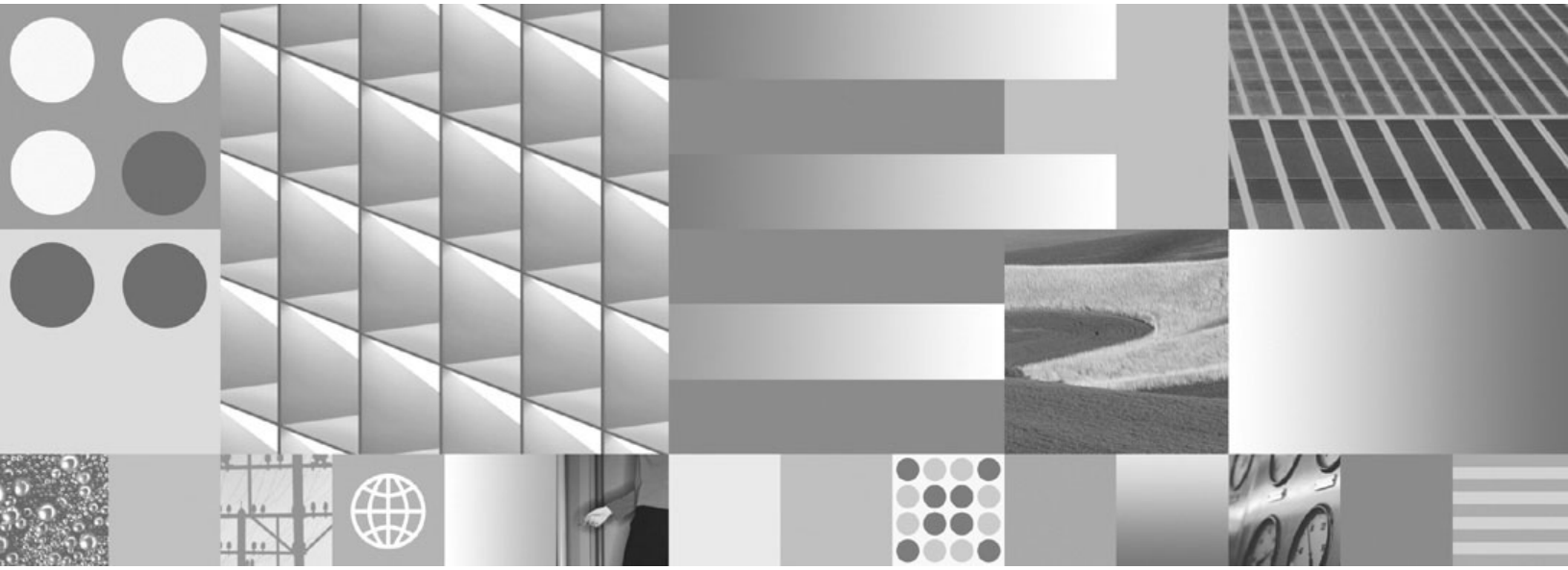
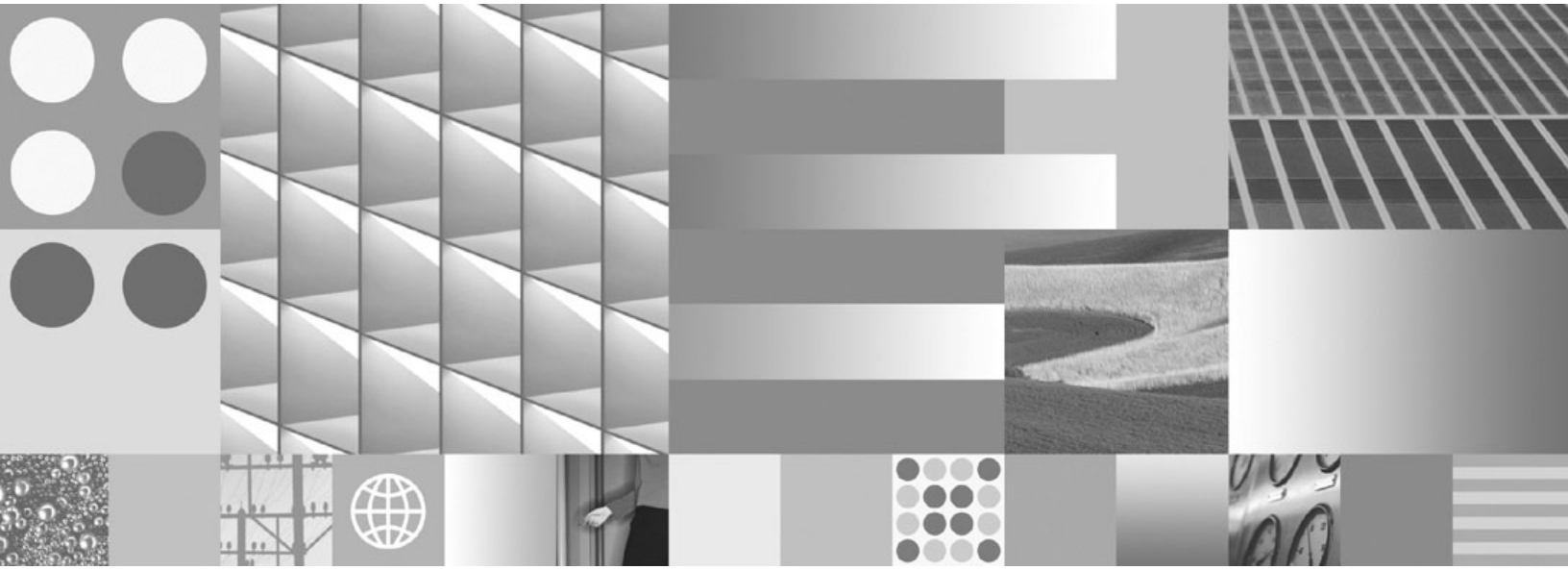
Linux, UNIX 및 Windows용
IBM DB2 9.7



문제점 해결 및 데이터베이스 성능 조정

SA30-3952-00

Linux, UNIX 및 Windows용
IBM DB2 9.7



문제점 해결 및 데이터베이스 성능 조정

SA30-3952-00

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 631 페이지의 부록 B『주의사항』의 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

이 안내서에는 DB2® 데이터베이스 클라이언트 및 서버의 데이터베이스 성능 조정 및 문제점 해결에 대한 정보가 있습니다.

이 책에 대한 정보를 사용하여 다음을 수행할 수 있습니다.

- 성능 모니터링 및 조정 전략 개발
- 매일의 조작에서 발생하는 문제점 해결 전략 개발
- 데이터베이스 서버의 구성 조정
- 데이터베이스 서버를 사용하는 응용프로그램 변경
- 간단한 방법으로 문제점 및 오류 식별
- 증상에 기반한 문제점 해결
- 사용 가능한 도구 학습

이 책의 사용자

이 안내서는 고객, 사용자, 시스템 관리자, 데이터베이스 관리자(DBA), 통신 전문가, 응용프로그램 개발자 및 DB2 데이터베이스 클라이언트 및 서버의 데이터베이스 성능 및 문제점 해결에 관심이 있는 기술 지원 영업대표를 위해 작성되었습니다. 이 안내서를 사용하려면 다음을 숙지하십시오.

- 통신, 관련 데이터베이스 및 LAN(Local Area Network) 개념
- 하드웨어 및 소프트웨어 요구사항 및 옵션
- 네트워크 전체 구성
- 네트워크에서 실행되는 응용프로그램 및 기타 기능
- 기본 DB2 데이터베이스 관리 태스크
- 설치한 제품의 *빠른 시작* 안내서에 설명된 설치 정보 및 초기 태스크

이 책의 구성

성능 모니터링 및 데이터베이스 시스템 조정에 도움을 주기 위해 이 책에서 제공된 정보는 데이터베이스 성능에 영향을 주는 요소를 이해하기 위한 필수 백그라운드 및 시스템 성능 조정에 대한 지시사항으로 구성되어 있습니다. DB2 소프트웨어 문제점을 이해, 분리 및 해결할 수 있도록 문제점 해결 및 지원 정보에는 DB2 제품과 함께 제공되는 문제점 판별 자원 사용 지시사항이 포함되어 있습니다.

제 1 부. 데이터베이스 성능 조정

데이터베이스 관리자는 데이터베이스 응용프로그램이 느리게 실행된다는 사용자의 보고서를 받는 경우가 있습니다. 이 책에 제공된 정보는 이전의 결과와 비교하여 데이터베이스 시스템 성능을 객관적으로 평가할 수 있도록 성능 모니터링 전략을 개발하는 방법, 데이터베이스 서버의 구성 조정 방법 및 데이터베이스 서버를 사용하는 응용프로그램 변경 방법에 대해 설명하며 이러한 모든 방법은 처리 비용의 증가 및 사용자에 대한 서비스 품질 저하 없이 데이터베이스 시스템 성능을 향상시키는 것을 목표로 합니다.

- 제 1 장 『성능 조정 도구 및 방법론』에서는 성능을 향상시킬 수 있도록 벤치마크 테스팅 프로그램을 설계 및 구현하는 방법에 대해 설명합니다.
- 제 2 장 『성능 모니터링 도구 및 방법론』에서는 정기적으로 키 시스템 성능 데이터를 수집하는 운영 모니터링 전략의 중요성에 대한 정보를 제공합니다.
- 제 3 장 『성능에 영향을 미치는 요소』에는 데이터베이스 시스템 성능에 영향을 줄 수 있는 여러 가지 인수에 대한 정보가 있습니다. 이 인수 중 일부는 조정 또는 재구성이 가능합니다.
- 제 4 장 『성능 조정 전략 설정』에서는 워크로드 성능을 상당히 향상시킬 수 있는 DB2 디자인 어드바이저 도구에 대해 설명합니다.

제 2 부. 문제점 해결

스스로 문제점을 해결하도록 돕기 위해 문제점의 원인을 식별하는 방법, 진단 정보 수집 방법, 수정사항을 얻는 위치 및 추가 정보를 검색할 지식 기반에 대한 정보가 포함되어 있습니다. IBM Software Support에 문의해야 하는 경우, 지원 문의 방법 및 문제점 해결을 돕기 위해 서비스 기술자가 필요로 하는 진단 정보에 대한 설명이 있습니다.

- 제 5 장 『문제점 해결 도구』에서는 체계적인 접근방법으로 문제점 해결에 도움을 줄 수 있는 문제점 해결 도구에 대해 설명합니다. 목적은 무언가가 예측한 대로 작동하지 않는 이유와 문제점 해결 방법을 판별하는 것입니다.
- 제 6 장 『DB2 데이터베이스 문제점 해결』에는 발생할 수 있는 알려진 여러 가지 문제점 및 문제점 해결 방법에 대한 정보가 있습니다.
- 제 7 장 『DB2® Connect™』에는 발생할 수 있는 알려진 여러 가지 문제점 및 문제점 해결 방법에 대한 정보가 있습니다.
- 제 8 장 『지식 기반 검색』에는 IBM 지식 기반을 검색하여 문제점에 대한 솔루션을 찾을 수 있는 방법에 대한 정보가 있습니다. 이 장에서는 사용 가능한 자원, 지원 도구 및 검색 방법을 사용하여 결과를 최적화하는 방법을 설명합니다.
- 제 9 장 『DB2 제품 수정사항 얻기』에는 문제점 해결에 이미 사용할 수 있는 제품 수정사항 얻기에 대한 정보가 있습니다. 여기에 설명된 단계를 수행하여 수정사항을 얻을 수 있습니다.

- 제 10 부 『문제점 해결에 대한 추가 정보』에서는 DB2 데이터베이스 서버의 문제점을 효과적으로 해결하는데 필요한 개념 정보 획득에 도움을 주는 다음과 같은 주제에 대해 설명합니다.
- 제 11 장 『IBM Software Support에 문의』에서는 IBM Software Support에 문의하는 방법과 제품 결함 및 데이터베이스 문제점 해결에 도움을 주기 위해 필요한 정보에 대해 설명합니다.

제 3 부. 부록

- 부록 A, 『DB2 기술 정보 개요』
- 부록 B, 『주의사항』

제 1 부 성능 개요

성능은 특정 워크로드에 대해 컴퓨터 시스템이 작동하는 방식을 나타냅니다. 성능은 시스템 응답 시간, 처리량 및 자원 활용 면에서 측정합니다.

성능은 다음의 영향도 받습니다.

- 시스템에서 사용 가능한 자원
- 이러한 자원을 얼마나 잘 사용하고 공유하는지 여부

일반적으로 비용 편익 비율을 향상시키도록 시스템을 조정하게 됩니다. 특정 목표에는 다음이 포함될 수 있습니다.

- 처리 비용을 늘리지 않고 크거나 요구가 많은 워크로드 처리
- 처리 비용을 늘리지 않고 신속한 시스템 응답 시간 또는 높은 처리량 확보
- 사용자에 대한 서비스를 줄이지 않고 처리 비용 감축

자원의 효율적인 사용과 같은 성능 조정의 일부 장점과 시스템에 사용자를 추가하는 기능은 실제로 확인할 수 있습니다. 빠른 응답 시간으로 인한 사용자의 만족도 증가와 같은 기타 장점은 실제로 파악하기 어렵습니다.

성능 조정 지침

성능 조정에 대한 전체적인 접근 방법을 개발할 때 다음의 지침을 잊지 마십시오.

- **수익 체감의 법칙 기억**: 가장 큰 성능 이점은 보통 초기 노력을 통해 얻을 수 있습니다.
- **조정만을 위해 조정하지 않음**: 식별된 제한조건을 완화할 경우에 조정하십시오. 성능 문제의 기본 원인이 아닌 자원을 조정하면 이후 조정 작업이 실제로 훨씬 어려워질 수 있습니다.
- **전체 시스템 고려**: 분리 상태에서 하나의 매개변수 또는 자원을 조정할 수 없습니다. 조정하기 전에 변경으로 인해 시스템 전체에 미치는 영향을 고려하십시오. 성능 조정은 다양한 시스템 자원 간 트래이드 오프가 필요합니다. 예를 들어, 입출력 성능을 향상시키기 위해 버퍼 풀 크기를 증가시킬 수 있지만 버퍼 풀이 크면 추가 메모리가 필요하고 이로 인해 성능의 다른 측면이 저하될 수 있습니다.
- **한 번에 하나의 매개변수 변경**: 한 번에 하나의 인수만 변경하십시오. 모든 변경이 이로울 것으로 확신하는 경우에도 각 변경의 기여도를 평가할 수 있는 방법은 없습니다.
- **레벨에 따른 측정 및 구성**: 한 번에 하나의 시스템 레벨만 조정하십시오. 시스템 레벨은 다음과 같습니다.
  - 하드웨어

  - 운영 체제
  - 응용프로그램 서버 및 리퀘스터
  - 데이터베이스 관리 프로그램
  - SQL 및 XQuery문
  - 응용프로그램
- **하드웨어 및 소프트웨어 문제점 점검**: 일부 성능 문제는 하드웨어, 소프트웨어 또는 둘 다에 서비스를 적용하여 정정할 수 있습니다. 하드웨어 또는 소프트웨어에 서비스를 적용하기 전에 시스템을 모니터링하고 조정하는 데 너무 많은 시간을 사용하지 마십시오.
- **하드웨어 업그레이드 전 문제점 이해**: 추가 스토리지 또는 프로세서 능력이 성능을 즉시 향상시킬 수 있다고 판단된 경우에도 시간을 들여서 병목 현상이 있는 위치를 파악하십시오. 처리 능력 또는 이를 이용하는 채널이 없음을 발견하기 위해서만 추가 디스크 스토리지에 비용을 많이 들 수 있습니다.
- **조정 시작 전 폴백(fallback) 프로시저를 올바른 위치에 배치**: 조정으로 인해 예기치 않은 성능 저하가 발생한 경우 대체 방법을 시도하기 전에 변경한 사항을 되돌려야 합니다. 보존하지 않을 변경사항을 쉽게 실행 취소할 수 있도록 원래의 설정을 저장하십시오.

성능 향상 프로세스 개발

성능 향상 프로세스는 성능의 모니터링 및 조정 측면에 대한 반복적인 접근 방식입니다. 이 성능 모니터링의 결과에 따라 데이터베이스 서버의 구성을 조정하고 데이터베이스 서버를 사용하는 응용프로그램을 변경합니다.

데이터를 사용하는 응용프로그램 유형 및 데이터 액세스 패턴에 대한 지식에 따라 성능 모니터링 및 조정 결정을 내리십시오. 응용프로그램 유형이 다르면 성능 요구사항이 다릅니다.

성능 향상 프로세스에는 다음과 같은 기본 단계가 포함됩니다.

1. 성능 목표 정의
2. 시스템의 주요 제한조건에 대한 성능 표시기 설정
3. 성능 모니터링 계획 개발 및 실행
4. 모니터링 결과의 계속적인 분석을 통해 조정이 필요한 자원 판별
5. 한 번에 하나의 조정 수행

일정한 시점에 데이터베이스 서버 또는 응용프로그램을 조정하여 성능을 더 이상 향상시킬 수 없는 경우 하드웨어를 업그레이드할 시기입니다.

사용자가 제공할 수 있는 성능 정보

시스템에 조정이 필요한 첫 번째 신호는 사용자의 불만일 수 있습니다. 성능 목표를 설정하고 종합적인 방식으로 모니터하고 조정할 수 있는 시간이 충분하지 않은 경우 사용자의 불만사항을 듣고 성능 문제를 해결할 수 있습니다. 다음과 같은 몇 가지 간단한 질문을 하는 것으로 시작하십시오.

- 『느린 응답』은 정확히 무슨 뜻입니까? 예상보다 10% 정도 느리거나 수십 배 느립니까?
- 언제 문제가 발견되었습니까? 최근 또는 계속해서 발생했습니까?
- 다른 사용자가 동일한 문제를 경험했습니까? 이러한 사용자는 한두 명의 개인이거나 전체 그룹입니까?
- 사용자 그룹이 동일한 문제를 경험한 경우 이러한 사용자들이 동일한 근거리 통신망에 연결되어 있습니까?
- 문제가 특정 유형의 트랜잭션 또는 응용프로그램과 관련이 있는 것 같습니까?
- 발생에 어떤 패턴이 있습니까? 예를 들어, 문제가 하루 중 특정 시간에 발생하거나 연속적으로 발생합니까?

성능 조정 한계

성능 조정의 이점은 한정되어 있습니다. 시스템 성능을 향상시키는 데 필요한 시간과 비용을 고려할 때 추가 시간 및 비용 투자가 시스템의 사용자에게 도움이 되는 수준을 평가해야 합니다.

시스템에 응답 시간 또는 처리량 문제가 발생한 경우 주로 조정으로 성능이 향상될 수 있습니다. 그러나 추가 조정만으로 도움이 되지 않는 경우가 있습니다. 이 경우 목표 및 예상을 수정하십시오. 성능을 상당히 높게 향상시키려면 추가 디스크 스토리지, 속도가 빠른 CPU, 추가 CPU, 추가 주기억장치, 속도가 빠른 통신 링크 또는 이들의 조합을 추가해야 합니다.

제 1 장 성능 조정 도구 및 방법론

벤치마크 테스팅

벤치마크 테스팅은 응용프로그램 개발 주기 중 발생하는 일반 과정입니다. 응용프로그램 개발자와 데이터베이스 관리자(DBA)가 모두 참여하는 팀 작업입니다.

시스템에 대해 벤치마크 테스팅을 수행하여 현재 성능을 판별하고 응용프로그램 성능을 향상시키는 데 사용할 수 있습니다. 응용프로그램 코드가 비교적 효율적으로 기록된 경우 데이터베이스 및 데이터베이스 관리 프로그램 구성 매개변수를 조정하여 추가로 성능을 향상시킬 수 있습니다.

여러 가지 유형의 벤치마크 테스트를 사용하여 특정 유형의 정보를 찾습니다. 예를 들어, 다음과 같습니다.

- *인프라스트럭처 벤치마크*는 제한된 특정 연구 조건에서 데이터베이스 관리 프로그램의 처리량 성능을 판별합니다.
- *응용프로그램 벤치마크*는 프로덕션 환경을 보다 근접하게 반영한 조건에서 데이터베이스 관리 프로그램의 처리 성능을 판별합니다.

구성 매개변수를 조정하기 위한 벤치마크 테스팅은 제어 조건을 기초로 합니다. 이와 같은 테스팅에는 응용프로그램이 가능한 한 효율적으로 실행될 때까지 시스템 구성(및 SQL)을 변경하면서 응용프로그램에서 SQL을 반복적으로 실행하는 작업이 포함됩니다.

동일한 접근 방법을 사용하여 성능에 영향을 주는 기타 요인(예: 인덱스, 테이블 스페이스 구성 및 하드웨어 구성)을 조정할 수 있습니다.

벤치마크 테스트는 데이터베이스 관리 프로그램이 다른 조건에 응답하는 방식을 이해하는 데 도움이 됩니다. 교착 상태 처리, 유틸리티 성능, 여러 가지 데이터 로드 방법, 사용자 추가에 따른 트랜잭션 비율 특성 및 데이터베이스 제품의 새 릴리스 사용이 응용프로그램에 미치는 영향을 테스트하는 시나리오를 작성할 수 있습니다.

벤치마크 테스트는 반복 가능한 환경에 근거하므로 동일한 조건에서 동일한 테스트를 실행할 경우 발생하는 결과를 정당하게 비교할 수 있습니다. 정상적인 환경에서 테스트 응용프로그램을 실행하여 시작할 수 있습니다. 성능 문제점에 대해 자세하게 조사하면 테스트 중인 기능의 범위를 제한하는 특수 테스트 케이스를 개발할 수 있습니다. 특수 테스트 케이스는 중요한 정보를 얻기 위해 전체 응용프로그램을 에뮬레이트할 필요가 없습니다. 간단한 측정부터 시작하여 필요한 경우에만 복잡도를 높이십시오.

좋은 벤치마크의 특성은 다음과 같습니다.

- 반복 가능한 테스트

- 동일한 시스템 상태에서 각 테스트 반복이 시작됨
- 시스템에서 기타 기능 또는 응용프로그램이 실수로 활성화되지 않음
- 벤치마크 테스팅에 사용되는 하드웨어 및 소프트웨어가 프로덕션 환경과 일치함

시작된 응용프로그램은 유휴 상태인 경우에도 메모리를 사용한다는 점을 참고하십시오. 이로 인해 페이징이 벤치마크의 결과를 왜곡시킬 가능성이 커지고 반복 가능성 기준을 위반합니다.

벤치마크 준비

성능 벤치마크 테스팅을 시작하려면 먼저 충족시켜야 하는 특정 전제조건이 있습니다.

성능 벤치마크 테스팅을 시작하기 전에 다음을 수행하십시오.

- 응용프로그램을 실행할 데이터베이스의 논리적 및 실제 설계를 모두 완료하십시오.
- 테이블, 뷰 및 인덱스를 작성하십시오.
- 테이블을 표준화하고 응용프로그램 패키지를 바인드한 후 테이블을 실제 데이터로 채우십시오. 적절한 통계가 사용 가능한지 확인하십시오.
- 응용프로그램이 나타내는 메모리 요구사항을 테스트할 수 있도록 프로덕션 *크기*의 데이터베이스에 대해 실행하도록 계획하십시오. 이것이 불가능한 경우에는 테스트와 프로덕션 시스템에서 데이터에 대해 사용 가능한 시스템 자원의 비율이 동일하도록 시도하십시오. (예를 들어, 테스트 시스템에 데이터의 10%가 있는 경우, 프로세서 시간의 10%와 프로덕션 시스템에 사용 가능한 메모리의 10%를 사용하십시오.)
- 최종 디스크 위치에 데이터베이스 오브젝트를 배치하고 로그 파일의 *크기*를 조정하고 작업 파일 및 백업 이미지의 위치를 판별하고 백업 프로시저를 테스트하십시오.
- 가능하면 행 블로킹과 같은 성능 옵션을 사용할 수 있는지 패키지를 점검하십시오.

벤치마크 테스팅 중 응용프로그램의 실제 한계가 나타날 수 있지만 벤치마크는 결함을 감지하기 위한 것이 아니고 성능을 측정하기 위한 것입니다.

벤치마크 테스팅 프로그램은 최종 프로덕션 환경이 정확히 표시되면 실행해야 합니다. 원래 동일한 메모리 및 디스크 구성의 동일한 서버 모델에서 실행해야 합니다. 응용프로그램이 나중에 많은 사용자에게 서비스를 제공하고 대량의 데이터를 처리하는 경우에 특히 이 점이 중요합니다. 벤치마크 테스팅 프로그램에서 직접 사용하는 운영 체제와 통신 또는 스토리지 기능도 이전에 조정되어 있어야 합니다.

벤치마크 테스팅할 SQL문은 다음 설명과 같이 대표 SQL 또는 worst-case SQL이어야 합니다.

대표 SQL

대표 SQL에는 벤치마크 테스팅 중인 응용프로그램의 일반 조작 중 실행되는 명령문이 들어 있습니다. 선택하는 명령문은 응용프로그램의 특성에 따라 다릅

니다. 예를 들어, 데이터 항목 응용프로그램은 INSERT문을 테스트할 수 있지만 은행 거래는 FETCH, UPDATE 및 여러 INSERT문을 테스트할 수 있습니다.

worst-case SQL

이 범주에 속하는 명령문은 다음과 같습니다.

- 자주 실행하는 명령문
- 대량의 데이터를 처리 중인 명령문
- 시간 제약이 있는 명령문. 예를 들어, 고객이 통화 대기 중인 동안 고객 정보를 검색하고 갱신하기 위해 실행되는 응용프로그램의 명령문이 있습니다.
- 조인이 매우 많은 명령문 또는 응용프로그램에서 가장 복잡한 명령문. 예를 들어, 모든 고객 계좌의 월별 활동에 대한 요약을 작성하는 은행 응용프로그램의 명령문이 있습니다. 일반 테이블은 고객의 주소 및 계좌 번호를 나열할 수도 있지만 기타 여러 테이블을 조인하여 필요한 모든 계좌 거래 정보를 처리 및 통합해야 합니다.
- 사용 가능한 인덱스에서 지원하지 않는 경로와 같이 잘못된 액세스 경로가 있는 명령문
- 실행 시간이 긴 명령문
- 응용프로그램 초기화 시에만 실행되지만 자원 요구사항이 균형에 맞지 않게 큰 명령문. 예를 들어, 하루 중 처리해야 하는 계좌 업무 목록을 생성하는 응용프로그램의 명령문이 있습니다. 응용프로그램이 시작되면 첫 번째 주요 SQL문은 이 응용프로그램 사용자가 담당하는 모든 계좌로 구성된 매우 큰 목록을 작성하는 7방향 조인을 생성합니다. 이 명령문은 매일 몇 번만 실행할 수 있지만 올바로 조정되지 않은 경우 실행하려면 몇 분이 걸립니다.

벤치마크 테스트 작성

벤치마크 테스팅 프로그램을 설계 및 구현할 때 다양한 요인을 고려해야 합니다.

테스팅 프로그램의 주요 목적은 사용자 응용프로그램을 가상화하기 위한 것이므로 프로그램의 전체 구조는 다양합니다. 전체 응용프로그램을 벤치마크로 사용하고 분석할 SQL문을 시간 제어하는 방법을 간단하게 도입할 수 있습니다. 대형 또는 복잡한 응용프로그램의 경우 중요한 명령문이 들어 있는 블록만 포함시키는 것이 훨씬 실용적입니다. 특정 SQL문의 성능을 테스트하려면 벤치마크 테스팅 프로그램에 시간 제어 메커니즘 외에 필요한 CONNECT, PREPARE, OPEN 및 기타 명령문과 함께 이러한 명령문만 포함시킬 수 있습니다.

고려해야 하는 다른 요인은 사용할 벤치마크의 유형입니다. 한 가지 옵션은 특정 간격 동안 SQL문 세트를 반복적으로 실행하는 것입니다. 이 간격 동안 실행되는 명령문 수는 응용프로그램의 처리량 수치입니다. 다른 옵션은 개별 SQL문을 실행하는 데 필요한 시간을 간단하게 판별하는 것입니다.

모든 벤치마크 테스팅에서 경과 시간을 측정할 신뢰할 수 있고 적절한 방법이 필요합니다. 개별 SQL문이 분리되어 실행되는 응용프로그램을 시뮬레이트하려면 각 명령문에 대해 PREPARE, EXECUTE, OPEN, FETCH 또는 CLOSE 시간을 측정하는 것이 최선입니다. 다른 응용프로그램의 경우 첫 번째 SQL문에서 COMMIT 문까지의 트랜잭션 시간을 측정하는 것이 더 적절할 수 있습니다.

각 쿼리의 경과 시간은 성능 분석에서 중요한 요인이지만 반드시 병목 현상을 나타내지는 않습니다. 예를 들어, CPU 사용량, 잠금 및 버퍼 풀 입출력에 대한 정보는 응용프로그램이 입출력 위주이며 CPU를 모두 사용하고 있지 않음을 나타낼 수 있습니다. 벤치마크 테스팅 프로그램을 사용하면 필요한 경우 보다 자세한 분석을 위해 이러한 유형의 데이터를 확보할 수 있습니다.

모든 응용프로그램이 쿼리에서 검색한 전체 행 세트를 출력 디바이스로 보내는 것은 아닙니다. 예를 들어, 결과 세트는 다른 응용프로그램의 입력일 수 있습니다. 화면 출력을 위해 데이터를 형식화하려면 CPU 비용이 많이 들고 사용자의 요구가 반영되지 않을 수도 있습니다. 정확하게 가상화하려면 벤치마크 테스팅 프로그램은 응용프로그램의 특정 행 처리 활동을 반영해야 합니다. 출력 디바이스로 행을 보내는 경우 비효율적인 형식화는 대부분의 CPU 시간을 사용할 수 있으며 SQL문 자체의 실제 성능을 잘못 표시할 수 있습니다.

DB2 명령행 처리기(CLP)는 사용하기 매우 쉽지만 추가되는 처리 오버헤드 때문에 벤치마킹에는 알맞지 않습니다. 벤치마크 도구(db2batch)가 인스턴스 `sqllib` 디렉토리의 `bin` 서브디렉토리에 제공됩니다. 이 도구는 플랫 파일 또는 표준 입력에서 SQL문을 읽고 동적으로 명령문을 준비하고 실행한 후 결과 세트를 리턴할 수 있습니다. 또한 db2batch로 리턴되는 행 수 및 표시되는 행 수도 제어할 수 있습니다. 데이터베이스 모니터에서 수집한 경과 시간, 프로세서 시간, 버퍼 풀 사용량, 잠금 및 기타 통계를 포함하여 리턴되는 성능 관련 정보의 레벨을 지정할 수 있습니다. SQL문 세트를 시간 제어하는 경우 db2batch는 성능 결과도 요약하며 산술 및 기하학 방식을 모두 제공합니다.

Perl 또는 콘 쉘 스크립트에서 db2batch 호출을 래핑하여 다중 사용자 환경을 쉽게 가상화할 수 있습니다. 적절한 db2batch 옵션을 선택하여 분리 수준과 같은 연결 속성이 동일한지 확인하십시오.

파티션된 데이터베이스 환경에서, db2batch는 경과 시간 측정에만 적합합니다. 리턴되는 다른 정보는 코디네이터 데이터베이스 파티션에 대한 활동에만 관련됩니다.

벤치마크 테스팅에 도움이 되는 드라이버 프로그램을 작성할 수 있습니다. Linux® 또는 UNIX® 시스템에서 드라이버 프로그램은 쉘 프로그램을 사용하여 작성할 수 있습니다. 드라이버 프로그램은 벤치마크 프로그램을 실행하고, 적합한 매개변수를 패스하고, 여러 번 반복하여 테스트를 실행하고, 환경을 일관적인 상태로 리스토어하고, 새 매개변수 값으로 다음 테스트를 설정하고, 테스트 결과를 수집 및 통합할 수 있습니다. 드

라이버 프로그램은 전체 벤치마크 테스트 세트를 실행하고, 결과를 분석하고, 지정된 테스트에 가장 적합한 매개변수 값에 대한 보고서를 제공할 수 있을 정도로 유연성이 있습니다.

벤치마크 테스트 실행

가장 일반적인 유형의 벤치마크 테스팅에서 구성 매개변수를 선택하고 최대 이점을 확인할 수 있을 때까지 이 매개변수에 다른 값을 사용하여 테스트를 실행하십시오.

단일 테스트에는 동일한 매개변수 값을 사용하는 응용프로그램의 반복 실행(예: 5회 또는 10회 반복)이 포함되어야 합니다. 따라서 다른 매개변수 값의 결과를 비교할 확실한 평균 성능 값을 구할 수 있습니다.

준비 실행이라고 하는 첫 번째 실행은 정상 실행이라고 하는 후속 실행과 별도로 고려해야 합니다. 준비 실행은 버퍼 풀 초기화와 같은 시작 활동을 포함하며 나중에 정상 실행보다 완료하는 데 시간이 오래 걸립니다. 준비 실행의 정보는 통계적으로 유효하지 않습니다. 특정 매개변수 값 세트의 평균을 계산할 때 정상 실행의 결과만 사용하십시오. 이는 종종 평균 계산 이전에 높은 값과 낮은 값을 삭제(drop)하기 위한 좋은 아이디어입니다.

실행 간의 일관성이 최대가 되도록 하려면 매번 새로 실행하기 전에 버퍼 풀이 알려진 상태로 돌아가는지 확인하십시오. 테스트하면 버퍼 풀이 데이터와 함께 로드되므로 필요한 디스크 입출력이 줄어들어 후속 실행이 빨라질 수 있습니다. 버퍼 풀의 컨텐츠는 다른 관련 없는 데이터를 버퍼 풀로 읽거나 모든 데이터베이스 연결이 임시로 제거된 경우 버퍼 풀을 할당 해제하여 강제로 삭제할 수 있습니다.

단일 매개변수 값 세트를 사용하여 테스트를 완료한 후 한 매개변수의 값을 변경할 수 있습니다. 각 반복 사이에 다음 태스크를 수행하여 벤치마크 환경을 원래의 상태로 리스토어하십시오.

- 테스트를 위해 카탈로그 통계가 갱신된 경우 통계의 모든 반복에 동일한 값이 사용되는지 확인하십시오.
- 테스트 중 갱신되는 테스트 데이터는 일관성이 있어야 합니다. 이 경우 다음을 수행해야 합니다.
  - 리스토어 유틸리티를 사용하여 전체 데이터베이스를 리스토어하십시오. 데이터베이스의 백업 사본에는 다음 테스트를 준비하기 위해 이전 상태가 들어 있습니다.
  - 임포트 또는 로드 유틸리티를 사용하여 데이터의 익스포트 사본을 리스토어하십시오. 이 방법을 사용하면 영향을 받은 데이터만 리스토어할 수 있습니다. 이 데이터가 들어 있는 테이블 및 인덱스에 대해 reorg 및 runstats 유틸리티를 실행해야 합니다.

즉, 다음 단계를 수행하여 데이터베이스 응용프로그램을 벤치마크 테스팅하십시오.

1단계 DB2 레지스트리, 데이터베이스 및 데이터베이스 관리 프로그램 구성 매개변수, 버퍼 풀을 표준 권장 값 그대로 사용하십시오. 여기에는 다음이 포함될 수 있습니다.

- 적절하고 오류 없는 응용프로그램 실행에 필요한 것으로 알려진 값
- 이전 조정에서 성능 향상을 제공한 값
- AUTOCONFIGURE 명령으로 제안된 값
- 디폴트값. 그러나 이 값은 적절하지 않을 수 있습니다.
  - 워크로드 및 테스트 목표에 중요한 매개변수
  - 응용프로그램의 장치 및 시스템 테스트 중 판별해야 하는 로그 크기의 경우
  - 응용프로그램을 실행하려면 변경해야 하는 매개변수의 경우

이 초기 경우에 대해 반복 세트를 실행하고 평균 경과 시간, 처리량 또는 프로세서 시간을 계산하십시오. 결과는 가능한 한 일관성이 있어야 하며 이상적으로는 실행 간에 별 차이가 없어야 합니다. 실행할 때마다 크게 달라지는 성능 측정은 조정하기가 매우 어렵습니다.

2단계 테스트할 하나의 메소드 또는 조정 매개변수만 선택하고 값을 변경하십시오.

3단계 다른 반복 세트를 실행하고 평균 경과 시간 또는 프로세서 시간을 계산하십시오.

4단계 벤치마크 테스트의 결과에 따라 다음 중 하나를 수행하십시오.

- 성능이 향상되면 동일한 매개변수의 값을 변경하고 3단계로 돌아가십시오. 최대 이점을 확인할 수 있을 때까지 이 매개변수를 계속 변경하십시오.
- 성능이 저하되었거나 변경되지 않은 경우 매개변수를 이전 값으로 되돌리고 2단계로 돌아가서 새 매개변수를 선택하십시오. 모든 매개변수를 테스트할 때까지 이 프로시저를 반복하십시오.

벤치마크 테스트 분석 예

벤치마크 테스팅 프로그램의 결과물에는 각 테스트의 ID, 반복 횟수, 명령문 수 및 각 실행의 경과 시간이 포함되어야 합니다.

이러한 샘플 보고서의 데이터는 설명을 위해서만 표시되었습니다. 실제 측정된 결과를 나타내지 않습니다.

벤치마크 테스팅 결과의 요약은 다음과 비슷합니다.

```
Test      Iter.   Stmt    Timing         SQL Statement
Numbr     Numbr   Numbr   (hh:mm:ss.ss)
 002       05      01     00:00:01.34    CONNECT TO SAMPLE
 002       05      10     00:02:08.15    OPEN cursor_01
 002       05      15     00:00:00.24    FETCH cursor_01
 002       05      15     00:00:00.23    FETCH cursor_01
 002       05      15     00:00:00.28    FETCH cursor_01
 002       05      15     00:00:00.21    FETCH cursor_01
 002       05      15     00:00:00.20    FETCH cursor_01
 002       05      15     00:00:00.22    FETCH cursor_01
 002       05      15     00:00:00.22    FETCH cursor_01
 002       05      20     00:00:00.84    CLOSE cursor_01
 002       05      99     00:00:00.03    CONNECT RESET
```

그림 1. 샘플 벤치마크 테스팅 결과

분석하면 CONNECT(명령문 01)를 완료하는 데 1.34초 걸리고, OPEN CURSOR(명령문 10)는 2분 8.15초 걸리고, FETCH(명령문 15)는 7행을 리턴했으며 가장 긴 지연 시간은 0.28초이고, CLOSE CURSOR(명령문 20)는 0.84초 걸리고, CONNECT RESET(명령문 99)은 완료하는 데 0.03초 걸렸습니다.

프로그램이 컬럼 식별자가 있는 ASCII 형식으로 데이터를 출력할 수 있는 경우 추가 통계 분석을 위해 나중에 데이터를 데이터베이스 테이블 또는 스프레드시트로 임포트할 수 있습니다.

요약 벤치마크 보고서는 다음과 비슷합니다.

```
PARAMETER        VALUES FOR EACH BENCHMARK TEST
TEST NUMBER      001     002     003     004     005
locklist         63      63      63       63      63
maxappls         8       8       8       8       8
applheapsz       48      48      48      48      48
dbheap           128     128     128     128     128
sortheap         256     256     256     256     256
maxlocks         22      22      22      22      22
stmtheap         1024    1024    1024    1024    1024
SQL STMT         AVERAGE TIMINGS (seconds)
  01             01.34   01.34   01.35   01.35   01.36
  10             02.15   02.00   01.55   01.24   01.00
  15             00.22   00.22   00.22   00.22   00.22
  20             00.84   00.84   00.84   00.84   00.84
  99             00.03   00.03   00.03   00.03   00.03
```

그림 2. 샘플 벤치마크 시간 제어 보고서

제 2 장 성능 모니터링 도구 및 방법론

시스템 성능 운영 모니터링

운영 모니터링은 시간에 따라 주기적인 간격으로 키 시스템 성능 메트릭을 수집하는 것을 의미합니다. 이 정보는 해당 초기 구성을 사용자의 요구사항에 맞게 세밀하게 조정하기 위한 중요한 데이터를 제공할 뿐 아니라 자체적으로 또는 소프트웨어 업그레이드, 데이터 또는 사용자 볼륨의 증가나 새 응용프로그램 전개에 따라 나타날 수 있는 새로운 문제점을 해결할 수 있도록 대비하게 해 줍니다.

운영 모니터링 고려사항

운영 모니터링 전략은 몇 가지 고려사항을 해결해야 합니다.

운영 모니터링은 매우 경량(측정하는 시스템을 많이 소비하지 않음)이고 일반적(시스템의 어느 곳에서나 나타날 수 있는 잠재적 문제점을 폭 넓게 『감시』)이어야 합니다.

시스템의 수명에 걸쳐 운영 메트릭의 일반 콜렉션을 계획하기 때문에, 모든 해당 데이터를 관리할 방법을 갖추는 것이 중요합니다. 성능의 장기 추세 등 데이터에 대해 가능한 여러 용도를 위해 수 개월씩 떨어진 데이터의 임의 콜렉션 간 비교를 수행해야 할 수 있습니다. DB2 제품 자체는 이러한 종류의 데이터 관리도 용이하게 합니다. 모니터링 데이터의 분석 및 비교가 매우 간단해졌고, 장기 데이터 스토리지 및 구성을 위한 강력한 인프라스트럭처가 이미 갖춰졌습니다.

DB2 데이터베이스(『DB2』) 시스템은 몇몇 훌륭한 모니터링 데이터 소스를 제공합니다. 기본 소스는 스냅샷 모니터 및 DB2 버전 9.5 이상에서 데이터 집계를 위한 워크로드 관리(WLM) 테이블 함수입니다. 이 두 기능 모두 카운터, 타이머 및 막대 그래프와 같은 도구에서 시스템의 활동 누계를 유지하는 요약 데이터에 중점을 둡니다. 시간에 따라 이러한 모니터 요소를 샘플링함으로써 시작 시간과 종료 시간 사이에 발생한 평균 활동을 도출할 수 있고, 이는 매우 유익할 수 있습니다.

DB2 제품에서 제공하는 메트릭만 사용할 이유는 없습니다. 사실, 비DB2 데이터는 생각보다 더 중요합니다. 컨텍스트 정보는 성능 문제점 판별에 매우 중요합니다. 사용자, 응용프로그램, 운영 체제, 스토리지 서브시스템 및 네트워크 등 모두는 시스템 성능에 대한 가치 있는 정보를 제공할 수 있습니다. DB2 데이터베이스 소프트웨어 외부의 메트릭을 포함시키는 것은 시스템 성능의 완전한 전체 그림을 생성하는 데 있어 중요한 부분입니다.

DB2 데이터베이스 제품 최근 릴리스의 추세는 SQL 인터페이스를 통해 점점 더 많은 모니터링 데이터를 사용할 수 있도록 만드는 것입니다. 이는, 예를 들어, 데이터를 관리

뷰에서 다시 DB2 테이블로 간편하게 리디렉션 할 수 있으므로 DB2에서 모니터링 데이터의 관리를 매우 간단하게 만듭니다. 더 나아가, 이벤트 및 활동 모니터 데이터 또한 DB2 테이블에 작성될 수 있으며 비슷한 이점을 제공합니다. 대부분의 모니터링 데이터는 DB2에서 저장하기가 매우 간편하므로, DB2에서 시스템 메트릭을 저장하기 위한 작은 투입(예:vmstat에서 CPU 사용)도 처리하기가 쉽습니다.

운영 모니터링을 위해 수집할 데이터 유형

진행 중인 운영 모니터링을 위해 수집할 유용한 몇 가지 유형의 데이터가 있습니다.

- DB2 시스템 성능 모니터링 메트릭 기본 세트
- DB2 구성 정보

  데이터베이스 및 데이터베이스 관리 프로그램 구성, DB2 레지스트리 변수 및 스키마 정의의 일반 사본을 사용하면 수행된 변경에 대한 실행기록을 제공하는 데 도움이 되며 모니터링 데이터에서 발생하는 변경을 설명하는 데 도움이 될 수 있습니다.
- 전체 시스템 로드

  CPU 또는 입출력 사용이 포화에 가까워지도록 허용될 경우, 이는 DB2 스냅샷만 사용하여 발견하기 어려울 수도 있는 시스템 병목 현상을 만들 수 있습니다. 따라서, UNIX 기반 시스템에서는 vmstat 및 iostat(및 네트워크 문제의 경우 netstat)를, Windows®에서는 perfmon을 사용하여 시스템 로드를 정기적으로 모니터하는 것이 가장 좋습니다. 또한 ENV_SYS_RESOURCES와 같은 관리 뷰를 사용하여 운영 체제, CPU, 메모리 및 시스템과 관련된 기타 정보를 검색할 수도 있습니다. 일반적으로 특정 범용 값보다는, 시스템의 일반적인 점에서의 변경을 찾으십시오.
- 비즈니스 논리 레벨에서 측정된 처리량 및 응답 시간

  DB2 위 비즈니스 논리 레벨에서 측정된 성능의 응용프로그램 뷰는 일반 사용자와 가장 밀접하다는 장점이 있을 뿐 아니라, 일반적으로 병목 현상을 만들 수 있는 프리젠테이션 논리,응용프로그램 서버, 웹 서버, 다중 네트워크 등과 같은 모든 것을 포함합니다. 이 데이터는 서비스 레벨 계약(SLA) 검증 및 설정 프로세스에 배우 중요할 수 있습니다.

DB2 시스템 성능 모니터링 요소 및 시스템 로드 데이터는 충분히 작아 5분에서 15분마다 수집된다 하더라도 시간 경과에 따른 총 데이터 볼륨이 대부분의 시스템에서 무관합니다. 마찬가지로, 이 데이트 수집의 오버헤드는 보통 추가 CPU 소비의 1-3퍼센트 범위에 있으며, 이는 중요한 시스템 메트릭스의 지속적 실행기록 대비 저렴한 비용입니다. 구성 정보는 보통 상대적으로 드물게 변경되므로, 일반적으로 하루에 한 번 구성 정보를 수집하는 것이 지나친 데이터 양을 생성하지 않으며 유용한 충분한 빈도입니다.

시스템 성능 모니터 요소 기본 세트

약 10개의 시스템 성능 메트릭이 진행 중인 운영 모니터링 노력에서 사용할 우수한 기본 세트를 제공합니다.

선택할 수백 개의 메트릭이 있지만, 이런 메트릭을 모두 수집하면 상당히 많은 데이터 볼륨이 생성되어 역효과가 날 수 있습니다. 다음과 같은 메트릭이 필요합니다.

- 수집하기 용이함 – 매일의 모니터링을 위해 복잡하거나 값비싼 도구를 사용하기를 원하지 않으며 시스템에 큰 부담을 주는 모니터링 작업을 원하지 않습니다.
- 이해하기 쉬움 – 메트릭을 볼 때마다 메트릭의 의미를 찾아보고 싶지 않습니다.
- 시스템과의 관련성 – 모든 메트릭은 모든 환경에서 의미 있는 정보를 제공하지는 않습니다.
- 민감하되 지나치게 민감하지 않을 것 – 메트릭의 변경은 시스템에서의 실제 변경을 나타냅니다. 메트릭은 자체적으로 변동되지 않아야 합니다.

이 시작 세트는 약 10개의 메트릭을 포함합니다.

- 실행된 트랜잭션 수:

  ```
  TOTAL_COMMITS
  ```

  이는 시스템 활동에 대한 훌륭한 기본 레벨 측정을 제공합니다.

- 데이터, 인덱스 및 임시 데이터에 대해 개별적으로 측정된 버퍼 풀 히트 비율:

  ```
  100 * (POOL_DATA_L_READS – POOL_DATA_P_READS) / POOL_DATA_L_READS
  100 * (POOL_INDEX_L_READS – POOL_INDEX_P_READS) / POOL_INDEX_L_READS
  100 * (POOL_TEMP_DATA_L_READS - POOL_TEMP_DATA_P_READS) / POOL_TEMP_DATA_L_READS
  100 * (POOL_TEMP_INDEX_L_READS - POOL_TEMP_INDEX_P_READS) / POOL_TEMP_INDEX_L_READS
  ```

  버퍼 풀 히트 비율은 가장 기본적인 메트릭 중 하나이고, 디스크 입출력을 피하기 위해 시스템이 얼마나 효과적으로 메모리를 이용하고 있는지에 대한 중요한 전체 척도를 제공합니다. 데이터에 대해 80-85% 이상의 히트 비율과 인덱스에 대해 90-95% 이상의 히트 비율이 일반적으로 OLTP 환경에 적당한 것으로 간주되며, 이러한 비율은 물론 버퍼 풀 스냅샷의 데이터를 사용하여 개별 버퍼 풀에 대해 계산될 수 있습니다.

  이러한 메트릭은 일반적으로 유용하지만 대형 테이블 스캔을 자주 수행하는 데이터 웨어하우스 같은 시스템의 경우, 데이터가 버퍼 풀에서 읽힌 후 다른 데이터를 위한 영역을 만들기 위해 제거되기 전에는 다시 사용되지 않기 때문에 데이터 히트 비율이 종종 회복할 수 없을 정도로 낮습니다.

- 트랜잭션당 버퍼 풀 실제 읽기 및 쓰기:

  ```
  (POOL_DATA_P_READS + POOL_INDEX_P_READS +
    POOL_TEMP_DATA_P_READS + POOL_TEMP_INDEX_P_READS)
    / TOTAL_COMMITS

  (POOL_DATA_WRITES + POOL_INDEX_WRITES)
    / TOTAL_COMMITS
  ```

  이러한 메트릭은 버퍼 풀 히트 비율과 밀접하게 관련되어 있지만 용도가 약간 다릅니다. 히트 비율에 대한 목표 값을 고려할 수 있지만, 트랜잭션당 읽기 및 쓰기에 대해 가능한 목표가 없습니다. 왜 이러한 계산이 방해가 될까요? 디스크 입출력은 데

이터베이스 성능에서 매우 중요한 인수이기 때문에 여러 방법으로 살펴보는 것이 유용합니다. 또한 이러한 계산에는 쓰기가 포함되는 반면, 히트 비율에서는 읽기만 다룹니다. 마지막으로, 분리에서, 예를 들어 94% 인덱스 히트 비율이 개선을 시도할 가치가 있는지 알기 어렵습니다. 시간당 100개의 논리적 인덱스 읽기만 있고 그 중 94개가 버퍼 풀에 있는 경우, 나머지 6개가 실제 읽기로 변하지 않도록 작업하는 것은 시간 낭비입니다. 그러나 94% 인덱스 히트 비율에 각 트랜잭션이 20개의 실제 읽기를 수행한다는 통계가 따를 경우 (데이터와 인덱스에 의해 추가 분석될 수 있음- 일반 및 임시), 버퍼 풀 히트 비율을 조사할 가치가 충분히 있을 수 있습니다.

메트릭은 단순히 실제적인 읽기 및 쓰기가 아니고 트랜잭션당 정규화됩니다. 많은 메트릭에서 이러한 추세를 따릅니다. 데이터가 수집된 기간 및 해당 시간에 시스템의 사용량이 높았는지 낮았는지 여부에서 메트릭을 분리하는 것이 목적입니다. 일반적으로, 이는 모니터링 데이터가 수집되는 방법 및 시기에 상관없이 메트릭에 대한 유사한 값을 얻는 데 도움이 됩니다. 데이터 콜렉션의 기간 및 시기에서 얼마간의 일관성은 좋은 것이지만, 정규화는 이를 매우 중요한 것에서 좋은 아이디어로 격하시킵니다.

- 선택한 행에 대한 읽은 데이터베이스 행의 비율:

  ```
  ROWS_READ / ROWS_RETURNED
  ```

  이 계산은 규정될 행을 찾기 위해 데이터베이스 테이블에서 읽힌 평균 행 수를 표시합니다. 낮은 수는 데이터 찾기의 효율성을 표시하며, 일반적으로 인덱스가 효과적으로 사용되고 있음을 나타냅니다. 예를 들어, 시스템에서 많은 테이블 스캔을 수행하고, 결과 세트로 규정할지 여부를 판별하기 위해 수 백만 개의 행을 검사해야 할 경우 이 수가 매우 높을 수 있습니다. 반면에, 완전한 고유 인덱스를 통해 테이블에 액세스할 경우 이 통계가 매우 낮을 수 있습니다. 인덱스만 액세스 플랜(테이블에서 행을 읽을 필요가 없는)은 ROWS_READ를 증가시키지 않습니다.

  OLTP 환경에서, 이 메트릭은 일반적으로 2또는 3보다 크지 않으며, 이는 대부분의 액세스가 테이블 스캔 대신 인덱스를 통해 이루어짐을 표시합니다. 이 메트릭은 시간 경과에 따른 플랜 안정성을 모니터하기 위한 간편한 방법입니다. 예기치 않은 증가는 종종 인덱스가 더 이상 사용되지 않으며 조사되어야 함을 표시합니다.

- 트랜잭션당 정렬에 소요된 시간:

  ```
  TOTAL_SORT_TIME / TOTAL_COMMITS
  ```

  넘치는 정렬로 인한 추가 오버헤드가 자동으로 여기에 포함되므로, 이는 정렬 통계를 처리하는 데 효과적인 방법입니다. 즉, 특히 시스템에 정렬에 대한 실행기록이 있는 경우, 분석을 쉽게 하기 위해 TOTAL_SORTS 및 SORT_OVERFLOWS를 수집하려 할 수도 있습니다.

- 트랜잭션 천 개당 누적된 잠금 대기 시간:

  ```
  1000 * LOCK_WAIT_TIME / TOTAL_COMMITS
  ```

지나친 잠금 대기 시간은 종종 부족한 응답 시간을 나타내므로 잠금 대기 시간을 모니터하는 것이 중요합니다. 단일 트랜잭션의 잠금 대기 시간은 일반적으로 매우 낮기 때문에 천 개의 트랜잭션에 대한 값으로 정규화됩니다. 천 개의 트랜잭션으로 범위를 확장하면 측정값을 처리하기가 쉬워집니다.

- 트랜잭션 천 개당 잠금 시간종료 및 교착 상태의 수:

  ```
  1000 * (DEADLOCKS + LOCK_TIMEOUTS) / TOTAL_COMMITS
  ```

  대부분의 프로덕션 제품에서는 교착 상태가 비교적 드물지만, 잠금 시간종료는 더 흔할 수 있습니다. 응용프로그램에서는 일반적으로 이들을 비슷한 방법으로 처리해야 합니다(트랜잭션을 처음부터 재실행). 이러한 발생 비율을 모니터하면 많은 교착 상태 또는 잠금 시간종료가 DBA에서 인식하지 않은 상태에서 시스템에 상당한 추가적인 부하를 주는 것을 방지하는 데 도움이 됩니다.

- 트랜잭션 천 개당 더티 스틸 트리거의 수:

  ```
  1000 * POOL_DRTY_PG_STEAL_CLNS / TOTAL_COMMITS
  ```

  『더티 스틸』은 버퍼 풀 정리를 트리거하는 데 있어 가장 선호되지 않는 방법입니다. 본질적으로, 새 버퍼 풀 페이지를 필요로 하는 SQL문의 처리는 희생(victim) 페이지의 갱신이 디스크에 적용 동안 인터럽트됩니다. 더티 스틸이 자주 발생하도록 허용된 경우, 처리량 및 응답 시간에 상당한 영향을 미칠 수 있습니다.

- 트랜잭션 천 개당 패키지 캐시 삽입의 수:

  ```
  1000 * PKG_CACHE_INSERTS / TOTAL_COMMITS
  ```

  패키지 캐시 삽입은 시스템의 일반 실행에 속하지만, 많은 수일 경우 CPU 시간의 상당한 소비자가 될 수 있습니다. 많은 잘 설계된 시스템에서는, 시스템이 안정된 상태에서 실행된 후 시스템에서 정적 SQL 또는 이전에 준비된 동적 SQL문을 사용하거나 재사용하기 때문에 매우 적은 패키지 캐시 삽입이 발생합니다. 임시 동적 SQL문의 트래픽이 높은 시스템에서는 SQL 컴파일 및 패키지 캐시 삽입을 피할 수 없습니다. 그러나 이 메트릭은 응용프로그램이 자주 실행되는 SQL에서 매개변수 표시 문자를 사용하지 않거나 준비된 명령문을 재사용하지 않음으로써 의도하지 않게 패키지 캐시 변동을 일으키는 등의 제 3의 유형의 상황을 경계하는 것을 목적으로 합니다.

- 에이전트가 로그 레코드를 디스크로 플러시하기 위해 기다리는 시간:

  ```
  LOG_WRITE_TIME
    / TOTAL_COMMITS
  ```

  트랜잭션 로그는 높은 레벨의 활동이나 부적절한 구성, 또는 기타 원인으로 인해 시스템 병목 현상을 일으킬 수 있는 가능성이 큽니다. 로그 활동을 모니터링하여 DB2 측의 문제(응용프로그램이 구동한 로그 요청 수의 증가를 의미)와 시스템 측의 문제(종종 하드웨어 또는 구성 문제로 인해 야기되는 로그 서브 시스템 성능의 감소로 인한)를 모두 감지할 수 있습니다.

- 파티션된 데이터베이스 환경에서 파티션 간에 주고 받는 FCM(Fast Communication Manager) 버퍼의 수:

  `FCM_SENDS_TOTAL, FCM_RECVS_TOTAL`

  이것은 클러스터에서 다른 파티션 간 데이터 플로우의 비율 및 특히, 플로우 밸런스 여부를 제공합니다. 다른 파티션에서 수신한 버퍼 수가 크게 다르다는 것은 각 파티션에 해시된 데이터 양의 비대칭을 나타낼 수 있습니다.

파티션된 데이터베이스 환경에서 파티션 간 모니터링

위에 언급된 거의 모든 개별 모니터링 요소 값은 파티션별로 보고됩니다.

일반적으로, 동일한 DB2 파티션 그룹의 모든 파티션에서는 대부분의 모니터링 통계가 매우 균일해야 합니다. 현저한 차이는 데이터 비대칭을 표시할 수 있습니다. 샘플 파티션 간 비교에는 다음이 포함됩니다.

- 데이터, 인덱스 및 임시 테이블의 논리적 및 실제 버퍼 풀 읽기
- 파티션 레벨에서 및 대형 테이블에 대한 행 읽기
- 정렬 시간 및 정렬 오버플로우
- FCM 버퍼 송신 및 수신
- CPU 및 입출력 사용

모니터링 데이터의 비정상 값

비정상 값을 식별하는 능력은 성능 문제점을 해결할 때 시스템 성능 모니터링 데이터를 해석하는 열쇠입니다.

모니터 요소는 값이 정상보다 잘못된 경우 즉, 값이 비정상인 경우 성능 문제점의 특성에 대한 실마리를 제공합니다. 일반적으로, 예를 들어 잘못된 값이 예측한 것보다 높으면 잠금 대기 시간이 길어집니다. 그러나 비정상 값은 예측한 것보다 낮을 수도 있습니다(예: 낮은 버퍼 풀 사용 비율). 상황에 따라 하나 이상의 방법을 사용하여 값이 정상보다 잘못되었는지 여부를 판별할 수 있습니다.

한 가지 접근 방법은 업계 경험 법칙 또는 우수 사례에 의존하는 것입니다. 예를 들어, 경험 법칙은 데이터에 대한 버퍼 풀 사용 비율이 80 - 85% 이상이면 OLTP 환경에 적합하다고 간주합니다. 이 경험 법칙은 OLTP 환경에 적용되며, 시스템 특성으로 인해 데이터 사용 비율이 훨씬 낮은 경우가 많은 데이터 웨어하우스의 경우에는 유용한 지침의 역할을 하지 못함에 유의하십시오.

다른 접근 방법은 현재 값을 이전에 수집된 기준선 값과 비교하는 것입니다. 이 접근 방법은 대단히 명확하며, 정상 조건 동안 주요 성능 메트릭을 수집하고 저장하기 위한 적절한 운영 모니터링 전략을 보유하는 데 따라 달라집니다. 예를 들어, 현재 버퍼 풀

사용 비율이 85%임을 알 수 있습니다. 이는 업계 표준에 따라 정상으로 간주되지만 성능 문제점이 보고되기 전에 99% 값을 기록한 것과 비교하면 비정상입니다.

마지막 접근 방법은 현재 값을 비교 가능한 시스템의 현재 값과 비교하는 것입니다. 예를 들어, 85%의 현재 버퍼 풀 사용 비율은 비교 가능한 시스템의 버퍼 풀 사용 비율이 99%이면 비정상으로 간주됩니다.

조정자 유틸리티

조정자(governor)는 데이터베이스에 실행되는 응용프로그램의 동작을 모니터하며 조정자 구성 파일에 지정한 규칙에 따라 동작을 변경할 수 있습니다.

중요사항: DB2 버전 9.5에서 새로운 전략적 DB2 워크로드 관리 프로그램 기능이 추가되어, DB2 조정자 유틸리티가 버전 9.7에서 사용되지 않았으며 추후 릴리스에서도 제거될 수 있습니다. 조정자 유틸리티의 사용되지 않음에 대한 자세한 정보는 『DB2 조정자 및 Query Patroller가 사용되지 않음』을 참조하십시오. DB2 워크로드 관리 프로그램과, 이 관리 프로그램이 조정자 유틸리티를 교체하는 방법에 대해 학습하려면 『DB2 워크로드 관리 프로그램 개념 소개』 및 『DB2 워크로드 관리 프로그램에 대한 자주 묻는 질문』을 참조하십시오.

조정자 인스턴스는 프론트엔드 유틸리티와 하나 이상의 디먼으로 구성되어 있습니다. 각 조정자 인스턴스는 데이터베이스 관리 프로그램 인스턴스마다 다릅니다. 디폴트로 조정자를 시작하면 파티션된 데이터베이스의 각 데이터베이스 파티션에서 조정자 디먼이 시작됩니다. 그러나 모니터할 단일 데이터베이스 파티션에서 디먼을 시작하도록 지정할 수 있습니다.

조정자는 조정자 구성 파일의 규칙에 따라 응용프로그램 트랜잭션을 관리합니다. 예를 들어, 규칙을 적용하면 응용프로그램이 특정 자원을 너무 많이 사용 중임을 확인할 수 있습니다. 규칙은 수행할 조치(예: 응용프로그램 우선순위 변경 또는 데이터베이스에서 강제 연결 끊기)도 지정합니다.

규칙과 연관된 조치가 응용프로그램의 우선순위를 변경한 경우 조정자는 자원 위반이 발생한 데이터베이스 파티션에서 에이전트의 우선순위를 변경합니다. 파티션된 데이터베이스에서 응용프로그램과 데이터베이스의 연결을 강제로 끊은 경우 위반을 감지한 디먼이 응용프로그램의 코디네이터 노드에서 실행되고 있어도 이 조치가 발생합니다.

조정자는 수행되는 조치를 로그합니다.

주: 조정자가 활성화되면 스냅샷 요청이 데이터베이스 관리 프로그램 성능에 영향을 미칠 수 있습니다. 성능을 향상시키려면 조정자 대기 간격을 늘려서 CPU 사용량을 줄이십시오.

조정자(governor) 시작 및 중지

조정자 유틸리티는 데이터베이스에 연결된 응용프로그램을 모니터하고 해당 데이터베이스에 대한 조정자(governor) 구성 파일에 지정된 규칙에 따라 이러한 응용프로그램의 동작을 변경합니다.

중요사항: 새 워크로드 관리 기능이 DB2 버전 9.5에 도입되었으므로 DB2 조정자는 버전 9.7에서 사용되지 않으며 이후 릴리스에서 제거될 수 있습니다. 자세한 정보는 *버전 9.7의 새로운 내용* 책에 있는 『DB2 조정자 및 Query Patroller는 사용되지 않음』 주제를 참조하십시오.

조정자(governor)를 시작하기 전에, 조정자(governor) 구성 파일을 작성해야 합니다.

조정자(governor)를 시작하거나 중지하려면, *sysadm* 또는 *sysctrl* 권한이 있어야 합니다.

1. 조정자(governor)를 시작하려면 db2gov 명령을 사용하며 다음 필수 매개변수를 지정하십시오.

   START *database-name*
   지정하는 데이터베이스 이름은 조정자(governor) 구성 파일의 데이터베이스 이름과 일치해야 합니다.

   config-file
   이 데이터베이스에 대한 조정자(governor) 구성 파일의 이름입니다. 이 파일이 디폴트 위치인 `sqllib` 디렉토리에 없는 경우, 파일 이름뿐 아니라 파일 경로를 포함해야 합니다.

   log-file
   이 조정자(governor)에 대한 로그 파일의 기본 이름입니다. 파티션된 데이터베이스의 경우, 이 인스턴스의 조정자(governor)에 대해 디먼이 실행 중인 각 데이터베이스 파티션에 대해 데이터베이스 파티션 번호가 추가됩니다.

   파티션된 데이터베이스의 단일 데이터베이스 파티션에서 조정자(governor)를 시작하려면 **dbpartitionnum** 옵션을 지정하십시오.

   예를 들어, `salescfg`라는 이름의 구성 파일과 `saleslog`라는 이름의 로그 파일을 사용하여 SALES라는 이름의 데이터베이스의 데이터베이스 파티션 3에서 조정자(governor)를 시작하려면 다음 명령을 입력하십시오.

   ```
   db2gov start sales dbpartitionnum 3 salescfg saleslog
   ```

   모든 데이터베이스 파티션에서 조정자(governor)를 시작하려면 다음 명령을 입력하십시오.

   ```
   db2gov start sales salescfg saleslog
   ```

2. 조정자(governor)를 중지하려면 db2gov 명령을 사용하며 STOP 옵션을 지정하십시오.

   예를 들어, SALES 데이터베이스의 모든 데이터베이스 파티션에서 조정자(governor)를 중지하려면 다음 명령을 입력하십시오.

   ```
   db2gov stop sales
   ```

   데이터베이스 3에서만 조정자(governor)를 중지하려면 다음 명령을 입력하십시오.

   ```
   db2gov stop sales dbpartitionnum 3
   ```

조정자 디먼

조정자 디먼은 데이터베이스에 실행되는 응용프로그램에 대한 정보를 수집합니다.

조정자 디먼은 시작될 때마다 다음의 태스크 루프를 실행합니다.

1. 디먼은 조정자 구성 파일이 변경되었거나 이 파일을 아직 읽지 않았는지 점검합니다. 이러한 상태 중 하나에 해당하는 경우 디먼은 파일에서 규칙을 읽습니다. 이 경우 실행 중인 상태에서 조정자 디먼의 동작을 변경할 수 있습니다.
2. 디먼은 데이터베이스에서 작동되는 각 응용프로그램 및 에이전트의 자원 사용 통계에 대한 스냅샷 정보를 요청합니다.
3. 디먼은 조정자 구성 파일의 규칙에 대해 각 응용프로그램의 통계를 점검합니다. 규칙이 적용되면 조정자는 지정된 조치를 수행합니다. 조정자는 누적된 정보를 구성 파일에 정의된 값과 비교합니다. 즉, 구성 파일이 응용프로그램이 이미 위반했을 수 있는 새 값으로 갱신된 경우 해당 위반과 관련된 규칙은 다음 조정자 간격 중 응용프로그램에 적용됩니다.
4. 디먼은 수행하는 조정자 로그 파일에 수행하는 조치에 대한 레코드를 기록합니다.

조정자가 태스크를 완료하면 구성 파일에 지정된 간격 동안 휴면 상태가 됩니다. 해당 간격이 경과하면 조정자는 대기 상태가 되고 태스크 루프를 다시 시작합니다.

조정자가 오류 또는 중지 신호에 직면한 경우 중지하기 전에 정리 처리를 수행합니다. 우선순위가 설정된 응용프로그램 목록을 사용하여 정리 처리는 모든 응용프로그램 에이전트 우선순위를 재설정합니다. 그런 다음 응용프로그램에서 더 이상 작동하지 않는 에이전트의 우선순위를 재설정합니다. 따라서 조정자가 종료된 후 에이전트가 비디폴트 우선순위로 계속 실행되지 않습니다. 오류가 발생하면 조정자는 관리 통지 로그에 비정상적으로 종료되었음을 표시하는 메시지를 기록합니다.

agentpri 데이터베이스 관리 프로그램 구성 매개변수의 값이 시스템 디폴트가 아닌 경우 조정자를 사용하여 에이전트 우선순위를 조정할 수 없습니다.

조정자 디먼은 데이터베이스 응용프로그램이 아니므로 데이터베이스와의 연결을 유지하지 않지만 인스턴스 접속을 가지고 있습니다. 스냅샷 요청을 발행할 수 있으므로 조정자 디먼은 데이터베이스 관리 프로그램이 종료되면 감지할 수 있습니다.

조정자 구성 파일

조정자 구성 파일에는 데이터베이스에 실행되는 응용프로그램을 조정하는 규칙이 들어 있습니다.

조정자는 각 규칙을 평가하고 규칙이 참으로 평가되면 지정된 조치를 수행합니다.

조정자 구성 파일에는 모니터할 데이터베이스(필수), CPU 사용량 통계가 포함된 어카운트 레코드를 기록하는 간격 및 조정자 디먼의 휴면 간격을 식별하는 일반 절이 있습니다. 구성 파일에는 하나 이상의 선택적 응용프로그램 모니터링 규칙 명령문도 포함될 수 있습니다. 다음 지침은 일반 절 및 규칙 명령문에 모두 적용됩니다.

- 일반 주석을 중괄호({ })로 구분하십시오.
- 대부분의 경우 대문자, 소문자 또는 대소문자 혼용을 사용하여 값을 지정하십시오. 대소문자를 구분하는 응용프로그램 이름의 경우는 예외입니다(applname절 뒤에 지정함).
- 각 일반 절 또는 규칙 명령문을 세미콜론(;)으로 종료하십시오.

규칙을 갱신해야 하는 경우 조정자를 중지하지 말고 구성 파일을 편집하십시오. 각 조정자 디먼은 파일이 변경되었음을 감지하고 다시 읽습니다.

파티션된 데이터베이스 환경에서는 각 데이터베이스 파티션의 조정자 디먼이 동일한 구성 파일을 읽을 수 있도록 모든 데이터베이스 파티션에서 마운트되는 디렉토리에서 조정자 구성 파일을 작성해야 합니다.

일반 절

조정자 구성 파일에서 다음 절은 한 번만 지정할 수 있습니다.

dbname

모니터할 데이터베이스의 이름 또는 별명입니다. 이 절은 필수 절입니다.

account *n*

각 연결의 CPU 사용량 통계가 들어 있는 어카운트 레코드를 기록하려면 경과해야 하는 간격(분)입니다. 이 옵션은 Windows 운영 체제에서 사용할 수 없습니다. 일부 플랫폼에서는 스냅샷 모니터에서 CPU 통계를 사용할 수 없습니다. 이 경우 account절을 무시합니다.

어카운트 간격에서만 간단한 세션이 발생한 경우 로그 레코드가 기록되지 않습니다. 로그 레코드가 기록되면 연결에 대한 이전 로그 레코드 이후의 CPU 사용량을 반영하는 CPU 통계가 포함됩니다. 조정자가 중지된 후 재시작되면 두 개의 로그 레코드에 CPU 사용량이 반영될 수 있습니다. 이러한 로그 레코드는 로그 레코드의 응용프로그램 ID를 통해 식별할 수 있습니다.

interval *n*

디먼이 대기 상태가 되려면 경과해야 하는 간격(초)입니다. 이 절을 지정하지 않은 경우 디폴트값인 120초를 사용합니다.

규칙 절

규칙 명령문은 응용프로그램을 조정하고 규칙 절이라고 하는 작은 구성요소에서 어셈블하는 방식을 지정합니다. 규칙 절을 사용할 경우 다음과 같이 규칙 명령문에 특정 순서로 표시됩니다.

1. **desc**: 작은따옴표 또는 큰따옴표로 묶은 규칙에 대한 주석
2. **time**: 규칙을 평가하는 시간
3. **authid**: 응용프로그램이 명령문을 실행할 때 사용하는 하나 이상의 권한 ID
4. **applname**: 데이터베이스에 연결된 실행 파일 또는 오브젝트 파일의 이름. 이 이름은 대소문자를 구분합니다. 응용프로그램 이름에 공백이 포함된 경우 이름을 큰따옴표로 묶어야 합니다.
5. **setlimit**: 조정자가 점검하는 한계(예: CPU 시간, 리턴되는 행 수 또는 유휴 시간). 일부 플랫폼에서는 스냅샷 모니터에서 CPU 통계를 사용할 수 없습니다. 이 경우 setlimit절을 무시합니다.
6. **action**: 한계에 도달하면 수행할 조치. 아무 조치도 지정되지 않은 경우 조정자는 한계에 도달하면 응용프로그램에 대해 작동하는 에이전트의 우선순위를 10으로 줄입니다. 응용프로그램에 대해 수행할 수 있는 조치로는 에이전트 우선순위 줄임, 데이터베이스에서 강제 연결 끊기 또는 조작의 스케줄링 옵션 설정이 있습니다.

규칙 절들을 결합하여 하나의 규칙 명령문을 구성하십시오(각 규칙 명령문에서 특정 절은 한 번만 사용).

```
desc "Allow no UOW to run for more than an hour"
setlimit uowtime 3600 action force;
```

응용프로그램에 여러 규칙이 적용되는 경우 모든 규칙이 적용됩니다. 보통 처음으로 발견된 한계와 연관된 조치가 가장 먼저 적용되는 조치입니다. 규칙 절에 -1 값을 지정하면 예외가 발생합니다. 동일한 절에 대해 나중에 지정한 값은 이전에 지정한 값을 겹쳐 쓸 수만 있습니다. 이전 규칙 명령문의 기타 절은 계속 작동 가능합니다.

예를 들어, 하나의 규칙 명령문이 `rowssel 100000` 및 `uowtime 3600`절을 사용하여 경과 시간이 1시간을 초과하거나 선택한 행이 100,000행을 초과할 경우 응용프로그램의 우선순위가 낮아짐을 지정합니다. 연속 규칙에서는 `uowtime -1`절을 사용하여 동일한 응용프로그램에 무제한 경과 시간이 있음을 지정합니다. 이 경우 응용프로그램이 1시간 이상 실행되어도 우선순위가 변경되지 않습니다. 즉, `uowtime -1`은 `uowtime 3600`을 겹쳐씁니다. 그러나 100,000행 이상을 선택하면 `rowssel 100000`이 계속 적용되므로 우선순위가 낮아집니다.

규칙 응용프로그램 순서

조정자는 구성 파일의 맨 위부터 맨 아래로 규칙을 처리합니다. 그러나 특정 규칙 명령문의 setlimit절이 이전 규칙 명령문의 동일한 절보다 제한 정도가 약한 경우 더 제한적인 규칙이 적용됩니다. 다음 예에서 첫 번째 규칙이 더 제한적인 경우 ADMIN은 계속 5000행으로 제한됩니다.

```
desc "Force anyone who selects 5000 or more rows."
setlimit rowssel 5000 action force;

desc "Allow user admin to select more rows."
authid admin setlimit rowssel 10000 action force;
```

제한 정도가 약한 규칙이 제한 정도가 강한 규칙을 겹쳐쓰도록 하려면 -1을 지정하여 새 규칙을 적용하기 전에 이전 규칙을 지우십시오. 예를 들어, 다음 구성 파일에서 초기 규칙은 모든 사용자를 5000행으로 제한합니다. 두 번째 규칙은 ADMIN에 대한 이 한계를 지우고 세 번째 규칙은 ADMIN의 한계를 10000행으로 재설정합니다.

```
desc "Force anyone who selects 5000 or more rows."
setlimit rowssel 5000 action force;

desc "Clear the rowssel limit for admin."
authid admin setlimit rowssel -1;

desc "Now set the higher rowssel limit for admin"
authid admin  setlimit rowssel 10000 action force;
```

조정자 구성 파일 예

```
{ The database name is SAMPLE; do accounting every 30 minutes;
  wake up once a second. }
dbname sample; account 30; interval 1;

desc "CPU restrictions apply to everyone 24 hours a day."
setlimit cpu 600 rowssel 1000000 rowsread 5000000;

desc "Allow no UOW to run for more than an hour."
setlimit uowtime 3600 action force;

desc 'Slow down a subset of applications.'
applname jointA, jointB, jointC, quryA
setlimit cpu 3 locks 1000 rowssel 500 rowsread 5000;

desc "Have the governor prioritize these 6 long apps in 1 class."
applname longq1, longq2, longq3, longq4, longq5, longq6
setlimit cpu -1
action schedule class;

desc "Schedule all applications run by the planning department."
authid planid1, planid2, planid3, planid4, planid5
setlimit cpu -1
action schedule;

desc "Schedule all CPU hogs in one class, which will control consumption."
setlimit cpu 3600
```

```
action schedule class;

desc "Slow down the use of the DB2 CLP by the novice user."
authid novice
applname db2bp.exe
setlimit cpu 5 locks 100 rowssel 250;

desc "During the day, do not let anyone run for more than 10 seconds."
time 8:30 17:00 setlimit cpu 10 action force;

desc "Allow users doing performance tuning to run some of
     their applications during the lunch hour."
time 12:00 13:00 authid ming, geoffrey, john, bill
applname tpcc1, tpcc2, tpcA, tpvG
setlimit cpu 600 rowssel 120000 action force;

desc "Increase the priority of an important application so it always
      completes quickly."
applname V1app setlimit cpu 1 locks 1 rowssel 1 action priority -20;

desc "Some people, such as the database administrator (and others),
     should not be limited. Because this is the last specification
     in the file, it will override what came before."
authid gene, hershel, janet setlimit cpu -1 locks -1 rowssel -1 uowtime -1;
```

조정자 규칙 절

조정자 구성 파일의 각 규칙은 규칙을 적용하는 조건과 규칙이 참으로 평가되면 발생하는 조치를 지정하는 절로 구성되어 있습니다.

규칙 절은 표시된 순서대로 지정해야 합니다.

선택적 시작 절

desc 규칙에 대한 설명을 지정합니다. 설명은 작은따옴표 또는 큰따옴표로 묶어야 합니다.

time 규칙을 평가하는 기간을 지정합니다. 기간은 `time hh:mm hh:mm` 형식(예: `time 8:00 18:00`)으로 지정해야 합니다. 이 절을 지정하지 않으면 규칙은 매일 24시간 평가합니다.

authid

응용프로그램을 실행할 때 사용하는 하나 이상의 권한 부여 ID를 지정합니다. 권한 부여 ID가 여러 개인 경우 쉼표(,)로 구분해야 합니다(예: `authid gene, michael, james`). 이 절을 지정하지 않으면 규칙은 모든 권한 부여 ID에 적용됩니다.

applname

데이터베이스에 연결된 실행 파일 또는 오브젝트 파일의 이름을 지정합니다. 응용프로그램 이름이 여러 개인 경우 쉼표(,)로 구분해야 합니다(예: `applname db2bp, batch, geneprog`). 이 절을 지정하지 않으면 규칙은 모든 응용프로그램 이름에 적용됩니다.

주:

1. 응용프로그램 이름은 대소문자를 구분합니다.
2. 데이터베이스 관리 프로그램이 모든 응용프로그램 이름을 20자로 자릅니다. 조정할 응용프로그램이 응용프로그램 이름의 처음 20자로 고유하게 식별되는지 확인해야 합니다. 조정자 구성 파일에 지정된 응용프로그램 이름을 내부 표현에 맞게 20자로 자릅니다.

한계 절

setlimit

조정자가 점검할 하나 이상의 한계를 지정합니다. 한계는 -1이거나 0보다 커야 합니다(예: `cpu -1 locks 1000 rowssel 10000`). 최소한 하나의 한계를 지정해야 하며 규칙 명령문에 지정되지 않은 한계는 해당 규칙으로 제한되지 않습니다. 조정자는 다음 한계를 점검할 수 있습니다.

cpu *n* 응용프로그램이 사용할 수 있는 CPU 시간(초)을 지정합니다. -1을 지정하면 응용프로그램의 CPU 사용량이 제한되지 않습니다.

idle *n* 연결에서 허용되는 유휴 시간(초)을 지정합니다. -1을 지정하면 연결의 유휴 시간이 제한되지 않습니다.

주: 일부 데이터베이스 유틸리티(예: 백업 및 리스토어)는 데이터베이스에 대한 연결을 설정한 후 조정자에게 표시되지 않는 EDU(Engine Dispatchable Unit)를 통해 작업을 수행합니다. 이러한 데이터베이스 연결은 유휴 상태로 나타나며 유휴 시간 한계를 초과할 수도 있습니다. 조정자가 이러한 유틸리티에 대해 조치를 수행하지 못하도록 방지하려면 이를 호출한 권한 부여 ID를 통해 -1을 지정하십시오. 예를 들어, 조정자가 권한 부여 ID DB2SYS에서 실행 중인 유틸리티에 대해 조치를 수행하지 못하도록 예방하려면 `authid DB2SYS setlimit idle -1`을 지정하십시오.

locks *n*

응용프로그램이 보유할 수 있는 잠금 수를 지정합니다. -1을 지정하면 응용프로그램이 보유한 잠금 수가 제한되지 않습니다.

rowsread *n*

응용프로그램이 선택할 수 있는 행 수를 지정합니다. -1을 지정하면 응용프로그램이 선택할 수 있는 행 수가 제한되지 않습니다. 지정할 수 있는 최대값은 4 294 967 298입니다.

주: 이 한계는 `rowssel`과 동일하지 않습니다. `rowsread`가 결과 세트를 리턴하기 위해 읽어야 하는 행 수라는 점이 다릅니다. 이 수에는 카탈로그 테이블에서 읽은 엔진 수가 포함되며 인덱스 사용 시 줄어들 수 있습니다.

rowssel *n*

응용프로그램에 리턴될 수 있는 행 수를 지정합니다. 이 값은 코디네이터 데이터베이스 파티션에서만 0이 아닙니다. -1을 지정하면 리턴될 수 있는 행 수는 제한되지 않습니다. 지정할 수 있는 최대값은 4 294 967 298입니다.

uowtime *n*

작업 단위(UOW)가 처음 활성화된 시간부터 경과할 수 있는 시간(초)을 지정합니다. -1을 지정하면 경과 시간은 제한되지 않습니다.

주: sqlmon API를 사용하여 작업 단위 모니터 스위치 또는 시간소인 모니터 스위치를 비활성화한 경우 작업 단위 경과 시간에 따라 응용프로그램을 조정하는 조정자의 기능에 영향을 줍니다. 조정자는 모니터를 사용하여 시스템에 대한 정보를 수집합니다. 데이터베이스 관리 프로그램 구성 파일에서 스위치를 끄면 전체 인스턴스에서 꺼지며 조정자는 더 이상 이 정보를 받을 수 없습니다.

조치 절

action 지정된 하나 이상의 한계를 초과한 경우 수행할 조치를 지정합니다. 만약 한계를 초과했으며 action절이 지정되지 않은 경우 조정자는 응용프로그램에 대해 작동하는 에이전트의 우선순위를 10으로 줄입니다.

force 응용프로그램에 서비스를 제공하는 에이전트를 강제 실행하도록 지정합니다(FORCE APPLICATION 명령이 코디네이터 에이전트를 종료함).

주: 파티션된 데이터베이스 환경에서 force 조치는 조정자 디먼이 응용프로그램의 코디네이터 데이터베이스 파티션에서 실행되고 있는 경우에만 수행됩니다. 따라서 조정자 디먼이 데이터베이스 파티션 A에서 실행되고 있으며 코디네이터 데이터베이스 파티션이 B 데이터베이스 파티션인 일부 응용프로그램의 한계를 초과한 경우 force 조치를 건너뜁니다.

nice *n* 응용프로그램에 대해 작동되는 에이전트의 상대적 우선순위에 대한 변경사항을 지정합니다. 유효한 값은 UNIX 기반 시스템의 경우 -20부터 +20까지이고 Windows 플랫폼의 경우 -1에서 6까지입니다.

- UNIX 기반 시스템에서 **agentpri** 데이터베이스 관리 프로그램 구성 매개변수를 디폴트값으로 설정해야 합니다. 그렇지 않으면 nice 값을 겹쳐씁니다.
- Windows 플랫폼에서 **agentpri** 데이터베이스 관리 프로그램 구성 매개변수 및 nice 값을 함께 사용할 수 있습니다.

조정자를 사용하여 디폴트 사용자 서비스 수퍼 클래스 SYSDEFAULTUSERCLASS에서 실행되는 응용프로그램의 우선순위

를 제어할 수 있습니다. 조정자를 사용하여 이 서비스 수퍼 클래스에서 실행되는 응용프로그램의 우선순위를 낮출 경우 에이전트는 아웃바운드 상관자와의 연관을 취소하고(연관된 경우) 조정자가 지정한 에이전트 우선순위에 따라 상대적 우선순위를 설정합니다. 조정자를 사용하여 사용자 정의 서비스 수퍼 클래스 및 서브클래스에서 에이전트의 우선순위를 변경할 수 없습니다. 대신 서비스 수퍼 클래스 또는 서브클래스의 에이전트 우선순위 설정을 사용하여 이러한 서비스 클래스에서 실행되는 응용프로그램을 제어해야 합니다. 그러나 조정자를 사용하여 모든 서비스 클래스에서 연결을 강제 실행할 수 있습니다.

주: AIX® 5.3에서 응용프로그램에 대해 작동되는 에이전트의 상대적 우선순위를 높이려면 인스턴스 소유자는 CAP_NUMA_ATTACH 성능을 가지고 있어야 합니다. 이 권한을 부여하려면 루트로 로그온하고 다음 명령을 실행하십시오.

```
chuser capabilities=CAP_NUMA_ATTACH,CAP_PROPAGATE
```

schedule [class]

스케줄링은 응용프로그램에서 작동되는 에이전트의 우선순위를 개선합니다. 모든 응용프로그램에서 공정성을 유지하면서 평균 응답 시간을 최소화하기 위한 것입니다.

조정자는 다음 기준에 따른 스케줄링을 위해 최상위 응용프로그램을 선택합니다.

- 보유하는 잠금 수가 가장 많은 응용프로그램(잠금 대기 수를 줄이는 시도)
- 가장 오래된 응용프로그램
- 남아 있는 추정 실행 시간이 가장 짧은 응용프로그램(간격 중 짧은 명령문을 가능하면 많이 완료하려는 시도)

각 기준에서 상위 세 개의 응용프로그램에는 다른 모든 응용프로그램보다 높은 우선순위가 부여됩니다. 즉, 각 기준 그룹의 상위 응용프로그램에 최고 우선순위가 부여되고 다음으로 높은 응용프로그램에 두 번째로 높은 우선순위가 부여되고 세 번째로 높은 응용프로그램에 세 번째로 높은 우선순위가 부여됩니다. 단일 응용프로그램이 여러 기준에서 상위 3위 안에 포함된 경우 최고 순위가 지정된 기준에 적합한 우선순위가 부여되고 다음으로 높은 응용프로그램에 기타 기준에 대한 다음 최고 우선순위가 부여됩니다. 예를 들어, A 응용프로그램이 가장 많은 잠금을 보유하지만 남은 추정 실행 시간이 세 번째로 가장 짧은 경우 첫 번째 기준에 따라 최고 우선순위가 부여됩니다. 남은 추정 실행 시간이 가장 짧은 네 번째 순위의 응용프로그램에는 해당 기준에 따라 세 번째 최고 우선순위가 부여됩니다.

이 조정자 규칙이 선택한 응용프로그램은 세 개의 클래스로 구성됩니다. 조정자는 위에 설명된 기준에 따라 각 클래스에서 상위 세 개의 응용프로그램을 선택하여 아홉 개의 응용프로그램을 선택합니다. class 옵션을 지정할 경우 이 규칙에 따라 선택한 모든 응용프로그램은 단일 클래스로 간주되며 위에 설명된 바와 같이 아홉 개의 응용프로그램을 선택하여 높은 우선순위를 부여합니다.

여러 조정자 규칙에서 응용프로그램을 선택한 경우 선택한 마지막 규칙에 따라 조정됩니다.

주: sqlmon API를 사용하여 명령문 스위치를 비활성화한 경우 명령문 경과 시간에 따라 응용프로그램을 제어하는 조정자의 기능에 영향을 줍니다. 조정자는 모니터를 사용하여 시스템에 대한 정보를 수집합니다. 데이터베이스 관리 프로그램 구성 파일에서 스위치를 끄면 전체 인스턴스에서 꺼지며 조정자는 더 이상 이 정보를 받을 수 없습니다.
스케줄 조치는 다음을 수행할 수 있습니다.

- 모든 응용프로그램에 시간을 균일하게 분할하지 않고 여러 그룹의 응용프로그램들이 시간을 사용하도록 할 수 있습니다. 예를 들어, 14 개의 응용프로그램(세 개: 짧음, 다섯 개: 중간, 여섯 개: 김)을 동시에 실행 중인 경우 CPU를 분할하여 사용하므로 모두 응답 시간이 느릴 수 있습니다. 데이터베이스 관리자는 중간 길이의 응용프로그램과 긴 길이의 응용프로그램으로 두 개의 그룹을 설정할 수 있습니다. 우선순위를 사용하여 조정자는 모든 짧은 응용프로그램을 실행하도록 허용하고 최대 세 개의 중간 응용프로그램과 세 개의 긴 응용프로그램을 동시에 실행할 수 있도록 합니다. 이를 위해서 조정자 구성 파일에는 중간 길이 응용프로그램을 위한 하나의 규칙과 긴 응용프로그램을 위한 하나의 규칙이 있습니다.

  다음 예는 이 내용에 대해 설명한 조정자 구성 파일의 일부분입니다.

  ```
  desc "Group together medium applications in 1 schedule class."
  applname medq1, medq2, medq3, medq4, medq5
  setlimit cpu -1
  action schedule class;

  desc "Group together long applications in 1 schedule class."
  applname longq1, longq2, longq3, longq4, longq5, longq6
  setlimit cpu -1
  action schedule class;
  ```

- 여러 개의 각 사용자 그룹(예: 조직 부서)에 동일한 우선순위가 있는지 확인하십시오. 한 그룹이 여러 응용프로그램을 실행 중인 경우 관리자는 다른 그룹이 해당 응용프로그램에 대해 계속 합리적인 응답 시간을 확보할 수 있는지 확인할 수 있습니다. 예를 들어, 세 개의 부서(재정, 재고 및 기획)가 관련된 경우 모든 재정 사용자를 한

그룹으로 묶고 모든 재고 사용자를 두 번째 그룹으로 묶고 모든 계획 사용자를 세 번째 그룹으로 묶을 수 있습니다. 처리 능력은 세 개의 부서로 다소 균일하게 분할됩니다.

다음 예는 이 내용에 대해 설명한 조정자 구성 파일의 일부분입니다.

```
desc "Group together Finance department users."
authid tom, dick, harry, mo, larry, curly
setlimit cpu -1
action schedule class;

desc "Group together Inventory department users."
authid pat, chris, jack, jill
setlimit cpu -1
action schedule class;

desc "Group together Planning department users."
authid tara, dianne, henrietta, maureen, linda, candy
setlimit cpu -1
action schedule class;
```

- 조정자가 모든 응용프로그램을 스케줄링하도록 하십시오.

  class 옵션이 지정되지 않은 경우 조정자는 스케줄 조치에 속한 활성 응용프로그램 수에 따라 직접 클래스를 작성하며 응용프로그램이 실행 중인 쿼리에 대한 쿼리 컴파일러의 비용 추정에 따라 여러 클래스로 응용프로그램을 분리합니다. 관리자는 선택되는 응용프로그램을 규정하지 않고(즉, applname, authid 또는 setlimit절을 지정하지 않음) 모든 응용프로그램을 스케줄링하도록 선택할 수 있습니다.

조정자 로그 파일

조정자 디먼이 조치를 수행할 때마다 로그 파일에 레코드를 기록합니다.

조치는 다음과 같습니다.

