를 위해 드라이브 C:에 데이터베이스를 작성하면 C:₩TEST 디렉토리 내에 데이터베이스 파일이 작성됩니다. 디폴트로 이러한 환경 변수의 값은 DB2 제품이 설치된 드라이브 이름입니다. 자세한 정보는 *dftdbpath* 데이터베이스 관리 프로그램 구성 매개변수를 참조하십시오.

여러 DB2 사본과 함께 시스템의 인스턴스를 사용하여 작업하려면 다음 방법 중 하나를 사용하십시오.

- 시작 → 프로그램 → IBM DB2 → *<DB2 사본 이름>* → 명령행 도구 → 명령 창에서 명령 창을 사용합니다. 명령 창은 이미 선택한 특정 DB2 사본의 올바른 환경 변수로 설정되어 있습니다.
- 다음과 같이 명령 창에서 db2envar.bat를 사용합니다.
  1. 명령 창을 엽니다.
  2. 응용프로그램에서 사용하도록 하려는 DB2 사본의 완전한 경로를 사용하여 db2envar.bat 파일을 실행합니다.

     ```
     <DB2 Copy install dir>₩bin₩db2envar.bat
     ```

제 3 장 자동 컴퓨팅 개요

DB2 자동 컴퓨팅 환경에서는 구성, 치료, 최적화 및 보호 기능을 자체적으로 수행 가능합니다. 자동 컴퓨팅은 발생한 상황을 발견하고 이러한 상황에 응답함으로써 데이터베이스 관리자가 아니라 기술을 통해 컴퓨팅 환경을 관리하도록 지원합니다.

25 페이지의 『자동 기능』에서는 DB2 자동 컴퓨팅 개요를 구성하는 성능의 상위 수준 요약을 제공합니다. 다음 표는 제품의 자동 성능의 더욱 상세하고 범주화된 개요를 제공합니다.

표 2. 자동 컴퓨팅 정보 개요

범주	관련 항목
자체 성능 조정 메모리	• *문제점 해결 및 데이터베이스 성능 조정*의 『메모리 사용』 • *문제점 해결 및 데이터베이스 성능 조정*의 『자체 성능 조정 메모리』 • *문제점 해결 및 데이터베이스 성능 조정*의 『자체 성능 조정 메모리 개요』 • 787 페이지의 『auto_maint - 자동 유지보수』 • *데이터베이스 모니터링 안내서 및 참조서*의 『db_storage_path - 자동 스토리지 경로 모니터 요소』 • *데이터베이스 모니터링 안내서 및 참조서*의 『num_db_storage_paths - 자동 스토리지 경로 모니터 요소』 • *데이터베이스 모니터링 안내서 및 참조서*의 『tablespace_using_auto_storage - 자동 스토리지 모니터 요소 사용』 • 44 페이지의 『메모리 및 메모리 힙 구성』 • 47 페이지의 『에이전트, 프로세스 모델 및 메모리 구성 개요』 • 52 페이지의 『공유된 파일 핸들 테이블』 • 52 페이지의 『분리 모드 프로세스에서 벤더 라이브러리 함수 실행』 • *관리 루틴 및 뷰*의 『admin_get_dbp_mem_usage - 전체 메모리 소비량 테이블 함수 가져오기』 • 47 페이지의 『에이전트 및 프로세스 모델 구성』 • 50 페이지의 『다중 파티션에서 데이터베이스 구성』
자동 스토리지	• 102 페이지의 『자동 스토리지 데이터베이스』 • 158 페이지의 『자동 스토리지 테이블 스페이스』 • 153 페이지의 『DMS 테이블 스페이스의 자동 크기 조정』 • *데이터 복구 및 고가용성 안내서 및 참조서*의 『자동 데이터베이스 백업』 • *데이터 복구 및 고가용성 안내서 및 참조서*의 『자동 백업 사용』

표 2. 자동 컴퓨팅 정보 개요 (계속)

범주	관련 항목
데이터 압축	• 53 페이지의 『스토리지 스페이스를 절약하기 위한 데이터 압축』 • – 276 페이지의 『테이블 압축』 – 368 페이지의 『인덱스 압축』 • 279 페이지의 『자동 압축 사전 작성』 • *데이터 이동 유틸리티 안내서 및 참조서*의 『로드 조작 중 압축 사전 작성』
자동 데이터베이스 백업	• *데이터 복구 및 고가용성 안내서 및 참조서*의 『자동 데이터베이스 백업』 • *데이터 복구 및 고가용성 안내서 및 참조서*의 『자동 백업 사용』 • *데이터 복구 및 고가용성 안내서 및 참조서*의 『백업 및 복구 전략 개발』
자동 재구성	*문제점 해결 및 데이터베이스 성능 조정*의 『자동 재구성』
자동 통계 콜렉션	• *문제점 해결 및 데이터베이스 성능 조정*의 『자동 통계 콜렉션』 • *문제점 해결 및 데이터베이스 성능 조정*의 『자동 통계 콜렉션 사용』 • *문제점 해결 및 데이터베이스 성능 조정*의 『자동 통계 콜렉션 및 프로파일에서 사용하는 스토리지』 • *문제점 해결 및 데이터베이스 성능 조정*의 『자동 통계 콜렉션 활동 로깅』
구성 어드바이저	• 59 페이지의 『데이터베이스 구성 권장사항 생성』 – 59 페이지의 『구성 어드바이저를 사용하여 구성 매개변수 조정』 – 60 페이지의 『예: 구성 어드바이저를 사용하여 구성 권장사항 요청』 – *명령어 참조서*의 『AUTOCONFIGURE 명령』 – *관리 루틴 및 뷰*의 『ADMIN_CMD 프로시저를 사용하는 AUTOCONFIGURE 명령』 – *관리 API 참조서*의 『db2AutoConfig API - 구성 어드바이저 액세스』 • *문제점 해결 및 데이터베이스 성능 조정*의 『성능 조정을 위한 빠른 시작 추가 정보』
Health Monitor	• *데이터베이스 모니터링 안내서 및 참조서*의 『Health Monitor』 • *데이터베이스 모니터링 안내서 및 참조서*의 『Health 표시기 프로세스 순환』 – *데이터베이스 모니터링 안내서 및 참조서*의 『Health 경보 통지 사용』 – *데이터베이스 모니터링 안내서 및 참조서*의 『클라이언트 응용프로그램을 사용하여 Health 표시기 구성』 • *데이터베이스 모니터링 안내서 및 참조서*의 『Health 표시기 요약』

표 2. 자동 컴퓨팅 정보 개요 (계속)

범주	관련 항목
유틸리티 조절 기능	• 62 페이지의 『유틸리티 조절 기능』 • *문제점 해결 및 데이터베이스 성능 조정*의 『비동기 인덱스 정리』 • *문제점 해결 및 데이터베이스 성능 조정*의 『MDC 테이블에 대한 비동기 인덱스 정리』 – *명령어 참조서*의 『LIST UTILITIES 명령』 – *명령어 참조서*의 『SET UTIL_IMPACT_PRIORITY 명령』 – 780 페이지의 『util_impact_lim - 인스턴스 영향 규정』 – *데이터베이스 모니터링 안내서 및 참조서*의 『utility_priority - 유틸리티 우선순위 모니터 요소』
업그레이드	• *DB2 버전 9.7로 업그레이드*의 『업그레이드된 데이터베이스에서 새로운 DB2 버전 9.7 기능 채택』

자동 기능

자동 기능은 사용자가 데이터베이스 시스템을 관리하는 것을 보조합니다. 자동 기능을 사용하면 실행 기록 문제점 데이터에 대해 실시간 데이터를 분석함으로써 사용자의 시스템에서 자체 진단을 수행하고 문제점이 발생하기 전에 이를 예측할 수 있습니다. 서비스 중단이 발생하지 않도록 개입하지 않아도 시스템을 변경할 수 있도록 일부 자동 도구를 구성할 수 있습니다.

데이터베이스 작성 시 다음 자동 기능 중 일부는 디폴트로 사용 가능하지만 기타 기능은 수동으로 사용 가능하게 설정해야 합니다.

자체 성능 조정 메모리(단일 파티션 데이터베이스 전용)

자체 성능 조정 메모리 기능은 메모리 구성 태스크를 간편하게 해 줍니다. 이 기능은 자동으로 그리고 대화식으로 몇 가지 메모리 구성 매개변수 값과 버퍼 풀 크기를 조정하여 워크로드의 큰 변경에 응답함으로써 성능을 최적화합니다. 메모리 조정 프로그램은 정렬 기능, 패키지 캐시, 잠금 목록 및 버퍼 풀을 포함한 여러 메모리 소비자 간에 사용 가능한 메모리 자원을 동적으로 분배합니다. 데이터베이스 구성 매개변수 **self_tuning_mem**을 OFF로 설정하여 데이터베이스 작성 후 자체 성능 조정 메모리를 사용 불가능하게 설정할 수 있습니다.

자동 스토리지

자동 스토리지 기능은 테이블 스페이스의 스토리지 관리를 간편하게 합니다. 데이터베이스 작성 시 데이터베이스 관리 프로그램이 사용자의 테이블 스페이스 데이터를 저장할 스토리지 경로가 지정됩니다. 그런 다음 데이터베이스 관리 프로그램이 컨테이너 및 사용자가 테이블 스페이스를 작성하여 채울 경우 테이블 스페이스의 스페이스 할당을 관리합니다.

데이터 압축

테이블 및 인덱스를 둘 다 압축하여 스토리지를 절약할 수 있습니다. 압축은 완전히 자동입니다. CREATE TABLE, ALTER TABLE, CREATE INDEX 또는 ALTER INDEX문의 COMPRESS YES절을 사용하여 테이블이나 인덱스가 압축되도록 지정한 후에는 압축을 관리하기 위해 사용자가 수행할 작업은 없습니다. 기존의 압축되지 않은 테이블이나 인덱스를 압축되도록 변환하려면 기존 데이터를 압축하기 위한 REORG가 필요합니다. 임시 테이블은 자동으로 압축되므로, 압축된 테이블에 대한 인덱스도 디폴트로 자동으로 압축됩니다.

자동 데이터베이스 백업

다양한 하드웨어 또는 소프트웨어 고장으로 인해 데이터베이스가 사용 불가능해질 수 있습니다. 데이터베이스의 최신, 전체 백업을 보유하도록 하는 것은 시스템의 재해 복구 전략을 계획하고 구현하는 데 필수적인 부분입니다. 자동 데이터베이스 백업을 재해 복구 전략의 파트로 사용하여 데이터베이스 관리 프로그램이 데이터베이스를 올바로, 그리고 정기적으로 백업하게 하십시오.

자동 재구성

테이블 데이터에 대한 많은 변경 후, 테이블 및 해당 인덱스가 분할될 수 있습니다. 논리적 순차 데이터가 비연속 페이지에 상주할 수 있으므로, 데이터베이스 관리 프로그램이 추가 읽기 조작을 수행하여 데이터에 액세스해야 합니다. 자동 재구성 프로세스는 통계를 갱신한 테이블 및 인덱스를 주기적으로 평가하여 재구성이 필요한지 확인하고, 필요할 때마다 그러한 조작을 스케줄합니다.

자동 통계 콜렉션

자동 통계 콜렉션은 사용자가 최신 테이블 통계를 갖도록 하여 데이터베이스 성능을 개선하는 데 유용합니다. 데이터베이스 관리 프로그램이 사용자의 워크로드에 필요한 통계 및 갱신해야 하는 통계를 판별합니다. SQL문을 컴파일할 때 런타임 통계를 수집하여 통계를 비동기적으로(백그라운드에서) 또는 동기적으로 수집할 수 있습니다. 그러면 DB2 옵티마이저가 정확한 통계를 바탕으로 액세스 플랜을 선택할 수 있습니다. 데이터베이스 구성 매개변수 **auto_runstats**를 OFF로 설정하여 데이터베이스 작성 후 자동 통계 콜렉션을 사용 불가능하게 설정할 수 있습니다. 자동 통계 콜렉션이 사용 가능한 경우에만 실시간 통계 수집을 사용할 수 있습니다. 실시간 통계 수집은 **auto_stmt_stats** 구성 매개변수를 통해 제어됩니다.

구성 어드바이저

데이터베이스 작성 시 이 도구가 자동으로 실행되어 데이터베이스 구성 매개변수와 디폴트 버퍼 풀(IBMDEFAULTBP) 크기를 판별하고 설정합니다. 시스템 자원 및 시스템 사용 용도에 따라 값이 선택됩니다. 이러한 초기 자동 성능 조정은 사용자의 데이터베이스가 디폴트값을 사용하여 작성할 수 있는 상응 데이터베이스보다 성능이 우수함을 의미합니다. 또한 데이터베이스 작성 후 시스템 조정에 더 적은 시간을 사용함을 의미하기도 합니다. 언제든지(데이터베이스가

채워진 이후에도) 구성 어드바이저를 실행하여 도구 권장사항을 가져올 수 있으며 선택적으로 구성 매개변수 세트를 적용하여 현재 시스템 특성에 따라 성능을 최적화할 수도 있습니다.

Health Monitor

Health Monitor는 성능 저하 또는 잠재적인 전원 꺼짐을 일으킬 수 있는 데이터베이스 환경의 변경사항 또는 상황을 혁신적으로 모니터하는 서버 측 도구입니다. 사용자 파트에서 어떤 양식의 활성 모니터링 없이도 여러 성능 상태 정보가 제공됩니다. 성능 상태 위험성에 직면하면 데이터베이스 관리 프로그램이 사용자에게 알리고 대처 방법에 대한 조언을 제공합니다. Health Monitor는 스냅샷 모니터를 사용하여 시스템에 대한 정보를 수집하며 성능 저하를 발생시키지 않습니다. 또한 Health Monitor는 정보를 수집하기 위해 스냅샷 모니터 스위치를 켜지 않습니다.

유틸리티 조절 기능

이 기능은 프로덕션 기간에 유지보수 유틸리티를 동시에 실행할 수 있도록 유지보수 유틸리티의 성능 영향을 규제합니다. 조절 유틸리티의 *영향 규정*이 디폴트로 정의되어 있지만 조정 유틸리티를 실행하려면 *영향 우선순위*를 설정해야 합니다. 조절 시스템은 영향 규정을 위반하지 않고 조절 유틸리티를 가능한 한 자주 실행할 수 있도록 합니다. 현재 통계 콜렉션, 백업 작업, 재조정 작업 및 비동기 인덱스 정리를 조정할 수 있습니다.

자동 유지보수

데이터베이스 관리 프로그램에서는 필요에 따라 데이터베이스 백업을 수행하고, 통계 경향을 보존하며 테이블 및 인덱스를 재구성하는 데 필요한 자동 유지보수 기능을 제공합니다. 데이터베이스에서 유지보수 활동을 수행하는 것은 데이터베이스 성능 및 복구 능력을 최적화하는 데 필수적입니다.

데이터베이스 유지보수에는 다음 활동 중 일부 또는 모두가 포함됩니다.

- **백업.** 데이터베이스를 백업하면 데이터베이스 관리 프로그램이 데이터베이스의 데이터를 복사하여 원래 데이터에 장애가 발생하거나 손상될 경우에 대비하여 다른 매체에 저장합니다. 자동 데이터베이스 백업은 사용자가 백업 시기 또는 BACKUP 명령 구문에 대해 걱정할 필요가 없도록 데이터베이스가 올바르게 정기적으로 백업되는지 확인하는 데 유용합니다.
- **데이터 조각 모음(테이블 또는 인덱스 재구성).** 이 유지보수 활동은 사용자의 테이블에 액세스하는 데이터베이스 관리 프로그램의 효율을 증가시킵니다. 자동 재구성은 사용자가 데이터 재구성 시기 및 방법에 대해 걱정할 필요가 없도록 오프라인 테이블 및 인덱스 재구성을 관리합니다.
- **데이터 액세스 최적화(통계 콜렉션).** 데이터베이스 관리 프로그램은 테이블의 데이터, 인덱스의 데이터 또는 테이블과 인덱스 둘 다에 있는 데이터에 대한 시스템 카탈로

그 통계를 갱신합니다. 옵티마이저에서 이러한 통계를 사용하여 데이터에 액세스하기 위해 사용할 경로를 판별합니다. 자동 통계 콜렉션은 최신 테이블 통계를 유지보수하여 데이터베이스의 성능을 개선하려 합니다. 정확한 통계를 바탕으로 옵티마이저가 액세스 플랜을 선택할 수 있도록 하는 것이 목표입니다.

- **통계 프로파일링.** 자동 통계 프로파일링은 오래되었거나, 누락되었거나 올바르지 않은 통계를 발견함으로써, 그리고 쿼리 피드백을 바탕으로 통계 프로파일을 생성함으로써 테이블 통계를 수집할 시기 및 방법을 권고합니다.

유지보수 활동 실행 여부 및 시기를 판별하는 일에는 시간이 많이 소요될 수 있으나 자동 유지보수를 사용하면 사용자에게서 이러한 부담이 제거됩니다. 자동 유지보수 데이터베이스 구성 매개변수를 사용하여 간단하고 융통성 있게 자동 유지보수 기능의 인에이블먼트를 관리할 수 있습니다. 자동 유지보수 구성 마법사를 사용하여 유지보수 목표를 지정할 수 있습니다. 데이터베이스 관리 프로그램에서는 이러한 목표를 사용하여 유지보수 활동을 수행해야 하는지 여부를 판별하고 가능한 다음 유지보수 기간(사용자가 정의한 기간)에 필수 유지보수 활동만 실행합니다.

유지보수 기간

유지보수 기간은 백업, 통계 수집, 통계 프로파일링 및 재구성과 같은 자동 유지보수 활동을 실행하기 위해 정의하는 기간입니다. 오프라인 기간은 데이터베이스에 대한 액세스가 불가능한 기간입니다. 온라인 기간은 사용자가 데이터베이스에 연결하도록 허용되는 기간입니다.

유지보수 기간은 태스크 스케줄과 다릅니다. 유지보수 기간에 각 자동 유지보수 활동을 실행할 필요는 없습니다. 대신, 데이터베이스 관리 프로그램이 시스템을 평가하여 각 유지보수 활동을 실행해야 하는지 여부를 판별합니다. 유지보수 요구사항이 충족되지 않은 경우 유지보수 활동이 실행됩니다. 데이터베이스가 이미 잘 유지보수된 경우에는 유지보수 활동이 실행되지 않습니다.

자동 유지보수 활동을 실행할 시기에 대해 검토하십시오. 자동 유지보수 활동은 시스템의 자원을 소비하며 활동 실행 시 데이터베이스 성능에 영향을 줄 수 있습니다. 일부 자동 유지보수 활동은 테이블, 인덱스 및 데이터베이스에 대한 액세스를 제한할 수도 있습니다. 따라서, 데이터베이스 관리 프로그램이 유지보수 활동을 실행할 수 있는 적절한 기간을 제공해야 합니다. 제어 센터 또는 Health Center에서 자동 유지보수 마법사를 사용하여 자동 유지보수 기간을 오프라인 및 온라인 유지보수 기간으로 지정할 수 있습니다.

오프라인 유지보수 활동

오프라인 유지보수 활동(오프라인 데이터베이스 백업 및 테이블과 인덱스 재구성)은 오프라인 유지보수 기간에만 발생할 수 있는 유지보수 활동입니다. 사용자 액세스가 영향을 받는 정도는 실행 중인 유지보수 활동 종류에 따라 다릅니다.

- 오프라인 백업 중에는 응용프로그램을 데이터베이스에 연결할 수 없습니다. 현대 연결되어 있는 모든 응용프로그램은 강제로 연결 해제됩니다.
- 오프라인 테이블 또는 인덱스 재구성(데이터 조각 모음) 중에 응용프로그램이 액세스할 수 있지만 테이블의 데이터를 갱신할 수는 없습니다.

오프라인 유지보수 활동은 지정된 기간이 지나도 완료될 때까지 실행됩니다. 시간이 경과함에 따라 내부 스케줄링 메커니즘이 작업 완료 시간을 가장 근접하게 추정하는 방법을 학습합니다. 오프라인 유지보수 기간이 특정 데이터베이스 백업 또는 재구성 활동을 수행하기에 너무 짧은 경우 스케줄러가 다음 번에 작업을 시작하지 않으며 Health Monitor를 통해 오프라인 유지보수 기간을 늘려야 함을 통지합니다.

온라인 유지보수 활동

온라인 유지보수 활동(자동 통계 콜렉션 및 프로파일링, 온라인 인덱스 재구성, 온라인 데이터베이스 백업)은 온라인 유지보수 기간에만 발생하는 유지보수 활동입니다. 온라인 유지보수 활동이 실행되면 현재 연결되어 있는 응용프로그램은 연결 상태로 남아 있도록 허용되고 새 연결을 설정할 수 있습니다. 시스템에 미칠 영향을 최소화하기 위해 적응 유틸리티 조절 기능 메커니즘을 통해 온라인 데이터베이스 백업과 자동 통계 콜렉션 및 프로파일링을 조절합니다.

온라인 유지보수 활동은 지정된 기간이 지나도 완료될 때까지 실행됩니다.

자체 성능 조정 메모리

DB2 버전 9에서부터 메모리 조정 기능은 여러 메모리 구성 매개변수에 대한 값을 자동으로 설정하여 메모리 구성 태스크를 단순화해 줍니다. 이 기능이 사용될 경우, 메모리 조정 프로그램이 버퍼 풀, 잠금 메모리, 패키지 캐시 및 정렬 메모리 간에 사용 가능한 메모리 자원을 동적으로 분배합니다.

조정 프로그램은 **database_memory** 구성 매개변수에 의해 정의된 메모리 제한 내에서 작동합니다. 이 매개변수의 값 역시 자동으로 조정될 수 있습니다. 자체 성능 조정이 사용될 경우(**database_memory**의 값이 AUTOMATIC으로 설정된 경우), 조정 프로그램에서 데이터베이스에 대한 전체 메모리 요구사항을 판별하고 현재 데이터베이스 요구사항에 따라 데이터베이스 공유 메모리에 대해 할당된 메모리 양을 늘리거나 줄입니다. 예를 들어, 현재 데이터베이스 요구사항이 높고 시스템에 여유 메모리가 충분한 경우, 데이터베이스 공유 메모리에 대해 더 많은 메모리가 할당됩니다. 데이터베이스 메모리 요구사항이 줄어들 경우, 또는 시스템의 여유 메모리가 너무 낮아질 경우, 일부 데이터베이스 공유 메모리가 릴리스됩니다.

database_memory 구성 매개변수가 AUTOMATIC으로 설정되지 않은 경우, 데이터베이스에서 이 매개변수에 대해 지정된 메모리 양을 사용하며 필요에 따라 메모리 소비자 간에 분배합니다. **database_memory**를 일부 숫자 값으로 설정하거나 COMPUTED

로 설정하는 두 가지 방법 중 하나로 메모리의 양을 지정할 수 있습니다. 후자의 경우, 총 메모리 양은 데이터베이스 시작 시간에 데이터베이스 메모리 힙의 초기 값 합계를 기반으로 합니다.

자체 성능 조정에 대해 메모리 소비자를 다음과 같이 사용할 수도 있습니다.

- 버퍼 풀의 경우, ALTER BUFFERPOOL 또는 CREATE BUFFERPOOL문 사용(AUTOMATIC 키워드 지정)
- 잠금 메모리의 경우, **locklist** 또는 **maxlocks** 데이터베이스 구성 매개변수 사용(AUTOMATIC 값 지정)
- 패키지 캐시의 경우, **pckcachesz** 데이터베이스 구성 매개변수 사용(AUTOMATIC 값 지정)
- 정렬 메모리의 경우, **sheapthres_shr** 또는 **sortheap** 데이터베이스 구성 매개변수 사용(AUTOMATIC 값 지정)

자체 성능 조정 조작으로 발생한 변경사항은 `stmmlog` 서브디렉토리에 있는 메모리 조정 로그 파일에 기록됩니다. 이러한 로그 파일에는 로그 항목의 시간소인으로 판별되는 특정 시간소인 조정 간격 동안 각 메모리 소비자에서의 자원 요구에 대한 요약이 포함됩니다.

사용 가능한 메모리가 적은 경우, 자체 성능 조정의 성능 이점이 제한됩니다. 성능 조정 결정은 데이터베이스 워크로드를 기반으로 하기 때문에 메모리 요구사항이 빠르게 변하는 워크로드에서는 자체 성능 조정 메모리 관리자(STMM)의 효율성을 제한합니다. 워크로드의 메모리 특성이 끊임없이 변경되는 경우, STMM은 변화하는 목표 조건 하에서 더 적은 빈도로 조정을 수행합니다. 이 시나리오에서, STMM은 절대적인 메모리 구성을 얻지 못하지만 대신 현재 워크로드에 맞게 조정된 메모리 구성을 유지하려고 노력합니다.

자체 성능 조정 메모리

DB2 버전 9에서부터 메모리 조정 기능은 여러 메모리 구성 매개변수에 대한 값을 자동으로 설정하여 메모리 구성 태스크를 단순화해 줍니다. 이 기능이 사용될 경우, 메모리 조정 프로그램이 버퍼 풀, 잠금 메모리, 패키지 캐시 및 정렬 메모리 간에 사용 가능한 메모리 자원을 동적으로 분배합니다.

조정 프로그램은 **database_memory** 구성 매개변수에 의해 정의된 메모리 제한 내에서 작동합니다. 이 매개변수의 값 역시 자동으로 조정될 수 있습니다. 자체 성능 조정이 사용될 경우(**database_memory**의 값이 AUTOMATIC으로 설정된 경우), 조정 프로그램에서 데이터베이스에 대한 전체 메모리 요구사항을 판별하고 현재 데이터베이스 요구사항에 따라 데이터베이스 공유 메모리에 대해 할당된 메모리 양을 늘리거나 줄입니다. 예를 들어, 현재 데이터베이스 요구사항이 높고 시스템에 여유 메모리가 충분한

경우, 데이터베이스 공유 메모리에 대해 더 많은 메모리가 할당됩니다. 데이터베이스 메모리 요구사항이 줄어들 경우, 또는 시스템의 여유 메모리가 너무 낮아질 경우, 일부 데이터베이스 공유 메모리가 릴리스됩니다.

database_memory 구성 매개변수가 AUTOMATIC으로 설정되지 않은 경우, 데이터베이스에서 이 매개변수에 대해 지정된 메모리 양을 사용하며 필요에 따라 메모리 소비자 간에 분배합니다. **database_memory**를 일부 숫자 값으로 설정하거나 COMPUTED로 설정하는 두 가지 방법 중 하나로 메모리의 양을 지정할 수 있습니다. 후자의 경우, 총 메모리 양은 데이터베이스 시작 시간에 데이터베이스 메모리 힙의 초기 값 합계를 기반으로 합니다.

자체 성능 조정에 대해 메모리 소비자를 다음과 같이 사용할 수도 있습니다.

- 버퍼 풀의 경우, ALTER BUFFERPOOL 또는 CREATE BUFFERPOOL문 사용(AUTOMATIC 키워드 지정)
- 잠금 메모리의 경우, **locklist** 또는 **maxlocks** 데이터베이스 구성 매개변수 사용(AUTOMATIC 값 지정)
- 패키지 캐시의 경우, **pckcachesz** 데이터베이스 구성 매개변수 사용(AUTOMATIC 값 지정)
- 정렬 메모리의 경우, **sheapthres_shr** 또는 **sortheap** 데이터베이스 구성 매개변수 사용(AUTOMATIC 값 지정)

자체 성능 조정 조작으로 발생한 변경사항은 `stmmlog` 서브디렉토리에 있는 메모리 조정 로그 파일에 기록됩니다. 이러한 로그 파일에는 로그 항목의 시간소인으로 판별되는 특정 시간소인 조정 간격 동안 각 메모리 소비자에서의 자원 요구에 대한 요약이 포함됩니다.

사용 가능한 메모리가 적은 경우, 자체 성능 조정의 성능 이점이 제한됩니다. 성능 조정 결정은 데이터베이스 워크로드를 기반으로 하기 때문에 메모리 요구사항이 빠르게 변하는 워크로드에서는 자체 성능 조정 메모리 관리자(STMM)의 효율성을 제한합니다. 워크로드의 메모리 특성이 끊임없이 변경되는 경우, STMM은 변화하는 목표 조건 하에서 더 적은 빈도로 조정을 수행합니다. 이 시나리오에서, STMM은 절대적인 메모리 구성을 얻지 못하지만 대신 현재 워크로드에 맞게 조정된 메모리 구성을 유지하려고 노력합니다.

자체 성능 조정 메모리 개요

자체 성능 조정 메모리는 자동으로 메모리 구성 매개변수의 값을 설정하고 버퍼 풀의 크기를 지정함으로써 메모리 구성 태스크를 간편하게 해 줍니다. 이 기능이 사용될 경우, 메모리 조정 프로그램이 버퍼 풀, 잠금 메모리, 패키지 캐시 및 정렬 메모리 간에 사용 가능한 메모리 자원을 동적으로 분배합니다.

자체 성능 조정 메모리는 **self_tuning_mem** 데이터베이스 구성 매개변수를 통해 사용됩니다.

다음 메모리 관련 데이터베이스 구성 매개변수가 자동으로 조정될 수 있습니다.

- database_memory - 데이터베이스 공유 메모리 *크기*
- locklist - 잠금 목록의 최대 스토리지
- maxlocks - 에스컬레이션 전 잠금 목록의 최대 퍼센트
- pckcachesz - 패키지 캐시 *크기*
- sheapthres_shr - 공유 정렬에 대한 정렬 힙 임계값
- sortheap - 정렬 힙 *크기*

메모리 할당

메모리 할당 및 할당 해제는 다양한 시점에 발생합니다. 특정 이벤트가 발생하면 (예: 응용프로그램 연결 시) 특정 메모리 영역으로 메모리를 할당하거나 구성 변경에 대한 응답으로 재할당할 수 있습니다.

33 페이지의 그림 1은 다양한 용도로 데이터베이스 관리 프로그램이 할당하는 여러 가지 다른 메모리 영역과 이러한 메모리 영역의 *크기*를 제어하는 데 사용할 수 있는 구성 매개변수를 나타냅니다. 파티션된 데이터베이스 환경에서 각 데이터베이스 파티션에는 자체의 데이터베이스 관리 프로그램 공유 메모리 세트가 있다는 점을 참고하십시오.

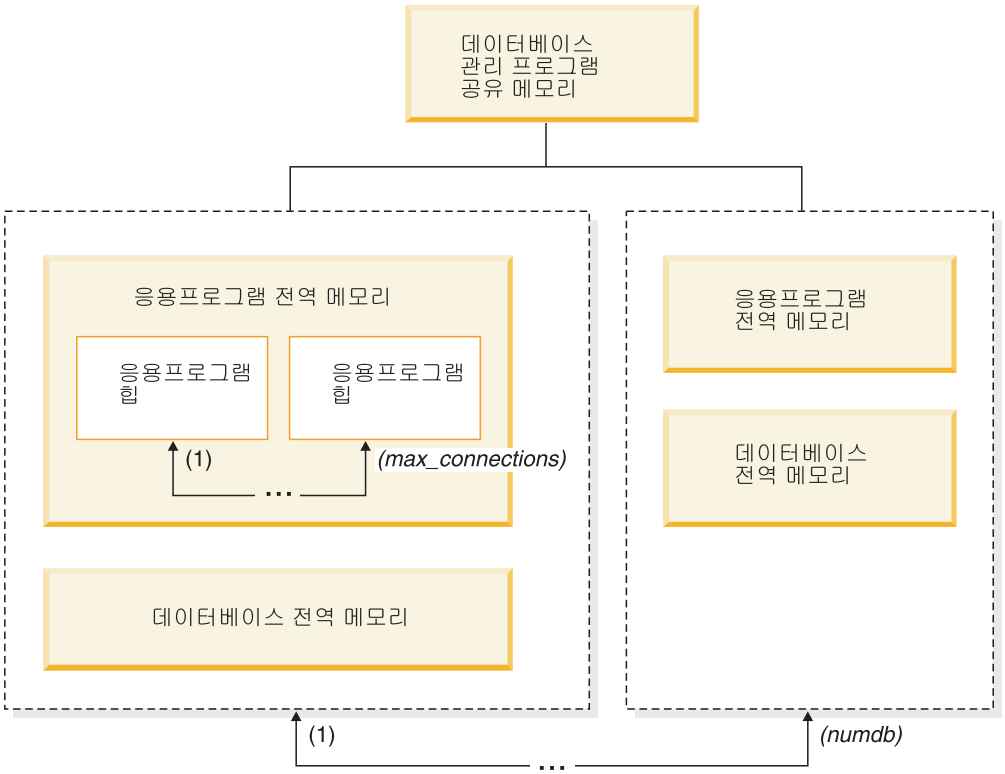


그림 1. 데이터베이스 관리 프로그램이 할당하는 메모리의 유형

다음 중 한 이벤트가 발생할 때마다 데이터베이스 관리 프로그램이 메모리를 할당합니다.

데이터베이스 관리 프로그램 시작 시(db2start)

데이터베이스 관리 프로그램 공유 메모리(*인스턴스 공유 메모리*라고도 함)는 데이터베이스 관리 프로그램이 중지되기 전까지(db2stop) 계속 할당되어 있습니다. 이 영역에는 데이터베이스 관리 프로그램이 모든 데이터베이스 연결을 통해 활동을 관리하는 데 사용하는 정보가 들어 있습니다. DB2는 데이터베이스 관리 프로그램 공유 메모리의 *크기*를 자동으로 제어합니다.

데이터베이스가 처음으로 활성화 또는 연결된 경우

데이터베이스에 연결되는 모든 응용프로그램에서 *데이터베이스 전역 메모리*를 사용합니다. 데이터베이스 전역 메모리의 *크기*는 **database_memory** 데이터베이스 구성 매개변수에 지정되어 있습니다. 디폴트로 이 매개변수는 자동으로 설정되어 있으므로 DB2는 데이터에 할당된 초기 메모리 양을 계산하고 데이터베이스의 필요에 따라 런타임 중 데이터베이스 메모리 *크기*를 자동으로 구성할 수 있습니다.

다음 메모리 영역을 동적으로 조정할 수 있습니다.

- 버퍼 풀(ALTER BUFFERPOOL문 사용)
- 데이터베이스 힙(로그 버퍼 포함)

- 유틸리티 힙
- 패키지 캐시
- 카탈로그 캐시
- 잠금 목록. 이 메모리 영역만 동적으로 증가될 수 있습니다(감소될 수 없음).

sortheap, **sheapthres_shr** 및 **sheapthres** 구성 매개변수도 동적으로 갱신할 수 있습니다. 유일한 제한사항은 **sheapthres**를 0에서 0보다 큰 값으로 동적으로 변경할 수 없거나 그 반대의 경우도 마찬가지라는 점입니다.

공유 정렬 조작이 디폴트로 수행되며 정렬 메모리 사용자가 한 번에 사용할 수 있는 데이터베이스 공유 메모리의 양은 **sheapthres_shr** 데이터베이스 구성 매개변수의 값으로 판별됩니다. 개인용 정렬 조작은 파티션 내 병렬 처리, 데이터베이스 파티션 및 연결 집중기(connection concentrator)가 모두 사용 불가능하며 **sheapthres** 데이터베이스 관리 프로그램 구성 매개변수가 0이 아닌 값으로 설정된 경우에만 수행됩니다.

응용프로그램이 데이터베이스에 연결된 경우

각 응용프로그램에는 *응용프로그램 전역 메모리*의 일부분인 자체의 *응용프로그램 힙*이 있습니다. **applheapsz** 데이터베이스 구성 매개변수를 사용하여 한 응용프로그램이 할당할 수 있는 메모리의 양을 제한하거나 **appl_memory** 데이터베이스 구성 매개변수를 사용하여 전체 응용프로그램 메모리 사용을 제한할 수 있습니다.

에이전트 작성 시

*에이전트 전용 메모리*는 해당 에이전트가 파티션된 데이터베이스 환경에서 연결 요청 또는 새 SQL 요청의 결과로 지정된 경우 에이전트에 할당됩니다. 에이전트 전용 메모리에는 이 특정 에이전트에서만 사용하는 메모리가 포함됩니다. 개인용 정렬 조작이 사용 가능한 경우 에이전트 전용 메모리에서 개인용 정렬 힙이 할당됩니다.

다음의 구성 매개변수는 각 메모리 영역 유형에 할당되는 메모리의 양을 제한합니다. 파티션된 데이터베이스 환경에서 이 메모리는 각 데이터베이스 파티션에 할당된다는 점을 참고하십시오.

numdb

이 데이터베이스 관리 프로그램 구성 매개변수는 다른 응용프로그램이 사용할 수 있는 최대 동시 활성 데이터베이스 수를 지정합니다. 각 데이터베이스에는 자체의 전역 메모리 영역이 있으므로 이 매개변수의 값을 높이면 할당할 수 있는 메모리의 양이 증가합니다.

maxappls

이 데이터베이스 구성 매개변수는 특정 데이터베이스에 동시에 연결할 수 있는

최대 응용프로그램 수를 지정합니다. 이 매개변수의 값은 해당 데이터베이스의 에이전트 전용 메모리 및 응용프로그램 전역 메모리로 할당할 수 있는 메모리의 양에 영향을 줍니다.

max_connections

이 데이터베이스 관리 프로그램 구성 매개변수는 한 번에 데이터 서버에 액세스할 수 있는 데이터베이스 연결 또는 인스턴스 접속 수를 제한합니다.

max_coordagents

이 데이터베이스 관리 프로그램 구성 매개변수는 한 인스턴스의 모든 활성 데이터베이스에서(그리고 파티션된 데이터베이스 환경에서 데이터베이스 파티션마다) 동시에 존재할 수 있는 데이터베이스 관리 프로그램 코디네이팅 에이전트 수를 제한합니다. **maxappls** 및 **max_connections**와 함께 이 매개변수는 에이전트 전용 메모리 및 응용프로그램 전역 메모리로 할당되는 메모리의 양을 제한합니다.

db2mtrk 명령을 통해 호출하는 메모리 추적 프로그램은 인스턴스에서 메모리의 현재 할당을 표시합니다. 또한 ADMIN_GET_DBP_MEM_USAGE 테이블 함수를 사용하면 전체 인스턴스 또는 단일 데이터베이스 파티션만의 총 메모리 사용량을 판별할 수 있습니다. GET SNAPSHOT 명령을 사용하면 인스턴스, 데이터베이스 또는 응용프로그램 레벨에서 현재 메모리 사용량을 조사할 수 있습니다.

메모리 매개변수 상호 작용 및 한계

자체 성능 조정 메모리를 사용하고 대부분의 메모리 관련 구성 매개변수에 대해 디폴트 AUTOMATIC 설정을 사용할 수 있지만, 설정값을 보다 제어하고 특정 상황에서 메모리 부족 오류가 여전히 발생 가능한 이유를 이해하기 위해 여러 가지 메모리 매개변수의 한계 및 매개변수 간의 상호 작용을 아는 것이 유용합니다.

메모리 유형

기본적으로 DB2 데이터베이스 관리 프로그램은 다음 두 유형의 메모리를 사용합니다.

캐시 기반 메모리

이 메모리는 자체 성능 조정 메모리 관리자(STMM)가 제어하며 다양한 성능 힙에 분배됩니다. **database_memory** 구성 매개변수를 최대 캐시 기반 메모리 양으로 설정하거나 **database_memory**를 AUTOMATIC으로 설정하여 STMM이 전체 캐시 기반 메모리를 관리하도록 할 수 있습니다.

기능 메모리

이 메모리는 응용프로그램에서 사용됩니다. **appl_memory** 구성 매개변수를 사용하여 데이터베이스 요청을 서비스하기 위해 DB2 데이터베이스 에이전트가 할당하는 기능 메모리 또는 응용프로그램 메모리의 최대량을 제어할 수 있습니다. 디폴트로 이 매개변수는 AUTOMATIC으로 설정되는데, 데이터베이스 파티션이 할당하는 메모리의 총량이 **instance_memory** 한계 미만인 경우 기능

메모리 요청이 허용됨을 의미합니다. **instance_memory** 구성 매개변수는 데이터베이스 파티션에 대해 할당할 수 있는 최대 메모리량을 지정합니다.

AUTOMATIC 설정을 사용할 수 있기 전에는 공유 메모리, 전용 메모리, 버퍼 풀 메모리, 잠금 목록, 정렬 메모리(힙) 등과 같은 여러 가지 유형의 메모리가 사용하는 스페이스의 양을 볼 수 있는 다양한 운영 체제 및 DB2 도구를 사용할 수 있었지만 DB2 데이터베이스 관리 프로그램이 사용하는 메모리의 총량을 아는 것은 거의 불가능했습니다. 힙 중 하나가 메모리 한계에 도달하면 응용프로그램의 명령문이 메모리 부족 오류 메시지와 함께 실패했습니다. 해당 힙에 대한 메모리를 늘리고 응용프로그램을 재실행하면 다른 힙에 대한 기타 명령문에서 메모리 부족 오류를 수신할 수 있습니다. 이제는 디폴트 AUTOMATIC 구성 매개변수 설정을 사용하여 개별 기능 메모리 힙에 대한 하드 상한을 제거할 수 있습니다.

필요한 경우(예를 들어 불충분하게 동작하는 데이터베이스 응용프로그램에서 극히 많은 양의 메모리가 필요한 경우) **appl_memory** 구성 매개변수를 사용하여 데이터베이스 레벨에서 전체 응용프로그램 메모리에 대한 한계를 적용할 수 있습니다. 또한 해당 힙에 대한 적절한 데이터베이스 구성 매개변수를 AUTOMATIC 설정에서 고정값으로 변경하여 개별 힙에 대한 한계를 적용할 수도 있습니다. 모든 기능 메모리 힙에 대한 모든 구성 매개변수 및 **appl_memory** 구성 매개변수가 디폴트 AUTOMATIC 설정을 사용하는 경우, 응용프로그램 메모리 소비에 대한 유일한 한계는 **instance_memory** 설정입니다. 또한 **instance_memory**를 AUTOMATIC으로 설정하면, DB2 데이터베이스 관리 프로그램이 자동으로 메모리 소비에 대한 상한을 결정합니다. admin_get_dbp_mem_usage 테이블 함수를 사용하여 쉽게 소비된 인스턴스 메모리의 총량과 현재 **instance_memory** 한계를 알 수 있습니다.

메모리 구성 매개변수 간의 상호 작용

자체 성능 조정 메모리 전체를 사용 가능한 경우(**self_tuning_mem**이 ON으로 설정되고, 모든 메모리 매개변수가 AUTOMATIC으로 설정됨), STMM이 시스템에서 사용 가능한 여유 메모리를 점검하고 최적 성능을 위해 캐시 기반 힙의 전용 메모리량을 자동으로 결정합니다. 모든 캐시 기반 힙이 전체 **database_memory** 크기에 영향을 미칩니다. 캐시 기반 메모리 요구사항 외에, DB2 데이터베이스 관리 프로그램의 조작 및 무결성을 보장하기 위해 일부 메모리가 필요합니다. **instance_memory**가 사용하는 스페이스와 이러한 두 메모리 소비자에서 필요한 스페이스 간의 차이를 응용프로그램 메모리(**appl_memory**)에 사용할 수 있습니다. 그런 다음 응용프로그램에 대한 기능 메모리가 **instance_memory** 한계 미만인 경우 필요한 대로 할당됩니다. 단일 응용프로그램이 할당할 수 있는 메모리의 양에 대한 추가 제한은 없습니다.

STMM은 또한 남아 있는 여유 시스템 메모리의 양과 남아 있는 여유 **instance_memory** 스페이스의 양을 주기적으로 쿼리합니다. 응용프로그램 오류를 막기 위해, STMM은 성능 기준보다 응용프로그램 요구사항을 우선시합니다. 필요한 경우 캐시 기반 힙에 사용 가능한 스페이스의 양을 줄여서 성능을 저하시킨 후 충분한 여

유 시스템 메모리와 **instance_memory** 스페이스를 제공하여 응용프로그램 메모리 요청을 충족시킵니다. 응용프로그램이 완료되면 사용된 메모리가 해제되고 다른 응용프로그램이 재사용할 준비가 되거나 STMM의 **database_memory** 사용을 위해 재개됩니다. 데이터베이스 시스템 성능이 응용프로그램 활동이 많은 기간 동안 허용할 수 없는 정도가 되는 경우, 데이터베이스 관리 프로그램이 실행할 수 있는 응용프로그램 수를 제어하거나(예를 들어 연결 집중기(connection concentrator) 또는 DB2 버전 9.5 의 새로운 워크로드 관리 기능을 사용하여) 시스템에 메모리 자원을 추가하는 것이 유용합니다.

한계

STMM이 메모리 사용이 갑자기 늘어난 상황에 대비할 시간이 부족한 경우 여전히 메모리 부족 오류가 발생할 수 있습니다. 이 오류는 예를 들어 응용프로그램에서 갑자기 매우 많은 양의 메모리가 필요한 경우 또는 데이터베이스 워크로드가 급격히 증가한 경우(예를 들어 많은 새 응용프로그램이 동시에 데이터베이스에 연결할 때) 발생할 수 있습니다. 이 경우 또는 대부분의 응용프로그램이 설정된 양의 메모리를 사용함을 알고 있는 경우, **appl_memory**에 대해 AUTOMATIC 설정 대신 하드 코딩된 값을 사용하는 것이 좋습니다. **appl_memory**를 하드 한계(예: 2GB)로 설정하면, DB2 데이터베이스 관리 프로그램은 전체 응용프로그램 메모리 소비가 이 양을 초과하는 것을 방지합니다. 각 응용프로그램은 전체 응용프로그램 메모리 소비가 **appl_memory** 한계보다 작은 경우에 한해 필요한 만큼 많은 메모리를 소비할 수 있습니다. **appl_memory** 한계 또는 **instance_memory** 한계에 도달되면, 데이터베이스 관리 프로그램이 한계에 접근하는 원인이 되는 응용프로그램 요청이 실패하고 적합한 SQL 코드를 리턴합니다(리턴되는 오류 코드는 응용프로그램 조작에서 메모리 부족 실패가 발생한 정확한 위치에 따라 다릅니다). 메모리 부족 오류가 발생하면 db2diag 로그 파일을 보고 오류가 발생했을 때 사용 중이던 메모리의 양을 판별할 수 있으며, 이것은 임의의 메모리 매개변수를 조정해야 하는지 여부를 판별하는 데 도움이 됩니다.

자체 성능 조정 메모리 사용

자체 성능 조정 메모리는 자동으로 메모리 구성 매개변수의 값을 설정하고 버퍼 풀의 크기를 지정함으로써 메모리 구성 태스크를 간편하게 해 줍니다.

이 태스크에 대한 정보

이 기능이 사용될 경우, 메모리 조정 프로그램이 버퍼 풀, 잠금 메모리, 패키지 캐시 및 정렬 메모리를 포함한 여러 메모리 소비자 간에 사용 가능한 메모리 자원을 동적으로 분배합니다.

프로시저

1. UPDATE DATABASE CONFIGURATION 명령 또는 db2CfgSet API를 사용해 **self_tuning_mem** 데이터베이스 구성 매개변수를 ON으로 설정하여 데이터베이스에 대해 자체 성능 조정 메모리를 사용할 수 있게 하십시오.
2. 메모리 구성 매개변수에 의해 제어되는 메모리 영역 자체 성능 조정을 사용할 수 있게 하려면 UPDATE DATABASE CONFIGURATION 명령 또는 db2CfgSet API를 사용하여 관련 구성 매개변수를 AUTOMATIC으로 설정하십시오.
3. 버퍼 풀 자체 성능 조정을 사용할 수 있게 하려면 CREATE BUFFERPOOL문 또는 ALTER BUFFERPOOL문을 사용하여 버퍼 풀 *크기*를 AUTOMATIC으로 설정하십시오. 파티션된 데이터베이스 환경에서는 해당 버퍼 풀의 SYSCAT.BUFFERPOOLDBPARTITIONS에 항목이 없어야 합니다.

결과

주:

1. 자체 성능 조정된 메모리는 다양한 메모리 소비자 간에 분배되므로, 주어진 시간에 자체 성능 조정에 대해 최소한 두 개의 메모리 영역이 동시에 사용 가능해야 합니다(예: 잠금 메모리 및 데이터베이스 공유 메모리). 메모리 조정 프로그램은 다음 조건 중 하나가 참이면 시스템의 메모리를 적극적으로 조정합니다(**self_tuning_mem** 데이터베이스 구성 매개변수 값이 ON임).
   - 하나의 구성 매개변수 또는 버퍼 풀 *크기*가 AUTOMATIC으로 설정되어 있고 **database_memory** 데이터베이스 구성 매개변수가 숫자 값 또는 AUTOMATIC으로 설정되어 있음
   - **locklist**, **sheapthres_shr**, **pckcachesz** 또는 버퍼 풀 *크기* 중 두 개가 AUTOMATIC으로 설정되어 있음
   - **sortheap** 데이터베이스 구성 매개변수가 AUTOMATIC으로 설정되어 있음
2. **locklist** 데이터베이스 구성 매개변수의 값이 **maxlocks** 데이터베이스 구성 매개변수와 함께 조정됩니다. **locklist** 매개변수의 자체 성능 조정을 사용 불가능하게 하면 **maxlocks** 매개변수의 자체 성능 조정이 자동으로 사용 불가능해지고, **locklist** 매개변수의 자체 성능 조정을 사용 가능하게 하면 **maxlocks** 매개변수의 자체 성능 조정이 자동으로 사용 가능해집니다.
3. **sortheap** 또는 **sheapthres_shr** 데이터베이스 구성 매개변수 자동 성능 조정은 데이터베이스 관리 프로그램 구성 매개변수 **sheapthres**가 0으로 설정된 경우에만 허용됩니다.
4. **sortheap**의 값은 **sheapthres_shr**과 함께 조정됩니다. **sortheap** 매개변수의 자체 성능 조정을 사용 불가능하게 하면 **sheapthres_shr** 매개변수의 자체 성능 조정이 자동으로 사용 불가능해지고, **sheapthres_shr** 매개변수의 자체 성능 조정을 사용 가능하게 하면 **sortheap** 매개변수의 자체 성능 조정이 자동으로 사용 가능해집니다.

5. 자체 성능 조정 메모리는 고가용성 재해 복구(HADR) 기본 서버에서만 실행됩니다. HADR 시스템에서 자체 성능 조정 메모리가 활성화된 경우, 보조 서버에서는 실행되지 않으며, 구성이 제대로 설정된 경우에만 기본 서버에서 실행됩니다. HADR 데이터베이스 역할이 전환된 경우, 자체 성능 조정 메모리 조작도 전환되어 새 기본 서버에서 실행됩니다. 기본 데이터베이스가 시작되거나, 대기 데이터베이스가 인계를 통해 기본 데이터베이스로 변환된 후, 첫 번째 클라이언트가 연결될 때까지 자체 성능 조정 메모리 관리자(STMM) 엔진 디스패치 가능 단위(EDU)가 시작되지 않을 수도 있습니다.

자체 성능 조정 메모리 사용 안함

전체 데이터베이스 또는 하나 이상의 구성 매개변수 또는 버퍼 풀에 대해 자체 성능 조정 메모리를 사용하지 않을 수 있습니다.

전체 데이터베이스에 대해 자체 성능 조정 메모리가 사용되지 않는 경우, AUTOMATIC으로 설정된 버퍼 풀과 메모리 구성 매개변수는 자동 성능 조정이 계속 사용 가능합니다. 그러나 메모리 영역은 현재 *크기*로 유지됩니다.

1. UPDATE DATABASE CONFIGURATION 명령 또는 db2CfgSet API를 사용해 **self_tuning_mem** 데이터베이스 구성 매개변수를 해제(OFF)로 설정하여 데이터베이스에 대해 자체 성능 조정 메모리를 사용하지 않도록 하십시오.
2. 메모리 구성 매개변수에 의해 제어되는 메모리 영역의 자체 성능 조정을 사용 불가능하게 하려면, UPDATE DATABASE CONFIGURATION 명령 또는 db2CfgSet API를 사용하여 관련 구성 매개변수를 MANUAL로 설정하거나 숫자 매개변수 값을 지정하십시오.
3. 버퍼 풀의 자체 성능 조정을 사용 불가능하게 하려면 CREATE BUFFERPOOL문 또는 ALTER BUFFERPOOL문을 사용하여 버퍼 풀 *크기*를 특정 값으로 설정하십시오.

주:

- 일부 경우에는, 다른 관련 메모리 구성 매개변수 역시 사용될 경우에만 메모리 구성 매개변수가 자체 성능 조정에 사용될 수 있습니다. 즉, 예를 들어, **locklist** 또는 **sortheap** 데이터베이스 구성 매개변수에 대한 자체 성능 조정 메모리를 사용 불가능하게 하면 각각 **maxlocks** 또는 **sheapthres_shr** 데이터베이스 구성 매개변수에 대한 자체 성능 조정 메모리가 사용 불가능해집니다.

자체 성능 조정을 사용 가능한 메모리 소비자 판별

구성 매개변수에 의해 제어되거나 버퍼 풀에 적용되는 자체 성능 조정 메모리 설정을 볼 수 있습니다.

- 명령행에서 구성 매개변수에 대한 설정을 보려면 SHOW DETAIL 옵션을 지정하여 GET DATABASE CONFIGURATION 명령을 사용하십시오. 자체 성능 조정을 사용할 수 있는 메모리 소비자가 출력에서 다음과 같이 그룹화됩니다.

```
설명                                    매개변수                          현재 값                  지연된 값
-------------------------------------------------------------------------------------------------------------
자체 조정 메모리                        (SELF_TUNING_MEM) = ON (Active)   ON
데이터베이스 공유 메모리 크기(4KB)      (DATABASE_MEMORY) = AUTOMATIC(37200)  AUTOMATIC(37200)
잠금 목록용 최대 스토리지(4KB)          (LOCKLIST) = AUTOMATIC(7456)      AUTOMATIC(7456)
잠금 목록의 잠금 목록의 백분율          (MAXLOCKS) = AUTOMATIC(98)        AUTOMATIC(98)
패키지 캐시 크기(4KB)                   (PCKCACHESZ) = AUTOMATIC(5600)    AUTOMATIC(5600)
공유 정렬의 정렬 힙 임계값(4KB)         (SHEAPTHRES_SHR) = AUTOMATIC(5000)  AUTOMATIC(5000)
정렬 목록 힙(4KB)                       (SORTHEAP) = AUTOMATIC(256)       AUTOMATIC(256)
```

- 또한 db2CfgGet API를 사용하여 조정이 사용 가능한지 여부를 판별할 수 있습니다. 다음 값이 리턴됩니다.

  ```
  SQLF_OFF                  0
  SQLF_ON_ACTIVE            2
  SQLF_ON_INACTIVE          3
  ```

  SQLF_ON_ACTIVE는 자체 성능 조정이 사용 가능하고 활성임을 표시하는 반면, SQLF_ON_INACTIVE는 자체 성능 조정이 사용 가능하지만 현재 비활성임을 표시합니다.

버퍼 풀에 대한 자체 성능 조정 설정을 보려면 다음 방법 중 하나를 사용하십시오.

- 명령행에서 자체 성능 조정을 사용 가능한 버퍼 풀의 목록을 검색하려면 다음 쿼리를 사용하십시오.

  ```
  SELECT BPNAME, NPAGES FROM SYSCAT.BUFFERPOOLS
  ```

  버퍼 풀에 대해 자체 성능 조정이 사용 가능한 경우, 해당 특정 버퍼 풀에 대한 SYSCAT.BUFFERPOOLS의 NPAGES 필드가 -2로 설정됩니다. 자체 성능 조정이 사용 불가능한 경우, NPAGES 필드가 버퍼 풀의 현재 *크기*로 설정됩니다.

- 자체 성능 조정이 사용 가능한 버퍼 풀의 현재 *크기*를 판별하려면, GET SNAPSHOT 명령을 사용하고 버퍼 풀의 현재 *크기*를 확인하십시오(**bp_cur_buffsz** 모니터 요소의 값).

  ```
  GET SNAPSHOT FOR BUFFERPOOLS ON database-alias
  ```

  특정 데이터베이스 파티션의 버퍼 풀의 *크기*를 지정하는 ALTER BUFFERPOOL문은 SYSCAT.BUFFERPOOLDBPARTITIONS 카탈로그 뷰에서 해당 버퍼 풀에 대한 예외 항목을 작성합니다(또는 기존 항목 갱신). 버퍼 풀에 대한 예외 항목이 존재하는 경우, 디폴트 버퍼 풀 *크기*가 AUTOMATIC으로 설정되어 있으면 해당 버퍼 풀이 자체 성능 조정 조작에 참여하지 않습니다.

메모리 소비자 *크기* 조정에 필요한 시간에 의해 메모리 조정 프로그램의 반응성이 제한된다는 점을 유의하는 것이 중요합니다. 예를 들어, 버퍼 풀의 *크기*를 줄이면 프로세스가 길어질 수 있고, 버퍼 풀 메모리와 정렬 메모리를 교환하는 성능 이점이 즉시 실현되지 않을 수도 있습니다.

파티션된 데이터베이스 환경에서 자체 성능 조정 메모리

파티션된 데이터베이스 환경에서 자체 성능 조정 메모리 기능을 사용할 경우, 이 기능으로 시스템을 적절히 조정할지 여부를 판별하는 몇 가지 인수가 있습니다.

파티션된 데이터베이스에 대해 자체 성능 조정 메모리가 사용될 경우, 단일 데이터베이스 파티션이 조정 파티션으로 지정되고 모든 메모리 조정 결정은 해당 데이터베이스 파티션의 메모리 및 워크로드 특성을 기반으로 합니다. 해당 파티션에 대한 조정이 결정된 후, 다른 데이터베이스 파티션에 메모리 조정이 분산되어 모든 데이터베이스 파티션이 유사한 구성을 유지하도록 해 줍니다.

단일 조정 파티션 모델에서는 모든 데이터베이스 파티션이 유사한 메모리 요구사항을 지니고 있을 경우에만 기능이 사용된다고 가정합니다. 파티션된 데이터베이스에서 자체 성능 조정 메모리를 사용할지 여부를 결정할 때 다음 지침을 사용하십시오.

파티션된 데이터베이스에 대한 자체 성능 조정 메모리가 권장되는 경우

모든 데이터베이스 파티션이 유사한 메모리 요구사항을 지니고 유사한 하드웨어에서 실행되고 있는 경우, 자체 성능 조정 메모리를 수정 없이 사용할 수 있습니다. 이러한 유형의 환경은 다음 특성을 공유합니다.

- 모든 데이터베이스 파티션이 동일한 하드웨어에 있고, 여러 논리적 데이터베이스 파티션이 여러 물리적 데이터베이스 파티션에 고르게 분산되어 있습니다.
- 데이터가 완벽하거나 거의 완벽하게 분산되어 있습니다.
- 워크로드가 데이터베이스 파티션에 걸쳐 고르게 분산되어 있습니다. 즉, 다른 데이터베이스 파티션보다 하나 이상의 힙에 대해 더 높은 메모리 요구사항을 지닌 데이터베이스 파티션이 없습니다.

이러한 환경에서 모든 데이터베이스 파티션이 균등하게 구성되어 있는 경우, 자체 성능 조정 메모리가 시스템을 적절히 구성합니다.

파티션된 데이터베이스에 대한 자체 성능 조정 메모리가 조건부로 권장되는 경우

환경에서 대부분의 데이터베이스 파티션이 유사한 메모리 요구사항을 지니고 유사한 하드웨어에서 실행되고 있는 경우, 초기 구성에서 약간의 주의를 기울이기만 하면 자체 성능 조정 메모리를 사용할 수 있습니다. 이러한 시스템에는 데이터에 대한 하나의 데이터베이스 파티션 세트와 훨씬 더 작은 코디네이터 파티션 및 카탈로그 파티션 세트가 있을 수 있습니다. 이런 환경에서는 코디네이터 파티션과 카탈로그 파티션을 데이터가 포함된 데이터베이스 파티션과 다르게 구성하는 것이 좋습니다.

데이터가 포함된 모든 데이터베이스 파티션에서 자체 성능 조정 메모리가 사용되어야 하고, 이러한 데이터베이스 파티션 중 하나가 조정 파티션으로 지정되어야 합니다. 그리고 코디네이터 파티션과 카탈로그 파티션이 다르게 구성될 수 있기 때문에 해당 파티션에서 자체 성능 조정 메모리를 사용하지 않아야 합니다. 코디네이터 파티션과 카탈로그 파티션에서 자체 성능 조정 메모리를 사용하지 않으려면 이러한 파티션에서 **self_tuning_mem** 데이터베이스 구성 매개변수를 해제(OFF)로 설정하십시오.

파티션된 데이터베이스에 대한 자체 성능 조정 메모리가 권장되지 않는 경우

각 데이터베이스 파티션의 메모리 요구사항이 다른 경우, 또는 크게 다른 하드웨어에서 다른 데이터베이스 파티션이 실행되고 있는 경우 자체 성능 조정 메모리 기능을 사용하지 않는 것이 좋습니다. 모든 파티션에서 **self_tuning_mem** 데이터베이스 구성 매개변수를 해제(OFF)로 설정하여 이 기능을 사용하지 않을 수 있습니다.

다른 데이터베이스 파티션의 메모리 요구사항 비교

다른 데이터베이스 파티션의 메모리 요구사항이 충분히 유사한지 여부를 판별하는 가장 좋은 방법은 스냅샷 모니터를 참조하는 것입니다. 다음 스냅샷 요소가 모든 데이터베이스 파티션에서 유사할 경우(20% 이하 차이), 데이터베이스 파티션이 메모리 요구사항이 충분히 유사한 것으로 고려될 수 있습니다.

`get snapshot for database on <dbname>` 명령을 발행하여 다음 데이터를 수집하십시오.

```
현재 보유된 잠금 수                      = 0
잠금 대기 수                             = 0
데이터베이스의 잠금 대기 시간(밀리초)     = 0
사용 중인 잠금 목록 메모리(바이트)        = 4968
잠금 에스컬레이션 수                     = 0
배타적 잠금 에스컬레이션 수              = 0

할당된 총 공유 정렬 힙                   = 0
공유 정렬 힙 상위 워터 마크              = 0
포스트 임계값 정렬(공유 메모리)          = 0
정렬 오버플로                            = 0

패키지 캐시 찾아보기 수                  = 13
패키지 캐시 삽입 수                      = 1
패키지 캐시 오버플로우 수                = 0
패키지 캐시 상위 워터 마크(바이트)       = 655360

해시 조인 수                             = 0
해시 루프 수                             = 0
해시 조인 오버플로우 수                  = 0
작은 해시 조인 오버플로우 수             = 0
포스트 임계값 해시 조인 수(공유 메모리)  = 0

OLAP 기능 수                             = 0
OLAP 기능 오버플로우 수                  = 0
활성 OLAP 기능                           = 0
```

`get snapshot for bufferpools on <dbname>` 명령을 발행하여 다음 데이터를 수집하십시오.

```
버퍼 풀 데이터 논리적 읽기 수            = 0
버퍼 풀 데이터 실제 읽기 수              = 0
버퍼 풀 인덱스 논리적 읽기 수            = 0
```

```
버퍼 풀 인덱스 실제 읽기 수                = 0
전체 버퍼 풀 읽기 시간(밀리초)             = 0
전체 버퍼 풀 쓰기 시간(밀리초)             = 0
```

파티션된 데이터베이스 환경에서 자체 성능 조정 메모리 사용

파티션된 데이터베이스 환경에서 자체 성능 조정 메모리를 사용할 경우, 메모리 구성을 모니터하고 구성 변경사항을 다른 모든 데이터베이스 파티션에 전파하여 모든 참여 데이터베이스 파티션 간에 일관성 있는 구성을 유지하는 단일 데이터베이스 파티션(*조정 파티션*이라고 함)이 있습니다.

조정 파티션은 버퍼 풀의 수 및 파티션 그룹에서 데이터베이스 파티션의 수와 같은 여러 특성을 기반으로 선택됩니다.

- 현재 어떤 데이터베이스 파티션이 조정 파티션으로 지정되어 있는지 판별하려면 ADMIN_CMD 프로시저를 다음과 같이 호출하십시오.

  ```
  CALL SYSPROC.ADMIN_CMD('get stmm tuning dbpartitionnum')
  ```

- 조정 파티션을 변경하려면 ADMIN_CMD 프로시저를 다음과 같이 호출하십시오.

  ```
  CALL SYSPROC.ADMIN_CMD('update stmm tuning dbpartitionnum <partitionnum>')
  ```

  조정 파티션은 비동기적으로 또는 다음 데이터베이스 시작 시 갱신됩니다. 메모리 조정 프로그램에서 조정 파티션을 자동으로 선택하도록 하려면 *partitionnum* 값으로 -1을 입력하십시오.

파티션된 데이터베이스 환경에서 메모리 조정 프로그램 시작

파티션된 데이터베이스 환경에서는, 자체 성능 조정 메모리를 사용하려면 모든 파티션이 활성이어야 하므로, 명시적 ACTIVATE DATABASE 명령에 의해 데이터베이스가 활성화된 경우에만 메모리 조정 프로그램이 시작됩니다.

특정 데이터베이스 파티션에 대해 자체 성능 조정 메모리 사용 안함

- 데이터베이스 파티션의 서브세트에 대해 자체 성능 조정 메모리를 사용하지 않으려면 해당 데이터베이스 파티션에 대해 **self_tuning_mem** 데이터베이스 구성 매개변수를 해제(OFF)로 설정하십시오.
- 특정 데이터베이스 파티션의 구성 매개변수에 의해 제어되는 메모리 소비자의 서브세트에 대해 자체 성능 조정 메모리를 사용하지 않으려면 관련 구성 매개변수의 값 또는 버퍼 풀 *크기*를 MANUAL 또는 해당 데이터베이스 파티션에 대한 특정 값으로 설정하십시오. 실행되는 모든 파티션에서 자체 성능 조정 메모리 구성 매개변수 값이 일관성이 있는 것이 좋습니다.
- 특정 데이터베이스 파티션의 특정 버퍼 풀에 대해 자체 성능 조정 메모리를 사용하지 않으려면 자체 성능 조정 메모리가 사용되지 않을 파티션과 *크기* 값을 지정하는 ALTER BUFFERPOOL문을 발행하십시오.

특정 데이터베이스 파티션의 버퍼 풀의 *크기*를 지정하는 ALTER BUFFERPOOL문은 SYSCAT.BUFFERPOOLDBPARTITIONS 카탈로그 뷰에서 해당 버퍼 풀에 대한 예외 항목을 작성합니다(또는 기존 항목 갱신). 버퍼 풀에 대한 예외 항목이 존재하는 경우, 디폴트 버퍼 풀 *크기*가 AUTOMATIC으로 설정되어 있으면 해당 버퍼 풀이 자체 성능 조정 조작에 참여하지 않습니다. 버퍼 풀이 자체 성능 조정에 사용될 수 있도록 예외 항목을 제거하려면 다음을 수행하십시오.

1. ALTER BUFFERPOOL문을 발행하고 버퍼 풀 *크기*를 특정 값으로 설정하여 이 버퍼 풀에 대해 자체 성능 조정을 사용하지 마십시오.
2. 다른 ALTER BUFFERPOOL문을 발행하여 이 데이터베이스 파티션의 버퍼 풀의 *크기*를 디폴트로 설정하십시오.
3. 다른 ALTER BUFFERPOOL문을 발행하고 버퍼 풀 *크기*를 AUTOMATIC으로 설정하여 이 버퍼 풀에 대해 자체 성능 조정을 사용하십시오.

비균일 환경에서 자체 성능 조정 메모리 사용

원칙적으로는, 데이터가 모든 데이터베이스 파티션 간에 고르게 분산되어야 하고, 각 파티션에서 실행되는 워크로드에 유사한 메모리 요구사항이 있어야 합니다. 데이터 분산이 비대칭으로 되어 하나 이상의 데이터베이스 파티션에 다른 데이터베이스 파티션보다 상당히 많거나 적은 데이터가 포함될 경우, 이러한 이례적 데이터베이스 파티션이 자체 성능 조정에 사용되지 않아야 합니다. 데이터베이스 파티션 간에 메모리 요구사항이 비대칭인 경우에도 마찬가지입니다. 이러한 경우는 예를 들어, 자원 중심 정렬이 한 파티션에서만 수행될 경우나 일부 데이터베이스 파티션이 다른 데이터베이스 파티션보다 더 많은 메모리 및 다른 하드웨어와 연관된 경우에 발생할 수 있습니다. 이런 유형의 환경에서 일부 데이터베이스 파티션에 자체 성능 조정 메모리가 여전히 사용될 수 있습니다. 비대칭 환경에서 자체 성능 조정 메모리를 이용하려면 유사한 데이터 및 메모리 요구사항을 지닌 데이터베이스 파티션 세트를 식별하고 자체 성능 조정에 사용하십시오. 나머지 파티션의 메모리는 수동으로 구성해야 합니다.

메모리 및 메모리 힙 구성

단순화된 메모리 구성 기능을 통해 대부분의 메모리 관련 구성 매개변수에 디폴트 AUTOMATIC 설정을 사용함으로써 조정을 줄여 DB2 Data Server에서 필요한 메모리 및 메모리 힙을 구성할 수 있습니다.

단순화된 메모리 구성 기능을 사용하면 다음과 같은 이점이 있습니다.

- 단일 매개변수 **instance_memory**를 사용하여 데이터베이스 관리 프로그램이 개인용 및 공유 메모리 힙에서 할당할 수 있는 모든 메모리를 지정할 수 있습니다. 또한 **appl_memory** 구성 매개변수를 사용하여 DB2 데이터베이스 에이전트가 서비스 응용프로그램 요청에 할당하는 최대 응용프로그램 메모리 *크기*를 제어할 수 있습니다.
- 기능 메모리에만 사용되는 매개변수를 수동으로 조정할 필요가 없습니다.

- Memory Visualizer를 사용하여 데이터베이스 관리 프로그램의 개인용 및 공유 메모리 힙에서 현재 사용 중인 총 메모리 *크기*를 쿼리할 수 있습니다. 또한 db2mtrk 명령을 사용하여 힙 사용량을 모니터하고 ADMIN_GET_DBP_MEM_USAGE() 테이블 함수를 사용하여 전체 메모리 사용량을 모니터할 수 있습니다.
- 디폴트 DB2 구성에서는 조정이 훨씬 적게 필요하며 이는 작성되는 새 인스턴스의 이점입니다.

다음 표에는 값이 디폴트로 AUTOMATIC으로 설정된 메모리 구성 매개변수가 나열되어 있습니다. 필요에 따라 이들 매개변수를 동적으로 구성할 수도 있습니다. AUTOMATIC 설정의 의미는 가장 오른쪽 컬럼의 설명대로 각 매개변수마다 다른 점에 유의하십시오.

표 3. 값이 디폴트로 AUTOMATIC으로 설정된 메모리 구성 매개변수

구성 매개변수 이름	설명	AUTOMATIC 설정의 의미
appl_memory	DB2 데이터베이스 에이전트가 서비스 응용프로그램 요청에 할당하는 응용프로그램 메모리의 최대 *크기*를 제어합니다.	AUTOMATIC으로 설정하면 데이터베이스 파티션에서 할당하는 메모리 총 *크기*가 **instance_memory** 한계를 넘지 않는 한 모든 응용프로그램 메모리 요청이 허용됩니다.
applheapsz	버전 9.5 이전에는 응용프로그램과 관련하여 작업 중인 각 데이터베이스 에이전트가 사용할 수 있는 응용프로그램 메모리 *크기*를 가리켰습니다. 버전 9.5에서 이 매개변수는 전체 응용프로그램에서 사용할 수 있는 응용프로그램 메모리의 총 *크기*를 가리킵니다. 파티션된 데이터베이스 환경, 집중기 또는 SMP 구성에서 이는 AUTOMATIC 설정을 사용하지 않는 경우에는 이전 릴리스에서 사용된 **applheapsz** 값을 늘려야 함을 의미합니다.	AUTOMATIC으로 설정하면 **appl_memory** 또는 **instance_memory** 한계에 도달할 때까지 필요한 만큼 응용프로그램 힙 *크기*를 늘릴 수 있습니다.
database_memory(버전 9.5 이전에는 디폴트 설정 AUTOMATIC이 Windows 및 AIX 플랫폼에만 적용되었습니다. 버전 9.5에서는 AUTOMATIC이 모든 DB2 서버 제품의 디폴트 설정입니다.)	데이터베이스 공유 메모리 영역에 예약된 공유 메모리 *크기*를 지정합니다.	자체 성능 조정이 사용되는 경우, 메모리 조정 프로그램은 데이터베이스에 대한 전체 메모리 요구사항을 판별하고 현재 데이터베이스 요구사항에 따라 데이터베이스 공유 메모리에 할당되는 메모리의 양을 늘리거나 줄입니다.
dbheap	데이터베이스 힙에서 사용하는 최대 메모리를 판별합니다.	AUTOMATIC으로 설정하면 **database_memory** 또는 **instance_memory** 한계에 도달할 때까지 필요한 만큼 데이터베이스 힙을 늘릴 수 있습니다.
instance_memory	데이터베이스 파티션에 할당할 수 있는 최대 메모리 *크기*를 지정합니다.	AUTOMATIC으로 설정하면 전체 데이터베이스 관리 프로그램 인스턴스에서 사용하는 전체 메모리를 컴퓨터의 실제 RAM의 한계인 75 - 95%까지 늘릴 수 있습니다. 이 한계는 db2start 처리 중에 계산됩니다.
mon_heap_sz	데이터베이스 시스템 모니터 데이터에 할당할 메모리 *크기*(페이지)를 판별합니다.	AUTOMATIC으로 설정하면 **instance_memory** 한계에 도달할 때까지 필요한 만큼 모니터 힙을 늘릴 수 있습니다.

표 3. 값이 디폴트로 AUTOMATIC으로 설정된 메모리 구성 매개변수 (계속)

구성 매개변수 이름	설명	AUTOMATIC 설정의 의미
stat_heap_sz	RUNSTATS 명령을 사용하여 통계를 수집하는 데 사용되는 최대 힙 *크기*를 표시합니다.	AUTOMATIC으로 설정하면 **appl_memory** 또는 **instance_memory** 한계에 도달할 때까지 필요한 만큼 통계 힙 *크기*를 늘릴 수 있습니다.
stmtheap	SQL 또는 XQuery 컴파일러가 SQL 또는 XQuery 문을 컴파일하기 위해 작업 스페이스로 사용하는 명령문 힙의 *크기*를 지정합니다.	AUTOMATIC으로 설정하면 **appl_memory** 또는 **instance_memory** 한계에 도달할 때까지 필요한 만큼 명령문 힙을 늘릴 수 있습니다.

주: DBMCFG 및 DBCFG 관리 뷰에서는 모든 데이터베이스 파티션과 관련하여 현재 연결된 데이터베이스의 데이터베이스 관리 프로그램 구성 매개변수 정보를 검색합니다. **mon_heap_sz**, **stmtheap** 및 **stat_heap_sz** 구성 매개변수의 경우 해당 뷰의 DEFERRED_VALUE 컬럼은 데이터베이스 사용 중에 유지되지 않습니다. 즉, get dbm cfg show detail 또는 get db cfg show detail 명령을 실행하면 쿼리의 출력에 갱신된 (메모리에서) 값이 표시됩니다.

다음 표는 인스턴스 업그레이드나 작성 중 및 데이터베이스 업그레이드나 작성 중에 구성 매개변수를 디폴트 AUTOMATIC 값으로 설정하는지 여부를 표시합니다.

표 4. 인스턴스와 데이터베이스 업그레이드 및 작성 중에 AUTOMATIC으로 설정되는 구성 매개변수

구성 매개변수	인스턴스 업그레이드 또는 작성 시 AUTOMATIC으로 설정	데이터베이스 업그레이드 시 AUTOMATIC으로 설정	데이터베이스 작성 시 AUTOMATIC으로 설정
applheapsz[1]		X	X
dbheap		X	X
instance_memory	X		
mon_heap_sz[1]	X		
stat_heap_sz[1]		X	X
stmtheap[1]			X

메모리 구성을 단순화하는 과정에서 다음 요소가 더 이상 사용되지 않습니다.

- 구성 매개변수 **appgroup_mem_sz**, **groupheap_ratio** 및 **app_ctl_heap_sz**. 이 구성 매개변수는 새 **appl_memory** 구성 매개변수로 교체됩니다.
- db2mtrk 메모리 추적 프로그램 명령의 **-p** 매개변수. 개인용 에이전트 메모리 힙을 나열하는 이 옵션은 모든 응용프로그램 메모리 사용을 나열하는 **-a** 매개변수로 교체됩니다.

Memory Visualizer는 새 **appl_memory** 구성 매개변수를 사용하여 데이터베이스에서 소비하는 최대 응용프로그램 메모리를 표시하고, 갱신된 **instance_memory** 구성 매개변수를 사용하여 인스턴스에서 소비하는 최대 메모리를 표시합니다. Memory Visualizer는 또한 AUTOMATIC 설정을 허용하는 모든 구성 매개변수의 값을 표시합니다. 사용

되지 않는 구성 매개변수의 값은 버전 9.5 데이터베이스의 Memory Visualizer에 표시되지 않지만 이전 버전의 데이터베이스에서는 표시됩니다.

instance_memory 매개변수를 이 목록에 지정된 값보다 큰 값으로 갱신하려 하면 실패하며 리턴 코드는 SQL5130N입니다.

- DB2 Express™ Edition 및 DB2 Express-C의 경우 4GB(1 048 576 * 4KB 페이지)
- DB2 Workgroup Server Edition의 경우 16GB(4 194 304 * 4KB 페이지)

FCM(Fast Communication Manager) 공유 메모리가 할당되면 각 로컬 데이터베이스 파티션이 시스템의 전체 FCM 공유 메모리 크기를 공유하는 부분이 해당 데이터베이스 파티션의 **instance_memory** 한계에 대해 고려됩니다. FCM 메모리의 특성으로 인해 (FCM 버퍼 할당에 실패하면 인스턴스 작동이 중지됨) FCM 메모리 요청은 **instance_memory** 한계 때문에 절대 실패하지 않습니다. 그러나 운영 체제에서 메모리를 할당할 수 없는 경우에는 실패할 수 있습니다. FCM 메모리 요청으로 인해 데이터베이스 파티션이 **instance_memory** 한계를 초과하는 경우 파티션의 메모리 사용량이 **instance_memory** 한계 아래 수준으로 돌아갈 때까지 기타 메모리 요청은 실패합니다.

에이전트 및 프로세스 모델 구성

버전 9.5에서는 프로세스 모델 관련 매개변수를 구성하기 위해 덜 복잡하고 더 융통성 있는 메커니즘을 제공합니다. 이와 같은 단순화된 구성을 사용하면 해당 매개변수를 정기적으로 조정할 필요가 없고 매개변수를 구성하는 데 필요한 시간과 노력을 줄일 수 있습니다. 또한 새 값을 적용하기 위해 DB2 인스턴스를 종료하고 재시작할 필요도 없습니다.

동적 및 자동 에이전트와 메모리 구성을 허용하려면 인스턴스 활성화 시 메모리 자원이 약간 더 필요합니다.

에이전트, 프로세스 모델 및 메모리 구성 개요

DB2 Data Server는 32비트 및 64비트 플랫폼에서 멀티스레드 아키텍처를 사용하여 모든 운영 체제에서 향상된 사용 편리성, 자원 공유, 메모리 풋프린트 감소 및 일관성 있는 스레딩 아키텍처와 같은 여러 가지 이점을 제공합니다.

다음 표에는 범주별로 에이전트, 프로세스 및 메모리 구성 항목이 표시되어 있습니다.

표 5. *에이전트, 프로세스 및 메모리 구성 정보 개요*

범주	관련 항목
일반 정보, 제한사항 및 비호환성	• 44 페이지의 『메모리 및 메모리 힙 구성』 • 47 페이지의 『에이전트 및 프로세스 모델 구성』 • *문제점 해결 및 데이터베이스 성능 조정*의 『DB2 프로세스 모델』 • 50 페이지의 『다중 파티션에서 데이터베이스 구성』
설치 및 업그레이드	• *DB2 Connect 사용자 안내서*의 『연결 집중기(connection concentrator)』 • *DB2 Connect 사용자 안내서*의 『DB2® Connect™ 조정』 • *DB2 Connect 사용자 안내서*의 『OS/390® 및 zSeries® SYSPLEX 사용에 대한 고려사항』 • *DB2 Server 설치*의 『디스크 및 메모리 요구사항』 • *DB2 Server 설치*의 『커널 매개변수 수정(Linux)』 • *DB2 버전 9.7로 업그레이드*의 『DB2 서버 동작 변경사항』
성능	• *문제점 해결 및 데이터베이스 성능 조정*의 『클라이언트 연결을 위한 연결 집중기(connection concentrator) 향상 기능』 • *문제점 해결 및 데이터베이스 성능 조정*의 『데이터베이스 에이전트』 • *문제점 해결 및 데이터베이스 성능 조정*의 『데이터베이스 에이전트 관리』 • *문제점 해결 및 데이터베이스 성능 조정*의 『데이터베이스 관리 프로그램 공유 메모리』 • *문제점 해결 및 데이터베이스 성능 조정*의 『DB2에서 메모리 할당』 • *문제점 해결 및 데이터베이스 성능 조정*의 『메모리 할당 매개변수 조정』

표 5. 에이전트, 프로세스 및 메모리 구성 정보 개요 (계속)

범주	관련 항목
명령, API, 레지스트리 변수, 함수 및 루틴	• *명령어 참조서*의 『db2pd - DB2 데이터베이스 명령 모니터 및 문제점 해결』 • *명령어 참조서*의 『DATABASE MANAGER CONFIGURATION 명령』 • *명령어 참조서*의 『RESET DATABASE MANAGER CONFIGURATION 명령』 • *명령어 참조서*의 『UPDATE DATABASE MANAGER CONFIGURATION 명령』 • *명령어 참조서*의 『db2mtrk - 메모리 추적 프로그램 명령』 • *관리 API 참조서*의 『sqlfupd 데이터 구조』 • 52 페이지의 『공유된 파일 핸들 테이블』 • 52 페이지의 『분리 모드 프로세스에서 벤더 라이브러리 함수 실행』 • *관리 루틴 및 뷰*의 『ADMIN_GET_DBP_MEM_USAGE - 전체 메모리 소비량 테이블 함수 가져오기』 • *SQL 참조서, 볼륨 1*의 『SQL 및 XML 한계』 • *SQL 참조서, 볼륨 1*의 『SYSCAT.PACKAGES 카탈로그 뷰』 • *관리 루틴 및 뷰*의 『DBMCFG 관리 뷰 - 데이터베이스 관리 프로그램 구성 매개변수 정보 검색』 • *관리 루틴 및 뷰*의 『ADMIN_CMD procedure–Run 관리 명령』 • *데이터베이스 모니터링 안내서 및 참조서*의 『Memory Visualizer 개요』 • *데이터베이스 모니터링 안내서 및 참조서*의 『Memory Visualizer와 작업』
구성 매개변수	• 682 페이지의 『구성 매개변수 요약』 • 784 페이지의 『appl_memory - 응용프로그램 메모리 구성 매개변수』 • 785 페이지의 『applheapsz - 응용프로그램 힙 크기』 • 798 페이지의 『database_memory - 데이터베이스 공유 메모리 크기』 • 800 페이지의 『dbheap - 데이터베이스 힙』 • 735 페이지의 『instance_memory - 인스턴스 메모리』 • 825 페이지의 『locklist - 잠금 목록의 최대 스토리지』 • 742 페이지의 『max_connections - 클라이언트 연결의 최대수』 • 743 페이지의 『max_coordagents - 최대 코디네이팅 에이전트 수』 • 841 페이지의 『maxappls - 최대 활성 응용프로그램 수』 • 748 페이지의 『mon_heap_sz - 데이터베이스 시스템 모니터 힙 크기』 • 752 페이지의 『num_poolagents - 에이전트 풀 크기』 • 878 페이지의 『stat_heap_sz - 통계 힙 크기』 • 878 페이지의 『stmtheap - 명령문 힙 크기』

표 5. 에이전트, 프로세스 및 메모리 구성 정보 개요 (계속)

범주	관련 항목
모니터 요소	• *데이터베이스 모니터링 안내서 및 참조서*의 『에이전트 및 연결』 • *데이터베이스 모니터링 안내서 및 참조서*의 『agents_from_pool - 풀로부터 할당된 에이전트 수』 • *데이터베이스 모니터링 안내서 및 참조서*의 『agents_registered - 등록된 에이전트 수』 • *데이터베이스 모니터링 안내서 및 참조서*의 『agents_registered_top - 등록된 최대 에이전트 수』 • *데이터베이스 모니터링 안내서 및 참조서*의 『agents_stolen - 분실 에이전트』 • *데이터베이스 모니터링 안내서 및 참조서*의 『appls_in_db2 - 데이터베이스에서 현재 실행 중인 응용프로그램』 • *데이터베이스 모니터링 안내서 및 참조서*의 『associated_agents_top - 최대 연관 에이전트 수』 • *데이터베이스 모니터링 안내서 및 참조서*의 『coord_agents_top - 최대 코디네이팅 에이전트 수』 • *데이터베이스 모니터링 안내서 및 참조서*의 『local_cons - 로컬 연결』 • *데이터베이스 모니터링 안내서 및 참조서*의 『local_cons_in_exec - 데이터베이스 관리 프로그램에서 실행 중인 로컬 연결』 • *데이터베이스 모니터링 안내서 및 참조서*의 『num_gw_conn_switches - 최대 에이전트 오버플로우 수』 • *데이터베이스 모니터링 안내서 및 참조서*의 『rem_cons_in - 데이터베이스 관리 프로그램과의 리모트 연결 수』 • *데이터베이스 모니터링 안내서 및 참조서*의 『rem_cons_in_exec - 데이터베이스 관리 프로그램에서 실행 중인 리모트 연결 수』

다중 파티션에서 데이터베이스 구성

데이터베이스 관리 프로그램은 다중 파티션의 모든 데이터베이스 구성 요소를 볼 수 있는 단일 뷰를 제공합니다. 이는 각 데이터베이스 파티션에 대해 db2_all 명령을 호출하지 않아도 모든 데이터베이스 파티션에서 데이터베이스 구성을 갱신하거나 재설정할 수 있음을 의미합니다.

데이터베이스가 상주하는 파티션에서 하나의 관리 명령 또는 하나의 SQL문을 실행하여 파티션에서 구성을 갱신할 수 있습니다. 디폴트로, 데이터베이스 구성 갱신 또는 재설정 방법은 모든 *데이터베이스 파티션*에 있습니다.

명령 스크립트 및 응용프로그램의 이전 버전과의 호환을 위해 세 가지 옵션을 사용할 수 있습니다.

- 다음과 같이 db2set 명령을 사용하여 **DB2_UPDDBCFG_SINGLE_DBPARTITION** 레지스트리 변수를 TRUE로 설정하십시오.

```
DB2_UPDDBCFG_SINGLE_DBPARTITION=TRUE
```

주: 레지스트리 변수 설정은 ADMIN_CMD 프로시저를 사용하여 작성한 UPDATE DATABASE CONFIGURATION 또는 RESET DATABASE CONFIGURATION 요청에는 적용되지 않습니다.

- **DBPARTITIONNUM** 매개변수를 UPDATE DATABASE CONFIGURATION 또는 RESET DATABASE CONFIGURATION 명령이나 ADMIN_CMD 프로시저와 함께 사용하십시오. 예를 들어, 모든 데이터베이스 파티션에서 데이터베이스 구성을 갱신하려면 다음과 같이 ADMIN_CMD 프로시저를 호출하십시오.

  ```
  CALL SYSPROC.ADMIN_CMD
    ('UPDATE DB CFG USING sortheap 1000')
  ```

  단일 데이터베이스 파티션을 갱신하려면 다음과 같이 ADMIN_CMD 프로시저를 호출하십시오.

  ```
  CALL SYSPROC.ADMIN_CMD
    ('UPDATE DB CFG DBPARTITIONNUM 10 USING sortheap 1000')
  ```

- **DBPARTITIONNUM** 매개변수를 db2CfgSet API와 함께 사용하십시오. **db2Cfg** 구조의 플래그는 데이터베이스 구성에 필요한 값을 단일 데이터베이스 파티션에 적용할지 여부를 나타냅니다. 플래그를 설정하는 경우 다음과 같이 **DBPARTITIONNUM** 값도 제공해야 합니다.

  ```
  #define db2CfgSingleDbpartition           256
  ```

  db2CfgSingleDbpartition 값을 설정하지 않으면 **DB2_UPDDBCFG_SINGLE_DBPARTITION** 레지스트리 변수를 TRUE로 설정하거나 *versionNumber*를 버전 9.5보다 낮은 버전 번호로 설정하지 않는 한, 데이터베이스 관리 프로그램이나 데이터베이스 구성 매개변수를 설정하는 db2CfgSet API와 관련하여 데이터베이스 구성에 필요한 값이 모든 데이터베이스 파티션에 적용됩니다.

데이터베이스를 버전 9.7로 업그레이드하면 기존 데이터베이스 구성 매개변수가 일반적인 규칙으로는 데이터베이스 업그레이드 이후에도 해당 값을 보유합니다. 그러나 디폴트값 및 일부 기존 매개변수를 사용하여 추가되는 새 매개변수는 새 버전 9.7 디폴트 값으로 설정됩니다. 기존 데이터베이스 구성 매개변수의 변경에 대한 자세한 내용은 *DB2 버전 9.7로 업그레이드*의 "DB2 서버 동작 변경" 주제를 참조하십시오. 업그레이드된 데이터베이스와 관련하여 후속 데이터베이스 구성 갱신 또는 재설정 요청은 디폴트로 모든 데이터베이스 파티션에 적용됩니다.

기존 갱신 또는 재설정 명령 스크립트의 경우 이전에 언급한 것과 동일한 규칙이 모든 데이터베이스 파티션에 적용됩니다. UPDATE DATABASE CONFIGURATION 또는 RESET DATABASE CONFIGURATION 명령의 **DBPARTITIONNUM** 옵션을 포함하도록 스크립트를 수정하거나 **DB2_UPDDBCFG_SINGLE_DBPARTITION** 레지스트리 변수를 설정할 수 있습니다.

db2CfgSet API를 호출하는 기존 응용프로그램에서는 버전 9.5 이상에 해당하는 명령어를 사용해야 합니다. 버전 9.5 이전의 동작을 수행하려면 **DB2_UPDDBCFG_SINGLE_DBPARTITION** 레지스트리 변수를 설정하거나 버전 9.5 이상의 버전 번호로 API를 호출하도록 응용프로그램을 수정할 수 있습니다(특정 데이터베이스 파티션의 데이터베이스 구성을 갱신하거나 재설정하기 위한 새 db2CfgSingleDbpartition 플래그 및 새 **dbpartitionnum** 필드 포함).

주: 데이터베이스 구성 값이 불일치하는 경우 각 데이터베이스 파티션을 개별적으로 갱신하거나 재설정할 수 있습니다.

공유된 파일 핸들 테이블

스레드된 데이터베이스 관리 프로그램은 각 데이터베이스 및 각 데이터베이스에서 작동하는 모든 에이전트에 대해 하나의 공유된 파일 핸들 테이블을 유지하므로 동일한 파일에 대한 I/O 요청 시 파일을 다시 열고 닫을 필요가 없습니다.

버전 9.5 이전에서 파일 핸들 테이블은 각 DB2 에이전트에 의해 별도로 유지되고 에이전트당 파일 핸들 테이블의 *크기*는 **maxfilop** 구성 매개변수에 의해 제어되었습니다. 버전 9.5부터는 데이터베이스 관리 프로그램이 전체 데이터베이스에 대해 하나의 공유된 파일 핸들 테이블을 유지하므로 동일한 데이터베이스 파일에서 작동하는 모든 에이전트 간에 동일한 파일 핸들을 공유할 수 있습니다. 따라서 **maxfilop** 구성 매개변수는 공유된 파일 핸들 테이블을 제어하는 데 사용됩니다.

이러한 변경사항으로 인해 버전 9.5에서는 **maxfilop** 구성 매개변수의 디폴트값이 다르고 최소값 및 최대값이 새로 지정되었습니다. 버전 9.5 이전의 릴리스에서 업그레이드하는 경우 데이터베이스 업그레이드 중 **maxfilop** 구성 매개변수는 이러한 디폴트값으로 자동으로 설정됩니다.

분리 모드 프로세스에서 벤더 라이브러리 함수 실행

데이터베이스 관리 프로그램은 데이터 압축, TSM 백업 및 로그 데이터 아카이브와 같은 태스크를 수행하는 분리 모드 프로세스에서 벤더 라이브러리 함수를 지원합니다.

버전 9.5 이전에서는 벤더 라이브러리 함수, 벤더 유틸리티 또는 루틴이 에이전트 프로세스 내부에서 실행되었습니다. 버전 9.5 이후부터는 DB2 데이터베이스 관리 프로그램 자체가 멀티스레드 응용프로그램이므로 벤더 라이브러리 함수는 더 이상 스레드 안전 상태가 아니고 DB2 데이터베이스에서 메모리 또는 스택 손상, 데이터 손상을 일으키지 않습니다. 이러한 이유로 인해 분리 모드 프로세스 내에서 실행되는 벤더 유틸리티 및 벤더 라이브러리 함수 또는 루틴의 각 호출에 대해 새로운 분리 모드 프로세스가 작성됩니다. 이렇게 해도 성능이 크게 저하되지 않습니다.

주: Windows 플랫폼의 경우 분리 모드 기능을 사용할 수 없습니다.

자동 스토리지

자동 스토리지는 테이블 스페이스의 스토리지 관리를 간편하게 합니다. 자동 스토리지 데이터베이스 작성 시 데이터베이스 관리 프로그램이 사용자의 데이터를 저장할 스토리지 경로가 지정됩니다. 그런 다음 데이터베이스 관리 프로그램이 컨테이너 및 사용자가 테이블 스페이스를 작성하여 채울 경우 테이블 스페이스의 스페이스 할당을 관리합니다.

스토리지 스페이스를 절약하기 위한 데이터 압축

DB2 제품에 빌드된 자동 테이블 및 인덱스 압축 성능을 사용하여 스토리지 요구사항을 줄일 수 있습니다.

테이블과 인덱스에는 보통 반복되는 정보가 들어있습니다. 이 반복은 전체 컬럼 값부터 공통 값에 대한 공통 접두부 또는 XML 데이터에서의 반복하는 패턴까지 다양할 수 있습니다. 테이블 및 인덱스를 저장하기 위해 필요한 스페이스를 줄이기 위해 사용할 수 있는 많은 압축 기능이 있습니다. 또한 압축이 제공할 수 있는 절약을 판별하는 데 도움이 되도록 채택할 수 있는 기능도 있습니다.

자동 통계 콜렉션

DB2 옵티마이저는 카탈로그 통계를 사용하여 쿼리에 대한 가장 효과적인 액세스 플랜을 판별합니다. 오래되거나 불완전한 테이블 또는 인덱스 통계는 옵티마이저에서 차선의 플랜을 선택하도록 하여 쿼리 실행을 느려지게 할 수도 있습니다. 그러나 지정된 워크로드에 대해 수집할 통계를 결정하는 일은 복잡하고, 이러한 통계를 최신 상태로 유지하는 데에는 시간이 많이 소모됩니다.

DB2 자동 테이블 유지보수 기능에 속하는 자동 통계 콜렉션을 통해 데이터베이스 관리 프로그램에서 통계를 갱신해야 하는지 여부를 판별하도록 할 수 있습니다. 자동 통계 콜렉션은 실시간 통계(RTS) 기능을 사용하여 명령문 컴파일 시간에 *동기적*으로 발생할 수 있습니다. 또는, runstats 유틸리티를 사용하여 *비동기* 콜렉션을 위해 백그라운드에서 단순히 실행되도록 할 수 있습니다. 실시간 통계 콜렉션이 사용되지 않는 동안 백그라운드 통계 콜렉션을 사용할 수 있다 하더라도 실시간 통계 콜렉션이 발생하도록 하려면 백그라운드 통계 콜렉션을 사용해야 합니다. 자동 백그라운드 통계 콜렉션은 새 데이터베이스를 작성할 때 디폴트로 사용됩니다. 자동 실시간 통계 콜렉션은 **auto_stmt_stats** 데이터베이스 구성 매개변수의 값을 설정(ON)으로 설정하여 사용할 수 있습니다. 이 매개변수는 **auto_runstats** 구성 매개변수의 하위입니다.

비동기 및 실시간 통계 콜렉션 이해

실시간 통계 콜렉션이 사용되는 경우 특정 메타데이터를 사용하여 통계를 제작할 수 있습니다. *제작*이란 일반 runstats 활동의 일부로 통계를 수집하기 보다는 통계를 파생하거나 작성하는 것을 의미합니다. 예를 들어, 테이블의 페이지 수, 페이지 *크기* 및 평균

행 너비에 대한 지식에서 테이블의 행 수를 파생할 수 있습니다. 일부 경우에는, 통계가 실제로 파생되지 않지만, 인덱스 및 데이터 관리 프로그램에 의해 유지보수되고 카탈로그에 직접 저장될 수 있습니다. 예를 들어, 인덱스 관리 프로그램이 각 인덱스에서 레벨 및 리프 페이지 수의 계수를 유지보수합니다.

쿼리 옵티마이저는 테이블 갱신 활동의 양(갱신, 삽입 또는 삭제 조작의 수) 및 쿼리에 대한 필요를 기반으로 통계가 수집되어야 하는 방법을 판별합니다.

실시간 통계 콜렉션은 보다 시기 적절하고 정확한 통계를 제공합니다. 정확한 통계는 더 나은 쿼리 실행 플랜과 향상된 성능을 가져올 수 있습니다. 실시간 통계 콜렉션이 사용되지 않는 경우 비동기 통계 콜렉션이 두 시간 간격으로 발생합니다. 이것은 일부 응용프로그램에 대해 정확한 통계를 제공하기에 충분하지 않은 빈도일 수 있습니다.

실시간 통계 콜렉션이 사용되는 경우에도 비동기 통계 콜렉션 검사가 여전히 두 시간 간격으로 발생합니다. 실시간 통계 콜렉션은 또한 다음 경우에 비동기 콜렉션 요청을 시작합니다.

- 테이블 활동이 동기 콜렉션을 필요로 할 만큼 충분히 높지 않지만 비동기 콜렉션을 필요로 할 만큼 충분히 높은 경우
- 테이블이 커서 동기 통계 콜렉션에서 샘플링을 사용한 경우
- 동기 통계가 제작된 경우
- 콜렉션 시간이 초과하여 동기 통계 콜렉션이 실패한 경우

동시에 최대 2개의 비동기 요청을 처리할 수 있습니다(서로 다른 테이블에 대해서만). 실시간 통계 콜렉션에 의해 하나의 요청이 시작되고 비동기 통계 콜렉션 검사에 의해 다른 요청이 시작되어야 합니다.

자동 통계 콜렉션의 성능 영향은 다음과 같은 여러 방법으로 최소화됩니다.

- 조절된 runstats 유틸리티를 사용하여 비동기 통계 콜렉션이 수행됩니다. 조절 기능은 현재 데이터베이스 활동을 기반으로, runstats 유틸리티에 의해 소비되는 자원의 양을 제어합니다. 데이터베이스 활동이 증가하면 유틸리티가 더 느리게 실행되어 자원 수요가 줄어듭니다.
- 동기 통계 콜렉션은 쿼리당 5초로 제한됩니다. 이 값은 RTS 최적화 지침에 의해 제어될 수 있습니다. 동기 콜렉션이 시간 제한을 초과할 경우, 비동기 콜렉션 요청이 제출됩니다.
- 동기 통계 콜렉션에서는 시스템 카탈로그에 통계를 저장하지 않습니다. 대신, 통계가 통계 캐시에 저장되고 나중에 비동기 조작에 의해 시스템 카탈로그에 저장됩니다. 이를 통해 시스템 카탈로그를 갱신할 때 발생할 수 있는 잠금 경합 및 오버헤드가 방지됩니다. 통계 캐시의 통계는 후속 SQL 컴파일 요청에 사용 가능합니다.
- 테이블당 하나의 동기 통계 콜렉션 조작만 발생합니다. 동기 통계 콜렉션을 필요로 하는 다른 에이전트에서는 통계를 제작하고, 가능한 경우, 통계 컴파일을 계속합니다.

이 동작은 서로 다른 데이터베이스 파티션의 에이전트에서 동기 통계를 요구할 수 있는 파티션된 데이터베이스 환경에서도 적용됩니다.

- 이전 데이터베이스 활동에 대한 정보를 사용하여 데이터베이스 워크로드에서 필요로 하는 통계를 판별하는 통계 프로파일링을 사용하거나 특정 테이블에 대한 사용자 고유의 통계 프로파일을 작성하여 수집되는 통계의 유형을 사용자 정의할 수 있습니다.
- 누락 통계 또는 높은 레벨의 활동(갱신, 삽입 또는 삭제 조작의 수로 측정된)이 있는 테이블만 통계 콜렉션 대상으로 고려됩니다. 테이블이 통계 콜렉션 기준을 충족한다 하더라도 쿼리 최적화에서 동기 통계를 요구하지 않는 한 동기 통계가 수집되지 않습니다. 일부 경우에는 쿼리 옵티마이저에서 통계 없이 액세스 플랜을 선택할 수 있습니다.
- 비동기 통계 콜렉션 검사에서는, 대형 테이블(400페이지가 넘는 테이블)을 샘플링하여 높은 테이블 활동이 통계를 변경했는지 여부를 판별합니다. 이러한 대형 테이블에 대한 통계는 보증된 경우에만 수집됩니다.
- 비동기 통계 콜렉션에서는, runstats 유틸리티가 유지보수 규정에서 지정된 최적 유지보수 창 동안 실행되도록 자동으로 스케줄됩니다. 이 규정은 또한 자동 통계 콜렉션의 범위 내에 있는 테이블 세트도 지정하여, 불필요한 자원 소비를 더욱 최소화합니다.
- 동기 통계 콜렉션과 제작은 동기 요청이 즉시 발생해야 하고 제한된 콜렉션 시간을 갖기 때문에 유지보수 규정에서 지정된 최적 유지보수 창을 따르지 않습니다. 동기 통계 콜렉션과 제작은 자동 통계 콜렉션의 범위 내에 있는 테이블 세트를 지정하는 규정을 따릅니다.
- 자동 통계 콜렉션이 수행되는 동안, 영향을 받는 테이블을 일반 데이터베이스 활동(갱신, 삽입 또는 삭제 조작)에 계속 사용할 수 있습니다.
- 별칭에 대해서는 실시간 통계(동기 또는 제작된) 수집되지 않습니다. 시스템 카탈로그에서 자동으로 별칭 통계를 새로 고치려면 (비동기 통계 콜렉션에 대해), SYSPROC.NNSTAT 프로시저를 호출하십시오.

실시간 동기 통계 콜렉션은 일반 테이블, 구체화된 쿼리 테이블(MQT) 및 전역 임시 테이블에 대해 수행됩니다. 전역 임시 테이블에 대해서는 비동기 통계가 수집되지 않습니다.

다음 경우 자동 통계 콜렉션(동기 또는 비동기)이 발생하지 않습니다.

- 통계 뷰
- VOLATILE이 표시된 테이블(SYSCAT.TABLES 카탈로그 뷰에 VOLATILE 필드가 설정된 테이블)
- SYSSTAT 카탈로그 뷰에 대해 직접 UPDATE문을 발행하여 해당 통계를 수동으로 갱신한 테이블

테이블 통계를 수동으로 수정할 경우, 데이터베이스 관리 프로그램에서는 사용자가 현재 통계 유지보수를 담당하고 있는 것으로 간주합니다. 데이터베이스 관리 프로그램에서 해당 통계를 수동으로 갱신한 테이블에 대한 통계를 유지보수하도록 하려면 LOAD 명령을 사용할 때 통계 콜렉션을 지정하거나 RUNSTATS 명령을 사용하여 통계를 수집하십시오. 업그레이드하기 전에 해당 통계를 수동으로 갱신한 버전 9.5 이전에 작성된 테이블은 영향을 받지 않으며, 해당 통계는 수동으로 갱신될 때까지 데이터베이스 관리 프로그램에 의해 자동으로 유지보수됩니다.

다음 경우 통계 작성이 발생하지 않습니다.

- 통계 뷰
- SYSSTAT 카탈로그 뷰에 대해 직접 UPDATE문을 발행하여 해당 통계를 수동으로 갱신한 테이블. 실시간 통계 콜렉션이 사용되지 않는 경우, 통계를 수동으로 갱신한 테이블에 대해 일부 통계 작성이 발생합니다.

파티션된 데이터베이스 환경에서는, 단일 데이터베이스 파티션에서 통계가 수집된 후 외삽됩니다. 데이터베이스 관리 프로그램에서는 항상 데이터베이스 파티션 그룹의 첫 번째 데이터베이스 파티션에서 통계를 수집합니다(동기와 비동기 모두).

데이터베이스 활성화 후 최소한 5분이 지날 때까지는 실시간 통계 콜렉션 활동이 발생하지 않습니다.

실시간 통계가 사용될 경우, 정의된 유지보수 창을 스케줄해야 합니다. 유지보수 창은 디폴트로 정의되어 있지 않습니다. 정의된 유지보수 창이 없는 경우 동기 통계 콜렉션만 발생합니다. 이 경우, 수집된 통계는 메모리 내부만이며, 일반적으로 샘플링을 사용하여 수집됩니다(소형 테이블의 경우 제외).

실시간 통계 처리는 정적 및 동적 SQL에 대해 모두 발생합니다.

IMPORT 명령을 사용하여 잘린 테이블은 시일이 지난 통계를 지닌 것으로 자동으로 인식됩니다.

자동 통계 콜렉션(동기 및 비동기 모두)은 통계가 수집된 대상 테이블을 참조하는 캐시된 동적문을 무효화합니다. 이는 캐시된 동적문을 최신 통계를 사용하여 다시 최적화할 수 있도록 하기 위해 수행됩니다.

실시간 통계 및 Explain 처리

Explain 기능을 사용하여 Explain만 되는(실행되지 않고) 쿼리에는 실시간 처리가 없습니다. 다음 표에는 다양한 값의 CURRENT EXPLAIN MODE 특수 레지스터에 따른 동작이 요약되어 있습니다.

표 6. CURRENT EXPLAIN MODE 특수 레지스터 값의 작용에 따른 실시간 통계 콜렉션

CURRENT EXPLAIN MODE 값	실시간 통계 콜렉션 고려
YES	예
EXPLAIN	아니오
NO	예
REOPT	예
RECOMMEND INDEXES	아니오
EVALUATE INDEXES	아니오

자동 통계 콜렉션 및 통계 캐시

통계 캐시는 모든 쿼리에 동기적으로 수집된 통계를 사용 가능하게 하기 위해 DB2 버전 9.5에서 도입되었습니다. 이 캐시는 카탈로그 캐시의 일부입니다. 파티션된 데이터베이스 환경에서, 이 캐시는 카탈로그 데이터베이스 파티션에만 상주합니다. 카탈로그 캐시는 동일한 SYSTABLES 오브젝트에 대한 다중 항목을 저장할 수 있고, 이는 모든 데이터베이스 파티션에서 카탈로그 캐시의 *크기*를 늘립니다. 실시간 통계 콜렉션이 사용되는 경우 **catalogcache_sz** 데이터베이스 구성 매개변수 값을 증가시키는 것을 고려하십시오.

DB2 버전 9부터 구성 어드바이저를 사용하여 새 데이터베이스에 대한 초기 구성을 판별할 수 있습니다. 구성 어드바이저는 **auto_stmt_stats** 데이터베이스 구성 매개변수를 설정(ON)으로 설정하도록 권장합니다.

자동 통계 콜렉션 및 통계 프로파일

동기 및 비동기 통계는 테이블에 적용되는 통계 프로파일에 따라 수집되며, 다음 예외를 지닙니다.

- 동기 통계 콜렉션의 오버헤드를 최소화하기 위해 데이터베이스 관리 프로그램에서 샘플링을 사용하여 통계를 수집할 수도 있습니다. 이 경우, 샘플링 비율과 방법은 통계 프로파일에 지정된 비율 및 방법과 다를 수 있습니다.
- 동기 통계 콜렉션에서 통계를 제작하도록 선택할 수 있지만, 통계 프로파일에 지정된 모든 통계를 제작할 수 없을 수도 있습니다. 예를 들어, 일부 인덱스에서 컬럼이 선행되지 않는 한 COLCARD, HIGH2KEY 및 LOW2KEY와 같은 컬럼 통계를 제작할 수 없습니다.

동기 통계 콜렉션에서 통계 프로파일에 지정된 모든 통계를 수집할 수 없는 경우, 비동기 콜렉션 요청이 제출됩니다.

실시간 통계 콜렉션이 통계 콜렉션 오버헤드를 최소화하도록 설계되었다 하더라도, 우선 테스트 환경에서 시도하여 부정적인 성능 영향이 없는지 확인하십시오. 일부 온라인 트랜잭션 처리(OLTP) 시나리오에서, 특히 쿼리가 실행될 수 있는 기간에 대한 상한이 있는 경우 이렇게 하십시오.

자동 통계 콜렉션 사용

정확하고 완전한 데이터베이스 통계를 갖는 것은 효율적인 데이터 액세스와 최적의 워크로드 성능에 매우 중요합니다. 자동 테이블 유지보수 기능의 자동 통계 콜렉션 기능을 사용하여 관련 데이터베이스 통계를 갱신하고 유지보수하십시오.

DB2 서버에서 워크로드에 필요한 정확한 통계 세트를 자동으로 수집하는 데 도움이 되는 통계 프로파일을 생성하고 쿼리 데이터를 수집함으로써 단일 데이터베이스 파티션이 단일 프로세서에서 작동하는 환경에서 이 기능을 향상시킬 수 있습니다. 이 옵션은 파티션된 데이터베이스 환경, 특정 페더레이티드 데이터베이스 환경 또는 파티션 내 병렬처리가 사용되는 환경에서는 사용 가능하지 않습니다.

자동 통계 콜렉션을 사용하려면 다음을 수행하십시오.

1. **auto_maint**, **auto_tbl_maint** 및 **auto_runstats** 데이터베이스 구성 매개변수를 설정(ON)으로 설정하여 데이터베이스 인스턴스를 구성하십시오. **auto_maint** 매개변수는 **auto_tbl_maint** 및 **auto_runstats**의 상위입니다.
2. 선택사항: 자동 통계 프로파일 생성을 사용하려면 **auto_stats_prof** 및 **auto_prof_upd** 데이터베이스 구성 매개변수를 설정(ON)으로 설정하십시오. **auto_maint** 매개변수는 **auto_stats_prof** 및 **auto_prof_upd**의 상위입니다.
3. 선택사항: 실시간 통계 수집을 사용하려면 **auto_stmt_stats** 구성 매개변수를 설정(ON)으로 설정하십시오. 쿼리를 최적화해야 할 때마다 명령문 컴파일 시간에 테이블 통계가 자동으로 수집됩니다. **auto_maint** 매개변수는 **auto_stmt_stats**의 상위입니다.

구성 어드바이저

구성 어드바이저를 사용하면 버퍼 풀 크기, 데이터베이스 구성 매개변수 및 데이터베이스 관리 프로그램 구성 매개변수의 초기 값에 대한 권장사항을 알 수 있습니다.

구성 어드바이저를 사용하려면 기존 데이터베이스의 경우 AUTOCONFIGURE 명령을 지정하거나 CREATE DATABASE 명령에서 옵션으로 AUTOCONFIGURE를 지정하십시오. 데이터베이스를 구성하려면 SYSADM, SYSCTRL 또는 SYSMAINT 권한이 있어야 합니다.

권장 값을 표시하거나 CREATE DATABASE 명령에서 APPLY 옵션을 사용하여 권장 값을 적용할 수 있습니다. 권장사항은 사용자가 제공하는 입력 및 어드바이저가 수집하는 시스템 정보에 따라 다릅니다.

구성 어드바이저에서 제안하는 값은 인스턴스당 한 개의 데이터베이스에만 관련이 있습니다. 둘 이상의 데이터베이스에서 해당 어드바이저를 사용하려면 각 데이터베이스가 개별 인스턴스에 속해야 합니다.

구성 어드바이저를 사용하여 구성 매개변수 조정

구성 어드바이저는 수정할 구성 매개변수를 제안하고 해당 매개변수에 대해 값을 제안하여 성능을 조정하고 인스턴스당 단일 데이터베이스에 대한 메모리 요구사항의 균형을 맞추도록 지원합니다. 구성 어드바이저는 데이터베이스 작성 시 자동으로 실행됩니다.

이 기능을 사용하지 않으려거나 명시적으로 사용하려면 데이터베이스를 작성하기 전에 다음과 같이 db2set 명령을 사용합니다.

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

여러 구성 매개변수의 값을 정의하고 이러한 매개변수의 적용 범위를 판별하려면 AUTOCONFIGURE 명령을 사용하여 다음 옵션 중 하나를 지정하십시오.

- NONE, 적용되는 값이 없음
- DB ONLY, 데이터베이스 구성 및 버퍼 풀 값만 적용됨
- DB AND DBM, 모든 매개변수 및 해당 값이 적용됨

주: CREATE DATABASE 명령을 실행했을 때 구성 어드바이저를 사용 가능하도록 자동으로 설정한 경우 AUTOCONFIGURE 명령을 지정할 수 있습니다. CREATE DATABASE 명령을 실행했을 때 구성 어드바이저를 사용 가능하도록 설정하지 않은 경우 이후에 구성 어드바이저를 수동으로 실행할 수 있습니다.

데이터베이스 구성 권장사항 생성

구성 어드바이저는 데이터베이스 작성 시 자동으로 실행됩니다. 명령행 처리기(CLP)에서 AUTOCONFIGURE 명령을 지정하거나 db2AutoConfig API를 호출하여 구성 어드바이저를 실행할 수도 있습니다.

CLP를 사용하여 구성 권장사항을 요청하려면 다음 명령을 입력하십시오.

```
 AUTOCONFIGURE
USING input_keyword param_value
APPLY value
```

다음은 데이터베이스가 사용되는 방법에 대한 입력을 기준으로 구성 권장사항을 요청하지만 해당 권장사항이 적용되지 않도록 지정하는 AUTOCONFIGURE 명령의 예입니다.

```
DB2 AUTOCONFIGURE USING
     MEM_PERCENT 60
     WORKLOAD_TYPE MIXED
     NUM_STMTS 500
     ADMIN_PRIORITY BOTH
     IS_POPULATED YES
     NUM_LOCAL_APPS 0
```

```
     NUM_REMOTE_APPS 20
     ISOLATION RR
     BP_RESIZEABLE YES
APPLY NONE
```

예: 구성 어드바이저를 사용하여 구성 권장사항 요청

이 시나리오에서는 권장사항을 생성하도록 명령행에서 구성 어드바이저를 실행하는 방법에 대해 설명하고 구성 어드바이저에서 생성되는 출력을 보여줍니다.

구성 어드바이저를 실행하려면 다음을 수행하십시오.

1. 명령행에서 다음 명령을 지정하여 PERSONL 데이터베이스에 연결합니다.

   ```
   DB2 CONNECT TO PERSONL
   ```

2. CLP에서 AUTOCONFIGURE 명령을 발행하여 데이터베이스가 사용되는 방법을 지정합니다. 다음 예에서처럼 **APPLY** 옵션에 대해 NONE 값을 설정하여 구성 권장사항을 보려고 하지만 적용하지는 않을 것임을 나타냅니다.

   ```
   DB2 AUTOCONFIGURE USING
        MEM_PERCENT 60
        WORKLOAD_TYPE MIXED
        NUM_STMTS 500
        ADMIN_PRIORITY BOTH
        IS_POPULATED YES
        NUM_LOCAL_APPS 0
        NUM_REMOTE_APPS 20
        ISOLATION RR
        BP_RESIZEABLE YES
   APPLY NONE
   ```

   명령에 대한 매개변수 값이 확실하지 않은 경우 해당 값을 생략하면 디폴트값이 사용됩니다. 이전 예에서처럼 MEM_PERCENT, WORKLOAD_TYPE 등의 값을 사용하지 않고 최대 10개의 매개변수를 전달할 수 있습니다.

AUTOCONFIGURE 명령이 생성한 권장사항은 화면에 61 페이지의 그림 2에 표시된 테이블 형식으로 표시됩니다.

```
      데이터베이스 관리 프로그램 구성의 이전 값 및 적용된 값
설명                                          매개변수     현재값                  권장값
-------------------------------------------------------------------------------------------
응용프로그램 지원 계층 힙 크기(4KB)          (ASLHEAPSZ)   = 15                    15
int. 통신 버퍼 수(4KB)                     (FCM_NUM_BUFFERS) = AUTOMATIC            AUTOMATIC
파티션내 병렬 처리 사용                     (INTRA_PARALLEL)  = NO                   NO
병렬 처리 등급의 최대 쿼리                  (MAX_QUERYDEGREE) = ANY                  1
에이전트 풀 크기                            (NUM_POOLAGENTS)  = 100(계산됨)          200
풀에 있는 초기 에이전트 수                  (NUM_INITAGENTS)  = 0                    0
리퀘스터 I/O 블록의 최대 크기(바이트)       (RQRIOBLK)        = 32767                32767
정렬 힙 임계값(4KB)                         (SHEAPTHRES)      = 0                    0

      데이터베이스 구성의 이전 값 및 적용된 값
설명                                          매개변수     현재값                  권장값
-------------------------------------------------------------------------------------------
디폴트 응용프로그램 힙(4KB)                   (APPLHEAPSZ)  = 256                  256
카탈로그 캐시 크기 (4KB)                    (CATALOGCACHE_SZ) = (MAXAPPLS*4)        260
변경된 페이지 임계값                         (CHNGPGS_THRESH) = 60                  80
데이터베이스 힙 (4KB)                              (DBHEAP) = 1200                  2791
병렬 처리 등급                                (DFT_DEGREE) = 1                      1
디폴트 테이블 스페이스 Extent 크기(페이지) (DFT_EXTENT_SZ)= 32                      32
디폴트 프리페치 크기(페이지)                (DFT_PREFETCH_SZ) = AUTOMATIC           AUTOMATIC
디폴트 쿼리 최적화 클래스                     (DFT_QUERYOPT) = 5                    5
잠금 목록용 최대 스토리지 (4KB)                (LOCKLIST) = 100                     AUTOMATIC
로그 버퍼 크기 (4KB)                          (LOGBUFSZ) = 8                       99
로그 파일 크기 (4KB)                          (LOGFILSIZ) = 1000                   1024
1차 로그 파일 수                              (LOGPRIMARY) = 3                     8
2차 로그 파일 수                               (LOGSECOND) = 2                     3
실행 중인 최대 프로그램 수                     (MAXAPPLS) = AUTOMATIC              AUTOMATIC
응용프로그램당 잠금 목록의 백분율              (MAXLOCKS) = 10                     AUTOMATIC
그룹 커미트 계수                              (MINCOMMIT) = 1                      1
비동기 페이지 클리너 수                      (NUM_IOCLEANERS) = 1                  1
I/O 서버 수                                   (NUM_IOSERVERS) = 3                  4
패키지 캐시 크기 (4KB)                        (PCKCACHESZ) = (MAXAPPLS*8)          1533
소프트 체크포인트 전에 수정된 로그 파일의 백분율 (SOFTMAX) = 100                    320
정렬 목록 힙 (4KB)                                 (SORTHEAP) = 256                AUTOMATIC
명령문 힙 (4KB)                                  (STMTHEAP) = 4096                 4096
통계 힙 크기 (4KB)                            (STAT_HEAP_SZ) = 4384                4384
유틸리티 힙 크기 (4KB)                        (UTIL_HEAP_SZ) = 5000                113661
자체 조정 메모리                              (SELF_TUNING_MEM) = ON                ON
 자동 runstats                                (AUTO_RUNSTATS) = ON                 ON
공유 정렬의 정렬 힙 임계값 (4KB)              (SHEAPTHRES_SHR) = 5000              AUTOMATIC

      버퍼 풀의 이전 값 및 적용된 값
설명                                          매개변수     현재값                  권장값
-------------------------------------------------------------------------------------------
IBMDEFAULTBP                                 버퍼 풀 크기 = -2                    340985

DB210203I AUTOCONFIGURE가 완료되었습니다. 변경사항을 적용하기로 선택한 경우
데이터베이스 관리 프로그램 또는 데이터베이스 구성 값이 변경될 수 있습니다. 변경사항을 적용하려면
인스턴스를 재시작해야 합니다. 새 값이 사용되도록 새 구성 매개변수가 적용된 후에
패키지를 리바인드할 수도 있습니다.
```

그림 2. 구성 어드바이저 샘플 출력

모든 권장사항에 동의하는 경우 AUTOCONFIGURE 명령을 다시 발행하고 APPLY 옵션을 사용하여 권장 값이 적용되도록 지정하거나 UPDATE DATABASE MANAGER CONFIGURATION 명령 및 UPDATE DATABASE CONFIGURATION 명령을 사용하여 개별 구성 매개변수를 갱신합니다.

유틸리티 조절 기능

유틸리티 조절 기능은 프로덕션 기간에 유지보수 유틸리티를 동시에 실행할 수 있도록 유지보수 유틸리티의 성능 영향을 규제합니다. 영향 규정(유틸리티를 조정 모드에서 실행할 수 있도록 하는 설정)이 디폴트로 정의되어 있지만, 영향을 조정하려면 유틸리티 실행 시 영향 우선순위(각 클리너가 조절 우선순위를 표시하는 설정)를 설정해야 합니다.

조정 시스템은 영향 규정을 위반하지 않고 조정 유틸리티를 가능한 한 자주 실행할 수 있도록 합니다. 통계 콜렉션, 백업 작업, 재조정 작업 및 비동기 인덱스 정리를 조정할 수 있습니다.

util_impact_lim 구성 매개변수를 설정하여 영향 규정을 정의합니다.

클리너는 유틸리티 조절 기능과 통합됩니다. 디폴트로, 각 (인덱스) 클리너는 50의 유틸리티 영향 우선순위를 갖습니다(승인할 수 있는 값은 1 - 100이며 0은 조절하지 않음을 표시). SET UTIL_IMPACT_PRIORITY 명령 또는 db2UtilityControl API를 사용하여 우선순위를 변경할 수 있습니다.

비동기 인덱스 정리

비동기 인덱스 정리(AIC)는 인덱스 항목을 무효화하는 조작에 따른 인덱스의 지연된 정리입니다. 인덱스의 유형에 따라, 항목이 레코드 ID(RID) 또는 블록 ID (BID)가 될 수 있습니다. 유효하지 않은 인덱스 항목은 백그라운드에서 비동기적으로 작동하는 인덱스 클리너에 의해 제거됩니다.

AIC는 파티션된 테이블에서의 데이터 파티션 접속 해제를 촉진하고, 파티션된 테이블에 하나 이상의 파티션되지 않은 인덱스가 포함되어 있는 경우 시작됩니다. 이 경우, AIC는 접속 해제된 데이터 파티션을 참조하는 모든 파티션되지 않은 인덱스 항목과 의사 삭제된 항목을 제거합니다. 인덱스가 모두 정리된 후, 접속 해제된 데이터 파티션과 연관된 ID가 시스템 카탈로그에서 제거됩니다.

파티션된 테이블에 종속 구체화된 쿼리 테이블(MQT)이 있는 경우, SET INTEGRITY 문이 실행될 때까지 AIC가 시작되지 않습니다.

AIC가 진행되는 동안 일반 테이블 액세스가 유지됩니다. 인덱스에 액세스하는 쿼리는 아직 정리되지 않은 유효하지 않은 항목을 무시합니다.

대부분의 경우, 파티션된 테이블과 연관된 각 파티션되지 않은 인덱스에 대해 하나의 클리너가 시작됩니다. 내부 태스크 분산 디먼이 해당 테이블 파티션에 AIC 태스크를 분산하고 데이터베이스 에이전트를 지정하는 일을 담당합니다. 분산 디먼과 클리너 에이전트는 LIST APPLICATIONS 명령 출력에서 각각 응용프로그램 이름 db2taskd와 db2aic로 표시되는 내부 시스템 응용프로그램입니다. 우발적인 중단을 예방하기 위해 시스템 응용프로그램을 강제 실행할 수 없습니다. 분산 디먼은 데이터베이스가 사용 중인 한 온라인으로 유지됩니다. 클리너는 정리가 완료될 때까지 활성으로 유지됩니다. 정리가 진행 중인 동안 데이터베이스가 비활성화될 경우 데이터베이스를 다시 활성화할 때 AIC가 재시작됩니다.

성능에 대한 AIC 영향

AIC는 최소 성능 영향을 미칩니다.

의사 삭제된 항목이 커미트되었는지 여부를 판별하기 위해 즉각적인 행 잠금 테스트가 필요합니다. 그러나 잠금이 획득되지 않으므로 동시성이 영향을 받지 않습니다.

각 클리너는 최소 테이블 스페이스 잠금(IX) 및 테이블 잠금 (IS)을 획득합니다. 이러한 잠금은 클리너에서 다른 응용프로그램이 잠금을 대기하고 있음을 판별할 경우 해제됩니다. 이 경우 클리너가 5분간 처리를 일시중단합니다.

클리너는 유틸리티 조절 기능과 통합됩니다. 디폴트로, 각 클리너는 50의 유틸리티 영향 우선순위를 갖습니다. SET UTIL_IMPACT_PRIORITY 명령 또는 db2UtilityControl API를 사용하여 우선순위를 변경할 수 있습니다.

AIC 모니터링

LIST UTILITIES 명령을 사용하여 AIC를 모니터할 수 있습니다. 각 인덱스 클리너는 출력에서 개별 유틸리티로 표시됩니다. 다음은 LIST UTILITIES SHOW DETAIL 명령의 출력 예입니다.

```
ID                                   = 2
유형                                 = ASYNCHRONOUS INDEX CLEANUP
데이터베이스 이름                    = WSDB
파티션 번호                          = 0
설명                                 = 테이블: USER1.SALES, 인덱스: USER1.I2
시작 시간                            = 12/15/2005 11:15:01.967939
상태                                 = 실행 중
호출 유형                            = 자동
조절 기능:
   우선순위                          = 50
진행 모니터링:
      전체 작업                      = 5페이지
      완료 작업                      = 0페이지
      시작 시간                      = 12/15/2005 11:15:01.979033

ID                                   = 1
유형                                 = ASYNCHRONOUS INDEX CLEANUP
데이터베이스 이름                    = WSDB
```

```
파티션 번호                     = 0
설명                            = 테이블: USER1.SALES, 인덱스: USER1.I1
시작 시간                       = 12/15/2005 11:15:01.978554
상태                            = 실행 중
호출 유형                       = 자동
조절 기능:
   우선순위                     = 50
진행 모니터링:
      전체 작업                 = 5페이지
      완료 작업                 = 0페이지
      시작 시간                 = 12/15/2005 11:15:01.980524
```

이 경우, USERS1.SALES 테이블에서 2개의 클리너가 작동하고 있습니다. 하나의 클리너는 인덱스 11을 처리 중이고, 다른 클리너는 인덱스 12를 처리 중입니다. 진행 모니터링 세션에서 정리가 필요한 인덱스 페이지의 총 추정 수와 현재 인덱스 페이지 정리 수를 보여줍니다.

상태 필드는 클리너의 현재 상태를 표시합니다. 일반 상태는 실행이지만, 잠금 경합 때문에 클리너가 일시중단된 경우 또는 사용 가능한 데이터베이스 에이전트에 지정되기를 대기 중인 경우 클리너가 대기 상태가 될 수도 있습니다.

각 데이터베이스 파티션은 해당 데이터베이스 파티션에서만 실행 중인 태스크에 ID를 지정하기 때문에 다른 데이터베이스 파티션의 다른 태스크가 동일한 유틸리티 ID를 가질 수 있습니다.

MDC 테이블에 대한 비동기 인덱스 정리

비동기 인덱스 정리(AIC)를 사용하여 롤아웃 삭제(다차원적으로 클러스터된(MDC) 테이블에서 데이터의 규정 블록을 삭제하기 위한 효과적인 방법)의 성능을 향상시킬 수 있습니다. AIC는 인덱스 항목을 무효화하는 조작에 따른 인덱스의 지연된 정리입니다.

표준 롤아웃 삭제 동안 인덱스가 비동기적으로 정리됩니다. 테이블에 많은 레코드 ID(RID) 인덱스가 포함된 경우, 삭제되는 테이블 행을 참조하는 인덱스 키를 제거하는 데 상당한 시간이 소요됩니다. 삭제 조작 커미트 후에 이러한 인덱스가 정리되도록 지정하여 롤아웃의 속도를 높일 수 있습니다.

MDC 테이블에 대해 AIC를 이용하려면, *지연된 인덱스 정리 롤아웃* 메커니즘을 명시적으로 사용해야 합니다. 지연된 롤아웃을 지정하는 데에는 **DB2_MDC_ROLLOUT** 레지스트리 변수를 DEFER로 지정하거나 SET CURRENT MDC ROLLOUT MODE 문을 발행하는 두 가지 방법이 있습니다. 지연된 인덱스 정리 롤아웃 조작 동안 트랜잭션 커미트 후까지 RID 인덱스에 대한 갱신 없이 블록이 롤아웃된 것으로 표시됩니다. 행 레벨 처리가 필요하지 않기 때문에 삭제 조작 동안 블록 ID(BID) 인덱스가 정리됩니다.

롤아웃 삭제가 커미트될 때, 또는 데이터베이스가 종료된 경우 데이터베이스 재시작 후 테이블에 첫 번째로 액세스할 때 AIC 롤아웃이 호출됩니다. AIC가 진행 중인 동안에는 정리되는 인덱스에 액세스하는 쿼리를 포함하여 인덱스에 대한 쿼리가 성공합니다.

MDC 테이블당 하나의 코디네이팅 클리너가 있습니다. 다중 롤아웃에 대한 인덱스 정리는 각 RID 인덱스에 대해 정리 에이전트를 제공하는 클리너 내에서 통합됩니다. 정리 에이전트는 RID 인덱스를 병렬로 갱신합니다. 클리너는 유틸리티 조절 기능과도 통합됩니다. 디폴트로, 각 클리너는 50의 유틸리티 영향 우선순위를 갖습니다(허용 가능한 값은 1-100이며, 0은 조절 기능이 아님을 표시함). SET UTIL_IMPACT_PRIORITY 명령 또는 db2UtilityControl API를 사용하여 이 우선순위를 변경할 수 있습니다.

지연된 인덱스 정리 롤아웃 조작의 진행 모니터링

MDC 테이블에서 롤아웃된 블록은 정리가 완료될 때까지 다시 사용할 수 없기 때문에 지연된 인덱스 정리 롤아웃 조작의 진행을 모니터하는 것이 유용합니다. LIST UTILITIES 명령을 사용하여 정리되는 각 인덱스에 대한 유틸리티 모니터 항목을 표시하십시오. SYSPROC.ADMIN_GET_TAB_INFO_V95 테이블 함수 또는 GET SNAPSHOT 명령을 사용하여 롤아웃 삭제 (BLOCKS_PENDING_CLEANUP)에 따라 비동기 정리를 보류 중인 데이터베이스의 MDC 테이블 블록 총 수를 검색할 수도 있습니다.

다음 LIST UTILITIES SHOW DETAILS 명령의 샘플 출력에서는 정리된 각 인덱스의 페이지 수에 따른 진행이 표시됩니다. 각 단계는 하나의 RID 인덱스를 나타냅니다.

```
ID                                     = 2
유형                                   = MDC ROLLOUT INDEX CLEANUP
데이터베이스 이름                      = WSDB
파티션 번호                            = 0
설명                                   = TABLE.<schema_name>.<table_name>
시작 시간                              = 06/12/2006 08:56:33.390158
상태                                   = 실행 중
호출 유형                              = 자동
조절 기능:
   우선순위                            = 50
진행 모니터링:
   예상된 백분율 완료                  = 83
   단계 번호                           = 1
      설명                             = <schema_name>.<index_name>
      전체 작업                        = 13페이지
      완료 작업                        = 13페이지
      시작 시간                        = 06/12/2006 08:56:33.391566
   단계 번호                           = 2
      설명                             = <schema_name>.<index_name>
      전체 작업                        = 13페이지
      완료 작업                        = 13페이지
      시작 시간                        = 06/12/2006 08:56:33.391577
   단계 번호                           = 3
      설명                             = <schema_name>.<index_name>
      전체 작업                        = 9페이지
      완료 작업                        = 3페이지
      시작 시간                        = 06/12/2006 08:56:33.391587
```

제 4 장 인스턴스

인스턴스는 논리적 데이터베이스 관리 프로그램 환경으로 여기서 데이터베이스를 카탈로그화하고 구성 매개변수를 설정합니다. 사용자의 필요에 따라 동일한 실제 서버에서 둘 이상의 인스턴스를 작성하여 각 인스턴스에 고유한 데이터베이스 서버 환경을 제공할 수 있습니다.

주: Linux 및 UNIX 운영 체제에서 루트 서버가 아닌 설치의 경우 DB2 제품 설치 시 단일 인스턴스가 작성됩니다. 추가적인 인스턴스를 작성할 수 없습니다.

다중 인스턴스를 사용하여 다음을 수행할 수 있습니다.

- 개발 환경에 하나의 인스턴스를 사용하고 프로덕션 환경에 다른 인스턴스를 사용합니다.
- 특정 환경에 적합하게 인스턴스를 조정합니다.
- 중요한 정보에 대한 액세스를 제한합니다.
- 각 인스턴스에 대한 SYSADM, SYSCTRL 및 SYSMAINT 권한 지정을 제어합니다.
- 각 인스턴스의 데이터베이스 관리 프로그램 구성을 최적화합니다.
- 인스턴스 실패 영향을 제한합니다. 인스턴스가 실패하는 경우 하나의 인스턴스만 영향을 받습니다. 기타 인스턴스는 계속해서 정상적으로 작동합니다.

다중 인스턴스에는 다음 항목이 필요합니다.

- 각 인스턴스에서 사용할 추가적인 시스템 자원(가상 메모리 및 디스크 스페이스).
- 관리할 추가적인 인스턴스로 인한 추가 관리.

인스턴스 디렉토리에는 데이터베이스 인스턴스와 관련된 모든 정보가 저장됩니다. 인스턴스 디렉토리를 작성한 후에는 위치를 변경할 수 없습니다. 디렉토리에는 다음 내용이 포함됩니다.

- 데이터베이스 관리 프로그램 구성 파일
- 시스템 데이터베이스 디렉토리
- 노드 디렉토리
- 노드 구성 파일(db2nodes.cfg)
- DB2 데이터베이스 프로세스에 필요한 호출 스택이나 레지스터 덤프 또는 예외와 같은 디버깅 정보가 들어 있는 기타 파일

용어:

비트 폭

가상 메모리를 가리키는 데 사용되는 비트 수입니다. 32비트 및 64비트가 가장 많이 사용됩니다. 이 용어는 인스턴스, 응용프로그램 코드, 외부 루틴 코드의 비트 폭을 가리키는 데 사용됩니다. 32비트 응용프로그램은 32비트 폭 응용프로그램과 동일한 의미입니다.

32비트 DB2 인스턴스

32비트 공유 라이브러리 및 실행 파일을 포함하여 모든 32비트 실행 파일이 포함된 DB2 인스턴스입니다.

64비트 DB2 인스턴스

64비트 공유 라이브러리와 실행 파일, 모든 32비트 클라이언트 응용프로그램 라이브러리(클라이언트와 서버 둘 다에 포함) 및 32비트 외부 루틴 지원(서버 인스턴스에만 포함)이 포함된 DB2 인스턴스입니다.

인스턴스 설계

DB2 데이터베이스는 데이터베이스 서버에 있는 DB2 인스턴스에 작성됩니다. 동일한 실제 서버에 여러 인스턴스를 작성하면 인스턴스마다 고유한 데이터베이스 서버 환경을 설정할 수 있습니다.

예를 들면, 동일한 컴퓨터에서 테스트 환경 및 프로덕션 환경을 유지보수하거나, 각 응용프로그램에 인스턴스를 작성한 다음 인스턴스가 사용되는 응용프로그램에 적합하도록 각 인스턴스를 정밀하게 조정하거나, 중요한 데이터를 보호하기 위해 동일한 서버에 있는 다른 인스턴스 소유자가 급여 데이터를 볼 수 없도록 사용자 고유 인스턴스에 급여 데이터베이스를 저장할 수 있습니다.

설치 프로세스에서 디폴트 DB2 인스턴스를 작성하며 이 인스턴스는 DB2INSTANCE 환경 변수로 정의됩니다. 이 인스턴스는 대부분의 조작에서 사용되는 인스턴스입니다. 그러나 설치 후에 인스턴스를 작성하거나 삭제할 수도 있습니다.

사용자 환경에 적합하게 인스턴스를 판별하고 설계할 경우 각 인스턴스가 하나 이상의 데이터베이스에 대한 액세스를 제어하는 사실에 유의하십시오. 인스턴스 내의 모든 데이터베이스에는 고유 이름이 지정되고 고유한 시스템 카탈로그 테이블 세트가 있으며(데이터베이스에 작성되는 오브젝트 트랙을 유지하는 데 사용됨) 고유한 구성 파일이 있습니다. 모든 데이터베이스에는 사용자가 데이터베이스에 저장된 데이터 및 데이터베이스 오브젝트와 상호 작용하는 방법을 제어하는 부여 가능한 고유 권한 및 특권 세트도 있습니다. 69 페이지의 그림 3에는 시스템, 인스턴스 및 데이터베이스 간의 계층 관계가 표시되어 있습니다.

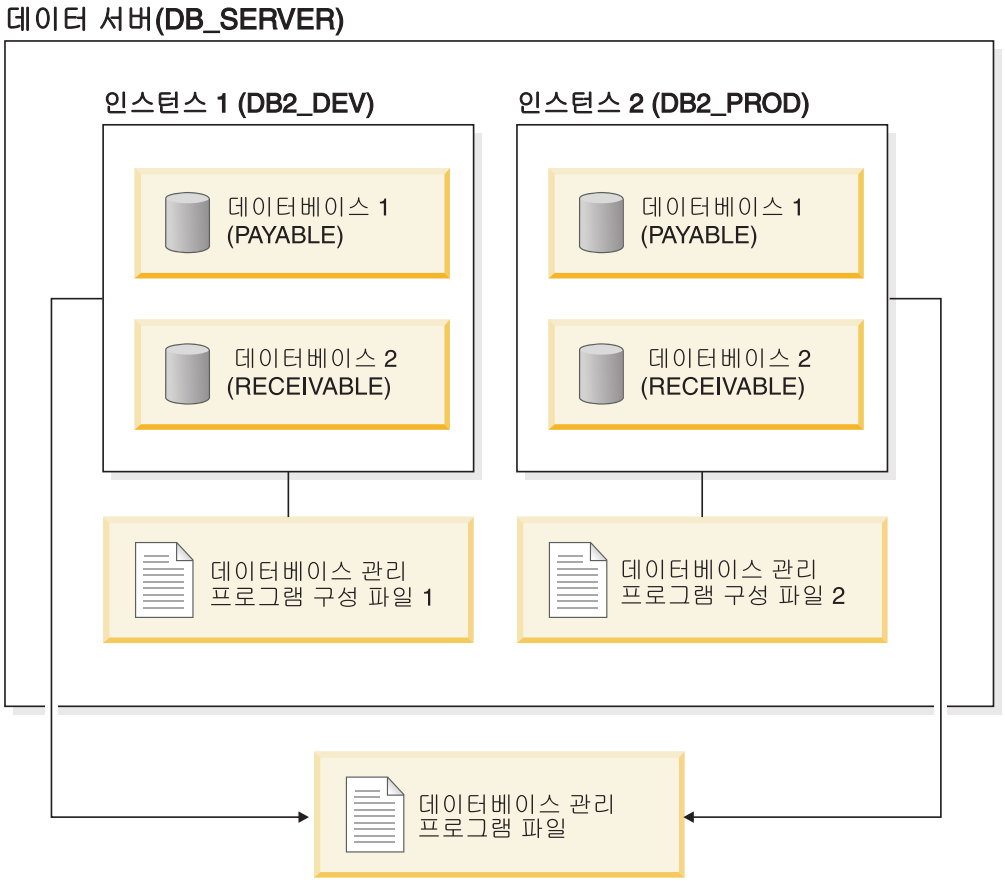


그림 3. DB2 시스템, 인스턴스 및 데이터베이스 간 계층 관계

*DAS(DB2 Administration Server)*라고 하는 특정 인스턴스 유형에 대해서도 유의해야 합니다. DAS는 다른 DB2 서버에서만 관리 태스크를 보조하는 데 사용되는 특수 DB2 관리 제어점입니다. 클라이언트 구성 지원 프로그램을 사용하여 리모트 데이터베이스 또는 DB2 제품과 함께 제공되는 그래픽 도구(예: 제어 센터 또는 태스크 센터)를 찾으려면 DAS를 실행해야 합니다. 다중 인스턴스가 있는 경우에도 DB2 데이터베이스 서버에는 하나의 DAS만 있습니다.

사용자의 인스턴스가 작성되면 다른 시스템의 인스턴스를 포함하여 사용 가능 한 기타 인스턴스에 접속할 수 있습니다. 접속한 후에는 인스턴스 레벨에서만 실행할 수 있는 유지보수 및 유틸리티 태스크를 수행할 수 있습니다. 예를 들면, 데이터베이스 작성, 데이터베이스에서 응용프로그램 강제 해제, 데이터베이스 활동 모니터 또는 해당 특정 인스턴스와 연관된 데이터베이스 관리 프로그램 구성 파일의 내용 변경을 수행할 수 있습니다.

디폴트 인스턴스

DB2 설치 작업의 파트로서 『DB2』라는 다른 인스턴스가 없는 경우에는 DB2라고 하는 데이터베이스 관리 프로그램의 초기 인스턴스를 작성할 수 있습니다. DB2 버전 8이 설치되어 있는 경우에는 버전 9.1 또는 버전 9.5로 업그레이드할 수 있으며 디폴트 인스턴스는 『DB2_01』입니다.

Linux 및 UNIX에서는 이름 지정 규칙을 준수하는 한 초기 인스턴스에 원하는 이름을 지정할 수 있습니다. 인스턴스 이름은 디렉토리 구조를 설정하는 데 사용됩니다.

이 인스턴스를 즉시 사용할 수 있도록 하기 위해 설치 시 다음을 설정합니다.

- 환경 변수 DB2INSTANCE를 『DB2』로 설정합니다.
- 레지스트리 변수 DB2INSTDEF를 『DB2』로 설정합니다.

이렇게 지정하면 『DB2』가 디폴트 인스턴스로 설정됩니다. 디폴트로 사용되는 인스턴스를 변경할 수 있으나 먼저 추가 인스턴스를 작성해야 합니다.

데이터베이스 관리 프로그램을 사용하기 전에, 인스턴스에 액세스하고 DB2 데이터베이스 프로그램을 실행할 수 있도록 각 사용자의 데이터베이스 환경을 갱신해야 합니다. 이는 관리 사용자를 포함한 모든 사용자에게 적용됩니다.

Linux 및 UNIX 운영 체제에서는 데이터베이스 환경 설정에 유용하도록 샘플 스크립트 파일이 제공됩니다. 이 파일은 Bourne 또는 Korn 쉘용 db2profile 및 C 쉘용 db2cshrc입니다. 샘플 스크립트는 인스턴스 소유자의 홈 디렉토리 아래 있는 sqllib 서브디렉토리에 있습니다. 인스턴스 소유자 또는 인스턴스의 SYSADM 그룹에 속하는 모든 사용자는 인스턴스의 모든 사용자에게 적합하게 스크립트를 사용자 정의할 수 있습니다. 각 사용자에 적합하게 스크립트를 사용자 정의하려면 sqllib/userprofile 및 sqllib/usercshrc를 사용하십시오.

인스턴스 작성 시 공백 파일 sqllib/userprofile과 sqllib/usercshrc가 작성되어 사용자 고유의 인스턴스 환경 설정을 추가할 수 있도록 합니다. db2profile 및 db2cshrc 파일은 DB2 FixPak 설치에서 인스턴스를 갱신하는 동안 겹쳐써집니다. db2profile 또는 db2cshrc 스크립트에서 새 환경 설정을 사용하지 않으려면 해당 *사용자 스크립트*를 사용하여 이들 파일을 겹쳐쓸 수 있습니다. 사용자 스크립트는 db2profile 또는 db2cshrc 스크립트의 마지막에 호출됩니다. db2iupgrade 명령을 사용하여 인스턴스를 업그레이드하는 동안 환경 수정사항을 계속 사용할 수 있도록 사용자 스크립트가 복사됩니다.

샘플 스크립트에는 다음을 수행하는 명령문이 포함되어 있습니다.

- 기존 검색 경로에 인스턴스 소유자 홈 디렉토리의 sqllib 서브디렉토리에 있는 bin, adm 및 misc 서브디렉토리를 추가하여 사용자의 PATH 갱신합니다.
- DB2INSTANCE 환경 변수를 인스턴스 이름으로 설정합니다.

인스턴스 디렉토리

인스턴스 디렉토리에는 데이터베이스 인스턴스와 관련된 모든 정보가 저장됩니다. 인스턴스 디렉토리를 작성한 후에는 디렉토리 위치를 변경할 수 없습니다.

인스턴스 디렉토리에는 다음 항목이 포함됩니다.

- 데이터베이스 관리 프로그램 구성 파일
- 시스템 데이터베이스 디렉토리
- 노드 디렉토리
- 노드 구성 파일(db2nodes.cfg)
- DB2 프로세스에 필요한 호출 스택이나 레지스터 덤프 또는 예외와 같은 디버깅 정보가 들어 있는 기타 파일

Linux 및 UNIX 운영 체제에서는 인스턴스 디렉토리가 INSTHOME/sqllib 디렉토리에 있습니다. 여기서 INSTHOME은 인스턴스 소유자의 홈 디렉토리입니다. 이름 지정 규칙을 준수하는 한 디폴트 인스턴스 이름을 원하는 대로 지정할 수 있습니다.

Windows 운영 체제에서는 인스턴스 디렉토리가 /sqllib 디렉토리에 있으며 여기에 DB2 데이터베이스 제품이 설치되어 있습니다. 인스턴스 이름은 서비스 이름과 동일하며 충돌하지 않아야 합니다. 인스턴스 이름은 다른 서비스 이름과 동일하지 않습니다. 서비스를 작성하려면 올바른 권한이 있어야 합니다.

파티션된 데이터베이스 환경에서는 인스턴스에 속하는 모든 데이터베이스 파티션 서버에서 인스턴스 디렉토리를 공유합니다. 따라서 인스턴스 디렉토리를 인스턴스의 모든 컴퓨터가 액세스할 수 있는 네트워크 공유 드라이브에 작성해야 합니다.

db2nodes.cfg

db2nodes.cfg 파일은 DB2 인스턴스에 참여하는 데이터베이스 파티션 서버를 정의하는 데 사용됩니다. db2nodes.cfg 파일은 또한 데이터베이스 파티션 서버 통신에 신속한 상호 연결망을 사용하려는 경우 신속한 상호 연결망의 IP 주소 또는 호스트 이름을 지정하는 데 사용됩니다.

다중 인스턴스(Linux, UNIX)

루트 특권을 사용하여 DB2 제품을 설치한 경우에는 Linux 또는 UNIX 운영 체제에서 둘 이상의 인스턴스를 사용할 수 있습니다. 모든 인스턴스가 동시에 실행되지만 각 인스턴스는 독립적입니다. 따라서 한 번에 하나의 데이터베이스 관리 프로그램 인스턴스에 대해서만 작업할 수 있습니다.

주: 둘 이상의 인스턴스 간에 환경 충돌이 발생하지 않도록 하려면 각 인스턴스에 고유한 홈 디렉토리가 있어야 합니다. 홈 디렉토리가 공유되면 오류가 리턴됩니다. 각각의 홈 디렉토리는 같거나 서로 다른 파일 시스템에 있을 수 있습니다.

시스템 관리(SYSADM) 그룹인 그룹과 인스턴스 소유자는 모든 인스턴스와 연관됩니다. 인스턴스 소유자 및 SYSADM 그룹은 인스턴스 작성 프로세스 중에 지정됩니다. 하나의 인스턴스에는 하나의 사용자 ID 또는 사용자 이름만 사용할 수 있으며 해당 사용자 ID 또는 사용자 이름을 *인스턴스 소유자*라고도 합니다.

각 인스턴스 소유자에게는 고유한 홈 디렉토리가 있어야 합니다. 인스턴스를 실행하는 데 필요한 모든 구성 파일은 인스턴스 소유자의 사용자 ID 또는 사용자 이름의 홈 디렉토리에 작성됩니다. 인스턴스 소유자의 사용자 ID 또는 사용자 이름을 시스템에서 제거해야 하는 경우에는 해당 인스턴스와 연관된 파일이 유실되고 이 인스턴스에 저장된 데이터에 액세스할 수 없게 됩니다. 이로 인해 사용하려는 인스턴스 소유자 사용자 ID 또는 사용자 이름을 고유하게 지정하여 데이터베이스 관리 프로그램을 실행해야 합니다.

인스턴스 소유자의 기본 그룹 또한 중요합니다. 이 기본 그룹은 자동으로 인스턴스의 시스템 관리 그룹이 되어 인스턴스에 대한 SYSADM 권한을 획득합니다. 인스턴스 소유자의 기본 그룹 구성원인 기타 사용자 ID 또는 사용자 이름도 해당 레벨의 권한을 획득합니다. 따라서 인스턴스 소유자의 사용자 ID 또는 사용자 이름을 인스턴스를 관리하도록 예약된 기본 그룹에 지정해야 합니다. (또한 기본 그룹을 인스턴스 소유자 사용자 ID 또는 사용자 이름에 지정해야 합니다. 그렇지 않으면 시스템 디폴트 기본 그룹이 사용됩니다.)

인스턴스의 시스템 관리 그룹으로 사용할 그룹이 이미 있는 경우에는 인스턴스 소유자 사용자 ID 또는 사용자 이름 작성 시 해당 그룹을 기본 그룹으로 지정하면 됩니다. 기타 사용자에게 인스턴스에 대한 관리 권한을 부여하려면 해당 사용자를 시스템 관리 그룹으로 지정된 그룹에 추가하십시오.

인스턴스 간 SYSADM 권한을 구분하려면 각 인스턴스 소유자 사용자 ID 또는 사용자 이름이 서로 다른 기본 그룹을 사용하게 하십시오. 그러나 다중 인스턴스에서 공통 SYSADM 권한을 사용하도록 선택한 경우에는 다중 인스턴스에 동일한 기본 그룹을 사용할 수 있습니다.

다중 인스턴스(Windows)

동일한 컴퓨터에서 데이터베이스 관리 프로그램의 다중 인스턴스를 실행할 수 있습니다. 데이터베이스 관리 프로그램의 각 인스턴스는 자체 데이터베이스를 유지보수하고 고유한 데이터베이스 관리 프로그램 구성 매개변수를 갖습니다.

주: 또한 인스턴스는 컴퓨터에서 서로 다른 데이터베이스 관리 프로그램 레벨에 있는 DB2 사본에 속할 수 있습니다.

데이터베이스 관리 프로그램의 인스턴스는 다음 항목으로 구성됩니다.

- 인스턴스를 나타내는 Windows 서비스. 서비스 이름은 인스턴스 이름과 동일합니다. 서비스 패널의 서비스 표시 이름은 인스턴스 이름이며 이름 앞에『DB2 - 』문자열

이 붙습니다. 예를 들어, 『DB2』라는 이름의 인스턴스에는 표시 이름이 『DB2 - <DB2 사본 이름> - DB2』인 『DB2』라는 Windows 서비스가 있습니다.

주: 클라이언트 인스턴스에는 Windows 서비스가 작성되지 않습니다.

- 인스턴스 디렉토리. 이 디렉토리에는 데이터베이스 관리 프로그램 구성 파일, 시스템 데이터베이스 디렉토리, 노드 디렉토리, 데이터베이스 연결 서비스(DCS) 디렉토리, 인스턴스와 연관된 모든 진단 로그 및 덤프 파일이 들어 있습니다. 인스턴스 디렉토리는 Windows 계열 운영 체제의 개정판에 따라 다릅니다. Windows에서 디폴트 디렉토리를 검증하려면 `db2set DB2INSTPROF` 명령을 사용하여 **DB2INSTPROF** 환경 변수 설정을 점검하십시오. **DB2INSTPROF** 환경 변수를 변경하여 디폴트 인스턴스 디렉토리를 변경할 수도 있습니다. 예를 들어, 디렉토리를 c:₩DB2PROFS로 설정하려면 다음을 수행하십시오.
  - db2set.exe -g 명령을 사용하여 **DB2INSTPROF**를 c:₩DB2PROFS로 설정하십시오.
  - DB2ICRT.exe 명령을 실행하여 인스턴스를 작성하십시오.
- Windows 운영 체제에서 인스턴스를 작성하는 경우 인스턴스 디렉토리 및 db2cli.ini 파일과 같은 사용자 데이터 파일의 디폴트 위치는 다음 디렉토리입니다.
  - Windows XP 및 Windows 2003 운영 체제: Documents and Settings₩All Users₩Application Data₩IBM₩DB2₩`Copy Name`
  - Windows 2008 및 Windows Vista 이상의 운영 체제: ProgramData₩IBM₩DB2₩`Copy Name`

  여기서 `Copy Name`은 DB2 사본 이름을 나타냅니다.

  주: `db2cli.ini` 파일의 위치는 Microsoft ODBC 드라이버 관리자 사용 여부, 사용되는 데이터 소스 이름(DSN)의 유형, 설치 중인 클라이언트 또는 드라이버의 유형, 레지스트리 변수 **DB2CLIINIPATH**가 설정되었는지 여부에 따라 변경됩니다. 자세한 정보는 *Call Level Interface Guide and Reference, Volume 1*의 『db2cli.ini initialization file』을 참조하십시오.

인스턴스 작성

인스턴스가 데이터베이스 관리 프로그램 설치의 일부로 작성되더라도 비즈니스 요구에 따라 추가 인스턴스를 작성해야 합니다.

전제조건

Windows에서 관리자 그룹에 속해 있는 경우 또는 Linux 또는 UNIX 플랫폼에서 루트 권한이 있는 경우 인스턴스를 추가할 수 있습니다. 인스턴스를 추가하는 컴퓨터는 인스턴스 소유 컴퓨터(노드 0)가 됩니다. DB2 관리 서버가 상주한 컴퓨터에서 인스턴스를 추가해야 합니다. 인스턴스 ID는 루트가 되거나 암호가 만기되어서는 안됩니다.

제한사항

- Linux 및 UNIX 운영 체제에서 비루트 설치를 위해 추가 인스턴스를 작성할 수 없습니다.
- DB2 인스턴스 작성에 기존 사용자 ID가 사용되는 경우 해당 사용자 ID가 다음과 같은지 확인하십시오.
  - 잠겨 있지 않음
  - 암호가 만기되지 않았음

명령행을 사용하여 인스턴스를 추가하려면 다음과 같이 입력하십시오.

```
db2icrt <instance_name>
```

AIX 서버에서 인스턴스를 작성하려면 예를 들어 다음과 같이 분리 사용자 ID를 제공해야 합니다.

```
DB2DIR/instance/db2icrt -u db2fenc1 db2inst1
```

db2icrt 명령을 사용하여 다른 DB2 인스턴스를 추가하려는 경우 인스턴스 소유자의 로그인 이름을 제공하고 해당 인스턴스의 인증 유형을 선택적으로 지정해야 합니다. 인증 유형은 해당 인스턴스 아래에서 작성된 모든 데이터베이스에 적용됩니다. 인증 유형은 사용자 인증이 발생하는 명령문입니다.

DB2INSTPROF 환경 변수를 사용하여 DB2PATH에서 인스턴스 디렉토리의 위치를 변경할 수 있습니다. 해당 인스턴스 디렉토리에 대한 쓰기 액세스 권한이 필요합니다. DB2PATH 이외의 경로에서 디렉토리를 작성하려면 db2icrt 명령을 입력하기 전에 DB2INSTPROF를 설정해야 합니다.

DB2 Enterprise Server Edition(ESE)의 경우, 파티션된 데이터베이스 시스템인 새 인스턴스를 추가 중임을 선언해야 합니다. 또한 하나 이상의 데이터베이스 파티션이 있는 ESE 인스턴스를 사용하여 작업하는 경우 및 FCM(Fast Communication Manager)을 사용하여 작업하는 경우 인스턴스 작성 시 추가 TCP/IP 포트를 정의하여 데이터베이스 파티션 간에 여러 연결을 설정할 수 있습니다.

예를 들어 Windows 운영 체제의 경우 -r <port range> 매개변수와 함께 db2icrt 명령을 사용하십시오. 포트 범위는 다음과 같이 표시됩니다. 여기서 base_port는 FCM에서 사용할 수 있는 첫 번째 포트이고 end_port는 FCM에서 사용할 수 있는 포트 번호 범위의 마지막 포트입니다.

```
-r:<base_port,end_port>
```

인스턴스 수정

인스턴스는 연속 설치 및 제품 제거의 영향을 가능한 한 받지 않도록 설계됩니다. Linux 및 UNIX에서는 실행 파일이나 구성요소 설치 또는 제거 이후 인스턴스를 갱신할 수 있습니다. Windows에서는 db2iupdt 명령을 실행합니다.

대부분의 경우 기존 인스턴스는 설치 또는 제거 중인 제품의 기능에 대한 액세스를 자동으로 상속하거나 잃습니다. 그러나 특정 실행 파일이나 구성요소가 설치 또는 제거되는 경우 기존 인스턴스는 자동으로 새 시스템 구성 매개변수를 상속하거나 모든 추가적인 기능에 대한 액세스 권한을 획득하지 않습니다. 인스턴스를 갱신해야 합니다.

프로그램 임시 수정(PTF) 또는 패치를 설치하여 데이터베이스 관리 프로그램이 갱신되는 경우 db2iupdt 명령(루트 설치) 또는 db2nrupdt 명령(루트 서버가 아닌 설치)을 사용하여 모든 기존 데이터베이스 인스턴스를 갱신해야 합니다.

인스턴스를 변경하거나 삭제하기 전에 인스턴스에 있는 인스턴스 및 데이터베이스 파티션 서버를 검토해야 합니다.

인스턴스 구성 갱신(Linux, UNIX)

이 항목은 루트 인스턴스에만 적용됩니다. 루트가 아닌 인스턴스를 갱신하려면 db2nrupdt 명령을 실행합니다.

db2iupdt 명령을 실행하면 다음을 수행하여 지정된 인스턴스가 갱신됩니다.

- 인스턴스 소유자의 홈 디렉토리의 `sqllib` 서브디렉토리에서 파일을 교체합니다.
- 노드 유형이 변경되면 새 데이터베이스 관리 프로그램 구성 파일이 작성됩니다. 이러한 작업은 기존 데이터베이스 관리 프로그램 파일의 관련 값을 새 노드 유형의 디폴트 데이터베이스 관리 프로그램 파일과 병합하여 수행됩니다. 새 데이터베이스 관리 프로그램 구성 파일이 작성되면 이전 파일은 인스턴스 소유자의 홈 디렉토리에 있는 `sqllib` 서브디렉토리의 `backup` 서브디렉토리에 백업됩니다.

db2iupdt 명령은 AIX의 `/usr/opt/db2_09_05/instance/` 디렉토리에 있습니다. db2iupdt 명령은 HP-UX, Solaris 또는 Linux의 `/opt/IBM/db2/V9.5/instance/` 디렉토리에 있습니다.

명령행을 사용하여 인스턴스를 갱신하려면 다음과 같이 입력하십시오.

```
db2iupdt InstName
```

`InstName`은 인스턴스 소유자의 로그인 이름입니다.

다음 명령과 연관된 다른 선택적 매개변수가 있습니다.

-h 또는 -?

이 명령에 대한 도움말 메뉴를 표시합니다.

-d 문제점 판별 중 사용할 디버그 모드를 설정합니다.

-a AuthType

인스턴스에 대한 인증 유형을 지정합니다. 유효한 인증 유형은 SERVER, SERVER_ENCRYPT 또는 CLIENT입니다. DB2 서버가 설치되어 있는 경우 인증 유형을 지정하지 않으면 디폴트값은 SERVER입니다. 그렇지 않으면 인증 유형은 CLIENT로 설정됩니다. 인스턴스의 인증 유형은 해당 인스턴스가 소유한 모든 데이터베이스에 적용됩니다.

-e 존재하는 각 인스턴스를 갱신할 수 있습니다. db2ilist를 사용하여 기존 인스턴스를 나열하십시오.

-u Fenced ID

사용자 정의 함수(UDF) 및 스토어드 프로시저가 실행되는 분리 사용자의 이름을 지정합니다. Data Server Client 또는 DB2 Software Developer's Kit를 설치한 경우에는 이름을 지정할 필요가 없습니다. 다른 DB2 제품의 경우 이는 필수 매개변수입니다. 주: 분리 ID는 ″루트″ 또는 ″bin″일 수 없습니다.

-k 이 매개변수는 현재 인스턴스 유형을 보존합니다. 이 매개변수를 지정하지 않으면 현재 인스턴스가 다음 순서에서 사용할 수 있는 최고 인스턴스 유형으로 업그레이드됩니다.

- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트

예를 들면, 다음과 같습니다.

- 인스턴스가 작성된 후 DB2 Workgroup Server Edition 또는 DB2 Enterprise Server Edition 설치를 수행하면 다음 명령을 입력하여 해당 인스턴스를 갱신하십시오.

```
db2iupdt -u db2fenc1 db2inst1
```

- 인스턴스 작성 후 DB2 Connect Enterprise Edition 설치를 수행하면 다음과 같이 해당 인스턴스 이름을 분리 ID로 사용할 수도 있습니다.

```
db2iupdt -u db2inst1 db2inst1
```

- 클라이언트 인스턴스를 갱신하려면 다음 명령을 호출하십시오.

```
db2iupdt db2inst1
```

인스턴스 구성 갱신(Windows)

Windows에서 인스턴스 구성을 갱신하려면 db2iupdt 명령을 사용합니다.

db2iupdt 명령을 실행하면 다음을 수행하여 지정된 인스턴스가 갱신됩니다.

- 인스턴스 소유자의 홈 디렉토리의 sqllib 서브디렉토리에서 파일을 교체합니다.

- 노드 유형이 변경되면 새 데이터베이스 관리 프로그램 구성 파일이 작성됩니다. 이러한 작업은 기존 데이터베이스 관리 프로그램 파일의 관련 값을 새 노드 유형의 디폴트 데이터베이스 관리 프로그램 파일과 병합하여 수행됩니다. 새 데이터베이스 관리 프로그램 구성 파일이 작성되면 이전 파일은 인스턴스 소유자의 홈 디렉토리에 있는 sqllib 서브디렉토리의 backup 서브디렉토리에 백업됩니다.

db2iupdt 명령은 ₩sqllib₩bin 디렉토리에 있습니다.

명령은 다음과 같이 사용됩니다.

```
db2iupdt InstName
```

InstName은 인스턴스 소유자의 로그인 이름입니다.

다음 명령과 연관된 다른 선택적 매개변수가 있습니다.

/h: hostname
현재 컴퓨터에 대해 하나 이상의 TCP/IP 호스트 이름이 있는 경우 디폴트 TCP/IP 호스트 이름을 겹쳐씁니다.

/p: instance profile path
갱신된 인스턴스에 대한 새 인스턴스 프로파일 경로를 지정합니다.

/r: baseport,endport
여러 데이터베이스 파티션과 함께 실행할 때 파티션된 데이터베이스 인스턴스가 사용할 TCP/IP 포트의 범위를 지정합니다.

/u: username,password
DB2 서비스의 어카운트 이름 및 암호를 지정합니다.

인스턴스 작업

인스턴스로 작업하는 경우 인스턴스를 시작 또는 중지하고 인스턴스에 접속하거나 인스턴스에서 접속 해제할 수 있습니다.

각 인스턴스는 *데이터베이스 관리 프로그램 구성 파일*이라고도 하는 *인스턴스 구성 파일*에 정의된 SYSADM_GROUP에 속한 사용자가 관리합니다. 각 운영 환경에서 사용자 ID 및 사용자 그룹 작성 방법은 다릅니다.

인스턴스 자동 시작

Windows 운영 체제에서 설치 중 작성된 데이터베이스 인스턴스는 디폴트로 자동 시작으로 설정되어 있습니다. db2icrt를 사용하여 작성된 인스턴스는 수동 시작으로 설정되어 있습니다. 시작 유형을 변경하려면 서비스 패널로 이동하여 DB2 서비스의 등록 정보를 변경해야 합니다.

UNIX 운영 체제에서 시스템 시작 후 인스턴스가 자동으로 시작되도록 하려면 다음 명령을 입력하십시오.

```
db2iauto -on <instance name>
```

여기서 <instance name>은 인스턴스의 로그인 이름입니다. UNIX 운영 체제에서 각 시스템 다시 시작 후 인스턴스의 자동 시작을 예방하려면 다음 명령을 입력하십시오.

```
db2iauto -off <instance name>
```

여기서 <instance name>은 인스턴스의 로그인 이름입니다.

인스턴스 시작(Linux, UNIX)

정상적인 비즈니스 조작 중 DB2 데이터베이스를 시작하거나 중지해야 합니다. 예를 들어 인스턴스에서 데이터베이스에 연결, 응용프로그램 프리컴파일, 데이터베이스로 패키지 바인드 또는 호스트 데이터베이스 액세스와 같은 일부 작업을 수행하려면 인스턴스를 시작해야 합니다.

Linux 또는 UNIX 시스템에서 인스턴스를 시작하려면 다음을 수행해야 합니다.

1. 인스턴스에 대한 SYSADM, SYSCTRL 또는 SYSMAINT 권한이 있는 사용자 ID 또는 이름으로 로그인하거나 인스턴스 소유자로 로그인합니다.
2. 다음과 같이 시작 스크립트를 실행합니다. 여기서 INSTHOME은 사용하려는 인스턴스의 홈 디렉토리입니다.

```
. INSTHOME/sqllib/db2profile      (for Bourne or Korn shell)
source INSTHOME/sqllib/db2cshrc   (for C shell)
```

명령행을 사용하여 인스턴스를 시작하려면 다음과 같이 입력하십시오.

```
db2start
```

주: 인스턴스의 데이터베이스 관리 프로그램을 시작하거나 중지하는 명령을 실행하면 DB2 데이터베이스 관리 프로그램에서 해당 명령을 현재 인스턴스에 적용합니다.

인스턴스 시작(Windows)

정상적인 비즈니스 조작 중 DB2 데이터베이스를 시작하거나 중지해야 합니다. 예를 들어 인스턴스에서 데이터베이스에 연결, 응용프로그램 프리컴파일, 데이터베이스로 패키지 바인드 또는 호스트 데이터베이스 액세스와 같은 일부 작업을 수행하려면 인스턴스를 시작해야 합니다.

db2start에서 DB2 데이터베이스 인스턴스를 서비스로 실행하려면 사용자 어카운트에는 Windows 서비스를 시작하도록 Windows 운영 체제에서 정의한 것처럼 올바른 특권이 필요합니다. 사용자 어카운트는 관리자, 서버 운영자 또는 권한이 있는 사용자 그룹의 구성원일 수 있습니다. 확장 보안이 사용 가능한 경우 디폴트로 DB2ADMNS 및 관리자 그룹의 구성원만 데이터베이스를 시작할 수 있습니다.

명령행을 사용하여 인스턴스를 시작하려면 다음과 같이 입력합니다.

```
db2start
```

주: 인스턴스의 데이터베이스 관리 프로그램을 시작하거나 중지하는 명령을 실행하면 DB2 데이터베이스 관리 프로그램에서 해당 명령을 현재 인스턴스에 적용합니다.

db2start 명령은 DB2 데이터베이스 인스턴스를 Windows 서비스로 시작합니다. db2start를 호출할 때 ″/D″ 스위치를 지정하여 Windows에서 DB2 데이터베이스 인스턴스를 프로세스로 실행할 수 있습니다. 또한 DB2 데이터베이스 인스턴스는 제어판이나 NET START 명령을 사용하여 서비스로서 시작될 수 있습니다.

파티션된 데이터베이스 환경에서 실행 중인 경우 각 데이터베이스 파티션 서버는 Windows 서비스로 시작됩니다. ″/D″ 스위치를 사용하여 파티션된 데이터베이스 환경에서 DB2 인스턴스를 프로세스로 시작할 수 없습니다.

인스턴스에 접속 및 인스턴스에서 접속 해제

모든 플랫폼에서 데이터베이스 관리 프로그램의 다른 인스턴스(리모트 상태임)에 접속하려면 ATTACH 명령을 사용합니다. 인스턴스에서 접속 해제하려면 DETACH 명령을 사용합니다.

하나 이상의 인스턴스가 존재해야 합니다.

명령행을 사용하여 인스턴스에 접속하려면 다음과 같이 입력하십시오.

```
db2 attach to <instance name>
```

예를 들어 노드 디렉토리에서 이전에 카탈로그된 testdb2 인스턴스에 접속하려면 다음과 같이 입력하십시오.

```
db2 attach to testdb2
```

예를 들어 testdb2 인스턴스에 대한 유지보수 활동을 수행한 후 명령행을 사용하여 인스턴스에서 접속 해제하려면 다음과 같이 입력하십시오.

```
db2 detach
```

클라이언트 응용프로그램에 접속하고 클라이언트 응용프로그램에서 접속 해제하려면 다음을 수행하십시오.

- 클라이언트 응용프로그램의 인스턴스에 접속하려면 sqleatin API를 호출합니다.
- 클라이언트 응용프로그램의 인스턴스에서 접속 해제하려면 sqledtin API를 호출합니다.

동일한 또는 다른 DB2 사본의 인스턴스로 작업

DB2 사본 또는 다른 DB2 사본에서 동시에 여러 인스턴스를 실행할 수 있습니다.

동일한 DB2 사본의 인스턴스에 대해 작업하려면 다음을 수행해야 합니다.

1. 인스턴스를 작성하거나 모든 인스턴스를 동일한 DB2 사본으로 업그레이드합니다.
2. 해당 인스턴스에 대해 명령을 발행하기 전에 작업하는 데 사용한 인스턴스의 이름으로 DB2INSTANCE 환경 변수를 설정합니다.

하나의 인스턴스가 다른 인스턴스의 데이터베이스에 액세스하도록 하지 않도록 인스턴스의 데이터베이스 파일이 해당 인스턴스 이름과 이름이 동일한 디렉토리 아래에 작성됩니다. 예를 들어 인스턴스 『DB2』를 위해 드라이브 C:에 데이터베이스를 작성하면 C:₩DB2 디렉토리 내에 데이터베이스 파일이 작성됩니다. 유사하게 인스턴스 TEST를 위해 드라이브 C:에 데이터베이스를 작성하면 C:₩TEST 디렉토리 내에 데이터베이스 파일이 작성됩니다. 디폴트로 이러한 환경 변수의 값은 DB2 제품이 설치된 드라이브 이름입니다. 자세한 정보는 *dftdbpath* 데이터베이스 관리 프로그램 구성 매개변수를 참조하십시오.

여러 DB2 사본과 함께 시스템의 인스턴스를 사용하여 작업하려면 다음 방법 중 하나를 사용하십시오.

- 시작 → 프로그램 → IBM DB2 → *<DB2 사본 이름>* → 명령행 도구 → 명령 창에서 명령 창을 사용합니다. 명령 창은 이미 선택한 특정 DB2 사본의 올바른 환경 변수로 설정되어 있습니다.
- 다음과 같이 명령 창에서 db2envar.bat를 사용합니다.
  1. 명령 창을 엽니다.
  2. 응용프로그램에서 사용하도록 하려는 DB2 사본의 완전한 경로를 사용하여 db2envar.bat 파일을 실행합니다.

     ```
     <DB2 Copy install dir>₩bin₩db2envar.bat
     ```

인스턴스 중지(Linux, UNIX)

데이터베이스 관리 프로그램의 현재 인스턴스를 중지해야 합니다.

Linux 또는 UNIX 시스템에서 인스턴스를 중지하려면 다음을 수행해야 합니다.

1. 인스턴스에 대한 SYSADM, SYSCTRL 또는 SYSMAINT 권한이 있는 사용자 ID 또는 이름으로 로그인하거나 인스턴스에 접속합니다. 또는 인스턴스 소유자로 로그인합니다.
2. 중지하려는 특정 데이터베이스에 연결된 모든 응용프로그램 및 사용자가 표시됩니다. 실행 중인 중요한 응용프로그램이 없는지 확인하려면 응용프로그램을 나열합니다. 이렇게 하려면 SYSADM, SYSCTRL 또는 SYSMAINT 권한이 필요합니다.

3. 모든 응용프로그램 및 사용자를 데이터베이스에서 강제로 로그아웃합니다. 사용자를 강제로 로그아웃하려면 SYSADM 또는 SYSCTRL 권한이 필요합니다.

db2stop 명령은 서버에서만 실행될 수 있습니다. 이 명령을 실행하면 데이터베이스 연결이 허용되지 않습니다. 접속된 인스턴스가 있는 경우 해당 인스턴스는 중지되기 전에 강제로 분리됩니다.

주: 명령행 처리기 세션이 인스턴스에 접속되면 terminate 명령을 사용하여 db2stop 명령 실행 전 각 세션을 종료해야 합니다. db2stop 명령은 DB2INSTANCE 환경 변수에 의해 정의된 인스턴스를 중지합니다.

명령행을 사용하여 인스턴스를 중지하려면 다음과 같이 입력하십시오.

```
db2stop
```

db2stop 명령을 사용하여 파티션된 데이터베이스 환경 내의 개별 데이터베이스 파티션을 중지 또는 삭제할 수 있습니다. 파티션된 데이터베이스 환경에서 작업하는 경우 다음과 같이 입력하여 논리적 파티션을 삭제할 수 있습니다.

```
db2stop drop nodenum <0>
```

이 경우 해당 데이터베이스에 액세스하려는 사용자가 없어야 합니다. 액세스하려는 사용자가 있는 경우 SQL6030N 오류 메시지가 표시됩니다.

주: 인스턴스의 데이터베이스 관리 프로그램을 시작하거나 중지하는 명령을 실행하면 DB2 데이터베이스 관리 프로그램에서 해당 명령을 현재 인스턴스에 적용합니다. 자세한 정보는 493 페이지의 『현재 인스턴스 환경 변수 설정』을 참조하십시오.

인스턴스 중지(Windows)

데이터베이스 관리 프로그램의 현재 인스턴스를 중지해야 합니다.

시스템에서 인스턴스를 중지하려면 다음을 수행해야 합니다.

1. DB2 데이터베이스 서비스를 중지하는 사용자 어카운트에는 Windows 운영 체제에서 정의한 것처럼 올바른 특권이 있어야 합니다. 사용자 어카운트는 관리자, 서버 운영자 또는 권한이 있는 사용자 그룹의 구성원일 수 있습니다.
2. 중지하려는 특정 데이터베이스에 연결된 모든 응용프로그램 및 사용자가 표시됩니다. 실행 중인 중요한 응용프로그램이 없는지 확인하려면 응용프로그램을 나열합니다. 이렇게 하려면 SYSADM, SYSCTRL 또는 SYSMAINT 권한이 필요합니다.
3. 모든 응용프로그램 및 사용자를 데이터베이스에서 강제로 로그아웃합니다. 사용자를 강제로 로그아웃하려면 SYSADM 또는 SYSCTRL 권한이 필요합니다.

db2stop 명령은 서버에서만 실행될 수 있습니다. 이 명령을 실행하면 데이터베이스 연결이 허용되지 않습니다. 그러나 접속된 인스턴스가 있는 경우 DB2 데이터베이스 서비스는 중지되기 전에 강제로 분리됩니다.

주: 명령행 처리기 세션이 인스턴스에 접속되면 terminate 명령을 사용하여 db2stop 명령 실행 전 각 세션을 종료해야 합니다. db2stop 명령은 DB2INSTANCE 환경 변수에 의해 정의된 인스턴스를 중지합니다.

시스템에서 인스턴스를 중지하려면 다음 방법 중 하나를 사용하십시오.

- db2stop 명령을 사용하여 중지
- NET STOP 명령을 사용하여 중지
- 응용프로그램 내에서 인스턴스 중지

파티션된 데이터베이스 환경에서 데이터베이스 관리 프로그램을 사용 중이면 각 데이터베이스 파티션 서버가 서비스로 시작된다는 점을 명심하십시오. 각 서비스는 중지되어야 합니다.

주: 인스턴스의 데이터베이스 관리 프로그램을 시작하거나 중지하는 명령을 실행하면 데이터베이스 관리 프로그램에서 해당 명령을 현재 인스턴스에 적용합니다.자세한 정보는 493 페이지의 『현재 인스턴스 환경 변수 설정』을 참조하십시오.

인스턴스 삭제

루트 인스턴스를 삭제하려면 db2idrop 명령을 발행하십시오. 루트가 아닌 인스턴스를 삭제하려면 DB2 데이터베이스 제품을 설치 제거해야 합니다.

프로시저

명령행을 사용하여 루트 인스턴스를 제거하려면 다음을 수행하십시오.

1. 현재 인스턴스를 사용 중인 모든 응용프로그램을 중지시키십시오.
2. 각 명령 창에서 terminate 명령을 실행하여 명령행 처리기를 중지하십시오.
3. db2stop 명령을 수행하여 인스턴스를 중지시키십시오.
4. **DB2INSTPROF** 레지스트리 변수로 표시되는 인스턴스 디렉토리를 백업하십시오.

   Linux 및 UNIX 운영 체제에서는, *INSTHOME*/sqllib 디렉토리의 파일 백업을 고려하십시오(*INSTHOME*은 인스턴스 소유자의 홈 디렉토리임). 예를 들어 데이터베이스 관리 프로그램 구성 파일, db2systm, db2nodes.cfg 파일, 사용자 정의 함수(UDF) 또는 분리 스토어드 프로시저 응용프로그램을 저장하려고 할 수 있습니다.
5. Linux 및 UNIX 운영 체제에서만 인스턴스 소유자로 로그오프하고 루트 권한을 사용하여 사용자로 로그인하십시오.
6. db2idrop 명령을 발행하십시오. 예를 들어, 다음과 같습니다.

   ```
   db2idrop InstName
   ```

   여기서 *InstName*은 삭제된 인스턴스의 이름입니다.

db2idrop 명령은 인스턴스 목록에서 인스턴스 항목을 제거하고 인스턴스 소유자의 홈 디렉토리에 있는 sqllib 서브디렉토리를 제거합니다.

주: Linux 및 UNIX 운영 체제에서는, db2idrop 명령을 발행하고 *INSTHOME*/sqllib 서브디렉토리를 이동할 수 없음을 알리는 메시지를 수신하는 경우 *INSTHOME*/adm 서브디렉토리에 .nfs 확장자를 갖는 파일이 들어있기 때문일 수 있습니다. adm 서브디렉토리는 NFS 마운트 시스템이고 파일은 서버에서 제어됩니다. 디렉토리가 마운트되는 파일 서버에서 *.nfs 파일을 삭제해야 합니다. 그런 다음 *INSTHOME*/sqllib 서브디렉토리를 제거할 수 있습니다.

7. Windows 운영 체제의 경우, 삭제한 인스턴스가 디폴트 인스턴스였던 경우 db2set 명령을 발행하여 새 디폴트 인스턴스를 설정하십시오.

   ```
   db2set db2instdef=instance_name -g
   ```

   여기서 *instance_name*은 기존 인스턴스의 이름입니다.

8. Linux 및 UNIX 운영 체제의 경우 인스턴스 소유자의 ID 및 그룹이 해당 인스턴스에만 사용되는 경우 제거합니다. 인스턴스를 다시 작성할 계획인 경우 이들을 제거하지 마십시오.

   인스턴스 소유자 및 인스턴스 소유자 그룹이 다른 용도로 사용될 수 있으므로 이 단계는 선택적입니다.

제 2 부 데이터베이스

제 5 장 데이터베이스

DB2 데이터베이스는 *관계형 데이터베이스*입니다. *데이터베이스*에는 서로 관련되어 있는 테이블의 모든 데이터가 저장됩니다. 데이터가 공유되고 중복이 최소화될 수 있도록 테이블 간에 관계가 설정됩니다.

*관계형 데이터베이스*는 테이블 세트로 취급되고 관계형 데이터 모델에 따라 조작되는 데이터베이스입니다. 관계형 데이터베이스에는 데이터 저장, 관리 및 액세스에 사용되는 오브젝트 세트가 들어 있습니다. 이러한 오브젝트에는 테이블, 뷰, 인덱스, 함수, 트리거 및 패키지가 있습니다. 오브젝트는 시스템을 통해 정의되거나(시스템 정의 오브젝트) 사용자를 통해 정의될 수 있습니다(사용자 정의 오브젝트).

*분산 관계형 데이터베이스*는 서로 연결된 여러 컴퓨터 시스템에 분산되어 있는 테이블 세트와 기타 오브젝트로 구성됩니다. 각 컴퓨터 시스템에는 해당 환경의 테이블을 관리하기 위한 관계형 데이터베이스 관리 프로그램이 있습니다. 데이터베이스 관리 프로그램은 지정된 데이터베이스 관리 프로그램이 다른 컴퓨터 시스템에서 SQL문을 실행할 수 있도록 서로 통신하고 협력합니다.

*파티션된 관계형 데이터베이스*는 다중 데이터베이스 파티션에서 데이터가 관리되는 관계형 데이터베이스입니다. 대부분의 SQL문은 이와 같이 데이터베이스 파티션에서 데이터가 분리되는 것을 명확히 알 수 있습니다. 그러나 일부 데이터 정의 언어(DDL) 명령문은 데이터베이스 파티션 정보를 고려합니다(예: CREATE DATABASE PARTITION GROUP). DDL은 데이터베이스에서 데이터 관계를 설명하는 데 사용되는 SQL문의 서브세트입니다.

*페더레이티드 데이터베이스*는 다중 데이터 소스(예: 개별 관계형 데이터베이스)에 데이터가 저장되어 있는 관계형 데이터베이스입니다. 데이터는 모두 단일 대형 데이터베이스에 있는 것처럼 표시되며 일반적인 SQL 쿼리를 통해 데이터에 액세스할 수 있습니다. 데이터의 변경사항은 명시적으로 해당 데이터 소스로 방향지정됩니다.

데이터베이스 설계

데이터베이스 설계 시, 엔티티 세트 및 해당 특성 또는 *속성*과 규칙 또는 이들 엔티티 간의 관계가 포함된 실제 비즈니스 시스템을 모델링합니다.

첫 번째 단계는 나타내려는 시스템에 대해 설명하는 것입니다. 예를 들어, 시스템을 발행하기 위해 데이터베이스를 작성하는 경우 시스템에는 책, 저자, 편집자 및 제공자와 같은 여러 유형의 엔티티가 포함됩니다. 이들 각각의 엔티티에는 기록해야 하는 특정 정보 또는 속성이 있습니다.

- *책:* 제목, ISBN, 발행 날짜, 위치, 제공자, ...

- *저자:* 이름, 주소, 전화 및 팩스 번호, 전자 우편 주소, ...
- *편집자:* 이름, 주소, 전화 및 팩스 번호, 전자 우편 주소, ...
- *제공자:* 이름, 주소, 전화 및 팩스 번호, 전자 우편 주소, ...

이와 같은 유형의 엔티티 및 해당 속성을 나타내기 위해서 데이터베이스가 필요할 뿐만 아니라 이러한 엔티티를 서로 관련시킬 방법도 필요합니다. 예를 들면, 책과 저자의 관계, 책/저자와 편집자의 관계 및 책/저자와 제공자의 관계를 나타내야 합니다.

데이터베이스의 엔티티 간 관계에는 세 가지 유형이 있습니다.

일대일 관계

이 유형의 관계에서는 엔티티의 각 인스턴스가 다른 엔티티의 하나의 인스턴스에만 관련됩니다. 현재, 위에 설명한 시나리오에서는 일대일 관계가 없습니다.

일대다 관계

이 유형의 관계에서는 엔티티의 각 인스턴스가 다른 엔티티의 하나 이상의 인스턴스와 관련됩니다. 예를 들면, 한 저자가 여러 책을 작성했을 수도 있고, 특정 책의 저자는 한 명뿐일 수도 있습니다. 이 유형은 관계형 데이터베이스에서 모델화된 가장 일반적인 관계 유형입니다.

다대다 관계

이 유형의 관계에서는 지정된 엔티티의 여러 인스턴스가 다른 엔티티의 하나 이상의 인스턴스와 관련됩니다. 예를 들면, 공동 작가가 여러 책을 썼을 수 있습니다.

데이터베이스는 테이블로 구성되기 때문에 테이블의 각 셀이 단일 뷰를 보유하는, 이 데이터를 가장 잘 보유할 테이블 세트를 구성해야 합니다. 여러 방법으로 이 태스크를 수행할 수 있습니다. 데이터베이스 설계자는 가능한 최상의 테이블 세트를 구성해야 합니다.

예를 들어, 여러 행과 컬럼이 있는 단일 테이블을 작성하여 모든 정보를 보유할 수 있습니다. 그러나 이 방법을 사용하면 일부 정보가 반복됩니다. 두 번째로 데이터 항목과 데이터 유지보수에 시간이 오래 걸리고 오류가 잘 발생합니다. 이러한 단일 테이블 설계에 비하여 *관계형 데이터베이스*를 작성하면 여러 개의 단순 테이블을 사용하여 중복을 줄이고 크기가 크고 관리가 불가능한 테이블로 인해 어려움이 발생하지 않도록 할 수 있습니다. 관계형 데이터베이스에서는 테이블에 단일 엔티티 유형에 대한 정보가 포함되어야 합니다.

또한 여러 사용자가 데이터에 액세스하고 변경하므로 관계형 데이터베이스의 데이터 무결성이 유지되어야 합니다. 데이터를 공유할 때마다 데이터베이스 테이블에 있는 값의 정확성을 확인해야 합니다.

다음을 수행할 수 있습니다.

- 분리 레벨을 사용하여 데이터에 액세스하는 동안 데이터를 잠그거나 다른 프로세스와 분리하는 방법을 판별할 수 있습니다.
- 비즈니스 규칙을 적용하도록 제한조건을 정의하여 데이터를 보호하고 데이터 간 관계를 정의할 수 있습니다.
- 복잡한 데이터 및 교차 테이블 데이터 유효성 검증을 수행하는 트리거를 작성할 수 있습니다.
- 데이터를 일관성 있는 상태로 리스토어할 수 있도록 데이터를 보호하기 위한 복구 전략을 구현할 수 있습니다.

데이터베이스 설계는 여기서 설명한 것보다 훨씬 복잡한 태스크이며 스페이스 요구사항, 키, 인덱스, 제한조건, 보안 및 권한 부여 등 고려해야 할 항목이 많습니다. DB2 정보 센터 및 이 주제에 관한 여러 DB2 판매 서적에서 해당 정보를 찾을 수 있습니다.

데이터베이스 디렉토리 및 파일

데이터베이스를 작성하는 경우 디폴트 정보를 포함한 데이터베이스에 대한 정보는 디렉토리 계층 구조에 저장됩니다.

계층 디렉토리 구조는 사용자가 CREATE DATABASE 명령에 제공한 정보로 판별된 위치에 작성되어 있습니다. 데이터베이스 작성 시 디렉토리 경로 또는 드라이브의 위치를 지정하지 않는 경우 디폴트 위치가 사용됩니다.

CREATE DATABASE 명령에서 데이터베이스 경로로 지정하는 디렉토리에서 *instance* 이름을 사용하는 서브디렉토리가 작성됩니다. 이 서브디렉토리는 동일한 디렉토리의 다른 인스턴스에 작성되는 데이터베이스가 동일한 경로를 사용하지 않도록 합니다. 인스턴스 이름 서브디렉토리 아래 NODE0000이라는 서브디렉토리가 작성됩니다. 이 서브디렉토리는 논리적으로 파티션된 데이터베이스 환경에서 데이터베이스 파티션을 구별합니다. 노드 이름 디렉토리 아래 SQL00001이라는 서브디렉토리가 작성됩니다. 이 서브디렉토리의 해당 이름은 데이터베이스 토큰을 사용하며 작성 중인 데이터베이스를 나타냅니다. SQL00001에는 첫 번째로 작성된 데이터베이스와 연관된 오브젝트가 포함되고 이후의 데이터베이스에는 SQL00002 등으로 높은 숫자가 지정됩니다. 이들 서브디렉토리는 CREATE DATABASE 명령에서 지정한 디렉토리의 해당 인스턴스에 작성된 데이터베이스를 구별합니다.

디렉토리 구조는 다음과 같이 표시됩니다. *your_database_path*/*your_instance*/NODE0000/SQL00001/

데이터베이스 디렉토리에는 CREATE DATABASE 명령의 파트로 작성되는 다음 파일이 들어 있습니다.

- SQLBP.1 및 SQLBP.2 파일에는 버퍼 풀 정보가 들어 있습니다. 이들 파일은 백업을 위해 서로를 백업한 파일입니다.

- SQLSPCS.1 및 SQLSPCS.2 파일에는 테이블 스페이스 정보가 들어 있습니다. 이들 파일은 백업을 위해 서로를 백업한 파일입니다.
- SQLSGF.1 및 SQLSGF.2 파일에는 데이터베이스의 자동 스토리지 기능과 연관된 스토리지 경로 정보가 들어 있습니다. 이들 파일은 유지보수 및 백업을 위해 서로를 백업한 파일입니다. CREATE DATABASE *dbname* **AUTOMATIC STORAGE YES** 명령이나 ALTER DATABASE *dbname ADD STORAGE ON*문 이후에 자동 스토리지가 사용 가능할 때 데이터베이스에 사용할 파일이 작성됩니다.
- SQLDBCONF 파일에는 데이터베이스 구성 정보가 들어 있습니다. 이 파일을 편집하지 마십시오.

  주: 이전 릴리스에서는 SQLDBCON 파일이 사용되었으며 SQLDBCONF가 손상되는 경우 사용할 수 있는 유사한 정보가 이 파일에 들어 있습니다.
  구성 매개변수를 변경하려면 UPDATE DATABASE CONFIGURATION 및 RESET DATABASE CONFIGURATION 명령을 사용하십시오.
- DB2RHIST.ASC 실행기록 파일과 해당 백업 DB2RHIST.BAK에는 백업, 리스토어, 테이블 로딩, 테이블 재구성, 테이블 스페이스 변경 및 기타 데이터베이스 변경사항에 대한 실행기록 정보가 들어 있습니다.

  DB2TSCHG.HIS 파일에는 로그 파일 레벨의 테이블 스페이스 변경사항에 대한 실행기록이 들어 있습니다. 각 로그 파일의 경우 DB2TSCHG.HIS에는 로그 파일의 영향을 받는 테이블 스페이스를 식별하는 데 유용한 정보가 들어 있습니다. 테이블 스페이스 복구에서 이 파일의 정보를 사용하여 테이블 스페이스 복구 시 처리할 로그 파일을 판별합니다. 텍스트 편집기에서 두 실행기록 파일 모드의 컨텐츠를 점검할 수 있습니다.
- 로그 제어 파일 SQLOGCTL.LFH.1과 해당 미러 사본 SQLOGCTL.LFH.2 및 SQLOGMIR.LFH에는 사용 중인 로그에 대한 정보가 들어 있습니다.

  복구 처리에서는 이들 파일의 정보를 사용하여 로그에서 복구를 시작할 지점을 판별합니다. SQLOGDIR 서브디렉토리에는 실제 로그 파일이 들어 있습니다.

  주: 사용자 데이터에 사용되는 디스크가 아닌 다른 디스크에 로그 서브디렉토리가 맵핑되도록 해야 합니다. 그러면 디스크 문제점이 데이터 또는 로그에만 한정되고 둘 다에 영향을 주지는 않습니다. 로그 파일과 데이터베이스 컨테이너는 동일한 디스크 헤드 동작을 두고 경합하지 않기 때문에 이는 상당한 성능상 이점을 제공합니다. 로그 서브디렉토리 위치를 변경하려면 **newlogpath** 데이터베이스 구성 매개변수를 변경하십시오.
- SQLINSLK 파일은 데이터베이스 관리 프로그램의 하나의 인스턴스에서만 데이터베이스가 사용되도록 하는 데 유용합니다.

데이터베이스가 작성되면 동시에 자세한 교착 상태 이벤트 모니터가 작성됩니다. 자세한 교착 상태 이벤트 모니터 파일은 카탈로그 노드의 데이터베이스 디렉토리에 저장됩

니다. 이벤트 모니터가 출력할 최대 파일 수에 접근하는 경우 이벤트 모니터가 비활성화되고 통지 로그에 메시지가 작성됩니다. 이는 이벤트 모니터가 너무 많은 디스크 스페이스를 사용하지 않도록 예방합니다. 더 이상 필요하지 않은 출력 파일을 제거하면 다음에 데이터베이스를 활성화할 때 이벤트 모니터가 다시 활성화됩니다.

자동이 아닌 스토리지 데이터베이스의 SMS 데이터베이스 디렉토리 관련 추가 정보

자동이 아닌 스토리지 데이터베이스에서 SQLT* 서브디렉토리에는 정상 작동 데이터베이스에 필요한 디폴트 시스템 관리 스페이스(SMS) 테이블 스페이스가 들어 있습니다. 세 개의 디폴트 테이블 스페이스가 작성됩니다.

- SQLT0000.0 서브디렉토리에는 시스템 카탈로그 테이블과 함께 카탈로그 테이블 스페이스가 들어 있습니다.
- SQLT0001.0 서브디렉토리에는 디폴트 임시 테이블 스페이스가 들어 있습니다.
- SQLT0002.0 서브디렉토리에는 디폴트 사용자 데이터 테이블 스페이스가 들어 있습니다.

각 서브디렉토리 또는 컨테이너에는 SQLTAG.NAM이라는 파일이 작성되어 있습니다. 이 파일은 서브디렉토리를 사용 중인 것으로 표시하여 후속 테이블 스페이스 작성 시 이들 서브디렉토리를 사용하지 않도록 합니다.

또한 SQL*.DAT라는 파일에는 서브디렉토리 또는 컨테이너에 들어 있는 각 테이블에 대한 정보가 저장됩니다. 별표(*)는 각 테이블을 식별하는 고유 숫자 세트로 교체됩니다. 테이블 유형, 테이블의 재구성 상태나 테이블 관련 인덱스, LOB 또는 LONG 필드가 존재하는지 여부에 따라서 각각의 SQL*.DAT 파일에 다음 파일 중 하나 이상이 있을 수 있습니다.

- SQL*.BKM(MDC 테이블인 경우 블록 할당 정보 포함)
- SQL*.LF(LONG VARCHAR 또는 LONG VARGRAPHIC 데이터 포함)
- SQL*.LB(BLOB, CLOB 또는 DBCLOB 데이터 포함)
- SQL*.XDA(XML 데이터 포함)
- SQL*.LBA(SQL*.LB 파일에 대한 할당 및 여유 공간 정보 포함)
- SQL*.INX(인덱스 테이블 데이터 포함)
- SQL*.IN1(인덱스 테이블 데이터 포함)
- SQL*.DTR(SQL*.DAT 파일의 재구성 관련 임시 데이터 포함)
- SQL*.LFR(SQL*.LF 파일의 재구성 관련 임시 데이터 포함)
- SQL*.RLB(SQL*.LB 파일의 재구성 관련 임시 데이터 포함)
- SQL*.RBA(SQL*.LBA 파일의 재구성 관련 임시 데이터 포함)

데이터베이스 구성 파일

각 데이터베이스에 *데이터베이스 구성 파일*이 작성됩니다. 버전 8.2 이전에는 이 파일을 SQLDBCON이라고 했으며 버전 8.2 이후로는 SQLDBCONF라고 합니다. 이 파일은 이미 작성되어 있습니다.

이 파일에는 다음과 같이 데이터베이스 사용에 영향을 미치는 여러 *구성 매개변수*의 값이 들어 있습니다.

- 데이터베이스 작성 시 지정되거나 사용되는 매개변수(예: 데이터베이스 코드 페이지, 조합 시퀀스, DB2 데이터베이스 릴리스 레벨)
- 데이터베이스의 현재 상태를 표시하는 매개변수(예: 백업 보류 플래그, 데이터베이스 일관성 플래그, 롤 포워드 보류 플래그)
- 데이터베이스 조작에서 사용하는 시스템 자원 양을 정의하는 매개변수(예: 버퍼 풀 *크기*, 데이터베이스 로깅, 정렬 메모리 *크기*)

주: DB2 데이터베이스 관리 프로그램이 제공한 것이 이외의 방법을 사용하여 db2system, SQLDBCON(버전 8.2 이전) 또는 SQLDBCONF(버전 8.2 이후) 파일을 편집하는 경우 데이터베이스를 사용할 수 없게 될 수 있습니다. 그러므로 데이터베이스 관리 프로그램에서 문서화되고 지원되는 방법 이외의 방법을 사용하여 해당 파일을 변경하지 마십시오.

성능 팁: 여러 구성 매개변수에 디폴트값이 지정되어 있지만 사용자의 데이터베이스에서 최적의 성능을 발휘하려면 매개변수 값을 갱신해야 합니다. 디폴트로, 일부 매개변수의 초기값이 사용자 환경에 적합하게 미리 구성되도록 CREATE DATABASE 명령의 파트로 구성 어드바이저가 호출됩니다.

다중 파티션 데이터베이스의 경우: 둘 이상의 데이터베이스 파티션에 분산된 데이터베이스가 있는 경우 모든 데이터베이스 파티션에서 구성 파일이 동일해야 합니다. 쿼리 컴파일러가 로컬 노드 구성 파일의 정보에 따라 분산 SQL문을 컴파일하고 SQL문의 요구사항에 부합하도록 액세스 플랜을 작성하므로 일관성이 있어야 합니다. 데이터베이스 파티션에서 여러 구성 파일을 유지보수하면 명령문이 준비된 데이터베이스 파티션에 따라 다른 액세스 플랜이 적용됩니다.

노드 디렉토리

첫 번째 데이터베이스 파티션이 카탈로그되는 경우 데이터베이스 관리 프로그램이 노드 *디렉토리*를 작성합니다.

데이터베이스 파티션을 카탈로그하려면 CATALOG NODE 명령을 사용하십시오. 로컬 노드 디렉토리 컨텐츠를 나열하려면 LIST NODE DIRECTORY 명령을 사용하십시오.

노드 디렉토리는 각 데이터베이스 클라이언트에서 작성되고 유지됩니다. 디렉토리에는 클라이언트가 액세스할 수 있는 데이터베이스가 하나 이상 있는 각 리모트 워크스테이션

의 항목이 포함됩니다. DB2 클라이언트는 데이터베이스 연결이나 인스턴스 접속이 요청될 때마다 노드 디렉토리의 통신 엔드 포인트 정보를 사용합니다.

디렉토리의 항목에도 클라이언트에서 리모트 데이터베이스 파티션으로 통신하는 데 사용할 통신 프로토콜 유형에 대한 정보가 들어 있습니다. 로컬 데이터베이스 파티션을 카탈로그하면 동일한 컴퓨터에 상주하는 인스턴스의 별명이 작성됩니다.

로컬 데이터베이스 디렉토리

로컬 데이터베이스 디렉토리 파일은 데이터베이스가 정의된 각 경로에 존재합니다(또는 Windows 운영 체제의 경우에는 『드라이브』에 존재). 이 디렉토리에는 해당 위치에서 액세스할 수 있는 각 데이터베이스당 하나의 항목이 포함되어 있습니다.

각 항목에 포함된 내용은 다음과 같습니다.

- CREATE DATABASE 명령을 통해 제공된 데이터베이스 이름
- 데이터베이스 별명(별명이 지정되지 않은 경우에는 데이터베이스 이름과 동일)
- CREATE DATABASE 명령을 통해 제공된, 데이터베이스에 대해 설명하는 주석
- 데이터베이스의 루트 디렉토리 이름
- 기타 시스템 정보

시스템 데이터베이스 디렉토리

시스템 데이터베이스 디렉토리 파일은 데이터베이스 관리 프로그램의 인스턴스마다 존재하며 해당 인스턴스에 적합하게 카탈로그된 각각의 데이터베이스마다 하나의 항목이 포함됩니다.

데이터베이스는 CREATE DATABASE 명령 실행 시 내재적으로 카탈로그되며 CATALOG DATABASE 명령을 사용하면 명시적으로 카탈로그됩니다.

작성된 데이터베이스마다 다음 정보가 포함된 항목이 디렉토리에 추가됩니다.

- CREATE DATABASE 명령을 통해 제공된 데이터베이스 이름
- 데이터베이스 별명(별명이 지정되지 않은 경우에는 데이터베이스 이름과 동일)
- CREATE DATABASE 명령을 통해 제공된 데이터베이스 주석
- 로컬 데이터베이스 디렉토리 위치
- 데이터베이스가 *간접적*임을 나타내는 표시기. 이는 데이터베이스가 현재 데이터베이스 관리 프로그램 인스턴스에 상주함을 의미합니다.
- 기타 시스템 정보

UNIX 플랫폼 및 파티션된 데이터베이스 환경에서는 모든 데이터베이스 파티션이 항상 인스턴스 홈 디렉토리의 `sqldbdir` 서브디렉토리에 있는 동일한 시스템 데이터베이스 디렉토리 파일 `sqldbdir`에 액세스하는지 확인해야 합니다. 동일한 `sqldbdir` 서브디렉

토리에 있는 시스템 인텐션 파일 sqldbins 또는 시스템 데이터베이스 디렉토리가 공유 파일 시스템에 있는 다른 파일의 기호 링크인 경우에는 예상할 수 없는 오류가 발생할 수 있습니다.

노드 구성 파일 작성

파티션된 데이터베이스 환경에서 데이터베이스를 작동시키려면 db2nodes.cfg라는 노드 구성 파일을 작성해야 합니다.

여러 데이터베이스 파티션에서 병렬 기능을 사용하여 데이터베이스 관리 프로그램을 시작하려면 인스턴스의 홈 디렉토리 아래에 있는 sqllib 서브디렉토리에 이 파일이 있어야 합니다. 이 파일에는 모든 데이터베이스 파티션에 대한 구성 정보가 포함되어 있으며 해당 인스턴스의 모든 데이터베이스 파티션에서 이 파일을 공유합니다.

Windows 고려사항

Windows에서 DB2 Enterprise Server Edition을 사용하는 경우 인스턴스를 작성하면 노드 구성 파일이 작성됩니다. 노드 구성 파일을 수동으로 작성 또는 수정하면 안됩니다. db2ncrt 명령을 사용하여 인스턴스에 데이터베이스 파티션 서버를 추가할 수 있습니다. db2ndrop 명령을 사용하여 인스턴스에서 데이터베이스 파티션 서버를 삭제할 수 있습니다. db2nchg 명령을 사용하여 하나의 컴퓨터에서 다른 컴퓨터로 데이터베이스 파티션 서버 수정, TCP/IP 호스트 이름 변경 또는 다른 논리적 포트 번호 또는 네트워크 이름 선택을 비롯하여 데이터베이스 파티션 서버 구성을 변경할 수 있습니다.

주: 인스턴스가 삭제되는 경우 데이터 손실을 예방하기 위해 데이터베이스 관리 프로그램에서 작성한 서브디렉토리 이외의 sqllib 서브디렉토리 아래에 파일 또는 디렉토리를 작성하면 안됩니다. 두 가지 예외가 있습니다. 시스템에서 스토어드 프로시저를 지원하는 경우 sqllib 서브디렉토리의 함수 서브디렉토리에 스토어드 프로시저 응용프로그램을 저장합니다. 다른 예외는 사용자 정의 함수(UDF)가 작성되는 경우입니다. UDF 실행 파일은 동일한 디렉토리에서 허용됩니다.

이 파일에는 인스턴스에 속한 각 데이터베이스 파티션에 대한 하나의 라인이 포함되어 있습니다. 각 라인의 형식은 다음과 같습니다.

dbpartitionnum hostname [*logical-port* [*netname*]]

토큰은 공백으로 구분됩니다. 변수는 다음과 같습니다.

dbpartitionnum

0에서 999 사이일 수 있는 데이터베이스 파티션 번호는 데이터베이스 파티션을 고유하게 정의합니다. 데이터베이스 파티션 번호는 오름차순이어야 합니다. 시퀀스에는 갭이 있을 수 있습니다.

데이터베이스 파티션 번호는 지정되면 변경할 수 없습니다. 그렇지 않으면 데이터가 분산되는 방법을 지정하는 분산 맵의 정보가 손상될 수 있습니다.

데이터베이스 파티션을 삭제하면 추가한 새 데이터베이스 파티션에 데이터베이스 파티션 번호를 사용할 수 있습니다.

데이터베이스 파티션 번호는 데이터베이스 디렉토리에 데이터베이스 파티션 이름을 생성하는 데 사용됩니다. 데이터베이스 파티션 번호의 형식은 다음과 같습니다.

NODE *nnnn*

*nnnn*은 왼쪽이 0으로 채워진 데이터베이스 파티션 번호입니다. 또한 데이터베이스 파티션 번호는 CREATE DATABASE 및 DROP DATABASE 명령에서 사용됩니다.

hostname

파티션 간 통신에 필요한 IP 주소의 호스트 이름입니다. 호스트 이름의 완전한 이름을 사용합니다. 또한 /etc/hosts 파일에서도 완전한 이름을 사용해야 합니다. db2nodes.cfg 파일 및 /etc/hosts 파일에 완전한 이름이 사용되지 않으면 오류 메시지 SQL30082N RC=3이 표시될 수 있습니다.

네트이름이 지정된 경우는 예외입니다. 이러한 경우 네트이름은 대부분의 통신에 사용되고 호스트 이름은 db2start, db2stop 및 db2_all에만 사용됩니다.

logical-port

이 매개변수는 선택적이고 데이터베이스 파티션의 논리적 포트 번호를 지정합니다. 이 번호는 데이터베이스 관리 프로그램 인스턴스 이름과 함께 사용되어 etc/services 파일의 TCP/IP 서비스 이름 항목을 식별합니다.

IP 주소와 논리적 포트의 조합은 잘 알려진 주소로 사용되고 데이터베이스 파티션 간에 통신 연결을 지원할 수 있도록 모든 응용프로그램에서 고유해야 합니다.

각 호스트 이름의 경우 *logical-port*는 0(영) 또는 공백이어야 합니다(디폴트값 : 0). *logical-port*와 연결된 데이터베이스 파티션은 클라이언트가 연결된 호스트의 디폴트 노드입니다. 이 값은 db2profile 스크립트의 **DB2NODE** 환경 변수 또는 sqlesetc() API로 겹쳐 쓸 수 있습니다.

netname

이 매개변수는 선택적이며 고유한 호스트 이름이 있는 하나 이상의 활성 TCP/IP 인터페이스가 있는 호스트를 지원하는 데 사용됩니다.

다음 예에서는 SP2EN1에 여러 TCP/IP 인터페이스와 두 개의 논리적 파티션이 있는 시스템에 대해 가능한 노드 구성 파일을 보여주고 SP2SW1을 DB2 데이터베이스 인터페이스로 사용합니다. 또한 1(0이 아님)에서 시작하고 *dbpartitionnum* 시퀀스에 갭이 있는 데이터베이스 파티션 번호를 보여줍니다.

표 7. 데이터베이스 파티션 번호 예 테이블

dbpartitionnum	*hostname*	*logical-port*	*netname*
1	SP2EN1.mach1.xxx.com	0	SP2SW1
2	SP2EN1.mach1.xxx.com	1	SP2SW1
4	SP2EN2.mach1.xxx.com	0	
5	SP2EN3.mach1.xxx.com		

원하는 편집기를 사용하여 `db2nodes.cfg` 파일을 갱신할 수 있습니다. (예외: 편집기는 Windows에서 사용하면 안됩니다.) 그러나 데이터베이스 파티션에서는 START DBM 발행 시 노드 구성 파일이 잠겨 있고 STOP DBM이 데이터베이스 관리 프로그램을 중지한 후에는 이 파일의 잠금이 해제되어야 하므로 파일에서 정보의 무결성을 보호하도록 주의해야 합니다. 필요한 경우 파일이 잠겨 있으면 START DBM 명령으로 파일을 갱신할 수 있습니다. 예를 들어 **RESTART** 옵션 또는 **ADD DBPARTITIONNUM** 옵션을 사용하여 START DBM을 발행할 수 있습니다.

주: STOP DBM 명령에 실패하고 노드 구성 파일을 잠금을 해제하지 못한 경우 STOP DBM **FORCE**를 발행하여 해당 파일의 잠금을 해제하십시오.

노드 및 데이터베이스 구성 파일 변경

데이터베이스 구성 파일을 갱신하려면 적절한 옵션과 함께 AUTOCONFIGURE 명령을 실행합니다.

구성 어드바이저는 수정할 구성 매개변수를 제안하고 해당 매개변수에 대해 제안된 값을 제공하여 성능을 조정하고 인스턴스당 단일 데이터베이스에 대한 메모리 요구사항의 균형을 맞추도록 지원합니다.

임의의 데이터베이스 파티션 그룹을 변경하려면(데이터베이스 파티션 추가, 삭제 또는 기존 데이터베이스 파티션 이동) 노드 구성 파일을 갱신해야 합니다. 데이터베이스 파일을 변경하지 않으려면 구성 매개변수의 값을 검토해야 합니다. 데이터베이스에 대해 진행 중인 변경(데이터베이스가 사용되는 방법에 따라 달라짐)의 일부로 일부 값을 주기적으로 조정할 수 있습니다.

주: 임의의 매개변수를 수정하면 다음과 같은 경우가 발생할 때까지 값이 갱신되지 않습니다.

- 데이터베이스 매개변수의 경우 모든 응용프로그램의 연결이 끊긴 후 데이터베이스에 대한 첫 번째 새 연결
- 데이터베이스 관리 프로그램 매개변수의 경우 인스턴스를 중지 및 시작한 다음

대부분의 경우 구성 어드바이저가 권장하는 값은 워크로드 및 자체 특정 서버에 대한 정보를 바탕으로 하므로 디폴트값보다 성능을 향상시킵니다. 그러나 반드시 최적화되지 않더라도 이러한 값은 데이터베이스 시스템의 성능을 향상시킵니다. 이러한 값을 최적화된 성능을 얻기 위해 추가로 조정할 수 있는 시작점으로 고려해 보십시오.

버전 9.1에서 데이터베이스를 작성하면 구성 어드바이저는 자동으로 호출됩니다. 이러한 기능을 사용하지 않거나 명시적으로 사용하려면 데이터베이스를 작성하기 전에 db2set 명령을 사용하십시오.
예:

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

디폴트로 사용 가능한 다른 기능은 25 페이지의 『자동 기능』의 내용을 참조하십시오.

명령행에서 구성 어드바이저를 사용하려면 AUTOCONFIGURE 명령을 사용하십시오.

명령행을 사용하여 데이터베이스 관리 프로그램 구성에서 개별 매개변수를 갱신하려면 다음과 같이 입력하십시오.

```
UPDATE DBM CFG USING <config_keyword>=<value>
```

명령행을 사용하여 데이터베이스 구성에서 개별 매개변수를 갱신하려면 다음과 같이 입력하십시오.

```
UPDATE DB CFG FOR <database_alias>
USING <config_keyword>=<value>
```

단일 명령에서 하나 이상의 <config_keyword>=<value> 조합을 갱신할 수 있습니다. 데이터베이스 관리 프로그램 구성 파일에 대한 대부분의 변경사항은 메모리로 로드된 후에만 유효하게 됩니다. 서버 구성 매개변수의 경우 START DATABASE MANAGER 명령을 실행하는 중 이러한 변경사항이 유효하게 됩니다. 클라이언트 매개변수의 경우 응용프로그램이 재시작되면 이러한 변경사항이 유효하게 됩니다.

현재 데이터베이스 관리 프로그램 구성 매개변수 목록을 보거나 인쇄하려면 GET DATABASE CONFIGURATION 명령을 사용하십시오.

클라이언트 응용프로그램에서 구성 어드바이저에 액세스하려면 db2AutoConfig API를 호출하십시오. 데이터베이스 관리 프로그램 구성에서 개별 매개변수를 갱신하거나 클라이언트 응용프로그램에서 데이터베이스 구성 파일을 갱신하려면 db2CfgSet API를 호출하십시오.

데이터베이스 복구 로그

*데이터베이스 복구 로그*에는 새 테이블 추가 또는 기존 테이블의 갱신사항을 포함하여 데이터베이스의 모든 변경사항에 대한 레코드가 보존됩니다.

이 로그는 여러 로그 *Extent*로 구성되며 각 로그 Extent는 로그 *파일*이라는 개별 파일에 들어 있습니다.

데이터베이스 복구 로그를 사용하여 시스템 전원 꺼짐이나 응용프로그램 오류와 같은 장애로 인해 데이터베이스가 불일치 상태로 남아 있지 않도록 할 수 있습니다. 장애가

발생하는 경우, 이미 작성되었으나 커미트되지는 않은 변경사항은 롤백되며 실제로 디스크에 기록되지는 않은 커미트된 모든 트랜잭션은 재실행됩니다. 이러한 조치는 데이터베이스의 무결성을 확인합니다.

데이터베이스 오브젝트의 스페이스 요구사항

데이터베이스 오브젝트 *크기* 계산은 정확하게 수행되지 않습니다. 사용할 수 있는 컬럼 유형 및 행 길이가 매우 다양하기 때문에 디스크 분할화로 인한 오버헤드, 여유 공간 및 다양한 길이의 컬럼 사용으로 *크기*를 계산하기가 어렵습니다.

데이터베이스 *크기*를 처음 계산한 후 테스트 데이터베이스를 작성하고 대표 데이터로 해당 데이터베이스를 채우십시오. 그런 다음 db2look 유틸리티를 사용하여 데이터베이스의 데이터 정의 명령문을 생성하십시오.

데이터베이스 *크기* 계산 시 다음 사항을 고려해야 합니다.

- 시스템 카탈로그 테이블
- 사용자 테이블 데이터
- Long 필드(LF) 데이터
- 대형 오브젝트(LOB) 데이터
- XML 데이터
- 인덱스 스페이스
- 로그 파일 스페이스
- 임시 작업 스페이스

또한 다음과 관련된 오버헤드 및 스페이스 요구사항을 고려하십시오.

- 로컬 데이터베이스 디렉토리 파일
- 시스템 데이터베이스 디렉토리 파일
- 다음 항목을 포함하는 운영 체제에 필요한 파일 관리 오버헤드:
  - 파일 블록 *크기*
  - 디렉토리 제어 스페이스

로그 파일 관련 스페이스 요구사항

로그 파일 관련 스페이스 요구사항은 사용자의 필요와 구성 매개변수 설정에 따라 다릅니다.

로그 제어 파일에는 56KB의 스페이스가 필요합니다. 또한 활성 로그 구성에 사용할 수 있는 최소한의 스페이스가 필요하며, 다음과 같이 *크기*를 계산할 수 있습니다.

(*logprimary* + *logsecond*) × (*logfilsiz* + 2) × 4096

각 항목에 대한 설명은 다음과 같습니다.

- logprimary는 데이터베이스 구성 파일에 정의된 1차 로그 파일 수입니다.
- logsecond는 데이터베이스 구성 파일에 정의된 2차 로그 파일 수입니다. 이 계산에서 logsecond를 -1로 설정할 수 없습니다. (logsecond를 -1로 설정하면 무한 활성 로그 스페이스가 요청됩니다.)
- logfilsiz는 데이터베이스 구성 파일에 정의된 각 로그 파일의 페이지 수입니다.
- 2는 각 로그 파일에 필요한 헤더 페이지 수입니다.
- 4096은 한 페이지의 바이트 수입니다.

롤 포워드 복구

데이터베이스에서 롤 포워드 복구가 사용 가능한 경우 특수 로그 스페이스 요구사항을 고려해야 합니다.

- **logarchmeth1** 구성 매개변수를 LOGRETAIN으로 설정하면 로그 파일이 log path 디렉토리에 아카이브됩니다. 로그 파일을 다른 위치로 이동하지 않는 한 온라인 디스크 스페이스가 채워집니다.
- **logarchmeth1** 구성 매개변수를 USEREXIT, DISK 또는 VENDOR로 설정하면 User Exit 프로그램이 아카이브된 로그 파일을 다른 위치로 이동시킵니다. 다음 항목에는 여전히 추가 로그 스페이스가 필요합니다.
  - User Exit 프로그램을 통한 이동 대기 중인 온라인 아카이브 로그
  - 나중에 사용하기 위해 형식화 중인 새 로그 파일

순환 로깅

데이터베이스에서 순환 로그가 사용 가능한 경우 이 공식의 결과 값은 로깅에 할당할 모든 스페이스가 됩니다. 즉, 추가 스페이스는 할당되지 않으며 로그 파일에 충분하지 않은 디스크 스페이스 오류가 수신되지 않습니다.

무한 로깅

데이터베이스에서 무한 로깅이 사용 가능한 경우(즉, **logsecond** 구성 매개변수를 -1로 설정한 경우) 아카이브 로깅을 사용하려면 **logarchmeth1** 구성 매개변수를 OFF 또는 logretain이 아닌 값으로 설정해야 합니다. 데이터베이스 관리 프로그램은 최소한 로그 경로에서 **logprimary** 구성 매개변수를 통해 지정된 활성 로그 파일 수를 유지하므로 위 공식에서 **logsecond** 구성 매개변수에 -1 값을 사용하지 말아야 합니다. 추가 디스크 스페이스를 제공하여 로그 파일 아카이브로 인해 발생하는 지연이 허용되도록 하십시오.

로그 경로 미러링

로그 경로를 미러링하는 경우 예상 로그 파일 스페이스 요구사항을 2배로 설정해야 합니다.

현재 커미트됨

쿼리가 데이터의 현재 커미트된 값을 리턴하는 경우, **cur_commit** 구성 매개변수가 DISABLED로 설정되지 않을 때 트랜잭션 중에 데이터 행의 첫 번째 갱신을 로그하기 위한 추가 로그 스페이스가 필요합니다. 워크로드 크기에 따

라서 사용되는 전체 로그 스페이스는 매우 다릅니다. 이는 주어진 워크로드에 필요한 로그 입출력, 필요한 활성 로그 스페이스 크기 및 필요한 로그 아카이브 스페이스 크기에 영향을 미칩니다.

주: cur_commit 구성 매개변수를 DISABLED로 설정하면 이전 릴리스에서와 동일한 동작을 유지하고 필요한 로그 스페이스는 변경되지 않습니다.

LDAP(Lightweight Directory Access Protocol) 디렉토리 서비스

디렉토리 서비스는 분산 환경에 있는 다중 시스템 및 서비스에 대한 자원 정보 저장소이며, 해당 자원에 대한 클라이언트 및 서버 액세스를 제공합니다.

클라이언트와 서버에서는 디렉토리 서비스를 사용하여 기타 자원에 액세스하는 방법을 찾습니다. 분산 환경에 있는 이들 기타 자원에 대한 정보는 디렉토리 서비스 저장소에 입력되어야 합니다.

*LDAP(Lightweight Directory Access Protocol)*은 디렉토리 서비스에 대한 산업 규격 액세스 메소드입니다. 각 데이터베이스 서버 인스턴스는 LDAP 서버에 존재 여부를 알리고 데이터베이스가 작성되는 경우 LDAP 디렉토리에 데이터베이스 정보를 제공합니다. 클라이언트가 데이터베이스에 연결되면 서버의 카탈로그 정보를 LDAP 디렉토리에서 검색할 수 있습니다. 각 컴퓨터에서 로컬로 카탈로그 정보를 저장하는 데 각각의 클라이언트가 더 이상 필요하지 않습니다. 클라이언트 응용프로그램은 LDAP 디렉토리에서 데이터베이스에 연결하는 데 필요한 정보를 검색합니다.

주: LDAP 서버에 데이터베이스 서버 인스턴스를 발행하는 것은 자동 프로세스가 아니므로 관리자가 수동으로 완료해야 합니다.

DB2 시스템의 관리자는 디렉토리 서비스를 설정하고 유지보수할 수 있습니다. 구성 지원 프로그램이 이 디렉토리 서비스 유지보수를 지원할 수 있습니다. LDAP(Lightweight Directory Access Protocol) 디렉토리 서비스를 통해 DB2 데이터베이스 관리 프로그램에서 디렉토리 서비스를 사용할 수 있습니다. LDAP 디렉토리 서비스를 사용하려면 디렉토리 정보를 저장할 수 있도록 DB2 데이터베이스 관리 프로그램이 지원하는 LDAP 서버가 있어야 합니다.

주: Windows 도메인 환경에서 실행 중인 경우에는 LDAP 서버가 Windows Active Directory와 통합되어 있기 때문에 이미 LDAP 서버가 사용 가능합니다. 따라서 Windows를 실행 중인 모든 컴퓨터에서 LDAP을 사용할 수 있습니다.

LDAP 디렉토리는 클라이언트 수가 많기 때문에 각 클라이언트 컴퓨터에서 로컬 디렉토리 카탈로그를 갱신하기가 어려운 엔터프라이즈 환경에서 유용합니다. 이런 상황에서는 한 위치(LDAP 서버)에서 카탈로그 항목을 유지보수할 수 있도록 LDAP 서버에 디렉토리 항목을 저장해야 합니다.

데이터베이스 작성

CREATE DATABASE 명령을 사용하여 데이터베이스를 작성합니다. 클라이언트 응용프로그램에서 데이터베이스를 작성하려면 sqlecrea API를 호출합니다.

데이터베이스를 작성하기 전에 컨텐츠, 레이아웃, 잠재적 성장 및 사용될 방법을 염두에 두고 데이터베이스를 계획하는 것이 중요합니다. 작성되고 데이터로 채워진 후에 변경할 수 있습니다. 그러나 처음에 데이터베이스를 설정한 방법에 따라서 변경이 수행되는 동안 더 많은 노력이 필요하고 데이터를 사용할 수 없게 만들 수 있습니다.

다음 데이터베이스 특권은 시스템 카탈로그 뷰의 PUBLIC인 CREATETAB, BINDADD, CONNECT, IMPLICIT_SCHEMA 및 SELECT에 자동으로 부여됩니다. 그러나 RESTRICTIVE 옵션이 있으면 PUBLIC에 자동으로 부여되는 특권이 없습니다. RESTRICTIVE 옵션에 대한 자세한 정보는 CREATE DATABASE 명령을 참조하십시오.

데이터베이스를 작성하는 경우 다음 태스크 중 하나가 수행됩니다.

- 데이터베이스에 필요한 모든 시스템 카탈로그 테이블 설정
- 데이터베이스 복구 로그 할당
- 데이터베이스 구성 파일 작성 및 디폴트값 설정
- 데이터베이스로 데이터베이스 유틸리티 바인드

명령행 처리기를 사용하여 데이터베이스를 작성하려면 다음과 같이 입력하십시오.

```
CREATE DATABASE <database name>
```

예를 들어 다음 명령은 디폴트 위치에 ″Personnel DB for BSchiefer Co″라는 연관 주석이 있는 PERSON1이라는 데이터베이스를 작성합니다.

```
CREATE DATABASE personl
  WITH "Personnel DB for BSchiefer Co"
```

구성 어드바이저

구성 어드바이저는 수정할 구성 매개변수를 제안하고 해당 매개변수에 대해 제안된 값을 제공하여 성능을 조정하고 인스턴스당 단일 데이터베이스에 대한 메모리 요구사항의 균형을 맞추도록 지원합니다. 구성 어드바이저는 데이터베이스 작성 시 자동으로 호출됩니다. 이 기능을 사용하지 않으려거나 명시적으로 사용하려면 데이터베이스를 작성하기 전에 db2set 명령을 사용합니다.

예:

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

디폴트로 사용 가능한 다른 기능은 25 페이지의 『자동 기능』의 내용을 참조하십시오.

이벤트 모니터

데이터베이스가 작성되면 동시에 자세한 교착 상태 이벤트 모니터가 작성됩니다. 모니터 사용 시, 이 이벤트 모니터와 연관된 일부 오버헤드가 있습니다. 자세한 교착 상태 이벤트 모니터를 원하지 않는 경우 다음 명령을 사용하여 이벤트 모니터를 삭제할 수 있습니다.

```
DROP EVENT MONITOR db2detaildeadlock
```

이 이벤트 모니터에서 사용하는 디스크 스페이스의 양을 제한하기 위해 이벤트 모니터가 비활성화되고 모니터가 최대 출력 파일 수에 접근하면 관리 통지 로그에 메시지가 기록됩니다. 더 이상 필요하지 않은 출력 파일을 제거하면 이벤트 모니터는 다음에 데이터베이스를 활성화할 때 다시 활성화됩니다.

리모트 데이터베이스

다른 인스턴스, 가능하면 리모트 인스턴스에서 데이터베이스를 작성할 수 있습니다. 다른 (리모트) 데이터베이스 파티션 서버에서 데이터베이스를 작성하려면, 우선 해당 서버에 접속해야 합니다. 다음 명령 처리 중 임시로 데이터베이스 연결이 설정됩니다.

```
CREATE DATABASE <database name> AT DBPARTITIONNUM <options>
```

이러한 유형의 환경에서 디폴트 인스턴스 이외의 인스턴스(리모트 인스턴스 포함)에 대해 인스턴스 레벨 관리를 수행할 수 있습니다. 이렇게 수행하는 방법에 대한 지시사항은 db2iupdt(인스턴스 갱신) 명령을 참조하십시오.

데이터베이스 코드 페이지

데이터베이스는 UTF-8(유니코드) 코드 세트에서 디폴트로 작성됩니다.

데이터베이스의 기본 코드 페이지를 겹쳐쓰려면 데이터베이스를 작성할 때 원하는 코드 세트 및 지역을 지정해야 합니다. 코드 세트 및 지역 설정에 대한 자세한 정보는 CREATE DATABASE 명령 또는 sqlecrea API를 참조하십시오.

자동 스토리지 데이터베이스

스토리지 관리를 간편히 하기 위해 자동 스토리지를 사용합니다. 명시적 컨테이너 정의를 사용하여 테이블 스페이스 레벨에서 스토리지를 관리하지 않고 데이터베이스 레벨에서 스토리지를 관리하며 컨테이너를 작성, 확장 및 추가하는 기능은 데이터베이스 관리 프로그램이 수행합니다.

사용자가 별도로 지정하지 않으면 모든 데이터베이스는 자동 스토리지로 작성됩니다. 자동 스토리지를 갖는 데이터베이스를 작성할 때 하나 이상의 초기 스토리지 경로를 설정합니다. 대조적으로, 자동 스토리지가 없는 데이터베이스를 작성할 때는 스토리지 경로를 데이터베이스 전체와 연관시키지 않습니다. 대신 스토리지가 사용자가 작성하는 개별 시스템 관리 또는 데이터베이스 관리(SMS 또는 DMS) 테이블 스페이스와 연관됩

니다. 자동 스토리지 데이터베이스가 커지면 데이터베이스 관리 프로그램은 해당 스토리지 경로 사이에 컨테이너를 작성하고 이들을 확장하거나 필요한 대로 새 컨테이너를 자동으로 작성합니다.

ALTER DATABASE문의 ADD STORAGE ON절을 사용하여 기존 데이터베이스(자동 스토리지를 갖고 작성되지 않은 데이터베이스 포함)를 수정하여 자동 스토리지를 사용할 수 있습니다. 이 명령문을 사용하면 데이터베이스에 새 스토리지 경로가 추가될 뿐 아니라 달리 지정하지 않는 한 데이터베이스에 추가되는 모든 새 테이블 스페이스가 자동 스토리지 테이블 스페이스가 되는 두 가지 효과가 있습니다.

중요사항:

- 스토리지 경로를 추가해도 기존의 자동이 아닌 스토리지 테이블 스페이스가 자동 스토리지를 사용하도록 변환되지 않습니다. 데이터베이스 관리(DMS) 테이블 스페이스를 자동 스토리지를 사용하도록 변환할 수 있습니다. 시스템 관리(SMS) 테이블 스페이스는 자동 스토리지로 변환될 수 없습니다. 자세한 정보는 163 페이지의 『자동 스토리지를 사용하도록 테이블 스페이스 변환』의 내용을 참조하십시오.
- 데이터베이스에서 자동 스토리지가 사용 가능하도록 설정되면 이를 사용 불가능하게 설정할 수 없습니다.

데이터베이스에서 자동 스토리지를 사용하지 않으려면 CREATE DATABASE 명령에 명시적으로 AUTOMATIC STORAGE NO절을 지정해야 합니다. 예를 들어,

```
CREATE DATABASE ASNODB1 AUTOMATIC STORAGE NO
```

스토리지 경로 목록을 데이터베이스 스냅샷의 파트로 표시할 수 있습니다(BUFFERPOOL 모니터 스위치가 켜진 경우 파일 시스템 정보와 함께 표시).

자동 스토리지 데이터베이스 작성

모든 데이터베이스는 사용자가 별도로 지정하지 않으면 자동 스토리지 데이터베이스로 작성됩니다. 자동 스토리지를 갖는 데이터베이스를 작성할 때 하나 이상의 초기 스토리지 경로를 설정합니다. 데이터베이스가 확장됨에 따라 데이터베이스 관리 프로그램이 해당 스토리지 경로에 컨테이너 작성, 확장 및 추가합니다.

시작하기 전에

DB2 데이터베이스가 실행 중이어야 합니다. 데이터베이스 관리 프로그램을 시작하려면 db2start를 사용하십시오.

이 태스크에 대한 정보

자동 스토리지 데이터베이스를 작성할 때 하나 이상의 스토리지 경로를 자동 스토리지 테이블 스페이스에 의해 사용되는 데이터베이스와 연관시킵니다. 다른 유형의 테이블 스페이스와 비교할 때 자동 스토리지 테이블 스페이스가 수행해야 하는 유지보수 태스크를 줄여줍니다.

제한사항

- 스토리지 경로는 상대 경로 이름을 사용하여 지정할 수 없습니다. 절대 경로 이름을 사용해야 합니다. 스토리지 경로의 길이는 175자까지 가능합니다.
- Windows 운영 체제에서, **DB2_CREATE_DB_ON_PATH** 레지스트리 변수가 YES로 설정되지 않으면 데이터베이스 경로는 드라이브 이름만이어야 합니다.
- CREATE DATABASE 명령의 **DBPATH ON** 절을 사용하여 데이터베이스 경로를 지정하지 않는 경우, 데이터베이스 관리 프로그램은 데이터베이스 경로에 대해 **ON** 절에 지정되는 첫 번째 스토리지 경로를 사용합니다. (Windows 운영 체제에서, 이 절이 경로로 지정되고 **DB2_CREATE_DB_ON_PATH** 레지스트리 변수가 YES로 설정되지 않는 경우 SQL1052N 오류 메시지가 수신됩니다.) **ON**절이 지정되지 않는 경우 데이터베이스는 데이터베이스 관리 프로그램 구성 파일(dftdbpath 매개변수)에서 지정되는 디폴트 데이터베이스 경로에 작성됩니다. 이것은 데이터베이스와 연관된 단일 스토리지 경로의 위치로도 사용됩니다.
- 파티션된 데이터베이스의 경우 데이터베이스 파티션 표현식을 사용하지 않으면 각 데이터베이스 파티션에서 동일한 스토리지 경로 세트를 사용해야 합니다.
- CREATE DATABASE 명령의 **DBPATH ON**절을 사용하여 명시적으로 지정하거나 첫 번째 스토리지 경로의 데이터베이스 파티션 표현식을 사용하여 내재적으로 지정하는지 여부에 관계없이 데이터베이스 경로에서 데이터베이스 파티션 표현식은 유효하지 않습니다.
- 자동 스토리지를 사용하여 작성된 경우 데이터베이스에 대해 자동 스토리지를 사용하지 않을 수 없습니다.
- 자동 스토리지 데이터베이스에는 해당 데이터베이스에 연결된 스토리지 경로가 하나 이상 있어야 합니다.

프로시저

자동 스토리지를 갖는 데이터베이스를 작성하려면 다음을 수행하십시오.

1. CREATE DATABASE 명령을 공식화하십시오. 사용자가 별도로 지정하지 않는 한, 디폴트로 새 데이터베이스는 자동 스토리지 데이터베이스로서 작성됩니다. 또한 CREATE DATABASE 명령에 AUTOMATIC STORAGE YES절을 포함할 수도 있습니다. 예를 들어, 다음 두 가지는 서로 동등합니다.

   ```
   CREATE DATABASE DATAB1
   CREATE DATABASE DATAB1 AUTOMATIC STORAGE YES
   ```

2. CREATE DATABASE 명령을 실행하십시오.

예

예 1: UNIX 또는 Linux 운영 체제에서 자동 스토리지 데이터베이스 작성:

/DATA1 및 /DATA2를 스토리지 경로로 사용하여 경로 /DPATH1에 TESTDB1이라는 데이터베이스를 작성하려면 다음 명령을 사용하십시오.

```
CREATE DATABASE TESTDB1 ON '/DATA1','/DATA2' DBPATH ON '/DPATH1'
```

예 2: Windows 운영 체제에서 스토리지 및 데이터베이스 경로를 둘 다 지정하여 자동 스토리지 데이터베이스 작성:

E:₩DATA의 스토리지를 사용하여 D: 드라이브에 TESTDB2라는 데이터베이스를 작성하려면 다음 명령을 사용하십시오.

```
CREATE DATABASE TESTDB2 ON 'E:₩DATA' DBPATH ON 'D:'
```

예 3: Windows 운영 체제에서 스토리지 경로만 지정하여 자동 스토리지 데이터베이스 작성:

F: 드라이브에 스토리지를 갖는 TESTDB3이라는 데이터베이스를 작성하려면 다음 명령을 사용하십시오.

```
CREATE DATABASE TESTDB3 AUTOMATIC STORAGE YES ON 'F:'
```

이 예에서 F:가 스토리지 경로 및 데이터베이스 경로로 사용됩니다.

스토리지 경로에 대해 F:₩DATA 같은 디렉토리 이름을 지정하는 경우 명령은 실패합니다. 다음과 같은 이유 때문입니다.

1. DBPATH가 지정되지 않을 때 스토리지 경로(이 경우에는 F:₩DATA)가 데이터베이스 경로로 사용됩니다.
2. Windows에서, 데이터베이스 경로는 드라이브 이름만 가능합니다 (**DB2_CREATE_DB_ON_PATH** 레지스트리 변수에 대한 디폴트를 NO에서 YES로 변경하지 않은 경우).

Windows 운영 체제에서 디렉토리를 스토리지 경로로 지정하려는 경우 예 2에서 보는 것처럼 DBPATH ON drive절도 포함해야 합니다.

예 4: UNIX 또는 Linux 운영 체제에서 데이터베이스 경로를 지정하지 않고 자동 스토리지 데이터베이스 작성:

/DATA1 및 /DATA2에 TESTDB4라는 데이터베이스를 작성하려면 다음 명령을 사용하십시오.

```
CREATE DATABASE TESTDB4 ON '/DATA1','/DATA2'
```

이 예에서 /DATA1 및 /DATA2가 스토리지 경로로 사용되고 /DATA1이 데이터베이스 경로입니다.

자동 스토리지 데이터베이스를 작성한 후에 CREATE TABLESPACE 명령을 사용하여 테이블, 인덱스 및 기타 데이터베이스 오브젝트를 저장할 자동 스토리지 테이블 스페이스를 작성할 수 있습니다.

자동 스토리지를 사용하도록 자동이 아닌 스토리지 데이터베이스 변환

ALTER DATABASE문을 사용하여 데이터베이스에 새 스토리지 경로를 추가하여 기존 비자동 스토리지 데이터베이스를 자동 스토리지를 사용하도록 변환할 수 있습니다.

자동 스토리지 테이블 스페이스에 대한 스토리지 경로로 사용할 수 있는 경로(Windows 운영 체제의 경우 경로 또는 드라이브 이름)를 사용하여 식별할 수 있는 스토리지 위치가 있어야 합니다.

이 태스크에 대한 정보

자동 스토리지를 사용하지 않는 데이터베이스는 연관된 *스토리지 경로*가 없습니다. 대신 스토리지가 데이터베이스에 대한 테이블 스페이스와 연관됩니다. 자동 스토리지가 이전에 사용 가능하지 않았던 데이터베이스에 새 스토리지 경로를 추가할 때 데이터베이스는 *자동 스토리지 데이터베이스*가 됩니다. 그러나 새 스토리지 경로를 데이터베이스에 추가하면 데이터베이스는 자동 스토리지만 사용할 수 있습니다. 사용자가 작성하는 추가 테이블 스페이스는 자동 스토리지를 사용하지만 기존 테이블 스페이스는 자동으로 변환되지 않습니다. 기존 테이블 스페이스를 자동 스토리지를 사용하도록 변환하려면 ALTER TABLESPACE문을 사용해야 합니다.

주: DMS 테이블 스페이스만 자동 스토리지를 사용하도록 변환할 수 있습니다.

제한사항

데이터베이스가 자동 스토리지를 갖고 작성되거나 자동 스토리지를 사용하도록 변환된 경우 데이터베이스에 대해 자동 스토리지를 사용하지 않을 수 없습니다.

프로시저

기존 데이터베이스를 자동 스토리지 데이터베이스로 변환하려면 ALTER DATABASE 문을 사용하여 스토리지 경로를 추가하십시오.

1. ADD STORAGE ON절을 사용하여 ALTER DATABASE문을 공식화하십시오. 예를 들어 DATABASE1 데이터베이스를 자동 스토리지를 사용하도록 변환하려면 다음 명령문을 사용하십시오.

   ```
   ALTER DATABASE DATABASE1 ADD STORAGE ON storagePath
   ```

   여기서 *storagePath*는 자동 스토리지 테이블 스페이스에 사용하려는 경로입니다.
2. 명령문을 실행하십시오.

예

예 1: UNIX 또는 Linux 운영 체제에서 데이터베이스 변환

데이터베이스 EMPLOYEE가 비자동 스토리지 데이터베이스이고 /data1/as 및 /data2/as가 자동 스토리지 테이블 스페이스에 사용하려는 경로라고 가정하십시오. EMPLOYEE를 자동 스토리지 데이터베이스로 변환하려면 다음 명령문을 사용하십시오.

```
ALTER DATABASE EMPLOYEE ADD STORAGE ON '/data1/as', '/data2/as'
```

예 2: Windows 운영 체제에서 데이터베이스 변환

데이터베이스 SALES가 비자동 스토리지 데이터베이스이고 F:₩DB2DATA 및 G:가 자동 스토리지 테이블 스페이스에 사용하려는 경로라고 가정하십시오. SALES를 자동 스토리지 데이터베이스로 변환하려면 다음 명령문을 사용하십시오.

```
ALTER DATABASE EMPLOYEE ADD STORAGE ON 'F:₩DB2DATA', 'G:'
```

다음 단계

자동 스토리지를 사용하도록 변환하려는 기존 DMS 테이블 스페이스가 있는 경우 MANAGED BY AUTOMATIC STORAGE절을 갖는 ALTER TABLESPACE문을 사용하여 변경하십시오.

자동 스토리지에 사용 가능한 데이터베이스에 스토리지 경로 추가

ALTER DATABASE문을 사용하여 자동 스토리지 데이터베이스에 스토리지 경로를 추가할 수 있습니다. 데이터베이스가 현재 자동 스토리지 데이터베이스가 아닌 경우 데이터베이스에 스토리지 경로를 추가하면 자동 스토리지 데이터베이스로 변환됩니다.

다중 파티션 데이터베이스 환경에 스토리지 경로를 추가하려면 각 데이터베이스 파티션에 스토리지 경로가 있어야 합니다. 모든 데이터베이스 파티션에 지정된 경로가 없으면 명령문이 롤백됩니다.

기존 데이터베이스에 스토리지 경로를 추가하려면 다음 ALTER DATABASE문을 발행하십시오.

```
ALTER DATABASE ADD STORAGE ON storage-path
```

데이터베이스에 하나 이상의 스토리지 경로를 추가한 후 ALTER TABLESPACE문을 선택적으로 사용하여 새 스토리지 경로를 즉시 사용할 수 있도록 데이터베이스에서 테이블 스페이스를 재조정할 수 있습니다. 그렇지 않으면 기존 스토리지 경로의 컨테이너 내에 증가할 여유 공간이 없을 때까지 새 스토리지 경로가 사용되지 않습니다.

자동 스토리지에 사용 가능한 데이터베이스에서 스토리지 경로 삭제

자동 스토리지 데이터베이스에서 하나 이상의 스토리지 경로를 제거하거나 스토리지 경로 외부로 데이터를 이동하고 이러한 경로에 대한 재조정을 수행할 수 있습니다.

시작하기 전에

스냅샷 모니터를 사용하여 데이터베이스 파티션 상태를 비롯하여 스토리지 경로에 대한 현재 정보를 표시합니다. 스토리지 경로의 상태는 다음 세 가지 중 하나일 수 있습니다.

사용되지 않음
: 스토리지 경로가 데이터베이스에 추가되었으나 테이블 스페이스에서 사용되지 않습니다.

사용 중
: 하나 이상의 테이블 스페이스가 스토리지 경로에 컨테이너를 보유하고 있습니다.

삭제 보류
: 경로 삭제를 위해 ALTER DATABASE database-name DROP STORAGE ON이 요청되었으나 테이블 스페이스에서 스토리지 경로를 계속 사용 중입니다. 경로를 사용하는 테이블 스페이스가 더 이상 없는 경우 데이터베이스에서 해당 경로가 제거됩니다.

또한 관리 뷰를 사용하여 갱신된 스토리지 경로 또는 테이블 스페이스 파티션에 대한 정보를 얻을 수도 있습니다. 관리 뷰를 사용하여 스토리지 경로 및 SNAPTBSP_PART 관리 뷰에 대한 정보를 얻고 특정 데이터베이스 파티션에 있는 테이블 스페이스에 대한 정보를 얻을 수 있습니다.

이 태스크에 대한 정보

스토리지 경로를 삭제하려면 경로가 삭제되도록 데이터를 경로 외부로 이동하는 ALTER TABLESPACE *tablespace-name* REBALANCE를 사용하여 스토리지 경로를 사용하는 모든 영구 테이블 스페이스에 대한 재조정을 수행해야 합니다. 이러한 경우 재조정 조작은 삭제하려는 스토리지 경로에서 나머지 스토리지 경로로 데이터를 이동하여 이러한 스토리지 경로에서 데이터가 일관성 있게 스프라이프되도록 하여 입출력 병렬 처리를 극대화합니다.

프로시저

1. 아래 테이블에 표시된 것처럼 ALTER DATABASE문을 사용하여 데이터베이스에서 스토리지 경로가 제거되도록 데이터베이스를 변경합니다.
2. 아래 예에 표시된 것처럼 ALTER TABLESPACE *tablespace-name* REBALANCE 문을 사용하여 삭제할 스토리지 경로를 벗어난 컨테이너에 대한 재조정을 수행합니다.
3. 임시 테이블 스페이스 삭제 및 다시 작성합니다. 해당 경로에 임시 테이블 스페이스가 있으면 삭제 보류 상태인 테이블 스페이스는 삭제되지 않습니다.

예

이 예에서는 현재 연결된 데이터베이스에서 스토리지 경로 /db2/filesystem1 및 /db2/filesystem2를 삭제하고 해당 테이블 스페이스에 대해 재조정을 수행하는 방법을 보여줍니다.

먼저 다음과 같이 ALTER문을 발행하여 데이터베이스에서 스토리지 경로를 삭제합니다.

```
ALTER DATABASE DROP STORAGE ON '/db2/filesystem1', '/db2/filesystem2'
```

그런 후 다음과 같이 이러한 스토리지 경로를 사용하는 모든 테이블 스페이스에 대해 ALTER TABLESPACE *tablespace-name* REBALANCE문을 발행하여 해당 스토리지 경로에서 컨테이너를 제거합니다.

```
ALTER TABLESPACE tablespace-name_1 REBALANCE
ALTER TABLESPACE tablespace-name_2 REBALANCE
ALTER TABLESPACE tablespace-name_n REBALANCE
```

마지막 재조정 조작이 완료되면 /db2/filesystem1 및 /db2/filesystem2가 데이터베이스에서 제거됩니다.

다음 단계

데이터베이스 스냅샷을 확인하거나 현재 관리 뷰를 쿼리하여 삭제된 스토리지 경로가 더 이상 나열되지 않는지 확인하십시오. 나열되면 해당 스토리지 경로를 사용하는 하나 이상의 테이블 스페이스가 있는 것입니다.

스토리지 경로 모니터링

데이터베이스 스냅샷에는 데이터베이스와 연관된 스토리지 경로 목록이 포함되어 있습니다.

자동 스토리지 경로 수가 0이면 데이터베이스에 자동 스토리지를 사용할 수 없습니다.

```
자동 스토리지 경로 수            = ##
자동 스토리지 경로 수             = <1st path>
자동 스토리지 경로 수             = <2nd path>
...
```

버퍼 풀 모니터 스위치가 설정되어 있으면 다음 요소도 설정됩니다.

```
파일 시스템                       = 12345
파일 시스템 여유 공간(바이트)           = 20000000000
파일 시스템 사용한 스페이스(바이트)      = 40000000000000
파일 시스템 총 스페이스(바이트)          = 40020000000000
```

이 데이터는 경로에 기초하여 설정됩니다. 단일 데이터베이스 파티션 시스템에서는 경로를 기초로, 다중 데이터베이스 파티션 환경에서는 각 데이터베이스 파티션을 기초로 설정됩니다.

또한 테이블 스페이스 스냅샷 내에서 다음 정보가 설정됩니다. 이 정보는 테이블 스페이스가 자동 스토리지 테이블 스페이스로 작성되었는지 여부를 나타냅니다.

```
Using automatic storage            = Yes or No
```

데이터베이스 리스토어의 영향

RESTORE DATABASE 명령은 백업 이미지에서 데이터베이스를 리스토어하는 데 사용됩니다.

리스토어 조작 중 데이터베이스 경로의 위치를 선택할 수 있고 데이터베이스와 연관된 스토리지 경로를 다시 정의할 수 있습니다. 데이터베이스 경로 및 스토리지 경로는 RESTORE DATABASE 명령과 TO, ON 및 DBPATH ON의 조합을 사용하여 설정됩니다.

예를 들어 자동 스토리지가 사용 가능하도록 설정된 데이터베이스에 대해 유효한, 다음과 같은 RESTORE 명령이 몇 가지 있습니다.

```
RESTORE DATABASE TEST1
RESTORE DATABASE TEST2 TO X:
RESTORE DATABASE TEST3 DBPATH ON D:
RESTORE DATABASE TEST3 ON /path1, /path2, /path3
RESTORE DATABASE TEST4 ON E:₩newpath1, F:₩newpath2 DBPATH ON D:
```

CREATE DATABASE 명령과 같이 데이터베이스 관리 프로그램은 스토리지 위치와 관련된 다음 두 가지 정보를 추출합니다.

- **데이터베이스 경로**(데이터베이스 관리 프로그램이 데이터베이스에 대한 다양한 제어 파일을 저장하는 위치)
  - TO 또는 DBPATH ON이 지정되어 있으면 이 값은 데이터베이스 경로를 나타냅니다.
  - ON이 사용되지만 이 값과 함께 DBPATH ON이 지정되어 있지 않으면 ON과 함께 나열된 첫 번째 경로가 데이터베이스 경로로 사용됩니다. 또한 이 경로는 스토리지 경로입니다.
  - TO, ON 또는 DBPATH ON 중 아무 것도 지정되지 않으면 *dftdbpath* 데이터베이스 관리 프로그램 구성 매개변수가 데이터베이스 경로를 판별합니다.

  주: 디스크에 이름이 동일한 데이터베이스가 있으면 데이터베이스 경로가 무시되고 해당 데이터베이스는 기존 데이터베이스와 동일한 위치에 저장됩니다.
- **스토리지 경로**(데이터베이스 관리 프로그램이 자동 스토리지 테이블 스페이스 컨테이너를 작성하는 위치)
  - ON이 지정되어 있으면 나열된 모든 경로가 스토리지 경로로 간주되고 백업 이미지 내에 저장된 경로 대신 이러한 경로가 사용됩니다.
  - ON이 지정되어 있지 않으면 스토리지 경로가 변경되지 않습니다. 백업 이미지 내에 저장된 스토리지 경로가 유지됩니다.

이러한 개념을 보다 명확히 설명하기 위해 다음 표에는 위에서 제공된 5개의 동일한 RESTORE 명령의 예가 해당 스토리지 경로와 함께 표시됩니다.

표 8. 데이터베이스 및 스토리지 경로와 관련된 리스토어의 의미

RESTORE DATABASE 명령	디스크에 이름이 동일한 데이터베이스가 없음		디스크에 이름이 동일한 데이터베이스가 있음	
	데이터베이스 경로	스토리지 경로	데이터베이스 경로	스토리지 경로
RESTORE DATABASE TEST1	*<dftdbpath>*	백업 이미지에서 정의된 스토리지 경로 사용	기존 데이터베이스의 데이터베이스 경로 사용	백업 이미지에서 정의된 스토리지 경로 사용
RESTORE DATABASE TEST2 TO X:	X:	백업 이미지에서 정의된 스토리지 경로 사용	기존 데이터베이스의 데이터베이스 경로 사용	백업 이미지에서 정의된 스토리지 경로 사용
RESTORE DATABASE TEST3 DBPATH ON /db2/databases	/db2/databases	백업 이미지에서 정의된 스토리지 경로 사용	기존 데이터베이스의 데이터베이스 경로 사용	백업 이미지에서 정의된 스토리지 경로 사용
RESTORE DATABASE TEST4 ON /path1, /path2, /path3	/path1	/path1, /path2, /path3	기존 데이터베이스의 데이터베이스 경로 사용	/path1, /path2, /path3
RESTORE DATABASE TEST5 ON E:¥newpath1, F:¥newpath2 DBPATH ON D:	D:	E:¥newpath1, F:¥newpath2	기존 데이터베이스의 데이터베이스 경로 사용	E:¥newpath1, F:¥newpath2

스토리지 경로가 리스토어 조작의 일부로 다시 정의된 경우 자동 스토리지를 사용하도록 정의된 테이블 스페이스의 경로가 새 경로로 자동으로 재지정됩니다. 그러나 SET TABLESPACE CONTAINERS 명령을 사용하여 자동 스토리지 테이블 스페이스와 연관된 컨테이너의 경로를 명시적으로 재지정할 수 없습니다. 이 조치는 허용되지 않습니다.

db2ckbkp 명령의 **-s** 옵션을 사용하여 백업 이미지 내에서 자동 스토리지를 데이터베이스에 사용할 수 있는지 여부를 표시합니다. 자동 스토리지가 사용 가능한 경우 데이터베이스와 연관된 스토리지 경로가 표시됩니다.

multi-partition 자동 스토리지 사용 가능 데이터베이스의 경우 RESTORE DATABASE 명령에는 몇 개의 추가 의미가 포함되어 있습니다.

1. 데이터베이스에서는 모든 데이터베이스 파티션에서 동일한 스토리지 경로 세트를 사용해야 합니다.
2. 카탈로그 데이터베이스 파티션에서만 새로운 스토리지 경로와 함께 RESTORE 명령을 발행할 수 있습니다. 이 명령은 모든 비카탈로그 데이터베이스 파티션에서 데이터베이스의 상태를 RESTORE_PENDING으로 설정합니다.

표 9. 다중 파티션 데이터베이스의 리스토어 의미

RESTORE DATABASE 명령	데이터베이스 파티션에서 발행 #	디스크에 이름이 동일한 데이터베이스가 없음		디스크에 이름이 동일한 데이터베이스가 있음(스켈레톤 데이터베이스 포함)	
		다른 데이터베이스 파티션의 결과	스토리지 경로	다른 데이터베이스 파티션의 결과	스토리지 경로
RESTORE DATABASE TEST1	카탈로그 데이터베이스 파티션	스켈레톤 데이터베이스는 카탈로그 데이터베이스 파티션에서 백업 이미지의 스토리지 경로를 사용하여 작성됩니다. 다른 모든 데이터베이스 파티션은 RESTORE_ PENDING 상태로 저장됩니다.	백업 이미지에서 정의된 스토리지 경로 사용	변경되지 않음. 스토리지 경로가 변경되지 않으므로 다른 데이터베이스 파티션에서 변경사항이 발생하지 않습니다.	백업 이미지에서 정의된 스토리지 경로 사용
	비카탈로그 데이터베이스 파티션	SQL2542N 또는 SQL2551N이 리턴됩니다. 데이터베이스가 없으면 카탈로그 데이터베이스 파티션을 먼저 리스토어해야 합니다.	N/A	변경되지 않음. 스토리지 경로가 변경되지 않으므로 다른 데이터베이스 파티션에서 변경사항이 발생하지 않습니다.	백업 이미지에서 정의된 스토리지 경로 사용
RESTORE DATABASE TEST2 ON /path1, /path2, /path3	카탈로그 데이터베이스 파티션	스켈레톤 데이터베이스는 RESTORE 명령에 지정된 스토리지 경로를 사용하여 작성됩니다. 다른 모든 데이터베이스 파티션은 RESTORE_ PENDING 상태로 저장됩니다.	/path1, /path2, /path3		/path1, /path2, /path3
	비카탈로그 데이터베이스 파티션	SQL1174N이 리턴됩니다. 데이터베이스가 없으면 카탈로그 데이터베이스 파티션을 먼저 리스토어해야 합니다.비카탈로그 데이터베이스 파티션의 RESTORE에 대해 스토리지 경로를 지정할 수 없습니다.	N/A	SQL1172N이 리턴됩니다. 비카탈로그 데이터베이스 파티션의 RESTORE에 대해 새 스토리지 경로를 지정할 수 없습니다.	N/A

데이터베이스 카탈로그

새 데이터베이스를 작성하면 해당 데이터베이스가 시스템 데이터베이스 디렉토리 파일에서 자동으로 카탈로그됩니다. 또한 CATALOG DATABASE 명령을 사용하여 동일한 시스템 데이터베이스 디렉토리 파일에 있는 데이터베이스를 명시적으로 카탈로그할 수 있습니다.

CATALOG DATABASE 명령을 사용하여 별명 이름이 다른 데이터베이스를 카탈로그하거나 UNCATALOG DATABASE 명령을 사용하여 이전에 삭제된 데이터베이스 항목을 카탈로그할 수 있습니다.

데이터베이스 작성 시 데이터베이스가 자동으로 카탈로그되더라도 해당 데이터베이스를 카탈로그해야 할 수 있습니다. 이렇게 하는 경우 데이터베이스가 있어야 합니다.

디폴트로 데이터베이스 디렉토리를 비롯한 디렉토리 파일은 디렉토리 캐시 지원(*dir_cache*) 구성 매개변수를 사용하여 메모리에 캐시됩니다. 디렉토리 캐싱이 사용 가능한 경우 예를 들어, 다른 응용프로그램에서 CATALOG DATABASE 또는 UNCATALOG DATABASE 명령을 사용하여 디렉토리에 대해 수행한 변경사항은 사용자의 응용프로그램이 재시작될 때까지 적용되지 않을 수 있습니다. 명령행 처리기 세션에서 사용되는 디렉토리 캐시를 새로 고치려면 TERMINATE 명령을 발생하십시오.

파티션된 데이터베이스에서 디렉토리 파일의 캐시는 각 데이터베이스 파티션에서 작성됩니다.

내부 데이터베이스 관리 프로그램 찾아보기에 응용프로그램 레벨 캐시 이외에도 데이터베이스 관리 프로그램 레벨 캐시를 사용할 수 있습니다. 이러한 『공유』 캐시를 새로 고치려면 db2stop 및 db2start 명령을 발행하십시오.

명령행 처리기를 사용하여 별명 이름이 다른 데이터베이스를 카탈로그하려면 CATALOG DATABASE 명령을 사용하십시오. 예를 들어 다음 명령행 처리기 명령은 PERSON1 데이터베이스를 HUMANRES로 카탈로그합니다.

```
CATALOG DATABASE personl AS humanres
  WITH "Human Resources Database"
```

여기서 시스템 데이터베이스 디렉토리 항목의 데이터베이스 별명은 HUMANRES입니다. 데이터베이스 별명은 데이터베이스 이름(PERSON1)과 다릅니다.

클라이언트 응용프로그램에서 시스템 데이터베이스 디렉토리의 데이터베이스를 카탈로그하려면 sqlecadb API를 호출하십시오.

명령행 처리기를 사용하여 디폴트 이외의 인스턴스에서 데이터베이스를 카탈로그하려면 CATALOG DATABASE 명령을 사용하십시오. 다음 예에서 데이터베이스 B에 대한 연결은 INSTNC_C에 대한 연결입니다. 인스턴스 instnc_c는 이 명령을 실행하기 전에 로컬 노드로 이미 카탈로그되어 있어야 합니다.

```
CATALOG DATABASE b as b_on_ic AT NODE instnc_c
```

주: 또한 데이터베이스 서버 컴퓨터에 상주한 데이터베이스를 카탈로그하는 데 클라이언트 노드에서 CATALOG DATABASE 명령이 사용됩니다.

데이터베이스로 유틸리티 바인드

데이터베이스가 작성되면 데이터베이스 관리 프로그램에서 db2ubind.lst 및 db2cli.lst의 유틸리티를 데이터베이스로 바인드합니다. 이러한 파일은 sqllib 디렉토리의 bnd 서브디렉토리에 저장됩니다.

유틸리티를 바인드하면 단일 소스 파일에서 특정 SQL 및 XQuery문을 처리하는 데 필요한 모든 정보가 포함된 오브젝트인 *패키지*가 작성됩니다.

주: 클라이언트에서 이러한 유틸리티를 사용하려면 해당 유틸리티를 명시적으로 바인드해야 합니다. 샘플 데이터베이스에서 패키지를 작성하려면 이러한 파일이 상주한 디렉토리에 있어야 합니다. 바인드 파일은 sqllib 디렉토리의 bnd 서브디렉토리에 있습니다. 클라이언트에서 데이터베이스를 작성 또는 업그레이드하는 경우 db2schema.bnd 파일도 바인드해야 합니다. 자세한 내용은 "DB2 CLI 바인드 파일 및 패키지 이름"을 참조하십시오.

유틸리티를 데이터베이스로 바인드하거나 리바인드하려면 다음 명령을 호출하십시오. 여기서 sample은 데이터베이스의 이름입니다.

```
connect to sample
bind @db2ubind.lst
```

데이터베이스 별명 작성

*별명*은 테이블, 별칭 또는 뷰를 참조하는 간접적인 방법입니다. 따라서 SQL 또는 XQuery문은 해당 테이블 또는 뷰의 규정 이름과 별개일 수 있습니다.

테이블 또는 뷰 이름이 변경되는 경우에만 별명 정의를 변경해야 합니다. 다른 별명에 대해 별명을 작성할 수도 있습니다. 기존 테이블 또는 뷰 이름을 참조할 수 있는 테이블 점검 제한 조건 정의를 제외한 보기 또는 트리거 정의와 SQL 또는 XQuery문에서 별명을 사용할 수 있습니다.

정의 시 존재하지 않은 테이블, 뷰 또는 별명에 대해서도 별명을 정의할 수 있습니다. 그러나 별명이 포함된 SQL 또는 XQuery문을 컴파일할 때에는 별명이 있어야 합니다.

별명 이름은 기존 테이블 이름을 사용할 수 있으면 항상 사용할 수 있고 별명 체인을 따라 순환적으로 또는 반복적으로 참조하지 않는 경우에는 다른 별명을 참조할 수 있습니다.

별명 이름은 기존 테이블, 뷰 또는 별명과 동일할 수 없으며 동일한 데이터베이스 내의 테이블만 참조할 수 있습니다. CREATE TABLE 또는 CREATE VIEW문에서 사용되는 테이블 또는 뷰 이름은 동일한 스키마의 별칭 이름과 동일할 수 없습니다.

별칭이 현재 권한 부여 ID에 따라 소유한 스키마 이외의 스키마에 없으면 별칭을 작성하는 데 특수 권한이 필요하지 않습니다. 이러한 경우 DBADM 권한이 필요합니다.

별명 또는 별명이 참조하는 오브젝트가 삭제된 경우 별칭에 대해 종속적인 모든 패키지는 유효하지 않은 것으로 표시되고 별명에 대해 종속적인 모든 뷰 및 트리거는 작동 불능으로 표시됩니다.

명령행을 사용하여 별명을 작성하려면 다음과 같이 입력하십시오.

```
CREATE ALIAS <alias_name> FOR <table_name>
```

별명은 명령문 컴파일 시 테이블 또는 뷰 이름으로 교체됩니다. 별명 또는 별명 체인을 테이블 또는 뷰 이름으로 분석할 수 없으면 오류가 표시됩니다. 예를 들어 WORKERS가 EMPLOYEE의 별명이면 컴파일 시

```
SELECT * FROM WORKERS
```

가 다음에 적용됩니다.

```
SELECT * FROM EMPLOYEE
```

다음 SQL문은 EMPLOYEE 테이블에 별명 WORKERS를 작성합니다.

```
CREATE ALIAS WORKERS FOR EMPLOYEE
```

주: OS/390 또는 z/Series용 DB2에서는 두 개의 구별되는 별명 개념인 ALIAS 및 SYNONYM을 채택합니다. 이러한 두 개념은 다음과 같은 점에서 DB2 데이터베이스와 다릅니다.

- OS/390 또는 z/Series용 DB2의 ALIAS:
  - 특수 권한 및 특권이 있는 작성자 필요
  - 다른 별명을 참조할 수 없음
- OS/390 또는 z/Series용 DB2의 SYNONYM:
  - 작성자만 사용할 수 있음
  - 항상 규정되지 않음
  - 참조된 테이블이 삭제되면 삭제됨
  - 테이블 또는 뷰와 이름 스페이스를 공유하지 않음

분산 관계형 데이터베이스에 연결

분산 관계형 데이터베이스는 정규 요청자-서버 프로토콜 및 함수를 기반으로 빌드됩니다.

*응용프로그램 리퀘스터(AR)*는 연결의 응용프로그램 측을 지원합니다. 응용프로그램 리퀘스터는 응용프로그램의 데이터베이스 요청을 분산 데이터베이스 네트워크에서 사용하기에 적합한 통신 프로토콜로 변환합니다. 이러한 요청은 연결의 다른 쪽 끝에 있는 *데이터베이스 서버*를 통해 수신되고 처리됩니다. 응용프로그램이 로컬 데이터베이스에 액세스 중인 것처럼 작동할 수 있도록 응용프로그램 리퀘스터(AR)와 데이터베이스 서버가 함께 작동하여 통신 및 위치 고려사항을 처리합니다.

테이블 또는 뷰를 참조하는 SQL문을 실행하려면 먼저 응용프로그램 프로세스를 데이터베이스 관리 프로그램의 응용프로그램 서버(AS)에 연결해야 합니다. CONNECT문은 응용프로그램 프로세스와 해당 서버 사이를 연결합니다.

CONNECT문에는 두 가지 유형이 있습니다.

- CONNECT(유형 1)는 작업 단위(UOW)(리모트 작업 단위(RUOW)) 시맨틱당 단일 데이터베이스를 지원합니다.
- CONNECT(유형 2)는 작업 단위(UOW)(응용프로그램 지향 분산 작업 단위(DUOW)) 시맨틱당 다중 데이터베이스를 지원합니다.

DB2 콜 레벨 인터페이스(CLI) 및 Embedded SQL은 *동시 트랜잭션*이라는 연결 모드를 지원하여 다중 연결이 가능하도록 하며 각각의 트랜잭션은 독립적인 트랜잭션입니다. 응용프로그램은 동일한 데이터베이스에 대한 여러 개의 동시 연결을 사용할 수 있습니다.

응용프로그램 서버(AS)는 프로세스가 시작된 환경에서 로컬이거나 리모트입니다. 해당 환경에서 분산 관계형 데이터베이스를 사용하고 있지 않더라도 응용프로그램 서버(AS)가 존재합니다. 이러한 환경에는 CONNECT문에서 식별할 수 있는 응용프로그램 서버(AS)에 대해 설명하는 로컬 디렉토리가 포함됩니다.

응용프로그램 서버(AS)는 테이블 또는 뷰를 참조하는 바운드 양식의 정적 SQL문을 실행합니다. 데이터베이스 관리 프로그램에서 바인드 조작을 통해 이전에 작성한 패키지에서 바운드 명령문을 가져옵니다.

대부분의 경우 응용프로그램 서버(AS)에 연결된 응용프로그램에서는 응용프로그램 서버(AS)의 데이터베이스 관리 프로그램이 지원하는 명령문 및 절을 사용할 수 있습니다. 해당 명령문 및 절의 일부를 지원하지 않는 데이터베이스 관리 프로그램의 응용프로그램 리퀘스터(AR)를 통해 응용프로그램을 실행 중인 경우에도 이들 명령문 및 절을 사용할 수 있습니다.

분산 관계형 데이터베이스의 리모트 작업 단위(RUOW)

*리모트 작업 단위(RUOW) 기능*은 SQL문을 리모트로 준비하고 실행하는 기능을 제공합니다.

컴퓨터 시스템 『A』의 응용프로그램 프로세스를 컴퓨터 시스템 『B』의 응용프로그램 서버에 연결할 수 있으며, 하나 이상의 작업 단위(UOW)에서 『B』의 오브젝트를 참조하는 임의의 수의 정적 또는 동적 SQL문을 실행할 수 있습니다. B에서 작업 단위(UOW)를 종료한 후, 응용프로그램 프로세스가 컴퓨터 시스템 C의 응용프로그램 서버에 연결하고 또 마찬가지로 수행할 수 있습니다.

대부분의 SQL문을 리모트로 준비하여 실행할 수 있으며 다음과 같은 제한사항이 있습니다.

- 단일 SQL문에서 참조되는 모든 오브젝트를 동일한 응용프로그램 서버(AS)로 관리해야 합니다.
- 하나의 작업 단위(UOW)에 있는 모든 SQL문을 동일한 응용프로그램 서버(AS)에서 실행해야 합니다.

임의의 지정된 시간에 응용프로그램 프로세스는 네 개의 가능한 *연결 상태* 중 하나의 상태에 있습니다.

- 연결 가능하고 연결되어 있음

  응용프로그램 프로세스가 응용프로그램 서버(AS)에 연결되고 CONNECT문을 실행할 수 있습니다.

  내재적 연결이 사용 가능한 경우 다음과 같이 수행됩니다.

  – 연결 가능하며 연결되지 않은 상태에서 CONNECT TO문 또는 피연산자가 없는 CONNECT문이 정상적으로 실행되면 응용프로그램 프로세스가 이 상태로 변경됩니다.

  – CONNECT RESET, DISCONNECT, SET CONNECTION 또는 RELEASE 이외의 SQL문이 실행되는 경우 응용프로그램 프로세스가 내재적으로 연결 가능한 상태에서 이 상태로 변경됩니다.

  내재적 연결이 사용 가능한지 여부에 관계 없이 다음과 같은 경우에 이 상태로 변경됩니다.

  – 연결 가능하며 연결되지 않은 상태에서 CONNECT TO문이 정상적으로 실행되는 경우.

  – COMMIT 또는 ROLLBACK문이 정상적으로 실행되거나 연결 가능하며 연결된 상태에서 강제 롤백이 발생하는 경우.

- 연결 불가능하지만 연결되어 있음

  응용프로그램 프로세스가 응용프로그램 서버(AS)에 연결되지만 응용프로그램 서버(AS)를 변경하도록 CONNECT TO문을 정상적으로 실행할 수 없습니다. CONNECT TO, 피연산자가 없는 CONNECT, CONNECT RESET, DISCONNECT, SET CONNECTION, RELEASE, COMMIT 또는 ROLLBACK 이외의 SQL문을 실행하는 경우 응용프로그램 프로세스가 연결 가능하며 연결된 상태에서 이 상태로 변경됩니다.

- 연결 가능하지만 연결되어 있지 않음

  응용프로그램 프로세스가 응용프로그램 서버(AS)에 연결되지 않습니다. 실행할 수 있는 SQL문은 CONNECT TO뿐입니다. 다른 SQL문을 실행하면 오류(SQLSTATE 08003)가 발생합니다.

  내재적 연결이 사용 가능한지 여부에 관계 없이 CONNECT TO문 실행 시 오류가 발생하거나 작업 단위(UOW) 내에서 오류가 발생하여 연결이 끊어지고 롤백이 발생하는 경우 응용프로그램 프로세스가 이 상태로 변경됩니다. 응용프로그램 프로세스가 연결 가능한 상태에 있지 않거나 서버 이름이 로컬 디렉토리에 있지 않아서 발생하는 오류로 인해 트랜잭션이 이 상태로 변경되지는 않습니다.

내재적 연결이 사용 불가능한 경우 다음과 같은 결과가 발생합니다.

- 응용프로그램 프로세스가 초기부터 이 상태에 있습니다.
- CONNECT RESET 및 DISCONNECT문으로 인해 트랜잭션이 이 상태로 변경됩니다.

• 내재적으로 연결 가능(내재적 연결이 사용 가능한 경우).

내재적 연결이 사용 가능한 경우 이 상태가 응용프로그램 프로세스의 초기 상태입니다. CONNECT RESET문을 실행하면 트랜잭션이 이 상태로 변경됩니다. 연결 불가능하지만 연결된 상태에서 COMMIT 또는 ROLLBACK문을 실행한 후 연결 가능하고 연결된 상태에서 DISCONNECT문을 실행해도 이 상태가 됩니다.

내재적 연결 사용 가능성은 설치 옵션, 환경 변수 및 인증 설정에 따라 판별됩니다.

CONNECT 자체는 연결 가능한 상태에서 응용프로그램 프로세스를 제거하지 않기 때문에 CONNECT문을 연속으로 실행하는 것은 오류가 아닙니다. 그러나 연속으로 CONNECT RESET문을 실행하는 것은 오류입니다. CONNECT TO, CONNECT RESET, 피연산자가 없는 CONNECT, SET CONNECTION, RELEASE, COMMIT 또는 ROLLBACK 이외의 SQL문을 실행한 다음 CONNECT TO문을 실행하는 것도 오류입니다. 이러한 오류가 발생하지 않도록 하려면 CONNECT RESET, DISCONNECT(COMMIT 또는 ROLLBACK문 다음에 실행), COMMIT 또는 ROLLBACK문을 CONNECT TO문 이전에 실행해야 합니다.

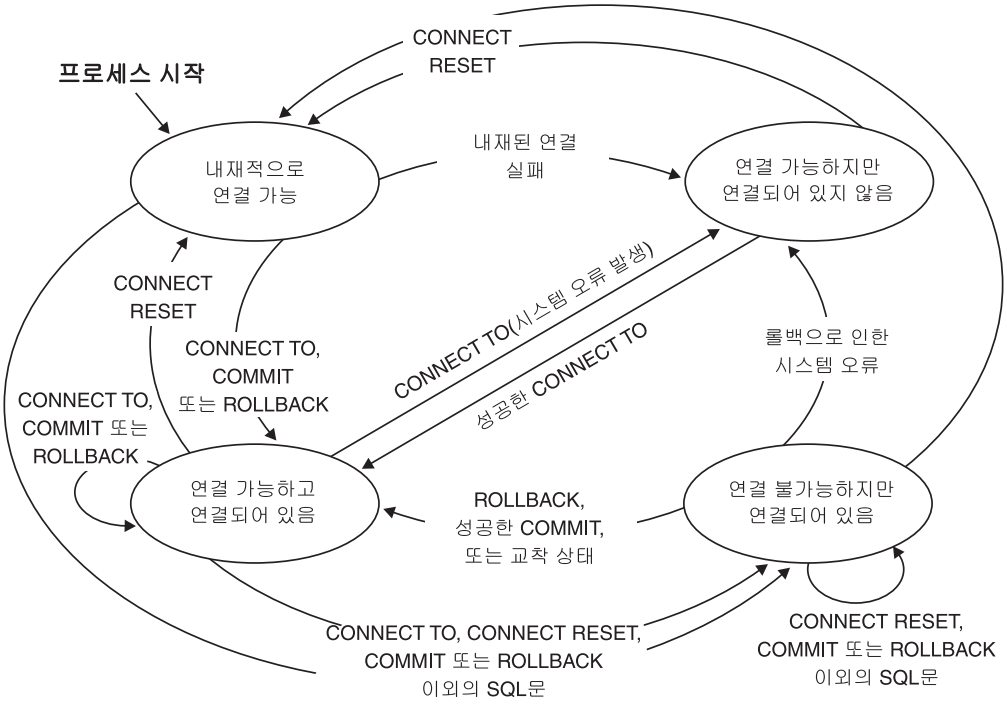


그림 4. 내재적 연결이 사용 가능한 경우 연결 상태 전이

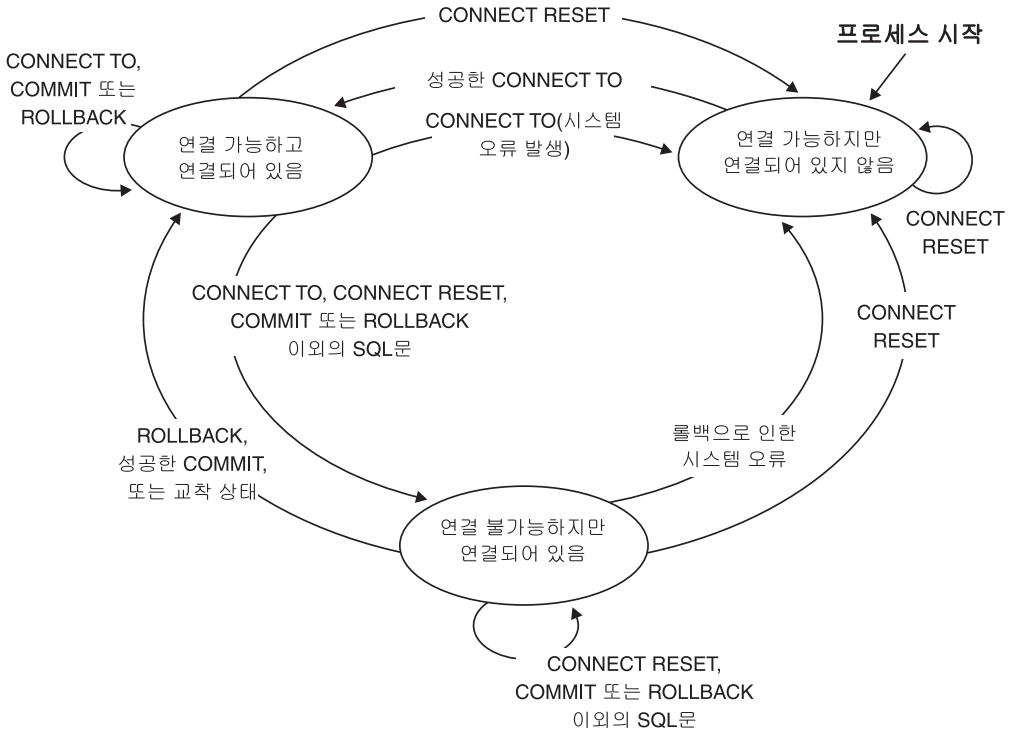


그림 5. 내재적 연결이 사용 불가능한 경우 연결 상태 전이

응용프로그램 지향 분산 작업 단위(DUOW)

*응용프로그램 지향 작업 단위(RUOW) 기능*은 SQL문을 리모트로 준비하고 실행하는 기능을 제공합니다.

CONNECT 또는 SET CONNECTION문을 실행하여 컴퓨터 시스템『A』의 응용프로그램 프로세스를 컴퓨터 시스템『B』의 응용프로그램 서버(AS)에 연결할 수 있습니다. 그런 다음 작업 단위(UOW)가 종료되기 전에 응용프로그램 프로세스가『B』의 오브젝트를 참조하는 임의의 수의 정적 및 동적 SQL문을 실행할 수 있습니다. 단일 SQL문에서 참조되는 모든 오브젝트를 동일한 응용프로그램 서버(AS)로 관리해야 합니다. 그러나 리모트 작업 단위(RUOW) 기능과는 달리 동일한 작업 단위(UOW)에 참여할 수 있는 응용프로그램 서버(AS) 수에 제한이 없습니다. 커미트 또는 롤백 조작은 작업 단위(UOW)를 종료합니다.

응용프로그램 지향 분산 작업 단위(DUOW)에서는 유형 2 연결을 사용합니다. *유형 2* 연결은 응용프로그램 프로세스를 식별된 응용프로그램 서버(AS)에 연결하고 응용프로그램 지향 분산 작업 단위(DUOW)에 적용할 규칙을 설정합니다.

유형 2 응용프로그램 프로세스 특성은 다음과 같습니다.

- 항상 연결 가능

- 연결된 상태에 있거나 연결되지 않은 상태에 있음
- 영(0)개 이상의 연결이 있음

각 응용프로그램 프로세스 연결은 연결에 사용되는 응용프로그램 서버(AS)의 데이터베이스 별명으로 식별됩니다.

개별 연결은 항상 다음 연결 상태 중 하나를 갖습니다.

- 현재 및 보류
- 현재 및 릴리스 보류
- 유휴 및 보류
- 유휴 및 릴리스 보류

유형 2 응용프로그램 프로세스는 초기에 연결되지 않은 상태이며 연결이 없습니다. 연결은 초기에는 현재 및 보류 상태입니다.

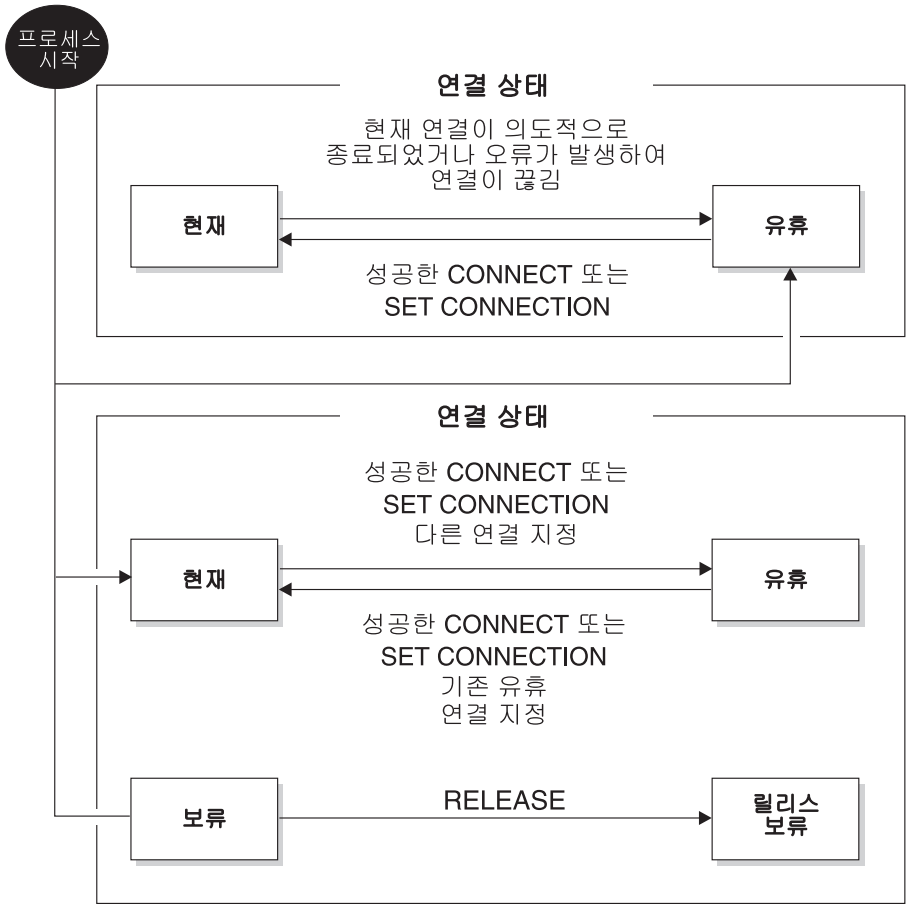


그림 6. 응용프로그램 지향 분산 작업 단위(DUOW) 연결 상태 전이

응용프로그램 프로세스 연결 상태

CONNECT문 실행에 적용되는 특정 규칙이 있습니다.

다음과 같은 규칙이 CONNECT문 실행에 적용됩니다.

- 컨텍스트에서는 동시에 동일한 응용프로그램 서버(AS)에 둘 이상을 연결할 수 없습니다.
- 응용프로그램 프로세스에서 SET CONNECTION문을 실행하는 경우 지정된 위치 이름은 응용프로그램 프로세스의 연결 세트에 있는 기존 연결이어야 합니다.
- 응용프로그램 프로세스에서 CONNECT문을 실행하고 SQLRULES(STD) 옵션이 적용되는 경우 지정된 서버 이름은 응용프로그램 프로세스의 연결 세트에 있는 기존 연결이 *아니어야* 합니다. SQLRULES 옵션에 대한 설명은 123 페이지의 『작업 단위(UOW) 시맨틱 제어 옵션』의 내용을 참조하십시오.

응용프로그램 프로세스에 현재 연결이 있는 경우 응용프로그램 프로세스는 *연결된* 상태입니다. CURRENT SERVER 특수 레지스터에는 현재 연결에 사용되는 응용프로그램 서버(AS) 이름이 들어 있습니다. 응용프로그램 프로세스는 해당 응용프로그램 서버(AS)에서 관리되는 오브젝트를 참조하는 SQL문을 실행할 수 있습니다.

연결되지 않은 상태에 있는 응용프로그램 프로세스에서 CONNECT 또는 SET CONNECTION문이 정상적으로 실행되면 해당 프로세스 상태가 연결된 상태로 변경됩니다. 연결이 없지만 SQL문이 실행되는 경우 디폴트 데이터베이스 이름으로 DB2DBDFT 환경 변수가 설정되었으면 내재적 연결이 작성됩니다.

응용프로그램 프로세스에 현재 연결이 없는 경우 응용프로그램 프로세스는 *연결되지 않은* 상태입니다. 실행할 수 있는 SQL문은 CONNECT, DISCONNECT ALL, DISCONNECT(데이터베이스 지정), SET CONNECTION, RELEASE, COMMIT, ROLLBACK 및 로컬 SET문뿐입니다.

*연결된 상태*에 있는 응용프로그램 프로세스의 현재 연결이 강제로 종료되거나 SQL문이 실패하면 *연결되지 않은 상태*로 변경되어 응용프로그램 서버(AS)에서 롤백 조작이 실행되고 연결이 끊어집니다. DISCONNECT문이 정상적으로 실행되거나 연결이 릴리스 보류 상태일 때 COMMIT문을 실행하면 연결이 강제로 종료됩니다. (DISCONNECT 프리컴파일러 옵션이 AUTOMATIC으로 설정되면 모든 연결이 종료됩니다. 이 옵션이 CONDITIONAL로 설정되면 열린 WITH HOLD 커서가 없는 모든 연결이 종료됩니다.)

연결 상태

『보류 및 릴리스 보류 상태』와 『현재 및 유휴 상태』의 두 가지 유형의 연결 상태가 있습니다.

응용프로그램 프로세스에서 CONNECT문을 실행하며 서버 이름이 응용프로그램 리퀘스터(AR)에 알려져 있지만 응용프로그램 프로세스의 기존 연결 세트에 있지 않은 경우 : (i) 현재 연결이 *유휴 연결 상태*가 되고 서버 이름이 연결 세트에 추가되며 새 연결이 *현재 연결 상태* 및 *보류 연결 상태* 두 가지로 설정됩니다.

서버 이름이 이미 응용프로그램 프로세스의 기존 연결 세트에 있으며 SQLRULES(STD) 옵션을 사용하여 응용프로그램을 프리컴파일한 경우에는 오류(SQLSTATE 08002)가 발생합니다.

보류 및 릴리스 보류 상태 RELEASE문은 연결이 보류 상태인지 또는 릴리스 보류 상태인지 여부를 제어합니다. *릴리스 보류* 상태는 다음 번 커미트 조작에 성공할 때 연결이 끊어짐을 의미합니다. (롤백은 연결에 영향을 주지 않습니다.) *보류* 상태는 다음 번 커미트 조작 시 연결이 끊어지지 않음을 의미합니다.

모든 연결은 초기에는 보류 상태이며 RELEASE문을 사용하여 릴리스 보류 상태로 이동할 수 있습니다. 릴리스 보류 상태가 되면 연결을 보류 상태로 되돌릴 수 없습니다. ROLLBACK문이 실행되거나 커미트 조작에 실패하여 롤백 조작이 발생하는 경우 작업 단위(UOW) 간에 연결이 릴리스 보류 상태로 남습니다.

연결을 릴리스하도록 명시적으로 표시되어 있지 않아도 커미트 조작이 DISCONNECT 프리컴파일러 옵션의 조건을 충족하면 커미트 조작을 통해 연결이 끊어집니다.

현재 및 유휴 상태 연결이 보류 상태인지 또는 릴리스 보류 상태인지 여부에 관계 없이 현재 상태 또는 유휴 상태일 수도 있습니다. *현재* 상태의 연결은 이 상태에서 SQL 문을 실행하는 데 사용 중인 연결입니다. *유휴* 상태의 연결은 현재 상태가 아닌 연결입니다.