- 조정자 시작 또는 중지
- 조정자 구성 파일 읽기
- 응용프로그램의 우선순위 변경
- 응용프로그램 강제 실행
- 오류 또는 경고 직면

각 조정자 디먼에는 여러 조정자 디먼이 동일한 파일에 동시에 쓰려고 할 때 발생할 수 있는 파일 잠금 병목 현상을 방지하는 개별 로그 파일이 있습니다. 조정자 로그 파일을 쿼리하려면 db2govlg 명령을 사용하십시오.

로그 파일은 sqllib 디렉토리의 log 서브디렉토리에 저장되지만 Windows 운영 체제의 경우에는 Windows 운영 체제가 응용프로그램 로그 파일을 호스트하는 데 사용하는 공통 응용프로그램 데이터 디렉토리에서 log 서브디렉토리가 있는 위치에 저장됩니다. db2gov 명령을 사용하여 조정자를 시작할 때 로그 파일의 기본 이름을 제공하십시오. 로그 파일 이름에 조정되는 각 데이터베이스 파티션의 로그 파일을 구별하는 데이터베이스 이름이 있는지 확인하십시오. 파티션된 데이터베이스 환경에서 각 조정자마다 고유한 파일 이름을 사용하도록 조정자 디먼이 실행되는 데이터베이스 파티션 수가 로그 파일 이름에 자동으로 추가됩니다.

로그 파일 레코드 형식

로그 파일의 각 레코드 형식은 다음과 같습니다.

```
Date Time DBPartitionNum RecType Message
```

Date 및 *Time* 필드의 형식은 *yyyy-mm-dd-hh.mm.ss*입니다. 이 필드에서 정렬을 실행하여 각 데이터베이스 파티션의 로그 파일을 병합할 수 있습니다. *DBPartitionNum* 필드에는 조정자가 실행 중인 데이터베이스 파티션 수가 있습니다.

로그에 기록 중인 레코드의 유형에 따라 *RecType* 필드 값이 달라집니다. 기록될 수 있는 값은 다음과 같습니다.

- ACCOUNT: 응용프로그램 어카운팅 통계
- ERROR: 오류가 발생함
- FORCE: 응용프로그램이 강제 실행됨
- NICE: 응용프로그램의 우선순위가 변경됨
- READCFG: 조정자가 구성 파일을 읽음
- SCHEDGRP: 에이전트 우선순위 변경이 발생함
- START: 조정자가 시작됨
- STOP: 조정자가 중지됨
- WARNING: 경고 발생

이러한 일부 값에 대한 자세한 설명은 아래를 참조하십시오.

ACCOUNT

다음과 같은 경우에 ACCOUNT 레코드가 기록됩니다.

- 응용프로그램의 **agent_usr_cpu_time** 또는 **agent_sys_cpu_time** 모니터 요소 값이 이 응용프로그램의 마지막 ACCOUNT 레코드가 기록된 이후로 변경된 경우
- 응용프로그램이 더 이상 활성 상태가 아닌 경우

ACCOUNT 레코드 형식은 다음과 같습니다.

```
<auth_id> <appl_id> <applname> <connect_time> <agent_usr_cpu_delta> <agent_sys_cpu_delta>
```

ERROR

조정자 디먼을 종료해야 하는 경우 ERROR 레코드가 기록됩니다.

FORCE

조정자 구성 파일의 규칙에 따라 조정자가 응용프로그램을 강제로 실행하면 FORCE 레코드가 기록됩니다. FORCE 레코드 형식은 다음과 같습니다.

```
<appl_name> <auth_id> <appl_id> <coord_partition> <cfg_line>
  <restriction_exceeded>
```

각 항목에 대한 설명은 다음과 같습니다.

coord_partition

응용프로그램의 코디네이터 데이터베이스 파티션 수를 지정합니다.

cfg_line

응용프로그램을 강제 실행하는 규칙이 있는 조정자 구성 파일의 라인 번호를 지정합니다.

restriction_exceeded

규칙이 어떻게 위반되었는지에 대한 세부사항을 제공합니다. 유효한 값은 다음과 같습니다.

- CPU: 전체 응용프로그램 USR CPU와 SYS CPU 시간의 합계(초)
- Locks: 응용프로그램이 보유한 전체 잠금 수
- Rowssel: 응용프로그램이 선택한 전체 행 수
- Rowsread: 응용프로그램이 읽은 전체 행 수
- Idle: 응용프로그램이 유휴 상태인 시간
- ET: 응용프로그램의 현재 작업 단위가 시작된 이후 경과 시간 (`uowtime setlimit`이 초과됨)

NICE 조정자가 조정자 구성 파일의 규칙에 따라 응용프로그램의 우선순위를 변경하면 NICE 레코드가 기록됩니다. NICE 레코드 형식은 다음과 같습니다.

```
<appl_name> <auth_id> <appl_id> <nice_value> <cfg_line>
  <restriction_exceeded>
```

각 항목에 대한 설명은 다음과 같습니다.

nice_value

응용프로그램의 에이전트 프로세스에 대한 우선순위 값에 적용되는 증분 또는 감분을 지정합니다.

cfg_line

응용프로그램 우선순위를 변경하는 규칙이 있는 조정자 구성 파일의 라인 번호를 지정합니다.

restriction_exceeded

규칙이 어떻게 위반되었는지에 대한 세부사항을 제공합니다. 유효한 값은 다음과 같습니다.

- CPU: 전체 응용프로그램 USR CPU와 SYS CPU 시간의 합계(초)
- Locks: 응용프로그램이 보유한 전체 잠금 수
- Rowssel: 응용프로그램이 선택한 전체 행 수
- Rowsread: 응용프로그램이 읽은 전체 행 수
- Idle: 응용프로그램이 유휴 상태인 시간
- ET: 응용프로그램의 현재 작업 단위가 시작된 이후 경과 시간 (uowtime setlimit이 초과됨)

SCHEDGRP

응용프로그램이 스케줄링 그룹에 추가되거나 하나의 스케줄링 그룹에서 다른 스케줄링 그룹으로 응용프로그램이 이동되는 경우 SCHEDGRP 레코드가 기록됩니다. SCHEDGRP 레코드 형식은 다음과 같습니다.

```
<appl_name> <auth_id> <appl_id> <cfg_line> <restriction_exceeded>
```

각 항목에 대한 설명은 다음과 같습니다.

cfg_line

응용프로그램을 스케줄링하는 규칙이 있는 조정자 구성 파일의 라인 번호를 지정합니다.

restriction_exceeded

규칙이 어떻게 위반되었는지에 대한 세부사항을 제공합니다. 유효한 값은 다음과 같습니다.

- CPU: 전체 응용프로그램 USR CPU와 SYS CPU 시간의 합계(초)
- Locks: 응용프로그램이 보유한 전체 잠금 수
- Rowssel: 응용프로그램이 선택한 전체 행 수
- Rowsread: 응용프로그램이 읽은 전체 행 수
- Idle: 응용프로그램이 유휴 상태인 시간
- ET: 응용프로그램의 현재 작업 단위가 시작된 이후 경과 시간 (uowtime setlimit이 초과됨)

START

조정자가 시작되면 START 레코드가 기록됩니다. START 레코드 형식은 다음과 같습니다.

```
Database = <database_name>
```

STOP 조정자가 중지되면 STOP 레코드가 기록됩니다. 형식은 다음과 같습니다.

```
Database = <database_name>
```

WARNING

다음과 같은 경우에 WARNING 레코드가 기록됩니다.

- sqlefrce API를 호출하여 응용프로그램을 강제 실행했지만 양수 SQLCODE를 리턴한 경우
- 스냅샷 호출이 1611(SQL1611W)이 아닌 양수 SQLCODE를 리턴한 경우
- 스냅샷 호출이 -1224(SQL1224N) 또는 -1032(SQL1032N)가 아닌 음수 SQLCODE를 리턴한 경우. 이러한 리턴 코드는 이전 활성 인스턴스가 중지된 경우에 발생합니다.
- UNIX 기반 환경에서 신호 핸들러를 설치하려는 시도가 실패했습니다.

표준 값이 기록되었으므로 로그 파일에서 여러 가지 유형의 조치를 쿼리할 수 있습니다. *Message* 필드는 레코드 유형에 따라 다른 기타 비표준 정보를 제공합니다. 예를 들어, FORCE 또는 NICE 레코드의 *Message* 필드에는 응용프로그램 정보가 들어 있지만 ERROR 레코드에는 오류 메시지가 있습니다.

조정자 로그 파일은 다음 예와 비슷합니다.

```
2007-12-11-14.54.52    0 START      Database = TQTEST
2007-12-11-14.54.52    0 READCFG    Config = /u/db2instance/sqllib/tqtest.cfg
2007-12-11-14.54.53    0 ERROR      SQLMON Error: SQLCode = -1032
2007-12-11-14.54.54    0 ERROR      SQLMONSZ Error: SQLCode = -1032
```

제 3 장 성능에 영향을 주는 요소

시스템 아키텍처

DB2 아키텍처 및 프로세스 개요

클라이언트 측에서 로컬 또는 리모트 응용프로그램은 DB2 클라이언트 라이브러리와 링크됩니다. 로컬 클라이언트는 공유 메모리 및 세마포어를 사용하여 통신하고 리모트 클라이언트는 Named Pipes(NPIPE) 또는 TCP/IP와 같은 프로토콜을 사용합니다. 서버 측에서 활동은 EDU(Engine Dispatchable Unit)로 제어합니다.

그림 3에서는 DB2 아키텍처 및 프로세스의 일반 개요를 제공합니다.

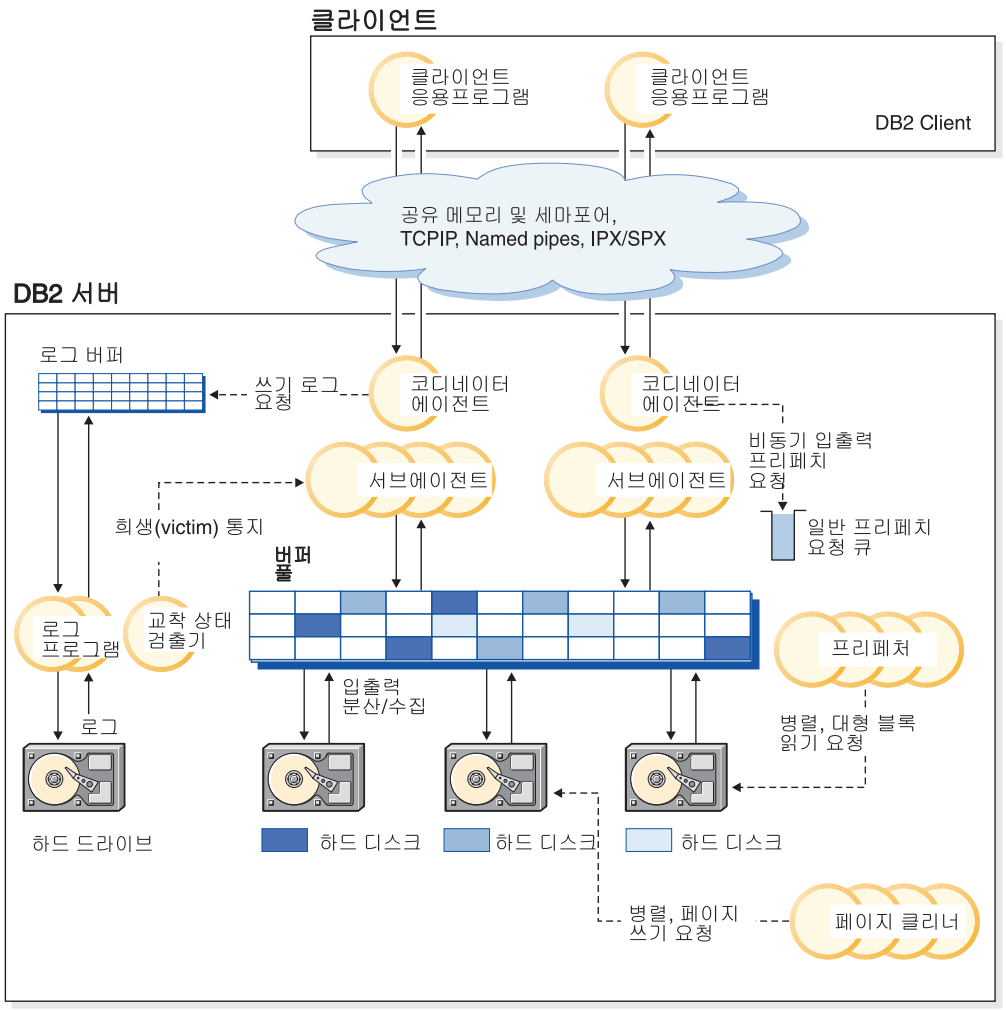


그림 3. 클라이언트 연결 및 데이터베이스 서버 구성요소

EDU는 원 또는 원 그룹으로 표시됩니다.

EDU는 모든 플랫폼에서 스레드로 구현됩니다. DB2 에이전트는 가장 일반적으로 EDU 유형입니다. 이러한 에이전트는 응용프로그램 대신 대부분의 SQL 및 XQuery 처리를 수행합니다. 기타 일반 EDU로는 프리페처 및 페이지 클리너가 있습니다.

클라이언트 응용프로그램 요청을 처리하도록 서브에이전트 세트를 지정할 수 있습니다. 서버가 상주하는 머신에 여러 프로세서가 있거나 파티션된 데이터베이스 환경의 일부분인 경우 여러 서브에이전트를 지정할 수 있습니다. 예를 들어, 대칭 멀티프로세싱(SMP) 환경에서 여러 SMP 서브에이전트가 여러 프로세서를 이용할 수 있습니다.

모든 에이전트 및 서브에이전트는 EDU의 작성 및 파괴를 최소화하는 풀링 알고리즘으로 관리합니다.

버퍼 풀은 사용자 데이터, 인덱스 데이터 및 카탈로그 데이터의 페이지를 임시로 이동하여 수정할 수 있는 데이터베이스 서버 메모리의 영역입니다. 데이터는 디스크보다 메모리에서 훨씬 신속하게 액세스할 수 있으므로 버퍼 풀은 데이터베이스 성능의 중요한 결정자입니다.

버퍼 풀과 프리페처 및 페이지 클리너 EDU의 구성은 응용프로그램이 데이터에 액세스할 수 있는 속도를 제어합니다.

- *프리페처*는 디스크에서 데이터를 검색하고 응용프로그램에 데이터가 필요하기 전에 버퍼 풀로 이동합니다. 예를 들어, 대량의 데이터를 스캔해야 하는 응용프로그램은 데이터 프리페처가 없는 경우 데이터가 디스크에서 버퍼 풀로 이동될 때까지 대기해야 합니다. 응용프로그램의 에이전트는 일반 프리페치 큐로 비동기 미리 읽기 요청을 보냅니다. 프리페처가 사용 가능해지면 디스크에서 버퍼 풀로 요청된 페이지를 가져오기 위해 분산 읽기 입력 조작을 사용하여 이러한 요청을 구현합니다. 데이터 저장을 위해 여러 디스크가 있는 경우 이러한 디스크로 데이터를 스트라이프할 수 있습니다. 스트라이핑을 사용할 경우 프리페처는 여러 디스크를 사용하여 데이터를 동시에 검색할 수 있습니다.
- *페이지 클리너*는 버퍼 풀에서 디스크로 다시 데이터를 이동합니다. 페이지 클리너는 응용프로그램 에이전트와 관계가 없는 백그라운드 EDU입니다. 수정된 페이지를 찾고 이와 같이 변경된 페에지를 디스크로 기록합니다. 페이지 클리너를 사용할 경우 버퍼 풀에는 프리페처가 검색 중인 페이지를 위한 공간이 있습니다.

독립 프리페처 및 페이지 클리너 EDU를 사용하지 않는 경우 응용프로그램 에이전트는 버퍼 풀과 디스크 스토리지 사이에서 모든 데이터 읽기 및 쓰기를 수행해야 합니다.

DB2 프로세스 모델

DB2 프로세스 모델에 대한 지식은 데이터베이스 관리 프로그램 및 연관된 구성요소가 상호작용하는 방식을 이해하는 데 도움이 되며 발생할 수 있는 문제점을 해결하는 데 도움이 될 수 있습니다.

모든 DB2 데이터베이스에서 사용하는 프로세스 모델은 데이터베이스 서버와 클라이언트 간의 통신을 용이하게 합니다. 또한 데이터베이스 응용프로그램이 데이터베이스 제어 블록 및 중요한 데이터베이스 파일과 같은 자원에서 분리됩니다.

DB2 데이터베이스 서버는 데이터베이스 응용프로그램 요청 처리 또는 로그 레코드가 디스크에 기록되는지 확인하는 것과 같은 여러 가지 태스크를 수행해야 합니다. 각 태스크는 일반적으로 별도의 *엔진 디스패치 가능 단위*(EDU)에서 수행합니다.

DB2 데이터베이스 서버에 멀티스레드 아키텍처를 사용할 경우 여러 가지 장점이 있습니다. 새 스레드는 프로세스보다 메모리와 운영 체제 자원이 적게 필요합니다. 일부 운영 체제 자원은 동일한 프로세스의 모든 스레드가 공유할 수 있습니다. 또한 일부 플랫폼에서 스레드의 컨텍스트 전환 시간이 프로세스의 경우보다 적게 걸리므로 성능이 향상될 수 있습니다. 모든 플랫폼에서 스레드 모델을 사용하면 DB2 데이터베이스 서버를 구성하기 쉬워집니다. 필요할 때 추가 EDU를 쉽게 할당할 수 있으며 여러 EDU가 공유해야 하는 메모리를 동적으로 할당할 수 있기 때문입니다.

액세스하는 각 데이터베이스마다 프리페치, 통신 및 로깅과 같은 다양한 데이터베이스 태스크를 처리하도록 개별 EDU가 시작됩니다. 데이터베이스 에이전트는 데이터베이스에 대한 응용프로그램 요청을 처리하기 위해 작성되는 EDU의 특수 클래스입니다.

일반적으로 DB2 데이터베이스 서버에 의존하여 EDU 세트를 관리할 수 있습니다. 그러나 EDU를 검색하는 DB2 도구가 있습니다. 예를 들어, db2pd 명령을 **-edus** 옵션과 함께 사용하여 활성화된 EDU 스레드를 모두 나열할 수 있습니다.

각 클라이언트 응용프로그램 연결에는 데이터베이스에서 작동하는 단일 코디네이터 에이전트가 있습니다. *코디네이터 에이전트*는 응용프로그램 대신 작동되고 필요에 따라 개인용 메모리, 프로세스간 통신(IPC) 또는 리모트 통신 프로토콜을 사용하여 다른 에이전트와 통신합니다.

DB2 아키텍처는 응용프로그램을 DB2 데이터베이스 서버와 다른 주소 스페이스에서 실행하도록 방화벽을 제공합니다(38 페이지의 그림 4). 방화벽은 응용프로그램, 스토어드 프로시저 및 사용자 정의 함수(UDF)로부터 데이터베이스 및 데이터베이스 관리 프로그램을 보호합니다. 방화벽은 응용프로그램 프로그래밍 오류가 내부 버퍼 또는 데이터베이스 관리 프로그램 파일을 겹쳐쓰지 못하도록 예방하므로 데이터베이스에서 데이터의 무결성을 유지합니다. 또한 응용프로그램 오류가 데이터베이스 관리 프로그램을 파손할 수 없으므로 방화벽은 신뢰성을 향상시킵니다.

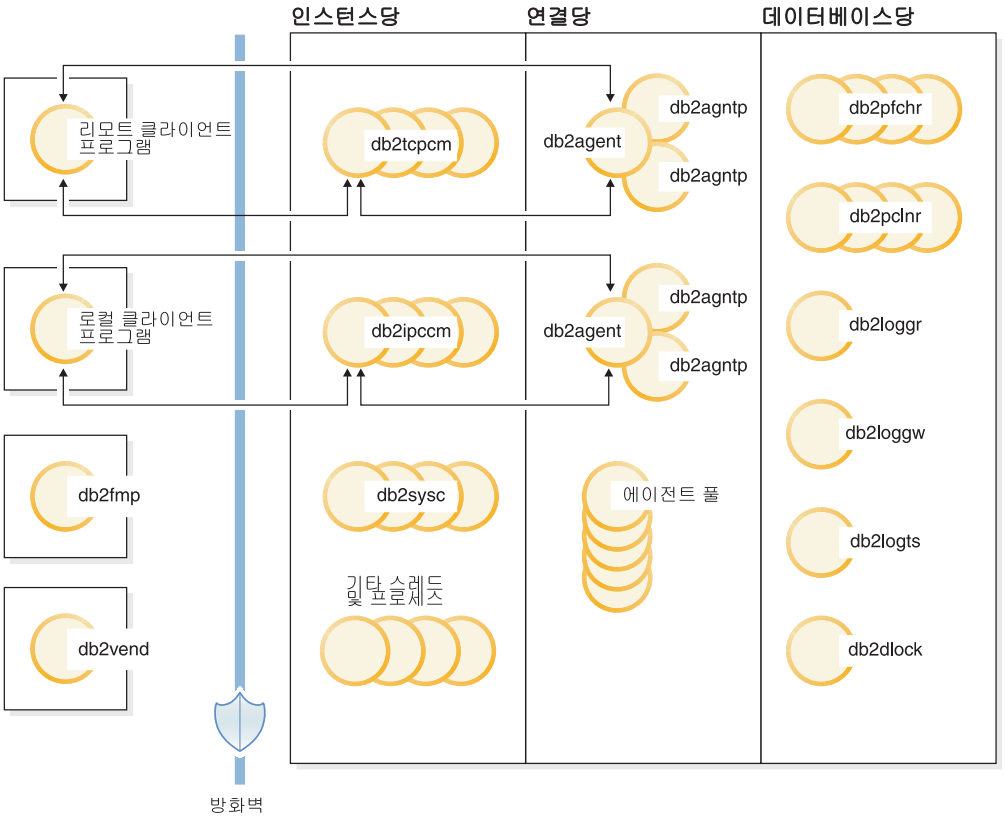


그림 4. DB2 데이터베이스 시스템의 프로세스 모델

클라이언트 프로그램

클라이언트 프로그램은 리모트 또는 로컬이며 데이터베이스 서버와 동일한 머신에서 실행됩니다. 클라이언트 프로그램은 통신 리스너를 통해 데이터베이스에 먼저 접속합니다.

리스너

DB2 데이터베이스 서버가 시작되면 통신 리스너가 시작됩니다. 구성된 각 통신 프로토콜의 리스너와 로컬 클라이언트 프로그램의 프로세스간 통신(IPC) 리스너(db2ipccm)가 있습니다. 리스너는 다음과 같습니다.

- db2ipccm: 로컬 클라이언트 연결용
- db2tcpcm: TCP/IP 연결용
- db2tcpdm: TCP/IP 감지 도구 요청용

에이전트

로컬 또는 리모트 클라이언트 프로그램(응용프로그램)으로부터의 모든 연결 요청에는 해당 코디네이터 에이전트(db2agent)가 할당됩니다. 코디네이터 에이전트가 작성되면 응용프로그램 대신 모든 데이터베이스 요청을 수행합니다.

파티션된 데이터베이스 환경 또는 *쿼리간 병렬 처리*를 사용할 수 있는 시스템에서 코디네이터 에이전트는 데이터베이스 요청을 서브에이전트(각각 db2agntp 및 db2agnts)로 분배합니다. 응용프로그램과 연관되어 있지만 현재 유휴 상태인 서브에이전트의 이름은 db2agnta입니다.

코디네이터 에이전트는 다음을 수행할 수 있습니다.

- 별명을 사용하여 데이터베이스에 연결될 수 있습니다. 예를 들어, db2agent (DATA1)가 데이터베이스 별명 DATA1에 연결됩니다.
- 인스턴스에 접속될 수 있습니다. 예를 들어, db2agent (user1)가 인스턴스 user1에 접속됩니다.

DB2 데이터베이스 서버는 특정 조작을 실행하도록 독립 코디네이터 에이전트 또는 서브코디네이터 에이전트와 같은 기타 에이전트 유형을 인스턴스화합니다. 예를 들어, 독립 코디네이터 에이전트 db2agnti는 이벤트 모니터를 실행하는 데 사용되고 서브코디네이터 에이전트 db2agnsc는 비정상 종료 후 데이터베이스 재시작 조작을 병렬 처리하는 데 사용합니다.

유휴 에이전트는 에이전트 풀에 상주합니다. 이러한 에이전트는 클라이언트 프로그램 대신 작동되는 코디네이터 에이전트 또는 기존 코디네이터 에이전트 대신 작동되는 서브에이전트의 요청에 사용할 수 있습니다. 적합한 *크기*의 유휴 에이전트 풀이 있으면 상당한 응용프로그램 *워크로드*가 있을 때 성능이 향상될 수 있습니다. 이 경우 유휴 에이전트를 필요하면 즉시 사용할 수 있으며 각 응용프로그램 연결마다 완전히 새로운 에이전트를 할당할 필요가 없습니다. 이와 같이 할당하려면 스레드를 작성하고 메모리 및 기타 자원을 할당 및 초기화해야 합니다. DB2 데이터베이스 서버가 유휴 에이전트 풀의 *크기*를 자동으로 관리합니다.

db2fmp

분리 모드 프로세스는 방화벽 외부에서 스토어드 프로시저 및 사용자 정의 함수(UDF)의 실행을 담당합니다. db2fmp 프로세스는 항상 별도의 프로세스이지만 실행하는 루틴의 유형에 따라 멀티스레드일 수 있습니다.

db2vend

EDU 대신 벤더 코드를 실행하는 프로세스입니다. 예를 들어, 로그 아카이브의 User Exit 프로그램을 실행합니다(UNIX 전용).

데이터베이스 EDU

다음의 목록은 각 데이터베이스에서 사용하는 중요한 EDU 중 일부입니다.

- db2pfchr: 버퍼 풀 프리페치용
- db2pclnr: 버퍼 풀 페이지 클리너용

- db2loggr: 트랜잭션 처리 및 복구 처리를 위한 로그 파일 유지보수용
- db2loggw: 로그 파일에 로그 레코드 쓰기용
- db2logts: 어느 테이블 스페이스에 어느 로그 파일의 로그 레코드가 있는지 트래킹. 이 정보는 데이터베이스 디렉토리의 DB2TSCHG.HIS 파일에 기록됩니다.
- db2dlock: 교착 상태 감지용. 파티션된 데이터베이스 환경에서 추가 스레드(db2glock)는 각 파티션의 db2dlock EDU에서 수집하는 정보를 조정하는 데 사용됩니다. db2glock은 카탈로그 파티션에서만 실행됩니다.
- db2stmm: 자체 조정 메모리 관리 기능용
- db2taskd: 백그라운드 데이터베이스 태스크 분산용. 이러한 태스크는 db2taskp라는 스레드가 실행합니다.
- db2hadrp: 고가용성 재해 복구(HADR) 기본 서버 스레드
- db2hadrs: HADR 대기 서버 스레드
- db2lfr: 개별 로그 파일을 처리하는 로그 파일 판독기용
- db2shred: 로그 페이지의 개별 로그 레코드 처리용
- db2redom: 재실행 마스터용. 복구 중 재실행 로그 레코드를 처리하고 재실행 작업자에게 처리할 로그 레코드를 지정합니다.
- db2redow: 재실행 작업자용. 복구 중 재실행 마스터의 요청에 따라 재실행 로그 레코드를 처리합니다.
- db2logmgr: 로그 관리 프로그램용. 복구 가능한 데이터베이스의 로그 파일을 관리합니다.
- db2wlmd: 워크로드 관리 통계의 자동 수집용
- 이벤트 모니터는 다음과 같이 식별됩니다.
  - db2evm*%1%2*(*%3*)

    여기서 *%1*은 다음과 같습니다.

    - g - 전역 파일 이벤트 모니터
    - gp - 전역 파이프 이벤트 모니터
    - l - 로컬 파일 이벤트 모니터
    - lp - 로컬 파이프 이벤트 모니터
    - t - 테이블 이벤트 모니터

    *%2*는 다음과 같습니다.

    - i - 코디네이터
    - p - 코디네이터가 아님

    *%3*은 이벤트 모니터 이름입니다.
- 백업 및 리스토어 스레드는 다음과 같이 식별됩니다.

– db2bm.*%1*.*%2*(백업 및 리스토어 버퍼 조작기) 및 db2med.*%1*.*%2*(백업 및 리스토어 미디어 제어기). 여기서,
  - *%1*은 백업 또는 리스토어 세션을 제어하는 에이전트의 EDU ID입니다.
  - *%2*는 특정 백업 또는 리스토어 세션에 속한 스레드(여러 개일 수 있음)를 구별하는 데 사용되는 순차 값입니다.

  예를 들어, **db2bm.13579.2**는 EDU ID가 13579인 db2agent 스레드에서 제어하는 두 번째 db2bm 스레드를 식별합니다.

데이터베이스 서버 스레드 및 프로세스

데이터베이스 서버를 작동할 경우 시스템 제어기(UNIX의 경우 db2sysc 및 Windows 운영 체제의 경우 db2syscs.exe)가 있어야 합니다. 다음의 스레드 및 프로세스가 다양한 태스크를 수행합니다.

- db2resync: 전역 재동기화 목록을 스캔하는 재동기화 에이전트
- db2wdog: 비정상 종료를 처리하는 UNIX 및 Linux 운영 체제의 감시기
- db2fcms: FCM(Fast Communication Manager) 전송자 디먼
- db2fcmr: FCM(Fast Communication Manager) 수신자 디먼
- db2pdbc: 리모트 데이터베이스 파티션의 병렬 요청을 처리하는 병렬 시스템 제어기(파티션된 데이터베이스 환경에서만 사용됨)
- db2cart: 로그 파일 아카이브용(**userexit** 데이터베이스 구성 매개변수를 사용할 수 있는 경우)
- db2fmtlg: 로그 파일 포맷팅용(**logretain** 데이터베이스 구성 매개변수를 사용할 수 있고 **userexit** 데이터베이스 구성 매개변수를 사용할 수 없는 경우)
- db2panic: 특정 데이터베이스 파티션에서 에이전트 한계에 도달한 후 긴급 요청을 처리하는 심각한 에이전트(파티션된 데이터베이스 환경에서만 사용됨)
- DB2 for z/OS®와 같은 시스템의 주소 목록을 관리하는 db2srvlst
- 결함 모니터 디먼 db2fmd
- 클라이언트 연결 집중기 디스패처 db2disp
- db2acd: Health Monitor 및 자동 유지보수 유틸리티 및 관리 태스크 스케줄러를 호스트하는 자동 컴퓨팅 디먼. 이 프로세스를 이전에는 db2hmon이라고 했습니다.
- db2licc: 설치된 DB2 라이센스를 관리
- db2thcln: EDU가 종료되면 자원을 재활용(UNIX 전용)
- db2aiothr: 데이터베이스 파티션의 비동기 입출력 요청을 관리(UNIX 전용)
- db2alarm: 요청한 타이머가 만기된 경우 EDU에 통지(UNIX 전용)
- db2sysc: 주 시스템 제어기 EDU. 중요한 DB2 서버 이벤트를 처리합니다.

데이터베이스 에이전트

응용프로그램이 데이터베이스에 액세스할 때 여러 프로세스 또는 스레드는 다양한 응용프로그램 태스크를 수행하기 시작합니다. 이러한 태스크로는 로깅, 통신 및 프리페치가 있습니다. 데이터베이스 에이전트는 응용프로그램 요청에 대한 서비스를 제공하는 데 사용되는 데이터베이스 관리 프로그램의 스레드입니다. 버전 9.5에서 에이전트는 모든 플랫폼에서 스레드로 실행됩니다.

최대 응용프로그램 연결 수는 **max_connections** 데이터베이스 관리 프로그램 구성 매개변수로 제어합니다. 각 응용프로그램 연결의 작업은 단일 작업자 에이전트에서 조정합니다. *작업자 에이전트*는 응용프로그램 요청을 수행하지만 특정 응용프로그램에 영구적으로 접속되지 않습니다. *코디네이터 에이전트*는 응용프로그램의 연결이 끊어질 때까지 계속 접속되어 있으므로 응용프로그램과 가장 오래된 연관을 나타냅니다. 엔진 집중기가 사용 가능한 경우에만 이 규칙이 적용되지 않습니다. 이 경우 코디네이터 에이전트는 트랜잭션 경계에서 해당 연관을 종료할 수 있습니다(COMMIT 또는 ROLLBACK).

세 가지 유형의 작업자 에이전트가 있습니다.

- 유휴 에이전트

  가장 간단한 형식의 작업자 에이전트입니다. 아웃바운드 연결이 없으며 로컬 데이터베이스 연결 또는 인스턴스 접속이 없습니다.

- 활성 코디네이터 에이전트

  클라이언트 응용프로그램의 각 데이터베이스 연결에는 데이터베이스에서 작업을 조정하는 단일 활성 에이전트가 있습니다. 코디네이터 에이전트가 작성된 후 응용프로그램 대신 모든 데이터베이스 요청을 수행하고 프로세스간 통신(IPC) 또는 리모트 통신 프로토콜을 사용하여 다른 에이전트와 통신합니다. 각 에이전트는 자체 전용 메모리로 작동되며 버퍼 풀과 같은 데이터베이스 관리 프로그램 및 데이터베이스 전역 자원을 다른 에이전트와 공유합니다. 트랜잭션이 완료되면 활성 코디네이터 에이전트가 비활성 에이전트가 될 수 있습니다. 클라이언트가 데이터베이스에서 연결을 끊거나 인스턴스에서 분리된 경우 코디네이터 에이전트는 다음 상태가 됩니다.
  - 다른 연결이 대기 중인 경우 활성 코디네이터 에이전트가 됩니다.
  - 연결 대기 중이 아니고 최대 풀 에이전트 수를 자동으로 관리 중이거나 이 수에 접근하지 않은 경우 사용 가능해지고 유휴 상태로 표시됩니다.
  - 연결 대기 중이 아니고 최대 풀 에이전트 수에 접근한 경우 종료되고 스토리지가 사용 가능해집니다.

- 서브에이전트

  코디네이터 에이전트는 서브에이전트로 데이터베이스 요청을 분산하고 이러한 서브에이전트는 응용프로그램에 대한 요청을 수행합니다. 코디네이터 에이전트가 작성된 후 데이터베이스에 대해 요청을 수행하는 서브에이전트를 조정하여 응용프로그램 대신

모든 데이터베이스 요청을 처리합니다. DB2 버전 9.5에서 서브에이전트는 파티션되지 않은 환경과 쿼리 내 병렬 처리가 사용 불가능한 환경에도 서브에이전트가 존재할 수 있습니다.

응용프로그램의 작업을 수행하지 않거나 지정 대기 중인 에이전트는 유휴 에이전트로 간주되며 *에이전트 풀*에 상주합니다. 이러한 에이전트는 클라이언트 프로그램 대신 작동되는 코디네이터 에이전트의 요청 또는 기존 코디네이터 에이전트 대신 작동되는 서브에이전트에 사용할 수 있습니다. 사용 가능한 에이전트 수는 **num_poolagents** 데이터베이스 관리 프로그램 구성 매개변수의 값에 따라 다릅니다.

에이전트가 필요할 때 유휴 에이전트가 없는 경우 새 에이전트가 자동으로 작성됩니다. 새 에이전트를 작성하는 데 일정한 양의 오버헤드가 필요하므로 클라이언트의 유휴 에이전트를 활성화할 수 있는 경우 CONNECT 및 ATTACH 성능이 향상됩니다.

서브에이전트가 응용프로그램의 작업을 수행하는 경우 해당 응용프로그램과 연관되어 있습니다. 지정된 작업을 완료한 후 에이전트 풀에 배치할 수 있지만 계속 원래의 응용프로그램과 연관되어 있습니다. 응용프로그램이 추가 작업을 요청하는 경우 데이터베이스 관리 프로그램은 새 에이전트를 작성하기 전에 먼저 유휴 풀에서 연관된 에이전트를 점검합니다.

데이터베이스 에이전트 관리

대부분의 응용프로그램은 연결된 응용프로그램 수와 데이터베이스 서버가 처리할 수 있는 응용프로그램 요청 수 사이에 일대일 관계를 설정합니다. 그러나 사용자의 환경에서는 연결된 응용프로그램 수와 처리할 수 있는 응용프로그램 요청 수 사이에 다대일 관계가 필요할 수도 있습니다.

두 개의 데이터베이스 관리 프로그램 구성 매개변수는 이러한 요인을 개별적으로 제어합니다.

- **max_connections** 매개변수는 최대 연결 응용프로그램 수를 지정합니다.
- **max_coordagents** 매개변수는 동시에 처리할 수 있는 최대 응용프로그램 요청 수를 지정합니다.

max_connections의 값이 **max_coordagents**의 값보다 큰 경우 연결 집중기가 사용 가능합니다. 각 활성 코디네이팅 에이전트에는 전역 데이터베이스 자원 오버헤드가 필요하므로 이러한 에이전트 수가 크면 사용 가능한 전역 자원의 상한에 접근할 기회가 커집니다. 이러한 상태가 발생하지 않도록 예방하려면 **max_connections**의 값을 **max_coordagents**의 값보다 높게 설정하거나 두 매개변수를 모두 AUTOMATIC으로 설정하십시오.

이러한 매개변수를 AUTOMATIC으로 설정하도록 권장되는 두 가지 특정 시나리오가 있습니다.

- 시스템이 필요한 모든 연결을 처리할 수 있다는 점을 확신하지만 사용되는 전역 자원의 양을 제한하려면(코디네이팅 에이전트의 수 제한) **max_connections**를 AUTOMATIC으로 설정하십시오. **max_connections**가 **max_coordagents**보다 큰 경우 연결 집중기를 사용할 수 있습니다.
- 최대 연결 또는 코디네이팅 에이전트 수를 제한하지 않지만 시스템이 연결된 응용프로그램 수와 처리되는 응용프로그램 요청 수 사이의 다대일 관계를 요구하거나 이러한 관계를 이용하면 유리할 경우 두 매개변수를 모두 AUTOMATIC으로 설정하십시오. 두 매개변수가 모두 AUTOMATIC으로 설정된 경우 데이터베이스 관리 프로그램은 코디네이팅 에이전트에 대한 연결의 이상적 비율로 지정하는 값을 사용합니다. 이 두 매개변수를 모두 시작 값 및 AUTOMATIC 설정을 사용하여 구성할 수 있다는 점을 참고하십시오. 예를 들어, 다음의 명령은 값 200 및 AUTOMATIC을 **max_coordagents** 매개변수와 연관시킵니다.
  ```
  update dbm config using max_coordagents 200 automatic.
  ```

예

다음의 시나리오를 참조하십시오.

- **max_connections** 매개변수가 AUTOMATIC으로 설정되며 현재 값이 300입니다.
- **max_coordagents** 매개변수가 AUTOMATIC으로 설정되며 현재 값이 100입니다.

max_connections와 **max_coordagents**의 비율은 300:100입니다. 데이터베이스 관리 프로그램은 연결이 들어오면 새 코디네이팅 에이전트를 작성하고 필요한 경우에만 연결 집중이 적용됩니다. 이러한 설정으로 인해 다음의 조치가 발생합니다.

- 연결 1 - 100은 새 코디네이팅 에이전트를 작성합니다.
- 연결 101 - 300은 새 코디네이팅 에이전트를 작성하지 않습니다. 이미 작성된 100개의 에이전트를 공유합니다.
- 연결 301 - 400은 새 코디네이팅 에이전트를 작성합니다.
- 연결 401 - 600은 새 코디네이팅 에이전트를 작성하지 않습니다. 이미 작성된 200개의 에이전트를 공유합니다.
- 계속 이러한 방식으로 진행됩니다.

이 예에서는 연결된 응용프로그램이 각 단계에서 새 코디네이팅 에이전트의 작성을 보장하는 충분한 작업을 진행하는 것으로 가정합니다. 일정 기간이 경과한 후 연결된 응용프로그램이 더 이상 충분한 작업을 진행하지 않는 경우 코디네이팅 에이전트는 비활성화되고 종료될 수 있습니다.

연결 수가 줄었지만 나머지 연결에서 진행 중인 작업의 양이 많은 경우 코디네이팅 에이전트 수가 즉시 줄어들지 않을 수도 있습니다. **max_connections** 및 **max_coordagents** 매개변수는 에이전트 풀링 또는 에이전트 종료에 직접 영향을 주지 않습니다. 일반 에이전트 종료 규칙이 계속 적용됩니다. 즉, 코디네이팅 에이전트에 대

한 연결 비율은 사용자가 지정한 값과 정확히 일치하지 않을 수도 있습니다. 에이전트가 종료되기 전에 재사용할 수 있도록 에이전트 풀로 반환할 수 있습니다.

세부 제어 단위가 필요한 경우 간단한 비율을 지정하십시오. 예를 들어, 이전 예의 300:100 비율을 3:1로 표시할 수 있습니다. **max_connections**가 3(AUTOMATIC)으로 설정되고 **max_coordagents**가 1(AUTOMATIC)로 설정된 경우 3회 연결 시마다 하나의 코디네이팅 에이전트가 작성될 수 있습니다.

클라이언트-서버 처리 모델

로컬 및 리모트 응용프로그램 프로세스는 모두 동일한 데이터베이스에서 작동할 수 있습니다. 리모트 응용프로그램은 데이터베이스 서버가 상주하는 머신의 리모트 머신에서 데이터베이스 조치를 시작하는 응용프로그램입니다. 로컬 응용프로그램은 서버 머신의 데이터베이스에 직접 접속되어 있습니다.

클라이언트 연결의 관리 방식은 연결 집중기의 설정 여부에 따라 다릅니다.
max_connections 데이터베이스 관리 프로그램 구성 매개변수의 값이 **max_coordagents** 구성 매개변수의 값보다 클 때마다 연결 집중기가 설정됩니다.

- 연결 집중기가 해제된 경우 각 클라이언트 응용프로그램에는 해당 응용프로그램의 처리를 조정하고 통신하는 *코디네이터 에이전트*라고 하는 고유 EDU(Engine Dispatchable Unit)가 지정됩니다.
- 연결 집중기가 설정된 경우 각 코디네이터 에이전트는 여러 클라이언트 연결을 한 번에 하나씩 관리할 수 있으며 다른 작업자 에이전트가 이 작업을 수행하도록 조정할 수 있습니다. 여러 임시 연결을 사용하는 인터넷 응용프로그램이나 비교적 작은 여러 개의 트랜잭션을 사용하는 응용프로그램의 경우 연결 집중기는 여러 개의 추가 클라이언트 응용프로그램을 동시에 연결할 수 있도록 허용하여 성능을 향상시킵니다. 또한 각 연결의 시스템 자원 사용이 줄어듭니다.

46 페이지의 그림 5에서 DB2 서버에 있는 각 원은 운영 체제 스레드를 사용하여 구현된 EDU를 나타냅니다.

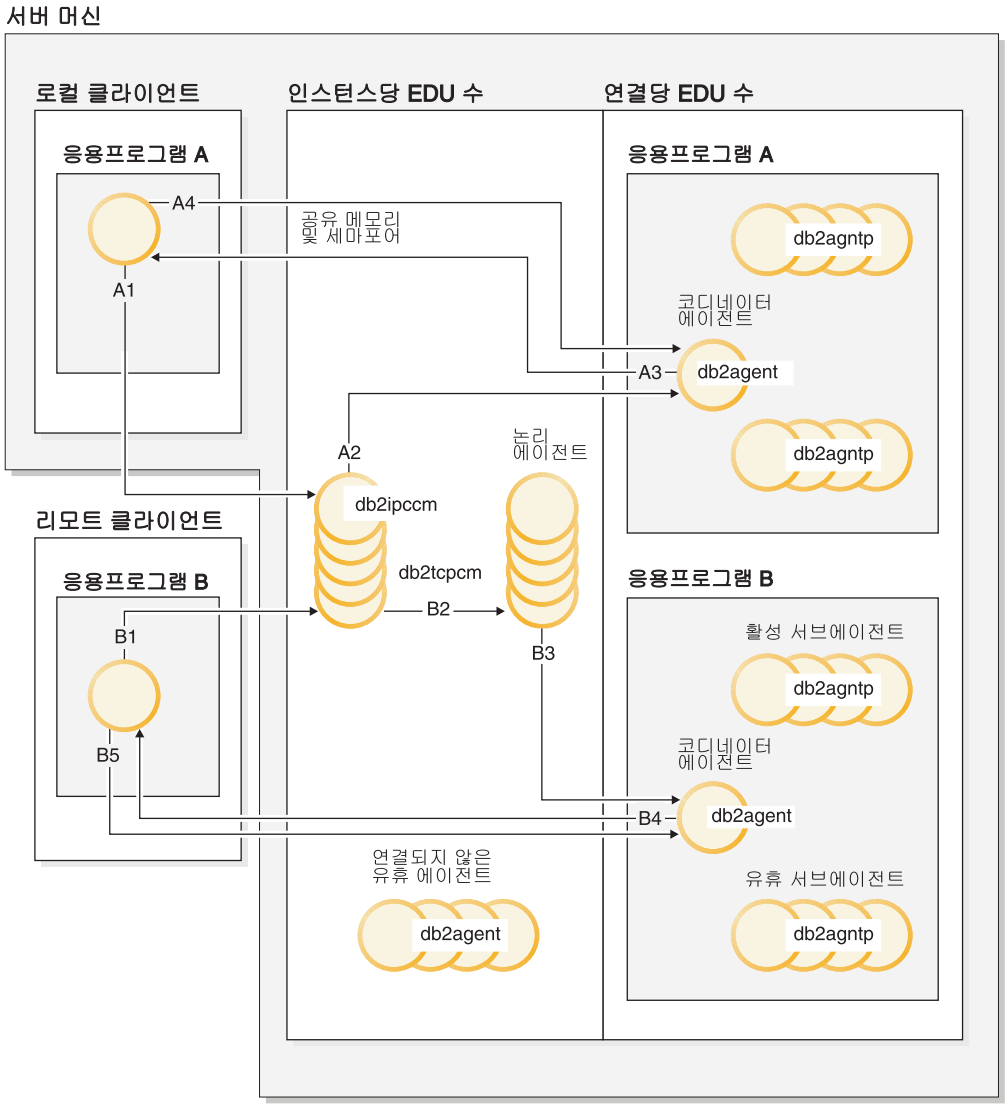


그림 5. 클라이언트-서버 처리 모델 개요

- A1에서 로컬 클라이언트는 db2ipccm을 통해 통신을 설정합니다.
- A2에서 db2ipccm은 로컬 클라이언트의 응용프로그램 요청에 따라 코디네이터 에이전트가 되는 db2agent EDU에 대해 작업합니다.
- A3에서 코디네이터 에이전트는 클라이언트 응용프로그램에 접속하여 클라이언트 응용프로그램과 코디네이터 간 공유 메모리 통신을 설정합니다.
- A4에서 로컬 클라이언트의 응용프로그램이 데이터베이스에 연결합니다.
- B1에서 리모트 클라이언트가 db2tcpcm을 통해 통신을 설정합니다. 다른 통신 프로토콜을 선택한 경우 적합한 통신 관리 프로그램을 사용합니다.
- B2에서 db2tcpcm은 응용프로그램의 코디네이터 에이전트가 되는 db2agent EDU에 대해 작업하고 이 에이전트로 연결을 전달합니다.
- B4에서 코디네이터 에이전트는 리모트 클라이언트 응용프로그램에 접속합니다.

- B5에서 리모트 클라이언트 응용프로그램이 데이터베이스에 연결합니다.

다음의 사항도 참고하십시오.

- 작업자 에이전트가 응용프로그램 요청을 수행합니다. 네 가지 유형의 작업자 에이전트 즉, 활성 코디네이터 에이전트, 활성 서브에이전트, 연관된 서브 에이전트 및 유휴 에이전트가 있습니다.
- 각 클라이언트 연결은 활성 코디네이터 에이전트에 링크됩니다.
- 파티션된 데이터베이스 환경 또는 파티션 내 병렬 처리가 사용 가능한 환경에서 코디네이터 에이전트는 서브에이전트로 데이터베이스 요청을 분산시킵니다(db2agntp).
- 유휴 에이전트가 새 작업을 기다리는 에이전트 풀(db2agent)이 있습니다.
- 다른 EDU는 클라이언트 연결, 로그, 2단계 커미트 조작, 백업 및 리스토어 작업과 기타 태스크를 관리합니다.

그림 6에서는 서버 머신 환경의 일부분인 추가 EDU에 대해 설명합니다. 각 활성 데이터베이스에는 자체의 프리페처(db2pfchr) 공유 풀과 페이지 클리너(db2pclnr) 및 자체 로그 프로그램(db2loggr)과 교착 상태 검출기(db2dlock)가 있습니다.

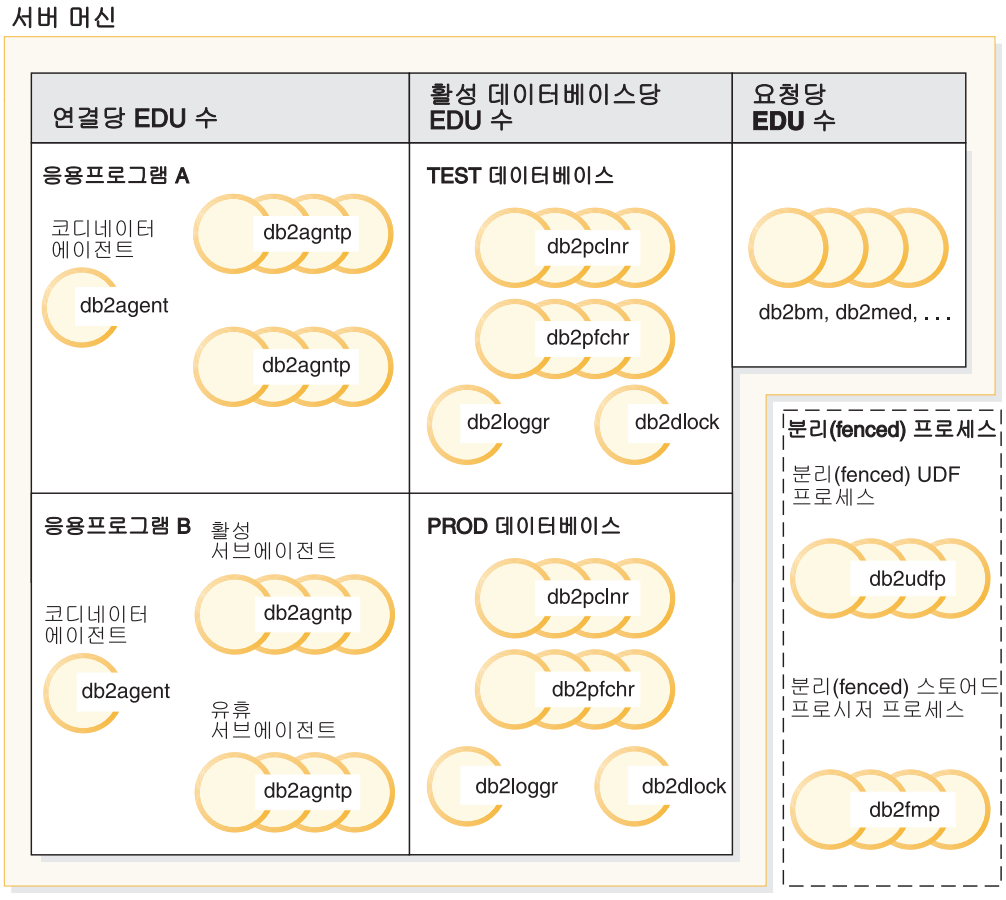


그림 6. 데이터베이스 서버의 EDU

그림에 표시되지 않은 분리(fenced) 사용자 정의 함수(UDF) 및 스토어드 프로시저를 관리하여 작성 및 폐기와 연관된 비용을 최소화합니다. **keepfenced** 데이터베이스 관리 프로그램 구성 매개변수의 디폴트값은 YES이므로 다음 프로시저 호출 시에 스토어드 프로시저 프로세스를 재사용할 수 있습니다.

주: 비분리 UDF 및 스토어드 프로시저는 성능 향상을 위해 에이전트의 주소 스페이스에서 직접 실행됩니다. 그러나 에이전트의 주소 스페이스에 대한 무제한 액세스가 가능하므로 사용하기 전에 엄격하게 테스트해야 합니다.

그림 7에서는 단일 데이터베이스 파티션 처리 모델과 다중 데이터베이스 파티션 처리 모델의 유사점과 차이점에 대해 설명합니다.

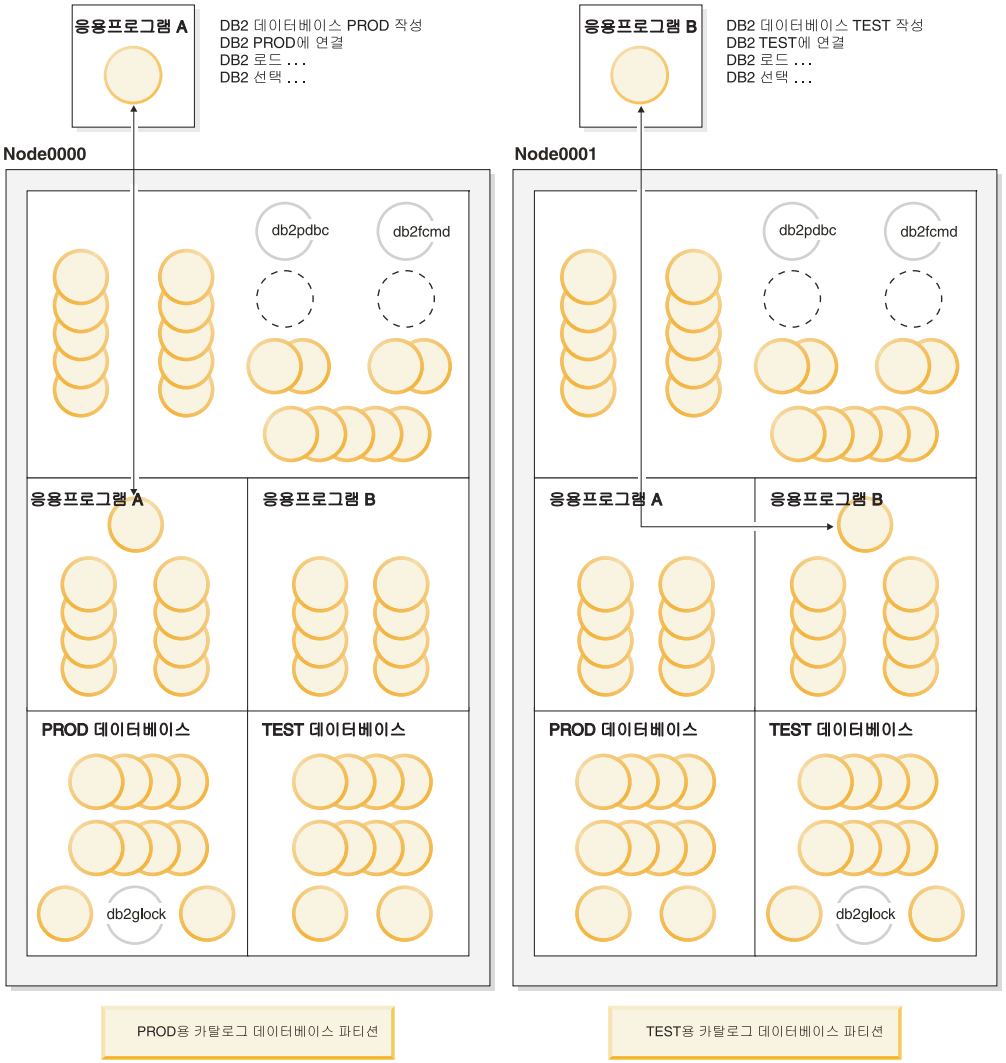


그림 7. 다중 데이터베이스 파티션의 프로세스 모델

다중 데이터베이스 파티션 환경에서 CREATE DATABASE 명령이 발행된 데이터베이스 파티션을 *카탈로그 데이터베이스 파티션*이라고 합니다. 이 데이터베이스 파티션에 시스템 카탈로그 테이블이 저장됩니다. 시스템 카탈로그는 데이터베이스의 오브젝트에 대한 모든 정보의 저장소입니다.

48 페이지의 그림 7에 설명된 바와 같이 A 응용프로그램은 Node0000에서 PROD 데이터베이스를 작성하므로 PROD 데이터베이스의 카탈로그도 이 데이터베이스 파티션에서 작성됩니다. 마찬가지로 B 응용프로그램은 Node0001에서 TEST 데이터베이스를 작성하므로 TEST 데이터베이스의 카탈로그가 이 데이터베이스 파티션에서 작성됩니다. 사용자 환경의 여러 데이터베이스 파티션들에서 각 데이터베이스의 카탈로그와 연관된 추가 활동의 균형을 맞추기 위해 다른 데이터베이스 파티션에서 데이터베이스를 작성하는 것이 좋습니다.

인스턴스와 연관된 추가 EDU(db2pdbc 및 db2fcmd)가 있으며 다중 데이터베이스 파티션 환경의 각 데이터베이스 파티션에서 찾을 수 있습니다. 이러한 EDU는 데이터베이스 파티션에서 요청을 조정하고 FCM(Fast Communication Manager)을 사용 가능하게 할 때 필요합니다.

카탈로그 데이터베이스 파티션과 연관된 추가 EDU(db2glock)가 있습니다. 이 EDU는 활성 데이터베이스가 있는 데이터베이스 파티션에서 전역 교착 상태를 제어합니다.

응용프로그램의 각 연결 요청은 코디네이터 에이전트와 연관된 연결로 표시합니다. *코디네이터 에이전트*는 요청을 받고 응답을 보내어 응용프로그램과 통신하는 에이전트입니다. 요청 자체를 충족시키거나 요청을 수행하기 위해 다중 서브에이전트를 조정할 수 있습니다. 코디네이터 에이전트가 상주하는 데이터베이스 파티션을 해당 응용프로그램의 *코디네이터 데이터베이스 파티션*이라고 합니다.

응용프로그램에서 발행하는 데이터베이스 요청의 일부분은 코디네이터 데이터베이스 파티션에 의해 다른 데이터베이스 파티션의 서브에이전트로 전송됩니다. 모든 결과를 응용프로그램으로 다시 보내기 전에 코디네이터 데이터베이스 파티션에서 통합합니다.

여러 데이터베이스 파티션을 동일한 머신에서 실행하도록 구성할 수 있습니다. 이러한 구성을 *다중 논리적 파티션* 구성이라고 합니다. 이러한 구성은 주기억장치가 매우 큰, 대형 대칭형 멀티프로세서(SMP)에서 매우 유용합니다. 이 환경에서 공유 메모리 및 세마포어를 사용하도록 데이터베이스 파티션들 간 통신을 최적화할 수 있습니다.

클라이언트 연결을 위한 연결 집중기 향상 기능

연결 집중기는 여러 개의 동시 클라이언트를 효율적으로 처리할 수 있도록 하여 자주 사용하지만 비교적 일시적으로 연결하는 응용프로그램의 성능을 향상시킵니다. 또한 각 연결 중 메모리 사용을 줄이고 컨텍스트 전환 수를 줄입니다.

max_connections 데이터베이스 관리 프로그램 구성 매개변수의 값이 **max_coordagents** 구성 매개변수의 값보다 큰 경우 연결 집중기가 사용 가능합니다.

여러 개의 동시 사용자 연결이 필요한 환경에서 시스템 자원을 효율적으로 사용할 수 있도록 연결 집중기를 사용 가능하게 할 수 있습니다. 이 기능은 이전에 DB2 Connect 연결 풀링에서만 발견되었던 장점들을 통합합니다. 첫 번째 연결 이후 연결 집중기는 호스트에 연결하는 데 필요한 시간을 줄입니다. 호스트에서 연결 끊기가 요청된 경우 인바운드 연결이 삭제되지만 호스트에 대한 아웃바운드 연결은 풀에 보존됩니다. 새 연결 요청을 받은 경우 데이터베이스 관리 프로그램은 풀에서 기존 아웃바운드 연결을 재사용하려고 합니다.

연결 풀링 또는 연결 집중기를 사용하는 응용프로그램의 성능을 향상시키려면 캐시되는 데이터의 블록 *크기*를 제어하는 매개변수를 조정하십시오. 자세한 정보는 DB2 Connect 제품 문서를 참조하십시오.

예

- 평균 1000명의 사용자가 동시에 연결되어 있는 단일 파티션 데이터베이스를 참조하십시오. 연결된 사용자 수가 높아지는 경우가 있습니다. 동시 트랜잭션 수로 200을 사용할 수 있지만 250을 초과할 수 없습니다. 트랜잭션은 짧습니다.

  이 워크로드에서 다음의 데이터베이스 관리 프로그램 구성 매개변수를 설정할 수 있습니다.

  – 최대 동시 트랜잭션 수를 지원하도록 **max_coordagents**를 250으로 설정하십시오.

  – 모든 수의 연결을 지원하도록 1000 값을 사용하여 **max_connections**를 AUTOMATIC으로 설정하십시오. 이 예에서 값이 250보다 크면 연결 집중기가 사용 가능합니다.

  – **num_poolagents**는 디폴트값으로 두십시오. 데이터베이스 에이전트가 수신 클라이언트 요청에 대한 서비스를 제공할 수 있으며 새 에이전트의 작성으로 오버헤드가 거의 없어집니다.

- 평균 1000명의 사용자가 동시에 연결되어 있는 단일 파티션 데이터베이스를 참조하십시오. 연결된 사용자 수가 2000에 접근하는 경우가 있습니다. 평균 500명의 사용자가 지정된 시간에 작업을 실행할 예정입니다. 동시 트랜잭션 수는 약 250입니다. 500개의 코디네이팅 에이전트는 일반적으로 너무 많습니다. 1000명의 연결된 사용자의 경우 코디네이팅 에이전트가 250개면 충분합니다.

  이 워크로드의 경우 다음과 같이 데이터베이스 관리 프로그램 구성을 갱신할 수 있습니다.

  ```
  update dbm cfg using max_connections 1000 automatic
  update dbm cfg using max_coordagents 250 automatic
  ```

즉, 연결 수가 1000을 초과하여 증가하면 필요에 따라 추가 코디네이팅 에이전트가 작성되며 최대값은 전체 연결 수에 따라 판별됩니다. 워크로드가 증가하면 데이터베이스 관리 프로그램은 코디네이팅 에이전트에 대한 연결을 비교적 안정적인 비율로 유지하려고 합니다.

- 연결 집중기를 사용 불가능하게 하지만 연결된 사용자 수를 제한하는 경우를 가정하십시오. 예를 들어, 동시에 연결된 사용자 수를 250으로 제한하려면 다음의 데이터베이스 관리 프로그램 구성 매개변수를 설정할 수 있습니다.
  - **max_coordagents**를 250으로 설정하십시오.
  - **max_connections**를 250으로 설정하십시오.
- 연결 집중기를 사용 불가능하게 하고 연결된 사용자 수를 제한하지 않는 경우를 가정하십시오. 다음과 같이 데이터베이스 관리 프로그램 구성을 갱신할 수 있습니다.

```
update dbm cfg using max_connections automatic
update dbm cfg using max_coordagents automatic
```

파티션된 데이터베이스의 에이전트

파티션된 데이터베이스 환경 또는 파티션 내 병렬 처리가 사용 가능한 환경에서 각 데이터베이스 파티션에는 서브에이전트를 작성할 수 있는 자체의 에이전트 풀이 있습니다.

이 풀로 인해 필요할 때 또는 작업을 완료했을 때마다 서브에이전트를 작성 및 파과할 필요가 없습니다. 서브에이전트는 풀에서 연관 에이전트로 계속 남아 있을 수 있으며 데이터베이스 관리 프로그램은 서브에이전트가 연결되는 응용프로그램 또는 새 응용프로그램의 새 요청에 사용할 수 있습니다.

시스템의 성능 및 메모리 사용에 대한 영향은 에이전트 풀이 조정된 방식과 깊은 관련이 있습니다. 에이전트 풀 *크기*의 데이터베이스 관리 프로그램 구성 매개변수(**num_poolagents**)는 데이터베이스 파티션의 응용프로그램과 계속 연관시킬 수 있는 총 에이전트 및 서브에이전트 수에 영향을 줍니다. 풀 *크기*가 너무 작고 풀이 가득 찬 경우 서브에이전트는 작업 중인 응용프로그램과의 연관을 취소하고 종료됩니다. 서브에이전트를 계속 작성하고 응용프로그램과 다시 연관시켜야 하므로 성능이 저하됩니다.

디폴트로 100 값을 사용하여 **num_poolagents**는 AUTOMATIC으로 설정되며 데이터베이스 관리 프로그램은 풀링할 유휴 에이전트 수를 자동으로 관리합니다.

num_poolagents의 값이 수동으로 너무 낮게 설정된 경우 한 응용프로그램이 풀을 연관된 서브에이전트로 가득 채울 수 있습니다. 그런 다음 다른 응용프로그램이 새 서브에이전트를 요구하지만 에이전트 풀에 서브에이전트가 없는 경우 다른 응용프로그램의 에이전트 풀에서 비활성 서브에이전트를 재활용합니다. 이러한 동작으로 자원을 완전히 활용할 수 있습니다.

num_poolagents의 값이 수동으로 너무 높게 설정된 경우 다른 태스크에 사용할 수 없는 데이터베이스 관리 프로그램 자원을 사용하여 오랜 기간 동안 풀에서 계속 사용되지 않습니다.

연결 집중기가 사용 가능한 경우 **num_poolagents**의 값은 한 번에 풀에서 유휴 상태일 수 있는 정확한 에이전트 수를 반드시 반영하지는 않습니다. 높은 워크로드 활동을 처리하기 위해 임시로 에이전트가 필요할 수도 있습니다.

데이터베이스 에이전트 외에 다음과 같은 기타 비동기 데이터베이스 관리 프로그램 활동이 자체 프로세스 또는 스레드로 실행됩니다.

- 데이터베이스 입출력 서버 또는 입출력 프리페처
- 데이터베이스 비동기 페이지 클리너
- 데이터베이스 로그 프로그램
- 데이터베이스 교착 상태 검출기
- 통신 및 IPC 리스너
- 테이블 스페이스 컨테이너 재조정 프로그램

좋은 성능을 위한 구성

InfoSphere™ Balanced Warehouse™(BW)와 같은 DB2 전개의 일부 유형 또는 SAP 시스템 내 유형은 고도로 지정된 구성을 지닙니다.

BW의 경우, CPU의 수, CPU에 대한 메모리 비율, 디스크의 구성 및 수와 같은 하드웨어 인수 및 버전이 최적 구성을 판별하기 위한 철저한 테스트를 기반으로 사전 지정됩니다. SAP의 경우, 하드웨어 구성이 정확하게 지정되지 않지만, 사용 가능한 샘플 구성이 매우 많이 있습니다. 또한 SAP 우수 사례에서는 권장 DB2 구성 설정을 제공합니다. 충분히 테스트된 구성 지침을 제공하는 시스템에 대해 DB2 전개를 사용할 경우, 대개 보다 일반적인 경험 대신에 해당 지침을 활용해야 합니다.

아직 자세한 하드웨어 구성이 없는 제안된 시스템을 고려해보십시오. 목표는 시스템이 좋은 성능을 얻을 수 있게 하는 몇몇 핵심 구성 결정을 식별하는 것입니다. 이 단계는 일반적으로 시스템이 작동되기 전에 발생하므로, 실제로 어떻게 동작할지를 파악하는 데 한계가 있을 수 있습니다. 한편으로는, 시스템이 수행할 내용에 대한 지식을 기반으로 "최상의 추측"을 해야 합니다.

하드웨어 구성

CPU 용량은 성능을 위한 시스템 구성에서 주요 독립 변수 중 하나입니다. 다른 모든 하드웨어 구성이 CPU로부터 흐르기 때문에 주어진 워크로드에 얼마나 많은 CPU 용량이 필요한지 예측하기가 쉽지 않습니다. 비즈니스 인텔리전스(BI) 환경에서는, 프로세서 코어당 200-300GB의 활성 원시 데이터가 적정한 것으로 추정됩니다. 다른 환경의

경우, 하나 이상의 기존 DB2 시스템을 기반으로 필요한 CPU의 양을 측정할 수 있습니다. 예를 들어, 새 시스템에서 각각 최소한 기존 시스템의 SQL만큼 복잡한 SQL을 실행 중인 사용자를 50% 더 많이 처리해야 할 경우, 50% 더 많은 CPU 용량이 필요하다고 가정하는 것이 합당할 것입니다. 마찬가지로, 다른 처리량 요구사항 또는 참조 무결성 또는 트리거 사용의 변경과 같이 CPU 사용의 변경을 예보하는 다른 인수도 고려해야 합니다.

CPU 요구사항을 제대로 파악한 후에(사용 가능한 정보에서 파생된), 하드웨어 구성의 다른 측면이 제자리를 잡기 시작합니다. GB 또는 TB 단위로 필요한 시스템 디스크 용량을 고려해야 하지만, 성능에 대한 가장 중요한 인수는 데이터 전송의 초당 MB 또는 초당 입출력(IOPS) 단위의 용량입니다. 실질적으로, 이것은 관련된 개별 디스크의 수에 의해 결정됩니다.

왜 그런 것일까요? 지난 10년 간 CPU는 속도에서 굉장한 증가를 보이며 진화한 한편, 디스크는 속도보다는 용량 및 비용 면에서 더욱 진화했습니다. 디스크 탐색 시간 및 전송률이 향상되어 왔지만 CPU 속도를 따라가지 못했습니다. 따라서 모던 시스템에서 필요한 총체적 성능을 얻기 위해서는, 특히 상당한 양의 무작위 디스크 입출력을 구동하는 시스템의 경우 복수 디스크를 사용하는 것이 어느 때보다 더 중요합니다. 종종 시스템의 모든 데이터를 저장할 수 있는 큰 용량의 디스크를 최소한으로 사용하고자 할 수도 있지만, 이렇게 하면 일반적으로 성능이 매우 저하됩니다.

RAID 스토리지의 경우, 또는 개별적으로 주소 지정 가능한 드라이브의 경우 프로세서 코어당 적어도 10개에서 20개의 디스크를 구성하는 것이 일반적입니다. 스토리지 서버의 경우, 비슷한 수가 권장되지만 이 경우에는 약간의 주의가 필요합니다. 스토리지 서버에서의 스페이스 할당은 종종 처리량보다는 용량을 더욱 중요시하여 수행됩니다. 논리적으로 별개의 스토리지가 실수로 오버랩되지 않도록 하기 위해 데이터베이스 스토리지의 실제 레이아웃을 이해하는 것은 매우 좋은 생각입니다. 예를 들어, 4-웨이 시스템에 대한 합당한 할당은 각각 8개 드라이브의 8개 배열일 수 있습니다. 그러나 8개 배열 모두에서 동일한 8개의 기본 실제 드라이브를 공유할 경우, 64개의 실제 드라이브에 걸쳐 있는 8개 배열에 비해 구성의 처리량이 크게 감소합니다.

DB2 트랜잭션 로그를 위한 일부 전용(비공유) 디스크를 따로 마련해 두는 것이 좋습니다. 이는, 예를 들어, 로그의 입출력 특성이 DB2 컨테이너와 매우 다르고, 특히 높은 등급의 쓰기 활동이 있는 시스템에서 로그 입출력과 다른 유형의 입출력 간 경쟁으로 인해 로깅 병목 현상이 발생할 수 있기 때문입니다.

일반적으로 디스크의 RAID-1 쌍은 초당 최대 400 적정 쓰기 위주 DB2 트랜잭션에 대해 충분한 로깅 처리량을 제공할 수 있습니다. 처리량 비율이 클수록, 또는 높은 볼륨 로깅(예를 들어, 대량 삽입 동안)일수록 더 큰 로그 처리량을 요구하고, 이는 쓰기-캐싱 디스크 제어기를 통해 시스템에 연결된 RAID-10 구성의 추가 디스크에 의해 제공될 수 있습니다.

CPU와 디스크는 다른 시간 스케일에서 효과적으로 작동하므로(나노초 대 마이크로초), 적당한 처리 성능을 가능하게 하려면 CPU와 디스크를 분리해야 합니다. 여기에 메모리가 개입합니다. 데이터베이스 시스템에서 메모리의 기본 용도는 입출력을 피하는 것이므로, 어느 정도까지 시스템에 메모리가 많을수록 시스템이 더 잘 수행할 수 있습니다. 다행히도, 지난 몇 년간 메모리 비용이 크게 하락했고, 수십에서 수백 기가바이트(GB)의 RAM을 지닌 시스템이 드물지 않습니다. 일반적으로, 프로세서 코어당 4-8GB가 대부분의 응용프로그램에 적절합니다.

AIX 구성

좋은 성능을 얻기 위해 변경해야 하는 AIX 매개변수는 상대적으로 적습니다. 이러한 권장사항을 위해, 5.3 이상의 AIX 레벨을 가정해봅니다. 또한 시스템에 특정 설정이 이미 있는 경우(예를 들어, BW 또는 SAP 구성), 해당 설정은 다음 일반 지침보다 우선해야 합니다.

- VMO 매개변수 **LRU_FILE_REPAGE**가 0으로 설정되어야 합니다. 이 매개변수는 AIX 전산 페이지 또는 파일 시스템 캐시 페이지를 희생시킵니다. 또한 **minperm**이 3으로 설정되어야 합니다. 이러한 값은 모두 AIX 6.1에서 디폴트값입니다.
- AIO 매개변수 **maxservers**는 초기에 CPU당 10의 디폴트값으로 남겨둘 수 있습니다. 시스템이 활성 상태가 된 후, **maxservers**는 다음과 같이 조정됩니다.
  1. `ps –elfk | grep aio` 명령의 출력을 수집하고 모든 비동기 I/O(AIO) 커널 프로세스(aioservers)에서 동일한 양의 CPU 시간을 사용하고 있는지 판별하십시오.
  2. 그럴 경우, **maxservers**가 너무 낮게 설정되었을 수 있습니다. **maxservers**를 10% 증가시키고 1단계를 반복하십시오.
  3. 일부 aioservers에서 다른 것보다 더 적은 CPU 시간을 사용할 경우, 시스템에 적어도 필요한 만큼의 aioservers가 있습니다. 10%가 넘는 aioservers에서 더 적은 CPU를 사용할 경우, **maxservers**를 10% 줄이고 1단계를 반복하십시오.
- AIO 매개변수 **maxreqs**는 `MAX( NUM_IOCLEANERS x 256, 4096)`로 설정되어야 합니다. 이 매개변수는 미결 AIO 요청의 최대 수를 제어합니다.
- hdisk 매개변수 **queue_depth**는 배열의 실제 디스크 수를 기반으로 해야 합니다. 예를 들어, IBM® 디스크의 경우, **queue_depth**의 디폴트값이 3이고, 권장 값은 3 x *디바이스의 수*가 됩니다. 이 매개변수는 큐에 대기 가능한 디스크 요청의 수를 제어합니다.
- 디스크 어댑터 매개변수 **num_cmd_elems**는 어댑터에 연결된 모든 디바이스에 대한 **queue_depth**의 합계로 설정되어야 합니다. 이 매개변수는 어댑터에 대한 큐에 대기될 수 있는 요청의 수를 제어합니다.

Solaris 및 HP-UX 구성

Solaris 또는 HP-UX에서 실행되는 DB2의 경우, db2osconf 유틸리티를 사용하여 시스템 크기를 기반으로 커널 매개변수를 점검 및 권장할 수 있습니다. db2osconf 유틸

리티를 통해 메모리 및 CPU를 기반으로, 또는 현재 시스템 구성을 필요한 향후 구성에 비교하는 일반 스케일 인수를 사용하여 커널 매개변수를 지정할 수 있습니다. SAP 응용프로그램과 같은 대형 시스템을 실행 중인 경우 2 이상의 스케일 인수를 사용하는 것이 좋습니다. 일반적으로, db2osconf는 Solaris 및 HP-UX를 구성하기 위한 좋은 초기 시작 지점을 제공하지만, 현재 및 향후 워크로드를 고려할 수 없기 때문에 최적 값을 제공하지는 않습니다.

Linux 구성

Linux 시스템이 DB2 서버로 사용될 경우, Linux 커널 매개변수 중 일부를 변경해야 할 수도 있습니다. Linux 분산이 변경되기 때문에, 그리고 이 환경은 매우 유연하기 때문에, Linux 구현을 기반으로 유효성 확인되어야 하는 일부 가장 중요한 설정만 고려되어야 합니다.

64비트 시스템의 **SHMMAX**(공유 메모리 세그먼트의 최대 *크기*)는 최소 1GB - 1 073 741 824바이트로 설정되어야 하는 한편, 매개변수 **SHMALL**은 데이터베이스 서버에서 사용 가능한 메모리의 90%로 설정되어야 합니다. **SHMALL**은 디폴트로 8GB입니다. 다음은 DB2에 대한 기타 중요한 Linux 커널 구성 매개변수 및 해당 권장 값입니다.

- **kernel.sem**(4개의 커널 세마포어 설정 지정 – SEMMSL, SEMMNS, SEMOPM 및 SEMMNI): 250 256000 32 1024
- **kernel.msgmni**(메시지 큐 ID의 수): 1024
- **kernel.msgmax**(메시지의 최대 *크기*, 바이트): 65536
- **kernel.msgmnb**(메시지 큐의 디폴트 *크기*, 바이트): 65536

DB2 데이터베이스 파티셔닝 기능

DB2 데이터베이스 파티셔닝 기능(DPF) 사용 결정은 일반적으로 순전히 데이터 볼륨을 기반으로 이루어지는 것이 아니라 워크로드를 더 기반으로 하여 이루어집니다. 일반적인 지침으로, 대부분의 DPF 전개는 데이터 웨어하우징 및 비즈니스 인텔리전스의 영역에 있습니다. DPF는 비공유 아키텍처가 뛰어난 확장성을 허용하므로 대형 복합 쿼리 환경에 매우 권장됩니다. 빠르게 확장할 가능성이 없는 보다 작은 데이터 마트의 경우(최대 약 300GB), 대개 DB2 Enterprise Server Edition(ESE) 구성을 선택하는 것이 좋습니다. 그러나 대형 또는 빠르게 확장하는 BI 환경은 DPF에서 큰 이점을 얻습니다.

일반 파티션된 데이터베이스 시스템에는 보통 데이터 파티션당 하나의 프로세서 코어가 있습니다. 예를 들어, *n*개의 프로세서 코어가 있는 시스템은 파티션 0에 카탈로그가 있고 *n*개의 추가 데이터 파티션을 가질 수 있습니다. 카탈로그 파티션이 비중 있게 사용되는 경우(예를 들어, 단일 파티션 차원 테이블을 보유하기 위해), 여기에도 프로세서

코어가 할당될 수 있습니다. 시스템에서 매우 많은 동시 활성 사용자를 지원할 경우 파티션당 2개의 코어가 필요할 수도 있습니다.

일반 안내서에 따르면, 파티션당 약 250GB의 활성 원시 데이터를 계획해야 합니다.

InfoSphere Balanced Warehouse 문서에 파티션된 데이터베이스 구성 우수 사례에 대한 상세한 정보가 포함되어 있습니다. 이 문서에는 비 Balanced Warehouse에 대한 유용한 정보도 포함되어 있습니다.

조합 및 코드 페이지의 선택

코드 페이지 또는 코드 세트 및 조합 시퀀스의 선택은 데이터베이스 동작에 영향을 줄 뿐 아니라 성능에 강력한 영향을 미칠 수 있습니다. 유니코드는 기존 단일 바이트 코드 페이지를 사용한 경우보다 데이터베이스에서 훨씬 더 다양한 문자열을 나타낼 수 있도록 해 주기 때문에 매우 널리 사용되게 되었습니다. DB2 버전 9.5의 새 데이터베이스의 경우 유니코드가 디폴트입니다. 그러나 유니코드 코드 세트는 다중 바이트를 사용하여 일부 개별 문자를 나타내기 때문에, 디스크 및 메모리 요구사항이 증가될 수 있습니다. 예를 들어, 가장 일반적인 유니코드 코드 세트 중 하나인 UTF-8 코드 세트는 문자당 1-4바이트를 사용합니다. 단일 바이트 코드 세트에서 UTF-8로의 이주로 인한 평균 문자열 확장 인수는 다중 바이트 문자가 사용되는 빈도에 따라 달라지므로 추정하기가 매우 어렵습니다. 전형적인 북미 컨텐츠의 경우, 일반적으로 확장이 없습니다. 대부분의 서유럽 언어에서는, 강조 문자의 사용이 일반적으로 약 10%의 확장을 가져옵니다.

이것 이외에도, 유니코드의 사용은 단일 바이트 코드 페이지에 비해 추가 CPU 소비의 원인이 될 수 있습니다. 첫째, 확장이 발생할 경우, 더 긴 문자열은 더 많은 조작 작업을 필요로 합니다. 둘째로, 더 중요하게는, UCA500R1_NO와 같은 보다 복잡한 유니코드 조합에 의해 사용되는 알고리즘이 단일 바이트 코드 페이지에서 사용되는 일반 SYSTEM 조합보다 훨씬 더 비용이 많이 들 수 있습니다. 이런 비용 증가는 문화적으로 올바른 방법으로 유니코드 문자열을 정렬하기가 복잡하기 때문입니다. 영향을 받는 조작으로는 정렬, 문자열 비교, LIKE 처리 및 인덱스 작성이 있습니다.

데이터를 적절하게 나타내기 위해 유니코드가 필요한 경우 주의하여 조합 시퀀스를 선택하십시오.

- 데이터베이스가 다중 언어의 데이터를 포함하고, 해당 데이터의 올바른 정렬 순서가 가장 중요한 경우, 문화적으로 올바른 조합 중 하나를 사용하십시오
  (예: UCA500R1_*). 데이터 및 응용프로그램에 따라, 이것은 IDENTITY 시퀀스에 비해 1.5 - 3배 많은 성능 오버헤드를 가질 수 있습니다.
- 다양한 정규화 및 비정규화된 문화적으로 올바른 조합이 있습니다. 정규화된 조합
  (예: UCA500R1_NO)은 형식이 잘못된 문자를 처리하기 위해 추가 점검을 하는 반면, 비정규화된 조합 (예: UCA500r1_NX)은 그렇지 않습니다. 형식이 잘못된 문자

의 처리가 쟁점이 아닌 경우, 정규화 코드를 사용하지 않으면 성능 이점이 있으므로 비정규화된 버전을 사용하십시오. 즉, 비정규화된 문화적으로 올바른 조합일지라도 비용이 매우 많이 듭니다.

- 데이터베이스가 단일 바이트 환경에서 유니코드 환경으로 이동되지만 다양한 언어 호스팅에 대한 엄격한 요구사항이 없는 경우(대부분의 전개가 이 범주에 속하게 됨), 언어 인식 조합이 적절할 수도 있습니다. *언어 인식 조합*(예를 들어, SYSTEM_819_BE)은 많은 유니코드 데이터베이스가 하나의 언어로만 데이터를 포함한다는 사실을 이용합니다. SYSTEM_819와 같은 단일 바이트 조합과 동일한 찾아보기 테이블 기반 조합 알고리즘을 사용하므로 매우 효과적입니다. 일반적으로, 원래 단일 바이트 데이터베이스의 조합 동작을 승인할 수 있는 경우, 유니코드로의 이동에 따라 언어 컨텐츠가 크게 변경되지 않는 한, 문화적 인식 조합이 고려되어야 합니다. 이는 문화적으로 올바른 조합에 비해 매우 큰 성능 이점을 제공할 수 있습니다.

실제 데이터베이스 설계

- 일반적으로, 파일 기만 데이터베이스 관리 스토리지(DMS) 일반 테이블 스페이스는 시스템 관리 스토리지(SMS) 일반 테이블 스페이스보다 더 좋은 성능을 제공합니다. SMS는 특히 임시 테이블이 매우 작을 경우 임시 테이블 스페이스에 자주 사용되지만, 이 경우 SMS의 성능은 시간에 따라 저하됩니다.
- 과거에는, DMS 원시 디바이스 테이블 스페이스가 DMS 파일 테이블 스페이스에 비해 꽤 많은 성능 장점을 지녔지만, 직접 입출력 (현재 CREATE TABLESPACE 및 ALTER TABLESPACE문에서 NO FILE SYSTEM CACHING 절을 통해 디폴트 지정됨)의 도입으로, DMS 파일 테이블 스페이스가 DMS 원시 디바이스 테이블 스페이스와 사실상 동일한 성능을 제공합니다.

초기 DB2 구성 설정

AUTOCONFIGURE 명령이라고도 하는 DB2 구성 어드바이저는 사용자가 제공하는 기본 시스템 지침을 사용하고, DB2 구성 값의 적절한 시작 세트를 판별합니다. AUTOCONFIGURE 명령은 디폴트 구성 설정에 대해 실질적인 향상을 제공할 수 있으며, 초기 구성 값을 얻는 방법으로 권장됩니다. 시스템의 특성에 따라, 종종 AUTOCONFIGURE 명령에 의해 생성된 권장사항에 대한 일부 추가 미세 조정이 필요합니다.

다음은 AUTOCONFIGURE 명령 사용에 대한 몇 가지 제안사항입니다.

- DB2 버전 9.1부터 AUTOCONFIGURE 명령이 데이터베이스 작성 시간에 자동으로 실행된다 하더라도, AUTOCONFIGURE 명령을 여전히 명시적으로 실행하는 것이 좋습니다. 그래야 시스템의 결과를 사용자 정의하는 데 도움이 되는 키워드/값 쌍을 지정할 수 있기 때문입니다.

- 데이터베이스가 적절한 양의 활성 데이터로 채워진 후 AUTOCONFIGURE 명령을 실행(또는 재실행)하십시오. 이렇게 하면 도구에 데이터베이스의 성질에 대한 자세한 정보가 제공됩니다. 데이터베이스를 채우는 데 사용하는 데이터의 양은 예를 들어 버퍼 풀 크기 계산과 같은 작업에 영향을 줄 수 있기 때문에 중요합니다. 데이터가 너무 많거나 너무 적으면 이러한 계산이 덜 정확해집니다.
- 중요한 AUTOCONFIGURE 명령 키워드로 다양한 값(예: **mem_percent**, **tpm** 및 **num_stmts**)을 시도하여 이런 변경에 의해 어떤 구성 값이, 어느 정도의 영향을 받는지 파악하십시오.
- 다른 키워드 및 값으로 시도할 경우, APPLY NONE 옵션을 사용하십시오. 이렇게 하면 현재 설정과 권장사항을 비교할 수 있습니다.
- 디폴트가 시스템에 적합하지 않을 수도 있으므로 모든 키워드에 대한 값을 지정하십시오. 예를 들어, **mem_percent** 디폴트는 25%이고, 이것은 전용 DB2 서버에서 너무 낮은 값입니다. 이 경우 권장되는 값은 85%입니다.

DB2 자율 및 자동 매개변수

DB2 데이터베이스 제품의 최근 릴리스에서는 인스턴스 또는 데이터베이스 시작 시간에 자동으로 설정되거나 조작 동안 동적으로 조정되는 매개변수의 수를 크게 늘렸습니다. 대부분의 시스템에서, 자동 설정은 매우 주의 깊게 수동 조정되는 시스템을 제외하고 다른 모든 설정보다 더 나은 성능을 제공합니다. 이는 특히 DB2 시스템에서 주 메모리 소비자 4개(버퍼 풀, 잠금 목록, 패키지 캐시 및 정렬 힙)뿐 아니라 총 데이터베이스 메모리 할당을 동적으로 조정하는 DB2 자체 조정 메모리 관리자(STMM) 때문입니다.

해당 매개변수는 파티션별로 적용되기 때문에 파티션된 환경에서의 STMM 사용은 어느 정도 주의를 기울여 수행되어야 합니다. 파티션된 데이터베이스 시스템에서, STMM은 단일 파티션의 메모리 요구사항을 계속해서 측정하고(DB2 시스템이 자동으로 선택하지만, 해당 선택을 겹쳐쓸 수 있음), 힙 크기 갱신을 STMM을 사용할 수 있는 모든 파티션으로 '푸시아웃' 합니다. 모든 파티션에서 동일한 값이 사용되므로, 파티션에 걸쳐 데이터의 양, 메모리 요구사항 및 활동의 일반 레벨이 매우 균일한 파티션된 데이터베이스 환경에서 STMM가 가장 잘 작동합니다. 작은 수의 파티션이 비대칭 데이터 볼륨이나 다른 메모리 요구사항을 지닌 경우, 해당 파티션에서 STMM이 사용되지 않아야 하고, 보다 균일한 파티션을 조정하도록 허용되어야 합니다. 예를 들어, 카탈로그 파티션에서 일반적으로 STMM이 사용되지 않아야 합니다.

데이터 분산이 편중된 파티션된 데이터베이스 환경(연속적 클러스터 간 메모리 조정이 권장되지 않음) 경우, '조정 단계' 동안 STMM을 선택적 및 일시적으로 사용하여 적절한 수동 힙 설정을 판별하는 데 도움을 줄 수 있습니다.

- 하나의 '일반' 파티션에서 STMM을 사용할 수 있게 하십시오. 다른 파티션에서는 계속 STMM이 사용되지 않습니다.

- 메모리 설정이 안정화된 후, STMM을 사용할 수 없게 하고 영향을 받은 매개변수를 조정된 값으로 수동으로 '고정'하십시오.
- 비슷한 데이터 볼륨과 메모리 요구사항을 지닌 다른 데이터베이스 파티션(예를 들어, 같은 파티션 그룹의 파티션)에 조정된 값을 전개하십시오.
- 비슷한 볼륨 및 데이터 유형을 포함하고 시스템에서 비슷한 역할을 수행하는 연결되지 않은 데이터베이스 파티션 세트가 여러 개 있는 경우 프로세스를 반복하십시오.

구성 어드바이저는 일반적으로 해당하는 경우 자율 설정을 사용하도록 선택합니다. 여기에는 RUNSTATS 명령의 자동 통계 갱신이 포함되지만(매우 유용함), 자동 재구성 및 자동 백업은 제외됩니다. 자동 재구성 및 자동 백업도 매우 유용할 수 있지만 최상의 결과를 위해서는 사용자 환경 및 스케줄에 따라 구성되어야 합니다. 디폴트로, 자동 통계 프로파일링은 사용되지 않도록 유지되어야 합니다. 이는 매우 높은 오버헤드를 지니며 제어된 조건 하에서 복합 명령문을 사용하여 임시적으로 사용되도록 설계되었습니다.

명시적 구성 설정

일부 매개변수는 자동 설정이 없으며, 구성 어드바이저에 의해 설정되지 않습니다. 이러한 매개변수는 명시적으로 다뤄져야 합니다. 여기서는 성능과 관련된 매개변수만 고려됩니다.

- **logpath** 또는 **newlogpath**는 트랜잭션 로그의 위치를 판별합니다. 구성 어드바이저가 로그의 위치를 결정하지 못할 수도 있습니다. 위에 언급했듯이, 가장 중요한 점은 테이블 스페이스와 같은 다른 DB2 오브젝트와 디스크 디바이스를 공유하지 않거나, 데이터베이스 경로 아래 디폴트 위치에 남아있도록 허용되어야 한다는 점입니다. 원칙적으로, 트랜잭션 로그는 병목 현상이 생기지 않도록 처리량 공간이 충분히 있는 전용 스토리지에 배치되어야 합니다.
- **logbufsz**는 4KB 페이지의 트랜잭션 로그 프로그램 내부 버퍼의 *크기*를 판별합니다. 디폴트값인 8페이지는 프로덕션 환경에서 좋은 성능을 얻기에는 너무 작습니다. 구성 어드바이저는 입력 매개변수에 따라 항상 이 *크기*를 늘리지만, 충분하지 않을 수 있습니다. 256-1000페이지 값이 적절한 일반 범위이며, 데이터베이스 서버의 전체 스킴에서 매우 작은 총 메모리 양만을 나타냅니다.
- **mincommit**는 DB2 시스템에서 *n*개의 커미팅 트랜잭션을 함께 일괄처리 시도하도록 하는 *그룹 커미트*를 제어합니다. 현재 트랜잭션 로그 프로그램 설계에서, 이것은 좀처럼 바람직한 동작이 아닙니다. **mincommit**를 디폴트값인 1로 남겨두십시오.
- **buffpage**는 -1의 *크기*로 정의된 각 버퍼 풀에 할당된 페이지의 수를 판별합니다. 우수 사례는 **buffpage**를 무시하고 SYSCAT.BUFFERPOOLS에 항목이 있는 버퍼 풀의 *크기*를 명시적으로 설정하거나, STMM에서 버퍼 풀 *크기*를 자동으로 조정하도록 하는 것입니다.
- **diagpath**는 여러 유용한 DB2 진단 파일의 위치를 판별합니다. 이것은 파티션된 데이터베이스 환경의 경우를 제외하고는 일반적으로 성능에 거의 영향을 미치지 않습

니다. 모든 파티션에서 **diagpath**의 디폴트 위치는 일반적으로 공유 NFS 마운트 경로에 있습니다. 우수 사례는 각 파티션에 대한 로컬, 비 NFS 디렉토리로 **diagpath**를 겹쳐쓰는 것입니다. 이렇게 하면 모든 파티션에서 진단 메시지를 사용하여 동일한 파일을 갱신하려고 시도하는 것이 예방됩니다. 대신, 진단 파일이 각 파티션에 대해 로컬로 보존되고, 경쟁이 크게 줄어듭니다.

- **DB2_PARALLEL_IO**는 구성 매개변수가 아니라, DB2 레지스트리 변수입니다. DB2 시스템에서 운영 체제에 단일 디바이스로 표시되는 디스크 배열로 구성된 스토리지를 사용하거나, 여러 디바이스에 걸쳐 있는 파일 시스템을 사용하는 것은 매우 일반적입니다. 그 결과, 디폴트로 DB2 데이터베이스 시스템에서 테이블 스페이스 컨테이너에 대해 한 번에 하나의 프리페치 요청만 수행합니다. 이것은 단일 디바이스에 대한 다중 요청이 결국 일련화된다는 사실을 통해 성립됩니다. 그러나 컨테이너가 디스크 배열에 상주할 경우, 일련화하지 않고 다중 프리페치 요청을 동시에 디스패치할 수 있습니다. 여기에 **DB2_PARALLEL_IO**가 개입합니다. 이 변수는 DB2 시스템에 프리페치 요청이 단일 컨테이너에 대해 병렬로 발행될 수 있음을 알려줍니다. 가장 단순한 설정은 DB2_PARALLEL_IO=*(모든 컨테이너가 복수 디스크에 상주함을 의미-이 경우에는 7개인 것으로 가정)이지만, 다른 설정도 병렬 처리 수준 및 영향을 받는 테이블을 제어합니다. 예를 들어, 컨테이너가 4개 디스크의 RAID-5 배열에 상주한다는 것을 알고 있을 경우, **DB2_PARALLEL_IO**를 *:3으로 설정할 수도 있습니다. 특정 값이 성능에 이익을 주는지 여부 또한 Extent 크기, RAID 세그먼트 크기 및 동일한 디스크 세트를 사용하는 컨테이너 수에 따라 달라집니다.

SAP 및 기타 ISV 환경에 대한 고려사항

SAP와 같은 ISV 응용프로그램에 대해 DB2를 실행 중인 경우, 특정 응용프로그램을 고려하는 우수 사례 지침이 사용 가능할 수도 있습니다. 가장 간단한 메커니즘은 SAP 워크로드에 대해 최적화된 집계 레지스트리 변수를 사용하도록 SAP 값으로 설정할 수 있는 DB2 레지스트리 변수 **DB2_WORKLOAD**입니다.

사전 결정된 값으로 설정되어야 하기 때문에, 조합 시퀀스 및 코드 페이지 또는 코드 세트의 선택과 같은 다른 권장사항 및 우수 사례가 적용될 수도 있습니다. 세부사항은 응용프로그램 벤더의 문서를 참조하십시오.

SAB Business One과 같은 여러 ISV 응용프로그램의 경우, AUTOCONFIGURE 명령을 사용하여 초기 구성을 정의할 수 있습니다. 그러나 SAP 설치 동안 DB2 구성 매개변수의 초기 세트가 적용되기 때문에, SAP NetWeaver 설치에서는 사용되지 않아야 합니다. 또한 SAP는 기본 DB2 매개변수 설정을 설명하는 강력한 대체 우수 사례 접근 방법(SAP Notes)을 지닙니다(예: SAP Note 1086130 - DB6: DB2 9.5 표준 매개변수 설정).

DB2 DPF 기능을 사용할 때 SAP 응용프로그램에 특별한 주의를 기울이십시오. SAP는 SAP NetWeaver 비즈니스 인텔리전스(비즈니스 웨어하우스) 제품에서 주로 DPF를 사

용합니다. 권장 레이아웃은 파티션 0에 DB2 시스템 카탈로그, 차원 및 마스터 테이블 및 SAP 기본 테이블을 포함합니다. 그 결과 이 파티션에 다른 DB2 DPF 설치와 비교하여 다른 워크로드가 있게 됩니다. SAP 응용프로그램 서버가 이 파티션에서 실행되므로, 이 파티션에만 최대 8개까지의 프로세스를 할당할 수도 있습니다. SAP BW 워크로드가 더 고도로 병렬 처리되면서, 많은 짧은 쿼리가 동시에 실행되므로, SAP BI에 대한 파티션 수는 일반적으로 다른 응용프로그램의 파티션 수보다 더 적습니다. 즉, 데이터 파티션당 두 개 이상의 CPU가 필요합니다.

인스턴스 구성

새 DB2 인스턴스를 시작한 경우 기본 구성을 설정하기 위해 수행할 수 있는 여러 단계가 있습니다.

- 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. 권장 값을 표시하거나 CREATE DATABASE 명령에서 APPLY 옵션을 사용하여 적용할 수 있습니다. 권장사항은 사용자가 제공하는 입력 및 어드바이저가 수집하는 시스템 정보에 따라 다릅니다.
- 구성 지원 프로그램을 사용하여 데이터베이스 오브젝트를 구성 및 유지보수하고, 새 오브젝트를 추가하고, 응용프로그램을 바인드하고, 데이터베이스 관리 프로그램 구성 매개변수를 설정하고, 구성 정보를 임포트 및 익스포트할 수 있습니다. 구성 지원 프로그램을 열려면 db2ca 명령을 호출하십시오. 인스턴스 구성의 경우 구성 지원 프로그램은 데이터베이스 관리 프로그램 구성 매개변수 설정, DB2 레지스트리 변수 설정, 다른 인스턴스 구성 또는 구성 재설정을 지원합니다.
- 데이터베이스 관리 프로그램 또는 데이터베이스에서 사용할 수 있는 각 구성 매개변수를 나열하고 간단하게 설명하는 요약 테이블을 참조하십시오(『구성 매개변수 요약』 참조). 이러한 요약 테이블에는 특정 매개변수 조정으로 성능 변경의 수준이 상중하 또는 없음인지 여부를 표시하는 컬럼이 있습니다. 이러한 테이블을 사용하여 사용자의 환경에서 최대 성능 향상을 실현하는 데 도움이 되는 매개변수를 찾으십시오.
- ACTIVATE DATABASE 명령을 사용하여 데이터베이스를 활성화하고 필요한 모든 데이터베이스 서비스를 시작하므로 데이터베이스를 모든 응용프로그램의 연결 및 사용에 이용할 수 있습니다. 파티션된 데이터베이스 환경에서 이 명령은 모든 데이터베이스 파티션에서 데이터베이스를 활성화하고 첫 번째 응용프로그램이 연결될 때 데이터베이스를 초기화하는 데 필요한 시작 시간을 방지합니다.

테이블 스페이스 설계

디스크 스토리지 성능 요인

디스크 스토리지 구성과 같은 하드웨어 특성은 시스템의 성능에 영향을 줄 수 있습니다.

다음과 같은 하나 이상의 디스크 스토리지 구성 측면이 성능에 영향을 줄 수 있습니다.

- 스토리지 분배

  인덱스와 데이터 사이, 테이블 스페이스들 사이에 제한된 스토리지 양을 분배하는 방식에 따라 여러 가지 다른 상황에서 시스템 성능이 상당히 달라집니다.

- 디스크 입출력 분산

  여러 디바이스 및 제어기에서 어느 정도 디스크 입출력 요구의 균형을 유지하느냐에 따라 데이터베이스 관리 프로그램이 디스크에서 데이터를 검색할 수 있는 속도가 다릅니다.

- 디스크 서브시스템 코어 성능 메트릭

  초당 디스크 조작 수나, 초당 전송된 MB 단위 용량은 전체 시스템 성능에 아주 많은 영향을 줍니다.

쿼리 최적화에 대한 테이블 스페이스 영향

테이블 스페이스의 일정한 특성은 쿼리 컴파일러가 선택한 액세스 플랜에 영향을 줄 수 있습니다.

이러한 특성은 다음과 같습니다.

- 컨테이너 특성

  컨테이너 특성은 쿼리 실행과 연관된 입출력 비용에 상당한 영향을 줄 수 있습니다. 액세스 플랜을 선택할 때 쿼리 옵티마이저는 다른 테이블 스페이스에서 데이터에 액세스할 때 비용 차이를 포함하여 이러한 입출력 비용을 고려합니다. 옵티마이저는 SYSCAT.TABLESPACES 카탈로그 뷰의 두 컬럼을 사용하여 테이블 스페이스에서 데이터에 액세스할 때 입출력 비용을 추정합니다.

  – OVERHEAD는 메모리로 데이터를 읽기 전에 컨테이너에 필요한 시간 추정(밀리초)을 제공합니다. 이 오버헤드 활동에는 컨테이너의 입출력 제어기 오버헤드와 디스크 탐색 시간을 포함한 디스크 대기 시간이 있습니다.

    다음 공식을 사용하여 오버헤드 비용을 추정할 수 있습니다.

    ```
    OVERHEAD = 평균 탐색 시간(밀리초)
               + (0.5 * 회전 대기 시간)
    ```

    각 항목에 대한 설명은 다음과 같습니다.

- 0.5는 절반 회전의 평균 오버헤드를 나타냅니다.
- 다음과 같이 각 전체 회전에서 회전 대기 시간(밀리초)을 계산합니다.

  ```
  (1 / RPM) * 60 * 1000
  ```

  각 항목에 대한 설명은 다음과 같습니다.

  - 분 당 회전으로 나누어 회전 당 분 수를 계산합니다.
  - 분 당 60초를 곱합니다.
  - 초 당 1000밀리초를 곱합니다.

  예를 들어, 디스크가 분 당 7200 회전을 수행한다고 가정하십시오. 회전 대기 시간 공식을 사용하십시오.

  ```
  (1 / 7200) * 60 * 1000 = 8.328밀리초
  ```

  이 값을 사용하면 11밀리초의 평균 탐색 시간을 가정할 때 다음과 같이 오버헤드를 추정할 수 있습니다.

  ```
  OVERHEAD = 11 + (0.5 * 8.328)
           = 15.164
  ```

– TRANSFERRATE는 메모리로 하나의 데이터 페이지를 읽는 데 필요한 추정 시간(밀리초)을 제공합니다.

  각 테이블 스페이스 컨테이너가 단일 실제 디스크인 경우 다음 공식을 사용하여 전송 비용(밀리초/페이지)을 추정할 수 있습니다.

  ```
  TRANSFERRATE = (1 / spec_rate) * 1000 / 1024000 * page_size
  ```

  각 항목에 대한 설명은 다음과 같습니다.

  - 전송률(MB/초)의 디스크 스펙을 나타내는 *spec_rate*로 나누어 MB 당 시간(초)을 계산할 수 있습니다.
  - 초 당 1000밀리초를 곱합니다.
  - MB 당 1 024 000바이트로 나눕니다.
  - 페이지 크기(바이트)를 곱합니다(예: 4KB 페이지의 경우 4096바이트).

  예를 들어, 디스크의 스펙 비율이 초 당 3MB라고 가정하십시오. 이 경우 다음과 같습니다.

  ```
  TRANSFERRATE = (1 / 3) * 1000 / 1024000 * 4096
               = 1.333248
  ```

  또는 페이지 당 약 1.3밀리초입니다.

  테이블 스페이스 컨테이너가 하나의 실제 디스크가 아닌 디스크 배열(예: RAID)인 경우 TRANSFERRATE를 추정할 때 조금 더 주의하여야 합니다.

배열이 상대적으로 작은 경우 디스크 레벨에서 병목 현상이 발생한다고 가정할 때 *spec_rate*에 디스크 수를 곱할 수 있습니다. 그러나 배열이 큰 경우 병목 현상은 디스크 레벨이 아닌 기타 입출력 서브시스템 구성요소(예: 디스크 제어기, 입출력 버스 또는 시스템 버스) 중 하나에서 발생할 수 있습니다. 이 경우 입출력 처리량 용량이 *spec_rate* 및 디스크 수의 결과라고 가정할 수 없습니다. 대신 순차 스캔 중 실제 입출력 비율(MB)을 측정해야 합니다. 예를 들어, `select count(*) from big_table`로 인한 순차 스캔의 크기는 몇 MB일 수 있습니다. 이 경우 BIG_TABLE이 상주하는 테이블 스페이스를 구성하는 컨테이너 수로 결과를 나누십시오. 이 결과를 사용하여 위에 제공된 공식에서 *spec_rate*를 대체하십시오. 예를 들어, 네 개의 컨테이너로 구성된 테이블 스페이스를 스캔하는 동안 측정된 순차 입출력 비율인 100MB는 컨테이너 당 25MB를 나타내거나 (1 / 25) * 1000 / 1 024 000 * 4096 = 0.16밀리초/페이지의 TRANSFERRATE를 나타냅니다.

테이블 스페이스에 지정된 컨테이너들은 여러 실제 디스크에 상주할 수도 있습니다. 최상의 결과를 얻기 위해 지정된 테이블 스페이스에 사용되는 모든 실제 디스크들은 OVERHEAD 및 TRANSFERRATE 특성이 동일해야 합니다. 이러한 특성이 동일하지 않은 경우 OVERHEAD 및 TRANSFERRATE를 설정할 때 평균값을 사용해야 합니다.

하드웨어 스펙에서 또는 조사를 통해 이러한 컬럼의 미디어별 값을 구할 수 있습니다. 이러한 값을 CREATE TABLESPACE 및 ALTER TABLESPACE문에 지정할 수 있습니다.

- 프리페치

  테이블 스페이스의 데이터에 액세스하는 입출력 비용을 고려할 때 옵티마이저는 디스크에서 데이터 및 인덱스 페이지를 프리페치할 경우 쿼리 성능에 미칠 수 있는 잠재적인 영향도 고려합니다. 프리페치는 버퍼 풀로 데이터를 읽을 때 연관된 오버헤드를 줄일 수 있습니다.

  옵티마이저는 SYSCAT.TABLESPACES 카탈로그 뷰의 PREFETCHSIZE 및 EXTENTSIZE 컬럼에 있는 정보를 사용하여 발생할 프리페치의 양을 추정합니다.

  – EXTENTSIZE는 테이블 스페이스를 작성할 경우에만 설정할 수 있습니다. 4 또는 8페이지의 Extent 크기면 보통 충분합니다.
  – PREFETCHSIZE는 테이블 스페이스를 작성하거나 변경할 때 설정할 수 있습니다. 디폴트 프리페치 크기는 **dft_prefetch_sz** 데이터베이스 구성 매개변수의 값으로 판별합니다. 이 매개변수를 크기 지정할 때 권장사항을 검토하고 필요하면 변경하거나, AUTOMATIC으로 설정하십시오.

테이블 스페이스를 변경한 후 응용프로그램을 리바인드하기 전에 runstats 유틸리티를 실행하여 인덱스에 대한 최신 통계를 수집하고 쿼리 옵티마이저가 가능한 최상의 데이터 액세스 플랜을 선택하는지 확인하십시오.

데이터베이스 설계

테이블

표준 테이블의 테이블 및 인덱스 관리

표준 테이블에서 데이터는 데이터 페이지 목록으로 논리적으로 구성됩니다. 이러한 데이터 페이지는 테이블 스페이스의 Extent 크기에 따라 논리적으로 그룹화됩니다.

예를 들어, Extent 크기가 4인 경우 0 - 3페이지는 첫 번째 Extent의 일부분이고 4 - 7페이지는 두 번째 Extent의 일부분이 되는 등 계속 이와 같은 방식입니다.

각 데이터 페이지에 들어 있는 레코드 수는 데이터 페이지의 크기 및 레코드의 크기에 따라 다를 수 있습니다. 대부분의 페이지에는 사용자 레코드만 들어 있습니다. 그러나 소수의 일부 페이지에만 데이터 서버가 테이블을 관리하기 위해 사용하는 특수 내부 레코드가 들어 있습니다. 예를 들어, 표준 테이블에는 500번째 데이터 페이지(66 페이지의 그림 8)마다 FSCR(Free Space Control Record)이 있습니다. 이러한 레코드는 다음의 각 500 데이터 페이지에 있는 새 레코드에 사용할 수 있는 여유 공간을 맵핑합니다(다음 FSCR이 발견될 때까지).

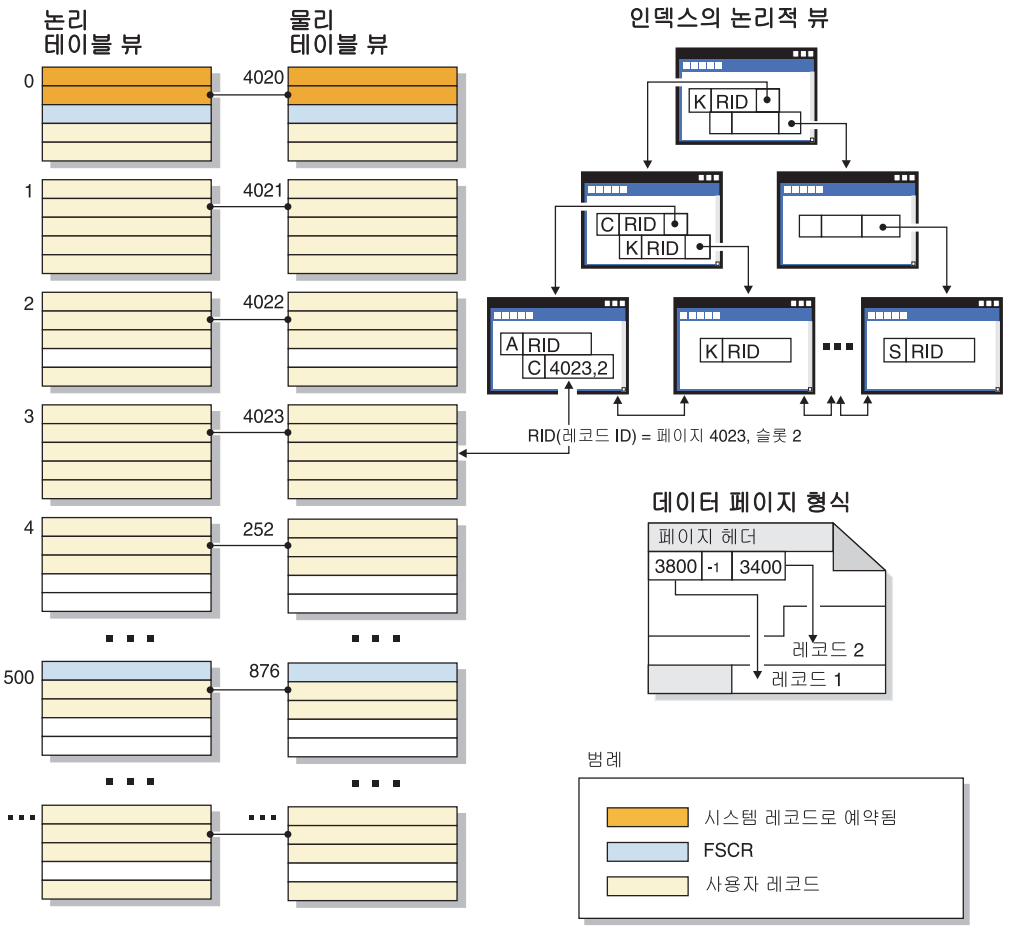


그림 8. 표준 테이블의 논리 테이블, 레코드 및 인덱스 구조

논리적으로 인덱스 페이지는 특정 키 값이 있는 테이블 레코드를 효율적으로 찾을 수 있는 B-트리로 구성되어 있습니다. 인덱스 페이지의 엔티티 수는 고정되어 있지 않지만 키의 크기에 따라 다릅니다. 데이터베이스 관리 스페이스(DMS) 테이블 스페이스에 있는 테이블의 경우 인덱스 페이지의 레코드 ID(RID)는 오브젝트 관련 페이지 번호가 아닌 테이블 스페이스 관련 페이지 번호를 사용합니다. 따라서 인덱스 스캔 시 맵핑 시 Extent 맵 페이지(EMP)가 없어도 데이터 페이지에 직접 액세스할 수 있습니다.

각 데이터 페이지는 형식이 동일합니다. 페이지는 페이지 헤더로 시작하며 뒤에 슬롯 디렉토리가 표시됩니다. 슬롯 디렉토리의 각 항목은 페이지의 다른 레코드에 해당합니다. 슬롯 디렉토리의 항목은 레코드가 시작되는 데이터 페이지의 바이트 오프셋을 나타냅니다. -1 항목은 삭제된 레코드에 해당합니다.

레코드 ID 및 페이지

레코드 ID는 페이지 번호와 그 뒤의 슬롯 번호(67 페이지의 그림 9)로 구성되어 있습니다. 인덱스 레코드에는 ridFlag라고 하는 추가 필드가 있습니다. ridFlag는 삭제된 것으로 표시되었는지 여부와 같이 키 상태에 대한 정보를 인덱스에 저장합니다. 인덱스를 사용하여 RID를 식별한 후 RID를 사용하여 올바른 데이터 페이지와 이 페이지의 슬

롯 번호를 식별합니다. RID에 레코드를 지정한 후 테이블이 재구성될 때까지 RID는 변경되지 않습니다.

데이터 페이지 및 RID 형식

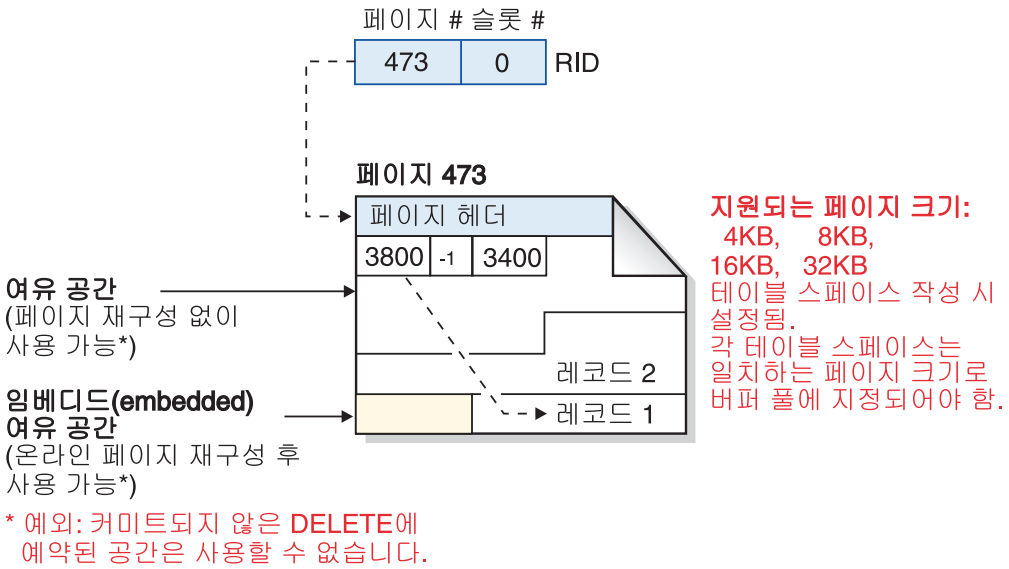


그림 9. 데이터 페이지 및 레코드 ID(RID) 형식

테이블 페이지를 재구성하는 경우 레코드가 실제로 삭제된 후 페이지에 남아 있는 임베디드(embedded) 여유 공간은 사용 가능한 여유 공간으로 변환됩니다.

DB2 데이터 서버는 다른 페이지 *크기*를 지원합니다. 행에 순차적으로 액세스하는 워크로드의 경우 큰 페이지 *크기*를 사용하십시오. 예를 들어, 순차적 액세스는 주로 결정 지원 응용프로그램 또는 임시 테이블을 광범위하게 사용 중인 경우에 사용됩니다. 행에 무작위로 액세스하는 워크로드의 경우 작은 페이지 *크기*를 사용하십시오. 예를 들어, 무작위 액세스는 주로 온라인 트랜잭션 처리(OLTP) 환경에서 사용됩니다.

표준 테이블의 인덱스 관리

DB2 인덱스는 미리 쓰기 로깅을 사용하는 효율적이고 동시성이 높은 인덱스 관리 방법에 기초한 최적화된 B-트리 구현을 사용합니다. B-트리 인덱스는 항목이 삽입되거나 삭제될 때 데이터 키를 재정렬하여 액세스 시간을 최소화하는 균형잡힌 페이지 계층 구조로 배열됩니다.

최적화된 B-트리 구현은 리프 페이지에 양방향 포인터가 있으므로 단일 인덱스가 정방향 또는 역방향으로 스캔을 지원할 수 있습니다. 인덱스 페이지는 90/10 분할(즉, 인덱스 키의 최고 10%가 새 페이지에 배치됨)이 사용된 하이 키 페이지를 제외하고 보통 절반으로 분할됩니다. 이러한 인덱스 페이지 분할 유형은 삽입 조작이 주로 새 하이 키 값으로 완료되는 워크로드의 경우에 유용합니다.

삭제된 인덱스 키는 테이블에 X 잠금이 있는 경우에만 인덱스 페이지에서 제거됩니다. 키를 즉시 제거할 수 없는 경우 삭제된 것으로 표시되며 실제로 나중에 제거됩니다.

인덱스가 작성되었을 때 MINPCTUSED에 양수 값을 지정하여 온라인 인덱스 조각 모음이 사용되도록 설정하면 인덱스 리프 페이지를 온라인으로 병합할 수 있습니다. MINPCTUSED는 인덱스 리프 페이지에서 사용된 스페이스의 최소 퍼센트를 표시합니다. 키가 제거된 후 인덱스 페이지에서 사용된 스페이스의 양이 이 값 아래로 감소된 경우 데이터베이스 관리 프로그램은 나머지 키를 인접한 페이지의 키와 병합하려고 시도합니다. 충분한 공간이 있는 경우 병합이 수행되고 인덱스 리프 페이지가 삭제됩니다. 인덱스 페이지에서 키가 제거되는 경우에만 온라인 조각 모음이 발생하므로 키가 삭제된 것으로 표시되고 페이지에서 실제로 제거되지 않은 경우 온라인 조각 모음이 발생하지 않습니다. 온라인 인덱스 조각 모음은 스페이스 재사용률을 높일 수 있지만 MINPCTUSED 값이 너무 높으면 병합에 필요한 시간이 증가하고 병합이 올바로 완료되지 않을 가능성이 증가합니다. MINPCTUSED에 권장되는 값은 50% 이하입니다.

CREATE INDEX문의 INCLUDE절을 사용하여 인덱스 리프 페이지에 대해 하나 이상의 컬럼(키 컬럼 이상)을 지정할 수 있습니다. 인덱스 B-트리에 대한 정렬 조작에 관련되지 않은 포함 컬럼은 인덱스 전용 액세스에 사용 가능한 쿼리 수를 증가시킬 수 있습니다. 그러나 인덱스 스페이스 요구사항도 증가시킬 수 있으므로 포함된 컬럼을 자주 갱신하는 경우 인덱스 유지보수 비용도 증가합니다. 포함 컬럼 갱신의 유지보수 비용은 키 컬럼 갱신의 비용보다 작지만 인덱스의 일부분이 아닌 컬럼의 갱신 비용보다 많습니다.

MDC 테이블의 테이블 및 인덱스 관리

MDC(다차원 클러스터링) 테이블의 테이블 및 인덱스 구성은 표준 테이블 구성과 동일한 논리 구조에 기초합니다.

표준 테이블과 마찬가지로 MDC 테이블은 컬럼으로 구분된 데이터 행이 들어 있는 페이지로 구성됩니다. 각 페이지의 행은 레코드 ID(RID)로 식별합니다. 그러나 MDC 테이블의 페이지는 Extent 크기 블록으로 그룹화됩니다. 예를 들어, 69 페이지의 그림 10에서는 Extent 크기가 4인 테이블을 표시합니다. 0 - 3의 처음 네 페이지는 테이블을 첫 번째 블록을 나타냅니다. 4 - 7의 다음 네 페이지는 테이블의 두 번째 블록을 나타냅니다.

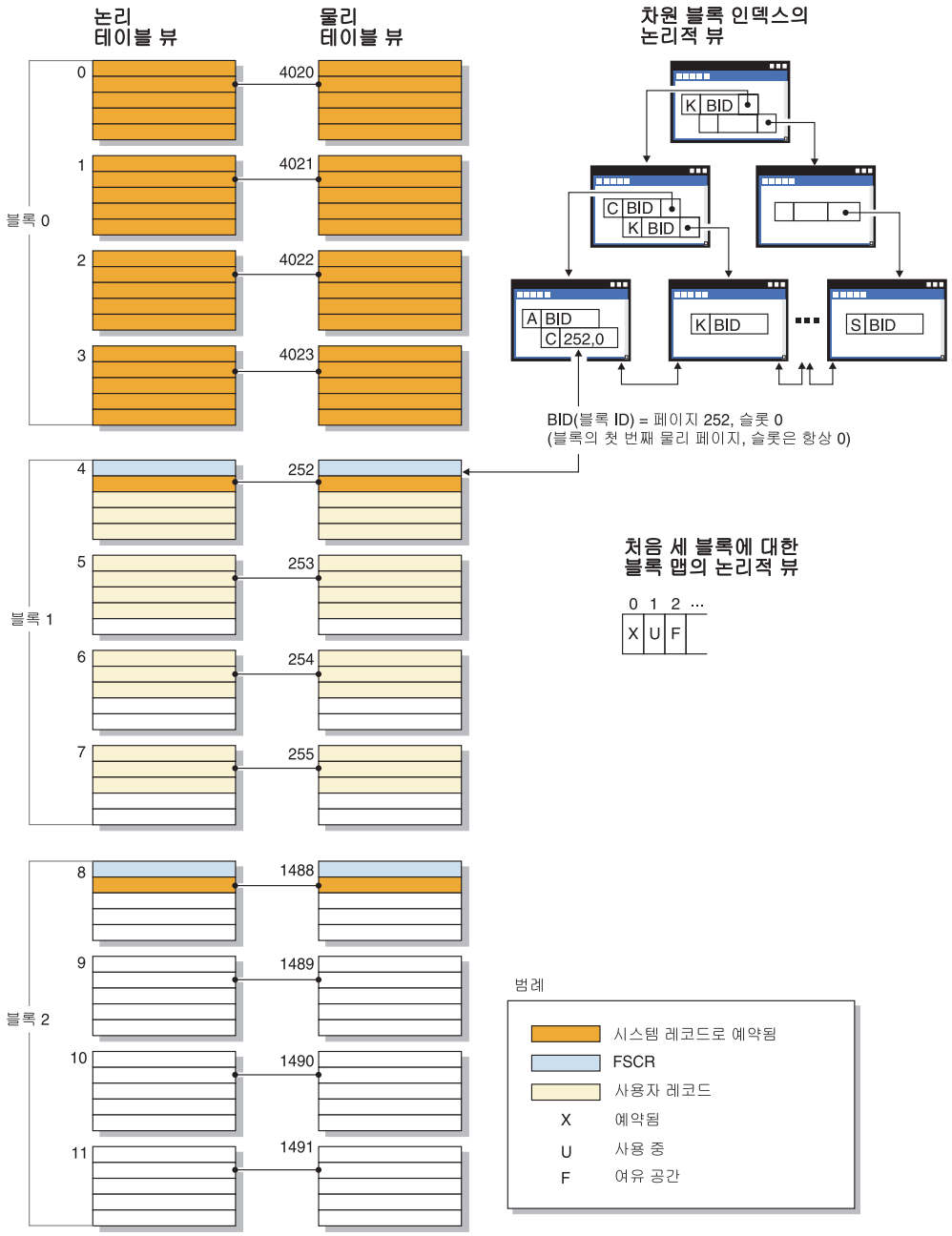


그림 10. MDC 테이블의 논리 테이블, 레코드 및 인덱스 구조

첫 번째 블록에는 DB2 서버가 테이블을 관리하는 데 사용하는 FSCR(Free Space Control Record)을 포함한 특수 내부 레코드가 들어 있습니다. 연속 블록에서 첫 번째 페이지에 FSCR이 있습니다. FSCR은 블록의 각 페이지에 있는 새 레코드의 여유 공간을 맵핑합니다. 이 사용 가능한 여유 공간은 테이블에 레코드를 삽입할 때 사용됩니다.

이름이 의미하는 바와 같이 MDC 테이블은 여러 차원에서 데이터를 클러스터합니다. 각 차원은 CREATE TABLE문의 ORGANIZE BY DIMENSIONS절에 지정하는 컬럼 또는 컬럼 세트로 판별합니다. MDC 테이블을 작성할 때 다음과 같은 두 개의 인덱스가 자동으로 작성됩니다.

- 단일 차원에 대해 차지하는 각 블록을 지정하는 포인터가 포함된 차원 블록 인덱스
- 모든 차원 키 컬럼이 들어 있으며 삽입 및 갱신 활동 중 클러스터링을 유지보수하는 데 사용되는 복합 블록 인덱스

옵티마이저는 특정 쿼리의 가장 효율적인 액세스 플랜을 판별할 때 차원 블록 인덱스를 사용하는 액세스 플랜을 고려합니다. 쿼리에 차원 값에 대한 술어가 있는 경우 옵티마이저는 차원 블록 인덱스를 사용하여 이러한 값이 포함된 Extent를 식별(및 이러한 Extent에서 페치)할 수 있습니다. Extent는 디스크에서 실제로 인접하는 페이지이므로 입출력이 최소화되고 이로 인해 성능이 향상됩니다.

데이터 액세스 플랜의 분석이 이러한 인덱스로 쿼리 성능이 향상됨을 표시하는 경우 특정 RID 인덱스를 작성할 수도 있습니다.

인덱스

인덱스 구조

데이터베이스 관리 프로그램은 인덱스 저장 시 B+ 트리 구조를 사용합니다.

71 페이지의 그림 11에 표시된 바와 같이 B+ 트리에는 여러 레벨이 있습니다. 『rid』는 레코드 ID(RID)를 나타냅니다.

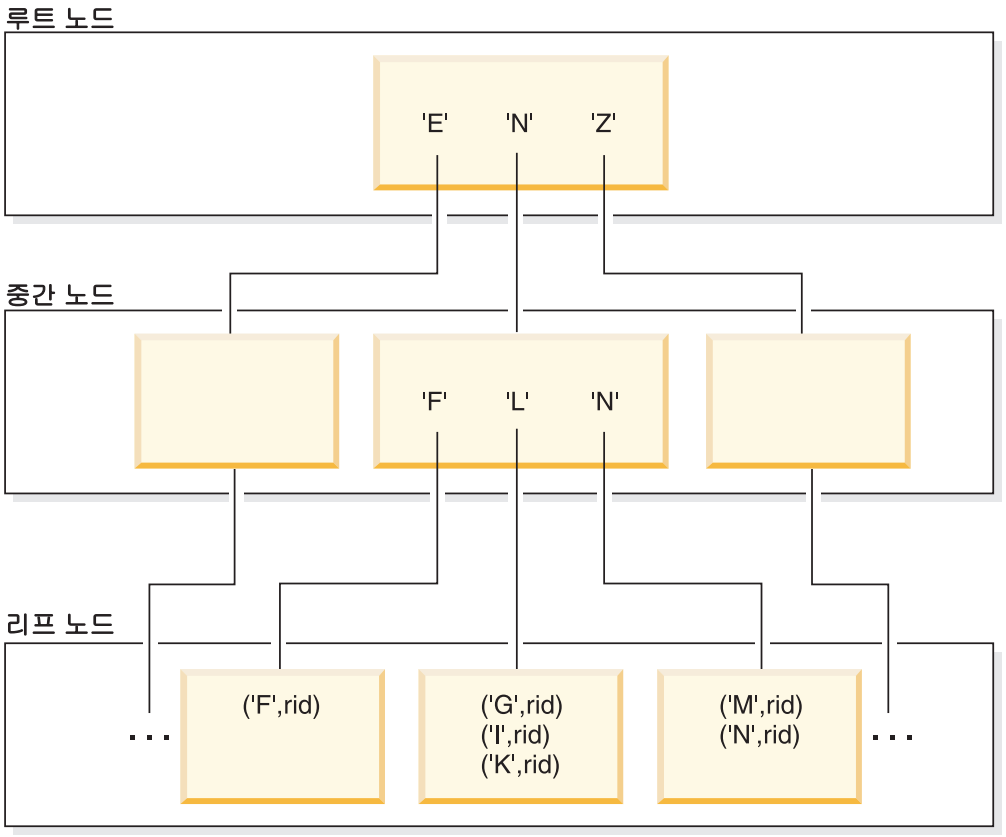


그림 11. B+ 트리 인덱스 구조

맨 위 레벨을 *루트 노드*라고 합니다. 맨 아래 레벨은 해당 데이터가 포함된 테이블 행을 지정하는 포인터를 사용하여 인덱스 키 값을 저장할 *리프 노드*로 구성되어 있습니다. 루트와 리프 노드 레벨 사이의 레벨을 *중간 노드*라고 합니다.

특정 인덱스 키 값을 찾을 때 인덱스 관리 프로그램은 루트 노드부터 시작하여 인덱스 트리를 검색합니다. 루트 노드에는 다음 레벨의 각 노드(중간)에 대한 하나의 키가 있습니다. 이러한 각 키의 값은 다음 레벨의 해당 노드에 대한 기존 최대 키 값입니다. 예를 들어, 그림과 같이 인덱스에 세 개의 레벨이 있다고 가정하십시오. 특정 인덱스 키 값을 찾기 위해 인덱스 관리 프로그램은 루트 노드에서 검색 키 값 이상의 첫 번째 키 값을 검색합니다. 루트 노드 키는 특정 중간 노드를 지정합니다. 인덱스 관리 프로그램은 필요한 인덱스 키가 들어 있는 리프 노드를 찾을 때까지 각 중간 노드를 통해 이 프로시저를 수행합니다.

그림 11에서 찾는 키가 『I』이라고 가정하십시오. 루트 노드에서 『I』 이상의 첫 번째 키가 다음 레벨의 중간 노드를 지정하는 『N』입니다. 해당 중간 노드에서 『I』 이상의 첫 번째 키가 『L』이며 『I』의 인덱스 키 및 해당 RID를 찾을 수 있는 특정 리프 노드를 지정합니다. RID는 기본 테이블에서 해당 행을 식별합니다.

리프 노드 레벨에는 이전 리프 노드를 지정하는 포인터가 포함될 수도 있습니다. 이러한 포인터로 인덱스 관리 프로그램은 한 방향으로 리프 노드를 스캔하여 범위에서 하

나의 값을 찾은 후 값의 범위를 검색할 수 있습니다. 한 방향으로 스캔하는 기능은 ALLOW REVERSE SCANS 옵션을 사용하여 인덱스를 작성한 경우에만 가능합니다.

MDC(다차원 클러스터링) 테이블의 경우 테이블에 대해 지정하는 각 클러스터링 차원마다 블록 인덱스가 자동으로 작성됩니다. 복합 블록 인덱스도 작성됩니다. 이 인덱스에는 테이블의 차원과 관련된 각 컬럼의 키 파트가 포함됩니다. 이러한 인덱스에는 RID 대신 블록 ID(BID)를 지정하는 포인터가 있으며 데이터 액세스 향상 기능을 제공합니다.

인덱스의 리프 페이지에서 각 RID마다 저장되는 1바이트 *ridFlag*는 RID를 논리적으로 삭제된 것으로 표시하는 데 사용되며 이 RID는 나중에 실제로 제거될 수 있습니다. 인덱스에서 각 가변 길이 컬럼마다 추가적인 1바이트는 컬럼 값의 실제 길이를 저장합니다. 갱신 또는 삭제 조작이 커미트된 후 삭제된 것으로 표시된 키가 제거될 수 있습니다.

인덱스 정리 및 유지보수

인덱스를 작성한 후 인덱스를 간단하게 유지하고 잘 구성하지 않으면 시간이 지남에 따라 성능이 저하될 수 있습니다.

다음의 권장사항은 인덱스를 가능하면 작고 효율적으로 유지하는 데 도움이 됩니다.

- 온라인 인덱스 조각 모음 사용

  MINPCTUSED절을 사용하여 인덱스를 작성하십시오. 필요하면 기존의 인덱스를 삭제하고 재작성하십시오.
- 커미트를 자주 수행하거나 명시적으로 또는 잠금 에스컬레이션을 통해 테이블 레벨 X 잠금을 획득하십시오(커미트를 자주 수행할 수 없는 경우).

  삭제된 것으로 표시된 인덱스 키는 커미트 후 테이블에서 실제로 제거할 수 있습니다. 테이블에 대한 X 잠금을 통해 삭제된 키가 삭제된 것으로 표시되었을 때 실제로 제거할 수 있습니다. 아래를 참조하십시오.
- REORGCHK 명령을 사용하여 인덱스 또는 테이블을 재구성하는 시기와 REORG INDEXES 명령을 CLEANUP ONLY 옵션과 함께 사용하는 시기를 판별하십시오.

  재구성 중 인덱스에 대한 읽기 및 쓰기 액세스를 허용하려면 REORG INDEXES 명령을 ALLOW WRITE ACCESS 옵션과 함께 사용하십시오. 파티션된 테이블의 경우, REORG INDEXES...ALL 명령의 ALLOW WRITE ACCESS절은 CLEANUP ONLY절도 지정된 경우에만 지정할 수 있습니다.

삭제된 것으로 표시된 인덱스 키가 정리됩니다.

- 연속 삽입, 갱신 또는 삭제 활동 중

키 삽입 중 삭제된 것으로 표시되고 커미트된 것으로 인식된 키는 페이지 분할을 수행해야 할 필요가 없어지고 인덱스의 크기 증가를 예방할 수 있는 경우 정리됩니다.

키 삭제 중 페이지의 모든 키가 삭제된 것으로 표시된 경우 모든 키가 삭제된 것으로 표시되었으며 이러한 모든 삭제가 커미트된 다른 인덱스 페이지를 찾으려고 시도합니다. 이러한 페이지를 찾으면 인덱스 트리에서 이를 삭제합니다. 키 삭제 시 테이블에 대한 X 잠금이 있는 경우에는 키는 삭제된 것으로 표시될 뿐만 아니라 실제로 삭제됩니다. 실제 삭제 중 동일한 페이지에서 삭제된 키는 삭제된 것으로 표시되고 커미트된 것으로 인식된 경우에도 제거됩니다.

- REORG INDEXES 명령을 CLEANUP 옵션과 함께 실행하는 경우

  CLEANUP ONLY PAGES 옵션은 모든 키가 삭제된 것으로 표시되고 커미트된 것으로 인식된 인덱스 페이지를 검색하고 사용 가능하게 합니다.

  CLEANUP ONLY ALL 옵션은 모든 키가 삭제된 것으로 표시되고 커미트된 것으로 인식된 인덱스 페이지를 사용 가능하게 할 뿐만 아니라 삭제된 것으로 표시되고 일부 삭제되지 않은 RID가 포함된 페이지에서 커미트된 것으로 인식된 레코드 ID(RID)도 제거합니다. 또한 이 옵션은 인접한 리프 페이지를 병합하려고 합니다(이와 같이 병합하여 최소한 PCTFREE 여유 공간이 있는 병합된 리프 페이지가 작성될 경우). 인덱스가 작성될 때 PCTFREE 값이 정의됩니다. 디폴트 PCTFREE 값은 10%입니다. 두 페이지를 병합할 수 있는 경우 한 페이지가 사용 가능해집니다.

  파티션된 테이블의 경우, 비동기 인덱스 정리가 완료된 후 runstats 유틸리티 호출을 권장합니다. 테이블에 분리된 데이터 파티션이 있는지 판별하려면 SYSCAT.DATAPARTITIONS 카탈로그 뷰에서 STATUS 필드를 쿼리하고 값 D(분리) 또는 I(인덱스 정리)를 찾으십시오.
- 인덱스가 재빌드되는 경우(또는 파티션된 인덱스의 경우, 인덱스 파티션이 재빌드되는 경우)

  인덱스를 재빌드하는 유틸리티는 다음과 같습니다.
  - CLEANUP 옵션을 사용하지 않는 REORG INDEXES
  - INPLACE 옵션을 사용하지 않는 REORG TABLE
  - REPLACE 옵션을 사용한 IMPORT
  - INDEXING MODE REBUILD 옵션을 사용한 LOAD

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 분리를 촉진하고, 파티션된 테이블에 하나 이상의 파티션되지 않은 인덱스가 포함되어 있는 경우 시작됩니다. 이 경우, AIC는 분리된 데이터 파티션을 참조하는 모든 파티션되지 않은 인덱스 항목과 의사 삭제된 항목을 제거합니다. 인덱스가 모두 정리된 후, 분리된 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 예방하게 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 스로틀링 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
Type                              = ASYNCHRONOUS INDEX CLEANUP
Database Name                     = WSDB
Partition Number                  = 0
Description                       = Table: USER1.SALES, Index: USER1.I2
Start Time                        = 12/15/2005 11:15:01.967939
State                             = Executing
Invocation Type                   = Automatic
Throttling:
   Priority                       = 50
Progress Monitoring:
      Total Work                  = 5 pages
      Completed Work              = 0 pages
      Start Time                  = 12/15/2005 11:15:01.979033

ID                                = 1
Type                              = ASYNCHRONOUS INDEX CLEANUP
Database Name                     = WSDB
Partition Number                  = 0
Description                       = Table: USER1.SALES, Index: USER1.I1
Start Time                        = 12/15/2005 11:15:01.978554
State                             = Executing
Invocation Type                   = Automatic
Throttling:
   Priority                       = 50
Progress Monitoring:
      Total Work                  = 5 pages
      Completed Work              = 0 pages
      Start Time                  = 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.

State 필드는 클리너의 현재 상태를 표시합니다. 일반 상태는 실행이지만, 잠금 경쟁 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(다차원 클러스터링(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 조정 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 스로틀링 기능과도 통합됩니다. 디폴트로, 각 클리너는 50의 유틸리티 영향 우선순위를 갖습니다(허용 가능한 값은 1-100이며, 0은 비스로틀링을 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 의해 진행이 표시됩니다. 각 단계는 하나의 RID 인덱스를 나타냅니다.

```
ID                                       = 2
Type                                     = MDC ROLLOUT INDEX CLEANUP
Database Name                            = WSDB
Partition Number                         = 0
Description                              = TABLE.<schema_name>.<table_name>
Start Time                               = 06/12/2006 08:56:33.390158
State                                    = Executing
Invocation Type                          = Automatic
Throttling:
```

```
   Priority                       = 50
Progress Monitoring:
   Estimated Percentage Complete = 83
   Phase Number                   = 1
      Description                 = <schema_name>.<index_name>
      Total Work                  = 13 pages
      Completed Work              = 13 pages
      Start Time                  = 06/12/2006 08:56:33.391566
   Phase Number                   = 2
      Description                 = <schema_name>.<index_name>
      Total Work                  = 13 pages
      Completed Work              = 13 pages
      Start Time                  = 06/12/2006 08:56:33.391577
   Phase Number                   = 3
      Description                 = <schema_name>.<index_name>
      Total Work                  = 9 pages
      Completed Work              = 3 pages
      Start Time                  = 06/12/2006 08:56:33.391587
```

온라인 인덱스 조각 모음

온라인 인덱스 조각 모음은 인덱스 리프 페이지에서 사용한 스페이스의 최소값에 대해 사용자가 정의할 수 있는 임계값으로 사용 가능합니다.

인덱스 키가 리프 페이지에서 삭제되었으며 이 임계값을 초과한 경우 인접한 리프 페이지를 점검하여 두 개의 리프 페이지를 병합할 수 있는지 여부를 판별합니다. 페이지에 충분한 스페이스가 있으며 두 개의 인접한 페이지를 병합할 수 있는 경우 백그라운드에서 병합이 즉시 발생하고 이로 인해 비어 있는 인덱스 리프 페이지가 삭제됩니다.

기존 인덱스에 온라인으로 병합할 수 있는 기능이 필요한 경우 삭제한 후 CREATE INDEX문에 MINPCTUSED절을 지정하여 다시 작성해야 합니다. 두 개의 인접한 인덱스 리프 페이지를 병합하는 것이 목적이므로 MINPCTUSED의 권장 값은 50 미만입니다. 디폴트인 0 값을 사용하면 온라인 조각 모음이 사용 불가능합니다.

인덱스 비리프 페이지는 온라인 인덱스 조각 모음 중 병합되지 않습니다. 그러나 비어 있는 비리프 페이지가 삭제되며 동일한 테이블의 다른 인덱스에서 재사용할 수 있게 됩니다. 데이터베이스 관리 스페이스(DMS) 스토리지 모델에서 다른 오브젝트를 위해 이러한 비리프 페이지를 사용 가능하게 하거나 시스템 관리 스페이스(SMS) 스토리지 모델에서 디스크 스페이스를 사용 가능하게 하려면 테이블 및 인덱스의 전체 재구성을 수행하십시오. 인덱스가 최소한의 크기로 작아집니다. 온라인 인덱스 조각 모음 중 인덱스의 레벨 수는 줄어들지 않습니다.

테이블에 X 잠금이 있는 경우 키 삭제 중 페이지에서 키가 실제로 제거됩니다. 이 경우 온라인 인덱스 조각 모음이 적용됩니다. 그러나 키 삭제 중 테이블에 X 잠금이 없는 경우 키는 삭제된 것으로 표시되지만 실제로 인덱스 페이지에서 제거되지 않으며 인덱스 조각 모음을 시도하지 않습니다.

MINPCTUSED의 값에 관계없이 인덱스 조각 모음을 수행하려면 REORG INDEXES 명령을 CLEANUP ONLY ALL 옵션과 함께 호출하십시오. 병합된 페이지에 최소한 PCTFREE 여유 공간이 남게 되면 두 개의 인접한 리프 페이지가 병합됩니다. PCTFREE는 인덱스 작성 시에 지정할 수 있으며 디폴트값은 10(%)입니다.

관계형 인덱스를 사용한 성능 향상

인덱스를 사용하면 테이블 데이터에 액세스할 때 성능을 향상시킬 수 있습니다. 관계형 인덱스는 관계형 데이터에 액세스할 때 사용하며 XML을 통한 인덱스는 XML 데이터에 액세스할 때 사용합니다.

쿼리 옵티마이저가 관계형 테이블 데이터에 액세스할 때 관계형 인덱스를 사용할 것인지 여부를 결정하지만 성능을 향상시킬 수 있는 인덱스를 결정하고 이러한 인덱스를 작성하는 작업은 사용자가 수행해야 합니다. MDC(다차원 클러스터링) 테이블을 작성할 때 각 차원에 대해 자동으로 작성되는 차원 블록 인덱스 및 복합 블록 인덱스의 경우는 예외입니다.

관계형 인덱스를 작성한 후 또는 프리페치 크기를 변경한 후 runstats 유틸리티를 실행하여 새 인덱스 통계를 수집하십시오. 통계를 최신 상태로 유지하기 위해 runstats 유틸리티를 주기적으로 실행해야 합니다. 인덱스에 대한 최신 통계가 없으면 옵티마이저는 쿼리에 가장 적합한 데이터 액세스 플랜을 판별할 수 없습니다.

특정 패키지에서 관계형 인덱스를 사용하는지 여부를 판별하려면 Explain 기능을 사용하십시오. 하나 이상의 SQL문에서 활용할 수 있는 관계형 인덱스에 대한 자세한 내용을 보려면 db2advis 명령을 사용하여 디자인 어드바이저를 실행하십시오.

인덱스가 없는 경우에 비해 관계형 인덱스를 사용하는 경우 장점

테이블의 인덱스가 없는 경우 SQL 쿼리에 참조된 각 테이블의 테이블 스캔을 수행해야 합니다. 테이블 스캔에서는 각 행에 순차적으로 액세스해야 하므로 테이블이 크면 스캔하는 시간도 오래 걸립니다. 테이블 스캔은 테이블에 있는 대부분의 행이 필요한 복합 쿼리의 경우에 효율적이지만 인덱스 스캔은 일부 테이블 행만 리턴하는 쿼리의 경우 테이블 행에 보다 효율적으로 액세스할 수 있습니다.

SELECT문에 관계형 인덱스 컬럼이 참조된 경우와 옵티마이저가 인덱스 스캔이 테이블 스캔보다 빠를 것으로 추정하는 경우 옵티마이저는 인덱스 스캔을 선택합니다. 인덱스 파일은 보통 크기가 작으며 전체 테이블보다 읽는 시간이 많이 필요하지 않습니다(특히 테이블이 클 경우). 또한 전체 인덱스를 스캔할 필요가 없습니다. 인덱스에 적용되는 술어는 데이터 페이지에서 읽어야 하는 행 수를 줄입니다.

출력에서 순서 요구사항을 인덱스 컬럼과 일치시킬 수 있는 경우 컬럼 순서에서 인덱스를 스캔하면 정렬 조작을 수행하지 않아도 행을 올바른 순서로 검색할 수 있습니다. 쿼리 중인 테이블에 관계형 인덱스가 존재한다고 해서 결과 세트가 정렬되지는 않는다는 점을 참고하십시오. ORDER BY절을 사용해야만 결과 세트가 순서대로 정렬됩니다.

관계형 인덱스에는 인덱스화된 행에서 인덱스화되지 않은 컬럼인 포함 컬럼도 있을 수 있습니다. 이러한 컬럼을 사용하면 옵티마이저는 테이블 자체에 액세스하지 않아도 인덱스에서만 필요한 정보를 검색할 수 있습니다.

인덱스가 없는 경우에 비해 관계형 인덱스를 사용하는 경우 단점

인덱스는 액세스 시간을 상당히 줄일 수 있지만 성능에 막대한 영향을 미칠 수도 있습니다. 인덱스를 작성하기 전에 디스크 공간 및 처리 시간에 대한 여러 인덱스의 영향을 고려하십시오. 응용프로그램의 필요에 맞게 인덱스를 주의하여 선택하십시오.

- 각 인덱스에는 스토리지 스페이스가 필요합니다. 정확한 양은 테이블의 *크기*와 관계형 인덱스에 있는 컬럼의 *크기* 및 수에 따라 다릅니다.
- 테이블에 대한 각 삽입 또는 삭제 조작을 수행하려면 해당 테이블에서 각 인덱스를 추가로 갱신해야 합니다. 인덱스 키의 값을 변경하는 각 갱신 조작의 경우에도 마찬가지입니다.
- 각 관계형 인덱스는 옵티마이저가 고려해야 하는 다른 잠재적인 액세스 플랜을 나타내며 이로 인해 쿼리 컴파일 시간이 늘어납니다.

관계형 인덱스 계획 추가 정보

적절하게 설계된 인덱스를 사용하면 쿼리는 관계형 데이터에 쉽게 액세스할 수 있습니다.

디자인 어드바이저(db2advis 명령)를 사용하여 특정 쿼리 또는 워크로드를 정의한 쿼리 세트에 가장 적합한 인덱스를 찾으십시오. 이 도구는 역방향 스캔에 사용할 수 있는 포함 컬럼 또는 인덱스와 같은 성능 향상 권장사항을 작성할 수 있습니다.

다음 지침은 유용한 관계형 인덱스를 작성하는 경우 도움이 될 수도 있습니다.

- 효율적인 데이터 검색
  - 데이터 검색을 향상시키려면 고유 인덱스에 *포함 컬럼*을 추가하십시오. 다음과 같은 컬럼이 해당됩니다.
    - 자주 액세스하며 인덱스 전용 액세스의 장점을 활용하는 컬럼
    - 인덱스 스캔 범위를 제한할 필요가 없는 컬럼
    - 인덱스 키의 순서 또는 고유성에 영향을 주지 않는 컬럼

    예를 들어, 다음과 같습니다.

    ```
    create unique index idx on employee (workdept) include (lastname)
    ```

    LASTNAME을 인덱스 키의 일부분이 아닌 포함 컬럼으로 지정하는 것은 LASTNAME이 인덱스의 리프 페이지에만 저장됨을 의미합니다.
  - 자주 실행하는 쿼리의 WHERE절에서 사용되는 컬럼에 관계형 인덱스를 작성하십시오.

다음 예에서 WHERE절은 WORKDEPT의 인덱스를 활용할 가능성이 높습니다 (WORKDEPT 컬럼에 여러 중복 값이 포함되지 않은 경우).

```
where workdept='A01' or workdept='E21'
```

– 쿼리에 참조된 각 컬럼의 이름을 지정하는 복합 키를 사용하여 관계형 인덱스를 작성하십시오. 이러한 방식으로 인덱스를 지정하는 경우 인덱스에서만 관계형 데이터를 검색할 수 있으므로 테이블에 액세스하는 것보다 효율적입니다.

예를 들어, 다음 쿼리를 참조하십시오.

```
select lastname
  from employee
  where workdept in ('A00','D11','D21')
```

EMPLOYEE 테이블의 WORKDEPT 및 LASTNAME 컬럼에 관계형 인덱스가 정의된 경우 전체 테이블이 아닌 인덱스를 스캔하여 쿼리를 효율적으로 처리할 수 있습니다. 술어는 WORKDEPT를 참조하므로 이 컬럼이 관계형 인덱스의 첫 번째 키 컬럼이 됩니다.

• 효율적인 테이블 검색

자주 사용하는 순서에 따라 오름차순 또는 내림차순을 결정하십시오. CREATE INDEX문에서 ALLOW REVERSE SCANS 옵션을 지정하면 값을 역방향으로 검색할 수 있지만 지정된 인덱스 순서로 스캔하는 것이 역방향 스캔보다 나을 수 있습니다.

• 효율적인 대형 테이블 액세스

관계형 인덱스를 사용하여 SYSCAT.TABLES 카탈로그 뷰의 NPAGES 컬럼에 기록된 바와 같이 데이터 페이지 수가 조금 더 있는 테이블에 대해 자주 쿼리를 수행하도록 최적화하십시오. 다음을 수행해야 합니다.

– 테이블을 조인하는 데 사용할 컬럼에 대한 인덱스를 작성하십시오.

– 주기적으로 특정 값을 검색할 컬럼에 대해 인덱스를 작성하십시오.

• 갱신 또는 삭제 조작의 성능 향상

– 상위 테이블에 대해 이러한 조작의 성능을 향상시키려면 외부 키에 관계형 인덱스를 작성하십시오.

– REFRESH IMMEDIATE 및 INCREMENTAL 구체화된 쿼리 테이블(MQT)에 대해 이러한 조작의 성능을 향상시키려면 MQT 정의의 GROUP BY절에 있는 컬럼으로 구성된 MQT의 내재 고유 키에 대해 고유 관계형 인덱스를 작성하십시오.

• 조인 성능 향상

복수 컬럼 관계형 인덱스에 첫 번째 키 컬럼에 대한 선택사항이 여러 개인 경우 equijoin 술어(*expression1 = expression2*)와 자주 지정된 컬럼 또는 최대 구별 값 수가 있는 컬럼을 첫 번째 키 컬럼으로 사용하십시오.

- 정렬
  - 정렬 조작을 신속하게 수행하려면 관계형 데이터를 정렬하기 위해 자주 사용하는 컬럼에 대해 관계형 인덱스를 작성하십시오.
  - 정렬을 방지하려면 가능할 때마다 CREATE INDEX문을 사용하여 1차 키 및 고유 키를 정의하십시오.
  - 자주 실행하는 쿼리에서 요구하는 순서대로 행을 정렬하도록 관계형 인덱스를 작성하십시오. 정렬은 DISTINCT, GROUP BY 및 ORDER BY절에서 필수사항입니다.

    다음 예에서는 DISTINCT절을 사용합니다.

    ```
    select distinct workdept
      from employee
    ```

    데이터베이스 관리 프로그램은 WORKDEPT 컬럼에 정의된 인덱스를 사용하여 중복 값을 제거할 수 있습니다. 동일한 인덱스를 사용하여 GROUP BY절을 사용하는 다음 예와 같이 값들을 그룹화할 수도 있습니다.

    ```
    select workdept, average(salary)
      from employee
      group by workdept
    ```

- 새로 삽입된 행 클러스터링 및 페이지 분할 방지

  클러스터링 인덱스를 정의하십시오. 테이블을 재구성할 필요가 거의 없습니다. CREATE TABLE문에서 PCTFREE 옵션을 사용하여 행을 적절하게 삽입할 수 있도록 각 페이지에 남아 있어야 하는 여유 공간을 지정하십시오. LOAD 명령에 pagefreespace 파일 유형 수정자를 지정할 수도 있습니다.

- 인덱스 유지보수 비용 및 스토리지 스페이스 절약
  - 기타 기존 인덱스의 부분적 키인 인덱스를 작성하지 않도록 방지하십시오. 예를 들어, A, B 및 C 컬럼에 대한 인덱스가 있는 경우 A 및 B 컬럼에 대한 다른 인덱스는 일반적으로 유용하지 않습니다.
  - 여러 컬럼에 대해 임의의 인덱스를 작성하지 마십시오. 불필요한 인덱스는 공간을 낭비할 뿐만 아니라 준비 시간도 길어질 수 있습니다.
    - 온라인 트랜잭션 처리(OLTP) 환경의 경우 테이블 당 하나 또는 두 개의 인덱스를 작성하십시오.
    - 읽기 전용 쿼리 환경의 경우 테이블 당 다섯 개 이상의 인덱스를 작성할 수도 있습니다.
    - 혼합 쿼리 및 OLTP 환경의 경우 테이블 당 2 - 5개의 인덱스를 작성하면 충분합니다.

- 온라인 인덱스 조각 모음 사용

  관계형 인덱스를 작성할 때 MINPCTUSED 옵션을 사용하십시오. MINPCTUSED를 사용하면 온라인 인덱스 조각 모음이 가능합니다. 이는 인덱스 리프 페이지에서 사용 중이어야 하는 최소 공간량을 지정합니다.

관계형 인덱스 성능 추가 정보

관계형 인덱스를 올바로 수행하기 위해 여러 조치를 취할 수 있습니다.

- 대용량 유틸리티 힙을 지정하십시오.

  관계형 인덱스를 작성 또는 재구성 중인 테이블에 대해 여러 갱신 활동이 예상되는 경우 이러한 조작의 속도를 높이는 데 유용한 대용량 유틸리티 힙을 구성하십시오 (**util_heap_sz** 데이터베이스 구성 매개변수).
- 대칭형 멀티프로세서(SMP) 환경에서 정렬 오버플로우를 방지하기 위해 **sheapthres** 데이터베이스 관리 프로그램 구성 매개변수의 값을 늘리십시오.
- 관계형 인덱스의 개별 테이블 스페이스를 작성하십시오.

  속도가 빠른 물리 디바이스에서 인덱스 테이블 스페이스를 작성하거나 다른 버퍼 풀에 인덱스 테이블 스페이스를 지정하면 인덱스 페이지가 데이터 페이지와 경합하지 않으므로 버퍼에서 인덱스 페이지를 오래 유지할 수 있습니다.

  인덱스에 다른 테이블 스페이스를 사용하는 경우 인덱스에 맞게 해당 테이블 스페이스의 구성을 최적화할 수 있습니다. 인덱스는 보통 테이블보다 작으며 일부 컨테이너에 분산되어 있으므로 인덱스의 Extent 크기는 보통 작습니다. 쿼리 옵티마이저는 액세스 플랜을 선택할 때 테이블 스페이스가 있는 디바이스의 속도를 고려합니다.
- 높은 수준의 클러스터링을 확인하십시오.

  SQL문에서 결과를 정렬해야 하는 경우(예: ORDER BY, GROUP BY 또는 DISTINCT절이 포함된 경우) 옵티마이저는 다음과 같은 경우에 사용 가능한 인덱스를 선택하지 않을 수도 있습니다.
  - 인덱스 클러스터링이 낮은 경우. 특정 인덱스에서 클러스터링 수준에 대한 정보는 SYSCAT.INDEXES 카탈로그 뷰의 CLUSTERRATIO 및 CLUSTERFACTOR 컬럼을 쿼리하십시오.
  - 테이블이 너무 작아서 테이블을 스캔하고 메모리에서 결과 세트를 정렬하는 것이 저렴한 경우.
  - 테이블에 액세스하는 경쟁 인덱스가 있는 경우.

  클러스터링 인덱스는 runstats 유틸리티에서 수집한 CLUSTERRATIO 또는 CLUSTERFACTOR 통계를 개선하여 특정 순서의 데이터를 유지보수합니다. 클러

스터링 인덱스를 작성한 후 오프라인 테이블 reorg 조작을 수행하십시오. 일반적으로 테이블은 하나의 인덱스에서만 클러스터링할 수 있습니다. 클러스터링 인덱스를 빌드한 후 추가 인덱스를 빌드하십시오.

테이블의 PCTFREE 값은 나중에 데이터를 삽입할 경우를 위해 페이지에서 비워둘 공간의 양을 결정하므로 삽입된 데이터를 적절하게 클러스터링할 수 있습니다. 테이블의 PCTFREE 값을 지정하지 않으면 재구성은 모든 추가 공간을 제거합니다.

갱신 조작 중 데이터 클러스터링을 유지하지 않는 RCT(range-clustered table)의 경우는 예외입니다. 즉, 클러스터링 인덱스의 키 값이 변경되도록 레코드를 갱신하는 경우 클러스터링 순서를 유지하기 위해 레코드가 반드시 새 페이지로 이동되지는 않습니다. 클러스터링을 유지하기 위해 레코드를 삭제한 후 갱신 조작을 사용하는 대신 레코드의 갱신된 버전을 삽입하십시오.

- 테이블 및 인덱스 통계를 최신 상태로 유지하십시오.

  새 관계형 인덱스를 작성한 후 runstats 유틸리티를 실행하여 인덱스 통계를 수집하십시오. 이러한 통계는 옵티마이저가 인덱스 사용이 데이터 액세스 성능을 향상시킬 수 있는지 여부를 판별하는 데 유용합니다.
- 온라인 인덱스 조각 모음 사용

  온라인 인덱스 조각 모음은 관계형 인덱스의 MINPCTUSED가 0보다 큰 값으로 설정된 경우 사용할 수 있습니다. 온라인 인덱스 조각 모음은 페이지의 여유 공간량이 지정된 MINPCTUSED 값 아래로 낮아지면 인덱스 리프 페이지를 병합하여 인덱스를 압축할 수 있도록 지원합니다.
- 필요한 경우 관계형 인덱스 재구성

  테이블을 갱신할 경우 인덱스 페이지 프리페치로 효율성이 떨어질 수 있으므로 최상의 성능을 얻으려면 주기적으로 인덱스를 재구성하십시오.

  인덱스를 재구성하려면 삭제(drop)하고 다시 작성하거나 reorg 유틸리티를 사용하십시오.

  자주 재구성을 실행해야 하는 불편함을 줄이기 위해 CREATE INDEX문에서 적합한 PCTFREE 값을 지정하여 작성 중 각 인덱스 리프 페이지에서 여유 공간을 충분히 유지하십시오. 이후 활동 중 페이지 연속성을 방해하여 인덱스 페이지 프리페치 효율성을 떨어뜨리는 인덱스 페이지 분할 가능성을 줄여서 인덱스에 레코드를 삽입할 수 있습니다. 관계형 인덱스를 작성할 때 지정한 PCTFREE 값은 인덱스 재구성 시 보존됩니다.
- 관계형 인덱스 사용에 대한 Explain 정보 분석

가장 자주 사용하는 쿼리에 대해 EXPLAIN문을 주기적으로 발행하고 각 관계형 인덱스를 최소한 한 번 사용하는지 검증하십시오. 쿼리에서 인덱스를 사용하지 않는 경우 해당 인덱스를 삭제하십시오.

Explain 정보를 사용하면 스캔 중인 대형 테이블을 중첩된 루프 조인의 내부 테이블로 처리하는지 여부를 판별할 수도 있습니다. 처리하는 경우 Join 술어 컬럼의 인덱스가 누락되었거나 Join 술어를 적용할 때 효과가 없는 것으로 간주됩니다.

- *크기*가 매우 다양한 테이블을 『일시적』으로 선언

  *일시적 테이블*은 런타임에 카디널리티(cardinality)가 매우 다양할 수 있는 테이블입니다. 이러한 유형의 테이블에 대해 옵티마이저는 인덱스 스캔 대신 테이블 스캔을 선호하는 액세스 플랜을 생성할 수 있습니다.

  ALTER TABLE문을 VOLATILE절과 함께 사용하여 이러한 테이블을 일시적으로 선언하십시오. 다음과 같은 경우에 통계에 관계없이 옵티마이저는 이러한 테이블에 대해 테이블 스캔 대신 인덱스 스캔을 사용합니다.

  – 참조된 모든 컬럼이 인덱스의 일부분인 경우
  – 인덱스 스캔 중 인덱스가 술어를 적용할 수 있는 경우

  유형이 지정된 테이블의 경우 ALTER TABLE...VOLATILE문은 유형이 지정된 테이블 계층 구조의 루트 테이블에만 지원됩니다.

파티셔닝 및 클러스터링

파티션된 테이블에서의 인덱스 동작

파티션된 테이블의 인덱스는 파티션되지 않은 테이블의 인덱스와 유사하게 작동하지만 파티션된 인덱스인지 또는 파티션되지 않은 인덱스인지 여부에 따라 다른 스토리지 모델을 사용하여 저장됩니다.

파티션되지 않은 일반 테이블의 인덱스는 모두 공유 인덱스 오브젝트에 상주하는 반면, 파티션된 테이블에서 *파티션되지 않은 인덱스*는 데이터 파티션이 여러 테이블 스페이스에 걸쳐 있다 하더라도 단일 테이블 스페이스에서 자체 인덱스 오브젝트에 작성됩니다. 데이터베이스 관리 스페이스(DMS)와 시스템 관리 스페이스(SMS) 테이블 스페이스 모두 테이블 데이터와 다른 위치에서의 인덱스 사용을 지원합니다. 각 파티션되지 않은 인덱스는 대형 테이블 스페이스를 비롯한 해당 고유 테이블 스페이스에 배치될 수 있습니다. 각 인덱스 테이블 스페이스에서는 데이터 파티션과 동일한 스토리지 메커니즘을 사용해야 합니다(DMS 또는 SMS). 대형 테이블 스페이스의 인덱스는 최대 2^{29}페이지를 포함할 수 있습니다. 모든 테이블 스페이스는 동일한 데이터베이스 파티션 그룹에 있어야 합니다.

*파티션된 인덱스*는 테이블의 파티셔닝 스킴에 따라 인덱스 데이터가 여러 *인덱스* 파티션으로 나뉘어지는 인덱스 구성 스킴을 사용합니다. 각 인덱스 파티션은 해당 데이터 파티션의 테이블 행만 참조합니다. 주어진 데이터 파티션의 모든 인덱스 파티션은 동일한 인덱스 오브젝트에 상주합니다.

XML 데이터에 대한 사용자 작성 인덱스만 파티션되지 않을 수 있습니다. 시스템 생성 XML 영역은 항상 파티션되지만 시스템 생성 컬럼 경로 인덱스는 항상 파티션되지 않습니다.

파티션되지 않은 인덱스의 장점은 다음과 같습니다.

- 각 인덱스에 대해 다른 테이블 스페이스 특성을 정의하는 기능(예를 들어, 다양한 페이지 크기는 더 나은 스페이스 이용률을 보장하는 데 도움이 될 수 있음)
- 인덱스를 서로 독립적으로 재구성할 수 있다는 점
- 인덱스 삭제(drop) 조작 성능 향상
- 인덱스 데이터로의 보다 효율적인 동시 액세스를 제공하는 데 도움이 되는 감소된 입출력 경쟁
- 개별 인덱스가 삭제될 때 인덱스를 재구성할 필요 없이 시스템에서 스페이스가 즉시 사용 가능하게 된다는 점

파티션된 인덱스의 장점은 다음과 같습니다.

- 향상된 데이터 롤인 및 롤아웃 성능
- 인덱스가 파티션되기 때문에 인덱스 페이지 경쟁이 줄어듬
- 다음 결과를 초래할 수 있는 각 인덱스 파티션에 대한 인덱스 B-트리 구조
  - 인덱스 파티션의 B-트리는 일반적으로 테이블의 모든 데이터를 참조하는 인덱스에 비해 레벨 수가 적기 때문에 삽입, 갱신, 삭제 및 스캔 성능 향상
  - 파티션 제거가 적용될 때 스캔 성능 및 동시성 향상. 파티션된 인덱스 스캔과 파티션되지 않은 인덱스 스캔 둘 다에 파티션 제거를 사용할 수 있지만, 각 인덱스 파티션은 해당 데이터 파티션에 대한 키만 포함하므로 파티션된 인덱스 스캔의 경우 보다 효과적입니다. 따라서 파티션되지 않은 인덱스를 통한 유사 쿼리에 비해 스캔하는 키와 인덱스 페이지 수가 줄어들 수 있습니다.

파티션되지 않은 인덱스는 항상 인덱스 컬럼에 순서를 보존하지만 파티션된 인덱스는 특정 시나리오(예: 파티션된 컬럼이 인덱스 컬럼과 일치하지 않고 두 개 이상의 파티션에 액세스할 경우)에서는 파티션에서 일부 순서를 유실할 수 있습니다.

온라인 인덱스 작성 중에 테이블에 대한 동시 읽기 및 쓰기 액세스가 허용됩니다. 이러한 인덱스가 빌드된 후, 인덱스 작성 중에 이루어진 테이블 변경사항은 새 인덱스에 적용됩니다. 인덱스 작성이 완료되고 트랜잭션이 커미트될 때까지 테이블에 대한 쓰기 액

세스가 차단됩니다. 파티션된 인덱스의 경우, 인덱스 파티션 작성 중에 이루어진 데이터 파티션 변경사항이 적용되는 동안만 각 데이터 파티션은 읽기 전용 액세스로 Quiesce 상태가 됩니다.

파티션된 인덱스는 ALTER TABLE...ATTACH PARTITION문을 사용하여 데이터를 롤링하는 경우에 특히 유용합니다. 파티션되지 않은 인덱스가 있는 경우(테이블에 XML 데이터가 있는 경우 XML 컬럼 경로 인덱스 제외), 파티션 첨부 후에 SET INTEGRITY 문을 실행하십시오. 이는 파티션되지 않은 인덱스 유지보수, 범위 유효성 확인, 제한조건 검사 및 구체화된 쿼리 테이블(MQT) 유지보수에 필요합니다. 파티션되지 않은 인덱스 유지보수는 시간이 걸릴 수 있으며 대량의 로그 스페이스를 필요로 합니다. 이러한 유지보수 비용이 들지 않게 하려면 파티션된 인덱스를 사용하십시오.

그림 12에서는 파티션된 테이블의 파티션되지 않은 두 개의 인덱스(각각의 인덱스는 별도의 상주함)를 표시합니다.

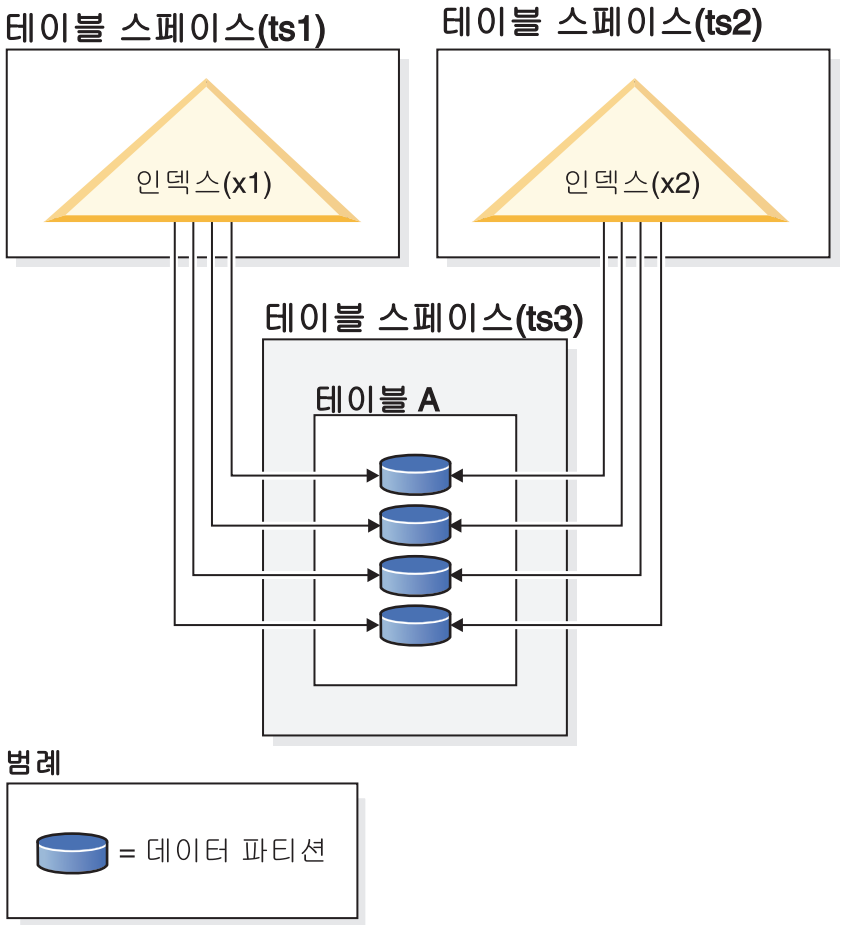


그림 12. 파티션된 테이블의 파티션되지 않은 인덱스

87 페이지의 그림 13에서는 두 개의 데이터베이스 파티션에 걸쳐 있고 단일 테이블 스페이스에 상주하는 파티션된 테이블의 파티션되지 않은 인덱스를 표시합니다.

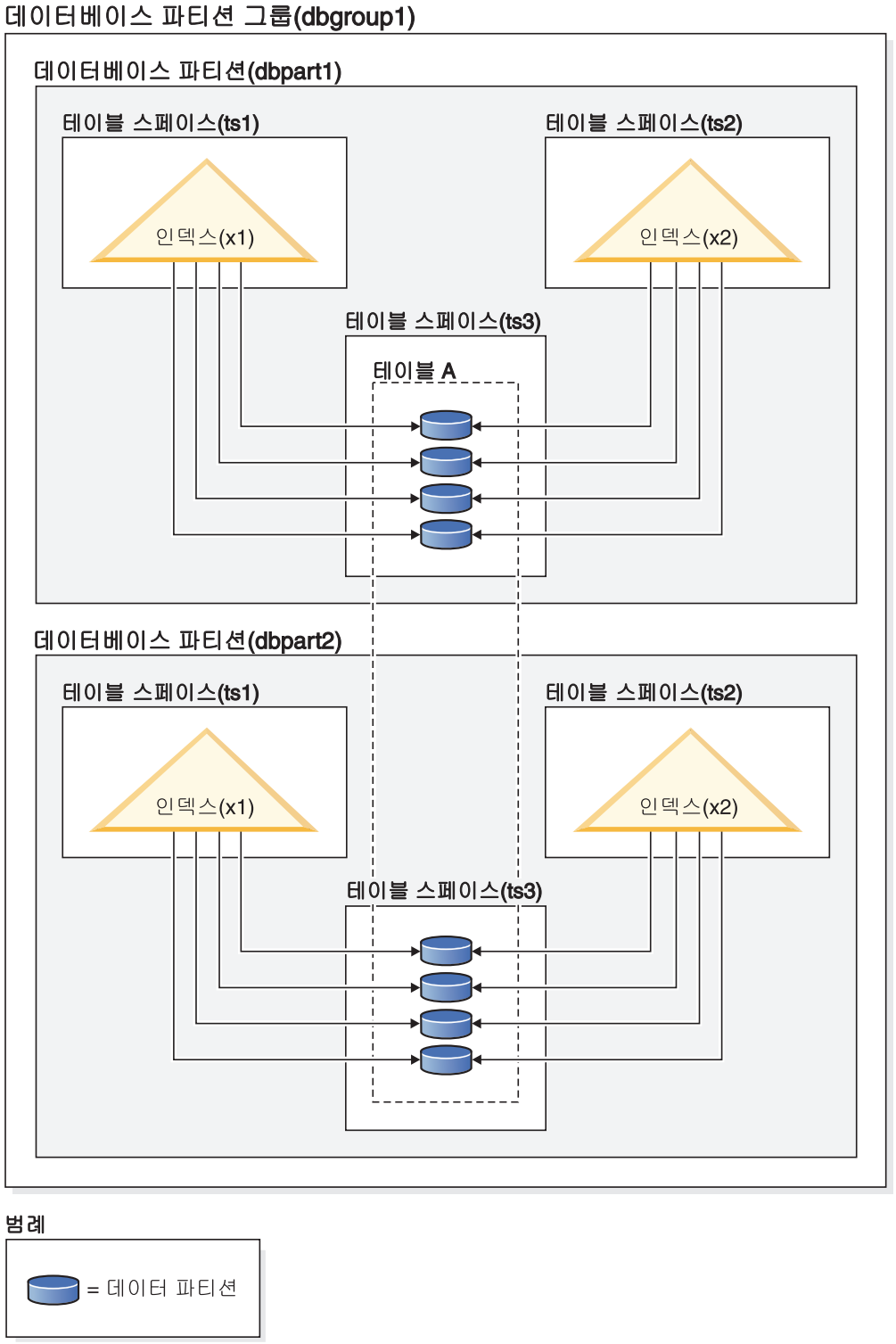


그림 13. 분산되고 파티션된 테이블에서 파티션되지 않은 인덱스

88 페이지의 그림 14에서는 파티션된 테이블의 파티션된 인덱스와 파티션되지 않은 인덱스 혼합을 표시합니다.

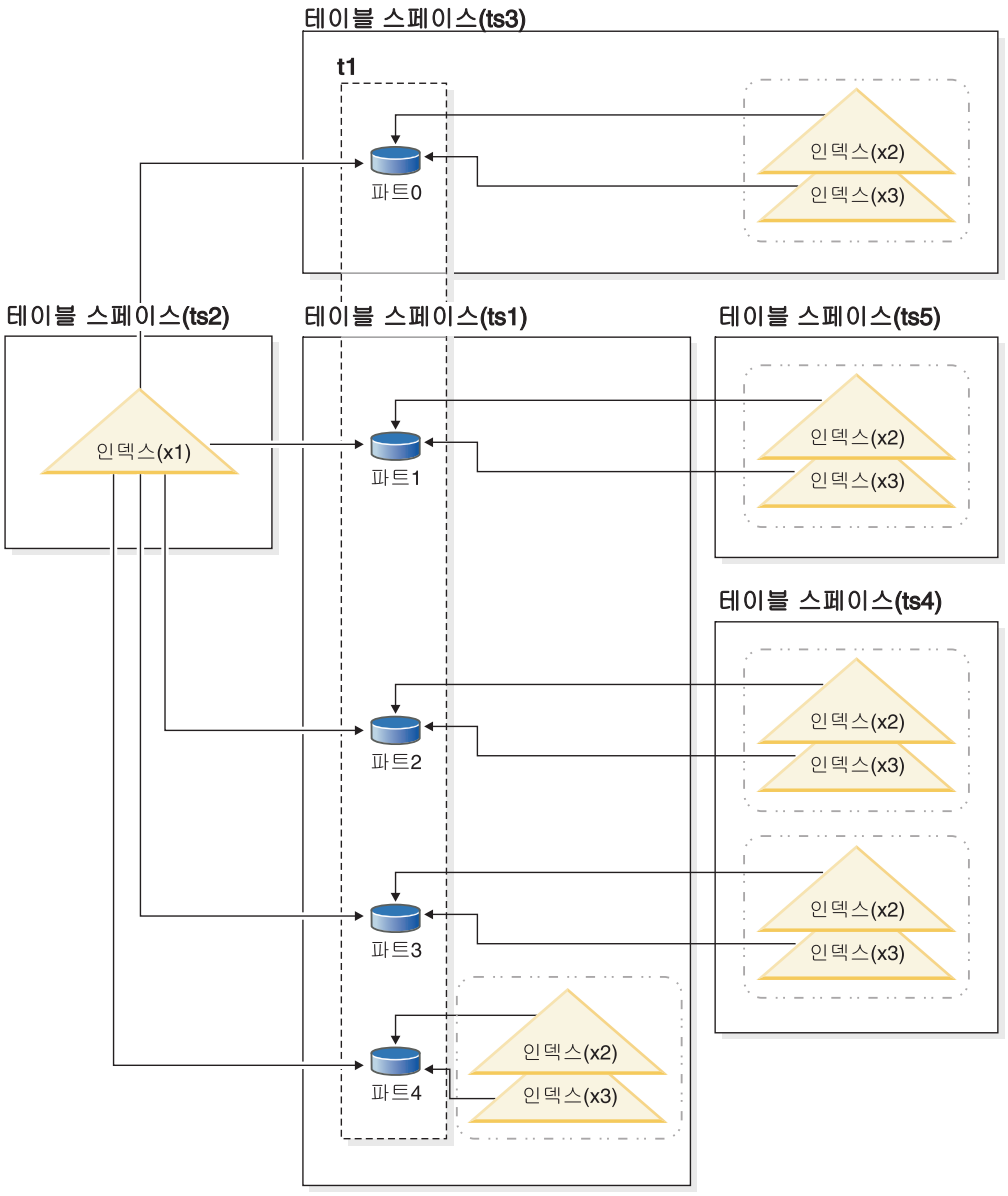


그림 14. 파티션된 테이블의 파티션된 인덱스 및 파티션되지 않은 인덱스

파티션되지 않은 인덱스 X1은 모든 데이터 파티션의 행을 참조합니다. 대조적으로, 파티션된 인덱스 X2 및 X3은 연관된 데이터 파티션의 행만 참조합니다. 테이블 스페이스 TS3은 연관된 데이터 파티션의 테이블 스페이스를 공유하는 인덱스 파티션도 표시합니다. 이는 파티션된 인덱스의 경우 디폴트입니다.

파티션되지 않은 인덱스와 파티션된 인덱스의 디폴트 위치를 겹쳐쓸 수 있습니다(겹쳐쓰는 방법은 각각 다를 수 있음). 파티션되지 않은 인덱스를 사용하여 인덱스 작성 시 테이블 스페이스를 지정할 수 있습니다. 파티션된 인덱스의 경우, 테이블을 작성할 때 저장될 테이블 스페이스 인덱스 파티션을 결정해야 합니다.

파티션되지 않은 인덱스

파티션되지 않은 인덱스의 인덱스 위치를 겹쳐쓰려면, 인덱스의 대체 테이블 스페이스 위치를 지정할 수 있게 하는 CREATE INDEX문에서 IN절을 사용하십시오. 필요하면 테이블 스페이스마다 다른 인덱스를 배치할 수 있습니다. 파티션되지 않은 인덱스를 배치할 위치를 지정하지 않고 파티션된 테이블을 작성한 후 테이블 스페이스를 지정하지 않는 CREATE INDEX문을 사용하여 인덱스를 작성할 경우, 인덱스가 첫 번째 첨부, 또는 표시된 데이터 파티션의 테이블 스페이스에서 작성됩니다. 다음 세 가지 가능한 경우 각각이 경우 1부터 순서대로 평가되어 인덱스가 작성될 위치를 판별합니다. 인덱스에 대한 테이블 스페이스 배치를 판별하기 위한 이 평가는 일치하는 경우가 발견되면 중지됩니다.

경우 1:
인덱스 테이블 스페이스가 CREATE INDEX...IN
*tbspace*문에서 지정된 경우 이 인덱스에 대해 지정된 테이블 스페이스를 사용하십시오.

경우 2:
인덱스 테이블 스페이스가 CREATE TABLE...
INDEX IN *tbspace*문에서 지정된 경우 이 인덱스에 대해 지정된 테이블 스페이스를 사용하십시오.

경우 3:
테이블 스페이스가 지정되지 않은 경우 첫 번째 첨부, 또는
표시된 데이터 파티션에서 사용되는 테이블 스페이스를 선택하십시오.

파티션된 인덱스

디폴트로, 인덱스 파티션은 참조하는 데이터 파티션과 동일한 테이블 스페이스에 배치됩니다. 이 디폴트 동작을 겹쳐쓰려면, CREATE TABLE문을 사용하여 정의하는 각 데이터 파티션에 INDEX IN절을 사용해야 합니다. 즉, 파티션된 테이블에 파티션된 인덱스를 사용하려면 테이블을 작성할 때 인덱스 파티션을 저장할 위치를 예상해야 합니다. 파티션된 인덱스 작성 시 INDEX IN절을 사용하려고 하면 오류 메시지를 수신합니다.

예 1: 파티션된 테이블 SALES(a int, b int, c int)의 경우 고유 인덱스 A_IDX를 작성하십시오.

```
create unique index a_idx on sales (a)
```

테이블 SALES는 파티션되어 있기 때문에 인덱스 a_idx도 파티션된 인덱스로 작성됩니다.

예 2: 인덱스 B_IDX를 작성하십시오.

```
create index b_idx on sales (b)
```

예 3: 파티션된 인덱스에서 인덱스 파티션의 디폴트 위치를 겹쳐쓰려면 파티션된 테이블을 작성할 때 정의하는 각 파티션에 INDEX IN절을 사용하십시오. 다음 예에서 테이블 Z의 인덱스는 테이블 스페이스 TS3에 작성됩니다.

```
create table z (a int, b int)
      partition by range (a) (starting from (1)
      ending at (100)  index in ts3)

   create index c_idx on z (a) partitioned
```

파티션된 테이블에서 파티션되지 않은 인덱스 클러스터링

클러스터링 인덱스는 일반 테이블에 대해 제공하는 이점과 동일한 이점을 파티션된 테이블에 제공합니다. 그러나 클러스터링 인덱스를 선택할 때 테이블 파티션 키 정의에서 주의를 기울여야 합니다.

클러스터링 키를 사용하여 파티션된 테이블의 클러스터링 인덱스를 작성할 수 있습니다. 데이터베이스 서버는 클러스터링 인덱스를 사용하여 각 데이터 파티션 내에서 로컬로 데이터를 클러스터하려고 시도합니다. 클러스터된 삽입 조작 동안 적절한 레코드 ID(RID)를 찾기 위해 인덱스 찾아보기가 수행됩니다. 이 RID는 레코드를 삽입할 스페이스를 찾을 때 테이블에서 시작점으로 사용됩니다. 좋은 성능의 최적의 클러스터링을 얻기 위해서는 인덱스 컬럼과 테이블 파티션 키 컬럼 사이에 상관이 있어야 합니다. 이러한 상관을 보장하는 한 가지 방법은 다음 예에 표시된 것처럼 인덱스 컬럼에 테이블 파티션 키 컬럼으로 접두부를 지정하는 것입니다.

```
partition by range (month, region)
create index...(month, region, department) cluster
```

데이터베이스 서버에서 이 상관을 강제 적용하지는 않지만, 좋은 클러스터링을 얻기 위해 인덱스의 모든 키가 파티션 ID별로 함께 그룹화될 가능성이 있습니다. 예를 들어, 테이블이 분기로 파티션되어 있고 클러스터링 인덱스가 날짜로 정의되어 있다고 가정해 보십시오. 분기와 날짜 간에 관계가 있고, 데이터 파티션의 모든 키가 인덱스 내에서 함께 그룹화되므로 좋은 성능의 최적의 데이터 클러스터링을 얻을 수 있습니다. 91 페이지의 그림 15에서는 클러스터링이 테이블 파티션 키와 상관될 경우에만 최적의 스캔 성능이 확보된다는 것을 보여줍니다.