유휴 연결에서 작동할 수 있는 SQL문은 COMMIT, ROLLBACK, DISCONNECT 또는 RELEASE뿐입니다. SET CONNECTION 및 CONNECT문은 지정된 서버의 연결 상태를 현재로 변경하고 다른 기존 연결은 유휴 상태로 남거나 변경됩니다. 어느 시점에서든 하나의 연결만 현재 상태로 있을 수 있습니다. 유휴 연결이 동일한 작업 단위(UOW)에서 현재 연결이 되는 경우 모든 잠금, 커서 및 준비된 명령문의 상태는 마지막으로 연결이 현재 상태였을 때와 동일합니다.

연결 종료 시

연결이 종료되면 연결을 통해 응용프로그램 프로세스가 획득한 모든 자원 및 연결을 작성하고 유지하는 데 사용된 모든 자원의 할당이 해제됩니다. 예를 들어, 응용프로그램 프로세스에서 RELEASE문을 실행하는 경우 다음 커미트 조작 시 연결이 종료되면 열려 있는 모든 커서가 닫힙니다.

통신 장애로 인해 연결이 종료될 수도 있습니다. 해당 연결이 현재 상태인 경우 응용프로그램 프로세스는 연결되지 않은 상태가 됩니다.

프로세스가 종료되면 응용프로그램 프로세스의 모든 연결이 종료됩니다.

작업 단위(UOW) 시맨틱 제어 옵션

유형 2 연결 관리의 시맨틱은 프리컴파일러 옵션 세트에 따라 판별됩니다. 아래에는 이러한 옵션(디폴트값은 밑줄이 그어진 굵은체 텍스트로 표시됨)이 요약되어 있습니다.

- CONNECT (**1** | 2). CONNECT문이 유형 1 또는 유형 2로 처리되는지 여부를 지정합니다.
- SQLRULES (**DB2** | STD). 유형 2 CONNECT가 DB2 규칙(CONNECT가 유휴 연결로 전환되도록 허용) 또는 SQL92 표준 규칙(CONNECT가 유휴 연결로 전환되도록 허용하지 않음)에 따라 처리되는지 여부를 지정합니다.
- DISCONNECT (**EXPLICIT** | CONDITIONAL | AUTOMATIC). 커미트 조작이 발생하는 경우 연결이 끊기는 다음 데이터베이스 연결을 지정합니다.
  - SQL RELEASE문으로 릴리스에 대해 명시적으로 표시된 연결(EXPLICIT)
  - 열려 있는 WITH HOLD 커서가 없는 연결 및 릴리스에 대해 표시된 연결(CONDITIONAL)
  - 모든 연결(AUTOMATIC)
- SYNCPOINT (**ONEPHASE** | TWOPHASE | NONE). COMMIT 또는 ROLLBACK이 여러 데이터베이스 연결 사이에 조정되는 방식을 지정합니다. 이 옵션은 무시되지만 이전 버전과의 호환성을 위해서만 포함됩니다.
  - 갱신은 작업 단위(UOW)의 하나의 데이터베이스에 대해서만 발생할 수 있으며 다른 모든 데이터베이스는 읽기 전용(ONEPHASE)입니다. 다른 데이터베이스에 대해 갱신을 시도하면 항상 오류가 발생합니다(SQLSTATE 25000).
  - TM(트랜잭션 관리 프로그램)은 런타임 시 이러한 프로토콜을 지원하는 데이터베이스 간에 2단계 COMMIT을 조정하는 데 사용됩니다(TWOPHASE).
  - TM을 사용하여 2단계 COMMIT을 수행하지 마십시오. 또한 단일 갱신자, 다중 갱신자를 실행하지 마십시오. COMMIT 또는 ROLLBACK문이 실행되면 개별 COMMIT 또는 ROLLBACK이 모든 데이터베이스에 게시됩니다. 하나 이상의 ROLLBACK에 실패하면 오류(SQLSTATE 58005)가 발생합니다. 하나 이상의 COMMIT에 실패하면 다른 오류(SQLSTATE 40003)가 발생합니다.

런타임 시 위의 옵션을 겹쳐쓰려면 SET CLIENT 명령 또는 sqlesetc API를 사용합니다. QUERY CLIENT 명령 또는 sqleqryc API를 사용하여 현재 설정을 얻을 수 있습니다. 이는 SQL문이 아니고 다양한 호스트 언어 및 명령행 처리기(CLP)에서 정의된 API입니다.

데이터 표현 고려사항

시스템이 다르면 데이터도 다르게 나타납니다. 하나의 시스템에서 다른 시스템으로 데이터가 이동되면 때때로 데이터 변환이 수행되어야 합니다.

DRDA®를 지원하는 제품은 수신 시스템에서 필수 변환을 자동으로 수행합니다.

숫자 데이터 변환을 수행하려면 시스템에서 전송 시스템이 데이터를 표현하는 방식 및 데이터 유형을 알아야 합니다. 문자열을 변환하려면 추가 정보가 필요합니다. 문자열 변환은 데이터의 코드 페이지와 해당 데이터에서 수행되는 조작에 따라 달라집니다. 문자열 변환은 IBM CDRA(Character Data Representation Architecture)에 따라 수행됩니다. 문자열 변환에 대한 자세한 정보는 *CDRA(Character Data Representation Architecture): 참조 및 레지스트리*(SC09-2190-00) 매뉴얼을 참조하십시오.

로컬 또는 시스템 데이터베이스 디렉토리 파일 보기

LIST DATABASE DIRECTORY 명령을 사용하여 시스템에 있는 데이터베이스와 연관된 정보를 봅니다.

로컬 또는 시스템 데이터베이스 디렉토리 파일을 보기 전에 먼저 인스턴스 및 데이터베이스를 작성해야 합니다.

로컬 데이터베이스 디렉토리 파일의 컨텐츠를 보려면 다음 명령을 발행하십시오. 여기서 <location>은 데이터베이스의 위치를 지정합니다.

```
LIST DATABASE DIRECTORY ON <location>
```

시스템 데이터베이스 디렉토리 파일의 컨텐츠를 보려면 데이터베이스 디렉토리 파일의 위치를 지정하지 않고 LIST DATABASE DIRECTORY 명령을 발행하십시오.

데이터베이스 삭제

데이터베이스 삭제는 오브젝트, 컨테이너 및 연관 파일을 모두 삭제하므로 광범위하게 영향을 미칩니다. 삭제된 데이터베이스는 데이터베이스 디렉토리에서 제거(카탈로그 해제)됩니다.

명령행을 사용하여 데이터베이스를 삭제하려면 다음과 같이 입력하십시오.

```
DROP DATABASE <name>
```

다음 명령은 SAMPLE 데이터베이스를 삭제합니다.

```
DROP DATABASE SAMPLE
```

주: SAMPLE 데이터베이스를 삭제했으나 다시 필요한 경우 다시 작성할 수 있습니다.

클라이언트 응용프로그램에서 데이터베이스를 삭제하려면 sqledrpd API를 호출하십시오. 지정한 데이터베이스 파티션 서버에서 데이터베이스를 삭제하려면 sqledpan API를 호출하십시오.

별명 삭제

별명을 삭제하면 해당 설명이 카탈로그에서 삭제되고 별명을 참조하는 모든 패키지 및 캐시된 동적 쿼리가 무효화되며 해당 별명에 종속된 모든 뷰 및 트리거가 작동 불능 상태로 표시됩니다.

명령을 삭제하려면 다음과 같이 명령행에서 DROP문을 발행하십시오.

```
DROP ALIAS EMPLOYEE-ALIAS
```

제 6 장 데이터베이스 파티션

*데이터베이스 파티션*은 고유의 데이터, 인덱스, 구성 파일 및 트랜잭션 로그로 이루어진 데이터베이스의 일부분입니다. 데이터베이스 파티션을 노드 또는 데이터베이스 노드라고도 합니다. 파티션된 데이터베이스 환경은 데이터베이스 파티션에 있는 데이터의 분산을 지원합니다.

데이터베이스 파티션에 대한 자세한 내용은 *파티셔닝 및 클러스터링 안내서*의 내용을 참조하십시오.

제 7 장 버퍼 풀

버퍼 풀은 테이블 및 인덱스 데이터를 디스크에서 읽을 때 이를 캐시하기 위해 데이터베이스 관리 프로그램을 통해 할당한 주기억장치 영역입니다. 모든 DB2 데이터베이스에는 버퍼 풀이 있어야 합니다.

각각의 새 데이터베이스에는 IBMDEFAULTBP라는 디폴트 버퍼 풀이 정의되어 있습니다. CREATE BUFFERPOOL, DROP BUFFERPOOL 및 ALTER BUFFERPOOL 명령문을 사용하여 추가 버퍼 풀을 작성, 삭제(drop) 및 수정할 수 있습니다.
SYSCAT.BUFFERPOOLS 카탈로그 뷰는 데이터베이스에 정의된 버퍼 풀의 정보에 액세스합니다.

버퍼 풀 사용 방법

테이블의 데이터 행에 처음으로 액세스하면 데이터베이스 관리 프로그램이 해당 데이터가 들어 있는 페이지를 버퍼 풀에 저장합니다. 데이터베이스가 종료되거나 페이지가 점유하고 있는 스페이스를 다른 페이지가 사용해야 할 때까지 해당 페이지는 버퍼 풀에 남아 있습니다.

버퍼 풀에 있는 페이지는 사용 중이거나 그렇지 않을 수도 있으며 더티하거나 깨끗할 수도 있습니다.

- 사용 중인 페이지는 현재 읽거나 갱신 중인 페이지입니다. 데이터 일관성을 유지하기 위해 데이터베이스 관리 프로그램은 버퍼 풀에서 한 번에 하나의 에이전트만 지정된 페이지를 갱신하도록 합니다. 페이지를 갱신 중인 경우 한 에이전트만 이 페이지에 액세스합니다. 페이지를 읽는 중인 경우 여러 에이전트가 동시에 페이지를 읽을 수 있습니다.
- ″더티″ 페이지에는 변경되었지만 아직 디스크에 기록되지 않은 데이터가 들어 있습니다.
- 변경된 페이지가 디스크에 기록되면 이 페이지는 깨끗해져 버퍼 풀에 남아 있습니다.

데이터베이스 조정의 상당 부분은 데이터를 버퍼 풀로 이동하고 버퍼 풀에서 디스크로 데이터를 기록하는 작업을 제어하는 구성 매개변수 설정과 관련됩니다. 최근 에이전트가 페이지를 요구하지 않는 경우 새 응용프로그램의 새 페이지 요청에 페이지 스페이스를 사용할 수 있습니다. 데이터베이스 관리 프로그램 성능은 추가 디스크 입출력 만큼 저하됩니다.

스냅샷 모니터를 사용하여 버퍼 풀 사용 비율을 계산할 수 있으며 이는 버퍼 풀을 조정하는 데 유용합니다.

버퍼 풀 설계

모든 버퍼 풀의 *크기*는 데이터베이스 성능에 주요한 영향을 미칩니다.

새 버퍼 풀을 작성하기 전에 다음 내용을 해결하십시오.

- 사용하려는 버퍼 풀 이름은 무엇입니까?
- 버퍼 풀을 즉시 작성하려고 합니까 아니면 다음에 데이터베이스가 비활성화되고 재활성화된 이후에 작성하려고 합니까?
- 모든 데이터베이스 파티션에 버퍼 풀이 있어야 합니까 아니면 데이터베이스 파티션 서브세트에 있어야 합니까?
- 버퍼 풀에 할당하려는 페이지 *크기*는 얼마입니까? 131 페이지의 『버퍼 풀 페이지 크기』를 참조하십시오.
- 버퍼 풀 *크기*가 고정됩니까 아니면 데이터베이스 관리 프로그램이 워크로드에 따라서 버퍼 풀 *크기*를 자동으로 조정합니까? 버퍼 풀 작성 시 SIZE 매개변수를 지정하지 않고 남겨두어 데이터베이스 관리 프로그램이 자동으로 버퍼 풀을 조정하도록 하는 것이 좋습니다. 자세한 내용은 『CREATE BUFFERPOOL문』의 SIZE 매개변수 및 131 페이지의 『버퍼 풀 메모리 고려사항』의 내용을 참조하십시오.
- 블록 기반 입출력에 사용하도록 버퍼 풀 일부를 예약하시겠습니까? 자세한 내용은 『향상된 순차 프리페치의 블록 기반 버퍼 풀』을 참조하십시오.

테이블 스페이스와 버퍼 풀의 관계

버퍼 풀 설계 시 테이블 스페이스와 버퍼 풀 간의 관계를 이해해야 합니다. 각 테이블 스페이스는 특정 버퍼 풀과 연관됩니다. IBMDEFAULTBP는 디폴트 버퍼 풀입니다. 또한 데이터베이스 관리 프로그램이 IBMSYSTEMBP4K, IBMSYSTEMBP8K, IBMSYSTEMBP16K 및 IBMSYSTEMBP32K와 같은 시스템 버퍼 풀을 할당합니다 (이전에는 『숨겨진 버퍼 풀』로 알려짐). 다른 버퍼 풀을 테이블 스페이스와 연관시키려면 버퍼 풀이 존재해야 하고 버퍼 풀과 테이블 스페이스의 페이지 *크기*가 같아야 합니다. CREATE TABLESPACE문을 사용하여 테이블 스페이스 작성 시 연관을 정의하지만 나중에 ALTER TABLESPACE문을 사용하여 변경할 수 있습니다.

둘 이상의 버퍼 풀을 사용하면 데이터베이스에서 사용되는 메모리를 구성하여 전반적인 성능을 향상시킬 수 있습니다. 예를 들어, 사용자가 무작위로 액세스하는 하나 이상의 대형(사용 가능한 메모리보다 큰) 테이블이 포함된 테이블 스페이스가 있는 경우 데이터 페이지를 캐시하는 것이 유리하지 않기 때문에 버퍼 풀 *크기*가 제한됩니다. 응용프로그램에서 사용하는 데이터 페이지를 오래 캐시할 수 있도록 온라인 트랜잭션 응용프로그램의 테이블 스페이스가 대형 버퍼 풀과 연관되어 응답 시간이 빨라집니다. 새 버퍼 풀 구성 시 주의해야 합니다.

버퍼 풀 페이지 크기

디폴트 버퍼 풀의 페이지 *크기*는 CREATE DATABASE 명령을 사용할 때 설정됩니다. 이 디폴트값은 향후 모든 CREATE BUFFERPOOL 및 CREATE TABLESPACE 문의 디폴트 페이지 *크기*를 나타냅니다. 데이터베이스 작성 시 페이지 *크기*를 지정하지 않는 경우 디폴트 페이지 *크기*는 4KB입니다.

주: 데이터베이스에 8KB, 16KB 또는 32KB의 페이지 *크기*가 필요한 것으로 판별된 경우 일치하는 페이지 *크기*가 정의되고 데이터베이스의 테이블 스페이스와 연관된 버퍼 풀이 하나 이상 있어야 합니다.

그러나 시스템 버퍼 풀과는 다른 특성을 가진 버퍼 풀이 필요할 수도 있습니다. 데이터베이스 관리 프로그램이 사용할 새 버퍼 풀을 작성할 수 있습니다. 테이블 스페이스 및 버퍼 풀 변경사항을 적용하려면 데이터베이스를 재시작해야 합니다. 테이블 스페이스에 지정하는 페이지 *크기*는 버퍼 풀에 사용하도록 선택하는 페이지 *크기*를 판별해야 합니다. 버퍼 풀을 작성한 후에는 페이지 *크기*를 변경할 수 없기 때문에 버퍼 풀에 사용되는 페이지 *크기*를 선택하는 것이 중요합니다.

버퍼 풀 메모리 고려사항

메모리 요구사항

버퍼 풀 설계 시 사용자 컴퓨터에 설치된 메모리 양에 따른 메모리 요구사항 및 동일한 컴퓨터에서 데이터베이스 관리 프로그램과 동시에 실행 중인 기타 응용프로그램에 필요한 메모리도 고려해야 합니다. 액세스되는 모든 데이터를 보유할 메모리가 부족한 경우에는 운영 체제 데이터 스와핑이 발생합니다. 이는 다른 데이터에 필요한 공간을 확보하기 위해 임시 디스크 스토리지에 일부 데이터를 작성하거나 스왑하는 경우 발생합니다. 임시 디스크 스토리지에 있는 데이터가 필요한 경우 이 데이터는 주기억장치로 다시 스왑됩니다.

버퍼 풀 메모리 보호

버전 9.5에서는 스토리지 키를 사용하여 버퍼 풀 메모리의 데이터 페이지를 보호하며, 스토리지 키는 DB2_MEMORY_PROTECT 레지스트리 변수를 통해 명시적으로 사용 가능하게 설정된 경우에만, 그리고 POWER6™에서 실행되는 AIX(5.3 TL06 5.4)에서만 사용할 수 있습니다.

버퍼 풀 메모리 보호는 개별 에이전트 레벨에서 적용됩니다. 특정 에이전트는 액세스가 필요한 경우에만 버퍼 풀 페이지에 액세스할 수 있습니다. 메모리 보호는 DB2 엔진 스레드가 버퍼 풀 메모리에 액세스할 수 있는 시간과 액세스할 수 없는 시간을 식별하여 작동합니다. 자세한 내용은 132 페이지의 『버퍼 풀 메모리 보호(POWER6에서 실행 중인 AIX)』의 내용을 참조하십시오.

AWE(Address Windowing Extensions) 및 ESTORE(Extended Storage)

주: ESTORE 관련 키워드, 모니터 요소 및 데이터 구조를 포함하여 AWE 및 ESTORE 기능은 더 이상 사용되지 않습니다. 추가 메모리를 할당하려면 64비트 하드웨어 운영 체제 및 연관된 DB2 제품으로 업그레이드해야 합니다. 또한 응용프로그램 및 스크립트를 수정하여 더 이상 사용되지 않는 이 기능에 대한 참조를 제거해야 합니다.

버퍼 풀 메모리 보호(POWER6에서 실행 중인 AIX)

데이터베이스 관리 프로그램에서는 버퍼 풀을 사용하여 추가사항, 수정사항 및 삭제사항을 여러 데이터베이스 데이터에 적용합니다. POWER6에서 실행 중인 AIX 5.3 TL06+에서는 *스토리지 키*를 사용하여 버퍼 풀 메모리를 보호할 수 있습니다.

*스토리지 키*는 IBM Power6 프로세서 및 AIX 운영 체제의 새 기능으로 커널 스레드 레벨에서 하드웨어 키를 사용하여 여러 메모리를 보호할 수 있습니다. 스토리지 키 보호를 사용하면 버퍼 풀 메모리 손상 문제를 줄이고 데이터베이스를 정지시킬 수도 있는 오류가 제한됩니다. 프로그래밍을 통해 올바르지 않게 버퍼 풀에 액세스하려 하면 오류 조건이 발생하며 데이터베이스 관리 프로그램이 이를 발견하여 처리할 수 있습니다.

주: 버퍼 풀 메모리 보호는 개별 에이전트 레벨에서 적용됩니다. 특정 에이전트는 액세스가 필요한 경우에만 버퍼 풀 페이지에 액세스할 수 있습니다.

데이터베이스 관리 프로그램은 버퍼 풀 메모리에 대한 액세스를 제한하여 버퍼 풀을 보호합니다. 에이전트가 버퍼 풀에 액세스하여 작업을 수행해야 하는 경우 일시적으로 버퍼 풀 메모리에 액세스할 수 있는 권한이 부여됩니다. 에이전트가 더 이상 버퍼 풀에 액세스할 필요가 없으면 액세스 권한이 취소됩니다. 이렇게 함으로써 반드시 필요한 경우에만 에이전트가 버퍼 풀 컨텐츠를 수정하도록 허용하여 버퍼 풀이 손상될 가능성을 줄일 수 있습니다. 버퍼 풀 메모리에 올바르지 않게 액세스하면 세그먼트 오류가 발생합니다. 이러한 오류를 진단하기 위한 도구(예: db2diag, db2fodc, db2pdcfg 및 db2support 명령)이 제공됩니다.

버퍼 풀 메모리 보호 기능을 사용하려면, 데이터베이스 엔진의 장애 허용성을 늘리기 위해 DB2_MEMORY_PROTECT 레지스트리 변수를 사용 가능하게 하십시오.

DB2_MEMORY_PROTECT 레지스트리 변수

이 레지스트리 변수는 버퍼 풀 메모리 보호 기능을 사용 가능 및 사용 불가능으로 설정합니다. DB2_MEMORY_PROTECT가 사용되고(YES로 설정) DB2 엔진 스레드가 올바르지 않게 버퍼 풀 메모리에 액세스하려고 하면 해당 엔진 스레드가 트랩됩니다. 디폴트값은 NO입니다.

메모: 버퍼 풀 메모리 보호 기능은 AIX 스토리지 보호 키 구현에 종속되며 고정된 공유 메모리에는 적용되지 않습니다. DB2_MEMORY_PROTECT를 DB2_PINNED_BP 또는 DB2_LARGE_PAGE_MEM 설정으로 지정하는 경우 AIX 스토리지 보호 키를 사용할 수 없습니다. AIX 스토리지 보호 키에 대

한 자세한 정보는 다음 링크를 참조하십시오. http://publib.boulder.ibm.com/infocenter/systems/scope/aix/index.jsp?topic=/com.ibm.aix.genprogc/doc/genprogc/storage_protect_keys.htm

DB2_LGPAGE_BP가 YES로 설정되면 메모리 보호를 사용할 수 없습니다. DB2_MEMORY_PROTECT가 YES로 설정되는 경우에도, DB2 데이터베이스 관리 프로그램은 버퍼 풀 메모리를 보호하지 못하고 기능을 사용 불가능하게 합니다.

버퍼 풀 작성

CREATE BUFFERPOOL문을 사용하여 데이터베이스 관리 프로그램이 사용할 새 버퍼 풀을 정의합니다.

기본 CREATE BUFFERPOOL문의 예는 다음과 같습니다.

```
CREATE BUFFERPOOL <buffer pool name>
  PAGESIZE 4096
```

사용 가능한 메모리가 충분한 경우 버퍼 풀을 바로 활성화할 수 있습니다. 디폴트로 IMMEDIATE 키워드를 사용하여 새 버퍼 풀이 작성되므로 대부분의 플랫폼에서 데이터베이스 관리 프로그램은 추가 메모리를 획득할 수 있습니다. 그러면 메모리 할당에 성공해야 합니다. 데이터베이스 관리 프로그램이 추가 메모리를 할당할 수 없는 경우에만 버퍼 풀을 시작할 수 없고 다음 데이터베이스 시작 시 버퍼 풀이 시작됨을 알리는 경고 상태가 리턴됩니다. 즉각적인 요청을 위해 데이터베이스를 재시작할 필요가 없습니다. 이 명령문이 커미트되면 시스템 카탈로그 테이블에 버퍼 풀이 반영된다고 하더라도 다음에 데이터베이스가 시작될 때까지 버퍼 풀이 활성화되지 않습니다. 다른 옵션을 비롯하여 이러한 명령문에 대한 자세한 정보는 『CREATE BUFFERPOOL문』을 참조하십시오.

CREATE BUFFERPOOL DEFERRED를 발행하면 버퍼 풀이 즉시 활성화되지 않습니다. 대신 다음 데이터베이스 시작 시 작성됩니다. 테이블 스페이스가 지연된 버퍼 풀을 명시적으로 사용하도록 작성되더라도 데이터베이스가 재시작될 때까지 모든 새 테이블 스페이스에서는 기존 버퍼 풀을 사용합니다.

컴퓨터에 작성한 전체 버퍼 풀에 대해 충분한 실제 메모리가 있어야 합니다. 또한 운영체제에는 조작을 위한 메모리도 필요합니다.

명령행을 사용하여 버퍼 풀을 작성하려면 다음을 수행하십시오.

1. 다음 SQL문을 발행하여 데이터베이스에 이미 있는 버퍼 풀 이름 목록을 가져옵니다.

   ```
   SELECT BPNAME FROM SYSCAT.BUFFERPOOLS
   ```

2. 현재 결과 목록에 없는 버퍼 풀 이름을 선택합니다.

3. 작성하려는 버퍼 풀의 특성을 결정합니다.
4. CREATE BUFFERPOOL문을 실행하는 데 올바른 권한 부여 ID가 있는지 확인합니다.
5. CREATE BUFFERPOOL문을 발행합니다.

파티션된 데이터베이스의 각 데이터베이스 파티션에서 버퍼 풀이 다르게 작성되도록 정의할 수도 있습니다(예: *크기*가 다르게 작성). 디폴트 ALL DBPARTITIONNUMS절은 이 버퍼 풀이 데이터베이스의 모든 데이터베이스 파티션에서 작성됨을 나타냅니다.

다음 예에서 선택적 DATABASE PARTITION GROUP절은 버퍼 풀 정의가 적용될 데이터베이스 파티션 그룹을 식별합니다.

```
CREATE BUFFERPOOL <buffer pool name>
  PAGESIZE 4096
  DATABASE PARTITION GROUP <db partition group name>
```

해당 매개변수가 지정되어 있는 경우, 해당 데이터베이스 파티션 그룹에 있는 데이터베이스 파티션에만 버퍼 풀이 작성됩니다. 각 데이터베이스 파티션 그룹은 현재 데이터베이스에 있어야 합니다. DATABASE PARTITION GROUP절이 지정되지 않은 경우, 이 버퍼 풀은 모든 데이터베이스 파티션(및 해당 데이터베이스에 이어서 추가되는 모든 데이터베이스 파티션)에 작성됩니다.

자세한 정보는 『CREATE BUFFERPOOL문』을 참조하십시오.

버퍼 풀 수정

버퍼 풀을 수정하려고 하는 여러 가지 이유가 있습니다(예: 자체 성능 조정 메모리를 위해). 이렇게 하려면 ALTER BUFFERPOOL문을 사용합니다.

명령문의 권한 부여 ID는 SYSCTRL 또는 SYSADM 권한을 가지고 있어야 합니다.

버퍼 풀을 사용하여 작업하려면 다음 태스크 중 하나를 수행해야 합니다.

- 버퍼 풀의 자체 성능 조정을 사용 가능하도록 설정하면 데이터베이스 관리 프로그램에서 작업 로드에 대한 응답으로 버퍼 풀의 *크기*를 조정할 수 있습니다.
- 블록 기반 입출력에 대한 버퍼 풀의 블록 영역 수정
- 해당 버퍼 풀 정의를 새 데이터베이스 파티션 그룹에 추가
- 일부 또는 모든 데이터베이스 파티션의 버퍼 풀 *크기* 수정

명령행을 사용하여 버퍼 풀을 변경하려면 다음을 수행하십시오.

1. 데이터베이스에 이미 있는 버퍼 풀 이름 목록을 가져오려면 다음 명령문을 발행하십시오.

   ```
   SELECT BPNAME FROM SYSCAT.BUFFERPOOLS
   ```

2. 결과 목록에서 버퍼 풀 이름을 선택합니다.

3. 변경해야 할 사항을 판별합니다.
4. ALTER BUFFERPOOL문을 실행하는 데 올바른 권한 부여 ID가 있는지 확인합니다.

   주: 두 개의 키 매개변수는 IMMEDIATE 및 DEFERRED입니다. 버퍼 풀에 대한 다음 데이터베이스 활성화가 실행될 때까지 기다릴 필요 없이 IMMEDIATE를 사용하여 버퍼 풀 *크기*가 변경됩니다. 새 스페이스를 할당하는 데 데이터베이스 공유 메모리가 부족한 경우 해당 명령문이 DEFERRED로 실행됩니다.

   DEFERRED를 사용하면 데이터베이스가 재활성화될 때까지 버퍼 풀에 대한 변경사항이 적용되지 않습니다. 활성화 시 데이터베이스 관리 프로그램이 필요한 메모리를 시스템에서 할당하므로 예약 메모리 스페이스가 필요하지 않습니다.
5. ALTER BUFFERPOOL문을 사용하여 버퍼 풀 오브젝트의 단일 속성을 변경합니다. 예를 들어, 다음과 같습니다.

   ```
   ALTER BUFFERPOOL buffer pool name SIZE number of pages
   ```

- *buffer pool name*은 시스템 카탈로그에 설명된 버퍼 풀을 식별하는 한 부분의 이름입니다.
- *number of pages*는 이러한 특정 버퍼 풀에 할당되는 새 페이지 수 입니다. 또한 값 -1을 사용할 수 있습니다. 이 값은 버퍼 풀의 *크기*가 **buffpage** 데이터베이스 구성 매개변수에 있는 값이어야 함을 나타냅니다.

명령문에는 버퍼 풀의 *크기*가 조정되는 데이터베이스 파티션을 지정하는 DBPARTITIONNUM <db partition number>절이 있을 수 있습니다. 이 절을 지정하지 않으면 버퍼 풀 *크기*가 SYSCAT.BUFFERPOOLDBPARTITIONS에서 예외 항목이 있는 경우를 제외하고 모든 데이터베이스 파티션에서 수정됩니다. 데이터베이스 파티션에 이러한 절 사용에 대한 자세한 내용은 ALTER BUFFERPOOL문을 참조하십시오.

이러한 명령문의 결과로 발생한 버퍼 풀에 대한 변경사항은 해당 명령문이 커미트되는 시스템 카탈로그 테이블에 반영됩니다. 그러나 다음에 데이터베이스가 시작될 때까지 실제 버퍼 풀에 대한 변경사항은 적용되지 않습니다. 그러나 디폴트 IMMEDIATE 키워드로 지정된 성공적인 ALTER BUFFERPOOL 요청의 경우는 예외입니다.

작성한 모든 버퍼 풀의 총계를 위한 충분한 실제 메모리가 컴퓨터에 있어야 합니다. 나머지 데이터베이스 관리 프로그램 및 응용프로그램을 위한 충분한 실제 메모리도 필요합니다.

버퍼 풀 삭제

버퍼 풀을 삭제하는 경우 해당 버퍼 풀에 지정된 테이블 스페이스가 없어야 합니다. IBMDEFAULTBP 버퍼 풀을 삭제할 수 없습니다.

디스크 스토리지는 다음에 데이터베이스에 연결할 때 해제됩니다. 스토리지 메모리는 데이터베이스가 중지될 때까지 삭제된 버퍼 풀에서 실제로 릴리스되지 않습니다. 데이터베이스 관리 프로그램에서 사용하도록 버퍼 풀 메모리가 즉시 릴리스됩니다.

DROP BUFFERPOOL문을 사용하여 다음과 같이 버퍼 풀을 삭제할 수 있습니다.

```
DROP BUFFERPOOL <buffer pool name>
```

제 8 장 테이블 스페이스

*테이블 스페이스*는 테이블, 인덱스, 대형 오브젝트(LOB) 및 Long 데이터가 포함된 스토리지 구조입니다. 데이터베이스의 데이터를 데이터가 시스템에서 저장되는 위치와 관련된 논리적 스토리지 그룹으로 구성하는 데 사용됩니다. 테이블 스페이스는 데이터베이스 파티션 그룹에 저장됩니다.

테이블 스페이스를 사용하여 스토리지를 구성하면 다음과 같은 많은 이점이 있습니다.

복구 가능성

함께 백업 또는 리스토어되어야 하는 오브젝트를 동일한 테이블 스페이스에 넣으면 하나의 명령으로 테이블 스페이스의 모든 오브젝트를 백업 또는 리스토어할 수 있으므로 백업 및 리스토어 작업이 더욱 편리해집니다. 테이블 스페이스에 분산되는 파티션된 테이블 및 인덱스가 있는 경우 주어진 테이블 스페이스에 상주하는 데이터 및 인덱스 파티션만 백업 또는 리스토어할 수 있습니다.

더 많은 테이블

하나의 테이블 스페이스에 저장할 수 있는 테이블 수에 한계가 있습니다. 테이블 스페이스에 포함될 수 있는 것보다 많은 추가 테이블이 필요한 경우 해당 테이블을 위한 추가 테이블 스페이스만 작성하면 됩니다.

스토리지 유연성

DMS 및 SMS 테이블 스페이스의 경우 데이터를 저장하는 데 사용되는 스토리지 디바이스를 지정할 수 있습니다. 예를 들어 현재 조작 데이터는 더 빠른 디바이스에 상주하는 테이블 스페이스에 저장하고 실행기록 데이터는 더 느린(그리고 덜 비싼) 디바이스에 상주하는 테이블 스페이스에 저장할 수 있습니다.

성능 또는 메모리 활용도 향상을 위해 버퍼 풀에 있는 데이터를 분리하는 기능

자주 쿼리되는 오브젝트(예: 테이블, 인덱스) 세트가 있는 경우 하나의 CREATE 또는 ALTER TABLESPACE문을 사용하여 버퍼 풀이 상주하는 테이블 스페이스를 지정할 수 있습니다. 임시 테이블 스페이스를 자체 버퍼 풀에 지정하여 정렬이나 조인 같은 활동을 성능을 향상시킬 수 있습니다. 일부 경우에는 자주 액세스되지 않는 데이터나 매우 큰 테이블에 대한 무작위 액세스가 필요한 응용프로그램에 대해 더 작은 버퍼 풀을 정의하는 것이 좋을 수 있습니다. 그런 경우 단일 쿼리보다 오래 동안 데이터를 버퍼 풀에 보존할 필요가 없습니다.

테이블 스페이스는 하나 이상의 *컨테이너*로 구성됩니다. 컨테이너는 디렉토리 이름, 디바이스 이름 또는 파일 이름일 수 있습니다. 하나의 테이블 스페이스가 여러 개의 컨테이너를 가질 수 있습니다. 다중 컨테이너(하나 이상의 테이블 스페이스의)가 동일한 실제 스토리지 디바이스에 작성될 수 있습니다(작성하는 각 컨테이너가 서로 다른 스토리지 디바이스를 사용하는 경우 최상의 성능을 얻을 수 있습니다). 자동 스토리지 테이블

스페이스를 사용 중인 경우 컨테이너 작성 및 관리는 데이터베이스 관리 프로그램에 의해 자동으로 처리됩니다. 자동 스토리지 테이블 스페이스를 사용 중이 아니면 사용자 스스로 컨테이너를 정의하고 관리해야 합니다.

그림 7은 데이터베이스 내의 테이블과 테이블 스페이스와의 관계 및 데이터베이스와 연관된 컨테이너를 나타냅니다.

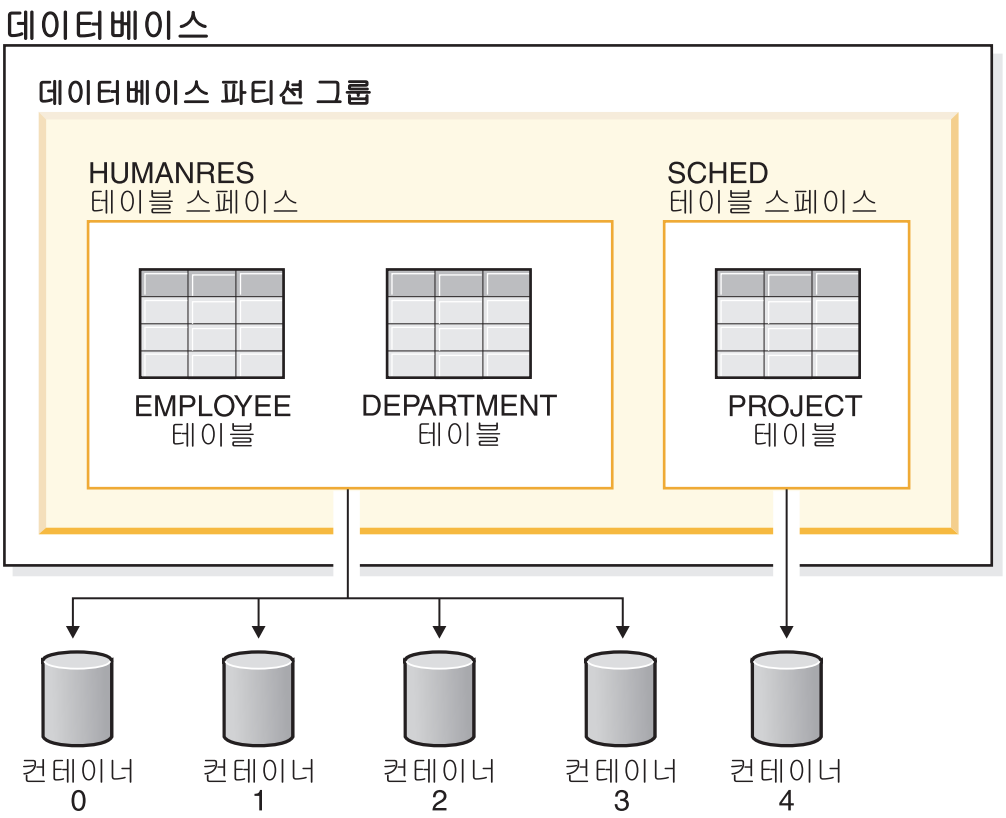


그림 7. 데이터베이스 내에 있는 테이블 스페이스 및 테이블

EMPLOYEE 및 DEPARTMENT 테이블은 컨테이너 0, 1, 2, 3에 포함되는 HUMANRES 테이블 스페이스에 있습니다. PROJECT 테이블은 컨테이너 4의 SCHED 테이블 스페이스에 있습니다. 이 예는 별도의 디스크에 존재하는 각 컨테이너를 보여줍니다.

데이터베이스 관리 프로그램은 가능한 한 컨테이너 간에 균형 있게 데이터를 로드하려고 시도합니다. 결과적으로 모든 컨테이너가 데이터를 저장하는데 사용됩니다. 다른 컨테이너를 사용하기 전에 데이터베이스 관리 프로그램이 해당 컨테이너에 기록하는 페이지 수를 *Extent 크기*라고 합니다. 데이터베이스 관리 프로그램은 항상 첫 번째 컨테이너에 테이블 데이터를 저장하는 것은 아닙니다.

139 페이지의 그림 8은 Extent 크기가 두 개의 4KB페이지이고, 4개의 컨테이너(각각에는 적은 수의 할당된 Extent가 있음)가 있는 HUMANRES 테이블 스페이스를 보여줍니다. DEPARTMENT 및 EMPLOYEE 테이블은 둘 다 7페이지를 가지고 있고 4

개의 컨테이너 모두에 포함됩니다.

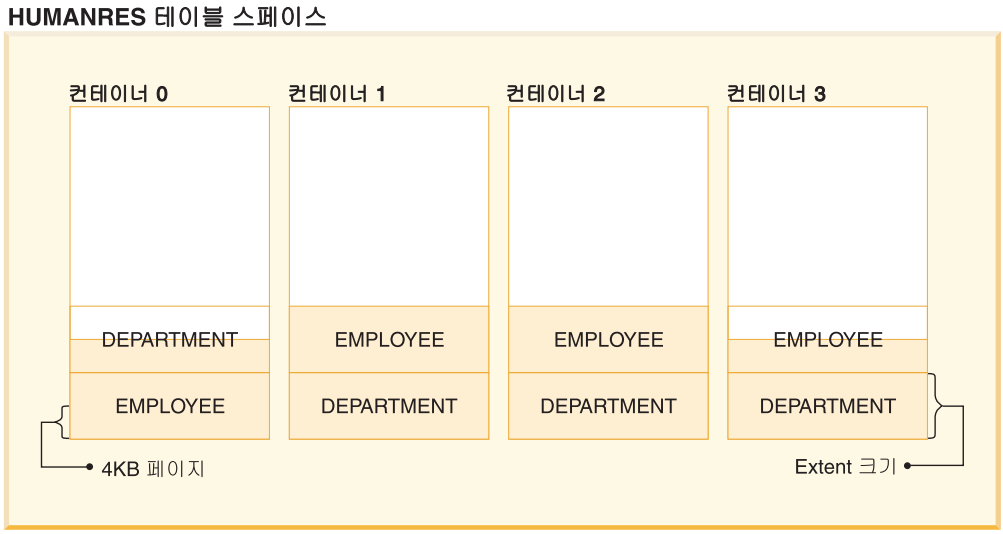


그림 8. 테이블 스페이스 내에 있는 컨테이너 및 Extent

시스템, 사용자 및 임시 데이터용 테이블 스페이스

각 데이터베이스는 시스템, 사용자 및 임시 데이터 저장에 사용되는 테이블 스페이스의 최소 세트를 가져야 합니다.

한 데이터베이스에는 최소한 세 개의 테이블 스페이스가 있어야 합니다.

- *카탈로그 테이블 스페이스*
- 하나 이상의 *사용자 테이블 스페이스*
- 하나 이상의 *임시 테이블 스페이스*

카탈로그 테이블 스페이스

카탈로그 테이블 스페이스에는 데이터베이스에 대한 모든 시스템 카탈로그 테이블이 들어있습니다. 이 테이블 스페이스를 SYSCATSPACE라 하며 삭제할 수 없습니다.

사용자 테이블 스페이스

사용자 테이블 스페이스에는 사용자 정의 테이블이 들어있습니다. 디폴트로 하나의 사용자 테이블 스페이스인 USERSPACE1이 작성됩니다.

테이블을 작성할 때 테이블에 대한 테이블 스페이스를 지정하지 않는 경우 데이터베이스 관리 프로그램이 하나를 선택합니다. 자세한 정보는 CREATE TABLE문의 IN *tablespace-name*절에 대한 문서를 참조하십시오.

테이블 스페이스의 페이지 *크기*가 테이블에서 가질 수 있는 최대 행 길이 또는 컬럼 수를 판별합니다. CREATE TABLE문에 대한 문서는 페이지 *크기*, 최대 행 *크기* 및 컬럼 계수 사이의 관계를 나타냅니다. 버전 9.1 이전에서는 디폴트 페이지 *크기*가 4KB입니다. 버전 9.1 및 이상에서는 디폴트 페이지 *크기*가 지원되는 기타 값일 수 있습니다. 디폴트 페이지 *크기*는 새 데이터베이스를 작성할 때 선언됩니다. 디폴트 페이지 *크기*가 선언되면 테이블에 대해 한 가지 페이지 *크기*를 갖는 테이블 스페이스를 작성하고, Long 또는 LOB 데이터에 대해서는 다른 페이지 *크기*를 갖는 다른 테이블 스페이스를 사용할 수 있습니다. 컬럼 수 또는 행 *크기*가 테이블 스페이스의 페이지 *크기* 제한을 초과하면, 오류가 리턴됩니다(SQLSTATE 42997).

임시 테이블 스페이스

임시 테이블 스페이스에는 임시 테이블이 들어있습니다. 임시 테이블 스페이스는 *시스템 임시 테이블 스페이스* 또는 *사용자 임시 테이블 스페이스*일 수 있습니다.

시스템 임시 테이블 스페이스는 정렬 또는 조인과 같은 조작을 수행하는 중에 데이터베이스 관리 프로그램에 필요한 임시 데이터를 보유합니다. 이러한 유형의 조작에서 결과 세트를 처리하려면 여분의 스페이스가 필요합니다. 한 데이터베이스에는 최소한 하나의 시스템 임시 테이블 스페이스가 있어야 합니다. 디폴트로 TEMPSPACE1이라는 하나의 시스템 임시 테이블 스페이스가 데이터베이스 작성시 작성됩니다.

쿼리를 처리할 때, 데이터베이스 관리 프로그램이 사용자 쿼리와 관련된 데이터를 조작하기에 충분히 큰 페이지 *크기*를 갖는 시스템 임시 테이블 스페이스에 액세스해야 합니다. 예를 들어, 쿼리가 8KB 길이의 행을 갖는 데이터를 리턴하고 최소 8KB의 페이지 *크기*를 갖는 시스템 임시 테이블 스페이스가 없는 경우 쿼리에 실패할 수 있습니다. 더 큰 페이지 *크기*를 갖는 시스템 임시 테이블 스페이스를 작성해야 합니다. 사용자 테이블 스페이스의 가장 큰 페이지 *크기*와 동일한 페이지 *크기*를 갖는 임시 테이블 스페이스를 정의하는 것이 이런 종류의 문제점을 피하는 데 도움이 됩니다.

사용자 임시 테이블 스페이스는 DECLARE GLOBAL TEMPORARY TABLE 또는 CREATE GLOBAL TEMPORARY TABLE문으로 작성된 테이블의 임시 데이터를 보유합니다. 디폴트로 데이터베이스 작성 시에 작성되지 않습니다. 또한 작성된 임시 테이블의 인스턴스화된 버전을 보유합니다. 선언되거나 작성된 임시 테이블의 정의를 허용하려면 적절한 USE 특권을 사용하여 최소한 하나의 사용자 임시 테이블 스페이스를 작성해야 합니다. USE 특권은 GRANT문을 사용하여 부여됩니다.

데이터베이스가 두 개 이상의 임시 테이블 스페이스를 사용하며 새 임시 오브젝트가 필요한 경우, 옵티마이저가 이 오브젝트에 적절한 페이지 *크기*를 선택합니다. 그런 다음, 이 오브젝트는 해당 페이지 *크기*를 가진 임시 테이블 스페이스에 할당됩니다. 해당 페이지 *크기*를 갖는 둘 이상의 임시 테이블 스페이스가 있는 경우, 테이블 스페이스는 해당 페이지 *크기*를 갖는 하나의 테이블 스페이스로 시작한 후 할당될 다음 오브젝트에 대해 다음으로 진행하는 방식으로 진행하고, 모든 적합한 테이블 스페이스가 사용된 후

첫 번째 테이블 스페이스로 리턴하는 라운드 로빈 방식으로 선택됩니다. 대부분의 경우에 동일한 페이지 크기를 갖는 둘 이상의 임시 테이블 스페이스를 갖는 것은 바람직하지 않습니다.

파티션된 데이터베이스 환경의 테이블 스페이스

파티션된 데이터베이스 환경에서 각 테이블 스페이스는 특정 데이터베이스 파티션 그룹과 연관됩니다. 이 경우, 테이블 스페이스의 특성이 데이터베이스 파티션 그룹의 각 데이터베이스 파티션에 적용될 수 있습니다.

테이블 스페이스를 데이터베이스 파티션 그룹에 할당할 때 데이터베이스 파티션 그룹이 이미 존재해야 합니다. 테이블 스페이스와 데이터베이스 파티션 그룹 사이의 연관은 CREATE TABLESPACE문을 사용하여 테이블 스페이스를 작성할 때 정의됩니다.

테이블 스페이스와 데이터베이스 파티션 그룹 사이의 연관을 변경할 수 없습니다. ALTER TABLESPACE문을 사용하여 데이터베이스 파티션 그룹의 개별 데이터베이스 파티션에 대한 테이블 스페이스 스펙만 변경할 수 있습니다.

단일 파티션 환경에서 각 테이블 스페이스는 다음과 같이 디폴트 데이터베이스 파티션 그룹과 연관됩니다.

- 카탈로그 테이블 스페이스 SYSCATSPACE가 IBMCATGROUP과 연관됩니다.
- 사용자 테이블 스페이스는 IBMDEFAULTGROUP과 연관됩니다.
- 임시 테이블 스페이스는 IBMTEMPGROUP과 연관됩니다.

파티션된 데이터베이스 환경에서 IBMCATGROUP 파티션에 세 개의 디폴트 테이블 스페이스가 모두 포함되며 기타 데이터베이스 파티션은 TEMPSPACE1 및 USERSPACE1만 포함합니다.

테이블 스페이스 및 스토리지 관리

테이블 스페이스가 사용 가능한 스토리지를 사용하기 원하는 방법에 따라서 여러 가지 방법으로 테이블 스페이스를 설정할 수 있습니다. 사용자가 지정하는 매개변수를 기초로 운영 체제가 스페이스의 할당을 관리하게 하거나, 데이터베이스 관리 프로그램이 데이터를 위한 스페이스를 할당하게 만들 수 있습니다. 또는 스토리지를 자동으로 할당하는 테이블 스페이스를 작성할 수 있습니다.

세 가지 유형의 테이블 스페이스는 다음과 같습니다.

- 시스템 관리 스페이스(SMS) - 사용자가 데이터베이스 파일 저장을 위한 위치를 정의한 후 운영 체제의 파일 관리 프로그램이 스토리지 스페이스를 제어합니다.
- 데이터베이스 관리 스페이스(DMS) - 사용자가 스토리지 컨테이너를 할당한 후 데이터베이스 관리 프로그램이 스토리지 스페이스의 사용을 제어합니다.
- 자동 스토리지 테이블 스페이스 - 데이터베이스 관리 프로그램이 필요할 때 컨테이너 작성을 제어합니다.

데이터베이스 내부에서 임의의 조합으로 각 유형을 함께 사용할 수 있습니다.

시스템 관리 스페이스

시스템 관리 스페이스(SMS) 테이블 스페이스에서 운영 체제의 파일 시스템 관리 프로그램은 테이블이 저장될 스페이스를 할당하고 관리합니다. 데이터베이스 관리(DMS) 테이블 스페이스와는 달리, 스토리지 스페이스는 테이블 스페이스가 작성될 때 사전 할당되지 않으며 요구 시 할당됩니다.

SMS 스토리지 모델은 데이터베이스 오브젝트를 나타내는 파일로 구성됩니다. 예를 들어 각 테이블에는 그와 연관된 실제 파일이 최소한 하나씩 들어 있습니다. 테이블 스페이스를 설정할 때 컨테이너를 작성하여 파일 위치를 결정합니다. SMS 테이블 스페이스의 각 컨테이너는 절대 또는 상대 디렉토리 이름과 연관됩니다. 이러한 각 디렉토리는 서로 다른 실제 스토리지 디바이스 또는 파일 시스템에 위치할 수 있습니다. 데이터베이스 관리 프로그램이 각 컨테이너의 오브젝트에 대해 작성되는 파일의 이름을 제어하고, 파일 시스템이 해당 파일의 관리를 담당합니다. 각 파일에 기록되는 데이터의 양을 제어함으로써, 데이터베이스 관리 프로그램은 여러 테이블 스페이스 컨테이너에 데이터를 균등하게 분배합니다.

스페이스가 할당되는 방법

SMS 테이블 스페이스에서, 테이블에 대한 스페이스는 요구시 할당됩니다. 할당되는 스페이스의 양은 *multipage_alloc* 데이터베이스 구성 매개변수의 설정에 따라 달라집니다. 이 구성 매개변수를 YES(디폴트)로 설정하면, 스페이스가 필요할 때 전체 Extent(일반적으로 두 개 이상의 페이지로 구성됨)가 할당됩니다. 아니면, 스페이스에 한 번에 한 페이지씩 할당됩니다.

다중 페이지 파일 할당은 테이블의 데이터와 인덱스 부분에만 영향을 줍니다. 이것은 Long 데이터(LONG VARCHAR, LONG VAR GRAPHIC), 대형 오브젝트(LOB)에 사용되는 파일이 한 번에 한 Extent씩 확장되지 않음을 의미합니다.

주: 다중 페이지 파일 할당은 시스템 관리 스페이스를 사용하는 임시 테이블 스페이스에는 적용되지 않습니다.

SMS 테이블 스페이스에 있는 단일 컨테이너의 모든 스페이스를 소비했을 때, 다른 컨테이너에 스페이스가 남아 있더라도 테이블 스페이스는 가득 찬 것으로 간주됩니다. DMS 테이블 스페이스와는 달리 컨테이너는 작성된 후에 SMS 테이블 스페이스에 추가할 수 없습니다. SMS 컨테이너에 추가 스페이스를 제공하려면 기본 파일 시스템에 추가 스페이스를 추가하십시오.

SMS 테이블 스페이스 계획

SMS 테이블 스페이스 사용을 고려할 때 두 가지 요소를 고려해야 합니다.

- **테이블 스페이스에서 필요한 컨테이너 수.** SMS 테이블 스페이스를 작성할 때 테이블 스페이스가 사용할 컨테이너 수를 지정해야 합니다. SMS 테이블 스페이스가 작성된 후에는 컨테이너를 추가하거나 삭제할 수 없기 때문에 사용하려는 컨테이너를 모두 식별하는 것은 매우 중요합니다. 한 가지 예외는 파티션된 데이터베이스 환경에 있는 경우입니다. 즉, 새 데이터베이스 파티션이 SMS 테이블 스페이스용 데이터베이스 파티션 그룹에 추가되면 ALTER TABLESPACE문을 사용하여 새 데이터베이스 파티션에 컨테이너를 추가할 수 있습니다.

  테이블 스페이스의 최대 크기는 다음 공식으로 계산할 수 있습니다.

  $n \times maxFileSystemSize$

  여기서 *n*은 컨테이너 수이고 *maxFileSystemSize*는 운영 체제가 지원하는 최대 파일 시스템 크기입니다.

  이 공식은 각 컨테이너가 고유한 파일 시스템에 맵핑되고 각 파일 시스템에 사용 가능한 최대 스페이스가 있으며 각 파일 시스템이 동일한 크기라고 가정합니다. 실제로는 이러한 경우가 발생하지 않을 수도 있고 최대 테이블 스페이스 크기가 더 작을 수도 있습니다. 데이터베이스 오브젝트의 크기에 대한 SQL 제한도 있으며, 이것이 테이블 스페이스의 최대 크기에 영향을 줄 수 있습니다.

  경고: SMS 테이블 스페이스에 대해 사용자가 지정하는 경로는 다른 파일이나 디렉토리를 포함해서는 안됩니다.

- **테이블 스페이스에 대한 Extent 크기.** *Extent 크기*는 다른 컨테이너를 사용하기 전에 데이터베이스 관리 프로그램이 컨테이너에 기록하는 페이지 수입니다. Extent 크기는 테이블 스페이스가 작성될 때만 지정할 수 있습니다. 이는 나중에 변경할 수 없기 때문에 적절한 Extent 크기 값을 선택하는 것이 중요합니다.

  테이블 스페이스를 작성할 때 Extent 크기를 지정하지 않으면 데이터베이스 관리 프로그램은 *dft_extent_sz* 데이터베이스 구성 매개변수에 정의된 디폴트 Extent 크기를 사용하여 테이블 스페이스를 작성합니다. 이 구성 매개변수는 데이터베이스를 작성할 때 제공한 정보를 기반으로 초기에 설정됩니다. DFT_EXTENT_SZ의 값이 CREATE DATABASE 명령에 대해 지정되지 않는 경우 디폴트 Extent 크기는 32로 설정됩니다.

컨테이너 및 Extent 크기

적절한 컨테이너 수와 테이블 스페이스에 대한 Extent 크기를 선택하려면, 다음을 이해해야 합니다.

- **운영 체제가 논리적 파일 시스템의 크기에 대해 설정하는 한계**. 예를 들어, 일부 운영 체제의 제한값은 2GB입니다. 따라서 64GB 테이블 오브젝트를 원할 경우, 이 유형의 시스템에 적어도 32개의 컨테이너가 필요합니다. 테이블 스페이스를 작성할 때,

다른 파일 시스템에 있는 컨테이너를 지정할 수 있습니다. 그러나 이로 인해 데이터베이스에 저장할 수 있는 데이터의 양이 증가합니다.

- **데이터베이스 관리 프로그램이 테이블 스페이스와 연관된 데이터 파일 및 컨테이너를 관리하는 방법.** 첫 번째 테이블 데이터 파일(규칙에 따라, SQL00002.DAT)은 테이블 스페이스 컨테이너 중 하나에 작성됩니다. 데이터베이스 관리 프로그램은 테이블 ID와 함께 컨테이너의 총 수를 고려하는 알고리즘을 기초로 사용할 컨테이너를 판별합니다. 이 파일은 Extent *크기*까지 확장할 수 있습니다. 이 *크기*에 도달한 후 데이터베이스 관리 프로그램이 다음 컨테이너의 SQL00002.DAT에 데이터를 기록합니다. 이 프로세스는 모든 컨테이너가 SQL00002.DAT 파일을 포함할 때까지 계속되며, 그 이후에는 데이터베이스 관리 프로그램이 시작 컨테이너로 리턴합니다. *스트라이핑*이라고 하는 이 프로세스는 컨테이너가 가득 차거나(SQL0289N) 또는 운영 체제로부터 더 이상의 스페이스를 할당받을 수 없을 때까지(디스크 가득참 오류) 컨테이너 디렉토리를 통해 계속됩니다. 스트라이핑은 테이블 데이터를 저장하는 데 사용되는 기타 오브젝트뿐 아니라 블록 맵 파일(SQLnnnnn.BKM), 인덱스 오브젝트에 적용됩니다. 데이터베이스 관리 프로그램이 제공하는 스트라이핑과 함께 디스크 스트라이핑을 구현하려는 경우 테이블 스페이스의 Extent *크기*와 디스크의 스트라이프 *크기*가 동일해야 합니다.

  주: 해당 컨테이너 중 하나가 가득 차면 SMS 테이블 스페이스는 즉시 바로 가득 찬 것으로 간주됩니다. 따라서 각 컨테이너에 동일한 양의 스페이스를 사용 가능하게 하는 것이 중요합니다.

  SMS 테이블 스페이스는 CREATE DATABASE 명령이나 CREATE TABLESPACE문에서 MANAGED BY SYSTEM 옵션을 사용하여 정의됩니다.

데이터베이스 관리 스페이스

데이터베이스 관리 스페이스(DMS) 테이블 스페이스에서는 데이터베이스 관리 프로그램이 스토리지 스페이스를 제어합니다. SMS 테이블 스페이스와는 달리, 스토리지 스페이스는 DMS 테이블 스페이스를 작성할 때 사용자가 지정하는 컨테이너 정의를 기초로 파일 시스템에 사전 할당됩니다.

DMS 스토리지 모델은 데이터베이스 관리 프로그램이 스페이스를 관리하는 제한된 수의 파일이나 디바이스로 구성됩니다. 사용자가 컨테이너를 작성할 때 사용할 파일과 디바이스를 결정하며, 해당 파일 및 디바이스에 대한 스페이스를 관리합니다.

사용자 정의 테이블 및 데이터가 들어있는 DMS 테이블 스페이스는 모든 테이블 데이터 또는 인덱스 데이터를 저장하는 *대형* (디폴트) 또는 *일반* 테이블 스페이스로 정의될 수 있습니다. 일반 체이블 스페이스의 최대 *크기*는 32KB 페이지에 대해 512GB입니다. 대형 테이블 스페이스의 최대 *크기*는 64TB입니다. 기타 페이지 *크기*에 대한 일반 테이블 스페이스의 최대 *크기*는 *SQL 참조서*의 『SQL 및 XML 한계』를 참조하십시오.

DMS 테이블 스페이스에 대해 작업할 때 컨테이너에 대한 두 가지 옵션인 파일 및 원시 디바이스가 있습니다. 파일 컨테이너에 대해 작업할 때 데이터베이스 관리 프로그램은 테이블 스페이스 작성시 전체 컨테이너를 할당합니다. 전체 테이블 스페이스에 대한 이러한 초기 할당의 결과로, 파일 시스템은 할당을 수행하지 않더라도 실제 할당이 일반적으로 연속하여 이루어집니다. 이때 항상 연속된다고 보증할 수는 없습니다. 원시 디바이스 컨테이너에 대해 작업할 때 데이터베이스 관리 프로그램이 전체 디바이스를 제어하며 항상 *Extent*의 페이지가 연속되도록 합니다. (*Extent*는 데이터베이스 관리 프로그램이 다른 컨테이너를 사용하기 전에 컨테이너에 기록하는 페이지 수로 정의됩니다.)

DMS 테이블 스페이스 계획

DMS 테이블 스페이스 및 컨테이너를 설계할 때 다음을 고려하십시오.

- 데이터베이스 관리 프로그램은 스트라이핑을 사용하여 모든 컨테이너에 데이터가 균등하게 배포되도록 합니다.이것은 테이블 스페이스의 모든 컨테이너에 데이터를 균등하게 기록하며, 테이블에 대한 Extent를 모든 컨테이너 사이에 라운드 로빈 방식으로 배치합니다. 다중 컨테이너에 데이터를 기록할 때는 DB2 스트라이핑을 권장합니다. DB2 스트라이핑과 함께 디스크 스트라이핑을 구현하려는 경우 테이블 스페이스의 Extent *크기*와 디스크의 스트라이프 *크기*가 동일해야 합니다.
- SMS 테이블 스페이스와는 달리, DMS 테이블 스페이스를 구성하는 컨테이너는 크기가 서로 동일할 필요가 없습니다. 그러나 이런 경우에는 컨테이너에서 스트라이핑이 고르지 않게 되며 성능도 최적화되지 않으므로 사용하지 않는 것이 좋습니다. 컨테이너가 가득 차면, DMS 테이블 스페이스는 다른 컨테이너에서 사용 가능한 스페이스를 사용합니다.
- 스페이스는 사전 할당되기 때문에 우선 스페이스가 사용 가능한 상태가 되어야 테이블 스페이스를 작성할 수 있습니다. 디바이스 컨테이너를 사용할 때 컨테이너 정의를 위한 스페이스도 충분해야 합니다. 각 디바이스는 디바이스에 대해 정의된 하나의 컨테이너만 가질 수 있습니다. 스페이스 낭비를 피하기 위해서는 디바이스 *크기*와 컨테이너 *크기*가 동일해야 합니다. 예를 들어 디바이스가 5,000페이지의 스토리지 공간을 갖고 디바이스 컨테이너가 3,000페이지로 정의되는 경우 디바이스의 2,000페이지는 사용할 수 없습니다.
- 디폴트로 모든 컨테이너에 하나의 Extent가 오버헤드용으로 예약되어 있습니다. 전체 Extent만 사용되므로, 최적의 스페이스 관리를 위해 다음 공식을 사용하여 컨테이너를 할당할 때 사용할 적절한 *크기*를 결정할 수 있습니다.

  $$extent\_size * (n + 1)$$

  여기서 *extent_size*는 테이블 스페이스에서 각 Extent *크기*이며 *n*은 컨테이너에 저장할 Extent의 수입니다.
- DMS 테이블 스페이스의 최소 *크기*는 5개의 Extent입니다.
  - 테이블 스페이스에 있는 세 개의 Extent는 오버헤드용으로 예약됩니다.

– 최소한 두 개의 Extent가 사용자 테이블 데이터를 저장하는데 필요합니다. (이들 Extent는 하나의 테이블에 대한 일반 데이터용이며 자체 Extent를 필요로 하는 인덱스, long 필드 또는 대형 오브젝트(LOB) 데이터용이 아닙니다.)

테이블 스페이스를 5개의 Extent보다 작게 작성하려고 하면 오류(SQL1422N)가 발생합니다.

- 디바이스 컨테이너는 물리적 볼륨이 아닌 논리적 볼륨을 『문자 특수 인터페이스』와 함께 사용해야 합니다.
- DMS 테이블 스페이스가 있는 디바이스 대신 파일을 사용할 수 있습니다. 디폴트 테이블 스페이스 속성(버전 9.5의 NO FILE SYSTEM CACHING)은 파일이 디바이스를 설정할 필요가 없는 장점과 함께 디바이스에 가깝게 수행할 수 있도록 합니다. 자세한 정보는 180 페이지의 『파일 시스템 캐싱 없는 테이블 스페이스』를 참조하십시오.
- 워크로드에 LOB 또는 LONG VARCHAR 데이터가 포함되는 경우 파일 시스템 캐시를 통해 성능을 높일 수 있습니다.

  주: LOB 및 LONG VARCHAR는 데이터베이스 관리 프로그램의 버퍼 풀로 버퍼되지 않습니다.
- 일부 운영 체제에서는 *크기*가 2GB를 넘는 물리적 디바이스를 설치할 수 있습니다. 물리적 디바이스를 다중 논리 디바이스로 나누어 운영 체제에서 허용하는 *크기*보다 큰 컨테이너가 없도록 해야 합니다.

DMS 테이블 스페이스에 대해 작업할 때 각각의 컨테이너를 서로 다른 디스크와 연관시키는 것을 고려해야 합니다. 그러면 큰 테이블 스페이스 용량과 병렬 입출력 조작의 이점을 이용할 수 있습니다.

CREATE TABLESPACE문은 데이터베이스 내에서 새 테이블 스페이스를 작성하여 컨테이너를 테이블 스페이스에 지정한 후 테이블 스페이스 정의 및 속성을 카탈로그에 기록합니다. 테이블 스페이스가 작성되면, Extent *크기*는 인접 페이지 수로서 정의됩니다. 하나의 테이블 또는 오브젝트(예: 인덱스)만 단일 Extent의 페이지를 사용할 수 있습니다. 테이블 스페이스에서 작성된 모든 오브젝트는 논리적 테이블 스페이스 주소 맵에서 할당된 Extent입니다. Extent 할당은 스페이스 맵 페이지를 통해 관리됩니다.

논리적 테이블 스페이스 주소 맵에서 첫 번째 Extent는 내부 제어 정보를 포함하는 테이블 스페이스에 대한 헤더이고, 두 번째 Extent는 테이블 스페이스에 대한 *스페이스 맵 페이지*(SMP)의 첫 번째 Extent입니다. SMP Extent는 테이블 스페이스에서 일정한 간격으로 분산됩니다. 각 SMP Extent는 현재 SMP Extent에서 다음 SMP Extent까지 Extent의 비트맵입니다. 비트맵은 사용 중인 중간 Extent를 추적하는 데 사용됩니다.

SMP를 따르는 다음 Extent는 테이블 스페이스에 대한 오브젝트 테이블입니다. 오브젝트 테이블은 테이블 스페이스에 존재하는 사용자 오브젝트 및 해당되는 첫 번째 EMP(Extent Map Page) Extent가 위치하는 곳을 추적하는 내부 테이블입니다. 각 오브젝트에는 논리적 테이블 스페이스 주소 맵에 저장되는 오브젝트의 각 페이지에 대한 맵을 제공하는 고유한 EMP가 있습니다. 그림 9에서는 Extent가 논리적 테이블 스페이스 주소 맵에서 할당되는 방법을 보여줍니다.

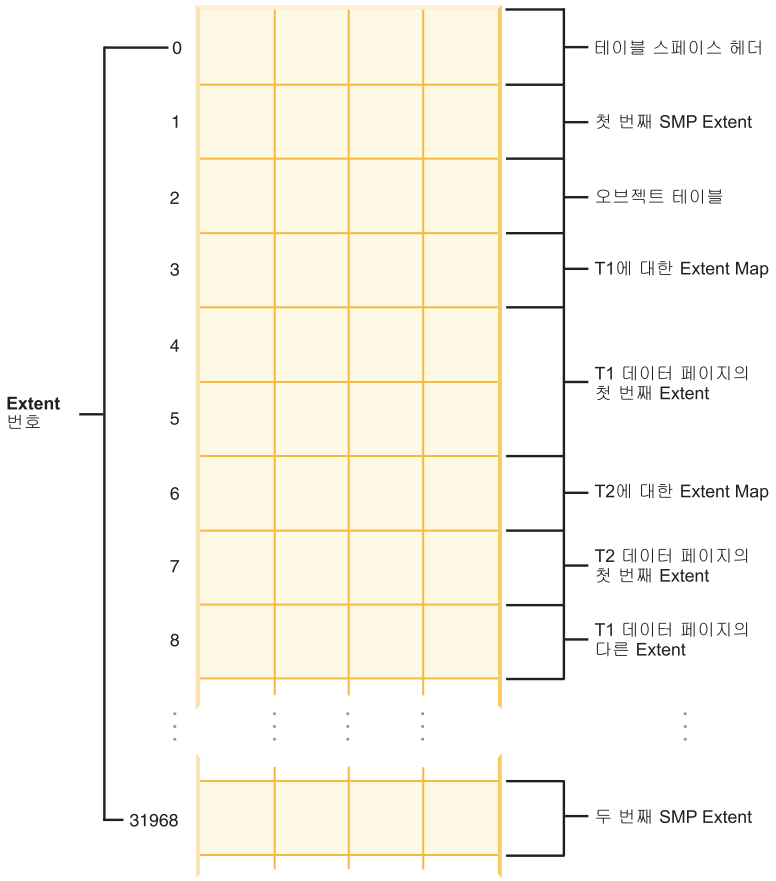


그림 9. 논리적 테이블 스페이스 주소 맵

데이터베이스 관리 테이블 스페이스에 대한 테이블 스페이스 맵:

테이블 스페이스 맵은 DMS 테이블 스페이스의 데이터베이스 관리 프로그램의 내부 표시로, 테이블 스페이스의 논리적 페이지 위치를 물리적 페이지 위치로 변환하여 설명합니다. 이 주제는 테이블 스페이스 맵이 유용한 이유와 테이블 스페이스 맵에 있는 정보의 출처에 대해 설명합니다.

파티션된 데이터베이스에서 DMS 테이블 스페이스의 페이지는 0부터 (N - 1)까지 논리적으로 번호가 지정되며, 여기서 N은 테이블 스페이스에서 사용 가능한 페이지의 수입니다.

테이블 스페이스의 페이지는 Extent 크기에 따라 Extent로 그룹화되며, 테이블 스페이스 관리 측면에서 모든 오브젝트 할당은 Extent를 기준으로 이뤄집니다. 즉, 테이블이 Extent의 페이지 중 절반만 사용해도 해당 오브젝트가 전체 Extent를 사용 중이며 소유하는 것으로 간주됩니다. 디폴트로 하나의 Extent가 컨테이너 태그를 보유하는데 사용되며 이 Extent의 페이지는 데이터를 보유하는데 사용할 수 없습니다. 그러나, DB2_USE_PAGE_CONTAINER_TAG 레지스트리 변수가 설정된 경우, 컨테이너 태그에 대해 한 페이지만 사용됩니다.

149 페이지의 그림 10은 DMS 테이블 스페이스에 대한 논리적 주소 맵을 나타냅니다.

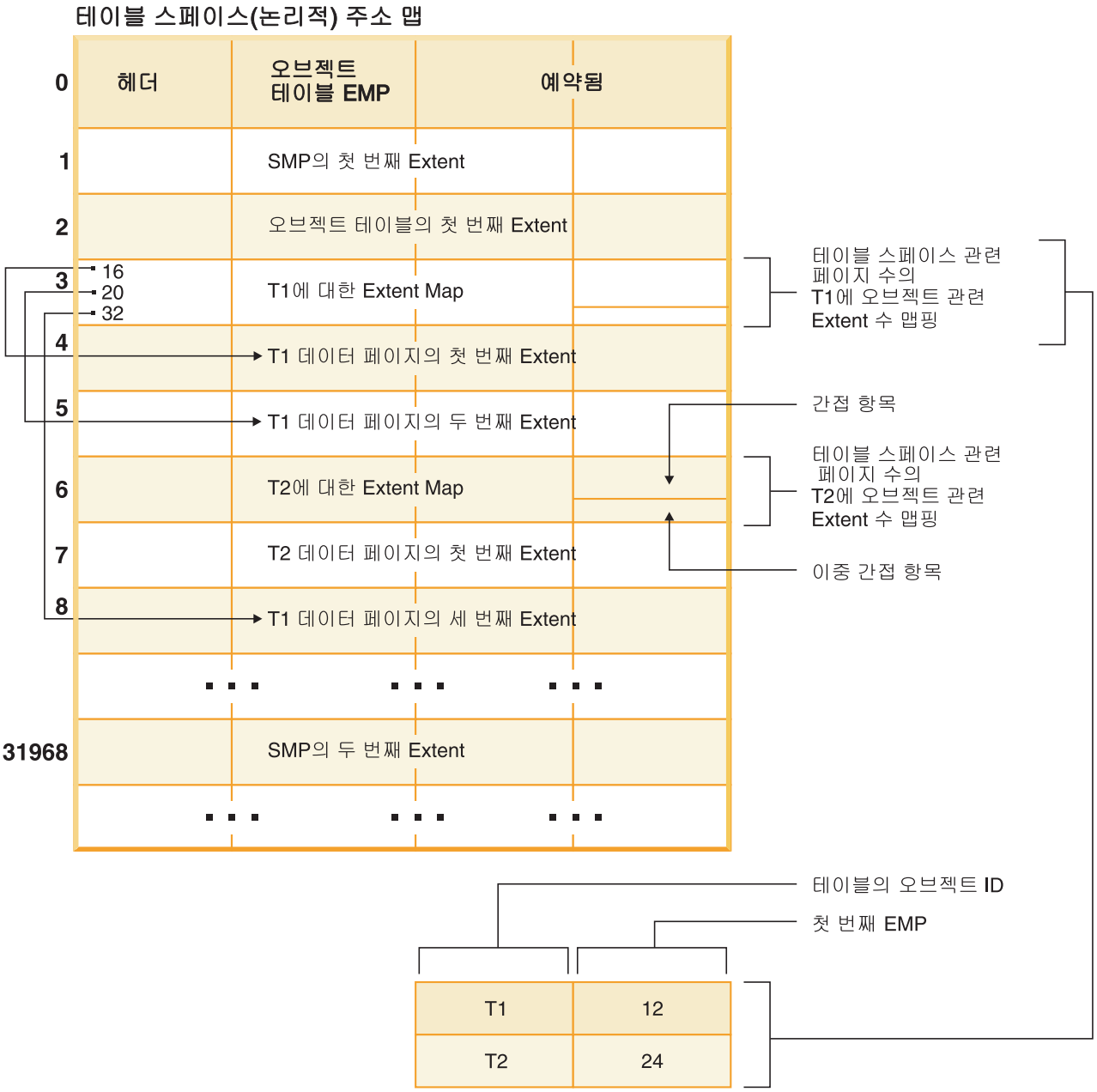


그림 10. DMS 테이블 스페이스

테이블 스페이스 주소 맵에는 EMP(Extent Map Page) 및 스페이스 맵 페이지(SMP)의 두 유형의 맵 페이지가 있습니다.

오브젝트 테이블은 오브젝트 ID를 테이블의 첫 번째 EMP Extent 위치에 맵핑하는 내부 관계 테이블입니다. 이 EMP Extent는 직접 또는 간접으로 오브젝트의 모든 Extent를 정밀하게 표시합니다. 각 EMP에는 하나의 항목 배열이 있습니다. 각 항목은 오브젝트 관련 Extent 번호를 오브젝트 Extent가 위치한 테이블 스페이스 관련 페이지 번호에 맵핑합니다. 직접 EMP 항목은 오브젝트 관련 주소를 직접 테이블 스페이스 관련 주소에 맵핑합니다. 첫 번째 EMP Extent의 마지막 EMP 페이지에는 간접 항목이 있

습니다. 간접 EMP 항목은 EMP 페이지에 맵핑되고, 이 페이지는 다시 오브젝트 페이지에 맵핑됩니다. 첫 번째 EMP Extent의 마지막 EMP 페이지에 있는 최종 16개 항목에 이중 간접 항목이 있습니다.

논리적 테이블 스페이스 주소 맵의 Extent는 테이블 스페이스와 연관되는 컨테이너에서 라운드 로빈으로 스트라이핑됩니다.

컨테이너의 스페이스는 Extent에 따라 할당되기 때문에 전체 Extent를 구성하지 않는 페이지는 사용되지 않습니다. 예를 들어, Extent *크기*가 10인 205페이지의 컨테이너가 있는 경우, 하나의 Extent를 태그에 사용하고 19개의 Extent를 데이터에 사용할 수 있으며 나머지 5개의 페이지는 버려집니다.

DMS 테이블 스페이스에 하나의 컨테이너가 포함된 경우, 논리 페이지 번호를 디스크 상의 실제 위치로 변환하는 것은 0, 1, 2 페이지가 디스크에 동일한 순서로 위치하게 되는 간단한 프로세스입니다.

둘 이상의 컨테이너가 있어 각 컨테이너의 *크기*가 동일한 경우에도 매우 간단한 프로세스입니다. 0에서 (Extent *크기* -1)개의 페이지를 포함하는 테이블 스페이스에서, 첫 번째 Extent는 첫 번째 컨테이너에 위치하고 두 번째 Extent는 두 번째 컨테이너에 위치하는 방식으로 이뤄집니다. 마지막 컨테이너 이후 프로세스가 첫 번째 컨테이너에서 다시 시작하여 반복합니다. 이 순환 프로세스가 데이터를 균형 있게 유지합니다.

*크기*가 서로 다른 컨테이너를 포함하는 테이블 스페이스의 경우, 보다 큰 컨테이너에 있는 추가 스페이스가 활용되지 않으므로 각 컨테이너를 순서대로 하나씩 처리하는 단순 접근 방식을 사용할 수 없습니다. 이러한 이유로 테이블 스페이스 맵이 도입되었습니다. 테이블 스페이스 맵은 테이블 스페이스에서 Extent가 어떻게 배치되는지 표시하기 때문에 물리적 컨테이너의 모든 Extent를 사용할 수 있도록 합니다.

주: 다음 예에서 컨테이너 *크기*는 컨테이너 태그의 *크기*를 고려하지 않습니다. 컨테이너 *크기*는 매우 작으며 설명을 위해 사용된 것일 뿐 권장되는 컨테이너 *크기*가 아닙니다. 이 예에서는 테이블 스페이스 내의 *크기*가 서로 다른 컨테이너를 보여주나, 동일한 컨테이너를 사용하는 것이 좋습니다.

예 1:

테이블 스페이스에는 3개의 컨테이너가 있으며 각 컨테이너에는 80개의 사용 가능한 페이지가 있고 테이블 스페이스의 Extent *크기*는 20입니다. 따라서 각 컨테이너에는 총 12개의 Extent에 대해 4개의 Extent(80 / 20)가 있게 됩니다. 이들 Extent는 151 페이지의 그림 11에 표시된 대로 디스크에 배치됩니다.

그림 11. 3개의 컨테이너와 12개의 Extent가 있는 테이블 스페이스

테이블 스페이스 맵을 보려면, 스냅샷 모니터를 사용하여 테이블 스페이스 스냅샷을 작성하십시오. 예 1에서 세 개의 컨테이너 크기가 동일한 테이블 스페이스 맵은 이와 같이 표시됩니다.

```
Range   Stripe Stripe Max      Max   Start   End    Adj.  Containers
Number  Set    Offset Extent   Page  Stripe  Stripe
[0]     [0]       0       11        239        0          3      0     3 (0, 1, 2)
```

*범위*는 모든 연속된 스트라이프 범위에 동일한 컨테이너 세트가 포함되는 맵의 일부입니다. 예 1에서 모든 스트라이프(0 - 3)에는 동일한 3개의 컨테이너 세트(0, 1, 2)가 있으므로 하나의 범위로 간주됩니다.

테이블 스페이스 맵의 표제는 범위 번호, 스트라이프 세트, 스트라이프 오프셋, 범위로 지정된 최대 Extent 번호, 범위로 지정된 최대 페이지 번호, 시작 스트라이프, 끝 스트라이프, 범위 조정 및 컨테이너 목록입니다. 이러한 내용에 대해서는 예 2에서 보다 자세하게 설명합니다.

152 페이지의 그림 12와 같이 이 테이블 스페이스는 각 세로줄이 컨테이너에 해당하고 각 가로줄이 *스트라이프*에 해당하며 각 셀 번호가 Extent에 해당하는 다이어그램으로 나타낼 수도 있습니다.

그림 12. 3개의 컨테이너와 12개의 Extent가 있으며 스트라이프가 강조표시된 테이블 스페이스

예 2

테이블 스페이스에는 두 개의 컨테이너가 있습니다. 첫 번째는 크기가 100페이지이고 두 번째는 크기가 50페이지이며 Extent 크기는 25입니다. 이것은 첫 번째 컨테이너에 4개의 Extent가 있으며 두 번째 컨테이너에 두 개의 Extent가 있음을 의미합니다. 테이블 스페이스는 그림 13과 같은 다이어그램으로 나타낼 수 있습니다.

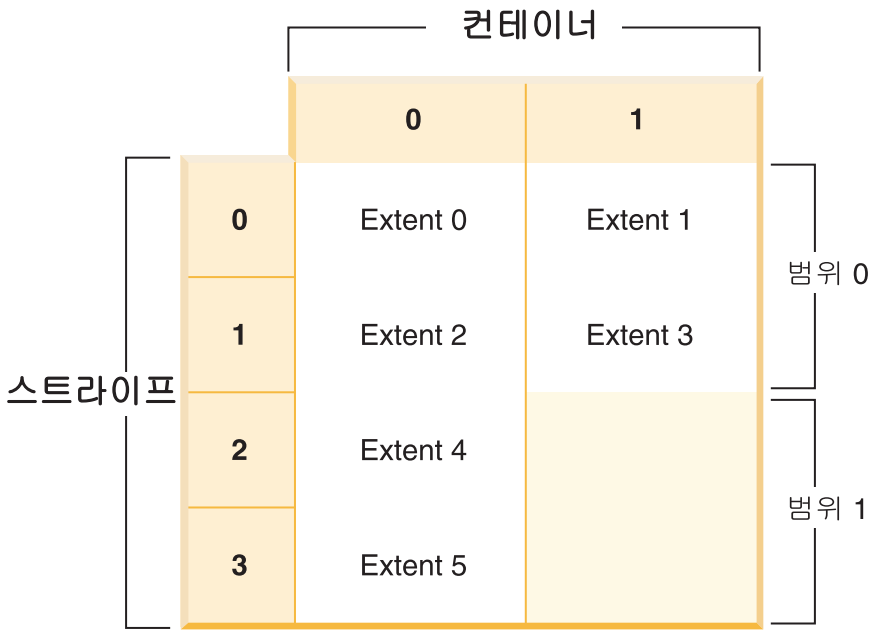


그림 13. 2개의 컨테이너가 있고 범위가 강조표시된 테이블 스페이스

스트라이프 0 및 1에는 두 개의 컨테이너(0과 1)가 모두 포함되지만 스트라이프 2 및 3에는 첫 번째 컨테이너(0)만 포함됩니다. 이들 각 스트라이프 세트가 범위입니다. 테이블 스페이스 스냅샷에 표시된 바와 같이, 해당 테이블 스페이스 맵은 다음과 같습니다.

```
Range   Stripe Stripe   Max      Max   Start   End    Adj.  Containers
Number  Set    Offset  Extent    Page  Stripe  Stripe
[0]     [0]        0         3         99        0        1       0      2 (0, 1)
[1]     [0]        0         5        149        2        3       0      1 (0)
```

첫 번째 범위에 네 개의 Extent가 있으므로, 이 범위에서 지정된 최대 Extent 번호(Max Extent)는 3입니다. 각 Extent는 25페이지를 가지므로 첫 번째 범위에 100페이지가 있습니다. 페이지 번호 지정도 0부터 시작하므로 이 범위에 지정된 최대 페이지 번호(최대 페이지)는 99입니다. 이 범위의 첫 번째 스트라이프(시작 스트라이프)는 0이고 범위의 마지막 스트라이프(끝 스트라이프)는 스트라이프 1입니다. 이 범위에는 두 개의 컨테이너가 있으며 0과 1입니다. 스트라이프 오프셋은 스트라이프 세트의 첫 번째 스트라이프이며, 이 경우에는 단 하나의 스트라이프 세트가 있기 때문에 0입니다. 범위 조정은 테이블 스페이스에서 데이터를 재조정할 때 사용되는 오프셋입니다. (테이블 스페이스에 스페이스가 추가되거나 삭제될 때 재조정이 발생할 수 있습니다.) 재조정이 이루어지지 않는 경우 이 값은 항상 0입니다.

두 번째 범위에는 두 개의 Extent가 있으며 이전 범위에서 지정된 최대 Extent 번호가 3이기 때문에 이 범위에서 지정되는 최대 Extent 번호는 5입니다. 두 번째 범위에 50페이지(2 Extent * 25페이지)가 있고 이전 범위에 지정된 최대 페이지 번호가 99이므로 이 범위에 지정된 최대 페이지 번호는 149입니다. 이 범위는 스트라이프 2에서 시작하여 스트라이프 3에서 종료합니다.

DMS 테이블 스페이스의 자동 크기 조정:

자동 *크기 조정*을 위해 파일 컨테이너를 사용하는 데이터베이스 관리(DMS) 테이블 스페이스를 사용 가능하게 하면 기존 컨테이너를 확장하여 데이터베이스 관리 프로그램이 자동으로 전체 테이블 스페이스 조건을 처리할 수 있습니다.

DMS 테이블 스페이스는 파일 컨테이너 또는 원시 디바이스 컨테이너로 구성되며 그 *크기*는 컨테이너가 테이블 스페이스에 지정될 때 설정됩니다. 컨테이너의 모든 스페이스가 사용되었을 때 테이블 스페이스가 가득 찬 것으로 간주됩니다. 그러나 SMS 테이블 스페이스와는 달리 ALTER TABLESPACE문을 사용하여 수동으로 컨테이너를 추가하거나 확장할 수 있으며, 따라서 추가 스토리지 스페이스가 테이블 스페이스에 제공될 수 있습니다. DMS 테이블 스페이스는 또한 *자동 크기 조정*이라는 기능을 갖습니다. 자동으로 *크기* 조정될 수 있는 DMS 테이블 스페이스에서 스페이스가 소비될 때 데이터베이스 관리 프로그램이 하나 이상의 파일 컨테이너를 확장하여 테이블 스페이스의 *크기*를 늘립니다.

DMS 테이블 스페이스에 대한 자동 *크기* 조정 기능은 자동 스토리지 테이블 스페이스의 성능과 관련되지만 서로 다릅니다. 자세한 정보는 172 페이지의 『SMS, DMS 및 자동 스토리지 테이블 스페이스의 비교』의 내용을 참조하십시오.

자동 *크기* 조정 기능 사용 및 사용 안함

디폴트로 자동 *크기* 조정 기능은 DMS 테이블 스페이스에 사용 가능하지 않습니다. 다음 명령문은 자동 *크기* 조정을 사용하지 않고 DMS 테이블 스페이스를 작성합니다.

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
    USING (FILE '/db2files/DMS1' 10 M)
```

자동 *크기* 조정 기능을 사용하려면 CREATE TABLESPACE문에 대해 AUTORESIZE YES절을 지정하십시오.

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
    USING (FILE '/db2files/DMS1' 10 M) AUTORESIZE YES
```

또한 AUTORESIZE절과 함께 ALTER TABLESPACE문을 사용하여 DMS 테이블 스페이스를 작성한 후 자동 *크기* 조정을 사용하거나 사용 안할 수 있습니다.

```
ALTER TABLESPACE DMS1 AUTORESIZE YES
ALTER TABLESPACE DMS1 AUTORESIZE NO
```

두 개의 다른 속성 MAXSIZE 및 INCREASESIZE는 자동 *크기* 조정 테이블 스페이스와 연관됩니다.

최대 *크기*(MAXSIZE)

CREATE TABLESPACE문의 MAXSIZE절은 테이블 스페이스에 대한 최대 *크기*를 정의합니다. 예를 들어, 다음 명령문은 100MB까지 증가할 수 있는(데이터베이스에 다중 데이터베이스 파티션이 있는 경우 데이터베이스 파티션마다) 테이블 스페이스를 작성합니다.

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
    USING (FILE '/db2files/DMS1' 10 M)          AUTORESIZE YES MAXSIZE 100 M
```

MAXSIZE NONE절은 테이블 스페이스에 대한 최대 한계가 없음을 지정합니다. 테이블 스페이스는 파일 시스템 한계 및 테이블 스페이스 한계에 도달할 때까지 커질 수 있습니다(*SQL 참조서*의 『SQL 및 XML 한계』를 참조). MAXSIZE절을 지정하지 않는 경우 자동 *크기* 조정 기능이 사용될 때는 최대 한계가 없습니다.

다음 예에서 보는 것처럼 자동 *크기* 조정이 이미 사용되는 테이블 스페이스에 대한 MAXSIZE 값을 변경하려면 ALTER TABLESPACE문을 사용하십시오.

```
ALTER TABLESPACE DMS1 MAXSIZE 1 G
ALTER TABLESPACE DMS1 MAXSIZE NONE
```

최대 크기를 지정하는 경우 데이터베이스 관리 프로그램이 컨테이너 성장을 일관성 있게 유지하려고 하기 때문에 데이터베이스 관리 프로그램이 강제하는 실제 값은 지정된 값보다 약간 더 작을 수 있습니다.

크기 늘리기(INCREASESIZE)

CREATE TABLESPACE문의 INCREASESIZE 절은 테이블 스페이스 안에 여유 Extent가 없는데 하나 이상의 Extent 요청이 있는 경우 테이블 스페이스를 증가시키기 위해 사용되는 스페이스 양을 정의합니다. 다음 예에서 보는 것처럼 값을 명시적 크기 또는 백분율로서 지정할 수 있습니다.

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
    USING (FILE '/db2files/DMS1' 10 M)          AUTORESIZE YES INCREASESIZE 5 M

CREATE TABLESPACE DMS1 MANAGED BY DATABASE
    USING (FILE '/db2files/DMS1' 10 M)          AUTORESIZE YES INCREASESIZE 50 PERCENT
```

백분율 값은 테이블 스페이스가 커져야할 때마다 늘릴 양이 계산됨을 의미합니다. 즉, 성장은 해당 특정 시점에서 테이블 스페이스 크기의 백분율을 기초로 합니다. 예를 들어 테이블 스페이스의 크기가 20MB이고 INCREASESIZE 값이 50%인 경우, 테이블 스페이스는 처음에는(30MB의 크기까지) 10MB씩 증가하고 다음에는 15MB씩 증가합니다.

자동 크기 조정 기능을 사용할 때 INCREASESIZE절을 지정하지 않는 경우 데이터베이스 관리 프로그램이 사용할 적절한 값을 판별하며, 이것은 테이블 스페이스의 수명 동안 변하지 않을 수 있습니다. AUTORESIZE 및 MAXSIZE에서와 같이, ALTER TABLESPACE문을 사용하여 INCREASESIZE의 값을 변경할 수 있습니다.

크기 증가를 지정하는 경우 데이터베이스 관리 프로그램이 사용할 실제 값은 사용자가 제공하는 값과 약간 다를 수 있습니다. 사용되는 값에 대한 이러한 조정은 테이블 스페이스의 컨테이너 간에 증가를 일관적으로 유지하도록 수행됩니다.

DMS 테이블 스페이스에서 AUTORESIZE 사용에 대한 제한사항

- 원시 디바이스 컨테이너를 사용하는 테이블 스페이스에 대해서는 이 기능을 사용할 수 없으며, 자동으로 크기 조정될 수 있는 테이블 스페이스에 원시 디바이스 컨테이너를 추가할 수 없습니다. 이 조작을 시도하면 오류(SQL0109N)가 발생합니다. 원시 디바이스 컨테이너를 추가해야 하는 경우 먼저 자동 크기 조정 기능을 사용 안해야 합니다.
- 자동 크기 조정 기능을 사용 안하는 경우 나중에 사용하면 INCREASESIZE 및 MAXSIZE와 연관되는 값은 보존되지 않습니다.
- 경로 재지정된 리스토어 조작은 원시 디바이스 컨테이너를 포함하도록 컨테이너 정의를 변경할 수 없습니다. 이런 종류의 조작을 시도하면 오류(SQL0109N)가 발생합니다.

- 데이터베이스 관리 프로그램이 자동으로 테이블 스페이스를 늘리는 방법을 제한하는 것 외에, 최대 *크기*도 사용자가 테이블 스페이스를 늘릴 수 있는 Extent를 제한합니다. 테이블 스페이스에 스페이스를 추가하는 조작을 수행하는 경우 *크기* 결과는 최대 *크기*보다 작거나 같아야 합니다. ALTER TABLESPACE문의 ADD, EXTEND, RESIZE 또는 BEGIN NEW STRIPE SET절을 사용하여 스페이스를 추가할 수 있습니다.

테이블 스페이스 확장 방식

AUTORESIZE가 사용될 때 데이터베이스 관리 프로그램은 모든 기존 스페이스가 사용되었고 추가 스페이스가 요청될 때 테이블 스페이스의 *크기*를 늘리려고 합니다. 데이터베이스 관리 프로그램은 테이블 스페이스에서 데이터의 재조정이 발생하지 않도록 테이블 스페이스에서 확장할 수 있는 컨테이너를 판별합니다. 데이터베이스 관리 프로그램은 테이블 스페이스 맵의 마지막 범위에 존재하는 컨테이너만 확장하고(맵은 테이블 스페이스에 대한 스토리지 레이아웃을 설명함 - 자세한 정보는 147 페이지의 『데이터베이스 관리 테이블 스페이스에 대한 테이블 스페이스 맵』의 내용을 참조) 컨테이너를 동일한 양만큼 확장합니다.

예를 들어 다음 명령문을 참조하십시오.

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE
    USING (FILE 'C:₩TS1CONT' 1000, FILE 'D:₩TS1CONT' 1000,
                  FILE 'E:₩TS1CONT' 2000, FILE 'F:₩TS1CONT' 2000)
    EXTENTSIZE 4
    AUTORESIZE YES
```

데이터베이스 관리 프로그램이 메타데이터에 대해 각 컨테이너의 작은 분할 영역(1개 Extent)을 사용함에 유의하십시오. 다음은 CREATE TABLESPACE문에 기초하여 테이블 스페이스에 작성된 테이블 스페이스 맵입니다. (테이블 스페이스 맵은 테이블 스페이스 테이블의 출력의 일부입니다.)

```
Table space map:

Range  Stripe Stripe  Max         Max  Start  End    Adj.   Containers
    Number Set    Offset  Extent      Page Stripe Stripe
[   0] [   0]      0     995      3983      0    248   0    4 (0,1,2,3)
[   1] [   0]      0    1495      5983    249    498   0    2 (2,3)
```

테이블 스페이스 맵은 ID 2 또는 3(E:₩TS1CONT 및 F:₩TS1CONT)을 가진 컨테이너가 맵의 마지막 범위에 있는 유일한 컨테이너임을 나타냅니다. 따라서 데이터베이스 관리 프로그램이 이 테이블 스페이스의 컨테이너를 자동으로 확장하면 두 개의 컨테이너만 확장됩니다.

주: 모두 동일한 *크기*의 컨테이너를 갖는 테이블 스페이스를 작성하는 경우 맵에는 한 개의 범위만 존재합니다. 이러한 경우 데이터베이스 관리 프로그램이 각 컨테이너를 확장합니다. 일부의 컨테이너만 확장되지 않게 하려면 동일한 *크기*의 컨테이너로 테이블 스페이스를 작성하십시오.

앞에서 설명한 대로, 테이블 스페이스의 최대 크기에 대한 한계를 지정하거나 증가를 제한하지 않는 NONE의 값을 지정할 수 있습니다. NONE을 지정하거나 한계를 지정하지 않는 경우 상한은 파일 시스템 한계나 테이블 스페이스 한계에 의해 정의됩니다. 데이터베이스 관리 프로그램은 테이블 스페이스 크기를 상한 이상으로 늘리려고 하지 않습니다. 그러나 한계에 도달하기 전에 파일 시스템이 가득 차서 컨테이너 증가 시도가 실패할 수 있습니다. 이 경우 데이터베이스 관리 프로그램은 테이블 스페이스 크기를 더 이상 늘리지 않으며 응용프로그램으로 스페이스 부족 상태를 리턴합니다. 이 상황을 해결하는 방법에는 다음 두 가지가 있습니다.

- 가득 찬 파일 시스템에서 사용 가능한 스페이스 양을 증가시키십시오.
- 문제의 컨테이너가 더 이상 테이블 스페이스 맵의 마지막 범위에 있지 않도록 테이블 스페이스에 대해 컨테이너 조작을 수행하십시오. 이를 수행하는 가장 쉬운 방법은 새 컨테이너 세트를 갖는 테이블 스페이스에 새 스트라이프 세트를 추가하는 것이며, 우수 사례는 컨테이너가 모두 동일한 크기가 되도록 하는 것입니다. BEGIN NEW STRIPE SET절을 갖는 ALTER TABLESPACE문을 사용하여 새 스트라이프 세트를 추가할 수 있습니다. 새 스트라이프 세트를 추가하여 새 범위가 테이블 스페이스 맵에 추가됩니다. 새 범위를 사용하면 데이터베이스 관리 프로그램이 자동으로 확장하려는 컨테이너는 이 새 스트라이프 세트에 있으며, 이전 컨테이너는 변경되지 않습니다.

  주: 사용자가 시작한 컨테이너 조작이 보류 중이거나 후속 재조정이 진행 중인 경우, 조작이 커미트되거나 입력이 완료될 때까지 자동 크기 조정 기능은 사용 불가능합니다.

예를 들어 DMS 테이블 스페이스의 경우 테이블 스페이스에 동일한 크기이고 각각이 자체 파일 시스템에 상주하는 컨테이너가 세 개 있다고 가정하십시오. 테이블 스페이스에 대해 작업이 완료되면 데이터베이스 관리 프로그램이 자동으로 이러한 세 개의 컨테이너를 확장합니다. 결국 파일 시스템 중 하나가 가득 차게 되고 해당 컨테이너를 더이상 증가시킬 수 있습니다. 파일 시스템에 더 이상의 여유 공간을 만들 수 없는 경우 문제의 컨테이너가 더 이상 테이블 스페이스 맵의 마지막 범위에 있지 않도록 테이블 스페이스에 대해 컨테이너 조작을 수행해야 합니다. 이 경우 두 개의 컨테이너(아직 스페이스가 있는 각 파일 시스템에 하나씩)를 지정하는 새 스트라이프 세트를 추가하거나, 이보다 추가 컨테이너를 지정할 수 있습니다(역시, 추가 중인 각 컨테이너가 같은 크기이며 사용 중인 각 파일 시스템에 증가에 필요한 충분한 여유가 있어야 함). 데이터베이스 관리 프로그램이 테이블 스페이스 크기를 증가시킬 때 이제 이전 컨테이너를 확장하려고 시도하는 대신 이 새 스트라이프 세트의 컨테이너를 확장하려고 합니다.

모니터링

DMS 테이블 스페이스에 대한 자동 크기 조정에 관한 정보는 테이블 스페이스 모니터 스냅샷 출력의 일부로 표시됩니다. 다음 샘플에서 보는 것처럼 증가 크기 및 최대 크기 값이 출력에 포함됩니다.

```
자동 크기 조정 사용 가능                = 예 또는 아니오
현재 테이블 스페이스 크기(바이트)       = ###
최대 테이블 스페이스 크기(바이트)       = ### 또는 없음
크기 늘리기(바이트)                     = ###
크기 늘리기(퍼센트)                     = ###
마지막 크기 조정 완료 시간              = DD/MM/YYYY HH:MM:SS.SSSSSS
마지막 크기 조정 시도 실패              = 예 또는 아니오
```

자동 스토리지 테이블 스페이스

자동 스토리지 테이블 스페이스를 사용하여 자동으로 스토리지를 관리합니다. 데이터베이스 관리 프로그램은 데이터베이스와 연관된 스토리지 경로로 인해 부과되는 한계까지 필요에 따라 컨테이너를 작성하고 확장합니다.

데이터베이스에서 자동 스토리지가 사용 가능한 경우 달리 지정하지 않는 한 작성하는 모든 테이블 스페이스도 자동 스토리지 테이블 스페이스로 관리됩니다. 자동 스토리지 테이블 스페이스를 사용하면 컨테이너 정의를 제공할 필요가 없습니다. 데이터베이스 관리 프로그램이 컨테이너 작성 및 확장을 관리하여 데이터베이스에 할당된 스토리지를 사용합니다. 데이터베이스에 스토리지를 추가하는 경우 기존 컨테이너가 최대 용량에 도달하면 새 컨테이너가 자동으로 작성됩니다. 새로 추가된 스토리지를 즉시 사용하려면 데이터를 확장된 새 컨테이너 및 스트라이프 세트에 재할당하여 테이블 스페이스를 재조정할 수 있습니다. 또는 입출력 병렬 처리를 반드시 수행할 필요는 없으며 테이블 스페이스에 용량을 추가하려는 경우에는 재조정을 설정할 수 있습니다. 이 경우, 새 스토리지가 필요하므로 새 스트라이프 세트가 작성됩니다.

CREATE TABLESPACE 명령을 사용하여 자동 스토리지 데이터베이스에 자동 스토리지 테이블 스페이스를 작성할 수 있습니다. 디폴트로 자동 스토리지가 사용 가능한 데이터베이스의 새 테이블 스페이스가 자동 스토리지 테이블 스페이스이므로 MANAGED BY AUTOMATIC STORAGE절은 선택적입니다. 또한 자동 스토리지 테이블 스페이스 작성 시 *초기 크기*, 테이블 스페이스가 꽉 찰 경우 테이블 스페이스가 증가되는 *크기* 및 테이블 스페이스가 늘어날 수 있는 최대 *크기* 등과 같은 옵션을 지정할 수 있습니다. 다음은 자동 스토리지 테이블 스페이스를 작성하는 명령문의 몇 가지 예입니다.

```
CREATE TABLESPACE TS1
CREATE TABLESPACE TS2 MANAGED BY AUTOMATIC STORAGE
CREATE TEMPORARY TABLESPACE TEMPTS
CREATE USER TEMPORARY TABLESPACE USRTMP MANAGED BY AUTOMATIC STORAGE
CREATE LONG TABLESPACE LONGTS
CREATE TABLESPACE TS3 INITIALSIZE 8K INCREASESIZE 20 PERCENT MANAGED BY AUTOMATIC STORAGE
CREATE TABLESPACE TS4 MAXSIZE 2G
```

이들 각 예에서는 해당 테이블 스페이스가 작성되고 있는 데이터베이스를 자동 스토리지 데이터베이스로 가정합니다. 자동 스토리지에 사용할 수 없는 데이터베이스에 테이블 스페이스를 작성할 때 MANAGED BY AUTOMATIC STORAGE절을 사용할 수 없습니다. 다음 중 하나를 수행해야 합니다.

- CREATE TABLESPACE문의 MANAGED BY SYSTEM 또는 MANAGED BY DATABASE 절을 지정하십시오. 이러한 절을 사용하면 시스템 관리 스페이스(SMS)

테이블 스페이스 또는 데이터베이스 관리 스페이스(DMS) 테이블 스페이스가 각각 작성됩니다. 두 경우 모두에서 컨테이너의 명시적 목록을 제공해야 합니다.

- 데이터베이스를 자동 스토리지 데이터베이스로 변환한 후 다시 자동 스토리지 테이블 스페이스를 작성해보십시오.

자동 스토리지 테이블 스페이스가 스토리지 확장을 관리하는 방법:

*자동 스토리지 테이블 스페이스*를 사용 중인 경우 데이터베이스 관리 프로그램이 필요한 대로 컨테이너를 작성하고 확장합니다. 데이터베이스에 스토리지를 추가하는 경우 새 컨테이너가 자동으로 작성됩니다. 그러나 새 스토리지 스페이스가 사용되는 방법은 테이블 스페이스를 재조정(REBALANCE)하는지 여부에 따라 다릅니다.

자동 스토리지 테이블 스페이스가 작성될 때 데이터베이스 관리 프로그램이 자동 스토리지 데이터베이스(스페이스가 허용하는)의 각 스토리지 경로에 컨테이너를 작성합니다. 테이블 스페이스의 모든 스페이스가 소비된 후에는 데이터베이스 관리 프로그램이 자동으로 기존 컨테이너를 확장하거나 컨테이너의 새 스트라이프 세트를 추가하여 테이블 스페이스의 *크기*를 늘립니다.

자동 테이블 스페이스에 대한 스토리지는 데이터베이스 레벨에서 관리됩니다. 즉, DMS 테이블 스페이스에서와 같이 테이블 스페이스가 아니라 *데이터베이스*에 스토리지를 추가합니다. 스토리지를 데이터베이스에 추가할 때 자동 스토리지 기능이 데이터를 수용하기 위해 필요한 대로 새 컨테이너를 작성합니다. 그러나 이미 존재하는 테이블 스페이스는 새 경로의 스토리지를 바로 소비하기 시작하지 않습니다. 테이블 스페이스가 커져야할 때 데이터베이스 관리 프로그램은 먼저 테이블 스페이스의 마지막 *범위*의 컨테이너를 확장합니다. 범위는 주어진 스트라이프 세트의 모든 컨테이너입니다. 이것이 성공하면 응용프로그램은 새 스페이스 사용을 시작합니다. 그러나 예를 들어 파일 시스템의 하나 이상이 가득 찰 때 발생하는 것처럼 컨테이너를 확장하려는 시도가 실패하는 경우, 데이터베이스 관리 프로그램은 컨테이너의 새 스트라이프 세트를 작성합니다. 이 시점에서만 데이터베이스 관리 프로그램이 테이블 스페이스에 대해 새로 추가되는 스토리지 경로 사용을 고려합니다. 160 페이지의 그림 14는 이 프로세스를 나타냅니다.

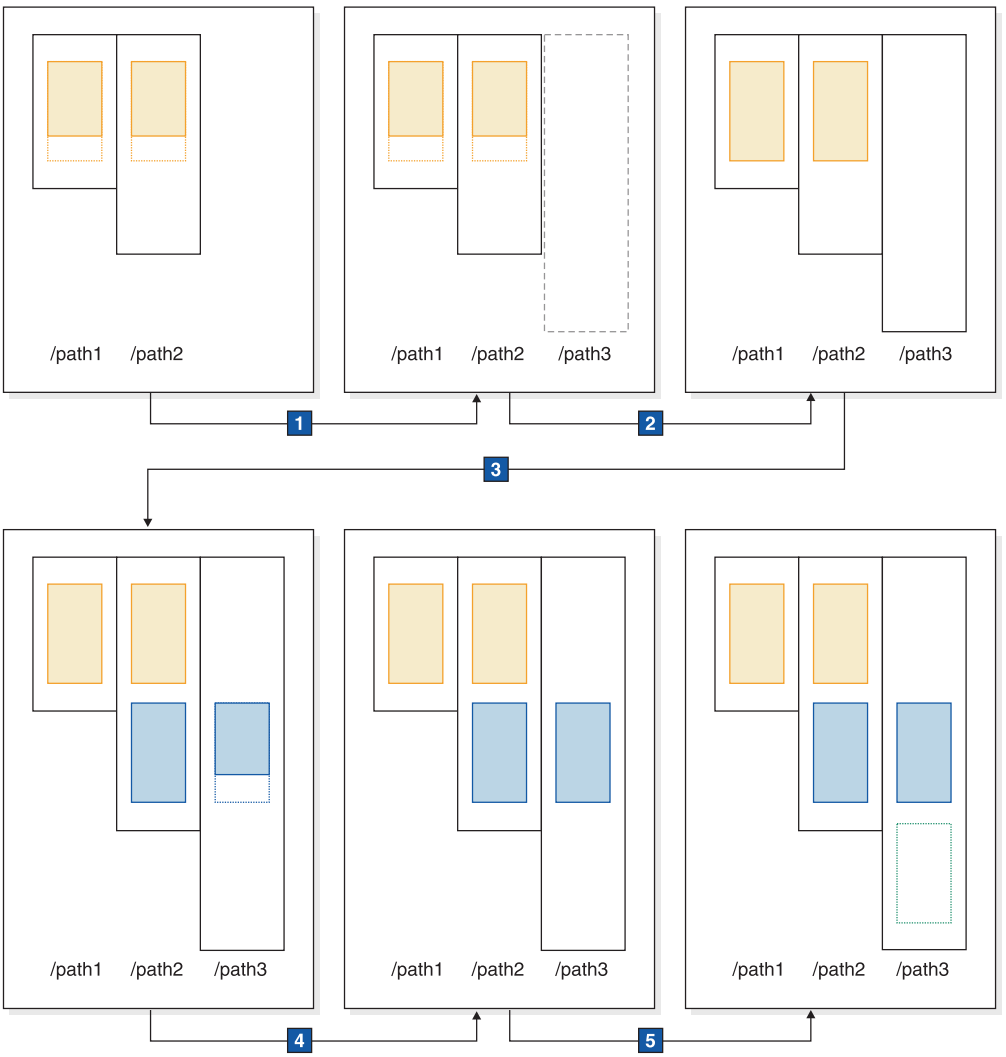


그림 14. 자동 스토리지가 필요한 대로 컨테이너를 추가하는 방법

앞의 다이어그램에서,

1. 테이블 스페이스는 아직 최대 용량에 도달하지 않은 두 컨테이너로 시작합니다. 새 스토리지 경로가 ADD STORAGE절을 갖는 ALTER DATABASE문을 사용하여 데이터베이스에 추가됩니다. 그러나 새 스토리지 경로가 아직 사용되지 않고 있습니다.
2. 두 개의 원래 컨테이너가 최대 용량에 도달합니다.
3. 새 컨테이너 스트라이프 세트가 추가되고 데이터로 채워지기 시작합니다.
4. 새 스트라이프 세트의 컨테이너가 최대 용량에 도달합니다.
5. 컨테이너가 확장될 영역이 없기 때문에 새 스트라이프 세트가 추가됩니다.

자동 스토리지 테이블 스페이스가 새로 추가된 스토리지 경로를 사용하여 즉시 시작하게 하려는 경우 ALTER TABLESPACE 명령의 REBALANCE절을 사용하여 재조정을 수행할 수 있습니다. 테이블 스페이스를 재조정하는 경우 데이터가 새로 추가된 스

토리지의 컨테이너 및 스트라이프 세트에 재할당됩니다. 이는 그림 15에 표시되어 있습니다.

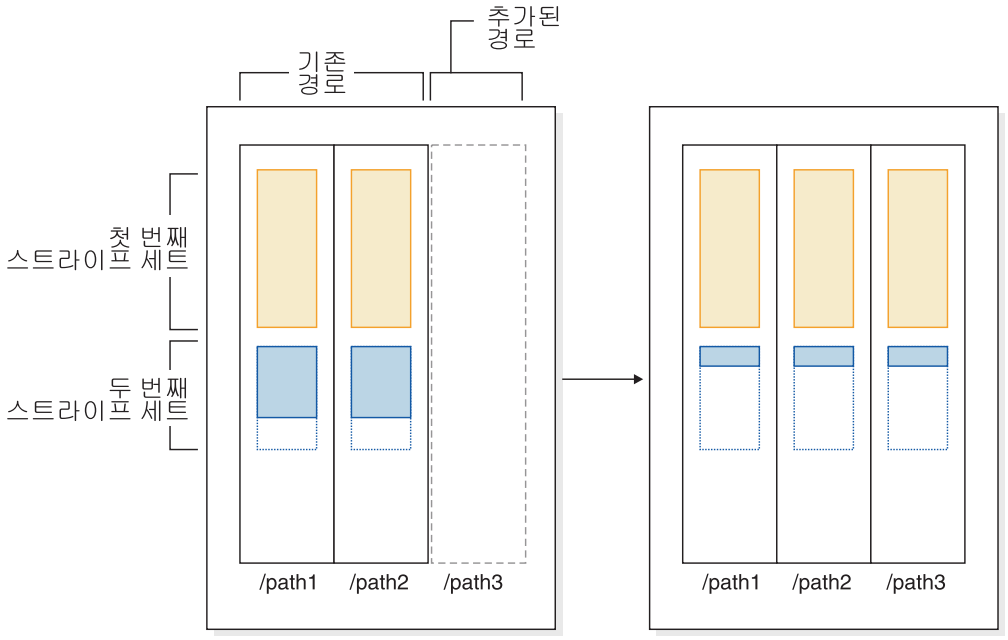


그림 15. 새 스토리지 추가 및 테이블 스페이스 재조정 결과

이 예에서 새 스트라이프 세트가 작성되는 대신 재조정이 필요한 대로 컨테이너를 작성하여 기존 스트라이프 세트를 새 스토리지 경로로 확장한 후 모든 컨테이너에 데이터를 재할당합니다.

자동 스토리지 테이블 스페이스의 컨테이너 이름:

자동 스토리지 테이블 스페이스에 대한 컨테이너 이름이 데이터베이스 관리 프로그램에 의해 지정되지만, LIST TABLESPACE CONTAINERS 또는 GET SNAPSHOT FOR TABLESPACES 명령 같은 명령을 실행하면 볼 수 있습니다. 이 주제는 컨테이너 이름에 사용되는 규칙을 설명하여 해당 이름이 나타날 때 인식할 수 있도록 합니다.

자동 스토리지 테이블 스페이스의 컨테이너에 지정되는 이름은 다음과 같은 구조를 갖습니다.

```
storage path/instance name/NODE####/database name/T#######/C#######.EXT
```

각 항목에 대한 설명은 다음과 같습니다.

storage path
: 데이터베이스와 연관된 스토리지 경로

instance name
: 데이터베이스가 작성된 인스턴스

database name

데이터베이스 이름

NODE####

데이터베이스 파티션 번호(예: NODE0000)

T#######

테이블 스페이스 ID(예: T0000003)

C#######

컨테이너 ID(예: C0000012)

EXT 저장될 데이터의 유형을 기초로 하는 확장자:

CAT 시스템 카탈로그 테이블 스페이스

TMP 시스템 임시 테이블 스페이스

UTM 사용자 임시 테이블 스페이스

USR 사용자 또는 일반 테이블 스페이스

LRG 대형 테이블 스페이스

예

예를 들어 자동 스토리지 테이블 스페이스 TBSAUTO가 SAMPLE 데이터베이스에 작성되었다고 가정하십시오. LIST TABLESPACES 명령이 실행되면 테이블 스페이스 ID **10**을 갖는 것으로 표시됩니다.

```
테이블 스페이스 ID                    = 10
이름                                  = TBSAUTO
유형                                  = 데이터베이스 관리 스페이스
내용                                  = 모든 영구 데이터. 대형 테이블 스페이스.
상태                                  = 0x0000
  세부사항 설명:
    정상
```

이제 ID가 10인 테이블 스페이스에 대해 LIST TABLESPACE CONTAINERS 명령을 실행하면 이 테이블 스페이스에 대한 컨테이너에 지정되는 이름을 볼 수 있습니다.

```
LIST TABLESPACE CONTAINERS FOR 10 SHOW DETAIL

          테이블 스페이스 10에 대한 테이블 스페이스 컨테이너

컨테이너 ID                           = 0
이름                                  = D:¥DB2¥NODE0000¥SAMPLE¥T0000010¥C0000000.LRG
유형                                  = 파일
전체 페이지 수                        = 4096
사용 가능한 페이지 수                 = 4064
액세스 가능                           = 예
```

이 예에서 이 테이블 스페이스에 대한 (위에서 컨테이너 ID 0)을 갖는 한 컨테이너의 이름은 다음과 같습니다.

```
D:¥DB2¥NODE0000¥SAMPLE¥T0000010¥C0000000.LRG
```

자동 스토리지를 사용하도록 테이블 스페이스 변환:

데이터베이스의 데이터베이스 관리 스페이스(DMS) 테이블 스페이스의 전부 또는 일부를 자동 스토리지를 사용하도록 변환할 수 있습니다. 자동 스토리지 사용은 스토리지 관리 태스크를 단순하게 만듭니다.

시작하기 전에

데이터베이스에서 자동 스토리지를 사용할 수 있으며 자동 스토리지와 함께 사용할 하나 이상의 스토리지 경로가 정의되었는지 확인하십시오. 이를 수행하려면 ALTER DATABASE문을 사용하십시오.

프로시저

DMS 테이블 스페이스를 자동 스토리지를 사용하도록 변환하려면 다음 방법 중 하나를 사용하십시오.

- 단일 테이블 스페이스를 변경하십시오. 이 방법은 테이블 스페이스를 온라인으로 유지하지만 비자동 스토리지 컨테이너에서 새로운 자동 스토리지 컨테이너로 데이터를 이동하는 시간이 걸리는 재조정 조작이 포함됩니다.
  1. 변환하려는 테이블 스페이스에 대한 MANAGED BY AUTOMATIC STORAGE 절을 지정하고 ALTER TABLESPACE문을 발행하십시오.
  2. 이번에는 REBALANCE 옵션을 지정하고 ALTER TABLESPACE문을 다시 발행하십시오. 이 옵션은 사용자 정의 컨테이너를 제거하므로 모든 테이블 스페이스 컨테이너가 자동 스토리지에 의해 관리됩니다.

     이제 REBALANCE 옵션을 지정하지 않고 나중에 REDUCE 옵션과 함께 ALTER TABLESPACE문을 발행하는 경우 자동 스토리지 컨테이너가 제거됩니다. 이 문제에서 복구하려면 REBALANCE 옵션을 지정하여 ALTER TABLESPACE문을 발행하십시오.
- 경로 재지정 리스토어 작업을 사용하십시오. 이 방법으로 단일 테이블 스페이스를 변환 중인 경우 조작이 진행되는 중에는 테이블 스페이스에 액세스할 수 없습니다. 다중 테이블 스페이스를 변환 중인 경우 조작이 진행되는 동안 전체 데이터베이스에 액세스할 수 없습니다.
  1. REDIRECT 매개변수를 지정하고 RESTORE DATABASE 명령을 실행하십시오. 단일 테이블 스페이스를 변환하려는 경우 TABLESPACE 매개변수를 지정하십시오.

     ```
     RESTORE DATABASE database_name TABLESPACE table_space_name REDIRECT
     ```
  2. 변환하려는 각 테이블 스페이스에 대해 USING AUTOMATIC STORAGE 매개변수를 지정하고 SET TABLESPACE CONTAINERS 명령을 실행하십시오.

     ```
     SET TABLESPACE CONTAINERS FOR tablespace_id USING AUTOMATIC STORAGE
     ```

3. 이번에는 **CONTINUE** 매개변수를 지정하고 RESTORE DATABASE 명령을 실행하십시오.

   ```
   RESTORE DATABASE database_name CONTINUE
   ```

4. **TO END OF LOGS** 및 **AND STOP** 매개변수를 지정하고 ROLLFORWARD DATABASE 명령을 실행하십시오.

   ```
   ROLLFORWARD DATABASE database_name TO END OF LOGS AND STOP
   ```

테이블 스페이스 상위 워터 마크(water mark)

*상위 워터 마크(water mark)*는 마지막 할당된 Extent가 뒤에 오는 Extent에 있는 첫 번째 페이지의 페이지 번호를 말합니다.

예를 들어 테이블 스페이스가 1000페이지와 10의 Extent 크기를 갖는 경우 100개 Extent가 있습니다. 42번째 Extent가 테이블 스페이스에서 가장 높이 할당된 Extent인 경우 상위 워터 마크(water mark)가 420임을 의미합니다.

팁: Extent는 0부터 인덱스화되므로, 상위 워터 마크(water mark)는 *최대 할당 Extent의 마지막 페이지* + 1입니다.
실제로, 사용자가 직접 상위 워터 마크(water mark)를 판별하는 것은 불가능합니다. 행 조작이 발생할 때 경우에 따라 변할 수는 있지만 현재 상위 워터 마크(water mark)의 위치를 판별하는 데 사용할 수 있는 관리 뷰 및 테이블 함수가 있습니다.

상위 워터 마크(water mark)는 상위 워터 마크(water mark) 아래의 일부 Extent가 데이터 삭제의 결과로 비워지지 않았을 수 있기 때문에 상위 워터 마크(water mark)가 사용된 페이지 번호의 표시기는 아님을 주의하십시오. 이 경우 그 아래에 사용 가능한 페이지가 있을 수 있지만 상위 워터 마크(water mark)는 여전히 테이블 스페이스에서 최대 할당된 페이지로 남아있습니다.

테이블 스페이스 크기 축소 조작을 통해 Extent를 통합하여 테이블 스페이스의 상위 워터 마크(water mark)를 낮출 수 있습니다.

예

165 페이지의 그림 16은 테이블 스페이스에 있는 일련의 할당된 Extent를 나타냅니다.

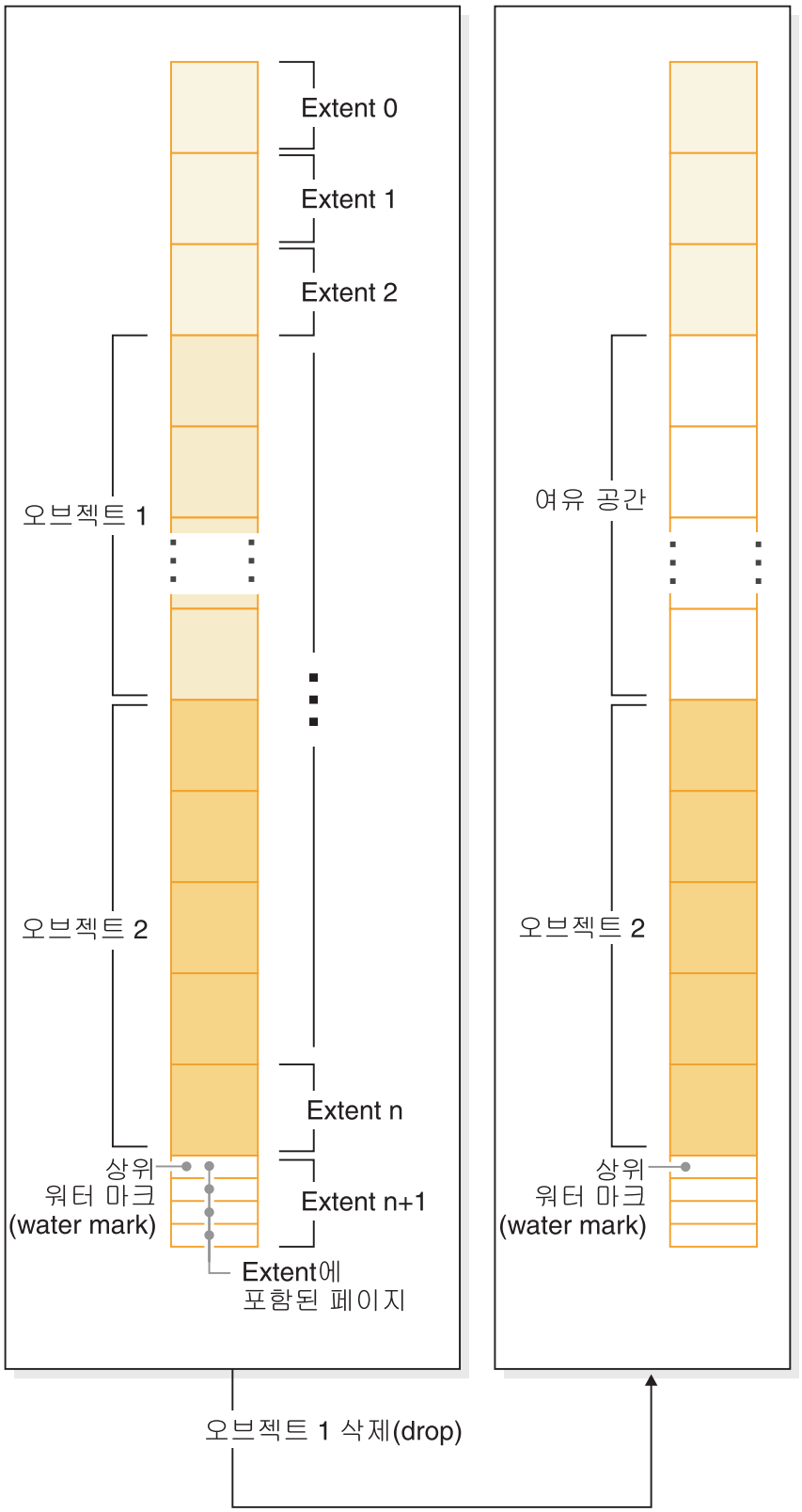


그림 16. 상위 워터 마크(water mark)

오브젝트가 삭제되면 스페이스가 테이블 스페이스에서 사용 가능해집니다. 그러나 임의 형태의 스토리지 총합 조작이 수행될 때까지 상위 워터 마크(water mark)는 이전 레벨

에 남아있습니다. 컨테이너에 대한 새 Extent가 추가되는 방법에 따라서는 위로 이동할 수도 있습니다.

재개 가능 스토리지

*재개 가능 스토리지*는 사용되지 않는 스토리지를 시스템이 재사용할 수 있도록 만드는 능력을 제공하는 비일시적 자동 스토리지 및 DMS 테이블 스페이스의 기능입니다. 데이터가 있는 Extent를 테이블 스페이스의 시작부에 더 가까운 *상위 워터 마크(water mark)* 아래의 사용되지 않는 Extent로 이동하여 이 기능을 수행합니다.

DB2 버전 9.7 이상에서 작성되는 모든 비일시적 자동 스토리지 및 DMS 테이블 스페이스가 이 기능을 제공합니다. DB2 제품의 이전 버전에서 작성된 테이블 스페이스에서 이 성능을 활용하려면 먼저 해당 테이블 스페이스를 버전 9.7에서 작성된 새 테이블 스페이스로 교체해야 합니다. 데이터를 언로드한 후 다시 로드하거나 SYSPROC.ADMIN_MOVE_TABLE 프로시저를 사용한 온라인 테이블 이동 조작으로 데이터를 이동할 수 있습니다. 그러나 이러한 이주는 필요없습니다. 재개 가능 스토리지가 사용 가능한 테이블 스페이스는 재개 가능 스토리지 없이 동일한 데이터베이스에서 테이블 스페이스로서 공존할 수 있습니다.

재개 가능 스토리지가 사용 가능한 테이블 스페이스에서 ALTER TABLESPACE문을 사용하여 테이블 스페이스에서 사용되지 않는 Extent를 재개할 수 있으며, 이것은 테이블 스페이스에 대한 상위 워터 마크(water mark)를 낮추고 크기를 축소하며 동시에 스페이스를 비우는 효과를 갖습니다. 167 페이지의 그림 17은 재개 가능 스토리지가 작업하는 방법의 상위 레벨 뷰를 나타냅니다.

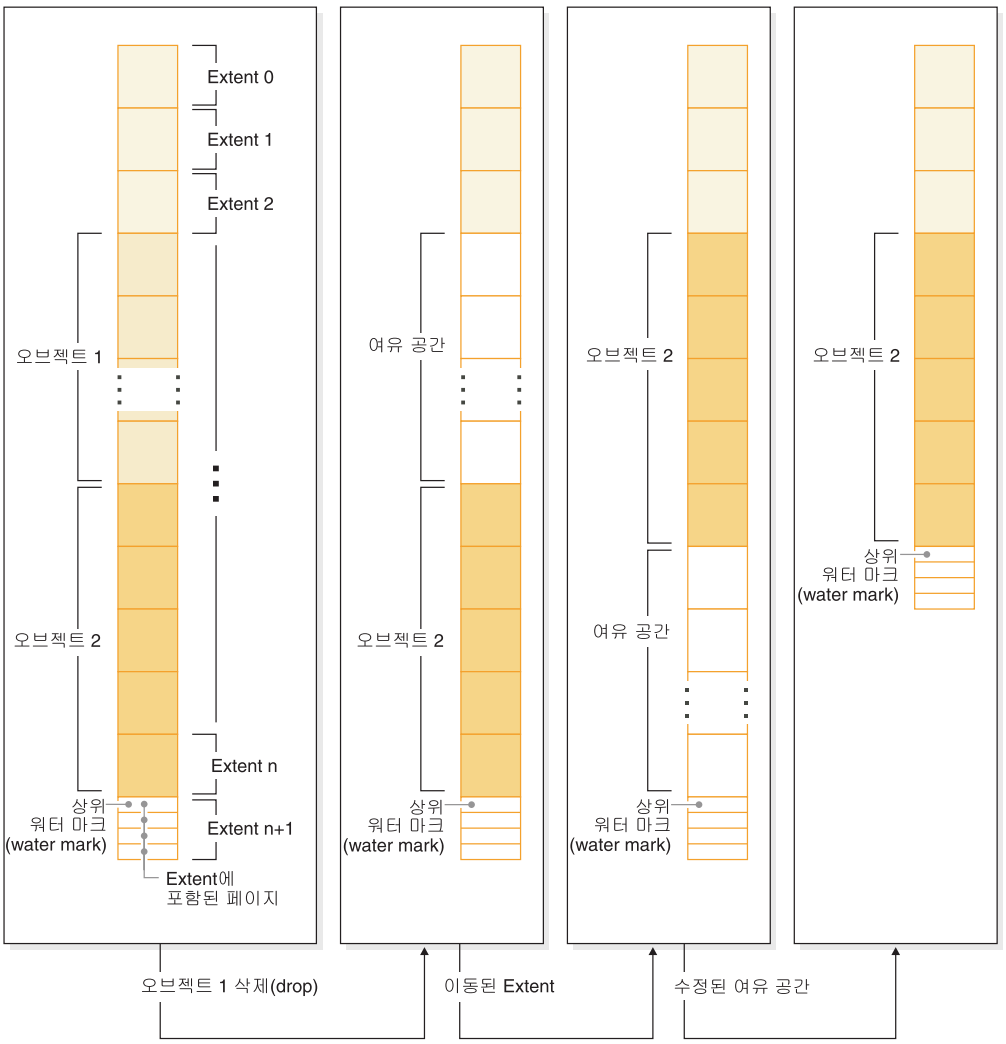


그림 17. 재개 가능 스토리지가 작업하는 방법. 재개 가능 스토리지가 테이블 스페이스에 대해 사용 가능하면 사용 중인 Extent를 이동하여 테이블 스페이스에서 더 낮은 사용되지 않는 Extent를 채울 수 있습니다.

Extent 이동을 통한 테이블 스페이스 크기 축소는 온라인 조작입니다. 즉, 축소 조작이 발생하고 있는 동안 데이터 처리 언어(DML) 및 데이터 정의 언어(DDL)가 계속 실행될 수 있습니다. 백업이나 리스토어 같은 일부 조작은 Extent 이동 조작과 동시에 실행될 수 없습니다. 이러한 경우 이동되는 Extent에 액세스해야 하는 프로세스(예: 백업)는 Extent의 (non-user-₩configurable) 수가 이동되었을 때까지 기다리며, 이 시점에서 백업 프로세스는 문제가 되는 Extent에 대한 잠금을 획득한 다음 이 시점에서 계속합니다.

MON_GET_EXTENT_MOVEMENT_STATUS 테이블 함수를 사용하여 Extent 이동 진행 상태를 모니터할 수 있습니다.

팁: 먼저 테이블 스페이스의 테이블 및 인덱스에 대해 REORG 조작을 수행하여 ALTER TABLESPACE문이 재개할 수 있는 스페이스의 양을 최대화할 수 있습니다.

자동 스토리지 테이블 스페이스

다음과 같은 많은 방법으로 자동 스토리지 테이블 스페이스를 줄일 수 있습니다.

컨테이너 축소만

이 옵션에서는 Extent가 이동되지 않습니다. 데이터베이스 관리 프로그램이 먼저 삭제가 보류 중인 Extent를 비워서 컨테이너의 *크기*를 줄입니다. (일부 『삭제 보류』 Extent는 복구 가능성 때문에 비울 수 없으므로 이러한 Extent의 일부는 남아있을 수 있습니다.) 상위 워터 마크(water mark)가 비워진 Extent 사이에 있었던 경우 상위 워터 마크(water mark)가 낮아지며, 그렇지 않으면 상위 워터 마크(water mark)는 변경되지 않습니다. 다음으로, 컨테이너가 테이블 스페이스에 있는 스페이스의 총량이 상위 워터 마크(water mark)와 같거나 약간 더 크도록 *크기* 조정됩니다. 이 조작은 REDUCE절만을 갖는 ALTER TABLESPACE를 사용하여 수행됩니다.

상위 워터 마크(water mark)만 낮추기

이 옵션에서는 상위 워터 마크(water mark)를 낮추기 위해 최대 수의 Extent가 이동되지만, 컨테이너 *크기* 조정 조작은 수행되지 않습니다. 이 조작은 LOWER HIGH WATER MARK절만을 갖는 ALTER TABLESPACE를 사용하여 수행됩니다.

상위 워터 마크(water mark) 낮추기 및 컨테이너를 특정 양만큼 축소

이 옵션에서는 테이블 스페이스를 축소하는 특정 양을 KB, MB 또는 GB 단위로 지정할 수 있습니다. 또는 백분율을 입력하여 축소할 상대적 양을 지정할 수 있습니다. 두 방법 모두 데이터베이스 관리 프로그램이 먼저 Extent를 이동하지 않고 요청된 양만큼 스페이스를 줄이려고 시도합니다. 즉, 컨테이너 축소만에서 설명하는 대로 컨테이너 *크기*만 줄이고 삭제 보류된 Extent를 비우고 상위 워터 마크(water mark)를 낮추려고 시도하여 테이블 스페이스를 줄이려고 합니다. 이 과정에서 테이블 스페이스가 충분히 줄어들지 않으면 데이터베이스 관리 프로그램은 사용된 Extent를 테이블 스페이스에서 더 아래로 이동하기 시작하여 상위 워터 마크(water mark)를 낮춥니다. Extent 이동이 완료된 후 컨테이너는 테이블 스페이스에 있는 스페이스의 총량이 상위 워터 마크(water mark)와 같거나 약간 더 크도록 *크기* 조정됩니다. 이동할 수 있는 충분한 Extent가 없기 때문에 테이블 스페이스를 요청된 양만큼 줄일 수 없는 경우, 상위 워터 마크(water mark)는 가능한 만큼 많이 축소됩니다. 이 조작은 테이블 스페이스 *크기*를 줄이기 위해 지정된 양을 포함하는 REDUCE절과 함께 ALTER TABLESPACE를 사용하여 수행됩니다.

상위 워터 마크(water mark) 낮추기 및 컨테이너를 가능한 최대양만큼 축소

이 경우에 데이터베이스 관리 프로그램이 테이블 스페이스 및 해당 컨테이너의 *크기*를 줄이기 위해 가능한 만큼 많은 Extent를 이동합니다. 이 조작은 REDUCE MAX절을 갖는 ALTER TABLESPACE를 사용하여 수행됩니다.

Extent 이동 프로세스가 시작된 후에는 REDUCE STOP절을 갖는 ALTER TABLESPACE문을 사용하여 중지할 수 있습니다. 이동된 모든 Extent가 커미트되고, 상위 워터 마크(water mark)는 새 값으로 줄어들고 컨테이너는 새로운 상위 워터 마크(water mark)로 *크기* 조정됩니다.

DMS 테이블 스페이스

DMS 테이블 *스페이스*는 두 가지 방법으로 축소할 수 있습니다.

컨테이너 축소만

이 옵션에서는 Extent가 이동되지 않습니다. 데이터베이스 관리 프로그램이 먼저 삭제가 보류 중인 Extent를 비워서 컨테이너의 *크기*를 줄입니다. (일부 『″삭제 보류″』 Extent는 복구 가능성 때문에 삭제할 수 없으므로 이러한 Extent의 일부는 남아있을 수 있습니다.) 상위 워터 마크(water mark)가 비워진 Extent 사이에 있었던 경우 상위 워터 마크(water mark)가 낮아지며, 그렇지 않으면 상위 워터 마크(water mark)는 변경되지 않습니다. 다음, 컨테이너가 테이블 스페이스에 있는 *스페이스*의 총량이 상위 워터 마크(water mark)와 같거나 약간 더 *크도록 크기* 조정됩니다. 이 조작은 REDUCE *database-container* 절만을 갖는 ALTER TABLESPACE를 사용하여 수행됩니다.

상위 워터 마크(water mark)만 낮추기

이 옵션에서는 상위 워터 마크(water mark)를 낮추기 위해 최대 수의 Extent가 이동되지만, 컨테이너 *크기* 조정 조작은 수행되지 않습니다. 이 조작은 LOWER HIGH WATER MARK절만을 갖는 ALTER TABLESPACE를 사용하여 수행됩니다.

상위 워터 마크(water mark) 낮추기 및 컨테이너 *크기* 축소가 자동 스토리지 테이블 스페이스에서의 결합된 자동 조작인 반면, DMS 테이블 *스페이스*에서는 상위 워터 마크(water mark) 낮추기 및 컨테이너 *크기* 축소를 둘 다 달성하려면 다음 두 조작을 수행해야 합니다.

1. 먼저, LOWER HIGH WATER MARK절을 갖는 ALTER TABLESPACE문을 사용하여 테이블 *스페이스*의 상위 워터 마크(water mark)를 낮춰야 합니다.
2. 다음에 REDUCE *database-container*절만을 갖는 ALTER TABLESPACE문을 사용하여 컨테이너 *크기* 조정 조작을 수행해야 합니다.

Extent 이동 프로세스가 시작된 후에는 LOWER HIGH WATER MARK STOP절을 갖는 ALTER TABLESPACE문을 사용하여 프로세스를 중지할 수 있습니다. 이동된 모든 Extent가 커미트되며 상위 워터 마크(water mark)가 새 값으로 축소됩니다.

예

예 1: 자동 스토리지 테이블 스페이스의 크기를 최대한 축소

하나의 자동 스토리지 테이블 스페이스 TS 및 3개의 테이블 t1, t2 및 t3을 갖는 데이터베이스를 가정합니다. 다음에, 테이블 t1 및 t3을 삭제합니다.

```
DROP TABLE T1
DROP TABLE T3
```

이제, Extent가 사용 가능하다고 가정하고 다음 명령문이 이전에 T1 및 T3으로 채워졌던 Extent가 재개되도록 하며 테이블 스페이스의 상위 워터 마크(water mark)가 축소됩니다.

```
ALTER TABLESPACE TS REDUCE MAX
```

예 2: 자동 스토리지 테이블 스페이스의 크기를 특정 양만큼 축소

하나의 자동 스토리지 테이블 스페이스 TS 및 2개의 테이블 t1과 t2를 갖는 데이터베이스를 가정합니다. 다음에 테이블 T1을 삭제합니다.

```
DROP TABLE T1
```

이제, 테이블 스페이스의 크기를 1MB 줄이기 위해 다음 명령문을 사용하십시오.

```
ALTER TABLESPACE TS REDUCE SIZE 1M
```

또는 다음과 같은 명령문으로 기존 크기의 백분율만큼 테이블 스페이스를 줄일 수 있습니다.

```
ALTER TABLESPACE TS REDUCE SIZE 5 PERCENT
```

예 3: 상위 워터 마크(water mark) 아래에 여유 공간이 있을 때 자동 스토리지 테이블 스페이스의 크기 축소

예 1과 같이 하나의 자동 스토리지 테이블 스페이스 TS 및 3개의 테이블 t1, T2 및 t3를 갖는 데이터베이스를 가정합니다. 이제는 T2 및 t3를 삭제할 때 상위 워터 마크(water mark) 바로 아래에 5개의 사용 가능한 Extent 세트가 있습니다. 이제 이 경우의 각 Extent가 2개의 4K 페이지로 구성되었다고 가정하면 이것은 실제로는 상위 워터 마크(water mark) 바로 아래에 40K의 여유 공간이 있음을 의미합니다. 다음과 같은 명령문을 발행하면

```
ALTER TABLESPACE TS REDUCE SIZE 32K
```

데이터베이스 관리 프로그램은 어떤 Extent 이동도 수행할 필요 없이 상위 워터 마크(water mark)를 낮추고 컨테이너 크기를 줄일 수 있습니다. 이것은 171 페이지의 그림 18에서 설명되어 있습니다.

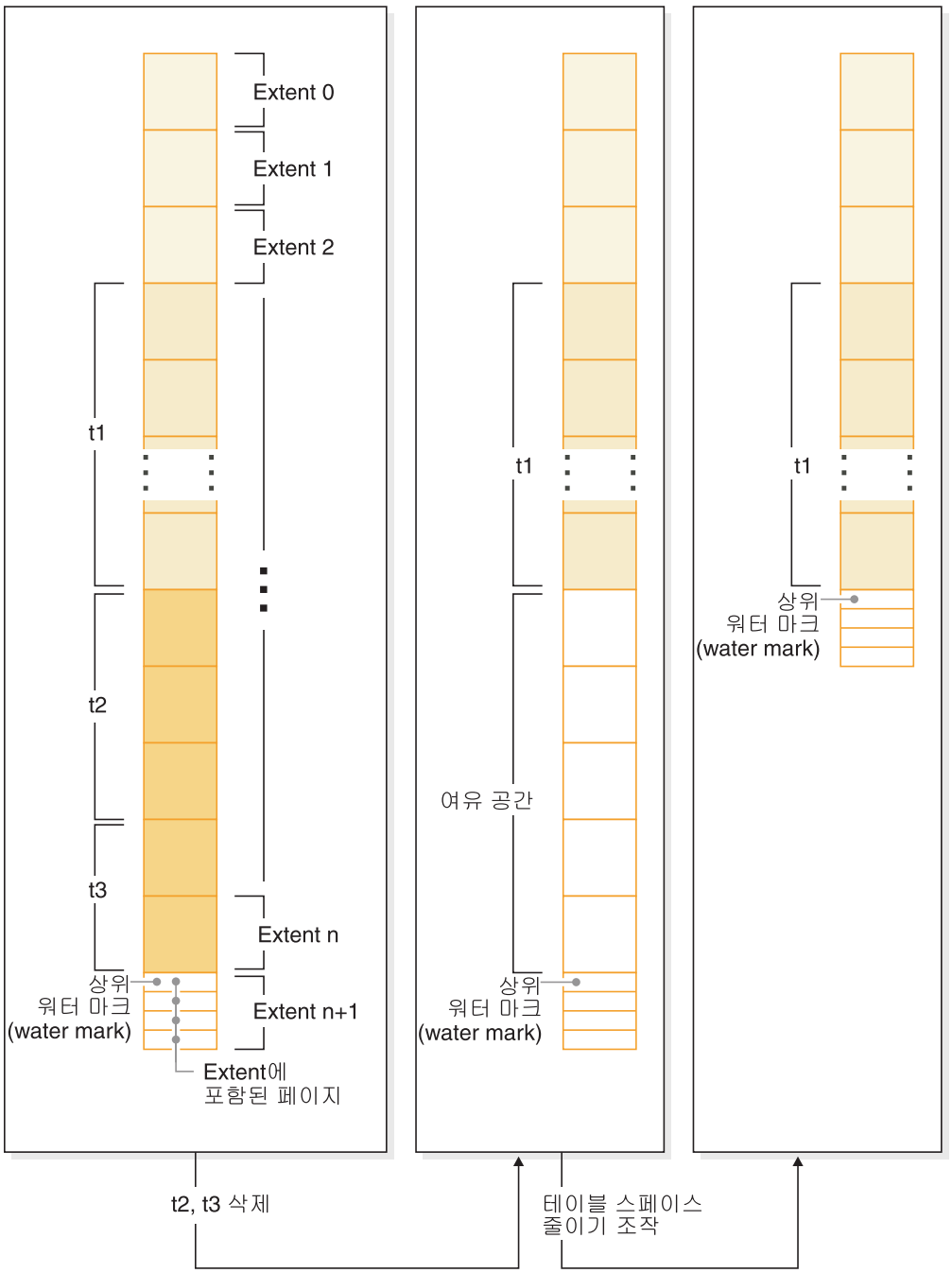


그림 18. Extent를 이동할 필요 없이 상위 워터 마크(water mark) 낮추기

예 4: DMS 테이블 스페이스의 크기 축소

하나의 DMS 테이블 스페이스 TS 및 3개의 테이블 t1, t2 및 t3를 갖는 데이터베이스를 가정합니다. 다음에, 테이블 t1 및 t3을 삭제합니다.

```
DROP TABLE T1
DROP TABLE T3
```

DMS 테이블 스페이스에서 상위 워터 마크(water mark)를 낮추고 컨테이너 크기를 줄이는 것은 2단계 조작입니다. 먼저 다음 명령문을 사용한 Extent 이동을 통해 상위 워터 마크(water mark)를 낮춥니다.

```
ALTER TABLESPACE TS LOWER HIGH WATER MARK
```

다음으로, 다음과 같은 명령문을 사용하여 컨테이너의 크기를 줄입니다.

```
ALTER TABLESPACE TS REDUCE (ALL CONTAINERS 5 M)
```

SMS, DMS 및 자동 스토리지 테이블 스페이스의 비교

SMS, DMS 및 자동 스토리지 테이블 스페이스는 서로 다른 상황에서 유익할 수 있는 여러 가지 성능을 제공합니다.

표 10. SMS, DMS 및 자동 스토리지 테이블 스페이스의 비교

	SMS 테이블 스페이스	**DMS 테이블 스페이스**	**자동 스토리지 테이블 스페이스**
작성되는 방법	CREATE TABLESPACE문의 MANAGED BY SYSTEM절을 사용하여 작성	CREATE TABLESPACE문의 MANAGED BY DATABASE절을 사용하여 작성	CREATE TABLESPACE문의 MANAGED BY AUTOMATIC STORAGE절을 사용하거나, MANAGED BY절을 완전히 생략하여 작성. 데이터베이스가 작성될 때 자동 스토리지가 사용 가능했던 경우, 사용자가 작성하는 모든 데이터베이스에 디폴트는 사용자가 별도로 지정하지 않는 한 자동 스토리지 테이블 스페이스로서 작성하는 것입니다.
초기 컨테이너 정의 및 위치	컨테이너가 디렉토리 이름으로 정의되어야 합니다.	• 컨테이너가 파일 또는 디바이스로서 정의되어야 합니다. • 각 컨테이너에 대한 초기 크기를 지정해야 합니다.	자동 스토리지 테이블 스페이스를 작성할 때 컨테이너 목록을 제공하지 않습니다. 대신, 데이터베이스 관리 프로그램이 자동으로 데이터베이스와 연결된 모든 스토리지 경로에 컨테이너를 작성합니다. 데이터는 모든 컨테이너 사이에 균등하게 스트라이프되어 스토리지 경로가 동등하게 사용됩니다.
스페이스의 초기 할당	필요한 대로 수행됩니다. 파일 시스템이 스토리지의 할당을 제어하기 때문에 페이지가 연속적일 가능성은 작으며, 이는 일부 유형의 쿼리 성능에 영향을 줄 수 있습니다.	테이블 스페이스가 작성될 때 수행됨 • Extent가 SMS 테이블 스페이스에서보다 연속적일 가능성이 더 큽니다. • Extent의 페이지는 항상 디바이스 컨테이너에 대해 연속적입니다.	• 비임시 자동 스토리지 테이블 스페이스의 경우, – 스페이스는 테이블 스페이스가 작성될 때 할당됩니다. – 사용자가 테이블 스페이스의 초기 크기를 지정할 수 있습니다. • 임시 자동 스토리지 테이블 스페이스의 경우 스페이스는 필요할 때 할당됩니다.

표 10. SMS, DMS 및 자동 스토리지 테이블 스페이스의 비교 (계속)

	SMS 테이블 스페이스	DMS 테이블 스페이스	자동 스토리지 테이블 스페이스
테이블 스페이스 컨테이너에 대한 변경	데이터 파티션이 필요할 때 새 데이터 파티션에 대한 컨테이너를 추가하는 대신 일단 작성된 후에는 변경되지 않습니다.	• 컨테이너가 확장 또는 추가될 수 있습니다. 테이블 스페이스에 대한 상위 워터 마크(water mark) 아래에 새 스페이스가 추가되는 경우 테이블 스페이스 데이터의 재조정이 발생합니다. • 컨테이너를 축소 또는 삭제할 수 있습니다. 삭제될 스페이스에 데이터가 있는 경우 재조정이 발생합니다.	• 테이블 스페이스 크기가 축소되는 경우 컨테이너를 삭제 또는 축소할 수 있습니다. • 테이블 스페이스를 재조정하여 새 스토리지가 데이터베이스에서 추가 또는 삭제될 때 컨테이너 사이에 균등하게 데이터를 분산할 수 있습니다.
늘어난 스토리지 수요 조절	컨테이너는 파일 시스템이 부과하는 용량에 도달할 때까지 성장합니다. 테이블 스페이스는 임의의 컨테이너 하나가 최대 용량에 도달할 때 가득 찬 것으로 간주됩니다.	컨테이너는 파일 시스템이 부과하는 제한조건까지 수동 또는 자동으로 (자동 크기 조정이 사용되는 경우) 초기에 할당된 크기 이상으로 확장될 수 있습니다.	• 컨테이너는 파일 시스템이 부과하는 제한조건까지 자동으로 확장됩니다. • 스토리지 경로가 데이터베이스에 추가되는 경우 컨테이너는 자동으로 확장 또는 작성됩니다.
여러 가지 유형의 오브젝트를 서로 다른 테이블 스페이스에 배치하는 기능	파티션된 테이블의 경우에만 인덱스와 인덱스 파티션이 테이블 데이터가 들어있는 것과는 별개인 테이블 스페이스에 상주할 수 있습니다.	테이블, 관련 대형 오브젝트(LOB)에 대한 스토리지 및 인덱스는 별도의 테이블 스페이스에 상주할 수 있습니다.	테이블, 관련 대형 오브젝트(LOB)에 대한 스토리지 및 인덱스는 별도의 테이블 스페이스에 상주할 수 있습니다.
진행 중인 유지보수 요구사항	없음	• 컨테이너 추가 또는 확장 • 컨테이너 삭제 또는 축소 • 상위 워터 마크(water mark) 낮추기 • 재조정	• 테이블 스페이스 크기 축소 • 상위 워터 마크(water mark) 낮추기 • 재조정
컨테이너 재정의를 위한 리스토어 사용	경로 재지정된 리스토어 작업을 사용하여 테이블 스페이스와 연관된 컨테이너를 재정의할 수 있습니다.	경로 재지정된 리스토어 작업을 사용하여 테이블 스페이스와 연관된 컨테이너를 재정의할 수 있습니다.	데이터베이스 관리 프로그램이 스페이스를 관리하기 때문에 경로 재지정된 리스토어 작업을 사용하여 테이블 스페이스와 연관된 컨테이너를 재정의할 수 있습니다.
성능	일반적으로, 특히 더 큰 테이블에 대한 DMS 및 자동 스토리지보다 낮습니다.	일반적으로 SMS에 비해 우수	DMS와 비슷함

테이블 스페이스의 세 가지 유형 중에서, 자동 스토리지 테이블 스페이스가 설정 및 유지보수하기에 가장 쉽고 대부분의 응용프로그램에 권장됩니다. 특히 다음 경우에 유용합니다.

- 더 큰 테이블 또는 빠르게 성장할 수 있는 테이블이 있습니다.
- 컨테이너 증가를 관리하는 방법에 대해 사용자가 일반적인 결정을 하지 않으려고 합니다.

- 성능을 향상시키기 위해 서로 다른 테이블 스페이스에 관련된 오브젝트의 여러 가지 유형(예: 테이블, LOB, 인덱스)을 저장할 수 있어야 합니다.

DMS 테이블 스페이스는 다음 경우에 사용됩니다.

- 더 큰 테이블 또는 빠르게 성장할 수 있는 테이블이 있습니다.
- 데이터가 실제로 저장되는 장소에 대한 더 큰 제어를 원합니다.
- 스토리지가 사용되는 방법(예: 컨테이너 추가)을 조정하거나 제어할 수 있기를 원합니다.
- 성능을 향상시키기 위해 서로 다른 테이블 스페이스에 관련된 오브젝트의 여러 가지 유형(예: 테이블, LOB, 인덱스)을 저장할 수 있어야 합니다.

SMS 테이블 스페이스는 다음 경우에 유용합니다.

- 빠르게 증가할 가능성이 작은 더 작은 테이블이 있습니다.
- 데이터가 실제로 저장되는 장소에 대한 더 큰 제어를 원합니다.
- 컨테이너 유지보수에 관여하지 않기를 원합니다.
- 서로 다른 테이블 스페이스에 관련된 오브젝트의 여러 가지 유형(예: 테이블, LOB, 인덱스)을 저장할 필요가 없습니다. (파티션된 테이블의 경우에만, 인덱스를 테이블 데이터와는 별개인 테이블 스페이스에 저장할 수 *있습니다*).

SMS 및 DMS 워크로드 고려사항:

사용자 환경에서 데이터베이스 관리 프로그램이 관리하는 워크로드의 기본 유형에 따라 사용할 테이블 스페이스의 유형 및 지정할 페이지 *크기*를 선택하는 방법이 달라집니다.

온라인 트랜잭션 처리(OLTP) 워크로드는 데이터에 대한 임의의 액세스가 필요한 트랜잭션이 특징이며 종종 잦은 삽입 또는 갱신 활동과 대개 작은 데이터 세트를 리턴하는 쿼리가 필요합니다. 액세스가 임의로 이루어지고 하나 또는 소수의 페이지가 관련될 경우, 프리페치가 거의 발생하지 않습니다.

이 경우, 디바이스 컨테이너를 사용하는 DMS 테이블 스페이스가 가장 우수한 성능을 발휘합니다. 최대의 성능이 필요하지 않을 경우, OLTP 워크로드에 대해 파일 컨테이너가 있는 DMS 테이블 스페이스 또는 SMS 테이블 스페이스를 선택하는 것이 합리적입니다. FILE SYSTEM CACHING이 꺼진 파일 컨테이너와 함께 DMS 테이블 스페이스를 사용하면 DMS 원시 테이블 스페이스 컨테이너에 비교할 수 있는 레벨에서 수행할 수 있습니다. 순차적 입출력이 거의 없거나 전혀 없을 경우, CREATE TABLESPACE문에서의 EXTENTSIZE 및 PREFETCHSIZE 매개변수 설정값은 입출력 효율성에 영향을 주지 않습니다. 그러나 *chngpgs_thresh* 구성 매개변수를 사용하여 충분한 페이지 클리너 수를 설정하는 것은 중요합니다.

쿼리 워크로드는 보통 대형 데이터 세트를 리턴하는 데이터에 순차적으로 또는 부분 순차적으로 액세해야 하는 트랜잭션이 특징입니다. 여러 디바이스 컨테이너를 사용하고 각

컨테이너가 별도의 디스크에 있는 DMS 테이블 스페이스에서 효율적인 프리페치가 이루어질 가능성이 가장 큽니다. CREATE TABLESPACE문의 PREFETCHSIZE 매개변수 값은 EXTENTSIZE 매개변수 값에 디바이스 컨테이너 수를 곱한 값으로 설정해야 합니다. 다른 방법으로는, 프리페치 *크기*를 -1로 지정할 수 있으며 데이터베이스 관리 프로그램이 자동으로 적절한 프리페치 *크기*를 선택합니다. 이 경우 데이터베이스 관리 프로그램이 모든 컨테이너로부터 병렬로 프리페치할 수 있습니다. 컨테이너 수가 변경되거나 프리페치를 다소 적극적인 것으로 만들려면 ALTER TABLESPACE문을 사용하여 PREFETCHSIZE 값을 적절하게 변경할 수 있습니다.

파일 시스템에 자체 프리페치가 없을 경우, 쿼리 워크로드에 대한 합리적인 대안은 파일을 사용하는 것입니다. 파일은 파일 컨테이너를 사용하는 DMS 유형이거나 SMS 유형일 수 있습니다. SMS를 사용할 경우, 입출력 병렬 처리를 달성하기 위해 디렉토리 컨테이너가 별도의 물리적 디스크에 맵핑하도록 만들 수 있습니다.

혼합 워크로드의 목적은 OLTP 워크로드에 대한 하나의 입출력 요청을 가능한 효율적으로 하여, 쿼리 워크로드에 대한 병렬 입출력의 효율성을 극대화하는 것입니다.

테이블 스페이스의 페이지 *크기*를 결정할 때 고려해야 할 사항은 다음과 같습니다.

- 임의의 행을 읽고 쓰는 OLTP 응용프로그램의 경우, 페이지 *크기*가 작으면 불필요한 행에 버퍼 풀 스페이스를 사용하지 않기 때문에 페이지 *크기*가 작을 수록 좋습니다.
- 한 번에 많은 수의 연속 행에 액세스하는 DSS 응용프로그램의 경우, 페이지 *크기*가 크면 특정 행 번호를 읽는데 필요한 입출력 요청 수가 줄어들기 때문에 페이지 *크기*가 클수록 좋습니다.
- 페이지 *크기*가 더 크면 인덱스의 레벨 수를 줄일 수 있습니다.
- 페이지 *크기*가 더 크면 보다 길이가 긴 행이 지원됩니다.
- 디폴트 4KB 페이지에서 테이블은 500 컬럼으로 제한되는 반면, 더 큰 페이지 *크기* (8KB, 16KB 및 32KB)는 1012 컬럼을 지원합니다.
- 테이블 스페이스의 최대 *크기*는 테이블 스페이스의 페이지 *크기*에 비례합니다.

SMS 및 DMS 디바이스 고려사항:

파일 시스템 파일 대 테이블 스페이스 컨테이너용 디바이스 사용 여부를 선택할 때 고려할 몇 가지 옵션이 있는데, 데이터 버퍼링과 LOB 또는 LOG 데이터 사용 여부입니다.

- **데이터 버퍼링**

  디스크로부터 읽혀지는 테이블 데이터는 보통 데이터베이스 버퍼 풀에서 사용할 수 있습니다. 일부 경우에는 응용프로그램이 실제 페이지를 사용하기 전에, 특히 버퍼 풀 스페이스가 다른 데이터 페이지에 대해 필요한 경우에 데이터 페이지가 버퍼 풀로부터 해제될 수 있습니다. 시스템 관리 스페이스(SMS) 또는 데이터베이스 관리 스페

이스(DMS) 파일 컨테이너를 사용하는 테이블 스페이스의 경우, 위의 파일 시스템 캐시는 달리 필요하지 않은 입출력을 제거할 수 있습니다.

데이터베이스 관리 스페이스(DMS) 디바이스 컨테이너를 사용하는 테이블 스페이스는 파일 시스템 또는 파일 시스템 캐시를 사용하지 않습니다. 결과적으로, 디바이스 컨테이너를 사용하는 DMS 테이블 스페이스를 가지고 이중 버퍼링을 수행할 수 없는 사실을 보완하기 위해 데이터베이스 버퍼 풀의 *크기*를 증가시키고 파일 시스템 캐시 *크기*를 줄일 수도 있습니다.

시스템 레벨 모니터링 도구 사용을 통해 디바이스 컨테이너를 사용한 DMS 테이블 스페이스의 입출력이 동일한 SMS 테이블 스페이스에 비해 높게 표시되는 경우, 이러한 차이는 이중 버퍼링으로 인한 것일 수도 있습니다.

- **LOB 또는 LONG 데이터 사용**

  응용프로그램이 LOB 또는 LONG 데이터를 검색하는 경우, 데이터베이스 관리 프로그램은 버퍼의 데이터를 캐시하지 않습니다. 응용프로그램이 이 페이지 중 하나를 필요로 할 때마다, 데이터베이스 관리 프로그램은 디스크로부터 이 페이지를 검색해야 합니다. 그러나 LOB 또는 LONG 데이터를 SMS 또는 DMS 파일 컨테이너에 저장할 경우, 파일 시스템 캐싱은 버퍼링을 제공할 수도 있으므로 결과적으로 성능이 향상됩니다.

  시스템 카탈로그에는 일부 LOB 컬럼이 들어 있기 때문에, DMS 파일 테이블 스페이스 또는 SMS 테이블 스페이스에 해당 컬럼을 보존해야 합니다.

임시 테이블 스페이스

임시 테이블 스페이스는 정렬 또는 조인과 같은 조작을 수행할 때 이러한 활동은 결과 세트를 처리하기 위해 추가 스페이스가 필요하므로 데이터베이스 관리 프로그램에 필요한 임시 데이터를 보유합니다.

데이터베이스는 카탈로그 테이블 스페이스와 동일한 페이지 *크기*를 갖는 *시스템* 임시 테이블 스페이스를 최소한 하나는 가져야 합니다. 디폴트로 데이터베이스 작성 시간에 TEMPSPACE1이라는 하나의 시스템 임시 테이블 스페이스가 작성됩니다. IBMTEMPGROUP은 이 테이블 스페이스의 디폴트 데이터베이스 파티션 그룹입니다. TEMPSPACE1의 페이지 *크기*는 데이터베이스 자체가 작성될 때 지정된 *크기*입니다 (디폴트로 4KB).

사용자 임시 테이블 스페이스는 DECLARE GLOBAL TEMPORARY TABLE 또는 CREATE GLOBAL TEMPORARY TABLE문으로 작성된 테이블의 임시 데이터를 보유합니다. 사용자 임시 테이블 스페이스는 데이터베이스 작성 시에 디폴트로 작성되지 않습니다. 또한 작성된 임시 테이블의 인스턴스화된 버전을 보유합니다.

대부분의 사용자 테이블 스페이스에서 사용되는 페이지 크기와 동일한 페이지 크기를 갖는 단일 임시 테이블 스페이스를 정의하는 것이 바람직합니다. 이는 일반적인 환경 및 워크로드에 적합해야 합니다. 그러나, 다른 임시 테이블 스페이스 구성 및 워크로드를 사용하면 더 나을 수 있습니다. 다음과 같은 사항을 고려해야 합니다.

- 대부분의 경우, 임시 테이블은 일괄처리 및 순차적으로 액세스됩니다. 즉, 행이 일괄적으로 삽입되거나 순차 행이 일괄적으로 페치됩니다. 그러므로 일반적으로 더 큰 페이지 크기를 사용하면 주어진 데이터량을 읽기 위해 더 적은 수의 논리 및 물리 페이지 입출력 요청이 필요하므로 성능이 더 좋아집니다.
- 임시 테이블 스페이스를 사용하여 테이블을 재구성하는 경우, 임시 테이블 스페이스의 페이지 크기는 테이블의 페이지 크기와 일치해야 합니다. 이러한 이유로, 임시 테이블 스페이스를 사용하여 재구성할 수 있는 기존 테이블에서 사용하는 서로 다른 페이지 크기에 대해 정의된 임시 테이블 스페이스가 있는지 확인해야 합니다.

  동일한 테이블 스페이스에서 직접 테이블을 재구성하여 임시 테이블 스페이스 없이 재구성할 수도 있습니다. 이런 재구성 유형은 재구성 프로세스를 위해 테이블의 테이블 스페이스에 추가 스페이스가 있어야 합니다.
- SMS 시스템 임시 테이블 스페이스를 사용할 때 레지스트리 변수 DB2_SMS_TRUNC_TMPTABLE_THRESH 사용을 고려할 수 있습니다. 시스템 임시 테이블에 대해 작성된 파일은 삭제될 때 0의 크기로 절단됩니다. DB2_SMS_TRUNC_TMPTABLE_THRESH를 사용하여 파일 시스템 방문을 피하고 잠재적으로 파일을 0이 아닌 크기로 두어서 파일의 반복되는 확장 및 절단의 성능 비용을 피할 수 있습니다.
- 일반적으로, 페이지 크기가 다른 임시 테이블 스페이스가 있는 경우 옵티마이저는 버퍼 풀이 가장 많은 수의 행을 포함할 수 있는 임시 테이블 스페이스(대부분 가장 큰 버퍼 풀을 의미함)를 선택합니다. 이 경우 임시 테이블 스페이스 중 하나에 충분히 큰 버퍼 풀을 지정하고 나머지는 작은 버퍼 풀을 지정하는 것이 바람직합니다. 이와 같은 버퍼 풀 지정은 주기억장치를 효율적으로 활용할 수 있게 해 줍니다. 예를 들어, 카탈로그 테이블 스페이스가 4KB 페이지를 사용하고 나머지 테이블 스페이스가 8KB를 사용할 경우, 최적의 임시 테이블 스페이스 구성은 큰 버퍼 풀을 사용하는 단일 8KB 임시 테이블 스페이스와 작은 버퍼 풀을 사용하는 단일 4KB 테이블 스페이스일 수 있습니다.
- 일반적으로, 단일 페이지 크기의 임시 테이블 스페이스를 둘 이상 정의해도 별 이득은 없습니다.

시스템 임시 테이블 스페이스 페이지 크기가 요구사항을 충족하는지 확인

보다 큰 레코드 ID(RID)를 사용하면 쿼리 또는 위치가 지정된 갱신의 결과 세트에서 행 크기가 늘어납니다. 결과 세트의 행 크기가 기존 시스템 임시 테이블 스페이스에 대한 최대 행 길이 한계에 가까운 경우 페이지 크기가 더 큰 시스템 임시 테이블 스페이스를 작성해야 합니다.

전제조건

필요한 경우 시스템 임시 테이블 스페이스를 작성할 수 있는 SYSCTRL 또는 SYSADM 권한이 있는지 확인하십시오.

프로시저

시스템 임시 테이블 스페이스의 페이지 최대 *크기*가 쿼리 또는 위치 지정된 갱신에 적합하도록 충분히 큰지 확인하려면 다음을 수행하십시오.

1. 쿼리 또는 위치 지정된 갱신의 결과 세트의 최대 행 *크기*를 판별하십시오. 쿼리를 모니터하거나 테이블 작성에 사용한 DDL문을 사용하여 최대 행 *크기*를 계산하십시오.
2. 각 시스템 임시 테이블 스페이스의 페이지 *크기* 및 쿼리 또는 갱신에서 참조된 테이블이 다음 쿼리를 발행하여 작성된 테이블 스페이스의 페이지 *크기*를 판별하십시오.

```
db2 "SELECT CHAR(TBSP_NAME,20) TBSP_NAME, TBSP_CONTENT_TYPE, TBSP_PAGE_SIZE
     FROM SYSIBMADM.SNAPTBSP"

TBSP_NAME            TBSP_CONTENT_TYPE TBSP_PAGE_SIZE
-------------------- ----------------- --------------------
SYSCATSPACE          ANY                            8192
TEMPSPACE1           SYSTEMP                        8192
USERSPACE1           LARGE                          8192
IBMDB2SAMPLEREL      LARGE                          8192
SYSTOOLSPACE         LARGE                          8192
SYSTOOLSTMPSPACE     USRTEMP                        8192

  6 레코드가 선택되었습니다.
```

   값이 SYSTEMPT인 BSP_CONTENT_TYPE 컬럼이 있는 테이블 스페이스를 찾아서 출력에 있는 시스템 임시 테이블 스페이스를 식별할 수 있습니다.

   버전 8.1에서 업그레이드하려면 다음 명령을 사용하십시오.

```
db2 LIST TABLESPACES SHOW DETAIL
```

3. 결과 세트의 최대 행 *크기*가 시스템 임시 테이블 페이지 *크기*에 맞는지 다음과 같이 확인하십시오.

```
maximum_row_size > maximum_row_length - 8바이트(단일 파티션의
                                                 구조 오버헤드)
maximum_row_size > maximum_row_length - 16바이트(DPF의 구조 오버헤드)
```

   여기에서 maximum_row_size는 결과 세트의 최대 행 *크기*이고 maximum_row_length는 모든 시스템 임시 테이블 스페이스의 최대 페이지 *크기*에 따라 허용된 최대 *크기*입니다. 테이블 스페이스 페이지 *크기*당 최소 행 길이를 판별하려면 *SQL 참조서, 볼륨 1*의 "SQL 및 XML 한계"를 검토하십시오.

   최대 행 *크기*가 계산된 값 보다 적으면 DB2 UDB 버전 8에서 수행한 방법과 같은 방법으로 쿼리가 실행되며 이 태스크를 계속할 필요가 없습니다.

4. 테이블이 작성된 테이블 스페이스 페이지 크기 보다 최소한 한 페이지 크기가 큰 시스템 임시 테이블 스페이스를 작성하십시오. (이 페이지 크기의 시스템 임시 테이블이 없는 경우) 예를 들어 Windows 운영 체제에서 페이지 크기가 8KB인 테이블 스페이스에 테이블을 작성한 경우 다음과 같이 16KB 페이지 크기를 사용하여 시스템 임시 테이블 스페이스를 추가로 작성하십시오.

```
db2 CREATE SYSTEM TEMPORARY TABLESPACE tmp_tbsp
      PAGESIZE 16K
            MANAGED BY SYSTEM
            USING ('d:₩tmp_tbsp','e:₩tmp_tbsp')
```

테이블 스페이스 페이지 크기가 32KB이면 쿼리에서 선택한 정보를 줄이거나 시스템 임시 테이블 스페이스 페이지에 맞게 쿼리를 분할할 수 있습니다. 예를 들어 테이블에서 모든 컬럼을 선택하는 경우 페이지 크기 한계를 초과하지 않도록 대신 필수 컬럼만을 선택하거나 특정 컬럼의 부속 문자열만 선택할 수 있습니다.

사용자 테이블의 테이블 스페이스 선택 시 고려사항

테이블을 테이블 스페이스에 맵핑하는 방법을 결정할 때 테이블의 분산, 테이블에 있는 데이터의 양과 유형 및 관리 문제를 고려해야 합니다.

테이블의 분산

최소한 선택한 테이블 스페이스는 사용자가 원하는 분산이 있는 데이터베이스 파티션 그룹에 있어야 합니다.

테이블의 데이터량

여러 개의 소형 테이블을 한 테이블 스페이스에 저장하기로 한 경우, 해당 테이블 스페이스에 대해 SMS를 사용하십시오. 입출력 및 스페이스 관리가 효율적이라는 DMS의 장점은 소형 테이블의 경우 그다지 중요하지 않습니다. SMS는 필요 시 소형 테이블에 사용하면 도움이 됩니다. 테이블 중 하나의 테이블이 더 크거나 테이블의 데이터에 더 빨리 액세스해야 하는 경우, 작은 Extent 크기의 DMS 테이블 스페이스를 사용하는 것이 좋습니다.

각 대형 테이블에 대해 개별적인 테이블 스페이스를 사용하고 모든 소형 테이블을 단일 테이블 스페이스로 그룹화할 수도 있습니다. 이러한 분리를 통해 테이블 스페이스 사용에 따라 적절한 Extent 크기를 선택할 수 있습니다.

테이블에 있는 데이터 유형

예를 들어, 가끔씩 사용되는 실행기록 데이터가 들어 있는 테이블이 있는데, 일반 사용자가 이 데이터에 대한 쿼리 응답 시간을 좀 더 길게 설정하려 할 수도 있습니다. 이 경우, 실행기록 테이블에 서로 다른 테이블 스페이스를 사용하고, 액세스 속도가 좀 더 느리고 좀 더 저렴한 가격의 물리적 디바이스에 이 테이블 스페이스를 할당할 수 있습니다.

또한 데이터가 쉽게 사용할 수 있어야 하며 빠른 응답 시간이 필요한 일부 필수 테이블을 식별할 수도 있습니다. 이와 같이 중요한 데이터 요구사항을 지원하는데 도움을 줄 수 있는 빠른 물리 디바이스에 할당된 테이블 스페이스에 이들 테이블을 배치할 수 있습니다.

DMS 테이블 스페이스를 사용하면 테이블 데이터를 네 개의 다른 테이블 스페이스에 분배할 수 있습니다. 하나는 인덱스 데이터용이고, 하나는 대형 오브젝트(LOB) 및 Long 필드(LF) 데이터용이고, 하나는 일반 테이블 데이터용이고 하나는 XML 데이터용입니다. 이는 사용자가 데이터에 가장 적합한 테이블 스페이스 특성과 테이블 스페이스를 지원하는 물리적 디바이스를 선택할 수 있도록 합니다. 예를 들어, 사용할 수 있는 디바이스 중 가장 빠른 디바이스에 인덱스 데이터를 위치시키게 되면, 성능이 많이 개선될 수 있습니다. DMS 테이블 스페이스에서 테이블을 분할하는 경우, 롤 포워드 복구를 사용할 수 있으면 해당 테이블 스페이스를 모두 백업한 후 리스토어하는 방안을 고려해야 합니다. SMS 테이블 스페이스는 테이블 스페이스에서 데이터를 이러한 형식으로 분배하는 것을 지원하지 않습니다.

관리 문제

관리 기능 중 일부는 데이터베이스 또는 테이블 레벨 대신 테이블 스페이스 레벨에서 수행될 수 있습니다. 예를 들어, 데이터베이스 대신 테이블 스페이스를 백업할 경우 시간과 자원을 보다 효율적으로 활용할 수 있습니다. 이와 같이 하면 변경사항이 많은 테이블 스페이스는 자주 백업하는 반면, 변경사항이 매우 적은 테이블 스페이스는 가끔씩 백업할 수 있습니다.

데이터베이스 또는 테이블 스페이스를 리스토어할 수 있습니다. 서로 상관없는 테이블이 테이블 스페이스를 공유하지 않는 경우, 데이터베이스의 더 작은 부분을 리스토어하여 비용을 줄일 수 있습니다.

좋은 접근 방식은 관련 테이블을 테이블 스페이스 세트로 그룹화하는 것입니다. 이들 테이블은 참조 제한조건을 통해 또는 기타 비즈니스 제한조건을 통해 관련될 수 있습니다.

특정 테이블을 자주 삭제하고 재정의해야 할 경우, 자체 테이블 스페이스에 테이블을 정의하는데, 그 이유는 테이블을 삭제하는 것보다 DMS 테이블 스페이스를 삭제하는 것이 더 효율적이기 때문입니다.

파일 시스템 캐싱 없는 테이블 스페이스

UNIX, Linux 및 Windows에서 버퍼되지 않은 입출력을 사용 또는 사용 불가능하게 하는 권장 방법은 테이블 스페이스 레벨에 있습니다.

이 메소드로 데이터베이스의 실제 레이아웃에 대한 종속성을 피하면서 특정 테이블 스페이스에서 버퍼되지 않은 입출력을 사용 또는 사용 안할 수 있습니다. 또한 데이터베이스 관리 프로그램이 버퍼 지정 또는 버퍼되지 않는 각 파일에 가장 잘 맞는 입출력을 판별할 수 있습니다.

NO FILE SYSTEM CACHING절이 버퍼되지 않는 입출력을 사용 가능하게 하는 데 사용되므로, 특정 테이블 스페이스에 대한 파일 캐싱을 사용 안합니다. 사용 가능한 경우 플랫폼을 기초로 데이터베이스 관리 프로그램이 자동으로 사용될 직접 입출력(DIO) 또는 동시 입출력(CIO)을 판별합니다. CIO에서 성능 향상이 제공되는 경우 데이터베이스 관리 프로그램은 CIO가 지원될 때마다 사용합니다. 사용될 사용자 인터페이스를 지정할 사용자 인터페이스는 없습니다.

버퍼되지 않는 I/O의 이점을 극대화하기 위해 버퍼 풀의 *크기*를 늘려야 합니다. 그러나 자체 성능 조정 메모리 관리자가 사용 가능하고 버퍼 풀 *크기*가 AUTOMATIC으로 설정된 경우 데이터베이스 관리 프로그램이 최적의 성능을 위해 버퍼 풀 *크기*를 자체 성능 조정합니다. 이 기능은 버전 9 이전에서는 사용할 수 없습니다.

파일 시스템 캐싱을 사용 안하거나 사용하려면 각각 CREATE TABLESPACE 또는 ALTER TABLESPACE문에서 NO FILE SYSTEM CACHING 또는 FILE SYSTEM CACHING절을 지정하십시오. 어떤 절도 지정하지 않으면 디폴트 설정이 사용됩니다. ALTER TABLESPACE의 경우, 새 캐싱 규정이 적용되기 전에 데이터베이스에 대한 기존 연결을 종료해야 합니다.

주: 속성이 디폴트에서 FILE SYSTEM CACHING 또는 NO FILE SYSTEM CACHING으로 변경되는 경우, 다시 디폴트로 변경하는 메커니즘이 없습니다.

파일 시스템 캐싱을 사용 및 사용 안하는 이 메소드는 테이블 스페이스 레벨에서 입출력 모드, 버퍼 또는 버퍼되지 않음을 제어합니다.

주: Long 필드(LF) 데이터 및 대형 오브젝트(LOB) 데이터에 대한 입출력 액세스는 문제가 되는 테이블 스페이스에 대한 설정과 상관 없이 SMS 및 DMS 컨테이너 모두에 대해 버퍼됩니다.

파일 시스템 캐싱이 사용되는지 여부를 판별하려면 MON_GET_TABLESPACE 테이블에 있는 테이블 스페이스에 대한 FS_CACHING 모니터 요소의 값을 쿼리하십시오.

UNIX, Linux 및 Windows에서 버퍼되지 않은 I/O를 사용 또는 사용하지 않기 위한 대체 방법

일부 UNIX 플랫폼에서는 MOUNT 옵션을 사용하여 파일 시스템 레벨에서 파일 시스템 캐싱을 사용 불가능하게 설정하도록 지원합니다. 자세한 정보는 운영 체제 문서를 참조하십시오. 그러나 파일 시스템 캐싱을 테이블 스페이스 레벨과 파일 시스템 레벨에서 사용 불가능하게 설정하는 것 사이에는 차이가 있음을 이해하는 것이 중요합니다. 테이블 스페이스 레벨에서 데이터베이스 관리

프로그램은 어떤 파일을 파일 시스템 캐싱으로 열지 및 파일 시스템 캐싱없이 열 것인지를 제어합니다. 파일 시스템 레벨에서는 특정 파일 시스템에 존재하는 모든 파일이 파일 시스템 캐싱없이 열립니다. 그러나 AIX와 같은 일부 플랫폼에서는 이 기능을 사용하기 전에 읽기 및 쓰기 액세스의 일련화와 같은 요구사항이 있습니다. 데이터베이스 관리 프로그램에서는 이러한 요구사항을 지키지만, 목표 파일 시스템에 비DB2 파일이 있는 경우 이 기능을 사용하기 전에 요구사항에 대해 운영 체제 문서를 참조하십시오.

주: 버전 8.1 FixPak 4에서 소개되었고 이제는 사용하지 않는 레지스트리 변수 DB2_DIRECT_IO는 AIX JFS2의 Long 필드 데이터, 대형 오브젝트(LOB) 데이터 및 임시 테이블 스페이스를 제외한 모든 SMS 컨테이너에 대한 파일 시스템 캐싱을 사용하지 않습니다. 이 레지스트리 변수를 DB2 버전 9.1 이상에서 설정하는 것은 NO FILE SYSTEM CACHING절을 사용하여 모든 테이블 스페이스, SMS 및 DMS를 변경하는 것과 동일합니다. 그러나 DB2_DIRECT_IO는 권장되지 않으며, 이 변수는 추후 릴리스에서 제거될 것입니다. 대신 테이블 스페이스 레벨에서 NO FILE SYSTEM CACHING을 사용해야 합니다.

Windows에서 버퍼되지 않은 I/O를 사용 또는 사용하지 않기 위한 대체 방법

이전 릴리스에서는 버퍼 풀 또는 sortheap을 늘릴 수 있도록 더 많은 메모리를 데이터베이스에 사용 가능하게 만들기 위해 성능 레지스트리 변수 DB2NTNOCACHE를 사용하여 모든 DB2 파일에 대한 파일 시스템 캐싱을 사용 안할 수 있습니다. 버전 9.5에서, DB2NTNOCACHE는 사용되지 않으며 추후 릴리스에서 제거될 수 있습니다. DB2NTNOCACHE와 NO FILE SYSTEM CACHING절을 사용할 때의 차이점은 선택적 테이블 스페이스에 대한 캐싱을 사용 불가능하게 설정할 수 있는 능력입니다. 버전 9.5부터 NO FILE SYSTEM CACHING이 디폴트로 사용되므로, FILE SYSTEM CACHING이 명시적으로 지정되지 않는 경우 인스턴스가 새로 작성된 테이블 스페이스만 포함하면 레지스트리 변수를 설정하여 전체 인스턴스 사이에 파일 시스템 캐싱을 사용 안할 필요가 없습니다.

성능 고려사항

버퍼되지 않은 입출력은 기본적으로 성능 향상을 위해 사용됩니다. 그러나 어떤 경우에는 작은 버퍼 풀 *크기*와 작은 파일 시스템 캐시의 조합으로 인해 성능이 저하될 수 있습니다(단, 이에 국한되지는 않음). 성능 향상을 위한 제안은 다음과 같습니다.

- 자체 성능 조정 메모리 관리자가 사용되지 않는 경우 사용 가능하게 하고 `ALTER BUFFERPOOL <name> SIZE AUTOMATIC`을 사용하여 버퍼 풀 *크기*를 자동으로 설정하십시오. 그러면 데이터베이스 관리 프로그램이 버퍼 풀 *크기*를 자체 성능 조정할 수 있습니다.

- 자체 성능 조정 메모리 관리자가 사용되지 않는 경우 성능이 개선될 때까지 버퍼 풀 크기를 10 또는 20% 단위로 늘리십시오.
- 자체 성능 조정 메모리 관리자가 사용되지 않는 경우 『FILE SYSTEM CACHING』을 사용하도록 테이블 스페이스를 변경하십시오. 그러면 버퍼되지 않은 입출력이 기본적으로 사용 불가능하며 컨테이너 액세스를 위한 버퍼된 입출력으로 되돌아갑니다.

성능 조정은 프로덕션 시스템에서 구현하기 전에 제어된 환경에서 테스트해야 합니다.

테이블 스페이스 컨테이너에 대해 파일 시스템 파일 대 디바이스를 사용할 것을 선택할 때 다음과 같이 수행되는 파일 시스템 캐싱을 고려해야 합니다.

- DMS 파일 컨테이너(및 모든 SMS 컨테이너)의 경우 운영 체제가 파일 시스템 캐시의 페이지를 캐시할 수 있습니다(테이블 스페이스가 NO FILESYSTEM CACHING으로 정의되지 않은 경우).
- DMS 디바이스 컨테이너 테이블 스페이스의 경우, 운영 체제는 파일 시스템 캐시에서 페이지를 캐시하지 않습니다.

CIO/DIO를 새 테이블 스페이스 컨테이너에 대한 디폴트 파일 시스템 캐시 메커니즘으로 사용

대부분의 AIX, Linux, Solaris 및 Windows 플랫폼에서 새로 작성되는 테이블 스페이스 컨테이너에 대한 디폴트 입출력 메커니즘은 CIO/DIO(동시 I/O 또는 직접 I/O)입니다. 이 디폴트는 트랜잭션 처리가 많은 워크로드 및 롤백에서 버퍼 지정 입출력에 대한 처리량을 늘립니다.

FILE SYSTEM CACHING 또는 NO FILE SYSTEM CACHING 속성은 입출력 조작이 파일 시스템 레벨에서 캐시되는지 여부를 지정합니다.

- FILE SYSTEM CACHING은 목표 테이블 스페이스의 모든 입출력 조작이 파일 시스템 레벨에서 캐시되도록 지정합니다.
- NO FILE SYSTEM CACHING은 모든 입출력 조작이 파일 시스템 레벨 캐시를 생략하도록 지정합니다.

주: DMS 테이블 스페이스를 사용할 때는 Long 필드(LF) 데이터 및 대형 오브젝트(LOB) 데이터에 대해 별도의 테이블 스페이스를 사용하여 일반 테이블 스페이스가 영향을 받지 않도록 해야 합니다. (SMS 테이블 스페이스의 경우 CIO/DIO(NO FILE SYSTEM CACHING) 속성이 사용되지 않습니다.)

다음 인터페이스에 FILE SYSTEM CACHING 속성이 들어있습니다.

- CREATE TABLESPACE문
- CREATE DATABASE 명령
- sqlecrea() API(SQLETSDESC 구조의 *sqlfscaching* 필드 사용)

이 속성이 CREATE TABLESPACE문이나 CREATE DATABASE 명령에서 지정되지 않으면 데이터베이스 관리 프로그램은 플랫폼 및 파일 시스템 유형을 기초로 디폴트 동작을 사용하여 요청을 처리합니다. 정확한 동작에 대해서는『파일 시스템 캐싱 구성』의 내용을 참조하십시오. sqlecrea() API의 경우 *sqlfscaching* 필드에 대한 0x2 값이 데이터베이스 관리 프로그램에게 디폴트 설정을 사용하도록 지시합니다.

다음 도구가 현재 FILE SYSTEM CACHING 속성에 대한 값을 해석합니다.

- GET SNAPSHOT FOR TABLESPACES 명령
- db2pd –tablespaces 명령
- db2look –d <dbname> –l 명령

db2look의 경우, FILE SYSTEM CACHING 속성이 지정되지 않으면 출력에 이 속성이 포함되지 않습니다.

예 :

데이터베이스 및 모든 관련 테이블 스페이스 컨테이너가 AIX JFS 파일 시스템에 상주하고 다음 명령문이 발행되었다고 가정하십시오.

```
DB2 CREATE TABLESPACE JFS2
```

이전 버전에서는 속성이 지정되지 않았으면 데이터베이스 관리 프로그램은 입출력 메커니즘에 대해 버퍼 지정된 입출력(FILE SYSTEM CACHING)을 사용했습니다. 버전 9.5에서는 데이터베이스 관리 프로그램이 NO FILE SYSTEM CACHING을 사용합니다.

파일 시스템 캐싱 구성

디폴트로 운영 체제에서는 디스크로부터 읽히고 디스크에 기록되는 파일 데이터를 캐시합니다.

일반적인 읽기 조작은 데이터를 디스크로부터 파일 시스템 캐시에서 읽은 다음 데이터를 캐시에서 응용프로그램 버퍼로 복사하는 실제 디스크 액세스를 포함합니다. 유사하게 쓰기 조작에는 데이터를 응용프로그램 버퍼에서 파일 시스템 캐시에 복사한 다음 이를 캐시에서 실제 디스크로 복사하는 실제 디스크 액세스를 포함합니다. 파일 시스템 레벨에서 이 데이터 캐싱 동작이 CREATE TABLESPACE문의 FILE SYSTEM CACHING절에서 반영됩니다. 데이터베이스 관리 프로그램이 버퍼 풀을 사용하여 자체 데이터 캐싱을 관리하기 때문에 버퍼 풀의 크기가 적절하게 조정된 경우 파일 시스템 레벨의 캐싱이 필요하지 않습니다.

주: 데이터베이스 관리 프로그램은 이미 AIX에서의 임시 데이터 및 LOB를 제외하고 캐시에서 페이지를 무효화하여 대부분의 DB2 데이터 캐싱을 금지합니다.

일부 경우에, 파일 시스템 레벨 및 버퍼 풀에서 캐싱할 경우, 더블 캐싱에 필요한 추가 CPU 주기 때문에 성능 저하가 발생합니다. 이 더블 캐시를 방지하기 위해 대부분의 파

일 시스템에는 파일 시스템 레벨에서 캐시를 사용 불가능하게 하는 기능이 있습니다. 이 기능은 일반적으로 *버퍼되지 않은 입출력*으로 불립니다. UNIX에서 이 기능은 일반적으로 *직접 입출력(*또는 *DIO)*으로 알려져 있습니다. Windows에서 이 기능은 FILE_FLAG_NO_BUFFERING 플래그를 사용하여 파일을 여는 것과 동일합니다. 또한 IBM JFS2 또는 Symantec VERITAS VxFS 같은 일부 파일 시스템은 또한 향상된 직접 입출력, 즉 고성능 *동시 I/O(CIO)* 기능을 지원합니다. 데이터베이스 관리 프로그램은 NO FILE SYSTEM CACHING 테이블 스페이스 절을 사용하여 이 기능을 지원합니다. 이 경우 데이터베이스 관리 프로그램이 자동으로 이 기능이 존재하는 파일 시스템에서 CIO를 활용합니다. 이 기능은 또한 파일 시스템 캐시의 메모리 요구사항을 줄여 다른 사용에서 사용 가능한 메모리의 *크기*를 늘립니다.

버전 9.5 이전에는, NO FILE SYSTEM CACHING 및 FILE SYSTEM CACHING이 지정되지 않은 경우 키워드 FILE SYSTEM CACHING이 암시되었습니다. 버전 9.5에서는, 어떤 키워드도 지정되지 않은 경우 디폴트인 NO FILE SYSTEM CACHING이 사용됩니다. 이 변경은 새로 작성되는 테이블 스페이스에만 영향을 줍니다. 버전 9.5 이전에 작성된 기존 테이블 스페이스는 영향을 받지 않습니다. 이 변경은 AIX, Linux, Solaris 및 Windows에 적용되며, 디폴트 동작이 FILE SYSTEM CACHING으로 남아 있는 다음 경우에는 예외입니다.

- AIX JFS
- Solaris 비VxFS
- System z®용 Linux
- 모든 SMS 임시 테이블 스페이스 파일
- SMS 영구 테이블 스페이스 파일에 있는 Long 필드(LF) 및 대형 오브젝트(LOB) 데이터 파일.

디폴트 설정을 겹쳐쓰려면 FILE SYSTEM CACHING 또는 NO FILE SYSTEM CACHING을 지정하십시오.

지원되는 구성

186 페이지의 표 11에서는 파일 시스템 캐싱없이 테이블 스페이스를 사용하기 위해 지원되는 구성을 표시합니다. 또한 (a) 각 경우에 DIO 또는 확장 DIO가 사용될지 여부 및 (b) 플랫폼 및 파일 시스템 유형을 기초로 테이블 스페이스에 대해 NO FILE SYSTEM CACHING 또는 FILE SYSTEM CACHING이 지정되지 않을 때의 디폴트 동작을 표시합니다.

표 11. 파일 시스템 캐싱 없이 테이블 스페이스에 대해 지원되는 구성

플랫폼	필수 파일 시스템 유형 및 최소 레벨	**NO FILE SYSTEM CACHING**이 지정될 때 데이터베이스 관리 프로그램이 제출하는 **DIO** 또는 **CIO** 요청	**NO FILE SYSTEM CACHING** 또는 **FILE SYSTEM CACHING**이 지정되지 않을 때의 디폴트 동작
AIX 5.3 이상	JFS(Journal File System)	DIO	FILE SYSTEM CACHING (주 1 참조)
AIX 5.3 이상	GPFS™(General Parallel File System)	DIO	NO FILE SYSTEM CACHING
AIX 5.3 이상	동시 저널 파일 시스템(JFS2)	CIO	NO FILE SYSTEM CACHING
AIX 5.3 이상	VERITAS Storage Foundation for DB2 4.1(VxFS)	CIO	NO FILE SYSTEM CACHING
HP-UX 버전 11i v2(Itanium®)	VERITAS Storage Foundation 4.1(VxFS)	CIO	FILE SYSTEM CACHING
Solaris 9	UFS(UNIX File System)	DIO	FILE SYSTEM CACHING (주 2 참조)
Solaris 10	UFS(UNIX File System)	CIO	FILE SYSTEM CACHING (주 2 참조)
Solaris 9, 10	VERITAS Storage Foundation for DB2 4.1(VxFS)	CIO	NO FILE SYSTEM CACHING
Linux 분산 SLES 10 SP2 이상 및 RHEL 5.2 이상 (x86, x64, POWER® 아키텍처에서)	ext2, ext3, reiserfs	DIO	NO FILE SYSTEM CACHING
Linux 분산 SLES 10 SP2 이상 및 RHEL 5.2 이상 (x86, x64, POWER 아키텍처에서)	VERITAS Storage Foundation 4.1(VxFS)	CIO	NO FILE SYSTEM CACHING
Linux 분산 SLES 10 SP2 이상 및 RHEL 5.2 이상 (zSeries 아키텍처에서)	FCP(Fibre Channel Protocol)을 사용하는 SCSI(Small Computer System Interface) 디스크의 ext2, ext3 또는 reiserfs	DIO	FILE SYSTEM CACHING
Windows	특정 요구사항이 없고, 모든 DB2 지원 파일 시스템에서 작업	DIO	NO FILE SYSTEM CACHING

주:

1. AIX JFS에서는, FILE SYSTEM CACHING이 디폴트입니다.
2. Solaris UFS에서는, NO FILE SYSTEM CACHING이 디폴트입니다.
3. 데이터베이스 관리 프로그램용 VERITAS Storage Foundation은 다른 운영 체제 전제조건을 가질 수 있습니다. 위에서 나열되는 플랫폼은 현재 릴리스에 대해 지원되는 플랫폼입니다. 전제조건 정보는 VERITAS Storage Foundation for DB2 지원을 참조하십시오.

4. SFDB2 5.0이 위의 최소 레벨 대신 사용되는 경우 SFDB2 5.0 MP1 RP1 릴리스를 사용해야 합니다. 이 릴리스에는 5.0 버전에 특정한 수정이 포함됩니다.
5. 데이터베이스 관리 프로그램에서 디폴트 설정에 대해 NO FILE SYSTEM CACHING을 선택하지 않도록 하려면 관련 SQL, 명령 또는 API에서 FILE SYSTEM CACHING을 지정하십시오.

예

예 1: 디폴트로 이 새 테이블 스페이스는 버퍼되지 않은 입출력을 사용하여 작성됩니다. NO FILE SYSTEM CACHING절이 내재됩니다.

```
CREATE TABLESPACE table space name ...
```

예 2: 다음 명령문에서 NO FILE SYSTEM CACHING절은 이 특정 테이블 스페이스에 대해 파일 시스템 레벨 캐싱이 OFF임을 표시합니다.

```
CREATE TABLESPACE table space name ... NO FILE SYSTEM CACHING
```

예 3: 다음 명령문은 기존 테이블 스페이스에 대해 파일 시스템 레벨 캐싱을 사용 안합니다.

```
ALTER TABLESPACE table space name ... NO FILE SYSTEM CACHING
```

예 4: 다음 명령문은 기존 테이블 스페이스에 대해 파일 시스템 레벨 캐싱을 사용 가능하게 합니다.

```
ALTER TABLESPACE table space name ... FILE SYSTEM CACHING
```

테이블 스페이스의 Extent 크기

*Extent*는 테이블 스페이스 컨테이너에 있는 스토리지의 블록입니다. 다음 컨테이너에 기록하기 전에 컨테이너에 기록될 데이터의 페이지 수를 나타냅니다. 테이블 스페이스를 작성할 때 성능 및 스토리지 관리에 대한 요구사항을 기초로 Extent 크기를 선택할 수 있습니다.

Extent 크기를 선택할 때 다음을 고려하십시오.

- 테이블 스페이스의 테이블 크기 및 유형

  DMS 테이블 스페이스 내의 스페이스는 한 번에 한 Extent만큼 테이블에 할당됩니다. 내용이 채워지고 Extent가 가득 차면, 새 Extent가 할당됩니다. DMS 테이블 스페이스 컨테이너 스토리지는 사전 예약되어 있습니다. 즉, 컨테이너가 완전히 사용될 때까지 새 Extent가 할당된다는 것을 의미합니다.

  SMS 테이블 스페이스 내의 스페이스는 한 번에 한 Extent 또는 한 페이지만큼 테이블에 할당됩니다. 내용이 채워지면서 Extent 또는 페이지가 가득 차면, 파일 시스템의 모든 Extent 또는 페이지가 사용될 때까지 새 Extent가 할당됩니다. SMS 테

이블 스페이스를 사용하는 경우에는 다중 페이지 파일 할당이 허용됩니다. 다중 페이지 파일 할당을 사용하면 한 번에 페이지 대신 Extent를 할당할 수 있습니다.

기본적으로 다중 페이지 파일 할당이 사용 가능합니다. *multipage_alloc* 데이터베이스 구성 매개변수의 값은 다중 파일 할당이 사용 가능한지 여부를 표시합니다.

주: 다중 페이지 파일 할당은 임시 테이블 스페이스에는 적용되지 않습니다.

테이블은 다음의 개별 테이블 오브젝트로 구성됩니다.

- 데이터 오브젝트. 일반 컬럼 데이터를 저장합니다.
- 인덱스 오브젝트. 테이블에 정의된 모든 인덱스를 저장합니다.
- Long 필드(LF) 데이터 오브젝트. 테이블에 하나 이상의 LONG 컬럼이 있는 경우, long 필드 데이터를 저장합니다.
- 두 개의 대형 오브젝트(LOB) 데이터 오브젝트. 테이블에 하나 이상의 LOB 컬럼이 있는 경우, 이 컬럼은 다음 두 테이블 오브젝트에 저장됩니다.
  - LOB 데이터용 테이블 오브젝트
  - LOB 데이터를 설명하는 메타데이터용 테이블 오브젝트
- 다차원적으로 클러스터된(MDC) 테이블에 대한 블록 맵 오브젝트
- XML 문서를 저장하는 여분의 XDA 오브젝트

각 테이블 오브젝트는 따로 저장되고 각 오브젝트는 필요에 따라 새 Extent를 할당합니다. 또한, 각 DMS 테이블 오브젝트는 테이블 오브젝트에 속하는 테이블 스페이스의 모든 Extent를 나타내는 Extent Map이라는 메타데이터 오브젝트와 한쌍이 됩니다. 또한, Extent 맵핑 스페이스는 한 번에 한 Extent만큼 할당됩니다. 그러므로 DMS 테이블 스페이스의 경우 오브젝트용 스페이스의 초기 할당은 두 개의 Extent입니다. (SMS 테이블 스페이스의 경우 오브젝트용 스페이스의 초기 할당은 하나의 페이지입니다.)

DMS 테이블 스페이스에 소형 테이블이 많이 있으면, 상대적으로 적은 데이터량을 저장하기 위해 상대적으로 많은 스페이스량이 할당될 수 있습니다. 이 경우, 작은 Extent *크기*를 지정해야 합니다. 반면에, 고속으로 증가하는 초대형 테이블을 가지고 있고 작은 Extent *크기*를 사용하는 DMS 테이블 스페이스를 사용하는 중인 경우, 추가 Extent의 잦은 할당으로 인해 불필요한 오버헤드가 발생할 수 있습니다.

- 테이블에 대한 액세스 유형

  테이블 액세스에 대용량 데이터를 처리하는 많은 쿼리 또는 트랜잭션이 포함되는 경우 테이블에서 데이터를 프리페치하여 성능을 상당히 향상시킬 수 있습니다.

- 필수 Extent의 최소 수

  테이블 스페이스의 5개의 Extent용 컨테이너에 충분한 스페이스가 없는 경우, 테이블 스페이스를 작성할 수 없습니다.

페이지, 테이블 및 테이블 스페이스 크기

DMS, 임시 DMS 및 임시가 아닌 자동 스토리지 테이블 스페이스의 경우, 데이터베이스에 대해 사용자가 선택하는 페이지 *크기*가 테이블 스페이스 *크기*의 상한을 결정합니다. SMS 및 임시 자동 스토리지 테이블 스페이스의 테이블의 경우 페이지 *크기*가 테이블 자체의 *크기*를 제한합니다.

4K, 8K, 16K 또는 32K 페이지 *크기* 한계를 사용할 수 있습니다. 이들 각 페이지 *크기*는 또한 사용자가 지켜야 하는 각 테이블 스페이스 유형에 대한 최대값을 갖습니다.

표 12는 DMS 및 임시가 아닌 자동 스토리지 테이블 스페이스에 대한 테이블 스페이스 *크기* 한계를 페이지 *크기*별로 표시합니다.

표 12. DMS 및 임시가 아닌 자동 스토리지 테이블 스페이스에 대한 크기 한계. DMS 및 임시가 아닌 자동 스토리지 테이블 스페이스는 페이지 *크기*에 의해 제한됩니다.

테이블 스페이스 유형	4K 페이지 *크기* 한계	8K 페이지 *크기* 한계	16K 페이지 *크기* 한계	32K 페이지 *크기* 한계
DMS 및 임시가 아닌 자동 스토리지 테이블 스페이스(일반)	64G	128G	256G	512G
DMS, 임시 DMS 및 임시가 아닌 자동 스토리지 테이블 스페이스(대형)	8192G	16,384G	32,768G	65,536G

표 13은 SMS 및 임시 자동 스토리지 테이블 스페이스에 있는 테이블의 테이블 *크기* 한계를 페이지 *크기*별로 표시합니다.

표 13. SMS 및 임시 자동 스토리지 테이블 스페이스에 있는 테이블에 대한 크기 한계. SMS 및 임시 자동 스토리지 테이블 스페이스의 테이블의 경우 페이지 *크기*로 제한되는 테이블 스페이스가 아니라 테이블 오브젝트의 *크기*입니다.

테이블 스페이스 유형	4K 페이지 *크기* 한계	8K 페이지 *크기* 한계	16K 페이지 *크기* 한계	32K 페이지 *크기* 한계
SMS	64G	128G	256G	512G
임시 SMS, 임시 자동 스토리지	8192G	16,384G	32,768G	65,536G

여러 가지 유형의 테이블 스페이스에 대한 데이터베이스 및 인덱스 페이지 *크기* 한계에 대해서는 *SQL 참조서*의 『SQL 및 XML 한계』에 있는 데이터베이스 관리 프로그램 페이지 *크기* 특정 한계를 참조하십시오.

시스템 임시 테이블 스페이스의 최대 페이지 *크기*가 쿼리 또는 위치지정된 갱신에 충분히 큰지 확인하려면 *DB2 버전 9.7로 업그레이드*의 『시스템 임시 테이블 스페이스 페이지 *크기*가 요구사항을 만족하는지 확인』을 참조하십시오.

디스크 입출력 효율 및 테이블 스페이스 설계

테이블 스페이스의 유형과 설계에 따라 테이블 스페이스에서 이뤄지는 입출력의 효율성이 결정됩니다.

테이블 스페이스 설계 및 사용에 관한 기타 문제를 고려하기 전에 다음 개념을 이해해야 합니다.

큰 블록 읽기
: 한 번의 요청으로 여러 페이지(보통 Extent)가 검색되는 읽기 방식. 동시에 여러 페이지를 읽는 것은 각 페이지를 별도로 읽는 것보다 더 효율적입니다.

프리페치
: 쿼리가 페이지를 참조하기 전에 페이지를 읽는 방식. 전반적인 목적은 응답 시간을 줄이는 것입니다. 쿼리 실행시 페이지 프리페치가 비동기식으로 발생할 경우 응답 시간을 줄일 수 있습니다. CPU 또는 입출력 서브시스템이 최대의 성능을 내며 작동할 때 응답 시간이 최적이 됩니다.

페이지 정리
: 페이지를 읽고 변경할 때, 이들 페이지는 데이터베이스 버퍼 풀에 쌓이게 됩니다. 읽은 페이지는 버퍼 페이지에서도 읽힙니다. 버퍼 풀이 변경된 페이지로 가득 찰 경우, 새 페이지를 읽기 전에 이들 변경된 페이지 중 하나를 디스크에 기록해야 합니다. 버퍼 풀이 가득 차지 않게 하기 위해, 페이지 클리너 에이전트는 변경된 페이지를 기록하여 이후 읽기 요청이 있을 때 버퍼 풀 페이지가 사용될 수 있도록 합니다.

큰 블록 읽기가 도움이 될 때마다 데이터베이스 관리 프로그램은 이를 수행합니다. 보통 완전히 순차적이거나 부분적으로 순차적이라는 특성을 갖는 데이터를 검색할 때 큰 블록 읽기가 발생합니다. 한 번의 읽기 조작으로 읽을 수 있는 데이터의 양은 Extent *크기*에 따라 다릅니다. Extent *크기*가 클수록, 더 많은 페이지를 한 번에 읽을 수 있습니다.

페이지를 디스크에서 버퍼 풀 내의 인접한 페이지로 읽을 수 있는 경우, 순차 프리페치 성능이 더욱 향상될 수 있습니다. 버퍼 풀은 디폴트로 페이지를 기반으로 하기 때문에 디스크에서 인접한 페이지들을 읽을 때 인접한 페이지 세트를 찾는다는 보장이 없습니다. 블록을 기반으로 하는 버퍼 풀에는 페이지 영역뿐만 아니라 인접한 페이지 세트의 블록 영역도 포함하기 때문에 이러한 용도로 사용할 수 있습니다. 각 인접 페이지 세트를 블록이라고 하며 각 블록에는 블록 *크기*라고 하는 많은 수의 페이지가 들어 있습니다. 페이지 및 블록 영역의 *크기*와 각 블록의 페이지 수는 구성할 수 있습니다.

Extent가 디스크에 저장되는 방식에 따라 입출력 효율성이 달라집니다. 디바이스 컨테이너를 사용하는 DMS 테이블 스페이스에서, 데이터는 디스크에 인접하려는 경향이 있으며 찾기 시간과 디스크 대기 시간을 최소로 설정하여 읽을 수 있습니다. 파일이 사용중인 경우 DMS 테이블 스페이스용으로 사전 할당된 대용량 파일은 디스크에 인접하려는 경향이 있습니다. 특히 파일이 새 파일 스페이스에 할당되는 경우에 더욱 그러합니다. 그러나 데이터가 파일 시스템에 의해 나누어져 디스크에서 둘 이상의 위치에 저장되었을 수 있습니다. 파일이 한 번에 한 페이지씩 확장되어 분할이 더 잘 일어날 수 있는 SMS 테이블 스페이스를 사용할 때 이러한 현상이 자주 발생할 수 있습니다.

CREATE TABLESPACE 또는 ALTER TABLESPACE문의 PREFETCHSIZE 옵션을 변경하여 프리페치 정도를 제어하거나, 프리페치 크기를 AUTOMATIC으로 설정하여 데이터베이스 관리 프로그램이 자동으로 사용할 최상의 크기를 선택하도록 할 수 있습니다. (데이터베이스의 모든 테이블 스페이스의 디폴트값은 *dft_prefetch_sz* 데이터베이스 구성 매개변수에 의해 설정됩니다.) PREFETCHSIZE 매개변수는 데이터베이스 관리 프로그램에 프리페치가 트리거될 때마다 읽을 페이지 수를 알려줍니다. CREATE TABLESPACE문에서 PREFETCHSIZE를 EXTENTSIZE 매개변수의 배수로 설정하면 여러 Extent를 병렬로 읽을 수 있습니다. (데이터베이스의 모든 테이블 스페이스의 디폴트값은 *dft_extent_sz* 데이터베이스 구성 매개변수에 의해 설정됩니다.) EXTENTSIZE 매개변수는 다음 컨테이너로 이동하기 전에 컨테이너에 쓰여지는 4KB 페이지의 수를 지정합니다.

예를 들어, 세 개의 디바이스를 사용한 테이블 스페이스가 있다고 가정하십시오. PREFETCHSIZE를 EXTENTSIZE의 세 배로 설정할 경우, 데이터베이스 관리 프로그램은 각 디바이스로부터 큰 블록 읽기를 병렬로 수행할 수 있으므로 입출력 처리량이 현저하게 늘어납니다. 이 경우, 각 디바이스는 별도의 물리적 디바이스이며, 제어기는 각 디바이스로부터의 데이터 스트림을 처리할 만큼 충분한 대역폭을 가지고 있는 것으로 가정됩니다. 데이터베이스 관리 프로그램은 쿼리 속도, 버퍼 풀 사용 및 기타 요소를 기본으로 하여 런타임시 프리페치 매개변수를 동적으로 조정해야 합니다.

일부 파일 시스템에서는 고유 프리페치 메소드(예: AIX의 JFS(Journaled File System))를 사용합니다. 어떤 경우에는, 파일 시스템 프리페치가 데이터베이스 관리 프로그램 프리페치보다 더 적극적으로 설정됩니다. 이 결과, 파일 컨테이너가 있는 SMS 및 DMS 테이블 스페이스에 대한 프리페치는 디바이스가 있는 DMS 테이블 스페이스의 프리페치보다 뛰어나게 됩니다. 파일 시스템에서 일어나는 추가 프리페치 레벨 결과와 같기 때문에 약간의 혼돈이 있을 수 있지만, DMS 테이블 스페이스는 어떤 동등한 구성보다 기능이 뛰어납니다.

프리페치(또는 읽기)를 효과적으로 수행하려면, 데이터를 읽어들일 빈 버퍼 풀 페이지가 충분히 있어야 합니다. 예를 들어, 테이블 스페이스로부터 세 개의 Extent를 읽어들이는 병렬 프리페치 요청이 있을 수 있으며 읽고 있는 각 페이지에 대해 하나의 수정된 페이지가 버퍼 풀에서 기록됩니다. 프리페치 요청이 쿼리를 계속할 수 없는 지점까지 느려질 수 있습니다. 프리페치 요청을 충족시킬 만큼 충분한 수의 페이지 클리너가 구성되어 있어야 합니다.

테이블 스페이스 작성

데이터베이스에서 테이블 스페이스를 작성하면 컨테이너가 테이블 스페이스에 지정되고 정의 및 속성이 데이터베이스 시스템 카탈로그에 기록됩니다.

자동 스토리지 테이블 스페이스의 경우 데이터베이스 관리 프로그램이 데이터베이스와 연관된 스토리지 경로에 기초하여 테이블 스페이스에 컨테이너를 지정합니다.

비자동 스토리지 테이블 스페이스의 경우 테이블 스페이스를 작성할 때 사용할 컨테이너의 경로, 디바이스 또는 파일 이름을 알아야 합니다. 또한 DMS 테이블 스페이스에 대해 작성하는 각 디바이스 또는 파일 컨테이너에 대해 각 컨테이너에 할당할 수 있는 스토리지 스페이스 수를 알아야 합니다.

프로시저

명령행을 사용하여 SMS 테이블 스페이스를 작성하려면 다음을 입력하십시오.

```
CREATE TABLESPACE name
        MANAGED BY SYSTEM
  USING ('path')
```

명령행을 사용하여 DMS 테이블 스페이스를 작성하려면 다음을 입력하십시오.

```
CREATE TABLESPACE name
        MANAGED BY DATABASE
  USING (FILE 'path' size)
```

디폴트로 DMS 테이블 스페이스는 대형 테이블 스페이스로서 작성됩니다.

명령행을 사용하여 자동 스토리지 테이블 스페이스를 작성하려면 다음 명령문 중 하나를 입력하십시오.

```
CREATE TABLESPACE  name
```

또는

```
CREATE TABLESPACE name
  MANAGED BY AUTOMATIC STORAGE
```

테이블 스페이스가 자동 스토리지 데이터베이스에 작성된다고 가정하면 위의 두 명령문은 동등합니다. 그러한 데이터베이스에 작성되는 테이블 스페이스는 사용자가 별도로 지정하지 않는 한 디폴트로 자동 스토리지 테이블 스페이스가 됩니다.

예 1: Windows에서 SMS 테이블 스페이스 작성 다음 SQL문은 세 개의 별도의 드라이브에 있는 세 개의 디렉토리에 RESOURCE라는 SMS 테이블 스페이스를 작성합니다.

```
 CREATE TABLESPACE RESOURCE
      MANAGED BY SYSTEM
USING ('d:₩acc_tbsp', 'e:₩acc_tbsp', 'f:₩acc_tbsp')
```

예 2: Windows에서 DMS 테이블 스페이스 작성 다음 SQL문은 각각 5,000페이지를 갖는 두 파일 컨테이너를 갖는 DMS 테이블 스페이스를 작성합니다.

```
 CREATE TABLESPACE RESOURCE
      MANAGED BY DATABASE
USING (FILE'd:₩db2data₩acc_tbsp' 5000,
        FILE'e:₩db2data₩acc_tbsp' 5000)
```

이전 두 예에서 컨테이너에 대해 명시적인 이름이 제공됩니다. 그러나 상대 컨테이너 이름을 지정하면, 데이터베이스용으로 작성된 서브디렉토리에 컨테이너가 작성됩니다.

테이블 스페이스 컨테이너를 작성할 때 데이터베이스 관리 프로그램이 존재하지 않는 모든 디렉토리 레벨을 작성합니다. 예를 들어 컨테이너가 /project/user_data/container1로 지정되고 /project 디렉토리가 존재하지 않는 경우 데이터베이스 관리 프로그램이 /project 및 /project/user_data 디렉토리를 작성합니다.

데이터베이스 관리 프로그램이 작성하는 모든 디렉토리는 PERMISSION 700으로 작성됩니다. 이것은 인스턴스 소유자만이 읽기, 쓰기 및 실행 액세스 권한이 있음을 의미합니다. 인스턴스 소유자만 이 액세스 권한을 갖기 때문에 다중 인스턴스가 작성되고 있는 경우 다음 시나리오가 발생할 수 있습니다.

- 위에서 설명한 것과 동일한 디렉토리 구조를 사용하여 디렉토리 레벨 /project/user_data가 없다고 가정하십시오.
- user1이 디폴트로 user1로 이름 지정된 인스턴스를 작성한 후 데이터베이스를 작성하고, 해당 컨테이너 중 하나로 /project/user_data/container1을 갖는 테이블 스페이스를 작성합니다.
- user2가 디폴트값으로 user2 이름의 인스턴스를 작성하고 데이터베이스를 작성한 후 컨테이너 중 하나로 /project/user_data/container2를 갖는 테이블 스페이스를 작성합니다.

데이터베이스 관리 프로그램이 첫 번째 요청에서 PERMISSION 700을 갖는 디렉토리 레벨 /project/user_data를 작성했기 때문에 user2는 이들 디렉토리 레벨에 액세스할 수 없으며 해당 디렉토리에 container2를 작성할 수 없습니다. 이 경우 CREATE TABLESPACE 조작이 실패합니다.

이 충돌을 해결할 수 있는 두 방법은 다음과 같습니다.

1. 테이블 스페이스를 작성하기 전에 /project/user_data 디렉토리를 작성하고 user1 및 user2가 테이블 스페이스를 작성하는데 필요한 모든 액세스에 대해 사용 권한을 설정하십시오. 테이블 스페이스 디렉토리의 모든 레벨이 존재하는 경우 데이터베이스 관리 프로그램은 액세스를 수정하지 않습니다.
2. user1이 /project/user_data/container1을 작성한 후 user2가 테이블 스페이스를 작성하는데 필요한 모든 액세스를 위해 /project/user_data의 사용 권한을 설정하십시오.

서브디렉토리가 데이터베이스 관리 프로그램에 의해 작성되는 경우 테이블 스페이스가 삭제될 때 데이터베이스 관리 프로그램이 삭제할 수도 있습니다.

이 시나리오는 테이블 스페이스가 특정 데이터베이스 파티션 그룹과 연관되지 않은 것으로 가정합니다. 명령문에 다음 매개변수를 지정하지 않으면 디폴트 데이터베이스 파티션 그룹 IBMDEFAULTGROUP이 사용됩니다.

```
    IN database_partition_group_name
```

예 3: AIX에서 DMS 테이블 스페이스 작성 다음 SQL문은 각각 10,000페이지의 세 논리적 볼륨을 사용하여 AIX 시스템에 DMS 테이블 스페이스를 작성하고 입출력 특성을 지정합니다.

```
CREATE TABLESPACE RESOURCE
     MANAGED BY DATABASE
     USING (DEVICE '/dev/rdblv6' 10000,
                   DEVICE'/dev/rdblv7' 10000,
                   DEVICE'/dev/rdblv8' 10000)
     OVERHEAD 7.5
TRANSFERRATE 0.06
```

이 SQL문에서 언급된 UNIX 디바이스가 이미 존재해야 하며, 인스턴스 소유자 및 SYSADM 그룹이 이 디바이스에 기록할 수 있어야 합니다.

예 4: UNIX 시스템에서 DMS 테이블 스페이스 작성 다음 예에서는 UNIX 다중 파티션 데이터베이스에서 ODDGROUP이라는 데이터베이스 파티션 그룹에 DMS 테이블 스페이스를 작성합니다. 이전에 ODDGROUP이 CREATE DATABASE PARTITION GROUP문으로 작성되어 있어야 합니다. 이 경우 ODDGROUP 데이터베이스 파티션 그룹은 1, 3, 5로 번호가 매겨진 데이터베이스 파티션으로 구성된 것으로 가정됩니다. 모든 데이터베이스 파티션에서 10,000개의 4KB 페이지에 대해 `/dev/hdisk0` 디바이스를 사용하십시오. 또한 각 데이터베이스 파티션에 대해 40,000개의 4KB 페이지의 디바이스를 선언하십시오.

```
CREATE TABLESPACE PLANS IN ODDGROUP
     MANAGED BY DATABASE
     USING (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n1hd01' 40000)
                 ON DBPARTITIONNUM 1
               (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n3hd03' 40000)
                 ON DBPARTITIONNUM 3
               (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n5hd05' 40000)
     ON DBPARTITIONNUM 5
```

데이터베이스 관리 프로그램은 시퀀스 프리페치 기능을 사용하여 순차 입출력의 성능을 크게 향상시킬 수 있는데, 이 기능은 병렬 I/O를 사용합니다.

예 5: 디폴트보다 큰 페이지 크기를 갖는 SMS 테이블 스페이스 작성 또한 디폴트값인 4KB보다 큰 페이지 크기를 사용하는 테이블 스페이스를 작성할 수도 있습니다. 다음 SQL문은 Linux 및 UNIX 시스템에 8KB 페이지 크기의 SMS 테이블 스페이스를 작성합니다.

```
CREATE TABLESPACE SMS8K
     PAGESIZE 8192
     MANAGED BY SYSTEM
     USING ('FSMS_8K_1')
BUFFERPOOL BUFFPOOL8K
```

연관된 버퍼 풀 또한 동일한 8KB 페이지 크기여야 한다는 점을 유의하십시오.

참조하는 버퍼 풀이 활성화될 때까지는 작성된 테이블 스페이스를 사용할 수 없습니다.

ALTER TABLESPACE문을 사용하여 DMS 테이블 스페이스에 컨테이너를 추가, 삭제하거나 컨테이너의 *크기*를 조정하고 테이블 스페이스에 대한 PREFETCHSIZE, OVERHEAD 및 TRANSFERRATE 설정을 수정할 수 있습니다. 시스템 카탈로그 경합을 막으려면 ALTER TABLESPACE SQL문 이 후 테이블 스페이스 명령문을 실행하는 트랜잭션을 가능한 빨리 커미트해야 합니다.

주: PREFETCHSIZE 값은 EXTENTSIZE 값의 배가 되어야 합니다. 예를 들어, EXTENTSIZE가 10이면 PREFETCHSIZE는 20 또는 30이 되어야 합니다. 테이블 스페이스를 작성할 경우 프리페치 *크기*를 수동으로 설정하려면 다음과 같은 등식을 사용해야 합니다.

프리페치 크기 = (*컨테이너 수*) × (*컨테이너당 물리적 스핀의 수*) × *Extent 크기*

또한 PREFETCHSIZE를 AUTOMATIC으로 설정하여 데이터베이스 관리 프로그램이 자동으로 프리페치 *크기*를 판별하도록 설정하는 것도 고려해야 합니다.

직접 입출력(DIO)은 파일 시스템 레벨에서 캐싱을 생략하기 때문에 메모리 성능을 향상시킵니다. 이 프로세스는 CPU 오버헤드를 줄이고 데이터베이스 인스턴스에 사용 가능한 메모리를 더 작성합니다.

동시 입출력(CIO)은 DIO의 장점을 지니며 *쓰기* 액세스의 순번 매김을 완화합니다.

DIO 및 CIO는 AIX에서 지원되며, DIO는 HP-UX, Solaris, Linux 및 Windows 운영 체제에서 지원됩니다.

NO FILE SYSTEM CACHING 및 FILE SYSTEM CACHING 키워드는 CREATE 및 ALTER TABLESPACE SQL문의 일부로 DIO 또는 CIO가 각 테이블 스페이스에 사용되는지 여부를 지정합니다. NO FILE SYSTEM CACHING이 적용되면, 데이터베이스 관리 프로그램에서는 가능한 모든 경우에 동시 입출력(CIO) 사용을 시도합니다. CIO가 지원되지 않을 경우(예: JFS가 사용되는 경우) DIO가 대신 사용됩니다.

CREATE TABLESPACE 명령문을 발행할 때 삭제된 테이블 복구 기능이 디폴트로 켜집니다. 이 기능을 사용하면 테이블 스페이스 레벨 리스토어 및 롤 포워드 조작을 사용하여 삭제된 테이블 데이터를 복구할 수 있습니다. 이는 데이터베이스 레벨 복구보다 빨라서 유용하며, 데이터베이스가 사용자에게 사용할 수 있는 상태로 유지됩니다.

그러나 삭제된 테이블 복구 기능은 복구할 테이블 삭제 조작이 많거나 실행기록 파일이 너무 큰 경우 포워드 복구 성능에 다소 영향을 줄 수 있습니다.

수많은 테이블 삭제 조작을 실행할 계획이고 순환 로깅을 사용하거나 어떤 삭제된 테이블도 복구할 것으로 생각하지 않는 경우에 이 기능을 사용하지 않을 수 있습니다. 이 기능을 사용하지 않기 위해, CREATE TABLESPACE 명령문을 발행할 때 명시적으

로 DROPPED TABLE RECOVERY 옵션을 OFF로 설정할 수 있습니다. 다른 방법으로는 ALTER TABLESPACE 명령문을 사용하여 기존 테이블 스페이스에 대한 삭제된 테이블 복구 기능을 끌 수 있습니다.

임시 테이블 스페이스 작성

임시 테이블 스페이스는 정렬 또는 조인과 같은 조작을 수행할 때 이러한 활동은 결과 세트를 처리하기 위해 추가 스페이스가 필요하므로 데이터베이스 관리 프로그램에 필요한 임시 데이터를 보유합니다. CREATE TABLESPACE 명령의 변형을 사용하여 임시 테이블 스페이스를 작성합니다.

이 태스크에 대한 정보

*시스템 임시 테이블 스페이스*는 시스템 임시 테이블을 저장하는 데 사용됩니다. 시스템 임시 테이블이 이러한 테이블 스페이스에 저장될 수 있으므로 데이터베이스는 항상 최소한 하나의 시스템 임시 테이블 스페이스를 가져야 합니다. 데이터베이스가 작성되면, 정의된 세 가지 디폴트 테이블 스페이스 중 하나는 "TEMPSPACE1"이라고 하는 시스템 임시 테이블 스페이스입니다. 데이터베이스에 존재하는 사용자 테이블 스페이스에 대한 각 페이지 크기의 시스템 임시 테이블 스페이스를 하나 이상 가져야 합니다. 그렇지 않으면 일부 쿼리가 실패할 수 있습니다. 자세한 정보는 139 페이지의 『시스템, 사용자 및 임시 데이터용 테이블 스페이스』의 내용을 참조하십시오.

*사용자 임시 테이블 스페이스*는 데이터베이스가 작성될 때 디폴트로 작성되지 않습니다. 응용프로그램이 임시 테이블을 사용해야 하는 경우, 임시 테이블이 상주할 사용자 임시 테이블 스페이스를 작성해야 합니다. 일반 테이블 스페이스와 같이 사용자 임시 테이블 스페이스도 IBMTEMPGROUP 이외의 데이터베이스 파티션 그룹에 작성될 수 있습니다. IBMDEFAULTGROUP은 사용자 임시 테이블을 작성할 때 사용되는 디폴트 데이터베이스 파티션 그룹입니다.

제한사항

파티션된 환경에 있는 시스템 임시 테이블 스페이스의 경우, 시스템 임시 테이블 스페이스를 작성할 때 지정할 수 있는 유일한 데이터베이스 파티션 그룹은 IBMTEMPGROUP입니다.

프로시저

디폴트 TEMPSPACE1 외에 추가로 시스템 임시 테이블 스페이스를 작성하려면 SYSTEM TEMPORARY 키워드를 포함하는 CREATE TABLESPACE문을 사용하십시오. 예를 들어,

```
CREATE SYSTEM TEMPORARY TABLESPACE tmp_tbsp
     MANAGED BY SYSTEM
     USING ('d:₩tmp_tbsp','e:₩tmp_tbsp')
```

사용자 임시 테이블 스페이스를 작성하려면 USER TEMPORARY 키워드와 함께 CREATE TABLESPACE문을 사용하십시오. 예를 들어,

```
CREATE USER TEMPORARY TABLESPACE usr_tbsp
     MANAGED BY DATABASE
     USING (FILE 'd:₩db2data₩user_tbsp' 5000,
FILE 'e:₩db2data₩user_tbsp' 5000)
```

데이터베이스 작성 시 초기 테이블 스페이스 정의

데이터베이스가 작성될 때 (1) 시스템 카탈로그 테이블을 위한 SYSCATSPACE, (2) 데이터베이스 처리 중에 작성되는 시스템 임시 테이블을 위한 TEMPSPACE1 및 (3) 사용자 정의 테이블 및 인덱스를 위한 USERSPACE1이라는 세 테이블 스페이스가 정의됩니다. 또한 동시에 추가 사용자 테이블 스페이스를 작성할 수도 있습니다.

주: 처음 데이터베이스를 작성할 때 사용자 임시 테이블 스페이스가 작성되지 않습니다.

별도로 지정하지 않으면 세 개의 디폴트 테이블 스페이스는 자동 스토리지에 의해 관리됩니다.

CREATE DATABASE 명령을 사용하여 디폴트 버퍼 풀 및 초기 테이블 스페이스의 페이지 *크기*를 지정할 수 있습니다. 이 디폴트값은 이후 모든 CREATE BUFFERPOOL 및 CREATE TABLESPACE문의 디폴트 페이지 *크기*를 나타냅니다. 데이터베이스 작성 시 페이지 *크기*를 지정하지 않는 경우 디폴트 페이지 *크기*는 4KB입니다.

명령행을 사용하여 초기 테이블 스페이스를 정의하려면 다음을 입력하십시오.

```
CREATE DATABASE name
  PAGESIZE page size
       CATALOG TABLESPACE
    MANAGED BY SYSTEM USING ('path')
          EXTENTSIZE value PREFETCHSIZE value
       USER TABLESPACE
          MANAGED BY DATABASE USING (FILE'path' 5000,
                                     FILE 'path' 5000)
          EXTENTSIZE value PREFETCHSIZE value
       TEMPORARY TABLESPACE
          MANAGED BY SYSTEM USING ('path')
  WITH "comment"
```

이러한 테이블 스페이스에 대해 디폴트 정의를 사용하지 않으려면, CREATE DATABASE 명령에 특성을 지정하십시오. 예를 들어, 다음 명령을 사용하여 Windows에서 데이터베이스를 작성할 수 있습니다.

```
CREATE DATABASE PERSONL
PAGESIZE 16384
     CATALOG TABLESPACE
         MANAGED BY SYSTEM USING ('d:₩pcatalog','e:₩pcatalog')
         EXTENTSIZE 16 PREFETCHSIZE 32
     USER TABLESPACE
         MANAGED BY DATABASE USING (FILE'd:₩db2data₩personl' 5000,
                                                          FILE'd:₩db2data₩personl' 5000)
```

```
        EXTENTSIZE 32 PREFETCHSIZE 64
    TEMPORARY TABLESPACE
        MANAGED BY SYSTEM USING ('f:₩db2temp₩personl')
WITH "Personnel DB for BSchiefer Co"
```

이 예에서 디폴트 페이지 *크기*는 16,384바이트로 설정되고 각 초기 테이블 스페이스에 대한 정의가 명시적으로 제공됩니다. 디폴트 정의를 사용하지 않으려는 경우에는 테이블 스페이스에 대해 테이블 스페이스 정의를 지정하기만 하면 됩니다.

주: 파티션된 데이터베이스 환경에서 작업할 경우 컨테이너를 작성하거나 특정 데이터베이스 파티션에 지정할 수 없습니다. 먼저, 디폴트 사용자 및 임시 테이블 스페이스로 데이터베이스를 작성해야 합니다. CREATE TABLESPACE문을 사용하여 필수 테이블 스페이스를 작성할 수 있습니다. 마지막으로 디폴트 테이블 스페이스를 삭제할 수 있습니다.

CREATE DATABASE 명령의 MANAGED BY절 코드 형식은 CREATE TABLESPACE 명령의 MANAGED BY절과 동일합니다.

원하는 경우 추가 사용자와 임시 테이블 스페이스를 추가할 수 있습니다. 카탈로그 테이블 스페이스 SYSCATSPACE를 삭제하거나 다른 것을 작성할 수 없으므로, 페이지 *크기*가 4KB인 시스템 임시 테이블 스페이스가 하나 이상 있어야 합니다. 기타 시스템 임시 테이블 스페이스를 작성할 수 있습니다. 또한 테이블 스페이스를 작성한 후에 페이지 *크기* 또는 테이블 스페이스의 Extent *크기*를 변경할 수 없습니다.

DMS 직접 디스크 액세스 디바이스 접속

데이터를 저장할 컨테이너에 대해 작업할 때 데이터베이스 관리 프로그램이 직접 디스크 액세스(원시 입출력)를 지원합니다.

이러한 유형의 지원을 통해 DB2 데이터베이스 시스템에 직접 디스크 액세스(원시) 디바이스를 접속할 수 있습니다.

테이블 스페이스를 작성할 때 참조할 컨테이너의 디바이스 또는 파일 이름을 알고 있어야 합니다. 테이블 스페이스에 할당될 각 디바이스 또는 파일 이름과 연관된 스페이스의 양을 알고 있어야 합니다. 컨테이너에 데이터를 읽고 쓸 수 있는 올바른 권한이 필요합니다.

직접 디스크 액세스를 식별하는 실제적 및 논리적 방법은 운영 체제에 따라 다릅니다.

- Windows 운영 체제:

  실제 하드 드라이브를 지정하려면, 구문을 사용하십시오.

  ₩₩.₩PhysicalDriveN

  여기서, N은 시스템의 실제 드라이브 중 하나를 나타냅니다. 이 경우, N을 0, 1, 2 또는 기타 양의 정수로 바꿀 수 있습니다.

  ₩₩.₩PhysicalDrive5

논리 드라이브, 즉 포맷되지 않은 데이터베이스 파티션을 지정하려면 다음 구문을 사용하십시오.

₩₩.₩N:

여기서, N:은 시스템의 논리 드라이브 이름을 나타냅니다. 예를 들어, N:은 E: 또는 임의의 기타 드라이브 이름으로 바꿀 수 있습니다. 드라이브 식별 문자의 사용으로 인한 제한을 극복하기 위해, 논리 드라이브와 함께 GUID(Globally Unique Identifier)를 사용할 수 있습니다.

Windows에서는 새로운 방법으로 DMS 원시 테이블 스페이스 컨테이너를 지정할 수 있습니다. 볼륨(즉, 기본 디스크 데이터베이스 파티션 또는 동적 볼륨)은 작성될 때 GUID(globally unique identifier)가 지정됩니다. GUID는 테이블 스페이스 정의에 컨테이너를 지정할 때 디바이스 ID로 사용할 수 있습니다. GUID는 시스템 전반에 걸쳐 고유하며, 이는 다중 파티션 데이터베이스에서 GUID는 디스크 파티션 정의가 동일하더라도 각 데이터베이스 파티션마다 다르다는 것을 의미합니다.

*db2listvolumes.exe*라는 도구는 Windows 시스템에 정의된 모든 디스크 볼륨의 GUID를 보다 쉽게 표시하는데 사용할 수 있습니다(Windows 운영 체제만 해당). 이 도구는 이 도구가 실행되는 현재 디렉토리에 두 개의 파일을 작성합니다. `volumes.xml`이라는 파일에는 XML 사용 브라우저에서 쉽게 볼 수 있도록 XML로 인코드된 각 디스크 볼륨에 대한 정보가 들어 있습니다. `tablespace.ddl`이라는 두번째 파일에는 테이블 스페이스 컨테이너를 지정하는데 필요한 구문이 들어 있습니다. 이 파일을 갱신하여 테이블 스페이스 정의에 필요한 나머지 정보를 채워야 합니다. db2listvolumes 명령에는 명령행 인수가 필요하지 않습니다.

- Linux 및 UNIX 플랫폼에서는 논리적 볼륨이 사용자와 응용프로그램에게 단일, 연속 및 확장 가능한 디스크 볼륨으로 나타날 수 있습니다. 이러한 방법으로 나타나더라도 비연속 물리적 데이터베이스 파티션 또는 둘 이상의 물리적 볼륨에 있을 수 있습니다. 또한 논리적 볼륨은 단일 볼륨 그룹 내에 포함되어야 합니다. 볼륨 그룹당 256개의 논리적 볼륨 한계가 있습니다. 볼륨 그룹당 32개의 물리적 볼륨 한계가 있습니다. mklv 명령을 사용하여 추가 논리적 볼륨을 작성할 수 있습니다. 이 명령을 사용하여 논리적 볼륨의 이름을 지정하고, 논리적 볼륨에 할당할 논리적 파티션 수 및 위치를 비롯해 특성을 정의할 수 있습니다.

  논리적 볼륨을 작성한 후 chlv 명령을 사용하여 이름 및 특성을 변경하고 extendlv 명령을 사용하여 할당된 논리적 파티션 수를 증가시킬 수 있습니다. 작성시 논리적 볼륨의 디폴트 최대 *크기*는 더 크게 지정되지 않는 한 512개의 논리적 파티션입니다. chlv 명령은 이 제한을 겹쳐쓰는데 사용됩니다.

  AIX 내에서 논리적 볼륨 스토리지를 설정 및 제어할 수 있는 운영 체제 명령 세트, 라이브러리 서브루틴 및 기타 도구를 LVM(Logical Volume Manager)이라고 합니다. LVM은 실제 물리적 디스크와 스토리지 스페이스의 단순하고 유연한 논리적 뷰 간의 데이터를 맵핑함으로써 디스크 자원을 제어합니다.

mklv 및 기타 논리적 볼륨 명령과 LVM에 대한 자세한 정보는 *AIX 5L Version 5.2 System Management Concepts: Operating System and Devices*를 참조하십시오.

DMS 직접 디스크 액세스 구성 및 설정(Linux)

데이터를 저장할 컨테이너에 대해 작업할 때, 데이터베이스 관리 프로그램은 블록 디바이스 인터페이스(즉, 원시 입출력)를 사용한 직접 디스크(원시) 액세스를 지원합니다.

Linux에 원시 입출력을 설정하려면 하나 이상의 여유 IDE 또는 SCSI 디스크 데이터베이스 파티션이 필요합니다. 테이블 스페이스를 작성할 때 디스크 파티션을 참조하려면, 디스크 파티션의 이름과 테이블 스페이스에 할당할 디스크 파티션과 연관된 스페이스의 양을 알고 있어야 합니다.

다음 정보는 Linux 환경에서 작업할 때 사용해야 합니다. Linux/390에서, 데이터베이스 관리 프로그램은 직접 디스크 액세스 디바이스를 지원하지 않습니다.

Linux에서 원시 입출력을 구성하려면 다음을 수행하십시오.

이 예에서 사용하는 원시 데이터베이스 파티션은 /dev/sda5입니다. 이 원시 파티션에는 어떠한 가치 있는 데이터도 없어야 합니다.

1. 이 데이터베이스 파티션의 4096바이트 페이지의 수를 계산하고 필요한 경우 반올림하십시오. 예를 들어, 다음과 같습니다.

```
  # fdisk /dev/sda
  Command (m for help): p

  Disk /dev/sda: 255 heads, 63 sectors, 1106 cylinders
Units = cylinders of 16065 * 512 bytes
```

표 14. Linux 원시 입출력 계산

디바이스 시동	Start	끝	블록	ID	시스템
/dev/sda1	1	523	4200997	83	Linux
/dev/sda2	524	1106	4682947+	5	Extended
/dev/sda5	524	1106	4682947	83	Linux

```
  Command (m for help): q
#
```

/dev/sda5의 페이지 수는 다음과 같습니다.

```
   num_pages = floor( (4682947 * 1024)/4096 )
num_pages = 1170736
```

2. 디스크 파티션 이름을 지정하여 테이블 스페이스를 작성하십시오. 예를 들어,

```
CREATE TABLESPACE dms1
      MANAGED BY DATABASE
USING (DEVICE '/dev/sda5' 1170736)
```

3. 접합점(또는 볼륨 마운트 위치)을 사용하여 논리적 파티션을 지정하려면 RAW 파티션을 다른 NTFS 포맷 볼륨에 접합점으로 마운트한 후 NTFS 볼륨의 접합점에 대한 경로를 컨테이너 경로로 지정하십시오. 예를 들어, 다음과 같습니다.

```
CREATE TABLESPACE TS4
   MANAGED BY DATABASE USING (DEVICE 'C:₩JUNCTION₩DISK_1' 10000,
      DEVICE 'C:₩JUNCTION₩DISK_2' 10000)
```

   데이터베이스 관리 프로그램이 먼저 파티션을 쿼리하여 파일 시스템 R이 있는지 확인합니다. 파티션이 존재하는 경우 파티션은 원시 디바이스로서 취급되지 않으며 파티션에 정상 파일 시스템 입출력 조작을 수행합니다.

원시 디바이스의 테이블 스페이스 또한 데이터베이스 관리 프로그램에서 지원하는 다른 모든 페이지 *크기*에 대해 지원됩니다.

버전 9 이전의 경우, Linux에서 원시 제어기 유틸리티를 사용한 직접 디스크 액세스가 사용되었습니다. 이 메소드는 이제 사용되지 않으며 메소드 사용은 권장되지 않습니다. Linux 운영 체제가 메소드를 지원하는 경우 데이터베이스 관리 프로그램에서 여전히 이 메소드를 사용할 수 있지만, db2diag 로그 파일에 해당 메소드의 사용이 지원되지 않음을 나타내는 메시지가 기록됩니다.

이전 메소드에서는 디스크 파티션을 원시 제어기에 "바인드"한 후 CREATE TABLESPACE 명령을 사용하여 해당 제어기를 데이터베이스 관리 프로그램에 지정해야 합니다.

```
CREATE TABLESPACE dms1
      MANAGED BY DATABASE
   USING (DEVICE '/dev/raw/raw1' 1170736)
```

테이블 스페이스 변경

명령행을 사용하여 테이블 스페이스를 변경하려면 ALTER TABLESPACE문을 사용하십시오.

테이블 스페이스의 유형에 따라서 다음과 같은 작업을 수행할 수 있습니다.

- 추가 컨테이너를 추가하여 테이블 스페이스의 *크기* 늘리기
- 기존 컨테이너 *크기* 조정
- 컨테이너 삭제
- 새 컨테이너를 사용하기 시작하거나 삭제되는 컨테이너 밖으로 데이터를 이동하기 위한 테이블 스페이스 재조정
- 테이블 스페이스의 상위 워터 마크(water mark) 낮추기
- 테이블 스페이스의 전체 *크기* 축소

또한 테이블 스페이스 이름을 바꾸고 오프라인에서 온라인 모드로 전환할 수 있습니다.

테이블 스페이스 사용량 계산

MON_GET_TABLESPACE 테이블 함수를 사용하여 테이블 스페이스에서 현재 사용 중인 양을 판별할 수 있습니다. 이 함수가 리턴하는 정보는 여유 스토리지를 재개해야 할지 여부를 판별하는 데 도움이 됩니다.

이 태스크에 대한 정보

이 태스크는 테이블 스페이스에 대한 상위 워터 마크(water mark) 아래에 사용되지 않는 스페이스가 있는 Extent를 판별하는 데 사용할 수 있는 정보를 제공합니다. 이를 기초로 여유 스토리지 재개가 도움이 될지 여부를 판별할 수 있습니다.

제한사항

모든 테이블 스페이스에 대한 다양한 사용량 속성을 판별할 수 있지만, DB2 버전 9.7 이상으로 작성된 테이블 스페이스만 재개 가능 스토리지 기능을 갖습니다. DB2 제품의 이전 버전을 사용하여 작성된 테이블 스페이스에서 스토리지를 재개할 수 있기 원하는 경우, 데이터를 언로드한 후 DB2 버전 9.7을 사용하여 작성된 테이블 스페이스에 다시 로드하거나 온라인 이동을 사용하여 데이터를 이동해야 합니다.

프로시저

상위 워터 마크(water mark) 아래에 여유 공간이 얼마나 존재하는지 판별하려면 다음을 수행하십시오.

1. MON_GET_TABLESPACE 테이블 함수를 통합하여 테이블 스페이스의 상태에 대해 보고하는 SELECT문을 공식화하십시오. 예를 들어 다음 명령문은 모든 데이터베이스 파티션 사이에서 모든 테이블 스페이스에 대한 총 페이지, 사용 가능한 페이지, 사용된 페이지를 표시합니다.

   ```
   SELECT varchar(tbsp_name, 30) as tbsp_name,
     reclaimable_space_enabled,
     tbsp_free_pages,
     tbsp_page_top,
     tbsp_usable_pages
   FROM TABLE(MON_GET_TABLESPACE('',-2)) AS t
   ORDER BY tbsp_free_pages ASC
   ```

2. 명령문을 실행하십시오. 다음과 비슷한 출력이 표시됩니다.

```
TBSP_NAME                      RECLAIMABLE_SPACE_ENABLED TBSP_FREE_PAGES      TBSP_PAGE_TOP        TBSP_USABLE_PAGES
------------------------------ ------------------------- ------------------- -------------------- --------------------
TEMPSPACE1                                             0                   0                    0                    1
SYSTOOLSTMPSPACE                                       0                   0                    0                    1
TBSP1                                                  1                   0                 1632                 1632
SMSDEMO                                                0                   0                    0                    1
SYSCATSPACE                                            1                2012                10272                12284
USERSPACE1                                             1                2496                 1696                 4064
IBMDB2SAMPLEREL                                        1                3328                  736                 4064
TS1                                                    1                3584                  480                 4064
TS2                                                    1                3968                   96                 4064
TBSP2                                                  1                3968                   96                 4064
```

```
TBSAUTO                                1                3968                96                4064
SYSTOOLSPACE                           1                3976               116                4092

  12 레코드가 선택되었습니다.
```

3. 다음 공식을 사용하여 상위 워터 마크(water mark) 아래에 있는 사용 가능한 페이지 수를 판별하십시오.

   freeSpaceBelowHWM = tbsp_free_pages - (tbsp_usable_pages - tbsp_page_top)

결과

202 페이지의 2 단계의 보고서에 있는 정보를 사용할 때, USERSPACE1에 대해 상위 워터 마크(water mark) 아래의 여유 공간은 2496 - (4064 - 1696) = 128페이지입니다. 이것은 테이블 스페이스에서 사용할 수 있는 총 사용 가능한 페이지의 5%를 약간 넘는 수준입니다.

다음 단계

이 경우에는 이 스페이스를 재개하려고 시도할 필요가 없습니다. 그러나 128페이지를 재개하려는 경우 ALTER TABLESPACE USERSPACE1 REDUCE MAX문을 실행할 수 있습니다. 명령문을 실행하고 MON_GET_TABLESPACE 테이블 함수를 다시 실행하면 다음이 표시됩니다.

```
TBSP_NAME                      RECLAIMABLE_SPACE_ENABLED TBSP_FREE_PAGES      TBSP_PAGE_TOP        TBSP_USABLE_PAGES
------------------------------ ------------------------- -------------------- -------------------- --------------------
TEMPSPACE1                                             0                    0                    0                    1
USERSPACE1                                             1                    0                 1568                 1568
SYSTOOLSTMPSPACE                                       0                    0                    0                    1
TBSP1                                                  1                    0                 1632                 1632
SMSDEMO                                                0                    0                    0                    1
SYSCATSPACE                                            1                 2012                10272                12284
IBMDB2SAMPLEREL                                        1                 3328                  736                 4064
TS1                                                    1                 3584                  480                 4064
TS2                                                    1                 3968                   96                 4064
TBSP2                                                  1                 3968                   96                 4064
TBSAUTO                                                1                 3968                   96                 4064
SYSTOOLSPACE                                           1                 3976                  116                 4092

  12 레코드가 선택되었습니다.
```

SMS 테이블 스페이스 변경

SMS 테이블 스페이스가 작성된 후에는 SMS 테이블 스페이스에 컨테이너를 추가하거나 컨테이너 크기를 변경할 수 없습니다. 한 가지 예외가 있는데, 새 데이터 파티션을 추가할 때 해당 파티션에 대한 SMS 테이블 스페이스에 새 컨테이너를 추가할 수 있습니다.

DMS 테이블 스페이스 변경

DMS 테이블 스페이스의 경우 컨테이너를 추가, 확장, 재조정, 크기 조정, 삭제 또는 축소할 수 있습니다.

DMS 컨테이너 추가

테이블 스페이스에 하나 이상의 컨테이너를 추가하여 DMS 테이블 스페이스(MANAGED BY DATABASE절로 작성된 테이블 스페이스)의 *크기*를 증가시킬 수 있습니다.

새 컨테이너를 추가하고 새 stripe 세트를 작성하면 균형 조정이 발생하지 않습니다. 새 stripe 세트는 ALTER TABLESPACE문에 BEGIN NEW STRIPE SET절을 사용하여 작성합니다. ALTER TABLESPACE문에서 ADD TO STRIPE SET절을 사용하여 기존 stripe 세트에 컨테이너를 추가할 수도 있습니다.

DMS 컨테이너(파일 및 원시 디바이스 컨테이너 둘 다)의 추가 또는 수정은 프리페처를 통해 병렬로 수행됩니다. 이러한 컨테이너 작성 또는 *크기* 조정 조작의 병렬 처리에서 증가를 하려면, 시스템에서 수행 중인 프리페처의 수를 증가시킵니다. 병렬에서 완료되지 않는 단 하나의 프로세스는 이 조치의 로깅과 컨테이너 작성의 경우, 컨테이너의 태그 처리입니다.

주: CREATE TABLESPACE 또는 ALTER TABLESPACE문(기존 테이블 스페이스에 새 컨테이너 추가에 관하여)의 병렬 처리를 최대화하려면, 프리페처의 수가 추가된 컨테이너의 수보다 더 많은지 또는 같은지 확인하십시오. 프리페처 수는 *num_ioservers* 데이터베이스 구성 매개변수로 제어됩니다. 새 매개변수 값을 적용하려면 데이터베이스를 중지해야 합니다. 즉, 변경사항을 적용하려면 데이터베이스에서 모든 응용프로그램 및 사용자 연결을 끊어야 합니다.

다음 예는 Linux 및 UNIX 시스템에서 두 개의 새 디바이스 컨테이너(각각 10,000페이지 포함)를 테이블 스페이스에 추가하는 방법을 나타냅니다.

```
ALTER TABLESPACE RESOURCE
        ADD (DEVICE '/dev/rhd9'  10000,
       DEVICE  '/dev/rhd10' 10000)
```

ALTER TABLESPACE문을 사용하여, 성능에 영향을 줄 수 있는 테이블 스페이스의 기타 등록 정보를 변경할 수 있습니다.

DMS 컨테이너 삭제

DMS 테이블 스페이스를 사용하면 ALTER TABLESPACE문을 사용하여 테이블 스페이스에서 컨테이너를 삭제(drop)할 수 있습니다.

컨테이너 삭제는 조작에 의해 삭제될 Extent의 수가 테이블 스페이스에서 상위 워터 마크(water mark) 위의 여유 Extent 수보다 작거나 같은 경우에만 허용됩니다. 이 조작을 통해 페이지 수를 변경할 수 없기 때문에 이러한 조작이 필요합니다. 따라서 최고 수위 표시까지의 모든 Extent가 테이블 스페이스 내에서 동일한 논리적 위치에 있어야 합니다. 따라서 결과 테이블 스페이스에는 최대 수위 표시까지의 모든 데이터를 보유할 충분한 스페이스가 있어야 합니다. 충분한 여유 공간이 없는 상황에서는 명령문이 실행되면 오류가 발생합니다.

컨테이너가 삭제되면 나머지 컨테이너는 컨테이너 ID가 0부터 시작하여 1씩 증가하도록 번호가 다시 지정됩니다. 스트라이프 세트의 모든 컨테이너가 삭제되는 경우, 스트라이프 세트가 맵에서 제거되며 맵에서 그 뒤의 모든 스트라이프 세트는 아래로 이동되고 누락된 스트라이프 세트 번호가 생기지 않도록 번호가 다시 지정됩니다.

컨테이너를 삭제하려면 ALTER TABLESPACE문에 DROP 옵션을 사용하십시오.

DMS 컨테이너 크기 조정

스토리지를 변경해야 할 때 데이터베이스 관리(DMS) 테이블 스페이스의 컨테이너를 크기 조정할 수 있습니다. DMS 컨테이너에 대해 자동 크기 조정 성능을 사용하는 경우 데이터베이스 관리 프로그램이 이를 대신 수행합니다. 자동 크기 조정 옵션을 사용하지 않은 경우 수동으로 조정할 수도 있습니다.

DMS 테이블 스페이스에 있는 하나 이상의 컨테이너의 크기를 지정된 양만큼 늘리려면, ALTER TABLESPACE 명령의 EXTEND 옵션을 사용하십시오. 기존 컨테이너의 크기를 줄이려면 REDUCE 옵션을 사용하십시오. EXTEND 또는 REDUCE를 사용할 때 현재 크기에서 늘리거나 줄이려는 크기로 원하는 만큼의 양을 지정합니다. 달리 말하면 크기는 현재 크기에 상대적으로 조정됩니다.

또한 ALTER TABLESPACE문에서 RESIZE 옵션을 사용할 수도 있습니다. RESIZE를 사용할 때 영향을 받는 컨테이너의 새 크기를 지정하십시오. 달리 말하면, 크기는 지정된 컨테이너에 대한 절대 크기로 해석됩니다. RESIZE 옵션을 사용하면 명령문의 일부로서 나열한 모든 컨테이너의 크기가 증가하거나 감소됩니다. 동일한 명령문을 사용하여 일부 컨테이너의 크기를 증가시키면서 다른 컨테이너의 크기를 감소시킬 수 없습니다.

DMS 컨테이너(파일 및 원시 디바이스 컨테이너 둘 다)의 추가 또는 수정은 프리페처를 통해 병렬로 수행됩니다. 이러한 컨테이너 작성 또는 크기 조정 조작의 병렬 처리에서 증가를 하려면, 시스템에서 수행 중인 프리페처의 수를 증가시킵니다. 병렬에서 완료되지 않는 단 하나의 프로세스는 이 조치의 로깅과 컨테이너 작성의 경우, 컨테이너의 태그 처리입니다.

주: CREATE TABLESPACE 또는 ALTER TABLESPACE문(기존 테이블 스페이스에 새 컨테이너 추가에 관하여)의 병렬 처리를 최대화하려면, 프리페처의 수가 추가된 컨테이너의 수보다 더 많은지 또는 같은지 확인하십시오. 프리페처 수는 *num_ioservers* 데이터베이스 구성 매개변수로 제어됩니다. 새 매개변수 값을 적용하려면 데이터베이스를 중지해야 합니다. 즉, 변경사항을 적용하려면 데이터베이스에서 모든 응용프로그램 및 사용자 연결을 끊어야 합니다.

제한사항

각 원시 디바이스는 하나의 컨테이너로서 사용될 수 있습니다. 원시 디바이스(raw device) 크기는 작성 후에 고정됩니다. RESIZE 또는 EXTEND 옵션을 사용하여 원시 디바이

스 컨테이너를 확장하고자 할 경우에는, 먼저 원시 디바이스 *크기*를 점검하여 디바이스 컨테이너 *크기*를 원시 디바이스 *크기*보다 더 크게 확장하지 않도록 해야 합니다.

예 1: 파일 컨테이너의 크기 늘리기. 다음 예는 Windows 기반 시스템의 테이블 스페이스에서 파일 컨테이너(각각 이미 1000페이지가 있는)를 늘리는 방법을 설명합니다.

```
 ALTER TABLESPACE PERSNEL
EXTEND (FILE 'e:₩wrkhist1'  200
        FILE  'f:₩wrkhist2'  200)
```

이 조치 후에, 두 파일은 *크기*가 1000페이지에서 1200페이지로 늘어났습니다. 컨테이너 전체에서 테이블 스페이스의 컨텐츠가 재조정될 수도 있습니다. 재조정하는 동안에도 테이블 스페이스에 대한 액세스는 제한되지 않습니다.

예 2: 디바이스 컨테이너의 크기 늘리기. 다음 예는 Linux 및 UNIX 시스템의 테이블 스페이스에서 두 디바이스 컨테이너(각각 이미 1000페이지를 갖는)를 늘리는 방법을 나타냅니다.

```
  ALTER TABLESPACE HISTORY
      RESIZE (DEVICE '/dev/rhd7'  2000,
        DEVICE  '/dev/rhd8' 2000)
```

이 조치 후에, 두 디바이스는 *크기*가 1,000페이지에서 2,000페이지로 증가했습니다. 컨테이너 전체에서 테이블 스페이스의 컨텐츠가 재조정될 수도 있습니다. 재조정하는 동안에도 테이블 스페이스에 대한 액세스는 제한되지 않습니다.

예 3: REDUCE 옵션을 사용하여 컨테이너 크기 축소 다음 예는 Windows 기반 시스템의 테이블 스페이스에서 파일 컨테이너(이미 1000페이지를 갖는)를 줄이는 방법을 나타냅니다.

```
  ALTER TABLESPACE PAYROLL
REDUCE (FILE 'd:₩hldr₩finance'  200)
```

이 조치 후에 파일 *크기*는 1,000페이지에서 800페이지로 줄어듭니다.

DMS 컨테이너 재조정

재조정 프로세스에는 테이블 스페이스 Extent를 한 위치에서 다른 위치로 이동하는 작업이 포함되며, 이 프로세스는 테이블 스페이스에서 데이터를 스트립된 상태로 유지하기 위해 수행됩니다. 일반적으로 데이터베이스에 스토리지 경로를 추가하거나 스토리지 경로를 삭제할 때 테이블 스페이스를 재조정합니다.

재조정 시 컨테이너 추가 또는 삭제 효과

테이블 스페이스가 작성될 때 테이블 스페이스 맵이 작성되며, 초기 컨테이너는 모두 스트라이프 0에서 시작되도록 정렬됩니다. 이는 개별 컨테이너가 가득 찰 때까지 데이터가 모든 테이블 스페이스 컨테이너에 균등하게 스트라이프됨을 의미합니다 (예 1 (『이전』)을 참조하십시오.)

기존의 컨테이너보다 작은 컨테이너를 추가하면 데이터가 균등하게 배포되지 않습니다. 이렇게 되면 데이터 프리페치 데이터와 같은 병렬 입출력 조작이 같은 크기의 컨테이너에서 실행하는 것보다 비효율적으로 실행됩니다.

새 컨테이너가 테이블 스페이스에 추가되거나 기존 컨테이너가 확장될 때, 새 스페이스가 테이블 스페이스에 대한 *상위 워터 마크(water mark)* 아래에 추가되는 경우 테이블 스페이스 데이터의 재조정이 발생합니다. 새 스페이스가 상위 워터 마크(water mark) 위에 추가되는 경우 또는 새 스트라이프 세트를 작성 중인 경우 재조정이 자동으로 발생하지 않습니다. 추가된 스토리지를 활용하기 위해 수행되는 재조정을 *포워드 재조정*이라고 합니다. 이 경우에 Extent 이동은 Extent 0(테이블 스페이스의 첫 번째 Extent)에서 시작하며 상위 워터 마크(water mark) 바로 아래에 있는 Extent쪽으로 진행됩니다.

컨테이너 추가는 거의 항상 상위 워터 마크(water mark) 아래에 스페이스를 추가하며, 이것이 컨테이너를 추가할 때 자주 재조정이 필요한 이유입니다. 새 컨테이너가 상위 워터 마크(water mark) 위쪽에 추가되도록 강제할 수 있는데, 이를 사용하면 테이블 스페이스의 내용이 재조정되지 않습니다. 이 방법의 이점은 새 컨테이너를 즉시 사용할 수 있다는 점입니다. 재조정없이 컨테이너를 테이블 스페이스에 추가하려면 새 *스트라이프 세트*를 추가하십시오. 스트라이프 세트는 해당 테이블 스페이스에 속한 다른 컨테이너와는 별도로 데이터가 스트라이프되는 테이블 스페이스의 컨테이너 세트입니다. 기존 스트라이프 세트의 기존 컨테이너는 그대로 남아 있으며 추가하는 컨테이너는 새 스트라이프 세트의 일부가 됩니다. 재조정없이 컨테이너를 추가하려면 ALTER TABLESPACE 문에서 BEGIN NEW STRIPE SET절을 사용하십시오.

컨테이너가 테이블 스페이스에서 삭제될 때 삭제되는 스페이스에 데이터가 있으면 재조정이 자동으로 발생합니다. 이 경우에 재조정을 *역방향 재조정*이라고 합니다. Extent 이동은 상위 워터 마크(water mark)에서 시작하며 테이블 스페이스의 첫 번째 Extent를 향해 아래쪽으로 진행됩니다.

재조정이 시작되기 전에 컨테이너 변경사항에 따라 새 테이블 스페이스 맵이 빌드됩니다. 재조정 프로그램은 현재 맵에 의해 판별된 위치에서 새 맵에 의해 판별된 위치로 Extent를 이동시킵니다.

포워드 재조정

재조정 프로그램은 상위 워터 마크(water mark)가 있는 Extent가 이동될 때까지 Extent 0부터 시작하여 한 번에 한 Extent씩 이동시킵니다. 각 Extent가 이동됨에 따라 현재 맵이 새 맵과 같이 표시되도록 한 번에 하나씩 변경됩니다. 재조정이 완료되면 현재 맵과 새 맵은 상위 워터 마크(water mark)가 있는 스트라이프까지 모양이 같습니다. 그러면 현재 맵은 새 맵과 완전히 모양이 같게 되며 재조정 프로세스가 완료됩니다. 현재 맵의 Extent 위치가 새 맵에서의 위치와 동일한 경우, 이 Extent는 이동되지 않으므로 입출력 조작이 발생하지 않습니다.

새 컨테이너를 추가할 때, 새 맵에서의 해당 컨테이너 위치는 해당 컨테이너의 *크기*와 스트라이프 세트에 있는 다른 컨테이너의 *크기*에 따라 달라집니다. 컨테이너가 스트라이프 세트의 첫 번째 스트라이프에서 시작할 수 있고 스트라이프 세트의 마지막 스트라이프(또는 그 아래)에서 끝날 수 있도록 충분히 큰 경우, 그와 같은 방식으로 배치됩니다(예 1(이후) 참조). 컨테이너가 이를 수행할 만큼 충분히 *크지* 않은 경우, 스트라이프 세트의 마지막 스트라이프에서 끝나도록 맵에서 위치됩니다(예 3 참조). 이는 재조정해야 할 데이터의 양을 최소화하기 위해서입니다.

재조정하는 동안에는 테이블 스페이스에 대한 액세스가 제한되지 않습니다. 오브젝트의 삭제, 작성, 채우기 및 쿼리가 평상시와 같이 가능합니다. 그러나 재조정 조작은 성능에 상당한 영향을 줄 수 있습니다. 둘 이상의 컨테이너를 추가해야 하며 컨테이너 재조정하려고 계획한 경우, 단일 ALTER TABLESPACE문 내에 컨테이너를 동시에 추가하여 데이터베이스 관리 프로그램이 데이터를 두 번 이상 재조정할 필요가 없도록 해야 합니다.

주: 다음 예에서 컨테이너 *크기*는 컨테이너 태그의 *크기*를 고려하지 않습니다. 컨테이너 *크기*는 매우 작으며 설명을 위해 사용된 것일 뿐 권장되는 컨테이너 *크기*가 아닙니다. 이 예에서는 테이블 스페이스 내의 *크기*가 서로 다른 컨테이너를 보여주나, 동일한 컨테이너를 사용하는 것이 좋습니다.

역방향 재조정

재조정 프로그램은 상위 워터 마크(water mark)가 있는 Extent부터 시작하여 한 번에 한 Extent씩 모두 이동시킵니다. 각 Extent가 이동되면 현재 맵은 새 맵과 모양이 비슷해지도록 한 번에 한 조각씩 변경됩니다. 현재 맵의 Extent 위치가 새 맵에서의 위치와 동일한 경우, 이 Extent는 이동되지 않으므로 입출력 조작이 발생하지 않습니다.

예

예 1(이전): 컨테이너가 추가되기 전의 테이블 스페이스 레이아웃

세 개의 컨테이너가 있고 Extent *크기*가 10이며 컨테이너가 각각 60, 40 및 80 페이지(6, 4 및 8개의 Extent)인 테이블 스페이스를 작성할 경우, 이 테이블 스페이스는 209 페이지의 그림 19에 표시된 것과 같이 다이어그램이 가능한 맵으로 작성됩니다.

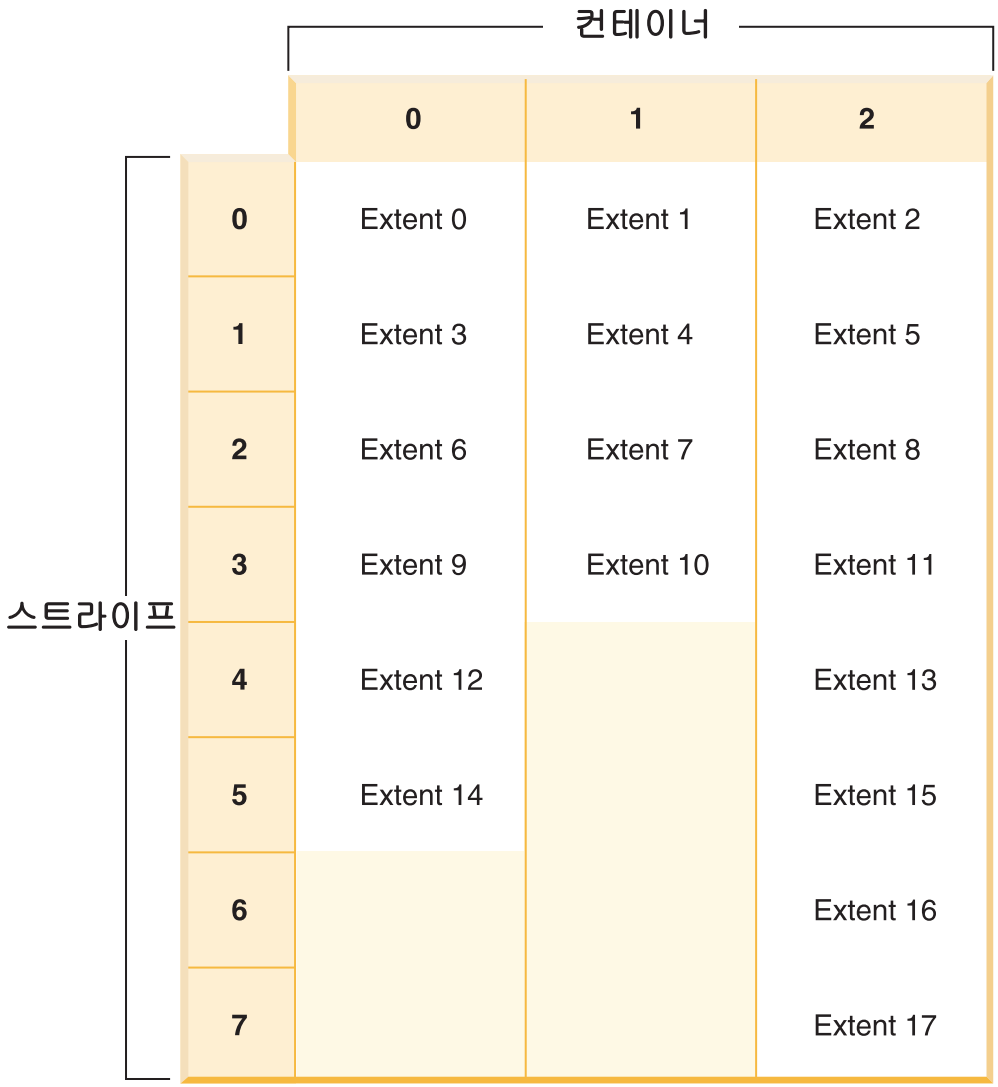


그림 19. 3개의 컨테이너와 18개의 Extent가 있는 테이블 스페이스

테이블 스페이스 스냅샷에 표시된 바와 같이 해당 테이블 스페이스 맵은 다음과 같습니다.

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number Set    Offset  Extent      Page Stripe Stripe
[0]      [0]          0         11       119       0           3       0     3 (0, 1, 2)
[1]      [0]          0         15       159       4           5       0     2 (0, 2)
[2]      [0]          0         17       179       6           7       0     1 (2)
```

테이블 스페이스 맵의 표제는 범위 번호, 스트라이프 세트, 스트라이프 오프셋, 범위로 지정된 최대 Extent 번호, 범위로 지정된 최대 페이지 번호, 시작 스트라이프, 끝 스트라이프, 범위 조정 및 컨테이너 목록입니다.

예 1(이후): 포워드 재조정이 수행되도록 컨테이너 추가

예 1의 테이블 스페이스에 80페이지 컨테이너를 추가할 경우, 컨테이너는 첫 번째 스트라이프(스트라이프 0)에서 시작하고 마지막 스트라이프(스트라이프 7)에서 끝날 수 있

을 만큼 충분히 큽니다. 이 컨테이너는 첫 번째 스트라이프에서 시작하도록 배치됩니다. 결과 테이블 스페이스는 그림 20와 같은 다이어그램으로 나타낼 수 있습니다.

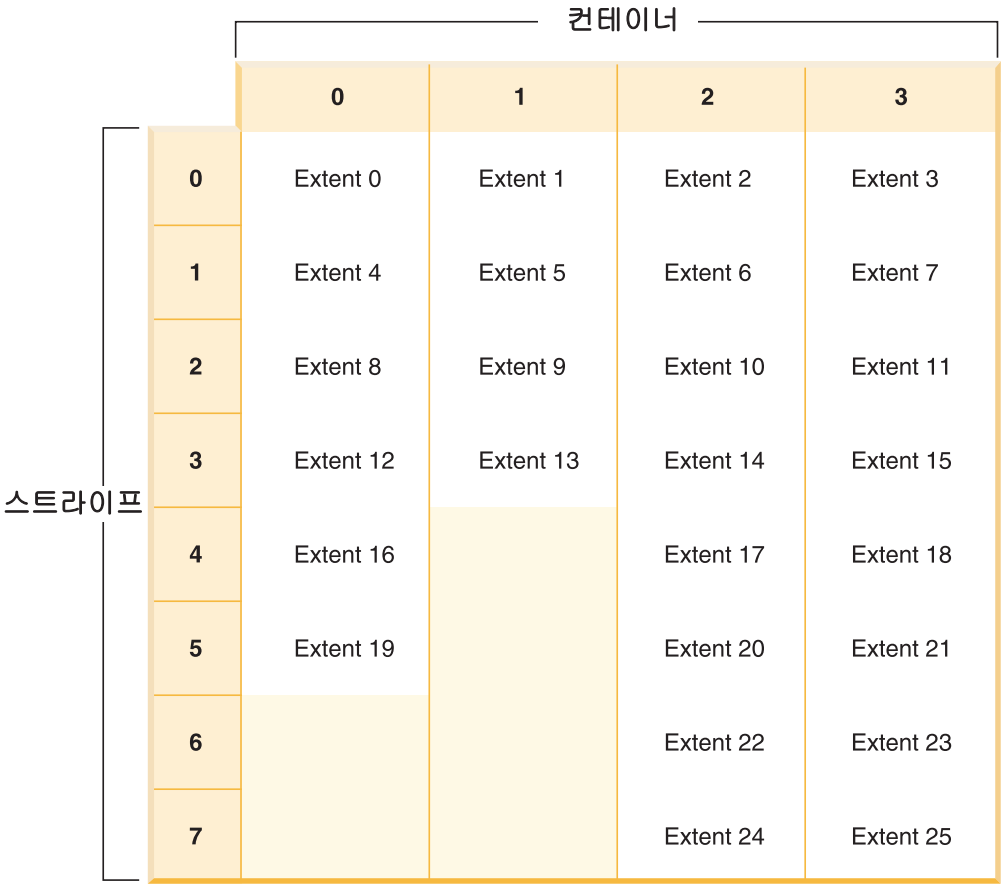


그림 20. 4개의 컨테이너와 26개의 Extent가 있는 테이블 스페이스

테이블 스페이스 스냅샷에 표시된 대로 해당 테이블 스페이스 맵의 모양은 다음과 같습니다.

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
   Number Set     Offset  Extent      Page Stripe Stripe
       [0]       [0]          0          15        159       0           3          0     4 (0, 1, 2, 3)
       [1]       [0]          0          21        219       4           5          0     3 (0, 2, 3)
       [2]       [0]          0          25        259       6           7          0     2 (2, 3)
```

상위 워터 마크(water mark)가 Extent 14 이내인 경우, 재조정 프로그램은 0 Extent에서 시작해 14까지의 모든 Extent를 이동시킵니다. 두 맵에서 Extent 0의 위치는 동일하기 때문에 이 Extent를 이동시킬 필요가 없습니다. Extent 1 및 2의 경우도 마찬가지입니다. Extent 3은 이동시킬 필요가 없으므로 Extent를 이전 위치(0 컨테이너에서 두 번째 Extent)에서 읽고 새 위치(컨테이너 3에서 첫 번째 Extent)에 기록합니다. 이 위치 이후부터 Extent 14까지의 모든 Extent가 이동됩니다. Extent 14가 이동되면 현재 맵은 새 맵과 모양이 같게 되고 재조정 프로그램이 종료됩니다.

새로 추가된 모든 스페이스가 상위 워터 마크(water mark) 뒤에 오도록 맵을 변경하는 경우, 재조정은 필요하지 않으므로 모든 스페이스를 즉시 사용할 수 있습니다. 스페이스의 일부가 상위 워터 마크(water mark) 뒤에 오도록 맵을 변경할 경우, 상위 워터 마크(water mark) 위의 스트라이프에 있는 스페이스를 사용할 수 있습니다. 그러나 나머지는 재조정이 완료될 때까지 사용할 수 없습니다.

컨테이너를 확장할 경우에도 재조정 프로그램은 비슷한 방법으로 작동합니다. 컨테이너가 스트라이프 세트의 마지막 스트라이프 이상으로 확장되는 경우, 스트라이프 세트는 이를 수용하도록 확장되며 다음의 스트라이프 세트는 이에 따라 이동됩니다. 결과적으로 컨테이너는 그 뒤의 스트라이프 세트로 확장되지 않습니다.

예 2: 컨테이너 확장

예제 1의 테이블 스페이스를 생각해 보십시오. 컨테이너 1을 40페이지에서 80페이지로 확장할 경우, 새 테이블 스페이스는 212 페이지의 그림 21과 같이 됩니다.

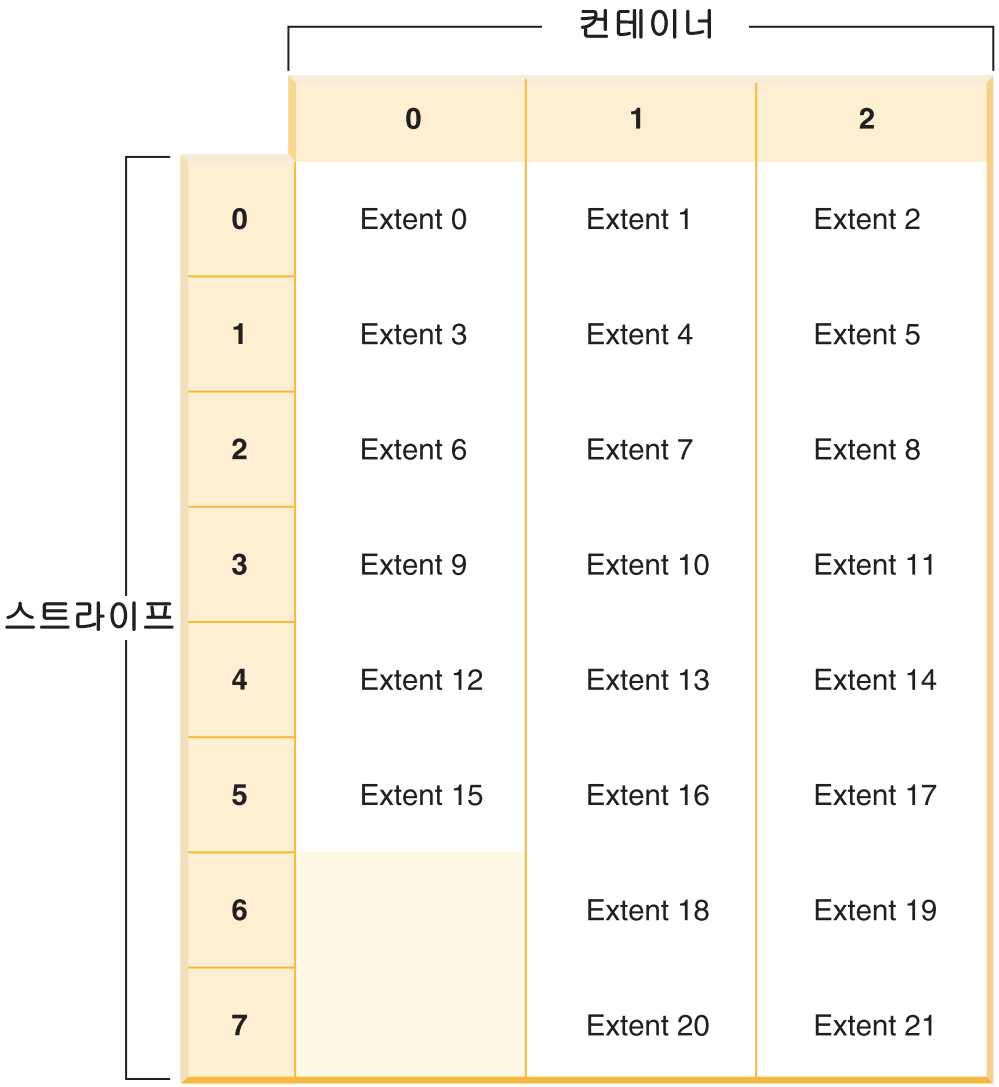


그림 21. 3개의 컨테이너와 22개의 Extent가 있는 테이블 스페이스

테이블 스페이스 스냅샷에 표시된 바와 같이 해당 테이블 스페이스 맵은 다음과 같습니다.

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
    Number Set    Offset  Extent      Page Stripe Stripe
       [0]     [0]          0         17         179      0           5          0    3 (0, 1, 2)
       [1]     [0]          0         21         219      6           7          0    2 (1, 2)
```

예 3: 첫 번째 스트라이프에서 시작하고 마지막에서 끝나기에 충분히 크지 않은 컨테이너 추가

예 1의 테이블 스페이스를 고려하십시오. 50페이지(5개의 Extent) 컨테이너를 추가할 경우, 이 컨테이너는 다음과 같이 새 맵에 추가됩니다. 컨테이너가 첫 번째 스트라이프(스트라이프 0)에서 시작하고 마지막 스트라이프(스트라이프 7)에서 끝날 수 있을 만큼 충분히 크지 않으므로, 이 컨테이너는 마지막 스트라이프에서 끝나도록 배치됩니다 (213 페이지의 그림 22 참조).

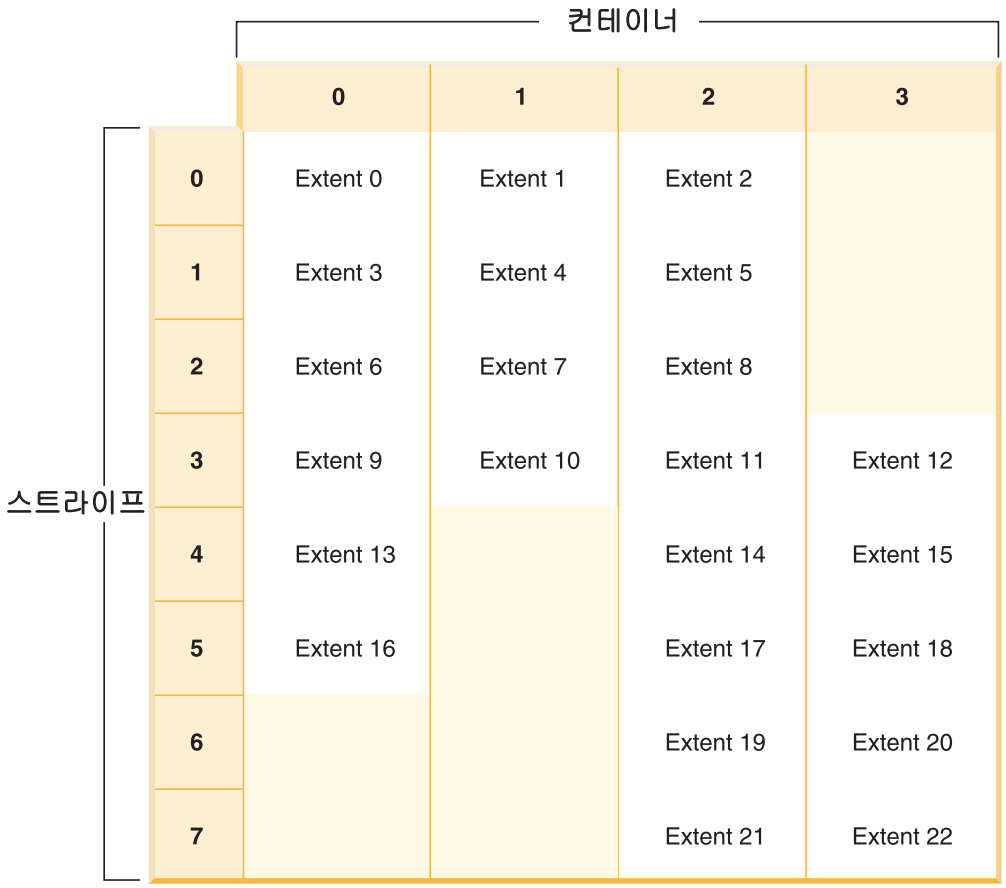


그림 22. 4개의 컨테이너와 23개의 Extent가 있는 테이블 스페이스

테이블 스페이스 스냅샷에 표시된 대로 해당 테이블 스페이스 맵의 모양은 다음과 같습니다.

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
    Number Set    Offset  Extent      Page Stripe Stripe
        [0]     [0]          0         8          89       0           2          0    3 (0, 1, 2)
        [1]     [0]          0        12         129       3           3          0    4 (0, 1, 2, 3)
        [2]     [0]          0        18         189       4           5          0    3 (0, 2, 3)
        [3]     [0]          0        22         229       6           7          0    2 (2, 3)
```

컨테이너를 확장하려면 ALTER TABLESPACE문에서 EXTEND 또는 RESIZE절을 사용하십시오. 컨테이너를 추가하고 데이터를 재조정하려면 ALTER TABLESPACE문에 ADD 옵션을 사용하십시오. 이미 둘 이상의 스트라이프 세트가 있는 테이블 스페이스에 컨테이너를 추가하는 경우, 추가하려는 스트라이프 세트를 지정할 수 있습니다. 이를 지정하려면 ALTER TABLESPACE문에 ADD TO STRIPE SET절을 사용하십시오. 스트라이프 세트를 지정하지 않은 경우, 디폴트로 현재 스트라이프 세트에 컨테이너가 추가됩니다. 현재 스트라이프 세트는 마지막으로 스페이스가 추가된 세트가 아닌 최근에 작성된 스트라이프 세트입니다.

스트라이프 세트를 변경하면 해당 스트라이프 세트와 그 뒤의 다른 스트라이프 세트에 대한 재조정이 이뤄질 수 있습니다.

테이블 스페이스 스냅샷을 사용하여 재조정 진행 상태를 모니터할 수 있습니다. 테이블 스페이스 스냅샷은 재조정의 시작 시간, 이동된 Extent 수 및 이동시켜야 하는 Extent 수와 같이 재조정에 대한 정보를 제공할 수 있습니다.

예 4: 역방향 재조정이 수행되게 하는 컨테이너 삭제

주: 다음 예에서 컨테이너 *크기*는 컨테이너 태그의 *크기*를 고려하지 않습니다. 컨테이너 *크기*는 매우 작으며 설명을 위해 사용된 것일 뿐 권장되는 컨테이너 *크기*가 아닙니다. 이 예에서는 테이블 스페이스 내에 있는 여러 가지 *크기*의 컨테이너를 나타내지만 설명하기 위한 것일 뿐입니다. 동일한 *크기*의 컨테이너를 사용하는 것이 바람직합니다.

예를 들어, 3개의 컨테이너가 있으며 Extent *크기*가 10인 테이블 스페이스를 고려해 보십시오. 컨테이너는 차례대로 20, 50 및 50페이지(2, 5 및 5개의 Extent)입니다. 이 테이블 스페이스에 대한 다이어그램은 그림 23과 같습니다.

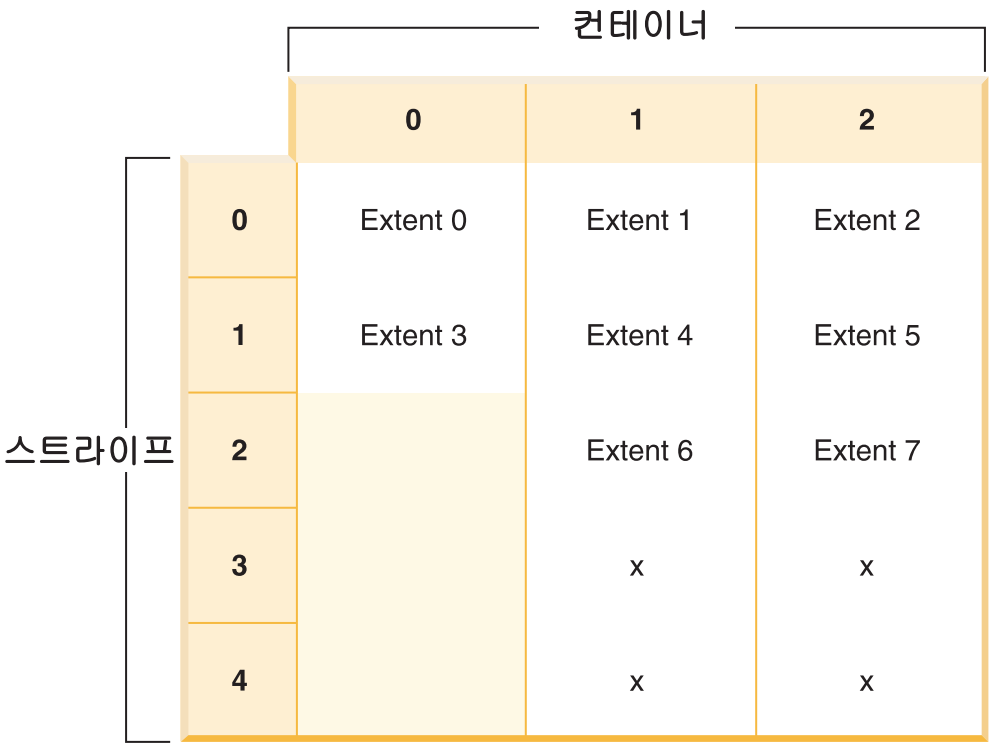


그림 23. 데이터가 없는 4개의 Extent를 포함하여 12개의 Extent가 있는 테이블 스페이스

X는 Extent는 있지만 그 안에 데이터가 없음을 나타냅니다.

두 개의 Extent가 있는 컨테이너 0을 삭제하려면 상위 워터 마크(water mark) 위에 여유 Extent가 최소한 두 개는 있어야 합니다. 상위 워터 마크(water mark)는 Extent 7에 있으며 4개의 여유 Extent가 남아 있으므로 컨테이너 0을 삭제할 수 있습니다.

테이블 스페이스 스냅샷에 표시된 대로 해당 테이블 스페이스 맵의 모양은 다음과 같습니다.

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number     Set    Offset  Extent     Page   Stripe   Stripe
[0]       [0]       0         5       59     0        1      0      3 (0, 1, 2)
[1]       [0]       0        11      119     2        4      0      2 (1, 2)
```

삭제 후 테이블 스페이스는 컨테이너 0 및 컨테이너 1만 포함합니다. 새 테이블 스페이스에 대한 다이어그램은 그림 24와 같습니다.

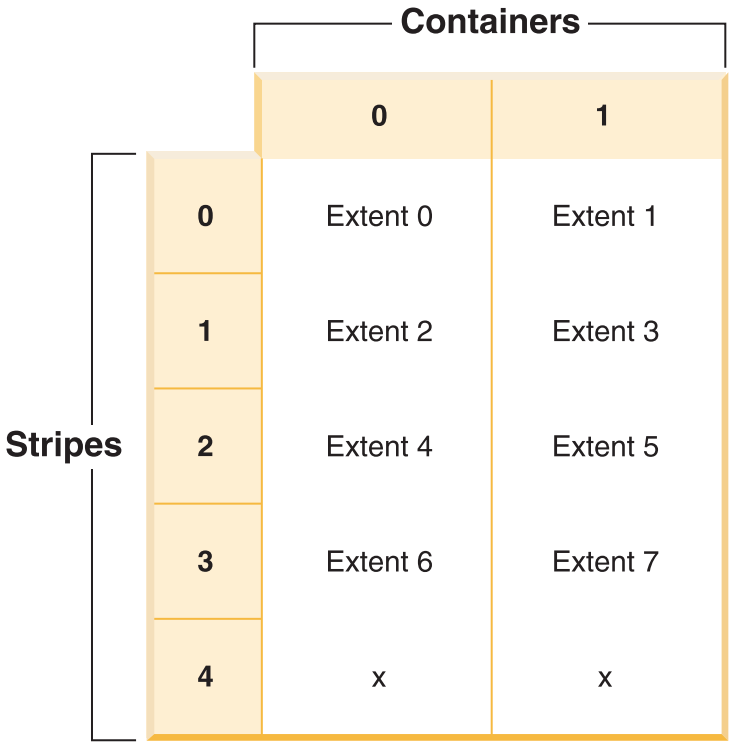


그림 24. 컨테이너 삭제 후의 테이블 스페이스

테이블 스페이스 스냅샷에 표시된 대로 해당 테이블 스페이스 맵의 모양은 다음과 같습니다.

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number     Set    Offset  Extent     Page   Stripe   Stripe
[0]       [0]       0         9       99     0        4      0      2 (0, 1)
```

예 5: 새 스트라이프 세트 추가

세 개의 컨테이너가 있고 Extent 크기가 10이며 컨테이너가 각각 30, 40 및 40페이지(3, 4 및 4개의 Extent)인 테이블 스페이스가 있는 경우, 이 테이블 스페이스는 216 페이지의

페이지의 그림 25에서와 같은 다이어그램으로 나타낼 수 있습니다.

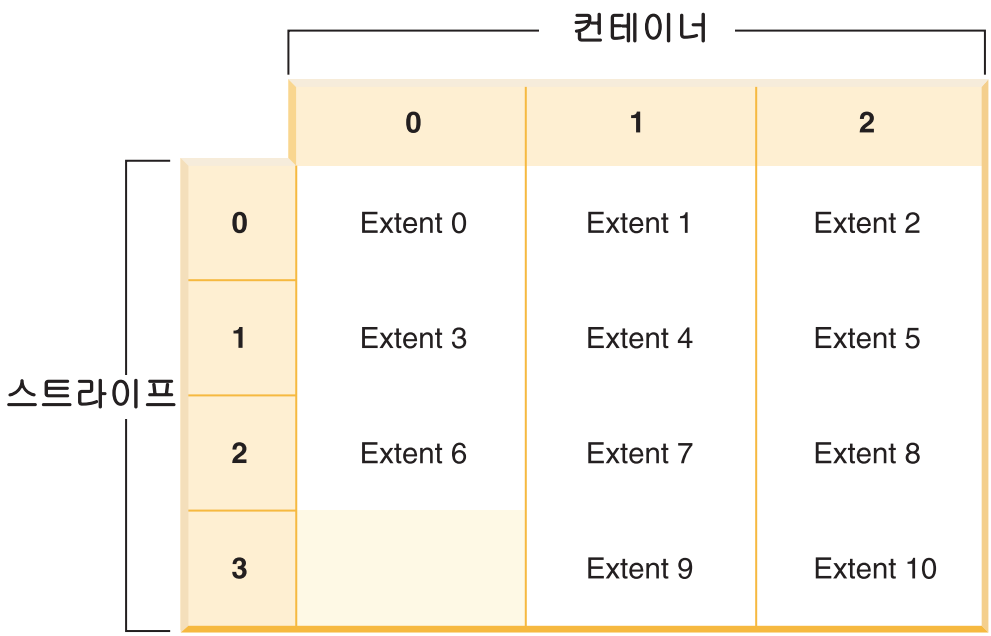


그림 25. 3개의 컨테이너와 11개의 Extent가 있는 테이블 스페이스

테이블 스페이스 스냅샷에 표시된 대로 해당 테이블 스페이스 맵의 모양은 다음과 같습니다.

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
   Number Set    Offset  Extent      Page Stripe Stripe
       [0]     [0]          0         8         89       0           2          0    3 (0, 1, 2)
       [1]     [0]          0        10        109       3           3          0    2 (1, 2)
```

BEGIN NEW STRIPE SET절을 사용하여 30페이지와 40페이지(각각 3 및 4개의 Extent)의 2개의 새 컨테이너를 추가하는 경우, 기존 범위가 영향을 받지 않는 대신 새 범위 세트가 작성됩니다. 이 새 범위 세트가 스트라이프 세트이고 최근에 작성된 것을 현재 스트라이프 세트라고 합니다. 두 개의 새 컨테이너가 추가된 후에는, 테이블 스페이스 모양이 217 페이지의 그림 26과 같이 표시됩니다.