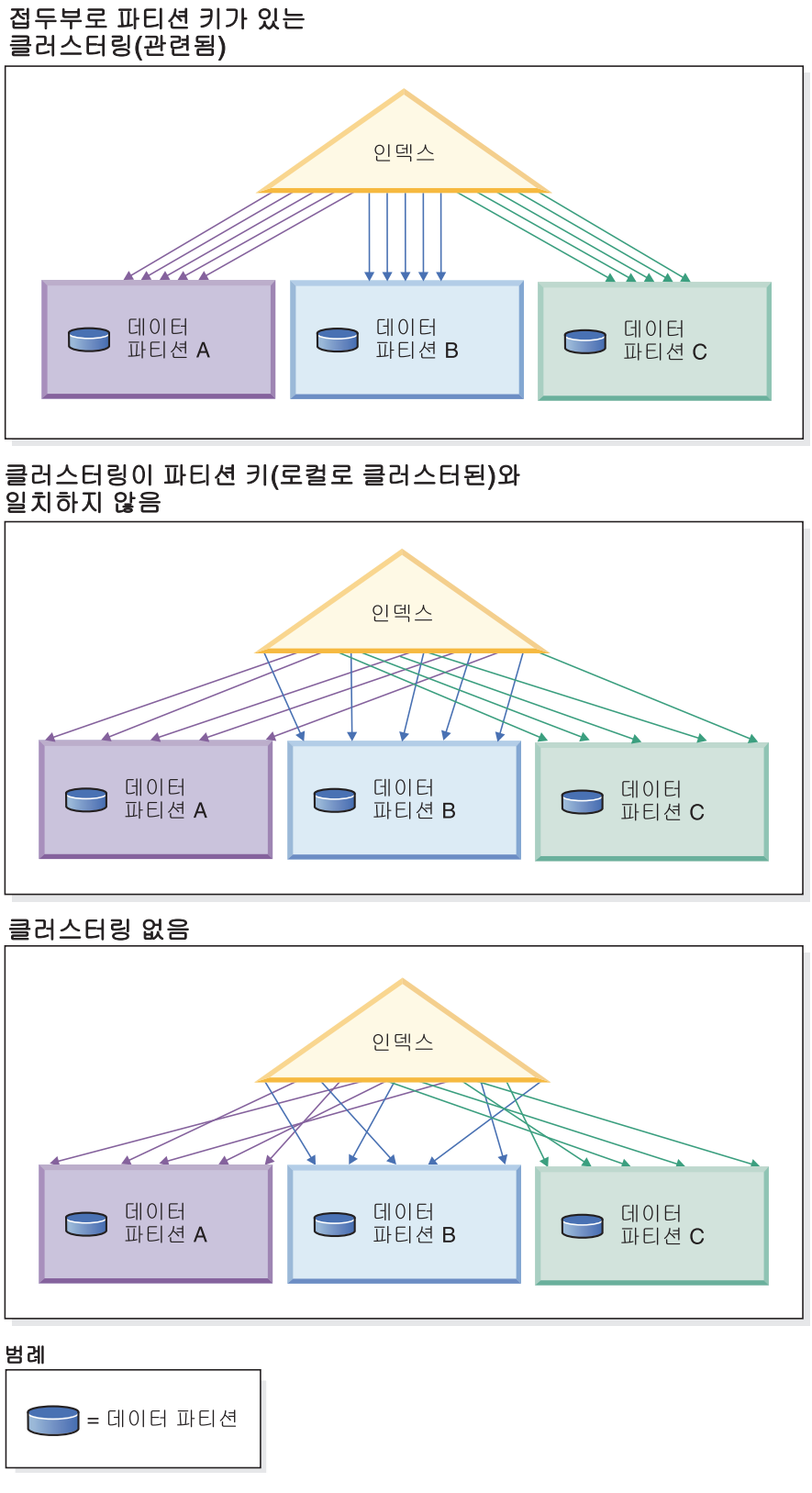


그림 15. 파티션된 테이블에서 클러스터된 인덱스의 가능한 효과

클러스터링의 이점은 다음과 같습니다.

- 각 데이터 파티션 내에서 행이 키 순서로 됩니다.
- 스캐너가 첫 번째 페이지의 첫 번째 행을 페치한 후 같은 페이지의 각 행을 페치한 후 다음 페이지로 넘어가므로 클러스터링 인덱스는 키 순서로 테이블을 횡단하는 스캔의 성능을 향상시킵니다. 즉, 지정된 시간에 테이블의 하나의 페이지만 버퍼 풀에 있어야 합니다. 반대로, 테이블이 클러스터되지 않은 경우, 다른 페이지에서 행이 페치될 수 있습니다. 버퍼 풀에서 전체 테이블을 수용할 수 없는 경우, 대부분의 페이지가 두 번 이상 페치될 수 있어 스캔 속도를 크게 저하시킵니다.

클러스터링 키가 테이블 파티션 키와 상관되지 않지만 데이터가 로컬로 클러스터되는 경우, 버퍼 풀에 각 데이터 파티션의 한 페이지를 수용할 충분한 스페이스가 있으면 클러스터된 인덱스의 전체 이점을 얻을 수 있습니다. 이는 특정 데이터 파티션에서 페치된 각 행이 같은 파티션에서 이전에 페치된 행 근처에 있기 때문입니다(91 페이지의 그림 15의 두 번째 예 참조).

페더레이티드 데이터베이스

페더레이티드 데이터베이스에 영향을 주는 서버 옵션

페더레이티드 데이터베이스 시스템은 DB2 데이터 서버(페더레이티드 데이터베이스) 및 하나 이상의 데이터 소스로 구성되어 있습니다. CREATE SERVER문을 발행할 때 페더레이티드 데이터베이스에 데이터 소스를 식별하십시오. 또한 페더레이티드 시스템 조작의 다양한 측면을 자세히 지정하고 제어하는 서버 옵션을 지정할 수 있습니다.

서버를 작성하고 서버 옵션을 지정하려면 먼저 분산 조인 설치 옵션을 설치하고 **federated** 데이터베이스 관리 프로그램 구성 매개변수를 YES로 설정해야 합니다. 나중에 서버 옵션을 변경하려면 ALTER SERVER문을 사용하십시오.

CREATE SERVER문에 지정하는 서버 옵션 값은 쿼리 푸시다운 분석, 전역 최적화 및 페더레이티드 데이터베이스 조작의 기타 측면에 영향을 줍니다. 예를 들어, 서버 옵션 값으로 성능 통계를 지정할 수 있습니다. *cpu_ratio* 옵션은 데이터 소스 및 페더레이티드 서버에서 프로세서의 상대적 속도를 지정하며 *io_ratio* 옵션은 데이터 소스 및 페더레이티드 서버에서 데이터 입출력 분산의 상대적 비율을 지정합니다.

서버 옵션 값은 시스템 카탈로그(SYSCAT.SERVEROPTIONS)로 기록되고 옵티마이저는 데이터 소스의 액세스 플랜을 개발할 때 이 정보를 사용합니다. 통계가 변경된 경우(예: 데이터 소스 프로세서가 업그레이드된 경우) ALTER SERVER문을 사용하여 카탈로그를 새 값으로 갱신하십시오.

자원 이용

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.

그림 16에서는 다양한 용도로 데이터베이스 관리 프로그램이 할당하는 여러 가지 다른 메모리 영역과 이러한 메모리 영역의 *크기*를 제어하는 데 사용할 수 있는 구성 매개변수를 보여줍니다. 파티션된 데이터베이스 환경에서 각 데이터베이스 파티션에는 자체의 데이터베이스 관리 프로그램 공유 메모리 세트가 있다는 점을 참고하십시오.

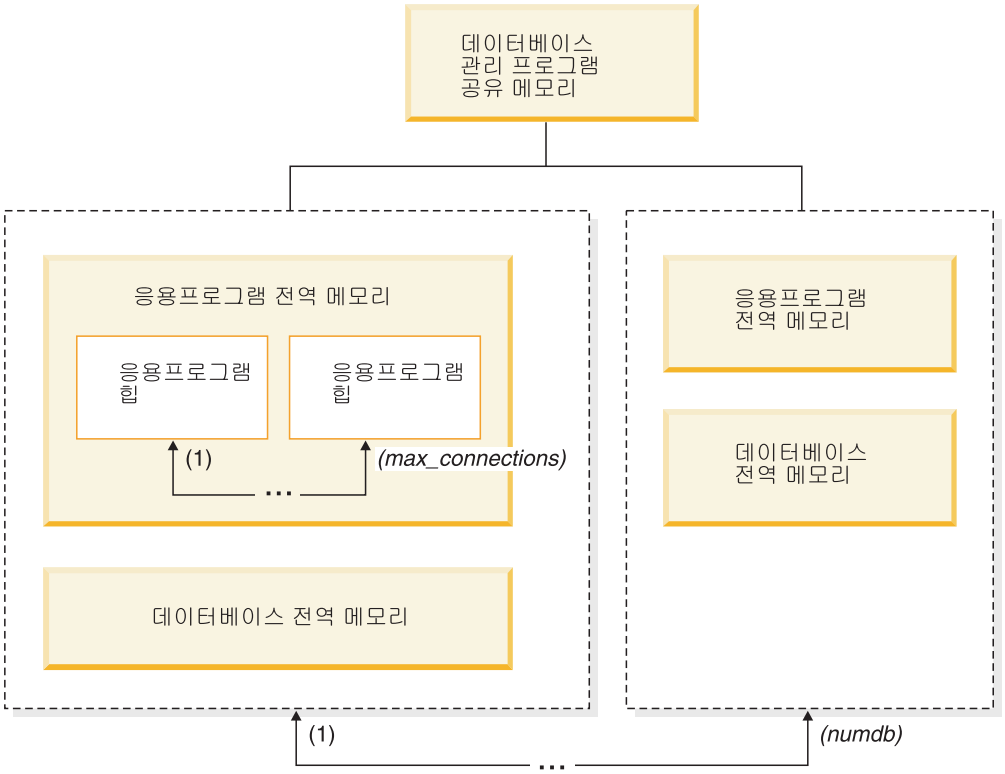


그림 16. 데이터베이스 관리 프로그램이 할당하는 메모리의 유형

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

데이터베이스 관리 프로그램 시작 시(db2start)

데이터베이스 관리 프로그램 공유 메모리(인스턴스 공유 메모리라고도 함)는 데이터베이스 관리 프로그램이 중지되기 전까지(db2stop) 계속 할당되어 있습니다. 이 영역에는 데이터베이스 관리 프로그램이 모든 데이터베이스 연결을 통해 활

동을 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.

데이터베이스 관리 프로그램 공유 메모리

데이터베이스 관리 프로그램 공유 메모리는 여러 가지 다른 메모리 영역으로 구성되어 있습니다. 구성 매개변수를 사용하면 이러한 영역의 *크기*를 제어할 수 있습니다.

96 페이지의 그림 17에서는 데이터베이스 관리 프로그램 공유 메모리가 할당되는 방식을 보여줍니다.

메모: 상자 크기는 메모리의 상대적 크기를 표시하지 않습니다.

그림 17. 데이터베이스 관리 프로그램에서 메모리를 사용하는 방식

모니터 힙

이 메모리 영역은 데이터베이스 시스템 모니터 데이터에 사용됩니다. 이 영역의 크기는 **mon_heap_sz** 데이터베이스 관리 프로그램 구성 매개변수로 판별됩니다.

감사 버퍼

이 메모리 영역은 데이터베이스 감사 활동에 사용됩니다. 이 버퍼의 *크기*는 **audit_buf_sz** 데이터베이스 관리 프로그램 구성 매개변수로 판별됩니다.

FCM(Fast Communication Manager) 버퍼 풀

파티션된 데이터베이스 시스템의 경우 FCM(Fast Communication Manager)은 충분한 메모리 스페이스가 필요합니다(특히 **fcm_num_buffers**의 값이 큰 경우). FCM 메모리 요구사항은 FCM 버퍼 풀에서 할당됩니다.

데이터베이스 전역 메모리

데이터베이스 전역 메모리는 버퍼 풀 *크기*와 다음의 데이터베이스 구성 매개변수에 따라 다릅니다.

- **catalogcache_sz**
- **database_memory**
- **dbheap**
- **locklist**
- **pckcachesz**
- **sheapthres_shr**
- **util_heap_sz**

응용프로그램 전역 메모리

응용프로그램 전역 메모리는 **appl_memory** 구성 매개변수로 제어할 수 있습니다. 다음의 데이터베이스 구성 매개변수를 사용하여 한 응용프로그램이 사용할 수 있는 메모리의 양을 제한할 수 있습니다.

- **applheapsz**
- **stat_heap_sz**
- **stmtheap**

에이전트 전용 메모리

각 에이전트에는 자체의 전용 메모리 영역이 필요합니다. 데이터 서버는 구성된 메모리 자원에 따라 필요한 모든 에이전트를 작성합니다. **max_coordagents** 데이터베이스 관리 프로그램 구성 매개변수를 사용하여 최대 코디네이터 에이전트 수를 제어할 수 있습니다. 각 에이전트의 전용 메모리 영역에 대한 최대 *크기*는 다음의 구성 매개변수 값으로 판별됩니다.

- **agent_stack_sz**
- **sheapthres** 및 **sortheap**

에이전트/응용프로그램 공유 메모리

로컬 클라이언트의 에이전트/응용프로그램 공유 메모리 세그먼트에 대한 전체 수는 다음의 두 값 중 작은 값으로 제한됩니다.

- 모든 활성 데이터베이스에 대한 **maxappls** 데이터베이스 구성 매개변수의 전체 값
- **max_coordagents** 데이터베이스 구성 매개변수의 값

주: 엔진 집중이 사용 가능한 구성(**max_connections** > **max_coordagents**)에서 응용프로그램 메모리 사용은 **max_connections**로 제한됩니다.

에이전트/응용프로그램 공유 메모리도 다음의 데이터베이스 구성 매개변수에 따라 다릅니다.

- **aslheapsz**
- **rqrioblk**

FCM 버퍼 풀 및 메모리 요구사항

파티션된 데이터베이스 시스템에서 FCM(Fast Communication Manager) 버퍼 공유 메모리는 데이터베이스 관리 프로그램 공유 메모리에서 할당됩니다.

이 요구사항은 그림 18에 표시되어 있습니다.

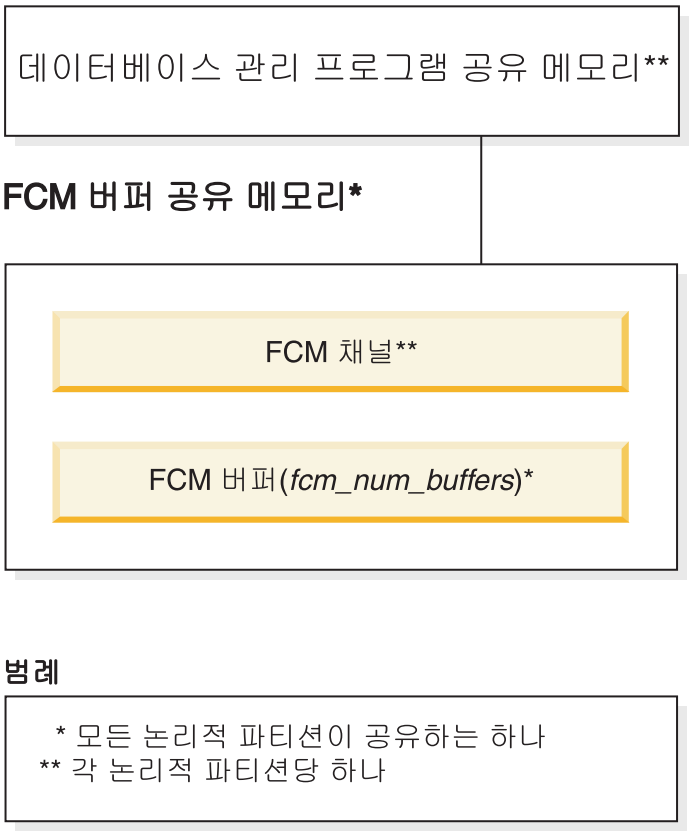


그림 18. 다중 논리 파티션 사용 시 FCM 버퍼 풀

각 데이터베이스 파티션의 FCM 버퍼 수는 **fcm_num_buffers** 데이터베이스 관리 프로그램 구성 매개변수로 제어됩니다. 디폴트로 이 매개변수는 자동으로 설정됩니다. 이 매개변수를 수동으로 조정하려면 **buff_free** 및 **buff_free_bottom** 시스템 모니터 요소의 데이터를 사용하십시오.

각 데이터베이스 파티션의 FCM 채널 수는 **fcm_num_channels** 데이터베이스 관리 프로그램 구성 매개변수로 제어됩니다. 디폴트로 이 매개변수는 자동으로 설정됩니다. 이 매개변수를 수동으로 조정하려면 **ch_free** 및 **ch_free_bottom** 시스템 모니터 요소의 데이터를 사용하십시오.

DB2 데이터베이스 관리 프로그램은 필요에 따라 추가 FCM 버퍼 및 채널을 할당하여 자동으로 FCM 메모리 자원을 관리할 수 있습니다. 그러면 성능이 향상되고 『FCM 자원 부족』 런타임 오류를 예방할 수 있습니다. Linux 운영 체제에서, 데이터베이스 관리 프로그램은 최대 디폴트 양 2GB까지 FCM 버퍼 및 채널의 시스템 메모리 양을 미리 할당할 수 있습니다. 메모리 스페이스는 추가 FCM 버퍼 또는 채널이 필요한 경우에만 영향을 받습니다. 이 동작이 가능하도록 하려면 **DB2_FCM_SETTINGS** 레지스트리 변수의 FCM_MAXIMIZE_SET_SIZE 옵션을 YES(또는 TRUE)로 설정하십시오. YES는 디폴트값입니다.

메모리 사용에 영향을 주는 매개변수 조정 지침

메모리를 수동으로 조정할 때(즉, 자체 조정 메모리 관리자을 사용하지 않는 경우) 벤치마크 테스트는 메모리 매개변수에 적합한 값을 설정하는 방법에 대한 최상의 정보를 제공합니다.

벤치마크 테스팅에서 대표 및 worst-case SQL문이 서버에 대해 실행되며 성능의 수익 체감 시점이 발견될 때까지 메모리 매개변수의 값이 변경됩니다. 이 시점은 추가 메모리 할당으로 응용프로그램에 더 이상 성능 이점이 제공되지 않는 시점입니다.

여러 매개변수의 메모리 할당 상한은 기존 하드웨어 및 운영 체제 범위에서 벗어날 수 있습니다. 이러한 한계는 이후 확장을 위해 허용됩니다. 이러한 값에 맞출 수 없는 경우 메모리 매개변수를 최고값으로 설정하지 않는 것이 좋습니다. 이 사항은 사용 가능한 메모리가 많은 시스템에도 적용됩니다. 데이터베이스 관리 프로그램이 시스템에서 사용 가능한 모든 메모리를 신속하게 소모하지 못하도록 예방하기 위한 것입니다. 또한 대량의 메모리를 관리할 경우 추가 오버헤드가 발생합니다.

대부분의 구성 매개변수의 경우 메모리는 필요할 때 커미트되고 매개변수 설정은 특정 메모리 힙의 최대 *크기*를 판별합니다. 그러나 버퍼 풀 및 다음의 구성 매개변수에 대해 지정된 모든 메모리가 할당됩니다.

- **aslheapsz**
- **fcm_num_buffers**
- **fcm_num_channels**

- **locklist**

메모리를 스왑 영역으로 페이지 아웃해야 하는 경우가 아니고 프로세스가 메모리를 할당할 때마다 일부 운영 체제가 스왑 영역을 할당합니다. 이러한 시스템의 경우 일반적으로 시스템 전체 메모리의 두 배에 이르는 페이지 공간을 제공하는 것이 좋습니다.

유효한 매개변수 범위는 각 매개변수에 대한 자세한 정보를 참조하십시오.

자체 조정 메모리 개요

자체 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 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

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

자체 조정 메모리 사용

자체 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 조정 메모리를 사용할 수 있게 하십시오.
2. 메모리 구성 매개변수에 의해 제어되는 메모리 영역 자체 조정을 사용할 수 있게 하려면 UPDATE DATABASE CONFIGURATION 명령 또는 db2CfgSet API를 사용하여 관련 구성 매개변수를 AUTOMATIC으로 설정하십시오.
3. 버퍼 풀 자체 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 조정된 메모리는 다양한 메모리 소비자 간에 분배되므로, 주어진 시간에 자체 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 조정을 사용 불가능하게 하면 **maxlocks** 매개변수의 자체 조정이 자동으로 사용 불가능해지고, **locklist** 매개변수의 자체 조정을 사용 가능하게 하면 **maxlocks** 매개변수의 자체 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 조정을 사용 불가능하게 하면 **sheapthres_shr** 매개변수의 자체 조정이 자동으로 사용 불가능해지고, **sheapthres_shr** 매개변수의 자체 조정을 사용 가능하게 하면 **sortheap** 매개변수의 자체 조정이 자동으로 사용 가능해집니다.
5. 자체 조정 메모리는 고가용성 재해 복구(HADR) 기본 서버에서만 실행됩니다. HADR 시스템에서 자체 조정 메모리가 활성화된 경우, 보조 서버에서는 실행되지 않으며,

구성이 제대로 설정된 경우에만 기본 서버에서 실행됩니다. HADR 데이터베이스 역할이 전환된 경우, 자체 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 조정 메모리 관리자(STMM) 엔진 디스패치 가능 단위(EDU)가 시작되지 않을 수도 있습니다.

자체 조정 메모리 사용 안함

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

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

1. UPDATE DATABASE CONFIGURATION 명령 또는 db2CfgSet API를 사용해 **self_tuning_mem** 데이터베이스 구성 매개변수를 해제(OFF)로 설정하여 데이터베이스에 대해 자체 조정 메모리를 사용하지 않도록 하십시오.
2. 메모리 구성 매개변수에 의해 제어되는 메모리 영역의 자체 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 조정을 사용 불가능하게 하려면 CREATE BUFFERPOOL문 또는 ALTER BUFFERPOOL문을 사용하여 버퍼 풀 크기를 특정 값으로 설정하십시오.

주:

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

자체 조정을 사용 가능한 메모리 소비자 판별

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

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

```
Description                     Parameter         Current Value       Delayed Value
--------------------------------------------------------------------------------------
Self tuning memory                   (SELF_TUNING_MEM) = ON (Active)        ON
Size of database shared memory (4KB) (DATABASE_MEMORY) = AUTOMATIC(37200)   AUTOMATIC(37200)
Max storage for lock list (4KB)            (LOCKLIST) = AUTOMATIC(7456)    AUTOMATIC(7456)
Percent. of lock lists per application     (MAXLOCKS) = AUTOMATIC(98)      AUTOMATIC(98)
```

```
Package cache size (4KB)                  (PCKCACHESZ) = AUTOMATIC(5600)    AUTOMATIC(5600)
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = AUTOMATIC(5000)    AUTOMATIC(5000)
Sort list heap (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 조정이 사용 가능하고 활성임을 표시하는 반면, SQLF_ON_INACTIVE는 자체 조정이 사용 가능하지만 현재 비활성임을 표시합니다.

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

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

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

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

- 자체 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 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 조정 메모리

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

파티션된 데이터베이스에 대해 자체 조정 메모리가 사용될 경우, 단일 데이터베이스 파티션이 조정 파티션으로 지정되고 모든 메모리 조정 결정은 해당 데이터베이스 파티션

의 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 조정 메모리를 사용할지 여부를 결정할 때 다음 지침을 사용하십시오.

파티션된 데이터베이스에 대한 자체 조정 메모리가 권장되는 경우

모든 데이터베이스 파티션이 유사한 메모리 요구사항을 지니고 유사한 하드웨어에서 실행되고 있는 경우, 자체 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.
- 완벽하거나 완벽에 가깝게 데이터가 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 조정 메모리가 시스템을 적절히 구성합니다.

파티션된 데이터베이스에 대한 자체 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 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 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 조정 메모리를 사용하지 않아야 합니다. 코디네이터 파티션과 카탈로그 파티션에서 자체 조정 메모리를 사용하지 않으려면 이러한 파티션에서 **self_tuning_mem** 데이터베이스 구성 매개변수를 해제(OFF)로 설정하십시오.

파티션된 데이터베이스에 대한 자체 조정 메모리가 권장되지 않는 경우

각 데이터베이스 파티션의 메모리 요구사항이 다른 경우, 또는 크게 다른 하드웨어에서 다른 데이터베이스 파티션이 실행되고 있는 경우 자체 조정 메모리 기능을 사용하지 않

는 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.

```
Locks held currently                        = 0
Lock waits                                  = 0
Time database waited on locks (ms)          = 0
Lock list memory in use (Bytes)             = 4968
Lock escalations                            = 0
Exclusive lock escalations                  = 0

Total Shared Sort heap allocated            = 0
Shared Sort heap high water mark            = 0
Post threshold sorts (shared memory)        = 0
Sort overflows                              = 0

Package cache lookups                       = 13
Package cache inserts                       = 1
Package cache overflows                     = 0
Package cache high water mark (Bytes)       = 655360

Number of hash joins                        = 0
Number of hash loops                        = 0
Number of hash join overflows               = 0
Number of small hash join overflows         = 0
Post threshold hash joins (shared memory)   = 0

Number of OLAP functions                    = 0
Number of OLAP function overflows           = 0
Active OLAP functions                       = 0
```

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

```
Buffer pool data logical reads              = 0
Buffer pool data physical reads             = 0
Buffer pool index logical reads             = 0
Buffer pool index physical reads            = 0
Total buffer pool read time (milliseconds) = 0
Total buffer pool write time (milliseconds)= 0
```

파티션된 데이터베이스 환경에서 자체 조정 메모리 사용

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

특정 데이터베이스 파티션에 대해 자체 조정 메모리 사용 안함

- 데이터베이스 파티션의 서브세트에 대해 자체 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 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 조정 메모리 구성 매개변수 값이 일관성이 있는 것이 좋습니다.
- 특정 데이터베이스 파티션의 특정 버퍼 풀에 대해 자체 조정 메모리를 사용하지 않으려면 자체 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 버

퍼 풀이 자체 조정 조작에 참여하지 않습니다. 버퍼 풀이 자체 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 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 조정을 사용하십시오.

비균일 환경에서 자체 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 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 조정 메모리가 여전히 사용될 수 있습니다. 비대칭 환경에서 자체 조정 메모리를 이용하려면 유사한 데이터 및 메모리 요구사항을 지닌 데이터베이스 파티션 세트를 식별하고 자체 조정에 사용하십시오. 나머지 파티션의 메모리는 수동으로 구성해야 합니다.

버퍼 풀 관리

버퍼 풀은 데이터베이스 페이지에 작업 메모리 및 캐시를 제공합니다.

버퍼 풀은 디스크 대신 메모리에서 데이터에 액세스할 수 있도록 하여 데이터베이스 시스템 성능을 향상시킵니다. 대부분의 페이지 데이터 조작은 버퍼 풀에서 발생하므로 버퍼 풀 구성은 가장 중요한 단일 조정 영역입니다.

응용프로그램이 테이블 행에 액세스할 때 데이터베이스 관리 프로그램은 버퍼 풀에서 해당 행이 있는 페이지를 찾습니다. 여기에서 페이지를 찾을 수 없는 경우 데이터베이스 관리 프로그램은 디스크에서 페이지를 읽고 버퍼 풀에 배치합니다. 그런 다음 데이터를 사용하여 쿼리를 처리할 수 있습니다.

데이터베이스가 활성화되면 메모리가 버퍼 풀에 할당됩니다. 처음 응용프로그램을 연결하면 내재된 데이터베이스 활성화가 발생합니다. 데이터베이스 관리 프로그램을 실행하는 동안 버퍼 풀을 작성하거나 *크기*를 조정하거나 삭제할 수 있습니다. ALTER BUFFERPOOL문을 사용하여 버퍼 풀의 *크기*를 늘릴 수 있습니다. 충분한 메모리가 사용 가능한 경우 디폴트로 명령문이 실행되면 즉시 버퍼 풀의 *크기*가 조정됩니다. 명령

문 실행 시 충분한 메모리가 사용 불가능한 경우 데이터베이스를 재활성화하면 메모리가 할당됩니다. 버퍼 풀의 *크기*를 줄이면 트랜잭션이 커미트될 때 메모리가 할당 해제됩니다. 데이터베이스가 비활성화되면 버퍼 풀 메모리가 사용 가능해집니다.

모든 경우에 적합한 버퍼 풀이 사용 가능한지 확인하기 위해 DB2는 4KB, 8KB, 16KB 및 32KB 페이지 *크기*마다 하나씩 소형 시스템 버퍼 풀을 작성합니다. 각 버퍼 풀의 *크기*는 16페이지입니다. 이러한 버퍼 풀은 숨겨져 있으며 시스템 카탈로그 또는 버퍼 풀 시스템 파일에 없습니다. 직접 사용하거나 변경할 수 없지만 DB2는 다음과 같은 경우에 이러한 버퍼 풀을 사용합니다.

- DEFERRED 키워드를 사용하여 작성되었으므로 지정된 버퍼 풀이 시작되지 않았거나 작성하는 데 사용할 수 있는 메모리 부족으로 인해 필요한 페이지 *크기*의 버퍼 풀이 비활성화된 경우

  관리 통지 로그로 메시지가 기록됩니다. 필요하면 테이블 스페이스를 시스템 버퍼 풀로 재맵핑합니다. 성능은 현저하게 저하될 수 있습니다.
- 데이터베이스 연결 시도 중 버퍼 풀을 시작할 수 없는 경우

  이 문제에 메모리 부족 조건과 같은 심각한 원인이 있을 수 있습니다. DB2는 시스템 버퍼 풀로 인해 계속해서 완전한 기능을 수행하지만 성능이 현저하게 저하됩니다. 이 문제를 즉시 해결해야 합니다. 이 문제가 발생하면 경고를 받으며 관리 통지 로그에 메시지가 기록됩니다.

버퍼 풀을 작성할 때 다른 페이지 *크기*를 명시적으로 지정하지 않는 한 데이터베이스 작성 시 페이지 *크기*를 지정합니다. 테이블 스페이스 페이지 *크기*가 버퍼 풀 페이지 크기와 동일한 경우에만 버퍼 풀로 페이지를 읽을 수 있으므로 테이블 스페이스의 페이지 *크기*는 버퍼 풀에 지정하는 페이지 *크기*를 판별해야 합니다. 버퍼 풀의 페이지 *크기*를 작성한 후 변경할 수 없습니다.

db2mtrk 명령을 발행하여 호출할 수 있는 메모리 추적 프로그램을 통해 버퍼 풀에 할당된 데이터베이스 메모리의 양을 볼 수 있습니다. 또한 GET SNAPSHOT 명령을 사용하고 버퍼 풀의 현재 *크기*(**bp_cur_buffsz** 모니터 요소의 값)를 조사할 수 있습니다.

활동에 대한 버퍼 풀 우선순위는 DB2 워크로드 관리 프로그램이 제공하는 대규모 워크로드 관리 기능 세트의 일부로 제어될 수 있습니다. 자세한 정보는 『DB2 워크로드 관리 프로그램 개념 소개』 및 『서비스 클래스의 버퍼 풀 우선순위』를 참조하십시오.

데이터 페이지의 버퍼 풀 관리

버퍼 풀 페이지는 사용 중 또는 사용 중이 아니고 더티 또는 정리입니다.

- *사용 중인 페이지*는 현재 읽거나 갱신 중인 페이지입니다. 페이지를 갱신 중인 경우 갱신자만 액세스할 수 있습니다. 그러나 페이지를 갱신하지 않는 경우 읽는 사용자가 동시에 여러 명일 수 있습니다.
- *더티 페이지*에는 변경되었지만 아직 디스크에 기록되지 않은 데이터가 포함됩니다.

데이터베이스가 종료되거나 페이지가 차지하는 스페이스가 다른 페이지에 필요하거나 예를 들어, 오브젝트 삭제의 일부분으로 버퍼 풀에서 페이지가 명시적으로 제거되기 전까지 페이지는 버퍼 풀에 남아 있습니다. 다음 기준은 다른 페이지에 스페이스가 필요하면 제거되는 페이지를 판별합니다.

- 얼마나 최근에 페이지가 참조되었습니까?
- 페이지가 다시 참조될 가능성이 있습니까?
- 페이지에 어느 데이터 유형이 들어 있습니까?
- 메모리에서 페이지가 변경되었지만 디스크에 기록되지 않았습니까?

변경된 페이지는 항상 겹쳐쓰기 전에 디스크에 기록됩니다. 디스크에 기록되는 변경된 페이지는 스페이스가 필요하지 않는 한 버퍼 풀에서 자동으로 제거되지 않습니다.

페이지 클리너 에이전트

올바로 조정된 시스템에서 보통 변경된 페이지 또는 더티 페이지를 디스크에 기록하는 페이지 클리너 에이전트입니다. 페이지 클리너 에이전트는 입출력을 백그라운드 프로세스로 수행하고 에이전트가 실제 트랜잭션 작업을 수행할 수 있으므로 응용프로그램을 빠르게 실행할 수 있도록 지원합니다. 페이지 클리너 에이전트는 다른 에이전트의 작업과 협력하지 않으며 필요한 경우에만 작동하므로 *비동기 페이지 클리너* 또는 *비동기 버퍼 기록기*라고도 합니다.

갱신에 집중된 워크로드의 성능을 향상시키기 위해 *혁신적인 페이지 정리*를 사용할 수 있습니다. 이 경우 지정된 시점에 기록되는 더티 페이지를 선택할 때 페이지 클리너를 혁신적으로 사용할 수 있습니다. 이러한 내용은 특히 스냅샷이 비동기 데이터 페이지 또는 인덱스 페이지 쓰기와 관련하여 동기 데이터 페이지 또는 인덱스 페이지 쓰기 수가 있음을 표시하는 경우에 적용됩니다.

111 페이지의 그림 19에서는 버퍼 풀 관리 작업을 페이지 클리너 에이전트와 데이터베이스 에이전트 사이에 공유할 수 있는 방식을 보여줍니다.

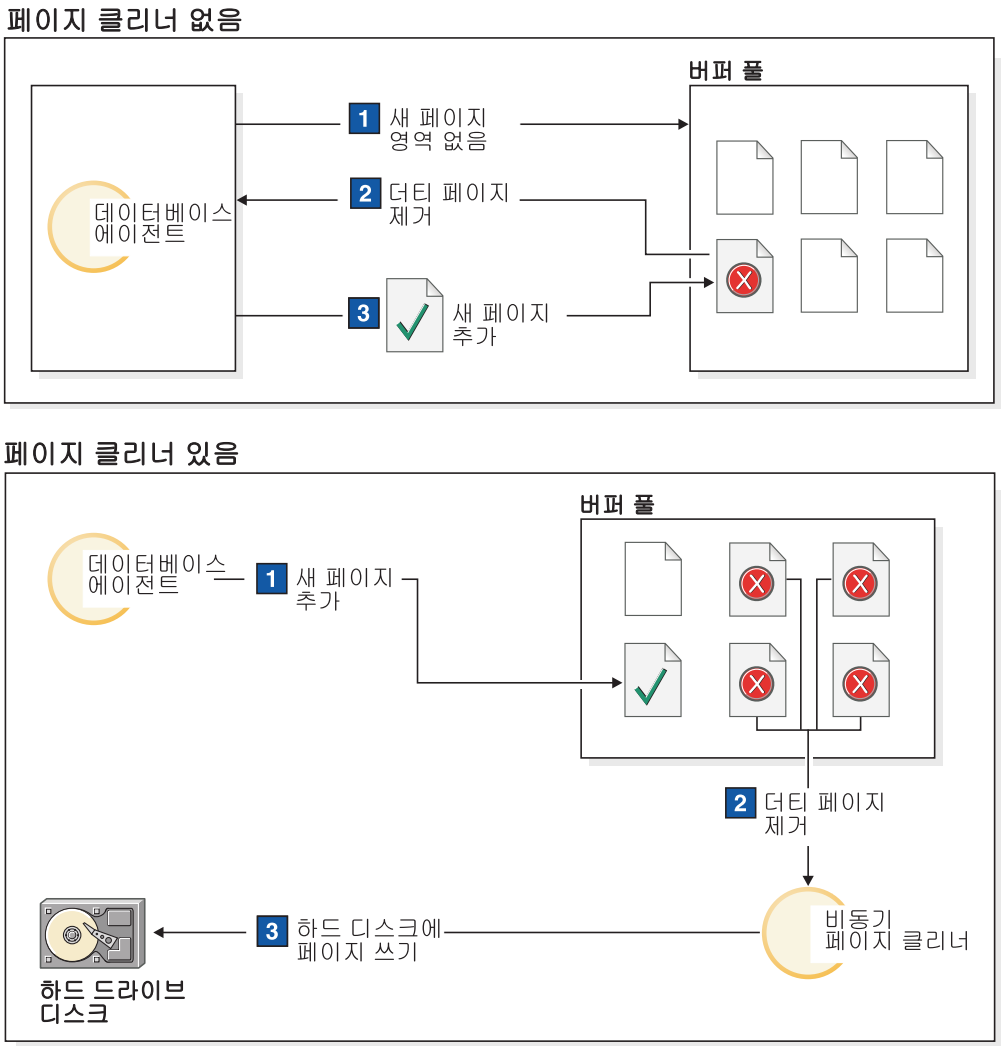


그림 19. 비동기 페이지 정리. 더티 페이지가 디스크에 기록됩니다.

페이지 정리 및 빠른 복구

데이터베이스 관리 프로그램은 데이터베이스 로그 파일에서 트랜잭션을 재실행하는 것보다 디스크에서 버퍼 풀을 추가로 재빌드할 수 있으므로 디스크에 추가 페이지가 기록된 경우 시스템 파손 후 데이터베이스 복구가 빠릅니다.

복구 중 읽어야 하는 로그의 크기는 로그에 있는 다음 레코드들의 위치마다 다릅니다.

- 최근에 기록한 로그 레코드
- 버퍼 풀에서 데이터의 가장 오래된 변경사항에 대해 설명하는 로그 레코드

복구 중 재실행해야 하는 로그의 크기가 다음을 초과하지 않는 방식으로 페이지 클리너는 디스크에 더티 페이지를 기록합니다.

```
logfilsiz * softmax / 100 (4-KB페이지 단위)
```

각 항목에 대한 설명은 다음과 같습니다.

- **logfilsiz**는 로그 파일의 *크기*를 나타냅니다.
- **softmax**는 데이터베이스 파손 후 복구할 로그 파일의 퍼센트를 나타냅니다. 예를 들어, **softmax**의 값이 250인 경우 2.5 로그 파일에는 파손이 발생하면 복구해야 하는 변경사항이 포함됩니다.

복구 중 로그 읽기 시간을 최소한으로 줄이려면 데이터베이스 시스템 모니터를 사용하여 페이지 정리를 수행하는 횟수를 추적하십시오. 데이터베이스의 혁신적 페이지 정리를 사용 가능하게 하지 않은 경우 **pool_lsn_gap_clns**(트리거된 버퍼 풀 로그 스페이스 클리너) 모니터 요소가 이 정보를 제공합니다. 혁신적 페이지 정리를 사용 가능하게 설정한 경우 이 조건이 발생하지 않으며 **pool_lsn_gap_clns**의 값은 0입니다.

log_held_by_dirty_pages 모니터 요소를 사용하여 페이지 클리너가 사용자가 설정한 복구 기준에 맞게 충분한 페이지를 정리하지 않는지 판별할 수 있습니다. **log_held_by_dirty_pages**가 **logfilsiz** * **softmax**보다 계속해서 상당히 큰 경우 추가 페이지 클리너가 필요하거나 **softmax**를 고정해야 합니다.

다중 데이터베이스 버퍼 풀의 관리

각 데이터베이스에는 최소한 하나의 버퍼 풀이 필요하지만 둘 이상의 페이지 *크기*로 구성된 테이블 공간이 있는 단일 데이터베이스마다 각각 다른 *크기* 또는 다른 페이지 *크기*의 여러 버퍼 풀을 작성할 수 있습니다.

ALTER BUFFERPOOL문을 사용하여 버퍼 풀의 *크기*를 조정할 수 있습니다.

새 데이터베이스에는 데이터베이스 작성 시 지정된 페이지 *크기*에 기초한 디폴트 페이지 *크기*를 사용하는 디폴트 버퍼 풀 IBMDEFAULTBP가 있습니다. 디폴트 페이지 *크기*는 **pagesize**라고 하는 정보용 데이터베이스 구성 매개변수로 저장됩니다. 디폴트 페이지 *크기*로 테이블 *스페이스*를 작성하며 특정 버퍼 풀에 지정하지 않는 경우 테이블 *스페이스*는 디폴트 버퍼 풀에 지정됩니다. 디폴트 버퍼 풀의 *크기*를 조정하고 해당 속성을 변경할 수 있지만 삭제(drop)할 수 없습니다.

버퍼 풀의 페이지 크기

데이터베이스를 작성하거나 업그레이드한 후 추가 버퍼 풀을 작성할 수 있습니다. 디폴트로 8-KB 페이지 *크기*의 데이터베이스를 작성하는 경우 디폴트 페이지 *크기*(이 경우 8KB)의 디폴트 버퍼 풀이 작성됩니다. 또는 8-KB 페이지 *크기*의 버퍼 풀과 동일한 페이지 *크기*를 사용하는 하나 이상의 테이블 *스페이스*를 작성할 수 있습니다. 이 방법은 데이터베이스를 작성할 때 4-KB 디폴트 페이지 *크기*를 변경할 필요가 없습니다. 다른 페이지 *크기*를 사용하는 버퍼 풀에 테이블 *스페이스*를 지정할 수 없습니다.

주: 4KB보다 큰 페이지 *크기*(예: 8KB, 16KB 또는 16KB)로 테이블 *스페이스*를 작성하는 경우 동일한 페이지 *크기*를 사용하는 버퍼 풀에 지정해야 합니다. 이 버퍼 풀이 현재 활성화되지 않은 경우 DB2는 동일한 페이지 *크기*를 사용하는 다른 활성 버퍼 풀(있는 경우) 또는 첫 번째 클라이언트가 데이터베이스에 연결할 때 DB2가 작성하는 디

폴트 시스템 버퍼 풀 중 하나에 임시로 테이블 스페이스를 지정하려고 합니다. 데이터베이스가 다시 활성화되었으며 원래 지정된 버퍼 풀이 활성화된 경우 DB2는 이 버퍼 풀에 테이블 스페이스를 지정합니다.

CREATE BUFFERPOOL문을 사용하여 버퍼 풀을 작성할 때 *크기*를 지정하지 않은 경우 버퍼 풀 *크기*는 AUTOMATIC으로 설정되며 DB2에서 관리합니다. 나중에 버퍼 풀 *크기*를 변경하려면 ALTER BUFFERPOOL문을 사용하십시오.

파티션된 데이터베이스 환경에서 데이터베이스의 각 버퍼 풀은 CREATE BUFFERPOOL 문에 달리 지정되지 않거나 특정 데이터베이스 파티션의 버퍼 풀 *크기*를 ALTER BUFFERPOOL문에서 변경하지 않는 한 모든 데이터베이스 파티션에서 동일한 디폴트 정의를 사용합니다.

대형 버퍼 풀의 장점

대형 버퍼 풀은 다음과 같은 장점을 제공합니다.

- 자주 요청되는 데이터 페이지를 버퍼 풀에서 보존할 수 있으므로 신속하게 액세스할 수 있습니다. 입출력 조작 수가 작으면 입출력 경쟁이 줄어들 수 있으므로 응답 시간이 개선되고 입출력 조작에 필요한 프로세서 자원이 줄어듭니다.
- 동일한 응답 시간을 사용하여 트랜잭션 비율을 높일 수 있는 기회를 제공합니다.
- 카탈로그 테이블과 자주 참조되는 사용자 테이블 및 인덱스를 저장하는 것과 같이 자주 사용되는 디스크 스토리지 디바이스에 대한 입출력 경쟁을 예방합니다. 쿼리에 필요한 정렬도 임시 테이블 스페이스가 있는 디스크 스토리지 디바이스에 대한 입출력 경쟁이 줄어들 경우 유리할 수 있습니다.

다중 버퍼 풀의 장점

다음의 조건이 시스템에 적용되는 경우 단일 버퍼 풀만 사용하십시오.

- 총 버퍼 풀 스페이스가 10 000 4-KB페이지 미만인 경우
- 전문 조정을 수행할 수 있는 응용프로그램 지식이 있는 사람이 없는 경우
- 테스트 시스템에서 작업 중인 경우

기타 모든 경우와 다음과 같은 이유로 인해 둘 이상의 버퍼 풀을 사용하는 것이 좋습니다.

- 임시 테이블 스페이스를 별도의 버퍼 풀에 지정하면 임시 스토리지가 필요한 쿼리(특히 정렬에 집중된 쿼리)의 성능이 향상될 수 있습니다.
- 간단한 여러 갱신 트랜잭션 응용프로그램이 데이터에 반복적으로 신속하게 액세스해야 하는 경우 데이터가 들어 있는 테이블 스페이스를 별도의 버퍼 풀로 지정하십시오. 이 버퍼 풀의 *크기*를 적절하게 조정한 경우 페이지를 쉽게 찾을 수 있으므로 응답 시간과 트랜잭션 비용이 줄어듭니다.

- 데이터를 별도의 버퍼 풀로 분리시켜서 선호하는 특정 응용프로그램, 데이터 및 인덱스를 구분할 수 있습니다. 예를 들어, 자주 갱신하는 테이블 및 인덱스를 자주 쿼리하지만 자주 갱신하지 않는 테이블 및 인덱스와 구분된 버퍼 풀에 저장할 수 있습니다.
- 거의 사용하지 않는 응용프로그램(특히 매우 큰 테이블에 무작위로 액세스해야 하는 응용프로그램)이 액세스하는 데이터에 작은 버퍼 풀을 사용할 수 있습니다. 이러한 경우 데이터를 단일 쿼리보다 오래 버퍼 풀에 보존할 필요가 없습니다. 이러한 데이터 유형을 위해 작은 버퍼 풀을 보존하고 다른 버퍼 풀을 위해 추가 메모리를 사용 가능하게 하는 것이 좋습니다.

데이터를 다른 버퍼 풀로 분리한 후 통계 및 어카운트 추적에서 비교적 저렴하고 좋은 성능 진단 데이터가 작성될 수 있습니다.

자체 조정 메모리 관리자(STMM)는 버퍼 풀이 많은 시스템을 조정할 때 이상적입니다.

시작 시 버퍼 풀 메모리 할당

버퍼 풀을 작성하거나 변경할 때 모든 버퍼 풀에 필요한 전체 메모리를 데이터베이스 관리 프로그램에 제공하여 데이터베이스 시작 시 모든 버퍼 풀을 할당할 수 있습니다. 데이터베이스 관리 프로그램이 온라인 상태일 때 버퍼 풀을 작성하거나 변경하는 경우 데이터베이스 전역 메모리에서 추가 메모리가 사용 가능해야 합니다. 새 버퍼 풀을 작성하거나 기존 버퍼 풀의 *크기*를 늘릴 때 DEFERRED 키워드를 지정하고 필요한 메모리가 사용 불가능한 경우 데이터베이스 관리 프로그램은 다음에 데이터베이스가 활성화되면 변경을 실행합니다.

데이터베이스 시작 시 메모리가 사용 불가능한 경우 데이터베이스 관리 프로그램은 최소 *크기*가 16페이지인 시스템 버퍼 풀(각 페이지 *크기*마다 하나씩)만 사용하고 경고가 리턴됩니다. 데이터베이스는 해당 구성이 변경되고 데이터베이스를 완전히 재시작할 수 있을 때까지 계속 이 상태에 있습니다. 성능은 준최적 상태일 수 있지만 데이터베이스에 연결하거나 버퍼 풀 *크기*를 재구성하거나 다른 중요한 태스크를 수행할 수 있습니다. 이러한 태스크가 완료되면 데이터베이스를 재시작하십시오. 이 상태에서 오랫동안 데이터베이스를 작동하지 마십시오.

시스템 버퍼 풀만 사용하여 데이터베이스를 시작하지 못하도록 방지하려면 **DB2_OVERRIDE_BPF** 레지스트리 변수를 사용하여 사용 가능한 메모리의 사용을 최적화하십시오.

혁신적 페이지 정리

DB2 버전 8.1.4부터 시스템에서 페이지 정리를 구성하는 대체 방법이 있습니다. 이 방법을 통해, 페이지 클리너는 지정된 특정 시점에 작성되는 더티 페이지를 선택하는 데 있어 보다 혁신적으로 동작합니다.

이 혁신적 페이지 정리 방법은 두 가지 주요사항에서 디폴트 페이지 정리 방법과 다릅니다.

- 페이지 클리너는 **chngpgs_thresh** 데이터베이스 구성 매개변수의 값에 더 이상 응답하지 않습니다.

  적당한 희생(victim) 페이지의 수가 승인할 수 있는 값 아래로 떨어질 경우, 페이지 클리너에서는 전체 버퍼 풀을 검색하여 잠재적 희생(victim) 페이지를 작성하고 이러한 페이지의 위치를 에이전트에 알립니다.
- 페이지 클리너가 로그 프로그램에 의해 발행된 로그 시퀀스 번호(LSN) 갭 트리거에 더 이상 응답하지 않습니다.

  버퍼 풀에서 가장 오래된 페이지를 갱신한 로그 레코드와 현재 로그 위치 지정 간 로그 스페이스의 양이 **softmax** 데이터베이스 구성 매개변수에 의해 허용된 값을 초과할 경우, 데이터베이스가 *LSN 갭*을 경험하고 있다고 합니다.

  디폴트 페이지 정리 방법에서, LSN 갭을 감지하는 로그 프로그램은 페이지 클리너가 LSN 갭의 원인이 되고 있는 모든 페이지를 작성하도록 트리거합니다. 즉, 페이지 클리너에서 **softmax**의 값에 의해 허용된 것보다 더 오래된 해당 페이지를 작성합니다. 페이지 클리너는 많은 수의 페이지를 쓰며 활동의 버스트와 대기 사이를 오고 갑니다. 이는 입출력 서브시스템의 포화를 초래하여 페이지를 읽거나 쓰는 다른 에이전트에 영향을 줄 수 있습니다. 또한 LSN 갭이 감지될 때쯤에는 페이지 클리너가 충분히 빨리 정리할 수 없고, DB2에 로그 스페이스가 바닥날 수도 있습니다.

  혁신적 페이지 정리 방법은 보다 오랜 기간에 걸쳐 동일한 수의 쓰기를 분배함으로써 이 동작을 조정합니다. 페이지 클리너는 LSN 갭의 원인이 되는 페이지뿐만 아니라 현재 활동 레벨을 기반으로 임박한 LSN 갭의 원인이 될 가능성이 있는 페이지도 정리하여 이를 수행합니다.

새 페이지 정리 방법을 사용하려면 **DB2_USE_ALTERNATE_PAGE_CLEANING** 레지스트리 변수를 설정하십시오.

갱신 성능 향상

에이전트가 페이지를 갱신하면 데이터베이스 관리 프로그램은 프로토콜을 사용하여 트랜잭션에 필요한 입출력을 최소화하고 복구 가능성을 보장합니다.

이 프로토콜에는 다음의 단계가 포함됩니다.

1. 갱신할 페이지는 배타적 잠금을 통해 고정되고 래치됩니다. 변경사항을 실행 취소하고 재실행하는 방법을 설명한 로그 레코드가 로그 버퍼에 기록됩니다. 이 조치의 일부분으로 로그 시퀀스 번호(LSN)를 확보하고 갱신 중인 페이지의 헤더에 저장합니다.
2. 갱신사항이 페이지에 적용됩니다.

3. 페이지의 래치가 해제됩니다. 페이지의 변경사항이 아직 디스크에 기록되지 않았으므로 페이지는『더티』페이지로 간주됩니다.
4. 로그 버퍼가 갱신됩니다. 로그 버퍼 및 더티 데이터 페이지의 데이터가 모두 디스크에 기록됩니다.

성능 향상을 위해 시스템 로드가 잠시 중지되었을 때나 복구 가능성 보장 또는 복구 시간 제한을 위해 필요할 때까지 이러한 입출력 조작은 지연됩니다. 자세히 표현하면 다음과 같은 경우 더티 페이지가 디스크에 기록됩니다.

- 다른 에이전트에서 희생(victim)으로 선택한 경우
- 페이지에서 페이지 클리너가 작동되는 경우. 다음과 같은 경우에 이러한 상태가 발생할 수 있습니다.
  - 다른 에이전트가 펭지ㅣ를 희생(victim)으로 선택한 경우
  - **chngpgs_thresh** 데이터베이스 구성 매개변수 값을 초과하여 비동기 페이지 클리너가 시작되고 변경된 페이지를 디스크에 기록하는 경우. 데이터베이스의 혁신적 페이지 정리가 사용 가능한 경우 이 값은 관계가 없으며 페이지 정리를 트리거하지 않습니다.
  - **softmax** 데이터베이스 구성 매개변수 값을 초과하여 비동기 페이지 클리너가 시작되고 변경된 페이지를 디스크에 기록하는 경우. 데이터베이스의 혁신적 페이지 정리가 사용 가능하며 데이터베이스에 대한 페이지 클리너 수가 올바로 구성된 경우 이 값을 초과하지 않습니다.
  - 정리 페이지 수가 너무 낮아진 경우. 페이지 클리너는 혁신적 페이지 정리에서 이러한 상태일 때에만 반응합니다.
  - 더티 페이지가 현재 또는 이후에 LSN 갭 조건에 기여하는 경우. 페이지 클리너는 혁신적 페이지 정리에서 이러한 상태일 때에만 반응합니다.
- 페이지가 NOT LOGGED INITIALLY절을 사용하여 정의된 테이블의 일부분이며 갱신 이후에 COMMIT문이 표시된 경우. COMMIT문이 실행된 경우 변경된 모든 페이지는 디스크로 플러시되어 복구 가능성이 보장됩니다.

버퍼 풀로 데이터 프리페치

페이지 *프리페치*는 응용프로그램에 필요하다는 예상에 따라 디스크에서 하나 이상의 페이지를 검색하는 것을 의미합니다.

인덱스 및 데이터 페이지를 버퍼 풀로 프리페치하면 입출력 대기 시간을 줄여서 성능을 향상시킬 수 있습니다. 또한 병렬 입출력은 프리페치 효율성을 향상시킵니다.

두 가지 범주의 프리페치가 있습니다.

- *순차 프리페치*는 응용프로그램에서 페이지를 요청하기 전에 버퍼 풀로 연속 페이지를 읽습니다.

- *리스트 프리페치*(*리스트 순차 프리페치*라고도 함)는 비연속 데이터 페이지 세트를 효율적으로 프리페치합니다.

데이터 페이지 프리페치는 하나 또는 일부 연속 페이지를 검색할 때 사용되지만 하나의 데이터 페이지만 응용프로그램으로 전송되는 데이터베이스 관리 프로그램 에이전트 읽기와 다릅니다.

프리페치 및 파티션 내 병렬 처리

프리페치는 인덱스 또는 테이블 스캔 시 여러 서브에이전트를 사용하는 파티션 내 병렬 처리의 성능에 중요한 영향을 미칩니다. 이러한 병렬 스캔 데이터 사용률이 크게 증가하므로 프리페치율도 높아야 합니다.

부적합한 프리페치의 비용은 직렬 스캔보다 병렬 스캔의 경우에 높습니다. 직렬 스캔 중 프리페치가 발생하지 않은 경우 에이전트가 입출력 대기 중이므로 쿼리 실행이 느려집니다. 병렬 스캔 중 프리페치가 발생하지 않은 경우 하나의 서브에이전트가 입출력 대기 중인 동안 모든 서브에이전트가 대기해야 할 수도 있습니다.

이 컨텍스트에서 중요도로 인해 파티션 내 병렬 처리에서 프리페치가 보다 공격적으로 수행됩니다. 순차 감지 메커니즘은 인접 페이지들 간 큰 갭을 허용하므로 페이지를 순차 페이지로 간주할 수 있습니다. 이러한 갭의 너비는 스캔에 관련된 서브에이전트 수로 인해 증가합니다.

순차 프리페치:

단일 입출력 조작을 사용하여 버퍼 풀로 여러 연속 페이지를 읽으면 응용프로그램 오버헤드를 상당히 줄일 수 있습니다.

데이터베이스 관리 프로그램이 순차 입출력이 적합하고 프리페치가 성능을 향상시킬 수 있다고 판별하면 프리페치가 시작됩니다. 테이블 스캔 및 테이블 정렬과 같은 경우 데이터베이스 관리 프로그램은 순차 프리페치를 자동으로 선택합니다. 테이블 스캔이 필요할 수도 있는 다음의 예는 순차 프리페치의 좋은 예입니다.

```
SELECT NAME FROM EMPLOYEE
```

순차 감지

순차 프리페치가 성능을 향상시킨다는 점이 즉시 명확하게 나타나지는 않습니다. 이러한 경우 데이터베이스 관리 프로그램은 입출력을 모니터하고 순차 페이지 읽기가 발생하면 프리페치를 활성화할 수 있습니다. *순차 감지*라고 하는 이러한 순차 프리페치 유형은 인덱스 및 데이터 페이지에 모두 적용됩니다. **seqdetect** 데이터베이스 구성 매개변수를 사용하여 데이터베이스 관리 프로그램의 순차 감지 수행 여부를 제어하십시오.

예를 들어, 순차 감지가 사용 가능한 경우 다음의 SQL문은 순차 프리페치를 이용할 수 있습니다.

```
SELECT NAME FROM EMPLOYEE
WHERE EMPNO BETWEEN 100 AND 3000
```

이 예에서 옵티마이저는 EMPNO 컬럼에서 인덱스를 사용하여 테이블 스캔을 시작했을 수 있습니다. 이 인덱스와 관련하여 테이블이 많이 클러스터된 경우 데이터 페이지 읽기는 거의 순차적으로 발생하며 프리페치가 성능을 향상시킬 수 있습니다. 마찬가지로 여러 인덱스 페이지를 조사해야 하며 데이터베이스 관리 프로그램이 인덱스 페이지의 순차 페이지 읽기가 발생한 것을 감지한 경우 인덱스 페이지 프리페치가 발생할 수 있습니다.

테이블 스페이스에 대한 PREFETCHSIZE 옵션의 포함

CREATE TABLESPACE 또는 ALTER TABLESPACE문의 PREFETCHSIZE절을 사용하여 데이터 프리페치를 수행할 때 테이블 스페이스에서 읽을 프리페치 페이지의 수를 지정할 수 있습니다. 지정하는 값(또는 'AUTOMATIC')은 SYSCAT.TABLESPACES 카탈로그 뷰의 PREFETCHSIZE 컬럼에 저장됩니다.

PREFETCHSIZE 값을 테이블 스페이스 컨테이너 수, 각 컨테이너의 실제 디스크 수(RAID 디바이스를 사용하는 경우) 및 테이블 스페이스의 EXTENTSIZE 값(데이터베이스 관리 프로그램이 다른 컨테이너를 사용하기 전에 컨테이너에 기록하는 페이지 수)의 배수로 명시적으로 설정하는 것이 좋습니다. 예를 들어, Extent *크기*가 16페이지이고 테이블 스페이스에 두 개의 컨테이너가 있는 경우 프리페치 *크기*를 32페이지로 설정할 수 있습니다. 컨테이너 당 다섯 개의 실제 디스*크*가 있는 경우 프리페치 *크기*를 160페이지로 설정할 수 있습니다.

데이터베이스 관리 프로그램은 버퍼 풀 사용량을 모니터하여 프리페치가 다른 작업 단위에 필요한 페이지를 버퍼 풀에서 제거하지 않는지 확인합니다. 문제를 방지하기 위해 데이터베이스 관리 프로그램은 프리페치 페이지 수를 테이블 스페이스에 지정된 수보다 적게 제한할 수 있습니다.

프리페치 *크기*는 성능에 상당한 영향을 미칠 수 있으며 특히 대형 테이블 스캔의 경우에 해당합니다. 데이터베이스 시스템 모니터 모니터 및 기타 시스템 모니터 도구를 사용하여 테이블 스페이스의 프리페치 *크기*를 조정하십시오. 다음과 같은 여부에 대한 정보를 수집할 수 있습니다.

- 운영 체제에 사용 가능한 모니터링 도구를 사용하여 쿼리에 대한 입출력 대기가 있는지 여부
- 데이터베이스 시스템 모니터에서 제공하는 **pool_async_data_reads**(버퍼 풀 비동기 데이터 읽기) 데이터 요소를 확인하여 프리페치가 발생하는지 여부

쿼리가 데이터를 프리페치하는 동안 입출력 대기가 있는 경우 PREFETCHSIZE의 값을 늘릴 수 있습니다. 프리페처가 이러한 입출력 대기의 원인이 아닌 경우 PREFETCHSIZE 값을 늘려도 쿼리의 성능이 향상되지 않습니다.

모든 프리페치 유형에서 프리페치 *크기*가 테이블 *스페이스*의 Extent *크기*에 대한 배수이며 Extent가 별도의 컨테이너에 있는 경우 다중 입출력 조작이 병렬식으로 수행될 수 있습니다. 성능 향상을 위해 개별 물리 디바이스를 사용하도록 컨테이너를 구성하십시오.

향상된 순차 프리페치용 블록 기반 버퍼 풀:

디스크에서 페이지를 프리페치하려면 입출력 오버헤드로 인해 많은 비용이 듭니다. 처리가 입출력과 오버랩되면 처리량을 상당히 향상할 수 있습니다.

대부분의 플랫폼은 디스크에서 인접하지 않은 메모리 부분들로 인접한 페이지를 읽는 높은 성능의 원형을 제공합니다. 이러한 원형을 보통 *분산 읽기* 또는 *벡터 입출력*이라고 합니다. 일부 플랫폼에서 이러한 원형의 성능은 대형 블록 *크기*에서 입출력을 수행하는 경우와 비교할 수 없습니다.

디폴트로 버퍼 풀은 페이지 기반입니다. 즉, 디스크에서 인접하는 페이지가 메모리에서 인접하지 않는 페이지로 프리페치됩니다. 디스크에서 버퍼 풀의 인접하는 페이지로 인접하는 페이지를 읽을 수 있는 경우 순차 프리페치가 향상될 수 있습니다.

이러한 용도로 블록 기반 버퍼 풀을 작성할 수 있습니다. 블록 기반 버퍼 풀은 페이지 영역과 블록 영역으로 구성되어 있습니다. 페이지는 비연속 프리페치 워크로드의 경우에 필요합니다. 블록 영역은 블록들로 구성되어 있으며 각 블록에는 *블록 크기*라고 하는 지정된 수의 인접하는 페이지가 들어 있습니다.

블록 기반 버퍼 풀의 최적 사용은 지정된 블록 *크기*에 따라 다릅니다. 블록 *크기*는 순차 프리페치를 수행하는 입출력 서버가 블록 기반 입출력 수행을 고려하는 단위입니다. Extent는 컨테이너에서 테이블 *스페이스*가 스트라이프되는 단위입니다. Extent *크기*가 다른 여러 테이블 *스페이스*를 동일한 블록 *크기*로 정의된 버퍼 풀로 바인드할 수 있으므로 버퍼 풀 메모리를 효율적으로 사용하는 데 Extent *크기*와 블록 *크기*가 상호작용하는 방식을 고려하십시오. 다음과 같은 경우 버퍼 풀 메모리를 낭비할 수 있습니다.

- 프리페치 요청 *크기*를 판별하는 Extent *크기*가 버퍼 풀에 지정된 블록 *크기*보다 작은 경우
- 프리페치 요청의 일부 페이지가 버퍼 풀의 페이지 영역에 이미 있는 경우

입출력 서버가 각 버퍼 풀 블록에서 페이지 낭비를 허용하지만 너무 많은 페이지를 낭비할 경우 입출력 서버가 버퍼 풀의 페이지 영역으로 비블록 기반 프리페치를 수행하여 성능이 준최적 상태가 됩니다.

최적의 성능을 위해서는 블록 *크기*가 테이블 *스페이스* Extent *크기*와 동일한 버퍼 풀로 동일한 Extent *크기*의 테이블 *스페이스*를 바인드하십시오. Extent *크기*가 블록 *크기*보다 *크지*만 Extent *크기*가 블록 *크기*보다 작지 않은 경우 성능이 좋아질 수 있습니다.

블록 기반 버퍼 풀을 작성하려면 CREATE BUFFERPOOL 또는 ALTER BUFFERPOOL문을 사용하십시오.

주: 블록 기반 버퍼 풀은 순차 프리페치에 사용됩니다. 응용프로그램이 순차 프리페치를 사용하지 않는 경우 버퍼 풀의 블록 영역을 낭비합니다.

리스트 프리페치:

리스트 프리페치(또는 *리스트 순차 프리페치*)는 데이터 페이지가 인접하지 않은 경우에도 이러한 페이지에 효율적으로 액세스하는 방법입니다.

리스트 프리페치는 단일 또는 다중 인덱스 액세스와 함께 사용할 수 있습니다.

옵티마이저가 인덱스를 사용하여 행에 액세스하는 경우 인덱스에서 모든 행 ID(RID)를 확보할 때까지 데이터 페이지 읽기를 지연시킬 수 있습니다. 예를 들어, 옵티마이저는 인덱스 스캔을 수행하여 검색할 행 및 데이터 페이지를 판별할 수 있습니다.

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

그런 다음 아래의 검색 기준을 사용할 수 있습니다.

```
WHERE NAME BETWEEN 'A' and 'I'
```

이 인덱스에 따라 데이터가 클러스터되지 않은 경우 리스트 프리페치에는 인덱스 스캔에서 확보한 RID 목록을 정렬하는 단계가 포함됩니다.

프리페치 및 병렬 처리를 위한 입출력 서버 구성:

프리페치를 사용하기 위해 데이터베이스 관리 프로그램은 *입출력 서버*라고 하는 개별 제어 스레드를 시작하여 데이터 페이지를 읽습니다.

따라서 쿼리 처리는 두 개의 병렬 활동 즉, 데이터 처리(CPU)와 데이터 페이지 입출력으로 구분됩니다. 입출력 서버는 CPU 활동에서 프리페치 요청을 기다립니다. 이러한 프리페치 요청에는 쿼리를 충족시키는 데 필요한 입출력에 대한 설명이 포함되어 있습니다.

충분한 입출력 서버를 구성하면(**num_ioservers** 데이터베이스 구성 매개변수 사용) 프리페치를 이용할 수 있는 쿼리 성능을 상당히 향상시킬 수 있습니다. 병렬 입출력 기회를 최대한으로 증가시키려면 **num_ioservers**를 최소한 데이터베이스의 실제 디스크 수로 설정하십시오.

입출력 서버 수를 적게 추정하는 것보다 크게 추정하는 것이 좋습니다. 추가 입출력 서버를 지정하는 경우 이러한 서버를 사용하지 않으며 메모리 페이지는 페이지 아웃되고 성능에 영향을 미치지 않습니다. 각 입출력 서버 프로세스에는 번호를 지정합니다. 데

이터베이스 관리 프로그램은 항상 번호가 가장 낮은 프로세스를 사용하므로 번호가 높은 일부 프로세스는 거의 사용되지 않았을 수 있습니다.

필요한 입출력 서버 수를 추정하려면 다음 사항을 고려하십시오.

- 입출력 서버 큐로 프리페치 요청을 동시에 기록할 수 있는 데이터베이스 에이전트 수
- 입출력 서버를 병렬식으로 작동할 수 있는 최고 수준

데이터베이스 관리 프로그램이 시스템 구성에 따라 지능형 값을 선택할 수 있도록 **num_ioservers**의 값을 AUTOMATIC으로 설정하십시오.

비동기 입출력을 위한 구성

일부 플랫폼에서 데이터베이스 관리 프로그램은 비동기 입출력(AIO)을 사용하여 페이지 정리 및 프리페치와 같은 활동의 성능을 향상시킵니다. AIO는 여러 디스크에 데이터를 분산시키는 경우 가장 효율적입니다. 기본 운영 체제 AIO 인프라스트럭처를 조정해도 성능이 향상됩니다.

예를 들어, AIX의 경우 운영 체제에서 AIO를 조정할 수 있습니다. AIO가 SMS 또는 DMS 파일 컨테이너에서 작동되는 경우 AIO 서버라고 하는 운영 체제 프로세스가 입출력을 관리합니다. 이러한 서버 수가 많지 않으므로 AIO 요청 수가 제한되어 활용할 수 있는 AIO의 이점이 크지 않습니다. AIX에서 AIO 서버 수를 구성하려면 **smit** AIO **minservers** 및 AIO **maxservers** 매개변수를 사용하십시오.

병렬 입출력을 사용한 프리페치 표시:

입출력 서버를 사용하여 버퍼 풀로 데이터를 프리페치합니다.

이 프로세스는 122 페이지의 그림 20에 표시되어 있습니다.

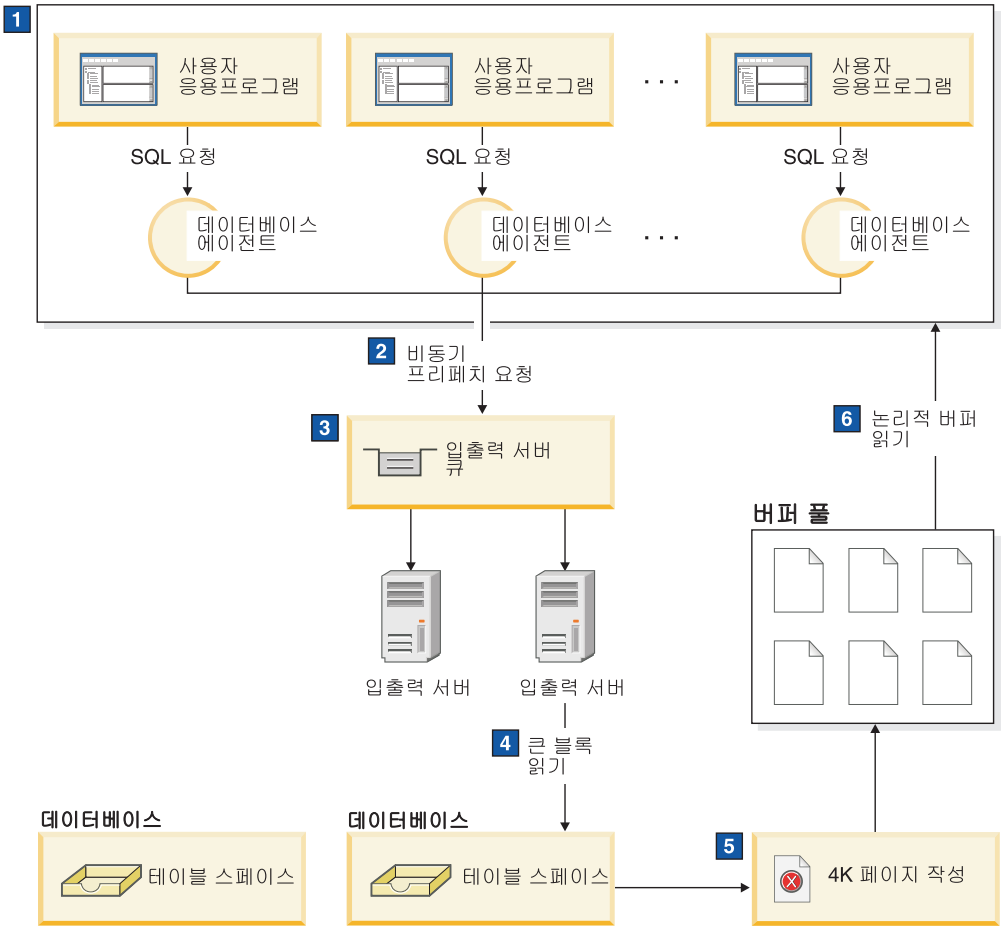


그림 20. 입출력 서버를 사용한 데이터 프리페치

1 사용자 응용프로그램은 데이터베이스 관리 프로그램이 사용자 응용프로그램에 지정한 데이터베이스 에이전트로 요청을 전달합니다.

2, **3**

데이터베이스 에이전트는 프리페치를 사용하여 요청을 충족시키는 데 필요한 데이터를 확보해야 한다는 점을 판별하고 입출력 서버 큐에 프리페치 요청을 기록합니다.

4, **5**

첫 번째 사용 가능한 입출력 서버는 큐에서 프리페치 요청을 읽은 후 테이블 스페이스에서 버퍼 풀로 데이터를 읽습니다. 테이블 스페이스에서 데이터를 동시에 페치할 수 있는 입출력 서버 수는 큐의 프리페치 요청 수와 **num_ioservers** 데이터베이스 구성 매개변수에 지정된 입출력 서버 수에 따라 다릅니다.

6 데이터베이스 에이전트는 버퍼 풀의 데이터 페이지에 대해 필요한 조작을 수행하고 사용자 응용프로그램으로 결과를 리턴합니다.

병렬 입출력 관리:

테이블 스페이스의 컨테이너가 여러 개인 경우 데이터베이스 관리 프로그램은 *병렬 입출력*을 시작할 수 있으므로 데이터베이스 관리 프로그램은 다중 입출력 서버를 사용하여 단일 쿼리의 입출력 요구사항을 처리합니다.

각 입출력 서버는 개별 컨테이너의 입출력 워크로드를 처리하므로 여러 컨테이너를 병렬식으로 읽을 수 있습니다. 병렬 입출력을 통해 입출력 처리량이 상당히 향상될 수 있습니다.

개별 입출력 서버가 각 컨테이너의 워크로드를 처리할 수 있지만 병렬 입출력을 수행할 수 있는 입출력 서버의 실제 수는 요청된 데이터가 분산되는 물리 디바이스의 수로 제한됩니다. 이러한 이유로 인해 물리 디바이스 수만큼 입출력 서버가 필요합니다.

다음과 같은 경우 병렬 입출력이 다르게 시작됩니다.

- *순차 프리페치*의 경우 프리페치 *크기*가 테이블 스페이스 Extent *크기*의 배수이면 병렬 입출력이 시작됩니다. 각 프리페치는 Extent 경계를 따라 작은 요청으로 나뉩니다. 그런 다음 이러한 작은 요청은 다른 입출력 서버로 지정됩니다.
- *리스트 프리페치*의 경우 각 페이지 목록은 데이터 페이지가 저장된 컨테이너에 따라 작은 목록으로 나뉩니다. 그런 다음 이러한 작은 목록은 다른 입출력 서버로 지정됩니다.
- *데이터베이스 또는 테이블 스페이스 백업 및 리스토어*의 경우 병렬 입출력 요청 수는 백업 버퍼 *크기*를 Extent *크기*로 나눈 값과 동일하며 컨테이너 수와 동일한 최대 값까지 가능합니다.
- *데이터베이스 또는 테이블 스페이스 리스토어*의 경우 병렬 입출력 요청이 시작되고 순차 프리페치에 대해 수행하는 방법과 동일하게 나눕니다. 데이터는 버퍼 풀로 리스토어되지 않습니다. 리스토어 버퍼에서 디스크로 직접 이동합니다.
- 데이터를 로드할 때 DISK_PARALLELISM 명령 옵션을 사용하여 입출력 병렬 처리 레벨을 지정할 수 있습니다. 이 옵션을 지정하지 않은 경우 데이터베이스 관리 프로그램은 테이블과 연관된 모든 테이블 스페이스의 누적된 테이블 스페이스 컨테이너 수에 기초한 디폴트값을 사용합니다.

최적의 병렬 입출력 성능을 위해 다음을 확인하십시오.

- 충분한 입출력 서버가 있는지 확인하십시오. 데이터베이스의 모든 테이블 스페이스에 사용되는 컨테이너 수보다 약간 많은 입출력 서버를 지정하십시오.
- Extent *크기* 및 프리페치 *크기*가 테이블 스페이스에 적합한지 확인하십시오. 버퍼 풀의 초과 사용을 예방하려면 프리페치 *크기*가 너무 크지 않아야 합니다. 최적의 *크기*는 Extent *크기*, 각 컨테이너의 실제 디스크 수(RAID 디바이스를 사용하는 경우) 및 테이블 스페이스 컨테이너 수의 배수입니다. Extent *크기*는 매우 작아야 하며 좋은 값의 범위는 8 - 32페이지입니다.
- 컨테이너가 개별 실제 드라이브에 상주하는지 확인하십시오.

- 병렬 처리 수준을 일관성 있게 유지하기 위해 모든 컨테이너가 동일한 *크기*인지 확인하십시오.

  하나 이상의 컨테이너가 다른 컨테이너보다 작은 경우 최적의 병렬 프리페치를 위한 가능성을 줄입니다. 다음 예를 고려해보십시오.

  – 작은 컨테이너가 가득 찬 후 추가 데이터를 나머지 컨테이너에 저장하므로 컨테이너의 균형이 맞지 않습니다. 데이터를 프리페치할 수 있는 컨테이너 수가 총 컨테이너 수보다 적을 수 있으므로 컨테이너의 균형이 맞지 않으면 병렬 프리페치의 성능이 낮아집니다.
  – 나중에 작은 컨테이너를 추가하고 데이터를 다시 조정할 경우 작은 컨테이너에는 다른 컨테이너보다 적은 데이터가 들어 있습니다. 다른 컨테이너에 비해 데이터의 양이 작으면 병렬 프리페치가 최적화되지 않습니다.
  – 한 컨테이너가 *크고* 다른 모든 컨테이너가 가득 찬 경우 큰 컨테이너에만 추가 데이터를 저장할 수 있습니다. 데이터베이스 관리 프로그램은 병렬 프리페치를 사용하여 이 추가 데이터에 액세스할 수 없습니다.
- 파티션 내 병렬 처리를 사용하는 경우 충분한 입출력 공간이 있는지 확인하십시오. SMP 머신에서 파티션 내 병렬 처리는 여러 프로세서에서 쿼리를 실행하여 쿼리의 경과 시간을 줄일 수 있습니다. 각 프로세서를 사용 중인 상태로 유지하려면 충분한 입출력 공간이 필요합니다. 이러한 입출력 공간을 제공하려면 보통 추가 실제 드라이브가 필요합니다.

  프리페치가 자주 발생하고 입출력 공간을 효과적으로 사용하도록 프리페치 *크기*가 커야 합니다.

  필요한 실제 드라이브 수는 드라이브 및 입출력 버스의 속도 및 용량과 프로세서의 속도에 따라 다릅니다.

AIX에서 IOCP 구성:

AIX 5.3 TL9 SP2 및 AIX 6.1 TL2는 기본 설치의 일부로 포함된 I/O 완료 포트(IOCP) 파일 세트를 갖습니다. 그러나 최소 운영 체제 요구사항이 새 운영 체제 설치를 사용하지 않고 운영 체제 업그레이드를 사용하여 적용된 경우, I/O 완료 포트(IOCP)를 별도로 구성해야 합니다.

1. 다음과 같이 lslpp 명령을 실행하여 IOCP 모듈이 시스템에 설치되어 있는지 확인하십시오.

   ```
   $ lslpp -l bos.iocp.rte
   ```

   결과 출력이 아래와 유사해야 합니다.

   ```
   Fileset                      Level     State       Description
   ----------------------------------------------------------------------------
   Path: /usr/lib/objrepos
     bos.iocp.rte               5.3.9.0   APPLIED     I/O Completion Ports API
   ```

```
Path: /etc/objrepos
  bos.iocp.rte             5.3.0.50  COMMITTED   I/O Completion Ports API
```

2. 다음과 같이 lsdev 명령을 실행하여 IOCP 상태가 Available(사용 가능)인지 확인하십시오.

   ```
   $ lsdev -Cc iocp
   ```

   결과 출력이 아래와 정확히 일치해야 합니다.

   ```
   iocp0 Available  I/O Completion Ports
   ```

   IOCP 상태가 Defined(정의된 상태)인 경우, 상태를 다음과 같이 Available (사용 가능)로 변경하십시오.

   a. 서버에 루트로 로그인한 후 다음 명령을 실행하십시오.

      ```
      # smitty iocp
      ```

   b. Change / Show Characteristics of I/O Completion Ports를 선택하십시오.

   c. 시스템 재시작 시 구성된 상태를 Defined에서 Available로 변경하십시오.

   d. lsdev 명령을 다시 실행하여 IOCP 포트 상태가 Available로 변경되었는지 확인하십시오.

첫 번째 사용자 연결 시나리오에서 데이터베이스 비활성화 동작

사용자가 처음 데이터베이스에 연결할 때 데이터베이스가 활성화됩니다. 단일 파티션 환경에서는 데이터베이스가 메모리에 로드되며 마지막 사용자가 연결을 끊을 때까지 이 상태를 유지합니다. 동일한 동작이 다중 파티션 환경(첫 번째 사용자 연결이 해당 데이터베이스에 대한 로컬 및 카탈로그 파티션 둘 다에서 데이터베이스를 활성화함)에 적용됩니다.

마지막 사용자가 연결을 끊으면 데이터베이스가 로컬 및 리모트 파티션(이 사용자는 데이터베이스에 대한 마지막 활성 사용자 연결임) 둘 다에서 종료됩니다. 첫 번째 연결 및 마지막 연결을 기반으로 하는 이 데이터베이스 활성화 및 비활성화를 *내재된 활성화*라고 합니다. 활성화는 첫 번째 사용자 연결로 시작되며, 사용자가 CONNECT RESET를 실행하거나 사용자가 연결을 종료 또는 삭제(drop)하여 결과적으로 데이터베이스가 내재적으로 비활성화될 때까지 활성화가 유효합니다.

데이터베이스를 메모리로 로드하는 프로세스는 매우 복잡합니다. 버퍼 풀을 포함한 모든 데이터베이스 구성요소 초기화를 포함하며, 특히 성능에 민감한 환경에서는 최소화해야 하는 처리 유형입니다. 한 데이터베이스 파티션에서 실행된 쿼리가 목표 데이터 세트의 일부를 포함하는 다른 파티션에 도달하는 다중 파티션 환경에서는 이 동작이 특히 중요합니다. 해당 데이터베이스 파티션은 사용자 응용프로그램의 연결 및 연결 끊기 동작에 따라 활성화되거나 비활성화됩니다. 사용자가 처음으로 데이터베이스 파티션에 도달하는 쿼리를 실행하는 경우, 쿼리는 해당 파티션을 처음 활성화하는 비용을 가정합니

다. 해당 사용자가 연결을 끊는 경우, 해당 리모트 파티션에 대해 다른 연결이 이전에 설정되지 않았으면 데이터베이스가 비활성화됩니다. 다음 수신 쿼리가 해당 리모트 파티션에 액세스해야 하는 경우, 해당 파티션의 데이터베이스가 먼저 활성화되어야 합니다. 각 데이터베이스(또는 적용 가능한 경우 데이터베이스 파티션) 활성화 및 비활성화에 대해 이 비용이 발생합니다.

이 동작의 유일한 예외는 사용자가 ACTIVATE DATABASE 명령을 실행하여 데이터베이스를 명시적으로 활성화하도록 선택하는 경우 발생합니다. 이 명령이 완료된 후에는 마지막 사용자가 연결을 끊더라도 데이터베이스가 메모리에 남아 있습니다. 이는 단일 및 다중 파티션 환경 둘 다에 적용됩니다. 이러한 데이터베이스를 비활성화하려면 DEACTIVATE DATABASE 명령을 실행하십시오. 두 명령 모두 범위면에서 전역적입니다(적용 가능한 경우 모든 데이터베이스 파티션에서 데이터베이스를 활성화하거나 비활성화함을 의미). 메모리로 데이터베이스를 로드하는 처리 위주의 특성이 제공되면, 데이터베이스 연결을 통한 내재된 활성화에 의존하지 말고 ACTIVATE DATABASE 명령을 사용하여 데이터베이스를 명시적으로 활성화하십시오.

정렬 성능 조정

쿼리에는 주로 정렬 또는 그룹화된 결과가 필요하므로 정렬 힙을 적절하게 구성하는 것이 좋은 쿼리 성능에 중요합니다.

다음과 같은 경우 정렬이 필요합니다.

- 요청된 순서를 충족시키는 인덱스가 없는 경우(예: ORDER BY절을 사용하는 SELECT문)
- 인덱스가 있지만 인덱스를 사용하는 것보다 정렬이 효율적인 경우
- 인덱스가 작성되는 경우
- 인덱스가 삭제되어 인덱스 페이지 번호가 정렬되는 경우

정렬에 영향을 주는 요소

다음과 같은 요인이 정렬 성능에 영향을 줍니다.

- 다음의 구성 매개변수에 대한 설정
  - 각 정렬에 사용할 메모리의 양을 지정하는 정렬 힙 *크기*(**sortheap**)
  - 인스턴스에서 정렬에 사용할 수 있는 전체 메모리 양을 제어하는 정렬 힙 임계값(**sheapthres**) 및 공유 정렬의 정렬 힙 임계값(**sheapthres_shr**)
- 대량의 정렬이 필요한 워크로드의 명령문 수
- 불필요한 정렬을 방지하는 데 도움이 되는 인덱스가 있거나 없는 경우
- 정렬의 필요성을 최소화하지 않는 응용프로그램 논리의 사용
- 정렬 성능을 향상시키지만 명령문이 파티션 내 병렬 처리를 사용하는 경우에만 발생할 수 있는 병렬 정렬

- 정렬의 *오버플로우* 여부. 정렬된 데이터가 정렬을 수행할 때마다 할당되는 메모리 블록인 정렬 힙에 맞지 않는 경우 데이터는 데이터베이스가 소유하는 임시 테이블로 오버플로우됩니다.
- 정렬 결과의 *파이프* 여부. 정렬된 목록을 저장할 임시 테이블이 없어도 정렬된 데이터를 직접 리턴할 수 있는 경우 파이프 정렬입니다.

  파이프 정렬에서 응용프로그램이 정렬과 연관된 커서를 닫기 전까지 정렬 힙은 사용 가능해지지 않습니다. 파이프 정렬은 커서가 닫히기 전까지 계속해서 메모리를 모두 사용할 수 있습니다.

정렬 메모리에서 정렬을 완전히 수행할 수 있지만 초과 페이지 스와핑이 발생할 수 있습니다. 이 경우 대형 정렬 힙의 장점은 없어집니다. 이러한 이유로 인해 정렬 구성 매개변수를 조정할 때마다 운영 체제 모니터를 사용하여 시스템 페이징의 변경사항을 추적해야 합니다.

정렬 성능을 관리하는 기술

정렬이 중요한 성능 문제가 되는 특정 응용프로그램 및 명령문을 식별하십시오.

1. 총 정렬 시간이 가장 긴 응용프로그램을 식별할 수 있도록 응용프로그램 및 명령문 레벨에서 이벤트 모니터를 설정하십시오.
2. 이러한 각 응용프로그램에서 *총 정렬 시간*이 가장 긴 명령문을 찾으십시오.

   또한 Explain 테이블을 검색하여 정렬 조작이 있는 쿼리를 식별할 수 있습니다.
3. 이러한 명령문을 디자인 어드바이저에 대한 입력으로 사용하십시오. 인덱스를 식별하고 작성할 수 있으므로 정렬의 필요성이 줄어듭니다.

자체 조정 메모리 관리자(STMM)를 사용하면 정렬에 필요한 메모리 자원을 자동 및 동적으로 할당하고 할당 해제할 수 있습니다. 이 기능을 사용하려면 다음을 수행하십시오.

- **self_tuning_mem** 구성 매개변수를 ON으로 설정하여 데이터베이스의 자체 조정 메모리를 사용하십시오.
- **sortheap** 및 **sheapthres_shr** 구성 매개변수를 AUTOMATIC으로 설정하십시오.
- **sheapthres** 구성 매개변수를 0으로 설정하십시오.

또한 데이터베이스 시스템 모니터 및 벤치마킹 기술을 사용하면 **sortheap**, **sheapthres_shr** 및 **sheapthres** 구성 매개변수를 설정하는 데 도움이 됩니다. 각 데이터베이스 관리 프로그램 및 각 데이터베이스마다 다음을 수행하십시오.

1. 대표적인 워크로드를 설정하고 실행하십시오.
2. 적용 가능한 각 데이터베이스마다 벤치마크 워크로드 기간에 다음의 성능 변수에 대한 평균 값을 수집하십시오.
   - 사용 중인 총 정렬 힙(**sort_heap_allocated** 모니터 요소의 값)

- 활성 정렬 및 활성 해시 조인(**active_sorts** 및 **active_hash_joins** 모니터 요소의 값)

3. 각 데이터베이스마다 **sortheap**을 *사용 중인 총 정렬 힙* 평균으로 설정하십시오.

   주: 정렬에 긴 키를 사용하는 경우 **sortheap** 구성 매개변수의 값을 늘려야 할 수도 있습니다.

4. **sheapthres**를 설정하십시오. 적합한 *크기*를 추정하려면 다음을 수행하십시오.
   a. 인스턴스에서 최대 **sortheap** 값이 있는 데이터베이스를 판별하십시오.
   b. 이 데이터베이스에 대한 정렬 힙의 평균 *크기*를 판별하십시오.

      이 *크기*를 판별하기 너무 어려운 경우 최대 정렬 힙의 80%를 사용하십시오.
   c. **sheapthres**를 평균 활성 정렬 수에 위에서 계산한 정렬 힙의 평균 *크기*를 곱한 값으로 설정하십시오. 이 설정은 권장되는 초기 설정입니다. 그런 다음 벤치마크 기술을 사용하여 이 값을 세부 조정할 수 있습니다.

데이터 구성

시간이 지나면서, 테이블의 데이터가 분할되어 레코드가 점점 더 많은 데이터 페이지에 분산되면서 테이블 및 인덱스의 *크기*가 증가될 수 있습니다. 이로 인해, 쿼리 실행 동안 읽어야 하는 페이지 수가 늘어날 수 있습니다. 테이블 및 인덱스의 재구성은 데이터를 간략화하여 쓸모 없는 스페이스를 재개시키고 데이터 액세스를 향상시킵니다.

테이블 또는 인덱스 재구성 수행 단계는 다음과 같습니다.

1. 테이블 또는 인덱스를 재구성해야 하는지 여부를 판별하십시오.
2. 재구성 방법을 선택하십시오.
3. 식별된 오브젝트의 재구성을 수행하십시오.
4. 선택사항: 재구성의 진행을 모니터하십시오.
5. 재구성에 성공했는지 여부를 판별하십시오. 오프라인 테이블 재구성 및 인덱스 재구성의 경우, 조작이 동기적이고, 조작 완료 시 결과가 분명합니다. 온라인 테이블 재구성의 경우, 조작이 비동기적이고, 실행기록 파일에서 세부사항을 사용할 수 있습니다.
6. 재구성된 오브젝트에 대한 통계를 수집하십시오.
7. 재구성된 오브젝트에 액세스하는 응용프로그램을 리바인드하십시오.

테이블 재구성

테이블 데이터를 여러 번 변경한 후 논리적 순차 데이터가 비순차 데이터 페이지에 상주할 수 있으므로 데이터베이스 관리 프로그램은 추가 읽기 조작을 수행하여 데이터에 액세스해야 합니다.

추가 읽기 조작은 여러 행이 삭제된 경우에도 필요합니다. 이 경우 인덱스를 일치시키고 스페이스를 재요청하도록 테이블을 재구성할 수 있습니다. 또한 시스템 카탈로그 테이블을 재구성할 수 있습니다.

테이블 재구성은 보통 통계 갱신보다 오래 걸리므로 RUNSTATS 명령을 실행하여 데이터의 현재 통계를 새로 고친 후 응용프로그램을 리바인드할 수 있습니다. 통계를 새로 고쳐도 성능이 향상되지 않는 경우 재구성이 도움이 될 수 있습니다.

다음 요인은 테이블 재구성이 필요함을 나타낼 수 있습니다.

- 쿼리에서 액세스하는 테이블에 대해 대량의 삽입, 갱신 및 삭제 활동이 있었습니다.
- 클러스터 비율이 높은 인덱스를 사용하는 쿼리의 성능이 상당히 많이 변경되었습니다.
- RUNSTATS 명령을 실행하여 테이블 통계를 새로 고쳐도 성능이 향상되지 않습니다.
- REORGCHK 명령의 출력이 테이블 재구성이 필요함을 나타냅니다.

테이블 재구성 방법 선택

테이블 재구성에는 *기본 재구성*(오프라인)과 *인플레이스 재구성*(온라인)의 두 가지 방법이 있습니다.

오프라인 재구성이 디폴트 동작입니다. 온라인 재구성 조작을 지정하려면 REORG TABLE 명령에서 INPLACE 옵션을 사용하십시오.

온라인 테이블 이동 스토어드 프로시저를 사용하여 인플레이스 재구성을 대체하는 접근 방법도 가능합니다. 『ADMIN_MOVE_TABLE 프로시저를 사용하여 온라인으로 테이블 이동』을 참조하십시오.

각 방법에는 아래에 요약된 장점 및 단점이 있습니다. 재구성 방법을 선택할 때 어떤 방법이 우선순위에 부합되는 장점을 제공하는지 고려하십시오. 예를 들어, 실패할 경우의 복구 가능성이 성능보다 더 중요한 경우, 온라인 재구성을 선택하십시오.

오프라인 재구성의 장점

이 방법의 장점은 다음과 같습니다.

- 특히 대형 오브젝트(LOB) 또는 Long 필드 데이터가 포함되지 않을 경우 가장 빠른 테이블 재구성 조작
- 완료 시 완벽하게 클러스터된 테이블 및 인덱스
- 테이블이 재구성된 후 자동으로 재빌드되는 인덱스. 인덱스 재빌드를 위한 별도의 단계가 없습니다.
- 쉐도우 사본을 빌드하기 위한 임시 테이블의 사용. 이는 대상 테이블 또는 인덱스를 포함하는 테이블 스페이스에 대한 스페이스 요구사항을 줄여줍니다.

- 데이터를 다시 클러스터링하는 데 클러스터링 인덱스 이외의 인덱스 사용

오프라인 재구성의 단점

이 방법의 단점은 다음과 같습니다.

- 제한된 테이블 액세스. Reorg 조작의 정렬 및 빌드 단계 동안 읽기 액세스만 가능합니다.
- 재구성되는 테이블의 쉐도우 사본을 위한 대형 스페이스 요구사항
- Reorg 프로세스에 대한 더 낮은 제어. 오프라인 Reorg 조작은 일시정지 및 재시작할 수 없습니다.

온라인 재구성의 장점

이 방법의 장점은 다음과 같습니다.

- Reorg 조작의 절단 단계 동안을 제외하고 전체 테이블 액세스
- Reorg 프로세스에 대한 더 높은 제어. 백그라운드에서 비동기적으로 실행되며 일시정지, 재시작 또는 중지할 수 있습니다. 예를 들어, 테이블에 대해 많은 수의 갱신 또는 삭제 조작이 실행 중인 경우 진행 중인 Reorg 조작을 일시정지할 수 있습니다.
- 실패할 경우 복구 가능한 프로세스
- 테이블이 증분적으로 처리되기 때문에 작업 스토리지에 대한 요구사항이 줄어듬
- Reorg 조작이 완료되기 전이라도 재구성의 즉각적인 혜택

온라인 재구성의 단점

이 방법의 단점은 다음과 같습니다.

- Reorg 조작 동안 테이블에 액세스하는 트랜잭션의 유형에 따라 불완전한 데이터 또는 인덱스 클러스터링
- 오프라인 Reorg 조작보다 낮은 성능
- 이동되는 행 수, 테이블에 정의된 인덱스 수 및 이러한 인덱스의 *크기*에 따라 잠재적으로 높은 로깅 요구사항
- 인덱스가 재빌드되지 않고 유지보수되기 때문에 후속 인덱스 재구성에 대한 잠재적인 필요

표 1. 온라인과 오프라인 재구성 비교

특성	오프라인 재구성	온라인 재구성
성능	빠름	느림
완료 시 데이터의 클러스터링 인수	좋음	완벽하게 클러스터되지 않음
동시성(테이블에 대한 액세스)	액세스 없음에서 읽기 전용까지의 범위	읽기 전용에서 전체 액세스까지의 범위

표 1. 온라인과 오프라인 재구성 비교 (계속)

특성	오프라인 재구성	온라인 재구성
데이터 스토리지 스페이스 요구사항	충분함	충분하지 않음
로깅 스토리지 스페이스 요구사항	충분하지 않음	충분할 수 있음
사용자 제어(프로세스 재시작, 일시정지 기능)	더 낮은 제어	더 높은 제어
복구 가능성	복구 불가능	복구 가능
인덱스 재빌드	완료	완료되지 않음
모든 유형의 테이블에 대해 지원됨	예	아니오
클러스터링 인덱스 이외의 인덱스 지정 기능	예	아니오
임시 테이블 스페이스의 사용	예	아니오

표 2. 온라인 및 오프라인 재구성에 대해 지원되는 테이블 유형

테이블 유형	오프라인 재구성 지원	온라인 재구성 지원
다차원 클러스터링 테이블(MDC)	예[1]	아니오
RCT(range-clustered table)	아니오[2]	아니오
추가 모드 테이블	아니오	아니오[3]
Long 필드 또는 대형 오브젝트(LOB) 포함 테이블	예[4]	아니오
시스템 카탈로그 테이블: SYSIBM.SYSDBAUTH, SYSIBM.SYSROUTINEAUTH, SYSIBM.SYSSEQUENCES, SYSIBM.SYSTABLES	예	아니오

참고:

1. 클러스터링은 MDC 블록 인덱스를 통해 자동으로 유지되므로, MDC 테이블의 재구성은 스페이스 재개에만 관련됩니다. 인덱스를 지정할 수 없습니다.
2. RCT의 범위 영역은 항상 클러스터된 상태로 유지됩니다.
3. 온라인 재구성은 추가 모드가 사용 불가능하게 된 후에 수행할 수 있습니다.
4. Long 필드 또는 대형 오브젝트(LOB) 데이터 재구성에는 상당한 시간이 걸릴 수 있고, 쿼리 성능을 향상시키지 않습니다. 이것은 스페이스 재개를 위해서만 수행되어야 합니다.

테이블 재구성의 진행 모니터링

현재 테이블 Reorg 조작의 진행에 대한 정보는 실행기록 파일에 기록됩니다. 실행기록 파일에는 각 재구성 이벤트에 대한 레코드가 포함됩니다. 이 파일을 보려면 재구성되는 테이블을 포함하는 데이터베이스에 대해 LIST HISTORY 명령을 실행하십시오.

테이블 스냅샷을 사용하여 테이블 Reorg 조작의 진행을 모니터할 수도 있습니다. 데이터베이스 시스템 모니터 테이블 스위치에 상관없이 테이블 재구성 모니터링 데이터가 기록됩니다.

오류가 발생할 경우, SQLCA 메시지가 실행기록 파일에 기록됩니다. 인플레이스 테이블 Reorg 조작의 경우, 상태가 PAUSED로 기록됩니다.

클래식(오프라인) 테이블 재구성

클래식 테이블 재구성에서는 재구성되는 테이블의 전체 사본을 빌드하는 쉐도우 복사 방법을 사용합니다.

기본 또는 오프라인 테이블 재구성 조작에는 4가지 단계가 있습니다.

1. 정렬 - 이 단계 동안에는, 인덱스가 REORG TABLE 명령에서 지정되었거나 클러스터링 인덱스가 테이블에서 정의된 경우, 해당 인덱스에 따라 테이블의 행이 첫 번째로 정렬됩니다. INDEXSCAN 옵션이 지정된 경우 테이블을 정렬하는 데 인덱스 스캔이 사용됩니다. 그렇지 않은 경우, 테이블 스캔 정렬이 사용됩니다. 이 단계는 클러스터링 테이블 Reorg 조작에만 적용됩니다. 스페이스 재개 Reorg 조작은 빌드 단계에서 시작됩니다.
2. 빌드 - 이 단계 동안에는 REORG TABLE 명령에서 지정된 임시 테이블 스페이스 또는 해당 테이블 스페이스에 재구성된 전체 테이블의 복사가 빌드됩니다.
3. 교체 - 이 단계 동안에는 원래 테이블 오브젝트가 임시 테이블 스페이스의 사본으로 교체되거나, 재구성되는 테이블의 테이블 스페이스 내에 새로 빌드된 오브젝트에 대한 포인터가 작성됩니다.
4. 모든 인덱스 교체 - 이 단계 동안에는 테이블에 정의된 모든 인덱스가 다시 작성됩니다.

스냅샷 모니터 또는 스냅샷 관리 뷰를 사용하여 테이블 Reorg 조작의 진행을 모니터하고 현재 단계를 식별할 수 있습니다.

잠금 조건은 온라인 모드에서보다 오프라인 모드에서 더 제한적입니다. 복사가 빌드되는 동안 테이블에 대한 읽기 액세스가 가능합니다. 그러나 원래 테이블이 재구성된 사본으로 교체될 때 또는 인덱스가 재빌드될 때에는 테이블에 대한 배타적 액세스가 필요합니다.

IX 테이블 스페이스 잠금은 전체 테이블 Reorg 프로세스 동안 필요합니다. 빌드 단계 동안, U 잠금이 획득되고 테이블에서 보유됩니다. U 잠금을 사용하면 잠금 소유자가 테이블에서 데이터를 갱신할 수 있습니다. 기타 다른 응용프로그램에서는 데이터를 갱신할 수 없지만 읽기 액세스가 허용됩니다. 교체 단계가 시작된 후 U 잠금이 Z 잠금으로 업그레이드됩니다. 이 단계 동안에는 다른 응용프로그램에서 데이터에 액세스할 수 없습니다. 이 잠금은 테이블 Reorg 조작이 완료될 때까지 보유됩니다.

오프라인 재구성 프로세스에 의해 많은 파일이 작성됩니다. 이러한 파일은 데이터베이스 디렉토리에 저장됩니다. 파일 이름에는 테이블 스페이스 및 오브젝트 ID가 앞에 붙습니다. 예를 들어, 0030002.ROR은 테이블 스페이스 ID가 3이고 테이블 ID가 2인 테이블 Reorg 조작에 대한 상태 파일입니다.

다음 목록은 오프라인 테이블 Reorg 조작 동안 시스템 관리 스페이스(SMS) 테이블 스페이스에서 작성된 임시 파일을 보여줍니다.

- .DTR - 데이터 쉐도우 사본 파일
- .LFR - Long 필드 파일
- .LAR - Long 필드 할당 파일
- .RLB - LOB 데이터 파일
- .RBA - LOB 할당 파일
- .BMR - 다차원 클러스터링(MDC) 테이블에 대한 블록 오브젝트 파일

인덱스 Reorg 조작 동안 다음 임시 파일이 작성됩니다.

- .IN1 - 쉐도우 사본 파일

다음 목록은 정렬 단계 동안 시스템 임시 테이블 스페이스에서 작성되는 임시 파일을 보여줍니다.

- .TDA - 데이터 파일
- .TIX - 인덱스 파일
- .TLF - Long 필드 파일
- .TLA - Long 필드 할당 파일
- .TLB - LOB 파일
- .TBA - LOB 할당 파일
- .TBM - 블록 오브젝트 파일

재구성 프로세스와 연관된 파일은 시스템에서 수동으로 제거해서는 안 됩니다.

오프라인 테이블 재구성:

오프라인 테이블 재구성은 테이블의 조각을 모으는 가장 빠른 방법입니다. 재구성은 테이블에 필요한 스페이스의 양을 감소시키고 데이터 액세스 및 쿼리 성능을 향상시킵니다.

재구성할 테이블에 대한 SYSADM, SYSCTRL, SYSMAINT, DBADM 또는 SQLADM 권한이나 CONTROL 특권이 있어야 합니다. 또한 테이블을 재구성하기 위한 데이터베이스 연결이 있어야 합니다.

재구성이 필요한 테이블을 식별한 후, 이러한 테이블에 대해, 그리고 이러한 테이블에 정의되어 있는 인덱스에 대해(선택적) REORG 유틸리티를 실행할 수 있습니다.

1. REORG TABLE 명령을 사용하여 테이블을 재구성하려면 간단히 테이블의 이름을 지정하십시오. 예를 들어, 다음과 같습니다.

   ```
   reorg table employee
   ```

특정 임시 테이블 스페이스를 사용하여 테이블을 재구성할 수 있습니다. 예를 들어, 다음과 같습니다.

```
reorg table employee use mytemp
```

특정 인덱스에 따라 테이블을 재구성하고 행이 재정렬되게 할 수 있습니다. 예를 들어, 다음과 같습니다.

```
reorg table employee index myindex
```

2. SQL CALL문을 사용하여 테이블을 재구성하려면 ADMIN_CMD 프로시저와 함께 REORG TABLE 명령을 지정하십시오. 예를 들어, 다음과 같습니다.

   ```
   call sysproc.admin_cmd ('reorg table employee')
   ```

3. 관리 API를 사용하여 테이블을 재구성하려면 db2Reorg API를 호출하십시오.

테이블을 재구성한 후, 해당 테이블에 대한 통계를 수집하여 옵티마이저가 쿼리 액세스 플랜을 평가하기 위한 가장 정확한 데이터를 지니도록 하십시오.

오프라인 테이블 재구성의 복구:

오프라인 테이블 재구성은 대체 단계가 시작되기까지는 전부가 아니면 무의 프로세스입니다. 정렬 또는 빌드 단계 동안 시스템이 파손될 경우, Reorg 조작이 롤백되고 응급 복구 동안 재실행되지 않습니다.

대체 단계가 시작된 후 시스템이 파손될 경우에는 모든 재구성 작업이 완료되고 원래 테이블이 더 이상 사용 가능하지 않을 수 있으므로 Reorg 조작이 완료되어야 합니다. 응급 복구 동안, 재구성된 테이블 오브젝트를 위한 임시 파일이 필요하지만 정렬에 사용되는 임시 테이블 스페이스는 필요하지 않습니다. 복구에서는 대체 단계가 처음부터 재시작되고, 사본 오브젝트의 모든 데이터가 복구에 필요합니다. 이 경우 시스템 관리 스페이스(SMS)와 데이터베이스 관리 스페이스(DMS) 테이블 스페이스 간에 차이가 있습니다. SMS의 재구성된 테이블 오브젝트는 한 오브젝트에서 다른 오브젝트로 복사되어야 하지만, DMS의 재구성된 테이블 오브젝트를 단순히 가리키기만 하고, 동일한 테이블 스페이스에서 재구성이 수행된 경우 원래 테이블이 삭제됩니다. 응급 복구 동안 인덱스가 재빌드되지 않지만 유효하지 않은 것으로 표시되고, 데이터베이스에서는 일반 규칙에 따라 인덱스가 재빌드되는 시기(데이터베이스 재시작 시 또는 첫 번째 인덱스 액세스 시)를 판별합니다.

인덱스 재빌드 단계 동안 파손이 발생할 경우, 새 테이블 오브젝트가 이미 존재하기 때문에 아무 것도 재실행되지 않습니다. 인덱스는 앞서 설명된 대로 처리됩니다.

롤 포워드 복구 동안에는, 이전 버전의 테이블이 디스크에 있는 경우 Reorg 조작이 재실행됩니다. 롤 포워드 유틸리티는 빌드 단계 동안 로그된 레코드 ID(RID)를 사용하여 빌드와 교체 단계를 반복하며 재구성된 테이블을 작성한 조작을 다시 적용합니다. 인덱스는 앞서 설명된 대로 처리됩니다. 원래 임시 테이블 스페이스가 사용된 경우에만 재

구성된 오브젝트의 사본을 위해 임시 테이블 스페이스가 필요합니다. 롤 포워드 복구 동안에는 여러 Reorg 조작이 동시에 재실행될 수 있습니다(병렬 복구).

오프라인 테이블 재구성의 성능 향상:

오프라인 테이블 재구성의 성능은 주로 데이터베이스 환경의 특성에 의해 판별됩니다.

ALLOW NO ACCESS 모드에서 실행 중인 Reorg 조작과 ALLOW READ ACCESS 모드에서 실행 중인 Reorg 조작 간에는 성능 차이가 거의 없습니다. 차이점은 ALLOW READ ACCESS 모드의 Reorg 조작 동안에는 유틸리티에서 다른 응용프로그램이 스캔을 완료하고 잠금을 해제하기를 대기한 후 테이블을 교체해야 할 수도 있다는 점입니다. 이러한 모드 중 하나에서 실행 중인 Reorg 조작의 인덱스 재빌드 단계 동안에는 테이블을 사용할 수 없습니다.

성능 개선에 대한 추가 정보

- 스페이스가 충분한 경우, 임시 테이블 스페이스를 사용하는 대신 원래 테이블과 재구성된 테이블 사본 둘 다에 대해 동일한 테이블 스페이스를 사용하십시오. 이렇게 하면 임시 테이블 스페이스에서 재구성된 테이블을 복사하는 데 필요한 시간이 절약됩니다.
- Reorg 조작 동안 더 적은 인덱스를 유지해야 하도록 테이블 재구성 전에 불필요한 인덱스의 삭제를 고려하십시오.
- 재구성된 테이블이 상주하는 테이블 스페이스의 프리페치 *크기*가 적절히 설정되었는지 확인하십시오.
- **sortheap** 및 **sheapthres** 데이터베이스 구성 매개변수를 조정하여 정렬에 사용 가능한 스페이스를 제어하십시오. 각 프로세서에서 개별 정렬을 실행하므로, **sheapthres**의 값이 최소한 **sortheap** x *프로세서의 수*여야 합니다.
- 버퍼 풀의 더티 인덱스 페이지가 가능한 한 빨리 정리되도록 페이지 클리너의 수를 조정하십시오.

인플레이스(온라인) 테이블 재구성

인플레이스 테이블 재구성을 사용하면 데이터에 대한 전체 액세스를 가지면서 테이블을 재구성할 수 있습니다. 이렇게 데이터에 중단 없이 액세스할 수 있는 대신 테이블 Reorg 조작이 더 느려집니다.

인플레이스 또는 온라인 Reorg 조작에서 테이블의 분할 영역은 순차적으로 재구성됩니다. 데이터가 임시 테이블 스페이스에 복사되지 않고, 대신 기존 테이블 오브젝트 내에서 행이 이동되어 클러스터링을 재구성하고, 여유 공간을 재개하며 오버플로우 행을 제거합니다.

온라인 테이블 Reorg 조작에는 4가지 주요 단계가 있습니다.

1. *n*페이지 선택

이 단계 동안에는 데이터베이스 관리 프로그램이 *n*페이지의 범위를 선택합니다. 여기서 *n*은 Reog 처리를 위한 최소 32 순차 페이지의 범위 크기입니다.

2. 범위 비우기

   REORG 유틸리티가 이 범위 내의 모든 행을 테이블의 사용 가능한 페이지로 이동시킵니다. 이동된 각 행은 행의 새 위치에 대한 레코드 ID(RID)를 포함하는 Reorg 테이블 포인터(RP) 레코드를 남깁니다. 행은 테이블의 사용 가능한 페이지에 데이터를 포함하는 Reorg 테이블 오버플로우(RO) 레코드로 배치됩니다. 유틸리티는 행 세트의 이동을 완료한 후 테이블의 데이터에 액세스 중인 모든 응용프로그램이 완료될 때까지 대기합니다. 이러한 『이전 스캐너』에서는 테이블에 액세스할 때 이전 RID를 사용합니다. 이 대기 기간 동안 시작된 모든 테이블 액세스(『새 스캐너』)에서는 새 RID를 사용하여 데이터에 액세스합니다. 이전 스캐너가 모두 완료된 후, REORG 유틸리티는 RP 레코드를 삭제하고 RO 레코드를 일반 레코드로 변환하여 이동된 행을 정리합니다.

3. 범위 채우기

   특정 범위의 모든 행이 비워진 후, 해당 행은 재구성된 형식으로 다시 작성되고, 사용된 인덱스에 따라 정렬되며, 정의된 PCTFREE 제한사항을 준수합니다. 범위의 모든 페이지가 다시 작성되면 테이블의 *n* 순차 페이지가 선택되고 프로세스가 반복됩니다.

4. 테이블 절단

   디폴트로, 테이블의 모든 페이지가 재구성된 경우 테이블이 재개 스페이스로 절단됩니다. NOTRUNCATE 옵션이 지정된 경우, 재구성된 테이블이 절단되지 않습니다.

온라인 테이블 Reorg 조작 동안 작성된 파일

온라인 테이블 Reorg 조작 동안 각 데이터베이스 파티션에 대해 `.0LR` 상태 파일이 작성됩니다. 이 2진 파일은 `xxxxyyyy.0LR` 형식의 이름을 갖습니다. 여기서 *xxxx*는 테이블 스페이스 ID이고 *yyyy*는 16진수 형식의 오브젝트 ID입니다. 이 파일은 일시정지된 상태에서 온라인 Reorg 조작을 재개하는 데 필요한 다음 정보를 포함합니다.

- Reorg 조작의 유형
- 재구성되는 테이블의 수명 시퀀스 번호(LSN)
- 비워질 다음 범위
- Reorg 조작에서 데이터를 클러스터할지 또는 스페이스를 재개하기만 할지 여부
- 데이터를 클러스터하는 데 사용되는 인덱스의 ID

`.0LR` 파일에서 체크섬이 수행됩니다. 파일이 손상되어 체크섬 오류를 일으키는 경우나 테이블 LSN이 수명 LSN과 일치하지 않는 경우, 새 Reorg 조작이 시작되고 새 상태 파일이 작성됩니다.

.OLR 상태 파일이 삭제될 경우 Reorg 프로세스를 재개할 수 없고, SQL2219N이 리턴되며, 새 Reorg 조작을 시작해야 합니다.

재구성 프로세스와 연관된 파일은 시스템에서 수동으로 제거해서는 안 됩니다.

온라인 테이블 재구성:

온라인 또는 인플레이스 테이블 재구성은 테이블이 재구성되는 동안 사용자가 테이블에 액세스할 수 있게 해 줍니다.

재구성할 테이블에 대한 SYSADM, SYSCTRL, SYSMAINT, DBADM 또는 SQLADM 권한이나 CONTROL 특권이 있어야 합니다. 또한 테이블을 재구성하기 위한 데이터베이스 연결이 있어야 합니다.

재구성이 필요한 테이블을 식별한 후, 이러한 테이블에 대해, 그리고 이러한 테이블에 정의되어 있는 인덱스에 대해(선택적) REORG 유틸리티를 실행할 수 있습니다.

1. REORG TABLE 명령을 사용하여 온라인으로 테이블을 재구성하려면 간단히 테이블의 이름과 INPLACE 옵션을 지정하십시오. 예를 들어, 다음과 같습니다.

   ```
   reorg table employee inplace
   ```

2. SQL CALL문을 사용하여 온라인으로 테이블을 재구성하려면 ADMIN_CMD 프로시저와 함께 REORG TABLE 명령을 지정하십시오. 예를 들어, 다음과 같습니다.

   ```
   call sysproc.admin_cmd ('reorg table employee inplace')
   ```

3. 관리 API를 사용하여 온라인으로 테이블을 재구성하려면 db2Reorg API를 호출하십시오.

테이블을 재구성한 후, 해당 테이블에 대한 통계를 수집하여 옵티마이저가 쿼리 액세스 플랜을 평가하기 위한 가장 정확한 데이터를 지니도록 하십시오.

온라인 테이블 재구성의 복구:

온라인 테이블 재구성은 종종 디스크 가득참 또는 로깅 오류로 인해 실패합니다. 온라인 테이블 재구성이 실패할 경우 SQLCA 메시지가 실행기록 파일에 기록됩니다.

런타임 동안 실패가 발생할 경우, 온라인 테이블 Reorg 조작이 일시정지된 후 응급 복구 동안 롤백됩니다. 그 후에 REORG TABLE 명령에서 RESUME 옵션을 지정하여 Reorg 조작을 재개할 수 있습니다. 프로세스가 완전히 로그되므로, 온라인 테이블 재구성 조장의 복구 가능성이 보장됩니다.

일부 환경에서는, 온라인 테이블 Reorg 조작이 **num_log_span** 데이터베이스 구성 매개변수의 값에 의해 설정된 한계를 초과할 수도 있습니다. 이 경우, 데이터베이스 관리 프로그램에서 REORG 유틸리티를 강제 실행하고 PAUSE 상태로 놓습니다. 스냅샷 모니터 출력에서, REORG 유틸리티의 상태가 PAUSED로 표시됩니다.

온라인 테이블 Reorg 일시정지는 인터럽트 지향입니다. 즉, 특정 상황(예: 시스템 파손의 경우)에서 데이터베이스 관리 프로그램에 의해서나 사용자(FORCE APPLICATION 명령 또는 REORG TABLE 명령에서 PAUSE 옵션 사용)에 의해 트리거될 수 있습니다.

파티션된 데이터베이스 환경의 하나 이상의 데이터베이스 파티션에서 오류가 발생할 경우, 리턴되는 SQLCODE는 오류를 보고하는 첫 번째 데이터베이스 파티션의 SQLCODE입니다.

온라인 테이블 재구성 일시정지 및 재시작:

사용자는 진행 중인 온라인 테이블 재구성을 일시정지했다가 재시작할 수 있습니다.

온라인 재구성을 일시정지하거나 재시작할 테이블에 대한 SYSADM, SYSCTRL, SYSMAINT, DBADM 또는 SQLADM 권한이나 CONTROL 특권이 있어야 합니다. 또한 온라인 테이블 재구성을 일시정지하거나 재시작하기 위한 데이터베이스 연결이 있어야 합니다.

1. REORG TABLE 명령을 사용하여 온라인 테이블 재구성을 일시정지하려면 테이블의 이름, INPLACE 옵션 및 PAUSE 옵션을 지정하십시오. 예를 들어, 다음과 같습니다.

   ```
   reorg table employee inplace pause
   ```

2. 일시정지된 온라인 테이블 재구성을 재시작하려면 RESUME 옵션을 지정하십시오. 예를 들어, 다음과 같습니다.

   ```
   reorg table employee inplace resume
   ```

   온라인 테이블 재구성이 일시정지된 경우, 해당 테이블의 새로운 재구성을 시작할 수 없습니다. 새 재구성 프로세스를 시작하려면 먼저 일시정지된 조작을 재시작하거나 중지해야 합니다.

   RESUME 요청에 따라, 재구성 프로세스는 현재 RESUME 요청에 어떤 절단 옵션이 지정되어 있는지 준수합니다. 예를 들어, 현재 RESUME 요청에 NONTRUNCATE 옵션이 지정되어 있지 않은 경우, 원래 REORG TABLE 명령에서(또는 이전 RESUME 요청을 통해) 지정된 NOTRUNCATE 옵션이 무시됩니다.

   테이블 Reorg 조작은 리스토어 및 롤 포워드 조작 후에 재시작될 수 없습니다.

온라인 테이블 재구성에 대한 잠금 및 동시성 고려사항:

온라인 테이블 재구성의 가장 중요한 측면 중 하나는 잠금이 제어되는 방식입니다 (응용프로그램 동시성에 매우 중요하기 때문).

온라인 테이블 Reorg 조작에서는 다음 잠금을 보유할 수 있습니다.

- 테이블 스페이스에 대한 쓰기 액세스를 보장하기 위해 Reorg 조작의 영향을 받는 테이블 스페이스에서 IX 잠금이 획득됩니다.
- 테이블 잠금은 전체 Reorg 조작 동안 획득되고 보유됩니다. 잠금의 레벨은 재구성 동안 적용되는 액세스 모드에 따라 달라집니다.
  - ALLOW WRITE ACCESS가 지정된 경우, IS 테이블 잠금이 획득됩니다.
  - ALLOW READ ACCESS가 지정된 경우, S 테이블 잠금이 획득됩니다.
- 절단 단계 동안 테이블에 대한 S 잠금이 요청됩니다. S 잠금이 획득될 때까지 동시 트랜잭션을 통해 행이 삽입될 수 있습니다. 이렇게 삽입된 행은 REORG 유틸리티에서 볼 수 없을 수도 있고, 테이블이 절단되지 못하게 할 수 있습니다. S 테이블 잠금이 획득된 후에는, 테이블이 절단되지 못하게 하는 행이 이동되어 테이블이 압축됩니다. 압축된 후 테이블이 잘리지만, 절단 지점이 결정될 때 테이블에 액세스 중인 모든 트랜잭션이 완료된 후에만 잘립니다.
- 행 잠금은 테이블 잠금의 유형에 따라 획득될 수 있습니다.
  - 테이블에서 S 잠금이 보유된 경우, 개별 행 레벨 S 잠금이 필요하지 않고 추가 잠금이 불필요합니다.
  - 테이블에서 IS 잠금이 보유된 경우, 행이 이동되기 전에 NS 행 잠금이 획득되고 이동이 완료된 후에 해제됩니다.
- 온라인 테이블 Reorg 조작 동안 특정 내부 잠금이 획득될 수도 있습니다.

잠금은 온라인 테이블 Reorg 조작과 동시 사용자 응용프로그램 둘 다의 성능에 영향을 미칩니다. 잠금 스냅샷 데이터를 사용하면 온라인 테이블 재구성 동안 발생하는 잠금 활동을 이해하는 데 도움이 됩니다.

테이블 재구성 모니터링

GET SNAPSHOT 명령, SNAPTAB_REORG 관리 뷰 또는 SNAP_GET_TAB_REORG 테이블 기능을 사용하여 테이블 재구성 조작의 상태에 대한 정보를 얻을 수 있습니다.

- SQL을 사용하여 재구성 조작에 대한 정보에 액세스하려면 SNAPTAB_REORG 관리 뷰를 사용하십시오. 예를 들어, 다음 쿼리는 현재 연결된 데이터베이스에 대한 모든 데이터베이스 파티션의 테이블 재구성 조작에 대한 세부사항을 리턴합니다. 테이블이 재구성되지 않은 경우, 행이 리턴되지 않습니다.

```
select
    substr(tabname, 1, 15) as tab_name,
    substr(tabschema, 1, 15) as tab_schema,
    reorg_phase,
    substr(reorg_type, 1, 20) as reorg_type,
    reorg_status,
    reorg_completion,
    dbpartitionnum
  from sysibmadm.snaptab_reorg
  order by dbpartitionnum
```

- 스냅샷 모니터를 사용하여 재구성 조작에 대한 정보에 액세스하려면 GET SNAPSHOT FOR TABLES 명령을 사용하고 테이블 재구성 모니터 요소의 값을 조사하십시오.

오프라인 테이블 Reorg 조작은 동기적이기 때문에, 유틸리티(응용프로그램 또는 명령행 처리기)의 호출자에 오류가 리턴됩니다. 그리고 온라인 테이블 Reorg 조작은 비동기적이기 때문에, 이 경우 오류 메시지가 CLP에 리턴되지 않습니다. 온라인 테이블 Reorg 조작 동안 리턴되는 SQL 오류 메시지를 보려면 LIST HISTORY REORG 명령을 사용하십시오.

온라인 테이블 Reorg 조작은 db2Reorg 프로세스로 백그라운드에서 실행됩니다. 이 프로세스는 호출 응용프로그램이 해당 데이터베이스 연결을 종료하더라도 계속 실행됩니다.

인덱스 재구성

테이블이 갱신되면 인덱스 성능이 저하될 수 있습니다.

다음과 같은 면에서 저하가 발생할 수 있습니다.

- 리프 페이지가 분할됩니다. 리프 페이지가 분할되면 테이블 페이지를 페치하기 위해 추가 리프 페이지를 읽어야 하므로 입출력 비용이 증가합니다.
- 실제 인덱스 페이지 순서는 더 이상 이러한 페이지의 키 시퀀스와 일치하지 않으므로 *잘못 클러스터된 인덱스*가 나타납니다. 리프 페이지가 잘못 클러스터된 경우 순차 프리페치는 비효율적이며 입출력 대기 수가 증가합니다.
- 인덱스가 너무 많은 레벨로 개발됩니다. 이 경우 인덱스를 재구성해야 합니다.

인덱스를 재구성하려면 다음이 필요합니다.

- SYSADM, SYSMAINT, SYSCTRL, DBADM 또는 SQLADM 권한이나 테이블 및 해당 인덱스에 대한 CONTROL 특권
- 인덱스가 저장된 테이블 스페이스에서 인덱스의 현재 *크기*와 동일한 여유 공간의 양. CREATE TABLE문을 발행할 때 대형 테이블 스페이스에 인덱스를 배치하십시오.
- 추가 로그 스페이스. 인덱스 REORG 유틸리티는 활동을 로그합니다.

CREATE INDEX문에 MINPCTUSED 옵션을 지정하는 경우 키가 삭제되고 여유 공간이 지정된 값보다 작아지면 데이터베이스 서버는 인덱스 리프 페이지를 자동으로 병합합니다. 이 프로세스를 *온라인 인덱스 조각 모음*이라고 합니다.

인덱스 클러스터링을 리스토어하고 스페이스를 사용 가능하게 하고 리프 레벨을 줄이려면 다음 중 한 방법을 사용할 수 있습니다.

- 인덱스를 삭제(drop)하고 재작성하십시오.
- 테이블 및 인덱스를 모두 오프라인으로 재구성할 수 있도록 지원하는 옵션과 함께 REORG TABLE 명령을 사용하십시오.

- REORG INDEXES 명령을 사용하여 인덱스를 온라인으로 재구성하십시오. 프로덕션 환경에서 이 방법을 선택할 수 있습니다. 이 방법을 사용하면 인덱스를 재빌드하는 중 사용자가 테이블에서 읽고 테이블에 쓸 수 있습니다.

온라인 인덱스 재구성

REORG INDEXES 명령을 ALLOW WRITE ACCESS 옵션과 함께 사용하는 경우 테이블에 대한 읽기 및 쓰기 액세스가 계속되는 동안 지정된 테이블의 모든 인덱스가 재빌드됩니다. 재구성 중 인덱스에 영향을 주는 기본 테이블의 변경사항이 로그됩니다. 인덱스를 재빌드하는 동안 reorg 조작이 이러한 로그된 변경사항을 처리합니다.

인덱스에 영향을 주는 기본 테이블의 변경사항은 내부 메모리 버퍼에도 기록됩니다(이러한 스페이스가 사용 가능한 경우). 내부 버퍼는 유틸리티 힙에서 요구에 따라 할당되는 지정된 메모리 영역입니다. 메모리 버퍼 스페이스를 사용하면 인덱스 reorg 유틸리티는 먼저 메모리에서 직접 읽은 후 필요하면 훨씬 나중에 로그를 읽어 변경사항을 처리할 수 있습니다. 할당된 메모리는 reorg 조작이 완료된 후에 사용 가능해집니다.

ALLOW WRITE ACCESS 모드(CLEANUP 옵션이 없는)의 온라인 인덱스 재구성은 공간 인덱스, 파티션된 테이블의 파티션된 인덱스 또는 다차원 클러스터링(MDC) 테이블의 경우 지원되지 않습니다.

테이블 및 인덱스 재구성 시기 판별

테이블 데이터에 대한 많은 변경 후에, 특히 많은 갱신 조작이 오버플로우 레코드를 생성한 경우 논리적 순차 데이터가 비연속 실제 데이터 페이지에 배치될 수도 있습니다. 데이터가 이런 식으로 구성되면, 데이터베이스 관리 프로그램에서 추가 읽기 조작을 수행하여 필요한 데이터에 액세스해야 합니다. 추가 읽기 조작은 여러 행이 삭제된 경우에도 필요합니다.

테이블 재구성은 쓸모 없는 스페이스를 제거하며 데이터를 조각을 모읍니다. 또한 행을 재정렬하여 오버플로우 레코드를 통합해 데이터 액세스와 궁극적으로 쿼리 성능을 향상시킵니다. 쿼리가 최소 읽기 조작 수를 통해 데이터에 액세스할 수 있게 인덱스에 따라 데이터를 재정렬해야 하도록 지정할 수 있습니다.

테이블 데이터에 대한 많은 변경은 인덱스 성능을 저하시킬 수 있습니다. 인덱스 리프 페이지가 분할되거나 심하게 클러스터될 수 있고, 인덱스가 최적 성능에 필요한 것보다 더 많은 레벨을 전개할 수 있습니다. 이러한 문제는 모두 더 많은 입출력을 야기하여 성능을 저하시킬 수 있습니다.

다음 인수는 모두 테이블 또는 인덱스를 재구성해야 함을 표시할 수도 있습니다.

- 테이블이 마지막으로 재구성된 이후 테이블에 대한 높은 볼륨의 삽입, 갱신 및 삭제 활동
- 높은 클러스터 비율과 함께 인덱스를 사용하는 쿼리 성능의 큰 변화

- RUNSTATS 명령을 실행하여 통계 정보를 새로 고쳐도 성능이 향상되지 않음
- REORGCHK 명령 출력은 테이블 또는 인덱스를 재구성하여 성능을 향상시킬 수 있음을 제안합니다.

  일부 경우에는, 테이블 Reorg 조작이 수행된 후라도 Reorgchk 유틸리티에서 테이블 재구성을 항상 권장합니다. 예를 들어, 평균 레코드 길이가 15바이트이고 페이지당 최대 253개 레코드를 지닌 32KB 페이지를 사용한다는 것은 각 페이지에 32700 - (15 x 253) = 28905의 사용할 수 없는 바이트가 있음을 의미합니다. 즉, 페이지의 약 88%가 여유 공간입니다. reorgchk 유틸리티 권장사항을 분석하고 재구성 수행 비용에 대한 잠재적인 이점을 평가해야 합니다.

REORGCHK 명령은 데이터 구성에 대한 통계 정보를 리턴하고 특정 테이블 또는 인덱스가 재구성되어야 하는지 여부에 대해 조언할 수 있습니다. 그러나 SYSSTAT 뷰에 대해 정기적으로나 특정 시간에 특정 쿼리를 실행하면 잠재적으로 상당한 성능 관련 추세를 식별하는 데 도움이 되는 실행기록을 빌드할 수 있습니다.

테이블 또는 인덱스를 재구성할 필요가 있는지 여부를 판별하려면, SYSSTAT 뷰를 쿼리하고 다음 통계를 모니터하십시오.

- 행의 오버플로우

  SYSSTAT.TABLES 뷰에서 OVERFLOW 컬럼을 쿼리하여 오버플로우 값을 모니터하십시오. 이 값은 원래 페이지에 맞지 않는 행의 수를 나타냅니다. 가변 길이 컬럼으로 인해 레코드 길이가 행이 데이터 페이지에서 지정된 위치에 더 이상 맞지 않는 지점까지 확장될 경우 행 데이터가 오버플로우될 수 있습니다. 컬럼이 테이블에 추가될 경우에도 길이 변경이 발생할 수 있습니다. 이 경우, 포인터가 행의 원래 위치에서 유지되고, 포인터에 의해 표시된 다른 위치에 실제 값이 저장됩니다. 데이터베이스 관리 프로그램이 포인터를 따라 컬럼의 컨텐츠를 찾아야 하기 때문에 이는 성능에 영향을 미칠 수 있습니다. 이 2단계 프로세스는 처리 시간을 증가시키고 필요한 입출력의 수를 증가시킬 수도 있습니다. 테이블 데이터를 재구성하면 행 오버플로우가 제거됩니다.
- 페치 통계

  SYSSTAT.INDEXES 카탈로그 뷰에서 다음 컬럼을 쿼리하여 테이블이 인덱스 순서로 액세스될 때 프리페처의 효과를 판별하십시오. 이러한 통계는 기본 테이블에 대한 프리페처의 평균 성능의 특성을 기술합니다.

  – AVERAGE_SEQUENCE_FETCH_PAGES 컬럼은 차례로 액세스될 수 있는 페이지의 평균 수를 저장합니다. 차례로 액세스될 수 있는 페이지는 프리페치 자격이 있습니다. 작은 수는 프리페처가 테이블 스페이스에 대한 PREFETCHSIZE 값에 의해 지정된 전체 페이지 수를 읽을 수 없기 때문에 가능한 만큼 효과적이지

않음을 표시합니다. 큰 수는 프리페처가 효과적으로 수행되고 있음을 표시합니다. 클러스터된 인덱스 및 테이블의 경우, 이 수는 행을 포함하는 페이지의 수인 NPAGES의 값에 가까워야 합니다.

- AVERAGE_RANDOM_FETCH_PAGES 컬럼은 인덱스를 사용하여 테이블을 페치할 때 순차 페이지 액세스 간에 페치되는 무작위 테이블 페이지의 평균 수를 저장합니다. 대부분의 페이지가 차례로 되어 있는 경우 프리페처는 작은 수의 무작위 페이지를 무시하고, 구성된 프리페치 크기로 계속 프리페치합니다. 테이블이 더 혼란하게 되면서 무작위 페치 페이지의 수가 증가합니다. 혼란은 보통 테이블의 끝이나 오버플로우 페이지에서 순서를 벗어나서 발생하는 삽입에 의해 야기되고, 값 범위에 액세스하는 데 인덱스가 사용될 때 쿼리 성능이 영향을 받습니다.
- AVERAGE_SEQUENCE_FETCH_GAP 컬럼은 인덱스를 사용하여 테이블 행을 페치할 때 테이블 페이지 시퀀스 간의 평균 갭을 저장합니다. 인덱스 리프 페이지의 스캔을 통해 발견된 각 갭은 테이블 페이지의 시퀀스 간에 무작위로 페치되어야 하는 테이블 페이지의 평균 수를 나타냅니다. 이것은 많은 페이지가 무작위로 액세스될 때 발생하며, 이는 프리페처를 방해합니다. 큰 수는 테이블이 혼란되었거나 인덱스에 제대로 클러스터되지 않았음을 표시합니다.

- 삭제된 것으로 표시되었지만 아직 제거되지 않은 레코드 ID(RID)를 포함하는 인덱스 리프 페이지의 수

  RID는 삭제된 것으로 표시된 경우 일반적으로 물리적으로는 삭제되지 않습니다. 즉, 이러한 논리적으로 삭제된 RID가 유용한 스페이스를 차지하고 있을 수도 있습니다. 모든 RID가 삭제된 것으로 표시된 리프 페이지의 수를 검색하려면 SYSSTAT.INDEXES 뷰의 NUM_EMPTY_LEAFS 컬럼을 쿼리하십시오. 일부 RID가 삭제된 것으로 표시된 리프 페이지의 경우, 논리적으로 삭제된 RID의 총 수가 NUMRIDS_DELETED 컬럼에 저장됩니다.

  이 정보를 사용하여 CLEANUP ALL 옵션과 함께 REORG INDEXES 명령을 호출함으로써 얼마나 많은 스페이스가 재개될 수 있는지 추정하십시오. 모든 RID가 삭제된 것으로 표시된 페이지의 스페이스만 재개하려면 CLEANUP ONLY PAGES 옵션과 함께 REORG INDEXES 명령을 호출하십시오.

- 인덱스에 대한 클러스터-비율 및 클러스터-인수 통계

  일반적으로, 테이블에 대한 인덱스 중 하나만 높은 클러스터링 등급을 지닐 수 있습니다. 클러스터-비율 통계는 SYSCAT.INDEXES 카탈로그 뷰의 CLUSTERRATIO 컬럼에 저장됩니다. 0에서 100 사이인 이 값은 인덱스에서 데이터 클러스터링의 등급을 나타냅니다. 자세한 인덱스 통계를 수집할 경우, 0과 1 사이의 더 미세한 클러스터-인수 통계가 CLUSTERFACTOR 컬럼에 대신 저장되고, CLUSTERRATIO의 값은 -1입니다. 이러한 두 클러스터링 통계 중 하나만 SYSCAT.INDEXES 카탈로그 뷰에 기록될 수 있습니다. CLUSTERFACTOR 값을 CLUSTERRATIO 값과 비교하려면 CLUSTERFACTOR 값을 100으로 곱해 퍼센트 값을 얻으십시오.

인덱스만 액세스가 아닌 인덱스 스캔은 더 높은 클러스터 비율을 사용하여 더 잘 수행될 수 있습니다. 데이터 페이지가 다시 액세스될 때까지 버퍼 풀에 남아 있을 가능성이 적기 때문에, 낮은 클러스터 비율은 이런 유형의 스캔에 대해 더 많은 입출력을 초래합니다. 버퍼 크기를 늘리면 덜 클러스터된 인덱스의 성능이 향상될 수도 있습니다.

테이블 데이터가 처음에 특정 인덱스에서 클러스터되고, 클러스터링 통계에서 동일한 인덱스에서 데이터가 현재 제대로 클러스터되지 않은 것으로 표시될 경우, 테이블을 재구성하여 데이터를 재클러스터할 수도 있습니다.

- 리프 페이지의 수

  SYSCAT.INDEXES 뷰에서 NLEAF 컬럼을 쿼리하여 인덱스가 차지하는 리프 페이지의 수를 판별하십시오. 이 수는 전체 인덱스 스캔에 얼마나 많은 인덱스 페이지 입출력이 필요한지 알려줍니다.

  원칙적으로, 인덱스는 인덱스 스캔에 필요한 입출력의 수를 줄이기 위해 가능한 한 작은 스페이스를 차지해야 합니다. 무작위 갱신 활동은 인덱스의 크기를 증가시키는 페이지 분할을 야기할 수 있습니다. 테이블 Reorg 조작 동안, 각 인덱스는 최소량의 스페이스를 사용하여 재빌드될 수 있습니다.

  디폴트로, 인덱스가 빌드될 때 각 인덱스 페이지에 10퍼센트의 여유 공간이 남겨집니다. 여유 공간 양을 늘리려면 인덱스를 작성할 때 PCTFREE 옵션을 지정하십시오. 인덱스를 재구성할 때마다 지정된 PCTFREE 값이 사용됩니다. 10퍼센트보다 큰 여유 공간 값은 여분의 공간이 추가 인덱스 삽입을 수용할 수 있기 때문에 인덱스 재구성의 빈도를 줄일 수도 있습니다.

- 비어 있는 데이터 페이지의 수

  테이블에서 비어 있는 페이지의 수를 계산하려면 SYSCAT.TABLES 뷰에서 FPAGES 및 NPAGES 컬럼을 쿼리한 다음 FPAGES 값(사용 중인 페이지의 총 수)에서 NPAGES 값(행을 포함하는 페이지의 수)을 빼십시오. 전체 행 범위가 삭제될 때 비어 있는 페이지가 발생할 수 있습니다.

  비어 있는 페이지의 수가 증가함에 따라 테이블 재구성에 대한 필요도 증가합니다. 테이블을 재구성하면 비어 있는 페이지가 재개되고 테이블에서 사용하는 스페이스의 양이 줄어듭니다. 또한 테이블 스캔 동안 비어 있는 페이지가 버퍼 풀로 읽히기 때문에, 사용하지 않는 페이지를 재개하면 스캔 성능이 향상될 수 있습니다.

  테이블에서 사용 중 페이지의 총 수(FPAGES)가 (NPARTITIONS * 1 Extent 크기) 이하인 경우, 테이블 재구성이 권장되지 않습니다. NPARTITIONS는 테이블이 파티션된 테이블인 경우 데이터 파티션의 수를 나타냅니다. 그렇지 않은 경우, 해당 값은 1입니다. 파티션된 데이터베이스 환경에서, FPAGES <= (테이블의 데이터베이

스 파티션 그룹에서 데이터베이스 파티션의 수) * (NPARTITIONS * 1 Extent 크기)인 경우 테이블 재구성이 권장되지 않습니다.

테이블 또는 인덱스를 재구성하기 전에, 점점 저하되는 쿼리 성능 비용과 테이블 또는 인덱스 재구성 비용 간의 교환 조건을 고려하십시오. 여기에는 처리 시간, 경과 시간 및 감소된 동시성이 포함됩니다.

테이블 및 인덱스 재구성 비용

테이블 또는 인덱스 재구성을 수행하면 오브젝트를 재구성할지 여부를 결정할 때 고려해야 하는 일정량의 오버헤드가 발생합니다.

테이블 및 인덱스 재구성 비용에는 다음이 포함됩니다.

- 실행 유틸리티의 처리 시간
- REORG 유틸리티를 실행하는 동안 잠금 때문에 감소된 동시성
- 추가 스토리지 요구사항
  - 오프라인 테이블 재구성에서는 테이블의 쉐도우 사본을 보유하기 위해 더 많은 스토리지 스페이스가 필요합니다.
  - 온라인 또는 인플레이스 테이블 재구성에서는 더 많은 로그 스페이스가 필요합니다.
  - 오프라인 인덱스 재구성에서는 더 적은 로그 스페이스가 필요하고 쉐도우 사본이 포함되지 않습니다.
  - 온라인 인덱스 재구성에서는 인덱스의 쉐도우 사본을 보유하기 위해 더 많은 스토리지 스페이스와 더 많은 로그 스페이스가 필요합니다.

일부 경우, 재구성된 테이블이 원래 파일보다 더 클 수도 있습니다. 다음 경우에 재구성 후 테이블이 커질 수도 있습니다.

- 인덱스가 행의 순서를 결정하는 데 사용되는 클러스터링 reorg 테이블 조작에서는 재구성된 테이블의 일부 페이지에 원래 테이블보다 더 적은 행이 포함될 수 있으므로 테이블 레코드가 변수 길이에 대한 것일 경우 더 많은 스페이스가 필요할 수도 있습니다.
- 각 페이지에 남은 여유 공간의 양(PCTFREE 값으로 표시됨)은 마지막 재구성 이후로 증가했을 수 있습니다.

오프라인 테이블 재구성에 대한 스페이스 요구사항

오프라인 재구성에서는 쉐도우 사본 방법을 사용하기 때문에 테이블의 다른 사본을 수용하기 위한 충분한 추가 스토리지가 필요합니다. 쉐도우 사본은 원래 테이블이 상주하는 테이블 스페이스 또는 사용자 지정 임시 테이블 스페이스에서 빌드됩니다.

테이블 스캔 정렬이 사용될 경우 정렬 처리를 위해 추가 임시 테이블 스페이스 스토리지가 필요할 수도 있습니다. 필요한 추가 스페이스는 재구성되는 테이블의 크기만큼일 수 있습니다. 클러스터링 인덱스가 시스템 관리 스페이스(SMS) 유형이거나 고유 데이터베이스 관리 스페이스(DMS) 유형인 경우, 이 인덱스의 재작성에 정렬이 필요하지 않습니다. 대신, 새로 재구성된 데이터를 스캔하여 이 인덱스가 재빌드됩니다. 재작성되는 기타 인덱스에는 정렬이 필요하며, 재구성되는 테이블의 크기까지의 임시 스페이스가 필요할 수 있습니다.

오프라인 테이블 Reorg 조작에서는 제어 로그 레코드를 거의 생성하지 않으므로 상대적으로 작은 양의 로그 스페이스를 사용합니다. REORG 유틸리티에서 인덱스를 사용하지 않을 경우, 테이블 데이터 로그 레코드만 작성됩니다. 인덱스가 지정된 경우, 또는 테이블에 클러스터링 인덱스가 있는 경우, 새 버전의 테이블에 배치된 순서로 레코드 ID(RID)가 로그됩니다. 각 RID 로그 레코드는 최대 8000개 RID를 보유하며, 각 RID는 4바이트를 사용합니다. 이는 오프라인 테이블 Reorg 조작 동안 로그 스페이스 문제의 원인이 될 수 있습니다. RID는 데이터베이스가 복구 가능한 경우에만 로그됩니다.

온라인 테이블 재구성에 대한 로그 스페이스 요구사항

온라인 테이블 Reorg 조작에 필요한 로그 스페이스는 일반적으로 오프라인 테이블 Reorg에 필요한 것보다 더 큽니다. 필요한 스페이스의 양은 재구성되는 행의 수, 인덱스 수, 인덱스 키의 크기 및 시작 시 테이블이 불충분하게 구성된 정도에 따라 결정됩니다. 테이블과 연관된 로그 스페이스 소비에 대한 일반적인 벤치 마크를 설정하는 것이 좋습니다.

테이블의 모든 행은 온라인 테이블 Reorg 조작 동안 두 번 이동될 수 있습니다. 각 인덱스에 대해, 각 테이블 행은 인덱스 키를 갱신하여 새 위치를 반영해야 하고, 이전 위치에 대한 모든 액세스가 완료된 후 인덱스 키가 다시 갱신되어 이전 RID에 대한 참조를 제거합니다. 행이 다시 이동될 때 인덱스 키에 대한 갱신이 다시 수행됩니다. 이러한 모든 활동은 로그되어 온라인 테이블 재구성을 완전히 복구 가능하게 만듭니다. 각 행에 대해(하나의 인덱스 가정) 최소 2개의 데이터 로그 레코드(각각 행 데이터 포함)와 4개의 인덱스 로그 레코드(각각 키 데이터 포함)가 있습니다. 특히 클러스터링 인덱스는 인덱스 페이지를 채워, 역시 로그되어야 하는 인덱스 분할 및 병합을 야기하는 경향이 있습니다.

온라인 테이블 REORG 유틸리티는 자주 내부 COMMIT문을 발행하므로, 일반적으로 많은 수의 사용 중인 로그를 보유하지 않습니다. 유틸리티에서 S 테이블 잠금을 요청할 경우 절단 단계 동안 예외가 발생할 수 있습니다. 유틸리티는 잠금을 획득할 수 없는 경우, 대기하고, 그 동안 다른 트랜잭션에서 로그를 재빨리 채울 수도 있습니다.

테이블 및 인덱스 재구성 필요 감소

다양한 전략을 사용하여 테이블 및 인덱스 재구성에 대한 필요(및 연관된 비용)를 감소시킬 수 있습니다.

테이블 재구성 필요 감소

테이블 재구성에 대한 필요를 줄이려면 다음을 수행하십시오.

- 다중 파티션 테이블을 사용하십시오.
- 다차원 클러스터링(MDC) 테이블을 작성하십시오. MDC 테이블의 경우, CREATE TABLE문의 ORGANIZE BY DIMENSIONS 절을 사용하여 지정하는 컬럼에서 클러스터링이 유지됩니다. 그러나 reorgchk 유틸리티에서 미사용 블록이 너무 많이 있거나 블록을 압축해야 한다고 판별할 경우 MDC 테이블의 재구성을 계속 권장할 수도 있습니다.
- 테이블에서 APPEND 모드를 사용하십시오. 예를 들어, 새 행에 대한 인덱스 키 값이 항상 새로운 높은 키 값인 경우, 테이블의 클러스터링 속성이 테이블의 마지막에 키 값을 배치하려고 시도합니다. 이 경우, 클러스터링 인덱스를 사용하는 것보다 APPEND 모드를 사용하는 것이 더 나은 선택이 될 수 있습니다.

테이블 재구성에 대한 필요를 한층 더 줄이려면 테이블을 작성한 후 다음 태스크를 수행하십시오.

- 테이블을 변경하여 로드 또는 테이블 재구성 조작 동안 여유 공간으로 남을 각 페이지의 퍼센트를 지정하십시오(PCTFREE).
- PCTFREE 옵션을 지정하여 클러스터링 인덱스를 작성하십시오.
- 테이블에 로드하기 전에 데이터를 정렬하십시오.

이러한 태스크를 수행한 후에는, 테이블의 PCTFREE 설정 및 클러스터링 인덱스를 통해 원래 정렬 순서를 유지하는 데 도움을 받을 수 있습니다. 테이블 페이지에 충분한 스페이스가 있는 경우, 올바른 페이지에서 새 데이터가 삽입되어 인덱스의 클러스터링 특성을 유지할 수 있습니다. 그러나 더 많은 데이터가 삽입되어 테이블 페이지가 가득 차게 되면, 레코드가 테이블의 마지막에 추가되고, 점차 클러스터 해제됩니다.

클러스터링 인덱스를 작성한 후 테이블 Reorg 조작 또는 정렬 및 로드 조작을 수행할 경우, 인덱스에서 데이터의 순서를 유지하려고 시도하여, runstats 유틸리티에 의해 수집된 CLUSTERRATIO 또는 CLUSTERFACTOR 통계를 향상시킵니다.

인덱스 재구성 필요 감소

인덱스 재구성에 대한 필요를 줄이려면 다음을 수행하십시오.

- PCTFREE 또는 LEVEL2 PCTFREE 옵션을 지정하여 클러스터링 인덱스를 작성하십시오.
- MINPCTUSED 옵션을 사용하여 인덱스를 작성하십시오. 또는, 리프 페이지를 병합하기 위해 REORG INDEXES 명령의 CLEANUP ONLY ALL 옵션 사용을 고려하십시오.

자동 재구성

테이블 데이터에 대한 많은 변경 후, 테이블 및 해당 인덱스가 분할될 수 있습니다. 논리적 순차 데이터가 비연속 페이지에 상주하여, 데이터베이스 관리 프로그램이 추가 읽기 조작을 수행하여 데이터에 액세스해야 하도록 만들 수 있습니다.

runstats 유틸리티에 의해 수집된 통계 정보는 테이블 내 데이터의 분산을 보여줍니다. 이러한 통계를 분석하면 언제, 어떤 종류의 재구성이 필요한지 알 수 있습니다.

자동 재구성 프로세스는 reorgchk 유틸리티에 속한 공식을 사용하여 테이블 또는 인덱스 재구성이 필요한지 판별합니다. 이 프로세스는 통계를 업데이트한 테이블 및 인덱스를 주기적으로 평가하여 재구성이 필요한지 여부를 확인하고, 필요할 때마다 그러한 조작을 스케줄합니다.

자동 재구성 기능은 **auto_reorg**, **auto_tbl_maint** 및 **auto_maint** 데이터베이스 구성 매개변수를 통해 사용하거나 사용하지 않도록 할 수 있습니다.

파티션된 데이터베이스 환경에서, 자동 재구성은 카탈로그 데이터베이스 파티션에서 시작되고 해당 파티션에서만 이러한 구성 매개변수를 사용해야 합니다. 그러나 재구성 조작은 목표 테이블이 상주하는 모든 데이터베이스 파티션에서 실행됩니다.

언제, 어떻게 테이블 및 인덱스를 재구성할지에 대해 확신이 없는 경우, 자동 재구성을 전체 데이터베이스 유지보수 플랜의 일부로 통합할 수 있습니다.

또한 다차원 클러스터링(MDC) 테이블을 작성하여 스페이스를 재개할 수 있습니다. MDC 테이블에서 Extent 비우기는 DMS 테이블 스페이스의 MDC 테이블에 대해서만 지원됩니다. MDC 테이블에서 Extent 비우기는 데이터베이스의 자동 유지보수 활동의 일부일 수 있습니다.

자동 테이블 및 인덱스 재구성 사용

자동 테이블 및 인덱스 재구성을 사용하면 데이터를 재구성할 시기 및 방법에 대해 걱정할 필요가 없습니다.

잘 구성된 테이블 및 인덱스 데이터를 갖는 것은 효율적인 데이터 액세스와 최적의 워크로드 성능에 매우 중요합니다. 많은 삽입, 갱신 및 삭제 조작 후, 논리적 순차 테이블 데이터가 비연속 데이터 페이지에 상주하게 될 수도 있으므로, 데이터베이스 관리 프로그램이 추가 읽기 조작을 수행하여 데이터에 액세스해야 합니다. 상당한 수의 행이 삭제된 테이블의 데이터에 액세스할 때에도 추가 읽기 조작이 필요합니다. DB2 서버를 사용하여 시스템 카탈로그 테이블뿐 아니라 사용자 테이블을 재구성할 수 있습니다.

데이터베이스를 자동 재구성하려면 다음 각 구성 매개변수를 ON으로 설정하십시오.

- **auto_maint**
- **auto_tbl_maint**

- **auto_reorg**

응용프로그램 설계

데이터베이스 응용프로그램 설계는 응용프로그램 성능에 영향을 미치는 인수 중 하나입니다. 데이터베이스 응용프로그램의 성능을 최대화하는 데 도움이 되는 응용프로그램 설계 고려사항에 대한 자세한 내용은 이 절을 검토하십시오.

응용프로그램 프로세스, 동시성 및 복구

모든 SQL 프로그램은 *응용프로그램 프로세스* 또는 에이전트의 일부분으로 실행됩니다. 응용프로그램 프로세스는 하나 이상의 프로그램을 실행해야 하며 데이터베이스 관리 프로그램이 자원 및 잠금을 할당하는 단위입니다. 응용프로그램 프로세스가 다르면 다른 프로그램을 실행하거나 동일한 프로그램을 다르게 실행할 수 있습니다.

여러 응용프로그램 프로세스가 동일한 데이터에 동시에 액세스를 요청할 수 있습니다. *잠금*은 두 개의 응용프로그램 프로세스가 동일한 데이터 행을 동시에 갱신하지 못하도록 예방하여 데이터 무결성을 유지하기 위해 사용하는 메커니즘입니다.

데이터베이스 관리 프로그램은 한 응용프로그램 프로세스의 커미트되지 않은 변경사항을 다른 프로세스에서 우연히 감지하는 경우를 예방하기 위해 잠금을 획득합니다. 데이터베이스 관리 프로그램은 응용프로그램 프로세스가 종료되면 해당 프로세스가 획득하고 보유한 모든 잠금을 해제합니다. 그러나 응용프로그램 프로세스는 잠금을 해제하도록 직접 요청할 수 있습니다. 이 경우 *커미트* 조작을 사용하며, 이 조작은 하나의 작업 단위(UOW) 중 획득한 잠금을 해제하고 이 작업 단위 중 데이터베이스 변경사항도 커미트합니다.

*작업 단위(UOW)*는 응용프로그램 프로세스에서 복구 가능한 조작 시퀀스입니다. 응용프로그램 프로세스가 시작되거나 응용프로그램 프로세스 종료 이외의 이유로 이전 UOW가 종료된 경우 작업 단위가 시작됩니다. 커미트 조작, 롤백 조작 또는 응용프로그램 프로세스 종료 시 작업 단위가 종료됩니다. 커미트 또는 롤백 조작은 종료 중인 UOW에서 발생한 데이터베이스 변경사항에만 영향을 줍니다.

데이터베이스 관리 프로그램은 응용프로그램 프로세스가 수행한 커미트되지 않은 변경사항을 백아웃하는 방법을 제공합니다. 응용프로그램 프로세스의 일부분에 장애가 발생했거나 교착 상태 또는 잠금 시간종료가 발생한 경우 이러한 방법을 사용해야 할 수도 있습니다. 응용프로그램 프로세스는 데이터베이스 변경을 취소하도록 명시적으로 요청할 수 있습니다. 이 경우 *롤백* 조작을 사용합니다.

이러한 변경사항이 계속 커미트되지 않으면 다른 응용프로그램 프로세스가 해당 변경사항을 볼 수 없으며 변경사항이 롤백될 수 있습니다. 그러나 일반 분리 수준이 커미트되지 않은 읽기(UR)인 경우에는 위의 내용이 적용되지 않습니다. 이러한 데이터베이스 변

경사항이 커미트되었으면 다른 응용프로그램 프로세스가 이러한 변경사항에 액세스할 수 있으며 변경사항은 더 이상 롤백되지 않습니다.

DB2 콜 레벨 인터페이스(CLI) 및 Embedded SQL은 각각 독립 트랜잭션인 여러 연결을 지원하는 *동시 트랜잭션*이라는 연결 모드를 허용합니다. 응용프로그램은 동일한 데이터베이스에 대한 여러 개의 동시 연결을 사용할 수 있습니다.

응용프로그램 프로세스 대신 데이터베이스 관리 프로그램이 획득한 잠금은 분리 수준이 커서 안정성(CS, 커서가 한 행에서 다른 행으로 이동하면 잠금이 해제됨)이거나 커미트되지 않은 읽기(UR)인 경우를 제외하고는 UOW가 종료될 때까지 보유합니다.

응용프로그램 프로세스는 자체 잠금으로 인해 조작 수행이 금지되지 않습니다. 그러나 응용프로그램이 동시 트랜잭션을 사용하는 경우 한 트랜잭션의 잠금이 동시 트랜잭션의 조작에 영향을 줄 수 있습니다.

UOW의 시작 및 종료는 응용프로그램 프로세스에서 *일관성 지점*을 정의합니다. 예를 들어, 한 계좌에서 다른 계좌로 예금을 이체하는 은행 거래가 있습니다. 이러한 거래에서는 해당 예금을 첫 번째 계좌에서 뺀 후 두 번째 계좌에 더해야 합니다. 빼기 단계 이후 데이터가 불일치합니다. 예금을 두 번째 계좌에 더한 후에만 다시 일치하게 됩니다. 두 단계가 모두 완료되었으면 커미트 조작을 사용하여 UOW를 종료하여 다른 응용프로그램 프로세스에서 변경사항을 사용할 수 있도록 합니다. UOW가 종료되기 전에 장애가 발생한 경우 데이터베이스 관리 프로그램은 커미트되지 않은 변경사항을 롤백하여 데이터 일관성을 복원합니다.

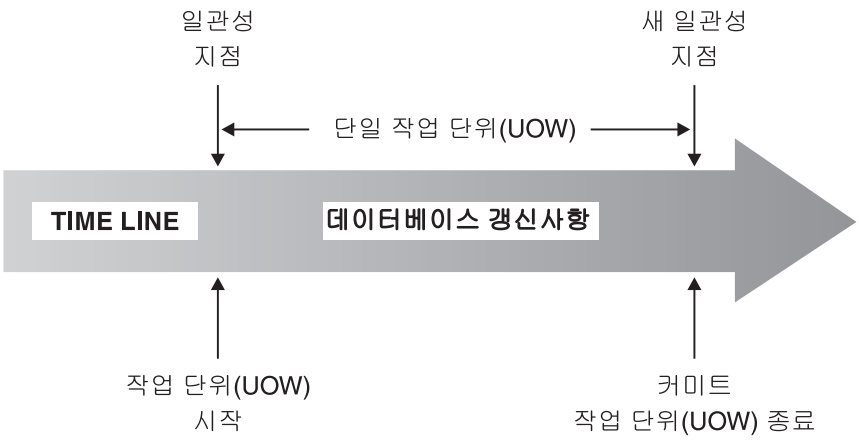


그림 21. COMMIT 문을 사용하는 작업 단위(UOW)

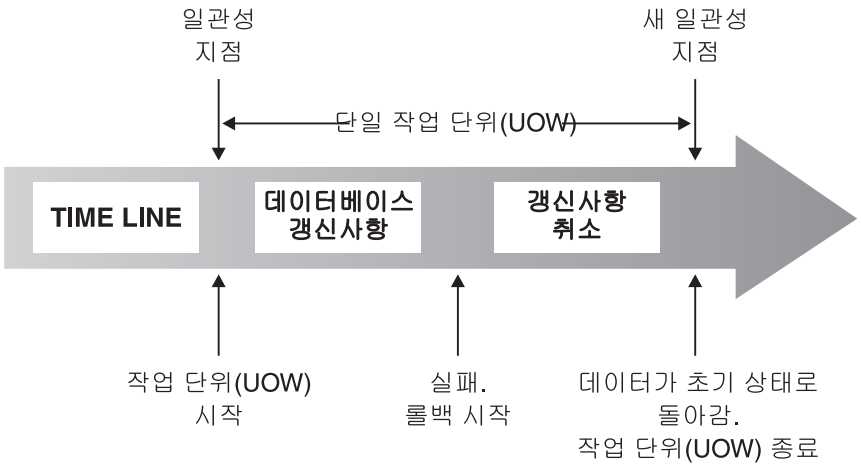


그림 22. ROLLBACK 문을 사용하는 작업 단위(UOW)

동시성 문제

여러 사용자가 관계형 데이터베이스의 데이터에 액세스하고 변경하므로 데이터베이스 관리 프로그램은 사용자가 데이터 무결성이 보존되는지 확인하는 동안 이러한 변경을 수행할 수 있도록 허용해야 합니다.

*동시성*은 여러 대화식 사용자 또는 응용프로그램이 동시에 자원을 공유함을 나타냅니다. 데이터베이스 관리 프로그램은 이 액세스를 제어하여 다음과 같이 바람직하지 못한 영향을 방지합니다.

- **갱신사항 유실**. 두 개의 응용프로그램 A와 B가 모두 동일한 행을 읽고 이러한 응용프로그램이 읽은 데이터에 따라 컬럼 중 하나의 새 값을 계산할 수 있습니다. A가 행을 갱신한 후 B도 이 행을 갱신한 경우 A의 갱신이 유실됩니다.
- **커미트되지 않은 데이터에 대한 액세스**. A 응용프로그램이 값을 갱신하고 이 값이 커미트되기 전에 B가 읽을 수 있습니다. 그런 다음 A가 이 갱신을 백아웃하면 B가 수행하는 계산은 유효하지 않은 데이터에 기초할 수 있습니다.
- **비반복 읽기**. A 응용프로그램은 다른 요청을 처리하기 전에 행을 읽을 수 있습니다. 그동안 B는 행을 수정하거나 삭제하고 변경사항을 커미트합니다. 나중에 A가 원래의 행을 다시 읽으려고 시도하면 수정된 행을 보거나 원래의 행이 삭제되었음을 발견합니다.
- **팬텀 읽기**. A 응용프로그램은 일부 검색 기준에 따라 행 세트를 읽는 쿼리를 실행할 수 있습니다. B 응용프로그램은 새 데이터를 삽입하거나 A 응용프로그램의 쿼리를 충족시키는 기존 데이터를 갱신합니다. A 응용프로그램은 동일한 작업 단위(UOW)에서 쿼리를 다시 실행하고 일부 추가(『팬텀』) 값이 리턴됩니다.

전역 임시 테이블은 해당 테이블을 선언하거나 작성한 응용프로그램만 사용할 수 있으므로 이 경우 동시성은 문제가 되지 않습니다.

페더레이티드 데이터베이스 시스템에서 동시처리 제어

*페더레이티드 데이터베이스 시스템*은 단일 명령문에 두 개 이상의 데이터베이스 관리 시스템(DMBS)을 참조하는 SQL문을 제출하여 응용프로그램 및 사용자를 지원합니다. 이러한 데이터 소스(각각 DMBS 및 데이터로 구성됨)를 참조하기 위해 DB2 서버는 별칭을 사용합니다. 별칭은 다른 DBMS에 있는 오브젝트의 별명입니다. 페더레이티드 시스템에서 DB2 서버는 요청된 데이터를 호스트하는 데이터베이스 관리 프로그램의 동시처리 제어 프로토콜에 의존합니다.

DB2 페더레이티드 시스템은 데이터베이스 오브젝트에 *위치 투명성*을 제공합니다. 예를 들어, 테이블 및 뷰에 대한 정보가 이동되는 경우 해당 정보에 대한 참조(별칭 이용)는 이 정보를 요청하는 응용프로그램을 변경하지 않고 갱신할 수 있습니다. 응용프로그램이 별칭을 통해 데이터에 액세스하는 경우 DB2 서버는 데이터 소스의 동시처리 제어 프로토콜에 의존하여 분리 수준이 시행되었는지 확인합니다. DB2 서버는 데이터 소스에서 요청된 분리 수준을 논리적으로 동등한 것과 일치시키려고 시도하지만 결과는 데이터 소스 기능에 따라 다를 수 있습니다.

분리 수준

응용프로그램 프로세스와 연관된 *분리 수준*은 프로세스에서 액세스 중인 데이터를 잠그거나 동시에 실행 중인 기타 프로세스에서 분리하는 정도를 결정합니다. 분리 수준은 작업 단위 중에만 적용됩니다.

따라서 응용프로그램 프로세스의 분리 수준은 다음을 지정합니다.

- 응용프로그램이 읽거나 갱신하는 행을 동시에 실행 중인 다른 응용프로그램 프로세스에서 사용할 수 있는 정도
- 동시에 실행 중인 다른 응용프로그램 프로세스의 갱신 활동이 응용프로그램에 영향을 줄 수 있는 정도

정적 SQL문의 분리 수준은 패키지의 속성으로 지정되며 해당 패키지를 사용하는 응용프로그램 프로세스에 적용됩니다. 분리 수준은 ISOLATION 바인드 또는 프리컴파일 옵션을 설정하여 프로그램 준비 프로세스 중 지정합니다. 동적 SQL문의 경우 디폴트 분리 수준은 명령문을 준비하는 패키지에 대해 지정된 분리 수준입니다. 세션에서 발행된 동적 SQL문에 대해 다른 분리 수준을 지정하려면 SET CURRENT ISOLATION문을 사용하십시오. 자세한 정보는 『CURRENT ISOLATION 특수 레지스터』를 참조하십시오. 정적 SQL문 및 동적 SQL문에서 *select-statement*의 *isolation-clause*는 특수 레지스터(설정된 경우) 및 바인드 옵션 값을 모두 겹쳐씁니다. 자세한 정보는 『Select-statement』를 참조하십시오.

분리 수준은 잠금으로 실행되며 사용되는 잠금 유형에 따라 동시 응용프로그램 프로세스가 데이터에 대한 액세스 제한 및 방지 수준이 다릅니다. 선언된 임시 테이블 및 해당 행은 이를 선언한 응용프로그램만 액세스할 수 있으므로 잠글 수 없습니다.

데이터베이스 관리 프로그램은 세 가지 일반 잠금 범주를 지원합니다.

공유(S)

S 잠금에서 동시 응용프로그램 프로세스는 데이터에 대한 읽기 전용 조작으로 제한됩니다.

갱신(U)

U 잠금에서 동시 응용프로그램 프로세스는 행을 갱신할 수 있음을 선언하지 않은 경우 데이터에 대한 읽기 전용 조작으로 제한됩니다. 데이터베이스 관리 프로그램은 현재 행을 보는 프로세스가 행을 갱신할 수도 있는 것으로 가정합니다.

독점(X)

X 잠금에서 동시 응용프로그램 프로세스는 어떠한 방식으로든 데이터에 액세스할 수 없습니다. 이 사항은 데이터를 읽을 수 있지만 수정할 수 없는 커미트되지 않은 읽기(UR) 분리 수준인 응용프로그램 프로세스의 경우에는 적용되지 않습니다.

분리 수준에 관계없이 데이터베이스 관리 프로그램은 삽입, 갱신 또는 삭제된 모든 행에 배타적 잠금을 설정합니다. 따라서 모든 분리 수준은 작업 단위 중 응용프로그램 프로세스가 변경한 행을 작업 단위가 완료되기 전까지 다른 응용프로그램 프로세스에서 변경하지 못하도록 보장합니다.

데이터베이스 관리 프로그램은 네 가지 분리 수준을 지원합니다.

- 『반복 읽기(RR)』
- 154 페이지의 『읽기 안정성(RS)』
- 155 페이지의 『커서 안정성(CS)』
- 155 페이지의 『커미트되지 않은 읽기(UR)』

주: 일부 호스트 데이터베이스 서버는 *커미트 없음(NC)* 분리 수준을 지원합니다. 기타 데이터베이스 서버에서 이 분리 수준은 커미트되지 않은 읽기 분리 수준과 비슷한 작업을 수행합니다.

각 분리 수준에 대한 자세한 설명은 성능 영향의 내림차순과 데이터 액세스 또는 갱신 시 필요한 주의사항의 오름차순으로 표시됩니다.

반복 읽기(RR)

반복 읽기(RR) 분리 수준은 응용프로그램이 작업 단위(UOW) 중 참조하는 모든 행을 잠급니다. 응용프로그램이 동일한 작업 단위에서 SELECT문을 발행하면 매번 동일한 결과가 리턴됩니다. RR에서 갱신사항 유실, 커미트되지 않은 데이터에 대한 액세스, 비반복 읽기 및 팬텀 읽기는 가능하지 않습니다.

RR에서 응용프로그램은 UOW가 완료될 때까지 필요하면 여러 번 행을 검색하고 해당 행에 대해 작업을 수행할 수 있습니다. 그러나 UOW가 완료되기 전까지 다른 응용프로그램이 결과 세트에 영향을 주는 행을 갱신, 삭제 또는 삽입할 수 없습니다. RR 분리 수준에서 실행 중인 응용프로그램은 다른 응용프로그램의 커미트되지 않은 변경사항을 볼 수 없습니다. 이 분리 수준은 응용프로그램이 데이터를 보기 전까지(임시 테이블 또는 행 블로킹 사용 시도 포함) 리턴된 모든 데이터를 그대로 변경하지 않습니다.

검색된 행만이 아닌 참조된 모든 행을 잠급니다. 예를 들어, 10,000개의 행을 스캔하고 이러한 행에 술어를 적용하는 경우 10개의 행만 규정된 경우에도 10,000개의 모든 행에서 잠금을 유지합니다. 쿼리를 다시 실행하는 경우 해당 쿼리가 참조한 행 목록에 추가되는 행을 다른 응용프로그램에서 삽입하거나 갱신할 수 없습니다. 따라서 팬텀 읽기가 방지됩니다.

RR이 상당한 수의 잠금을 획득할 수 있으므로 이 수는 **locklist** 및 **maxlocks** 데이터베이스 구성 매개변수에서 지정한 한계를 초과할 수 있습니다. 잠금 에스컬레이션을 방지하기 위해 옵티마이저는 잠금 에스컬레이션이 유망한 경우 인덱스 스캔을 위해 단일 테이블 레벨 잠금을 획득하도록 선택할 수 있습니다. 테이블 레벨 잠금을 원하지 않는 경우 읽기 안정성 분리 수준을 사용하십시오.

참조 제한조건을 평가하는 동안 DB2 서버는 사용자가 이전에 설정한 분리 수준에 관계없이 하위 테이블의 스캔에서 사용되는 분리 수준을 RR로 업그레이드하는 경우가 있습니다. 이 경우 커미트되기 전까지 추가 잠금을 유지하므로 교착 상태 또는 잠금 시간 종료가 발생할 가능성이 증가합니다. 이러한 문제를 방지하려면 외부 키 컬럼만 들어 있으며 참조 무결성 스캔에서 대신 사용할 수 있는 인덱스를 작성하십시오.

읽기 안정성(RS)

읽기 안정성 분리 수준은 작업 단위 중 응용프로그램이 검색하는 행만 잠급니다. RS를 사용하면 UOW가 완료될 때까지 UOW 중 규정 행 읽기를 다른 응용프로그램 프로세스에서 변경할 수 없으며 해당 프로세스에서 변경사항을 커미트할 때까지 다른 응용프로그램 프로세스에서 변경한 행을 읽을 수 없습니다. RS에서는 커미트되지 않은 데이터 및 비반복 읽기에 액세스할 수 없습니다. 그러나 팬텀 읽기는 가능합니다.

이 분리 수준은 응용프로그램이 데이터를 보기 전까지(임시 테이블 또는 행 블로킹 사용 시도 포함) 리턴된 모든 데이터를 그대로 변경하지 않습니다.

RS 분리 수준은 높은 수준의 동시성과 안정적인 데이터 뷰를 모두 제공합니다. 이로 인해 옵티마이저를 사용하면 잠금 에스컬레이션이 발생할 때까지 테이블 레벨 잠금을 확보할 수 없습니다.

RS 분리 수준은 다음과 같은 응용프로그램에 적합합니다.

- 동시 환경에서 작동되는 응용프로그램
- 작업 단위 중에 계속 안정적인 상태의 규정 행이 필요한 응용프로그램

- 작업 단위 중 동일한 쿼리를 여러 번 발행하지 않거나 작업 단위 중 쿼리가 여러 번 발행된 경우 동일한 결과 세트가 필요하지 않은 응용프로그램

커서 안정성(CS)

커서 안정성 분리 수준은 해당 행에 커서가 있는 동안 트랜잭션 중 액세스하는 행을 잠급니다. 다음 행을 페치하거나 트랜잭션이 종료되기 전까지 이 잠금은 계속 유효합니다. 그러나 행의 데이터가 변경된 경우 변경사항이 커미트되기 전까지 잠금을 유지합니다.

이 분리 수준에서는 갱신 가능한 커서가 해당 행에 있는 동안 다른 응용프로그램이 행을 갱신하거나 삭제할 수 없습니다. CS에서는 다른 응용프로그램의 커미트되지 않은 데이터에 액세스할 수 없습니다. 그러나 비반복 읽기 및 팬텀 읽기는 가능합니다.

CS가 디폴트 분리 수준입니다. 동시성을 최대한 확대하고 커미트된 데이터만 확인해야 하는 경우에 적합합니다.

주: 버전 9.7에서 도입된 *현재 커미트됨* 시맨틱에서는 이전의 경우와 마찬가지로 커미트된 데이터만 리턴되지만 이제 데이터를 읽을 사용자는 갱신자가 행 잠금을 해제할 때까지 대기할 필요가 없습니다. 대신, 데이터를 읽을 사용자는 현재 커미트된 버전을 따르는 데이터(즉, 쓰기 조작이 시작되기 전의 데이터)를 리턴합니다.

커미트되지 않은 읽기(UR)

커미트되지 않은 읽기 분리 수준을 사용하면 응용프로그램은 다른 트랜잭션의 커미트되지 않은 변경사항에 액세스할 수 있습니다. 또한 UR은 응용프로그램이 테이블을 변경하거나 삭제하려고 하지 않는 한, 읽고 있는 행에 다른 응용프로그램이 액세스할 수 없도록 예방하지 않습니다.

UR에서는 커미트되지 않은 데이터에 대한 액세스, 비반복 읽기 및 팬텀 읽기가 가능합니다. 이 분리 수준은 읽기 전용 테이블에 대해 쿼리를 실행하거나 SELECT문만 발행하는 경우 및 다른 응용프로그램에서 커미트하지 않은 데이터를 보는 것이 문제가 되지 않을때 적합합니다.

UR은 읽기 전용 및 갱신 가능한 커서의 경우에 다르게 작동합니다.

- 읽기 전용 커서는 다른 트랜잭션의 커미트되지 않은 대부분의 변경사항에 액세스할 수 있습니다.
- 다른 트랜잭션에서 작성 또는 삭제 중인 테이블, 뷰 및 인덱스는 트랜잭션이 처리되는 동안 사용할 수 없습니다. 다른 트랜잭션의 기타 변경사항은 커미트 또는 롤백되기 전에 읽을 수 있습니다. UR에서 작동되는 갱신 가능한 커서는 분리 수준이 CS일 때와 마찬가지로 작동합니다.

커미트되지 않은 읽기 응용프로그램이 앰비규어스 커서를 사용하는 경우 실행 시 CS 분리 수준을 사용할 수 있습니다. PREP 또는 BIND 명령에서 BLOCKING 옵션의 값이 UNAMBIG(디폴트)인 경우 앰비규어스 커서는 CS로 에스컬레이션될 수 있습니다. 이 에스컬레이션을 방지하려면 다음을 수행하십시오.

- 응용프로그램의 커서를 명확한 것으로 수정하십시오. FOR READ ONLY절을 포함하여 SELECT문을 변경하십시오.
- 응용프로그램의 커서를 앰비규어스로 유지하지만 BLOCKING ALL 및 STATICREADONLY YES 옵션을 사용하여 프로그램을 프리컴파일하거나 바인드하여 프로그램 실행 시 앰비규어스 커서를 읽기 전용으로 처리하도록 하십시오.

분리 수준 비교

표 3에서는 지원되는 분리 수준을 요약합니다.

표 3. 분리 수준 비교

	UR	CS	RS	RR
응용프로그램이 다른 응용프로그램 프로세스에서 수행한 커미트되지 않은 변경사항을 볼 수 있습니까?	예	아니오	아니오	아니오
응용프로그램이 다른 응용프로그램 프로세스에서 수행한 커미트되지 않은 변경사항을 갱신할 수 있습니까?	아니오	아니오	아니오	아니오
명령문의 재실행이 다른 응용프로그램 프로세스의 영향을 받을 수 있습니까? [1]	예	예	예	아니오[2]
갱신된 행을 다른 응용프로그램 프로세스에서 갱신할 수 있습니까? [3]	아니오	아니오	아니오	아니오
갱신된 행을 UR이 아닌 분리 수준에서 실행 중인 다른 응용프로그램 프로세스에서 읽을 수 있습니까?	아니오	아니오	아니오	아니오
갱신된 행을 UR 분리 수준에서 실행 중인 다른 응용프로그램 프로세스에서 읽을 수 있습니까?	예	예	예	예
액세스한 행을 다른 응용프로그램 프로세스에서 갱신할 수 있습니까? [4]	예	예	아니오	아니오
액세스한 행을 다른 응용프로그램 프로세스에서 읽을 수 있습니까?	예	예	예	예
갱신할 현재 행을 다른 응용프로그램 프로세스에서 갱신하거나 삭제할 수 있습니까? [5]	예/아니오[6]	예/아니오[6]	아니오	아니오

표 3. 분리 수준 비교 (계속)

	UR	CS	RS	RR

주:

1. *팬텀 읽기 현상*의 예는 다음과 같습니다. 작업 단위 UW1은 일부 검색 조건을 충족시키는 *n*개의 행 세트를 읽습니다. 작업 단위 UW2는 동일한 검색 조건을 충족시키는 하나 이상의 행을 삽입한 후 커미트합니다. UW1이 동일한 검색 조건으로 읽기를 반복하는 경우 다른 결과 세트가 표시됩니다. 원래 읽은 행에 UW2에서 삽입한 행이 함께 표시됩니다.
2. 두 번의 읽기 사이에 레이블 기반 액세스 제어(LBAC) 증명서가 변경된 경우 다른 행에 액세스할 수 있으므로 두 번째 읽기의 결과가 다를 수 있습니다.
3. 응용프로그램이 테이블에서 읽고 테이블에 쓰는 경우 분리 수준은 응용프로그램에 대한 보호 기능을 제공하지 않습니다. 예를 들어, 응용프로그램은 테이블에서 커서를 열고 동일한 테이블에서 삽입, 갱신 또는 삭제 조작을 수행합니다. 열린 커서에서 추가 행이 페치되는 경우 응용프로그램이 일치하지 않는 데이터를 볼 수도 있습니다.
4. *비반복 읽기 현상*의 예는 다음과 같습니다. 작업 단위 UW1이 한 행을 읽습니다. 작업 단위 UW2가 이 행을 수정하고 커미트합니다. UW1이 이후에 이 행을 다시 읽는 경우 다른 값을 볼 수도 있습니다.
5. *DR(Dirty Read)* 현상의 예는 다음과 같습니다. 작업 단위 UW1이 한 행을 수정합니다. 작업 단위 UW2가 UW1이 커미트하기 전에 해당 행을 읽습니다. UW1이 이후에 변경사항을 롤백하는 경우 UW2가 존재하지 않는 데이터를 읽었습니다.
6. UR 또는 CS에서 커서를 갱신할 수 없는 경우 다른 응용프로그램 프로세스에서 현재 행을 갱신하거나 삭제할 수도 있습니다. 예를 들어, 버퍼링으로 클라이언트의 현재 행이 서버의 현재 행과 다를 수 있습니다. 또한 CS에서 현재 커미트된 시맨틱을 사용하는 경우 읽고 있는 행에 보류 중인 커미트되지 않은 갱신사항이 포함될 수도 있습니다. 이 경우 현재 커미트된 행 버전은 항상 응용프로그램으로 리턴됩니다.

분리 수준 요약

표 4에서는 여러 가지 다른 분리 수준과 연관된 동시성 문제를 보여줍니다.

표 4. 분리 수준 요약

분리 수준	커미트되지 않은 데이터에 대한 액세스	비반복 읽기	팬텀 읽기
반복 읽기(RR)	불가능	불가능	불가능
읽기 안정성(RS)	불가능	불가능	가능
커서 안정성(CS)	불가능	가능	가능
커미트되지 않은 읽기(UR)	가능	가능	가능

잠금을 확보하고 해제하는 데 필요한 처리 및 메모리 자원은 분리 수준에 따라 다르므로 분리 수준은 여러 응용프로그램들 간 분리 정도와 개별 응용프로그램의 성능에 영향을 줍니다. 교착 상태 가능성도 분리 수준에 따라 다릅니다. 158 페이지의 표 5에서는 응용프로그램의 초기 분리 수준을 선택하는 데 유용한 간단한 방법을 제공합니다.

표 5. 분리 수준 선택 지침

응용프로그램 유형	높은 데이터 안정성 필요	높은 데이터 안정성 불필요
읽기 쓰기 트랜잭션	RS	CS
읽기 전용 트랜잭션	RR 또는 RS	UR

분리 수준 지정

분리 수준은 데이터가 액세스되는 동안 데이터가 다른 프로세스에서 분리되는 방법을 판별하므로, 데이터 무결성과 동시성 요구사항의 균형을 맞추는 분리 수준을 선택해야 합니다.

지정하는 분리 수준은 작업 단위(UOW)의 지속기간 동안 적용됩니다. SQL 또는 XQuery 문을 컴파일할 때 어떤 분리 수준이 사용되는지 판별하는 데 다음 발견적 방법이 사용됩니다.

- 정적 SQL의 경우:
  - 명령문에서 *isolation-clause*가 지정된 경우, 해당 절의 값이 사용됩니다.
  - 명령문에서 *isolation-clause*가 지정되지 않은 경우, 패키지가 데이터베이스에 바인드될 때 패키지에 대해 지정된 분리 수준이 사용됩니다.
- 동적 SQL의 경우:
  - 명령문에서 *isolation-clause*가 지정된 경우, 해당 절의 값이 사용됩니다.
  - 명령문에서 *isolation-clause*가 지정되어 있지 않고 현재 세션 내에서 SET CURRENT ISOLATION문이 발행된 경우, CURRENT ISOLATION 특수 레지스터의 값이 사용됩니다.
  - 명령문에 *isolation-clause*가 지정되어 있지 않고 현재 세션 내에서 SET CURRENT ISOLATION문이 발행되지 않은 경우, 패키지가 데이터베이스에 바인드될 때 패키지에 대해 지정된 분리 수준이 사용됩니다.
- 정적 또는 동적 XQuery문의 경우, 환경의 분리 수준이 XQuery 표현식이 평가될 때 사용되는 분리 수준을 결정합니다.

주: 많은 상업적으로 작성된 응용프로그램에서는 분리 수준 선택 방법을 제공합니다. 자세한 내용은 응용프로그램 문서를 참조하십시오.

분리 수준은 여러 다양한 방법으로 지정할 수 있습니다.

- **명령문 레벨에서:**

  주: XQuery문에 대한 분리 수준은 명령문 레벨에서 지정할 수 없습니다.

  WITH 절을 사용하십시오. WITH 절을 서브쿼리에서 사용할 수 없습니다. WITH UR 옵션은 읽기 전용 조작에만 적용됩니다. 기타 경우에는, 명령문이 자동으로 UR에서 CS로 변경됩니다.

이 분리 수준은 명령문이 나타나는 패키지에 대해 지정된 분리 수준을 겹쳐씁니다. 다음 SQL문에 대한 분리 수준을 지정할 수 있습니다.

- DECLARE CURSOR
- 검색된 DELETE
- INSERT
- SELECT
- SELECT INTO
- 검색된 UPDATE

- **현재 세션 내 동적 SQL의 경우:**

  SET CURRENT ISOLATION문을 사용하여 세션 내 발행된 동적 SQL에 대한 분리 수준을 설정하십시오. 이 명령문을 발행하면 CURRENT ISOLATION 특수 레지스터가 현재 세션 내에서 발행된 동적 SQL문에 대한 분리 수준을 지정하는 값으로 설정됩니다. 일단 설정되면, CURRENT ISOLATION 특수 레지스터는 어떤 패키지에서 명령문을 발행했든지 상관없이, 세션 내에서 컴파일 된 후속 동적 SQL문에 대한 분리 수준을 제공합니다. 이 분리 수준은 세션이 종료되거나 SET CURRENT ISOLATION...RESET문이 발행될 때까지 유효합니다.

- **프리컴파일 또는 바인드 시간에:**

  지원되는 컴파일된 언어로 작성된 응용프로그램의 경우, PREP 또는 BIND 명령의 ISOLATION 옵션을 사용하십시오. 또한 sqlaprep 또는 sqlabndx API를 사용하여 분리 수준을 지정할 수 있습니다.

  - 프리컴파일 시간에 바인드 파일을 작성할 경우, 분리 수준이 바인드 파일에 저장됩니다. 바인드 시간에 분리 수준을 지정하지 않을 경우, 디폴트는 프리컴파일 동안 사용된 분리 수준입니다.
  - 분리 수준을 지정하지 않을 경우, 커서 안정성(CS)의 디폴트 수준이 사용됩니다.

  패키지의 분리 수준을 판별하려면, 다음 쿼리를 실행하십시오.