그림 26. 2개의 스트라이프 세트가 있는 테이블 스페이스

테이블 스페이스 스냅샷에 표시된 바와 같이 해당 테이블 스페이스 맵은 다음과 같습니다.

```
Range     Stripe  Stripe  Max       Max    Start    End    Adj.   Containers
    Number Set     Offset  Extent     Page Stripe Stripe
         [0]     [0]          0          8          89       0           2          0    3 (0, 1, 2)
         [1]     [0]          0         10         109       3           3          0    2 (1, 2)
         [2]     [1]          4         16         169       4           6          0    2 (3, 4)
         [3]     [1]          4         17         179       7           7          0    1 (4)
```

테이블 스페이스에 새 컨테이너를 추가할 때 ADD절에 TO STRIPE SET절을 사용하지 않는 경우, 컨테이너는 현재 스트라이프 세트(최고 스트라이프 세트)에 추가됩니다. ADD TO STRIPE SET절을 사용하면 테이블 스페이스의 어느 스트라이프 세트에나 컨테이너를 추가할 수 있습니다. 유효한 스트라이프 세트를 지정해야 합니다.

데이터베이스 관리 프로그램은 테이블 스페이스 맵을 사용하여 스트라이프 세트를 추적하며, 재조정 없이 새 컨테이너를 추가하면 일반적으로 컨테이너가 재조정되었을 때보다 맵 크기가 빠르게 확장됩니다. 테이블 스페이스 맵이 너무 커지면 컨테이너를 추가하려고 할 때 SQL0259N 오류가 발생합니다.

DMS 테이블 스페이스에서 사용되지 않는 스토리지 재개

데이터베이스 관리 프로그램에 테이블 스페이스에서 낮은 사용 중 Extent를 통합하도록 하여 DMS 테이블 스페이스에서 사용되지 않는 스토리지를 재개할 수 있습니다. 이것은 상위 워터 마크(water mark)를 낮추는 효과도 있습니다. DMS 테이블 스페이스에서 컨테이너 크기를 줄이려면 별도의 REDUCE 조작도 수행해야 합니다.

시작하기 전에

DB2 버전 9.7 이상을 사용하여 작성된 DMS 테이블 스페이스가 있어야 합니다. 재개 가능 스토리지는 DB2 제품의 이전 버전을 사용하여 작성된 테이블 스페이스에서 사용할 수 없습니다. MON_GET_TABLESPACE 테이블 함수를 사용하여 데이터베이스의 어떤 테이블 스페이스가 재개 가능 스토리지를 지원하는지 확인할 수 있습니다.

이 태스크에 대한 정보

DMS 테이블 스페이스에서 사용되지 않는 스토리지를 재개하려면 먼저 테이블의 Extent가 재배열되도록 하여 테이블 스페이스에서 낮은 여유 Extent를 사용하도록 만드는 조작을 시작해야 합니다. 이것은 ALTER TABLESPACE문의 LOWER HIGH WATER MARK절을 사용하여 수행됩니다. 다음으로, 테이블 스페이스에 있는 컨테이너의 *크기*를 지정된 양만큼 줄일 수 있습니다.

DMS 테이블 스페이스에서 컨테이너의 *크기*를 줄일 때 줄일 컨테이너 이름을 지정하거나 ALL CONTAINERS절을 사용해야 합니다.

제한사항

- DB2 버전 9.7 이상을 사용하여 작성된 테이블 스페이스에서만 컨테이너를 재개할 수 있습니다.
- ALTER TABLESPACE문에 REDUCE 또는 LOWER HIGH WATER MARK 절을 지정할 때 다른 매개변수는 지정할 수 없습니다.
- 현재 상위 워터 마크(water mark)로 지정된 페이지를 보유한 Extent가 『삭제 보류』 상태에 있는 경우 Extent 이동을 통해 상위 워터 마크(water mark)를 낮추려는 시도는 실패할 수 있으며 메시지 ADM6008I가 로그됩니다. 『삭제 보류』 상태에 있는 Extent는 항상 복구 가능성 때문에 이동할 수 없습니다. 이들 Extent는 결국 일반 데이터베이스 유지보수 프로세스를 통해 비워지며, 이 때 이동할 수 있습니다.

프로시저

1. 테이블 스페이스 컨테이너 내부에서 Extent의 재배열을 통해 가능한 많이 상위 워터 마크(water mark)를 줄이려면 ALTER TABLESPACE문에서 LOWER HIGH WATER MARK절을 사용하십시오.
2. 일부 또는 모든 컨테이너의 크기를 지정된 양만큼 줄이려면 ALTER TABLESPACES문에서 REDUCE절을 사용하십시오.

예

예 1: 상위 워터 마크(water mark) 낮추기 및 모든 컨테이너를 5MB 축소. 다음 예는 테이블 스페이스 ts에 대한 상위 워터 마크(water mark)를 낮추고, 테이블 스페이스에 있는 모든 컨테이너의 *크기*를 5MB 줄입니다.

```
ALTER TABLESPACE ts LOWER HIGH WATER MARK
ALTER TABLESPACE ts REDUCE (ALL CONTAINERS 5 M)
```

예 2: 상위 워터 마크(water mark) 낮추기 및 『Container1』 컨테이너를 2000페이지 축소 다음 예는 테이블 스페이스 ts에 대한 상위 워터 마크(water mark)를 낮추고, 『Container1』 크기를 2000페이지 줄입니다.

```
ALTER TABLESPACE ts LOWER HIGH WATER MARK
ALTER TABLESPACE ts REDUCE (FILE "Container1" 2000)
```

컨테이너 추가 또는 삭제 후 자동 prefetchsize 조정

데이터베이스 관리 프로그램은 자동 프리페치 크기가 버전 8.2 이상을 사용하여 작성된 모든 테이블 스페이스의 디폴트값이 되도록 설정됩니다.

컨테이너를 추가 또는 삭제한 후 프리페치 크기 조정을 기억할 필요가 없습니다. 데이터베이스 관리 프로그램이 프리페치 크기를 자동으로 조정하기 위한 디폴트값이 사용되기 때문입니다. 컨테이너 추가 또는 삭제 후 테이블 스페이스의 프리페치 크기 갱신을 잊어버릴 가능성이 있을 경우, 디폴트를 변경하지 말고 데이터베이스 관리 프로그램이 프리페치 크기를 자동으로 결정하게 하십시오. 디폴트를 변경하여 자동 프리페치 크기 조정을 허용하지 않고 프리페치 크기 갱신을 잊은 경우, 데이터베이스 성능이 눈에 띄게 저하될 수 있습니다.

테이블 스페이스의 프리페치 크기를 AUTOMATIC으로 설정하지 않는 방법으로는 다음과 같은 세 가지 방법이 있습니다.

- 특정 프리페치 크기의 테이블 스페이스를 작성하십시오. 프리페치 크기에 대한 값을 수동으로 선택하면 테이블 스페이스와 연관된 컨테이너 수가 조정될 때마다 필요한 경우 프리페치 크기를 조정해야 합니다.
- 테이블 스페이스 작성시 프리페치 크기를 사용하지 않고 *dft_prefetch_sz* 데이터베이스 구성 매개변수를 AUTOMATIC이 아닌 값으로 설정하십시오. 데이터베이스 관리 프로그램은 테이블 스페이스 작성 시 프리페치 크기에 대한 명시적 언급이 없을 경우 이 매개변수를 점검합니다. AUTOMATIC 이외의 값이 존재할 경우, 해당하는 값이 디폴트 프리페치 크기로 사용됩니다. 또한 테이블 스페이스와 연관된 컨테이너의 수가 조정될 때마다 필요한 경우 프리페치 크기를 조정해야 합니다.
- ALTER TABLESPACE문을 사용하여 프리페치 크기를 수동으로 변경하십시오.

DB2_PARALLEL_IO의 사용

프리페치 요청은 테이블 스페이스의 병렬 처리에 따라, 요청이 프리페치 큐에 제출되기 전에 몇 개의 작은 프리페치 요청으로 분할됩니다.

DB2_PARALLEL_IO 레지스트리 변수는 컨테이너당 물리적 스핀들의 수를 정의하는데 사용되며 테이블 스페이스에 관한 병렬 I/O에 영향을 미칩니다. 병렬 I/O가 해제된 경우, 테이블 스페이스의 병렬 처리는 컨테이너 수와 같습니다. 병렬 I/O가 설정된 경우, 테이블 스페이스의 병렬 처리는 컨테이너 수를 DB2_PARALLEL_IO 레지스트리 변수에 제공된 값으로 곱한 것과 같습니다. (다르게 말하면, 테이블 스페이스의 병렬 처리는 프리페치 크기를 테이블 스페이스의 Extent 크기로 나눈 것과 같습니다.)

DB2_PARALLEL_IO 레지스트리 변수가 프리페치 *크기*에 영향을 미치는 방법에 대한 몇 가지 예는 다음과 같습니다. (다음 테이블 스페이스 모두가 AUTOMATIC 프리페치 *크기*로 정의되었다고 가정하십시오.)

- DB2_PARALLEL_IO=*
  - 모든 테이블 스페이스가 디폴트값을 사용하며, 여기서 스핀들의 수는 각 컨테이너에 대하여 6입니다. 프리페치 *크기*는 병렬 I/O 설정시 6배 더 큽니다.
  - 모든 테이블 스페이스에 병렬 I/O가 설정됩니다. 프리페치 요청이 몇 개의 작은 요청으로 분할되며, 각각은 프리페치 *크기*를 Extent *크기*로 나눈 값(또는 컨테이너 수 X 스핀들 수)과 같습니다.
- DB2_PARALLEL_IO=*:3
  - 모든 테이블 스페이스가 컨테이너당 스핀들 수로 3을 사용합니다.
  - 모든 테이블 스페이스에 병렬 I/O가 설정됩니다.
- DB2_PARALLEL_IO=*:3,1:1
  - 모든 테이블 스페이스가 컨테이너당 스핀들 수로 3을 사용합니다(1을 사용하는 테이블 스페이스 1은 제외).
  - 모든 테이블 스페이스에 병렬 I/O가 설정됩니다.

자동 스토리지를 사용하도록 테이블 스페이스 변환

데이터베이스의 데이터베이스 관리 스페이스(DMS) 테이블 스페이스의 전부 또는 일부를 자동 스토리지를 사용하도록 변환할 수 있습니다. 자동 스토리지 사용은 스토리지 관리 태스크를 단순하게 만듭니다.

시작하기 전에

데이터베이스에서 자동 스토리지를 사용할 수 있으며 자동 스토리지와 함께 사용할 하나 이상의 스토리지 경로가 정의되었는지 확인하십시오. 이를 수행하려면 ALTER DATABASE문을 사용하십시오.

프로시저

DMS 테이블 스페이스를 자동 스토리지를 사용하도록 변환하려면 다음 방법 중 하나를 사용하십시오.

- 단일 테이블 스페이스를 변경하십시오. 이 방법은 테이블 스페이스를 온라인으로 유지하지만 비자동 스토리지 컨테이너에서 새로운 자동 스토리지 컨테이너로 데이터를 이동하는 시간이 걸리는 재조정 조작이 포함됩니다.
  1. 변환하려는 테이블 스페이스에 대한 MANAGED BY AUTOMATIC STORAGE 절을 지정하고 ALTER TABLESPACE문을 발행하십시오.

2. 이번에는 REBALANCE 옵션을 지정하고 ALTER TABLESPACE문을 다시 발행하십시오. 이 옵션은 사용자 정의 컨테이너를 제거하므로 모든 테이블 스페이스 컨테이너가 자동 스토리지에 의해 관리됩니다.

   이제 REBALANCE 옵션을 지정하지 않고 나중에 REDUCE 옵션과 함께 ALTER TABLESPACE문을 발행하는 경우 자동 스토리지 컨테이너가 제거됩니다. 이 문제에서 복구하려면 REBALANCE 옵션을 지정하여 ALTER TABLESPACE문을 발행하십시오.

- 경로 재지정 리스토어 작업을 사용하십시오. 이 방법으로 단일 테이블 스페이스를 변환 중인 경우 조작이 진행되는 중에는 테이블 스페이스에 액세스할 수 없습니다. 다중 테이블 스페이스를 변환 중인 경우 조작이 진행되는 동안 전체 데이터베이스에 액세스할 수 없습니다.

  1. REDIRECT 매개변수를 지정하고 RESTORE DATABASE 명령을 실행하십시오. 단일 테이블 스페이스를 변환하려는 경우 TABLESPACE 매개변수를 지정하십시오.

     ```
     RESTORE DATABASE database_name TABLESPACE table_space_name REDIRECT
     ```

  2. 변환하려는 각 테이블 스페이스에 대해 USING AUTOMATIC STORAGE 매개변수를 지정하고 SET TABLESPACE CONTAINERS 명령을 실행하십시오.

     ```
     SET TABLESPACE CONTAINERS FOR tablespace_id USING AUTOMATIC STORAGE
     ```

  3. 이번에는 **CONTINUE** 매개변수를 지정하고 RESTORE DATABASE 명령을 실행하십시오.

     ```
     RESTORE DATABASE database_name CONTINUE
     ```

  4. **TO END OF LOGS** 및 **AND STOP** 매개변수를 지정하고 ROLLFORWARD DATABASE 명령을 실행하십시오.

     ```
     ROLLFORWARD DATABASE database_name TO END OF LOGS AND STOP
     ```

자동 스토리지 테이블 스페이스 변경

많은 자동 스토리지 테이블 스페이스의 유지보수는 자동으로 처리됩니다. 사용자가 자동 스토리지 테이블 스페이스에 수행할 수 있는 변경은 전체 테이블 스페이스의 크기 재조정 및 축소로 제한됩니다.

자동 스토리지 테이블 스페이스는 사용자를 위한 스토리지 할당을 관리하여 스토리지 경로가 부과하는 한계까지 필요한 대로 컨테이너를 작성하고 확장합니다. 사용자가 자동 스토리지 스페이스에 수행할 수 있는 유일한 유지보수는 다음과 같습니다.

- 재조정
- 상위 워터 마크(water mark)를 낮춰서 사용되지 않은 스토리지 재개
- 전체 테이블 스페이스의 크기 축소

데이터베이스에 스토리지를 추가할 때 자동 스토리지 테이블 스페이스를 재조정할 수 있습니다. 그러면 테이블 스페이스가 새 스토리지를 즉시 사용하기 시작합니다. 비슷하게, 데이터베이스에서 스토리지를 삭제할 때 재조정이 삭제되고 있는 스토리지 경로의 컨테이너 밖으로 이동하고 나머지 컨테이너 사이에 할당합니다.

새 스토리지 경로 추가 또는 경로 삭제는 데이터베이스 레벨에서 처리됩니다. 자동 스토리지 데이터베이스에 스토리지를 추가하려면 ALTER DATABASE문의 ADD STORAGE절을 사용합니다. 재조정을 수행하지 않는 경우 이전에 존재했던 컨테이너가 용량까지 채워질 때까지 새 스토리지가 사용되지 않지만, 사용자가 원하는 경우 재조정 여부를 선택할 수 있습니다. 재조정하는 경우 새로 추가되는 모든 스토리지 경로를 즉시 사용할 수 있습니다.

스토리지를 삭제(drop)하려면 ALTER DATABASE문의 DROP STORAGE절을 사용하십시오. 이 조치는 스토리지 경로를『삭제 보류』상태가 되도록 합니다. 사용자가 지정하는 스토리지 경로의 컨테이너 증가가 멈춥니다. 그러나 경로를 데이터베이스에서 완전히 제거할 수 있기 전에 ALTER TABLESPACE 명령의 REBALANCE 절을 사용하는 스토리지 경로를 사용하여 모든 테이블 스페이스를 재조정해야 합니다. 임시 테이블 스페이스가 삭제 보류 상태에 있는 스토리지 경로의 컨테이너를 갖는 경우 테이블 스페이스를 삭제하고 재작성하거나 데이터베이스를 재시작하여 스토리지 경로에서 제거할 수 있습니다.

제한사항: 임시 자동 스토리지 테이블 스페이스는 재조정할 수 없습니다. 재조정은 보통 및 대형 자동 스토리지 테이블 스페이스에 대해서만 지원됩니다.

ALTER TABLESPACE문의 LOWER HIGH WATER MARK절을 사용하여 테이블 스페이스의 상위 워터 마크(water mark) 아래의 스토리지를 재개할 수 있습니다. 이것은 가능한 많은 Extent를 테이블 스페이스에서 낮은 사용되지 않은 Extent로 이동하는 효과를 갖습니다. 테이블 스페이스에 대한 상위 워터 마크(water mark)가 프로세스에서 낮아지지만, 컨테이너는 조작이 수행되기 전에 있었던 *크기*로 남아있습니다.

자동 스토리지 테이블 스페이스는 ALTER TABLESPACE문의 REDUCE 옵션을 사용하여 *크기*를 줄일 수 있습니다. 자동 스토리지 테이블 스페이스의 *크기*를 줄일 때 데이터베이스 관리 프로그램은 테이블 스페이스에 대한 상위 워터 마크(water mark)를 낮추고 테이블 스페이스 컨테이너의 *크기*를 줄입니다. 상위 워터 마크(water mark)를 낮추려는 시도에서, 데이터베이스 관리 프로그램이 빈 컨테이너를 삭제(drop)하고 사용된 Extent를 테이블 스페이스의 시작에 더 가까운 여유 공간으로 이동할 수 있습니다. 다음으로, 컨테이너는 테이블 스페이스의 스페이스 총량이 상위 워터 마크(water mark)와 같거나 약간 더 크도록 *크기*가 조정됩니다.

자동 스토리지 테이블 스페이스에서 사용되지 않는 스토리지 재개

자동 스토리지 테이블 스페이스의 *크기*를 줄일 때 데이터베이스 관리 프로그램은 테이블 스페이스에 대한 *상위 워터 마크(water mark)*를 낮추고 테이블 스페이스 컨테이너

의 크기를 줄입니다. 상위 워터 마크(water mark)를 낮추려는 시도에서, 데이터베이스 관리 프로그램이 빈 컨테이너를 삭제(drop)하고 사용된 Extent를 테이블 스페이스의 시작에 더 가까운 여유 공간으로 이동할 수 있습니다. 다음으로, 컨테이너는 테이블 스페이스의 스페이스 총량이 상위 워터 마크(water mark)와 같거나 약간 더 크도록 크기가 조정됩니다.

시작하기 전에

DB2 버전 9.7 이상을 사용하여 작성된 자동 스토리지 테이블 스페이스가 있어야 합니다. 재개 가능 스토리지는 DB2 제품의 이전 버전을 사용하여 작성된 테이블 스페이스에서 사용할 수 없습니다. MON_GET_TABLESPACE 테이블 함수를 사용하여 데이터베이스의 어떤 테이블 스페이스가 재개 가능 스토리지를 지원하는지 확인할 수 있습니다.

이 태스크에 대한 정보

많은 방법으로 재개 가능 스토리지가 사용 가능한 자동 스토리지 스페이스의 크기를 줄일 수 있습니다. 데이터베이스 관리 프로그램이 다음에 따라 테이블 스페이스를 줄이도록 지정할 수 있습니다.

- 가능한 최대량
- KB, MB 또는 GB나 페이지 수로 지정하는 양
- 테이블 스페이스 현재 크기의 백분율.

어느 경우에나 데이터베이스 관리 프로그램은 Extent를 테이블 스페이스의 시작으로 이동하여 크기를 줄이려고 하며, 이 방법은 충분한 여유 공간을 사용할 수 있는 경우 테이블 스페이스의 상위 워터 마크(water mark)를 줄입니다. Extent 이동이 완료되면 테이블 스페이스 크기가 새 상위 워터 마크(water mark)로 줄어듭니다.

ALTER TABLESPACE문의 REDUCE절을 사용하여 자동 스토리지 테이블 스페이스에 대한 테이블 스페이스 크기를 줄입니다. 위에서 언급한 것처럼 테이블 스페이스를 줄일 양을 지정할 수 있습니다.

주:

- 테이블 스페이스를 줄일 양을 지정하지 않으면 테이블 스페이스 크기는 Extent를 이동하지 않고 가능한 만큼 많이 줄어듭니다. 데이터베이스 관리 프로그램이 먼저 삭제가 보류 중인 Extent를 비워서 컨테이너의 크기를 줄입니다. (일부 『삭제 보류』 Extent는 복구 가능성 때문에 비울 수 없으므로 이러한 Extent의 일부는 남아있을 수 있습니다.) 상위 워터 마크(water mark)가 비워진 Extent 사이에 있었던 경우 상위 워터 마크(water mark)가 낮아지며, 그렇지 않으면 상위 워터 마크(water mark)는 변경되지 않습니다. 다음으로, 컨테이너가 테이블 스페이스에 있는 스페이스의 총량이 상위 워터 마크(water mark)와 같거나 약간 더 크도록 크기 조정됩니다. 이 조작은 REDUCE절만을 갖는 ALTER TABLESPACE를 사용하여 수행됩니다.

- 어떤 컨테이너 조작도 수행하지 않고 테이블 스페이스에서 낮은 상태로 사용 중인 Extent를 통합하여 상위 워터 마크(water mark)만을 낮추려는 경우, LOWER HIGH WATER MARK절을 갖는 ALTER TABLESPACE문을 사용할 수 있습니다.
- REDUCE 또는 LOWER HIGH WATER MARK 조작이 진행 중이면 ALTER TABLESPACE문의 REDUCE STOP 또는 LOWER HIGH WATER MARK STOP절을 사용하여 중지할 수 있습니다. 이동된 모든 Extent가 커미트되고, 상위 워터 마크(water mark)는 새 값으로 줄어들고 컨테이너는 새로운 상위 워터 마크(water mark)로 *크기* 조정됩니다.

제한사항

- DB2 버전 9.7 이상을 사용하여 작성된 테이블 스페이스에서만 컨테이너를 재개할 수 있습니다.
- ALTER TABLESPACE문에 REDUCE 또는 LOWER HIGH WATER MARK 절을 지정할 때 다른 매개변수는 지정할 수 없습니다.
- 현재 상위 워터 마크(water mark)로 지정된 페이지를 보유한 Extent가 『삭제 보류』 상태에 있는 경우 Extent 이동을 통해 상위 워터 마크(water mark)를 낮추려는 시도는 실패할 수 있으며 메시지 ADM6008I가 로그됩니다. 『삭제 보류』 상태에 있는 Extent는 항상 복구 가능성 때문에 이동할 수 없습니다. 이들 Extent는 결국 일반 데이터베이스 유지보수 프로세스를 통해 비워지며, 이 때 이동할 수 있습니다.

프로시저

자동 스토리지 테이블 스페이스의 *크기*를 줄이려면 다음을 수행하십시오.

1. REDUCE절을 포함하는 ALTER TABLESPACE문을 공식화하십시오.

   ```
   ALTER TABLESPACE table-space-name REDUCE reduction-clause
   ```

2. ALTER TABLESPACE문을 실행하십시오.

예

예 1: 가능한 최대량만큼 자동 스토리지 테이블 스페이스 줄이기.

```
ALTER TABLESPACE TS1 REDUCE MAX
```

이 경우에 키워드 MAX가 REDUCE절의 일부로 지정되어 데이터베이스 관리 프로그램이 Extent의 최대 수를 테이블 스페이스의 시작으로 이동하도록 표시합니다.

예 2: 현재 테이블 스페이스 크기의 백분율만큼 자동 스토리지 테이블 스페이스 줄이기.

```
ALTER TABLESPACE TS1 REDUCE 25 PERCENT
```

이 명령문은 테이블 스페이스 TS1의 *크기*를 가능한 경우 현재 *크기*의 75%로 줄입니다.

시나리오: 자동 스토리지 테이블 스페이스를 갖는 스토리지 추가 및 제거

이 절의 세 시나리오는 자동 스토리지 테이블 스페이스에 스토리지 경로 추가 및 제거의 영향을 설명합니다.

테이블 스페이스를 재조정할 때의 고려사항:

- 어떤 이유로든 데이터베이스 관리 프로그램이 컨테이너를 추가 또는 삭제할 필요가 없다고 결정하는 경우 또는 컨테이너가 『스페이스 부족』 조건으로 인해 추가될 수 없는 경우 경고가 수신됩니다.
- REBALANCE절은 해당 절에 지정해야 합니다.
- 임시 자동 스토리지 테이블 스페이스를 재조정할 수 없습니다. 일반 및 대형 자동 스토리지 테이블 스페이스만 재조정할 수 있습니다.
- 재조정 호출은(스토리지 레이아웃이 다를 수 있지만) 롤 포워드 중에 재생되는 로그 조작입니다.
- 파티션된 데이터베이스 환경에서 재조정은 테이블 스페이스가 상주하는 모든 데이터베이스 파티션에서 시작됩니다.
- 스토리지 경로가 추가 또는 삭제될 때 사용자에게 재조정하도록 강요하지 않습니다. 사실 후속 스토리지 경로 조작은 재조정 조작을 수행하기 전에도 시간에 따라 수행될 수 있습니다. 스토리지 경로가 삭제되고 『사용 중이 아님』 상태에 있는 경우 ALTER DATABASE 조작의 파트로서 즉시 삭제됩니다. 스토리지 경로가 『사용 중』 상태에 있고 삭제되지만 테이블 스페이스가 재조정되지 않는 경우 스토리지 경로(이제는 『삭제 보류』 상태에 있는)가 추가 컨테이너나 데이터를 저장하는 데 사용되지 않습니다.

시나리오: 스토리지 경로 추가 및 자동 스토리지 테이블 스페이스 재조정:

이 시나리오는 스토리지 경로가 자동 스토리지 데이터베이스에 추가되는 방법 및 REBALANCE 조작이 새 스토리지 경로에 하나 이상의 컨테이너를 작성하는 방법을 나타냅니다.

이 시나리오는 데이터베이스에 새 스토리지 경로를 추가할 것이며 기존 테이블 스페이스를 새 경로에 스트라이프하려는 경우를 가정하고 있습니다. 입출력 병렬 처리가 각 테이블 스페이스의 스트라이프 세트에 새 컨테이너를 추가하여 개선됩니다.

ALTER DATABASE문에서 ADD STORAGE절을 사용하여 데이터베이스에 새 스토리지 경로를 추가합니다. 그런 다음 ALTER TABLESPACE문에서 REBALANCE 절을 사용하여 새 스토리지 경로에 컨테이너를 할당하고 데이터를 기존 컨테이너에서 새 컨테이너로 재조정하십시오. 작성되는 컨테이너의 수와 크기는 테이블 스페이스에 대한 현재 스트라이프 세트의 정의와 새 스토리지 경로의 여유 공간에 의존합니다.

그림 27은 재조정되는 테이블 스페이스의 "이전" 및 "이후" 레이아웃과 함께 추가되는 스토리지 경로를 나타냅니다.

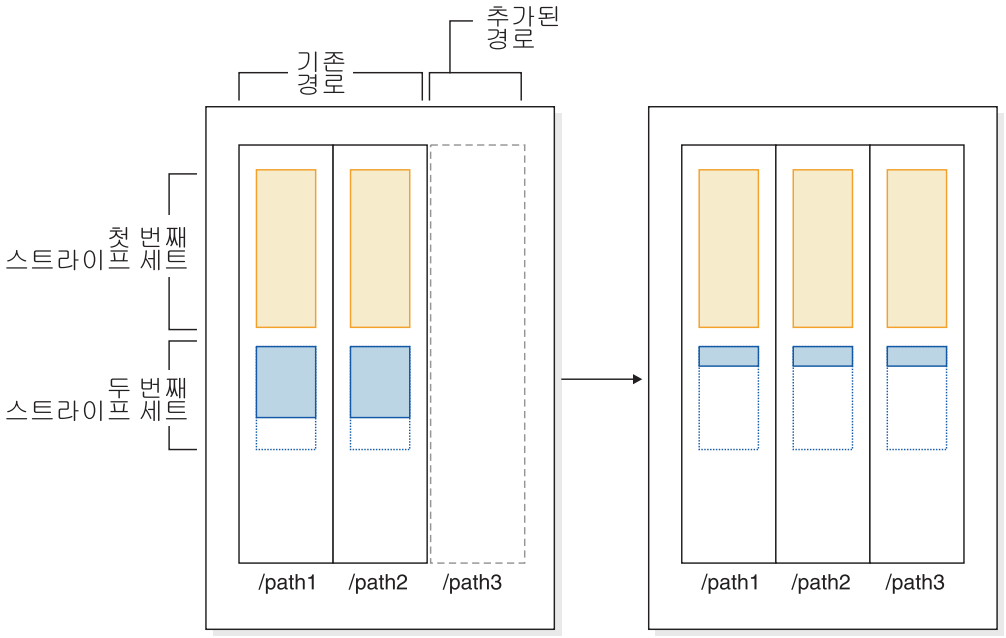


그림 27. 스토리지 경로 추가 및 자동 스토리지 테이블 스페이스 재조정

주: 이 주제에 표시되는 다이어그램은 단지 설명을 위한 것입니다. 스토리지 레이아웃에 대한 특정 접근이나 우수 사례를 제안하려는 것이 아닙니다. 또한 다이어그램은 단일 테이블 스페이스만을 설명합니다. 실제 사례에서는 동일한 스토리지 경로를 공유하는 여러 개의 자동 스토리지 테이블 스페이스를 가질 수 있습니다.

기존 테이블 스페이스가 서로 다른 수의 컨테이너를 갖는 여러 개의 스트라이프 세트를 가질 때 비슷한 상황이 발생할 수 있는데, 테이블 스페이스의 수명 동안 하나 이상의 스토리지 경로에서 *디스크 가득참* 조건으로 인해 발생할 수 있습니다. 이 경우에 데이터베이스 관리 프로그램이 기존 스토리지 경로에 컨테이너를 추가하여 스트라이프 세트의 『홀(hole)』을 채우는 것이 도움이 될 수 있습니다(물론 여유 공간이 있다고 가정). REBALANCE 조작을 사용하여 이 작업을 수행할 수도 있습니다.

227 페이지의 그림 28은 재조정될 테이블 스페이스의 스트라이프 세트에 『홀(hole)』이 존재하는 예이며(예를 들어 테이블 행을 삭제하여 유발될 수 있음), 스토리지 경로의 『이전』 및 『이후』 레이아웃이 포함되어 있습니다.

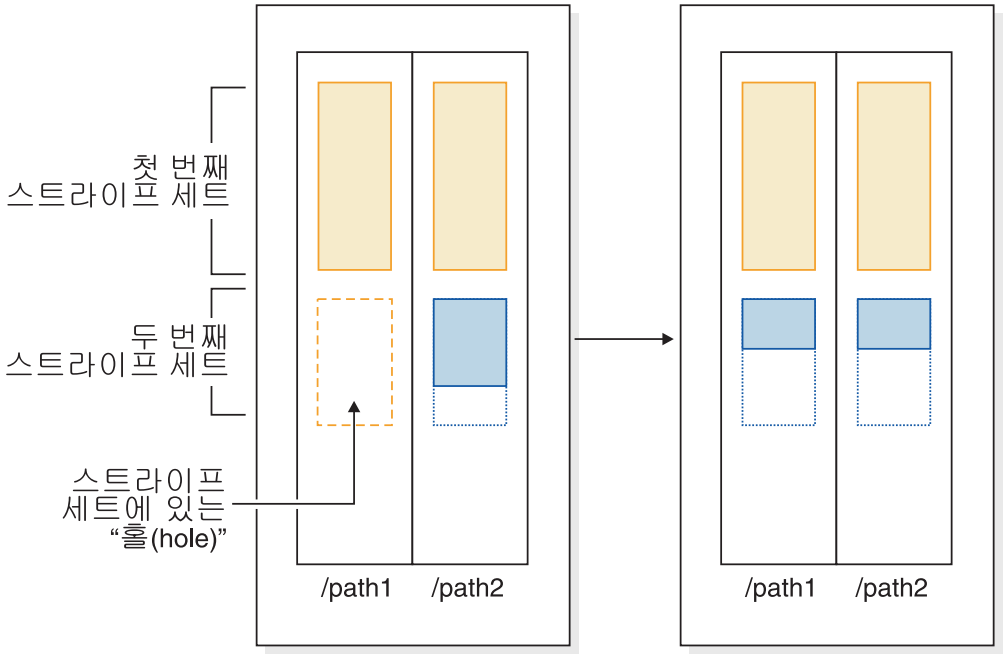


그림 28. 갭을 채우기 위해 자동 스토리지 테이블 스페이스 재조정

예

두 개의 스토리지 경로를 갖는 데이터베이스를 작성했습니다.

```
CREATE DATABASE TESTDB1 ON /databaseDataPath1, /databaseDataPath2
  DBPATH ON /databasePath
```

데이터베이스 작성 후, 자동 스토리지 테이블 스페이스가 작성되었습니다.

데이터베이스에 추가 스토리지 경로(/databaseDataPath3)를 추가하기로 결정했으며 모든 자동 스토리지 테이블 스페이스가 새 스토리지 경로를 사용하도록 허용하려고 합니다.

1. 첫 단계는 데이터베이스에 스토리지 경로를 추가하는 것입니다.

   ```
   ALTER DATABASE ADD STORAGE ON '/databaseDataPath3'
   ```

2. 다음 단계는 영향을 받는 모든 영구 테이블 스페이스를 판별하는 것입니다. 이것은 테이블 스페이스 스냅샷 출력을 수동으로 스캔하거나 SQL을 통해 수행될 수 있습니다. 다음 SQL문은 데이터베이스에 있는 모든 일반 및 대형 자동 스토리지 테이블 스페이스의 목록을 생성합니다.

   ```
   SELECT TBSP_NAME
     FROM SYSIBMADM.SNAPTBSP
     WHERE TBSP_USING_AUTO_STORAGE = 1
       AND TBSP_CONTENT_TYPE IN ('ANY','LARGE')
       ORDER BY TBSP_ID
   ```

3. 테이블 스페이스가 식별된 후에 다음 단계는 나열되는 각 테이블 스페이스에 대해 다음 명령문을 수행하는 것입니다. 남은 스토리지 경로에 충분한 스페이스가 있는 경우 일반적으로 재조정이 수행되는 순서(및 병렬로 실행될 수 있는 순서)는 문제가 되지 않습니다.

   ```
   ALTER TABLESPACE tablespace_name REBALANCE
   ```

이후에 임시 테이블 스페이스를 처리할 방법을 결정해야 합니다. 하나의 옵션은 데이터베이스를 중지(비활성화)한 후 시작(활성화)하는 것입니다. 그러면 컨테이너가 재정의됩니다. 다른 방법으로는, 임시 테이블 스페이스를 삭제하고 재작성하거나 새 임시 테이블 스페이스를 먼저 작성한 후 이전 스페이스를 삭제하는 것입니다. 이 방법에서는 데이터베이스의 마지막 임시 테이블 스페이스를 삭제하려고 시도하지 않으며 이것은 허용되지 않습니다. 영향을 받는 테이블 스페이스의 목록을 판별하기 위해 테이블 스페이스 스냅샷 출력을 수동으로 스캔하거나 SQL문을 실행할 수 있습니다. 다음 SQL문은 데이터베이스에 있는 모든 시스템 임시 및 사용자 임시 자동 스토리지 테이블 스페이스의 목록을 생성합니다.

```
SELECT TBSP_NAME
  FROM SYSIBMADM.SNAPTBSP
  WHERE TBSP_USING_AUTO_STORAGE = 1
    AND TBSP_CONTENT_TYPE IN ('USRTEMP','SYSTEMP')
    ORDER BY TBSP_ID
```

시나리오: 스토리지 경로 삭제 및 자동 스토리지 테이블 스페이스 재조정:

이 시나리오는 스토리지 경로가 삭제되는 방법 및 REBALANCE 조작이 해당 경로를 사용 중인 테이블 스페이스에서 컨테이너를 삭제하는 방법을 나타냅니다.

스토리지 경로 삭제 조작을 완료할 수 있기 전에 해당 경로의 모든 테이블 스페이스 컨테이너를 제거해야 합니다. 전체 테이블 스페이스가 더 이상 필요없는 경우 데이터베이스에서 스토리지 경로를 삭제하기 전에 스페이스를 삭제할 수 있습니다. 이 상황에서는 재조정이 필요없습니다. 그러나 테이블 스페이스를 보존하려는 경우 REBALANCE 조작이 필요합니다. 이 경우 『삭제 보류』 상태에 있는 스토리지 경로가 있을 때 데이터베이스 관리 프로그램이 *역방향 재조정*을 수행하는데, Extent 이동이 상위 워터 마크(water mark) Extent(데이터 스페이스에서 데이터를 포함하는 마지막으로 가능한 Extent)에서 시작하고 Extent 0에서 끝납니다.

REBALANCE 조작이 실행될 때 다음이 발생합니다.

- 역방향 재조정이 수행됩니다. 『삭제 보류』 상태에 있는 모든 컨테이너의 데이터가 나머지 컨테이너로 이동됩니다.
- 『삭제 보류』 상태의 컨테이너가 삭제됩니다.
- 이것이 스토리지 경로를 사용하는 마지막 테이블 스페이스인 경우 스토리지 경로도 삭제됩니다.

남은 스토리지 경로의 컨테이너가 이동될 모든 데이터를 보유하기에 충분히 크지 않은 경우, 데이터베이스 관리 프로그램이 재조정을 수행하기 전에 먼저 남은 스토리지 경로에 컨테이너를 작성하거나 확장해야 합니다.

그림 29는 삭제되는 스토리지 경로의 예이며, 테이블 스페이스가 재조정된 후 스토리지 경로의 『이전』 및 『이후』 레이아웃이 포함되어 있습니다.

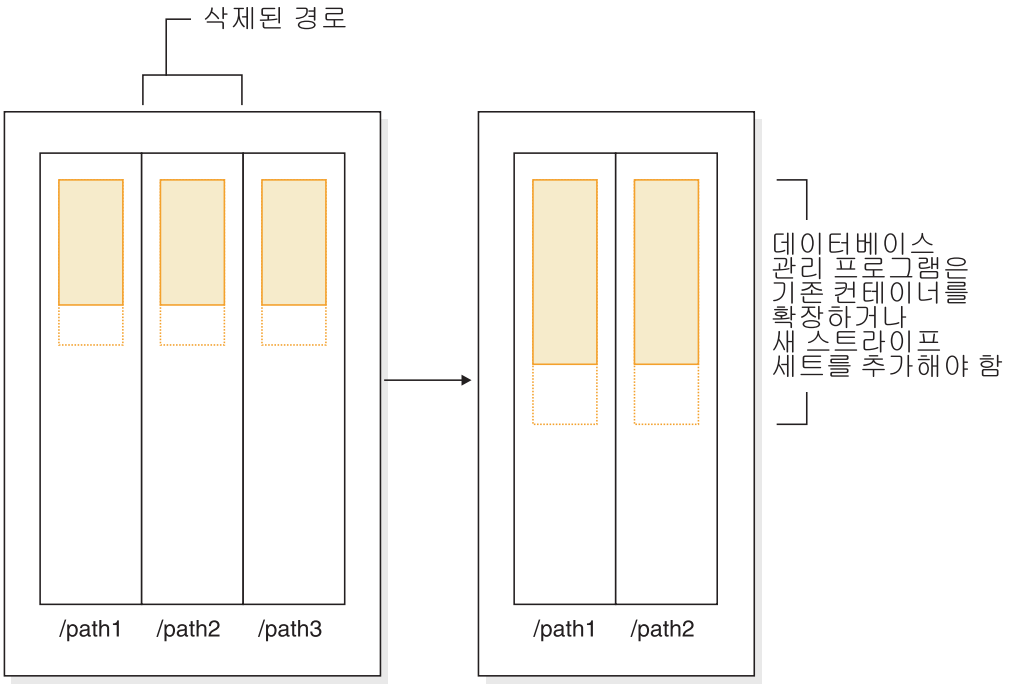


그림 29. 스토리지 경로 삭제 및 자동 스토리지 테이블 스페이스 재조정

예

다음 세 스토리지 경로를 갖는 데이터베이스를 작성했습니다.

```
CREATE DATABASE TESTDB2 ON D:₩DBDATA, E:₩DBDATA, F:₩DBDATA DBPATH ON C:
{Automatic storage tablespaces were subsequently created}
```

데이터베이스에서 F:₩DBDATA 스토리지 경로를 삭제하여 "삭제 보류" 상태로 한 후 이 스토리지 경로를 사용하는 모든 테이블 스페이스를 재조정하여 이를 삭제합니다.

1. 첫 단계는 데이터베이스에서 스토리지 경로의 삭제를 시작하는 것입니다.

   ```
   ALTER DATABASE DROP STORAGE ON 'F:₩DBDATA'
   ```

2. 다음 단계는 영향을 받는 모든 비임시 테이블 스페이스를 판별하는 것입니다. 이것은 테이블 스페이스 스냅샷 출력을 수동으로 스캔하거나 SQL을 통해 수행될 수 있습니다. 다음 SQL문은 『삭제 보류』 경로에 상주하는 컨테이너를 가진 데이터베이스에 있는 모든 일반 및 대형 자동 스토리지 테이블 스페이스의 목록을 생성합니다.

```
SELECT DISTINCT A.TBSP_NAME
  FROM SYSIBMADM.SNAPTBSP A, SYSIBMADM.SNAPTBSP_PART B
  WHERE A.TBSP_ID = B.TBSP_ID
    AND A.TBSP_CONTENT_TYPE IN ('ANY','LARGE')
    AND B.TBSP_PATHS_DROPPED = 1
```

3. 테이블 스페이스가 식별된 후에 다음 단계는 나열되는 각 테이블 스페이스에 대해 다음 명령문을 수행하는 것입니다.

   ```
   ALTER TABLESPACE <tablespace_name> REBALANCE
   ```

   a. 위에서 설명한 것 외에, 다중 경로가 데이터베이스에서 삭제되었지만 특히 그 중 하나에서 테이블 스페이스를 제거하는 우선순위가 고려되는 경우가 있을 수 있습니다. 이 경우 문제가 되는 경로와 일치하는 경로를 찾아서 데이터베이스의 컨테이너 목록을 쿼리할 수 있습니다. 예를 들어 /db2/path1이라는 경로를 고려하십시오. 다음 쿼리는 /db2/path1 경로에 상주하는 컨테이너를 갖는 테이블 스페이스의 목록을 제공합니다.

   ```
   SELECT TBSP_NAME FROM SYSIBMADM.SNAPCONTAINER
     WHERE CONTAINER_NAME LIKE '/db2/path1/%%'
     GROUP BY TBSP_NAME;
   ```

   b. 그런 다음 결과 세트의 각 테이블 스페이스에 대해 REBALANCE문을 발행할 수 있습니다.

4. 이후에 임시 테이블 스페이스를 처리할 방법을 결정해야 합니다. 하나의 옵션은 데이터베이스를 중지(비활성화)한 후 시작(활성화)하는 것입니다. 그러면 컨테이너가 재정의됩니다. 다른 방법으로, 삭제 후 재작성(또는 먼저 새 버전을 작성한 후 이전 버전 삭제)할 수 있습니다. 영향을 받는 테이블 스페이스의 목록을 판별하기 위해 테이블 스페이스 스냅샷 출력을 수동으로 스캔하거나 SQL문을 실행할 수 있습니다. 다음 SQL문은『삭제 보류』경로에 상주하는 컨테이너를 가진 데이터베이스의 모든 시스템 임시 및 사용자 임시 자동 스토리지 테이블 스페이스의 목록을 생성합니다.

   ```
   SELECT DISTINCT A.TBSP_NAME
     FROM SYSIBMADM.SNAPTBSP A, SYSIBMADM.SNAPTBSP_PART B
     WHERE A.TBSP_ID = B.TBSP_ID
       AND A.TBSP_CONTENT_TYPE IN ('USRTEMP','SYSTEMP')
       AND B.TBSP_PATHS_DROPPED = 1
   ```

시나리오: 스토리지 경로 추가, 제거 및 자동 스토리지 테이블 스페이스 재조정:

이 시나리오는 스토리지 경로를 추가 및 제거하는 방법 및 REBALANCE 조작이 모든 자동 스토리지 테이블 스페이스를 재조정하는 방법을 나타냅니다.

스토리지를 동시에 데이터베이스에 추가하고 삭제할 수 있습니다. 이것은 하나의 ALTER DATABASE문을 통해서 또는 일정한 시간(그 사이에 테이블 스페이스가 재조정되지 않은 시간)으로 구분되는 다중 ALTER DATABASE문을 통해 수행될 수 있습니다.

225 페이지의『시나리오: 스토리지 경로 추가 및 자동 스토리지 테이블 스페이스 재조정』에서 설명하는 대로, 데이터베이스 관리 프로그램이 스토리지 경로를 삭제할 때 스

트라이프 세트의 『홀(hole)』을 채우는 상황이 발생할 수 있습니다. 이 경우 데이터베이스 관리 프로그램이 프로세스의 일부로 컨테이너를 작성하고 컨테이너를 삭제합니다. 이러한 모든 시나리오에서 데이터베이스 관리 프로그램은 일부 컨테이너가 추가되고(여유 공간이 허용하는) 일부는 제거되어야 함을 인식합니다. 이들 시나리오에서 데이터베이스 관리 프로그램이 2단계 재조정 조작(단계 및 상태는 스냅샷 모니터 출력에서 설명됨)을 수행해야 합니다.

1. 첫 번째, 새 컨테이너가 새 경로(또는 『홀』을 채우는 경우 기존 경로)에 할당됩니다.
2. 포워드 재조정이 수행됩니다.
3. 역방향 재조정이 수행되어 데이터를 삭제될 경로의 컨테이너 밖으로 이동합니다.
4. 컨테이너가 물리적으로 삭제됩니다.

그림 30은 재조정되는 테이블 스페이스의 "이전" 및 "이후" 레이아웃과 함께 추가 및 삭제되는 스토리지 경로의 예입니다.

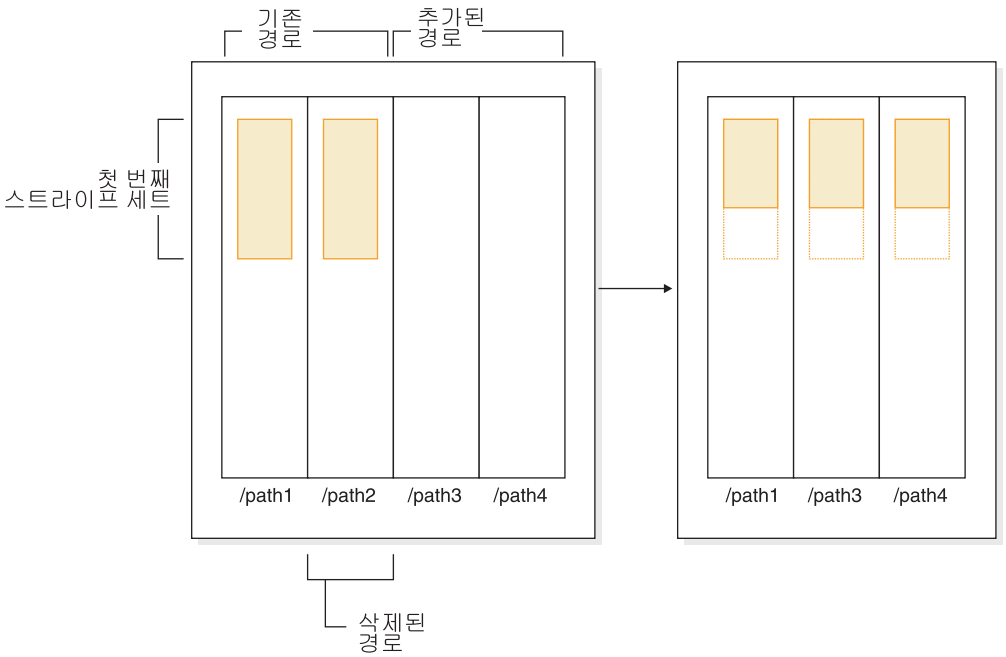


그림 30. 스토리지 경로 추가 및 삭제 후 자동 스토리지 테이블 스페이스 재조정

예

데이터베이스가 두 스토리지 경로와 함께 작성되었습니다.

```
CREATE DATABASE TESTDB3 ON /fs/data, /anotherfs DBPATH ON /fs/homePath
{Automatic storage tablespaces were subsequently created}
```

데이터베이스에 다른 스토리지 경로(/fs/data2)를 추가하고 기존 경로(/anotherfs) 중 하나를 제거하며 모든 자동 스토리지 테이블 스페이스를 재조정할 것으로 가정합니다. 첫 단계는 새 스토리지 경로 /fs/data2를 데이터베이스에 추가하고 /anotherfs의 제거를 시작하는 것입니다.

```
ALTER DATABASE ADD STORAGE ON '/fs/data2' DROP STORAGE ON '/anotherfs'
```

다음 단계는 영향을 받는 모든 테이블 스페이스를 판별하는 것입니다. 이것은 테이블 스페이스 스냅샷 출력을 수동으로 스캔하거나 SQL문을 통해 수행될 수 있습니다. 다음 SQL문은 데이터베이스에 있는 모든 일반 및 대형 자동 스토리지 테이블 스페이스의 목록을 생성합니다.

```
SELECT DISTINCT TBSP_ID, TBSP_NAME
  FROM SYSIBMADM.SNAPTBSP
  WHERE TBSP_USING_AUTO_STORAGE = 1
    AND TBSP_CONTENT_TYPE IN ('ANY','LARGE')
    ORDER BY TBSP_ID
```

테이블 스페이스가 식별된 후에 다음 단계는 나열되는 각 테이블 스페이스에 대해 다음 명령문을 수행하는 것입니다.

```
ALTER TABLESPACE <tablespace_name> REBALANCE
```

여기서 <tablespace_name>은 이전 단계에서 식별된 테이블 스페이스의 이름입니다.

주: 자동 스토리지가 관리하는 임시 테이블 스페이스는 재조정할 수 없습니다. 임시 테이블 스페이스에 할당된 스토리지 사용을 중지하려는 경우, 하나의 옵션은 임시 테이블 스페이스를 삭제한 후 다시 작성하는 것입니다.

테이블 스페이스 이름 바꾸기

테이블 스페이스 이름을 바꾸려면 RENAME TABLESPACE문을 사용하십시오.

SYSCATSPACE 테이블 스페이스의 이름을 바꿀 수 없습니다. 롤 포워드 보류 또는 롤 포워드 진행 중 상태에 있는 테이블 스페이스는 이름을 바꿀 수 없습니다.

백업된 이후에 이름이 바뀐 테이블 스페이스를 리스토어할 때, RESTORE DATABASE 명령에서 새 테이블 스페이스 이름을 사용해야 합니다. 이전 테이블 스페이스 이름을 사용하는 경우, 찾지 못합니다. 마찬가지로, ROLLFORWARD DATABASE 명령으로 테이블 스페이스를 롤 포워드하는 경우, 새 이름을 사용하십시오. 이전 테이블 스페이스 이름을 사용하는 경우, 찾지 못합니다.

테이블 스페이스에서 개별 오브젝트와 관련되지 않고 새 이름을 기존 테이블 스페이스에 부여할 수 있습니다. 테이블 스페이스의 이름을 바꿀 때, 해당 테이블 스페이스를 참조하는 모든 카탈로그 레코드가 변경됩니다.

오프라인에서 온라인으로 테이블 스페이스 전환

테이블 스페이스에 연관된 컨테이너가 액세스 가능한 경우, ALTER TABLESPACE 문의 SWITCH ONLINE절은 해당 테이블 스페이스에서 OFFLINE 상태를 제거하는 데 사용합니다.

테이블 스페이스는 제거된 OFFLINE 상태를 가지고 있는 반면, 나머지 데이터베이스는 여전히 위에 있고 사용되고 있습니다.

이 절의 사용에 대한 대체는 데이터베이스로부터 모든 응용프로그램을 연결해제했다가, 데이터베이스에 응용프로그램을 다시 연결하는 것입니다. 이것은 테이블 스페이스에서 OFFLINE 상태를 제거해 줍니다.

명령행을 사용하여 테이블 스페이스에서 OFFLINE 상태를 제거하려면, 다음을 입력하십시오.

```
db2 ALTER TABLESPACE <name>
SWITCH ONLINE
```

데이터가 RAID 디바이스에 있을 때 테이블 스페이스 성능 최적화

데이터가 RAID(Redundant Array of Independent Disks) 디바이스에 저장될 때 성능을 최적화하려면 이 지침을 따르십시오.

1. RAID 디바이스 세트에 테이블 스페이스를 작성할 때 별도의 디바이스에 주어진 테이블 스페이스(SMS 또는 DMS)에 대한 컨테이너를 작성하십시오.

   15개의 146GB 디스크가 각 배열에 5개 디스크를 갖는 세 개의 RAID-5 배열로서 구성된 예를 고려하십시오. 포맷팅 후 각 디스크는 대략 136GB의 데이터를 보유할 수 있습니다. 그러므로 각 배열은 대략 544GB(4 활성 디스크 x 136GB)를 저장할 수 있습니다. 300GB의 스토리지가 필요한 테이블 스페이스가 있는 경우 세 개의 컨테이너를 작성하고 각 컨테이너를 별도 디바이스에 넣으십시오. 각 컨테이너는 디바이스에서 100GB(300GB/3)를 사용하며, 각 디바이스에 추가 테이블 스페이스를 위한 444GB(544GB - 100GB)가 남아 있습니다.
2. 테이블 스페이스에 대해 적절한 Extent 크기를 선택하십시오. 테이블 스페이스의 Extent 크기는 데이터베이스 관리 프로그램이 다음 컨테이너에 쓰기 전에 한 컨테이너에 기록하는 데이터의 양입니다. 이상적으로는 Extent 크기는 디스크의 기초 세그먼트 크기의 배수여야 하며, 세그먼트 크기는 디스크 제어기가 다음 실제 디스크에 쓰기 전에 한 실제 디스크에 기록하는 데이터량입니다. 세그먼트 크기의 배수인 Extent 크기를 선택하면 프리페칭 시 병렬 시퀀스 읽기 같은 Extent 기반 조작이 동일한 실제 디스크에 대해 경합하지 않습니다. 또한 페이지 크기의 배수인 Extent 크기를 고려하십시오.

예에서 세그먼트 *크기*가 64KB이고 페이지 *크기*가 16KB인 경우 적합한 Extent *크기*는 256KB일 수 있습니다.

3. DB2_PARALLEL_IO 레지스트리 변수를 사용하여 모든 테이블 스페이스에 대해 병렬 I/O를 사용 가능하게 하고 컨테이너당 실제 디스크 수를 지정하십시오.

   예의 상황에 대해서는 DB2_PARALLEL_IO = *:4를 지정하십시오.

   테이블 스페이스의 프리페치 *크기*를 AUTOMATIC으로 설정하면, 데이터베이스 관리 프로그램은 사용자가 DB2_PARALLEL_IO에 대해 지정한 실제 디스크 수 값을 사용하여 프리페치 *크기* 값을 판별합니다. 프리페치 *크기*가 AUTOMATIC으로 설정되지 않는 경우, 세그먼트 *크기*에 활성 디스크 수를 곱한 값인 RAID 스트라이프 값을 고려하여 수동으로 설정할 수 있습니다. 다음 조건을 만족하는 프리페치 크기 값을 선택하십시오.

   - 이는 RAID 스트라이프 *크기*에 RAID 병렬 디바이스 수(또는 이 곱의 정수 표시)를 곱한 값과 같습니다.
   - Extent *크기*의 정수 표시입니다.

   예에서는 프리페치 *크기*를 768KB로 설정할 수 있습니다. 이 값은 RAID 스트라이프 *크기*(256KB)에 RAID 병렬 디바이스 수(3)를 곱한 것입니다. 또한 Extent *크기*(256KB)의 배수입니다. 이 프리페치 *크기* 선택은 단일 프리페치가 모든 배열의 모든 디스크에 관여함을 의미합니다. 워크로드가 주로 순차 스캔과 관련되기 때문에 프리페처가 보다 공격적으로 작업하기 원하는 경우 이 값의 배수(예: 1536KB(768KB x 2))를 대신 사용할 수 있습니다.

4. DB2_USE_PAGE_CONTAINER_TAG 레지스트리 변수를 설정하지 마십시오. 앞에서 설명한 것처럼, RAID 스트라이프 *크기*와 같거나 그의 배수인 Extent *크기*를 갖는 테이블 스페이스를 작성해야 합니다. 그러나 DB2_USE_PAGE_CONTAINER_TAG를 ON으로 설정하면, 1페이지 컨테이너 태그가 사용되고 Extent가 RAID 스트라이프와 정렬되지 않습니다. 결과적으로 입출력 요청 중에 최적의 조건보다 많은 실제 디스크에 액세스해야 합니다.

테이블 스페이스 삭제

테이블 스페이스를 삭제(drop)하는 경우, 해당 테이블 스페이스의 모든 데이터를 삭제하고, 컨테이너를 해제하고, 카탈로그 항목을 제거하십시오. 그러면 테이블 스페이스의 모든 정의된 오브젝트는 제거되거나 올바르지 않은 것으로 표시됩니다.

테이블 스페이스를 삭제하여 빈 테이블 스페이스의 컨테이너를 재사용할 수 있지만, 컨테이너를 재사용하기 전에 DROP TABLESPACE문을 커미트해야 합니다.

주: 그와 연관된 모든 테이블 스페이스를 삭제하지 않으면 테이블 스페이스를 삭제할 수 없습니다. 예를 들어 한 테이블 스페이스에 테이블이 있고 해당 인덱스가 다른 테이블 스페이스에 작성된 경우, 하나의 DROP TABLESPACE 명령에서 인덱스와 데이터 테이블 스페이스를 둘 다 삭제해야 합니다.

사용자 테이블 스페이스 삭제

단일 사용자 테이블 스페이스에서 인덱스 및 LOB 데이터를 비롯한 모든 테이블 데이터가 들어 있는 사용자 테이블 스페이스를 제거할 수 있습니다. 여러 개의 테이블 스페이스에 스팬되어 있는 사용자 테이블 스페이스를 제거할 수도 있습니다. 즉, 테이블 데이터를 테이블 스페이스 하나에 보관하고 인덱스는 또 다른 테이블 스페이스에 보관하여 모든 LOB를 세 번째 테이블 스페이스에 보관할 수 있습니다. 단일 명령문에서는 세 가지 테이블 스페이스를 동시에 모두 제거해야 합니다. 테이블이 들어 있는 모든 스팬된 테이블 스페이스는 이 단일 명령문의 일부가 되지 않으면, 제거 요청이 실패합니다.

명령행을 사용하여 사용자 테이블 스페이스를 삭제하려면 다음을 입력하십시오.

```
DROP TABLESPACE <name>
```

다음 SQL문은 ACCOUNTING 테이블 스페이스를 제거합니다.

```
DROP TABLESPACE ACCOUNTING
```

사용자 임시 테이블 스페이스 삭제

현재 해당 테이블 스페이스에 정의된 선언된 또는 작성된 임시 테이블이 없는 경우에만 사용자 임시 테이블 스페이스를 삭제할 수 있습니다. 테이블 스페이스를 제거할 때, 테이블 스페이스에 있는 모든 선언된 또는 작성된 임시 테이블을 제거하려는 시도는 수행되지 않습니다.

주: 선언된 또는 작성된 임시 테이블은 이를 선언한 응용프로그램이 데이터베이스에서 연결해제되면 내재적으로 삭제됩니다.

시스템 임시 테이블 스페이스 삭제

먼저 다른 시스템 임시 테이블 스페이스를 작성하지 않고는 페이지 *크기*가 4KB인 시스템 임시 테이블 스페이스를 삭제할 수 없습니다. 데이터베이스에는 항상 페이지 *크기*가 4KB인 시스템 임시 테이블 스페이스가 하나 이상 있어야 하므로, 새 시스템 임시 테이블 스페이스의 페이지 *크기*는 4KB이어야 합니다. 예를 들어, 페이지 *크기*가 4KB인 단일 시스템 임시 테이블 스페이스가 있고 여기에 컨테이너를 추가하고자 하며 *그것*이 SMS 테이블 스페이스인 경우, 먼저 적절한 수의 컨테이너가 있는 4KB 페이지 *크기*의 새 시스템 임시 테이블 스페이스를 추가한 다음 기존 시스템 임시 테이블 스페이스를 삭제하십시오 (DMS를 사용할 경우에는 테이블 스페이스를 삭제 및 재작성하지 않고 컨테이너를 추가할 수 있습니다).

디폴트 테이블 스페이스 페이지 *크기*는 데이터베이스가 작성된 페이지 *크기*입니다(4KB가 디폴트이지만, 8KB, 16KB 또는 32KB일 수 있음).

다음은 시스템 임시 테이블 스페이스를 작성하는 명령문입니다.

```
 CREATE SYSTEM TEMPORARY TABLESPACE <name>
MANAGED BY SYSTEM USING ('<directories>')
```

명령행을 사용하여 시스템 테이블 스페이스를 삭제하려면 다음을 입력하십시오.

```
DROP TABLESPACE <name>
```

다음 SQL문은 TEMPSPACE2라고 하는 새 시스템 임시 테이블 스페이스를 작성합니다.

```
 CREATE SYSTEM TEMPORARY TABLESPACE TEMPSPACE2
MANAGED BY SYSTEM USING ('d:₩systemp2')
```

일단 TEMPSPACE2가 작성되면, 원래의 시스템 임시 테이블 스페이스인 TEMPSPACE1을 다음 명령으로 제거할 수 있습니다.

```
DROP TABLESPACE TEMPSPACE1
```

제 9 장 스키마

*스키마*는 이름 지정된 오브젝트의 콜렉션입니다. 스키마를 통해 해당 오브젝트를 논리적으로 그룹화할 수 있습니다. 스키마는 이름 규정자이기도 합니다. 스키마를 사용함으로써 여러 오브젝트에 동일한 자연 이름을 사용할 수 있고 해당 오브젝트를 명확하게 참조할 수 있습니다.

예를 들어, 'INTERNAL' 및 'EXTERNAL'이라는 스키마 이름을 사용하면 두 개의 서로 다른 SALES 테이블(INTERNAL.SALES, EXTERNAL.SALES)을 쉽게 구별할 수 있습니다.

또한 스키마를 사용하면 이름 스페이스가 충돌하는 일 없이 다중 응용프로그램이 단일 데이터베이스에 데이터를 저장할 수 있습니다.

스키마는 *XML 스키마*와 구별되며, 이와 혼동하면 안 됩니다. XML 스키마는 XML 문서 컨텐츠의 유효성을 확인하고 구조에 대해 설명하는 표준입니다.

스키마에는 테이블, 뷰, 별칭, 트리거, 함수, 패키지 및 기타 오브젝트가 포함될 수 있습니다. 스키마 자체가 데이터베이스 오브젝트입니다. CREATE SCHEMA문을 사용하여 명시적으로 스키마를 작성하며 현재 사용자 또는 지정된 권한 부여 ID가 스키마 소유자로 기록됩니다. 사용자에게 IMPLICIT_SCHEMA 권한이 있는 경우에는 기타 오브젝트가 작성될 때 내재적으로 작성될 수도 있습니다.

*스키마 이름*은 두 부분으로 된 오브젝트 이름에서 상위 순서 파트로 사용됩니다. 오브젝트 작성 시 스키마 이름을 사용하여 구체적으로 오브젝트를 규정하는 경우 오브젝트가 해당 스키마에 지정됩니다. 오브젝트 작성 시 스키마 이름을 지정하지 않는 경우에는 CURRENT SCHEMA 특수 레지스터에 지정된 디폴트 스키마 이름이 사용됩니다.

예를 들어, DBADM 권한을 가진 사용자가 다음과 같이 사용자 A가 사용할 C라는 스키마를 작성합니다.

```
CREATE SCHEMA C AUTHORIZATION A
```

그러면 사용자 A가 다음 명령문을 실행하여 스키마 C에 X라는 테이블을 작성할 수 있습니다(사용자 A에게 CREATETAB 데이터베이스 권한이 있는 경우).

```
CREATE TABLE C.X (COL1 INT)
```

일부 스키마 이름은 예약되어 있습니다. 예를 들면, SYSIBM 스키마에 속하는 내장 함수와 SYSFUN 스키마에 속하는 사전 설치된 사용자 정의 함수(UDF)가 있습니다.

데이터베이스 작성 시 RESTRICTIVE 옵션을 사용하여 작성하지 않을 경우 모든 사용자가 IMPLICIT_SCHEMA 권한을 갖습니다. 이 권한이 있으면 사용자가 아직 존

재하지 않는 스키마 이름을 사용하여 오브젝트를 작성할 때마다 내재적으로 스키마를 작성합니다. 스키마가 내재적으로 작성되면 CREATEIN 특권이 부여되어 모든 사용자가 이 스키마에서 기타 오브젝트를 작성할 수 있습니다. 내재적으로 작성된 스키마에서도 별명, 구별 유형, 함수 및 트리거와 같은 오브젝트 작성 기능을 사용할 수 있습니다. 내재적으로 작성된 스키마에 대한 디폴트 특권을 통해 이전 버전과 호환이 가능합니다.

PUBLIC에서 IMPLICIT_SCHEMA 권한이 취소되는 경우 CREATE SCHEMA문을 사용하여 명시적으로 스키마를 작성하거나 IMPLICIT_SCHEMA 권한이 부여된 사용자(예: DBADM 권한을 가진 사용자)가 내재적으로 스키마를 작성할 수 있습니다. PUBLIC에서 IMPLICIT_SCHEMA 권한이 취소되면 스키마 이름 사용에 대한 제어가 증가하지만 기존 응용프로그램에서 오브젝트를 작성하려 할 경우 권한 부여 오류가 발생할 수 있습니다.

스키마에도 특권이 있어 스키마 소유자가 스키마에서 오브젝트를 작성, 변경, 복사 및 삭제(drop)할 특권을 갖는 사용자를 제어할 수 있습니다. 이는 데이터베이스의 오브젝트 서브세트 처리를 제어할 방법을 제공합니다. 처음에 스키마 소유자에게는 스키마에 대한 모든 해당 특권이 부여되고, 다른 사용자에게 특권을 부여하는 기능도 주어집니다. 내재적으로 작성된 스키마는 시스템이 소유하며 모든 사용자에게는 처음에 해당 스키마에서 오브젝트를 작성할 특권이 부여됩니다. ACCESSCTRL 또는 SECADM 권한을 가진 사용자는 모든 스키마에 대해 사용자가 보유하고 있는 특권을 변경할 수 있습니다. 따라서 내재적으로 작성된 스키마를 포함하여 모든 스키마에서 오브젝트를 작성, 변경, 복사 및 삭제(drop)하기 위해 액세스하는 것을 제어할 수 있습니다.

스키마 설계

데이터를 테이블로 구성할 때 테이블 및 기타 관련 오브젝트를 함께 그룹화하면 좋습니다. CREATE SCHEMA문을 사용하여 스키마를 정의함으로써 그룹화를 수행할 수 있습니다.

스키마에 대한 정보는 사용자가 연결되어 있는 데이터베이스의 시스템 카탈로그 테이블에 보존됩니다. 다른 오브젝트가 작성되면 이 오브젝트를 사용자가 작성하는 스키마 내에 저장할 수 있지만 하나의 스키마에 하나의 오브젝트만 있을 수 있습니다.

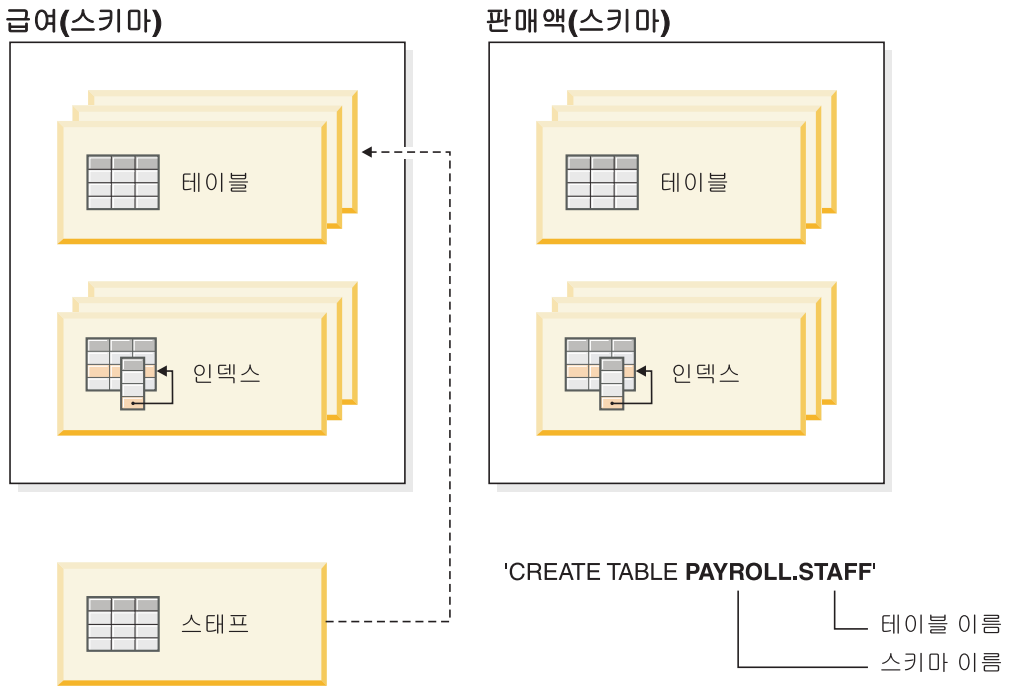
스키마를 디렉토리와 비교하면 현재 스키마는 현재 디렉토리가 됩니다. 이와 같은 유추를 적용하면, SET SCHEMA는 change directory 명령에 해당됩니다.

중요사항: 디폴트 CURRENT SCHEMA 설정(아래에 설명) 이외에는 권한 부여 ID와 스키마 간에 아무 관계가 없음을 이해하는 것이 중요합니다.

데이터베이스 및 테이블을 설계할 때 시스템의 스키마도 고려해야 하며 스키마 이름 및 각 스키마와 연관될 오브젝트에 대해서도 고려해야 합니다.

데이터베이스의 대부분의 오브젝트에는 두 파트로 구성된 고유 이름이 지정됩니다. 첫 번째(가장 왼쪽) 파트를 규정자 또는 스키마라고 하고 두 번째(가장 오른쪽) 파트를 단순(또는 규정되지 않은) 이름이라고 합니다. 구문상 이 두 파트는 마침표로 구분되는 단일 문자열로 연결됩니다. 스키마 이름으로 규정할 수 있는 오브젝트(예: 테이블, 인덱스, 뷰, 사용자 정의 데이터 유형, 사용자 정의 함수, 별칭, 패키지 또는 트리거)가 먼저 작성되면 이 오브젝트는 이름의 규정자에 따라서 특정 스키마에 지정됩니다.

예를 들어, 다음 다이어그램에서는 테이블 작성 프로세스 중에 테이블을 특정 스키마에 지정하는 방법에 대해 설명합니다.



사용자에게 올바른 권한 및 명령어를 지정하려면 스키마 액세스 권한을 부여하는 방법도 잘 알고 있어야 합니다.

스키마 이름

새 스키마 작성 시 스키마 이름은 카탈로그에서 이미 설명한 스키마 이름을 식별해서는 안 되며 이름을 "SYS"로 시작할 수 없습니다. 기타 제한사항 및 권장사항은 242 페이지의 『스키마 이름 제한사항 및 권장사항』을 참조하십시오.

스키마에 액세스

스키마는 데이터베이스에서 고유성을 부여하는 데 사용되므로 스키마에 있는 오브젝트에 대한 규정되지 않은 액세스는 허용되지 않습니다. 이는 두 명의 사용자가 동일한 이름을 가진 두 개의 테이블(또는 기타 오브젝트)을 작성할 가능성을 고려하면 분명히 알 수 있습니다. 고유성을 부여할 스키마가 없으면 세 번째 사용자가 테이블을 쿼리하려는 경우 모호한 점이 존재합니다. 추가로 규정하지 않으면 사용할 테이블을 판별하기가 불가능합니다.

CREATE SCHEMA문의 파트로 작성된 오브젝트의 정의자는 스키마 소유자입니다. 이 소유자는 다른 사용자에게 스키마 특권을 부여하고 권한 취소할 수 있습니다.

사용자에게 DBADM 권한이 있는 경우 해당 사용자는 유효한 이름을 사용하여 스키마를 작서할 수 있습니다. 데이터베이스가 작성되면 IMPLICIT_SCHEMA 권한이 PUBLIC에(즉, 모든 사용자에게) 부여됩니다.

사용자에게 IMPLICIT_SCHEMA 또는 DBADM 권한이 없는 경우 사용자는 자신의 권한 부여 ID와 동일한 이름을 가진 스키마만 작성할 수 있습니다.

디폴트 스키마

스키마 또는 규정자가 작성할 오브젝트 이름의 파트로 지정되지 않은 경우 해당 오브젝트는 CURRENT SCHEMA 특수 레지스터에 표시된 대로 디폴트 스키마에 지정됩니다. 이 특수 레지스터의 디폴트값은 세션 권한 부여 ID의 값입니다.

동적문의 규정되지 않은 오브젝트 참조에 디폴트 스키마가 필요합니다. CURRENT SCHEMA 특수 레지스터를 디폴트로 사용할 스키마에 설정하여 특정 DB2 연결에 사용할 디폴트 스키마를 지정할 수 있습니다. 이 특수 레지스터를 설정하기 위해 지정된 권한이 필요하지 않으므로 모든 사용자가 CURRENT SCHEMA를 설정할 수 있습니다.

SET SCHEMA문의 구문은 다음과 같습니다.

```
SET SCHEMA = <schema-name>
```

대화식으로 또는 응용프로그램 내에서 이 명령문을 실행할 수 있습니다. CURRENT SCHEMA 특수 레지스터의 초기값은 현재 세션 사용자의 권한 부여 ID와 같습니다. 자세한 정보는 SET SCHEMA문을 참조하십시오.

주:

- 다른 방법으로 연결 시 디폴트 스키마를 설정할 수도 있습니다. 예를 들면, CLI/ODBC 응용프로그램에서는 `cli.ini` 파일을 사용하거나 JDBC 응용프로그램 인터페이스에서는 연결 등록 정보를 사용할 수 있습니다.
- 디폴트 스키마 레코드는 시스템 카탈로그에 작성되지 않지만 작성할 오브젝트 이름의 파트로 스키마 또는 규정자가 지정되지 않을 때마다 데이터베이스 관리 프로그램이 CURRENT SCHEMA 특수 레지스터에서 가져올 수 있는 값으로만 존재합니다.

내재된 작성

IMPLICIT_SCHEMA 권한이 있으면 내재적으로 스키마를 작성할 수 있습니다. 이 권한이 있으면 아직 존재하지 않는 스키마 이름을 사용하여 오브젝트를 작성할 때마다 내재적으로 스키마를 작성할 수 있습니다. 오브젝트를 작성하는

사용자에게 IMPLICIT_SCHEMA 권한이 있는 경우에는 종종 스키마의 데이터 오브젝트가 처음으로 작성될 때 스키마가 내재적으로 작성됩니다.

명시적 작성

명령행이나 응용프로그램에서 CREATE SCHEMA 및 DROP SCHEMA문을 실행하여 명시적으로 스키마를 작성하고 삭제(drop)할 수도 있습니다. 자세한 정보는 CREATE SCHEMA 및 DROP SCHEMA문을 참조하십시오.

스키마별 테이블 및 뷰 별명

다른 사용자가 테이블 또는 뷰에 대한 자격의 파트인 스키마 이름을 입력하지 않아도 테이블 또는 뷰에 액세스할 수 있도록 하려면 해당 사용자의 별명을 설정해야 합니다. 별명의 정의는 사용자의 스키마를 포함한 완전한 테이블 또는 뷰 이름을 정의합니다. 그러면 사용자가 별명을 사용하여 쿼리할 수 있습니다. 별명은 별명 정의의 파트로서 사용자의 스키마를 통해 완전해집니다.

스키마별 오브젝트 그룹화

데이터베이스 오브젝트 이름은 단일 ID로 구성되거나, 두 개의 ID로 구성된 *스키마 규정 오브젝트*입니다. 스키마 규정 오브젝트의 스키마(또는 상위 순서 파트)는 데이터베이스의 오브젝트를 분류하거나 그룹화할 수 있는 수단을 제공합니다. 테이블, 뷰, 별명, 구별 유형, 함수, 인덱스, 패키지 또는 트리거와 같은 오브젝트가 작성되면 스키마에 오브젝트가 지정됩니다. 명시적 또는 내재적으로 이러한 지정이 수행됩니다.

명령문에서 해당 오브젝트를 가리키는 경우 두 파트로 이루어진 오브젝트 이름의 상위 순서 파트를 사용하면 명시적으로 스키마를 사용하게 됩니다. 예를 들면, 사용자 A가 다음과 같이 스키마 C에서 CREATE TABLE문을 실행합니다.

```
CREATE TABLE C.X (COL1 INT)
```

두 파트로 이루어진 오브젝트 이름의 상위 순서 파트를 사용하지 않으면 내재적으로 스키마를 사용하게 됩니다. 이런 경우 오브젝트 이름의 상위 순서 파트를 완료하기 위해 CURRENT SCHEMA 특수 레지스터를 사용하여 스키마 이름을 식별합니다. CURRENT SCHEMA의 초기 값은 현재 세션 사용자의 권한 부여 ID입니다. 현재 세션 중에 이 값을 변경하려면 SET SCHEMA문을 사용하여 다른 스키마 이름으로 특수 레지스터를 설정할 수 있습니다.

일부 오브젝트는 데이터베이스 작성 시 특정 스키마 내에서 작성되어 시스템 카탈로그 테이블에 저장됩니다.

오브젝트를 작성할 스키마를 명시적으로 지정할 필요가 없습니다. 스키마를 지정하지 않는 경우 명령문의 권한 부여 ID가 사용됩니다. 예를 들어, 다음 CREATE TABLE문에서 스키마 이름은 디폴트로 현재 로그온되어 있는 권한 부여 ID로 설정됩니다 (즉, CURRENT SCHEMA 특수 레지스터 값).

```
CREATE TABLE X (COL1 INT)
```

동적 SQL 및 XQuery 명령문에서는 일반적으로 CURRENT SCHEMA 특수 레지스터 값을 사용하여 규정되지 않은 오브젝트 이름 참조를 내재적으로 규정합니다.

사용자 고유 오브젝트를 작성하기 전에 먼저 오브젝트를 사용자의 스키마에서 작성할지 또는 논리적으로 오브젝트를 그룹화하는 다른 스키마를 사용하여 작성할지 고려해야 합니다. 공유할 오브젝트를 작성하려는 경우에는 다른 스키마 이름을 사용하는 것이 더 좋습니다.

스키마 이름 제한사항 및 권장사항

스키마 이름을 지정할 때 알아야 하는 몇 가지 제한사항과 권장사항이 있습니다.

- 사용자 정의 유형(UDT)은 *SQL 참조서*의 『SQL 및 XML 한계』에 나열되는 스키마 길이보다 긴 스키마 이름을 가질 수 없습니다.
- 스키마 이름 SYSCAT, SYSFUN, SYSIBM, SYSSTAT, SYSPROC은 예약어이므로 사용해서는 안 됩니다.
- 나중에 데이터베이스를 업그레이드할 때 문제점이 발생하지 않도록 하려면 SYS로 시작하는 스키마 이름을 사용하지 마십시오. 데이터베이스 관리 프로그램에서는 SYS로 시작되는 스키마 이름을 사용하여 트리거, 사용자 정의 유형 또는 사용자 정의 함수를 작성할 수 없습니다.
- SESSION을 스키마 이름으로 사용하지 않는 것이 좋습니다. 선언된 임시 테이블을 SESSION으로 규정해야 합니다. 따라서 응용프로그램에서 영구 테이블의 이름과 동일한 이름으로 임시 테이블을 선언할 수 있습니다. 그럴 경우 응용프로그램 논리가 지나치게 복잡해집니다. 선언된 임시 테이블을 처리할 때를 제외하고는 SESSION 스키마를 사용하지 마십시오.

스키마 작성

스키마를 사용하여 오브젝트를 작성할 때 해당 오브젝트를 그룹화할 수 있습니다. 하나의 오브젝트는 하나의 스키마에만 속할 수 있습니다. 스키마를 작성하려면 CREATE SCHEMA문을 사용합니다. 스키마에 대한 정보는 연결한 데이터베이스의 시스템 카탈로그 테이블에 저장됩니다.

스키마를 작성하고 다른 사용자를 해당 스키마의 소유자로 선택적으로 지정하려면 DBADM 권한이 필요합니다. DBADM 권한이 없으면 권한 부여 ID를 사용하여 스키마를 작성할 수 있습니다. CREATE SCHEMA문의 파트로 작성된 오브젝트의 정의자는 스키마 소유자입니다. 이러한 소유자는 다른 사용자에게 스키마 특권에 대한 권한을 부여하고 취소할 수 있습니다.

명령행에서 스키마를 작성하려면 다음 명령문을 입력하십시오.

```
CREATE SCHEMA <schema-name> [ AUTHORIZATION <schema-owner-name> ]
```

여기서 <schema-name>은 스키마 이름입니다. 이 이름은 카탈로그에 이미 기록된 스키마 내에서 고유해야 하고 SYS로 시작할 수 없습니다. 선택적 AUTHORIZATION절이 지정되어 있으면 <schema-owner-name>이 스키마의 소유자가 됩니다. 이 절이 지정되어 있지 않으면 이 명령문을 발행한 권한 부여 ID가 스키마의 소유자가 됩니다.

자세한 정보는 CREATE SCHEMA문을 참조하십시오. 242 페이지의 『스키마 이름 제한사항 및 권장사항』의 내용도 참조하십시오.

스키마 복사

db2move 유틸리티 및 ADMIN_COPY_SCHEMA 프로시저를 통해 데이터베이스 스키마의 사본을 빠르게 만들 수 있습니다. 모델 스키마가 설정되면 이 스키마를 새 버전 작성을 위한 템플리트로 사용할 수 있습니다.

ADMIN_COPY_SCHEMA 프로시저를 사용하여 동일한 데이터베이스 내에서 단일 스키마를 복사합니다. 또는 -co COPY 조치와 함께 db2move 유틸리티를 사용하여 소스 데이터베이스에서 목표 데이터베이스로 단일 스키마 또는 여러 스키마를 복사합니다. 새 스키마 아래에서 소스 스키마의 데이터베이스 오브젝트 대부분이 목표 데이터베이스로 복사됩니다.

문제점 해결 추가 정보

ADMIN_COPY_SCHEMA 프로시저 및 db2move 유틸리티는 모두 LOAD 명령을 호출합니다. 로드가 처리되는 중 데이터베이스 목표 오브젝트가 상주하고 있는 테이블 스페이스는 백업 보류 상태가 됩니다.

ADMIN_COPY_SCHEMA 프로시저

COPYNO 옵션과 함께 이 프로시저를 사용하면 위의 주에서 설명한 것처럼 목표 오브젝트가 상주하고 있는 테이블 스페이스가 백업 보류 상태가 됩니다. 테이블 스페이스를 무결성 설정 보류 상태에서 벗어나도록 하기 위해 이 프로시저는 SET INTEGRITY문을 발행합니다. 목표 테이블 오브젝트에 정의된 참조 제한조건이 있는 경우 목표 테이블도 무결성 설정 보류 상태가 됩니다. 테이블 스페이스가 이미 백업 보류 상태이므로 ADMIN_COPY_SCHEMA 프로시저는 SET INTEGRITY문을 발행하지 못합니다.

이러한 상황을 해결하려면 영향을 받는 테이블 스페이스를 백업 보류 상태에서 벗어나도록 BACKUP DATABASE 명령을 발행하십시오. 다음으로 이 프로시저에서 생성한 오류 테이블의 **Statement_text** 컬럼을 검색하여 무결성 설정 보류 상태인 테이블 목록을 찾으십시오. 그런 다음 나열된 각 테이블에 대해 SET INTEGRITY문을 발행하여 각 테이블을 무결성 설정 보류 상태에서 벗어나도록 하십시오.

db2move 유틸리티

이 유틸리티는 허용되는 모든 스키마 오브젝트의 복사를 시도하지만 다음과 같은 유형의 예외가 적용됩니다.

- 테이블 계층
- 스테이징 테이블(다중 파티션 데이터베이스 환경에서는 로드 유틸리티에 의해 지원되지 않음)
- jars(Java™ 루틴 아카이브)
- 별칭
- 패키지
- 뷰 계층 구조
- 오브젝트 특권(모든 새 오브젝트는 디폴트 권한 부여를 사용하여 작성됨)
- 통계(새 오브젝트에는 통계 정보가 없음)
- 인덱스 확장자(사용자 정의 구조화된 유형 관련)
- 사용자 정의 구조화된 유형 및 해당 변환 함수

지원되지 않는 유형 오류

소스 스키마에서 한 가지 지원되지 않는 유형의 오브젝트가 발견된 경우 항목이 오류 파일에 로그되어 지원되지 않는 오브젝트 유형이 발견되었음을 나타냅니다. 조작에 성공하고 로그된 항목은 이 조작에 의해 오브젝트가 복사되지 않았음을 나타냅니다.

스키마로 결합되지 않은 오브젝트

스키마로 결합되지 않은 오브젝트(예: 테이블 스페이스 및 이벤트 모니터)는 스키마 복사 조작 중 조작되지 않습니다. 스키마 복사 조작이 호출되기 전에 목표 데이터베이스에서 해당 오브젝트를 작성해야 합니다.

복제된 테이블

복제된 테이블을 복사하는 경우 테이블의 새 사본을 복제할 수 없습니다. 해당 테이블은 일반 테이블로 다시 작성됩니다.

다른 인스턴스

소스 데이터베이스는 목표 데이터베이스와 동일한 인스턴스에 상주하지 않은 경우 카탈로그되어야 합니다.

SCHEMA_MAP 옵션

SCHEMA_MAP 옵션을 사용하여 목표 데이터베이스에서 다른 스키마 이름을 지정하면 스키마 복사 조작은 오브젝트 정의 명령문에 대한 최소한의 구문 분석만 수행하여 새 스키마 이름으로 원래 스키마 이름을 교체합니다. 예를 들어 SQL 프로시저의 컨텐츠 내에 나타나는 원래 스키마의 모든 인스턴스는 새 스키마 이름으로 교체되지 않습니다. 따라서 스키마 복사 조작이 이러한 오브젝

트를 다시 생성하는 데 실패할 수 있습니다. 오류 파일에서 DDL을 사용하여 복사 조작 완료 후 이러한 실패한 오브젝트를 수동으로 다시 작성할 수 있습니다.

오브젝트 간에 상호 종속성

스키마 복사 조작은 이러한 오브젝트 간 상호 종속성을 충족하는 순서대로 오브젝트를 다시 작성합니다. 예를 들어 테이블 T1에 사용자 정의 함수(UDF) U1을 참조하는 컬럼이 있는 경우 T1을 다시 작성하기 전에 먼저 U1을 다시 작성합니다. 그러나 프로시저에 대한 종속성 정보는 카탈로그에서 쉽게 사용할 수 없습니다. 따라서 프로시저를 다시 작성할 때 스키마 복사 조작은 먼저 모든 프로시저를 다시 작성한 다음 실패한 프로시저를 다시 작성하도록 시도합니다. 실패한 프로시저가 이전 시도에서 성공적으로 작성된 프로시저에 종속되어 있다고 가정하면 이후 시도에서 성공적으로 다시 작성됩니다. 후속 시도에서 하나 이상을 성공적으로 다시 작성할 수 있을 때까지 스키마 복사 조작은 실패한 프로시저를 다시 작성하도록 계속해서 시도합니다. 프로시저를 다시 작성하는 모든 시도 중 오류(및 DDL)는 오류 파일에 로그됩니다. 오류 파일에서 동일한 프로시저에 대해 여러 항목을 확인할 수 있더라도 다음 시도에서 해당 프로시저가 성공적으로 다시 작성되었을 수 있습니다. 스키마 복사 조작 완료 시 SYSCAT.PROCEDURES 테이블을 쿼리하여 오류 파일에 나열된 이러한 프로시저가 성공적으로 다시 작성되었는지 판별해야 합니다.

자세한 정보는 ADMIN_COPY_SCHEMA 프로시저 및 db2move 유틸리티를 참조하십시오.

ADMIN_COPY_SCHEMA 프로시저를 사용하는 스키마 사본의 예

아래에 표시된 것처럼 ADMIN_COPY SCHEMA 프로시저를 사용하여 동일한 데이터베이스 내에서 하나의 스키마를 복사합니다.

```
DB2 "SELECT SUBSTR(OBJECT_SCHEMA,1, 8)
  AS OBJECT_SCHEMA, SUBSTR(OBJECT_NAME,1, 15)
  AS OBJECT_NAME, SQLCODE, SQLSTATE, ERROR_TIMESTAMP, SUBSTR(DIAGTEXT,1, 80)
  AS DIAGTEXT, SUBSTR(STATEMENT,1, 80)
  AS STATEMENT FROM COPYERRSCH.COPYERRTAB"

CALL SYSPROC.ADMIN_COPY_SCHEMA('SOURCE_SCHEMA', 'TARGET_SCHEMA',
  'COPY', NULL, 'SOURCETS1 , SOURCETS2', 'TARGETTS1, TARGETTS2,
  SYS_ANY', 'ERRORSCHEMA', 'ERRORNAME')
```

SELECT문의 출력은 다음과 같습니다.

```
OBJECT_SCHEMA OBJECT_NAME     SQLCODE     SQLSTATE ERROR_TIMESTAMP
------------- --------------- ----------- -------- --------------------------
SALES         EXPLAIN_STREAM         -290 55039    2006-03-18-03.22.34.810346

DIAGTEXT
--------------------------------------------------------------------------------
[IBM][CLI Driver][DB2/LINUXX8664] SQL0290N  테이블 스페이스 액세스가 허용되지 않습니다.

STATEMENT
```

```
------------------------------------------------------------------------------
set integrity for "SALES  "."ADVISE_INDEX" , "SALES"."ADVISE_MQT" , "SALES"."

1 레코드가 선택되었습니다.
```

db2move 유틸리티를 사용하는 스키마 사본의 예

-co COPY 조치와 함께 db2move 유틸리티를 사용하여 소스 데이터베이스에서 목표 데이터베이스로 하나 이상의 스키마를 복사합니다. 모델 스키마가 설정되면 이 스키마를 새 버전 작성을 위한 템플리트로 사용할 수 있습니다.

예 1: -c COPY 옵션 사용

db2move -co COPY 옵션의 다음 예에서는 스키마 BAR를 복사하고 샘플 데이터베이스에서 목표 데이터베이스로 해당 스키마의 이름을 FOO로 바꿉니다.

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" -u userid -p password
```

새(목표) 스키마 오브젝트는 소스 스키마의 오브젝트와 동일한 오브젝트 이름을 사용하여 작성되지만 목표 스키마 규정자가 포함되어 있습니다. 소스 테이블의 데이터가 포함되는지 여부와 관계없이 테이블 사본을 작성할 수 있습니다. 소스 및 목표 데이터베이스는 다른 시스템에 있을 수 있습니다.

예 2: COPY 조작 중 테이블 스페이스 이름 맵핑 지정

다음 예에서는 db2move COPY 조작 중 소스 시스템의 테이블 스페이스 대신 사용될 특정 테이블 스페이스 이름 맵핑을 지정하는 방법을 보여줍니다. 목표 테이블 스페이스가 디폴트 테이블 스페이스 선택 알고리즘을 사용하여 선택되어야 함을 표시하는 SYS_ANY 키워드를 지정할 수 있습니다. 이 경우 db2move 유틸리티는 다음과 같이 목표로 사용할 수 있는 모든 사용 가능한 테이블 스페이스를 선택합니다.

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" tablespace_map "(SYS_ANY)" -u userid -p password
```

모든 테이블 스페이스에 SYS_ANY 키워드를 사용할 수 있습니다. 또는 사용자는 다음과 같이 일부 테이블 스페이스에 대해 특정 맵핑을 지정하고 나머지에 대해서는 디폴트 테이블 스페이스 선택 알고리즘을 지정할 수 있습니다.

```
db2move sample COPY -sn BAR -co target_db  target schema_map "
  ((BAR,FOO))" tablespace_map "((TS1, TS2),(TS3, TS4), SYS_ANY)"
  -u userid -p password
```

이것은 테이블 스페이스 TS1이 TS2로 맵핑되고, TS3은 TS4에 맵핑되지만 나머지 테이블 스페이스는 디폴트 테이블 스페이스 선택 알고리즘을 사용함을 나타냅니다.

예 3: COPY 조작 후 오브젝트 소유자 변경

사용자가 성공적인 COPY 후에 목표 스키마에 작성된 각 새 오브젝트의 소유자를 변경할 수 있습니다. 목표 오브젝트의 기본 소유자는 연결 사용자입니다. 이 옵션을 지정하면 아래와 같이 소유권이 새 소유자에게 이전됩니다.

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" tablespace_map "(SYS_ANY)" owner jrichards
  -u userid -p password
```

목표 오브젝트의 새 소유자는 jrichards입니다.

소스 및 목표 스키마가 다른 시스템 상주하고 있는 경우 목표 시스템에서 db2move 유틸리티가 호출되어야 합니다. 하나의 데이터베이스에서 다른 데이터베이스로 스키마를 복사하려면 이 조치에는 소스 데이터베이스에서 복사되는 스키마 이름 목록(쉼표로 구분됨)과 목표 데이터베이스 이름이 필요합니다.

스키마를 복사하려면 OS 명령 프롬프트에서 다음과 같이 db2move를 발생하십시오.

```
db2move <dbname> COPY -co <COPY- options>
 -u <userid> -p <password>
```

실패한 스키마 복사 조작 재시작

db2move COPY 조작 중 발생한 오류는 복사된 오브젝트의 유형 또는 COPY 조작에 실패한 단계(즉, 오브젝트 다시 작성 단계 또는 데이터 로드 단계)에 따라 여러 가지 방법으로 처리될 수 있습니다.

db2move 유틸리티는 메시지 및 오류 파일을 사용하여 사용자에게 오류 및 메시지를 보고합니다. 스키마 복사 조작은 COPYSCHEMA_<timestamp>.MSG 메시지 파일 및 COPYSCHEMA_<timestamp>.err 오류 파일을 사용합니다. 이러한 파일은 현재 작업 디렉토리에 작성됩니다. 파일의 고유성을 보장하도록 현재 시간이 파일 이름에 추가됩니다. 사용자의 필요에 따라 이러한 메시지 및 오류 파일이 더 이상 필요하지 않으면 삭제할 수 있습니다.

주: 여러 db2move 인스턴스를 동시에 실행할 수 있습니다. COPY 옵션은 어떠한 SQLCODES도 리턴하지 않습니다. 이 옵션은 db2move 동작과 일치합니다.

오브젝트 유형

복사된 오브젝트 유형을 실제 오브젝트 및 비즈니스 오브젝트 중 하나로 구분할 수 있습니다.

실제 오브젝트는 컨테이너(예: 테이블, 인덱스 및 사용자 정의 구조화된 유형)에 실제로 상주하는 오브젝트를 참조합니다. 비즈니스 오브젝트는 뷰와 같은 컨테이너(예: 뷰, 사용자 정의 구조화된 유형(UDT) 및 별명)에 상주하지 않는 카탈로그된 오브젝트를 참조합니다.

실제 오브젝트를 다시 작성하는 중 발생한 오류로 인해 유틸리티가 롤백될 수 있습니다. 반면에 논리적 오브젝트를 다시 작성하는 중 발생한 오류로 인해서는 유틸리티가 롤백되지 않습니다.

스키마 복사 조작 재시작

오류 파일에 설명된 로드 조작 실패로 인해 발생한 문제를 처리한 후 다음 구문에서 표시된 것처럼 LOADTABLE.err 파일 이름을 통해 전달되는 데이터를 복사하여 해당 데이터로 채울 테이블을 지정하도록 -tf 옵션을 사용하여 db2move -COPY 명령을 다시 발행할 수 있습니다.

```
db2move sourcedb COPY -tf LOADTABLE.err -co TARGET_DB mytarget_db
  -mode load_only
```

또한 다음 구문에 표시된 것처럼 -tn 옵션을 사용하여 테이블 이름을 입력할 수도 있습니다.

```
db2move sourcedb COPY -tn "FOO"."TABLE1","FOO 1"."TAB 444",
  -co TARGET_DB mytarget_db -mode load_only
```

주: load_only 모드는 -tn 또는 -tf 옵션을 사용하여 최소 하나의 테이블을 입력해야 합니다.

예

db2move COPY 스키마 조작 중 발생한 오류는 복사된 오브젝트의 유형 또는 COPY 조작 실패 단계에 따라 여러 가지 방법으로 처리될 수 있습니다.

db2move 유틸리티는 다음과 같이 메시지 및 오류 파일의 스키마 복사 오류 및 메시지를 보고합니다.

- COPYSCHEMA <timestamp>.MSG 메시지 파일
- COPYSCHEMA_<timestamp>.err 오류 파일

이러한 파일은 현재 작업 디렉토리에서 작성됩니다. 파일의 고유성을 보장하도록 현재 시간이 파일 이름에 추가됩니다. 이러한 메시지 및 오류 파일이 더 이상 필요하지 않으면 삭제해야 합니다.

주: 여러 db2move 인스턴스를 동시에 실행할 수 있습니다. COPY 옵션은 어떠한 SQLCODES도 리턴하지 않습니다. 이 옵션은 db2move 동작과 일치합니다.

예 1: 실제 오브젝트와 관련된 스키마 복사 오류

목표 데이터베이스에서 실제 오브젝트 다시 작성 중 발생한 실패는 오류 파일 COPYSCHEMA_<timestamp>.err에 로그됩니다. 실패한 각 오브젝트에 대해 오류 파일에는 오브젝트 이름, 오브젝트 유형, DDL 텍스트, 시간소인 및 sqlca로 형식화된 문자열(뒤에 데이터 값이 따라오는 sqlca 파일 이름)과 같은 정보가 포함되어 있습니다.

COPYSCHEMA_<timestamp>.err 오류 메시지의 샘플 출력은 다음과 같습니다.

```
1. schema: FOO.T1
Type:      TABLE
Error Msg: SQL0104N An unexpected token 'FOO.T1'...
```

```
Timestamp: 2005-05-18-14.08.35.65
DDL:       create view FOO.v1

2. schema:  FOO.T3
Type:      TABLE
Error Msg: SQL0204N FOO.V1 is an undefined name.
Timestamp: 2005-05-18-14.08.35.68
DDL:       create table FOO.T3
```

실제 오브젝트 작성 중 발생한 오류가 다시 작성 단계의 마지막 부분 및 로드 단계 이전에서 로그되면 db2move 유틸리티가 실패하고 오류가 리턴됩니다. 목표 데이터베이스에 대한 모든 오브젝트 작성이 롤백되고 내부적으로 작성된 모든 테이블이 소스 데이터베이스에서 제거됩니다. 가능한 모든 오류를 오류 파일에 수집하기 위해 처음 실패 이후가 아니라 각 오브젝트를 다시 작성하도록 시도한 후 다시 작성 단계의 마지막 부분에서 롤백이 발생합니다. 따라서 db2move 조작을 재시작하기 전에 모든 문제점을 수정할 수 있습니다. 실패하지 않으면 오류 파일이 삭제됩니다.

예 2: 비즈니스 오브젝트와 관련된 스키마 복사 오류

목표 데이터베이스에서 비즈니스 오브젝트 다시 작성에 실패하더라도 db2move 유틸리티에 실패하지 않습니다. 대신 이러한 실패는 COPYSCHEMA_<timestamp>.err 오류 파일에 로그됩니다. db2move 유틸리티 완료 시 실패를 살펴보고, 모든 문제를 처리하고, 실패한 각 오브젝트를 수동으로 다시 작성할 수 있습니다. 사용자의 편의를 위해 오류 파일에 DDL이 제공됩니다.

db2move가 로드 유틸리티를 사용하여 테이블 데이터를 다시 채우려고 하는 중에 오류가 발생하면 db2move 유틸리티에 실패하지 않습니다. 대신 일반 실패 정보(오브젝트 이름, 오브젝트 유형, DDL 텍스트, 시간소인, sqlca 등)가 COPYSCHEMA_<timestamp>.err 파일에 로그되고 다른 파일인 LOADTABLE_<timestamp>.err에 테이블의 완전한 이름이 로그됩니다. db2move -tf 옵션 형식을 충족하도록 다음과 유사하게 각 테이블이 행당 나열됩니다.

```
"FOO"."TABLE1"
"FOO 1"."TAB 444"
```

예 3: db2move 실패의 다른 유형

내부 조작(예: 메모리 오류 및 파일 시스템 오류)으로 인해 db2move 유틸리티가 실패할 수 있습니다.

ddl 다시 작성 단계에서 내부 조작 오류가 발생하면 성공적으로 작성된 모든 오브젝트가 목표 스키마에서 롤백되고 내부적으로 작성된 모든 테이블(예: DMT 테이블 및 db2look 테이블)이 소스 데이터베이스에서 제거됩니다.

로드 단계 중 내부 조작 오류가 발생하면 성공적으로 작성된 모든 오브젝트가 목표 스키마에 남아 있습니다. 로드 조작 중 실패가 발생한 모든 테이블 및 아직 로드되지 않은 모든 테이블은 LOADTABLE.err 오류 파일에 로그됩니다. 예

2에서 설명한 것처럼 LOADTABLE.err을 사용하여 db2move COPY 명령을 발행할 수 있습니다. db2move 유틸리티가 이상 종료되면(예: 시스템 손상, 유틸리티 트랩, 유틸리티 종료 등) 로드되어야 하는 테이블에 대한 정보가 손상됩니다. 이러한 경우 ADMIN_DROP_SCHEMA 프로시저를 사용하여 목표 스키마를 제거하고 db2move COPY 명령을 다시 발행할 수 있습니다.

스키마 복사 조작 시도 중 발생한 오류에 관계없이 항상 ADMIN_DROP_SCHEMA 프로시저를 사용하여 목표 스키마를 삭제하고 db2move COPY 명령 다시 발행할 수 있습니다.

스키마 삭제

스키마를 삭제하기 전에 해당 스키마에 있는 모든 오브젝트를 삭제하거나 다른 스키마로 이동해야 합니다. DROP문을 실행할 때 스키마 이름은 카탈로그에 있어야 합니다. 그렇지 않으면 오류가 리턴됩니다.

명령행을 사용하여 스키마를 삭제하려면 다음과 같이 입력하십시오.

```
DROP SCHEMA <name> RESTRICT
```

다음 예에서는 스키마 "joeschma"가 삭제됩니다.

```
DROP SCHEMA joeschma RESTRICT
```

RESTRICT 키워드는 스키마가 데이터베이스에서 삭제되도록 지정된 스키마에 오브젝트를 정의할 수 없고 해당 스키마를 지정해야 한다는 규칙을 적용합니다.

제 3 부 데이터베이스 오브젝트

논리 데이터베이스 설계는 데이터베이스 오브젝트 정의로 구성됩니다.

다음과 같은 데이터베이스 오브젝트를 DB2 데이터베이스에 작성할 수 있습니다.

- 테이블
- 제한조건
- 인덱스
- 트리거
- 시퀀스
- 뷰

그래픽 사용자 인터페이스를 사용하거나 명시적으로 명령문을 실행하여 이와 같은 데이터베이스 오브젝트를 작성할 수 있습니다. 이러한 데이터베이스 오브젝트를 작성하는 데 사용되는 명령문을 데이터 정의 언어(DDL) 명령문이라고 하며 일반적으로 CREATE 또는 ALTER 키워드가 앞에 붙습니다.

시간이 지남에 따라 확장 및 성장을 수용할 수 있는 충분한 유연성을 유지하면서 현재 비즈니스의 데이터 스토리지 요구사항에 부합하는 데이터베이스 설계를 구현하기 위해서는 이들 각각의 데이터베이스 오브젝트가 제공하는 특성 및 기능을 이해하는 것이 중요합니다.

제 10 장 대부분의 데이터베이스 오브젝트의 일반적인 개념

별명

*별명*은 모듈, 테이블 또는 다른 별명과 같은 오브젝트의 대체 이름입니다. 해당 오브젝트를 바로 참조할 수 있는 위치 어디서나 오브젝트를 참조하는 데 별명을 사용할 수 있습니다.

일부 컨텍스트에서는 별명을 사용할 수 없습니다. 예를 들어, 점검 제한조건의 점검 조건에서는 별명을 사용할 수 없습니다. 별명은 선언된 임시 테이블을 참조할 수 없지만 작성된 임시 테이블을 참조할 수는 있습니다.

기타 오브젝트와 마찬가지로 별명을 작성하거나 삭제할 수 있으며 별명과 연관된 주석이 있을 수 있습니다. 순환 참조가 없는 한 별명은 *연결*이라는 프로세스에 있는 기타 별명을 참조할 수 있습니다. 별명을 사용하기 위해 특수한 권한 또는 특권이 필요하지 않습니다. 그러나 별명을 통해 참조하는 오브젝트에 액세스하려면 해당 오브젝트와 연관된 권한이 필요합니다.

별명이 공용 별명으로 정의된 경우 현재 디폴트 스키마 이름의 영향을 받지 않고 규정되지 않은 이름으로 별명을 참조할 수 있습니다. SYSPUBLIC 규정자를 사용하여 참조할 수도 있습니다.

*동의어*는 별명의 대체 이름입니다.

자세한 정보는 *SQL 참조서, 볼륨 1*의 "ID의 별명"을 참조하십시오.

데이터베이스 오브젝트의 소프트 무효화

*소프트 무효화*가 활성화되면 기타 실행 중인 트랜잭션에서 오브젝트를 사용 중이어도 오브젝트를 삭제할 수 있습니다. 삭제된 오브젝트를 사용 중이던 트랜잭션은 계속 수행하도록 허용되지만 새 트랜잭션이 삭제된 오브젝트에 액세스하는 것은 거부됩니다.

삭제 또는 변경 중인 오브젝트를 직/간접적으로 참조하는 모든 캐시된 명령문 및 패키지는 유효하지 않은 것으로 표시되며 *무효화*되었다고 합니다. 소프트 무효화는 참조되는 오브젝트에 DDL이 영향을 미치는 것을 허용하여 참조하는 오브젝트에 대한 잠금 보류를 실행 중인 명령문으로 인해 대기하지 않도록 하며, 캐시된 버전의 오브젝트를 사용하여 활성 상태인 오브젝트가 계속 사용되도록 허용함으로써 잠금 시간종료가 발생할 가능성을 제거합니다.

이에 비하여 *하드 무효화*를 사용하는 경우에는 오브젝트 참조 시 배타적 잠금이 사용됩니다. 이는 모든 프로세스에서 동일한 버전의 오브젝트를 사용하고 삭제된 오브젝트에는 액세스할 수 없도록 보장합니다.

소프트 무효화는 **DB2_DDL_SOFT_INVAL** 레지스트리 변수를 통해 사용 가능해집니다. 디폴트로, 이 레지스트리 변수는 ON으로 설정되어 있습니다.

다음 목록은 소프트 무효화가 지원되는 데이터 정의 언어(DDL)문을 표시합니다.

- CREATE OR REPLACE ALIAS
- CREATE OR REPLACE FUNCTION
- CREATE OR REPLACE TRIGGER
- CREATE OR REPLACE VIEW
- DROP ALIAS
- DROP FUNCTION
- DROP TRIGGER
- DROP VIEW

소프트 무효화 지원은 커서 안정성(CS) 및 언커미트된 읽기(UR) 분리 레벨에서 수행된 스캔과 동적 SQL에만 적용됩니다.

예

VIEW1이라는 뷰가 존재한다고 가정합니다. 커서를 열고 명령문 `SELECT * from VIEW1`을 실행합니다. 실행 즉시 데이터베이스 관리자가 `DROP VIEW VIEW1` 명령을 실행하여 데이터베이스에서 VIEW1을 삭제(drop)합니다. 하드 무효화를 사용하면 DROP VIEW 문이 SELECT 트랜잭션이 완료될 때까지 강제로 VIEW1에 대한 배타적 잠금을 기다립니다. 소프트 무효화를 사용하면 DROP VIEW문에 뷰에 대한 배타적 잠금이 지정되지 않습니다. 그러나 뷰는 삭제되고 뷰의 최근 정의를 사용하여 SELECT문은 계속 실행됩니다. SELECT문이 완료된 후에는 그 후로 VIEW1을 사용하려 하면 이전에 해당 뷰를 사용한 사용자 또는 프로세스에서 사용하려는 경우에도 오류가 발생합니다 (SQL0204N).

데이터베이스 오브젝트의 자동 유효성 다시 확인

*자동 유효성 다시 확인*은 유효성이 확인된 데이터베이스 오브젝트(예: DROP문 이후)의 유효성을 자동으로 다시 확인하는 메커니즘입니다.

일반적으로 데이터베이스 관리 프로그램은 다음에 유효하지 않은 오브젝트를 사용하려고 할 때 해당 오브젝트의 유효성을 다시 확인하려 합니다. 자동 유효성 다시 확인은 **auto_reval** 레지스트리 변수를 통해 사용 가능하게 설정됩니다. 디폴트로, 이 레지스트

리 변수는 DEFERRED로 설정되지만 버전 9.5 이전에서 업그레이드된 데이터베이스의 경우에는 **auto_reval**이 DISABLED로 설정됩니다.

오브젝트 삭제 시 영향을 받는 종속 오브젝트에 대한 정보 및 해당 종속 오브젝트 유효성을 다시 확인하는 시기는 *SQL 참조서, 볼륨 1*의 『DROP문』을 참조하십시오.

다음 목록은 현재 자동 유효성 다시 확인이 지원되는 데이터 정의 언어(DDL)문을 표시합니다.

- ALTER MODULE DROP FUNCTION
- ALTER MODULE DROP PROCEDURE
- ALTER MODULE DROP TYPE
- ALTER MODULE DROP VARIABLE
- ALTER NICKNAME(로컬 이름 또는 로컬 유형 변경)
- ALTER TABLE ALTER COLUMN
- ALTER TABLE DROP COLUMN
- ALTER TABLE RENAME COLUMN
- CREATE OR REPLACE ALIAS
- CREATE OR REPLACE FUNCTION
- CREATE OR REPLACE NICKNAME
- CREATE OR REPLACE PROCEDURE
- CREATE OR REPLACE SEQUENCE
- CREATE OR REPLACE TRIGGER
- CREATE OR REPLACE VARIABLE
- CREATE OR REPLACE VIEW
- DROP FUNCTION
- DROP NICKNAME
- DROP PROCEDURE
- DROP SEQUENCE
- DROP TABLE
- DROP TRIGGER
- DROP TYPE
- DROP VARIABLE
- DROP VIEW
- RENAME TABLE

데이터베이스 오브젝트 작성 및 유지보수

몇 가지 유형의 데이터베이스 오브젝트를 작성하는 경우 REPLACE 옵션 및 오류 지원 기능을 갖춘 CREATE에 대해 알고 있어야 합니다.

특정 데이터베이스 오브젝트에 대한 오류가 지원되는 기능을 갖춘 CREATE

컴파일하는 중에 오류가 발생하더라도 일부 유형의 오브젝트를 작성할 수 있습니다 (예: 참조되는 테이블이 없는 경우 뷰 작성).

이러한 오브젝트는 액세스될 때까지 유효하지 않은 상태로 남아 있습니다. 오류 지원 기능을 갖춘 CREATE는 현재 뷰 및 인라인 SQL 함수(컴파일된 함수 아님)까지 확장됩니다. **auto_reval** 데이터베이스 구성 매개변수가 IMMEDIATE 또는 DEFERRED로 설정된 경우 이러한 기능을 사용할 수 있습니다.

오브젝트 작성 중 허용되는 오류는 다음과 같은 유형으로 제한됩니다.

- 이름 분석 오류: 참조 테이블이 없음(SQLSTATE 42704, SQL0204N), 참조 컬럼이 없음(SQLSTATE 42703, SQL0206N) 또는 참조 함수가 없음(SQLSTATE 42884, SQL0440N)
- 중첩된 유효성 다시 확인 실패. 작성되는 오브젝트가 유효하지 않은 오브젝트를 참조할 수 있으므로 유효하지 않은 이러한 오브젝트에 대한 유효성 다시 확인이 호출됩니다. 참조된 유효하지 않은 모든 오브젝트의 유효성을 다시 확인하는 데 실패하면 CREATE문에 성공하고 작성된 오브젝트는 다음에 액세스될 때까지 유효하지 않은 상태로 남아 있습니다.
- 권한 부여 오류(SQLSTATE 42501, SQL0551N)

본문에 여러 오류가 있더라도 오브젝트가 작성될 수 있습니다. 리턴된 경고 메시지에는 컴파일 중 발생한 처음으로 정의되지 않은 오브젝트, 유효하지 않은 오브젝트 또는 권한이 부여되지 않은 오브젝트의 이름이 포함되어 있습니다.
SYSCAT.INVALIDOBJECTS 카탈로그 뷰에는 유효하지 않은 오브젝트에 대한 정보가 포함되어 있습니다.

예

```
create view v2 as select * from v1
```

v1이 없으면 CREATE VIEW문은 완료되지만 v2는 유효하지 않은 상태로 남아 있습니다.

여러 CREATE문에 대한 REPLACE 옵션

별명, 함수, 모듈, 별칭, 프로시저(페더레이티드 프로시저 포함), 시퀀스, 트리거, 변수 및 뷰를 비롯한 여러 오브젝트에 절 대한 CREATE문의 **OR REPLACE**절은 오브젝트

가 이미 있으면 오브젝트가 대체되도록 허용하고 그렇지 않으면 오브젝트가 작성됩니다. 따라서 데이터베이스 스키마를 변경하는 데 필요한 수고가 크게 줄어듭니다.

오브젝트에 이전에 부여된 특권은 해당 오브젝트가 교체되는 경우 보존됩니다. 다른 측면에서 CREATE OR REPLACE는 CREATE가 뒤따르는 DROP과 의미상 유사합니다. 함수, 프로시저 및 트리거의 경우 인라인 오브젝트 및 컴파일된 오브젝트 모두에 지원이 적용됩니다.

함수 및 프로시저의 경우 SQL 및 외부 함수와 프로시저 모두에 지원이 적용됩니다. 모듈이 대체되면 해당 모듈 내의 모든 오브젝트가 삭제됩니다. 새 버전의 모듈에는 오브젝트가 없습니다.

교체된 오브젝트에 직접 또는 간접적으로 종속된 오브젝트는 무효화됩니다. **auto_reval** 데이터베이스 구성 매개변수가 DISABLED로 설정되어 있더라도 교체 조작 뒤에 이어지는 모든 종속 오브젝트의 유효성 다시 확인은 무효화된 후 바로 완료됩니다.

예

종속 오브젝트가 있는 뷰 v1을 교체합니다.

```
create table t1 (c1 int, c2 int);
create table t2 (c1 int, c2 int);

create view v1 as select * from t1;
create view v2 as select * from v1;

create function foo1()
 language sql
 returns int
 return select c1 from v2;

create or replace v1 as select * from t2;

select * from v2;

values foo1();
```

v1의 교체된 버전은 t1 대신 t2를 참조합니다. v2 및 foo1은 모두 CREATE OR REPLACE문에 의해 무효화됩니다. 의미상 유효성 다시 확인이 지연된 경우 `select * from v2`는 v2의 유효성을 다시 확인하지만 foo1() 값에 의해 유효성이 다시 확인되는 foo1의 유효성은 다시 확인하지 않습니다. 의미상 유효성이 즉시 다시 확인되는 경우 CREATE OR REPLACE문에 의해 v2 및 foo1의 유효성이 모두 다시 확인됩니다.

제 11 장 테이블

테이블은 데이터베이스 관리 프로그램에서 유지보수하는 논리적 구조입니다. 테이블은 컬럼과 행으로 구성됩니다.

모든 컬럼과 행의 교차에 *값*이라고 부르는 특정 데이터 항목이 있습니다. *컬럼*은 동일한 유형 또는 부속 유형 중 하나의 값 세트입니다. *행*은 *n*번째 값이 테이블의 *n*번째 컬럼의 값이 되도록 정렬되는 값의 시퀀스입니다.

응용프로그램은 행이 테이블에 채워지는 순서를 판별할 수 있지만, 행의 실제 순서는 데이터베이스 관리 프로그램에 의해 판별되며 일반적으로 제어할 수 없습니다. 다차원 클러스터링(MDC)이 어느 정도 클러스터링을 제공하지만 행 사이의 실제 순서 지정을 제공하지는 않습니다.

테이블 유형

DB2 데이터베이스는 데이터를 테이블에 저장합니다. 지속적 데이터를 저장하는 데 사용되는 테이블 외에, 결과 표시에 사용되는 테이블, 요약 테이블 및 임시 테이블도 있습니다. 다차원적으로 클러스터된(MDC) 테이블은 웨어하우스 환경에서 특정한 장점을 제공하는 반면, 파티션된 테이블은 둘 이상의 데이터베이스 파티션 사이에 데이터를 분산할 수 있습니다.

기본 테이블

이 테이블 유형은 지속적 데이터를 보유합니다. 다음을 포함한 여러 가지 종류의 기본 테이블이 있습니다.

일반 테이블

인덱스를 갖는 일반 테이블은 "범용" 테이블 선택사항입니다.

다차원적으로 클러스터된(MDC) 테이블

이런 유형의 테이블은 둘 이상의 키 또는 차원에서 동시에 물리적으로 클러스터되는 테이블로서 구현됩니다. MDC 테이블은 데이터 웨어하우징 및 대형 데이터베이스 환경에서 사용됩니다. 일반 테이블에 있는 클러스터링 인덱스는 일차원 데이터 클러스터링을 지원합니다. MDC 테이블은 2차원 이상에서의 데이터 클러스터링을 지원합니다. MDC 테이블은 복합 차원 내부에서 *보증된 클러스터링*을 제공합니다. 대조적으로, 일반 테이블에서 클러스터된 인덱스를 가질 수 있지만 이 경우에 클러스터링은 데이터베이스 관리 프로그램에 의해 시도되지만 보증되지 않으며 일반적으로 시간에 따라 저하됩니다. MDC 테이블은 파티션된 테이블과 공존할 수 있으며 그 자체가 파티션된 테이블일 수 있습니다.

범위 클러스터 테이블(RCT)

이런 유형의 테이블은 빠른 직접 액세스를 제공하는 데이터의 순차 클러스터로서 구현됩니다. 테이블의 각 레코드는 테이블의 레코드를 찾는 데 사용되는 내부 ID인 사전 결정된 레코드 ID(RID)를 갖습니다. RCT 테이블은 테이블에 있는 하나 이상의 컬럼에서 데이터가 단단하게 클러스터되어 있는 경우에 사용합니다. 컬럼에 있는 가장 큰 값과 가장 작은 값은 가능한 값의 범위를 정의합니다. 이러한 컬럼을 사용하여 테이블의 레코드에 액세스할 수 있습니다. 이것이 RCT 테이블의 사전 결정된 레코드 ID(RID) 측면을 이용하는 최적 방법입니다.

임시 테이블

이런 유형의 테이블은 다양한 데이터베이스 조작에 대한 임시 작업 테이블로 사용됩니다. *선언된 임시 테이블*(DGTT)은 시스템 카탈로그에 나타나지 않으며, 이것은 해당 테이블을 지속적으로 사용하지 못하고 다른 응용프로그램과 공유될 수 없게 만듭니다. 이 테이블을 사용하는 응용프로그램이 데이터베이스에서 종료되거나 연결이 끊어질 때 테이블의 모든 데이터가 삭제되며 테이블이 삭제됩니다. 대조적으로 *작성된 임시 테이블*(CGTT)은 시스템 카탈로그에 나타나며 해당 테이블이 사용되는 모든 세션에서 정의될 필요가 없습니다. 결국 이들 테이블은 지속적이며 여러 연결 사이에서 기타 응용프로그램과 공유될 수 있습니다.

어떤 유형의 임시 테이블도 다음을 지원하지 않습니다.

- 사용자 정의 참조 또는 사용자 정의 구조화된 유형 컬럼
- LONG VARCHAR 컬럼

또한 XML 컬럼은 작성된 임시 테이블에서 사용할 수 없습니다.

구체화된 쿼리 테이블

이러한 유형의 테이블은 테이블의 데이터를 판별하는 데도 사용되는 쿼리로 정의됩니다. 이 구체화된 쿼리 테이블을 사용하면 쿼리 성능을 높일 수 있습니다. 데이터베이스 관리 프로그램이 쿼리의 일부가 요약 테이블을 사용하여 해결될 수 있다고 판별할 경우, 데이터베이스 관리 프로그램은 요약 테이블을 사용하도록 쿼리를 재작성할 수 있습니다. 이 결정은 CURRENT REFRESH AGE 및 CURRENT QUERY OPTIMIZATION 특수 레지스터 같은 데이터베이스 구성 설정을 기초로 합니다. 요약 테이블은 구체화된 쿼리 테이블의 특수한 유형입니다.

CREATE TABLE문을 사용하여 앞의 모든 유형의 테이블을 작성할 수 있습니다.

데이터 유형에 따라 한 가지 테이블 유형이 스토리지 및 쿼리 성능을 최적화할 수 있는 특정 성능을 제공할 수 있음을 알 수 있습니다. 예를 들어 느슨하게 클러스터되는(일정하게 증가하지 않음) 데이터 레코드의 경우에는 일반 테이블 및 인덱스를 사용하도록 고려하십시오. 데이터 레코드가 키에 중복(고유하지 않은) 값을 갖는 경우 범위 클러스

터 테이블(RCT)을 사용해서는 안됩니다. 또한 사용하려는 범위 클러스터 테이블(RCT)용 디스크에 일정량의 스토리지를 미리 할당할 수 없는 경우 이런 유형의 테이블을 사용해서는 안됩니다. 지리적 영역, 부서 및 공급자별로 소매 매출을 추적하는 테이블 같이 다중 차원을 따라 클러스터될 가능성이 있는 데이터가 있는 경우, 다차원적으로 클러스터된(MDC) 테이블이 목적에 부합할 수 있습니다.

위에서 설명한 다양한 테이블 유형 외에, 테이블 데이터 롤인 같은 태스크에 대한 성능을 향상시킬 수 있는 *파티셔닝* 같은 특성에 대한 옵션도 있습니다. 파티션된 테이블은 또한 일반 파티션되지 않은 테이블보다 훨씬 더 많은 정보를 보유할 수 있습니다. 또한 데이터 스토리지 비용을 상당히 줄이는 데 도움이 되는 압축 같은 기능을 이용할 수도 있습니다.

테이블 디자인

테이블을 디자인할 때 특정 개념에 익숙하고 테이블 및 사용자 데이터에 대한 스페이스 요구사항을 판별하고 압축 및 낙관적 잠금 같은 특정 기능을 활용할지 여부를 판별해야 합니다.

파티션된 테이블을 디자인할 때 다음과 같은 파티셔닝 개념에 친숙해야 합니다.

- 데이터 구성 스킴
- 테이블 파티셔닝 키
- 데이터 파티션 사이에 데이터 분배에 사용되는 키
- MDC 차원에 사용되는 키

이들 및 기타 파티셔닝 개념에 대해서는 298 페이지의 『테이블 파티션 및 데이터 구성 스킴』의 개념을 참조하십시오.

테이블 설계 개념

테이블을 설계할 때 몇 가지 관련 개념에 친숙해야 합니다.

데이터 유형 및 테이블 컬럼

테이블을 작성할 때 각 컬럼이 저장할 데이터의 유형을 표시해야 합니다. 관리하려는 데이터의 특성을 고려함으로써, 최적 쿼리 성능을 제공하고 물리적 스토리지 요구사항을 최소화하고 숫자 데이터에 대한 산술 연산이나 날짜 또는 시간 값을 서로 비교하는 등의 여러 가지 종류의 데이터 처리를 위한 특수 기능을 제공하는 방식으로 테이블을 설정할 수 있습니다.

262 페이지의 그림 31은 DB2 데이터베이스가 지원하는 데이터 유형을 나타냅니다.

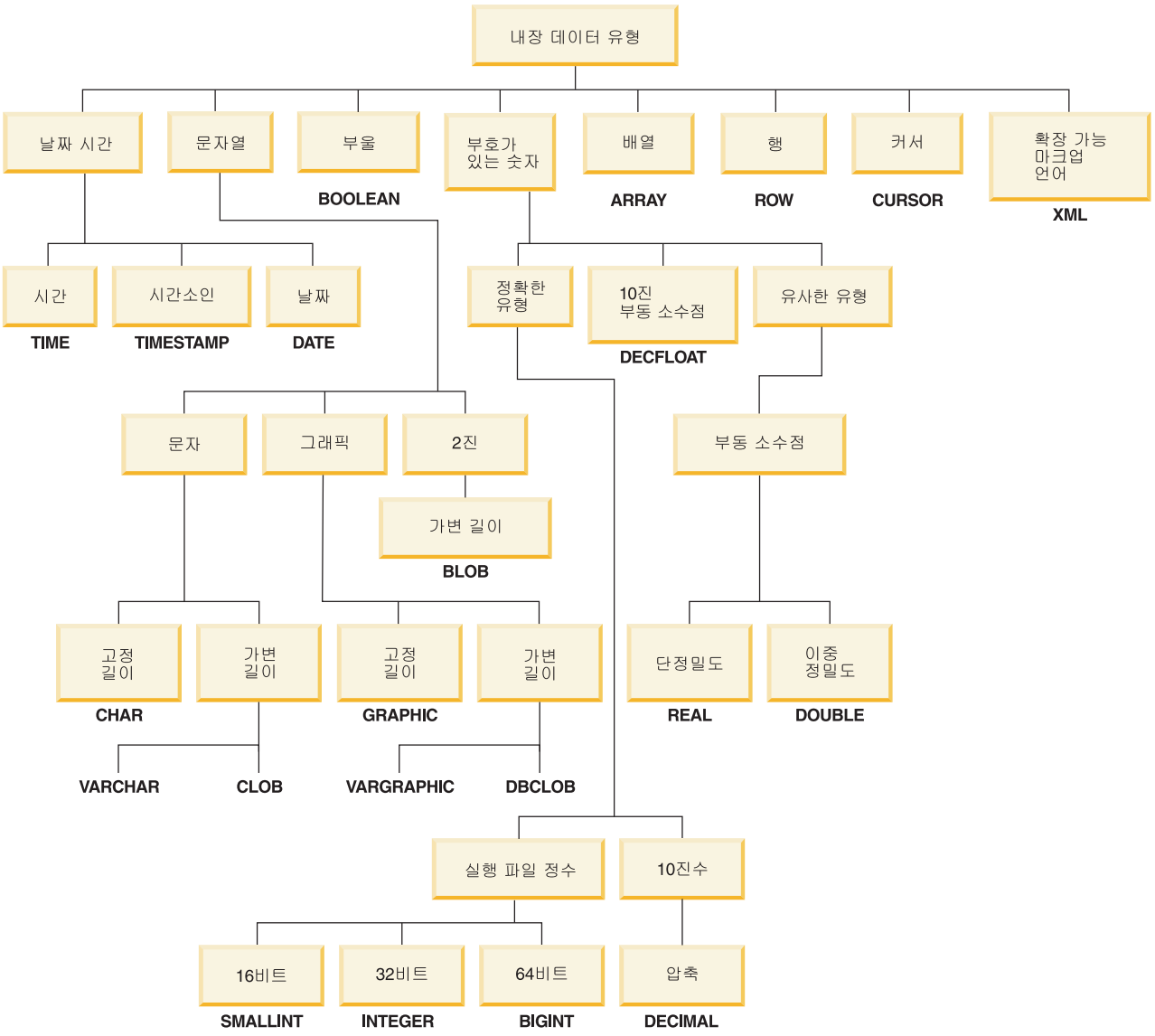


그림 31. 내장 데이터 유형

데이터베이스 컬럼을 선언할 때 이러한 모든 데이터 유형에서 선택할 수 있습니다. 내장 유형 외에, 내장 유형을 기초로 하는 사용자 고유의 *사용자 정의* 데이터 유형을 작성할 수도 있습니다. 예를 들어 이름, 직위, 작업 레벨, 고용일 및 급여 속성을 갖는 직원을 VARCHAR(이름, 직위), SMALLINT(작업 레벨), DATE(고용일) 및 DECIMAL(급여) 데이터를 통합하는 사용자 정의 *구조화된* 유형으로 나타낼 수 있습니다.

생성된 컬럼

생성된 컬럼은 저장된 값이 삽입 또는 갱신 조작을 통해 지정되기 보다는 표현식을 사용하여 계산되는 테이블에 정의됩니다.

특정 표현식 또는 술어가 항상 사용될 것으로 알려진 테이블을 작성할 때, 하나 이상의 생성된 컬럼을 해당 테이블에 추가할 수 있습니다. 생성된 컬럼을 사용하여 테이블 데이터를 쿼리할 때 성능을 향상시킬 수 있습니다.

예를 들어, 표현식의 평가로 성능이 저하되는 두 가지 경우가 있습니다.

1. 표현식의 평가가 쿼리 중 여러 번 수행된 경우
2. 계산이 복잡합니다.

쿼리 성능을 개선시키기 위해, 표현식의 결과를 포함하는 추가 컬럼을 정의할 수 있습니다. 그런 다음, 동일한 표현식을 포함하는 쿼리를 발행할 때, 생성되는 컬럼이 직접 사용될 수 있거나, 옵티마이저의 쿼리 재작성 구성요소가 표현식을 생성된 컬럼으로 바꿀 수 있습니다.

쿼리가 둘 이상의 테이블의 데이터의 조인에 관련된 곳에서 생성된 컬럼을 추가하면, 옵티마이저가 가능한 더 나은 조인 전략을 선택할 수 있습니다.

생성된 컬럼을 사용하여 쿼리 성능을 높일 수 있습니다. 결과적으로, 생성된 컬럼은 테이블이 작성되어 데이터가 채워진 후에 추가될 수 있습니다.

예

다음은 CREATE TABLE문에서 생성된 컬럼을 정의하는 예입니다.

```
CREATE TABLE t1 (c1 INT,
                                    c2 DOUBLE,
                                    c3 DOUBLE GENERATED ALWAYS AS (c1 + c2)
                                    c4 GENERATED ALWAYS AS
                  (CASE WHEN c1 > c2 THEN 1 ELSE NULL END))
```

이 테이블을 작성한 후, 인덱스는 생성된 컬럼을 사용하여 작성될 수 있습니다. 예를 들면 다음 선언을

```
CREATE INDEX i1 ON t1(c4)
```

쿼리는 생성된 컬럼의 이점을 사용할 수 있습니다. 예를 들면, 다음과 같습니다.

```
SELECT COUNT(*) FROM t1 WHERE c1 > c2
```

를 다음과 같이 쓸 수 있습니다.

```
 SELECT COUNT(*) FROM t1 WHERE c4 IS NOT NULL
```

다른 예의 경우를 보십시오.

```
SELECT c1 + c2 FROM t1 WHERE (c1 + c2) * c1 > 100
```

은 다음과 같이 쓸 수 있습니다.

```
SELECT c3 FROM t1 WHERE c3 * c1 > 100
```

자동 번호 지정 및 ID 컬럼

ID 컬럼은 테이블에 추가되는 각 행의 고유 숫자 값을 자동으로 생성하는 방법을 DB2에 제공합니다.

테이블에 추가될 각 행을 고유하게 식별해야 하는 테이블을 작성할 때, ID 컬럼을 테이블에 추가할 수 있습니다. 테이블에 추가된 각 행의 고유 숫자 값을 생성하려면, 식별 컬럼에 고유 인덱스를 정의하거나 기본 키를 선언해야 합니다.

식별 컬럼의 기타 사용은 주문 번호, 사원 번호, 재고 번호 또는 변환 기록 번호입니다. DB2가 생성하는 ID 컬럼 값은 ALWAYS 또는 BY DEFAULT입니다.

GENERATED ALWAYS로 정의되는 식별 컬럼에는 DB2 데이터베이스 관리 프로그램에 의해 항상 생성되는 값이 주어집니다. 응용프로그램은 명시적 값을 제공하도록 허용되지 않습니다. GENERATED BY DEFAULT로서 정의된 식별 컬럼은 식별 컬럼의 값을 명시적으로 제공하는 방법을 응용프로그램에 제공합니다. 응용프로그램이 값을 제공하지 않는 경우, DB2는 값을 생성합니다. 응용프로그램이 값을 제어하므로, DB2가 값의 고유성을 보장할 수 없습니다. GENERATED BY DEFAULT절은 기존 테이블의 내용을 복사하거나 테이블을 언로드 및 다시 로드할 의도가 있는 곳에서 데이터 보급을 위해 사용됩니다.

작성된 후에는 먼저 DEFAULT 옵션을 갖는 컬럼을 추가하여 기존 디폴트값을 가져와야 합니다. 그런 다음 디폴트를 변경(ALTER)하여 ID 컬럼이 되게 할 수 있습니다.

행을 지정된 명시적 식별 컬럼 값으로 테이블에 삽입하는 경우, 그 다음 내부적으로 생성된 값은 갱신되어 있지 않아 테이블의 기존 값과 충돌할 수 있습니다. 식별 컬럼에 있는 값의 고유성이 기본 키 또는 식별 컬럼에 정의된 고유 인덱스에 의해 실행되는 경우 중복 값은 오류 메시지를 생성합니다.

새 테이블에 식별 컬럼을 정의하려면 AS IDENTITY절을 CREATE TABLE문에 사용하십시오.

예 :

다음은 CREATE TABLE문에서 ID 컬럼을 정의하는 예입니다.

```
CREATE TABLE table (col1 INT,
                                    col2 DOUBLE,
                                    col3 INT NOT NULL GENERATED ALWAYS AS IDENTITY
                                (START WITH 100, INCREMENT BY 5))
```

이 예에서 세 번째 컬럼은 식별 컬럼입니다. 또한 컬럼을 추가할 때 각 행을 고유하게 식별하기 위해 컬럼에서 사용되는 값을 지정할 수 있습니다. 입력된 첫 번째 행은 컬럼에 값『100』을 가지며, 테이블에 추가되는 차후의 모든 행은 5씩 증가되는 연관된 값을 갖습니다.

컬럼 데이터를 제한조건, 디폴트 및 널(null) 설정으로 제한

데이터는 보통 특정 제한사항이나 규칙을 따라야 합니다. 그러한 제한사항은 형식 및 시퀀스 번호 같은 단일 정보에 적용될 수 있거나, 여러 부분의 정보에 적용될 수 있습니다.

컬럼 데이터 값의 널(null) 가능성

널(NULL) 값은 알 수 없는 상태를 나타냅니다. 디폴트로 모든 내장 데이터 유형은 널(NULL) 값의 존재를 지원합니다. 그러나 일부 비즈니스 규칙은 일부 컬럼(예: 비상 정보)에 대한 값이 항상 제공되어야 함을 규정합니다. 이 조건의 경우 NOT NULL 제한조건을 사용하여 테이블의 주어진 컬럼에 널(NULL) 값이 지정되지 않도록 보장할 수 있습니다. 특정 컬럼에 NOT NULL 제한조건이 정의되면 해당 컬럼에 널(NULL) 값을 입력하려는 모든 삽입 또는 갱신 조작은 실패합니다.

디폴트 컬럼 데이터 값

값이 항상 제공되어야 한다고 규정하는 일부 비즈니스 규칙처럼, 기타 비즈니스 규칙은 값(예: 직원의 성별)이 M 또는 F여야 한다고 규정할 수 있습니다. 컬럼 디폴트 제한조건은 해당 컬럼에 대해 특정 값을 갖지 않는 행이 테이블에 추가될 때마다 테이블의 주어진 컬럼에 항상 사전 정의된 값이 지정되도록 보장하는 데 사용됩니다. 컬럼에 대해 제공되는 디폴트값은 널(null), 컬럼의 데이터 유형과 호환 가능한 제한조건 또는 데이터베이스 관리 프로그램이 제공하는 값일 수 있습니다. 자세한 정보는 266 페이지의 『디폴트 컬럼 및 데이터 유형 정의』를 참조하십시오.

키

키는 데이터의 특정 행을 식별하거나 액세스하는 데 사용될 수 있는 단일 컬럼 또는 테이블 또는 인덱스의 컬럼 세트입니다. 모든 컬럼은 키의 파트일 수 있으며 동일한 컬럼이 둘 이상의 키의 파트일 수 있습니다. 단일 컬럼으로 구성되는 키를 원자 키라고 합니다. 둘 이상의 컬럼으로 구성되는 키는 복합 키라고 합니다. 원자 또는 복합 속성을 갖는 것 외에, 키는 제한조건을 구현하는 데 사용되는 방법에 따라서 분류됩니다.

- 고유 키는 고유 제한조건을 구현하는 데 사용됩니다.
- 기본 키는 엔티티 무결성 제한조건을 구현하는 데 사용됩니다. (기본 키는 널(NULL) 값을 지원하지 않는 고유 키의 특수한 유형입니다.)
- 외부 키는 참조 무결성 제한조건을 구현하는 데 사용됩니다. (외부 키는 기본 키나 고유 키를 참조해야 합니다. 외부 키는 대응하는 인덱스를 갖지 않습니다.)

키는 일반적으로 테이블, 인덱스 또는 참조 제한조건 정의의 선언 중에 지정됩니다.

제한조건

제한조건은 테이블에서 삽입, 삭제 또는 갱신될 수 있는 값을 제한하는 규칙입

니다. 점검 제한조건, 기본 키 제한조건, 참조 제한조건, 고유 제한조건, 고유 키 제한조건, 외부 키 제한조건 및 정보용 제한조건이 있습니다. 이러한 각 유형의 제한조건에 대한 자세한 내용은 321 페이지의 제 12 장『제한조건』또는 321 페이지의『제한조건 유형』의 내용을 참조하십시오.

디폴트 컬럼 및 데이터 유형 정의:

특정 컬럼 및 데이터 유형은 사전 정의되거나 지정된 디폴트값을 갖고 있습니다.

예를 들어 다양한 데이터 유형에 대한 디폴트 컬럼 값은 다음과 같습니다.

- *NULL*
- *0* : 작은 정수, 정수, 10진수, 단정밀도 부동 소수점(single-precision floating point), 배정밀도 부동 소수점(double-precision floating point) 및 10진 부동 소수점(decimal floating point) 데이터 유형에 사용됩니다.
- *공백:* 고정 길이 및 고정 길이 2바이트 문자열에 사용됩니다.
- *0길이 문자열:* 가변 길이 문자열, 실행 파일 대형 오브젝트(BLOB), 문자 대형 오브젝트(CLOB) 및 2바이트 문자 대형 오브젝트(LOB)에 사용됩니다.
- *날짜:* 이것은 행이 삽입되는 시간의 시스템 날짜입니다(CURRENT_DATE 특수 레지스터에서 얻음). 날짜 컬럼이 기존 테이블에 추가되면 기존 행에 0001년 1월 01일이라는 날짜가 지정됩니다.
- *시간 또는 시간소인:* 이것은 명령문이 삽입되는 시간의 시스템 시간 또는 시스템 날짜/시간입니다(CURRENT_TIME 특수 레지스터에서 얻음). 시간 컬럼이 기존 테이블에 추가되면 기존 행에 시간 00:00:00 또는 0001년 1월 01일 00:00:00시가 들어 있는 시간소인이 지정됩니다.

  주: 모든 행은 주어진 명령문에 대해 동일한 디폴트 시간/시간소인 값을 갖습니다.
- *구별 사용자 정의 데이터 유형:* 이것은 구별 사용자 정의 데이터 유형의 기본 데이터 유형에 대한 시스템 정의 디폴트값입니다(구별 사용자 정의 데이터 유형에 캐스트됨).

갱신 로깅을 최소화하기 위한 컬럼 정렬:

CREATE TABLE문을 사용하여 컬럼을 정의할 때 특히 갱신 집약적인 워크로드의 경우 컬럼의 순서를 고려하십시오. 자주 갱신되는 컬럼은 함께 그룹화되고 테이블 정의를 향해서 또는 정의의 끝에서 정의되어야 합니다. 그러면 성능이 더 좋아지고 더 적은 바이트가 로깅되며 더 적은 로그 페이지가 기록될 뿐 아니라 많은 수의 갱신을 수행하는 트랜잭션에 대한 활성 로그 스페이스 요구사항이 더 작아집니다.

데이터베이스 관리 프로그램은 자동으로 UPDATE문의 SET절에서 지정되는 컬럼의 값이 변경되고 있다고 가정하지 않습니다. 인덱스 유지보수 및 로깅되어야 하는 행의 양을 제한하기 위해, 데이터베이스는 이전 컬럼 값에 대해 새 컬럼 값을 비교하여 컬럼이

변하고 있는지 판별합니다. 값이 변하고 있는 컬럼만 갱신되는 것으로 취급됩니다. 이 UPDATE 동작의 예외는 데이터가 데이터 행 밖에 저장되는 컬럼(long, LOB, ADT 및 XML 컬럼 유형) 및 레지스트리 변수 DB2ASSUMEUPDATE가 사용 가능할 때 고정 길이 컬럼의 경우에 발생합니다. 이러한 예외의 경우 컬럼 값은 변하고 있는 것으로 가정되므로 새 컬럼 값과 이전 컬럼 값 사이에 비교가 이루어지지 않습니다.

UPDATE 로그 레코드에는 4가지 유형이 있습니다.

- 전체 사전 및 사후 행 이미지 로깅. 행의 전체 사전 및 사후 이미지가 로그됩니다. 이것이 DATA CAPTURE CHANGES로 사용 가능한 테이블에 대해 수행되는 유일한 로깅 유형이며, 대부분의 바이트 수가 행 갱신에 대해 로그됩니다.
- 전체 사전 행 이미지, 변경된 바이트 및 크기가 늘어나는 갱신의 경우 행 끝에 추가되는 새 데이터. 이것은 DATA CAPTURE CHANGES가 테이블에 적용되지 않을 때, 갱신이 트랜잭션을 위해 이 행에 대한 첫 번째 조치일 때 현재 커미트됨을 지원하는 데이터베이스의 경우 로그됩니다. 이것은 현재 커미트됨에 필요한 사전 이미지 및 재수행/실행 취소에 대해 맨 위에서 필요한 최소를 로그합니다. 자주 갱신되는 컬럼을 끝에서 정렬하면 행의 변경된 분할 영역에 대한 로깅을 최소화합니다.
- 전체 XOR 로깅. 변하고 있는 첫 번째 바이트부터 더 작은 행의 끝까지, 그런 다음 더 긴 행의 모든 남은 바이트까지 사전 및 사후 행 이미지 사이의 XOR 차이. 이 경우 전체 사전 및 사후 이미지 로깅보다 로그되는 바이트가 적어지며, 로그 레코드 헤더 정보를 넘어서는 데이터의 바이트 수가 가장 큰 행 이미지의 크기가 됩니다.
- 부분 XOR 로깅. 변하고 있는 첫 번째 바이트부터 변하고 있는 마지막 바이트까지 사전 및 사후 행 이미지 사이의 XOR 차이. 바이트 위치는 컬럼의 첫 번째 또는 마지막 바이트일 수 있습니다. 그러면 가장 적은 수의 바이트가 로깅되며 행 갱신에 대한 가장 효율적인 로그 레코드 유형이 됩니다.

위에서 나열되는 UPDATE 로그 레코드의 처음 두 유형의 경우, DATA CAPTURE CHANGES가 테이블에서 사용 불가능할 때 갱신에 대해 로그되는 데이터 양은 다음에 따라 다릅니다.

- 갱신되는 컬럼의 근접(COLNO)
- 갱신됨 컬럼이 고정 길이 또는 가변 길이인지 여부
- 행 압축(COMPRESS YES)이 사용 가능한지 여부

행이 전체 길이가 변하지 않고 있을 때는 행 압축이 사용 가능할 때도 데이터베이스 관리 프로그램이 최적 부분 XOR 로그 레코드를 계산하고 기록합니다.

행의 전체 길이가 변하고 있을 때(가변 길이 컬럼이 갱신되고 행 압축이 사용 가능할 때는 일반적임), 데이터베이스 관리 프로그램이 첫 번째로 변경될 바이트를 판별하고 전체 XOR 로그 레코드를 기록합니다.

기본 키, 참조 무결성, 점검 및 고유 제한조건

제한조건은 테이블에서 삽입, 삭제 또는 갱신될 수 있는 값을 제한하는 규칙입니다.

기본 키 제한조건

기본 키 제한조건은 하나의 컬럼 또는 동일한 등록 정보를 고유 제한조건으로 갖고 있는 컬럼의 조합입니다. 기본 키 제한조건과 외부 키 제한조건을 사용하여 테이블 간의 관계를 정의할 수 있습니다.

참조 무결성(또는 외부 키) 제한조건

외부 키 제한조건(참조 제한조건 또는 참조 무결성 제한조건이라고도 함)은 하나 이상의 테이블에 있는 하나 이상의 컬럼의 값에 대한 논리적 규칙입니다. 예를 들면, 테이블 세트가 기업의 제공업체에 대한 정보를 공유합니다. 제공업체 이름이 변경되는 경우도 있습니다. 테이블에 있는 제공업체 ID가 제공업체 정보의 제공업체 ID와 일치해야 함을 지정하는 참조 제한조건을 정의할 수 있습니다. 이 제한조건은 삽입, 갱신 또는 삭제 조작을 예방함으로써 제공업체 정보가 누락되지 않도록 합니다.

점검 제한조건

(테이블) 점검 제한조건은 특정 테이블에 추가되는 데이터에 대한 제한사항을 설정합니다.

고유 제한조건

고유 제한조건(고유 키 제한조건이라고도 함)은 테이블의 하나 이상의 컬럼에 있는 값이 중복되지 않도록 하는 규칙입니다. 고유 키 및 기본 키가 고유 제한조건으로 지원됩니다.

유니코드 테이블 및 데이터 고려사항

유니코드 문자 코드화 표준은 사용 중인 거의 모든 세계 언어의 문자를 포함하는 고정 길이의 코드화 체계입니다.

유니코드 테이블 및 데이터 고려사항에 대한 자세한 정보는 다음을 참조하십시오.

- *자국어 안내서*의 『유니코드 문자 인코딩』
- *자국어 안내서*의 『조합 순서를 기초로 하는 문자 비교』
- *자국어 안내서*의 『지역 코드별 날짜 및 시간 형식』
- *자국어 안내서*의 『유로 사용 코드 페이지에 대한 변환표 파일』

*유니코드 표준*의 최신 개정판 및 www.unicode.org의 Unicode Consortium 웹 사이트에서 유니코드에 대한 추가 정보를 찾을 수 있습니다.

테이블의 스페이스 요구사항

테이블을 디자인할 때 테이블이 포함할 데이터에 대한 스페이스 요구사항을 고려해야 합니다. 특히 LOB 또는 XML 같이 더 큰 데이터 유형을 갖는 컬럼에 주의를 기울여야 합니다.

대형 오브젝트(LOB) 데이터

대형 오브젝트(LOB) 데이터는 다른 데이터 유형에 대한 스토리지 스페이스와 다르게 구조화된 별도의 두 테이블 오브젝트에 저장됩니다. LOB 데이터에 필요한 스페이스를 계산하려면, 이러한 데이터 유형을 사용하여 정의된 데이터를 저장하는 데 사용되는 두 테이블 오브젝트를 고려해야 합니다.

- *LOB 데이터 오브젝트:* 데이터는 *크기*가 1024바이트의 "2의 제곱"배인 세그먼트로 나뉘는 64MB 영역에 저장됩니다. 즉, 이 세그먼트의 *크기*는 1024바이트, 2048바이트, 4069바이트 등으로 64MB까지 가능합니다.

  LOB 데이터에 의해 사용되는 디스크 양을 줄이기 위해 CREATE TABLE 및 ALTER TABLE문의 lob-options절에 COMPACT 옵션을 지정할 수 있습니다. COMPACT 옵션은 LOB 데이터를 더 작은 세그먼트로 분할함으로써 필요한 디스크 스페이스를 최소화합니다. 이 프로세스에서는 데이터를 압축하지 않으며, 1KB 정도의 최소 스페이스를 사용할 뿐입니다. LOB 값에 추가할 때 COMPACT 옵션을 사용하면 성능이 떨어질 수 있습니다.

  LOB 데이터 오브젝트에 포함된 사용 가능한 스페이스의 양은 삽입될 LOB 값의 크기뿐만 아니라 갱신 및 삭제 활동량에도 영향을 받습니다.
- *LOB 할당 오브젝트:* 할당 및 여유 공간 정보가 실제 데이터와는 구별되는 할당 페이지에 저장됩니다. 이 4KB 페이지의 수는 대형 오브젝트(LOB) 데이터에 할당된 데이터량(사용되지 않는 스페이스도 포함)에 따라 달라집니다. 오버헤드는 다음과 같이 계산됩니다.

표 15. 페이지 크기를 기초로 한 할당 페이지 오버헤드

페이지 *크기*	할당 페이지 수
4KB	모든 4MB당 한 페이지, 더하기 모든 1GB당 한 페이지
8KB	모든 8MB당 한 페이지, 더하기 모든 2GB당 한 페이지
16KB	모든 16MB당 한 페이지, 더하기 모든 4GB당 한 페이지
32KB	모든 32MB당 한 페이지, 더하기 모든 8GB당 한 페이지

  문자 데이터가 페이지 *크기*보다 작고 나머지 데이터의 레코드와 길이가 같을 경우, BLOB, CLOB 또는 DBCLOB 대신 CHAR, GRAPHIC, VARCHAR 또는 VARGRAPHIC 데이터 유형을 사용해야 합니다.

주: 일부 LOB 데이터는 CREATE 및 ALTER TABLE문의 INLINE LENGTH 옵션을 사용하여 기본 테이블 행에 저장될 수 있습니다.

Long 필드(LF) 데이터

Long 필드(LF) 데이터는 다른 데이터 유형의 스토리지 스페이스와는 다르게 구조화된 별도의 테이블 오브젝트에 저장됩니다. 데이터는 *크기*가 512바이트의 "2의 제곱"배인 세

그먼트로 나뉜 32KB 영역에 저장됩니다. (즉, 이 세그먼트의 *크기*는 512바이트, 1024 바이트, 2048바이트 등으로 32,768바이트까지 가능합니다.)

Long 필드 데이터 유형(LONG VARCHAR 또는 LONG VARGRAPHIC)은 사용 가능한 스페이스를 쉽게 재생시킬 수 있는 방식으로 저장됩니다. 할당 및 사용 가능한 스페이스 정보는 4KB의 할당 페이지에 저장되는데, 이 페이지는 오브젝트 전반에 걸쳐 가끔 나타납니다.

오브젝트 스페이스 중 사용되지 않는 스페이스의 양은 long 필드 데이터의 *크기*와 이 *크기*가 모든 데이터 어커런스에 대해 상대적으로 일정한지 여부에 따라 달라집니다. 256 바이트를 초과하는 데이터 항목인 경우, 사용되지 않는 이 스페이스가 long 필드 데이터 *크기*의 50퍼센트까지 될 수 있습니다.

문자 데이터가 페이지 *크기*보다 작고 나머지 데이터의 레코드와 길이가 같을 경우, LONG VARCHAR 또는 LONG VARGRAPHIC 대신 CHAR, GRAPHIC, VARCHAR 또는 VARGRAPHIC 데이터 유형을 사용해야 합니다.

시스템 카탈로그 테이블

시스템 카탈로그 테이블은 데이터베이스가 작성될 때 작성됩니다. 시스템 테이블은 데이터베이스 오브젝트 및 특권이 데이터베이스에 추가되면서 커집니다. 처음에는 약 3.5MB의 디스크 스페이스를 사용합니다.

카탈로그 테이블에 할당되는 스페이스는 테이블 스페이스의 유형 및 카탈로그 테이블이 들어 있는 테이블 스페이스의 Extent *크기*에 따라 다릅니다. 예를 들어, Extent *크기*가 32인 DMS 테이블 스페이스가 사용될 경우 처음에는 20MB의 스페이스가 카탈로그 테이블 스페이스에 할당됩니다. 주: 다중 파티션이 있는 데이터베이스의 경우, 카탈로그 테이블은 CREATE DATABASE 명령이 발행된 데이터베이스 파티션에만 상주합니다. 카탈로그 테이블용 디스크 스페이스는 해당 데이터베이스 파티션에만 필요합니다.

임시 테이블

일부 명령문은 처리를 위한 임시 테이블(예: 메모리에서 수행할 수 없는 정렬 조작용 작업 파일)이 필요합니다. 이 임시 테이블에는 디스크 스페이스가 필요합니다. 필요한 스페이스량은 쿼리의 *크기*, 개수 및 특성과 리턴된 테이블의 *크기*에 따라 달라집니다.

사용자의 작업 환경은 고유하기 때문에 임시 테이블에 대한 스페이스 요구사항의 판별을 예측하기 어렵습니다. 예를 들어, 다양한 시스템 임시 테이블의 긴 수명 주기로 인해 실제 사용 중인 것보다 많은 스페이스가 시스템 임시 테이블 스페이스에 할당된 것처럼 보일 수 있습니다. DB2_SMS_TRUNC_TMPTABLE_THRESH 레지스트리 변수가 사용될 때 이것이 발생할 수 있습니다.

데이터베이스 시스템 모니터와 테이블 스페이스 쿼리 API를 사용하면 일반 조작 과정에서 사용되는 작업 스페이스량을 추적할 수 있습니다.

DB2_OPT_MAX_TEMP_SIZE 레지스트리 변수를 사용하여 쿼리가 사용하는 임시 테이블 스페이스의 크기를 제한할 수 있습니다.

XML 데이터

XML 유형의 컬럼에 삽입하는 XML 문서는 디폴트 스토리지 오브젝트 또는 기본 테이블 행에 직접 상주할 수 있습니다. 기본 테이블 행 스토리지는 사용자의 제어 하에 있으며 작은 문서에만 사용할 수 있습니다. 더 큰 문서는 항상 디폴트 스토리지 오브젝트에 저장됩니다. 자세한 정보는 *pureXML Guide*의 『XML storage』를 참조하십시오.

테이블 페이지 크기

테이블 데이터의 행은 페이지라고 하는 블록으로 구성됩니다. 페이지 크기에는 네 가지 유형(4, 8, 16 및 32 KB)이 있습니다. LOB 또는 XML 문서가 INLINE LENGTH 옵션의 사용을 통해 인라인되지 않으면 테이블 데이터 페이지에는 LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DCLOB 또는 XML 데이터 유형으로 정의된 컬럼에 대한 데이터가 포함되지 않습니다. 그러나 테이블 데이터 페이지의 행에는 이 컬럼에 대한 디스크립터는 포함되어 있지 않습니다.

주: 일부 LOB 및 XML 데이터는 CREATE 및 ALTER TABLE문의 INLINE LENGTH 옵션을 사용하여 기본 테이블 행에 저장될 수 있습니다.

페이지 크기가 4KB, 8KB, 16KB 또는 32KB인 버퍼 풀 또는 테이블 스페이스를 작성할 수 있습니다. 특정 크기의 테이블 스페이스에서 작성된 모든 테이블의 페이지 크기는 일치합니다. 단일 테이블이나 인덱스 오브젝트는 32KB 페이지 크기를 가정할 때 64TB까지 커질 수 있습니다.

8KB, 16KB 또는 32KB의 페이지 크기를 사용할 경우에는 최대 1012개의 컬럼을 사용할 수 있습니다. 4KB 페이지 크기를 사용할 경우에는 최대 500개의 컬럼을 사용할 수 있습니다. 페이지당 사용할 수 있는 최대 행 수는 페이지 크기에 관계없이 255입니다.

최대 행 길이는 사용된 페이지 크기에 따라 변합니다.

- 페이지 크기가 4KB이면, 행 길이는 최대 4,005바이트일 수 있습니다.
- 페이지 크기가 8KB이면, 행 길이는 최대 8,101바이트일 수 있습니다.
- 페이지 크기가 16KB이면, 행 길이는 최대 16,293바이트일 수 있습니다.
- 페이지 크기가 32KB이면, 행 길이는 최대 32,677바이트일 수 있습니다.

테이블 스페이스의 페이지 크기를 결정할 때 고려해야 할 사항은 다음과 같습니다.

- 무작위 행 읽기 및 쓰기를 수행하는 OLTP 응용프로그램의 경우, 페이지 크기가 작으면 불필요한 행에 소비되는 버퍼 스페이스가 적기 때문에 작은 페이지 크기가 보다 바람직합니다.

- 한 번에 많은 수의 연속 행에 액세스하는 DSS 응용프로그램의 경우, 페이지 *크기*가 크면 특정 수의 행을 읽는데 필요한 입출력 요청 수가 줄어들기 때문에 페이지 *크기*가 클수록 유리합니다. 그러나 예외도 있습니다. 행 *크기*가 `pagesize / maximum rows`보다 작은 경우 각 페이지에 소비된 스페이스가 있습니다. 이 경우 더 작은 페이지 *크기*가 더 적합할 수 있습니다.

페이지 *크기*가 더 크면 인덱스의 레벨 수를 줄일 수 있습니다. 페이지 *크기*가 더 크면 보다 길이가 긴 행이 지원됩니다. 디폴트값인 4KB 페이지를 사용하면 테이블이 500컬럼으로 제한됩니다. 큰 페이지 *크기*(8KB, 16KB 및 32KB)는 1012컬럼을 지원합니다. 테이블 스페이스의 최대 *크기*는 테이블 스페이스의 페이지 *크기*에 비례합니다.

사용자 테이블 데이터에 대한 스페이스 요구사항

디폴트로 테이블 데이터는 테이블이 있는 테이블 스페이스 페이지 *크기*를 기초로 저장됩니다. 각 페이지(페이지 *크기*와 무관)에는 데이터베이스 관리 프로그램에 대한 68바이트 오버헤드가 포함됩니다. 행은 여러 페이지로 분리되지 *않습니다*. 4KB 페이지 크기를 사용할 경우 최대 500개의 컬럼을 사용할 수 있습니다.

테이블 데이터 페이지에는 LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB 또는 XML 데이터 유형으로 정의되는 컬럼에 대한 데이터가 들어 있지 *않습니다*. 그러나 테이블 데이터 페이지 내의 행에는 이 컬럼에 대한 디스크립터가 들어 있습니다.

주: 일부 LOB 데이터는 CREATE 및 ALTER TABLE문의 INLINE LENGTH 옵션을 사용하여 기본 테이블 행에 저장될 수 있습니다.

행은 대개 순서대로 일반 테이블에 삽입됩니다. 새 행을 저장할 정도의 *크기*를 가진 스페이스 중 첫 번째로 사용 가능한 스페이스를 파일에서 검색(여유 공간맵을 통해)합니다. 행이 갱신될 때, 페이지에 갱신된 행을 포함할 수 있는 충분한 스페이스가 남아 있어야만 해당 행이 갱신됩니다. 이러한 경우, 갱신된 행이 있는 테이블 파일 내의 새 위치를 가리키는 레코드가 원래 행의 위치에 작성됩니다.

ALTER TABLE문이 APPEND ON 옵션과 함께 호출되는 경우, 데이터는 항상 추가되며 해당 데이터 페이지의 여유 공간에 대한 정보는 저장되지 않습니다.

테이블에 클러스터링 인덱스가 정의되어 있으면 데이터베이스 관리 프로그램이 해당 클러스터링 인덱스의 키 순서에 따라 실제로 데이터 클러스터링을 시도합니다. 테이블에 행이 삽입되면, 데이터베이스 관리 프로그램이 먼저 클러스터링 인덱스에서 해당 키 값을 찾습니다. 키 값이 있으면 데이터베이스 관리 프로그램이 해당 키가 가리키는 데이터 페이지에서 레코드 삽입을 시도합니다. 키 값이 없으면 다음으로 높은 키 값이 사용되며, 레코드가 다음으로 높은 키 값을 갖는 레코드가 포함된 페이지에 삽입되도록 합니다. 테이블의 목표 페이지에 스페이스가 충분하지 않으면 이웃 페이지에서 스페이스를 찾기 위해 여유 공간 맵이 사용됩니다. 시간이 지남에 따라 데이터 페이지의 스페이

스를 모두 사용한 경우 레코드는 테이블의 목표 페이지에서 멀리 떨어진 위치에 배치됩니다. 그러면 테이블 데이터는 클러스터되지 않은 것으로 간주되며 테이블 재구성을 통해 클러스터된 순서를 복원할 수 있습니다.

테이블이 다차원적으로 클러스터된(MDC) 테이블이면, 데이터베이스 관리 프로그램은 하나 이상의 정의된 차원 또는 클러스터링 인덱스에서 레코드가 항상 실제로 클러스터됨을 보장합니다. MDC 테이블이 특정 차원을 사용하여 정의된 경우에는 블록 인덱스가 차원마다 작성되며 셀(차원 값의 고유 조합)을 블록에 맵핑하는 복합 블록 인덱스가 작성됩니다. 이 복합 블록 인덱스는 특정 레코드가 속하는 셀을 판별하며 해당 셀에 속하는 레코드가 포함된 테이블의 블록 또는 Extent를 정확히 판별하는데 사용됩니다. 결과적으로 레코드를 삽입할 때 데이터베이스 관리 프로그램은 복합 블록 인덱스에서 동일한 차원 값을 갖는 레코드가 포함된 블록 목록을 검색하며 스페이스 검색을 해당 블록으로만 제한합니다. 셀이 아직 존재하지 않거나 셀의 기존 블록에 스페이스가 충분하지 않으면 다른 블록이 셀에 지정되며 레코드가 이 셀에 삽입됩니다. 블록에서 사용 가능한 스페이스를 신속하게 찾기 위해 여유 공간 맵핑이 블록에서 계속 사용됩니다.

데이터베이스의 각 사용자 테이블에 대한 4KB 페이지 수는 다음과 같이 산정됩니다.

```
ROUND DOWN(4028/(average row size + 10)) = records_per_page
```

이 결과를 다음에 삽입합니다.

```
(number_of_records/records_per_page) * 1.1 = number_of_pages
```

여기서, 평균 행 *크기*는 평균 컬럼 *크기*의 합계이며, 인수 ″1.1″은 오버헤드 값입니다.

주: 이 공식은 단지 추정값만을 제공합니다. 단편화 및 오버플로우 레코드로 인해 레코드 길이가 다양한 경우 추정의 정확도가 감소합니다.

또한 8KB, 16KB 또는 32KB 페이지 *크기*를 갖는 버퍼 풀 또는 테이블 스페이스를 작성할 수도 있습니다. 특정 *크기*의 테이블 스페이스에서 작성된 모든 테이블의 페이지 *크기*는 일치합니다. 단일 테이블이나 인덱스 오브젝트는 32KB 페이지 *크기*를 가정할 때 64TB까지 커질 수 있습니다. 8KB, 16KB 또는 32KB 페이지 *크기*를 사용할 때에는 최대 1012개의 컬럼을 사용할 수 있습니다. 최대 컬럼 수는 4KB 페이지 *크기*일 때 500개 입니다. 페이지 *크기*에 따라 최대 행 길이도 변합니다.

- 페이지 *크기*가 4KB이면, 행 길이는 최대 4,005바이트일 수 있습니다.
- 페이지 *크기*가 8KB이면, 행 길이는 최대 8,101바이트일 수 있습니다.
- 페이지 *크기*가 16KB이면, 행 길이는 최대 16,293바이트일 수 있습니다.
- 페이지 *크기*가 32KB이면, 행 길이는 최대 32,677바이트일 수 있습니다.

페이지 *크기*가 클수록 인덱스의 레벨 수가 줄어들 가능성이 높아집니다. 무작위 행 읽기 및 *쓰기*를 수행하는 온라인 트랜잭션 처리(OLTP) 응용프로그램에서 작업할 경우, 페이지 *크기*가 작으면 불필요한 행에는 버퍼 스페이스를 덜 사용하기 때문에 페이지 크기가 작을수록 좋습니다. 한 번에 많은 수의 연속 행에 액세스하는 결정 지원 시스템

(DSS) 응용프로그램에서 작업하는 경우, 페이지 *크기*가 크면 특정 수의 행을 읽는데 필요한 입출력 요청 횟수가 줄어들기 때문에 페이지 *크기*가 클수록 좋습니다.

백업 이미지를 다른 페이지 *크기*로 리스토어할 수 없습니다.

756개 이상의 컬럼을 나타내는 IXF 데이터 파일은 임포트할 수 없습니다.

선언 또는 작성된 임시 테이블은 사용자 고유의 임시 테이블 스페이스 유형에서만 선언 또는 작성될 수 있습니다. 디폴트 사용자 임시 테이블 스페이스는 없습니다. 응용프로그램이 데이터베이스에서 연결을 끊을 때 임시 테이블이 내재적으로 삭제되며, 이들 테이블에 대한 스페이스 요구사항을 추정할 때 이 점을 고려해야 합니다.

테이블 행에 LOB 인라인 저장

대형 오브젝트(LOB)는 일반적으로 해당 오브젝트를 참조하는 테이블과는 다른 위치에 저장됩니다. 그러나 기본 테이블 행에 32,673바이트 길이의 인라인 LOB를 포함하여 액세스를 단순화할 수 있습니다.

기본 테이블 행에 큰 데이터 오브젝트를 포함하는 것은 비실용적(및 데이터에 따라서는 불가능)할 수 있습니다. 그림 32는 행에 LOB를 포함하려는 시도 및 그렇게 하면 문제가 될 수 있는 이유의 예를 나타냅니다. 이 예에서 행은 각각 길이가 500 및 145KB인 두 개의 LOB 컬럼을 갖는 것으로 정의됩니다. 그러나 DB2 테이블의 최대 행 *크기*는 32KB이므로, 그러한 행 정의는 사실 구현될 수 없습니다.

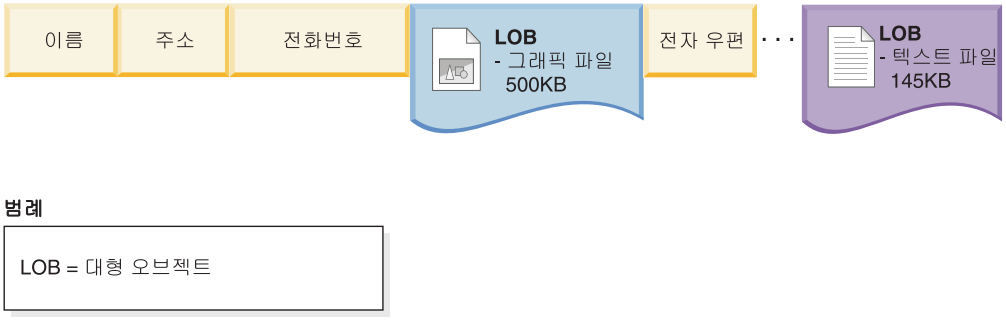


그림 32. 기본 테이블 행에 LOB 데이터 포함의 문제점

LOB 작업과 연관된 어려움을 줄이기 위해 LOB는 다른 데이터 유형과는 다르게 취급됩니다. 275 페이지의 그림 33은 LOB 자체가 아니라 LOB 디스크립터가 기본 테이블 행에 저장됨을 나타냅니다. 각 LOB 자체는 데이터베이스 관리 프로그램이 제어하는 별도의 LOB 위치에 저장됩니다. 이 방식에서 버퍼 풀과 디스크 스토리지 사이의 행 이동이 전체 LOB를 포함하는 경우보다 LOB 디스크립터를 갖는 행의 경우에 시간이 덜 걸립니다.

그러나 실제 LOB가 기본 테이블 행과는 별개인 위치에 저장되기 때문에 LOB 데이터의 처리는 더욱 어려워집니다.

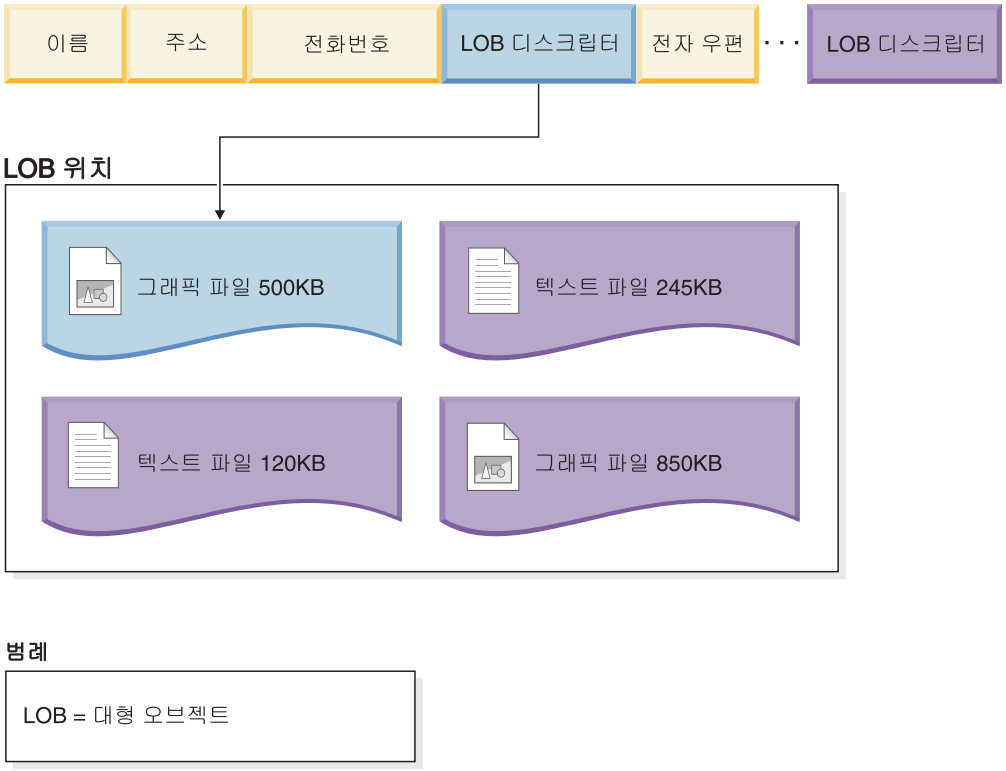


그림 33. 기본 테이블 행의 LOB 디스크립터는 별도 LOB 위치의 LOB를 참조함

더 작은 LOB의 처리를 단순화하기 위해 사용자가 지정하는 *크기* 임계값 아래로 떨어지는 LOB 데이터가 기본 테이블 행에 인라인 포함되게 만들 수 있습니다. 이러한 LOB 데이터 유형은 기본 테이블 행의 파트로서 처리될 수 있으며, 이는 버퍼 풀과의 이동 같은 조작을 더 단순하게 합니다. 또한 인라인 LOB는 행 압축이 사용 가능한 경우 행 압축에 대해 *규정*합니다.

CREATE 및 ALTER TABLE문의 INLINE LENGTH 옵션을 사용하여 사용자가 지정하는 길이 제한보다 더 작은 LOB 데이터가 기본 테이블 행에 포함될 수 있습니다. 디폴트로, INLINE LENGTH에 대해 명시적 값을 지정하지 않은 경우에도 컬럼에 대한 최대 *크기* LOB 디스크립터보다 작은 LOB는 항상 기본 테이블 행에 포함됩니다.

인라인 LOB에서 276 페이지의 그림 34에 표시된 것과 같은 기본 테이블 행을 가질 수 있습니다.

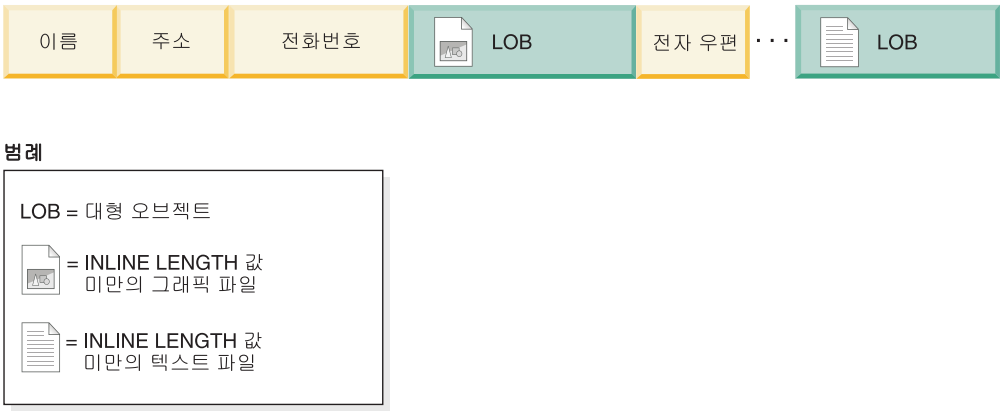


그림 34. 기본 테이블 행에 포함된 작은 LOB

LOB를 인라인 포함하기 위해 선택할 임계값을 고려할 때 데이터베이스의 현재 페이지 *크기*, 인라인 LOB로 인해 행 *크기*가 현재 페이지 *크기*를 초과할지 여부를 고려하십시오. 테이블에 있는 행의 최대 *크기*는 32,677바이트입니다. 그러나 각 인라인 LOB는 4바이트의 스토리지 오버헤드를 갖습니다. 따라서 인라인 저장하는 각 LOB는 행의 사용 가능한 스토리지를 4바이트씩 줄입니다. 따라서 인라인 LOB에 대한 최대 *크기*는 32,673바이트입니다.

주: LOB를 인라인으로 저장할 수 있는 것과 동일한 방법으로 XML 데이터도 인라인 저장할 수 있습니다.

테이블 압축

테이블을 디스크에 저장할 때 데이터 행 수, NULL 값 및 시스템 디폴트값 압축과 같은 기능을 이용하여 스페이스를 덜 사용하도록 할 수 있습니다. 압축을 통해 데이터 저장에 데이터베이스 페이지를 덜 사용하여 디스크 스토리지 스페이스를 절약할 수 있습니다.

또한 페이지당 더 많은 논리 데이터를 저장할 수 있기 때문에 동일한 양의 논리 데이터에 액세스하기 위해 읽어야 하는 페이지의 수가 적습니다. 즉, 압축으로 디스크 입출력을 저장할 수 있습니다. 버퍼 풀에 더 많은 논리 데이터가 캐시될 수 있기 때문에 입출력 속도가 증가할 수도 있습니다.

DB2 데이터베이스에서 데이터 압축을 구현하기 위해 다음 두 메소드를 사용할 수 있습니다.

- 행 압축
- 값 압축

신규 및 기존 테이블 모두에서 압축을 사용할 수 있습니다. 또한 임시 테이블에서도 사용할 수 있습니다.

행 압축

이 압축 메소드는 행 내부에서 다중 컬럼 값에 걸쳐있는 반복적인 패턴을 더 짧은 기호 문자열로 바꿔 데이터 행을 압축합니다. 데이터 행 압축의 목적은 디스크 스토리지 스페이스를 절약하는 것입니다. 디스크 입출력도 절약할 수 있습니다. 또한, 버퍼 풀에 더 많은 데이터를 캐시할 수 있으므로 버퍼 풀 사용 비율이 증가됩니다. 그러나 데이터를 압축하고 압축을 해제하는 데 필요한 추가 CPU 주기의 형식으로 트래이드 오프가 발생합니다. 스토리지 절약 및 데이터 행 압축의 성능 영향은 데이터베이스에 있는 데이터의 특성, 데이터베이스의 레이아웃과 조정 및 응용프로그램 워크로드에 영향을 받습니다. 데이터 페이지 또는 로그 레코드에 있는 데이터만 압축됩니다. 예를 들어 LOB 또는 Long 데이터 오브젝트는 압축되지 않습니다.

주: 기본 테이블 행으로의 인라인에 저장되는 LOB 데이터는 압축할 수 있습니다. INLINE LENGTH 옵션을 갖는 CREATE 및 ALTER TABLE문을 사용하여 규정 LOB 데이터를 기본 테이블 행에 저장할 수 있습니다.

행 압축은 정적 사전 기반 압축 알고리즘을 사용하여 데이터를 행별로 압축합니다. 사전은 테이블 데이터 행과 함께 테이블의 데이터 오브젝트 분할 영역에 저장됩니다. 테이블 데이터를 압축하려면 테이블의 COMPRESS 속성을 YES로 설정해야 합니다. 테이블을 작성할 때 CREATE TABLE문의 COMPRESS절을 사용하여 이를 수행할 수 있습니다. 또한 ALTER TABLE문에서 동일한 절을 사용하여 압축을 사용하도록 기존 테이블을 변경할 수도 있습니다. 압축이 사용 가능하면 INSERT, LOAD INSERT 또는 IMPORT INSERT 같이 데이터를 테이블에 추가하는 조작은 행 압축을 사용합니다. 또한 압축이 사용 가능하면 테이블에 대해 인덱스 압축도 사용 가능합니다. 디폴트로 압축할 수 있는 인덱스의 유형인 경우 작성되는 모든 인덱스는 압축 인덱스로 작성됩니다.

- CREATE TABLE문에서 이 속성을 설정할 때 테이블에서 반복되는 패턴을 추적하기 위해 테이블에 대한 압축 사전이 작성됩니다.
- 행 압축을 사용하도록 기존 테이블을 변경할 때, 압축 사전이 작성되며 테이블의 모든 행에 대해 압축을 사용할 수 있지만, 사용자가 오프라인 테이블 재구성을 수행할 때까지 기존 행에 압축이 실제로 적용되지 않습니다. 사용자가 즉시 재구성을 수행하지 않는 경우 기존 행은 압축되지 않습니다. 그러나 그 뒤에 추가 또는 갱신되는 행은 압축됩니다. 따라서 행 압축이 전체 테이블에 대해 사용 가능한 경우에도 다른 행은 압축되지 않는데 압축되는 일부 행이 존재할 수 있습니다. 테이블의 모든 행에 압축이 적용되도록 하려면 테이블 재구성을 수행해야 합니다.

행 압축 통계는 RUNSTATS 명령을 사용하여 생성될 수 있으며 시스템 카탈로그 테이블 SYSCAT.TABLES 및 SYSCAT.DATAPARTITIONS에 저장됩니다. 압축 예상 옵션은 ADMIN_GET_TAB_COMPRESS_INFO 유틸리티를 사용하여 사용할 수 있습니다. 쿼리 옵티마이저에서는 비용 모델에 압축 해제 비용이 포함됩니다.

UPDATE 활동 및 데이터 행에서의 갱신 변경사항의 위치에 따라 로그 사용량이 늘어날 수 있습니다. 갱신 로깅 및 컬럼 정렬에 대한 정보는 266 페이지의 『갱신 로깅을 최소화하기 위한 컬럼 정렬』의 내용을 참조하십시오.

*크기*가 증가하는 갱신이 진행 중인 행의 경우, 행의 새 버전이 현재 데이터 페이지에 맞지 않을 수 있습니다. 오히려 행의 새 이미지는 오버플로우 페이지에 저장됩니다. 이러한 포인터 오버플로우 레코드 작성을 최소화하기 위해 데이터 페이지에 여유 공간을 추가할 수 있습니다. 예를 들어, 압축 없이 5%의 여유 공간이 사용된 경우 압축할 때는 10%의 여유 공간을 할당하십시오. 이 권장사항은 특히 많이 갱신되는 데이터의 경우 중요합니다.

테이블의 압축을 사용 안하려면 ALTER TABLE문을 COMPRESS NO절과 함께 사용하십시오. 추가되는 후속 행은 압축되지 않습니다. 전체 테이블의 압축을 해제하려면 REORG TABLE 명령을 사용하여 테이블 재구성을 수행해야 합니다.

값 압축

이 압축 방법은 데이터 표시를 위한 스페이스 사용 및 데이터를 저장하기 위해 데이터베이스 관리 시스템이 처음에 사용하는 스토리지 구조를 최적화합니다. 값의 중복 항목 삭제 및 한 개의 복사만 저장하는 것이 값 압축에 포함됩니다. 저장된 사본이 저장된 값에 대한 모든 참조 위치를 추적합니다.

테이블을 작성할 때 CREATE TABLE문의 선택적인 VALUE COMPRESSION절을 사용하여 테이블이 값 압축을 사용하도록 지정할 수 있습니다. ALTER TABLE문의 ACTIVATE VALUE COMPRESSION절을 사용하여 기존 테이블에서 값 압축을 사용할 수 있습니다. 테이블에서 값 압축을 사용 안하려면 ALTER TABLE문의 DEACTIVATE VALUE COMPRESSION절을 사용합니다.

VALUE COMPRESSION을 사용하면, 정의된 가변 길이 데이터 유형(VARCHAR, VARGRAPHICS, LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB 및 DBCLOB)에 지정되었던 0 길이 데이터 및 널(NULL)은 디스크에 저장되지 않습니다. 이러한 데이터 유형과 연관된 오버헤드 값만이 디스크 스페이스를 차지합니다.

VALUE COMPRESSION을 사용할 경우에는 선택적 COMPRESS SYSTEM DEFAULT 옵션을 사용하여 추가로 디스크 스페이스 사용량을 줄일 수 있습니다. 디폴트값은 디스크에 저장되지 않기 때문에 삽입 또는 갱신되는 값이 컬럼의 데이터 유형에 대한 시스템 기본값과 동일한 경우 최소 디스크 스페이스가 사용됩니다. COMPRESS SYSTEM DEFAULT를 지원하는 데이터 유형에는 모든 숫자 유형 컬럼, 고정 길이 문자 및 고정 길이 그래픽 문자열 데이터 유형이 포함됩니다. 이는 0 및 공백을 압축할 수 있음을 의미합니다.

값 압축을 사용할 때, 행에 있는 압축된 컬럼의 바이트 수가 동일한 컬럼의 압축되지 않은 버전의 바이트 수보다 더 클 수 있습니다. 행 *크기*가 페이지 *크기*에 대해 허용되

는 최대값에 접근하는 경우 압축 및 압축되지 않은 컬럼에 대한 바이트 수 합계가 테이블 스페이스에서 테이블의 허용 가능한 행 길이를 초과하지 않도록 해야 합니다. 예를 들어 4KB 페이지 크기를 갖는 테이블 스페이스에서 허용 가능한 행 길이는 4005 바이트입니다. 이 값을 초과하면 오류 메시지 SQL0670N이 리턴됩니다. 압축 및 압축되지 않은 컬럼에 대한 바이트 수를 판별하는 데 사용되는 공식은 CREATE TABLE 문의 파트로서 문서화됩니다.

값 압축을 비활성화하는 경우,

- COMPRESS SYSTEM DEFAULTS가 이전에 사용 가능했던 경우 암시적으로 비활성화됩니다.
- 압축되지 않은 컬럼으로 인해 행 크기가 현재 테이블 스페이스의 현재 페이지 크기가 허용하는 최대값을 초과할 수 있습니다. 이 경우 오류 메시지 SQL0670N이 리턴됩니다.
- 기존의 압축 데이터는 행이 갱신되거나 사용자가 REORG 명령으로 테이블 재구성을 수행할 때까지 압축된 상태로 있습니다.

임시 테이블 압축

임시 테이블 압축은 스토리지 최적화 기능 라이센스와 함께 자동으로 사용 가능합니다. 쿼리를 실행할 때 DB2 옵티마이저의 비용 모델이 스토리지 절약 및 쿼리 성능에 대한 영향을 고려하여 임시 테이블 압축이 의미가 있는지 여부를 판별합니다.

EXPLAIN 기능 또는 db2pd 도구를 사용하여 옵티마이저가 임시 테이블 압축을 사용할지 여부를 확인할 수 있습니다.

자동 압축 사전 작성

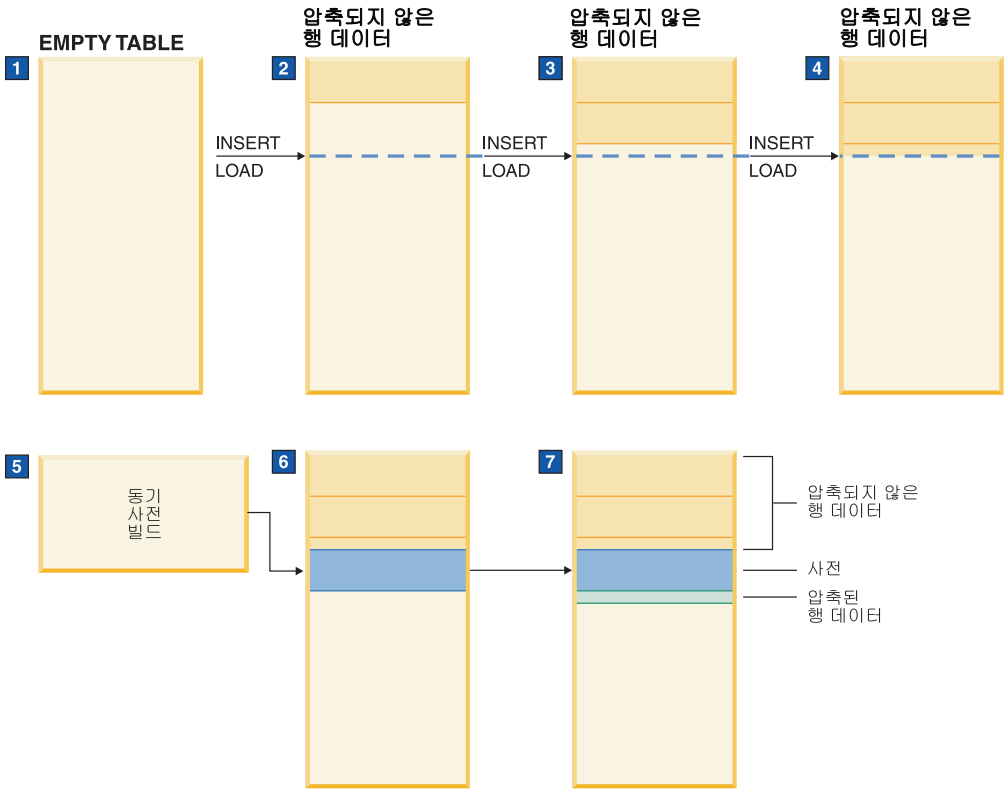
압축 사전은 테이블에 더 많은 데이터를 추가할 수 있도록 여유 공간을 확보하기 위해 테이블에 압축 데이터를 이동하는 데 사용됩니다. 특정 조건이 충족되면 로드 또는 삽입 조작과 같은 데이터 채우기 조작을 수행하는 동안 테이블에 압축 사전이 자동으로 작성되어 삽입되거나 추가됩니다.

테이블에 대해 COMPRESS 속성을 정의했거나 압축 사전이 아직 물리 테이블 또는 파티션에 없는 경우 및 테이블에 충분한 데이터가 있는 경우 테이블과 관련하여 자동 사전 작성(ADC)이 발생합니다. 그 후에 테이블로 이동되는 데이터는 테이블 COMPRESS 속성이 사용 가능한 상태로 남아 있는 경우 압축 사전을 사용하여 압축됩니다.

테이블이 DB2 버전 9.7에서 작성되었고 XML 유형의 컬럼이 하나 이상 포함되어 있는 경우 두 번째 압축 사전을 사용하여 테이블과 연관된 디폴트 XML 스토리지 오브젝트에 저장된 XML 데이터를 압축합니다. 테이블에 대해 COMPRESS 속성을 정의했거나 해당 XML 스토리지 오브젝트에 아직 압축 사전이 없거나 테이블의 XML 스토리지 오브젝트에 데이터가 충분히 있는 경우 이 사전의 ADC가 발생합니다. XML 데이터 압축에 사용되는 압축 사전은 디폴트 XML 스토리지 오브젝트에 저장됩니다.

제한사항: DB2 버전 9.5 또는 DB2 버전 9.1로 작성된 XML 컬럼의 데이터는 압축할 수 없습니다. 그러나, DB2 버전 9.7을 사용하여 제품의 이전 버전에서 작성된 테이블에 추가하는 XML 컬럼은 압축할 수 있습니다.

다음 다이어그램은 압축 사전을 자동으로 작성하는 프로세스를 나타냅니다.



1. 테이블이 비어 있기 때문에 압축 사전이 작성되지 않습니다.
2. 삽입 또는 로드 조작을 사용하여 테이블에 데이터가 삽입되고 압축되지 않은 상태로 남습니다.
3. 테이블에 추가 데이터를 삽입하거나 로그하면 압축되지 않은 상태로 남아 있습니다.
4. 임계값에 도달하면 COMPRESS 속성이 YES로 설정되어 있는 경우 사전 작성이 자동으로 트리거됩니다.
5. 사전이 작성됩니다.
6. 사전이 테이블에 추가됩니다.
7. 이후로 데이터가 압축됩니다.

다음 표에는 릴리스별 압축 사전 작성 차이점이 표시되어 있습니다.

표 16. 릴리스별 압축 사전 작성 차이점

명령 및 속성	버전 9.7	버전 9.5	버전 9.1
RESETDICTIONARY 옵션을 사용하는 LOAD REPLACE 명령	버전 9.5와 비슷합니다. 또한 테이블 COMPRESS 속성을 YES로 설정하면 최소한 하나의 XML 문서가 디폴트 XML 스토리지 오브젝트에 로드되거나 삽입되는 경우 디폴트 XML 스토리지 오브젝트의 기존 압축 사전이 제거되고 새 사전이 생성됩니다.	테이블 COMPRESS 속성을 YES로 설정하면 최소한 하나의 데이터 행이 테이블에 로드되거나 삽입되는 경우 기존 압축 사전이 제거되고 새 사전이 생성됩니다.	적용할 수 없습니다.
COMPRESS 속성이 YES로 설정된 CREATE 또는 ALTER TABLE문	버전 9.5와 비슷합니다. 또한 테이블이 DB2 버전 9.7에서 작성되었고 최소한 하나의 XML 유형 컬럼이 포함되어 있는 경우 테이블 COMPRESS 속성을 YES로 설정하면 해당 테이블의 디폴트 XML 스토리지 오브젝트와 연관된 압축 사전과 관련하여 테이블이 ADC에 적격이 됩니다.	테이블 COMPRESS 속성을 YES로 설정하면 최소한 하나의 데이터 행이 테이블에 로드되거나 삽입되는 경우 테이블이 ADC에 적격이 됩니다.	자동으로 사전이 작성되지 않습니다. 테이블 데이터를 압축하려면 테이블 재구성 프로세스를 사용하여 명시적으로 압축을 작성해야 합니다.
COMPRESS 속성이 YES로 설정된 CREATE 또는 ALTER INDEX문	인덱스 압축을 CREATE INDEX 또는 ALTER INDEX문에서 명시적으로 사용 불가능하게 설정하지 않는 한 새 인덱스에서 인덱스 압축이 사용 가능합니다. ALTER INDEX문을 사용하여 인덱스 압축을 사용하거나 사용 안할 수 있습니다. 그러나 이전에 압축되지 않은 인덱스에 대해 압축을 사용하는 경우 압축이 적용되기 전에 REORG INDEXES 명령을 사용하여 인덱스 재구성을 수행해야 합니다. **제한사항:** 인덱스 압축은 다음 유형의 인덱스에 대해서는 지원되지 않습니다. • MDC 블록 인덱스 • XML 경로 인덱스 또한, • 인덱스 스펙은 압축할 수 없습니다. • 임시 테이블의 인덱스에 대한 압축 속성은 ALTER INDEX 명령으로 변경할 수 없습니다. DROP INDEX문과 마찬가지로 ALTER INDEX문은 XML 인덱스의 논리적 인덱스 이름에만 적용됩니다. XML 실제 인덱스 이름을 지정하면 오류가 리턴됩니다(SQLSTATE 42704). 또한 임시 테이블에서는 ALTER INDEX문이 지원되지 않습니다.	적용할 수 없습니다.	적용할 수 없습니다.

표 16. 릴리스별 압축 사전 작성 차이점 (계속)

명령 및 속성	버전 9.7	버전 9.5	버전 9.1
INSERT, LOAD INSERT, IMPORT INSERT 또는 REDISTRIBUTE 명령	버전 9.5와 비슷합니다. 또한 테이블 COMPRESS 속성을 YES로 설정하면 테이블에 XML 데이터가 충분히 있는 경우 디폴트 XML 스토리지 오브젝트와 연관된 압축 사전이 아직 없는 테이블이 ADC의 대상이 됩니다.	테이블 COMPRESS 속성을 YES로 설정하면 테이블에 데이터가 충분히 있는 경우(즉, 임계값을 지난 경우) 아직 압축 사전이 없는 테이블이 ADC의 대상이 됩니다. **주:** REDISTRIBUTE 명령은 새로 추가된 데이터베이스 파티션에서만 ADC를 트리거합니다.	적용할 수 없습니다.
KEEPDICTIONARY 옵션을 사용하는 REORG TABLE 명령	버전 9.5와 비슷합니다. 또한 테이블에 하나 이상의 XML 유형 컬럼이 포함되어 있는 경우, 디폴트 XML 스토리지 오브젝트 크기가 ADC XML 임계값과 같고 테이블에 임계값을 초과한 충분한 XML 데이터가 있는 경우에만 디폴트 XML 스토리지 오브젝트의 압축 사전이 삽입됩니다.	테이블 크기가 ADC 테이블 크기 임계값과 같고 테이블에 임계값을 초과한 충분한 데이터가 있는 경우에만 테이블에 사전이 삽입됩니다.	테이블 COMPRESS 속성을 YES로 설정했으며 테이블에 아직 압축 사전이 없는 경우, 테이블에 있는 데이터 볼륨에 관계 없이 테이블에 압축 사전을 빌드, 삽입 또는 추가하려 했습니다.

압축을 사용하는 테이블 작성

새 테이블을 작성할 때 압축이 사용 가능하도록 CREATE TABLE 명령에 대해 COMPRESS 속성을 사용할 수 있습니다.

이 태스크에 대한 정보

행 압축만, 값 압축만 또는 두 유형의 압축을 모두 사용할지 여부를 결정해야 합니다. 행 압축은 한 행에서 여러 컬럼에 걸쳐있는 데이터 패턴을 더 짧은 기호 문자열로 바꾸려고 하기 때문에 거의 항상 스토리지를 절약할 수 있습니다. 값 압축은 동일한 값을 포함하는 컬럼이 있는 행이 많을 때 또는 컬럼의 데이터 유형에 대해 디폴트값을 포함하는 컬럼이 있을 때 절약을 제공할 수 있습니다. 값 압축이 사용 가능한 경우 데이터 유형에 대해 시스템 기본값을 가정하는 컬럼이 COMPRESS SYSTEM DEFAULT 옵션으로 더욱 압축될 수 있다고 지정할 수도 있습니다.

압축된 테이블에 대해 작성된 모든 인덱스도 디폴트로 압축됩니다.

제한사항

COMPRESS SYSTEM DEFAULT절을 사용하여 시스템 기본값을 포함하는 컬럼에 압축을 적용하려는 경우 VALUE COMPRESSION도 지정해야 합니다. 그렇지 않으면 경고가 리턴되고 시스템 기본값은 최소 스페이스를 사용하여 저장되지 않습니다.

값 압축을 사용하려고 계획하는 경우, 특정 데이터 유형을 다루기 위해 데이터베이스 관리 프로그램에 의해 부과되는 오버헤드의 결과로 일부 경우에는 행 *크기*가 늘어날 수 있음을 알아야 합니다. CREATE TABLE문에 대한 문서에서 이 옵션에 대해 제공되는 정보를 사용하여 값 압축이 행 *크기*에 대해 갖는 영향을 판별할 수 있습니다.

프로시저

1. CREATE TABLE문을 공식화하십시오.
   - 행 압축을 사용하려는 경우 COMPRESS YES절을 사용하십시오.
   - 값 압축을 사용하려는 경우 VALUE COMPRESSION절을 포함하십시오. 시스템 기본값을 압축하려는 경우 COMPRESS SYSTEM DEFAULT절을 포함하십시오.
2. CREATE TABLE문을 실행하십시오.

테이블이 작성된 후, 나중에 테이블에 추가되는 모든 데이터가 압축됩니다. 사용자가 명시적으로 압축되지 않도록 지정하는 경우가 아니면, 테이블과 연관된 모든 인덱스도 압축됩니다.

예

예 1: 행 압축이 사용 가능한 고객 정보에 대한 테이블 작성

```
CREATE TABLE CUSTOMER
     (CUSTOMERNUM      INTEGER,
      CUSTOMERNAME     VARCHAR(80),
      ADDRESS          VARCHAR(200),
      CITY             VARCHAR(50),
      COUNTRY          VARCHAR(50),
      CODE             VARCHAR(15),
      CUSTOMERNUMDIM   INTEGER)
      COMPRESS YES;
```

*예 2:*SALARY 필드에 대해 행 및 시스템 기본값 압축이 사용 가능하고 급여 필드에 대해 디폴트 0이 가정되는 직원 급여에 대한 테이블 작성

```
CREATE TABLE EMPLOYEE_SALARY
          (DEPTNO    CHAR(3)     NOT NULL,
     DEPTNAME VARCHAR(36)  NOT NULL,
     EMPNO    CHAR(6)      NOT NULL,
     SALARY   DECIMAL(9,2) NOT NULL WITH DEFAULT COMPRESS SYSTEM DEFAULT)
     COMPRESS YES;
```

그러나 VALUE COMPRESSION 절이 이 명령에서 생략되었음을 주의하십시오. 이 명령은 EMPLOYEE_SALARY라는 테이블을 작성하지만, 다음 경고 메시지가 리턴됩니다.

```
SQL20140W  COMPRESS column attribute ignored because VALUE COMPRESSION is
deactivated for the table.  SQLSTATE=01648
```

이 경우에 COMPRESS SYSTEM DEFAULT가 SALARY 컬럼에 실제로 적용되지 않습니다.

예 3: SALARY 필드에 대해 행 및 시스템 기본값 압축이 사용 가능하고 급여 필드에 대해 디폴트 0이 가정되는 직원 급여에 대한 테이블 작성

```
CREATE TABLE EMPLOYEE_SALARY
          (DEPTNO    CHAR(3)     NOT NULL,
      DEPTNAME VARCHAR(36)  NOT NULL,
      EMPNO    CHAR(6)      NOT NULL,
      SALARY   DECIMAL(9,2) NOT NULL WITH DEFAULT COMPRESS SYSTEM DEFAULT)
      VALUE COMPRESSION COMPRESS YES;
```

이 예에서는 VALUE COMPRESSION이 명령문에 포함되어 있으며 이로 인해 SALARY 필드에 대한 디폴트값이 압축될 수 있습니다.

기존 테이블에서 압축 사용

기존 테이블을 수정하여 ALTER TABLE 명령을 사용한 압축의 스토리지 절약 이점을 활용할 수 있습니다.

시작하기 전에

이 태스크는 현재 행 또는 값 압축을 채택하지 않은 테이블이 있고 이러한 스토리지 절약 기능 중 하나 또는 둘 다를 활성화하기 원한다고 가정합니다.

이 태스크에 대한 정보

행 압축만, 값 압축만 또는 두 유형의 압축을 모두 사용할지 여부를 결정해야 합니다. 행 압축은 한 행에서 여러 컬럼에 걸쳐있는 데이터 패턴을 더 짧은 기호 문자열로 바꾸려고 하기 때문에 거의 항상 스토리지를 절약할 수 있습니다. 값 압축은 동일한 값을 포함하는 컬럼이 있는 행이 많을 때 또는 컬럼의 데이터 유형에 대해 디폴트값을 포함하는 컬럼이 있을 때 절약을 제공할 수 있습니다. 값 압축이 사용 가능한 경우 데이터 유형에 대해 시스템 기본값을 가정하는 컬럼이 COMPRESS SYSTEM DEFAULT 옵션으로 더욱 압축될 수 있다고 지정할 수도 있습니다.

제한사항

COMPRESS SYSTEM DEFAULT절을 사용하여 시스템 기본값을 포함하는 컬럼에 압축을 적용하려는 경우 VALUE COMPRESSION도 지정해야 합니다. 그렇지 않으면 경고가 리턴되고 시스템 기본값은 최소 스페이스를 사용하여 저장되지 않습니다.

값 압축을 사용하려고 계획하는 경우, 특정 데이터 유형을 다루기 위해 데이터베이스 관리 프로그램에 의해 부과되는 오버헤드의 결과로 일부 경우에는 행 *크기*가 늘어날 수 있음을 알아야 합니다. CREATE TABLE문에 대한 문서에서 이 옵션에 대해 제공되는 정보를 사용하여 값 압축이 행 *크기*에 대해 갖는 영향을 판별할 수 있습니다.

프로시저

1. ALTER TABLE문을 공식화하십시오.
   - 행 압축을 사용하려는 경우 COMPRESS YES절을 사용하십시오.

- 값 압축을 사용하려는 경우 ACTIVATE VALUE COMPRESSION절을 포함하십시오. 시스템 기본값을 압축하려는 경우 COMPRESS SYSTEM DEFAULT 절을 포함하십시오.

2. ALTER TABLE문을 실행하십시오. 이 시점에서, 추가, 삽입, 로드 또는 갱신되는 모든 후속 행은 새로운 압축 형식을 사용합니다. 그러나 기존의 압축되지 않은 행은 계속 압축되지 않은 상태로 있습니다.
3. 선택적: 압축이 테이블의 기존 행에 적용되기 원하는 경우 REORG 명령을 사용하여 테이블 재구성을 수행하십시오. 다른 방법으로는 압축되지 않은 행이 다음에 갱신될 때까지 기다릴 수 있습니다. 해당 시점에서 변경되는 모든 행은 새 압축된 형식으로 저장됩니다.

결과

테이블을 변경했지만 REORG를 수행하지 않은 경우, 추가, 갱신, 삽입 또는 로드되는 모든 후속 행이 사용자가 사용하는 어떤 압축을 활용하는 경우에도 테이블의 기존 행 또는 컬럼의 형식은 어떤 방법으로도 수정되지 않습니다. REORG를 수행하는 경우 ALTER TABLE문을 사용하여 사용 가능하게 한 압축의 모든 유형이 테이블의 모든 행에 적용됩니다.

예

예 1: 기존 테이블 CUSTOMER에 행 압축 적용.

```
ALTER TABLE CUSTOMER      COMPRESS YES
```

예 2: 기존 테이블 EMPLOYEE_SALARY의 SALARY 컬럼에 행, 값 및 시스템 기본값 압축 적용.

```
ALTER TABLE EMPLOYEE_SALARY
ALTER SALARY COMPRESS SYSTEM DEFAULT
COMPRESS YES ACTIVATE VALUE COMPRESSION;

REORG TABLE EMPLOYEE_SALARY
```

압축된 테이블 압축 해제

값 또는 행 압축이 사용된 테이블의 압축을 해제하려면 ALTER TABLE 명령의 다양한 압축 관련 속성 중 하나 이상을 사용하십시오.

이 태스크에 대한 정보

- 값 압축을 비활성화하는 경우:
  - COMPRESS SYSTEM DEFAULT가 사용 가능한 컬럼을 갖는 테이블의 경우 이들 컬럼에 대한 압축을 더 이상 사용할 수 없습니다.

– 압축되지 않은 컬럼으로 인해 행 *크기*가 현재 테이블 스페이스의 현재 페이지 크기가 허용하는 최대값을 초과할 수 있습니다. 이 경우 오류 메시지 SQL0670N이 리턴됩니다.

- 행 압축을 비활성화하면 인덱스 압축은 영향을 받지 않습니다. 인덱스를 압축 해제하려면 경우 ALTER INDEX 명령을 사용해야 합니다.
- 행 또는 값 압축 중 하나를 비활성화하는 경우 압축된 데이터의 압축을 해제하려면 테이블 재구성을 수행해야 합니다.

프로시저

1. ALTER TABLE문을 공식화하십시오.
   - 행 압축을 비활성화하려는 경우 COMPRESS NO절을 사용하십시오.
   - 값 압축을 비활성화하려는 경우 DEACTIVATE VALUE COMPRESSION절을 포함하십시오.
   - 시스템 디폴트값의 압축을 비활성화하려는 경우 ALTER *컬럼 이름*절에 대한 COMPRESS OFF 옵션을 포함시키십시오.
2. ALTER TABLE문을 실행하십시오.
3. 오프라인 테이블 재구성을 수행하십시오.

예

예 1: 기존 테이블 CUSTOMER에서 행 압축 끄기

```
ALTER TABLE CUSTOMER      COMPRESS NO

REORG TABLE CUSTOMER
```

복제 소스 테이블의 여러 압축 사전

CREATE TABLE 및 ALTER TABLE문의 COMPRESS YES 및 DATA CAPTURE CHANGES 옵션을 사용하여 복제의 소스 테이블인 테이블에 대해 행을 압축할 수 있습니다. 이러한 옵션을 사용하여 REORG 또는 LOAD REPLACE 조작을 통해 소스 테이블에 활성 *데이터 압축 사전* 및 *히스토리 압축 사전*이라는 두 개의 사전을 포함할 수 있습니다.

히스토리 압축 사전은 데이터 압축 사전의 이전 버전입니다. 잠재적 REORG 또는 절단 조작으로 인해 로그 판독기가 현재 활동보다 지연될 때마다 히스토리 압축 사전이 필요합니다. REORG 또는 LOAD에서 RESETDICTIONARY의 결과로 최신 사전이 작성된 경우 또는 행 컨텐츠가 REORG보다 이전에 기록된 경우 히스토리 압축 사전은 db2ReadLog API가 로그 레코드의 행 컨텐츠를 압축 해제하도록 허용합니다.

주: 로그 판독기가 로그 레코드 내의 데이터를 압축되지 않은 형식으로 리턴하도록 하려면 원시 압축 형식 대신 db2ReadLog API의 **iFilterOption** 매개변수를 DB2READLOG_FILTER_ON으로 설정해야 합니다.

REORG TABLE 및 테이블 절단 조작(LOAD REPLACE, IMPORT REPLACE 또는 TRUNCATE TABLE) 중 테이블의 히스토리 압축 사전이 제거됩니다. 그러나 테이블에 대해 DATA CAPTURE NONE 옵션이 지정된 경우에만 제거됩니다.

전체 사전 *크기*(바이트)를 확인하려면 ADMIN_GET_TAB_INFO_V97 테이블 함수 또는 ADMIN_GET_TAB_COMPRESS_INFO 테이블 함수의 REPORT 옵션을 사용합니다. 자세한 내용은 이러한 테이블 함수를 참조하십시오.

낙관적 잠금 개요

확장 낙관적 잠금 지원은 행 선택, 갱신 또는 삭제 중에 행 잠금을 보유하지 않는 SQL 데이터베이스 응용프로그램에 대한 기술을 제공합니다.

응용프로그램은 잠기지 않은 행은 갱신 또는 삭제 조작 전에 변하지 않을 것이라고 낙관적으로 가정하여 기록됩니다. 행이 변하는 경우 갱신 또는 삭제는 실패하며 응용프로그램의 논리가 예를 들어 선택을 재시도하여 그런 실패를 처리할 수 있습니다.

이 확장 낙관적 잠금의 장점은 다른 응용프로그램이 동일한 행을 읽고 쓸 수 있으므로 동시성이 향상되는 것입니다. 비즈니스 트랜잭션이 데이터베이스 트랜잭션과 상관 관계를 갖지 않는 3티어 환경에서는 비즈니스 트랜잭션 사이에서 잠금을 유지할 수 없기 때문에 이 낙관적 잠금 기술이 사용됩니다.

표 17에는 각 범주의 관련 항목이 나열되어 있습니다.

표 17. 낙관적 잠금 정보의 개요

범주	관련 항목
일반 정보 및 제한사항	• 288 페이지의『낙관적 잠금』 • 291 페이지의『행 변경 토큰 및 거짓 부정(false negative)의 세분화도』 • 290 페이지의『낙관적 잠금 제한사항 및 고려사항』
시간별 갱신	• 292 페이지의『시간별 갱신 발견』 • 293 페이지의『ROW CHANGE TIMESTAMP에 대해 생성되는 시간 값』
사용 가능	• 296 페이지의『낙관적 잠금 사용 계획』 • 297 페이지의『응용프로그램에서 낙관적 잠금 사용』
사용 시나리오	• 315 페이지의『시나리오: 낙관적 잠금 및 시간별 발견』 – 315 페이지의『시나리오: 응용프로그램에서 낙관적 잠금 사용』 – 318 페이지의『시나리오: 시간별 갱신 발견』 – 317 페이지의『시나리오: 내재적으로 숨겨진 컬럼을 사용하는 낙관적 잠금』

표 17. 낙관적 잠금 정보의 개요 (계속)

범주	관련 항목
참조 정보	• 294 페이지의 『RID_BIT() 및 RID() 내장 함수』 • *SQL 참조서, 볼륨 1*의 ALTER TABLE문 • *SQL 참조서, 볼륨 2*의 CREATE TABLE문 • *SQL 참조서, 볼륨 2*의 DELETE문 • *SQL 참조서, 볼륨 2*의 SELECT문 • *SQL 참조서, 볼륨 2*의 UPDATE문

주: 낙관적 잠금 주제 전체에서 행이 삽입 또는 갱신되는 것으로 참조될 때마다 이것은 행이 어떤 방법으로든 테이블에 삽입되거나 갱신되도록 만들 수 있는 SQL문의 모든 양식을 의미합니다. 예를 들어 INSERT, UPDATE, MERGE 또는 DELETE문(참조 제한조건 포함)도 모두 시간소인 컬럼이 작성 또는 갱신되도록 할 수 있습니다.

낙관적 잠금

*낙관적 잠금*은 행 선택 및 갱신 또는 삭제 중에 행 잠금을 보유하지 않는 SQL 데이터베이스 응용프로그램에 대한 기술입니다.

응용프로그램은 잠기지 않은 행은 갱신 또는 삭제 조작 전에 변하지 않을 것이라고 낙관적으로 가정하여 기록됩니다. 행이 변하는 경우 갱신 또는 삭제는 실패하며 응용프로그램 논리가 예를 들어 선택을 재시도하여 그런 실패를 처리합니다. 낙관적 잠금의 한 가지 장점은 다른 응용프로그램이 해당 행을 읽고 쓸 수 있기 때문에 동시성이 향상되는 것입니다. 비즈니스 트랜잭션이 데이터베이스 트랜잭션과 상관 관계를 갖지 않는 3 티어 환경에서는 비즈니스 트랜잭션 사이에서 잠금을 유지할 수 없기 때문에 낙관적 잠금 기술이 사용됩니다.

그러나 값에 의한 낙관적 잠금은 다음과 같은 몇 가지 단점이 있습니다.

- 추가 데이터 서버 지원 없이 *거짓 긍정(false positive)*이 될 수 있습니다. 낙관적 잠금을 사용할 때 선택된 이후에 *변경*되는 행이 먼저 해당 행을 다시 선택하지 않으면 갱신될 수 있는 상태입니다. (이것은 *거짓 부정(false negative)*과 대비될 수 있는데, 선택된 이후 *변하지* 않는 행은 먼저 다시 선택하지 않으면 갱신할 수 없는 상태입니다.)
- 응용프로그램에서 더 많은 재시도 논리가 필요함
- 응용프로그램이 UPDATE 검색 조건을 빌드하기 위해 복잡함
- DB2 서버가 값을 기초로 대상 행을 검색하기에 비효율적임
- 데이터 유형이 일부 클라이언트 유형과 데이터베이스 유형 사이에서 불일치합니다. 예를 들어 시간소인은 모든 컬럼이 검색된 갱신에서 사용되지 못하게 합니다.

*거짓 긍정(false positive)*을 갖지 않는 더 쉽고 빠른 낙관적 잠금에 대한 지원은 다음의 새 SQL 함수, 표현식 및 기능을 갖습니다.

- 행 ID(RID_BIT 또는 RID) 내장 함수
- ROW CHANGE TOKEN 표현식
- 시간별 갱신 발견
- 내재적으로 숨겨진 컬럼

DB2 응용프로그램은 선택된 것과 정확하게 동일한 값을 갖는 행을 찾는 검색된 UPDATE문을 빌드하여 *값에 의한 낙관적 잠금*을 사용할 수 있습니다. 검색된 UPDATE는 행의 컬럼 값이 변경된 경우 실패합니다.

이 프로그래밍 모델을 사용하는 응용프로그램은 확장된 낙관적 잠금 기능을 활용합니다. 이 프로그래밍 모델을 사용하지 않는 응용프로그램은 낙관적 잠금 응용프로그램으로 간주되지 않으며 계속해서 이전과 같이 작동합니다.

행 ID(RID_BIT 또는 RID) 내장 함수

이 내장 함수는 SELECT 목록 또는 술어 명령문에서 사용할 수 있습니다. 술어에서(예: WHERE RID_BIT(tab)=?), RID_BIT 등호 술어는 행을 효과적으로 찾기 위한 새로운 직접 액세스 메소드로서 구현됩니다. 이전에는 모든 선택된 컬럼 값을 술어에 추가하고 일부 고유 컬럼 조합에 의존하여 단일 행만을 규정함으로써 덜 효율적인 액세스 메소드를 사용하여 소위 값을 갖는 값 낙관적 잠금이 수행되었습니다.

ROW CHANGE TOKEN 표현식

이 새 표현식은 토큰을 BIGINT로 리턴합니다. 토큰은 행의 수정 시퀀스에서 상대적 위치를 나타냅니다. 응용프로그램은 행의 현재 ROW CHANGE TOKEN 값을 행이 마지막으로 페치되었을 때 저장된 ROW CHANGE TOKEN 값과 비교하여 행이 변경되었는지 판별할 수 있습니다.

시간별 갱신 발견

이 기능은 RID_BIT() 및 ROW CHANGE TOKEN을 사용하여 SQL에 추가됩니다. 이 기능을 지원하려면 테이블에 시간소인 값을 저장하도록 정의된 새로 생성된 컬럼이 있어야 합니다. ALTER TABLE문을 사용하여 기존 테이블에 추가할 수 있거나 새 테이블을 작성할 때 컬럼을 정의할 수 있습니다. 컬럼의 존재도 페이지 레벨에서 행 레벨로 ROW CHANGE TOKEN의 세분화도를 개선하는 데 사용되는 경우 해당 컬럼의 낙관적 잠금 동작에 영향을 주는데, 이 경우 낙관적 잠금 응용프로그램에 크게 도움이 될 수 있습니다. 이 기능은 z/OS®용 DB2에도 추가되었습니다.

내재적으로 숨겨진 컬럼:

호환성을 위해 이 기능은 기존 테이블 및 응용프로그램에 대한 RID_BIT 및 ROW CHANGE TOKEN 컬럼 채택을 용이하게 합니다. 내재적으로 숨겨진 컬럼은 내재된 컬럼 목록이 사용될 때 구체화되지 않습니다. 예를 들어,

- 테이블에 대한 SELECT *는 결과 테이블에 내재적으로 숨겨진 컬럼을 리턴하지 않습니다.

- 컬럼 목록이 없는 INSERT문은 내재적으로 숨겨진 컬럼에 대한 값을 예상하지 않지만, 해당 컬럼이 NULL 값을 허용하거나 다른 디폴트값을 갖도록 정의되어야 합니다.

주: *최적 동시처리 제어*, *비관적 잠금*, *ROWID* 및 *갱신 발견* 같은 낙관적 잠금 용어의 정의는 DB2 용어집을 참조하십시오.

낙관적 잠금 제한사항 및 고려사항

이 주제는 사용자가 알아야 하는 낙관적 잠금 제한사항을 나열합니다.

- ROW CHANGE TIMESTAMP 컬럼은 다음 키, 컬럼 및 이름에서 지원되지 않습니다(사용되면 sqlstate 429BV가 리턴됨).
  - 기본 키
  - 외부 키
  - 다차원 클러스터된(MDC) 컬럼
  - 범위 파티션 컬럼
  - 데이터베이스 해시 파티션 키
  - DETERMINED BY 제한 컬럼
  - 별칭
- RID() 함수는 파티션된 데이터베이스 구성에서 지원되지 않습니다.
- 낙관적 잠금 시나리오에서 페치 및 갱신 조작 사이에 수행된 온라인 또는 오프라인 테이블 REORG는 갱신이 실패하도록 만들 수 있지만, 이것은 일반 응용프로그램 재시도 논리에 의해 처리되어야 합니다.
- IMPLICITLY HIDDEN 속성은 낙관적 잠금에 대한 ROW CHANGE TIMESTAMP 컬럼만으로 제한됩니다.
- 현장 재구성은 모든 행이 구체화되었다고 보증될 때까지 ROW CHANGE TIMESTAMP 컬럼이 기존 테이블에 추가된 테이블에 대해 제한됩니다(이 오류의 경우 SQL2219, 이유 코드 13이 리턴됨). 이것은 LOAD REPLACE 명령을 사용하여 클래식 테이블 재구성으로 수행될 수 있습니다. 이것은 *거짓 긍정(false positive)*을 예방합니다. ROW CHANGE TIMESTAMP 컬럼으로 작성된 테이블은 제한사항을 갖지 않습니다.

내재적으로 숨겨진 컬럼에 대한 고려사항

IMPLICITLY HIDDEN으로 정의되는 컬럼은 SELECT 목록에서 *를 지정하는 쿼리의 결과 테이블의 파트가 아닙니다. 그러나 내재적으로 숨겨진 컬럼은 쿼리에서 명시적으로 참조될 수 있습니다.

컬럼 목록이 삽입 시 지정되지 않으면 삽입에 대한 VALUES절이나 SELECT LIST가 이 컬럼을 포함하지 않아야 합니다(일반적으로 생성된, 디폴트 가능 또는 널(NULL) 입력 가능 컬럼이어야 함).

예를 들어 내재적으로 숨겨진 컬럼은 SELECT 목록이나 쿼리의 술어에서 참조될 수 있습니다. 또한 내재적으로 숨겨진 컬럼은 CREATE INDEX문, ALTER TABLE문, INSERT문, MERGE문 또는 UPDATE문에서 명시적으로 참조될 수 있습니다. 내재적으로 숨겨진 컬럼은 참조 제한조건에서 참조될 수 있습니다. 컬럼 목록을 포함하지 않는 REFERENCES절은 상위 테이블의 기본 키를 내재적으로 참조합니다. 상위 테이블의 기본 키가 내재적으로 숨겨진 것으로 정의되는 컬럼을 포함할 수 있습니다. 그런 참조 제한조건이 허용됩니다.

- 구체화된 쿼리 정의의 fullselect의 SELECT 목록이 내재적으로 숨겨진 컬럼을 명시적으로 참조하는 경우 해당 컬럼이 구체화된 쿼리 테이블의 파트가 됩니다. 그렇지 않으면 내재적으로 숨겨진 컬럼은 내재적으로 숨겨진 컬럼을 포함하는 테이블을 참조하는 구체화된 쿼리 테이블의 파트가 아닙니다.
- 뷰 정의(CREATE VIEW문)의 fullselect의 SELECT 목록이 내재적으로 숨겨진 컬럼을 명시적으로 참조하는 경우 해당 컬럼은 뷰의 파트가 됩니다(그러나 뷰 컬럼은 숨겨진 것으로 간주되지 않음). 그렇지 않으면 내재적으로 숨겨진 컬럼은 내재적으로 숨겨진 컬럼을 포함하는 테이블을 참조하는 뷰의 파트가 아닙니다.

레이블 기반 액세스 제어(LBAC)에 대한 고려사항

컬럼이 LBAC에서 보호되는 경우 해당 컬럼에 대한 사용자 액세스는 LBAC 규정 및 사용자의 보안 레이블에 의해 판별됩니다. 이 보호가 행 변경 시간소인 컬럼에 적용되는 경우 해당 컬럼에서 파생되는 ROW CHANGE TIMESTAMP 및 ROW CHANGE TOKEN 표현식을 통해 해당 컬럼에 대한 참조로 확장됩니다.

그러므로 테이블에 대한 보안 규정을 판별할 때 행 변경 시간소인 컬럼에 대한 액세스가 낙관적 잠금 또는 시간별 갱신 발견을 적절히 사용해야 하는 모든 사용자에게 사용 가능한지 확인하십시오. 행 변경 시간소인 컬럼이 없는 경우 ROW CHANGE TOKEN 표현식이 LBAC에 의해 차단될 수 없음을 주의하십시오. 그러나 테이블이 행 변경 시간소인 컬럼을 추가하도록 변경되는 경우 모든 LBAC 고려사항이 적용됩니다.

행 변경 토큰 및 거짓 부정(false negative)의 세분화도

RID_BIT() 내장 함수 및 행 변경 토큰이 낙관적 잠금을 위한 유일한 요구사항입니다. 그러나 테이블의 스키마도 낙관적 잠금의 동작에 영향을 미칩니다.

예를 들어 아래에 표시된 다음 명령문 절 중 하나를 사용하여 정의되는 행 변경 시간소인 컬럼은 DB2 서버가 행이 마지막으로 변경(또는 처음에 삽입)되는 시간을 저장하게 합니다. 이것은 행에 대한 가장 최근 변경의 시간소인을 캡처하는 방법을 제공합니다. 이것이 시간소인 컬럼이며 GENERATED BY DEFAULT절을 사용하여 사용자 제공 입력 값을 허용하지 않으면 데이터베이스 관리 프로그램에 의해 유지됩니다.

```
GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
GENERATED BY DEFAULT FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
```

그러므로 응용프로그램이 테이블에서 새 ROW CHANGE TOKEN 표현식을 사용할 때 고려할 두 가지 가능성이 있습니다.

- *테이블에 행 변경 시간소인 컬럼이 없음:* ROW CHANGE TOKEN 표현식이 동일한 페이지에 위치하는 모든 행이 공유하는 파생된 BIGINT 값을 리턴합니다. 페이지의 한 행이 갱신되는 경우 동일한 페이지의 모든 행에 대한 ROW CHANGE TOKEN이 변경됩니다. 이것은 다른 행이 변경될 때 갱신이 실패할 수 있음을 의미하며, 거짓 부정(false negative)이라는 등록 정보입니다.

  주: 응용프로그램이 *거짓 부정(false negative)*을 허용할 수 있으며 ROW CHANGE TIMESTAMP 컬럼에 대해 각 행에 추가 스토리지를 추가하지 않으려는 경우에만 이 모드를 사용하십시오.
- *테이블에 행 변경 시간소인 컬럼이 있음:* ROW CHANGE TOKEN 표현식이 컬럼의 시간소인 값에서 파생되는 BIGINT 값을 리턴합니다. 이 경우에는 *거짓 부정(false negative)*이 발생할 수 있지만 덜 빈번합니다. 테이블이 재구성 또는 재분배되는 경우 *거짓 부정(false negative)*는 행이 제거되고 응용프로그램이 이전 RID_BIT() 값을 사용하는 경우에 발생할 수 있습니다.

시간별 갱신 발견

일부 응용프로그램은 특정 시간 범위에 대한 데이터베이스 갱신을 알아야 하며, 시간 범위를 데이터 복제, 시나리오 감사 등에 사용될 수 있습니다. ROW CHANGE TIMESTAMP 표현식이 이 정보를 제공합니다.

```
ROW CHANGE TIMESTAMP FOR <table designator>
```

는 CURRENT TIMESTAMP와 비슷한 로컬 시간으로 표현되며 행이 마지막으로 변경된 시간을 나타내는 시간소인을 리턴합니다. 갱신된 행의 경우 이것은 행에 대한 가장 최근 갱신을 반영합니다. 그렇지 않으면 값이 행의 원래 삽입에 대응합니다.

데이터베이스 파티션별로 행별로 데이터베이스에 의해 지정될 때 고유하다고 보증된다는 점에서 ROW CHANGE TIMESTAMP의 값은 CURRENT TIMESTAMP와 다릅니다. 삽입 또는 갱신되는 각 개별 행의 수정 시간의 로컬 시간소인 근사입니다. 값이 항상 초기에서 나중으로 늘어나고 있으므로, 다음 경우에 시스템 클럭과 동기화되지 않을 수 있습니다.

- 시스템 클럭이 변경됩니다.
- 행 변경 시간소인 컬럼이 GENERATED BY DEFAULT(데이터 전파만을 위해)행에 동기화되지 않은 값이 제공됩니다.

ROW CHANGE TIMESTAMP 표현식 사용에 대한 전제조건은 테이블이 시간소인 데이터 유형 TIMESTAMP(6)에 대한 디폴트 정밀도(또는 TIMESTAMP - 디폴트 정밀

도는 6)를 사용하여 행 변경 시간소인 컬럼이 정의되는 것입니다. 모든 행은 행이 삽입 또는 마지막 갱신된 시간의 시간소인을 리턴합니다. 행 변경 시간소인 컬럼이 테이블의 파트가 될 수 있는 두 가지 방법이 있습니다.

- 테이블이 CREATE TABLE 명령의 FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP절을 사용하여 작성되었습니다. ROW CHANGE TIMESTAMP 표현식이 컬럼의 값을 리턴합니다. 이 범주의 경우 시간소인이 정확합니다. 일반적으로 데이터베이스에 의해 생성될 때 행 변경 시간소인은 삽입 속도 및 DST 조정을 포함한 가능한 클럭 조작에 의해 제한됩니다.
- 테이블이 행 변경 시간소인 컬럼과 함께 작성되지 않았지만, 하나가 나중에 ALTER TABLE문의 FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP절을 사용하여 추가되었습니다. ROW CHANGE TIMESTAMP 표현식이 컬럼의 값을 리턴합니다. 이 범주의 경우 이전(변경 전) 행에는 먼저 갱신되거나 오프라인 테이블 재구성이 수행될 때까지 실제 시간소인이 없습니다.

  주: 시간소인은 해당 시점에서의 시스템 클럭 및 데이터베이스/테이블 파티션 내부에서 시간소인이 반복될 수 없다는 한계를 고려하여 데이터베이스에서 실제 갱신이 발생한 시간의 근사치입니다. 실제로 이것은 대개 갱신 시간의 매우 정확한 표시입니다. 행 변경 시간소인은 일반적으로 데이터베이스에 의해 생성될 때 삽입 속도 및 DST 조정을 포함한 가능한 클럭 조작에 의해 제한됩니다.

  ALTER TABLE문 이후 갱신되지 않은 행은 컬럼에 대한 유형 디폴트 값을 리턴하며, 이것은 1년 1월 1일입니다. 갱신된 행만 고유 시간소인을 갖습니다. 오프라인 테이블 재구성을 통해 구체화된 시간소인을 갖는 행은 테이블의 재구성 중에 생성된 고유 시간소인을 리턴합니다. INPLACE 옵션을 사용한 Reorg는 스키마 변경을 구체화하지 않기 때문에 충분하지 않습니다.

어느 경우에나 행의 시간소인은 재분배가 수행되는 경우에도 갱신됩니다. 재분배 중에 행이 한 데이터베이스 파티션에서 다른 파티션으로 이동되는 경우 목표에서 고유한 것으로 보장되는 새 시간소인이 생성되어야 합니다.

ROW CHANGE TIMESTAMP에 대해 생성되는 시간 값

파티션당 고유 값 강제 실행으로 인해 행 변경 시간소인 컬럼에 대해 생성되는 정확한 값에 대한 몇 가지 경계 조건이 있습니다.

시스템 클럭이 클럭 교정 또는 DB2 서버에 대한 일광 절약 시간 규정 때문에 과거로 조정될 때마다, 시간소인이 시스템 클럭의 현재 값 또는 CURRENT TIMESTAMP 특수 레지스터의 값에 상대적으로 미래에 있는 것으로 나타날 수 있습니다. 이것은 시간소인이 항상 고유성을 유지하기 위해 오름차순으로 생성되므로 시간소인이 시스템 클럭 조정 이전, 즉 조정된 시간보다 늦게 생성되었을 때 발생합니다.

REORG 조작에 의해 또는 LOAD 조작의 파트로서 테이블에 추가된 컬럼에 대해 시간소인이 생성될 때, 시간소인은 초기 시간소인 값에서 시작하여 유틸리티의 처리의 특

정 지점에 순차적으로 생성됩니다. 유틸리티가 시간소인 세분화도(즉, 초당 1백만 행 이상)보다 빨리 행을 처리할 수 있는 경우, 일부 행에 대해 생성되는 값도 시스템 클럭 또는 CURRENT TIMESTAMP 특수 레지스터에 상대적으로 미래에 있는 것으로 나타날 수 있습니다.

각 경우에 시스템 클럭이 행 변경 시간소인 값까지 따라잡은 후 행이 삽입된 시간에 가까운 근사입니다. 그런 시간까지 오름차순은 timestamp(6) 데이터 유형에서 허용하는 가장 미세한 세분화도에 의해 오름차순으로 생성됩니다.

RID_BIT() 및 RID() 내장 함수

RID_BIT() 및 ROW CHANCE TOKEN은 테이블의 모든 행에 대해 선택할 수 있습니다. SELECT는 응용프로그램에서 필요한 분리 레벨에서 발생할 수 있습니다.

응용프로그램이 다음을 둘 다 검색하여 낙관적 잠금을 갖는 동일한(변경되지 않은) 행을 수정할 수 있습니다.

- 목표 행을 직접 액세스(스캔하지 않음)하는 RID_BIT()
- 이것이 동일한 변경되지 않은 행임을 보장하는 ROW CHANGE TOKEN

이 갱신(또는 삭제)은 선택 이후의 모든 시점, 동일한 작업 단위(UOW) 내부에서 또는 연결 경계를 넘어서도 발생할 수 있습니다. 유일한 요구사항은 임의의 특정 시점에서 주어진 행에 대해 위의 두 값을 얻은 것입니다.

낙관적 잠금은 “WebSphere-Oriented Programming Model”에서 사용됩니다. 예를 들어 Microsoft .NET은 이 모델을 사용하여 UPDATE 또는 DELETE문이 뒤에 오는 SELECT문을 다음과 같이 처리합니다.

- 데이터베이스 서버에 연결하고 테이블에서 파생된 행 선택(SELECT)
- 데이터베이스의 연결을 끊거나 행 잠금을 해제하여 다른 응용프로그램이 해당 응용프로그램이 보유하는 잠금 및 자원으로 인한 모든 동시성 충돌 없이 데이터를 읽기, 갱신, 삭제 및 삽입할 수 있습니다(분리 『언커미트된 읽기』는 더 높은 동시성 및 다른 응용프로그램이 갱신 및 삭제 트랜잭션을 커미트(COMMIT)하는 것으로 가정을 허용하므로, 이 낙관적 잠금 응용프로그램은 갱신된 값을 읽고 최적 검색 갱신/삭제가 성공함).
- 선택된SELECT)된 행 데이터에 대해 로컬 계산 수행
- 데이터베이스 서버에 다시 연결하고, 하나 이상의 특정 목표 행에서 UPDATE 또는 DELETE 검색(및 목표 행이 변경된 경우 실패한 UPDATE 또는 DELETE문 처리)

이 프로그래밍 모델을 사용하는 응용프로그램은 확장된 낙관적 잠금 기능을 활용합니다. 이 프로그래밍 모델을 사용하지 않는 응용프로그램은 낙관적 잠금 응용프로그램으로 간주되지 않으며 계속해서 이전과 같이 작동합니다.

RID_BIT() 및 RID() 내장 함수 기능

다음은 향상된 낙관적 잠금 및 갱신 발견을 위해 구현되는 새 기능입니다.

RID_BIT(<테이블 지정자>)

행의 레코드 ID(RID)를 VARCHAR(16) FOR BIT DATA로서 리턴하는 새 내장 함수입니다.

주: z/OS용 DB2는 내장 함수 RID를 리턴 유형 BIGINT로 구현하지만, Linux, UNIX Windows RID에는 충분히 크지 않습니다. 호환성을 위해 이 RID() 내장 함수는 RID_BIT() 외에 BIGINT를 리턴합니다.

이 RID() 내장 함수는 파티션된 데이터베이스 환경에서 작동하지 않으며 테이블 버전 정보를 포함하지 않습니다. 그렇지 않으면 RID_BIT와 동일하게 작업합니다. z/OS 서버에 게시될 응용프로그램을 코딩할 때만 사용해야 합니다. 필요한 경우를 제외하면 이 주제는 RID_BIT만을 참조합니다.

RID_BIT() 내장 함수

이 내장 함수는 SELECT 목록 또는 술어 명령문에서 사용할 수 있습니다. 술어에서(예: `WHERE RID_BIT(tab)=?,`) RID_BIT 등호 술어는 행을 효과적으로 찾기 위한 새로운 직접 액세스 메소드로서 구현됩니다. 이전에는 모든 선택된 컬럼 값을 술어에 추가하고 일부 고유 컬럼 조합에 의존하여 단일 행만을 규정함으로써 덜 효율적인 액세스 메소드를 사용하여 소위 *값을 갖는 낙관적 잠금* 값이 수행되었습니다.

ROW CHANGE TOKEN FOR <테이블 지정자>

토큰을 BIGINT로 리턴하는 새 표현식입니다. 이 토큰은 행의 수정 시퀀스에서 상대적 위치를 나타냅니다. 응용프로그램은 행의 현재 ROW CHANGE TOKEN 값을 행이 마지막으로 페치되었을 때 저장된 ROW CHANGE TOKEN 값과 비교하여 행이 변경되었는지 판별할 수 있습니다.

ROW CHANGE TIMESTAMP 컬럼

다음 중 하나로 정의할 수 있는 TIMESTAMP의 디폴트 유형을 갖는 GENERATED 컬럼:

```
GENERATED ALWAYS FOR EACH ROW ON UPDATE
AS ROW CHANGE TIMESTAMP
```

또는 (데이터 전파 또는 언로드 및 다시 로드 조작에 대해서만 제안됨):

```
GENERATED BY DEFAULT FOR EACH ROW ON UPDATE
AS ROW CHANGE TIMESTAMP
```

이 컬럼의 데이터는 행이 변경될 때마다 변경됩니다. 이 컬럼이 정의될 때 ROW CHANGE TOKEN 값이 해당 컬럼에서 파생됩니다. GENERATED ALWAYS

가 사용될 때 데이터베이스 관리 프로그램이 이 값이 데이터베이스 파티션 또는 테이블 파티션 내부에서 고유함을 보장하여 *거짓 긍정(false positive)*이 불가능함을 보장합니다.

처음 두 요소 RID_BIT 및 ROW CHANGE TOKEN을 사용하기 위해 데이터베이스 스키마를 변경할 필요가 없습니다. 그러나 ROW CHANGE TIMESTAMP 컬럼이 없으면 ROW CHANGE TOKEN이 동일한 페이지의 모든 행에 의해 공유됩니다. 페이지의 임의의 행을 갱신하면 동일한 페이지에 저장된 다른 행에 대한 *거짓 부정(false negative)*이 발생할 수 있습니다. 이 컬럼을 사용할 때 ROW CHANGE TOKEN은 시간소인에서 파생되며 테이블 또는 데이터베이스 파티션의 다른 어떤 행과도 공유되지 않습니다. 291 페이지의 『행 변경 토큰 및 거짓 부정(false negative)의 세분화도』의 내용을 참조하십시오.

시간별 갱신 발견 및 RID_BIT(), RID() 함수

ROW CHANGE TIMESTAMP 표현식은 테이블 지정자로 식별되는 테이블의 행이 마지막으로 변경되었을 때를 표시하는 시간소인 값을 리턴합니다. RID_BIT() 및 RID() 내장 함수와 시간별 갱신 발견 기능의 내부 관계에도 불구하고, ROW CHANGE TOKEN 및 ROW CHANGE TIMESTAMP 표현식의 사용은 상호 교환할 수 없음을 주의하는 것이 중요합니다. 특히 ROW CHANGE TIMESTAMP 표현식이 낙관적 잠금 사용의 파트가 아님을 주의하십시오.

낙관적 잠금 사용 계획

낙관적 잠금을 위한 새 SQL 표현식 및 속성을 테이블에 대한 DDL 변경 없이 사용할 수 있으므로 테스트 응용프로그램에서 낙관적 잠금을 쉽게 시도할 수 있습니다.

DDL 변경이 없으면 낙관적 잠금 응용프로그램은 DDL 변경이 있는 경우보다 더 많은 *거짓 부정(false negative)*을 가져올 수 있습니다. 거짓 부정(false negative)을 유발하는 응용프로그램은 프로덕션 환경에서 확장성이 좋지 않은데 이는 거짓 부정(false negative)로 인해 재시도가 너무 많이 발생하기 때문입니다. 그러므로 거짓 부정(false negative)을 피하려면 낙관적 잠금 목표 테이블이 다음 중 하나여야 합니다.

- ROW CHANGE TIMESTAMP 컬럼과 함께 작성됩니다.
- ROW CHANGE TIMESTAMP 컬럼을 포함하도록 변경됩니다.

권장 DDL 변경이 수행되는 경우 거짓 부정(false negative)은 거의 발생하지 않습니다. 유일한 거짓 부정(false negative)은 다른 행에서 동작하는 동시 응용프로그램이 아니라 reorg 같은 테이블 레벨 조작으로 인해 발생합니다.

일반적으로 데이터베이스 관리 프로그램은 거짓 부정(false negative)(예: 온라인 또는 오프라인 reorg)을 허용하며 행 변경 시간소인 컬럼이 있어 페이지 또는 행 레벨 단위의 사용 여부를 판별할 수 있습니다. 또한 ROWCHANGETIMESTAMP 컬럼에서 YES를 갖는 행이 있는 테이블에 대한 SYSCAT.COLUMNS를 쿼리할 수 있습니다.

응용프로그램 및 데이터베이스의 철저한 분석에서 예를 들어 페이지당 행이 하나인 경우 또는 갱신 또는 삭제 조작이 동일한 데이터 페이지에서 매우 드물거나 전혀 없는 경우 이 DDL이 필수가 아님을 나타낼 수 있습니다. 그런 분석은 예외입니다.

갱신 시간소인 발견 사용 시 테이블에 대한 DDL을 변경해야 하며 가능하면 테이블을 재구성하여 값을 구체화해야 합니다. 이러한 변경이 프로덕션 데이터베이스에 부정적인 영향을 미칠 수 있는 경우 먼저 테스트 환경에서 변경사항에 대한 프로토타입을 만들어야 합니다. 인스턴스의 경우 추가 컬럼이 행 크기 제한 및 플랜 선택에 영향을 줄 수 있습니다.

알아야 하는 조건

- 시스템 클럭 및 시간소인 값의 세분화도에 관한 조건을 알아야 합니다. 테이블에 ROW CHANGE TIMESTAMP 컬럼이 있으면, 삽입 또는 갱신 후에 해당 데이터베이스 파티션의 해당 테이블에서 새 행은 고유한 ROW CHANGE TIMESTAMP 값을 갖게 됩니다.
- 고유성을 보장하기 위해 시스템 클럭이 역방향으로 조정되는지 여부 또는 데이터의 갱신 또는 삽입이 시간소인 세분화도보다 더 빨리 발생하고 있는지 여부와 상관없이 행의 생성된 시간소인은 항상 늘어납니다. 그러므로 ROW CHANGE TIMESTAMP가 시스템 시간 및 DB2의 CURRENT TIMESTAMP 특수 레지스터와 비교하여 미래에 있을 수 있습니다. 시스템 클럭이 완전히 동기 상태를 벗어나거나 데이터베이스 관리 프로그램이 초당 1백만 이상의 행을 삽입 또는 갱신 중인 경우 이것은 일반적으로 실제 시간에 매우 가까워야 합니다. CURRENT TIMESTAMP와 대조적으로 이 값은 갱신 시점에서 행별로 생성되므로 일반적으로 CURRENT TIMESTAMP보다 훨씬 더 가까우며, 영향을 받는 행의 수와 복잡도에 따라서 완료하는 데 매우 긴 시간이 걸릴 수 있는 전체 명령에 대해 한 번 생성됩니다.

응용프로그램에서 낙관적 잠금 사용

응용프로그램에서 낙관적 잠금 지원을 사용하기 위해 수행해야 하는 많은 단계가 있습니다.

1. 초기 쿼리에서, 처리해야 하는 각 행에 대해 행 ID(294 페이지의 『RID_BIT() 및 RID() 내장 함수』 사용) 및 ROW CHANGE TOKEN을 선택하십시오.
2. 행 잠금을 해제하여 다른 응용프로그램이 테이블에서 SELECT, INSERT, UPDATE 및 DELETE할 수 있도록 하십시오.
3. 잠기지 않은 행이 원래 SELECT문 이후에 변경되지 않았다고 낙관적으로 가정하면서, 검색 조건에서 행 ID와 ROW CHANGE TOKEN을 사용하여 목표 행에 대해 검색된 UPDATE 또는 DELETE를 수행하십시오.
4. 행이 변경된 경우, UPDATE 조작은 실패하며 응용프로그램 논리가 실패를 처리해야 합니다. 예를 들어 응용프로그램이 SELECT 및 UPDATE 조작을 재시도합니다.

위의 단계를 실행한 경우 다음과 같습니다.

- 응용프로그램이 수행하는 재시도 횟수가 예상하거나 원하는 것보다 높을 경우, 테이블에 행 변경 시간소인 컬럼을 추가하여 RID_BIT 함수로 식별되는 행에 대한 변경이 ROW CHANGE TOKEN만을 무효화하고 동일한 데이터 페이지에 다른 활동은 무효화하지 않습니다.
- 주어진 시간 범위에서 삽입 또는 갱신된 행을 보려면 테이블을 작성하거나 변경하여 행 변경 시간소인 컬럼을 포함하게 하십시오. 이 컬럼은 데이터베이스 관리 프로그램에 의해 자동으로 유지되며 컬럼 이름 또는 ROW CHANGE TIMESTAMP 표현식 중 하나를 사용하여 쿼리할 수 있습니다.
- 행 변경 시간소인 컬럼의 경우에만, 컬럼이 IMPLICITLY HIDDEN 속성으로 정의되는 경우 컬럼은 테이블의 컬럼에 대한 내재된 참조가 있을 때 외부화되지 않습니다. 그러나 내재적으로 숨겨진 컬럼은 항상 SQL문에서 명시적으로 참조할 수 있습니다. 이것은 테이블에 컬럼을 추가할 때 내재된 컬럼 목록을 사용하는 기존 응용프로그램이 실패할 수 있는 경우 유용할 수 있습니다.

테이블 파티션 및 데이터 구성 스킴

테이블 파티션은 테이블의 하나 이상의 파티션 테이블에 있는 값에 따라서 테이블 데이터가 여러 데이터 파티션에 분배되어 있는 데이터 구성 스킴입니다. 주어진 테이블의 데이터가 여러 스토리지 오브젝트로 파티션되며, 이것은 여러 테이블 스페이스에서 존재할 수 있습니다.

테이블 파티션 및 데이터 구성 스킴에 대한 자세한 내용은 *파티셔닝 및 클러스터링 안내서*의 내용을 참조하십시오.

테이블 작성

데이터베이스 관리 프로그램은 테이블에 저장된 데이터에 대한 변경 및 액세스를 제어합니다. CREATE TABLE문을 사용하여 테이블을 작성할 수 있습니다. 복합 명령문을 사용하면 테이블의 모든 속성 및 품질을 정의할 수 있습니다. 그러나 모든 디폴트값이 사용되는 경우 테이블을 작성하는 명령문은 극히 단순합니다.

```
CREATE TABLE <table name> (<column name> <data type> <column options>,
                           (<column name> <data type> <column options>, ...)
```

<table name>은 규정자를 포함하거나 포함하지 않을 수 있습니다. 이름은 시스템 카탈로그의 모든 테이블, 뷰 및 별명 이름과 비교할 때 고유해야 합니다. 이름은 또한 SYSIBM, SYSCAT, SYSFUN 또는 SYSSTAT일 수 없습니다.

<column name>은 테이블의 컬럼 이름을 지정합니다. 이 이름은 규정될 수 없으며 테이블의 기타 컬럼 내부에서 고유해야 합니다.

컬럼에 대해 존재하는 모든 <column options>가 컬럼의 속성을 추가로 정의합니다. 옵션에는 컬럼이 널(NULL) 값을 포함하지 않도록 하기 위해 NOT NULL, LOB 데이터 유형에 대한 특정 옵션 및 참조 유형 컬럼, 컬럼에 대한 모든 제한조건 및 컬럼에 대한 모든 디폴트의 SCOPE가 포함됩니다. 자세한 정보는 CREATE TABLE문을 참조하십시오.

임시 테이블 선언

응용프로그램 내부에서 임시 테이블을 정의하려면 DECLARE GLOBAL TEMPORARY TABLE문을 사용하십시오.

임시 테이블(사용자 정의 임시 테이블이라고도 함)은 데이터베이스의 데이터에 대해 작업하는 응용프로그램이 사용합니다. 데이터 조작 결과가 테이블에 임시로 저장되어야 합니다. 사용자 임시 테이블 스페이스가 임시 테이블을 선언하기 전에 존재해야 합니다.

주: 이 테이블의 설명은 시스템 카탈로그에 나타나지 않으므로 다른 응용프로그램에 대해 지속적이지 않으며 다른 응용프로그램과 공유될 수 없습니다. 이 테이블을 사용하는 응용프로그램이 데이터베이스에서 종료되거나 연결해제될 때 테이블에 있는 임의의 데이터가 삭제되며 테이블이 내재적으로 삭제됩니다.
임시 테이블은 다음을 지원하지 않습니다.

- 사용자 정의 유형 컬럼
- LONG VARCHAR 컬럼
- 작성된 전역 임시 테이블에 대한 XML 컬럼

예 :

```
DECLARE GLOBAL TEMPORARY TABLE temptbl
         LIKE empltabl
         ON COMMIT DELETE ROWS
   NOT LOGGED      IN usr_tbsp
```

이 명령문은 temptbl이라는 임시 테이블을 정의합니다. 이 테이블은 empltabl의 컬럼과 정확하게 동일한 이름과 설명을 갖는 컬럼으로 정의됩니다. 내재된 정의만이 컬럼 이름, 데이터 유형, 널(null) 가능성 특성 및 컬럼 디폴트값 속성을 포함합니다. 고유 제한조건, 외부 키 제한조건, 트리거 및 인덱스를 포함하는 다른 모든 컬럼 속성은 정의되지 않습니다. ON COMMIT DELETE ROWS(모든 DELETE ROWS 옵션)을 사용할 때 데이터베이스 관리 프로그램은 테이블에 열린 HOLD를 갖는 커서가 있는지 여부와 상관없이 항상 행을 삭제합니다. 데이터베이스 관리 프로그램은 WITH HOLD 커서가 열리지 않은 경우 내부 TRUNCATE를 구현하여 NOT LOGGED 삭제를 최적화하며, 그렇지 않으면 데이터베이스 관리 프로그램이 한 번에 하나씩 행을 삭제합니다.

테이블은 응용프로그램이 데이터베이스에서 연결을 끊을 때 내재적으로 삭제됩니다. 자세한 정보는 DECLARE GLOBAL TEMPORARY TABLE문을 참조하십시오.

임시 테이블 작성 및 작성된 테이블에 연결

*작성된 임시 테이블*은 CREATE GLOBAL TEMPORARY TABLE문을 사용하여 작성됩니다. 처음 응용프로그램이 연결을 사용하여 작성된 임시 테이블을 참조할 때 작성된 임시 테이블의 개인용 버전이 연결을 사용하는 응용프로그램이 사용하도록 인스턴스화됩니다.

선언된 임시 테이블과 비슷하게, 작성된 임시 테이블은 데이터베이스의 데이터에 대해 작업하고 데이터 처리의 결과가 임시로 테이블에 저장되어야 하는 응용프로그램에 의해 사용됩니다. 선언된 임시 테이블 정보는 시스템 카탈로그 테이블에 저장되지 않고 사용되는 모든 세션에서 정의되어야 하는데 비해, 작성된 임시 테이블 정보는 시스템 카탈로그에 *저장되고* 사용되는 모든 세션에서 정의될 필요가 없으므로 지속적이고 서로 다른 연결 사이에서 다른 응용프로그램과 공유될 수 있습니다. 사용자 임시 테이블 스페이스는 작성된 임시 테이블을 작성할 수 있기 전에 존재해야 합니다.

주: 연결을 사용하는 모든 프로그램이 실행하는 작성된 임시 테이블에 대한 첫 번째 내재된 또는 명시적 참조가 주어진 작성된 임시 테이블의 빈 인스턴스를 작성합니다. 이 작성된 임시 테이블을 참조하는 각 연결은 작성된 임시 테이블의 고유한 인스턴스를 갖고, 인스턴스는 연결 수명 이상으로 지속되지 않습니다.

다중 연결에서 작성된 임시 테이블 이름에 대한 참조는 하나의 동일한 지속적 작성된 임시 테이블 정의 및 현재 서버에서 각 연결에 대한 작성된 임시 테이블의 고유한 인스턴스를 참조합니다. 참조되는 작성된 임시 테이블 이름이 규정되지 않는 경우 SQL문에 적용되는 표준 자격 규칙을 사용하여 내재적으로 규정됩니다.

소유자는 내재적으로 테이블 삭제 권한을 포함하여 작성된 임시 테이블에 대한 모든 테이블 특권을 갖습니다. ALL절만을 사용하여 소유자의 테이블 특권을 권한 부여 및 권한 취소할 수 있지만, 개별 테이블 특권을 권한 부여 또는 취소할 수 없습니다. 또 다른 권한 부여 ID는 적절한 특권이 권한 부여된 경우에만 작성된 임시 테이블에 액세스할 수 있습니다.

데이터를 수정하는 인덱스 및 SQL문(예: INSERT, UPDATE 및 DELETE)이 지원됩니다. 인덱스는 작성된 임시 테이블과 동일한 테이블 스페이스에만 작성될 수 있습니다.

CREATE GLOBAL TEMPORARY TABLE문의 경우 잠금 및 복구는 적용되지 않습니다. 로깅은 LOGGED절이 지정될 때만 적용됩니다. 추가 옵션에 대해서는 CREATE GLOBAL TEMPORARY문을 참조하십시오.

작성된 임시 테이블은 보안 규정과 연관될 수 없으며, 범위 파티션, 다차원 클러스터링(MDC) 또는 범위 클러스터(RCT)를 사용하여 파티션할 수 없고, 복제에 의해 분산될 수 없습니다.

작성된 임시 테이블에 대한 구체화된 쿼리 테이블(MQT)은 작성할 수 없습니다.

작성된 임시 테이블은 다음 컬럼 유형, 오브젝트 유형 및 테이블 또는 인덱스 조작을 지원하지 않습니다.

- XML 컬럼
- 구조화된 유형
- 참조된 유형
- 제한조건
- 인덱스 확장
- LOAD
- LOAD TABLE
- ALTER TABLE
- RENAME TABLE
- RENAME INDEX
- REORG TABLE
- REORG INDEX
- LOCK TABLE

자세한 정보는 CREATE GLOBAL TEMPORARY TABLE문을 참조하십시오.

예 :

```
CREATE GLOBAL TEMPORARY TABLE temptbl
         LIKE empltabl
         ON COMMIT DELETE ROWS
   NOT LOGGED      IN usr_tbsp
```

이 명령문은 temptbl이라는 임시 테이블을 작성합니다. 이 테이블은 empltabl의 컬럼과 정확하게 동일한 이름과 설명을 갖는 컬럼으로 정의됩니다. 내재된 정의는 empltabl에 있는 컬럼의 컬럼 이름, 데이터 유형, 널(null) 가능성 특성 및 컬럼 디폴트값 속성만 포함합니다. 고유 제한조건, 외부 키 제한조건, 트리거 및 인덱스를 포함한 다른 모든 컬럼 속성은 내재적으로 정의되지 않습니다.

COMMIT는 항상 테이블에서 행을 삭제합니다. 테이블에 열린 HOLD 커서가 있는 경우 TRUNCATE문을 사용하여 삭제할 수 있으며, 이것이 더 빠르지만『보통』행별로 삭제해야 합니다. 임시 테이블에 작성된 변경사항은 로그되지 않습니다. 임시 테이블은 지정된 사용자 임시 테이블 스페이스인 usr tbsp에 위치합니다. 이 테이블 스페이스가 존재해야 하며 그렇지 않으면 이 테이블의 작성에 실패합니다.

작성된 임시 테이블을 인스턴스화한 응용프로그램이 데이터베이스에서 연결을 끊을 때 작성된 임시 테이블의 응용프로그램 인스턴스가 삭제됩니다.

기존 테이블과 비슷한 테이블 작성

ATTACH PARTITION절을 사용하여 ALTER TABLE문을 발행할 때 목표 테이블의 특성이 소스 테이블의 특성과 충분히 일치하지 않을 경우, 소스 테이블을 새로 작성해야 할 수 있습니다. 새 소스 테이블을 작성하기 전에 기존 소스 테이블과 목표 테이블 간의 불일치를 정정할 수 있습니다.

테이블을 작성하려면, 명령문의 권한 부여 ID가 보유한 특권은 최소한 다음 권한 및 특권 중 하나를 포함해야 합니다.

- 다음 권한 중 하나를 비롯하여 테이블 스페이스에 대한 USE 특권 및 데이터베이스에 대한 CREATETAB 권한
  - 테이블의 내재적 또는 명시적 스키마 이름이 존재하지 않을 경우, 데이터베이스에 대한 IMPLICIT_SCHEMA 권한
  - 테이블의 스키마 이름이 기존의 스키마를 참조할 경우, 스키마에 대한 CREATEIN 특권
- DBADM 권한

불일치를 정정하는 데 실패한 경우 SQL20408N 또는 SQL20307N 오류가 리턴됩니다.

새 소스 테이블을 작성하려면 다음을 수행하십시오.

1. CREATE TABLE문을 사용하여 목표 테이블과 동일한 테이블을 작성하려면 db2look 명령을 사용하십시오.

   ```
   db2look -d <source database name> -t <target database name> -e -p
   ```

2. db2look 출력에서 파티션 절을 제거하고 작성된 테이블의 이름을 새 이름으로 변경하십시오(이 예에서는 sourceC라고 함).
3. 그런 다음, LOAD FROM CURSOR 명령을 사용하여 모든 데이터를 원래 소스 테이블에서 새로 작성된 소스 테이블(sourceC)로 로드하십시오.

   ```
   DECLARE mycurs CURSOR FOR SELECT * FROM source
   LOAD FROM mycurs OF CURSOR REPLACE INTO sourceC
   ```

   원본 테이블 데이터가 sourceC 테이블의 정의와 호환되지 않아 실패할 경우, sourceC에서 전송할 때 원본 테이블의 데이터를 변환해야 합니다.
4. 데이터를 sourceC로 복사한 후 ALTER TABLE target ...ATTACH sourceC문을 제출하십시오.

스테이징 데이터용 테이블 작성

*스테이징 테이블*은 지연된 구체화된 쿼리 테이블에 대한 점진적 유지보수 지원을 가능하게 합니다. 스테이징 테이블은 기초가 되는 테이블의 컨텐츠와 동기화하기 위해 구체화된 쿼리 테이블에 적용되어야 할 변경사항을 수집합니다. 스테이징 테이블을 사용하

면, 구체화된 쿼리 테이블의 즉시 새로 고침이 요청될 때 즉시 유지보수 컨텐츠로 인해 초래되는 높은 잠금 경합이 제거됩니다. 또한 REFRESH TABLE이 수행될 때마다 구체화된 쿼리 테이블을 전체적으로 재생성하지 않아도 됩니다.

구체화된 쿼리 테이블은 복잡한 쿼리, 특히 다음 중 일부 조작이 필요한 쿼리의 응답 시간을 개선하는 강력한 방법입니다.

- 하나 이상의 차원에 대한 집계 데이터
- 하나의 테이블 그룹에 있는 데이터 조인 및 집계
- 일반적으로 액세스되는 데이터 서브세트의 데이터
- 파티션된 데이터베이스 환경의 테이블 또는 테이블 파트에서 다시 파티션된 데이터

다음은 스테이징 테이블과 관련된 몇 가지 주요 제한사항입니다.

1. 스테이징 테이블을 작성하는데 사용되는 쿼리는 점진적으로 유지보수가 가능해야 합니다. 즉, 즉시 새로 고침 옵션이 있는 구체화된 쿼리 테이블과 동일한 규칙을 따라야 합니다.
2. 지연된 새로 고침에만 스테이징 테이블이 지원될 수 있습니다. 쿼리 또한 스테이징 테이블과 연관된 구체화된 쿼리 테이블을 정의합니다. 구체화된 쿼리 테이블은 REFRESH DEFERRED로 정의해야 합니다.
3. 스테이징 테이블을 사용하여 새로 고침을 수행할 때, 현재 시점까지의 새로 고침만이 지원됩니다.
4. 파티션된 계층 구조 테이블 및 파티션된 유형이 지정된 테이블은 지원되지 않습니다. (파티션된 테이블은 CREATE TABLE문의 PARTITION BY절에 제공된 스펙을 기본으로 데이터를 다중 스토리지 오브젝트로 파티션한 테이블입니다.)

몇 가지 다른 조작이 발생하지 않으면, 불일치하거나, 불완전하거나 또는 보류 상태인 스테이징 테이블을 연관된 구체화된 쿼리 테이블을 점진적으로 새로 고치는데 사용할 수 없습니다. 이러한 조작은 스테이징 테이블의 컨텐츠를 그와 연관된 구체화된 쿼리 테이블 및 기초가 되는 테이블과 일치하게 하여 스테이징 테이블을 보류 상태에서 벗어나게 합니다. 구체화된 쿼리 테이블이 새로 고쳐지면, 해당 스테이징 테이블의 컨텐츠가 지워지고 스테이징 테이블이 정상 상태로 설정됩니다. SET INTEGRITY문을 적절한 옵션과 함께 사용하여 스테이징 테이블을 의도적으로 프룬(prune)할 수도 있습니다. 프룬(prune)은 스테이징 테이블을 불일치하는 상태로 변경합니다. 예를 들어, 다음 명령문은 STAGTAB1이라는 스테이징 테이블의 프룬(prune)을 강제 실행합니다.

```
SET INTEGRITY FOR STAGTAB1 PRUNE;
```

스테이징 테이블은 작성시 보류 상태에 놓이며, 해당 테이블이 기초가 되는 테이블 및 연관된 구체화된 쿼리 테이블의 컨텐츠와 일치하지 않고 불완전함을 표시하는 표시기를 가집니다. 스테이징 테이블이 기초가 되는 테이블로부터 변경사항을 수집하려면 보류 및

불일치 상태에서 벗어나야 합니다. 보류 상태에 있는 동안에는 스테이징 테이블의 기본 테이블을 수정하려는 모든 시도가 실패하며 연관된 구체화된 쿼리 테이블을 새로 고치려는 시도도 실패합니다.

스테이징 테이블을 보류 상태에서 벗어나게 하는 몇 가지 방법이 있습니다.

- SET INTEGRITY FOR <staging table name> STAGING IMMEDIATE UNCHECKED
- SET INTEGRITY FOR <staging table name> IMMEDIATE CHECKED

DB2 기본 테이블과 임시 테이블의 차이

DB2 기본 테이블과 두 유형의 임시 테이블은 여러 가지 차이점을 갖습니다.

다음 표는 기본 테이블, 작성된 임시 테이블 및 선언된 임시 테이블 사이의 중요한 차이를 요약합니다.

표 18. DB2 기본 테이블과 DB2 임시 테이블 간의 중요한 차이

차이 영역	차이
작성, 지속성 및 테이블 설명을 공유하는 기능	**기본 테이블:** CREATE TABLE문이 카탈로그 뷰 SYSCAT.TABLES에 테이블의 설명을 입력합니다. 테이블 설명은 지속적이며 여러 연결 사이에 공유할 수 있습니다. CREATE TABLE문에 있는 테이블의 이름을 규정할 수 있습니다. 테이블 이름이 규정되지 않는 경우 SQL문에 적용되는 표준 자격 규칙을 사용하여 내재적으로 규정됩니다.
	작성된 임시 테이블: CREATE GLOBAL TEMPORARY TABLE문이 카탈로그 뷰 SYSCAT.TABLES에 테이블의 설명을 입력합니다. 테이블 설명은 지속적이며 여러 연결 사이에 공유할 수 있습니다. CREATE GLOBAL TEMPORARY TABLE문에 있는 테이블의 이름을 규정할 수 있습니다. 테이블 이름이 규정되지 않는 경우 SQL문에 적용되는 표준 자격 규칙을 사용하여 내재적으로 규정됩니다.
	선언된 임시 테이블: DECLARE GLOBAL TEMPORARY TABLE문이 카탈로그에 테이블의 설명을 입력하지 않습니다. 테이블 설명은 DECLARE GLOBAL TEMPORARY TABLE문을 발행한 연결의 수명을 지나 지속되지 않으며 설명은 해당 연결에만 알려집니다. 따라서 각 연결은 동일한 선언된 임시 테이블의 가능한 고유한 설명을 가질 수 있습니다. DECLARE GLOBAL TEMPORARY TABLE문의 테이블 이름을 규정할 수 있습니다. 테이블 이름이 규정되면 SESSION을 스키마 규정자로 사용해야 합니다. 테이블 이름이 규정되지 않으면 SESSION이 내재적으로 규정자로 사용됩니다.
테이블 인스턴스화 및 데이터 공유 기능	**기본 테이블:** CREATE TABLE문이 테이블의 하나의 빈 인스턴스를 작성하고, 모든 연결이 테이블의 해당 인스턴스를 사용합니다. 테이블과 데이터가 지속됩니다.
	작성된 임시 테이블: CREATE GLOBAL TEMPORARY TABLE문이 테이블의 인스턴스를 작성하지 않습니다. 연결을 사용하는 임의의 프로그램이 실행하는 열기, 선택, 삽입, 갱신 또는 삭제 조작에서 테이블에 대한 첫 번째 내재 또는 명시적 참조가 주어진 테이블의 빈 인스턴스를 작성합니다. 테이블을 참조하는 각 연결은 테이블의 고유한 인스턴스를 갖고, 인스턴스는 연결 수명 이상으로 지속되지 않습니다.
	선언된 임시 테이블: DECLARE GLOBAL TEMPORARY TABLE문이 연결에 대해 테이블의 빈 인스턴스를 작성합니다. 테이블을 선언하는 각 연결이 테이블의 고유한 인스턴스를 갖고, 인스턴스는 연결 수명 이상으로 지속되지 않습니다.

표 18. DB2 기본 테이블과 DB2 임시 테이블 간의 중요한 차이 (계속)

차이 영역	차이
연결 중 테이블에 대한 참조	**기본 테이블:** 다중 연결에서 테이블 이름에 대한 참조는 동일한 단일 지속적 테이블 설명 및 현재 서버에 있는 동일한 인스턴스를 참조합니다. 참조되는 테이블 이름이 규정되지 않는 경우 SQL문에 적용되는 표준 자격 규칙을 사용하여 내재적으로 규정됩니다.
	선언된 임시 테이블: 다중 연결에서 테이블 이름에 대한 참조는 동일한 단일 지속적 테이블 설명을 참조하지만 현재 서버에 있는 각 연결에 대한 테이블의 고유 인스턴스를 참조하지 않습니다. 참조되는 테이블 이름이 규정되지 않는 경우 SQL문에 적용되는 표준 자격 규칙을 사용하여 내재적으로 규정됩니다.
	선언된 임시 테이블: 다중 연결에서 테이블 이름에 대한 참조는 현재 서버에 있는 각 연결에 대한 테이블의 고유한 설명 및 인스턴스를 참조합니다. SQL문(DECLARE GLOBAL TEMPORARY TABLE문 제외)에서 테이블 이름에 대한 참조는 SESSION을 스키마 규정자로 포함해야 합니다. 테이블 이름이 SESSION으로 규정되지 않으면 참조는 기본 테이블에 대한 것으로 가정됩니다.
테이블 특권 및 권한 부여	**기본 테이블:** 소유자는 내재적으로 테이블에 대한 모든 테이블 특권 및 테이블 삭제 권한을 갖습니다. 소유자의 테이블 특권은 개별적으로 또는 ALL절을 사용하여 권한 부여 및 권한 취소될 수 있습니다. 또 다른 권한 부여 ID는 테이블에 대한 적절한 특권이 권한 부여된 경우에만 테이블에 액세스할 수 있습니다.
	작성된 임시 테이블: 소유자는 내재적으로 테이블에 대한 모든 테이블 특권 및 테이블 삭제 권한을 갖습니다. 소유자의 테이블 특권은 개별적으로 또는 ALL절을 사용하여 권한 부여 및 권한 취소될 수 있습니다. 또 다른 권한 부여 ID는 테이블에 대한 적절한 특권이 권한 부여된 경우에만 테이블에 액세스할 수 있습니다.
	선언된 임시 테이블: PUBLIC이 내재적으로 GRANT 권한을 제외한 테이블에 대한 모든 테이블 특권을 갖고 테이블 삭제 권한도 갖습니다. 이러한 테이블 특권은 권한 부여 또는 권한 취소할 수 없습니다. 권한 부여 ID는 테이블에 대한 특권의 권한 부여 없이 테이블에 액세스할 수 있습니다.
인덱스 및 기타 SQL문 지원	**기본 테이블:** 인덱스 및 데이터를 수정할 수 있는 SQL문(INSERT, UPDATE, DELETE 등)이 지원됩니다. 인덱스는 서로 다른 테이블 스페이스에 존재할 수 있습니다.
	작성된 임시 테이블: 인덱스 및 데이터를 수정할 수 있는 SQL문(INSERT, UPDATE, DELETE 등)이 지원됩니다. 인덱스는 테이블과 동일한 테이블 스페이스에만 존재할 수 있습니다.
	선언된 임시 테이블: 인덱스 및 데이터를 수정할 수 있는 SQL문(INSERT, UPDATE, DELETE 등)이 지원됩니다. 인덱스는 테이블과 동일한 테이블 스페이스에만 존재할 수 있습니다.
잠금, 로깅 및 복구	**기본 테이블:** 잠금, 로깅 및 복구가 적용됩니다.
	작성된 임시 테이블: 잠금 및 복구는 적용되지 않지만, 로깅은 LOGGED가 명시적으로 지정될 때 적용됩니다. 복구 실행 취소(세이브포인트 또는 가장 최근 커미트 지점으로 변경 롤백)는 LOGGED가 *명시적*으로 지정될 때만 지원됩니다.
	선언된 임시 테이블: 잠금 및 복구는 적용되지 않지만, 로깅은 LOGGED가 명시적으로 또는 내재적으로 지정될 때만 적용됩니다. 복구 실행 취소(세이브포인트 또는 가장 최근 커미트 지점으로 변경 롤백)는 LOGGED가 명시적 또는 내재적으로 지정될 때 지원됩니다.

테이블 수정

이 절에서는 테이블을 수정하는 방법에 대한 주제를 제공합니다.

테이블 변경

테이블을 변경할 때 ALTER COLUMN SET DATA TYPE 옵션 및 단일 트랜잭션에서 수행할 수 있는 무제한 REORG 권장 조작 같이 알아야 하는 몇 가지 유용한 옵션이 있습니다.

테이블 변경 SET DATA TYPE 지원

ALTER TABLE문의 ALTER COLUMN SET DATA TYPE 옵션은 호환 가능한 모든 유형을 지원합니다.

컬럼 데이터 유형 변경은 데이터 유실을 유발할 수 있습니다. 이 유실의 일부는 캐스팅 규칙과 일관성이 있습니다. 예를 들어 공백은 오류를 리턴하지 않고 문자열에서 절단될 수 있으며 DECIMAL을 INTEGER로 변환하면 절단이 발생합니다. 오버플로우 오류, 절단 오류 또는 캐스팅이 리턴하는 다른 모든 유형의 오류 같은 예기치 않은 오류를 막기 위해, 기존 컬럼 데이터가 스캔되며 충돌하는 행에 대한 메시지가 통지 로그에 기록됩니다. 컬럼 디폴트값도 점검하여 해당 값이 새 데이터 유형을 준수하는지 확인합니다.

데이터 스캔이 어떤 오류도 보고하지 않는 경우 컬럼 유형은 새 데이터 유형으로 설정되고 기존 컬럼 데이터가 새 데이터 유형으로 캐스트됩니다. 오류가 보고되면 ALTER TABLE문이 실패합니다.

VARCHAR, VARGRAPHIC 또는 LOB 컬럼을 데이터 유형 우선순위 목록(*데이터 유형의 승격* 주제 참조)에서 더 유사한 데이터 유형으로 변경하는 것은 지원되지 않습니다.

예

SALES 테이블의 SALES 컬럼의 데이터 유형을 INTEGER에서 SMALLINT로 변경하십시오.

```
alter table sales alter column sales set data type smallint
DB20000I  The SQL command completed successfully.
```

SALES 테이블의 REGION 컬럼의 데이터 유형을 VARCHAR(15)에서 VARCHAR(14)로 변경하십시오.

```
alter table sales alter column region set data type varchar(14)
...
SQL0190N  ALTER TABLE "ADMINISTRATOR.SALES" specified attributes for column
"REGION" that are not compatible with the existing column.  SQLSTATE=42837
```

기본 테이블에서 컬럼 유형을 변경하십시오. 기본 테이블에 직접 또는 간접적으로 종속되는 뷰와 기능이 있습니다.

```
create table t1 (c1 int, c2 int);

create view v1 as select c1, c2 from t1;
```

```
create view v2 as select c1, c2 from v1;

create function foo1 ()
 language sql
 returns int
 return select c2 from t1;

create view v3 as select c2 from v2
 where c2 = foo1();

create function foo2 ()
 language sql
 returns int
 return select c2 from v3;

alter table t1
 alter column c1
  set data type smallint;

select * from v2;
```

ALTER TABLE문은 INTEGER에서 SMALLINT로 컬럼 유형을 하향 캐스트하며 v1, v2, v3 및 foo2를 무효화합니다. 의미상 유효성 다시 확인이 지연된 경우 `select * from v2`는 v1 및 v2의 유효성을 다시 확인하고, v1 및 v2의 c1 컬럼이 SMALLINT로 변경됩니다. 그러나 v3 및 foo2는 다시 유효성 확인되지 않는데, 무효화된 후 참조되지 않으며 종속성 계층 구조 체인에서 v2 위에 있기 때문입니다. 의미상 즉시 유효성 다시 확인하는 경우 ALTER TABLE문은 모든 종속 오브젝트를 성공적으로 다시 유효성 확인합니다.

단일 트랜잭션에서 무제한 REORG 권장 조작 수행

디스크에 있는 데이터의 형식을 변경하는 모든 ALTER TABLE 조작은 새 버전의 데이터 디스크립터가 필요하며 **REORG 권장 조작**이라고 합니다.

이런 조작에는 컬럼 삭제, 컬럼 유형 변경 또는 컬럼의 널(null) 가능성 등록 정보 변경이 포함됩니다. 이전 버전에서는 재구성이 필요하기 전에 테이블에 대해 이 유형의 ALTER TABLE문을 세 개까지 실행할 수 있었습니다. 이 한계는 부분적으로 제거되었습니다. 작업 단위(UOW)당 무제한 수의 REORG 권장 조작이 이제 허용됩니다. REORG TABLE 명령이 동일한 테이블에 대한 REORG 권장 조작을 갖는 세 작업 단위(UOW)의 총계가 발생한 후 발행되어야 하며, 테이블 재구성이 해당 테이블에 대한 모든 추가 REORG 권장 조작의 전제조건입니다.

행 버전은 첫 번째 REORG 권장 조작이 실행될 때 작업 단위당 한 번만 변경됩니다. 동일한 UOW의 후속 REORG 권장 조작은 행의 새 버전을 작성하지 않습니다. 이전 버전에서와 같이, 디스크의 데이터는 후속 ALTER TABLE문에서 최소 하나의 컬럼 삭제 조작이 있는 경우에만 갱신됩니다.

구체화된 쿼리 테이블 등록 정보 변경

구체화된 쿼리 테이블을 테이블로 변경하거나 일반 테이블을 구체화된 쿼리 테이블로 변경할 수 있으나 몇 가지 제한사항이 있습니다. 다른 테이블 유형은 변경할 수 없습니다. 일반 및 구체화된 쿼리 테이블만 변경할 수 있습니다. 예를 들어, 복제된 구체화된 쿼리 테이블을 일반 테이블로 변경하거나 일반 테이블을 복제된 구체화된 쿼리 테이블로 변경할 수는 없습니다.

일반 테이블이 구체화된 쿼리 테이블로 변경되면 이 테이블은 무결성 설정 보류 상태에 놓입니다. 이와 같은 방법으로 변경할 때, 구체화된 쿼리 테이블 정의의 fullselect는 원래 테이블 정의와 일치해야 합니다. 즉 다음과 같아야 합니다.

- 컬럼 수가 동일해야 합니다.
- 컬럼 이름 및 위치가 일치해야 합니다.
- 데이터 유형이 동일해야 합니다.

구체화된 쿼리 테이블이 원래의 테이블에 정의된 경우, 원래의 테이블 자체는 구체화된 쿼리 테이블로 변경할 수 없습니다. 원래 테이블에 트리거, 점검 제한조건, 참조 제한조건 또는 정의된 고유 인덱스가 있으면, 이 테이블을 구체화된 쿼리 테이블로 변경할 수 없습니다. 테이블 등록 정보를 변경하여 구체화된 쿼리 테이블을 정의할 경우, 동일한 ALTER TABLE문에서 다른 방식으로 테이블을 변경할 수 없습니다.

일반 테이블을 구체화된 쿼리 테이블로 변경할 때, 구체화된 쿼리 테이블 정의의 fullselect는 뷰, 별명 또는 구체화된 쿼리 테이블을 통해 직접 또는 간접으로 원래 테이블을 참조할 수 없습니다.

구체화된 쿼리 테이블을 일반 테이블로 변경하려면 다음을 사용하십시오.

```
ALTER TABLE sumtable
SET SUMMARY AS DEFINITION ONLY
```

일반 테이블을 구체화된 쿼리 테이블로 변경하려면 다음을 사용하십시오.

```
 ALTER TABLE regtable
SET SUMMARY AS <fullselect>
```

일반 테이블을 구체화된 쿼리 테이블로 변경할 때 fullselect의 제한사항은 CREATE SUMMARY TABLE문을 사용하여 요약 테이블을 작성할 때의 제한사항과 매우 유사합니다.

구체화된 쿼리 테이블의 데이터 새로 고침

REFRESH TABLE문을 사용하여 하나 이상의 구체화된 쿼리 테이블에 있는 데이터를 새로 고칠 수 있습니다. 응용프로그램에 해당 명령문을 삽입하거나 동적으로 발행할 수 있습니다. 이 명령문을 사용하려면 DATAACCESS 권한이나 새로 고칠 테이블에 대한 CONTROL 특권이 있어야 합니다.

다음 예는 구체화된 쿼리 테이블의 데이터를 새로 고치는 방법을 보여줍니다.

```
REFRESH TABLE SUMTAB1
```

컬럼 등록 정보 변경

널(null) 가능성, LOB 옵션, 범위, 제한조건과 압축 속성, 데이터 유형 등과 같은 컬럼 등록 정보를 변경하려면 ALTER TABLE문을 사용하십시오. 전체 세부사항에 대해서는 ALTER TABLE문을 참조하십시오.

테이블을 변경하려면 변경 대상 테이블에 다음 중 하나 이상의 특권이 있어야 합니다.

- ALTER 특권
- CONTROL 특권
- DBADM 권한
- 테이블의 스키마에 대한 ALTERIN 특권

기존 컬럼의 정의를 변경하거나, 테이블 컬럼을 변경할 때 SQL을 편집 및 테스트하거나, 테이블 컬럼을 변경할 때 관련된 오브젝트의 유효성을 확인하려면 DBADM 권한을 가지고 있어야 합니다.

예를 들어 명령행에서 다음을 입력하십시오.

```
ALTER TABLE EMPLOYEE
        ALTER COLUMN WORKDEPT
      SET DEFAULT '123'
```

컬럼 추가 및 삭제

기존 테이블에 컬럼을 추가하거나 기존 테이블에서 컬럼을 삭제하려면 각각 ADD COLUMN 또는 DROP COLUMN절을 갖는 ALTER TABLE문을 사용하십시오. 테이블은 유형이 지정된 테이블이 아니어야 합니다.

테이블에 있는 모든 기존 행의 경우 새 컬럼의 값은 디폴트값으로 설정됩니다. 새 컬럼은 테이블의 마지막 컬럼이 되는데, 이는 초기에 *n*개의 컬럼이 있으면 추가되는 컬럼은 컬럼 *n*+1임을 의미합니다. 새 컬럼을 추가해도 모든 컬럼의 총 바이트 수가 최대 행 크기 한계를 초과해서는 안됩니다.

컬럼을 추가하려면 다음 명령문을 발행하십시오.

```
ALTER TABLE SALES
  ADD COLUMN SOLD_QTY
  SMALLINT NOT NULL DEFAULT 0
```

컬럼을 삭제하거나 삭제(drop)하려면 다음 명령문을 발행하십시오.

```
ALTER TABLE SALES
  DROP COLUMN SOLD_QTY
```

DEFAULT절 컬럼 정의 수정

DEFAULT절은 값이 INSERT에서 제공되지 않거나 INSERT 또는 UPDATE에서 DEFAULT로 지정되는 이벤트 시에 컬럼에 대한 디폴트값을 제공합니다. 특정 디폴트 값이 DEFAULT 키워드 뒤에 지정되지 않는 경우 디폴트값은 데이터 유형에 따라 달라집니다. 컬럼이 XML 또는 구조화된 유형으로 정의되는 경우 DEFAULT절을 지정할 수 없습니다.

컬럼 정의에서 DEFAULT를 생략하면 266 페이지의 『디폴트 컬럼 및 데이터 유형 정의』에서 설명하는 것처럼 널(NULL) 값의 사용이 컬럼에 대한 디폴트가 됩니다.

DEFAULT 키워드와 함께 지정할 수 있는 값의 특정 유형에 대해서는 ALTER TABLE 문을 참조하십시오.

생성된 컬럼 또는 컬럼의 ID 등록 정보 수정

ALTER TABLE문에서 ALTER COLUMN절을 사용하여 테이블의 생성된 컬럼 또는 컬럼의 ID 등록 정보를 추가하고 삭제할 수 있습니다.

다음 조치 중 하나를 수행할 수 있습니다.

- 기존 비생성 컬럼에 대해 작업할 때, 생성된 표현식 속성을 추가할 수 있습니다. 수정된 컬럼이 생성된 컬럼이 됩니다.
- 기존 생성된 컬럼에 대해 작업할 때, 생성된 표현식 속성을 삭제할 수 있습니다. 수정된 컬럼이 보통 비생성 컬럼이 됩니다.
- 기존 비식별 컬럼에 대해 작업할 때, 식별 속성을 추가할 수 있습니다. 수정된 컬럼이 식별 컬럼이 됩니다.
- 기존 식별 컬럼에 대해 작업할 때, 식별 속성을 삭제할 수 있습니다. 수정된 컬럼이 보통 비생성 비식별 컬럼이 됩니다.
- 기존 ID 컬럼에 대해 작업할 때, 컬럼을 GENERATED ALWAYS에서 GENERATED BY DEFAULT로 변경할 수 있습니다. 반대의 경우도 또한 올바릅니다. 즉, 컬럼을 GENERATED BY DEFAULT에서 GENERATED ALWAYS로 변경할 수 있습니다. 이는 ID 컬럼에 대해 작업할 때만 가능합니다.
- 사용자 정의 디폴트 컬럼에서 디폴트 속성을 삭제할 수 있습니다. 이를 수행할 때, 새 디폴트값은 널(null)입니다.
- 디폴트, 식별 또는 생성 속성을 삭제한 후 동일한 ALTER COLUMN문에서 새 디폴트, 식별 또는 생성 속성을 설정할 수 있습니다.
- CREATE TABLE 및 ALTER TABLE문 둘 다에 대해 『ALWAYS』는 GENERATED절의 선택적 단어입니다. 이는 ALTER TABLE에서 사용될 때 GENERATED ALWAYS가 GENERATED와 같음을 의미합니다.

컬럼 정의 수정

컬럼을 삭제하거나 해당 유형 및 속성을 변경하려면 ALTER TABLE문을 사용하십시오. 예를 들어, 기존 VARCHAR 또는 VARGRAPHIC 컬럼의 길이를 늘릴 수 있습니다. 문자 수는 사용되는 페이지 크기에 맞는 값까지 증가될 수 있습니다.

컬럼과 연관된 디폴트값을 수정하려면, 새 디폴트값을 정의한 후에 디폴트 사용이 표시되는 모든 후속 SQL 조작에서 컬럼에 대해 새 값이 사용됩니다. 새 값은 지정에 대한 규칙을 따라야 하며 CREATE TABLE문에서 설명한 것과 동일한 제한사항을 갖습니다.

주: 컬럼 생성시 해당 명령문으로 디폴트값을 변경할 수 없습니다.

SQL을 사용하여 해당 테이블 속성을 변경할 때 테이블을 삭제한 후 재작성할 필요가 없어졌습니다. 이 프로세스는 오브젝트 종속사항이 있을 때 복잡하고 시간이 많이 드는 프로세스였습니다.

명령행을 사용하여 기존 테이블의 컬럼 길이 및 유형을 수정하려면 다음을 입력하십시오.

```
ALTER TABLE <table_name>
      ALTER COLUMN <column_name>
<modification_type>
```

예를 들어, 컬럼을 최대 4000자까지 증가시키려면, 다음과 유사한 것을 사용하십시오.

```
ALTER TABLE t1
      ALTER COLUMN colnam1
SET DATA TYPE VARCHAR(4000)
```

다른 예에서 컬럼이 새 VARGRAPHIC 값을 가질 수 있도록 다음과 유사한 명령문을 사용하십시오.

```
ALTER TABLE t1
      ALTER COLUMN colnam2
SET DATA TYPE VARGRAPHIC(2000)
```

입력된 테이블의 컬럼은 변경할 수 없습니다. 그러나 아직 범위가 정의되지 않은 기존 참조 유형 컬럼에 범위를 추가할 수는 있습니다. 예를 들어, 다음과 같습니다.

```
ALTER TABLE t1
      ALTER COLUMN colnamt1
ADD SCOPE typtab1
```

LOB가 인라인 포함될 수 있도록 컬럼을 수정하려면 다음을 입력하십시오.

```
ALTER TABLE <table_name>
      ALTER COLUMN <column_name>
SET INLINE LENGTH <new_LOB_length>
```

예를 들어 1000바이트 이하의 LOB가 기본 테이블에 포함되기 원한다고 결정한 경우 다음과 비슷한 명령문을 사용합니다.

```
ALTER TABLE t1
      ALTER COLUMN colnam1
SET INLINE LENGTH 1004
```

이 경우에 길이는 1000이 아니라 1004로 설정됩니다. 이것은 인라인 LOB가 LOB 자체의 크기 이상으로 추가 4바이트의 스토리지가 필요하기 때문입니다.

명령행을 사용하여 기존 테이블의 컬럼 디폴트값을 수정하려면 다음을 입력하십시오.

```
ALTER TABLE <table_name>
      ALTER COLUMN <column_name>
SET DEFAULT 'new_default_value'
```

예를 들어, 컬럼의 디폴트값을 변경하려면, 다음과 유사하게 사용하십시오.

```
ALTER TABLE t1
      ALTER COLUMN colnam1
SET DEFAULT '123'
```

테이블 및 컬럼 이름 바꾸기

RENAME문을 사용하여 기존 테이블의 이름을 바꿀 수 있습니다. 컬럼 이름을 바꾸려면 ALTER TABLE문을 사용하십시오.

테이블 이름을 바꿀 때 소스 테이블이 어떠한 기존 정의(뷰 또는 구체화된 쿼리 테이블), 트리거, SQL 함수 또는 제한조건에서 참조되지 않아야 합니다. 또한 생성된 컬럼(ID 컬럼이 아닌)을 갖거나 상위 또는 종속 테이블이 아니어야 합니다. 카탈로그 항목이 새 테이블 이름을 반영하도록 갱신됩니다. 자세한 정보와 예는 RENAME문을 참조하십시오.

RENAME COLUMN절은 ALTER TABLE문의 옵션입니다. 저장된 데이터를 유실하거나 테이블과 연관된 모든 특권 또는 레이블 기반 액세스 제어(LBAC) 규정에 영향을 주지 않고 기본 테이블에 있는 기존 컬럼의 이름을 바꿀 수 있습니다.

기본 테이블 컬럼의 이름 바꾸기만 지원됩니다. 뷰, 구체화된 쿼리 테이블(MQT), 선언 및 작성된 임시 테이블 및 기타 테이블형 오브젝트의 컬럼 이름 바꾸기는 지원되지 않습니다.

컬럼 이름 바꾸기 조작에 대한 무효화 및 유효성 다시 확인 시맨틱은 컬럼 삭제 조작의 경우와 비슷합니다. 즉, 모든 종속 오브젝트가 무효화됩니다. **auto_reval** 데이터베이스 구성 매개변수가 DISABLED로 설정되어 있는 경우에도 컬럼 이름 바꾸기 조작 뒤에 이어지는 모든 종속 오브젝트의 유효성 다시 확인은 무효화된 후 바로 완료됩니다.

다음 예는 ALTER TABLE문을 사용한 컬럼 이름 바꾸기를 나타냅니다.