```
select isolation from syscat.packages
  where pkgname = 'pkgname'
    and pkgschema = 'pkgschema'
```

  여기서 *pkgname*은 패키지의 규정되지 않은 이름이고 *pkgschema*는 패키지의 스키마 이름입니다. 이러한 이름은 둘 다 대문자로 지정되어야 합니다.

- **런타임에서 JDBC 또는 SQLJ를 사용하여 작업할 때:**

  주: JDBC 및 SQLJ는 DB2 서버에서 CLI를 통해 구현되며, 이는 `db2cli.ini` 설정이 JDBC 및 SQLJ를 사용하여 작성 및 실행되는 내용에 영향을 줄 수도 있음을 의미합니다.

SQLJ에서 패키지를 작성하려면(그리고 해당 분리 수준 지정), SQLJ 프로파일 커스터마이저(db2sqljcustomize 명령)를 사용하십시오.

- **런타임의 CLI 또는 ODBC에서:**

  CHANGE ISOLATION LEVEL 명령을 사용하십시오. DB2 콜 레벨 인터페이스(CLI)를 사용하여 CLI 구성의 일부로 분리 수준을 변경할 수 있습니다. 런타임에서, SQL_ATTR_TXN_ISOLATION 속성과 함께 SQLSetConnectAttr 기능을 사용하여 *ConnectionHandle* 인수에 의해 참조되는 현재 연결에 대한 트랜잭션 분리 수준을 설정하십시오. `db2cli.ini` 파일에서 TXNISOLATION 키워드를 사용하십시오.

- **REXX™를 지원하는 데이터베이스 서버에서:**

  데이터베이스가 작성되면, REXX에서 SQL에 대한 다양한 분리 수준을 지원하는 여러 바인드 파일이 데이터베이스에 바인드됩니다. 다른 명령행 처리기(CLP) 패키지 또한 데이터베이스가 작성될 때 데이터베이스에 바인드됩니다.

  REXX 및 CLP는 디폴트 CS 분리 수준을 사용하여 데이터베이스에 연결됩니다. 이 분리 수준을 변경해도 연결 상태가 변경되지 않습니다.

  REXX 응용프로그램에서 사용되고 있는 분리 수준을 판별하려면 SQLISL 사전 정의 REXX 변수의 값을 확인하십시오. 이 값은 CHANGE ISOLATION LEVEL 명령이 실행될 때마다 갱신됩니다.

현재 커미트된 시맨틱이 동시성을 향상시킴

행 레벨 잠금, 특히 해당 문제를 예방하기 위해 설계되지 않은 응용프로그램의 CS 분리 수준에서 잠금 시간종료 및 교착 상태가 발생할 수 있습니다. 몇몇 높은 처리량을 가진 데이터베이스 응용프로그램에서는 트랜잭션 처리 동안 발행된 잠금에 대한 대기를 허용할 수 없고, 몇몇 응용프로그램에서는 커미트되지 않은 데이터의 처리를 허용할 수 없지만 읽기 트랜잭션에 대한 비블로킹 동작이 여전히 필요합니다.

새 *현재 커미트된* 시맨틱에서는 앞의 경우에서와 같이 커미트된 데이터만 리턴되지만, 이제 판독기는 기록기가 행 잠금을 해제하기를 기다리지 않습니다. 대신, 데이터를 읽을 사용자는 현재 커미트된 버전을 따르는 데이터(즉, 쓰기 조작이 시작되기 전의 데이터)를 리턴합니다.

현재 커미트된 시맨틱은 새 데이터베이스에 대해 디폴트로 켜집니다. 이것은 응용프로그램에서 새 동작을 이용할 수 있도록 해 주고, 응용프로그램 자체에 대한 변경이 필요하지 않습니다. 새 데이터베이스 구성 매개변수 **cur_commit**를 사용하여 이 동작을 겹쳐쓸 수 있습니다. 이것은, 예를 들어, 내부 논리를 동기화하기 위해 기록기에 대한 블로킹이 필요한 응용프로그램의 경우에 유용할 수 있습니다.

마찬가지로, 해당 내부 논리를 동기화하기 위해 응용프로그램에서 기록기 블로킹이 필요한 경우 업그레이드된 데이터베이스에서 **cur_commit**가 디폴트로 사용되지 않고, 필요한 경우 나중에 이 매개변수를 켤 수 있습니다.

현재 커미트된 시맨틱은 제한조건을 평가하거나 강제 적용하는 데 사용되는 내부 스캔 또는 카탈로그 테이블과 관련되지 않은 읽기 전용 스캔에만 적용됩니다. 현재 커미트된 시맨틱은 스캔 레벨에서 결정되므로, 기록기의 액세스 플랜에 현재 커미트된 스캔이 포함될 수도 있습니다. 예를 들어, 읽기 전용 서브쿼리에 대한 스캔이 현재 커미트된 시맨틱을 포함할 수 있습니다. 현재 커미트된 시맨틱은 분리 수준 시맨틱을 따르기 때문에, 현재 커미트된 시맨틱 하에서 실행되는 응용프로그램은 계속해서 분리 수준을 준수합니다.

현재 커미트된 시맨틱은 기록기용으로 증가된 로그 스페이스를 필요로 합니다. 트랜잭션 동안 데이터 행의 첫 번째 갱신을 로깅하기 위해 추가 스페이스가 필요합니다. 이 데이터는 행의 현재 커미트된 이미지를 검색하는 데 필요합니다. 워크로드에 따라, 이것은 사용되는 전체 로그 스페이스에 대수롭지 않은 영향을 미치거나 상당한 영향을 미칠 수 있습니다. 추가 로그 스페이스에 대한 요구사항은 **cur_commit**가 사용되지 않는 경우 적용되지 않습니다.

예

현재 커미트된 시맨틱 하에서 교착 상태가 회피되는 다음 시나리오를 고려해보십시오. 이 시나리오에서는, 두 개의 응용프로그램이 두 개의 개별 테이블을 갱신하지만 아직 커미트하지는 않습니다. 그런 다음 각 응용프로그램이 다른 응용프로그램이 갱신한 테이블에서 읽기를 시도합니다(읽기 전용 커서를 사용).

단계	응용프로그램 **A**	응용프로그램 **B**
1	update T1 set col1 = ? where col2 = ?	update T2 set col1 = ? where col2 = ?
2	select col1, col3, col4 from T2 where col2 >= ?	select col1, col5, from T1 where col5 = ? and col2 = ?
3	commit	commit

현재 커미트된 시맨틱이 없을 경우, 커서 안정성 분리 수준 하에서 실행되는 이러한 응용프로그램은 응용프로그램 중 하나가 실패하게 하는 교착 상태를 만들 수도 있습니다. 각 응용프로그램에서 다른 응용프로그램에 의해 갱신되고 있는 데이터를 읽어야 할 경우 이렇게 될 수 있습니다.

현재 커미트된 시맨틱 하에서는, 2단계의 쿼리(응용 프로그램 중 하나에 대한)에서 다른 응용프로그램에 의해 현재 갱신되고 있는 데이터를 필요로 할 경우, 해당 응용프로그램이 잠금이 해제되기를 대기하지 않고, 교착 상태를 불가능하게 만듭니다. 대신 이전에 커미트된 버전의 데이터가 배치되어 사용됩니다.

커미트되지 않은 삽입 무시 옵션

DB2_SKIPINSERTED 레지스트리 변수는 커서 안정성(CS) 또는 읽기 안정성(RS) 분리 수준을 사용하는 명령문에 대해 커미트되지 않은 데이터 삽입을 무시할 수 있는지 여부를 제어합니다.

커미트되지 않은 삽입은 **DB2_SKIPINSERTED** 레지스트리 변수 값에 따라 두 가지 방법 중 하나로 처리됩니다.

- 값이 ON인 경우, DB2 서버에서 커미트되지 않은 삽입을 무시하고, 이는 많은 경우에 동시성을 향상시킬 수 있으며 대부분의 응용프로그램에서 선호되는 동작입니다. 커미트되지 않은 삽입이 아직 발생하지 않은 것처럼 처리됩니다.
- 값이 해제(OFF)(디폴트)인 경우, DB2 서버에서 삽입 조작이 완료(커미트 또는 롤백)될 때까지 대기한 후 데이터를 알맞게 처리합니다. 이 방법은 특정 경우에 적절합니다. 예를 들어, 다음과 같습니다.
  - 두 개의 응용 프로그램에서 테이블을 사용하여 첫 번째 응용프로그램에서 테이블에 데이터를 삽입하고 두 번째 응용프로그램에서 해당 데이터를 읽는 방식으로 서로 간에 데이터를 전달한다고 가정하십시오. 표시된 순서로 두 번째 응용프로그램에 의해 데이터가 처리되어야 합니다. 예를 들어, 읽을 다음 행을 첫 번째 응용프로그램에서 삽입할 경우 두 번째 응용프로그램에서 삽입 조작이 커미트될 때까지 대기해야 합니다.
  - 응용프로그램이 데이터를 삭제한 후 데이터의 새 이미지를 삽입하여 UPDATE문을 회피합니다.

잠금 지연을 통해 커미트되지 않은 데이터 평가

동시성을 향상시키기 위해 일부 경우 데이터베이스 관리 프로그램은 행이 쿼리의 술어를 충족시킬 때까지 CS 또는 RS 분리 스캔에 대한 행 잠금 지연을 허용합니다.

기본적으로, 테이블 또는 인덱스 스캔 동안 행 레벨 잠금이 수행될 경우, 데이터베이스 관리 프로그램은 행이 쿼리의 술어를 충족시키는지 여부를 판별하기 전에 커미트 상태를 알 수 없는 각 스캔된 행을 잠급니다.

이러한 스캔의 동시성을 향상시키려면, 커미트되지 않은 데이터에 대해 술어 평가가 발생할 수 있도록 **DB2_EVALUNCOMMITTED** 레지스트리 변수를 사용하십시오. 커미트되지 않은 갱신을 포함하는 행은 쿼리를 충족시키지 않을 수도 있지만, 트랜잭션이 완료되는 후까지 술어 평가가 지연된 경우, 행이 실제로 쿼리를 충족시킬 수도 있습니다.

테이블 스캔 동안 커미트되지 않은 삭제된 행을 건너뛰고, **DB2_SKIPDELETED** 레지스트리 변수가 사용되는 경우 데이터베이스 관리 프로그램에서 인덱스 스캔 동안 삭제된 키를 건너뜁니다.

DB2_EVALUNCOMMITTED 레지스트리 변수 설정은 동적 SQL 또는 XQuery문의 경우 컴파일 시간에 적용되고, 정적 SQL 또는 XQuery문의 경우 바인드 시간에 적용됩니다. 즉, 런타임에서 레지스트리 변수가 사용 가능하다 하더라도 바인드 시간에 **DB2_EVALUNCOMMITTED**가 사용 가능하지 않으면 잠금 회피 전략이 전개되지 않습니다. 바인드 시간에 레지스트리 변수가 사용 가능하지만 런타임에서 사용 가능하지 않은 경우에는 잠금 회피 전략이 계속 적용됩니다. 정적 SQL 또는 XQuery문의 경우, 패키지가 리바인드될 경우, 바인드 시간에 사용되는 레지스트리 변수 설정이 적용되는 설정입니다. 정적 SQL 또는 XQuery문의 내재된 리바인드에서는 **DB2_EVALUNCOMMITTED** 레지스트리 변수의 현재 설정을 사용합니다.

다양한 액세스 플랜에 대한 커미트되지 않은 평가 적용 가능성

표 6. RID 인덱스만 액세스

술어	커미트되지 않은 평가
없음	아니오
SARGable	예

표 7. 데이터만 액세스(관계형 또는 지연된 RID 목록)

술어	커미트되지 않은 평가
없음	아니오
SARGable	예

표 8. RID 인덱스 + 데이터 액세스

술어		커미트되지 않은 평가	
인덱스	데이터	인덱스 액세스	데이터 액세스
없음	없음	아니오	아니오
없음	SARGable	아니오	아니오
SARGable	없음	예	아니오
SARGable	SARGable	예	아니오

표 9. 블록 인덱스 + 데이터 액세스

술어		커미트되지 않은 평가	
인덱스	데이터	인덱스 액세스	데이터 액세스
없음	없음	아니오	아니오
없음	SARGable	아니오	예
SARGable	없음	예	아니오
SARGable	SARGable	예	예

예

다음 예에서는 디폴트 잠금 동작과 커미트되지 않은 평가 동작을 비교합니다. 테이블은 샘플 데이터베이스의 ORG 테이블입니다.

```
DEPTNUMB DEPTNAME       MANAGER DIVISION   LOCATION
-------- -------------- ------- ---------- -------------
      10 Head Office        160 Corporate  New York
      15 New England         50 Eastern    Boston
      20 Mid Atlantic        10 Eastern    Washington
      38 South Atlantic      30 Eastern    Atlanta
      42 Great Lakes        100 Midwest    Chicago
      51 Plains             140 Midwest    Dallas
      66 Pacific            270 Western    San Francisco
      84 Mountain           290 Western    Denver
```

디폴트 커서 안정성(CS) 분리 수준에서 다음 트랜잭션이 발생합니다.

표 10. CS 분리 수준에서 ORG 테이블에 대한 트랜잭션

세션 **1**	세션 **2**
`connect to sample`	`connect to sample`
`+c update org set  deptnumb=5 where manager=160`	
	`select * from org where deptnumb >= 10`

세션 1의 커미트되지 않은 UPDATE문은 테이블의 첫 번째 행에서 배타적 잠금을 보유하여, 세션 1에서 갱신되고 있는 행이 현재 세션 2의 쿼리를 충족시키지 않더라도, 세션 2의 쿼리가 결과 세트를 리턴하지 못하게 합니다. CS 분리 수준은 커서가 쿼리에서 액세스하는 행에 위치한 동안 해당 행이 잠겨야 하도록 지정합니다. 세션 2에서는 세션 1에서 잠금을 해제할 때까지 첫 번째 행에 대한 잠금을 얻을 수 없습니다.

먼저 술어를 평가한 후 행을 잠그는 커미트되지 않은 평가 기능을 사용하면 세션 2에서 잠금을 대기하지 않아도 됩니다. 그러면 세션 2의 쿼리가 테이블의 첫 번째 행을 잠그려고 시도하지 않게 되므로 응용프로그램 동시성이 증가됩니다. 이것은 또한 세션 1의 커미트되지 않은 값 `deptnumb=5`에 대해 세션 2에서 술어 평가가 발생한다는 것을 의미합니다. 세션 1의 갱신 롤백이 세션 2의 쿼리를 충족시킨다는 사실에도 불구하고 세션 2의 쿼리가 해당 결과 세트에서 첫 번째 행을 생략합니다.

조작 순서가 반대로 되는 경우, 커미트되지 않은 평가 기능을 통해 여전히 동시성이 향상될 수 있습니다. 디폴트 잠금 동작에서, 세션 2는 세션 1 UPDATE문이 세션 2 쿼리에 의해 잠긴 행을 변경하지 않더라도 먼저 행 잠금을 수행하여 세션 1에서 검색된 갱신이 실행되지 않도록 합니다. 세션 1에서 검색된 갱신이 먼저 행 검사를 시도한 후 자격이 부여된 경우에만 행을 잠그는 경우, 세션 1 쿼리는 비블로킹됩니다.

제한사항

- **DB2_EVALUNCOMMITTED** 레지스트리 변수가 사용 가능해야 합니다.
- 분리 수준은 CS 또는 RS여야 합니다.
- 행 레벨 잠금이 적용됩니다.
- SARGable 평가 술어가 존재합니다.

- 커미트되지 않은 평가가 시스템 카탈로그 테이블의 스캔에 적용 가능하지 않습니다.
- 다차원 클러스터링(MDC) 테이블의 경우, 인덱스 스캔에 대해 블록 레벨 잠금을 지연할 수 있지만, 테이블 스캔에 대해서는 블록 레벨 잠금을 지연할 수 없습니다.
- 인플레이스 테이블 재구성을 실행 중인 테이블에서는 잠금 지연이 발생하지 않습니다.
- Iscan-Fetch 플랜의 경우, 데이터 액세스에 대해 행 레벨 잠금이 지연되지 않습니다. 반대로, 테이블의 행으로 이동하기 전 인덱스 액세스 동안 행이 잠깁니다.
- 테이블 스캔 동안 삭제된 행은 무조건 건너뛰지만, 삭제된 인덱스 키는 **DB2_SKIPDELETED** 레지스트리 변수가 사용되는 경우에만 건너뜁니다.

최적의 성능을 위한 쿼리 쓰기 및 조정

DB2 데이터베이스 성능에 대한 SQL문의 영향을 최소화할 수 있는 몇 가지 방법이 있습니다.

다음을 수행하여 이 영향을 최소화할 수 있습니다.

- DB2 옵티마이저에서 보다 쉽게 최적화할 수 있는 SQL문 쓰기. DB2 옵티마이저는 비균등 조인 술어, 조인 컬럼의 데이터 유형 불일치, 불필요한 외부 조인 및 기타 복합 검색 조건을 포함하는 SQL문을 효과적으로 실행하지 못할 수도 있습니다.
- DB2 데이터베이스를 올바르게 구성하여 DB2 최적화 기능 이용. DB2 옵티마이저는 사용자가 정확한 카탈로그 통계를 지니고 워크로드에 대한 최상의 최적화 클래스를 선택할 경우 최적의 쿼리 액세스 플랜을 선택할 수 있습니다.
- DB2 Explain 기능을 사용하여 잠재적 쿼리 액세스 플랜을 검토하고 최상의 성능을 위해 쿼리를 조정하는 방법 판별

우수 사례는 일반 워크로드, 웨어하우스 워크로드 및 SAP 워크로드에 적용됩니다.

응용프로그램이 작성된 후 특정 쿼리 성능 문제를 처리하기 위한 많은 방법이 있지만, DB2 데이터베이스 성능 향상을 돕기 위해 적절한 기본 쓰기 및 조정 사례가 조기에 광범위하게 적용될 수 있습니다.

쿼리 성능은 일회성 고려사항이 아닙니다. 응용프로그램 전개 수명 주기의 설계, 개발 및 프로덕션 단계에 걸쳐 이를 고려해야 합니다.

SQL은 매우 유연한 언어입니다. 즉, 동일한 올바른 결과를 얻는 방법이 많이 있습니다. 이러한 유연성은 또한 DB2 옵티마이저의 강점을 이용하는 데 있어 일부 쿼리가 다른 쿼리보다 더 낫다는 것을 의미합니다.

쿼리 실행 동안 DB2 옵티마이저는 각 SQL문에 대한 쿼리 액세스 플랜을 선택합니다. 옵티마이저는 많은 대체 액세스 플랜의 실행 비용을 모델링하고 최소 계산된 비용을 지닌 액세스 플랜을 선택합니다. 쿼리에 많은 복합 검색 조건이 포함된 경우, DB2 옵티마이저는 일부 경우에 술어를 다시 쓸 수 있지만, 그럴 수 없는 경우도 있습니다.

비즈니스 인텔리전스(BI) 응용프로그램에 사용되는 것과 같은, 복합 쿼리의 경우 SQL 문을 준비하거나 컴파일하기 위한 시간이 길 수 있습니다. 데이터베이스를 올바르게 설계 및 구성하면 명령문 컴파일 시간을 최소화하는 데 도움이 될 수 있습니다. 여기에는 올바른 최적화 클래스를 선택하는 것과 기타 레지스트리 변수를 올바르게 설정하는 것이 포함됩니다.

옵티마이저는 또한 정확한 액세스 플랜 결정을 위해 정확한 입력을 필요로 합니다. 즉, 정확한 통계를 모으고, 통계 뷰 및 컬럼 그룹 통계와 같은 고급 통계 기능을 잠재적으로 사용해야 합니다.

DB2 도구, 특히 DB2 Explain 기능을 사용하여 쿼리를 조정할 수 있습니다. DB2 컴파일러는 정적 또는 동적 쿼리의 환경 및 액세스 플랜에 대한 정보를 캡처할 수 있습니다. 이 캡처된 정보를 사용하여 개별 명령문이 실행되는 방법을 이해하면, 해당 명령문 및 데이터베이스 관리 프로그램 구성을 조정하여 성능을 향상시킬 수 있습니다.

SQL문 쓰기

SQL은 구문상으로는 다르지만 의미구조적으로는 동등하게 관계 표현식을 지정할 수 있게 해 주는 강력한 언어입니다. 그러나 일부 의미구조적으로 동등한 변형이 다른 변형보다 최적화하기가 더 간편합니다. DB2 옵티마이저에는 강력한 쿼리 재작성 기능이 있지만, 일부 경우에는 가장 최적 양식으로 SQL문을 다시 쓸 수 없을 수도 있습니다.

특정 SQL 구문은 쿼리 옵티마이저가 고려하는 액세스 플랜을 제한할 수 있으며, 가능하면 이러한 구문을 피하거나 교체해야 합니다.

검색 조건에서 복합 표현식 회피:

표현식으로 인해 옵티마이저가 카탈로그 통계를 사용하여 정확한 선택 빈도를 추정할 수 없는 검색 조건에서는 복합 표현식을 사용하지 않도록 하십시오.

표현식은 또한 술어를 적용하는 데 사용될 수 있는 액세스 플랜의 선택을 제한할 수도 있습니다. 최적화의 쿼리 재작성 단계 동안, 옵티마이저는 정확한 선택 빈도를 추정할 수 있도록 많은 표현식을 다시 쓸 수 있습니다. 옵티마이저가 모든 가능성을 처리할 수는 없습니다.

표현식에서 Join 술어 회피:

표현식에서 Join 술어를 사용하면 조인 메소드가 중첩된 루프로 제한됩니다.

또한 카디널리티(cardinality) 추정이 부정확해질 수도 있습니다. 표현식에서 조인의 몇 가지 예는 다음과 같습니다.

```
WHERE SALES.PRICE * SALES.DISCOUNT = TRANS.FINAL_PRICE
WHERE UPPER(CUST.LASTNAME) = TRANS.NAME
```

로컬 술어의 컬럼에 대한 표현식 회피:

로컬 술어의 컬럼에 대해 표현식을 적용하는 대신 표현식의 역을 사용하십시오.

다음 예를 고려해보십시오.

```
XPRESSN(C) = 'constant'
INTEGER(TRANS_DATE)/100 = 200802
```

이러한 명령문을 다음과 같이 다시 쓸 수 있습니다.

```
C = INVERSEXPRESSN('constant')
TRANS_DATE BETWEEN 20080201 AND 20080229
```

컬럼에 대해 표현식을 적용하면 인덱스 시작 및 중지 키의 사용이 방지되어, 부정확한 선택 빈도 추정으로 이어질 수 있고, 쿼리 실행 시간에 추가 처리가 요구됩니다.

이러한 표현식은 또한 컬럼이 동등할 때 인식, 컬럼을 상수로 교체 및 많아야 하나의 행이 리턴될 때 인식과 같은 쿼리 재작성 최적화를 막습니다. 많아야 하나의 행이 리턴됨이 증명될 수 있어야 추가 최적화가 가능하므로, 상실된 최적화 기회가 더 악화됩니다. 다음 쿼리를 고려해보십시오.

```
SELECT LASTNAME, CUST_ID, CUST_CODE FROM CUST
WHERE (CUST_ID * 100) + INT(CUST_CODE) = 123456 ORDER BY 1,2,3
```

이것을 다음과 같이 다시 쓸 수 있습니다.

```
SELECT LASTNAME, CUST_ID, CUST_CODE FROM CUST
WHERE CUST_ID = 1234 AND CUST_CODE = '56' ORDER BY 1,2,3
```

CUST_ID에 정의된 고유 인덱스가 있는 경우, 다시 쓴 버전의 쿼리를 통해 쿼리 옵티마이저는 많아야 하나의 행이 리턴되는 것을 인식할 수 있습니다. 이는 불필요한 SORT 조작의 사용을 막아줍니다. 또한 CUST_ID 및 CUST_CODE 컬럼이 1234와 '56'으로 대체될 수 있게 하여 데이터 또는 인덱스 페이지에서 값을 복사하지 않게 해 줍니다. 마지막으로, CUST_ID의 술어가 인덱스 시작 또는 중지 키로 적용될 수 있게 해 줍니다.

표현식이 술어에 있는 경우에는 항상 분명하지 않을 수도 있습니다. 이것은 표현식에 의해 뷰 컬럼이 정의된 경우 뷰를 참조하는 쿼리에서 종종 발생할 수 있습니다. 예를 들어, 다음 뷰 정의 및 쿼리를 고려해보십시오.

```
CREATE VIEW CUST_V AS
  (SELECT LASTNAME, (CUST_ID * 100) + INT(CUST_CODE) AS CUST_KEY
    FROM CUST)

SELECT LASTNAME FROM CUST_V WHERE CUST_KEY = 123456
```

쿼리 옵티마이저는 뷰 정의와 쿼리를 병합하여, 다음 쿼리를 만듭니다.

```
SELECT LASTNAME FROM CUST
WHERE (CUST_ID * 100) + INT(CUST_CODE) = 123456
```

이것은 위의 예에서 설명된 동일한 문제가 있는 술어입니다. Explain 기능을 사용해 뷰 병합의 결과를 관찰하여 최적화된 SQL을 표시할 수 있습니다.

역 함수를 표현하기 어려운 경우, 생성된 컬럼 사용을 고려하십시오. 예를 들어, LASTNAME IN ('Woo', 'woo', 'WOO', 'WOo',...)에 의해 표현된 기준에 맞는 성을 찾으려는 경우, 다음과 같이 생성된 컬럼 UCASE(LASTNAME) = 'WOO'를 작성할 수 있습니다.

```
CREATE TABLE CUSTOMER (
  LASTNAME VARCHAR(100),
  U_LASTNAME VARCHAR(100) GENERATED ALWAYS AS (UCASE(LASTNAME))
)

CREATE INDEX CUST_U_LASTNAME ON CUSTOMER(U_LASTNAME)
```

Linux, UNIX 및 Windows용 DB2 Database 버전 9.5 FixPack 1의 대소문자 구분 안함 검색에 대한 지원은 이 특정 예에서의 상황을 해결하기 위해 설계되었습니다. UCA500R1 조합 이름에서 _Sx 속성을 사용하여 조합의 강도를 제어할 수 있습니다. 예를 들어, UCA500R1_LFR_S1은 대소문자와 액센트를 무시하는 프랑스어 조합입니다.

조인 컬럼에서 데이터 유형 불일치 회피:

일부 경우, 데이터 유형 불일치로 인해 해시 조인을 사용하지 못합니다.

해시 조인에는 다른 조인 메소드에는 없는 Join 술어에 대한 몇몇 추가 제한사항이 있습니다. 특히, 조인 컬럼의 데이터 유형이 정확히 동일해야 합니다. 예를 들어, 하나의 조인 컬럼이 FLOAT이고 다른 조인 컬럼이 REAL인 경우, 해시 조인이 지원되지 않습니다. 또한 조인 컬럼 데이터 유형이 CHAR, GRAPHIC, DECIMAL 또는 DECFLOAT인 경우, 길이가 동일해야 합니다.

옵티마이저 추정을 변경하기 위해 술어에서 no-op 표현식 회피:

COALESCE(X, X) = X 양식의 "no-op" coalesce() 술어는 이를 사용하는 쿼리의 플랜에 추정 오류를 가져옵니다. 현재 DB2 쿼리 컴파일러에는 해당 술어를 분석하고 모든 행이 실제로 이를 충족하는지 판별하는 기능이 없습니다.

그 결과, 술어가 쿼리 플랜의 일부에서 나오는 행 추정 수를 인위적으로 줄입니다. 이 더 작은 행 추정은 다른 후보 플랜 간 상대적 추정이 변경되기 때문에 때때로 다른 플랜이 선택되는 결과를 가져오고, 일반적으로 나머지 쿼리 플랜에 대한 행 및 비용 추정을 감소시킵니다.

이와 같이 아무 것도 안하는 술어가 때때로 쿼리 성능을 향상시킬 수 있는 이유는 무엇일까요? 『no-op』 coalesce() 술어 추가는 최적 성능을 방해하는 그 밖의 무언가를 마스크하는 오류를 가져옵니다.

일부 성능 향상 도구가 수행하는 것은 억지 테스트: 도구는 다양한 컬럼에서 작동하는 쿼리의 다양한 위치로 술어를 반복적으로 들여와 오류를 가져옴으로써 더 나은 수행 플

랜에 걸림돌이 되는 경우를 찾으려고 노력합니다. 이는 ″no-op″ 술어를 쿼리에 핸드 코딩하는 쿼리 개발자도 해당됩니다. 일반적으로, 개발자는 데이터에 대한 약간의 통찰력을 갖고 술어의 배치를 안내합니다.

쿼리 성능을 향상시키기 위해 이 방법을 사용하는 것은 근본 원인을 해결하지 않는 단기 솔루션이며 다음과 같은 의미를 지닐 수도 있습니다.

- 성능 향상을 위한 잠재적 영역이 숨겨집니다.
- DB2 쿼리 컴파일러가 결국 술어를 더 낫게 처리할 수도 있고, 또는 다른 무작위 인수가 술어에 영향을 미칠 수도 있기 때문에, 이 일시적인 해결책이 영구 성능 향상을 제공한다는 보장이 없습니다.
- 동일한 근본 원인에 의해 영향을 받는 다른 쿼리가 있을 수도 있고, 그 결과 전반적인 시스템의 성능이 피해를 입을 수도 있습니다.

이러한 우수 사례 권장사항을 따랐지만 여전히 최적 성능에 못 미친다고 생각되면, 『no-op』 술어를 도입하지 않고 DB2 옵티마이저에 명시적 최적화 지침을 제공할 수 있습니다. 『최적화 프로파일 및 지침』을 참조하십시오.

비등식 Join 술어 회피:

조인 메소드가 중첩된 루프로 제한되기 때문에 등식이 아닌 비교 연산자를 사용하는 Join 술어를 피해야 합니다.

또한 옵티마이저가 Join 술어에 대한 정확환 선택 빈도 추정을 계산할 수 없을 수도 있습니다. 그러나 비등식 Join 술어를 항상 피할 수는 없습니다. 이러한 술어가 필요한 경우, Join 술어는 중첩된 루프 조인 내부 테이블에 적용되므로 어느 한쪽 테이블에 적절한 인덱스가 존재하는지 확인하십시오.

비등식 Join 술어의 한 가지 일반적인 예는 다른 시점의 차원 상태를 정확히 반영하도록 스타 스키마의 차원 데이터를 버전 지정해야 하는 경우입니다. 이를 종종 *느리게 변경되는 차원*이라고 합니다. 느리게 변경되는 차원의 한 가지 유형으로는 각 차원 행에 대한 유효한 시작 및 종료 날짜 포함이 있습니다. 사실 테이블과 차원 테이블 간의 조인에서는 사실과 연관된 날짜가 차원의 시작 및 종료 날짜 범위에 속하는지 점검할 뿐만 아니라 차원 기본 키에서 조인해야 합니다. 이를 종종 *유형 6 느리게 변경되는 차원*이라고 합니다. 일부 사실 트랜잭션 날짜에 의해 차원 버전을 추가로 규정하기 위해 사실 테이블로 다시 돌아가는 조인 범위는 광범위할 수 있습니다. 예를 들어, 다음과 같습니다.

```
SELECT...
  FROM PRODUCT P, SALES F
  WHERE
    P.PROD_KEY = F.PROD_KEY AND
    F.SALE_DATE BETWEEN P.START_DATE AND
    P.END_DATE
```

이 경우, (F.PROD_KEY, F.SALE_DATE)에 인덱스가 있는지 확인하십시오.

이 시나리오에서 옵티마이저가 더 나은 선택 빈도 추정을 계산하도록 돕기 위해 통계 뷰 작성을 고려하십시오. 예를 들어, 다음과 같습니다.

```
CREATE STATISTICAL VIEW V_PROD_FACT AS
  SELECT P.*
    FROM PRODUCT P, SALES F
    WHERE
      P.PROD_KEY = F.PROD_KEY AND
      F.SALE_DATE BETWEEN P.START_DATE AND
      P.END_DATE

ALTER VIEW V_PROD_FACT ENABLE QUERY OPTIMIZATION

RUNSTATS ON TABLE DB2USER.V_PROD_FACT WITH DISTRIBUTION
```

쿼리 블록에 비등식 Join 술어가 있는 경우, 허브 조인 및 인덱스 ANDing이 있는 스타 조인과 같은 전문화된 스타 스키마 조인은 고려되지 않습니다(『쿼리가 스타 스키마 조인에 필요한 기준을 충족하는지 확인』 참조).

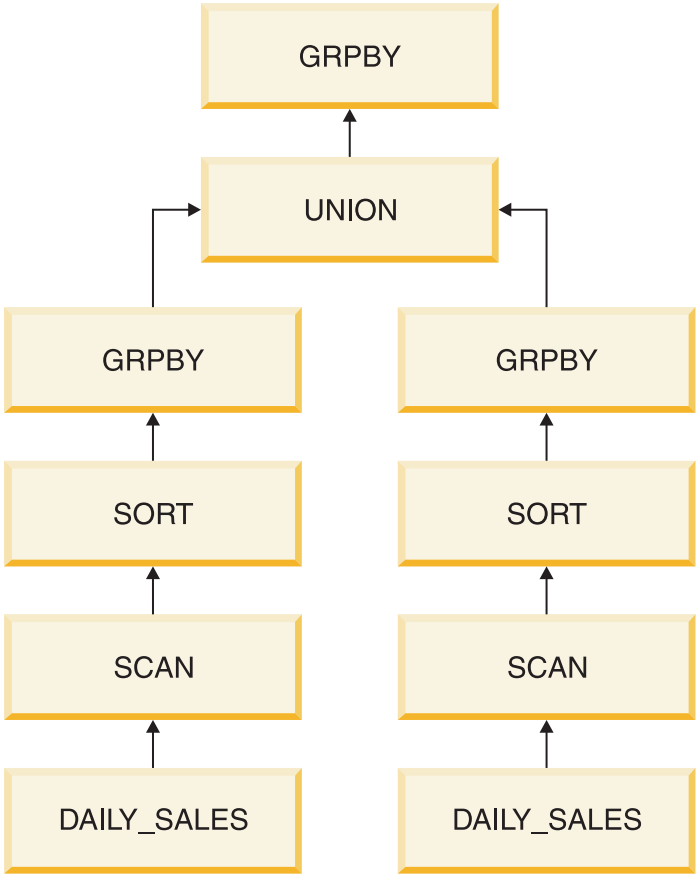
DISTINCT 키워드를 사용한 다중 집계 회피:

동일한 subselect에서 다중 DISTINCT 집계를 수행하는 쿼리를 사용하지 않도록 하십시오. 실행하는 데 비용이 많이 듭니다.

다음 예를 고려해보십시오.

```
SELECT SUM(DISTINCT REBATE), AVG(DISTINCT DISCOUNT)
  FROM DAILY_SALES
  GROUP BY PROD_KEY
```

고유 REBATE 값과 고유 DISCOUNT 값 세트를 판별하려면, PROD_KEY 테이블의 입력 스트림을 두 번 정렬해야 합니다. 이 쿼리에 대한 쿼리 액세스 플랜은 다음과 같을 수 있습니다.



옵티마이저는 각각에 대해 DISTINCT 키워드를 지정하여 원래 쿼리를 개별 집계에 다시 쓴 다음 UNION 키워드를 사용하여 다중 집계를 조합합니다. 내부적으로 다시 쓴 명령문:

```
SELECT Q8.MAXC0, (Q8.MAXC1 / Q8.MAXC2)
FROM
  (SELECT MAX(Q7.C0) AS MAXC0, MAX(Q7.C1) AS MAXC1, MAX(Q7.C2) AS MAXC2
    FROM
      (SELECT SUM(DISTINCT Q2.REBATE) AS C0, CAST(NULL AS INTEGER) AS C1,
        0 AS C2, Q2.PROD_KEY
      FROM
        (SELECT Q1.PROD_KEY, Q1.REBATE
          FROM DB2USER.DAILY_SALES AS Q1) AS Q2
      GROUP BY Q2.PROD_KEY
      UNION ALL
      SELECT CAST(NULL AS INTEGER) AS C0, SUM(DISTINCT Q5.DISCOUNT) AS C1,
          COUNT(DISTINCT Q5.DISCOUNT) AS C2, Q5.PROD_KEY
        FROM
          (SELECT Q4.PROD_KEY, Q4.DISCOUNT
            FROM DB2USER.DAILY_SALES AS Q4) AS Q5
        GROUP BY Q5.PROD_KEY) AS Q7
      GROUP BY Q7.PROD_KEY) AS Q8
```

다중 DISTINCT 집계를 피할 수 없는 경우, ENHANCED_MULTIPLE_DISTINCT 옵션과 함께 **DB2_EXTENDED_OPTIMIZATION** 레지스트리 변수 사용을 고려하십시오. 이 옵션을 사용하면 다중 고유 집계에 대한 입력 스트림이 한 번 읽힌 다음

UNION의 각 가지에서 재사용됩니다. 이 옵션은 데이터베이스 파티션 수에 대한 프로세서의 비율이 낮은(예를 들어, 비율이 1보다 작거나 같은) 유형의 쿼리 성능을 향상시킬 수도 있습니다. 이 설정은 대칭 멀티프로세서(SMP) 없는 파티션된 데이터베이스 환경에서 사용되어야 합니다. 이 최적화 확장은 일부 환경에서 쿼리 성능을 향상시키지 않을 수도 있습니다. 개별 쿼리 성능 향상을 판별하려면 테스트를 수행해야 합니다.

불필요한 외부 조인 회피:

특정 쿼리의 시맨틱에는 외부 조인(왼쪽, 오른쪽 또는 전체)이 필요합니다. 그러나 쿼리 시맨틱에 외부 조인이 필요하지 않고 쿼리가 불일치 데이터를 처리하는 데 사용되는 경우, 해당 근본 원인에서 불일치 데이터 문제점을 다루는 것이 가장 좋습니다.

예를 들어, 스타 스키마가 있는 데이터 마트에서, 데이터 일관성 문제로 인해 사실 테이블이 트랜잭션에 대한 행을 포함하지만 일부 차원에 대해 일치하는 상위 차원 행을 포함하지 않을 수도 있습니다. 이는 추출, 변환 및 로드(ETL) 프로세스에서 몇몇 이유로 일부 비즈니스 키를 조정할 수 없기 때문에 발생할 수 있습니다. 이 시나리오에서, 사실 테이블 행은 상위가 없더라도 리턴될 수 있도록 차원과 왼쪽 외부 조인되어 있습니다. 예를 들어, 다음과 같습니다.

```
SELECT...
  FROM DAILY_SALES F
    LEFT OUTER JOIN CUSTOMER C ON F.CUST_KEY = C.CUST_KEY
    LEFT OUTER JOIN STORE S ON F.STORE_KEY = S.STORE_KEY
  WHERE
    C.CUST_NAME = 'SMITH'
```

왼쪽 외부 조인은 전문화된 스타 스키마 조인 액세스 방법을 포함하여 많은 최적화를 막을 수 있습니다. 그러나 일부 경우에 쿼리 옵티마이저에 의해 왼쪽 외부 조인이 내부 조인에 자동으로 다시 작성될 수 있습니다. 이 예에서는, 술어 `C.CUST_NAME = 'SMITH'`가 이 컬럼에서 널(Null) 값이 있는 행을 제거하여 왼쪽 외부 조인을 의미구조적으로 불필요하게 만들기 때문에 CUSTOMER와 DAILY_SALES 간 왼쪽 외부 조인이 내부 조인으로 변환될 수 있습니다. 따라서 외부 조인의 존재로 인한 일부 최적화의 손실이 모든 쿼리에 역효과를 주지 않을 수도 있습니다. 그러나 이러한 제한사항을 알아 두고 절대적으로 필요하지 않은 경우 외부 조인을 피하는 것이 중요합니다.

FETCH FIRST N ROWS ONLY절과 함께 OPTIMIZE FOR N ROWS절 사용:

OPTIMIZE FOR *n* ROWS 절은 옵티마이저에 응용프로그램이 *n*개 행만 검색하려고 하지만 쿼리에서 전체 결과 세트를 리턴한다는 것을 표시합니다. FETCH FIRST *n* ROWS ONLY 절은 쿼리가 *n*개 행만 리턴해야 함을 표시합니다.

DB2 데이터 서버는 외부 하위 선택에 대해 FETCH FIRST *n* ROWS ONLY가 지정되어 있는 경우 자동으로 OPTIMIZE FOR *n* ROWS를 가정하지 않습니다. OPTIMIZE FOR *n* ROWS를 FETCH FIRST *n* ROWS ONLY와 함께 지정하여,

임시 테이블로의 삽입, 정렬 또는 해시 조인 해시 테이블로의 삽입과 같은 버퍼링 조작을 먼저 수행하지 않고 참조된 테이블에서 바로 행을 리턴하는 쿼리 액세스 플랜을 조장하십시오.

OPTIMIZE FOR *n* ROWS를 지정하여 버퍼링 조작을 피하는 쿼리 액세스 플랜을 조장하지만 전체 결과 세트를 검색하는 응용프로그램은 좋지 않은 성능을 경험할 수도 있습니다. 이는 전체 결과 세트가 검색될 경우 처음으로 *n*개 행을 가장 빨리 리턴하는 쿼리 액세스 플랜이 최상의 쿼리 액세스 플랜이 아닐 수도 있기 때문입니다.

쿼리가 스타 스키마 조인에 필요한 기준을 충족하는지 확인:

옵티마이저는 스타 스키마에 대해 스타 조인 또는 허브 조인이라고 하는 두 개의 전문화된 조인 메소드를 고려하고, 이는 성능을 크게 향상시키는 데 도움이 될 수 있습니다.

그러나 쿼리가 다음 조건을 충족해야 합니다.

- 각 쿼리 블록에 대해
  - 적어도 3개의 다른 테이블이 조인되어야 합니다.
  - 모든 Join 술어가 등식 술어여야 합니다.
  - 서브쿼리가 존재할 수 없습니다.
  - 테이블 또는 쿼리 블록의 외부 간에 상관 또는 종속성이 존재할 수 없습니다.
  - 인덱스 ANDing의 경우, 세미 조인을 용이하게 하기 위해 인덱스에 의해 사실 테이블 술어가 적용되어야 하므로 비결정론적 기능이 없어야 합니다.
  - 사실 테이블
    - 쿼리 블록에서 가장 큰 테이블입니다.
    - 적어도 10 000개 행 포함합니다.
    - 하나의 유일한 테이블로 고려됩니다.
    - 적어도 두 개의 차원 테이블 또는 눈송이라는 그룹에 조인되어야 합니다.
  - 차원 테이블
    - 사실 테이블이 아닙니다.
    - 사실 테이블 또는 눈송이에 개별적으로 조인될 수 있습니다.
  - 차원 테이블 또는 눈송이
    - 사실 테이블을 필터링해야 합니다(필터링은 옵티마이저의 추정을 기반으로 함).
    - 사실 테이블 인덱스에서 선행 컬럼을 사용하는 사실 테이블에 대한 Join 술어를 포함해야 합니다. 허브 조인에서 단일 사실 테이블 인덱스만 사용해야 하더라도, 스타 조인 또는 허브 조인이 고려되려면 이 기준이 충족되어야 합니다.

왼쪽 또는 오른쪽 외부 조인을 나타내는 쿼리 블록은 두 개의 테이블만 참조할 수 있으므로, 스타 스키마 조인은 자격이 없습니다.

옵티마이저에서 스타 스키마 조인을 인식하도록 하기 위해 참조 무결성을 명시적으로 선언할 필요가 없습니다.

중복 술어 회피:

특히 서로 다른 테이블 간에 발생할 경우, 중복 술어를 피하십시오. 일부 경우에는, 옵티마이저에서 술어가 중복인지 감지할 수 없습니다. 이는 카디널리티(cardinality) 과소평가를 초래할 수도 있습니다.

예를 들어, SAP 비즈니스 인텔리전스(BI) 응용프로그램 내에서는 사실 및 차원 테이블이 있는 눈송이 스키마가 쿼리 최적화된 데이터 구조로 사용됩니다. 일부 경우, 사실 및 차원 테이블에 정의된 중복 시간 특성 컬럼(월에 대해 ″SID_0CALMONTH″ 또는 연에 대해 ″SID_0FISCPER″)이 있습니다.

SAP BI 온라인 분석 처리(OLAP) 프로세서는 차원 및 사실 테이블의 시간 특성 컬럼에서 중복 술어를 생성합니다.

이러한 중복 술어는 쿼리 런타임을 더 길어지게 할 수도 있습니다.

다음 섹션에서는 SAP BI 쿼리의 WHERE 절에서 정의된 두 개의 중복 술어에 대한 예를 제공합니다. 동일한 술어가 시간 차원(DT) 및 사실(F) 테이블에 정의되어 있습니다.

```
     AND (     "DT"."SID_0CALMONTH" = 199605
           AND "F". "SID_0CALMONTH" = 199605
            OR "DT"."SID_0CALMONTH" = 199705
           AND "F". "SID_0CALMONTH" = 199705 )
 AND NOT (     "DT"."SID_0CALMONTH" = 199803
           AND "F". "SID_0CALMONTH" = 199803 )
```

DB2 옵티마이저는 술어를 동일한 것으로 인식하지 않고, 독립적으로 처리합니다. 이로 인해 카디널리티(cardinality)의 과소평가, 차선 쿼리 액세스 플랜 및 더 긴 쿼리 런타임이 초래됩니다.

이런 이유에서, 중복 술어가 DB2 데이터베이스 플랫폼별 소프트웨어 계층에 의해 제거됩니다.

위의 술어는 아래에 표시된 술어로 전송됩니다. 사실 테이블 컬럼 ″SID_0CALMONTH″의 술어만 유지됩니다.

```
    AND (     "F". "SID_0CALMONTH" = 199605
           OR "F". "SID_0CALMONTH" = 199705 )
AND NOT (     "F". "SID_0CALMONTH" = 199803 )
```

SAP 노트 957070 및 1144883의 지시사항에 따라 중복 술어를 제거하십시오.

제한조건을 사용하여 쿼리 최적화 향상

고유한, 점검 및 참조 무결성 제한조건 정의를 고려하십시오. 이러한 제한조건은 DB2 옵티마이저가 쿼리를 재작성하여 조인을 제거하고, 조인을 통해 집계를 푸시다운하며, 조인을 통해 FETCH FIRST *n* ROWS를 푸시다운하고, 불필요한 DISTINCT 연산을 제거하며, 다른 많은 최적화를 수행할 수 있게 하는 시맨틱 정보를 제공합니다.

또한 정보용 제한조건은 응용프로그램 자체가 관계를 보장할 수 있을 때 점검 제한조건과 참조 무결성 제한조건 둘 다에 사용될 수 있습니다. 동일한 최적화가 가능합니다. 행이 삽입, 갱신 또는 삭제될 때 데이터베이스 관리 프로그램에 의해 적용되는 제한조건은 특히 참조 무결성 제한조건이 있는 다수의 행을 갱신할 때 높은 시스템 오버헤드를 야기할 수 있습니다. 응용프로그램이 행을 갱신하기 전에 이미 정보를 검증한 경우, 일반 제한조건보다 정보용 제한조건을 사용하는 것이 더 효과적일 수도 있습니다.

예를 들어, 두 개의 테이블 DAILY_SALES와 CUSTOMER를 고려해보십시오. CUSTOMER 테이블의 각 행에는 고유 고객 키(CUST_KEY)가 있습니다. DAILY_SALES는 CUST_KEY 컬럼을 포함하고 각 행은 CUSTOMER 테이블의 고객 키를 참조합니다. CUSTOMER와 DAILY_SALES 간 이 1:N 관계를 나타내기 위해 참조 무결성 제한조건이 작성될 수 있습니다. 응용프로그램이 이 관계를 적용할 경우, 제한 조건이 정보용으로 정의될 수 있습니다. CUSTOMER에서 컬럼이 검색되지 않고 DAILY_SALES의 모든 행은 CUSTOMER에서 일치 항목을 찾기 때문에 다음 쿼리는 CUSTOMER와 DAILY_SALES 간 조인 수행을 막을 수 있습니다. 쿼리 옵티마이저가 자동으로 조인을 제거합니다.

```
SELECT AMT_SOLD, SALE PRICE, PROD_DESC
  FROM DAILY_SALES, PRODUCT, CUSTOMER
  WHERE
    DAILY_SALES.PROD_KEY = PRODUCT.PRODKEY AND
    DAILY_SALES.CUST_KEY = CUSTOMER.CUST_KEY
```

응용프로그램은 정보용 제한조건을 적용해야 합니다. 그렇지 않으면 쿼리에서 잘못된 결과를 리턴할 수도 있습니다. 이 예에서는, DAILY_SALES의 행이 CUSTOMER 테이블에 해당 고객 키가 없는 경우, 쿼리에서 이러한 행을 잘못 리턴합니다.

복합 쿼리에서 입력 변수와 함께 REOPT 바인드 옵션 사용

입력 변수는 명령문이 보다 단순하고 쿼리 액세스 플랜 선택이 보다 간단한 온라인 트랜잭션 처리(OLTP) 환경에서 적절한 명령문 준비 시간에 필수적입니다.

입력 변수 값이 다른 동일한 쿼리를 다중 실행하면 동적문 캐시에서 컴파일된 액세스 섹션을 재사용할 수 있으므로, 입력 값이 변경될 때마다 비용이 많이 드는 SQL문 컴파일을 피할 수 있습니다.

그러나 입력 변수는 쿼리 액세스 플랜 선택이 더 복잡하고 옵티마이저가 적절한 결정을 내리기 위해 더 많은 정보를 필요로 하는 복합 쿼리 워크로드에서 문제를 일으킬 수 있

습니다. 또한 명령문 컴파일 시간은 보통 총 실행 시간의 작은 구성요소이고, 반복되는 경향이 없는 비즈니스 인텔리전스(BI) 쿼리는 동적문 캐시의 이점이 없습니다.

복합 쿼리 워크로드에서 입력 변수가 사용되어야 할 경우, REOPT(ALWAYS) 바인드 옵션 사용을 고려하십시오. REOPT 바인드 옵션은 입력 변수 값이 알려진 경우 PREPARE에서 OPEN 또는 EXECUTE 시간까지 명령문 컴파일을 지연시킵니다. 옵티마이저가 값을 사용하여 더 정확한 선택 빈도 추정을 계산할 수 있도록 값이 SQL 컴파일러로 전달됩니다. REOPT(ALWAYS)는 모든 실행에서 명령문이 재컴파일되어야 하도록 지정합니다. REOPT(ALWAYS)는 또한 예를 들어, `WHERE TRANS_DATE = CURRENT DATE - 30 DAYS`와 같은 특수 레지스터를 참조하는 복합 쿼리에 사용될 수 있습니다. 입력 변수가 OLTP 워크로드에 대해 부적절한 액세스 플랜을 선택하도록 하고, REOPT(ALWAYS)가 명령문 컴파일로 인한 과도한 오버헤드를 만들 경우, 선택한 쿼리에 대해 REOPT(ONCE)를 사용하는 것을 고려하십시오. REOPT(ONCE)는 첫 번째 입력 변수 값이 바운드될 때까지 명령문 컴파일을 지연시킵니다. SQL문은 이 첫 번째 입력 변수 값을 사용하여 컴파일 및 최적화됩니다. 값이 다른 명령문의 연속 실행에서는 첫 번째 입력 값을 기반으로 컴파일된 액세스 섹션을 재사용합니다. 이는 첫 번째 입력 변수값이 후속 값을 대표하고 입력 변수값이 알려지지 않았을 때 디폴트 값을 기반으로 하는 쿼리 액세스 플랜보다 우수한 쿼리 액세스 플랜을 제공하는 경우 좋은 접근 방법일 수 있습니다.

다양한 방법으로 REPOT를 지정할 수 있습니다.

- C/C++ 응용프로그램의 Embedded SQL의 경우, REOPT 바인드 옵션을 사용하십시오. 이 바인드 옵션은 정적 및 동적 SQL 둘 다의 재최적화 동작에 영향을 줍니다.
- CLP 패키지의 경우 REOPT 바인드 옵션을 사용하여 CLP 패키지를 리바인드하십시오. 예를 들어, REOPT ALWAYS를 사용하여 분리 수준 CS에 사용된 CLP 패키지를 리바인드하려면 다음 명령을 지정하십시오.

  ```
  rebind nullid.SQLC2G13 reopt always
  ```

- Legacy JDBC 드라이버를 사용하는 JDBC 응용프로그램 또는 CLI 응용프로그램의 경우, `db2cli.ini` 구성 파일에서 REOPT 키워드 설정을 사용하십시오. 값 및 해당 옵션은 다음과 같습니다.
  - 2 - NONE
  - 3 - ONCE
  - 4 - ALWAYS
- JCC 범용 드라이버를 사용하는 JDBC 응용프로그램의 경우, 다음 방법 중 하나를 사용하십시오.
  - SQL_ATTR_REOPT 연결 또는 명령문 속성을 사용하십시오.
  - SQL_ATTR_CURRENT_PACKAGE_SET 연결 또는 명령문 속성을 사용하여 NULLID, NULLIDR1 또는 NULLIDRA 패키지 세트를 지정하십시오.

NULLIDR1과 NULLIDRA는 예약된 패키지 세트 이름입니다. 사용될 경우, 각각 REOPT ONCE 또는 REOPT ALWAYS가 수반됩니다. 이러한 패키지 세트는 다음 명령을 사용하여 명시적으로 작성되어야 합니다.

```
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA
```

- SQL PL 프로시저의 경우, 다음 방법 중 하나를 사용하십시오.
  - SET_ROUTINE_OPTS 스토어드 프로시저를 사용하여 현재 세션 내 SQL PL 프로시저의 작성에 사용될 바인드 옵션을 설정하십시오. 예를 들어, 다음을 호출하십시오.

    ```
    sysproc.set_routine_opts('reopt always')
    ```

  - **DB2_SQLROUTINE_PREPOPTS** 레지스트리 변수를 사용하여 인스턴스 레벨에서 SQL PL 프로시저 옵션을 설정하십시오. SET_ROUTINE_OPTS 스토어드 프로시저를 사용하여 설정된 값은 **DB2_SQLROUTINE_PREPOPTS**를 사용하여 지정된 값을 겹쳐씁니다.

다음 예에 표시된 것처럼 최적화 프로파일을 사용하여 정적문과 동적문에 대한 REOPT를 설정할 수도 있습니다.

```
<STMTPROFILE ID="REOPT example ">
     <STMTKEY>
       <![CDATA[select acct_no from customer where name = ? ]]>
     </STMTKEY>
     <OPTGUIDELINES>
        <REOPT VALUE='ALWAYS'/>
    </OPTGUIDELINES>
</STMTPROFILE>
```

매개변수 표시문자를 사용하여 동적 쿼리에 대한 컴파일 시간 단축

DB2 데이터 서버는 액세스 섹션 및 명령문 텍스트를 동적문 캐시에 저장하여 이전에 실행된 동적 SQL문 재컴파일을 피할 수 있습니다.

이 명령문에 대한 후속 준비 요청에서는 컴파일을 피하면서, 동적문 캐시에서 액세스 섹션을 찾으려고 시도합니다. 그러나 술어에 사용된 리터럴만 다른 명령문은 일치되지 않습니다. 예를 들어, 다음 두 명령문은 동적문 캐시에서 다른 것으로 간주됩니다.

```
SELECT AGE FROM EMPLOYEE WHERE EMP_ID = 26790
SELECT AGE FROM EMPLOYEE WHERE EMP_ID = 77543
```

매우 자주 실행될 경우, 상대적으로 단순한 SQL문이라 하더라도 명령문 컴파일로 인해 과도한 시스템 CPU 사용을 초래할 수 있습니다. 시스템이 이런 유형의 성능 문제점을 겪을 경우, 매개변수 표시문자를 사용하도록 응용프로그램을 변경하여, SQL문에 명시적으로 포함하기보다는 DB2 컴파일러에 술어 값을 전달하는 것을 고려하십시오. 그러나 술어에서 매개변수 표시문자를 사용하는 복합 쿼리에 대해 액세스 플랜이 최적이 아닐 수도 있습니다. 자세한 정보는 『복합 쿼리에서 입력 변수와 함께 REOPT 바인드 옵션 사용』을 참조하십시오.

DB2_REDUCED_OPTIMIZATION 레지스트리 변수 설정

최적화 클래스 설정을 통해 응용프로그램에 대한 컴파일 시간이 충분히 단축되지 않을 경우, **DB2_REDUCED_OPTIMIZATION** 레지스트리 변수 설정을 시도하십시오.

이 레지스트리 변수는 옵티마이저의 검색 스페이스에 대해 최적화 클래스 설정보다 더 많은 제어를 제공합니다. 이 레지스트리 변수는 지정된 최적화 클래스에서 최적화 기능의 엄격한 사용 또는 감소된 최적화 기능을 요청할 수 있게 해 줍니다. 사용되는 최적화 기술의 수를 줄일 경우, 최적화 동안의 시간 및 자원 사용도 줄어듭니다.

최적화 시간 및 자원 사용이 감소될 수도 있지만, 덜 최적화된 쿼리 액세스 플랜을 생성할 위험이 커집니다.

먼저, 레지스트리 변수를 예로 설정해보십시오. 최적화 클래스가 5(디폴트) 이하인 경우, 옵티마이저는 상당한 준비 시간 및 자원을 사용할 수도 있지만 일반적으로 더 나은 쿼리 액세스 플랜을 생성하지 않는 일부 최적화 기술을 사용하지 않도록 합니다. 최적화 클래스가 정확히 5인 경우, 옵티마이저는 최적화 시간 및 자원 사용을 더 줄일 수도 있지만 덜 최적화된 쿼리 액세스 플랜을 생성할 위험을 증가시키기도 하는 일부 추가 기술을 줄이거나 사용하지 않도록 합니다. 5보다 낮은 최적화 클래스의 경우, 이러한 기술 중 일부가 어떤 경우에도 효력이 없을 수도 있습니다. 그러나 그럴 경우, 여전히 적용됩니다.

예 설정이 컴파일 시간의 충분한 단축을 제공하지 않을 경우, 레지스트리 변수를 다른 정수 값으로 설정해보십시오. 효과는 예 설정과 동일하며, 클래스 5로 최적화된 동적으로 준비된 쿼리에 대한 다음 추가 동작이 있습니다. 쿼리 블록의 총 조인 수가 설정을 초과할 경우, 옵티마이저는 추가 최적화 기술을 사용하지 않도록 하는 대신 그리디(Greedy) 조인 열거로 전환합니다. 그 결과 쿼리는 최적화 클래스 2와 유사한 레벨에서 최적화됩니다.

삽입 성능 향상

테이블에 데이터를 삽입하기 전에 삽입 검색 알고리즘은 FSCR(Free Space Control Record)을 조사하여 새 데이터를 위한 충분한 공간이 있는 페이지를 찾습니다.

그러나 FSCR이 페이지에 충분한 여유 공간이 있음을 표시하는 경우에도 다른 트랜잭션의 커미트되지 않은 삭제 조작으로 예약된 경우 해당 스페이스를 사용할 수 없습니다.

DB2MAXFSCRSEARCH 레지스트리 변수는 테이블에 레코드를 추가할 때 검색할 FSCR의 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 사

용하십시오. 값을 -1로 설정하면 데이터베이스 관리 프로그램이 모든 FSCR을 강제로 검색합니다. FSCR을 검색하는 동안 충분한 스페이스를 찾을 수 없는 경우 테이블의 끝에 데이터를 추가합니다.

ALTER TABLE문의 APPEND ON 옵션은 테이블 데이터가 추가되며 페이지의 여유 공간에 대한 정보를 보존하지 않음을 지정합니다. 이러한 테이블에는 클러스터링 인덱스가 없어야 합니다. 이 옵션은 증가하기만 하는 테이블의 성능을 향상시킬 수 있습니다.

테이블에 클러스터링 인덱스가 정의된 경우 데이터베이스 관리 프로그램은 비슷한 인덱스 키 값을 사용하여 다른 레코드와 동일한 페이지에서 레코드를 삽입하려고 시도합니다. 해당 페이지에 스페이스가 없는 경우 주변 페이지를 고려합니다. 이러한 페이지가 적합하지 않은 경우 위에 설명된 바와 같이 FSCR을 검색합니다. 그러나 이 경우 『최초 적합』 대신 『최악 적합』 접근 방법을 사용합니다. 최악 적합 접근 방법은 여유 공간이 많은 페이지를 선택합니다. 이 방법은 비슷한 키 값을 사용하여 행의 새 클러스터링 영역을 설정합니다.

테이블에서 클러스터링 인덱스를 정의한 경우 테이블을 로드하거나 재구성하기 전에 ALTER TABLE문에서 PCTFREE절을 사용하십시오. PCTFREE절은 로드 또는 reorg 조작 이후에 데이터 페이지에 남아 있어야 하는 여유 공간의 퍼센트를 지정합니다. 이 경우 클러스터 인덱스 조작이 적합한 페이지에서 여유 공간을 찾을 가능성이 높아집니다.

효율적인 SELECT문

SQL은 유연한 고급 언어이므로 동일한 데이터를 검색하는 다른 SELECT문을 작성할 수 있습니다. 그러나 명령문 형식과 최적화 클래스에 따라 성능이 다양할 수 있습니다.

효율적인 SELECT문을 작성할 때 다음 지침을 고려하십시오.

- 필요한 컬럼만 지정하십시오. 별표(*)를 사용하여 모든 컬럼을 지정하면 불필요한 처리가 발생합니다.
- 응답 세트를 필요한 행만으로 제한하는 술어를 사용하십시오.
- 리턴될 수 있는 전체 행 수보다 훨씬 적게 필요한 경우 OPTIMIZE FOR절을 지정하십시오. 이 절은 액세스 플랜 선택 및 통신 버퍼에서 블로킹된 행 수에 영향을 줍니다.
- 행 블로킹을 활용하고 성능을 향상시키려면 FOR READ ONLY 또는 FOR FETCH ONLY절을 지정하십시오. 검색된 행에서 배타적 잠금을 보유하지 않으므로 동시성도 향상됩니다. 추가 쿼리를 다시 쓸 수도 있습니다. 이러한 절과 BLOCKING ALL 바인드 옵션을 지정해도 페더레이티드 데이터베이스 시스템의 별칭에 대해 실행하는 쿼리의 성능을 향상시킬 수 있습니다.

- 위치 지정된 갱신사항과 함께 사용할 커서의 경우 데이터베이스 관리 프로그램이 처음에 적합한 잠금 레벨을 선택하고 잠재적인 교착 상태를 방지하려면 FOR UPDATE OF절을 지정하십시오. FOR UPDATE 커서는 행 블로킹을 활용할 수 없다는 점을 참고하십시오.
- 검색된 갱신사항에 사용되는 커서의 경우 교착 상태를 방지하고 영향을 받은 행에서 U 잠금을 강제로 실행하여 행 블로킹을 계속 허용하려면 FOR READ ONLY 및 USE AND KEEP UPDATE LOCKS절을 지정하십시오.
- 가능하면 숫자 데이터 유형 변환을 방지하십시오. 값을 비교할 때 동일한 데이터 유형의 항목을 사용하십시오. 변환이 필요한 경우 정밀도 제한으로 인한 부정확성과 런타임 변환으로 인한 성능 비용이 발생할 수 있습니다.

  가능하면 다음의 데이터 유형을 사용하십시오.
  - 간단한 컬럼의 경우 가변 문자 대신 문자
  - 부동수, 10진수 또는 DECFLOAT 대신 정수
  - 10진수 대신 DECFLOAT
  - 문자 대신 날짜 및 시간
  - 문자 대신 숫자
- 정렬 조작이 발생할 가능성을 줄이려면 DISTINCT 또는 ORDER BY절 등을 생략하십시오(이러한 연산이 필요하지 않은 경우).
- 테이블에서 행 존재를 확인하려면 단일 행을 선택하십시오. 커서를 열고 한 행을 페치하거나 단일 행 SELECT INTO 연산을 수행하십시오. 여러 행이 발견된 경우 SQLCODE -811 오류가 있는지 확인하십시오.

  테이블이 매우 작은 것으로 판단되지 않는 한, 0이 아닌 값이 있는지 확인하기 위해 다음 명령문을 사용하지 마십시오.

  ```
  select count(*) from <table-name>
  ```

  테이블이 큰 경우 모든 행을 계산하면 성능에 영향을 줄 수 있습니다.
- 갱신 활동이 낮고 테이블이 큰 경우 술어에서 자주 사용하는 컬럼에 대해 인덱스를 정의하십시오.
- 동일한 컬럼이 여러 술어에 표시되는 경우 IN 목록을 사용하십시오. 호스트 변수에 사용되는 큰 IN 목록의 경우 호스트 변수 서브세트를 순환시키면 성능이 향상될 수 있습니다.

다음의 제안사항은 특히 여러 테이블에 액세스하는 SELECT문에 적용됩니다.

- Join 술어를 사용하여 테이블을 조인하십시오. *Join 술어*는 조인에서 다른 테이블의 두 컬럼을 비교합니다.

- Join 술어에 컬럼에 대한 인덱스를 정의하면 조인을 효율적으로 처리할 수 있습니다. 인덱스는 여러 테이블에 액세스하는 SELECT문을 포함한 UPDATE 및 DELETE 문을 활용할 수도 있습니다.
- 가능하면 Join 술어에 OR 절 또는 표현식을 사용하지 마십시오.
- 파티션된 데이터베이스 환경에서는 조인되는 테이블을 조인 컬럼에서 파티션하도록 권장됩니다.

SELECT문 제한 지침

옵티마이저는 응용프로그램이 SELECT문으로 식별된 모든 행을 검색해야 한다고 가정합니다. 이 가정은 온라인 트랜잭션 처리(OLTP) 및 일괄처리 환경에 가장 적합합니다.

그러나『찾아보기』응용프로그램에서 쿼리는 잠재적인 대형 응답 세트를 자주 정의하지만 보통 특정 표시 형식에 필요한 행 수와 같은 처음 일부 행만 검색합니다.

이러한 응용프로그램의 성능을 향상시키려면 다음과 같은 방법으로 SELECT문을 수정할 수 있습니다.

- FOR UPDATE절을 사용하여 나중에 위치 지정된 UPDATE문으로 갱신할 수 있는 컬럼을 지정하십시오.
- FOR READ 또는 FETCH ONLY절을 사용하여 리턴된 컬럼을 읽기 전용 컬럼으로 지정하십시오.
- OPTIMIZE FOR *n* ROWS절을 사용하여 전체 결과 세트에서 처음 *n*행 검색에 우선순위를 부여하십시오.
- FETCH FIRST *n* ROWS ONLY절을 사용하여 지정된 행 수만 검색하십시오.
- DECLARE CURSOR WITH HOLD문을 사용하여 한 번에 한 행씩 검색하십시오.

다음 절에서는 각 메소드의 성능 이점에 대해 설명합니다.

FOR UPDATE절

FOR UPDATE절은 나중에 위치 지정된 UPDATE문으로 갱신할 수 있는 컬럼만 포함하여 결과 세트를 제한합니다. FOR UPDATE절을 컬럼 이름 없이 지정하면 테이블 또는 뷰에서 갱신할 수 있는 모든 컬럼이 포함됩니다. 컬럼 이름을 지정할 경우 각 이름은 규정되지 않은 이름이어야 하며 테이블 또는 뷰의 컬럼을 식별해야 합니다.

다음과 같은 경우 FOR UPDATE절을 사용할 수 없습니다.

- SELECT문과 연관된 커서를 삭제할 수 없는 경우
- 선택한 컬럼 중 최소한 하나가 카탈로그 테이블에서 갱신할 수 없으며 FOR UPDATE 절에서 제외되지 않은 컬럼인 경우

DB2 CLI 응용프로그램에서 동일한 용도로 CLI 연결 속성 SQL_ATTR_ACCESS_MODE를 사용할 수 있습니다.

FOR READ 또는 FETCH ONLY절

FOR READ ONLY절 또는 FOR FETCH ONLY절을 사용하면 읽기 전용 결과가 리턴됩니다. 갱신 및 삭제가 허용되는 결과 테이블의 경우 FOR READ ONLY절을 지정하면 데이터베이스 관리 프로그램이 배타적 잠금을 사용하는 대신 데이터 블록을 검색할 수 있는 경우 페치 조작의 성능을 향상시킬 수 있습니다. 위치 지정된 UPDATE 또는 DELETE문에서 사용되는 쿼리에 FOR READ ONLY절을 지정하지 마십시오.

DB2 CLI 응용프로그램에서 동일한 용도로 CLI 연결 속성 SQL_ATTR_ACCESS_MODE를 사용할 수 있습니다.

OPTIMIZE FOR n ROWS절

OPTIMIZE FOR절은 결과의 서브세트만 검색하거나 처음 일부 행만 검색하도록 우선순위를 부여할 의도가 있음을 선언합니다. 그런 다음 옵티마이저는 처음 일부 행을 검색할 수 있도록 응답 시간을 최소화하는 액세스 플랜을 선택할 수 있습니다. 또한 단일 블록으로 클라이언트에 전송되는 행 수는 *n* 값으로 제한됩니다. 따라서 OPTIMIZE FOR 절은 서버가 데이터베이스에서 규정 행을 검색하는 방식과 서버가 클라이언트에 이러한 행을 리턴하는 방식에 영향을 줍니다.

예를 들어, EMPLOYEE 테이블을 주기적으로 쿼리하여 최고 급여를 받는 직원을 판별한다고 가정하십시오.

```
select lastname, firstnme, empno, salary
  from employee
  order by salary desc
```

SALARY 컬럼에서 내림차순 인덱스를 이전에 정의한 경우에도 직원은 직원 번호별로 정렬되어 있으므로 이 인덱스는 클러스터링이 잘못될 가능성이 있습니다. 많은 무작위 동기 입출력을 방지하기 위해 옵티마이저는 규정된 모든 행의 행 ID를 정렬해야 하는 리스트 프리페치 액세스 메소드를 선택하게 됩니다. 이 정렬로 인해 응용프로그램으로 첫 번째 규정 행을 리턴하기 전에 지연이 발생할 수 있습니다. 이러한 지연을 예방하려면 다음과 같이 명령문에 OPTIMIZE FOR절을 추가하십시오.

```
select lastname, firstnme, empno, salary
  from employee
  order by salary desc
  optimize for 20 rows
```

이 경우 최고 급여를 받는 20명의 직원만 검색하므로 옵티마이저는 SALARY 인덱스를 직접 사용하도록 선택할 가능성이 높습니다. 블로킹할 수 있는 행 수에 관계없이 20 행마다 클라이언트로 행 블록이 리턴됩니다.

OPTIMIZE FOR절에서 옵티마이저는 대량 조작이나 플로우 인터럽트(예: 정렬 조작으로 인해 발생하는 경우)를 방지하는 액세스 플랜을 선호합니다. OPTIMIZE FOR 1 ROW절을 사용하여 액세스 경로에 영향을 줄 수 있습니다. 이 절을 사용하면 다음과 같은 효과가 있습니다.

- 복합 내부 테이블과 조인 시퀀스는 임시 테이블이 필요하므로 적합하지 않을 수 있습니다.
- 조인 방법이 변경될 수 있습니다. 중첩된 루프 조인은 오버헤드 비용이 낮고 보통 일부 행을 검색할 때 효율적이므로 가장 적합한 선택사항입니다.
- ORDER BY에는 정렬이 필요하지 않으므로 ORDER BY절과 일치하는 인덱스가 적합할 수 있습니다.
- 리스트 프리페치는 정렬이 필요한 액세스 메소드이므로 적합하지 않을 수 있습니다.
- 순차 프리페치는 적은 행 수만 필요하므로 적합하지 않을 수 있습니다.
- 조인 쿼리에서 외부 테이블에 대한 인덱스가 ORDER BY절에 필요한 정렬을 제공하는 경우 ORDER BY절에 컬럼이 있는 테이블을 외부 테이블로 선택할 수 있습니다.

OPTIMIZE FOR절은 모든 최적화 레벨에 적용되지만 3 미만의 클래스는 *greedy 조인 열거* 검색 전략을 사용하므로 최적화 클래스 3 이상의 경우에 가장 적합합니다. 이 방법을 사용할 경우 처음 일부 행을 신속하게 검색하는 데 다중 테이블 조인용 액세스 플랜을 사용할 수 없습니다.

패키지화된 응용프로그램이 호출 레벨 인터페이스(DB2 CLI 또는 ODBC)를 사용하는 경우 `db2cli.ini` 구성 파일에서 **OPTIMIZEFORNROWS** 키워드를 사용하여 DB2 CLI가 각 쿼리 명령문의 끝에 OPTIMIZE FOR절을 자동으로 추가하도록 설정할 수 있습니다.

별칭에서 데이터를 선택할 때 데이터 소스 지원에 따라 결과가 다양할 수 있습니다. 별칭에 참조된 데이터 소스가 OPTIMIZE FOR절을 지원하며 DB2 옵티마이저가 전체 쿼리를 데이터 소스에 푸시다운하는 경우 데이터 소스에 전송되는 리모트 SQL에서 이 절이 생성됩니다. 데이터 소스가 이 절을 지원하지 않거나 옵티마이저가 가장 비용이 적게 드는 플랜이 로컬 실행이라고 결정하면 OPTIMIZE FOR절이 로컬로 적용됩니다. 이 경우 DB2 옵티마이저는 쿼리의 처음 일부 행을 검색하기 위해 응답 시간을 최소화하는 액세스 플랜을 선호하지만 옵티마이저가 플랜을 생성하는 데 사용할 수 있는 옵션은 일부 제한되며 OPTIMIZE FOR절의 성능 이점은 작습니다.

OPTIMIZE FOR절 및 FETCH FIRST절이 모두 지정된 경우 두 개의 *n* 값 중 작은 값이 통신 버퍼 크기에 영향을 줍니다. 두 값은 최적화 용도에서 서로 관계가 없는 것으로 간주됩니다.

FETCH FIRST n ROWS ONLY절

FETCH FIRST *n* ROWS ONLY절은 검색할 수 있는 최대 행 수를 설정합니다. 결과 테이블을 처음 일부 행으로 제한하면 성능이 향상될 수 있습니다. 결과 세트가 포함할 수 있는 행 수에 관계없이 *n*행만 검색합니다.

FETCH FIRST절 및 OPTIMIZE FOR절이 모두 지정된 경우 두 개의 *n* 값 중 작은 값이 통신 버퍼 *크기*에 영향을 줍니다. 두 값은 최적화 용도에서 서로 관계가 없는 것으로 간주됩니다.

DECLARE CURSOR WITH HOLD문

WITH HOLD절을 포함하는 DECLARE CURSOR문을 사용하여 커서를 선언하는 경우 열린 커서는 트랜잭션이 커미트할 때 계속 열려 있고 현재 커서 위치를 보호하는 잠금을 제외한 모든 잠금은 해제됩니다. 트랜잭션이 롤백되는 경우 열린 커서는 모두 닫히고 모든 잠금이 해제되고 LOB 로케이터가 해제됩니다.

DB2 CLI 응용프로그램에서 CLI 연결 속성 SQL_ATTR_CURSOR_HOLD를 동일한 용도로 사용할 수 있습니다. 패키지화된 응용프로그램이 콜 레벨 인터페이스(DB2 CLI 또는 ODBC)를 사용하는 경우 `db2cli.ini` 구성 파일에서 **CURSORHOLD** 키워드를 사용하여 DB2 CLI가 선언된 모든 커서에 대해 WITH HOLD절을 자동으로 가정하도록 설정하십시오.

오버헤드를 줄이기 위한 행 블로킹 지정

모든 명령문 및 데이터 유형(LOB 데이터 유형 포함)에 대해 지원되는 행 블로킹은 단일 조작으로 행의 블록을 검색함으로써 커서에 대한 데이터베이스 관리 프로그램 오버헤드를 줄입니다.

이 행 블록은 메모리의 페이지 수를 나타냅니다. 이것은 실제로 디스크의 Extent에 맵핑되는 다차원(MDC) 테이블 블록이 아닙니다.

행 블로킹은 BIND 또는 PREP 명령에서 다음 옵션을 통해 지정됩니다.

BLOCKING ALL
: FOR READ ONLY 절을 통해 선언되거나 FOR UPDATE로 지정되지 않은 커서가 차단됩니다.

BLOCKING NO
: 커서가 차단되지 않습니다.

BLOCKING UNAMBIG
: FOR READ ONLY 절을 통해 선언된 커서가 차단됩니다. FOR READ ONLY 절 또는 FOR UPDATE 절을 통해 선언되지 않거나, 모호하지 않거나, 읽기 전용인 커서가 차단됩니다. 앰비규어스 커서는 차단되지 않습니다.

다음 데이터베이스 관리 프로그램 구성 매개변수가 블록 크기 계산 동안 사용됩니다.

- **aslheapsz** 매개변수는 로컬 응용프로그램에 대한 응용프로그램 지원 계층 힙의 크기를 지정합니다. 이것은 블로킹 커서가 열려 있을 때 입출력 블록 크기를 판별하는 데 사용됩니다.
- **rqrioblk** 매개변수는 데이터베이스 서버에서 해당 데이터베이스 에이전트와 리모트 응용프로그램 간 통신 버퍼의 크기를 지정합니다. 이것은 또한 블로킹 커서가 열려 있을 때 데이터 서버 런타임 클라이언트의 입출력 블록 크기를 판별하는 데 사용됩니다.

LOB 데이터 유형에 대한 행 데이터의 블로킹을 사용하기 전에 시스템 자원에 대한 영향을 파악하는 것이 중요합니다. LOB 컬럼이 리턴될 때 각 데이터 블록의 LOB 값에 대한 참조를 저장하기 위해 서버에서 더 많은 공유 메모리가 소비됩니다. 이러한 참조의 수는 **rqrioblk** 구성 매개변수의 값에 따라 달라집니다.

힙에 할당된 메모리의 양을 늘리려면 다음을 수행하여 **database_memory** 데이터베이스 구성 매개변수를 수정하십시오.

- 해당 값을 AUTOMATIC으로 설정
- 매개변수가 현재 사용자 정의 숫자 값으로 설정되어 있는 경우 해당 값을 256페이지만큼 늘림

LOB 값을 참조하는 Embedded SQL 응용프로그램의 성능을 향상시키려면 BIND 명령을 사용하고 BLOCKING ALL 절 또는 BLOCKING UNAMBIG 절을 지정해 블로킹을 요청하여 응용프로그램을 리바인드하십시오. 임베디드 응용프로그램은 행의 블록이 서버에서 검색된 후, 한 번에 한 행씩 LOB 값을 검색합니다. LOB 결과를 리턴하는 사용자 정의 함수(UDF)는 서버에서 대량의 메모리가 소비되고 있을 때 DB2 서버가 LOB 데이터의 단일 행 검색으로 되돌리게 할 수도 있습니다.

행 블로킹을 지정하려면 다음을 수행하십시오.

1. **aslheapsz** 및 **rqrioblk** 구성 매개변수의 값을 사용하여 각 블록에 대해 리턴되는 행 수를 추정하십시오. 두 공식에서 *orl*은 출력 행 길이(바이트)입니다.
   - 로컬 응용프로그램에 대해 다음 공식을 사용하십시오.

     ```
     Rows per block = aslheapsz * 4096 / orl
     ```

     페이지당 바이트 수는 4096입니다.
   - 리모트 응용프로그램에 대해 다음 공식을 사용하십시오.

     ```
     Rows per block = rqrioblk / orl
     ```
2. 행 블로킹을 사용하려면 BIND 또는 PREP 명령에서 BLOCKING 옵션에 대한 적절한 값을 지정하십시오.

BLOCKING 옵션을 지정하지 않을 경우, 디폴트 행 블로킹 유형은 UNAMBIG입니다. 명령행 처리기(CLP) 및 콜 레벨 인터페이스(CLI)의 경우, 디폴트 행 블로킹 유형은 ALL입니다.

쿼리의 데이터 샘플링

쿼리에 관련된 모든 데이터에 액세스하는 것은 종종 비실용적이고 때로는 불필요합니다. 일부 경우에는 데이터의 서브세트에서 패턴이나 추세를 찾는 것으로 충분합니다. 이를 수행하는 한 가지 방법은 데이터베이스의 무작위 샘플에 대해 쿼리를 실행하는 것입니다.

DB2 제품을 통해 SQL 및 XQuery 쿼리에 대한 데이터를 효과적으로 샘플링하고, 잠재적으로는 높은 등급의 정밀도를 유지하면서 대규모 쿼리의 성능을 크게 향상시킬 수 있습니다.

샘플링은 보통 AVG, COUNT 및 SUM과 같은 집계 쿼리에 사용되며, 이 경우 데이터 샘플에서 집계를 위한 정확한 값을 얻을 수 있습니다. 샘플링은 데이터 마이닝 및 분석 속도를 높이기 위해서나 감사 용도로 테이블에서 행의 무작위 서브세트를 얻는 데에도 사용될 수 있습니다.

두 가지 샘플링 방법, 행 레벨 샘플링과 페이지 레벨 샘플링이 사용 가능합니다.

행 레벨 Bernoulli 샘플링

행 레벨 Bernoulli 샘플링은 샘플에 *P*/100의 확률로 각 행을 포함시키고 1-*P*/100의 확률로 각 행을 제외시키는 SARGable 술어를 통해 *P* 퍼센트의 테이블 열 샘플을 얻습니다.

행 레벨 Bernoulli 샘플링은 데이터 클러스터링 등급과 관계없이 항상 올바른 무작위 샘플을 생성합니다. 그러나 모든 행이 검색되어야 하고 샘플링 술어가 적용되어야 하므로 인덱스가 사용 가능하지 않은 경우 이 샘플링 유형의 성능은 매우 낮습니다. 인덱스가 없는 경우, 샘플링을 하지 않고 쿼리를 실행하는 데 비해 입출력 절감이 없습니다. 인덱스를 사용 가능한 경우에는 샘플링 술어가 인덱스 리프 페이지 내 RIDS에 적용되므로 성능이 향상됩니다. 일반적으로 이 경우, 선택된 RID당 하나의 입출력과 인덱스 리프 페이지당 하나의 입출력이 필요합니다.

시스템 페이지 레벨 샘플링

시스템 페이지 레벨 샘플링은 행이 아닌 페이지가 샘플링된다는 점을 제외하고는 행 레벨 샘플링과 비슷합니다. 샘플에 포함되는 페이지의 확률은 *P*/100입니다. 페이지가 포함된 경우 해당 페이지의 모든 행이 포함됩니다.

샘플에 포함된 각 페이지에 대해 하나의 입출력만 필요하므로 시스템 페이지 레벨 샘플링의 성능은 매우 좋습니다. 샘플링을 하지 않는 것에 비해, 페이지 레벨 샘플링은 성

능을 크게 향상시킵니다. 그러나 집계 추정의 정밀도는 행 레벨 샘플링보다 페이지 레벨 샘플링에서 더 나쁜 경향이 있습니다. 이러한 차이는 페이지당 많은 행이 있는 경우나 쿼리에서 참조되는 컬럼이 페이지 내에서 높은 클러스터링 등급을 표시할 경우 가장 잘 나타납니다.

샘플링 방법 지정

TABLESAMPLE 절을 사용하여 테이블에서 데이터의 무작위 샘플에 대해 쿼리를 실행합니다. TABLESAMPLE BERNOULLI는 행 레벨 Bernoulli 샘플링이 수행되도록 지정합니다. TABLESAMPLE SYSTEM은 옵티마이저에서 행 레벨 Bernoulli 샘플링을 대신 수행하는 것이 더 효과적이라고 판별하지 않는 한, 시스템 레벨 샘플링이 수행되도록 지정합니다.

응용프로그램의 병렬 처리

DB2 제품은 주로 대칭형 멀티프로세서(SMP) 머신에서 병렬 환경을 지원합니다.

SMP 머신에서는 여러 프로세서가 데이터베이스에 액세스할 수 있으므로 복잡한 SQL 요청 실행을 여러 프로세서로 분산시킬 수 있습니다. 이 *파티션 내 병렬 처리*는 단일 데이터베이스 조작(예: 인덱스 작성)을 여러 부분으로 세부 구분하여 단일 데이터베이스 파티션에서 병렬로 실행합니다.

응용프로그램을 컴파일할 때 병렬 처리 수준을 지정하려면 CURRENT DEGREE 특수 레지스터 또는 DEGREE 바인드 옵션을 사용하십시오. 수준은 동시에 실행할 수 있는 쿼리 부분 수를 나타냅니다. 프로세서 수와 병렬 처리 수준으로 선택하는 값 사이에는 확실한 관계가 없습니다. 머신의 프로세서 수보다 크거나 작은 값을 지정할 수 있습니다. 단일프로세서 머신의 경우도 성능을 향상시키기 위해 수준을 1보다 크게 설정할 수 있습니다. 그러나 각 병렬 처리 수준은 시스템 메모리 및 프로세서 오버헤드에 추가된다는 점을 참고하십시오.

쿼리 병렬 실행을 사용하는 경우 성능을 최적화하려면 일부 구성 매개변수를 수정해야 합니다. 병렬 처리 수준이 높은 환경에서는 공유 메모리 양과 프리페치를 제어하는 구성 매개변수를 검토하고 수정해야 합니다.

다음 구성 매개변수는 병렬 처리를 제어하고 관리합니다.

- **intra_parallel** 데이터베이스 관리 프로그램 구성 매개변수는 병렬 처리를 사용하거나 사용하지 않습니다.
- **max_querydegree** 데이터베이스 관리 프로그램 구성 매개변수는 데이터베이스의 쿼리에 대한 병렬 처리 수준의 상한을 설정합니다. 이 값은 CURRENT DEGREE 특수 레지스터 및 DEGREE 바인드 옵션을 겹쳐씁니다.
- **dft_degree** 데이터베이스 구성 매개변수는 CURRENT DEGREE 특수 레지스터 및 DEGREE 바인드 옵션의 디폴트값을 설정합니다.

DEGREE = ANY를 사용하여 쿼리가 컴파일된 경우 데이터베이스 관리 프로그램은 프로세서 수와 쿼리 특성을 포함하여 인수의 수에 따라 파티션 내 병렬 처리 수준을 선택합니다. 런타임에 사용된 실제 수준은 이러한 인수와 시스템에서 활동량에 따라 프로세서 수보다 낮을 수도 있습니다. 시스템이 사용 중인 경우 쿼리 실행 전에 병렬 처리 수준이 낮아질 수 있습니다.

DB2 Explain 기능을 사용하여 옵티마이저가 선택한 병렬 처리 수준에 대한 정보를 표시하십시오. 데이터베이스 시스템 모니터를 사용하여 런타임에 실제로 사용 중인 병렬 처리 수준에 대한 정보를 표시하십시오.

비SMP 환경에서 병렬 처리

SMP 머신이 없어도 병렬 처리 수준을 지정할 수 있습니다. 예를 들어, 단일프로세서에서 입출력 위주의 쿼리는 2 이상의 수준을 선언하면 좋습니다. 이 경우 프로세서는 새 쿼리를 처리하기 전에 입출력 태스크가 완료될 때까지 대기할 필요가 없습니다. 로드와 같은 유틸리티는 입출력 병렬 처리를 개별적으로 제어할 수 있습니다.

잠금 관리

잠금 관리는 응용프로그램 성능에 영향을 미치는 인수 중 하나입니다. 데이터베이스 응용프로그램의 성능을 최대화하는 데 도움이 되는 잠금 관리 고려사항에 대한 자세한 내용은 이 절을 검토하십시오.

잠금 및 동시처리 제어

동시처리 제어를 제공하고 제어할 수 없는 데이터 액세스를 방지하기 위해 데이터베이스 관리 프로그램은 버퍼 풀, 테이블, 데이터 파티션, 테이블 블록 또는 테이블 행에 대한 잠금을 설정합니다.

*잠금*은 데이터베이스 관리 프로그램 자원을 *잠금 소유자*라고 하는 응용프로그램과 연관시켜서 다른 응용프로그램이 동일한 자원에 액세스하는 방식을 제어합니다.

데이터베이스 관리 프로그램은 다음 사항에 따라 행 레벨 잠금 또는 테이블 레벨 잠금을 적절하게 사용합니다.

- 프리컴파일 시 또는 데이터베이스에 응용프로그램이 바인드될 때 지정된 분리 수준. 분리 수준은 다음 중 하나입니다.
  - 커미트되지 않은 읽기(UR)
  - 커서 안정성(CS)
  - 읽기 안정성(RS)
  - 반복 읽기(RR)

여러 가지 분리 수준을 사용하여 커미트되지 않은 데이터에 대한 액세스를 제어하고 갱신사항 유실을 방지하고 데이터의 비반복 읽기를 허용하고 팬텀 읽기를 방지합니다. 성능 영향을 최소화하려면 응용프로그램 필요를 충족시키는 최소 분리 수준을 사용하십시오.

- 옵티마이저가 선택한 액세스 플랜. 테이블 스캔, 인덱스 스캔 및 기타 데이터 액세스 방법에는 각각 데이터에 대한 다른 액세스 유형이 필요합니다.
- 테이블의 LOCKSIZE 속성. ALTER TABLE문의 LOCKSIZE절은 테이블에 액세스할 때 사용되는 잠금 단위를 표시합니다. 선택사항은 다음과 같습니다. 행 잠금의 경우 ROW, 테이블 잠금의 경우 TABLE 또는 MDC(다차원 클러스터링) 테이블에서 블록 잠금의 경우 BLOCKINSERT입니다. MDC 테이블에서 BLOCKINSERT 절을 사용하는 경우 블록 레벨 잠금을 대신 수행할 때 삽입 조작을 수행하는 동안을 제외하고 행 레벨 잠금을 수행합니다. 트랜잭션이 불연속 셀에 대량의 삽입을 수행할 경우 MDC 테이블에 대해 ALTER TABLE...LOCKSIZE BLOCKINSERT 문을 사용하십시오. 읽기 전용 테이블에 대해 ALTER TABLE...LOCKSIZE TABLE 문을 사용하십시오. 이 경우 데이터베이스 활동에 필요한 잠금 수가 줄어듭니다. 파티션된 테이블의 경우 테이블 잠금을 먼저 획득한 후 액세스할 데이터에서 지시하는 바와 같이 데이터 파티션 잠금을 획득합니다.
- **locklist** 데이터베이스 구성 매개변수가 제어하는 잠금 전용 메모리의 양. 잠금 목록이 가득 찬 경우 데이터베이스에서 공유 오브젝트들 간 잠금 에스컬레이션 및 동시성 감소로 인해 성능이 저하될 수 있습니다. 잠금 에스컬레이션이 자주 발생하는 경우 **locklist**, **maxlocks** 또는 두 값을 모두 늘리십시오. 한 번에 보유하는 잠금 수를 줄이려면 트랜잭션이 자주 커미트되는지 확인하십시오.

버퍼 풀을 작성, 변경 또는 삭제할 때마다 버퍼 풀 잠금(독점)을 설정합니다. 시스템 모니터링 데이터를 수집할 때 이러한 유형의 잠금에 직면할 수도 있습니다. 잠금의 이름은 버퍼 풀 자체의 ID입니다.

다음 중 하나에 해당하지 않으면 일반적으로 행 레벨 잠금을 사용합니다.

- 분리 수준이 커미트되지 않은 읽기인 경우
- 분리 수준이 반복 읽기이며 액세스 플랜에 인덱스 범위 술어가 없는 스캔이 필요한 경우
- 테이블 LOCKSIZE 속성이 TABLE인 경우
- 잠금 목록이 가득 차서 잠금 에스컬레이션이 발생한 경우
- 동시 응용프로그램 프로세스가 테이블을 변경하거나 사용하지 못하도록 LOCK TABLE문을 통해 명시적 테이블 잠금이 획득된 경우

MDC 테이블의 경우 다음에 해당할 때 행 레벨 잠금 대신 블록 레벨 잠금을 사용합니다.

- 테이블 LOCKSIZE 속성이 BLOCKINSERT인 경우

- 분리 수준이 반복 읽기이며 액세스 플랜에 술어가 필요한 경우
- 검색한 갱신 또는 삭제 조작이 차원 컬럼에서만 술어가 필요한 경우

행 잠금 지속기간은 사용 중인 분리 수준에 따라 다릅니다.

- UR 스캔: 행 데이터가 변경되지 않는 경우 행 잠금을 보유하지 않습니다.
- CS 스캔: 행 잠금은 일반적으로 커서가 행에 있는 동안에만 보유됩니다. 어떤 경우에는 잠금이 CS 스캔 중에 전혀 보유되지 않을 수도 있습니다.
- RS 스캔: 트랜잭션 지속기간 동안에만 규정 행 잠금을 보유합니다.
- RR 스캔: 트랜잭션 지속기간 동안 모든 행 잠금을 보유합니다.

잠금 단위

한 응용프로그램이 데이터베이스 오브젝트에 대한 잠금을 보유한 경우 다른 응용프로그램이 해당 오브젝트에 액세스할 수 없습니다. 이러한 이유로 인해 잠겨 있어서 액세스할 수 없는 데이터의 양을 최소화하는 행 레벨 잠금이 블록 레벨, 데이터 파티션 레벨 또는 테이블 레벨 잠금보다 동시성을 극대화하는 데 적합합니다.

그러나 잠금에는 스토리지와 처리 시간이 필요하므로 단일 테이블 잠금이 잠금 오버헤드를 최소화합니다.

ALTER TABLE문의 LOCKSIZE절은 행, 데이터 파티션, 블록 또는 테이블 레벨에서 잠금 단위를 지정합니다. 디폴트로 행 잠금을 사용합니다. 테이블 정의에서 이 옵션을 사용하면 정상 잠금 에스컬레이션 발생을 예방하지 않습니다.

ALTER TABLE문은 전역으로 잠금을 지정하며 이 테이블에 액세스하는 모든 응용프로그램 및 사용자에게 영향을 줍니다. 개별 응용프로그램은 LOCK TABLE문을 사용하여 대신 응용프로그램 레벨에서 테이블 잠금을 지정할 수 있습니다.

ALTER TABLE문에 정의된 영구 테이블 잠금은 다음과 같은 경우 LOCK TABLE 문을 사용하는 단일 트랜잭션 테이블 잠금보다 나을 수 있습니다.

- 테이블은 읽기 전용이며 항상 S 잠금만 필요합니다. 다른 사용자가 테이블에 대한 S 잠금을 확보할 수도 있습니다.
- 보통 읽기 전용 응용프로그램이 테이블에 액세스하지만 간단한 유지보수를 위해 단일 사용자가 액세스하기도 하며 해당 사용자는 X 잠금이 필요합니다. 유지보수 프로그램을 실행하는 동안 읽기 전용 응용프로그램이 잠겨 있지만 다른 경우에는 읽기 전용 응용프로그램이 최소한의 잠금 오버헤드로 테이블에 동시에 액세스할 수 있습니다.

MDC(다차원 클러스터링) 테이블의 경우 삽입 조작 중에만 블록 레벨 잠금을 사용하려면 LOCKSIZE절을 사용하여 BLOCKINSERT를 지정할 수 있습니다. BLOCKINSERT가 지정되면 다른 모든 조작의 경우 행 레벨 잠금을 수행하지만 삽입 조작의 경우 제한적으로 수행합니다. 즉, 블록 레벨 잠금은 행 삽입 중 사용되지만 행

레벨 잠금은 레코드 ID(RID) 인덱스 갱신 중 이 인덱스에서 반복 읽기(RR) 스캔이 발생한 경우 다음 키를 잠그는 데 사용됩니다. 다음과 같은 경우 BLOCKINSERT 잠금이 유리할 수 있습니다.

- 개별 셀에 대한 대량의 삽입을 수행하는 여러 트랜잭션이 있는 경우
- 여러 트랜잭션이 동일한 셀에 동시에 삽입하는 상황이 발생하지 않거나, 사용자가 각 트랜잭션이 개별 블록에 삽입하는 것을 관여하지 않아서 각 트랜잭션이 셀 당 충분한 데이터를 삽입하는 경우 이러한 상황이 발생합니다.

잠금 속성

데이터베이스 관리 프로그램에는 여러 가지 기본 속성이 있습니다.

이러한 속성에는 다음과 같은 내용이 포함됩니다.

모드 잠금 소유자에게 허용되는 액세스 유형과 잠긴 오브젝트의 현재 사용자에게 허용되는 액세스 유형. 잠금 *상태*라고도 합니다.

오브젝트

잠기는 자원. 명시적으로 잠글 수 있는 오브젝트 유형은 테이블 뿐입니다. 데이터베이스 관리 프로그램은 기타 자원 유형(예: 행 및 테이블 스페이스)에 대한 잠금도 설정합니다. MDC(다차원 클러스터링) 테이블의 경우 블록 잠금도 설정할 수 있으며 파티션된 테이블의 경우 데이터 파티션 잠금을 설정할 수 있습니다. 잠기는 오브젝트에 따라 잠금 *단위*가 판별됩니다.

잠금 계수

잠금을 보유하는 시간의 길이. 쿼리를 실행하는 분리 수준은 잠금 계수에 영향을 줍니다.

표 11에는 잠금 모드가 표시되며 자원에 대한 제어가 증가하는 순서대로 해당 영향에 대해 설명합니다.

표 11. 잠금 모드 요약

잠금 모드	적용 가능한 오브젝트 유형	설명
IN(의도 없음)	테이블 스페이스, 블록, 테이블, 데이터 파티션	잠금 소유자는 커미트되지 않은 데이터를 포함하여 오브젝트의 모든 데이터를 읽을 수 있지만 갱신할 수는 없습니다. 다른 동시 응용프로그램이 테이블을 읽거나 갱신할 수 있습니다.
IS(부분 공유)	테이블 스페이스, 블록, 테이블, 데이터 파티션	잠금 소유자는 잠긴 테이블의 데이터를 읽을 수 있지만 이 데이터를 갱신할 수는 없습니다. 다른 응용프로그램이 테이블을 읽거나 갱신할 수 있습니다.
IX(의도를 가진 독점)	테이블 스페이스, 블록, 테이블, 데이터 파티션	잠금 소유자 및 동시 응용프로그램은 데이터를 읽고 갱신할 수 있습니다. 다른 동시 응용프로그램이 테이블을 읽고 갱신할 수 있습니다.
NS(스캔 공유)	행	잠금 소유자 및 모든 동시 응용프로그램은 잠긴 행을 읽을 수 있지만 갱신할 수는 없습니다. 이 잠금은 S 잠금 대신 테이블의 행에서 획득되며 여기서 응용프로그램의 분리 수준은 RS 또는 CS입니다.

표 11. 잠금 모드 요약 (계속)

잠금 모드	적용 가능한 오브젝트 유형	설명
NW(다음 키 약한 독점)	행	인덱스에 행을 삽입하는 경우 다음 행에서 NW 잠금을 획득합니다. 다음 행이 현재 RR 스캔에 의해 잠긴 경우에만 이러한 상태가 발생합니다. 잠금 소유자는 잠긴 행을 읽을 수 있지만 갱신할 수는 없습니다. 이 잠금 모드는 NS 잠금과도 호환 가능하다는 점을 제외하고는 X 잠금과 비슷합니다.
S(공유)	행, 블록, 테이블, 데이터 파티션	잠금 소유자 및 모든 동시 응용프로그램은 잠긴 데이터를 읽을 수 있지만 갱신할 수는 없습니다.
SIX(의도를 가진 독점 공유)	테이블, 블록, 데이터 파티션	잠금 소유자는 데이터를 읽고 갱신할 수 있습니다. 다른 동시 응용프로그램이 테이블을 읽을 수 있습니다.
U(갱신)	행, 블록, 테이블, 데이터 파티션	잠금 소유자는 데이터를 갱신할 수 있습니다. 다른 작업 단위(UOW)가 잠긴 오브젝트의 데이터를 읽을 수 있지만 갱신할 수는 없습니다.
X(독점)	행, 블록, 테이블, 버퍼 풀, 데이터 파티션	잠금 소유자는 잠긴 오브젝트의 데이터를 읽고 갱신할 수 있습니다. 커미트되지 않은 읽기(UR) 응용프로그램만 잠긴 오브젝트에 액세스할 수 있습니다.
Z(강한 독점)	테이블 스페이스, 테이블, 데이터 파티션	이 잠금은 예를 들어, 테이블이 변경 또는 삭제되었거나 테이블의 인덱스가 작성 또는 삭제되었거나 일부 테이블 재구성 유형과 같은 특정 조건의 테이블에서 획득됩니다. 다른 동시 응용프로그램이 테이블을 읽거나 갱신할 수 있습니다.

잠금에 영향을 주는 요인

여러 요인인 데이터베이스 관리 프로그램 잠금의 모드 및 단위에 영향을 줍니다.

이러한 요인은 다음과 같습니다.

- 응용프로그램이 수행하는 처리 유형
- 데이터 액세스 메소드
- 다양한 구성 매개변수의 값

잠금 및 응용프로그램 처리 유형

잠금 속성을 판별하기 위해 응용프로그램 처리를 읽기 전용, 변경 의도, 변경 및 커서 제어와 같은 유형 중 하나로 분류할 수 있습니다.

- 읽기 전용

  이 처리 유형은 원래 읽기 전용이거나 명시적 FOR READ ONLY절이 있거나 모호한 모든 SELECT문을 포함하지만 쿼리 컴파일러는 PREP 또는 BIND 명령이 지정하는 BLOCKING 옵션 값으로 인해 읽기 전용인 것으로 가정합니다. 이 유형에는 공유 잠금(IS, NS 또는 S)만 필요합니다.

- 변경 의도

  이 처리 유형은 FOR UPDATE절, USE AND KEEP UPDATE LOCKS절, USE AND KEEP EXCLUSIVE LOCKS절 또는 모호한 모든 SELECT문을 포함하지

만 쿼리 컴파일러는 변경을 의도하는 것으로 가정합니다. 이 유형은 공유 및 갱신 잠금(행의 경우 S, U 또는 X, 블록의 경우 IX, S, U 또는 X, 테이블의 경우 IX, U 또는 X)을 사용합니다.

- 변경

  이 처리 유형은 UPDATE, INSERT 및 DELETE문을 포함하지만 UPDATE WHERE CURRENT OF 또는 DELETE WHERE CURRENT OF를 포함하지 않습니다. 이 유형에는 배타적 잠금(IX 또는 X)이 필요합니다.

- 커서 제어

  이 처리 유형은 UPDATE WHERE CURRENT OF 및 DELETE WHERE CURRENT OF를 포함합니다. 이 유형에는 배타적 잠금(IX 또는 X)이 필요합니다.

subselect문의 결과에 따라 목표 테이블에서 데이터를 삽입, 갱신 또는 삭제하는 명령문은 두 가지 유형의 처리를 수행합니다. 읽기 전용 처리의 규칙은 subselect문에 데이터를 리턴하는 테이블의 잠금을 판별합니다. 변경 처리의 규칙은 목표 테이블의 잠금을 판별합니다.

잠금 및 데이터 액세스 메소드

*액세스 플랜*은 옵티마이저가 특정 테이블에서 데이터를 검색할 때 선택하는 방법입니다. 액세스 플랜은 잠금 모드에 상당한 영향을 줄 수 있습니다.

인덱스 스캔을 사용하여 특정 행을 찾는 경우 옵티마이저는 보통 테이블에 대해 행 레벨 잠금(IS)을 선택합니다. 예를 들어, EMPLOYEE 테이블에 직원 번호에 대한 인덱스(EMPNO)가 있는 경우 이 인덱스를 통한 액세스를 사용하여 단일 직원의 정보를 선택할 수 있습니다.