```
ALTER TABLE org RENAME COLUMN deptnumb TO deptnum
```

기존 컬럼의 정의를 변경하려면 ″컬럼 등록 정보 변경″ 주제나 ALTER TABLE문을 참조하십시오.

작동 불능 요약 테이블 복구

기본 테이블의 SELECT 특권을 권한 취소하면 요약 테이블이 작동 불능 상태가 될 수 있습니다.

다음 단계는 작동 불능 요약 테이블을 복구하는데 도움이 됩니다.

- 요약 테이블을 작성하기 위해 초기에 사용된 명령문을 판별하십시오. SYSCAT.VIEW 카탈로그 뷰의 TEXT 컬럼에서 이러한 정보를 얻을 수 있습니다.
- 동일한 요약 테이블 이름 및 동일한 정의를 통해 CREATE SUMMARY TABLE 문을 사용하여 요약 테이블을 재작성하십시오.
- GRANT문을 사용하여, 요약 테이블에 이전에 권한 부여된 모든 특권을 다시 권한 부여하십시오. (작동 불능 요약 테이블에 권한 부여된 특권이 모두 권한 취소됨을 유의하십시오.)

작동 불능 요약 테이블을 복구하지 않으려는 경우, DROP TABLE문을 사용하여 작동 불능 요약 테이블을 명시적으로 제거하거나 정의는 다르지만 동일한 이름을 사용하여 새 요약 테이블을 작성할 수 있습니다.

작동 불능 요약 테이블에는 SYSCAT.TABLES 및 SYSCAT.VIEWS 카탈로그 뷰에 있는 항목만 있습니다. SYSCAT.TABDEP, SYSCAT.TABAUTH, SYSCAT.COLUMNS 및 SYSCAT.COLAUTH 카탈로그 뷰의 모든 항목은 제거됩니다.

테이블 정의 보기

SYSCAT.TABLES 및 SYSCAT.COLUMNS 카탈로그 뷰를 사용하여 테이블 정의를 볼 수 있습니다. SYSCAT.COLUMNS의 경우, 각 행은 테이블, 뷰 또는 별칭에 대해 정의된 컬럼을 나타냅니다. 컬럼의 데이터를 보려면 SELECT문을 사용하십시오.

또한 다음 뷰 및 테이블 함수를 사용하여 테이블 정의를 볼 수도 있습니다.

- ADMINTEMPCOLUMNS 관리 뷰
- ADMINTEMPTABLES 관리 뷰
- ADMIN_GET_TEMP_COLUMNS 테이블 함수 - 임시 테이블에 대한 컬럼 정보 검색
- ADMIN_GET_TEMP_TABLES 테이블 함수 - 임시 테이블에 대한 정보 검색

테이블 삭제

테이블은 DROP TABLE문으로 삭제될 수 있습니다. 테이블이 삭제되면, 삭제된 테이블에 대한 정보가 들어있는 SYSCAT.TABLES 카탈로그의 행과, 테이블에 의존하는 다른 모든 오브젝트가 영향을 받습니다.

예를 들어,

- 모든 컬럼 이름이 삭제됩니다.
- 테이블의 컬럼에 대해 작성된 인덱스가 삭제됩니다.
- 테이블에 근거한 모든 뷰가 작동 불능 상태로 표시됩니다.
- 삭제된 테이블과 종속 뷰에 대한 모든 특권도 내재적으로 권한 취소됩니다.
- 상위 테이블 또는 종속 테이블에 있는 모든 참조 제한조건이 삭제됩니다.
- 삭제된 테이블에 종속된 모든 패키지 및 캐시된 동적 SQL 및 XQuery문은 유효하지 않은 것으로 표시되며 종속 오브젝트가 재작성될 때까지 이 상태로 남아 있습니다. 계층 구조에서 삭제하려는 서브테이블 위에 있는 모든 슈퍼 테이블에 종속된 패키지도 여기에 포함됩니다.
- 삭제된 테이블이 참조 범위로서 정의된 모든 참조 컬럼은 『범위 없는 상태』가 됩니다.
- 별명이 정의될 수 없기 때문에 테이블에 대한 별명 정의는 효력이 없습니다.
- 삭제된 테이블에 종속적인 모든 트리거는 작동 불능으로 표시됩니다.

명령행을 사용하여 테이블을 삭제하려면 다음을 입력하십시오.

```
DROP TABLE <table_name>
```

다음 명령문은 DEPARTMENT라고 하는 테이블을 삭제합니다.

```
DROP TABLE DEPARTMENT
```

각각의 테이블은 하위테이블이 있을 때는 삭제될 수 없습니다. 그러나 테이블 계층 구조에 있는 모든 테이블은 다음 예에서처럼 단일 DROP TABLE HIERARCHY문으로 삭제될 수 있습니다.

```
DROP TABLE HIERARCHY person
```

DROP TABLE HIERARCHY문은 삭제될 계층 구조의 루트 테이블을 이름 지정해야 합니다.

특정 테이블을 삭제하는 것과 테이블 계층 구조를 삭제하는 것에는 차이가 있습니다.

- DROP TABLE HIERARCHY는 개별 DROP 테이블 명령문으로 활성화되는 삭제된 트리거를 활성화시키지 않습니다. 예를 들어, 개별 서브테이블을 삭제하면, 상위 테이블에 대한 삭제 트리거가 활성화됩니다.

- DROP TABLE HIERARCHY는 제거된 테이블의 개별 행에 대해 로그 항목을 작성하지 않습니다. 대신 계층 구조 제거는 단일 이벤트로 로그됩니다.

구체화된 쿼리 또는 스테이징 테이블 삭제

구체화된 쿼리 또는 스테이징 테이블을 변경할 수는 없지만 삭제할 수는 있습니다. 테이블을 참조하는 모든 인덱스, 기본 키, 외부 키 및 점검 제한조건이 제거됩니다. 테이블을 참조하는 모든 뷰 및 트리거는 사용 불가능 상태가 됩니다. 제거되거나 작동 불능 상태로 표시된 오브젝트에 종속되는 모든 패키지는 유효하지 않음으로 표시됩니다.

명령행을 사용하여 구체화된 쿼리 테이블 또는 스테이징 테이블을 삭제하려면 다음을 입력하십시오.

```
DROP TABLE <table_name>
```

다음 명령문은 구체화된 쿼리 테이블 XT™를 삭제합니다.

```
DROP TABLE XT
```

구체화된 쿼리 테이블은 DROP TABLE문으로 명시적으로 삭제되거나 또는 기초가 되는 테이블이 삭제될 때 내재적으로 삭제될 수 있습니다.

스테이징 테이블은 DROP TABLE문으로 명시적으로 삭제되거나 또는 연관된 구체화된 쿼리 테이블이 삭제될 때 내재적으로 삭제될 수 있습니다.

시나리오 및 테이블 예

이 절에서는 시나리오 및 테이블 예를 제공합니다.

시나리오: 낙관적 잠금 및 시간별 발견

시간별 발견을 사용하거나 사용하지 않고, 내재적으로 숨겨진 컬럼을 사용하거나 사용하지 않고 응용프로그램에서 낙관적 잠금을 사용하고 구현하는 방법을 나타내는 세 가지 시나리오가 제공됩니다.

시나리오: 응용프로그램에서 낙관적 잠금 사용

이 시나리오는 여섯 가지 시나리오를 포함하여 응용프로그램에서 낙관적 잠금이 구현되는 방법을 설명합니다.

낙관적 잠금에 대해 설계되고 사용 가능한 응용프로그램에서 다음 이벤트 시퀀스를 고려하십시오.

```
SELECT QUANTITY, row change token FOR STOCK, RID_BIT(STOCK)
 INTO :h_quantity, :h_rct, :h_rid
 FROM STOCK WHERE PARTNUM = 3500
```

이 시나리오에서 응용프로그램 논리가 각 행을 읽습니다. 이 응용프로그램이 297 페이지의 『응용프로그램에서 낙관적 잠금 사용』에서 설명하는 대로 낙관적 잠금을 사용할

수 있으므로, 선택 목록에 :h_rid 호스트 변수에 저장된 RID_BIT() 값과 :h_rct 호스트 변수에 저장된 행 변경 토큰이 포함됩니다.

낙관적 잠금이 사용 가능한 경우 응용프로그램은 갱신 또는 삭제의 목표가 되는 모든 행이 잠금에 의해 보호되지 않는 경우에도 변경되지 않은 채로 있다고 낙관적으로 가정합니다. 데이터베이스 동시성을 개선하기 위해 응용프로그램은 다음 메소드 중 하나를 사용하여 행 잠금을 제거합니다.

- 작업 단위(UOW) 커미트. 이 경우 행 잠금이 제거됩니다.
- WITH RELEASE절을 사용한 커서 닫기. 이 경우 행 잠금이 제거됩니다.
- 더 낮은 분리 레벨 사용:
  - CURSOR STABILITY(CS) - 이 경우 커서가 다음 행 또는 결과 테이블의 끝에 페치된 후 행이 잠기지 않습니다.
  - UNCOMMITED READ(UR) - 이 경우에는 언커미트된 모든 데이터가 새(언커미트된) 행 변경 토큰 값을 갖습니다. 언커미트된 데이터가 롤백되면 이전에 커미트된 행 변경 토큰이 다른 값이 됩니다.

    주: 갱신이 정상적으로 롤백되지 않는다고 가정하면 UR 사용이 대부분의 동시성을 허용합니다.
- 데이터베이스에서 연결 해제와 이에 따른 응용프로그램에 대한 모든 DB2 서버 자원 해제(.NET 응용프로그램이 종종 이 모드를 사용함)

응용프로그램이 행을 처리하고 다음 중 하나를 낙관적으로 갱신하기 원한다고 결정합니다.

```
UPDATE STOCK SET QUANTITY = QUANTITY – 1
 WHERE row change token FOR STOCK = :h_rct AND
 RID_BIT(STOCK) = :h_rid
```

UPDATE문이 위에 표시된 SELECT문에서 식별되는 행을 갱신합니다.

검색된 UPDATE 술어는 테이블에 대한 직접 페치로 계획됩니다.

```
RID_BIT(STOCK) = :h_rid
```

직접 페치는 DB2 옵티마이저가 비용을 산정하기에 간단한 매우 효율적인 액세스 플랜입니다. RID_BIT() 술어가 행을 찾지 못하는 경우 행은 삭제되고 갱신은 행을 찾을 수 없음과 함께 실패합니다.

RID_BIT() 술어가 행을 찾는다고 가정하면, 술어 행 변경 토큰 FOR STOCK = :h_rct는 행 변경 토큰이 변경되지 않은 경우에 행을 찾습니다. 행 변경 토큰이 SELECT 이후 변경된 경우, 검색된 UPDATE는 행을 찾을 수 없음과 함께 실패합니다.

317 페이지의 표 19는 낙관적 잠금이 사용 가능할 때 발생할 수 있는 가능한 시나리오를 나열합니다.

표 19. 낙관적 잠금이 사용 가능할 때 발생할 수 있는 시나리오

시나리오 ID	조치	결과
시나리오 1	테이블에 정의된 행 변경 시간소인 컬럼이 있고 다른 응용프로그램이 행을 변경하지 않았습니다.	행 변경 토큰 술어가 :h_rid로 식별되는 행에 대해 성공하므로 갱신이 성공합니다.
시나리오 2	테이블에 정의된 ROW CHANGE TIMESTAMP가 있습니다. 다른 응용프로그램이 선택 이후 및 갱신(및 커미트) 이전에 행을 갱신하므로, 행 변경 시간소인 컬럼을 갱신합니다.	선택 시점에서 행에 있는 시간소인으로부터 생성된 토큰을 현재 행에 있는 시간소인의 토큰 값과 비교하여 행 변경 토큰 술어가 실패합니다. 따라서 UPDATE문이 행을 찾지 못합니다.
시나리오 3	테이블에 정의된 ROW CHANGE TIMESTAMP가 있습니다. 다른 응용프로그램이 행을 갱신하므로 행에 새 행 변경 토큰이 있습니다. 이 응용프로그램이 분리 UR에서 행을 선택하고 새로운 언커미트된 행 변경 토큰을 가져옵니다.	이 응용프로그램이 UPDATE를 실행하며, 이것은 다른 응용프로그램이 행 잠금을 해제할 때까지 잠금 대기합니다. 다른 응용프로그램이 새 토큰으로 변경을 커미트하여 UPDATE가 성공하는 경우 행 변경 토큰 술어가 성공합니다. 다른 응용프로그램이 이전 토큰으로 롤백하여 UPDATE가 행을 찾지 못하는 경우 행 변경 토큰 술어는 실패합니다.
시나리오 4	테이블에 정의된 행 변경 시간소인 컬럼이 없습니다. 선택 이후 및 갱신 이전에 동일한 페이지에서 다른 행이 갱신, 삭제 또는 삽입됩니다.	행 변경 토큰 술어는 토큰 비교에 실패합니다. 페이지의 모든 행에 대한 행 변경 토큰 값이 변경되었으므로 UPDATE문이 한 행이 실제로 변경되지 않은 경우에도 행을 찾지 못하기 때문입니다. 이 거짓 부정(false negative) 시나리오의 결과는 행 변경 시간소인 컬럼이 추가된 경우 UPDATE 실패입니다.
시나리오 5	테이블이 행 변경 시간소인 컬럼을 포함하도록 변경되었고, 선택에서 리턴된 행이 변경 시간 이후에 수정되지 않았습니다. 다른 응용프로그램이 현재 시간소인을 갖는 프로세스에서 해당 행에 행 변경 시간소인 컬럼을 추가하여 행을 갱신합니다.	행 변경 토큰 술어가 이전에 생성된 토큰을 행 변경 시간소인 컬럼에서 작성된 토큰 값과 비교에 실패하므로 UPDATE문이 행을 찾지 못합니다. 관심이 있는 행이 실제로는 변경되었으므로 이것은 거짓 부정(false negative) 시나리오가 아닙니다.
시나리오 6	테이블이 선택 이후 및 갱신 이전에 재구성됩니다. :h_rid로 식별되는 행 ID가 행을 찾지 못하거나 다른 토큰을 갖는 행을 포함하므로 갱신에 실패합니다. 이것은 행에 행 변경 시간소인 컬럼이 존재하는 경우에도 피할 수 없는 거짓 부정(false negative)의 양식입니다.	행 자체는 재구성에 의해 갱신되지 않지만 술어의 RID_BIT 분할 영역이 재구성 후 원래 행을 식별할 수 없습니다.

시나리오: 내재적으로 숨겨진 컬럼을 사용하는 낙관적 잠금

다음 시나리오는 응용프로그램에서 내재적으로 숨겨진 컬럼, 즉 IMPLICITLY HIDDEN 속성으로 정의된 컬럼을 사용하여 낙관적 잠금이 구현되는 방법을 나타냅니다.

이 시나리오의 경우 SALARY_INFO 테이블이 세 개의 컬럼을 갖고 정의되며 첫 번째 컬럼은 값이 항상 생성되는 내재적으로 숨겨진 ROW CHANGE TIMESTAMP 컬럼이라고 가정합니다.

시나리오 1:

다음 명령문에서 내재적으로 숨겨진 컬럼은 컬럼 목록에서 명시적으로 참조되며 값은 VALUES절에서 제공됩니다.

```
INSERT INTO SALARY_INFO (UPDATE_TIME, LEVEL, SALARY)
  VALUES (DEFAULT, 2, 30000)
```

시나리오 2:

다음 INSERT문은 내재적 컬럼 목록을 사용합니다. 내재적 컬럼 목록은 내재적으로 숨겨진 컬럼을 포함하지 않으므로 VALUES절은 다른 두 개의 컬럼에 대한 값만을 포함합니다.

```
INSERT INTO SALARY_INFO
  VALUES (2, 30000)
```

이 경우 UPDATE_TIME 컬럼이 디폴트 값을 갖도록 정의되어야 하며 디폴트 값은 삽입된 행으로 사용됩니다.

시나리오 3:

다음 명령문에서 내재적으로 숨겨진 컬럼이 선택 목록에서 명시적으로 참조되며 그에 대한 값이 결과 세트에 나타납니다.

```
SELECT UPDATE_TIME, LEVEL, SALARY FROM SALARY_INFO
  WHERE LEVEL = 2

  UPDATE_TIME        LEVEL    SALARY
  -------------------------- ----------- -----------
  2006-11-28-10.43.27.560841      2    30000
```

시나리오 4:

다음 명령문에서 컬럼 목록이 * 표기법의 사용을 통해 내재적으로 생성되며, 내재적으로 숨겨진 컬럼이 결과 세트에 나타나지 않습니다.

```
SELECT * FROM SALARY_INFO
  WHERE LEVEL = 2

  LEVEL    SALARY
  ----------- -----------
       2    30000
```

시나리오 5:

다음 명령문에서는 컬럼 목록이 * 표기법의 사용을 통해 내재적으로 생성되며, 내재적으로 숨겨진 컬럼 값도 ROW CHANGE TIMESTAMP FOR 표현식을 사용하여 나타납니다.

```
SELECT ROW CHANGE TIMESTAMP FOR SALARY_INFO
  AS ROW_CHANGE_STAMP, SALARY_INFO.*
  FROM SALARY_INFO WHERE LEVEL = 2
```

결과 테이블은 시나리오 3과 비슷합니다(UPDATE_TIME 컬럼이 ROW_CHANGE_STAMP가 됨).

시나리오: 시간별 갱신 발견

이 시나리오는 세 가지 시나리오를 포함하여 시간소인에 의한 갱신 발견을 사용하여 낙관적 잠금이 응용프로그램에서 구현되는 방법을 나타냅니다.

이 시나리오에서 응용프로그램은 지난 30일 동안 변경된 모든 행을 선택합니다.

```
SELECT * FROM TAB WHERE
  ROW CHANGE TIMESTAMP FOR TAB <=
  CURRENT TIMESTAMP AND
  ROW CHANGE TIMESTAMP FOR TAB >=
  CURRENT TIMESTAMP - 30 days;
```

시나리오 1:

테이블에서 행 변경 시간소인 컬럼이 정의되지 않았습니다. 명령문은 SQL20431N과 함께 실패합니다. 이 SQL 표현식은 행 변경 시간소인 컬럼이 정의된 테이블에서만 지원됩니다.

주: 이 시나리오는 z/OS에서 작동합니다.

시나리오 2:

테이블이 작성될 때 행 변경 시간소인 컬럼이 정의되었습니다.

```
CREATE TABLE TAB ( ..., RCT TIMESTAMP NOT NULL
                        GENERATED ALWAYS
                        FOR EACH ROW ON UPDATE AS
                        ROW CHANGE TIMESTAMP)
```

이 명령문이 지난 30일 동안 삽입 또는 갱신된 모든 행을 리턴합니다.

시나리오 3:

지난 30일 중에 ALTER TABLE문을 사용하여 테이블에 행 변경 시간소인 컬럼이 추가되었습니다.

```
ALTER TABLE TAB ADD COLUMN RCT TIMESTAMP NOT NULL
                                GENERATED ALWAYS
                                FOR EACH ROW ON UPDATE AS
                                ROW CHANGE TIMESTAMP
```

이 명령문은 테이블의 모든 행을 리턴합니다. ALTER TABLE문 이후 수정되지 않은 모든 행은 ALTER TABLE문 자체의 시간소인의 디폴트값을 사용하며, 그 이후에 수정된 다른 모든 행은 고유한 시간소인을 갖습니다.

제 12 장 제한조건

모든 비즈니스에서 데이터는 종종 특정 제한사항 또는 규칙을 준수해야 합니다. 예를 들어, 직원 번호는 고유해야 합니다. 데이터베이스 관리 프로그램은 이러한 규칙을 적용하기 위한 방법으로 *제한조건*을 제공합니다.

다음과 같은 제한조건 유형이 있습니다.

- NOT NULL 제한조건
- 고유(또는 고유 키) 제한조건
- 기본 키 제한조건
- 외부 키(또는 참조 무결성) 제한조건
- (테이블) 점검 제한조건
- 정보용 제한조건

제한조건은 테이블하고만 연관되며, CREATE TABLE문을 사용하여 테이블 작성 프로세스의 파트로 정의되거나 ALTER TABLE문을 사용하여 테이블 작성 후 테이블의 정의에 추가됩니다. ALTER TABLE문을 사용하여 제한조건을 수정할 수 있습니다. 대부분의 경우 언제든지 기존 제한조건을 삭제할 수 있습니다. 삭제 조치는 테이블의 구조나 테이블에 저장되어 있는 데이터에 영향을 주지 않습니다.

주: 고유 제한조건 및 1차 제한조건은 테이블 오브젝트하고만 연관되며, 종종 하나 이상의 고유 또는 기본 키 인덱스 사용을 통해 적용됩니다.

제한조건 유형

*제한조건*은 최적화를 위해 사용되는 규칙입니다.

다섯 가지 유형의 제한조건이 있습니다.

- *NOT NULL 제한조건*은 테이블의 하나 이상의 컬럼에 널(NULL) 값이 입력되지 않도록 하는 규칙입니다.
- *고유 제한조건*(*고유 키 제한조건*이라고도 함)은 테이블의 하나 이상의 컬럼에서 값이 중복되지 않도록 하는 규칙입니다. 고유 키 및 기본 키가 고유 제한조건으로 지원됩니다. 예를 들어, 제공업체 테이블의 제공업체 ID에 대해 고유 제한조건을 정의하여 동일한 제공업체 ID가 두 제공업체에 지정되지 않도록 할 수 있습니다.
- *기본 키 제한조건*은 고유 제한조건과 동일한 등록 정보를 가진 하나의 컬럼 또는 여러 컬럼의 조합입니다. 기본 키 제한조건 및 외부 키 제한조건을 사용하여 테이블 간의 관계를 정의할 수 있습니다.

- *외부 키 제한조건(참조 제한조건 또는 참조 무결성 제한조건*이라고도 함)은 하나 이상의 테이블에 있는 하나 이상의 컬럼의 값에 대한 논리적 규칙입니다. 예를 들면, 테이블 세트가 기업의 제공업체에 대한 정보를 공유합니다. 제공업체 이름이 변경되는 경우도 있습니다. 테이블에 있는 제공업체 ID가 제공업체 정보의 제공업체 ID와 일치해야 함을 지정하는 참조 제한조건을 정의할 수 있습니다. 이 제한조건은 삽입, 갱신 또는 삭제 조작을 예방함으로써 제공업체 정보가 누락되지 않도록 합니다.
- *(테이블) 점검 제한조건(점검 제한조건*이라고도 함)은 특정 테이블에 추가되는 데이터에 대한 제한사항을 설정합니다. 예를 들어, 테이블 점검 제한조건은 인사 정보가 들어 있는 테이블에서 급여 데이터가 추가되거나 갱신될 때마다 직원의 급여 수준이 최소 $20,000가 되도록 할 수 있습니다.

*정보용 제한조건*은 특정 제한조건 유형의 속성이지만 데이터베이스 관리 프로그램이 이를 강제 적용할 수 없습니다.

NOT NULL 제한조건

NOT NULL 제한조건은 널(NULL) 값이 컬럼에 입력되지 않도록 합니다.

널(NULL) 값은 데이터베이스에서 알 수 없는 상태를 나타내기 위해 사용됩니다. 디폴트로, 데이터베이스 관리 프로그램에서 제공되는 모든 내장 데이터 유형은 널(NULL) 값 표시를 지원합니다. 그러나 일부 비즈니스 규칙은 항상 값을 제공하도록 규정합니다. 예를 들면, 모든 직원은 비상 연락처 정보를 제공해야 합니다. 지정된 테이블 컬럼에 절대 널(NULL) 값이 지정되지 않도록 하기 위해 NOT NULL 제한조건을 사용합니다. 특정 컬럼에 NOT NULL 제한조건이 정의되면 해당 컬럼에 널(NULL) 값을 입력하려는 모든 삽입 또는 갱신 조작은 실패합니다.

제한조건이 특정 테이블에만 적용되기 때문에 제한조건은 일반적으로 테이블 작성 프로세스 중에 테이블의 속성과 함께 정의됩니다. 다음 CREATE TABLE문은 특정 컬럼에 NOT NULL 제한조건을 정의하는 방법을 표시합니다.

```
CREATE TABLE EMPLOYEES    (. . .
                           EMERGENCY_PHONE    CHAR(14)  NOT NULL,
                           . . .
                           );
```

고유 제한조건

*고유 제한조건*은 컬럼 세트의 값이 고유하고 테이블의 모든 행에서 널(NULL)이 되지 않도록 합니다. 고유 제한조건에서 지정된 컬럼은 NOT NULL로 정의되어야 합니다. 데이터베이스 관리 프로그램은 고유 인덱스를 사용하여 고유 제한조건의 컬럼을 변경하는 동안 키의 고유성을 강제 적용합니다.

CREATE TABLE 또는 ALTER TABLE문에서 UNIQUE절을 사용하여 고유 제한조건을 정의할 수 있습니다. 예를 들면, DEPARTMENT 테이블의 일반 고유 제한조건은 부서 번호가 고유하고 널(NULL)이 아니어야 한다는 것입니다.

그림 35에서는 테이블에 대한 고유 제한조건이 존재하는 경우 중복 레코드를 테이블에 추가할 수 없음을 표시합니다.

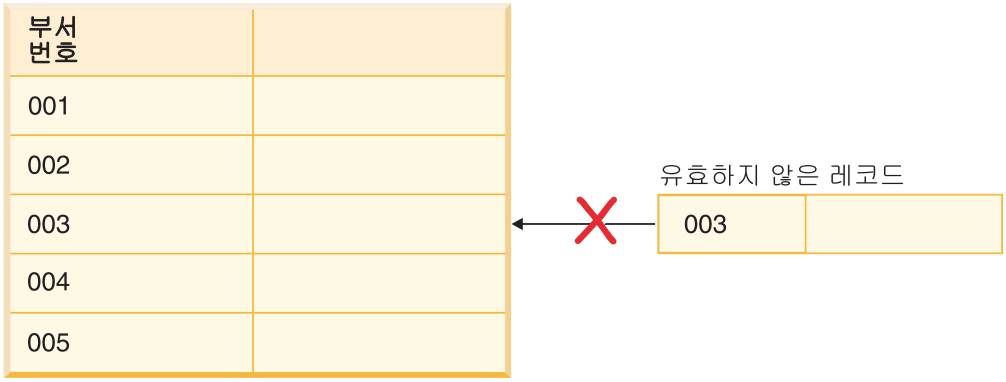


그림 35. 중복 데이터를 예방하는 고유 제한조건

데이터베이스 관리 프로그램은 삽입 및 갱신 조작 중에 제한조건을 강제 적용하여 데이터 무결성을 확인합니다.

테이블에는 임의의 수의 고유 제한조건이 있을 수 있으며 최대 하나의 고유 제한조건이 기본 키로 정의됩니다. 테이블에서는 동일한 컬럼 세트에 대해 고유 제한조건이 둘 이상 있을 수 없습니다.

참조 제한조건의 외부 키에서 참조하는 고유 제한조건을 *상위 키*라고 합니다.

- CREATE TABLE문에 고유 제한조건이 정의된 경우 데이터베이스 관리 프로그램이 자동으로 고유 인덱스를 작성하며 고유 인덱스가 1차 또는 고유 시스템 필수 인덱스로 지정됩니다.
- ALTER TABLE문에 고유 제한조건이 정의되고 동일한 컬럼에 인덱스가 있는 경우 해당 인덱스는 고유 및 시스템 필수 인덱스로 지정됩니다. 이러한 인덱스가 존재하지 않는 경우에는 데이터베이스 관리 프로그램이 자동으로 고유 인덱스를 작성하며 고유 인덱스가 1차 또는 고유 시스템 필수 인덱스로 지정됩니다.

주: 고유 제한조건 정의와 고유 인덱스 작성에는 차이점이 있습니다. 둘 다 고유성을 강제 적용하지만 고유 인덱스는 널(NULL) 입력이 가능한 컬럼을 허용하며 일반적으로 상위 키로 사용될 수 없습니다.

기본 키 제한조건

기본 키 제한조건 및 외부 키 제한조건을 사용하여 테이블 간의 관계를 정의할 수 있습니다.

기본 키는 하나의 컬럼 또는 동일한 등록 정보를 고유 제한조건으로 갖고 있는 컬럼의 조합입니다. 테이블에서 행을 식별하는 데 기본 키가 사용되기 때문에 기본 키는 고유

해야 하며 NOT NULL 속성을 갖고 있어야 합니다. 테이블에는 기본 키가 둘 이상 있을 수 없지만 고유 키는 여러 개 있을 수 있습니다. 기본 키는 선택적이며 테이블을 작성하거나 변경할 때 정의될 수 있습니다. 기본 키를 사용하면 데이터를 익스포트하거나 재구성할 경우 기본 키가 데이터 순서를 지정하므로 이점이 있습니다.

(테이블) 점검 제한조건

점검 제한조건(*테이블 점검 제한조건*이라고도 함)은 테이블의 모든 행의 하나 이상의 컬럼에서 사용할 수 있는 값을 지정하는 데이터베이스 규칙입니다. 제한된 검색 조건 양식을 통해 점검 제한조건을 지정합니다.

외부 키(참조) 제한조건

외부 키 제한조건(*참조 제한조건* 또는 *참조 무결성 제한조건*으로도 알려짐)은 사용자가 테이블 간 및 테이블 내에서 필수 관계를 정의할 수 있도록 합니다.

예를 들어, 일반 외부 키 제한조건은 DEPARTMENT 테이블에 정의된 것처럼 EMPLOYEE 테이블의 모든 직원이 기존 부서의 구성원이어야 함을 지정합니다.

*참조 무결성*은 모든 외부 키의 모든 값이 유효한 데이터베이스 상태입니다. *외부 키*는 테이블에 있는 하나의 컬럼 또는 컬럼 세트로 그 값이 해당 상위 테이블에 있는 행의 기본 키 또는 고유 키 값 중 하나 이상과 일치해야 합니다. *참조 제한조건*은 다음 조건 중 하나가 참인 경우에만 외부 키 값이 유효한 규칙입니다.

- 외부 키 값이 상위 키의 값으로 표시됩니다.
- 외부 키의 일부 구성요소가 널(NULL)입니다.

이 관계를 설정하려면 EMPLOYEE 테이블의 부서 번호를 외부 키로 정의하고 DEPARTMENT 테이블의 부서 번호를 기본 키로 정의하십시오.

325 페이지의 그림 36에서는 두 테이블 사이에 외부 키 제한조건이 있는 경우 테이블에 유효하지 않은 키를 가진 레코드가 추가되지 않도록 하는 방법을 표시합니다.

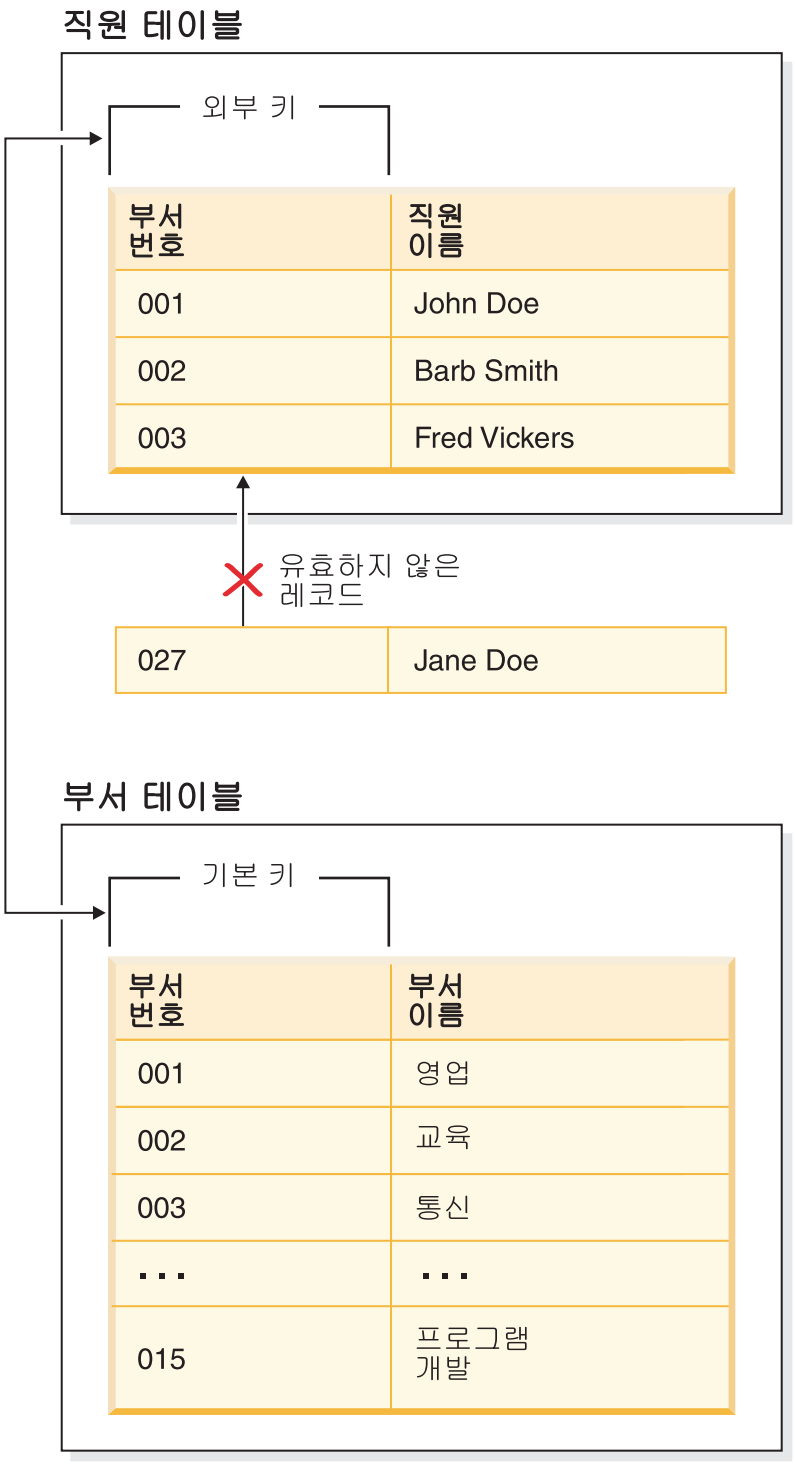


그림 36. 외부 및 기본 키 제한조건

상위 키를 포함하고 있는 테이블을 참조 제한조건의 *상위 테이블*이라고 하며 외부 키를 포함하고 있는 테이블을 해당 테이블의 *종속*이라고 합니다.

CREATE TABLE문 또는 ALTER TABLE문에서 참조 제한조건을 정의할 수 있습니다. INSERT, UPDATE, DELETE, ALTER TABLE, MERGE, ADD CONSTRAINT 및 SET INTEGRITY문 실행 중에 데이터베이스 관리 프로그램이 참조 제한조건을 적용합니다.

참조 무결성 규칙에는 다음과 같은 용어가 사용됩니다.

표 20. 참조 무결성 용어

개념	용어
상위 키	참조 제한조건의 기본 키 또는 고유 키입니다.
상위 행	하나 이상의 종속 행이 있는 행입니다.
상위 테이블	참조 제한조건의 상위 키가 포함된 테이블입니다. 임의의 수의 참조 제한조건에서 테이블이 상위 테이블이 될 수 있습니다. 참조 제한조건에서 상위 테이블인 테이블이 참조 제한조건에서 종속 테이블이 될 수도 있습니다.
종속 테이블	정의에 참조 제한조건이 하나 이상 포함된 테이블입니다. 임의의 수의 참조 제한조건에서 테이블이 종속 테이블이 될 수 있습니다. 참조 제한조건에서 종속 테이블인 테이블이 참조 제한조건에서 상위 테이블이 될 수도 있습니다.
하위 테이블	테이블이 T 테이블에 종속적이거나 T의 종속 테이블의 하위 테이블인 경우 이 테이블은 T의 하위 테이블입니다.
종속 행	하나 이상의 상위 행이 있는 행입니다.
하위 행	행이 r 행에 종속적이거나 r의 종속 행의 하위 행인 경우 이 행은 r의 하위 행입니다.
참조 주기	세트의 각 테이블이 자신의 하위 테이블이 되는 참조 제한조건 세트입니다.
자체 참조 테이블	동일한 참조 제한조건에서 상위이자 종속 테이블인 테이블입니다. 이 제한조건을 *자체 참조 제한조건*이라고 합니다.
자체 참조 행	자체의 상위 행인 행입니다.

참조 제한조건의 용도는 테이블 관계를 유지하고 데이터 항목 규칙을 준수하도록 하기 위한 것입니다. 이는 참조 제한조건이 적용되는 한 데이터베이스 관리 프로그램이 외부 키 컬럼에 널(NULL)이 아닌 값이 있는 하위 테이블의 각 행과 관련하여 상위 키에 일치하는 값이 있는 해당 상위 테이블에 행이 존재하도록 보장함을 의미합니다.

SQL 조작에서 참조 무결성이 손상되는 방식으로 데이터를 변경하려고 하면 외부 키(또는 참조) 제한조건에 위배됩니다. 데이터베이스 관리 프로그램은 각 참조 제한조건과 연관된 규칙 세트를 적용하여 이러한 유형의 상황을 처리합니다. 이러한 규칙 세트는 다음 규칙으로 구성됩니다.

- 삽입 규칙
- 갱신 규칙
- 삭제 규칙

SQL 조작에서 참조 무결성이 손상되는 방식으로 데이터를 변경하려고 하면 참조 제한조건에 위배됩니다. 예를 들면, 다음과 같습니다.

- 삽입 조작에서 해당 상위 테이블의 상위 키에 있는 값과 일치하지 않는 외부 키 컬럼의 값을 가진 하위 테이블에 데이터 행을 추가하려 합니다.

- 갱신 조작에서 하위 테이블의 외부 키 컬럼에 있는 값을 해당 상위 테이블에서 일치하는 값이 없는 값으로 변경하려 합니다.
- 갱신 조작에서 상위 테이블의 상위 키에 있는 값을 하위 테이블의 외부 키 컬럼에 일치하는 값이 없는 값으로 변경하려 합니다.
- 삭제 조작에서 하위 테이블의 외부 키 컬럼에 일치하는 값이 있는 상위 테이블에서 레코드를 제거하려 합니다.

데이터베이스 관리 프로그램은 각 참조 제한조건과 연관된 규칙 세트를 적용하여 이러한 유형의 상황을 처리합니다. 이러한 규칙 세트는 다음 규칙으로 구성됩니다.

- 삽입 규칙
- 갱신 규칙
- 삭제 규칙

삽입 규칙

참조 제한조건의 삽입 규칙은 외부 키의 널(NULL)이 아닌 삽입 값이 상위 테이블의 일부 상위 키 값과 일치해야 하는 것입니다. 복합 외부 키 값은 값의 구성요소에 널(NULL)이 있는 경우 널(NULL)입니다. 외부 키 지정 시 이 규칙이 내재됩니다.

갱신 규칙

참조 제한조건의 갱신 규칙은 참조 제한조건 정의 시 지정됩니다. NO ACTION 및 RESTRICT를 선택할 수 있습니다. 갱신 규칙은 상위 테이블의 행 또는 종속 테이블의 행이 갱신되는 경우 적용됩니다.

상위 행의 경우 상위 키 컬럼의 값이 갱신되면 다음 규칙이 적용됩니다.

- 종속 테이블의 행 중 키의 원래 값과 일치하는 행이 있는 경우 갱신 규칙이 RESTRICT이면 갱신이 거부됩니다.
- 갱신 명령문 완료 시 종속 테이블의 행 중 해당 상위 키가 없는 행이 있는 경우 (AFTER 트리거 제외) 갱신 규칙이 NO ACTION이면 갱신이 거부됩니다.

갱신 규칙이 RESTRICT이고 종속 행이 하나 이상 있는 경우에는 상위 고유 키의 값을 변경할 수 없습니다. 그러나 갱신 규칙이 NO ACTION인 경우 Update문이 완료될 때까지 모든 하위 키가 상위 키를 갖게 되면, 상위 고유 키를 갱신할 수 있습니다. 외부 키의 널(NULL)이 아닌 갱신 값은 관계의 상위 테이블에서 기본 키 값과 같아야 합니다.

또한 참조 제한조건의 갱신 규칙으로 NO ACTION 또는 RESTRICT를 사용하면 제한조건이 적용되는 시기를 판별할 수 있습니다. RESTRICT 갱신 규칙은 CASCADE 또는 SET NULL과 같은 수정 규칙과 함께 이러한 참조 제한조건을 포함하는 다른 모

든 제한조건 이전에 적용됩니다. NO ACTION 갱신 규칙은 다른 참조 제한조건 이후에 적용됩니다. 리턴되는 SQLSTATE는 갱신 규칙이 RESTRICT인지 또는 NO ACTION인지 여부에 따라 다릅니다.

종속 행의 경우 외부 키 지정 시 NO ACTION 갱신 규칙이 내재됩니다. NO ACTION은 갱신 명령문 완료 시 널(NULL)이 아닌 외부 키 값이 상위 테이블의 일부 상위 키 값과 일치해야 함을 의미합니다.

복합 외부 키 값은 값의 구성요소에 널(NULL)이 있는 경우 널(NULL)입니다.

삭제 규칙

참조 제한조건 정의 시 참조 제한조건의 삭제 규칙이 지정됩니다. NO ACTION, RESTRICT, CASCADE 또는 SET NULL을 선택할 수 있습니다. 외부 키의 일부 컬럼에서 널(NULL) 값이 허용되는 경우에만 SET NULL을 지정할 수 있습니다.

식별된 테이블 또는 식별된 뷰의 기본 테이블이 상위 테이블인 경우, 삭제하도록 선택된 행에는 RESTRICT 삭제 규칙과 관련하여 종속된 것이 없어야 하며 DELETE는 RESTRICT 삭제 규칙과 관련하여 종속 항목이 있는 하위 행에 연쇄될 수 없습니다.

삭제 조작이 RESTRICT 삭제 규칙으로 금지되지 않는 경우 선택된 행이 삭제됩니다. 선택된 행에 종속되는 행도 영향을 받습니다.

- 또한 SET NULL의 삭제 규칙과의 관계에서 종속인 행에 대한 외부 키의 널(NULL) 값이 될 수 있는 컬럼은 널(NULL) 값으로 설정됩니다.
- CASCADE의 삭제 규칙과의 관계에서 종속인 행도 삭제되고 같은 규칙이 해당 행에 적용됩니다.

다른 참조 제한조건이 시행된 후 널(NULL) 값이 아닌 외부 키가 기존의 상위 행을 참조할 수 있도록 하기 위해 NO ACTION의 삭제 규칙이 점검됩니다.

참조 제한조건의 삭제 규칙은 상위 테이블의 *행*이 삭제되는 경우에만 적용됩니다. 정확히 설명하자면, 삭제 규칙은 상위 테이블의 *행*이 삭제 또는 전파된 삭제 조작의 오브젝트이고(아래 정의됨) 해당 행이 참조 제한조건의 종속 테이블에 종속되는 경우에 적용됩니다. 예를 들어 P는 상위 테이블, D는 종속 테이블이고 p는 상위 행으로서 삭제 또는 전파된 삭제 조작의 오브젝트라고 합니다. 삭제 규칙은 다음과 같이 적용됩니다.

- RESTRICT 또는 NO ACTION을 지정하는 경우 오류가 발생하고 행이 삭제되지 않습니다.
- CASCADE를 지정하는 경우 삭제 조작이 테이블 D에 있는 p의 종속항목에 전파됩니다.
- SET NULL을 지정하는 경우 테이블 D에 있는 p의 각 종속항목에 대한 외부 키의 모든 널(NULL) 입력이 가능한 컬럼이 널(NULL)로 설정됩니다.

P의 삭제 조작에 관련되는 모든 테이블을 P와 *연속 삭제*된다고 말합니다. 따라서 테이블이 P의 종속 테이블이거나 P에서 삭제 조작이 연쇄되는 테이블의 종속 테이블인 경우 테이블이 P 테이블과 연속 삭제됩니다.

다음 제한사항이 연속 삭제 관계에 적용됩니다.

- 둘 이상의 테이블의 참조 주기에서 테이블이 자신과 연속 삭제되는 경우 주기에 RESTRICT 또는 SET NULL의 삭제 규칙이 포함될 수 없습니다.
- 테이블은 CASCADE 관계(자체 참조 또는 다른 테이블 참조)에서 종속 테이블이면서 동시에 RESTRICT 또는 SET NULL의 삭제 규칙과 자체 참조 관계에 있을 수 없습니다.
- 테이블이 다중 관계를 통해 다른 테이블과 연속 삭제되고 이러한 관계에 오버랩되는 외부 키가 있는 경우, 이들 관계에는 동일한 삭제 규칙이 있어야 하며 삭제 규칙은 SET NULL이 될 수 없습니다.
- 테이블이 관계 중 하나가 SET NULL 삭제 규칙으로 지정된 다중 관계를 통해 다른 테이블과 연속 삭제되는 경우, 이 관계의 외부 키 정의에 분산 키 또는 MDC 키 컬럼, 테이블 파티션 키 컬럼 또는 RCT 키 컬럼이 없어야 합니다.
- 두 개의 테이블이 CASCADE 관계를 통해 동일한 테이블과 연속 삭제되는 경우 해당 두 테이블이 서로 연속 삭제되어서는 안 됩니다. 여기서 연속 삭제 경로는 삭제 규칙 RESTRICT 또는 SET NULL로 끝납니다.

정보용 제한조건

*정보용 제한조건*은 SQL 컴파일러에서 데이터에 대한 액세스를 향상시키기 위해 사용할 수 있는 제한조건 속성입니다. 정보용 제한조건은 데이터베이스 관리 프로그램을 통해 적용되지 않으며, 추가적인 데이터 검증에 사용되지 않습니다. 정보용 제한조건은 쿼리 성능을 향상시키는 데 사용됩니다.

CREATE TABLE 또는 ALTER TABLE문을 사용하여 정보용 제한조건을 정의합니다. 제일 먼저 참조 무결성 또는 점검 제한조건을 추가한 다음 제한조건 속성을 이들 제한조건과 연관시켜 데이터베이스 관리 프로그램이 제한조건을 적용할지 여부 및 쿼리 최적화에 제한조건을 사용할지 여부를 지정합니다.

제한조건 설계

제한조건을 설계 및 작성하는 경우 다른 유형의 제한조건을 적절하게 식별하는 이름 지정 규칙을 사용하는 것이 좋습니다. 이는 발생할 수 있는 오류 진단에 특히 중요합니다.

다음 유형의 제한조건을 설계할 수 있습니다.

- NOT NULL 제한조건
- 고유 제한조건

- 기본 키 제한조건
- (테이블) 점검 제한조건
- 외부 키(참조) 제한조건
- 정보 제한조건

고유 제한조건 설계

*고유 제한조건*은 지정한 키의 모든 값이 고유함을 보장합니다. 테이블에는 기본 키로 정의된 하나의 고유 제한조건과 함께 여러 개의 고유 제한조건이 있을 수 있습니다.

제한사항

- 고유 제한조건은 서브테이블에서 정의될 수 없습니다.
- 테이블당 기본 키는 하나여야 합니다.

CREATE TABLE 또는 ALTER TABLE문에서 UNIQUE절을 사용하여 고유 제한조건을 정의합니다. 고유 키는 하나 이상의 컬럼으로 구성될 수 있습니다. 테이블에는 하나 이상의 고유 제한조건이 있을 수 있습니다.

설정되면 INSERT 또는 UPDATE문이 테이블의 데이터를 수정하는 경우 데이터베이스 관리 프로그램에서 고유 제한조건을 자동으로 적용합니다. 고유 제한조건은 고유 인덱스를 통해 적용됩니다.

ALTER TABLE문에 고유 제한조건이 정의되어 있고 인덱스가 해당 고유 키의 동일한 컬럼 세트에 있는 경우 해당 인덱스가 고유 인덱스가 되고 제한조건에서 사용됩니다.

임의의 고유 제한조건 하나를 취하여 기본 키로 사용할 수 있습니다. 기본 키는 다른 고유 제한조건과 함께 참조 제한조건의 상위 키로 사용될 수 있습니다. CREATE TABLE 또는 ALTER TABLE문에서 PRIMARY KEY절을 사용하여 기본 키를 정의합니다. 기본 키는 하나 이상의 컬럼으로 구성될 수 있습니다.

1차 인덱스는 기본 키의 값을 고유하게 만듭니다. 기본 키를 사용하여 테이블이 작성되면 데이터베이스 관리 프로그램은 해당 키에 1차 인덱스를 작성합니다.

고유 제한조건으로 사용되는 인덱스의 몇 가지 성능 추가 정보는 다음과 같습니다.

인덱스를 사용하여 빈 테이블을 처음 로드하는 경우 IMPORT보다 LOAD를 사용하는 것이 성능 향상에 더욱 도움이 됩니다. 이 때 LOAD의 INSERT 또는 REPLACE 모드를 사용하는지 여부는 관계가 없습니다. IMPORT INSERT 또는 LOAD INSERT를 사용하여 인덱스가 포함된 기존 테이블에 상당한 양의 데이터를 추가하는 경우 IMPORT보다 LOAD를 사용하는 것이 성능에 약간 더 도움이 됩니다. 처음으로 IMPORT 명령을 사용하여 많은 양의 데이터를 로드하는 경우 해당 데이터가 임포트 또는 로드된 후 고유 키를 작성합니다. 이렇게 하면 테이블이 로드되는 동안 인덱스의 유지보수로 인

한 오버헤드를 피할 수 있습니다. 또한 이렇게 하면 적은 스토리지 공간을 사용하는 인덱스가 생성됩니다. REPLACE 모드에서 로드 유틸리티를 사용하면 데이터를 로드하기 전에 고유 키를 작성합니다. 이러한 경우 로드 중 인덱스를 작성하는 것이 로드 후 CREATE INDEX문을 사용하는 것보다 효과적입니다.

기본 키 제한조건 설계

각 테이블에는 하나의 기본 키가 있을 수 있습니다. 기본 키는 하나의 컬럼 또는 동일한 등록 정보를 고유 제한조건으로 갖고 있는 컬럼의 조합입니다. 기본 키 제한조건 및 외부 키 제한조건을 사용하여 테이블 간의 관계를 정의할 수 있습니다.

기본 키는 테이블에서 행을 식별하는 데 사용되므로 고유해야 하고 추가 또는 삭제되는 경우가 드물어야 합니다. 테이블에는 기본 키가 둘 이상 있을 수 없지만 고유 키는 여러 개 있을 수 있습니다. 기본 키는 선택적이며 PRIMARY KEY절을 사용하여 테이블을 작성 또는 변경하는 경우 정의될 수 있습니다. 기본 키를 사용하면 데이터를 익스포트하거나 재구성할 경우 기본 키가 데이터 순서를 지정하므로 이점이 있습니다.

기본 키 제한조건은 330 페이지의 『고유 제한조건 설계』에 설명된 것처럼 고유 제한조건과 같이 설계됩니다. 유일한 차이점은 테이블당 하나의 기본 키 제한조건만 지정할 수 있는 반면 여러 고유 제한조건을 지정할 수 있다는 것입니다.

주: 복합 기본 키를 바탕으로 기본 키 제한조건을 지정할 수 있습니다.

점검 제한조건 설계

점검 제한조건을 작성할 때 다음 중 한 가지 경우가 발생할 수 있습니다. (i) 모든 행이 점검 제한조건 충족하는 경우 또는 (ii) 일부 또는 모든 행이 점검 제한조건을 충족하지 않는 경우

모든 행이 점검 제한조건 충족

모든 행이 점검 제한조건을 충족하는 경우 점검 제한조건이 성공적으로 작성됩니다. 제한조건 비즈니스 규칙을 충족하지 않는 데이터 삽입 또는 갱신 시도는 거부됩니다.

일부 또는 모든 행이 점검 제한조건을 충족하지 않음

점검 제한조건을 충족하지 않는 일부 행이 있는 경우 점검 제한조건이 작성되지 않습니다. 즉, ALTER TABLE문에 실패합니다. EMPLOYEE 테이블에 새 제한조건을 추가하는 ALTER TABLE문은 아래와 같이 표시됩니다. 점검 제한조건의 이름은 CHECK_JOB입니다. 데이터베이스 관리 프로그램은 이 이름을 사용하여 INSERT 또는 UPDATE문에 실패한 경우 제한조건이 위반되었는지 여부를 알려줍니다. CHECK절은 테이블 점검 제한조건을 정의하는 데 사용됩니다.

```
ALTER TABLE EMPLOYEE
  ADD CONSTRAINT check_job
  CHECK (JOB IN ('Engineer', 'Sales', 'Manager'));
```

테이블이 이미 정의되어 있으므로 ALTER TABLE문이 사용되었습니다. EMPLOYEE 테이블에 정의된 제한조건과 충돌하는 값이 있는 경우 ALTER STATEMENT는 완료되지 않습니다.

비즈니스 규칙 구현에 점검 제한조건 및 기타 유형의 제한조건이 사용되므로 이러한 제한조건을 간혹 변경해야 합니다. 이러한 경우는 조직에서 비즈니스 규칙이 변경되면 발생합니다. 점검 제한조건을 변경해야 할 때마다 해당 제한조건을 삭제하고 제한조건을 새로 작성해야 합니다. 점검 제한사항은 언제든지 삭제할 수 있고 이 조치는 포함된 테이블 및 데이터에 영향을 주지 않습니다. 점검 제한사항을 삭제하는 경우 제한조건에 의해 수행된 데이터 유효성 검증이 더 이상 적용되지 않는 다는 점을 명심해야 합니다.

점검 제한조건과 BEFORE 트리거 비교

데이터의 무결성을 유지하기 위해 트리거를 사용할지 또는 점검 제한조건을 사용할지 여부를 고려할 경우 점검 제한조건 간의 차이를 검토해야 합니다.

관계형 데이터베이스의 데이터에 여러 사용자가 액세스하여 변경하므로 데이터 무결성을 유지해야 합니다. 데이터를 공유할 때마다 데이터베이스에 있는 값의 정확성을 확인해야 합니다.

점검 제한조건

(테이블) 점검 제한조건은 특정 테이블에 추가되는 데이터에 대한 제한사항을 설정합니다. 테이블 점검 제한조건을 사용하여 데이터 유형의 제한사항 이외에 테이블의 컬럼에 허용되는 값에 대한 제한사항을 정의할 수 있습니다. 테이블 점검 제한조건은 범위 점검 또는 동일한 테이블의 동일한 행에 있는 기타 값에 대한 점검 양식을 갖습니다.

해당 데이터를 사용하는 모든 응용프로그램에 규칙이 적용되는 경우 테이블 점검 제한조건을 사용하여 테이블에서 허용되는 데이터에 대한 제한사항을 적용하십시오. 테이블 점검 제한조건을 사용하면 일반적으로 제한사항이 적용 가능하며 유지하기가 간편합니다.

점검 제한조건 적용은 데이터 무결성을 유지하는 데 중요하지만 특정한 양의 오버헤드를 발생시켜 볼륨이 큰 데이터를 수정할 때마다 성능에 영향을 미칠 수 있습니다.

BEFORE 트리거

갱신 또는 삽입 이전에 실행되는 트리거를 사용하여 데이터베이스를 실제로 수정하기 *이전*에 갱신 또는 삽입되는 값을 수정할 수 있습니다. 이러한 트리거를 사용하여 응용프로그램의 입력(사용자 뷰의 데이터)을 원하는 지점에서 내부 데이터베이스 형식으로 변환할 수 있습니다. 또한 BEFORE 트리거를 사용하여 기타 데이터베이스 조작이 아닌 조작을 사용자 정의 함수(UDF)를 통해 활성화시킬 수 있습니다.

수정 이외에 SIGNAL절을 사용하여 데이터를 검증하는 데 BEFORE 트리거를 일반적으로 사용합니다.

데이터 검증에 사용되는 경우 BEFORE 트리거와 점검 제한조건 사이에는 두 가지 차이점이 있습니다.

1. 점검 제한조건과는 달리 BEFORE 트리거는 동일한 테이블의 동일한 행에 있는 기타 값에 액세스하는 데 제한을 받지 않습니다.
2. 로드 조작 이후 테이블에서 무결성 설정 조작을 실행하는 동안에는 트리거(BEFORE 트리거 포함)가 실행되지 않습니다. 그러나 점검 제한조건은 검증됩니다.

외부 키(참조) 제한조건 설계

*참조 무결성*은 테이블 및 컬럼 정의에 외부 키(또는 참조) 제한조건을 추가하고 모든 외부 키 컬럼에 인덱스를 작성하여 적용됩니다. 인덱스 및 외부 키 제한조건이 정의되면 정의된 제한조건을 기준으로 테이블 및 컬럼 내에서 데이터에 대한 변경사항이 점검됩니다. 요청된 조치의 완료는 제한조건 점검 결과에 따라 달라집니다.

참조 제한조건은 CREATE TABLE 또는 ALTER TABLE문에서 FOREIGN KEY절 및 REFERENCES절로 설정됩니다. 참조 제한조건의 효과는 유형이 지정된 테이블과 참조 제한조건 작성 전에 고려해야 하는 유형이 지정된 테이블인 상위 테이블에 적용됩니다.

외부 키 식별은 테이블의 행 내 또는 두 개 테이블의 행 간의 값에 제한조건을 강제로 적용합니다. 데이터베이스 관리 프로그램은 테이블 정의에 지정된 제한조건을 점검하고 이에 따라 관계를 유지보수합니다. 이는 하나의 데이터베이스 오브젝트가 다른 데이터베이스 오브젝트를 참조할 때마다 성능을 저하시키지 않고 무결성을 유지하기 위함입니다.

예를 들어 기본 키 및 외부 키에는 각각 부서 번호 컬럼이 있습니다. EMPLOYEE 테이블의 경우 컬럼 이름은 WORKDEPT이고 DEPARTMENT 테이블의 경우 컬럼 이름은 DEPTNO입니다. 이러한 두 개 테이블 간의 관계는 다음 제한조건에 의해 정의됩니다.

- EMPLOYEE 테이블에는 각 사원에 대한 부서 번호가 하나만 있으며 해당 번호는 DEPARTMENT 테이블에 있습니다.
- EMPLOYEE 테이블의 각 행은 DEPARTMENT 테이블의 열 하나 이상과 관련되어 있습니다. 이러한 테이블 간에는 고유한 관계가 있습니다.
- WORKDEPT에 대해 널(NULL)이 아닌 값이 있는 EMPLOYEE 테이블의 각 행은 DEPARTMENT 테이블에 있는 DEPTNO 컬럼의 행과 관련이 있습니다.
- DEPARTMENT 테이블은 상위 테이블이고 EMPLOYEE 테이블은 종속 테이블입니다.

상위 테이블 DEPARTMENT를 정의하는 명령문은 다음과 같습니다.

```
CREATE TABLE DEPARTMENT
       (DEPTNO    CHAR(3)     NOT NULL,
        DEPTNAME  VARCHAR(29) NOT NULL,
               MGRNO     CHAR(6),
        ADMRDEPT  CHAR(3)     NOT NULL,
        LOCATION  CHAR(16),
           PRIMARY KEY (DEPTNO))
    IN RESOURCE
```

종속 테이블 EMPLOYEE를 정의하는 명령문은 다음과 같습니다.

```
CREATE TABLE EMPLOYEE
       (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
        FIRSTNME  VARCHAR(12) NOT NULL,
        LASTNAME  VARCHAR(15) NOT NULL,
        WORKDEPT  CHAR(3),
        PHONENO   CHAR(4),
        PHOTO     BLOB(10m)   NOT NULL,
           FOREIGN KEY DEPT (WORKDEPT)
           REFERENCES DEPARTMENT ON DELETE NO ACTION)
    IN RESOURCE
```

DEPTNO 컬럼을 DEPARTMENT 테이블의 기본 키로 지정하고 WORKDEPT를 EMPLOYEE 테이블의 외부 키로 지정하여 WORKDEPT 값에 대한 참조 제한조건을 정의합니다. 이러한 제한조건은 두 테이블의 값 간에 참조 무결성을 구현합니다. 이러한 경우 EMPLOYEE 테이블에 추가된 사원에게는 DEPARTMENT 테이블에 있는 부서 번호가 있어야 합니다.

EMPLOYEE 테이블의 참조 제한조건에 대한 삭제 규칙은 NO ACTION입니다. 즉, 해당 부서에 사원이 있으면 DEPARTMENT 테이블에서 부서를 삭제할 수 없습니다.

이전 예에서 CREATE TABLE문을 사용하여 참조 제한조건을 추가했더라도 ALTER TABLE문을 사용할 수 있습니다.

다른 예: 이전 예의 테이블 정의와 동일한 테이블 정의가 사용됩니다. 또한 DEPARTMENT 테이블이 EMPLOYEE 테이블보다 먼저 작성됩니다. 각 부서에는 관리자가 있고 해당 관리자는 EMPLOYEE 테이블에 나열됩니다. DEPARTMENT 테이블의 MGRNO는 EMPLOYEE 테이블의 실제 외부 키입니다. 이러한 참조 순환으로 인해 이 제한조건에는 약간의 문제가 있습니다. 이후에 외부 키를 추가할 수 있습니다. 또한 CREATE SCHEMA문을 사용하여 EMPLOYEE 및 DEPARTMENT 테이블을 동시에 작성할 수 있습니다.

337 페이지의 『참조 제한조건의 외부 키』의 내용도 참조하십시오.

트리거 및 참조 제한조건 사이의 상호 작용 예

갱신 조작으로 인해 참조 제한조건 및 점검 제한조건과 트리거의 상호 작용이 발생할 수 있습니다.

그림 37 및 관련 설명은 데이터베이스에서 데이터를 갱신하는 명령문에 대해 수행되는 처리를 나타냅니다.

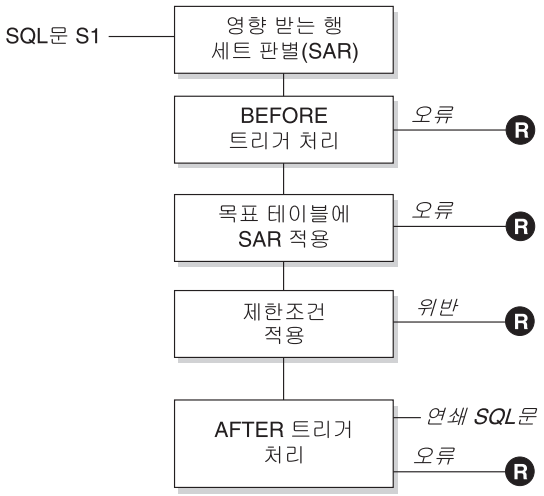


그림 37. 연관된 트리거 및 제한조건을 사용하여 명령문 처리

그림 37에서는 테이블을 갱신하는 명령문 처리의 일반적인 순서에 대해 설명합니다. 여기서는 테이블에 연쇄적으로 적용되는 BEFORE 트리거, 참조 제한조건, 점검 제한조건 및 AFTER 트리거가 포함되어 있는 상황을 가정합니다. 다음은 그림 37에 있는 상자 및 기타 항목에 대한 설명입니다.

- 명령문 S_1

  프로세스를 시작하는 DELETE, INSERT 또는 UPDATE 문입니다. 명령문 S_1은 이 설명에서 *주제 테이블*이라고 하는 테이블(또는 일부 테이블에 대한 갱신 가능 뷰)을 식별합니다.

- 영향을 받는 행 세트 판별

  이 단계에서는 AFTER 트리거에서 연쇄 명령문과 CASCADE 및 SET NULL에 대한 참조 제한조건 삭제 규칙에 대해 반복되는 프로세스가 시작됩니다.

  이 단계의 목적은 명령문의 *영향을 받는 행 세트*를 판별하는 것입니다. 포함된 행 세트는 다음 명령문에 따라 달라집니다.

  – DELETE의 경우: 명령문의 검색 조건을 충족하는 모든 행(또는 위치가 지정된 DELETE의 현재 행)
  – INSERT의 경우: VALUES절 또는 fullselect에 의해 식별되는 행
  – UPDATE의 경우: 검색 조건을 충족하는 모든 행(또는 위치가 지정된 UPDATE의 현재 행)

영향을 받는 행이 비어 있으면 BEFORE 트리거, 주제 테이블에 적용되는 변경사항 또는 명령문 프로세스에 대한 제한조건이 없습니다.

- BEFORE 트리거 처리

  모든 BEFORE 트리거는 작성 순서에 따라 오름차순으로 처리됩니다. 각 BEFORE 트리거는 영향 받은 행 세트의 각 행에 대해 한 번씩 트리거 조치를 처리합니다.

  지금까지 원래 명령문인 S_1의 결과로 변경된 모든 사항이 롤백되는 경우 트리거 조치 처리 중 오류가 발생할 수 있습니다.

  BEFORE 트리거가 없거나 영향을 받는 행 세트가 비어 있는 경우 이 단계를 건너뜁니다.

- 주제 테이블에 영향을 받는 행 세트 적용

  영향을 받는 행 세트를 사용하여 실제 삭제, 삽입 또는 갱신이 데이터베이스의 주제 테이블에 적용됩니다.

  지금까지 원래 명령문인 S_1의 결과로 변경된 모든 사항이 롤백되는 경우 영향을 받는 행 세트를 적용할 때(예: 고유한 인덱스가 존재하는 중복 키와 함께 행을 삽입하려는 경우) 트리거 조치 처리 중 오류가 발생할 수 있습니다.

- 제한조건 적용

  영향을 받는 행 세트가 비어 있는 경우 주제 테이블과 연관된 제한 조건이 적용됩니다. 이러한 제한조건에는 고유 제한조건, 고유 인덱스, 참조 제한조건, 점검 제한조건 및 뷰에서 WITH CHECK OPTION 관련 점검이 있습니다. CASCADE 또는 SET NULL 삭제 규칙이 포함된 참조 제한조건으로 인해 추가 트리거가 활성화될 수 있습니다.

  모든 제한조건 또는 WITH CHECK OPTION 위반으로 인해 오류가 발생하고 지금까지 원래 명령문인 S_1의 결과로 변경된 모든 사항이 롤백됩니다.

- AFTER 트리거 처리

  S_1로 활성화된 모든 AFTER 트리거는 작성 순서에 따라 오름차순으로 처리됩니다.

  영향을 받는 행 세트가 비어 있더라도 FOR EACH STATEMENT 트리거는 트리거 조치를 정확히 한 번만 처리합니다. FOR EACH ROW 트리거는 영향을 받은 행 세트의 각 행에 대해 한 번씩 트리거 조치를 처리합니다.

  지금까지 원래 S_1의 결과로 변경된 모든 사항이 롤백되는 경우 트리거 조치 처리 중 오류가 발생할 수 있습니다.

트리거의 트리거 조치에는 DELETE, INSERT 또는 UPDATE문인 트리거 명령문이 포함될 수 있습니다. 이렇게 설명하는 이유는 이러한 각 명령문을 *연쇄 명령문*으로 간주하기 위함입니다.

연쇄 명령문은 AFTER 트리거의 트리거 조치의 일부로 처리되는 DELETE, INSERT 또는 UPDATE문입니다. 이 명령문은 트리거 처리의 연쇄 레벨을 시작합니다. 연쇄 레벨은 트리거 명령문을 새 S_1으로 지정하고 새 여기서 설명한 모든 단계를 반복적으로 수행할 수 있습니다.

각 S_1에서 활성화한 모든 AFTER 트리거의 모든 트리거 명령문이 완료되도록 처리되면 원래 S_1의 처리가 완료됩니다.

- R = 이전 S_1에 대한 변경사항 롤백

  처리 중 발생한 모든 오류(제한조건 위반 포함)로 인해 원래 명령문 S_1의 결과로 직접 또는 간접적으로 변경된 모든 사항이 롤백됩니다. 원래 명령문 S_1이 실행되기 직전에 데이터베이스가 동일한 상태로 롤백됩니다.

참조 제한조건의 외부 키

외부 키는 동일한 테이블이나 다른 테이블에 있는 기본 키 또는 고유 키를 참조합니다. 외부 키 지정은 지정된 참조 제한조건에 따라 참조 무결성이 유지됨을 표시합니다.

CREATE TABLE 또는 ALTER TABLE문에서 FOREIGN KEY절을 사용하여 외부 키를 정의합니다. 외부 키가 정의되면 해당 테이블이 상위 테이블이라는 다른 테이블에 종속됩니다. 하나의 테이블에서 외부 키를 구성하는 컬럼 또는 컬럼 세트의 값은 상위 테이블의 고유 키 또는 기본 키 값과 일치해야 합니다.

외부 키의 컬럼 수는 상위 테이블의 해당 1차 또는 고유 제한조건(상위 키라고 함)의 컬럼 수와 같아야 합니다. 또한 키 컬럼 정의의 해당 파트에서 데이터 유형 및 길이는 동일해야 합니다. 외부 키에 *제한조건 이름*을 지정할 수 있습니다. 이름을 지정하지 않으면 임의의 이름이 자동으로 지정됩니다. 사용이 편리하도록 *제한조건 이름*을 지정하고 시스템 생성 이름을 사용하지 않는 것이 좋습니다.

만일 외부 키의 각 컬럼 값이 상위 키의 해당 컬럼의 값과 같으면 복합 외부 키의 값이 상위 키 값과 일치합니다. 정의에 따라 상위 키에는 널(NULL) 값이 있을 수 없으므로 널(NULL) 값이 포함된 외부 키는 상위 키 값과 일치할 수 없습니다. 그러나 널(NULL)이 아닌 파트의 값에 관계 없이 널(NULL) 외부 키 값은 항상 유효합니다.

다음 규칙이 외부 키 정의에 적용됩니다.

- 테이블에 여러 외부 키가 있을 수 있습니다.
- 외부 키에 널(NULL) 입력이 가능한 파트가 있는 경우 외부 키에 널(NULL) 입력이 가능합니다.
- 외부 키에 널(NULL) 값이 있는 경우 외부 키 값은 널(NULL)입니다.

외부 키를 사용하여 작업하는 경우 다음을 수행할 수 있습니다.

- 영(0)개 이상의 외부 키로 테이블을 작성합니다.
- 테이블 작성 또는 변경 시 외부 키를 정의합니다.
- 테이블 변경 시 외부 키를 삭제(drop)합니다.

유틸리티 조작에 대한 테이블 제한조건 영향

로드 중인 테이블에 참조 무결성 제한조건이 있는 경우, 로드되는 행의 참조 무결성을 검증하기 위해 로드 유틸리티가 테이블을 무결성 설정 보류 상태로 설정하여 사용자에게 테이블에서 실행할 SET INTEGRITY문이 필요하다고 알립니다. 로드 유틸리티가 완료된 후에는 SET INTEGRITY문을 실행하여 로드되는 행에 대한 참조 무결성 검사를 수행하고 테이블이 무결성 설정 보류 상태에서 벗어나도록 해야 합니다.

예를 들어, DEPARTMENT 및 EMPLOYEE 테이블만 무결성 설정 보류 상태로 설정된 경우 다음 명령문을 실행할 수 있습니다.

```
SET INTEGRITY FOR DEPARTMENT ALLOW WRITE ACCESS,
   EMPLOYEE ALLOW WRITE ACCESS,
   IMMEDIATE CHECKED FOR EXCEPTION IN DEPARTMENT,
   USE DEPARTMENT_EX,
   IN EMPLOYEE USE EMPLOYEE_EX
```

임포트 유틸리티는 다음과 같은 방식으로 참조 제한조건의 영향을 받습니다.

- 오브젝트 테이블에 그 자신 이외의 종속 항목이 있는 경우 REPLACE 및 REPLACE CREATE 함수를 사용할 수 없습니다.

  이들 함수를 사용하려면 먼저 해당 테이블이 상위 테이블인 모든 외부 키를 삭제하십시오. 임포트가 완료되면 ALTER TABLE문을 사용하여 외부 키를 다시 작성하십시오.
- 자체 참조 제한조건이 있는 테이블에 대한 임포트 성공 여부는 행이 임포트되는 순서에 따라 결정됩니다.

오브젝트 변경 시 명령문 종속성

명령문 종속성에는 패키지 및 캐시된 동적 SQL 및 XQuery문이 포함됩니다. *패키지*는 특정 응용프로그램에 가장 효과적인 방법으로 데이터에 액세스하기 위해 데이터베이스 관리 프로그램에 필요한 정보가 포함되어 있는 데이터베이스 오브젝트입니다. *바인드*는 응용프로그램 실행 시 데이터베이스 관리 프로그램에서 데이터베이스에 액세스하기 위해 필요한 패키지를 작성하는 프로세스입니다.

패키지 및 캐시된 동적 SQL 및 XQuery문은 오브젝트의 여러 유형에 따라 달라질 수 있습니다.

이러한 오브젝트는 명시적으로 참조될 수 있습니다. 예: SQL SELECT문에서 사용되는 테이블 또는 사용자 정의 함수(UDF). 또한 해당 오브젝트는 내재적으로도 참조될

수 있습니다. 예: 상위 테이블의 행이 삭제된 경우 **참조 제한조건**이 위반되지 않았는지 점검해야 하는 종속 테이블. 또한 패키지 작성자에게 부여된 특권에 따라 패키지가 달라질 수도 있습니다.

패키지 또는 캐시된 동적 쿼리 명령문이 오브젝트에 따라 달라지는 경우 해당 오브젝트가 삭제되면 패키지 또는 동적 쿼리 명령문은 『유효하지 않은』 상태로 저장됩니다. 패키지가 사용자 정의 함수에 따라 달라지는 경우 해당 함수가 삭제되면 해당 패키지는 다음과 같은 조건에서 『작동 불능』 상태로 저장됩니다.

- 유효하지 않은 상태인 캐시된 동적 SQL 및 XQuery문은 다음 사용 시 자동으로 다시 최적화됩니다. 명령문에 필요한 오브젝트가 삭제되면 동적 SQL 및 XQuery문의 실행에 실패하고 오류 메시지가 표시됩니다.
- 유효하지 않은 상태인 패키지는 다음 사용 시 내재적으로 다시 리바인드됩니다. 또한 이러한 패키지를 명시적으로 리바인드할 수도 있습니다. 트리거가 삭제되어 패키지가 유효하지 않다고 표시되면 리바인드된 패키지에서는 더 이상 해당 트리거를 호출하지 않습니다.
- 작동 불능 상태인 패키지는 사용되기 전에 명시적으로 리바인드되어야 합니다.

페더레이티드 데이터베이스 오브젝트에는 유사한 종속성이 있습니다. 예를 들어 서버를 삭제하거나 서버 정의를 변경하면 해당 서버와 연결된 별칭을 참조하는 모든 패키지 또는 캐시된 동적 SQL이 무효화됩니다.

패키지를 리바인드할 수 없는 경우도 있습니다. 예를 들어 케이블이 삭제되었으나 다시 작성되지 않은 경우 해당 패키지를 리바인드할 수 없습니다. 이러한 경우 해당 오브젝트를 다시 작성하거나 응용프로그램을 변경하여 삭제된 오브젝트를 사용하지 않도록 해야 합니다.

기타 여러 가지 경우에서 예를 들어, **제한조건** 중 하나가 삭제되면 패키지를 리바운드할 수 없습니다.

다음 시스템 카탈로그 뷰를 통해 패키지 상태 및 해당 패키지의 종속성을 판별할 수 있습니다.

- SYSCAT.PACKAGEAUTH
- SYSCAT.PACKAGEDEP
- SYSCAT.PACKAGES

정보용 제한조건 설계

레코드가 삽입되거나 갱신될 때 데이터베이스 관리 프로그램을 통해 적용되는 제한조건은 특히 참조 무결성 제한조건이 있는 대량의 레코드를 로드할 때 높은 시스템 오버헤드를 발생시킬 수 있습니다. 테이블에 레코드를 삽입하기 전에 응용프로그램이 이미 정보를 검증한 경우 일반 제한조건보다 정보용 제한조건을 사용하는 것이 더 효율적일 수 있습니다.

정보용 제한조건은 데이터베이스 관리 프로그램에 데이터가 준수하는 규칙에 대해 알리지만 데이터베이스 관리 프로그램이 해당 규칙을 적용하지는 않습니다. 그러나 DB2 옵티마이저에서 이 정보를 사용하여 SQL 쿼리 성능을 개선할 수 있습니다.

다음 예는 정보용 제한조건 사용 및 적용 방법을 보여줍니다. 이 단순 테이블에는 지원자의 연령 및 성별에 대한 정보가 포함됩니다.

```
CREATE TABLE APPLICANTS
  (
   AP_NO INT NOT NULL,
     GENDER  CHAR(1) NOT NULL,
CONSTRAINT GENDEROK
   CHECK (GENDER IN ('M', 'F'))
   NOT ENFORCED
   ENABLE QUERY OPTIMIZATION,
      AGE  INT NOT NULL,
CONSTRAINT AGEOK
   CHECK (AGE BETWEEN 1 AND 80)
   NOT ENFORCED
   ENABLE QUERY OPTIMIZATION,
 );
```

이 예에는 컬럼 제한조건의 동작을 변경하는 두 개의 절이 들어 있습니다. 첫 번째 옵션은 NOT ENFORCED로, 데이터가 삽입되거나 갱신될 때 이 컬럼의 점검을 실행하지 않도록 데이터베이스 관리 프로그램에 명령합니다.

두 번째 옵션은 ENABLE QUERY OPTIMIZATION으로, 이 테이블에 대해 SELECT 문이 실행되는 경우 데이터베이스 관리 프로그램에서 사용됩니다. 이 값이 지정되면 데이터베이스 관리 프로그램이 SQL 최적화 시 제한조건의 정보를 사용합니다.

테이블에 NOT ENFORCED 옵션이 포함되어 있는 경우 INSERT문 동작이 비정상적으로 나타날 수 있습니다. 다음 SQL을 APPLICANTS 테이블에 대해 실행하는 경우 오류가 발생하지 않습니다.

```
INSERT INTO APPLICANTS VALUES
 (1, 'M', 54),
 (2, 'F', 38),
 (3, 'M', 21),
 (4, 'F', 89),
 (5, 'C', 10),
 (6, 'S',100),
```

지원자 5번의 성별에 어린이를 나타내는 (C)가 있으며 지원자 6번의 경우 성별도 비정상적이고 AGE 컬럼의 연령 한계도 초과합니다. 제한조건이 NOT ENFORCED이므로 두 경우 모두 데이터베이스 관리 프로그램이 삽입이 발생하는 것을 허용합니다. 테이블에 대한 SELECT문의 결과가 아래 표시되어 있습니다.

```
SELECT * FROM APPLICANTS
  WHERE GENDER = 'C';
```

```
APPLICANT  GENDER  AGE
---------  ------  ---

0 레코드가 선택되었습니다.
```

테이블에서 'C' 값이 발견되었지만 데이터베이스 관리 프로그램이 올바르지 않은 답을 쿼리에 리턴했습니다. 그러나 이 컬럼에 대한 제한조건이 데이터베이스 관리 프로그램에 올바른 값은 'M' 또는 'F'뿐임을 알립니다. ENABLE QUERY OPTIMIZATION 키워드 또한 명령문 최적화 시 데이터베이스 관리 프로그램이 이 제한조건 정보를 사용하도록 허용합니다. 이와 같은 동작이 필요하지 않은 경우에는 아래 표시된 것처럼 ALTER TABLE문을 사용하여 제한조건을 변경해야 합니다.

```
ALTER TABLE APPLICANTS
  ALTER CHECK AGEOK DISABLE QUERY OPTIMIZATION
```

쿼리가 재발행되면 데이터베이스 관리 프로그램이 다음과 같이 올바른 결과를 리턴합니다.

```
SELECT * FROM APPLICANTS
  WHERE SEC = 'C';

APPLICANT GENDER  AGE
---------  ------  ---
        5  C       10

1 레코드가 선택되었습니다.
```

데이터 삽입 및 갱신을 수행하는 응용프로그램이 응용프로그램뿐임을 보장할 수 있는 경우 정보용 제한조건 사용에 대한 최상의 시나리오가 발생합니다. 응용프로그램에서 이미 성별 및 연령과 같은 모든 정보를 점검한 경우 정보용 제한조건을 사용하면 성능이 더 빨라지고 이중으로 작업할 필요가 없습니다. 데이터 웨어하우스 설계에도 정보용 제한조건을 사용할 수 있습니다.

제한조건 작성 및 수정

ALTER TABLE문을 사용하여 기존 테이블에 제한조건을 추가할 수 있습니다.

제한조건 이름은 ALTER TABLE문 내에서 지정된 다른 제한조건의 이름과 동일할 수 없으며 테이블 내에서 고유해야 합니다. 이 이름에는 정의된 참조 무결성 제한조건의 이름이 포함됩니다. 해당 명령문을 실행하려면 새 조건을 기준으로 기존 데이터를 점검합니다.

고유 제한조건 작성 및 수정

기존 테이블에 고유 제한조건을 추가할 수 있습니다. 제한조건 이름은 ALTER TABLE문 내에서 지정된 다른 제한조건의 이름과 동일할 수 없으며 테이블 내에서 고유해야 합니다. 이 이름에는 정의된 참조 무결성 제한조건의 이름이 포함됩니다. 해당 명령문을 실행하려면 새 조건을 기준으로 기존 데이터를 점검합니다.

명령행을 사용하여 고유 제한조건을 정의하려면 ALTER TABLE문의 ADD CONSTRAINT 옵션을 사용하십시오. 예를 들어 다음 명령문은 테이블에서 사원을 고유하게 식별하는 새로운 방법을 나타내는 고유 제한조건을 EMPLOYEE 테이블에 추가합니다.

```
ALTER TABLE EMPLOYEE
   ADD CONSTRAINT NEWID UNIQUE(EMPNO,HIREDATE)
```

이 제한조건을 수정하려면 삭제한 다음 다시 작성합니다.

기본 키 제한조건 작성 및 수정

기존 테이블에 기본 키 제한조건을 추가할 수 있습니다. 제한조건은 테이블 내에서 고유해야 합니다. 이 이름에는 정의된 참조 무결성 제한조건의 이름이 포함됩니다. 해당 명령문을 실행하려면 새 조건을 기준으로 기존 데이터를 점검합니다.

명령행을 사용하여 기본 키를 추가하려면 다음과 같이 입력하십시오.

```
ALTER TABLE <name>
   ADD CONSTRAINT <column_name>
   PRIMARY KEY <column_name>
```

기존 제한조건은 수정할 수 없습니다. 다른 컬럼 또는 컬럼 세트를 기본 키로 정의하려면 먼저 기존 기본 키 정의를 삭제한 다음 다시 작성해야 합니다.

점검 제한조건 작성 및 수정

테이블 점검 제한조건이 추가되면 테이블을 삽입 또는 갱신하는 패키지 및 캐시된 동적 SQL문이 유효하지 않은 것으로 표시됩니다.

명령행을 사용하여 테이블 점검 제한조건을 추가하려면 다음과 같이 입력하십시오.

```
ALTER TABLE EMPLOYEE
   ADD CONSTRAINT REVENUE CHECK (SALARY + COMM > 25000)
```

이 제한조건을 수정하려면 삭제한 다음 다시 작성합니다.

외부 키(참조) 제한조건 작성 및 수정

외부 키는 다른 테이블에 있는 데이터 값에 대한 참조입니다. 다른 유형의 외부 키 제한조건이 있습니다.

외부 키가 테이블에 추가되면 다음 명령문이 포함된 패키지 및 동적 SQL문이 유효하지 않은 것으로 표시됩니다.

- 외부 키가 포함된 테이블을 삽입 또는 갱신하는 명령문
- 상위 테이블을 갱신 또는 삭제하는 명령문입니다.

명령행을 사용하여 외부 키를 추가하려면 다음과 같이 입력하십시오.

```
ALTER TABLE <name>
   ADD CONSTRAINT <column_name>
    FOREIGN KEY <column_name>
    ON DELETE <action_type>
    ON UPDATE <action_type>
```

다음 예는 기본 키 및 외부 키를 테이블에 추가하는 ALTER TABLE문을 보여줍니다.

```
ALTER TABLE PROJECT
  ADD CONSTRAINT PROJECT_KEY
      PRIMARY KEY (PROJNO)
ALTER TABLE EMP_ACT
  ADD CONSTRAINT ACTIVITY_KEY
      PRIMARY KEY (EMPNO, PROJNO, ACTNO)
  ADD CONSTRAINT ACT_EMP_REF
      FOREIGN KEY (EMPNO)
      REFERENCES EMPLOYEE
      ON DELETE  RESTRICT
  ADD CONSTRAINT ACT_PROJ_REF
      FOREIGN KEY (PROJNO)
      REFERENCES PROJECT
      ON DELETE CASCADE
```

이 제한조건을 수정하려면 삭제한 다음 다시 작성합니다.

정보용 제한조건 작성 및 수정

쿼리 성능을 향상시키기 위해 테이블에 정보용 제한조건을 추가할 수 있습니다. DDL에 대해 NOT ENFORCED 옵션을 지정하면 CREATE TABLE 또는 ALTER TABLE문을 사용하여 정보용 제한조건을 추가합니다.

제한사항: 테이블에 대한 정보용 제한조건을 정의하면 정보용 제한조건을 제거한 후 해당 테이블의 컬럼 이름만 변경할 수 있습니다.

명령행을 사용하여 테이블에 대한 정보용 제한조건을 지정하려면 새 테이블에 대해 다음 명령을 입력하십시오.

```
ALTER TABLE <name> <constraint attributes> NOT ENFORCED
```

ENFORCED 또는 NOT ENFORCED: 제한조건이 삽입, 갱신 또는 삭제 등 정상 조작 중에 데이터베이스 관리 프로그램에 의해 시행되는지 여부를 지정합니다.

- ENFORCED를 함수적 종속성에 대해서는 지정할 수 없습니다(SQLSTATE 42621).
- NOT ENFORCED 옵션은 테이블 데이터가 개별적으로 제한조건을 확인해야 할 경우에만 지정해야 합니다. 데이터가 실제로 제한조건을 준수하지 않는 경우 쿼리 결과는 예상할 수 없습니다.

이 제한조건을 수정하려면 삭제한 다음 다시 작성합니다.

고유 또는 기본 키 제한조건을 사용하는 인덱스 재사용

ALTER TABLE 명령을 사용하여 파티션된 인덱스가 포함된 파티션된 테이블에 고유 또는 기본 키 제한조건을 추가하려는 경우 이미 존재하는 인덱스에 따라서 새 제한조건을 적용할 수 있도록 인덱스를 변경하거나 새 인덱스를 작성할 수 있습니다.

ALTER TABLE문을 실행하여 테이블의 고유 또는 기본 키를 추가하거나 변경하면 정의 중인 고유 또는 기본 키와 일치하는 기존 인덱스가 있는지 여부를 판별하기 위해 점검이 수행됩니다(INCLUDE 컬럼은 무시). 컬럼 순서나 방향(예: ASC/DESC)에 관계없이, 동일한 컬럼 세트를 식별하는 경우에는 인덱스 정의가 일치하는 것입니다.

파티션되고 고유하지 않은 인덱스가 있는 파티션된 인덱스에서 변경 중인 테이블의 인덱스 컬럼이 파티션 키를 구성하는 컬럼에 포함되지 않는 경우 해당 인덱스는 일치하는 인덱스가 아닙니다.

테이블에 일치하는 인덱스 정의가 없는 경우 UNIQUE 인덱스가 아직 없으면 UNIQUE 인덱스로 변경되고 시스템에서 필수적인 것으로 표시됩니다. 테이블에 일치하는 기존 인덱스가 하나 이상 있는 경우에는 기존 고유 인덱스가 선택됩니다. 일치하는 고유 인덱스가 하나 이상 있는 경우 또는 일치하는 고유하지 않은 인덱스가 하나 이상 있고 일치하는 고유 인덱스는 없는 경우에는 파티션된 인덱스가 선택됩니다. 그렇지 않은 경우에는 임의로 인덱스가 선택됩니다.

일치하는 인덱스를 찾을 수 없는 경우에는 컬럼에 고유 양방향 인덱스가 자동으로 작성됩니다.

테이블에 대한 제한조건 정의 보기

테이블에 대한 제한조건 정의는 SYSCAT.INDEXES 및 SYSCAT.REFERENCES 카탈로그 뷰에서 찾을 수 있습니다.

SYSCAT.INDEXES 뷰의 UNIQUERULE 컬럼은 인덱스의 특성을 나타냅니다. 이 컬럼의 값이 P이면 인덱스는 기본 키이고 이 값이 U이면 인덱스는 고유 인덱스이지만 기본 키는 아닙니다.

SYSCAT.REFERENCES 카탈로그 뷰에는 참조 무결성(외부 키) 제한조건 정보가 들어 있습니다.

제한조건 삭제

ALTER TABLE문을 사용하여 테이블 점검 제한조건을 명시적으로 삭제할 수 있습니다. 또는 DROP TABLE문의 결과로 내재적으로 삭제할 수도 있습니다.

제한조건을 삭제하려면 DROP 또는 DROP CONSTRAINT절과 함께 ALTER TABLE 문을 사용하십시오. 이렇게 하면 영향을 받는 컬럼이 포함된 테이블을 바인드하고 해당 테이블에 계속해서 액세스할 수 있습니다. 테이블에 대한 모든 고유 제한조건의 이름은 SYSCAT.INDEXES 시스템 카탈로그 뷰에서 찾을 수 있습니다.

고유 제한조건 삭제

ALTER TABLE문을 사용하여 고유 제한조건을 명시적으로 삭제할 수 있습니다.

ALTER TABLE문의 DROP UNIQUE절은 고유 제한조건 constraint-name의 정의와 이 고유 제한조건에 종속된 모든 참조 제한조건의 정의를 삭제합니다. constraint-name은 기존 고유 제한조건을 식별해야 합니다.

```
ALTER TABLE <table-name>
  DROP UNIQUE <constraint-name>
```

이 고유 제한조건을 삭제하면 해당 제한조건을 사용하는 모든 패키지 또는 캐시된 동적 SQL이 무효화됩니다.

기본 키 제한조건 삭제

ALTER TABLE문의 DROP PRIMARY KEY절을 사용하여 기본 키 제한조건을 삭제합니다.

ALTER TABLE문의 DROP PRIMARY KEY절은 기본 키의 정의와 이 기본 키가 종속된 모든 참조 제한조건의 정의를 삭제합니다. 테이블에 기본 키가 있어야 합니다. 명령행을 사용하여 기본 키를 삭제하려면 다음과 같이 입력하십시오.

```
ALTER TABLE <table-name>
  DROP PRIMARY KEY
```

(테이블) 점검 제한조건 삭제

점검 제한조건을 삭제하는 경우 해당 테이블의 INSERT 또는 UPDATE 종속성이 포함된 모든 패키지 및 캐시된 동적문이 무효화됩니다. 테이블에 대한 모든 점검 제한조건의 이름은 SYSCAT.CHECKS 카탈로그 뷰에서 찾을 수 있습니다. 이름이 시스템에서 생성된 테이블 점검 제한조건을 삭제하려면 SYSCAT.CHECKS 카탈로그 뷰에서 해당 이름을 찾으십시오.

다음 명령문은 점검 제한조건 constraint-name을 삭제합니다. constraint-name은 테이블에 대해 정의된 기존의 점검 제한조건을 식별해야 합니다. 명령행을 사용하여 테이블 점검 제한조건을 삭제하려면 다음과 같이 입력하십시오.

```
ALTER TABLE <table_name>
  DROP <check_constraint_name>
```

외부 키(참조) 제한조건 삭제

ALTER TABLE문의 DROP CONSTRAINT절을 사용하여 외부 키 제한조건을 삭제합니다.

ALTER TABLE문의 DROP CONSTRAINT절은 제한조건 *constraint-name*을 삭제합니다. *constraint-name*은 테이블에 정의된 기존의 외부 키 제한조건, 기본 키 또는 고유 제한조건을 식별해야 합니다. 명령행을 사용하여 외부 키를 삭제하려면 다음과 같이 입력하십시오.

```
ALTER TABLE <table-name>
  DROP FOREIGN KEY <foreign_key_name>
```

다음 예에서는 ALTER TABLE문에서 DROP PRIMARY KEY 및 DROP FOREIGN KEY절을 사용하여 테이블에서 기본 키 및 외부 키를 제거합니다.

```
ALTER TABLE EMP_ACT
  DROP PRIMARY KEY
  DROP FOREIGN KEY ACT_EMP_REF
  DROP FOREIGN KEY ACT_PROJ_REF
ALTER TABLE PROJECT
  DROP PRIMARY KEY
```

외부 키 제한조건이 삭제되면 다음 명령문이 포함된 패키지 또는 동적문이 유효하지 않은 것으로 표시됩니다.

- 외부 키가 포함된 테이블을 삽입 또는 갱신하는 명령문
- 상위 테이블을 갱신 또는 삭제하는 명령문

제 13 장 인덱스

*인덱스*는 하나 이상의 키 값에 의해 논리적으로 정렬되는 포인터 세트입니다. 포인터는 테이블의 행, MDC 테이블의 블록, XML 스토리지 오브젝트의 XML 데이터 등을 의미할 수 있습니다.

인덱스를 사용하여 다음을 수행합니다.

- 성능 향상. 대부분의 경우 인덱스를 사용하면 데이터 액세스가 더 빠릅니다. 뷰에 대한 인덱스를 작성할 수 없지만, 뷰가 기초로 하는 테이블에 대해 작성된 인덱스는 가끔 해당 뷰에 대한 조작 성능을 향상시킬 수 있습니다.
- 고유성 보장. 고유 인덱스를 갖는 테이블은 동일한 키를 갖는 행을 가질 수 없습니다.

데이터는 테이블에 추가될 때, 테이블이나 추가될 데이터에 대해 다른 조치가 수행되지 않았으면 맨 아래에 추가됩니다. 데이터에 내재된 순서는 없습니다. 특정 데이터의 행을 검색할 때 테이블의 첫 번째 행부터 마지막 행까지 점검해야 합니다. 인덱스가 사용 가능할 경우 테이블에서 순서대로 데이터에 액세스하기 위한 수단으로 사용됩니다.

일반적으로 테이블에서 데이터를 검색할 때 특정 값을 갖는 컬럼이 있는 행을 찾습니다. 데이터 행에 있는 컬럼 값을 사용하여 전체 행을 식별할 수 있습니다. 예를 들어 직원 번호는 대개 특정 개별 직원을 고유하게 정의할 수 있습니다. 또는 행을 식별하기 위해 둘 이상의 컬럼이 필요할 수 있습니다. 예를 들어 고객 이름과 전화번호의 조합이 필요할 수 있습니다. 데이터 행을 식별하는 데 사용되는 인덱스의 컬럼을 *키*라고 합니다. 한 컬럼이 둘 이상의 키에서 사용될 수 있습니다.

인덱스는 키의 값에 따라 순서가 정해집니다. 키는 고유하거나 고유하지 않을 수 있습니다. 각 테이블에는 최소 하나의 고유 키가 있어야 하지만 다른 고유하지 않은 키도 있을 수 있습니다. 각 인덱스는 정확히 한 개의 키를 가집니다. 예를 들어, 사원 ID 번호(고유)를 한 인덱스의 키로 사용하고 부서 번호(고유하지 않음)를 다른 인덱스의 키로 사용할 수 있습니다.

모든 인덱스가 테이블의 행을 가리키지는 않습니다. MDC 블록 인덱스는 데이터의 Extent(또는 블록)를 가리킵니다. XML 데이터에 대한 XML 인덱스는 특정 XML 패턴 표현식을 사용하여 단일 컬럼 안에 저장된 XML 문서의 경로 및 값을 인덱싱합니다. 이 컬럼의 데이터 유형은 XML이어야 합니다. MDC 블록 인덱스와 XML 인덱스 둘 다 시스템 생성 인덱스입니다.

예 :

그림 38의 테이블 A는 테이블에 있는 직원 번호를 기초로 하는 인덱스를 갖습니다. 이 키 값은 테이블 A의 행에 대한 포인터입니다. 예를 들어, 사원 번호 19가 사원 KMP를 지시합니다. 인덱스는 포인터를 통해 데이터에 대한 경로를 작성하므로 테이블에 있는 행에 효율적으로 액세스할 수 있습니다.

인덱스 키가 고유하도록 고유 인덱스를 작성할 수 있습니다. *인덱스 키*는 인덱스가 정의된 여러 개의 정렬된 컬럼 콜렉션 또는 한 개의 컬럼을 의미합니다. 고유 인덱스를 사용하면 인덱스가 정의된 컬럼에 있는 각 인덱스 키 값을 고유하게 만들 수 있습니다.

그림 38은 인덱스와 테이블의 관계를 보여줍니다.

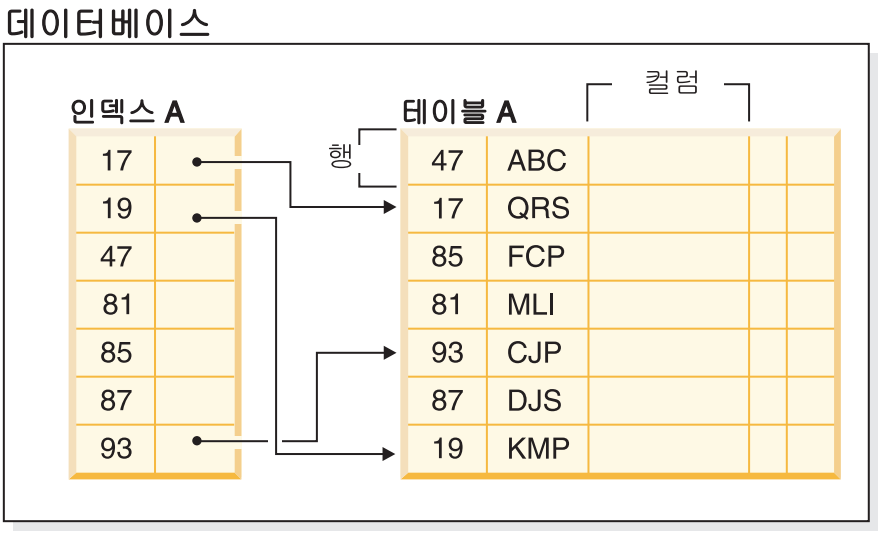


그림 38. 인덱스와 테이블의 관계

349 페이지의 그림 39는 일부 데이터베이스 오브젝트 간의 관계를 보여줍니다. 또한 테이블, 인덱스 및 LONG 데이터가 테이블 스페이스에 저장된다는 사실도 나타냅니다.

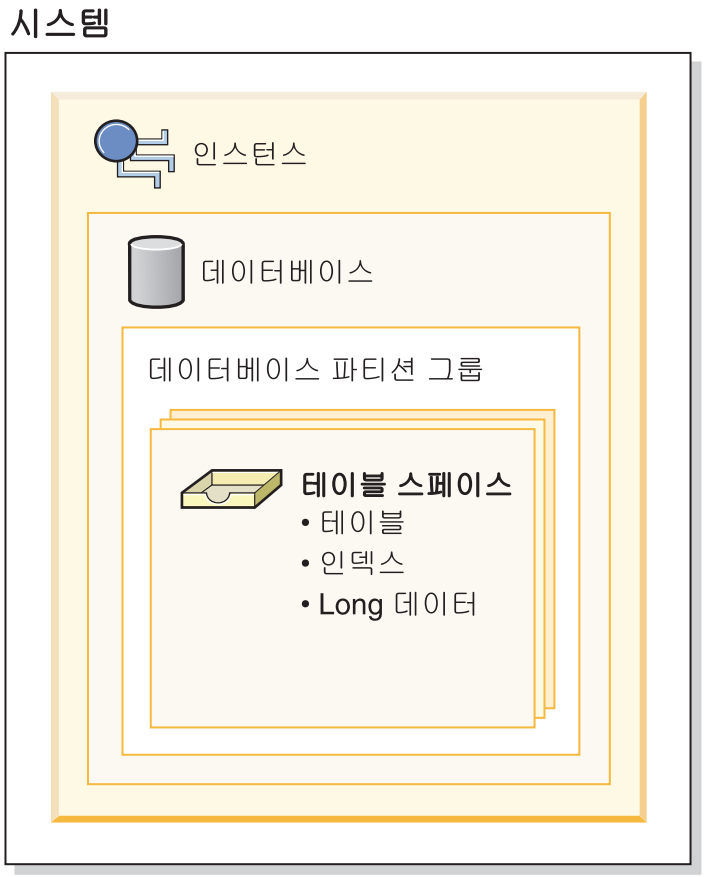


그림 39. 선택된 데이터베이스 오브젝트 간의 관계

인덱스 유형

여러 가지 목적으로 작성할 수 있는 여러 가지 유형의 인덱스가 있습니다. 예를 들어 고유 인덱스는 인덱스 키에서 고유성 제한조건을 강제합니다. 양방향 인덱스는 정방향 및 역방향 모두에서의 스캔을 허용합니다. 클러스터된 인덱스는 키순으로 테이블을 트래버스하는 쿼리의 성능을 향상시키는 데 도움이 됩니다.

고유 및 비고유 인덱스

고유 인덱스는 한 테이블에 있는 두 개의 행 데이터가 동일한 키 값을 갖지 않도록 보장하여 데이터 무결성을 유지하는 데 도움이 되는 인덱스입니다.

이미 데이터를 포함하는 테이블에 대한 고유 인덱스를 작성할 때, 인덱스를 구성하는 컬럼의 값이 고유성을 갖는지 점검됩니다. 테이블에 중복 키 값을 갖는 행이 있는 경우 인덱스 작성 프로세스가 실패합니다. 테이블에 대해 고유 인덱스가 정의된 후에는 키가 인덱스에서 추가 또는 변경될 때마다 고유성이 적용됩니다. (몇 가지 예를 들면, 여기에

는 삽입, 갱신, 로드, 임포트 및 무결성 설정이 포함됩니다.) 데이터 값의 고유성를 강제하는 것 외에, 고유 인덱스를 사용하여 쿼리 처리 중에 데이터 검색 성능을 향상시킬 수도 있습니다.

한편, 비고유 인덱스는 해당 인덱스가 연관되는 테이블에 제한조건을 강제하는 데 사용되지 않습니다. 대신, 비고유 인덱스는 자주 사용되는 데이터 값의 정렬된 순서를 유지하여 쿼리 성능을 향상시키는 데만 사용됩니다.

클러스터 및 클러스터되지 않은 인덱스

인덱스 아키텍처는 클러스터 또는 클러스터되지 않은 것으로 분류됩니다. 클러스터된 인덱스는 데이터 페이지에서 행의 순서가 인덱스에서 행의 순서에 대응하는 인덱스입니다. 이것은 주어진 테이블에 클러스터된 인덱스가 하나만 존재할 수 있는 반면, 많은 클러스터되지 않은 인덱스가 테이블에 존재할 수 있는 이유입니다. 일부 관계형 데이터베이스 관리 시스템에서 클러스터된 인덱스의 리프 노드가 다른 곳에 상주하는 데이터에 대한 포인터가 아니라 실제 데이터에 대응합니다.

클러스터 및 클러스터되지 않은 인덱스는 둘 다 인덱스 구조에 키와 레코드 ID만 포함합니다. 레코드 ID는 항상 데이터 페이지의 행을 가리킵니다. 클러스터 및 클러스터되지 않은 인덱스 사이의 유일한 차이점은 데이터베이스 관리 프로그램이 인덱스 페이지에서 대응하는 키가 나타나는 것과 동일한 순서로 데이터 페이지에서 데이터를 보존하는 것입니다. 따라서 데이터베이스 관리 프로그램은 비슷한 키를 갖는 행을 동일한 페이지에 삽입하려고 합니다. 테이블이 재구성되면 인덱스 키의 순서로 데이터 페이지에 삽입됩니다.

선택된 키에 대해 테이블을 재구성하면 데이터가 다시 클러스터링됩니다. 클러스터된 인덱스는 테이블에 있는 데이터의 순차적 액세스를 향상시킬 수 있기 때문에 범위 술어를 갖는 컬럼에 가장 유용합니다. 그 결과, 비슷한 값이 동일한 데이터 페이지에 위치하므로 페이지 페치(fetch)가 더 적어집니다.

일반적으로 한 테이블 내에서 한 인덱스만 높은 등급으로 클러스터화될 수 있습니다.

클러스터링 인덱스는 페이지에 저장된 데이터에 대한 더욱 선형적인 액세스 경로를 제공하기 때문에 대부분의 쿼리 조작의 성능을 향상시킬 수 있습니다. 또한 비슷한 인덱스 키를 갖는 행이 함께 저장되기 때문에 클러스터링 인덱스가 사용될 때 프리페치가 대개 더 효율적입니다.

그러나, 클러스터링 인덱스는 CREATE TABLE문과 함께 사용되는 테이블 정의의 파트로서 지정할 수 없습니다. 대신 클러스터링 인덱스는 CLUSTER 옵션이 지정된 CREATE INDEX문을 실행해서만 작성됩니다. 그런 다음 ALTER TABLE문을 사용하여 테이블에 작성된 클러스터링 인덱스에 대응하는 기본 키를 추가해야 합니다. 그러면 이 클러스터링 인덱스가 테이블의 기본 키 인덱스로 사용됩니다.

주: ALTER TABLE문을 사용하여 테이블의 PCTFREE를 적절한 값으로 설정하면 비슷한 값을 갖는 페이지에 행을 삽입할 충분한 여유 공간을 남겨둠으로써 테이블이 클러스터된 상태를 유지하는 데 도움이 됩니다. 자세한 정보는 *SQL 참조서*의 『ALTER TABLE문』 및 *문제점 해결 및 데이터베이스 성능 조정*의 『테이블 및 인덱스 재구성 필요성 축소』를 참조하십시오.

클러스터된 인덱스를 사용한 성능 향상

일반적으로, 클러스터링 인덱스가 고유하므로 클러스터링이 더 효과적으로 유지보수됩니다.

기본 키 또는 고유 키 제한조건과 고유 인덱스 간의 다른점

기본 고유 키 제한조건과 고유 인덱스 간에는 특별한 차이점이 없음을 이해하는 것이 중요합니다. 데이터베이스 관리 프로그램은 고유 인덱스와 NOT NULL 제한조건의 조합을 사용하여 기본 및 고유 키 제한조건의 관계형 데이터베이스 개념을 구현합니다. 그러므로 고유 인덱스는 널(NULL) 값을 허용하기 때문에 스스로 기본 키 제한조건을 강제하지 않습니다. (널(NULL) 값이 알 수 없는 값을 나타내지만, 인덱싱이 될 때는 널(NULL) 값이 다른 널(NULL) 값과 같은 것으로 취급됩니다.)

그러므로 고유 인덱스가 단일 컬럼으로 구성되는 경우 하나의 널(NULL) 값만 허용됩니다. 둘 이상의 널(NULL) 값은 고유 제한조건을 위반합니다. 비슷하게, 고유 인덱스가 다중 컬럼으로 구성되는 경우 값과 NULL 값의 특정 조합을 한 번만 사용할 수 있습니다.

양방향 인덱스

디폴트로 양방향 인덱스는 정방향 및 역방향 스캔을 둘 다 허용합니다. CREATE INDEX문의 ALLOW REVERSE SCANS절이 정방향 및 역방향 스캔, 즉 인덱스 작성 시간에 정의된 순서와 반대(또는 역방향) 순서로의 스캔을 둘 다 사용 가능하게 합니다. 이 옵션으로 다음을 수행할 수 있습니다.

- MIN 및 MAX 함수 사용
- 이전 키 페치
- 데이터베이스 관리 프로그램이 역방향 스캔을 위한 임시 테이블을 작성할 필요성 제거
- 중복 역방향 순서 인덱스 제거

DISALLOW REVERSE SCANS가 지정되면 인덱스를 역방향으로 스캔할 수 없습니다. (그러나 실제로는 ALLOW REVERSE SCANS 인덱스와 정확하게 동일합니다.)

파티션 및 파티션되지 않은 인덱스

파티션된 데이터는 데이터베이스 파티션의 단일 테이블 스페이스에 존재하는 파티션되지 않은 인덱스, 그 자체가 데이터베이스 파티션의 하나 이상의 테이블 스페이스 사이에 파티션되는 인덱스 또는 둘의 조합을 가질 수 있습니다. 파티션된 인덱스는 특히 파티션된 테이블과의 롤인 조작을 수행할 때(즉, ALTER TABLE문에서 ATTACH PARTITION절을 사용하여 데이터 파티션을 다른 테이블에 첨부할 때) 유용합니다.

파티션된 테이블에 대한 인덱스

파티션된 테이블은 데이터베이스 파티션의 단일 테이블 스페이스에 존재하는 파티션되지 않은 인덱스, 그 자체가 데이터베이스 파티션의 하나 이상의 테이블 스페이스 사이에 파티션되는 인덱스 또는 둘의 조합을 가질 수 있습니다.

파티션된 인덱스는 파티션된 테이블과의 롤인 조작을 수행할 때(즉, ALTER TABLE 문에서 ATTACH PARTITION절을 사용하여 데이터 파티션을 다른 테이블에 첨부할 때) 유용합니다. 파티션된 인덱스를 사용하면 그렇지 않은 경우 파티션되지 않은 인덱스에 대해 수행해야 하는 인덱스 유지보수를 피할 수 있습니다. 파티션된 테이블이 파티션되지 않은 인덱스를 사용할 때 SET INTEGRITY문을 사용하여 새로 조합되는 데이터 파티션에 대해 인덱스 유지보수를 수행해야 합니다. 이것이 시간이 걸릴 뿐 아니라, 롤인되는 행 수에 따라서는 많은 양의 로그 스페이스가 필요할 수도 있습니다.

파티션할 수 없는 몇 가지 유형의 인덱스가 있습니다.

- XML 데이터에 대한 인덱스
- 공간 데이터에 대한 인덱스
- MDC 블록 인덱스(시스템 생성)
- XML 컬럼 경로 인덱스(시스템 생성)

이러한 경우 인덱스는 반드시 파티션되지 않은 것으로 작성되어야 합니다. 또한 파티션된 고유 인덱스에 대한 인덱스 키는 사용자 생성 키 또는 시스템 생성 키 모두 테이블 파티션 키의 모든 컬럼을 포함해야 합니다. 후자는 데이터에 대한 고유성 또는 기본 제한조건을 강제하기 위해 시스템이 작성하는 인덱스의 경우입니다.

파티션된 테이블에 대한 파티션되지 않은 인덱스

*파티션되지 않은 인덱스*는 파티션된 테이블의 모든 행을 참조하는 단일 인덱스 오브젝트입니다. 파티션되지 않은 인덱스는 테이블 데이터 파티션이 여러 테이블 스페이스에 걸쳐있는 경우에도 항상 단일 테이블 스페이스에 독립 인덱스 오브젝트로서 작성됩니다.

파티션된 테이블에 대한 인덱스를 작성할 때, 다음을 작성 중인 경우가 아니면 디폴트로 인덱스가 *파티션된 인덱스*입니다.

- 인덱스 키가 모든 테이블 파티션 컬럼을 포함하지 않는 고유 인덱스

- 공간 인덱스
- XML 데이터에 대한 인덱스

이 경우에는 인덱스가 파티션되지 않습니다. 그러나 데이터가 파티션됨에도 불구하고 파티션되지 않은 인덱스를 작성하는 것이 유용하거나 필요한 시기가 있습니다. 이런 경우에는 CREATE INDEX문의 NOT PARTITIONED절을 사용하여 파티션된 테이블에 대한 파티션되지 않은 인덱스를 작성하십시오. 파티션되지 않은 인덱스를 작성하는 경우, 디폴트로 첫 번째 볼 수 있거나 첨부된 데이터 파티션과 동일한 테이블 스페이스에 저장됩니다. 그림 40은 테이블의 모든 파티션을 참조하는 단일 인덱스 X1의 예를 나타냅니다. 인덱스는 테이블에 대해 첫 번째로 볼 수 있는 파티션과 동일한 테이블 스페이스에 작성되었습니다.

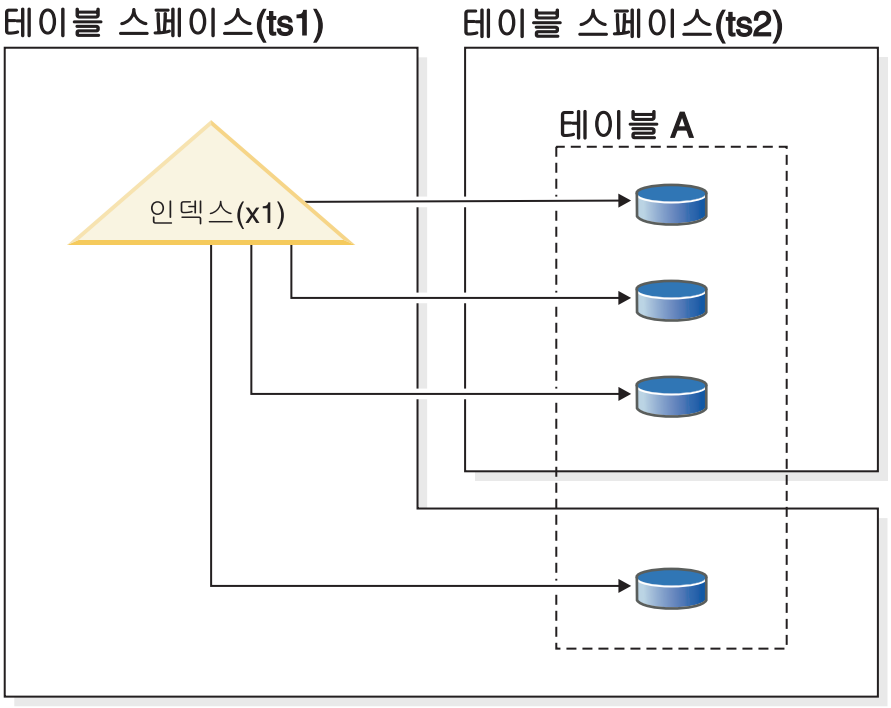


그림 40. 파티션된 테이블에 대한 파티션되지 않은 인덱스

354 페이지의 그림 41은 두 개의 파티션되지 않은 인덱스의 예를 나타냅니다. 이 경우 각 인덱스 파티션은 데이터 파티션과는 다른 테이블 스페이스에 있습니다. 다시 각 인덱스가 테이블의 모든 파티션을 참조하는 방법에 주의하십시오.

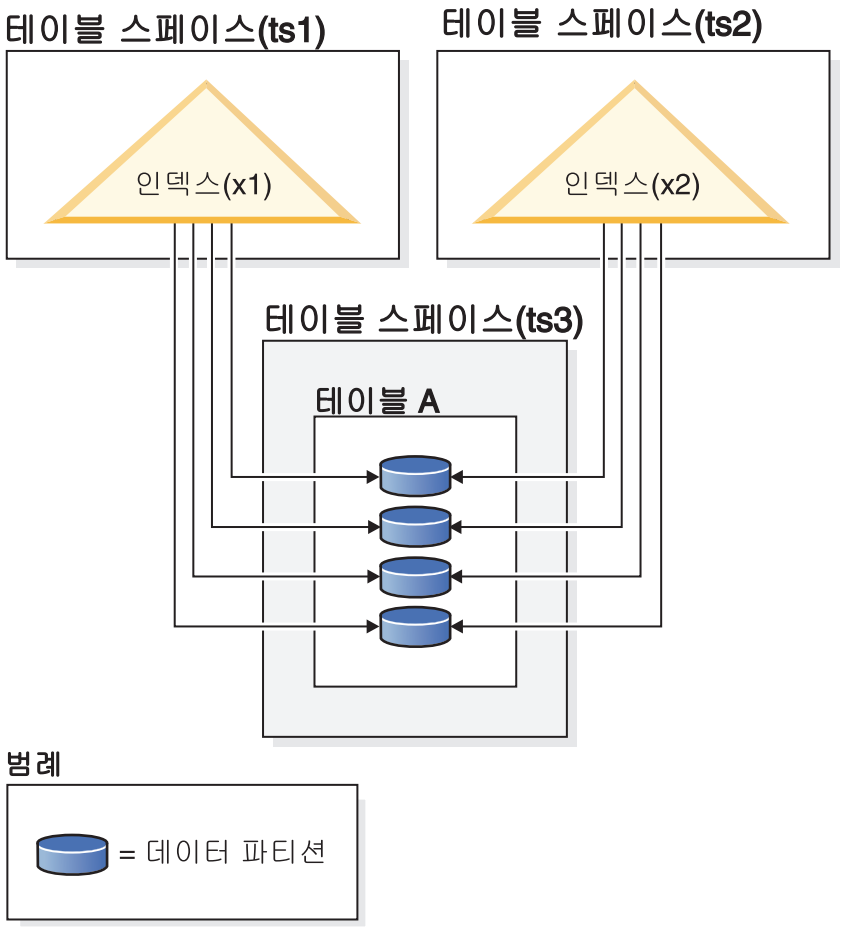


그림 41. 고유한 테이블 스페이스에 인덱스를 갖는 파티션된 테이블에 대한 파티션되지 않은 인덱스

다음 시간에 파티션되지 않은 인덱스에 대한 위치를 겹쳐쓸 수 있습니다.

- CREATE TABLE문의 INDEX IN절을 사용하여 테이블을 작성할 때
- CREATE INDEX문의 IN절을 사용하여 인덱스를 작성할 때

두 번째 접근이 항상 첫 번째 접근에 우선합니다.

ALTER TABLE문의 ATTACH PARTITION절을 사용하여 파티션된 테이블에 데이터를 롤인하는 경우 SET INTEGRITY문을 사용하여 테이블 데이터를 쿼리에 대해 온라인으로 만들어야 합니다. 인덱스가 파티션되지 않은 경우, 테이블을 온라인으로 만드는 것은 상당한 양의 로그 스페이스를 사용하는 시간 소비 조작일 수 있습니다. SET INTEGRITY는 새로 첨부되는 파티션의 데이터를 파티션되지 않은 인덱스에 삽입해야 하기 때문입니다.

파티션을 접속 해제한 후에는 SET INTEGRITY를 실행할 필요가 없습니다.

파티션된 테이블의 파티션된 인덱스

*파티션된 인덱스*는 각각 단일 데이터 파티션에 대한 인덱스 항목을 포함하는 *인덱스 파티션*의 세트로 구성됩니다. 각 인덱스 파티션에는 대응하는 데이터 파티션의 데이터에 대한 참조만 들어있습니다. 시스템 및 사용자 생성 인덱스가 둘 다 파티션될 수 있습니다.

파티션된 인덱스는 특히 ALTER TABLE문의 ATTACH PARTITION절을 사용하여 파티션된 테이블의 데이터를 롤인 또는 롤아웃 중일 때 유용합니다. 파티션되지 않은 인덱스를 사용할 때는 새로 추가되는 파티션의 데이터가 온라인이 되기 전에 SET INTEGRITY문을 발행해야 합니다. 이 작업은 시간이 걸릴 수 있으며 많은 양의 로그 스페이스가 필요할 수 있습니다. 파티션된 인덱스를 사용하는 테이블 파티션을 첨부하면 이 오버헤드가 제거됩니다.

파티션할 수 없는 몇 가지 유형의 인덱스가 있습니다.

- XML 데이터에 대한 인덱스
- 공간 데이터에 대한 인덱스
- MDC 블록 인덱스(시스템 생성)
- XML 컬럼 경로 인덱스(시스템 생성)

이러한 경우 인덱스는 반드시 파티션되지 않은 것으로 작성되어야 합니다. 또한 파티션된 고유 인덱스에 대한 인덱스 키는 사용자 생성 키 또는 시스템 생성 키 모두 테이블 파티션 키의 모든 컬럼을 포함해야 합니다. 후자는 데이터에 대한 고유성 또는 기본 제한조건을 강제하기 위해 시스템이 작성하는 인덱스의 경우입니다.

356 페이지의 그림 42는 파티션된 인덱스의 예를 나타냅니다.

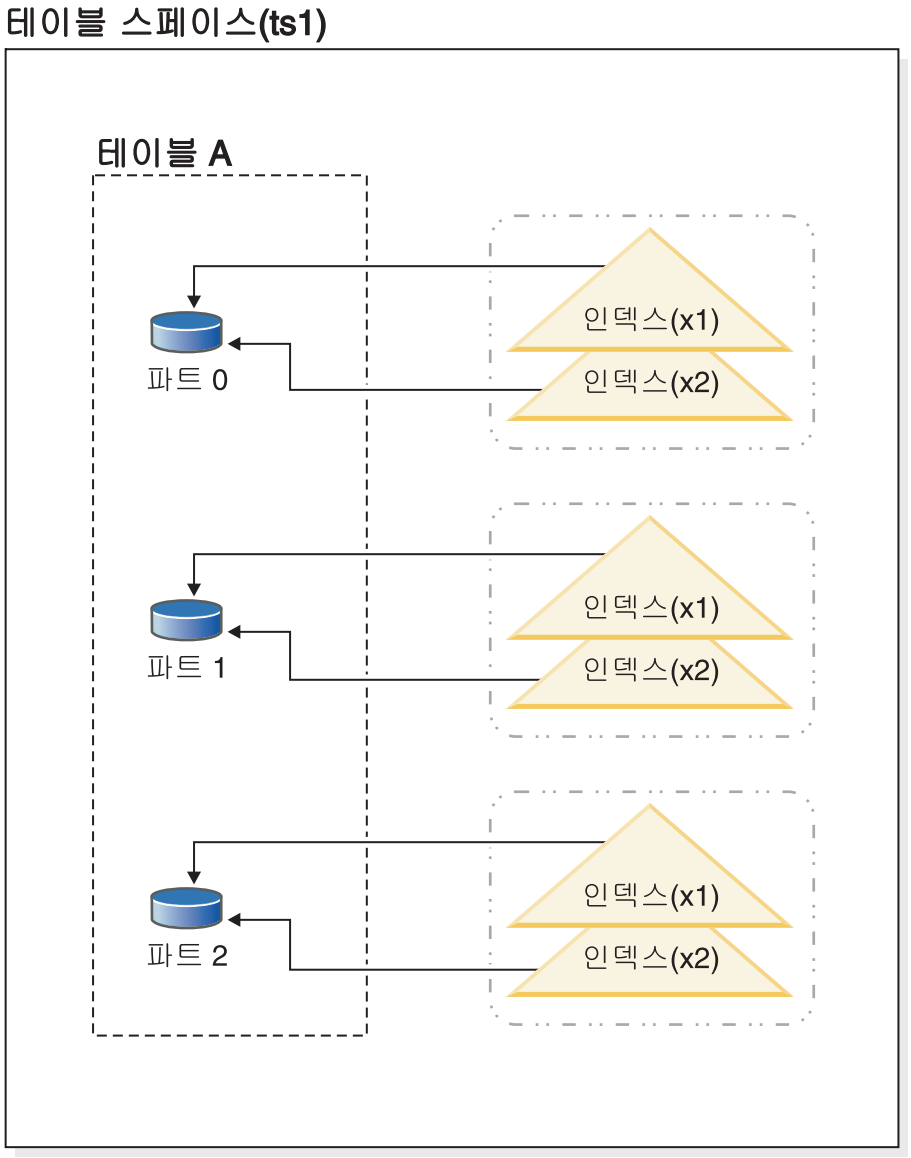


그림 42. 테이블의 데이터 파티션과 테이블 스페이스를 공유하는 파티션된 인덱스

이 예에서, 테이블 A의 모든 데이터 파티션 및 테이블 A에 대한 모든 인덱스 파티션은 단일 테이블 스페이스에 있습니다. 인덱스 파티션은 해당 파티션이 연관되는 데이터 파티션에 있는 행만을 참조합니다. (인덱스가 모든 데이터 파티션 사이의 모든 행을 참조하는 파티션되지 않은 인덱스와 파티션된 인덱스를 비교하십시오). 또한 주어진 데이터 파티션에 대한 인덱스 파티션은 동일한 인덱스 오브젝트에 있습니다. 인덱스 및 인덱스 파티션의 이 특별한 배열은 다음과 같은 명령문을 사용하여 설정되었습니다.

```
CREATE TABLE A (columns) in ts1
      PARTITION BY RANGE (column expression)
      (PARTITION PART0 STARTING FROM constant ENDING constant,
       PARTITION PART1 STARTING FROM constant ENDING constant,
       PARTITION PART2 STARTING FROM constant ENDING constant,

   CREATE INDEX x1 ON A (...) PARTITIONED;
   CREATE INDEX x2 ON A (...) PARTITIONED;
```

그림 43은 파티션된 인덱스의 다른 예를 나타냅니다.

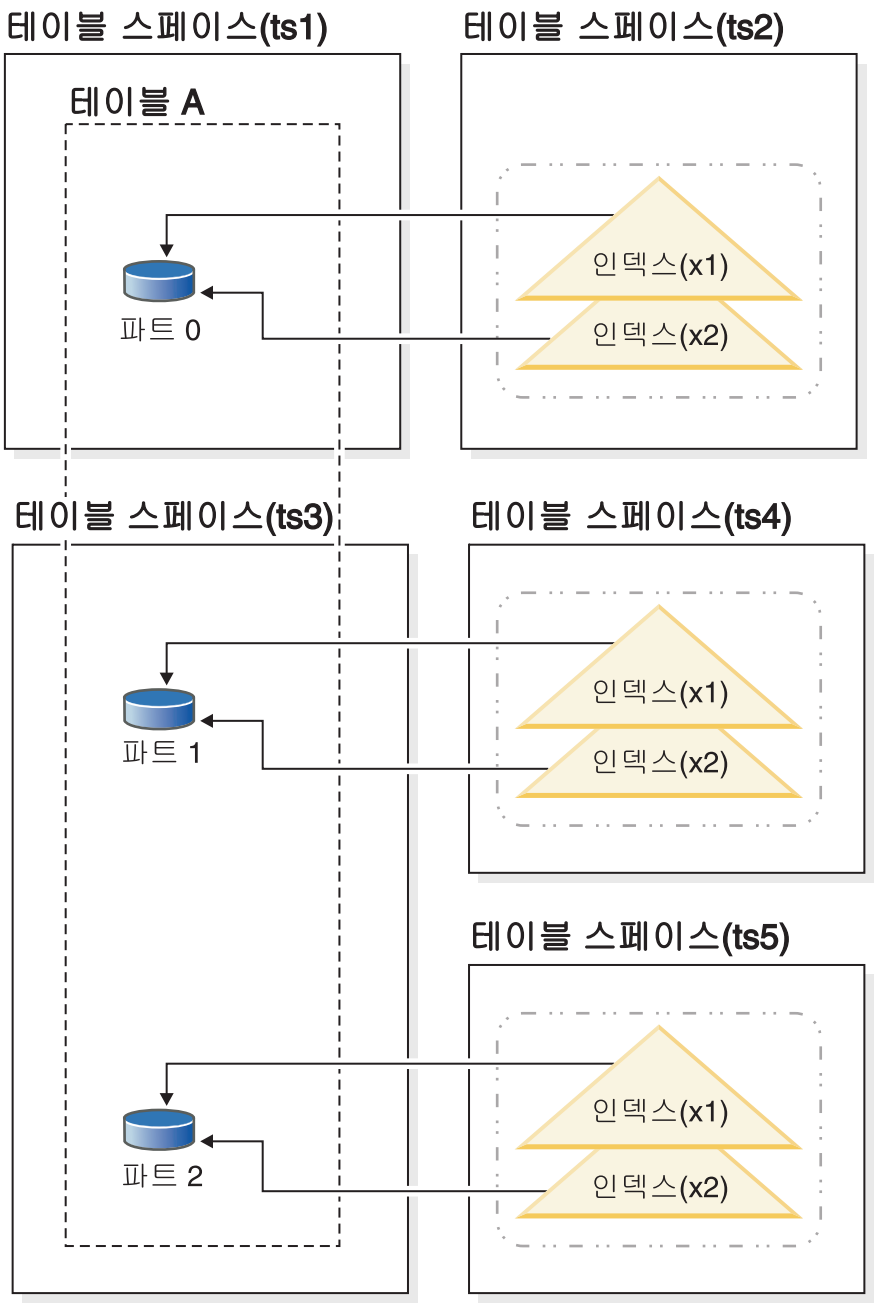


그림 43. 서로 다른 테이블 스페이스에 데이터 파티션과 인덱스 파티션을 갖는 파티션된 인덱스

이 예에서, 테이블 A에 대한 데이터 파티션은 두 개의 테이블 스페이스 TS1 및 TS3에 분산되어 있습니다. 인덱스 파티션도 서로 다른 테이블 스페이스에 있습니다. 인덱스 파티션은 해당 파티션이 연관되는 데이터 파티션에 있는 행만을 참조합니다. 인덱스 및 인덱스 파티션의 이 특별한 배열은 다음과 같은 명령문을 사용하여 설정되었습니다.

```
CREATE TABLE A (columns)
      PARTITION BY RANGE (column expression)
      (PARTITION PART0 STARTING FROM constant ENDING constant IN ts1 INDEX IN ts2,
```

```
    PARTITION PART1 STARTING FROM constant ENDING constant IN ts3 INDEX IN ts4,
    PARTITION PART2 STARTING FROM constant ENDING constant IN ts3,INDEX IN ts5)

CREATE INDEX x1 ON A (...);
CREATE INDEX x2 ON A (...);
```

이 경우에 PARTITIONED절이 CREATE INDEX문에서 생략되었음을 주의하십시오. 인덱스는 여전히 파티션된 인덱스로서 작성되며, 이것이 파티션된 테이블에 대한 디폴트이기 때문입니다.

그림 44는 파티션되지 않은 인덱스와 파티션된 인덱스를 둘 다 갖는 파티션된 테이블의 예를 나타냅니다.

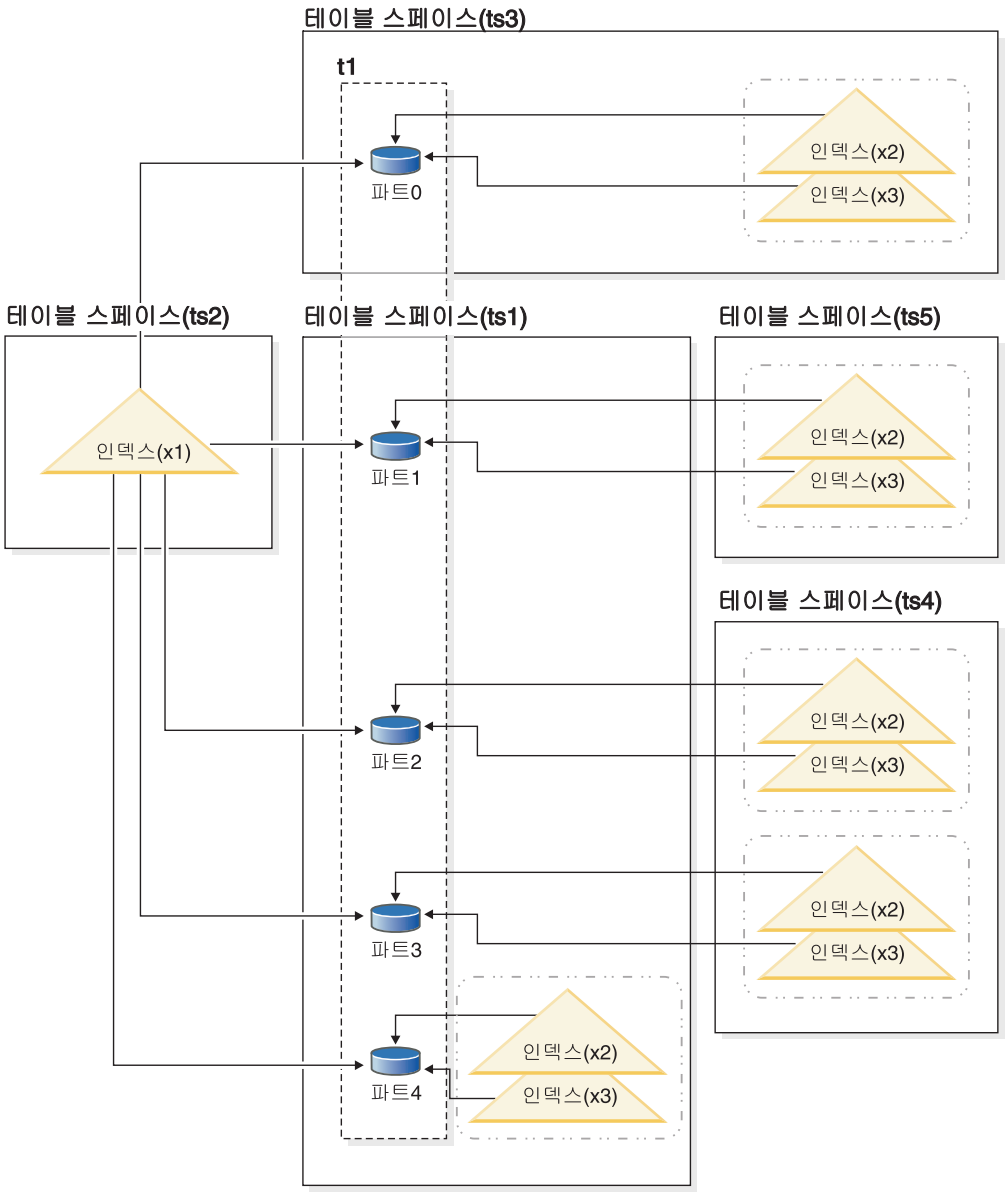


그림 44. 파티션된 테이블에 대한 파티션되지 않은 인덱스 및 파티션된 인덱스의 조합.

이 다이어그램에서 인덱스 X1은 테이블 T1의 모든 파티션을 참조하는 파티션되지 않은 인덱스입니다. 인덱스 X2 및 X3은 다양한 테이블 스페이스에 상주하는 파티션된 인덱스입니다. 인덱스 및 인덱스 파티션의 이 특별한 배열은 다음과 같은 명령문을 사용하여 설정되었습니다.

```
CREATE TABLE t1 (columns) in ts1 INDEX IN ts2 1
   PARTITION BY RANGE (column expression)
   (PARTITION PART0 STARTING FROM constant ENDING constant  IN ts3, 2
    PARTITION PART1 STARTING FROM constant ENDING constant  INDEX IN ts5,
    PARTITION PART2 STARTING FROM constant ENDING constant  INDEX IN ts4,
    PARTITION PART3 STARTING FROM constant ENDING constant  INDEX IN ts4,
    PARTITION PART4 STARTING FROM constant ENDING constant)

CREATE INDEX x1 ON t1 (...) NONPARTITIONED;
CREATE INDEX x2 ON t1 (...) PARTITIONED;
CREATE INDEX x3 ON t1 (...) PARTITIONED;
```

다음을 주의하십시오.

- 파티션되지 않은 인덱스 X1은 테이블 스페이스 ts2에 저장되는데, 이것이 테이블 T1에 대한 파티션되지 않은 인덱스에 대해 지정된 디폴트이기 때문입니다(1 참조).
- 데이터 파티션 0(Part0)에 대한 인덱스 파티션은 테이블 스페이스 TS3에 저장되는데, 인덱스 파티션에 대한 디폴트 위치가 해당 파티션이 참조하는 데이터 파티션과 동일하기 때문입니다(2 참조).
- Part4는 TS1에 저장되는데, 이것이 테이블 T1의 데이터 파티션에 대한 디폴트 테이블 스페이스입니다(1 참조). 이 데이터 파티션에 대한 인덱스 파티션도 TS1에 상주하며, 역시 인덱스 파티션에 대한 디폴트 위치가 해당 파티션이 참조하는 데이터 파티션과 동일하기 때문입니다.

중요사항: 파티션되지 않은 인덱스와는 달리, 파티션된 인덱스를 사용하는 경우 CREATE INDEX문의 INDEX IN절을 사용하여 인덱스 파티션을 저장할 테이블 스페이스를 지정할 수 없습니다. 인덱스 파티션에 대한 디폴트 스토리지 위치를 겹쳐쓰는 유일한 방법은 테이블을 작성할 때 CREATE TABLE문의 *파티션 레벨* INDEX IN절을 사용하여 위치를 지정하는 것입니다. 테이블 레벨 INDEX IN절은 인덱스 파티션 배치에 아무 효과도 없습니다.

CREATE INDEX문에 PARTITIONED 옵션을 포함시켜서 파티션된 테이블에 대한 파티션된 인덱스를 작성합니다. 예를 들어 sales_date를 테이블 파티션 키로 갖고 파티션된 SALES라는 테이블의 경우, 파티션된 인덱스를 작성하기 위해 다음과 같은 명령문을 사용할 수 있습니다.

```
CREATE INDEX partIDbydate on SALES (sales_date, partID) PARTITIONED
```

파티션된 고유 인덱스를 작성하려는 경우 테이블 파티션 컬럼이 인덱스 키 컬럼에 포함되어야 합니다. 그러므로 앞의 예에서 다음 명령문을 사용하여 파티션된 인덱스를 작성하려고 시도한 경우,

```
CREATE UNIQUE INDEX uPartID on SALES (partID) PARTITIONED
```

테이블 파티션 키를 형성하는 sales_date 컬럼이 인덱스 키에 포함되지 않기 때문에 해당 명령문은 실패합니다.

파티션된 테이블에 대한 인덱스를 작성할 때 PARTITIONED 키워드를 생략하면 데이터베이스 관리 프로그램은 다음 경우를 제외하고 파티션된 인덱스를 디폴트로 작성합니다.

- 고유 인덱스를 작성 중이고, 인덱스 키에 모든 테이블 파티션 키가 포함되지 않음
- 이 주제의 시작에서 파티션된 인덱스로서 작성될 수 없는 것으로 설명한 인덱스 유형 중 하나를 작성 중인 경우

이러한 경우에는 인덱스가 파티션되지 않은 인덱스로 작성됩니다.

기존 파티션되지 않은 인덱스의 정의와 일치하는 정의를 사용하여 파티션되지 않은 인덱스를 작성하면 SQL0605W 오류가 발생하지만, 파티션된 인덱스는 비슷한 정의를 갖는 파티션되지 않은 인덱스와 공존할 수 있습니다. 이것은 파티션된 인덱스를 더 쉽게 채택할 수 있도록 하기 위해서입니다.

인덱스 디자인

인덱스는 일반적으로 테이블에 대한 액세스 속도를 높이기 위해 사용됩니다. 그러나 논리 데이터 설계 목적으로도 사용될 수 있습니다.

예를 들어, 고유 인덱스는 컬럼에 값이 중복된 항목이 오도록 허용하지 않기 때문에, 테이블의 어떠한 행도 같지 않다고 보증합니다. 컬럼의 값을 오름차순 또는 내림차순으로 정렬하기 위해 인덱스를 작성할 수도 있습니다.

주: 인덱스를 작성할 때, 인덱스가 읽기 성능을 향상시킬 수 있지만 쓰기 성능에는 부정적인 영향을 주는 것을 기억하십시오. 이것은 데이터베이스 관리 프로그램이 테이블에 쓰는 모든 행의 경우 영향을 받는 모든 인덱스도 갱신해야 하기 때문입니다. 그러므로 명백한 전체 성능 이점이 있을 때만 인덱스를 작성해야 합니다.

인덱스를 작성할 때 테이블의 구조 및 테이블에 대해 가장 자주 수행되는 쿼리의 유형도 고려해야 합니다. 예를 들어 자주 발행되는 쿼리의 WHERE절에 나타나는 컬럼은 인덱스를 위한 좋은 후보입니다. 그러나 덜 자주 실행되는 쿼리에서는 인덱스가 INSERT 및 UPDATE문의 성능에 부과하는 비용이 이점을 상쇄할 수 있습니다.

비슷하게, 빈번한 쿼리의 GROUP BY절에 나타나는 컬럼은 특히 행을 그룹화하는 데 사용되는 값의 수가 그룹화되는 행 수에 비해 상대적으로 작은 경우 인덱스 작성이 이익일 수 있습니다.

인덱스를 작성할 때 인덱스를 압축할 수도 있음을 기억하십시오. ALTER INDEX문을 사용하여 압축을 사용하거나 사용 안하여 나중에 인덱스를 수정할 수 있습니다.

인덱스를 제거하거나 삭제하기 위해 DROP INDEX 명령을 사용할 수 있습니다. 인덱스 삭제는 인덱스 삽입과는 반대의 요구사항을 갖습니다. 즉, 인덱스 항목을 제거(또는 삭제됨으로 표시)해야 합니다.

인덱스 디자인 시 지침 및 고려사항

- 인덱스 키를 구성하는 컬럼 순서는 인덱스 키 작성에 영향을 주지 않지만, 옵티마이저가 인덱스의 사용 여부를 결정하는 경우 옵티마이저에 영향을 줄 수 있습니다. 예를 들어 쿼리에 `ORDER BY col1,col2`절이 있는 경우, `(col1,col2)`에 작성된 인덱스를 사용할 수 있지만 `(col2,col1)`에 작성된 인덱스는 도움이 되지 않습니다. 비슷하게, 쿼리가 `where col1 >= 50 and col1 <= 100` 또는 `where col1=74` 같은 조건을 지정한 경우, `(col1)` 또는 `(col1,col2)`에 대한 인덱스는 도움이 되지만 `(col2,col1)`에 대한 인덱스는 별로 도움이 되지 않습니다.

  주: 가능할 때는 언제나 인덱스 키에 있는 컬럼은 가장 특정적인 것부터 가장 덜 특정적인 것으로 정렬하십시오. 이것이 최고의 성능을 제공합니다.
- 특정 테이블에 대해 최대 32,767까지 임의 숫자의 인덱스를 정의할 수 있으며, 쿼리 성능에 긍정적인 효과를 줄 수 있습니다. 인덱스 관리 프로그램이 갱신, 삭제 및 삽입 조작 동안 인덱스를 유지보수해야 합니다. 많은 내용이 갱신되는 테이블에 대해 대규모의 인덱스를 작성하면 요청 처리 시간이 더 길어질 수 있습니다. 마찬가지로, 대형 인덱스 키도 요청 처리 속도를 느리게 할 수 있습니다. 그러므로 잦은 액세스로 인해 분명한 이점이 있는 경우에만 인덱스를 사용하십시오.
- 고유 인덱스 키의 파트는 아니지만 인덱스에 저장 또는 유지보수되는 컬럼 데이터를 Include 컬럼이라고 합니다. Include 컬럼은 고유 인덱스 전용으로 지정될 수 있습니다. Include 컬럼으로 인덱스 작성시, 고유 키 컬럼만이 고유성을 위해 저장 및 고려됩니다. Include 컬럼을 사용하면 데이터 검색을 위한 인덱스 전용 액세스가 가능하므로 성능이 향상됩니다.
- 인덱스화된 테이블이 비어 있을 경우에도 인덱스가 계속 작성되지만, 테이블이 로드되거나 행이 삽입될 때까지 어떠한 인덱스 항목도 작성되지 않습니다. 테이블이 비어 있지 않으면 데이터베이스 관리 프로그램은 CREATE INDEX문을 처리하는 동안에 인덱스 항목을 작성합니다.
- *클러스터링 인덱스*의 경우, 데이터베이스 관리 프로그램이 테이블에 대한 새 행을 비슷한 키 값(인덱스에 의해 정의되는)을 갖는 기존 행에 물리적으로 가깝게 배치하려고 합니다.
- *기본 키 인덱스*가 클러스터링 인덱스이기 원하는 경우 CREATE TABLE문에서 기본 키를 지정하지 않아야 합니다. 일단 기본 키가 작성되면, 연관된 인덱스는 수정될 수 없습니다. 대신, 기본 키 절없이 CREATE TABLE을 발행하십시오. 그런 다음, CREATE INDEX문을 발행하여 클러스터링 속성을 지정하십시오. 마지막으로, ALTER TABLE문을 사용하여 방금 작성한 인덱스에 해당하는 기본 키를 추가하십시오. 이 인덱스는 기본 키 인덱스로서 사용됩니다.

- *파티션된 테이블*이 있는 경우, 다음 경우가 아니면 디폴트로 사용자가 작성하는 모든 인덱스는 *파티션된 인덱스*가 됩니다.
  - XML 데이터에 대한 인덱스를 작성 중입니다.
  - 파티션 키를 포함하지 않는 고유 인덱스를 작성 중입니다.

  또한 인덱스가 *파티션되지 않은 인덱스*로서 작성되도록 선택할 수도 있습니다.

  파티션된 인덱스는 파티션된 테이블과의 롤인 조작을 수행할 때(즉, ALTER TABLE 문에서 ATTACH PARTITION절을 사용하여 데이터 파티션을 다른 테이블에 첨부할 때) 유용합니다. 파티션된 인덱스를 사용하면 그렇지 않은 경우 파티션되지 않은 인덱스에 대해 수행해야 하는 인덱스 유지보수를 피할 수 있습니다. 파티션된 테이블이 파티션되지 않은 인덱스를 사용할 때 SET INTEGRITY문을 사용하여 새로 조합되는 데이터 파티션에 대해 인덱스 유지보수를 수행해야 합니다. 이것이 시간이 걸릴 뿐 아니라, 롤인되는 행 수에 따라서는 많은 양의 로그 스페이스가 필요할 수도 있습니다.
- 인덱스는 디스크 스페이스를 소비합니다. 디스크 스페이스의 양은 키 컬럼의 길이 및 인덱스되는 행 수에 따라 다릅니다. 인덱스 *크기*는 테이블에 삽입되는 데이터가 많을수록 늘어납니다. 그러므로 데이터베이스의 *크기*를 계획할 때 인덱스되는 데이터의 양을 고려하십시오. 몇 가지 인덱싱 *크기* 지정 고려사항은 다음과 같습니다.
  - 기본 및 고유 키 제한조건은 항상 시스템 생성 고유 인덱스를 작성합니다.
  - MDC 테이블의 작성은 항상 시스템 생성 블록 인덱스를 작성합니다.
  - XML 컬럼은 항상 시스템 생성 인덱스가 작성되게 합니다.
  - 대개 외부 키 제한조건 컬럼에 대한 인덱스를 작성하는 것이 좋습니다.
  - 인덱스가(COMPRESS 옵션을 사용하여) 압축되는지 여부

  주: 인덱스에서 컬럼의 최대 수는 64입니다. 그러나 유형이 지정된 테이블을 인덱스 중이라면, 인덱스의 최대 컬럼 수는 63입니다. 인덱스 키의 최대 길이는 모든 오버헤드를 포함하여 *IndexPageSize* ÷ 4입니다. 테이블에서 허용되는 최대 인덱스는 32,767입니다. 인덱스 키의 최대 길이는 페이지 *크기*에 대한 인덱스 키 길이의 한계보다 커서는 안됩니다. 컬럼에 저장된 길이에 대해서는 『CREATE TABLE문』을 참조하십시오. 키 길이 한계에 대해서는 『SQL 및 XQuery 한계』 주제를 참조하십시오.

  주:

- 데이터베이스 업그레이드 중에 기존 인덱스는 압축되지 않습니다. 테이블에서 데이터 행 압축이 가능한 경우, CREATE INDEX문에서 COMPRESS NO 옵션이 지정되지 않으면 업그레이드 이후에 작성되는 새 인덱스는 압축될 수 있습니다.

인덱스 디자인 도구

테이블을 작성한 후에 데이터베이스 관리 프로그램이 얼마나 빨리 테이블에서 데이터를 검색할 수 있을지를 고려해야 합니다. 디자인 어드바이저 또는 db2advis 명령을 사용하여 인덱스를 디자인할 수 있습니다.

중요사항: 제어 센터의 디자인 어드바이저 GUI는 버전 9.7에서 사용되지 않으며 이후 릴리스에서 제거될 수 있습니다. 자세한 정보는 *버전 9.7의 새로운 내용* 책에 있는 『제어 센터 도구 및 DB2 Administration Server(DAS)는 사용되지 않음』 주제를 참조하십시오.

테이블에 대한 유용한 인덱스를 작성하면 쿼리 성능을 크게 향상시킬 수 있습니다. 책의 색인과 같이, 테이블에 대한 인덱스는 최소한의 검색으로 특정 정보를 빨리 찾을 수 있습니다. 인덱스를 사용하여 테이블에서 특정 행을 검색하면 데이터베이스 관리 프로그램이 수행해야 하는 방대한 입출력 조작 수를 줄일 수 있습니다. 이것은 데이터베이스 관리 프로그램이 인덱스를 사용하여 모든 일치가 발견될 때까지 모든 데이터 페이지의 소모적인 검색을 수행하는 대신 상대적으로 적은 수의 데이터 페이지를 읽어서 행을 찾을 수 있기 때문입니다.

DB2 디자인 어드바이저는 워크로드 성능을 상당히 향상시킬 수 있는 도구입니다. 복잡한 워크로드를 위해 작성할 인덱스, MQT, 클러스터링 차원 또는 데이터베이스 파티션을 선택하는 일은 상당히 어려울 수 있습니다. 디자인 어드바이저가 워크로드의 성능을 향상시키는데 필요한 모든 오브젝트를 식별해줍니다. 워크로드의 SQL문 세트에 대해 디자인 어드바이저는 다음에 대한 권장사항을 생성합니다.

- 새 인덱스
- 새로 구체화된 쿼리 테이블(MQT)
- 다차원적으로 클러스터된(MDC) 테이블
- 테이블의 재분산
- 지정한 워크로드에서 사용하지 않는 인덱스 및 MQT 삭제(GUI 도구를 통해)

디자인 어드바이저에서 이런 권장사항 중 일부나 전부를 즉시 구현하도록 하거나 나중을 위해 스케줄할 수 있습니다.

디자인 어드바이저에서는 디자인 어드바이저 GUI나 명령행 도구를 사용하여 다음 태스크를 간단하게 할 수 있습니다.

- 새 데이터베이스를 설정하기 위한 계획
- 워크로드 성능 조정

인덱스에 대한 스페이스 요구사항

인덱스를 설계할 때 스페이스 요구사항을 알아야 합니다. 압축 인덱스의 경우 이 주제에 있는 공식에서 유도하는 추정값을 상한으로 사용할 수 있지만 실제로는 훨씬 더 작을 것입니다.

압축되지 않은 인덱스에 대한 스페이스 요구사항

각 압축되지 않은 인덱스에 필요한 스페이스는 다음과 같이 계산할 수 있습니다.

(*평균 인덱스 키 크기* + *인덱스 키 오버헤드*) × *행 수* × 2

각 항목에 대한 설명은 다음과 같습니다.

- *평균 인덱스 키 크기*는 인덱스 키에 있는 각 컬럼의 바이트 수입니다. VARCHAR 및 VARGRAPHIC 컬럼의 평균 컬럼 *크기*를 계산하려면, 현재 데이터 *크기*의 평균에 2바이트를 더하여 사용하십시오.
- *인덱스 키 오버헤드*는 인덱스가 작성되는 테이블의 유형에 따라 다릅니다.

표 21. 여러 가지 테이블의 인덱스 키 오버헤드

테이블 스페이스의 유형	테이블 유형	인덱스 유형	인덱스 키 오버헤드
모두	모두	XML 경로 또는 영역	11바이트
일반	파티션되지 않음	모두	9바이트
	파티션됨	파티션됨	9
		파티션되지 않음	11
대형	파티션됨	파티션됨	11
		파티션되지 않음	13

- *행 수*는 테이블의 행 수 또는 주어진 데이터 파티션에 있는 행 수입니다. 이 계산에서 전체 테이블에 있는 행 수를 사용하면 인덱스(파티션되지 않은 인덱스) 또는 결합된 모든 인덱스 파티션(파티션된 인덱스)에 대한 *크기*를 추정할 수 있습니다. 데이터 파티션의 행 수를 사용하면 인덱스 파티션에 대한 *크기*를 추정할 수 있습니다.
- 인수 『2』는 비리프 페이지 및 여유 공간과 같은 오버헤드 값입니다.

주:

1. 널(null) 값을 허용하는 모든 컬럼의 경우 널(NULL) 표시용으로 1바이트를 추가하십시오.
2. 다차원적으로 클러스터된(MDC) 테이블에 내부적으로 작성되는 블록 인덱스의 경우 『행 수』는 『블록 수』로 대체됩니다.

XML 인덱스의 스페이스 요구사항

XML 컬럼의 각 인덱스의 경우 필요한 스페이스는 다음과 같이 계산할 수 있습니다.

(*평균 인덱스 키* + *인덱스 키 오버헤드*) × *인덱스된 노드 수* × 2

각 항목에 대한 설명은 다음과 같습니다.

- *평균 인덱스 키*는 인덱스를 구성하는 키 파트의 합입니다. XML 인덱스는 여러 가지 XML 키 파트에 값(sql-data-type)을 더하여 구성됩니다.

  14 + *가변 오버헤드* + *sql-data-type의 바이트 수*

  각 항목에 대한 설명은 다음과 같습니다.

  – 14는 고정 오버헤드의 바이트 수입니다.

  – *가변 오버헤드*는 인덱스된 노드의 평균 용량에 4바이트를 더한 값입니다.

  – *sql-data-type의 바이트 수*는 SQL과 동일한 규칙을 따릅니다.

- *인덱스된 노드 수*는 삽입되는 문서 수에 인덱스 정의에서 XML 패턴 표현식(XMLPATTERN)을 충족하는 샘플 문서의 노드 수를 곱한 값입니다. *인덱스된 노드 수*는 파티션 또는 전체 테이블에 있는 노드 수일 수 있습니다.

인덱스 작성을 위한 임시 스페이스 요구사항

인덱스 작성시 임시 스페이스가 필요합니다. 인덱스 작성 중에 필요한 최대 임시 스페이스량은 다음과 같이 계산할 수 있습니다.

(*평균 인덱스 키 크기* + *인덱스 키 오버헤드*) × *행 수* × 3.2

공간 인덱스, XML 컬럼의 인덱스 및 내부 XML 영역 인덱스 같이 행당 둘 이상의 인덱스 키가 존재할 수 있는 인덱스의 경우 필요한 임시 스페이스는 다음과 같이 계산할 수 있습니다.

(*평균 인덱스 키 크기* + *인덱스 키 오버헤드*) × *인덱스된 노드 수* × 3.2

여기서 인수『3.2』는 인덱스 오버헤드 값이며 인덱스 작성시 정렬에 필요한 공간입니다. *행 수* 또는 *인덱스된 노드 수*는 전체 테이블 또는 주어진 데이터 파티션의 숫자입니다.

주: 비고유 인덱스의 경우 주어진 중복 키 항목의 하나의 사본만이 주어진 리프 노드에 저장됩니다. LARGE 테이블 스페이스의 테이블에 대한 인덱스의 경우 중복 키의 크기는 파티션되지 않은 인덱스의 경우 9, 파티션된 인덱스 및 파티션되지 않은 테이블에 대한 인덱스의 경우 7입니다. REGULAR 테이블 스페이스의 테이블에 대한 인덱스의 경우 이들 값은 파티션되지 않은 인덱스의 경우 7, 파티션된 인덱스 및 파티션되지 않은 테이블의 인덱스의 경우 5입니다. 이러한 규칙에 대한 유일한 예외는 중복 키의 크기가 항상 7인 XML 경로 및 XML 영역 인덱스입니다. 위에 표시된 추정값은 중복값을 가정하지 않습니다. 인덱스 저장에 필요한 스페이스는 위에 표시된 수식을 사용하여 실제보다 높게 평가될 수 있습니다.

임시 스페이스는 인덱스 노드 수가 64KB의 데이터를 초과하는 경우 삽입할 때 필요합니다. 임시 스페이스 양은 다음과 같이 계산할 수 있습니다.

평균 인덱스 키 크기 × *인덱스된 노드 수* × 1.2

리프 페이지당 키 수 계산

다음 두 공식을 사용하여 인덱스 리프 페이지별 키의 수를 계산할 수 있습니다. 두 번째 공식이 더 정확히 계산됩니다. 이 계산의 정확도는 평균이 실제 데이터를 얼마나 반영하는가에 따라 크게 달라집니다.

주: SMS 테이블 스페이스의 경우 리프 페이지의 최소 필수 스페이스는 페이지 크기의 세 배입니다. DMS 테이블 스페이스의 최소 스페이스는 Extent입니다.

1. 리프 페이지당 평균 키 수는 대략 다음과 같습니다.

   $$((.9 * (U - (M \times 2))) \times (D + 1)) \div (K + 7 + (Ds \times D))$$

   각 항목에 대한 설명은 다음과 같습니다.

   - *U*는 페이지의 사용 가능한 스페이스로서 대략 페이지 크기에서 100을 뺀 것과 같습니다. 예를 들어 페이지 크기가 4096이면 U는 3996입니다.
   - $M = U \div (9 + minimumKeySize)$
   - $Ds = duplicateKeySize$ (『인덱스 작성에 대한 임시 스페이스 요구사항』 메모를 참조하십시오.)
   - *D* = 키 값당 평균 중복 수
   - $K = averageKeySize$

   minimumKeySize 및 *averageKeysize*는 각 널(NULL) 입력 가능 키 부분을 위한 여분의 바이트 및 각 가변 길이 키 부분의 길이를 위한 별도의 2바이트를 포함해야 합니다.

   include 컬럼이 있는 경우, 이 컬럼은 *minimumKeySize* 및 *averageKeySize*로 나타내야 합니다.

   *최소 키 크기*는 인덱스를 구성하는 키 파트의 합입니다.

   고정 오버헤드 + 가변 오버헤드 + sql-data-type의 바이트 수

   각 항목에 대한 설명은 다음과 같습니다.

   - *고정 오버헤드*는 13바이트입니다.
   - *가변 오버헤드*는 인덱스된 노드의 최소 용량에 4바이트를 더한 값입니다.
   - *sql-data-type의 바이트 수* 값은 SQL과 동일한 규칙을 따릅니다.

   인덱스 작성시 디폴트값 10퍼센트 대신 다른 여유 공간 비율 값으로 지정할 경우, .9는 (100 - pctfree)/100 값으로 대체될 수 있습니다.

2. 리프 페이지당 평균 키 수에 대한 보다 정확한 계산은 다음과 같습니다.

   리프 페이지 수 = x / (리프 페이지의 평균 키 수)

   여기서 *x*는 테이블 또는 파티션의 총 행 수입니다.

XML 컬럼에 대한 인덱스의 경우 *x*는 컬럼에 있는 인덱스된 노드의 총 수입니다.

인덱스의 원래 *크기*는 다음과 같이 계산할 수 있습니다.

$(L + 2L/(\text{리프 페이지의 평균 키 수})) \times \text{페이지 크기}$

DMS 테이블 스페이스의 경우 테이블에 있는 모든 인덱스의 *크기*를 더하고 인덱스가 있는 테이블 스페이스의 Extent *크기*의 배수로 반올림하십시오.

INSERT/UPDATE 활동으로 인한 인덱스 확장에 대해 추가 스페이스를 제공해야 하는데, 그 결과로 페이지가 분할될 수 있습니다.

인덱스 레벨의 예상 수뿐만이 아니라 인덱스의 원래 *크기*를 보다 정확하게 계산하려면, 다음 계산식을 사용하십시오. (include 컬럼이 인덱스 정의에 사용되는 경우 이 수치는 특히 중요할 수 있습니다.) 비리프 페이지의 평균 키 수는 대략 다음과 같습니다.

$((.9 \times (U - (M \times 2))) \times (D + 1)) \div (K + 13 + (9 * D))$

각 항목에 대한 설명은 다음과 같습니다.

- *U*는 페이지의 사용 가능한 스페이스로서 대략 페이지 *크기*에서 100을 뺀 것과 같습니다. 페이지 *크기*가 4096인 경우, U는 3996입니다.
- *D*는 비리프 페이지의 키 값당 평균 중복 수입니다(이 값은 리프 페이지에 대한 값보다 훨씬 적으며 0으로 설정하면 계산이 간단해짐).
- $M = U \div (9 +$ *minimumKeySize* (비리프 페이지의 경우))
- $K =$ *averageKeySize* (비리프 페이지의 경우)

include 컬럼이 있는 경우를 제외하고는 비리프 페이지의 *minimumKeySize*와 *averageKeySize*는 리프 페이지의 경우와 동일합니다. include 컬럼은 비리프 페이지에 저장되지 않습니다.

인덱스 작성 시 최대 10 퍼센트의 여유 공간이 비리프 페이지에 남게 되므로, 해당 값이 .9보다 클 경우 .9를 (100 - *pctfree*)÷100으로 대체해서는 안됩니다.

비리프 페이지 수는 다음과 같이 계산할 수 있습니다.

```
if L > 1 then {P++; Z++}
While (Y > 1)
{
   P = P + Y
   Y = Y / N
  Z++
}
```

각 항목에 대한 설명은 다음과 같습니다.

- *P*는 페이지 수(초기값 0)입니다.
- *L*은 리프 페이지 수입니다.

- *N*은 각 비리프 페이지의 키 수입니다.
- $Y = L \div N$
- *Z*는 인덱스 트리에 있는 레벨 수입니다(초기값 1).

주: 위의 계산은 하나의 파티션되지 않은 인덱스나 파티션된 인덱스의 경우 단일 인덱스 파티션에 적용됩니다.

총 페이지 수는 다음과 같습니다.

$$T = (L + P + 2) \times 1.0002$$

추가 0.02%(1.0002)는 스페이스 맵 페이지를 포함한 오버헤드용입니다.

인덱스 작성에 필요한 스페이스량은 다음과 같이 계산됩니다.

$$T \times \text{페이지 크기}$$

인덱스 압축

인덱스(선언 또는 작성된 임시 테이블의 인덱스 포함)는 스토리지 비용을 줄이기 위해 압축할 수 있습니다. 이것은 특히 대형 OLTP 및 데이터 웨어하우스 환경에 유용합니다.

디폴트로 인덱스 압축은 압축 테이블의 경우 사용 가능하고 압축되지 않은 테이블은 사용 불가능합니다. CREATE INDEX문의 COMPRESS YES 옵션을 사용하여 이 디폴트 동작을 겹쳐쓸 수 있습니다. 인덱스가 작성된 후에는 ALTER INDEX문을 사용하여 인덱스 압축을 사용 또는 사용 불가능하게 하십시오. 그런 다음 INDEX REORG를 수행하여 인덱스를 재빌드해야 합니다.

제한사항: 인덱스 압축은 다음 유형의 인덱스에 대해서는 지원되지 않습니다.

- MDC 블록 인덱스
- XML 경로 인덱스

또한,

- 인덱스 스펙은 압축할 수 없습니다.
- 임시 테이블의 인덱스에 대한 압축 속성은 ALTER INDEX 명령으로 변경할 수 없습니다.

인덱스 압축이 사용 가능할 때, 인덱스 페이지의 온디스크 및 메모리 형식은 스토리지 스페이스를 최소화하도록 데이터베이스 관리 프로그램이 선택하는 압축 알고리즘을 기초로 수정됩니다. 달성되는 압축 정도는 작성 중인 인덱스 유형 및 인덱스가 포함하는 데이터에 따라 달라집니다. 예를 들어 데이터베이스 관리 프로그램은 중복 키에 대한 레코드 ID(RID)의 단축된 형식을 저장하여 많은 중복 키를 갖는 인덱스를 압축할 수

있습니다. 인덱스 키의 접두부에 높은 수준의 공통성이 있는 인덱스에서, 데이터베이스 관리 프로그램은 인덱스 키의 접두부에 있는 유사성을 기초로 압축을 적용할 수 있습니다.

압축과 연관된 한계 및 트래이드 오프가 있습니다. 인덱스가 공통 인덱스 컬럼 값이나 부분 공통 접두부를 공유하지 않는 경우, 스토리지 축소로 표현되는 인덱스 압축의 이점을 무시할 수 있습니다. 또한 시간소인 컬럼의 고유 인덱스가 동일한 리프 페이지의 년, 월, 일, 시, 분 또는 초에 대한 공통 값으로 인해 매우 높은 압축 성능을 갖는 경우에도, 공통 접두부가 존재하는지 확인하는 작업의 오버헤드로 인해 성능이 저하될 수 있습니다.

압축이 사용자의 특정한 상황에서 이점을 제공하지 않는 경우, 압축 없이 인덱스를 재작성하거나 인덱스를 변경한 후 인덱스 재구성을 수행하여 인덱스 압축을 사용 안할 수 있습니다.

인덱스 압축 사용을 고려할 경우 주의해야 할 몇 가지 사항이 있습니다.

- CREATE TABLE 또는 ALTER TABLE 명령에서 COMPRESS 옵션을 사용하여 행 압축을 사용 가능하게 하는 경우, CREATE INDEX 또는 ALTER INDEX 명령에 의해 명시적으로 사용 안되는 경우가 아니면 디폴트로 압축이 지원되고 해당 테이블에 대해 해당 지점 후에 작성되는 모든 인덱스에 대해 압축이 사용됩니다. 비슷하게 CREATE TABLE 또는 ALTER TABLE 명령으로 행 압축을 사용 안하는 경우, CREATE INDEX 또는 ALTER INDEX 명령에 의해 명시적으로 사용되지 않으면 해당 테이블에 대해 해당 지점 후에 작성되는 모든 인덱스에 대해 인덱스 압축이 사용되지 않습니다.
- ALTER INDEX 명령을 사용하여 인덱스 압축을 사용하는 경우, 압축은 인덱스 재구성이 수행될 때까지 발생하지 않습니다. 비슷하게, 압축을 사용하지 않으면 인덱스는 인덱스 재구성을 수행할 때까지 압축된 상태로 있습니다.
- 데이터베이스 이주 중에, 이주되었을 수 있는 모든 인덱스에 대해 압축이 사용되지 않습니다. 압축이 사용되기 원하는 경우 ALTER INDEX 명령을 사용한 후 인덱스 재구성을 수행해야 합니다.
- CPU 사용량이 인덱스 압축 또는 압축 해제에 필요한 처리의 결과로 약간 늘어날 수 있습니다. 허용할 수 없는 수준인 경우 신규 또는 기존 인덱스에 대해 인덱스 압축을 사용하지 않을 수 있습니다.

예

예 1: 인덱스가 압축되는지 여부 확인

뒤에 오는 두 명령문은 행 압축이 가능한 새 테이블 T1을 작성하고 T1에 인덱스 I1을 작성합니다.

```
CREATE TABLE T1 (C1 INT, C2 INT, C3 INT) COMPRESS YES
CREATE INDEX I1 ON T1(C1)
```

디폴트로 T1에 대한 인덱스가 압축됩니다. 압축이 사용 가능한지 여부를 표시하는 인덱스 T1에 대한 *압축 속성*은 카탈로그 테이블 또는 admin 테이블 함수를 사용하여 검사할 수 있습니다.

```
SELECT COMPRESSION FROM SYSCAT.INDEXES WHERE TABNAME='T1'

COMPRESSION
-----------
Y

  1 레코드가 선택되었습니다.
```

예 2: 압축된 인덱스가 재구성이 필요한지 여부 판별

압축된 인덱스가 재구성이 필요한지 확인하려면 REORGCHK 명령을 사용하십시오. 371 페이지의 표 22에서는 T1이라는 테이블에서 실행되는 명령을 나타냅니다.

표 22. REORGCHK 명령의 출력

```
REORGCHK ON TABLE SCHEMA1.T1

RUNSTATS 실행 중 ....

테이블 통계:

F1: 100 * OVERFLOW / CARD < 5
F2: 100 * (데이터 페이지의 실제 스페이스 사용) > 70
F3: 100 * (필수 페이지 수 / 총 페이지 수) > 80

SCHEMA.NAME                CARD     OV     NP     FP ACTBLK    TSIZE  F1  F2  F3 REORG
-------------------------------------------------------------------------------------
Table: SCHEMA1.T1

                            879      0     14     14      -    51861   0 100 100 ---
-------------------------------------------------------------------------------------

인덱스 통계:

F4: CLUSTERRATIO 또는 표준화된 CLUSTERFACTOR > 80
F5: 100 * (리프 페이지에 사용된 스페이스 / 비어 있지 않은 리프 페이지에서 사용할 수 있는 스페이스) > MIN(50,(100 - PCTFREE))
F6: (100 - PCTFREE) * (한 레벨 낮은 인덱스에서 사용할 수 있는 스페이스의 양 / 모든 키에 필요한 스페이스의 양) < 100
F7: 100 * (의사 삭제된 RID 수 / 총 RID 수) < 20
F8: 100 * (의사 빈 리프 페이지 수 / 총 리프 페이지 수) < 20

SCHEMA.NAME                INDCARD  LEAF ELEAF LVLS  NDEL    KEYS LEAF_RECSIZE NLEAF_RECSIZE LEAF_PAGE_OVERHEAD NLEAF_PAGE_OVERHEAD  PCT_PAGES_SAVED  F4  F5  F6  F7  F8 REORG
-------------------------------------------------------------------------------------------------------------------------------------------
Table: SCHEMA1.T1
Index: SCHEMA1.I1

                            879    15     0     2     0     682           20                596                596                28  56  31   -   0   0 ----
-------------------------------------------------------------------------------------------------------------------------------------------
```

예 3: 인덱스 압축의 잠재적인 스페이스 절약 판별

부분 인덱스 압축 절약을 계산할 수 있는 방법의 예는
ADMIN_GET_INDEX_COMPRESS_INFO 테이블 함수에 대한 문서를 참조하십시오.

인덱스 작성

인덱스는 여러 가지 이유 중에서 쿼리를 더 효율적으로 실행하기 위해, 컬럼의 값에 따라서 테이블의 행을 오름차순 또는 내림차순으로 정렬하기 위해 또는 인덱스 키에 대한 고유성 같은 제한조건을 시행하기 위해 작성할 수 있습니다. CREATE INDEX문, DB2 디자인 어드바이저 또는 db2advis 디자인 어드바이저 명령을 사용하여 인덱스를 작성할 수 있습니다.

이 태스크는 사용자가 파티션되지 않은 테이블에 인덱스를 작성 중이라고 가정합니다.

명령행에서 CREATE INDEX문을 사용하여 인덱스를 작성하려면 다음을 입력하십시오.

```
CREATE UNIQUE INDEX EMP_IX
  ON EMPLOYEE(EMPNO)
  INCLUDE(FIRSTNAME, JOB)
```

고유 인덱스에만 적용할 수 있는 INCLUDE절은 인덱스 키 컬럼 세트에 추가될 추가 컬럼을 지정합니다. 이 절에 포함된 어떤 컬럼도 고유성을 강요하기 위해 사용할 수 없습니다. 이들 포함된 컬럼은 인덱스 전용 액세스를 통해 일부 쿼리의 성능을 개선할 수 있습니다. 이 옵션으로 다음을 수행할 수 있습니다.

- 추가 쿼리를 위해 데이터 페이지에 액세스할 필요성 제거
- 중복 인덱스 제거

SELECT EMPNO, FIRSTNAME, JOB FROM EMPLOYEE가 이 인덱스가 상주하는 테이블에 대해 발행되는 경우 데이터 페이지를 읽지 않고 필요한 모든 데이터를 인덱스에서 검색할 수 있습니다. 따라서 성능이 향상됩니다.

주: 행이 삭제되거나 갱신될 때, 삭제 또는 갱신이 커미트된 이후에 정리가 수행될 때까지 인덱스 키는 삭제된 것으로 표시되며 페이지에서 실제로 제거되지 않습니다. 이런 키를 의사 삭제된 키라고 합니다. 이러한 정리는 키가 삭제된 것으로 표시된 페이지를 변경하는 후속 트랜잭션에 의해 수행될 수 있습니다. 의사 삭제된 키의 정리는 REORG INDEXES 유틸리티의 CLEANUP ONLY ALL 옵션을 사용하여 명시적으로 트리거할 수 있습니다.

주: Solaris 플랫폼에서는 원시 디바이스에 인덱스를 작성하기 위해 Solaris 9에 대한 패치 122300-11 또는 Solaris 10에 대한 125100-07이 필요합니다. 이 패치가 없으면 원시 디바이스가 사용되는 경우 CREATE INDEX문이 정지합니다.

파티션된 테이블에 대한 파티션되지 않은 인덱스 작성

파티션된 테이블에 대한 *파티션되지 않은 인덱스*를 작성할 때 테이블의 모든 행을 참조하는 단일 인덱스 오브젝트를 작성합니다. 파티션되지 않은 인덱스는 테이블 데이터 파티션이 여러 테이블 스페이스에 걸쳐있는 경우에도 항상 단일 테이블 스페이스에 작성됩니다.

시작하기 전에

이 태스크는 파티션된 테이블이 이미 작성되었다고 가정합니다.

이 태스크에 대한 정보

프로시저

1. NOT PARTITIONED절을 사용하여 테이블에 대한 CREATE INDEX문을 공식화하십시오. 예를 들어,

   ```
   CREATE INDEX indexName ON tableName(column) NOT PARTITIONED
   ```

2. 지원되는 DB2 인터페이스에서 CREATE INDEX문을 실행하십시오.

예

예 1: 데이터 파티션과 동일한 테이블 스페이스에 파티션되지 않은 인덱스 작성

SALES 테이블이 다음과 같이 정의된다고 가정하십시오.

```
CREATE TABLE sales(store_num INT, sales_date DATE, total_sales DECIMAL (6,2)) IN ts1
   PARTITION BY RANGE(store_num)
      (STARTING FROM (1) ENDING AT (100),
    STARTING FROM (101) ENDING AT (150),
    STARTING FROM (151) ENDING AT (200))
```

SALES 테이블의 세 파티션이 테이블 스페이스 TS1에 저장됩니다. 디폴트로 이 테이블에 대해 작성되는 모든 인덱스도 TS1에 저장되는데, 이것이 이 테이블에 대해 지정된 테이블 스페이스이기 때문입니다. STORE_NUM 컬럼에 대한 파티션되지 않은 인덱스 STORENUM을 작성하려면 다음 명령문을 사용하십시오.

```
CREATE INDEX StoreNum ON sales(store_num) NOT PARTITIONED
```

NOT PARTITIONED절이 필수이며, 그렇지 않으면 인덱스는 파티션된 테이블에 대한 디폴트인 파티션된 인덱스로서 작성됩니다.

예 2: 디폴트가 아닌 테이블 스페이스에 파티션되지 않은 인덱스 작성

PARTS 테이블이 다음과 같이 정의되었다고 가정하십시오.

```
CREATE TABLE parts(part_number INT, manufacturer CHAR, description CLOB, price DECIMAL (4,2)) IN ts1 INDEX in ts2
PARTITION BY RANGE (part_number)
       (STARTING FROM (1) ENDING AT (10) IN ts3,
        STARTING FROM (11) ENDING AT (20) INDEX IN ts1,
        STARTING FROM (21) ENDING AT (30) IN ts2 INDEX IN ts4);
```

PARTS 테이블은 세 파티션으로 구성되는데, 첫 번째는 테이블 스페이스 TS3에 있고, 두 번째는 TS2, 세 번째는 TS3에 있습니다. 다음 명령문을 발행하면 행을 제조업체 이름의 내림차순으로 정렬하는 파티션되지 않은 인덱스가 작성됩니다.

```
CREATE INDEX manufct on parts(manufacturer DESC) NOT PARTITIONED IN TS3;
```

이 인덱스는 테이블 스페이스 TS3에 작성됩니다. CREATE TABLE문의 INDEX IN 절은 CREATE TABLE문의 *tablespace* 절로 겹쳐써집니다. PARTS 테이블이 파티션되기 때문에, 파티션되지 않은 인덱스를 작성하려면 CREATE INDEX문에 NOT PARTITIONED절을 포함해야 합니다.

파티션된 인덱스 작성

파티션된 테이블에 대한 *파티션된 인덱스*를 작성할 때 각 데이터 파티션은 고유한 인덱스 파티션에서 인덱스화됩니다. 디폴트로 인덱스 파티션은 인덱스화하는 데이터 파티션과 동일한 테이블 스페이스에서 저장됩니다. 인덱스의 데이터는 테이블의 분산 키에 기반하여 분산됩니다.

시작하기 전에

이 태스크는 파티션된 테이블이 이미 작성되었다고 가정합니다.

제한사항

파티션할 수 없는 몇 가지 유형의 인덱스가 있습니다.

- XML 데이터에 대한 인덱스
- 공간 데이터에 대한 인덱스
- MDC 블록 인덱스(시스템 생성)
- XML 컬럼 경로 인덱스(시스템 생성)

이러한 경우 인덱스는 반드시 파티션되지 않은 것으로 작성되어야 합니다. 또한 파티션된 고유 인덱스에 대한 인덱스 키는 사용자 생성 키 또는 시스템 생성 키 모두 테이블 파티션 키의 모든 컬럼을 포함해야 합니다. 후자는 데이터에 대한 고유성 또는 기본 제한조건을 강제하기 위해 시스템이 작성하는 인덱스의 경우입니다.

또한, CREATE INDEX문의 IN절이 파티션된 인덱스 작성에 대해 지원되지 않습니다. 디폴트로 인덱스 파티션은 인덱스화하는 데이터 파티션과 동일한 테이블 스페이스에 작성됩니다. 인덱스 파티션을 저장할 대체 테이블 스페이스를 지정하려면, CREATE TABLE문의 파티션 레벨 INDEX IN절을 사용하여 파티션별로 인덱스에 대한 테이블 스페이스를 지정해야 합니다. 이 절을 생략하면 인덱스 파티션은 인덱스화하는 데이터 파티션과 동일한 테이블 스페이스에서 상주합니다.

프로시저

1. PARTITIONED절을 사용하여 테이블에 대한 CREATE INDEX문을 공식화하십시오.
2. 지원되는 DB2 인터페이스에서 CREATE INDEX문을 실행하십시오.

예

주: 이들 예는 설명만을 위한 것이며, 파티션된 테이블 또는 인덱스 작성에 대한 우수 사례를 반영하지 않습니다.

예 1: 데이터 파티션과 동일한 테이블 스페이스에 파티션된 인덱스 작성

이 예에서는 SALES 테이블이 다음과 같이 정의되었다고 가정하십시오.

```
   CREATE TABLE sales(store_num INT, sales_date DATE, total_sales DECIMAL (6,2)) IN ts1
 PARTITION BY RANGE(store_num)
         (STARTING FROM (1) ENDING AT (100),
        STARTING FROM (101) ENDING AT (150),
        STARTING FROM (151) ENDING AT (200))
```

이 경우에 SALES 테이블의 세 파티션이 테이블 스페이스 ts1에 저장됩니다. 이 테이블에 대해 작성되는 모든 파티션된 인덱스도 ts1에 저장됩니다. 이것이 테이블에 대한 각 파티션이 저장될 테이블 스페이스입니다. 저장 번호에 대한 파티션된 인덱스를 작성하려면 다음 명령문을 사용하십시오.

```
CREATE INDEX StoreNum ON sales(store_num) PARTITIONED
```

예 2: 모든 인덱스 파티션에 대한 대체 위치 선택

이 예에서는 EMPLOYEE 테이블이 다음과 같이 정의되었다고 가정하십시오.

```
CREATE TABLE employee(employee_number INT, employee_name CHAR, job_code INT, city CHAR, salary DECIMAL (6,2))
IN ts1 INDEX in ts2 PARTITION BY RANGE (job_code)
        (STARTING FROM (1) ENDING AT (10),
         STARTING FROM (11) ENDING AT (20),
         STARTING FROM (21) ENDING AT (30))
```

작업 코드에 대한 파티션된 인덱스를 작성하려면 다음 명령문을 사용하십시오.

```
CREATE INDEX JobCode ON employee(job_code) PARTITIONED
```

이 예에서 EMPLOYEE 테이블의 파티션은 테이블 스페이스 ts1에 저장되지만, 모든 인덱스 파티션은 ts2에 저장됩니다.

예 3: 여러 파티션에 작성되는 인덱스

PARTS 테이블이 다음과 같이 정의되었다고 가정하십시오.

```
CREATE TABLE parts(part_number INT, manufacturer CHAR, description CLOB, price DECIMAL (4,2))
IN ts1 INDEX in ts2 PARTITION BY RANGE (part_number)
        (STARTING FROM (1) ENDING AT (10) IN ts3,
         STARTING FROM (11) ENDING AT (20) INDEX IN ts1,
         STARTING FROM (21) ENDING AT (30) IN ts2 INDEX IN ts4);
```

이 경우에 PARTS 테이블은 세 파티션으로 구성되는데, 첫 번째는 테이블 스페이스 ts3, 두 번째는 ts1 및 세 번째는 ts2에 있습니다. 다음 명령문이 발행되는 경우,

```
CREATE INDEX partNoasc ON parts(part_number ASC)  PARTITIONED
CREATE INDEX manufct on parts(manufacturer DESC) NOT PARTITIONED IN TS3;
```

두 인덱스가 작성됩니다. 첫 번째는 행을 부품 번호의 오름차순으로 정렬하기 위한 파티션된 인덱스입니다. 첫 번째 인덱스 파티션은 테이블 스페이스 ts3에 작성되고, 두 번

째는 ts1, 세 번째는 ts4에 작성됩니다. 두 번째 인덱스는 행을 제조업체 이름의 내림차순으로 정렬하는 파티션되지 않은 인덱스입니다. 이 인덱스는 ts3에 작성됩니다. IN 절은 파티션되지 않은 인덱스에 대한 CREATE INDEX문에서 허용됨을 주의하십시오. 또한 이 경우에 PARTS 테이블이 파티션되기 때문에 파티션되지 않은 인덱스를 작성하려면 CREATE INDEX문에 NOT PARTITIONED절을 포함해야 합니다.

인덱스 수정

ALTER INDEX문을 사용하여 인덱스 압축을 사용 또는 사용 안하는 대신 인덱스를 수정하려는 경우 먼저 인덱스를 삭제한 후 인덱스를 다시 작성해야 합니다.

예를 들어 이전 정의를 삭제하고 새 인덱스를 작성하지 않고 키 컬럼 목록에 컬럼을 추가할 수 없습니다. 그러나 COMMENT문을 사용하여 인덱스의 목적을 설명하는 주석을 추가할 수 있습니다.

인덱스 이름 바꾸기

RENAME문을 사용하여 기존 인덱스의 이름을 바꿀 수 있습니다.

기존 인덱스의 이름을 바꾸려면 명령행에서 다음 명령문을 발행하십시오.

```
RENAME INDEX <source index name> TO <target index name>
```

- <source index name>은 이름을 바꿀 기존 인덱스의 이름입니다. 스키마 이름을 포함한 이름은 데이터베이스에 이미 존재하는 인덱스를 식별해야 합니다. 선언된 임시 테이블 또는 작성된 임시 테이블에 대한 인덱스의 이름이 아니어야 합니다. 스키마 이름이 SYSIBM, SYSCAT, SYSFUN 또는 SYSSTAT가 아니어야 합니다.
- <target index name>은 스키마 이름 없이 인덱스에 대한 새 이름을 지정합니다. 소스 오브젝트의 스키마 이름이 오브젝트에 대한 새 이름을 규정하는 데 사용됩니다. 규정된 이름은 데이터베이스에 이미 존재하는 인덱스를 식별하지 않아야 합니다.

인덱스 이름을 바꿀 때 소스 인덱스가 시스템 생성 인덱스가 아니어야 합니다. 명령문이 성공하면 시스템 카탈로그 테이블이 갱신되어 새 인덱스 이름을 반영합니다.

인덱스 재빌드

완전히 로그되지 않은 인덱스 작성을 통한 롤 포워드 같은 일부 데이터베이스 조작은 인덱스가 롤 포워드 조작 중에 작성되지 않기 때문에 인덱스 오브젝트가 유효하지 않게 될 수 있습니다. 인덱스 오브젝트는 인덱스를 다시 작성하여 복구할 수 있습니다.

데이터베이스 관리 프로그램이 인덱스가 더 이상 유효하지 않음을 발견할 때 자동으로 인덱스 재빌드를 시도합니다. 재빌드가 발생할 때 이는 데이터베이스 또는 데이터베이스 관리 프로그램 구성 파일의 **indexrec** 매개변수에 의해 제어됩니다. 이에 대한 5가지 가능한 설정이 있습니다.

- SYSTEM

- RESTART
- RESTART_NO_REDO
- ACCESS
- ACCESS_NO_REDO

RESTART_NO_REDO 및 ACCESS_NO_REDO는 RESTART 및 ACCESS와 유사합니다.

NO_REDO 옵션은 CREATE INDEX 같은 원래 조작 중에 인덱스가 완전히 로그된 경우에도 인덱스는 롤 포워드 중에 다시 작성되지 않고 대신 재시작 시간이나 첫 번째 액세스 시에 작성됨을 의미합니다. 자세한 정보는 **indexrec** 매개변수를 참조하십시오.

데이터베이스 재시작 시간이 문제가 아닌 경우, 유효하지 않은 인덱스가 데이터베이스를 일관성 있는 상태로 리턴하는 프로세스의 일부로 재빌드되는 것이 더 좋습니다. 이 방법을 사용할 때 데이터베이스를 재시작하는 데 필요한 시간은 인덱스 재작성 프로세스로 인해 더 길어지지만, 데이터베이스가 다시 일관성 있는 상태로 돌아간 후에는 정상 처리가 영향을 받지 않습니다.

한편, 인덱스가 액세스될 때 재빌드되면 데이터베이스를 재시작하는 데 걸리는 시간이 더 빠르지만, 재작성되는 인덱스의 결과로 예기치 않은 응답 시간의 저하가 발생할 수 있습니다. 예를 들어 유효하지 않은 인덱스를 갖는 테이블에 액세스하는 사용자는 인덱스가 재빌드되기를 기다려야 합니다. 또한, 특히 인덱스 재작성이 발생하도록 만든 트랜잭션(즉, 변경이 작성한 커미트 또는 롤백)이 종료하지 않는 경우 유효하지 않은 인덱스가 재작성된 후 예기치 않은 잠금을 획득하고 오래 동안 보유될 수 있습니다.

인덱스 삭제

인덱스의 COMPRESSION 속성을 변경하는 것 외에는 인덱스 정의의 어떤 절도 변경할 수 없습니다. 인덱스를 삭제(drop)하고 다시 작성해야 합니다. (인덱스를 삭제해도 다른 오브젝트가 삭제되지 않지만 일부 패키지가 무효화될 수 있습니다.) 인덱스를 삭제하려면 DROP문을 사용하십시오.

기본 키 또는 고유 키는 명시적으로 삭제할 수 없습니다. 다음 방법 중 하나를 사용하여 이를 제거해야 합니다.

- 1차 인덱스 또는 고유 제한조건이 기본 키 또는 고유 키를 사용하여 자동으로 작성된 경우, 기본 키 또는 고유 키를 삭제하면 인덱스가 삭제됩니다. 삭제는 ALTER TABLE문을 사용하여 완료됩니다.
- 1차 인덱스 또는 고유 제한조건이 사용자 정의된 경우, 기본 키 또는 고유 키는 ALTER TABLE문을 사용하여 먼저 삭제되어야 합니다. 기본 키 또는 고유 키가 제거된 후, 인덱스는 더 이상 1차 인덱스 또는 고유 인덱스로 간주되지 않고, 명시적으로 삭제될 수 있습니다.

명령행을 사용하여 인덱스를 삭제하려면 다음을 입력하십시오.

```
DROP INDEX <index_name>
```

다음 명령문은 PH라는 인덱스를 삭제합니다.

```
DROP INDEX PH
```

삭제된 인덱스에 종속된 모든 패키지 및 캐시된 동적 SQL 및 XQuery문은 유효하지 않은 것으로 표시됩니다. 응용프로그램은 인덱스의 추가 또는 제거로 인한 변경사항의 영향을 받지 않습니다.

제 14 장 트리거

*트리거*는 지정된 테이블에서 삽입, 갱신 또는 삭제 조작에 대한 응답으로 수행되는 조치 세트를 정의합니다. 이러한 SQL 조작이 실행되면 트리거가 *활성화*되었다고 합니다. 트리거는 선택적이며 CREATE TRIGGER문을 사용하여 정의됩니다.

참조 제한조건 및 점검 제한조건과 함께 트리거를 사용하여 데이터 무결성 규칙을 적용할 수 있습니다. 또한 트리거를 사용하여 기타 테이블이 갱신되도록 하거나, 삽입되거나 갱신된 행의 값을 자동으로 생성 또는 변환하거나, 함수를 호출하여 경보 발행과 같은 태스크를 수행할 수 있습니다.

트리거는 *트랜잭션* 비즈니스 규칙을 정의하고 적용하는 데 유용한 메커니즘으로, 트랜잭션 비즈니스 규칙은 여러 데이터 상태를 포함하는 규칙입니다(예: 급여를 10% 넘게 인상할 수 없음).

트리거를 사용하면 비즈니스 규칙을 적용하는 논리가 데이터베이스 내에 한정됩니다. 이는 응용프로그램에서는 이러한 규칙을 적용할 책임이 없다는 의미입니다. 모든 테이블에 적용되는 중심 논리를 사용하면 논리가 변경될 경우 응용프로그램을 변경할 필요가 없기 때문에 유지보수가 간편합니다.

트리거 작성 시 다음 내용이 지정됩니다.

- *주제 테이블*은 트리거가 정의된 테이블을 지정합니다.
- *트리거 이벤트*는 주제 테이블을 수정하는 특정 SQL 조작을 정의합니다. 이벤트는 삽입, 갱신 또는 삭제 조작 중 하나입니다.
- *트리거 활성화 시간*은 트리거 이벤트 발생 이전 또는 이후에 트리거를 활성화할지 여부를 지정합니다.

트리거를 활성화시키는 명령문에는 *영향받은 행 세트*가 포함됩니다. 이들 행은 삽입, 갱신 또는 삭제되는 중인 주제 테이블의 행입니다. *트리거 수준*은 트리거의 조치를 명령문마다 한 번 수행할지 또는 각각의 영향받은 행마다 한 번 수행할지 여부를 지정합니다.

*트리거 조치*는 선택적 검색 조건과 트리거가 활성화될 때마다 실행되는 명령문 세트로 구성됩니다. 명령문은 검색 조건이 참으로 평가하는 경우에만 실행됩니다. 트리거 활성화 시간이 트리거 이벤트 이전인 경우 트리거 조치에는 선택하거나, 전이 변수를 설정하거나, SQL 상태를 신호하는 명령문이 포함됩니다. 트리거 활성화 시간이 트리거 이벤트 이후인 경우에는 트리거 조치에 선택, 삽입, 갱신, 삭제 또는 SQL 상태를 신호하는 명령문이 포함됩니다.

트리거 조치에서는 *전이 변수*를 사용하여 영향받은 행 세트의 값을 참조할 수 있습니다. 전이 변수에서는 주제 테이블의 컬럼 이름을 사용하며 이 이름은 참조가 갱신 이전의 이전 값에 대한 것인지 또는 갱신 이후의 새 값에 대한 것이지 여부를 식별하는 지정된 이름으로 규정됩니다. 사전, 삽입 또는 갱신 트리거의 SET 변수 명령문을 사용하여 새 값을 변경할 수도 있습니다.

영향받은 행 세트의 값을 참조하는 또 다른 방법은 *전이 테이블*을 사용하는 것입니다. 전이 테이블에서도 주제 테이블의 컬럼 이름을 사용하지만 영향받은 행의 전체 세트를 테이블로 취급할 수 있도록 이름을 지정합니다. AFTER 트리거에서만 전이 테이블을 사용할 수 있으며(즉, 사전 및 INSTEAD OF 트리거에서는 사용할 수 없음) 이전 및 새 값에 개별 전이 테이블을 정의할 수 있습니다.

테이블, 이벤트(삽입, 갱신, 삭제, INSTEAD OF) 또는 활성화 시간(사전, 사후)의 조합에 다중 트리거를 지정할 수 있습니다. 특정 테이블, 이벤트 및 활성화 시간에 둘 이상의 트리거가 존재하는 경우 트리거가 활성화되는 순서는 트리거 작성 순서와 동일합니다. 따라서 최근에 작성된 트리거가 마지막으로 활성화되는 트리거입니다.

트리거를 활성화시키면 *트리거 연쇄*가 발생할 수 있습니다. 트리거 연쇄는 다른 트리거 또는 동일한 트리거를 다시 활성화시키는 명령문을 실행하는 하나의 트리거를 활성화시킴으로써 발생합니다. 트리거된 조치는 또한 삭제에 적용되는 참조 무결성 규칙의 응용프로그램으로 인해 발생하는 갱신의 원인이 될 수 있습니다. 그런 다음 추가적인 트리거 활성화가 발생합니다. 트리거 연쇄가 발생하면 트리거 및 참조 무결성 삭제 규칙의 체인이 활성화되어 단일 INSERT, UPDATE 또는 DELETE문 실행으로 데이터베이스가 상당히 변경될 수 있습니다.

다중 트리거에 동일한 오브젝트에 대한 삽입, 갱신 또는 삭제 조치가 있는 경우, 임시 테이블과 마찬가지로 액세스 충돌을 해결하기 위해 충돌 해결 메커니즘이 사용되며 이는 성능에(특히 파티션된 데이터베이스 환경에서) 상당한 영향을 미칠 수 있습니다.

트리거 유형

*트리거*는 지정된 테이블에서 삽입, 갱신 또는 삭제 조작에 대한 응답으로 수행되는 조치 세트를 정의합니다. 이러한 SQL 조작이 실행되면 트리거가 *활성화*되었다고 합니다. 트리거는 선택적이며 CREATE TRIGGER문을 사용하여 정의됩니다.

참조 제한조건 및 점검 제한조건과 함께 트리거를 사용하여 데이터 무결성 규칙을 적용할 수 있습니다. 또한 트리거를 사용하여 기타 테이블이 갱신되도록 하거나, 삽입되거나 갱신된 행의 값을 자동으로 생성 또는 변환하거나, 함수를 호출하여 경보 발행과 같은 태스크를 수행할 수 있습니다.

다음과 같은 트리거 유형이 지원됩니다.

BEFORE 트리거

갱신 또는 삽입 이전에 실행합니다. 갱신 또는 삽입 중인 값을 데이터베이스가 실제로 수정되기 전에 수정할 수 있습니다. 여러가지 방법으로 갱신 또는 삽입 이전에 실행되는 트리거를 사용할 수 있습니다.

- 데이터베이스에서 값이 실제로 갱신되거나 삽입되기 전에 값을 점검하거나 수정합니다. 데이터를 사용자에게 표시되는 방식에서 내부 데이터베이스 형식으로 변환해야 하는 경우 유용합니다.
- 사용자 정의 함수(UDF)로 코딩된, 데이터베이스 조작이 아닌 기타 조작을 실행합니다.

BEFORE DELETE 트리거

삭제 이전에 실행합니다. 값을 점검하고 필요한 경우 오류를 발생시킵니다.

AFTER 트리거

갱신, 삽입 또는 삭제 이후에 실행합니다. 여러가지 방법으로 갱신 또는 삽입 이후에 실행되는 트리거를 사용할 수 있습니다.

- 다른 테이블의 데이터를 갱신합니다. 이 기능은 데이터 간 관계를 유지하거나 감사 추적 정보를 보존하는 데 유용합니다.
- 테이블 또는 다른 테이블의 기타 데이터에 대해 점검합니다. 이 기능은 참조 무결성 제한조건이 적합하지 않거나 테이블 점검 제한조건이 현재 테이블만 점검하도록 제한하는 경우 데이터 무결성을 확인하는 데 유용합니다.
- 사용자 정의 함수(UDF)로 코딩된, 데이터베이스 조작이 아닌 조작을 실행합니다. 이 기능은 경보를 발행하거나 데이터베이스 외부에서 정보를 갱신하는 데 유용합니다.

INSTEAD OF 트리거

너무 복잡해서 삽입, 갱신 및 삭제 조작을 지원할 수 없는 뷰에 대해 이들 조작을 수행하는 방법에 대해서 설명합니다. 응용프로그램에서 모든 SQL 조작(삽입, 삭제, 갱신 및 선택)을 수행할 단일 인터페이스로 뷰를 사용하도록 허용합니다.

BEFORE 트리거

갱신 또는 삽입 이전에 실행되는 트리거를 사용하여 데이터베이스를 실제로 수정하기 이전에 갱신 또는 삽입되는 값을 수정할 수 있습니다. 이러한 트리거를 사용하여 응용프로그램의 입력(사용자 뷰의 데이터)을 원하는 지점에서 내부 데이터베이스 형식으로 변환할 수 있습니다.

BEFORE 트리거를 사용하여 기타 데이터베이스 조작이 아닌 조작을 사용자 정의 함수(UDF)를 통해 활성화시킬 수 있습니다.

BEFORE DELETE 트리거는 삭제 조작 이전에 실행됩니다. 이 트리거는 값을 점검하고 필요한 경우 오류를 발생시킵니다.

예

다음 예에서는 복합 디폴트값으로 DELETE TRIGGER를 정의합니다.

```
CREATE TRIGGER trigger1
  BEFORE UPDATE ON table1
  REFERENCING NEW AS N
  WHEN (N.expected_delivery_date IS NULL)
  SET N.expected_delivery_date = N.order_date + 5 days;
```

다음 예에서는 참조 무결성 제한조건이 아닌 교차 테이블 제한조건으로 DELETE TRIGGER를 정의합니다.

```
CREATE TRIGGER trigger2
  BEFORE UPDATE ON table2
  REFERENCING NEW AS N
  WHEN (n.salary > (SELECT maxsalary FROM salaryguide WHERE rank = n.position))
  SIGNAL SQLSTATE '78000' SET MESSAGE_TEXT = 'Salary out of range');
```

AFTER 트리거

갱신, 삽입 또는 삭제 이후 실행되는 트리거를 여러 가지 방법으로 사용할 수 있습니다.

- 트리거는 동일한 테이블이나 다른 테이블에 있는 데이터를 갱신, 삽입 또는 삭제할 수 있습니다. 이는 데이터 간 관계를 유지하거나 감사 추적 정보를 보존하는 데 유용합니다.
- 트리거는 테이블의 나머지 부분 또는 기타 테이블에 있는 데이터의 값에 대해 데이터를 점검할 수 있습니다. 이는 해당 테이블 또는 기타 테이블의 다른 행의 데이터에 대한 참조 때문에 참조 무결성 제한조건이나 점검 제한조건을 사용할 수 없는 경우에 유용합니다.
- 트리거는 사용자 정의 함수(UDF)를 사용하여 데이터베이스가 아닌 조작을 활성화시킬 수 있습니다. 이는, 예를 들어, 경보를 발행하거나 데이터베이스 외부에서 정보를 갱신하는 데 유용합니다.

예 :

다음 예는 새 직원 고용 시 직원 수를 늘리는 AFTER 트리거를 나타냅니다.

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

INSTEAD OF 트리거

INSTEAD OF 트리거는 복합 뷰에 대해서 삽입, 갱신 및 삭제 조작을 수행하는 방법에 대해 설명합니다. INSTEAD OF 트리거를 사용하면 응용프로그램에서 뷰를 모든 SQL 조작(삽입, 삭제, 갱신 및 선택)의 단일 인터페이스로 사용할 수 있습니다.

일반적으로 INSTEAD OF 트리거는 뷰 본문에 적용되는 논리의 역 논리를 포함합니다. 예를 들어, 소스 테이블에서 컬럼 암호를 해독하는 뷰가 있습니다. 이 뷰의 INSTEAD OF 트리거는 데이터를 암호화한 다음 이를 소스 테이블에 삽입하여 대칭 조작을 수행합니다.

INSTEAD OF 트리거를 사용하면 뷰에 대해 요청된 수정 조작이 뷰 대신 조작을 수행하는 트리거 논리로 교체됩니다. 응용프로그램에서는 모든 조작이 뷰에 대해 수행되는 것으로 인식하므로 이러한 조작이 투명하게 이루어지는 것으로 인식합니다. 주어진 주제 뷰에서 각 조작의 종류마다 하나의 INSTEAD OF 트리거만 허용됩니다.

뷰 자체는 유형이 지정되지 않은 뷰이거나 유형이 지정되지 않은 뷰를 해석하는 별명이어야 합니다. 또한 WITH CHECK OPTION(대칭 뷰)을 사용하여 정의되는 뷰 또는 직/간접적으로 대칭 뷰가 정의된 뷰가 될 수 없습니다.

예 :

다음 예에는 정의된 뷰(EMPV)에 INSERT, UPDATE 및 DELETE의 논리를 제공하는 세 개의 INSTEAD OF 트리거가 있습니다. EMPV 뷰의 FROM절에 조인이 포함되어 있어 기본적으로 수정 조작을 지원할 수 없습니다.

```
CREATE VIEW EMPV(EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO,
                 HIREDATE, DEPTNAME)
AS SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO,
          HIREDATE, DEPTNAME
          FROM EMPLOYEE, DEPARTMENT WHERE
               EMPLOYEE.WORKDEPT = DEPARTMENT.DEPTNO

CREATE TRIGGER EMPV_INSERT INSTEAD OF INSERT ON EMPV
REFERENCING NEW AS NEWEMP FOR EACH ROW
INSERT INTO EMPLOYEE (EMPNO, FIRSTNME, MIDINIT, LASTNAME,
                      WORKDEPT, PHONENO, HIREDATE)
        VALUES(EMPNO, FIRSTNME, MIDINIT, LASTNAME,
               COALESCE((SELECT DEPTNO FROM DEPARTMENT AS D
                         WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                        RAISE_ERROR('70001', 'Unknown dept name')),
               PHONENO, HIREDATE)

CREATE TRIGGER EMPV_UPDATE INSTEAD OF UPDATE ON EMPV
REFERENCING NEW AS NEWEMP OLD AS OLDEMP
  FOR EACH ROW
BEGIN ATOMIC
 VALUES(CASE WHEN NEWEMP.EMPNO = OLDEMP.EMPNO THEN 0
             ELSE RAISE_ERROR('70002', 'Must not change EMPNO') END);
 UPDATE EMPLOYEE AS E
   SET (FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE)
     = (NEWEMP.FIRSTNME, NEWEMP.MIDINIT, NEWEMP.LASTNAME,
               COALESCE((SELECT DEPTNO FROM DEPARTMENT AS D
                         WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                        RAISE_ERROR ('70001', 'Unknown dept name')),
               NEWEMP.PHONENO, NEWEMP.HIREDATE)
 WHERE NEWEMP.EMPNO = E.EMPNO;
END
```

```
CREATE TRIGGER EMPV_DELETE INSTEAD OF DELETE ON EMPV
REFERENCING OLD AS OLDEMP FOR EACH ROW
DELETE FROM EMPLOYEE AS E WHERE E.EMPNO = OLDEMP.EMPNO
```

트리거 설계

트리거를 작성할 때 해당 트리거를 테이블과 연결시켜야 합니다. INSTEAD OF 트리거를 작성하는 경우에는 해당 트리거를 뷰와 연결시켜야 합니다. 이러한 테이블 또는 뷰를 트리거의 *목표 테이블*이라고 합니다. 수정 조작이라는 용어는 목표 테이블의 상태에 대한 변경을 지칭합니다.

*수정 조작*은 다음에 의해 시작됩니다.

- INSERT문
- UPDATE문 또는 UPDATE를 수행하는 참조 제한조건
- DELETE문 또는 DELETE를 수행하는 참조 제한조건
- MERGE문

수정 조작의 이러한 세 가지 유형 중 하나와 각 트리거를 연결시켜야 합니다. 이러한 연관을 특정 트리거의 *트리거 이벤트*라고 합니다.

또한 *트리거 조치*라는 조치를 정의해야 합니다. 이러한 조치는 트리거 이벤트가 발생한 경우 트리거가 수행하는 조치입니다. 트리거 조치는 데이터베이스 관리 프로그램이 트리거 이벤트를 수행하기 전 또는 수행한 후 실행할 수 있는 하나 이상의 명령문으로 구성됩니다. 트리거 이벤트가 발생하면 데이터베이스 관리 프로그램은 수정 조작이 영향을 미치는 주제 테이블의 행 세트를 판별하고 트리거를 실행합니다.

트리거 작성 시 지침:

트리거를 작성하는 경우 다음 속성 및 동작을 선언해야 합니다.

- 트리거 이름
- 주제 테이블의 이름
- 트리거 활성화 이름(수정 조작 실행 전/후)
- 트리거 이벤트(INSERT, DELETE 또는 UPDATE)
- 이전 전이 변수 값(있는 경우)
- 새 전이 변수 값(있는 경우)
- 이전 전이 테이블 값(있는 경우)
- 새 전이 테이블 값(있는 경우)
- 트리거 수준(FOR EACH STATEMENT 또는 FOR EACH ROW).
- 트리거의 트리거 조치(트리거 조치 조건 및 트리거 명령문 포함)

- 트리거 이벤트가 UPDATE인 경우 특정 컬럼이 갱신 명령문에 지정된 경우에만 트리거가 실행되어야 하면 trigger-column이 나열됩니다.

다중 트리거 설계:

CREATE TRIGGER문을 사용하여 트리거가 정의되면 트리거의 작성 시간이 시간소인으로 데이터베이스에 등록됩니다. 이 시간소인의 값은 동시에 실행되어야 하는 트리거가 하나 이상 있는 경우 이후에 트리거 활성화 순서를 지정하는 데 사용됩니다. 예를 들어 동일한 주제 테이블에 이벤트와 활성화 시간이 동일한 트리거가 하나 이상 있는 경우 시간소인이 사용됩니다. 또한 시간소인은 트리거 조치에 의해 직접 또는 간접적으로(즉, 다른 참조 제한조건에 의해 반복적임) 발생하는 트리거 이벤트 및 참조 제한조건 조치에 의해 활성화되는 AFTER 또는 INSTEAD OF 트리거가 하나 이상 있는 경우에도 사용됩니다.

다음 두 가지 트리거를 자세히 살펴보십시오.

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  BEGIN ATOMIC
    UPDATE COMPANY_STATS
    SET NBEMP = NBEMP + 1;
  END

CREATE TRIGGER NEW_HIRED_DEPT
   AFTER INSERT ON EMPLOYEE
   REFERENCING NEW AS EMP
   FOR EACH ROW
     BEGIN ATOMIC
       UPDATE DEPTS
       SET NBEMP = NBEMP + 1
       WHERE DEPT_ID = EMP.DEPT_ID;
          END
```

위 트리거는 Employee 테이블에서 INSERT 조작을 실행하는 경우 활성화됩니다. 이 경우 트리거 작성 시간소인은 위의 트리거 두 개 중 먼저 활성화되는 트리거를 정의합니다.

트리거 활성화는 시간소인 값에 따라 오름차순으로 실행됩니다. 따라서 데이터베이스에 새로 추가된 트리거는 이전에 정의된 다른 모든 트리거 이후에 실행됩니다.

새 트리거가 데이터베이스에 영향을 미치는 변경사항에 대한 증분 추가로 사용될 수 있도록 이전 트리거는 새 트리거보다 먼저 활성화됩니다. 예를 들어 트리거 T1의 트리거 명령문이 테이블 T에 새 행을 삽입하면 트리거 T1 뒤에 실행되는 트리거 T2를 특정 값이 포함된 테이블 T의 동일한 행을 갱신하는 데 사용할 수 있습니다. 트리거의 활성화 순서는 예상할 수 있으므로 테이블에 여러 트리거가 있을 수 있고 이전 트리거에 의해 이미 수정된 테이블에서 최신 트리거가 작동할 수 있음을 알 수 있습니다.

참조 제한조건을 사용한 트리거 상호 작용:

트리거 이벤트는 참조 제한조건 적용으로 인한 변경의 결과로 발생할 수 있습니다. 예를 들어 DEPT 및 EMP라는 두 개의 테이블이 있다고 가정해 봅니다. DEPT를 삭제 또는 갱신하면 참조 무결성 제한조건으로 인해 삭제 또는 갱신의 영향이 EMP로 전달되는 경우 DEPT에 대해 정의된 참조 제한조건의 결과로 EMP에 정의된 삭제 또는 갱신 트리거가 활성화됩니다. 활성화 시간에 따라 EMP에 대한 트리거가 삭제(ON DELETE CASCADE의 경우) 이전 또는 이후에 실행되고 EMP의 행을 갱신합니다(ON DELETE SET NULL의 경우).

트리거를 실행하는 항목 지정(트리거링 명령문 또는 이벤트)

모든 트리거는 이벤트와 연관되어 있습니다. 트리거는 데이터베이스에서 해당 이벤트가 발생하는 경우 활성화됩니다. 이러한 트리거 이벤트는 목표 테이블에서 지정한 조치, UPDATE, INSERT 또는 DELETE문(참조 제한조건의 조치로 인한 조치 포함)이 수행되는 경우 발생합니다.

예를 들어, 다음과 같습니다.

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

위의 명령문은 Eemployee 테이블에 대한 삽입 조치가 수행된 경우 활성화되는 new_hire 트리거를 정의합니다.

모든 트리거 이벤트 및 그에 따른 모든 트리거를 정확히 하나의 목표 테이블과 하나의 수정 조작과 연관시킵니다. 수정 조작은 다음과 같습니다.

삽입 조작

삽입 조작은 INSERT문 또는 MERGE문의 삽입 조작을 통해서만 수행될 수 있습니다. 따라서 INSERT를 사용하지 않는 유틸리티(예: LOAD 명령)를 사용하여 데이터를 로드하는 경우 트리거는 활성화되지 않습니다.

삭제 조작

삭제 조작은 DELETE문 또는 MERGE문의 삭제 조작을 통해서나 ON DELETE CASCADE의 참조 제한조건 규칙의 결과로 수행될 수 있습니다.

갱신 조작

갱신 조작은 UPDATE문 또는 MERGE문의 갱신 조작을 통해서나 ON DELETE SET의 참조 제한조건 규칙의 결과로 수행될 수 있습니다.

트리거 이벤트가 갱신 조작이면 해당 이벤트는 목표 테이블의 특정 컬럼과 연관될 수 있습니다. 이러한 경우 갱신 조작이 지정한 컬럼 중 일부를 갱신하려고 하는 경우에만 트리거가 활성화됩니다. 그러면 트리거를 활성화하는 이벤트를 보다 세분화할 수 있습니다.

예를 들어 다음 트리거 REORDER는 PARTS 테이블의 ON_HAND 또는 MAX_STOCKED 컬럼에 대한 갱신 조작이 수행되는 경우에만 활성화됩니다.

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
  FOR EACH ROW
  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
  BEGIN ATOMIC
  VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                            N_ROW.ON_HAND,
                            N_ROW.PARTNO));
      END
```

트리거가 활성화되면 다음과 같이 트리거 수준 레벨에 따라 트리거가 실행됩니다.

FOR EACH ROW

영향을 받는 행 세트의 행 수만큼 실행됩니다. 트리거 조치의 영향을 받는 특정 행을 참조해야 하는 경우 FOR EACH ROW 트리거 수준을 사용합니다. 이러한 경우의 예는 AFTER UPDATE 트리거에서 갱신된 행의 새 값과 이전 값을 비교하는 것입니다.

FOR EACH STATEMENT

전체 트리거 이벤트에 대해 한 번만 실행됩니다.

영향을 받는 행 세트가 비어 있으면(즉, WHERE절이 어떠한 행도 규정하지 않는 검색된 UPDATE 또는 DELETE의 경우) FOR EACH ROW 트리거가 실행되지 않습니다. 그러나 FOR EACH STATEMENT 트리거는 한 번만 실행됩니다.

예를 들어 FOR EACH ROW를 사용하여 직원 수를 계산할 수 있습니다.

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

트리거 수준을 사용하여 한 번의 갱신으로 동일한 효과를 얻을 수 있습니다.

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  REFERENCING NEW_TABLE AS NEWEMPS
  FOR EACH STATEMENT
  UPDATE COMPANY_STATS
  SET NBEMP = NBEMP + (SELECT COUNT(*) FROM NEWEMPS)
```

주:

- FOR EACH STATEMENT 트리거 수준은 BEFORE 트리거를 지원하지 않습니다.
- 트리거의 최대 중첩 레벨은 16입니다. 즉, 연속 트리거 활성화의 최대 수는 16입니다. 트리거 활성화는 트리거링 이벤트(예: 테이블의 컬럼에 있는 데이터 또는 일반적으로 테이블 삽입, 갱신 또는 삭제) 발생 시 트리거의 활성화를 의미합니다.

트리거 실행 시점 지정(BEFORE, AFTER 및 INSTEAD OF절)

*트리거 활성화 시간*은 트리거 이벤트와 관련하여 트리거가 활성화되어야 하는 시점을 지정합니다.

BEFORE, AFTER 또는 INSTEAD OF와 같은 세 가지 활성화 시간을 지정할 수 있습니다.

- 활성화 시간이 BEFORE이면 트리거 이벤트가 실행되기 전에 영향을 받은 행 세트의 각 행에 대해 트리거 조치가 활성화됩니다. 그러므로 각 행에 대해 BEFORE 트리거의 실행이 완료된 후에만 주제 테이블이 수정됩니다. BEFORE 트리거에는 FOR EACH ROW 트리거 수준이 있어야 합니다.
- 활성화 시간이 AFTER이면 트리거 수준에 따라 영향을 받은 행 세트의 각 행 또는 명령문에 대해 트리거 조치가 활성화됩니다. 이는 참조 제한조건 조치를 비롯하여 트리거 이벤트가 완료된 후 및 데이터베이스 관리 프로그램이 모든 제한조건을 점검한 후 발생합니다. AFTER 트리거에는 FOR EACH ROW 또는 FOR EACH STATEMENT 트리거 수준이 있을 수 있습니다.

  예를 들어 다음 트리거의 활성화 시간은 employee에 대한 INSERT 조작 이후입니다.

  ```
  CREATE TRIGGER NEW_HIRE
    AFTER INSERT ON EMPLOYEE
    FOR EACH ROW
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
  ```

- 활성화 시간이 INSTEAD이면 트리거 이벤트를 실행하는 대신 영향을 받은 행 세트의 각 행에 대해 트리거 조치가 활성화됩니다. INSTEAD OF 트리거에는 FOR EACH ROW 트리거 수준이 있어야 하고 주제 테이블은 뷰여야 합니다. 그 외 다른 트리거는 주제 테이블을 뷰로 사용할 수 없습니다.

다음 다이어그램은 BEFORE 및 AFTER 트리거의 실행 모델에 대해 설명합니다.

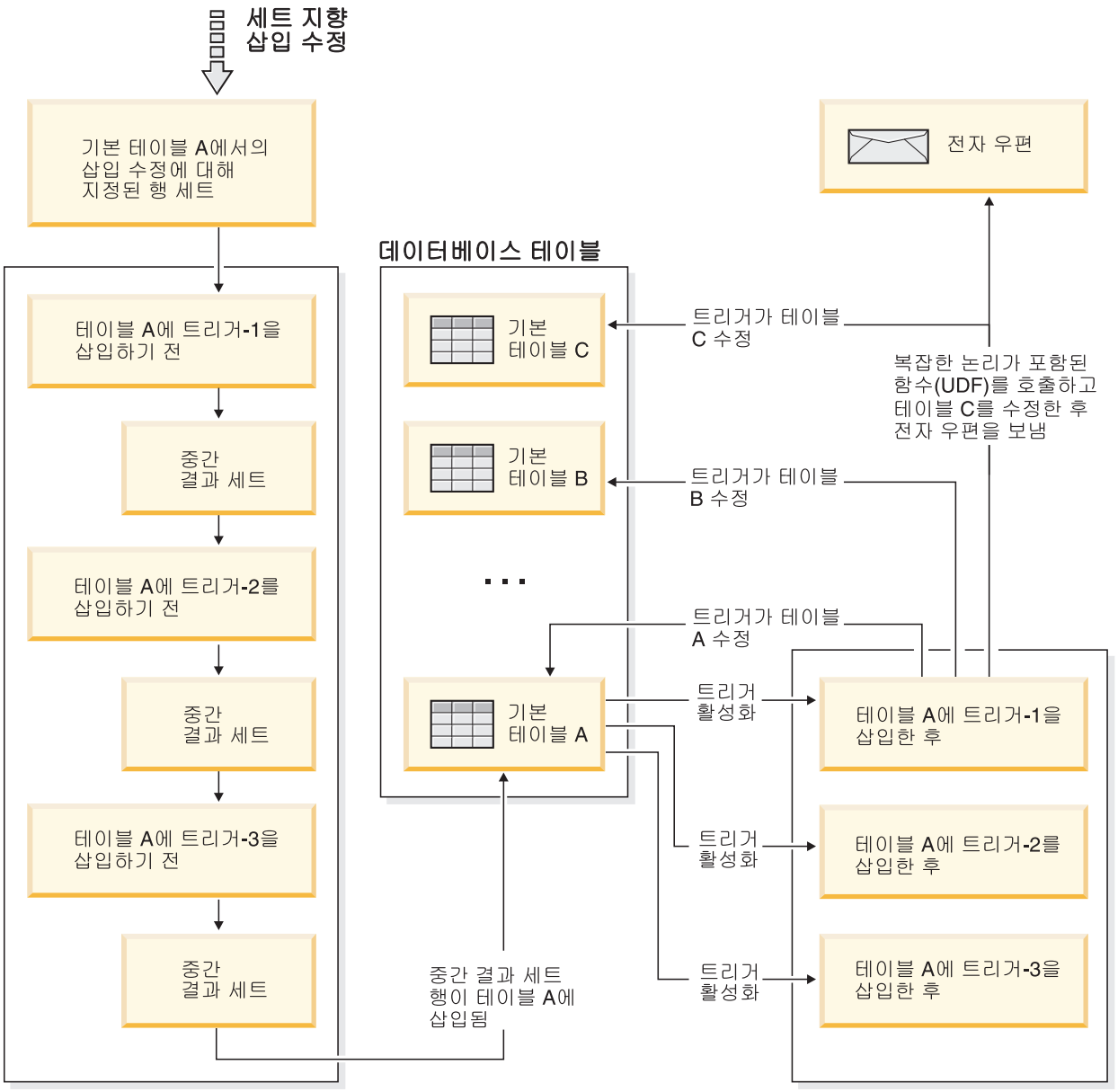


그림 45. 트리거의 실행 모델

BEFORE 및 AFTER 트리거가 지정된 테이블과 이러한 트리거와 연관된 수정 이벤트의 경우 모든 BEFORE 트리거가 먼저 활성화됩니다. 지정된 이벤트에 대해 먼저 활성화된 BEFORE 트리거는 조작의 대상이 되는 행 세트에 대해 작동되고 해당 트리거의 논리가 규정하는 세트에 대해 갱신 수정을 수행합니다. BEFORE 트리거의 출력은 다음 BEFORE 트리거에 의한 입력으로 승인됩니다. 이벤트에 의해 활성화된 BEFORE 트리거가 모두 실행되면 중간 결과 세트 즉, 트리거 이벤트 조작의 대상인 행에 대한 BEFORE 트리거 수정의 결과가 테이블에 적용됩니다. 그러면 해당 이벤트와 연관된 각 AFTER 트리거가 실행됩니다. AFTER 트리거는 동일한 테이블 또는 다른 테이블을 수정하거나 데이터베이스에 대한 외부 조치를 수행할 수 있습니다.

트리거의 활성화 시간이 다른 것은 트리거의 용도가 다름을 나타냅니다. 기본적으로 BEFORE 트리거는 데이터베이스 관리 프로그램 시스템의 제한 서브시스템에 대한 확장입니다. 따라서 이러한 트리거를 사용하여 다음을 수행할 수 있습니다.

- 입력 데이터의 유효성 확인 수행
- 새로 삽입된 행에 대한 값을 자동으로 생성
- 상호 참조를 위해 다른 테이블에서 읽어오기

BEFORE 트리거는 트리거 이벤트가 데이터베이스에 적용되기 전에 활성화되므로 데이터베이스 추가 수정에 사용되지 않습니다. 따라서 이러한 트리거는 무결성 제한조건을 점검하기 전에 활성화됩니다.

반대로 특정 이벤트가 발생할 때마다 데이터베이스에서 실행되는 응용프로그램 논리의 모듈로 AFTER 트리거를 볼 수 있습니다. AFTER 트리거는 항상 데이터베이스를 응용프로그램의 일부로 일관성 있는 상태에서 봅니다. 이러한 트리거는 무결성 제한조건 유효성 확인 후 실행됩니다. 따라서 이러한 트리거를 사용하여 응용프로그램도 수행할 수 있는 대부분의 조작을 수행할 수 있습니다. 예를 들어,

- 데이터베이스에서 후속 수정 조작 수행
- 데이터베이스 외부에서 조치 수행(예: 경보 지원). 트리거가 롤백되면 데이터베이스 외부에서 수행된 조치는 롤백되지 않습니다.

반대로 INSTEAD OF 트리거가 정의된 뷰의 역조작에 대한 설명처럼 INSTEAD OF 트리거를 볼 수 있습니다. 예를 들어 뷰의 선택 목록에 테이블에 대한 표현식이 포함되어 있는 경우 INSTEAD OF INSERT 트리거의 본문에 있는 INSERT문에는 역방향 표현식이 포함될 수 있습니다.

BEFORE, AFTER 및 INSTEAD OF 트리거의 특성이 다르므로 BEFORE 및 AFTER, INSTEAD OF 트리거의 트리거 조치를 정의하는 데 다른 SQL 조작을 사용할 수 있습니다. 예를 들어 트리거 조치가 무결성 제한조건을 위반하지 않음을 보장할 수 없으므로 BEFORE 트리거에서는 갱신 조작이 허용되지 않습니다. 이와 유사하게 BEFORE, AFTER 및 INSTEAD OF 트리거에서는 다른 트리거 수준이 지원됩니다.

모든 트리거의 트리거 SQL문은 동적 복합 명령문일 수 있습니다. 그러나 BEFORE 트리거에는 몇 가지 제한사항이 있어 다음 SQL문을 포함할 수 없습니다.

- UPDATE
- DELETE
- INSERT
- MERGE

트리거 조치가 실행되는 시점에 대한 조건 정의(WHEN절)

트리거가 활성화되면 연관된 트리거 조치가 실행됩니다. 모든 트리거에는 정확히 하나의 트리거 조치가 있습니다. 이러한 조치에는 선택적 *트리거 조치 조건* 즉, WHEN절과 *트리거 명령문 세트*라는 두 개의 구성요소가 있습니다.

*트리거 조치 조건*은 트리거 조치 내에서 명령문 실행을 위해 참으로 평가되어야 하는 검색 조건을 지정하는 트리거 조치의 선택적 절입니다. WHEN절이 생략되면 트리거 조치 내의 명령문이 항상 실행됩니다.

트리거가 FOR EACH ROW 트리거이면 트리거 조치 조건은 각 행에 대해 한 번 평가되고 트리거가 FOR EACH STATEMENT 트리거이면 명령문에 대해 한 번 평가됩니다.

이러한 절은 통해 트리거 대신 활성화된 조치를 보다 잘 조정할 수 있도록 돕는 제어 기능을 제공합니다. WHEN절이 유용한 경우의 예는 수신 값이 특정 범위 내에 포함되거나 해당 범위를 벗어나는 경우에만 트리거된 조치가 활성화되는 데이터 종속 규칙을 적용하는 경우입니다.

트리거가 활성화되면 연관된 트리거 조치가 실행됩니다. 모든 트리거에는 다음과 같은 두 개의 구성요소가 있는 정확히 하나의 트리거 조치가 있습니다.

트리거 조치 조건은 트리거 명령문 세트가 트리거 조치가 실행 중인 행 또는 명령문에 대해 수행되는지 여부를 정의합니다. 트리거 명령문 세트는 발생한 이벤트의 결과로 데이터베이스에서 트리거에 의해 수행되는 조치 세트를 정의합니다.

예를 들어 다음 트리거 조치는 on_hand 컬럼의 값이 max_stocked 컬럼 값의 10% 미만인 행에 대해서만 트리거 명령문 세트가 활성화되어야 함을 지정합니다. 이러한 경우 트리거 명령문 세트는 issue_ship_request 함수의 호출입니다.

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
  FOR EACH ROW

  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
  BEGIN ATOMIC
    VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                              N_ROW.ON_HAND,
                              N_ROW.PARTNO));
  END
```

트리거 명령문 세트는 트리거 활성화로 인해 발생한 실제 조치를 수행합니다. 일부 SQL 조작은 트리거에서 유용하지 않습니다. 트리거 활성화 시간이 BEFORE 또는 AFTER인지 여부에 따라 다른 종류의 조작이 트리거 명령문으로 적절할 수 있습니다.

대부분의 경우 트리거 명령문이 음수의 리턴 코드를 리턴하면 모든 트리거 및 참조 제한조건 조치와 함께 트리거링 명령문이 롤백됩니다. 실패한 트리거 명령문의 트리거 이름, SQLCODE, SQLSTATE 및 여러 토큰이 오류 메시지에 리턴됩니다.

트리거에서 지원되는 SQL PL문

모든 트리거의 트리거 SQL문은 동적 복합 명령문일 수 있습니다.

즉, 트리거 SQL문에는 다음 요소가 하나 이상 포함될 수 있습니다.

- CALL문
- DECLARE variable문
- SET variable문
- WHILE 루프
- FOR 루프
- IF문
- SIGNAL문
- ITERATE문
- LEAVE문
- GET DIGNOSTIC문
- fullselect

그러나 AFTER 및 INSTEAD OF 트리거에만 다음 SQL문이 하나 이상 포함될 수 있습니다.

- UPDATE문
- DELETE문
- INSERT문
- MERGE문

전이 변수를 사용하여 트리거에서 이전 및 새 컬럼 값에 액세스

FOR EACH ROW 트리거를 구현하는 경우 트리거가 현재 실행 중인 영향을 받는 행 세트에 있는 행의 컬럼 값을 참조해야 합니다. 데이터베이스에 있는 테이블(주제 테이블 포함)의 컬럼을 참조하기 위해 일반 SELECT문을 사용할 수 있습니다.

FOR EACH ROW 트리거는 CREATE TRIGGER문의 REFERENCING절에서 지정할 수 있는 두 개의 전이 변수를 사용하여 현재 해당 트리거가 실행 중인 행의 컬럼을 참조할 수 있습니다. 이러한 컬럼에는 correlation-name과 함께 OLD 및 NEW로 지정된 두 가지 종류의 전이 변수가 있습니다. 이러한 변수의 시맨틱은 다음과 같습니다.

OLD AS correlation-name

행의 원래 상태(즉, 트리거 조치가 데이터베이스에 적용되기 전)를 캡처하는 상관 이름을 지정합니다.

NEW AS correlation-name

트리거 조치가 데이터베이스에 적용되는 경우 데이터베이스의 행을 갱신하는 데 사용되거나 사용된 값을 캡처하는 상관 이름을 지정합니다.

다음 예를 고려하십시오.

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
  FOR EACH ROW
  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED
  AND N_ROW.ORDER_PENDING = 'N')
  BEGIN ATOMIC
    VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                              N_ROW.ON_HAND,
                              N_ROW.PARTNO));
    UPDATE PARTS SET PARTS.ORDER_PENDING = 'Y'
    WHERE PARTS.PARTNO = N_ROW.PARTNO;
       END
```

위에서 지정된 OLD 및 NEW 전이 변수의 정의에 따라 모든 트리거에 대해 일부 전이 변수를 정의하지 못할 수 있습니다. 다음과 같이 트리거 이벤트의 종류에 따라 전이 변수를 정의할 수 있습니다.

UPDATE

UPDATE 트리거는 OLD 및 NEW 전이 변수를 모두 참조할 수 있습니다.

INSERT

INSERT 조작 활성화 이전에는 데이터베이스에 영향을 받는 행이 존재하지 않으므로 INSERT 트리거는 NEW 전이 변수만 참조할 수 있습니다. 즉, 트리거 조치가 데이터베이스에 적용되기 전에 이전 값을 정의하는 행의 원래 상태가 존재하지 않습니다.

DELETE

삭제 조작에 지정된 새 값이 없으므로 DELETE 트리거는 OLD 전이 변수만 참조할 수 있습니다.

주: 전이 변수는 FOR EACH ROW 트리거에 대해서만 지정될 수 있습니다. FOR EACH STATEMENT 트리거에서 전이 변수가 참조하는 영향을 받는 행 세트의 여러 행을 지정하는 데 전이 변수 참조가 충분하지 않습니다. 대신 CREATE TRIGGER절의 OLD TABLE 및 NEW TABLE절을 사용하여 새 행 및 이전 행 세트를 참조하십시오. 이러한 절에 대한 자세한 정보는 CREATE TRIGGER문을 참조하십시오.

전이 테이블을 사용하여 이전 및 새 테이블 결과 세트 참조

FOR EACH ROW 및 FOR EACH STATEMENT 트리거 모두에서 영향을 받는 행 세트 전체를 참조해야 할 수 있습니다. 예를 들어 트리거 내용이 영향을 받는 행 세트에 집계를 적용해야 하는 경우가 이에 해당합니다(예: 일부 컬럼 값의 MAX, MIN 또는 AVG).

트리거는 REATE TRIGGER문의 REFERENCING절에서 지정할 수 있는 두 개의 전이 테이블을 사용하여 영향을 받는 행 세트를 참조할 수 있습니다. 전이 변수와 같이 전이 테이블에도 테이블 이름과 함께 OLD_TABLE 및 NEW_TABLE로 지정된 두 개의 테이블 종류가 있습니다. 이러한 테이블의 시맨틱은 다음과 같습니다.

OLD_TABLE AS table-name

영향을 받는 행 세트의 원래 상태(즉, 트리거링 SQL 조작이 데이터베이스에 적용되기 전)를 캡처하는 테이블 이름을 지정합니다.

NEW_TABLE AS table-name

트리거 조치가 데이터베이스에 적용되는 경우 데이터베이스의 행을 갱신하는 데 사용되거나 사용된 값을 캡처하는 테이블의 이름을 지정합니다.

예를 들어,

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW_TABLE AS N_TABLE
  NEW AS N_ROW
  FOR EACH ROW
  WHEN ((SELECT AVG (ON_HAND) FROM N_TABLE) > 35)
  BEGIN ATOMIC
    VALUES(INFORM_SUPERVISOR(N_ROW.PARTNO,
                             N_ROW.MAX_STOCKED,
                             N_ROW.ON_HAND));
      END
```

NEW_TABLE에는 FOR EACH ROW 트리거에 대해서도 항상 갱신된 행의 전체 세트가 있습니다. 트리거가 정의된 테이블에 대해 트리거가 작동하면 NEW_TABLE에는 트리거를 활성화한 명령문에서 변경된 행이 포함됩니다. 그러나 NEW_TABLE 테이블에는 트리거 내의 명령문에 의해 변경된 행은 포함되지 않습니다. 이는 해당 명령문이 트리거를 별도로 활성화하기 때문입니다.

전이 테이블은 읽기 전용입니다 트리거 이벤트에 대해 정의할 수 있는 트리거 변수 종류를 정의하는 규칙과 동일한 규칙이 전이 테이블에도 적용됩니다.

UPDATE

UPDATE 트리거는 OLD_TABLE 및 NEW_TABLE 전이 테이블을 모두 참조할 수 있습니다.

INSERT

INSERT 조작 활성화 이전에는 데이터베이스에 영향 받는 행이 존재하지 않

으므로 INSERT 트리거는 NEW_TABLE 전이 테이블만 참조할 수 있습니다. 즉, 트리거 조치가 데이터베이스에 적용되기 전에 이전 값을 정의하는 행의 원래 상태가 존재하지 않습니다.

DELETE

삭제 조작에 지정된 새 값이 없으므로 DELETE 트리거는 OLD_TABLE 전이 테이블만 참조할 수 있습니다.

주: 전이 테이블을 AFTER 트리거의 트리거 수준 FOR EACH ROW 및 FOR EACH STATEMENT 둘 다에 대해 정의할 수 있는지 살펴보아야 합니다.

OLD_TABLE 및 NEW_TABLE table-name은 트리거 본문입니다. 이러한 범위에서 이 이름은 스키마에 존재하는 규정되지 않은 동일한 *table-name*을 비롯하여 다른 모든 테이블 이름보다 우선합니다. 따라서 예를 들어 OLD_TABLE 또는 NEW_TABLE table-name이 X이면 트리거 작성자의 스키마에 X라는 테이블이 있더라도 X(즉, 규정되지 않은 X)에 대한 참조는 항상 전이 테이블을 참조합니다. 이러한 경우 사용자는 스키마에서 테이블 X를 참조하기 위해 완전한 이름을 사용해야 합니다.

트리거 작성

트리거는 지정한 테이블 또는 유형이 지정된 테이블에서 INSERT, UPDATE 또는 DELETE절과 함께 실행되거나 이러한 절로 트리거된 조치 세트를 정의합니다.

트리거를 사용하여 다음을 수행합니다.

- 입력 데이터 유효성 확인
- 새로 삽입된 행에 대한 값 생성
- 상호 참조를 위해 다른 테이블에서 읽어오기
- 감사 추적을 위해 다른 테이블에 쓰기

트리거를 사용하여 무결성 또는 비즈니스 규칙의 일반적인 형태를 지원할 수 있습니다. 예를 들어 트리거는 주문이 승인되거나 요약 데이터 테이블을 갱신하기 전에 고객의 신용 한도를 확인할 수 있습니다.

이점:

- 빠른 응용프로그램 개발: 트리거가 데이터베이스에 저장되므로 모든 응용프로그램에서 트리거가 수행하는 조치를 코딩할 필요가 없습니다.
- 한층 쉬운 유지보수: 트리거가 정의되면 트리거가 작성된 테이블에 액세스할 때 해당 트리거가 자동으로 호출됩니다.
- 비즈니스 방침의 전역 적용: 비즈니스 방침이 변경되면 각 응용프로그램이 아니라 트리거만 변경하면 됩니다.

제한사항:

- 별칭으로 트리거를 사용할 수 없습니다.
- 트리거가 BEFORE 트리거이면 트리거된 조치에서 지정한 컬럼 이름은 ID 컬럼 이외의 생성된 컬럼일 수 없습니다. 즉, 생성된 ID 값이 BEFORE 트리거에 표시됩니다.

자동 트리거를 작성하는 경우 명령문 끝 문자에 주의해야 합니다. 명령행 처리기에서는 디폴트로 『;』을 명령문 끝 표시문자로 간주합니다. 『;』 이외의 문자를 사용할 수 있도록 스크립트에서 명령문 끝 문자를 수동으로 편집하여 ATOMIC 트리거를 작성해야 합니다. 예를 들어 『;』은 『#』과 같은 다른 특수 문자로 교체할 수 있습니다. 또한 CREATE TRIGGER DDL 앞에 다음 구문을 입력할 수도 있습니다.

```
--#SET TERMINATOR @
```

CLP에서 종단자를 변경하려면 다음 구문을 뒤에 배치합니다.

```
--#SET TERMINATOR
```

명령행에서 트리거를 작성하려면 다음과 같이 입력하십시오.

```
db2 -td <delimiter> -vf <script>
```

여기서 <delimiter>는 대체 명령문 끝 문자이고 <script>는 새로운 <delimiter>가 포함된 수정된 스크립트입니다.

명령행에서 트리거를 작성하려면 다음과 같이 입력하십시오.

```
CREATE TRIGGER <name>
   <action> ON <table_name>
   <operation>
   <triggered_action>
```

다음 명령문은 신입 사원이 고용될 때마다 즉, 새로운 행이 EMPLOYEE 테이블에 추가될 때마다 COMPANY_STATS 테이블의 사원 수(NBEMP) 컬럼에 1씩 추가하여 사원의 수를 늘리는 트리거를 작성합니다.

```
CREATE TRIGGER NEW_HIRED
   AFTER INSERT ON EMPLOYEE
   FOR EACH ROW
   UPDATE COMPANY_STATS SET NBEMP = NBEMP+1;
```

트리거 내용에는 INSERT, 검색된 UPDATE, 검색된 DELETE, fullselect, SET Variable 및 SIGNAL SQLSTATE문 중 하나 이상이 포함될 수 있습니다. 트리거는 자신이 참조하는 INSERT, UPDATE 또는 DELETE문 앞 또는 뒤에서 활성화될 수 있습니다.

트리거 수정 및 삭제

트리거는 수정할 수 없습니다. 트리거는 삭제한 다음 필요한 새 정의에 따라 다시 작성해야 합니다.

트리거 종속성

- 일부 다른 오브젝트에 대한 트리거의 모든 종속성은 SYSCAT.TRIGDEP 시스템 카탈로그 뷰에 기록됩니다. 트리거는 여러 오브젝트에 대해 종속될 수 있습니다.
- 트리거가 종속된 오브젝트가 삭제되면 해당 트리거는 작동 불능 상태가 되지만 해당 정의는 시스템 카탈로그 뷰에 보유됩니다. 이러한 트리거를 다시 유효하게 만들려면 시스템 카탈로그 뷰에서 해당 정의를 검색한 다음 새 CREATE TRIGGER문을 제출해야 합니다.
- 트리거가 삭제되면 해당 설명이 SYSCAT.TRIGGERS 시스템 카탈로그 뷰에서 삭제되고 모든 종속성이 SYSCAT.TRIGDEP 시스템 카탈로그 뷰에서 삭제됩니다. 트리거에 대한 UPDATE, INSERT 또는 DELETE 종속성이 포함된 모든 패키지는 무효화됩니다.
- 뷰가 트리거에 종속되어 있고 작동 불능 상태가 되면 해당 트리거도 작동 불능 상태가 됩니다. 작동 불능 상태인 트리거에 종속된 모든 패키지는 무효화됩니다.

DROP TRIGGER문을 사용하여 트리거 오브젝트를 삭제할 수 있습니다. 그러나 이러한 절차는 다음과 같이 종속 패키지를 유효하지 않은 상태로 만듭니다.

- 명시적인 컬럼 목록없이 갱신 트리거를 작성하면, 목표 테이블에서 갱신을 수행할 수 있는 패키지가 유효하지 않게 됩니다.
- 컬럼 목록과 함께 갱신 트리거를 삭제하면, CREATE TRIGGER문의 column-name 목록에 있는 최소한 하나의 컬럼에서 패키지가 갱신을 수행할 수 있는 경우 목표 테이블에서 갱신을 수행할 수 없는 패키지만 유효하지 않게 됩니다.
- insert 트리거가 삭제되면, 목표 테이블에서 삽입을 수행할 수 있는 패키지가 유효하지 않게 됩니다.
- 삭제 트리거가 삭제되면, 목표 테이블에서 삭제를 수행할 수 있는 패키지가 유효하지 않게 됩니다.

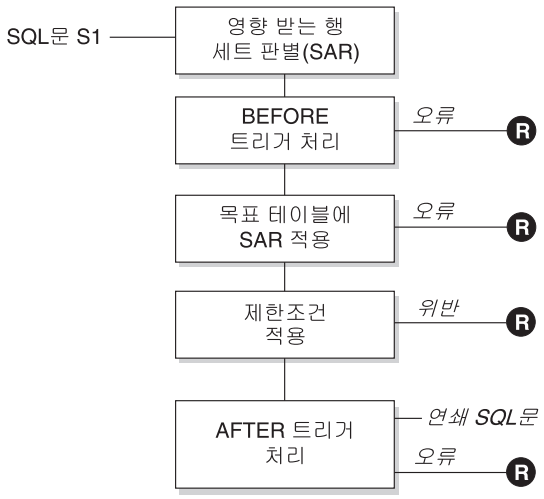
응용프로그램이 명시적으로 바인딩되거나 리바인딩될 때까지 또는 실행된 후 데이터베이스 관리 프로그램에 의해 자동으로 리바인딩될 때까지 패키지는 유효하지 않은 상태로 유지됩니다.

트리거 및 트리거 사용 예

트리거 및 참조 제한조건 사이의 상호 작용 예

갱신 조작으로 인해 참조 제한조건 및 점검 제한조건과 트리거의 상호 작용이 발생할 수 있습니다.

335 페이지의 그림 37 및 관련 설명은 데이터베이스에서 데이터를 갱신하는 명령문에 대해 수행되는 처리를 나타냅니다.



R = S1 이전에 대한 롤백 변경사항

그림 46. 연관된 트리거 및 제한조건을 사용하여 명령문 처리

335 페이지의 그림 37에서는 테이블을 갱신하는 명령문 처리의 일반적인 순서에 대해 설명합니다. 여기서는 테이블에 연쇄적으로 적용되는 BEFORE 트리거, 참조 제한조건, 점검 제한조건 및 AFTER 트리거가 포함되어 있는 상황을 가정합니다. 다음은 335 페이지의 그림 37에 있는 상자 및 기타 항목에 대한 설명입니다.

- 명령문 S_1

  프로세스를 시작하는 DELETE, INSERT 또는 UPDATE 문입니다. 명령문 S_1은 이 설명에서 *주제 테이블*이라고 하는 테이블(또는 일부 테이블에 대한 갱신 가능 뷰)을 식별합니다.

- 영향을 받는 행 세트 판별

  이 단계에서는 AFTER 트리거에서 연쇄 명령문과 CASCADE 및 SET NULL에 대한 참조 제한조건 삭제 규칙에 대해 반복되는 프로세스가 시작됩니다.

  이 단계의 목적은 명령문의 *영향을 받는 행 세트*를 판별하는 것입니다. 포함된 행 세트는 다음 명령문에 따라 달라집니다.

- DELETE의 경우: 명령문의 검색 조건을 충족하는 모든 행(또는 위치가 지정된 DELETE의 현재 행)
- INSERT의 경우: VALUES절 또는 fullselect에 의해 식별되는 행
- UPDATE의 경우: 검색 조건을 충족하는 모든 행(또는 위치가 지정된 UPDATE의 현재 행)

영향을 받는 행이 비어 있으면 BEFORE 트리거, 주제 테이블에 적용되는 변경사항 또는 명령문 프로세스에 대한 제한조건이 없습니다.

- BEFORE 트리거 처리

  모든 BEFORE 트리거는 작성 순서에 따라 오름차순으로 처리됩니다. 각 BEFORE 트리거는 영향 받은 행 세트의 각 행에 대해 한 번씩 트리거 조치를 처리합니다.

  지금까지 원래 명령문인 S_1의 결과로 변경된 모든 사항이 롤백되는 경우 트리거 조치 처리 중 오류가 발생할 수 있습니다.

  BEFORE 트리거가 없거나 영향을 받는 행 세트가 비어 있는 경우 이 단계를 건너뜁니다.

- 주제 테이블에 영향을 받는 행 세트 적용

  영향을 받는 행 세트를 사용하여 실제 삭제, 삽입 또는 갱신이 데이터베이스의 주제 테이블에 적용됩니다.

  지금까지 원래 명령문인 S_1의 결과로 변경된 모든 사항이 롤백되는 경우 영향을 받는 행 세트를 적용할 때(예: 고유한 인덱스가 존재하는 중복 키와 함께 행을 삽입하려는 경우) 트리거 조치 처리 중 오류가 발생할 수 있습니다.

- 제한조건 적용

  영향을 받는 행 세트가 비어 있는 경우 주제 테이블과 연관된 제한 조건이 적용됩니다. 이러한 제한조건에는 고유 제한조건, 고유 인덱스, 참조 제한조건, 점검 제한조건 및 뷰에서 WITH CHECK OPTION 관련 점검이 있습니다. CASCADE 또는 SET NULL 삭제 규칙이 포함된 참조 제한조건으로 인해 추가 트리거가 활성화될 수 있습니다.

  모든 제한조건 또는 WITH CHECK OPTION 위반으로 인해 오류가 발생하고 지금까지 원래 명령문인 S_1의 결과로 변경된 모든 사항이 롤백됩니다.

- AFTER 트리거 처리

  S_1로 활성화된 모든 AFTER 트리거는 작성 순서에 따라 오름차순으로 처리됩니다.

  영향을 받는 행 세트가 비어 있더라도 FOR EACH STATEMENT 트리거는 트리거 조치를 정확히 한 번만 처리합니다. FOR EACH ROW 트리거는 영향을 받은 행 세트의 각 행에 대해 한 번씩 트리거 조치를 처리합니다.

지금까지 원래 S_1의 결과로 변경된 모든 사항이 롤백되는 경우 트리거 조치 처리 중 오류가 발생할 수 있습니다.

트리거의 트리거 조치에는 DELETE, INSERT 또는 UPDATE문인 트리거 명령문이 포함될 수 있습니다. 이렇게 설명하는 이유는 이러한 각 명령문을 *연쇄 명령문*으로 간주하기 위함입니다.

연쇄 명령문은 AFTER 트리거의 트리거 조치의 일부로 처리되는 DELETE, INSERT 또는 UPDATE문입니다. 이 명령문은 트리거 처리의 연쇄 레벨을 시작합니다. 연쇄 레벨은 트리거 명령문을 새 S_1으로 지정하고 새 여기서 설명한 모든 단계를 반복적으로 수행할 수 있습니다.

각 S_1에서 활성화한 모든 AFTER 트리거의 모든 트리거 명령문이 완료되도록 처리되면 원래 S_1의 처리가 완료됩니다.

- R = 이전 S_1에 대한 변경사항 롤백

  처리 중 발생한 모든 오류(제한조건 위반 포함)로 인해 원래 명령문 S_1의 결과로 직접 또는 간접적으로 변경된 모든 사항이 롤백됩니다. 원래 명령문 S_1이 실행되기 직전에 데이터베이스가 동일한 상태로 롤백됩니다.

트리거를 사용하여 조치 정의의 예

총 책임자가 지난 72시간 동안 3번 이상의 불만이 접수된 고객의 이름을 다른 테이블에 저장하려고 한다고 가정합니다. 또한 고객의 이름이 해당 테이블에 한 번 이상 삽입되면 알림을 받도록 하려고 합니다.

이러한 조치를 정의하려면 다음과 같이 정의합니다.

- UNHAPPY_CUSTOMERS 테이블:

```
CREATE TABLE UNHAPPY_CUSTOMERS (
  NAME           VARCHAR (30),
  EMAIL_ADDRESS  VARCHAR (200),
  INSERTION_DATE DATE)
```

- 다음은 지난 3일간 메시지가 3번 이상 수신된 경우 UNHAPPY_CUSTOMERS에 하나의 행이 자동으로 삽입되는 트리거입니다. NAME 컬럼과 E_MAIL_ADDRESS 컬럼이 포함된 CUSTOMERS 테이블이 있다고 가정합니다.

```
CREATE TRIGGER STORE_UNHAPPY_CUST
  AFTER INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (3 <= (SELECT COUNT(*)
              FROM ELECTRONIC_MAIL
              WHERE SENDER = N.SENDER
                AND SENDING_DATE(MESSAGE) > CURRENT DATE - 3 DAYS)
       )
  BEGIN ATOMIC
    INSERT INTO UNHAPPY_CUSTOMERS
```

```
    VALUES ((SELECT NAME
    FROM CUSTOMERS
    WHERE EMAIL_ADDRESS = N.SENDER), N.SENDER, CURRENT DATE);
       END
```

- 다음은 UNHAPPY_CUSTOMERS에 동일한 고객이 한 번 이상 삽입되는 경우 총 책임자에게 메모를 보내는 트리거입니다. 2자의 문자열이 입력되는 SEND_NOTE 함수가 있다고 가정합니다.

```
CREATE TRIGGER INFORM_GEN_MGR
  AFTER INSERT ON UNHAPPY_CUSTOMERS
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (1 <(SELECT COUNT(*)
            FROM UNHAPPY_CUSTOMERS
            WHERE EMAIL_ADDRESS = N.EMAIL_ADDRESS)
       )
  BEGIN ATOMIC
    VALUES(SEND_NOTE('Check customer:' CONCAT N.NAME,
                     'bigboss@vnet.ibm.com'));
       END
```

트리거를 사용하여 비즈니스 규칙 정의의 예

회사에 고객의 불만을 처리하는 모든 전자 우편은 마케팅 책임자인 Mr. Nelson에게 참조(CC) 목록으로 전달되어야 한다는 규정이 있다고 가정합니다.

이는 규칙이므로 해당 규칙을 다음 중 하나와 같이 제한조건으로 나타낼 수 있습니다. 해당 규칙을 점검하는 CC_LIST UDF가 있다고 가정합니다.

```
ALTER TABLE ELECTRONIC_MAIL ADD
  CHECK (SUBJECT <> 'Customer complaint' OR
         CONTAINS (CC_LIST(MESSAGE), 'nelson@vnet.ibm.com') = 1)
```

그러나 이러한 제한조건으로 인해 참조 목록에 마케팅 책임자가 포함되지 않은 고객 불만 처리 전자 우편이 삽입되지 않습니다. 이는 회사에서 비즈니스 규칙을 통해 의도하는 바가 아닙니다. 이러한 규칙은 마케팅 관리자에게 보고되지 않은 모든 고객 불만 처리 전자 우편을 마케팅 관리자에게 전달하도록 작성되었습니다. 이러한 비즈니스 규칙에는 선언적 제한조건으로는 표현할 수 없는 조치가 필요하므로 해당 규칙은 트리거를 사용하는 경우에만 표현할 수 있습니다. 트리거는 E_MAIL 및 문자열 유형의 매개변수가 포함된 SEND_NOTE 함수가 있다고 가정합니다.

```
CREATE TRIGGER INFORM_MANAGER
  AFTER INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (N.SUBJECT = 'Customer complaint' AND
    CONTAINS (CC_LIST(MESSAGE), 'nelson@vnet.ibm.com') = 0)
  BEGIN ATOMIC
    VALUES(SEND_NOTE(N.MESSAGE, 'nelson@vnet.ibm.com'));
       END
```

트리거를 사용하여 테이블에 대한 조작 방지의 예

배달이 불가능한 전자 우편을 ELECTRONIC_MAIL 테이블에 저장하지 않으려고 한다고 가정합니다. 이렇게 하려면 특정 SQL INSERT문이 실행되지 않도록 해야 합니다.

다음과 같은 두 가지 방법으로 테이블에 전자 우편을 저장하지 않을 수 있습니다.

- 전자 메일의 제목이 *배달되지 않은 메일*이면 항상 오류를 리턴하는 BEFORE 트리거를 정의합니다.

```
CREATE TRIGGER BLOCK_INSERT
  NO CASCADE BEFORE INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (SUBJECT(N.MESSAGE) = 'undelivered mail')
  BEGIN ATOMIC
    SIGNAL SQLSTATE '85101'
      SET MESSAGE_TEXT = ('Attempt to insert undelivered mail');
     END
```

- 점검 제한조건이 새 컬럼 SUBJECT의 값을 *배달되지 않은 메일*과 강제로 다르게 만들도록 정의합니다.

```
ALTER TABLE ELECTRONIC_MAIL
  ADD CONSTRAINT NO_UNDELIVERED
  CHECK (SUBJECT <> 'undelivered mail')
```

제 15 장 시퀀스

*시퀀스*는 예를 들어, 수표 번호와 같은 값을 자동으로 생성할 수 있도록 하는 데이터베이스 오브젝트입니다. 시퀀스는 고유 키 값을 생성하는 태스크에 가장 적합합니다. 응용프로그램에서 시퀀스를 사용하여 번호를 추적하는 데 사용되는 컬럼 값으로 인해 발생할 수 있는 동시성 및 성능 문제가 일어나지 않게 할 수 있습니다. 데이터베이스 외부에서 작성되는 번호에 비해 시퀀스가 갖는 장점은 데이터베이스 서버가 생성된 숫자의 트랙을 유지한다는 것입니다. 손상되어 재시작해도 숫자가 중복해서 생성되지 않습니다.

생성되는 시퀀스 번호에는 다음과 같은 등록 정보가 있습니다.

- 값은 스케일이 영(0)인 정확한 숫자 데이터 유형입니다. 이러한 데이터 유형에는 SMALLINT, BIGINT, INTEGER 및 DECIMAL이 있습니다.
- 연속되는 값은 지정된 정수 증분 만큼 다릅니다. 디폴트 증분값은 1입니다.
- 카운터 값은 복구 가능합니다. 카운터 값은 복구가 필요한 경우 로그에서 재생성됩니다.
- 값을 캐시하여 성능을 향상시킬 수 있습니다. 캐시에 값을 사전 할당하고 저장하면 시퀀스에 사용할 값이 생성될 때 로그에 대한 동기 입출력이 줄어듭니다. 시스템 오류가 발생하는 경우 사용되지 않은 모든 캐시된 값이 유실됩니다. CACHE에 지정되는 값이 유실될 수 있는 최대 시퀀스 값의 수입니다.

시퀀스에서 사용할 수 있는 표현식이 두 가지 있습니다.

- **NEXT VALUE 표현식:** 지정된 시퀀스의 다음 값을 리턴합니다. NEXT VALUE 표현식이 시퀀스 이름을 지정하면 새 시퀀스 번호가 생성됩니다. 그러나 쿼리 내에 동일한 시퀀스 이름을 지정하는 NEXT VALUE 표현식의 다중 인스턴스가 있는 경우 각 결과 행에서 시퀀스의 카운터가 한 번만 증분되고 NEXT VALUE의 모든 인스턴스는 각 결과 행에 동일한 값을 리턴합니다.
- **PREVIOUS VALUE 표현식:** 현재 응용프로그램 프로세스에서 이전 명령문에 지정된 시퀀스에 사용할 최근에 생성된 값을 리턴합니다. 즉, 지정된 연결에서 PREVIOUS VALUE는 다른 연결이 NEXT VALUE를 호출해도 일정하게 유지됩니다.

이 표현식에 대한 전체 세부사항 및 예는 『시퀀스 참조』(*SQL 참조서, 볼륨 1*)를 참조하십시오.

시퀀스 설계

시퀀스를 설계하는 경우 ID 컬럼 및 시퀀스 간의 다른점과, 어느 것이 사용자 환경에 더욱 적합한지 고려해야 합니다. 시퀀스를 사용하도록 결정하면 사용 가능한 옵션 및 매개변수에 대해 잘 알고 있어야 합니다.

시퀀스를 설계하기 전에 406 페이지의 『시퀀스와 ID 컬럼 비교』의 내용을 참조하십시오.

시퀀스에는 간단한 설정 및 작성 이외에도 더욱 유연하게 값을 생성할 수 있는 다음과 같은 다양한 추가 옵션이 있습니다.

- 다양한 데이터 유형에서 선택(SMALLINT, INTEGER, BIGINT, DECIMAL)
- 시작 값 변경(START WITH)
- 값 증가 또는 감소를 비롯하여 시퀀스 증분 변경(INCREMENT BY)
- 시퀀스 번호가 시작 및 중지되는 최소값 및 최대값 설정(MINVALUE/MAXVALUE)
- 시퀀스가 다시 시작하거나 순환을 허용하지 않도록 값 래핑 허용(CYCLE/NO CYCLE)
- 성능 향상을 위해 시퀀스 값 캐싱 허용 또는 캐싱 허용하지 않음(CACHE/NO CACHE)

시퀀스가 생성되면 이러한 여러 값을 변경할 수 있습니다. 예를 들어 요일에 따라 다른 시작 값을 설정할 수 있습니다. 시퀀스 사용의 다른 실례는 은행 수표의 생성 및 처리입니다. 은행 수표 번호 시퀀스는 매우 중요합니다. 따라서 시퀀스 번호의 일괄처리에 문제가 생기면 심각한 결과가 발생합니다.

또한 성능 향상을 위해 CACHE 옵션에 대해 알고 있고 이러한 옵션을 사용해야 합니다. 이 옵션은 다른 시퀀스 세트를 생성하기 위해 카탈로그로 돌아가기 전에 시스템에서 생성해야 하는 시퀀스 값의 개수를 데이터베이스 관리 프로그램에 알려줍니다. 지정되지 않은 경우 디폴트 CACHE 값은 20입니다. 첫 번째 시퀀스 값이 요청되면 데이터베이스 관리 프로그램은 디폴트값을 예로 사용하여 메모리에 순차적인 20개의 값을 자동으로 생성합니다. 새 시퀀스 번호가 필요한 경우 값의 이러한 메모리 캐시를 사용하여 다음 값을 리턴합니다. 이러한 값 캐시가 사용되면 데이터베이스 관리 프로그램은 다음 20개의 값(21, 22, ..., 40)을 생성합니다.

데이터베이스 관리 프로그램은 시퀀스 번호의 캐싱을 구현하므로 다음 값을 얻기 위해 카탈로그 테이블로 계속해서 돌아갈 필요가 없습니다. 이렇게 하면 시퀀스 번호 검색과 관련된 오버헤드가 줄어듭니다. 그러나 시스템 오류가 발생하거나 시스템이 종료되면 시퀀스에 갭이 발생할 수 있습니다. 예를 들어 시퀀스 캐시를 100으로 설정하려고 하면 데이터베이스 관리 프로그램은 이러한 번호의 값 100개를 캐시하고 값의 다음 시퀀스가 201에서 시작되어야 함을 나타내도록 시스템 카탈로그를 설정합니다. 데이터베이스

가 종료되면 다음 시퀀스 번호 세트가 201에서 시작됩니다. 101 ~ 200 사이에서 생성되는 번호가 사용되지 않으면 시퀀스 세트에서 손실됩니다. 응용프로그램에서 생성된 값의 갭을 허용할 수 없으면 한층 높은 시스템 오버헤드가 발생하더라도 캐싱 값을 NO CACHE로 설정해야 합니다.

사용 가능한 모든 옵션 및 연관된 값에 대한 자세한 정보는 CREATE SEQUENCE문을 참조하십시오.

시퀀스 동작 관리

사용자 응용프로그램의 필요에 부합하도록 시퀀스 동작을 조정할 수 있습니다. 새 시퀀스를 작성하기 위해 CREATE SEQUENCE문을 실행하는 경우 및 기존 시퀀스에 적용할 ALTER SEQUENCE문을 실행하는 경우 시퀀스 속성을 변경합니다.

지정할 수 있는 시퀀스 속성에는 다음이 있습니다.

데이터 유형

CREATE SEQUENCE문의 AS절은 시퀀스의 숫자 데이터 유형을 지정합니다. 데이터 유형은 가능한 최소 및 최대 시퀀스 값을 판별합니다. 데이터 유형에 대한 최소 및 최대값은 *SQL 참조서*에 나열되어 있습니다. 시퀀스의 데이터 유형을 변경할 수 없습니다. 대신에 DROP SEQUENCE문을 실행하여 시퀀스를 삭제(drop)하고 새 데이터 유형으로 CREATE SEQUENCE문을 실행해야 합니다.

시작 값

CREATE SEQUENCE문의 START WITH절은 시퀀스의 초기값을 설정합니다. ALTER SEQUENCE문의 RESTART WITH절은 시퀀스 값을 지정된 값으로 재설정합니다.

최소값 MINVALUE절은 시퀀스의 최소값을 설정합니다.

최대값 MAXVALUE절은 시퀀스의 최대값을 설정합니다.

증분값 INCREMENT BY절은 각 NEXT VALUE 표현식이 현재 시퀀스 값에 추가하는 값을 설정합니다. 시퀀스의 값을 감소시키려면 음수를 지정하십시오.

시퀀스 순환

CYCLE절은 최대값 또는 최소값에 접근하는 시퀀스 값이 다음 NEXT VALUE 표현식에 대한 각각의 최소값 또는 최대값을 생성하도록 합니다.

주: 고유한 번호가 필요하지 않은 경우 또는 시퀀스가 순환한 후 이전 시퀀스 값을 더 이상 사용 중이 아닌 것이 확실한 경우에만 CYCLE을 사용해야 합니다.

예를 들어, 최소값 0으로 시작하고 최대값이 1000이며 각 NEXT VALUE 표현식에서 2씩 증분되고 최대값에 도달하면 최소값을 리턴하는 id_values라는 시퀀스를 작성하려면 다음 명령문을 실행하십시오.

```
CREATE SEQUENCE id_values
  START WITH 0
  INCREMENT BY 2
  MAXVALUE 1000
  CYCLE
```

응용프로그램 성능 및 시퀀스

ID 컬럼과 마찬가지로 시퀀스를 사용하여 값을 생성하면 일반적으로 다른 방법에 비해 응용프로그램 성능이 향상됩니다. 시퀀스를 사용하는 대신 현재 값이 저장된 단일 컬럼 테이블을 작성하고 해당 값을 트리거를 사용하거나 응용프로그램의 제어를 통해 증분시킬 수 있습니다. 그러나 응용프로그램이 단일 컬럼 테이블에 동시에 액세스하는 분산 환경에서는 테이블에 순차적인 액세스를 강제 실행하기 위해 필요한 잠금이 성능에 심각한 영향을 미칠 수 있습니다.

시퀀스를 사용하면 단일 컬럼 테이블 접근 방식과 연관된 잠금 문제를 피할 수 있으며 시퀀스 값을 메모리에서 캐시하여 응답 시간을 개선할 수 있습니다. 시퀀스를 사용하는 응용프로그램 성능을 최대화하려면 시퀀스가 적합한 양의 시퀀스 값을 캐시하는지 확인하십시오. CREATE SEQUENCE 및 ALTER SEQUENCE문의 CACHE절은 데이터베이스 관리 프로그램이 생성하여 메모리에 저장하는 최대 시퀀스 값 수를 지정합니다.

시퀀스에서 순서 대로 값을 생성해야 하고 시스템 오류 또는 데이터베이스 비활성화 때문에 해당 순서에 갭이 없어야 하는 경우 CREATE SEQUENCE문에서 ORDER 및 NO CACHE절을 사용하십시오. NO CACHE절을 사용하면 생성된 값에 갭이 나타나지 않지만 새 값을 생성할 때마다 시퀀스가 데이터베이스 로그에 이를 기록하도록 강제 실행하기 때문에 일부 응용프로그램 성능이 저하됩니다. 롤백하고 실제적으로는 요청한 해당 시퀀스 값을 사용하지 않는 트랜잭션으로 인해 여전히 갭이 나타날 수 있습니다.

시퀀스와 ID 컬럼 비교

시퀀스와 ID 컬럼은 DB2 응용프로그램에서 비슷한 용도로 사용되지만 중요한 차이가 있습니다. ID 컬럼은 LOAD 유틸리티를 사용하여 자동으로 단일 테이블의 컬럼 값을 생성합니다. 시퀀스는 CREATE SEQUENCE문을 사용하여 모든 SQL문에서 사용할 수 있는 순차 값을 요청받을 경우에 생성합니다.

ID 컬럼

데이터베이스 관리 프로그램이 테이블에 추가되는 각 행의 고유 숫자 값을 자동으로 생성할 수 있게 합니다. 테이블을 작성 중이고 해당 테이블에 추가되는 각 행을 식별할 필요가 있을 경우 CREATE TABLE문의 파트로 테이블 정의에 ID 컬럼을 추가할 수 있습니다.

```
CREATE TABLE <table name>
  (<column name 1> INT,
   <column name 2>, DOUBLE,
   <column name 3> INT NOT NULL GENERATED ALWAYS AS IDENTITY
     (START WITH <value 1>, INCREMENT BY <value 2>))
```

이 예에서는 세 번째 컬럼에서 ID 컬럼을 식별합니다. 사용자가 정의할 수 있는 속성 중에는 행이 추가될 경우에 각 행을 고유하게 정의하기 위해 컬럼에서 사용되는 값이 있습니다. INCREMENT BY절 뒤에 오는 값은 테이블에 행이 추가될 때마다 ID 컬럼 컨텐츠의 후속 값을 증분시키는 크기를 표시합니다.

ID 컬럼이 작성되면 ALTER TABLE문을 사용하여 ID 등록 정보를 변경하거나 제거할 수 있습니다. ALTER TABLE문을 사용하여 다른 컬럼에 대한 ID 등록 정보를 추가할 수도 있습니다.

시퀀스 자동으로 값을 생성할 수 있게 합니다. 시퀀스는 고유 키 값을 생성하는 태스크에 가장 적합합니다. 응용프로그램에서 시퀀스를 사용하여, 다른 방법을 통한 고유 카운터 생성으로 인해 발생할 수 있는 동시성 및 성능 문제가 일어나지 않게 할 수 있습니다. ID 컬럼과는 달리 시퀀스는 특정 테이블 컬럼에 고정되지 않으며 고유 테이블 컬럼에 바운드되지도 않고 해당 테이블 컬럼을 통해서만 액세스할 수 있습니다.

제한 없이 값을 증분시키거나 감소시키거나, 사용자 정의 한계까지 증분/감소시킨 다음 중지하거나, 사용자 정의 한계까지 증분/감소시킨 다음 시작 부분으로 되돌아가 다시 시작함으로써 값을 생성하도록 시퀀스를 작성하고 나중에 변경할 수 있습니다. 시퀀스는 단일 파티션 데이터베이스에서만 지원됩니다.

다음 예에서는 orderseq라는 시퀀스 작성 방법을 보여줍니다.

```
CREATE SEQUENCE orderseq
  START WITH 1
  INCREMENT BY 1
  NOMAXVALUE
  NOCYCLE
  CACHE 50
```

이 예제에서 시퀀스는 1에서 시작하고 1씩 증가하며 상한은 없습니다. 지정된 상한이 없기 때문에 처음으로 되돌아가 1부터 다시 시작할 이유가 없습니다. CACHE 매개변수는 데이터베이스 관리 프로그램이 메모리에 사전에 할당하여 보존하는 최대 시퀀스 값 수를 지정합니다.

시퀀스 작성

시퀀스를 작성하려면 CREATE SEQUENCE문을 사용합니다. ID 컬럼 속성과 달리 시퀀스는 특정 테이블 컬럼에 연결되거나 고유한 테이블 컬럼에 바운드되지 않고 해당 테이블 컬럼을 통해서만 액세스할 수 있습니다.

NEXT VALUE 또는 PREVIOUS VALUE 표현식을 사용할 수 있는 경우 여러 가지 제한사항이 있습니다. 시퀀스는 작성 또는 변경할 수 있으므로 시퀀스는 다음 중 한 가지 방법으로 값을 생성합니다.

- 바운드하지 않고 단순히 상수의 크기를 변경하여 증분 또는 감소
- 사용자 정의 한계 및 중지로 단순히 증분 또는 감소
- 사용자 정의 한계로 단순히 증분 또는 감소하고 시작 지점으로 다시 순환

주: 시퀀스를 사용하는 데이터베이스를 복구하는 경우 주의하십시오. 데이터베이스 외부에서 사용되는 시퀀스 값의 경우 예를 들어, 은행 수표에 사용되는 시퀀스 번호의 경우 데이터베이스 오류가 발생하기 이전 시점으로 데이터베이스가 복구되면 이로 인해 일부 시퀀스에 대해 중복된 값이 생성될 수 있습니다. 값이 중복되는 경우를 피하려면 데이터베이스 외부에서 시퀀스 값을 사용하는 데이터베이스를 이전 시점으로 복구하지 말아야 합니다.

모든 옵션의 디폴트값을 사용하여 order_seq 시퀀스를 작성하려면 응용프로그램에서 또는 동적 SQL문을 사용하여 다음 명령문을 발행하십시오.

```
CREATE SEQUENCE order_seq
```

이 시퀀스는 1에서 시작하고 상한 없이 1씩 증가합니다.

다음 예는 101에서 시작해서 200까지 이어지는 은행 수표의 일괄처리를 보여줍니다. 첫 번째 순서는 1-100이었습니다. 시퀀스는 101에서 시작하여 200까지 1씩 증가합니다. 중복된 수표 번호가 생성되지 않도록 NOCYCLE이 지정됩니다. CACHE 매개변수와 연결된 숫자는 데이터베이스 관리 프로그램에서 미리 할당하여 메모리에 저장하는 시퀀스 값의 최대 수를 지정합니다.

```
CREATE SEQUENCE order_seq
   START WITH 101
   INCREMENT BY 1
   MAXVALUE 200
   NOCYCLE
   CACHE 25
```

이러한 옵션 및 기타 옵션과 권한 부여 요구사항에 대한 자세한 정보는 CREATE SEQUENCE문을 참조하십시오.

순차 값 생성

순차 값 생성은 일반적인 데이터베이스 응용프로그램 개발 문제점입니다. 이 문제의 최선의 해결책은 SQL로 된 시퀀스 표현식 및 시퀀스를 사용하는 것입니다. 각 *시퀀스*는 고유한 이름이 지정된 데이터베이스 오브젝트로서 시퀀스 표현식을 통해서만 액세스할 수 있습니다.

*시퀀스 표현식*에는 PREVIOUS VALUE 표현식과 NEXT VALUE 표현식 두 가지가 있습니다. PREVIOUS VALUE 표현식은 지정된 시퀀스에 사용할 응용프로그램 프로

세스에서 최근에 생성된 값을 리턴합니다. PREVIOUS VALUE 표현식과 동일한 명령문에서 발생하는 모든 NEXT VALUE 표현식은 해당 명령문의 PREVIOUS VALUE 표현식을 통해 생성되는 값에 영향을 주지 않습니다. NEXT VALUE 시퀀스 표현식은 시퀀스의 값을 증분시키고 시퀀스의 새 값을 리턴합니다.

시퀀스를 작성하려면 CREATE SEQUENCE 명령문을 실행하십시오. 예를 들어, 디폴트 속성을 사용하여 id_values라는 시퀀스를 작성하려면 다음 명령문을 실행하십시오.

```
CREATE SEQUENCE id_values
```

시퀀스의 응용프로그램 세션에서 첫 번째 값을 생성하려면 NEXT VALUE 표현식을 사용하여 VALUES문을 실행하십시오.

```
VALUES NEXT VALUE FOR id_values

1
-----------
          1

              1 레코드가 선택되었습니다.
```

시퀀스의 다음 값으로 컬럼 값을 갱신하려면 다음과 같이 UPDATE문에 NEXT VALUE 표현식을 포함시키십시오.

```
UPDATE staff
  SET id = NEXT VALUE FOR id_values
  WHERE id = 350
```

시퀀스의 다음 값을 사용하여 테이블에 새 행을 삽입하려면 다음과 같이 INSER문에 NEXT VALUE 표현식을 포함시키십시오.

```
INSERT INTO staff (id, name, dept, job)
  VALUES (NEXT VALUE FOR id_values, 'Kandil', 51, 'Mgr')
```

ID 컬럼 또는 시퀀스 사용 시기 판별

ID 컬럼과 시퀀스 사이에는 유사점이 있지만 다른점도 있습니다. 데이터베이스 또는 응용프로그램 설계 시 각각의 특성을 사용할 수 있습니다.

사용자의 데이터베이스 설계와 해당 데이터베이스를 사용하는 응용프로그램에 따라 다음과 같은 특성을 활용하면 ID 컬럼 사용 시기 및 시퀀스 사용 시기를 판별하는 데 도움이 됩니다.

ID 컬럼 특성

- ID 컬럼은 자동으로 단일 테이블의 값을 생성합니다.
- ID 컬럼이 GENERATED ALWAYS로 정의된 경우에는 항상 데이터베이스 관리 프로그램이 사용되는 값을 생성합니다. 응용프로그램은 테이블 컨텐츠를 수정하는 동안 응용프로그램 고유의 값을 제공할 수 없습니다.

- 행을 삽입한 후, IDENTITY_VAL_LOCAL() 함수를 사용하거나 SELECT FROM INSERT문을 사용해서 삽입 항목에서 ID 컬럼을 다시 선택하여 생성된 ID 값을 검색할 수 있습니다.
- LOAD 유틸리티가 IDENTITY 값을 생성할 수 있습니다.

시퀀스 특성

- 시퀀스는 하나의 테이블에만 고정되지 않습니다.
- 시퀀스는 모든 SQL 또는 XQuery 명령문에서 사용할 수 있는 순차 값을 생성합니다.

  모든 응용프로그램에서 시퀀스를 사용할 수 있으므로 지정된 시퀀스에서 다음 값 검색을 제어하고 실행 중인 명령문 이전에 생성된 값의 검색을 제어하는 데 사용되는 두 가지 표현식이 있습니다. PREVIOUS VALUE 표현식은 현재 세션에 있는 이전 명령문에 지정된 시퀀스의 최근 생성 값을 리턴합니다. NEXT VALUE 표현식은 지정된 시퀀스의 다음 값을 리턴합니다. 이러한 표현식을 사용하면 여러 테이블에 있는 여러 SQL 및 XQuery 명령문에서 동일한 값을 사용할 수 있습니다.

시퀀스 수정

ALTER SEQUENCE문을 사용하여 기존 시퀀스의 속성을 수정합니다.

수정할 수 있는 시퀀스 속성은 다음과 같습니다.

- 이후 값 간의 증분 변경
- 새로운 최소값 또는 최대값 설정
- 캐시 시퀀스 번호 수 변경
- 시퀀스가 순환되는지 여부 변경
- 시퀀스 번호가 요청 순서대로 생성되어야 하는지의 변경
- 시퀀스 재시작

시퀀스 작성의 일부로 찾을 수 없는 두 개의 태스크가 있습니다. 이러한 태스크는 다음과 같습니다.

- **RESTART:** 시퀀스 작성 시 시작 값으로 내재적 또는 명시적으로 지정된 값으로 시퀀스를 재설정합니다.
- **RESTART WITH <numeric-constant>:** 정확한 숫자 상수 값으로 시퀀스를 재설정합니다. 숫자 상수는 소수점 오른쪽에 0이 아닌 숫자가 있는 양수 또는 음수 값일 수 있습니다.

시퀀스를 재시작하거나 CYCLE로 변경한 이후 중복 시퀀스 번호를 생성할 수 있습니다. 이후 시퀀스 번호만 ALTER SEQUENCE문의 영향을 받습니다.

시퀀스의 데이터 유형은 변경할 수 없습니다. 대신 현재 시퀀스를 삭제한 다음 새 시퀀스를 작성하여 새 데이터 유형을 지정해야 합니다.

시퀀스 변경 시 데이터베이스 관리 프로그램에서 사용되지 않는 캐시된 모든 시퀀스 값은 손실됩니다.

시퀀스 정의 보기

PREVIOUS VALUE 옵션을 통해 VALUES문을 사용하여 시퀀스와 연관된 참조 정보를 확인하거나 시퀀스 자체를 확인합니다.

시퀀스의 현재 값을 표시하려면 PREVIOUS VALUE 표현식을 사용하여 VALUES문을 발행하십시오.

```
VALUES PREVIOUS VALUE FOR id_values

1

-----------
          1

  1 레코드가 선택되었습니다.
```

시퀀스의 현재 값을 반복적으로 검색할 수 있고 해당 시퀀스에서 리턴하는 값은 NEXT VALUE 표현식을 발행할 때까지 변경되지 않습니다. 다음 예에서 PREVIOUS VALUE 표현식은 현재 연결의 NEXT VALUE 표현식이 시퀀스의 값을 증분할 때까지 값 1을 리턴합니다.

```
VALUES PREVIOUS VALUE FOR id_values

1

-----------
          1

  1 레코드가 선택되었습니다.

VALUES PREVIOUS VALUE FOR id_values

1

-----------
          1

  1 레코드가 선택되었습니다.

VALUES NEXT VALUE FOR id_values

1

-----------
```

```
          2

            1 레코드가 선택되었습니다.

VALUES PREVIOUS VALUE FOR id_values

1

-----------
          2

            1 레코드가 선택되었습니다.
```

이는 다른 연결에서 동시에 시퀀스 값을 사용하는 경우에도 참입니다.

시퀀스 삭제

시퀀스를 삭제하려면 DROP문을 사용합니다.

시퀀스를 삭제하는 경우 명령문의 권한 부여 ID는 DBADM 권한을 보유하고 있어야 합니다.

다음 명령문을 사용하여 특정 시퀀스를 삭제할 수 있습니다.

```
DROP SEQUENCE <sequence_name>
```

여기서 <sequence_name>은 삭제되는 시퀀스의 이름이고 기존 시퀀스를 정확히 식별하는 내재적 또는 명시적 스키마 이름이 포함되어 있습니다.

IDENTITY 컬럼에 대해 시스템이 작성한 시퀀스는 DROP SEQUENCE문을 사용하여 삭제할 수 없습니다.

시퀀스가 삭제되면 시퀀스에 대한 모든 특권도 삭제됩니다.

시퀀스 코딩 방법 예

작성된 여러 응용프로그램에서는 송장 번호, 고객 번호 및 새 항목이 필요할 때마다 1씩 증분되는 기타 오브젝트를 추적하기 위해 시퀀스 번호를 사용해야 합니다. 데이터베이스 관리 프로그램에서는 ID 컬럼을 사용하여 테이블의 값을 자동 증분시킬 수 있습니다. 개별 테이블에서는 이 기술이 제대로 적용되지만 다중 테이블에서 사용되어야 하는 고유 값을 생성하는 데는 편리한 방법이 아닐 수 있습니다.

시퀀스 오브젝트를 사용하면 프로그래머의 제어 아래 증분되어 여러 테이블에서 사용할 수 있는 값을 작성할 수 있습니다. 다음 예는 정수 데이터 유형을 사용하여 고객 번호로 사용하도록 작성 중인 시퀀스 번호를 표시합니다.

```
CREATE SEQUENCE customer_no AS INTEGER
```

디폴트로 시퀀스 번호는 1에서 시작하여 한 번에 1씩 증분되며 데이터 유형은 INTEGER입니다. 응용프로그램은 NEXT VALUE 함수를 사용하여 시퀀스의 다음 값을 가져와야 합니다. 이 함수는 후속 SQL문에서 사용할 수 있는 시퀀스의 다음 값을 생성합니다.

```
VALUES NEXT VALUE FOR customer_no
```

VALUES 함수를 사용하여 다음 번호를 생성하는 대신 프로그래머가 INSERT문에서 이 함수를 사용할 수 있습니다. 예를 들어, 고객 테이블의 첫 번째 컬럼에 고개 번호가 들어 있는 경우 다음과 같이 INSERT문을 작성할 수 있습니다.

```
INSERT INTO customers VALUES
  (NEXT VALUE FOR customer_no, 'comment', ...)
```

다른 테이블에 삽입하기 위해 시퀀스 번호를 사용해야 하는 경우 PREVIOUS VALUE 함수를 사용하여 이전에 생성된 값을 검색할 수 있습니다. 예를 들어, 방금 작성된 고객 번호를 후속 송장 레코드에 사용해야 하는 경우 SQL에 PREVIOUS VALUE 함수가 포함됩니다.

```
INSERT INTO invoices
  (34,PREVIOUS VALUE FOR customer_no, 234.44, ...)
```

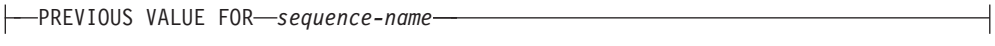
응용프로그램에서 PREVIOUS VALUE 함수를 여러 번 사용할 수 있으며 이 함수는 해당 응용프로그램이 생성한 마지막 값만 리턴합니다. 후속 트랜잭션에서 이미 시퀀스를 다른 값으로 증분시켰을 가능성이 있지만 사용자에게는 생성된 마지막 번호만 표시됩니다.

시퀀스 참조

sequence-reference:

nextval-expression:

prevval-expression:



NEXT VALUE FOR *sequence-name*

NEXT VALUE 표현식은 *sequence-name*으로 지정되는 시퀀스에 대한 다음 값을 생성하고 리턴합니다.

PREVIOUS VALUE FOR *sequence-name*

PREVIOUS VALUE 표현식은 현재 응용프로그램 프로세스의 이전 명령문에 대해 지정된 시퀀스에 대한 가장 최근에 생성된 값을 리턴합니다. 이 값은 시퀀스 이름을 지정하는 PREVIOUS VALUE 표현식을 사용하여 반복적으로 참조할 수 있습니다. 단일 명령문 내부에서 동일한 시퀀스 이름을 지정하는 PREVIOUS VALUE 표현식의 다중 인스턴스가 있을 수 있는데, 모두 동일한 값을 리턴합니다. 파티션된 데이터베이스 환경에서 PREVIOUS VALUE 표현식은 가장 최근에 생성된 값을 리턴할 수 없습니다.

PREVIOUS VALUE 표현식은 동일한 시퀀스 이름을 지정하는 NEXT VALUE 표현식이 현재 또는 이전 트랜잭션 중 하나에서 이미 현재 응용프로그램 프로세스에서 참조된 경우에만 사용할 수 있습니다(SQLSTATE 51035).

주

- NEXT VALUE 표현식이 시퀀스의 이름을 지정할 때 해당 시퀀스에 대해 새 값이 생성됩니다. 그러나 쿼리 내부에서 동일한 시퀀스 이름을 지정하는 NEXT VALUE 표현식의 다중 인스턴스가 있는 경우, 시퀀스 카운터는 결과의 각 행에 대해 한 번만 증분되고 NEXT VALUE의 모든 인스턴스가 결과 행에 대해 동일한 값을 리턴합니다.
- 아래 표시된 것처럼, 첫 번째 행에 대해 NEXT VALUE 표현식으로 시퀀스 번호를 지정하고(시퀀스 값을 생성함) 기타 행에 대해 PREVIOUS VALUE 표현식으로 시퀀스 번호를 참조하여(PREVIOUS VALUE의 인스턴스는 현재 세션에서 기장 최근에 생성된 시퀀스 값을 참조함) 두 개의 개별 테이블에서 동일한 시퀀스 번호를 고유 키 값으로 사용할 수 있습니다.

```
INSERT INTO order(orderno, cutno)
  VALUES (NEXT VALUE FOR order_seq, 123456);

INSERT INTO line_item (orderno, partno, quantity)
  VALUES (PREVIOUS VALUE FOR order_seq, 987654, 1);
```

- NEXT VALUE 및 PREVIOUS VALUE 표현식은 다음 위치에 지정할 수 있습니다.
  - Select문 또는 SELECT INTO문(select절 내에서 명령문이 DISTINCT 키워드, GROUPBY절, ORDER BY절, UNION 키워드, INTERSECT 키워드 또는 EXCEPT 키워드를 포함하지 않는 경우)
  - INSERT문(VALUES절 내)
  - INSERT문(fullselect의 select-clause 내)
  - UPDATE문(SET절 내(검색 또는 위치 지정된 UPDATE문), NEXT VALUE는 SET절의 표현식의 fullselect의 select-clause에 지정할 수 없음)

  - SET 변수 명령문(표현식 fullselect의 select-clause 내부는 제외. NEXT VALUE 표현식을 트리거에 지정할 수 있지만 PREVIOUS VALUE 표현식은 지정할 수 없음)
  - VALUES INTO문(표현식 fullselect의 select-clause 내)
  - CREATE PROCEDURE문(SQL 프로시저의 routine-body 내)
  - 트리거된 조치의 CREATE TRIGGER문(NEXT VALUE 표현식은 지정할 수 있지만 PREVIOUS VALUE 표현식은 지정할 수 없음)
- 다음 위치에는 NEXT VALUE 및 PREVIOUS VALUE 표현식을 지정할 수 없습니다(SQLSTATE 428F9).
  - 완전 외부 조인의 조인 조건
  - CREATE 또는 ALTER TABLE문의 컬럼에 대한 DEFAULT 값
  - CREATE 또는 ALTER TABLE문의 생성된 컬럼 정의
  - CREATE TABLE 또는 ALTER TABLE문의 요약 테이블 정의
  - CHECK 제한조건의 조건
  - CREATE TRIGGER문(NEXT VALUE 표현식은 지정할 수 있지만 PREVIOUS VALUE 표현식은 지정할 수 없음)
  - CREATE VIEW문
  - CREATE METHOD문
  - CREATE FUNCTION문
  - XMLQUERY, XMLEXISTS 또는 XMLTABLE 표현식의 인수
- 또한 다음 위치에 NEXT VALUE 표현식을 지정할 수 없습니다(SQLSTATE 428F9).
  - CASE 표현식
  - 집계 함수의 매개변수 목록
  - 위에서 명시적으로 허용되는 것을 제외한 컨텍스트의 서브쿼리
  - 외부 SELECT가 DISTINCT 연산자를 포함하는 SELECT문
  - 조인의 조인 조건
  - 외부 SELECT가 GROUP BY절을 포함하는 SELECT문
  - 외부 SELECT가 UNION, INTERSECT 또는 EXCEPT 집합 연산자를 사용하여 다른 SELECT문과 결합되는 SELECT문
  - 중첩 테이블 표현식
  - 테이블 함수의 매개변수 목록
  - 가장 외부의 SELECT문, DELETE 또는 UPDATE문의 WHERE절
  - 가장 외부의 SELECT문의 ORDER BY절
  - UPDATE문의 SET절에서 표현식의 fullselect의 select-clause
  - SQL 루틴의 IF, WHILE, DO ... UNTIL 또는 CASE문

- 값이 시퀀스에 대해 생성되고, 해당 값이 이용되고, 다음에 값이 요청될 때 새 값이 생성됩니다. 이는 NEXT VALUE 표현식을 포함하는 명령문이 실패하거나 롤백될 때도 적용됩니다.

  INSERT문이 컬럼에 대한 VALUES 목록에 NEXT VALUE 표현식을 포함하는 경우 및 INSERT 실행 중에 어느 지점에서 오류가 발생하는 경우(다음 시퀀스 값 생성 문제 또는 또 다른 컬럼에 대한 값의 문제일 수 있음), 삽입 실패가 발생하며 (SQLSTATE 23505), 시퀀스에 대해 생성된 값은 이용된 것으로 간주됩니다. 일부 경우에는 동일한 INSERT문을 다시 발행하면 성공할 수도 있습니다.

  예를 들어 NEXT VALUE가 사용되었고 생성된 시퀀스 값이 이미 인덱스에 존재하는 컬럼에 대한 고유 인덱스의 존재 결과인 오류를 고려하십시오. 시퀀스에 대해 생성되는 다음 값이 인덱스에 존재하지 않으므로 후속 INSERT가 성공할 수 있습니다.
- ***PREVIOUS VALUE의 범위:*** PREVIOUS VALUE의 값은 현재 세션에서 시퀀스의 다음 값이 생성되고 시퀀스가 삭제 또는 변경되거나 응용프로그램 세션을 종료할 때까지 지속됩니다. 값은 COMMIT 또는 ROLLBACK문에 영향을 받지 않습니다. PREVIOUS VALUE의 값은 직접 설정될 수 없으며 시퀀스의 NEXT VALUE 표현식의 실행 결과입니다.

  흔히 사용되는 기술, 특히 성능은 연결 설정을 관리하고 임의 연결에 대한 트랜잭션을 라우트할 응용프로그램 또는 제품을 위한 것입니다. 이러한 상황에서 트랜잭션을 종료할 때까지만 시퀀스의 PREVIOUS VALUE의 영향을 받아야 합니다. 이러한 상황 유형의 예에는 다음과 같은 응용프로그램을 포함할 수 있습니다(XA 프로토콜, 연결 풀링, 연결 집중기(connection concentrator) 및 HADR을 사용하여 장애 복구 달성).
- 시퀀스에 대한 값을 생성할 때 시퀀스의 최대값(또는 내림차순 시퀀스의 경우 최소값)이 초과되었고 순환이 허용되지 않는 경우 오류가 발생합니다(SQLSTATE 23522). 이 경우에 사용자는 시퀀스를 ALTER(변경)하여 승인할 수 있는 값의 범위를 확장하거나 시퀀스의 순환을 사용 가능하게 하거나, DROP(삭제)하고 더 큰 값의 범위를 갖는 다른 데이터 유형을 갖는 새 시퀀스를 CREATE(작성)할 수 있습니다.

  예를 들어, 데이터 유형이 SMALLINT인 시퀀스를 정의했을 수 있으며 결과적으로 해당 시퀀스는 지정 가능한 값을 모두 소비합니다. 시퀀스를 삭제(Drop)하고 새 정의를 갖는 시퀀스를 다시 작성하여 시퀀스를 INTEGER로 재정의하십시오.
- 커서의 선택 명령문에서 NEXT VALUE 표현식에 대한 참조는 결과 테이블의 행에 대해 생성되는 값을 참조합니다. 데이터베이스에서 페치되는 각 행에 대한 NEXT VALUE 표현식에 대해 시퀀스 값 하나가 생성됩니다. 클라이언트에서 블로킹이 수행되는 경우 값은 FETCH문의 처리 전에 서버에 생성되었을 수 있습니다. 결과 테이블의 행 블로킹이 있을 때 발생할 수 있습니다. 클라이언트 응용프로그램이 데이

터베이스가 구체화한 모든 행을 명시적으로 FETCH(페치)하지 않는 경우 응용프로그램은(리턴되지 않은 구체화된 행에 대한) 모든 생성된 시퀀스 값의 결과를 확인하지 않습니다.

- 커서의 선택 명령문에서 PREVIOUS VALUE 표현식에 대한 참조는 커서를 열기 전에 지정된 시퀀스에 대해 생성된 값을 참조합니다. 그러나 커서를 닫으면 후속 명령문에서 지정된 시퀀스 또는 커서가 다시 열리는 이벤트에서 동일한 명령문에 대한 PREVIOUS VALUE로 리턴되는 값에 영향을 미칠 수 있습니다. 커서의 선택 명령문이 동일한 시퀀스 이름에 대한 NEXT VALUE 참조를 포함한 경우입니다.
- *호환성*
  - 이전 버전 DB2와의 호환성을 위해,
    - NEXTVAL 및 PREVVAL을 NEXT VALUE 및 PREVIOUS VALUE 대신 지정할 수 있습니다.
  - IBM IDS와의 호환성을 위해,
    - *sequence-name*.NEXTVAL을 NEXT VALUE FOR *sequence-name* 대신 지정할 수 있습니다.
    - *sequence-name*.CURRVAL을 PREVIOUS VALUE FOR *sequence-name* 대신 지정할 수 있습니다.

예:

다음과 같이 ″order″라는 테이블이 있고, ″order_seq″라는 시퀀스가 작성된다고 가정하십시오.