```
select * from employee
  where empno = '000310'
```

인덱스를 사용하지 않는 경우 전체 테이블을 순서대로 스캔하여 필요한 행을 찾아야 하며 옵티마이저가 단일 테이블 레벨 잠금(S)을 선택할 수 있습니다. 예를 들어, 컬럼 SEX에 대한 인덱스가 없는 경우 다음과 같이 테이블 스캔을 사용하여 모든 남자 직원을 선택할 수 있습니다.

```
select * from employee
  where sex = 'M'
```

주: 커서 제어 처리는 응용프로그램이 갱신 또는 삭제할 행을 찾을 때까지 기본 커서의 잠금 모드를 사용합니다. 이러한 처리 유형의 경우 커서의 잠금 모드와 관계없이 갱신 또는 삭제 조작을 수행할 때 항상 배타적 잠금을 설정합니다.

RCT(range-clustered table)에서 잠금은 표준 키 잠금과 약간 다르게 작동합니다. RCT(range-clustered table)에서 행 범위에 액세스할 때 이러한 일부 행이 비어 있는 경우에도 범위의 모든 행은 잠깁니다. 표준 키 잠금에서는 기존 데이터가 있는 행만 잠깁니다.

데이터 페이지에 대한 지연된 액세스는 행에 대한 액세스가 두 단계에서 발생함을 의미하며 이 경우 잠금 시나리오가 복잡해집니다. 잠금 획득의 시간 제어와 잠금의 지속성은 분리 수준에 따라 다릅니다. 반복 읽기(RR) 분리 수준은 트랜잭션의 끝까지 모든 잠금을 보유하므로 첫 번째 단계에서 획득한 잠금을 보유하고 두 번째 단계 중 추가 잠금을 획득할 필요가 없습니다. 읽기 안정성(RS) 및 커서 안정성(CS) 분리 수준의 경우 두 번째 단계 중 잠금을 획득합니다. 동시성을 최대화하기 위해 첫 번째 단계 중 잠금을 획득하지 않으며 모든 술어를 재적용하여 규정 행만 리턴됩니다.

잠금 유형 호환성

잠금 호환성은 하나의 응용프로그램이 오브젝트에 대한 잠금을 보유하고 다른 응용프로그램이 동일한 오브젝트에 대한 잠금을 요청할 때 논점이 됩니다. 두 개의 잠금 모드가 호환 가능한 경우, 오브젝트에 대한 두 번째 잠금 요청에 권한이 부여될 수 있습니다.

요청된 잠금의 잠금 모드가 이미 보유된 잠금과 호환되지 않는 경우, 잠금 요청에 권한이 부여될 수 없습니다. 대신, 첫 번째 응용프로그램에서 잠금을 해제하고, 기타 모든 기존 호환되지 않는 잠금이 해제될 때까지 요청이 대기해야 합니다.

표 12에서는 호환 가능한 잠금 유형(**예**로 표시됨) 및 호환되지 않는 잠금 유형(**아니오**로 표시됨)을 보여줍니다. 요청자가 잠금을 대기할 때 시간종료가 발생할 수 있습니다.

표 12. 잠금 유형 호환성

	보유된 자원 상태										
요청되는 상태	**없음**	**IN**	**IS**	**NS**	**S**	**IX**	**SIX**	**U**	**X**	**Z**	**NW**
없음	예	예	예	예	예	예	예	예	예	예	예
IN(의도 없음)	예	예	예	예	예	예	예	예	예	아니오	예
IS(부분 공유)	예	예	예	예	예	예	예	예	아니오	아니오	아니오
NS(스캔 공유)	예	예	예	예	예	아니오	아니오	예	아니오	아니오	예
S(공유)	예	예	예	예	예	아니오	아니오	예	아니오	아니오	아니오
IX(의도를 가진 독점)	예	예	예	아니오	아니오	예	아니오	아니오	아니오	아니오	아니오
SIX(의도를 가진 독점 공유)	예	예	예	아니오	아니오	아니오	아니오	아니오	아니오	아니오	아니오
U(갱신)	예	예	예	예	예	아니오	아니오	아니오	아니오	아니오	아니오
X(독점)	예	예	아니오	아니오	아니오	아니오	아니오	아니오	아니오	아니오	아니오
Z(강한 독점)	예	아니오	아니오	아니오	아니오	아니오	아니오	아니오	아니오	아니오	아니오
NW(다음 키 약한 독점)	예	예	아니오	예	아니오	아니오	아니오	아니오	아니오	아니오	아니오

다음 키 잠금

인덱스에 키를 삽입하는 동안 인덱스의 새 키 다음에 표시되는 키에 해당하는 행은 반복 읽기(RR) 인덱스 스캔으로 해당 행이 현재 잠겨 있는 경우에만 잠깁니다.

다음 키 잠금에 사용되는 잠금 모드는 NW(next key weak exclusive)입니다. 이 다음 키 잠금은 키 삽입이 발생하기 전 즉, 행이 테이블에 삽입되기 전에 발생합니다.

키 삽입은 행 갱신으로 인해 해당 행의 인덱스 키 값이 변경된 경우에도 발생합니다. 원래의 키 값이 삭제된 것으로 표시되었으며 새 키 값이 인덱스에 삽입되기 때문입니다. 인덱스의 포함 컬럼에만 영향을 주는 갱신의 경우 키를 적절하게 갱신할 수 있으며 다음 키 잠금은 발생하지 않습니다.

RR 스캔 중 스캔 범위 끝 뒤에 표시되는 키에 해당하는 행은 S 모드로 잠깁니다. 스캔 범위 끝 뒤에 키가 표시되지 않은 경우 테이블 끝 잠금을 획득하여 인덱스 끝을 잠급니다. 파티션된 테이블의 파티션된 인덱스의 경우, 인덱스 끝에 대해 오직 하나의 잠금 대신 각 인덱스 파티션 끝을 잠그기 위해 잠금이 획득됩니다. 스캔 범위 끝에 이어지는 키가 삭제됨으로 표시된 경우, 다음 조치 중 하나가 발생합니다.

- 스캔은 삭제됨으로 표시되지 않은 키를 찾을 때까지 대응하는 행을 계속 잠급니다.
- 스캔은 해당 키의 대응하는 행을 잠급니다.
- 스캔은 인덱스 끝을 잠급니다.

표준 테이블에 대한 잠금 모드 및 액세스 플랜

표준 테이블이 갖는 잠금의 유형은 적용되는 분리 수준 및 사용되는 데이터 액세스 플랜에 따라 달라집니다.

다음 테이블에서는 여러 액세스 플랜에 대한 각 분리 수준에서 표준 테이블에 대해 얻어지는 잠금의 유형을 보여줍니다. 각 항목은 테이블 잠금과 행 잠금의 두 부분으로 되어 있습니다. 하이픈은 특정 잠금 단위가 사용 가능하지 않음을 표시합니다.

테이블 7-12는 행 목록이 다중 인덱스를 사용하여 추가로 규정되거나 효율적인 프리페치를 위해 정렬될 수 있도록 데이터 페이지 읽기가 지연될 경우 얻어지는 잠금의 유형을 보여줍니다.

- 테이블 1. 술어가 없는 테이블 스캔에 대한 잠금 모드
- 테이블 2. 술어가 있는 테이블 스캔에 대한 잠금 모드
- 테이블 3. 술어가 없는 RID 인덱스 스캔에 대한 잠금 모드
- 테이블 4. 단일 규정 행이 있는 RID 인덱스 스캔에 대한 잠금 모드
- 테이블 5. 시작 및 중지 술어만 있는 RID 인덱스 스캔에 대한 잠금 모드
- 테이블 6. 인덱스 및 기타 술어(sargs, resids)만 있는 RID 인덱스 스캔에 대한 잠금 모드

- 테이블 7. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 RID 인덱스 스캔
- 테이블 8. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 RID 인덱스 스캔 후
- 테이블 9 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어(sargs, resids)가 있는 RID 인덱스 스캔
- 테이블 10. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어(sargs, resids)가 있는 RID 인덱스 스캔 후
- 테이블 11. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 RID 인덱스 스캔
- 테이블 12. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 RID 인덱스 스캔 후

주:

1. 블록 레벨 잠금은 다차원 클러스터링(MDC) 테이블에 대해서도 사용 가능합니다.
2. 잠금 모드는 SELECT문의 *lock-request-clause*를 사용하여 명시적으로 변경할 수 있습니다.

표 13. 술어가 없는 테이블 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	S/-	U/-	SIX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

표 14. 술어가 있는 테이블 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	S/-	U/-	SIX/X	U/-	SIX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

주: UR 분리 수준에서, 인덱스의 Include 컬럼에 술어가 있는 경우, 분리 수준이 CS로 업그레이드되고 잠금이 IS 테이블 잠금 또는 NS 행 잠금으로 업그레이드됩니다.

표 15. 술어가 없는 RID 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	S/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

표 16. 단일 규정 행이 있는 RID 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S	IX/U	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

표 17. 시작 및 중지 술어만 있는 RID 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

표 18. 인덱스 및 기타 술어(sargs, resids)만 있는 RID 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

표 19. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 RID 인덱스 스캔

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S	IX/S		X/-	
RS	IN/-	IN/-		IN/-	

표 19. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 RID 인덱스 스캔 (계속)

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

표 20. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 RID 인덱스 스캔 후

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IN/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

표 21. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어(sargs, resids)가 있는 RID 인덱스 스캔

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S	IX/S		IX/S	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

표 22. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어(sargs, resids)가 있는 RID 인덱스 스캔 후

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IN/-	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

표 23. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 RID 인덱스 스캔

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S	IX/S		IX/X	

표 23. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 RID 인덱스 스캔 (계속)

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

표 24. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 RID 인덱스 스캔 후

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IN/-	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IS/-	IX/U	IX/X	IX/U	IX/X

MDC 테이블 및 RID 인덱스 스캔에 대한 잠금 모드

테이블 또는 RID 인덱스 스캔 동안 다차원 클러스터링(MDC) 테이블이 갖는 잠금의 유형은 적용되는 분리 수준 및 사용되는 데이터 액세스 플랜에 따라 달라집니다.

다음 테이블에서는 여러 액세스 플랜에 대한 각 분리 수준에서 MDC 테이블에 대해 얻어지는 잠금의 유형을 보여줍니다. 각 항목은 테이블 잠금, 블록 잠금 및 행 잠금의 세 부분으로 되어 있습니다. 하이픈은 특정 잠금 단위가 사용 가능하지 않음을 표시합니다.

테이블 9-14는 데이터 페이지 읽기가 지연될 경우 RID 인덱스 스캔에 대해 얻어지는 잠금의 유형을 표시합니다. UR 분리 수준에서, 인덱스의 Include 컬럼에 술어가 있는 경우, 분리 수준이 CS로 업그레이드되고 잠금이 IS 테이블 잠금, IS 블록 잠금 또는 NS 행 잠금으로 업그레이드됩니다.

- 테이블 1. 술어가 없는 테이블 스캔에 대한 잠금 모드
- 테이블 2. 차원 컬럼에만 술어가 있는 테이블 스캔에 대한 잠금 모드
- 테이블 3. 기타 술어(sargs, resids)가 있는 테이블 스캔에 대한 잠금 모드
- 테이블 4. 술어가 없는 RID 인덱스 스캔에 대한 잠금 모드
- 테이블 5. 단일 규정 행이 있는 RID 인덱스 스캔에 대한 잠금 모드
- 테이블 6 시작 및 중지 술어만 있는 RID 인덱스 스캔에 대한 잠금 모드
- 테이블 7. 인덱스 술어만 있는 RID 인덱스 스캔에 대한 잠금 모드
- 테이블 8. 기타 술어(sargs, resids)가 있는 RID 인덱스 스캔에 대한 잠금 모드

- 테이블 9. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 RID 인덱스 스캔
- 테이블 10. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 RID 인덱스 스캔 후
- 테이블 11. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어(sargs, resids)가 있는 RID 인덱스 스캔
- 테이블 12. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어(sargs, resids)가 있는 RID 인덱스 스캔 후
- 테이블 13. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 RID 인덱스 스캔
- 테이블 14. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 RID 인덱스 스캔 후

주: 잠금 모드는 SELECT문의 *lock-request-clause*를 사용하여 명시적으로 변경할 수 있습니다.

표 25. 술어가 없는 테이블 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	S/-/-	U/-/-	SIX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/U	IX/X/-	IX/I/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

표 26. 차원 컬럼에만 술어가 있는 테이블 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/X/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/U/-	X/X/-

표 27. 기타 술어(sargs, resids)가 있는 테이블 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

표 28. 술어가 없는 RID 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	S/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

표 29. 단일 규정 행이 있는 RID 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/IS/S	IX/IX/U	IX/IX/X	X/X/X	X/X/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

표 30. 시작 및 중지 술어만 있는 RID 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/IS/S	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

표 31. 인덱스 술어만 있는 RID 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

표 32. 기타 술어(sargs, resids)가 있는 RID 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

표 32. 기타 술어(sargs, resids)가 있는 RID 인덱스 스캔에 대한 잠금 모드 (계속)

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

표 33. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 RID 인덱스 스캔

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S/S	IX/IX/S		X/-/-	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

표 34. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 RID 인덱스 스캔 후

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IN/IN/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

표 35. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어(sargs, resids)가 있는 RID 인덱스 스캔

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S/-	IX/IX/S		IX/IX/S	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

표 36. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어(sargs, resids)가 있는 RID 인덱스 스캔 후

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

표 36. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어(sargs, resids)가 있는 RID 인덱스 스캔 후 (계속)

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

표 37. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 RID 인덱스 스캔

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/IS/S	IX/IX/S		IX/IX/X	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

표 38. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 RID 인덱스 스캔 후

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IS/-/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

MDC 블록 인덱스 스캔에 대한 잠금 모드

블록 인덱스 스캔 동안 다차원 클러스터링(MDC) 테이블이 갖는 잠금의 유형은 적용되는 분리 수준 및 사용되는 데이터 액세스 플랜에 따라 달라집니다.

다음 테이블에서는 여러 액세스 플랜에 대한 각 분리 수준에서 MDC 테이블에 대해 얻어지는 잠금의 유형을 보여줍니다. 각 항목은 테이블 잠금, 블록 잠금 및 행 잠금의 세 부분으로 되어 있습니다. 하이픈은 특정 잠금 단위가 사용 가능하지 않음을 표시합니다.

테이블 5-12는 데이터 페이지 읽기가 지연될 경우 블록 인덱스 스캔에 대해 얻어지는 잠금의 유형을 표시합니다.

- 테이블 1. 술어가 없는 인덱스 스캔에 대한 잠금 모드
- 테이블 2.차원 컬럼에만 술어가 있는 인덱스 스캔에 대한 잠금 모드
- 테이블 3. 시작 및 중지 술어만 있는 인덱스 스캔에 대한 잠금 모드

- 테이블 4. 술어가 있는 인덱스 스캔에 대한 잠금 모드
- 테이블 5. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 블록 인덱스 스캔
- 테이블 6. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 블록 인덱스 스캔 후
- 테이블 7. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 차원 컬럼에만 술어가 있는 블록 인덱스 스캔
- 테이블 8. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 차원 컬럼에만 술어가 있는 블록 인덱스 스캔 후
- 테이블 9. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 블록 인덱스 스캔
- 테이블 10. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 블록 인덱스 스캔 후
- 테이블 11. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 기타 술어(sargs, resids)가 있는 블록 인덱스 스캔
- 테이블 12. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 기타 술어(sargs, resids)가 있는 블록 인덱스 스캔 후

주: 잠금 모드는 SELECT문의 *lock-request-clause*를 사용하여 명시적으로 변경할 수 있습니다.

표 39. 술어가 없는 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	S/--/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/--	X/X/--

표 40. 차원 컬럼에만 술어가 있는 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

표 41. 시작 및 중지 술어만 있는 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S/-	IX/IX/S	IX/IX/S	IX/IX/S	IX/IX/S
RS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
CS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-

표 42. 술어가 있는 인덱스 스캔에 대한 잠금 모드

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

표 43. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 블록 인덱스 스캔

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S/--	IX/IX/S		X/--/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

표 44. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 술어가 없는 블록 인덱스 스캔 후

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IN/IN/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	X/X/--	X/X/--

표 45. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 차원 컬럼에만 술어가 있는 블록 인덱스 스캔

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S/--	IX/IX/--		IX/S/--	

표 45. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 차원 컬럼에만 술어가 있는 블록 인덱스 스캔 (계속)

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RS	IS/IS/NS	IX/--/--		IX/--/--	
CS	IS/IS/NS	IX/--/--		IX/--/--	
UR	IN/IN/--	IX/--/--		IX/--/--	

표 46. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 차원 컬럼에만 술어가 있는 블록 인덱스 스캔 후

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/S/--	IX/X/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--

표 47. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 블록 인덱스 스캔

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S/--	IX/IX/--		IX/X/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

표 48. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 시작 및 중지 술어만 있는 블록 인덱스 스캔 후

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IN/IN/--	IX/IX/X		IX/X/--	
RS	IS/IS/NS	IN/IN/--		IN/IN/--	
CS	IS/IS/NS	IN/IN/--		IN/IN/--	
UR	IS/--/--	IN/IN/--		IN/IN/--	

표 49. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 기타 술어(sargs, resids)가 있는 블록 인덱스 스캔

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IS/S/--	IX/IX/--		IX/IX/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

표 50. 지연된 데이터 페이지 액세스에 사용된 인덱스 스캔에 대한 잠금 모드: 기타 술어(sargs, resids)가 있는 블록 인덱스 스캔 후

분리 수준	읽기 전용 및 앰비규어스 스캔	커서 조작		검색된 갱신 또는 삭제	
		스캔	Where current of	스캔	갱신 또는 삭제
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

파티션된 테이블에서의 잠금 동작

전체 테이블 잠금 외에도, 파티션된 테이블의 각 데이터 파티션에 대한 잠금이 있습니다.

이는 파티션되지 않은 테이블에 비해 더 미세한 세밀도와 증가된 동시성을 제공합니다. 데이터 파티션 잠금은 테이블 함수, 관리 뷰, 이벤트 모니터 및 db2pd 명령의 출력에서 식별됩니다.

테이블에 액세스할 때 테이블 잠금이 우선 획득된 후 필요에 따라 파티션 잠금이 획득됩니다. 액세스 메소드 및 분리 수준에서 결과 세트에 표시되지 않은 데이터 파티션의 잠금을 요구할 수도 있습니다. 이러한 데이터 파티션 잠금은 획득된 후 테이블 잠금만큼 오래 보유될 수도 있습니다. 예를 들어, 인덱스에 대한 커서 안정성(CS) 스캔에서 이전에 액세스한 데이터 파티션에 대한 잠금을 유지하여 나중에 데이터 파티션 잠금을 다시 획득하는 비용을 줄일 수 있습니다.

데이터 파티션 잠금은 또한 테이블 스페이스에 대한 액세스를 보장하는 비용도 감당합니다. 파티션되지 않은 테이블의 경우, 테이블 스페이스 액세스는 테이블 잠금에 의해 처리됩니다. 테이블 레벨에서 독점 또는 공유 잠금이 있는 경우라도 데이터 파티션 잠금이 발생합니다.

보다 미세한 세밀도를 통해 한 트랜잭션에서 특정 데이터 파티션에 대한 배타적 액세스를 가질 수 있고 다른 트랜잭션에서 다른 데이터 파티션에 액세스하는 동안 행 잠금을 피할 수 있습니다. 이는 모두 갱신에 대해 선택한 플랜의 결과이거나 데이터 파티션 레

벨에 대한 잠금의 에스컬레이션 때문일 수 있습니다. 많은 액세스 메소드에서 테이블 잠금은 데이터 파티션이 공유 또는 독점 모드로 잠긴다 하더라도, 일반적으로 의도 잠금입니다. 이는 동시성을 증가시켜 줍니다. 그러나 데이터 파티션 레벨에서 비의도 잠금이 필요하고 플랜에서 모든 데이터 파티션에 액세스할 수 있다고 표시할 경우, 동시 트랜잭션 간 데이터 파티션 교착 상태를 방지하기 위해 테이블 레벨에서 비의도 잠금이 선택될 수도 있습니다.

LOCK TABLE문

파티션 테이블의 경우, LOCK TABLE문에 의해 획득되는 유일한 잠금은 테이블 레벨 잠금입니다. 이것은 후속 데이터 처리 언어(DML) 명령문에 의한 행 잠금을 예방하고, 행, 블록 또는 데이터 파티션 레벨에서의 교착 상태를 방지합니다. IN EXCLUSIVE MODE 옵션을 사용하여 인덱스 갱신 시 배타적 액세스를 보장할 수 있으며, 이는 대형 갱신 동안 인덱스의 증가를 제한하는 데 유용합니다.

ALTER TABLE문에서 LOCKSIZE TABLE 옵션의 효과

LOCKSIZE TABLE 옵션은 의도 잠금 없는 독점 또는 공유 모드에서 테이블이 잠기도록 보장합니다. 파티션된 테이블의 경우, 이 잠금 전략은 테이블 잠금과 데이터 파티션 잠금에 모두 적용됩니다.

행 및 블록 레벨 잠금 에스컬레이션

파티션된 테이블에서 행 및 블록 레벨 잠금은 데이터 파티션 레벨로 에스컬레이션될 수 있습니다. 이렇게 될 경우, 데이터 파티션이 공유, 독점 또는 강한 독점 모드로 에스컬레이션된다 하더라도 다른 데이터 파티션이 영향을 받지 않으므로, 다른 트랜잭션에서 테이블에 더 잘 액세스할 수 있습니다. 에스컬레이션에 대한 통지 로그 항목에는 테이블의 이름 및 영향을 받는 데이터 파티션이 포함됩니다.

파티션되지 않은 인덱스에 대한 배타적 액세스는 잠금 에스컬레이션에 의해 보장될 수 없습니다. 배타적 액세스의 경우, 다음 조건 중 하나가 참이어야 합니다.

- 명령문이 독점 테이블 레벨 잠금을 사용해야 함
- 명시적 LOCK TABLE IN EXCLUSIVE MODE문을 실행함
- 테이블에 LOCKSIZE TABLE 속성이 있어야 함

파티션된 인덱스의 경우, 인덱스 파티션에 대한 배타적 액세스는 독점 또는 강한 독점 액세스 모드로의 데이터 파티션 잠금 에스컬레이션으로 보장됩니다.

잠금 정보 해석

SNAPLOCK 관리 뷰가 파티션된 테이블에 대해 리턴되는 잠금 정보를 해석하는 데 도움이 될 수 있습니다. 오프라인 인덱스 재구성 동안 다음과 같은 SNAPLOCK 관리 뷰가 캡처되었습니다.

```
SELECT SUBSTR(TABNAME, 1, 15) TABNAME, TAB_FILE_ID, SUBSTR(TBSP_NAME, 1, 15) TBSP_NAME, DATA_PARTITION_ID, LOCK_OBJECT_TYPE,
LOCK_MODE, LOCK_ESCALATION FROM SYSIBMADM.SNAPLOCK where TABNAME like 'TP1' and LOCK_OBJECT_TYPE like 'TABLE_%'
ORDER BY TABNAME, DATA_PARTITION_ID, LOCK_OBJECT_TYPE, TAB_FILE_ID, LOCK_MODE

TABNAME         TAB_FILE_ID          TBSP_NAME       DATA_PARTITION_ID LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_ESCALATION
--------------- -------------------- --------------- ----------------- ------------------ ---------- ---------------
TP1                           32768 -                              -1 TABLE_LOCK         Z                       0
TP1                               4 USERSPACE1                      0 TABLE_PART_LOCK    Z                       0
TP1                               5 USERSPACE1                      1 TABLE_PART_LOCK    Z                       0
TP1                               6 USERSPACE1                      2 TABLE_PART_LOCK    Z                       0
TP1                               7 USERSPACE1                      3 TABLE_PART_LOCK    Z                       0
TP1                               8 USERSPACE1                      4 TABLE_PART_LOCK    Z                       0
TP1                               9 USERSPACE1                      5 TABLE_PART_LOCK    Z                       0
TP1                              10 USERSPACE1                      6 TABLE_PART_LOCK    Z                       0
TP1                              11 USERSPACE1                      7 TABLE_PART_LOCK    Z                       0
TP1                              12 USERSPACE1                      8 TABLE_PART_LOCK    Z                       0
TP1                              13 USERSPACE1                      9 TABLE_PART_LOCK    Z                       0
TP1                              14 USERSPACE1                     10 TABLE_PART_LOCK    Z                       0
TP1                              15 USERSPACE1                     11 TABLE_PART_LOCK    Z                       0
TP1                               4 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                               5 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                               6 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                               7 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                               8 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                               9 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                              10 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                              11 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                              12 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                              13 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                              14 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                              15 USERSPACE1                      - TABLE_LOCK         Z                       0
TP1                              16 USERSPACE1                      - TABLE_LOCK         Z                       0

  26 record(s) selected.
```

이 예에서는 파티션된 테이블 TP1에 대한 액세스 및 동시성을 제어하는 데 DATA_PARTITION_ID -1 및 TABLE_LOCK 유형의 잠금 오브젝트가 사용됩니다. 각 데이터 파티션에 대한 대부분의 액세스 및 동시성을 제어하는 데에는 TABLE_PART_LOCK 유형의 잠금 오브젝트가 사용됩니다.

이 출력에는 DATA_PARTITION_ID에 대한 값이 없는 TABLE_LOCK 유형의 추가 잠금 오브젝트가 캡처되어 있습니다(TAB_FILE_ID 4-16). 이 유형의 잠금(오브젝트의 TAB_FILE_ID 및 TBSP_NAME이 파티션된 테이블의 데이터 파티션 또는 인덱스에 해당하는)은 온라인 백업 유틸리티에서 동시성을 제어하는 데 사용될 수 있습니다.

잠금 변환

이미 보유된 잠금 모드를 변경하는 것을 *잠금 변환*이라고 합니다.

잠금 변환은 프로세스에서 이미 잠금을 보유하고 있는 데이터 오브젝트에 액세스하고 액세스 모드에 이미 보유된 잠금보다 더 제한적인 잠금이 필요할 경우 발생합니다. 프로세스에서 쿼리를 통해 간접적으로 동일한 데이터 오브젝트에 대해 잠금을 여러 번 요청할 수 있지만, 지정된 시간에 데이터 오브젝트에 대해 하나의 잠금만 보유할 수 있습니다.

일부 잠금 모드는 테이블에만 적용되고, 기타 잠금 모드는 행, 블록 또는 데이터 파티션에만 적용됩니다. 행 또는 블록의 경우, X 잠금이 필요하고 S 또는 U 잠금이 보유된 경우 일반적으로 변환이 발생합니다.

IX 및 S 잠금은 잠금 변환에서 특수한 경우입니다. 어느 쪽도 서로에 대해 더 제한적인 것으로 간주되지 않으므로, 이러한 잠금 중 하나가 보유되고 다른 하나가 필요한 경우, 변환 결과는 SIX(의도를 가진 독점 공유) 잠금이 됩니다. 기타 모든 변환에서는 요청된 모드가 더 제한적인 경우 요청된 잠금 모드가 보유된 잠금 모드가 됩니다.

쿼리에서 행을 갱신할 경우 이중 변환이 발생할 수도 있습니다. 행이 인덱스 액세스를 통해 읽히고 S로 잠긴 경우, 이 행을 포함하는 테이블에 적용되는 의도 잠금이 있습니다. 그러나 잠금 유형이 IX 대신 IS이고, 행이 그 후에 변경되는 경우, 테이블 잠금은 IX로 변환되고 행 잠금은 X로 변환됩니다.

잠금 변환은 보통 쿼리가 실행되면서 내재적으로 발생합니다. 시스템 모니터 요소 **lock_current_mode** 및 **lock_mode**가 데이터베이스에서 발생하는 잠금 변환에 대한 정보를 제공할 수 있습니다.

잠금 대기 및 시간종료

잠금 시간종료 감지는 응용프로그램에서 잠금이 해제되기를 무기한으로 대기하는 것을 방지하는 데이터베이스 관리 프로그램 기능입니다.

예를 들어, 트랜잭션에서 다른 사용자의 응용프로그램에서 보유한 잠금을 대기하고 있을 수 있는데 다른 사용자가 응용프로그램에서 잠금을 해제하는 트랜잭션을 커미트하도록 허용하지 않고 워크스테이션을 나갔습니다. 이런 경우 응용프로그램 정체를 피하려면 **locktimeout** 데이터베이스 구성 매개변수를 응용프로그램이 잠금을 얻기 위해 대기해야 하는 최대 시간으로 설정하십시오.

이 매개변수를 설정하면 특히 분산 작업 단위(DUOW) 응용프로그램에서 전역 교착 상태를 피하는 데 도움이 됩니다. 잠금 요청이 보류되는 동안의 시간이 **locktimeout** 값보다 큰 경우, 요청 응용프로그램에 오류가 리턴되고 해당 트랜잭션이 롤백됩니다. 예를 들어, APPL1에서 APPL2가 이미 보유한 잠금을 획득하려고 시도할 경우, 시간종료 기간이 만기되면 APPL1이 이유 코드 68과 함께 SQLCODE -911(SQLSTATE 40001)을 수신합니다. **locktimeout**의 디폴트값은 -1이고, 이는 잠금 시간종료 감지가 사용되지 않음을 의미합니다.

테이블, 행, 데이터 파티션 및 다차원 클러스터링(MDC) 블록 잠금의 경우, 응용프로그램에서 CURRENT LOCK TIMEOUT 특수 레지스터의 값을 변경하여 **locktimeout** 값을 겹쳐쓸 수 있습니다.

잠금 시간종료에 대한 보고서 파일을 생성하려면 **DB2_CAPTURE_LOCKTIMEOUT** 레지스트리 변수를 ON으로 설정하십시오. 잠금 시간종료 보고서에는 잠금 시간종료로

끝나는 잠금 경합과 관련된 키 응용프로그램에 대한 정보뿐 아니라 잠금 이름, 잠금 유형, 행 ID, 테이블 스페이스 ID 및 테이블 ID와 같은 잠금에 대한 세부사항도 포함됩니다. 이 변수는 사용되지 않으며, CREATE EVENT MONITOR FOR LOCKING 문을 사용하여 잠금 시간종료 이벤트를 수집하는 새 메소드가 있으므로 추후 릴리스에서는 제거될 수 있습니다.

db2diag 로그 파일에 잠금-요청 시간종료에 대한 자세한 정보를 로그하려면, **diaglevel** 데이터베이스 관리 프로그램 구성 매개변수 값을 4로 설정하십시오. 로그된 정보에는 잠금을 보유하고 있는 응용프로그램, 잠금 모드 및 잠긴 오브젝트의 이름이 포함됩니다. 현재 동적 SQL 또는 XQuery문이나 정적 패키지 이름이 로그될 수도 있습니다. 동적 SQL 또는 XQuery문은 **diaglevel** 4에서만 로그됩니다.

잠금 대기 정보 시스템 모니터 요소나 **db.apps_waiting_locks** Health 표시기에서 잠금 대기 및 잠금 시간종료에 대한 추가 정보를 얻을 수 있습니다.

잠금 대기 모드 전략 지정

세션에서 즉시 얻을 수 없는 잠금이 필요한 경우 사용되는 잠금 대기 모드 전략을 지정할 수 있습니다.

이 전략은 세션에서 다음을 수행할지 여부를 표시합니다.

- 잠금을 얻을 수 없는 경우 SQLCODE 및 SQLSTATE 리턴
- 무기한으로 잠금 대기
- 지정된 시간 동안 잠금 대기
- 잠금에 대기할 때 **locktimeout** 데이터베이스 구성 매개변수 값 사용

잠금 대기 모드 전략은 CURRENT LOCK TIMEOUT 특수 레지스터의 값을 변경하는 SET CURRENT LOCK TIMEOUT문을 통해 지정됩니다. 이 특수 레지스터는 잠금을 얻을 수 없음을 표시하는 오류를 리턴하기 전 잠금에 대기하는 시간(초)을 지정합니다.

일반 잠금 방법을 사용하면 응용프로그램이 서로 블로킹될 수 있습니다. 한 응용프로그램에서 다른 응용프로그램의 잠금 해제에 대기해야 할 경우 이런 일이 일어납니다. 이러한 블로킹의 영향을 처리하는 전략에서는 일반적으로 최대 허용 가능한 블록 기간을 지정하는 메커니즘을 제공합니다. 이것은 응용프로그램이 잠금 없이 리턴하기 전 대기하는 시간입니다. 이전에는 이것이 데이터베이스 레벨에서 **locktimeout** 데이터베이스 구성 매개변수의 값을 변경해서만 가능했습니다.

locktimeout의 값은 모든 잠금에 적용되지만, 잠금 대기 모드 전략에 의해 영향을 받는 잠금 유형에는 행, 테이블, 인덱스 키 및 다차원 클러스터링(MDC) 블록 잠금이 포함됩니다.

교착 상태

두 응용프로그램이 상대방에 필요한 데이터를 잠근 경우 교착 상태가 발생하므로 어느 응용프로그램도 계속 실행할 수 없는 상황이 발생합니다.

예를 들어, 그림 23에는 동시에 실행 중인 두 응용프로그램인 응용프로그램 A와 응용프로그램 B가 있습니다. 응용프로그램 A의 첫 번째 트랜잭션은 테이블 1의 첫 번째 행을 갱신하는 것이고 두 번째 트랜잭션은 테이블 2의 두 번째 행을 갱신하는 것입니다. 응용프로그램 B는 우선 테이블 2의 두 번째 행을 갱신한 후 테이블 1의 첫 번째 행을 갱신합니다. T1 시간에 응용프로그램 A는 테이블 1의 첫 번째 행을 잠급니다. 동시에 응용프로그램 B는 테이블 2의 두 번째 행을 잠급니다. T2 시간에 응용프로그램 A는 테이블 2의 두 번째 행에 대한 잠금을 요청합니다. 그러나 동시에 응용프로그램 B는 테이블 1의 첫 번째 행을 잠그려고 시도합니다. 응용프로그램 A는 테이블 2의 두 번째 행에 대한 갱신을 완료할 때까지 테이블 1의 첫 번째 행에 대한 잠금을 해제하지 않으므로 교착 상태가 발생합니다. 이들 중 하나가 데이터에 대한 잠금을 해제할 때까지 응용프로그램이 대기합니다.

교착 상태 개념

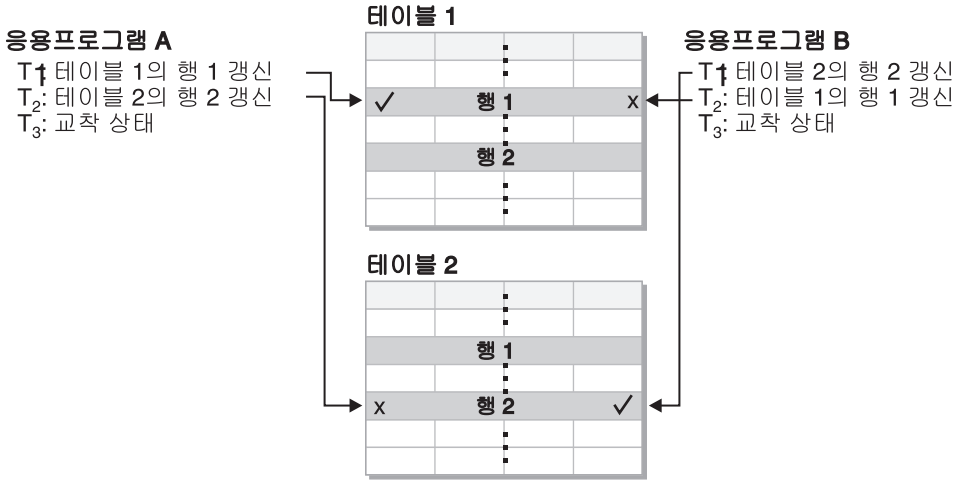


그림 23. 응용프로그램들 간 교착 상태

응용프로그램은 필요한 데이터에 대한 잠금을 직접 해제하지 않으므로 교착 상태를 중단하려면 교착 상태 검출기 프로세스가 필요합니다. 교착 상태 검출기는 잠금 대기 중인 에이전트에 대한 정보를 모니터하고 **dlchktime** 데이터베이스 구성 매개변수에 지정된 간격에 따라 시작됩니다.

교착 상태를 발견하면 교착 상태 검출기는 *희생(victim) 프로세스*가 롤백될 때 하나의 교착 상태 프로세스를 임의로 선택합니다. 희생 프로세스가 시작되고 이유 코드 2의 SQLCODE -911(SQLSTATE 40001)을 호출 응용프로그램으로 리턴합니다. 데이터베

이스 관리 프로그램은 선택된 프로세스에서 커미트되지 않은 트랜잭션을 자동으로 롤백합니다. 롤백 조작이 완료되면 희생 프로세스에 속한 잠금이 해제되고 교착 상태에 관련된 다른 프로세스를 계속할 수 있습니다.

성능 향상을 위해 **dlchktime**의 적합한 값을 선택하십시오. 간격이 너무 짧으면 불필요한 오버헤드가 발생하며 간격이 너무 길면 교착 상태가 지체됩니다.

파티션된 데이터베이스 환경에서 **dlchktime**의 값은 카탈로그 데이터베이스 파티션에만 적용됩니다. 대량의 교착 상태가 발생한 경우 잠금 대기 및 통신 대기를 고려하여 **dlchktime**의 값을 늘리십시오.

응용프로그램이 나중에 갱신할 데이터를 읽을 때 교착 상태를 방지하기 위해 다음을 수행하십시오.

- 선택 조작을 수행할 때 FOR UPDATE절을 사용하십시오. 이 절을 사용하면 프로세스가 데이터를 읽으려고 시도할 때 U 잠금이 설정되고 행 블로킹이 허용되지 않습니다.
- 쿼리에서 WITH RR 또는 WITH RS 및 USE AND KEEP UPDATE LOCKS절을 사용하십시오. 이러한 절을 사용하면 프로세스가 데이터를 읽으려고 시도할 때 U 잠금이 설정되며 행 블로킹이 허용되지 않습니다.

페더레이티드 시스템에서 응용프로그램이 요청하는 데이터는 데이터 소스의 교착 상태로 인해 사용 불가능해질 수 있습니다. 이러한 상태가 발생하면 DB2 서버는 데이터 소스의 교착 상태 처리 기능에 의존합니다. 여러 데이터 소스에서 교착 상태가 발생한 경우 DB2 서버는 데이터 소스 시간종료 메커니즘에 의존하여 교착 상태를 중단합니다.

교착 상태에 대한 자세한 정보를 로그하려면 **diaglevel** 데이터베이스 관리 프로그램 구성 매개변수의 값을 4로 설정하십시오. 로그된 정보에는 잠근 오브젝트의 이름, 잠금 모드 및 잠금을 보유 중인 응용프로그램이 포함됩니다. 현재 동적 SQL 및 XQuery문 또는 정적 패키지 이름도 로그될 수 있습니다.

쿼리 최적화

쿼리 최적화는 응용프로그램 성능에 영향을 미치는 인수 중 하나입니다. 데이터베이스 응용프로그램의 성능을 최대화하는 데 도움이 되는 쿼리 최적화 고려사항에 대한 자세한 내용은 이 절을 검토하십시오.

SQL 및 XQuery 컴파일러 프로세스

SQL 및 XQuery 컴파일러는 실행할 수 있는 액세스 플랜을 작성하기 위해 여러 단계를 수행합니다.

*쿼리 그래프 모델*은 214 페이지의 그림 24에 표시되어 있으며 아래에 설명되어 있는 단계를 통해 처리되는 쿼리를 표시한 메모리에 있는 내부 데이터베이스입니다. 일부 단계

는 페더레이티드 데이터베이스에 대해 실행되는 쿼리의 경우에만 발생한다는 점을 참고하십시오.

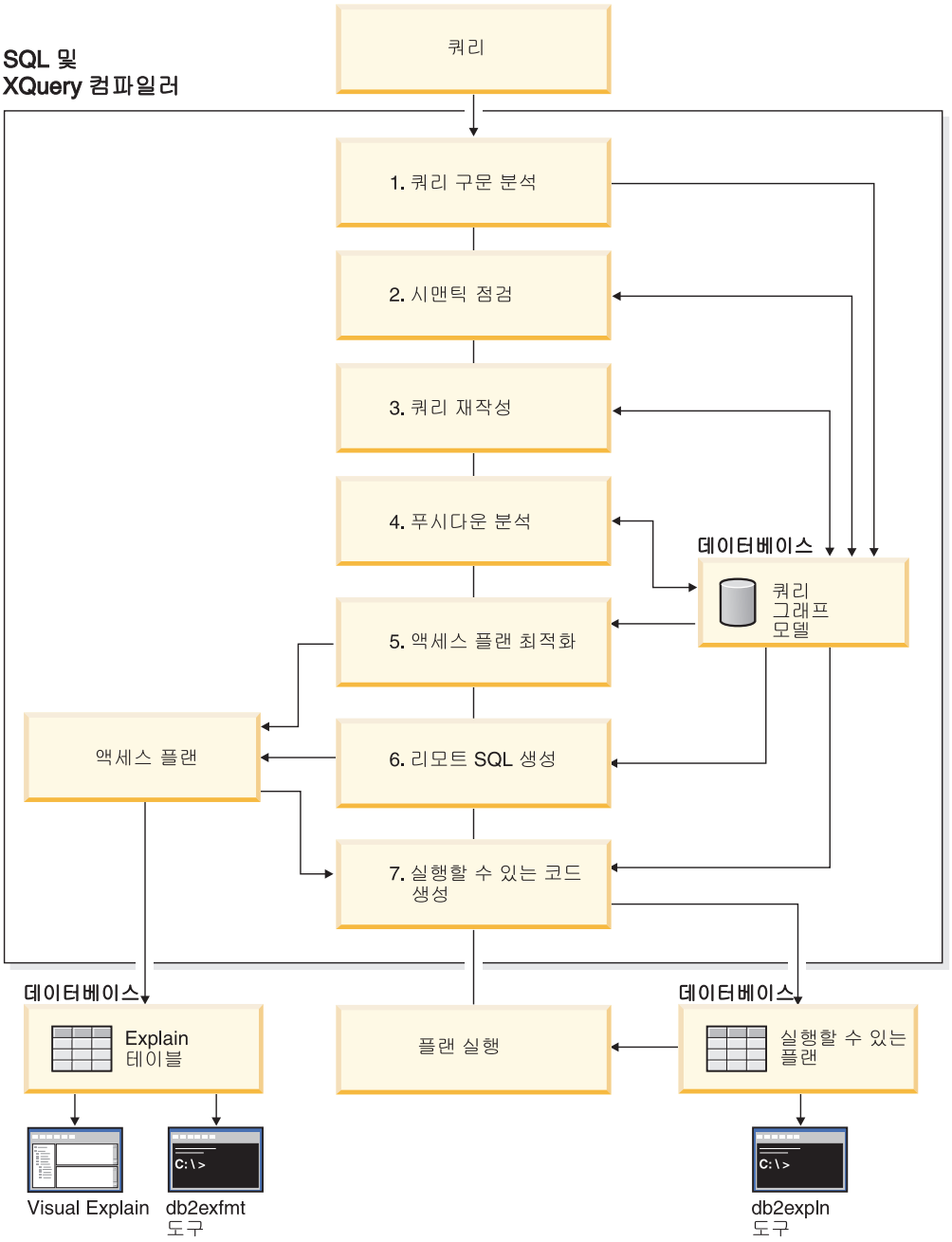


그림 24. SQL 및 XQuery 컴파일러가 수행하는 단계

1. 쿼리 구문 분석

   SQL 및 XQuery 컴파일러는 쿼리를 분석하여 구문의 유효성을 확인합니다. 구문 오류가 발견되면 쿼리 컴파일러는 처리를 중지하고 쿼리를 제출한 응용프로그램으로 해당 오류를 리턴합니다. 구문 분석이 완료되면 쿼리의 내부 표현이 작성되고 쿼리 그래프 모델에 저장됩니다.

2. 시맨틱 점검

   컴파일러는 명령문 부분들의 불일치가 없는지 확인합니다. 예를 들어, 컴파일러는 YEAR 스칼라 함수에 지정한 컬럼이 날짜 및 시간 데이터 유형을 사용하여 정의되었는지 검증합니다.

   또한 컴파일러는 참조 제한조건, 테이블 점검 제한조건, 트리거 및 뷰의 효과를 포함하여 쿼리 그래프 모델에 작동 시맨틱을 추가합니다. 쿼리 그래프 모델에는 쿼리 블록, 서브쿼리, 상관, 파생된 테이블, 표현식, 데이터 유형, 데이터 유형 변환, 코드 페이지 변환 및 분산 키와 같이 쿼리에 대한 모든 시맨틱이 들어 있습니다.

3. 쿼리 재작성

   컴파일러는 쿼리 그래프 모델에 저장된 전역 시맨틱을 사용하여 쿼리를 보다 쉽게 최적화할 수 있는 형식으로 변환합니다. 그런 다음 쿼리 그래프 모델에 결과를 저장합니다.

   예를 들어, 컴파일러는 술어가 적용된 레벨을 변경하여 술어를 이동할 수 있으므로 잠재적으로 쿼리 성능이 향상됩니다. 이러한 연산 이동 유형을 *일반 술어 푸시다운*이라고 합니다. 파티션된 데이터베이스 환경에서 다음 쿼리 연산은 계산 집중적입니다.

   - 집계
   - 행 재분산
   - 서브쿼리 외부에 있는 테이블의 컬럼에 대한 참조가 있는 서브쿼리인 관련된 서브쿼리

   파티션된 데이터베이스 환경에 있는 일부 쿼리의 경우 쿼리 재작성의 일부분으로 관련 해제가 발생할 수 있습니다.

4. 푸시다운 분석(페더레이티드 데이터베이스의 경우만)

   이 단계의 주요 태스크는 데이터 소스에서 연산을 리모트로 평가하거나 *푸시다운*할 수 있는지 여부를 옵티마이저에 권장하는 것입니다. 이러한 푸시다운 활동 유형은 데이터 소스 쿼리에만 해당하며 일반 술어 푸시다운 연산의 확장을 표시합니다.

5. 액세스 플랜 최적화

   쿼리 그래프 모델을 입력으로 사용하여 컴파일러의 옵티마이저 부분은 쿼리를 충족시키는 여러 대체 실행 플랜을 생성합니다. 이러한 각 플랜의 실행 비용을 추정하기 위해 옵티마이저는 테이블, 인덱스, 컬럼 및 함수의 통계를 사용합니다. 그런 다음 추정된 실행 비용이 가장 작은 플랜을 선택합니다. 옵티마이저는 쿼리 그래프 모델을 사용하여 쿼리 시맨틱을 분석하고 인덱스, 기본 테이블, 파생된 테이블, 서브쿼리, 상관 및 재귀와 같은 다양한 인수에 대한 정보를 얻습니다.

또한 옵티마이저는 이러한 연산의 평가를 DMS(Data Management Services) 구성 요소로 푸시하여 성능을 향상시킬 수 있는 다른 푸시다운 연산 유형인 *집계 및 정렬*을 고려할 수 있습니다.

옵티마이저는 페이지 *크기* 선택사항을 판별할 때 여러 가지 *크기*의 버퍼 풀이 있는지 여부도 고려합니다. 데이터베이스의 파티션 여부나 대칭형 멀티프로세서(SMP) 환경의 쿼리 내 병렬 처리가 옵션인지 여부를 고려합니다. 옵티마이저는 이 정보를 사용하여 쿼리에 가장 적합한 액세스 플랜을 선택할 수 있습니다.

이 단계의 출력으로 액세스 플랜이 작성되며 이 액세스 플랜의 세부사항은 Explain 테이블에 캡처되어 있습니다. 액세스 플랜을 생성하는 데 사용되는 정보는 Explain 스냅샷에서 캡처할 수 있습니다.

6. 리모트 SQL 생성(페더레이티드 데이터베이스의 경우만)

   옵티마이저가 선택하는 최종 플랜은 리모트 데이터 소스에서 작동되는 단계 세트로 구성되어 있습니다. 리모트 SQL 생성 단계에서는 해당 데이터 소스에서 SQL 통용어에 따라 각 데이터 소스에서 수행하는 조작의 효율적인 SQL문을 작성합니다.

7. 실행할 수 있는 코드 생성

   최종 단계에서 컴파일러는 액세스 플랜 및 쿼리 그래프 모델을 사용하여 쿼리에 대해 실행할 수 있는 액세스 플랜 또는 섹션을 작성합니다. 이 코드 생성 단계는 쿼리 그래프 모델의 정보를 사용하여 한 번만 계산해야 하는 표현식의 반복 실행을 방지합니다. 이러한 최적화 유형은 코드 페이지를 변환하는 경우와 호스트 변수를 사용하는 경우에 가능합니다.

   호스트 변수, 특수 레지스터 또는 매개변수 표시문자가 있는 정적이나 동적 SQL 또는 XQuery문의 쿼리 최적화 또는 재최적화를 사용하려면 REOPT 바인드 옵션을 사용하여 패키지를 바인드하십시오. 이러한 패키지에 속하고 호스트 변수, 특수 레지스터 또는 매개변수 표시문자가 포함된 명령문의 액세스 경로는 컴파일러가 선택한 디폴트 추정 값이 아닌 이러한 변수의 값을 사용하여 최적화됩니다. 이러한 최적화는 값이 사용 가능한 쿼리 실행 시간에 발생합니다.

   정적 SQL 및 XQuery문의 액세스 플랜에 대한 정보는 시스템 카탈로그 테이블에 저장됩니다. 패키지가 실행되면 데이터베이스 관리 프로그램은 시스템 카탈로그에 저장된 정보를 사용하여 데이터에 액세스하는 방법을 판별하고 쿼리의 결과를 제공합니다. 이 정보는 db2expln 도구에서 사용합니다.

주: 자주 변경되는 테이블에 대해 적절한 간격으로 RUNSTATS 명령을 실행하십시오. 가장 효율적인 액세스 플랜을 작성하려면 옵티마이저는 테이블 및 데이터에 대한 최신 통계 정보가 필요합니다. 응용프로그램을 리바인드하여 갱신된 통계를 활용하십시오. RUNSTATS가 실행되지 않았거나 옵티마이저가 이 명령이 비어 있거나 거의 비어 있

는 테이블에 대해 실행되었다고 가정하면 디폴트값을 사용하거나 디스크에 테이블을 저장하는 데 사용되는 파일 페이지 수에 따라 특정 통계를 파생시키려고 시도할 수 있습니다. 『자동 통계 콜렉션』도 참조하십시오.

쿼리 재작성 방법 및 예

쿼리 재작성 단계 중 쿼리 컴파일러는 SQL 및 XQuery문을 쉽게 최적화할 수 있는 형식으로 변환합니다. 이 경우 가능한 액세스 플랜을 향상시킬 수 있습니다. 쿼리 재작성은 특히 서브쿼리나 조인이 많은 쿼리와 같이 매우 복잡한 쿼리의 경우에 중요합니다. 쿼리 생성 프로그램 도구는 이와 같은 유형의 매우 복잡한 쿼리를 주로 작성합니다.

SQL 또는 XQuery문에 적용되는 쿼리 재작성 규칙 수에 영향을 주려면 최적화 클래스를 변경하십시오. 쿼리 재작성 프로세서의 일부 결과를 보려면 Explain 기능 또는 Visual Explain을 사용하십시오.

쿼리는 다음 중 한 방법으로 재작성할 수 있습니다.

- 연산 병합

  특히 SELECT 연산과 같은 연산 수가 가장 적은 쿼리를 구성하려면 SQL 및 XQuery 컴파일러는 쿼리를 재작성하여 쿼리 연산을 병합합니다. 다음 예는 병합할 수 있는 일부 연산을 나타냅니다.

  – 예 - 뷰 병합

    뷰를 사용하는 SELECT문은 테이블의 조인 순서를 제한할 수 있으며 테이블의 중복 조인을 도입할 수도 있습니다. 쿼리 재작성 중 뷰를 병합하는 경우 이러한 제한사항을 높아질 수 있습니다.

  – 예 - 변환을 조인할 서브쿼리

    SELECT문에 서브쿼리가 포함된 경우 테이블 처리 순서 선택이 제한될 수 있습니다.

  – 예 - 중복 조인 제거

    쿼리 재작성 중 중복 조인을 제거하면 SELECT문이 간단해집니다.

  – 예 - 공유 집계

    쿼리에서 다른 함수를 사용하는 경우 재작성하면 수행해야 하는 계산 수가 줄어듭니다.

- 연산 이동

  최소한의 연산 및 술어로 쿼리를 구성하기 위해 컴파일러는 쿼리를 재작성하여 쿼리 연산을 이동합니다. 다음 예는 이동할 수 있는 일부 연산을 나타냅니다.

  – 예 - DISTINCT 제거

쿼리 재작성 중 옵티마이저는 DISTINCT 연산을 수행하는 지점을 이동하여 이 연산의 비용을 줄일 수 있습니다. DISTINCT 연산을 완전히 제거할 수 있는 경우도 있습니다.

– 예 - 일반 술어 푸시다운

쿼리 재작성 중 옵티마이저는 술어를 적용하는 순서를 변경할 수 있으므로 가능하면 신속하게 여러 개의 선택적 술어가 쿼리에 적용됩니다.

– 예 - 관련 해제

파티션된 데이터베이스 환경에서 데이터베이스 파티션들 간 결과 세트 이동은 비용이 많이 듭니다. 다른 데이터베이스 파티션으로 브로드캐스트해야 하는 내용의 크기를 줄이거나 브로드캐스트 수를 줄이거나 두 가지 사항을 모두 줄이는 것이 쿼리 재작성 프로세스의 목표입니다.

• 술어 변환

SQL 및 XQuery 컴파일러는 쿼리를 재작성하여 기존 술어를 최적의 형식으로 변환합니다. 다음 예는 변환할 수 있는 일부 술어를 나타냅니다.

– 예 - 내재된 술어 추가

쿼리 재작성 중 쿼리에 술어를 추가하여 옵티마이저가 쿼리에 대한 최상의 액세스 플랜을 선택할 때 추가 테이블 조인을 고려할 수 있습니다.

– 예 - OR에서 IN으로 변환

쿼리 재작성 중 효율적인 액세스 플랜을 위해 OR 술어를 IN 술어로 변환할 수 있습니다. SQL 및 XQuery 컴파일러도 IN 술어를 OR 술어로 변환할 수 있습니다(이 변환이 효율적인 액세스 플랜을 작성하는 경우).

컴파일러 재작성 예: 뷰 병합:

뷰를 사용하는 SELECT문은 테이블의 조인 순서를 제한할 수 있으며 테이블의 중복 조인을 도입할 수도 있습니다. 쿼리 재작성 중 뷰를 병합하는 경우 이러한 제한사항을 높아질 수 있습니다.

EMPLOYEE 테이블에 기초하는 다음 두 개의 뷰에 액세스할 수 있다고 가정하십시오. 한 뷰는 높은 수준의 교육을 받은 직원을 나타내고 다른 뷰는 연봉이 $35,000을 초과하는 직원을 나타냅니다.

```
create view emp_education (empno, firstnme, lastname, edlevel) as
  select empno, firstnme, lastname, edlevel
    from employee
    where edlevel > 17

create view emp_salaries (empno, firstname, lastname, salary) as
  select empno, firstnme, lastname, salary
    from employee
    where salary > 35000
```

다음의 사용자 작성 쿼리는 높은 수준의 교육을 받고 연봉이 $35,000을 초과하는 직원을 나열합니다.

```
select e1.empno, e1.firstnme, e1.lastname, e1.edlevel, e2.salary
  from emp_education e1, emp_salaries  e2
  where e1.empno = e2.empno
```

쿼리 재작성 중 이 두 개의 뷰를 병합하여 다음의 쿼리를 작성할 수 있습니다.

```
select e1.empno, e1.firstnme, e1.lastname, e1.edlevel, e2.salary
  from employee e1, employee e2
  where
    e1.empno = e2.empno and
    e1.edlevel > 17 and
    e2.salary  > 35000
```

두 뷰의 SELECT문을 사용자 작성 SELECT문과 병합하여 옵티마이저는 액세스 플랜 선택 시 추가 선택사항을 고려할 수 있습니다. 또한 병합된 두 개의 뷰가 동일한 기본 테이블을 사용하는 경우 추가 재작성이 수행될 수도 있습니다.

예 - 변환을 조인할 서브쿼리

SQL 및 XQuery 컴파일러는 다음과 같이 서브쿼리를 포함하는 쿼리를 사용합니다.

```
select empno, firstnme, lastname, phoneno
  from employee
  where workdept in
    (select deptno
      from department
      where deptname = 'OPERATIONS')
```

또한 다음과 같은 형식의 조인 쿼리로 변환합니다.

```
select distinct empno, firstnme, lastname, phoneno
  from employee emp, department dept
  where
    emp.workdept = dept.deptno and
    dept.deptname = 'OPERATIONS'
```

조인은 일반적으로 서브쿼리보다 실행하는 데 훨씬 효율적입니다.

예 - 중복 조인 제거

쿼리에 불필요한 조인이 포함된 경우가 있습니다.

```
select e1.empno, e1.firstnme, e1.lastname, e1.edlevel, e2.salary
  from employee e1,
       employee e2
 where e1.empno = e2.empno
   and e1.edlevel > 17
   and e2.salary  > 35000
```

SQL 및 XQuery 컴파일러는 조인을 제거하고 쿼리를 다음과 같이 간소화할 수 있습니다.

```
select empno, firstnme, lastname, edlevel, salary
  from employee
  where
    edlevel > 17 and
    salary  > 35000
```

다음 예에서는 부서 번호에 대한 EMPLOYEE 테이블과 DEPARTMENT 테이블에 참조 제한조건이 존재한다고 가정합니다. 먼저 뷰를 작성합니다.

```
create view peplview as
  select firstnme, lastname, salary, deptno, deptname, mgrno
    from employee e department d
    where e.workdept = d.deptno
```

그런 다음 아래와 같은 쿼리를 작성합니다.

```
select lastname, salary
  from peplview
```

이 경우 다음과 같이 변환됩니다.

```
select lastname, salary
  from employee
  where workdept not null
```

이 상황에서 쿼리를 재작성할 수 있음을 아는 경우에도 기본 테이블에 액세스할 수 없으므로 쿼리를 재작성할 수 없습니다. 뷰(위에 표시됨)에만 액세스할 수 있습니다. 따라서 이러한 유형의 최적화는 데이터베이스 관리 프로그램이 수행해야 합니다.

다음과 같은 경우 참조 무결성 조인의 중복이 발생할 수 있습니다.

- 조인을 사용하여 뷰를 정의한 경우
- 쿼리가 자동으로 생성된 경우

예 - 공유 집계

쿼리에서 여러 함수를 사용하면 시간이 오래 걸리는 여러 계산이 생성될 수 있습니다. 필요한 계산 수를 줄이면 플랜이 향상됩니다. 컴파일러는 다음과 같이 여러 함수를 사용하는 쿼리를 사용합니다.

```
select sum(salary+bonus+comm) as osum,
    avg(salary+bonus+comm) as oavg,
    count(*) as ocount
  from employee
```

그리고 다음과 같이 변환합니다.

```
select osum, osum/ocount ocount
  from (
    select sum(salary+bonus+comm) as osum,
        count(*) as ocount
      from employee
  ) as shared_agg
```

이 재작성으로 인해 필요한 합계 및 계수의 수가 절반으로 줄어듭니다.

컴파일러 재작성 예: DISTINCT 제거:

쿼리 재작성 중 옵티마이저는 DISTINCT 연산을 수행하는 지점을 이동하여 이 연산의 비용을 줄일 수 있습니다. DISTINCT 연산을 완전히 제거할 수 있는 경우도 있습니다.

예를 들어, EMPLOYEE 테이블의 EMPNO 컬럼이 1차 키로 정의된 경우 다음의 쿼리는

```
select distinct empno, firstnme, lastname
  from employee
```

DISTINCT절을 제거하여 재작성할 수 있습니다.

```
select empno, firstnme, lastname
  from employee
```

이 예에서 1차 키를 선택하므로 컴파일러는 리턴된 각 행이 고유한 것으로 판단합니다. 이 경우 DISTINCT 키워드는 중복됩니다. 쿼리가 재작성되지 않은 경우 컬럼 값이 고유하도록 옵티마이저는 필요한 처리(예: 정렬)를 수행하여 플랜을 빌드해야 합니다.

예 - 일반 술어 푸시다운

술어가 일반적으로 적용되는 레벨을 변경하면 성능이 향상될 수 있습니다. 예를 들어, 다음 뷰는 D11 부서의 모든 직원 목록을 제공합니다.

```
create view d11_employee
  (empno, firstnme, lastname, phoneno, salary, bonus, comm) as
    select empno, firstnme, lastname, phoneno, salary, bonus, comm
      from employee
      where workdept = 'D11'
```

이 뷰에 대한 다음의 쿼리는 효율성이 크지 않습니다.

```
select firstnme, phoneno
  from d11_employee
  where lastname = 'BROWN'
```

쿼리 재작성 중 컴파일러는 `lastname = 'BROWN'` 술어를 D11_EMPLOYEE 뷰로 푸시다운합니다. 이 경우 술어를 즉시 적용할 수 있으며 잠재적인 효율성이 높습니다. 이 예에서 실행할 수 있는 실제 쿼리는 다음과 같습니다.

```
select firstnme, phoneno
  from employee
  where
    lastname = 'BROWN' and
    workdept = 'D11'
```

술어 푸시다운은 뷰로만 제한되지 않습니다. 술어를 푸시다운할 수 있는 기타 상황으로는 UNION, GROUP BY 및 파생된 테이블(중첩된 테이블 표현식 또는 공통 테이블 표현식)이 있습니다.

예 - 관련 해제

파티션된 데이터베이스 환경에서 컴파일러는 프로그래밍 프로젝트에 대해 작업 중이며 급여를 적게 받는 모든 직원을 찾는 다음의 쿼리를 재작성할 수 있습니다.

```
select p.projno, e.empno, e.lastname, e.firstname,
    e.salary+e.bonus+e.comm as compensation
  from employee e, project p
  where
    p.empno = e.empno and
    p.projname like '%PROGRAMMING%' and
    e.salary+e.bonus+e.comm <
      (select avg(e1.salary+e1.bonus+e1.comm)
        from employee e1, project p1
        where
          p1.projname like '%PROGRAMMING%' and
          p1.projno = a.projno and
          e1.empno = p1.empno)
```

이 쿼리는 관련되어 있으며 PROJECT 및 EMPLOYEE가 모두 PROJNO에서 파티션되지 않을 수 있으므로 각 데이터베이스 파티션으로 각 프로젝트가 브로드캐스트될 수 있습니다. 또한 서브쿼리를 여러 번 평가해야 합니다.

컴파일러는 다음과 같이 쿼리를 재작성할 수 있습니다.

- 프로그래밍 프로젝트에 대해 작업 중인 직원의 구별 목록을 판별하고 DIST_PROJS로 이름을 지정하십시오. 각 프로젝트마다 집계를 한 번만 수행하도록 구별된 목록이어야 합니다.

  ```
  with dist_projs(projno, empno) as
    (select distinct projno, empno
      from project p1
      where p1.projname like '%PROGRAMMING%')
  ```

- DIST_PROJS를 EMPLOYEE 테이블과 조인하여 프로젝트 당 평균 수당 AVG_PER_PROJ를 확인하십시오.

  ```
  avg_per_proj(projno, avg_comp) as
    (select p2.projno, avg(e1.salary+e1.bonus+e1.comm)
      from employee e1, dist_projs p2
      where e1.empno = p2.empno
      group by p2.projno)
  ```

- 재작성된 쿼리는 다음과 같습니다.

  ```
  select p.projno, e.empno, e.lastname, e.firstname,
      e.salary+e.bonus+e.comm as compensation
    from project p, employee e, avg_per_prog a
    where
      p.empno = e.empno and
  ```

```
        p.projname like '%PROGRAMMING%' and
        p.projno = a.projno and
        e.salary+e.bonus+e.comm < a.avg_comp
```

이 쿼리는 프로젝트 당 avg_comp(avg_per_proj)를 계산합니다. 결과는 EMPLOYEE 테이블이 포함된 모든 데이터베이스 파티션으로 브로드캐스트할 수 있습니다.

컴파일러 재작성 예: 조합 SQL/XQuery문에 대한 술어 푸시다운:

관계형 SQL 쿼리의 최적화에 대한 한 가지 기본 기술은 포함하는 쿼리 블록의 WHERE 절에 있는 술어를 포함된 더 낮은 쿼리 블록(예: 뷰)으로 이동함으로써 조기 데이터 필터링 및 잠재적으로 더 나은 인덱스 사용을 가능하게 하는 것입니다.

조기 필터링은 잠재적으로 데이터베이스 파티션 간에 적재되어야 하는 데이터의 양을 줄여주므로 이것은 파티션된 데이터베이스 환경에서 훨씬 더 중요합니다.

유사한 기술을 사용하여 XQuery 내부의 XPath 필터 또는 술어를 이동할 수 있습니다. 기본 전략은 항상 필터링 표현식을 데이터 소스에 가능한 한 가깝게 이동하는 것입니다. 이 최적화 기술을 SQL에서는 *술어 푸시다운*이라고 하고 XQuery에서는 *추출 푸시다운*(필터 및 XPath 추출에 대해)이라고 합니다.

SQL과 XQuery에서 사용하는 데이터 모델이 다르기 때문에 두 언어 간 경계에서 술어, 필터 또는 추출을 이동해야 합니다. SQL 술어를 의미구조적으로 동등한 필터로 전환하고 XPath 추출로 푸시다운할 때 데이터 맵핑 및 캐스팅 규칙을 고려해야 합니다. 다음 예에서는 XQuery 쿼리 블록으로의 관계 술어 푸시다운을 설명합니다.

고객 정보를 포함하는 다음 두 XML 문서를 고려해 보십시오.

```
Document 1

<customer>
  <name>John</name>
  <lastname>Doe</lastname>
  <date_of_birth>
   1976-10-10
  </date_of_birth>
  <address>
    <zip>95141.0</zip>
  </address>
<volume>80000.0</volume>
</customer>
<customer>
  <name>Jane</name>
  <lastname>Doe</lastname>
  <date_of_birth>
    1975-01-01
  </date_of_birth>
  <address>
    <zip>95141.4</zip>
  </address>
<volume>50000.00</volume>
</customer>
```

```
Document 2

<customer>
  <name>Michael</name>
  <lastname>Miller </lastname>
  <date_of_birth>
  1975-01-01
  </date_of_birth>
  <address>
    <zip>95142.0</zip>
  </address>
<volume>100000.00</volume>
</customer>
<customer>
  <name>Michaela</name>
  <lastname>Miller</lastname>
  <date_of_birth>
   1980-12-23
  </date_of_birth>
  <address>
    <zip>95140.5</zip>
  </address>
<volume>100000</volume>
</customer>
```

예 - 정수 술어 푸시

다음 쿼리를 고려해보십시오.

```
select temp.name, temp.zip
  from xmltable('db2-fn:xmlcolumn("T.XMLDOC")'
    columns name varchar(20) path 'customer/name',
    zip integer path 'customer/zip'
    ) as temp
  where zip = 95141
```

T.XMLDOC에서 가능한 인덱스를 사용하고 원하지 않는 개인을 조기에 필터링하기 위해 `zip = 95141` 술어가 다음 동등한 XPATH 필터링 표현식으로 내부적으로 변환됩니다.

```
T.XMLCOL/customer/zip[. >= 95141.0 and . < 95142.0]
```

XML 조각에 대한 스키마 정보가 컴파일러에서 사용되지 않기 때문에 ZIP에 정수만 포함되었다고 가정할 수 없습니다. 이 특정 XPath 추출에서 해당 이중 XML 인덱스 및 분수 부분이 있는 다른 숫자 값이 있을 수 있습니다. XML2SQL 캐스트는 값을 INTEGER로 캐스트하기 전에 분수 부분을 절단하여 이 변환을 처리합니다. 이 동작은 푸시다운 프로시저에 반영되어야 하고, 의미구조적으로 올바르게 유지되도록 술어가 변경되어야 합니다.

예 - VARCHAR(*n*) 술어 푸시

다음 쿼리를 고려해보십시오.

```
select temp.name, temp.lastname
  from xmltable('db2-fn:xmlcolumn("T.XMLDOC")'
    columns name varchar(20) path 'customer/name',
    lastname varchar(20) path 'customer/lastname'
    ) as temp
  where lastname = 'Miller'
```

T.XMLDOC에서 가능한 인덱스를 사용하고 원하지 않는 개인을 조기에 필터링하기 위해 `lastname = 'Miller'` 술어가 동등한 XPATH 필터링 표현식으로 내부적으로 변환됩니다.

```
T.XMLCOL/customer/lastname[. > rtrim("Miller") and
< blank_padd("Miller", max(20,length("Miller"))]
```

SQL에서 뒤 공백은 XPath 또는 XQuery에서와 다르게 처리됩니다. 원래 SQL 술어는 이 중 하나(Michale)에 뒤 공백이 있다 하더라도 성이 『Miller』인 두 고객을 구별하지 못합니다. 따라서, 두 고객이 모두 리턴됩니다(변경되지 않은 술어가 푸시다운된 경우에는 그렇지 않음).

해결 방법은 술어를 범위 필터로 변환하는 것입니다.

- RTRIM() 기능을 사용하여 비교 값에서 모든 뒤 공백을 절단함으로써 첫 번째 경계가 작성됩니다.

- 두 번째 경계는 뒤 공백을 포함하는 모든 가능한『Miller』문자열보다 *크거나 같아야* 합니다. 더 긴 경우, 원래 문자열이 최대 컬럼 길이 또는 비교 문자열의 길이까지 공백으로 채워집니다.

예 - DECIMAL(*x*,*y*) 술어 푸시

다음 쿼리를 고려해보십시오.

```
select temp.name, temp.volume
  from xmltable('db2-fn:xmlcolumn("T.XMLDOC")'
    columns name varchar(20) path 'customer/name',
    volume decimal(10,2) path 'customer/volume'
    ) as temp
  where volume = 100000.00
```

T.XMLDOC에서 가능한 이중 인덱스를 사용하고 원하지 않는 개인을 조기에 필터링하기 위해 `volume = 100000.00` 술어가 다음 동등한 XPATH 필터링 표현식으로 내부적으로 변환됩니다.

```
T.XMLCOL/customer/volume[.=100000.00]
```

캐스팅 제한조건으로 인해 XML 값이 대상 SQL 유형과 동일한 분수 부분 길이 및 정밀도를 가져야 하므로 술어를 범위 필터로 변환할 필요가 없습니다. 이 제한조건을 위반할 경우 오류가 리턴됩니다. DOUBLE 값이 DECIMAL(*x*,*y*)로 캐스트될 때 정밀도가 감소하지 않습니다. 비교 값을 반올림하거나 절단할 필요가 없습니다.

컴파일러 재작성 예: 내재된 술어:

쿼리 재작성 중 쿼리에 술어를 추가하여 옵티마이저가 쿼리에 대한 최상의 액세스 플랜을 선택할 때 추가 테이블 조인을 고려할 수 있습니다.

다음 쿼리는 부서가 E01 부서로 보고되는 관리자와 이러한 관리자가 담당하는 프로젝트 목록을 리턴합니다.

```
select dept.deptname dept.mgrno, emp.lastname, proj.projname
  from department dept, employee emp, project proj
  where
    dept.admrdept = 'E01' and
    dept.mgrno = emp.empno and
    emp.empno = proj.respemp
```

*이행 종결을 위한 술어*라고 하는 다음의 내재된 술어로 이 쿼리를 재작성할 수 있습니다.

```
dept.mgrno = proj.respemp
```

옵티마이저는 쿼리에 대한 최상의 액세스 플랜을 선택할 때 추가 조인을 고려할 수 있습니다.

쿼리 재작성 단계 중 등호 술어에 내재된 이행에 따라 추가 로컬 술어가 파생됩니다. 예를 들어, 다음의 쿼리는 부서 번호가 E00보다 큰 부서의 이름과 이러한 부서에서 근무하는 직원을 리턴합니다.

```
select empno, lastname, firstname, deptno, deptname
  from employee emp, department dept
  where
    emp.workdept = dept.deptno and
    dept.deptno > 'E00'
```

이 쿼리는 다음의 내재된 술어를 사용하여 재작성할 수 있습니다.

```
emp.workdept > 'E00'
```

이 재작성으로 인해 조인해야 하는 행 수가 줄어듭니다.

예 - OR에서 IN으로 변환

다음 예와 같이 OR절이 동일한 컬럼에서 둘 이상의 간단한 등호 술어를 연결한다고 가정하십시오.

```
select *
  from employee
  where
    deptno = 'D11' or
    deptno = 'D21' or
    deptno = 'E21'
```

DEPTNO 컬럼에 인덱스가 없는 경우 OR 대신 IN 술어를 사용하면 쿼리가 효율적으로 처리됩니다.

```
select *
  from employee
  where deptno in ('D11', 'D21', 'E21')
```

어떤 경우에는 데이터베이스 관리 프로그램이 IN 술어를 OR절 세트로 변환하여 Index ORing이 수행될 수 있습니다.

술어 유형 및 액세스 플랜

술어는 비교 연산을 표현하거나 내재하는 검색 조건의 요소입니다. 보통 쿼리의 WHERE 절에 표시되는 술어는 쿼리에서 리턴하는 결과 세트의 범위를 줄이는 데 사용됩니다.

술어를 평가 프로세스에서 사용하는 방식과 시점에 따라 네 가지 범주로 분류할 수 있습니다. 이러한 범주는 가장 좋은 성능부터 가장 좋지 못한 성능 순서로 나열됩니다.

1. 범위 구분 술어
2. 인덱스 SARGable 술어
3. 데이터 SARGable 술어
4. 레지듀얼(Residual) 술어

SARGable 용어는 검색 인수(*s*earch *arg*ument)로 사용할 수 있는 용어입니다.

표 51에서는 이러한 술어 범주의 특성을 요약합니다.

표 51. 술어 유형 특성 요약

특성	술어 유형			
	범위 구분	인덱스 SARGable	데이터 SARGable	레지듀얼(Residual)
인덱스 입출력 축소	예	아니오	아니오	아니오
데이터 페이지 입출력 축소	예	예	아니오	아니오
내부적으로 전달되는 행 수 축소	예	예	예	아니오
규정 행 수 축소	예	예	예	예

범위 구분 및 인덱스 SARGable 술어

범위 구분 술어는 인덱스 스캔의 범위를 제한합니다. 인덱스 검색의 시작 및 중지 키 값을 제공합니다. 인덱스 SARGable 술어는 검색의 범위를 제한할 수 없지만 술어에 참조된 컬럼이 인덱스 키의 일부분이므로 인덱스에서 평가할 수 있습니다. 예를 들어, 다음 인덱스를 참조하십시오.

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

다음의 WHERE절이 있는 쿼리도 참조하십시오.

```
where
  name = :hv1 and
  dept = :hv2 and
  years > :hv5
```

처음 두 개의 술어(`name = :hv1 and dept = :hv2`)는 범위 구분 술어이고 `years > :hv5`는 인덱스 SARGable 술어입니다.

옵티마이저는 이러한 술어를 평가할 때 기본 테이블을 읽는 대신 인덱스 데이터를 사용합니다. 인덱스 SARGable 술어는 테이블에서 읽어야 하는 행 수를 줄이지만 액세스하는 인덱스 페이지 수에는 영향을 주지 않습니다.

데이터 SARGable 술어

인덱스 관리 프로그램으로 평가할 수 없지만 DMS(Data Management Services)로 평가할 수 있는 술어를 데이터 SARGable 술어라고 합니다. 이러한 술어는 보통 테이블의 개별 행에 액세스해야 합니다. 필요하면 DMS는 술어를 평가하는 데 필요한 컬럼과 SELECT 목록에 필요하지만 인덱스에서 확보할 수 없는 다른 컬럼을 검색합니다.

예를 들어, PROJECT 테이블에 정의된 다음의 인덱스를 참조하십시오.

```
INDEX IX0:  PROJNO ASC
```

다음 쿼리에서 deptno = 'D11'은 데이터 SARGable 술어로 간주됩니다.

```
select projno, projname, respemp
  from project
  where deptno = 'D11'
  order by projno
```

레지듀얼(Residual) 술어

레지듀얼(Residual) 술어는 입출력 비용 면에서 테이블에 액세스하는 경우보다 비용이 많이 듭니다. 이러한 술어는 다음과 같은 특성이 있습니다.

- 상관 서브쿼리를 사용할 수 있습니다.
- ANY, ALL, SOME 또는 IN절을 포함한 정량화 서브쿼리를 사용할 수 있습니다.
- 테이블과 구별된 파일에 저장되어 있는 LONG VARCHAR 또는 LOB 데이터를 읽을 수 있습니다.

이러한 술어는 RDS(Relational Data Services)로 평가합니다.

인덱스에만 적용되는 일부 술어는 데이터 페이지에 액세스할 때 재적용해야 합니다. 예를 들어, Index ORing 또는 Index ANDing을 사용하는 액세스 플랜은 항상 데이터 페이지에 액세스할 때 술어를 레지듀얼(Residual) 술어로 재적용합니다.

페더레이티드 데이터베이스 쿼리-컴파일러 단계

페더레이티드 데이터베이스 푸시다운 분석:

페더레이티드 데이터베이스에 대해 실행할 쿼리의 경우 옵티마이저는 푸시다운 분석을 수행하여 리모트 데이터 소스에서 특정 연산을 수행할 수 있는지 여부를 판별합니다.

연산은 관계 연산자와 같은 함수이거나 시스템 또는 사용자 함수 또는 ORDER BY나 GROUP BY와 같은 SQL 연산자일 수도 있습니다.

DB2 쿼리 컴파일러가 리모트 데이터 소스에서 SQL 지원에 대한 정확한 정보에 액세스할 수 잇도록 로컬 카탈로그 정보를 주기적으로 갱신해야 합니다. 예를 들어, CREATE FUNCTION MAPPING 또는 ALTER SERVER와 같은 DB2 DDL(Data Definition Language)문을 사용하여 카탈로그를 갱신하십시오.

리모트 데이터 소스로 함수를 푸시다운할 수 없는 경우 쿼리 성능에 상당한 영향을 줄 수 있습니다. 선택적 술어를 데이터 소스 대신 로컬로 강제 평가하는 경우 미치는 영향을 고려하십시오. 이와 같이 평가할 경우 DB2 서버는 리모트 데이터 소스에서 전체 테이블을 검색한 후 술어와 비교하여 로컬로 필터해야 합니다. 네트워크 제한조건 및 대형 테이블은 성능 저하를 발생시킬 수 있습니다.

푸시다운되지 않은 연산자도 쿼리 성능에 상당한 영향을 줄 수 있습니다. 예를 들어, GROUP BY 연산자가 리모트 데이터를 로컬로 집계하도록 설정할 경우에도 DB2 서버가 리모트 데이터 소스에서 전체 테이블을 검색해야 합니다.

예를 들어, z/OS용 DB2 데이터 소스의 데이터 소스 테이블 EMPLOYEE를 참조하는 별칭 N1을 참조하십시오. 테이블에는 10,000개의 행이 있으며 컬럼 중 하나에 직원의 성이 있고 컬럼 중 하나에 급여가 있습니다. 옵티마이저는 로컬 및 리모트 조합 순서가 동일한지 여부에 따라 다음 명령문을 처리할 때 여러 가지 옵션을 사용할 수 있습니다.