```
CREATE SEQUENCE order_seq
  START WITH 1
  INCREMENT BY 1
  NO MAXVALUE
  NO CYCLE     CACHE 24
```

다음은 NEXT VALUE 표현식으로 ″order_seq″ 시퀀스 번호를 생성하는 방법의 몇 가지 예입니다.

```
INSERT INTO order(orderno, custno)
  VALUES (NEXT VALUE FOR order_seq, 123456);
```

또는

```
UPDATE order
  SET orderno = NEXT VALUE FOR order_seq
  WHERE custno = 123456;
```

또는

```
VALUES NEXT VALUE FOR order_seq INTO :hv_seq;
```

제 16 장 뷰

*뷰*는 데이터를 유지하지 않고도 효율적으로 데이터를 나타낼 수 있는 방법입니다. 뷰는 실제 테이블이 아니며 영구 스토리지가 필요하지 않습니다. 『가상 테이블』이 작성되어 사용됩니다.

뷰는 하나 이상의 테이블에서 데이터를 다르게 검토할 수 있는 방법을 제공합니다. 뷰는 결과 테이블의 이름 지정된 스펙입니다. 스펙은 SQL문에서 뷰가 참조될 때마다 실행되는 SELECT문입니다. 뷰에는 테이블과 마찬가지로 컬럼과 행이 있습니다. 모든 뷰를 테이블처럼 데이터 검색에 사용할 수 있습니다. 삽입, 갱신 또는 삭제에 뷰를 사용할 수 있는지 여부는 뷰의 정의에 따라 다릅니다.

뷰의 기본이 되는 테이블에 포함된 모든 또는 일부 컬럼이나 행을 뷰에 포함시킬 수 있습니다. 예를 들어, 특정 부서의 모든 직원을 나열할 수 있도록 하나의 뷰에 부서 테이블과 직원 테이블을 결합시킬 수 있습니다.

그림 47에는 테이블과 뷰 간의 관계가 표시되어 있습니다.

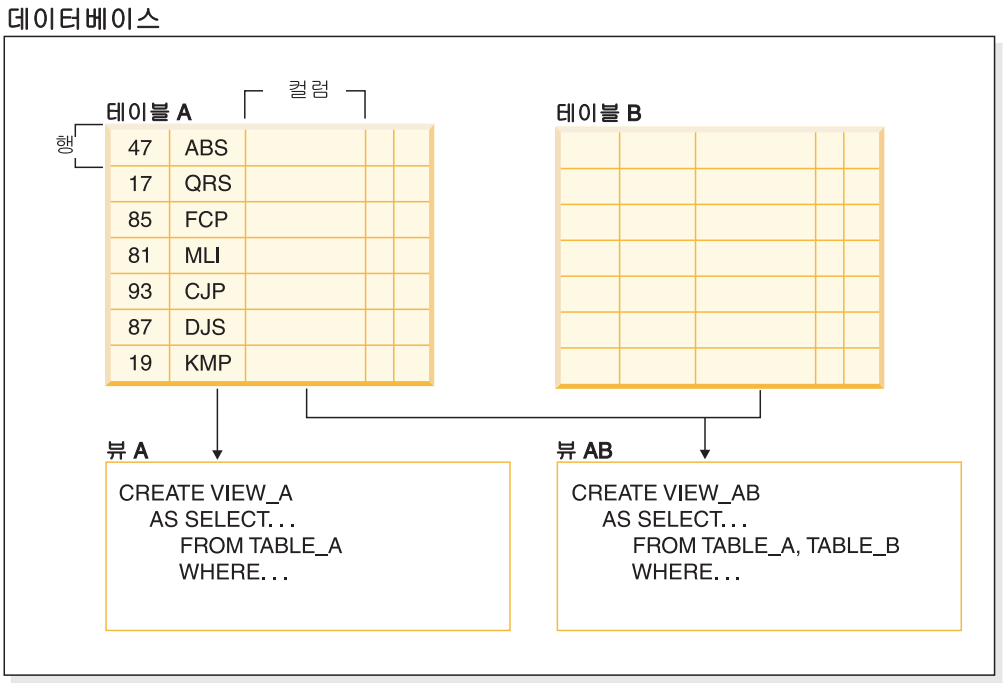


그림 47. 테이블과 뷰의 관계

뷰를 사용하면 여러 사용자가 동일한 데이터를 서로 다른 표시 형식으로 볼 수 있기 때문에 뷰를 사용하여 중요한 데이터에 액세스하는 것을 제어할 수 있습니다. 예를 들어, 몇 명의 사용자가 직원에 대한 데이터 테이블에 액세스 중인 경우가 있습니다. 관리자

는 자신의 직원에 대한 데이터를 확인할 수 있지만 다른 부서 직원의 데이터는 볼 수 없습니다. 채용 담당자는 모든 직원의 채용 날짜를 확인할 수 있지만 직원의 급여는 볼 수 없으며 재무 담당자는 급여를 확인할 수 있지만 채용 날짜는 볼 수 없습니다. 이들 각각의 사용자는 테이블에서 파생된 뷰를 사용하여 작업합니다. 각 뷰는 테이블로 표시되고 고유 이름을 갖습니다.

뷰의 컬럼이 기본 테이블 컬럼에서 직접 파생된 경우 해당 뷰 컬럼은 테이블 컬럼에 적용되는 모든 제한조건을 상속합니다. 예를 들어, 뷰에 해당 테이블의 외부 키가 포함된 경우 이 뷰를 사용하는 삽입 및 갱신 조작에는 테이블과 동일한 참조 제한조건이 적용됩니다. 또한 뷰의 테이블이 상위 테이블인 경우 해당 뷰를 사용하는 삭제 및 갱신 조작에는 테이블의 삭제 및 갱신 조작에 적용되는 것과 동일한 규칙이 적용됩니다.

뷰는 결과 테이블에서 각 컬럼의 데이터 유형을 추출하거나 사용자 정의 구조화된 유형의 속성을 기본으로 유형을 지정할 수 있습니다. 이를 *유형이 지정된 뷰*라고 합니다. 유형이 지정된 테이블과 마찬가지로 유형이 지정된 뷰는 뷰 계층 구조의 파트가 될 수 있습니다. *서브뷰*는 *수퍼 뷰*에서 컬럼을 상속합니다. *서브뷰*라는 용어는 하나의 유형이 지정된 뷰 및 뷰 계층 구조에서 해당 뷰 아래 있는 모든 유형이 지정된 뷰에 적용됩니다. V 뷰의 *올바른 서브뷰*는 유형이 지정된 뷰 계층 구조에서 V 아래 있는 뷰입니다.

예를 들어, 테이블이 삭제된 경우 뷰가 작동 불능 상태가 될 수 있습니다. 이런 경우 해당 뷰를 더 이상 SQL 조작에 사용할 수 없습니다.

뷰 설계

*뷰*는 하나 이상의 테이블에서 데이터를 다르게 검토할 수 있는 방법을 제공합니다. 뷰는 결과 테이블의 이름 지정된 스펙입니다.

스펙은 SQL문에서 뷰가 참조될 때마다 실행되는 SELECT문입니다. 뷰에는 기본 테이블과 마찬가지로 컬럼과 행이 있습니다. 모든 뷰를 테이블처럼 데이터 검색에 사용할 수 있습니다. 삽입, 갱신 또는 삭제에 뷰를 사용할 수 있는지 여부는 뷰의 정의에 따라 다릅니다.

뷰가 허용하는 조작에 따라 뷰를 분류합니다. 뷰에는 다음과 같은 특성이 있습니다.

- 삭제 가능
- 갱신 가능 여부
- 삽입 가능
- 읽기 전용

뷰 유형은 갱신 성능에 따라 설정됩니다. 분류는 뷰에 대해 허용되는 SQL 조작의 종류를 표시합니다.

참조 및 점검 제한조건은 개별적으로 처리됩니다. 이들 제한조건은 뷰 분류에 영향을 미치지 않습니다.

예를 들어, 참조 제한조건 때문에 테이블에 행을 삽입하지 못할 수도 있습니다. 해당 테이블을 사용하여 뷰를 작성하는 경우 해당 뷰를 사용하여 행을 삽입할 수도 없습니다. 그러나 뷰가 삽입 가능한 뷰의 모든 규칙을 충족하는 경우에는 여전히 삽입 가능한 뷰로 취급됩니다. 이는 삽입 제한조건이 테이블에 관한 것이고 뷰 정의에 대한 것이 아니기 때문입니다.

자세한 정보는 CREATE VIEW문을 참조하십시오.

시스템 카탈로그 뷰

데이터베이스 관리 프로그램은 자신이 제어하는 데이터에 대한 정보가 포함되어 있는 뷰 및 테이블 세트를 유지보수합니다. 이러한 테이블 및 뷰는 통틀어 *시스템 카탈로그*라고 알려져 있습니다.

시스템 카탈로그에는 테이블, 뷰, 인덱스, 패키지 및 함수와 같은 데이터베이스 오브젝트의 논리적 구조 및 물리적 구조에 대한 정보가 포함되어 있습니다. 여기에는 통계 정보도 들어 있습니다. 데이터베이스 관리 프로그램은 시스템 카탈로그에 있는 설명이 항상 정확하도록 관리합니다.

시스템 카탈로그 뷰는 기타 모든 데이터베이스 뷰와 같습니다. SQL문을 사용하여 시스템 카탈로그 뷰의 데이터를 쿼리할 수 있습니다. 갱신 가능한 시스템 카탈로그 뷰 세트를 사용하여 시스템 카탈로그의 특정 값을 수정할 수 있습니다.

점검 옵션이 있는 뷰

WITH CHECK OPTION으로 정의된 뷰는 수정되거나 삽입된 모든 행을 해당 뷰의 SELECT문에 따라 적용합니다. 점검 옵션이 있는 뷰 를 *대칭 뷰*라고도 합니다. 예를 들어, 부서 10의 직원만 리턴하는 대칭 뷰는 다른 부서의 직원을 삽입하는 것을 허용하지 않습니다. 따라서 이 옵션을 사용하면 데이터베이스에서 수정 중인 데이터의 무결성을 확보할 수 있으며 삽입 또는 갱신 조작 중에 조건에 위배되면 오류를 리턴합니다.

응용프로그램에서 원하는 규칙을 테이블 점검 제한조건으로 정의할 수 없거나 규칙이 모든 데이터 사용에 적용되지 않는 경우 다른 방법으로 규칙을 응용프로그램 논리에 적용할 수 있습니다. 데이터에 대한 조건을 지정된 WHERE절 및 WITH CHECK OPTION절의 파트로 사용하여 테이블의 뷰를 작성할 수 있습니다. 이 뷰 정의는 사용자의 응용프로그램에 유효한 세트에서만 데이터를 검색할 수 있도록 제한합니다. 또한 뷰를 갱신할 수 있는 경우에는 WITH CHECK OPTION절이 사용자의 응용프로그램에 적용 가능한 행만 갱신, 삽입 및 삭제할 수 있도록 제한합니다.

다음 뷰에는 WITH CHECK OPTION을 지정하면 안 됩니다.

- 읽기 전용 옵션으로 정의된 뷰(읽기 전용 뷰)

- NODENUMBER 또는 PARTITION 함수, 비결정적 함수(예: RAND) 또는 외부 조치가 포함된 함수를 참조하는 뷰
- 유형이 지정된 뷰

예 1

다음은 WITH CHECK OPTION을 사용하는 뷰 정의 예입니다. 조건이 항상 점검되는지 확인하기 위해서는 이 옵션이 필수입니다. 뷰에서는 DEPT가 항상 10이 되도록 합니다. 이렇게 하면 DEPT 컬럼의 입력 값이 제한됩니다. 새 값을 삽입하는 데 뷰를 사용하는 경우 WITH CHECK OPTION이 항상 적용됩니다.

```
CREATE VIEW EMP_VIEW2
  (EMPNO, EMPNAME, DEPTNO, JOBTITLE, HIREDATE)
AS SELECT ID, NAME, DEPT, JOB, HIREDATE FROM EMPLOYEE
  WHERE DEPT=10
WITH CHECK OPTION;
```

이 뷰가 INSERT문에서 사용되는 경우 DEPTNO 컬럼의 값이 10이 아니면 행이 거부됩니다. WITH CHECK OPTION이 지정되지 않은 경우에는 수정 중에 데이터 유효성이 확인되지 않습니다.

SELECT문에서 이 뷰를 사용하는 경우 조건부(WHERE절)가 호출되고 결과 테이블에는 데이터와 일치하는 행만 포함됩니다. 즉, WITH CHECK OPTION은 SELECT문의 결과에 영향을 미치지 않습니다.

예 2:

뷰를 사용하여 응용프로그램에 사용 가능한 테이블 데이터의 서브세트를 작성하고 삽입하거나 갱신할 데이터의 유효성을 확인할 수 있습니다. 뷰에서 원래 테이블의 해당 컬럼 이름과 다른 컬럼 이름을 사용할 수 있습니다. 예를 들어,

```
CREATE VIEW <name> (<column>, <column>, <column>)
   SELECT <column_name> FROM <table_name>
   WITH CHECK OPTION
```

예 3:

뷰를 사용하면 사용자의 프로그램 및 일반 사용자 쿼리에서 테이블 데이터를 확인할 수 있는 유연성이 제공됩니다.

다음 SQL문은 A00 부서의 모든 직원과 직원 번호 및 전화 번호가 나열된 EMPLOYEE 테이블에 뷰를 작성합니다.

```
CREATE VIEW EMP_VIEW (DA00NAME, DA00NUM, PHONENO)
   AS SELECT LASTNAME, EMPNO, PHONENO FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

이 명령문의 첫 번째 라인은 뷰의 이름을 지정하고 컬럼을 정의합니다. 이름 EMP_VIEW는 SYSCAT.TABLES에 있는 해당 스키마 내에서 고유해야 합니다. 뷰 이름에 데이터가 없어도 뷰 이름이 테이블 이름으로 표시됩니다. 뷰에는 DA00NAME, DA00NUM 및 PHONENO라는 세 개의 컬럼이 작성되고 이들 컬럼은 EMPLOYEE 테이블의 LASTNAME, EMPNO 및 PHONENO 컬럼에 해당됩니다. 나열된 컬럼 이름은 SELECT문의 선택 목록에 일대일로 적용됩니다. 컬럼 이름을 지정하지 않은 경우 뷰에서는 SELECT문 결과 테이블의 컬럼과 동일한 이름을 사용합니다.

두 번째 라인은 데이터베이스에서 선택할 값에 대해 설명하는 SELECT문입니다. 여기에는 ALL, DISTINCT, FROM, WHERE, GROUP BY 및 HAVING이 포함될 수 있습니다. 뷰의 컬럼을 선택할 데이터 오브젝트의 이름은 FROM절 뒤에 와야 합니다.

예 4

WITH CHECK OPTION절은 뷰에 갱신되거나 삽입된 행을 뷰 정의에 따라 점검하고 정의를 준수하지 않는 경우에는 거부해야 함을 표시합니다. 이는 데이터 무결성을 확장하지만 추가적인 처리가 필요합니다. 이 절을 생략하면 뷰 정의에 따라 삽입 및 갱신을 점검하지 않습니다.

다음 SQL문은 SELECT AS절을 사용하여 EMPLOYEE 테이블에 동일한 뷰를 작성합니다.

```
CREATE VIEW EMP_VIEW
   SELECT LASTNAME AS DA00NAME,
          EMPNO AS DA00NUM,
          PHONENO
   FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

이 예에서 EMPLOYEE 테이블에 급여 정보가 있을 수 있으며 모든 사람이 이 정보를 사용할 수 있으면 안 됩니다. 그러나 직원의 전화번호에는 일반적으로 액세스할 수 있어야 합니다. 이런 경우 LASTNAME 및 PHONENO 컬럼에서만 뷰를 작성할 수 있습니다. 뷰에 대한 액세스 권한은 PUBLIC에 부여할 수 있지만 전체 EMPLOYEE 테이블에 대한 액세스 권한은 급여 정보를 볼 수 있는 권한을 가진 사용자에게만 부여할 수 있습니다.

중첩된 뷰 정의

뷰가 다른 뷰를 기본으로 하는 경우 평가해야 하는 술어의 수는 WITH CHECK OPTION 스펙을 기본으로 합니다.

WITH CHECK OPTION 없이 뷰가 정의되는 경우 뷰의 정의는 삽입 또는 갱신 조작의 데이터 유효성 검사에 사용되지 않습니다. 그러나 뷰가 직접 또는 간접적으로 WITH CHECK OPTION을 사용하여 정의된 다른 뷰에 종속되는 경우 해당 수퍼 뷰의 정의가 삽입 또는 갱신 조작 점검에 사용됩니다.

WITH CASCADED CHECK OPTION을 사용하거나 WITH CHECK OPTION(CASCADED는 WITH CHECK OPTION의 디폴트값)만을 사용하여 뷰를 정의하는 경우 해당 뷰의 정의가 삽입 또는 갱신 조작 점검에 사용됩니다. 또한 뷰는 해당 뷰가 종속된 갱신 가능한 뷰에서 검색 조건을 상속합니다. 해당 뷰에 WITH CHECK OPTION이 포함되어 있지 않아도 검색 조건이 상속됩니다. 그런 다음 상속된 조건은 해당 뷰 또는 해당 뷰에 종속되는 뷰의 삽입 또는 갱신 조작에 적용되는 제한조건을 준수하기 위해 두 배가 됩니다.

예를 들어, V2 뷰가 V1 뷰를 기본으로 하며 WITH CASCADED CHECK OPTION으로 V2의 점검 옵션을 정의하는 경우 두 개의 뷰의 술어는 V2 뷰에 대해 INSERT 및 UPDATE문을 수행할 때 평가됩니다.

```
CREATE VIEW EMP_VIEW2 AS
  SELECT EMPNO, EMPNAME, DEPTNO FROM EMP
    WHERE DEPTNO = 10
WITH CHECK OPTION;
```

다음 예에는 WITH CASCADED CHECK OPTION을 사용하는 CREATE VIEW 문이 표시되어 있습니다. EMP_VIEW3 뷰는 WITH CHECK OPTION을 사용하여 작성된 EMP_VIEW2 뷰를 기본으로 하여 작성됩니다. EMP_VIEW3에 레코드를 삽입하거나 갱신하려면 해당 레코드에 DEPTNO=10 및 EMPNO=20 값이 있어야 합니다.

```
CREATE VIEW EMP_VIEW3 AS
  SELECT EMPNO, EMPNAME, DEPTNO FROM EMP_VIEW2
    WHERE EMPNO > 20
WITH CASCADED CHECK OPTION;
```

주: 조건 DEPTNO=10은 EMP_VIEW2에 WITH CHECK OPTION이 포함되어 있지 않은 경우에도 EMP_VIEW3에 조작을 삽입하거나 갱신하는 데 적용됩니다.

뷰 작성 시 WITH LOCAL CHECK OPTION도 지정할 수 있습니다. LOCAL CHECK OPTION을 사용하여 뷰를 정의하는 경우 뷰의 정의가 삽입 또는 갱신 조작 점검에 사용됩니다. 그러나 뷰는 해당 뷰가 종속되는 갱신 가능 뷰에서 검색 조건을 상속하지 않습니다.

삭제 가능한 뷰

뷰를 정의하는 방법에 따라 뷰를 삭제할 수 있습니다. 삭제 가능한 뷰는 DELETE문을 실행할 수 있는 대상 뷰입니다.

뷰가 삭제 가능한 것으로 간주되기 위해 따라야 하는 몇 가지 규칙이 있습니다.

- 외부 fullselect의 각 FROM절은 OUTER절이 없는 하나의 테이블, OUTER절이 없는 삭제 가능한 뷰, 삭제 가능한 중첩된 테이블 표현식 또는 삭제 가능한 공통 테이블 표현식을 식별합니다.
- 데이터베이스 관리 프로그램은 뷰 정의를 사용하여 테이블에서 삭제할 행을 추출할 수 있어야 합니다. 다음과 같은 특정 조작에서는 이 작업이 불가능합니다.

  - GROUP BY절 또는 컬럼 함수를 사용하여 여러 행을 그룹화하면 원래 행이 유실되어 뷰를 삭제할 수 없습니다.
  - 마찬가지로 VALUES에서 행이 파생되면 삭제할 테이블이 없습니다. 이 경우에도 뷰를 삭제할 수 없습니다.
- 외부 fullselect에서는 GROUP BY 또는 HAVING절을 사용하지 않습니다.
- 외부 fullselect에서는 선택 목록에 컬럼 함수가 포함되어 있지 않습니다.
- 외부 fullselect에서는 UNION ALL 이외의 집합 연산(UNION, EXCEPT 또는 INTERSECT)을 사용하지 않습니다.
- UNION ALL 피연산자의 테이블은 서로 같은 테이블이 아니어야 하며 각 피연산자는 삭제 가능해야 합니다.
- 외부 fullselect의 선택 목록에는 DISTINCT가 포함되지 않습니다.

뷰가 위에 나열된 모든 규칙을 충족해야 삭제 가능한 뷰입니다. 예를 들어, 다음 뷰는 삭제 가능합니다. 이 뷰는 삭제 가능한 뷰에 적용되는 모든 규칙을 따릅니다.

```
CREATE VIEW deletable_view
  (number, date, start, end)
AS      SELECT number, date, start, end
  FROM employee.summary
  WHERE date='01012007'
```

삽입 가능한 뷰

삽입 가능한 뷰를 사용하면 뷰 정의를 사용하여 행을 삽입할 수 있습니다. 삽입 조작의 INSTEAD OF 트리거가 뷰에 정의되어 있거나 하나 이상의 뷰의 컬럼이 갱신 가능하고(갱신과 관련하여 INSTEAD OF 트리거에 독립적) 뷰의 fullselect에 UNION ALL이 포함되어 있지 않은 경우 뷰가 삽입 가능합니다. 지정된 행이 하위 기본 테이블 중 하나의 점검 제한조건을 충족하는 경우에만 행을 뷰에 삽입할 수 있습니다(UNION ALL 포함). 갱신할 수 없는 컬럼이 있는 뷰에 삽입하려면 컬럼 목록에서 해당 컬럼을 생략해야 합니다.

아래 표시된 뷰는 삽입 가능한 뷰입니다. 그러나 이 예에서 뷰를 삽입하는 데 실패합니다. 이는 테이블에 널(NULL) 값을 승인하지 않는 컬럼이 있기 때문입니다. 이러한 컬럼 중 일부는 뷰 정의에 표시되지 않습니다. 뷰를 사용하여 값을 삽입하려고 하면 데이터베이스 관리 프로그램이 널(NULL) 값을 NOT NULL 컬럼에 삽입하려 합니다. 이 조치는 허용되지 않습니다.

```
CREATE VIEW insertable_view
  (number, name, quantity)
AS    SELECT number, name, quantify FROM ace.supplies
```

주: 테이블에 정의된 제한조건은 해당 테이블을 기본으로 뷰를 사용하여 수행할 수 있는 조작에 독립적입니다.

갱신 가능 뷰

갱신 가능 뷰는 특수한 삭제 가능 뷰입니다. 삭제 가능 뷰의 컬럼 중 하나 이상이 갱신 가능한 경우 삭제 가능 뷰는 갱신 가능 뷰가 됩니다.

다음 규칙이 모두 참인 경우 뷰의 컬럼을 갱신할 수 있습니다.

- 뷰를 삭제할 수 있습니다.
- 컬럼이 비참조 연산을 사용하지 않고 테이블의 컬럼이 되며 READ ONLY 옵션이 지정되어 있지 않습니다.
- 뷰의 fullselect에 UNION ALL이 포함된 경우, UNION ALL 피연산자의 모든 해당 컬럼에 정확하게 일치하는 데이터 유형(길이 또는 정밀도과 스케일 포함) 및 일치하는 디폴트값이 있습니다.

다음 예에서는 갱신할 수 없는 상수 값을 사용합니다. 그러나 뷰는 삭제 가능 뷰이며 하나 이상의 컬럼이 갱신 가능합니다. 따라서 이 뷰는 갱신 가능 뷰입니다.

```
CREATE VIEW updatable_view
  (number, current_date, current_time, temperature)
AS      SELECT number, CURRENT DATE, CURRENT TIME, temperature)
FROM weather.forecast
WHERE number = 300
```

읽기 전용 뷰

뷰를 삭제, 갱신 또는 삽입할 수 *없으면 읽기 전용* 뷰입니다. 뷰가 삭제 가능한 뷰에 적용되는 규칙 중 하나 이상을 준수하지 않는 경우 읽기 전용 뷰일 수 있습니다.

SYSCAT.VIEWS 카탈로그 뷰의 READONLY 컬럼은 뷰가 읽기 전용(R)임을 나타냅니다.

아래 예제에서는 뷰가 DISTINCT절을 사용하고 SQL문에 하나 이상의 테이블이 관련되어 있으므로 삭제 가능한 뷰가 아닙니다.

```
CREATE VIEW read_only_view
  (name, phone, address)
AS    SELECT DISTINCT viewname, viewphone, viewaddress
FROM employee.history adam, employer.dept sales
WHERE adam.id = sales.id
```

뷰 작성

뷰는 하나 이상의 테이블, 별칭 또는 뷰에서 파생되고 데이터 검색 시 테이블과 교환 가능하도록 사용될 수 있습니다. 뷰에서 표시된 데이터가 변경되면 해당 데이터는 테이블 자체에서도 변경됩니다. 뷰를 작성하려면 해당 뷰가 기반으로 하는 테이블, 별칭 또는 뷰가 있어야 합니다.

중요한 데이터에 대한 액세스를 제한하면서 동시에 다른 데이터에 대해서는 일반 액세스를 허용하도록 뷰를 작성할 수 있습니다.

뷰 정의의 선택 목록에 직접 또는 간접적으로 테이블의 식별 컬럼 이름이 포함되는 뷰에 삽입할 경우, INSERT문이 직접 테이블의 식별 컬럼을 참조할 때와 같은 규칙이 적용됩니다.

위에서 설명한 것처럼 뷰를 사용하는 것 외에도 다음과 같이 뷰를 사용할 수 있습니다.

- 응용프로그램에 영향을 주지 않으면서 테이블 변경. 기본 테이블을 바탕으로 뷰를 작성하면 가능합니다. 새 뷰를 작성해도 기본 테이블을 사용하는 응용프로그램이 영향을 받지 않습니다. 기본 테이블을 사용하는 응용프로그램과 다른 용도로 새 응용프로그램에서 작성된 뷰를 사용할 수 있습니다.
- 컬럼의 값 합산, 최대값 선택 또는 값의 평균 산출
- 하나 이상의 데이터 소스에서 정보에 대한 액세스 권한 제공. CREATE VIEW문 내에서 별칭을 참조하여 다중 위치/전역 뷰를 작성할 수 있습니다. 이러한 뷰는 다른 시스템에 있는 여러 데이터 소스의 정보를 조인할 수 있습니다.

  CREATE VIEW 구문을 사용하여 별칭을 참조하는 뷰를 작성하는 경우 뷰 사용자의 인증 ID가 뷰 작성자 인증 ID를 대신하여 데이터 소스의 기본 오브젝트에 액세스하는 데 사용될 수 있음을 알리는 경고가 표시됩니다. 이 경고를 표시하지 않으려면 FEDERATED 키워드를 사용하십시오.

유형이 지정된 뷰는 사전 정의되어 구조화된 유형을 바탕으로 작성됩니다. CREATE VIEW문을 사용하여 유형이 지정된 뷰를 작성할 수 있습니다.

뷰를 작성하는 다른 방법은 카탈로그 찾아보기 횟수를 줄이고 성능을 향상시키도록 중첩된 테이블 표현식 또는 공통 테이블 표현식을 사용하는 것입니다.

샘플 CREATE VIEW문은 아래와 같이 표시됩니다. 기본 테이블 EMPLOYEE에는 SALARY 및 COMM 컬럼이 있습니다. 보안상의 이유로 이 뷰는 ID, NAME, DEPT, JOB 및 HIREDATE 컬럼에서 작성됩니다. 또한 DEPT 컬럼에 대한 액세스가 제한됩니다. 이 정의는 DEPTNO가 0인 부서에 속한 직원의 정보만 표시합니다.

```
CREATE VIEW EMP_VIEW1
  (EMPID, EMPNAME, DEPTNO, JOBTITLE, HIREDATE)
  AS SELECT ID, NAME, DEPT, JOB, HIREDATE FROM EMPLOYEE
    WHERE DEPT=10;
```

뷰가 정의되면 액세스 특권을 지정할 수 있습니다. 이렇게 하면 테이블의 제한된 뷰에 액세스할 수 있으므로 데이터 보안을 보장할 수 있습니다. 위에 표시된 것처럼 뷰에는 특정 행에 대한 액세스를 제한하는 WHERE절이 포함될 수 있고 데이터의 특정 컬럼에 대한 액세스를 제한하는 컬럼의 서브세트가 포함될 수 있습니다.

뷰의 컬럼 이름은 기본 테이블의 컬럼 이름과 일치하지 않습니다. 뷰 이름과 마찬가지로 테이블 이름에는 연관된 스키마가 있습니다.

뷰가 정의되면 제한사항과 함께 SELECT, INSERT UPDATE 및 DELETE와 같은 명령문에 사용할 수 있습니다. DBA는 사용자 그룹에게 테이블보다 뷰에 대해 더 높은 레벨의 특권을 제공하도록 결정할 수 있습니다.

사용자 정의 함수(UDF)를 사용하는 뷰 작성

UDF를 사용하는 뷰를 작성하면 이후에 동일한 이름을 사용하여 다른 UDF를 작성하더라도 해당 뷰가 존재하는 이 뷰에서는 항상 동일한 UDF를 사용합니다. 새 UDF를 선택하려면 뷰를 다시 작성해야 합니다.

다음 SQL문은 정의에 함수가 포함된 뷰를 작성합니다.

```
CREATE VIEW EMPLOYEE_PENSION (NAME, PENSION)
  AS SELECT NAME, PENSION(HIREDATE,BIRTHDATE,SALARY,BONUS)
  FROM   EMPLOYEE
```

UDF 함수인 PENSION은 직원의 HIREDATE, BIRTHDATE, SALARY 및 BONUS와 연관된 공식을 바탕으로 직원이 받을 수 있는 현재 연금을 계산합니다.

유형이 지정된 뷰 수정

뷰를 삭제하거나 다시 작성할 필요 없이 유형이 지정된 뷰의 특정 등록 정보를 변경할 수 있습니다. 그러한 등록 정보 중 하나는 유형이 지정된 뷰의 참조 컬럼에 범위를 추가하는 것입니다.

ALTER VIEW문은 범위를 추가하도록 참조 유형 컬럼을 변경하여 기존의 유형이 지정된 뷰 정의를 수정합니다. DROP문은 유형이 지정된 뷰를 삭제합니다. 또한 다음을 수행할 수도 있습니다.

- INSTEAD OF 트리거를 통해 유형이 지정된 뷰의 컨텐츠 수정
- 통계 콜렉션을 사용 가능하도록 유형이 지정된 뷰 변경

유형이 지정된 뷰의 기본 컨텐츠를 변경하려면 트리거를 사용해야 합니다. 유형이 지정된 뷰의 다른 부분을 변경하려면 유형이 지정된 뷰를 삭제한 다음 다시 작성해야 합니다.

ALTER VIEW문의 column-name의 데이터 유형은 REF(유형이 지정된 테이블 이름 또는 유형이 지정된 뷰 이름의 유형)여야 합니다.

패키지 및 캐시된 동적문이 유효하지 않은 것으로 표시되더라도 테이블 및 인덱스와 같은 다른 데이터베이스 오브젝트는 영향을 받지 않습니다.

명령행을 사용하여 유형이 지정된 뷰를 변경하려면 다음과 같이 입력하십시오.

```
ALTER VIEW <view_name> ALTER <column_name>
   ADD SCOPE <typed table or view name>
```

작동 불능 뷰 복구

작동 불능 뷰는 SQL문에 대해 더 이상 사용할 수 없는 뷰입니다.

다음과 같이 뷰는 작동 불능 상태가 될 수 있습니다.

- 기본 테이블에서 특권에 대한 권한 취소의 결과
- 테이블, 별명 또는 함수가 삭제된 경우
- 수퍼 뷰가 작동 불능 상태가 된 경우. 수퍼 뷰는 유형이 지정된 다른 뷰, 서브뷰의 바탕이 되는 유형이 지정된 뷰입니다.
- 해당 뷰가 종속된 뷰가 삭제된 경우

다음 단계를 수행하여 작동 불능 뷰를 복구할 수 있습니다.

1. 처음에 뷰 작성에 사용된 SQL문을 판별합니다. SYSCAT.VIEW 카탈로그 뷰의 TEXT 컬럼에서 이러한 정보를 얻을 수 있습니다.
2. 현재 스키마를 QUALIFIER 컬럼의 컨텐츠로 설정합니다.
3. 함수 경로를 FUNC_PATH 컬럼의 컨텐츠로 설정합니다.
4. 동일한 뷰 이름 및 정의와 함께 CREATE VIEW문을 사용하여 뷰를 다시 작성합니다.
5. GRANT문을 사용하여 이전에 뷰에 부여된 모든 특권을 다시 부여합니다. 작동 불능 뷰에 부여된 모든 특권이 취소됩니다.

작동 불능 뷰를 복구하지 않으려면 DROP VIEW문을 사용하여 해당 뷰를 명시적으로 삭제하거나 이름은 동일하지만 정의는 다른 새로운 뷰를 작성할 수 있습니다.

작동 불능 뷰에는 SYSCAT.TABLES 및 SYSCAT.VIEWS 카탈로그 뷰의 항목만 포함될 수 있습니다. SYSCAT.TABDEP, SYSCAT.TABAUTH, SYSCAT.COLUMNS 및 SYSCAT.COLAUTH 카탈로그 뷰의 모든 항목은 제거됩니다.

뷰 삭제

뷰를 삭제하려면 DROP VIEW문을 사용합니다. 삭제된 뷰에 종속된 모든 뷰는 작동 불능 상태가 됩니다.

명령행을 사용하여 뷰를 삭제하려면 다음과 같이 입력하십시오.

```
DROP VIEW <view_name>
```

다음 예는 EMP_VIEW 뷰를 삭제하는 방법을 보여줍니다.

```
DROP VIEW EMP_VIEW
```

테이블 계층에서 다음 예에서와 같이 계층의 루트 뷰의 이름을 지정하여 하나의 명령문으로 전체 뷰 계층을 삭제할 수 있습니다.

```
DROP VIEW HIERARCHY VPerson
```

제 4 부 참조

제 17 장 이름 지정 규칙 준수

이름 지정 규칙

모든 데이터베이스 오브젝트, 사용자, 그룹, 파일 및 경로의 이름을 지정하는 데 적용되는 규칙이 있습니다. 이러한 규칙 중 일부는 사용자가 작업 중인 플랫폼에 따라 다릅니다.

예를 들면, 이름에서 대소문자 사용과 관련된 규칙이 있습니다.

- UNIX 플랫폼의 경우 이름은 소문자여야 합니다.
- Windows 플랫폼의 경우 이름에 대문자, 소문자 및 대소문자를 혼용할 수 있습니다.

달리 지정되지 않는 한, 모든 이름에 다음 문자를 사용할 수 있습니다.

- A - Z. 대부분의 이름에서 사용될 때, A에서 Z까지의 문자는 소문자에서 대문자로 변환됩니다.
- 0 - 9
- ! % ( ) { } . – ^ ~ _(밑줄) @, #, $ 및 스페이스
- ₩(백슬래시)

이름은 숫자 또는 밑줄 문자로 시작될 수 없습니다.

테이블, 뷰, 컬럼, 인덱스 또는 권한 부여 ID의 이름을 지정하는 데 SQL 예약어를 사용하지 마십시오.

운영 체제 및 DB2 데이터베이스에 대해 작업 중인 위치에 따라 개별적으로 사용할 수 있는 기타 특수 문자가 있습니다. 그러나 특수 문자 사용이 가능하지만 보장되지는 않습니다. 데이터베이스에서 오브젝트 이름 지정 시 이러한 기타 특수 문자를 사용하지 않는 것이 좋습니다.

사용자 및 그룹 이름도 관련 시스템을 통해 특정 운영 체제에 적용되는 규칙을 따라야 합니다. 예를 들어, Linux 및 UNIX 플랫폼의 경우, 사용자 이름과 기본 그룹 이름에 사용할 수 있는 문자는 소문자 a - z, 0 - 9 및 _(밑줄)이어야 하고 이름은 0 - 9로 시작되지 않습니다.

길이는 *SQL 참조서*에 있는 『SQL 및 XML 한계』에 나열된 길이 이하여야 합니다.

오브젝트 이름 지정 규칙, NLS 환경의 이름 지정 규칙 및 유니코드 환경의 이름 지정 규칙도 고려해야 합니다.

AUTHID ID에 대한 제한사항: 버전 9.5 이상의 DB2 데이터베이스 시스템에서는 128 바이트 권한 부여 ID를 사용할 수 있지만 권한 부여 ID가 운영 체제 사용자 ID 또는 그룹 이름으로 해석되는 경우에는 운영 체제 이름 지정 제한사항이 적용됩니다. 예를 들어, 사용자 ID 및 그룹 이름 길이는 Linux 및 UNIX 운영 체제에서는 8자, Windows 운영 체제에서는 30자로 제한됩니다. 따라서 128바이트 권한 부여 ID를 부여할 수 있지만 해당 권한 부여 ID를 가진 사용자로 연결할 수는 없습니다. 사용자 고유의 보안 플러그인을 작성하는 경우 확장된 권한 부여 ID의 크기로 인한 이점을 활용할 수 있습니다. 예를 들어, 사용자는 보안 플러그인에 30바이트 사용자 ID를 지정할 수 있으며 보안 플러그인은 사용자가 연결할 수 있는 인증 중에 128바이트 권한 부여 ID를 리턴할 수 있습니다.

DB2 오브젝트 이름 지정 규칙

모든 오브젝트는 일반 이름 지정 규칙을 따릅니다. 또한 일부 오브젝트의 추가적인 제한사항은 아래 표를 참조하십시오.

표 23. 데이터베이스, 데이터베이스 별명 및 인스턴스 이름 지정 규칙

오브젝트	가이드라인
• 데이터베이스 • 데이터베이스 별명 • 인스턴스	• 데이터베이스 이름은 카탈로그된 위치 내에서 고유해야 합니다. Linux 및 UNIX 구현에서는 이 위치가 디렉토리 경로이지만, Windows 구현에서는 논리적 디스크입니다. • 데이터베이스 별명은 시스템 데이터베이스 디렉토리 내에서 고유해야 합니다. 새 데이터베이스가 작성되면 별명이 디폴트로 데이터베이스 이름으로 설정됩니다. 따라서 데이터베이스 별명으로 존재하는 이름을 가진 데이터베이스가 없더라도 해당 이름을 사용하여 데이터베이스를 작성할 수 없습니다. • 데이터베이스, 데이터베이스 별명 및 인스턴스 이름의 길이는 8바이트 이하여야 합니다. • Windows에서는 인스턴스에 서비스 이름과 동일한 이름을 사용할 수 없습니다. **주:** 통신 환경에서 데이터베이스를 사용하려는 경우 잠재적인 문제점이 발생하지 않도록 하려면 데이터베이스 이름에 특수 문자 @, # 및 $를 사용하지 마십시오. 또한, 이들 문자는 모든 키보드에서 공통적인 문자가 아니므로 데이터베이스를 다른 언어로 사용하려는 경우에는 이 문자를 사용하지 마십시오.

표 24. 데이터베이스 오브젝트 이름 지정 규칙

오브젝트	가이드라인
• 별명 • 감사 규정 • 버퍼 풀 • 컬럼 • 이벤트 모니터 • 인덱스 • 메소드 • 노드 그룹 • 패키지 • 패키지 버전 • 역할 • 스키마 • 스토어드 프로시저 • 테이블 • 테이블 스페이스 • 트리거 • 트러스트된 컨텍스트 • UDF • UDT • 뷰	이들 오브젝트의 길이는 *SQL 참조서*의 『SQL 및 XML 한계』에 나열된 길이 이하여야 합니다. 오브젝트 이름에 다음 문자도 사용할 수 있습니다. • 유효한 강조 문자(예: ö) • 멀티바이트 스페이스를 제외한 멀티바이트 문자(멀티바이트 환경의 경우) 패키지 이름 및 패키지 버전에서 마침표(.), 하이픈(-) 및 콜론(:)도 사용할 수 있습니다.

표 25. 페더레이티드 데이터베이스 오브젝트 이름 지정 규칙

오브젝트	가이드라인
• 함수 맵핑 • 인덱스 스펙 • 별칭 • 서버 • 유형 맵핑 • 사용자 맵핑 • 랩퍼	이들 오브젝트의 길이는 *SQL 참조서* 의 『SQL 및 XML 한계』에 나열된 길이 이하여야 합니다. 페더레이티드 데이터베이스 오브젝트의 이름에 다음 문자도 사용할 수 있습니다. • 유효한 강조 문자(예: ö) • 멀티바이트 스페이스를 제외한 멀티바이트 문자(멀티바이트 환경의 경우)

분리 ID 및 오브젝트 이름

키워드를 사용할 수 있습니다. 키워드를 SQL 키워드로도 해석할 수 있는 컨텍스트에서 키워드가 사용되는 경우 키워드를 분리 ID로 지정해야 합니다.

분리 ID를 사용하면 해당 이름 지정 규칙에 위배되는 오브젝트를 작성할 수 있습니다. 그러나 연속적으로 이러한 오브젝트를 사용하면 오류가 발생할 수 있습니다. 예를 들어, 이름에 + 또는 - 기호가 포함된 컬럼을 작성하고 연속적으로 해당 컬럼을 인덱스에서 사용하는 경우 테이블을 재구성하려 하면 문제가 발생합니다.

추가적인 스키마 이름 정보

- 사용자 정의 유형(UDT)은 *SQL 참조서*의 『SQL 및 XML 한계』에 나열된 길이보다 긴 스키마 이름을 가질 수 없습니다.
- 스키마 이름 SYSCAT, SYSFUN, SYSIBM, SYSSTAT, SYSPUBLIC은 예약어이므로 사용해서는 안 됩니다.
- 나중에 데이터베이스를 업그레이드할 때 문제점이 발생하지 않도록 하려면 SYS로 시작하는 스키마 이름을 사용하지 마십시오. 데이터베이스 관리 프로그램에서는 SYS로 시작되는 스키마 이름을 사용하여 트리거, 사용자 정의 유형 또는 사용자 정의 함수를 작성할 수 없습니다.
- SESSION을 스키마 이름으로 사용하지 않는 것이 좋습니다. 선언된 임시 테이블을 SESSION으로 규정해야 합니다. 따라서 응용프로그램에서 영구 테이블의 이름과 동일한 이름으로 임시 테이블을 선언할 수 있습니다. 그럴 경우 응용프로그램 논리가 지나치게 복잡해집니다. 선언된 임시 테이블을 처리할 때를 제외하고는 SESSION 스키마를 사용하지 마십시오.

분리 ID 및 오브젝트 이름

키워드를 사용할 수 있습니다. 키워드를 SQL 키워드로도 해석할 수 있는 컨텍스트에서 키워드가 사용되는 경우 키워드를 분리 ID로 지정해야 합니다.

분리 ID를 사용하면 해당 이름 지정 규칙에 위배되는 오브젝트를 작성할 수 있습니다. 그러나 연속적으로 이러한 오브젝트를 사용하면 오류가 발생할 수 있습니다. 예를 들어, 이름에 + 또는 - 기호가 포함된 컬럼을 작성하고 연속적으로 해당 컬럼을 인덱스에서 사용하는 경우 테이블을 재구성하려 하면 문제가 발생합니다.

사용자, 사용자 ID 및 그룹 이름 지정 규칙

사용자, 사용자 ID 및 그룹 이름은 이름 지정 규칙을 따라야 합니다.

표 26. 사용자, 사용자 ID 및 그룹 이름 지정 규칙

오브젝트	가이드라인
• 그룹 이름 • 사용자 이름 • 사용자 ID	• 그룹 이름은 *SQL 참조서* 의『SQL 및 XML 한계』에 나열된 그룹 이름 길이 이하여야 합니다. • Linux and UNIX 운영 체제의 사용자 ID에는 최대 8자를 사용할 수 있습니다. • Windows의 사용자 이름에는 최대 30자를 사용할 수 있습니다. • 클라이언트 인증을 사용하지 않을 때, *SQL 참조서* 의『SQL 및 XML 한계』에 나열되는 사용자 이름 길이보다 긴 사용자 이름을 사용하여 Windows에 연결하는 비Windows 32비트 클라이언트는 사용자 이름과 암호가 명시적으로 지정될 때 지원됩니다. • 이름 및 ID에는 다음을 사용할 수 없습니다. – USERS, ADMINS, GUESTS, PUBLIC, LOCAL 또는 모든 SQL 예약어를 사용할 수 없습니다. – IBM, SQL 또는 SYS로 시작할 수 없습니다.

주:

1. 일부 운영 체제에서는 대소문자를 구분하는 사용자 ID 및 암호를 사용할 수 있습니다. 대소문자 구분 사용자 ID 및 암호를 사용할 수 있는지 확인하려면 운영 체제 문서를 확인해야 합니다.
2. 성공적인 CONNECT 또는 ATTACH로부터 리턴되는 권한 부여 ID는 *SQL 참조서*의『SQL 및 XML 한계』에 나열되는 권한 부여 이름으로 절단됩니다. 생략(...) 기호가 권한 부여 ID에 추가되고 SQLWARN 필드에는 절단을 표시하는 경고가 포함됩니다.
3. 사용자 ID 및 암호의 뒤 공백은 제거됩니다.

NLS 환경의 이름 지정 규칙

데이터베이스 이름에서 사용할 수 있는 기본 문자 세트는 1바이트 대소문자 라틴어 문자(A...Z, a...z), 아라비아 숫자(0...9) 및 밑줄 문자(_)입니다.

이 목록은 3개의 특수 문자(#, @ 및 $)가 보강되어 호스트 데이터베이스 제품과 호환할 수 있도록 합니다. 특수 문자 #, @ 및 $는 NLS 호스트(EBCDIC) 불변 문자 세트에 포함되어 있지 않으므로 NLS 환경에서는 주의하여 이들 문자를 사용하십시오. 사용 중인 코드 페이지에 따라서 확장 문자 세트의 문자도 사용할 수 있습니다. 다중 코드 페이지 환경에서 데이터베이스를 사용 중인 경우 모든 코드 페이지가 사용하려는 확장 문자 세트의 모든 요소를 지원하는지 확인해야 합니다.

테이블 및 뷰와 같은 데이터베이스 오브젝트 이름을 지정할 경우 프로그램 레이블, 호스트 변수, 커서 및 확장 문자 세트의 요소(예: 분음 기호가 있는 문자)도 사용할 수 있습니다. 정확히 어떤 문자를 사용할 수 있는지 여부는 사용 중인 코드 페이지에 따라 다릅니다.

DBCS ID의 확장 문자 세트 정의: DBCS 환경에서 확장 문자 세트는 기본 문자 세트에 있는 모든 문자와 다음 문자로 구성됩니다.

- 각 DBCS 코드 페이지에 있는 모든 2바이트 문자(2바이트 스페이스 제외)는 유효한 문자입니다.
- 2바이트 스페이스는 특수 문자입니다.
- 각 혼합 코드 페이지에서 사용 가능한 1바이트 문자는 다음과 같은 여러 범주에 지정됩니다.

범주	각 혼합 코드 페이지에서 유효한 코드 포인트
숫자	x30-39
문자	x23-24, x40-5A, x61-7A, xA6-DF(A6-DF는 코드 페이지 932 및 942 전용)
특수 문자	기타 모든 유효 1바이트 문자 코드 포인트

유니코드 환경의 이름 지정 규칙

유니코드 데이터베이스에서는 모든 ID가 멀티바이트 UTF-8입니다. 따라서 DB2 데이터베이스 시스템이 확장 문자 세트(예: 강조 문자 또는 멀티바이트 문자)의 문자를 사용하도록 허용하는 ID에 UCS-2 문자를 사용하는 것이 가능합니다.

클라이언트는 해당 환경에서 지원되는 모든 문자를 입력할 수 있으며 ID의 모든 문자는 데이터베이스 관리 프로그램을 통해 UTF-8로 변환됩니다. 유니코드 데이터베이스의 ID에 자국어 문자를 지정할 경우 다음과 같은 두 가지 사항을 유념해야 합니다.

- ASCII가 아닌 각 문자에는 2 - 4바이트가 필요합니다. 따라서 *n*바이트 ID에는 ASCII 대 비ASCII 문자의 비율에 따라 *n*/4 - *n*개의 문자만 사용할 수 있습니다. 비ASCII 문자가 한 두 개뿐인 경우에는(예: 강조 문자) *n*개에 가까운 문자를 사용할 수 있지만, 전체가 비ASCII 문자인 ID(예: 일본어)에서는 *n*/4 - *n*/3개의 문자만 사용할 수 있습니다.
- 다른 클라이언트 환경에서 ID를 입력할 경우 해당 클라이언트에 사용 가능한 문자의 일반 서브세트를 사용하여 ID를 정의해야 합니다. 예를 들어, 라틴-1, 아랍어 및 일본어 환경에서 유니코드 데이터베이스에 액세스할 경우 모든 ID에는 실질적으로 ASCII만 사용되어야 합니다.

제 18 장 LDAP(Lightweight Directory Access Protocol)

LDAP(Lightweight Directory Access Protocol)은 디렉토리 서비스에 대한 산업 규격 액세스 메소드입니다. 디렉토리 서비스는 분산 환경에 있는 다중 시스템 및 서비스에 대한 자원 정보 저장소이며, 해당 자원에 대한 클라이언트 및 서버 액세스를 제공합니다.

각 데이터베이스 서버 인스턴스는 자신의 존재를 LDAP 서버에 발행하고 데이터베이스가 작성되는 경우 LDAP 디렉토리에 데이터베이스 정보를 제공합니다. 클라이언트가 데이터베이스에 연결되면 서버의 카탈로그 정보를 LDAP 디렉토리에서 검색할 수 있습니다. 각 머신에서 로컬로 카탈로그 정보를 저장하는 데 각각의 클라이언트가 더 이상 필요하지 않습니다. 클라이언트 응용프로그램은 LDAP 디렉토리에서 데이터베이스에 연결하는 데 필요한 정보를 검색합니다.

클라이언트가 LDAP Directory Server를 한 번만 검색하면 되도록 캐싱 메커니즘이 존재합니다. LDAP Directory Server에서 정보를 검색하면 *dir_cache* 데이터베이스 관리 프로그램 구성 매개변수 및 DB2LDAPCACHE 레지스트리 변수의 값에 따라 로컬 컴퓨터에서 해당 정보가 저장되거나 캐시됩니다. 메모리 캐시에 데이터베이스, 노드 및 DCS 디렉토리 파일을 저장하는 데 *dir_cache* 데이터베이스 관리 프로그램 구성 매개변수가 사용됩니다. 응용프로그램이 닫힐 때까지 응용프로그램에서 디렉토리 캐시가 사용됩니다. 로컬 디스크 캐시에 데이터베이스, 노드 및 DCS 디렉토리 파일을 저장하는 데 DB2LDAPCACHE 레지스트리 변수가 사용됩니다.

- DB2LDAPCACHE=NO이고 *dir_cache*=NO인 경우에는 항상 LDAP에서 정보를 읽습니다.
- DB2LDAPCACHE=NO이고 *dir_cache*=YES인 경우에는 LDAP에서 정보를 한 번 읽은 다음 이를 DB2 캐시에 삽입합니다.
- DB2LDAPCACHE=YES이거나 설정되지 않은 경우에는 LDAP에서 정보를 한 번 읽은 다음 이를 로컬 데이터베이스, 노드 및 DCS 디렉토리에 캐시합니다.

주: DB2LDAPCACHE 레지스트리 변수는 데이터베이스 및 노드 디렉토리에만 적용할 수 있습니다.

LDAP 환경의 보안 고려사항

LDAP 디렉토리의 정보에 액세스하기 전에 LDAP 서버에서 응용프로그램 또는 사용자를 인증합니다. 인증 프로세스를 LDAP 서버에 *바인딩*한다고 합니다. LDAP 디렉토리에 저장된 정보에 액세스 제어를 적용하여 익명의 사용자가 정보를 추가, 삭제 또는 수정하지 못하도록 하는 것이 중요합니다.

액세스 제어는 디폴트로 상속되고 컨테이너 레벨에서 적용할 수 있습니다. 새 오브젝트가 작성되면 해당 오브젝트는 상위 오브젝트와 동일한 보안 속성을 상속합니다. LDAP 서버에서 사용 가능한 관리 도구를 사용하여 컨테이너 오브젝트에 대한 액세스 제어를 정의할 수 있습니다.

디폴트로 다음과 같이 액세스 제어를 정의합니다.

- LDAP의 데이터베이스 및 노드 엔트리의 경우 모든 사용자(또는 모든 익명 사용자)에게 읽기 액세스 권한이 있습니다. 디렉토리 관리자 및 오브젝트 소유자 또는 작성자에게만 읽기/쓰기 액세스 권한이 있습니다.
- 사용자 프로파일의 경우 프로파일 소유자와 디렉토리 관리자에게만 읽기/쓰기 액세스 권한이 있습니다. 사용자에게 디렉토리 관리자 권한이 없는 경우에는 다른 사용자의 프로파일에 액세스할 수 없습니다.

주: 권한 점검은 항상 DB2가 아니라 LDAP 서버를 통해 수행됩니다. LDAP 권한 점검은 DB2 권한과 관련이 없습니다. SYSADM 권한을 가진 계정 또는 권한 부여 ID는 LDAP 디렉토리에 액세스할 수 없습니다.

LDAP 명령 또는 API를 실행할 때, 바인드 식별 이름(bindDN) 및 암호가 지정되지 않은 경우, DB2는 요청된 명령을 수행하는데 충분한 권한이 없을 수 있는 디폴트 증명서를 사용하여 LDAP 서버에 바인드하고 오류를 리턴합니다.

DB2 명령 또는 API에서 USER 및 PASSWORD절을 사용하여 사용자의 bindDN과 암호를 명시적으로 지정할 수 있습니다.

DB2에서 사용되는 LDAP 오브젝트 클래스 및 속성

다음 표는 DB2 데이터베이스 관리 프로그램에서 사용되는 오브젝트 클래스에 대해 설명합니다.

표 27. cimManagedElement

클래스	**cimManagedElement**
Active Directory LDAP 표시 이름	해당되지 않음
Active Directory CN(공통 이름)	해당되지 않음
설명	IBM 스키마에서 여러 시스템 관리 오브젝트 클래스의 기본 클래스를 제공합니다.
SubClassOf	top
필수 속성	
선택적 속성	설명
유형	추상
OID(오브젝트 ID)	1.3.18.0.2.6.132
GUID(고유한 전역 ID)	b3afd63f-5c5b-11d3-b818-002035559151

표 28. *cimSetting*

클래스	**cimSetting**
Active Directory LDAP 표시 이름	해당되지 않음
Active Directory CN(공통 이름)	해당되지 않음
설명	IBM 스키마의 구성 및 설정에 기본 클래스를 제공합니다.
SubClassOf	cimManagedElement
필수 속성	
선택적 속성	settingID
유형	추상
OID(오브젝트 ID)	1.3.18.0.2.6.131
GUID(고유한 전역 ID)	b3afd64d-5c5b-11d3-b818-002035559151

표 29. *eProperty*

클래스	**eProperty**
Active Directory LDAP 표시 이름	ibm-eProperty
Active Directory CN(공통 이름)	ibm-eProperty
설명	사용자 환경 설정 등록 정보에 대한 응용프로그램 관련 설정을 지정하는 데 사용됩니다.
SubClassOf	cimSetting
필수 속성	
선택적 속성	propertyType cisPropertyType cisProperty cesPropertyType cesProperty binPropertyType binProperty
유형	구조적
OID(오브젝트 ID)	1.3.18.0.2.6.90
GUID(고유한 전역 ID)	b3afd69c-5c5b-11d3-b818-002035559151

표 30. *DB2Node*

클래스	**DB2Node**
Active Directory LDAP 표시 이름	ibm-db2Node
Active Directory CN(공통 이름)	ibm-db2Node
설명	DB2 Server를 나타냅니다.
SubClassOf	eSap/ServiceConnectionPoint
필수 속성	db2nodeName

표 30. DB2Node (계속)

클래스	**DB2Node**
선택적 속성	db2nodeAlias db2instanceName db2Type host/dNSHostName (see Note 2) protocolInformation/ServiceBindingInformation
유형	구조적
OID(오브젝트 ID)	1.3.18.0.2.6.116
GUID(고유한 전역 ID)	b3afd65a-5c5b-11d3-b818-002035559151
특별 참고사항	1. *DB2Node* 클래스는 IBM Tivoli® Directory Server 아래의 *eSap* 오브젝트 클래스 및 Microsoft Active Directory 아래의 *ServiceConnectionPoint* 오브젝트 클래스에서 파생됩니다. 2. 호스트는 IBM Tivoli Directory Server 환경에서 사용됩니다. *dNSHostName* 속성은 Microsoft Active Directory 아래에서 사용됩니다. 3. *protocolInformation*은 IBM Tivoli Directory Server 환경에서만 사용됩니다. Microsoft Active Directory의 경우 ServiceConnectionPoint에서 상속된 *ServiceBindingInformation* 속성은 프로토콜 정보를 포함하는 데 사용됩니다.

DB2Node 오브젝트의 *protocolInformation* 속성(IBM Tivoli Directory Server) 또는 *ServiceBindingInformation* 속성(Microsoft Active Directory)에는 DB2 데이터베이스 서버를 바인드하기 위한 통신 프로토콜 정보가 들어 있습니다. 이러한 속성은 지원되는 네트워크 프로토콜에 대해 설명하는 토큰으로 구성됩니다. 각 토큰은 세미콜론으로 구분됩니다. 토큰 사이에는 스페이스가 없어야 합니다. 별표(*)는 선택적 매개변수를 지정하는데 사용될 수 있습니다.

TCP/IP의 토큰은 다음과 같습니다.

- 『TCPIP』
- 서버 호스트 이름 또는 IP 주소
- 서비스 이름(svcename) 또는 포트 번호(예: 50000)
- (선택적) 보안(『NONE』 또는 『SOCKS』)

Named Pipe의 토큰은 다음과 같습니다.

- 『NPIPE』
- 서버의 컴퓨터 이름
- 서버의 인스턴스 이름

표 31. DB2Database

클래스	**DB2Database**
Active Directory LDAP 표시 이름	ibm-db2Database
Active Directory CN(공통 이름)	ibm-db2Database
설명	DB2 데이터베이스를 나타냅니다.
SubClassOf	top
필수 속성	db2databaseName db2nodePtr
선택적 속성	db2databaseAlias db2additionalParameter db2ARLibrary db2authenticationLocation db2gwPtr db2databaseRelease DCEPrincipalName db2altgwPtr db2altnodePtr
유형	구조적
OID(오브젝트 ID)	1.3.18.0.2.6.117
GUID(고유한 전역 ID)	b3afd659-5c5b-11d3-b818-002035559151

표 32. db2additionalParameters

속성	**db2additionalParameters**
Active Directory LDAP 표시 이름	ibm-db2AdditionalParameters
Active Directory CN(공통 이름)	ibm-db2AdditionalParameters
설명	호스트 데이터베이스 서버에 연결되는 경우 사용되는 추가 매개변수가 들어 있습니다.
구문	Case Ignore 문자열
최대 길이	1024
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.426
GUID(고유한 전역 ID)	b3afd315-5c5b-11d3-b818-002035559151

표 33. db2authenticationLocation

속성	**db2authenticationLocation**
Active Directory LDAP 표시 이름	ibm-db2AuthenticationLocation
Active Directory CN(공통 이름)	ibm-db2AuthenticationLocation
설명	인증이 수행되는 위치를 지정합니다.
구문	Case Ignore 문자열

표 33. *db2authenticationLocation (계속)*

속성	**db2authenticationLocation**
최대 길이	64
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.425
GUID(고유한 전역 ID)	b3afd317-5c5b-11d3-b818-002035559151
Notes®	올바른 값: CLIENT, SERVER, DCS, DCE, KERBEROS, SVRENCRYPT 또는 DCSENCRYPT

표 34. *db2ARLibrary*

속성	**db2ARLibrary**
Active Directory LDAP 표시 이름	ibm-db2ARLibrary
Active Directory CN(공통 이름)	ibm-db2ARLibrary
설명	응용프로그램 리퀘스터 라이브러리 이름입니다.
구문	Case Ignore 문자열
최대 길이	256
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.427
GUID(고유한 전역 ID)	b3afd316-5c5b-11d3-b818-002035559151

표 35. *db2databaseAlias*

속성	**db2databaseAlias**
Active Directory LDAP 표시 이름	ibm-db2DatabaseAlias
Active Directory CN(공통 이름)	ibm-db2DatabaseAlias
설명	데이터베이스 별명 이름
구문	Case Ignore 문자열
최대 길이	1024
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.422
GUID(고유한 전역 ID)	b3afd318-5c5b-11d3-b818-002035559151

표 36. *db2databaseName*

속성	**db2databaseName**
Active Directory LDAP 표시 이름	ibm-db2DatabaseName
Active Directory CN(공통 이름)	ibm-db2DatabaseName
설명	데이터베이스 이름
구문	Case Ignore 문자열
최대 길이	1024
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.421
GUID(고유한 전역 ID)	b3afd319-5c5b-11d3-b818-002035559151

표 37. *db2databaseRelease*

속성	**db2databaseRelease**
Active Directory LDAP 표시 이름	ibm-db2DatabaseRelease
Active Directory CN(공통 이름)	ibm-db2DatabaseRelease
설명	데이터베이스 릴리스 번호
구문	Case Ignore 문자열
최대 길이	64
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.429
GUID(고유한 전역 ID)	b3afd31a-5c5b-11d3-b818-002035559151

표 38. *db2nodeAlias*

속성	**db2nodeAlias**
Active Directory LDAP 표시 이름	ibm-db2NodeAlias
Active Directory CN(공통 이름)	ibm-db2NodeAlias
설명	노드 별명 이름
구문	Case Ignore 문자열
최대 길이	1024
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.420
GUID(고유한 전역 ID)	b3afd31d-5c5b-11d3-b818-002035559151

표 39. *db2nodeName*

속성	**db2nodeName**
Active Directory LDAP 표시 이름	ibm-db2NodeName
Active Directory CN(공통 이름)	ibm-db2NodeName
설명	노드 이름
구문	Case Ignore 문자열
최대 길이	64
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.419
GUID(고유한 전역 ID)	b3afd31e-5c5b-11d3-b818-002035559151

표 40. *db2nodePtr*

속성	**db2nodePtr**
Active Directory LDAP 표시 이름	ibm-db2NodePtr
Active Directory CN(공통 이름)	ibm-db2NodePtr
설명	데이터베이스를 소유한 데이터베이스 서버를 나타내는 노드(DB2Node) 오브젝트를 가리키는 포인터입니다.
구문	식별 이름
최대 길이	1000
여러 값 지정	하나의 값 지정

표 40. db2nodePtr (계속)

속성	db2nodePtr
OID(오브젝트 ID)	1.3.18.0.2.4.423
GUID(고유한 전역 ID)	b3afd31f-5c5b-11d3-b818-002035559151
특별 참고사항	이러한 관계는 데이터베이스에 연결하기 위해 프로토콜 통신 정보를 검색하도록 허용합니다.

표 41. db2altnodePtr

속성	db2altnodePtr
Active Directory LDAP 표시 이름	ibm-db2AltNodePtr
Active Directory CN(공통 이름)	ibm-db2AltNodePtr
설명	대체 데이터베이스 서버를 나타내는 노드(DB2Node) 오브젝트를 가리키는 포인터입니다.
구문	식별 이름
최대 길이	1000
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.3093
GUID(고유한 전역 ID)	5694e266-2059-4e32-971e-0778909e0e72

표 42. db2gwPtr

속성	db2gwPtr
Active Directory LDAP 표시 이름	ibm-db2GwPtr
Active Directory CN(공통 이름)	ibm-db2GwPtr
설명	데이터베이스에 액세스할 수 있는 게이트웨이 서버를 나타내는 노드 오브젝트를 가리키는 포인터입니다.
구문	식별 이름
최대 길이	1000
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.424
GUID(고유한 전역 ID)	b3afd31b-5c5b-11d3-b818-002035559151

표 43. db2altgwPtr

속성	db2altgwPtr
Active Directory LDAP 표시 이름	ibm-db2AltGwPtr
Active Directory CN(공통 이름)	ibm-db2AltGwPtr
설명	대체 게이트웨이 서버를 나타내는 노드 오브젝트를 가리키는 포인터입니다.
구문	식별 이름
최대 길이	1000
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.3092
GUID(고유한 전역 ID)	70ab425d-65cc-4d7f-91d8-084888b3a6db

표 44. *db2instanceName*

속성	**db2instanceName**
Active Directory LDAP 표시 이름	ibm-db2InstanceName
Active Directory CN(공통 이름)	ibm-db2InstanceName
설명	데이터베이스 서버 인스턴스의 이름입니다.
구문	Case Ignore 문자열
최대 길이	256
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.428
GUID(고유한 전역 ID)	b3afd31c-5c5b-11d3-b818-002035559151

표 45. *db2Type*

속성	**db2Type**
Active Directory LDAP 표시 이름	ibm-db2Type
Active Directory CN(공통 이름)	ibm-db2Type
설명	데이터베이스 서버 유형입니다.
구문	Case Ignore 문자열
최대 길이	64
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.418
GUID(고유한 전역 ID)	b3afd320-5c5b-11d3-b818-002035559151
주	데이터베이스 서버의 유효한 유형: SERVER, MPP 및 DCS

표 46. *DCEPrincipalName*

속성	**DCEPrincipalName**
Active Directory LDAP 표시 이름	ibm-DCEPrincipalName
Active Directory CN(공통 이름)	ibm-DCEPrincipalName
설명	DCE 핵심부 이름
구문	Case Ignore 문자열
최대 길이	2048
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.443
GUID(고유한 전역 ID)	b3afd32d-5c5b-11d3-b818-002035559151

표 47. *cesProperty*

속성	**cesProperty**
Active Directory LDAP 표시 이름	ibm-cesProperty
Active Directory CN(공통 이름)	ibm-cesProperty
설명	응용프로그램 관련 환경 설정 구성 매개변수를 제공하는 데 이 속성의 값을 사용할 수 있습니다. 예를 들어, 값은 XML 형식화 데이터를 포함할 수 있습니다. 이 속성의 모든 값은 cesPropertyType 속성 값에서 동일해야 합니다.

표 47. *cesProperty (계속)*

속성	**cesProperty**
구문	Case Exact 문자열
최대 길이	32700
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.307
GUID(고유한 전역 ID)	b3afd2d5-5c5b-11d3-b818-002035559151

표 48. *cesPropertyType*

속성	**cesPropertyType**
Active Directory LDAP 표시 이름	ibm-cesPropertyType
Active Directory CN(공통 이름)	ibm-cesPropertyType
설명	이 속성의 값은 구문, 시맨틱 또는 cesProperty 속성의 모든 값의 다른 특성을 설명하기 위해 사용될 수 있습니다. 예를 들어『XML』 값을 사용하여 cesProperty 속성의 모든 값이 XML 구문으로 인코딩되었음을 나타낼 수 있습니다.
구문	Case Ignore 문자열
최대 길이	128
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.308
GUID(고유한 전역 ID)	b3afd2d6-5c5b-11d3-b818-002035559151

표 49. *cisProperty*

속성	**cisProperty**
Active Directory LDAP 표시 이름	ibm-cisProperty
Active Directory CN(공통 이름)	ibm-cisProperty
설명	응용프로그램 관련 환경 설정 구성 매개변수를 제공하는 데 이 속성의 값을 사용할 수 있습니다. 예를 들어, 값은 INI 파일을 포함할 수 있습니다. 이 속성의 모든 값은 cisPropertyType 속성 값에서 동일해야 합니다.
구문	Case Ignore 문자열
최대 길이	32700
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.309
GUID(고유한 전역 ID)	b3afd2e0-5c5b-11d3-b818-002035559151

표 50. *cisPropertyType*

속성	**cisPropertyType**
Active Directory LDAP 표시 이름	ibm-cisPropertyType
Active Directory CN(공통 이름)	ibm-cisPropertyType
설명	이 속성의 값을 사용하여 모든 cisProperty 속성 값의 구문, 시맨틱 또는 다른 특성을 설명할 수 있습니다. 예를 들어『INI 파일』 값을 사용하여 cisProperty 속성의 모든 값이 INI 파일임을 나타낼 수 있습니다.

표 50. *cisPropertyType (계속)*

속성	**cisPropertyType**
구문	Case Ignore 문자열
최대 길이	128
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.310
GUID(고유한 전역 ID)	b3afd2e1-5c5b-11d3-b818-002035559151

표 51. *binProperty*

속성	**binProperty**
Active Directory LDAP 표시 이름	ibm-binProperty
Active Directory CN(공통 이름)	ibm-binProperty
설명	응용프로그램 관련 환경 설정 구성 매개변수를 제공하는 데 이 속성의 값을 사용할 수 있습니다. 예를 들어, 값에는 2진 암호화 Lotus® 123 등록 정보 세트가 들어 있을 수 있습니다. 이 속성의 모든 값은 binPropertyType 속성 값에서 동일해야 합니다.
구문	2진
최대 길이	250000
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.305
GUID(고유한 전역 ID)	b3afd2ba-5c5b-11d3-b818-002035559151

표 52. *binPropertyType*

속성	**binPropertyType**
Active Directory LDAP 표시 이름	ibm-binPropertyType
Active Directory CN(공통 이름)	ibm-binPropertyType
설명	이 속성의 값은 구문, 시맨틱 또는 binProperty 속성의 모든 값의 다른 특성을 설명하기 위해 사용될 수 있습니다. 예를 들어 『Lotus 123』의 값을 사용하여 binProperty 속성의 모든 값이 2진 인코딩된 Lotus 123 등록 정보임을 나타낼 수 있습니다.
구문	Case Ignore 문자열
최대 길이	128
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.306
GUID(고유한 전역 ID)	b3afd2bb-5c5b-11d3-b818-002035559151

표 53. *PropertyType*

속성	**PropertyType**
Active Directory LDAP 표시 이름	ibm-propertyType
Active Directory CN(공통 이름)	ibm-propertyType
설명	이 속성의 값은 eProperty 오브젝트의 의미 특성에 대해 설명합니다.
구문	Case Ignore 문자열

표 53. PropertyType (계속)

속성	PropertyType
최대 길이	128
여러 값 지정	여러 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.320
GUID(고유한 전역 ID)	b3afd4ed-5c5b-11d3-b818-002035559151

표 54. settingID

속성	settingID
Active Directory LDAP 표시 이름	해당되지 않음
Active Directory CN(공통 이름)	해당되지 않음
설명	eProperty와 같은 오브젝트 항목에서 파생한 cimSetting을 식별하기 위해 사용될 수 있는 이름 지정 속성
구문	Case Ignore 문자열
최대 길이	256
여러 값 지정	하나의 값 지정
OID(오브젝트 ID)	1.3.18.0.2.4.325
GUID(고유한 전역 ID)	b3afd596-5c5b-11d3-b818-002035559151

DB2 오브젝트 클래스 및 속성으로 LDAP 디렉토리 스키마 확장

LDAP 디렉토리 스키마는 LDAP 디렉토리 엔트리에 저장된 정보의 오브젝트 클래스 및 속성을 정의합니다. 오브젝트 클래스는 필수 및 선택적 속성 세트로 구성됩니다. LDAP 디렉토리의 모든 항목에는 연관된 오브젝트 클래스가 있습니다.

DB2 데이터베이스 관리 프로그램이 LDAP에 정보를 저장하려면 먼저 LDAP 서버의 디렉토리 스키마에 DB2 데이터베이스 시스템이 사용하는 오브젝트 클래스 및 속성이 포함되어야 합니다. 기본 스키마에 새 오브젝트 클래스 및 속성을 추가하는 프로세스를 디렉토리 스키마 확장이라고 합니다.

주: IBM Tivoli Directory Server를 사용 중인 경우 DB2 UDB 버전 8.1 및 이전 버전에 필요한 모든 오브젝트 클래스 및 속성은 기본 스키마에 포함되어 있습니다. 이런 경우, DB2 오브젝트 클래스 및 속성을 사용하여 기본 스키마를 확장할 필요가 없습니다. 그러나, DB2 UDB 버전 8.2 이상에서 필요하지만 기본 스키마에 포함되지 않은 두 개의 새 속성이 있습니다. 이 경우, 두 개의 새 DB2 데이터베이스 속성으로 기본 스키마를 확장해야 합니다.

지원되는 LDAP 클라이언트 및 서버 구성

다음 표에는 지원되는 LDAP 클라이언트 및 서버 구성이 요약되어 있습니다.

IBM Tivoli Directory Server는 LDAP 버전 6.2 서버로, Windows, AIX, Solaris, Linux, 및 HP-UX에서 사용할 수 있으며, AIX 및 System i®에서 기본 운영 체제의 일부로 OS/390 Security Server와 함께 제공됩니다.

DB2 데이터베이스는 AIX, Solaris, HP-UX 11.11, Windows 및 Linux에서 IBM LDAP 클라이언트를 지원합니다.

Microsoft Active Directory 서버는 LDAP 버전 3 서버로, 운영 체제인 Windows 2000 Server 및 Windows Server 2003 제품군의 일부로 사용할 수 있습니다.

Microsoft LDAP 클라이언트는 Windows 운영 체제와 함께 포함됩니다.

표 55. 지원되는 LDAP 클라이언트 및 서버 구성

지원되는 **LDAP** 클라이언트 및 서버 구성	**IBM Tivoli Directory** 서버	**Microsoft Active Directory** 서버	**Sun One LDAP** 서버
IBM LDAP 클라이언트	지원됨	지원됨	지원됨
Microsoft LDAP/ADSI 클라이언트	지원됨	지원됨	지원됨

주: Windows 운영 체제에서 실행되는 경우 DB2 데이터베이스 관리 프로그램은 IBM LDAP 클라이언트 또는 Microsoft LDAP 클라이언트 사용을 지원합니다. IBM LDAP 클라이언트를 명시적으로 선택하려면 db2set 명령을 사용하여 DB2LDAP_CLIENT_PROVIDER 레지스트리 변수를『IBM』으로 설정합니다. Microsoft LDAP 클라이언트는 Windows 운영 체제와 함께 포함됩니다.

LDAP 지원 및 DB2 Connect

DB2 Connect 게이트웨이에서 LDAP이 지원되고 게이트웨이 데이터베이스 디렉토리에서 데이터베이스를 찾을 수 없는 경우 DB2 데이터베이스 관리 프로그램이 LDAP에서 데이터베이스 위치를 찾으며 발견한 정보를 보존하려 합니다.

LDAP에서 호스트 데이터베이스 등록

LDAP에서 호스트 데이터베이스를 등록하면 호스트 데이터베이스에 직접 연결 또는 게이트웨이를 통해 호스트 데이터베이스에 연결과 같은 두 가지 방법으로 구성할 수 있습니다.

호스트 데이터베이스에 직접 연결하려면 LDAP에서 호스트 서버를 등록한 다음 LDAP에서 호스트 데이터베이스를 카탈로그하여 호스트 서버의 호스트 이름을 지정합니다. 게이트웨이를 통해 호스트 데이터베이스에 연결하려면 LDAP에서 게이트웨이 서버를 등록한 다음 LDAP에서 호스트 데이터베이스를 카탈로그하여 게이트웨이 서버의 호스트 이름을 지정합니다.

DB2 Connect 게이트웨이에서 LDAP 지원이 가능하고 게이트웨이 데이터베이스 디렉토리에 데이터베이스가 없는 경우 DB2 데이터베이스 시스템에서는 LDAP를 검색하고 찾은 정보를 보존합니다.

다음 예에서는 두 가지 경우를 보여줍니다. 이 때 NIAGARA_FALLS라는 호스트 데이터베이스가 있다고 가정합니다. TCP/IP를 사용하여 수신 연결을 승인할 수 있습니다. DB2 Connect가 없어 클라이언트에서 호스트에 직접 연결할 수 없는 경우 goto@niagara라는 게이트웨이를 사용하여 연결합니다.

다음 단계를 수행해야 합니다.

1. TCP/IP 연결을 위해 LDAP에서 호스트 데이터베이스 서버 등록. 서버의 TCP/IP 호스트 이름은 ″myhost″이고 포트 번호는 ″446″입니다. 호스트 데이터베이스 서버임을 나타내기 위해 **NODETYPE**절이 DCS로 설정됩니다.

   ```
   db2 register ldap as nftcpip tcpip hostname myhost svcename 446
      remote mvssys instance mvsinst nodetype dcs
   ```

2. TCP/IP 연결을 위해 LDAP에서 DB2 Connect 게이트웨이 서버 등록. 게이트웨이 서버의 TCP/IP 호스트 이름은 ″niagara″이고 포트 번호는 ″50000″입니다.

   ```
   db2 register ldap as whasf tcpip hostname niagara svcename 50000
      remote niagara instance goto nodetype server
   ```

3. TCP/IP 연결을 사용하여 LDAP에서 호스트 데이터베이스 카탈로그. 호스트 데이터베이스 이름은 ″NIAGARA_FALLS″이고 데이터베이스 별명 이름은 ″nftcpip″입니다. **GWNODE**절은 DB2 Connect 게이트웨이 서버의 노드 이름을 지정하는 데 사용됩니다.

   ```
   db2 catalog ldap database NIAGARA_FALLS as nftcpip at node nftcpip
      gwnode whasf authentication server
   ```

위와 같이 등록 및 카탈로그를 완료한 후 TCPIP를 사용하여 호스트에 연결하려면 nftcpip에 연결합니다. 클라이언트 워크스테이션에 DB2 Connect가 없으면 TCPIP를 사용하여 게이트웨이를 통해 연결합니다. 게이트웨이에서 TCP/IP를 사용하여 호스트에 연결합니다.

일반적으로 각 클라이언트가 각 머신에서 로컬로 데이터베이스 및 노드를 수동으로 카탈로그할 필요가 없도록 LDAP에서 호스트 데이터베이스 정보를 수동으로 구성할 수 있습니다. 프로세스는 다음과 같습니다.

1. LDAP에서 호스트 데이터베이스 서버를 등록합니다. REGISTER 명령에서 **REMOTE**, **INSTANCE** 및 **NODETYPE**절을 각각 사용하여 호스트 데이터베이스 서버의 리모트 컴퓨터 이름, 인스턴스 이름 및 노드 유형을 지정해야 합니다. **REMOTE**절은 호스트 서버 머신의 호스트 이름 또는 LU 이름으로 설정될 수 있습니다. **INSTANCE**절은 8자 이하인 문자열로 설정될 수 있습니다. 예를 들어 인스턴스 이름은 ″DB2″로 설정될 수 있습니다. 호스트 데이터베이스 서버임을 나타내기 위해 **NODETYPE**절은 DCS로 설정되어야 합니다.

2. CATALOG LDAP DATABASE 명령을 사용하여 LDAP에서 호스트 데이터베이스를 등록하십시오. **PARMS**절을 사용하여 추가 DRDA 매개변수를 지정할 수 있습니다. 데이터베이스 인증 유형은 SERVER로 설정되어야 합니다.

IBM Tivoli Directory Server의 디렉토리 스키마 확장

IBM Tivoli Directory Server를 사용 중인 경우 버전 8.2 이전의 DB2 데이터베이스에 필요한 모든 오브젝트 클래스 및 속성은 기본 스키마에 포함되어 있습니다.

버전 8.2 이상에 추가된 새 DB2 데이터베이스 속성으로 기본 스키마를 확장하려면 다음 명령을 실행하십시오.

```
ldapmodify -c -h <machine_name>:389 -D <dn> -w <password> -f altgwnode.ldif
```

다음은 `altgwnode.ldif` 파일의 컨텐츠입니다.

```
dn:                cn=schema
changetype:        modify
add:               attributetypes
attributetypes:    (
 1.3.18.0.2.4.3092
 NAME 'db2altgwPtr'
        DESC 'DN pointer to DB2 alternate gateway (node) object'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12)
-
add:               ibmattributetypes
ibmattributetypes: (
 1.3.18.0.2.4.3092
 DBNAME ('db2altgwPtr' 'db2altgwPtr')
 ACCESS-CLASS NORMAL
 LENGTH 1000)


dn:            cn=schema
changetype:        modify
add:               attributetypes
attributetypes:    (
 1.3.18.0.2.4.3093
 NAME 'db2altnodePtr'
        DESC 'DN pointer to DB2 alternate node object'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12)
-
add:               ibmattributetypes
ibmattributetypes: (
 1.3.18.0.2.4.3093
 DBNAME ('db2altnodePtr' 'db2altnodePtr')
 ACCESS-CLASS NORMAL
 LENGTH 1000)


dn:            cn=schema
changetype:    modify
replace:       objectclasses
objectclasses: (
 1.3.18.0.2.6.117
 NAME 'DB2Database'
 DESC 'DB2 database'
 SUP cimSetting
 MUST ( db2databaseName $ db2nodePtr )
 MAY ( db2additionalParameters $ db2altgwPtr $ db2altnodePtr
    $ db2ARLibrary $ db2authenticationLocation $ db2databaseAlias
    $ db2databaseRelease $ db2gwPtr $ DCEPrincipalName ) )
```

`altgwnode.ldif` 및 `altgwnode.readme` 파일은 URL: ftp://ftp.software.ibm.com/ps/products/db2/tools/ldap에서 찾을 수 있습니다.

DB2 스키마 정의를 추가한 후 모든 변경사항을 활성화하려면 Directory Server를 재시작해야 합니다.

Netscape LDAP 디렉토리 지원 및 속성 정의

Netscape LDAP Server의 지원되는 레벨은 버전 4.12 이상입니다.

Netscape LDAP Server 버전 4.12 이상에서 Netscape Directory Server는 `slapd.user_oc.conf` 및 `slapd.user_at.conf` 파일에 속성 및 오브젝트 클래스 정

의를 추가하여 응용프로그램에서 스키마를 확장하도록 허용합니다. 이러한 두 파일은 <Netscape_install path>₩slapd-<machine_name>₩config 디렉토리에 있습니다.

주: Sun One Directory Server 5.0를 사용하는 경우 Sun One Directory Server의 디렉토리 스키마 확장에 대한 주제를 참조하십시오.

다음과 같이 DB2 속성을 slapd.user_at.conf에 추가해야 합니다.

```
########################################################################
#
# IBM DB2 Database
# Attribute Definitions
#
# bin -> binary
# ces -> case exact string
# cis -> case insensitive string
# dn -> distinguished name
#
########################################################################

attribute binProperty                    1.3.18.0.2.4.305     bin
attribute binPropertyType                1.3.18.0.2.4.306     cis
attribute cesProperty                    1.3.18.0.2.4.307     ces
attribute cesPropertyType                1.3.18.0.2.4.308     cis
attribute cisProperty                    1.3.18.0.2.4.309     cis
attribute cisPropertyType                1.3.18.0.2.4.310     cis
attribute propertyType                   1.3.18.0.2.4.320     cis
attribute systemName                     1.3.18.0.2.4.329     cis
attribute db2nodeName                    1.3.18.0.2.4.419     cis
attribute db2nodeAlias                   1.3.18.0.2.4.420     cis
attribute db2instanceName                1.3.18.0.2.4.428     cis
attribute db2Type                        1.3.18.0.2.4.418     cis
attribute db2databaseName                1.3.18.0.2.4.421     cis
attribute db2databaseAlias               1.3.18.0.2.4.422     cis
attribute db2nodePtr                     1.3.18.0.2.4.423     dn
attribute db2gwPtr                       1.3.18.0.2.4.424     dn
attribute db2additionalParameters        1.3.18.0.2.4.426     cis
attribute db2ARLibrary                   1.3.18.0.2.4.427     cis
attribute db2authenticationLocation      1.3.18.0.2.4.425     cis
attribute db2databaseRelease             1.3.18.0.2.4.429     cis
attribute DCEPrincipalName               1.3.18.0.2.4.443     cis
```

다음과 같이 DB2 오브젝트 클래스를 slapd.user_oc.conf 파일에 추가해야 합니다.

```
########################################################################
#
# IBM DB2 Database
# Object Class Definitions
#
########################################################################

objectclass eProperty
        oid 1.3.18.0.2.6.90
        requires
                objectClass
        allows
                cn,
                propertyType,
                binProperty,
```

```
                binPropertyType,
                cesProperty,
                cesPropertyType,
                cisProperty,
                cisPropertyType

objectclass eApplicationSystem
        oid 1.3.18.0.2.6.84
        requires
                objectClass,
                systemName


objectclass DB2Node
        oid 1.3.18.0.2.6.116
        requires
                objectClass,
                db2nodeName
        allows
                db2nodeAlias,
                host,
                db2instanceName,
                db2Type,
                description,
                protocolInformation

objectclass DB2Database
        oid 1.3.18.0.2.6.117
        requires
                objectClass,
                db2databaseName,
                db2nodePtr
        allows
                db2databaseAlias,
                description,
                db2gwPtr,
                db2additionalParameters,
                db2authenticationLocation,
                DCEPrincipalName,
                db2databaseRelease,
                db2ARLibrary
```

DB2 스키마 정의를 추가한 후 모든 변경사항을 활성화하려면 Directory Server를 재시작해야 합니다.

Sun One Directory Server의 디렉토리 스키마 확장

Sun One Directory Server는 Netscape 또는 iPlanet Directory Server라고도 알려져 있습니다.

사용자의 환경에서 Sun One Directory Server가 작동하도록 하려면 60ibmdb2.ldif 파일을 다음 디렉토리에 추가하십시오.

Windows의 경우 C:¥iPlanet¥Servers에 iPlanet이 설치되어 있으면 위의 파일을 .¥sldap-<machine_name>¥config¥schema에 추가하십시오.

UNIX의 경우 /usr/iplanet/servers에 iPlanet이 설치되어 있으면 위의 파일을 ./slapd-<machine_name>/config/schema에 추가하십시오.

파일 내용은 다음과 같습니다.

```
##############################################################################
# IBM DB2 Database
##############################################################################
dn: cn=schema
##############################################################################
# Attribute Definitions (Before V8.2)
##############################################################################
attributetypes: ( 1.3.18.0.2.4.305  NAME 'binProperty'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.5              X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.306  NAME 'binPropertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.307  NAME 'cesProperty'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.308  NAME 'cesPropertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.309  NAME 'cisProperty'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.310  NAME 'cisPropertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.320  NAME 'propertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.329  NAME 'systemName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.419  NAME 'db2nodeName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.420  NAME 'db2nodeAlias'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.428  NAME 'db2instanceName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.418  NAME 'db2Type'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.421  NAME 'db2databaseName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.422  NAME 'db2databaseAlias'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.426  NAME 'db2additionalParameters'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.427  NAME 'db2ARLibrary'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.425  NAME 'db2authenticationLocation'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.429  NAME 'db2databaseRelease'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.443  NAME 'DCEPrincipalName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.423  NAME 'db2nodePtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.424  NAME 'db2gwPtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
##############################################################################
# Attribute Definitions (V8.2 and later)
##############################################################################
attributetypes: ( 1.3.18.0.2.4.3092 NAME 'db2altgwPtr'
```

```
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12              X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.3093 NAME 'db2altnodePtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12              X-ORIGIN 'IBM DB2' )
##########################################################################
# Object Class Definitions
# DB2Database for V8.2 has the above two new optional attributes.
##########################################################################
objectClasses: ( 1.3.18.0.2.6.90  NAME 'eProperty'
   SUP top STRUCTURAL MAY ( cn $ propertyType $ binProperty
   $ binPropertyType $ cesProperty $ cesPropertyType $ cisProperty
   $ cisPropertyType ) X-ORIGIN 'IBM DB2' )
objectClasses: ( 1.3.18.0.2.6.84  NAME 'eApplicationSystem'
   SUP top STRUCTURAL MUST systemName
   X-ORIGIN 'IBM DB2' )
objectClasses: ( 1.3.18.0.2.6.116 NAME 'DB2Node'
   SUP top STRUCTURAL MUST db2nodeName MAY ( db2instanceName $ db2nodeAlias
   $ db2Type $ description $ host $ protocolInformation )
   X-ORIGIN 'IBM DB2' )
objectClasses: ( 1.3.18.0.2.6.117 NAME 'DB2Database'
   SUP top STRUCTURAL MUST (db2databaseName $ db2nodePtr ) MAY
   ( db2additionalParameters $ db2altgwPtr $ db2altnodePtr $ db2ARLibrary
   $ db2authenticationLocation $ db2databaseAlias $ db2databaseRelease
   $ db2gwPtr $ DCEPrincipalName $ description )
   X-ORIGIN 'IBM DB2' )
```

60ibmdb2.ldif 및 60ibmdb2.readme 파일은 URL: ftp://ftp.software.ibm.com/ps/products/db2/tools/ldap에서 찾을 수 있습니다.

DB2 스키마 정의를 추가한 후 모든 변경사항을 활성화하려면 Directory Server를 재시작해야 합니다.

Windows Active Directory

Active Directory 지원

DB2 데이터베이스 서버는 Active Directory에 ibm_db2Node 오브젝트로 발행됩니다. ibm_db2Node 오브젝트 클래스는 SCP(ServiceConnectionPoint) 오브젝트 클래스의 서브클래스입니다.

각 ibm_db2Node 오브젝트에는 클라이언트 응용프로그램을 DB2 데이터베이스 서버에 연결할 수 있도록 하는 프로토콜 구성 정보가 들어 있습니다. 새 데이터베이스가 작성되면 해당 데이터베이스는 Active Directory에서 ibm_db2Node 오브젝트 아래 ibm_db2Database 오브젝트로 발행됩니다.

리모트 데이터베이스에 연결할 경우 DB2 Client는 LDAP 인터페이스를 통해 Active Directory에서 ibm_db2Database 오브젝트를 쿼리합니다. 데이터베이스 서버에 연결하기 위한 프로토콜 통신(바인딩 정보)은 ibm_db2Database 오브젝트가 작성되어 있는 ibm_db2Node 오브젝트에서 가져옵니다.

도메인 제어기에서 *Active Directory 사용자 및 컴퓨터* 관리 콘솔(MMC)을 사용하여 ibm_db2Node 및 ibm_db2Database 오브젝트의 등록 정보 페이지를 보거나 수정할 수

있습니다. 등록 정보 페이지를 설정하려면 regsrv32 명령을 실행하여 다음과 같이 DB2 오브젝트의 등록 정보 페이지를 등록하십시오.

```
regsvr32 %DB2PATH%₩bin₩db2ads.dll
```

도메인 제어기에서 *Active Directory 사용자 및 컴퓨터* 관리 콘솔(MMC)을 사용하여 오브젝트를 볼 수 있습니다. 이 관리 도구로 이동하려면 시작 —>프로그램—> 관리 도구—> Active Directory 사용자 및 컴퓨터로 이동하십시오.

주: 컴퓨터 오브젝트 아래 DB2 데이터베이스 오브젝트를 표시하려면 뷰 메뉴에서 *컨테이너인 사용자, 그룹 및 컴퓨터*를 선택해야 합니다.

주: DB2 데이터베이스 시스템이 도메인 제어기에 설치되지 않은 경우에도 %DB2PATH%₩bin의 db2ads.dll 파일과 %DB2PATH%₩msg₩locale-name의 자원 DLL db2adsr.dll을 도메인 제어기의 로컬 디렉토리에 복사하여 DB2 데이터베이스 오브젝트의 등록 정보 페이지를 볼 수 있습니다. (이 두 개의 복사된 파일을 저장하는 디렉토리는 PATH 환경 변수에서 찾은 디렉토리 중 하나여야 합니다.) 그런 다음 로컬 디렉토리에서 regsvr32 명령을 실행하여 DLL을 등록하십시오.

Active Directory를 사용하도록 DB2 데이터베이스 관리 프로그램 구성

Microsoft Active Directory에 액세스하려면 다음 조건을 충족해야 합니다.

1. DB2 데이터베이스를 실행하는 컴퓨터는 Windows 2000 또는 Windows Server 2003 도메인에 속해야 합니다.
2. Microsoft LDAP 클라이언트가 설치되어 있어야 합니다. Microsoft LDAP 클라이언트는 Windows 2000, Windows XP 및 Windows Server 2003 운영 체제의 일부입니다.
3. LDAP 지원이 가능해야 합니다. Windows 2000, Windows XP 또는 Windows Server 2003의 경우 설치 프로그램을 통해 LDAP 지원을 가능하도록 할 수 있습니다.
4. Active Directory에서 정보를 읽어올 수 있도록 DB2 데이터베이스 시스템을 실행하는 경우 도메인 사용자 계정에 로그인해야 합니다.

Active Directory의 보안 고려사항

DB2 데이터베이스 및 노드 오브젝트는 Active Directory에 DB2 서버가 설치되어 있는 머신의 컴퓨터 오브젝트 아래 작성됩니다. Active Directory에서 데이터베이스 서버를 등록하거나 데이터베이스를 카탈로그하려면 컴퓨터 오브젝트 아래에 오브젝트를 작성하거나 갱신하기에 충분한 액세스 권한이 있어야 합니다.

디폴트로, 컴퓨터 오브젝트 아래 있는 오브젝트를 인증된 모든 사용자가 읽을 수 있으며 관리자(관리자, 도메인 관리자 및 엔터프라이즈 관리자 그룹에 속하는 사용자)가 갱신할 수 있습니다. 특정 사용자 또는 그룹에 액세스를 부여하려면 다음과 같이 *Active Directory 사용자 및 컴퓨터* 관리 콘솔(MMC)을 사용하십시오.

1. *Active Directory 사용자 및 컴퓨터* 관리 도구 시작

   (시작—> 프로그램—> 관리 도구—> Active Directory 사용자 및 컴퓨터)
2. *뷰*에서 *고급 기능* 선택
3. *컴퓨터* 컨테이너 선택
4. DB2가 설치된 서버 머신을 나타내는 컴퓨터 오브젝트를 마우스 오른쪽 단추로 누른 다음 등록 *정보* 선택
5. *보안* 탭을 선택한 다음 지정된 사용자 또는 그룹에 필요한 액세스 권한 추가

사용자 레벨의 DB2 레지스트리 변수 및 CLI 설정은 사용자 오브젝트 아래 있는 DB2 등록 정보 오브젝트에서 유지보수됩니다. DB2 레지스트리 변수 또는 CLI 설정을 사용자 레벨에서 설정하려면 사용자 오브젝트 아래 오브젝트를 작성하기에 충분한 액세스 권한이 있어야 합니다.

디폴트로, 관리자에게만 사용자 오브젝트 아래 오브젝트를 작성할 액세스 권한이 있습니다. 사용자에게 액세스를 부여하여 DB2 레지스트리 변수 또는 CLI 설정을 사용자 레벨에서 설정하도록 하려면 다음과 같이 *Active Directory 사용자 및 컴퓨터* 관리 콘솔(MMC)을 사용하십시오.

1. *Active Directory 사용자 및 컴퓨터* 관리 도구 시작

   (시작—> 프로그램—> 관리 도구—> Active Directory 사용자 및 컴퓨터)
2. 사용자 컨테이너 아래에 있는 사용자 오브젝트 선택
3. 사용자 오브젝트를 마우스 오른쪽 단추로 누르고 등록 *정보* 선택
4. *보안* 탭 선택
5. 추가 단추를 사용하여 목록에 사용자 이름 추가
6. 『*쓰기*』 권한 부여 및 『모든 하위 오브젝트 작성』 액세스
7. 고급 설정을 사용하여 『이 오브젝트 및 모든 하위 오브젝트』에 적용할 권한 설정
8. 『상위에서 이 오브젝트로 전달할 권한을 상속하도록 허용』 선택란 선택

Active Directory의 DB2 오브젝트

DB2 데이터 관리 프로그램은 다음과 같은 Active Directory의 두 위치에서 오브젝트를 작성합니다.

1. DB2 데이터베이스 및 노드 오브젝트는 DB2 서버가 설치된 머신의 컴퓨터 오브젝트 아래에 작성됩니다. Windows 도메인에 속하지 않는 DB2 서버의 경우, DB2 데이터베이스 및 노드 오브젝트는 『System』 컨테이너 아래에 작성됩니다.
2. 사용자 레벨의 DB2 레지스트리 변수 및 CLI 설정은 사용자 오브젝트 아래의 DB2 등록 정보 오브젝트에 저장됩니다. 이러한 오브젝트에는 해당 사용자와 관련된 정보가 들어 있습니다.

Active Directory의 디렉토리 스키마 확장

DB2 지원을 사용하지 않도록 설정하려면 다음 프로시저를 사용합니다. 데이터베이스 관리 프로그램에서 Active Directory에 정보를 저장하려면 새 DB2 데이터베이스 오브젝트 클래스 및 속성을 포함하도록 디렉토리 스키마를 확장해야 합니다. 새 오브젝트 클래스와 속성을 디렉토리 스키마에 추가하는 프로세스를 *스키마 확장*이라고 합니다.

Windows 도메인의 일부인 컴퓨터에 DB2 데이터베이스 시스템을 처음으로 설치하려면 DB2 스키마 설치 프로그램인 db2schex를 실행하여 Active Directory의 스키마를 확장해야 합니다.

db2schex 프로그램은 제품 CD-ROM의 x:₩db2₩windows₩utilities₩ 위치에 있습니다. 여기서 x:는 CD-ROM 드라이브입니다.

명령은 다음과 같이 사용됩니다.

```
db2schex
```

다음 명령과 연관된 다른 선택적 절이 있습니다.

-b UserDN
사용자 식별 이름을 지정합니다.

-w Password
바인드 암호를 지정합니다.

-u 스키마를 설치 제거합니다.

-k 오류를 무시하고 설치 제거를 계속 하도록 강제 실행합니다.

주:

1. UserDN과 암호가 지정되어 있으면 db2schex는 현재 로그인한 사용자로 바인드됩니다.
2. userDN절은 Windows 사용자 이름으로 지정될 수 있습니다.
3. 스키마를 갱신하려면 스키마 관리자 그룹의 구성원이거나 스키마 갱신 권한을 위임받아야 합니다.

디렉토리 스키마를 확장하려면 DB2 UDB 버전 8.2 이상과 함께 제공되는 db2schex 명령을 실행해야 합니다.

이전 버전의 DB2 데이터베이스 관리 시스템에서 db2schex 명령을 실행한 경우 DB2 UDB 버전 8.2 이상에서 동일한 명령을 다시 실행하면 다음 두 개의 선택적 속성이 ibm-db2Database 클래스에 추가됩니다.

```
ibm-db2AltGwPtr
ibm-db2NodePtr
```

Windows에서 이전 버전의 DB2 데이터베이스 관리 시스템에서 db2schex 명령을 실행할 수 없는 경우 DB2 UDB 버전 8.2 이상에서 동일한 명령을 실행하면 DB2 데이터베이스 시스템 LDAP 지원에 대한 모든 클래스 및 속성이 추가됩니다.

예를 들면, 다음과 같습니다.

- DB2 데이터베이스 스키마를 설치하려면 다음 명령을 실행하십시오.

  ```
  db2schex
  ```

- DB2 데이터베이스 스키마를 설치하고 바인드 DN 및 암호를 지정하려면 다음 명령을 실행하십시오.

  ```
  db2schex -b "cn=A Name,dc=toronto1,dc=ibm,dc=com"
           -w password
  ```

  또는

  ```
  db2schex -b Administrator -w password
  ```

- DB2 데이터베이스 스키마를 설치 제거하려면 다음 명령을 실행하십시오.

  ```
  db2schex -u
  ```

- DB2 데이터베이스 스키마를 설치 제거하고 오류를 무시하려면 다음 명령을 실행하십시오.

  ```
  db2schex -u -k
  ```

Active Directory용 DB2 스키마 설치 프로그램은 다음 태스크를 수행합니다.

1. 스키마 마스터인 서버 발견
2. 스키마 마스터인 도메인 제어기에 바인드
3. 사용자에게 스키마에 클래스와 속성을 추가할 수 있는 권한이 있는지 확인
4. 스키마 마스터가 쓰기 가능한지(즉, 레지스트리의 안전 인터록이 제거되는지) 확인
5. 모든 새 속성 작성
6. 모든 새 오브젝트 클래스 작성
7. 오류 발견 및 오류 발생 시 프로그램이 스키마의 변경사항 롤백

설치 완료 후 LDAP 지원을 사용하도록 설정

설치 후 LDAP 지원을 사용하도록 설정해야 합니다.

각 컴퓨터에서 다음 프로시저를 사용하십시오.

1. LDAP 지원 2진 파일을 설치하려면 설치 프로그램을 실행하여 사용자 설치에서 LDAP Directory Exploitation 지원을 선택하십시오. 설치 프로그램은 2진 파일을 설치하고 DB2 프로파일 레지스트리 변수 DB2_ENABLE_LDAP를 ″YES″로 설정합니다.

주: Windows 및 UNIX 플랫폼의 경우 db2set 명령을 사용하여 DB2_ENABLE_LDAP 레지스트리 변수를 ″YES″로 설정하여 LDAP를 사용 가능하도록 명시적으로 설정해야 합니다.

2. UNIX 플랫폼에만 해당: 다음 명령을 사용하여 LDAP 서버의 TCP/IP 호스트 이름 및 (선택적) 포트 번호를 선언합니다.
   `db2set DB2LDAPHOST=<base_domain_name>[:port_number]` 여기서 base_domain_name은 LDAP 서버의 TCP/IP 호스트 이름이고 [:port_number]는 포트 번호입니다. 포트 번호를 지정하지 않으면 디폴트 LDAP 포트(389)가 사용됩니다.

   DB2 오브젝트는 LDAP 기본 식별 이름(baseDN)에 있습니다. 다음과 같이 DB2SET 명령을 사용하여 각 컴퓨터에서 LDAP 기본 식별 이름을 구성할 수 있습니다.

   ```
   db2set DB2LDAP_BASEDN=<baseDN>
   ```

   여기서 baseDN은 LDAP 서버에서 정의된 LDAP 접미부의 이름입니다. LDAP 접미부는 DB2 오브젝트를 포함하는 데 사용됩니다.

3. REGISTER LDAP AS 명령을 사용하여 LDAP에서 DB2 서버의 현재 인스턴스를 등록하십시오. 예를 들어,

   ```
   db2 register ldap as <node-name> protocol tcpip
   ```

4. LDAP에 등록하려는 데이터베이스가 있으면 CATALOG LDAP DATABASE 명령을 실행하십시오. 예를 들어, 다음과 같습니다.

   ```
   db2 catalog ldap database <dbname> as <alias_dbname>
   ```

5. LDAP 사용자의 식별 이름(DN) 및 암호를 입력하십시오. 이러한 값은 LDAP를 사용하여 DB2 사용자 관련 정보를 저장하는 경우에만 필요합니다.

IBM LDAP 환경에서 DB2 구성

IBM LDAP 환경에서 DB2를 사용하려면 각 컴퓨터에서 다음 설정을 구성해야 합니다.

- LDAP 지원이 가능해야 합니다. Windows의 경우 설치 프로그램을 통해 LDAP 지원을 가능하도록 할 수 있습니다. 모든 Windows 운영 체제에서 사용하는 디폴트 LDAP 클라이언트는 Microsoft의 제품입니다. IBM LDAP 클라이언트를 사용하려면 db2set 명령을 사용하여 DB2LDAP_CLIENT_PROVIDER 레지스트리 변수를 ″IBM″으로 설정해야 합니다.
- LDAP 서버의 TCP/IP 호스트 이름 및 포트 번호입니다. 이러한 값은 DB2LDAPHOST 응답 키워드를 사용하여 자동 설치 중 입력하거나 db2set 명령을 사용하여 이후에 수동으로 설정할 수도 있습니다.

  ```
  db2set DB2LDAPHOST=<hostname[:port]>
  ```

여기서 hostname은 LDAP 서버의 TCP/IP 호스트 이름이고 [:port]는 포트 번호입니다. 포트 번호를 지정하지 않으면 DB2에서는 디폴트 LDAP 포트(389)를 사용합니다.

DB2 오브젝트는 LDAP 기본 식별 이름(baseDN)에 있습니다. 다음과 같이 db2set 명령을 사용하여 각 컴퓨터에서 LDAP 기본 식별 이름을 구성할 수 있습니다.

```
db2set DB2LDAP_BASEDN=<baseDN>
```

여기서 baseDN은 LDAP 서버에서 정의된 LDAP 접미부의 이름입니다. LDAP 접미부는 DB2 오브젝트를 포함하는 데 사용됩니다.

- LDAP 사용자의 식별 이름(DN) 및 암호입니다. 이러한 값은 LDAP를 사용하여 DB2 사용자 관련 정보를 저장하는 경우에만 필요합니다.

LDAP 항목 등록

설치 후 DB2 서버 등록

DB2 서버 인스턴스에 연결하기 위해 클라이언트 응용프로그램에서 사용하는 프로토콜 구성 정보를 게시하려면 각 DB2 서버 인스턴스를 LDAP에 등록해야 합니다.

데이터베이스 서버 인스턴스를 등록할 때, 노드 *이름*을 지정해야 합니다. 노드 이름은 서버에 연결 또는 접속하는 경우 클라이언트 응용프로그램에서 사용됩니다. CATALOG LDAP NODE 명령을 사용하여 LDAP 노드의 다른 별명 이름을 카탈로그할 수 있습니다.

주: Windows 도메인 환경에서 작업하는 경우 설치 중 DB2 서버 인스턴스가 다음 정보와 함께 Active Directory에 자동으로 등록됩니다.

```
nodename: TCP/IP hostname
protocol type: TCP/IP
```

TCP/IP 호스트 이름이 8자보다 길면 8자로 절단됩니다.

REGISTER 명령은 다음과 같이 나타납니다.

```
db2 register db2 server in ldap
   as <ldap_node_name>
   protocol tcpip
```

프로토콜절은 이 데이터베이스 서버에 연결하는 경우 사용되는 통신 프로토콜을 지정합니다.

여러 실제 머신이 포함되어 있는 DB2 Enterprise Server Edition의 인스턴스를 작성하는 경우 각 컴퓨터에 대해 한 번씩 REGISTER 명령을 호출해야 합니다. 모든 컴퓨터에 대해 REGISTER 명령을 발행하려면 rah 명령을 사용하십시오.

주: 각 컴퓨터는 LDAP에서 고유 이름을 가져야 하므로 동일한 ldap_node_name을 사용할 수 없습니다. REGISTER 명령에서 각 컴퓨터의 호스트 이름을 ldap_node_name으로 이름으로 대체하려고 할 수 있습니다. 예를 들면, 다음과 같습니다.

```
rah ">DB2 REGISTER DB2 SERVER IN LDAP AS <> PROTOCOL TCPIP"
```

″<>″은 rah 명령이 실행되는 각 컴퓨터에서 호스트 이름으로 대체됩니다. 드물게 여러 DB2 Enterprise Server Edition 인스턴스가 있는 경우 인스턴스와 호스트 인덱스의 조합이 rah 명령에서 노드 이름으로 사용될 수 있습니다.

리모트 DB2 서버에 대해 REGISTER 명령을 발행할 수 있습니다. 이렇게 하려면 리모트 서버 등록 시 리모트 컴퓨터 이름, 인스턴스 이름 및 프로토콜 구성 매개변수를 지정해야 합니다. 다음과 같이 명령을 사용할 수 있습니다.

```
db2 register db2 server in ldap
   as <ldap_node_name>
   protocol tcpip
   hostname <host_name>
   svcename <tcpip_service_name>
   remote <remote_computer_name>
   instance <instance_name>
```

다음 규칙은 컴퓨터 이름에 사용됩니다.

- TCP/IP가 구성되면 컴퓨터 이름은 TCP/IP 호스트 이름과 동일해야 합니다.

고가용성 또는 장애 복구 환경에서 실행하고 TCP/IP를 통신 프로토콜로 사용하는 경우 클러스터 IP 주소를 사용해야 합니다. 클러스터 IP 주소를 사용하면 클라이언트가 각 컴퓨터에 대해 별도의 TCP/IP 노드를 카탈로그화하지 않고도 컴퓨터에 있는 서버에 연결할 수 있도록 합니다. 클러스터 IP 주소는 아래와 같이 호스트 이름절을 사용하여 지정됩니다.

```
db2 register db2 server in ldap
   as <ldap_node_name>
   protocol tcpip
   hostname n.nn.nn.nn
```

여기서 n.nn.nn.nn은 클러스터 IP 주소입니다.

클라이언트 응용프로그램에서 LDAP에 DB2 서버를 등록하려면 db2LdapRegister API를 호출하십시오.

ATTACH의 노드 별명 카탈로그

LDAP에서 서버를 등록하는 경우 DB2 서버의 노드 이름을 지정해야 합니다. 응용프로그램에서는 데이터베이스 서버에 접속하는 데 노드 이름을 사용합니다.

다른 노드 이름이 필요한 경우(예: 응용프로그램에서 노드 이름이 하드 코딩된 경우) 예를 들어 다음과 같이 CATALOG LDAP NODE 명령을 사용하여 변경하십시오.

```
db2 catalog ldap node <ldap_node_name>
   as <new_alias_name>
```

LDAP 노드를 카탈로그 해제하려면 예를 들어 다음과 같이 UNCATALOG LDAP NODE 명령을 사용하십시오.

```
db2 uncatalog ldap node <ldap_node_name>
```

LDAP 디렉토리에서 데이터베이스 등록

인스턴스 내에서 데이터베이스를 작성하는 동안 데이터베이스는 LDAP에 자동으로 등록됩니다. 이 등록은 클라인트 컴퓨터에 데이터베이스 및 노드를 카탈로그화하지 않고도 리모트 클라이언트가 데이터베이스에 연결할 수 있게 합니다. 클라이언트가 데이트베이스에 연결하려 할 때, 데이터베이스가 로컬 컴퓨터의 데이터베이스 디렉토리에 존재하지 않으면, LDAP 디렉토리가 검색됩니다.

이 이름이 LDAP 디렉토리에 이미 있을 경우, 데이터베이스가 로컬 컴퓨터에서 여전히 작성되지만 LDAP 디렉토리에 이름 지정 충돌이 있음을 나타내는 경고 메시지가 리턴됩니다. 이러한 이유로 인해 LDAP 디렉토리에서 데이터베이스를 수동으로 카탈로그할 수 있습니다. CATALOG LDAP DATABASE 명령을 사용하여 LDAP에서 리모트 서버에 데이터베이스를 등록할 수 있습니다. 리모트 데이터베이스를 등록할 때, 리모트 데이터베이스 서버를 나타내는 LDAP 노드의 이름을 지정합니다. 데이터베이스를 등록하기 전에 REGISTER DB2 SERVER IN LDAP 명령을 사용하여 LDAP에서 리모트 데이터베이스 서버를 등록할 수 있습니다. LDAP에서 데이터베이스를 수동으로 등록하려면 CATALOG LDAP DATABASE 명령을 사용하십시오.

```
db2 catalog ldap database <dbname>
   at node <node_name>
   with "My LDAP database"
```

클라이언트 응용프로그램에서 LDAP에 데이터베이스를 등록하려면 db2LdapCatalogDatabase API를 호출하십시오.

LDAP 항목 등록 해제

DB2 서버 등록 해제

LDAP에서 인스턴스의 등록을 해제하면 해당 인스턴스를 참조하는 모든 노드 또는 별명, 오브젝트 및 데이터베이스 오브젝트가 제거됩니다.

로컬 또는 리모트 컴퓨터에 있는 DB2 서버를 등록 해제하려면 LDAP 노드 이름이 서버에 지정되어야 합니다.

```
db2 deregister db2 server in ldap
   node <node_name>
```

클라이언트 응용프로그램에서 LDAP으로부터 DB2 서버의 등록을 해제하려면 db2LdapDeregister API를 호출하십시오.

DB2 서버의 등록이 해제되면 DB2 서버의 동일한 인스턴스를 참조하는 모든 LDAP 노드 항목 및 LDAP 데이터베이스 항목도 카탈로그 해제됩니다.

LDAP 디렉토리에서 데이터베이스 등록 해제

데이터베이스는 데이터베이스가 삭제되는 경우 또는 소유 인스턴스가 LDAP에서 등록 해제될 때 자동으로 LDAP에서 등록 해제됩니다.

다음 명령을 사용하여 LDAP에서 데이터베이스를 수동으로 등록 해제할 수 있습니다.

```
db2 uncatalog ldap database <dbname>
```

클라이언트 응용프로그램에서 LDAP로부터 데이터베이스를 등록 해제하려면 db2LdapUncatalogDatabase API를 호출하십시오.

LDAP 사용자 구성

LDAP 사용자 작성

IBM Tivoli 디렉토리를 사용하는 경우 LDAP에 사용자 레벨 정보를 저장하려면 LDAP 사용자를 정의해야 합니다. 사용자 오브젝트의 모든 속성을 포함하도록 LDIF 파일을 작성하여 LDAP 사용자를 작성한 다음 LDIF 임포트 유틸리티를 실행하여 오브젝트를 LDAP 디렉토리로 임포트할 수 있습니다.

DB2 데이터베이스 시스템은 사용자 레벨에서 DB2 레지스트리 변수 및 CLI 구성 설정을 지원합니다. 이러한 기능은 Linux 및 UNIX 플랫폼에서는 사용할 수 없습니다. 사용자 레벨 지원은 다중 사용자 환경에서 사용자별 설정을 제공합니다. 이러한 예로 로그온한 각 사용자가 시스템 환경 또는 다른 사용자의 환경을 방해하지 않고 자신의 환경을 사용자 정의할 수 있는 Windows Terminal Server가 있습니다.

IBM Tivoli Directory Server의 LDIF 유틸리티는 LDIF2DB입니다.

개별 오브젝트의 속성이 포함된 LDIF 파일은 다음과 유사하게 나타납니다.

```
File name: newuser.ldif

dn: cn=Mary Burnnet, ou=DB2 Development, ou=Toronto, o=ibm, c=ca
objectclass: ePerson
cn: Mary Burnnet
sn: Burnnet
uid: mburnnet
userPassword: password
telephonenumber: 1-416-123-4567
facsimiletelephonenumber: 1-416-123-4568
title: Software Developer
```

다음은 IBM LDIF 임포트 유틸리티를 사용하여 LDIF 파일을 임포트하는 LDIF 명령의 예입니다.

```
LDIF2DB -i newuser.ldif
```

주:

1. LDIF2DB명령은 LDAP 서버에서 실행해야 합니다.
2. LDAP 사용자가 자신의 고유한 오브젝트를 추가하고, 삭제하고, 읽고 해당 오브젝트에 쓸 수 있도록 LDAP 사용자에게 필수 액세스 권한(ACL)을 부여해야 합니다. 사용자 오브젝트에 ACL을 부여하려면 LDAP Directory Server 웹 관리 도구를 사용하십시오.

DB2 응용프로그램의 LDAP 사용자 구성

Microsoft LDAP 클라이언트를 사용하는 경우 LDAP 사용자는 운영 체제의 사용자 계정과 동일합니다. 그러나 IBM LDAP 클라이언트를 사용하는 경우 DB2 데이터베이스 관리 프로그램을 사용하려면 현재 로그온한 사용자에 대한 LDAP 사용자 식별 이름(DN)과 암호를 구성해야 합니다.

LDAP 사용자 식별 이름(DN) 및 암호를 구성하려면 db2ldcfg 유틸리티를 사용하십시오.

```
db2ldcfg -u <userDN> -w <password> --> set the user's DN and password
         -r                        --> clear the user's DN and password
```

예:

```
db2ldcfg -u "cn=Mary Burnnet,ou=DB2 Development,ou=Toronto,o=ibm,c=ca"
         -w password
```

환경의 사용자 레벨에서 DB2 레지스트리 변수 설정

LDAP 환경에서는 사용자가 자신의 DB2 환경을 사용자 정의할 수 있는 사용자 레벨에서 DB2 프로파일 레지스트리 변수를 설정할 수 있습니다.

사용자 레벨에서 DB2 프로파일 레지스트리 변수를 설정하려면 -ul 옵션을 사용하십시오.

```
db2set -ul <variable>=<value>
```

주: 이는 AIX 또는 Solaris 운영 체제에서 지원되지 않습니다.

DB2에는 캐싱 메커니즘이 있습니다. 사용자 레벨의 DB2 프로파일 레지스트리 변수는 로컬 컴퓨터에서 캐시에 저장됩니다. -ul 매개변수를 지정하면 DB2는 항상 DB2 레지스트리 변수의 캐시에서 읽습니다. 캐시는 다음 상황에서 새로 고쳐집니다.

- 사용자 레벨에서 DB2 레지스트리 변수를 갱신 또는 재설정합니다.
- 사용자 레벨에서 LDAP 프로파일 변수를 새로 고치는 명령은 다음과 같습니다.

```
db2set -ur
```

LDAP 지원을 사용하지 않도록 설정

지원을 사용하지 않도록 설정하려면 다음 프로시저를 사용하십시오.

1. DB2 서버의 각 인스턴스의 경우 다음과 같이 LDAP에서 DB2 서버의 등록을 해제하십시오.

   ```
   db2 deregister db2 server in ldap node <nodename>
   ```

2. DB2 프로파일 레지스트리 변수 DB2_ENABLE_LDAP를 ″NO″로 설정하십시오.

DB2 서버에 대한 프로토콜 정보 갱신

LDAP의 DB2 서버 정보는 항상 최신 상태여야 합니다. 예를 들어 프로토콜 구성 매개변수 또는 서버 네트워크 주소를 변경하면 LDAP를 갱신해야 합니다.

로컬 컴퓨터의 LDAP에 있는 DB2 서버를 갱신하려면, 다음 명령을 사용하십시오.

```
db2 update ldap ...
```

갱신할 수 있는 프로토콜 구성 매개변수의 예에는 TCP/IP 호스트 이름, 서비스 이름 또는 포트 번호 매개변수가 있습니다.

리모트 DB2 서버 프로토콜 구성 매개변수를 갱신하려면 다음과 같이 node절과 함께 UPDATE LDAP 명령을 사용하십시오.

```
db2 update ldap
   node <node_name>
   hostname <host_name>
   svcename <tcpip_service_name>
```

다른 서버로 LDAP 클라이언트 리라우트

시스템 오류 시 클라이언트를 리라우트하는 기능처럼 LDAP를 사용하여 작업하는 경우에도 동일한 기능을 사용할 수 있습니다.

DB2_ENABLE_LDAP 레지스트리 변수를 『Yes』로 설정해야 합니다.

환경에서 모든 데이터베이스 및 노드 디렉토리 정보는 LDAP 서버에서 유지보수됩니다. 클라이언트는 LDAP 디렉토리에서 정보를 검색합니다. DB2LDAPCACHE 레지스트리 변수가 『Yes』로 설정되어 있으면 이러한 정보는 로컬 데이터베이스 및 노드 디렉토리에서 갱신됩니다.

UPDATE ALTERNATE SERVER FOR LDAP DATABASE 명령을 사용하여 LDAP의 DB2 데이터베이스를 나타내는 데이터베이스의 대체 서버를 정의합니다. 또한

클라이언트 응용프로그램에서 db2LdapUpdateAlternateServerForDB API를 호출하여 LDAP에서 데이터베이스의 대체 서버를 갱신할 수 있습니다.

설정되면 이러한 대체 서버 정보는 연결 시 클라이언트로 리턴됩니다.

주: LDAP 서버에 저장된 대체 서버 정보를 데이터베이스 서버 인스턴스에 저장된 대체 서버 정보와 계속해서 동기화할 것을 적극 권장합니다. 데이터베이스 서버 인스턴스에서 UPDATE ALTERNATE SERVER FOR DATABASE 명령(″FOR LDAP DATABASE″가 아님)을 발행하면 데이터베이스 서버 인스턴스와 LDAP 서버 간에 동기화됨을 확인할 수 있습니다.

서버 인스턴스에 UPDATE ALTERNATE SERVER FOR DATABASE 명령을 입력하면 LDAP 지원이 사용 가능한 경우(DB2_ENABLE_LDAP=Yes) 및 LDAP 사용자 ID 및 암호가 캐시된 경우(db2ldcfg가 이전에 실행됨) 데이터베이스의 대체 서버가 LDAP 서버에서 자동으로 또는 내재적으로 갱신됩니다. 이는 마치 UPDATE ALTERNATE SERVER FOR LDAP DATABASE를 명시적으로 입력한 것처럼 작동합니다.

데이터베이스 서버 인스턴스 이외의 인스턴스에서 UPDATE ALTERNATE SERVER FOR LDAP DATABASE 명령이 발행되면 대체 서버 정보도 UPDATE ALTERNATE SERVER FOR DATABASE 명령을 사용하여 데이터베이스 서버 인스턴스에서 동일하게 구성됩니다. 처음에 클라이언트에서 데이터베이스 서버 인스턴스에 연결하면 해당 데이터베이스 서버 인스턴스에서 리턴된 대체 서버 정보가 LDAP 서버에서 구성된 내용보다 우선합니다. 데이터베이스 서버 인스턴스에 구성된 대체 서버 정보가 없는 경우 처음 연결 후 클라이언트 리라우트가 사용되지 않는 것으로 간주됩니다.

LDAP 환경에서 리모트 서버에 접속

LDAP 환경에서 다음 ATTACH 명령에 LDAP 노드 이름을 사용하여 리모트 데이터베이스 서버에 접속할 수 있습니다. `db2 attach to <ldap_node_name>`

클라이언트 응용프로그램에서 처음으로 노드에 접속하거나 데이터베이스에 연결하는 경우 로컬 노드 디렉토리에 노드가 없으므로 데이터베이스 관리 프로그램에서는 목표 노드 항목을 찾기 위해 LDAP 디렉토리를 검색합니다. 해당 항목이 LDAP 디렉토리에 없으면 리모트 서버의 프로토콜 정보가 검색됩니다. 데이터베이스에 연결하는 경우 및 LDAP 디렉토리에 항목이 있는 경우 데이터베이스 정보도 검색됩니다. 이 정보를 사용하여 데이터베이스 관리가 자동으로 로컬 컴퓨터에 데이터베이스 항목 및 노드 항목을 카탈로그화합니다. 다음에 클라이언트 응용프로그램에서 동일한 노드 또는 데이터베이스에 접속할 때 LDAP 디렉토리가 검색되지 않고 로컬 데이터베이스 디렉토리의 정보가 사용됩니다.

자세한 내용: 클라이언트에서 LDAP 서버를 한 번만 검색하도록 캐싱 메커니즘이 존재합니다. 정보가 검색되면 *dir_cache* 데이터베이스 관리 프로그램 구성 매개변수 및 DB2LDAPCACHE 레지스트리 변수의 값에 따라 로컬 컴퓨터에 해당 정보가 저장 또는 캐시됩니다.

- DB2LDAPCACHE=NO 및 dir_cache=NO이면 항상 LDAP에서 정보를 읽어옵니다.
- DB2LDAPCACHE=NO 및 dir_cache=YES이면 LDAP에서 정보를 한 번만 읽어오고 해당 정보를 DB2(R) 캐시에 삽입합니다.
- DB2LDAPCACHE=YES이거나 설정되어 있지 않으면 LDAP 서버에서 정보를 한 번만 읽어오고 해당 정보를 로컬 데이터베이스, 노드 및 DCS 디렉토리에 캐시합니다.

주: LDAP 정보 캐싱은 사용자 레벨 CLI 또는 DB2 프로파일 레지스트리 변수에 적용되지 않습니다.

로컬 데이터베이스 및 노드 디렉토리에서 LDAP 항목 새로 고침

DB2 데이터베이스 시스템에서는 클라이언트에서 LDAP 서버를 검색하는 횟수를 줄이는 캐싱 메커니즘을 제공합니다.

정보가 검색되면 dir_cache 데이터베이스 관리 프로그램 구성 매개변수 및 DB2LDAPCACHE 레지스트리 변수의 값에 따라 로컬 컴퓨터에 해당 정보가 저장 또는 캐시됩니다.

- DB2LDAPCACHE=NO 및 dir_cache=NO이면 항상 LDAP에서 정보를 읽어옵니다.
- DB2LDAPCACHE=NO 및 dir_cache=YES이면 LDAP에서 정보를 한 번만 읽어오고 해당 정보를 DB2 캐시에 삽입합니다.
- DB2LDAPCACHE=YES이거나 설정되어 있지 않으면 LDAP 서버에서 정보를 한 번만 읽어오고 해당 정보를 로컬 데이터베이스, 노드 및 DCS 디렉토리에 캐시합니다.

주: LDAP 정보 캐싱은 사용자 레벨 CLI 또는 DB2 프로파일 레지스트리 변수에 적용되지 않습니다.LDAP의 정보는 변경될 수 있으므로 로컬 데이터베이스 및 노드 디렉토리에 캐시된 LDAP 항목을 새로 고쳐야 합니다. 다음과 같은 몇 가지 방법으로 새로 고칠 수 있습니다.

LDAP에서 검색된 로컬 데이터베이스 및 노드 항목을 모두 새로 고치려면 다음 명령을 사용하십시오.

```
db2 refresh ldap immediate
```

이와 유사하게 다음 명령을 사용하여 기존 로컬 데이터베이스 및 노드 항목을 모두 새로 고치고 LDAP의 새 항목을 추가할 수 있습니다.

```
db2 refresh ldap immediate all
```

IMMEDIATE ALL 옵션을 지정하면 LDAP 서버와 함께 포함된 모든 NODE 및 DB 항목이 로컬 디렉토리에 추가됩니다.

또는 DB2에서 다음 데이터베이스 연결 또는 인스턴스 접속 시 LDAP 자원을 참조하는 데이터베이스 항목을 강제로 새로 고치려면 다음 명령을 사용하십시오.

```
db2 refresh ldap database directory
```

이와 유사하게 DB2 데이터베이스 관리 프로그램에서 다음 데이터베이스 연결 또는 인스턴스 접속 시 LDAP 자원을 참조하는 노드 항목을 강제로 새로 고치려면 다음 명령을 사용하십시오.

```
db2 refresh ldap node directory
```

새로 고침의 일부로 로컬 데이터베이스 및 노드 디렉토리에 저장된 모든 LDAP 항목이 제거됩니다. 다음에 응용프로그램에서 데이터베이스 또는 노드에 액세스할 때 해당 응용프로그램은 LDAP에서 직접 정보를 읽어오고 로컬 데이터베이스 또는 노드에 새 항목을 생성합니다.

시기 적절한 방법으로 새로 고침이 수행되었는지 확인하기 위해 다음을 수행하고자 할 수 있습니다.

- 주기적으로 실행되는 새로 고침 스케줄
- 시스템 시동 중 REFRESH 명령 실행
- 사용 가능한 관리 패키지를 사용하여 모든 클라이언트 컴퓨터에서 REFRESH 명령 호출
- LDAP 정보가 데이터베이스, 노드 및 DCS 디렉토리에 캐시되지 않도록 DB2LDAPCACHE="NO" 설정

LDAP 서버 검색

DB2 데이터베이스 시스템은 현재 LDAP 서버를 검색하지만 여러 LDAP 서버가 있는 환경에서는 검색 범위를 정의할 수 있습니다.

예를 들어 현재 LDAP 서버에 정보가 없는 경우 다른 모든 LDAP 서버의 자동 검색을 지정할 수 있습니다. 또는 현재 LDAP 서버 또는 DB2 데이터베이스 카탈로그로 검색 범위를 제한할 수도 있습니다.

검색 범위를 설정하면 전체 엔터프라이즈의 디폴트 검색 범위가 설정됩니다. 검색 범위는 DB2 데이터베이스 프로파일 레지스트리 변수인 DB2LDAP_SEARCH_SCOPE를 통해 제어됩니다. 검색 범위 값을 설정하려면 LDAP 내에서 전역을 의미하는 -gl 옵션을 db2set 명령에서 사용하십시오.

```
db2set -gl db2ldap_search_scope=<value>
```

가능한 값은 로컬, 도메인 또는 전역입니다. 이러한 값을 설정하지 않으면 디폴트값은 검색 범위를 현재 LDAP 서버의 디렉토리로 제한하는 도메인입니다.

예를 들어 새 데이터베이스를 작성한 후 처음에 검색 범위를 『전역』으로 설정할 수 있습니다. 이렇게 하면 LDAP를 사용하도록 구성된 DB2 클라이언트에서 데이터베이스를 찾기 위해 모든 LDAP 서버를 검색할 수 있습니다. 각 클라이언트에 대한 첫 연결 또는 접속 후 항목이 각 클라이언트에 기록되면 캐싱이 사용 가능한 경우 검색 범위가 『로컬』로 변경될 수 있습니다. 『로컬』로 변경되면 각 클라이언트는 임의의 LDAP 서버를 스캔할 수 없습니다.

주: DB2 데이터베이스 프로파일 레지스트리 변수인 DB2LDAP_KEEP_CONNECTION 및 DB2LDAP_SEARCH_SCOPE는 LDAP에서 전역 레벨에서만 설정할 수 있는 유일한 레지스트리 변수입니다.

제 19 장 SQL 및 XML 한계

다음 테이블은 특정 SQL 및 XML 한계를 설명합니다. 가장 제한적인 사례를 충실히 지키면 쉽게 이식할 수 있는 응용프로그램을 설계할 수 있습니다.

표 56은 목록을 바이트로 나열합니다. ID 작성 시 응용프로그램 코드로부터 데이터베이스 코드 페이지로 변환 후 이들 제한이 강제 실행됩니다. 데이터베이스에서 ID 검색 시 데이터베이스 코드 페이지에서 응용프로그램 코드 페이지로 변환 후 제한이 강제 실행됩니다. 이들 프로세스 중 하나의 프로세스가 처리되는 동안 ID 길이 제한이 초과되면, 절단이 발생하거나 오류가 리턴됩니다.

데이터베이스의 코드 페이지 및 응용프로그램의 코드 페이지에 따라 문자 제한이 다를 수 있습니다. 예를 들어 UTF-8 문자의 너비가 1 - 4바이트 사이의 범위이기 때문에, 사용된 문자에 따라 128바이트가 제한인 유니코드 테이블의 ID에 대한 문자 제한이 32 - 128문자입니다. 데이터베이스 코드 페이지로 변환 후 이 테이블에 대한 제한보다 긴 이름의 ID를 작성하도록 시도하면, 오류가 리턴됩니다.

코드 페이지 변환이 발생한 다음 ID 이름을 저장하는 응용프로그램은 잠재적으로 늘어난 *크기*의 ID를 처리할 수 있어야 합니다. ID가 카탈로그에서 검색되면 응용프로그램 코드 페이지로 변환됩니다. 데이터베이스 코드 페이지에서 응용프로그램 코드 페이지로 변환되면 ID가 테이블에 대한 바이트 제한 보다 더 길게 됩니다. 코드 페이지 변환 후 응용프로그램에서 선언한 호스트 변수가 전체 ID를 저장할 수 없으면 절단됩니다. 승인할 수 없으면 전체 ID 이름을 승인할 수 있도록 호스트 변수의 *크기*를 늘릴 수 있습니다.

동일한 규칙이 사용자 지정 코드 페이지로 변환되며 데이터를 검색하는 DB2 유틸리티에 적용됩니다. 익스포트와 같은 DB2 유틸리티는 데이터를 검색하며 사용자 지정 코드 페이지로 강제 변환되고(CODEPAGE 수정자 또는 **DB2CODEPAGE** 레지스트리 변수 사용), 코드 페이지 변환때문에 이 테이블에서 문서화된 제한 이상으로 ID가 확장되면, 오류가 리턴되거나 ID가 절단될 수 있습니다.

표 56. ID 길이 한계

설명	바이트 단위 최대값
별명 이름	128
속성 이름	128
감사 규정 이름	128
권한 부여 이름(1바이트 문자만 가능)	128
버퍼 풀 이름	18
컬럼 이름[2]	128
제한조건 이름	128

표 56. ID 길이 한계 (계속)

설명	바이트 단위 최대값
상관 이름	128
커서 이름	128
데이터 파티션 이름	128
데이터 소스 컬럼 이름	255
데이터 소스 인덱스 이름	128
데이터 소스 이름	128
데이터 소스 테이블 이름(*remote-table-name*)	128
데이터베이스 파티션 그룹 이름	128
데이터베이스 파티션 이름	128
이벤트 모니터 이름	128
외부 프로그램 이름	128
함수 맵핑 이름	128
그룹 이름	128
호스트 ID[1]	255
데이터 소스 사용자 ID(*remote-authorization-name*)	128
SQL 프로시저의 ID(조건 이름, 루프 ID의 경우, 레이블, 결과 세트 로케이터, 명령문 이름, 변수 이름)	128
인덱스 이름	128
인덱스 확장 이름	18
인덱스 스펙 이름	128
레이블 이름	128
이름 스페이스 단일 자원 ID(URI)	1000
별칭	128
패키지 이름	128
패키지 버전 ID	64
매개변수 이름	128
데이터 소스를 액세스하기 위한 암호	32
프로시저 이름	128
역할 이름	128
세이브포인트 이름	128
스키마 이름[2]	128
보안 레이블 구성요소 이름	128
보안 레이블 이름	128
보안 규정 이름	128
시퀀스 이름	128
서버(데이터베이스 별명) 이름	8
특정 이름	128
SQL 조건 이름	128
SQL 변수 이름	128
명령문 이름	128

표 56. ID 길이 한계 (계속)

설명	바이트 단위 최대값
테이블 이름	128
테이블 스페이스 이름	18
변환 그룹 이름	18
트리거 이름	128
트러스트된 컨텍스트 이름	128
유형 맵핑 이름	18
사용자 정의 함수(UDF) 이름	128
사용자 정의 메소드 이름	128
사용자 정의 유형 이름[2]	128
뷰 이름	128
랩퍼 이름	128
XML 요소 이름, 속성 이름 또는 접두어 이름	1000
XML 스키마 위치 단일 자원 ID(URI)	1000

주:

1. 개별 호스트 언어 컴파일러에는 변수 이름에 대해 더 제한적인 한계가 있을 수 있습니다.
2. SQLDA 구조는 사용자 정의 유형에 대해 30바이트 컬럼 이름, 18바이트 사용자 정의 유형 이름 및 8바이트 스키마 이름을 저장하는 데 제한이 있습니다. SQLDA가 DESCRIBE문에서 사용되기 때문에 컬럼이나 사용자 정의 유형 이름을 검색하는 DESCRIBE문을 사용하는 Embedded SQL 응용프로그램은 이들 제한을 준수해야 합니다.

표 57. 숫자 한계

설명	한계
최소 SMALLINT 값	-32 768
최대 SMALLINT 값	+32 767
최소 INTEGER 값	-2 147 483 648
최대 INTEGER 값	+2 147 483 647
최소 BIGINT 값	-9 223 372 036 854 775 808
최대 BIGINT 값	+9 223 372 036 854 775 807
최대 Decimal 정밀도	31
REAL 값에 대한 최대 지수(E_{max})	38
최소 REAL 값	-3.402E+38
최대 REAL 값	+3.402E+38
REAL 값에 대한 최소 지수(E_{min})	-37
최소 양수 REAL 값	+1.175E-37
최대 음수 REAL 값	-1.175E-37
DOUBLE 값에 대한 최대 지수 (E_{max})	308
최소 DOUBLE 값	-1.79769E+308
최대 DOUBLE 값	+1.79769E+308

표 57. 숫자 한계 (계속)

설명	한계
DOUBLE 값에 대한 최소 지수 (E_{min})	-307
최소 양수 DOUBLE 값	+2.225E-307
최대 음수 DOUBLE 값	-2.225E-307
DECFLOAT(16) 값에 대한 최대 지수(E_{max})	384
가장 작은 DECFLOAT(16) 값[1]	-9.999999999999999E+384
가장 큰 DECFLOAT(16) 값	9.999999999999999E+384
DECFLOAT(16) 값에 대한 최소 지수(E_{min})	-383
가장 작은 양수 DECFLOAT(16) 값	1.000000000000000E-383
가장 큰 음수 DECFLOAT(16) 값	-1.000000000000000E-383
DECFLOAT(34) 값에 대한 최대 지수(E_{max})	6144
가장 작은 DECFLOAT(34) 값[1]	-9.999999999999999999999999999999999E+6144
가장 큰 DECFLOAT(34) 값	9.999999999999999999999999999999999E+6144
DECFLOAT(34) 값에 대한 최소 지수(E_{min})	-6143
가장 작은 양수 DECFLOAT(34) 값	1.000000000000000000000000000000000E-6143
가장 큰 음수 DECFLOAT(34) 값	-1.000000000000000000000000000000000E-6143

주:

1. 일반 10진수 부동 소수점의 제한이 있습니다. 유효한 10진수 부동 소수점 값은 특수 값 NAN, -NAN, SNAN, -SNAN, INFINITY 및 -INFINITY를 포함합니다. 또한 유효한 값은 subnormal 수를 포함합니다.

   subnormal 숫자는 조정된 지수가 E_{min} 미만인 0이 아닌 숫자입니다. subnormal 숫자의 경우 지수의 최소 값은 E_{tiny}라는 E_{min} - (*precision*-1)이며, 여기서 *precision*은 사용 중인 정밀도(16 또는 34)입니다. 즉 subnormal 숫자는 0에 가까운 숫자 범위를 DECFLOAT(16) 또는 DECFLOAT(34)에 대한 *크기*의 15나 33 순서로 각각 확장합니다. Subnormal 숫자는 subnormal 숫자의 최대 자릿수가 사용할 수 있는 정밀도(16 또는 34)보다 작으므로 일반 숫자와 다릅니다. 10진수 부동 소수점은 일반 숫자를 표시할 수 있는 같은 정밀도 subnormal 숫자를 표시할 수 없습니다. DECFLOAT(34)에 대한 가장 작은 양수의 subnormal 숫자는 $1x10^{-6176}$이며, 하나의 숫자만을 포함하는 데 반해, DECFLOAT(34)의 가장 작은 양수의 일반 숫자는 $1.000000000000000000000000000000000x10^{-6143}$이며, 34개의 숫자를 포함합니다. DECFLOAT(16)에 대한 가장 작은 양수의 subnormal 숫자는 $1x10^{-398}$입니다.

표 58. 문자열 한계

설명	한계
CHAR 최대 길이(바이트)	254
VARCHAR 최대 길이(바이트)	32 672

표 58. 문자열 한계 (계속)

설명	한계
LONG VARCHAR 최대 길이(바이트)[1]	32 700
CLOB 최대 길이(바이트)	2 147 483 647
연속 XML의 최대 길이(바이트)	2 147 483 647
GRAPHIC의 길이(2바이트 문자)	127
VARGRAPHIC의 최대 길이(2바이트 문자)	16 336
LONG VARGRAPHIC의 최대 길이(2바이트 문자)[1]	16 350
DBCLOB의 최대 길이(2바이트 문자)	1 073 741 823
BLOB 최대 길이(바이트)	2 147 483 647
문자 상수 최대 길이	32 672
그래픽 상수 최대 길이	16 336
병합 문자열 최대 길이	2 147 483 647
병합 그래픽 문자열 최대 길이	1 073 741 823
병합 실행 파일 문자열 최대 길이	2 147 483 647
16진 상수 최대 자릿수	32 672
런타임시(GB에서) 구조화된 유형 컬럼 오브젝트의 최대 인스턴스	1
카탈로그 주석 최대 크기(바이트)	254
주: 1. LONG VARCHAR 및 LONG VARGRAPHIC 데이터 유형은 사용되지 않으며 추후 릴리스에서 제거될 수 있습니다.	

표 59. XML 한계

설명	한계
XML 문서의 최대 용량(레벨)	125
XML 스키마 문서의 최대 크기(바이트)	31 457 280

표 60. 날짜 시간 한계

설명	한계
최소 DATE 값	0001-01-01
최대 DATE 값	9999-12-31
최소 TIME 값	00:00:00
최대 TIME 값	24:00:00
최소 TIMESTAMP 값	0001-01-01-00.00.00.000000000000
최대 TIMESTAMP 값	9999-12-31-24.00.00.000000000000
최소 시간소인 정밀도	0
최대 시간소인 정밀도	12

표 61. 데이터베이스 관리 프로그램 한계

설명	한계
응용프로그램	
프리컴파일된 프로그램의 최대 호스트 변수 선언 수[3]	스토리지
호스트 변수 값의 최대 길이(바이트)	2 147 483 647
프로그램의 선언된 커서의 최대수	스토리지
작업 단위(UOW)에서 변경된 최대 행 수	스토리지
한 번에 열리는 최대 커서 수	스토리지
DB2 클라이언트 내의 프로세스당 최대 연결 수	512
트랜잭션에서 동시에 열리는 최대 LOB 로케이터 수	4 194 304
SQLDA의 최대 *크기*(바이트)	스토리지
준비된 명령문의 최대 수	스토리지
버퍼 풀	
32비트 릴리스용 버퍼 풀의 최대 NPAGES	1 048 576
64비트 릴리스용 버퍼 풀의 최대 NPAGES	2 147 483 647
모든 버퍼 풀 슬롯의 최대 전체 *크기*(4K)	2 147 483 646
동시성	
서버의 최대 동시 사용자 수[4]	64 000
인스턴스당 최대 동시 사용자 수	64 000
데이터베이스당 최대 동시 응용프로그램 수	60 000
동시 사용되는 인스턴스당 최대 데이터베이스 수	256
제한조건	
테이블의 최대 제한조건 수	스토리지
UNIQUE 인덱스를 통해 지원되는 UNIQUE 제한조건의 최대 컬럼 수	64
UNIQUE 인덱스를 통해 지원되는 UNIQUE 제한조건에 있는 컬럼의 최대 조인 길이(바이트)[9]	8192
외부 키의 최대 참조 컬럼 수	64
외부 키에 있는 참조 컬럼의 최대 조인 길이(바이트)[9]	8192
점검 제한조건 스펙의 최대 길이(바이트)	65 535
데이터베이스	
최대 데이터베이스 파티션 번호	999
인덱스	
최대 테이블 인덱스 수	32 767 또는 스토리지
최대 인덱스 키 컬럼 수	64
오버헤드를 모두 포함한 인덱스 키의 최대 길이[7] [9]	*indexpagesize*/4
변수 인덱스 키 파트의 최대 길이(바이트)[8]	1022 또는 스토리지
SMS 테이블 스페이스의 데이터베이스 파티션당 인덱스의 최대 *크기*(GB)[7]	16 384
일반 DMS 테이블 스페이스의 데이터베이스 파티션당 인덱스의 최대 *크기*(GB)[7]	512
대형 DMS 테이블 스페이스의 데이터베이스 파티션당 인덱스의 최대 *크기*(GB)[7]	16 384

표 61. 데이터베이스 관리 프로그램 한계 (계속)

설명	한계
데이터베이스 파티션당 XML 데이터에 있는 인덱스의 최대 크기(TB)	2
XML 데이터에 있는 인덱스용 변수 인덱스 키 파트의 최대 길이(바이트)[7]	*pagesize*/4 - 207
로그 레코드	
최대 로그 시퀀스 번호	17 984 000 000 000 000 000
모니터링	
동시 사용 중인 이벤트 모니터의 최대 수	128
With DB2 데이터베이스 파티션 기능(DPF)을 통한 동시 활성 GLOBAL 이벤트 모니터의 최대 수	32
루틴	
프로시저의 최대 매개변수 수	32 767
커서 값 컨스트럭터의 최대 매개변수 수	32 767
사용자 정의 함수의 최대 매개변수 수	90
루틴의 최대 중첩 레벨 수	64
SQL 경로의 최대 스키마 수	64
SQL 경로의 최대 길이(바이트)	2048
보안	
유형 세트 또는 트리의 보안 레이블 구성요소의 최대 요소 수	64
유형 배열의 보안 레이블 구성요소의 최대 요소 수	65 535
보안 규정의 최대 보안 레이블 구성요소 수	16
SQL	
SQL문의 최대 총 길이(바이트)	2 097 152
SQL문 또는 뷰에서 참조되는 최대 테이블 수	스토리지
SQL문의 최대 호스트 변수 참조 수	32 767
명령문의 최대 상수 수	스토리지
선택 목록의 최대 요소 수[7]	1012
WHERE 또는 HAVING절의 최대 술어 수	스토리지
GROUP BY절의 최대 컬럼 수[7]	1012
GROUP BY절의 최대 총 컬럼 길이(바이트)[7]	32 677
ORDER BY절의 최대 컬럼 수[7]	1012
ORDER BY절의 최대 총 컬럼 길이(바이트)[7]	32 677
서브쿼리 중첩의 최대 레벨	스토리지
단일 명령문의 최대 서브쿼리 수	스토리지
삽입 조작에 있는 최대 값의 수[7]	1012
단일 갱신 조작에 있는 최대 SET절의 수[7]	1012
테이블 및 뷰	
테이블의 최대 컬럼 수[7]	1012
뷰의 최대 컬럼 수[1]	5000
별칭으로 참조되는 데이터 소스 테이블이나 뷰에서의 최대 컬럼 수	5000
분산 키의 최대 컬럼 수[5]	500

표 61. 데이터베이스 관리 프로그램 한계 (계속)

설명	한계
오버헤드를 모두 포함한 행의 최대 길이[2] [7]	32 677
데이터베이스 파티션당 파티션되지 않은 테이블의 최대 행 수	128 x 10^{10}
데이터베이스 파티션당 데이터 파티션의 최대 행 수	128 x 10^{10}
일반 테이블 스페이스의 데이터베이스 파티션당 테이블의 최대 크기 (GB)[3] [7]	512
대형 DMS 테이블 스페이스의 데이터베이스 파티션당 테이블의 최대 크기(GB)[7]	16 384
단일 테이블에 대한 최대 데이터 파티션 수	32 767
최대 테이블 파티션 컬럼 수	16
사용자 정의 행 유형의 최대 필드 수	1012
테이블 스페이스	
LOB 오브젝트의 최대 크기(TB)	4
LF 오브젝트의 최대 크기(TB)	2
데이터베이스 내의 최대 테이블 스페이스 수	32 768
SMS 테이블 스페이스의 최대 테이블 수	65 534
일반 DMS 테이블 스페이스의 최대 크기(GB)[3] [7]	512
대형 DMS 테이블 스페이스의 최대 크기(TB)[3] [7]	64
임시 DMS 테이블 스페이스의 최대 크기(TB) [3] [7]	64
DMS 테이블 스페이스의 최대 테이블 오브젝트 수[6]	51 000
자동 스토리지 데이터베이스의 최대 스토리지 경로 수	128
자동 스토리지 데이터베이스와 연결된 스토리지 경로 최대 길이(바이트)	175
트리거	
연쇄 트리거의 최대 런타임 용량	16
사용자 정의 유형	
구조화된 유형의 최대 속성 수	4082

표 61. 데이터베이스 관리 프로그램 한계 (계속)

설명	한계
주: 1. 이 최대값은 CREATE VIEW문에서 조인을 사용하여 얻을 수 있습니다. 이러한 뷰에서 선택하면 선택 목록의 대부분 요소가 제한됩니다. 2. BLOB, CLOB, LONG VARCHAR, DBCLOB 및 LONG VARGRAPHIC 컬럼의 실제 데이터는 이 계수에 포함되지 않습니다. 그러나 해당 데이터의 위치에 대한 정보는 행에서 일부 스페이스를 차지합니다. 3. 표시된 수는 구조적인 한계 및 근사값입니다. 실제 한계는 이것보다 작을 수 있습니다. 4. 실제 값은 **max_connections** 및 **max_coordagents** 데이터베이스 관리 프로그램 구성 매개변수에서 제어됩니다. 5. 이 한계는 구조적인 한계입니다. 인덱스 키의 최대 컬럼에 대한 제한이 실질적인 한계로 사용되어야 합니다. 6. 테이블 오브젝트는 데이터, 인덱스, LONG VARCHAR 또는 VARGRAPHIC 컬럼 및 LOB 컬럼을 포함합니다. 테이블 데이터와 동일한 테이블 스페이스에 있는 테이블 오브젝트의 한계에는 여분이 포함되어 있지 않습니다. 단, 테이블 데이터와 다른 테이블 스페이스에 있는 각 테이블 오브젝트는 테이블 오브젝트가 있는 테이블 스페이스의 테이블마다 있는 각 테이블 오브젝트 유형의 한계에 1을 더합니다. 7. 페이지 크기 특정 값에 대해서는 표 62의 내용을 참조하십시오. 8. 이는 모든 오버헤드를 포함하여 인덱스 키의 최대 길이(바이트)로만 제한됩니다. 인덱스 키 파트의 수가 증가함에 따라 각 키 파트의 최대 길이는 감소됩니다. 9. 인덱스 옵션에 따라 최대 길이가 더 짧아질 수 있습니다.	

표 62. 데이터베이스 관리 프로그램 페이지 크기 특정 한계

설명	4K 페이지 크기 한계	8K 페이지 크기 한계	16K 페이지 크기 한계	32K 페이지 크기 한계
테이블의 최대 컬럼 수	500	1012	1012	1012
오버헤드를 모두 포함한 행의 최대 길이	4005	8101	16 293	32 677
일반 테이블 스페이스의 데이터베이스 파티션당 테이블의 최대 크기(GB)	64	128	256	512
대형 DMS 테이블 스페이스의 데이터베이스 파티션당 테이블의 최대 크기(GB)	2048	4096	8192	16 384
오버헤드를 모두 포함한 인덱스 키의 최대 길이(바이트)	1024	2048	4096	8192
SMS 테이블 스페이스의 데이터베이스 파티션당 인덱스의 최대 크기(GB)	2048	4096	8192	16 384
일반 DMS 테이블 스페이스의 데이터베이스 파티션당 인덱스의 최대 크기(GB)	64	128	256	512

표 62. 데이터베이스 관리 프로그램 페이지 크기 특정 한계 (계속)

설명	4K 페이지 크기 한계	8K 페이지 크기 한계	16K 페이지 크기 한계	32K 페이지 크기 한계
대형 DMS 테이블 스페이스의 데이터베이스 파티션당 인덱스의 최대 크기(GB)	2048	4096	8192	16 384
데이터베이스 파티션당 XML 데이터에 있는 인덱스의 최대 크기 (TB)	2	2	2	2
정규 DMS 테이블 스페이스의 최대 크기(GB)	64	128	256	512
대형 DMS 테이블 스페이스의 최대 크기(TB)	8	14	32	64
임시 DMS 테이블 스페이스의 최대 크기(TB)	8	14	32	64
선택 목록의 최대 요소 수	500	1012	1012	1012
GROUP BY절의 최대 컬럼수	500	1012	1012	1012
GROUP BY절의 최대 컬럼 합계 길이(바이트)	4005	8101	16 293	32 677
ORDER BY절의 최대 컬럼 수	500	1012	1012	1012
ORDER BY절의 최대 컬럼 합계 길이(바이트)	4005	8101	16 293	32 677
삽입 조작에 있는 최대 값의 수	500	1012	1012	1012
단일 갱신 조작에 있는 최대 SET 절의 수	500	1012	1012	1012

제 20 장 레지스트리 및 환경 변수

환경 변수 및 프로파일 레지스트리

환경 변수 및 레지스트리 변수는 데이터베이스 환경을 제어합니다.

구성 지원 프로그램(db2ca)을 사용하여 레지스트리 변수 및 구성 매개변수를 구성할 수 있습니다.

DB2 데이터베이스 프로파일 레지스트리 도입 전에는 Windows 워크스테이션 등에서 환경 변수를 변경하려면 환경 변수를 변경한 다음 재시작해야 했습니다. 이제 몇 가지 예외 외에는 DB2 프로파일 레지스트리에 저장된 레지스트리 변수로 환경을 제어합니다. 지정된 인스턴스에 대한 시스템 관리(SYSADM) 권한이 있는 UNIX 운영 체제의 사용자는 해당 인스턴스의 레지스트리 값을 갱신할 수 있습니다. Windows에서 프로파일 레지스트리 변수를 갱신하려면 다음 조건에 따라 로컬 관리자 권한 또는 SYSADM 권한이 필요합니다.

- 확장 보안이 사용 가능한 경우, SYSADM 사용자는 DB2ADMNS 그룹에 속해야 합니다.
- 확장 보안이 사용 가능하지 않은 경우, Windows 레지스트리에서 해당 권한이 SYSADM 사용자에게 부여된 경우 해당 사용자가 갱신할 수 있습니다.

재시작하지 않고 레지스트리 변수를 갱신하려면 db2set 명령을 사용하십시오. 이 명령은 즉시 프로파일 레지스트리에 저장됩니다. 그러나 현재 실행 중인 DB2 응용프로그램 또는 사용자에게는 변경사항이 영향을 미치지 않습니다. DB2 레지스트리는 변경된 후 시작된 DB2 서버 인스턴스 및 DB2 응용프로그램에 갱신된 정보를 적용합니다.

주: DB2 프로파일 레지스트리에 저장되지 않는 DB2 환경 변수 **DB2INSTANCE** 및 **DB2NODE**가 있습니다. 일부 운영 체제에서는 이러한 환경 변수를 갱신하기 위해 set 명령을 사용해야 합니다. 이러한 변경사항은 다음에 시스템이 재시작될 때까지 적용됩니다. Linux 및 UNIX 플랫폼에서는 set 명령 대신에 export 명령이 사용됩니다.

프로파일 레지스트리는 중앙집중식 환경 변수 제어에 사용할 수 있습니다. 이제 여러 가지 프로파일을 통해 여러 가지 레벨의 지원이 제공됩니다. 또한 DB2 Administration Server를 사용하면 환경 변수의 리모트 관리도 사용 가능합니다.

네 가지 프로파일 레지스트리가 있습니다.

- DB2 인스턴스 레벨 프로파일 레지스트리. 대부분의 DB2 환경 변수는 이 레지스트리에 있습니다. 특정 인스턴스에 대한 환경 변수 설정은 이 레지스트리에 보존됩니다. 이 레벨에 정의된 값은 전역 레벨의 설정값을 겹쳐씁니다.

- DB2 전역 레벨 프로파일 레지스트리. 환경 변수가 특정 인스턴스에 대해 설정되지 않은 경우 이 레지스트리가 사용됩니다. 이 레지스트리는 특정 DB2 ESE 사본과 관련된 모든 인스턴스에 표시되며 하나의 글로벌 레벨 프로파일이 설치 경로에 있습니다.
- DB2 인스턴스 노드 레벨 프로파일 레지스트리. 이 레지스트리 레벨에는 파티션된 데이터베이스 환경의 데이터베이스 파티션에 특정한 변수 설정값이 포함되어 있습니다. 이 레벨에 정의된 값은 인스턴스 레벨 및 전역 레벨의 설정값을 겹쳐씁니다.
- DB2 인스턴스 프로파일 레지스트리. 이 레지스트리에는 현재 사본과 연관된 모든 인스턴스 이름 목록이 포함되어 있습니다. 각 설치에는 고유 목록이 있습니다. db2ilist를 실행하여 시스템에서 사용 가능한 모든 인스턴스의 전체 목록을 확인할 수 있습니다.

DB2는 다음 순서로 레지스트리 값 및 환경 변수를 점검하고 분석하여 운영 환경을 구성합니다.

1. set 명령을 사용하여 설정된 환경 변수. (UNIX 플랫폼에서는 export 명령)
2. 인스턴스 노드 레벨 프로파일을 사용하여 설정된 레지스트리 값(db2set -i <instance name> <nodenum> 명령 사용).
3. 인스턴스 레벨 프로파일을 사용하여 설정된 레지스트리 값(db2set -i 명령 사용).
4. 전역 레벨 프로파일을 사용하여 설정된 레지스트리 값(db2set -g 명령 사용).

인스턴스 레벨 프로파일 레지스트리

파티션된 데이터베이스 환경에서 작업하는 경우, UNIX와 Windows의 몇 가지 다른점이 있습니다. 이러한 다른점은 다음 예에서 설명됩니다.

『빨강』, 『흰색』 및 『파랑』으로 식별되는 세 가지 실제 데이터베이스 파티션이 있는 파티션된 데이터베이스 환경이 있다고 가정하십시오. UNIX 플랫폼에서 인스턴스 소유자가 데이터베이스 파티션 중 하나에서 다음을 실행하는 경우

```
db2set -i FOO=BAR
```

또는

```
db2set FOO=BAR        ('-i' is implied)
```

FOO 값은 현재 인스턴스의 모든 노드에 표시됩니다(즉, 『빨강』, 『흰색』 및 『파랑』).

UNIX 플랫폼에서는 인스턴스 레벨 프로파일 레지스트리는 sqllib 디렉토리 내부의 텍스트 파일에 저장됩니다. 파티션된 데이터베이스 환경에서 sqllib 디렉토리는 모든 실제 데이터베이스 파티션이 공유하는 파일 시스템에 있습니다.

Windows 플랫폼에서 사용자가 『빨강』에서 동일한 명령을 수행하는 경우, FOO 값은 현재 인스턴스의 『빨강』에만 표시됩니다. DB2 데이터베이스 관리 프로그램은 Windows 레지스트리 내부에 인스턴스 레벨 프로파일 레지스트리를 저장합니다. 실제 데이터베이

스 파티션 간에는 공유되지 않습니다. 모든 실제 컴퓨터에 레지스트리 변수를 설정하려면 다음과 같이 『rah』 명령을 사용하십시오.

```
rah db2set -i FOO=BAR
```

rah는 『빨강』, 『흰색』 및 『파랑』에서 db2set 명령을 리모트로 실행합니다.

DB2REMOTEPREG를 사용하여 인스턴스를 소유하지 않는 컴퓨터의 레지스트리 변수가 인스턴스 소유 컴퓨터의 레지스트리 변수를 참조하도록 구성할 수 있습니다. 이는 인스턴스 소유 컴퓨터의 레지스트리 변수가 인스턴스의 모든 컴퓨터 사이에서 공유되는 환경을 효과적으로 작성합니다.

위의 예를 사용하고 『빨강』이 소유 컴퓨터라고 가정하면 다음을 수행하여 『빨강』의 레지스트리 변수를 공유하도록 『흰색』 및 『파랑』 컴퓨터에 **DB2REMOTEPREG**를 설정합니다.

```
(on red) do nothing
(on white and blue) db2set DB2REMOTEPREG=WWred
```

DB2REMOTEPREG의 설정값은 설정된 후 변경되지 않아야 합니다.

REMOTEPREG가 작동하는 방식은 다음과 같습니다.

DB2 데이터베이스 관리 프로그램이 Windows에서 레지스트리 변수를 읽는 경우, 해당 프로그램은 먼저 **DB2REMOTEPREG** 값을 읽습니다. **DB2REMOTEPREG**가 설정된 경우에는 **DB2REMOTEPREG** 변수에 컴퓨터 이름이 지정된 리모트 컴퓨터에서 레지스트리를 엽니다. 이후의 레지스트리 변수 읽기 및 갱신은 지정된 리모트 컴퓨터로 경로 재지정됩니다.

리모터 레지스트리에 액세스하려면 리모트 레지스트리 서비스가 목표 컴퓨터에서 실행 중이어야 합니다. 또한 사용자 로그온 어카운트 및 모든 DB2 서비스 로그온 어카운트에 리모트 레지스트리에 대한 충분한 액세스 권한이 있어야 합니다. 따라서 **DB2REMOTEPREG**를 사용하려면 도메인 어카운트에 필수 레지스트리 액세스 권한이 부여되도록 Windows 도메인 환경에서 운영해야 합니다.

MSCS(Microsoft Cluster Server) 고려사항이 있습니다. MSCS 환경에서는 **DB2REMOTEPREG**를 사용하지 않아야 합니다. 모든 컴퓨터가 같은 MSCS 클러스터에 속하는 MSCS 구성에서 실행 중인 경우, 레지스트리 변수는 클러스터 레지스트리에 유지됩니다. 따라서 레지스트리 변수는 같은 MSCS 클러스터의 모든 컴퓨터 사이에서 이미 공유되며 이 경우 **DB2REMOTEPREG**를 사용할 필요가 없습니다.

데이터베이스 파티션이 여러 MSCS 클러스터에 속하는 다중 파티션 장애 복구 환경에서 실행 중인 경우, 인스턴스 소유 컴퓨터의 레지스트리 변수가 클러스터 레지스트리에 상주하므로 **DB2REMOTEPREG**를 사용하여 인스턴스 소유 컴퓨터를 지정할 수 없습니다.

레지스트리 및 환경 변수 선언, 표시, 변경, 재설정 및 삭제

모든 특정 레지스트리 변수를 DB2 데이터베이스 프로파일 레지스트리에 정의하는 것이 좋습니다. DB2 변수를 레지스트리 외부에 설정하는 경우, 해당 변수를 리모트로 관리할 수 없으며 변수값을 적용하려면 워크스테이션을 재시작해야 합니다.

db2set 명령은 레지스트리 변수 및 환경 변수의 로컬 선언을 지원합니다.

해당 명령에 대한 도움말 정보를 표시하려면 다음을 사용하십시오.

```
db2set -?
```

지원되는 모든 레지스트리 변수의 전체 세트를 나열하려면 다음을 사용하십시오.

```
db2set -lr
```

현재 또는 디폴트 인스턴스에 대해 정의된 모든 레지스트리 변수를 나열하려면 다음을 사용하십시오.

```
db2set
```

프로파일 레지스트리에 정의된 모든 레지스트리 변수를 나열하려면 다음을 사용하십시오.

```
db2set -all
```

현재 또는 디폴트 인스턴스의 레지스트리 변수 값을 표시하려면 다음을 사용하십시오.

```
db2set registry_variable_name
```

모든 레벨의 레지스트리 변수 값을 표시하려면 다음을 사용하십시오.

```
db2set registry_variable_name -all
```

현재 또는 디폴트 인스턴스의 레지스트리 변수를 변경하려면 다음을 사용하십시오.

```
db2set registry_variable_name=new_value
```

인스턴스의 모든 데이터베이스에 대한 레지스트리 변수 디폴트를 변경하려면 다음을 사용하십시오.

```
db2set registry_variable_name=new_value
    -i instance_name
```

인스턴스의 특정 데이터베이스 파티션에 대한 레지스트리 변수 디폴트를 변경하려면 다음을 사용하십시오.

```
db2set registry_variable_name=new_value
    -i instance_name database_partition_number
```

시스템의 특정 설치와 관련된 모든 인스턴스에 대한 레지스트리 변수 디폴트를 변경하려면 다음을 사용하십시오.

```
db2set registry_variable_name=new_value -g
```

DB2_WORKLOAD와 같은 집계 레지스트리 변수를 사용하여 SAP 환경에 대해 레지스트리 변수를 구성하는 경우, 다음을 사용하여 해당 변수를 설정할 수 있습니다.

```
db2set DB2_WORKLOAD=SAP
```

LDAP(Lightweight Directory Access Protocol)을 사용하는 경우, 다음을 사용하여 LDAP에 레지스트리 변수를 설정할 수 있습니다.

- LDAP 내에 사용자 레벨의 레지스트리 변수를 설정하려면 다음을 사용하십시오.

  ```
  db2set -ul
  ```

- LDAP 내에 전역 레벨의 레지스트리 변수를 설정하려면 다음을 사용하십시오.

  ```
  db2set -gl user_name
  ```

LDAP 환경에서 실행 중인 경우, 해당 범위가 디렉토리 파티션 또는 Windows 도메인에 속하는 모든 서버 및 모든 사용자에 대해 전역이 되도록 DB2 레지스트리 변수값을 설정할 수 있습니다. 현재 LDAP 글로벌 레벨에서 설정할 수 있는 DB2 레지스트리 변수는 **DB2LDAP_KEEP_CONNECTION** 및 **DB2LDAP_SEARCH_SCOPE** 두 가지뿐입니다.

예를 들어, LDAP에서 글로벌 레벨의 검색 범위 값을 설정하려면 다음을 사용하십시오.

```
db2set -gl db2ldap_search_scope = value
```

여기서 값은 로컬, 도메인 또는 전역 중 하나입니다.

주:

1. 둘 이상의 사용자가 동시 또는 매우 근접한 시간에 db2set 명령을 사용하여 DB2 profile.env 파일을 갱신하는 경우, profile.env 파일의 크기는 0으로 줄어듭니다. 또한 db2set -all의 출력은 불일치 값을 표시합니다.
2. DB2 ESE의 같은 설치와 관련된 모든 인스턴스에 적용되는 DB2 레지스트리 변수를 설정하는 데 사용되는 -g 옵션과 LDAP 전역 레벨에 특정적으로 사용되는 -gl 옵션 간에는 차이가 있습니다.
3. LDAP 환경에서 실행 중인 경우 Windows에서는 사용자 레벨 레지스트리 변수만 지원됩니다.
4. 사용자 레벨의 변수 설정값에는 사용자 특정 변수 설정값이 포함됩니다. 사용자 레벨에 대한 모든 변경사항은 LDAP 디렉토리에 기록됩니다.
5. "-i", "-g", "-gl" 및 "-ul" 매개변수는 같은 명령에 동시에 사용할 수 없습니다.
6. 일부 변수는 전역 레벨 프로파일에 대해 항상 디폴트로 설정됩니다(전역이란 같은 DB2 사본에서 실행 중인 모든 인스턴스 간에 변수가 공유됨을 의미함). 이러한 변

수는 인스턴스 또는 데이터베이스 파티션 레벨 프로파일에서 설정할 수 없습니다 (예: **DB2SYSTEM** 및 **DB2INSTDEF**).

7. UNIX에서 인스턴스의 레지스트리 값을 변경하려면 시스템 관리(SYSADM) 권한이 있어야 합니다. 루트 권한이 있는 사용자만 전역 레벨 레지스트리의 매개변수를 변경할 수 있습니다.

인스턴스의 레지스트리 변수를 전역 프로파일 레지스트리에 있는 디폴트로 재설정하려면 다음을 사용하십시오.

```
db2set -r registry_variable_name
```

인스턴스의 데이터베이스 파티션에 대한 레지스트리 변수를 전역 프로파일 레지스트리에 있는 디폴트로 재설정하려면 다음을 사용하십시오.

```
db2set -r registry_variable_name database_partition_number
```

지정된 레벨의 변수 값을 삭제하기 위해 동일한 명령 구문을 사용하여 변수를 설정하고 변수값을 지정하지 않을 수 있습니다. 예를 들어, 데이터베이스 파티션 레벨의 변수 설정값을 삭제하려면 다음을 입력하십시오.

```
db2set registry_variable_name= -i instance_name
   database_partition_number
```

변수 값을 삭제하고 사용을 제한하려면 상위 프로파일 레벨에 정의된 경우, 다음을 입력하십시오.

```
db2set registry_variable_name= -null -r instance_name
```

이 명령은 지정한 매개변수의 설정을 삭제하며 상위 레벨 프로파일에서 이 변수 값을 변경하는 것을 제한합니다(이 경우, DB2 전역 레벨 프로파일). 그러나 하위 레벨 프로파일에서는 지정한 변수를 계속 설정할 수 있습니다(이 경우, DB2 데이터베이스 파티션 레벨 프로파일).

Windows에서 환경 변수 설정

Windows 운영 체제에는 프로파일 레지스트리 외부에만 설정할 수 있는 하나의 시스템 환경 변수 **DB2INSTANCE**가 있습니다. 그러나 **DB2INSTANCE**를 설정할 필요는 없습니다. **DB2INSTANCE**가 정의되지 않은 경우, DB2 프로파일 레지스트리 변수 **DB2INSTDEF**를 전역 레벨 프로파일에 설정하여 사용하려는 인스턴스 이름을 지정하십시오.

Windows의 DB2 Enterprise Server Edition 서버에는 프로파일 레지스트리 외부에만 설정할 수 있는 두 가지 시스템 환경 변수 **DB2INSTANCE** 및 **DB2NODE**가 있습니다. **DB2INSTANCE**를 설정할 필요는 없습니다. **DB2INSTANCE**가 정의되지 않은 경우, DB2 프로파일 레지스트리 변수 **DB2INSTDEF**를 전역 레벨 프로파일에 설정하여 사용하려는 인스턴스 이름을 지정하십시오.

DB2NODE 환경 변수는 컴퓨터 내의 목표 논리 노드에 대한 요청의 경로를 지정하는 데 사용됩니다. 이 환경 변수는 DB2 프로파일 레지스트리가 아니라 응용프로그램 또는 명령이 발행된 세션에 설정해야 합니다. 이 변수가 설정되지 않은 경우, 목표 논리 노드는 디폴트로 컴퓨터에서 0으로 정의된 논리 노드로 설정됩니다.

환경 변수의 설정값을 판별하려면 echo 명령을 사용하십시오. 예를 들어, **DB2PATH** 환경 변수 값을 점검하려면 다음을 입력하십시오.

```
echo %db2path%
```

다음과 같이 DB2 환경 변수 **DB2INSTANCE** 및 **DB2NODE**를 설정할 수 있습니다 (이 설명에서 **DB2INSTANCE** 사용).

- 내 컴퓨터를 마우스 오른쪽 단추로 누르고 등록 정보를 선택하십시오.
- 고급 탭을 선택하고 환경 변수를 누른 후 다음을 수행하십시오.
  1. **DB2INSTANCE** 변수가 없는 경우:
     a. 새로 작성을 누르십시오.
     b. 변수 이름 필드를 **DB2INSTANCE**로 채우십시오.
     c. 변수 값 필드를 인스턴스 이름으로 채우십시오(예: db2inst).
  2. **DB2INSTANCE** 변수가 이미 있는 경우, 새 값을 추가하십시오.
     a. **DB2INSTANCE** 환경 변수를 선택하십시오.
     b. 값 필드를 인스턴스 이름으로 변경하십시오(예: db2inst).
  3. 이러한 변경사항이 적용되도록 시스템을 재시작하십시오.

주: 환경 변수 **DB2INSTANCE**를 세션(프로세스) 레벨에서 설정할 수도 있습니다. 예를 들어, TEST라는 두 번째 DB2 인스턴스를 시작하려는 경우, 명령 창에서 다음 명령을 발행하십시오.

```
set DB2INSTANCE=TEST
db2start
```

C 쉘에서 작업 중인 경우, 명령 창에서 다음 명령을 발행하십시오.

```
setenv DB2INSTANCE TEST
```

프로파일 레지스트리는 다음 위치에 있습니다.

- Windows 운영 체제 레지스트리의 DB2 인스턴스 레벨 프로파일 레지스트리. 경로는 다음과 같습니다.

  ```
  ₩HKEY_LOCAL_computer₩SOFTWARE₩IBM₩DB2₩PROFILES₩instance_name
  ```

  주: *instance_name*은 DB2 인스턴스의 이름입니다.

- Windows 레지스트리의 DB2 전역 레벨 프로파일 레지스트리. 경로는 다음과 같습니다.

  ```
  ₩HKEY_LOCAL_computer₩SOFTWARE₩IBM₩DB2₩GLOBAL_PROFILE
  ```

- Windows 레지스트리의 DB2 인스턴스 노드 레벨 프로파일 레지스트리. 경로는 다음과 같습니다.

  ...₩SOFTWARE₩IBM₩DB2₩PROFILES₩*instance_name*₩NODES₩*node_number*

  주: *instance_name* 및 *node_number*는 작업 중인 데이터베이스 파티션에 따라 다릅니다.
- DB2 인스턴스 프로파일 레지스트리는 필요하지 않습니다. 시스템의 각 DB2 인스턴스에 대해 키는 다음 경로에 작성됩니다.

  ₩HKEY_LOCAL_computer₩SOFTWARE₩IBM₩DB2₩PROFILES₩*instance_name*

  인스턴스 목록은 PROFILES 키 아래에 있는 키를 카운팅해서 얻을 수 있습니다.

Linux 및 UNIX 운영 체제에서 환경 변수 설정

UNIX 운영 체제에서는 시스템 환경 변수 **DB2INSTANCE**를 설정해야 합니다.

인스턴스 환경 설정에 도움이 되도록 db2profile(본 쉘 또는 콘 쉘의 경우) 및 db2cshrc(C 쉘의 경우) 스크립트가 예로 제공됩니다. 해당 파일은 insthome/sqllib에서 찾을 수 있으며 여기서 insthome은 인스턴스 소유자의 홈 디렉토리입니다.

이러한 스크립트에는 다음을 수행하는 명령문이 포함되어 있습니다.

- 다음 디렉토리로 사용자 경로를 갱신합니다.
  - insthome/sqllib/bin
  - insthome/sqllib/adm
  - insthome/sqllib/misc
- 실행을 위해 **DB2INSTANCE**를 디폴트 로컬 instance_name으로 설정합니다.

주: PATH 및 **DB2INSTANCE**를 제외한 지원되는 기타 모든 변수는 DB2 프로파일 레지스트리에 설정해야 합니다. DB2 데이터베이스 관리 프로그램에서 지원되지 않는 변수를 설정하려면 해당 변수를 사용자 스크립트 파일 userprofile 및 usercshrc에 정의하십시오.

인스턴스 소유자 또는 SYSADM 사용자는 인스턴스의 모든 사용자를 위해 이러한 스크립트를 사용자 정의할 수 있습니다. 또는 사용자가 스크립트를 복사 및 사용자 정의한 다음 스크립트를 직접 호출하거나 .profile 또는 .login 파일에 추가할 수 있습니다.

현재 세션의 환경 변수를 변경하려면 다음과 유사한 명령을 발행하십시오.

- 콘 쉘의 경우:

  ```
  DB2INSTANCE=<inst1>
  export DB2INSTANCE
  ```
- 본 쉘의 경우:

```
export DB2INSTANCE=<inst1>
```

- C 쉘의 경우:

```
setenv DB2INSTANCE <inst1>
```

DB2 프로파일을 올바르게 관리하기 위해 UNIX 운영 체제에서는 다음과 같은 파일 소유권 규칙을 따라야 합니다.

- DB2 인스턴스 레벨 프로파일 레지스트리 파일은 다음 아래에 있습니다.

  ```
  INSTHOME/sqllib/profile.env
  ```

  이 파일의 액세스 권한 및 소유권은 다음과 같아야 합니다.

  ```
  -rw-rw-r-- <db2inst1> <db2iadm1> profile.env
  ```

  여기서 <db2inst1>은 인스턴스 소유자이며 <db2iadm1>은 인스턴스 소유자의 그룹입니다.

  INSTHOME은 인스턴스 소유자의 홈 경로입니다.

- DB2 전역 레벨 프로파일 레지스트리는 전역 레지스트리에 저장됩니다(global.reg).

  루트 설치에서 전역 레지스트리 변수를 수정하려면 루트 권한으로 로그온하고 -g 매개변수를 포함한 db2set 명령을 실행해야 합니다.

- DB2 인스턴스 노드 레벨 프로파일 레지스트리는 다음 아래에 있습니다.

  ```
  INSTHOME/sqllib/nodes/<node_number>.env
  ```

  디렉토리 및 이 파일의 액세스 권한 및 소유권은 다음과 같아야 합니다.

  ```
  drwxrwsr-w <Instance_Owner> <Instance_Owner_Group> nodes

  -rw-rw-r-- <Instance_Owner> <Instance_Owner_Group> <node_number>.env
  ```

  INSTHOME은 인스턴스 소유자의 홈 경로입니다.

현재 인스턴스 환경 변수 설정

인스턴스의 데이터베이스 관리 프로그램을 시작하거나 중지하는 명령을 실행하면 DB2에서 해당 명령을 현재 인스턴스에 적용합니다. DB2는 현재 인스턴스를 다음과 같이 판별합니다.

- **DB2INSTANCE** 환경 변수가 현재 세션에 대해 설정된 경우, 해당 값은 현재 인스턴스입니다. **DB2INSTANCE**를 설정하려면 다음을 입력하십시오.

  ```
  set db2instance=<new_instance_name>
  ```

- **DB2INSTANCE**가 현재 세션에 대해 설정되지 않은 경우, DB2 데이터베이스 관리 프로그램은 시스템 환경 변수에서 **DB2INSTANCE** 환경 변수에 대한 설정을 사용하지 않습니다. Windows에서 시스템 환경 변수는 시스템 환경 레지스트리에 설정됩니다.

- **DB2INSTANCE**가 설정되지 않은 경우, DB2 데이터베이스 관리 프로그램은 레지스트리 변수 **DB2INSTDEF**를 사용합니다.

  DB2INSTDEF 레지스트리 변수를 레지스트리 전역 레벨에서 설정하려면 다음을 입력하십시오.

  ```
  db2set db2instdef=<new_instance_name> -g
  ```

현재 세션에 적용되는 인스턴스를 판별하려면 다음을 입력하십시오.

```
db2 get instance
```

집계 레지스트리 변수

집계 레지스트리 변수를 사용하여 여러 가지 레지스트리 변수를 다른 레지스트리 변수 이름으로 식별되는 구성으로 그룹화할 수 있습니다. 그룹의 일부인 각 레지스트리 변수에는 사전 정의된 설정이 있습니다. 집계 레지스트리 변수에는 여러 가지 레지스트리 변수 선언으로 해석되는 값이 지정됩니다.

집계 레지스트리 변수는 광범위한 운영 목적에 적합한 레지스트리 구성을 쉽게 하기 위한 것입니다.

유효한 집계 레지스트리 변수는 **DB2_WORKLOAD**뿐입니다.

이 변수에 유효한 값은 다음과 같습니다.

- 1C
- CM
- SAP
- TPM
- WC

집계 레지스트리 변수를 통해 내재적으로 구성된 레지스트리 변수는 명시적으로도 정의됩니다. 이전에 집계 레지스트리 변수 사용을 통해 값이 지정된 레지스트리 변수의 명시적 설정은 성능 또는 진단 테스트를 수행할 때 유용합니다. 집계에 의해 내재적으로 구성된 변수의 명시적 설정을 변수 겹쳐쓰기라고 합니다.

집계 레지스트리 변수를 사용하여 명시적으로 설정된 레지스트리 변수를 수정하려고 시도하는 경우, 경고가 발행되고 명시적으로 설정된 값이 보존됩니다. 이 경고는 명시된 값이 유지됨을 나타냅니다. 집계 레지스트리 변수가 먼저 사용된 다음 명시된 레지스트리 변수를 지정하면 경고가 표시되지 않습니다.

집계 레지스트리 변수 설정을 통해 구성된 레지스트리 변수는 각 변수에 대해 표시 요청이 명시적으로 작성되지 않은 경우 표시되지 않습니다. 집계 레지스트리 변수를 쿼리하면 해당 변수에 지정된 값만 표시됩니다. 대부분의 사용자는 각 개별 변수의 값을 고려할 필요가 없습니다.

다음 예에서는 집계 레지스트리 변수 사용과 레지스트리 변수 명시적 설정 간의 상호작용에 대해 설명합니다. 예를 들어, **DB2_WORKLOAD** 집계 레지스트리 변수를 SAP로 설정하고 **DB2_SKIPDELETED** 레지스트리 변수를 NO로 겹쳐씁니다. db2set을 입력하면 다음 결과가 수신됩니다.

```
DB2_WORKLOAD=SAP
DB2_SKIPDELETED=NO
```

다른 상황에서는 **DB2ENVLIST**를 설정하고 **DB2_WORKLOAD** 집계 레지스트리 변수를 SAP로 설정한 다음 **DB2_SKIPDELETED** 레지스트리 변수를 NO로 겹쳐씁니다. (**DB2_SKIPDELETED** 레지스트리 변수가 SAP 환경을 구성하는 그룹의 일부라고 가정합니다.) 또한 집계 레지스트리 변수 설정을 통해 자동 구성된 레지스트리 변수는 값 옆에 있는 대괄호 내에 집계 이름을 표시합니다. **DB2_SKIPDELETED** 레지스트리 변수는 NO 값을 표시하고 값 옆에 [O]을 표시합니다.

DB2_WORKLOAD와 연관된 구성이 더 이상 필요하지 않은 경우, 다음 명령을 통해 집계 레지스트리 변수 값을 삭제하여 그룹에 있는 각 레지스트리 변수의 내재된 값을 사용하지 않도록 할 수 있습니다.

```
db2set DB2_WORKLOAD=
```

DB2_WORKLOAD 집계 레지스트리 변수 값을 삭제한 후 데이터베이스를 재시작하십시오. 데이터베이스가 재시작되면 집계 레지스트리 변수에 의해 내재적으로 구성된 레지스트리 변수는 더 이상 적용되지 않습니다. 집계 레지스트리 값을 삭제하는 데 사용되는 메소드는 개별 레지스트리 변수 삭제와 동일합니다.

집계 레지스트리 변수 값 삭제는 명시적으로 설정된 레지스트리 변수 값을 삭제하지 않습니다. 레지스트리 변수가 사용하지 않는 그룹 정의의 구성원인지는 중요하지 않습니다. 레지스트리 변수의 명시적 설정은 유지됩니다.

DB2_WORKLOAD 집계 레지스트리 변수의 구성원인 각 레지스트리 변수의 값을 확인해야 할 수 있습니다. 예를 들어, **DB2_WORKLOAD**를 SAP로 구성한 경우 사용되는 값을 확인하려고 할 수 있습니다. **DB2_WORKLOAD**=SAP인 경우 사용될 값을 찾으려면 db2set -gd DB2_WORKLOAD=SAP를 실행하십시오.

DB2 레지스트리 및 환경 변수

DB2 데이터베이스 제품은 시작 및 실행을 위해 알아야 하는 여러 가지 레지스트리 변수 및 환경 변수를 제공합니다.

지원되는 모든 레지스트리 변수 목록을 보려면 다음 명령을 실행하십시오.

```
db2set -lr
```

현재 또는 디폴트 인스턴스의 변수 값을 변경하려면 다음 명령을 실행하십시오.

```
db2set registry_variable_name=new_value
```

DB2 환경 변수 **DB2INSTANCE**, **DB2NODE**, **DB2PATH** 및 **DB2INSTPROF**가 DB2 프로파일 레지스트리에 저장되는지 여부는 운영 체제에 따라 다릅니다. 이러한 환경 변수를 갱신하려면 set 명령을 사용하십시오. 이러한 변경사항은 로컬(현재) 명령 프롬프트에서만 유효하며 다음에 시스템이 재부트될 때까지 적용됩니다. Linux 및 UNIX 운영 체제에서는 set 명령 대신에 export 명령을 사용할 수 있습니다.

db2start 명령을 실행하기 전에 변경된 레지스트리 변수에 대한 값을 설정해야 합니다.

주: 레지스트리 변수에 부울 값이 인수로 필요한 경우, YES, 1 및 ON 값은 모두 동등한 값이며 NO, 0 및 OFF 값 또한 동등한 값입니다. 모든 변수에 대해 동등한 해당 값 중 하나를 지정할 수 있습니다.

다음 표에서는 범주별 모든 레지스트리 변수를 나열합니다.

표 63. 레지스트리 및 환경 변수 요약

변수 범주	레지스트리 또는 환경 변수 이름
일반	**DB2ACCOUNT** **DB2BIDI** **DB2_CAPTURE_LOCKTIMEOUT** **DB2CODEPAGE** **DB2_COLLECT_TS_REC_INFO** **DB2_CONNRETRIES_INTERVAL** **DB2CONSOLECP** **DB2COUNTRY** **DB2DBDFT** **DB2DBMSADDR** **DB2DISCOVERYTIME** **DB2FFDC** **DB2FODC** **DB2_FORCE_APP_ON_MAX_LOG** **DB2GRAPHICUNICODESERVER** **DB2INCLUDE** **DB2INSTDEF** **DB2INSTOWNER** **DB2_LIC_STAT_SIZE** **DB2LOCALE** **DB2_MAX_CLIENT_CONNRETRIES** **DB2_OBJECT_TABLE_ENTRIES** **DB2_SYSTEM_MONITOR_SETTINGS** **DB2TERRITORY** **DB2_VIEW_REOPT_VALUES**

표 63. 레지스트리 및 환경 변수 요약 (계속)

변수 범주	레지스트리 또는 환경 변수 이름
시스템 환경	**DB2_ALTERNATE_GROUP_LOOKUP** **DB2CONNECT_ENABLE_EURO_CODEPAGE** **DB2CONNECT_IN_APP_PROCESS** **DB2_COPY_NAME** **DB2DBMSADDR** **DB2_DIAGPATH** **DB2DOMAINLIST** **DB2ENVLIST** **DB2INSTANCE** **DB2INSTPROF** **DB2LDAPSecurityConfig** **DB2LIBPATH** **DB2LOGINRESTRICTIONS** **DB2NODE** **DB2OPTIONS** **DB2_PARALLEL_IO** **DB2PATH** **DB2_PMAP_COMPATIBILITY** **DB2PROCESSORS** **DB2RCMD_LEGACY_MODE** **DB2RESILIENCE** **DB2SYSTEM** **DB2_UPDDBCFG_SINGLE_DBPARTITION** **DB2_USE_PAGE_CONTAINER_TAG** **DB2_WORKLOAD**
통신	**DB2CHECKCLIENTINTERVAL** **DB2COMM** **DB2FCMCOMM** **DB2_FORCE_NLS_CACHE** **DB2RSHCMD** **DB2RSHTIMEOUT** **DB2SORCVBUF** **DB2SOSNDBUF** **DB2TCP_CLIENT_CONTIMEOUT** **DB2TCP_CLIENT_RCVTIMEOUT** **DB2TCPCONNMGRS**
명령행	**DB2BQTIME** **DB2BQTRY** **DB2_CLP_EDITOR** **DB2_CLP_HISTSIZE** **DB2_CLPPROMPT** **DB2IQTIME** **DB2RQTIME**
파티션된 데이터베이스 환경	**DB2CHGPWD_EEE** **DB2_FCM_SETTINGS** **DB2_FORCE_OFFLINE_ADD_PARTITION** **DB2_NUM_FAILOVER_NODES** **DB2_PARTITIONEDLOAD_DEFAULT** **DB2PORTRANGE**
쿼리 컴파일러	**DB2_ANTIJOIN** **DB2_DEFERRED_PREPARE_SEMANTICS** **DB2_INLIST_TO_NLJN** **DB2_LIKE_VARCHAR** **DB2_MINIMIZE_LISTPREFETCH** **DB2_NEW_CORR_SQ_FF** **DB2_OPT_MAX_TEMP_SIZE** **DB2_REDUCED_OPTIMIZATION** **DB2_SELECTIVITY** **DB2_SQLROUTINE_PREPOPTS**

표 63. 레지스트리 및 환경 변수 요약 (계속)

변수 범주	레지스트리 또는 환경 변수 이름
성능	**DB2_ALLOCATION_SIZE** **DB2_APM_PERFORMANCE** **DB2ASSUMEUPDATE** **DB2_ASYNC_IO_MAXFILOP** **DB2_AVOID_PREFETCH** **DB2BPVARS** **DB2CHKPTR** **DB2CHKSQLDA** **DB2_EVALUNCOMMITTED** **DB2_EXTENDED_IO_FEATURES** **DB2_EXTENDED_OPTIMIZATION** **DB2_IO_PRIORITY_SETTING** **DB2_IO_PRIORITY_SETTING** **DB2_KEEP_AS_AND_DMS_CONTAINERS_OPEN** **DB2_KEEPTABLELOCK** **DB2_LARGE_PAGE_MEM** **DB2_LOGGER_NON_BUFFERED_IO** **DB2MAXFSCRSEARCH** **DB2_MAX_INACT_STMTS** **DB2_MAX_NON_TABLE_LOCKS** **DB2_MDC_ROLLOUT** **DB2MEMDISCLAIM** **DB2MEMMAXFREE** **DB2_MEM_TUNING_RANGE** **DB2_MMAP_READ** **DB2_MMAP_WRITE** **DB2_NO_FORK_CHECK** **DB2NTMEMSIZE** **DB2NTNOCACHE** **DB2NTPRICLASS** **DB2NTWORKSET** **DB2_OVERRIDE_BPF** **DB2_PINNED_BP** **DB2PRIORITIES** **DB2_RESOURCE_POLICY** **DB2_SET_MAX_CONTAINER_SIZE** **DB2_SKIPDELETED** **DB2_SKIPINSERTED** **DB2_SMS_TRUNC_TMPTABLE_THRESH** **DB2_SORT_AFTER_TQ** **DB2_SELUDI_COMM_BUFFER** **DB2_TRUSTED_BINDIN** **DB2_USE_ALTERNATE_PAGE_CLEANING** **DB2_USE_IOCP**

표 63. 레지스트리 및 환경 변수 요약 (계속)

변수 범주	레지스트리 또는 환경 변수 이름
기타	**DB2ADMINSERVER** **DB2_ATS_ENABLE** **DB2AUTH** **DB2CLIINIPATH** **DB2_COMMIT_ON_EXIT** **DB2_CREATE_DB_ON_PATH** **DB2_DDL_SOFT_INVAL** **DB2DEFPREP** **DB2_DISABLE_FLUSH_LOG** **DB2_DISPATCHER_PEEKTIMEOUT** **DB2_DJ_INI** **DB2DMNBCKCTLR** **DB2_DOCHOST** **DB2_DOCPORT** **DB2_ENABLE_AUTOCONFIG_DEFAULT** **DB2_ENABLE_LDAP** **DB2_EVMON_EVENT_LIST_SIZE** **DB2_EVMON_STMT_FILTER** **DB2_EXTSECURITY** **DB2_FALLBACK** **DB2_FMP_COMM_HEAPSZ** **DB2_GRP_LOOKUP** **DB2_HADR_BUF_SIZE** **DB2_HADR_NO_IP_CHECK** **DB2_HADR_PEER_WAIT_LIMIT** **DB2_HADR_SORCVBUF** **DB2_HADR_SOSNDBUF** **DB2LDAP_BASEDN** **DB2LDAPCACHE** **DB2LDAP_CLIENT_PROVIDER** **DB2LDAPHOST** **DB2LDAP_KEEP_CONNECTION** **DB2LDAP_SEARCH_SCOPE** **DB2_LOAD_COPY_NO_OVERRIDE** **DB2LOADREC** **DB2LOCK_TO_RB** **DB2_MAP_XML_AS_CLOB_FOR_DLC** **DB2_MAX_LOB_BLOCK_SIZE** **DB2_MEMORY_PROTECT** **DB2NOEXITLIST** **DB2_NUM_CKPW_DAEMONS** **DB2_OPTSTATS_LOG** **DB2REMOTEPREG** **DB2_RESOLVE_CALL_CONFLICT** **DB2ROUTINE_DEBUG** **DB2SATELLITEID** **DB2_SERVER_CONTIMEOUT** **DB2_SERVER_ENCALG** **DB2SORT** **DB2_TRUNCATE_REUSESTORAGE** **DB2_USE_DB2JCCT2_JROUTINE** **DB2_UTIL_MSGPATH** **DB2_VENDOR_INI** **DB2_XBSA_LIBRARY**

일반 레지스트리 변수

DB2ACCOUNT

- 운영 체제: 모두

- 디폴트: NULL
- 이 변수는 리모트 호스트에 보내는 어카운팅 문자열을 정의합니다. 자세한 내용은 DB2 Connect 사용자 안내서를 참조하십시오.

DB2BIDI

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- 이 변수는 양방향 지원을 사용하며 **DB2CODEPAGE** 변수는 사용할 코드 페이지를 선언하는 데 사용됩니다.

DB2_CAPTURE_LOCKTIMEOUT

- 운영 체제: 모두
- 디폴트: NULL, 값: ON 또는 NULL
- 이 변수는 잠금 시간종료가 발생할 때 이에 대한 설명적 정보를 로그하도록 지정합니다. 로그된 정보는 잠금 시간종료에 이르게 한 잠금 경합에 관련된 키 응용프로그램, 해당 응용프로그램이 잠금 시간종료 시 실행 중이던 작업에 대한 세부사항 및 경쟁의 원인이 되는 잠금에 대한 세부사항을 식별합니다. 잠금 요청자(잠금 시간종료 오류를 수신한 응용프로그램)와 현재 잠금 소유자 모두에 대한 정보가 캡처됩니다. 각 잠금 시간종료에 대해 텍스트 보고서가 작성되고 파일에 저장됩니다.

  파일은 다음과 같은 이름 지정 규칙을 사용하여 작성됩니다.
  db2locktimeout.*par*.*AGENTID*.*yyyy-mm-dd-hh-mm-ss*, 여기서 *par*은 데이터베이스 파티션 번호, *AGENTID*는 에이전트 ID, *yyyy-mm-dd-hh-mm-ss*는 년, 월, 일, 시, 분 및 초로 구성된 시간소인입니다. 파티션되지 않은 데이터베이스 환경에서 *par*은 0으로 설정됩니다.

  파일의 위치는 **diagpath** 데이터베이스 구성 매개변수에 설정된 값을 기본으로 합니다. **diagpath**가 설정되지 않은 경우, 파일은 다음 디렉토리 중 하나에 있습니다.

  – Windows 환경의 경우:
    - **DB2INSTPROF** 환경 변수가 설정되지 않은 경우, 정보는 *x*:₩*SQLLIB*₩*DB2INSTANCE*에 기록되며 여기서 *x*는 드라이브 참조, *SQLLIB*는 **DB2PATH** 레지스트리 변수에 지정한 디렉토리이며 *DB2INSTANCE*는 인스턴스 소유자의 이름입니다.
    - **DB2INSTPROF** 환경 변수를 설정한 경우, 정보는 *x*:₩*DB2INSTPROF*₩*DB2INSTANCE*에 기록되며 여기서 *x*는 드라이브 참조, *DB2INSTPROF*는 인스턴스 프로파일 디렉토리의 이름이며 *DB2INSTANCE*는 인스턴스 소유자의 이름입니다.

– Linux 및 UNIX 환경에서 정보는 *INSTHOME*/sqllib/db2dump에 기록되며 여기서 *INSTHOME*은 인스턴스의 홈 디렉토리입니다.

잠금 시간종료 보고서 파일이 더 이상 필요하지 않은 경우 삭제하십시오. 보고서 파일은 다른 진단 로그와 동일한 위치에 있으므로 디렉토리가 가득 차게 될 경우 DB2 시스템에서 종료할 수 있습니다. 일부 잠금 시간종료 보고서 파일을 보존해야 할 경우, 해당 파일을 DB2 로그가 저장된 위치와 다른 디렉토리 또는 폴더로 이동하십시오.

중요사항: 이 변수는 사용되지 않으며 CREATE EVENT MONITOR FOR LOCKING문을 사용하여 잠금 시간종료 이벤트를 수집하는 새 메소드가 있으므로 추후 릴리스에서는 제거됩니다.

DB2CODEPAGE

- 운영 체제: 모두
- 디폴트: 운영 체제에서 지정한 대로 언어 ID에서 파생됩니다.
- 이 변수는 데이터베이스 클라이언트 응용프로그램에 대해 DB2에 제공되는 데이터의 코드 페이지를 지정합니다. DB2 문서에 명시적으로 표시되었거나 DB2 서비스에서 요청한 경우를 제외하면 **DB2CODEPAGE**를 설정하지 않아야 합니다. **DB2CODEPAGE**를 운영 체제에서 지원되지 않는 값으로 설정하면 예기치 않은 결과가 발생할 수 있습니다. 일반적으로 DB2는 운영 체제에서 코드 페이지 정보를 자동으로 얻으므로 **DB2CODEPAGE**를 설정할 필요가 없습니다.

  주: Windows는 Windows 지역 설정에서 ANSII 코드 페이지 대신 유니코드 코드 페이지를 보고하지 않으므로 Windows 응용프로그램은 유니코드 클라이언트처럼 작동하지 않습니다. 이 동작을 겹쳐쓰려면 응용프로그램이 유니코드 응용프로그램처럼 작동하도록 **DB2CODEPAGE** 레지스트리 변수를 1208(유니코드 코드 페이지)로 설정하십시오.

DB2_COLLECT_TS_REC_INFO

- 운영 체제: 모두
- 디폴트: ON, 값: ON 또는 OFF
- 이 변수는 테이블 스페이스를 롤 포워드할 때 테이블 스페이스에 영향을 미치는 로그 레코드가 로그 파일에 포함되어 있는지 여부와 관계없이 DB2가 모든 로그 파일을 처리하는지 여부를 지정합니다. 테이블 스페이스에 영향을 미치는 로그 레코드를 포함하지 않는 것으로 알려진 로그 파일을 건너뛰려면 이 변수를 ON으로 설정하십시오. 로그 파일을 건너뛰는 데 필요한 정보를 수집하기 위해 로그 파일을 작성하고 사용하기 전에 **DB2_COLLECT_TS_REC_INFO**를 설정해야 합니다.

DB2_CONNRETRIES_INTERVAL

- 운영 체제: 모두
- 디폴트: 설정되지 않음, 값: 초 수(정수)
- 이 변수는 자동 클라이언트 리라우트 기능에 대한 연속 연결 재시도 수 간의 휴먼 시간(초)을 지정합니다. 이 변수를 **DB2_MAX_CLIENT CONNRETRIES**와 함께 사용하여 자동 클라이언트 리라우트에 대한 재시도 동작을 구성할 수 있습니다.

  DB2_MAX_CLIENT_CONNRETRIES는 설정되었지만 **DB2_CONNRETRIES_INTERVAL**은 설정되지 않은 경우, **DB2_CONNRETRIES_INTERVAL**은 디폴트로 30으로 설정됩니다. **DB2_MAX_CLIENT_CONNRETRIES**는 설정되지 않았지만 **DB2_CONNRETRIES_INTERVAL**은 설정된 경우, **DB2_MAX_CLIENT_CONNRETRIES**는 디폴트로 10으로 설정됩니다. **DB2_MAX_CLIENT_CONNRETRIES** 및 **DB2_CONNRETRIES_INTERVAL**이 모두 설정되지 않은 경우, 자동 클라이언트 리라우트 기능은 최대 10분 동안 반복적으로 데이터베이스에 대한 연결을 재시도하는 디폴트 동작으로 되돌아갑니다.

DB2CONSOLECP

- 운영 체제: Windows
- 디폴트: NULL, 값: 모든 유효한 코드 페이지 값
- DB2 메시지 텍스트 표시를 위한 코드 페이지를 지정합니다. 지정된 경우, 이 값은 운영 체제 코드 페이지 설정값을 겹쳐씁니다.

DB2COUNTRY

- 운영 체제: Windows
- 디폴트: NULL, 값: 숫자로 된 모든 유효한 나라, 지역 또는 영역 코드
- 이 변수는 클라이언트 응용프로그램의 나라, 지역 또는 영역 코드를 지정합니다. 지정된 경우, 이 값은 운영 체제 설정값을 겹쳐씁니다.

주: DB2COUNTRY는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 대신에 **DB2COUNTRY**와 같은 값을 승인하는 **DB2TERRITORY**를 사용하십시오.

DB2DBDFT

- 운영 체제: 모두
- 디폴트: NULL
- 이 변수는 내재된 연결에 사용되는 데이터베이스의 데이터베이스 별명 이름을 지정합니다. 응용프로그램에 데이터베이스 연결이 없지만 SQL 또는 XQuery문이 발행되는 경우, 디폴트 데이터베이스를 사용하여 **DB2DBDFT** 환경 변수가 정의되었으면 내재된 연결이 작성됩니다.

DB2DBMSADDR

- 운영 체제: Windows 32비트
- 디폴트: 0x20000000, 값: 0x20000000 - 0xB0000000(0x10000씩 증가)
- 이 변수는 디폴트 데이터베이스 관리 프로그램 공유 메모리 주소를 16진수 형식으로 지정합니다. 공유 메모리 주소 충돌로 인해 db2start가 실패하는 경우, 이 레지스트리 변수를 수정하여 데이터베이스 관리 프로그램 인스턴스가 공유 메모리를 다른 주소에 할당하도록 강제 실행할 수 있습니다.

DB2DISCOVERYTIME

- 운영 체제: Windows
- 디폴트: 40초, 최소값: 20초
- 이 변수는 SEARCH 발견이 DB2 시스템을 검색하는 시간 *크기*를 지정합니다.

DB2_EXPRESSION_RULES

- 운영 체제: 모두
- 디폴트: 비어 있음, 값: RAISE_ERROR_PERMIT_SKIP 또는 RAISE_ERROR_PERMIT_DROP
- **DB2_EXPRESSION_RULES** 레지스트리 변수의 설정값은 DB2 옵티마이저가 RAISE_ERROR 기능을 포함하는 쿼리에 대한 액세스 플랜을 판별하는 방법을 제어합니다. RAISE_ERROR 함수의 디폴트 동작은 이 함수를 포함하는 표현식의 범위 밖에서는 필터링이 수행되지 않습니다. 그 결과, 표현식의 초과 계산, 초과 잠금 및 낮은 쿼리 성능으로 이어질 수 있는 테이블 액세스 중에는 술어가 적용되지 않을 수 있습니다.

  응용프로그램의 특정 비즈니스 요구사항에 따라 이 동작이 과도하게 엄격한 특정한 경우에는 술어와 조인이 RAISE_ERROR 응용프로그램 앞에 적용되는지 여부가 중요하지 않을 수 있습니다. 예를 들어, 행 레벨 보안 구현의 컨텍스트에는 일반적으로 다음과 같은 양식의 표현식이 있습니다.

  ```
  CASE WHEN <conditions for validatin access to this row>
       THEN NULL
       ELSE RAISE_ERROR(...)
  END
  ```

  응용프로그램은 테이블의 모든 행에 대한 액세스의 유효성 확인이 아니라 쿼리에 의해 선택된 행에 대한 액세스의 유효성 확인에만 관여합니다. 따라서 술어는 기본 테이블 액세스에서 적용될 수 있으며 모든 필터링이 수행된 후에는 RAISE_ERROR를 포함하는 표현식만 실행되어야 합니다. 이 경우에는 **DB2_EXPRESSION_RULES**=RAISE_ERROR_PERMIT_SKIP 값이 적당합니다.

다른 대체 표현식은 COLUMN LEVEL 보안의 컨텍스트에 있습니다. 이 경우에는 일반적으로 다음과 같은 양식의 표현식이 있습니다.

```
CASE WHEN <conditions for validating access to this row and column>
     THEN <table.column>
     ELSE RAISE_ERROR(...)
END
```

이 경우, 사용자가 특정 행에 대한 데이터를 수신하려고 시도하고 컬럼에 사용자가 검색할 수 없는 값이 포함되어 있으면 응용프로그램에서는 오류만 발생할 수 있습니다. 이 경우,
DB2_EXPRESSION_RULES=RAISE_ERROR_PERMIT_DROP 설정은 특정 컬럼이 술어 또는 컬럼 함수에서 사용되거나 쿼리의 출력으로 리턴되면 RAISE_ERROR 함수를 포함하는 표현식만 평가하게 됩니다.

DB2FFDC

- 운영 체제: 모두
- 디폴트: ON, 값: ON, CORE:OFF
- 이 변수는 코어 파일 생성을 비활성화하는 기능을 제공합니다. 디폴트로 이 레지스트리 변수는 ON으로 설정됩니다. 이 레지스트리 변수가 설정되지 않거나 CORE:OFF 이외의 값으로 설정된 경우, DB2 서버가 이상 종료되면 코어 파일이 생성될 수 있습니다.

  문제점 판별에 사용되고 **diagpath** 디렉토리에 작성되는 코어 파일은 DB2 종료 프로세스의 전체 프로세스 이미지를 포함합니다. 코어 파일이 매우 클 수 있으므로 사용 가능한 파일 시스템 스페이스에 주의해야 합니다. *크기*는 문제점이 발생한 시간의 프로세스 상태 및 DB2 구성에 따라 다릅니다.

  Linux 운영 체제에서 디폴트 코어 파일 *크기* 한계는 `0`으로 설정됩니다 (즉, `ulimit -c`). 이 설정으로는 코어 파일이 생성되지 않습니다. 코어 파일을 Linux 운영 체제에서 작성할 수 있으려면 값을 무제한으로 설정하십시오.

주: DB2FFDC는 버전 9.5에서는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 새 레지스트리 변수 **DB2FODC**가 **DB2FFDC**의 기능을 통합합니다.

DB2FODC

- 운영 체제: 모두
- 디폴트: 모든 FODC 매개변수의 병합(아래 참조)
  - Linux 및 UNIX의 경우: "CORELIMIT=*val* DUMPCORE=ON DUMPDIR=*diagpath*"
  - Windows의 경우: "DUMPCORE=ON DUMPDIR=*diagpath*"

  매개변수는 공백으로 구분됩니다.

- 이 레지스트리 변수는 FODC(First Occurrence Data Collection)에 사용되는 문제점 해결 관련 매개변수 세트를 제어합니다. **DB2FODC**를 사용하여 사용 불능 상황에서 데이터 콜렉션의 여러 가지 측면을 제어할 수 있습니다.

  이 레지스트리 변수는 DB2 인스턴스 시작 중에 한 번 읽힙니다. FODC 매개변수를 온라인으로 갱신하려면 db2pdcfg 도구를 사용하십시오. **DB2FODC** 레지스트리 변수를 사용하여 재부트 시 구성을 유지할 수 있습니다. 매개변수를 모두 지정하거나 특정 순서로 지정할 필요가 없습니다. 지정되지 않은 모든 매개변수에는 디폴트값이 지정됩니다. 예를 들어, 코어 파일이 덤프되는 것은 원하지 않지만 다른 매개변수의 디폴트 동작은 원하는 경우에는 다음 명령을 발행하십시오.

  ```
  db2set DB2FODC="DUMPCORE=OFF"
  ```

  매개변수:

  CORELIMIT

  - 운영 체제: Linux 및 UNIX
  - 디폴트: Current® ulimit 설정, 값: 0 - 무제한
  - 이 옵션은 작성된 코어 파일의 최대 *크기*를 지정합니다(GB). 이 값은 현재 코어 파일 크기 한계 설정을 겹쳐씁니다. 코어 파일이 매우 클 수 있으므로 사용 가능한 파일 시스템 스페이스에 주의해야 합니다. *크기*는 문제점이 발생한 시간의 프로세스 상태 및 DB2 구성에 따라 다릅니다.

    CORELIMIT이 설정된 경우, DB2는 이 값을 사용하여 코어 파일을 생성하기 위한 현재 사용자 코어 한계(ulimit) 설정을 겹쳐씁니다.

    CORELIMIT이 설정되지 않은 경우, DB2는 코어 파일 *크기*를 현재 ulimit 설정과 같은 값으로 설정합니다. ″무제한″의 ulimit 설정이 DB2 서버 전용의 8GB 값을 겹쳐쓰는 AIX는 예외입니다. 8GB보다 큰 코어 덤프가 필요한 경우, ulimit를 RAM *크기*와 같이 적당히 큰 값으로 설정하거나 충분히 큰 값을 가진 **CORELIMIT**을 설정하십시오.

    주: 사용자 코어 한계 또는 **CORELIMIT**에 대한 변경사항은 다음에 DB2 인스턴스를 재사용할 때까지 적용되지 않습니다.

  DUMPCORE

  - 운영 체제: Linux, Solaris, AIX
  - 디폴트: AUTO, 값: AUTO, ON 또는 OFF

– 이 옵션은 코어 파일 생성이 발생하는지 여부를 지정합니다. 문제점 판별에 사용되고 **diagpath** 디렉토리에 작성되는 코어 파일은 DB2 종료 프로세스의 전체 프로세스 이미지를 포함합니다. 그러나 실제 코어 파일 덤프 발생 여부는 현재 ulimit 설정 및 **CORELIMIT** 매개변수의 값에 따라 다릅니다. 또한 일부 운영 체제에는 응용프로그램 코어 덤핑의 동작을 지시하는 코어 덤프에 대한 구성 설정이 있습니다. AUTO 설정은 **DB2RESILIENCE** 레지스트리 변수가 ON으로 설정되었을 때 트랩을 유지할 수 없는 경우에 코어 파일이 생성되도록 합니다. **DUMPCORE**=ON 설정은 항상 **DB2RESILIENCE** 레지스트리 변수 설정을 겹쳐써서 코어 파일을 생성합니다.

  코어 파일 덤프를 사용하지 않기 위한 권장 메소드는 **DUMPCORE**를 OFF로 설정하는 것입니다.

DUMPDIR

– 운영 체제: 모두

– 디폴트: **diagpath** 디렉토리 또는 **diagpath**가 정의되지 않은 경우 디폴트 진단 디렉토리, 값: *디렉토리 경로*

– 이 옵션은 코어 파일 작성을 위한 디렉토리의 절대 경로 이름을 지정합니다. 이 옵션은 코어 파일뿐만 아니라 FODC 패키지 밖에 저장해야 하는 기타 대형 실행 파일 덤프에 사용됩니다.

DB2_FORCE_APP_ON_MAX_LOG

- 운영 체제: 모두
- 디폴트: TRUE, 값: TRUE 또는 FALSE
- **max_log** 구성 매개변수 값이 초과될 때 발생하는 일을 지정합니다. TRUE로 설정되면 응용프로그램이 강제 실행되어 데이터베이스를 종료하고 작업 단위(UOW)가 롤백됩니다.

  FALSE로 설정되면 현재 명령문이 실패합니다. 응용프로그램은 작업 단위(UOW)의 이전 명령문에 의해 완료된 작업을 계속 커미트하거나 완료된 작업을 롤백하여 작업 단위를 실행 취소할 수 있습니다.

  주: 이 DB2 레지스트리 변수는 로그가 가득 찬 상황에서 복구하기 위한 임포트 유틸리티의 기능에 영향을 미칩니다. **DB2_FORCE_APP_ON_MAX_LOG**가 TRUE로 설정되고 **COMMITCOUNT** 명령 옵션과 함께 IMPORT 명령을 발행하는 경우, 임포트 유틸리티는 사용 중인 로그 스페이스 밖에서 실행되는 것을 방지하기

위해 커미트를 수행할 수 없습니다. 임포트 유틸리티가 SQL0964C(가득 찬 트랜잭션 로그)를 발견하면 유틸리티가 강제 실행되어 데이터베이스를 종료하고 현재 작업 단위는 롤백됩니다.

DB2GRAPHICUNICODESERVER

- 운영 체제: 모두
- 디폴트: OFF, 값: ON 또는 OFF
- 이 레지스트리 변수는 그래픽 데이터를 유니코드 데이터베이스에 삽입하기 위해 작성된 기존 응용프로그램을 수용하기 위해 사용됩니다. 이 변수는 클라이언트의 코드 페이지 대신 유니코드로 된 sqldbchar(그래픽) 데이터를 특정적으로 전송하는 응용프로그램에 대해서만 사용해야 합니다. (sqldbchar은 C 및 C++에서 단일 2바이트 문자를 가질 수 있는 지원되는 SQL 데이터 유형입니다.) ON으로 설정되면 데이터베이스에 그래픽 데이터가 유니코드로 생성되고 응용프로그램이 유니코드로 된 그래픽 데이터를 수신할 예정임을 알려줍니다.

DB2INCLUDE

- 운영 체제: 모두
- 디폴트: 현재 디렉토리
- DB2 PREP 처리 중에 SQL INCLUDE 텍스트 파일 명령문 처리 동안 사용되는 경로를 지정합니다. 이 변수는 INCLUDE 파일을 찾을 수 있는 디렉토리 목록을 제공합니다. 프리컴파일된 다른 언어로 **DB2INCLUDE**를 사용하는 방법에 대한 설명은 Developing Embedded SQL Applications의 내용을 참조하십시오.

DB2INSTDEF

- 운영 체제: Windows
- 디폴트: DB2
- 이 변수는 **DB2INSTANCE**가 정의되지 않은 경우에 사용되는 값을 설정합니다.

DB2INSTOWNER

- 운영 체제: Windows
- 디폴트: **NULL**
- 이 레지스트리 변수는 인스턴스가 먼저 작성된 경우에 DB2 프로파일 레지스트리에 작성됩니다. 이 변수는 인스턴스 소유 머신의 이름으로 설정됩니다.

DB2_LIC_STAT_SIZE

- 운영 체제: 모두
- 디폴트: NULL, 범위: 0 - 32,767

- 이 변수는 시스템에 대한 라이센스 통계를 포함하는 파일의 최대 *크기*(MB)를 판별합니다. 0 값은 라이센스 통계 수집을 끕니다. 변수가 인식되지 않거나 정의되지 않은 경우 디폴트로 무제한으로 설정됩니다. 통계는 라이센스 센터를 사용하여 표시됩니다.

DB2LOCALE

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- 이 변수는 DB2 호출 후 프로세스의 디폴트 ″C″ 로케일이 디폴트 ″C″ 로케일로 리스토어되는지 및 DB2 함수 호출 후 프로세스 로케일을 원래 'C'로 리스토어하는지 여부를 지정합니다. 원래 로케일이 'C'가 아닌 경우 이 레지스트리 변수는 무시됩니다.

DB2_MAX_CLIENT_CONNRETRIES

- 운영 체제: 모두
- 디폴트: 설정되지 않음, 값: 연결을 재시도할 최대 횟수(정수)
- 이 변수는 자동 클라이언트 리라우트 기능이 시도할 최대 연결 재시도 수를 지정합니다. 이 변수를 **DB2_CONNRETRIES_INTERVAL**과 함께 사용하여 자동 클라이언트 리라우트에 대한 재시도 동작을 구성할 수 있습니다.

  DB2_MAX_CLIENT_CONNRETRIES는 설정되었지만 **DB2_CONNRETRIES_INTERVAL**은 설정되지 않은 경우, **DB2_CONNRETRIES_INTERVAL**은 디폴트로 30으로 설정됩니다. **DB2_MAX_CLIENT_CONNRETRIES**는 설정되지 않았지만 **DB2_CONNRETRIES_INTERVAL**은 설정된 경우, **DB2_MAX_CLIENT_CONNRETRIES**는 디폴트로 10으로 설정됩니다. **DB2_MAX_CLIENT_CONNRETRIES** 및 **DB2_CONNRETRIES_INTERVAL**이 모두 설정되지 않은 경우, 자동 클라이언트 리라우트 기능은 최대 10분 동안 반복적으로 데이터베이스에 대한 연결을 재시도하는 디폴트 동작으로 되돌아갑니다.

DB2_OBJECT_TABLE_ENTRIES

- 운영 체제: 모두
- 디폴트: 0, 값: 0–65,532

  시스템에서 가능한 실제 최대값은 페이지 *크기* 및 Extent *크기*에 따라 다르지만 65,532를 초과할 수 없습니다.
- 이 변수는 테이블 스페이스의 예상 오브젝트 수를 지정합니다. 예를 들어, 1000개 이상의 많은 오브젝트가 DMS 테이블 스페이스에 작성될 것임을 아는 경우에는 테이블 스페이스를 작성하기 전에 이 레지스트리 변수를 근사치로 설정해야 합니다. 이는 테이블 스페이스 작성 동안 오브젝트 메타데이

터에 대해 인접하는 스토리지를 예약합니다. 인접하는 스토리지를 예약하면 온라인 백업이 메타데이터의 항목을 갱신하는 조작을 차단할 가능성을 낮춥니다(예: CREATE INDEX, IMPORT REPLACE). 또한 메타데이터가 테이블 스페이스의 시작 부분에 저장되므로 테이블 스페이스의 크기를 더 쉽게 조정할 수 있습니다.

테이블 스페이스의 초기 크기가 인접하는 스토리지를 예약할 만큼 크지 않은 경우, 추가 스페이스를 예약하지 않고 테이블 스페이스 작성이 진행됩니다.

DB2_SYSTEM_MONITOR_SETTINGS

- 운영 체제: 모두
- 이 레지스트리 변수는 DB2 모니터링의 다양한 측면의 동작을 수정할 수 있는 매개변수 세트를 제어합니다. 다음 예와 같이 각 매개변수를 세미콜론으로 구분하십시오.

  ```
  db2set DB2_SYSTEM_MONITOR_SETTINGS=OLD_CPU_USAGE:TRUE;
   DISABLE_CPU_USAGE:TRUE
  ```

  DB2_SYSTEM_MONITOR_SETTINGS를 설정할 때마다 각 매개변수가 명시적으로 설정됩니다. 이 변수를 설정할 때 지정하지 않은 매개변수는 디폴트값으로 되돌아갑니다. 따라서 다음 예에서는 아래 사항을 참조하십시오.

  ```
  db2set DB2_SYSTEM_MONITOR_SETTINGS=DISABLE_CPU_USAGE:TRUE
  ```

  주: 현재 이 레지스트리 변수에는 Linux에 대한 설정값만 있습니다. 기타 운영 체제에 대한 추가 설정은 추후 릴리스에서 추가됩니다.
  OLD_CPU_USAGE는 디폴트 설정으로 리스토어됩니다.

- 매개변수:

  OLD_CPU_USAGE

  - 운영 체제: Linux
  - 값: TRUE/ON, FALSE/OFF
  - RHEL4 및 SLES9의 경우 디폴트값:
    TRUE(메모: OLD_CPU_USAGE의 FALSE 설정값은 무시되며 이전 동작만 사용됩니다.)
  - RHEL5, SLES10 및 기타의 경우 디폴트값: FALSE
  - 이 매개변수는 인스턴스가 Linux 플랫폼에서 CPU 사용 시간을 확보하는 방법을 제어합니다. TRUE로 설정된 경우, CPU 사용 시간을 확보하는 이전 메소드가 사용됩니다. 이 메소드는 시스템 및 사용자 CPU 사용 시간을 모두 리턴하지만 이를 수행하는 데 더 많은 CPU를 사용합니다(즉, 오버헤드가 더 높음). FALSE로

설정된 경우, CPU 사용 시간을 확보하는 새 메소드가 사용됩니다. 이 메소드는 사용자 CPU 사용값만 리턴하지만 오버헤드가 적기 때문에 더 빠릅니다.

DISABLE_CPU_USAGE

- 운영 체제: Linux
- 값: TRUE/ON, FALSE/OFF
- RHEL4 및 SLES9의 경우 디폴트값: TRUE
- RHEL5, SLES10 및 기타의 경우 디폴트값: FALSE
- 이 매개변수를 사용하여 CPU 사용이 읽히는지 여부를 판별할 수 있습니다. DISABLE_CPU_USAGE가 사용 가능하면(TRUE로 설정), CPU 사용이 읽히지 않아서 때때로 CPU 사용 검색 중에 발생할 수 있는 오버헤드를 피할 수 있습니다.

DB2TERRITORY

- 운영 체제: 모두
- 디폴트: 운영 체제에서 지정한 대로 언어 ID에서 파생됩니다.
- 이 변수는 클라이언트 응용프로그램의 영역 또는 지역 코드를 지역하여 날짜 및 시간 형식에 영향을 미칩니다.

DB2_VIEW_REOPT_VALUES

- 운영 체제: 모두
- 디폴트: NO, 값: YES, NO
- 이 변수를 사용하면 다시 최적화된 SQL 또는 XQuery문이 설명될 때 모든 사용자가 EXPLAIN_PREDICATE 테이블에 해당 명령문의 캐시된 값을 저장할 수 있습니다. 이 변수가 NO로 설정되면 DBADM만 이러한 변수를 EXPLAIN_PREDICATE 테이블에 저장할 수 있습니다.

시스템 환경 변수

DB2_ALTERNATE_GROUP_LOOKUP

- 운영 체제: AIX
- 디폴트: NULL, 값: NULL 또는 GETGRSET
- 이 변수를 사용하여 DB2는 운영 체제에서 제공한 대체 소스에서 그룹 정보를 확보할 수 있습니다. AIX에서는 getgrset 함수가 사용됩니다. 이 함수는 LAM(Loadable Authentication Module)을 통해 로컬 파일 이외의 위치에서 그룹을 확보하는 기능을 제공합니다.

DB2_CLP_EDITOR

자세한 내용은 『명령행 변수』의 DB2_CLP_EDITOR를 참조하십시오.

DB2_CLP_HISTSIZE

자세한 내용은 『명령행 변수』의 DB2_CLP_HISTSIZE를 참조하십시오.

DB2CONNECT_ENABLE_EURO_CODEPAGE

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- 유로 지원이 필요한 IBM i 서버용 DB2 또는 z/OS 서버용 DB2에 연결되는 모든 DB2 Connect 클라이언트 및 서버에서는 이 변수를 YES로 설정하십시오. 이 변수를 YES로 설정하면 현재 응용프로그램 코드 페이지가 유로 기호에 대한 지원을 명시적으로 표시하는 동등한 코드화된 문자 세트 ID(CCSID)에 맵핑됩니다. 그 결과, DB2 Connect는 현재 응용프로그램 코드의 CCSID 수퍼 세트이고 또한 유로 기호를 지원하는 CCSID를 사용하여 IBM i 서버용 DB2 또는 z/OS 서버용 DB2에 연결됩니다. 예를 들어, 클라이언트가 CCSID 1252에 맵핑되는 코드 페이지를 사용 중인 경우, 클라이언트는 CCSID 5348를 사용하여 연결됩니다.

DB2CONNECT_IN_APP_PROCESS

- 운영 체제: 모두
- 디폴트: YES, 값: YES 또는 NO
- 이 변수를 NO로 설정하면 DB2 Enterprise Server Edition 머신의 로컬 DB2 Connect 클라이언트가 에이전트 내에서 강제 실행됩니다. 에이전트 내에서 실행하면 로컬 클라이언트를 모니터할 수 있고 SYSPLEX 지원을 사용할 수 있는 이점이 있습니다.

DB2_COPY_NAME

- 운영 체제: Windows
- 디폴트: 머신에 설치된 DB2의 디폴트 사본 이름. 값: 머신에 설치된 DB2의 사본 이름. 이름의 길이는 최대 128자입니다.
- **DB2_COPY_NAME** 변수는 현재 사용 중인 DB2 사본의 이름을 저장합니다. 여러 DB2 사본이 머신에 설치된 경우, **DB2_COPY_NAME**을 사용하여 다른 DB2 사본으로 전환할 수 있으며 현재 사용 중인 사본을 변경하려면 *INSTALLPATH*₩bin₩db2envar.bat 명령을 실행해야 합니다.

DB2DBMSADDR

- 운영 체제: x86의 Linux 및 zSeries의 Linux(31비트)
- 디폴트: NULL, 값: 0x09000000 - 0xB0000000 범위의 가상 주소(0x10000씩 증가)
- **DB2DBMSADDR** 레지스트리 변수는 16진수 형식의 디폴트 데이터베이스 공유 메모리 주소를 지정합니다.

주: 올바르지 않은 주소는 DB2 시스템에 대해 DB2 인스턴스 시작 기능 장애부터 데이터베이스에 연결하는 기능에 이르는 심각한 문제를 일으킬 수 있습니다. 올바르지 않은 주소는 이미 사용 중인 메모리의 영역과 충돌하거나 다른 용도로 사용되는 주소입니다. 이 문제점을 처리하려면 다음 명령을 사용하여 **DB2DBMSADDR** 레지스트리 변수를 NULL로 재설정하십시오.

```
db2set DB2DBMSADDR=
```

이 변수를 사용하여 DB2 프로세스의 어드레스 스페이스 레이아웃을 미세 조정할 수 있습니다. 이 변수는 인스턴스 공유 메모리의 위치를 가상 주소 0x10000000의 현재 위치에서 새 값으로 변경합니다.

DB2_DIAGPATH

- 운영 체제: 모두
- 디폴트: 디폴트값은 UNIX 및 Linux 운영 체제에서는 인스턴스 db2dump 디렉토리이며 Windows 운영 체제에서는 인스턴스 db2 디렉토리입니다.
- 이 매개변수는 ODBC 및 DB2 CLI 응용프로그램에만 적용됩니다.

  이 매개변수를 사용하여 DB2 진단 정보의 완전한 경로를 지정할 수 있습니다. 이 디렉토리에는 사용자의 플랫폼에 따라 덤프 파일, 트랩 파일, 오류 로그, 통지 파일 및 경보 로그 파일이 포함될 수 있습니다.

  이 환경 변수 설정은 해당 환경 범위의 ODBC 및 CLI 응용프로그램에 대해 DB2 데이터베이스 관리 프로그램 구성 매개변수 **diagpath** 설정 및 CLI/ODBC 구성 키워드 **DiagPath** 설정과 동일한 효과가 있습니다.

DB2DOMAINLIST

- 운영 체제: 모두
- 디폴트: NULL, 값: 쉼표로 구분된 Windows 도메인 이름 목록(『,』)
- 이 변수는 하나 이상의 Windows 도메인을 정의합니다. 서버에서 유지보수되는 이 목록은 요청 사용자 ID가 인증되는 도메인을 정의합니다. 이러한 도메인에 속하는 사용자에게만 연결 또는 접속 요청이 승인됩니다.

  이 변수는 CLIENT 인증이 데이터베이스 관리 프로그램 구성에 설정된 경우에만 유효합니다. 이 변수는 Windows 데스크탑에서의 단일 로그온이 Windows 도메인 환경에서 필수인 경우에 필요합니다.

  DB2 서버 버전 V7.1 이상에서는 순수 Windows 도메인 환경에서만 **DB2DOMAINLIST**를 지원합니다. 버전 8 FixPak 15 및 버전 9.1 FixPack 3부터 **DB2DOMAINLIST**는 클라이언트 또는 서버가 Windows 환경에서 실행 중인 경우에 지원됩니다.

DB2ENVLIST

- 운영 체제: UNIX
- 디폴트: NULL
- 이 변수는 스토어드 프로시저 또는 사용자 정의 함수(UDF)의 특정 변수 이름을 나열합니다. 디폴트로 db2start 명령은 **DB2** 또는 **db2**가 앞에 붙은 경우를 제외한 모든 사용자 환경 변수를 필터링합니다. 특정 환경 변수를 스토어드 프로시저 또는 사용자 정의 함수에 전달해야 하는 경우, **DB2ENVLIST** 환경 변수에 변수 이름을 나열할 수 있습니다. 하나 이상의 공백으로 각 변수 이름을 구분하십시오.

DB2INSTANCE

- 운영 체제: 모두
- 디폴트: Windows 32비트 운영 체제의 경우 **DB2INSTDEF**.
- 이 환경 변수는 디폴트로 사용 중인 인스턴스를 지정합니다. UNIX에서는 사용자가 **DB2INSTANCE**의 값을 지정해야 합니다.

DB2INSTPROF

- 운영 체제: Windows
- 디폴트: `Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥`*`Copy Name`*(Windows XP, Windows 2003), `ProgramData¥IBM¥DB2¥`*`Copy Name`*(Windows Vista)
- 이 환경 변수는 Windows 운영 체제에서 인스턴스 디렉토리의 위치를 지정합니다. 버전 9.5부터 인스턴스 디렉토리(및 기타 사용자 데이터 파일)는 `sqllib` 디렉토리 아래에 있을 수 없습니다.

DB2LDAPSecurityConfig

- 운영 체제: 모두
- 디폴트: NULL, 값: IBM LDAP 보안 플러그인 구성 파일의 유효한 이름 및 경로
- 이 변수는 IBM LDAP 보안 플러그인 구성 파일의 위치를 지정하는 데 사용됩니다. 변수가 설정되지 않은 경우, IBM LDAP 보안 플러그인 구성 파일은 `IBMLDAPSecurity.ini`로 이름 지정되고 다음 중 한 위치에 있습니다.
  - Linux 및 UNIX 운영 체제의 경우: *`INSTHOME`*`/sqllib/cfg/`
  - Windows 운영 체제의 경우: **`%DB2PATH%¥cfg¥`**

  Windows 운영 체제의 경우, DB2 서비스에서 이 변수를 선택하도록 하려면 이를 전역 시스템 환경에 설정해야 합니다.

DB2LIBPATH

- 운영 체제: UNIX
- 디폴트: NULL

- DB2는 고유의 공유 라이브러리 경로를 구성합니다. PATH를 엔진의 라이브러리 경로에 추가하려면 **DB2LIBPATH**를 설정해야 합니다(예를 들어, AIX에서는 사용자 정의 함수에 LIBPATH의 특정 항목이 필요함). **DB2LIBPATH**의 실제 값은 DB2 구성 공유 라이브러리 경로의 끝에 추가됩니다.

DB2LOGINRESTRICTIONS

- 운영 체제: AIX
- 디폴트: LOCAL, 값: LOCAL, REMOTE, SU, NONE
- 이 레지스트리 변수를 사용하여 AIX 운영 체제 API loginrestrictions()를 사용할 수 있습니다. 이 API는 사용자가 시스템에 액세스할 수 있는지 여부를 판별합니다. 이 API를 호출하여 DB2 데이터베이스 보안은 운영 체제에서 지정한 로그인 제한사항을 강제 실행할 수 있습니다. 이 레지스트리 변수를 사용할 때 이 API에 제출할 수 있는 여러 가지 값이 있습니다. 값은 다음과 같습니다.
  - REMOTE

    DB2가 *rlogind* 또는 *telnetd* 프로그램을 통해 어카운트를 리모트 로그인에 사용할 수 있는지 검증하는 로그인 제한사항만 강제 실행합니다.
  - SU

    DB2 버전 9.1이 su 명령이 허용되고 현재 프로세스에 su 명령을 호출하여 어카운트로 전환할 수 있는 그룹 ID가 있는지 검증하는 su 제한사항만 강제 실행합니다.
  - NONE

    DB2가 로그인 제한사항을 강제 실행하지 않습니다.
  - LOCAL(또는 변수가 설정되지 않음)

    DB2가 이 어카운트에 대해 로컬 로그인이 허용되었는지 검증하는 로그인 제한사항만 강제 실행합니다. 이는 로그인할 때 일반 동작입니다.

  어떠한 옵션을 설정했든 지정된 특권을 가지고 있는 사용자 어카운트 또는 ID는 서버에서 로컬로 또는 리모트 클라이언트에서 DB2를 사용할 수 있습니다. loginrestrictions() API에 대한 설명은 AIX 문서를 참조하십시오.

DB2NODE

- 운영 체제: 모두
- 디폴트: NULL, 값: 1 - 999
- 접속 또는 연결하려는 데이터베이스 파티션 서버의 목표 논리 노드를 지정하는 데 사용됩니다. 이 변수가 설정되지 않은 경우, 목표 논리 노드는 디폴트

로 머신에서 0 포트로 정의된 논리 노드로 설정됩니다. 파티션된 데이터베이스 환경에서 연결 설정은 트러스트된 연결 획득에 영향을 미칠 수 있습니다. 예를 들어, **DB2NODE** 변수가 노드에서 연결을 설정하려면 중간 노드(홉 노드)를 통과해야 하는 노드로 설정된 경우, 이 연결이 트러스트된 연결로 표시될 수 있는지 여부를 판별하기 위해 해당 연결을 평가할 때 고려하는 사항은 해당 중간 노드의 IP 주소 및 홉 노드와 연결 노드 간의 통신에 사용되는 통신 프로토콜입니다. 즉, 연결이 시작된 원래 노드는 고려되지 않습니다. 대신에 홉 노드가 고려됩니다.

DB2OPTIONS

- 운영 체제: 모두
- 디폴트: NULL
- 명령행 처리기 옵션을 설정하는 데 사용됩니다.

DB2_PARALLEL_IO

- 운영 체제: 모두
- 디폴트: NULL, 값: *TablespaceID*:[*n*],... – 쉼표로 구분된 정의된 테이블 스페이스 목록(숫자 테이블 스페이스 ID로 식별됨). 테이블 스페이스의 프리페치 *크기*가 AUTOMATIC인 경우, 테이블 스페이스 ID, 콜론, 컨테이너당 디스크 수 *n*을 차례로 지정하여 DB2 데이터베이스 관리 프로그램에 컨테이너당 디스크 수를 표시할 수 있습니다. *n*이 지정되지 않은 경우, 디폴트는 6입니다.

  *TablespaceID*를 별표(*)로 바꿔 모든 테이블 스페이스를 지정할 수 있습니다. 예를 들어, **DB2_PARALLEL_IO** =*인 경우, 모든 테이블 스페이스에서 6을 컨테이너당 디스크 수로 사용합니다. 별표(*)와 테이블 스페이스 ID를 모두 지정하는 경우, 테이블 스페이스 ID 설정이 우선합니다. 예를 들어, **DB2_PARALLEL_IO** =*,1:3인 경우, 컨테이너당 디스크 수로 3을 사용하는 테이블 스페이스 1을 제외하고 모든 테이블 스페이스에서 6을 컨테이너당 디스크 수로 사용합니다.
- 이 레지스트리 변수는 DB2가 테이블 스페이스의 입출력 병렬 처리를 계산하는 방식을 변경하는 데 사용됩니다. 입출력 병렬 처리가 사용 가능한 경우(다중 컨테이너 사용으로 내재적으로 또는 **DB2_PARALLEL_IO** 설정으로 명시적으로), 올바른 프리페치 요청 수를 발행하여 수행됩니다. 각 프리페치 요청은 페이지 Extent에 대한 요청입니다. 예를 들어, 테이블 스페이스에 두 개의 컨테이너가 있고 프리페치 *크기*는 Extent *크기*의 네 배입니다. 레지스트리 변수가 설정된 경우, 이 테이블 스페이스에 대한 프리페치 요청은 네 개의 요청(요청당 하나의 Extent)으로 구분되며 네 개의 프리페처가 병렬로 요청을 서비스할 수 있습니다.

테이블 스페이스의 개별 컨테이너가 여러 개의 실제 디스크에 걸쳐서 스트라이프되거나 테이블 스페이스의 컨테이너가 둘 이상의 실제 디스크로 구성된 단일 RAID 디바이스에서 작성되는 경우에 이 레지스트리 변수를 설정하려고 하게 됩니다.

이 레지스트리 변수가 설정되지 않은 경우, 테이블 스페이스의 병렬 처리 수준은 테이블 스페이스의 컨테이너 수입니다. 예를 들어, **DB2_PARALLEL_IO**가 NULL로 설정되고 테이블 스페이스에 네 개의 컨테이너가 있는 경우, 네 개의 Extent *크기* 프리페치 요청이 발행됩니다. 또는 테이블 스페이스에 두 개의 컨테이너가 있고 프리페치 *크기*가 Extent 크기의 네 배인 경우, 이 테이블 스페이스에 대한 프리페치 요청은 두 개의 요청으로 구분됩니다(각 요청은 두 Extent에 대한 것임).

이 레지스트리 변수가 설정되고 테이블의 프리페치 *크기*가 AUTOMATIC이 아닌 경우, 테이블 스페이스의 병렬 처리 수준은 Extent *크기*로 나눈 프리페치 *크기*입니다. 예를 들어, **DB2_PARALLEL_IO**가 프리페치 *크기*가 160이고 Extent *크기*가 32페이지인 테이블 스페이스에 대해 설정된 경우, 다섯 개의 Extent *크기* 프리페치 요청이 발행됩니다.

이 레지스트리 변수가 설정되고 테이블 스페이스의 프리페치 *크기*가 AUTOMATIC인 경우, DB2는 다음의 등식을 사용하여 테이블 스페이스의 프리페치 *크기*를 자동으로 계산합니다.

```
프리페치 크기 =
 (컨테이너 수) * (컨테이너당 디스크 수)
  * Extent 크기
```

콜론 뒤의 숫자는 DB2가 등식의 *컨테이너당 디스크 수*에 사용합니다. 별표만 사용되고 숫자는 지정되지 않은 경우, 디폴트인 컨테이너당 6개의 디스크가 사용됩니다.

다음 표에서는 사용 가능한 여러 가지 옵션 및 각 상황에서 병렬 처리가 계산되는 방법에 대해 설명합니다.

표 64. 병렬 처리 계산법

테이블 스페이스의 프리페치 *크기*	**DB2_PARALLEL_IO 설정**	**병렬 처리 방법**
AUTOMATIC(프리페치 *크기* = 컨테이너 수 * 1 * Extent *크기*)	설정되지 않음	컨테이너 수
AUTOMATIC(프리페치 *크기* = 컨테이너 수 * 6 * Extent *크기*)	*테이블 스페이스 ID*	컨테이너 수 * 6
AUTOMATIC(프리페치 *크기* = 컨테이너 수 * *n* * Extent *크기*)	*테이블 스페이스 ID*:*n*	컨테이너 수 * *n*
자동 아님	설정되지 않음	컨테이너 수
자동 아님	*테이블 스페이스 ID*	프리페치 *크기*/Extent 크기

표 64. 병렬 처리 계산법 (계속)

테이블 스페이스의 프리페치 크기	DB2_PARALLEL_IO 설정	병렬 처리 방법
자동 아님	*테이블 스페이스 ID*:*n*	프리페치 크기/Extent 크기

예를 들어, 세 개의 테이블 스페이스가 있고 ID가 각각 3, 4 및 5인 경우를 고려하십시오. Extent 크기는 모두 4,096바이트이고 모두 각각 두 개의 컨테이너를 가지고 있습니다. 테이블 스페이스 3 및 4의 프리페치 크기는 모두 AUTOMATIC이고 테이블 스페이스 5의 프리페치 크기는 16,384바이트입니다. **DB2_PARALLEL_IO**=*:5,4:10으로 설정하면 다음과 같은 테이블 스페이스의 병렬 처리를 얻습니다.

- 테이블 스페이스 3: *n*(컨테이너당 디스크 수) 값은 5, Extent 크기는 4,096, 컨테이너 수는 2이고 프리페치 크기는 AUTOMATIC입니다. 따라서 프리페치 크기는 2*5*4,096이고 병렬 처리=컨테이너 수 * *n*=2*5=10입니다.
- 테이블 스페이스 4: 이 테이블 스페이스에 대해 *n*(컨테이너당 디스크 수) 값은 특정적으로 10으로 설정됩니다. Extent 크기는 4096, 컨테이너 수는 2, *n*은 10이고 프리페치 크기는 AUTOMATIC입니다. 따라서 프리페치 크기는 2*10*4,096이고 병렬 처리=컨테이너 수 * *n*=2*10=20입니다.
- 테이블 스페이스 5: *n* 값은 여전히 5이지만 프리페치 크기가 AUTOMATIC이 아니므로 적용되지 않습니다. Extent 크기는 4096, 컨테이너 수는 2, 프리페치 크기는 16384입니다. 따라서 병렬 처리=프리페치 크기/Extent 크기=16,384/4,096=4입니다.

일부 시나리오에서는 이 변수를 사용하면 그 결과 디스크 장치 경쟁이 발생합니다. 예를 들어, 테이블 스페이스에 두 개의 컨테이너가 있고 두 컨테이너에 각각 전용 단일 디스크가 있는 경우, 레지스트리 변수를 설정하면 그 결과 두 개의 프리페처가 동시에 두 디스크 각각에 액세스하게 되므로 해당 디스크에서 경쟁이 발생합니다. 그러나 각 컨테이너가 여러 디스크에서 스트라이프된 경우에 레지스트리 변수를 설정하면 동시에 네 개의 다른 디스크에 액세스할 가능성이 있습니다.

이 레지스트리 변수에 대한 변경사항을 활성화하려면 db2stop 명령을 발행한 다음 db2start 명령을 입력하십시오.

DB2PATH

- 운영 체제: Windows
- 디폴트: 운영 체제에 따라 다름

- 이 환경 변수는 제품이 Windows 32비트 운영 체제에서 설치되는 디렉토리를 지정하는 데 사용됩니다.

DB2_PMAP_COMPATIBILITY

- 운영 체제: 모두
- 디폴트: ON, 값: ON 또는 OFF
- 사용자는 이 변수를 사용하면 sqlugtpi 및 sqlugrpn API를 계속 사용하여 테이블에 대한 분산 정보와 행에 대한 데이터베이스 파티션 번호 및 데이터베이스 파티션 서버 번호를 각각 리턴할 수 있습니다. 디폴트 설정(ON)은 분산 맵 *크기*가 4,096개 항목을 유지함을 표시합니다(버전 9.7 이전 동작). 이 변수가 OFF로 설정된 경우, 새 데이터베이스 또는 업그레이된 데이터베이스의 분산 맵 *크기*는 32,768개 항목으로 늘어납니다(버전 9.7 동작). 32K 분산 맵을 사용하는 경우, 새 db2GetDistMap 및 db2GetRowPartNum API를 사용해야 합니다.

DB2PROCESSORS

- 운영 체제: Windows
- 디폴트: NULL, 값: 0–*n*-1(여기서 *n*=프로세서 수)
- 이 변수는 특정 db2syscs 프로세스에 대한 프로세스 친화도 마스크를 설정합니다. 여러 개의 논리 노드가 실행되는 환경에서 이 변수는 논리 노드를 프로세서 또는 프로세서 세트에 연관시키는 데 사용됩니다.

  이 변수가 지정되면 DB2가 SetProcessAffinityMask() API를 발행합니다. 이 변수가 지정되지 않으면 db2syscs 프로세스가 서버의 모든 프로세서와 연관됩니다.

DB2RCMD_LEGACY_MODE

- 운영 체제: Windows,
- 디폴트: NULL, 값: YES, ON, TRUE 또는 1 또는 NO, OFF, FALSE 또는 0
- 이 변수를 사용하여 사용자는 DB2 리모트 명령 서비스의 확장된 보안을 사용하거나 사용하지 않을 수 있습니다. 보안 방식으로 DB2 리모트 명령 서비스를 실행하려면 **DB2RCMD_LEGACY_MODE**를 NO, OFF, FALSE, 0 또는 NULL로 설정하십시오. Legacy 모드(확장된 보안을 사용하지 않음)에서 실행하려면 **DB2RCMD_LEGACY_MODE**를 YES, ON, TRUE 또는 1로 설정하십시오. 보안 모드는 도메인 제어기가 Windows 2000 이상을 실행 중인 경우에만 사용 가능합니다.

  주: DB2RCMD_LEGACY_MODE가 YES, ON, TRUE 또는 1로 설정된 경우, DB2 리모트 명령 서비스로 전송된 모든 요청은 요청자의 컨텍스트 아래에서 처리됩니다. 이를 쉽게 하려면 도메인 제어기에서 머신 또는 서비

스 로그온 어카운트를 사용 가능하게 하여 머신 또는 서비스 로그온 어카운트 또는 모두 클라이언트를 가장할 수 있도록 허용해야 합니다.

주: DB2RCMD_LEGACY_MODE가 NO, OFF, FALSE 또는 0으로 설정된 경우, DB2 리모트 명령 서비스 실행 명령을 가지려면 SYSADM 권한이 있어야 합니다.

DB2RESILIENCE

- 운영 체제: 모두
- 디폴트: ON, 값: ON(TRUE 또는 1) 또는 OFF(FALSE 또는 0)
- 이 레지스트리 변수를 사용하여 실제 읽기 오류가 허용되고 확장된 트랩 복구를 활성화하는지 여부를 제어할 수 있습니다. 디폴트 동작은 읽기 오류를 허용하고 확장된 트랩 복구를 활성하는 것입니다. 이전 릴리스의 동작으로 되돌리고 데이터베이스 관리 프로그램이 인스턴스를 종료하도록 강제 실행하려면 이 레지스트리 변수를 OFF로 설정하십시오. 이 레지스트리 변수는 기존 스토리지 키 지원에 영향을 미치지 않습니다.

DB2SYSTEM

- 운영 체제: Windows 및 UNIX
- 디폴트: NULL
- 사용자 및 데이터베이스 관리자가 DB2 서버 시스템을 식별하는 데 사용할 이름을 지정합니다. 가능한 경우, 이 이름은 네트워크 내에서 고유해야 합니다.

  이 이름은 제어 센터 오브젝트 트리의 시스템 레벨에 표시되어 관리자가 제어 센터에서 관리할 수 있는 서버 시스템을 식별하도록 도와줍니다.

  구성 지원 프로그램의 네트워크 검색 기능을 사용하는 경우, DB2 발견은 이 이름을 리턴하며 결과 오브젝트 트리의 시스템 레벨에 이 이름이 표시됩니다. 이 이름은 사용자가 액세스하려는 데이터베이스가 있는 시스템을 식별하는 데 도움이 됩니다. **DB2SYSTEM**의 값은 설치 시 다음과 같이 설정됩니다.

  – Windows의 경우 설치 프로그램은 이 값을 Windows 시스템에 지정된 컴퓨터 이름과 동일하게 설정합니다.
  – UNIX 시스템의 경우 이 값은 UNIX 시스템의 TCP/IP 호스트 이름과 동일하게 설정됩니다.

DB2_UPDDBCFG_SINGLE_DBPARTITION

- 운영 체제: 모두
- 디폴트: 설정되지 않음, 값: 0/FALSE/NO, 1/TRUE/YES

- 1, TRUE 또는 YES로 설정되면 이 레지스트리 변수를 사용하여 데이터베이스에 대한 갱신사항 및 재설정이 특정 파티션에만 영향을 주도록 지정할 수 있습니다. 이 변수가 설정되지 않으면 갱신사항 및 요청이 버전 9.5 동작을 따릅니다.
- 버전 9.5부터 파티션 절을 지정하지 않으면 데이터베이스 구성에 대한 갱신사항 또는 변경사항이 모든 데이터베이스 파티션에 걸쳐서 작동합니다. **DB2_UPDDBCFG_SINGLE_DBPARTITION**을 사용하여 데이터베이스 구성에 대한 갱신사항이 **DB2NODE** 레지스트리 변수로 설정한 데이터베이스 파티션 또는 로컬 데이터베이스 파티션에만 적용되는 DB2 이전 버전의 동작으로 되돌릴 수 있습니다. 이는 이 동작을 요구하는 기존 명령 스크립트 또는 응용프로그램에 대한 이전 버전과의 호환을 지원합니다.

주: 이 변수는 ADMIN_CMD 루틴 호출로 작성되는 갱신 또는 재설정 요청에는 적용되지 않습니다.

DB2_USE_PAGE_CONTAINER_TAG

- 운영 체제: 모두
- 디폴트:NULL, 값: ON, NULL
- 디폴트로 DB2는 DMS 컨테이너가 파일이든 디바이스든 각 컨테이너의 첫 번째 Extent에 컨테이너 태그를 저장합니다. 컨테이너 태그는 컨테이너의 메타데이터입니다. DB2 버전 8.1 이전에는 컨테이너 태그가 단일 페이지에 저장되었고 따라서 컨테이너의 스페이스가 적게 필요했습니다. 컨테이너 태그를 계속 단일 페이지에 저장하려면 **DB2_USE_PAGE_CONTAINER_TAG**를 ON으로 설정하십시오.

  그러나 컨테이너에 RAID 디바이스를 사용할 때 이 레지스트리 변수를 ON으로 설정하면 입출력 성능이 떨어집니다. RAID 디바이스의 경우 Extent 크기가 RAID 스트라이프 크기와 같거나 이의 배수인 테이블 스페이스를 작성하므로 **DB2_USE_PAGE_CONTAINER_TAG**를 ON으로 설정하면 Extent가 RAID 스트라이프와 정렬하지 않게 됩니다. 그 결과, 입출력 요청이 최적의 경우보다 더 많은 실제 디스크에 액세스해야 합니다. 매우 엄격한 스페이스 제한조건이 있거나 예비 버전 8 데이터베이스와 일치하는 동작이 필요한 경우 외에는 이 레지스트리 변수를 사용하지 않는 것이 좋습니다.

  이 레지스트리 변수에 대한 변경사항을 활성화하려면 db2stop 명령을 발행한 다음 db2start 명령을 입력하십시오.

DB2_WORKLOAD

- 운영 체제: 모두
- 디폴트: 설정되지 않음, 값: 1C, CM, SAP, TPM, WC

- **DB2_WORKLOAD**의 각 값은 사전 정의된 설정이 있는 여러 가지 레지스트리 변수의 특정 그룹을 나타냅니다.
- 유효한 값은 다음과 같습니다.

  1C 1C 응용프로그램에 대한 레지스트리 변수 세트를 데이터베이스에 구성하려면 이 설정을 사용하십시오.

  CM IBM Content Manager에 대한 레지스트리 변수 세트를 데이터베이스에 구성하려면 이 설정을 사용하십시오.

  SAP SAP 환경에 대한 레지스트리 변수 세트를 데이터베이스에 구성하려면 이 설정을 사용하십시오.

  DB2_WORKLOAD=SAP를 설정하면 사용자 테이블 스페이스 SYSTOOLSPACE 및 사용자 임시 테이블 스페이스 SYSTOOLSTMPSPACE는 자동으로 작성되지 않습니다. 이러한 테이블 스페이스는 다음의 마법사, 유틸리티 또는 함수에 의해 자동으로 작성되는 테이블에 사용됩니다.

  – 자동 유지보수
  – 디자인 어드바이저
  – 제어 센터 데이터베이스 정보 패널
  – 테이블 스페이스 입력 매개변수가 지정되지 않은 경우 SYSINSTALLOBJECTS 스토어드 프로시저
  – GET_DBSIZE_INFO 스토어드 프로시저

  SYSTOOLSPACE 및 SYSTOOLSTMPSPACE 테이블 스페이스가 없으면 이러한 마법사, 유틸리티 또는 함수를 사용할 수 없습니다.

  이러한 마법사, 유틸리티 또는 함수를 사용할 수 있으려면 다음 중 하나를 수행하십시오.

  – 도구에 필요한 오브젝트를 보유할 SYSTOOLSPACE 테이블 스페이스를 수동으로 작성하십시오(파티션된 데이터베이스 환경에서는 이 테이블 스페이스를 카탈로그 파티션에 작성하십시오). 예를 들어, 다음과 같습니다.

    ```
    CREATE REGULAR TABLESPACE SYSTOOLSPACE
    IN IBMCATGROUP
    MANAGED BY SYSTEM
    USING ('SYSTOOLSPACE')
    ```

  – 유효한 테이블 스페이스를 지정하고 SYSINSTALLOBJECTS 스토어드 프로시저를 호출하여 도구에 대한 오브젝트를 작성한 다음 특정 도구에 대한 ID를 지정하십시오.
    SYSINSTALLOBJECTS는 사용자를 위해 테이블 스페이스를 작

성합니다. 오브젝트에 대해 SYSTOOLSSPACE를 사용하지 않으려면 다른 사용자 정의 테이블 스페이스를 지정하십시오.

해당 선택사항 중 하나 이상을 완료한 후 SYSTOOLSTMPSPACE 임시 테이블 스페이스를 작성하십시오(파티션된 데이터베이스 환경에서 작업 중인 경우 역시 카탈로그 파티션에 작성). 예를 들어, 다음과 같습니다.

```
CREATE USER TEMPORARY TABLESPACE SYSTOOLSTMPSPACE
IN IBMCATGROUP
MANAGED BY SYSTEM
USING ('SYSTOOLSTMPSPACE')
```

테이블 스페이스 SYSTOOLSPACE 및 임시 테이블 스페이스 SYSTOOLSTMPSPACE가 작성되면 앞에서 설명한 마법사, 유틸리티 또는 함수를 사용할 수 있습니다.

TPM Tivoli Provisioning Manager에 대한 레지스트리 변수 세트를 데이터베이스에 구성하려면 이 설정을 사용하십시오.

WC Websphere Commerce에 대한 레지스트리 변수 세트를 데이터베이스에 구성하려면 이 설정을 사용하십시오.

통신 변수

DB2CHECKCLIENTINTERVAL

- 운영 체제: 모두, 서버 전용
- 디폴트=50, 값: 0 이상인 숫자 값
- 이 변수는 TCP/IP 클라이언트 연결 검증 빈도를 지정합니다. 이 변수를 사용하면 쿼리가 완료될 때까지 대기하는 대신에 클라이언트 종료를 일찍 발견할 수 있습니다. 이 변수가 0으로 설정되는 경우, 검증이 수행되지 않습니다.

  값이 낮으면 더 자주 점검합니다. 지침에 따라 낮은 빈도에는 100을 사용하고 중간 빈도에는 50, 높은 빈도에는 10을 사용하십시오. 값은 내부 DB2 메트릭으로 측정됩니다. 값은 선형 스케일을 나타냅니다. 즉, 50에서 100으로 값이 증가하면 간격이 두 배로 늘어납니다. 데이터베이스 요청을 실행하는 동안 클라이언트 상태를 자주 점검하면 쿼리를 완료하는 데 걸리는 시간이 길어집니다. DB2 워크로드가 과중한 경우(즉, 여러 가지 내부 요청이 포함된 경우), **DB2CHECKCLIENTINTERVAL**을 낮은 값으로 설정하면 워크로드가 적은 상황에서보다 성능에 더 큰 영향을 미칩니다.

  DB2® Universal Database™ 버전 8.1.4 이후
  DB2CHECKCLIENTINTERVAL의 디폴트값은 50입니다. 버전 8.1.4 이전에 디폴트값은 0이었습니다.

DB2COMM

- 운영 체제: 모두, 서버 전용
- 디폴트=NULL, 값: NPIPE, TCPIP, SSL
- 이 변수는 데이터베이스 관리 프로그램이 시작될 때 시작되는 통신 관리 프로그램을 지정합니다. 이 변수가 설정되지 않은 경우, DB2 통신 관리 프로그램이 서버에서 시작되지 않습니다.

DB2FCMCOMM

- 운영 체제: 지원되는 모든 DB2 Enterprise Server Edition 플랫폼
- 디폴트=TCPIP4, 값: TCPIP4 또는 TCPIP6
- 이 변수는 db2nodes.cfg 파일의 호스트 이름이 분석되는 방법을 지정합니다. 모든 호스트 이름은 IPv4 또는 IPv6로 분석됩니다. 호스트 이름 대신에 IP 주소가 db2nodes.cfg에 지정된 경우, IP 형식으로 IPv4가 사용되었는지 또는 IPv6가 사용되었는지를 판별합니다. **DB2FCMCOMM**이 설정되지 않은 경우, 디폴트 설정인 IPv4는 IPv4 호스트만 시작할 수 있음을 의미합니다.

  주: db2nodes.cfg에 지정된 호스트 이름에서 분석된 IP 형식 또는 db2nodes.cfg에 직접 지정된 IP 형식이 **DB2FCMCOMM**의 설정과 일치하지 않는 경우, db2start는 실패합니다.

DB2_FORCE_NLS_CACHE

- 운영 체제: AIX, HP_UX, Solaris
- 디폴트=FALSE, 값: TRUE 또는 FALSE
- 이 변수는 다중 스레드 응용프로그램에서 잠금 경합 가능성을 제거하는 데 사용됩니다. 이 레지스트리 변수가 TRUE인 경우, 코드 페이지 및 지역 코드 정보는 스레드가 해당 정보에 처음 액세스할 때 저장됩니다. 이후 캐시된 정보는 해당 정보를 요청하는 다른 모든 스레드에 사용됩니다. 이로 인해 잠금 경합이 제거되고 그 결과 특정 상황에서 성능이 높아집니다. 응용프로그램이 연결 간 로케일 설정을 변경하는 경우에는 이 설정을 사용하면 안 됩니다. 로케일 설정을 변경하는 것이 *스레드 안전*하지 않기 때문에 다중 스레드 응용프로그램은 일반적으로 로케일 설정을 변경하지 않으므로 이러한 상황에서는 이 설정이 필요하지 않을 수 있습니다.

DB2RSHCMD

- 운영 체제: UNIX
- 디폴트=rsh(HP-UX의 경우 remsh), 값은 rsh, remsh 또는 ssh의 전체 경로 이름임
- 디폴트로 DB2 데이터베이스 시스템은 리모트 데이터베이스 파티션을 시작하고 db2_all 스크립트를 사용하여 모든 데이터베이스 파티션에서 유틸리티

및 명령을 실행할 때 통신 프로토콜로 rsh를 사용합니다. 예를 들어, 이 레지스트리 변수를 ssh의 전체 경로 이름으로 설정하면 DB2 데이터베이스 제품이 ssh를 요청된 유틸리티 및 명령 실행을 위한 통신 프로토콜로 사용하게 됩니다. 이 레지스트리 변수를 해당 디폴트 매개변수를 포함한 리모트 명령 프로그램을 호출하는 스크립트의 전체 이름 경로로 설정할 수도 있습니다. 이 변수는 DB2 제품이 설치된 서버와 다른 서버에서 db2start 명령이 실행되는 단일 파티션 환경 또는 파티션된 데이터베이스에만 필요합니다. 인스턴스 소유자는 추가 검증 또는 인증(즉, 암호 또는 암호 구문)을 프롬프트하지 않고 각 DB2 데이터베이스 노드에서 다른 각 DB2 데이터베이스 노드에 로그인하기 위해 지정된 리모트 쉘 프로그램을 사용할 수 있어야 합니다.

DB2에 ssh 쉘을 사용하기 위해 DB2RSHCMD 레지스트리 변수 설정에 대한 자세한 지시사항은 백서 “Configure DB2 Universal Database for UNIX to use OpenSSH″를 참조하십시오.

DB2RSHTIMEOUT

- 운영 체제: UNIX
- 디폴트=30초, 값: 1 - 120
- 이 변수는 **DB2RSHCMD**가 널(NULL)이 아닌 값으로 설정된 경우에만 적용 가능합니다. 이 레지스트리 변수는 DB2 데이터베이스 시스템이 리모트 명령을 대기하는 시간종료 기간을 제어하는 데 사용됩니다. 이 시간종료 기간 후 응답이 수신되지 않으면 리모트 데이터베이스 파티션에 접근할 수 없으며 조작이 실패했다고 가정합니다.

  주: 지정된 시간 값은 리모트 명령을 실행하는 데 필요한 시간이 아니라 요청을 인증하는 데 필요한 시간입니다.

DB2SORCVBUF

- 운영 체제: 모두
- 디폴트=65,536
- TCP/IP 수신 버퍼 값을 지정합니다.

DB2SOSNDBUF

- 운영 체제: 모두
- 디폴트=65,536
- TCP/IP 송신 버퍼 값을 지정합니다.

DB2TCP_CLIENT_CONTIMEOUT

- 운영 체제: 모두, 클라이언트 전용
- 디폴트=0(시간종료 없음), 값: 0 - 32,767초

- **DB2TCP_CLIENT_CONTIMEOUT** 레지스트리 변수는 TCP/IP 연결 조작 시 클라이언트가 완료를 대기하는 초 수를 지정합니다. 지정된 초 내에 연결이 설정되지 않는 경우, DB2 데이터베이스 관리 프로그램은 -30081 selectForConnectTimeout 오류를 리턴합니다.

  레지스트리 변수가 설정되지 않거나 0으로 설정된 경우 시간종료는 없습니다.

  주: 운영 체제에는 또한 **DB2TCP_CLIENT_CONTIMEOUT**을 사용하여 설정한 시간종료 전에 적용되는 연결 시간종료 값이 있습니다. 예를 들어, AIX에는 75초 후 연결을 종료하는 디폴트 *tcp_keepinit*=150(0.5초 단위)이 있습니다.

DB2TCP_CLIENT_RCVTIMEOUT

- 운영 체제: 모두, 클라이언트 전용
- 디폴트=0(시간종료 없음), 값: 0 - 32,767초
- **DB2TCP_CLIENT_RCVTIMEOUT** 레지스트리 변수는 TCP/IP 수신 조작 시 클라이언트가 데이터를 대기하는 초 수를 지정합니다. 지정된 초 내에 서버의 데이터가 수신되지 않는 경우, DB2 데이터베이스 관리 프로그램은 -30081 selectForRecvTimeout 오류를 리턴합니다.

  레지스트리 변수가 설정되지 않거나 0으로 설정된 경우 시간종료는 없습니다.

  주: DB2TCP_CLIENT_RCVTIMEOUT의 값을 db2cli.ini 키워드 ReceiveTimeout 또는 연결 속성 SQL_ATTR_RECEIVE_TIMEOUT을 사용하여 CLI로 겹쳐쓸 수 있습니다.

DB2TCPCONNMGRS

- 운영 체제: 모두
- 디폴트=직렬 머신의 경우 1, 대칭형 멀티프로세서 머신의 경우 최대 16개의 연결 관리 프로그램으로 반올림되는 프로세서 수의 제곱근. 값: 1 - 16
- 레지스트 변수가 설정되지 않은 경우, 디폴트 수의 연결 관리 프로그램이 작성됩니다. 레지스트리 변수가 설정된 경우, 여기에 지정된 값이 디폴트값을 겹쳐씁니다. 최대 16으로 지정된 수의 TCP/IP 연결 관리 프로그램이 작성됩니다. 1 미만이 지정되면 **DB2TCPCONNMGRS**의 값은 1로 설정되고 값이 범위밖에 있다는 경고가 로그됩니다. 16 이상이 지정되면 **DB2TCPCONNMGRS**이 값은 16으로 설정되고 값이 범위밖에 있다는 경고가 로그됩니다. 1과 16 사이의 값이 지정 값으로 사용됩니다. 둘 이상의 연결 관리 프로그램이 작성되어 있으면 여러 클라이언트 연결이 동시에 수신될 때 연결 처리량이 향상됩니다. 사용자가 SMP 머신에서 실행 중이거나 **DB2TCPCONNMGRS** 레지스트리 변수를 수정한 경우, 추가 TCP/IP 연

결 관리 프로그램 프로세스(UNIX의 경우) 또는 스레드(Windows 운영 체제의 경우)가 있습니다. 추가 프로세스 또는 스레드에는 추가 스토리지가 필요합니다.

주: 연결 관리 프로그램 수를 1로 설정하면 사용자가 여러 명인 시스템, 빈번한 연결 및 연결 끊기가 있는 시스템 또는 두 가지 모두 해당되는 시스템에서 리모트 연결 시 성능이 떨어지는 원인이 됩니다.

명령행 변수

DB2BQTIME

- 운영 체제: 모두
- 디폴트=1초, 최소값: 1초
- 이 변수는 명령행 처리기 프론트엔드에서 백엔드 프로세스가 사용 중이고 해당 프론트엔드에 대한 연결을 설정하는지 여부를 점검하기 전에 해당 프론트엔드가 유휴하는 시간 크기를 지정합니다.

DB2BQTRY

- 운영 체제: 모두
- 디폴트=재시도 60회, 최소값: 재시도 0회
- 이 변수는 명령행 처리기 프론트엔드 프로세스에서 백엔드 프로세스가 이미 사용 중인지 여부를 판별하려고 시도하는 횟수를 지정합니다. 이 변수는 **DB2BQTIME**과 함께 작동합니다.

DB2_CLP_EDITOR

- 운영 체제: 모두
- 디폴트: 메모장(Windows), vi(UNIX), 값: 운영 체제 경로에 있는 모든 유효한 편집기

  주: 이 레지스트리 변수는 설치 중에 디폴트값으로 설정되지 않습니다. 대신에 레지스트리 변수가 설정되지 않은 경우 이 변수를 사용하는 코드가 디폴트값을 사용합니다.
- 이 변수는 EDIT 명령을 실행할 때 사용되는 편집기를 판별합니다. CLP 대화식 세션에서 EDIT 명령은 편집 및 실행될 수 있는 사용자 지정 명령으로 사전 로드된 편집기를 시작합니다.

DB2_CLP_HISTSIZE

- 운영 체제: 모두
- 디폴트: 20, 값: 1–500(두 값 포함)

주: 이 레지스트리 변수는 설치 중에 디폴트값으로 설정되지 않습니다. 대신에 레지스트리 변수가 설정되지 않았거나 유효한 범위밖의 값으로 설정된 경우, 이 변수를 사용하는 코드가 디폴트값 20을 사용합니다.

- 이 변수는 CLP 대화식 세션 중에 명령 실행기록에 저장되는 명령 수를 판별합니다. 명령 실행기록은 메모리에 보유되므로 이 변수의 값이 매우 높으면 세션에서 실행된 명령 수 및 길이에 따라 성능에 영향을 미칩니다.

DB2_CLPPROMPT

- 운영 체제: 모두
- 디폴트=없음(정의되지 않은 경우, 『db2 => 』가 디폴트 대화식 프롬프트로 사용됨), 값: 0개 이상의 %i, %d, %ia, %da 또는 %n 토큰을 포함하는 100자 미만의 텍스트 문자열. 디폴트 CLP 대화식 프롬프트(db2 =>)를 명시적으로 변경하려는 경우를 제외하면 사용자는 이 변수를 설정할 필요가 없습니다.
- 이 레지스트리 변수를 사용하여 사용자는 명령행 처리기(CLP) 대화식 모드에서 사용할 프롬프트를 정의할 수 있습니다. 0개 이상의 선택적 토큰 %i, %d, %ia, %da 또는 %n을(를) 포함하는 100자 미만의 텍스트 문자열로 변수를 설정할 수 있습니다. CLP 대화식 모드에서 실행 중인 경우, 사용되는 프롬프트는 **DB2_CLPPROMPT** 레지스트리 변수에 지정된 텍스트 문자열을 가져와서 %i, %d, %ia, %da 또는 %n 토큰의 모든 어커런스를 각각 현재 첨부된 인스턴스의 로컬 별명, 현재 데이터베이스 연결의 로컬 별명, 현재 첨부된 인스턴스의 권한 부여 ID, 현재 데이터베이스 연결의 권한 부여 ID 및 라인 바꾸기(즉, 캐리지 리턴)로 모두 바꿔 구성됩니다.

  주:

  1. **DB2_CLPPROMPT** 레지스트리 변수가 CLP 대화식 모드에서 변경되는 경우, **DB2_CLPPROMPT**의 새 값은 CLP 대화식 모드가 닫히고 다시 열릴 때까지 적용되지 않습니다.
  2. 인스턴스 첨부가 존재하지 않는 경우, %ia은(는) 빈 문자열로 바뀌고 %i은(는) **DB2INSTANCE** 레지스트리 변수의 값으로 바뀝니다. Windows 플랫폼에서만 **DB2INSTANCE**가 설정되지 않은 경우 %i은(는) **DB2INSTDEF** 레지스트리 변수의 값으로 바뀝니다. 이러한 변수가 모두 설정되지 않은 경우, %i은(는) 빈 문자열로 바뀝니다.
  3. 데이터베이스 연결이 존재하지 않는 경우, %da은(는) 빈 문자열로 바뀌고 %d은(는) **DB2DBDFT** 레지스트리 변수의 값으로 바뀝니다. **DB2DBDFT** 변수가 설정되지 않은 경우, %d은(는) 빈 문자열로 바뀝니다.
  4. 대화식 입력 프롬프트는 항상 위 케이스의 권한 부여 ID, 데이터베이스 이름 및 인스턴스 이름의 값을 표시합니다.

DB2IQTIME

- 운영 체제: 모두
- 디폴트=5초, 최소값: 1초
- 이 변수는 명령행 처리기 벡엔드 프로세스가 명령을 전달하기 위해 프론트 엔드 프로세스의 입력 큐에서 대기하는 시간 크기를 지정합니다.

DB2RQTIME

- 운영 체제: 모두
- 디폴트=5초, 최소값: 1초
- 이 변수는 명령행 처리기 백엔드 프로세스가 프론트 엔드 프로세스의 요청을 대기하는 시간 크기를 지정합니다.

파티션된 데이터베이스 환경 변수

DB2CHGPWD_EEE

- 운영 체제: AIX, Linux 및 Windows의 DB2 ESE
- 디폴트=NULL, 값: YES 또는 NO
- 이 변수는 다른 사용자가 AIX 또는 Windows ESE 시스템에서 암호를 변경하도록 허용하는지 여부를 지정합니다. 모든 데이터베이스 파티션 또는 노드의 암호가 Windows의 Windows 도메인 제어기나 AIX의 LDAP를 사용하여 중앙집중식으로 유지되는지 확인하십시오. 중앙집중식으로 유지되지 않는 경우, 모든 데이터베이스 파티션 또는 노드 전체에서 암호가 불일치할 수 있습니다. 그 결과, 변경을 수행하기 위해 사용자가 연결하는 데이터베이스 파티션에서만 암호가 변경될 수 있습니다.

DB2_FCM_SETTINGS

- 운영 체제: Linux
- 디폴트=YES, 값: FCM_MAXIMIZE_SET_SIZE:[YES|TRUE|NO|FALSE]. FCM_MAXIMIZE_SET_SIZE의 디폴트값은 YES입니다.
- FCM_MAXIMIZE_SET_SIZE 토큰을 사용하여 **DB2_FCM_SETTINGS** 레지스트리 변수를 설정하여 FCM(Fast Communication Manager) 버퍼에 디폴트 2GB의 스페이스를 사전 할당할 수 있습니다. 이 기능을 사용하려면 토큰의 값이 YES 또는 TRUE여야 합니다.

DB2_FORCE_OFFLINE_ADD_PARTITION

- 운영 체제: 모두
- 디폴트=FALSE, 값: FALSE 또는 TRUE
- 이 변수를 사용하여 데이터베이스 파티션 서버 추가 조작이 오프라인으로 수행되도록 지정할 수 있습니다. 디폴트 설정인 FALSE는 데이터베이스를 오프라인으로 설정하지 않고 DB2 데이터베이스 파티션 서버를 추가할 수 있

음을 표시합니다. 그러나 조작을 오프라인으로 수행할 경우 또는 일부 제한사항으로 인해 데이터베이스가 온라인 상태일 때 데이터베이스 파티션 서버를 추가할 수 없는 경우, **DB2_FORCE_OFFLINE_ADD_PARTITION**을 TRUE로 설정하십시오. 이 변수가 TRUE로 설정된 경우, 버전 9.5 및 이전 버전의 동작에 따라 새 DB2 데이터베이스 파티션 서버가 추가됩니다. 즉, 인스턴스가 종료된 후 재시작될 때까지 새 데이터베이스 파티션 서버가 인스턴스에 표시되지 않습니다.

DB2_NUM_FAILOVER_NODES

- 운영 체제: 모두
- 디폴트=2, 값: 0 - 필수 데이터베이스 파티션 수
- 장애 복구 시 머신에서 시작되어야 하는 추가 데이터베이스 파티션 수를 지정하려면 **DB2_NUM_FAILOVER_NODES**를 설정하십시오.

  DB2 데이터베이스 고가용성 솔루션에서 데이터베이스 서버가 실패하는 경우, 실패한 머신의 데이터베이스 파티션을 다른 머신에서 재시작할 수 있습니다. FCM(Fast Communication Manager)은 **DB2_NUM_FAILOVER_NODES**를 사용하여 이 장애 복구를 쉽게 하기 위해 각 머신에서 예약할 메모리 크기를 계산합니다.

  예를 들어, 다음 구성을 고려하십시오.

  – 머신 A에는 두 개의 데이터베이스 파티션이 있습니다(1 및 2).

  – 머신 B에는 두 개의 데이터베이스 파티션이 있습니다(3 및 4).

  – A와 B 모두에서 **DB2_NUM_FAILOVER_NODES**는 2로 설정됩니다.

  START DBM에서 FCM은 한 머신이 실패하는 경우 실패한 머신의 두 데이터베이스 파티션을 다른 머신에서 시작할 수 있도록 최대 네 개의 데이터베이스 파티션을 관리하기 위해 A와 B 모두에서 충분한 메모리를 예약합니다. 머신 A가 실패하는 경우, 데이터베이스 파티션 1 및 2를 머신 B에서 재시작할 수 있습니다. 머신 B가 실패하는 경우, 데이터베이스 파티션 3 및 4를 머신 A에서 재시작할 수 있습니다.

DB2_PARTITIONEDLOAD_DEFAULT

- 운영 체제: 지원되는 모든 ESE 플랫폼
- 디폴트=YES, 값: YES 또는 NO
- **DB2_PARTITIONEDLOAD_DEFAULT** 레지스트리 변수를 사용하면 ESE 특정 로드 옵션이 지정되지 않은 경우 사용자가 ESE 환경에서 로드 유틸리티의 디폴트 동작을 변경할 수 있습니다. 디폴트값은 YES이며 이는 ESE 환경에서 ESE 특정 로드 옵션을 지정하지 않은 경우 목표 테이블이 정의된 모

든 데이터베이스 파티션에서 로딩이 시도되도록 지정합니다. 값이 NO이면 로드 유틸리티가 현재 연결되어 있는 데이터베이스 파티션에서만 로딩이 시도됩니다.

주: 이 변수는 사용되지 않으며 추후 릴리스에서는 제거됩니다. LOAD 명령에는 동일한 동작을 수행하는 데 사용할 수 있는 여러 가지 옵션이 있습니다. LOAD 명령에 다음을 지정하여 이 변수의 NO 설정과 동일한 결과를 얻을 수 있습니다. `PARTITIONED DB CONFIG MODE LOAD_ONLY OUTPUT_DBPARTNUMS x`, 여기서 x는 데이터를 로드하려는 파티션의 파티션 번호입니다.

DB2PORTRANGE

- 운영 체제: Windows
- 값: nnnn:nnnn
- 이 값은 다른 머신에서 작성된 모든 추가 데이터베이스 파티션의 포트 범위도 동일하도록 FCM에서 사용되는 TCP/IP 포트 범위로 설정됩니다.

쿼리 컴파일러 변수

DB2_ANTIJOIN

- 운영 체제: 모두
- ESE 환경에서는 디폴트=NO, 비ESE 환경에서는 디폴트=YES, 값: YES, NO 또는 EXTEND
- DB2 Enterprise Server Edition의 경우: YES가 지정되면 옵티마이저는『NOT EXISTS』서브쿼리를 DB2에서 더 효과적으로 처리할 수 있는 부정형 조인으로 변환할 기회를 검색합니다 . 비ESE 환경의 경우: NO가 지정되면 옵티마이저는『NOT EXISTS』서브쿼리를 부정형 조인으로 변환할 기회를 제한합니다.

  ESE 및 비ESE 환경 모두에서 EXTEND가 지정되면 옵티마이저는 ″NOT IN″과 ″NOT EXISTS″ 서브쿼리를 모두 부정형 조인으로 변환할 기회를 검색합니다.

DB2_DEFERRED_PREPARE_SEMANTICS

- 운영 체제: 모두
- 디폴트=YES, 값: YES 또는 NO
- YES로 설정되면 이 변수는 PREPARE문에 사용된 유형이 지정되지 않은 모든 매개변수 표시문자가 후속 OPEN 또는 EXECUTE문과 연관된 입력 디스크립터를 기반으로 데이터 유형 및 길이 속성을 이끌어내도록 지연된 준비 시맨틱을 사용합니다. 이를 통해 유형이 지정되지 않은 매개변수 표시문자를 이전보다 더 많은 위치에 사용할 수 있습니다.

- **DB2_DEFERRED_PREPARE_SEMANTICS**는 db2start 앞에 설정해야 합니다.

DB2_INLIST_TO_NLJN

- 운영 체제: 모두
- 디폴트=NO, 값: YES 또는 NO
- 일부 상황에서 SQL 및 XQuery 컴파일러는 조인에 대한 IN 목록 술어를 재작성할 수 있습니다. 예를 들어,

```
SELECT *
 FROM EMPLOYEE
 WHERE DEPTNO IN ('D11', 'D21', 'E21')
```

  쿼리를 다음과 같이 재작성할 수 있습니다.

```
SELECT *
 FROM EMPLOYEE, (VALUES 'D11', 'D21', 'E21) AS V(DNO)
 WHERE DEPTNO = V.DNO
```

  DEPTNO에 대한 인덱스가 있는 경우 이 개정판은 향상된 성능을 제공합니다. 값 목록을 먼저 액세스한 다음 Join 술어를 적용할 인덱스를 사용하여 중첩된 루프 조인과 함께 EMPLOYEE에 조인합니다.

  옵티마이저에 쿼리의 재작성 버전에 가장 적합한 조인 메소드를 판별하는 데 필요한 정확한 정보가 없는 경우가 있습니다. 이러한 상황은 옵티마이저가 선택성을 판별하는 데 카탈로그 통계를 사용하지 못하게 하는 매개변수 표시 문자 또는 호스트 변수가 IN 목록에 포함된 경우에 발생합니다. 이 레지스트리 변수는 옵티마이저가 IN 목록을 조인의 내부 테이블로 제공하는 테이블을 사용하여 값 목록을 조인하는 중첩 루프 조인을 기본으로 설정하는 원인이 됩니다.

  주: DB2 쿼리 컴파일러 변수 **DB2_MINIMIZE_LISTPREFETCH** 및 **DB2_INLIST_TO_NLJN** 중 하나 또는 두 변수 모두 YES로 설정되면 REOPT(ONCE)가 지정된 경우에도 해당 변수는 활성 상태를 유지합니다.

DB2_LIKE_VARCHAR

- 운영 체제: 모두
- 디폴트=Y,Y,
- 하위 요소 통계의 사용을 제어합니다. 데이터에 공백으로 구분된 일련의 하위 필드 또는 하위 요소 형식의 구조가 있는 경우 컬럼의 데이터 컨텐츠에 대한 통계가 있습니다. 하위 요소 통계의 콜렉션은 선택적이며 RUNSTATS 명령 또는 API의 옵션으로 제어됩니다.

  이 레지스트리 변수는 옵티마이저가 다음 형식의 술어를 처리하는 방법에 영향을 미칩니다.

```
COLUMN LIKE '%xxxxxx%'
```

여기서 xxxxxx는 문자로 구성된 임의의 문자열입니다.

이 레지스트리 변수가 사용되는 방법을 표시하는 구문은 다음과 같습니다.

```
db2set DB2_LIKE_VARCHAR=[Y|N|S|num1] [,Y|N|S|num2]
```

where

– 쉼표 앞에 있는 용어 또는 술어 오른쪽에 있는 용어는 두 번째 용어가 N으로 지정되거나 컬럼에 양의 하위 요소 통계가 없는 경우에만 다음을 의미합니다.
  - S – 옵티마이저는 함께 병합되어 컬럼을 형성하는 일련의 요소에서 % 문자로 묶인 문자열의 길이를 기반으로 각 요소의 길이를 추정합니다.
  - Y – 디폴트입니다. 알고리즘 매개변수에 디폴트값 1.9를 사용하십시오. 알고리즘 매개변수와 함께 변수 길이 하위 요소 알고리즘을 사용하십시오.
  - N – 고정 길이 하위 요소 알고리즘을 사용하십시오.
  - num1 – 변수 길이 하위 요소 알고리즘에 num1 값을 알고리즘 매개변수로 사용하십시오.

– 쉼표 뒤에 오는 용어는 양의 하위 요소 통계가 있는 컬럼에 대해서만 다음을 의미합니다.
  - N – 하위 요소 통계를 사용하지 않습니다. 첫 번째 용어가 적용됩니다.
  - Y – 디폴트입니다. 양의 하위 요소 통계가 있는 컬럼의 경우에 알고리즘 매개변수에 1.9 디폴트값과 함께 하위 요소 통계를 사용하는 변수 길이 하위 요소 알고리즘을 사용하십시오.
  - num2 – 양의 하위 요소 통계가 있는 컬럼의 경우에 알고리즘 매개변수로 num2 값과 함께 하위 요소 통계를 사용하는 변수 길이 하위 요소 알고리즘을 사용하십시오.

DB2_MINIMIZE_LISTPREFETCH

- 운영 체제: 모두
- 디폴트=NO, 값: YES 또는 NO
- 리스트 프리페치는 인덱스에서 규정 RID 검색, 페이지 번호순 정렬 및 데이터 페이지 프리페치를 포함하는 특수 테이블 액세스 메소드입니다. 옵티마이저에 리스트 프리페치가 적합한 액세스 메소드인지를 판별하는 데 필요한 정확한 정보가 없는 경우가 있습니다. 이러한 상황은 옵티마이저가 선택성을 판별하는 데 카탈로그 통계를 사용하지 못하게 하는 매개변수 표시문자 또는 호스트 변수가 술어 선택성에 포함된 경우에 발생합니다.

이 레지스트리 변수는 옵티마이저가 이러한 상황에서 리스트 프리페치를 고려하는 것을 방지합니다.

주: DB2 쿼리 컴파일러 변수 **DB2_MINIMIZE_LISTPREFETCH** 및 **DB2_INLIST_TO_NLJN** 중 하나 또는 두 변수 모두 YES로 설정되면 REOPT(ONCE)가 지정된 경우에도 해당 변수를 활성 상태를 유지합니다.

DB2_NEW_CORR_SQ_FF

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- ON으로 설정된 경우 특정 서브쿼리 술어에 대해 쿼리 옵티마이저에서 계산한 선택성 값에 영향을 줍니다. 이 변수는 서브쿼리의 SELECT 목록에서 MIN 또는 MAX 집계 함수를 사용하는 등식 서브쿼리 술어의 선택성 값의 정확성을 높이기 위해 사용할 수 있습니다. 예를 들어, 다음과 같습니다.

```
SELECT * FROM T WHERE
T.COL = (SELECT MIN(T.COL)
FROM T WHERE ...)
```

DB2_OPT_MAX_TEMP_SIZE

- 운영 체제: 모두
- 디폴트=NULL, 값: 모든 임시 테이블 스페이스에서 쿼리에 사용할 수 있는 스페이스 *크기*(MB)
- 임시 테이블 스페이스에서 쿼리가 사용할 수 있는 스페이스 *크기*를 제한합니다. **DB2_OPT_MAX_TEMP_SIZE** 설정으로 옵티마이저는 그렇지 않은 경우에 선택되는 플랜보다 비용은 많이 들지만 임시 테이블 스페이스의 스페이스를 적게 사용하는 플랜을 선택하게 됩니다.
  DB2_OPT_MAX_TEMP_SIZE를 설정하는 경우, 설정으로 인해 선택되는 플랜의 유효성에 대해 임시 테이블 스페이스의 사용을 제한할 필요성의 균형을 맞추십시오.

  DB2_WORKLOAD=SAP가 설정된 경우, **DB2_OPT_MAX_TEMP_SIZE**는 자동으로 10 240(10GB)으로 설정됩니다.

  DB2_OPT_MAX_TEMP_SIZE의 값 세트를 초과하여 임시 테이블 스페이스를 사용하는 쿼리를 실행하는 경우, 쿼리는 실패하지 않지만 모든 자원이 사용 가능하지는 않으므로 성능이 준최적 상태일 수 있다는 경고가 수신됩니다.

  DB2_OPT_MAX_TEMP_SIZE로 설정된 한계의 영향을 받는 옵티마이저에서 고려하는 조작은 다음과 같습니다.
  - ORDER BY, DISTINCT, GROUP BY, 병합 스캔 조인 및 중첩된 루프와 같은 조작에 대한 명시적 정렬

– 명시적 임시 테이블
– 해시 조인 및 중복 병합 조인에 대한 내재된 임시 테이블

DB2_REDUCED_OPTIMIZATION

- 운영 체제: 모두
- 디폴트=NO, 값: NO, YES, 임의의 정수, DISABLE, NO_SORT_NLJOIN, 또는 NO_SORT_MGJOIN
- 이 레지스트리 변수를 사용하여 지정된 최적화 레벨에서 최적화 기능의 엄격한 사용 또는 감소된 최적화 기능을 요청할 수 있습니다. 사용되는 최적화 기술의 수를 줄일 경우, 최적화 동안의 시간 및 자원 사용도 줄어듭니다.

  주: 최적화 시간 및 자원 사용이 감소될 수도 있지만, 덜 최적화된 데이터 액세스 플랜을 생성할 위험이 커집니다. IBM 또는 협력업체에서 권고하는 경우에만 이 레지스트리 변수를 사용하십시오.

  – NO로 설정된 경우

    옵티마이저는 최적화 기술을 변경하지 않습니다.

  – YES로 설정된 경우

    최적화 레벨이 5(디폴트) 이하인 경우, 옵티마이저는 상당한 준비 시간 및 자원을 사용하지만 일반적으로 더 나은 액세스 플랜을 생성하지 않는 일부 최적화 기술을 사용하지 않습니다.

    최적화 레벨이 정확히 5인 경우, 옵티마이저는 최적화 시간 및 자원 사용은 줄이지만 덜 최적화된 액세스 플랜을 생성할 위험을 증가시키기도 하는 일부 추가 기술을 줄이거나 사용하지 않습니다. 최적화 레벨이 5보다 낮은 경우, 이러한 기술 중 일부는 어떠한 경우에도 효력이 없을 수 있습니다. 그러나 그럴 경우, 여전히 적용됩니다.

  – 임의의 정수로 설정된 경우

    효과는 YES 설정과 동일하며, 레벨 5로 최적화된 동적으로 준비된 쿼리에 대한 다음 추가 동작이 있습니다. 쿼리 블록의 총 조인 수가 설정을 초과할 경우, 옵티마이저는 레벨 5 최적화 레벨에 대해 위에 설명된 대로 추가 최적화 기술을 사용하지 않는 대신 그리디(greedy) 조인 열거로 전환합니다. 그 결과 쿼리는 최적화 레벨 2와 유사한 레벨에서 최적화됩니다.

  – DISABLE로 설정된 경우

    이 **DB2_REDUCED_OPTIMIZATION** 변수에 의해 제한되지 않을 때 옵티마이저의 동작은 때때로 최적화 레벨 5에서 동적 쿼리의 최적화를 동

적으로 줄일 수 있습니다. 이 설정은 이러한 동작을 사용 불가능하게 하며 옵티마이저는 전체 레벨 5 최적화를 수행해야 합니다.

– NO_SORT_NLJOIN으로 설정된 경우

옵티마이저는 중첩된 루프 조인(NLJN)에 대한 정렬을 강제 실행하는 쿼리 플랜을 생성하지 않습니다. 이러한 정렬 유형은 성능 향상에 유용할 수 있습니다. 따라서 NO_SORT_NLJOIN 옵션을 사용할 때는 성능에 심각하게 영향을 미칠 수 있으므로 주의하십시오.

– NO_SORT_MGJOIN으로 설정된 경우

옵티마이저는 병합 스캔 조인(MSJN)에 대한 정렬을 강제 실행하는 쿼리 플랜을 생성하지 않습니다. 이러한 정렬 유형은 성능 향상에 유용할 수 있습니다. 따라서 NO_SORT_MGJOIN 옵션을 사용할 때는 성능에 심각하게 영향을 미칠 수 있으므로 주의하십시오.

최적화 레벨 5에서의 동적 최적화 감소는 **DB2_REDUCED_OPTIMIZATION**이 YES로 설정되었을 때 정확히 5인 최적화 레벨에 대해 설명된 동작 및 정수 설정에 대해 설명된 동작에 우선합니다.

DB2_SELECTIVITY

- 운영 체제: 모두
- 디폴트=NO, 값: YES 또는 NO
- 이 레지스트리 변수는 SELECTIVITY절을 SQL문의 검색 조건에 사용할 수 있는 위치를 제어합니다.

  이 레지스트리 변수를 YES로 설정하면 SELECTIVITY절을 다음 술어에 지정할 수 있습니다.
  – 하나 이상의 표현식이 호스트 변수를 포함하는 BASIC 술어
  – MATCH 표현식, 술어 표현식 또는 Esc 표현식이 호스트 변수를 포함하는 LIKE 술어

DB2_SQLROUTINE_PREPOPTS

- 운영 체제: 모두
- 디폴트=빈 문자열, 값:
  – APREUSE {YES | NO}
  – BLOCKING {UNAMBIG | ALL | NO}
  – CONCURRENTACCESSRESOLUTION { USE CURRENTLY COMMITTED | WAIT FOR OUTCOME }
  – DATETIME {DEF | USA | EUR | ISO | JIS | LOC}

- DEGREE {1 | *degree-of-parallelism* | ANY}
- DYNAMICRULES {BIND | INVOKEBIND | DEFINEBIND | RUN | INVOKERUN | DEFINERUN}
- EXPLAIN {NO | YES | ALL}
- EXPLSNAP {NO | YES | ALL}
- FEDERATED {NO | YES}
- INSERT {DEF | BUF}
- ISOLATION {CS | RR | UR | RS | NC}
- OPTPROFILE {profile_name | schema_name.profile_name}
- QUERYOPT *optimization-level*
- REOPT {NONE | ONCE | ALWAYS}
- STATICREADONLY {YES|NO}
- VALIDATE {RUN | BIND}

• **DB2_SQLROUTINE_PREPOPTS** 레지스트리 변수를 사용하여 SQL 및 XQuery 프로시저에 대한 프리컴파일 및 바인드 옵션을 사용자 정의할 수 있습니다. 이 변수를 설정할 때 다음과 같이 각 옵션을 공백으로 구분하십시오.

```
db2set DB2_SQLROUTINE_PREPOPTS="BLOCKING ALL VALIDATE RUN"
```

각 옵션 및 해당 설정의 전체 설명은 "BIND 명령"을 참조하십시오.

인스턴스를 재시작하지 않고 개별 프로시저 선택에 대해 **DB2_SQLROUTINE_PREPOPTS**와 동일한 결과를 얻으려면 SET_ROUTINE_OPTS 프로시저를 사용하십시오.

성능 변수

DB2_ALLOCATION_SIZE

• 운영 체제: 모두

• 디폴트=128KB, 범위: 64KB - 256MB

• 버퍼 풀에 대한 메모리 할당의 *크기*를 지정합니다.

이 레지스트리 변수에 높은 값을 설정하면 버퍼 풀에 대해 원하는 메모리 크기에 접근하는 데 할당이 적게 필요하게 되는 이점이 있을 수 있습니다.

이 레지스트리 변수에 높은 값을 설정하면 버퍼 풀이 할당 *크기*의 배수가 아닌 수로 변경되는 경우 메모리가 낭비되는 비용이 들 수 있습니다. 예를 들어, **DB2_ALLOCATION_SIZE**의 값이 8MB이고 버퍼 풀이 4MB로 감소되는 경우, 전체 8MB의 세그먼트를 해제할 수 없으므로 이 4MB는 낭비됩니다.

주: **DB2_ALLOCATION_SIZE**는 사용되지 않으며 추후 릴리스에서는 제거됩니다.

DB2_APM_PERFORMANCE

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- 쿼리 캐시(패키지 캐시)의 동작에 영향을 주는 액세스 플랜 관리 프로그램(APM)의 성능 관련 변경사항을 사용하려면 이 변수를 ON으로 설정하십시오. 이러한 설정은 일반적으로 프로덕션 시스템에 권장됩니다. 이로 인해 패키지 캐시 부족 오류 또는 증가된 메모리 사용 또는 둘 다의 가능성과 같은 제한사항이 발생합니다.

  또한 **DB2_APM_PERFORMANCE**를 ON으로 설정하면 NO PACKAGE LOCK 모드를 사용할 수 있습니다. 이 모드를 사용하여 캐시된 패키지 항목이 제거되는 것을 방지하는 내부 시스템 잠금인 패키지 잠금을 사용하지 않고 전역 쿼리 캐시를 작동할 수 있습니다. NO PACKAGE LOCK 모드는 결과적으로 약간의 성능 향상을 가져오지만 일부 데이터베이스 조작이 허용되지 않습니다. 이러한 금지된 조작에는 패키지를 무효화하는 조작, 패키지를 작동 불가능하게 하는 조작 및 PRECOMPILE, BIND 및 REBIND가 포함됩니다.

DB2ASSUMEUPDATE

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- 사용 가능한 경우, 이 변수를 사용하여 DB2 데이터베이스 시스템은 UPDATE문에 제공된 모든 고정 길이 컬럼이 변경된다고 가정할 수 있습니다. 이로 인해 DB2 데이터베이스 시스템은 컬럼이 실제로 변경되고 있는지 판별하기 위해 기존 컬럼 값과 새 값을 비교할 필요가 없어집니다. 이 레지스트리 변수를 사용하면 갱신할 컬럼이 제공되었지만(예를 들어, SET절에) 실제로는 수정되고 있지 않을 때 추가 로깅 및 인덱스 유지보수가 발생할 수 있습니다.

  DB2ASSUMEUPDATE 레지스트리 변수의 활성화는 db2start 명령에서 유효합니다.

DB2_ASYNC_IO_MAXFILOP

- 운영 체제: 모두
- 디폴트t=*maxfilop* 구성 매개변수의 값, 값: *maxfilop*의 값 - *max_int*의 값
- **DB2_ASYNC_IO_MAXFILOP**는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 이 변수는 스레드된 데이터베이스 관리 프로그램에 의해 유지보수

되는 공유 파일 핸들 테이블 때문에 사용되지 않습니다. 자세한 정보는 ″공유 파일 핸들 테이블″ 주제를 참조하십시오.

DB2_ASYNC_IO_MAXFILOP는 버전 9.5에서 계속 설정될 수 있지만 영향을 미치지 않습니다. 각 데이터베이스에 대해 열 수 있는 파일 핸들 수를 제한하려면 *maxfilop* 구성 매개변수를 참조하십시오.

DB2_AVOID_PREFETCH

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- 응급 복구 중에 프리페치가 사용되는지 여부를 지정합니다. **DB2_AVOID_PREFETCH**=ON이면 프리페치가 사용되지 않습니다.

DB2BPVARS

- 운영 체제: 각 매개변수에 지정된 대로 사용됨
- 디폴트=경로
- 버퍼 풀을 조정하는 데 두 개의 매개변수 세트를 사용할 수 있습니다. Windows에서만 사용 가능한 한 매개변수 세트는 버퍼 풀이 특정 컨테이너 유형에 대해 분산 읽기를 사용하도록 지정합니다. 모든 플랫폼에서 사용 가능한 다른 매개변수 세트는 프리페칭 동작에 영향을 미칩니다.

  매개변수는 `parameter=value` 형식으로 각 라인에 하나씩 ASCII 파일에 지정됩니다. 예를 들어, `bpvars.vars` 파일에는 다음 라인이 포함됩니다.

  ```
  NO_NT_SCATTER = 1
  NUMPREFETCHQUEUES = 2
  ```

  `bpvars.vars`가 `F:₩vars₩`에 저장되어 있다고 가정하고 이러한 변수를 설정하려면 다음 명령을 실행하십시오.

  ```
  db2set DB2BPVARS=F:₩vars₩bpvars.vars
  ```

  분산 읽기 매개변수

  분산 읽기 매개변수는 컨테이너의 각 유형에 대해 대량의 순차 프리페칭이 있는 시스템 및 **DB2NTNOCACHE**를 이미 ON으로 설정한 시스템에 권장됩니다. Windows 플랫폼에서만 사용 가능한 이러한 매개변수는 NT_SCATTER_DMSFILE, NT_SCATTER_DMSDEVICE 및 NT_SCATTER_SMS입니다. 컨테이너에 대해 분산 읽기를 명시적으로 승인 불가하려면 NO_NT_SCATTER 매개변수를 지정하십시오. 특정 매개변수가 표시된 유형의 모든 컨테이너에 대해 분산 읽기를 켜는 데 사용됩니다. 이러한 매개변수 각각에 대해 디폴트는 0(또는 OFF)이고 가능한 값은 0(또는 OFF) 및 1(또는 ON)입니다.

주: DB2NTNOCACHE가 ON으로 설정되어 Windows 파일 캐싱을 끈 경우에만 분산 읽기를 켤 수 있습니다. **DB2NTNOCACHE**가 OFF로 설정되거나 설정되지 않은 경우, 컨테이너에 대해 분산 읽기를 켜려고 시도하면 경고 메시지가 관리 통지 로그에 기록되며 분산 읽기는 사용 안함을 유지합니다.

프리페치 조정 매개변수

프리페치 조정 매개변수는 NUMPREFETCHQUEUES 및 PREFETCHQUEUESIZE입니다. 해당 매개변수는 모든 플랫폼에서 사용 가능하며 버퍼 풀 데이터 프리페칭을 향상시키는 데 사용할 수 있습니다. 예를 들어, 원하는 PREFETCHSIZE가 PREFETCHSIZE/EXTENTSIZE 프리페치 요청으로 나뉘는 순차 프리페칭을 고려하십시오. 이 경우, 요청은 입출력 서버가 디스패치되어 비동기 입출력을 수행하는 프리페치 큐에 배치됩니다. 디폴트로 DB2 데이터베이스 관리 프로그램은 각 데이터베이스 파티션에 대해 하나의 *큐 크기* max(200,2*NUM_IOSERVERS)를 유지합니다. 일부 환경에서는 더 많은 큐 또는 다른 *크기*의 큐 또는 둘 다를 사용하면 성능이 향상됩니다. 프리페치 큐 수는 입출력 서버 수의 2분의 1 이하여야 합니다. 이러한 매개변수를 설정할 때 PREFETCHSIZE, EXTENTSIZE, NUM_IOSERVERS 및 버퍼 풀 *크기*와 같은 다른 매개변수 및 현재 사용자 수와 같은 워크로드 특성를 고려하십시오.

사용자 환경에 대해 디폴트값이 너무 작다고 생각하면 먼저 값을 약간만 증가시키십시오. 예를 들어, NUMPREFETCHQUEUES=4 및 PREFETCHQUEUESIZE=200을 설정합니다. 변경 효과를 모니터하고 평가할 수 있도록 제어된 방식으로 이러한 매개변수를 변경하십시오.

NUMPREFETCHQUEUES의 경우, 디폴트는 1이고 값의 범위는 1 - NUM_IOSERVERS입니다. NUMPREFETCHQUEUES를 1 미만으로 설정하면 1로 조정됩니다. NUM_IOSERVERS보다 크게 설정하면 NUM_IOSERVERS로 조정됩니다.

PREFETCHQUEUESIZE의 경우, 디폴트값은 max(200,2*NUM_IOSERVERS)입니다. 값의 범위는 1 - 32,767입니다. PREFETCHQUEUESIZE를 1 미만으로 설정하면 디폴트로 조정됩니다. 32,767보다 크게 설정하면 32,767로 조정됩니다.

주: DB2BPVARS는 사용되지 않으며 추후 릴리스에서는 제거됩니다.

DB2CHKPTR

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF

- 입력에 대한 포인터 검사가 필요한지 여부를 지정합니다.

DB2CHKSQLDA

- 운영 체제: 모두
- 디폴트=ON, 값: ON 또는 OFF
- 입력에 대한 SQLDA 검사가 필요한지 여부를 지정합니다.

DB2_EVALUNCOMMITTED

- 운영 체제: 모두
- 디폴트=OFF, 값: ON, OFF
- 이 변수를 사용할 수 있는 경우, 이를 사용하면 가능한 경우에 술어 평가를 만족시키기 위해 데이터가 알려질 때까지 스캔이 행 잠금을 지연하거나 방지할 수 있습니다. 이 변수가 사용 가능한 경우, 술어 평가는 언커미트된 데이터에서 발생합니다.

  DB2_EVALUNCOMMITTED는 현재 커미트된 시맨틱이 잠금 경합을 방지하는 데 도움이 되지 않는 경우에만 적용 가능합니다. 이 변수가 설정되고 현재 커미트되어 스캔에 적용 가능한 경우, 삭제된 행을 건너뛰지 않고 언커미트된 데이터에서 술어 평가가 발생하지 않습니다. 대신에 현재 커미트된 행 및 데이터 버전이 처리됩니다.

  마찬가지로 **DB2_EVALUNCOMMITTED**는 커서 안정성 또는 읽기 안정성 분리 레벨을 사용하는 명령문에만 적용됩니다. 또한 레지스트리 변수 **DB2_SKIPDELETED**도 설정된 경우가 아니면 인덱스 스캔에 대해서는 삭제된 키를 건너뛰지 않는 반면에 테이블 액세스 스캔 시에는 삭제된 행을 무조건 건너뜁니다.

  DB2_EVALUNCOMMITTED 레지스트리 변수의 활성화는 db2start 명령에서 유효합니다. 지연된 잠금이 적용 가능한지 여부는 명령문 컴파일 또는 바인드 시간에 결정됩니다.

DB2_EXTENDED_IO_FEATURES

- 운영 체제: AIX
- 디폴트=OFF, 값: ON, OFF
- 입출력 성능을 향상시키는 기능을 사용하려면 이 변수를 ON으로 설정하십시오. 이 향상된 기능에는 메모리 캐시 적중률 향상 및 높은 우선순위 입출력 시 대기 시간 줄이기가 포함됩니다. 이러한 기능은 특정 조합의 소프트웨어 및 하드웨어 구성에서만 사용 가능합니다. 다른 구성에 대해 이 변수를 ON으로 설정하면 DB2 데이터베이스 관리 시스템 또는 운영 체제에서 무시됩니다. 최소 구성 요구사항은 다음과 같습니다.
  - 데이터베이스 버전: DB2 V9.1

– 데이터베이스 컨테이너에 RAW 디바이스를 사용해야 함(파일 시스템의 컨테이너는 지원되지 않음)
– 운영 체제: AIX 5.3 TL4
– 스토리지 서브시스템: Shark DS8000®은 향상된 모든 입출력 성능 기능을 지원합니다. 설치 및 전제조건 정보는 Shark DS8000 문서를 참조하십시오.

HIGH, MEDIUM 및 LOW의 디폴트 입출력 우선순위 설정값은 각각 3, 8 및 12입니다. **DB2_IO_PRIORITY_SETTING** 레지스트리 변수를 사용하여 이러한 설정값을 변경할 수 있습니다.

DB2_EXTENDED_OPTIMIZATION

- 운영 체제: 모두
- 디폴트=OFF, 값: ON, OFF 또는 ENHANCED_MULTIPLE_DISTINCT
- 이 변수는 쿼리 옵티마이저가 최적화 확장을 사용하여 쿼리 성능을 향상시키는지 여부를 지정합니다. ON 및 ENHANCED_MULTIPLE_DISTINCT 값은 서로 다른 최적화 확장을 지정합니다. 두 값 모두 사용하려면 쉼표로 구분된 목록을 사용하십시오.

  ENHANCED_MULTIPLE_DISTINCT 값은 하나의 단일 선택 조작에 여러 개의 구별 집계 조작이 포함되어 있는 경우 및 데이터베이스 파티션 수에 대한 프로세서의 비율이 낮은 경우(예를 들어, 비율이 1 이하임)의 쿼리 성능을 향상시킵니다. 이 설정은 대칭 멀티프로세서(SMP)가 없는 DPF(Database Partitioning Feature) 환경에서 사용되어야 합니다.

  최적화 확장은 일부 환경에서는 쿼리 성능을 향상시키지 않습니다. 개별 쿼리 성능 향상을 판별하려면 테스트를 수행해야 합니다.

DB2_HASH_JOIN

- 운영 체제: 모두
- 디폴트=YES, 값: YES 또는 NO
- 액세스 플랜을 컴파일할 때 해시 조인을 가능한 조인 메소드로 지정합니다. 최상의 성능을 얻으려면 **DB2_HASH_JOIN**을 조정해야 합니다. 해시 루프 및 디스크에 대한 오버플로우를 방지할 수 있는 경우 해시 조인 성능은 최상의 상태가 됩니다. 해시 조인 성능을 조정하려면 *sheapthres* 구성 매개변수에 사용할 수 있는 최대 메모리 양을 추정한 후 *sortheap* 구성 매개변수를 조정하십시오. 가능하면 많은 해시 루프 및 디스크 오버플로우를 방지할 수 있을 때까지 값을 늘리지만 *sheapthres* 구성 매개변수에 지정된 한계에는 접근하지 마십시오.

주: DB2_HASH_JOIN은 버전 9.5에서는 사용되지 않으며 추후 릴리스에서는 제거됩니다.

DB2_IO_PRIORITY_SETTING

- 운영 체제: AIX
- 값: HIGH:#, MEDIUM:#, LOW:#, 여기서 #는 1 - 15 사이의 값입니다.
- 이 변수는 **DB2_EXTENDED_IO_FEATURES** 레지스트리 변수와 조합하여 사용됩니다. 이 레지스트리 변수는 DB2 데이터베이스 시스템에 대한 디폴트 HIGH, MEDIUM 및 LOW 입출력 우선순위 설정값(각각 3, 8 및 12)을 겹쳐쓰는 수단을 제공합니다. 이 레지스트리 변수는 인스턴스 시작 전에 설정해야 합니다. 수정하면 인스턴스를 재시작해야 합니다. 이 레지스트리 변수 설정만으로는 향상된 입출력 기능을 사용할 수 없습니다. 해당 기능을 사용하려면 **DB2_EXTENDED_IO_FEATURES**를 설정해야 합니다. 또한 **DB2_EXTENDED_IO_FEATURES**에 대한 모든 시스템 요구사항이 이 레지스트리 변수에도 적용됩니다.

DB2_KEEP_AS_AND_DMS_CONTAINERS_OPEN

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- 이 변수를 ON으로 설정하면 데이터베이스가 비활성화될 때까지 각 DMS 테이블 스페이스 컨테이너의 파일 핸들이 열려 있습니다. 컨테이너는 열기 위한 오버헤드가 제거되므로 쿼리 성능이 향상됩니다. 이 레지스트리 변수는 순수 DMS 환경에서만 사용해야 합니다. 그렇지 않으면 SMS 테이블 스페이스에 대한 쿼리 성능에 부정적인 영향이 미칩니다.

DB2_KEEPTABLELOCK

- 운영 체제: 모두
- 디폴트: OFF, 값: ON, TRANSACTION, OFF, CONNECTION
- 이 변수가 ON 또는 TRANSACTION으로 설정되면 언커미트된 읽기 또는 커서 안정성 분리 레벨이 닫힌 경우 이 변수를 사용하여 DB2 데이터베이스 시스템이 테이블 잠금을 유지할 수 있습니다. 보존되는 테이블 잠금은 읽기 안정성 및 반복 읽기 스캔에 대해 릴리스되는 것처럼 트랜잭션 끝에 릴리스됩니다.

  이 변수가 CONNECTION으로 설정되면 응용프로그램이 트랜잭션을 롤백하거나 연결이 재설정될 때까지 테이블 잠금이 응용프로그램에 대해 릴리스됩니다. 테이블 잠금은 커미트 전체에 걸쳐서 계속 보유되며 테이블 잠금을 삭제하려는 응용프로그램 요청은 데이터베이스에서 무시됩니다. 테이블 잠금은 응용프로그램에 할당된 채로 남아 있습니다. 따라서 응용프로그램이 테이블 잠금을 다시 요청할 때 잠금은 이미 사용 가능합니다.

이 최적화를 가공할 수 있는 응용프로그램 워크로드의 경우, 성능이 향상되어야 합니다. 그러나 동시에 실행 중인 다른 응용프로그램의 워크로드는 영향을 받습니다. 다른 응용프로그램은 결과적으로 낮은 동시성을 가져오는 지정된 테이블에 대한 액세스로부터 차단됩니다. DB2 SQL 카탈로그 테이블은 이 설정의 영향을 받지 않습니다. CONNECTION 설정은 또한 ON 또는 TRANSACTION 설정으로 설명한 동작도 포함합니다.

이 레지스트리 변수는 명령문 컴파일 또는 바인드 시간에 점검됩니다.

DB2_LARGE_PAGE_MEM

- 운영 체제: AIX, Linux, Windows Server 2003
- 디폴트=NULL, 값: 대형 페이지 메모리를 사용해야 하는 모든 적용 가능한 메모리 영역 또는 대형 페이지 메모리를 사용해야 하는 특정 메모리 영역의 쉼표로 구분된 목록을 표시하려면 *를 사용하십시오. 사용 가능한 영역은 운영 체제에 따라 다릅니다. AIX에서는 DB, DBMS, FCM 또는 PRIVATE 영역을 지정할 수 있습니다. Linux에서는 DB 영역을 지정할 수 있습니다. Windows Server 2003에서는 DB 영역을 지정할 수 있습니다. 초대형 페이지 메모리는 AIX에서만 사용 가능합니다.
- **DB2_LARGE_PAGE_MEM** 레지스트리 변수는 대형 페이지 또는 초대형 페이지 지원을 사용 가능하게 하는 데 사용됩니다.
  DB2_LARGE_PAGE_MEM=DB 설정은 데이터베이스 공유 메모리 영역에 대해 대형 페이지 메모리를 사용 가능하게 하며 **database_memory**가 AUTOMATIC으로 설정되어 있으면 STMM에 의한 이 공유 메모리 영역의 자동 성능 조정을 사용 불가능하게 합니다. AIX에서 DB2_LARGE_PAGE_MEM=DB:16GB 설정은 데이터베이스 공유 메모리 영역에 대해 초대형 페이지 메모리를 사용 가능하게 합니다.

  많은 양의 가상 메모리를 사용하는 메모리 액세스 집중형 응용프로그램은 대형 또는 초대형 페이지를 사용하여 성능 향상을 얻습니다. DB2 데이터베이스 시스템이 이러한 페이지를 사용할 수 있게 하려면 먼저 대형 또는 초대형 페이지를 사용하도록 운영 체제를 구성해야 합니다.

  AIX용 64비트 DB2에서 에이전트 전용 메모리의 대형 페이지를 사용 가능하게 하려면(**DB2_LARGE_PAGE_MEM**=PRIVATE 설정), 운영 체제에서 대형 페이지를 구성해야 하며 인스턴스 소유자는 CAP_BYPASS_RAC_VMM 및 CAP_PROPAGATE 기능을 소유해야 합니다.

AIX 5L™에서는 이 변수를 FCM으로 설정할 수 있습니다. FCM 메모리는 고유 메모리 세트에 상주하므로 FCM 메모리에 대한 대형 페이지를 사용하려면 FCM 키워드를 **DB2_LARGE_PAGE_MEM** 레지스트리 변수의 값에 추가해야 합니다.

Linux의 경우, *libcap.so.1* 라이브러리의 사용 가능성에 대한 추가적인 요구사항이 있습니다. 이 옵션이 작동하려면 이 라이브러리가 설치되어 있어야 합니다. 이 옵션은 커져 있고 라이브러리가 시스템에 없는 경우, DB2 데이터베이스는 대형 커널 페이지를 사용 불가능하게 하고 계속 해당 페이지가 없을 때처럼 작동합니다.

Linux에서 대형 커널 페이지가 사용 가능한지 검증하려면 다음 명령을 발행하십시오.

```
cat /proc/meminfo
```

대형 커널 페이지가 사용 가능한 경우, 다음의 세 라인이 표시되어야 합니다(사용자 서버에 구성된 메모리 *크기*에 따라 다른 숫자가 함께 표시됨).

```
HugePages_Total:   200
HugePages_Free:    200
Hugepagesize:    16384 kB
```

이러한 라인이 표시되지 않거나 HugePages_Total이 0인 경우, 운영 체제 또는 커널을 구성해야 합니다.

Windows의 경우, 시스템에서 사용 가능한 대형 페이지 메모리 *크기*는 사용 가능한 전체 메모리보다 적습니다. 일정 시간 동안 시스템이 실행된 후에는 메모리가 분할될 수 있고 대형 페이지 메모리의 *크기*는 줄어듭니다.

DB2_LOGGER_NON_BUFFERED_IO

- 운영 체제: 모두
- 디폴트=AUTO, 값: AUTOMATIC, ON, 또는 OFF
- 이 변수를 사용하여 로그 파일 시스템에서 직접 입출력(DIO)을 사용할 수 있습니다. **DB2_LOGGER_NON_BUFFERED_IO**가 AUTOMATIC으로 설정되어 있으면 활성 로그 창(즉, 1차 로그 파일)이 DIO로 열리고 다른 모든 로그 프로그램 파일이 버퍼됩니다. ON으로 설정되어 있으면 모든 로그 파일 핸들이 DIO로 열립니다. OFF으로 설정되어 있으면 모든 로그 파일 핸들이 버퍼됩니다.

DB2MAXFSCRSEARCH

- 운영 체제: 모두
- 디폴트=5, 값: -1, 1 - 33,554

- 테이블에 레코드를 추가할 때 검색할 여유 공간 제어 레코드(FSCR)의 수를 지정합니다. 디폴트는 다섯 개의 FSCR을 검색하는 것입니다. 이 값을 수정하면 스페이스 재사용과 삽입 속도의 균형을 맞출 수 있습니다. 스페이스 재사용에 맞게 최적화하려면 큰 값을 사용하십시오. 삽입 속도에 맞게 최적화하려면 작은 값을 사용하십시오. 값을 -1로 설정하면 데이터베이스 관리 프로그램이 모든 FSCR을 강제로 검색합니다.

DB2_MAX_INACT_STMTS

- 운영 체제: 모두
- 디폴트=설정되지 않음, 값: 최대 4GB
- 이 변수는 하나의 응용프로그램에서 보존되는 비활성 명령문 수에 대한 디폴트 한계를 겹쳐씁니다. 비활성 명령문 정보에 사용되는 시스템 모니터 힙 크기를 늘리거나 줄이기 위해 다른 값을 선택할 수 있습니다. 디폴트 한계는 250입니다.

  응용프로그램이 작업 단위에 매우 많은 수의 명령문을 포함하고 있거나 동시에 실행 중인 응용프로그램이 많은 경우 시스템 모니터 힙이 모두 사용될 수 있습니다.

DB2_MAX_NON_TABLE_LOCKS

- 운영 체제: 모두
- 디폴트=YES, 값: 설명 참조
- 이 변수는 트랜잭션이 NON 테이블 잠금을 모두 릴리스하기 전에 가질 수 있는 최대 해당 잠금 수를 정의합니다. NON 테이블 잠금은 트랜잭션에서 사용을 완료한 경우에도 해시 테이블 및 트랜잭션 체인에 보존되는 테이블 잠금입니다. 트랜잭션이 자주 동일한 테이블에 두 번 이상 액세스하므로 잠금을 보유하고 상태를 NON으로 변경하면 성능이 향상될 수 있습니다.

  최상의 결과를 위해서 이 변수에 권장되는 값은 연결에 의해 액세스될 것으로 예측되는 최대 테이블 수입니다. 사용자 정의 값이 지정되지 않는 경우 디폴트값은 다음과 같습니다. 잠금 목록 크기가 다음보다 크거나 같은 경우

  ```
  SQLP_THRESHOLD_VAL_OF_LRG_LOCKLIST_SZ_FOR_MAX_NON_LOCKS
  ```

  (현재 8,000), 디폴트값은 다음과 같습니다.

  ```
  SQLP_DEFAULT_MAX_NON_TABLE_LOCKS_LARGE
  ```

  (현재 150). 그렇지 않은 경우 디폴트값은 다음과 같습니다.

  ```
  SQLP_DEFAULT_MAX_NON_TABLE_LOCKS_SMALL
  ```

  (현재 0).

DB2_MDC_ROLLOUT

- 운영 체제: 모두
- 디폴트=IMMEDIATE, 값: IMMEDIATE, OFF 또는 DEFER
- 이 변수는 MDC 테이블에서의 삭제에 『롤아웃』으로 알려진 성능 향상 기능을 사용 가능하게 합니다. 롤아웃은 전체 셀(차원 값의 교차)이 검색 DELETE문에서 삭제될 때 MDC 테이블에서 행을 삭제하는 빠른 방법입니다. 로깅 수가 감소되고 더 효율적으로 처리되는 것이 이점입니다.
- 세 가지 변수 설정 결과가 가능합니다.
  - 롤아웃 사용 안함 - OFF가 지정된 경우
  - 즉시 롤아웃 - IMMEDIATE가 지정된 경우
  - 인덱스 정리가 지연된 롤아웃 - DEFER가 지정된 경우
- 시작 후 값이 변경되면 명령문의 새 컴파일은 새 레지스트리 값 설정을 고려합니다. 패키지 캐시에 있는 명령문의 경우, 명령문이 재컴파일될 때까지 삭제 처리를 변경하지 않습니다. SET CURRENT MDC ROLLOUT MODE 명령문은 응용프로그램 연결 레벨의 **DB2_MDC_ROLLOUT** 값을 겹쳐씁니다.

DB2MEMDISCLAIM

- 운영 체제: 모두
- 디폴트=YES, 값: YES 또는 NO
- DB2 데이터베이스 시스템 프로세스에서 사용한 메모리에는 연관된 페이징 스페이스가 있을 수 있습니다. 이 페이징 스페이스는 연관된 메모리가 해제된 경우에도 예약된 채로 남아 있을 수 있습니다. 이에 대한 여부는 운영 체제의 조정 가능한 가상 메모리 관리 할당 규정에 따라 다릅니다. **DB2MEMDISCLAIM** 레지스트리 변수는 운영 체제가 해제된 메모리에서 예약된 페이징 스페이스를 연관 해제하도록 DB2 에이전트가 명시적으로 요청하는지 여부를 제어합니다.

  DB2MEMDISCLAIM을 YES로 설정하면 그 결과 페이징 스페이스 요구사항이 작아지고 페이징에서 디스크 활동이 적어질 수 있습니다. **DB2MEMDISCLAIM**을 NO로 설정하면 그 결과 페이징 스페이스 요구사항이 커지고 페이징에서 디스크 활동이 많아질 수 있습니다. 페이징 스페이스가 충분하고 실제 메모리도 충분하여 페이징이 발생하지 않는 경우와 같은 일부 상황에서 NO를 설정하면 성능이 약간 향상됩니다.

DB2MEMMAXFREE

- 운영 체제: 모두
- 디폴트=NULL, 값: 0 - 2^{32}-1 바이트

- 사용되지 않은 메모리가 운영 체제에 리턴되기 전에 DB2 데이터베이스 시스템에서 보유하는 사용되지 않은 전용 메모리의 최대 바이트 수를 지정합니다.

  DB2MEMMAXFREE가 설정되지 않은 경우, DB2 데이터베이스 시스템 프로세스는 사용할 수 있는 메모리를 다시 운영 체제에 돌려주기 전에 사용되지 않은 전용 메모리의 최대 20%를 보유합니다(현재 사용된 전용 메모리의 양을 기반으로 함).

주: DB2MEMMAXFREE는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 데이터베이스 관리 프로그램이 이제 스레드된 엔진 모델을 사용하므로 이 변수는 더 이상 필요하지 않습니다. 이 변수를 설정하지 마십시오. 이 변수를 설정하면 성능이 떨어지고 예기치 않은 동작을 일으킬 수 있습니다.

DB2_MEM_TUNING_RANGE

- 운영 체제: 모두
- 디폴트=NULL, 값: 백분율 시퀀스 n, m 여기서 n=*최소 여유 공간*이고 m=*최대 여유 공간*
- DB2 인스턴스가 여유 공간으로 남겨두는 실제 메모리의 *크기*는 같은 머신에서 실행 중인 다른 응용프로그램이 사용할 수 있는 메모리 *크기*를 결정하므로 중요합니다. 데이터베이스 공유 메모리의 자체 성능 조정이 사용 가능한 경우, 지정된 인스턴스가 여유 공간으로 남긴 실제 메모리의 *크기*는 인스턴스(및 해당 활성 데이터베이스)의 메모리 필요성에 따라 다릅니다. 인스턴스에 추가 메모리가 긴급하게 필요한 경우, 인스턴스는 시스템의 사용할 수 있는 실제 메모리가 *최소 여유 공간*에서 지정한 백분율에 접근할 때까지 메모리를 할당합니다. 인스턴스에 메모리가 적게 필요한 경우, 인스턴스는 *최대 여유 공간*에서 백분율로 지정한 많은 양의 사용할 수 있는 실제 메모리를 유지합니다. 그 결과, *최소 여유 공간*에 설정된 값이 *최대 여유 공간*의 값보다 작아야 합니다.

  이 변수가 설정되지 않은 경우, DB2 데이터베이스 관리 프로그램은 서버의 메모리 *크기*를 기준으로 *최소 여유 공간* 및 *최대 여유 공간*의 값을 계산합니다. 자체 성능 조정 메모리 관리자(STMM)를 실행 중이고 **database_memory**가 AUTOMATIC으로 설정되어 있지 않으면 이러한 변수 설정은 적용되지 않습니다.

DB2_MMAP_READ

- 운영 체제: AIX
- 디폴트=OFF, 값: ON 또는 OFF
- 이 변수를 **DB2_MMAP_WRITE**와 함께 사용하면 DB2 데이터베이스 시스템이 mmap를 대체 입출력 메소드로 사용할 수 있습니다.

두 변수가 ON으로 설정되면, 읽혀서 DB2 버퍼 풀에 전달되고 해당 버퍼 풀에서 작성된 데이터는 AIX 메모리 캐시를 무시합니다. 상대적으로 작은 DB2 버퍼 풀을 가지고 있고 이 버퍼 풀의 *크기*를 늘릴 수 없거나 늘리지 않도록 선택한 경우, **DB2_MMAP_READ** 및 **DB2_MMAP_WRITE**를 OFF로 설정하여 AIX 메모리 캐싱을 이용해야 합니다.

DB2_MMAP_WRITE

- 운영 체제: AIX
- 디폴트=OFF, 값: ON 또는 OFF
- 이 변수를 **DB2_MMAP_READ**와 함께 사용하면 DB2 데이터베이스 시스템이 mmap를 대체 입출력 메소드로 사용할 수 있습니다.

  두 변수가 ON으로 설정되면, 읽혀서 DB2 버퍼 풀에 전달되고 해당 버퍼 풀에서 작성된 데이터는 AIX 메모리 캐시를 무시합니다. 상대적으로 작은 DB2 버퍼 풀을 가지고 있고 이 버퍼 풀의 *크기*를 늘릴 수 없거나 늘리지 않도록 선택한 경우, **DB2_MMAP_READ** 및 **DB2_MMAP_WRITE**를 OFF로 설정하여 AIX 메모리 캐싱을 이용해야 합니다.

DB2_NO_FORK_CHECK

- 운영 체제: UNIX
- 디폴트=OFF, 값: ON 또는 OFF
- 이 변수가 사용 가능하면 DB2 런타임 클라이언트가 현재 프로세스가 포크 호출의 결과인지를 판별하기 위한 검사를 최소화합니다. 이는 fork() API를 사용하지 않는 DB2 응용프로그램의 성능을 향상시킬 수 있습니다.

주: 이 변수는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 프로세스가 시작되거나 새로 분기될 때 현재 프로세스 ID(pid)가 캐시되므로 필요하지 않습니다.

DB2NTMEMSIZE

- 운영 체제: Windows
- 디폴트=(메모리 세그먼트에 따라 다름)
- Windows에서는 프로세스 전체에 걸쳐서 주소 일치를 보장하기 위해 DLL 초기화 시간에 모든 공유 메모리 세그먼트가 예약되어야 합니다. **DB2NTMEMSIZE**를 사용하여 사용자는 필요한 경우 Windows의 DB2 디폴트를 겹쳐씁니다. 대부분의 상황에서는 디폴트값으로 충분합니다. 메모리 세그먼트, 디폴트 *크기* 및 겹쳐쓰기 옵션은 다음과 같습니다.
  1. 병렬 FCM 버퍼: 디폴트 *크기*는 32비트 플랫폼의 경우 512MB이고 64비트 플랫폼의 경우 4.5GB입니다. 겹쳐쓰기 옵션은 FCM:<바이트 수>입니다.

2. 분리(Fenced) 모드 통신: 디폴트 크기는 32비트 플랫폼의 경우 80MB이고 64비트 플랫폼의 경우 512MB입니다. 겹쳐쓰기 옵션은 APLD:<바이트 수>입니다.

세미콜론(;)으로 겹쳐쓰기 옵션을 구분하여 둘 이상의 세그먼트를 겹쳐쓸 수 있습니다. 예를 들어, 32비트 버전 DB2에서 FCM 버퍼를 1GB로 제한하고 분리 스토어드 프로시저를 256MB로 제한하려면 다음을 사용하십시오.

```
db2set DB2NTMEMSIZE=FCM:1073741824;APLD:268435456
```

DB2NTNOCACHE

- 운영 체제: Windows
- 디폴트=OFF, 값: ON 또는 OFF
- **DB2NTNOCACHE** 레지스트리 변수는 DB2 데이터베이스 시스템이 NOCACHE 옵션을 사용하여 데이터베이스 파일을 여는지 여부를 지정합니다. **DB2NTNOCACHE**=ON이면 파일 시스템 캐싱이 제거됩니다. **DB2NTNOCACHE**=OFF이면 운영 체제가 DB2 파일을 캐시합니다. 이는 Long 필드 또는 LOB를 포함하는 파일을 제외한 모든 데이터에 적용됩니다. 시스템 캐싱을 제거하면 버퍼 풀 또는 정렬 힙을 늘릴 수 있도록 더 많은 메모리를 데이터베이스에 사용할 수 있습니다.

  Windows에서는 파일이 열려 있으면 캐시되며 이는 디폴트 동작입니다. 파일의 1GB마다 1MB가 시스템 풀에서 예약됩니다. 캐시의 미등록 한계 192MB를 겹쳐쓰려면 이 레지스트리 변수를 사용하십시오. 캐시 한계에 접근하면 자원 부족 오류가 표시됩니다.

주: DB2NTNOCACHE는 버전 8.2 이후 사용되지 않고 있으며 추후 릴리스에서는 제거됩니다. CREATE TABLESPACE 및 ALTER TABLESPACE SQL문을 사용하여 테이블 스페이스 컨테이너에 대해 동일한 이점을 얻을 수 있습니다.

DB2NTPRICLASS

- 운영 체제: Windows
- 디폴트=NULL, 값: R, H, (기타 모든 값)
- DB2 인스턴스의 우선순위 클래스를 설정합니다(프로그램 DB2SYSCS.EXE). 세 가지 우선순위 클래스가 있습니다.
  - NORMAL_PRIORITY_CLASS(디폴트 우선순위 클래스)
  - REALTIME_PRIORITY_CLASS(『R』을 사용하여 설정)
  - HIGH_PRIORITY_CLASS(『H』를 사용하여 설정)

이 변수는 개별 스레드 우선순위(**DB2PRIORITIES**를 사용하여 설정)와 함께 사용되어 시스템의 기타 스레드와 비교하여 DB2 스레드의 절대 우선순위를 판별합니다.

주: DB2NTPRICLASS는 사용되지 않으며 서비스 권장사항에서만 사용되어야 합니다. 에이전트 우선순위 및 프리페치 우선순위를 조정하려면 DB2 서비스 클래스를 사용하십시오. 이 변수를 사용하려면 주의하십시오. 잘못 사용하면 전체 시스템 성능에 나쁜 영향을 미칠 수 있습니다.

자세한 정보는 Win32 문서에서 SetPriorityClass() API를 참조하십시오.

DB2NTWORKSET

- 운영 체제: Windows
- 디폴트=1,1
- DB2 데이터베이스 관리 프로그램에 사용 가능한 최소 및 최대 작업 세트 *크기*를 수정하는 데 사용됩니다. 디폴트로 Windows가 페이징 상황에 있지 않으면 프로세스의 작업 세트는 필요한 만큼 커질 수 있습니다. 그러나 페이징이 발생하면 프로세스가 가질 수 있는 최대 작업 세트는 약 1MB입니다. **DB2NTWORKSET**를 사용하여 이 디폴트 동작을 겹쳐쓸 수 있습니다.

  DB2NTWORKSET=min, max 구문을 사용하여 **DB2NTWORKSET**를 지정하십시오. 여기서 min 및 max는 MB로 표시됩니다.

DB2_OVERRIDE_BPF

- 운영 체제: 모두
- 디폴트=설정되지 않음, 값: 양의 페이지 수 또는 <항목>[;<항목>...] 여기서 <항목>=<버퍼 풀 ID>,<페이지 수>
- 이 변수는 데이터베이스 활성화, 롤 포워드 복구 또는 응급 복구 시 작성될 버퍼 풀의 *크기*를 지정합니다(페이지 단위). 이 변수는 메모리 제한조건으로 인해 데이터베이스 활성화, 롤 포워드 복구 또는 응급 복구 동안 오류가 발생하는 경우에 유용합니다. 메모리 제한조건은 드물게 실제 메모리 부족인 경우나 또는 데이터베이스 관리 프로그램이 대형 버퍼 풀을 할당하려는 시도 때문에 잘못 구성된 버퍼 풀이 있는 경우에 발생할 수 있습니다. 예를 들어, 데이터베이스 관리 프로그램이 16페이지의 최소 버퍼 풀을 불러오지 않은 경우에도 이 환경 변수를 사용하여 더 작은 페이지 수를 지정하십시오. 이 변수에 지정된 값이 현재 버퍼 풀 *크기*를 겹쳐씁니다.

  또한 <항목>[;<항목>...]을 사용하여 버퍼 풀이 시작될 수 있도록 모든 버퍼 풀 또는 버퍼 풀의 서브세트 *크기*를 임시로 변경할 수 있습니다. 여기서 <항목>=<버퍼 풀 ID>,<페이지 수>입니다.

DB2_PINNED_BP

- 운영 체제: AIX, HP-UX, Linux
- 디폴트=NO, 값: YES 또는 NO
- 이 변수를 YES로 설정하면 운영 체제가 DB2의 데이터베이스 공유 메모리를 고정하도록 DB2가 요청하는 원인이 됩니다. 데이터베이스 공유 메모리를 고정하도록 DB2를 구성하는 경우, 운영 체제가 메모리 관리의 유연성을 줄이므로 시스템이 오버커미트되지 않았는지 주의해서 확인해야 합니다.

  Linux의 경우, 이 레지스트리 변수 수정에 더하여 *libcap.so.1* 라이브러리도 필요합니다.

  이 변수를 YES로 설정하면 데이터베이스 공유 메모리의 자체 성능 조정을 사용할 수 없습니다. 데이터베이스 공유 메모리의 자체 성능 조정은 *database_memory* 구성 매개변수를 AUTOMATIC으로 설정하여 활성화됩니다.

  64비트 환경의 HP-UX의 경우, 이 레지스트리 변수 수정과 함께 DB2 인스턴스 그룹에 MLOCK 특권을 지정해야 합니다. 이를 수행하기 위해 루트 액세스 권한이 있는 사용자가 다음 조치를 수행합니다.

  1. DB2 인스턴스 그룹을 `/etc/privgroup` 파일에 추가합니다. 예를 들어, DB2 인스턴스 그룹이 `db2iadm1` 그룹에 속하면 `/etc/privgroup` 파일에 다음 라인을 추가해야 합니다.

     ```
     db2iadm1 MLOCK
     ```

  2. 다음 명령을 발행하십시오.

     ```
     setprivgrp -f /etc/privgroup
     ```

DB2PRIORITIES

- 운영 체제: 모두
- 값 설정은 플랫폼에 따라 다름
- DB2 프로세스 및 스레드의 우선순위를 제어합니다.

주: DB2PRIORITIES는 사용되지 않으며 서비스 권장사항에서만 사용되어야 합니다. 에이전트 우선순위 및 프리페치 우선순위를 조정하려면 DB2 서비스 클래스를 사용하십시오.

DB2_RESOURCE_POLICY

- 운영 체제: AIX 5 이상, zSeries를 제외한 모든 Linux(32비트), Windows Server 2003 이상
- 디폴트=설정되지 않음, 값: 구성 파일의 유효한 경로
- DB2 데이터베이스에서 사용하는 운영 체제 자원을 제한하는 데 사용할 수 있는 자원 규정을 정의하거나 특정 운영 체제 자원을 특정 DB2 데이터베이스 오브젝트에 할당하는 규칙을 포함합니다. 예를 들어, AIX, Linux 또는

Windows 운영 체제에서 이 레지스트리 변수는 DB2 데이터베이스 시스템이 사용하는 프로세서 세트를 제한하는 데 사용할 수 있습니다. 자원 제어의 범위는 운영 체제에 따라 다릅니다.

AIX NUMA 및 Linux NUMA 사용 가능 머신에서는 DB2 데이터베이스 시스템이 사용하는 자원 세트를 지정하는 규정을 정의할 수 있습니다. 자원 세트 바인딩이 사용되는 경우, 각 개별 DB2 프로세스는 특정 자원 세트에 바인드됩니다. 이는 일부 성능 조정 시나리오에서 유용할 수 있습니다.

이 레지스트리 변수를 설정하여 DB2 프로세스를 운영 체제 자원에 바인딩하기 위한 규정을 정의하는 구성 파일의 경로를 표시할 수 있습니다. 자원 규정을 사용하여 DB2 데이터베이스 시스템을 제한하는 운영 체제 자원 세트를 지정할 수 있습니다. 각 DB2 프로세스는 세트의 단일 자원에 바인드됩니다. 자원 지정은 순환 라운드 로빈 방식으로 발생합니다.

샘플 구성 파일은 다음과 같습니다.

예 1: 모든 DB2 프로세스를 CPU 1 또는 3에 바인드하십시오.

```
<RESOURCE_POLICY>
  <GLOBAL_RESOURCE_POLICY>
   <METHOD>CPU</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>1</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>3</RESOURCE>
   </RESOURCE_BINDING>
  </GLOBAL_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

예 2: (AIX 전용) DB2 프로세스를 자원 세트 sys/node.03.00000, sys/node.03.00001,
sys/node.03.00002 및 sys/node.03.00003 중 하나에 바인드하십시오.

```
<RESOURCE_POLICY>
  <GLOBAL_RESOURCE_POLICY>
   <METHOD>RSET</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00000</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00001</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00002</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00003</RESOURCE>
   </RESOURCE_BINDING>
  </GLOBAL_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

메모: AIX 전용의 경우, RSET 메소드를 사용하려면 CAP_NUMA_ATTACH 기능이 필요합니다.

예 3: (Linux 전용) SAMPLE 데이터베이스와 연관된 버퍼 풀 ID 2 및 3의 모든 메모리를 NUMA 노드 3에 바인드하십시오. 또한 NUMA 노드 3에 바인드하는 데 전체 데이터베이스 메모리의 80퍼센트를 사용하고 20퍼센트는 버퍼 풀 특정이 아닌 메모리의 모든 노드에 걸쳐서 스트라이프되도록 남겨 두십시오.

```
<RESOURCE_POLICY>
  <DATABASE_RESOURCE_POLICY>
   <DBNAME>sample</DBNAME>
   <METHOD>NODEMASK</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>3</RESOURCE>
    <DBMEM_PERCENTAGE>80</DBMEM_PERCENTAGE>
    <BUFFERPOOL_BINDING>
     <BUFFERPOOL_ID>2</BUFFERPOOL_ID>
     <BUFFERPOOL_ID>3</BUFFERPOOL_ID>
    </BUFFERPOOL_BINDING>
   </RESOURCE_BINDING>
  </DATABASE_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

예 4: (Linux 및 Windows 전용) CPU 마스크 0x0F 및 0xF0으로 지정한 두 가지 구별 프로세서 세트를 정의하십시오. DB2 프로세스 및 버퍼 풀 ID 2를 프로세서 세트 0x0F에 바인드하고 DB2 프로세스 및 버퍼 풀 ID 3을 프로세서 세트 0xF0에 바인드하십시오. 각 프로세서 세트에 대해 전체 데이터베이스 메모리의 50퍼센트를 바인딩에 사용하십시오.

이 자원 규정은 프로세서와 NUMA 노드 간의 맵핑을 원하는 경우에 유용합니다. 이러한 시나리오의 예는 8개의 프로세서와 2개의 NUMA 노드가 있으며 프로세서 0 - 3은 NUMA 노드 0에 속하고 프로세서 4 - 7은 NUMA 노드 1에 속하는 시스템입니다. 이 자원 규정은 메모리 집약성(즉, CPU 메소드 및 NODEMASK 메소드의 하이브리드)을 내재적으로 유지하는 동안 프로세서 바인딩을 허용합니다.

```
 <RESOURECE_POLICY>
   <DATABASE_RESOURCE_POLICY>
     <DBNAME>sample</DBNAME>
     <METHOD>CPUMASK</METHOD>
     <RESOURCE_BINDING>
       <RESOURCE>0x0F</RESOURCE>
       <DBMEM_PERCENTAGE>50</DBMEM_PERCENTAGE>
       <BUFFERPOOL_BINDING>
         <BUFFERPOOL_ID>2</BUFFERPOOL_ID>
       </BUFFERPOOL_BINDING>
     </RESOURCE_BINDING>
     <RESOURCE_BINDING>
       <RESOURCE>0x0F</RESOURCE>
       <DBMEM_PERCENTAGE>50</DBMEM_PERCENTAGE>
```

```
      <BUFFERPOOL_BINDING>
        <BUFFERPOOL_ID>3</BUFFERPOOL_ID>
      </BUFFERPOOL_BINDING>
    </RESOURCE_BINDING>
  </DATABASE_RESOURCE_POLICY>
</RESOURCE_POLICY>
```

주: RSET 메소드를 사용하려면 CAP_NUMA_ATTACH 기능이 필요하며 Linux에서는 지원되지 않습니다.

DB2_RESOURCE_POLICY 레지스트리 변수에 의해 지정된 구성 파일은 SCHEDULING_POLICY 요소를 승인합니다. 일부 플랫폼에서는 SCHEDULING_POLICY를 사용하여 다음을 선택할 수 있습니다.

– DB2 서버에서 사용되는 운영 체제 스케줄링 규정

  DB2NTPRICLASS 레지스트리 변수를 사용하여 AIX의 DB2 및 Windows의 DB2에 대한 운영 체제 스케줄링 규정을 설정할 수 있습니다.

– 개별 DB2 서버 에이전트에서 사용되는 운영 체제 우선순위

또는 레지스트리 변수 **DB2PRIORITIES** 및 **DB2NTPRICLASS**를 사용하여 운영 체제 스케줄링 규정을 제어하고 DB2 에이전트 우선순위를 설정할 수 있습니다. 그러나 자원 규정 구성 파일에 있는 SCHEDULING_POLICY 요소의 스펙은 스케줄링 규정 및 연관된 에이전트 우선순위를 모두 지정하는 단일 위치를 제공합니다.

예 1: db2 로그 기록기 및 판독기 프로세스에 대한 우선순위 높임을 포함한 AIX SCHED_FIFO2 스케줄링의 선택.

```
<RESOURCE_POLICY>
 <SCHEDULING_POLICY>
  <POLICY_TYPE>SCHED_FIFO2</POLICY_TYPE>
  <PRIORITY_VALUE>60</PRIORITY_VALUE>

  <EDU_PRIORITY>
   <EDU_NAME>db2loggr</EDU_NAME>
   <PRIORITY_VALUE>56</PRIORITY_VALUE>
  </EDU_PRIORITY>

  <EDU_PRIORITY>
   <EDU_NAME>db2loggw</EDU_NAME>
   <PRIORITY_VALUE>56</PRIORITY_VALUE>
  </EDU_PRIORITY>
 </SCHEDULING_POLICY>
</RESOURCE_POLICY>
```

예 2: Windows에서 DB2NTPRICLASS=H 교체.

```
<RESOURCE_POLICY>
 <SCHEDULING_POLICY>
  <POLICY_TYPE>HIGH_PRIORITY_CLASS</POLICY_TYPE>
 </SCHEDULING_POLICY>
</RESOURCE_POLICY>
```

DB2_SET_MAX_CONTAINER_SIZE

- 운영 체제: 모두
- 디폴트=설정되지 않음, 값: -1, 65,536바이트보다 큰 임의의 양의 정수
- 이 레지스트리 변수를 사용하여 AutoResize 기능이 사용 가능한 자동 스토리지 테이블 스페이스에 대한 개별 컨테이너 크기를 제한할 수 있습니다.

  주: **DB2_SET_MAX_CONTAINER_SIZE**를 바이트, KB 또는 MB로 지정할 수 있지만 db2set은 해당 값을 바이트로 표시합니다.
- 값이 -1로 설정된 경우에는 컨테이너 크기의 한계가 없습니다.

DB2_SKIPDELETED

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- 사용 가능한 경우, 이 변수를 사용하면 커서 안정성 또는 읽기 안정성 분리 레벨을 사용하는 명령문이 무조건 인덱스 액세스 중에는 삭제된 키를 건너뛰고 테이블 액세스 중에는 삭제된 행을 건너뛸 수 있습니다. **DB2_EVALUNCOMMITTED**가 사용 가능한 경우, 삭제된 행은 자동으로 건너뛰지만 인덱스의 언커미트된 의사(pseudo) 삭제된 키는 **DB2_SKIPDELETED**도 사용 가능한 경우 외에는 건너뛰지 않습니다.

  DB2_SKIPDELETED는 현재 커미트된 시맨틱이 잠금 경합을 방지하는 데 도움이 되지 않는 경우에만 적용 가능합니다. 이 변수가 설정되고 현재 커미트된 스캔에 적용 가능하면 삭제된 행을 건너뛰지 않습니다. 대신에 현재 커미트된 버전이 처리됩니다.

  이 레지스트리 변수는 DB2 카탈로그 테이블의 커서 동작에 영향을 미치지 않습니다.

  이 레지스트리 변수는 db2start 명령으로 활성화됩니다.

DB2_SKIPINSERTED

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- **DB2_SKIPINSERTED** 레지스트리 변수가 사용 가능한 경우, 이 변수를 사용하면 커서 안정성 또는 읽기 안정성 분리 레벨을 사용하는 명령문이 언커미트된 삽입된 행을 삽입되지 않은 것처럼 건너뛸 수 있습니다. 이 레지스트리 변수는 DB2 카탈로그 테이블의 커서 동작에 영향을 미치지 않습니다. 이

레지스트리 변수는 데이터베이스 시작 시 활성화되는 반면에 언커미트된 삽입된 행을 건너뛰는 결정은 명령문 컴파일 또는 바인드 시간에 수행됩니다.

현재 커미트된 시맨틱이 사용 중인 경우 이 레지스트리 변수는 효력이 없습니다. 즉, **DB2_SKIPINSERTED**가 OFF로 설정되었고 현재 커미트된 동작이 사용 가능한 경우에도 언커미트된 삽입된 행은 계속 건너뜁니다.

주: 삽입된 행 건너뛰기 동작은 보류 중인 롤아웃 정리가 있는 테이블과 호환되지 않습니다. 그 결과, 스캐너는 RID에서 잠금을 대기하다가 RID가 롤아웃된 블록의 일부임을 발견하게 됩니다.

DB2_SMS_TRUNC_TMPTABLE_THRESH

- 운영 체제: 모두
- 디폴트=0, 값: -1, 0-n, 여기서 n=SMS 테이블 스페이스 컨테이너에서 유지되는 임시 테이블당 Extent 수
- 이 변수는 임시 테이블을 나타내는 파일이 SMS 테이블 스페이스에서 유지되는 최소 파일 크기 임계값을 지정합니다.

  디폴트로 이 변수는 0으로 설정되며 이는 특수 임계값 조절이 수행되지 않음을 의미합니다. 대신에 임시 테이블이 더 이상 필요하지 않으면 해당 파일은 0 Extent로 절단됩니다.

  이 변수의 값이 0보다 크면 더 큰 파일이 유지됩니다. 이는 임시 테이블이 사용될 때마다 파일 삭제 및 재작성과 관련된 시스템 오버헤드를 일부 줄입니다.

  이 변수가 -1로 설정된 경우, 파일은 절단되지 않고 무한정 커질 수 있으며 시스템 자원에 의해서만 제한됩니다.

DB2_SORT_AFTER_TQ

- 운영 체제: 모두
- 디폴트=NO, 값: YES 또는 NO
- 수신 종료 시 데이터를 정렬해야 하고 수신 노드 수와 보내기 노드 수가 같을 때 파티션된 데이터베이스 환경에서 옵티마이저가 방향지정된 테이블 큐에 대해 작업하는 방법을 지정합니다.

  DB2_SORT_AFTER_TQ=NO로 설정되면 옵티마이저는 보내기 종료 시 정렬하고 수신 종료 시 행을 병합하는 경향이 있습니다.

  DB2_SORT_AFTER_TQ=YES로 설정되면 옵티마이저는 수신 종료 시 정렬되지 않은 행을 전송하고 병합하지 않고 모든 행을 수신한 후 수신 종료 시 행을 정렬하는 경향이 있습니다.

DB2_SELUDI_COMM_BUFFER

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- 이 변수는 UPDATE, INSERT 또는 DELETE(UDI) 쿼리에서 SELECT에 대한 블로킹 커서를 처리하는 동안 사용됩니다. 사용 가능한 경우, 이 레지스트리 변수는 쿼리 결과가 임시 테이블에 저장되지 않도록 합니다. 대신에 UDI 쿼리에서 SELECT에 대한 블로킹 커서의 OPEN 처리 중에 DB2 데이터베이스 시스템은 쿼리의 전체 결과를 통신 버퍼 메모리 영역에 직접 버퍼하려고 시도합니다.

  주: 통신 버퍼 스페이스가 전체 쿼리 결과를 보유할 만큼 크지 않은 경우, SQLCODE -906 오류가 발행되고 트랜잭션은 롤백됩니다. 로컬 및 리모트 응용프로그램 각각에 대한 통신 버퍼 메모리 영역의 크기 조정에 대한 정보는 *aslheapsz* 및 *rqrioblk* 데이터베이스 관리 프로그램 구성 매개변수를 참조하십시오.

  이 레지스트리 변수는 파티션된 데이터베이스 환경이나 또는 파티션 내 병렬처리가 사용 가능한 경우에 지원되지 않습니다.

DB2_TRUSTED_BINDIN

- 운영 체제: 모두
- 디폴트=OFF, 값: OFF, ON 또는 CHECK
- **DB2_TRUSTED_BINDIN**이 사용 가능한 경우, 이 변수는 분리되지 않은 임베디드(embedded) 스토어드 프로시저 내에 호스트 변수를 포함하는 쿼리 명령문의 실행 속도를 높입니다.

  이 변수가 사용 가능한 경우, 분리되지 않은 임베디드 스토어드 프로시저 내에 포함된 SQL 및 XQuery문을 바인드하는 동안 외부 SQLDA 형식에서 내부 DB2 형식으로의 변환이 발생하지 않습니다. 이는 Embedded SQL 및 XQuery문의 처리 속도를 높입니다.

  이 변수가 사용 가능한 경우 다음 데이터 유형은 분리되지 않은 임베디드 스토어드 프로시저에서 지원되지 않습니다.

  – SQL_TYP_DATE
  – SQL_TYP_TIME
  – SQL_TYP_STAMP
  – SQL_TYP_CGSTR
  – SQL_TYP_BLOB
  – SQL_TYP_CLOB
  – SQL_TYP_DBCLOB
  – SQL_TYP_CSTR

- SQL_TYP_LSTR
- SQL_TYP_BLOB_LOCATOR
- SQL_TYP_CLOB_LOCATOR
- SQL_TYP_DCLOB_LOCATOR
- SQL_TYP_BLOB_FILE
- SQL_TYP_CLOB_FILE
- SQL_TYP_DCLOB_FILE
- SQL_TYP_BLOB_FILE_OBSOLETE
- SQL_TYP_CLOB_FILE_OBSOLETE
- SQL_TYP_DCLOB_FILE_OBSOLETE

이러한 데이터 유형이 발견되면 SQLCODE -804, SQLSTATE 07002 오류가 리턴됩니다.

주: 입력 호스트 변수의 데이터 유형 및 길이는 해당 요소의 내부 데이터 유형 및 길이와 정확히 일치해야 합니다. 호스트 변수의 경우, 이 요구사항은 항상 충족됩니다. 그러나 매개변수 표시문자의 경우에는 일치하는 데이터 유형이 사용되는지 주의해서 확인해야 합니다. CHECK 옵션을 사용하여 모든 입력 호스트 변수에 대해 데이터 유형 및 길이가 일치하는지 확인할 수 있지만 이 옵션은 대부분의 성능 향상을 무효화합니다.

주: DB2_TRUSTED_BINDIN은 사용되지 않으며 추후 릴리스에서는 제거됩니다.

DB2_USE_ALTERNATE_PAGE_CLEANING

- 운영 체제: 모두
- 디폴트=설정되지 않음, 값: ON 또는 OFF
- 이 변수는 DB2 데이터베이스가 대체 페이지 정리 알고리즘 메소드를 사용하는지 또는 디폴트 페이지 정리 메소드를 사용하는지를 지정합니다. 이 변수가 ON으로 설정되면 DB2 시스템이 변경된 페이지를 디스크에 기록하여 LSN_GAP의 어헤드를 보존하고 혁신적으로 희생(victim)을 찾습니다. 이를 수행하면 페이지 클리너가 사용 가능한 디스크 입출력 대역폭을 더 잘 사용할 수 있습니다. 이 변수가 ON으로 설정되면 *chngpgs_thresh* 데이터베이스 구성 매개변수는 페이지 클리너 활동을 제어하지 않으므로 더 이상 관련되지 않습니다.

DB2_USE_IOCP

- 운영 체제: AIX 5.3 TL9 SP2 또는 AIX 6.1 TL2
- 디폴트: ON 값: OFF 또는 ON

- 이 변수를 사용하면 비동기 I/O(AIO) 요청을 제출하고 수집할 때 AIX I/O 완료 포트(IOCP)를 사용할 수 있습니다. 이 기능은 리모트 메모리 액세스를 방지하여 NUMA(Non-Uniform Memory Access) 환경에서 성능을 향상하는 데 사용됩니다.

기타 변수

DB2ADMINSERVER

- 운영 체제: Windows 및 UNIX
- 디폴트: NULL
- DB2 Administration Server를 지정합니다.

DB2_ATS_ENABLE

- 운영 체제: 모두
- 디폴트: NULL, 값: YES/TRUE/ON/1 또는 NO/FALSE/OFF/0
- 이 변수는 관리 태스크 스케줄러가 실행 중인지 여부를 제어합니다. 관리 태스크 스케줄러는 디폴트로 사용하지 않습니다. 스케줄러를 사용하지 않는 경우, 내장 프로시저 및 보기를 사용하여 태스크를 정의하고 수정할 수 있지만 스케줄러는 태스크를 실행하지 않습니다.

DB2AUTH

- 운영 체제: 모두
- 디폴트: 설정되지 않음. 값: TRUSTEDCLIENT_SRVRENC, TRUSTEDCLIENT_DATAENC, DISABLE_CHGPASS, OSAUTHDB
- 이 변수를 사용하여 사용자 인증 동작을 조정할 수 있습니다.
  - TRUSTEDCLIENT_SRVRENC: 이 값은 트러스트되지 않은 클라이언트가 SERVER_ENCRYPT를 사용하도록 강제합니다.
  - TRUSTEDCLIENT_DATAENC: 이 값은 트러스트되지 않은 클라이언트가 DATA_ENCRYPT를 사용하도록 강제합니다.
  - DISABLE_CHGPASS: 이 값은 클라이언트의 암호 변경 기능을 사용하지 않도록 설정합니다.
  - OSAUTHDB: 이 값은 DB2 데이터베이스 관리 프로그램이 AIX 운영 체제의 사용자에 대한 인증 및 그룹 설정을 사용하도록 지시합니다. LDAP 서버는 다음 중 하나일 수 있습니다.
    - IBM Tivoli Directory Server(ITDS)
    - Microsoft Active Directory(MSAD)
    - Sun One Directory Server

DB2CLIINIPATH

- 운영 체제: 모두

- 디폴트: NULL
- DB2 CLI/ODBC 구성 파일의 디폴트 경로(db2cli.ini)를 겹쳐쓰고 클라이언트에 다른 위치를 지정하는 데 사용됩니다. 지정된 값은 클라이언트 시스템에서 유효한 경로여야 합니다.

DB2_COMMIT_ON_EXIT

- 운영 체제: UNIX
- 디폴트: OFF, 값: OFF/NO/0 또는 ON/YES/1
- UNIX 운영 체제에서 DB2 UDB 버전 8 이전에 DB2는 성공적인 응용프로그램 종료 시 남아 있는 모든 인플라이트(inflight) 트랜잭션을 커미트했습니다. DB2 UDB 버전 8에서는 종료 시 인플라이트 트랜잭션이 롤백되도록 동작이 변경되었습니다. 이 레지스트리 변수를 사용하여 이전 동작에 따라 다른 Embedded SQL 응용프로그램을 사용하는 사용자가 DB2 버전 9에서 이를 계속 사용할 수 있습니다. 이 레지스트리 변수는 JDBC, CLI 및 ODBC 응용프로그램에 영향을 미치지 않습니다.

  이 레지스트리 변수는 사용되지 않으며 추후 릴리스에서는 종료 시 커미트 동작이 더 이상 지원되지 않습니다. 사용자는 DB2 버전 9 이전에 개발된 응용프로그램이 계속 이 기능에 종속되는지를 판별하고 필요에 따라 응용프로그램에 적당한 명시적 COMMIT 또는 ROLLBACK문을 추가해야 합니다. 레지스트리 변수가 켜져 있으면 종료 전에는 명시적으로 커미트할 수 없는 새 응용프로그램을 구현하지 않도록 주의해야 합니다.

  대부분의 사용자는 이 레지스트리 변수를 디폴트 설정으로 두어야 합니다.

DB2CONNECT_DISCONNECT_ON_INTERRUPT

- 운영 체제: 모두
- 디폴트: NO, 값: YES/TRUE/1 또는 NO/FALSE/0
- YES(TRUE 또는 1)로 설정되면 이 변수는 인터럽트가 발생하면 버전 8(또는 그 이상) DB2 Universal Database z/OS 서버에 대한 연결이 즉시 끊어지도록 지정합니다. 다음과 같은 구성에 이 변수를 사용할 수 있습니다.
  - 버전 8(또는 그 이상) DB2 UDB z/OS 서버와 함께 DB2 클라이언트를 실행 중인 경우, 클라이언트에서 **DB2CONNECT_DISCONNECT_ON_INTERRUPT**를 YES로 설정하십시오.
  - 버전 8(또는 그 이상) DB2 UDB z/OS 서버에 대한 DB2 Connect 게이트웨이를 통해 DB2 클라이언트를 실행 중인 경우, 게이트웨이에서 **DB2CONNECT_DISCONNECT_ON_INTERRUPT**를 YES로 설정하십시오.

DB2_CREATE_DB_ON_PATH

- 운영 체제: Windows
- 디폴트: NULL, 값: YES 또는 NO
- 경로(및 드라이브)를 데이터베이스 경로로 사용하기 위한 지원을 사용하려면 이 레지스트리 변수를 YES로 설정하십시오.

  DB2_CREATE_DB_ON_PATH 설정은 데이터베이스가 작성될 때, 데이터베이스 관리 프로그램 구성 매개변수 **dftdbpath**가 설정될 때 및 데이터베이스가 리스토어될 때 점검됩니다. 완전한 데이터베이스 경로의 길이는 최대 215자입니다.

  DB2_CREATE_DB_ON_PATH는 설정하지 않고(또는 NO로 설정) 데이터베이스를 작성하거나 리스토어할 때 데이터베이스 경로에 경로를 지정하면 SQL1052N 오류가 리턴됩니다.

  DB2_CREATE_DB_ON_PATH는 설정하지 않고(또는 NO로 설정) **dftdbpath** 데이터베이스 관리 프로그램 구성 매개변수를 갱신하면 SQL5136N 오류가 리턴됩니다.

  주의:
  경로 지원을 사용하여 새 데이터베이스를 작성하는 경우, db2DbDirGetNextEntry() API 또는 해당 API의 이전 버전을 사용하여 DB2 버전 9.1 이전에 작성된 응용프로그램은 올바르게 작동하지 않습니다. 여러 가지 시나리오 및 적절한 조치 과정에 대한 자세한 내용은 http://www.ibm.com/software/data/db2/support/db2_9/를 참조하십시오.

DB2_DDL_SOFT_INVAL

- 운영 체제: 모두
- 디폴트: ON, 값: ON 또는 OFF
- 적용 가능한 데이터베이스 오브젝트가 삭제 또는 변경될 때 소프트 무효화를 사용 가능하게 합니다.

  DB2_DDL_SOFT_INVAL이 ON으로 설정된 경우, 동일한 오브젝트를 참조하는 트랜잭션이 완료되기를 기다리지 않고 DDL 조작(예: 삭제(drop), 변경 또는 접속 해제)을 시작할 수 있습니다. 오브젝트에 종속된 현재 실행은 원래 오브젝트 정의를 사용하여 계속 진행되는 반면, 새 실행은 변경된 오브젝트를 사용합니다. 이는 DDL문 발행 시 동시성을 높입니다.

  주: 새 소프트 무효화 성능은 동적 패키지에만 적용됩니다. 정적 패키지가 있는 오브젝트는 여전히 하드 무효화를 필요로 합니다.

DB2DEFPREP

- 운영 체제: 모두
- 디폴트: NO, 값: ALL, YES 또는 NO

- 이 옵션을 사용할 수 있기 전에 프리컴파일된 응용프로그램에 대해 **DEFERRED_PREPARE** 프리컴파일 옵션의 런타임 동작을 가상합니다. 예를 들어, DB2 v2.1.1 또는 이전 응용프로그램이 DB2 v2.1.2 또는 이후 환경에서 실행된 경우, **DB2DEFPREP**를 사용하여 원하는 『지연된 준비』 동작을 표시할 수 있습니다.

주: DB2DEFPREP는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 이 변수는 **DEFERRED_PREPARE** 프리컴파일 옵션이 사용 불가능한 DB2 이전 버전을 사용하는 사용자에게만 필요합니다.

DB2_DISABLE_FLUSH_LOG

- 운영 체제: 모두
- 디폴트: OFF, 값: ON 또는 OFF
- 온라인 백업이 완료되었을 때 사용 중인 로그 파일 닫기를 사용하지 않는지 여부를 지정합니다.

  온라인 백업이 완료되면 마지막 사용 중인 로그 파일이 절단되고 닫히고 아카이브될 수 있습니다. 이는 온라인 백업이 복구에 사용할 수 있는 완전한 아카이브 로그 세트를 갖도록 보장합니다. 로그 시퀀스 번호(LSN) 스페이스의 분할 영역을 낭비하는 것을 걱정하는 경우, 마지막 사용 중인 로그 파일 닫기를 사용 불가능하게 할 수 있습니다. 사용 중인 로그 파일이 절단될 때마다 절단된 스페이스에 비례하는 양만큼 LSN이 증가됩니다. 매일 여러 번 온라인 백업을 수행하는 경우, 마지막 사용 중인 로그 파일 닫기를 사용 불가능하게 할 수 있습니다.

  또한 온라인 백업이 완료되자마자 로그가 가득 찬 메시지를 검색하는 경우에도 마지막 사용 중인 로그 파일 닫기를 사용 불가능하게 할 수 있습니다. 로그 파일이 절단되면 절단된 로그 크기에 비례하는 양만큼 예약된 사용 중인 로그 스페이스가 증가됩니다. 절단된 로그 파일이 재개되면 사용 중인 로그 스페이스가 해제됩니다. 재개는 로그 파일이 비활성화된 후 곧 발생합니다. 이러한 두 이벤트 사이의 짧은 시간 간격 동안 로그가 가득 찬 메시지를 수신할 수도 있습니다.

  백업이 로그를 포함하려면 사용 중인 로그 파일을 절단하고 닫아야 하기 때문에 로그를 포함하는 백업 중에 이 레지스트리 변수는 무시됩니다.

DB2_DISPATCHER_PEEKTIMEOUT

- 운영 체제: 모두
- 디폴트: 1, 값: 0 - 32,767초; 0은 곧 시간종료임을 표시함
- **DB2_DISPATCHER_PEEKTIMEOUT**을 사용하여 클라이언트를 에이전트에 전달하기 전에 디스패처가 클라이언트의 연결 요청을 대기하는 시간(초)을 조정할 수 있습니다. 대부분의 경우, 이 레지스트리 변수를 조정할 필요

가 없습니다. 이 레지스트리 변수는 사용 가능한 DB2 Connect 연결 집중기(connection concentrator)를 가지고 있는 인스턴스에만 영향을 미칩니다.

이 레지스트리 변수와 **DB2_SERVER_CONTIMEOUT** 레지스트리 변수는 모두 연결 시간 동안 새 클라이언트의 처리를 구성합니다. 느린 클라이언트가 인스턴스에 여러 개 연결된 경우, 각 클라이언트를 시간종료시키기 위해 디스패처는 최대 1초 동안 보류되며 이는 많은 클라이언트가 동시에 연결되어 있는 경우에 디스패처가 병목이 되는 원인이 됩니다. 여러 개의 활성 데이터베이스를 포함한 인스턴스에서 매우 느린 연결 시간이 발생하는 경우, **DB2_DISPATCHER_PEEKTIMEOUT**은 0까지 내려갑니다. **DB2_DISPATCHER_PEEKTIMEOUT**을 낮추면 디스패처는 클라이언트의 연결 요청이 이미 있는 경우에 해당 요청을 살펴보기만 합니다. 디스패처는 연결 요청이 도착할 때까지 기다리지 않습니다. 유효하지 않은 값이 설정된 경우, 디폴트값이 사용됩니다. 이 레지스트리 변수는 동적이 아닙니다.

DB2_DJ_INI

- 운영 체제: 모두
- 디폴트:
  - UNIX: *db2_instance_directory*/cfg/db2dj.ini
  - Windows: *db2_install_directory*₩cfg₩db2dj.ini
- 페더레이션 구성 파일의 절대 경로 이름을 지정합니다(예: `db2set DB2_DJ_INI=$HOME/sqllib/cfg/my_db2dj.ini`). 이 파일은 데이터 소스 환경 변수에 대한 설정을 포함합니다. 이러한 환경 변수는 Informix® 랩퍼 및 InfoSphere™ Federation Server에서 제공하는 랩퍼에서 사용됩니다.

  샘플 페더레이션 구성 파일은 다음과 같습니다.

```
INFORMIXDIR=/informix/client_sdk
INFORMIXSERVER=inf93
ORACLE_HOME=/usr/oracle9i
SYBASE=/sybase/V12
SYBASE_OCS=OCS-12_5
```

  다음 제한사항이 db2dj.ini 파일에 적용됩니다.
  - 항목은 *evname=value* 형식을 따라야 합니다. 여기서 *evname*은 환경 변수 이름이고 *value*는 해당 값입니다.
  - 환경 변수 이름의 최대 길이는 255바이트입니다.
  - 환경 변수 값의 최대 길이는 765바이트입니다.

  이 변수는 데이터베이스 관리 프로그램 매개변수 **federated**가 YES로 설정된 경우 외에는 무시됩니다.

DB2DMNBCKCTLR

- 운영 체제: Windows
- 디폴트: NULL, 값: ? 또는 도메인 이름
- DB2 서버가 백업 도메인 제어기인 도메인의 이름을 아는 경우, **DB2DMNBCKCTLR=***DOMAIN_NAME*을 설정하십시오. *DOMAIN_NAME*은 대문자여야 합니다. DB2가 로컬 머신이 백업 도메인 제어기인 도메인을 판별하도록 하려면 **DB2DMNBCKCTLR=?**를 설정하십시오. **DB2DMNBCKCTLR** 프로파일 변수가 설정되지 않거나 공백으로 설정된 경우, DB2는 기본 도메인 제어기에서 인증을 수행합니다.

  주: 백업 도메인 제어기는 기본 도메인 제어기와의 동기화에서 이탈할 수 있고 이는 보안 노출의 원인이 되므로 DB2는 디폴트로 기존의 백업 도메인 제어기를 사용하지 않습니다. 동기화 이탈은 기본 도메인 제어기의 보안 데이터베이스가 갱신되지만 변경사항이 백업 도메인 제어기에 전달되지 않는 경우에 발생할 수 있습니다. 이는 네트워크 대기 시간이 있거나 컴퓨터 브라우저 서비스가 작동하지 않는 경우에 발생할 수 있습니다.

주: DB2DMNBCKCTLR은 사용되지 않으며 추후 릴리스에서는 제거됩니다. Active Directory에는 더 이상 백업 도메인 제어기가 없으므로 이 변수는 더 이상 필요하지 않습니다.

DB2_DOCHOST

- 운영 체제: 모두
- 디폴트: 설정되지 않음(그러나 DB2는 계속 IBM 웹 사이트 publib.boulder.ibm.com/infocenter/db2luw/v9r7에서 정보 센터에 액세스를 시도함), 값: http://*hostname* 여기서 *hostname*= 유효한 호스트 이름 또는 IP 주소
- *DB2 정보 센터*가 설치된 호스트 이름을 지정합니다. 이 변수는 DB2 설치 마법사에서 자동 구성 옵션이 선택된 경우 *DB2 정보 센터* 설치 중에 자동으로 설정될 수 있습니다.

DB2_DOCPORT

- 운영 체제: 모두
- 디폴트: NULL, 값: 임의의 유효한 포트 번호
- DB2 도움말 시스템이 DB2 문서를 제공하는 포트 번호를 지정합니다. 이 변수는 DB2 설치 마법사에서 자동 구성 옵션이 선택된 경우 *DB2 정보 센터* 설치 중에 자동으로 설정될 수 있습니다.

DB2_ENABLE_AUTOCONFIG_DEFAULT

- 운영 체제: 모두
- 디폴트: NULL, 값: YES 또는 NO

- 이 변수는 데이터베이스 작성 시 구성 어드바이저가 자동으로 실행되는지 여부를 제어합니다. **DB2_ENABLE_AUTOCONFIG_DEFAULT**가 설정되지 않은 경우(NULL), 변수가 YES로 설정되고 데이터베이스 작성 시 구성 어드바이저가 실행되는 것과 효과가 동일합니다. 이 변수를 설정한 후 인스턴스를 재시작할 필요가 없습니다. AUTOCONFIGURE 명령을 실행하거나 CREATE DB AUTOCONFIGURE를 실행하는 경우, 해당 명령은 **DB2_ENABLE_AUTOCONFIG_DEFAULT**의 설정값을 겹쳐씁니다.

DB2_ENABLE_LDAP

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- LDAP(Lightweight Directory Access Protocol)이 사용되는지 여부를 지정합니다. LDAP은 디렉토리 서비스에 대한 액세스 메소드입니다.

DB2_EVMON_EVENT_LIST_SIZE

- 운영 체제: 모두
- 디폴트: 0(한계 없음), 값: KB/Kb/kb, MB/Mb/mb 또는 GB/Gb/gb로 지정된 값. 이 변수의 상한은 고정되어 있지 않지만 모니터 힙에서 사용 가능한 메모리 양에 의해 제한됩니다.
- 이 레지스트리 변수는 특정 이벤트 모니터에 기록되기 위해 대기 중인 큐에 대기될 수 있는 최대 바이트 수를 지정합니다. 이 한계에 일단 접근하면 이벤트 모니터 레코드 전송을 시도하는 에이전트는 *큐 크기*가 이 임계값 아래로 떨어질 때까지 대기합니다.

주: 활성 레코드가 모니터 힙에서 할당될 수 없는 경우, 해당 레코드는 삭제됩니다. 이러한 일이 발생하지 않게 하려면 **mon_heap_sz** 구성 매개변수를 AUTOMATIC으로 설정하십시오. **mon_heap_sz**를 특정 값으로 설정한 경우, **DB2_EVMON_EVENT_LIST_SIZE**가 더 작은 값으로 설정되었는지 확인하십시오. 그러나 모니터 힙은 또한 다른 모니터 요소를 트래킹하는 데도 사용되므로 이러한 조치가 활성 레코드가 삭제되지 않도록 보장할 수는 없습니다.

DB2_EVMON_STMT_FILTER

- 운영 체제: 모두
- 디폴트: 설정되지 않음; 값:
  - ALL: 모든 명령문 이벤트 모니터의 출력이 필터링됨을 표시합니다. 이 옵션은 독점입니다.
  - '*nameA nameB nameC*': 여기서 문자열의 각 이름은 레코드가 필터링될 이벤트 모니터의 이름을 나타냅니다. 둘 이상의 이름이 제공되는 경우, 각 이름은 단일 공백으로 구분되어야 합니다. 모든 입력 이름은 DB2에

서 대문자로 작성됩니다. 지정할 수 있는 이벤트 모니터의 최대수는 32입니다. 각 모니터 이름의 길이는 최대 18자입니다.

– '*nameA:op1,op2 nameB:op1,op2 nameC:op1*': 여기서 문자열의 각 이름은 레코드가 필터링될 이벤트 모니터의 이름을 나타냅니다. 각 옵션은 특정 SQL 조작에 맵핑되는 정수값을 나타냅니다(*op1*, *op2* 등). 정수값을 지정하여 사용자는 이벤트 모니터와 이에 적용되는 규칙을 판별할 수 있습니다.

- **DB2_EVMON_STMT_FILTER**를 사용하여 명령문 이벤트 모니터에 의해 기록되는 레코드 수를 줄일 수 있습니다. 이 레지스트리 변수가 설정되면 다음 SQL 조작에 대한 레코드만 지정된 이벤트 모니터에 기록됩니다.

표 65. 이벤트 모니터 출력을 특정 SQL 조작으로 제한하기 위해 **DB2_EVMON_STMT_FILTER***에 사용하는 값*

SQL 조작	맵핑되는 정수값
SELECT	15
EXECUTE	2
EXECUTE_IMMEDIATE	3
CLOSE	6
STATIC COMMIT	8
STATIC ROLLBACK	9
CALL	12
PRE_EXEC	17

기타 모든 조작은 명령문 이벤트 모니터의 출력에 표시되지 않습니다. 레코드가 이벤트 모니터에 기록되는 조작 세트를 사용자 정의하려면 정수값을 사용하십시오.

예 1:

```
db2set DB2_EVMON_STMT_FILTER= 'mon1 monitor3'
```

이 예에서 mon1 및 monitor3 이벤트 모니터는 제한된 응용프로그램 요청 목록에 대한 레코드를 수신합니다. 예를 들어, mon1 명령문 이벤트 모니터에 의해 모니터되는 응용프로그램이 동적 SQL문을 준비하고 해당 명령문을 기반으로 커서를 열고 해당 커서에서 10,000행을 프리페치한 다음 커서 닫기 요청을 발행하면 닫기 요청에 대한 레코드만 mon1 이벤트 모니터 출력에 표시됩니다.

예 2:

```
db2set DB2_EVMON_STMT_FILTER='evmon1:3,8 evmon2:9,15
```

이 예에서 evmon1 및 evmon2는 제한된 응용프로그램 요청 목록에 대한 레코드를 수신합니다. 예를 들어, evmon1 명령문 이벤트 모니터에 의해 모니

터되는 응용프로그램이 작성 명령문을 발행하면 즉시 실행 및 정적 커미트 조작만 evmon1 이벤트 모니터 출력에 표시됩니다. evmon2 명령문 이벤트 모니터에 의해 모니터되는 응용프로그램이 선택 및 정적 롤백을 모두 포함하는 SQL을 수행하면 이 두 가지 조작만 evmon2 이벤트 모니터 출력에 표시됩니다.

주: 데이터베이스 시스템 모니터 상수의 정의에 대해서는 `sqlmon.h` 헤더 파일을 참조하십시오.

DB2_EXTSECURITY

- 운영 체제: Windows
- 디폴트: ON, 값: ON 또는 OFF
- DB2 시스템 파일 잠금을 사용하여 DB2에 대한 권한 부여되지 않은 액세스를 예방합니다. 잠재적인 문제점을 방지하려면 이 레지스트리 변수를 끄지 않아야 합니다.

DB2_FALLBACK

- 운영 체제: Windows
- 디폴트: OFF, 값: ON 또는 OFF
- 이 변수를 사용하여 폴백(fallback) 처리 중에 모든 데이터베이스 연결을 강제로 해제할 수 있습니다. 이 변수는 MCSC(Microsoft Cluster Server)가 있는 Windows 환경에서 장애 복구 지원과 함께 사용됩니다.
  DB2_FALLBACK이 설정되지 않거나 OFF로 설정되고 폴백 중에 데이터베이스 연결이 존재하는 경우, DB2 자원을 오프라인으로 가져올 수 없습니다. 이는 폴백 처리가 실패함을 의미합니다.

DB2_FMP_COMM_HEAPSZ

- 운영 체제: Windows, UNIX
- 디폴트: 20MB 또는 10개의 분리 루틴을 실행하기에 충분한 스페이스 (둘 중 큰 값). AIX의 경우 디폴트는 256MB입니다.
- 이 변수는 4KB 페이지에 분리 루틴 호출(예: 스토어드 프로시저 또는 사용자 정의 함수(UDF) 호출)에 사용되는 풀의 크기를 지정합니다. 각 분리 루틴에서 사용되는 스페이스는 **aslheapsz** 구성 매개변수 값의 두 배입니다.

  사용자 시스템에서 여러 개의 분리 루틴을 실행 중인 경우에는 이 변수의 값을 늘려야 합니다. 아주 적은 수의 분리 루틴을 실행 중인 경우에는 변수의 값을 줄일 수 있습니다.

  이 값을 0으로 설정하면 설정이 작성되지 않고 그 결과 분리 루틴을 호출할 수 없습니다. 이는 또한 Health Monitor와 자동 데이터베이스 유지보수 기

능(예: 자동 백업, 통계 콜렉션 및 REORG)이 사용 불가능하게 됨을 의미합니다. 이는 이 기능이 분리 루틴 인프라스트럭처에 종속되어 있기 때문입니다.

DB2_GRP_LOOKUP

- 운영 체제: Windows
- 디폴트: NULL, 값: LOCAL, DOMAIN, TOKEN, TOKENLOCAL, TOKENDOMAIN
- 이 변수는 사용자가 속한 그룹을 열거하는 데 사용되는 Windows 보안 메커니즘을 지정합니다.

DB2_HADR_BUF_SIZE

- 운영 체제: 모두
- 디폴트: 2***logbufsz**
- 이 변수는 로그 페이지 단위의 버퍼 크기를 수신하는 대기 로그를 지정합니다. 설정되지 않으면 DB2는 버퍼 크기를 수신하는 대기의 기본 **logbufsz** 구성 매개변수 값의 두 배를 사용합니다. 이 변수는 대기 인스턴스에 설정해야 합니다. 기본 데이터베이스에서는 무시됩니다.

  HADR 동기화 모드(**hadr_syncmode** 데이터베이스 구성 매개변수)가 ASYNC로 설정된 경우, 피어 상태 동안 느린 대기는 기본에서의 전송 조작을 스톨시키는 원인이 되며 따라서 기본에서의 트랜잭션 처리를 차단합니다. 대기 데이터베이스가 처리되지 않은 로그 데이터를 더 많이 보유할 수 있도록 대기 데이터베이스에서 디폴트 로그 수신 버퍼보다 큰 값을 구성할 수 있습니다. 이로 인해 트랜잭션 처리를 차단하지 않고 짧은 기간에 대기가 로그 데이터를 사용할 수 있는 속도보다 빠르게 기본이 로그 데이터를 생성할 수 있습니다.

DB2_HADR_NO_IP_CHECK

- 운영 체제: 모두
- 디폴트: OFF, 값: ON|OFF
- HADR 연결에 대한 IP 점검 생략 여부를 지정합니다.
- 이 변수는 기본적으로 HADR 연결에 대한 IP 교차 점검을 생략하기 위해 NAT(Network Address Translation) 환경에서 사용됩니다. 이 변수는 HADR 구성의 정상성 점검을 약화시키므로 다른 환경에서는 사용하지 않는 것이 좋습니다. 디폴트로 HADR 연결이 설정되면 로컬 및 리모트 호스트 매개변수의 구성 일관성이 검증됩니다. 호스트 이름은 교차 점검을 위해 IP 주소에 맵핑됩니다. 두 가지 점검이 수행됩니다.
  - 기본의 **HADR_LOCAL_HOST** 매개변수 = 대기의 **HADR_REMOTE_HOST** 매개변수

– 기본의 **HADR_REMOTE_HOST** 매개변수 = 대기의 **HADR_LOCAL_HOST** 매개변수

점검이 실패하면 연결이 닫힙니다.

이 매개변수가 켜져 있으면 IP 점검이 발생하지 않습니다.

DB2_HADR_PEER_WAIT_LIMIT

- 운영 체제: 모두
- 디폴트: **0**(한계 없음을 의미) 값: 0 - 부호 없는 최대 32비트 정수(두 값 포함)
- **DB2_HADR_PEER_WAIT_LIMIT** 레지스트리 변수가 설정되면 대기 데이터베이스에 대한 로그 복제 때문에 기본 데이터베이스의 로깅이 지정된 초수 동안 차단된 경우 HADR 기본 데이터베이스가 피어 상태를 중단합니다. 이 한계에 접근하면 기본 데이터베이스가 대기 데이터베이스에 대한 연결을 끊습니다. 피어 창이 사용 불가능하면 기본 데이터베이스는 연결이 끊긴 상태에 들어가고 로깅이 다시 시작됩니다. 피어 창이 사용 가능하면 기본 데이터베이스는 연결이 끊긴 피어 상태에 들어가고 로깅은 계속 차단됩니다. 재연결 또는 피어 창이 만기되자마자 기본 데이터베이스는 연결이 끊긴 피어 상태를 벗어납니다. 기본 데이터베이스가 연결이 끊긴 피어 상태를 벗어나면 로깅이 다시 시작됩니다. 이 매개변수는 대기 데이터베이스에서는 적용되지 않습니다. 그렇지만 기본 및 대기 데이터베이스 모두에서 같은 값을 사용하는 것이 좋습니다. 유효하지 않은 값(숫자가 아니거나 음수)은 한계가 없음을 뜻하는 ″0″으로 해석됩니다. 이 매개변수는 정적입니다. 이 매개변수를 적용하려면 데이터베이스 인스턴스를 재시작해야 합니다.

DB2_HADR_SORCVBUF

- 운영 체제: 모두
- 디폴트: 운영 체제 TCP 소켓 수신 버퍼 *크기*, 값: 1,024 - 4,294,967,295
- 이 변수는 HADR 연결에 대한 운영 체제(OS) TCP 소켓 수신 버퍼 *크기*를 지정하며 이를 사용하여 사용자는 다른 연결과 구분되게 HADR TCP/IP 동작을 사용자 정의할 수 있습니다. 일부 운영 체제에서는 사용자 지정 값을 자동으로 반올림하거나 상한을 지정합니다. HADR 연결에 사용된 실제 버퍼 *크기*는 db2diag 로그 파일에 로그됩니다. 사용자의 네트워크 트래픽을 기본으로 하는 이 매개변수에 대한 최적의 설정값은 운영 체제 네트워크 조정 안내서를 참조하십시오. 이 변수는 **DB2_HADR_SOSNDBUF**와 함께 사용해야 합니다.

DB2_HADR_SOSNDBUF

- 운영 체제: 모두
- 디폴트: 운영 체제 TCP 소켓 보내기 버퍼 *크기*, 값: 1,024 - 4,294,967,295

- 이 변수는 HADR 연결에 대한 운영 체제(OS) TCP 소켓 보내기 버퍼 크기를 지정하며 이를 사용하여 사용자는 다른 연결과 구분되게 the HADR TCP/IP 동작을 사용자 정의할 수 있습니다. 일부 운영 체제에서는 사용자 지정 값을 자동으로 반올림하거나 상한을 지정합니다. HADR 연결에 사용된 실제 버퍼 크기는 db2diag 로그 파일에 로그됩니다. 사용자의 네트워크 트래픽을 기본으로 하는 이 매개변수에 대한 최적의 설정값은 운영 체제 네트워크 조정 안내서를 참조하십시오. 이 변수는 **DB2_HADR_SORCVBUF**와 함께 사용해야 합니다.

DB2LDAP_BASEDN

- 운영 체제: 모두
- 디폴트: NULL, 값: 임의의 유효한 기본 식별 도메인 이름.
- 이 변수가 설정되면 DB2의 LDAP 오브젝트가 LDAP 디렉토리의 다음 아래에 저장됩니다.

```
CN=System
CN=IBM
CN=DB2
```

  이는 지정된 기본 식별 이름 아래에 있습니다.

  이 변수가 Microsoft Active Directory Server로 설정되며 CN=DB2, CN=IBM 및 CN=System이 이 식별 이름 아래에 정의되어 있는지 확인하십시오.

DB2LDAPCACHE

- 운영 체제: 모두
- 디폴트: YES, 값: YES 또는 NO
- LDAP 캐시가 사용 가능한지 지정합니다. 이 캐시는 로컬 머신에서 데이터베이스, 노드 및 DCS 디렉토리를 카탈로그하는 데 사용됩니다.

  캐시에 최신 항목이 들어 있는지 확인하려면 다음을 수행하십시오.

```
REFRESH LDAP IMMEDIATE ALL
```

  이 명령은 데이터베이스 디렉토리 및 노드 디렉토리에서 올바르지 않은 항목을 갱신하고 제거합니다.

DB2LDAP_CLIENT_PROVIDER

- 운영 체제: Windows
- 디폴트: NULL(사용 가능한 경우 Microsoft가 사용되고 그렇지 않은 경우 IBM이 사용됩니다.) 값: IBM 또는 Microsoft

- Windows 환경에서 실행될 때 DB2는 LDAP 디렉토리에 액세스하는 데 Microsoft LDAP 클라이언트 또는 IBM LDAP 클라이언트 사용을 지원합니다. 이 레지스트리 변수는 DB2에서 사용할 LFAP 클라이언트를 명시적으로 선택하는 데 사용됩니다.

  주: 이 레지스트리 변수의 현재 값을 표시하려면 db2set 명령을 사용하십시오.

  ```
  db2set DB2LDAP_CLIENT_PROVIDER
  ```

DB2LDAPHOST

- 운영 체제: 모두
- 디폴트: NULL, 값: 임의의 유효한 호스트 이름
- LDAP 디렉토리 위치의 호스트 이름을 지정합니다.

DB2LDAP_KEEP_CONNECTION

- 운영 체제: 모두
- 디폴트: YES, 값: YES 또는 NO
- DB2가 해당 내부 LDAP 연결 핸들을 캐시하는지 여부를 지정합니다. 이 변수가 NO로 설정되면 DB2가 Directory Server에 대한 해당 LDAP 연결 핸들을 캐시하지 않습니다. 그 결과 부정적인 성능 영향을 미칠 수 있지만 Directory Server에 대해 동시에 사용 중인 LDAP 클라이언트 연결 수를 최소화해야 하는 경우 **DB2LDAP_KEEP_CONNECTION**을 NO로 설정하는 것이 좋습니다.

  성능을 최대화하기 위해 이 변수는 디폴트로 YES로 설정됩니다.

  DB2LDAP_KEEP_CONNECTION 레지스트리 변수는 LDAP에서 전역 레벨 프로파일 레지스트리 변수로만 구현되므로 다음과 같이 db2set 명령에 **-gl** 옵션을 지정하여 이 변수를 설정해야 합니다.

  ```
  db2set -gl DB2LDAP_KEEP_CONNECTION=NO
  ```

DB2LDAP_SEARCH_SCOPE

- 운영 체제: 모두
- 디폴트: DOMAIN, 값: LOCAL, DOMAIN 또는 GLOBAL
- LDAP(Lightweight Directory Access Protocol)의 도메인 또는 데이터베이스 파티션에서 찾을 수 있는 정보의 검색 범위를 지정합니다. LOCAL은 LDAP 디렉토리 검색을 사용 불가능하게 합니다. DOMAIN은 LDAP에서 현재 디렉토리 파티션만 검색합니다. GLOBAL은 오브젝트를 찾을 때까지 LDAP의 모든 디렉토리 파티션을 검색합니다.

DB2_LOAD_COPY_NO_OVERRIDE

- 운영 체제: 모두

- 디폴트: NONRECOVERABLE, 값: COPY YES 또는 NONRECOVERABLE
- 이 변수는 변수의 값에 따라 모든 LOAD **COPY NO**를 LOAD **COPY YES** 또는 **NONRECOVERABLE**로 변환합니다. 이 변수는 HADR 기본 데이터베이스 및 표준(비HADR) 데이터베이스에 적용 가능합니다. 이 변수는 HADR 대기 데이터베이스에서는 무시됩니다. HADR 기본 데이터베이스에서 이 변수가 설정되지 않으면 LOAD **COPY NO**는 LOAD **NONRECOVERABLE**로 변환됩니다. 이 변수의 값은 **COPY YES**절과 동일한 구문을 사용하여 복사 대상을 지정하거나 복구 불가능한 로드를 지정합니다.

DB2LOADREC

- 운영 체제: 모두
- 디폴트: NULL
- 롤 포워드 중에 로드 사본의 위치를 겹쳐쓰는 데 사용됩니다. 사용자가 로드 사본의 물리적 위치를 변경한 경우, 롤 포워드를 실행하기 전에 **DB2LOADREC**를 설정해야 합니다.

DB2LOCK_TO_RB

- 운영 체제: 모두
- 디폴트: NULL, 값: STATEMENT
- 잠금 시간종료로 인해 전체 트랜잭션이 롤백되는지 또는 현재 명령문만 롤백되는지를 지정합니다. **DB2LOCK_TO_RB**가 STATEMENT로 설정된 경우, 잠금 시간종료는 현재 명령문만 롤백시킵니다. 기타 모든 설정은 트랜잭션 롤백을 일으킵니다.

DB2_MAP_XML_AS_CLOB_FOR_DLC

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- **DB2_MAP_XML_AS_CLOB_FOR_DLC** 레지스트리 변수는 XML을 데이터 유형으로 지원하지 않는 클라이언트(또는 DRDA 응용프로그램 요청자)에 대한 XML 값의 디폴트 DESCRIBE 및 FETCH 동작을 겹쳐쓰는 기능을 제공합니다. 디폴트값은 NO이며 이는 해당 클라이언트에 대해 XML 값의 DESCRIBE는 BLOB(2GB)를 리턴하고 XML 값의 FETCH는 UTF-8의 인코딩을 표시하는 XML 선언을 포함하는 BLOB에 대해 내재적으로 XML 순번을 매기게 됩니다.

  값이 YES로 설정되면 XML 값의 DESCRIBE는 CLOB(2GB)를 리턴하고 XML 값의 FETCH는 XML 선언을 포함하지 않는 CLOB에 대해 내재적으로 XML 순번을 매기게 됩니다.

주: DB2_MAP_XML_AS_CLOB_FOR_DLC는 사용되지 않으며 추후 릴리스에서는 제거됩니다. XML 값에 액세스하는 대부분의 기존 DB2 응용프로그램은 XML 가능 클라이언트를 사용하여 이를 수행하므로 이 변수는 더 이상 필요하지 않습니다.

DB2_MAX_LOB_BLOCK_SIZE

- 운영 체제: 모두
- 디폴트: 0(한계 없음), 값: 0 - 21,487,483,647
- 블록에 리턴할 LOB 또는 XML 데이터의 최대 크기를 설정합니다. 이 값은 엄격한 최대값이 아닙니다. 데이터 검색 중에 서버에서 이 최대값에 접근하는 경우, 서버는 클라이언트에 대해 FETCH와 같은 명령에 대한 응답을 생성하기 전에 현재 행에서 쓰기를 완료합니다.

DB2_MEMORY_PROTECT

- 운영 체제: 스토리지 키 지원이 있는 AIX
- 디폴트: NO, 값: NO 또는 YES
- 이 레지스트리 변수는 유효하지 않은 메모리 액세스가 원인인 버퍼 풀의 데이터 손상을 방지하기 위해 스토리지 키를 사용하는 메모리 보호 기능을 사용 가능하게 합니다. 메모리 보호는 DB2 엔진 스레드가 버퍼 풀 메모리에 액세스할 수 있는 시간과 액세스할 수 없는 시간을 식별하여 작동합니다. **DB2_MEMORY_PROTECT**가 YES로 설정되면 DB2 엔진 스레드가 버퍼 풀 메모리에 올바르지 않게 액세스하려고 할 때마다 해당 엔진 스레드가 트랩됩니다.

  주: DB2_LGPAGE_BP가 YES로 설정된 경우 메모리 보호를 사용할 수 없습니다. **DB2_MEMORY_PROTECT**가 YES로 설정된 경우에도 DB2는 버퍼 풀 메모리를 보호하지 못하고 기능을 사용 불가능하게 합니다.

DB2NOEXITLIST

- 운영 체제: 모두
- 디폴트: OFF, 값: ON 또는 OFF
- 이 변수는 **DB2_COMMIT_ON_EXIT** 레지스트리 변수 설정과 관계없이 응용프로그램이 종료될 때 DB2가 종료 목록 핸들러를 로드하지 않고 커미트를 수행하지 않음을 표시합니다.

  DB2NOEXITLIST를 끄고 **DB2_COMMIT_ON_EXIT**를 켜면 Embedded SQL 응용프로그램에 대한 모든 인플라이트 트랜잭션이 자동으로 커미트됩니다. 응용프로그램이 종료될 때 COMMIT 또는 ROLLBACK문을 명시적으로 추가하는 것이 좋습니다.

응용프로그램이 종료되기 전에 DB2 라이브러리를 동적으로 로드 및 언로드하는 응용프로그램은 DB2 종료 핸들러를 호출할 때 손상됩니다. 이러한 손상은 응용프로그램이 메모리에 있지 않은 함수를 호출하려고 시도하기 때문에 발생합니다. 이러한 상황을 방지하려면 **DB2NOEXITLIST** 레지스트리 변수를 설정하십시오.

DB2_NUM_CKPW_DAEMONS

- 운영 체제: UNIX
- 디폴트: 3, 값: 1[:FORK] - 100[:FORK]
- **DB2_NUM_CKPW_DAEMONS** 레지스트리 변수를 사용하여 구성 가능한 개수의 암호 점검 디먼을 시작할 수 있습니다. 디먼은 db2start 중에 작성되며 디폴트 IBMOSauthserver 보안 플러그인이 사용 중일 때 암호 점검 요청을 처리합니다. **DB2_NUM_CKPW_DAEMONS**의 설정값을 늘리면 데이터베이스 연결을 설정하는 데 필요한 시간이 줄어들지만 이는 동시에 여러 개의 연결이 수행되고 인증 비용이 많이 드는 시나리오에서만 효과적입니다.

  DB2_NUM_CKPW_DAEMONS는 1 - 100 사이의 값으로 설정할 수 있습니다. 데이터베이스 관리 프로그램은 **DB2_NUM_CKPW_DAEMONS**로 지정한 디먼 수를 작성합니다. 각 디먼은 암호 점검 요청을 직접 처리할 수 있습니다.

  선택적 FORK 매개변수를 추가하여 암호 점검 요청을 처리하는 외부 암호 점검 프로그램(db2ckpw)을 명시적으로 생성하기 위해 암호 점검 디먼을 사용 가능하게 할 수 있습니다. 이는 이전 릴리스에서 **DB2_NUM_CKPW_DAEMONS**를 0으로 설정한 것과 유사합니다. FORK 모드에서 각 암호 점검 디먼은 암호를 점검하려는 각 요청에 대해 암호 점검 프로그램을 생성합니다. FORK 모드의 디먼은 인스턴스 소유자로 시작됩니다.

  DB2_NUM_CKPW_DAEMONS가 0으로 설정된 경우, 유효 값은 3:FORK로 설정되며 이 경우 3개의 암호 점검 디먼이 FORK 모드에서 시작됩니다.

DB2_OPTSTATS_LOG

- 운영 체제: 모두
- 디폴트: 설정되지 않음(아래 세부사항 참조), 값: OFF, ON {NUM | SIZE | NAME | DIR}
- **DB2_OPTSTATS_LOG**는 통계 콜렉션 관련 활동을 모니터하고 분석하는 데 사용되는 통계 이벤트 로깅 파일의 속성을 지정합니다. **DB2_OPTSTATS_LOG**가 설정되지 않거나 ON으로 설정된 경우, 통계 이벤트 로깅이 사용 가능하여 시스템 성능을 모니터하고 더 나은 문제점 판별

을 위해 실행 기록을 보존할 수 있습니다. 로그 레코드는 첫 번째 로그 파일이 가득 찰 때까지 해당 파일에 기록됩니다. 후속 레코드는 다음 사용 가능한 로그 파일에 기록됩니다. 파일의 최대수에 접근하면 새 레코드가 가장 오래된 로그 파일을 겹쳐씁니다. 시스템 자원 사용이 걱정되는 경우, 이 레지스트리 변수를 OFF로 설정하여 사용 불가능하게 하십시오.

통계 이벤트 로깅이 명시적으로 사용 가능한 경우(ON으로 설정), 수정할 수 있는 여러 가지 옵션이 있습니다.

- NUM: 회전 로그 파일의 최대수. 디폴트: 5, 값 1 - 15
- SIZE: 회전 로그 파일의 최대 *크기*. (각 회전 파일의 *크기*는 SIZE/NUM입니다.) 디폴트: 100Mb, 값 1Mb – 4096Mb
- NAME: 회전 로그 파일의 기본 이름. 디폴트: db2optstats.*number*.log, 예: db2optstats.0.log, db2optstats.1.log 등.
- DIR: 회전 로그 파일의 기본 디렉토리. 디폴트: **diagpath**/events

이러한 옵션에 원하는 만큼 값을 지정할 수 있으며 통계 로깅을 사용 가능하게 하려는 경우 ON이 첫 번째 값인지 확인하십시오. 예를 들어, 최대 6개의 로그 파일, 25Mb의 최대 파일 *크기*, 기본 파일 이름 mystatslog 및 디렉토리 mystats를 사용하여 통계 로깅을 사용 가능하게 하려면 다음 명령을 발행하십시오.

```
db2set DB2_OPTSTATS_LOG=ON,NUM=6,SIZE=25,NAME=mystatslog,DIR=mystats
```

DB2REMOTEPREG

- 운영 체제: Windows
- 디폴트: NULL, 값: 임의의 유효한 Windows 머신 이름
- DB2 인스턴스 및 DB2 인스턴스 프로파일의 Win32 레지스트리 목록을 포함하는 리모트 머신 이름을 지정합니다. **DB2REMOTEPREG**의 값은 DB2가 설치된 후 한 번만 설정되어야 하며 수정되지 않아야 합니다. 매우 주의하여 이 변수를 사용하십시오.

DB2_RESOLVE_CALL_CONFLICT

- 운영 체제: AIX, HP-UX, Solaris, Linux, Windows
- 디폴트: YES, 값: YES, NO
- 트리거 컨텍스트에서 SQLCODE -746 오류를 제거합니다. 트리거에서 CALL 문을 발행할 때 SQLCODE SQL0746이 런타임 시 발행될 수 있습니다. SQL0746 오류는 트리거에 의해 호출된 프로시저가 호출 명령문의 컨텍스트 내에서 이전에 수정된 테이블에 액세스하지 못하게 합니다. 이 변수가 설정되면 DB2 데이터베이스 관리 프로그램은 CALL문을 실행하기 전에 테이블에 대한 모든 수정이 트리거에 대한 SQL 표준 규칙에 따라 완료되도록 강제 실행합니다.

DB2_RESOLVE_CALL_CONFLICT를 재설정하기 전에 인스턴스를 중지한 다음 재시작하십시오. 그런 다음 트리거 호출의 원인이 되는 패키지를 리바인드하십시오. SQL 프로시저를 리바인드하려면 다음을 사용하십시오.
CALL SYSPROC.REBIND_ROUTINE_PACKAGE
('P','*procedureschema.procedurename*','CONSERVATIVE');

DB2_RESOLVE_CALL_CONFLICT가 성능에 영향을 미칠 수 있음을 알아야 합니다. **DB2_RESOLVE_CALL_CONFLICT**를 YES로 설정하면 DB2 데이터베이스 관리 프로그램은 필요에 따라 임시 테이블의 삽입을 통해 모든 잠재적인 읽기 및 쓰기 충돌을 해결하게 됩니다. 일반적으로 최대 하나의 임시 테이블이 삽입되므로 영향은 작습니다. 트리거링 명령문은 하나의(또는 소수의) 행만 트리거링 명령문에 의해 수정되므로 OLTP 환경에서 이 효과는 작습니다. 일반적으로 임시 테이블 스페이스에 SMS(system managed space)를 사용하는 일반적인 권장사항을 따르는 경우 **DB2_RESOLVE_CALL_CONFLICT** 설정의 성능 영향은 낮을 것으로 예측됩니다.

DB2ROUTINE_DEBUG

- 운영 체제: AIX 및 Windows
- 디폴트: OFF, 값: ON 또는 OFF
- Java 스토어드 프로시저에 대한 디버그 기능을 사용하는지 여부를 지정합니다. Java 스토어드 프로시저를 디버그하지 않으려면 디폴트인 OFF를 사용하십시오. 디버깅을 사용 가능하게 하면 성능 영향이 있습니다.

주: DB2ROUTINE_DEBUG는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 이 스토어드 프로시저 디버거는 통합 디버거로 교체되었습니다.

DB2SATELLITEID

- 운영 체제: 모두
- 디폴트: NULL, 값: Satellite 제어 데이터베이스에서 선언된 유효한 Satellite ID
- Satellite이 동기화할 때 Satellite 제어 서버에 전달되는 Satellite ID를 지정합니다. 이 변수에 값이 지정되지 않으면 로그온 ID가 Satellite ID로 사용됩니다.

DB2_SERVER_CONTIMEOUT

- 운영 체제: 모두
- 디폴트: 180, 값: 0 - 32,767초
- 이 레지스트리 변수와 **DB2_DISPATCHER_PEEKTIMEOUT** 레지스트리 변수는 모두 연결 시간 동안 새 클라이언트의 처리를 구성합니다. **DB2_SERVER_CONTIMEOUT**을 사용하여 연결을 종료하기 전에 에이

전트가 클라이언트의 연결 요청을 대기하는 시간(초)을 조정할 수 있습니다. 대부분의 경우에는 이 레지스트리 변수를 조정할 필요가 없지만 DB2 클라이언트가 연결 시간에 서버에 의해 지속적으로 시간종료되는 경우에는 **DB2_SERVER_CONTIMEOUT**에 높은 값을 설정하여 시간종료 기간을 연장할 수 있습니다. 유효하지 않은 값이 설정된 경우, 디폴트값이 사용됩니다. 이 레지스트리 변수는 동적이 아닙니다.

DB2_SERVER_ENCALG

- 운영 체제: 모두
- 디폴트: NULL, 값: AES_CMP 또는 AES_ONLY
-

  주: DB2_SERVER_ENCALG는 버전 9.7에서는 사용되지 않으며, 추후 릴리스에서는 제거될 수 있습니다.

  DB2 버전 9.7로 인스턴스를 업그레이드할 때 **DB2_SERVER_ENCALG** 레지스트리 변수가 설정된 경우, **alternate_auth_enc** 구성 매개변수는 **DB2_SERVER_ENCALG** 설정에 따라 AES_ONLY 또는 AES_CMP로 설정됩니다. 따라서 사용자 ID와 암호를 암호화하기 위한 암호화 알고리즘을 지정하려면 **alternate_auth_enc** 구성 매개변수를 갱신하십시오. **alternate_auth_enc** 구성 매개변수가 설정된 경우, 해당 값은 **DB2_SERVER_ENCALG** 레지스트리 변수값에 우선합니다.

DB2SORT

- 운영 체제: 모두, 서버 전용
- 디폴트: NULL
- 이 변수는 런타임 시 로드 유틸리티가 로드할 라이브러리의 위치를 지정합니다. 라이브러리에는 인덱싱 데이터 정렬에 사용되는 함수의 시작점이 포함되어 있습니다. 테이블 인덱스 작성에 로드 유틸리티와 함께 사용하기 위해 벤더 제공 정렬 제품을 이용하려면 **DB2SORT**를 사용하십시오. 제공된 경로는 데이터베이스 서버와 관련되어 있어야 합니다.

DB2_TRUNCATE_REUSESTORAGE

- 운영 체제: 모두
- 디폴트: NULL(설정되지 않음), 값: IMPORT, import
- 이 변수를 사용하여 **REPLACE**가 있는 IMPORT 명령과 BACKUP ... ONLINE 명령 간의 잠금 경합을 해결할 수 있습니다. 일부 상황에서는 온라인 백업 및 절단 조작을 동시에 실행할 수 없습니다. 이 경우, **DB2_TRUNCATE_REUSESTORAGE**를 IMPORT 또는 import로 설정할 수 있으며 데이터, 인덱스, Long 필드, 대형 오브젝트 및 블록 맵(다차원적으로 클러스터된(MDC) 테이블)을 포함하는 오브젝트의 실제 절단은 건너

뛰고 논리적 절단만 수행됩니다. 즉, **REPLACE**가 있는 IMPORT 명령은 테이블을 비워 오브젝트의 논리적 크기가 줄어들게 하지만 디스크의 스토리지는 할당된 채로 유지됩니다.

이 레지스트리 변수는 동적입니다. 인스턴스를 중지 및 시작하지 않고 변수를 설정하거나 설정 해제할 수 있습니다. 온라인 백업이 시작되기 전에 **DB2_TRUNCATE_REUSESTORAGE**를 설정한 다음 온라인 백업이 완료된 후 이를 설정 해제할 수 있습니다. 다중 파티션된 환경의 경우, 레지스트리 변수는 변수가 설정된 노드에서만 활성화됩니다.
DB2_TRUNCATE_REUSESTORAGE는 DMS 영구 오브젝트에서만 유효합니다.

SAP 환경에서 **DB2_WORKLOAD**=SAP가 설정되면 이 레지스트리 변수의 디폴트값은 IMPORT입니다.

DB2_USE_DB2JCCT2_JROUTINE

- 운영 체제: 모두
- 디폴트: 설정되지 않음, 값: ON/YES/1/TRUE 또는 OFF/NO/0/FALSE
- Java 스토어드 프로시저 및 사용자 정의 함수의 디폴트 드라이버는 IBM Data Server Driver for JDBC and SQLJ입니다. 사용되지 않는 드라이버인 Linux, UNIX 및 Windows용 DB2 JDBC 유형 2 드라이버를 사용하여 Java 루틴에 대한 SQL 요청을 제공하려면 **DB2_USE_DB2JCCT2_JROUTINE**을 OFF, NO, 0 또는 FALSE 중 하나로 설정하십시오.

DB2_UTIL_MSGPATH

- 운영 체제: 모두
- 디폴트: *instanceName*/tmp 디렉토리
- **DB2_UTIL_MSGPATH** 레지스트리 변수는 SYSPROC.ADMIN_CMD 프로시저, SYSPROC.ADMIN_REMOVE_MSGS 프로시저 및 SYSPROC.ADMIN_GET_MSGS UDF와 함께 사용됩니다. 이 변수는 인스턴스 레벨에서 적용됩니다. **DB2_UTIL_MSGPATH**를 사용하여 분리 사용자 ID가 파일을 읽고 쓰고 삭제할 수 있는 서버의 디렉토리 경로를 표시할 수 있습니다. 이 디렉토리는 모든 코디네이터 파티션에서 액세스 가능해야 하며 유틸리티 메시지 파일을 포함할 충분한 스페이스가 있어야 합니다.

  이 경로가 설정되지 않으면 *instanceName*/tmp 디렉토리가 디폴트로 사용됩니다(DB2가 설치 제거되면 *instanceName*/tmp는 제거됩니다).

  이 경로가 변경되면 이전 설정으로 지정된 디렉토리에 있는 파일은 자동으로 이동하거나 삭제되지 않습니다. 이전 경로 아래에서 작성된 메시지의 컨텐츠를 검색하려면 이러한 메시지(유틸리티 이름이 앞에 붙고 사용자 ID가 뒤에

붙음)를 **DB2_UTIL_MSGPATH**가 지정하는 새 디렉토리로 수동으로 이동해야 합니다. 다음 유틸리티 메시지 파일은 새 위치에서 작성되고 읽히고 제거됩니다.

DB2_UTIL_MSGPATH 디렉토리 아래에 있는 파일은 트랜잭션에 종속된 것이 아니라 유틸리티에 따라 다릅니다. 이러한 파일은 백업 이미지의 일부가 아닙니다. **DB2_UTIL_MSGPATH** 디렉토리 아래에 있는 파일은 사용자 관리됩니다. 이는 사용자가 SYSPROC.ADMIN_REMOVE_UTILMSG 프로시저를 사용하여 메시지 파일을 삭제할 수 있음을 의미합니다. 이러한 파일은 DB2 설치 제거로 제거되지 않습니다.

DB2_VENDOR_INI

- 운영 체제: AIX, HP-UX, Solaris 및 Windows
- 디폴트: NULL, 값: 임의의 유효한 경로 및 파일.
- 모든 벤더 특정 환경 설정을 포함하는 파일을 지정합니다. 값은 데이터베이스 관리 프로그램이 시작될 때 읽힙니다.

주: DB2_VENDOR_INI는 버전 9.5에서는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 대신에 이 변수가 포함하는 환경 변수 설정을 **DB2_DJ_INI** 변수에서 지정한 파일에 저장할 수 있습니다.

DB2_XBSA_LIBRARY

- 운영 체제: AIX, HP-UX, Solaris 및 Windows
- 디폴트: NULL, 값: 임의의 유효한 경로 및 파일.
- 벤더 제공 XBSA 라이브러리를 지정합니다. AIX에서는 공유 오브젝트 이름이 `shr.o`로 지정되지 않은 경우, 설정에 공유 오브젝트가 포함되어야 합니다. HP-UX, Solaris 및 Windows에서는 공유 오브젝트 이름이 필요하지 않습니다. 예를 들어, Legato의 DB2용 NetWorker 비즈니스 제품 모듈을 사용하려면 레지스트리 변수를 다음과 같이 설정해야 합니다.

```
db2set DB2_XSBA_LIBRARY="/usr/lib/libxdb2.a(bsashr10.o)"
```

XBSA 인터페이스는 BACKUP DATABASE 또는 RESTORE DATABASE 명령을 통해 호출할 수 있습니다. 예를 들어, 다음과 같습니다.

```
db2 backup db sample use XBSA
db2 restore db sample use XBSA
```

제 21 장 레지스트리 및 환경 변수

환경 변수 및 프로파일 레지스트리

환경 변수 및 레지스트리 변수는 데이터베이스 환경을 제어합니다.

구성 지원 프로그램(db2ca)을 사용하여 레지스트리 변수 및 구성 매개변수를 구성할 수 있습니다.

DB2 데이터베이스 프로파일 레지스트리 도입 전에는 Windows 워크스테이션 등에서 환경 변수를 변경하려면 환경 변수를 변경한 다음 재시작해야 했습니다. 이제 몇 가지 예외 외에는 DB2 프로파일 레지스트리에 저장된 레지스트리 변수로 환경을 제어합니다. 지정된 인스턴스에 대한 시스템 관리(SYSADM) 권한이 있는 UNIX 운영 체제의 사용자는 해당 인스턴스의 레지스트리 값을 갱신할 수 있습니다. Windows에서 프로파일 레지스트리 변수를 갱신하려면 다음 조건에 따라 로컬 관리자 권한 또는 SYSADM 권한이 필요합니다.

- 확장 보안이 사용 가능한 경우, SYSADM 사용자는 DB2ADMNS 그룹에 속해야 합니다.
- 확장 보안이 사용 가능하지 않은 경우, Windows 레지스트리에서 해당 권한이 SYSADM 사용자에게 부여된 경우 해당 사용자가 갱신할 수 있습니다.

재시작하지 않고 레지스트리 변수를 갱신하려면 db2set 명령을 사용하십시오. 이 명령은 즉시 프로파일 레지스트리에 저장됩니다. 그러나 현재 실행 중인 DB2 응용프로그램 또는 사용자에게는 변경사항이 영향을 미치지 않습니다. DB2 레지스트리는 변경된 후 시작된 DB2 서버 인스턴스 및 DB2 응용프로그램에 갱신된 정보를 적용합니다.

주: DB2 프로파일 레지스트리에 저장되지 않는 DB2 환경 변수 **DB2INSTANCE** 및 **DB2NODE**가 있습니다. 일부 운영 체제에서는 이러한 환경 변수를 갱신하기 위해 set 명령을 사용해야 합니다. 이러한 변경사항은 다음에 시스템이 재시작될 때까지 적용됩니다. Linux 및 UNIX 플랫폼에서는 set 명령 대신에 export 명령이 사용됩니다.

프로파일 레지스트리는 중앙집중식 환경 변수 제어에 사용할 수 있습니다. 이제 여러 가지 프로파일을 통해 여러 가지 레벨의 지원이 제공됩니다. 또한 DB2 Administration Server를 사용하면 환경 변수의 리모트 관리도 사용 가능합니다.

네 가지 프로파일 레지스트리가 있습니다.

- DB2 인스턴스 레벨 프로파일 레지스트리. 대부분의 DB2 환경 변수는 이 레지스트리에 있습니다. 특정 인스턴스에 대한 환경 변수 설정은 이 레지스트리에 보존됩니다. 이 레벨에 정의된 값은 전역 레벨의 설정값을 겹쳐씁니다.

- DB2 전역 레벨 프로파일 레지스트리. 환경 변수가 특정 인스턴스에 대해 설정되지 않은 경우 이 레지스트리가 사용됩니다. 이 레지스트리는 특정 DB2 ESE 사본과 관련된 모든 인스턴스에 표시되며 하나의 글로벌 레벨 프로파일이 설치 경로에 있습니다.
- DB2 인스턴스 노드 레벨 프로파일 레지스트리. 이 레지스트리 레벨에는 파티션된 데이터베이스 환경의 데이터베이스 파티션에 특정한 변수 설정값이 포함되어 있습니다. 이 레벨에 정의된 값은 인스턴스 레벨 및 전역 레벨의 설정값을 겹쳐씁니다.
- DB2 인스턴스 프로파일 레지스트리. 이 레지스트리에는 현재 사본과 연관된 모든 인스턴스 이름 목록이 포함되어 있습니다. 각 설치에는 고유 목록이 있습니다. db2ilist를 실행하여 시스템에서 사용 가능한 모든 인스턴스의 전체 목록을 확인할 수 있습니다.

DB2는 다음 순서로 레지스트리 값 및 환경 변수를 점검하고 분석하여 운영 환경을 구성합니다.

1. `set` 명령을 사용하여 설정된 환경 변수. (UNIX 플랫폼에서는 `export` 명령)
2. 인스턴스 노드 레벨 프로파일을 사용하여 설정된 레지스트리 값(db2set -i <instance name> <nodenum> 명령 사용).
3. 인스턴스 레벨 프로파일을 사용하여 설정된 레지스트리 값(db2set -i 명령 사용).
4. 전역 레벨 프로파일을 사용하여 설정된 레지스트리 값(db2set -g 명령 사용).

인스턴스 레벨 프로파일 레지스트리

파티션된 데이터베이스 환경에서 작업하는 경우, UNIX와 Windows의 몇 가지 다른점이 있습니다. 이러한 다른점은 다음 예에서 설명됩니다.

『빨강』, 『흰색』 및 『파랑』으로 식별되는 세 가지 실제 데이터베이스 파티션이 있는 파티션된 데이터베이스 환경이 있다고 가정하십시오. UNIX 플랫폼에서 인스턴스 소유자가 데이터베이스 파티션 중 하나에서 다음을 실행하는 경우

```
db2set -i FOO=BAR
```

또는

```
db2set FOO=BAR        ('-i' is implied)
```

FOO 값은 현재 인스턴스의 모든 노드에 표시됩니다(즉, 『빨강』, 『흰색』 및 『파랑』).

UNIX 플랫폼에서는 인스턴스 레벨 프로파일 레지스트리는 `sqllib` 디렉토리 내부의 텍스트 파일에 저장됩니다. 파티션된 데이터베이스 환경에서 `sqllib` 디렉토리는 모든 실제 데이터베이스 파티션이 공유하는 파일 시스템에 있습니다.

Windows 플랫폼에서 사용자가 『빨강』에서 동일한 명령을 수행하는 경우, FOO 값은 현재 인스턴스의 『빨강』에만 표시됩니다. DB2 데이터베이스 관리 프로그램은 Windows 레지스트리 내부에 인스턴스 레벨 프로파일 레지스트리를 저장합니다. 실제 데이터베이

스 파티션 간에는 공유되지 않습니다. 모든 실제 컴퓨터에 레지스트리 변수를 설정하려면 다음과 같이 『rah』 명령을 사용하십시오.

```
rah db2set -i FOO=BAR
```

rah는 『빨강』, 『흰색』 및 『파랑』에서 db2set 명령을 리모트로 실행합니다.

DB2REMOTEPREG를 사용하여 인스턴스를 소유하지 않는 컴퓨터의 레지스트리 변수가 인스턴스 소유 컴퓨터의 레지스트리 변수를 참조하도록 구성할 수 있습니다. 이는 인스턴스 소유 컴퓨터의 레지스트리 변수가 인스턴스의 모든 컴퓨터 사이에서 공유되는 환경을 효과적으로 작성합니다.

위의 예를 사용하고 『빨강』이 소유 컴퓨터라고 가정하면 다음을 수행하여 『빨강』의 레지스트리 변수를 공유하도록 『흰색』 및 『파랑』 컴퓨터에 **DB2REMOTEPREG**를 설정합니다.

```
(on red) do nothing
(on white and blue) db2set DB2REMOTEPREG=WWred
```

DB2REMOTEPREG의 설정값은 설정된 후 변경되지 않아야 합니다.

REMOTEPREG가 작동하는 방식은 다음과 같습니다.

DB2 데이터베이스 관리 프로그램이 Windows에서 레지스트리 변수를 읽는 경우, 해당 프로그램은 먼저 **DB2REMOTEPREG** 값을 읽습니다. **DB2REMOTEPREG**가 설정된 경우에는 **DB2REMOTEPREG** 변수에 컴퓨터 이름이 지정된 리모트 컴퓨터에서 레지스트리를 엽니다. 이후의 레지스트리 변수 읽기 및 갱신은 지정된 리모트 컴퓨터로 경로 재지정됩니다.

리모터 레지스트리에 액세스하려면 리모트 레지스트리 서비스가 목표 컴퓨터에서 실행 중이어야 합니다. 또한 사용자 로그온 어카운트 및 모든 DB2 서비스 로그온 어카운트에 리모트 레지스트리에 대한 충분한 액세스 권한이 있어야 합니다. 따라서 **DB2REMOTEPREG**를 사용하려면 도메인 어카운트에 필수 레지스트리 액세스 권한이 부여되도록 Windows 도메인 환경에서 운영해야 합니다.

MSCS(Microsoft Cluster Server) 고려사항이 있습니다. MSCS 환경에서는 **DB2REMOTEPREG**를 사용하지 않아야 합니다. 모든 컴퓨터가 같은 MSCS 클러스터에 속하는 MSCS 구성에서 실행 중인 경우, 레지스트리 변수는 클러스터 레지스트리에 유지됩니다. 따라서 레지스트리 변수는 같은 MSCS 클러스터의 모든 컴퓨터 사이에서 이미 공유되며 이 경우 **DB2REMOTEPREG**를 사용할 필요가 없습니다.

데이터베이스 파티션이 여러 MSCS 클러스터에 속하는 다중 파티션 장애 복구 환경에서 실행 중인 경우, 인스턴스 소유 컴퓨터의 레지스트리 변수가 클러스터 레지스트리에 상주하므로 **DB2REMOTEPREG**를 사용하여 인스턴스 소유 컴퓨터를 지정할 수 없습니다.

레지스트리 및 환경 변수 선언, 표시, 변경, 재설정 및 삭제

모든 특정 레지스트리 변수를 DB2 데이터베이스 프로파일 레지스트리에 정의하는 것이 좋습니다. DB2 변수를 레지스트리 외부에 설정하는 경우, 해당 변수를 리모트로 관리할 수 없으며 변수값을 적용하려면 워크스테이션을 재시작해야 합니다.

db2set 명령은 레지스트리 변수 및 환경 변수의 로컬 선언을 지원합니다.

해당 명령에 대한 도움말 정보를 표시하려면 다음을 사용하십시오.

```
db2set -?
```

지원되는 모든 레지스트리 변수의 전체 세트를 나열하려면 다음을 사용하십시오.

```
db2set -lr
```

현재 또는 디폴트 인스턴스에 대해 정의된 모든 레지스트리 변수를 나열하려면 다음을 사용하십시오.

```
db2set
```

프로파일 레지스트리에 정의된 모든 레지스트리 변수를 나열하려면 다음을 사용하십시오.

```
db2set -all
```

현재 또는 디폴트 인스턴스의 레지스트리 변수 값을 표시하려면 다음을 사용하십시오.

```
db2set registry_variable_name
```

모든 레벨의 레지스트리 변수 값을 표시하려면 다음을 사용하십시오.

```
db2set registry_variable_name -all
```

현재 또는 디폴트 인스턴스의 레지스트리 변수를 변경하려면 다음을 사용하십시오.

```
db2set registry_variable_name=new_value
```

인스턴스의 모든 데이터베이스에 대한 레지스트리 변수 디폴트를 변경하려면 다음을 사용하십시오.

```
db2set registry_variable_name=new_value
    -i instance_name
```

인스턴스의 특정 데이터베이스 파티션에 대한 레지스트리 변수 디폴트를 변경하려면 다음을 사용하십시오.

```
db2set registry_variable_name=new_value
    -i instance_name database_partition_number
```

시스템의 특정 설치와 관련된 모든 인스턴스에 대한 레지스트리 변수 디폴트를 변경하려면 다음을 사용하십시오.

```
db2set registry_variable_name=new_value -g
```

DB2_WORKLOAD와 같은 집계 레지스트리 변수를 사용하여 SAP 환경에 대해 레지스트리 변수를 구성하는 경우, 다음을 사용하여 해당 변수를 설정할 수 있습니다.

```
db2set DB2_WORKLOAD=SAP
```

LDAP(Lightweight Directory Access Protocol)을 사용하는 경우, 다음을 사용하여 LDAP에 레지스트리 변수를 설정할 수 있습니다.

- LDAP 내에 사용자 레벨의 레지스트리 변수를 설정하려면 다음을 사용하십시오.

  ```
  db2set -ul
  ```

- LDAP 내에 전역 레벨의 레지스트리 변수를 설정하려면 다음을 사용하십시오.

  ```
  db2set -gl user_name
  ```

LDAP 환경에서 실행 중인 경우, 해당 범위가 디렉토리 파티션 또는 Windows 도메인에 속하는 모든 서버 및 모든 사용자에 대해 전역이 되도록 DB2 레지스트리 변수값을 설정할 수 있습니다. 현재 LDAP 글로벌 레벨에서 설정할 수 있는 DB2 레지스트리 변수는 **DB2LDAP_KEEP_CONNECTION** 및 **DB2LDAP_SEARCH_SCOPE** 두 가지뿐입니다.

예를 들어, LDAP에서 글로벌 레벨의 검색 범위 값을 설정하려면 다음을 사용하십시오.

```
db2set -gl db2ldap_search_scope = value
```

여기서 값은 로컬, 도메인 또는 전역 중 하나입니다.

주:

1. 둘 이상의 사용자가 동시 또는 매우 근접한 시간에 db2set 명령을 사용하여 DB2 profile.env 파일을 갱신하는 경우, profile.env 파일의 크기는 0으로 줄어듭니다. 또한 db2set -all의 출력은 불일치 값을 표시합니다.
2. DB2 ESE의 같은 설치와 관련된 모든 인스턴스에 적용되는 DB2 레지스트리 변수를 설정하는 데 사용되는 -g 옵션과 LDAP 전역 레벨에 특정적으로 사용되는 -gl 옵션 간에는 차이가 있습니다.
3. LDAP 환경에서 실행 중인 경우 Windows에서는 사용자 레벨 레지스트리 변수만 지원됩니다.
4. 사용자 레벨의 변수 설정값에는 사용자 특정 변수 설정값이 포함됩니다. 사용자 레벨에 대한 모든 변경사항은 LDAP 디렉토리에 기록됩니다.
5. ″-i″, ″-g″, ″-gl″ 및 ″-ul″ 매개변수는 같은 명령에 동시에 사용할 수 없습니다.
6. 일부 변수는 전역 레벨 프로파일에 대해 항상 디폴트로 설정됩니다(전역이란 같은 DB2 사본에서 실행 중인 모든 인스턴스 간에 변수가 공유됨을 의미함). 이러한 변

수는 인스턴스 또는 데이터베이스 파티션 레벨 프로파일에서 설정할 수 없습니다 (예: **DB2SYSTEM** 및 **DB2INSTDEF**).

7. UNIX에서 인스턴스의 레지스트리 값을 변경하려면 시스템 관리(SYSADM) 권한이 있어야 합니다. 루트 권한이 있는 사용자만 전역 레벨 레지스트리의 매개변수를 변경할 수 있습니다.

인스턴스의 레지스트리 변수를 전역 프로파일 레지스트리에 있는 디폴트로 재설정하려면 다음을 사용하십시오.

```
db2set -r registry_variable_name
```

인스턴스의 데이터베이스 파티션에 대한 레지스트리 변수를 전역 프로파일 레지스트리에 있는 디폴트로 재설정하려면 다음을 사용하십시오.

```
db2set -r registry_variable_name database_partition_number
```

지정된 레벨의 변수 값을 삭제하기 위해 동일한 명령 구문을 사용하여 변수를 설정하고 변수값을 지정하지 않을 수 있습니다. 예를 들어, 데이터베이스 파티션 레벨의 변수 설정값을 삭제하려면 다음을 입력하십시오.

```
db2set registry_variable_name= -i instance_name
   database_partition_number
```

변수 값을 삭제하고 사용을 제한하려면 상위 프로파일 레벨에 정의된 경우, 다음을 입력하십시오.

```
db2set registry_variable_name= -null -r instance_name
```

이 명령은 지정한 매개변수의 설정을 삭제하며 상위 레벨 프로파일에서 이 변수 값을 변경하는 것을 제한합니다(이 경우, DB2 전역 레벨 프로파일). 그러나 하위 레벨 프로파일에서는 지정한 변수를 계속 설정할 수 있습니다(이 경우, DB2 데이터베이스 파티션 레벨 프로파일).

Windows에서 환경 변수 설정

Windows 운영 체제에는 프로파일 레지스트리 외부에만 설정할 수 있는 하나의 시스템 환경 변수 **DB2INSTANCE**가 있습니다. 그러나 **DB2INSTANCE**를 설정할 필요는 없습니다. **DB2INSTANCE**가 정의되지 않은 경우, DB2 프로파일 레지스트리 변수 **DB2INSTDEF**를 전역 레벨 프로파일에 설정하여 사용하려는 인스턴스 이름을 지정하십시오.

Windows의 DB2 Enterprise Server Edition 서버에는 프로파일 레지스트리 외부에만 설정할 수 있는 두 가지 시스템 환경 변수 **DB2INSTANCE** 및 **DB2NODE**가 있습니다. **DB2INSTANCE**를 설정할 필요는 없습니다. **DB2INSTANCE**가 정의되지 않은 경우, DB2 프로파일 레지스트리 변수 **DB2INSTDEF**를 전역 레벨 프로파일에 설정하여 사용하려는 인스턴스 이름을 지정하십시오.

DB2NODE 환경 변수는 컴퓨터 내의 목표 논리 노드에 대한 요청의 경로를 지정하는 데 사용됩니다. 이 환경 변수는 DB2 프로파일 레지스트리가 아니라 응용프로그램 또는 명령이 발행된 세션에 설정해야 합니다. 이 변수가 설정되지 않은 경우, 목표 논리 노드는 디폴트로 컴퓨터에서 0으로 정의된 논리 노드로 설정됩니다.

환경 변수의 설정값을 판별하려면 echo 명령을 사용하십시오. 예를 들어, **DB2PATH** 환경 변수 값을 점검하려면 다음을 입력하십시오.

```
echo %db2path%
```

다음과 같이 DB2 환경 변수 **DB2INSTANCE** 및 **DB2NODE**를 설정할 수 있습니다 (이 설명에서 **DB2INSTANCE** 사용).

- 내 컴퓨터를 마우스 오른쪽 단추로 누르고 등록 정보를 선택하십시오.
- 고급 탭을 선택하고 환경 변수를 누른 후 다음을 수행하십시오.
  1. **DB2INSTANCE** 변수가 없는 경우:
     a. 새로 작성을 누르십시오.
     b. 변수 이름 필드를 **DB2INSTANCE**로 채우십시오.
     c. 변수 값 필드를 인스턴스 이름으로 채우십시오(예: db2inst).
  2. **DB2INSTANCE** 변수가 이미 있는 경우, 새 값을 추가하십시오.
     a. **DB2INSTANCE** 환경 변수를 선택하십시오.
     b. 값 필드를 인스턴스 이름으로 변경하십시오(예: db2inst).
  3. 이러한 변경사항이 적용되도록 시스템을 재시작하십시오.

주: 환경 변수 **DB2INSTANCE**를 세션(프로세스) 레벨에서 설정할 수도 있습니다. 예를 들어, TEST라는 두 번째 DB2 인스턴스를 시작하려는 경우, 명령 창에서 다음 명령을 발행하십시오.

```
set DB2INSTANCE=TEST
db2start
```

C 쉘에서 작업 중인 경우, 명령 창에서 다음 명령을 발행하십시오.

```
setenv DB2INSTANCE TEST
```

프로파일 레지스트리는 다음 위치에 있습니다.

- Windows 운영 체제 레지스트리의 DB2 인스턴스 레벨 프로파일 레지스트리. 경로는 다음과 같습니다.

  ```
  ₩HKEY_LOCAL_computer₩SOFTWARE₩IBM₩DB2₩PROFILES₩instance_name
  ```

  주: *instance_name*은 DB2 인스턴스의 이름입니다.

- Windows 레지스트리의 DB2 전역 레벨 프로파일 레지스트리. 경로는 다음과 같습니다.

  ```
  ₩HKEY_LOCAL_computer₩SOFTWARE₩IBM₩DB2₩GLOBAL_PROFILE
  ```

- Windows 레지스트리의 DB2 인스턴스 노드 레벨 프로파일 레지스트리. 경로는 다음과 같습니다.

  ...₩SOFTWARE₩IBM₩DB2₩PROFILES₩*instance_name*₩NODES₩*node_number*

  주: *instance_name* 및 *node_number*는 작업 중인 데이터베이스 파티션에 따라 다릅니다.
- DB2 인스턴스 프로파일 레지스트리는 필요하지 않습니다. 시스템의 각 DB2 인스턴스에 대해 키는 다음 경로에 작성됩니다.

  ₩HKEY_LOCAL_computer₩SOFTWARE₩IBM₩DB2₩PROFILES₩*instance_name*

  인스턴스 목록은 PROFILES 키 아래에 있는 키를 카운팅해서 얻을 수 있습니다.

Linux 및 UNIX 운영 체제에서 환경 변수 설정

UNIX 운영 체제에서는 시스템 환경 변수 **DB2INSTANCE**를 설정해야 합니다.

인스턴스 환경 설정에 도움이 되도록 db2profile(본 쉘 또는 콘 쉘의 경우) 및 db2cshrc(C 쉘의 경우) 스크립트가 예로 제공됩니다. 해당 파일은 insthome/sqllib에서 찾을 수 있으며 여기서 insthome은 인스턴스 소유자의 홈 디렉토리입니다.

이러한 스크립트에는 다음을 수행하는 명령문이 포함되어 있습니다.

- 다음 디렉토리로 사용자 경로를 갱신합니다.
  - insthome/sqllib/bin
  - insthome/sqllib/adm
  - insthome/sqllib/misc
- 실행을 위해 **DB2INSTANCE**를 디폴트 로컬 instance_name으로 설정합니다.

주: PATH 및 **DB2INSTANCE**를 제외한 지원되는 기타 모든 변수는 DB2 프로파일 레지스트리에 설정해야 합니다. DB2 데이터베이스 관리 프로그램에서 지원되지 않는 변수를 설정하려면 해당 변수를 사용자 스크립트 파일 userprofile 및 usercshrc에 정의하십시오.

인스턴스 소유자 또는 SYSADM 사용자는 인스턴스의 모든 사용자를 위해 이러한 스크립트를 사용자 정의할 수 있습니다. 또는 사용자가 스크립트를 복사 및 사용자 정의한 다음 스크립트를 직접 호출하거나 .profile 또는 .login 파일에 추가할 수 있습니다.

현재 세션의 환경 변수를 변경하려면 다음과 유사한 명령을 발행하십시오.

- 콘 쉘의 경우:

  ```
  DB2INSTANCE=<inst1>
  export DB2INSTANCE
  ```
- 본 쉘의 경우:

```
export DB2INSTANCE=<inst1>
```

- C 쉘의 경우:

```
setenv DB2INSTANCE <inst1>
```

DB2 프로파일을 올바르게 관리하기 위해 UNIX 운영 체제에서는 다음과 같은 파일 소유권 규칙을 따라야 합니다.

- DB2 인스턴스 레벨 프로파일 레지스트리 파일은 다음 아래에 있습니다.

  ```
  INSTHOME/sqllib/profile.env
  ```

  이 파일의 액세스 권한 및 소유권은 다음과 같아야 합니다.

  ```
  -rw-rw-r-- <db2inst1> <db2iadm1> profile.env
  ```

  여기서 <db2inst1>은 인스턴스 소유자이며 <db2iadm1>은 인스턴스 소유자의 그룹입니다.

  INSTHOME은 인스턴스 소유자의 홈 경로입니다.

- DB2 전역 레벨 프로파일 레지스트리는 전역 레지스트리에 저장됩니다(global.reg).

  루트 설치에서 전역 레지스트리 변수를 수정하려면 루트 권한으로 로그온하고 -g 매개변수를 포함한 db2set 명령을 실행해야 합니다.

- DB2 인스턴스 노드 레벨 프로파일 레지스트리는 다음 아래에 있습니다.

  ```
  INSTHOME/sqllib/nodes/<node_number>.env
  ```

  디렉토리 및 이 파일의 액세스 권한 및 소유권은 다음과 같아야 합니다.

  ```
  drwxrwsr-w <Instance_Owner> <Instance_Owner_Group> nodes

  -rw-rw-r-- <Instance_Owner> <Instance_Owner_Group> <node_number>.env
  ```

  INSTHOME은 인스턴스 소유자의 홈 경로입니다.

현재 인스턴스 환경 변수 설정

인스턴스의 데이터베이스 관리 프로그램을 시작하거나 중지하는 명령을 실행하면 DB2에서 해당 명령을 현재 인스턴스에 적용합니다. DB2는 현재 인스턴스를 다음과 같이 판별합니다.

- **DB2INSTANCE** 환경 변수가 현재 세션에 대해 설정된 경우, 해당 값은 현재 인스턴스입니다. **DB2INSTANCE**를 설정하려면 다음을 입력하십시오.

  ```
  set db2instance=<new_instance_name>
  ```

- **DB2INSTANCE**가 현재 세션에 대해 설정되지 않은 경우, DB2 데이터베이스 관리 프로그램은 시스템 환경 변수에서 **DB2INSTANCE** 환경 변수에 대한 설정을 사용하지 않습니다. Windows에서 시스템 환경 변수는 시스템 환경 레지스트리에 설정됩니다.

- **DB2INSTANCE**가 설정되지 않은 경우, DB2 데이터베이스 관리 프로그램은 레지스트리 변수 **DB2INSTDEF**를 사용합니다.

  DB2INSTDEF 레지스트리 변수를 레지스트리 전역 레벨에서 설정하려면 다음을 입력하십시오.

  ```
  db2set db2instdef=<new_instance_name> -g
  ```

현재 세션에 적용되는 인스턴스를 판별하려면 다음을 입력하십시오.

```
db2 get instance
```

집계 레지스트리 변수

집계 레지스트리 변수를 사용하여 여러 가지 레지스트리 변수를 다른 레지스트리 변수 이름으로 식별되는 구성으로 그룹화할 수 있습니다. 그룹의 일부인 각 레지스트리 변수에는 사전 정의된 설정이 있습니다. 집계 레지스트리 변수에는 여러 가지 레지스트리 변수 선언으로 해석되는 값이 지정됩니다.

집계 레지스트리 변수는 광범위한 운영 목적에 적합한 레지스트리 구성을 쉽게 하기 위한 것입니다.

유효한 집계 레지스트리 변수는 **DB2_WORKLOAD**뿐입니다.

이 변수에 유효한 값은 다음과 같습니다.

- 1C
- CM
- SAP
- TPM
- WC

집계 레지스트리 변수를 통해 내재적으로 구성된 레지스트리 변수는 명시적으로도 정의됩니다. 이전에 집계 레지스트리 변수 사용을 통해 값이 지정된 레지스트리 변수의 명시적 설정은 성능 또는 진단 테스트를 수행할 때 유용합니다. 집계에 의해 내재적으로 구성된 변수의 명시적 설정을 변수 겹쳐쓰기라고 합니다.

집계 레지스트리 변수를 사용하여 명시적으로 설정된 레지스트리 변수를 수정하려고 시도하는 경우, 경고가 발행되고 명시적으로 설정된 값이 보존됩니다. 이 경고는 명시된 값이 유지됨을 나타냅니다. 집계 레지스트리 변수가 먼저 사용된 다음 명시된 레지스트리 변수를 지정하면 경고가 표시되지 않습니다.

집계 레지스트리 변수 설정을 통해 구성된 레지스트리 변수는 각 변수에 대해 표시 요청이 명시적으로 작성되지 않은 경우 표시되지 않습니다. 집계 레지스트리 변수를 쿼리하면 해당 변수에 지정된 값만 표시됩니다. 대부분의 사용자는 각 개별 변수의 값을 고려할 필요가 없습니다.

다음 예에서는 집계 레지스트리 변수 사용과 레지스트리 변수 명시적 설정 간의 상호작용에 대해 설명합니다. 예를 들어, **DB2_WORKLOAD** 집계 레지스트리 변수를 SAP로 설정하고 **DB2_SKIPDELETED** 레지스트리 변수를 NO로 겹쳐씁니다. db2set을 입력하면 다음 결과가 수신됩니다.

```
DB2_WORKLOAD=SAP
DB2_SKIPDELETED=NO
```

다른 상황에서는 **DB2ENVLIST**를 설정하고 **DB2_WORKLOAD** 집계 레지스트리 변수를 SAP로 설정한 다음 **DB2_SKIPDELETED** 레지스트리 변수를 NO로 겹쳐씁니다. (**DB2_SKIPDELETED** 레지스트리 변수가 SAP 환경을 구성하는 그룹의 일부라고 가정합니다.) 또한 집계 레지스트리 변수 설정을 통해 자동 구성된 레지스트리 변수는 값 옆에 있는 대괄호 내에 집계 이름을 표시합니다. **DB2_SKIPDELETED** 레지스트리 변수는 NO 값을 표시하고 값 옆에 [O]을 표시합니다.

DB2_WORKLOAD와 연관된 구성이 더 이상 필요하지 않은 경우, 다음 명령을 통해 집계 레지스트리 변수 값을 삭제하여 그룹에 있는 각 레지스트리 변수의 내재된 값을 사용하지 않도록 할 수 있습니다.

```
db2set DB2_WORKLOAD=
```

DB2_WORKLOAD 집계 레지스트리 변수 값을 삭제한 후 데이터베이스를 재시작하십시오. 데이터베이스가 재시작되면 집계 레지스트리 변수에 의해 내재적으로 구성된 레지스트리 변수는 더 이상 적용되지 않습니다. 집계 레지스트리 값을 삭제하는 데 사용되는 메소드는 개별 레지스트리 변수 삭제와 동일합니다.

집계 레지스트리 변수 값 삭제는 명시적으로 설정된 레지스트리 변수 값을 삭제하지 않습니다. 레지스트리 변수가 사용하지 않는 그룹 정의의 구성원인지는 중요하지 않습니다. 레지스트리 변수의 명시적 설정은 유지됩니다.

DB2_WORKLOAD 집계 레지스트리 변수의 구성원인 각 레지스트리 변수의 값을 확인해야 할 수 있습니다. 예를 들어, **DB2_WORKLOAD**를 SAP로 구성한 경우 사용되는 값을 확인하려고 할 수 있습니다. **DB2_WORKLOAD**=SAP인 경우 사용될 값을 찾으려면 db2set -gd DB2_WORKLOAD=SAP를 실행하십시오.

DB2 레지스트리 및 환경 변수

DB2 데이터베이스 제품은 시작 및 실행을 위해 알아야 하는 여러 가지 레지스트리 변수 및 환경 변수를 제공합니다.

지원되는 모든 레지스트리 변수 목록을 보려면 다음 명령을 실행하십시오.

```
db2set -lr
```

현재 또는 디폴트 인스턴스의 변수 값을 변경하려면 다음 명령을 실행하십시오.

```
db2set registry_variable_name=new_value
```

DB2 환경 변수 **DB2INSTANCE**, **DB2NODE**, **DB2PATH** 및 **DB2INSTPROF**가 DB2 프로파일 레지스트리에 저장되는지 여부는 운영 체제에 따라 다릅니다. 이러한 환경 변수를 갱신하려면 set 명령을 사용하십시오. 이러한 변경사항은 로컬(현재) 명령 프롬프트에서만 유효하며 다음에 시스템이 재부트될 때까지 적용됩니다. Linux 및 UNIX 운영 체제에서는 set 명령 대신에 export 명령을 사용할 수 있습니다.

db2start 명령을 실행하기 전에 변경된 레지스트리 변수에 대한 값을 설정해야 합니다.

주: 레지스트리 변수에 부울 값이 인수로 필요한 경우, YES, 1 및 ON 값은 모두 동등한 값이며 NO, 0 및 OFF 값 또한 동등한 값입니다. 모든 변수에 대해 동등한 해당 값 중 하나를 지정할 수 있습니다.

다음 표에서는 범주별 모든 레지스트리 변수를 나열합니다.

표 66. 레지스트리 및 환경 변수 요약

변수 범주	레지스트리 또는 환경 변수 이름
일반	**DB2ACCOUNT** **DB2BIDI** **DB2_CAPTURE_LOCKTIMEOUT** **DB2CODEPAGE** **DB2_COLLECT_TS_REC_INFO** **DB2_CONNRETRIES_INTERVAL** **DB2CONSOLECP** **DB2COUNTRY** **DB2DBDFT** **DB2DBMSADDR** **DB2DISCOVERYTIME** **DB2FFDC** **DB2FODC** **DB2_FORCE_APP_ON_MAX_LOG** **DB2GRAPHICUNICODESERVER** **DB2INCLUDE** **DB2INSTDEF** **DB2INSTOWNER** **DB2_LIC_STAT_SIZE** **DB2LOCALE** **DB2_MAX_CLIENT_CONNRETRIES** **DB2_OBJECT_TABLE_ENTRIES** **DB2_SYSTEM_MONITOR_SETTINGS** **DB2TERRITORY** **DB2_VIEW_REOPT_VALUES**

표 66. 레지스트리 및 환경 변수 요약 (계속)

변수 범주	레지스트리 또는 환경 변수 이름
시스템 환경	**DB2_ALTERNATE_GROUP_LOOKUP** **DB2CONNECT_ENABLE_EURO_CODEPAGE** **DB2CONNECT_IN_APP_PROCESS** **DB2_COPY_NAME** **DB2DBMSADDR** **DB2_DIAGPATH** **DB2DOMAINLIST** **DB2ENVLIST** **DB2INSTANCE** **DB2INSTPROF** **DB2LDAPSecurityConfig** **DB2LIBPATH** **DB2LOGINRESTRICTIONS** **DB2NODE** **DB2OPTIONS** **DB2_PARALLEL_IO** **DB2PATH** **DB2_PMAP_COMPATIBILITY** **DB2PROCESSORS** **DB2RCMD_LEGACY_MODE** **DB2RESILIENCE** **DB2SYSTEM** **DB2_UPDDBCFG_SINGLE_DBPARTITION** **DB2_USE_PAGE_CONTAINER_TAG** **DB2_WORKLOAD**
통신	**DB2CHECKCLIENTINTERVAL** **DB2COMM** **DB2FCMCOMM** **DB2_FORCE_NLS_CACHE** **DB2RSHCMD** **DB2RSHTIMEOUT** **DB2SORCVBUF** **DB2SOSNDBUF** **DB2TCP_CLIENT_CONTIMEOUT** **DB2TCP_CLIENT_RCVTIMEOUT** **DB2TCPCONNMGRS**
명령행	**DB2BQTIME** **DB2BQTRY** **DB2_CLP_EDITOR** **DB2_CLP_HISTSIZE** **DB2_CLPPROMPT** **DB2IQTIME** **DB2RQTIME**
파티션된 데이터베이스 환경	**DB2CHGPWD_EEE** **DB2_FCM_SETTINGS** **DB2_FORCE_OFFLINE_ADD_PARTITION** **DB2_NUM_FAILOVER_NODES** **DB2_PARTITIONEDLOAD_DEFAULT** **DB2PORTRANGE**
쿼리 컴파일러	**DB2_ANTIJOIN** **DB2_DEFERRED_PREPARE_SEMANTICS** **DB2_INLIST_TO_NLJN** **DB2_LIKE_VARCHAR** **DB2_MINIMIZE_LISTPREFETCH** **DB2_NEW_CORR_SQ_FF** **DB2_OPT_MAX_TEMP_SIZE** **DB2_REDUCED_OPTIMIZATION** **DB2_SELECTIVITY** **DB2_SQLROUTINE_PREPOPTS**

표 66. 레지스트리 및 환경 변수 요약 (계속)

변수 범주	레지스트리 또는 환경 변수 이름
성능	**DB2_ALLOCATION_SIZE** **DB2_APM_PERFORMANCE** **DB2ASSUMEUPDATE** **DB2_ASYNC_IO_MAXFILOP** **DB2_AVOID_PREFETCH** **DB2BPVARS** **DB2CHKPTR** **DB2CHKSQLDA** **DB2_EVALUNCOMMITTED** **DB2_EXTENDED_IO_FEATURES** **DB2_EXTENDED_OPTIMIZATION** **DB2_IO_PRIORITY_SETTING** **DB2_IO_PRIORITY_SETTING** **DB2_KEEP_AS_AND_DMS_CONTAINERS_OPEN** **DB2_KEEPTABLELOCK** **DB2_LARGE_PAGE_MEM** **DB2_LOGGER_NON_BUFFERED_IO** **DB2MAXFSCRSEARCH** **DB2_MAX_INACT_STMTS** **DB2_MAX_NON_TABLE_LOCKS** **DB2_MDC_ROLLOUT** **DB2MEMDISCLAIM** **DB2MEMMAXFREE** **DB2_MEM_TUNING_RANGE** **DB2_MMAP_READ** **DB2_MMAP_WRITE** **DB2_NO_FORK_CHECK** **DB2NTMEMSIZE** **DB2NTNOCACHE** **DB2NTPRICLASS** **DB2NTWORKSET** **DB2_OVERRIDE_BPF** **DB2_PINNED_BP** **DB2PRIORITIES** **DB2_RESOURCE_POLICY** **DB2_SET_MAX_CONTAINER_SIZE** **DB2_SKIPDELETED** **DB2_SKIPINSERTED** **DB2_SMS_TRUNC_TMPTABLE_THRESH** **DB2_SORT_AFTER_TQ** **DB2_SELUDI_COMM_BUFFER** **DB2_TRUSTED_BINDIN** **DB2_USE_ALTERNATE_PAGE_CLEANING** **DB2_USE_IOCP**

표 66. 레지스트리 및 환경 변수 요약 (계속)

변수 범주	레지스트리 또는 환경 변수 이름
기타	**DB2ADMINSERVER** **DB2_ATS_ENABLE** **DB2AUTH** **DB2CLIINIPATH** **DB2_COMMIT_ON_EXIT** **DB2_CREATE_DB_ON_PATH** **DB2_DDL_SOFT_INVAL** **DB2DEFPREP** **DB2_DISABLE_FLUSH_LOG** **DB2_DISPATCHER_PEEKTIMEOUT** **DB2_DJ_INI** **DB2DMNBCKCTLR** **DB2_DOCHOST** **DB2_DOCPORT** **DB2_ENABLE_AUTOCONFIG_DEFAULT** **DB2_ENABLE_LDAP** **DB2_EVMON_EVENT_LIST_SIZE** **DB2_EVMON_STMT_FILTER** **DB2_EXTSECURITY** **DB2_FALLBACK** **DB2_FMP_COMM_HEAPSZ** **DB2_GRP_LOOKUP** **DB2_HADR_BUF_SIZE** **DB2_HADR_NO_IP_CHECK** **DB2_HADR_PEER_WAIT_LIMIT** **DB2_HADR_SORCVBUF** **DB2_HADR_SOSNDBUF** **DB2LDAP_BASEDN** **DB2LDAPCACHE** **DB2LDAP_CLIENT_PROVIDER** **DB2LDAPHOST** **DB2LDAP_KEEP_CONNECTION** **DB2LDAP_SEARCH_SCOPE** **DB2_LOAD_COPY_NO_OVERRIDE** **DB2LOADREC** **DB2LOCK_TO_RB** **DB2_MAP_XML_AS_CLOB_FOR_DLC** **DB2_MAX_LOB_BLOCK_SIZE** **DB2_MEMORY_PROTECT** **DB2NOEXITLIST** **DB2_NUM_CKPW_DAEMONS** **DB2_OPTSTATS_LOG** **DB2REMOTEPREG** **DB2_RESOLVE_CALL_CONFLICT** **DB2ROUTINE_DEBUG** **DB2SATELLITEID** **DB2_SERVER_CONTIMEOUT** **DB2_SERVER_ENCALG** **DB2SORT** **DB2_TRUNCATE_REUSESTORAGE** **DB2_USE_DB2JCCT2_JROUTINE** **DB2_UTIL_MSGPATH** **DB2_VENDOR_INI** **DB2_XBSA_LIBRARY**

일반 레지스트리 변수

DB2ACCOUNT

- 운영 체제: 모두

- 디폴트: NULL
- 이 변수는 리모트 호스트에 보내는 어카운팅 문자열을 정의합니다. 자세한 내용은 DB2 Connect 사용자 안내서를 참조하십시오.

DB2BIDI

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- 이 변수는 양방향 지원을 사용하며 **DB2CODEPAGE** 변수는 사용할 코드 페이지를 선언하는 데 사용됩니다.

DB2_CAPTURE_LOCKTIMEOUT

- 운영 체제: 모두
- 디폴트: NULL, 값: ON 또는 NULL
- 이 변수는 잠금 시간종료가 발생할 때 이에 대한 설명적 정보를 로그하도록 지정합니다. 로그된 정보는 잠금 시간종료에 이르게 한 잠금 경합에 관련된 키 응용프로그램, 해당 응용프로그램이 잠금 시간종료 시 실행 중이던 작업에 대한 세부사항 및 경쟁의 원인이 되는 잠금에 대한 세부사항을 식별합니다. 잠금 요청자(잠금 시간종료 오류를 수신한 응용프로그램)와 현재 잠금 소유자 모두에 대한 정보가 캡처됩니다. 각 잠금 시간종료에 대해 텍스트 보고서가 작성되고 파일에 저장됩니다.

  파일은 다음과 같은 이름 지정 규칙을 사용하여 작성됩니다.
  db2locktimeout.*par*.*AGENTID*.*yyyy-mm-dd-hh-mm-ss*, 여기서 *par*은 데이터베이스 파티션 번호, *AGENTID*는 에이전트 ID, *yyyy-mm-dd-hh-mm-ss*는 년, 월, 일, 시, 분 및 초로 구성된 시간소인입니다. 파티션되지 않은 데이터베이스 환경에서 *par*은 0으로 설정됩니다.

  파일의 위치는 **diagpath** 데이터베이스 구성 매개변수에 설정된 값을 기본으로 합니다. **diagpath**가 설정되지 않은 경우, 파일은 다음 디렉토리 중 하나에 있습니다.

  – Windows 환경의 경우:
    - **DB2INSTPROF** 환경 변수가 설정되지 않은 경우, 정보는 *x*:₩*SQLLIB*₩*DB2INSTANCE*에 기록되며 여기서 *x*는 드라이브 참조, *SQLLIB*는 **DB2PATH** 레지스트리 변수에 지정한 디렉토리이며 *DB2INSTANCE*는 인스턴스 소유자의 이름입니다.
    - **DB2INSTPROF** 환경 변수를 설정한 경우, 정보는 *x*:₩*DB2INSTPROF*₩*DB2INSTANCE*에 기록되며 여기서 *x*는 드라이브 참조, *DB2INSTPROF*는 인스턴스 프로파일 디렉토리의 이름이며 *DB2INSTANCE*는 인스턴스 소유자의 이름입니다.

– Linux 및 UNIX 환경에서 정보는 *INSTHOME*/sqllib/db2dump에 기록되며 여기서 *INSTHOME*은 인스턴스의 홈 디렉토리입니다.

잠금 시간종료 보고서 파일이 더 이상 필요하지 않은 경우 삭제하십시오. 보고서 파일은 다른 진단 로그와 동일한 위치에 있으므로 디렉토리가 가득 차게 될 경우 DB2 시스템에서 종료할 수 있습니다. 일부 잠금 시간종료 보고서 파일을 보존해야 할 경우, 해당 파일을 DB2 로그가 저장된 위치와 다른 디렉토리 또는 폴더로 이동하십시오.

중요사항: 이 변수는 사용되지 않으며 CREATE EVENT MONITOR FOR LOCKING문을 사용하여 잠금 시간종료 이벤트를 수집하는 새 메소드가 있으므로 추후 릴리스에서는 제거됩니다.

DB2CODEPAGE

- 운영 체제: 모두
- 디폴트: 운영 체제에서 지정한 대로 언어 ID에서 파생됩니다.
- 이 변수는 데이터베이스 클라이언트 응용프로그램에 대해 DB2에 제공되는 데이터의 코드 페이지를 지정합니다. DB2 문서에 명시적으로 표시되었거나 DB2 서비스에서 요청한 경우를 제외하면 **DB2CODEPAGE**를 설정하지 않아야 합니다. **DB2CODEPAGE**를 운영 체제에서 지원되지 않는 값으로 설정하면 예기치 않은 결과가 발생할 수 있습니다. 일반적으로 DB2는 운영 체제에서 코드 페이지 정보를 자동으로 얻으므로 **DB2CODEPAGE**를 설정할 필요가 없습니다.

  주: Windows는 Windows 지역 설정에서 ANSII 코드 페이지 대신 유니코드 코드 페이지를 보고하지 않으므로 Windows 응용프로그램은 유니코드 클라이언트처럼 작동하지 않습니다. 이 동작을 겹쳐쓰려면 응용프로그램이 유니코드 응용프로그램처럼 작동하도록 **DB2CODEPAGE** 레지스트리 변수를 1208(유니코드 코드 페이지)로 설정하십시오.

DB2_COLLECT_TS_REC_INFO

- 운영 체제: 모두
- 디폴트: ON, 값: ON 또는 OFF
- 이 변수는 테이블 스페이스를 롤 포워드할 때 테이블 스페이스에 영향을 미치는 로그 레코드가 로그 파일에 포함되어 있는지 여부와 관계없이 DB2가 모든 로그 파일을 처리하는지 여부를 지정합니다. 테이블 스페이스에 영향을 미치는 로그 레코드를 포함하지 않는 것으로 알려진 로그 파일을 건너뛰려면 이 변수를 ON으로 설정하십시오. 로그 파일을 건너뛰는 데 필요한 정보를 수집하기 위해 로그 파일을 작성하고 사용하기 전에 **DB2_COLLECT_TS_REC_INFO**를 설정해야 합니다.

DB2_CONNRETRIES_INTERVAL

- 운영 체제: 모두
- 디폴트: 설정되지 않음, 값: 초 수(정수)
- 이 변수는 자동 클라이언트 리라우트 기능에 대한 연속 연결 재시도 수 간의 휴먼 시간(초)을 지정합니다. 이 변수를 **DB2_MAX_CLIENT CONNRETRIES**와 함께 사용하여 자동 클라이언트 리라우트에 대한 재시도 동작을 구성할 수 있습니다.

  DB2_MAX_CLIENT_CONNRETRIES는 설정되었지만 **DB2_CONNRETRIES_INTERVAL**은 설정되지 않은 경우, **DB2_CONNRETRIES_INTERVAL**은 디폴트로 30으로 설정됩니다. **DB2_MAX_CLIENT_CONNRETRIES**는 설정되지 않았지만 **DB2_CONNRETRIES_INTERVAL**은 설정된 경우, **DB2_MAX_CLIENT_CONNRETRIES**는 디폴트로 10으로 설정됩니다. **DB2_MAX_CLIENT_CONNRETRIES** 및 **DB2_CONNRETRIES_INTERVAL**이 모두 설정되지 않은 경우, 자동 클라이언트 리라우트 기능은 최대 10분 동안 반복적으로 데이터베이스에 대한 연결을 재시도하는 디폴트 동작으로 되돌아갑니다.

DB2CONSOLECP

- 운영 체제: Windows
- 디폴트: NULL, 값: 모든 유효한 코드 페이지 값
- DB2 메시지 텍스트 표시를 위한 코드 페이지를 지정합니다. 지정된 경우, 이 값은 운영 체제 코드 페이지 설정값을 겹쳐씁니다.

DB2COUNTRY

- 운영 체제: Windows
- 디폴트: NULL, 값: 숫자로 된 모든 유효한 나라, 지역 또는 영역 코드
- 이 변수는 클라이언트 응용프로그램의 나라, 지역 또는 영역 코드를 지정합니다. 지정된 경우, 이 값은 운영 체제 설정값을 겹쳐씁니다.

주: DB2COUNTRY는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 대신에 **DB2COUNTRY**와 같은 값을 승인하는 **DB2TERRITORY**를 사용하십시오.

DB2DBDFT

- 운영 체제: 모두
- 디폴트: NULL
- 이 변수는 내재된 연결에 사용되는 데이터베이스의 데이터베이스 별명 이름을 지정합니다. 응용프로그램에 데이터베이스 연결이 없지만 SQL 또는 XQuery문이 발행되는 경우, 디폴트 데이터베이스를 사용하여 **DB2DBDFT** 환경 변수가 정의되었으면 내재된 연결이 작성됩니다.

DB2DBMSADDR

- 운영 체제: Windows 32비트
- 디폴트: 0x20000000, 값: 0x20000000 - 0xB0000000(0x10000씩 증가)
- 이 변수는 디폴트 데이터베이스 관리 프로그램 공유 메모리 주소를 16진수 형식으로 지정합니다. 공유 메모리 주소 충돌로 인해 db2start가 실패하는 경우, 이 레지스트리 변수를 수정하여 데이터베이스 관리 프로그램 인스턴스가 공유 메모리를 다른 주소에 할당하도록 강제 실행할 수 있습니다.

DB2DISCOVERYTIME

- 운영 체제: Windows
- 디폴트: 40초, 최소값: 20초
- 이 변수는 SEARCH 발견이 DB2 시스템을 검색하는 시간 *크기*를 지정합니다.

DB2_EXPRESSION_RULES

- 운영 체제: 모두
- 디폴트: 비어 있음, 값: RAISE_ERROR_PERMIT_SKIP 또는 RAISE_ERROR_PERMIT_DROP
- **DB2_EXPRESSION_RULES** 레지스트리 변수의 설정값은 DB2 옵티마이저가 RAISE_ERROR 기능을 포함하는 쿼리에 대한 액세스 플랜을 판별하는 방법을 제어합니다. RAISE_ERROR 함수의 디폴트 동작은 이 함수를 포함하는 표현식의 범위 밖에서는 필터링이 수행되지 않습니다. 그 결과, 표현식의 초과 계산, 초과 잠금 및 낮은 쿼리 성능으로 이어질 수 있는 테이블 액세스 중에는 술어가 적용되지 않을 수 있습니다.

  응용프로그램의 특정 비즈니스 요구사항에 따라 이 동작이 과도하게 엄격한 특정한 경우에는 술어와 조인이 RAISE_ERROR 응용프로그램 앞에 적용되는지 여부가 중요하지 않을 수 있습니다. 예를 들어, 행 레벨 보안 구현의 컨텍스트에는 일반적으로 다음과 같은 양식의 표현식이 있습니다.

  ```
  CASE WHEN <conditions for validatin access to this row>
       THEN NULL
       ELSE RAISE_ERROR(...)
  END
  ```

  응용프로그램은 테이블의 모든 행에 대한 액세스의 유효성 확인이 아니라 쿼리에 의해 선택된 행에 대한 액세스의 유효성 확인에만 관여합니다. 따라서 술어는 기본 테이블 액세스에서 적용될 수 있으며 모든 필터링이 수행된 후에는 RAISE_ERROR를 포함하는 표현식만 실행되어야 합니다. 이 경우에는 **DB2_EXPRESSION_RULES**=RAISE_ERROR_PERMIT_SKIP 값이 적당합니다.

다른 대체 표현식은 COLUMN LEVEL 보안의 컨텍스트에 있습니다. 이 경우에는 일반적으로 다음과 같은 양식의 표현식이 있습니다.

```
CASE WHEN <conditions for validating access to this row and column>
     THEN <table.column>
     ELSE RAISE_ERROR(...)
END
```

이 경우, 사용자가 특정 행에 대한 데이터를 수신하려고 시도하고 컬럼에 사용자가 검색할 수 없는 값이 포함되어 있으면 응용프로그램에서는 오류만 발생할 수 있습니다. 이 경우,
DB2_EXPRESSION_RULES=RAISE_ERROR_PERMIT_DROP 설정은 특정 컬럼이 술어 또는 컬럼 함수에서 사용되거나 쿼리의 출력으로 리턴되면 RAISE_ERROR 함수를 포함하는 표현식만 평가하게 됩니다.

DB2FFDC

- 운영 체제: 모두
- 디폴트: ON, 값: ON, CORE:OFF
- 이 변수는 코어 파일 생성을 비활성화하는 기능을 제공합니다. 디폴트로 이 레지스트리 변수는 ON으로 설정됩니다. 이 레지스트리 변수가 설정되지 않거나 CORE:OFF 이외의 값으로 설정된 경우, DB2 서버가 이상 종료되면 코어 파일이 생성될 수 있습니다.

  문제점 판별에 사용되고 **diagpath** 디렉토리에 작성되는 코어 파일은 DB2 종료 프로세스의 전체 프로세스 이미지를 포함합니다. 코어 파일이 매우 클 수 있으므로 사용 가능한 파일 시스템 스페이스에 주의해야 합니다. *크기*는 문제점이 발생한 시간의 프로세스 상태 및 DB2 구성에 따라 다릅니다.

  Linux 운영 체제에서 디폴트 코어 파일 *크기* 한계는 `0`으로 설정됩니다 (즉, `ulimit -c`). 이 설정으로는 코어 파일이 생성되지 않습니다. 코어 파일을 Linux 운영 체제에서 작성할 수 있으려면 값을 무제한으로 설정하십시오.

주: DB2FFDC는 버전 9.5에서는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 새 레지스트리 변수 **DB2FODC**가 **DB2FFDC**의 기능을 통합합니다.

DB2FODC

- 운영 체제: 모두
- 디폴트: 모든 FODC 매개변수의 병합(아래 참조)
  - Linux 및 UNIX의 경우: "CORELIMIT=*val* DUMPCORE=ON DUMPDIR=*diagpath*"
  - Windows의 경우: "DUMPCORE=ON DUMPDIR=*diagpath*"

  매개변수는 공백으로 구분됩니다.

- 이 레지스트리 변수는 FODC(First Occurrence Data Collection)에 사용되는 문제점 해결 관련 매개변수 세트를 제어합니다. **DB2FODC**를 사용하여 사용 불능 상황에서 데이터 콜렉션의 여러 가지 측면을 제어할 수 있습니다.

  이 레지스트리 변수는 DB2 인스턴스 시작 중에 한 번 읽힙니다. FODC 매개변수를 온라인으로 갱신하려면 db2pdcfg 도구를 사용하십시오. **DB2FODC** 레지스트리 변수를 사용하여 재부트 시 구성을 유지할 수 있습니다. 매개변수를 모두 지정하거나 특정 순서로 지정할 필요가 없습니다. 지정되지 않은 모든 매개변수에는 디폴트값이 지정됩니다. 예를 들어, 코어 파일이 덤프되는 것은 원하지 않지만 다른 매개변수의 디폴트 동작은 원하는 경우에는 다음 명령을 발행하십시오.

  ```
  db2set DB2FODC="DUMPCORE=OFF"
  ```

  매개변수:

  CORELIMIT

  - 운영 체제: Linux 및 UNIX
  - 디폴트: 현재 ulimit 설정, 값: 0 - 무제한
  - 이 옵션은 작성된 코어 파일의 최대 *크기*를 지정합니다(GB). 이 값은 현재 코어 파일 *크기* 한계 설정을 겹쳐씁니다. 코어 파일이 매우 클 수 있으므로 사용 가능한 파일 시스템 스페이스에 주의해야 합니다. *크기*는 문제점이 발생한 시간의 프로세스 상태 및 DB2 구성에 따라 다릅니다.

    CORELIMIT이 설정된 경우, DB2는 이 값을 사용하여 코어 파일을 생성하기 위한 현재 사용자 코어 한계(ulimit) 설정을 겹쳐씁니다.

    CORELIMIT이 설정되지 않은 경우, DB2는 코어 파일 *크기*를 현재 ulimit 설정과 같은 값으로 설정합니다. ″무제한″의 ulimit 설정이 DB2 서버 전용의 8GB 값을 겹쳐쓰는 AIX는 예외입니다. 8GB보다 큰 코어 덤프가 필요한 경우, ulimit를 RAM *크기*와 같이 적당히 큰 값으로 설정하거나 충분히 큰 값을 가진 **CORELIMIT**을 설정하십시오.

    주: 사용자 코어 한계 또는 **CORELIMIT**에 대한 변경사항은 다음에 DB2 인스턴스를 재사용할 때까지 적용되지 않습니다.

  DUMPCORE

  - 운영 체제: Linux, Solaris, AIX
  - 디폴트: AUTO, 값: AUTO, ON 또는 OFF

– 이 옵션은 코어 파일 생성이 발생하는지 여부를 지정합니다. 문제점 판별에 사용되고 **diagpath** 디렉토리에 작성되는 코어 파일은 DB2 종료 프로세스의 전체 프로세스 이미지를 포함합니다. 그러나 실제 코어 파일 덤프 발생 여부는 현재 ulimit 설정 및 **CORELIMIT** 매개변수의 값에 따라 다릅니다. 또한 일부 운영 체제에는 응용프로그램 코어 덤핑의 동작을 지시하는 코어 덤프에 대한 구성 설정이 있습니다. AUTO 설정은 **DB2RESILIENCE** 레지스트리 변수가 ON으로 설정되었을 때 트랩을 유지할 수 없는 경우에 코어 파일이 생성되도록 합니다. **DUMPCORE**=ON 설정은 항상 **DB2RESILIENCE** 레지스트리 변수 설정을 겹쳐써서 코어 파일을 생성합니다.

  코어 파일 덤프를 사용하지 않기 위한 권장 메소드는 **DUMPCORE**를 OFF로 설정하는 것입니다.

DUMPDIR

– 운영 체제: 모두

– 디폴트: **diagpath** 디렉토리 또는 **diagpath**가 정의되지 않은 경우 디폴트 진단 디렉토리, 값: *디렉토리 경로*

– 이 옵션은 코어 파일 작성을 위한 디렉토리의 절대 경로 이름을 지정합니다. 이 옵션은 코어 파일뿐만 아니라 FODC 패키지 밖에 저장해야 하는 기타 대형 실행 파일 덤프에 사용됩니다.

DB2_FORCE_APP_ON_MAX_LOG

- 운영 체제: 모두
- 디폴트: TRUE, 값: TRUE 또는 FALSE
- **max_log** 구성 매개변수 값이 초과될 때 발생하는 일을 지정합니다. TRUE로 설정되면 응용프로그램이 강제 실행되어 데이터베이스를 종료하고 작업 단위(UOW)가 롤백됩니다.

  FALSE로 설정되면 현재 명령문이 실패합니다. 응용프로그램은 작업 단위(UOW)의 이전 명령문에 의해 완료된 작업을 계속 커미트하거나 완료된 작업을 롤백하여 작업 단위를 실행 취소할 수 있습니다.

  주: 이 DB2 레지스트리 변수는 로그가 가득 찬 상황에서 복구하기 위한 임포트 유틸리티의 기능에 영향을 미칩니다. **DB2_FORCE_APP_ON_MAX_LOG**가 TRUE로 설정되고 **COMMITCOUNT** 명령 옵션과 함께 IMPORT 명령을 발행하는 경우, 임포트 유틸리티는 사용 중인 로그 스페이스 밖에서 실행되는 것을 방지하기

위해 커미트를 수행할 수 없습니다. 임포트 유틸리티가 SQL0964C(가득 찬 트랜잭션 로그)를 발견하면 유틸리티가 강제 실행되어 데이터베이스를 종료하고 현재 작업 단위는 롤백됩니다.

DB2GRAPHICUNICODESERVER

- 운영 체제: 모두
- 디폴트: OFF, 값: ON 또는 OFF
- 이 레지스트리 변수는 그래픽 데이터를 유니코드 데이터베이스에 삽입하기 위해 작성된 기존 응용프로그램을 수용하기 위해 사용됩니다. 이 변수는 클라이언트의 코드 페이지 대신 유니코드로 된 sqldbchar(그래픽) 데이터를 특정적으로 전송하는 응용프로그램에 대해서만 사용해야 합니다. (sqldbchar은 C 및 C++에서 단일 2바이트 문자를 가질 수 있는 지원되는 SQL 데이터 유형입니다.) ON으로 설정되면 데이터베이스에 그래픽 데이터가 유니코드로 생성되고 응용프로그램이 유니코드로 된 그래픽 데이터를 수신할 예정임을 알려줍니다.

DB2INCLUDE

- 운영 체제: 모두
- 디폴트: 현재 디렉토리
- DB2 PREP 처리 중에 SQL INCLUDE 텍스트 파일 명령문 처리 동안 사용되는 경로를 지정합니다. 이 변수는 INCLUDE 파일을 찾을 수 있는 디렉토리 목록을 제공합니다. 프리컴파일된 다른 언어로 **DB2INCLUDE**를 사용하는 방법에 대한 설명은 Developing Embedded SQL Applications의 내용을 참조하십시오.

DB2INSTDEF

- 운영 체제: Windows
- 디폴트: DB2
- 이 변수는 **DB2INSTANCE**가 정의되지 않은 경우에 사용되는 값을 설정합니다.

DB2INSTOWNER

- 운영 체제: Windows
- 디폴트: **NULL**
- 이 레지스트리 변수는 인스턴스가 먼저 작성된 경우에 DB2 프로파일 레지스트리에 작성됩니다. 이 변수는 인스턴스 소유 머신의 이름으로 설정됩니다.

DB2_LIC_STAT_SIZE

- 운영 체제: 모두
- 디폴트: NULL, 범위: 0 - 32,767

- 이 변수는 시스템에 대한 라이센스 통계를 포함하는 파일의 최대 *크기*(MB)를 판별합니다. 0 값은 라이센스 통계 수집을 끕니다. 변수가 인식되지 않거나 정의되지 않은 경우 디폴트로 무제한으로 설정됩니다. 통계는 라이센스 센터를 사용하여 표시됩니다.

DB2LOCALE

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- 이 변수는 DB2 호출 후 프로세스의 디폴트 ″C″ 로케일이 디폴트 ″C″ 로케일로 리스토어되는지 및 DB2 함수 호출 후 프로세스 로케일을 원래 'C'로 리스토어하는지 여부를 지정합니다. 원래 로케일이 'C'가 아닌 경우 이 레지스트리 변수는 무시됩니다.

DB2_MAX_CLIENT_CONNRETRIES

- 운영 체제: 모두
- 디폴트: 설정되지 않음, 값: 연결을 재시도할 최대 횟수(정수)
- 이 변수는 자동 클라이언트 리라우트 기능이 시도할 최대 연결 재시도 수를 지정합니다. 이 변수를 **DB2_CONNRETRIES_INTERVAL**과 함께 사용하여 자동 클라이언트 리라우트에 대한 재시도 동작을 구성할 수 있습니다.

  DB2_MAX_CLIENT_CONNRETRIES는 설정되었지만 **DB2_CONNRETRIES_INTERVAL**은 설정되지 않은 경우, **DB2_CONNRETRIES_INTERVAL**은 디폴트로 30으로 설정됩니다. **DB2_MAX_CLIENT_CONNRETRIES**는 설정되지 않았지만 **DB2_CONNRETRIES_INTERVAL**은 설정된 경우, **DB2_MAX_CLIENT_CONNRETRIES**는 디폴트로 10으로 설정됩니다. **DB2_MAX_CLIENT_CONNRETRIES** 및 **DB2_CONNRETRIES_INTERVAL**이 모두 설정되지 않은 경우, 자동 클라이언트 리라우트 기능은 최대 10분 동안 반복적으로 데이터베이스에 대한 연결을 재시도하는 디폴트 동작으로 되돌아갑니다.

DB2_OBJECT_TABLE_ENTRIES

- 운영 체제: 모두
- 디폴트: 0, 값: 0–65,532

  시스템에서 가능한 실제 최대값은 페이지 *크기* 및 Extent *크기*에 따라 다르지만 65,532를 초과할 수 없습니다.
- 이 변수는 테이블 스페이스의 예상 오브젝트 수를 지정합니다. 예를 들어, 1000개 이상의 많은 오브젝트가 DMS 테이블 스페이스에 작성될 것임을 아는 경우에는 테이블 스페이스를 작성하기 전에 이 레지스트리 변수를 근사치로 설정해야 합니다. 이는 테이블 스페이스 작성 동안 오브젝트 메타데이

터에 대해 인접하는 스토리지를 예약합니다. 인접하는 스토리지를 예약하면 온라인 백업이 메타데이터의 항목을 갱신하는 조작을 차단할 가능성을 낮춥니다(예: CREATE INDEX, IMPORT REPLACE). 또한 메타데이터가 테이블 스페이스의 시작 부분에 저장되므로 테이블 스페이스의 크기를 더 쉽게 조정할 수 있습니다.

테이블 스페이스의 초기 크기가 인접하는 스토리지를 예약할 만큼 크지 않은 경우, 추가 스페이스를 예약하지 않고 테이블 스페이스 작성이 진행됩니다.

DB2_SYSTEM_MONITOR_SETTINGS

- 운영 체제: 모두
- 이 레지스트리 변수는 DB2 모니터링의 다양한 측면의 동작을 수정할 수 있는 매개변수 세트를 제어합니다. 다음 예와 같이 각 매개변수를 세미콜론으로 구분하십시오.

  ```
  db2set DB2_SYSTEM_MONITOR_SETTINGS=OLD_CPU_USAGE:TRUE;
   DISABLE_CPU_USAGE:TRUE
  ```

  DB2_SYSTEM_MONITOR_SETTINGS를 설정할 때마다 각 매개변수가 명시적으로 설정됩니다. 이 변수를 설정할 때 지정하지 않은 매개변수는 디폴트값으로 되돌아갑니다. 따라서 다음 예에서는 아래 사항을 참조하십시오.

  ```
  db2set DB2_SYSTEM_MONITOR_SETTINGS=DISABLE_CPU_USAGE:TRUE
  ```

  주: 현재 이 레지스트리 변수에는 Linux에 대한 설정값만 있습니다. 기타 운영 체제에 대한 추가 설정은 추후 릴리스에서 추가됩니다.
  OLD_CPU_USAGE는 디폴트 설정으로 리스토어됩니다.

- 매개변수:

  OLD_CPU_USAGE

  - 운영 체제: Linux
  - 값: TRUE/ON, FALSE/OFF
  - RHEL4 및 SLES9의 경우 디폴트값:
    TRUE(메모: OLD_CPU_USAGE의 FALSE 설정값은 무시되며 이전 동작만 사용됩니다.)
  - RHEL5, SLES10 및 기타의 경우 디폴트값: FALSE
  - 이 매개변수는 인스턴스가 Linux 플랫폼에서 CPU 사용 시간을 확보하는 방법을 제어합니다. TRUE로 설정된 경우, CPU 사용 시간을 확보하는 이전 메소드가 사용됩니다. 이 메소드는 시스템 및 사용자 CPU 사용 시간을 모두 리턴하지만 이를 수행하는 데 더 많은 CPU를 사용합니다(즉, 오버헤드가 더 높음). FALSE로

설정된 경우, CPU 사용 시간을 확보하는 새 메소드가 사용됩니다. 이 메소드는 사용자 CPU 사용값만 리턴하지만 오버헤드가 적기 때문에 더 빠릅니다.

DISABLE_CPU_USAGE

- 운영 체제: Linux
- 값: TRUE/ON, FALSE/OFF
- RHEL4 및 SLES9의 경우 디폴트값: TRUE
- RHEL5, SLES10 및 기타의 경우 디폴트값: FALSE
- 이 매개변수를 사용하여 CPU 사용이 읽히는지 여부를 판별할 수 있습니다. DISABLE_CPU_USAGE가 사용 가능하면(TRUE로 설정), CPU 사용이 읽히지 않아서 때때로 CPU 사용 검색 중에 발생할 수 있는 오버헤드를 피할 수 있습니다.

DB2TERRITORY

- 운영 체제: 모두
- 디폴트: 운영 체제에서 지정한 대로 언어 ID에서 파생됩니다.
- 이 변수는 클라이언트 응용프로그램의 영역 또는 지역 코드를 지역하여 날짜 및 시간 형식에 영향을 미칩니다.

DB2_VIEW_REOPT_VALUES

- 운영 체제: 모두
- 디폴트: NO, 값: YES, NO
- 이 변수를 사용하면 다시 최적화된 SQL 또는 XQuery문이 설명될 때 모든 사용자가 EXPLAIN_PREDICATE 테이블에 해당 명령문의 캐시된 값을 저장할 수 있습니다. 이 변수가 NO로 설정되면 DBADM만 이러한 변수를 EXPLAIN_PREDICATE 테이블에 저장할 수 있습니다.

시스템 환경 변수

DB2_ALTERNATE_GROUP_LOOKUP

- 운영 체제: AIX
- 디폴트: NULL, 값: NULL 또는 GETGRSET
- 이 변수를 사용하여 DB2는 운영 체제에서 제공한 대체 소스에서 그룹 정보를 확보할 수 있습니다. AIX에서는 getgrset 함수가 사용됩니다. 이 함수는 LAM(Loadable Authentication Module)을 통해 로컬 파일 이외의 위치에서 그룹을 확보하는 기능을 제공합니다.

DB2_CLP_EDITOR

자세한 내용은 『명령행 변수』의 DB2_CLP_EDITOR를 참조하십시오.

DB2_CLP_HISTSIZE

자세한 내용은 『명령행 변수』의 DB2_CLP_HISTSIZE를 참조하십시오.

DB2CONNECT_ENABLE_EURO_CODEPAGE

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- 유로 지원이 필요한 IBM i 서버용 DB2 또는 z/OS 서버용 DB2에 연결되는 모든 DB2 Connect 클라이언트 및 서버에서는 이 변수를 YES로 설정하십시오. 이 변수를 YES로 설정하면 현재 응용프로그램 코드 페이지가 유로 기호에 대한 지원을 명시적으로 표시하는 동등한 코드화된 문자 세트 ID(CCSID)에 맵핑됩니다. 그 결과, DB2 Connect는 현재 응용프로그램 코드의 CCSID 수퍼 세트이고 또한 유로 기호를 지원하는 CCSID를 사용하여 IBM i 서버용 DB2 또는 z/OS 서버용 DB2에 연결됩니다. 예를 들어, 클라이언트가 CCSID 1252에 맵핑되는 코드 페이지를 사용 중인 경우, 클라이언트는 CCSID 5348를 사용하여 연결됩니다.

DB2CONNECT_IN_APP_PROCESS

- 운영 체제: 모두
- 디폴트: YES, 값: YES 또는 NO
- 이 변수를 NO로 설정하면 DB2 Enterprise Server Edition 머신의 로컬 DB2 Connect 클라이언트가 에이전트 내에서 강제 실행됩니다. 에이전트 내에서 실행하면 로컬 클라이언트를 모니터할 수 있고 SYSPLEX 지원을 사용할 수 있는 이점이 있습니다.

DB2_COPY_NAME

- 운영 체제: Windows
- 디폴트: 머신에 설치된 DB2의 디폴트 사본 이름. 값: 머신에 설치된 DB2의 사본 이름. 이름의 길이는 최대 128자입니다.
- **DB2_COPY_NAME** 변수는 현재 사용 중인 DB2 사본의 이름을 저장합니다. 여러 DB2 사본이 머신에 설치된 경우, **DB2_COPY_NAME**을 사용하여 다른 DB2 사본으로 전환할 수 있으며 현재 사용 중인 사본을 변경하려면 *INSTALLPATH*₩bin₩db2envar.bat 명령을 실행해야 합니다.

DB2DBMSADDR

- 운영 체제: x86의 Linux 및 zSeries의 Linux(31비트)
- 디폴트: NULL, 값: 0x09000000 - 0xB0000000 범위의 가상 주소(0x10000씩 증가)
- **DB2DBMSADDR** 레지스트리 변수는 16진수 형식의 디폴트 데이터베이스 공유 메모리 주소를 지정합니다.

주: 올바르지 않은 주소는 DB2 시스템에 대해 DB2 인스턴스 시작 기능 장애부터 데이터베이스에 연결하는 기능에 이르는 심각한 문제를 일으킬 수 있습니다. 올바르지 않은 주소는 이미 사용 중인 메모리의 영역과 충돌하거나 다른 용도로 사용되는 주소입니다. 이 문제점을 처리하려면 다음 명령을 사용하여 **DB2DBMSADDR** 레지스트리 변수를 NULL로 재설정하십시오.

```
db2set DB2DBMSADDR=
```

이 변수를 사용하여 DB2 프로세스의 어드레스 스페이스 레이아웃을 미세 조정할 수 있습니다. 이 변수는 인스턴스 공유 메모리의 위치를 가상 주소 0x10000000의 현재 위치에서 새 값으로 변경합니다.

DB2_DIAGPATH

- 운영 체제: 모두
- 디폴트: 디폴트값은 UNIX 및 Linux 운영 체제에서는 인스턴스 db2dump 디렉토리이며 Windows 운영 체제에서는 인스턴스 db2 디렉토리입니다.
- 이 매개변수는 ODBC 및 DB2 CLI 응용프로그램에만 적용됩니다.

  이 매개변수를 사용하여 DB2 진단 정보의 완전한 경로를 지정할 수 있습니다. 이 디렉토리에는 사용자의 플랫폼에 따라 덤프 파일, 트랩 파일, 오류 로그, 통지 파일 및 경보 로그 파일이 포함될 수 있습니다.

  이 환경 변수 설정은 해당 환경 범위의 ODBC 및 CLI 응용프로그램에 대해 DB2 데이터베이스 관리 프로그램 구성 매개변수 **diagpath** 설정 및 CLI/ODBC 구성 키워드 **DiagPath** 설정과 동일한 효과가 있습니다.

DB2DOMAINLIST

- 운영 체제: 모두
- 디폴트: NULL, 값: 쉼표로 구분된 Windows 도메인 이름 목록(『,』)
- 이 변수는 하나 이상의 Windows 도메인을 정의합니다. 서버에서 유지보수되는 이 목록은 요청 사용자 ID가 인증되는 도메인을 정의합니다. 이러한 도메인에 속하는 사용자에게만 연결 또는 접속 요청이 승인됩니다.

  이 변수는 CLIENT 인증이 데이터베이스 관리 프로그램 구성에 설정된 경우에만 유효합니다. 이 변수는 Windows 데스크탑에서의 단일 로그온이 Windows 도메인 환경에서 필수인 경우에 필요합니다.

  DB2 서버 버전 V7.1 이상에서는 순수 Windows 도메인 환경에서만 **DB2DOMAINLIST**를 지원합니다. 버전 8 FixPak 15 및 버전 9.1 FixPack 3부터 **DB2DOMAINLIST**는 클라이언트 또는 서버가 Windows 환경에서 실행 중인 경우에 지원됩니다.

DB2ENVLIST

- 운영 체제: UNIX
- 디폴트: NULL
- 이 변수는 스토어드 프로시저 또는 사용자 정의 함수(UDF)의 특정 변수 이름을 나열합니다. 디폴트로 db2start 명령은 **DB2** 또는 **db2**가 앞에 붙은 경우를 제외한 모든 사용자 환경 변수를 필터링합니다. 특정 환경 변수를 스토어드 프로시저 또는 사용자 정의 함수에 전달해야 하는 경우, **DB2ENVLIST** 환경 변수에 변수 이름을 나열할 수 있습니다. 하나 이상의 공백으로 각 변수 이름을 구분하십시오.

DB2INSTANCE

- 운영 체제: 모두
- 디폴트: Windows 32비트 운영 체제의 경우 **DB2INSTDEF**.
- 이 환경 변수는 디폴트로 사용 중인 인스턴스를 지정합니다. UNIX에서는 사용자가 **DB2INSTANCE**의 값을 지정해야 합니다.

DB2INSTPROF

- 운영 체제: Windows
- 디폴트: `Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥`*`Copy Name`*(Windows XP, Windows 2003), `ProgramData¥IBM¥DB2¥`*`Copy Name`*(Windows Vista)
- 이 환경 변수는 Windows 운영 체제에서 인스턴스 디렉토리의 위치를 지정합니다. 버전 9.5부터 인스턴스 디렉토리(및 기타 사용자 데이터 파일)는 `sqllib` 디렉토리 아래에 있을 수 없습니다.

DB2LDAPSecurityConfig

- 운영 체제: 모두
- 디폴트: NULL, 값: IBM LDAP 보안 플러그인 구성 파일의 유효한 이름 및 경로
- 이 변수는 IBM LDAP 보안 플러그인 구성 파일의 위치를 지정하는 데 사용됩니다. 변수가 설정되지 않은 경우, IBM LDAP 보안 플러그인 구성 파일은 `IBMLDAPSecurity.ini`로 이름 지정되고 다음 중 한 위치에 있습니다.
  - Linux 및 UNIX 운영 체제의 경우: *`INSTHOME`*`/sqllib/cfg/`
  - Windows 운영 체제의 경우: **`%DB2PATH%¥cfg¥`**

  Windows 운영 체제의 경우, DB2 서비스에서 이 변수를 선택하도록 하려면 이를 전역 시스템 환경에 설정해야 합니다.

DB2LIBPATH

- 운영 체제: UNIX
- 디폴트: NULL

- DB2는 고유의 공유 라이브러리 경로를 구성합니다. PATH를 엔진의 라이브러리 경로에 추가하려면 **DB2LIBPATH**를 설정해야 합니다(예를 들어, AIX에서는 사용자 정의 함수에 LIBPATH의 특정 항목이 필요함). **DB2LIBPATH**의 실제 값은 DB2 구성 공유 라이브러리 경로의 끝에 추가됩니다.

DB2LOGINRESTRICTIONS

- 운영 체제: AIX
- 디폴트: LOCAL, 값: LOCAL, REMOTE, SU, NONE
- 이 레지스트리 변수를 사용하여 AIX 운영 체제 API loginrestrictions()를 사용할 수 있습니다. 이 API는 사용자가 시스템에 액세스할 수 있는지 여부를 판별합니다. 이 API를 호출하여 DB2 데이터베이스 보안은 운영 체제에서 지정한 로그인 제한사항을 강제 실행할 수 있습니다. 이 레지스트리 변수를 사용할 때 이 API에 제출할 수 있는 여러 가지 값이 있습니다. 값은 다음과 같습니다.
  - REMOTE

    DB2가 *rlogind* 또는 *telnetd* 프로그램을 통해 어카운트를 리모트 로그인에 사용할 수 있는지 검증하는 로그인 제한사항만 강제 실행합니다.
  - SU

    DB2 버전 9.1이 su 명령이 허용되고 현재 프로세스에 su 명령을 호출하여 어카운트로 전환할 수 있는 그룹 ID가 있는지 검증하는 su 제한사항만 강제 실행합니다.
  - NONE

    DB2가 로그인 제한사항을 강제 실행하지 않습니다.
  - LOCAL(또는 변수가 설정되지 않음)

    DB2가 이 어카운트에 대해 로컬 로그인이 허용되었는지 검증하는 로그인 제한사항만 강제 실행합니다. 이는 로그인할 때 일반 동작입니다.

  어떠한 옵션을 설정했든 지정된 특권을 가지고 있는 사용자 어카운트 또는 ID는 서버에서 로컬로 또는 리모트 클라이언트에서 DB2를 사용할 수 있습니다. loginrestrictions() API에 대한 설명은 AIX 문서를 참조하십시오.

DB2NODE

- 운영 체제: 모두
- 디폴트: NULL, 값: 1 - 999
- 접속 또는 연결하려는 데이터베이스 파티션 서버의 목표 논리 노드를 지정하는 데 사용됩니다. 이 변수가 설정되지 않은 경우, 목표 논리 노드는 디폴트

로 머신에서 0 포트로 정의된 논리 노드로 설정됩니다. 파티션된 데이터베이스 환경에서 연결 설정은 트러스트된 연결 획득에 영향을 미칠 수 있습니다. 예를 들어, **DB2NODE** 변수가 노드에서 연결을 설정하려면 중간 노드(홉 노드)를 통과해야 하는 노드로 설정된 경우, 이 연결이 트러스트된 연결로 표시될 수 있는지 여부를 판별하기 위해 해당 연결을 평가할 때 고려하는 사항은 해당 중간 노드의 IP 주소 및 홉 노드와 연결 노드 간의 통신에 사용되는 통신 프로토콜입니다. 즉, 연결이 시작된 원래 노드는 고려되지 않습니다. 대신에 홉 노드가 고려됩니다.

DB2OPTIONS

- 운영 체제: 모두
- 디폴트: NULL
- 명령행 처리기 옵션을 설정하는 데 사용됩니다.

DB2_PARALLEL_IO

- 운영 체제: 모두
- 디폴트: NULL, 값: *TablespaceID*:[*n*],... – 쉼표로 구분된 정의된 테이블 스페이스 목록(숫자 테이블 스페이스 ID로 식별됨). 테이블 스페이스의 프리페치 *크기*가 AUTOMATIC인 경우, 테이블 스페이스 ID, 콜론, 컨테이너당 디스크 수 *n*을 차례로 지정하여 DB2 데이터베이스 관리 프로그램에 컨테이너당 디스크 수를 표시할 수 있습니다. *n*이 지정되지 않은 경우, 디폴트는 6입니다.

  *TablespaceID*를 별표(*)로 바꿔 모든 테이블 스페이스를 지정할 수 있습니다. 예를 들어, **DB2_PARALLEL_IO** =*인 경우, 모든 테이블 스페이스에서 6을 컨테이너당 디스크 수로 사용합니다. 별표(*)와 테이블 스페이스 ID를 모두 지정하는 경우, 테이블 스페이스 ID 설정이 우선합니다. 예를 들어, **DB2_PARALLEL_IO** =*,1:3인 경우, 컨테이너당 디스크 수로 3을 사용하는 테이블 스페이스 1을 제외하고 모든 테이블 스페이스에서 6을 컨테이너당 디스크 수로 사용합니다.
- 이 레지스트리 변수는 DB2가 테이블 스페이스의 입출력 병렬 처리를 계산하는 방식을 변경하는 데 사용됩니다. 입출력 병렬 처리가 사용 가능한 경우(다중 컨테이너 사용으로 내재적으로 또는 **DB2_PARALLEL_IO** 설정으로 명시적으로), 올바른 프리페치 요청 수를 발행하여 수행됩니다. 각 프리페치 요청은 페이지 Extent에 대한 요청입니다. 예를 들어, 테이블 스페이스에 두 개의 컨테이너가 있고 프리페치 *크기*는 Extent *크기*의 네 배입니다. 레지스트리 변수가 설정된 경우, 이 테이블 스페이스에 대한 프리페치 요청은 네 개의 요청(요청당 하나의 Extent)으로 구분되며 네 개의 프리페처가 병렬로 요청을 서비스할 수 있습니다.

테이블 스페이스의 개별 컨테이너가 여러 개의 실제 디스크에 걸쳐서 스트라이프되거나 테이블 스페이스의 컨테이너가 둘 이상의 실제 디스크로 구성된 단일 RAID 디바이스에서 작성되는 경우에 이 레지스트리 변수를 설정하려고 하게 됩니다.

이 레지스트리 변수가 설정되지 않은 경우, 테이블 스페이스의 병렬 처리 수준은 테이블 스페이스의 컨테이너 수입니다. 예를 들어, **DB2_PARALLEL_IO**가 NULL로 설정되고 테이블 스페이스에 네 개의 컨테이너가 있는 경우, 네 개의 Extent *크기* 프리페치 요청이 발행됩니다. 또는 테이블 스페이스에 두 개의 컨테이너가 있고 프리페치 *크기*가 Extent 크기의 네 배인 경우, 이 테이블 스페이스에 대한 프리페치 요청은 두 개의 요청으로 구분됩니다(각 요청은 두 Extent에 대한 것임).

이 레지스트리 변수가 설정되고 테이블의 프리페치 *크기*가 AUTOMATIC이 아닌 경우, 테이블 스페이스의 병렬 처리 수준은 Extent *크기*로 나눈 프리페치 *크기*입니다. 예를 들어, **DB2_PARALLEL_IO**가 프리페치 *크기*가 160이고 Extent *크기*가 32페이지인 테이블 스페이스에 대해 설정된 경우, 다섯 개의 Extent *크기* 프리페치 요청이 발행됩니다.

이 레지스트리 변수가 설정되고 테이블 스페이스의 프리페치 *크기*가 AUTOMATIC인 경우, DB2는 다음의 등식을 사용하여 테이블 스페이스의 프리페치 *크기*를 자동으로 계산합니다.

```
프리페치 크기 =
 (컨테이너 수) * (컨테이너당 디스크 수)
  * Extent 크기
```

콜론 뒤의 숫자는 DB2가 등식의 *컨테이너당 디스크 수*에 사용합니다. 별표만 사용되고 숫자는 지정되지 않은 경우, 디폴트인 컨테이너당 6개의 디스크가 사용됩니다.

다음 표에서는 사용 가능한 여러 가지 옵션 및 각 상황에서 병렬 처리가 계산되는 방법에 대해 설명합니다.

표 67. 병렬 처리 계산법

테이블 스페이스의 프리페치 *크기*	**DB2_PARALLEL_IO 설정**	**병렬 처리 방법**
AUTOMATIC(프리페치 *크기* = 컨테이너 수 * 1 * Extent *크기*)	설정되지 않음	컨테이너 수
AUTOMATIC(프리페치 *크기* = 컨테이너 수 * 6 * Extent *크기*)	*테이블 스페이스 ID*	컨테이너 수 * 6
AUTOMATIC(프리페치 *크기* = 컨테이너 수 * *n* * Extent *크기*)	*테이블 스페이스 ID*:*n*	컨테이너 수 * *n*
자동 아님	설정되지 않음	컨테이너 수
자동 아님	*테이블 스페이스 ID*	프리페치 *크기*/Extent 크기

표 67. 병렬 처리 계산법 (계속)

테이블 스페이스의 프리페치 크기	DB2_PARALLEL_IO 설정	병렬 처리 방법
자동 아님	*테이블 스페이스 ID*:*n*	프리페치 크기/Extent 크기

예를 들어, 세 개의 테이블 스페이스가 있고 ID가 각각 3, 4 및 5인 경우를 고려하십시오. Extent 크기는 모두 4,096바이트이고 모두 각각 두 개의 컨테이너를 가지고 있습니다. 테이블 스페이스 3 및 4의 프리페치 크기는 모두 AUTOMATIC이고 테이블 스페이스 5의 프리페치 크기는 16,384바이트입니다. **DB2_PARALLEL_IO**=*:5,4:10으로 설정하면 다음과 같은 테이블 스페이스의 병렬 처리를 얻습니다.

- 테이블 스페이스 3: *n*(컨테이너당 디스크 수) 값은 5, Extent 크기는 4,096, 컨테이너 수는 2이고 프리페치 크기는 AUTOMATIC입니다. 따라서 프리페치 크기는 2*5*4,096이고 병렬 처리=컨테이너 수 * *n*=2*5=10입니다.
- 테이블 스페이스 4: 이 테이블 스페이스에 대해 *n*(컨테이너당 디스크 수) 값은 특정적으로 10으로 설정됩니다. Extent 크기는 4096, 컨테이너 수는 2, *n*은 10이고 프리페치 크기는 AUTOMATIC입니다. 따라서 프리페치 크기는 2*10*4,096이고 병렬 처리=컨테이너 수 * *n*=2*10=20입니다.
- 테이블 스페이스 5: *n* 값은 여전히 5이지만 프리페치 크기가 AUTOMATIC이 아니므로 적용되지 않습니다. Extent 크기는 4096, 컨테이너 수는 2, 프리페치 크기는 16384입니다. 따라서 병렬 처리=프리페치 크기/Extent 크기=16,384/4,096=4입니다.

일부 시나리오에서는 이 변수를 사용하면 그 결과 디스크 장치 경쟁이 발생합니다. 예를 들어, 테이블 스페이스에 두 개의 컨테이너가 있고 두 컨테이너에 각각 전용 단일 디스크가 있는 경우, 레지스트리 변수를 설정하면 그 결과 두 개의 프리페처가 동시에 두 디스크 각각에 액세스하게 되므로 해당 디스크에서 경쟁이 발생합니다. 그러나 각 컨테이너가 여러 디스크에서 스트라이프된 경우에 레지스트리 변수를 설정하면 동시에 네 개의 다른 디스크에 액세스할 가능성이 있습니다.

이 레지스트리 변수에 대한 변경사항을 활성화하려면 db2stop 명령을 발행한 다음 db2start 명령을 입력하십시오.

DB2PATH

- 운영 체제: Windows
- 디폴트: 운영 체제에 따라 다름

- 이 환경 변수는 제품이 Windows 32비트 운영 체제에서 설치되는 디렉토리를 지정하는 데 사용됩니다.

DB2_PMAP_COMPATIBILITY

- 운영 체제: 모두
- 디폴트: ON, 값: ON 또는 OFF
- 사용자는 이 변수를 사용하면 sqlugtpi 및 sqlugrpn API를 계속 사용하여 테이블에 대한 분산 정보와 행에 대한 데이터베이스 파티션 번호 및 데이터베이스 파티션 서버 번호를 각각 리턴할 수 있습니다. 디폴트 설정(ON)은 분산 맵 크기가 4,096개 항목을 유지함을 표시합니다(버전 9.7 이전 동작). 이 변수가 OFF로 설정된 경우, 새 데이터베이스 또는 업그레이된 데이터베이스의 분산 맵 크기는 32,768개 항목으로 늘어납니다(버전 9.7 동작). 32K 분산 맵을 사용하는 경우, 새 db2GetDistMap 및 db2GetRowPartNum API를 사용해야 합니다.

DB2PROCESSORS

- 운영 체제: Windows
- 디폴트: NULL, 값: 0–*n*-1(여기서 *n*=프로세서 수)
- 이 변수는 특정 db2syscs 프로세스에 대한 프로세스 친화도 마스크를 설정합니다. 여러 개의 논리 노드가 실행되는 환경에서 이 변수는 논리 노드를 프로세서 또는 프로세서 세트에 연관시키는 데 사용됩니다.

  이 변수가 지정되면 DB2가 SetProcessAffinityMask() API를 발행합니다. 이 변수가 지정되지 않으면 db2syscs 프로세스가 서버의 모든 프로세서와 연관됩니다.

DB2RCMD_LEGACY_MODE

- 운영 체제: Windows,
- 디폴트: NULL, 값: YES, ON, TRUE 또는 1 또는 NO, OFF, FALSE 또는 0
- 이 변수를 사용하여 사용자는 DB2 리모트 명령 서비스의 확장된 보안을 사용하거나 사용하지 않을 수 있습니다. 보안 방식으로 DB2 리모트 명령 서비스를 실행하려면 **DB2RCMD_LEGACY_MODE**를 NO, OFF, FALSE, 0 또는 NULL로 설정하십시오. Legacy 모드(확장된 보안을 사용하지 않음)에서 실행하려면 **DB2RCMD_LEGACY_MODE**를 YES, ON, TRUE 또는 1로 설정하십시오. 보안 모드는 도메인 제어기가 Windows 2000 이상을 실행 중인 경우에만 사용 가능합니다.

  주: DB2RCMD_LEGACY_MODE가 YES, ON, TRUE 또는 1로 설정된 경우, DB2 리모트 명령 서비스로 전송된 모든 요청은 요청자의 컨텍스트 아래에서 처리됩니다. 이를 쉽게 하려면 도메인 제어기에서 머신 또는 서비

스 로그온 어카운트를 사용 가능하게 하여 머신 또는 서비스 로그온 어카운트 또는 모두 클라이언트를 가장할 수 있도록 허용해야 합니다.

주: DB2RCMD_LEGACY_MODE가 NO, OFF, FALSE 또는 0으로 설정된 경우, DB2 리모트 명령 서비스 실행 명령을 가지려면 SYSADM 권한이 있어야 합니다.

DB2RESILIENCE

- 운영 체제: 모두
- 디폴트: ON, 값: ON(TRUE 또는 1) 또는 OFF(FALSE 또는 0)
- 이 레지스트리 변수를 사용하여 실제 읽기 오류가 허용되고 확장된 트랩 복구를 활성화하는지 여부를 제어할 수 있습니다. 디폴트 동작은 읽기 오류를 허용하고 확장된 트랩 복구를 활성하는 것입니다. 이전 릴리스의 동작으로 되돌리고 데이터베이스 관리 프로그램이 인스턴스를 종료하도록 강제 실행하려면 이 레지스트리 변수를 OFF로 설정하십시오. 이 레지스트리 변수는 기존 스토리지 키 지원에 영향을 미치지 않습니다.

DB2SYSTEM

- 운영 체제: Windows 및 UNIX
- 디폴트: NULL
- 사용자 및 데이터베이스 관리자가 DB2 서버 시스템을 식별하는 데 사용할 이름을 지정합니다. 가능한 경우, 이 이름은 네트워크 내에서 고유해야 합니다.

  이 이름은 제어 센터 오브젝트 트리의 시스템 레벨에 표시되어 관리자가 제어 센터에서 관리할 수 있는 서버 시스템을 식별하도록 도와줍니다.

  구성 지원 프로그램의 네트워크 검색 기능을 사용하는 경우, DB2 발견은 이 이름을 리턴하며 결과 오브젝트 트리의 시스템 레벨에 이 이름이 표시됩니다. 이 이름은 사용자가 액세스하려는 데이터베이스가 있는 시스템을 식별하는 데 도움이 됩니다. **DB2SYSTEM**의 값은 설치 시 다음과 같이 설정됩니다.
  - Windows의 경우 설치 프로그램은 이 값을 Windows 시스템에 지정된 컴퓨터 이름과 동일하게 설정합니다.
  - UNIX 시스템의 경우 이 값은 UNIX 시스템의 TCP/IP 호스트 이름과 동일하게 설정됩니다.

DB2_UPDDBCFG_SINGLE_DBPARTITION

- 운영 체제: 모두
- 디폴트: 설정되지 않음, 값: 0/FALSE/NO, 1/TRUE/YES

- 1, TRUE 또는 YES로 설정되면 이 레지스트리 변수를 사용하여 데이터베이스에 대한 갱신사항 및 재설정이 특정 파티션에만 영향을 주도록 지정할 수 있습니다. 이 변수가 설정되지 않으면 갱신사항 및 요청이 버전 9.5 동작을 따릅니다.
- 버전 9.5부터 파티션 절을 지정하지 않으면 데이터베이스 구성에 대한 갱신사항 또는 변경사항이 모든 데이터베이스 파티션에 걸쳐서 작동합니다. **DB2_UPDDBCFG_SINGLE_DBPARTITION**을 사용하여 데이터베이스 구성에 대한 갱신사항이 **DB2NODE** 레지스트리 변수로 설정한 데이터베이스 파티션 또는 로컬 데이터베이스 파티션에만 적용되는 DB2 이전 버전의 동작으로 되돌릴 수 있습니다. 이는 이 동작을 요구하는 기존 명령 스크립트 또는 응용프로그램에 대한 이전 버전과의 호환을 지원합니다.

주: 이 변수는 ADMIN_CMD 루틴 호출로 작성되는 갱신 또는 재설정 요청에는 적용되지 않습니다.

DB2_USE_PAGE_CONTAINER_TAG

- 운영 체제: 모두
- 디폴트:NULL, 값: ON, NULL
- 디폴트로 DB2는 DMS 컨테이너가 파일이든 디바이스든 각 컨테이너의 첫 번째 Extent에 컨테이너 태그를 저장합니다. 컨테이너 태그는 컨테이너의 메타데이터입니다. DB2 버전 8.1 이전에는 컨테이너 태그가 단일 페이지에 저장되었고 따라서 컨테이너의 스페이스가 적게 필요했습니다. 컨테이너 태그를 계속 단일 페이지에 저장하려면 **DB2_USE_PAGE_CONTAINER_TAG**를 ON으로 설정하십시오.

  그러나 컨테이너에 RAID 디바이스를 사용할 때 이 레지스트리 변수를 ON으로 설정하면 입출력 성능이 떨어집니다. RAID 디바이스의 경우 Extent 크기가 RAID 스트라이프 크기와 같거나 이의 배수인 테이블 스페이스를 작성하므로 **DB2_USE_PAGE_CONTAINER_TAG**를 ON으로 설정하면 Extent가 RAID 스트라이프와 정렬하지 않게 됩니다. 그 결과, 입출력 요청이 최적의 경우보다 더 많은 실제 디스크에 액세스해야 합니다. 매우 엄격한 스페이스 제한조건이 있거나 예비 버전 8 데이터베이스와 일치하는 동작이 필요한 경우 외에는 이 레지스트리 변수를 사용하지 않는 것이 좋습니다.

  이 레지스트리 변수에 대한 변경사항을 활성화하려면 db2stop 명령을 발행한 다음 db2start 명령을 입력하십시오.

DB2_WORKLOAD

- 운영 체제: 모두
- 디폴트: 설정되지 않음, 값: 1C, CM, SAP, TPM, WC

- **DB2_WORKLOAD**의 각 값은 사전 정의된 설정이 있는 여러 가지 레지스트리 변수의 특정 그룹을 나타냅니다.
- 유효한 값은 다음과 같습니다.

  1C 1C 응용프로그램에 대한 레지스트리 변수 세트를 데이터베이스에 구성하려면 이 설정을 사용하십시오.

  CM IBM Content Manager에 대한 레지스트리 변수 세트를 데이터베이스에 구성하려면 이 설정을 사용하십시오.

  SAP SAP 환경에 대한 레지스트리 변수 세트를 데이터베이스에 구성하려면 이 설정을 사용하십시오.

  DB2_WORKLOAD=SAP를 설정하면 사용자 테이블 스페이스 SYSTOOLSPACE 및 사용자 임시 테이블 스페이스 SYSTOOLSTMPSPACE는 자동으로 작성되지 않습니다. 이러한 테이블 스페이스는 다음의 마법사, 유틸리티 또는 함수에 의해 자동으로 작성되는 테이블에 사용됩니다.

  – 자동 유지보수
  – 디자인 어드바이저
  – 제어 센터 데이터베이스 정보 패널
  – 테이블 스페이스 입력 매개변수가 지정되지 않은 경우 SYSINSTALLOBJECTS 스토어드 프로시저
  – GET_DBSIZE_INFO 스토어드 프로시저

  SYSTOOLSPACE 및 SYSTOOLSTMPSPACE 테이블 스페이스가 없으면 이러한 마법사, 유틸리티 또는 함수를 사용할 수 없습니다.

  이러한 마법사, 유틸리티 또는 함수를 사용할 수 있으려면 다음 중 하나를 수행하십시오.

  – 도구에 필요한 오브젝트를 보유할 SYSTOOLSPACE 테이블 스페이스를 수동으로 작성하십시오(파티션된 데이터베이스 환경에서는 이 테이블 스페이스를 카탈로그 파티션에 작성하십시오). 예를 들어, 다음과 같습니다.

    ```
    CREATE REGULAR TABLESPACE SYSTOOLSPACE
    IN IBMCATGROUP
    MANAGED BY SYSTEM
    USING ('SYSTOOLSPACE')
    ```

  – 유효한 테이블 스페이스를 지정하고 SYSINSTALLOBJECTS 스토어드 프로시저를 호출하여 도구에 대한 오브젝트를 작성한 다음 특정 도구에 대한 ID를 지정하십시오.
    SYSINSTALLOBJECTS는 사용자를 위해 테이블 스페이스를 작

성합니다. 오브젝트에 대해 SYSTOOLSSPACE를 사용하지 않으려면 다른 사용자 정의 테이블 스페이스를 지정하십시오.

해당 선택사항 중 하나 이상을 완료한 후 SYSTOOLSTMPSPACE 임시 테이블 스페이스를 작성하십시오(파티션된 데이터베이스 환경에서 작업 중인 경우 역시 카탈로그 파티션에 작성). 예를 들어, 다음과 같습니다.

```
CREATE USER TEMPORARY TABLESPACE SYSTOOLSTMPSPACE
IN IBMCATGROUP
MANAGED BY SYSTEM
USING ('SYSTOOLSTMPSPACE')
```

테이블 스페이스 SYSTOOLSPACE 및 임시 테이블 스페이스 SYSTOOLSTMPSPACE가 작성되면 앞에서 설명한 마법사, 유틸리티 또는 함수를 사용할 수 있습니다.

TPM Tivoli Provisioning Manager에 대한 레지스트리 변수 세트를 데이터베이스에 구성하려면 이 설정을 사용하십시오.

WC Websphere Commerce에 대한 레지스트리 변수 세트를 데이터베이스에 구성하려면 이 설정을 사용하십시오.

통신 변수

DB2CHECKCLIENTINTERVAL

- 운영 체제: 모두, 서버 전용
- 디폴트=50, 값: 0 이상인 숫자 값
- 이 변수는 TCP/IP 클라이언트 연결 검증 빈도를 지정합니다. 이 변수를 사용하면 쿼리가 완료될 때까지 대기하는 대신에 클라이언트 종료를 일찍 발견할 수 있습니다. 이 변수가 0으로 설정되는 경우, 검증이 수행되지 않습니다.

  값이 낮으면 더 자주 점검합니다. 지침에 따라 낮은 빈도에는 100을 사용하고 중간 빈도에는 50, 높은 빈도에는 10을 사용하십시오. 값은 내부 DB2 메트릭으로 측정됩니다. 값은 선형 스케일을 나타냅니다. 즉, 50에서 100으로 값이 증가하면 간격이 두 배로 늘어납니다. 데이터베이스 요청을 실행하는 동안 클라이언트 상태를 자주 점검하면 쿼리를 완료하는 데 걸리는 시간이 길어집니다. DB2 워크로드가 과중한 경우(즉, 여러 가지 내부 요청이 포함된 경우), **DB2CHECKCLIENTINTERVAL**을 낮은 값으로 설정하면 워크로드가 적은 상황에서보다 성능에 더 큰 영향을 미칩니다.

  DB2 Universal Database 버전 8.1.4 이후
  DB2CHECKCLIENTINTERVAL의 디폴트값은 50입니다. 버전 8.1.4 이전에 디폴트값은 0이었습니다.

DB2COMM

- 운영 체제: 모두, 서버 전용
- 디폴트=NULL, 값: NPIPE, TCPIP, SSL
- 이 변수는 데이터베이스 관리 프로그램이 시작될 때 시작되는 통신 관리 프로그램을 지정합니다. 이 변수가 설정되지 않은 경우, DB2 통신 관리 프로그램이 서버에서 시작되지 않습니다.

DB2FCMCOMM

- 운영 체제: 지원되는 모든 DB2 Enterprise Server Edition 플랫폼
- 디폴트=TCPIP4, 값: TCPIP4 또는 TCPIP6
- 이 변수는 db2nodes.cfg 파일의 호스트 이름이 분석되는 방법을 지정합니다. 모든 호스트 이름은 IPv4 또는 IPv6로 분석됩니다. 호스트 이름 대신에 IP 주소가 db2nodes.cfg에 지정된 경우, IP 형식으로 IPv4가 사용되었는지 또는 IPv6가 사용되었는지를 판별합니다. **DB2FCMCOMM**이 설정되지 않은 경우, 디폴트 설정인 IPv4는 IPv4 호스트만 시작할 수 있음을 의미합니다.

  주: db2nodes.cfg에 지정된 호스트 이름에서 분석된 IP 형식 또는 db2nodes.cfg에 직접 지정된 IP 형식이 **DB2FCMCOMM**의 설정과 일치하지 않는 경우, db2start는 실패합니다.

DB2_FORCE_NLS_CACHE

- 운영 체제: AIX, HP_UX, Solaris
- 디폴트=FALSE, 값: TRUE 또는 FALSE
- 이 변수는 다중 스레드 응용프로그램에서 잠금 경합 가능성을 제거하는 데 사용됩니다. 이 레지스트리 변수가 TRUE인 경우, 코드 페이지 및 지역 코드 정보는 스레드가 해당 정보에 처음 액세스할 때 저장됩니다. 이후 캐시된 정보는 해당 정보를 요청하는 다른 모든 스레드에 사용됩니다. 이로 인해 잠금 경합이 제거되고 그 결과 특정 상황에서 성능이 높아집니다. 응용프로그램이 연결 간 로케일 설정을 변경하는 경우에는 이 설정을 사용하면 안 됩니다. 로케일 설정을 변경하는 것이 *스레드 안전*하지 않기 때문에 다중 스레드 응용프로그램은 일반적으로 로케일 설정을 변경하지 않으므로 이러한 상황에서는 이 설정이 필요하지 않을 수 있습니다.

DB2RSHCMD

- 운영 체제: UNIX
- 디폴트=rsh(HP-UX의 경우 remsh), 값은 rsh, remsh 또는 ssh의 전체 경로 이름임
- 디폴트로 DB2 데이터베이스 시스템은 리모트 데이터베이스 파티션을 시작하고 db2_all 스크립트를 사용하여 모든 데이터베이스 파티션에서 유틸리티

및 명령을 실행할 때 통신 프로토콜로 rsh를 사용합니다. 예를 들어, 이 레지스트리 변수를 ssh의 전체 경로 이름으로 설정하면 DB2 데이터베이스 제품이 ssh를 요청된 유틸리티 및 명령 실행을 위한 통신 프로토콜로 사용하게 됩니다. 이 레지스트리 변수를 해당 디폴트 매개변수를 포함한 리모트 명령 프로그램을 호출하는 스크립트의 전체 이름 경로로 설정할 수도 있습니다. 이 변수는 DB2 제품이 설치된 서버와 다른 서버에서 db2start 명령이 실행되는 단일 파티션 환경 또는 파티션된 데이터베이스에만 필요합니다. 인스턴스 소유자는 추가 검증 또는 인증(즉, 암호 또는 암호 구문)을 프롬프트하지 않고 각 DB2 데이터베이스 노드에서 다른 각 DB2 데이터베이스 노드에 로그인하기 위해 지정된 리모트 쉘 프로그램을 사용할 수 있어야 합니다.

DB2에 ssh 쉘을 사용하기 위해 DB2RSHCMD 레지스트리 변수 설정에 대한 자세한 지시사항은 백서 “Configure DB2 Universal Database for UNIX to use OpenSSH″를 참조하십시오.

DB2RSHTIMEOUT

- 운영 체제: UNIX
- 디폴트=30초, 값: 1 - 120
- 이 변수는 **DB2RSHCMD**가 널(NULL)이 아닌 값으로 설정된 경우에만 적용 가능합니다. 이 레지스트리 변수는 DB2 데이터베이스 시스템이 리모트 명령을 대기하는 시간종료 기간을 제어하는 데 사용됩니다. 이 시간종료 기간 후 응답이 수신되지 않으면 리모트 데이터베이스 파티션에 접근할 수 없으며 조작이 실패했다고 가정합니다.

  주: 지정된 시간 값은 리모트 명령을 실행하는 데 필요한 시간이 아니라 요청을 인증하는 데 필요한 시간입니다.

DB2SORCVBUF

- 운영 체제: 모두
- 디폴트=65,536
- TCP/IP 수신 버퍼 값을 지정합니다.

DB2SOSNDBUF

- 운영 체제: 모두
- 디폴트=65,536
- TCP/IP 송신 버퍼 값을 지정합니다.

DB2TCP_CLIENT_CONTIMEOUT

- 운영 체제: 모두, 클라이언트 전용
- 디폴트=0(시간종료 없음), 값: 0 - 32,767초

- **DB2TCP_CLIENT_CONTIMEOUT** 레지스트리 변수는 TCP/IP 연결 조작 시 클라이언트가 완료를 대기하는 초 수를 지정합니다. 지정된 초 내에 연결이 설정되지 않는 경우, DB2 데이터베이스 관리 프로그램은 -30081 selectForConnectTimeout 오류를 리턴합니다.

  레지스트리 변수가 설정되지 않거나 0으로 설정된 경우 시간종료는 없습니다.

  주: 운영 체제에는 또한 **DB2TCP_CLIENT_CONTIMEOUT**을 사용하여 설정한 시간종료 전에 적용되는 연결 시간종료 값이 있습니다. 예를 들어, AIX에는 75초 후 연결을 종료하는 디폴트 *tcp_keepinit*=150(0.5초 단위)이 있습니다.

DB2TCP_CLIENT_RCVTIMEOUT

- 운영 체제: 모두, 클라이언트 전용
- 디폴트=0(시간종료 없음), 값: 0 - 32,767초
- **DB2TCP_CLIENT_RCVTIMEOUT** 레지스트리 변수는 TCP/IP 수신 조작 시 클라이언트가 데이터를 대기하는 초 수를 지정합니다. 지정된 초 내에 서버의 데이터가 수신되지 않는 경우, DB2 데이터베이스 관리 프로그램은 -30081 selectForRecvTimeout 오류를 리턴합니다.

  레지스트리 변수가 설정되지 않거나 0으로 설정된 경우 시간종료는 없습니다.

  주: DB2TCP_CLIENT_RCVTIMEOUT의 값을 db2cli.ini 키워드 ReceiveTimeout 또는 연결 속성 SQL_ATTR_RECEIVE_TIMEOUT을 사용하여 CLI로 겹쳐쓸 수 있습니다.

DB2TCPCONNMGRS

- 운영 체제: 모두
- 디폴트=직렬 머신의 경우 1, 대칭형 멀티프로세서 머신의 경우 최대 16개의 연결 관리 프로그램으로 반올림되는 프로세서 수의 제곱근. 값: 1 - 16
- 레지스트 변수가 설정되지 않은 경우, 디폴트 수의 연결 관리 프로그램이 작성됩니다. 레지스트리 변수가 설정된 경우, 여기에 지정된 값이 디폴트값을 겹쳐씁니다. 최대 16으로 지정된 수의 TCP/IP 연결 관리 프로그램이 작성됩니다. 1 미만이 지정되면 **DB2TCPCONNMGRS**의 값은 1로 설정되고 값이 범위밖에 있다는 경고가 로그됩니다. 16 이상이 지정되면 **DB2TCPCONNMGRS**이 값은 16으로 설정되고 값이 범위밖에 있다는 경고가 로그됩니다. 1과 16 사이의 값이 지정 값으로 사용됩니다. 둘 이상의 연결 관리 프로그램이 작성되어 있으면 여러 클라이언트 연결이 동시에 수신될 때 연결 처리량이 향상됩니다. 사용자가 SMP 머신에서 실행 중이거나 **DB2TCPCONNMGRS** 레지스트리 변수를 수정한 경우, 추가 TCP/IP 연

결 관리 프로그램 프로세스(UNIX의 경우) 또는 스레드(Windows 운영 체제의 경우)가 있습니다. 추가 프로세스 또는 스레드에는 추가 스토리지가 필요합니다.

주: 연결 관리 프로그램 수를 1로 설정하면 사용자가 여러 명인 시스템, 빈번한 연결 및 연결 끊기가 있는 시스템 또는 두 가지 모두 해당되는 시스템에서 리모트 연결 시 성능이 떨어지는 원인이 됩니다.

명령행 변수

DB2BQTIME

- 운영 체제: 모두
- 디폴트=1초, 최소값: 1초
- 이 변수는 명령행 처리기 프론트엔드에서 백엔드 프로세스가 사용 중이고 해당 프론트엔드에 대한 연결을 설정하는지 여부를 점검하기 전에 해당 프론트엔드가 유휴하는 시간 크기를 지정합니다.

DB2BQTRY

- 운영 체제: 모두
- 디폴트=재시도 60회, 최소값: 재시도 0회
- 이 변수는 명령행 처리기 프론트엔드 프로세스에서 백엔드 프로세스가 이미 사용 중인지 여부를 판별하려고 시도하는 횟수를 지정합니다. 이 변수는 **DB2BQTIME**과 함께 작동합니다.

DB2_CLP_EDITOR

- 운영 체제: 모두
- 디폴트: 메모장(Windows), vi(UNIX), 값: 운영 체제 경로에 있는 모든 유효한 편집기

  주: 이 레지스트리 변수는 설치 중에 디폴트값으로 설정되지 않습니다. 대신에 레지스트리 변수가 설정되지 않은 경우 이 변수를 사용하는 코드가 디폴트값을 사용합니다.
- 이 변수는 EDIT 명령을 실행할 때 사용되는 편집기를 판별합니다. CLP 대화식 세션에서 EDIT 명령은 편집 및 실행될 수 있는 사용자 지정 명령으로 사전 로드된 편집기를 시작합니다.

DB2_CLP_HISTSIZE

- 운영 체제: 모두
- 디폴트: 20, 값: 1–500(두 값 포함)

주: 이 레지스트리 변수는 설치 중에 디폴트값으로 설정되지 않습니다. 대신에 레지스트리 변수가 설정되지 않았거나 유효한 범위밖의 값으로 설정된 경우, 이 변수를 사용하는 코드가 디폴트값 20을 사용합니다.

- 이 변수는 CLP 대화식 세션 중에 명령 실행기록에 저장되는 명령 수를 판별합니다. 명령 실행기록은 메모리에 보유되므로 이 변수의 값이 매우 높으면 세션에서 실행된 명령 수 및 길이에 따라 성능에 영향을 미칩니다.

DB2_CLPPROMPT

- 운영 체제: 모두
- 디폴트=없음(정의되지 않은 경우, 『db2 => 』가 디폴트 대화식 프롬프트로 사용됨), 값: 0개 이상의 %i, %d, %ia, %da 또는 %n 토큰을 포함하는 100자 미만의 텍스트 문자열. 디폴트 CLP 대화식 프롬프트(db2 =>)를 명시적으로 변경하려는 경우를 제외하면 사용자는 이 변수를 설정할 필요가 없습니다.
- 이 레지스트리 변수를 사용하여 사용자는 명령행 처리기(CLP) 대화식 모드에서 사용할 프롬프트를 정의할 수 있습니다. 0개 이상의 선택적 토큰 %i, %d, %ia, %da 또는 %n을(를) 포함하는 100자 미만의 텍스트 문자열로 변수를 설정할 수 있습니다. CLP 대화식 모드에서 실행 중인 경우, 사용되는 프롬프트는 **DB2_CLPPROMPT** 레지스트리 변수에 지정된 텍스트 문자열을 가져와서 %i, %d, %ia, %da 또는 %n 토큰의 모든 어커런스를 각각 현재 첨부된 인스턴스의 로컬 별명, 현재 데이터베이스 연결의 로컬 별명, 현재 첨부된 인스턴스의 권한 부여 ID, 현재 데이터베이스 연결의 권한 부여 ID 및 라인 바꾸기(즉, 캐리지 리턴)로 모두 바꿔 구성됩니다.

  주:

  1. **DB2_CLPPROMPT** 레지스트리 변수가 CLP 대화식 모드에서 변경되는 경우, **DB2_CLPPROMPT**의 새 값은 CLP 대화식 모드가 닫히고 다시 열릴 때까지 적용되지 않습니다.
  2. 인스턴스 첨부가 존재하지 않는 경우, %ia은(는) 빈 문자열로 바뀌고 %i은(는) **DB2INSTANCE** 레지스트리 변수의 값으로 바뀝니다. Windows 플랫폼에서만 **DB2INSTANCE**가 설정되지 않은 경우 %i은(는) **DB2INSTDEF** 레지스트리 변수의 값으로 바뀝니다. 이러한 변수가 모두 설정되지 않은 경우, %i은(는) 빈 문자열로 바뀝니다.
  3. 데이터베이스 연결이 존재하지 않는 경우, %da은(는) 빈 문자열로 바뀌고 %d은(는) **DB2DBDFT** 레지스트리 변수의 값으로 바뀝니다. **DB2DBDFT** 변수가 설정되지 않은 경우, %d은(는) 빈 문자열로 바뀝니다.
  4. 대화식 입력 프롬프트는 항상 위 케이스의 권한 부여 ID, 데이터베이스 이름 및 인스턴스 이름의 값을 표시합니다.

DB2IQTIME

- 운영 체제: 모두
- 디폴트=5초, 최소값: 1초
- 이 변수는 명령행 처리기 벡엔드 프로세스가 명령을 전달하기 위해 프론트 엔드 프로세스의 입력 큐에서 대기하는 시간 크기를 지정합니다.

DB2RQTIME

- 운영 체제: 모두
- 디폴트=5초, 최소값: 1초
- 이 변수는 명령행 처리기 백엔드 프로세스가 프론트 엔드 프로세스의 요청을 대기하는 시간 크기를 지정합니다.

파티션된 데이터베이스 환경 변수

DB2CHGPWD_EEE

- 운영 체제: AIX, Linux 및 Windows의 DB2 ESE
- 디폴트=NULL, 값: YES 또는 NO
- 이 변수는 다른 사용자가 AIX 또는 Windows ESE 시스템에서 암호를 변경하도록 허용하는지 여부를 지정합니다. 모든 데이터베이스 파티션 또는 노드의 암호가 Windows의 Windows 도메인 제어기나 AIX의 LDAP를 사용하여 중앙집중식으로 유지되는지 확인하십시오. 중앙집중식으로 유지되지 않는 경우, 모든 데이터베이스 파티션 또는 노드 전체에서 암호가 불일치할 수 있습니다. 그 결과, 변경을 수행하기 위해 사용자가 연결하는 데이터베이스 파티션에서만 암호가 변경될 수 있습니다.

DB2_FCM_SETTINGS

- 운영 체제: Linux
- 디폴트=YES, 값: FCM_MAXIMIZE_SET_SIZE:[YES|TRUE|NO|FALSE]. FCM_MAXIMIZE_SET_SIZE의 디폴트값은 YES입니다.
- FCM_MAXIMIZE_SET_SIZE 토큰을 사용하여 **DB2_FCM_SETTINGS** 레지스트리 변수를 설정하여 FCM(Fast Communication Manager) 버퍼에 디폴트 2GB의 스페이스를 사전 할당할 수 있습니다. 이 기능을 사용하려면 토큰의 값이 YES 또는 TRUE여야 합니다.

DB2_FORCE_OFFLINE_ADD_PARTITION

- 운영 체제: 모두
- 디폴트=FALSE, 값: FALSE 또는 TRUE
- 이 변수를 사용하여 데이터베이스 파티션 서버 추가 조작이 오프라인으로 수행되도록 지정할 수 있습니다. 디폴트 설정인 FALSE는 데이터베이스를 오프라인으로 설정하지 않고 DB2 데이터베이스 파티션 서버를 추가할 수 있

음을 표시합니다. 그러나 조작을 오프라인으로 수행할 경우 또는 일부 제한사항으로 인해 데이터베이스가 온라인 상태일 때 데이터베이스 파티션 서버를 추가할 수 없는 경우, **DB2_FORCE_OFFLINE_ADD_PARTITION**을 TRUE로 설정하십시오. 이 변수가 TRUE로 설정된 경우, 버전 9.5 및 이전 버전의 동작에 따라 새 DB2 데이터베이스 파티션 서버가 추가됩니다. 즉, 인스턴스가 종료된 후 재시작될 때까지 새 데이터베이스 파티션 서버가 인스턴스에 표시되지 않습니다.

DB2_NUM_FAILOVER_NODES

- 운영 체제: 모두
- 디폴트=2, 값: 0 - 필수 데이터베이스 파티션 수
- 장애 복구 시 머신에서 시작되어야 하는 추가 데이터베이스 파티션 수를 지정하려면 **DB2_NUM_FAILOVER_NODES**를 설정하십시오.

  DB2 데이터베이스 고가용성 솔루션에서 데이터베이스 서버가 실패하는 경우, 실패한 머신의 데이터베이스 파티션을 다른 머신에서 재시작할 수 있습니다. FCM(Fast Communication Manager)은 **DB2_NUM_FAILOVER_NODES**를 사용하여 이 장애 복구를 쉽게 하기 위해 각 머신에서 예약할 메모리 크기를 계산합니다.

  예를 들어, 다음 구성을 고려하십시오.

  – 머신 A에는 두 개의 데이터베이스 파티션이 있습니다(1 및 2).

  – 머신 B에는 두 개의 데이터베이스 파티션이 있습니다(3 및 4).

  – A와 B 모두에서 **DB2_NUM_FAILOVER_NODES**는 2로 설정됩니다.

  START DBM에서 FCM은 한 머신이 실패하는 경우 실패한 머신의 두 데이터베이스 파티션을 다른 머신에서 시작할 수 있도록 최대 네 개의 데이터베이스 파티션을 관리하기 위해 A와 B 모두에서 충분한 메모리를 예약합니다. 머신 A가 실패하는 경우, 데이터베이스 파티션 1 및 2를 머신 B에서 재시작할 수 있습니다. 머신 B가 실패하는 경우, 데이터베이스 파티션 3 및 4를 머신 A에서 재시작할 수 있습니다.

DB2_PARTITIONEDLOAD_DEFAULT

- 운영 체제: 지원되는 모든 ESE 플랫폼
- 디폴트=YES, 값: YES 또는 NO
- **DB2_PARTITIONEDLOAD_DEFAULT** 레지스트리 변수를 사용하면 ESE 특정 로드 옵션이 지정되지 않은 경우 사용자가 ESE 환경에서 로드 유틸리티의 디폴트 동작을 변경할 수 있습니다. 디폴트값은 YES이며 이는 ESE 환경에서 ESE 특정 로드 옵션을 지정하지 않은 경우 목표 테이블이 정의된 모

든 데이터베이스 파티션에서 로딩이 시도되도록 지정합니다. 값이 NO이면 로드 유틸리티가 현재 연결되어 있는 데이터베이스 파티션에서만 로딩이 시도됩니다.

주: 이 변수는 사용되지 않으며 추후 릴리스에서는 제거됩니다. LOAD 명령에는 동일한 동작을 수행하는 데 사용할 수 있는 여러 가지 옵션이 있습니다. LOAD 명령에 다음을 지정하여 이 변수의 NO 설정과 동일한 결과를 얻을 수 있습니다. `PARTITIONED DB CONFIG MODE LOAD_ONLY OUTPUT_DBPARTNUMS x`, 여기서 x는 데이터를 로드하려는 파티션의 파티션 번호입니다.

DB2PORTRANGE

- 운영 체제: Windows
- 값: nnnn:nnnn
- 이 값은 다른 머신에서 작성된 모든 추가 데이터베이스 파티션의 포트 범위도 동일하도록 FCM에서 사용되는 TCP/IP 포트 범위로 설정됩니다.

쿼리 컴파일러 변수

DB2_ANTIJOIN

- 운영 체제: 모두
- ESE 환경에서는 디폴트=NO, 비ESE 환경에서는 디폴트=YES, 값: YES, NO 또는 EXTEND
- DB2 Enterprise Server Edition의 경우: YES가 지정되면 옵티마이저는『NOT EXISTS』서브쿼리를 DB2에서 더 효과적으로 처리할 수 있는 부정형 조인으로 변환할 기회를 검색합니다 . 비ESE 환경의 경우: NO가 지정되면 옵티마이저는『NOT EXISTS』서브쿼리를 부정형 조인으로 변환할 기회를 제한합니다.

  ESE 및 비ESE 환경 모두에서 EXTEND가 지정되면 옵티마이저는 ″NOT IN″과 ″NOT EXISTS″ 서브쿼리를 모두 부정형 조인으로 변환할 기회를 검색합니다.

DB2_DEFERRED_PREPARE_SEMANTICS

- 운영 체제: 모두
- 디폴트=YES, 값: YES 또는 NO
- YES로 설정되면 이 변수는 PREPARE문에 사용된 유형이 지정되지 않은 모든 매개변수 표시문자가 후속 OPEN 또는 EXECUTE문과 연관된 입력 디스크립터를 기반으로 데이터 유형 및 길이 속성을 이끌어내도록 지연된 준비 시맨틱을 사용합니다. 이를 통해 유형이 지정되지 않은 매개변수 표시문자를 이전보다 더 많은 위치에 사용할 수 있습니다.

- **DB2_DEFERRED_PREPARE_SEMANTICS**는 db2start 앞에 설정해야 합니다.

DB2_INLIST_TO_NLJN

- 운영 체제: 모두
- 디폴트=NO, 값: YES 또는 NO
- 일부 상황에서 SQL 및 XQuery 컴파일러는 조인에 대한 IN 목록 술어를 재작성할 수 있습니다. 예를 들어,

```
SELECT *
 FROM EMPLOYEE
 WHERE DEPTNO IN ('D11', 'D21', 'E21')
```

  쿼리를 다음과 같이 재작성할 수 있습니다.

```
SELECT *
 FROM EMPLOYEE, (VALUES 'D11', 'D21', 'E21) AS V(DNO)
 WHERE DEPTNO = V.DNO
```

  DEPTNO에 대한 인덱스가 있는 경우 이 개정판은 향상된 성능을 제공합니다. 값 목록을 먼저 액세스한 다음 Join 술어를 적용할 인덱스를 사용하여 중첩된 루프 조인과 함께 EMPLOYEE에 조인합니다.

  옵티마이저에 쿼리의 재작성 버전에 가장 적합한 조인 메소드를 판별하는 데 필요한 정확한 정보가 없는 경우가 있습니다. 이러한 상황은 옵티마이저가 선택성을 판별하는 데 카탈로그 통계를 사용하지 못하게 하는 매개변수 표시 문자 또는 호스트 변수가 IN 목록에 포함된 경우에 발생합니다. 이 레지스트리 변수는 옵티마이저가 IN 목록을 조인의 내부 테이블로 제공하는 테이블을 사용하여 값 목록을 조인하는 중첩 루프 조인을 기본으로 설정하는 원인이 됩니다.

  주: DB2 쿼리 컴파일러 변수 **DB2_MINIMIZE_LISTPREFETCH** 및 **DB2_INLIST_TO_NLJN** 중 하나 또는 두 변수 모두 YES로 설정되면 REOPT(ONCE)가 지정된 경우에도 해당 변수는 활성 상태를 유지합니다.

DB2_LIKE_VARCHAR

- 운영 체제: 모두
- 디폴트=Y,Y,
- 하위 요소 통계의 사용을 제어합니다. 데이터에 공백으로 구분된 일련의 하위 필드 또는 하위 요소 형식의 구조가 있는 경우 컬럼의 데이터 컨텐츠에 대한 통계가 있습니다. 하위 요소 통계의 콜렉션은 선택적이며 RUNSTATS 명령 또는 API의 옵션으로 제어됩니다.

  이 레지스트리 변수는 옵티마이저가 다음 형식의 술어를 처리하는 방법에 영향을 미칩니다.

```
COLUMN LIKE '%xxxxxx%'
```

여기서 xxxxxx는 문자로 구성된 임의의 문자열입니다.

이 레지스트리 변수가 사용되는 방법을 표시하는 구문은 다음과 같습니다.

```
db2set DB2_LIKE_VARCHAR=[Y|N|S|num1] [,Y|N|S|num2]
```

where

– 쉼표 앞에 있는 용어 또는 술어 오른쪽에 있는 용어는 두 번째 용어가 N으로 지정되거나 컬럼에 양의 하위 요소 통계가 없는 경우에만 다음을 의미합니다.
  - S – 옵티마이저는 함께 병합되어 컬럼을 형성하는 일련의 요소에서 % 문자로 묶인 문자열의 길이를 기반으로 각 요소의 길이를 추정합니다.
  - Y – 디폴트입니다. 알고리즘 매개변수에 디폴트값 1.9를 사용하십시오. 알고리즘 매개변수와 함께 변수 길이 하위 요소 알고리즘을 사용하십시오.
  - N – 고정 길이 하위 요소 알고리즘을 사용하십시오.
  - num1 – 변수 길이 하위 요소 알고리즘에 num1 값을 알고리즘 매개변수로 사용하십시오.

– 쉼표 뒤에 오는 용어는 양의 하위 요소 통계가 있는 컬럼에 대해서만 다음을 의미합니다.
  - N – 하위 요소 통계를 사용하지 않습니다. 첫 번째 용어가 적용됩니다.
  - Y – 디폴트입니다. 양의 하위 요소 통계가 있는 컬럼의 경우에 알고리즘 매개변수에 1.9 디폴트값과 함께 하위 요소 통계를 사용하는 변수 길이 하위 요소 알고리즘을 사용하십시오.
  - num2 – 양의 하위 요소 통계가 있는 컬럼의 경우에 알고리즘 매개변수로 num2 값과 함께 하위 요소 통계를 사용하는 변수 길이 하위 요소 알고리즘을 사용하십시오.

DB2_MINIMIZE_LISTPREFETCH

- 운영 체제: 모두
- 디폴트=NO, 값: YES 또는 NO
- 리스트 프리페치는 인덱스에서 규정 RID 검색, 페이지 번호순 정렬 및 데이터 페이지 프리페치를 포함하는 특수 테이블 액세스 메소드입니다. 옵티마이저에 리스트 프리페치가 적합한 액세스 메소드인지를 판별하는 데 필요한 정확한 정보가 없는 경우가 있습니다. 이러한 상황은 옵티마이저가 선택성을 판별하는 데 카탈로그 통계를 사용하지 못하게 하는 매개변수 표시문자 또는 호스트 변수가 술어 선택성에 포함된 경우에 발생합니다.

이 레지스트리 변수는 옵티마이저가 이러한 상황에서 리스트 프리페치를 고려하는 것을 방지합니다.

주: DB2 쿼리 컴파일러 변수 **DB2_MINIMIZE_LISTPREFETCH** 및 **DB2_INLIST_TO_NLJN** 중 하나 또는 두 변수 모두 YES로 설정되면 REOPT(ONCE)가 지정된 경우에도 해당 변수를 활성 상태를 유지합니다.

DB2_NEW_CORR_SQ_FF

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- ON으로 설정된 경우 특정 서브쿼리 술어에 대해 쿼리 옵티마이저에서 계산한 선택성 값에 영향을 줍니다. 이 변수는 서브쿼리의 SELECT 목록에서 MIN 또는 MAX 집계 함수를 사용하는 등식 서브쿼리 술어의 선택성 값의 정확성을 높이기 위해 사용할 수 있습니다. 예를 들어, 다음과 같습니다.

```
SELECT * FROM T WHERE
T.COL = (SELECT MIN(T.COL)
FROM T WHERE ...)
```

DB2_OPT_MAX_TEMP_SIZE

- 운영 체제: 모두
- 디폴트=NULL, 값: 모든 임시 테이블 스페이스에서 쿼리에 사용할 수 있는 *스페이스 크기*(MB)
- 임시 테이블 스페이스에서 쿼리가 사용할 수 있는 *스페이스 크기*를 제한합니다. **DB2_OPT_MAX_TEMP_SIZE** 설정으로 옵티마이저는 그렇지 않은 경우에 선택되는 플랜보다 비용은 많이 들지만 임시 테이블 스페이스의 스페이스를 적게 사용하는 플랜을 선택하게 됩니다.
  DB2_OPT_MAX_TEMP_SIZE를 설정하는 경우, 설정으로 인해 선택되는 플랜의 유효성에 대해 임시 테이블 스페이스의 사용을 제한할 필요성의 균형을 맞추십시오.

  DB2_WORKLOAD=SAP가 설정된 경우, **DB2_OPT_MAX_TEMP_SIZE**는 자동으로 10 240(10GB)으로 설정됩니다.

  DB2_OPT_MAX_TEMP_SIZE의 값 세트를 초과하여 임시 테이블 스페이스를 사용하는 쿼리를 실행하는 경우, 쿼리는 실패하지 않지만 모든 자원이 사용 가능하지는 않으므로 성능이 준최적 상태일 수 있다는 경고가 수신됩니다.

  DB2_OPT_MAX_TEMP_SIZE로 설정된 한계의 영향을 받는 옵티마이저에서 고려하는 조작은 다음과 같습니다.
  - ORDER BY, DISTINCT, GROUP BY, 병합 스캔 조인 및 중첩된 루프와 같은 조작에 대한 명시적 정렬

– 명시적 임시 테이블
– 해시 조인 및 중복 병합 조인에 대한 내재된 임시 테이블

DB2_REDUCED_OPTIMIZATION

- 운영 체제: 모두
- 디폴트=NO, 값: NO, YES, 임의의 정수, DISABLE, NO_SORT_NLJOIN, 또는 NO_SORT_MGJOIN
- 이 레지스트리 변수를 사용하여 지정된 최적화 레벨에서 최적화 기능의 엄격한 사용 또는 감소된 최적화 기능을 요청할 수 있습니다. 사용되는 최적화 기술의 수를 줄일 경우, 최적화 동안의 시간 및 자원 사용도 줄어듭니다.

  주: 최적화 시간 및 자원 사용이 감소될 수도 있지만, 덜 최적화된 데이터 액세스 플랜을 생성할 위험이 커집니다. IBM 또는 협력업체에서 권고하는 경우에만 이 레지스트리 변수를 사용하십시오.

  – NO로 설정된 경우

    옵티마이저는 최적화 기술을 변경하지 않습니다.

  – YES로 설정된 경우

    최적화 레벨이 5(디폴트) 이하인 경우, 옵티마이저는 상당한 준비 시간 및 자원을 사용하지만 일반적으로 더 나은 액세스 플랜을 생성하지 않는 일부 최적화 기술을 사용하지 않습니다.

    최적화 레벨이 정확히 5인 경우, 옵티마이저는 최적화 시간 및 자원 사용은 줄이지만 덜 최적화된 액세스 플랜을 생성할 위험을 증가시키기도 하는 일부 추가 기술을 줄이거나 사용하지 않습니다. 최적화 레벨이 5보다 낮은 경우, 이러한 기술 중 일부는 어떠한 경우에도 효력이 없을 수 있습니다. 그러나 그럴 경우, 여전히 적용됩니다.

  – 임의의 정수로 설정된 경우

    효과는 YES 설정과 동일하며, 레벨 5로 최적화된 동적으로 준비된 쿼리에 대한 다음 추가 동작이 있습니다. 쿼리 블록의 총 조인 수가 설정을 초과할 경우, 옵티마이저는 레벨 5 최적화 레벨에 대해 위에 설명된 대로 추가 최적화 기술을 사용하지 않는 대신 그리디(greedy) 조인 열거로 전환합니다. 그 결과 쿼리는 최적화 레벨 2와 유사한 레벨에서 최적화됩니다.

  – DISABLE로 설정된 경우

    이 **DB2_REDUCED_OPTIMIZATION** 변수에 의해 제한되지 않을 때 옵티마이저의 동작은 때때로 최적화 레벨 5에서 동적 쿼리의 최적화를 동

적으로 줄일 수 있습니다. 이 설정은 이러한 동작을 사용 불가능하게 하며 옵티마이저는 전체 레벨 5 최적화를 수행해야 합니다.

– NO_SORT_NLJOIN으로 설정된 경우

옵티마이저는 중첩된 루프 조인(NLJN)에 대한 정렬을 강제 실행하는 쿼리 플랜을 생성하지 않습니다. 이러한 정렬 유형은 성능 향상에 유용할 수 있습니다. 따라서 NO_SORT_NLJOIN 옵션을 사용할 때는 성능에 심각하게 영향을 미칠 수 있으므로 주의하십시오.

– NO_SORT_MGJOIN으로 설정된 경우

옵티마이저는 병합 스캔 조인(MSJN)에 대한 정렬을 강제 실행하는 쿼리 플랜을 생성하지 않습니다. 이러한 정렬 유형은 성능 향상에 유용할 수 있습니다. 따라서 NO_SORT_MGJOIN 옵션을 사용할 때는 성능에 심각하게 영향을 미칠 수 있으므로 주의하십시오.

최적화 레벨 5에서의 동적 최적화 감소는 **DB2_REDUCED_OPTIMIZATION**이 YES로 설정되었을 때 정확히 5인 최적화 레벨에 대해 설명된 동작 및 정수 설정에 대해 설명된 동작에 우선합니다.

DB2_SELECTIVITY

- 운영 체제: 모두
- 디폴트=NO, 값: YES 또는 NO
- 이 레지스트리 변수는 SELECTIVITY절을 SQL문의 검색 조건에 사용할 수 있는 위치를 제어합니다.

  이 레지스트리 변수를 YES로 설정하면 SELECTIVITY절을 다음 술어에 지정할 수 있습니다.
  – 하나 이상의 표현식이 호스트 변수를 포함하는 BASIC 술어
  – MATCH 표현식, 술어 표현식 또는 Esc 표현식이 호스트 변수를 포함하는 LIKE 술어

DB2_SQLROUTINE_PREPOPTS

- 운영 체제: 모두
- 디폴트=빈 문자열, 값:
  – APREUSE {YES | NO}
  – BLOCKING {UNAMBIG | ALL | NO}
  – CONCURRENTACCESSRESOLUTION { USE CURRENTLY COMMITTED | WAIT FOR OUTCOME }
  – DATETIME {DEF | USA | EUR | ISO | JIS | LOC}

- DEGREE {1 | *degree-of-parallelism* | ANY}
- DYNAMICRULES {BIND | INVOKEBIND | DEFINEBIND | RUN | INVOKERUN | DEFINERUN}
- EXPLAIN {NO | YES | ALL}
- EXPLSNAP {NO | YES | ALL}
- FEDERATED {NO | YES}
- INSERT {DEF | BUF}
- ISOLATION {CS | RR | UR | RS | NC}
- OPTPROFILE {profile_name | schema_name.profile_name}
- QUERYOPT *optimization-level*
- REOPT {NONE | ONCE | ALWAYS}
- STATICREADONLY {YES|NO}
- VALIDATE {RUN | BIND}

• **DB2_SQLROUTINE_PREPOPTS** 레지스트리 변수를 사용하여 SQL 및 XQuery 프로시저에 대한 프리컴파일 및 바인드 옵션을 사용자 정의할 수 있습니다. 이 변수를 설정할 때 다음과 같이 각 옵션을 공백으로 구분하십시오.

```
db2set DB2_SQLROUTINE_PREPOPTS="BLOCKING ALL VALIDATE RUN"
```

각 옵션 및 해당 설정의 전체 설명은 "BIND 명령"을 참조하십시오.

인스턴스를 재시작하지 않고 개별 프로시저 선택에 대해 **DB2_SQLROUTINE_PREPOPTS**와 동일한 결과를 얻으려면 SET_ROUTINE_OPTS 프로시저를 사용하십시오.

성능 변수

DB2_ALLOCATION_SIZE

• 운영 체제: 모두

• 디폴트=128KB, 범위: 64KB - 256MB

• 버퍼 풀에 대한 메모리 할당의 *크기*를 지정합니다.

이 레지스트리 변수에 높은 값을 설정하면 버퍼 풀에 대해 원하는 메모리 크기에 접근하는 데 할당이 적게 필요하게 되는 이점이 있을 수 있습니다.

이 레지스트리 변수에 높은 값을 설정하면 버퍼 풀이 할당 *크기*의 배수가 아닌 수로 변경되는 경우 메모리가 낭비되는 비용이 들 수 있습니다. 예를 들어, **DB2_ALLOCATION_SIZE**의 값이 8MB이고 버퍼 풀이 4MB로 감소되는 경우, 전체 8MB의 세그먼트를 해제할 수 없으므로 이 4MB는 낭비됩니다.

주: **DB2_ALLOCATION_SIZE**는 사용되지 않으며 추후 릴리스에서는 제거됩니다.

DB2_APM_PERFORMANCE

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- 쿼리 캐시(패키지 캐시)의 동작에 영향을 주는 액세스 플랜 관리 프로그램(APM)의 성능 관련 변경사항을 사용하려면 이 변수를 ON으로 설정하십시오. 이러한 설정은 일반적으로 프로덕션 시스템에 권장됩니다. 이로 인해 패키지 캐시 부족 오류 또는 증가된 메모리 사용 또는 둘 다의 가능성과 같은 제한사항이 발생합니다.

  또한 **DB2_APM_PERFORMANCE**를 ON으로 설정하면 NO PACKAGE LOCK 모드를 사용할 수 있습니다. 이 모드를 사용하여 캐시된 패키지 항목이 제거되는 것을 방지하는 내부 시스템 잠금인 패키지 잠금을 사용하지 않고 전역 쿼리 캐시를 작동할 수 있습니다. NO PACKAGE LOCK 모드는 결과적으로 약간의 성능 향상을 가져오지만 일부 데이터베이스 조작이 허용되지 않습니다. 이러한 금지된 조작에는 패키지를 무효화하는 조작, 패키지를 작동 불가능하게 하는 조작 및 PRECOMPILE, BIND 및 REBIND가 포함됩니다.

DB2ASSUMEUPDATE

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- 사용 가능한 경우, 이 변수를 사용하여 DB2 데이터베이스 시스템은 UPDATE문에 제공된 모든 고정 길이 컬럼이 변경된다고 가정할 수 있습니다. 이로 인해 DB2 데이터베이스 시스템은 컬럼이 실제로 변경되고 있는지 판별하기 위해 기존 컬럼 값과 새 값을 비교할 필요가 없어집니다. 이 레지스트리 변수를 사용하면 갱신할 컬럼이 제공되었지만(예를 들어, SET절에) 실제로는 수정되고 있지 않을 때 추가 로깅 및 인덱스 유지보수가 발생할 수 있습니다.

  DB2ASSUMEUPDATE 레지스트리 변수의 활성화는 db2start 명령에서 유효합니다.

DB2_ASYNC_IO_MAXFILOP

- 운영 체제: 모두
- 디폴트t=*maxfilop* 구성 매개변수의 값, 값: *maxfilop*의 값 - *max_int*의 값
- **DB2_ASYNC_IO_MAXFILOP**는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 이 변수는 스레드된 데이터베이스 관리 프로그램에 의해 유지보수

되는 공유 파일 핸들 테이블 때문에 사용되지 않습니다. 자세한 정보는 ″공유 파일 핸들 테이블″ 주제를 참조하십시오.

DB2_ASYNC_IO_MAXFILOP는 버전 9.5에서 계속 설정될 수 있지만 영향을 미치지 않습니다. 각 데이터베이스에 대해 열 수 있는 파일 핸들 수를 제한하려면 *maxfilop* 구성 매개변수를 참조하십시오.

DB2_AVOID_PREFETCH

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- 응급 복구 중에 프리페치가 사용되는지 여부를 지정합니다. **DB2_AVOID_PREFETCH**=ON이면 프리페치가 사용되지 않습니다.

DB2BPVARS

- 운영 체제: 각 매개변수에 지정된 대로 사용됨
- 디폴트=경로
- 버퍼 풀을 조정하는 데 두 개의 매개변수 세트를 사용할 수 있습니다. Windows에서만 사용 가능한 한 매개변수 세트는 버퍼 풀이 특정 컨테이너 유형에 대해 분산 읽기를 사용하도록 지정합니다. 모든 플랫폼에서 사용 가능한 다른 매개변수 세트는 프리페칭 동작에 영향을 미칩니다.

  매개변수는 `parameter=value` 형식으로 각 라인에 하나씩 ASCII 파일에 지정됩니다. 예를 들어, `bpvars.vars` 파일에는 다음 라인이 포함됩니다.

  ```
  NO_NT_SCATTER = 1
  NUMPREFETCHQUEUES = 2
  ```

  `bpvars.vars`가 `F:₩vars₩`에 저장되어 있다고 가정하고 이러한 변수를 설정하려면 다음 명령을 실행하십시오.

  ```
  db2set DB2BPVARS=F:₩vars₩bpvars.vars
  ```

  분산 읽기 매개변수

  분산 읽기 매개변수는 컨테이너의 각 유형에 대해 대량의 순차 프리페칭이 있는 시스템 및 **DB2NTNOCACHE**를 이미 ON으로 설정한 시스템에 권장됩니다. Windows 플랫폼에서만 사용 가능한 이러한 매개변수는 NT_SCATTER_DMSFILE, NT_SCATTER_DMSDEVICE 및 NT_SCATTER_SMS입니다. 컨테이너에 대해 분산 읽기를 명시적으로 승인 불가하려면 NO_NT_SCATTER 매개변수를 지정하십시오. 특정 매개변수가 표시된 유형의 모든 컨테이너에 대해 분산 읽기를 켜는 데 사용됩니다. 이러한 매개변수 각각에 대해 디폴트는 0(또는 OFF)이고 가능한 값은 0(또는 OFF) 및 1(또는 ON)입니다.

주: DB2NTNOCACHE가 ON으로 설정되어 Windows 파일 캐싱을 끈 경우에만 분산 읽기를 켤 수 있습니다. **DB2NTNOCACHE**가 OFF로 설정되거나 설정되지 않은 경우, 컨테이너에 대해 분산 읽기를 켜려고 시도하면 경고 메시지가 관리 통지 로그에 기록되며 분산 읽기는 사용 안함을 유지합니다.

프리페치 조정 매개변수

프리페치 조정 매개변수는 NUMPREFETCHQUEUES 및 PREFETCHQUEUESIZE입니다. 해당 매개변수는 모든 플랫폼에서 사용 가능하며 버퍼 풀 데이터 프리페칭을 향상시키는 데 사용할 수 있습니다. 예를 들어, 원하는 PREFETCHSIZE가 PREFETCHSIZE/EXTENTSIZE 프리페치 요청으로 나뉘는 순차 프리페칭을 고려하십시오. 이 경우, 요청은 입출력 서버가 디스패치되어 비동기 입출력을 수행하는 프리페치 큐에 배치됩니다. 디폴트로 DB2 데이터베이스 관리 프로그램은 각 데이터베이스 파티션에 대해 하나의 *큐 크기* max(200,2*NUM_IOSERVERS)를 유지합니다. 일부 환경에서는 더 많은 큐 또는 다른 *크기*의 큐 또는 둘 다를 사용하면 성능이 향상됩니다. 프리페치 큐 수는 입출력 서버 수의 2분의 1 이하여야 합니다. 이러한 매개변수를 설정할 때 PREFETCHSIZE, EXTENTSIZE, NUM_IOSERVERS 및 버퍼 풀 *크기*와 같은 다른 매개변수 및 현재 사용자 수와 같은 워크로드 특성를 고려하십시오.

사용자 환경에 대해 디폴트값이 너무 작다고 생각하면 먼저 값을 약간만 증가시키십시오. 예를 들어, NUMPREFETCHQUEUES=4 및 PREFETCHQUEUESIZE=200을 설정합니다. 변경 효과를 모니터하고 평가할 수 있도록 제어된 방식으로 이러한 매개변수를 변경하십시오.

NUMPREFETCHQUEUES의 경우, 디폴트는 1이고 값의 범위는 1 - NUM_IOSERVERS입니다. NUMPREFETCHQUEUES를 1 미만으로 설정하면 1로 조정됩니다. NUM_IOSERVERS보다 크게 설정하면 NUM_IOSERVERS로 조정됩니다.

PREFETCHQUEUESIZE의 경우, 디폴트값은 max(200,2*NUM_IOSERVERS)입니다. 값의 범위는 1 - 32,767입니다. PREFETCHQUEUESIZE를 1 미만으로 설정하면 디폴트로 조정됩니다. 32,767보다 크게 설정하면 32,767로 조정됩니다.

주: DB2BPVARS는 사용되지 않으며 추후 릴리스에서는 제거됩니다.

DB2CHKPTR

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF

- 입력에 대한 포인터 검사가 필요한지 여부를 지정합니다.

DB2CHKSQLDA

- 운영 체제: 모두
- 디폴트=ON, 값: ON 또는 OFF
- 입력에 대한 SQLDA 검사가 필요한지 여부를 지정합니다.

DB2_EVALUNCOMMITTED

- 운영 체제: 모두
- 디폴트=OFF, 값: ON, OFF
- 이 변수를 사용할 수 있는 경우, 이를 사용하면 가능한 경우에 술어 평가를 만족시키기 위해 데이터가 알려질 때까지 스캔이 행 잠금을 지연하거나 방지할 수 있습니다. 이 변수가 사용 가능한 경우, 술어 평가는 언커미트된 데이터에서 발생합니다.

  DB2_EVALUNCOMMITTED는 현재 커미트된 시맨틱이 잠금 경합을 방지하는 데 도움이 되지 않는 경우에만 적용 가능합니다. 이 변수가 설정되고 현재 커미트되어 스캔에 적용 가능한 경우, 삭제된 행을 건너뛰지 않고 언커미트된 데이터에서 술어 평가가 발생하지 않습니다. 대신에 현재 커미트된 행 및 데이터 버전이 처리됩니다.

  마찬가지로 **DB2_EVALUNCOMMITTED**는 커서 안정성 또는 읽기 안정성 분리 레벨을 사용하는 명령문에만 적용됩니다. 또한 레지스트리 변수 **DB2_SKIPDELETED**도 설정된 경우가 아니면 인덱스 스캔에 대해서는 삭제된 키를 건너뛰지 않는 반면에 테이블 액세스 스캔 시에는 삭제된 행을 무조건 건너뜁니다.

  DB2_EVALUNCOMMITTED 레지스트리 변수의 활성화는 db2start 명령에서 유효합니다. 지연된 잠금이 적용 가능한지 여부는 명령문 컴파일 또는 바인드 시간에 결정됩니다.

DB2_EXTENDED_IO_FEATURES

- 운영 체제: AIX
- 디폴트=OFF, 값: ON, OFF
- 입출력 성능을 향상시키는 기능을 사용하려면 이 변수를 ON으로 설정하십시오. 이 향상된 기능에는 메모리 캐시 적중률 향상 및 높은 우선순위 입출력 시 대기 시간 줄이기가 포함됩니다. 이러한 기능은 특정 조합의 소프트웨어 및 하드웨어 구성에서만 사용 가능합니다. 다른 구성에 대해 이 변수를 ON으로 설정하면 DB2 데이터베이스 관리 시스템 또는 운영 체제에서 무시됩니다. 최소 구성 요구사항은 다음과 같습니다.
  - 데이터베이스 버전: DB2 V9.1

– 데이터베이스 컨테이너에 RAW 디바이스를 사용해야 함(파일 시스템의 컨테이너는 지원되지 않음)
– 운영 체제: AIX 5.3 TL4
– 스토리지 서브시스템: Shark DS8000은 향상된 모든 입출력 성능 기능을 지원합니다. 설치 및 전제조건 정보는 Shark DS8000 문서를 참조하십시오.

HIGH, MEDIUM 및 LOW의 디폴트 입출력 우선순위 설정값은 각각 3, 8 및 12입니다. **DB2_IO_PRIORITY_SETTING** 레지스트리 변수를 사용하여 이러한 설정값을 변경할 수 있습니다.

DB2_EXTENDED_OPTIMIZATION

- 운영 체제: 모두
- 디폴트=OFF, 값: ON, OFF 또는 ENHANCED_MULTIPLE_DISTINCT
- 이 변수는 쿼리 옵티마이저가 최적화 확장을 사용하여 쿼리 성능을 향상시키는지 여부를 지정합니다. ON 및 ENHANCED_MULTIPLE_DISTINCT 값은 서로 다른 최적화 확장을 지정합니다. 두 값 모두 사용하려면 쉼표로 구분된 목록을 사용하십시오.

  ENHANCED_MULTIPLE_DISTINCT 값은 하나의 단일 선택 조작에 여러 개의 구별 집계 조작이 포함되어 있는 경우 및 데이터베이스 파티션 수에 대한 프로세서의 비율이 낮은 경우(예를 들어, 비율이 1 이하임)의 쿼리 성능을 향상시킵니다. 이 설정은 대칭 멀티프로세서(SMP)가 없는 DPF(Database Partitioning Feature) 환경에서 사용되어야 합니다.

  최적화 확장은 일부 환경에서는 쿼리 성능을 향상시키지 않습니다. 개별 쿼리 성능 향상을 판별하려면 테스트를 수행해야 합니다.

DB2_HASH_JOIN

- 운영 체제: 모두
- 디폴트=YES, 값: YES 또는 NO
- 액세스 플랜을 컴파일할 때 해시 조인을 가능한 조인 메소드로 지정합니다. 최상의 성능을 얻으려면 **DB2_HASH_JOIN**을 조정해야 합니다. 해시 루프 및 디스크에 대한 오버플로우를 방지할 수 있는 경우 해시 조인 성능은 최상의 상태가 됩니다. 해시 조인 성능을 조정하려면 *sheapthres* 구성 매개변수에 사용할 수 있는 최대 메모리 양을 추정한 후 *sortheap* 구성 매개변수를 조정하십시오. 가능하면 많은 해시 루프 및 디스크 오버플로우를 방지할 수 있을 때까지 값을 늘리지만 *sheapthres* 구성 매개변수에 지정된 한계에는 접근하지 마십시오.

주: DB2_HASH_JOIN은 버전 9.5에서는 사용되지 않으며 추후 릴리스에서는 제거됩니다.

DB2_IO_PRIORITY_SETTING

- 운영 체제: AIX
- 값: HIGH:#, MEDIUM:#, LOW:#, 여기서 #는 1 - 15 사이의 값입니다.
- 이 변수는 **DB2_EXTENDED_IO_FEATURES** 레지스트리 변수와 조합하여 사용됩니다. 이 레지스트리 변수는 DB2 데이터베이스 시스템에 대한 디폴트 HIGH, MEDIUM 및 LOW 입출력 우선순위 설정값(각각 3, 8 및 12)을 겹쳐쓰는 수단을 제공합니다. 이 레지스트리 변수는 인스턴스 시작 전에 설정해야 합니다. 수정하면 인스턴스를 재시작해야 합니다. 이 레지스트리 변수 설정만으로는 향상된 입출력 기능을 사용할 수 없습니다. 해당 기능을 사용하려면 **DB2_EXTENDED_IO_FEATURES**를 설정해야 합니다. 또한 **DB2_EXTENDED_IO_FEATURES**에 대한 모든 시스템 요구사항이 이 레지스트리 변수에도 적용됩니다.

DB2_KEEP_AS_AND_DMS_CONTAINERS_OPEN

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- 이 변수를 ON으로 설정하면 데이터베이스가 비활성화될 때까지 각 DMS 테이블 스페이스 컨테이너의 파일 핸들이 열려 있습니다. 컨테이너는 열기 위한 오버헤드가 제거되므로 쿼리 성능이 향상됩니다. 이 레지스트리 변수는 순수 DMS 환경에서만 사용해야 합니다. 그렇지 않으면 SMS 테이블 스페이스에 대한 쿼리 성능에 부정적인 영향이 미칩니다.

DB2_KEEPTABLELOCK

- 운영 체제: 모두
- 디폴트: OFF, 값: ON, TRANSACTION, OFF, CONNECTION
- 이 변수가 ON 또는 TRANSACTION으로 설정되면 언커미트된 읽기 또는 커서 안정성 분리 레벨이 닫힌 경우 이 변수를 사용하여 DB2 데이터베이스 시스템이 테이블 잠금을 유지할 수 있습니다. 보존되는 테이블 잠금은 읽기 안정성 및 반복 읽기 스캔에 대해 릴리스되는 것처럼 트랜잭션 끝에 릴리스됩니다.

  이 변수가 CONNECTION으로 설정되면 응용프로그램이 트랜잭션을 롤백하거나 연결이 재설정될 때까지 테이블 잠금이 응용프로그램에 대해 릴리스됩니다. 테이블 잠금은 커미트 전체에 걸쳐서 계속 보유되며 테이블 잠금을 삭제하려는 응용프로그램 요청은 데이터베이스에서 무시됩니다. 테이블 잠금은 응용프로그램에 할당된 채로 남아 있습니다. 따라서 응용프로그램이 테이블 잠금을 다시 요청할 때 잠금은 이미 사용 가능합니다.

이 최적화를 가공할 수 있는 응용프로그램 워크로드의 경우, 성능이 향상되어야 합니다. 그러나 동시에 실행 중인 다른 응용프로그램의 워크로드는 영향을 받습니다. 다른 응용프로그램은 결과적으로 낮은 동시성을 가져오는 지정된 테이블에 대한 액세스로부터 차단됩니다. DB2 SQL 카탈로그 테이블은 이 설정의 영향을 받지 않습니다. CONNECTION 설정은 또한 ON 또는 TRANSACTION 설정으로 설명한 동작도 포함합니다.

이 레지스트리 변수는 명령문 컴파일 또는 바인드 시간에 점검됩니다.

DB2_LARGE_PAGE_MEM

- 운영 체제: AIX, Linux, Windows Server 2003
- 디폴트=NULL, 값: 대형 페이지 메모리를 사용해야 하는 모든 적용 가능한 메모리 영역 또는 대형 페이지 메모리를 사용해야 하는 특정 메모리 영역의 쉼표로 구분된 목록을 표시하려면 *를 사용하십시오. 사용 가능한 영역은 운영 체제에 따라 다릅니다. AIX에서는 DB, DBMS, FCM 또는 PRIVATE 영역을 지정할 수 있습니다. Linux에서는 DB 영역을 지정할 수 있습니다. Windows Server 2003에서는 DB 영역을 지정할 수 있습니다. 초대형 페이지 메모리는 AIX에서만 사용 가능합니다.
- **DB2_LARGE_PAGE_MEM** 레지스트리 변수는 대형 페이지 또는 초대형 페이지 지원을 사용 가능하게 하는 데 사용됩니다.
  DB2_LARGE_PAGE_MEM=DB 설정은 데이터베이스 공유 메모리 영역에 대해 대형 페이지 메모리를 사용 가능하게 하며 **database_memory**가 AUTOMATIC으로 설정되어 있으면 STMM에 의한 이 공유 메모리 영역의 자동 성능 조정을 사용 불가능하게 합니다. AIX에서 DB2_LARGE_PAGE_MEM=DB:16GB 설정은 데이터베이스 공유 메모리 영역에 대해 초대형 페이지 메모리를 사용 가능하게 합니다.

  많은 양의 가상 메모리를 사용하는 메모리 액세스 집중형 응용프로그램은 대형 또는 초대형 페이지를 사용하여 성능 향상을 얻습니다. DB2 데이터베이스 시스템이 이러한 페이지를 사용할 수 있게 하려면 먼저 대형 또는 초대형 페이지를 사용하도록 운영 체제를 구성해야 합니다.

  AIX용 64비트 DB2에서 에이전트 전용 메모리의 대형 페이지를 사용 가능하게 하려면(**DB2_LARGE_PAGE_MEM**=PRIVATE 설정), 운영 체제에서 대형 페이지를 구성해야 하며 인스턴스 소유자는 CAP_BYPASS_RAC_VMM 및 CAP_PROPAGATE 기능을 소유해야 합니다.

AIX 5L에서는 이 변수를 FCM으로 설정할 수 있습니다. FCM 메모리는 고유 메모리 세트에 상주하므로 FCM 메모리에 대한 대형 페이지를 사용하려면 FCM 키워드를 **DB2_LARGE_PAGE_MEM** 레지스트리 변수의 값에 추가해야 합니다.

Linux의 경우, *libcap.so.1* 라이브러리의 사용 가능성에 대한 추가적인 요구사항이 있습니다. 이 옵션이 작동하려면 이 라이브러리가 설치되어 있어야 합니다. 이 옵션은 커져 있고 라이브러리가 시스템에 없는 경우, DB2 데이터베이스는 대형 커널 페이지를 사용 불가능하게 하고 계속 해당 페이지가 없을 때처럼 작동합니다.

Linux에서 대형 커널 페이지가 사용 가능한지 검증하려면 다음 명령을 발행하십시오.

```
cat /proc/meminfo
```

대형 커널 페이지가 사용 가능한 경우, 다음의 세 라인이 표시되어야 합니다(사용자 서버에 구성된 메모리 *크기*에 따라 다른 숫자가 함께 표시됨).

```
HugePages_Total:   200
HugePages_Free:    200
Hugepagesize:    16384 kB
```

이러한 라인이 표시되지 않거나 HugePages_Total이 0인 경우, 운영 체제 또는 커널을 구성해야 합니다.

Windows의 경우, 시스템에서 사용 가능한 대형 페이지 메모리 *크기*는 사용 가능한 전체 메모리보다 적습니다. 일정 시간 동안 시스템이 실행된 후에는 메모리가 분할될 수 있고 대형 페이지 메모리의 *크기*는 줄어듭니다.

DB2_LOGGER_NON_BUFFERED_IO

- 운영 체제: 모두
- 디폴트=AUTO, 값: AUTOMATIC, ON, 또는 OFF
- 이 변수를 사용하여 로그 파일 시스템에서 직접 입출력(DIO)을 사용할 수 있습니다. **DB2_LOGGER_NON_BUFFERED_IO**가 AUTOMATIC으로 설정되어 있으면 활성 로그 창(즉, 1차 로그 파일)이 DIO로 열리고 다른 모든 로그 프로그램 파일이 버퍼됩니다. ON으로 설정되어 있으면 모든 로그 파일 핸들이 DIO로 열립니다. OFF으로 설정되어 있으면 모든 로그 파일 핸들이 버퍼됩니다.

DB2MAXFSCRSEARCH

- 운영 체제: 모두
- 디폴트=5, 값: -1, 1 - 33,554

- 테이블에 레코드를 추가할 때 검색할 여유 공간 제어 레코드(FSCR)의 수를 지정합니다. 디폴트는 다섯 개의 FSCR을 검색하는 것입니다. 이 값을 수정하면 스페이스 재사용과 삽입 속도의 균형을 맞출 수 있습니다. 스페이스 재사용에 맞게 최적화하려면 큰 값을 사용하십시오. 삽입 속도에 맞게 최적화하려면 작은 값을 사용하십시오. 값을 -1로 설정하면 데이터베이스 관리 프로그램이 모든 FSCR을 강제로 검색합니다.

DB2_MAX_INACT_STMTS

- 운영 체제: 모두
- 디폴트=설정되지 않음, 값: 최대 4GB
- 이 변수는 하나의 응용프로그램에서 보존되는 비활성 명령문 수에 대한 디폴트 한계를 겹쳐씁니다. 비활성 명령문 정보에 사용되는 시스템 모니터 힙 크기를 늘리거나 줄이기 위해 다른 값을 선택할 수 있습니다. 디폴트 한계는 250입니다.

  응용프로그램이 작업 단위에 매우 많은 수의 명령문을 포함하고 있거나 동시에 실행 중인 응용프로그램이 많은 경우 시스템 모니터 힙이 모두 사용될 수 있습니다.

DB2_MAX_NON_TABLE_LOCKS

- 운영 체제: 모두
- 디폴트=YES, 값: 설명 참조
- 이 변수는 트랜잭션이 NON 테이블 잠금을 모두 릴리스하기 전에 가질 수 있는 최대 해당 잠금 수를 정의합니다. NON 테이블 잠금은 트랜잭션에서 사용을 완료한 경우에도 해시 테이블 및 트랜잭션 체인에 보존되는 테이블 잠금입니다. 트랜잭션이 자주 동일한 테이블에 두 번 이상 액세스하므로 잠금을 보유하고 상태를 NON으로 변경하면 성능이 향상될 수 있습니다.

  최상의 결과를 위해서 이 변수에 권장되는 값은 연결에 의해 액세스될 것으로 예측되는 최대 테이블 수입니다. 사용자 정의 값이 지정되지 않는 경우 디폴트값은 다음과 같습니다. 잠금 목록 크기가 다음보다 크거나 같은 경우

  ```
  SQLP_THRESHOLD_VAL_OF_LRG_LOCKLIST_SZ_FOR_MAX_NON_LOCKS
  ```

  (현재 8,000), 디폴트값은 다음과 같습니다.

  ```
  SQLP_DEFAULT_MAX_NON_TABLE_LOCKS_LARGE
  ```

  (현재 150). 그렇지 않은 경우 디폴트값은 다음과 같습니다.

  ```
  SQLP_DEFAULT_MAX_NON_TABLE_LOCKS_SMALL
  ```

  (현재 0).

DB2_MDC_ROLLOUT

- 운영 체제: 모두
- 디폴트=IMMEDIATE, 값: IMMEDIATE, OFF 또는 DEFER
- 이 변수는 MDC 테이블에서의 삭제에 『롤아웃』으로 알려진 성능 향상 기능을 사용 가능하게 합니다. 롤아웃은 전체 셀(차원 값의 교차)이 검색 DELETE문에서 삭제될 때 MDC 테이블에서 행을 삭제하는 빠른 방법입니다. 로깅 수가 감소되고 더 효율적으로 처리되는 것이 이점입니다.
- 세 가지 변수 설정 결과가 가능합니다.
  - 롤아웃 사용 안함 - OFF가 지정된 경우
  - 즉시 롤아웃 - IMMEDIATE가 지정된 경우
  - 인덱스 정리가 지연된 롤아웃 - DEFER가 지정된 경우
- 시작 후 값이 변경되면 명령문의 새 컴파일은 새 레지스트리 값 설정을 고려합니다. 패키지 캐시에 있는 명령문의 경우, 명령문이 재컴파일될 때까지 삭제 처리를 변경하지 않습니다. SET CURRENT MDC ROLLOUT MODE 명령문은 응용프로그램 연결 레벨의 **DB2_MDC_ROLLOUT** 값을 겹쳐씁니다.

DB2MEMDISCLAIM

- 운영 체제: 모두
- 디폴트=YES, 값: YES 또는 NO
- DB2 데이터베이스 시스템 프로세스에서 사용한 메모리에는 연관된 페이징 스페이스가 있을 수 있습니다. 이 페이징 스페이스는 연관된 메모리가 해제된 경우에도 예약된 채로 남아 있을 수 있습니다. 이에 대한 여부는 운영 체제의 조정 가능한 가상 메모리 관리 할당 규정에 따라 다릅니다. **DB2MEMDISCLAIM** 레지스트리 변수는 운영 체제가 해제된 메모리에서 예약된 페이징 스페이스를 연관 해제하도록 DB2 에이전트가 명시적으로 요청하는지 여부를 제어합니다.

  DB2MEMDISCLAIM을 YES로 설정하면 그 결과 페이징 스페이스 요구사항이 작아지고 페이징에서 디스크 활동이 적어질 수 있습니다. **DB2MEMDISCLAIM**을 NO로 설정하면 그 결과 페이징 스페이스 요구사항이 커지고 페이징에서 디스크 활동이 많아질 수 있습니다. 페이징 스페이스가 충분하고 실제 메모리도 충분하여 페이징이 발생하지 않는 경우와 같은 일부 상황에서 NO를 설정하면 성능이 약간 향상됩니다.

DB2MEMMAXFREE

- 운영 체제: 모두
- 디폴트=NULL, 값: 0 - 2^{32}-1 바이트

- 사용되지 않은 메모리가 운영 체제에 리턴되기 전에 DB2 데이터베이스 시스템에서 보유하는 사용되지 않은 전용 메모리의 최대 바이트 수를 지정합니다.

  DB2MEMMAXFREE가 설정되지 않은 경우, DB2 데이터베이스 시스템 프로세스는 사용할 수 있는 메모리를 다시 운영 체제에 돌려주기 전에 사용되지 않은 전용 메모리의 최대 20%를 보유합니다(현재 사용된 전용 메모리의 양을 기반으로 함).

주: DB2MEMMAXFREE는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 데이터베이스 관리 프로그램이 이제 스레드된 엔진 모델을 사용하므로 이 변수는 더 이상 필요하지 않습니다. 이 변수를 설정하지 마십시오. 이 변수를 설정하면 성능이 떨어지고 예기치 않은 동작을 일으킬 수 있습니다.

DB2_MEM_TUNING_RANGE

- 운영 체제: 모두
- 디폴트=NULL, 값: 백분율 시퀀스 n, m 여기서 n=*최소 여유 공간*이고 m=*최대 여유 공간*
- DB2 인스턴스가 여유 공간으로 남겨두는 실제 메모리의 *크기*는 같은 머신에서 실행 중인 다른 응용프로그램이 사용할 수 있는 메모리 *크기*를 결정하므로 중요합니다. 데이터베이스 공유 메모리의 자체 성능 조정이 사용 가능한 경우, 지정된 인스턴스가 여유 공간으로 남긴 실제 메모리의 *크기*는 인스턴스(및 해당 활성 데이터베이스)의 메모리 필요성에 따라 다릅니다. 인스턴스에 추가 메모리가 긴급하게 필요한 경우, 인스턴스는 시스템의 사용할 수 있는 실제 메모리가 *최소 여유 공간*에서 지정한 백분율에 접근할 때까지 메모리를 할당합니다. 인스턴스에 메모리가 적게 필요한 경우, 인스턴스는 *최대 여유 공간*에서 백분율로 지정한 많은 양의 사용할 수 있는 실제 메모리를 유지합니다. 그 결과, *최소 여유 공간*에 설정된 값이 *최대 여유 공간*의 값보다 작아야 합니다.

  이 변수가 설정되지 않은 경우, DB2 데이터베이스 관리 프로그램은 서버의 메모리 *크기*를 기준으로 *최소 여유 공간* 및 *최대 여유 공간*의 값을 계산합니다. 자체 성능 조정 메모리 관리자(STMM)를 실행 중이고 **database_memory**가 AUTOMATIC으로 설정되어 있지 않으면 이러한 변수 설정은 적용되지 않습니다.

DB2_MMAP_READ

- 운영 체제: AIX
- 디폴트=OFF, 값: ON 또는 OFF
- 이 변수를 **DB2_MMAP_WRITE**와 함께 사용하면 DB2 데이터베이스 시스템이 mmap를 대체 입출력 메소드로 사용할 수 있습니다.

두 변수가 ON으로 설정되면, 읽혀서 DB2 버퍼 풀에 전달되고 해당 버퍼 풀에서 작성된 데이터는 AIX 메모리 캐시를 무시합니다. 상대적으로 작은 DB2 버퍼 풀을 가지고 있고 이 버퍼 풀의 *크기*를 늘릴 수 없거나 늘리지 않도록 선택한 경우, **DB2_MMAP_READ** 및 **DB2_MMAP_WRITE**를 OFF로 설정하여 AIX 메모리 캐싱을 이용해야 합니다.

DB2_MMAP_WRITE

- 운영 체제: AIX
- 디폴트=OFF, 값: ON 또는 OFF
- 이 변수를 **DB2_MMAP_READ**와 함께 사용하면 DB2 데이터베이스 시스템이 mmap를 대체 입출력 메소드로 사용할 수 있습니다.

  두 변수가 ON으로 설정되면, 읽혀서 DB2 버퍼 풀에 전달되고 해당 버퍼 풀에서 작성된 데이터는 AIX 메모리 캐시를 무시합니다. 상대적으로 작은 DB2 버퍼 풀을 가지고 있고 이 버퍼 풀의 *크기*를 늘릴 수 없거나 늘리지 않도록 선택한 경우, **DB2_MMAP_READ** 및 **DB2_MMAP_WRITE**를 OFF로 설정하여 AIX 메모리 캐싱을 이용해야 합니다.

DB2_NO_FORK_CHECK

- 운영 체제: UNIX
- 디폴트=OFF, 값: ON 또는 OFF
- 이 변수가 사용 가능하면 DB2 런타임 클라이언트가 현재 프로세스가 포크 호출의 결과인지를 판별하기 위한 검사를 최소화합니다. 이는 fork() API를 사용하지 않는 DB2 응용프로그램의 성능을 향상시킬 수 있습니다.

주: 이 변수는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 프로세스가 시작되거나 새로 분기될 때 현재 프로세스 ID(pid)가 캐시되므로 필요하지 않습니다.

DB2NTMEMSIZE

- 운영 체제: Windows
- 디폴트=(메모리 세그먼트에 따라 다름)
- Windows에서는 프로세스 전체에 걸쳐서 주소 일치를 보장하기 위해 DLL 초기화 시간에 모든 공유 메모리 세그먼트가 예약되어야 합니다. **DB2NTMEMSIZE**를 사용하여 사용자는 필요한 경우 Windows의 DB2 디폴트를 겹쳐씁니다. 대부분의 상황에서는 디폴트값으로 충분합니다. 메모리 세그먼트, 디폴트 *크기* 및 겹쳐쓰기 옵션은 다음과 같습니다.
  1. 병렬 FCM 버퍼: 디폴트 *크기*는 32비트 플랫폼의 경우 512MB이고 64비트 플랫폼의 경우 4.5GB입니다. 겹쳐쓰기 옵션은 FCM:<바이트 수>입니다.

2. 분리(Fenced) 모드 통신: 디폴트 크기는 32비트 플랫폼의 경우 80MB이고 64비트 플랫폼의 경우 512MB입니다. 겹쳐쓰기 옵션은 APLD:<바이트 수>입니다.

세미콜론(;)으로 겹쳐쓰기 옵션을 구분하여 둘 이상의 세그먼트를 겹쳐쓸 수 있습니다. 예를 들어, 32비트 버전 DB2에서 FCM 버퍼를 1GB로 제한하고 분리 스토어드 프로시저를 256MB로 제한하려면 다음을 사용하십시오.

```
db2set DB2NTMEMSIZE=FCM:1073741824;APLD:268435456
```

DB2NTNOCACHE

- 운영 체제: Windows
- 디폴트=OFF, 값: ON 또는 OFF
- **DB2NTNOCACHE** 레지스트리 변수는 DB2 데이터베이스 시스템이 NOCACHE 옵션을 사용하여 데이터베이스 파일을 여는지 여부를 지정합니다. **DB2NTNOCACHE**=ON이면 파일 시스템 캐싱이 제거됩니다. **DB2NTNOCACHE**=OFF이면 운영 체제가 DB2 파일을 캐시합니다. 이는 Long 필드 또는 LOB를 포함하는 파일을 제외한 모든 데이터에 적용됩니다. 시스템 캐싱을 제거하면 버퍼 풀 또는 정렬 힙을 늘릴 수 있도록 더 많은 메모리를 데이터베이스에 사용할 수 있습니다.

  Windows에서는 파일이 열려 있으면 캐시되며 이는 디폴트 동작입니다. 파일의 1GB마다 1MB가 시스템 풀에서 예약됩니다. 캐시의 미등록 한계 192MB를 겹쳐쓰려면 이 레지스트리 변수를 사용하십시오. 캐시 한계에 접근하면 자원 부족 오류가 표시됩니다.

주: DB2NTNOCACHE는 버전 8.2 이후 사용되지 않고 있으며 추후 릴리스에서는 제거됩니다. CREATE TABLESPACE 및 ALTER TABLESPACE SQL문을 사용하여 테이블 스페이스 컨테이너에 대해 동일한 이점을 얻을 수 있습니다.

DB2NTPRICLASS

- 운영 체제: Windows
- 디폴트=NULL, 값: R, H, (기타 모든 값)
- DB2 인스턴스의 우선순위 클래스를 설정합니다(프로그램 DB2SYSCS.EXE). 세 가지 우선순위 클래스가 있습니다.
  - NORMAL_PRIORITY_CLASS(디폴트 우선순위 클래스)
  - REALTIME_PRIORITY_CLASS(『R』을 사용하여 설정)
  - HIGH_PRIORITY_CLASS(『H』를 사용하여 설정)

이 변수는 개별 스레드 우선순위(**DB2PRIORITIES**를 사용하여 설정)와 함께 사용되어 시스템의 기타 스레드와 비교하여 DB2 스레드의 절대 우선순위를 판별합니다.

주: DB2NTPRICLASS는 사용되지 않으며 서비스 권장사항에서만 사용되어야 합니다. 에이전트 우선순위 및 프리페치 우선순위를 조정하려면 DB2 서비스 클래스를 사용하십시오. 이 변수를 사용하려면 주의하십시오. 잘못 사용하면 전체 시스템 성능에 나쁜 영향을 미칠 수 있습니다.

자세한 정보는 Win32 문서에서 SetPriorityClass() API를 참조하십시오.

DB2NTWORKSET

- 운영 체제: Windows
- 디폴트=1,1
- DB2 데이터베이스 관리 프로그램에 사용 가능한 최소 및 최대 작업 세트 *크기*를 수정하는 데 사용됩니다. 디폴트로 Windows가 페이징 상황에 있지 않으면 프로세스의 작업 세트는 필요한 만큼 커질 수 있습니다. 그러나 페이징이 발생하면 프로세스가 가질 수 있는 최대 작업 세트는 약 1MB입니다. **DB2NTWORKSET**를 사용하여 이 디폴트 동작을 겹쳐쓸 수 있습니다.

  DB2NTWORKSET=min, max 구문을 사용하여 **DB2NTWORKSET**를 지정하십시오. 여기서 min 및 max는 MB로 표시됩니다.

DB2_OVERRIDE_BPF

- 운영 체제: 모두
- 디폴트=설정되지 않음, 값: 양의 페이지 수 또는 <항목>[;<항목>...] 여기서 <항목>=<버퍼 풀 ID>,<페이지 수>
- 이 변수는 데이터베이스 활성화, 롤 포워드 복구 또는 응급 복구 시 작성될 버퍼 풀의 *크기*를 지정합니다(페이지 단위). 이 변수는 메모리 제한조건으로 인해 데이터베이스 활성화, 롤 포워드 복구 또는 응급 복구 동안 오류가 발생하는 경우에 유용합니다. 메모리 제한조건은 드물게 실제 메모리 부족인 경우나 또는 데이터베이스 관리 프로그램이 대형 버퍼 풀을 할당하려는 시도 때문에 잘못 구성된 버퍼 풀이 있는 경우에 발생할 수 있습니다. 예를 들어, 데이터베이스 관리 프로그램이 16페이지의 최소 버퍼 풀을 불러오지 않은 경우에도 이 환경 변수를 사용하여 더 작은 페이지 수를 지정하십시오. 이 변수에 지정된 값이 현재 버퍼 풀 *크기*를 겹쳐씁니다.

  또한 <항목>[;<항목>...]을 사용하여 버퍼 풀이 시작될 수 있도록 모든 버퍼 풀 또는 버퍼 풀의 서브세트 *크기*를 임시로 변경할 수 있습니다. 여기서 <항목>=<버퍼 풀 ID>,<페이지 수>입니다.

DB2_PINNED_BP

- 운영 체제: AIX, HP-UX, Linux
- 디폴트=NO, 값: YES 또는 NO
- 이 변수를 YES로 설정하면 운영 체제가 DB2의 데이터베이스 공유 메모리를 고정하도록 DB2가 요청하는 원인이 됩니다. 데이터베이스 공유 메모리를 고정하도록 DB2를 구성하는 경우, 운영 체제가 메모리 관리의 유연성을 줄이므로 시스템이 오버커미트되지 않았는지 주의해서 확인해야 합니다.

  Linux의 경우, 이 레지스트리 변수 수정에 더하여 *libcap.so.1* 라이브러리도 필요합니다.

  이 변수를 YES로 설정하면 데이터베이스 공유 메모리의 자체 성능 조정을 사용할 수 없습니다. 데이터베이스 공유 메모리의 자체 성능 조정은 *database_memory* 구성 매개변수를 AUTOMATIC으로 설정하여 활성화됩니다.

  64비트 환경의 HP-UX의 경우, 이 레지스트리 변수 수정과 함께 DB2 인스턴스 그룹에 MLOCK 특권을 지정해야 합니다. 이를 수행하기 위해 루트 액세스 권한이 있는 사용자가 다음 조치를 수행합니다.

  1. DB2 인스턴스 그룹을 `/etc/privgroup` 파일에 추가합니다. 예를 들어, DB2 인스턴스 그룹이 `db2iadm1` 그룹에 속하면 `/etc/privgroup` 파일에 다음 라인을 추가해야 합니다.

     ```
     db2iadm1 MLOCK
     ```

  2. 다음 명령을 발행하십시오.

     ```
     setprivgrp -f /etc/privgroup
     ```

DB2PRIORITIES

- 운영 체제: 모두
- 값 설정은 플랫폼에 따라 다름
- DB2 프로세스 및 스레드의 우선순위를 제어합니다.

주: DB2PRIORITIES는 사용되지 않으며 서비스 권장사항에서만 사용되어야 합니다. 에이전트 우선순위 및 프리페치 우선순위를 조정하려면 DB2 서비스 클래스를 사용하십시오.

DB2_RESOURCE_POLICY

- 운영 체제: AIX 5 이상, zSeries를 제외한 모든 Linux(32비트), Windows Server 2003 이상
- 디폴트=설정되지 않음, 값: 구성 파일의 유효한 경로
- DB2 데이터베이스에서 사용하는 운영 체제 자원을 제한하는 데 사용할 수 있는 자원 규정을 정의하거나 특정 운영 체제 자원을 특정 DB2 데이터베이스 오브젝트에 할당하는 규칙을 포함합니다. 예를 들어, AIX, Linux 또는

Windows 운영 체제에서 이 레지스트리 변수는 DB2 데이터베이스 시스템이 사용하는 프로세서 세트를 제한하는 데 사용할 수 있습니다. 자원 제어의 범위는 운영 체제에 따라 다릅니다.

AIX NUMA 및 Linux NUMA 사용 가능 머신에서는 DB2 데이터베이스 시스템이 사용하는 자원 세트를 지정하는 규정을 정의할 수 있습니다. 자원 세트 바인딩이 사용되는 경우, 각 개별 DB2 프로세스는 특정 자원 세트에 바인드됩니다. 이는 일부 성능 조정 시나리오에서 유용할 수 있습니다.

이 레지스트리 변수를 설정하여 DB2 프로세스를 운영 체제 자원에 바인딩하기 위한 규정을 정의하는 구성 파일의 경로를 표시할 수 있습니다. 자원 규정을 사용하여 DB2 데이터베이스 시스템을 제한하는 운영 체제 자원 세트를 지정할 수 있습니다. 각 DB2 프로세스는 세트의 단일 자원에 바인드됩니다. 자원 지정은 순환 라운드 로빈 방식으로 발생합니다.

샘플 구성 파일은 다음과 같습니다.

예 1: 모든 DB2 프로세스를 CPU 1 또는 3에 바인드하십시오.

```
<RESOURCE_POLICY>
  <GLOBAL_RESOURCE_POLICY>
   <METHOD>CPU</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>1</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>3</RESOURCE>
   </RESOURCE_BINDING>
  </GLOBAL_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

예 2: (AIX 전용) DB2 프로세스를 자원 세트 sys/node.03.00000, sys/node.03.00001,
sys/node.03.00002 및 sys/node.03.00003 중 하나에 바인드하십시오.

```
<RESOURCE_POLICY>
  <GLOBAL_RESOURCE_POLICY>
   <METHOD>RSET</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00000</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00001</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00002</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00003</RESOURCE>
   </RESOURCE_BINDING>
  </GLOBAL_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

메모: AIX 전용의 경우, RSET 메소드를 사용하려면 CAP_NUMA_ATTACH 기능이 필요합니다.

예 3: (Linux 전용) SAMPLE 데이터베이스와 연관된 버퍼 풀 ID 2 및 3의 모든 메모리를 NUMA 노드 3에 바인드하십시오. 또한 NUMA 노드 3에 바인드하는 데 전체 데이터베이스 메모리의 80퍼센트를 사용하고 20퍼센트는 버퍼 풀 특정이 아닌 메모리의 모든 노드에 걸쳐서 스트라이프되도록 남겨 두십시오.

```
<RESOURCE_POLICY>
  <DATABASE_RESOURCE_POLICY>
   <DBNAME>sample</DBNAME>
   <METHOD>NODEMASK</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>3</RESOURCE>
    <DBMEM_PERCENTAGE>80</DBMEM_PERCENTAGE>
    <BUFFERPOOL_BINDING>
     <BUFFERPOOL_ID>2</BUFFERPOOL_ID>
     <BUFFERPOOL_ID>3</BUFFERPOOL_ID>
    </BUFFERPOOL_BINDING>
   </RESOURCE_BINDING>
  </DATABASE_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

예 4: (Linux 및 Windows 전용) CPU 마스크 0x0F 및 0xF0으로 지정한 두 가지 구별 프로세서 세트를 정의하십시오. DB2 프로세스 및 버퍼 풀 ID 2를 프로세서 세트 0x0F에 바인드하고 DB2 프로세스 및 버퍼 풀 ID 3을 프로세서 세트 0xF0에 바인드하십시오. 각 프로세서 세트에 대해 전체 데이터베이스 메모리의 50퍼센트를 바인딩에 사용하십시오.

이 자원 규정은 프로세서와 NUMA 노드 간의 맵핑을 원하는 경우에 유용합니다. 이러한 시나리오의 예는 8개의 프로세서와 2개의 NUMA 노드가 있으며 프로세서 0 - 3은 NUMA 노드 0에 속하고 프로세서 4 - 7은 NUMA 노드 1에 속하는 시스템입니다. 이 자원 규정은 메모리 집약성(즉, CPU 메소드 및 NODEMASK 메소드의 하이브리드)을 내재적으로 유지하는 동안 프로세서 바인딩을 허용합니다.

```
 <RESOURECE_POLICY>
   <DATABASE_RESOURCE_POLICY>
     <DBNAME>sample</DBNAME>
     <METHOD>CPUMASK</METHOD>
     <RESOURCE_BINDING>
       <RESOURCE>0x0F</RESOURCE>
       <DBMEM_PERCENTAGE>50</DBMEM_PERCENTAGE>
       <BUFFERPOOL_BINDING>
         <BUFFERPOOL_ID>2</BUFFERPOOL_ID>
       </BUFFERPOOL_BINDING>
     </RESOURCE_BINDING>
     <RESOURCE_BINDING>
       <RESOURCE>0x0F</RESOURCE>
       <DBMEM_PERCENTAGE>50</DBMEM_PERCENTAGE>
```

```
      <BUFFERPOOL_BINDING>
        <BUFFERPOOL_ID>3</BUFFERPOOL_ID>
      </BUFFERPOOL_BINDING>
    </RESOURCE_BINDING>
  </DATABASE_RESOURCE_POLICY>
</RESOURCE_POLICY>
```

주: RSET 메소드를 사용하려면 CAP_NUMA_ATTACH 기능이 필요하며 Linux에서는 지원되지 않습니다.

DB2_RESOURCE_POLICY 레지스트리 변수에 의해 지정된 구성 파일은 SCHEDULING_POLICY 요소를 승인합니다. 일부 플랫폼에서는 SCHEDULING_POLICY를 사용하여 다음을 선택할 수 있습니다.

– DB2 서버에서 사용되는 운영 체제 스케줄링 규정

  DB2NTPRICLASS 레지스트리 변수를 사용하여 AIX의 DB2 및 Windows의 DB2에 대한 운영 체제 스케줄링 규정을 설정할 수 있습니다.

– 개별 DB2 서버 에이전트에서 사용되는 운영 체제 우선순위

또는 레지스트리 변수 **DB2PRIORITIES** 및 **DB2NTPRICLASS**를 사용하여 운영 체제 스케줄링 규정을 제어하고 DB2 에이전트 우선순위를 설정할 수 있습니다. 그러나 자원 규정 구성 파일에 있는 SCHEDULING_POLICY 요소의 스펙은 스케줄링 규정 및 연관된 에이전트 우선순위를 모두 지정하는 단일 위치를 제공합니다.

예 1: db2 로그 기록기 및 판독기 프로세스에 대한 우선순위 높임을 포함한 AIX SCHED_FIFO2 스케줄링의 선택.

```
<RESOURCE_POLICY>
 <SCHEDULING_POLICY>
  <POLICY_TYPE>SCHED_FIFO2</POLICY_TYPE>
  <PRIORITY_VALUE>60</PRIORITY_VALUE>

  <EDU_PRIORITY>
   <EDU_NAME>db2loggr</EDU_NAME>
   <PRIORITY_VALUE>56</PRIORITY_VALUE>
  </EDU_PRIORITY>

  <EDU_PRIORITY>
   <EDU_NAME>db2loggw</EDU_NAME>
   <PRIORITY_VALUE>56</PRIORITY_VALUE>
  </EDU_PRIORITY>
 </SCHEDULING_POLICY>
</RESOURCE_POLICY>
```

예 2: Windows에서 DB2NTPRICLASS=H 교체.

```
<RESOURCE_POLICY>
 <SCHEDULING_POLICY>
  <POLICY_TYPE>HIGH_PRIORITY_CLASS</POLICY_TYPE>
 </SCHEDULING_POLICY>
</RESOURCE_POLICY>
```

DB2_SET_MAX_CONTAINER_SIZE

- 운영 체제: 모두
- 디폴트=설정되지 않음, 값: -1, 65,536바이트보다 큰 임의의 양의 정수
- 이 레지스트리 변수를 사용하여 AutoResize 기능이 사용 가능한 자동 스토리지 테이블 스페이스에 대한 개별 컨테이너 크기를 제한할 수 있습니다.

  주: **DB2_SET_MAX_CONTAINER_SIZE**를 바이트, KB 또는 MB로 지정할 수 있지만 db2set은 해당 값을 바이트로 표시합니다.
- 값이 -1로 설정된 경우에는 컨테이너 크기의 한계가 없습니다.

DB2_SKIPDELETED

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- 사용 가능한 경우, 이 변수를 사용하면 커서 안정성 또는 읽기 안정성 분리 레벨을 사용하는 명령문이 무조건 인덱스 액세스 중에는 삭제된 키를 건너뛰고 테이블 액세스 중에는 삭제된 행을 건너뛸 수 있습니다. **DB2_EVALUNCOMMITTED**가 사용 가능한 경우, 삭제된 행은 자동으로 건너뛰지만 인덱스의 언커미트된 의사(pseudo) 삭제된 키는 **DB2_SKIPDELETED**도 사용 가능한 경우 외에는 건너뛰지 않습니다.

  DB2_SKIPDELETED는 현재 커미트된 시맨틱이 잠금 경합을 방지하는 데 도움이 되지 않는 경우에만 적용 가능합니다. 이 변수가 설정되고 현재 커미트된 스캔에 적용 가능하면 삭제된 행을 건너뛰지 않습니다. 대신에 현재 커미트된 버전이 처리됩니다.

  이 레지스트리 변수는 DB2 카탈로그 테이블의 커서 동작에 영향을 미치지 않습니다.

  이 레지스트리 변수는 db2start 명령으로 활성화됩니다.

DB2_SKIPINSERTED

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- **DB2_SKIPINSERTED** 레지스트리 변수가 사용 가능한 경우, 이 변수를 사용하면 커서 안정성 또는 읽기 안정성 분리 레벨을 사용하는 명령문이 언커미트된 삽입된 행을 삽입되지 않은 것처럼 건너뛸 수 있습니다. 이 레지스트리 변수는 DB2 카탈로그 테이블의 커서 동작에 영향을 미치지 않습니다. 이

레지스트리 변수는 데이터베이스 시작 시 활성화되는 반면에 언커미트된 삽입된 행을 건너뛰는 결정은 명령문 컴파일 또는 바인드 시간에 수행됩니다.

현재 커미트된 시맨틱이 사용 중인 경우 이 레지스트리 변수는 효력이 없습니다. 즉, **DB2_SKIPINSERTED**가 OFF로 설정되었고 현재 커미트된 동작이 사용 가능한 경우에도 언커미트된 삽입된 행은 계속 건너뜁니다.

주: 삽입된 행 건너뛰기 동작은 보류 중인 롤아웃 정리가 있는 테이블과 호환되지 않습니다. 그 결과, 스캐너는 RID에서 잠금을 대기하다가 RID가 롤아웃된 블록의 일부임을 발견하게 됩니다.

DB2_SMS_TRUNC_TMPTABLE_THRESH

- 운영 체제: 모두
- 디폴트=0, 값: -1, 0-n, 여기서 n=SMS 테이블 스페이스 컨테이너에서 유지되는 임시 테이블당 Extent 수
- 이 변수는 임시 테이블을 나타내는 파일이 SMS 테이블 스페이스에서 유지되는 최소 파일 크기 임계값을 지정합니다.

  디폴트로 이 변수는 0으로 설정되며 이는 특수 임계값 조절이 수행되지 않음을 의미합니다. 대신에 임시 테이블이 더 이상 필요하지 않으면 해당 파일은 0 Extent로 절단됩니다.

  이 변수의 값이 0보다 크면 더 큰 파일이 유지됩니다. 이는 임시 테이블이 사용될 때마다 파일 삭제 및 재작성과 관련된 시스템 오버헤드를 일부 줄입니다.

  이 변수가 -1로 설정된 경우, 파일은 절단되지 않고 무한정 커질 수 있으며 시스템 자원에 의해서만 제한됩니다.

DB2_SORT_AFTER_TQ

- 운영 체제: 모두
- 디폴트=NO, 값: YES 또는 NO
- 수신 종료 시 데이터를 정렬해야 하고 수신 노드 수와 보내기 노드 수가 같을 때 파티션된 데이터베이스 환경에서 옵티마이저가 방향지정된 테이블 큐에 대해 작업하는 방법을 지정합니다.

  DB2_SORT_AFTER_TQ=NO로 설정되면 옵티마이저는 보내기 종료 시 정렬하고 수신 종료 시 행을 병합하는 경향이 있습니다.

  DB2_SORT_AFTER_TQ=YES로 설정되면 옵티마이저는 수신 종료 시 정렬되지 않은 행을 전송하고 병합하지 않고 모든 행을 수신한 후 수신 종료 시 행을 정렬하는 경향이 있습니다.

DB2_SELUDI_COMM_BUFFER

- 운영 체제: 모두
- 디폴트=OFF, 값: ON 또는 OFF
- 이 변수는 UPDATE, INSERT 또는 DELETE(UDI) 쿼리에서 SELECT에 대한 블로킹 커서를 처리하는 동안 사용됩니다. 사용 가능한 경우, 이 레지스트리 변수는 쿼리 결과가 임시 테이블에 저장되지 않도록 합니다. 대신에 UDI 쿼리에서 SELECT에 대한 블로킹 커서의 OPEN 처리 중에 DB2 데이터베이스 시스템은 쿼리의 전체 결과를 통신 버퍼 메모리 영역에 직접 버퍼하려고 시도합니다.

  주: 통신 버퍼 스페이스가 전체 쿼리 결과를 보유할 만큼 크지 않은 경우, SQLCODE -906 오류가 발행되고 트랜잭션은 롤백됩니다. 로컬 및 리모트 응용프로그램 각각에 대한 통신 버퍼 메모리 영역의 크기 조정에 대한 정보는 *aslheapsz* 및 *rqrioblk* 데이터베이스 관리 프로그램 구성 매개변수를 참조하십시오.

  이 레지스트리 변수는 파티션된 데이터베이스 환경이나 또는 파티션 내 병렬처리가 사용 가능한 경우에 지원되지 않습니다.

DB2_TRUSTED_BINDIN

- 운영 체제: 모두
- 디폴트=OFF, 값: OFF, ON 또는 CHECK
- **DB2_TRUSTED_BINDIN**이 사용 가능한 경우, 이 변수는 분리되지 않은 임베디드(embedded) 스토어드 프로시저 내에 호스트 변수를 포함하는 쿼리 명령문의 실행 속도를 높입니다.

  이 변수가 사용 가능한 경우, 분리되지 않은 임베디드 스토어드 프로시저 내에 포함된 SQL 및 XQuery문을 바인드하는 동안 외부 SQLDA 형식에서 내부 DB2 형식으로의 변환이 발생하지 않습니다. 이는 Embedded SQL 및 XQuery문의 처리 속도를 높입니다.

  이 변수가 사용 가능한 경우 다음 데이터 유형은 분리되지 않은 임베디드 스토어드 프로시저에서 지원되지 않습니다.

  – SQL_TYP_DATE
  – SQL_TYP_TIME
  – SQL_TYP_STAMP
  – SQL_TYP_CGSTR
  – SQL_TYP_BLOB
  – SQL_TYP_CLOB
  – SQL_TYP_DBCLOB
  – SQL_TYP_CSTR

- SQL_TYP_LSTR
- SQL_TYP_BLOB_LOCATOR
- SQL_TYP_CLOB_LOCATOR
- SQL_TYP_DCLOB_LOCATOR
- SQL_TYP_BLOB_FILE
- SQL_TYP_CLOB_FILE
- SQL_TYP_DCLOB_FILE
- SQL_TYP_BLOB_FILE_OBSOLETE
- SQL_TYP_CLOB_FILE_OBSOLETE
- SQL_TYP_DCLOB_FILE_OBSOLETE

이러한 데이터 유형이 발견되면 SQLCODE -804, SQLSTATE 07002 오류가 리턴됩니다.

주: 입력 호스트 변수의 데이터 유형 및 길이는 해당 요소의 내부 데이터 유형 및 길이와 정확히 일치해야 합니다. 호스트 변수의 경우, 이 요구사항은 항상 충족됩니다. 그러나 매개변수 표시문자의 경우에는 일치하는 데이터 유형이 사용되는지 주의해서 확인해야 합니다. CHECK 옵션을 사용하여 모든 입력 호스트 변수에 대해 데이터 유형 및 길이가 일치하는지 확인할 수 있지만 이 옵션은 대부분의 성능 향상을 무효화합니다.

주: DB2_TRUSTED_BINDIN은 사용되지 않으며 추후 릴리스에서는 제거됩니다.

DB2_USE_ALTERNATE_PAGE_CLEANING

- 운영 체제: 모두
- 디폴트=설정되지 않음, 값: ON 또는 OFF
- 이 변수는 DB2 데이터베이스가 대체 페이지 정리 알고리즘 메소드를 사용하는지 또는 디폴트 페이지 정리 메소드를 사용하는지를 지정합니다. 이 변수가 ON으로 설정되면 DB2 시스템이 변경된 페이지를 디스크에 기록하여 LSN_GAP의 어헤드를 보존하고 혁신적으로 희생(victim)을 찾습니다. 이를 수행하면 페이지 클리너가 사용 가능한 디스크 입출력 대역폭을 더 잘 사용할 수 있습니다. 이 변수가 ON으로 설정되면 *chngpgs_thresh* 데이터베이스 구성 매개변수는 페이지 클리너 활동을 제어하지 않으므로 더 이상 관련되지 않습니다.

DB2_USE_IOCP

- 운영 체제: AIX 5.3 TL9 SP2 또는 AIX 6.1 TL2
- 디폴트: ON 값: OFF 또는 ON

- 이 변수를 사용하면 비동기 I/O(AIO) 요청을 제출하고 수집할 때 AIX I/O 완료 포트(IOCP)를 사용할 수 있습니다. 이 기능은 리모트 메모리 액세스를 방지하여 NUMA(Non-Uniform Memory Access) 환경에서 성능을 향상하는 데 사용됩니다.

기타 변수

DB2ADMINSERVER

- 운영 체제: Windows 및 UNIX
- 디폴트: NULL
- DB2 Administration Server를 지정합니다.

DB2_ATS_ENABLE

- 운영 체제: 모두
- 디폴트: NULL, 값: YES/TRUE/ON/1 또는 NO/FALSE/OFF/0
- 이 변수는 관리 태스크 스케줄러가 실행 중인지 여부를 제어합니다. 관리 태스크 스케줄러는 디폴트로 사용하지 않습니다. 스케줄러를 사용하지 않는 경우, 내장 프로시저 및 보기를 사용하여 태스크를 정의하고 수정할 수 있지만 스케줄러는 태스크를 실행하지 않습니다.

DB2AUTH

- 운영 체제: 모두
- 디폴트: 설정되지 않음. 값: TRUSTEDCLIENT_SRVRENC, TRUSTEDCLIENT_DATAENC, DISABLE_CHGPASS, OSAUTHDB
- 이 변수를 사용하여 사용자 인증 동작을 조정할 수 있습니다.
  - TRUSTEDCLIENT_SRVRENC: 이 값은 트러스트되지 않은 클라이언트가 SERVER_ENCRYPT를 사용하도록 강제합니다.
  - TRUSTEDCLIENT_DATAENC: 이 값은 트러스트되지 않은 클라이언트가 DATA_ENCRYPT를 사용하도록 강제합니다.
  - DISABLE_CHGPASS: 이 값은 클라이언트의 암호 변경 기능을 사용하지 않도록 설정합니다.
  - OSAUTHDB: 이 값은 DB2 데이터베이스 관리 프로그램이 AIX 운영 체제의 사용자에 대한 인증 및 그룹 설정을 사용하도록 지시합니다. LDAP 서버는 다음 중 하나일 수 있습니다.
    - IBM Tivoli Directory Server(ITDS)
    - Microsoft Active Directory(MSAD)
    - Sun One Directory Server

DB2CLIINIPATH

- 운영 체제: 모두

- 디폴트: NULL
- DB2 CLI/ODBC 구성 파일의 디폴트 경로(db2cli.ini)를 겹쳐쓰고 클라이언트에 다른 위치를 지정하는 데 사용됩니다. 지정된 값은 클라이언트 시스템에서 유효한 경로여야 합니다.

DB2_COMMIT_ON_EXIT

- 운영 체제: UNIX
- 디폴트: OFF, 값: OFF/NO/0 또는 ON/YES/1
- UNIX 운영 체제에서 DB2 UDB 버전 8 이전에 DB2는 성공적인 응용프로그램 종료 시 남아 있는 모든 인플라이트(inflight) 트랜잭션을 커미트했습니다. DB2 UDB 버전 8에서는 종료 시 인플라이트 트랜잭션이 롤백되도록 동작이 변경되었습니다. 이 레지스트리 변수를 사용하여 이전 동작에 따라 다른 Embedded SQL 응용프로그램을 사용하는 사용자가 DB2 버전 9에서 이를 계속 사용할 수 있습니다. 이 레지스트리 변수는 JDBC, CLI 및 ODBC 응용프로그램에 영향을 미치지 않습니다.

  이 레지스트리 변수는 사용되지 않으며 추후 릴리스에서는 종료 시 커미트 동작이 더 이상 지원되지 않습니다. 사용자는 DB2 버전 9 이전에 개발된 응용프로그램이 계속 이 기능에 종속되는지를 판별하고 필요에 따라 응용프로그램에 적당한 명시적 COMMIT 또는 ROLLBACK문을 추가해야 합니다. 레지스트리 변수가 켜져 있으면 종료 전에는 명시적으로 커미트할 수 없는 새 응용프로그램을 구현하지 않도록 주의해야 합니다.

  대부분의 사용자는 이 레지스트리 변수를 디폴트 설정으로 두어야 합니다.

DB2CONNECT_DISCONNECT_ON_INTERRUPT

- 운영 체제: 모두
- 디폴트: NO, 값: YES/TRUE/1 또는 NO/FALSE/0
- YES(TRUE 또는 1)로 설정되면 이 변수는 인터럽트가 발생하면 버전 8(또는 그 이상) DB2 Universal Database z/OS 서버에 대한 연결이 즉시 끊어지도록 지정합니다. 다음과 같은 구성에 이 변수를 사용할 수 있습니다.
  - 버전 8(또는 그 이상) DB2 UDB z/OS 서버와 함께 DB2 클라이언트를 실행 중인 경우, 클라이언트에서 **DB2CONNECT_DISCONNECT_ON_INTERRUPT**를 YES로 설정하십시오.
  - 버전 8(또는 그 이상) DB2 UDB z/OS 서버에 대한 DB2 Connect 게이트웨이를 통해 DB2 클라이언트를 실행 중인 경우, 게이트웨이에서 **DB2CONNECT_DISCONNECT_ON_INTERRUPT**를 YES로 설정하십시오.

DB2_CREATE_DB_ON_PATH

- 운영 체제: Windows
- 디폴트: NULL, 값: YES 또는 NO
- 경로(및 드라이브)를 데이터베이스 경로로 사용하기 위한 지원을 사용하려면 이 레지스트리 변수를 YES로 설정하십시오.

  DB2_CREATE_DB_ON_PATH 설정은 데이터베이스가 작성될 때, 데이터베이스 관리 프로그램 구성 매개변수 **dftdbpath**가 설정될 때 및 데이터베이스가 리스토어될 때 점검됩니다. 완전한 데이터베이스 경로의 길이는 최대 215자입니다.

  DB2_CREATE_DB_ON_PATH는 설정하지 않고(또는 NO로 설정) 데이터베이스를 작성하거나 리스토어할 때 데이터베이스 경로에 경로를 지정하면 SQL1052N 오류가 리턴됩니다.

  DB2_CREATE_DB_ON_PATH는 설정하지 않고(또는 NO로 설정) **dftdbpath** 데이터베이스 관리 프로그램 구성 매개변수를 갱신하면 SQL5136N 오류가 리턴됩니다.

  주의:
  경로 지원을 사용하여 새 데이터베이스를 작성하는 경우, db2DbDirGetNextEntry() API 또는 해당 API의 이전 버전을 사용하여 DB2 버전 9.1 이전에 작성된 응용프로그램은 올바르게 작동하지 않습니다. 여러 가지 시나리오 및 적절한 조치 과정에 대한 자세한 내용은 http://www.ibm.com/software/data/db2/support/db2_9/를 참조하십시오.

DB2_DDL_SOFT_INVAL

- 운영 체제: 모두
- 디폴트: ON, 값: ON 또는 OFF
- 적용 가능한 데이터베이스 오브젝트가 삭제 또는 변경될 때 소프트 무효화를 사용 가능하게 합니다.

  DB2_DDL_SOFT_INVAL이 ON으로 설정된 경우, 동일한 오브젝트를 참조하는 트랜잭션이 완료되기를 기다리지 않고 DDL 조작(예: 삭제(drop), 변경 또는 접속 해제)을 시작할 수 있습니다. 오브젝트에 종속된 현재 실행은 원래 오브젝트 정의를 사용하여 계속 진행되는 반면, 새 실행은 변경된 오브젝트를 사용합니다. 이는 DDL문 발행 시 동시성을 높입니다.

  주: 새 소프트 무효화 성능은 동적 패키지에만 적용됩니다. 정적 패키지가 있는 오브젝트는 여전히 하드 무효화를 필요로 합니다.

DB2DEFPREP

- 운영 체제: 모두
- 디폴트: NO, 값: ALL, YES 또는 NO

- 이 옵션을 사용할 수 있기 전에 프리컴파일된 응용프로그램에 대해 **DEFERRED_PREPARE** 프리컴파일 옵션의 런타임 동작을 가상합니다. 예를 들어, DB2 v2.1.1 또는 이전 응용프로그램이 DB2 v2.1.2 또는 이후 환경에서 실행된 경우, **DB2DEFPREP**를 사용하여 원하는 『지연된 준비』 동작을 표시할 수 있습니다.

주: DB2DEFPREP는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 이 변수는 **DEFERRED_PREPARE** 프리컴파일 옵션이 사용 불가능한 DB2 이전 버전을 사용하는 사용자에게만 필요합니다.

DB2_DISABLE_FLUSH_LOG

- 운영 체제: 모두
- 디폴트: OFF, 값: ON 또는 OFF
- 온라인 백업이 완료되었을 때 사용 중인 로그 파일 닫기를 사용하지 않는지 여부를 지정합니다.

  온라인 백업이 완료되면 마지막 사용 중인 로그 파일이 절단되고 닫히고 아카이브될 수 있습니다. 이는 온라인 백업이 복구에 사용할 수 있는 완전한 아카이브 로그 세트를 갖도록 보장합니다. 로그 시퀀스 번호(LSN) 스페이스의 분할 영역을 낭비하는 것을 걱정하는 경우, 마지막 사용 중인 로그 파일 닫기를 사용 불가능하게 할 수 있습니다. 사용 중인 로그 파일이 절단될 때마다 절단된 스페이스에 비례하는 양만큼 LSN이 증가됩니다. 매일 여러 번 온라인 백업을 수행하는 경우, 마지막 사용 중인 로그 파일 닫기를 사용 불가능하게 할 수 있습니다.

  또한 온라인 백업이 완료되자마자 로그가 가득 찬 메시지를 검색하는 경우에도 마지막 사용 중인 로그 파일 닫기를 사용 불가능하게 할 수 있습니다. 로그 파일이 절단되면 절단된 로그 크기에 비례하는 양만큼 예약된 사용 중인 로그 스페이스가 증가됩니다. 절단된 로그 파일이 재개되면 사용 중인 로그 스페이스가 해제됩니다. 재개는 로그 파일이 비활성화된 후 곧 발생합니다. 이러한 두 이벤트 사이의 짧은 시간 간격 동안 로그가 가득 찬 메시지를 수신할 수도 있습니다.

  백업이 로그를 포함하려면 사용 중인 로그 파일을 절단하고 닫아야 하기 때문에 로그를 포함하는 백업 중에 이 레지스트리 변수는 무시됩니다.

DB2_DISPATCHER_PEEKTIMEOUT

- 운영 체제: 모두
- 디폴트: 1, 값: 0 - 32,767초; 0은 곧 시간종료임을 표시함
- **DB2_DISPATCHER_PEEKTIMEOUT**을 사용하여 클라이언트를 에이전트에 전달하기 전에 디스패처가 클라이언트의 연결 요청을 대기하는 시간(초)을 조정할 수 있습니다. 대부분의 경우, 이 레지스트리 변수를 조정할 필요

가 없습니다. 이 레지스트리 변수는 사용 가능한 DB2 Connect 연결 집중기(connection concentrator)를 가지고 있는 인스턴스에만 영향을 미칩니다.

이 레지스트리 변수와 **DB2_SERVER_CONTIMEOUT** 레지스트리 변수는 모두 연결 시간 동안 새 클라이언트의 처리를 구성합니다. 느린 클라이언트가 인스턴스에 여러 개 연결된 경우, 각 클라이언트를 시간종료시키기 위해 디스패처는 최대 1초 동안 보류되며 이는 많은 클라이언트가 동시에 연결되어 있는 경우에 디스패처가 병목이 되는 원인이 됩니다. 여러 개의 활성 데이터베이스를 포함한 인스턴스에서 매우 느린 연결 시간이 발생하는 경우, **DB2_DISPATCHER_PEEKTIMEOUT**은 0까지 내려갑니다. **DB2_DISPATCHER_PEEKTIMEOUT**을 낮추면 디스패처는 클라이언트의 연결 요청이 이미 있는 경우에 해당 요청을 살펴보기만 합니다. 디스패처는 연결 요청이 도착할 때까지 기다리지 않습니다. 유효하지 않은 값이 설정된 경우, 디폴트값이 사용됩니다. 이 레지스트리 변수는 동적이 아닙니다.

DB2_DJ_INI

- 운영 체제: 모두
- 디폴트:
  - UNIX: *db2_instance_directory*/cfg/db2dj.ini
  - Windows: *db2_install_directory*₩cfg₩db2dj.ini
- 페더레이션 구성 파일의 절대 경로 이름을 지정합니다(예: `db2set DB2_DJ_INI=$HOME/sqllib/cfg/my_db2dj.ini`). 이 파일은 데이터 소스 환경 변수에 대한 설정을 포함합니다. 이러한 환경 변수는 Informix 랩퍼 및 InfoSphere Federation Server에서 제공하는 랩퍼에서 사용됩니다.

  샘플 페더레이션 구성 파일은 다음과 같습니다.

```
INFORMIXDIR=/informix/client_sdk
INFORMIXSERVER=inf93
ORACLE_HOME=/usr/oracle9i
SYBASE=/sybase/V12
SYBASE_OCS=OCS-12_5
```

  다음 제한사항이 db2dj.ini 파일에 적용됩니다.
  - 항목은 *evname=value* 형식을 따라야 합니다. 여기서 *evname*은 환경 변수 이름이고 *value*는 해당 값입니다.
  - 환경 변수 이름의 최대 길이는 255바이트입니다.
  - 환경 변수 값의 최대 길이는 765바이트입니다.

  이 변수는 데이터베이스 관리 프로그램 매개변수 **federated**가 YES로 설정된 경우 외에는 무시됩니다.

DB2DMNBCKCTLR

- 운영 체제: Windows
- 디폴트: NULL, 값: ? 또는 도메인 이름
- DB2 서버가 백업 도메인 제어기인 도메인의 이름을 아는 경우, **DB2DMNBCKCTLR=***DOMAIN_NAME*을 설정하십시오. *DOMAIN_NAME*은 대문자여야 합니다. DB2가 로컬 머신이 백업 도메인 제어기인 도메인을 판별하도록 하려면 **DB2DMNBCKCTLR=?**를 설정하십시오. **DB2DMNBCKCTLR** 프로파일 변수가 설정되지 않거나 공백으로 설정된 경우, DB2는 기본 도메인 제어기에서 인증을 수행합니다.

  주: 백업 도메인 제어기는 기본 도메인 제어기와의 동기화에서 이탈할 수 있고 이는 보안 노출의 원인이 되므로 DB2는 디폴트로 기존의 백업 도메인 제어기를 사용하지 않습니다. 동기화 이탈은 기본 도메인 제어기의 보안 데이터베이스가 갱신되지만 변경사항이 백업 도메인 제어기에 전달되지 않는 경우에 발생할 수 있습니다. 이는 네트워크 대기 시간이 있거나 컴퓨터 브라우저 서비스가 작동하지 않는 경우에 발생할 수 있습니다.

주: DB2DMNBCKCTLR은 사용되지 않으며 추후 릴리스에서는 제거됩니다. Active Directory에는 더 이상 백업 도메인 제어기가 없으므로 이 변수는 더 이상 필요하지 않습니다.

DB2_DOCHOST

- 운영 체제: 모두
- 디폴트: 설정되지 않음(그러나 DB2는 계속 IBM 웹 사이트 publib.boulder.ibm.com/infocenter/db2luw/v9r7에서 정보 센터에 액세스를 시도함), 값: http://*hostname* 여기서 *hostname*= 유효한 호스트 이름 또는 IP 주소
- *DB2 정보 센터*가 설치된 호스트 이름을 지정합니다. 이 변수는 DB2 설치 마법사에서 자동 구성 옵션이 선택된 경우 *DB2 정보 센터* 설치 중에 자동으로 설정될 수 있습니다.

DB2_DOCPORT

- 운영 체제: 모두
- 디폴트: NULL, 값: 임의의 유효한 포트 번호
- DB2 도움말 시스템이 DB2 문서를 제공하는 포트 번호를 지정합니다. 이 변수는 DB2 설치 마법사에서 자동 구성 옵션이 선택된 경우 *DB2 정보 센터* 설치 중에 자동으로 설정될 수 있습니다.

DB2_ENABLE_AUTOCONFIG_DEFAULT

- 운영 체제: 모두
- 디폴트: NULL, 값: YES 또는 NO

- 이 변수는 데이터베이스 작성 시 구성 어드바이저가 자동으로 실행되는지 여부를 제어합니다. **DB2_ENABLE_AUTOCONFIG_DEFAULT**가 설정되지 않은 경우(NULL), 변수가 YES로 설정되고 데이터베이스 작성 시 구성 어드바이저가 실행되는 것과 효과가 동일합니다. 이 변수를 설정한 후 인스턴스를 재시작할 필요가 없습니다. AUTOCONFIGURE 명령을 실행하거나 CREATE DB AUTOCONFIGURE를 실행하는 경우, 해당 명령은 **DB2_ENABLE_AUTOCONFIG_DEFAULT**의 설정값을 겹쳐씁니다.

DB2_ENABLE_LDAP

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- LDAP(Lightweight Directory Access Protocol)이 사용되는지 여부를 지정합니다. LDAP은 디렉토리 서비스에 대한 액세스 메소드입니다.

DB2_EVMON_EVENT_LIST_SIZE

- 운영 체제: 모두
- 디폴트: 0(한계 없음), 값: KB/Kb/kb, MB/Mb/mb 또는 GB/Gb/gb로 지정된 값. 이 변수의 상한은 고정되어 있지 않지만 모니터 힙에서 사용 가능한 메모리 양에 의해 제한됩니다.
- 이 레지스트리 변수는 특정 이벤트 모니터에 기록되기 위해 대기 중인 큐에 대기될 수 있는 최대 바이트 수를 지정합니다. 이 한계에 일단 접근하면 이벤트 모니터 레코드 전송을 시도하는 에이전트는 *큐 크기*가 이 임계값 아래로 떨어질 때까지 대기합니다.

주: 활성 레코드가 모니터 힙에서 할당될 수 없는 경우, 해당 레코드는 삭제됩니다. 이러한 일이 발생하지 않게 하려면 **mon_heap_sz** 구성 매개변수를 AUTOMATIC으로 설정하십시오. **mon_heap_sz**를 특정 값으로 설정한 경우, **DB2_EVMON_EVENT_LIST_SIZE**가 더 작은 값으로 설정되었는지 확인하십시오. 그러나 모니터 힙은 또한 다른 모니터 요소를 트래킹하는 데도 사용되므로 이러한 조치가 활성 레코드가 삭제되지 않도록 보장할 수는 없습니다.

DB2_EVMON_STMT_FILTER

- 운영 체제: 모두
- 디폴트: 설정되지 않음; 값:
  - ALL: 모든 명령문 이벤트 모니터의 출력이 필터링됨을 표시합니다. 이 옵션은 독점입니다.
  - '*nameA nameB nameC*': 여기서 문자열의 각 이름은 레코드가 필터링될 이벤트 모니터의 이름을 나타냅니다. 둘 이상의 이름이 제공되는 경우, 각 이름은 단일 공백으로 구분되어야 합니다. 모든 입력 이름은 DB2에

서 대문자로 작성됩니다. 지정할 수 있는 이벤트 모니터의 최대수는 32입니다. 각 모니터 이름의 길이는 최대 18자입니다.

– '*nameA:op1,op2 nameB:op1,op2 nameC:op1*': 여기서 문자열의 각 이름은 레코드가 필터링될 이벤트 모니터의 이름을 나타냅니다. 각 옵션은 특정 SQL 조작에 맵핑되는 정수값을 나타냅니다(*op1*, *op2* 등). 정수값을 지정하여 사용자는 이벤트 모니터와 이에 적용되는 규칙을 판별할 수 있습니다.

- **DB2_EVMON_STMT_FILTER**를 사용하여 명령문 이벤트 모니터에 의해 기록되는 레코드 수를 줄일 수 있습니다. 이 레지스트리 변수가 설정되면 다음 SQL 조작에 대한 레코드만 지정된 이벤트 모니터에 기록됩니다.

표 68. 이벤트 모니터 출력을 특정 SQL 조작으로 제한하기 위해 **DB2_EVMON_STMT_FILTER***에 사용하는 값*

SQL 조작	맵핑되는 정수값
SELECT	15
EXECUTE	2
EXECUTE_IMMEDIATE	3
CLOSE	6
STATIC COMMIT	8
STATIC ROLLBACK	9
CALL	12
PRE_EXEC	17

기타 모든 조작은 명령문 이벤트 모니터의 출력에 표시되지 않습니다. 레코드가 이벤트 모니터에 기록되는 조작 세트를 사용자 정의하려면 정수값을 사용하십시오.

예 1:

```
db2set DB2_EVMON_STMT_FILTER= 'mon1 monitor3'
```

이 예에서 mon1 및 monitor3 이벤트 모니터는 제한된 응용프로그램 요청 목록에 대한 레코드를 수신합니다. 예를 들어, mon1 명령문 이벤트 모니터에 의해 모니터되는 응용프로그램이 동적 SQL문을 준비하고 해당 명령문을 기반으로 커서를 열고 해당 커서에서 10,000행을 프리페치한 다음 커서 닫기 요청을 발행하면 닫기 요청에 대한 레코드만 mon1 이벤트 모니터 출력에 표시됩니다.

예 2:

```
db2set DB2_EVMON_STMT_FILTER='evmon1:3,8 evmon2:9,15
```

이 예에서 evmon1 및 evmon2는 제한된 응용프로그램 요청 목록에 대한 레코드를 수신합니다. 예를 들어, evmon1 명령문 이벤트 모니터에 의해 모니

터되는 응용프로그램이 작성 명령문을 발행하면 즉시 실행 및 정적 커미트 조작만 evmon1 이벤트 모니터 출력에 표시됩니다. evmon2 명령문 이벤트 모니터에 의해 모니터되는 응용프로그램이 선택 및 정적 롤백을 모두 포함하는 SQL을 수행하면 이 두 가지 조작만 evmon2 이벤트 모니터 출력에 표시됩니다.

주: 데이터베이스 시스템 모니터 상수의 정의에 대해서는 `sqlmon.h` 헤더 파일을 참조하십시오.

DB2_EXTSECURITY

- 운영 체제: Windows
- 디폴트: ON, 값: ON 또는 OFF
- DB2 시스템 파일 잠금을 사용하여 DB2에 대한 권한 부여되지 않은 액세스를 예방합니다. 잠재적인 문제점을 방지하려면 이 레지스트리 변수를 끄지 않아야 합니다.

DB2_FALLBACK

- 운영 체제: Windows
- 디폴트: OFF, 값: ON 또는 OFF
- 이 변수를 사용하여 폴백(fallback) 처리 중에 모든 데이터베이스 연결을 강제로 해제할 수 있습니다. 이 변수는 MCSC(Microsoft Cluster Server)가 있는 Windows 환경에서 장애 복구 지원과 함께 사용됩니다.
  DB2_FALLBACK이 설정되지 않거나 OFF로 설정되고 폴백 중에 데이터베이스 연결이 존재하는 경우, DB2 자원을 오프라인으로 가져올 수 없습니다. 이는 폴백 처리가 실패함을 의미합니다.

DB2_FMP_COMM_HEAPSZ

- 운영 체제: Windows, UNIX
- 디폴트: 20MB 또는 10개의 분리 루틴을 실행하기에 충분한 스페이스 (둘 중 큰 값). AIX의 경우 디폴트는 256MB입니다.
- 이 변수는 4KB 페이지에 분리 루틴 호출(예: 스토어드 프로시저 또는 사용자 정의 함수(UDF) 호출)에 사용되는 풀의 크기를 지정합니다. 각 분리 루틴에서 사용되는 스페이스는 **aslheapsz** 구성 매개변수 값의 두 배입니다.

  사용자 시스템에서 여러 개의 분리 루틴을 실행 중인 경우에는 이 변수의 값을 늘려야 합니다. 아주 적은 수의 분리 루틴을 실행 중인 경우에는 변수의 값을 줄일 수 있습니다.

  이 값을 0으로 설정하면 설정이 작성되지 않고 그 결과 분리 루틴을 호출할 수 없습니다. 이는 또한 Health Monitor와 자동 데이터베이스 유지보수 기

능(예: 자동 백업, 통계 콜렉션 및 REORG)이 사용 불가능하게 됨을 의미합니다. 이는 이 기능이 분리 루틴 인프라스트럭처에 종속되어 있기 때문입니다.

DB2_GRP_LOOKUP

- 운영 체제: Windows
- 디폴트: NULL, 값: LOCAL, DOMAIN, TOKEN, TOKENLOCAL, TOKENDOMAIN
- 이 변수는 사용자가 속한 그룹을 열거하는 데 사용되는 Windows 보안 메커니즘을 지정합니다.

DB2_HADR_BUF_SIZE

- 운영 체제: 모두
- 디폴트: 2***logbufsz**
- 이 변수는 로그 페이지 단위의 버퍼 크기를 수신하는 대기 로그를 지정합니다. 설정되지 않으면 DB2는 버퍼 크기를 수신하는 대기의 기본 **logbufsz** 구성 매개변수 값의 두 배를 사용합니다. 이 변수는 대기 인스턴스에 설정해야 합니다. 기본 데이터베이스에서는 무시됩니다.

  HADR 동기화 모드(**hadr_syncmode** 데이터베이스 구성 매개변수)가 ASYNC로 설정된 경우, 피어 상태 동안 느린 대기는 기본에서의 전송 조작을 스톨시키는 원인이 되며 따라서 기본에서의 트랜잭션 처리를 차단합니다. 대기 데이터베이스가 처리되지 않은 로그 데이터를 더 많이 보유할 수 있도록 대기 데이터베이스에서 디폴트 로그 수신 버퍼보다 큰 값을 구성할 수 있습니다. 이로 인해 트랜잭션 처리를 차단하지 않고 짧은 기간에 대기가 로그 데이터를 사용할 수 있는 속도보다 빠르게 기본이 로그 데이터를 생성할 수 있습니다.

DB2_HADR_NO_IP_CHECK

- 운영 체제: 모두
- 디폴트: OFF, 값: ON|OFF
- HADR 연결에 대한 IP 점검 생략 여부를 지정합니다.
- 이 변수는 기본적으로 HADR 연결에 대한 IP 교차 점검을 생략하기 위해 NAT(Network Address Translation) 환경에서 사용됩니다. 이 변수는 HADR 구성의 정상성 점검을 약화시키므로 다른 환경에서는 사용하지 않는 것이 좋습니다. 디폴트로 HADR 연결이 설정되면 로컬 및 리모트 호스트 매개변수의 구성 일관성이 검증됩니다. 호스트 이름은 교차 점검을 위해 IP 주소에 맵핑됩니다. 두 가지 점검이 수행됩니다.
  - 기본의 **HADR_LOCAL_HOST** 매개변수 = 대기의 **HADR_REMOTE_HOST** 매개변수

– 기본의 **HADR_REMOTE_HOST** 매개변수 = 대기의 **HADR_LOCAL_HOST** 매개변수

점검이 실패하면 연결이 닫힙니다.

이 매개변수가 켜져 있으면 IP 점검이 발생하지 않습니다.

DB2_HADR_PEER_WAIT_LIMIT

- 운영 체제: 모두
- 디폴트: **0**(한계 없음을 의미) 값: 0 - 부호 없는 최대 32비트 정수(두 값 포함)
- **DB2_HADR_PEER_WAIT_LIMIT** 레지스트리 변수가 설정되면 대기 데이터베이스에 대한 로그 복제 때문에 기본 데이터베이스의 로깅이 지정된 초수 동안 차단된 경우 HADR 기본 데이터베이스가 피어 상태를 중단합니다. 이 한계에 접근하면 기본 데이터베이스가 대기 데이터베이스에 대한 연결을 끊습니다. 피어 창이 사용 불가능하면 기본 데이터베이스는 연결이 끊긴 상태에 들어가고 로깅이 다시 시작됩니다. 피어 창이 사용 가능하면 기본 데이터베이스는 연결이 끊긴 피어 상태에 들어가고 로깅은 계속 차단됩니다. 재연결 또는 피어 창이 만기되자마자 기본 데이터베이스는 연결이 끊긴 피어 상태를 벗어납니다. 기본 데이터베이스가 연결이 끊긴 피어 상태를 벗어나면 로깅이 다시 시작됩니다. 이 매개변수는 대기 데이터베이스에서는 적용되지 않습니다. 그렇지만 기본 및 대기 데이터베이스 모두에서 같은 값을 사용하는 것이 좋습니다. 유효하지 않은 값(숫자가 아니거나 음수)은 한계가 없음을 뜻하는 ″0″으로 해석됩니다. 이 매개변수는 정적입니다. 이 매개변수를 적용하려면 데이터베이스 인스턴스를 재시작해야 합니다.

DB2_HADR_SORCVBUF

- 운영 체제: 모두
- 디폴트: 운영 체제 TCP 소켓 수신 버퍼 *크기*, 값: 1,024 - 4,294,967,295
- 이 변수는 HADR 연결에 대한 운영 체제(OS) TCP 소켓 수신 버퍼 *크기*를 지정하며 이를 사용하여 사용자는 다른 연결과 구분되게 HADR TCP/IP 동작을 사용자 정의할 수 있습니다. 일부 운영 체제에서는 사용자 지정 값을 자동으로 반올림하거나 상한을 지정합니다. HADR 연결에 사용된 실제 버퍼 *크기*는 db2diag 로그 파일에 로그됩니다. 사용자의 네트워크 트래픽을 기본으로 하는 이 매개변수에 대한 최적의 설정값은 운영 체제 네트워크 조정 안내서를 참조하십시오. 이 변수는 **DB2_HADR_SOSNDBUF**와 함께 사용해야 합니다.

DB2_HADR_SOSNDBUF

- 운영 체제: 모두
- 디폴트: 운영 체제 TCP 소켓 보내기 버퍼 *크기*, 값: 1,024 - 4,294,967,295

- 이 변수는 HADR 연결에 대한 운영 체제(OS) TCP 소켓 보내기 버퍼 크기를 지정하며 이를 사용하여 사용자는 다른 연결과 구분되게 the HADR TCP/IP 동작을 사용자 정의할 수 있습니다. 일부 운영 체제에서는 사용자 지정 값을 자동으로 반올림하거나 상한을 지정합니다. HADR 연결에 사용된 실제 버퍼 크기는 db2diag 로그 파일에 로그됩니다. 사용자의 네트워크 트래픽을 기본으로 하는 이 매개변수에 대한 최적의 설정값은 운영 체제 네트워크 조정 안내서를 참조하십시오. 이 변수는 **DB2_HADR_SORCVBUF**와 함께 사용해야 합니다.

DB2LDAP_BASEDN

- 운영 체제: 모두
- 디폴트: NULL, 값: 임의의 유효한 기본 식별 도메인 이름.
- 이 변수가 설정되면 DB2의 LDAP 오브젝트가 LDAP 디렉토리의 다음 아래에 저장됩니다.

```
CN=System
CN=IBM
CN=DB2
```

  이는 지정된 기본 식별 이름 아래에 있습니다.

  이 변수가 Microsoft Active Directory Server로 설정되며 CN=DB2, CN=IBM 및 CN=System이 이 식별 이름 아래에 정의되어 있는지 확인하십시오.

DB2LDAPCACHE

- 운영 체제: 모두
- 디폴트: YES, 값: YES 또는 NO
- LDAP 캐시가 사용 가능한지 지정합니다. 이 캐시는 로컬 머신에서 데이터베이스, 노드 및 DCS 디렉토리를 카탈로그하는 데 사용됩니다.

  캐시에 최신 항목이 들어 있는지 확인하려면 다음을 수행하십시오.

```
REFRESH LDAP IMMEDIATE ALL
```

  이 명령은 데이터베이스 디렉토리 및 노드 디렉토리에서 올바르지 않은 항목을 갱신하고 제거합니다.

DB2LDAP_CLIENT_PROVIDER

- 운영 체제: Windows
- 디폴트: NULL(사용 가능한 경우 Microsoft가 사용되고 그렇지 않은 경우 IBM이 사용됩니다.) 값: IBM 또는 Microsoft

- Windows 환경에서 실행될 때 DB2는 LDAP 디렉토리에 액세스하는 데 Microsoft LDAP 클라이언트 또는 IBM LDAP 클라이언트 사용을 지원합니다. 이 레지스트리 변수는 DB2에서 사용할 LFAP 클라이언트를 명시적으로 선택하는 데 사용됩니다.

  주: 이 레지스트리 변수의 현재 값을 표시하려면 db2set 명령을 사용하십시오.

  ```
  db2set DB2LDAP_CLIENT_PROVIDER
  ```

DB2LDAPHOST

- 운영 체제: 모두
- 디폴트: NULL, 값: 임의의 유효한 호스트 이름
- LDAP 디렉토리 위치의 호스트 이름을 지정합니다.

DB2LDAP_KEEP_CONNECTION

- 운영 체제: 모두
- 디폴트: YES, 값: YES 또는 NO
- DB2가 해당 내부 LDAP 연결 핸들을 캐시하는지 여부를 지정합니다. 이 변수가 NO로 설정되면 DB2가 Directory Server에 대한 해당 LDAP 연결 핸들을 캐시하지 않습니다. 그 결과 부정적인 성능 영향을 미칠 수 있지만 Directory Server에 대해 동시에 사용 중인 LDAP 클라이언트 연결 수를 최소화해야 하는 경우 **DB2LDAP_KEEP_CONNECTION**을 NO로 설정하는 것이 좋습니다.

  성능을 최대화하기 위해 이 변수는 디폴트로 YES로 설정됩니다.

  DB2LDAP_KEEP_CONNECTION 레지스트리 변수는 LDAP에서 전역 레벨 프로파일 레지스트리 변수로만 구현되므로 다음과 같이 db2set 명령에 **-gl** 옵션을 지정하여 이 변수를 설정해야 합니다.

  ```
  db2set -gl DB2LDAP_KEEP_CONNECTION=NO
  ```

DB2LDAP_SEARCH_SCOPE

- 운영 체제: 모두
- 디폴트: DOMAIN, 값: LOCAL, DOMAIN 또는 GLOBAL
- LDAP(Lightweight Directory Access Protocol)의 도메인 또는 데이터베이스 파티션에서 찾을 수 있는 정보의 검색 범위를 지정합니다. LOCAL은 LDAP 디렉토리 검색을 사용 불가능하게 합니다. DOMAIN은 LDAP에서 현재 디렉토리 파티션만 검색합니다. GLOBAL은 오브젝트를 찾을 때까지 LDAP의 모든 디렉토리 파티션을 검색합니다.

DB2_LOAD_COPY_NO_OVERRIDE

- 운영 체제: 모두

- 디폴트: NONRECOVERABLE, 값: COPY YES 또는 NONRECOVERABLE
- 이 변수는 변수의 값에 따라 모든 LOAD **COPY NO**를 LOAD **COPY YES** 또는 **NONRECOVERABLE**로 변환합니다. 이 변수는 HADR 기본 데이터베이스 및 표준(비HADR) 데이터베이스에 적용 가능합니다. 이 변수는 HADR 대기 데이터베이스에서는 무시됩니다. HADR 기본 데이터베이스에서 이 변수가 설정되지 않으면 LOAD **COPY NO**는 LOAD **NONRECOVERABLE**로 변환됩니다. 이 변수의 값은 **COPY YES**절과 동일한 구문을 사용하여 복사 대상을 지정하거나 복구 불가능한 로드를 지정합니다.

DB2LOADREC

- 운영 체제: 모두
- 디폴트: NULL
- 롤 포워드 중에 로드 사본의 위치를 겹쳐쓰는 데 사용됩니다. 사용자가 로드 사본의 물리적 위치를 변경한 경우, 롤 포워드를 실행하기 전에 **DB2LOADREC**를 설정해야 합니다.

DB2LOCK_TO_RB

- 운영 체제: 모두
- 디폴트: NULL, 값: STATEMENT
- 잠금 시간종료로 인해 전체 트랜잭션이 롤백되는지 또는 현재 명령문만 롤백되는지를 지정합니다. **DB2LOCK_TO_RB**가 STATEMENT로 설정된 경우, 잠금 시간종료는 현재 명령문만 롤백시킵니다. 기타 모든 설정은 트랜잭션 롤백을 일으킵니다.

DB2_MAP_XML_AS_CLOB_FOR_DLC

- 운영 체제: 모두
- 디폴트: NO, 값: YES 또는 NO
- **DB2_MAP_XML_AS_CLOB_FOR_DLC** 레지스트리 변수는 XML을 데이터 유형으로 지원하지 않는 클라이언트(또는 DRDA 응용프로그램 요청자)에 대한 XML 값의 디폴트 DESCRIBE 및 FETCH 동작을 겹쳐쓰는 기능을 제공합니다. 디폴트값은 NO이며 이는 해당 클라이언트에 대해 XML 값의 DESCRIBE는 BLOB(2GB)를 리턴하고 XML 값의 FETCH는 UTF-8의 인코딩을 표시하는 XML 선언을 포함하는 BLOB에 대해 내재적으로 XML 순번을 매기게 됩니다.

  값이 YES로 설정되면 XML 값의 DESCRIBE는 CLOB(2GB)를 리턴하고 XML 값의 FETCH는 XML 선언을 포함하지 않는 CLOB에 대해 내재적으로 XML 순번을 매기게 됩니다.

주: DB2_MAP_XML_AS_CLOB_FOR_DLC는 사용되지 않으며 추후 릴리스에서는 제거됩니다. XML 값에 액세스하는 대부분의 기존 DB2 응용프로그램은 XML 가능 클라이언트를 사용하여 이를 수행하므로 이 변수는 더 이상 필요하지 않습니다.

DB2_MAX_LOB_BLOCK_SIZE

- 운영 체제: 모두
- 디폴트: 0(한계 없음), 값: 0 - 21,487,483,647
- 블록에 리턴할 LOB 또는 XML 데이터의 최대 크기를 설정합니다. 이 값은 엄격한 최대값이 아닙니다. 데이터 검색 중에 서버에서 이 최대값에 접근하는 경우, 서버는 클라이언트에 대해 FETCH와 같은 명령에 대한 응답을 생성하기 전에 현재 행에서 쓰기를 완료합니다.

DB2_MEMORY_PROTECT

- 운영 체제: 스토리지 키 지원이 있는 AIX
- 디폴트: NO, 값: NO 또는 YES
- 이 레지스트리 변수는 유효하지 않은 메모리 액세스가 원인인 버퍼 풀의 데이터 손상을 방지하기 위해 스토리지 키를 사용하는 메모리 보호 기능을 사용 가능하게 합니다. 메모리 보호는 DB2 엔진 스레드가 버퍼 풀 메모리에 액세스할 수 있는 시간과 액세스할 수 없는 시간을 식별하여 작동합니다. **DB2_MEMORY_PROTECT**가 YES로 설정되면 DB2 엔진 스레드가 버퍼 풀 메모리에 올바르지 않게 액세스하려고 할 때마다 해당 엔진 스레드가 트랩됩니다.

  주: DB2_LGPAGE_BP가 YES로 설정된 경우 메모리 보호를 사용할 수 없습니다. **DB2_MEMORY_PROTECT**가 YES로 설정된 경우에도 DB2는 버퍼 풀 메모리를 보호하지 못하고 기능을 사용 불가능하게 합니다.

DB2NOEXITLIST

- 운영 체제: 모두
- 디폴트: OFF, 값: ON 또는 OFF
- 이 변수는 **DB2_COMMIT_ON_EXIT** 레지스트리 변수 설정과 관계없이 응용프로그램이 종료될 때 DB2가 종료 목록 핸들러를 로드하지 않고 커미트를 수행하지 않음을 표시합니다.

  DB2NOEXITLIST를 끄고 **DB2_COMMIT_ON_EXIT**를 켜면 Embedded SQL 응용프로그램에 대한 모든 인플라이트 트랜잭션이 자동으로 커미트됩니다. 응용프로그램이 종료될 때 COMMIT 또는 ROLLBACK문을 명시적으로 추가하는 것이 좋습니다.

응용프로그램이 종료되기 전에 DB2 라이브러리를 동적으로 로드 및 언로드하는 응용프로그램은 DB2 종료 핸들러를 호출할 때 손상됩니다. 이러한 손상은 응용프로그램이 메모리에 있지 않은 함수를 호출하려고 시도하기 때문에 발생합니다. 이러한 상황을 방지하려면 **DB2NOEXITLIST** 레지스트리 변수를 설정하십시오.

DB2_NUM_CKPW_DAEMONS

- 운영 체제: UNIX
- 디폴트: 3, 값: 1[:FORK] - 100[:FORK]
- **DB2_NUM_CKPW_DAEMONS** 레지스트리 변수를 사용하여 구성 가능한 개수의 암호 점검 디먼을 시작할 수 있습니다. 디먼은 db2start 중에 작성되며 디폴트 IBMOSauthserver 보안 플러그인이 사용 중일 때 암호 점검 요청을 처리합니다. **DB2_NUM_CKPW_DAEMONS**의 설정값을 늘리면 데이터베이스 연결을 설정하는 데 필요한 시간이 줄어들지만 이는 동시에 여러 개의 연결이 수행되고 인증 비용이 많이 드는 시나리오에서만 효과적입니다.

  DB2_NUM_CKPW_DAEMONS는 1 - 100 사이의 값으로 설정할 수 있습니다. 데이터베이스 관리 프로그램은 **DB2_NUM_CKPW_DAEMONS**로 지정한 디먼 수를 작성합니다. 각 디먼은 암호 점검 요청을 직접 처리할 수 있습니다.

  선택적 FORK 매개변수를 추가하여 암호 점검 요청을 처리하는 외부 암호 점검 프로그램(db2ckpw)을 명시적으로 생성하기 위해 암호 점검 디먼을 사용 가능하게 할 수 있습니다. 이는 이전 릴리스에서 **DB2_NUM_CKPW_DAEMONS**를 0으로 설정한 것과 유사합니다. FORK 모드에서 각 암호 점검 디먼은 암호를 점검하려는 각 요청에 대해 암호 점검 프로그램을 생성합니다. FORK 모드의 디먼은 인스턴스 소유자로 시작됩니다.

  DB2_NUM_CKPW_DAEMONS가 0으로 설정된 경우, 유효 값은 3:FORK로 설정되며 이 경우 3개의 암호 점검 디먼이 FORK 모드에서 시작됩니다.

DB2_OPTSTATS_LOG

- 운영 체제: 모두
- 디폴트: 설정되지 않음(아래 세부사항 참조), 값: OFF, ON {NUM | SIZE | NAME | DIR}
- **DB2_OPTSTATS_LOG**는 통계 콜렉션 관련 활동을 모니터하고 분석하는 데 사용되는 통계 이벤트 로깅 파일의 속성을 지정합니다. **DB2_OPTSTATS_LOG**가 설정되지 않거나 ON으로 설정된 경우, 통계 이벤트 로깅이 사용 가능하여 시스템 성능을 모니터하고 더 나은 문제점 판별

을 위해 실행 기록을 보존할 수 있습니다. 로그 레코드는 첫 번째 로그 파일이 가득 찰 때까지 해당 파일에 기록됩니다. 후속 레코드는 다음 사용 가능한 로그 파일에 기록됩니다. 파일의 최대수에 접근하면 새 레코드가 가장 오래된 로그 파일을 겹쳐씁니다. 시스템 자원 사용이 걱정되는 경우, 이 레지스트리 변수를 OFF로 설정하여 사용 불가능하게 하십시오.

통계 이벤트 로깅이 명시적으로 사용 가능한 경우(ON으로 설정), 수정할 수 있는 여러 가지 옵션이 있습니다.

- NUM: 회전 로그 파일의 최대수. 디폴트: 5, 값 1 - 15
- SIZE: 회전 로그 파일의 최대 *크기*. (각 회전 파일의 *크기*는 SIZE/NUM입니다.) 디폴트: 100Mb, 값 1Mb – 4096Mb
- NAME: 회전 로그 파일의 기본 이름. 디폴트: db2optstats.*number*.log, 예: db2optstats.0.log, db2optstats.1.log 등.
- DIR: 회전 로그 파일의 기본 디렉토리. 디폴트: **diagpath**/events

이러한 옵션에 원하는 만큼 값을 지정할 수 있으며 통계 로깅을 사용 가능하게 하려는 경우 ON이 첫 번째 값인지 확인하십시오. 예를 들어, 최대 6개의 로그 파일, 25Mb의 최대 파일 *크기*, 기본 파일 이름 mystatslog 및 디렉토리 mystats를 사용하여 통계 로깅을 사용 가능하게 하려면 다음 명령을 발행하십시오.

```
db2set DB2_OPTSTATS_LOG=ON,NUM=6,SIZE=25,NAME=mystatslog,DIR=mystats
```

DB2REMOTEPREG

- 운영 체제: Windows
- 디폴트: NULL, 값: 임의의 유효한 Windows 머신 이름
- DB2 인스턴스 및 DB2 인스턴스 프로파일의 Win32 레지스트리 목록을 포함하는 리모트 머신 이름을 지정합니다. **DB2REMOTEPREG**의 값은 DB2가 설치된 후 한 번만 설정되어야 하며 수정되지 않아야 합니다. 매우 주의하여 이 변수를 사용하십시오.

DB2_RESOLVE_CALL_CONFLICT

- 운영 체제: AIX, HP-UX, Solaris, Linux, Windows
- 디폴트: YES, 값: YES, NO
- 트리거 컨텍스트에서 SQLCODE -746 오류를 제거합니다. 트리거에서 CALL 문을 발행할 때 SQLCODE SQL0746이 런타임 시 발행될 수 있습니다. SQL0746 오류는 트리거에 의해 호출된 프로시저가 호출 명령문의 컨텍스트 내에서 이전에 수정된 테이블에 액세스하지 못하게 합니다. 이 변수가 설정되면 DB2 데이터베이스 관리 프로그램은 CALL문을 실행하기 전에 테이블에 대한 모든 수정이 트리거에 대한 SQL 표준 규칙에 따라 완료되도록 강제 실행합니다.

DB2_RESOLVE_CALL_CONFLICT를 재설정하기 전에 인스턴스를 중지한 다음 재시작하십시오. 그런 다음 트리거 호출의 원인이 되는 패키지를 리바인드하십시오. SQL 프로시저를 리바인드하려면 다음을 사용하십시오.
CALL SYSPROC.REBIND_ROUTINE_PACKAGE
('P','*procedureschema.procedurename*','CONSERVATIVE');

DB2_RESOLVE_CALL_CONFLICT가 성능에 영향을 미칠 수 있음을 알아야 합니다. **DB2_RESOLVE_CALL_CONFLICT**를 YES로 설정하면 DB2 데이터베이스 관리 프로그램은 필요에 따라 임시 테이블의 삽입을 통해 모든 잠재적인 읽기 및 쓰기 충돌을 해결하게 됩니다. 일반적으로 최대 하나의 임시 테이블이 삽입되므로 영향은 작습니다. 트리거링 명령문은 하나의(또는 소수의) 행만 트리거링 명령문에 의해 수정되므로 OLTP 환경에서 이 효과는 작습니다. 일반적으로 임시 테이블 스페이스에 SMS(system managed space)를 사용하는 일반적인 권장사항을 따르는 경우 **DB2_RESOLVE_CALL_CONFLICT** 설정의 성능 영향은 낮을 것으로 예측됩니다.

DB2ROUTINE_DEBUG

- 운영 체제: AIX 및 Windows
- 디폴트: OFF, 값: ON 또는 OFF
- Java 스토어드 프로시저에 대한 디버그 기능을 사용하는지 여부를 지정합니다. Java 스토어드 프로시저를 디버그하지 않으려면 디폴트인 OFF를 사용하십시오. 디버깅을 사용 가능하게 하면 성능 영향이 있습니다.

주: DB2ROUTINE_DEBUG는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 이 스토어드 프로시저 디버거는 통합 디버거로 교체되었습니다.

DB2SATELLITEID

- 운영 체제: 모두
- 디폴트: NULL, 값: Satellite 제어 데이터베이스에서 선언된 유효한 Satellite ID
- Satellite이 동기화할 때 Satellite 제어 서버에 전달되는 Satellite ID를 지정합니다. 이 변수에 값이 지정되지 않으면 로그온 ID가 Satellite ID로 사용됩니다.

DB2_SERVER_CONTIMEOUT

- 운영 체제: 모두
- 디폴트: 180, 값: 0 - 32,767초
- 이 레지스트리 변수와 **DB2_DISPATCHER_PEEKTIMEOUT** 레지스트리 변수는 모두 연결 시간 동안 새 클라이언트의 처리를 구성합니다. **DB2_SERVER_CONTIMEOUT**을 사용하여 연결을 종료하기 전에 에이

전트가 클라이언트의 연결 요청을 대기하는 시간(초)을 조정할 수 있습니다. 대부분의 경우에는 이 레지스트리 변수를 조정할 필요가 없지만 DB2 클라이언트가 연결 시간에 서버에 의해 지속적으로 시간종료되는 경우에는 **DB2_SERVER_CONTIMEOUT**에 높은 값을 설정하여 시간종료 기간을 연장할 수 있습니다. 유효하지 않은 값이 설정된 경우, 디폴트값이 사용됩니다. 이 레지스트리 변수는 동적이 아닙니다.

DB2_SERVER_ENCALG

- 운영 체제: 모두
- 디폴트: NULL, 값: AES_CMP 또는 AES_ONLY
-

  주: DB2_SERVER_ENCALG는 버전 9.7에서는 사용되지 않으며, 추후 릴리스에서는 제거될 수 있습니다.

  DB2 버전 9.7로 인스턴스를 업그레이드할 때 **DB2_SERVER_ENCALG** 레지스트리 변수가 설정된 경우, **alternate_auth_enc** 구성 매개변수는 **DB2_SERVER_ENCALG** 설정에 따라 AES_ONLY 또는 AES_CMP로 설정됩니다. 따라서 사용자 ID와 암호를 암호화하기 위한 암호화 알고리즘을 지정하려면 **alternate_auth_enc** 구성 매개변수를 갱신하십시오. **alternate_auth_enc** 구성 매개변수가 설정된 경우, 해당 값은 **DB2_SERVER_ENCALG** 레지스트리 변수값에 우선합니다.

DB2SORT

- 운영 체제: 모두, 서버 전용
- 디폴트: NULL
- 이 변수는 런타임 시 로드 유틸리티가 로드할 라이브러리의 위치를 지정합니다. 라이브러리에는 인덱싱 데이터 정렬에 사용되는 함수의 시작점이 포함되어 있습니다. 테이블 인덱스 작성에 로드 유틸리티와 함께 사용하기 위해 벤더 제공 정렬 제품을 이용하려면 **DB2SORT**를 사용하십시오. 제공된 경로는 데이터베이스 서버와 관련되어 있어야 합니다.

DB2_TRUNCATE_REUSESTORAGE

- 운영 체제: 모두
- 디폴트: NULL(설정되지 않음), 값: IMPORT, import
- 이 변수를 사용하여 **REPLACE**가 있는 IMPORT 명령과 BACKUP ... ONLINE 명령 간의 잠금 경합을 해결할 수 있습니다. 일부 상황에서는 온라인 백업 및 절단 조작을 동시에 실행할 수 없습니다. 이 경우, **DB2_TRUNCATE_REUSESTORAGE**를 IMPORT 또는 import로 설정할 수 있으며 데이터, 인덱스, Long 필드, 대형 오브젝트 및 블록 맵(다차원적으로 클러스터된(MDC) 테이블)을 포함하는 오브젝트의 실제 절단은 건너

뛰고 논리적 절단만 수행됩니다. 즉, **REPLACE**가 있는 IMPORT 명령은 테이블을 비워 오브젝트의 논리적 크기가 줄어들게 하지만 디스크의 스토리지는 할당된 채로 유지됩니다.

이 레지스트리 변수는 동적입니다. 인스턴스를 중지 및 시작하지 않고 변수를 설정하거나 설정 해제할 수 있습니다. 온라인 백업이 시작되기 전에 **DB2_TRUNCATE_REUSESTORAGE**를 설정한 다음 온라인 백업이 완료된 후 이를 설정 해제할 수 있습니다. 다중 파티션된 환경의 경우, 레지스트리 변수는 변수가 설정된 노드에서만 활성화됩니다.
DB2_TRUNCATE_REUSESTORAGE는 DMS 영구 오브젝트에서만 유효합니다.

SAP 환경에서 **DB2_WORKLOAD**=SAP가 설정되면 이 레지스트리 변수의 디폴트값은 IMPORT입니다.

DB2_USE_DB2JCCT2_JROUTINE

- 운영 체제: 모두
- 디폴트: 설정되지 않음, 값: ON/YES/1/TRUE 또는 OFF/NO/0/FALSE
- Java 스토어드 프로시저 및 사용자 정의 함수의 디폴트 드라이버는 IBM Data Server Driver for JDBC and SQLJ입니다. 사용되지 않는 드라이버인 Linux, UNIX 및 Windows용 DB2 JDBC 유형 2 드라이버를 사용하여 Java 루틴에 대한 SQL 요청을 제공하려면 **DB2_USE_DB2JCCT2_JROUTINE**을 OFF, NO, 0 또는 FALSE 중 하나로 설정하십시오.

DB2_UTIL_MSGPATH

- 운영 체제: 모두
- 디폴트: *instanceName*/tmp 디렉토리
- **DB2_UTIL_MSGPATH** 레지스트리 변수는 SYSPROC.ADMIN_CMD 프로시저, SYSPROC.ADMIN_REMOVE_MSGS 프로시저 및 SYSPROC.ADMIN_GET_MSGS UDF와 함께 사용됩니다. 이 변수는 인스턴스 레벨에서 적용됩니다. **DB2_UTIL_MSGPATH**를 사용하여 분리 사용자 ID가 파일을 읽고 쓰고 삭제할 수 있는 서버의 디렉토리 경로를 표시할 수 있습니다. 이 디렉토리는 모든 코디네이터 파티션에서 액세스 가능해야 하며 유틸리티 메시지 파일을 포함할 충분한 스페이스가 있어야 합니다.

  이 경로가 설정되지 않으면 *instanceName*/tmp 디렉토리가 디폴트로 사용됩니다(DB2가 설치 제거되면 *instanceName*/tmp는 제거됩니다).

  이 경로가 변경되면 이전 설정으로 지정된 디렉토리에 있는 파일은 자동으로 이동하거나 삭제되지 않습니다. 이전 경로 아래에서 작성된 메시지의 컨텐츠를 검색하려면 이러한 메시지(유틸리티 이름이 앞에 붙고 사용자 ID가 뒤에

붙음)를 **DB2_UTIL_MSGPATH**가 지정하는 새 디렉토리로 수동으로 이동해야 합니다. 다음 유틸리티 메시지 파일은 새 위치에서 작성되고 읽히고 제거됩니다.

DB2_UTIL_MSGPATH 디렉토리 아래에 있는 파일은 트랜잭션에 종속된 것이 아니라 유틸리티에 따라 다릅니다. 이러한 파일은 백업 이미지의 일부가 아닙니다. **DB2_UTIL_MSGPATH** 디렉토리 아래에 있는 파일은 사용자 관리됩니다. 이는 사용자가 SYSPROC.ADMIN_REMOVE_UTILMSG 프로시저를 사용하여 메시지 파일을 삭제할 수 있음을 의미합니다. 이러한 파일은 DB2 설치 제거로 제거되지 않습니다.

DB2_VENDOR_INI

- 운영 체제: AIX, HP-UX, Solaris 및 Windows
- 디폴트: NULL, 값: 임의의 유효한 경로 및 파일.
- 모든 벤더 특정 환경 설정을 포함하는 파일을 지정합니다. 값은 데이터베이스 관리 프로그램이 시작될 때 읽힙니다.

주: DB2_VENDOR_INI는 버전 9.5에서는 사용되지 않으며 추후 릴리스에서는 제거됩니다. 대신에 이 변수가 포함하는 환경 변수 설정을 **DB2_DJ_INI** 변수에서 지정한 파일에 저장할 수 있습니다.

DB2_XBSA_LIBRARY

- 운영 체제: AIX, HP-UX, Solaris 및 Windows
- 디폴트: NULL, 값: 임의의 유효한 경로 및 파일.
- 벤더 제공 XBSA 라이브러리를 지정합니다. AIX에서는 공유 오브젝트 이름이 `shr.o`로 지정되지 않은 경우, 설정에 공유 오브젝트가 포함되어야 합니다. HP-UX, Solaris 및 Windows에서는 공유 오브젝트 이름이 필요하지 않습니다. 예를 들어, Legato의 DB2용 NetWorker 비즈니스 제품 모듈을 사용하려면 레지스트리 변수를 다음과 같이 설정해야 합니다.

  ```
  db2set DB2_XSBA_LIBRARY="/usr/lib/libxdb2.a(bsashr10.o)"
  ```

  XBSA 인터페이스는 BACKUP DATABASE 또는 RESTORE DATABASE 명령을 통해 호출할 수 있습니다. 예를 들어, 다음과 같습니다.

  ```
  db2 backup db sample use XBSA
  db2 restore db sample use XBSA
  ```

제 22 장 구성 매개변수

DB2 데이터베이스 인스턴스 또는 데이터베이스를 작성할 때 해당 구성 파일이 디폴트 매개변수 값으로 작성됩니다. 이러한 매개변수 값을 수정하여 인스턴스 또는 데이터베이스의 성능 및 기타 특성을 향상시킬 수 있습니다.

매개변수의 디폴트값에 따라 데이터베이스 관리 프로그램이 할당하는 디스크 스페이스 및 메모리로 사용자의 필요를 충족시키기에 충분합니다. 그러나 디폴트값으로는 최대 성능을 달성하지 못하는 상황도 있을 수 있습니다.

*구성 파일*에는 DB2 데이터베이스 제품 및 개별 데이터베이스에 할당된 자원 및 진단 레벨과 같은 값을 정의하는 매개변수가 들어 있습니다. 두 가지 유형의 구성 파일이 있습니다.

- 각 DB2 인스턴스의 데이터베이스 관리 프로그램 구성 파일
- 각 개별 데이터베이스의 데이터베이스 구성 파일

DB2 데이터베이스 작성 시 *데이터베이스 관리 프로그램 구성 파일*이 작성됩니다. 이 파일에 있는 매개변수는 인스턴스 레벨에서 시스템 자원에 영향을 줍니다(해당 인스턴스의 일부분인 한 데이터베이스와 관계없음). 시스템의 구성에 따라 이러한 여러 매개변수의 값을 시스템 디폴트값에서 변경하여 성능을 향상시키거나 용량을 증가시킬 수 있습니다.

각 클라이언트 설치마다 하나의 데이터베이스 관리 프로그램 구성 파일도 있습니다. 이 파일에는 특정 워크스테이션의 클라이언트 인에이블러에 대한 정보가 있습니다. 서버에 사용 가능한 매개변수 서브세트가 클라이언트에 적용됩니다.

데이터베이스 관리 프로그램 구성 매개변수는 `db2systm` 파일에 저장됩니다. 이 파일은 데이터베이스 관리 프로그램의 인스턴스 작성 시 작성됩니다. Linux 및 UNIX 환경에서 데이터베이스 관리 프로그램의 인스턴스에 대한 `sqllib` 서브디렉토리에 이 파일이 있습니다. Windows에서는 이 파일의 디폴트 위치가 Windows 계열 운영 체제의 버전에 따라 다릅니다. Windows에서 디폴트 디렉토리 위치를 확인하려면 DB2SET DB2INSTPROF 명령을 사용하여 DB2INSTPROF 레지스트리 변수 설정을 점검하십시오. DB2INSTPROF 레지스트리 변수를 변경하여 디폴트 인스턴스 디렉토리를 변경할 수도 있습니다. DB2INSTPROF 변수가 설정된 경우 파일은 DB2INSTPROF 변수를 통해 지정된 디렉토리의 서브디렉토리 `instance`에 있습니다.

런타임 데이터 파일의 저장 위치를 지정하는 기타 프로파일 레지스트리 변수는 DB2INSTPROF의 값을 쿼리합니다. 여기에는 다음의 변수가 있습니다.

- DB2CLINIPATH

- DIAGPATH
- SPM_LOG_PATH

버전 8.2 이전에 작성된 데이터베이스의 경우 데이터베이스 구성 매개변수는 SQLDBCON 파일에 저장됩니다. 버전 8.2와 이후에 작성된 데이터베이스의 경우 모든 데이터베이스 구성 매개변수는 SQLDBCONF 파일에 저장됩니다. 이러한 파일은 직접 편집할 수 없으며 제공된 API를 사용하거나 해당 API를 호출하는 도구를 사용해야만 변경하거나 볼 수 있습니다.

파티션된 데이터베이스 환경에서 이 파일은 공유 파일 시스템에 상주하므로 모든 데이터베이스 파티션 서버가 동일한 파일에 액세스할 수 있습니다. 데이터베이스 관리 프로그램의 구성은 모든 데이터베이스 파티션 서버에서 동일합니다.

대부분의 매개변수는 데이터베이스 관리 프로그램의 단일 인스턴스에 할당되는 시스템 자원 양에 영향을 주거나 환경적 고려사항에 따라 데이터베이스 관리 프로그램 및 여러 가지 통신 서브시스템의 설정을 구성합니다. 또한 정보용으로만 제공되며 변경할 수 없는 기타 매개변수가 있습니다. 이러한 모든 매개변수에는 해당 데이터베이스 관리 프로그램 인스턴스에 저장된 단일 데이터베이스와 관계없이 전역으로 적용됩니다.

*데이터베이스 구성 파일*은 데이터베이스 작성 시 작성되며 데이터베이스가 상주하는 위치에 상주합니다. 데이터베이스마다 하나의 구성 파일이 있습니다. 해당 매개변수는 무엇보다 해당 데이터베이스에 할당할 자원의 양을 지정합니다. 여러 매개변수의 값을 변경하여 성능을 향상시키거나 용량을 증가시킬 수 있습니다. 특정 데이터베이스에서 활동 유형에 따라 필요한 변경이 다를 수 있습니다.

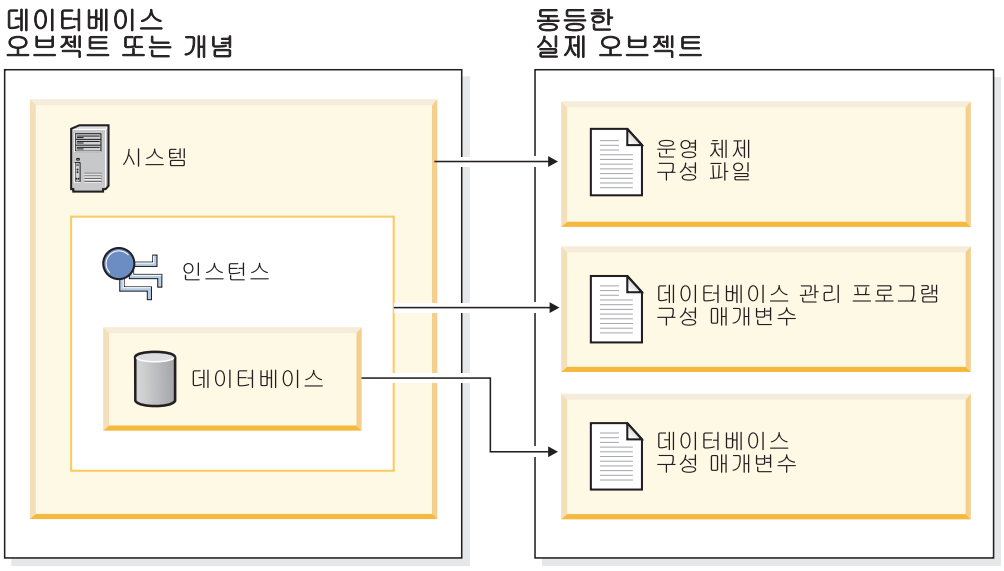


그림 48. 데이터베이스 오브젝트와 구성 파일의 관계

구성 매개변수를 사용하여 DB2 데이터베이스 관리 프로그램 구성

매개변수의 디폴트값에 따라 데이터베이스 관리 프로그램이 할당하는 디스크 스페이스 및 메모리로 사용자의 필요를 충족시키기에 충분합니다. 그러나 디폴트값으로는 최대 성능을 달성하지 못하는 상황도 있을 수 있습니다.

디폴트값은 상대적으로 작은 메모리 자원이 있고 데이터베이스 서버 전용인 머신을 지향하므로 환경에 다음이 있는 경우 이 값을 수정해야 합니다.

- 대형 데이터베이스
- 많은 연결
- 특정 응용프로그램을 위한 높은 성능 요구사항
- 고유 쿼리 또는 트랜잭션 로드 또는 유형

각 트랜잭션 처리 환경은 하나 이상의 측면에서 고유합니다. 이러한 차이는 디폴트 구성을 사용하는 경우 데이터베이스 관리 프로그램의 성능에 충분히 영향을 미칠 수 있습니다. 따라서 환경에 맞게 구성을 조정할 것을 적극 권장합니다.

구성을 시작할 때는 워크로드 특성에 대해 질문하고 사용자의 응답에 따라 매개변수 값을 생성하는 구성 어드바이저 또는 AUTOCONFIGURE 명령을 사용하는 것이 좋습니다.

일부 구성 매개변수는 AUTOMATIC으로 설정되어, 데이터베이스 관리 프로그램이 자동으로 현재 자원 요구사항을 반영하도록 이 매개변수를 관리할 수 있습니다. 현재 내부 설정을 유지하면서 구성 매개변수의 AUTOMATIC 설정을 끄려면 UPDATE DATABASE CONFIGURATION 명령과 함께 MANUAL 키워드를 사용하십시오. 데이터베이스 관리 프로그램이 이 매개변수의 값을 갱신하는 경우, get db/dbm cfg show detail 명령은 새 값을 표시합니다.

개별 데이터베이스의 매개변수는 SQLDBCONF라는 구성 파일에 저장됩니다. 이 파일은 데이터베이스의 다른 제어 파일과 함께 SQLnnnnn 디렉토리에 저장됩니다. 여기서 nnnnn은 데이터베이스를 작성할 때 지정한 번호입니다. 각 데이터베이스에는 자신의 구성 파일이 있으며 파일에서 대부분의 매개변수는 해당 데이터베이스에 할당된 자원의 양을 지정합니다. 이 파일에는 설명 정보와 데이터베이스의 상태를 표시하는 플래그도 있습니다.

주의: 데이터베이스 관리 프로그램이 제공한 것이 아닌 방법을 사용하여 db2systm, SQLDBCON 또는 SQLDBCONF를 편집하는 경우 데이터베이스를 사용할 수 없게 될 수도 있습니다. 데이터베이스 관리 프로그램이 문서화하고 지원하는 방법이 아닌 방법을 사용하여 이러한 파일을 변경하지 마십시오.

파티션된 데이터베이스 환경에서는 각 데이터베이스 파티션마다 분리된 SQLDBCONF 파일이 있습니다. SQLDBCONF 파일에 있는 값은 각 데이터베이스 파티션에서 같을 수도

있고 다를 수도 있으나 권장사항은 동일한 환경에 대한 것이므로 구성 매개변수 값은 모든 데이터베이스 파티션에서 동일해야 합니다. 일반적으로 여러 데이터베이스 구성 매개변수 설정이 필요한 카탈로그 노드가 있을 수 있으며, 다른 데이터 파티션에는 머신 유형 및 기타 정보에 따라 다시 여러 값이 있을 수 있습니다.

명령행 처리기를 사용하여 구성 매개변수 갱신:

설정을 변경하는 명령은 다음과 같이 입력할 수 있습니다.

데이터베이스 관리 프로그램 구성 매개변수의 경우:

- GET DATABASE MANAGER CONFIGURATION(또는 GET DBM CFG)
- UPDATE DATABASE MANAGER CONFIGURATION(또는 UPDATE DBM CFG)
- 모든 데이터베이스 관리 프로그램 매개변수를 디폴트값으로 재설정하는 RESET DATABASE MANAGER CONFIGURATION(또는 RESET DBM CFG)
- AUTOCONFIGURE.

데이터베이스 구성 매개변수의 경우:

- GET DATABASE CONFIGURATION(또는 GET DB CFG)
- UPDATE DATABASE CONFIGURATION(또는 UPDATE DB CFG)
- 모든 데이터베이스 매개변수를 디폴트값으로 재설정하는 RESET DATABASE CONFIGURATION(또는 RESET DB CFG)
- AUTOCONFIGURE.

API를 사용하여 구성 매개변수 갱신

API는 응용프로그램 또는 호스트 언어 프로그램에서 호출할 수 있습니다. 구성 매개변수를 보거나 갱신하려면 다음 DB2 API를 호출하십시오.

- db2AutoConfig - 구성 어드바이저에 액세스
- db2CfgGet - 데이터베이스 관리 프로그램 또는 데이터베이스 구성 매개변수 가져오기
- db2CfgSet - 데이터베이스 관리 프로그램 또는 데이터베이스 구성 매개변수 설정

일반 SQL API 프로시저를 사용하여 구성 매개변수 갱신

SQL 기반 응용프로그램, DB2 명령행 또는 명령 스크립트에서 일반 SQL API 프로시저를 호출할 수 있습니다. 구성 매개변수를 보거나 갱신하려면 다음 프로시저를 호출하십시오.

- GET_CONFIG - 데이터베이스 관리 프로그램 또는 데이터베이스 구성 매개변수 가져오기

- SET_CONFIG - 데이터베이스 관리 프로그램 또는 데이터베이스 구성 매개변수 설정

구성 지원 프로그램을 사용하여 구성 매개변수 갱신

구성 지원 프로그램을 사용하여 클라이언트의 데이터베이스 관리 프로그램 구성 매개변수를 설정할 수도 있습니다. 다른 매개변수는 온라인으로 변경할 수 있습니다. 이러한 매개변수를 *온라인으로 구성 가능한 구성 매개변수*라고 합니다.

갱신된 구성 값 보기

일부 데이터베이스 관리 프로그램 구성 매개변수의 경우 새 매개변수 값을 적용하려면 데이터베이스 관리 프로그램을 중지(db2stop)한 후 재시작(db2start)해야 합니다.

일부 데이터베이스 매개변수의 경우 데이터베이스를 다시 활성화하거나 오프라인에서 온라인으로 전환해야만 변경사항이 적용됩니다. 이 경우 모든 응용프로그램은 우선 데이터베이스에서 연결을 끊어야 합니다. (데이터베이스가 활성화되었거나 오프라인에서 온라인으로 전환된 경우에는 비활성화한 후 다시 활성화해야 합니다.) 그런 다음 데이터베이스에 새로 연결할 때 변경사항이 적용됩니다.

인스턴스에 접속한 상태에서 온라인으로 구성 가능한 데이터베이스 관리 프로그램 구성 매개변수의 설정을 변경하는 경우, UPDATE DBM CFG 명령의 디폴트 동작은 변경사항을 즉시 적용합니다. 변경사항을 즉시 적용하지 않으려면 UPDATE DBM CFG 명령에 DEFERRED 옵션을 사용하십시오.

데이터베이스 관리 프로그램 구성 매개변수를 온라인으로 변경하려면 다음을 수행하십시오.

```
db2 attach to <instance-name>
db2 update dbm cfg using <parameter-name> <value>
db2 detach
```

클라이언트의 경우 데이터베이스 관리 프로그램 구성 매개변수에 대한 변경사항은 다음 번에 클라이언트가 서버에 연결할 때 적용됩니다.

연결된 동안에 온라인으로 구성 가능한 데이터베이스 구성 매개변수를 변경하는 경우 디폴트 동작은 가능한 경우 온라인으로 변경사항을 적용하는 것입니다. 일부 매개변수 변경사항은 스페이스 할당과 관련된 오버헤드 때문에 적용되는 데 상당한 시간이 걸립니다. 명령행 처리기에서 구성 매개변수를 온라인으로 변경하려면 데이터베이스에 연결되어 있어야 합니다. 데이터베이스 구성 매개변수를 온라인으로 변경하려면 다음을 수행하십시오.

```
db2 connect to <dbname>
db2 update db cfg using <parameter-name> <parameter-value>
db2 connect reset
```

온라인으로 구성 가능한 각 구성 매개변수에는 연결된 *전파 클래스*가 있습니다. 전파 클래스는 구성 매개변수에 대한 변경사항이 적용될 것으로 예상할 수 있는 시기를 표시합니다. 세 가지 전파 클래스가 있습니다.

- **즉시**: 명령 또는 API를 호출하면 즉시 변경되는 매개변수입니다. 예를 들어, *diaglevel*의 전파 클래스는 즉시입니다.
- **명령문 경계**: 명령문 또는 명령문과 비슷한 경계에서 변경되는 매개변수입니다. 예를 들어, *sortheap* 값을 변경하는 경우 모든 새 요청은 새 값을 사용하기 시작합니다.
- **트랜잭션 경계**: 트랜잭션 경계에서 변경되는 매개변수입니다. 예를 들어, *dl_expint*의 새 값은 COMMIT 문 후에 갱신됩니다.

새 매개변수 값이 즉시 적용되지 않는 경우에도 GET DATABASE MANAGER CONFIGURATION 또는 GET DATABASE CONFIGURATION 명령을 사용하여 매개변수 설정을 보면 항상 최신 갱신사항이 표시됩니다. 이러한 명령에 SHOW DETAIL절을 사용하여 매개변수 설정을 보면 최신 갱신사항과 메모리의 값이 둘 다 표시됩니다.

데이터베이스 구성 매개변수를 갱신한 후 응용프로그램 리바인드

일부 데이터베이스 구성 매개변수를 변경하면 SQL 및 XQuery 옵티마이저가 선택한 액세스 플랜에 영향을 미칠 수 있습니다. 이러한 매개변수를 변경한 후에는 SQL 및 XQuery문에 최선의 액세스 플랜이 사용되도록 응용프로그램을 리바인드할 것을 고려해야 합니다. UPDATE DATABASE CONFIGURATION IMMEDIATE 명령을 사용하여 온라인으로 매개변수를 수정하면 SQL 및 XQuery 옵티마이저는 새 쿼리 명령문을 위해 새 액세스 플랜을 선택합니다. 그러나 쿼리 명령문 캐시는 기존 항목에서 제거되지 않습니다. 쿼리 캐시의 컨텐츠를 지우려면 FLUSH PACKAGE CACHE 문을 사용하십시오.

주: 많은 구성 매개변수(예: *userexit*)는 도움말 및 기타 DB2 문서에서 『Yes』나 『No』 또는 『On』이나 『Off』의 승인할 수 있는 값을 가지는 것으로 설명됩니다. 분명하게 하기 위해 『Yes』는 『On』과 같은 것으로 간주되고 『No』는 『Off』와 같은 것으로 간주되어야 합니다.

구성 매개변수 요약

다음 표는 데이터베이스 서버의 *데이터베이스 관리 프로그램* 및 *데이터베이스* 구성 파일에서 사용되는 매개변수를 나열합니다. 데이터베이스 관리 프로그램 및 데이터베이스 구성 매개변수를 변경하는 경우, 각 매개변수에 대한 자세한 정보를 고려하십시오. 디폴트를 포함한 특정 운영 환경 정보도 매개변수 설명에 포함됩니다.

데이터베이스 관리 프로그램 구성 매개변수 요약

일부 데이터베이스 관리 프로그램 구성 매개변수의 경우 새 매개변수 값을 적용하려면 데이터베이스 관리 프로그램을 중지(db2stop)한 후 재시작(db2start)해야 합니다. 다른 매개변수는 온라인으로 변경할 수 있습니다. 이러한 매개변수를 *온라인으로 구성 가능한 구성 매개변수*라고 합니다. 인스턴스에 접속한 상태에서 온라인으로 구성 가능한 데이터베이스 관리 프로그램 구성 매개변수의 설정을 변경하는 경우, UPDATE DBM CFG 명령의 디폴트 동작은 변경사항을 즉시 적용합니다. 변경사항을 즉시 적용하지 않으려면 UPDATE DBM CFG 명령에 **DEFERRED** 옵션을 사용하십시오.

다음 표의 『자동』 컬럼은 매개변수가 UPDATE DBM CFG 명령에 **AUTOMATIC** 키워드를 지원하는지 여부를 표시합니다.

매개변수를 자동으로 갱신하는 경우, 시작 값도 **AUTOMATIC** 키워드로 지정할 수 있습니다. 값은 매개변수에 따라 의미가 다를 수 있으며 적용 불가능한 경우도 있습니다. 값을 지정하기 전에 매개변수의 문서를 읽어서 의미를 판별하십시오. 다음 예에서 **num_poolagents**는 AUTOMATIC으로 갱신되며 DB2는 풀에 대한 유휴 에이전트의 최소수로 20을 사용합니다.

```
db2 update dbm cfg using num_poolagents 20 automatic
```

AUTOMATIC 기능을 해제하려면 매개변수 값을 갱신할 수 있거나 **MANUAL** 키워드를 사용할 수 있어야 합니다. 매개변수가 MANUAL로 갱신되면 그 매개변수는 더 이상 자동이 아니며, 현재 값(GET DBM CFG SHOW DETAIL 및 GET DB CFG SHOW DETAIL 명령에서 현재 값 컬럼에 표시되는)으로 설정됩니다.

『성능 영향』 컬럼은 시스템 성능과 관계된 각 매개변수의 상대적인 중요성을 표시합니다. 이 컬럼을 모든 환경에 정확하게 적용하는 것은 불가능합니다. 이 정보는 일반화로만 참조해야 합니다.

- **높음** — 매개변수가 성능에 상당한 영향을 미칠 수 있음을 표시합니다. 이 매개변수의 값은 주의해서 결정해야 합니다. 때때로 디폴트값을 승인하는 의미가 될 수 있습니다.
- **중간** — 매개변수가 성능에 어느 정도 영향을 미칠 수 있음을 표시합니다. 특정 환경 및 수요에 따라 이 매개변수에 얼마나 많은 조정 노력이 필요한지 결정됩니다.
- **낮음** — 매개변수가 덜 일반적이거나 성능에 별로 영향을 미치지 않음을 표시합니다.
- **없음** — 매개변수가 성능에 직접 영향을 미치지 않음을 표시합니다. 성능 향상을 위해 이 매개변수를 조정할 필요가 없는 경우에도 시스템 구성의 다른 면(예: 통신 지원)에서 매우 중요할 수 있습니다.

『토큰』, 『토큰 값』 및 『데이터 유형』 컬럼은 db2CfgGet 또는 db2CfgSet API를 호출할 때 필요합니다. 이 정보에는 구성 매개변수 ID, db2CfgParam 데이터 구조의 *token* 요소 항목 및 구조로 전달되는 값의 데이터 유형이 포함됩니다.

표 69. 구성 가능한 데이터베이스 관리 프로그램 구성 매개변수

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보
agent_stack_sz	아니오	아니오	낮음	SQLF_KTN_AGENT_STACK_SZ	61	Uint16	702 페이지의 『agent_stack_sz - 에이전트 스택 크기』
agentpri	아니오	아니오	높음	SQLF_KTN_AGENTPRI	26	Sint16	704 페이지의 『agentpri - 에이전트 우선순위』
alternate_auth_enc [6]	아니오	아니오	낮음	SQLF_KTN_ALTERNATE_AUTH_ENC	938	Uint16	706 페이지의 『alternate_auth_enc - 서버에서 수신 연결의 대체 암호화 알고리즘 구성 매개변수』
aslheapsz	아니오	아니오	높음	SQLF_KTN_ASLHEAPSZ	15	Uint32	707 페이지의 『aslheapsz - 응용프로그램 지원 계층 힙 크기』
audit_buf_sz	아니오	아니오	높음	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32	708 페이지의 『audit_buf_sz - 감사 버퍼 크기』
authentication[1]	아니오	아니오	낮음	SQLF_KTN_AUTHENTICATION	78	Uint16	709 페이지의 『authentication - 인증 유형』
catalog_noauth	예	아니오	없음	SQLF_KTN_CATALOG_NOAUTH	314	Uint16	712 페이지의 『catalog_noauth - 권한 없이 카탈로그화 가능』
clnt_krb_plugin	아니오	아니오	없음	SQLF_KTN_CLNT_KRB_PLUGIN	812	char(33)	712 페이지의 『clnt_krb_plugin - 클라이언트 Kerberos 플러그인』
clnt_pw_plugin	아니오	아니오	없음	SQLF_KTN_CLNT_PW_PLUGIN	811	char(33)	713 페이지의 『clnt_pw_plugin - 클라이언트 사용자 ID 암호 플러그인』
cluster_mgr	아니오	아니오	없음	SQLF_KTN_CLUSTER_MGR	920	char(262)	713 페이지의 『cluster_mgr - 클러스터 관리 프로그램 이름』
comm_bandwidth	예	아니오	중간	SQLF_KTN_COMM_BANDWIDTH	307	float	714 페이지의 『comm_bandwidth - 통신 대역폭』
conn_elapse	예	아니오	중간	SQLF_KTN_CONN_ELAPSE	508	Uint16	715 페이지의 『conn_elapse - 연결 경과 시간』
cpuspeed	예	아니오	높음	SQLF_KTN_CPUSPEED	42	float	715 페이지의 『cpuspeed - CPU 속도』
dft_account_str	예	아니오	없음	SQLF_KTN_DFT_ACCOUNT_STR	28	char(25)	718 페이지의 『dft_account_str - 디폴트 접미부 어카운트』
dft_monswitches • **dft_mon_bufpool** • **dft_mon_lock** • **dft_mon_sort** • **dft_mon_stmt** • **dft_mon_table** • **dft_mon_timestamp** • **dft_mon_uow**	예	아니오	중간	SQLF_KTN_DFT_MONSWITCHES[2] • SQLF_KTN_DFT_MON_BUFPOOL • SQLF_KTN_DFT_MON_LOCK • SQLF_KTN_DFT_MON_SORT • SQLF_KTN_DFT_MON_STMT • SQLF_KTN_DFT_MON_TABLE • SQLF_KTN_DFT_MON_TIMESTAMP • SQLF_KTN_DFT_MON_UOW	29 • 33 • 34 • 35 • 31 • 32 • 36 • 30	Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16	719 페이지의 『dft_monswitches - 디폴트 데이터베이스 시스템 모니터 스위치』
dftdbpath	예	아니오	없음	SQLF_KTN_DFTDBPATH	27	char(215)	720 페이지의 『dftdbpath - 디폴트 데이터베이스 경로』
diaglevel	예	아니오	낮음	SQLF_KTN_DIAGLEVEL	64	Uint16	721 페이지의 『diaglevel - 진단 오류 캡처 레벨』

표 69. 구성 가능한 데이터베이스 관리 프로그램 구성 매개변수 (계속)

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보
diagpath	예	아니오	없음	SQLF_KTN_DIAGPATH	65	char(215)	722 페이지의 『diagpath - 진단 데이터 디렉토리 경로』
dir_cache	아니오	아니오	중간	SQLF_KTN_DIR_CACHE	40	Uint16	723 페이지의 『dir_cache - 디렉토리 캐시 지원』
discover[3]	아니오	아니오	중간	SQLF_KTN_DISCOVER	304	Uint16	725 페이지의 『discover - 발견 모드』
discover_inst	예	아니오	낮음	SQLF_KTN_DISCOVER_INST	308	Uint16	726 페이지의 『discover_inst - 서버 인스턴스 발견』
fcm_num_buffers	예	예	중간	SQLF_KTN_FCM_NUM_ BUFFERS	503	Uint32	726 페이지의 『fcm_num_buffers - FCM 버퍼 수』
fcm_num_ channels	예	예	중간	SQLF_KTN_FCM_NUM_ CHANNELS	902	Uint32	727 페이지의 『fcm_num_channels - FCM 채널 수』
fed_noauth	예	아니오	없음	SQLF_KTN_FED_NOAUTH	806	Uint16	728 페이지의 『fed_noauth - 페더레이티드 인증 생략』
federated	예	아니오	중간	SQLF_KTN_FEDERATED	604	Sint16	729 페이지의 『federated - 페더레이티드 데이터베이스 시스템 지원』
federated_async	예	예	중간	SQLF_KTN_FEDERATED_ ASYNC	849	Sint32	729 페이지의 『federated_async - 쿼리당 최대 비동기 TQ 수 구성 매개변수』
fenced_pool	예	예	중간	SQLF_KTN_FENCED_POOL	80	Sint32	730 페이지의 『fenced_pool - 최대 분리(fenced) 프로세스 수』
group_plugin	아니오	아니오	없음	SQLF_KTN_GROUP_PLUGIN	810	char(33)	732 페이지의 『group_plugin - 그룹 플러그인』
health_mon	예	아니오	낮음	SQLF_KTN_HEALTH_MON	804	Uint16	732 페이지의 『health_mon - 상태 모니터링』
indexrec[4]	예	아니오	중간	SQLF_KTN_INDEXREC	20	Uint16	733 페이지의 『indexrec - 인덱스 재작성 시간』
instance_memory	예	예	중간	SQLF_KTN_INSTANCE_ MEMORY	803	Uint64	735 페이지의 『instance_memory - 인스턴스 메모리』
intra_parallel	아니오	아니오	높음	SQLF_KTN_INTRA_PARALLEL	306	Sint16	738 페이지의 『intra_parallel - 파티션 내 병렬 처리 사용』
java_heap_sz	아니오	아니오	높음	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32	739 페이지의 『java_heap_sz - 최대 Java 인터프리터 힙 크기』
jdk_path	아니오	아니오	없음	SQLF_KTN_JDK_PATH	311	char(255)	740 페이지의 『jdk_path - Java용 SDK(Software Developer's Kit) 설치 경로』
keepfenced	아니오	아니오	중간	SQLF_KTN_KEEPFENCED	81	Uint16	740 페이지의 『keepfenced - 분리(fenced) 프로세스 보존』
local_gssplugin	아니오	아니오	없음	SQLF_KTN_LOCAL_ GSSPLUGIN	816	char(33)	742 페이지의 『local_gssplugin - 로컬 인스턴스 레벨 권한 부여에 사용되는 GSS API 플러그인』
max_connections	예	예	중간	SQLF_KTN_MAX_ CONNECTIONS	802	Sint32	742 페이지의 『max_connections - 클라이언트 연결의 최대수』
max_connretries	예	아니오	중간	SQLF_KTN_MAX_ CONNRETRIES	509	Uint16	743 페이지의 『max_connretries - 노드 연결 재시도 수』
max_coordagents	예	예	중간	SQLF_KTN_MAX_ COORDAGENTS	501	Sint32	743 페이지의 『max_coordagents - 최대 코디네이팅 에이전트 수』
max_querydegree	예	아니오	높음	SQLF_KTN_MAX_ QUERYDEGREE	303	Sint32	744 페이지의 『max_querydegree - 병렬 처리의 최대 쿼리 수준』
max_time_diff	아니오	아니오	중간	SQLF_KTN_MAX_TIME_DIFF	510	Uint16	745 페이지의 『max_time_diff - 노드 간 최대 시간 차이』
mon_heap_sz	예	예	낮음	SQLF_KTN_MON_HEAP_SZ	79	Uint16	748 페이지의 『mon_heap_sz - 데이터베이스 시스템 모니터 힙 크기』

표 69. 구성 가능한 데이터베이스 관리 프로그램 구성 매개변수 (계속)

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보
notifylevel	예	아니오	낮음	SQLF_KTN_NOTIFYLEVEL	605	Sint16	750 페이지의 『notifylevel - 통지 레벨』
num_initagents	아니오	아니오	중간	SQLF_KTN_NUM_ INITAGENTS	500	Uint32	751 페이지의 『num_initagents - 풀의 초기 에이전트 수』
num_initfenced	아니오	아니오	중간	SQLF_KTN_NUM_ INITFENCED	601	Sint32	752 페이지의 『num_initfenced - 분리(fenced) 프로세스의 초기 수』
num_poolagents	예	예	높음	SQLF_KTN_NUM_ POOLAGENTS	502	Sint32	752 페이지의 『num_poolagents - 에이전트 풀 크기』
numdb	아니오	아니오	낮음	SQLF_KTN_NUMDB	6	Uint16	753 페이지의 『numdb - 호스트 및 System i 데이터베이스를 포함한 최대 동시 활성 데이터베이스 수』
query_heap_sz	아니오	아니오	중간	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32	754 페이지의 『query_heap_sz - 쿼리 힙 크기』
resync_interval	아니오	아니오	없음	SQLF_KTN_RESYNC_ INTERVAL	68	Uint16	756 페이지의 『resync_interval - 트랜잭션 재동기화 간격』
rqrioblk	아니오	아니오	높음	SQLF_KTN_RQRIOBLK	1	Uint16	756 페이지의 『rqrioblk - 클라이언트 I/O 블록 크기』
sheapthres	아니오	아니오	높음	SQLF_KTN_SHEAPTHRES	21	Uint32	758 페이지의 『sheapthres - 정렬 힙 임계값』
spm_log_file_sz	아니오	아니오	낮음	SQLF_KTN_SPM_LOG_FILE_ SZ	90	Sint32	760 페이지의 『spm_log_file_sz - 동기점 관리 프로그램 로그 파일 크기』
spm_log_path	아니오	아니오	중간	SQLF_KTN_SPM_LOG_PATH	313	char(226)	760 페이지의 『spm_log_path - 동기점 관리 프로그램 로그 파일 경로』
spm_max_resync	아니오	아니오	낮음	SQLF_KTN_SPM_MAX_ RESYNC	91	Sint32	761 페이지의 『spm_max_resync - 동기점 관리 프로그램 재동기 에이전트 한계』
spm_name	아니오	아니오	없음	SQLF_KTN_SPM_NAME	92	char(8)	761 페이지의 『spm_name - 동기점 관리 프로그램 이름』
srvcon_auth	아니오	아니오	없음	SQLF_KTN_SRVCON_AUTH	815	Uint16	762 페이지의 『srvcon_auth - 서버에서 수신 연결의 인증 유형』
srvcon_gssplugin_ list	아니오	아니오	없음	SQLF_KTN_SRVCON_ GSSPLUGIN_ LIST	814	char(256)	762 페이지의 『srvcon_gssplugin_list - 서버에서 수신 연결의 GSS API 플러그인 목록』
srv_plugin_mode	아니오	아니오	없음	SQLF_KTN_SRV_PLUGIN_ MODE	809	Uint16	764 페이지의 『srv_plugin_mode - 서버 플러그인 모드』
srvcon_pw_plugin	아니오	아니오	없음	SQLF_KTN_SRVCON_PW_ PLUGIN	813	char(33)	763 페이지의 『srvcon_pw_plugin - 서버에서 수신 연결의 사용자 ID 암호 플러그인』
ssl_svr_keydb	아니오	아니오	없음	SQLF_KTN_SSL_SVR_KEYDB	930	char(1023)	766 페이지의 『ssl_svr_keydb - 서버에서 수신 SSL 연결의 SSL 키 파일 경로 구성 매개변수』
ssl_svr_stash	아니오	아니오	없음	SQLF_KTN_SSL_SVR_STASH	931	char(1023)	767 페이지의 『ssl_svr_stash - 서버에서 수신 SSL 연결의 SSL 숨김 파일 경로 구성 매개변수』
ssl_svr_label	아니오	아니오	없음	SQLF_KTN_SSL_SVR_LABEL	932	char(1023)	767 페이지의 『ssl_svr_label - 서버에서 수신 SSL 연결의 키 파일에 있는 레이블 구성 매개변수』
ssl_svcename	아니오	아니오	없음	SQLF_KTN_SSL_SVCENAME	933	char(14)	769 페이지의 『ssl_svcename - SSL 서비스 이름 구성 매개변수』
ssl_cipherspecs	아니오	아니오	없음	SQLF_KTN_SSL_CIPHERSPECS	934	char(255)	764 페이지의 『ssl_cipherspecs - 서버에서 지원되는 암호 스펙 구성 매개변수』

표 69. 구성 가능한 데이터베이스 관리 프로그램 구성 매개변수 (계속)

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보
ssl_versions	아니오	아니오	없음	SQLF_KTN_SSL_VERSIONS	935	char(255)	770 페이지의 『ssl_versions - 서버에서 지원되는 SSL 버전 구성 매개변수』
ssl_clnt_keydb	아니오	아니오	없음	SQLF_KTN_SSL_CLNT_KEYDB	936	char(1023)	765 페이지의 『ssl_clnt_keydb - 클라이언트에서 아웃바운드 SSL 연결의 SSL 키 파일 경로 구성 매개변수』
ssl_clnt_stash	아니오	아니오	없음	SQLF_KTN_SSL_CLNT_STASH	937	char(1023)	765 페이지의 『ssl_clnt_stash - 클라이언트에서 아웃바운드 SSL 연결의 SSL 숨김 파일 경로 구성 매개변수』
start_stop_time	예	아니오	낮음	SQLF_KTN_START_STOP_ TIME	511	Uint16	768 페이지의 『start_stop_time - 시작 및 중지 시간종료』
svcename	아니오	아니오	없음	SQLF_KTN_SVCENAME	24	char(14)	771 페이지의 『svcename - TCP/IP 서비스 이름』
sysadm_group	아니오	아니오	없음	SQLF_KTN_SYSADM_ GROUP	39	char(128)	772 페이지의 『sysadm_group - 시스템 관리 권한 그룹 이름』
sysctrl_group	아니오	아니오	없음	SQLF_KTN_SYSCTRL_ GROUP	63	char(128)	773 페이지의 『sysctrl_group - 시스템 제어 권한 그룹 이름』
sysmaint_group	아니오	아니오	없음	SQLF_KTN_SYSMAINT_ GROUP	62	char(128)	774 페이지의 『sysmaint_group - 시스템 유지보수 권한 그룹 이름』
sysmon_group	아니오	아니오	없음	SQLF_KTN_SYSMON GROUP	808	char(128)	775 페이지의 『sysmon_group - 시스템 모니터 권한 그룹 이름』
tm_database	아니오	아니오	없음	SQLF_KTN_TM_DATABASE	67	char(8)	775 페이지의 『tm_database - 트랜잭션 관리 프로그램 데이터베이스 이름』
tp_mon_name	아니오	아니오	없음	SQLF_KTN_TP_MON_NAME	66	char(19)	776 페이지의 『tp_mon_name - 트랜잭션 프로세서 모니터 이름』
trust_allclnts[5]	아니오	아니오	없음	SQLF_KTN_TRUST_ALLCLNTS	301	Uint16	778 페이지의 『trust_allclnts - 모든 클라이언트 신뢰』
trust_clntauth	아니오	아니오	없음	SQLF_KTN_TRUST_ CLNTAUTH	302	Uint16	779 페이지의 『trust_clntauth - 트러스트된 클라이언트 인증』
util_impact_lim	예	아니오	높음	SQLF_KTN_UTIL_IMPACT_ LIM	807	Uint32	780 페이지의 『util_impact_lim - 인스턴스 영향 규정』

표 69. 구성 가능한 데이터베이스 관리 프로그램 구성 매개변수 (계속)

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보

주:

1. 유효한 값은 sqlenv.h에 정의되어 있습니다.

2.

```
Bit 1 (xxxx xxx1): dft_mon_uow
Bit 2 (xxxx xx1x): dft_mon_stmt
Bit 3 (xxxx x1xx): dft_mon_table
Bit 4 (xxxx 1xxx): dft_mon_buffpool
Bit 5 (xxx1 xxxx): dft_mon_lock
Bit 6 (xx1x xxxx): dft_mon_sort
Bit 7 (x1xx xxxx): dft_mon_timestamp
```

3. 유효한 값(sqlutil.h에 정의되어 있음):

```
SQLF_DSCVR_KNOWN (1)
SQLF_DSCVR_SEARCH (2)
```

4. 유효한 값(sqlutil.h에 정의되어 있음):

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
```

5. 유효한 값(sqlutil.h에 정의되어 있음):

```
SQLF_TRUST_ALLCLNTS_NO (0)
SQLF_TRUST_ALLCLNTS_YES (1)
SQLF_TRUST_ALLCLNTS_DRDAONLY (2)
```

6. 유효한 값(sqlenv.h에 정의되어 있음):

```
SQL_ALTERNATE_AUTH_ENC_AES (0)
SQL_ALTERNATE_AUTH_ENC_AES_CMP (1)
SQL_ALTERNATE_AUTH_ENC_NOTSPEC (255)
```

표 70. 정보용 데이터베이스 관리 프로그램 구성 매개변수

매개변수	토큰	토큰 값	데이터 유형	추가 정보
nodetype[1]	SQLF_KTN_NODETYPE	100	Uint16	749 페이지의 『nodetype - 머신 노드 유형』
release	SQLF_KTN_RELEASE	101	Uint16	755 페이지의 『release - 구성 파일 릴리스 레벨』

주:

1. 유효한 값(sqlutil.h에 정의되어 있음):

```
SQLF_NT_STANDALONE (0)
SQLF_NT_SERVER (1)
SQLF_NT_REQUESTOR (2)
SQLF_NT_STAND_REQ (3)
SQLF_NT_MPP (4)
SQLF_NT_SATELLITE (5)
```

데이터베이스 구성 매개변수 요약

다음 표는 데이터베이스 구성 파일의 매개변수를 나열합니다. 데이터베이스 구성 매개변수를 변경하는 경우, 매개변수에 대한 자세한 정보를 고려하십시오.

일부 데이터베이스 구성 매개변수의 경우 데이터베이스가 다시 활성화되어야만 변경사항이 적용됩니다. 이 경우 모든 응용프로그램은 우선 데이터베이스에서 연결을 끊어야 합니다. (데이터베이스가 활성화 상태인 경우에는 비활성화한 후 다시 활성화해야 합니

다.) 변경사항은 다음 번에 데이터베이스에 연결할 때 적용됩니다. 다른 매개변수는 온라인으로 변경할 수 있습니다. 이러한 매개변수를 *온라인으로 구성 가능한 구성 매개변수*라고 합니다.

『자동』, 『성능 영향』, 『토큰』, 『토큰 값』 및 『데이터 유형』 컬럼에 대한 설명은 위의 데이터베이스 관리 프로그램 구성 매개변수 요약 절을 참조하십시오.

AUTOMATIC 키워드는 UPDATE DB CFG 명령어에도 지원됩니다. 다음 예에서 **database_memory**는 AUTOMATIC으로 갱신되며 데이터베이스 관리 프로그램은 이 매개변수를 더 변경할 때 20000을 시작 값으로 사용합니다.

```
db2 update db cfg using for sample using database_memory 20000 automatic
```

버전 9.5에서 시작하는 경우, db2_all 명령을 실행하거나 각 파티션을 개별적으로 갱신 또는 재설정하지 않고 일부 또는 전체 플랫폼에서 데이터베이스 구성 매개변수 값을 갱신하거나 재설정할 수 있습니다. 자세한 내용은 다중 파티션에서 데이터베이스 구성을 참조하십시오.

표 71. 구성 가능한 데이터베이스 구성 매개변수

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보
alt_collate	아니오	아니오	없음	SQLF_DBTN_ALT_COLLATE	809	Uint32	781 페이지의 『alt_collate - 대체 조합 시퀀스』
applheapsz	예	예	중간	SQLF_DBTN_APPLHEAPSZ	51	Uint16	785 페이지의 『applheapsz - 응용프로그램 힙 크기』
appl_memory	예	예	중간	SQLF_DBTN_APPL_MEMORY	904	Uint64	784 페이지의 『appl_memory - 응용프로그램 메모리 구성 매개변수』
archretrydelay	예	아니오	없음	SQLF_DBTN_ ARCHRETRYDELAY	828	Uint16	786 페이지의 『archretrydelay - 오류 시 아카이브 재시도 대기 시간』
• **auto_maint** • **auto_db_backup** • **auto_tbl_maint** • **auto_runstats** • **auto_stats_prof** • **auto_stmt_stats** • **auto_prof_upd** • **auto_reorg**	예	아니오	중간	• SQLF_DBTN_AUTO_MAINT • SQLF_DBTN_AUTO_DB_ BACKUP • SQLF_DBTN_AUTO_TBL_ MAINT • SQLF_DBTN_AUTO_ RUNSTATS • SQLF_DBTN_AUTO_STATS_ PROF • SQLF_DBTN_AUTO_STMT_ STATS • SQLF_DBTN_AUTO_PROF_ UPD • SQLF_DBTN_AUTO_REORG	• 831 • 833 • 835 • 837 • 839 • 905 • 844 • 841	Uint16	787 페이지의 『auto_maint - 자동 유지보수』
auto_del_rec_obj	예	아니오	중간	SQLF_DBTN_AUTO_DEL_ REC_OBJ	912	Uint16	786 페이지의 『auto_del_rec_obj - 복구 오브젝트의 자동화된 삭제 구성 매개변수』
autorestart	예	아니오	낮음	SQLF_DBTN_AUTO_RESTART	25	Uint16	790 페이지의 『autorestart - 자동 재시작 사용』

표 71. 구성 가능한 데이터베이스 구성 매개변수 (계속)

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보
auto_reval 구성 매개변수	예	아니오	중간	SQLF_DBTN_AUTO_REVAL	920	Uint16	711 페이지의 『auto_reval - 자동 유효성 다시 확인 및 무효화 구성 매개변수』
avg_appls	예	예	높음	SQLF_DBTN_AVG_APPLS	47	Uint16	790 페이지의 『avg_appls - 활성 응용프로그램의 평균 수』
blk_log_dsk_ful	예	아니오	없음	SQLF_DBTN_BLK_LOG_DSK_FUL	804	Uint16	791 페이지의 『blk_log_dsk_ful - 디스크 가득참 로그 시 블록』
catalogcache_sz	예	아니오	중간	SQLF_DBTN_CATALOGCACHE_SZ	56	Uint32	793 페이지의 『catalogcache_sz - 카탈로그 캐시 크기』
chngpgs_thresh	아니오	아니오	높음	SQLF_DBTN_CHNGPGS_THRESH	38	Uint16	794 페이지의 『chngpgs_thresh - 변경된 페이지 임계값』
cur_commit	아니오	아니오	중간	SQLF_DBTN_CUR_COMMIT	917	Uint32	716 페이지의 『cur_commit - 현재 커미트됨 구성 매개변수』
database_memory	예	예	중간	SQLF_DBTN_DATABASE_MEMORY	803	Uint64	798 페이지의 『database_memory - 데이터베이스 공유 메모리 크기』
dbheap	예	예	중간	SQLF_DBTN_DB_HEAP	58	Uint64	800 페이지의 『dbheap - 데이터베이스 힙』
db_mem_thresh	예	아니오	낮음	SQLF_DBTN_DB_MEM_THRESH	849	Uint16	801 페이지의 『db_mem_thresh - 데이터베이스 메모리 임계값』
decflt_rounding	아니오	아니오	없음	SQLF_DBTN_DECFLT_ROUNDING	913	Uint16	803 페이지의 『decflt_rounding - 10진 부동 소수점 근사값 구성 매개변수』
dec_to_char_fmt	예	예	중간	SQLF_DBTN_DEC_TO_CHAR_FMT	• 0(v95) • 1(NEW)	Uint16	717 페이지의 『dec_to_char_fmt - 10진수를 문자로 함수 구성 매개변수』
dft_degree	예	아니오	높음	SQLF_DBTN_DFT_DEGREE	301	Sint32	805 페이지의 『dft_degree - 디폴트 등급』
dft_extent_sz	예	아니오	중간	SQLF_DBTN_DFT_EXTENT_SZ	54	Uint32	805 페이지의 『dft_extent_sz - 테이블 스페이스의 디폴트 Extent 크기』
dft_loadrec_ses	예	아니오	중간	SQLF_DBTN_DFT_LOADREC_SES	42	Sint16	806 페이지의 『dft_loadrec_ses - 디폴트 로그 복구 세션 수』
dft_mttb_types	아니오	아니오	없음	SQLF_DBTN_DFT_MTTB_TYPES	843	Uint32	807 페이지의 『dft_mttb_types - 최적화를 위해 유지되는 디폴트 테이블 유형』
dft_prefetch_sz	예	예	중간	SQLF_DBTN_DFT_PREFETCH_SZ	40	Sint16	807 페이지의 『dft_prefetch_sz - 디폴트 프리페치 크기』
dft_queryopt	예	아니오	중간	SQLF_DBTN_DFT_QUERYOPT	57	Sint32	808 페이지의 『dft_queryopt - 디폴트 쿼리 최적화 클래스』
dft_refresh_age	아니오	아니오	중간	SQLF_DBTN_DFT_REFRESH_AGE	702	char(22)	809 페이지의 『dft_refresh_age - 디폴트 새로 고침 유효 기간』
dft_sqlmathwarn	아니오	아니오	없음	SQLF_DBTN_DFT_SQLMATHWARN	309	Sint16	810 페이지의 『dft_sqlmathwarn - 산술 예외에 따라 계속』
discover_db	예	아니오	중간	SQLF_DBTN_DISCOVER	308	Uint16	813 페이지의 『discover_db - 데이터베이스 발견』
dlchktime	예	아니오	중간	SQLF_DBTN_DLCHKTIME	9	Uint32	813 페이지의 『dlchktime - 교착 상태를 점검하는 시간 간격』
dyn_query_mgmt	아니오	아니오	낮음	SQLF_DBTN_DYN_QUERY_MGMT	604	Uint16	814 페이지의 『dyn_query_mgmt - 동적 SQL 및 XQuery 쿼리 관리』
enable_xmlchar	예	아니오	없음	SQLF_DBTN_ENABLE_XMLCHAR	853	Uint32	815 페이지의 『enable_xmlchar - XML로 변환 사용 구성 매개변수』
failarchpath	예	아니오	없음	SQLF_DBTN_FAILARCHPATH	826	char(243)	816 페이지의 『failarchpath - 장애 복구 로그 아카이브 경로』
hadr_local_host	아니오	아니오	없음	SQLF_DBTN_HADR_LOCAL_HOST	811	char(256)	817 페이지의 『hadr_local_host - HADR 로컬 호스트 이름』

표 71. 구성 가능한 데이터베이스 구성 매개변수 (계속)

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보
hadr_local_svc	아니오	아니오	없음	SQLF_DBTN_HADR_LOCAL_ SVC	812	char(41)	818 페이지의 『hadr_local_svc - HADR 로컬 서비스 이름』
hadr_peer_ window	아니오	아니오	낮음(주 4 참조)	SQLF_DBTN_HADR_PEER_ WINDOW	914	Uint32	818 페이지의 『hadr_peer_window - HADR 피어 창 구성 매개변수』
hadr_remote_host	아니오	아니오	없음	SQLF_DBTN_HADR_REMOTE_ HOST	813	char(256)	819 페이지의 『hadr_remote_host - HADR 리모트 호스트 이름』
hadr_remote_inst	아니오	아니오	없음	SQLF_DBTN_HADR_REMOTE_ INST	815	char(9)	819 페이지의 『hadr_remote_inst - 리모트 서버의 HADR 인스턴스 이름』
hadr_remote_svc	아니오	아니오	없음	SQLF_DBTN_HADR_REMOTE_ SVC	814	char(41)	820 페이지의 『hadr_remote_svc - HADR 리모트 서비스 이름』
hadr_syncmode	아니오	아니오	없음	SQLF_DBTN_HADR_ SYNCMODE	817	Uint32	820 페이지의 『hadr_syncmode - 피어 상태에서 로그 쓰기용 HADR 동기화 모드』
hadr_timeout	아니오	아니오	없음	SQLF_DBTN_HADR_TIMEOUT	816	Uint32	821 페이지의 『hadr_timeout - HADR 시간종료 값』
indexrec[2]	예	아니오	중간	SQLF_DBTN_INDEXREC	30	Uint16	733 페이지의 『indexrec - 인덱스 재작성 시간』
locklist	예	예	에스컬레이션에 영향을 미치는 경우 높음	SQLF_DBTN_LOCK_LIST	704	Uint64	825 페이지의 『locklist - 잠금 목록의 최대 스토리지』
locktimeout	아니오	아니오	중간	SQLF_DBTN_LOCKTIMEOUT	34	Sint16	828 페이지의 『locktimeout - 잠금 시간종료』
logarchmeth1	예	아니오	없음	SQLF_DBTN_ LOGARCHMETH1	822	char(252)	829 페이지의 『logarchmeth1 - 1차 로그 아카이브 방법』
logarchmeth2	예	아니오	없음	SQLF_DBTN_ LOGARCHMETH2	823	char(252)	831 페이지의 『logarchmeth2 - 2차 로그 아카이브 방법』
logarchopt1	예	아니오	없음	SQLF_DBTN_LOGARCHOPT1	824	char(243)	832 페이지의 『logarchopt1 - 1차 로그 아카이브 옵션』
logarchopt2	예	아니오	없음	SQLF_DBTN_LOGARCHOPT2	825	char(243)	832 페이지의 『logarchopt2 - 2차 로그 아카이브 옵션』
logbufsz	아니오	아니오	높음	SQLF_DBTN_LOGBUFSZ	33	Uint16	833 페이지의 『logbufsz - 로그 버퍼 크기』
logfilsiz	아니오	아니오	중간	SQLF_DBTN_LOGFIL_SIZ	92	Uint32	834 페이지의 『logfilsiz - 로그 파일 크기』
logindexbuild	예	아니오	없음	SQLF_DBTN_ LOGINDEXBUILD	818	Uint32	835 페이지의 『logindexbuild - 작성된 로그 인덱스 페이지』
logprimary	아니오	아니오	중간	SQLF_DBTN_LOGPRIMARY	16	Uint16	836 페이지의 『logprimary - 1차 로그 파일 수』
logretain[3]	아니오	아니오	낮음	SQLF_DBTN_LOG_RETAIN	23	Uint16	838 페이지의 『logretain - 로그 유지 사용』
logsecond	예	아니오	중간	SQLF_DBTN_LOGSECOND	17	Uint16	839 페이지의 『logsecond - 2차 로그 파일 수』
max_log	예	예		SQLF_DBTN_MAX_LOG	807	Uint16	840 페이지의 『max_log - 트랜잭션당 최대 로그』
maxappls	예	예	중간	SQLF_DBTN_MAXAPPLS	6	Uint16	841 페이지의 『maxappls - 최대 활성 응용프로그램 수』
maxfilop	예	아니오	중간	SQLF_DBTN_MAXFILOP	3	Uint16	842 페이지의 『maxfilop - 응용프로그램당 열린 최대 데이터베이스 파일 수』
maxlocks	예	예	에스컬레이션에 영향을 미치는 경우 높음	SQLF_DBTN_MAXLOCKS	15	Uint16	843 페이지의 『maxlocks - 에스컬레이션 전 잠금 목록의 최대 퍼센트』

표 71. 구성 가능한 데이터베이스 구성 매개변수 (계속)

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보
min_dec_div_3	아니오	아니오	높음	SQLF_DBTN_MIN_DEC_DIV_3	605	Sint32	845 페이지의 『min_dec_div_3 - 10진수 나누기 스케일 3으로』
mincommit	예	아니오	높음	SQLF_DBTN_MINCOMMIT	32	Uint16	846 페이지의 『mincommit - 그룹화할 커미트 수』
mirrorlogpath	아니오	아니오	낮음	SQLF_DBTN_ MIRRORLOGPATH	806	char(242)	848 페이지의 『mirrorlogpath - 미러 로그 경로』
mon_act_metrics	예	아니오	중간	SQLF_DBTN_MON_ACT_METRICS	931	Uint16	849 페이지의 『mon_act_metrics - 활동 메트릭 모니터링 구성 매개변수』
mon_deadlock	예	아니오	중간	SQLF_DBTN_MON_DEADLOCK	934	Uint16	849 페이지의 『mon_deadlock - 교착 상태 모니터링 구성 매개변수』
mon_locktimeout	예	아니오	중간	SQLF_DBTN_MON_LOCKTIMEOUT	933	Uint16	850 페이지의 『mon_locktimeout - 잠금 시간종료 모니터링 구성 매개변수』
mon_lockwait	예	아니오	중간	SQLF_DBTN_MON_LOCKWAIT	935	Uint16	851 페이지의 『mon_lockwait - 잠김 대기 모니터링 구성 매개변수』
mon_lw_thresh	예	아니오	중간	SQLF_DBTN_MON_LW_THRESH	936	Uint32	852 페이지의 『mon_lw_thresh - 잠금 대기 임계값 모니터링 구성 매개변수』
mon_obj_metrics	예	아니오	중간	SQLF_DBTN_MON_OBJ_METRICS	937	Uint16	852 페이지의 『mon_obj_metrics - 오브젝트 메트릭 모니터링 구성 매개변수』
mon_req_metrics	예	아니오	중간	SQLF_DBTN_MON_REQ_METRICS	930	Uint16	853 페이지의 『mon_req_metrics - 요청 메트릭 모니터링 구성 매개변수』
mon_uow_data	예	아니오	중간	SQLF_DBTN_MON_UOW_DATA	932	Uint16	854 페이지의 『mon_uow_data - 작업 단위(UOW) 이벤트 모니터링 구성 매개변수』
newlogpath	아니오	아니오	낮음	SQLF_DBTN_NEWLOGPATH	20	char(242)	855 페이지의 『newlogpath - 데이터베이스 로그 경로 변경』
num_db_backups	예	아니오	없음	SQLF_DBTN_NUM_DB_ BACKUPS	601	Uint16	856 페이지의 『num_db_backups - 데이터베이스 백업 수』
num_freqvalues	예	아니오	낮음	SQLF_DBTN_NUM_ FREQVALUES	36	Uint16	857 페이지의 『num_freqvalues - 보유한 자주 사용되는 값 수』
num_iocleaners	아니오	예	높음	SQLF_DBTN_NUM_ IOCLEANERS	37	Uint16	858 페이지의 『num_iocleaners - 비동기 페이지 클리너 수』
num_ioservers	아니오	예	높음	SQLF_DBTN_NUM_ IOSERVERS	39	Uint16	860 페이지의 『num_ioservers - 입출력 서버 수』
num_log_span	예	예		SQLF_DBTN_NUM_LOG_ SPAN	808	Uint16	861 페이지의 『num_log_span - 로그 범위 수』
num_quantiles	예	아니오	낮음	SQLF_DBTN_NUM_ QUANTILES	48	Uint16	861 페이지의 『num_quantiles - 컬럼의 Quantile 수』
numarchretry	예	아니오	없음	SQLF_DBTN_ NUMARCHRETRY	827	Uint16	863 페이지의 『numarchretry - 오류 시 재시도 수』
overflowlogpath	아니오	아니오	중간	SQLF_DBTN_ OVERFLOWLOGPATH	805	char(242)	864 페이지의 『overflowlogpath - 오버플로우 로그 경로』
pckcachesz	예	예	높음	SQLF_DBTN_PCKCACHE_SZ	505	Uint32	866 페이지의 『pckcachesz - 패키지 캐시 *크기*』
rec_his_retentn	아니오	아니오	없음	SQLF_DBTN_REC_HIS_ RETENTN	43	Sint16	868 페이지의 『rec_his_retentn - 복구 실행기록 보존 기간』
self_tuning_mem	예	아니오	높음	SQLF_DBTN_SELF_TUNING_ MEM	848	Uint16	870 페이지의 『self_tuning_mem - 자체 성능 조정 메모리』
seqdetect	예	아니오	높음	SQLF_DBTN_SEQDETECT	41	Uint16	872 페이지의 『seqdetect - 순차적 발견 플래그』
sheapthres_shr	예	예	높음	SQLF_DBTN_SHEAPTHRES_ SHR	802	Uint32	873 페이지의 『sheapthres_shr - 공유 정렬에 대한 정렬 힙 임계값』

표 71. 구성 가능한 데이터베이스 구성 매개변수 (계속)

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보
softmax	아니오	아니오	중간	SQLF_DBTN_SOFTMAX	5	Uint16	874 페이지의 『softmax - 복구 범위 및 소프트 체크포인트 간격』
sortheap	예	예	높음	SQLF_DBTN_SORT_HEAP	52	Uint32	876 페이지의 『sortheap - 정렬 힙 크기』
stat_heap_sz	예	예	낮음	SQLF_DBTN_STAT_HEAP_SZ	45	Uint32	878 페이지의 『stat_heap_sz - 통계 힙 크기』
stmt_conc	예	아니오	중간	SQLF_DBTN_STMT_CONC	919	Uint32	770 페이지의 『stmt_conc - 명령문 집중기 구성 매개변수』
stmtheap	예	예	중간	SQLF_DBTN_STMT_HEAP	821	Uint32	878 페이지의 『stmtheap - 명령문 힙 크기』
trackmod	아니오	아니오	낮음	SQLF_DBTN_TRACKMOD	703	Uint16	879 페이지의 『trackmod - 수정된 페이지 추적 사용』
tsm_mgmtclass	예	아니오	없음	SQLF_DBTN_TSM_ MGMTCLASS	307	char(30)	880 페이지의 『tsm_mgmtclass - Tivoli Storage Manager 관리 클래스』
tsm_nodename	예	아니오	없음	SQLF_DBTN_TSM_ NODENAME	306	char(64)	880 페이지의 『tsm_nodename - Tivoli Storage Manager 노드 이름』
tsm_owner	예	아니오	없음	SQLF_DBTN_TSM_OWNER	305	char(64)	881 페이지의 『tsm_owner - Tivoli Storage Manager 소유자 이름』
tsm_password	예	아니오	없음	SQLF_DBTN_TSM_PASSWORD	501	char(64)	881 페이지의 『tsm_password - Tivoli Storage Manager 암호』
userexit	아니오	아니오	낮음	SQLF_DBTN_USER_EXIT	24	Uint16	882 페이지의 『userexit - User Exit 사용』
util_heap_sz	예	아니오	낮음	SQLF_DBTN_UTIL_HEAP_SZ	55	Uint32	883 페이지의 『util_heap_sz - 유틸리티 힙 크기』
vendoropt	예	아니오	없음	SQLF_DBTN_VENDOROPT	829	char(242)	884 페이지의 『vendoropt - 벤더 옵션』<
wlm_collect_int	예	아니오	낮음	SQLF_DBTN_WLM_COLLECT_ INT	907	Sint32	884 페이지의 『wlm_collect_int - 워크로드 관리 수집 간격 구성 매개변수』

표 71. 구성 가능한 데이터베이스 구성 매개변수 (계속)

매개변수	구성 가능한 온라인	자동	성능 영향	토큰	토큰 값	데이터 유형	추가 정보

주: SQLF_DBTN_AUTONOMIC_SWITCHES의 비트는 자동 유지보수 구성 매개변수 수의 디폴트 설정을 표시합니다. 이 복합 매개변수를 구성하는 개별 비트는 다음과 같습니다.

1.

```
Default => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint
Bit 2 off (xxxx xxxx xxxx xx0x): auto_db_backup
Bit 3 on  (xxxx xxxx xxxx x1xx): auto_tbl_maint
Bit 4 on  (xxxx xxxx xxxx 1xxx): auto_runstats
Bit 5 off (xxxx xxxx xxx0 xxxx): auto_stats_prof
Bit 6 off (xxxx xxxx xx0x xxxx): auto_prof_upd
Bit 7 off (xxxx xxxx x0xx xxxx): auto_reorg
Bit 8 off (xxxx xxxx 0xxx xxxx): auto_storage
Bit 9 off (xxxx xxx0 xxxx xxxx): auto_stmt_stats
0 0 0 D

Maximum => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint
Bit 2 off (xxxx xxxx xxxx xx1x): auto_db_backup
Bit 3 on  (xxxx xxxx xxxx x1xx): auto_tbl_maint
Bit 4 on  (xxxx xxxx xxxx 1xxx): auto_runstats
Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof
Bit 6 off (xxxx xxxx xx1x xxxx): auto_prof_upd
Bit 7 off (xxxx xxxx x1xx xxxx): auto_reorg
Bit 8 off (xxxx xxxx 1xxx xxxx): auto_storage
Bit 9 off (xxxx xxx1 xxxx xxxx): auto_stmt_stats
0 1 F F
```

2. 유효한 값(sqlutil.h에 정의되어 있음):

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
SQLF_INX_REC_RESTART (2)
```

3. 유효한 값(sqlutil.h에 정의되어 있음):

```
SQLF_LOGRETAIN_NO (0)
SQLF_LOGRETAIN_RECOVERY (1)
SQLF_LOGRETAIN_CAPTURE (2)
```

4. **hadr_peer_window** 매개변수를 0이 아닌 시간 값으로 설정하면 연결이 끊어진 피어 상태인 경우 기본 데이터베이스는 대기 데이터베이스의 확인을 기다리는 중이므로 대기 데이터베이스에 연결되어 있어도 트랜잭션에서 정지한 것처럼 보일 수 있습니다.

표 72. 정보용 데이터베이스 구성 매개변수

매개변수	토큰	토큰 값	데이터 유형	추가 정보
backup_pending	SQLF_DBTN_BACKUP_PENDING	112	Uint16	791 페이지의 『backup_pending - 백업 보류 표시기』
codepage	SQLF_DBTN_CODEPAGE	101	Uint16	795 페이지의 『codepage - 데이터베이스의 코드 페이지』
codeset	SQLF_DBTN_CODESET	120	char(9)[1]	795 페이지의 『codeset - 데이터베이스의 코드 세트』
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)	796 페이지의 『collate_info - 조합 정보』
country/region	SQLF_DBTN_COUNTRY	100	Uint16	797 페이지의 『country/region - 데이터베이스 지역 코드』
database_consistent	SQLF_DBTN_CONSISTENT	111	Uint16	797 페이지의 『database_consistent - 데이터베이스 일관성』
database_level	SQLF_DBTN_DATABASE_LEVEL	124	Uint16	797 페이지의 『database_level - 데이터베이스 릴리스 레벨』
hadr_db_role	SQLF_DBTN_HADR_DB_ROLE	810	Uint32	817 페이지의 『hadr_db_role - HADR 데이터베이스 역할』
log_retain_status	SQLF_DBTN_LOG_RETAIN_STATUS	114	Uint16	829 페이지의 『log_retain_status - 로그 유지 상태 표시기』
loghead	SQLF_DBTN_LOGHEAD	105	char(12)	835 페이지의 『loghead - 처음에 사용되는 로그 파일』

표 72. 정보용 데이터베이스 구성 매개변수 (계속)

매개변수	토큰	토큰 값	데이터 유형	추가 정보
logpath	SQLF_DBTN_LOGPATH	103	char(242)	836 페이지의 『logpath - 로그 파일의 위치』
multipage_alloc	SQLF_DBTN_MULTIPAGE_ALLOC	506	Uint16	854 페이지의 『multipage_alloc - 다중 페이지 파일 할당 사용』
numsegs	SQLF_DBTN_NUMSEGS	122	Uint16	863 페이지의 『numsegs - 디폴트 SMS 컨테이너 수』
pagesize	SQLF_DBTN_PAGESIZE	846	Uint32	865 페이지의 『pagesize - 데이터베이스 디폴트 페이지 크기』
release	SQLF_DBTN_RELEASE	102	Uint16	755 페이지의 『release - 구성 파일 릴리스 레벨』
restore_pending	SQLF_DBTN_RESTORE_PENDING	503	Uint16	869 페이지의 『restore_pending - 리스토어 보류』
restrict_access	SQLF_DBTN_RESTRICT_ACCESS	852	Sint32	869 페이지의 『restrict_access - 데이터베이스 액세스 제한 구성 매개변수』
rollfwd_pending	SQLF_DBTN_ROLLFWD_PENDING	113	Uint16	870 페이지의 『rollfwd_pending - 롤포워드 보류 표시기』
territory	SQLF_DBTN_TERRITORY	121	char(5)[2]	879 페이지의 『territory - 데이터베이스 지역』
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16	882 페이지의 『user_exit_status - User Exit 상태 표시기』

주:

1. HP-UX, Linux 및 Solaris 운영 체제에서 char(17)
2. HP-UX, Linux 및 Solaris 운영 체제에서 char(33)

DB2 Administration Server(DAS) 구성 매개변수 요약

표 73. DAS 구성 매개변수

매개변수	매개변수 유형	추가 정보
authentication	구성 가능	885 페이지의 『authentication - 인증 유형 DAS』
contact_host	온라인으로 구성 가능	886 페이지의 『contact_host - 문의처 목록의 위치』
das_codepage	온라인으로 구성 가능	886 페이지의 『das_codepage - DAS 코드 페이지』
das_territory	온라인으로 구성 가능	887 페이지의 『das_territory - DAS 지역』
dasadm_group	구성 가능	887 페이지의 『dasadm_group - DAS 관리 권한 그룹 이름』
db2system	온라인으로 구성 가능	888 페이지의 『db2system - DB2 서버 시스템 이름』
discover	온라인으로 구성 가능	889 페이지의 『discover - DAS 발견 모드』
exec_exp_task	구성 가능	889 페이지의 『exec_exp_task - 만기된 태스크 실행』
jdk_64_path	온라인으로 구성 가능	824 페이지의 『jdk_64_path - Java용 64비트 SDK(Software Developer's Kit) 설치 경로 DAS』
jdk_path	온라인으로 구성 가능	890 페이지의 『jdk_path - Java용 SDK(Software Developer's Kit) 설치 경로 DAS』
sched_enable	구성 가능	890 페이지의 『sched_enable - 스케줄러 모드』
sched_userid	정보용	891 페이지의 『sched_userid - 스케줄러 사용자 ID』
smtp_server	온라인으로 구성 가능	891 페이지의 『smtp_server - SMTP 서버』
toolscat_db	구성 가능	892 페이지의 『toolscat_db - 도구 카탈로그 데이터베이스』
toolscat_inst	구성 가능	892 페이지의 『toolscat_inst - 도구 카탈로그 데이터베이스 인스턴스』
toolscat_schema	구성 가능	893 페이지의 『toolscat_schema - 도구 카탈로그 데이터베이스 스키마』

구성 매개변수 섹션 표제

각 구성 매개변수 설명에는 다음 섹션 표제 중 적용 가능한 일부 또는 전부가 포함됩니다. 일부 경우 상호 독점적입니다. 예를 들어, [범위]가 지정된 경우에는 유효한 값이 필요하지 않습니다. 대부분의 경우 이러한 표제 자체가 설명적입니다.

표 74.

섹션 표제	설명 및 가능한 값
구성 유형	가능한 값은 다음과 같습니다. • 데이터베이스 관리 프로그램 • 데이터베이스 • DB2 Administration Server
적용 대상	적용 가능한 경우 구성 매개변수를 적용할 데이터 서버 유형을 나열합니다. 가능한 값은 다음과 같습니다. • 클라이언트 • 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버 • 로컬 클라이언트가 있는 데이터베이스 서버 • DB2 Administration Server • OLAP 기능 • 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버 • 페더레이션이 사용되는 경우 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버 • 로컬 클라이언트가 있는 Satellite 데이터베이스 서버
매개변수 유형	가능한 값은 다음과 같습니다. • 구성 가능(변경사항을 적용하려면 데이터베이스 관리 프로그램을 재시작해야 함) • 온라인으로 구성 가능(데이터베이스 관리 프로그램을 재시작하지 않고 온라인으로 동적으로 갱신할 수 있음) • 정보용(값은 정보 전용이며 갱신할 필요 없음)
디폴트 [범위]	적용 가능한 경우 디폴트값 및 가능한 범위(널(NULL) 값 또는 자동 설정 포함)를 나열합니다. 플랫폼에 따라 범위가 다른 경우, 값은 플랫폼 또는 플랫폼 유형별(예: 32비트 또는 64비트 플랫폼)로 나열됩니다. 대부분의 경우 디폴트값은 범위의 파트로 나열되지 않습니다.
수치 단위	적용 가능한 경우, 수치 단위를 나열합니다. 가능한 값: • 바이트 • 카운터 • 초 당 MB • 밀리초 • 분 • 페이지(4KB) • 퍼센트 • 초
유효한 값	적용 가능한 경우, 유효한 값을 나열합니다. 이 표제는 디폴트 [범위] 표제와 상호 독점적입니다.
예	적용 가능한 경우, 예를 나열합니다.
전파 클래스	적용 가능한 경우, 가능한 값은 다음과 같습니다. • 즉시 • 명령문 경계
할당 시기	적용 가능한 경우, 데이터베이스 관리 프로그램이 구성 매개변수를 할당하는 시기를 표시합니다.
사용 가능한 시기	적용 가능한 경우, 데이터베이스 관리 프로그램이 구성 매개변수를 해제하는 시기를 표시합니다.
제한사항	적용 가능한 경우, 구성 매개변수에 적용되는 모든 제한사항을 나열합니다.
한계	적용 가능한 경우, 구성 매개변수에 적용되는 모든 한계를 나열합니다.
권장사항	적용 가능한 경우, 구성 매개변수에 적용되는 모든 권장사항을 나열합니다.
사용법 참고사항	적용 가능한 경우, 구성 매개변수에 적용되는 모든 사용법 참고사항을 나열합니다.

에이전트 수에 영향을 주는 구성 매개변수

데이터베이스 에이전트 및 이러한 에이전트의 관리 방법과 관련된 여러 가지 데이터베이스 관리 프로그램 구성 매개변수가 있습니다.

다음의 데이터베이스 관리 프로그램 구성 매개변수는 작성되는 데이터베이스 에이전트 수와 이러한 에이전트의 관리 방법을 판별합니다.

- 에이전트 풀 *크기*(*num_poolagents*): 시스템에서 계속 사용 가능한 풀의 전체 유휴 에이전트 수입니다. 이 매개변수의 디폴트값은 100, AUTOMATIC입니다.
- 풀의 초기 에이전트 수(*num_initagents*): 데이터베이스 관리 프로그램이 시작된 경우 이 값에 따라 작업자 에이전트 풀이 작성됩니다. 이 경우 초기 쿼리의 성능이 향상됩니다. 작업자 에이전트는 모두 유휴 에이전트로 시작됩니다.
- 최대 연결 수(*max_connections*): 각 데이터베이스 파티션에서 데이터베이스 관리 프로그램 시스템에 허용되는 최대 연결 수를 지정합니다.
- 최대 코디네이팅 에이전트 수(*max_coordagents*): 파티션된 데이터베이스 환경 및 **연결 집중기**를 사용할 때 파티션 내 병렬 처리를 사용하는 환경의 경우 이 값은 코디네이팅 에이전트 수를 제한합니다.

쿼리 최적화에 영향을 주는 구성 매개변수

여러 가지 구성 매개변수가 SQL 또는 XQuery 컴파일러가 선택하는 액세스 플랜에 영향을 줍니다. 이러한 여러 매개변수는 단일 파티션 데이터베이스 환경에 적합하며 일부는 파티션된 데이터베이스 환경에만 적용됩니다. 하드웨어가 동일한 동종 파티션된 데이터베이스 환경을 가정할 때 각 매개변수에 사용된 값은 모든 데이터베이스 파티션에서 동일해야 합니다.

주: 구성 매개변수를 동적으로 변경하는 경우 옵티마이저는 패키지 캐시에서 이전 액세스 플랜으로 인해 변경된 매개변수 값을 즉시 읽지 못할 수도 있습니다. 패키지 캐시를 재설정하려면 FLUSH PACKAGE CACHE 명령을 실행하십시오.

페더레이티드 시스템에서 대부분의 쿼리가 별칭에 액세스하는 경우 환경을 변경하기 전에 보내는 쿼리의 유형을 평가하십시오. 예를 들어, 페더레이티드 데이터베이스에서 버퍼 풀은 페더레이티드 시스템의 DBMS 및 데이터인 데이터 소스에서 페이지를 캐시하지 않습니다. 이러한 이유로 인해 버퍼의 *크기*를 늘려도 옵티마이저는 별칭이 포함된 쿼리의 액세스 플랜을 선택할 때 추가 액세스 플랜 대체를 고려하지 않습니다. 그러나 옵티마이저는 데이터 소스 테이블의 로컬 구체화를 가장 비용이 적게 드는 경로나 정렬 조작의 필요한 단계로 결정할 수도 있습니다. 이 경우 사용 가능한 자원 수를 늘리면 성능이 향상될 수 있습니다.

다음의 구성 매개변수 또는 인수는 SQL 또는 XQuery 컴파일러가 선택한 액세스 플랜에 영향을 줍니다.

- 버퍼 풀을 작성하거나 변경할 때 지정한 버퍼 풀의 *크기*.

  옵티마이저는 액세스 플랜을 선택할 때 디스크에서 버퍼 풀로 페이지를 페치하는 데 필요한 입출력 비용을 고려하고 쿼리를 충족시키는 데 필요한 입출력 수를 추정합니다. 이미 버퍼 풀에 있는 페이지에서 행을 읽는 데 추가적인 실제 입출력이 필요하지 않으므로 이러한 추정 값에는 버퍼 풀 사용량 예측이 포함됩니다.

  옵티마이저는 SYSCAT.BUFFERPOOLS 시스템 카탈로그 테이블 및 파티션된 데이터베이스 환경의 SYSCAT.BUFFERPOOLDBPARTITIONS 시스템 카탈로그 테이블에서 *npages* 컬럼의 값을 고려합니다.

  테이블을 읽는 입출력 비용은 다음 사항에 영향을 줄 수 있습니다.
  - 두 테이블의 조인 방식
  - 클러스터링 해제된 인덱스를 사용하여 데이터를 읽는지 여부
- 디폴트 수준(dft_degree)

  `dft_degree` 구성 매개변수는 CURRENT DEGREE 특수 레지스터 및 DEGREE 바인드 옵션에 디폴트값을 제공하여 병렬 처리를 지정합니다. 1 값은 파티션 내 병렬 처리가 없음을 의미합니다. -1 값은 옵티마이저가 프로세서 수 및 쿼리 유형에 따라 파티션 내 병렬 처리 수준을 판별함을 의미합니다.

  주: *intra_parallel* 데이터베이스 관리 프로그램 구성 매개변수를 설정하여 사용 가능하게 설정하지 않은 경우 파티션 내 병렬 처리는 발생하지 않습니다.
- 디폴트 쿼리 최적화 클래스(dft_queryopt)

  SQL 또는 XQuery 쿼리를 컴파일할 때 쿼리 최적화 클래스를 지정할 수 있지만 디폴트 쿼리 최적화 클래스를 설정할 수도 있습니다.
- 평균 활성 응용프로그램 수(avg_appls)

  옵티마이저는 *avg_appls* 매개변수를 사용하므로 선택한 액세스 플랜에 대해 런타임에 사용 가능한 버퍼 풀의 *크기*를 추정하는 데 도움이 됩니다. 이 매개변수의 값이 크면 옵티마이저가 버퍼 풀 사용에 신중한 액세스 플랜을 선택하게 됩니다. 1 값을 지정하면 옵티마이저는 응용프로그램이 전체 버퍼 풀을 사용할 수 있는 것으로 고려합니다.
- 정렬 힙 *크기*(sortheap)

  정렬할 행이 정렬 힙에서 사용 가능한 스페이스보다 많은 스페이스를 차지하는 경우 여러 정렬 패스가 수행되며 이 경우 각 패스에서 전체 행 세트의 서브세트를 정렬합니다. 각 정렬 패스는 버퍼 풀의 시스템 임시 테이블에 저장되며 디스크에 기록될 수 있습니다. 모든 정렬 패스가 완료되면 이와 같이 정렬된 서브세트를 정렬된 단일 행 세트로 병합합니다. 정렬된 최종 데이터 목록을 저장할 시스템 임시 테이블이 필요하

지 않은 경우 정렬을 『파이프』하도록 고려할 수 있습니다. 즉, 정렬의 결과를 단일 순차 액세스에서 읽을 수 있습니다. 파이프 정렬은 파이프되지 않은 정렬보다 성능을 향상시키며 가능할 경우 사용합니다.

액세스 플랜을 선택할 때 옵티마이저는 다음을 수행하여 정렬을 파이프할 수 있는지 평가하는 등 정렬 조작의 비용을 추정합니다.

- 정렬할 데이터의 양 추정
- 정렬을 파이프할 충분한 스페이스가 있는지 판별하기 위해 *sortheap* 매개변수 확인

- 잠금 목록의 최대 스토리지(locklist) 및 에스컬레이션 전 잠금 목록의 최대 퍼센트(maxlocks)

  분리 레벨이 **반복 읽기(RR)**인 경우 옵티마이저는 *locklist* 및 *maxlocks* 매개변수를 고려하여 행 레벨 잠금을 테이블 레벨 잠금으로 에스컬레이션할 수 있는지 여부를 판별합니다. 옵티마이저가 테이블 액세스에 대한 잠금 에스컬레이션이 발생할 것인지 추정하는 경우 쿼리 실행 중 잠금 에스컬레이션의 오버헤드를 발생시키는 대신 액세스 플랜에 대한 테이블 레벨 잠금을 선택합니다.

- CPU 속도(cpuspeed)

  옵티마이저는 CPU 속도를 사용하여 특정 조작을 수행하는 비용을 추정합니다. CPU 비용 추정 및 다양한 입출력 비용 추정을 사용하여 쿼리에 가장 적합한 액세스 플랜을 선택할 수 있습니다.

  머신의 CPU 속도는 선택한 액세스 플랜에 상당한 영향을 줄 수 있습니다. 이 구성 매개변수는 데이터베이스 설치 또는 업그레이드 시 적합한 값으로 자동 설정됩니다. 테스트 시스템에서 프로덕션 환경을 모델링하거나 하드웨어 변경의 영향을 평가하는 경우가 아니면 이 매개변수를 조정하지 마십시오. 이 매개변수를 사용하여 다른 하드웨어 환경을 모델링하면 해당 환경에 대해 선택할 수 있는 액세스 플랜을 확인할 수 있습니다. 데이터베이스 관리 프로그램이 이 자동 구성 매개변수의 값을 다시 계산하도록 하려면 -1로 설정하십시오.

- 명령문 힙 크기(stmtheap)

  명령문 힙의 크기가 다른 액세스 경로를 선택하는 옵티마이저에 영향을 주지 않지만 복잡한 SQL 또는 XQuery문에 대해 수행되는 최적화의 양에 영향을 줄 수 있습니다.

  stmtheap 매개변수가 충분히 크게 설정되지 않은 경우 명령문을 처리할 수 있도록 충분한 메모리가 사용 가능하지 않음을 표시하는 경고를 받을 수 있습니다. 예를 들어, SQLCODE +437(SQLSTATE 01602)은 명령문을 컴파일하는 데 사용된 최적화의 양이 요청한 양보다 작음을 표시할 수 있습니다.

- 통신 대역폭(comm_bandwidth)

통신 대역폭은 옵티마이저가 액세스 경로를 판별하는 데 사용합니다. 옵티마이저는 이 매개변수의 값을 사용하여 파티션된 데이터베이스 환경에서 데이터베이스 파티션 서버들 간에 특정 조작을 수행하는 비용을 추정합니다.

- 응용프로그램 힙 크기(applheapsz)

  대형 스키마의 경우 응용프로그램 힙에 충분한 스페이스가 필요합니다.

max_coordagents 및 max_connections를 구성하는 경우 제한사항 및 동작

max_coordagents 및 *max_connections* 매개변수에 대한 버전 9.5의 디폴트는 AUTOMATIC이며 *max_coordagents*는 200으로 설정되고 *max_connections*는 -1로 설정됩니다(즉, *max_coordagents*의 값으로 설정됨). 이 설정은 집중기를 OFF로 설정합니다.

max_coordagents 또는 *max_connections*를 온라인으로 구성하는 경우, 다음과 같은 일부 제한사항 및 동작이 있습니다.

- *max_coordagents*의 값이 증가하는 경우 설정은 즉시 적용되며 새 요청은 새 코디네이팅 에이전트를 작성할 수 있습니다. 이 값이 감소하는 경우 코디네이팅 에이전트의 수는 즉시 줄어들지 않습니다. 오히려 코디네이팅 에이전트 에이전트의 수는 더 이상 증가하지 않으며, 코디네이팅 에이전트의 전체 수를 줄이기 위해 기존 코디네이팅 에이전트는 현재 작업 세트를 완료한 후 종료됩니다. 코디네이팅 에이전트가 필요한 작업에 대한 새 요청은 코디네이팅 에이전트의 총 수가 새 값 아래로 내려가서 코디네이팅 에이전트가 해제될 때까지 서비스되지 않습니다.
- *max_connections*의 값이 증가하는 경우 설정은 즉시 적용되며, 이 매개변수 때문에 이전에 차단되었던 새 연결이 허용됩니다. 값이 줄어드는 경우 데이터베이스 관리 프로그램은 기존 연결을 적극적으로 종료하지는 않습니다. 대신 기존 연결이 충분히 종료되어 값이 새로운 최대값 아래로 내려갈 때까지 새 연결이 허용되지 않습니다.
- *max_connections*가 -1(디폴트)로 설정된 경우 허용되는 연결의 최대수는 *max_coordagents*와 동일하며 *max_coordagents*가 오프라인 또는 온라인으로 갱신되는 경우 허용되는 연결의 최대수도 갱신됩니다.

max_coordagents 또는 *max_connections*의 값을 온라인으로 변경하는 동안에는 해당 연결 집중기를 변경할 수 없습니다. OFF이면 ON으로 변경되고 ON이면 OFF로 변경됩니다. 예를 들어, START DBM 시간에 *max_coordagents*가 *max_connections*(집중기는 ON)인 경우 이 두 매개변수에 대해 온라인으로 수행되는 모든 갱신사항은 *max_coordagents* < *max_connections* 관계를 유지해야 합니다. 마찬가지로, START DBM 시간에 *max_coordagents*가 *max_connections*(집중기는 OFF)보다 크거나 같은 경우 온라인으로 수행되는 모든 갱신사항은 이 관계를 유지해야 합니다.

이 유형의 갱신을 온라인으로 수행하는 경우 데이터베이스 관리 프로그램은 조작에 실패하지 않으며 갱신이 지연됩니다. IMMEDIATE가 지정된 데이터베이스 관리 프로그램 구성 매개변수를 갱신하는 경우처럼 경고 SQL1362W 메시지가 리턴되지만 가능하지 않습니다.

max_coordagents 또는 *max_connections*를 AUTOMATIC으로 설정하는 경우 다음 동작을 예상할 수 있습니다.

- 이 매개변수는 둘 다 시작 값과 AUTOMATIC 설정으로 구성될 수 있습니다. 예를 들어, 다음 명령은 값 200과 AUTOMATIC을 *max_coordagents* 매개변수에 연결합니다.

  ```
  UPDATE DBM CONFIG USING max_coordagents 200 AUTOMATIC
  ```

  이러한 매개변수에는 항상 연결된 값이 있습니다(디폴트로 설정된 값 또는 사용자가 지정한 값). 매개변수를 갱신할 때 AUTOMATIC만 지정되고(즉 값이 지정되지 않고) 매개변수에 이전에 연결된 값이 있는 경우에는 그 값이 남아 있게 됩니다. AUTOMATIC 설정만 영향을 받습니다.

  주: 집중기가 ON인 경우 이 두 구성 매개변수에 지정된 값은 매개변수가 AUTOMATIC으로 설정된 경우에도 중요합니다.
- 두 매개변수가 모두 AUTOMATIC으로 설정된 경우 데이터베이스 관리 프로그램은 연결 및 코디네이팅 에이전트의 수를 워크로드에 맞게 늘리도록 허용합니다. 그러나 다음 경고가 적용됩니다.
  1. 집중기가 OFF인 경우 데이터베이스 관리 프로그램은 일대일 비율을 유지합니다. 즉 모든 연결에는 각각 *하나의* 코디네이팅 에이전트만 있게 됩니다.
  2. 집중기가 ON인 경우 데이터베이스 관리 프로그램은 매개변수의 값으로 연결에 대한 코디네이팅 에이전트의 비율 설정을 유지하려 합니다.

     주:

     – 비율을 유지하기 위해 사용되는 방법은 비강제적으로 디자인되었으며 비율의 완벽한 유지를 보증하지 않습니다. 이 시나리오에서는 사용 가능한 코디네이팅 에이전트를 기다려야 하는 경우에도 항상 새 연결이 허용됩니다. 비율을 유지하기 위해 필요한 만큼 새 코디네이팅 에이전트가 작성됩니다. 연결이 종료되면 데이터베이스 관리 프로그램은 비율을 유지하기 위해 코디네이팅 에이전트도 종료합니다.

     – 데이터베이스 관리 프로그램은 사용자가 설정한 비율을 줄이지 않습니다. 사용자가 설정한 *max_coordagents* 및 *max_connections*의 초기값은 하한으로 간주됩니다.
- 이 두 매개변수의 현재 값 및 지연된 값은 CLP 또는 API와 같은 다양한 수단을 통해 표시할 수 있습니다. 표시되는 값은 항상 사용자가 설정한 값입니다. 예를 들

어, 다음 명령을 실행했고 인스턴스에서 작업을 수행하는 30개의 동시 연결이 시작된 경우 *max_connections* 및 *max_coordagents*의 표시 값은 여전히 20, AUTOMATIC입니다.

```
UPDATE DBM CFG USING max_connections 20 AUTOMATIC,
  max_coordagents 20 AUTOMATIC
```

현재 모니터 요소를 실행 중인 실제 연결 수 및 코디네이팅 에이전트 수를 판별하려면 Health Monitor를 사용하십시오.

- *max_connections*가 AUTOMATIC으로 설정되고 *max_coordagents*보다 큰 값이며 (즉 집중기가 ON) *max_coordagents*가 AUTOMATIC으로 설정되지 않은 경우, 데이터베이스 관리 프로그램은 제한된 수의 코디네이팅 에이전트만 사용할 수 있는 무제한의 연결 수를 허용합니다.

  주: 연결은 사용 가능한 코디네이팅 에이전트를 기다려야 할 수도 있습니다.

max_coordagents 및 max_connections 구성 매개변수에 AUTOMATIC 옵션을 사용하는 것은 다음 두 시나리오에서만 유효합니다.

1. 두 매개변수를 모두 AUTOMATIC으로 설정
2. 집중기는 *max_connections*를 AUTOMATIC으로 설정하고 *max_coordagents*는 그렇지 않은 경우에 사용 가능합니다.

이 매개변수에 대해 AUTOMATIC을 사용하는 다른 모든 구성은 차단되며 이 두 매개변수에 대해 유효한 AUTOMATIC 설정을 설명하는 이유 코드와 함께 SQL6112N을 리턴합니다.

데이터베이스 관리 프로그램 구성 매개변수

agent_stack_sz - 에이전트 스택 크기

이 매개변수는 DB2가 각 에이전트에 할당하는 가상 메모리를 판별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Linux(32비트)

256 [16 – 1024]

Linux(64비트) 및 UNIX

1024 [256 – 32768]

Windows

16 [8 – 1000]

수치 단위

페이지(4KB)

할당 시기

응용프로그램의 작업을 수행하기 위해 에이전트가 초기화될 때

사용 가능한 시기

에이전트가 응용프로그램에 대해 수행할 작업을 완료할 때

이 매개변수를 사용하면 지정된 응용프로그램 세트에 대해 서버의 메모리 사용을 최적화할 수 있습니다. 간단한 쿼리에 사용되는 스페이스에 비해 복잡한 쿼리가 스택 스페이스를 많이 사용합니다.

이 매개변수를 사용하여 Windows 환경에서 각 에이전트에 대해 초기에 커미트된 스택 *크기*를 설정합니다. 디폴트로 각 에이전트 스택은 디폴트 예약 스택 *크기* 256KB(64 4KB 페이지)까지 확장될 수 있습니다. 대부분의 데이터베이스 조작에서 이 한계를 사용하면 충분합니다. UNIX 및 Linux에서는 *agent_stack_sz*가 다음으로 높은 제곱 값으로 반올림됩니다. UNIX의 디폴트 설정을 사용하면 대부분의 워크로드에서 충분합니다.

그러나 큰 SQL 또는 XQuery문을 준비할 때 에이전트가 스택 스페이스를 모두 사용할 수 있으며 시스템은 스택 오버플로우 예외(0xC00000FD)를 생성합니다. 이러한 상태가 발생하면 오류는 복구 불가능하므로 서버가 종료됩니다.

주: 버전 9.5 이상에서는 스택 오버플로우 예외 대신 sqlcode -973이 리턴됩니다.

*agent_stack_sz*를 디폴트 예약 스택 *크기* 64바이트보다 큰 값으로 설정하여 에이전트 스택 *크기*를 늘릴 수 있습니다. *agent_stack_sz*의 값이 디폴트 예약 스택 *크기*보다 크면 Windows 운영 체제는 1MB의 가장 근접한 배수로 반올림합니다. 에이전트 스택 *크기*를 128 4KB 페이지로 설정하면 각 에이전트의 1MB 스택을 실제로 예약합니다. *agent_stack_sz*의 값을 디폴트 예약 스택 *크기*보다 적게 설정하면 필요한 경우 스택이 디폴트 예약 스택 *크기*까지 계속 확장되므로 최대 한계에는 영향을 미치지 않습니다. 이 경우 *agent_stack_sz*의 값은 에이전트 작성 시 스택에 대해 커미트된 초기 메모리입니다.

db2hdr 유틸리티를 사용하여 `db2syscs.exe` 파일에 대한 헤더 정보를 변경하면 디폴트 예약 스택 *크기*를 변경할 수 있습니다. *agent_stack_sz*를 변경하면 에이전트의 스택

*크기*만 영향을 받지만 디폴트 예약 스택 *크기*를 변경하면 모든 스레드가 영향을 받습니다. db2hdr 유틸리티를 사용하여 디폴트 스택 *크기*를 변경하면 세분화도 높아지므로 스택 *크기*를 필요한 최소 스택 *크기*로 설정할 수 있습니다. 그러나 db2syscs.exe의 변경사항을 적용하려면 DB2를 중지하고 재시작해야 합니다.

권장사항: 32비트 환경에서 대형 또는 복잡한 XML 데이터에 대해 작업하는 경우 최소한 256 4KB 페이지로 *agent_stack_sz*를 갱신해야 합니다. 스키마 등록 또는 XML 문서 유효성 확인 중 스택 오버플로우 예외를 방지하기 위해 XML 스키마가 매우 복잡하면 *agent_stack_sz*를 한계에 가깝게 설정해야 합니다.

사용자의 환경이 다음과 일치하는 경우 다른 클라이언트가 추가 어드레스 스페이스를 사용할 수 있도록 스택 *크기*를 줄일 수 있습니다.

- 복잡한 쿼리가 없는 간단한 응용프로그램(예: 간단한 OLTP)만 있는 경우
- 상대적으로 많은 수의 동시 클라이언트가 필요한 경우(예: 100개 이상)

Windows에서 에이전트 스택 *크기*와 동시 클라이언트 수는 반비례 관계입니다. 스택 크기가 크면 실행할 수 있는 동시 클라이언트의 잠재적인 수치가 줄어듭니다. Windows 플랫폼에서는 어드레스 스페이스가 제한되어 있으므로 이러한 상태가 발생합니다.

agentpri - 에이전트 우선순위

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 이전 버전과 동일하게 계속 작동되고 지속적으로 이 매개변수를 지원합니다. 워크로드 관리(WLM)에 이 매개변수를 사용하는 경우 WLM 서비스 클래스 에이전트 우선순위를 무시합니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에만 적용됩니다.

이 매개변수는 운영 체제 스케줄러가 모든 에이전트 및 기타 데이터베이스 관리 프로그램 인스턴스 프로세스와 스레드에 부여하는 우선순위를 제어합니다. 이 우선순위는 머신에서 실행 중인 기타 프로세스 및 스레드와 관련하여 데이터베이스 관리 프로그램 프로세스, 에이전트 및 스레드에 CPU 시간을 제공하는 방식을 판별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

AIX -1 (system) [41 - 125]

기타 UNIX

-1 (system) [41 - 128]

Windows

-1 (system) [0 - 6]

Solaris

-1 (system) [0 - 59]

매개변수가 -1 또는 system으로 설정된 경우 특별한 조치를 수행하지 않으며 운영 체제가 모든 프로세스 및 스레드를 스케줄하는 일반적인 방식으로 데이터베이스 관리 프로그램을 스케줄합니다. 매개변수가 -1 또는 system 이외의 값으로 설정된 경우 데이터베이스 관리 프로그램은 정적 우선순위를 매개변수의 값으로 설정하여 프로세스 및 스레드를 작성합니다. 따라서 이 매개변수를 사용하면 머신에서 데이터베이스 관리 프로그램 프로세스 및 스레드(파티션된 데이터베이스 환경에서는 코디네이팅 및 서브에이전트, 병렬 시스템 제어기 및 FCM 디먼도 포함)를 실행하는 우선순위를 제어할 수 있습니다.

이 매개변수를 사용하여 데이터베이스 관리 프로그램 처리량을 늘릴 수 있습니다. 이 매개변수에 설정하는 값은 데이터베이스 관리 프로그램을 실행 중인 운영 체제에 따라 다릅니다. 예를 들어, Linux 또는 UNIX 환경에서는 숫자가 낮으면 우선순위가 높습니다. 매개변수가 41과 125 사이의 값으로 설정된 경우 데이터베이스 관리 프로그램은 UNIX 정적 우선순위를 매개변수의 값으로 설정하여 에이전트를 작성합니다. 값의 숫자가 낮으면 데이터베이스 관리 프로그램의 우선순위가 높으므로 Linux 및 UNIX 환경에서는 이 점이 중요하지만 기타 프로세스(응용프로그램 및 사용자 포함)가 충분한 CPU 시간을 확보할 수 없으므로 지연이 발생할 수 있습니다. 머신에서 예상되는 기타 활동과 이 매개변수 설정의 균형을 맞춰야 합니다.

제한사항:

- Linux 및 UNIX 플랫폼에서 이 매개변수를 디폴트값이 아닌 다른 값으로 설정한 경우 조정자(governor)를 사용하여 에이전트 우선순위를 변경할 수 없습니다.
- Solaris 운영 체제에서는 디폴트값(-1)을 변경할 수 없습니다. 디폴트값을 변경하면 DB2 프로세스의 우선순위가 실시간으로 설정되므로 시스템에서 사용 가능한 모든 자원을 독점할 수 있습니다.

권장사항: 처음에 디폴트값을 사용해야 합니다. 이 값은 기타 사용자/응용프로그램에 대한 응답 시간과 데이터베이스 관리 프로그램 처리량의 적절한 절충안을 제공합니다.

데이터베이스 성능이 문제가 되는 경우 벤치마킹 기술을 사용하여 이 매개변수에 대한 최적의 설정을 판별할 수 있습니다. 기타 사용자 프로세스의 성능이 심하게 저하될 수

있으므로 특히 CPU 사용이 매우 높을 때 데이터베이스 관리 프로그램의 우선순위를 높이려면 주의해야 합니다. 데이터베이스 관리 프로그램 프로세스 및 스레드의 우선순위를 높이면 상당한 성능 이점이 있습니다.

alternate_auth_enc - 서버에서 수신 연결의 대체 암호화 알고리즘 구성 매개변수

이 구성 매개변수는 인증을 위해 DB2 데이터베이스 서버에 제출된 사용자 ID 및 암호를 암호화하는 데 사용되는 대체 암호화 알고리즘을 지정합니다. 특히 이 매개변수는 DB2 클라이언트와 DB2 데이터베이스 서버 간에 협상된 인증 메소드가 SERVER_ENCRYPT일 때 암호화 알고리즘에 영향을 줍니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

NOT_SPECIFIED [AES_CMP; AES_ONLY]

DB2 데이터베이스 서버에서 인증을 위해 제출된 사용자 ID와 암호는 DB2 클라이언트와 DB2 서버 간에 협상된 인증 메소드가 SERVER_ENCRYPT일 때 암호화됩니다. 협상된 인증 메소드는 서버에 설정된 인증 유형 및 클라이언트가 요청한 인증 유형에 종속됩니다. 사용자 ID와 암호를 암호화하는 데 사용되는 암호화 알고리즘의 선택은 **alternate_auth_enc** 데이터베이스 관리 프로그램 구성 매개변수 설정에 종속됩니다. 이 설정에 따라 DES 또는 AES가 될 수 있습니다.

디폴트(NOT_SPECIFIED) 값이 사용되는 경우 데이터베이스 서버는 클라이언트가 제안하는 암호화 알고리즘을 승인합니다.

alternate_auth_enc가 AES_ONLY로 설정되면 데이터베이스 서버는 AES 인증을 사용하는 연결만 승인합니다. 클라이언트가 AES 인증을 지원하지 않는 경우 연결은 거부됩니다.

alternate_auth_enc가 AES_CMP로 설정되면 데이터베이스 서버는 AES 또는 DES를 사용하여 암호화되는 사용자 ID와 암호를 승인하지만, 클라이언트가 AES 암호화를 지원하는 경우 AES에 대해 협상하지 않습니다.

aslheapsz - 응용프로그램 지원 계층 힙 크기

응용프로그램 지원 계층 힙은 로컬 응용프로그램과 연관된 에이전트 사이의 통신 버퍼를 나타냅니다. 시작된 각 데이터베이스 관리 프로그램 에이전트가 이 버퍼를 공유 메모리로 할당합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

15 [1 - 524 288]

수치 단위

페이지(4KB)

할당 시기

로컬 응용프로그램의 데이터베이스 관리 프로그램 에이전트 프로세스가 시작될 때

사용 가능한 시기

데이터베이스 관리 프로그램 프로세스가 종료될 때

데이터베이스 관리 프로그램에 대한 요청 또는 연관된 응답이 버퍼에 맞지 않는 경우 둘 이상의 송신 및 수신 쌍으로 분할됩니다. 단일 송신 및 수신 쌍을 사용하여 대부분의 요청을 처리하도록 이 버퍼의 *크기*를 설정해야 합니다. 요청의 *크기*는 다음을 보유하는 데 필요한 스토리지에 따라 다릅니다.

- 입력 SQLDA
- SQLVAR의 연관된 모든 데이터
- 출력 SQLDA
- 일반적으로 250바이트를 초과하지 않는 기타 필드

이 통신 버퍼 외에 이 매개변수를 두 가지 다른 용도로 사용합니다.

- 이것은 블로킹 커서가 열려 있을 때 입출력 블록 *크기*를 판별하는 데 사용됩니다. 블로킹 커서의 이 메모리는 응용프로그램의 전용 어드레스 스페이스로부터 할당되므로 각 응용프로그램에 할당할 최적의 전용 메모리 양을 판별해야 합니다. Data Server Runtime Client가 응용프로그램의 전용 메모리로부터 블로킹 커서의 스페이스를 할당할 수 없는 경우 비블로킹 커서가 열립니다.

- 에이전트와 db2fmp 프로세스 사이의 통신 *크기*를 판별하는 데 사용됩니다(db2fmp 프로세스는 사용자 정의 함수(UDF)이거나 분리(fenced) 스토어드 프로시저임). 시스템에서 사용 중인 각 db2fmp 프로세스 또는 스레드에 대해 공유 메모리에서 바이트 수가 할당됩니다.

로컬 응용프로그램에서 보낸 데이터는 데이터베이스 관리 프로그램이 쿼리 힙에서 할당된 인접 메모리 세트로 받습니다. *aslheapsz* 매개변수를 사용하여 쿼리 힙의 *초기 크기*를 판별합니다(로컬 및 리모트 클라이언트에 모두 해당). 쿼리 힙의 최대 *크기*는 *query_heap_sz* 매개변수에 정의되어 있습니다.

권장사항: 응용프로그램의 요청이 일반적으로 작으며 메모리 제한 시스템에서 응용프로그램을 실행 중인 경우 이 매개변수의 값을 줄일 수 있습니다. 쿼리가 일반적으로 매우 커서 여러 개의 송신 및 수신 요청이 필요하고 시스템의 메모리가 제한되지 않은 경우 이 매개변수의 값을 늘릴 수 있습니다.

다음 공식을 사용하여 *aslheapsz*의 최소 페이지 수를 계산하십시오.

```
aslheapsz >= ( sizeof(input SQLDA)
             + sizeof(each input SQLVAR)
             + sizeof(output SQLDA)
             + 250 ) / 4096
```

여기서 sizeof(x)는 제공된 입력 또는 출력 값의 페이지 수를 계산하는 바이트 단위의 x *크기*입니다.

또한 블로킹 커서의 수 및 잠재적 *크기*에 대한 이 매개변수의 영향을 고려해야 합니다. 전송 중인 행의 수 또는 *크기*가 큰 경우(예: 데이터의 양이 4096바이트보다 큰 경우) 성능이 향상될 수 있습니다. 그러나 큰 레코드 블록으로 각 연결의 작업 세트 메모리 *크기*가 커진다는 점에서 트레이드 오프가 있습니다.

또한 큰 레코드 블록으로 응용프로그램에 실제로 필요한 것보다 많은 페치 요청이 발생할 수 있습니다. 응용프로그램에서 SELECT문에 OPTIMIZE FOR절을 사용하여 페치 요청 수를 제어할 수 있습니다.

audit_buf_sz - 감사 버퍼 크기

이 매개변수는 데이터베이스 감사 시 사용되는 버퍼의 *크기*를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

0 [0 - 65 000]

수치 단위

페이지(4KB)

할당 시기

DB2가 시작될 때

사용 가능한 시기

DB2가 중지될 때

이 매개변수의 디폴트값은 영(0)입니다. 값이 영(0)인 경우 감사 버퍼를 사용하지 않습니다. 값이 영(0)보다 큰 경우 감사 기능으로 생성된 감사 레코드가 저장될 감사 버퍼에 스페이스가 할당됩니다. 이 값에 4KB 페이지를 곱하면 감사 버퍼에 할당되는 스페이스의 양입니다. 감사 버퍼를 동적으로 할당할 수 없습니다. 이 매개변수의 새 값을 적용하려면 먼저 DB2를 중지한 후 재시작해야 합니다.

이 매개변수를 디폴트에서 영(0)보다 큰 값으로 변경하면 감사 기능은 감사 레코드를 생성하는 명령문의 실행에 비해 비동기로 디스크에 레코드를 기록합니다. 이 경우 매개변수 값 영(0)을 그대로 사용하므로 DB2 성능이 향상됩니다. 영(0) 값은 감사 기능이 감사 레코드를 생성하는 명령문을 실행하여 동기식으로(동시에) 디스크에 레코드를 기록함을 의미합니다. 감사 중 동기 조작으로 DB2에서 실행 중인 응용프로그램의 성능이 낮아집니다.

authentication - 인증 유형

이 매개변수는 사용자의 인증이 발생하는 방식과 위치를 지정하고 판별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

SERVER [CLIENT; SERVER; SERVER_ENCRYPT; DATA_ENCRYPT;

DATA_ENCRYPT_CMP; KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

authentication이 SERVER인 경우 사용자 ID와 암호가 클라이언트에서 서버로 전송되므로 서버에서 인증이 발생할 수 있습니다. SERVER_ENCRYPT 값은 네트워크를 통해 전송되는 사용자 ID와 암호가 암호화된다는 점을 제외하고 SERVER와 동작이 동일합니다.

DATA_ENCRYPT 값은 서버가 암호화된 SERVER 인증 스킴과 사용자 데이터의 암호화를 승인함을 의미합니다. 인증은 SERVER_ENCRYPT와 정확히 동일한 방식으로 작동됩니다.

다음의 사용자 데이터는 이 인증 유형을 사용할 때 암호화됩니다.

- SQL문
- SQL 프로그램 변수 데이터
- SQL문을 처리하고 데이터에 대한 설명이 들어 있는 서버의 출력 데이터
- 쿼리에서 발생한 일부 또는 모든 응답 세트 데이터
- 대형 오브젝트(LOB) 스트리밍
- SQLDA 디스크립터

DATA_ENCRYPT_CMP 값은 서버가 암호화된 SERVER 인증 스킴과 사용자 데이터의 암호화를 승인함을 의미합니다. 또한 이 인증 유형을 통해 DATA_ENCRYPT 인증 유형을 지원하지 않는 이전 제품과 호환할 수 있습니다. 이러한 제품은 사용자 데이터를 암호화하지 않고 SERVER_ENCRYPT 인증 유형을 사용하여 연결할 수 있습니다. 새 인증 유형을 지원하는 제품이 이를 사용해야 합니다. 이 인증 유형은 서버의 데이터베이스 관리 프로그램 구성 파일에서만 유효하며 CATALOG DATABASE 명령에서 사용할 경우 유효하지 않습니다.

주: 표준 준수(『표준 준수』 주제에 정의됨) 구성을 위해 SERVER 값만 지원됩니다.

CLIENT 값은 모든 인증이 클라이언트에서 발생함을 표시합니다. 서버에서는 인증을 수행할 필요가 없습니다.

KERBEROS 값은 인증을 위해 Kerberos 보안 프로토콜을 사용하여 Kerberos 서버에서 인증이 수행됨을 의미합니다. Kerberos 보안 시스템을 지원하는 클라이언트 및 서버에서 인증 유형 KRB_SERVER_ENCRYPT를 사용하는 경우 효과적인 시스템 인증 유형은 KERBEROS입니다. 클라이언트가 Kerberos 보안 시스템을 지원하지 않는 경우 시스템 인증 유형은 사실상 SERVER_ENCRYPT와 동등합니다.

GSSPLUGIN 값은 외부 GSSAPI 기반 보안 메커니즘을 사용하여 인증이 수행됨을 의미합니다. GSSPLUGIN 보안 메커니즘을 지원하는 클라이언트 및 서버에서 인증 유형 GSS_SERVER_ENCRYPT를 사용하는 경우 효과적인 시스템 인증 유형은

GSSPLUGIN입니다(즉, 클라이언트가 서버의 플러그인 중 하나를 지원하는 경우). 클라이언트가 GSSPLUGIN 보안 메커니즘을 지원하지 않는 경우 시스템 인증 유형은 사실상 SERVER_ENCRYPT와 동등합니다.

권장사항: 일반적으로 로컬 클라이언트의 경우 디폴트값(SERVER)이면 충분합니다. 리모트 클라이언트가 데이터베이스 서버에 연결되는 경우 사용자 ID와 암호를 보호하기 위해 SERVER_ENCRYPT 값을 사용하는 것이 좋습니다.

auto_reval - 자동 유효성 다시 확인 및 무효화 구성 매개변수

이 구성 매개변수는 유효성 다시 확인 및 무효화 시맨틱을 제어합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

DEFERRED [IMMEDIATE, DISABLED, DEFERRED, DEFERRED_FORCE]

새 데이터베이스를 작성하는 경우 기본적으로 이 구성 매개변수가 DEFERRED로 설정됩니다.

버전 9.5 또는 그 이전 버전에서 데이터베이스를 업그레이드하는 경우 **auto_reval**이 DISABLED로 설정됩니다. 유효성 다시 확인 동작은 이전 릴리스와 동일합니다.

이 매개변수를 IMMEDIATE로 설정하는 경우, 모든 종속 오브젝트는 유효하지 않게 되는 즉시 유효성이 다시 확인됩니다. 이는 ALTER TABLE, ALTER COLUMN 또는 CREATE OR REPLACE와 같은 일부 DDL문에도 적용됩니다. 종속 오브젝트의 유효성 다시 확인 성공 여부는 기타 DDL 변경사항에 따라 달라지지 않습니다. 따라서 유효성 다시 확인을 즉시 완료할 수 있습니다.

이 매개변수를 DEFERRED로 설정하는 경우, 모든 종속 오브젝트는 다음에 이들 오브젝트에 액세스할 때 유효성이 다시 확인됩니다.

이 매개변수를 IMMEDIATE 또는 DEFERRED로 설정하고 유효성 다시 확인 조작에 실패하는 경우, 유효하지 않은 종속 오브젝트는 다음에 이들 오브젝트에 액세스할 때까지 유효하지 않은 상태로 남아 있습니다.

이 매개변수를 DEFERRED_FORCE로 설정하면 DEFERRED로 설정한 경우와 동일하게 작동하고 추가적인 CREATE 기능이 사용 가능하며 오류가 발생합니다.

사용자가 명시적으로 지정하는 구문이 **auto_reval** 설정을 겹쳐쓰는 경우도 있습니다. 예를 들어, CASCADE 또는 RESTRICT를 지정하지 않고 ALTER TABLE 문의

DROP COLUMN절을 사용하는 경우, 시맨틱은 **auto_reval**이 제어합니다. 그러나 CASCADE 또는 RESTRICT를 지정하는 경우에는 **auto_reval**이 지정하는 새 시맨틱을 겹쳐써서 이전 연쇄 및 제한 시맨틱이 사용됩니다.

이 구성 매개변수는 동적입니다. 즉 이 값을 변경하면 즉시 적용됩니다. 변경사항을 적용하기 위해 데이터베이스에 다시 연결할 필요가 없습니다.

catalog_noauth - 권한 없이 카탈로그화 가능

이 매개변수는 사용자가 SYSADM 권한이 없어도 데이터베이스 및 노드 또는 DCS 및 ODBC 디렉토리를 카탈로그 및 카탈로그 해제할 수 있는지를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

로컬 및 리모트 클라이언트가 있는 데이터베이스 서버

NO [NO (0) — YES (1)]

클라이언트(로컬 클라이언트가 있는 데이터베이스 서버)

YES [NO (0) — YES (1)]

이 매개변수의 디폴트값(0)은 SYSADM 권한이 필요함을 표시합니다. 이 매개변수가 1(예)로 설정된 경우 SYSADM 권한이 필요하지 않습니다.

clnt_krb_plugin - 클라이언트 Kerberos 플러그인

이 매개변수는 클라이언트 측 인증 및 로컬 권한 부여에 사용할 디폴트 Kerberos 플러그인 라이브러리의 이름을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버

- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null 또는 IBMkrb5 [유효한 문자열]

이 값은 디폴트로 Linux 및 UNIX 시스템에서 널(NULL)이며, Windows 운영 체제에서 IBMkrb5입니다. 클라이언트가 KERBEROS 인증을 사용하여 인증되거나 로컬 권한 부여가 수행되고 DBM CFG에서 인증 유형이 KERBEROS인 경우 이 플러그인이 사용됩니다.

clnt_pw_plugin - 클라이언트 사용자 ID 암호 플러그인

이 매개변수는 클라이언트 측 인증 및 로컬 권한 부여에 사용할 사용자 ID 암호 플러그인 라이브러리의 이름을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 문자열]

디폴트로 이 값은 널(NULL)이며 DB2가 제공하는 사용자 ID 암호 플러그인 라이브러리가 사용됩니다. 클라이언트가 CLIENT 인증을 사용하여 인증되거나 로컬 권한 부여가 수행되고 DBM CFG에서 인증 유형이 CLIENT, SERVER, SERVER_ENCRYPT 또는 DATA_ENCRYPT인 경우 이 플러그인이 사용됩니다. 루트 서버 설치가 아닌 경우 DB2 사용자 ID와 암호 플러그인 라이브러리가 사용되면 DB2 제품을 사용하기 전에 db2rfe 명령을 실행해야 합니다.

cluster_mgr - 클러스터 관리 프로그램 이름

데이터베이스 관리 프로그램은 이 매개변수를 사용하여 지정된 클러스터 관리 프로그램에 대한 증분 클러스터 구성 변경사항을 통신할 수 있습니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 다중 파티션 데이터베이스 서버

매개변수 유형

정보용

디폴트 디폴트 없음

유효한 값

- TSA

이 매개변수는 DB2 High Availability Instance Configuration Utility(db2haicu)를 사용하여 고가용성 클러스터를 구성할 때 설정됩니다.

comm_bandwidth - 통신 대역폭

이 매개변수는 쿼리 옵티마이저가 데이터베이스 파티션 서버들 간 대역폭을 표시하여 액세스 경로를 판별하는 데 도움이 됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

명령문 경계

디폴트 [범위]

-1 [.1 - 100 000]

-1 값을 사용할 경우 매개변수 값이 디폴트로 재설정됩니다. 디폴트값은 기본 통신 어댑터의 속도에 따라 계산됩니다. 기가비트 이더넷을 사용 중인 시스템의 경우 100 값을 예상할 수 있습니다.

수치 단위

초 당 MB

통신 대역폭으로 계산된 값(초 당 MB)은 쿼리 옵티마이저가 파티션된 데이터베이스 시스템의 데이터베이스 파티션 서버들 간 특정 조작을 수행하는 비용을 계산하는 데 사용

합니다. 옵티마이저는 클라이언트와 서버 간 통신 비용을 모델링하지 않으므로 이 매개변수는 데이터베이스 파티션 서버들 간 명목상 대역폭(있는 경우)만 반영해야 합니다.

테스트 시스템에서 프로덕션 환경을 모델링하거나 하드웨어 업그레이드의 영향을 평가하도록 이 값을 명시적으로 설정할 수 있습니다.

권장사항: 다른 환경을 모델링할 경우에만 이 매개변수를 조정해야 합니다.

통신 대역폭은 옵티마이저가 액세스 경로를 판별하는 데 사용합니다. 이 매개변수를 변경한 후 응용프로그램을 리바인드해야 합니다(REBIND PACKAGE 명령 사용).

conn_elapse - 연결 경과 시간

이 매개변수는 두 개의 데이터베이스 파티션 서버들 간 TCP/IP 연결을 설정해야 하는 제한 시간(초)을 지정합니다.

구성 유형
데이터베이스 관리 프로그램

적용 대상
로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형
온라인으로 구성 가능

전파 클래스
즉시

디폴트 [범위]
10 [0–100]

수치 단위
초

이 매개변수에 지정된 시간 안에 연결 시도가 성공하면 통신이 설정됩니다. 실패한 경우 연결을 다시 시도합니다. *max_connretries* 매개변수에 지정된 횟수만큼 연결을 시도했으나 항상 시간종료된 경우 오류가 발행됩니다.

cpuspeed - CPU 속도

이 매개변수는 데이터베이스가 설치된 머신의 CPU 속도를 반영합니다.

구성 유형
데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버

- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

명령문 경계

디폴트 [범위]

-1 [$1x10^{-10}$ — 1] -1 값을 사용하면 측정 프로그램의 실행에 따라 매개변수 값이 재설정됩니다.

수치 단위

밀리초

IBM RS/6000® 모델 530H의 데이터가 파일에 없거나 사용자 머신의 데이터가 파일에 없는 경우 벤치마크 결과가 사용 가능하지 않으면 이 프로그램이 실행됩니다.

테스트 시스템에서 프로덕션 환경을 모델링하거나 하드웨어 업그레이드의 영향을 평가하도록 이 값을 명시적으로 설정할 수 있습니다. -1로 설정하여 *cpuspeed*를 다시 계산합니다.

권장사항: 다른 환경을 모델링할 경우에만 이 매개변수를 조정해야 합니다.

CPU 속도는 옵티마이저가 액세스 경로를 판별하는 데 사용합니다. 이 매개변수를 변경한 후 응용프로그램을 리바인드해야 합니다(REBIND PACKAGE 명령 사용).

cur_commit - 현재 커미트됨 구성 매개변수

이 매개변수는 커서 안정성(CS) 스캔의 동작을 제어합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

ON [ON, AVAILABLE, DISABLED]

새 데이터베이스의 경우 디폴트는 ON으로 설정됩니다. 디폴트가 ON으로 설정된 경우 쿼리는 쿼리가 제출될 때 데이터의 현재 커미트된 값을 리턴합니다.

데이터베이스를 업그레이드하는 동안 **cur_commit** 구성 매개변수는 DISABLED로 설정되어 이전 릴리스와 동일한 동작을 유지합니다. 커서 안정성 스캔에서 현재 커미트된 값을 사용하려면 업그레이드 후에 **cur_commit** 구성 매개변수를 ON으로 설정해야 합니다.

cur_commit 구성 매개변수를 명시적으로 AVAILABLE로 설정할 수 있습니다. 이 매개변수를 설정한 후 현재 커미트된 결과를 보려면 현재 커미트된 동작을 명시적으로 요청해야 합니다.

주: 세 가지 레지스트리 변수 DB2_EVALUNCOMMITTED, DB2_SKIPDELETED 및 DB2_SKIPINSERTED는 커서 안정성 분리 레벨이 사용되는 경우 현재 커미트됨의 영향을 받습니다. 이러한 레지스트리 변수는 USE CURRENTLY COMMITTED 또는 WAIT FOR OUTCOME이 바인드 또는 명령문 준비 시간에 명시적으로 지정된 경우에는 무시됩니다.

권장사항: 전용 로그 디스크에 상당한 양의 읽기 활동이 있거나 하드 디스크를 많이 이용하는 경우 로그 버퍼 *크기* 매개변수 **logbufsz**를 사용하여 버퍼 영역의 *크기*를 늘리십시오. 로그 버퍼 영역은 **dbheap** 매개변수가 제어하는 스페이스를 사용하므로 **logbufsz** 매개변수의 값을 늘리는 경우에는 **dbheap** 매개변수도 고려해야 합니다.

date_compat - 날짜 호환성 데이터베이스 구성 매개변수

이 매개변수는 TIMESTAMP(0) 데이터 유형에 연결된 DATE 호환성 시맨틱이 연결된 데이터베이스에 적용되는지 여부를 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

이 값은 데이터베이스를 작성할 때 DATE 지원의 DB2_COMPATIBILITY_VECTOR 레지스트리 변수 설정에 따라 결정됩니다. 이 값은 변경할 수 없습니다.

dec_to_char_fmt - 10진수를 문자로 함수 구성 매개변수

이 매개변수는 10진수를 문자 값으로 변환하는 경우 CHAR 스칼라 함수와 CAST 스펙의 결과를 제어하는 데 사용됩니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

아래의 dec_to_char_fmt 변경 결과를 참조하십시오.

디폴트 [범위]

NEW [NEW, V95]

이 매개변수 설정은 선행 영(0) 및 트레일링 10진수 문자가 CHAR 함수의 결과에 포함되는지 여부를 결정합니다. 이 매개변수를 NEW로 설정하면 선행 영(0) 및 트레일링 10진수 문자가 포함되지 않습니다. 이 매개변수를 V95로 설정하면 선행 영(0) 및 트레일링 10진수 문자가 포함됩니다.

선행 영(0) 및 트레일링 10진수 문자는 CHAR 함수와 동일한 구문이 있는 CHAR_OLD 스칼라 함수의 결과에도 포함됩니다.

dec_to_char_fmt 값 변경의 효과

- 버전 9.7 이전에 작성한 구체화된 쿼리 테이블(MQT)에는 NEW 설정을 사용하여 작성한 MQT와 다른 결과가 있을 수 있습니다. 이전에 작성한 MQT에 새 형식을 따르는 데이터만 포함되도록 하려면 REFRESH TABLE 문을 사용하여 MQT를 새로 고치십시오.
- 트리거 결과는 변경된 형식의 영향을 받을 수 있습니다. 매개변수의 값을 NEW로 설정하여 형식을 변경해도 이미 기록된 데이터에는 영향을 미치지 않습니다.
- 데이터를 테이블에 삽입하도록 허용한 제한조건이 재평가되면 동일한 데이터를 거부할 수 있습니다. 마찬가지로 데이터를 테이블에 삽입하도록 허용하지 않은 제한조건이 재평가되면 동일한 데이터를 승인할 수 있습니다. SET INTEGRITY 문을 사용하여 테이블에서 더 이상 제한사항을 충족하지 않을 수 있는 데이터를 점검하고 정정하십시오.
- **dec_to_char_fmt**의 값을 변경한 후에는 **dec_to_char_fmt** 값 변경에 따라 결과가 영향을 받는 생성된 컬럼의 값에 의존하는 모든 정적 SQL 패키지를 재컴파일하십시오. 영향을 받는 SQL 패키지를 알아내려면 컴파일하고 db2rbind 명령을 사용하여 모든 패키지를 리바인드해야 합니다.

dft_account_str - 디폴트 접미부 어카운트

이 매개변수는 기본 ID의 디폴트 접미부 역할을 수행합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

Null [유효한 문자열]

각 응용프로그램 연결 요청마다 DB2 Connect 생성 접두부와 사용자 제공 접미부로 구성되어 있는 어카운팅 ID가 응용프로그램 리퀘스터(AR)에서 DRDA 응용프로그램 서버로 전송됩니다. 이 어카운팅 정보는 시스템 관리자가 자원 사용을 각 사용자 액세스와 연관시킬 수 있는 메커니즘을 제공합니다.

주: 이 매개변수는 DB2 Connect에만 적용됩니다.

접미부는 `sqlesact()` API를 호출하는 응용프로그램이나 환경 변수 DB2ACCOUNT를 설정하는 사용자가 제공합니다. API 또는 환경 변수가 접미부를 제공하지 않는 경우 DB2 Connect는 이 매개변수의 값을 디폴트 접미부 값으로 사용합니다. 이 매개변수는 특히 DB2 Connect로 어카운팅 문자열을 전달할 수 있는 기능이 없는 이전 데이터베이스 클라이언트(버전 2 이전)의 경우에 유용합니다.

권장사항: 다음을 사용하여 이 어카운팅 문자열을 설정하십시오.

- 영문자(A - Z)
- 숫자(0 - 9)
- 밑줄(_)

dft_monswitches - 디폴트 데이터베이스 시스템 모니터 스위치

이 매개변수를 사용하면 매개변수 비트에 따라 각각 내부적으로 표시되는 여러 스위치를 설정할 수 있습니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

주: dft_mon_xxxx 스위치 설정을 수정하기 전에 인스턴스에 명시적으로 ATTACH하는 경우 변경사항이 즉시 적용됩니다. 그렇지 않으면 다음에 인스턴스가 재시작될 때 설정이 적용됩니다.

디폴트 디폴트로 켜지는 dft_mon_timestamp를 제외하고 모든 스위치가 꺼집니다.

이 매개변수는 다음의 매개변수를 설정하여 이러한 각 스위치를 독립적으로 갱신할 수 있다는 점에서 고유합니다.

dft_mon_uow
스냅샷 모니터의 작업 단위(UOW) 스위치에 대한 디폴트값

dft_mon_stmt
스냅샷 모니터의 명령문 스위치에 대한 디폴트값

dft_mon_table
스냅샷 모니터의 테이블 스위치에 대한 디폴트값

dft_mon_bufpool
스냅샷 모니터의 버퍼 풀 스위치에 대한 디폴트값

dft_mon_lock
스냅샷 모니터의 잠금 스위치에 대한 디폴트값

dft_mon_sort
스냅샷 모니터의 정렬 스위치에 대한 디폴트값

dft_mon_timestamp
스냅샷 모니터의 시간소인 스위치에 대한 디폴트값

권장사항: ON으로 전환된 스위치(dft_mon_timestamp 제외)는 데이터베이스 관리 프로그램에 해당 스위치 관련 모니터 데이터를 수집하도록 지시합니다. 추가 모니터 데이터를 수집하면 데이터베이스 관리 프로그램 오버헤드가 증가하여 시스템 성능에 영향을 줄 수 있습니다. dft_mon_timestamp 스위치를 OFF로 전환하는 경우 CPU 사용이 100%에 접근할 때 중요해집니다. 이 상태가 발생하면 시간 소인을 발생하는 데 필요한 CPU 시간이 갑자기 증가합니다. 또한 시간소인 스위치가 OFF로 전환되면 모니터 스위치 제어를 받는 다른 데이터의 전체 비용이 상당히 감소합니다.

응용프로그램이 첫 번째 모니터링 요청(예: 스위치 설정, 이벤트 모니터 활성화, 스냅샷 작성)을 발생할 때 모든 모니터링 응용프로그램은 디폴트 스위치 설정을 상속합니다. 데이터베이스 관리 프로그램이 시작된 후에 데이터를 수집할 경우에만 구성 파일에서 스위치를 켜야 합니다. (그렇지 않으면 각 모니터링 응용프로그램이 자체 스위치를 설정할 수 있으며 수집하는 데이터는 스위치가 설정되는 시간에 비례하게 됩니다.)

dftdbpath - 디폴트 데이터베이스 경로

이 매개변수에는 데이터베이스 관리 프로그램에서 데이터베이스를 작성하는 데 사용되는 디폴트 파일 경로가 있습니다. 데이터베이스 작성 시 경로가 지정되지 않은 경우 *dftdbpath* 매개변수에 지정된 경로에서 데이터베이스가 작성됩니다.

구성 유형
데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

UNIX 인스턴스 소유자의 홈 디렉토리 [기존의 경로]

Windows

DB2가 설치된 드라이브 [기존의 경로]

파티션된 데이터베이스 환경에서는 데이터베이스가 작성되는 경로가 NFS 마운트 경로(Linux 및 UNIX 플랫폼의 경우) 또는 네트워크 드라이브(Windows 환경의 경우)가 아닌지 확인해야 합니다. 지정된 경로는 각 데이터베이스 파티션 서버에 실제로 존재해야 합니다. 혼동을 방지하기 위해 각 데이터베이스 파티션 서버에서 로컬로 마운트된 경로를 지정하는 것이 가장 좋습니다. 경로의 최대 길이는 205자입니다. 시스템은 경로의 끝에 데이터베이스 파티션 이름을 추가합니다.

데이터베이스가 크게 확장될 수 있으며 여러 사용자가 데이터베이스를 작성할 수 있는 경우(사용자의 환경 및 의도에 따라 다름) 지정된 위치에서 모든 데이터베이스를 작성하고 저장하면 편리합니다. 또한 무결성을 보장하고 쉽게 백업 및 복구할 수 있도록 데이터베이스를 다른 응용프로그램 및 데이터에서 분리할 수 있으면 좋습니다.

Linux 및 UNIX 환경의 경우 *dftdbpath* 이름의 길이는 215자를 초과할 수 없으며 유효한 절대 경로 이름이어야 합니다. Windows의 경우 *dftdbpath*에는 드라이브 이름과 선택적으로 뒤에 콜론을 사용할 수 있습니다.

권장사항: 가능하면 운영 체제 파일 및 데이터베이스 로그와 같이 자주 액세스하는 다른 데이터와 다른 디스크에 대용량 데이터베이스를 저장하십시오.

diaglevel - 진단 오류 캡처 레벨

이 매개변수는 db2diag 로그 파일에 기록되는 진단 오류 유형을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트

- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

3 [0 — 4]

이 매개변수에 유효한 값은 다음과 같습니다.

- **0** – 진단 데이터가 캡처되지 않음
- **1** – 심각한 오류만
- **2** – 모든 오류
- **3** – 모든 오류 및 경고
- **4** – 모든 오류, 경고 및 정보 메시지

diagpath 구성 매개변수는 *diaglevel* 매개변수의 값에 따라 오류 파일, 경보 로그 파일 및 생성될 수 있는 모든 덤프 파일이 포함되는 디렉토리를 지정하는 데 사용됩니다.

권장사항: 이 매개변수의 값을 늘려서 문제를 해결하는 데 도움이 되는 추가 문제점 판별 데이터를 수집할 수 있습니다.

diagpath - 진단 데이터 디렉토리 경로

이 매개변수를 사용하여 DB2 진단 정보의 완전한 경로를 지정할 수 있습니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

Null [유효한 모든 경로 이름]

이 디렉토리에는 사용자의 플랫폼에 따라 덤프 파일, 트랩 파일, 오류 로그, 통지 파일, 경보 로그 파일 및 FODC(First Occurrence Data Collection) 패키지가 포함될 수 있습니다.

이 매개변수가 널(NULL)인 경우 진단 정보는 다음 디렉토리 또는 폴더 중 하나에 있는 파일에 기록됩니다.

- Windows 환경의 경우:
  - 사용자 데이터 파일(예: 인스턴스 디렉토리의 파일)은 다음과 같이 코드가 설치된 위치와 다른 위치에 기록됩니다.
    - Windows Vista 환경에서는 사용자 데이터 파일이 `ProgramData¥IBM¥DB2¥` 로 기록됩니다.
    - Windows 2003 및 XP 환경에서는 사용자 데이터 파일이 `Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥`*Copy Name*으로 기록됩니다. 여기서 *Copy Name*은 DB2 사본의 이름입니다.
- Linux 및 UNIX 환경의 경우 정보는 *INSTHOME*`/sqllib/db2dump`에 기록되며 여기서 *INSTHOME*은 인스턴스의 홈 디렉토리입니다.

버전 9.5의 경우 전역 레벨에서 **DB2INSTPROF**의 디폴트값은 위에 표시된 새 위치에 저장됩니다. 런타임 데이터 파일의 위치를 지정하는 다른 프로파일 레지스트리 변수는 **DB2INSTPROF**의 값을 쿼리해야 합니다. 기타 변수는 다음과 같습니다.

- **DB2CLIINIPATH**
- **DIAGPATH**
- **SPM_LOG_PATH**

권장사항: *diagpath* 구성 매개변수의 디폴트 설정을 사용하거나 여러 인스턴스의 *diagpath* 값으로 중앙 위치를 사용하십시오.

파티션된 데이터베이스 환경에서 *diagpath* 매개변수는 호스트의 로컬 스토리지를 사용하여 로깅의 성능을 향상시킵니다. 이 경우 각 실제 파티션의 개별 로깅 및 진단 디렉토리가 작성됩니다. PD_GET_DIAG_HIST 테이블 함수를 사용하면 다른 파티션에서 로그 레코드를 검색할 수 있으며 PD_GET_LOG_MSGS 테이블 함수를 사용하면 모든 파티션에서 통지 로그를 검색할 수 있습니다.

dir_cache - 디렉토리 캐시 지원

이 매개변수는 데이터베이스, 노드 및 DCS 디렉토리 파일이 메모리에서 캐싱되는지 여부를 판별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Yes [Yes; No]

할당 시기

- 응용프로그램이 첫 번째 연결을 발행하면 응용프로그램 디렉토리 캐시가 할당됩니다.
- 데이터베이스 관리 프로그램 인스턴스가 시작되면(db2start) 서버 디렉토리 캐시가 할당됩니다.

사용 가능한 시기

- 응용프로그램 프로세스가 종료되면 응용프로그램 디렉토리 캐시가 사용 가능해집니다.
- 데이터베이스 관리 프로그램 인스턴스가 중지되면(db2stop) 서버 디렉토리 캐시가 사용 가능해집니다.

디렉토리 캐시를 사용하면 디렉토리 파일 입출력이 제거되고 디렉토리 정보를 검색하는 데 필요한 디렉토리 검색이 최소화되어 연결 비용이 줄어듭니다. 두 가지 유형의 디렉토리 캐시가 있습니다.

- 응용프로그램을 실행 중인 머신에서 각 응용프로그램 프로세스마다 할당 및 사용되는 응용프로그램 디렉토리 캐시
- 내부 데이터베이스 관리 프로그램 프로세스의 일부에 할당 및 사용되는 서버 디렉토리 캐시

응용프로그램 디렉토리 캐시의 경우 응용프로그램이 첫 번째 연결을 발행하면 각 디렉토리 파일을 읽고 이 응용프로그램의 전용 메모리에 정보가 캐싱됩니다. 캐시는 연속 연결 요청에서 응용프로그램 프로세스가 사용하며 응용프로그램 프로세스가 활성화된 동안에 유지됩니다. 응용프로그램 디렉토리 캐시에 데이터베이스가 없으면 디렉토리 파일에서 정보를 검색하지만 캐시는 갱신되지 않습니다. 응용프로그램이 디렉토리 항목을 수정한 경우 이 응용프로그램에서 다음 연결을 시도하면 이 응용프로그램의 캐시를 새로 고칩니다. 기타 응용프로그램의 응용프로그램 디렉토리 캐시를 새로 고치지 않습니다. 응용프로그램 프로세스가 종료되면 캐시가 사용 가능해집니다. (명령행 처리기 세션에서 사용하는 디렉토리 캐시를 새로 고치려면 db2 terminate 명령을 발행하십시오.)

서버 디렉토리 캐시의 경우 데이터베이스 관리 프로그램 인스턴스가 시작되면(db2start) 각 디렉토리 파일을 읽고 서버 메모리에 정보가 캐싱됩니다. 이 캐시는 인스턴스가 중지될 때까지(db2stop) 유지됩니다. 이 캐시에 디렉토리 항목이 없는 경우 디렉토리 파일에서 정보를 검색합니다. 이 서버 디렉토리 캐시는 인스턴스를 실행하는 동안 새로 고치지 않습니다.

권장사항: 디렉토리 파일이 자주 변경되지 않으며 성능이 중요한 경우 디렉토리 캐싱을 사용하십시오.

또한 리모트 클라이언트에서는 응용프로그램이 다른 여러 연결 요청을 발행하는 경우 디렉토리 캐싱이 유용할 수 있습니다. 이 경우 캐싱은 단일 응용프로그램이 디렉토리 파일을 읽어야 하는 횟수를 줄입니다.

디렉토리 캐싱은 데이터베이스 시스템 모니터 스냅샷의 성능을 향상시킬 수도 있습니다. 또한 데이터베이스 별명을 사용하는 대신 스냅샷 호출에서 데이터베이스 이름을 명시적으로 참조해야 합니다.

주: 디렉토리 캐싱이 작동되었으며 데이터베이스 관리 프로그램이 시작된 후 데이터베이스가 카탈로그됨, 카탈로그 해제, 작성 또는 삭제된 경우 스냅샷 호출을 수행하면 오류가 발생할 수 있습니다.

discover - 발견 모드

이 매개변수는 클라이언트가 작성할 수 있는 발견 요청(있는 경우)의 종류를 판별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

SEARCH [DISABLE, KNOWN, SEARCH]

클라이언트 관점에서 다음 중 하나가 발생합니다.

- *discover* = SEARCH인 경우 클라이언트는 검색 발견 요청을 발행하여 네트워크에서 DB2 서버 시스템을 찾을 수 있습니다. 검색 발견은 KNOWN 발견에서 제공하

는 기능의 수퍼 세트를 제공합니다. *discover* = SEARCH인 경우 클라이언트가 검색 및 알려진 발견 요청을 모두 발행할 수 있습니다.

- *discover* = KNOWN인 경우 클라이언트에서 알려진 발견 요청만 발행할 수 있습니다. 특정 시스템의 관리 서버에 대한 연결 정보를 지정하여 DB2 시스템에 대한 모든 인스턴스 및 데이터베이스 정보가 클라이언트로 리턴됩니다.
- *discover* = DISABLE인 경우 클라이언트에서 발견을 사용하지 않습니다.

디폴트 발견 모드는 SEARCH입니다.

discover_inst - 서버 인스턴스 발견

이 매개변수는 DB2 발견에서 이 인스턴스를 발견할 수 있는지 여부를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

ENABLE [ENABLE, DISABLE]

매개변수의 디폴트인 enable은 인스턴스를 발견할 수 있음을 지정하지만 disable은 인스턴스를 발견하지 못하도록 방지합니다.

fcm_num_buffers - FCM 버퍼 수

이 매개변수는 데이터베이스 서버들 간 및 데이터베이스 서버 내에서 내부 통신(메시지)에 사용되는 4KB 버퍼 수를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

32비트 플랫폼

Automatic [128 - 65 300]

64비트 플랫폼

Automatic [128 - 524 288]

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버: 디폴트는 1024임
- 로컬 클라이언트가 있는 데이터베이스 서버: 디폴트는 512임
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버: 디폴트는 4096임

단일 파티션 데이터베이스 시스템에서 이 매개변수는 *intra_parallel* 매개변수를 디폴트값 NO에서 YES로 변경하여 파티션 내 병렬 처리가 사용 가능한 경우에만 사용됩니다.

초기값 및 AUTOMATIC 속성을 모두 설정할 수 있습니다.

AUTOMATIC으로 설정된 경우 FCM은 자원 사용도를 모니터하고 30분 안에 사용되지 않은 경우 자원을 늘리거나 줄일 수 있습니다. 자원을 늘리거나 줄이는 양은 플랫폼에 따라 다르며 특히 Linux에서는 시작 값보다 25%만 늘릴 수 있습니다. 데이터베이스 관리 프로그램이 인스턴스 시작 시 지정된 자원 수를 할당할 수 없는 경우 인스턴스를 시작할 수 있을 때까지 구성 값을 조금씩 다시 늘립니다.

동일한 머신에 여러 개의 논리 노드가 있는 경우 이 매개변수의 값을 늘려야 할 수도 있습니다. 또한 시스템의 사용자 수, 시스템의 데이터베이스 파티션 서버 수 또는 복잡한 응용프로그램으로 인해 메시지 버퍼를 모두 사용한 경우 이 매개변수의 값을 늘려야 합니다.

여러 개의 논리 노드를 사용 중인 경우 하나의 *fcm_num_buffers* 버퍼 풀을 동일한 버퍼의 모든 논리 노드가 공유합니다. 풀의 크기는 *fcm_num_buffers* 값에 해당 실제 머신의 논리 노드 수를 곱하여 판별합니다. 사용 중인 값을 다시 조사하십시오. 여러 논리 노드가 상주하는 머신에 할당할 전체 FCM 버퍼 수를 고려하십시오.

fcm_num_channels - FCM 채널 수

이 매개변수는 각 데이터베이스 파티션의 FCM 채널 수를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버
- 로컬 클라이언트가 있는 Satellite 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

UNIX 32비트 플랫폼

자동, 시작 값은 256, 512, 2 048 [128 - 120 000]

UNIX 64비트 플랫폼

자동, 시작 값은 256, 512, 2 048 [128 - 524 288]

Windows 32비트

자동, 시작 값은 10 000 [128 - 120 000]

Windows 64비트

자동, 시작 값은 256, 512, 2 048 [128 - 524 288]

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버의 경우 시작 값은 512입니다.
- 로컬 클라이언트가 있는 데이터베이스 서버의 경우 시작 값은 256입니다.
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 환경 서버의 경우 시작 값은 2 048입니다.

파티션되지 않은 데이터베이스 환경에서는 *intra_parallel* 매개변수가 활성이어야 *fcm_num_channels*를 사용할 수 있습니다.

FCM 채널은 DB2 엔진에서 실행 중인 EDU 간의 논리적 통신 엔드 포인트를 나타냅니다. 제어 순서(요청 및 응답)와 데이터 순서(테이블 큐 데이터) 둘 다 채널에 의존하여 파티션 간에 데이터를 전송합니다.

AUTOMATIC으로 설정된 경우 FCM은 채널 사용량을 모니터하며 요구사항 변화에 따라 증분하여 자원을 할당하고 해제합니다.

fed_noauth - 페더레이티드 인증 생략

이 매개변수는 인스턴스에서 페더레이티드 인증이 생략되는지 여부를 결정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

No [Yes; No]

*fed_noauth*가 *yes*로 설정되고 *authentication*이 *server* 또는 *server_encrypt*로 설정되고 *federated*가 *yes*로 설정된 경우, 인스턴스에서 인증이 생략됩니다. 인증은 데이터 소스에서 수행된 것으로 가정합니다. *fed_noauth*가 *yes*로 설정되는 경우 주의하십시오. 인증은 클라이언트와 DB2 둘 다에서 수행되지 않습니다. SYSADM 인증 이름을 아는 사용자는 페더레이티드 서버의 SYSADM 권한을 가정할 수 있습니다.

federated - 페더레이티드 데이터베이스 시스템 지원

이 매개변수는 데이터 소스(예: DB2 계열 및 Oracle)별로 관리되는 데이터에 대해 분산 요청(DR)을 제출하는 응용프로그램 지원의 사용 가능 여부를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

No [Yes; No]

federated_async - 쿼리당 최대 비동기 TQ 수 구성 매개변수

이 매개변수는 페더레이티드 서버가 지원하는 액세스 플랜에서 비동기 테이블 큐(ATQ)의 최대수를 판별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 페더레이션이 사용되는 경우 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

0 [0 - 32 767 포함, -1, ANY]

ANY 또는 -1이 지정된 경우, 옵티마이저가 액세스 플랜의 ATQ 수를 판별합니다. 옵티마이저는 해당 플랜의 모든 적격 SHIP 또는 리모트 푸시다운 연산자에 ATQ를 지정합니다. DB2_MAX_ASYNC_REQUESTS_PER_QUERY 서버 옵션에 지정된 값은 비동기 요청의 수를 제한합니다.

권장사항

federated_async 구성 매개변수는 특수 레지스터 및 바인드 옵션의 디폴트 또는 시작 값을 제공합니다. CURRENT FEDERATED ASYNCHRONY 특수 레지스터, FEDERATED_ASYNCHRONY 바인드 옵션 또는 FEDERATED_ASYNCHRONY prepcompile 옵션의 값을 더 크거나 작은 수로 설정하여 이 매개변수 값을 겹쳐쓸 수 있습니다.

특수 레지스터 또는 바인드 옵션이 *federated_async* 구성 매개변수를 겹쳐쓰지 않는 경우 매개변수의 값은 액세스 플랜에서 페더레이티드 서버가 허용하는 ATQ의 최대수를 결정합니다. 특수 레지스터 또는 바인드 옵션이 이 매개변수를 겹쳐쓰는 경우, 특수 레지스터 또는 바인드 옵션의 값이 플랜에서 ATQ의 최대수를 결정합니다.

federated_async 구성 매개변수에 대한 변경사항은 현재 작업 단위(UOW)를 커미트하자마자 동적문에 영향을 미칩니다. 연속 동적문은 새 값을 자동으로 인식합니다. 페더레이티드 데이터베이스를 재시작할 필요는 없습니다. Embedded SQL 패키지는 *federated_async* 구성 매개변수의 값이 변경될 때 무효화되지도 않고 내재적으로 리바인드되지도 않습니다.

federated_async 구성 매개변수의 새 값이 정적 SQL문에 영향을 미치도록 하려면 패키지를 리바인드해야 합니다.

fenced_pool - 최대 분리(fenced) 프로세스 수

이 매개변수는 스레드 db2fmp 프로세스(스레드 안전 스토어드 프로시저 및 UDF를 처리하는 프로세스)의 각 db2fmp 프로세스에서 캐싱되는 스레드 수를 나타냅니다. 비스레드 db2fmp 프로세스의 경우 이 매개변수는 캐싱된 프로세스 수를 나타냅니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

-1 (*max_coordagents*), Automatic [-1; 0–64 000]

수치 단위

카운터

제한사항:

- 이 매개변수가 동적으로 갱신되고 값이 낮아진 경우 데이터베이스 관리 프로그램은 db2fmp 스레드 또는 프로세스를 미리 종료하지 않고 대신 캐싱된 db2fmp의 수를 새 값으로 줄이기 위해 사용할 때 캐싱을 중지합니다.
- 이 매개변수가 동적으로 갱신되고 값이 증가한 경우 데이터베이스 관리 프로그램은 작성되는 추가 db2fmp 스레드 및 프로세스를 캐싱합니다.
- 이 매개변수가 -1(디폴트)로 설정된 경우 *max_coordagents* 구성 매개변수의 값을 가정합니다. 자동 설정 또는 동작이 아닌 *max_coordagents*의 값만 가정합니다.
- 이 매개변수가 AUTOMATIC(이 값도 디폴트임)으로 설정된 경우:
  - 데이터베이스 관리 프로그램은 코디네이팅 에이전트의 상위 워터 마크(water mark)에 따라 캐싱되는 db2fmp 스레드 및 프로세스 수를 늘릴 수 있도록 합니다. 특히 이 매개변수의 자동 동작을 사용하면 데이터베이스 관리 프로그램이 등록하여 시작된 코디네이팅 에이전트의 최대 수에 따라 증가할 수 있습니다.
  - 이 매개변수에 지정된 값은 캐싱할 db2fmp 스레드 및 프로세스의 하한을 나타냅니다.

권장사항: 사용자의 환경에서 분리 스토어드 프로시저 또는 사용자 정의 함수를 사용하는 경우 이 매개변수를 사용하면 인스턴스에서 실행되는 동시 스토어드 프로시저 및 UDF의 최대 수를 처리할 수 있도록 적합한 db2fmp 프로세스 수가 사용 가능한지 확인할 수 있으므로 스토어드 프로시저 및 UDF 실행의 일부분으로 새 분리 모드 프로세스를 작성할 필요가 없습니다.

db2fmp 프로세스에 제공되는 시스템 자원의 양이 적합하지 않아서 데이터베이스 관리 프로그램의 성능에 영향을 주므로 디폴트값이 사용자의 환경에 적합하지 않다고 판단된 경우 이 매개변수를 조정하는 시작점으로 다음을 사용할 수 있습니다.

```
fenced_pool = 스토어드 프로시저 및 UDF 호출을 한 번에
작성할 수 있는 응용프로그램의 수
```

*keepfenced*가 YES로 설정된 경우 캐시 풀에서 작성된 각 db2fmp 프로세스는 계속 존재하며 분리 루틴 호출이 처리되어 에이전트로 리턴된 후에도 시스템 자원을 사용합니다.

*keepfenced*가 NO로 설정된 경우 비스레드 db2fmp 프로세스가 실행이 완료된 후 종료되고 캐시 풀이 없습니다. 멀티스레드 db2fmp 프로세스는 계속 존재하지만 이러한 프로세스에서 스레드가 풀링되지 않습니다. 즉, *keepfenced*가 NO로 설정된 경우에도 하나의 스레드 C db2fmp 프로세스와 하나의 스레드 Java db2fmp 프로세스가 시스템에 있습니다.

이전 버전에서 이 매개변수는 *maxdari*입니다.

group_plugin - 그룹 플러그인

이 매개변수는 그룹 플러그인 라이브러리의 이름을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 문자열]

디폴트로 이 값은 널(NULL)이며, DB2는 운영 체제 그룹 찾아보기를 사용합니다. 모든 그룹 찾아보기에 대해 플러그인이 사용됩니다. 루트 서버 설치가 아닌 경우 DB2 사용자 ID와 암호 플러그인 라이브러리가 사용되면 DB2 제품을 사용하기 전에 db2rfe 명령을 실행해야 합니다.

health_mon - 상태 모니터링

이 매개변수를 사용하여 다양한 Health 표시기에 따라 인스턴스, 연결된 데이터베이스 및 데이터베이스 오브젝트를 모니터할 것인지 여부를 지정할 수 있습니다.

구성 유형

데이터베이스 관리 프로그램

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

On [On; Off]

관련 매개변수

*health_mon*이 켜진 경우(디폴트), 에이전트는 선택된 오브젝트의 상태 정보를 수집합니다. 오브젝트의 상태가 좋지 않은 경우, 설정한 임계값에 따라 통지를 보낼 수 있으며 조치가 자동으로 수행될 수 있습니다. *health_mon*을 끄면 오브젝트의 상태가 모니터되지 않습니다.

Health Center 또는 CLP를 사용하여 모니터할 인스턴스와 데이터베이스 오브젝트를 선택할 수 있습니다. 또한 Health Monitor에서 수집한 데이터에 따라 통지를 보낼 위치 및 수행할 조치를 지정할 수도 있습니다.

indexrec - 인덱스 재작성 시간

이 매개변수는 데이터베이스 관리 프로그램이 유효하지 않은 인덱스를 재빌드하려고 시도하는 시기와 대기 데이터베이스에서 DB2 롤 포워드 또는 HADR 로그 재생 중 인덱스 빌드가 재실행되는지 여부를 표시합니다.

구성 유형

데이터베이스 및 데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

UNIX 데이터베이스 관리 프로그램

restart [restart; restart_no_redo; access; access_no_redo]

Windows 데이터베이스 관리 프로그램

restart [restart; restart_no_redo; access; access_no_redo]

데이터베이스

시스템 설정값 사용 [system; restart; restart_no_redo; access; access_no_redo]

이 매개변수에는 다섯 가지 설정이 가능합니다.

SYSTEM

데이터베이스 관리 프로그램 구성 파일에 지정된 *시스템 설정값 사용*은 유효하지 않은 인덱스를 재빌드하는 시기와 DB2 롤 포워드 또는 HADR 로그 재생 중 인덱스 빌드 로그 레코드가 재실행되는지 여부를 결정합니다(주: 이 설정은 데이터베이스 구성의 경우에만 유효함).

ACCESS

인덱스에 처음 액세스할 때 유효하지 않은 인덱스가 재빌드됩니다. 완전히 로깅된 인덱스 빌드가 DB2 롤 포워드 또는 HADR 로그 재생 중 재실행됩니다. HADR이 시작되고 HADR 인계가 발생하면 기본 테이블에 처음 액세스할 때 인계 후 유효하지 않은 인덱스가 재빌드됩니다.

ACCESS_NO_REDO

기본 테이블에 처음 액세스할 때 유효하지 않은 인덱스가 재빌드됩니다. 완전히 로깅된 인덱스 빌드가 DB2 롤 포워드 또는 HADR 로그 재생 중 재실행되며 이러한 인덱스는 계속 유효하지 않습니다. HADR이 시작되고 HADR 인계가 발생하면 기본 테이블에 처음 액세스할 때 인계 후 유효하지 않은 인덱스가 재빌드됩니다.

RESTART

*indexrec*의 디폴트값입니다. RESTART DATABASE 명령이 명시적 또는 내재적으로 발행된 경우 유효하지 않은 인덱스가 재빌드됩니다. 완전히 로깅된 인덱스 빌드가 DB2 롤 포워드 또는 HADR 로그 재생 중 재실행됩니다. HADR이 시작되고 HADR 인계가 발생하면 인계 종료 시 유효하지 않은 인덱스가 재빌드됩니다.

RESTART DATABASE 명령은 *autorestart* 매개변수를 사용하는 경우 내재적으로 발행됩니다.

RESTART_NO_REDO

RESTART DATABASE 명령이 명시적 또는 내재적으로 발행된 경우 유효하지 않은 인덱스가 재빌드됩니다(*autorestart* 매개변수를 사용하는 경우 RESTART DATABASE 명령이 내재적으로 발행됨). 완전히 로깅된 인덱스 빌드는 DB2 롤 포워드 또는 HADR 로그 재생 중 재실행되지 않으며 대신 이러한 인덱스는 롤 포워드가 완료되거나 HADR 인계가 발생할 때 재빌드됩니다.

치명적 디스크 문제가 발생하면 인덱스가 유효하지 않게 될 수 있습니다. 데이터 자체에 이러한 상태가 발생하면 데이터가 유실될 수 있습니다. 그러나 인덱스에 이러한 상

태가 발생하면 인덱스를 재작성하여 복구할 수 있습니다. 사용자가 데이터베이스에 연결되어 있는 동안 인덱스를 재빌드하는 경우 두 가지 문제가 발생할 수 있습니다.

- 인덱스 파일이 재작성될 때 응답 시간이 예기치 않게 느려질 수 있습니다. 테이블에 액세스하고 이 특정 인덱스를 사용하는 사용자는 인덱스가 재빌드되는 동안 대기합니다.
- 인덱스 재작성 후 특히 인덱스 재작성의 원인이 되는 사용자 트랜잭션이 COMMIT 또는 ROLLBACK을 수행하지 않은 경우 예기치 않은 잠금이 보유될 수 있습니다.

권장사항: 사용자가 많은 서버와 재시작 시간이 문제가 되지 않는 경우 이 옵션에 대해 가장 적합한 선택은 손상 이후 데이터베이스를 다시 온라인 상태로 전환하는 프로세스의 일부분으로 DATABASE RESTART 시 인덱스를 재빌드하는 것입니다.

이 매개변수를 『ACCESS』 또는 『ACCESS_NO_REDO』로 설정하면 인덱스가 재작성되는 동안 데이터베이스 관리 프로그램의 성능이 저하됩니다. 해당 특정 인덱스 또는 테이블에 액세스하는 사용자는 인덱스가 재작성될 때까지 기다려야 합니다.

이 매개변수가 『RESTART』로 설정된 경우 데이터베이스를 재시작하는 데 걸리는 시간은 인덱스 재작성으로 인해 길어지지만 데이터베이스가 다시 온라인 상태로 전환된 후 정상 처리는 영향을 받지 않습니다.

주: 데이터베이스 복구 시 복구 중인 데이터베이스에 속한 파일 시스템의 모든 SQL 프로시저 실행 파일이 제거됩니다. *indexrec*가 RESTART로 설정된 경우 모든 SQL 프로시저 실행 파일을 데이터베이스 카탈로그에서 추출하여 다음에 데이터베이스에 연결할 때 파일 시스템에 다시 저장합니다. *indexrec*가 RESTART로 설정되지 않은 경우 SQL 프로시저를 처음 실행할 때에만 파일 시스템으로 SQL 실행 파일을 추출합니다.

RESTART 값과 RESTART_NO_REDO 값의 차이 또는 ACCESS 값과 ACCESS_NO_REDO 값의 차이는 CREATE INDEX 및 REORG INDEX 조작과 같은 인덱스 빌드 조작이나 인덱스 재빌드를 위해 전체 로깅이 활성화된 경우에만 중요합니다. *logindexbuild* 데이터베이스 구성 매개변수를 사용하거나 테이블 변경 시 LOG INDEX BUILD를 사용하여 로깅을 활성화할 수 있습니다. *indexrec*를 RESTART 또는 ACCESS로 설정하면 로깅된 인덱스 빌드와 관련된 조작은 인덱스 오브젝트를 유효하지 않은 상태로 두지 않고 롤 포워드할 수 있으므로 나중에 인덱스를 재빌드해야 합니다.

instance_memory - 인스턴스 메모리

이 매개변수는 데이터베이스 파티션에 할당할 수 있는 메모리의 최대량을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버

- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

32비트 플랫폼

Automatic [0 - 1 000 000]

64비트 플랫폼

Automatic [0 - 68 719 476 736]

수치 단위

페이지(4KB)

할당 시기

인스턴스가 시작될 때

사용 가능한 시기

인스턴스가 중지될 때

instance_memory의 디폴트값은 AUTOMATIC이며, 실제 값은 데이터베이스 파티션 활성화 시간(db2start)에 계산됨을 의미합니다. 사용되는 실제 값은 시스템에 있는 실제 RAM의 75퍼센트 - 95퍼센트를 인스턴스에 구성된 로컬 데이터베이스 파티션 수로 나눈 값입니다. 이 값은 전용 데이터베이스 서버 시스템에 알맞아야 합니다.

주:

- **instance_memory**에 지정된 값이 시스템의 실제 메모리 양보다 큰 경우 db2start는 SQL1220N(데이터베이스 관리 프로그램 공유 메모리 설정을 할당할 수 없음)이 발생하여 실패합니다.
- **instance_memory**가 동적으로 실제 RAM 양보다 작은 값으로 갱신되는 경우, 요청이 처리되고 새로 더 높은 한계가 설정됩니다. 사용 중인 **instance_memory**의 현재 양보다 새 설정이 큰 경우에만 **instance_memory**를 동적으로 감소할 수 있습니다. 그렇지 않으면 요청이 다음 db2start까지 지연됩니다.
- 인스턴스가 활성일 때 **instance_memory**가 실제 RAM의 양보다 큰 값으로 동적으로 갱신되는 경우, 요청은 지연되고 다음 db2start는 SQL1220N(데이터베이스 관리 프로그램 공유 메모리 설정을 할당할 수 없음)이 발생하여 실패합니다.

FCM(Fast Communication Manager) 공유 메모리가 할당된 경우, 전체 FCM 공유 메모리 *크기* 중 각 로컬 데이터베이스 파티션의 몫은 데이터베이스 파티션의 **instance_memory** 한계 내에서 결정됩니다.

특정 힙에 대해 메모리가 요청되었을 때 데이터베이스 파티션 한계(**instance_memory**)에 이미 접근한 경우, DB2는 우선 요청된 메모리 양만큼 모든 공유 및 개인용 힙에서 메모리 사용량을 줄이려고 시도합니다. 여유 공간 **instance_memory**가 여전히 충분하지 않은 경우에는 요청이 실패하고 요청을 시작한 응용프로그램은 힙에서 메모리 부족 장애가 발생했음을 설명하는 적절한 SQLCODE를 수신합니다.

이 동작의 예외는 DB2의 조작에 큰 영향을 미치는 메모리 요청입니다. (즉, 메모리 요청이 실패하면 데이터베이스가 유효하지 않은 것으로 표시되거나 인스턴스가 종료됩니다.) 중요한 요청은 우선 데이터베이스 파티션의 현재 메모리 사용량을 줄이려고 시도합니다. 여전히 여유 공간 **instance_memory**가 충분하지 않은 경우, DB2는 운영 체제에서 메모리를 계속 요청합니다. 운영 체제가 메모리 요청을 허용하면 **instance_memory**의 현재 값이 구성된 한계를 초과하지만 메모리 여유 공간이 충분할 때까지 중요하지 않은 다른 모든 메모리 요청이 실패합니다.

주: DPF 인스턴스의 제한사항: instance_memory가 DB2 데이터베이스 파티션이 할당할 수 있는 메모리 양을 지정하더라도 이 매개변수는 인스턴스 레벨 구성 매개변수이므로 모든 데이터베이스 파티션에는 동일한 **instance_memory** 설정이 있습니다. 그러나 **instance_memory**가 AUTOMATIC으로 설정되는 경우 실제 상한은 RAM의 양 및 정의된 로컬 파티션 수에 따라 각 독립 머신에서 개별적으로 계산되므로 다른 파티션에 다른 메모리 한계가 유효할 수 있습니다.

DB2 메모리 사용량 제어:

instance_memory가 AUTOMATIC으로 설정되는 경우 인스턴스에 대한 총 메모리 사용량의 고정 상한은 인스턴스 시작 시(db2start) 설정됩니다. DB2의 실제 메모리 사용량은 워크로드에 따라 다양합니다. 런타임에 STMM이 **database_memory** 조정을 수행할 수 있는 경우(새 데이터베이스의 디폴트로), STMM은 기능 메모리 요구사항에 사용 가능한 **instance_memory**를 충분히 유지하면서 데이터베이스 공유 메모리 세트에서 시스템의 여유 공간 실제 메모리에 따라 성능에 중요한 힙의 *크기*를 갱신합니다.

워크로드에 따라 DB2의 디폴트 메모리 구성은 전체 인스턴스 메모리의 명시적 자체 성능 조정을 요구하지 않고 메모리 요구사항에 맞게 조정됩니다. 예를 들면,

- 많이 사용되는 인스턴스의 경우 STMM은 성능에 중요한 힙 *크기*를 필요한 만큼 늘립니다. 더 많은 데이터베이스 에이전트가 응용프로그램을 서비스하고 기능 메모리를 사용하므로 기능 메모리가 더 많이 사용됩니다. **instance_memory**는 충분히 있으나 시스템의 실제 메모리에 여유 공간이 거의 없는 경우 STMM은 시스템이 페이징을 시작하지 않도록 성능에 중요한 힙의 *크기*를 줄이기 시작합니다. 기능 메모리 요구사항이 삭제(drop)되면 시스템의 실제 메모리 여유 공간은 증가해야 하며 STMM은 성능에 중요한 힙을 다시 늘리기 시작합니다.

- 덜 사용되는 인스턴스의 경우 인스턴스는 기능 메모리를 덜 사용하므로 시스템에 실제 메모리 여유 공간이 충분히 있는 경우 STMM은 성능에 중요한 힘을 줄입니다.

instance_memory가 특정 값으로 설정되고 최소 하나의 활성 데이터베이스의 **database_memory** 값이 AUTOMATIC이며 이 데이터베이스에 STMM이 사용되는 경우, STMM은 기능 메모리 요청에 대해서만 충분한 여유 공간 **instance_memory**가 사용 가능하도록 하면서 DB2가 **instance_memory**로 지정된 거의 전체 메모리 양을 사용하도록 **database_memory** *크기*를 늘립니다. 이 시나리오에서 STMM은 머신의 여유 공간 실제 메모리를 모니터하지 않으므로 **instance_memory**는 페이징이 발생하지 않도록 적절하게 구성되어야 합니다.

새 admin_get_dbp_mem_usage 사용자 정의 함수(UDF)를 사용하여 DB2 인스턴스가 특정 데이터베이스 파티션 또는 모든 데이터베이스 파티션에 대해 사용하는 총 메모리 사용량을 구할 수 있습니다. 이 UDF는 현재 상한 값도 리턴합니다.

일부 Linux 커널의 한계:

일부 Linux 커널에서는 운영 체제 한계 때문에 **instance_memory**가 특정 값(AUTOMATIC이 아닌)으로 설정되지 않는 한 STMM은 **database_memory**를 AUTOMATIC으로 설정할 수 없습니다. **database_memory**를 AUTOMATIC으로 설정하고 **instance_memory**가 나중에 다시 AUTOMATIC으로 설정된 경우, **database_memory** 구성 매개변수는 다음 데이터베이스 활성화 동안에 COMPUTED로 갱신됩니다. 일부 데이터베이스가 이미 활성인 경우 STMM은 전체 **database_memory** *크기*에 대한 조정을 중지합니다. 이 한계는 Red Hat Enterprise Linux(RHEL) 5 및 SUSE Linux Enterprise Server 10 SP1 이상의 플랫폼에서는 제거되었습니다.

intra_parallel - 파티션 내 병렬 처리 사용

이 매개변수는 데이터베이스 관리 프로그램이 파티션 내 병렬 처리를 사용할 수 있는지 여부를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

NO (0) [SYSTEM (-1), NO (0), YES (1)]

-1 값은 데이터베이스 관리 프로그램을 실행 중인 하드웨어에 따라 매개변수 값을 『YES』 또는 『NO』로 설정합니다.

이 매개변수가 ″YES″인 경우 병렬 성능 향상을 활용할 수 있는 일부 조작은 데이터베이스 쿼리 및 인덱스 작성입니다.

주:

- 병렬 인덱스 작성에서는 이 구성 매개변수를 사용하지 않습니다.
- 이 매개변수 값을 변경하면 패키지가 데이터베이스에 리바인드되고 일부 성능 저하가 발생할 수 있습니다.

java_heap_sz - 최대 Java 인터프리터 힙 크기

이 매개변수는 Java DB2 스토어드 프로시저 및 UDF를 서비스하기 위해 시작된 Java 인터프리터에서 사용하는 힙의 최대 *크기*를 판별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

HP-UX

4096 [0 - 524 288]

기타 모든 운영 체제

2048 [0 - 524 288]

수치 단위

페이지(4KB)

할당 시기

Java 스토어드 프로시저 또는 UDF 시작 시

사용 가능한 시기

db2fmp 프로세스(분리) 또는 db2agent 프로세스(트러스트된) 종료 시.

각 DB2 프로세스마다 하나의 힙이 있습니다(Linux 및 UNIX 플랫폼의 각 에이전트 또는 서브에이전트마다 하나와 다른 플랫폼의 각 인스턴스마다 하나). 각 분리(fenced) UDF 및 분리 스토어드 프로시저 프로세스마다 하나가 있습니다. 트러스트된 루틴의 경우 에이전트(서브에이전트는 제외) 당 하나의 힙이 있습니다. Java 스토어드 프로시저를 실행 중인 db2fmp 프로세스마다 하나의 힙이 있습니다. 멀티스레드 db2fmp 프로세스의 경우 스레드 안전 분리(fenced) 루틴을 사용 중인 여러 응용프로그램은 단일 힙에서 서비스됩니다. 모든 경우에 Java UDF 또는 스토어드 프로시저를 실행하는 에이전트 또는 프로세스만 이 메모리를 할당합니다. 파티션된 데이터베이스 시스템에서는 각 데이터베이스 파티션에서 동일한 값을 사용합니다.

XML 데이터는 스토어드 프로시저에 IN, OUT 또는 INOUT 매개변수로 전달되는 경우 구체화됩니다. Java 스토어드 프로시저를 사용 중인 경우 XML 인수의 양 및 크기와 동시에 실행 중인 외부 스토어드 프로시저의 수에 따라 늘려야 할 수도 있습니다.

jdk_path - Java용 SDK(Software Developer's Kit) 설치 경로

이 매개변수는 Java 스토어드 프로시저 및 사용자 정의 함수를 실행하는 데 사용할 Java용 SDK(Software Developer's Kit)가 설치되는 디렉토리를 지정합니다. Java 인터프리터에서 사용하는 CLASSPATH 및 기타 환경 변수는 이 매개변수의 값에서 계산합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 경로]

Java용 SDK가 DB2 제품과 함께 설치된 경우 이 매개변수가 올바로 설정되어 있습니다. 그러나 데이터베이스 관리 프로그램(dbm cfg) 매개변수를 재설정하는 경우 Java용 SDK가 설치되는 위치를 지정해야 합니다.

keepfenced - 분리(fenced) 프로세스 보존

이 매개변수는 분리 모드 루틴 호출이 완료된 후 분리 모드 프로세스의 보존 여부를 표시합니다. 분리 모드 프로세스는 데이터베이스 관리 프로그램 에이전트 프로세스에서 사용자 작성 분리 모드 코드를 분리하기 위해 별도의 시스템 엔티티로 작성됩니다. 이 매개변수는 데이터베이스 서버에만 적용됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Yes [Yes; No]

*keepfenced*가 *no*로 설정되었으며 실행 중인 루틴이 스레드 안전하지 않은 경우 각 분리 모드 호출마다 새 분리 모드 프로세스가 작성되고 파괴됩니다. *keepfenced*가 *no*로 설정되었으며 실행 중인 루틴이 스레드 안전한 경우 분리 모드 프로세스가 지속되지만 호출에 대해 작성된 스레드가 종료됩니다. *keepfenced*가 *yes*로 설정된 경우 분리 모드 프로세스 또는 스레드를 후속 분리 모드 호출에 재사용합니다. 데이터베이스 관리 프로그램이 중지된 경우 모든 미결 분리 모드 프로세스 및 스레드가 종료됩니다.

이 매개변수를 *yes*로 설정하면 최대 *fenced_pool* 매개변수에 있는 값까지 활성화된 각 분리 모드 프로세스마다 데이터베이스 관리 프로그램이 추가 시스템 자원을 사용하게 됩니다. 새 프로세스는 기존의 분리 모드 프로세스가 후속 분리 루틴 호출을 처리할 수 없는 경우에만 작성됩니다. *fenced_pool*이 0으로 설정된 경우 이 매개변수는 무시됩니다.

권장사항: 비분리 모드 요청 수에 비해 분리 모드 요청 수가 크고 시스템 자원이 제한되지 않은 환경에서는 이 매개변수를 *yes*로 설정할 수 있습니다. 기존 분리 모드 프로세스를 사용하여 호출을 처리하므로 초기 분리 모드 프로세스 작성 오버헤드를 방지하여 분리 모드 프로세스 성능이 향상됩니다. 특히 Java 루틴의 경우 Java Virtual Machine(JVM)을 시작하는 비용이 절약되어 성능이 상당히 향상됩니다.

예를 들어, OLTP 인출/예금 은행 거래 응용프로그램에서 각 거래를 수행하는 코드를 분리 모드 프로세스에서 실행되는 스토어드 프로시저에서 수행할 수 있습니다. 이 응용프로그램에서 기본 워크로드는 분리 모드 프로세스로부터 수행됩니다. 이 매개변수가 *no*로 설정된 경우 각 거래로 새 분리 모드 프로세스를 작성하는 오버헤드가 발생하므로 성능이 상당히 저하됩니다. 그러나 이 매개변수가 *yes*로 설정된 경우 각 거래는 기존의 분리 모드 프로세스를 사용하려고 시도하므로 이 오버헤드가 방지됩니다.

이전 DB2 버전에서 이 매개변수는 *keepdari*입니다.

local_gssplugin - 로컬 인스턴스 레벨 권한 부여에 사용되는 GSS API 플러그인

이 매개변수는 *authentication* 데이터베이스 관리 프로그램 구성 매개변수의 값이 GSSPLUGIN 또는 GSS_SERVER_ENCRYPT로 설정된 경우 인스턴스 레벨 로컬 권한 부여에 사용할 디폴트 GSS API 플러그인 라이브러리의 이름을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 문자열]

max_connections - 클라이언트 연결의 최대수

이 매개변수는 데이터베이스 파티션당 허용되는 클라이언트 연결의 최대수를 표시합니다.

구성 유형

데이터베이스 관리 프로그램

매개변수 유형

온라인으로 구성 가능

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 또는 리모트 클라이언트가 있는 데이터베이스 서버 또는 연결 서버 (*max_connections*, *max_coordagents*, *num_initagents*, *num_poolagents* 및 페더레이티드 환경을 사용 중인 경우 *federated_async*)

디폴트 [범위]

-1 및 AUTOMATIC(*max_coordagents*) [-1 및 AUTOMATIC; 1 - 64 000]

-1로 설정하면 자동 설정 또는 동작이 아니라 *max_coordagents*에 연결된 값이 사용됨을 의미합니다. AUTOMATIC은 데이터베이스 관리 프로그램이 최상의 기술을 사용하여 이 매개변수의 값을 선택함을 의미합니다. AUTOMATIC

은 구성 파일에 있는 ON/OFF 스위치이며 값에 대해 독립적입니다. 그러므로 -1과 AUTOMATIC은 둘 다 디폴트 설정이 될 수 있습니다.

자세한 내용은 700 페이지의 『max_coordagents 및 max_connections를 구성하는 경우 제한사항 및 동작』을 참조하십시오.

집중기

*max_connections*가 *max_coordagents*보다 작거나 같은 경우 집중기는 OFF입니다. *max_connections*가 *max_coordagents*보다 큰 경우 집중기는 ON입니다.

이 매개변수는 인스턴스의 데이터베이스 파티션에 연결할 수 있는 클라이언트 응용프로그램의 최대수를 제어합니다. 일반적으로 각 응용프로그램에는 코디네이터 에이전트가 지정됩니다. 에이전트를 사용하면 응용프로그램과 데이터베이스 간의 조작이 쉬워집니다. 이 매개변수의 디폴트값이 사용되는 경우 집중기 기능은 활성화되지 않습니다. 따라서 각 에이전트는 자신의 전용 메모리 내에서 작동하고 데이터베이스 관리 프로그램 및 전역 자원(예: 버퍼 풀)을 다른 에이전트와 공유합니다. 이 매개변수가 디폴트보다 큰 값으로 설정되는 경우에는 집중기 기능이 활성화됩니다.

max_connretries - 노드 연결 재시도 수

이 매개변수는 두 개의 데이터베이스 파티션 서버들 간 TCP/IP 연결을 설정하는 최대 연결 시도 수를 지정합니다.

구성 유형
데이터베이스 관리 프로그램

적용 대상
로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형
온라인으로 구성 가능

전파 클래스
즉시

디폴트 [범위]
5 [0–100]

두 개의 데이터베이스 파티션 서버들 간 통신 설정 시도가 실패한 경우(예: *conn_elapse* 매개변수에 지정된 값과 같아짐) *max_connretries*는 데이터베이스 파티션 서버에 대한 연결 재시도 수를 지정합니다. 이 매개변수에 지정된 값을 초과한 경우 오류가 리턴됩니다.

max_coordagents - 최대 코디네이팅 에이전트 수

이 매개변수는 코디네이팅 에이전트의 수를 제한하는 데 사용됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

200, Automatic [-1; 0–64 000]

-1 설정은 200 값으로 변환됩니다.

자세한 내용은 700 페이지의 『max_coordagents 및 max_connections를 구성하는 경우 제한사항 및 동작』을 참조하십시오.

집중기

집중기가 OFF인 경우 즉, *max_connections*가 *max_coordagents* 이하인 경우 이 매개변수는 서버 노드에서 한 번에 존재할 수 있는 최대 코디네이팅 에이전트 수를 판별합니다.

데이터베이스로 연결되거나 인스턴스로 접속되는 각 로컬 또는 리모트 응용프로그램마다 하나의 코디네이팅 에이전트를 획득합니다. 인스턴스 접속이 필요한 요청은 CREATE DATABASE, DROP DATABASE 및 데이터베이스 시스템 모니터 명령입니다.

집중기가 ON인 경우(*max_connections*가 *max_coordagents*보다 큰 경우) 코디네이터 에이전트가 서비스할 수 있는 것보다 많은 연결이 있을 수 있습니다. 서비스하는 코디네이터 에이전트가 있는 경우에만 응용프로그램이 활성 상태입니다. 그렇지 않으면 응용프로그램은 비활성 상태입니다. 활성 응용프로그램의 요청은 데이터베이스 코디네이터 에이전트(SMP 또는 MPP 구성에서는 서브에이전트도 포함)에서 서비스합니다. 비활성 응용프로그램의 요청은 응용프로그램이 활성화되면 응용프로그램을 서비스하기 위해 데이터베이스 코디네이터 에이전트가 지정될 때까지 큐에 대기됩니다. 따라서 시스템의 로드를 제어할 때 이 매개변수를 사용할 수 있습니다.

max_querydegree - 병렬 처리의 최대 쿼리 수준

이 매개변수는 이 데이터베이스 관리 프로그램 인스턴스에서 실행 중인 SQL문에 사용되는 파티션 내 병렬 처리의 최대 수준을 지정합니다. SQL문은 명령문 실행 시 데이터베이스 파티션에서 이 병렬 조작 수를 초과하여 사용하지 않습니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

명령문 경계

디폴트 [범위]

-1 (ANY) [ANY, 1 - 32 767] (ANY는 시스템이 판별함을 의미함)

데이터베이스 파티션이 SQL문에 대해 파티션 내 병렬 처리를 사용할 수 있도록 하려면 *intra_parallel* 구성 매개변수를 『YES』로 설정해야 합니다. 병렬 인덱스 작성 시 더 이상 *intra_parallel* 매개변수가 필요하지 않습니다.

이 구성 매개변수의 디폴트값은 -1입니다. 이 값은 시스템이 옵티마이저가 판별한 병렬 처리 수준을 사용함을 의미합니다. 그렇지 않으면 사용자 지정 값을 사용합니다.

주: CURRENT DEGREE 특수 레지스터 또는 DEGREE 바인드 옵션을 사용하여 명령문 컴파일 시 SQL문의 병렬 처리 수준을 지정할 수 있습니다.

활성 응용프로그램에 대한 병렬 처리의 최대 쿼리 수준은 SET RUNTIME DEGREE 명령을 사용하여 수정할 수 있습니다. 사용되는 실제 런타임 수준은 다음의 값 중 낮은 값입니다.

- *max_querydegree* 구성 매개변수
- 응용프로그램 런타임 등급
- SQL문 컴파일 등급

이 구성 매개변수는 쿼리에만 적용됩니다.

max_time_diff - 노드 간 최대 시간 차이

이 매개변수는 노드 구성 파일에 나열된 데이터베이스 파티션 서버 간에 허용되는 최대 시간 차이를 분 단위로 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

60 [1 - 1 440]

수치 단위

분

각 데이터베이스 파티션 서버에는 자신의 시스템 클럭이 있습니다. 둘 이상의 데이터베이스 파티션 서버가 트랜잭션과 연관되어 있고 이들 서버 클럭 간의 시간 차이가 MAX_TIME_DIFF 매개변수에서 지정한 시간보다 클 경우 트랜잭션이 거부되고 SQLCODE가 리턴됩니다. (트랜잭션은 데이터 수정이 연결된 경우에만 거부됩니다.)

데이터베이스가 파티션된 환경에서도 SQLCODE가 리턴됩니다. 이 환경에서는 DB2가 시스템 클럭을 SQLOGCTL.LFH 로그 제어 파일에 저장된 가상 시간소인(VTS)과 비교합니다. .LFH 파일의 시간소인이 시스템 시간보다 작은 경우 시스템 클럭이 이 시간과 일치할 때까지 데이터베이스 로그의 시간이 VTS로 설정됩니다. 다중 노드 간의 시간 차이가 MAX_TIME_DIFF 매개변수보다 작지만 SQL1473N 오류 메시지도 리턴됩니다.

DB2는 *세계 표준시*(UTC)를 사용하므로 이 매개변수를 설정할 때 다른 시간대는 고려사항이 아닙니다. 세계 표준시는 그리니치 평균시와 같습니다.

maxagents - 최대 에이전트 수

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서는 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 DB2 버전 9.5 데이터베이스 관리 프로그램에서는 사용되지 않습니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에만 적용됩니다.

이 매개변수는 응용프로그램 요청을 승인하기 위해 지정된 시간에 사용 가능한 데이터베이스 관리 프로그램 에이전트(코디네이터 에이전트 또는 서브에이전트에 관계없이)의 최대 수를 표시합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

200 [1 - 64 000]

로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버에서는 400® [1 - 64 000]

수치 단위

카운터

코디네이팅 에이전트의 수를 제한하려면 *max_coordagents* 매개변수를 사용하십시오.

각 추가 에이전트는 추가 메모리가 필요하므로 데이터베이스 관리 프로그램의 전체 메모리 사용을 제한하기 위해 메모리 제한 환경에서 이 매개변수가 유용할 수 있습니다.

권장사항: *maxagents*의 값은 동시에 액세스할 수 있는 각 데이터베이스에서 최소한 *maxappls* 값의 합계여야 합니다. 데이터베이스 수가 *numdb* 매개변수보다 큰 경우 *maxappls*의 최대값으로 *numdb*의 결과를 사용하는 것이 가장 안전합니다.

각 추가 에이전트는 데이터베이스 관리 프로그램 시작 시 할당되는 자원 오버헤드가 필요합니다.

데이터베이스에 연결하려고 시도할 때 메모리 오류가 발생한 경우 다음과 같이 구성을 조정하십시오.

- 쿼리간 병렬 처리가 사용 불가능한 파티션되지 않은 데이터베이스 환경에서 *maxagents* 데이터베이스 구성 매개변수의 값을 늘리십시오.
- 파티션된 데이터베이스 환경 또는 쿼리간 병렬 처리가 사용 가능한 환경에서 *maxagents* 또는 *max_coordagents* 중 큰 값을 늘리십시오.

maxcagents - 최대 동시 에이전트 수

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 DB2 버전 9.5 데이터베이스 관리 프로그램에서는 사용되지 않습니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에만 적용됩니다.

이 매개변수를 사용하면 데이터베이스 관리 프로그램 트랜잭션을 동시에 실행할 수 있는 최대 데이터베이스 관리 프로그램을 제한하여 동시 응용프로그램 활동이 많은 시간 동안 시스템의 로드를 제어할 수 있습니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

-1 (*max_coordagents*) [-1; 1 – *max_coordagents*]

수치 단위

카운터

이 매개변수는 데이터베이스에 연결할 수 있는 응용프로그램 수를 제한하지 않습니다. 한 번에 데이터베이스 관리 프로그램이 동시에 처리할 수 있는 데이터베이스 관리 프로그램 에이전트 수만 제한하므로 최대 처리 시간 동안 시스템 자원 사용이 제한됩니다. 예를 들어, 많은 연결이 필요하지만 이러한 연결에 사용할 수 있는 메모리의 양이 제한된 시스템이 있을 수 있습니다. 이 매개변수를 조정하면 이러한 환경에서 유용할 수 있습니다. 동시 활동이 많은 기간이 지속되면 초과 운영 체제 페이징이 발생할 수 있습니다.

-1 값은 한계가 *max_coordagents*임을 표시합니다.

권장사항: 대부분의 경우 이 매개변수의 디폴트값을 승인할 수 없습니다. 응용프로그램의 동시성이 높아서 문제가 발생한 경우 벤치마크 테스팅을 사용하여 이 매개변수를 조정하면 데이터베이스의 성능을 최적화할 수 있습니다.

mon_heap_sz - 데이터베이스 시스템 모니터 힙 크기

이 매개변수는 데이터베이스 시스템 모니터 데이터에 할당할 메모리의 양(페이지)을 판별합니다. 스냅샷 작성, 모니터 스위치 작동, 모니터 재설정 또는 이벤트 모니터 활성화와 같은 데이터베이스 모니터링 활동을 수행하면 모니터 힙에서 메모리가 할당됩니다.

버전 9.5에서 이 데이터베이스 구성 매개변수의 디폴트값은 AUTOMATIC이며 *instance_memory* 한계에 접근할 때까지 필요에 따라 모니터 힙이 증가할 수 있음을 의미합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

Automatic [0 - 60 000]

수치 단위

페이지(4KB)

할당 시기

db2start 명령을 사용하여 데이터베이스 관리 프로그램이 시작될 때

사용 가능한 시기

db2stop 명령을 사용하여 데이터베이스 관리 프로그램이 중지될 때

영(0) 값을 사용하면 데이터베이스 관리 프로그램이 데이터베이스 시스템 모니터 데이터를 수집하지 못합니다.

권장사항: 모니터링 활동에 필요한 메모리 양은 모니터링 응용프로그램(스냅샷 또는 이벤트 모니터를 작성하는 응용프로그램) 수, 설정된 스위치 및 데이터베이스 활동 레벨에 따라 다릅니다.

이 힙에 구성된 메모리를 모두 사용했으며 인스턴스 공유 메모리 영역에 예약되지 않은 메모리가 더 이상 없는 경우 다음 중 하나가 발생합니다.

- 첫 번째 응용프로그램이 이 이벤트 모니터가 정의된 데이터베이스에 연결할 때 관리 통지 로그에 오류 메시지가 기록됩니다.
- SET EVENT MONITOR문을 사용하여 동적으로 시작되는 이벤트 모니터가 실패한 경우 응용프로그램으로 오류 코드가 리턴됩니다.
- 모니터 명령 또는 API 서브루틴이 실패한 경우 응용프로그램으로 오류 코드가 리턴됩니다.

nodetype - 머신 노드 유형

이 매개변수는 머신에 설치한 DB2 제품에 대한 정보와 데이터베이스 관리 프로그램 구성 유형에 대한 정보를 제공합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

정보용

다음은 이 매개변수가 리턴할 수 있는 값과 해당 노드 유형과 연관된 제품입니다.

- **로컬 및 리모트 클라이언트가 있는 데이터베이스 서버** – 로컬 및 리모트 Data Server Runtime Client를 지원하고 다른 리모트 데이터베이스 서버에 액세스할 수 있는 DB2 서버 제품입니다.
- **클라이언트** – 리모트 데이터베이스 서버에 액세스할 수 있는 Data Server Runtime Client입니다.
- **로컬 클라이언트가 있는 데이터베이스 서버** – 로컬 Data Server Runtime Client를 지원하며 다른 리모트 데이터베이스 서버에 액세스할 수 있는 DB2 관계형 데이터베이스 관리 시스템입니다.
- **로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버** – 로컬 및 리모트 Data Server Runtime Client를 지원하고 다른 리모트 데이터베이스 서버에 액세스할 수 있으며 병렬 처리가 가능한 DB2 서버 제품입니다.

notifylevel - 통지 레벨

이 매개변수는 관리 통지 로그에 기록되는 관리 통지 메시지의 유형을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

3 [0 — 4]

Linux 및 UNIX 플랫폼에서 관리 통지 로그는 *instance*.nfy라는 텍스트 파일입니다. Windows에서 모든 관리 통지 메시지는 이벤트 로그에 기록됩니다. 오류는 DB2, Health Monitor, Capture 및 Apply 프로그램, 사용자 응용프로그램이 기록할 수 있습니다.

이 매개변수에 유효한 값은 다음과 같습니다.

- **0** — 알림 통지 메시지를 캡처하지 않음. (이 설정은 권장하지 않습니다.)
- **1** — 치명적 또는 복구 불가능 오류. 치명적이고 복구 불가능한 오류만 로그됩니다. 이 조건 중 일부에서 복구하려면 DB2 서비스의 도움이 필요합니다.

- **2** — 즉시 조치 필요. 시스템 관리자 또는 데이터베이스 관리자가 즉시 주의해야 하는 조건이 로그됩니다. 조건이 해결되지 않으면 치명적 오류가 발생할 수 있습니다. 매우 중요하고 오류 없는 활동(예: 복구)의 통지도 이 레벨에서 로그될 수 있습니다. 이 레벨은 Health Monitor 알람을 캡처합니다.
- **3** — 중요 정보, 즉시 조치 불필요. 처리되지 않으며 즉시 조치할 필요가 없으나 최적이 아닌 시스템을 표시하는 조건이 로그됩니다. 이 레벨은 Health Monitor 알람, Health Monitor 경고 및 Health Monitor 주의를 캡처합니다.
- **4** — 정보 메시지.

관리 통지 로그에는 기준이 되는 값이 있고 *notifylevel*의 값을 포함하는 메시지가 있습니다. 예를 들어, *notifylevel*을 3으로 설정하면 관리 통지 로그에는 1, 2 및 3 레벨에 적용 가능한 메시지가 포함됩니다.

사용자 응용프로그램이 통지 파일 또는 Windows 이벤트 로그에 쓰려면 db2AdminMsgWrite API를 호출해야 합니다.

권장사항: 이 매개변수의 값을 늘려서 문제를 해결하는 데 도움이 되는 추가 문제점 판별 데이터를 수집할 수 있습니다. Health Monitor가 구성에서 정의된 문의처로 통지를 보내도록 하려면 *notifylevel*을 2 이상으로 설정해야 합니다.

num_initagents - 풀의 초기 에이전트 수

이 매개변수는 DB2START 시에 에이전트 풀에서 작성되는 초기 유휴 에이전트 수를 판별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

0 [0–64 000]

시작 중 이 매개변수의 값이 *num_poolagents* 보다 크고 *num_poolagents*가 AUTOMATIC으로 설정되지 않은 경우를 제외하고 데이터베이스 관리 프로그램은 항상 *num_initagents* 유휴 에이전트를 db2start 명령의 일부분으로 시작합니다. 이 경우 풀링 가능한 것보다 많은 유휴 에이전트를 시작할 이유가 없으므로 데이터베이스 관리 프로그램은 num_poolagents개의 유휴 에이전트만 시작합니다.

num_initfenced - 분리(fenced) 프로세스의 초기 수

이 매개변수는 START DBM 시간에 db2fmp 풀에서 작성된 스레드되지 않은 유휴 db2fmp 프로세스의 초기 수를 표시합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

0 [0–64 000]

이 매개변수를 설정하면 비스레드 안전 C 및 Cobol 루틴 실행의 초기 시작 시간이 줄어듭니다. *keepfenced*가 지정되지 않은 경우 이 매개변수는 무시됩니다.

*fenced_pool*을 시스템에 알맞은 크기로 설정하는 것이 START DBM 시간에 db2fmp의 수를 시작하는 것보다 훨씬 중요합니다.

이전 버전에서 이 매개변수는 *num_initdaris*였습니다.

num_poolagents - 에이전트 풀 크기

이 매개변수는 유휴 에이전트 풀의 최대 크기를 설정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 100, Automatic [-1, 0–64 000]

이 구성 매개변수는 디폴트로 100 값을 사용하여 AUTOMATIC으로 설정됩니다. -1 설정은 계속 지원되며 100 값으로 변환됩니다. 이 매개변수가 AUTOMATIC으로 설정된 경우 데이터베이스 관리 프로그램은 풀링할 유휴 에이전트 수를 자동으로 관리합니

다. 일반적으로 에이전트가 작업을 완료한 후 종료되지 않고 일정 기간 동안 유휴 상태가 됨을 의미합니다. 에이전트의 워크로드 및 유형에 따라 일정 시간 이후에 종료될 수 있습니다.

AUTOMATIC을 사용하는 경우 계속 num_poolagents 구성 매개변수의 값을 지정할 수 있습니다. 현재 풀링된 유휴 에이전트 수가 지정한 값 이하인 경우 항상 추가 유휴 에이전트가 풀링됩니다.

예:

*num_poolagents*가 **100 및 AUTOMATIC으로 설정됨**
: 에이전트가 사용 가능해지면 유휴 에이전트 풀에 추가되며 일정 시점에 데이터베이스 관리 프로그램이 에이전트를 종료해야 하는지 여부를 평가합니다. 데이터베이스 관리 프로그램이 에이전트 종료를 고려 중인 경우 전체 유휴 에이전트 수가 100보다 크면 이 에이전트가 종료됩니다. 100개 미만의 유휴 에이전트가 있는 경우 유휴 에이전트는 계속 작업 대기 중입니다. AUTOMATIC 설정을 사용하면 100을 초과하는 추가 유휴 에이전트가 풀링되므로 작업의 빈도가 크게 불안정한 경우 시스템 활동이 많아졌을 때 유용할 수 있습니다. 지정된 시간에 100개 미만의 유휴 에이전트가 있을 수 있는 경우 에이전트는 반드시 풀링됩니다. 따라서 간단한 시스템 활동만 지속되는 경우 새 작업의 시작 비용이 줄어들 수 있습니다.

*num_poolagents*가 **동적으로 구성됨**
: 매개변수 값이 풀링된 에이전트 수보다 큰 값으로 증가한 경우 효과는 즉시 나타납니다. 새 에이전트가 유휴 상태가 되면 풀링됩니다. 매개변수 값이 감소된 경우 데이터베이스 관리 프로그램은 풀에서 에이전트의 수를 즉시 줄이지 않습니다. 대신 풀 크기는 그대로 유지되고 에이전트가 사용될 때 종료되고 다시 유휴 상태가 됩니다. 즉, 풀의 에이전트 수가 새 한계로 점차 줄어듭니다.

권장사항: 대부분의 환경에서 디폴트 영(0) 및 AUTOMATIC이면 충분합니다. 너무 많은 에이전트가 작성 및 종료된다고 판단되는 특정 워크로드가 있는 경우 AUTOMATIC으로 설정된 매개변수를 그대로 두고 *num_poolagents*의 값을 늘릴 수 있습니다.

numdb - 호스트 및 System i 데이터베이스를 포함한 최대 동시 활성 데이터베이스 수

이 매개변수는 동시에 활성화할 수 있는(즉, 응용프로그램을 연결함) 로컬 데이터베이스 수 또는 DB2 Connect 서버에서 카탈로그화할 수 있는 여러 가지 데이터베이스 별명의 최대 수를 지정합니다.

구성 유형
: 데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

UNIX 8 [1 — 256]

로컬 및 리모트 클라이언트가 있는 Windows 데이터베이스 서버

8 [1 — 256]

로컬 클라이언트가 있는 Windows 데이터베이스 서버

3 [1 — 256]

수치 단위

카운터

각 데이터베이스가 스토리지를 차지하며 활성 데이터베이스는 새 공유 메모리 세그먼트를 사용합니다.

권장사항: 일반적으로 이 값을 데이터베이스 관리 프로그램에 이미 정의된 실제 데이터베이스 수로 설정하고 확장을 예상하여 이 값에 약 10%를 추가하면 가장 적합합니다.

numdb 매개변수를 변경하면 할당되는 총 메모리 양이 달라질 수 있습니다. 따라서 이 매개변수를 자주 갱신하지 않는 것이 좋습니다. 이 매개변수를 갱신할 때에는 데이터베이스 또는 데이터베이스에 연결된 응용프로그램에 메모리를 할당할 수 있는 다른 구성 매개변수를 고려해야 합니다.

query_heap_sz - 쿼리 힙 크기

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 DB2 버전 9.5 데이터베이스 관리 프로그램에서는 사용되지 않습니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에만 적용됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

1 000 [2 - 524 288]

수치 단위

페이지(4KB)

할당 시기

응용프로그램(로컬 또는 리모트)이 데이터베이스에 연결될 때

사용 가능한 시기

응용프로그램의 연결이 데이터베이스에서 끊어졌거나 인스턴스에서 접속 해제될 때

이 매개변수는 쿼리 힙에 할당할 수 있는 **최대** 메모리 양을 지정하므로 응용프로그램 이 에이전트에서 불필요한 대량의 가상 메모리를 사용하지 않습니다.

쿼리 힙은 에이전트의 전용 메모리에 각 쿼리를 저장하는 데 사용됩니다. 각 쿼리에 대한 정보는 입력 및 출력 SQLDA, 명령문 텍스트, SQLCA, 패키지 이름, 작성자, 섹션 번호 및 일관성 토큰으로 구성되어 있습니다.

쿼리 힙은 블로킹 커서에 할당되는 메모리에도 사용됩니다. 이 메모리는 커서 제어 블록 및 완전히 분석된 출력 SQLDA로 구성되어 있습니다.

할당된 *초기* 쿼리 힙의 *크기*는 *aslheapsz* 매개변수에 지정된 응용프로그램 지원 계층 힙과 동일합니다. 쿼리 힙 *크기*는 2보다 *크거나* 같아야 하며 *aslheapsz* 매개변수보다 *크거나* 같아야 합니다. 이 쿼리 힙의 *크기*가 지정된 요청을 처리하기에 충분하지 않은 경우 요청에 필요한 *크기*로 재할당됩니다(*query_heap_sz*를 초과하지 않음). 이 새 쿼리 힙이 *aslheapsz*보다 1.5배 큰 경우 쿼리가 종료되면 *aslheapsz*의 *크기*로 쿼리 힙이 재할당됩니다.

권장사항: 대부분의 경우 디폴트값이면 충분합니다. 최소값으로 *query_heap_sz*를 최소한 *aslheapsz*보다 큰 값으로 설정해야 합니다. 이 경우 쿼리가 *aslheapsz*보다 클 수 있으며 지정된 시간에 세 개 또는 네 개의 블로킹 커서를 열도록 추가 메모리가 제공됩니다.

매우 큰 LOB가 있는 경우 쿼리 힙의 *크기*가 이러한 LOB를 수용할 수 있도록 이 매개변수의 값을 늘려야 합니다.

release - 구성 파일 릴리스 레벨

이 매개변수는 구성 파일의 릴리스 레벨을 지정합니다.

구성 유형

데이터베이스 관리 프로그램, 데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

정보용

resync_interval - 트랜잭션 재동기화 간격

이 매개변수는 트랜잭션 관리 프로그램(TM), 자원 관리 프로그램(RM) 또는 동기점 관리 프로그램(SPM)이 TM, RM 또는 SPM에서 발견된 미결 인다우트(Indoubt) 트랜잭션의 복구를 재시도해야 하는 시간 간격(초)을 지정합니다. 분산 작업 단위(DUOW) 환경에서 실행 중인 트랜잭션이 있는 경우 이 매개변수가 적용됩니다. 이 매개변수는 페더레이티드 데이터베이스 시스템의 복구에도 적용됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

180 [1 - 60 000]

수치 단위

초

권장사항: 사용자의 환경에서 인다우트(Indoubt) 트랜잭션이 데이터베이스에 대한 다른 트랜잭션을 방해하지 않는 경우 이 매개변수의 값을 늘릴 수 있습니다. DB2 Connect 게이트웨이를 사용하여 DRDA2 응용프로그램 서버에 액세스하는 경우 로컬 데이터 액세스에 대한 방해가 없어도 응용프로그램 서버에서 인다우트(Indoubt) 트랜잭션이 미칠 수 있는 영향을 고려해야 합니다. 인다우트 트랜잭션이 없는 경우 성능 영향은 최소화됩니다.

rqrioblk - 클라이언트 I/O 블록 크기

이 매개변수는 리모트 응용프로그램과 데이터베이스 서버에서 해당 데이터베이스 에이전트 간 통신 버퍼의 *크기*를 지정합니다. 이 매개변수는 블로킹 커서가 열려 있을 때 Data Server Runtime Client에서 입출력 블록 *크기*를 판별하는 데에도 사용됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

32 767 [4 096 - 65 535]

수치 단위

바이트

할당 시기

- 리모트 클라이언트 응용프로그램이 서버 데이터베이스의 연결 요청을 발행할 때
- 블로킹 커서가 열려 있는 경우 클라이언트에서 추가 블록이 열립니다.

사용 가능한 시기

- 서버 데이터베이스에서 리모트 응용프로그램의 연결이 끊어졌을 때
- 블로킹 커서가 닫혔을 때

Data Server Runtime Client가 리모트 데이터베이스에 연결하도록 요청한 경우 이 통신 버퍼가 클라이언트에 할당됩니다. 데이터베이스 서버에서는 연결이 설정되고 서버가 클라이언트에서 *rqrioblk*의 값을 판별할 수 있을 때까지 처음에 32 767바이트의 통신 버퍼가 할당됩니다. 서버가 이 값을 아는 경우 클라이언트의 버퍼가 32 767바이트가 아니면 통신 버퍼를 재할당합니다.

블로킹 커서의 메모리는 응용프로그램의 전용 어드레스 스페이스로부터 할당되므로 각 응용프로그램에 할당할 최적의 전용 메모리 양을 판별해야 합니다. Data Server Runtime Client가 응용프로그램의 전용 메모리로부터 블로킹 커서의 스페이스를 할당할 수 없는 경우 비블로킹 커서가 열립니다.

권장사항: 비블로킹 커서의 경우 단일 쿼리 명령문이 전송할 데이터(예: 대형 오브젝트 데이터)가 너무 커서 디폴트값으로 충분하지 않으면 이 매개변수의 값을 늘립니다.

또한 블로킹 커서의 수 및 잠재적 *크기*에 대한 이 매개변수의 영향을 고려해야 합니다. 전송 중인 행의 수 또는 *크기*가 큰 경우(예: 데이터의 양이 4 096바이트보다 큰 경우)

큰 행 블록으로 성능이 향상될 수 있습니다. 그러나 큰 레코드 블록으로 각 연결의 작업 세트 메모리 크기가 커진다는 점에서 트래이드 오프가 있습니다.

또한 큰 레코드 블록으로 응용프로그램에 실제로 필요한 것보다 많은 페치 요청이 발생할 수 있습니다. 응용프로그램에서 SELECT문에 OPTIMIZE FOR절을 사용하여 페치 요청 수를 제어할 수 있습니다.

sheapthres - 정렬 힙 임계값

이 매개변수는 지정된 시간에 개인용 정렬이 사용할 수 있는 총 메모리 양에 대한 인스턴스 전체에 해당하는 소프트 한계입니다. 인스턴스의 총 개인용 정렬 메모리 사용량이 이 한계에 접근하면 추가 수신 개인용 정렬 요청을 위해 할당된 메모리가 상당히 줄어듭니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버
- OLAP 기능

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

UNIX 32비트 플랫폼

0 [0 - 2 097 152]

Windows 32비트 플랫폼

0 [0 - 2 097 152]

64비트 플랫폼

0 [0 - 2 147 483 647]

수치 단위

페이지(4KB)

정렬 힙을 사용하는 조작의 예로 정렬, 해시 조인, 동적 비트맵(인덱스 ANDing 및 스타 조인에 사용됨) 및 메모리의 테이블 조작이 있습니다.

임계값을 명시적으로 정의하면 데이터베이스 관리 프로그램이 여러 정렬에 메모리의 양을 초과로 사용할 수 없습니다.

파티션되지 않은 데이터베이스 환경에서 파티션된 데이터베이스 환경으로 이동할 때 이 매개변수의 값을 늘릴 필요가 없습니다. 단일 데이터베이스 파티션 환경에서 데이터베이스 및 데이터베이스 관리 프로그램 구성 매개변수를 조정한 후 대부분의 경우 파티션된 데이터베이스 환경에서 동일한 값을 사용해도 문제가 없습니다. 다른 노드 또는 데이터베이스 파티션에서 이 매개변수를 다른 값으로 설정하려면 여러 DB2 인스턴스를 작성하는 방법만 사용해야 합니다. 이 경우 다른 데이터베이스 파티션 그룹에서 다른 DB2 데이터베이스를 관리해야 합니다. 이러한 정리로 파티션된 데이터베이스 환경의 여러 장점은 필요하지 않게 되었습니다.

인스턴스 레벨 *sheapthres*가 0으로 설정된 경우 정렬 메모리 사용의 추적은 데이터베이스 레벨에서만 수행되며 정렬의 메모리 할당은 데이터베이스 레벨 *sheapthres_shr* 구성 매개변수의 값으로 제한됩니다.

데이터베이스 관리 프로그램 구성 매개변수 *sheapthres*가 0으로 설정된 경우에만 *sheapthres_shr*의 자동 성능 조정이 허용됩니다.

다음의 조건 중 일부가 참인 경우 이 매개변수를 동적으로 갱신할 수 없습니다.

- *sheapthres*의 시작 값이 0이고 대상 값이 0이 아닌 값입니다.
- *sheapthres*의 시작 값이 0이 아닌 값이고 대상 값이 0입니다.

권장사항: 원래 이 매개변수는 데이터베이스 관리 프로그램 인스턴스에 있는 최대 *sortheap* 매개변수의 합리적인 배수로 설정해야 합니다. 이 매개변수는 **최소한** 인스턴스의 데이터베이스에 대해 정의된 최대 *sortheap*의 2배여야 합니다.

개인용 정렬을 수행 중이며 시스템의 메모리가 제한되지 않은 경우 이 매개변수의 이상적인 값은 다음 단계를 사용하여 계산할 수 있습니다.

1. 각 데이터베이스의 일반 정렬 힙 사용량을 계산하십시오.

   ```
   (데이터베이스에 대해 실행 중인 일반 동시 에이전트 수)
   * (해당 데이터베이스에 대해 정의된 sortheap)
   ```

2. 위 결과의 합계를 계산하십시오. 인스턴스의 모든 데이터베이스에 대해 일반 상황에서 사용할 수 있는 총 정렬 힙을 제공합니다.

벤치마킹 기술을 사용하여 이 매개변수를 조정하고 정렬 성능과 메모리 사용의 적절한 균형을 찾아야 합니다.

데이터베이스 시스템 모니터를 사용하면 포스트 임계값 정렬(*post_threshold_sorts*) 모니터 요소를 사용하여 정렬 활동을 추적할 수 있습니다.

spm_log_file_sz - 동기점 관리 프로그램 로그 파일 크기

이 매개변수는 동기점 관리 프로그램(SPM) 로그 파일 *크기*를 4KB 페이지 단위로 식별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

256 [4 - 1000]

수치 단위

페이지(4KB)

로그 파일은 `sqllib` 아래의 `spmlog` 서브디렉토리에 있으며 SPM이 처음 시작될 때 작성됩니다.

권장사항: 동기점 관리 프로그램 로그 파일 *크기*는 성능을 유지할 수 있도록 충분히 커야 하며 스페이스를 낭비하지 않도록 충분히 작아야 합니다. 필요한 *크기*는 보호 대화를 사용하는 트랜잭션 수와 COMMIT 또는 ROLLBACK이 발행되는 빈도에 따라 달라집니다.

SPM 로그 파일의 *크기*를 변경하려면 다음을 수행하십시오.

1. LIST DRDA INDOUBT TRANSACTIONS 명령을 사용하여 인다우트(Indoubt) 트랜잭션이 없는지 판별하십시오.
2. 없는 경우, 데이터베이스 관리 프로그램을 중지하십시오.
3. 새 SPM 로그 파일 *크기*로 데이터베이스 관리 프로그램 구성을 갱신하십시오.
4. `$HOME/sqllib` 디렉토리를 찾아가서 `rm -fr spmlog`를 실행하여 현재 SPM 로그를 삭제하십시오. (메모: 이는 AIX 명령을 표시합니다. 다른 시스템에서는 다른 제거 또는 삭제 명령이 필요할 수 있습니다.)
5. 데이터베이스 관리 프로그램을 시작하십시오. 데이터베이스 관리 프로그램을 시작하는 동안 지정된 *크기*의 새 SPM 로그 파일이 작성됩니다.

spm_log_path - 동기점 관리 프로그램 로그 파일 경로

이 매개변수는 동기점 관리 프로그램(SPM) 로그가 기록되는 디렉토리를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

sqllib/spmlog [모든 유효한 경로 또는 디바이스]

디폴트로 로그는 sqllib/spmlog 디렉토리에 기록되며 볼륨이 많은 트랜잭션 환경에서는 입출력 병목 현상이 발생할 수 있습니다. SPM 로그 파일을 현재 sqllib/spmlog 디렉토리가 아닌 고속 디스크에 저장하려면 이 매개변수를 사용하십시오. 이 경우 SPM 에이전트들 간 동시성이 증가합니다.

spm_max_resync - 동기점 관리 프로그램 재동기 에이전트 한계

이 매개변수는 재동기 조작을 동시에 수행할 수 있는 에이전트 수를 식별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

20 [10 — 256]

spm_name - 동기점 관리 프로그램 이름

이 매개변수는 데이터베이스 관리 프로그램에 대한 동기점 관리 프로그램(SPM) 인스턴스를 식별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버

- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 TCP/IP 호스트 이름에서 파생

srvcon_auth - 서버에서 수신 연결의 인증 유형

이 매개변수는 서버에서 수신 연결을 조절하는 경우 사용자 인증이 발생하는 방법 및 위치를 지정합니다. 현재 인증 유형을 겹쳐쓰는 데 사용됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null [CLIENT; SERVER; SERVER_ENCRYPT; KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

값이 지정되지 않으면 DB2는 *authentication* 데이터베이스 관리 프로그램 구성 매개변수의 값을 사용합니다.

각 인증 유형에 대한 설명은 709 페이지의 『authentication - 인증 유형』의 내용을 참조하십시오.

srvcon_gssplugin_list - 서버에서 수신 연결의 GSS API 플러그인 목록

이 매개변수는 데이터베이스 서버에서 지원되는 GSS API 플러그인 라이브러리를 지정합니다. 이 매개변수는 *srvcon_auth* 매개변수가 KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN 또는 GSS_SERVER_ENCRYPT로 지정되거나 *srvcon_auth*가 지정되고 authentication이 KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN 또는 GSS_SERVER_ENCRYPT로 지정된 경우 서버에서 수신 연결을 처리합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 문자열]

디폴트로 이 값은 널(NULL)입니다. 인증 유형이 GSSPLUGIN이고 이 매개변수가 NULL인 경우 오류가 리턴됩니다. 인증 유형이 KERBEROS이고 이 매개변수가 NULL인 경우에는 DB2가 제공하는 Kerberos 모듈 또는 라이브러리가 사용됩니다. 이 매개변수는 다른 인증 유형이 사용되는 경우에는 사용되지 않습니다.

인증 유형이 KERBEROS이고 이 매개변수의 값이 NULL이 아닌 경우 목록에는 단 하나의 Kerberos 플러그인이 있어야 하며 그 플러그인이 인증에 사용됩니다. (목록에 있는 다른 모든 GSS 플러그인은 무시됩니다.) 둘 이상의 Kerberos 플러그인이 있는 경우 오류가 리턴됩니다.

각 GSS API 플러그인 이름은 쉼표(,)로 구분해야 하며 쉼표 앞뒤에는 스페이스가 없어야 합니다. 플러그인 이름은 환경 설정의 순서로 나열되어야 합니다.

srvcon_pw_plugin - 서버에서 수신 연결의 사용자 ID 암호 플러그인

이 매개변수는 서버 측 인증에 사용할 디폴트 사용자 ID 암호 플러그인 라이브러리 이름을 지정합니다. 이 매개변수는 *srvcon_auth* 매개변수가 CLIENT, SERVER, SERVER_ENCRYPT 또는 DATA_ENCRYPT로 지정되거나 *srvcon_auth*가 지정되지 않고 *authentication*이 CLIENT, SERVER, SERVER_ENCRYPT 또는 DATA_ENCRYPT로 지정된 경우 서버에서 수신 연결을 처리합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 문자열]

디폴트로 이 값은 널(NULL)이며 DB2가 제공하는 사용자 ID 암호 플러그인 라이브러리가 사용됩니다. 모든 그룹 찾아보기에 대해 플러그인이 사용됩니다. 루트 서버 설치가 아닌 경우 DB2 사용자 ID와 암호 플러그인 라이브러리가 사용되면 DB2 제품을 사용하기 전에 db2rfe 명령을 실행해야 합니다.

srv_plugin_mode - 서버 플러그인 모드

이 매개변수는 플러그인을 분리 모드로 실행할지 비분리 모드로 실행할지 지정합니다. 비분리 모드만 지원됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

UNFENCED

ssl_cipherspecs - 서버에서 지원되는 암호 스펙 구성 매개변수

이 구성 매개변수는 SSL 프로토콜을 사용하는 경우 서버가 수신 연결 요청에 허용하는 암호 제품을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null [TLS_RSA_WITH_AES_256_CBC_SHA; TLS_RSA_WITH_AES_128_CBC_SHA ; TLS_RSA_WITH_3DES_EDE_CBC_SHA]

TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_128_CBC_SHA 또는 TLS_RSA_WITH_3DES_EDE_CBC_SHA와 같은 다중 암호 스펙을 지정할 수 있습니다. 이들은 쉼표로 구분되어야 하며 쉼표 앞뒤에 스페이스가 없어야 합니다.

SSL 응답 확인 방식에서 널(NULL) 또는 다중 값이 지정되면 클라이언트와 서버는 협상하여 사용할 더 안전한 암호 제품을 찾습니다. 호환 가능 암호 제품을 찾을 수 없는 경우에는 연결이 실패합니다. 차례로 지정해서 암호 제품에 우선순위를 부여할 수는 없습니다.

ssl_clnt_keydb - 클라이언트에서 아웃바운드 SSL 연결의 SSL 키 파일 경로 구성 매개변수

이 구성 매개변수는 클라이언트 측에서 SSL 연결에 사용할 키 파일의 완전한 파일 경로를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null

SSL 키 파일의 디폴트 확장자는 **.kbd**이며 서버 개인용 인증서의 서명자 인증서를 저장합니다. 자체 서명 서버 개인용 인증서의 경우 서명자 인증서는 공용 키입니다. 인증권한 서명된 서버 개인용 인증서의 경우 서명자 인증서는 루트 CA 인증서입니다. 키 파일은 클라이언트가 SSL 응답 확인 방식을 수행하는 동안 서버 개인용 인증서를 확인하기 위해 액세스됩니다.

디폴트로 이 값은 널(NULL)입니다. 응용프로그램 유형에 따라 SSL 연결 요청에 데이터베이스 관리 프로그램 구성 매개변수 **ssl_clnt_keydb**, 연결 문자열 **ssl_clnt_keydb** 또는 db2cli.ini 키워드 **ssl_clnt_keydb**를 사용하여 클라이언트 SSL 키 파일 경로를 지정해야 합니다. 이들을 전혀 지정하지 않으면 SSL 연결은 실패합니다.

ssl_clnt_stash - 클라이언트에서 아웃바운드 SSL 연결의 SSL 숨김 파일 경로 구성 매개변수

이 구성 매개변수는 클라이언트 측에서 SSL 연결에 사용할 숨김 파일의 완전한 파일 경로를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null

SSL 숨김 파일의 디폴트 확장자는 .sth이며 키 데이터베이스 암호의 암호화된 버전을 저장합니다. 숨김 파일에 보유되는 암호는 SSL 연결 요청 시 SSL 키 파일에 액세스하는 데 사용됩니다.

디폴트값은 널(NULL)입니다. 응용프로그램 유형에 따라 SSL 연결 요청에 데이터베이스 관리 프로그램 구성 매개변수 **ssl_clnt_stash**, 연결 문자열 **ssl_clnt_stash** 또는 db2cli.ini 키워드 **ssl_clnt_stash**를 사용하여 클라이언트 SSL 숨김 파일 경로를 지정할 수 있습니다. 이들을 전혀 지정하지 않으면 SSL 연결은 실패합니다.

ssl_svr_keydb - 서버에서 수신 SSL 연결의 SSL 키 파일 경로 구성 매개변수

이 구성 매개변수는 서버 측에서 SSL 설정에 사용할 키 파일의 완전한 파일 경로를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null

SSL 키 파일의 디폴트 확장자는 **.kdb**이며 개인용 인증서, 개인용 인증서 요청 및 서명자 인증서를 저장합니다. 이 키 파일은 인스턴스를 시작하고 SSL 응답 확인 방식을 수행하는 동안 서버 개인용 인증서가 서버 인증을 위해 클라이언트로 전송될 때 액세스됩니다.

디폴트로 이 값은 널(NULL)입니다. 인스턴스를 시작할 때 DB2COMM 레지스트리 변수에 SSL이 있는지 여부를 정의해야 합니다. 그렇지 않으면 인스턴스는 SSL 프로토콜을 지원하지 않고 시작합니다.

ssl_svr_label - 서버에서 수신 SSL 연결의 키 파일에 있는 레이블 구성 매개변수

이 구성 매개변수는 키 데이터베이스에서 서버의 개인용 인증서 레이블을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null

디폴트로 이 값은 널(NULL)입니다. SSL 연결을 설정하는 경우, 이 구성 매개변수로 지정한 서버 인증서가 서버 인증을 위해 클라이언트로 전송됩니다. 이 값이 널(NULL)이면 키 파일에서 정의된 디폴트 인증서가 사용됩니다. 디폴트가 존재하지 않는 경우에는 연결이 실패합니다.

ssl_svr_stash - 서버에서 수신 SSL 연결의 SSL 숨김 파일 경로 구성 매개변수

이 구성 매개변수는 서버 측에서 SSL 설정에 사용할 숨김 파일의 완전한 파일 경로를 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버

- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null

SSL 숨김 파일의 디폴트 확장자는 **.sth**이며 키 데이터베이스 암호의 암호화된 버전을 저장합니다. 숨김 파일에 보유되는 암호는 인스턴스를 시작할 때 SSL 키 파일에 액세스하는 데 사용됩니다.

디폴트로 이 값은 널(NULL)입니다. 인스턴스를 시작할 때 DB2COMM 레지스트리 변수에 SSL이 있는지 여부를 정의해야 합니다. 그렇지 않으면 인스턴스는 SSL 프로토콜을 지원하지 않고 시작합니다.

start_stop_time - 시작 및 중지 시간종료

이 매개변수는 모든 데이터베이스 파티션 서버가 START DBM 또는 STOP DBM 명령에 응답해야 하는 시간을 분 단위로 지정합니다. ADD DBPARTITIONNUM 조작의 시간종료 값으로도 사용됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

로컬 및 리모트 클라이언트가 있는 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

10 [1 - 1 440]

수치 단위

분

지정된 시간 내에 DB2START 명령에 응답하지 않는 데이터베이스 파티션 서버는 해당 인스턴스의 홈 디렉토리 아래의 `sqllib` 서브디렉토리의 `log` 서브디렉토리에 있는 `db2start` 오류 로그에 메시지를 전송합니다. 재시작하기 전에 이 노드에서 DB2STOP을 실행해야 합니다.

지정된 시간 내에 DB2STOP 명령에 응답하지 않는 데이터베이스 파티션 서버는 해당 인스턴스의 홈 디렉토리 아래의 `sqllib` 서브디렉토리의 `log` 서브디렉토리에 있는 `db2stop` 오류 로그에 메시지를 전송합니다. 응답하지 않는 각 데이터베이스 파티션 서

버에 대해 db2stop을 실행하거나 모두에 대해 실행할 수 있습니다. (이미 중지된 서버는 중지되었음을 알리는 메시지를 리턴합니다.)

다중 파티션 데이터베이스에서 db2start 또는 db2stop 조작이 *start_stop_time* 데이터베이스 관리 프로그램이 지정하는 값 내에 완료되지 않는 경우, 시간종료된 데이터베이스 파티션은 내부적으로 제거됩니다. *start_stop_time* 값이 낮은 많은 데이터베이스 파티션이 있는 환경에서 이 동작이 발생할 수 있습니다. 이 동작을 해결하려면 *start_stop_time*의 값을 늘리십시오.

DB2START, START DATABASE MANAGER 또는 ADD DBPARTITIONNUM 명령 중 하나를 사용하여 새 데이터베이스 파티션을 추가하는 경우, 데이터베이스 파티션 추가 조작은 인스턴스 내의 각 데이터베이스가 자동 스토리지에 대해 사용 가능한지 여부를 판별해야 합니다. 이는 각 데이터베이스의 카탈로그 파티션과 통신하여 수행됩니다. 자동 스토리지가 사용되는 경우 스토리지 경로 정의가 이 통신의 파트로 검색됩니다. 마찬가지로, 데이터베이스 파티션이 있는 시스템 임시 테이블 스페이스를 작성하는 경우 조작은 다른 데이터베이스 파티션 서버와 통신하여 해당 서버에 있는 데이터베이스 파티션의 테이블 스페이스 정의를 검색해야 합니다. *start_stop_time* 매개변수의 값을 결정할 때 이러한 요소를 고려해야 합니다.

ssl_svcename - SSL 서비스 이름 구성 매개변수

이 구성 매개변수는 데이터베이스 서버가 SSL 프로토콜을 사용하는 리모트 클라이언트 노드의 통신을 기다리는 데 사용하는 포트 이름을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null

이 구성 매개변수는 데이터베이스 서버가 SSL 프로토콜을 통해 리모트 클라이언트 노드의 통신을 기다리는 데 사용하는 포트를 포함합니다. 이 서비스 이름은 데이터베이스 관리 프로그램이 사용하도록 예약되어 있어야 합니다. 인스턴스를 시작할 때 DB2COMM 레지스트리 변수에 SSL이 있는지 여부를 정의해야 합니다. 그렇지 않으면 인스턴스는 SSL 프로토콜을 지원하지 않고 시작합니다.

DB2COMM에 TCP/IP와 SSL이 둘 다 있는 경우 **ssl_svcename**으로 지정한 포트는 **svcename**과 달라야 합니다. 그렇지 않으면 인스턴스는 SSL 또는 TCP/IP 프로토콜을 지원하지 않고 시작합니다.

UNIX 시스템에서 서비스 파일은 /etc/services에 있습니다.

데이터베이스 서버 SSL 포트(숫자 *n*)와 해당 서비스 이름은 데이터베이스 클라이언트의 서비스 파일에 정의되어야 합니다.

ssl_versions - 서버에서 지원되는 SSL 버전 구성 매개변수

이 구성 매개변수는 서버가 수신 연결 요청에 대해 지원하는 SSL(Secure Socket Layer) 및 TLS(Transport Layer Security) 버전을 지정합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 Null [TLSv1]

매개변수를 널(NULL)이나 TLSv1로 설정하면 매개변수는 TLS 버전 1.0(RFC2246) 및 TLS 버전 1.1(RFC4346)에 대한 지원을 사용 가능하게 합니다.

SSL 응답 확인 방식에서 클라이언트와 서버가 협상하여 TLS 버전 1.0 또는 TLS 버전 1.1을 사용할 최선의 보안 버전을 찾습니다. 클라이언트와 서버 간에 호환 가능한 버전이 없는 경우 연결은 실패합니다. 클라이언트가 TLS 버전 1.0과 TLS 버전 1.1을 지원하지만 서버가 TLS 버전 1.0만 지원하는 경우 TLS 버전 1.0이 사용됩니다.

stmt_conc - 명령문 집중기 구성 매개변수

이 구성 매개변수는 디폴트 명령문 집중기 동작을 설정합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

전파 클래스

명령문 경계

디폴트 [범위]

OFF [OFF, LITERALS]

이 구성 매개변수를 사용하여 동적문에 대해 명령문 집중을 사용할 수 있습니다. 데이터베이스 구성의 설정은 클라이언트가 명령문 집중기를 명시적으로 사용하거나 사용하지 않도록 설정하지 않는 경우에만 사용됩니다.

사용되는 경우, 명령문 집중기는 증가된 패키지 캐시 항목 공유를 허용하도록 동적문을 수정합니다.

명령문 집중기는 구성 매개변수가 OFF로 설정된 경우에는 사용되지 않습니다. 구성 매개변수가 LITERALS로 설정되는 경우에는 명령문 집중기가 사용됩니다. 명령문 집중기가 사용되는 경우 명령문에서 리터럴 값 외에 동일한 SQL문은 패키지 캐시 항목을 공유할 수 있습니다.

예를 들어, STMT_CONC를 LITERALS로 설정한 경우

```
SELECT FIRSTNME, LASTNAME FROM EMPLOYEE WHERE EMPNO='000020'
```

및

```
SELECT FIRSTNME, LASTNAME FROM EMPLOYEE WHERE EMPNO='000070'
```

명령문이 패키지 캐시에서 동일한 항목을 공유합니다. 그러면 패키지 캐시의 항목에서

```
SELECT FIRSTNME, LASTNAME FROM EMPLOYEE WHERE EMPNO=:L0
```

명령문을 사용하고 DB2가 원래 명령문에서 사용되는 리터럴에 따라서

```
:L0('000020' 또는 '000070')
```

에 적합한 값을 제공합니다.

이 매개변수는 명령문 텍스트를 변경하므로 플랜 선택에 상당한 영향을 미칠 수 있습니다. 명령문 집중기는 패키지 캐시의 유사한 명령문에 유사한 플랜이 있는 경우에만 사용해야 합니다. 예를 들어, 명령문에 다른 리터럴 값이 있는 경우에는 상당히 다른 플랜이 되므로 명령문 집중기를 LITERALS로 설정하지 말아야 합니다.

svcename - TCP/IP 서비스 이름

이 매개변수에는 데이터베이스 서버가 리모트 클라이언트 노드에서 통신을 대기하는 데 사용할 TCP/IP 포트의 이름이 있습니다. 이 이름은 데이터베이스 관리 프로그램이 사용할 수 있도록 예약되어 있습니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버

- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 Null

TCP/IP를 사용하여 Data Server Runtime Client에서 연결 요청을 승인하려면 데이터베이스 서버는 해당 서버에 지정된 포트에서 대기(listen)해야 합니다. 데이터베이스 서버의 시스템 관리자가 포트(번호 *n*)를 예약하고 서버에서 서비스 파일에 연관된 TCP/IP 서비스 이름을 정의해야 합니다.

데이터베이스 클라이언트의 서비스 파일에 데이터베이스 서버 포트(번호 *n*) 및 해당 TCP/IP 서비스 이름을 정의해야 합니다.

Linux 및 UNIX 시스템에서 서비스 파일은 `/etc/services`에 있습니다.

데이터베이스 서버가 시작되면 수신 연결 요청을 대기(listen)할 포트를 판별할 수 있도록 *svcename* 매개변수를 기본 연결 포트와 연관된 서비스 이름으로 설정해야 합니다.

sysadm_group - 시스템 관리 권한 그룹 이름

이 매개변수는 데이터베이스 관리 프로그램 인스턴스에 대한 SYSADM 권한을 사용하여 그룹 이름을 정의합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 NULL

SYSADM 권한 레벨은 인스턴스 레벨에서 최상위 레벨의 관리 권한입니다. SYSADM 권한을 갖는 사용자는 인스턴스에서 일부 유틸리티를 실행하고 일부 데이터베이스 및 데이터베이스 관리 프로그램 명령을 발행할 수 있습니다.

SYSADM 권한은 특정 운영 환경에서 사용되는 보안 기능으로 판별됩니다.

- Windows 운영 체제의 경우 이 매개변수를 로컬 또는 도메인 그룹으로 설정할 수 있습니다. 그룹 이름은 SQL 및 XML 한계에 지정된 길이 한계를 준수해야 합니다. 다음 사용자는 **sysadm_group** 데이터베이스 관리 프로그램 구성 매개변수에 대해 ″NULL″이 지정된 경우 SYSADM 권한을 갖습니다.
  - 로컬 관리자 그룹의 구성원
  - DB2_GRP_LOOKUP이 설정되지 않거나 DOMAIN으로 설정되는 경우 도메인 제어기에 있는 관리자 그룹의 구성원
  - 확장 보안 기능이 사용 가능한 경우 DB2ADMNS 그룹의 구성원. DB2ADMNS 그룹의 위치는 설치 중에 결정되었습니다.
  - LocalSystem 어카운트
- Linux 및 UNIX 시스템의 경우 이 매개변수의 값으로 『NULL』이 지정된 경우 SYSADM 그룹은 인스턴스 소유자의 기본 그룹을 디폴트로 사용합니다.

  값이 『NULL』이 아닌 경우 SYSADM 그룹은 유효한 UNIX 그룹 이름입니다.

매개변수를 디폴트값(NULL)으로 리스토어하려면 UPDATE DBM CFG USING SYSADM_GROUP NULL을 사용하십시오. 『NULL』 키워드를 대문자로 지정해야 합니다.

sysctrl_group - 시스템 제어 권한 그룹 이름

이 매개변수는 시스템 제어(SYSCTRL) 권한을 사용하여 그룹 이름을 정의합니다. SYSCTRL은 시스템 자원에 영향을 주는 조작을 허용하는 권한이 있으며 데이터에 대한 직접 액세스를 허용하지 않습니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 Null

모든 플랫폼의 그룹 이름이 SQL 및 XML 한계에 지정된 길이 한계를 준수하면 승인됩니다.

주의: 시스템 보안을 사용하는 경우(즉, **authentication**이 CLIENT, SERVER 또는 기타 유효한 인증인 경우) 이 매개변수는 Windows 클라이언트에서 NULL이어야 합니다. Windows 운영 체제가 그룹 정보를 저장하지 않으므로 사용자가 지정된 SYSCTRL 그룹의 구성원인지 여부를 판별할 수 있는 방법이 없기 때문입니다. 그룹 이름이 지정된 경우 사용자는 이 그룹의 구성원이 될 수 없습니다.

매개변수를 디폴트값(NULL)으로 리스토어하려면 UPDATE DBM CFG USING SYSCTRL_GROUP NULL을 사용하십시오. 키워드 NULL은 대문자로 지정해야 합니다.

sysmaint_group - 시스템 유지보수 권한 그룹 이름

이 매개변수는 시스템 유지보수(SYSMAINT) 권한이 있는 그룹 이름을 정의합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 Null

SYSMAINT에는 데이터에 직접 액세스할 수 있는 권한이 없어도 인스턴스와 연관된 모든 데이터베이스에서 유지보수 조작을 수행할 수 있는 특권이 있습니다.

모든 플랫폼의 그룹 이름이 SQL 및 XML 한계에 지정된 길이 한계를 준수하면 승인됩니다.

주의: 시스템 보안을 사용하는 경우(즉, **authentication**이 CLIENT, SERVER 또는 기타 유효한 인증인 경우) 이 매개변수는 Windows 클라이언트에서 NULL이어야 합니다. Windows 운영 체제가 그룹 정보를 저장하지 않으므로 사용자가 지정된 SYSMAINT 그룹의 구성원인지 여부를 판별할 수 있는 방법이 없기 때문입니다. 그룹 이름이 지정된 경우 사용자는 이 그룹의 구성원이 될 수 없습니다.

매개변수를 디폴트값(NULL)으로 리스토어하려면 UPDATE DBM CFG USING SYSMAINT_GROUP NULL을 사용하십시오. 키워드 NULL은 대문자로 지정해야 합니다.

sysmon_group - 시스템 모니터 권한 그룹 이름

이 매개변수는 시스템 모니터(SYSMON) 권한이 있는 그룹 이름을 정의합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 Null

인스턴스 레벨에서 SYSMON 권한이 있는 사용자는 데이터베이스 관리 프로그램 인스턴스 또는 해당 데이터베이스의 데이터베이스 시스템 모니터 스냅샷을 작성할 수 있습니다. SYSMON 권한이 있는 경우 다음 명령을 사용할 수 있습니다.

- GET DATABASE MANAGER MONITOR SWITCHES
- GET MONITOR SWITCHES
- GET SNAPSHOT
- LIST ACTIVE DATABASES
- LIST APPLICATIONS
- LIST DCS APPLICATIONS
- RESET MONITOR
- UPDATE MONITOR SWITCHES

SYSADM, SYSCTRL 또는 SYSMAINT 권한이 있는 사용자는 자동으로 데이터베이스 시스템 모니터 스냅샷을 작성할 수 있으며 해당 명령을 사용할 수 있습니다.

모든 플랫폼에서 그룹 이름은 SQL 및 XML 한계에 지정된 길이 한계를 준수하는 한 승인됩니다.

매개변수를 기본값(NULL)으로 리스토어하려면 UPDATE DBM CFG USING SYSMON_GROUP NULL을 사용하십시오. 키워드 NULL은 대문자로 지정해야 합니다.

tm_database - 트랜잭션 관리 프로그램 데이터베이스 이름

이 매개변수는 각 DB2 인스턴스에 대한 트랜잭션 관리 프로그램(TM) 데이터베이스의 이름을 식별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

1ST_CONN [유효한 데이터베이스 이름]

TM 데이터베이스는 다음과 같습니다.

- 로컬 DB2 데이터베이스
- 호스트 또는 AS/400 시스템에 상주하지 않는 리모트 DB2 데이터베이스
- TCP/IP를 통해 액세스하며 동기점 관리 프로그램(SPM)이 사용되지 않는 경우 OS/390용 DB2 OS/390 버전 5 데이터베이스

TM 데이터베이스는 로그 프로그램 및 코디네이터로 사용되는 데이터베이스이며 인다우트(Indoubt) 트랜잭션의 복구를 수행하는 데 사용됩니다.

이 매개변수를 **1ST_CONN**으로 설정하여 TM 데이터베이스를 사용자가 연결된 첫 번째 데이터베이스로 설정할 수 있습니다.

권장사항: 간단한 관리 및 조작을 위해 여러 인스턴스에서 일부 데이터베이스를 작성하고 이러한 데이터베이스를 TM 데이터베이스로 독점 사용할 수 있습니다.

tp_mon_name - 트랜잭션 프로세서 모니터 이름

이 매개변수는 사용 중인 트랜잭션 처리(TP) 모니터 제품의 이름을 식별합니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 디폴트 없음

유효한 값

- CICS®
- MQ
- ENCINA
- CB
- SF
- TUXEDO
- TOPEND
- 공백이거나 일부 다른 값(UNIX 및 Windows의 경우. Solaris 또는 SINIX의 경우 사용 가능한 다른 값이 없음)

- 응용프로그램이 WebSphere® Enterprise Server Edition CICS 환경에서 실행되는 경우 이 매개변수를 『CICS』로 설정해야 합니다.
- 응용프로그램이 WebSphere Enterprise Server Edition Encina® 환경에서 실행되는 경우 이 매개변수를 『ENCINA』로 설정해야 합니다.
- 응용프로그램이 WebSphere Enterprise Server Edition Component Broker 환경에서 실행되는 경우 이 매개변수를 『CB』로 설정해야 합니다.
- 응용프로그램이 IBM MQSeries® 환경에서 실행되는 경우 이 매개변수를 『MQ』로 설정해야 합니다.
- 응용프로그램이 BEA Tuxedo 환경에서 실행되는 경우 이 매개변수를 『TUXEDO』로 설정해야 합니다.
- 응용프로그램이 IBM San Francisco 환경에서 실행되는 경우 이 매개변수를 『SF』로 설정해야 합니다.

IBM WebSphere EJB 및 **Microsoft Transaction Server** 사용자는 이 매개변수에 맞게 값을 구성할 필요가 없습니다.

위의 제품이 사용되지 않는 경우 이 매개변수를 구성하지 말고 비워 두어야 합니다.

Windows의 IBM DB2 이전 버전에서 이 매개변수에는 XA Transaction Manager의 함수 *ax_reg* 및 *ax_unreg*가 들어 있는 DLL의 경로 및 이름이 포함되었습니다. 이 형식은 계속 지원됩니다. 이 매개변수의 값이 위의 TP 모니터 이름과 일치하지 않는 경우 값으로 *ax_reg* 및 *ax_unreg* 함수가 포함된 라이브러리 이름을 가정합니다. 이 사항은 UNIX 및 Windows 환경의 경우에 해당합니다.

TXSeries® CICS 및 Encina 사용자: Windows에서 이 제품의 이전 버전에서는 이 매개변수를 『libEncServer:C』 또는 『libEncServer:E』로 구성해야 했습니다. 이러한 기능이 계속 지원되지만 더 이상 필요하지 않습니다. 매개변수를 『CICS』 또는 『ENCINA』로 구성하면 충분합니다.

MQSeries 사용자: Windows에서 이 제품의 이전 버전에서는 이 매개변수를 『mqmax』로 구성해야 했습니다. 이러한 기능이 계속 지원되지만 더 이상 필요하지 않습니다. 매개변수를 『MQ』로 구성하면 충분합니다.

Component Broker 사용자: Windows에서 이 제품의 이전 버전에서는 이 매개변수를 『somtrx1i』로 구성해야 했습니다. 이러한 기능이 계속 지원되지만 더 이상 필요하지 않습니다. 매개변수를 『CB』로 구성하면 충분합니다.

San Francisco 사용자: Windows에서 이 제품의 이전 버전에서는 이 매개변수를 『ibmsfDB2』로 구성해야 했습니다. 이러한 기능이 계속 지원되지만 더 이상 필요하지 않습니다. 매개변수를 『SF』로 구성하면 충분합니다.

이 매개변수에 지정할 수 있는 최대 문자열 길이는 19자입니다.

또한 IBM DB2 버전 9.1의 XA OPEN 문자열에서 이 정보를 구성할 수 있습니다. 여러 트랜잭션 처리 모니터가 단일 DB2 인스턴스를 사용 중인 경우 이 기능을 사용해야 합니다.

trust_allclnts - 모든 클라이언트 신뢰

이 매개변수와 *trust_clntauth*는 데이터베이스 환경에 대해 사용자의 유효성이 확인되는 위치를 판별하는 데 사용됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

YES [NO, YES, DRDAONLY]

이 매개변수는 *authentication* 매개변수가 CLIENT로 설정된 경우에만 활성입니다.

이 매개변수의 디폴트『YES』를 승인하면 모든 클라이언트가 트러스트된 클라이언트로 취급됩니다. 즉 서버는 클라이언트에서 보안 레벨이 사용 가능하며 사용자가 클라이언트에서 유효성 확인될 가능성을 가정합니다.

이 매개변수는 *authentication* 매개변수가 CLIENT로 설정된 경우에만『NO』로 변경될 수 있습니다. 이 매개변수가『NO』로 설정된 경우, 트러스트되지 않은 클라이언트는 서버에 연결할 때 사용자 ID와 암호 조합을 제공해야 합니다. 트러스트되지 않은 클라이언트는 사용자 인증을 위한 보안 서브시스템이 없는 운영 체제 플랫폼입니다.

이 매개변수를『DRDAONLY』로 설정하면 OS/390 및 z/OS용 DB2, VM 및 VSE용 DB2 및 OS/400®용 DB2의 클라이언트를 제외한 모든 클라이언트에 대해 보호됩니다. 이 클라이언트만 클라이언트 측 인증을 수행하도록 트러스트될 수 있습니다. 다른 모든 클라이언트는 서버가 인증할 사용자 ID 및 암호를 제공해야 합니다.

*trust_allclnts*를『DRDAONLY』로 설정하면 *trust_clntauth* 매개변수는 클라이언트가 인증될 위치를 판별하는 데 사용됩니다. *trust_clntauth*가『CLIENT』로 설정되면 인증은 클라이언트에서 발생합니다. *trust_clntauth*가『SERVER』로 설정되면 인증은 암호가 제공되지 않는 경우 클라이언트에서 발생하며 암호가 제공되는 경우 서버에서 발생합니다.

trust_clntauth - 트러스트된 클라이언트 인증

이 매개변수는 클라이언트가 연결에 대해 사용자 ID와 암호 조합을 제공하는 경우 트러스트된 클라이언트가 인증되는 위치가 서버인지 클라이언트인지를 지정합니다. 이 매개변수(와 *trust_allclnts*)는 *authentication* 매개변수가 CLIENT로 설정된 경우에만 활성입니다. 사용자 ID와 암호가 제공되지 않는 경우 클라이언트는 사용자의 유효성을 확인한 것으로 가정되며, 서버에서 더 이상 유효성 확인이 수행되지 않습니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

CLIENT [CLIENT, SERVER]

이 매개변수가 CLIENT(디폴트)로 설정된 경우, 트러스트된 클라이언트는 사용자 ID와 암호 조합을 제공하지 않고 연결할 수 있으며, 운영 체제가 이미 사용자를 인증한 것으로 가정됩니다. 이 매개변수가 SERVER로 설정된 경우에는 서버에서 사용자 ID와 암호가 유효성 확인됩니다.

CLIENT의 숫자 값은 0이며, SERVER의 숫자 값은 1입니다.

util_impact_lim - 인스턴스 영향 규정

데이터베이스 관리자(DBA)는 이 매개변수를 사용하여 워크로드에서 조정 유틸리티의 성능 저하를 제한할 수 있습니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

10 [1 - 100]

수치 단위

워크로드에서 허용 가능한 영향의 퍼센트

성능 저하가 제한되는 경우 DBA는 중요한 프로덕션 기간에 온라인 유틸리티를 실행하여 성능이 프로덕션 작업에 미치는 영향이 승인할 수 있는 한계 이내가 되도록 보증할 수 있습니다.

예를 들어, DBA가 *util_impact_lim*(영향 규정) 값을 10으로 지정하는 경우, 조정 백업 호출이 워크로드에 10퍼센트 이상 영향을 미치지 않을 것을 예상할 수 있습니다.

*util_impact_lim*이 100인 경우에는 유틸리티 호출이 조정되지 않습니다. 이 경우 유틸리티는 워크로드에 임의의(원하지 않는) 영향을 미칠 수 있습니다. *util_impact_lim*이 100 미만의 값으로 설정되는 경우에는 유틸리티를 조정 모드로 호출할 수 있습니다. 조정 모드로 실행하려면 유틸리티는 또한 0이 아닌 우선순위로 호출되어야 합니다.

권장사항: 대부분의 사용자는 *util_impact_lim*을 낮은 값(예: 1 - 10)으로 설정하는 것이 이롭습니다.

조정 유틸리티는 일반적으로 비조정 유틸리티보다 완료하는 데 더 많은 시간이 걸립니다. 유틸리티가 너무 오래 실행되는 경우에는 *util_impact_lim* 값을 늘리거나 *util_impact_lim*을 100으로 설정하여 조절을 사용하지 마십시오.

데이터베이스 구성 매개변수

alt_collate - 대체 조합 시퀀스

이 매개변수는 비유니코드 데이터베이스에서 유니코드 테이블에 사용할 조합 시퀀스를 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

Null [IDENTITY_16BIT]

이 매개변수가 설정될 때까지 비유니코드 데이터베이스에는 유니코드 테이블 및 루틴을 작성할 수 없습니다. 이 매개변수를 일단 설정한 후에는 변경하거나 재설정할 수 없습니다.

유니코드 데이터베이스에는 이 매개변수를 설정할 수 없습니다.

app_ctl_heap_sz - 응용프로그램 제어 힙 크기

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 DB2 버전 9.5 데이터베이스 관리 프로그램에서는 사용되지 않습니다. 버전 9.5에서는 *appl_memory* 구성 매개변수로 교체되었습니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에서만 적용됩니다.

파티션된 데이터베이스 및 파티션 내 병렬 처리를 사용하는(intra_parallel=ON) 파티션되지 않은 데이터베이스의 경우 이 매개변수는 응용프로그램에 할당된 공유 메모리 영역의 평균 크기를 지정합니다. 파티션 내 병렬 처리를 사용하지 않는(intra_parallel=OFF)

파티션되지 않은 데이터베이스의 경우 힙에 할당되는 최대 전용 메모리입니다. 데이터베이스 파티션마다 연결 당 하나의 응용프로그램 제어 힙이 있습니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

로컬 및 리모트 클라이언트가 있는 데이터베이스 서버

- INTRA_PARALLEL을 사용하지 않는 경우 128 [1 - 64 000]
- INTRA_PARALLEL을 사용하는 경우 512 [1 - 64 000]

로컬 클라이언트가 있는 데이터베이스 서버

- INTRA_PARALLEL을 사용하지 않는 경우 64 [1 - 64 000](비UNIX 플랫폼의 경우)
- INTRA_PARALLEL을 사용하는 경우 512 [1 - 64 000](비UNIX 플랫폼의 경우)
- INTRA_PARALLEL을 사용하는 경우 128 [1 - 64 000](Linux 및 UNIX 플랫폼의 경우)
- INTRA_PARALLEL을 사용하는 경우 512 [1 - 64 000](Linux 및 UNIX 플랫폼의 경우)

로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

512 [1 - 64 000]

수치 단위

페이지(4KB)

할당 시기

응용프로그램 시작 시

사용 가능한 시기

응용프로그램 완료 시

응용프로그램 제어 힙은 주로 동일한 요청 대신 작업 중인 에이전트들 간 정보를 공유하는 데 필요합니다. 병렬 처리 수준이 1인 쿼리를 실행할 때 파티션되지 않은 데이터베이스의 경우 이 힙의 사용이 최소화됩니다.

이 힙은 선언된 임시 테이블에 대한 디스크립터 정보를 저장하는 데에도 사용됩니다. 명시적으로 삭제되지 않은 선언된 모든 임시 테이블에 대한 디스크립터 정보는 이 힙의 메모리에 보존되며 선언된 임시 테이블이 삭제되기 전까지 삭제할 수 없습니다.

권장사항: 초기에는 디폴트값으로 시작하십시오. 복잡한 응용프로그램을 실행하거나 여러 개의 데이터베이스 파티션이 속한 시스템이 있거나 선언된 임시 테이블을 사용하는 경우 값을 높게 설정해야 할 수도 있습니다. 동시에 사용 중인 선언된 임시 테이블 수로 인해 필요한 메모리의 양이 증가합니다. 여러 컬럼이 있는 선언된 임시 테이블에는 컬럼이 적은 테이블보다 큰 테이블 디스크립터가 있으므로 응용프로그램의 선언된 임시 테이블에 컬럼 수가 많아도 응용프로그램 제어 힙에 대한 요구가 증가합니다.

appgroup_mem_sz - 응용프로그램 그룹 메모리 세트의 최대 크기

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서는 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 DB2 버전 9.5 데이터베이스 관리 프로그램에서는 사용되지 않습니다. 버전 9.5에서 *appl_memory* 구성 매개변수로 교체되었습니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에만 적용됩니다.

이 매개변수는 응용프로그램 그룹 공유 메모리 세그먼트의 크기를 결정합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

로컬 클라이언트가 있는 UNIX 데이터베이스 서버(32비트 HP-UX가 아닌)

20 000 [1 - 1 000 000]

32비트 HP-UX

- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

10 000 [1 - 1 000 000]

로컬 클라이언트가 있는 Windows 데이터베이스 서버

10 000 [1 - 1 000 000]

로컬 및 리모트 클라이언트가 있는 데이터베이스 서버(32비트 HP-UX가 아닌)

30 000 [1 - 1 000 000]

로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버(32비트 HP-UX가 아닌)

40 000 [1 - 1 000 000]

수치 단위

페이지(4KB)

동일한 응용프로그램에서 작동하는 에이전트 간에 공유해야 하는 정보는 응용프로그램 그룹 공유 메모리 세그먼트에 저장됩니다.

파티션된 데이터베이스에서 또는 파티션 내 병렬 처리를 사용하거나 집중기를 사용하는 파티션되지 않은 데이터베이스에서는 다중 응용프로그램이 단일 응용프로그램 그룹을 공유합니다. 단일 응용프로그램 그룹 공유 메모리 세그먼트가 응용프로그램 그룹에 할당됩니다. 응용프로그램 그룹 공유 메모리 세그먼트 내에서 각 응용프로그램에는 각자의 응용프로그램 제어 힙이 있으며 모든 응용프로그램 그룹은 단일 응용프로그램 그룹 공유 힙을 공유합니다.

단일 응용프로그램 그룹 내의 응용프로그램 수는 다음과 같이 계산합니다.

```
appgroup_mem_sz / app_ctl_heap_sz
```

응용프로그램 그룹 공유 힙 *크기*는 다음과 같이 계산합니다.

```
appgroup_mem_sz * groupheap_ratio / 100
```

각 응용프로그램 제어 힙의 *크기*는 다음과 같이 계산합니다.

```
app_ctl_heap_sz * (100 - groupheap_ratio) / 100
```

권장사항: 성능 문제점이 없는 경우에는 이 매개변수를 디폴트값으로 유지하십시오.

appl_memory - 응용프로그램 메모리 구성 매개변수

이 매개변수를 사용하여 DBA와 ISV는 DB2 데이터베이스 에이전트가 서비스 응용프로그램 요청에 할당하는 응용프로그램 메모리의 최대량을 제어할 수 있습니다. 디폴트로 이 값은 AUTOMATIC으로 설정되며, 데이터베이스 파티션이 할당하는 메모리의 총량이 *instance_memory* 한계 내인 한 모든 응용프로그램 메모리 요청이 허용됨을 의미합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

Automatic [128 - 4 294 967 295]

수치 단위

페이지(4KB)

할당 시기

데이터베이스를 활성화하는 동안

사용 가능한 시기

데이터베이스를 비활성화하는 동안

주: *appl_memory*가 AUTOMATIC으로 설정된 경우 데이터베이스 활성화 시 초기 응용프로그램 메모리는 최소로 할당되며 필요에 따라 증가(또는 감소)됩니다. 변경사항은 메모리에 적용되며 *appl_memory* 값은 db2 get db cfg show detail가 표시하는 것처럼 디스크에서 변경되지 않습니다. 다음 활성화 시 값이 재계산됩니다. *appl_memory*가 특정 값으로 설정된 경우, 데이터베이스 활성화 시 요청된 메모리 양이 할당되고 응용프로그램 메모리 *크기*가 변경되지 않습니다. 운영 체제에서 응용프로그램 메모리의 초기 양을 할당할 수 없는 경우 또는 *instance_memory* 한계를 초과한 경우에는 SQL1084C 오류(공유 메모리 세그먼트를 할당할 수 없음)가 발생하여 데이터베이스 활성화가 실패합니다.

applheapsz - 응용프로그램 힙 크기

이전 릴리스에서는 *applheapsz* 데이터베이스 구성 매개변수가 해당 응용프로그램에 대해 작업 중인 개별 데이터베이스 에이전트가 사용할 수 있는 응용프로그램 메모리의 양을 참조했습니다. 버전 9.5에서 *applheapsz*는 전체 응용프로그램이 사용할 수 있는 총 응용프로그램 메모리 양을 나타냅니다. DPF, 집중기 또는 SMP 구성의 경우 AUTOMATIC 설정이 사용되지 않았으면 이전 릴리스에서 사용된 *applheapsz* 값은 유사한 워크로드에서 늘려야 할 수도 있습니다.

버전 9.5에서 이 데이터베이스 구성 매개변수는 디폴트값 AUTOMATIC이며 *appl_memory* 한계 또는 *instance_memory* 한계에 접근할 때까지 필요에 따라 증가함을 의미합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

Automatic [16 - 60 000]

수치 단위

페이지(4KB)

할당 시기

응용프로그램이 데이터베이스와 연관되거나 데이터베이스에 연결될 때

사용 가능한 시기

응용프로그램과 데이터베이스의 연관이 해제되거나 데이터베이스에서 응용프로그램의 연결이 끊어질 때

주: 이 매개변수는 최대 응용프로그램 힙 크기를 정의합니다. 응용프로그램이 데이터베이스에 처음 연결할 때 데이터베이스 응용프로그램당 하나의 응용프로그램 힙이 할당됩니다. 해당 응용프로그램에 대해 작업 중인 모든 데이터베이스 에이전트가 힙을 공유합니다(이전 릴리스에서 각 데이터베이스 에이전트는 자체 응용프로그램 힙을 할당함). 메모리는 응용프로그램을 처리할 때 필요하면 응용프로그램 힙에서 할당되며 이 매개변수에 지정된 한계까지 할당됩니다. AUTOMATIC으로 설정된 경우 응용프로그램 힙은 필요하면 데이터베이스의 *appl_memory* 한계 또는 데이터베이스 파티션의 *instance_memory* 한계까지 확장됩니다. 데이터베이스에서 응용프로그램의 연결을 끊으면 전체 응용프로그램 힙이 사용 가능해집니다.

온라인으로 변경된 값은 응용프로그램 연결 경계에서 적용됩니다. 즉, 동적으로 변경된 후 현재 연결된 응용프로그램은 계속 이전 값을 사용하지만 새로 연결된 모든 응용프로그램은 새 값을 사용합니다.

archretrydelay - 오류 시 아카이브 재시도 대기 시간

이 매개변수는 아카이브 시도가 실패한 후 로그 파일을 다시 아카이브하려고 시도하기 전에 대기할 시간을 초 단위로 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

20 [0 - 65 535]

연속 재시도는 *numarchretry* 데이터베이스 구성 매개변수 값이 1 이상으로 설정된 경우에만 유효합니다.

auto_del_rec_obj - 복구 오브젝트의 자동화된 삭제 구성 매개변수

이 매개변수는 연결된 복구 실행기록 파일 항목이 프룬되는 경우 데이터베이스 로그 파일, 백업 이미지 및 로드 사본 이미지를 삭제해야 하는지 여부를 지정합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

OFF [ON; OFF]

PRUNE HISTORY 명령 또는 db2Prune API를 사용하여 복구 실행기록 파일의 항목을 프룬(prune)할 수 있습니다. 전체 데이터베이스 백업을 완료할 때마다 복구 실행기록 파일을 자동으로 프룬하도록 IBM Data Server 데이터베이스 관리 프로그램을 구성할 수도 있습니다. *auto_del_rec_obj* 데이터베이스 구성 매개변수를 ON으로 설정하면 데이터베이스 관리 프로그램은 실행기록 파일을 프룬할 때 해당 물리 로그 파일, 백업 이미지 및 로드 사본 이미지도 삭제합니다. 데이터베이스 관리 프로그램은 스토리지 미디어가 디스크인 경우 또는 Tivoli Storage Manager와 같은 스토리지 관리자를 사용 중인 경우에만 데이터베이스 로그, 백업 이미지 및 로드 사본 이미지와 같은 복구 오브젝트를 삭제할 수 있습니다.

auto_maint - 자동 유지보수

이 매개변수는 다른 모든 자동 유지보수 데이터베이스 구성 매개변수(*auto_db_backup*, *auto_tbl_maint*, *auto_runstats*, *auto_stats_prof*, *auto_stmt_stats*, *auto_prof_upd* 및 *auto_reorg*)의 상위입니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

ON [ON; OFF]

이 매개변수를 사용하지 않는 경우 모든 하위 매개변수도 사용되지 않으나 데이터베이스 구성 파일에 기록된 설정은 변경되지 않습니다. 이 상위 매개변수가 사용되는 경우 하위 매개변수에 기록된 값이 적용됩니다. 이 방법으로 자동 유지보수를 전역으로 사용하거나 사용하지 않을 수 있습니다.

디폴트로 이 매개변수는 ON으로 설정됩니다.

다음 매개변수를 설정하여 개별 자동 유지보수 기능을 독립적으로 사용하거나 사용하지 않을 수 있습니다.

auto_db_backup

이 자동화된 유지보수 매개변수는 데이터베이스에 대해 자동 백업 조작을 사용하거나 사용하지 않도록 설정합니다. 백업 규정(규칙 또는 지침의 정의된 세트)은 자동화된 동작을 지정하는 데 사용할 수 있습니다. 백업 규정의 목표는 데이터베이스를 일정하게 백업하는 것입니다. 데이터베이스의 백업 규정은 DB2 Health Monitor를 처음 실행할 때 자동으로 작성됩니다. 디폴트로 이 매개변수는 OFF로 설정됩니다. 사용 가능하게 하려면 이 매개변수를 ON으로 설정해야 하며 상위 매개변수도 사용 가능해야 합니다.

auto_tbl_maint

이 매개변수는 모든 테이블 유지보수 매개변수(*auto_runstats*, *auto_stats_prof*, *auto_prof_upd* 및 *auto_reorg*)의 상위입니다. 이 매개변수를 사용하지 않는 경우 모든 하위 매개변수도 사용되지 않으나 데이터베이스 구성 파일에 기록된 설정은 변경되지 않습니다. 이 상위 매개변수가 사용되는 경우 하위 매개변수에 기록된 값이 적용됩니다. 이 방법으로 테이블 유지보수를 전역으로 사용하거나 사용하지 않을 수 있습니다.

디폴트로 이 매개변수는 ON으로 설정됩니다.

auto_runstats

이 자동화된 테이블 유지보수 매개변수는 데이터베이스에 대해 자동 테이블 runstats 조작을 사용하거나 사용하지 않도록 설정합니다. runstats 규정(규칙 또는 지침의 정의된 세트)은 자동화된 동작을 지정하는 데 사용할 수 있습니다. runstats 유틸리티가 수집한 통계는 옵티마이저가 실제 데이터 액세스에 가장 효율적인 플랜을 판별하는 데 사용됩니다. 사용 가능하게 하려면 이 매개변수를 On으로 설정해야 하며 상위 매개변수도 사용 가능해야 합니다.

디폴트로 이 매개변수는 ON으로 설정됩니다.

auto_stats_prof

사용되는 경우 이 자동화된 테이블 유지보수 매개변수는 여러 테이블에 대해 복합 쿼리, 많은 술어, 조인 및 그룹화 조작을 포함한 워크로드가 있는 응용프로그램을 개선하도록 디자인된 통계 프로파일 생성 기능을 켭니다. 사용 가능하게 하려면 이 매개변수를 ON으로 설정해야 하며 상위 매개변수도 사용 가능해야 합니다.

디폴트로 이 매개변수는 OFF로 설정됩니다.

auto_stmt_stats

이 매개변수는 실시간 통계 수집을 사용하거나 사용하지 않도록 설정합니다. 이 매개변수는 *auto_runstats* 구성 매개변수의 하위입니다. 이 기능은 상위인 *auto_runstats* 구성 매개변수도 사용되는 경우에만 사용됩니다. 예를 들어, *auto_stmt_stats*를 사용하려면 *auto_maint*, *auto_tbl_maint* 및 *auto_runstats*를 ON으로 설정하십시오. 하위 값을 보존하려면 *auto_maint* 구성 매개변수가 OFF인 상태에서 *auto_runstats* 구성 매개변수를 ON으로 설정할 수 있습니다. 해당 자동 Runstats 기능은 여전히 OFF가 됩니다.

자동 Runstats와 자동 Reorg를 둘 다 사용한다고 가정할 때 설정은 다음과 같습니다.

```
Automatic maintenance                        (AUTO_MAINT) = ON
  Automatic database backup                (AUTO_DB_BACKUP) = OFF
    Automatic table maintenance            (AUTO_TBL_MAINT) = ON
      Automatic runstats                    (AUTO_RUNSTATS) = ON
        Automatic statement statistics     (AUTO_STMT_STATS) = OFF
      Automatic statistics profiling      (AUTO_STATS_PROF) = OFF
        Automatic profile updates           (AUTO_PROF_UPD) = OFF
        Automatic reorganization               (AUTO_REORG) = ON
```

*auto_tbl_maint*를 OFF로 설정하여 자동 Runstats와 자동 Reorg 기능을 둘 다 임시로 사용하지 않을 수 있습니다. 나중에 *auto_tbl_maint*를 다시 ON으로 설정하면 두 기능이 모두 사용됩니다. db2stop 또는 db2start 명령을 실행하여 변경사항을 적용할 필요는 없습니다.

디폴트로 이 매개변수는 OFF로 설정됩니다.

auto_prof_upd

사용되는 경우, 이 자동화된 테이블 유지보수 매개변수(*auto_stats_prof*의 하위)는 runstats 프로파일이 권장사항으로 갱신되도록 지정합니다. 이 매개변수가 사용되지 않는 경우 권장사항은 opt_feedback_ranking 테이블에 저장되며 수동으로 runstats 프로파일을 갱신하는 경우 이 테이블을 조사할 수 있습니다. 사용 가능하게 하려면 이 매개변수를 ON으로 설정해야 하며 상위 매개변수도 사용 가능해야 합니다.

디폴트로 이 매개변수는 OFF로 설정됩니다.

auto_reorg

이 자동화된 테이블 유지보수 매개변수는 데이터베이스에 대해 자동 테이블 및 인덱스 재구성을 사용하거나 사용하지 않도록 설정합니다. 재구성 규정(규칙 또는 지침의 정의된 세트)은 자동화된 동작을 지정하는 데 사용할 수 있습니다. 사용 가능하게 하려면 이 매개변수를 ON으로 설정해야 하며 상위 매개변수도 사용 가능해야 합니다.

디폴트로 이 매개변수는 OFF로 설정됩니다.

autorestart - 자동 재시작 사용

이 매개변수는 데이터베이스가 비정상적으로 종료된 경우 응용프로그램이 데이터베이스에 연결할 때 데이터베이스 관리 프로그램이 데이터베이스 재시작 유틸리티를 자동으로 호출할 수 있는지 여부를 판별합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
즉시

디폴트 [범위]
On [On; Off]

응용프로그램이 데이터베이스에 연결되어 있는 동안 데이터베이스가 비정상적으로 종료된 경우(예: 정전 또는 시스템 소프트웨어 결함으로 인해) 데이터베이스 재시작 유틸리티가 *응급 복구*를 수행합니다. 데이터베이스 버퍼 풀에 있지만 장애가 발생한 경우 디스크에 기록되지 않은 커미트된 트랜잭션을 적용합니다. 또한 디스크에 기록되었을 수 있는 언커미트된 트랜잭션을 백아웃합니다.

*autorestart*를 사용하지 않는 경우 응급 복구를 수행해야 하는(재시작해야 하는) 데이터베이스에 연결하려고 시도하는 응용프로그램이 SQL1015N 오류를 받습니다. 이 경우 응용프로그램은 데이터베이스 재시작 유틸리티를 호출하거나 복구 도구의 재시작 조작을 선택하여 데이터베이스를 재시작할 수 있습니다.

avg_appls - 활성 응용프로그램의 평균 수

이 매개변수는 쿼리 옵티마이저가 선택한 액세스 플랜에 대해 런타임에 사용 가능한 버퍼 풀의 *크기*를 추정하는 데 사용합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
명령문 경계

디폴트 [범위]
Automatic [1 – maxappls]

수치 단위
카운터

권장사항: 다중 사용자 환경에서 DB2를 실행하는 경우 특히 복잡한 쿼리 및 대형 버퍼 풀을 사용하는 경우 여러 쿼리 사용자가 시스템을 사용 중임을 쿼리 옵티마이저가 인식하도록 설정할 수 있으므로 옵티마이저는 버퍼 풀 가용성을 가정할 때 신중해야 합니다.

이 매개변수를 설정할 때 일반적으로 데이터베이스를 사용하는 복잡한 쿼리 응용프로그램 수를 추정해야 합니다. 이 추정에서는 모든 간단한 OLTP 응용프로그램이 제외되어야 합니다. 이 수를 추정하는 데 문제가 있는 경우 다음을 곱할 수 있습니다.

- 데이터베이스에 대해 실행 중인 모든 응용프로그램의 평균 수. 데이터베이스 시스템 모니터는 지정된 시간에 응용프로그램 수에 대한 정보를 제공할 수 있으며 샘플링 기술을 사용하여 일정 기간 동안 평균을 계산할 수 있습니다. 데이터베이스 시스템 모니터의 정보에는 OLTP 및 비OLTP 응용프로그램이 포함됩니다.
- 복합 쿼리 응용프로그램의 퍼센트 추정.

옵티마이저에 영향을 주는 다른 구성 매개변수를 사용할 때와 마찬가지로 이 매개변수를 조금씩 조정해야 합니다. 따라서 경로 선택 차이가 최소화됩니다.

이 매개변수를 변경한 후 응용프로그램을 리바인드해야 합니다(REBIND PACKAGE 명령 사용).

backup_pending - 백업 보류 표시기

이 매개변수는 데이터베이스에 액세스하기 전에 데이터베이스의 전체 백업을 수행해야 하는지 여부를 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

데이터베이스가 복구 불가능한 상태에서 복구 가능한 상태로 이동하도록 데이터베이스 구성이 변경된 경우에만 이 매개변수가 on입니다(즉, 처음에 *logretain* 및 *userexit* 매개변수가 모두 NO로 설정된 후 이 매개변수 중 하나 또는 둘 다 YES로 설정되었으며 데이터베이스 구성에 대한 갱신이 승인됨).

blk_log_dsk_ful - 디스크 가득참 로그 시 블록

이 매개변수는 DB2가 사용 중인 로그 경로에 새 로그 파일을 작성할 수 없는 경우 디스크 가득참 오류가 발생하지 않도록 예방하기 위해 설정할 수 있습니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

No [Yes; No]

디스크 가득참 오류를 생성하는 대신 DB2는 성공할 때까지 5분마다 로그 파일을 작성하려고 시도합니다. 각 시도 후 DB2는 관리 통지 로그에 메시지를 기록합니다. 로그 디스크 가득참 조건 때문에 응용프로그램이 정지되었는지 확인하는 유일한 방법은 관리 통지 로그를 모니터하는 것입니다. 테이블 데이터를 갱신하려고 시도하는 사용자 응용프로그램은 로그 파일이 작성될 때까지 트랜잭션을 커미트할 수 없습니다. 읽기 전용 쿼리는 직접 영향을 받지 않을 수 있습니다. 그러나 쿼리가 갱신 요청이 잠근 데이터 또는 갱신 응용프로그램이 버퍼 풀에서 고정한 데이터 페이지에 액세스해야 하는 경우, 읽기 전용 쿼리도 정지한 것처럼 보일 수 있습니다.

*blk_log_dsk_ful*을 *yes*로 설정하면 DB2가 로그 디스크 가득참 오류에 직면하는 경우 응용프로그램이 정지하게 되므로 오류를 해결하고 트랜잭션을 완료할 수 있습니다. 이전 로그 파일을 다른 파일 시스템으로 이동하거나 파일 시스템을 확장하여 디스크 가득참 상황을 해결하면 정지된 응용프로그램을 완료할 수 있습니다.

*blk_log_dsk_ful*이 *no*로 설정된 경우에는 로그 디스크 가득참 오류를 수신하는 트랜잭션이 실패하고 롤백됩니다. 일부 상황에서는 트랜잭션이 로그 디스크 가득참 오류를 유발하는 경우 데이터베이스가 중지됩니다.

blocknonlogged - 로깅되지 않는 활동을 허용하는 테이블 작성 블록

이 매개변수는 데이터베이스 관리 프로그램이 테이블에서 NOT LOGGED 또는 NOT LOGGED INITIALLY 속성이 활성화되도록 허용할지 여부를 지정합니다. 또한 1GB보다 큰 LOB 컬럼은 로깅할 수 없으므로 1GB보다 큰 LOB 컬럼을 갖는 테이블 작성을 막기 위해 사용할 수 있습니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

No [Yes, No]

디폴트로 **blocknonlogged**는 NO로 설정됩니다. 로깅되지 않는 조작이 허용되고 사용자는 로깅 축소와 연관된 성능 이점을 얻습니다. 그러나 특히 고가용성 재해 복구(HADR) 데이터베이스 환경에서는 이 구성과 연관된 몇 가지 잠재적인 단점이 있습니다. DB2 HADR 데이터베이스 환경은 데이터베이스 로그를 사용하여 기본 데이터베이스에서 대기 데이터베이스로 데이터를 복제합니다. 로깅되지 않는 조작은 기본 데이터

베이스에서는 허용되지만, 대기 데이터베이스에 복제되지 않습니다. 로깅되지 않는 조작이 대기 데이터베이스에 반영되기 원하는 경우 이 작업이 발생하도록 추가 단계를 수행해야 합니다. 예를 들어 온라인 분할 미러나 일시중단된 입출력 지원을 사용하여 로깅되지 않는 조작 이후 대기 데이터베이스를 재동기화할 수 있습니다.

blocknonlogged가 YES로 설정되는 경우, 다음 중 하나에 해당할 때 CREATE TABLE 및 ALTER TABLE문은 실패합니다.

- NOT LOGGED INITIALLY 매개변수가 지정되었습니다.
- NOT LOGGED 매개변수가 LOB 컬럼에 대해 지정되었습니다.
- 크기가 1GB보다 큰 CLOB, DBCLOB 또는 BLOB 컬럼이 지정됩니다.

catalogcache_sz - 카탈로그 캐시 크기

이 매개변수는 카탈로그 캐시가 데이터베이스 힙에서 사용할 수 있는 최대 스페이스를 페이지 단위로 지정합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
즉시

디폴트 [범위]
-1 [MAXAPPLS*5]

수치 단위
페이지(4KB)

할당 시기
데이터베이스가 초기화될 때

사용 가능한 시기
데이터베이스가 종료될 때

이 매개변수는 데이터베이스 공유 메모리에서 할당되며 시스템 카탈로그 정보를 캐시하는 데 사용됩니다. 파티션된 데이터베이스 시스템에서는 각 데이터베이스 파티션마다 하나의 카탈로그 캐시가 있습니다.

개별적 데이터베이스 파티션에서 카탈로그 정보를 캐시하면 데이터베이스 관리 프로그램이 시스템 카탈로그(또는 파티션된 데이터베이스 환경의 카탈로그 노드)에 액세스하여 이전에 검색한 정보를 가져올 필요가 없으므로 내부 오버헤드를 줄일 수 있습니다. 카탈로그 캐시를 사용하면 다음과 같은 전체 성능이 향상됩니다.

- 패키지 바인딩과 SQL 및 XQuery문 컴파일

- 데이터베이스 레벨 특권, 루틴 특권, 전역 변수 특권 및 역할 권한 부여와 관련된 조작
- 파티션된 데이터베이스 환경의 비카탈로그 노드에 연결된 응용프로그램

서버 또는 파티션된 데이터베이스 환경에서 디폴트(-1)인 경우 페이지 할당을 계산하는 데 사용되는 값은 *maxappls* 구성 매개변수에 지정된 값의 5배입니다. *maxappls*의 5배가 8 미만인 경우는 예외입니다. 이 경우 디폴트값 -1이 사용되면 *catalogcache_sz*가 8로 설정됩니다.

권장사항: 우선 디폴트값을 사용하고 이후 데이터베이스 시스템 모니터를 사용하여 조정하십시오. 이 매개변수를 조정하는 경우, 카탈로그 캐시에 여분의 메모리(버퍼 풀이나 패키지 캐시와 같이 다른 용도로 할당된 경우)를 예약하는 것이 더 효율적인지 고려해야 합니다.

이 매개변수를 조정하는 것은 워크로드가 짧은 기간에 많은 SQL 또는 XQuery 컴파일을 포함하고 그 후 컴파일이 거의 없거나 전혀 없는 경우에 특히 중요합니다. 캐시가 너무 크면 더 이상 사용되지 않는 정보 사본을 유지하느라 메모리가 낭비됩니다.

비카탈로그 노드에서 필요한 카탈로그 정보가 카탈로그 노드에서 항상 먼저 캐시되므로 파티션된 데이터베이스 환경에서는 카탈로그 노드의 *catalogcache_sz*를 더 크게 설정해야 하는지 고려하십시오.

cat_cache_lookups(카탈로그 캐시 찾아보기), *cat_cache_inserts*(카탈로그 캐시 삽입), *cat_cache_overflows*(카탈로그 캐시 오버플로우) 및 *cat_cache_size_top*(카탈로그 캐시 상위 워터 마크(water mark)) 모니터 요소를 사용하면 이 구성 매개변수를 조정해야 하는지 판별하는 데 도움이 됩니다.

주: 카탈로그 캐시는 파티션된 데이터베이스 환경의 모든 노드에 존재합니다. 각 노드에 대한 로컬 데이터베이스 구성 파일이 있으므로 각 노드의 *catalogcache_sz* 값은 로컬 카탈로그 캐시의 크기를 정의합니다. 효율적인 캐싱을 제공하고 오버플로우 시나리오를 피하기 위해 각 노드에서 *catalogcache_sz* 값을 명시적으로 설정하고 비카탈로그 노드의 *catalogcache_sz*를 카탈로그 노드보다 작게 설정할 수 있는지 고려하십시오. 비카탈로그 노드에서 캐시해야 하는 정보는 카탈로그 노드의 캐시에서 검색된다는 점을 기억하십시오. 그러므로 비카탈로그 노드의 카탈로그 캐시는 카탈로그 노드의 카탈로그 캐시에 있는 정보의 서브세트와 비슷합니다.

일반적으로 작업 단위에 여러 동적 SQL 또는 XQuery문이 있거나 많은 정적 SQL 또는 XQuery문을 포함하는 패키지를 바인드하는 경우 더 많은 캐시 공간이 필요합니다.

chngpgs_thresh - 변경된 페이지 임계값

이 매개변수는 비동기 페이지 클리너가 현재 활성화되지 않은 경우 시작되는 변경된 페이지 레벨(퍼센트)을 지정합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

60 [5 – 99]

수치 단위

퍼센트

데이터베이스 에이전트가 버퍼 풀의 스페이스를 요구하기 전에 비동기 페이지 클리너는 버퍼 풀에서 디스크로 변경된 페이지를 기록합니다. 그 결과 데이터베이스 에이전트는 버퍼 풀의 스페이스를 사용할 수 있도록 변경된 페이지가 작성되기를 기다릴 필요가 없습니다. 따라서 데이터베이스 응용프로그램의 전체 성능이 향상됩니다.

페이지 클리너가 시작되면 디스크에 기록할 페이지 목록을 빌드합니다. 디스크에 페이지를 모두 기록했으면 다시 비활성화되고 다음 트리거가 시작될 때까지 대기합니다.

DB2_USE_ALTERNATE_PAGE_CLEANING 레지스트리 변수가 설정된 경우(즉, 페이지 정리 대체 방법이 사용되는 경우) *chngpgs_thresh* 매개변수가 적용되지 않으며 데이터베이스 관리 프로그램이 버퍼 풀에서 유지되는 더티 페이지 수를 자동으로 판별합니다.

권장사항: 갱신 트랜잭션 워크로드가 많은 데이터베이스의 경우 일반적으로 매개변수 값을 디폴트값과 같거나 보다 작게 설정하면 버퍼 풀에 정리 페이지가 충분하게 됩니다. 퍼센트가 디폴트보다 큰 경우 데이터베이스에 매우 큰 테이블이 적게 있으면 성능에 도움이 될 수 있습니다.

codepage - 데이터베이스의 코드 페이지

이 매개변수는 데이터베이스를 작성하는 데 사용된 코드 페이지를 표시합니다. *codepage* 매개변수는 *codeset* 매개변수를 기초로 파생됩니다.

구성 유형

데이터베이스

매개변수 유형

정보용

codeset - 데이터베이스의 코드 세트

이 매개변수는 데이터베이스를 작성하는 데 사용된 코드 세트를 표시합니다. 코드 세트는 데이터베이스 관리 프로그램이 *codepage* 매개변수 값을 판별하는 데 사용합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

collate_info - 조합 정보

이 매개변수는 데이터베이스의 조합 시퀀스를 판별합니다. 언어 인식 조합의 경우 처음 256바이트에 조합 이름의 문자열 표현(예: ″SYSTEM_819_US″)이 있습니다.

이 매개변수는 db2CfgGet API를 사용해야만 표시할 수 있습니다. 명령행 처리기 또는 제어 센터를 통해서는 표시할 수 **없습니다**.

구성 유형

데이터베이스

매개변수 유형

정보용

이 매개변수는 260바이트의 데이터베이스 조합 정보를 제공합니다. 처음 256바이트는 데이터베이스 조합 시퀀스를 지정하며 『n』바이트에는 데이터베이스의 코드 페이지에서 기본 10진수 표현이 『n』인 코드 포인트의 정렬 가중치가 있습니다.

마지막 4바이트에는 조합 시퀀스 유형에 대한 내부 정보가 있습니다. 매개변수의 마지막 4바이트는 정수입니다. 정수는 플랫폼의 엔디안 순서에 따라 다릅니다. 사용할 수 있는 값은 다음과 같습니다.

- **0** – 시퀀스에 고유하지 않은 가중치가 있습니다.
- **1** – 시퀀스에 고유한 모든 가중치가 있습니다.
- **2** – 시퀀스는 동일 시퀀스이며 문자열을 바이트별로 비교합니다.
- **3** – 시퀀스는 NLSCHAR이며 TIS620-1(코드 페이지 874) 태국어 데이터베이스에서 문자를 정렬하는 데 사용됩니다.
- **4** – 시퀀스는 IDENTITY_16BIT이며 Unicode Technical Consortium 웹 사이트 http://www.unicode.org에서 사용 가능한 Unicode Technical Report #26에 지정된 『CESU-8 Compatibility Encoding Scheme for UTF–16: 8–bit』 알고리즘을 구현합니다.
- **X‘8001’** – 시퀀스는 UCA400_NO이며 정규화를 내재적으로 ON으로 설정하여 Unicode Standard 버전 4.00에 따라 Unicode Collation® Algorithm(UCA)을 구현합니다.
- **X‘8002’** – 시퀀스는 UCA400_LTH이며 Unicode Standard 버전 4.00에 따라 Unicode Collation Algorithm(UCA)을 구현하며 Royal Thai Dictonary 순서에 따라 모든 태국어 문자를 정렬합니다.

- **X'8003'** – 시퀀스는 UCA400_LSK이며 Unicode Standard 버전 4.00에 따라 Unicode Collation Algorithm(UCA)을 구현하고 모든 슬로바키아어 문자를 적절하게 정렬합니다.

주: 언어 인식 조합의 경우 처음 256바이트에 조합 이름의 문자열 표현이 있습니다.

이 내부 유형 정보를 사용하는 경우 다른 플랫폼에서 데이터베이스에 대한 정보를 검색할 때 바이트 리버설을 고려해야 합니다.

데이터베이스 작성 시 조합 시퀀스를 지정할 수 있습니다.

country/region - 데이터베이스 지역 코드

이 매개변수는 데이터베이스를 작성하는 데 사용된 *지역* 코드를 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

database_consistent - 데이터베이스 일관성

이 매개변수는 데이터베이스가 일관성 있는 상태인지 여부를 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

YES는 데이터의 일관성을 위해 모든 트랜잭션이 커미트 또는 롤백되었음을 표시합니다. 데이터베이스가 일관성이 있을 때 시스템이 『손상』된 경우 데이터베이스를 사용할 수 있도록 특별한 조치를 수행할 필요가 없습니다.

NO는 데이터베이스에서 트랜잭션이 보류되었거나 일부 기타 태스크가 보류되었으며 이 시점에 데이터가 일관성이 없음을 표시합니다. 데이터베이스가 일관성이 없을 때 시스템이 『손상』된 경우 RESTART DATABASE 명령을 사용하여 데이터베이스를 재시작해야만 데이터베이스가 사용 가능해집니다.

database_level - 데이터베이스 릴리스 레벨

이 매개변수는 데이터베이스를 사용할 수 있는 데이터베이스 관리 프로그램의 릴리스 레벨을 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

데이터베이스 업그레이드가 완료되지 않았거나 실패한 경우 이 매개변수는 업그레이드 전 데이터베이스의 릴리스 레벨을 반영하며 *release* 매개변수(데이터베이스 구성 파일의 릴리스 레벨)와 다를 수 있습니다. 그렇지 않으면 *database_level*의 값은 *release* 매개변수의 값과 동일합니다.

database_memory - 데이터베이스 공유 메모리 크기

이 매개변수는 데이터베이스 공유 메모리 영역에 예약된 메모리 양을 지정합니다. 이 양이 개별 메모리 매개변수(예: locklist, utility heap, bufferpools 등)에서 계산된 양보다 적은 경우, 더 큰 양이 사용됩니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

Automatic [Computed, 0 - 4 294 967 295]

수치 단위

페이지(4KB)

할당 시기

데이터베이스가 활성화될 때

사용 가능한 시기

데이터베이스가 비활성화될 때

이 매개변수를 AUTOMATIC으로 설정하면 자체 성능 조정이 가능합니다. 자체 성능 조정이 사용되는 경우, 메모리 조정 프로그램은 데이터베이스에 대한 전체 메모리 요구사항을 판별하고 현재 데이터베이스 요구사항에 따라 데이터베이스 공유 메모리에 할당되는 메모리의 양을 늘리거나 줄입니다. 예를 들어, 현재 데이터베이스 요구사항이 높고 시스템에 충분한 여유 공간이 있는 경우, 데이터베이스 공유 메모리가 더 많은 메모리를 사용합니다. 데이터베이스 메모리 요구사항이 줄어들거나 시스템의 여유 메모리 양이 너무 적어지면 일부 데이터베이스 공유 메모리가 해제됩니다.

메모리 조정 프로그램은 인스턴스에 추가적인 메모리를 제공하는 경우 계산되는 이점에 따라 항상 최소량의 메모리를 여유 공간으로 유지합니다. 인스턴스에 메모리를 더 추가하는 경우 이점이 많이 있으면 메모리 조정 프로그램은 여유 메모리 양을 더 낮게 유지합니다. 이점이 별로 없으면 더 많은 여유 메모리를 유지합니다. 그러면 데이터베이스는 시스템 메모리의 분산에 협력할 수 있습니다.

메모리 조정 프로그램은 여러 메모리 사용자 간에 메모리 자원을 교환하므로 자체 성능 조정이 활성화되려면 최소 두 개의 메모리 사용자가 사용 가능해야 합니다.

이 구성 매개변수의 자동 성능 조정은 데이터베이스의 자체 성능 조정 메모리가 사용되는 경우(*self_tuning_mem* 구성 매개변수가 ON으로 설정된 경우)에만 발생합니다.

이 매개변수의 관리를 간단히 하기 위해 COMPUTED 설정은 데이터베이스 관리 프로그램에게 필요한 메모리의 양을 계산하도록 지시한 후 데이터베이스 활성화 시간에 할당합니다. 데이터베이스 관리 프로그램은 또한 데이터베이스 공유 메모리 영역의 힙에서 힙이 구성된 *크기*를 초과할 때마다 최대 메모리 요구사항을 충족하는 일부 메모리를 추가로 할당합니다. 동적 구성 갱신과 같은 다른 조작도 이 추가 메모리에 액세스할 수 있어야 합니다. db2pd 명령과 -memsets 옵션을 사용하여 데이터베이스 공유 메모리 영역에서 사용되지 않고 남아 있는 메모리의 양을 모니터할 수 있습니다.

권장사항: 이 값은 일반적으로 AUTOMATIC으로 남아 있습니다. AUTOMATIC 설정을 지원하지 않는 환경에서는 COMPUTED로 설정되어야 합니다. 예를 들어, 추가 메모리를 사용하여 새 버퍼 풀을 작성하거나 기존 버퍼 풀의 *크기*를 늘릴 수 있습니다.

주: 버전 9.5에서 *database_memory* 구성 매개변수를 AUTOMATIC으로 설정한 경우 초기 데이터베이스 공유 메모리 할당은 데이터베이스에 대해 정의된 모든 힙 및 버퍼 풀의 구성된 *크기*이며, 필요에 따라 메모리가 증가합니다. *database_memory*가 특정 값으로 설정된 경우, 처음에 데이터베이스가 활성화될 때 요청된 메모리 양이 할당됩니다. 운영 체제에서 메모리의 초기 양을 할당할 수 없는 경우 또는 *instance_memory* 한계를 초과한 경우에는 SQL1084C 오류(공유 메모리 세그먼트를 할당할 수 없음)가 발생하여 데이터베이스 활성화가 실패합니다.

DB2 메모리 사용량 제어:

*instance_memory*가 AUTOMATIC으로 설정되는 경우 인스턴스에 대한 총 메모리 사용량의 고정 상한은 인스턴스 시작 시(db2start) 설정됩니다. 데이터베이스 관리 프로그램의 실제 메모리 사용량은 워크로드에 따라 다양합니다. 런타임에 자체 성능 조정 메모리 관리자가 *database_memory* 조정을 수행할 수 있는 경우(새 데이터베이스의 디폴트로), 자체 성능 조정 메모리 관리자는 기능 메모리 요구사항에 사용 가능한 *instance_memory*를 충분히 유지하면서 데이터베이스 공유 메모리 세트에서 시스템의 여유 공간 실제 메모리에 따라 성능에 중요한 힙의 *크기*를 갱신합니다. 자세한 정보는 *instance_memory* 구성 매개변수를 참조하십시오.

일부 Linux[1] 커널의 한계:

일부 Linux 커널에서는 운영 체제 한계 때문에 자체 성능 조정 메모리 관리자가 현재 *database_memory*를 AUTOMATIC으로 설정할 수 없습니다. 그러나 이 설정은 현재 *instance_memory*가 AUTOMATIC이 아닌 특정 값으로 설정되는 경우에만 이 커널에서 허용됩니다. *database_memory*가 AUTOMATIC으로 설정되고 *instance_memory*가 나중에 다시 AUTOMATIC으로 설정되면 *database_memory* 구성 매개변수는 다음 데이터베이스 활성화 시 자동으로 COMPUTED로 갱신됩니다. 일부 데이터베이스가 이미 활성인 경우 자체 성능 조정 메모리 관리자는 전체 *database_memory* 크기에 대한 조정을 중지합니다.

[1]Linux의 경우, 이 매개변수는 RHEL5 및 SUSE 10 SP1 이상에서 AUTOMATIC 설정을 지원합니다. 커널이 이 기능을 지원하지 않는 경우, 유효성 확인된 다른 모든 Linux 분산은 COMPUTED로 돌아갑니다.

dbheap - 데이터베이스 힙

이 매개변수는 데이터베이스 힙이 사용하는 최대 메모리를 판별합니다.

버전 9.5에서 이 데이터베이스 구성 매개변수의 디폴트값은 AUTOMATIC이며 *database_memory* 한계 또는 *instance_memory* 한계에 접근할 때까지 필요에 따라 데이터베이스 힙이 증가할 수 있음을 의미합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

Automatic [32 - 524 288]

수치 단위

페이지(4KB)

할당 시기

데이터베이스가 활성화될 때

사용 가능한 시기

데이터베이스가 비활성화될 때

데이터베이스 당 하나의 데이터베이스 힙이 있으며 데이터베이스 관리 프로그램이 데이터베이스에 연결된 모든 응용프로그램 대신 이를 사용합니다. 테이블, 인덱스, 테이블 스페이스 및 버퍼 풀에 대한 제어 블록 정보가 있습니다. 또한 로그 버퍼(*logbufsz*) 및 유

틸리티에서 사용하는 임시 메모리의 스페이스가 있습니다. 따라서 힙의 *크기*는 여러 변수에 따라 다릅니다. 제어 블록 정보는 데이터베이스에서 모든 응용프로그램의 연결이 끊어질 때까지 힙에 보존됩니다.

데이터베이스 관리 프로그램이 시작되는 데 필요한 최소의 양이 처음 연결할 때 할당됩니다. 구성된 상한에 접근할 때까지 또는 AUTOMATIC으로 설정되었으면 모든 *database_memory*나 *instance_memory* 또는 둘 다에 대한 메모리를 소모할 때까지 필요하면 데이터 영역을 확장합니다.

dbheap 구성 매개변수에 지정할 값을 결정할 때 다음의 공식을 대략적인 지침으로 사용할 수 있습니다.

```
테이블 스페이스당 10K + 테이블당 4K + (1K + 4*사용된 Extent 수),
  RCT(range clustered table)당
```

구성하는 *dbheap* 값은 할당된 데이터베이스 힙의 일부분만 나타냅니다. 데이터베이스 힙은 전역 데이터베이스 메모리 요구사항을 충족시키는 데 사용되는 주기억장치 영역입니다. *dbheap* 값 외에 데이터베이스의 시작에 필요한 기본 할당을 포함하여 *크기*를 조정합니다. 메모리 추적 프로그램, 스냅샷 모니터 및 db2pd와 같이 메모리 사용을 보고하는 도구는 이 대형 데이터베이스 힙의 통계를 보고합니다. *dbheap* 구성 매개변수가 나타내는 할당에 대한 개별 추적은 없습니다. 따라서 이러한 도구에서 보고한 데이터베이스 힙 메모리 사용에 대한 통계가 *dbheap* 매개변수에 대해 구성된 값을 초과하기 쉽습니다.

데이터베이스 시스템 모니터를 사용하면 *db_heap_top*(할당된 최대 데이터베이스 힙) 요소를 사용하여 데이터베이스 힙에 사용된 최대 메모리 양을 추적할 수 있습니다.

주:

- 워크로드 관리(WLM) 작업 클래스 세트 및 작업 조치 세트는 데이터베이스 힙에 저장됩니다. 그러나 이 경우 메모리의 일부분만 사용합니다.
- 트러스트된 컨텍스트, 워크로드 관리 및 감사 규정 정보는 신속한 처리를 위해 메모리에 캐싱됩니다. 이 메모리는 데이터베이스 힙에서 할당됩니다. 따라서 사용자 정의 트러스트된 컨텍스트, 워크로드 관리 및 감사 규정 오브젝트는 데이터베이스 힙에 대한 추가 메모리 요구사항을 지정합니다. 이 경우 데이터베이스 관리 프로그램이 데이터베이스 힙 *크기*를 자동으로 관리하도록 데이터베이스 힙 구성을 AUTOMATIC으로 설정하는 것이 좋습니다.

db_mem_thresh - 데이터베이스 메모리 임계값

이 매개변수는 데이터베이스 관리 프로그램이 이전에 커미트된 메모리 페이지를 운영 체제로 다시 해제하도록 허용한, 커미트되었으나 사용되지 않은 데이터베이스 공유 메모리의 최대 퍼센트를 나타냅니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

10 [0 - 100]

수치 단위

퍼센트

이 데이터베이스 구성 매개변수는 데이터베이스 관리 프로그램이 사용되지 않은 초과 데이터베이스 공유 메모리를 처리하는 방법과 관련됩니다. 일반적으로 메모리 페이지는 프로세스가 접촉할 때 커미트됩니다. 즉 메모리의 한 페이지는 운영 체제가 할당하며 실제 메모리 또는 디스크의 페이지 파일 스페이스를 차지합니다. 데이터베이스 워크로드에 따라 하루 중 특정 시간에 데이터베이스 공유 메모리 요구사항이 최대가 될 수 있습니다. 운영 체제가 해당 최대 요구사항에 맞게 충분한 메모리를 커미트한 경우, 최대 메모리 요구사항이 지난 후에도 메모리가 커미트된 채로 남아 있습니다.

승인할 수 있는 값의 범위는 0(사용하지 않는 모든 데이터베이스 공유 메모리를 즉시 해제) - 100(사용하지 않는 데이터베이스 공유 메모리를 해제하지 않음)입니다. 디폴트는 10(데이터베이스 공유 메모리의 10% 이상이 현재 사용되지 않는 경우에만 사용되지 않는 메모리를 해제)이며 대부분의 워크로드에 적합합니다.

이 구성 매개변수는 동적으로 갱신할 수 있습니다. 이 값을 너무 낮게 설정하면 상자에 과도한 메모리 스레싱이 발생(메모리 페이지가 지속적으로 커미트된 후 해제됨)하며 값을 너무 높게 설정하면 데이터베이스 관리 프로그램이 다른 프로세스가 사용하도록 데이터베이스 공유 메모리를 다시 운영 체제에 반환할 수 없으므로 이 매개변수를 갱신하는 경우에는 주의해야 합니다.

데이터베이스 공유 메모리 영역이 DB2_PINNED_BP 레지스트리 변수를 통해 고정되거나 DB2_LARGE_PAGE_MEM 레지스트리 변수를 통해 큰 페이지로 구성되거나 DB2MEMDISCLAIM 레지스트리 변수를 통해 메모리 해제가 명시적으로 사용 불가능하게 된 경우 이 구성 매개변수는 무시됩니다. (즉 사용되지 않는 데이터베이스 공유 메모리 페이지가 커미트된 상태로 남아 있습니다.)

Linux의 일부 버전은 운영 체제에 반환하기 위한 공유 메모리 세그먼트의 부분 해제를 지원하지 않습니다. 이러한 플랫폼에서는 이 매개변수가 무시됩니다.

date_compat - 날짜 호환성 데이터베이스 구성 매개변수

이 매개변수는 TIMESTAMP(0) 데이터 유형에 연결된 DATE 호환성 시맨틱이 연결된 데이터베이스에 적용되는지 여부를 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

이 값은 데이터베이스를 작성할 때 DATE 지원의 DB2_COMPATIBILITY_VECTOR 레지스트리 변수 설정에 따라 결정됩니다. 이 값은 변경할 수 없습니다.

decflt_rounding - 10진 부동 소수점 근사값 구성 매개변수

이 매개변수를 사용하여 10진 부동 소수점(DECFLOAT)의 근사값 모드를 지정할 수 있습니다. 근사값 모드는 서버와 LOAD에서 10진 부동 소수점에 영향을 미칩니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

아래의 804 페이지의 『decflt_rounding 변경 결과』를 참조하십시오.

디폴트 [범위]

ROUND_HALF_EVEN [ROUND_CEILING, ROUND_FLOOR, ROUND_HALF_UP, ROUND_DOWN]

DB2는 다섯 가지 IEEE 준수 10진 부동 소수점 근사값 모드를 지원합니다. 근사값 모드는 결과가 정밀도를 초과하는 경우 계산 결과를 반올림하는 방법을 지정합니다. 모든 근사값 모드의 정의는 다음과 같습니다.

ROUND_CEILING

+무한대로 올림. 버린 숫자가 모두 0이거나 부호가 음수인 경우 결과는 변경되지 않습니다. 그렇지 않으면 결과 계수는 1만큼 증가합니다(올림).

ROUND_FLOOR

-무한대로 올림. 버린 숫자가 모두 0이거나 부호가 양수인 경우 결과는 변경되지 않습니다. 그렇지 않으면 부호가 음수이고 결과 계수는 1만큼 증가합니다.

ROUND_HALF_UP

가장 가까운 값으로 반올림. 거리가 같은 경우 1 올립니다. 버린 숫자가 다음 왼쪽 위치에서 값 1의 반(0.5)보다 크거나 같은 경우 결과 계수는 1만큼 증가해야 합니다(반올림). 그렇지 않으면 버린 숫자(0.5 이하)가 무시됩니다.

ROUND_HALF_EVEN

가장 가까운 값으로 반올림. 거리가 같은 경우에는 마지막 자리가 짝수가 되도록 반올림합니다. 버린 숫자가 다음 왼쪽 위치에서 값 1의 반(0.5)보다 크거나 같은 경우 결과 계수는 1만큼 증가해야 합니다(반올림). 반 미만을 나타내는 경우에는 결과 계수가 조정되지 않으며, 버린 숫자는 무시됩니다. 또는 정확히 반을 나타내는 경우 결과 계수는 맨 오른쪽 숫자가 짝수이면 변경되지 않고 맨 오른쪽 숫자가 홀수이면 1만큼 증가되어 짝수로 만듭니다. 이 근사값 모드는 IEEE 10진 부동 소수점 지정의 디폴트 근사값 모드이며 DB2 제품의 디폴트 근사값 모드입니다.

ROUND_DOWN

0으로 버림(자름). 버린 숫자는 무시됩니다.

표 75는 여러 근사값 모드에서 12.341, 12.345, 12.349, 12.355 및 -12.345를 각각 4자리로 반올림한 결과를 표시합니다.

표 75. 10진 부동 소수점 근사값 모드

근사값 모드	**12.341**	**12.345**	**12.349**	**12.355**	**-12.345**
ROUND_DOWN	12.34	12.34	12.34	12.35	-12.34
ROUND_HALF_UP	12.34	12.35	12.35	12.36	-12.35
ROUND_HALF_EVEN	12.34	12.34	12.35	12.36	-12.34
ROUND_FLOOR	12.34	12.34	12.34	12.35	-12.35
ROUND_CEILING	12.35	12.35	12.35	12.36	-12.34

decflt_rounding 변경 결과

- 이전에 작성한 구체화된 쿼리 테이블(MQT)에는 새 근사값 모드로 작성한 결과와 다른 결과가 포함될 수 있습니다. 이 문제점을 정정하기 위해 잠재적으로 영향을 받은 MQT를 새로 고치십시오.
- 트리거 결과는 새 근사값 모드의 영향을 받을 수 있습니다. 변경해도 이미 기록된 데이터에는 영향을 미치지 않습니다.
- 데이터를 테이블에 삽입하도록 허용한 제한조건이 재평가되면 동일한 데이터를 거부할 수 있습니다. 마찬가지로 데이터를 테이블에 삽입하도록 허용하지 않은 제한조건이 재평가되면 동일한 데이터를 승인할 수 있습니다. SET INTEGRITY 문을 사용하여 이러한 문제를 점검하고 정정하십시오. 한 행은 decflt_rounding으로 변경되기 전에 삽입되고 다른 행은 변경된 후에 삽입된 경우, decflt_rounding에 계산이 종속되는 생성된 컬럼의 값은 생성된 컬럼 값을 제외한 두 동등한 행에서 다를 수 있습니다.
- 근사값 모드는 섹션으로 컴파일되지 않습니다. 그러므로 정적 SQL은 *decflt_rounding*을 변경한 후에 재컴파일할 필요가 없습니다.

주: 이 구성 매개변수의 값은 동적으로 변경되지 않지만 모든 응용프로그램이 데이터베이스에서 연결이 끊어진 후에만 유효하게 됩니다. 데이터베이스가 활성화된 경우에는 비활성화되어야 합니다.

dft_degree - 디폴트 등급

이 매개변수는 CURRENT DEGREE 특수 레지스터 및 DEGREE 바인드 옵션의 디폴트값을 지정합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

연결

디폴트 [범위]

1 [-1(ANY), 1 - 32 767]

디폴트값은 1입니다.

값 1은 파티션 내 병렬 처리가 없음을 의미합니다. 값 -1(또는 ANY)은 옵티마이저가 프로세서 수 및 쿼리 유형에 따라 파티션 내 병렬 처리 등급을 판별함을 의미합니다.

SQL문의 파티션 내 병렬 처리 등급은 컴파일 시간에 명령문에서 CURRENT DEGREE 특수 레지스터 또는 DEGREE 바인드 옵션을 사용하여 지정됩니다. 활성 응용프로그램에 대한 파티션 내 병렬 처리의 최대 런타임 등급은 SET RUNTIME DEGREE 명령을 사용하여 지정됩니다. 병렬 처리의 최대 쿼리 등급(*max_querydegree*) 구성 매개변수는 모든 SQL 쿼리에 대한 파티션 내 병렬 처리의 최대 쿼리 등급을 지정합니다.

실제 런타임 등급은 다음 중 가장 낮은 것이 사용됩니다.

- *max_querydegree* 구성 매개변수
- 응용프로그램 런타임 등급
- SQL문 컴파일 등급

dft_extent_sz - 테이블 스페이스의 디폴트 Extent 크기

이 매개변수는 테이블 스페이스의 디폴트 Extent 크기를 설정합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

32 [2 – 256]

수치 단위

페이지

테이블 스페이스를 작성할 때 선택적으로 EXTENTSIZE n을 지정할 수 있습니다. 여기서 n은 Extent 크기입니다. CREATE TABLESPACE문에 Extent 크기를 지정하지 않은 경우 데이터베이스 관리 프로그램은 이 매개변수에 지정된 값을 사용합니다.

권장사항: 대부분의 경우 테이블 스페이스를 작성할 때 Extent 크기를 명시적으로 지정합니다. 이 매개변수의 값을 선택하기 전에 CREATE TABLESPACE문의 Extent 크기를 명시적으로 선택하는 방식을 알아야 합니다.

dft_loadrec_ses - 디폴트 로그 복구 세션 수

이 매개변수는 테이블 로드 복구 중 사용되는 디폴트 세션 수를 지정합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

1 [1 - 30 000]

수치 단위

카운터

이 값은 로드 사본을 검색하는 데 사용되는 최적의 입출력 세션 수로 설정해야 합니다. 로드 사본의 검색은 리스토어와 비슷한 조작입니다. 환경 변수 DB2LOADREC에 지정된 사본 위치 파일의 항목을 통해 이 매개변수를 겹쳐쓸 수 있습니다.

로드 검색에 사용되는 디폴트 버퍼 수는 이 매개변수 값에 2를 더한 값입니다. 또한 사본 위치 파일에서 버퍼 수를 겹쳐쓸 수 있습니다.

이 매개변수는 롤 포워드 복구가 사용 가능한 경우에만 적용됩니다.

dft_mttb_types - 최적화를 위해 유지되는 디폴트 테이블 유형

이 매개변수는 CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 특수 레지스터의 디폴트값을 지정합니다. 이 레지스터의 값은 쿼리를 최적화할 때 사용할 지연된 구체화된 쿼리 테이블 유형을 판별합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

SYSTEM [ALL, NONE, FEDERATED_TOOL, SYSTEM, USER 또는 값 목록]

값 목록은 쉼표로 구분하여 지정할 수 있습니다(예: 'USER,FEDERATED_TOOL'). ALL 또는 NONE을 다른 값과 함께 나열할 수 없으며 동일한 값을 두 번 이상 지정할 수 없습니다. db2CfgSet 및 db2CfgGet API에 대해 사용하는 경우 승인할 수 있는 매개변수 값은 8(ALL), 4(NONE), 16(FEDERATED_TOOL), 1(SYSTEM) 및 2(USER)입니다. 비트 방향의 OR를 사용하여 다중 값을 함께 지정할 수 있습니다. (예: 18은 USER,FEDERATED_TOOL과 같습니다.) 앞에서 설명했듯이, 값 4와 8은 다른 값과 함께 사용할 수 없습니다.

dft_prefetch_sz - 디폴트 프리페치 크기

이 매개변수는 테이블 스페이스의 디폴트 프리페치 크기를 설정합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

Automatic [0 - 32 767]

수치 단위

페이지

테이블 스페이스를 작성할 때 선택적으로 PREFETCHSIZE *n*을 지정할 수 있습니다. 여기서 *n*은 프리페치를 수행하는 경우 데이터베이스 관리 프로그램이 읽는 페이지 수를 표시합니다. CREATE TABLESPACE문을 호출할 때 프리페치 크기를 지정하지 않으면 데이터베이스 관리 프로그램이 *dft_prefetch_sz* 매개변수의 현재 값을 사용합니다.

AUTOMATIC DFT_PREFETCH_SZ를 사용하여 테이블 스페이스가 작성된 경우 테이블 스페이스의 프리페치 *크기*는 AUTOMATIC이 되며 DB2가 다음의 등식을 사용하여 테이블 스페이스의 프리페치 *크기*를 자동으로 계산하고 갱신합니다.

```
프리페치 크기 = (컨테이너 수) * (실제 스핀들 수) * Extent 크기
```

여기서 실제 스핀들 수는 디폴트 1을 사용하며 DB2 레지스트리 변수 DB2_PARALLEL_IO를 통해 지정할 수 있습니다. 이 계산은 다음과 같은 경우에 수행됩니다.

- 데이터베이스 시작 시
- AUTOMATIC 프리페치 *크기*를 사용하여 테이블 스페이스를 처음 작성하는 경우
- ALTER TABLESPACE문을 실행하여 테이블 스페이스의 컨테이너 수가 변경된 경우
- ALTER TABLESPACE문을 실행하여 테이블 스페이스의 프리페치 *크기*가 AUTOMATIC으로 갱신된 경우

ALTER TABLESPACE문을 호출하여 프리페치 *크기*를 수동으로 갱신한 후 즉시 프리페치 *크기*의 AUTOMATIC 상태를 설정하거나 해제할 수 있습니다.

권장사항: 시스템 모니터링 도구를 사용하여 시스템이 입출력 대기 중인 동안 CPU가 유휴 상태인지 판별할 수 있습니다. 이 매개변수의 값을 늘리면 사용 중인 테이블 스페이스에 프리페치 *크기*가 정의되지 않은 경우에 도움이 됩니다.

이 매개변수는 전체 데이터베이스의 디폴트를 제공하며 데이터베이스의 모든 테이블 스페이스에 적합한 것은 아닙니다. 예를 들어, 32 값은 Extent *크기*가 32페이지인 테이블 스페이스에는 적합하지만 Extent *크기*가 25페이지인 테이블 스페이스에는 적합하지 않습니다. 원래 각 테이블 스페이스의 프리페치 *크기*를 명시적으로 설정해야 합니다.

디폴트 Extent *크기*(*dft_extent_sz*)로 정의된 테이블 스페이스의 입출력을 최소화하려면 이 매개변수를 *dft_extent_sz* 매개변수 값의 인수 또는 전체 배수로 설정해야 합니다. 예를 들어, *dft_extent_sz* 매개변수가 32인 경우 *dft_prefetch_sz*를 16(32의 분수) 또는 64(32의 전체 배수)로 설정할 수 있습니다. 프리페치 *크기*가 Extent *크기*의 배수인 경우 다음 조건이 참이면 데이터베이스 관리 프로그램은 입출력을 병렬식으로 수행할 수 있습니다.

- 프리페치 중인 Extent가 다른 물리 디바이스에 있습니다.
- 여러 입출력 서버가 구성되었습니다(*num_ioservers*).

dft_queryopt - 디폴트 쿼리 최적화 클래스

쿼리 최적화 클래스는 SQL 및 XQuery 쿼리를 컴파일할 때 옵티마이저가 다른 등급의 최적화를 사용하도록 지시하는 데 사용됩니다. 이 매개변수는 바인드 명령에서 SET

CURRENT QUERY OPTIMIZATION 문과 QUERYOPT 옵션을 둘 다 사용하지 않는 경우 사용되는 디폴트 쿼리 최적화 클래스를 설정하여 추가적인 유연성을 제공합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

연결

디폴트 [범위]

5 [0 — 9]

수치 단위

쿼리 최적화 클래스(아래 참조)

현재 정의된 쿼리 최적화 클래스는 다음과 같습니다.

- 0 - 최소 쿼리 최적화
- 1 - DB2 버전 1과 대체로 비교 가능
- 2 - 가벼운 최적화
- 3 - 중급 쿼리 최적화
- 5 - 액세스 플랜 선택 확장 노력을 제한하는 원리가 있는 고급 쿼리 최적화. 디폴트입니다.
- 7 - 고급 쿼리 최적화
- 9 - 최대 쿼리 최적화

dft_refresh_age - 디폴트 새로 고침 유효 기간

이 매개변수는 특정 REFRESH DEFERRED 구체화된 쿼리 테이블에서 REFRESH TABLE 문이 처리된 이후 최대 시간 지속 기간을 나타냅니다. 이 시간 제한을 초과하는 경우 구체화된 쿼리 테이블은 구체화된 쿼리 테이블을 새로 고칠 때까지 쿼리를 충족하는 데 사용되지 않습니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

0 [0, 99999999999999 (ANY)]

이 매개변수에는 CURRENT REFRESH AGE 특수 레지스터가 지정되지 않은 경우 REFRESH AGE에 사용되는 디폴트값이 있습니다. 이 매개변수는 DECIMAL(20,6) 데이터 유형으로 시간소인 지속기간 값을 지정합니다. CURRENT REFRESH AGE의 값이 99999999999999 (ANY)이고 QUERY OPTIMIZATION 클래스의 값이 2나 5 또는 그 이상인 경우 REFRESH DEFERRED 구체화된 쿼리 테이블은 동적 쿼리의 처리를 최적화하는 것으로 간주됩니다.

dft_sqlmathwarn - 산술 예외에 따라 계속

이 매개변수는 SQL문 컴파일 중 산술 오류 및 검색 변환 오류의 처리를 오류 또는 경고로 판별하는 디폴트값을 설정합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

No [No, Yes]

정적 SQL문의 경우 이 매개변수의 값은 바인드 시 패키지와 연관됩니다. 동적 SQL DML문의 경우 명령문이 준비되면 이 매개변수의 값을 사용합니다.

주의: 데이터베이스에 대한 *dft_sqlmathwarn* 값을 변경하는 경우 연산식이 포함된 점검 제한조건, 트리거 및 뷰의 동작이 변경될 수 있습니다. 이 경우 데이터베이스의 데이터 무결성에 영향을 줄 수 있습니다. 새 산술 예외 처리 동작이 점검 제한조건, 트리거 및 뷰에 어떠한 영향을 줄 수 있는지 신중하게 평가한 후에만 데이터베이스에 대한 *dft_sqlmathwarn*의 설정을 변경해야 합니다. 이와 같이 변경한 후 나중에 변경할 때에도 동일하게 신중히 평가해야 합니다.

한 예로 나눗셈 산술 연산이 포함된 다음의 점검 제한조건을 참조하십시오.

```
A/B > 0
```

*dft_sqlmathwarn*이 『No』이고 B=0인 상태에서 INSERT를 시도한 경우 영(0)으로 나누기는 산술 오류로 처리됩니다. DB2가 제한조건을 점검할 수 없으므로 삽입 조작이 실패합니다. *dft_sqlmathwarn*이 『Yes』로 변경된 경우 영(0)으로 나누기는 NULL 결과의 산술 경고로 처리됩니다. NULL 결과는 술어를 UNKNOWN으로 평가하고 삽입 조작이 성공합니다. *dft_sqlmathwarn*이 『No』로 다시 변경된 경우 영(0)으로 나누기 오류는 DB2가 제한조건을 평가하지 못하도록 방지하므로 동일한 행을 삽입하는 시도가 실패합니다. *dft_sqlmathwarn*이 『Yes』인 경우 B=0인 상태로 삽입된 행은 테이블에 그대로 남아 있으며 선택이 가능합니다. 제한조건 재평가가 필요하지 않은 행의 갱신은 성공하지만 제한조건을 평가하는 행의 갱신은 실패합니다.

*dft_sqlmathwarn*을 『No』에서 『Yes』로 변경하기 전에 연산식에서 널(Null)을 명시적으로 처리하도록 제한조건을 재작성해야 합니다. 예:

```
( A/B > 0 ) AND ( CASE
                    WHEN A IS NULL THEN 1
                    WHEN B IS NULL THEN 1
                    WHEN A/B IS NULL THEN 0
                    ELSE 1
                    END
                  = 1 )
```

A 및 B가 모두 널(NULL) 입력 가능한 경우 이와 같은 예를 사용할 수 있습니다. 또한 A 또는 B가 널(NULL) 입력 가능하지 않은 경우 해당 IS NULL WHEN절을 제거할 수 있습니다.

*dft_sqlmathwarn*을 『Yes』에서 『No』로 변경하기 전에 먼저 예를 들어, 다음과 같은 술어를 사용하여 일치하지 않게 되는 데이터가 있는지 점검해야 합니다.

```
WHERE A IS NOT NULL AND B IS NOT NULL AND A/B IS NULL
```

일치하지 않는 행이 분리된 경우 *dft_sqlmathwarn*을 변경하기 전에 불일치를 정정하는 해당 조치를 수행해야 합니다. 또한 변경 후 연산식을 사용하여 제한조건을 수동으로 다시 점검할 수 있습니다. 이 경우 먼저 영향을 받은 테이블을 점검 보류 상태로 전환한 후(SET CONSTRAINTS문의 OFF절 사용) 테이블을 점검하도록 요청하십시오(SET CONSTRAINTS문의 IMMEDIATE CHECKED절 사용). 일치하지 않는 데이터는 산술 오류로 표시되며 제한조건의 평가를 방지합니다.

권장사항: 산술 예외가 포함된 쿼리를 특별히 처리해야 하는 경우가 아니면 디폴트 설정 no를 사용하십시오. 그런 다음 값 yes를 지정하십시오. 다른 데이터베이스 관리 프로그램에서 발생하는 산술 예외와 관계없이 결과를 제공하는 SQL문을 처리 중인 경우 이러한 상태가 발생할 수 있습니다.

diagsize - 회전 진단 및 관리 통지 로그 구성 매개변수

이 매개변수는 진단 로그 및 관리 통지 로그 파일의 최대 크기를 제어하는 데 유용합니다.

구성 유형

데이터베이스 관리 프로그램

매개변수 유형

온라인으로 구성 불가능

디폴트 0

회전 로그 크기로 지정할 수 있는 최소값:

2

회전 로그 크기로 지정할 수 있는 최대값:

diagpath에서 지정하는 디렉토리의 여유 공간 크기

수치 단위

MB

이 매개변수의 값을 디폴트인 0으로 설정하는 경우 db2diag.log 파일이라는 하나의 진단 로그 파일만 있습니다. 또한 <*instance*>.nfy 파일이라는 하나의 관리 통지 로그 파일만 있으며 이 로그 파일은 Linux 및 UNIX 운영 체제에서만 사용됩니다. 이들 파일 크기는 무제한으로 증가할 수 있습니다.

매개변수를 0이 아닌 값으로 설정하고 <인스턴스>를 재시작하는 경우 일련의 회전 진단 로그 파일 및 일련의 회전 관리 통지 로그 파일이 사용됩니다. 이러한 파일을 db2diag.*n*.log 및 <*instance*>.*n*.nfy 파일이라고 합니다. 여기서 *n*은 정수입니다. <*instance*>.*n*.nfy 파일은 Linux 및 UNIX 운영 체제에만 적용됩니다. db2diag.*n*.log 파일 및 <*instance*>.*n*.nfy 파일 수는 각각 10개를 초과할 수 없습니다. 10번 째 파일 크기가 가득 찬 경우에는 가장 오래된 파일이 삭제되고 새 파일이 작성됩니다.

예를 들어 Linux 및 UNIX 운영 체제에서는 **diagpath**에서의 회전 로그 파일이 다음과 유사합니다.

```
db2diag.14.log, db2diag.15.log, ... , db2diag.22.log, db2diag.23.log
<instance>.0.nfy, <instance>.1.nfy...,<instance>.8.nfy,<instance>.9.nfy
```

db2diag.23.log가 가득 차면 db2diag.14.log가 삭제되고, db2diag 로깅을 위해 db2diag.24.log가 작성됩니다.

<instance>.9.nfy가 가득 차면 <instance>.0.nfy가 삭제되고, 관리 통지 로깅을 위해 <instance>.10.nfy가 작성됩니다.

메시지는 항상 가장 큰 인덱스 번호 db2diag.*largest n*.log, <*instance*>.*largest n*.nfy를 갖는 회전 로그 파일에 로깅됩니다.

db2diag.*n*.log 및 <*instance*>.*n*.nfy 파일의 전체 크기는 **diagsize** 구성 매개변수의 값에 따라 결정됩니다. Windows 운영 체제의 경우를 제외하고, 기본적으로 **diagsize** 값의 90%는 db2diag.*n*.log 파일에, **diagsize** 값의 10%는 <*instance*>.*n*.nfy 파일에 할당됩니다. 예를 들어, Linux 또는 UNIX 운영 체제에서 **diagsize**를 1024로 설정하는 경우 db2diag.*n*.log 파일의 전체 크기는 921.6MB를 초과할 수 없으며 <*instance*>.*n*.nfy 파일의 전체 크기는 102.4MB를 초과할 수 없습니다. Windows 운영 체제에서는 **diagsize** 전체 값이 db2diag.*n*.log 파일에 할당됩니다. 각각의 로그 파일 크기는 각 유형의 로그 파일에 할당된 스페이스 전체 크기를 10으로 나눈 값으로 결정됩니다. 예를 들어, db2diag.*n*.log 파일의 전체 크기가 921.6MB를 초과할 수 없는 경우 각 db2diag.*n*.log 파일의 크기는 92.16MB넙니다.

diagsize 구성 매개변수에 지정하는 최대 값은 **diagpath** 구성 매개변수에 지정하는 디렉토리의 여유 공간 크기를 초과할 수 없습니다. 진단 및 관리 통지 로그 파일은 이 디렉토리에 저장됩니다. 파일 회전(가장 오래된 로그 파일 삭제)으로 인해 정보가 너무 빨리 유실되지 않게 하려면 **diagsize**를 50MB보다 큰 값으로 지정하되 **diagpath**에 지정하는 디렉토리 여유 공간의 80%를 초과하지 않게 하십시오.

예를 들면 다음 선언을

- **diagsize**를 DB2가 재시작될 때 회전 로깅 동작으로 전환되는 1024MB로 설정하려면, 다음 명령을 사용하십시오.

  ```
  db2 update dbm cfg using diagsize 1024
  ```

- DB2가 재시작될 때 디폴트 로깅 동작으로 전환하는 0으로 **diagsize**를 설정하려면 다음 명령을 사용하십시오.

  ```
  db2 update dbm cfg using diagsize 0
  ```

discover_db - 데이터베이스 발견

이 매개변수를 사용하면 서버에서 발견 요청을 받은 경우 데이터베이스에 대한 정보가 클라이언트로 리턴되지 않습니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
즉시

디폴트 [범위]
Enable [Disable, Enable]

이 매개변수의 디폴트는 이 데이터베이스에 대해 발견을 사용하는 것입니다.

이 매개변수 값을 『Disable』로 변경하면 발견 프로세스에서 중요한 데이터가 있는 데이터베이스를 숨길 수 있습니다. 데이터베이스에 대한 기타 데이터베이스 보안 제어 외에 이러한 기능을 수행할 수 있습니다.

dlchktime - 교착 상태를 점검하는 시간 간격

이 매개변수는 데이터베이스 관리 프로그램이 데이터베이스에 연결된 모든 응용프로그램들 중 교착 상태를 점검하는 빈도를 정의합니다.

구성 유형
데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

10 000 (10 seconds) [1 000 - 600 000]

수치 단위

밀리초

동일한 데이터베이스에 연결된 두 개 이상의 응용프로그램이 응답을 무한정 기다리는 경우 교착 상태가 발생합니다. 각 응용프로그램은 다른 응용프로그램이 계속하려면 필요한 자원을 보유하고 있으므로 대기 상태는 해결되지 않습니다.

주:

1. 파티션된 데이터베이스 환경에서 이 매개변수는 카탈로그 노드에만 적용됩니다.
2. 파티션된 데이터베이스 환경에서 두 번째 반복 이후까지 교착 상태에는 해결되지 않습니다.

권장사항: 이 매개변수를 늘리면 교착 상태 점검 빈도가 낮아지므로 해당 응용프로그램이 교착 상태가 해결될 때까지 기다려야 하는 시간이 증가합니다.

이 매개변수를 줄이면 교착 상태 점검 빈도가 높아지므로 응용프로그램이 교착 상태가 해결될 때까지 기다려야 하는 시간은 감소하지만 데이터베이스 관리 프로그램이 교착 상태를 점검하는 데 걸리는 시간은 증가합니다. 교착 상태 간격이 너무 작은 경우 데이터베이스 관리 프로그램이 교착 상태 발견을 자주 수행하므로 런타임 성능이 저하될 수 있습니다. 동시성을 높이기 위해 이 매개변수가 낮게 설정된 경우 추가 잠금 경합이 발생하여 교착 상태가 많이 발생하게 되는 불필요한 잠금 에스컬레이션을 방지하기 위해 *maxlocks* 및 *locklist*가 적절하게 설정되었는지 확인해야 합니다.

dyn_query_mgmt - 동적 SQL 및 XQuery 쿼리 관리

이 매개변수는 Query Patroller가 제출된 쿼리에 대한 정보를 캡처하는지 여부를 결정합니다.

중요사항: 이 매개변수는 Query Patroller 기능과 연관되어 있기 때문에 더 이상 사용되지 않습니다. DB2 버전 9.5에서 새 워크로드 관리 기능이 추가되어 Query Patroller 및 관련 구성요소는 버전 9.7에서 사용되지 않으며 추후 릴리스에서 제거됩니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

0(DISABLE) [1(ENABLE), 0(DISABLE)]

이 매개변수는 DB2 Query Patroller가 설치된 위치에 상대적입니다. 이 매개변수가 『ENABLE』로 설정된 경우, Query Patroller는 제출자 ID 및 계산된 실행 비용 등 옵티마이저가 계산한 쿼리에 대한 정보를 캡처합니다. 이 값은 사용자 또는 시스템 레벨 임계값에 따라 Query Patroller가 쿼리를 관리해야 하는지 여부를 판별하는 데 사용됩니다.

이 매개변수가 『DISABLE』로 설정되면 Query Patroller는 제출된 쿼리에 대한 정보를 캡처하지 않으며, 쿼리 관리도 발생하지 않습니다.

enable_xmlchar - XML로 변환 사용 구성 매개변수

이 매개변수는 SQL문에서 BIT DATA CHAR가 아닌(또는 CHAR 유형) 표현식에 대해 XMLPARSE 조작을 수행할 수 있는지 여부를 결정합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

Yes [Yes; No]

유니코드가 아닌 데이터베이스에서 pureXML® 기능이 사용되는 경우, XMLPARSE 기능은 SQL 문자열 데이터를 클라이언트 코드 페이지에서 데이터베이스 코드 페이지로 변환한 후 내부 스토리지의 유니코드로 변환하는 문자 대체를 유발할 수 있습니다. *enable_xmlchar*을 NO로 설정하면 XML 구문 분석 중에 문자 데이터 유형 사용이 차단되고 문자 유형을 유니코드가 아닌 데이터베이스에 삽입하려는 모든 시도는 오류를 생성합니다. BLOB 데이터 유형 및 FOR BIT DATA 데이터 유형은 이 유형을 사용하여 XML 데이터를 데이터베이스로 전달할 때 코드 페이지 변환이 발생하지 않으므로 *enable_xmlchar*이 NO로 설정된 경우에도 허용됩니다.

디폴트로 *enable_xmlchar*은 YES로 설정되므로 문자 데이터 유형의 구문 분석이 허용됩니다. 이 경우 XML 데이터를 삽입하는 동안 대체 문자가 나타나지 않도록 하려면 삽입할 XML 데이터에 데이터베이스 코드 페이지의 일부인 코드 포인트만 있는지 확인해야 합니다.

주: 이 변경사항을 적용하려면 클라이언트의 연결을 끊은 후 에이전트에 다시 연결해야 합니다.

failarchpath - 장애 복구 로그 아카이브 경로

이 매개변수는 해당 대상에 영향을 미치는 미디어 문제점이 발생하여 DB2가 기본 또는 보조(설정된 경우) 아카이브 대상에 로그 파일을 아카이브할 수 없는 경우 로그 파일을 아카이브할 경로를 지정합니다. 지정된 경로는 디스크를 참조해야 합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

Null [ ]

현재 *failarchpath*의 값으로 지정된 경로에 로그 파일이 있는 경우, *failarchpath*의 갱신사항은 즉시 적용되지 않습니다. 대신 모든 응용프로그램의 연결이 끊어질 때 갱신사항이 적용됩니다.

groupheap_ratio - 응용프로그램 그룹 힙의 메모리 퍼센트

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서는 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 DB2 버전 9.5 데이터베이스 관리 프로그램에서는 사용되지 않습니다. 버전 9.5에서 이 매개변수는 *appl_memory* 구성 매개변수로 교체되었습니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에만 적용됩니다.

이 매개변수는 응용프로그램 제어 공유 메모리 세트에서 응용프로그램 그룹 공유 힙에 충당된 메모리 퍼센트를 지정합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

70 [1 - 99]

수치 단위

퍼센트

이 매개변수는 집중기가 OFF 상태이고 파티션 내 병렬 처리가 사용되지 않는 파티션 되지 않은 데이터베이스에 영향을 미치지 않습니다.

권장사항: 성능 문제점이 없는 경우에는 이 매개변수를 디폴트값으로 유지하십시오.

hadr_db_role - HADR 데이터베이스 역할

이 매개변수는 데이터베이스가 온라인인지 오프라인인지에 관계없이 데이터베이스의 현재 역할을 표시합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버

매개변수 유형

정보용

유효한 값은 STANDARD, PRIMARY 또는 STANDBY입니다.

주: 데이터베이스가 활성인 경우 데이터베이스의 HADR 역할은 GET SNAPSHOT FOR DATABASE 명령을 사용하여 판별할 수도 있습니다.

hadr_local_host - HADR 로컬 호스트 이름

이 매개변수는 고가용성 재해 복구(HADR) TCP 통신용 로컬 호스트를 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 Null

호스트 이름 또는 IP 주소를 사용할 수 있습니다. 호스트 이름이 지정되고 다중 IP 주소에 맵핑되는 경우에는 오류가 리턴되며 HADR이 시작되지 않습니다. 호스트 이름이 다중 IP 주소에 맵핑되는 경우(기본 및 대기에 동일한 호스트 이름을 지정한 경우에도) 기본 및 대기는 이 호스트 이름을 다른 IP 주소에 맵핑하는 결과가 됩니다. 일부 DNS 서버는 IP 주소를 비결정적 순서로 나열하기 때문입니다.

호스트 이름은 myserver.ibm.com 형식입니다. IP 주소는 ″12.34.56.78″ 형식입니다.

hadr_local_svc - HADR 로컬 서비스 이름

이 매개변수는 로컬 고가용성 재해 복구(HADR) 프로세스가 연결을 승인하는 TCP 서비스 이름 또는 포트 번호를 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 Null

기본 또는 대기 데이터베이스 시스템에서 hadr_local_svc의 값은 각 노드의 svcename 또는 svcename +1 값과 동일할 수 없습니다.

SSL을 사용하는 경우에는 기본 또는 대기 데이터베이스 시스템에서 hadr_local_svc를 ssl_svcename과 동일하게 설정하지 마십시오.

hadr_peer_window - HADR 피어 창 구성 매개변수

hadr_peer_window를 0이 아닌 시간 값으로 설정하면 HADR 기본-대기 데이터베이스 쌍은 기본 데이터베이스에서 대기 데이터베이스에 대한 연결이 끊어져도 구성된 시간 동안 여전히 피어 상태인 것처럼 동작합니다. 이것은 데이터 일관성을 확인하는 데 도움이 됩니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

0 [0 – 4 294 967 295]

수치 단위

초

사용법 참고:

- 기본 데이터베이스와 대기 데이터베이스에서 값이 같아야 합니다.
- 권장 최소값은 120초입니다.
- **hadr_syncmode** 값이 ASYNC로 설정되면 **hadr_peer_window** 값은 무시됩니다. 즉 ASYNC 모드에서는 의미가 없으므로 값이 0으로 설정된 것처럼 처리됩니다.

- 대기 데이터베이스를 고의로 종료하는 경우(예: 유지보수를 위해) 기본 데이터베이스의 사용 가능성에 영향을 미치지 않기 위해 HADR 쌍이 피어 상태일 때 대기 데이터베이스가 명시적으로 비활성화되면 피어 창이 호출되지 않습니다.
- 기본 데이터베이스 클럭과 대기 데이터베이스 클럭이 5초 이내에 동기화되지 않는 경우에는 **hadr_peer_window** 매개변수를 사용한 인계 조작이 제대로 작동하지 않을 수 있습니다. 즉 실패해야 하는 경우에 성공하거나, 성공해야 하는 경우에 실패할 수 있습니다. 클럭이 동일한 소스로 동기화된 상태를 유지하려면 시간 동기화 서비스(예: NTP)를 사용해야 합니다.
- 대기 데이터베이스에서 피어 창 종료 시간은 연결 끊어짐이 아니라 기본 데이터베이스에서 마지막으로 수신한 하트비트 메시지를 기본으로 합니다. 따라서 S-RemoteCatchupPending으로 전이되기 전에 S-DisconnectedPeer 상태에 있는 대기 데이터베이스의 남아 있는 시간 범위는 연결이 끊어진 시기 및 방법에 따라 (**hadr_peer_window** - **hadr_timeout**)초에서 (**hadr_peer_window**)초까지입니다.

hadr_remote_host - HADR 리모트 호스트 이름

이 매개변수는 리모트 고가용성 재해 복구(HADR) 데이터베이스 서버의 TCP/IP 호스트 이름 또는 IP 주소를 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 Null

*hadr_local_host*와 마찬가지로, 이 매개변수도 단일 IP 주소에만 맵핑해야 합니다.

hadr_remote_inst - 리모트 서버의 HADR 인스턴스 이름

이 매개변수는 리모트 서버의 인스턴스 이름을 지정합니다. DB2 제어 센터와 같은 관리 도구는 이 매개변수를 사용하여 리모트 서버에 접속합니다. 고가용성 재해 복구(HADR)는 리모트 데이터베이스가 선언된 리모트 인스턴스에 속하는 연결을 요청하는지 여부도 점검합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 Null

hadr_remote_svc - HADR 리모트 서비스 이름

이 매개변수는 리모트 고가용성 재해 복구(HADR) 데이터베이스 서버가 사용할 TCP 서비스 이름 또는 포트 번호를 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 Null

hadr_syncmode - 피어 상태에서 로그 쓰기용 HADR 동기화 모드

이 매개변수는 시스템이 피어 상태일 때 1차 로그 쓰기가 대기와 동기화되는 방법을 결정하는 동기화 모드를 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

NEARSYNC [ASYNC; SYNC]

이 매개변수에 유효한 값은 다음과 같습니다.

SYNC 이 모드는 트랜잭션 손실을 최대로 보호하지만 트랜잭션 응답 시간이 가장 오래 걸립니다.

이 모드에서 로그 쓰기는 로그가 기본 데이터베이스의 로그 파일에 기록되고 기본 데이터베이스가 대기 데이터베이스로부터 대기 데이터베이스의 로그 파일에

도 로그를 기록했다는 수신확인을 받은 경우에만 성공으로 간주됩니다. 로그 데이터가 두 사이트 모두에 저장되었음이 보증됩니다.

NEARSYNC

이 모드는 SYNC 모드에 비해 트랜잭션 손실을 약간 덜 보호하지만 트랜잭션 응답 시간이 짧아집니다.

이 모드에서 로그 쓰기는 로그 레코드가 기본 데이터베이스의 로그 파일에 기록되고 기본 데이터베이스가 대기 시스템으로부터 대기 시스템의 주기억장치에 로그가 기록되었다는 수신확인을 받은 경우에만 성공으로 간주됩니다. 두 사이트가 동시에 실패하고 목표 사이트가 수신한 로그 데이터를 비휘발성 스토리지로 모두 전송하지 않은 경우에만 데이터 손실이 발생합니다.

ASYNC

이 모드는 기본 실패 시 트랜잭션 손실 가능성이 가장 높지만 세 가지 모드 중 트랜잭션 응답 시간이 가장 짧습니다.

이 모드에서 로그 쓰기는 로그 레코드가 기본 데이터베이스의 로그 파일에 기록되고 기본 시스템 호스트 머신의 TCP 계층으로 전달된 경우에만 성공으로 간주됩니다. 기본 시스템이 대기 시스템의 수신확인을 기다리지 않으므로 트랜잭션이 대기로 전달되는 중에도 커미트로 간주될 수 있습니다.

hadr_timeout - HADR 시간종료 값

이 매개변수는 통신 시도가 실패한 것으로 간주하기 전에 고가용성 재해 복구(HADR) 프로세스가 대기할 시간을 초 단위로 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

120 [1 - 4 294 967 295]

indexrec - 인덱스 재작성 시간

이 매개변수는 데이터베이스 관리 프로그램이 유효하지 않은 인덱스를 재빌드하려고 시도하는 시기와 대기 데이터베이스에서 DB2 롤 포워드 또는 HADR 로그 재생 중 인덱스 빌드가 재실행되는지 여부를 표시합니다.

구성 유형

데이터베이스 및 데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

UNIX 데이터베이스 관리 프로그램

restart [restart; restart_no_redo; access; access_no_redo]

Windows 데이터베이스 관리 프로그램

restart [restart; restart_no_redo; access; access_no_redo]

데이터베이스

시스템 설정값 사용 [system; restart; restart_no_redo; access; access_no_redo]

이 매개변수에는 다섯 가지 설정이 가능합니다.

SYSTEM

데이터베이스 관리 프로그램 구성 파일에 지정된 *시스템 설정값 사용*은 유효하지 않은 인덱스를 재빌드하는 시기와 DB2 롤 포워드 또는 HADR 로그 재생 중 인덱스 빌드 로그 레코드가 재실행되는지 여부를 결정합니다(주: 이 설정은 데이터베이스 구성의 경우에만 유효함).

ACCESS

인덱스에 처음 액세스할 때 유효하지 않은 인덱스가 재빌드됩니다. 완전히 로깅된 인덱스 빌드가 DB2 롤 포워드 또는 HADR 로그 재생 중 재실행됩니다. HADR이 시작되고 HADR 인계가 발생하면 기본 테이블에 처음 액세스할 때 인계 후 유효하지 않은 인덱스가 재빌드됩니다.

ACCESS_NO_REDO

기본 테이블에 처음 액세스할 때 유효하지 않은 인덱스가 재빌드됩니다. 완전히 로깅된 인덱스 빌드가 DB2 롤 포워드 또는 HADR 로그 재생 중 재실행되며 이러한 인덱스는 계속 유효하지 않습니다. HADR이 시작되고 HADR 인계가 발생하면 기본 테이블에 처음 액세스할 때 인계 후 유효하지 않은 인덱스가 재빌드됩니다.

RESTART

*indexrec*의 디폴트값입니다. RESTART DATABASE 명령이 명시적 또는 내재적으로 발행된 경우 유효하지 않은 인덱스가 재빌드됩니다. 완전히 로깅된 인

덱스 빌드가 DB2 롤 포워드 또는 HADR 로그 재생 중 재실행됩니다. HADR이 시작되고 HADR 인계가 발생하면 인계 종료 시 유효하지 않은 인덱스가 재빌드됩니다.

RESTART DATABASE 명령은 *autorestart* 매개변수를 사용하는 경우 내재적으로 발행됩니다.

RESTART_NO_REDO

RESTART DATABASE 명령이 명시적 또는 내재적으로 발행된 경우 유효하지 않은 인덱스가 재빌드됩니다(*autorestart* 매개변수를 사용하는 경우 RESTART DATABASE 명령이 내재적으로 발행됨). 완전히 로깅된 인덱스 빌드는 DB2 롤 포워드 또는 HADR 로그 재생 중 재실행되지 않으며 대신 이러한 인덱스는 롤 포워드가 완료되거나 HADR 인계가 발생할 때 재빌드됩니다.

치명적 디스크 문제가 발생하면 인덱스가 유효하지 않게 될 수 있습니다. 데이터 자체에 이러한 상태가 발생하면 데이터가 유실될 수 있습니다. 그러나 인덱스에 이러한 상태가 발생하면 인덱스를 재작성하여 복구할 수 있습니다. 사용자가 데이터베이스에 연결되어 있는 동안 인덱스를 재빌드하는 경우 두 가지 문제가 발생할 수 있습니다.

- 인덱스 파일이 재작성될 때 응답 시간이 예기치 않게 느려질 수 있습니다. 테이블에 액세스하고 이 특정 인덱스를 사용하는 사용자는 인덱스가 재빌드되는 동안 대기합니다.
- 인덱스 재작성 후 특히 인덱스 재작성의 원인이 되는 사용자 트랜잭션이 COMMIT 또는 ROLLBACK을 수행하지 않은 경우 예기치 않은 잠금이 보유될 수 있습니다.

권장사항: 사용자가 많은 서버와 재시작 시간이 문제가 되지 않는 경우 이 옵션에 대해 가장 적합한 선택은 손상 이후 데이터베이스를 다시 온라인 상태로 전환하는 프로세스의 일부분으로 DATABASE RESTART 시 인덱스를 재빌드하는 것입니다.

이 매개변수를 『ACCESS』 또는 『ACCESS_NO_REDO』로 설정하면 인덱스가 재작성되는 동안 데이터베이스 관리 프로그램의 성능이 저하됩니다. 해당 특정 인덱스 또는 테이블에 액세스하는 사용자는 인덱스가 재작성될 때까지 기다려야 합니다.

이 매개변수가 『RESTART』로 설정된 경우 데이터베이스를 재시작하는 데 걸리는 시간은 인덱스 재작성으로 인해 길어지지만 데이터베이스가 다시 온라인 상태로 전환된 후 정상 처리는 영향을 받지 않습니다.

주: 데이터베이스 복구 시 복구 중인 데이터베이스에 속한 파일 시스템의 모든 SQL 프로시저 실행 파일이 제거됩니다. *indexrec*가 RESTART로 설정된 경우 모든 SQL 프로시저 실행 파일을 데이터베이스 카탈로그에서 추출하여 다음에 데이터베이스에 연결할 때 파일 시스템에 다시 저장합니다. *indexrec*가 RESTART로 설정되지 않은 경우 SQL 프로시저를 처음 실행할 때에만 파일 시스템으로 SQL 실행 파일을 추출합니다.

RESTART 값과 RESTART_NO_REDO 값의 차이 또는 ACCESS 값과 ACCESS_NO_REDO 값의 차이는 CREATE INDEX 및 REORG INDEX 조작과 같은 인덱스 빌드 조작이나 인덱스 재빌드를 위해 전체 로깅이 활성화된 경우에만 중요합니다. *logindexbuild* 데이터베이스 구성 매개변수를 사용하거나 테이블 변경 시 LOG INDEX BUILD를 사용하여 로깅을 활성화할 수 있습니다. *indexrec*를 RESTART 또는 ACCESS로 설정하면 로깅된 인덱스 빌드와 관련된 조작은 인덱스 오브젝트를 유효하지 않은 상태로 두지 않고 롤 포워드할 수 있으므로 나중에 인덱스를 재빌드해야 합니다.

jdk_64_path - Java용 64비트 SDK(Software Developer's Kit) 설치 경로 DAS

이 매개변수는 DB2 Administration Server 기능을 실행하는 데 사용할 Java용 64비트 SDK(Software Developer's Kit)가 설치된 디렉토리를 지정합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

Null [유효한 경로]

주: 이 매개변수는 Java용 32비트 SDK를 지정하는 **jdk_path** 구성 매개변수와 다릅니다.

Java 인터프리터에서 사용되는 환경 변수는 이 매개변수의 값에서 계산됩니다. 이 매개변수는 32비트와 64비트 인스턴스를 둘 다 지원하는 플랫폼에서만 사용됩니다.

해당 플랫폼은 다음과 같습니다.

- AIX, HP-UX 및 Solaris 운영 체제의 64비트 커널
- X64 및 IPF의 64비트 Windows
- AMD64 및 Intel® EM64T 시스템(x64), POWER 및 zSeries의 64비트 Linux 커널

다른 모든 플랫폼에서는 **jdk_path**만 사용됩니다.

이 매개변수에는 디폴트값이 없으므로 Java용 SDK를 설치할 때 값을 지정해야 합니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

locklist - 잠금 목록의 최대 스토리지

이 매개변수는 잠금 목록에 할당된 스토리지의 양을 표시합니다. 데이터베이스 당 하나의 잠금 목록이 있으며 데이터베이스에 동시에 연결된 모든 응용프로그램이 보유한 잠금이 들어 있습니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

UNIX Automatic [4 - 524 288]

로컬 및 리모트 클라이언트가 있는 Windows 데이터베이스 서버

Automatic [4 - 524 288]

로컬 클라이언트가 있는 Windows 64비트 데이터베이스 서버

Automatic [4 - 524 288]

로컬 클라이언트가 있는 Windows 32비트 데이터베이스 서버

Automatic [4 - 524 288]

수치 단위

페이지(4KB)

할당 시기

응용프로그램이 데이터베이스에 처음 연결할 때

사용 가능한 시기

데이터베이스에서 마지막 응용프로그램의 연결이 끊어졌을 때

잠금은 여러 응용프로그램이 데이터베이스의 데이터에 동시에 액세스하는 것을 데이터베이스 관리 프로그램이 제어하는 데 사용하는 메커니즘입니다. 행 및 테이블을 모두 잠글 수 있습니다. 또한 데이터베이스 관리 프로그램은 내부용으로 잠금을 획득할 수 있습니다.

이 매개변수가 AUTOMATIC으로 설정된 경우 자체 성능 조정할 수 있습니다. 이 경우 워크로드 요구사항이 변경되면 메모리 조정 프로그램이 이 매개변수가 제어하는 메모리 영역의 크기를 동적으로 조정합니다. 메모리 조정 프로그램은 여러 메모리 사용자 간에 메모리 자원을 교환하므로 자체 성능 조정이 활성화되려면 자체 성능 조정에 대해 최소 두 개의 메모리 사용자가 사용 가능해야 합니다.

*locklist*의 값이 *maxlocks* 매개변수와 함께 조정되므로 *locklist* 매개변수의 자체 성능 조정을 사용 불가능하게 하면 *maxlocks* 매개변수의 자체 성능 조정이 자동으로 사용 불가능해집니다. *locklist* 매개변수의 자체 성능 조정을 사용 가능하게 하면 *maxlocks* 매개변수의 자체 성능 조정이 자동으로 사용 가능해집니다.

이 구성 매개변수의 자동 성능 조정은 데이터베이스에 자체 성능 조정 메모리를 사용할 수 있는 경우에만 발생합니다(*self_tuning_mem* 구성 매개변수가 ″ON″으로 설정됨).

32비트 플랫폼에서는 오브젝트에 기타 잠금이 보유되어 있는지 여부에 따라 각 잠금은 48 또는 96바이트의 잠금 목록이 필요합니다.

- 96바이트는 기타 잠금이 보유되어 있지 않은 오브젝트에서 잠금을 보유할 때 필요합니다.
- 48바이트는 기존 잠금이 보유되어 있는 오브젝트에서 잠금을 기록할 때 필요합니다.

64비트 플랫폼(HP-UX/PA-RISC 제외)에서는 오브젝트에 기타 잠금이 보유되어 있는지 여부에 따라 각 잠금은 64 또는 128바이트의 잠금 목록이 필요합니다.

- 128바이트는 기타 잠금이 보유되어 있지 않은 오브젝트에서 잠금을 보유할 때 필요합니다.
- 64바이트는 기존 잠금이 보유되어 있는 오브젝트에서 잠금을 기록할 때 필요합니다.

64비트 HP-UX/PA-RISC에서는 오브젝트에 기타 잠금이 보유되어 있는지 여부에 따라 각 잠금은 80 또는 160바이트의 잠금 목록이 필요합니다.

한 응용프로그램이 사용하는 잠금 목록의 퍼센트가 *maxlocks*에 접근하면 데이터베이스 관리 프로그램은 응용프로그램이 보유한 잠금에 대해 행에서 테이블로 잠금 에스컬레이션을 수행합니다. 에스컬레이션 프로세스 자체는 시간이 많이 걸리지 않지만 전체 테이블(개별 행과 대조)을 잠그면 동시성이 줄어들고 영향을 받은 테이블에 대한 후속 액세스에서 전체 데이터베이스 성능이 저하될 수 있습니다. 잠금 목록의 *크기*를 제어하는 방법으로 다음과 같이 제안할 수 있습니다.

- COMMIT를 자주 수행하여 잠금을 해제하십시오.
- 여러 갱신을 수행할 때 갱신하기 전에 전체 테이블을 잠그십시오(SQL LOCK TABLE 문 사용). 이 경우 하나의 잠금만 사용하므로 다른 잠금이 갱신을 방해하지 못하는 장점이 있지만 데이터의 동시성은 줄어듭니다.

  또한 ALTER TABLE문의 LOCKSIZE 옵션을 사용하면 특정 테이블의 잠금을 수행하는 방식을 제어할 수 있습니다.

  RR(Repeatable Read) 분리 레벨을 사용하면 자동 테이블 잠금이 발생할 수 있습니다.
- 커서 안정성 분리 레벨이 사용 가능한 경우 이 분리 레벨을 사용하여 보유한 공유 잠금 수를 줄이십시오. 응용프로그램 무결성 요구사항이 절충되지 않은 경우 커서 안정성 대신 언커미트된 읽기를 사용하여 잠금의 양을 추가로 줄이십시오.

- *locklist*를 AUTOMATIC으로 설정하십시오. 잠금 목록이 동기식으로 증가하여 잠금 에스컬레이션 또는 잠금 목록이 가득 차는 상태를 방지합니다.

잠금 목록이 가득 차면 잠금 에스컬레이션이 테이블 잠금을 추가로 생성하고 행 잠금은 적게 생성하여 성능이 저하될 수 있으므로 데이터베이스에서 공유 오브젝트의 동시성이 줄어듭니다. 또한 응용프로그램들 간 추가 교착 상태가 발생하여(제한된 수의 테이블 잠금에서 모두 대기 중이므로) 트랜잭션이 롤백됩니다. 데이터베이스의 최대 잠금 요청 수에 접근하면 응용프로그램은 SQLCODE -912를 받습니다.

권장사항: 잠금 에스컬레이션으로 인해 성능 문제가 발생하는 경우 이 매개변수 또는 *maxlocks* 매개변수의 값을 늘려야 합니다. 데이터베이스 시스템 모니터를 사용하여 잠금 에스컬레이션이 발생하는지 여부를 판별할 수 있습니다. *lock_escals(잠금 에스컬레이션)* 모니터 요소를 참조하십시오.

다음의 단계는 잠금 목록에 필요한 페이지 수를 판별하는 데 유용합니다.

1. 사용자의 환경에 따라 다음 계산 중 *하나*를 사용하여 잠금 목록 *크기*의 하한을 계산하십시오.

   a.
   ```
   (512 * x * maxappls) / 4096
   ```
   b. 집중기가 사용 가능한 경우:
   ```
   (512 * x * max_coordagents) / 4096
   ```
   c. 집중기가 사용 가능한 파티션된 데이터베이스에서:
   ```
   (512 * x * max_coordagents * 데이터베이스 파티션 수) / 4096
   ```

   여기서 512는 응용프로그램당 평균 잠금 수 추정이고 x는 기존 잠금이 있는 오브젝트에 대해 각 잠금에 필요한 바이트 수입니다(32비트 플랫폼의 경우 40바이트, 64비트 플랫폼의 경우 64바이트).
2. 잠금 목록 *크기*의 상한을 계산하십시오.
   ```
   (512 * y * maxappls) / 4096
   ```

   여기서 y는 오브젝트에 대해 첫 번째 잠금에 필요한 바이트 수입니다(32비트 플랫폼의 경우 80바이트, 64비트 플랫폼의 경우 128바이트).
3. 데이터에 대한 동시처리의 양을 추정하고 이러한 예상에 따라 계산한 상한과 하한 사이에서 *locklist*의 *초기*값을 선택하십시오.
4. 아래에 설명된 데이터베이스 시스템 모니터를 사용하여 이 매개변수의 값을 조정하십시오.

적용 가능한 시나리오에서 *maxappls* 또는 *max_coordagents*가 AUTOMATIC으로 설정된 경우 *locklist*도 AUTOMATIC으로 설정해야 합니다.

데이터베이스 시스템 모니터를 사용하여 지정된 트랜잭션이 보유한 최대 잠금 수를 판별할 수 있습니다. *locks_held_top(최대 보유된 잠금 수)* 모니터 요소를 참조하십시오.

이 정보는 응용프로그램당 추정 잠금 수의 유효성을 확인하거나 조정하는 데 유용합니다. 이 유효성 확인을 수행하려면 여러 응용프로그램을 샘플링해야 하며 모니터 정보는 응용프로그램 레벨이 아닌 트랜잭션 레벨에서 제공됩니다.

또한 *maxappls*가 증가했거나 실행 중인 응용프로그램이 커미트를 자주 수행하지 않는 경우 *locklist*를 늘릴 수 있습니다.

이 매개변수를 변경한 후 응용프로그램 리바인드를 고려해야 합니다(REBIND 명령 사용).

locktimeout - 잠금 시간종료

이 매개변수는 응용프로그램의 전역 교착 상태를 방지하기 위해 응용프로그램이 잠금을 확보될 때까지 대기하는 시간(초)을 지정합니다.

구성 유형
데이터베이스

매개변수 유형
구성 가능

디폴트 [범위]
-1 [-1; 0 - 32 767]

수치 단위
초

이 매개변수를 0으로 설정한 경우 잠금이 발생할 때까지 대기하지 않습니다. 이 경우 요청 시 잠금이 사용 가능하지 않으면 응용프로그램이 즉시 -911을 수신합니다.

이 매개변수를 -1로 설정한 경우 잠금 시간종료 감지가 해제됩니다. 이 경우 다음 중 하나가 발생할 때까지 잠금 대기합니다(요청 시 잠금이 사용 가능하지 않은 경우).

- 잠금에 권한 부여될 때
- 교착 상태 발생 시

권장사항: 트랜잭션 처리(OLTP) 환경에서 초기 시작 값 30초를 사용할 수 있습니다. 쿼리 전용 환경에서는 높은 값부터 시작할 수 있습니다. 두 경우에 모두 벤치마킹 기술을 사용하여 이 매개변수를 조정해야 합니다.

정지된 트랜잭션(사용자가 워크스테이션을 사용하지 않아서 발생할 수 있음)과 같은 비정상적인 상황으로 인해 발생하는 대기를 신속하게 발견하도록 값을 설정해야 합니다. 잠금 대기 중인 동안 최대 워크로드로 인해 유효한 잠금 요청이 시간종료되지 않도록 이 값을 충분히 높게 설정해야 합니다.

데이터베이스 시스템 모니터를 사용하면 응용프로그램(연결)에서 잠금 시간종료가 발생한 횟수나 데이터베이스가 연결된 모든 응용프로그램의 시간종료 상황을 발견한 횟수를 추적할 수 있습니다.

다음과 같은 경우 *lock_timeout*(잠금 시간종료 수) 모니터 요소의 값이 높아질 수 있습니다.

- 이 구성 매개변수의 값이 너무 낮은 경우
- 오랜 기간 동안 잠금을 응용프로그램(트랜잭션)이 잠금을 보유하는 경우. 데이터베이스 시스템 모니터를 사용하여 이러한 응용프로그램을 추가로 조사할 수 있습니다.
- 잠금 에스컬레이션(행 레벨에서 테이블 레벨 잠금으로)으로 인해 발생할 수 있는 동시처리 문제.

log_retain_status - 로그 유지 상태 표시기

매개변수 *logretain* 설정이 Recovery인 경우 이 매개변수는 로그 파일을 롤 포워드 복구에서 사용하도록 유지함을 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

logarchmeth1 - 1차 로그 아카이브 방법

이 매개변수는 아카이브 로그 기본 대상의 미디어 유형을 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

OFF [LOGRETAIN, USEREXIT, DISK, TSM, VENDOR]

OFF 로그 아카이브 방법이 사용되지 않음을 지정합니다. **logarchmeth1**과 **logarchmeth2**가 둘 다 OFF로 설정되는 경우, 데이터베이스는 순환 로그를 사용하는 것으로 간주되며 롤 포워드 복구 가능하지 않습니다. 디폴트입니다.

LOGRETAIN

이 값은 **logarchmeth1**에만 사용할 수 있으며, **logretain** 구성 매개변수를 RECOVERY로 설정하는 것과 동일합니다. 이 값을 지정하면 **logretain** 구성 매개변수는 자동으로 갱신됩니다.

USEREXIT

이 값은 **logarchmeth1**에만 사용할 수 있으며, **userexit** 구성 매개변수를 ON으로 설정하는 것과 동일합니다. 이 값을 지정하면 **userexit** 구성 매개변수는 자동으로 갱신됩니다.

DISK 이 값 다음에는 콜론(:)과 로그 파일을 아카이브할 완전한 기존 경로 이름이 있어야 합니다. 예를 들어, **logarchmeth1**을 DISK:/u/dbuser/archived_logs로 설정한 경우 아카이브 로그 파일은 `/u/dbuser/archived_logs`라는 디렉토리에 놓이게 됩니다.

주: 테이프에 아카이브하는 경우에는 db2tapemgr 유틸리티를 사용하여 로그 파일을 저장하고 검색할 수 있습니다.

TSM 추가적인 구성 매개변수를 사용하지 않고 이 값을 지정하면 로그 파일이 디폴트 관리 클래스를 사용하여 로컬 TSM 서버에 아카이브되어야 함을 표시합니다. 다음에 콜론(:)과 TSM 관리 클래스가 있는 경우, 로그 파일은 지정된 관리 클래스를 사용하여 아카이브됩니다.

TSM을 사용하여 로그를 아카이브하는 경우, TSM은 데이터베이스 구성 매개변수가 지정한 관리 클래스를 사용하기 전에 우선 TSM 클라이언트 옵션 파일에서 찾은 INCLUDE-EXCLUDE 목록에서 지정된 관리 클래스에 오브젝트를 바인드하려고 시도합니다. 일치하는 클래스를 찾을 수 없는 경우 TSM 서버에서 지정된 디폴트 TSM 관리 클래스가 사용됩니다. 그런 다음 TSM은 데이터베이스 구성 매개변수가 지정한 관리 클래스에 오브젝트를 리바인드합니다.

그러므로 기본 관리 클래스와 데이터베이스 구성 매개변수가 지정한 관리 클래스에는 아카이브 사본 그룹이 있어야 하며, 그렇지 않으면 아카이브 조작이 실패합니다.

VENDOR

로그 파일을 아카이브하는 데 사용할 벤더 라이브러리를 지정합니다. 이 값 다음에는 콜론(:)과 라이브러리 이름이 있어야 합니다. 라이브러리에서 제공된 API는 벤더 제품에 대해 백업 및 리스토어 API를 사용해야 합니다.

메모:

1. **logarchmeth1** 또는 **logarchmeth2**가 OFF 이외의 값으로 설정된 경우, 데이터베이스는 롤 포워드 복구용으로 구성됩니다.

2. **userexit** 또는 **logretain** 구성 매개변수를 갱신하는 경우 **logarchmeth1**도 자동으로 갱신되며, 그 반대도 마찬가지입니다. 그러나 **userexit** 또는 **logretain**을 사용 중인 경우 **logarchmeth2**를 OFF로 설정해야 합니다.

logarchmeth2 - 2차 로그 아카이브 방법

이 매개변수는 아카이브 로그 보조 대상의 미디어 유형을 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

OFF [LOGRETAIN, USEREXIT, DISK, TSM, VENDOR]

OFF 로그 아카이브 방법이 사용되지 않음을 지정합니다. *logarchmeth1*과 *logarchmeth2*가 둘 다 OFF로 설정된 경우, 데이터베이스는 순환 로그를 사용하는 것으로 간주되며 복구 가능한 롤 포워드가 아닙니다. 디폴트입니다.

LOGRETAIN

이 값은 *logarchmeth1*에만 사용되며 *logretain* 구성 매개변수를 RECOVERY로 설정하는 것과 같습니다. 이 값을 지정하면 *logretain* 구성 매개변수는 자동으로 갱신됩니다.

USEREXIT

이 값은 *logarchmeth1*에만 유효하며 *userexit* 구성 매개변수를 ON으로 설정하는 것과 같습니다. 이 값을 지정하면 *userexit* 구성 매개변수는 자동으로 갱신됩니다.

DISK 이 값 다음에는 콜론(:)과 로그 파일을 아카이브할 완전한 기존 경로 이름이 있어야 합니다. 예를 들어, *logarchmeth1*을 `DISK:/u/dbuser/archived_logs`로 설정한 경우 아카이브 로그 파일은 `/u/dbuser/archived_logs`라는 디렉토리에 놓이게 됩니다.

주: 테이프에 아카이브하는 경우에는 db2tapemgr 유틸리티를 사용하여 로그 파일을 저장하고 검색할 수 있습니다.

TSM 추가적인 구성 매개변수를 사용하지 않고 이 값을 지정하면 로그 파일이 디폴트 관리 클래스를 사용하여 로컬 TSM 서버에 아카이브되어야 함을 표시합니다. 다음에 콜론(:)과 TSM 관리 클래스가 있는 경우, 로그 파일은 지정된 관리 클래스를 사용하여 아카이브됩니다.

VENDOR
로그 파일을 아카이브하는 데 사용할 벤더 라이브러리를 지정합니다. 이 값 다음에는 콜론(:)과 라이브러리 이름이 있어야 합니다. 라이브러리에서 제공된 API는 벤더 제품에 대해 백업 및 리스토어 API를 사용해야 합니다.

메모:

1. *logarchmeth1* 또는 *logarchmeth2*가 OFF가 아닌 값으로 설정된 경우, 데이터베이스는 롤 포워드 복구에 대해 구성됩니다.
2. *userexit* 또는 *logretain* 구성 매개변수를 갱신하는 경우 *logarchmeth1*도 자동으로 갱신되며, 그 반대도 마찬가지입니다. 그러나 *userexit* 또는 *logretain*을 사용 중인 경우 *logarchmeth2*는 OFF로 설정되어야 합니다.

이 경로가 지정된 경우 로그 파일은 이 대상과 *logarchmeth1* 데이터베이스 구성 매개변수가 지정하는 대상 둘 다에 아카이브됩니다.

logarchopt1 - 1차 로그 아카이브 옵션

이 매개변수는 아카이브 로그 기본 대상의 옵션 필드를 지정합니다(필요한 경우).

구성 유형
데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형
온라인으로 구성 가능

디폴트 [범위]
Null [적용 불가능]

logarchopt2 - 2차 로그 아카이브 옵션

이 매개변수는 아카이브 로그 보조 대상의 옵션 필드를 지정합니다(필요한 경우).

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

Null [적용 불가능]

logbufsz - 로그 버퍼 크기

이 매개변수를 사용하면 디스크에 이러한 레코드를 기록하기 전에 로그 레코드의 버퍼로 사용할 데이터베이스 힙의 양(*dbheap* 매개변수에 정의됨)을 지정할 수 있습니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

32비트 플랫폼

8 [4 - 4 096]

64비트 플랫폼

8 [4 - 131 070]

수치 단위

페이지(4KB)

다음 중 하나가 발생한 경우 디스크에 로그 레코드가 기록됩니다.

- *mincommit* 구성 매개변수에 정의된 바와 같이 트랜잭션이 커미트되었거나 트랜잭션 그룹이 커미트됨
- 로그 버퍼가 가득 참
- 일부 다른 내부 데이터베이스 관리 프로그램 이벤트의 결과로

또한 이 매개변수는 *dbheap* 매개변수 이하여야 합니다. 로그 레코드가 디스크에 자주 기록되지 않고 매번 추가 로그 레코드가 기록되므로 로그 레코드를 버퍼링하면 파일 입출력을 보다 효율적으로 로깅합니다.

권장사항: 전용 로그 디스크에 상당한 읽기 활동이 있거나 디스크 활용도가 높은 경우 이 버퍼 영역의 *크기*를 늘리십시오. 이 매개변수의 값을 늘릴 때 로그 버퍼 영역은 *dbheap* 매개변수로 제어되는 스페이스를 사용하므로 *dbheap* 매개변수도 고려해야 합니다.

데이터베이스 시스템 모니터를 사용하면 특정 트랜잭션(또는 작업 단위)에 사용되는 로그 버퍼 스페이스의 양을 판별할 수 있습니다. *log_space_used*(사용한 작업 단위 로그 스페이스) 모니터 요소를 참조하십시오.

logfilsiz - 로그 파일 크기

이 매개변수는 각 기본 및 2차 로그 파일의 *크기*를 정의합니다. 이러한 로그 파일의 크기는 가득 차서 새 로그 파일이 필요하기 전에 여기에 기록될 수 있는 로그 레코드 수를 제한합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

UNIX 1000 [4 - 1 048 572]

Windows

1000 [4 - 1 048 572]

수치 단위

페이지(4KB)

기본 및 2차 로그 파일의 사용과 로그 파일이 가득 찼을 때 취하는 조치는 수행 중인 로깅의 유형에 따라 다릅니다.

- 순환 로깅

  기록된 변경사항이 커미트된 경우 1차 로그 파일을 재사용할 수 있습니다. 로그 파일 *크기*가 작으며 응용프로그램이 변경사항을 커미트하지 않고 데이터베이스에 대량의 변경사항을 처리한 경우 1차 로그 파일이 신속하게 가득 찰 수 있습니다. 모든 1차 로그 파일이 가득 차면 데이터베이스 관리 프로그램은 새 로그 레코드를 보유할 2차 로그 파일을 할당합니다.

- 로그 보존 로깅

  1차 로그 파일이 가득 차면 로그를 아카이브하고 새 1차 로그 파일이 할당됩니다.

권장사항: 1차 로그 파일 수에 맞게 로그 파일 *크기*를 조정해야 합니다.

- 데이터베이스에서 갱신, 삭제 또는 삽입 트랜잭션을 많이 실행하여 로그 파일이 매우 신속하게 가득 차는 경우 *logfilsiz* 값을 늘려야 합니다.

  주: 로그 파일 크기의 상한을 로그 파일 수의 상한과 결합하면(*logprimary* + *logsecond*) 1024GB의 활성 로그 스페이스 상한이 계산됩니다.

  이전 로그 파일의 아카이브, 새 로그 파일의 할당 및 사용 가능한 로그 파일 대기 오버헤드로 인해 로그 파일이 너무 작을 경우 시스템 성능에 영향을 미칠 수 있습니다.
- 1차 로그가 이 크기로 사전 할당되었으므로 디스크 스페이스가 부족한 경우 *logfilsiz*의 값을 줄여야 합니다.

  일부 미디어는 전체 로그 파일을 보유할 수 없으므로 아카이브된 로그 파일 및 로그 파일 사본을 관리할 때 로그 파일이 너무 크면 유연성이 낮아질 수 있습니다.

로그 보존을 사용 중인 경우 데이터베이스에서 마지막 응용프로그램의 연결이 끊어지면 현재 활성 로그 파일이 닫히고 절단됩니다. 나중에 데이터베이스에 연결할 때는 다음 로그 파일이 사용됩니다. 따라서 동시 응용프로그램의 로깅 요구사항을 파악한 경우 로그 파일 크기를 판별할 수 있으므로 스페이스를 초과로 할당하지 않게 됩니다.

loghead - 처음에 사용되는 로그 파일

이 매개변수에는 현재 사용 중인 로그 파일의 이름이 있습니다.

구성 유형

데이터베이스

매개변수 유형

정보용

logindexbuild - 작성된 로그 인덱스 페이지

이 매개변수는 DB2 롤 포워드 조작 또는 고가용성 재해 복구(HADR) 로그 재생 프로시저에서 인덱스 작성, 재작성 또는 재구성 조작이 로그되어 인덱스가 다시 작성될 수 있는지 여부를 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

Off [On; Off]

logpath - 로그 파일의 위치

이 매개변수에는 로깅에 사용 중인 현재 경로가 들어 있습니다.

구성 유형

데이터베이스

매개변수 유형

정보용

newlogpath 매개변수에 대한 변경이 적용된 후 데이터베이스 관리 프로그램이 설정한 대로 이 매개변수를 직접 변경할 수 없습니다.

데이터베이스가 작성되면 데이터베이스가 있는 디렉토리의 서브디렉토리에 데이터베이스의 복구 로그 파일이 작성됩니다. 기본값은 데이터베이스용으로 작성된 디렉토리에 있는 이름이 SQLOGDIR인 서브디렉토리입니다.

logprimary - 1차 로그 파일 수

이 매개변수를 통해 사전 할당할 1차 로그 파일의 수를 지정할 수 있습니다. 1차 로그 파일은 복구 로그 파일에 할당되는 고정된 스토리지 양을 설정합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

3 [2 - 256]

수치 단위

카운터

할당 시기

- 데이터베이스가 작성될 때
- 로그가 다른 위치로 이동될 때(*logpath* 매개변수 갱신 시 발생함)
- 데이터베이스가 HADR 대기 데이터베이스로 시작되지 않았으면 이 매개변수(*logprimary*)의 값 증가 이후 다음에 데이터베이스가 시작될 때
- 로그 파일이 아카이브되었으며 새 로그 파일이 할당될 때(*logretain* 또는 *userexit* 매개변수를 사용하도록 설정해야 함)

- *logfilsiz* 매개변수가 변경된 경우 HADR 대기 데이터베이스로 시작되지 않았으면 다음 데이터베이스 시작 중 로그 파일의 크기가 조정될 때

사용 가능한 시기

이 매개변수가 낮아지지 않았으면 사용 가능하지 않습니다. 낮아진 경우 데이터베이스에 대한 다음 연결 중 불필요한 로그 파일이 삭제됩니다.

순환 로깅에서 1차 로그는 시퀀스에 따라 반복적으로 사용됩니다. 즉, 로그가 가득 차면 시퀀스에서 다음 1차 로그(사용 가능한 경우)를 사용합니다. 로그 레코드가 있는 모든 작업 단위가 커미트 또는 롤백된 경우 로그는 사용 가능한 것으로 간주됩니다. 시퀀스에서 다음 1차 로그가 사용 불가능한 경우 2차 로그가 할당되어 사용됩니다. 시퀀스의 다음 1차 로그가 사용 가능해지거나 *logsecond* 매개변수가 지정한 한계에 접근하기 전까지 추가 2차 로그가 할당되어 사용됩니다. 이러한 2차 로그 파일이 더 이상 데이터베이스 관리 프로그램에 필요하지 않으면 동적으로 할당 해제됩니다.

기본 및 2차 로그 파일 수는 다음 사항을 준수해야 합니다.

- *logsecond*의 값이 -1인 경우 *logprimary* <= 256입니다.
- *logsecond*의 값이 -1이 아닌 경우 (*logprimary* + *logsecond*) <= 256입니다.

권장사항: 이 매개변수에 선택한 값은 사용 중인 로깅 유형, 로그 파일의 크기 및 처리 환경의 유형(예: 트랜잭션의 길이 및 커미트 빈도)과 같은 여러 가지 요인에 따라 다릅니다.

이 값을 늘리면 데이터베이스에 처음 연결할 때 1차 로그 파일이 사전 할당되므로 로그의 디스크 요구사항이 증가합니다.

2차 로그 파일이 자주 할당된다고 판단된 경우 로그 파일 크기(*logfilsiz*)를 늘리거나 1차 로그 파일 수를 늘려서 시스템 성능을 향상시킬 수 있습니다.

자주 액세스하지 않는 데이터베이스의 경우 디스크 스토리지를 절약하기 위해 매개변수를 2로 설정하십시오. 롤 포워드 복구에 사용되는 데이터베이스의 경우 새 로그를 거의 즉시 할당하는 오버헤드를 방지하기 위해 매개변수를 크게 설정하십시오.

데이터베이스 시스템 모니터를 사용하여 1차 로그 파일의 크기를 조정할 수 있습니다. 평균값이 현재 요구사항을 표시할 수 있으므로 일정 시간 동안 다음의 모니터 값을 관찰하면 결정을 조정하는 데 도움이 됩니다.

- *sec_log_used_top*(사용한 최대 2차 로그 스페이스)
- *tot_log_used_top*(사용한 최대 총 로그 스페이스)
- *sec_logs_allocated*(현재 할당된 2차 로그)

logretain - 로그 유지 사용

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 DB2 버전 9.5 데이터베이스 관리 프로그램에서는 사용되지 않습니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에만 적용됩니다.

이 매개변수는 활성 로그 파일이 유지되며 롤 포워드 복구에 사용 가능한지 여부를 판별합니다.

구성 유형
데이터베이스

매개변수 유형
구성 가능

디폴트 [범위]
No [Recovery; No]

값은 다음과 같습니다.

- No - 로그가 유지되지 않음을 표시합니다.
- Recovery - 로그가 유지됨을 표시하며 포워드 복구에 사용할 수 있습니다.

logretain이 Recovery로 설정되었거나 **userexit**이 Yes로 설정된 경우 활성 로그 파일이 유지되고 롤 포워드 복구에서 사용할 수 있도록 온라인 아카이브 로그 파일이 됩니다. 이를 로그 유지 로깅이라고 합니다.

logretain이 Recovery로 설정되었거나 **userexit**이 Yes로 설정된 경우(또는 둘 다 설정된 경우) 데이터베이스의 전체 백업을 작성해야 합니다. 이 상태는 **backup_pending** 플래그 매개변수로 표시됩니다.

주:

logarchmeth1 또는 **logretain**은 롤 포워드 복구를 사용합니다. 그러나 한 번에 하나의 메소드만 데이터베이스에 사용해야 합니다.

logarchmeth1을 사용 중인 경우 **logretain** 및 **userexit** 구성 매개변수를 설정하지 마십시오. **logretain** 구성 매개변수가 recover로 설정된 경우 **logarchmeth1**의 값은 **logretain**으로 자동 설정됩니다.

아카이브 로깅 및 롤 포워드 복구를 활성하려면 **logretain** 및 **userexit**보다는 **logarchmeth1**(및 **logarchmeth2**)을 사용하는 것이 좋습니다. **logarchmeth1**으로 아직 이주하지 않은 사용자를 지원하기 위해 **logretain** 및 **userexit** 옵션을 유지합니다.

logsecond - 2차 로그 파일 수

이 매개변수는 복구 로그 파일에 사용된 2차 로그 파일의 수를 지정합니다(필요한 경우에만).

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

2 [-1; 0 – 254]

수치 단위

카운터

할당 시기

*logprimary*가 충분하지 않은 경우 필요할 때(아래의 세부사항 참조)

사용 가능한 시기

데이터베이스 관리 프로그램이 더 이상 필요하지 않다고 판별하면 반복적으로.

1차 로그 파일이 가득 차면 2차 로그 파일(*logfilsiz 크기*)은 필요하면 한 번에 하나씩 이 매개변수가 제어하는 최대 수까지 할당됩니다. 오류 코드가 응용프로그램으로 리턴되고 이 매개변수에서 허용하는 것보다 많은 2차 로그 파일이 필요한 경우 데이터베이스가 종료됩니다.

*logsecond*를 -1로 설정한 경우 무한 활성 로그 스페이스를 사용하여 데이터베이스를 구성합니다. 데이터베이스에서 실행 중인 인플라이트(inflight) 트랜잭션의 크기 또는 수에는 제한이 없습니다. *logsecond*를 -1로 설정한 경우 계속 *logprimary* 및 *logfilsiz* 구성 매개변수를 사용하여 데이터베이스 관리 프로그램이 활성 로그 경로에서 보존해야 하는 로그 파일 수를 지정하십시오. 데이터베이스 관리 프로그램이 로그 파일에서 로그 데이터를 읽어야 하지만 이 파일이 활성 로그 경로에 없는 경우 데이터베이스 관리 프로그램은 아카이브에서 활성 로그 경로로 로그 파일을 검색합니다. (데이터베이스 관리 프로그램은 오버플로우 로그 경로(구성한 경우)로 파일을 검색합니다.) 로그 파일이 검색되었으면 데이터베이스 관리 프로그램은 활성 로그 경로에서 이 파일을 캐시하므로 동일한 파일에서 로그 데이터의 기타 읽기 속도가 빨라집니다. 데이터베이스 관리 프로그램은 필요에 따라 이러한 로그 파일의 검색, 캐시 및 제거를 관리합니다.

로그 경로가 원시 디바이스인 경우 *logsecond*를 -1로 설정하기 위해 *overflowlogpath* 구성 매개변수를 구성해야 합니다.

*logsecond*를 -1로 설정하면 작업 단위의 크기 또는 동시 작업 단위 수에 제한이 없습니다. 그러나 아카이브에서 로그 파일을 검색해야 하므로 롤백(세이브포인트 레벨 및 작업 단위 레벨에서)이 매우 느릴 수 있습니다. 동일한 이유로 인해 응급 복구도 매우 느릴 수 있습니다. 데이터베이스 관리 프로그램은 관리 통지 로그에 현재 활성 작업 단위 세트가 1차 로그 파일을 초과했음을 경고하는 메시지를 기록합니다. 이 메시지는 롤백 또는 응급 복구가 매우 느릴 수 있음을 나타내는 것입니다.

*logsecond*를 -1로 설정하려면 *logarchmeth1* 구성 매개변수를 OFF 또는 LOGRETAIN 이외의 다른 값으로 설정해야 합니다.

권장사항: 대량의 로그 스페이스가 주기적으로 필요한 데이터베이스의 경우 2차 로그 파일을 사용하십시오. 예를 들어, 한 달에 한 번 실행되는 응용프로그램은 1차 로그 파일이 제공하는 로그 스페이스보다 많은 로그 스페이스가 필요할 수 있습니다. 2차 로그 파일에는 영구 파일 스페이스가 필요하지 않으므로 이러한 경우에 2차 로그 파일이 유용합니다.

무한 로깅이 사용 가능한 경우(logsecond가 -1로 설정), 데이터베이스 관리 프로그램은 롤백하고 로그 레코드를 작성해야 하는 트랜잭션에 필요한 활성 로그 스페이스를 예약하지 않습니다. 롤백 처리 중에 사용 중인 로그 경로 및 아카이브 목표 둘 다 가득 찬 경우 또는 아카이브 목표에 액세스할 수 없는 경우 blk_log_dsk_ful(디스크 가득참 로그 시 블록 db 구성 매개변수)도 사용 가능으로 설정하여 데이터베이스 오류가 발생하지 않도록 해야 합니다.

max_log - 트랜잭션당 최대 로그

이 매개변수는 트랜잭션이 사용할 수 있는 로그 스페이스 퍼센트에 한계가 있는지 여부 및 그 한계를 지정합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

0 [0 — 100]

수치 단위

퍼센트

값이 0이 아닌 경우 이 매개변수는 한 트랜잭션에서 사용할 수 있는 1차 로그 스페이스의 퍼센트를 표시합니다.

값이 0으로 설정된 경우에는 단일 트랜잭션이 사용할 수 있는 스페이스의 양(총 1차 로그 스페이스의 퍼센트)에 제한이 없습니다. 이것은 버전 8 이전의 트랜잭션 동작이었습니다.

maxappls - 최대 활성 응용프로그램 수

이 매개변수는 데이터베이스에 연결할 수 있는(로컬 및 리모트) 최대 동시 응용프로그램 수를 지정합니다. 데이터베이스에 접속된 각 응용프로그램마다 일부 전용 메모리가 할당되므로 여러 동시 응용프로그램이 잠재적으로 추가 메모리를 사용할 수 있습니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

Automatic [1 - 60 000]

수치 단위

카운터

*maxappls*를 *automatic*으로 설정하면 모든 수의 연결된 응용프로그램을 허용할 수 있습니다. 데이터베이스 관리 프로그램은 새 응용프로그램을 지원하는 데 필요한 자원을 동적으로 할당합니다.

이 매개변수를 *automatic*으로 설정하지 않으려면 이 매개변수의 값은 연결된 응용프로그램 수에 2단계 커미트 또는 롤백을 완료하는 프로세스에서 동시에 존재할 수 있는 동일한 응용프로그램 수를 더한 합계와 같거나 이 합계보다 커야 합니다. 그런 다음 한 번에 존재할 수 있는 예상 인다우트(Indoubt) 트랜잭션 수를 이 합계에 더하십시오.

응용프로그램이 데이터베이스에 연결하려고 시도하지만 *maxappls*에 이미 접근한 경우 데이터베이스에 최대 응용프로그램 수가 연결되었음을 표시하는 오류가 응용프로그램으로 리턴됩니다.

파티션된 데이터베이스 환경에서는 데이터베이스 파티션에 대해 동시에 활성화할 수 있는 최대 응용프로그램 수입니다. 이 매개변수는 서버가 응용프로그램의 코디네이터 노드인지 여부에 관계없이 데이터베이스 파티션 서버의 데이터베이스 파티션에 대해 활성 응용프로그램 수를 제한합니다. 파티션된 데이터베이스 환경에서 모든 응용프로그램은 카탈로그 노드에 연결해야 하므로 파티션된 데이터베이스 환경의 카탈로그 노드는 다른 유형의 환경보다 높은 *maxappls* 값이 필요합니다.

권장사항: *maxlocks* 매개변수를 줄이거나 *locklist* 매개변수를 늘리지 않고 이 매개변수의 값을 늘리면 응용프로그램 한계보다는 잠금에 대한 데이터베이스 한계(*locklist*)에 접근할 수 있으므로 일반적인 잠금 에스컬레이션 문제가 발생합니다.

어느 정도까지는 *max_coordagents*로도 최대 응용프로그램 수를 제어할 수 있습니다. 사용 가능한 연결(*maxappls*)과 사용 가능한 코디네이팅 에이전트(*max_coordagents*)가 있는 경우에만 응용프로그램이 데이터베이스에 연결할 수 있습니다.

maxfilop - 응용프로그램당 열린 최대 데이터베이스 파일 수

이 매개변수는 각 데이터베이스마다 열 수 있는 최대 파일 핸들 수를 지정합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
트랜잭션 경계

디폴트 [범위]

AIX, Sun, HP 및 Linux 64비트
61 440 [64 - 61 440]

Linux 32비트
30 720 [64 - 30 720]

Windows 32비트
32 768 [64 - 32 768]

Windows 64비트
65 335 [64 - 65 335]

수치 단위
카운터

파일을 열었을 때 이 값을 초과한 경우 이 데이터베이스에서 사용 중인 일부 파일이 닫힙니다. *maxfilop*가 너무 작은 경우 파일을 열고 닫는 오버헤드가 많아져서 성능이 저하될 수 있습니다.

SMS 테이블 스페이스 및 DMS 테이블 스페이스 파일 컨테이너는 모두 운영 체제와의 데이터베이스 관리 프로그램 상호 작용에서 파일로 처리되며 파일 핸들이 필요합니다. 일반적으로 DMS 파일 테이블 스페이스에 사용되는 컨테이너 수에 비해 많은 파일을 SMS 테이블 스페이스에서 사용합니다. 따라서 SMS 테이블 스페이스를 사용하는 경우 이 매개변수에는 DMS 파일 테이블 스페이스에 필요한 수에 비해 큰 값을 지정해야 합니다.

또한 이 매개변수를 사용하면 데이터베이스 당 핸들 수를 특정 값으로 제한하여 데이터베이스 관리 프로그램이 사용하는 전체 파일 핸들 수가 운영 체제 한계를 초과하지 않습니다. 실제 수는 동시에 실행 중인 데이터베이스의 수에 따라 다릅니다.

maxlocks - 에스컬레이션 전 잠금 목록의 최대 퍼센트

이 매개변수는 데이터베이스 관리 프로그램이 잠금 에스컬레이션을 수행하기 전에 채워야 하는 응용프로그램이 보유한 잠금 목록의 퍼센트를 정의합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
즉시

디폴트 [범위]
Automatic [1 - 100]

수치 단위
퍼센트

잠금 에스컬레이션은 목록에서 잠금 수를 줄여서 행 잠금을 테이블 잠금으로 교체하는 프로세스입니다. 한 응용프로그램이 보유한 잠금 수가 이러한 총 잠금 목록 *크기* 퍼센트에 접근하면 해당 응용프로그램이 보유한 잠금의 잠금 에스컬레이션이 발생합니다. 잠금 목록에 스페이스가 부족해도 잠금 에스컬레이션이 발생합니다.

데이터베이스 관리 프로그램은 응용프로그램의 잠금 목록을 보고 행 잠금이 가장 많은 테이블을 찾아서 에스컬레이션할 잠금을 판별합니다. 이를 단일 테이블 잠금으로 교체한 후 **maxlocks** 값을 더 이상 초과하지 않는 경우 잠금 에스컬레이션이 중지됩니다. 그렇지 않으면 보유된 잠금 목록의 퍼센트가 **maxlocks** 값 이하가 될 때까지 잠금 에스컬레이션이 계속됩니다. **maxlocks** 매개변수에 **maxappls** 매개변수를 곱한 값은 100 이상이어야 합니다.

이 매개변수가 AUTOMATIC으로 설정된 경우 자체 성능 조정할 수 있습니다. 이 경우 워크로드 요구사항이 변경되면 메모리 조정 프로그램이 이 매개변수가 제어하는 메모리 영역의 *크기*를 동적으로 조정합니다. 메모리 조정 프로그램은 여러 메모리 사용자 간에 메모리 자원을 교환하므로 자체 성능 조정이 활성화되려면 자체 성능 조정에 대해 최소 두 개의 메모리 사용자가 사용 가능해야 합니다.

locklist의 값이 **maxlocks** 매개변수와 함께 조정되므로 **locklist** 매개변수의 자체 성능 조정을 사용 불가능하게 하면 **maxlocks** 매개변수의 자체 성능 조정이 자동으로 사용 불가능해집니다. **locklist** 매개변수의 자체 성능 조정을 사용 가능하게 하면 **maxlocks** 매개변수의 자체 성능 조정이 자동으로 사용 가능해집니다.

이 구성 매개변수의 자동 성능 조정은 데이터베이스에 자체 성능 조정 메모리를 사용할 수 있는 경우에만 발생합니다(**self_tuning_mem** 구성 매개변수가 ON으로 설정됨).

32비트 플랫폼에서는 오브젝트에 기타 잠금이 보유되어 있는지 여부에 따라 각 잠금은 48 또는 96바이트의 잠금 목록이 필요합니다.

- 96바이트는 기타 잠금이 보유되어 있지 않은 오브젝트에서 잠금을 보유할 때 필요합니다.
- 48바이트는 기존 잠금이 보유되어 있는 오브젝트에서 잠금을 기록할 때 필요합니다.

64비트 플랫폼(HP-UX/PA-RISC 제외)에서는 오브젝트에 기타 잠금이 보유되어 있는지 여부에 따라 각 잠금은 64 또는 128바이트의 잠금 목록이 필요합니다.

- 128바이트는 기타 잠금이 보유되어 있지 않은 오브젝트에서 잠금을 보유할 때 필요합니다.
- 64바이트는 기존 잠금이 보유되어 있는 오브젝트에서 잠금을 기록할 때 필요합니다.

64비트 HP-UX/PA-RISC에서는 오브젝트에 기타 잠금이 보유되어 있는지 여부에 따라 각 잠금은 80 또는 160바이트의 잠금 목록이 필요합니다.

권장사항: 응용프로그램이 평균 잠금 수의 2배를 보유할 수 있도록 다음의 공식을 사용하여 **maxlocks**를 설정할 수 있습니다.

```
maxlocks = 2 * 100 / maxappls
```

여기서 2는 평균의 2배로 올리는 데 사용되며 100은 허용되는 최대 퍼센트 값을 나타냅니다. 동시에 실행되는 응용프로그램이 많지 않은 경우 첫 번째 공식 대신 다음 공식을 사용할 수 있습니다.

```
maxlocks = 2 * 100 / (동시에 실행 중인 평균 응용프로그램
수)
```

maxlocks를 설정할 때 고려사항 중 하나는 잠금 목록 *크기*(**locklist**)와 함께 사용하는 것입니다. 잠금 에스컬레이션이 발생하기 전에 응용프로그램이 보유한 잠금 수의 실제 한계는 다음과 같습니다.

- 32비트 시스템의 경우 `maxlocks * locklist * 4096 /(100 * 48)`
- 64비트 시스템 HP-UX/PA-RISC 환경의 경우 `maxlocks * locklist * 4096 /(100 * 80)`
- 기타 64비트 시스템의 경우 `maxlocks * locklist * 4096 /(100 * 64)`

여기서 4096은 한 페이지의 바이트 수이고 100은 **maxlocks**에 허용되는 최대 퍼센트 값이고 48은 32비트 시스템에서 잠금 당 바이트 수이고 80은 HP-UX/PA-RISC 64비트 시스템에서 잠금 당 바이트 수이고 64는 기타 64비트 시스템에서 잠금 당 바이트 수입니다. 응용프로그램 중 하나에 1000개의 잠금이 필요하고 잠금 에스컬레이션이 발생하지 않도록 하려면 결과가 1000보다 크도록 이 공식에서 **maxlocks** 및 **locklist**의

값을 선택해야 합니다. (**maxlocks**에 10을 사용하고 **locklist**에 100을 사용하면 이 공식의 결과는 필요한 1000개의 잠금보다 큽니다.)

maxlocks가 너무 낮게 설정된 경우 다른 동시 응용프로그램에 계속 충분한 잠금이 있으면 잠금 에스컬레이션이 발생합니다. **maxlocks**가 너무 높게 설정된 경우 일부 응용프로그램이 대부분의 잠금 스페이스를 사용할 수 있으며 다른 응용프로그램이 잠금 에스컬레이션을 수행해야 합니다. 이 경우 잠금 에스컬레이션이 필요하여 동시성이 낮아집니다.

데이터베이스 시스템 모니터를 사용하면 이 구성 매개변수를 추적 및 조정하는 데 도움이 됩니다.

min_dec_div_3 - 10진수 나누기 스케일 3으로

이 매개변수는 SQL에서 10진수 나누기의 스케일 계산을 빠르게 변경하는 방법을 제공합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

No [Yes, No]

min_dec_div_3 데이터베이스 구성 매개변수는 나눗셈을 포함하는 10진수 산술 연산의 결과 스케일을 변경합니다. 이 매개변수는 ″Yes″ 또는 ″No″로 설정할 수 있습니다. *min_dec_div_3*의 디폴트값은 ″No″입니다. 값이 ″No″인 경우 스케일은 31-p+s-s'로 계산됩니다. ″Yes″로 설정되면 스케일은 MAX(3, 31-p+s-s')로 계산됩니다. 따라서 10진수 나누기 결과는 항상 최소 3의 배수가 됩니다. 정밀도는 항상 31입니다.

이 데이터베이스 구성 매개변수를 변경하면 기존 데이터베이스의 응용프로그램이 변경될 수 있습니다. 이 데이터베이스 구성 매개변수를 변경하여 10진수 나누기의 결과 스케일이 영향을 받는 경우 이러한 상황이 발생할 수 있습니다. 다음은 응용프로그램에 영향을 줄 수 있는 몇 가지 가능한 시나리오입니다. 이 시나리오는 기존 데이터베이스가 있는 데이터베이스 서버에서 *min_dec_div_3*을 변경하기 전에 고려해야 합니다.

- 뷰 컬럼 중 하나의 결과 스케일이 변경된 경우 데이터베이스 구성 매개변수가 변경된 후에 참조되면 한 설정이 있는 환경에 정의된 뷰는 SQLCODE -344로 실패할 수 있습니다. 메시지 SQL0344N은 재귀 공통 테이블 표현식을 참조하지만 오브젝트 이름(첫 번째 토큰)이 뷰인 경우 이 오류를 피하려면 뷰를 삭제하고 다시 작성해야 합니다.
- 정적 패키지는 내재적 또는 명시적으로 패키지가 리바인드될 때까지 동작을 변경하지 않습니다. 예를 들어, 값을 NO에서 YES로 변경한 후 리바인드가 발생할 때까지

결과에는 추가적인 스케일 숫자가 포함될 수 없습니다. 변경된 정적 패키지의 경우 명시적 REBIND 명령을 사용하여 강제로 리바인드할 수 있습니다.

- 10진수 나누기와 관련된 점검 제한조건은 이전에 승인된 일부 값을 제한할 수 있습니다. 이러한 행은 이제 제한조건을 위반하지만 점검 제한조건 행에 포함된 컬럼 중 하나가 갱신되거나 SET INTEGRITY 문과 IMMEDIATE CHECKED 옵션이 처리될 때까지 발견되지 않습니다. 이러한 제한조건을 강제로 검사하려면 ALTER TABLE 문을 수행하여 점검 제한조건을 삭제(drop)한 후 ALTER TABLE 문을 수행하여 제한조건을 다시 추가하십시오.

주: *min_dec_div_3*에는 또한 다음과 같은 제한이 있습니다.

1. `GET DB CFG FOR DBNAME` 명령은 *min_dec_div_3* 설정을 표시하지 않습니다. 현재 설정을 판별하는 최선의 방법은 10진수 나누기 결과의 부작용을 관찰하는 것입니다. 예를 들어, 다음 명령문을 참조하십시오.

   ```
   VALUES (DEC(1,31,0)/DEC(1,31,5))
   ```

   이 명령문이 sqlcode SQL0419N을 리턴하는 경우 데이터베이스는 *min_dec_div_3*을 지원하지 않습니다. 이 명령문이 `1.000`을 리턴하는 경우 *min_dec_div_3*은 "Yes"로 설정되었습니다.
2. 다음 명령문을 실행할 때 구성 키워드 목록에 *min_dec_div_3*이 나타나지 않습니다. `? UPDATE DB CFG`

mincommit - 그룹화할 커미트 수

이 매개변수를 사용하면 최소 커미트 수를 수행할 때까지 디스크에 로그 레코드 기록이 지연되므로 로그 레코드 기록과 연관된 데이터베이스 관리 프로그램 오버헤드가 줄어듭니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
즉시

디폴트 [범위]
1 [1 – 25]

수치 단위
카운터

데이터베이스에 대해 여러 응용프로그램을 실행 중이며 매우 짧은 시간 프레임 안에 응용프로그램이 여러 커미트를 요청한 경우 이러한 지연으로 성능이 향상됩니다.

이러한 커미트 그룹화는 이 매개변수의 값이 1보다 큰 경우와 데이터베이스에 연결된 응용프로그램 수가 이 매개변수 값 이상인 경우에만 발생합니다. 커미트 그룹화를 수행 중인 경우 1초가 경과되거나 커미트 요청 수가 이 매개변수 값과 동일할 때까지 응용프로그램 커미트 요청이 보류될 수 있습니다.

이 매개변수는 소량(예: 1)씩만 증분시켜야 합니다. 또한 다중 사용자 테스트를 사용하여 이 매개변수 값을 늘리면 예상한 결과가 나타나는지 검증해야 합니다.

이 매개변수에 지정한 값을 변경하면 즉시 적용됩니다. 데이터베이스에서 모든 응용프로그램의 연결이 끊어질 때까지 기다릴 필요가 없습니다.

권장사항: 여러 읽기 쓰기 응용프로그램이 일반적으로 동시 데이터베이스 커미트를 요청하는 경우 이 매개변수를 디폴트값에서 늘리십시오. 이 경우 파일 입출력 로깅이 자주 발생하지 않으며 발생할 때마다 추가 로그 레코드를 작성하므로 효율적입니다.

또한 초 당 트랜잭션 수를 샘플링하고 초 당 최대 트랜잭션 수(또는 이러한 수의 큰 비율)를 수용하여 이 매개변수를 조정할 수 있습니다. 최대 활동을 수용하면 트랜잭션 집중 기간 동안 로그 레코드를 작성하는 오버헤드를 최소화할 수 있습니다.

*mincommit*를 늘릴 경우 *logbufsz* 매개변수도 늘려서 이러한 트랜잭션 집중 기간 중 전체 로그 버퍼가 쓰기를 강제 실행하는 것을 방지해야 합니다. 이 경우 *logbufsz*는 다음과 같아야 합니다.

```
mincommit * (트랜잭션이 사용한 평균 로그 스페이스)
```

데이터베이스 시스템 모니터를 사용하면 다음과 같은 방법으로 이 매개변수를 조정할 수 있습니다.

- 초 당 최대 트랜잭션 수 계산:

  일반적으로 하루 전체의 모니터 샘플을 작성하면 트랜잭션 집중 기간을 판별할 수 있습니다. 다음의 모니터 요소들을 더하여 전체 트랜잭션을 계산할 수 있습니다.

  – *commit_sql_stmts(시도한 커미트 명령문)*
  – *rollback_sql_stmts(시도한 롤백 명령문)*

  이 정보 및 사용 가능한 시간소인을 사용하면 초 당 트랜잭션 수를 계산할 수 있습니다.

- 트랜잭션 당 사용되는 로그 스페이스 계산:

  일정 기간 동안 여러 트랜잭션에 대해 샘플링 기술을 사용하면 다음 모니터 요소에 사용되는 평균 로그 스페이스를 계산할 수 있습니다.

  – *log_space_used(사용한 작업 단위 로그 스페이스)*

mirrorlogpath - 미러 로그 경로

이 매개변수를 사용하여 최대 242바이트의 미러 로그 경로 문자열을 지정할 수 있습니다. 이 문자열은 경로 이름을 가리켜야 하며, 상대 경로 이름이 아니라 완전한 경로여야 합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 경로 또는 디바이스]

주: 단일 또는 다중 파티션 DB2 ESE 환경에서 노드 번호는 자동으로 경로에 추가됩니다. 여러 논리 노드 구성에서 경로의 고유성을 유지하기 위해 이러한 작업이 수행됩니다.

*mirrorlogpath*가 구성된 경우, DB2는 로그 경로와 미러 로그 경로 둘 다에 사용 중인 로그 파일을 작성합니다. 모든 로그 데이터가 두 경로 모두에 작성됩니다. 미러 로그 경로에는 활성 로그 파일의 중복된 설정이 있어서, 경로 중 하나에서 사용 중인 로그 파일을 파괴하는 디스크 오류 또는 사용자 오류가 발생하는 경우 데이터베이스가 계속 작동할 수 있습니다.

미러 로그 경로가 변경되는 경우 이전 미러 로그 경로에 로그 파일이 있습니다. 이 로그 파일은 아카이브되지 않으므로 수동으로 이 로그 파일을 아카이브해야 합니다. 또한 이 데이터베이스에서 복제를 실행 중인 경우 복제에는 로그 경로를 변경하기 전의 로그 파일이 필요할 수 있습니다. User Exit 사용(*userexit*) 데이터베이스 구성 매개변수를 Yes로 설정하여 데이터베이스를 구성하고 모든 로그 파일을 DB2가 자동으로 또는 사용자가 수동으로 아카이브한 경우, DB2는 로그 파일을 검색하여 복제 프로세스를 완료할 수 있습니다. 그렇지 않으면 이전 미러 로그 경로에서 새 미러 로그 경로로 파일을 복사할 수 있습니다.

logpath 또는 *newlogpath*가 로그 파일이 저장된 위치로 원시 디바이스를 지정하는 경우, *mirrorlogpath*로 표시되는 미러 로깅이 허용되지 않습니다. *logpath* 또는 *newlogpath*가 로그 파일이 저장된 위치로 파일 경로를 지정하는 경우 미러 로깅이 허용되며 *mirrorlogpath*도 파일 경로를 지정해야 합니다.

권장사항: 로그 파일과 마찬가지로 미러 로그 파일도 입출력이 높지 않은 실제 디스크에 있어야 합니다.

이 경로를 1차 로그 경로가 아닌 독립 디바이스에 설정할 것을 적극 권장합니다.

데이터베이스 시스템 모니터를 사용하여 데이터베이스 로깅과 관련된 입출력 수를 추적할 수 있습니다.

다음 데이터 요소는 데이터베이스 로깅과 관련된 입출력 활동의 양을 리턴합니다. 운영 체제 모니터 도구를 사용하여 다른 디스크 입출력 활동에 대한 정보를 수집한 다음 두 가지 유형의 입출력 활동을 비교할 수 있습니다.

- *log_reads*(읽은 로그 페이지 수)
- *log_writes*(쓴 로그 페이지 수)

mon_act_metrics - 활동 메트릭 모니터링 구성 매개변수

이 매개변수는 전체 데이터베이스의 활동 메트릭 콜렉션을 제어하며 모든 DB2 워크로드 정의와 연관된 연결을 통해 제출되는 활동에 영향을 미칩니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

BASE [NONE,BASE,EXTENDED]

이 구성 매개변수를 BASE 또는 EXTENDED로 설정하면, 제출된 활동이 연관되는 DB2 워크로드 연결과 상관없이 다음 인터페이스를 통해 수집되는 모든 메트릭이 데이터 서버에서 실행되는 모든 활동에 대해 수집됩니다.

- MON_GET_ACTIVITY_DETAILS
- MON_GET_PKG_CACHE_STMT
- 활동 이벤트 모니터(event_activity 논리 데이터 그룹의 DETAILS_XML 모니터 요소)

이 구성 매개변수를 NONE으로 설정하는 경우, 위의 인터페이스를 통해 보고되는 메트릭은 COLLECT ACTIVITY METRICS절이 BASE 또는 EXTENDED로 설정된 DB2 워크로드와 연관되는 연결이 제출하는 활동의 서브세트에 대해서만 수집됩니다.

mon_deadlock - 교착 상태 모니터링 구성 매개변수

이 매개변수는 잠금 이벤트 모니터와 관련하여 데이터베이스 레벨에서 교착 상태 이벤트 생성을 제어합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

W I T H O U T _ H I S T

[NONE,WITHOUT_HIST,WITH_HIST,HIST_AND_VALUES]

매개변수를 WITHOUT_HIST로 설정하면, 잠금 이벤트가 발생할 때 잠금 이벤트에 관한 데이터가 모든 활성 잠금 이벤트 모니터로 보내집니다. 지나간 활동 실행기록 및 입력 값은 이벤트 모니터로 보내지지 않습니다.

매개변수를 WITH_HIST로 설정하면 이 유형의 모든 잠금 이벤트에 대한 현재 작업 단위(UOW)에서 지나간 활동 실행기록을 수집할 수 있습니다. 활동 실행기록 버퍼는 최대 크기 한계가 사용된 후 래핑됩니다. 보존할 지나간 활동 수에 대한 디폴트 한계는 250입니다. 지나간 활동 수가 한계보다 큰 경우 최신 활동만 보고됩니다.

매개변수 값을 HIST_AND_VALUES로 설정하면 입력 데이터 값이 해당 값을 갖는 활동에 대한 모든 활성 잠금 이벤트 모니터로 보내집니다. 이러한 데이터 값은 LOB 데이터, 변경 시작 LONG VARCHAR 데이터, LONG VARGRAPHIC 데이터, 변경 끝 구조화된 유형 데이터 또는 XML 데이터를 포함합니다.

잠금 이벤트 모니터를 통해 교착 상태를 캡처하려면 잠금 대기자 또는 잠금 보유자에 워크로드가 포함될 수 있으므로 데이터베이스 레벨에서 교착 상태 콜렉션을 사용 가능하게 하십시오. 교착 상태에서 수집된 데이터 레벨을 워크로드 레벨에서 개별적으로 제어하거나 이 매개변수를 사용하여 데이터베이스 레벨에서 설정할 수 있습니다.

mon_locktimeout - 잠금 시간종료 모니터링 구성 매개변수

이 매개변수는 잠금 이벤트 모니터와 관련하여 데이터베이스 레벨에서 잠금 시간종료 이벤트 생성을 제어하며 모든 DB2 워크로드 정의에 영향을 미칩니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

디폴트 [데이터베이스의 모든 워크로드 또는 서비스 클래스에 사용 가능한 콜렉션의 최소 레벨]

NONE [NONE,WITHOUT_HIST,WITH_HIST,HIST_AND_VALUES]

매개변수 값을 NONE으로 설정하면 워크로드에 대한 잠금 시간종료 데이터가 어떤 파티션에서도 수집되지 않습니다.

매개변수를 WITHOUT_HIST로 설정하면, 잠금 이벤트가 발생할 때 잠금 이벤트에 관한 데이터가 모든 활성 잠금 이벤트 모니터로 보내집니다. 지나간 활동 실행기록 및 입력 값은 이벤트 모니터로 보내지지 않습니다.

매개변수를 WITH_HIST로 설정하면 이 유형의 모든 잠금 이벤트에 대한 현재 작업 단위(UOW)에서 지나간 활동 실행기록을 수집할 수 있습니다. 활동 실행기록 버퍼는 최대 *크기* 한계가 사용된 후 래핑됩니다. 보존할 지나간 활동 수에 대한 디폴트 한계는 250입니다. 지나간 활동 수가 한계보다 큰 경우 최신 활동만 보고됩니다.

매개변수 값을 HIST_AND_VALUES로 설정하면 입력 데이터 값이 해당 값을 갖는 활동에 대한 모든 활성 잠금 이벤트 모니터로 보내집니다. 이러한 데이터 값은 LOB 데이터, 변경 시작 LONG VARCHAR 데이터, LONG VARGRAPHIC 데이터, 변경 끝 구조화된 유형 데이터 또는 XML 데이터를 포함합니다.

지정되는 디폴트값은 데이터베이스에서 모든 워크로드 또는 서비스 클래스에 사용되는 최소 콜렉션 레벨을 나타냅니다. 개별 워크로드 또는 서비스 클래스가 상위 레벨의 콜렉션을 지정하는 경우 디폴트값 대신 현재 설정이 해당 서비스 클래스 또는 워크로드에 사용됩니다.

workload1 과 workload2라는 2개의 워크로드가 있고 데이터베이스 레벨 구성 매개변수가 WITHOUT_HIST로 설정된 경우, 데이터베이스 레벨 제어에서 WITHOUT_HIST를 지정하므로 workload1에 필요한 데이터를 수집할 수 있습니다. 매개변수가 NONE 및 WITH_HIST로 설정되면, workload2에 대한 잠금 시간종료 데이터 수집 설정이 WITH_HIST이기 때문에 workload2에 대한 데이터를 수집할 수 있습니다.

mon_lockwait - 잠김 대기 모니터링 구성 매개변수

이 매개변수는 잠금 이벤트 모니터와 관련하여 데이터베이스 레벨에서 잠금 대기 이벤트 생성을 제어합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

NONE [NONE,WITHOUT_HIST,WITH_HISTORY,HIST_AND_VALUES]

매개변수를 NONE으로 설정하면 워크로드에 대한 잠금 시간종료 데이터가 어떤 파티션에서도 수집되지 않습니다.

매개변수를 WITHOUT_HIST로 설정하면, 잠금 이벤트가 발생할 때 잠금 이벤트에 관한 데이터가 모든 활성 잠금 이벤트 모니터로 보내집니다. 지나간 활동 실행기록 및 입력 값은 이벤트 모니터로 보내지지 않습니다.

매개변수를 WITH_HIST로 설정하면 이 유형의 모든 잠금 이벤트에 대한 현재 작업 단위(UOW)에서 지나간 활동 실행기록을 수집할 수 있습니다. 활동 실행기록 버퍼는 최

대 크기 한계가 사용된 후 래핑됩니다. 보존할 지나간 활동 수에 대한 디폴트 한계는 250입니다. 지나간 활동 수가 한계보다 큰 경우 최신 활동만 보고됩니다.

매개변수 값을 HIST_AND_VALUES로 설정하면 입력 데이터 값이 해당 값을 갖는 활동에 대한 모든 활성 잠금 이벤트 모니터로 보내집니다. 이러한 데이터 값은 LOB 데이터, 변경 시작 LONG VARCHAR 데이터, LONG VARGRAPHIC 데이터, 변경 끝 구조화된 유형 데이터 또는 XML 데이터를 포함합니다.

이 매개변수는 잠금 이벤트 모니터에 대한 데이터베이스 레벨에서의 **잠금 대기 콜렉션** 이벤트를 제어하며 모든 DB2 워크로드 정의에 영향을 줍니다. **mon_lockwait** 구성 매개변수는 잠금 대기 이벤트가 수집되기 전에 잠금 대기에서 소비되는 시간을 제어하는 **mon_lw_thresh** 구성 매개변수와 함께 사용됩니다.

mon_lw_thresh - 잠금 대기 임계값 모니터링 구성 매개변수

이 매개변수는 **mon_lockwait**의 이벤트가 생성되기 전에 잠금 대기 상태에 있는 기간을 제어합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

디폴트 [범위]
5000000 [1000 ... MAX_INT]

수치 단위
마이크로초

이 매개변수를 데이터베이스 레벨과 워크로드 레벨 둘 다에서 설정하는 경우, 구성된 두 개의 시간 중 짧은 시간이 지정된 워크로드에 적용됩니다.

mon_obj_metrics - 오브젝트 메트릭 모니터링 구성 매개변수

이 매개변수는 전체 데이터베이스의 데이터 오브젝트 메트릭 콜렉션을 제어합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

디폴트 [범위]
BASE [NONE,BASE,EXTENDED]

이 구성 매개변수를 BASE 또는 EXTENDED로 설정하면, 다음 인터페이스를 통해 보고되는 모든 메트릭이 수집됩니다.

- MON_GET_BUFFERPOOL
- MON_GET_TABLESPACE
- MON_GET_CONTAINER

이 구성 매개변수를 NONE으로 설정하면, 위에서 언급한 인터페이스를 통해 보고되는 메트릭이 갱신되지 않습니다.

mon_req_metrics - 요청 메트릭 모니터링 구성 매개변수

이 매개변수는 전체 데이터베이스의 요청 메트릭 콜렉션을 제어하고 모든 DB2 서비스 클래스에서 실행 중인 요청에 영향을 줍니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

BASE [NONE,BASE,EXTENDED]

이 구성 매개변수를 BASE 또는 EXTENDED로 설정하면 다음 인터페이스를 통해 보고되는 모든 메트릭이 요청이 실행되는 DB2 서비스 클래스와 상관없이 데이터 서버에서 실행되는 모든 요청에 대해 수집됩니다.

- MON_GET_UNIT_OF_WORK
- MON_GET_UNIT_OF_WORK_DETAILS
- MON_GET_CONNECTION
- MON_GET_CONNECTION_DETAILS
- MON_GET_SERVICE_SUBCLASS
- MON_GET_SERVICE_SUBCLASS_DETAILS
- MON_GET_WORKLOAD
- MON_GET_WORKLOAD_DETAILS
- 통계 이벤트 모니터(event_wlstats 및 event_scstats 논리 데이터 그룹의 DETAILS_XML 모니터 요소)
- 작업 단위(UOW) 이벤트 모니터

이 구성 매개변수를 NONE으로 설정하면 위의 인터페이스를 통해 보고되는 메트릭은 서비스 수퍼 클래스가 BASE 또는 EXTENDED로 설정된 COLLECT REQUEST METRICS 절을 갖는 DB2 서비스 클래스에서 실행 중인 요청의 서브세트에 대해서만 수집됩니다.

mon_uow_data - 작업 단위(UOW) 이벤트 모니터링 구성 매개변수

이 매개변수는 작업 단위(UOW) 이벤트 모니터에 대한 데이터베이스 레벨에서의 작업 단위 생성 이벤트를 제어하며 데이터 서버의 작업 단위에 영향을 줍니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

NONE [NONE,BASE]

이 매개변수는 작업 단위(UOW)가 완료될 때 작업 단위에 관한 정보(트랜잭션이라고도 함)가 활성 작업 단위 이벤트 모니터로 보내지는지 여부를 지정합니다.

이 매개변수가 BASE로 설정되면, 데이터 서버에서 실행되는 모든 작업 단위(UOW)에 관한 정보가 작업 단위가 완료될 때 활성 작업 단위 이벤트 모니터로 보내집니다. 매개변수가 NONE으로 설정되면, 정보는 COLLECT UNIT OF WORK DATA 절이 BASE로 설정되는 DB2 워크로드에서 실행되는 작업 단위에 대한 작업 단위 이벤트 모니터로만 보내집니다.

디폴트 설정은 NONE입니다. 작업 단위 종료 시에 수집되는 정보에는 해당 작업 단위에 대한 시스템 레벨 요청 메트릭(예: 작업 단위 중에 사용되는 CPU의 양)이 포함됩니다. 이러한 요청 메트릭의 콜렉션은 DB2 서비스 수퍼 클래스의 COLLECT REQUEST METRICS절이나 **mon_req_metrics** 데이터베이스 구성 매개변수를 사용하여 작업 단위 데이터의 콜렉션으로부터 독립적으로 제어됩니다. 요청 메트릭 콜렉션을 사용할 수 없는 경우 작업 단위 데이터의 파트로서 수집되는 모든 요청 메트릭의 값은 0입니다.

multipage_alloc - 다중 페이지 파일 할당 사용

다중 페이지 파일 할당은 삽입 성능을 향상시키는 데 사용됩니다. 이 매개변수는 SMS 테이블 스페이스에만 적용됩니다. 사용할 경우 모든 SMS 테이블 스페이스가 영향을 받습니다. 즉, 개별 SMS 테이블 스페이스를 선택할 수 없습니다.

구성 유형

데이터베이스

매개변수 유형

정보용

매개변수의 디폴트는 Yes이며 다중 페이지 파일 할당을 사용합니다.

데이터베이스 작성 후 이 매개변수를 No로 설정할 수 없습니다. 다중 페이지 파일 할당을 사용하도록 설정한 후 사용하지 않도록 설정할 수 없습니다. db2empfa 도구를 사용하여 현재 사용하지 않는 데이터베이스의 다중 페이지 파일 할당을 사용하도록 설정할 수 있습니다.

newlogpath - 데이터베이스 로그 경로 변경

이 매개변수를 사용하면 최대 242바이트의 문자열을 지정하여 로그 파일이 저장되는 위치를 변경할 수 있습니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

Null [모든 유효한 경로 또는 디바이스]

문자열은 경로 이름 또는 원시 디바이스를 지정할 수 있습니다. DB2 버전 9 이후로 데이터베이스 로깅 시 원시 디바이스가 사용되지 않습니다. 원시 로그를 사용하는 대신 직접 입출력(DIO) 또는 동시 입출력(CIO)을 사용할 수 있습니다.

문자열이 경로 이름을 지정하는 경우 상대 경로 이름이 아닌 완전한 경로 이름이어야 합니다.

단일 또는 다중 파티션 DB2 ESE 환경에서 노드 번호는 자동으로 경로에 추가됩니다. 여러 논리 노드 구성에서 경로의 고유성을 유지하기 위해 이러한 작업이 수행됩니다.

복제를 사용하려고 하며 로그 경로가 원시 디바이스인 경우 *overflowlogpath* 구성 매개변수를 구성해야 합니다.

디바이스를 지정하려면 운영 체제가 디바이스로 식별한 문자열을 지정하십시오. 예:

- Windows에서는 `\\.\d:` 또는 `\\.\PhysicalDisk5`

  주: 디바이스에 로그를 기록하려면 Windows 버전 4.0과 서비스 팩 3을 설치해야 합니다.

- Linux 및 UNIX 플랫폼에서는 `/dev/rdblog8`

주: AIX, Windows 2000, Windows, Solaris, HP-UX 및 Linux 플랫폼에서만 디바이스를 지정할 수 있습니다.

다음의 작업이 모두 발생할 때까지 새 설정이 *logpath*의 값이 되지 않습니다.

- *database_consistent* 매개변수에 표시된 바와 같이 데이터베이스가 일관성 있는 상태입니다.

- 모든 응용프로그램이 데이터베이스에서 연결이 끊어집니다.

데이터베이스에 처음 새로 연결된 경우 데이터베이스 관리 프로그램은 *logpath*에 지정된 새 위치로 로그를 이동합니다.

이전 로그 경로에 로그 파일이 있을 수 있습니다. 이러한 로그 파일은 아카이브되지 않았을 수 있습니다. 이 경우 로그 파일을 수동으로 아카이브해야 합니다. 또한 이 데이터베이스에서 복제를 실행 중인 경우 복제에는 로그 경로를 변경하기 전의 로그 파일이 필요할 수 있습니다. User Exit 사용(*userexit*) 데이터베이스 구성 매개변수를 Yes로 설정하여 데이터베이스를 구성하고 DB2가 자동으로 또는 사용자 자신이 수동으로 모든 로그 파일을 아카이브한 경우 DB2는 로그 파일을 검색하여 복제 프로세스를 완료할 수 있습니다. 그렇지 않으면 이전 로그 경로에서 새 로그 경로로 파일을 복사할 수 있습니다.

logpath 또는 *newlogpath*가 로그 파일이 저장된 위치로 원시 디바이스를 지정한 경우 *mirrorlogpath*에 표시된 미러 로그는 허용되지 않습니다. *logpath* 또는 *newlogpath*가 로그 파일이 저장된 위치로 파일 경로를 지정하는 경우 미러 로깅이 허용되며 *mirrorlogpath*도 파일 경로를 지정해야 합니다.

권장사항: 원래 로그 파일은 입출력이 많지 **않은** 실제 디스크에 있습니다. 예를 들어, 운영 체제 또는 대용량 데이터베이스와 동일한 디스크에 로그를 저장하지 않도록 방지하십시오. 따라서 입출력 대기와 같은 최소의 오버헤드로 효율적인 로그 활동이 가능합니다.

데이터베이스 시스템 모니터를 사용하여 데이터베이스 로깅과 관련된 입출력 수를 추적할 수 있습니다.

모니터 요소 *log_reads*(읽은 로그 페이지 수) 및 *log_writes*(쓴 로그 페이지 수)는 데이터베이스 로깅과 관련된 입출력 활동의 양을 리턴합니다. 운영 체제 모니터 도구를 사용하여 다른 디스크 입출력 활동에 대한 정보를 수집한 다음 두 가지 유형의 입출력 활동을 비교할 수 있습니다.

DB2 고가용성 재해 복구(HADR) 데이터베이스 쌍의 기본 및 대기 데이터베이스에 대한 로그 경로로 공유하는 네트워크 또는 로컬 파일 시스템을 사용하지 마십시오. 기본 및 대기 데이터베이스에는 각각 트랜잭션 로그의 사본이 있으며 기본 데이터베이스는 대기 데이터베이스로 로그를 전달합니다. 기본 및 대기 데이터베이스의 로그 경로가 동일한 실제 위치를 지정한 경우 기본 및 대기 데이터베이스는 각 로그 사본에 동일한 실제 파일을 사용합니다. 데이터베이스 관리 프로그램이 공유 로그 경로를 발견한 경우 데이터베이스 관리 프로그램이 오류를 리턴합니다.

num_db_backups - 데이터베이스 백업 수

이 매개변수는 데이터베이스에 대해 보유할 데이터베이스 백업 수를 지정합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

트랜잭션 경계

디폴트 [범위]

12 [1 - 32 767]

지정된 백업 수에 접근한 후 이전 백업은 복구 실행기록 파일에서 만기됨으로 표시됩니다. 만기된 데이터베이스 백업에 관련된 테이블 스페이스 백업 및 로드 사본 백업의 복구 실행기록 파일 항목도 만기됨으로 표시됩니다. 백업이 만기됨으로 표시된 경우 실제 백업은 저장된 위치(예: 디스크, 테이프, TSM)에서 제거될 수 있습니다. 다음 데이터베이스 백업은 만기된 항목을 복구 실행기록 파일에서 프룬(prune)합니다.

rec_his_retentn 구성 매개변수는 *num_db_backups* 값과 호환 가능한 값으로 설정해야 합니다. 예를 들어, *num_db_backup*이 큰 값으로 설정된 경우 *rec_his_retentn* 값은 백업 수를 지원할 수 있도록 충분히 커야 합니다.

num_freqvalues - 보유한 자주 사용되는 값 수

이 매개변수를 사용하여 RUNSTATS 명령에 WITH DISTRIBUTION 옵션이 지정된 경우 수집할 『가장 자주 사용되는 값』의 수를 지정할 수 있습니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

10 [0 - 32 767]

수치 단위

카운터

이 매개변수의 값을 늘리면 통계를 수집할 때 사용되는 통계 힙(*stat_heap_sz*)의 양이 늘어납니다.

『가장 자주 사용되는 값』 통계는 옵티마이저가 컬럼 내의 데이터 값 분산을 이해하는 데 도움이 됩니다. 값이 클수록 쿼리 옵티마이저는 더 많은 정보를 사용할 수 있으나

추가적인 카탈로그 스페이스가 필요합니다. 0이 지정되면 분산 통계를 수집하도록 요청하는 경우에도 자주 사용되는 값 통계가 보유되지 않습니다.

또한 테이블이나 컬럼 레벨에서 RUNSTATS 명령의 파트로 보유되는 자주 사용되는 값 수를 지정할 수 있습니다. NUM_FREQVALUES 옵션 사용. 아무것도 지정하지 않으면 *num_freqvalues* 구성 매개변수 값이 사용됩니다. RUNSTATS 명령을 통해 보유되는 자주 사용되는 값 수를 변경하는 것이 *num_freqvalues* 데이터베이스 구성 매개변수를 사용하여 변경하는 것보다 쉽습니다.

이 매개변수를 갱신하면 불균등하게 분산되는 데이터의 일부 술어(=, <, >)에 대해 옵티마이저가 더 나은 선택성 추정치를 얻을 수 있습니다. 선택성 계산이 정확할수록 더 효율적인 액세스 플랜이 선택됩니다.

이 매개변수의 값을 변경한 후 다음을 수행해야 합니다.

- RUNSTATS 명령을 다시 실행하여 변경된 자주 사용되는 값 수로 통계를 수집하십시오.
- 정적 SQL 또는 XQuery문을 포함하는 패키지를 리바인드하십시오.

RUNSTATS를 사용하는 경우, 테이블 레벨과 컬럼 레벨 둘 다에서 수집되는 자주 사용되는 값 수를 제한할 수 있습니다. 그러면 분산 통계가 이용할 수 없는 컬럼에 대한 분산 통계를 줄이고 중요한 컬럼의 정보를 사용하여 카탈로그에서 차지하는 스페이스를 최적화할 수 있습니다.

권장사항: 이 매개변수를 갱신하려면 일반적으로 선택 술어가 있는 가장 중요한 테이블의 가장 중요한 컬럼에서 불균등 분포의 등급을 판별해야 합니다. 컬럼에 있는 각 값의 발생 수 순위를 제공하는 SQL SELECT 문을 사용하여 이를 수행할 수 있습니다. 불균등 분산, 고유, Long 또는 LOB 컬럼을 고려할 필요가 없습니다. 이 매개변수에 알맞은 실제 값 범위는 10 - 100입니다.

자주 사용되는 값 통계를 수집하려면 상당한 CPU 및 메모리(*stat_heap_sz*) 자원이 필요합니다.

num_iocleaners - 비동기 페이지 클리너 수

이 매개변수를 사용하여 데이터베이스의 비동기 페이지 클리너 수를 지정할 수 있습니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

Automatic [0 – 255]

수치 단위

카운터

이 페이지 클리너 쓰기는 데이터베이스 에이전트에서 버퍼 풀의 스페이스가 필요하기 전에 버퍼 풀에서 디스크로 페이지를 변경했습니다. 그 결과 데이터베이스 에이전트는 버퍼 풀의 스페이스를 사용할 수 있도록 변경된 페이지가 작성되기를 기다릴 필요가 없습니다. 따라서 데이터베이스 응용프로그램의 전체 성능이 향상됩니다.

이 매개변수를 0으로 설정하면 페이지 클리너가 시작되지 않으며, 따라서 데이터베이스 에이전트가 버퍼 풀에서 디스크로 전체 페이지 쓰기를 수행합니다. 이 경우 디바이스 중 하나가 유휴 상태가 될 기회가 많아지므로 이 매개변수는 많은 실제 스토리지 디바이스에 저장된 데이터베이스의 성능에 상당한 영향을 미칠 수 있습니다. 페이지 클리너가 구성되지 않은 경우 응용프로그램은 주기적 로그 전체 조건에 직면할 수 있습니다.

이 매개변수가 AUTOMATIC으로 설정된 경우, 시작된 페이지 클리너 수는 현재 머신에 구성된 CPU의 수와 파티션된 데이터베이스의 로컬 논리적 데이터베이스 파티션의 수에 따라 달라집니다. 이 매개변수가 AUTOMATIC으로 설정된 경우에는 항상 하나 이상의 페이지 클리너가 시작됩니다.

이 매개변수가 AUTOMATIC으로 설정된 경우 시작할 페이지 클리너 수는 다음 공식을 사용하여 계산됩니다.

```
페이지 클리너 수 = max( ceil( # CPUs / # local logical DPs ) – 1, 1 )
```

이 공식을 사용하면 페이지 클리너 수가 논리적 데이터베이스 파티션에 거의 균등하게 분산되고 CPU에 있는 만큼의 페이지 클리너만 있도록 할 수 있습니다.

데이터베이스의 응용프로그램이 주로 데이터를 갱신하는 트랜잭션으로 구성되는 경우, 클리너 수가 증가하면 성능이 좋아집니다. 페이지 클리너 수를 늘리면 디스크에 있는 데이터베이스의 컨텐츠가 최신 상태로 유지되므로 정전 등 소프트 장애로부터의 복구 시간도 줄어듭니다.

권장사항: 이 매개변수의 값을 설정하는 경우 다음 인수를 고려하십시오.

- 응용프로그램 유형
  - 갱신사항이 없는 쿼리 전용 데이터베이스인 경우에는 이 매개변수를 0으로 설정하십시오. 쿼리 작업 로드의 결과로 많은 TEMP 테이블이 작성되는 경우는 예외입니다. (Explain 유틸리티를 사용하여 이를 판별할 수 있습니다.)
  - 트랜잭션이 데이터베이스에 대해 실행되는 경우, 이 매개변수를 1과 데이터베이스에 사용되는 실제 스토리지 디바이스 수 사이의 값으로 설정하십시오.
- 워크로드

  갱신 트랜잭션 비율이 높은 환경에서는 페이지 클리너를 더 많이 구성해야 합니다.
- 버퍼 풀 크기

버퍼 풀이 큰 환경에서도 페이지 클리너를 더 많이 구성해야 합니다.

데이터베이스 시스템 모니터를 사용하면 버퍼 풀에서 쓰기 활동에 대한 이벤트 모니터의 정보를 사용하여 이 구성 매개변수를 조정하는 데 도움이 됩니다.

- 다음 조건이 둘 다 참인 경우에는 매개변수를 줄일 수 있습니다.
  - *pool_data_writes*가 *pool_async_data_writes*와 거의 같음
  - *pool_index_writes*가 *pool_async_index_writes*와 거의 같음
- 다음 조건 중 하나가 참인 경우에는 매개변수를 증가시켜야 합니다.
  - *pool_data_writes*가 *pool_async_data_writes*보다 훨씬 큼
  - *pool_index_writes*가 *pool_async_index_writes*보다 훨씬 큼

num_ioservers - 입출력 서버 수

이 매개변수는 데이터베이스의 입출력 서버 수를 지정합니다. 언제든지 프리페치 및 유틸리티에 대해 이 입출력 수만 데이터베이스에 대해 진행 중일 수 있습니다.

구성 유형
데이터베이스

매개변수 유형
구성 가능

디폴트 [범위]
Automatic [1 - 255]

수치 단위
카운터

할당 시기
응용프로그램이 데이터베이스에 연결된 경우

사용 가능한 시기
응용프로그램이 데이터베이스에서 연결이 끊어질 때

입출력 서버(프리페처라고도 함)는 백업과 리스토어 등의 유틸리티로 프리페치 입출력 및 비동기 입출력을 수행하는 데이터베이스 에이전트의 동작에 사용됩니다. 입출력 서버는 시작된 입출력 조작이 진행 중인 동안 기다립니다. 비프리페치 입출력은 데이터베이스 에이전트에서 직접 스케줄되므로 결과는 *num_ioservers*의 제한을 받지 않습니다.

이 매개변수가 AUTOMATIC으로 설정된 경우, 시작된 프리페치 수는 현재 데이터베이스 파티션에 있는 테이블 스페이스의 병렬 처리 설정에 따라 달라집니다. (병렬 처리 설정은 DB2_PARALLEL_IO 환경 변수가 제어합니다.) 각 DMS 테이블 스페이스의 경우, 이 병렬 처리 설정의 값에 테이블 스페이스 스트라이프 세트에 있는 컨테이너의 최대수가 곱해집니다. 각 SMS 테이블 스페이스의 경우, 이 병렬 처리 설정의 값에 테이블 스페이스에 있는 컨테이너의 수가 곱해집니다. 현재 데이터베이스 파티션에 있는

모든 테이블 스페이스에 대한 최대 결과가 시작할 프리페치의 수로 사용됩니다. 이 매개변수가 AUTOMATIC으로 설정된 경우, 항상 세 개 이상의 프리페처가 시작됩니다.

이 매개변수가 AUTOMATIC으로 설정된 경우, 시작할 프리페처의 수는 데이터베이스 활성화 시간에 다음 공식에 따라 계산됩니다.

```
number of prefetchers = max( max over all table spaces
( parallelism setting * [SMS: # containers;
  DMS: max # containers in stripe set] ), 3 )
```

권장사항: 시스템의 모든 입출력 디바이스를 전체적으로 이용하려면 일반적으로 데이터베이스가 있는 실제 디바이스의 수보다 1이나 2 더 큰 값을 사용하는 것이 좋습니다. 각 입출력 서버와 연결된 최소 오버헤드가 있고 사용되지 않는 입출력 서버가 유휴 상태로 남아 있으므로 추가적인 입출력 서버를 구성하는 것이 더 좋습니다.

num_log_span - 로그 범위 수

이 매개변수는 한 트랜잭션이 걸칠 수 있는 로그 파일의 수에 한계가 있는지 여부 및 그 한계를 지정합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
즉시

디폴트 [범위]
0 [0 - 65 535]

수치 단위
카운터

값이 0이 아닌 경우 이 매개변수는 한 활성 트랜잭션이 걸칠 수 있는 활성 로그 파일의 수를 표시합니다.

값이 0으로 설정된 경우에는 단일 트랜잭션이 걸칠 수 있는 로그 파일의 수에 제한이 없습니다. 이것은 버전 8 이전의 트랜잭션 동작이었습니다.

num_quantiles - 컬럼의 Quantile 수

이 매개변수는 RUNSTATS 명령에 WITH DISTRIBUTION 옵션이 지정된 경우 수집할 Quantile 수를 제어합니다.

구성 유형
데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

20 [0 - 32 767]

수치 단위

카운터

이 매개변수의 값을 늘리면 통계를 수집할 때 사용되는 통계 힙(*stat_heap_sz*)의 양이 늘어납니다.

『Quantile』 통계는 옵티마이저가 컬럼 내의 데이터 값 분산을 이해하는 데 도움이 됩니다. 값이 클수록 쿼리 옵티마이저는 더 많은 정보를 사용할 수 있으나 추가적인 카탈로그 스페이스가 필요합니다. 0 또는 1이 지정되면 분산 통계를 수집하도록 요청하는 경우에도 Quantile 통계가 보유되지 않습니다.

NUM_QUANTILES 옵션을 사용하여 테이블 또는 컬럼 레벨에서 RUNSTATS 명령의 파트로 수집되는 Quantile 수를 지정할 수도 있습니다. 아무것도 지정하지 않으면 *num_quantiles* 구성 매개변수 값이 사용됩니다. RUNSTATS 명령을 통해 수집되는 Quantile 수를 변경하는 것이 *num_quantiles* 데이터베이스 구성 매개변수를 사용하여 변경하는 것보다 쉽습니다.

이 매개변수를 갱신하면 불균등하게 분산되는 데이터의 범위 술어에 대해 더 나은 선택성 추정을 얻을 수 있습니다. 다른 옵티마이저 결정 중에서 이 정보는 인덱스 스캔과 테이블 스캔 중 어느 것을 선택할 것인지에 강력한 영향을 미칩니다. (자주 발생하는 값 범위에 액세스하는 경우에는 테이블 스캔을 사용하는 것이 더 효율적이며, 자주 발생하지 않는 값 범위의 경우에는 인덱스 스캔을 사용하는 것이 더 효율적입니다.)

이 매개변수의 값을 변경한 후 다음을 수행해야 합니다.

- RUNSTATS 명령을 다시 실행하여 변경된 자주 사용되는 값 수로 통계를 수집하십시오.
- 정적 SQL 또는 XQuery문을 포함하는 패키지를 리바인드하십시오.

RUNSTATS를 사용하는 경우, 테이블 레벨과 컬럼 레벨 둘 다에서 수집되는 Quantile 수를 제한할 수 있습니다. 그러면 분산 통계가 이용할 수 없는 컬럼에 대한 분산 통계를 줄이고 중요한 컬럼의 정보를 사용하여 카탈로그에서 차지하는 스페이스를 최적화할 수 있습니다.

권장사항: 이 매개변수의 기본값은 단면 범위 술어(>, >=, < 또는 <=)에 대해 약 2.5%의 최대 추정 오차와 BETWEEN 술어에 대해 약 5%의 최대 오차를 보증합니다. Quantile 수에 근접하는 간단한 방법은 다음과 같습니다.

- 범위 쿼리의 행 수를 추정할 때 퍼센트 P로 허용 가능한 최대 오차를 판별하십시오.
- Quantile 수는 대부분의 BETWEEN 술어에서 약 100/P이고 기타 유형의 범위 술어(<, <=, > 또는 >=)에서 약 50/P여야 합니다.

예를 들어, 25 Quantile은 최대 추정 오차가 BETWEEN 술어의 경우 4%이고 ″>″ 술어의 경우 2%여야 합니다. 이 매개변수에 알맞은 실제 값 범위는 10 - 50입니다.

numarchretry - 오류 시 재시도 수

이 매개변수는 DB2가 장애 복구 디렉토리에 로그 파일을 아카이브하려고 시도하기 전에 기본 또는 보조 디렉토리에 로그 파일을 아카이브하려고 시도하는 횟수를 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

5 [0 - 65 535]

이 매개변수는 *failarchpath* 데이터베이스 구성 매개변수가 설정된 경우에만 사용됩니다. *numarchretry*가 설정되지 않으면 DB2는 1차 또는 2차 로그 경로에 아카이브를 연속적으로 재시도합니다.

numsegs - 디폴트 SMS 컨테이너 수

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 DB2 버전 9.5 데이터베이스 관리 프로그램에서는 사용되지 않습니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에만 적용됩니다.

구성 유형

데이터베이스

매개변수 유형

정보용

수치 단위

카운터

이 매개변수는 디폴트 테이블 스페이스에서 작성되는 컨테이너 수를 표시합니다. 또한 CREATE DATABASE 명령에 명시적 또는 내재적으로 지정되었는지 여부에 관계없이 데이터베이스 작성 시 사용한 정보를 표시합니다.

이 매개변수는 SMS 테이블 스페이스에만 적용됩니다. CREATE TABLESPACE문은 어떠한 방식으로든 이 매개변수를 **사용하지 않습니다**.

number_compat - 숫자 호환성 데이터베이스 구성 매개변수

이 매개변수는 NUMBER 데이터 유형에 연결된 호환성 시맨틱이 연결된 데이터베이스에 적용되는지 여부를 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

이 값은 데이터베이스를 작성할 때 NUMBER 지원의 DB2_COMPATIBILITY_VECTOR 레지스트리 변수 설정에 따라 결정됩니다. 이 값은 변경할 수 없습니다.

overflowlogpath - 오버플로우 로그 경로

이 매개변수는 DB2가 롤 포워드 조작에 필요한 로그 파일을 찾을 위치와 아카이브에서 검색한 사용 중인 로그 파일을 저장할 위치를 지정합니다. 또한 db2ReadLog API를 사용하는 데 필요한 로그 파일을 찾고 저장할 위치를 제공합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

NULL [유효한 경로]

이 매개변수는 로깅 요구사항에 따라 여러 기능에서 사용됩니다.

- 이 매개변수를 사용하여 DB2가 롤 포워드 조작에 필요한 로그 파일을 찾을 위치를 지정할 수 있습니다. 이것은 ROLLFORWARD 명령의 OVERFLOW LOG PATH 옵션과 비슷합니다. 모든 ROLLFORWARD 명령에 항상 OVERFLOW LOG PATH를 지정하는 대신 이 구성 매개변수를 한 번만 설정하면 됩니다. 그러나 둘 다 사용되는 경우에는 OVERFLOW LOG PATH 옵션이 특정 롤 포워드 조작에 대해 *overflowlogpath* 구성 매개변수를 겹쳐씁니다.
- *logsecond*가 -1로 설정된 경우에는 *overflowlogpath*를 사용하여 DB2가 아카이브에서 검색한 사용 중인 로그 파일을 저장할 디렉토리를 지정할 수 있습니다. (사용 중인 로그 파일이 더 이상 사용 중인 로그 경로에 있지 않은 경우, 롤백 조작을 위해 사용 중인 로그 파일을 검색해야 합니다.) overflowlogpath가 없는 경우 DB2는 로그 파일을 사용 중인 로그 경로에서 검색합니다. *overflowlogpath*를 사용하면 DB2가 검색한 로그 파일을 저장할 추가적인 자원을 제공할 수 있습니다. 다른 디스크로 입출력 비용을 전개하고 사용 중인 로그 경로에 더 많은 로그 파일이 저장되도록 허용할 수 있다는 이점이 있습니다.
- 복제를 위해 db2ReadLog API(DB2 V8 이전에서는 db2ReadLog를 sqlurlog라고 했음)를 사용해야 하는 경우, 예를 들어 *overflowlogpath*를 사용하여 DB2가 이 API에 필요한 로그 파일을 검색할 위치를 지정할 수 있습니다. 로그 파일을 찾을 수 없는 경우(사용 중인 로그 경로 또는 오버플로우 로그 경로에서) 및 데이터베이스가 *userexit* 사용 가능으로 구성된 경우, DB2는 로그 파일을 검색합니다. *overflowlogpath*를 사용하여 DB2가 검색한 로그 파일을 저장할 디렉토리를 지정할 수도 있습니다. 사용 중인 로그 경로에서 입출력 비용을 줄이고 사용 중인 로그 경로에 더 많은 로그 파일을 저장할 수 있는 이점이 있습니다.
- 사용 중인 로그 경로에 대해 원시 디바이스를 구성한 경우, *logsecond*를 -1로 설정하거나 db2ReadLog API를 사용하려면 *overflowlogpath*를 구성해야 합니다.

*overflowlogpath*를 설정하려면 최대 242바이트의 문자열을 지정하십시오. 이 문자열은 경로 이름을 가리켜야 하며, 상대 경로 이름이 아니라 완전한 경로여야 합니다. 경로 이름은 원시 디바이스가 아니라 디렉토리여야 합니다.

주: 단일 또는 다중 파티션 DB2 ESE 환경에서 노드 번호는 자동으로 경로에 추가됩니다. 여러 논리 노드 구성에서 경로의 고유성을 유지하기 위해 이러한 작업이 수행됩니다.

pagesize - 데이터베이스 디폴트 페이지 크기

이 매개변수에는 데이터베이스를 작성할 때 디폴트 페이지 크기로 사용된 값이 있습니다. 가능한 값은 4 096, 8 192, 16 384 및 32 768입니다. 데이터베이스에서 버퍼 풀 또는 테이블 스페이스가 작성되는 경우 동일한 디폴트 페이지 크기가 적용됩니다.

구성 유형

데이터베이스

매개변수 유형

정보용

pckcachesz - 패키지 캐시 크기

이 매개변수는 데이터베이스 공유 메모리로부터 할당되며 데이터베이스의 정적 및 동적 SQL문과 XQuery문에 대한 섹션 캐시에 사용됩니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

32비트 운영 체제

Automatic [-1, 32 - 128 000]

64비트 운영 체제

Automatic [-1, 32 - 2 147 483 646]

수치 단위

페이지(4KB)

할당 시기

데이터베이스가 초기화될 때

사용 가능한 시기

데이터베이스가 종료될 때

파티션된 데이터베이스 시스템에는 각 데이터베이스 파티션마다 하나의 패키지 캐시가 있습니다.

패키지 캐싱을 통해 데이터베이스 관리 프로그램은 패키지를 다시 로드할 때 시스템 카탈로그에 액세스할 필요가 없거나 동적 SQL 또는 XQuery문의 경우 컴파일할 필요가 없으므로 내부 오버헤드를 줄일 수 있습니다. 다음 중 하나가 발생할 때까지 패키지 캐시에 섹션이 보존됩니다.

- 데이터베이스 종료
- 패키지 또는 동적 SQL문이나 XQuery문 무효화
- 캐시 스페이스 부족

정적 또는 동적 SQL문이나 XQuery문에 대한 섹션 캐싱은 특히 데이터베이스에 연결된 응용프로그램이 동일한 명령문을 여러 번 사용하는 경우 성능을 향상시킬 수 있습니다. 트랜잭션 처리 환경에서 특히 중요합니다.

이 매개변수가 AUTOMATIC으로 설정된 경우 자체 성능 조정할 수 있습니다. *self_tuning_mem*이 ON으로 설정된 경우 워크로드 요구사항이 변경되면 메모리 조정 프로그램이 *pckcachesz*가 제어하는 메모리 영역의 크기를 동적으로 조정합니다. 메모리 조정 프로그램은 여러 메모리 사용자 간에 메모리 자원을 교환하므로 자체 성능 조정이 활성화되려면 자체 성능 조정에 대해 최소 두 개의 메모리 사용자가 사용 가능해야 합니다.

이 구성 매개변수의 자동 성능 조정은 데이터베이스에 자체 성능 조정 메모리를 사용할 수 있는 경우에만 발생합니다(*self_tuning_mem* 구성 매개변수가 "ON"으로 설정됨).

이 매개변수가 -1로 설정된 경우 페이지 할당을 계산하는 데 사용되는 값은 *maxappls* 구성 매개변수에 지정된 값의 8배입니다. *maxappls*의 8배가 32 미만인 경우에는 예외입니다. 이 경우 디폴트값 -1은 *pckcachesz*를 32로 설정합니다.

권장사항: 이 매개변수를 조정할 때 패키지 캐시를 위해 확보 중인 추가 메모리가 버퍼 풀 또는 카탈로그 캐시와 같은 다른 용도로 할당된 경우 보다 효과적인지 여부를 고려해야 합니다. 이러한 이유로 인해 매개변수를 조정할 때 벤치마킹 기술을 사용해야 합니다.

처음에 여러 섹션을 사용한 후 일부만 반복적으로 실행되는 경우 특히 이 매개변수를 조정하는 것이 중요합니다. 캐시가 너무 큰 경우 초기 섹션의 사본을 보유하는 데 메모리를 낭비할 수 있습니다.

다음의 모니터 요소는 이 구성 매개변수를 조정해야 하는지 여부를 판별하는 데 유용할 수 있습니다.

- *pkg_cache_lookups*(패키지 캐시 찾아보기)
- *pkg_cache_inserts*(패키지 캐시 삽입)
- *pkg_cache_size_top*(패키지 캐시 상위 워터 마크(water mark))
- *pkg_cache_num_overflows*(패키지 캐시 오버플로우)

주: 패키지 캐시는 작업 캐시이므로 이 매개변수를 0으로 설정할 수 없습니다. 현재 실행 중인 SQL 또는 XQuery문의 모든 섹션을 보유할 수 있도록 이 캐시에 충분한 메모리가 할당되어 있어야 합니다. 현재 필요한 것보다 많은 스페이스가 할당된 경우 섹션을 캐싱합니다. 이러한 섹션은 로드 또는 컴파일하지 않아도 간단하게 다음에 필요할 때 실행할 수 있습니다.

pckcachesz 매개변수에 지정된 한계는 소프트 한계입니다. 데이터베이스 공유 세트에서 메모리가 계속 사용 가능한 경우 필요하면 이 한계를 초과할 수 있습니다. *pkg_cache_size_top* 모니터 요소를 사용하여 패키지 캐시가 확장된 최대값을 판별하고 *pkg_cache_num_overflows* 모니터 요소를 사용하여 *pckcachesz* 매개변수에 지정된 한계를 초과한 횟수를 판별할 수 있습니다.

priv_mem_thresh - 전용 메모리 임계값

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 DB2 버전 9.5 데이터베이스 관리 프로그램에서는 사용되지 않습니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에만 적용됩니다.

구성 유형

데이터베이스 관리 프로그램

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

구성 가능

디폴트 [범위]

20 000 [-1; 32 - 112 000]

수치 단위

페이지(4KB)

이 매개변수를 사용하여 계속 할당되며 시작된 새 에이전트가 사용할 준비가 된 미사용 에이전트 전용 메모리의 양을 판별합니다. 이 사항은 Linux 및 UNIX 플랫폼에 적용되지 않습니다.

-1 값을 사용하면 이 매개변수가 *min_priv_mem* 매개변수의 값을 사용합니다.

권장사항: 이 매개변수를 설정할 때 클라이언트 연결/연결 끊기 패턴과 동일한 머신에서 다른 프로세스의 메모리 요구사항을 고려해야 합니다.

짧은 기간 동안에만 데이터베이스에 여러 클라이언트가 동시에 연결되어 있는 경우 최고 임계값을 사용하면 미사용 메모리가 커미트 취소되어 다른 프로세스가 사용할 수 있게 됩니다. 이 경우 메모리 관리가 약해져서 메모리가 필요한 다른 프로세스에 영향을 줄 수 있습니다.

동시 클라이언트 수가 일정하고 이 수가 자주 변동되는 경우 최고 임계값을 사용하면 클라이언트 프로세스에 메모리를 사용할 수 있게 되며 메모리를 할당 및 할당 해제하는 오버헤드가 감소합니다.

rec_his_retentn - 복구 실행기록 보존 기간

이 매개변수는 백업에서 실행기록 정보를 보유할 일 수를 지정합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

366 [-1; 0 - 30 000]

수치 단위

일 수

복구 실행기록 파일의 백업, 리스토어 및 로드를 계속 추적할 필요가 없는 경우 이 매개변수를 작은 수로 설정할 수 있습니다.

이 매개변수의 값이 -1인 경우, 전체 데이터베이스 백업을 표시하는 항목 수(및 데이터베이스 백업에 연결된 테이블 스페이스 백업 수)는 *num_db_backups* 매개변수가 지정하는 값에 해당합니다. 복구 실행기록 파일의 기타 항목은 사용 가능 명령 또는 API를 사용하여 명시적으로만 프룬(prune)할 수 있습니다.

보존 기간의 길이에 관계없이 PRUNE 유틸리티와 FORCE 옵션을 사용하는 경우 외에는 최신 전체 데이터베이스 백업과 리스토어 세트가 항상 보존됩니다.

restore_pending - 리스토어 보류

이 매개변수는 데이터베이스에 RESTORE PENDING 상태의 존재 여부를 나타냅니다.

구성 유형

데이터베이스

매개변수 유형

정보용

restrict_access - 데이터베이스 액세스 제한 구성 매개변수

이 매개변수는 디폴트 조치의 제한된 세트를 사용하여 데이터베이스가 작성되었는지 여부를 표시합니다. 즉, CREATE DATABASE 명령에 RESTRICTIVE절이 사용되어 작성되었는지 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

YES 이 데이터베이스를 작성할 때 CREATE DATABASE 명령에 RESTRICTIVE 절이 사용되었습니다.

NO 이 데이터베이스를 작성할 때 CREATE DATABASE 명령에 RESTRICTIVE절이 사용되지 않았습니다.

rollfwd_pending - 롤 포워드 보류 표시기

이 매개변수는 롤 포워드 복구가 필요한지 여부와 필요한 위치를 알립니다.

구성 유형
데이터베이스

매개변수 유형
정보용

이 매개변수는 다음의 상태 중 하나를 표시할 수 있습니다.

- **DATABASE** - 이 데이터베이스에 롤 포워드 복구 프로시저가 필요함을 의미합니다.
- **TABLESPACE** - 하나 이상의 테이블 스페이스를 롤 포워드해야 함을 의미합니다.
- **NO** - 데이터베이스가 사용 가능하며 롤 포워드 복구가 필요하지 않음을 의미합니다.

데이터베이스 또는 테이블 스페이스에 액세스하려면 먼저 복구(ROLLFORWARD DATABASE 사용)를 완료해야 합니다.

self_tuning_mem - 자체 성능 조정 메모리

이 매개변수는 메모리 조정 프로그램이 자체 성능 조정을 사용하는 메모리 사용자 간에 필요한 사용 가능 메모리 자원을 동적으로 분산하는지 여부를 결정합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
즉시

디폴트 [범위]

단일 데이터베이스 파티션 환경
ON [ON; OFF]

다중 데이터베이스 파티션 환경
OFF [ON; OFF]

이전 버전에서 업그레이드된 데이터베이스의 경우 *self_tuning_mem*은 OFF로 설정됩니다.

메모리는 메모리 사용자 간에 교환되므로 메모리 조정 프로그램이 활성화되려면 최소 두 개의 메모리 사용자가 사용 가능해야 합니다. *self_tuning_mem*이 ON으로 설정되었으나 자체 성능 조정에 대해 사용 가능한 메모리 사용자가 둘 미만인 경우에는 메모리 조정 프로그램이 비활성입니다. (정렬 힙 메모리 영역의 경우는 예외입니다. 이 경우에는 다른 메모리 소비자가 자체 성능 조정이 가능한지 여부에 관계 없이 조정될 수 있습니다.) *database_memory*가 숫자 값으로 설정되는 경우에는 자체 성능 조정이 가능한 것으로 간주됩니다.

이 매개변수는 단일 데이터베이스 파티션 환경에서 디폴트로 ON입니다. 다중 데이터베이스 파티션 환경에서는 디폴트로 OFF입니다.

자체 성능 조정을 사용할 수 있는 메모리 사용자에는 다음이 포함됩니다.

- 버퍼 풀(ALTER BUFFERPOOL 및 CREATE BUFFERPOOL 문의 size 매개변수로 제어)
- 패키지 캐시(*pckcachesz* 구성 매개변수로 제어)
- 잠금 목록(*locklist* 및 *maxlocks* 구성 매개변수로 제어)
- 정렬 힙(*sheapthres_shr* 및 *sortheap* 구성 매개변수로 제어)
- 데이터베이스 공유 메모리(*database_memory* 구성 매개변수로 제어)

이 매개변수의 현재 설정을 보려면 GET DATABASE CONFIGURATION 명령문에 SHOW DETAIL 매개변수를 지정하여 사용하십시오. 이 매개변수에 설정할 수 있는 값은 다음과 같습니다.

```
Self Tuning Memory          (SELF_TUNING_MEM) = OFF
Self Tuning Memory          (SELF_TUNING_MEM) = ON (Active)
Self Tuning Memory          (SELF_TUNING_MEM) = ON (Inactive)
Self Tuning Memory          (SELF_TUNING_MEM) = ON
```

다음 값이 표시됩니다.

- `ON (Active)` - 메모리 조정 프로그램이 시스템의 메모리를 적극적으로 조정합니다.
- `ON (Inactive)` - 매개변수가 ON으로 설정되었으나 자체 성능 조정에 대해 사용 가능한 메모리 사용자가 둘 미만이므로 자체 성능 조정이 발생하지 않습니다.
- ON (Active) 또는 (Inactive) 없음 - SHOW DETAIL 옵션 또는 데이터베이스 연결이 없는 쿼리에서 발생합니다.

파티션된 환경에서 *self_tuning_mem* 구성 매개변수는 조정 프로그램이 실행 중인 데이터베이스 파티션에 대해 ON (Active)만 표시합니다. 다른 모든 노드에서 *self_tuning_mem*은 ON (Inactive)을 표시합니다. 따라서 파티션된 환경에서 메모리 조정 프로그램이 활성인지 판별하려면 모든 데이터베이스 파티션에서 *self_tuning_mem* 매개변수를 확인해야 합니다.

DB2의 이전 버전에서 DB2 버전 9로 업그레이드한 경우 자체 성능 조정 메모리 기능을 사용하려면 다음 Health 표시기가 임계값 또는 상태 점검을 사용하지 않도록 구성해야 합니다.

- 공유 정렬 메모리 활용 - db.sort_shrmem_util
- 오버플로우된 정렬의 퍼센트 - db.spilled_sorts
- 장기 공유 정렬 메모리 활용 - db.max_sort_shrmem_util
- 잠금 목록 활용 - db.locklist_util
- 잠금 에스컬레이션 비율 - db.lock_escal_rate
- 패키지 캐시 사용 비율 - db.pkgcache_hitratio

자체 성능 조정 메모리 기능의 목표 중 하나는 즉시 필요하지 않은 메모리를 메모리 사용자에게 할당하지 않는 것입니다. 그러므로 메모리 사용자에게 할당된 메모리의 활용이 100%에 접근해야 더 많은 메모리가 할당됩니다. 이 Health 표시기를 사용하지 않으면 메모리 사용자의 메모리 활용 비율이 높을 때 불필요한 경보가 표시되지 않도록 할 수 있습니다.

DB2 버전 9에서 작성된 인스턴스에서는 Health 표시기가 디폴트로 사용 안함 상태입니다.

seqdetect - 순차적 발견 플래그

이 매개변수는 데이터베이스 관리 프로그램이 입출력 활동 중에 순차 페이지 읽기를 발견할 수 있는지 여부를 제어합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
즉시

디폴트 [범위]
Yes [Yes; No]

데이터베이스 관리 프로그램은 입출력을 모니터할 수 있으며, 순차 페이지 읽기가 발생하는 경우 입출력 프리페치를 활성화할 수 있습니다. 이 유형의 순차 프리페치를 *순차적 발견*이라고 합니다.

이 매개변수가 No로 설정되면 프리페치는 데이터베이스 관리 프로그램이 프리페치가 유용함을 아는 경우에만 발생합니다(예: 테이블 정렬, 테이블 스캔 또는 리스트 프리페치).

권장사항: 대부분의 경우, 이 매개변수에 디폴트값을 사용해야 합니다. 심각한 쿼리 성능 문제점을 정정하기 위해 다른 조정 방법을 사용할 수 없는 경우에만 순차적 발견을 해제하십시오.

sheapthres_shr - 공유 정렬에 대한 정렬 힙 임계값

이 매개변수는 정렬 메모리 사용자가 한 번에 사용할 수 있는 데이터베이스 공유 메모리의 총량에 대한 소프트 한계를 나타냅니다.

구성 유형

데이터베이스

적용 대상

OLAP 기능

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

32비트 플랫폼

Automatic [250 - 524 288]

64비트 플랫폼

Automatic [250 - 2 147 483 647]

수치 단위

페이지(4KB)

정렬 외에 해시 조인, 인덱스 ANDing, 블록 인덱스 ANDing, 병합 조인 및 메모리 내 테이블과 같은 다른 정렬 메모리 사용자가 있습니다. 공유 정렬 메모리 사용자의 총 공유 메모리 양이 *sheapthres_shr* 한계에 접근하는 경우, 메모리 억제 메커니즘이 활성화되어 나중의 공유 정렬 메모리 사용자 요청에는 요청보다 적은 메모리가 부여되지만 항상 태스크를 완료하는 데 필요한 최소량보다 많은 양이 부여됩니다. 일단 *sheapthres_shr* 한계를 초과하면 정렬 메모리 사용자의 모든 공유 정렬 메모리 요청에는 태스크를 완료하는 데 필요한 최소량의 메모리가 부여됩니다. 사용 중인 공유 정렬 메모리 사용자의 총 공유 메모리 양이 이 한계에 접근하면 후속 정렬이 실패할 수 있습니다(SQL0955C).

데이터베이스 관리 프로그램 구성 매개변수의 값 *sheapthres*가 0인 경우 데이터베이스의 모든 정렬 메모리 사용자는 개인용 정렬 메모리 대신 *sheapthres_shr*로 데이터베이스 공유 메모리를 사용합니다.

*sheapthres_shr*이 AUTOMATIC으로 설정된 경우, 자체 성능 조정에 대해 사용 가능합니다. 이 경우 워크로드 요구사항이 변경되면 메모리 조정 프로그램이 이 매개변수가 제어하는 메모리 영역의 크기를 동적으로 조정합니다. 메모리 조정 프로그램은 여러 메모리 사용자 간에 메모리 자원을 교환하므로 자체 성능 조정이 활성화되려면 자체 성능 조정에 대해 최소 두 개의 메모리 사용자가 사용 가능해야 합니다. 메모리 사용자에는 SHEAPTHRES_SHR, PCKCACHESZ, BUFFER POOL(각 버퍼 풀을 하나로 셈), LOCKLIST 및 DATABASE_MEMORY가 포함됩니다.

*sheapthres_shr*의 자동 성능 조정은 데이터베이스 관리 프로그램 구성 매개변수 *sheapthres*가 0으로 설정된 경우에만 허용됩니다.

*sortheap*의 값은 *sheapthres_shr* 매개변수와 함께 조정됩니다. 그러므로 *sortheap* 매개변수의 자체 성능 조정을 사용하지 않으면 자동으로 *sheapthres_shr* 매개변수의 자체 성능 조정이 사용 불가능해 집니다. *sheapthres_shr* 매개변수의 자체 성능 조정을 사용 가능하게 하면 *sortheap* 매개변수의 자체 성능 조정이 자동으로 사용 가능해집니다.

이 구성 매개변수의 자동 성능 조정은 데이터베이스에 자체 성능 조정 메모리를 사용할 수 있는 경우에만 발생합니다(*self_tuning_mem* 구성 매개변수가 "ON"으로 설정됨).

이 매개변수의 값이 온라인으로 갱신되는 경우에는 갱신 후 공유 정렬 메모리에 대한 새 요청만 새 값을 사용합니다. *sheapthres_shr* 값을 줄이기 전에 *sortheap*의 값을 줄이고 *sortheap*의 값을 늘리기 전에 *sheapthres_shr*의 값을 늘릴 것을 권장합니다.

데이터베이스 관리 프로그램 구성 매개변수 *sheapthres*가 0보다 큰 경우, *sheapthres_shr*은 다음 두 가지 경우에만 의미가 있습니다.

- *intra_parallel* 데이터베이스 관리 프로그램 구성 매개변수가 *yes*로 설정된 경우. *intra_parallel*이 *no*로 설정된 경우에는 공유 정렬이 없습니다.
- 집중기가 작동 상태인 경우(즉, *max_connections*가 *max_coordagents*보다 큰 경우). 정렬은 WITH HOLD 옵션으로 선언한 커서를 사용하므로 공유 메모리에서 할당됩니다.

softmax - 복구 범위 및 소프트 체크포인트 간격

이 매개변수는 소프트 체크포인트의 빈도 및 복구 범위를 판별하므로 응급 복구 프로세스에서 도움이 됩니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

100 [1 – 100 * *logprimary*]

수치 단위

하나의 1차 로그 파일 크기 퍼센트

이 매개변수는 다음과 같은 경우에 사용됩니다.

- 손상(예: 정전) 이후 복구해야 하는 로그 수에 영향을 주는 경우. 예를 들어, 디폴트 값을 사용하는 경우 데이터베이스 관리 프로그램은 복구해야 하는 로그 수를 1로 유지합니다. 이 매개변수의 값으로 300을 지정하면 데이터베이스 관리 프로그램은 복구해야 하는 로그 수를 3으로 유지합니다.

  응급 복구에 필요한 로그 수에 영향을 주기 위해 데이터베이스 관리 프로그램은 이 매개변수를 사용하여 페이지 클리너를 트리거하므로 지정된 복구 창보다 오래된 페이지가 디스크에 이미 기록되어 있습니다.
- 소프트 체크포인트의 빈도를 판별하는 경우.

정전과 같은 상태로 인해 발생하는 데이터베이스 오류 시 다음과 같은 데이터베이스가 변경되었을 수 있습니다.

- 커미트되지 않았지만 버퍼 풀의 데이터를 갱신한 데이터베이스
- 커미트되었지만 버퍼 풀에서 디스크로 기록되지 않은 데이터베이스
- 커미트되었으며 버퍼 풀에서 디스크로 기록된 데이터베이스

데이터베이스를 재시작하면 로그 파일을 사용하여 데이터베이스의 응급 복구를 수행하므로 데이터베이스는 일관성 있는 상태가 됩니다(즉, 커미트된 모든 트랜잭션이 데이터베이스에 적용되고 언커미트된 모든 트랜잭션이 데이터베이스에 적용되지 않음).

데이터베이스에 적용해야 하는 로그 파일의 레코드를 판별하기 위해 데이터베이스 관리 프로그램은 로그 제어 파일에 기록된 정보를 사용합니다. (데이터베이스 관리 프로그램은 두 개의 로그 제어 파일 사본 SQLOGCTL.LFH.1 및 SQLOGCTL.LFH.2를 실제로 유지보수하므로 한 사본이 손상되면 데이터베이스 관리 프로그램은 계속 다른 사본을 사용할 수 있습니다.) 이러한 로그 제어 파일은 디스크에 주기적으로 기록되며 이 이벤트의 빈도에 따라 데이터베이스 관리 프로그램은 커미트된 트랜잭션의 로그 레코드를 적용하거나 버퍼 풀에서 디스크로 이미 기록된 변경사항에 대해 설명하는 로그 레코드를 적용할 수 있습니다. 이러한 로그 레코드는 데이터베이스에 영향을 주지 않지만 이러한 레코드를 적용하면 데이터베이스 재시작 프로세스에 일부 오버헤드가 발생할 수 있습니다.

로그 파일이 가득 찬 경우 및 소프트 체크포인트 중 로그 제어 파일이 항상 디스크에 기록됩니다. 이 구성 매개변수를 사용하여 추가 소프트 체크포인트를 트리거할 수 있습니다.

소프트 체크포인트의 시간 제어는 *logfilsiz*의 퍼센트로 제공된 『현재 상태』와 『기록된 상태』 간 차이에 따라 다릅니다. 『기록된 상태』는 디스크의 로그 제어 파일에 표시된 가장 오래된 유효한 로그 레코드로 판별되지만 『현재 상태』는 메모리의 로그 제어 정보로 판별됩니다. (가장 오래된 유효한 로그 레코드는 복구 프로세스가 읽는 첫 번째 로그 레코드입니다.) 다음 공식으로 계산된 값이 이 매개변수 값 이상인 경우 소프트 체크포인트를 수행합니다.

```
( (기록된 상태와 현재 상태 간 공백) / logfilsiz ) * 100
```

권장사항: 승인할 수 있는 복구 창이 하나의 로그 파일보다 크거나 작은지 여부에 따라 이 매개변수의 값을 늘리거나 줄일 수 있습니다. 이 매개변수의 값을 줄이면 데이터베이스 관리 프로그램은 페이지 클리너를 자주 트리거하고 소프트 체크포인트를 자주 수행합니다. 이러한 조치로 인해 처리해야 하는 로그 레코드 수와 응급 복구 중 처리되는 남은 로그 레코드 수가 모두 줄어들 수 있습니다.

그러나 추가 페이지 클리너가 트리거되고 소프트 체크포인트를 자주 수행할 경우 데이터베이스 로깅과 연관된 오버헤드가 증가하므로 데이터베이스 관리 프로그램의 성능에 영향을 줄 수 있습니다. 또한 소프트 체크포인트를 자주 수행하면 다음과 같은 경우 데이터베이스를 재시작하는 데 필요한 시간이 줄어듭니다.

- 커미트 지점이 거의 없는 매우 긴 트랜잭션이 있는 경우
- 커미트된 트랜잭션이 포함된 매우 큰 버퍼 풀 및 페이지가 매우 자주 디스크에 다시 기록되지 않는 경우(비동기 페이지 클리너를 사용하면 이 상황을 방지할 수 있음)

이러한 두 경우에 모두 메모리에 보존된 로그 제어 정보는 자주 변경되지 않으며 변경되지 않은 경우 디스크에 로그 제어 정보를 기록해도 아무 효과가 없습니다.

sortheap - 정렬 힙 크기

이 매개변수는 개인용 정렬에 사용할 최대 전용 메모리 페이지 수 또는 공유 정렬에 사용할 최대 공유 메모리 페이지 수를 정의합니다.

구성 유형

데이터베이스

적용 대상

OLAP 기능

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

32비트 플랫폼

Automatic [16 - 524 288]

64비트 플랫폼

Automatic [16 - 4 194 303]

수치 단위

페이지(4KB)

할당 시기

정렬을 수행할 때 필요하면

사용 가능한 시기

정렬이 완료되었을 때

정렬이 개인용 정렬인 경우 이 매개변수는 에이전트 전용 메모리에 영향을 줍니다. 정렬이 공유 정렬인 경우 이 매개변수는 데이터베이스 공유 메모리에 영향을 줍니다. 각 정렬에는 데이터베이스 관리 프로그램이 필요에 따라 할당한 개별 정렬 힙이 있습니다. 이 정렬 힙에서 데이터가 정렬됩니다. 옵티마이저가 지시한 경우 이 매개변수가 지정한 것보다 작은 정렬 힙이 옵티마이저가 제공한 정보를 사용하여 할당됩니다.

이 매개변수가 AUTOMATIC으로 설정된 경우 자체 성능 조정할 수 있습니다. 이 경우 워크로드 요구사항이 변경되면 메모리 조정 프로그램이 이 매개변수가 제어하는 메모리 영역의 *크기*를 동적으로 조정합니다.

sortheap의 값이 **sheapthres_shr** 매개변수와 함께 조정되므로 **sheapthres_shr** 매개변수의 자체 성능 조정을 사용 불가능하게 해야만 **sortheap** 매개변수의 자체 성능 조정을 사용 불가능하게 할 수 있습니다. **sheapthres_shr** 매개변수의 자체 성능 조정을 사용 가능하게 하면 **sortheap** 매개변수의 자체 성능 조정이 자동으로 사용 가능해집니다. 그러나 **sheapthres_shr** 매개변수를 AUTOMATIC으로 설정하지 않아도 **sortheap** 매개변수를 자체 성능 조정에 사용할 수 있습니다.

데이터베이스 관리 프로그램 구성 매개변수 **sheapthres**가 0으로 설정된 경우에만 **sortheap**의 자동 성능 조정이 허용됩니다.

이 구성 매개변수의 자동 성능 조정은 데이터베이스에 자체 성능 조정 메모리를 사용할 수 있는 경우에만 발생합니다(**self_tuning_mem** 구성 매개변수가 ON으로 설정됨).

권장사항: 정렬 힙에 대해 작업 중인 경우 다음 사항을 고려해야 합니다.

- 적합한 인덱스는 정렬 힙의 사용을 최소화할 수 있습니다.
- 해시 조인 버퍼, 블록 인덱스 ANDing, 병합 조인, 메모리의 테이블 및 동적 비트맵(인덱스 ANDing 및 스타 조인에 사용됨)이 정렬 힙 메모리를 사용합니다. 이러한 기술을 사용하는 경우 이 매개변수의 *크기*를 늘리십시오.
- 대규모 정렬이 자주 필요한 경우 이 매개변수의 *크기*를 늘리십시오.
- 이 매개변수의 값을 늘릴 때 데이터베이스 관리 프로그램 구성 파일의 **sheapthres** 및 **sheapthres_shr** 매개변수도 조정해야 하는지 여부를 조사해야 합니다.

- 정렬 힙 *크기*는 옵티마이저가 액세스 경로를 판별할 때 사용합니다. 이 매개변수를 변경한 후 응용 프로그램을 리바인드해야 합니다(REBIND 명령 사용).

sortheap 값이 갱신된 경우 데이터베이스 관리 프로그램은 현재 또는 새 정렬의 새 값을 즉시 사용합니다.

stat_heap_sz - 통계 힙 크기

이 매개변수는 RUNSTATS 명령을 사용하여 통계를 수집하는 데 사용되는 *최대* 힙 크기를 표시합니다.

버전 9.5에서 이 데이터베이스 구성 매개변수는 디폴트값 AUTOMATIC이며 *appl_memory* 한계 또는 *instance_memory* 한계에 접근할 때까지 필요에 따라 증가함을 의미합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

디폴트 [범위]
Automatic [1 096 - 524 288]

수치 단위
페이지(4KB)

할당 시기
RUNSTATS 유틸리티가 시작될 때

사용 가능한 시기
RUNSTATS 유틸리티가 완료될 때

권장사항: 디폴트 설정 AUTOMATIC을 권장합니다.

stmtheap - 명령문 힙 크기

이 매개변수는 SQL 또는 XQuery문의 컴파일 중 이 컴파일러의 작업 스페이스로 사용되는 명령문 힙의 *크기*를 지정합니다.

버전 9.5에서 이 데이터베이스 구성 매개변수는 디폴트값 AUTOMATIC이며 *appl_memory* 한계 또는 *instance_memory* 한계에 접근할 때까지 필요에 따라 증가함을 의미합니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스

명령문 경계

디폴트 [범위]

32비트 및 64비트 플랫폼의 경우

Automatic [128 - 524 288]

수치 단위

페이지(4KB)

할당 시기

프리컴파일 또는 바인드 중 각 명령문마다

사용 가능한 시기

각 명령문의 프리컴파일 또는 바인드가 완료되었을 때

이 영역은 영구적으로 할당되지 않지만 처리된 모든 SQL 또는 XQuery문마다 할당 및 해제됩니다. 동적 SQL 또는 XQuery문의 경우 프로그램을 실행하는 동안 이 작업 영역을 사용하는 반면 정적 SQL 또는 XQuery문의 경우 프로그램 실행 중이 아닌 바인드 프로세스 중 사용됩니다.

권장사항: 대부분의 경우 이 매개변수의 디폴트 AUTOMATIC 설정을 승인할 수 있습니다. AUTOMATIC으로 설정된 경우 컴파일의 동적 프로그래밍 조인 열거 단계 중 할당되는 총 메모리 양에 대한 내부 한계가 있습니다. 이 한계를 초과한 경우 명령문은 그리디(Greedy) 조인 열거를 사용하여 컴파일하고 나머지 *appl_memory* 또는 *instance_memory*의 양 또는 둘 다로만 제한됩니다. 응용프로그램이 SQL0437W 경고를 수신하며 쿼리의 런타임 성능을 승인할 수 없는 경우 동적 조인 열거를 항상 사용하도록 수동 *stmtheap* 값을 충분히 크게 설정할 수 있습니다.

주: 동적 조인 열거는 최적화 클래스 3 이상(디폴트는 5)에서만 발생합니다.

territory - 데이터베이스 지역

이 매개변수는 데이터베이스를 작성하는 데 사용된 지역을 표시합니다. *territory*는 데이터베이스 관리 프로그램이 지역에 따라 다른 데이터를 처리하는 데 사용합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

trackmod - 수정된 페이지 추적 사용

이 매개변수는 백업 유틸리티가 증분식 백업에서 데이터베이스 페이지의 어느 서브세트를 조사해야 하는지 발견하고 잠재적으로 백업 이미지에 포함할 수 있도록 데이터베이스 관리 프로그램이 데이터베이스 수정을 추적할지 여부를 지정합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

No [Yes, No]

이 매개변수를 ″Yes″로 설정한 후에는 증분식 백업에 대해 기준선이 생기도록 전체 데이터베이스 백업을 수행해야 합니다. 또한 이 매개변수가 사용되고 테이블 스페이스가 작성되는 경우에는 해당 테이블 스페이스를 포함하는 백업을 수행해야 합니다. 이 백업은 데이터베이스 백업 또는 테이블 스페이스 백업이 될 수 있습니다. 백업한 후 증분식 백업에는 이 테이블 스페이스가 포함될 수 있습니다.

tsm_mgmtclass - Tivoli Storage Manager 관리 클래스

Tivoli Storage Manager 관리 클래스는 TSM 서버가 백업 중인 오브젝트의 백업 버전을 관리하는 방식을 판별합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

Null [모든 문자열]

디폴트는 DB2 지정 관리 클래스가 없는 것입니다.

TSM 백업을 수행할 때 데이터베이스 구성 매개변수가 지정한 관리 클래스를 사용하기 전에 TSM은 먼저 TSM 클라이언트 옵션 파일에 있는 INCLUDE-EXCLUDE 목록에 지정된 관리 클래스로 백업 오브젝트를 바인드하려고 시도합니다. 일치하는 클래스를 찾을 수 없는 경우 TSM 서버에서 지정된 디폴트 TSM 관리 클래스가 사용됩니다. 그런 다음 TSM은 데이터베이스 구성 매개변수에 지정된 관리 클래스로 백업 오브젝트를 리바인드합니다.

따라서 디폴트 관리 클래스 및 데이터베이스 구성 매개변수에 지정된 관리 클래스에는 백업 사본 그룹이 포함되어야 합니다. 그렇지 않으면 백업 조작이 실패합니다.

tsm_nodename - Tivoli Storage Manager 노드 이름

이 매개변수는 Tivoli Storage Manager(TSM) 제품에 연결된 노드 이름의 디폴트 설정을 겹쳐쓰는 데 사용됩니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
명령문 경계

디폴트 [범위]
Null [모든 문자열]

노드 이름은 다른 노드에서 TSM에 백업한 데이터베이스를 리스토어하는 데 필요합니다.

디폴트는 백업을 수행한 것과 같은 노드에 있는 TSM에서만 데이터베이스를 리스토어할 수 있습니다. DB2에 대해 백업을 수행하는 동안(예: BACKUP DATABASE 명령을 사용하여) *tsm_nodename*을 겹쳐쓸 수 있습니다.

tsm_owner - Tivoli Storage Manager 소유자 이름

이 매개변수는 Tivoli Storage Manager(TSM) 제품에 연결된 소유자의 디폴트 설정을 겹쳐쓰는 데 사용됩니다.

구성 유형
데이터베이스

매개변수 유형
온라인으로 구성 가능

전파 클래스
명령문 경계

디폴트 [범위]
Null [모든 문자열]

소유자 이름은 다른 노드에서 TSM에 백업한 데이터베이스를 리스토어하는 데 필요합니다. DB2에 대해 백업을 수행하는 동안(예: BACKUP DATABASE 명령을 사용하여) *tsm_owner*를 겹쳐쓸 수 있습니다.

주: 소유자 이름은 대소문자를 구분합니다.

디폴트는 백업을 수행한 것과 같은 노드에 있는 TSM에서만 데이터베이스를 리스토어할 수 있습니다.

tsm_password - Tivoli Storage Manager 암호

이 매개변수는 Tivoli Storage Manager(TSM) 제품에 연결된 암호의 디폴트 설정을 겹쳐쓰는 데 사용됩니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

명령문 경계

디폴트 [범위]

Null [모든 문자열]

암호는 다른 노드에서 TSM에 백업한 데이터베이스를 리스토어하는 데 필요합니다.

주: DB2에 대해 백업을 수행하는 동안(예: BACKUP DATABASE 명령으로) *tsm_nodename*을 겹쳐쓴 경우, *tsm_password*도 설정해야 합니다.

디폴트는 백업을 수행한 것과 같은 노드에 있는 TSM에서만 데이터베이스를 리스토어할 수 있습니다. DB2에 대해 백업을 수행하는 동안 *tsm_nodename*을 겹쳐쓸 수 있습니다.

user_exit_status - User Exit 상태 표시기

On으로 설정된 경우 이 매개변수는 데이터베이스 관리 프로그램을 롤 포워드 복구에 사용할 수 있으며 User Exit 프로그램을 사용하여 데이터베이스 관리 프로그램이 호출 시 로그 파일을 아카이브 및 검색함을 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

userexit - User Exit 사용

이 매개변수는 버전 9.5에서 사용되지 않지만 버전 9.5 이전의 데이터 서버 및 클라이언트에서 계속 사용됩니다. 이 구성 매개변수에 지정한 값은 DB2 버전 9.5 데이터베이스 관리 프로그램에서는 사용되지 않습니다.

주: 다음의 정보는 버전 9.5 이전의 데이터 서버 및 클라이언트에만 적용됩니다.

이 매개변수를 사용하는 경우 *logretain* 매개변수의 설정에 관계없이 로그 유지 로깅이 수행됩니다. 또한 이 매개변수는 User Exit 프로그램을 사용하여 로그 파일을 아카이브하고 검색해야 함을 표시합니다.

구성 유형

데이터베이스

매개변수 유형

구성 가능

디폴트 [범위]

Off [On; Off]

로그 파일이 가득 차면 로그 파일을 아카이브합니다. ROLLFORWARD 유틸리티가 로그 파일을 사용하여 데이터베이스를 리스토어해야 하는 경우 로그 파일을 검색합니다.

*logretain*이나 *userexit* 또는 이러한 매개변수를 모두 사용하도록 설정한 후 데이터베이스의 전체 백업을 작성해야 합니다. 이 상태는 *backup_pending* 플래그 매개변수로 표시됩니다.

이러한 두 매개변수를 선택 취소한 경우 로그를 더 이상 유지하지 않으므로 데이터베이스의 롤 포워드 복구가 사용 불가능해집니다. 이 경우 데이터베이스 관리 프로그램은 *logpath* 디렉토리에서 모든 로그 파일(온라인 아카이브 로그 파일 포함)을 삭제하고 새 활성 로그 파일을 할당하고 순환 로깅으로 되돌립니다.

util_heap_sz - 유틸리티 힙 크기

이 매개변수는 BACKUP, RESTORE 및 LOAD(로드 복구 포함) 유틸리티가 동시에 사용할 수 있는 최대 메모리 양을 표시합니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

5000 [16 - 524 288]

수치 단위

페이지(4KB)

할당 시기

데이터베이스 관리 프로그램 유틸리티에 필요할 때

사용 가능한 시기

유틸리티에 메모리가 더 이상 필요하지 않을 때

권장사항: 유틸리티가 스페이스를 모두 사용하지 않는 한, 디폴트값을 사용하십시오(모두 사용하는 경우 이 값을 늘려야 함). 시스템의 메모리가 제한된 경우 이 매개변수의 값을 줄여서 데이터베이스 유틸리티가 사용하는 메모리를 제한할 수 있습니다. 매개변수가 너무 낮게 설정된 경우 유틸리티를 동시에 실행할 수 없습니다. 필요하면 이 매개

변수를 동적으로 갱신해야 합니다. 유틸리티가 적은 경우 이 매개변수를 작은 값으로 설정하십시오. 유틸리티가 많거나 메모리 집중 유틸리티의 경우 이 매개변수를 큰 값으로 설정해야 합니다.

varchar2_compat - varchar2 호환성 데이터베이스 구성 매개변수

이 매개변수는 VARCHAR2 데이터 유형에 연결된 호환성 시맨틱이 연결된 데이터베이스에 적용되는지 여부를 표시합니다.

구성 유형

데이터베이스

매개변수 유형

정보용

이 값은 데이터베이스를 작성할 때 VARCHAR2 지원의 DB2_COMPATIBILITY_VECTOR 레지스트리 변수 설정에 따라 결정됩니다. 이 값은 변경할 수 없습니다.

vendoropt - 벤더 옵션

이 매개변수는 DB2가 사본의 백업, 리스토어 또는 로드 조작 중에 스토리지 시스템과 통신하는 데 사용해야 하는 추가적인 매개변수를 지정합니다.

구성 유형

데이터베이스

적용 대상

- 로컬 및 리모트 클라이언트가 있는 데이터베이스 서버
- 클라이언트
- 로컬 클라이언트가 있는 데이터베이스 서버
- 로컬 및 리모트 클라이언트가 있는 파티션된 데이터베이스 서버

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

Null [ ]

제한 **vendoropt** 구성 매개변수를 사용하여 스냅샷 백업 또는 리스토어 조작을 위한 벤더 특정 옵션을 지정할 수는 없습니다. 대신 백업 또는 리스토어 유틸리티의 OPTIONS 매개변수를 사용해야 합니다.

wlm_collect_int - 워크로드 관리 수집 간격 구성 매개변수

이 매개변수는 워크로드 관리(WLM) 통계의 수집 및 재설정 간격을 분 단위로 지정합니다.

x *wlm_collect_int*분마다(여기서 x는 *wlm_collect_int* 매개변수의 값) 모든 워크로드 관리 통계가 수집되어 활성 통계 이벤트 모니터로 전송됩니다. 그런 다음 통계가 재설정됩니다. 활성 이벤트 모니터가 있는 경우, 작성된 방식에 따라 통계가 파일 또는 테이블에 기록됩니다. 활성 이벤트 모니터가 없는 경우, 통계는 재설정만 되고 수집되지 않습니다.

수집 및 재설정 프로세스는 카탈로그 파티션에서 시작됩니다. *wlm_collect_int* 매개변수는 카탈로그 파티션에서 지정해야 합니다. 다른 파티션에서는 사용되지 않습니다.

구성 유형

데이터베이스

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

0 [0(수집이 수행되지 않음), 5 - 32 767]

통계 이벤트 모니터가 수집한 워크로드 관리 통계는 단기 및 장기 시스템 동작을 둘 다 모니터하는 데 사용될 수 있습니다. 결과를 함께 병합하여 장기 동작을 구할 수 있으므로, 단기 및 장기 시스템 동작을 수집하는 데는 둘 다 작은 간격을 사용할 수 있습니다. 그러나 여러 간격의 결과를 수동으로 병합해야 하는 경우 분석이 복잡해집니다. 꼭 필요하지 않은 경우 간격을 작게 하면 오버헤드가 불필요하게 늘어납니다. 그러므로 더 짧은 기간의 동작을 캡처하려면 간격을 줄이고, 장기 동작 분석만으로 충분한 경우 오버헤드를 줄이려면 간격을 늘리십시오.

간격은 각 SQL 요청, 명령 호출 또는 응용프로그램에 대해서가 아니라 데이터베이스마다 사용자 정의되어야 합니다. 고려해야 하는 다른 구성 매개변수는 없습니다.

주: 모든 WLM 통계 테이블 함수는 통계가 마지막으로 재설정된 후 누적된 통계를 리턴합니다. 통계는 이 구성 매개변수가 지정하는 간격마다 정기적으로 재설정됩니다.

DB2 Administration Server(DAS) 구성 매개변수

authentication - 인증 유형 DAS

이 매개변수는 사용자 인증이 발생하는 방법 및 위치를 판별합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

구성 가능

디폴트 [범위]

SERVER_ENCRYPT [SERVER_ENCRYPT; KERBEROS_ENCRYPT]

authentication이 SERVER_ENCRYPT인 경우, 사용자 ID 및 암호는 클라이언트에서 서버로 전송되어 서버에서 인증이 발생합니다. 네트워크를 통해 전송되는 사용자 ID와 암호는 암호화됩니다.

KERBEROS_ENCRYPT 값은 Kerberos 서버에서 인증용 Kerberos 보안 프로토콜을 사용하여 인증이 수행되었음을 의미합니다.

주: KERBEROS_ENCRYPT 인증 유형은 Windows를 실행 중인 서버에서만 지원됩니다.

이 매개변수는 버전 9 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

contact_host - 문의처 목록의 위치

이 매개변수는 스케줄러 및 Health Monitor가 통지에 사용할 문의처 정보가 저장된 위치를 지정합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

Null [유효한 DB2 Administration Server TCP/IP 호스트 이름]

위치는 DB2 Administration Server의 TCP/IP 호스트 이름이 되도록 정의됩니다. *contact_host*가 리모트 DAS에 있도록 허용하면 다중 DB2 Administration Server에서 문의처 목록을 공유할 수 있습니다. *contact_host*가 지정되지 않으면 DAS는 연락 정보를 로컬로 가정합니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

das_codepage - DAS 코드 페이지

이 매개변수는 DB2 Administration Server가 사용하는 코드 페이지를 표시합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

Null [유효한 DB2 코드 페이지]

매개변수가 널(NULL)인 경우 시스템의 디폴트 코드 페이지가 사용됩니다. 이 매개변수는 로컬 DB2 인스턴스의 로케일과 호환 가능해야 합니다. 그렇지 않으면 DB2 Administration Server가 DB2 인스턴스와 통신할 수 없습니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

das_territory - DAS 지역

이 매개변수는 DB2 Administration Server가 사용하는 지역을 표시합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

Null [유효한 DB2 지역]

이 매개변수가 널(NULL)인 경우, 시스템의 디폴트 지역이 사용됩니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

dasadm_group - DAS 관리 권한 그룹 이름

이 매개변수는 DAS의 DAS 관리(DASADM) 권한이 있는 그룹 이름을 정의합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 그룹 이름]

DASADM 권한은 DAS에서 가장 높은 레벨의 권한입니다.

DASADM 권한은 특정 운영 환경에서 사용되는 보안 기능으로 판별합니다.

- Windows 운영 체제의 경우 이 매개변수는 Windows 보안 데이터베이스에 정의된 로컬 그룹으로 설정될 수 있습니다. 승인되는 그룹 이름 길이는 30바이트 이하입니다. 이 매개변수에 『NULL』이 지정된 경우 Administrators 그룹의 모든 구성원은 DASADM 권한이 있습니다.
- Linux 및 UNIX 시스템의 경우 이 매개변수의 값으로 『NULL』이 지정되면 DASADM 그룹 디폴트값은 인스턴스 소유자의 기본 그룹이 됩니다.

  값이 『NULL』이 아니면 DASADM 그룹은 유효한 모든 UNIX 그룹 이름이 될 수 있습니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

db2system - DB2 서버 시스템 이름

이 매개변수는 사용자 및 데이터베이스 관리자가 DB2 서버 시스템을 식별하는 데 사용할 이름을 지정합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

온라인으로 구성 가능

디폴트 [범위]

TCP/IP 호스트 이름 [유효한 시스템 이름]

가능한 경우, 이 이름은 네트워크 내에서 고유해야 합니다.

이 이름은 제어 센터 오브젝트 트리의 시스템 레벨에 표시되어 관리자가 제어 센터에서 관리할 수 있는 서버 시스템을 식별하도록 도와줍니다.

구성 지원 프로그램의 '네트워크 검색' 기능을 사용하는 경우, DB2 발견은 이 이름을 리턴하며 결과 오브젝트 트리의 시스템 레벨에 이 이름이 표시됩니다. 이 이름은 사용자가 액세스할 데이터베이스가 있는 시스템을 식별하도록 도와줍니다. *db2system*의 값은 설치 시 다음과 같이 설정됩니다.

- Windows의 경우 설치 프로그램은 이 값을 Windows 시스템에 지정된 컴퓨터 이름과 동일하게 설정합니다.
- UNIX 시스템의 경우 이 값은 UNIX 시스템의 TCP/IP 호스트 이름과 동일하게 설정됩니다.

discover - DAS 발견 모드

이 매개변수는 DB2 Administration Server를 시작할 때 시작되는 발견 모드 유형을 판별합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

SEARCH [DISABLE; KNOWN; SEARCH]

- discover = SEARCH이면 관리 서버는 클라이언트의 SEARCH 발견 요청을 처리합니다. SEARCH는 KNOWN 발견이 제공하는 기능의 수퍼 세트를 제공합니다. discover = SEARCH이면 관리 서버는 클라이언트의 SEARCH 및 KNOWN 발견 요청을 둘 다 처리합니다.
- discover = KNOWN이면 관리 서버는 클라이언트의 KNOWN 발견 요청만 처리합니다.
- discover = DISABLE이면 관리 서버는 어떤 유형의 발견 요청도 처리하지 않습니다. 이 서버 시스템에 대한 정보는 기본적으로 클라이언트가 볼 수 없습니다.

디폴트 발견 모드는 SEARCH입니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

exec_exp_task - 만기된 태스크 실행

이 매개변수는 스케줄러가 과거에 스케줄되었으나 아직 실행되지 않은 태스크를 실행할 것인지 여부를 지정합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

구성 가능

디폴트 [범위]

No [Yes; No]

스케줄러는 시작할 때만 만기된 작업을 발견합니다. 예를 들어, 매주 토요일에 실행하도록 스케줄된 작업이 있고 스케줄러를 금요일에 껐다가 월요일에 재시작한 경우, 토요일에 스케줄된 작업은 이제 과거에 스케줄된 작업입니다. *exec_exp_task*가 Yes로 설정된 경우에는 스케줄러를 재시작할 때 토요일 작업이 실행됩니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

jdk_path - Java용 SDK(Software Developer's Kit) 설치 경로 DAS

이 매개변수는 DB2 Administration Server 기능을 실행하는 데 사용할 Java용 SDK(Software Developer's Kit)가 설치된 디렉토리를 지정합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

디폴트 Java 설치 경로 [유효한 경로]

Java 인터프리터에서 사용되는 환경 변수는 이 매개변수의 값에서 계산됩니다.

Windows 운영 체제의 경우, Java 파일(필요한 경우)은 DB2 설치 시 sqllib 디렉토리(java₩jdk)에 있습니다. 이 때 *jdk_path* 구성 매개변수는 sqllib₩java₩jdk로 설정됩니다. Windows 플랫폼의 DB2는 실제로 Java를 설치하지 않으며, 파일은 단순히 sqllib 디렉토리에 놓이고, 이는 Java가 이미 설치되었는지 여부에 관계 없이 수행됩니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

sched_enable - 스케줄러 모드

이 매개변수는 관리 서버가 스케줄러를 시작했는지 여부를 표시합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

구성 가능

디폴트 [범위]

Off [On; Off]

스케줄러를 사용하면 태스크 센터와 같은 도구가 관리 서버에서 태스크를 스케줄하고 실행할 수 있습니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

sched_userid - 스케줄러 사용자 ID

이 매개변수는 스케줄러가 도구 카탈로그 데이터베이스에 연결하는 데 사용하는 사용자 ID를 지정합니다. 이 매개변수는 도구 카탈로그 데이터베이스가 DB2 Administration Server에 대해 리모트인 경우에만 관련이 있습니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

정보용

디폴트 [범위]

Null [유효한 사용자 ID]

스케줄러가 리모트 도구 카탈로그 데이터베이스에 연결하는 데 사용하는 사용자 ID와 암호는 db2admin 명령을 사용하여 지정합니다.

smtp_server - SMTP 서버

스케줄러가 작동 중인 경우, 이 매개변수는 스케줄러가 전자 우편 및 호출기 통지를 보내는 데 사용할 SMTP 서버를 식별합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

온라인으로 구성 가능

전파 클래스

즉시

디폴트 [범위]

Null [유효한 SMTP 서버 TCP/IP 호스트 이름]

이 매개변수는 스케줄러 및 Health Monitor에서 사용됩니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

toolscat_db - 도구 카탈로그 데이터베이스

이 매개변수는 스케줄러에서 사용되는 도구 카탈로그 데이터베이스를 표시합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 데이터베이스 별명]

이 데이터베이스는 *toolscat_inst*가 지정하는 인스턴스의 데이터베이스 디렉토리에 있어야 합니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

toolscat_inst - 도구 카탈로그 데이터베이스 인스턴스

이 매개변수는 스케줄러가 사용하는 인스턴스 이름과 도구 카탈로그 데이터베이스를 식별하기 위한 toolscat_db 및 toolscat_schema를 표시합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 인스턴스]

도구 카탈로그 데이터베이스에는 태스크 센터 및 제어 센터가 작성한 태스크 정보가 있습니다. 도구 카탈로그 데이터베이스는 이 구성 매개변수가 지정하는 인스턴스의 데이

터베이스 디렉토리에 나열되어야 합니다. 데이터베이스는 로컬 또는 리모트가 될 수 있습니다. 도구 카탈로그 데이터베이스가 로컬인 경우, 인스턴스는 TCP/IP에 대해 구성되어야 합니다. 데이터베이스가 리모트인 경우 데이터베이스 디렉토리에 카탈로그된 데이터베이스 파티션은 TCP/IP 노드여야 합니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

toolscat_schema - 도구 카탈로그 데이터베이스 스키마

이 매개변수는 스케줄러에서 사용되는 도구 카탈로그 데이터베이스의 스키마를 표시합니다.

구성 유형

DB2 Administration Server

적용 대상

DB2 Administration Server

매개변수 유형

구성 가능

디폴트 [범위]

Null [유효한 스키마]

스키마는 데이터베이스 내의 도구 카탈로그 테이블 및 뷰 세트를 고유하게 식별하는 데 사용됩니다.

이 매개변수는 버전 8 명령행 처리기(CLP)에서만 갱신할 수 있습니다.

제 5 부 부록

색인

[가]

[나]

[다]

[라]

[마]

[바]

[사]

[아]

[자]

[차]

[카]

[타]

[파]

[하]

[숫자]

A

B

C

D

E

F

G

H

I

J

K

L

M

T

U

V

W

X

SA30-3951-00

Spine information:

Linux, UNIX 및 Windows용 IBM DB2 9.7

데이터베이스 관리 개념 및 구성 참조서