```
select lastname, count(*)  from n1
  where
    lastname > 'B' and
    salary > 50000
  group by lastname
```

- 조합 순서가 동일한 경우 쿼리 술어를 z/OS용 DB2로 푸시다운할 수도 있습니다. 데이터 소스에서 결과를 필터링하고 그룹화하면 보통 전체 테이블을 복사하고 연산을 로컬로 수행하는 것보다 효율적입니다. 이 쿼리의 경우 술어 및 GROUP BY 연산이 데이터 소스에서 발생할 수 있습니다.
- 조합 순서가 동일하지 않은 경우 데이터 소스에서 두 술어를 모두 평가할 수 없습니다. 그러나 옵티마이저는 `salary > 50000` 술어를 푸시다운하도록 결정할 수 있습니다. 범위 비교는 계속 로컬로 수행합니다.
- 조합 순서가 동일하고 옵티마이저가 로컬 DB2 서버가 매우 빠른 것으로 인식한 경우 옵티마이저는 GROUP BY 연산을 로컬로 수행하는 것을 가장 저렴한 방법으로 판단할 수 있습니다. 술어는 데이터 소스에서 평가합니다. 전역 최적화와 결합된 푸시다운 분석의 한 예입니다.

일반적으로 옵티마이저가 리모트 데이터 소스에서 함수 및 연산자를 평가하는지 확인하기 위한 것입니다. 다음과 같은 여러 가지 요인이 리모트 데이터 소스에서 함수 또는 SQL 연산자를 평가할 수 있는지 여부에 영향을 줍니다.

- 서버 특성
- 별칭 특성
- 쿼리 특성

푸시다운 기회에 영향을 주는 서버 특성

특정 데이터 소스별 요인이 푸시다운 기회에 영향을 줄 수 있습니다. 일반적으로 이러한 요인은 DB2 제품에서 지원하는 다양한 SQL 통용어로 인해 존재합니다. DB2 Data Server는 다른 데이터 서버에서 사용 가능한 함수 부족을 보충할 수 있지만 이와 같이 보충할 경우 연산은 DB2 서버에서 발생해야 합니다.

- SQL 기능

  각 데이터 소스는 SQL 통용어의 변형과 여러 가지 레벨의 기능을 지원합니다. 예를 들어, 대부분의 데이터 소스는 GROUP BY 연산자를 지원하지만 일부는 GROUP BY 목록에서 항목 수를 제한하거나 GROUP BY 목록에서 표현식 허용 여부에 대한 제한사항을 설정합니다. 리모트 데이터 소스에 제한사항이 있는 경우 DB2 서버는 GROUP BY 연산을 로컬로 수행해야 합니다.

- SQL 제한사항

  각 데이터 소스마다 SQL 제한사항이 다를 수 있습니다. 예를 들어, 일부 데이터 소스에는 리모트 SQL문에 값을 바인드할 때 매개변수 표시문자가 필요합니다. 따라서 각 데이터 소스가 이러한 바인드 메커니즘을 지원할 수 있도록 매개변수 표시문자 제한사항을 점검해야 합니다. DB2 서버가 함수의 값을 바인드하는 좋은 방법을 판별할 수 없는 경우 이 함수를 로컬로 평가해야 합니다.

- SQL 한계

  DB2 서버가 리모트 데이터 소스에서 허용되는 것보다 큰 정수의 사용을 지원할 수 있지만 리모트 한계를 초과하는 값을 데이터 소스로 보내는 명령문에 임베드할 수 없으며 영향을 받은 함수나 연산자를 로컬로 평가해야 합니다.

- 서버 특정사항

  여러 요인이 이 범주에 속합니다. 예를 들어, 데이터 소스의 널(NULL) 값이 DB2 서버의 정렬 방식과 다르게 정렬된 경우 널(NULL) 입력 가능 표현식의 ORDER BY 연산을 리모트로 평가할 수 없습니다.

- 조합 시퀀스

  로컬 정렬 및 비교에 대한 데이터를 검색하면 보통 성능이 저하됩니다. 데이터 소스가 사용하는 동일한 조합 시퀀스를 사용하도록 페더레이티드 데이터베이스를 구성한 후 COLLATING_SEQUENCE 서버 옵션을 Y로 설정한 경우 옵티마이저는 여러 쿼리 연산의 푸시다운을 고려할 수 있습니다. 조합 시퀀스가 동일하면 다음의 연산이 푸시다운될 수 있습니다.

  – 문자 또는 숫자 데이터의 비교
  – 문자 범위 비교 술어
  – 정렬

  그러나 페더레이티드 데이터베이스와 데이터 소스의 널(NULL) 문자 가중치가 다른 경우 결과에 이상이 있을 수 있습니다. 대소문자를 구분하지 않는 데이터 소스에 명령문을 제출하는 경우 비교 시 예기치 않은 결과를 리턴할 수 있습니다. 대소문자를 구분하지 않는 데이터 소스의 문자 『I』 및 『i』에 지정된 가중치가 동일합니다. 디폴트로 DB2 서버는 대소문자를 구분하고 이러한 문자에 다른 가중치를 지정합니다.

  성능을 향상시키기 위해 페더레이티드 서버는 데이터 소스에서 정렬 및 비교를 수행할 수 있도록 합니다. 예를 들어, z/OS용 DB2에서 ORDER BY절에 정의된 정렬은 EBCDIC 코드 페이지에 기초한 조합 시퀀스로 구현됩니다. 페더레이티드 서버를 사용하여 ORDER BY절에 따라 정렬된 z/OS용 DB2 데이터를 검색하려면 EBCDIC 코드 페이지에 따라 사전 정의된 조합 시퀀스를 사용하도록 페더레이티드 데이터베이스를 구성하십시오.

페더레이티드 데이터베이스와 데이터 소스의 조합 시퀀스가 다른 경우 DB2 서버는 페더레이티드 데이터베이스로 데이터를 검색합니다. 사용자는 페더레이티드 서버에 대해 정의된 조합 시퀀스별로 정렬된 쿼리 결과를 예상하므로 데이터를 로컬로 정렬하면 예상대로 결과가 나타납니다. passthrough 모드에서 쿼리를 제출하거나 데이터 소스의 조합 시퀀스로 정렬된 데이터를 표시해야 하는 경우 데이터 소스 뷰에 쿼리를 정의하십시오.

- 서버 옵션

  여러 서버 옵션은
  COLLATING_SEQUENCE, VARCHAR_NO_TRAILING_BLANKS 및 PUSHDOWN과 같은 푸시다운 기회에 영향을 줄 수 있습니다.

- DB2 유형 맵핑 및 함수 맵핑 인수

  DB2 서버의 디폴트 로컬 데이터 유형 맵핑은 각 데이터 소스 데이터 유형에 충분한 버퍼 스페이스를 제공하여 데이터 손실을 방지하기 위한 것입니다. 특정 응용프로그램에 맞게 특정 데이터 소스의 유형 맵핑을 사용자 정의할 수 있습니다. 예를 들어, 디폴트로 DB2 TIMESTAMP 데이터 유형에 맵핑되는 DATE 데이터 유형을 사용하여 Oracle 데이터 소스 컬럼에 액세스하는 경우 로컬 데이터 유형을 DB2 DATE 데이터 유형으로 변경할 수 있습니다.

다음과 같은 세 가지 경우에 DB2 서버는 데이터 소스가 지원하지 않는 함수를 보충할 수 있습니다.

- 기능이 리모트 데이터 소스에 없는 경우.
- 기능이 있지만 피연산자의 특성이 함수 제한사항을 위반하는 경우. 이러한 경우에 해당하는 예로 IS NULL 관계 연산자가 있습니다. 대부분의 데이터 소스는 이를 지원하지만 컬럼 이름을 IS NULL 연산자의 왼쪽에만 표시하도록 허용할 때와 같이 일부는 제한사항을 설정할 수 있습니다.
- 함수가 리모트로 평가되는 경우 다른 결과를 리턴할 수 있습니다. 이러한 경우의 예로 초과('>') 연산자가 있습니다. 데이터 소스의 조합 시퀀스가 다르면 DB2 서버가 로컬로 평가하는 초과 연산자는 다른 결과를 리턴할 수 있습니다.

푸시다운 기회에 영향을 주는 별칭 특성

다음의 별칭 특정 요인은 푸시다운 기회에 영향을 줄 수 있습니다.

- 별칭 컬럼의 로컬 데이터 유형

  컬럼의 로컬 데이터 유형이 술어를 데이터 소스에서 평가하지 못하도록 예방하지 않는지 확인하십시오. 디폴트 데이터 유형 맵핑을 사용하여 오버플로우 가능성을 방지하십시오. 그러나 길이가 다른 두 컬럼의 조인 술어는 DB2가 긴 조인 컬럼을 바인드하는 방식에 따라 조인 컬럼이 짧은 데이터 소스에서 고려하지 않을 수 있습니다. 이러한 경우 DB2 옵티마이저가 조인 시퀀스로 평가할 수 있는 가능성의 수에 영향

을 줄 수 있습니다. 예를 들어, INTEGER 또는 INT 데이터 유형을 사용하여 작성된 Oracle 데이터 소스 컬럼에 NUMBER(38) 유형이 지정됩니다. DB2 정수의 범위가 2**31에서 (-2**31)-1까지이며 대략 NUMBER(9)와 동일하므로 이 Oracle 데이터 유형의 별칭 컬럼에는 로컬 데이터 유형 FLOAT가 지정됩니다. 이 경우 DB2 정수 컬럼과 Oracle 정수 컬럼의 조인은 DB2 데이터 소스에서 발생할 수 없습니다(조인 컬럼이 짧기 때문). 그러나 이 Oracle 정수 컬럼의 도메인을 DB2 INTEGER 데이터 유형에서 수용할 수 있는 경우 DB2 데이터 소스에서 조인이 발생할 수 있도록 ALTER NICKNAME문을 사용하여 로컬 데이터 유형을 변경하십시오.

- 컬럼 옵션

  ALTER NICKNAME문을 사용하여 별칭의 컬럼 옵션을 추가하거나 변경하십시오.

  VARCHAR_NO_TRAILING_BLANKS 옵션을 사용하여 뒤 공백이 없는 컬럼을 식별하십시오. 그런 다음 컴파일러 푸시다운 분석 단계에서는 이러한 컬럼에서 수행되는 모든 연산을 점검할 때 이 정보를 고려합니다. DB2 서버는 데이터 소스로 보내는 SQL문에서 사용할 다르지만 동일한 형식의 술어를 생성할 수 있습니다. 데이터 소스에 대해 평가되는 술어가 다를 수 있지만 최종 결과는 동일해야 합니다.

  NUMERIC_STRING 옵션을 사용하여 해당 컬럼의 값이 항상 뒤 공백이 없는 숫자인지 여부를 표시하십시오.

  표 52에서는 이러한 옵션에 대해 설명합니다.

표 52. 컬럼 옵션 및 해당 설정

옵션	유효한 설정	디폴트 설정
NUMERIC_STRING	Y: 이 컬럼에 숫자 데이터의 문자열만 있음을 지정합니다. 컬럼 데이터의 정렬을 방해할 수 있는 공백 문자가 포함되어 있지 않습니다. 이 옵션은 데이터 소스와 DB2 서버의 조합 시퀀스가 다른 경우에 유용합니다. 다른 조합 시퀀스로 인해 이 옵션으로 표시된 컬럼은 로컬(데이터 소스) 평가에서 제외되지 않습니다. 컬럼에 뒤 공백 문자가 있는 숫자 문자열만 포함된 경우 Y를 지정하지 마십시오. N: 이 컬럼이 숫자 데이터 문자열로 제한되지 않음을 지정합니다.	N

표 52. 컬럼 옵션 및 해당 설정 (계속)

옵션	유효한 설정	디폴트 설정
VARCHAR_NO_TRAILING_BLANKS	Y: 이 데이터 소스가 DB2 Data Server와 비슷하게 공백으로 채워지지 않은 VARCHAR 비교 시맨틱을 사용함을 지정합니다. 뒤 공백 문자가 없는 가변 길이 문자열의 경우 일부 데이터 서버의 공백으로 채워지지 않은 비교 시맨틱은 DB2 비교 시맨틱과 동일한 결과를 리턴합니다. 데이터 소스의 모든 VARCHAR 테이블 또는 뷰 컬럼에 뒤 공백 문자가 포함되지 않은 경우 이 값을 지정하십시오. N: 이 데이터 소스가 DB2 Data Server와 비슷하게 공백으로 채워지지 않은 VARCHAR 비교 시맨틱을 사용하지 않음을 지정합니다.	N

푸시다운 기회에 영향을 주는 쿼리 조회

쿼리는 여러 데이터 소스의 별칭을 사용하는 SQL 연산자를 참조할 수 있습니다. 집합 연산자(예: UNION)와 같은 하나의 연산자를 사용하는 두 개의 참조된 데이터 소스 결과를 결합하려면 DB2 서버에서 연산이 발생해야 합니다. 연산자는 리모트 데이터 소스에서 직접 평가할 수 없습니다.

페더레이티드 쿼리를 평가하는 위치를 판별하기 위한 지침:

db2expln 명령을 호출하여 시작할 수 있는 DB2 Explain 유틸리티는 쿼리가 평가되는 위치를 나타냅니다. 각 연산자의 실행 위치가 명령 출력에 포함되어 있습니다.

- 쿼리가 푸시다운되는 경우 표준 DB2 연산자인 RETURN 연산자가 표시됩니다. SELECT문이 별칭에서 데이터를 검색하는 경우 페더레이티드 데이터베이스 연산에 고유한 SHIP 연산자도 표시됩니다. 데이터 플로우의 서버 등록 정보를 변경하고 로컬 연산자를 리모트 연산자와 구분합니다. SELECT문은 데이터 소스에서 지원하는 SQL 통용어를 사용하여 생성됩니다.
- INSERT, UPDATE 또는 DELETE문을 리모트 데이터 소스에 완전히 푸시다운할 수 있는 경우 액세스 플랜에 SHIP 연산자가 표시되지 않을 수 있습니다. RETURN 연산자에 대해 리모트로 실행되는 모든 INSERT, UPDATE 또는 DELETE문이 표시됩니다. 그러나 쿼리를 완전히 푸시다운할 수 없는 경우 SHIP 연산자는 리모트로 수행된 연산을 표시합니다.

DB2 서버 대신 데이터 소스에서 쿼리가 평가되는 이유 이해

푸시다운 기회를 증가시킬 수 있는 방법을 조사할 때 다음과 같은 중요한 질문을 고려하십시오.

- 왜 이 술어가 리모트로 평가되지 않습니까?

선택적 술어를 사용하여 행을 필터링하고 네트워크 트래픽을 줄일 수 있는 경우 이러한 질문이 생성됩니다. 또한 리모트 술어 평가는 동일한 데이터 소스의 두 테이블 간 조인을 리모트로 평가할 수 있는지 여부에 영향을 줍니다.

조사할 영역은 다음과 같습니다.

- 서브쿼리 술어. 이 술어에 다른 데이터 소스와 관련된 서브쿼리가 있습니까? 이 술어에 이 데이터 소스에서 지원하지 않는 SQL 연산자를 사용하는 서브쿼리가 있습니까? 모든 데이터 소스가 서브쿼리 술어에서 집합 연산자를 지원하는 것은 아닙니다.
- 술어 함수. 이 술어에 이 리모트 데이터 소스에서 평가할 수 없는 함수가 포함되어 있습니까? 관계 연산자는 함수로 분류됩니다.
- 술어 바인드 요구사항. 리모트로 평가하는 경우 이 술어에 일부 값의 바인드가 필요합니까? 이 경우 이 데이터 소스에서 SQL 제한사항을 위반합니까?
- 전역 최적화. 옵티마이저는 로컬 처리를 더욱 비용 효율적인 것으로 결정할 수 있습니다.

• 왜 GROUP BY 연산자가 리모트로 평가되지 않습니까?

조사할 영역은 다음과 같습니다.

- GROUP BY 연산자에 대한 입력이 리모트로 평가됩니까? 답이 아니오인 경우 입력을 조사하십시오.
- 데이터 소스에 이 연산자에 대한 제한사항이 있습니까? 이에 대한 예는 다음과 같습니다.
  - 제한된 GROUP BY 항목 수
  - 결합된 GROUP BY 항목의 제한된 바이트 수
  - GROUP BY 목록에만 해당되는 컬럼 스펙
- 데이터 소스가 이 SQL 연산자를 지원합니까?
- 전역 최적화. 옵티마이저는 로컬 처리를 더욱 비용 효율적인 것으로 결정할 수 있습니다.
- GROUP BY절에 문자 표현식이 있습니까? 있는 경우 리모트 데이터 소스 및 DB2 서버에서 대소문자 구분 여부가 동일한지 검증하십시오.

• 왜 집합 연산자가 리모트로 평가되지 않습니까?

조사할 영역은 다음과 같습니다.

- 동일한 리모트 데이터 소스에서 두 피연산자를 완전히 평가합니까? 예가 답이어야 하지만 아니오인 경우 각 피연산자를 조사하십시오.
- 데이터 소스는 이 집합 연산자에 대한 제한사항을 설정합니까? 예를 들어, 대형 오브젝트(LOB) 또는 LONG 필드 데이터는 이 특정 집합 연산자의 유효한 입력입니까?

- 왜 ORDER BY 연산은 리모트로 평가되지 않습니까?

  조사할 영역은 다음과 같습니다.

  – ORDER BY 연산에 대한 입력이 리모트로 평가됩니까? 답이 아니오인 경우 입력을 조사하십시오.
  – ORDER BY절에 문자 표현식이 있습니까? 답이 예인 경우 리모트 데이터 소스 및 DB2 서버에서는 다른 조합 시퀀스 또는 대소문자 구분을 사용합니다.
  – 리모트 데이터 소스는 이 연산자에 대한 제한사항을 설정합니까? 예를 들어, ORDER BY 항목 수에 제한이 있습니까? 리모트 데이터 소스가 ORDER BY 목록에 대한 컬럼 스펙을 제한합니까?

페더레이티드 데이터베이스에서 리모트 SQL 생성 및 전역 최적화:

관계 별칭을 사용하는 페더레이티드 데이터베이스 쿼리의 경우 액세스 전략에는 원래의 쿼리를 리모트 쿼리 단위 세트로 구분한 후 결과를 결합하는 작업이 포함될 수 있습니다. 이러한 리모트 SQL 생성은 쿼리에 맞게 전역으로 최적화된 액세스 전략을 작성하는 데 도움이 됩니다.

옵티마이저는 푸시다운 분석의 출력을 사용하여 각 연산을 DB2 서버에서 로컬로 평가하거나 데이터 소스에서 리모트로 평가할 것인지 결정합니다. 이 결정은 연산 평가 비용 뿐만 아니라 DB2 서버와 리모트 데이터 소스 간 데이터 및 메시지 전달 비용을 포함하여 비용 모델의 출력에 대한 결정에 기초합니다.

최적화된 쿼리를 작성하기 위한 것이지만 다음의 요인은 전역 최적화 및 쿼리 성능에 상당한 영향을 줍니다.

- 서버 특성
- 별칭 특성

전역 최적화에 영향을 주는 서버 옵션

다음의 데이터 소스 서버 옵션은 전역 최적화에 영향을 줄 수 있습니다.

- 처리 속도의 상대적 비율

  CPU_RATIO 서버 옵션을 사용하여 데이터 소스에서 처리 속도를 DB2 서버에서 처리 속도와 관련하여 지정하십시오. 비율이 낮으면 데이터 소스에서 처리 속도가 DB2 서버에서 처리 속도보다 빠름을 표시합니다. 이 경우 DB2 옵티마이저는 프로세서 집중 연산을 데이터 소스로 푸시다운하려고 고려할 수 있습니다.

- 입출력 속도의 상대적 비율

  IO_RATIO 서버 옵션을 사용하여 데이터 소스의 시스템 입출력 속도를 DB2 서버의 시스템 입출력 속도와 관련하여 지정하십시오. 비율이 낮으면 DB2 서버에서 입

출력 속도가 DB2 서버에서 입출력 속도보다 빠름을 표시합니다. 이 경우 옵티마이저는 입출력 집중 연산을 데이터 소스로 푸시다운하려고 고려할 수 있습니다.

- DB2 서버와 데이터 소스 간 통신 비율

  COMM_RATE 서버 옵션을 사용하여 네트워크 용량을 지정하십시오. 비율이 낮으면 DB2 서버와 데이터 소스 간 네트워크 통신 속도가 느림을 표시하며 DB2 옵티마이저는 이 데이터 소스와 서로 송수신하는 메시지 수를 줄이는 것이 좋습니다. 비율이 0으로 설정된 경우 옵티마이저는 최소 네트워크 트래픽이 필요한 액세스 플랜을 작성합니다.

- 데이터 소스 조합 시퀀스

  COLLATING_SEQUENCE 서버 옵션을 사용하여 데이터 소스 조합 시퀀스와 로컬 DB2 데이터베이스 조합 시퀀스의 일치 여부를 지정하십시오. 이 옵션이 Y로 설정되지 않은 경우 DB2 옵티마이저는 이 데이터 소스에서 검색된 데이터가 정렬되지 않은 것으로 간주합니다.

- 리모트 플랜 힌트

  PLAN_HINTS 서버 옵션을 사용하여 데이터 소스에서 플랜 힌트를 생성하거나 사용해야 함을 지정하십시오. 디폴트로 DB2 서버는 데이터 소스로 플랜 힌트를 보내지 않습니다.

  플랜 힌트는 데이터 소스에서 옵티마이저에게 추가 정보를 제공하는 명령문 조각입니다. 일부 쿼리의 경우 이 정보를 통해 성능을 향상시킬 수 있습니다. 플랜 힌트는 데이터 소스의 옵티마이저가 인덱스 사용 여부, 사용할 인덱스 또는 사용할 테이블 조인 시퀀스를 결정하는 데 도움이 될 수 있습니다.

  플랜 힌트를 사용할 수 있는 경우 데이터 소스로 보내는 쿼리에는 추가 정보가 들어 있습니다. 예를 들어, Oracle 옵티마이저로 보내는 플랜 힌트가 포함된 명령문은 다음과 비슷합니다.

  ```
  select /*+ INDEX (table1, t1index)*/
      col1
    from table1
  ```

  플랜 힌트는 문자열 `/*+ INDEX (table1, t1index)*/`입니다.

- DB2 옵티마이저 지식 기반의 정보

  DB2 서버에는 원시 데이터 소스에 대한 데이터가 들어 있는 옵티마이저 지식 기반이 있습니다. DB2 옵티마이저는 특정 데이터베이스 관리 시스템(DBMS)에서 생성할 수 없는 리모트 액세스 플랜을 생성하지 않습니다. 즉, DB2 서버는 리모트 데이터 소스의 옵티마이저가 이해하거나 승인할 수 없는 플랜을 생성하지 않습니다.

전역 최적화에 영향을 주는 별칭 특성

다음의 별칭 특정 요인이 전역 최적화에 영향을 줄 수 있습니다.

- 인덱스 고려사항

  쿼리를 최적화하기 위해 DB2 서버는 데이터 소스에서 인덱스에 대한 정보를 사용할 수 있습니다. 이러한 이유로 인해 사용 가능한 인덱스 정보가 최신 상태여야 합니다. 별칭에 대한 인덱스 정보는 별칭을 작성할 때 처음 획득합니다. 뷰 별칭에 대한 인덱스 정보는 수집하지 않습니다.

- 별칭에 대한 인덱스 스펙 작성

  별칭의 인덱스 스펙을 작성할 수 있습니다. 인덱스 스펙은 DB2 옵티마이저가 사용할 카탈로그에서 인덱스 정의(실제 인덱스가 아님)를 빌드합니다. CREATE INDEX SPECIFICATION ONLY문을 사용하여 인덱스 스펙을 작성하십시오. 별칭에 대한 인덱스 스펙을 작성할 때 사용하는 구문은 로컬 테이블에 대한 인덱스를 작성할 때 사용하는 구문과 비슷합니다. 다음과 같은 경우에 인덱스 스펙을 작성하십시오.

  – DB2 서버가 별칭 작성 중 데이터 소스에서 인덱스 정보를 검색할 수 없는 경우
  – 뷰 별칭의 인덱스를 원하는 경우
  – DB2 옵티마이저가 특정 별칭을 중첩된 루프 조인의 내부 테이블로 사용하도록 권장할 경우. 조인 컬럼에서 인덱스를 작성할 수 있습니다(없는 경우).

  뷰의 별칭에 대해 CREATE INDEX문을 발행하기 전에 필요 여부를 고려하십시오. 뷰가 인덱스가 있는 테이블에서 간단한 SELECT인 경우 데이터 소스에서 테이블에 대한 인덱스와 일치시키기 위해 별칭에 대한 로컬 인덱스를 작성하면 쿼리 성능을 상당히 향상시킬 수 있습니다. 그러나 간단한 SELECT문이 아닌 뷰(예: 두 테이블을 조인하여 작성하는 뷰)에서 인덱스를 로컬로 작성하는 경우 쿼리 성능이 낮아질 수 있습니다. 예를 들어, 두 테이블 간 조인인 뷰에서 인덱스를 작성하는 경우 옵티마이저는 이 뷰를 중첩된 루프 조인의 내부 요소로 선택할 수 있습니다. 조인이 여러 번 평가되므로 쿼리 성능이 낮아집니다. 대체 방법으로 데이터 소스 뷰에 참조된 각 테이블의 별칭을 작성한 후 두 별칭을 참조하는 DB2 서버에서 로컬 뷰를 작성할 수 있습니다.

- 카탈로그 통계 고려사항

  시스템 카탈로그 통계는 별칭의 전체 *크기*와 연관된 컬럼에서 값의 범위에 대해 설명합니다. 옵티마이저는 별칭이 들어 있는 쿼리를 처리하기 위해 최저 비용 경로를 계산할 때 이러한 통계를 사용합니다. 별칭 통계는 테이블 통계와 동일한 카탈로그 뷰에 저장됩니다.

  DB2 서버는 데이터 소스에 저장된 통계 데이터를 검색할 수 있지만 이 데이터에 대한 갱신사항을 자동으로 감지할 수 없습니다. 또한 DB2 서버는 데이터 소스에서 오

브젝트 정의의 변경사항을 자동으로 감지할 수 없습니다. 오브젝트의 통계 데이터(또는 정의)가 변경된 경우 다음을 수행할 수 있습니다.

- 데이터 소스에서 RUNSTATS 명령과 동등한 명령을 실행하고 현재 별칭을 삭제(drop)한 후 재작성할 수 있습니다. 오브젝트의 정의가 변경된 경우에 이 방법을 사용하십시오.
- SYSSTAT.TABLES 카탈로그 뷰에서 통계를 수동으로 갱신할 수 있습니다. 이 방법에서는 많은 단계가 필요하지 않지만 오브젝트의 정의가 변경된 경우에는 작동하지 않습니다.

페더레이티드 데이터베이스 쿼리의 전역 분석:

db2expln 명령을 호출하여 시작할 수 있는 DB2 Explain 유틸리티는 리모트 Explain 함수에서 지원하는 이러한 데이터 소스의 리모트 옵티마이저에서 생성하는 액세스 플랜을 나타냅니다. 각 연산자의 실행 위치가 명령 출력에 포함되어 있습니다.

쿼리의 유형에 따라 SHIP 또는 RETURN 연산자의 각 데이터 소스에 대해 생성된 리모트 SQL문을 찾을 수도 있습니다. 각 연산자의 세부사항을 조사하여 DB2 옵티마이저가 각 연산자에 대한 입력 및 출력으로 추정한 행 수를 확인할 수 있습니다.

DB2 최적화 결정 이해

성능을 증가시킬 수 있는 방법을 조사할 때 다음과 같은 중요한 질문을 고려하십시오.

- 왜 동일한 데이터 소스의 두 별칭 간 조인을 리모트로 평가하지 않습니까?

  조사할 영역은 다음과 같습니다.

  - 조인 연산. 리모트 데이터 소스가 이 연산을 지원할 수 있습니까?
  - Join 술어. Join 술어를 리모트 데이터 소스에서 평가할 수 있습니까? 답이 아니오인 경우 Join 술어를 조사하십시오.
- 왜 GROUP BY 연산자를 리모트로 평가하지 않습니까?

  연산자 구문을 확인하고 연산자를 리모트 데이터 소스에서 평가할 수 있는지 검증하십시오.
- 왜 리모트 데이터 소스에서 명령문을 완전히 평가하지 않습니까?

  DB2 옵티마이저는 비용 기반 최적화를 수행합니다. 푸시다운 분석이 모든 연산자를 리모트 데이터 소스에서 평가할 수 있다고 표시해도 옵티마이저는 비용 추정에 따라 전역 최적화 플랜을 생성합니다. 이 플랜에 영향을 줄 수 있는 매우 다양한 여러 가지 요인이 있습니다. 예를 들어, 리모트 데이터 소스는 원래의 쿼리에서 모든 연산을 처리할 수 있지만 처리 속도는 DB2 서버의 처리 속도보다 훨씬 느릴 수 있으며 대신 DB2 서버에서 연산을 수행하는 편이 더 유리할 수도 있습니다. 결과가 만족스럽지 않은 경우 SYSCAT.SERVEROPTIONS 카탈로그 뷰에서 서버 통계를 검증하십시오.

- 옵티마이저가 생성하고 리모트 데이터 소스에서 모두 평가하는 플랜은 리모트 데이터 소스에서 직접 실행되는 원래의 쿼리보다 왜 성능이 훨씬 낮습니까?

  조사할 영역은 다음과 같습니다.

  – DB2 옵티마이저에서 생성하는 리모트 SQL문. 이 명령문이 원래의 쿼리와 동일한지 확인하십시오. 술어 순서가 변경되었는지 점검하십시오. 좋은 쿼리 옵티마이저는 쿼리에서 술어 순서에 민감하지 않습니다. 리모트 데이터 소스의 옵티마이저는 입력 술어의 순서에 따라 다른 플랜을 생성할 수 있습니다. DB2 서버에 입력할 때 술어의 순서를 수정하거나 리모트 데이터 소스의 서비스 센터에 문의하여 지원을 받으십시오.

    술어 교체를 점검할 수도 있습니다. 좋은 쿼리 옵티마이저는 동등한 술어 교체에 민감하지 않습니다. 리모트 데이터 소스의 옵티마이저는 입력 술어에 따라 다른 플랜을 생성할 수 있습니다. 예를 들어, 일부 옵티마이저는 술어의 이행 종결문을 생성할 수 없습니다.

  – 추가 함수. 리모트 SQL문에 원래의 쿼리에 없는 함수가 있습니까? 이러한 일부 함수는 데이터 유형을 변환하는 데 사용될 수 있습니다. 필요한 함수인지 검증하십시오.

데이터 액세스 메소드

쿼리 옵티마이저는 SQL 또는 XQuery문을 컴파일할 때 쿼리를 충족시키는 여러 가지 방법의 실행 비용을 추정합니다.

이러한 추정 값에 따라 옵티마이저는 최적의 액세스 플랜을 선택합니다. *액세스 플랜*은 SQL 또는 XQuery문을 분석하는 데 필요한 연산의 순서를 지정합니다. 응용프로그램이 바인드되면 패키지가 작성됩니다. 이 *패키지*에는 해당 응용프로그램에 있는 모든 정적 SQL 및 XQuery문에 대한 액세스 플랜이 들어 있습니다. 동적 SQL 및 XQuery문의 액세스 플랜이 런타임에 작성됩니다.

테이블의 데이터에 액세스하는 세 가지 방법이 있습니다.

- 전체 테이블을 순차적으로 스캔
- 테이블의 인덱스에 액세스하여 특정 행 찾기
- 스캔 공유

보통 WHERE절에 지정된 술어에 정의되어 있는 조건에 따라 행을 필터링할 수 있습니다. 액세스한 테이블에서 선택된 행들을 조인하여 결과 세트를 작성하고 출력을 그룹화하거나 정렬하여 이 데이터를 추가로 처리할 수 있습니다.

DB2 9.7부터 시작하여 다른 스캔의 버퍼 풀 페이지를 사용하는 스캔 기능인 *스캔 공유*는 디폴트 동작입니다. 스캔 공유는 워크로드 동시성 및 성능을 향상시킵니다. 스캔 공유를 사용하여 시스템은 많은 동시 응용프로그램을 지원할 수 있으며 쿼리 성능이 증

가하고 시스템 처리량이 증가하므로 스캔 공유에 참여하지 않는 쿼리도 혜택을 받을 수 있습니다. 스캔 공유는 특히 테이블 스캔 또는 대형 테이블의 MDC(다차원 클러스터링) 블록 인덱스 스캔 등을 수행하는 응용프로그램이 있는 환경에서 효과적입니다. 컴파일러는 스캔 유형, 목적, 분리 수준, 레코드 당 완료된 작업량 등에 따라 스캔이 스캔 공유에 참여할 수 있는지 여부를 판별합니다.

인덱스 스캔을 통한 데이터 액세스

데이터베이스 관리 프로그램이 출력을 정렬하거나 요청된 컬럼 데이터를 직접 검색하기 위해(*인덱스 전용 액세스*) 기본 테이블에 액세스하기 전에 인덱스에 액세스하여 규정 행 세트를 제한할 때 *인덱스 스캔*이 발생합니다(지정된 인덱스 범위에서 행 스캔).

지정된 인덱스 범위에서 행을 스캔할 때 인덱스 컬럼을 비교하는 쿼리의 값으로 *인덱스 스캔 범위*(스캔의 시작 및 중지 지점)를 판별합니다. 인덱스 전용 액세스의 경우 요청된 모든 데이터가 인덱스에 있으므로 인덱스화된 테이블에 액세스할 필요가 없습니다.

ALLOW REVERSE SCANS 옵션을 사용하여 인덱스를 작성하는 경우 정의할 때 사용한 방향과 반대 방향으로 스캔을 수행할 수도 있습니다.

옵티마이저는 적합한 인덱스가 작성되지 않았거나 인덱스 스캔 비용이 많이 드는 경우 테이블 스캔을 선택합니다. 테이블이 작거나 인덱스 클러스터링 비율이 낮거나 쿼리에 대부분의 테이블 행이 필요하거나 파티션된 인덱스(특정의 경우 순서를 보존할 수 없는)가 사용될 때 추가 정렬이 필요한 경우 인덱스 스캔 비용이 증가할 수 있습니다. 액세스 플랜이 테이블 스캔 또는 인덱스 스캔을 사용할 것인지를 판별하려면 DB2 Explain 기능을 사용하십시오.

범위를 제한하는 인덱스 스캔

특정 쿼리에 인덱스를 사용할 수 있는지 여부를 판별하기 위해 옵티마이저는 첫 번째 컬럼부터 인덱스의 각 컬럼을 평가하여 WHERE절에서 등식 및 기타 술어를 충족시키는 데 사용할 수 있는지 확인합니다. 술어는 비교 연산을 표현하거나 내재한 WHERE 절의 검색 조건 요소입니다. 술어를 사용하면 다음과 같은 경우 인덱스 스캔의 범위를 제한할 수 있습니다.

- IS NULL 또는 IS NOT NULL에 대한 테스트
- 완전 및 포함 부등식에 대한 테스트
- 상수, 호스트 변수, 상수로 평가되는 표현식 또는 키워드와 비교한 등식에 대한 테스트
- ANY, ALL 또는 SOME이 포함되지 않은 서브쿼리와 비교한 등식에 대한 테스트. 이 서브쿼리에는 바로 위의 상위 쿼리 블록(즉, 이 서브쿼리가 subselect인 select)에 대한 상관 컬럼 참조가 없어야 합니다.

다음 예는 인덱스를 사용하여 스캔 범위를 제한하는 방법을 나타냅니다.

- 다음과 같은 정의가 있는 인덱스를 가정하십시오.

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

  다음의 술어를 사용하여 IX1 인덱스를 사용하는 스캔 범위를 제한할 수 있습니다.

```
where
  name = :hv1 and
  dept = :hv2
```

  또는

```
where
  mgr = :hv1 and
  name = :hv2 and
  dept = :hv3
```

  두 번째 WHERE절은 술어를 인덱스에 키 컬럼이 표시되는 순서대로 지정할 필요가 없음을 나타냅니다. 예에서는 호스트 변수를 사용하지만 매개변수 표시문자, 표현식 또는 상수와 같은 기타 변수를 대신 사용할 수 있습니다.

  다음 WHERE절에서 NAME 및 DEPT를 참조하는 술어만 인덱스 스캔의 범위를 제한하는 데 사용됩니다.

```
where
  name = :hv1 and
  dept = :hv2 and
  salary = :hv4 and
  years = :hv5
```

  이러한 컬럼을 마지막 두 개의 인덱스 키 컬럼과 구분하는 키 컬럼(MGR)이 있으므로 정렬은 해제됩니다. 그러나 `name = :hv1` 및 `dept = :hv2` 술어로 범위가 판별된 후 다른 술어를 나머지 인덱스 키 컬럼과 비교하여 평가할 수 있습니다.

- ALLOW REVERSE SCANS 옵션을 사용하여 작성한 인덱스를 고려하십시오.

```
create index iname on tname (cname desc) allow reverse scans
```

  이 경우 인덱스(INAME)는 CNAME 컬럼의 내림차순 값에 기초합니다. 내림차순으로 실행되는 스캔에 대한 인덱스가 정의되어 있지만 스캔을 오름차순으로 수행할 수 있습니다. 인덱스 사용은 액세스 플랜 작성 및 고려 시 옵티마이저가 제어합니다.

부등식을 테스트하는 인덱스 스캔

특정 부등식 술어는 인덱스 스캔의 범위를 제한할 수 있습니다. 두 가지 유형의 부등식 술어가 있습니다.

- 완전 부등식 술어

  범위 제한 술어에 사용되는 완전 부등식 연산자는 초과(>) 및 미만(<)입니다.

완전 부등식 술어가 있는 하나의 컬럼만 인덱스 스캔 범위 제한 시 고려합니다. 다음 예에서 NAME 및 DEPT 컬럼의 술어를 사용하여 범위를 제한할 수 있지만 MGR 컬럼의 술어는 이러한 용도로 사용할 수 없습니다.

```
where
  name = :hv1 and
  dept > :hv2 and
  dept < :hv3 and
  mgr < :hv4
```

- 포함 부등식 술어

  범위 제한 술어에 사용되는 포함 부등식 연산자는 다음과 같습니다.

  – >= 및 <=

  – BETWEEN

  – LIKE

  포함 부등식 술어가 있는 여러 컬럼을 인덱스 스캔 범위 제한 시 고려할 수 있습니다. 다음 예에서 모든 술어는 범위를 제한하는 데 사용할 수 있습니다.

```
where
  name = :hv1 and
  dept >= :hv2 and
  dept <= :hv3 and
  mgr <= :hv4
```

`:hv2 = 404`, `:hv3 = 406` 및 `:hv4 = 12345`로 가정하십시오. 데이터베이스 관리 프로그램은 404 및 405 부서의 인덱스를 스캔하지만 직원 번호(MGR 컬럼)가 12345보다 큰 첫 번째 관리자에 도달하면 406 부서 스캔을 중지합니다.

데이터를 정렬하는 인덱스 스캔

쿼리에 정렬된 출력이 필요하면 인덱스를 사용하여 첫 번째 인덱스 키 컬럼부터 시작하여 정렬 컬럼이 인덱스에 연속으로 표시되는 경우 인덱스를 사용하여 데이터를 정렬할 수 있습니다. 정렬은 ORDER BY, DISTINCT, GROUP BY, 『= ANY』 서브쿼리, 『> ALL』 서브쿼리, 『< ALL』 서브쿼리, INTERSECT 또는 EXCEPT 및 UNION과 같은 연산을 사용할 경우 발생합니다. 예외는 다음과 같습니다.

- 인덱스가 파티션된 경우, 인덱스 키 컬럼 앞에 테이블 파티셔닝 키 컬럼이 붙거나, 파티션이 하나의 파티션을 제외하고 모두 제거하는 경우에만 데이터 순서에 사용할 수 있습니다.
- 『상수 값』 또는 상수로 평가되는 표현식과 비교하는 등식에 대해 인덱스 키 컬럼을 테스트하는 경우 순서 지정된 컬럼은 첫 번째 인덱스 키 컬럼과 다를 수 있습니다.

다음 쿼리를 고려해보십시오.

```
where
  name = 'JONES' and
  dept = 'D93'
order by mgr
```

이 쿼리에서 NAME과 DEPT는 항상 동일한 값이고 정렬되므로 인덱스를 사용하여 행을 정렬할 수 있습니다. 즉, 이전 WHERE 및 ORDER BY절은 다음과 동등합니다.

```
where
  name = 'JONES' and
  dept = 'D93'
order by name, dept, mgr
```

고유 인덱스를 사용하여 정렬 순서 요구사항을 자를 수도 있습니다. 다음의 인덱스 정의 및 ORDER BY절을 고려하십시오.

```
UNIQUE INDEX IX0: PROJNO  ASC

select projno, projname, deptno
  from project
  order by projno, projname
```

IX0 인덱스는 PROJNO가 고유함을 보장하므로 PROJNAME 컬럼에서 추가적인 정렬은 필요하지 않습니다. 각 PROJNO 값마다 하나의 PROJNAME 값만 있습니다.

인덱스 액세스 유형

쿼리가 테이블에서 필요한 모든 데이터를 테이블의 인덱스에서 검색할 수 있음을 옵티마이저가 확인하는 경우가 있습니다. 다른 경우에는 옵티마이저가 여러 인덱스를 사용하여 테이블에 액세스할 수 있습니다. RCT(range-clustered table)의 경우 데이터 레코드의 위치를 계산하는『가상』인덱스를 통해 데이터에 액세스할 수 있습니다.

인덱스 전용 액세스

테이블에 액세스하지 않고 필요한 모든 데이터를 인덱스에서 검색할 수도 있습니다. 이 기능을 *인덱스 전용 액세스*라고 합니다. 예를 들어, 다음의 인덱스 정의를 참조하십시오.

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

다음 쿼리는 기본 테이블을 읽지 않고 인덱스에만 액세스하여 충족시킬 수 있습니다.

```
select name, dept, mgr, salary
  from employee
  where name = 'SMITH'
```

그러나 보통 필수 컬럼은 인덱스에 표시되지 않습니다. 이러한 컬럼에서 데이터를 검색하려면 테이블 행을 읽어야 합니다. 옵티마이저가 인덱스 전용 액세스를 선택할 수 있도록 하려면 포함 컬럼을 사용하여 고유 인덱스를 작성하십시오. 예를 들어, 다음의 인덱스 정의를 참조하십시오.

```
create unique index ix1 on employee
  (name asc)
  include (dept, mgr, salary, years)
```

이 인덱스는 NAME 컬럼이 고유함을 나타내며 DEPT, MGR, SALARY 및 YEARS 컬럼의 데이터를 저장하고 유지보수합니다. 이러한 방식으로 다음의 쿼리는 인덱스에만 액세스하여 충족시킬 수 있습니다.

```
select name, dept, mgr, salary
  from employee
  where name = 'SMITH'
```

포함 컬럼의 추가 스토리지 스페이스 및 유지보수 비용을 맞췄는지 여부를 고려해야 합니다. 포함 컬럼을 이용하는 쿼리가 거의 실행되지 않은 경우 비용을 맞출 수 없습니다.

복수 인덱스 액세스

옵티마이저는 동일한 테이블의 여러 인덱스를 스캔하도록 선택하여 WHERE절의 술어를 충족시킬 수 있습니다. 예를 들어, 다음 두 개의 인덱스 정의를 참조하십시오.

```
INDEX IX2:  DEPT    ASC
INDEX IX3:  JOB     ASC,
            YEARS   ASC
```

다음 술어는 이러한 두 개의 인덱스를 사용하여 충족시킬 수 있습니다.

```
where
  dept = :hv1 or
  (job = :hv2 and
  years >= :hv3)
```

IX2 인덱스를 스캔하면 `dept = :hv1` 술어를 충족시키는 레코드 ID(RID) 목록이 작성됩니다. IX3 인덱스를 스캔하면 `job = :hv2 and years >= :hv3` 술어를 충족시키는 RID 목록이 작성됩니다. 이러한 두 개의 RID 목록을 결합하고 테이블에 액세스하기 전에 중복을 제거합니다. 이를 *Index ORing*이라고 합니다.

Index ORing은 다음 예와 같이 IN절에서 지정하는 술어에도 사용할 수 있습니다.

```
where
  dept in (:hv1, :hv2, :hv3)
```

Index ORing은 중복 RID를 제거하기 위한 것이지만 *Index ANDing*은 공통 RID를 찾기 위한 것입니다. Index ANDing은 동일한 테이블의 해당 컬럼에서 여러 인덱스를 작성하는 응용프로그램이 해당 테이블에 대해 여러 AND 술어를 사용하여 쿼리를 실행하는 경우 발생합니다. 각 인덱스 컬럼에 대한 다중 인덱스 스캔은 비트맵을 작성하

도록 해시된 값을 작성합니다. 두 번째 비트맵은 최종 결과 세트의 규정 행을 생성하기 위해 첫 번째 비트맵을 프로브하는 데 사용됩니다. 예를 들어, 다음의 인덱스를 참조하십시오.

```
INDEX IX4: SALARY   ASC
INDEX IX5: COMM     ASC
```

이 인덱스는 다음의 술어를 분석하는 데 사용할 수 있습니다.

```
where
  salary between 20000 and 30000 and
  comm between 1000 and 3000
```

이 예에서 IX4 인덱스를 스캔하면 `salary between 20000 and 30000` 술어를 충족시키는 비트맵이 작성됩니다. IX5를 스캔하고 IX4의 비트맵을 프로브하면 두 술어를 모두 충족시키는 규정 RID 목록이 작성됩니다. 이 기능을 *동적 비트맵 AND 작업*이라고 합니다. 이 작업은 테이블에 충분한 카디널리티(cardinality)가 있거나 해당 컬럼에 규정 범위 내의 충분한 값이 있거나 등호 술어를 사용하는 경우 충분한 중복이 있는 경우에만 발생합니다.

다중 인덱스 스캔 시 동적 비트맵의 성능 이점을 실현하려면 **sortheap** 데이터베이스 구성 매개변수 및 **sheapthres** 데이터베이스 관리 프로그램 구성 매개변수의 값을 변경해야 합니다. 액세스 플랜에서 동적 비트맵을 사용하는 경우 추가적인 정렬 힙 스페이스가 필요합니다. **sheapthres**가 **sortheap**에 비교적 근접하게 설정된 경우(즉, 동시 쿼리당 2 또는 3배의 인수보다 작음) 다중 인덱스 액세스를 사용하는 동적 비트맵은 옵티마이저의 예상보다 훨씬 적은 메모리를 사용하여 작업해야 합니다. 해결 방법은 **sheapthres**의 값을 **sortheap** 관련 값으로 증가시키는 것입니다.

옵티마이저는 단일 테이블에 액세스할 때 Index ANDing과 Index ORing을 결합하지 않습니다.

RCT(range-clustered table)에서 인덱스 액세스

표준 테이블과 달리 RCT(range-clustered table)에는 행에 키 값을 맵핑하는 실제 인덱스(이전의 B-트리 인덱스와 비슷함)가 필요하지 않습니다. 대신 컬럼 도메인의 순차적인 특성을 가공하고 기능 맵핑을 사용하여 테이블에서 특정 행의 위치를 생성합니다. 이러한 맵핑 유형의 가장 간단한 예에서 범위 내의 첫 번째 키 값은 테이블의 첫 번째 행이고 범위 내의 두 번째 값은 테이블의 두 번째 행이 되는 등 계속 이와 같은 방식으로 진행됩니다.

옵티마이저는 테이블의 범위 클러스터링 등록 정보를 사용하여 완전히 클러스터된 인덱스(비용만 범위 클러스터링 기능을 계산함)에 기초한 액세스 플랜을 작성합니다. RCT(range-clustered table)는 원래의 키 값 순서를 보유하므로 테이블에 있는 행의 클러스터링이 보장됩니다.

인덱스 액세스 및 클러스터 비율

옵티마이저는 액세스 플랜을 선택할 때 디스크에서 버퍼 풀로 페이지를 페치하는 데 필요한 입출력 수를 추정합니다. 이미 버퍼 풀에 있는 페이지에서 행을 읽는 데 추가적인 입출력이 필요하지 않으므로 이 추정 값에는 버퍼 풀 사용량 예측이 포함됩니다.

인덱스 스캔의 경우 시스템 카탈로그의 정보는 옵티마이저가 버퍼 풀로 데이터 페이지를 읽는 입출력 비용을 추정하는 데 도움이 됩니다. SYSCAT.INDEXES 뷰에서 다음 컬럼의 정보를 사용합니다.

- CLUSTERRATIO 정보는 이 인덱스와 관련하여 테이블 데이터가 클러스터된 수준을 표시합니다. 이 수가 높으면 인덱스 키 시퀀스로 행이 잘 정렬되어 있습니다. 테이블 행이 인덱스 키 시퀀스에 가까운 경우 페이지가 버퍼에 있는 동안 데이터 페이지에서 행을 읽을 수 있습니다. 이 컬럼의 값이 -1인 경우 옵티마이저는 PAGE_FETCH_PAIRS 및 CLUSTERFACTOR 정보(사용 가능한 경우)를 사용합니다.
- PAGE_FETCH_PAIRS 컬럼에는 CLUSTERFACTOR 정보와 함께 다양한 크기의 버퍼 풀로 데이터 페이지를 읽는 데 필요한 입출력 수를 모델링한 숫자 쌍이 있습니다. DETAILED절을 지정하여 인덱스에 대해 RUNSTATS 명령을 호출하는 경우에만 이러한 컬럼의 데이터를 수집합니다.

인덱스 클러스터링 통계가 사용 불가능한 경우 옵티마이저는 인덱스와 관련하여 데이터의 클러스터링 수준이 낮음을 가정하는 디폴트값을 사용합니다. 데이터의 클러스터링 수준은 성능에 상당한 영향을 미칠 수 있으며 테이블에 정의된 인덱스 중 하나를 100%에 가까운 클러스터링 수준으로 유지해야 합니다. 일반적으로 인덱스의 키가 클러스터링 인덱스 키의 수퍼 세트를 표시하거나 두 인덱스의 키 컬럼 사이에 실제 상관이 있는 경우를 제외하고는 하나의 인덱스만 100% 클러스터될 수 있습니다.

테이블을 재구성할 때 행을 클러스터링하고 삽입 처리 중 계속 클러스터링을 유지하는 데 사용되는 인덱스를 지정할 수 있습니다. 갱신 및 삽입 조작은 테이블을 인덱스와 관련하여 거의 클러스터링하지 않으므로 테이블을 주기적으로 재구성해야 합니다. 삽입, 갱신 또는 삭제 조작을 자주 수행하는 테이블의 재구성 수를 줄이려면 ALTER TABLE 문에 PCTFREE절을 지정하십시오.

스캔 공유

*스캔 공유*는 한 스캔이 다른 스캔에서 수행한 작업을 이용할 수 있는 기능을 나타냅니다. 공유 작업의 예로는 디스크 페이지 읽기, 디스크 탐색, 버퍼 풀 컨텐츠 재사용, 압축 해제 등이 있습니다.

대형 테이블의 다차원 클러스터링(MDC) 블록 인덱스 스캔 또는 테이블 스캔과 같은 대량 스캔이 때때로 다른 스캔과의 페이지 읽기 공유에 적합합니다. 이러한 공유 스캔은 테이블의 임의 지점에서 시작되어 이미 버퍼 풀에 있는 페이지를 이용할 수 있습니다. 공유 스캔이 테이블의 끝에 이를 경우 처음부터 계속되고 시작된 지점에 이르면 완

료됩니다. 이것을 *래핑 스캔*이라고 합니다. 그림 25에서는 테이블 및 인덱스 모두에서 일반 스캔과 래핑 스캔 간 차이점을 보여줍니다.

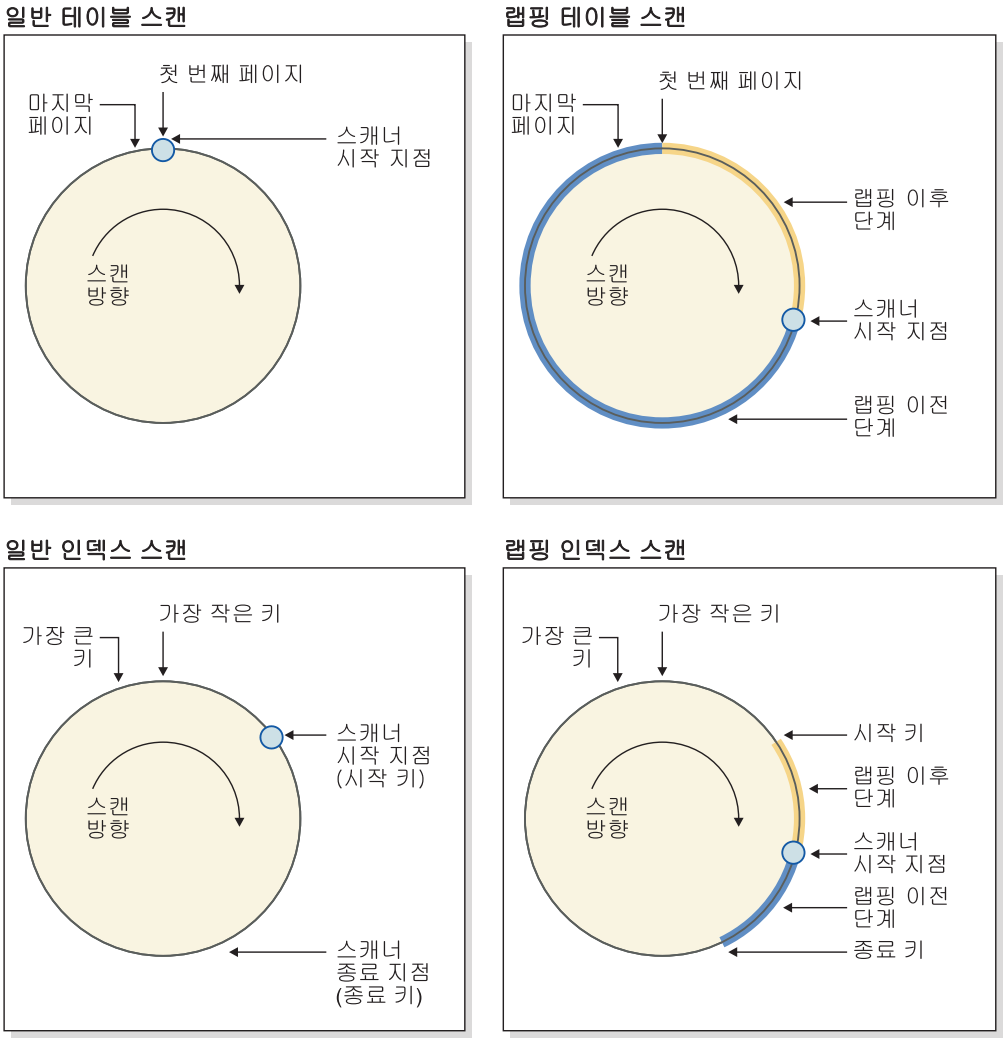


그림 25. 일반 스캔과 래핑 스캔의 개념 뷰

스캔 공유 기능은 디폴트로 사용되고, 스캔 공유 및 래핑에 대한 적격성은 SQL 컴파일러에 의해 자동으로 판별됩니다. 런타임에서 적격 스캔은 컴파일 시간에 알려지지 않았던 인수를 기반으로 공유 또는 래핑에 참여하거나 참여하지 않을 수 있습니다.

공유 스캐너는 *공유 그룹*에서 관리됩니다. 이러한 그룹은 공유의 이점이 최대화되도록 구성원을 가능한 한 오래 함께 보존합니다. 한 스캔이 다른 스캔보다 더 빠른 경우, 페이지 공유의 이점이 손실될 수 있습니다. 이 경우, 다른 스캔이 해당 페이지에 액세스하기 전에 첫 번째 스캔에서 액세스하는 버퍼 풀 페이지가 버퍼 풀에서 지워질 수도 있습니다. 데이터 서버는 두 스캔 간에 있는 버퍼 풀 페이지의 수를 통해 동일한 공유 그룹의 두 스캔 간 거리를 측정합니다. 데이터 서버는 또한 스캔의 속도를 모니터합니다. 동일한 공유 그룹의 두 스캔 간 거리가 너무 큰 경우, 버퍼 풀 페이지를 공유할 수 없을 수도 있습니다. 이런 효과를 줄이려면 더 빠른 스캔을 스로틀하여 더 느린 스캔이

데이터 페이지가 지워지기 전에 액세스할 수 있도록 할 수 있습니다. 그림 26에서는 두 개의 공유 세트(테이블용 하나와 블록 인덱스용 하나)를 보여줍니다. *공유 세트*는 동일한 액세스 메커니즘(예: 테이블 스캔 또는 블록 인덱스 스캔)을 통해 동일한 오브젝트(예: 테이블)에 액세스하는 공유 그룹의 콜렉션입니다. 테이블 스캔의 경우, 페이지 ID에 따라 페이지 읽기 순서가 증가하고, 블록 인덱스 스캔의 경우 키 값에 따라 페이지 읽기 순서가 증가합니다.

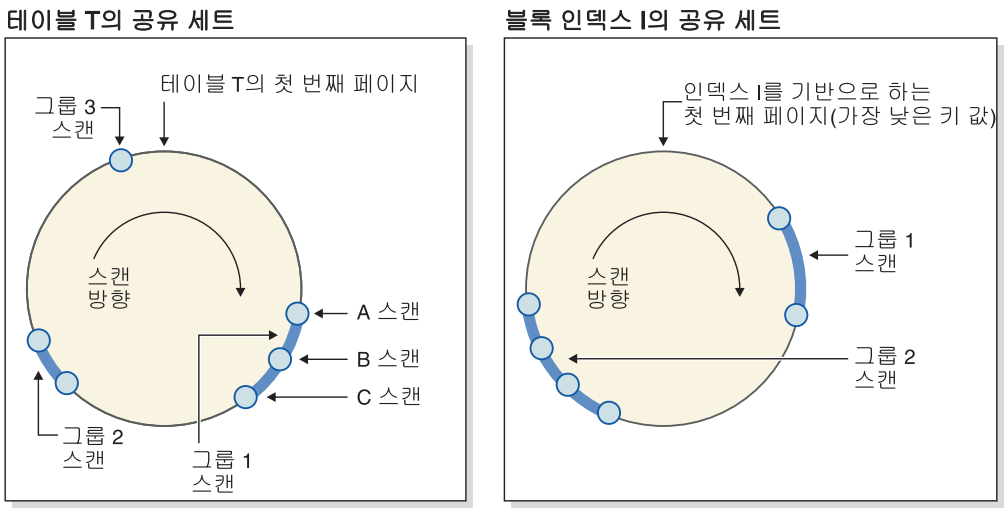


그림 26. 테이블 및 블록 인덱스 스캔 공유에 대한 공유 세트

그림에서는 버퍼 풀 컨텐츠가 그룹 내에서 재사용되는 방법도 보여줍니다. 그룹 1의 선행 스캔인 스캔 C를 고려해보십시오. 다음 스캔(A와 B)이 가까이에 있고 C에서 버퍼 풀로 가져온 페이지를 재사용할 가능성이 크므로 C와 그룹화됩니다.

높은 우선순위 스캐너는 낮은 우선순위 스캐너에 의해 스로틀되지 않으며, 대신 다른 공유 그룹으로 이동될 수 있습니다. 높은 우선순위 스캐너는 그룹의 더 낮은 우선순위 스캐너에 의해 수행되고 있는 작업에서 이익을 얻을 수 있는 그룹에 배치될 수 있습니다. 이런 이익이 사용 가능한 동안 해당 그룹에 있게 됩니다. 빠른 스캐너를 스로틀하거나 더 빠른 공유 그룹으로 이동함으로써(스캐너에서 발견할 경우) 데이터 서버는 공유 그룹을 조정하여 공유가 최적화 상태로 유지되도록 합니다.

db2pd 명령을 사용하여 스캔 공유에 대한 정보를 볼 수 있습니다. 예를 들어, 개별적 공유 스캔의 경우, db2pd 출력에서 스캔이 스로틀된 기간 및 스캔 속도와 같은 데이터가 표시됩니다. 공유 그룹의 경우, 명령 출력에서 그룹에 의해 공유된 페이지의 수 및 그룹 내 스캔 수가 표시됩니다.

EXPLAIN_ARGUMENT 테이블에는 테이블 스캔 및 인덱스 스캔에 대한 스캔 공유 정보를 포함하기 위한 새 행이 있습니다(db2exfmt 명령을 사용하여 이 테이블의 컨텐츠를 형식화하고 볼 수 있음).

최적화 프로파일을 사용하여 컴파일러가 스캔 공유에 대해 내린 결정을 겹쳐쓸 수 있습니다(『액세스 유형』 참조). 이러한 겹쳐쓰기는 특별히 필요한 경우에만 사용됩니다. 예를 들어, 결과 세트에서 반복 가능한 레코드 순서가 필요한 경우 래핑 힌트가 유용할 수 있지만 ORDER BY절(정렬을 트리거할 수 있음)을 피하게 됩니다. 그렇지 않으면, DB2 서비스에서 요청하지 않는 한 이러한 최적화 프로파일을 사용하지 말 것을 권장합니다.

조인

*조인*은 일반 정보 도메인에 기초하여 둘 이상의 테이블에 있는 데이터를 결합하는 프로세스입니다. 한 테이블의 행을 다른 테이블의 행과 쌍으로 지정합니다(해당 행의 정보가 결합 기준(*Join* 술어)에 따라 일치하는 경우).

예를 들어, 다음 두 개의 테이블을 참조하십시오.

TABLE1

PROJ	PROJ_ID
A	1
B	2
C	3
D	4

TABLE2

PROJ_ID	NAME
1	Sam
3	Joe
4	Mary
1	Sue
2	Mike

PROJ_ID 컬럼의 값이 동일하도록 TABLE1과 TABLE2를 조인하려면 다음 SQL문을 사용하십시오.

```
select proj, x.proj_id, name
  from table1 x, table2 y
  where x.proj_id = y.proj_id
```

이 경우 적합한 Join 술어는 `where x.proj_id = y.proj_id`입니다.

이 쿼리로 다음의 결과 세트가 작성됩니다.

PROJ	PROJ_ID	NAME
A	1	Sam
A	1	Sue
B	2	Mike
C	3	Joe
D	4	Mary

Join 술어의 특성과 테이블 및 인덱스 통계를 기초로 판별된 비용에 따라 옵티마이저는 다음의 조인 방법 중 하나를 선택합니다.

- 중첩된 루프 조인
- 병합 조인
- 해시 조인

두 테이블이 조인된 경우 한 테이블을 외부 테이블로 선택하고 다른 테이블을 조인의 내부 테이블로 간주합니다. 다른 테이블에 먼저 액세스하고 한 번만 스캔합니다. 내부 테이블을 여러 번 스캔하는지 여부는 조인 유형과 사용 가능한 인덱스에 따라 다릅니다. 쿼리가 두 개 이상의 테이블을 조인하는 경우에도 옵티마이저는 한 번에 두 테이블만 조인합니다. 필요하면 중간 결과를 보유할 임시 테이블이 작성됩니다.

INNER 또는 LEFT OUTER JOIN과 같은 명시적 조인 연산자를 제공하여 조인에서 테이블을 사용하는 방식을 판별할 수 있습니다. 그러나 이러한 방식으로 쿼리를 변경하기 전에 옵티마이저가 테이블을 조인하는 방법을 판별한 후 쿼리 성능을 분석하여 조인 연산자 추가 여부를 결정하도록 해야 합니다.

조인 방법

쿼리에서 테이블 조인을 요구하는 경우 옵티마이저는 세 개의 기본 조인 전략 즉, 중첩된 루프 조인, 병합 조인 또는 해시 조인 중 하나를 선택할 수 있습니다.

중첩된 루프 조인

중첩된 루프 조인은 다음의 두 가지 방법 중 하나로 수행합니다.

- 내부 테이블에서 액세스하는 외부 테이블의 각 행 스캔

  예를 들어, T1 테이블의 A 컬럼과 T2 테이블의 A 컬럼에는 다음의 값이 있습니다.

외부 테이블 **T1: A** 컬럼	내부 테이블 **T2: A** 컬럼
2	3
3	2
3	2
	3
	1

  T1 테이블과 T2 테이블 사이의 중첩된 루프 조인을 완료하기 위해 데이터베이스 관리 프로그램은 다음의 단계를 수행합니다.

  1. T1에서 첫 번째 행을 읽습니다. A의 값은 2입니다.
  2. 일치 (2)가 발견될 때까지 T2를 스캔한 후 두 행을 조인합니다.
  3. 테이블 끝에 접근할 때까지 2단계를 반복합니다.
  4. T1으로 돌아가서 다음 행 (3)을 읽습니다.
  5. 일치 (3)이 발견될 때까지 T2를 스캔한 후(첫 번째 행부터 시작) 두 행을 조인합니다.
  6. 테이블 끝에 접근할 때까지 5단계를 반복합니다.
  7. T1으로 돌아가서 다음 행 (3)을 읽습니다.
  8. (3)과 일치하는 모든 행을 조인하여 이전과 같이 T2를 스캔합니다.

- 액세스하는 외부 테이블의 각 행에 대해 내부 테이블에서 인덱스 찾아보기를 수행합니다.

이 방법은 다음과 같은 형식의 술어가 있는 경우에 사용할 수 있습니다.

```
expr(outer_table.column) relop inner_table.column
```

여기서 relop는 상대 연산자(예: =, >, >=, < 또는 <=)이고 expr은 외부 테이블의 유효한 표현식입니다. 예를 들어, 다음과 같습니다.

```
outer.c1 + outer.c2 <= inner.c1
outer.c4 < inner.c3
```

이 방법은 외부 테이블의 각 액세스에 대해 내부 테이블에서 액세스하는 행 수를 상당히 줄일 수 있습니다. 혜택의 정도는 Join 술어의 선택 빈도와 같인 여러 요인에 따라 다릅니다.

중첩된 루프 조인을 평가할 때 옵티마이저는 조인을 수행하기 전 외부 테이블 정렬 여부도 결정합니다. 조인 컬럼에 따라 외부 테이블을 정렬하는 경우 디스크의 페이지에 액세스하기 위한 내부 테이블에 대한 읽기 조작 수가 줄어들 수 있습니다. 버퍼 풀에 이미 있을 가능성이 높습니다. 조인이 많이 클러스터된 인덱스를 사용하여 내부 테이블에 액세스하며 외부 테이블이 정렬된 경우 액세스하는 인덱스 페이지 수가 최소한으로 줄어들 수 있습니다.

옵티마이저가 조인으로 나중에 정렬 비용이 매우 증가할 것으로 예상하는 경우 조인하기 전에 정렬을 수행하도록 선택할 수도 있습니다. GROUP BY, DISTINCT, ORDER BY 또는 병합 조인 조작을 지원하려면 이후에 정렬을 수행해야 할 수 있습니다.

병합 조인

병합 스캔 조인 또는 *병합 정렬 조인*이라고도 하는 병합 조인은 table1.column = table2.column 형식의 술어가 필요합니다. 이를 *동등 조인 술어*라고 합니다. 병합 조인은 인덱스 액세스 또는 정렬을 통해 조인 컬럼에서 정렬된 입력이 필요합니다. 조인 컬럼이 LONG 필드 컬럼 또는 대형 오브젝트(LOB) 컬럼인 경우 병합 조인을 사용할 수 없습니다.

병합 조인에서 조인된 테이블을 동시에 스캔합니다. 병합 조인의 외부 테이블은 한 번만 스캔합니다. 반복값이 외부 테이블에 나타나지 않으면 내부 테이블도 한 번만 스캔합니다. 반복값이 나타나면 내부 테이블에 있는 행 그룹을 다시 스캔할 수 있습니다.

예를 들어, T1 테이블의 A 컬럼과 T2 테이블의 A 컬럼에는 다음의 값이 있습니다.

외부 테이블 T1: A 컬럼	내부 테이블 T2: A 컬럼
2	1
3	2
3	2
	3
	3

T1 테이블과 T2 테이블 사이의 병합 조인을 완료하기 위해 데이터베이스 관리 프로그램은 다음의 단계를 수행합니다.

1. T1에서 첫 번째 행을 읽습니다. A의 값은 2입니다.
2. 일치 (2)가 발견될 때까지 T2를 스캔한 후 두 행을 조인합니다.
3. 행을 조인하여 컬럼이 일치하는 동안 계속 T2를 스캔합니다.
4. T2의 3을 읽은 후 T1으로 돌아가서 다음 행을 읽습니다.
5. T1의 다음 값은 T2와 일치하는 3이므로 행을 조인합니다.
6. 행을 조인하여 컬럼이 일치하는 동안 계속 T2를 스캔합니다.
7. T2의 끝에 접근하면 T1으로 돌아가서 다음 행을 확인합니다. T1의 다음 값이 T1의 이전 값과 동일하므로 T2의 첫 번째 3부터 시작하여 T2를 다시 스캔한다는 점을 참고하십시오. 데이터베이스 관리 프로그램은 이 위치를 기억합니다.

해시 조인

해시 조인은 컬럼 유형이 동일하도록 `table1.columnX = table2.columnY` 형식을 사용하는 하나 이상의 술어가 필요합니다. CHAR 유형인 컬럼의 경우 길이가 동일해야 합니다. DECIMAL 유형인 컬럼의 경우 정밀도 및 스케일이 동일해야 합니다. DECFLOAT 유형인 컬럼의 경우 정밀도가 동일해야 합니다. 컬럼은 LONG 필드 컬럼 또는 LOB 컬럼이 될 수 없습니다.

먼저 지정된 내부 테이블을 스캔하고 **sortheap** 데이터베이스 구성 매개변수에 지정된 정렬 힙에서 가져온 메모리 버퍼로 행을 복사합니다. Join 술어의 컬럼에서 계산한 해시 값에 따라 메모리 버퍼를 여러 섹션으로 나눕니다. 내부 테이블의 *크기*가 사용 가능한 정렬 힙 스페이스를 초과하면 선택한 섹션의 버퍼를 임시 테이블에 기록합니다.

내부 테이블이 처리되었으면 두 번째(또는 외부) 테이블을 스캔하고 해당 행을 먼저 Join 술어의 컬럼에 대해 계산된 해시 값과 비교하여 내부 테이블의 행과 일치시킵니다. 외부 행 컬럼의 해시 값이 내부 행 컬럼의 해시 값과 일치하면 실제 Join 술어 컬럼 값을 비교합니다.

임시 테이블에 기록되지 않은 테이블의 일부분에 해당하는 외부 테이블 행을 메모리에 있는 내부 테이블 행과 즉시 일치시킵니다. 내부 테이블의 해당 부분이 임시 테이블에 기록되었으면 외부 행도 임시 테이블에 기록됩니다. 마지막으로 임시 테이블에서 일치하는 테이블 부분 쌍을 읽고 해당 행의 해시 값을 일치시킨 후 Join 술어를 점검합니다.

해시 조인의 성능 이점을 완전히 활용하려면 **sortheap** 데이터베이스 구성 매개변수 및 **sheapthres** 데이터베이스 관리 프로그램 구성 매개변수의 값을 변경해야 합니다.

해시 루프 및 디스크에 대한 오버플로우를 방지할 수 있는 경우 해시 조인 성능은 최상의 상태가 됩니다. 해시 조인 성능을 조정하려면 **sheapthres**에 사용할 수 있는 최대

메모리 양을 추정한 후 **sortheap** 매개변수를 조정하십시오. 가능하면 많은 해시 루프 및 디스크 오버플로우를 방지할 수 있을 때까지 설정을 늘리지만 **sheapthres** 매개변수에 지정된 한계에 접근하지 마십시오.

sortheap 값을 늘리면 여러 정렬이 있는 쿼리의 성능도 향상됩니다.

최적의 조인을 선택하기 위한 전략

옵티마이저는 다양한 방법을 사용하여 쿼리에 대한 최적의 조인 전략을 선택합니다. 쿼리의 최적화 클래스로 판별된 이러한 방법 중 여러 검색 전략, 스타 스키마 조인, 초기 제외 조인 및 복합 테이블이 있습니다.

조인 열거 알고리즘은 옵티마이저가 탐색하는 플랜 결합 수의 중요한 결정자입니다.

- greedy 조인 열거
  - 스페이스 및 시간 요구사항과 관련하여 효율적입니다.
  - 단방향 열거를 사용합니다. 즉, 두 테이블의 조인 방법을 선택하면 추가 최적화 중 변경되지 않습니다.
  - 여러 테이블을 조인하면 최상의 액세스 플랜을 간과할 수 있습니다. 쿼리가 두 개 또는 세 개의 테이블만 조인하는 경우 greedy 조인 열거로 선택한 액세스 플랜은 동적 프로그래밍 조인 열거로 선택한 액세스 플랜과 동일합니다. 특히 쿼리에 명시적으로 지정되었거나 술어 이행 종결을 통해 내재적으로 생성된 동일한 컬럼에 여러 Join 술어가 있는 경우에 이 사항이 적용됩니다.
- 동적 프로그래밍 조인 열거
  - 스페이스 및 시간 요구사항과 관련하여 비효율적이므로 조인된 테이블 수가 증가하면 급격하게 증가합니다.
  - 최상의 액세스 플랜을 검색할 때 효율적이고 철저합니다.
  - DB2 for z/OS에서 사용하는 전략과 비슷합니다.

스타 스키마 조인

쿼리에 참조된 테이블들은 거의 항상 Join 술어로 관련되어 있습니다. 두 개의 테이블이 Join 술어 없이 조인된 경우 두 테이블의 카테시안 곱이 형성됩니다. 카테시안 곱에서 첫 번째 테이블의 모든 규정 행은 두 번째 테이블의 모든 규정 행과 조인됩니다. 이 경우 *크기*가 두 소스 테이블 *크기*의 외적이므로 보통 매우 큰 결과 테이블이 작성됩니다. 이러한 플랜은 올바로 수행되지 않을 수 있으므로 옵티마이저는 이러한 유형의 액세스 플랜 비용 판별도 방지합니다.

최적화 클래스가 9로 설정되었거나 스타 스키마와 같은 특별한 경우에만 예외가 발생합니다. *스타 스키마*에는 *사실 테이블*이라고 하는 중앙 테이블과 *차원 테이블*이라고 하는 다른 테이블이 있습니다. 차원 테이블에는 쿼리에 관계없이 사실 테이블에 차원 테이블을 연결하는 단일 조인만 있습니다. 각 차원 테이블에는 사실 테이블의 특정 컬럼

에 대한 정보를 확장하는 추가 값이 있습니다. 일반 쿼리는 차원 테이블의 값을 참조하는 여러 로컬 술어로 구성되어 있으며 차원 테이블을 사실 테이블에 연결하는 조인 술어를 포함하고 있습니다. 이러한 쿼리의 경우 대형 사실 테이블에 액세스하기 전에 여러 개의 작은 차원 테이블의 카테시안 곱을 계산하면 좋습니다. 이러한 기술은 여러 개의 Join 술어가 다중 컬럼 인덱스와 일치하는 경우에 유용합니다.

DB2 데이터 서버는 최소한 두 개의 차원 테이블이 있는 스타 스키마를 사용하여 설계된 데이터베이스에 대한 쿼리를 인식할 수 있으며 차원 테이블의 카테시안 곱을 계산하는 가능한 플랜을 포함하도록 검색 스페이스를 늘릴 수 있습니다. 카테시안 곱을 계산하는 플랜의 추정된 비용이 가장 낮은 경우 옵티마이저가 선택합니다.

이 스타 스키마 조인 전략은 조인에서 1차 키 인덱스를 사용하는 것으로 가정합니다. 다른 시나리오는 외부 키 인덱스와 관련되어 있습니다. 사실 테이블의 외부 키 컬럼이 단일 컬럼 인덱스이며 모든 차원 테이블에서 비교적 선택 빈도가 높은 경우 다음의 스타 스키마 조인 기술을 사용할 수 있습니다.

1. 다음을 수행하여 각 차원 테이블을 처리하십시오.
   - 차원 테이블과 사실 테이블의 외부 키 인덱스 간 세미조인 수행
   - 비트맵을 동적으로 작성하도록 레코드 ID(RID) 값 해싱
2. 각 비트맵마다 이전 비트맵에 대해 AND 술어를 사용하십시오.
3. 최종 비트맵 처리 후 남아 있는 RID를 판별하십시오.
4. 이러한 RID를 선택적으로 정렬하십시오.
5. 기본 테이블 행을 페치하십시오.
6. SELECT절에 필요한 차원 테이블의 컬럼에 액세스하여 사실 테이블을 각 차원 테이블과 다시 조인하십시오.
7. 레지듀얼(Residual) 술어를 재적용하십시오.

이 기술에는 다중 컬럼 인덱스가 필요하지 않습니다. 사실 테이블과 차원 테이블 간 명시적 참조 무결성 제한조건은 반드시 필요하지는 않지만 권장됩니다.

스타 스키마 조인 기술에서 작성 및 사용되는 동적 비트맵에는 정렬 힙 메모리가 필요하며 크기는 **sortheap** 구성 매개변수에 지정되어 있습니다.

초기 제외 조인

옵티마이저는 테이블 중 한 테이블만의 각 행을 다른 테이블의 최대 한 행과 조인해야 함을 감지하면 초기 제외 조인을 선택할 수 있습니다.

한 테이블의 키 컬럼(들)에 Join 술어가 있는 경우 초기 제외 조인이 가능합니다. 예를 들어, 직원의 이름 및 각 직원의 직속 상사를 리턴하는 다음의 쿼리를 참조하십시오.

```
select employee.name as employee_name,
    manager.name as manager_name
  from employee as employee, employee as manager
  where employee.manager_id = manager.id
```

ID 컬럼이 EMPLOYEE 테이블에서 키 컬럼이고 모든 직원에게 최대 한 명의 상사가 있는 경우 이 조인을 사용하면 MANAGER 테이블에서 일치하는 후속 행을 검색할 필요가 없습니다.

쿼리에 DISTINCT절이 있는 경우에도 초기 제외 조인이 가능합니다. 예를 들어, $30000 이상의 가격으로 판매되는 모델을 생산하는 자동차 제조업체의 이름을 리턴하는 다음의 쿼리를 참조하십시오.

```
select distinct make.name
  from make, model
  where
    make.make_id = model.make_id and
    model.price > 30000
```

각 자동차 제조업체마다 제조된 모델 중 하나가 $30000 이상의 가격으로 판매되는지 여부만을 판별해야 합니다. 자동차 제조업체를 $30000 이상으로 판매되는 모든 제조 모델과 조인할 필요가 없습니다. 쿼리 결과의 정확성에 도움이 되지 않습니다.

조인이 MIN 또는 MAX 집계 함수에 GROUP BY절을 제공하는 경우에도 초기 제외 조인이 가능합니다. 예를 들어, 특정 주식의 *종가*가 *개장가*보다 최소한 10% 이상인 2000년 이전의 최근 날짜로 주식 기호를 리턴하는 다음의 쿼리를 참조하십시오.

```
select dailystockdata.symbol, max(dailystockdata.date) as date
  from sp500, dailystockdata
  where
    sp500.symbol = dailystockdata.symbol and
    dailystockdata.date < '01/01/2000' and
    dailystockdata.close / dailystockdata.open >= 1.1
  group by dailystockdata.symbol
```

*규정 세트*는 날짜 및 가격 요구사항을 충족시키고 SP500 테이블의 특정 주식 기호와 조인되는 DAILYSTOCKDATA 테이블의 행 세트입니다. DAILYSTOCKDATA 테이블의 규정 세트(SP500 테이블의 각 주식 기호 행마다 다름)가 DATE에서 내림차순으로 정렬된 경우 첫 번째 행이 특정 기호의 최근 날짜를 표시하므로 각 기호의 규정 세트에서 첫 번째 행만 리턴해야 합니다. 규정 세트의 다른 행은 필요하지 않습니다.

복합 테이블

테이블 쌍을 조인하는 결과로 새 테이블(*복합 테이블*이라고 함)이 작성된 경우 이 테이블은 보통 다른 내부 테이블과 조인할 외부 테이블이 됩니다. 이를 *복합 외부 조인*이라고 합니다. 특히 greedy 조인 열거 기술을 사용할 때 두 테이블을 조인하는 결과로 나중에 조인할 내부 테이블을 작성하면 유용합니다. 조인의 내부 테이블이 둘 이상의 테이블을 조인한 결과로 구성된 경우 이 플랜을 *복합 내부 조인*이라고 합니다. 예를 들어, 다음 쿼리를 참조하십시오.

```
select count(*)
  from t1, t2, t3, t4
  where
    t1.a = t2.a and
    t3.a = t4.a and
    t2.z = t3.z
```

T1 테이블과 T2 테이블을 조인(T1xT2)한 후 T3와 T4를 조인하고 마지막으로 외부 테이블로 첫 번째 조인 결과를 선택하고 내부 테이블로 두 번째 조인 결과를 선택하면 좋습니다. 최종 플랜((T1xT2) x (T3xT4))에서 조인 결과(T3xT4)를 *복합 내부 조인*이라고 합니다. 쿼리 최적화 클래스에 따라 옵티마이저는 조인의 내부 테이블이 되는 최대 테이블 수에 다른 제한조건을 설정합니다. 복합 내부 조인은 최적화 클래스 5, 7 및 9에서 허용됩니다.

파티션된 데이터베이스 환경에서 복제된 구체화된 쿼리 테이블

복제된 구체화된 쿼리 테이블(MQT)을 통해 데이터베이스가 테이블 데이터의 사전 계산된 값을 관리할 수 있으므로 파티션된 데이터베이스 환경에서 자주 실행하는 조인의 성능이 향상됩니다.

이 컨텍스트에서 복제된 MQT는 데이터베이스 내 복제와 관계가 있다는 점을 참고하십시오. 데이터베이스 내 복제는 서브스크립션, 제어 테이블, 다른 데이터베이스 및 다른 운영 체제에 있는 데이터와 관련되어 있습니다.

다음 예에서 아래 사항을 참조하십시오.

- SALES 테이블은 다중 파티션 테이블 스페이스 REGIONTABLESPACE에 있으며 REGION 컬럼에서 분할되어 있습니다.
- EMPLOYEE 및 DEPARTMENT 테이블은 단일 파티션 데이터베이스 파티션 그룹에 있습니다.

EMPLOYEE 테이블의 정보에 따라 복제된 MQT를 작성하십시오.

```
create table r_employee as (
  select empno, firstnme, midinit, lastname, workdept
    from employee
  )
  data initially deferred refresh immediate
  in regiontablespace
  replicated
```

복제된 MQT의 내용을 갱신하십시오.

```
refresh table r_employee
```

REFRESH문을 사용한 후 다른 테이블과 마찬가지로 복제된 테이블에 대해 runstats 유틸리티를 호출해야 합니다.

다음의 쿼리는 직원별 급여, 부서 총액 및 전체 총액을 계산합니다.

```
select d.mgrno, e.empno, sum(s.sales)
  from department as d, employee as e, sales as s
  where
    s.sales_person = e.lastname and
    e.workdept = d.deptno
  group by rollup(d.mgrno, e.empno)
  order by d.mgrno, e.empno
```

한 데이터베이스 파티션에만 상주하는 EMPLOYEE 테이블을 사용하는 대신 데이터베이스 관리 프로그램은 SALES 테이블이 저장된 각 데이터베이스 파티션에서 복제된 MQT인 R_EMPLOYEE를 사용합니다. 조인을 수행할 때 네트워크를 통해 직원 정보를 각 데이터베이스 파티션으로 이동할 필요가 없으므로 성능이 향상됩니다.

공동 배치 조인에서 복제된 구체화된 쿼리 테이블

복제된 MQT는 조인의 공동 배치에서도 도움이 됩니다. 예를 들어, 스타 스키마에 20개의 데이터베이스 파티션에 분산된 대형 사실 테이블이 있으면 사실 테이블과 차원 테이블 간 조인은 이러한 테이블이 공동 배치된 경우 가장 효율적입니다. 모든 테이블이 동일한 데이터베이스 파티션 그룹에 있는 경우 공동 배치 조인을 위해 최대 하나의 차원 테이블이 올바로 파티션됩니다. 사실 테이블의 조인 컬럼이 사실 테이블의 분배 키에 해당하지 않으므로 공동 배치 조인에서 다른 차원 테이블을 사용할 수 없습니다.

C1에서 분할된 FACT 테이블(C1, C2, C3, ...), C1에서 분할된 DIM1 테이블(C1, dim1a, dim1b, ...), C2에서 분할된 DIM2 테이블(C2, dim2a, dim2b, ...) 등을 참조하십시오. 이 경우 `dim1.c1 = fact.c1` 술어가 공동 배치되므로 FACT와 DIM1 간 조인이 가장 좋습니다. 이 두 테이블은 모두 C1 컬럼에서 분할됩니다.

그러나 FACT는 C2 컬럼이 아닌 C1 컬럼에서 분할되므로 DIM2와 `dim2.c2 = fact.c2` 술어가 관련된 조인을 공동 배치할 수 없습니다. 이 경우 사실 테이블의 데이터베이스 파티션 그룹에서 DIM2를 복제할 수 있으므로 각 데이터베이스 파티션에서 조인이 로컬로 발생합니다.

복제된 MQT를 작성할 때 소스 테이블은 데이터베이스 파티션 그룹에서 단일 파티션 테이블 또는 다중 파티션 테이블이 될 수 있습니다. 대부분의 경우 복제된 테이블은 작으며 단일 파티션 데이터베이스 파티션 그룹에 배치될 수 있습니다. 테이블에서 컬럼의 서브세트만 지정하거나 술어를 통해 규정 행 수를 제한하여 복제할 데이터를 제한할 수 있습니다.

복제된 MQT는 다중 파티션 데이터베이스 파티션 그룹에서도 작성할 수 있으므로 소스 테이블의 사본이 모든 데이터베이스 파티션에서 작성됩니다. 대형 사실 테이블과 차원 테이블 간 조인은 소스 테이블을 모든 데이터베이스 파티션으로 브로드캐스트하는 경우보다 이 환경에서 로컬로 발생할 가능성이 높습니다.

복제된 테이블의 인덱스는 자동으로 작성되지 않습니다. 소스 테이블의 인덱스와 다른 인덱스를 작성할 수 있습니다. 그러나 소스 테이블에 없는 제한조건 위반을 예방하기

위해 소스 테이블에 동일한 제한조건이 발생하는 경우에도 복제된 테이블에서 고유 인덱스를 작성하거나 제한조건을 정의할 수 없습니다.

복제된 테이블을 쿼리에 직접 참조할 수 있지만 특정 데이터베이스 파티션의 테이블 데이터를 보기 위해 DBPARTITIONNUM 스칼라 함수를 복제된 테이블과 함께 사용할 수 없습니다.

DB2 Explain 기능을 사용하여 복제된 MQT를 액세스 플랜이 쿼리에 사용했는지 여부를 판별하십시오. 옵티마이저가 선택한 액세스 플랜이 복제된 MQT를 사용하는지 여부는 조인할 데이터에 따라 다릅니다. 옵티마이저가 데이터베이스 파티션 그룹의 다른 데이터베이스 파티션으로 원래의 소스 테이블을 브로드캐스트하는 것이 저렴하다고 판별한 경우 복제된 MQT를 사용하지 않을 수도 있습니다.

파티션된 데이터베이스의 조인 전략

파티션된 데이터베이스 환경의 조인 전략은 파티션되지 않은 데이터베이스 환경의 전략과 다를 수 있습니다. 표준 조인 방법에 추가 기술을 적용하여 성능을 향상시킬 수 있습니다.

자주 조인하는 테이블의 경우 테이블 공동 배치를 고려해야 합니다. 파티션된 데이터베이스 환경에서 *테이블 공동 배치*는 동일한 수의 호환 가능한 파티션 키가 있는 두 개의 테이블이 동일한 데이터베이스 파티션 그룹에 저장된 경우 발생하는 상태를 나타냅니다. 이러한 상태가 발생하면 데이터가 저장된 데이터베이스 파티션에서 조인 처리가 수행될 수 있으며 결과 세트만 코디네이터 데이터베이스 파티션으로 이동해야 합니다.

테이블 큐

파티션된 데이터베이스 환경에서 조인 기술에 대한 설명에서는 다음의 용어를 사용합니다.

- *테이블 큐*(TQ라고도 함)는 데이터베이스 파티션 간 또는 단일 파티션 데이터베이스의 프로세서 간 행을 전송하는 메커니즘입니다.
- *방향지정 테이블 큐*(DTQ라고도 함)는 행이 수신 데이터베이스 파티션 중 하나로 해시되는 테이블 큐입니다.
- *브로드캐스트 테이블 큐*(BTQ라고도 함)는 행이 모든 수신 데이터베이스로 전송되지만 해시되지 않는 테이블 큐입니다.

테이블 큐를 사용하여 다음과 같이 테이블 데이터를 전달합니다.

- 파티션 간 병렬 처리 사용 시 한 데이터베이스 파티션에서 다른 데이터베이스 파티션으로
- 파티션 내 병렬 처리 사용 시 데이터베이스 파티션 내에서
- 단일 파티션 데이터베이스 사용 시 데이터베이스 파티션 내에서

각 테이블 큐는 한 방향으로 데이터를 전달합니다. 컴파일러는 테이블 큐가 필요한 위치를 결정하고 플랜에 포함시킵니다. 플랜이 실행되면 데이터베이스 파티션들 간 연결이 테이블 큐를 시작합니다. 프로세스가 종료되면 테이블 큐가 닫힙니다.

여러 가지 유형의 테이블 큐가 있습니다.

- 비동기 테이블 큐

  이러한 테이블 큐는 응용프로그램에서 패치 요청에 앞서서 행을 읽으므로 비동기라고 합니다. FETCH문이 발행된 경우 테이블 큐에서 행을 검색합니다.

  SELECT문에서 FOR FETCH ONLY절을 지정할 때 비동기 테이블 큐를 사용합니다. 행만 페치하는 경우 비동기 테이블 큐가 빠릅니다.
- 동기 테이블 큐

  이러한 테이블 큐는 응용프로그램에서 발행한 각 FETCH문에 대한 한 행을 읽으므로 동기라고 합니다. 각 데이터베이스 파티션에서 커서는 해당 데이터베이스 파티션에서 읽을 다음 행에 위치합니다.

  SELECT문에서 FOR FETCH ONLY절을 지정하지 않는 경우 동기 테이블 큐를 사용합니다. 파티션된 데이터베이스 환경에서 행을 갱신하는 경우 데이터베이스 관리 프로그램은 동기 테이블 큐를 사용합니다.
- 병합 테이블 큐

  이러한 테이블 큐는 순서를 보존합니다.
- 비병합 테이블 큐

  *일반 테이블 큐*라고도 하는 이러한 테이블 큐는 순서를 보존하지 않습니다.
- 리스너 테이블 큐(LTQ라고도 함)

  이러한 테이블 큐는 상관 서브쿼리에서 사용합니다. 상관값은 서브쿼리까지 전달되며 이러한 유형의 테이블 큐를 사용하여 결과는 상위 쿼리 블록까지 다시 전달됩니다.

파티션된 데이터베이스의 조인 방법

일부 조인 방법은 공동 배치 조인, 브로드캐스트 외부 테이블 조인, 방향지정 외부 테이블 조인, 방향지정 내부 테이블 및 외부 테이블 조인, 브로드캐스트 내부 테이블 조인 및 방향지정 내부 테이블 조인과 같은 파티션된 조인 환경에 사용할 수 있습니다.

다음 다이어그램에서 q1, q2 및 q3는 테이블 큐를 나타냅니다. 참조된 테이블은 두 개의 데이터베이스 파티션으로 나뉘며 화살표는 테이블 큐를 보내는 방향을 나타냅니다. 코디네이터 데이터베이스 파티션은 데이터베이스 파티션 0입니다.

공동 배치 조인

공동 배치 조인은 데이터가 상주하는 데이터베이스 파티션에서 로컬로 발생합니다. 데이터베이스 파티션은 조인이 완료된 후 다른 데이터베이스 파티션으로 데이터를 보냅니다. 옵티마이저가 공동 배치 조인을 고려할 수 있도록 조인된 테이블을 공동 배치해야 하며 해당 분산 키 쌍이 모두 동등 Join 술어에 있어야 합니다. 그림 27에서 예를 제공합니다.

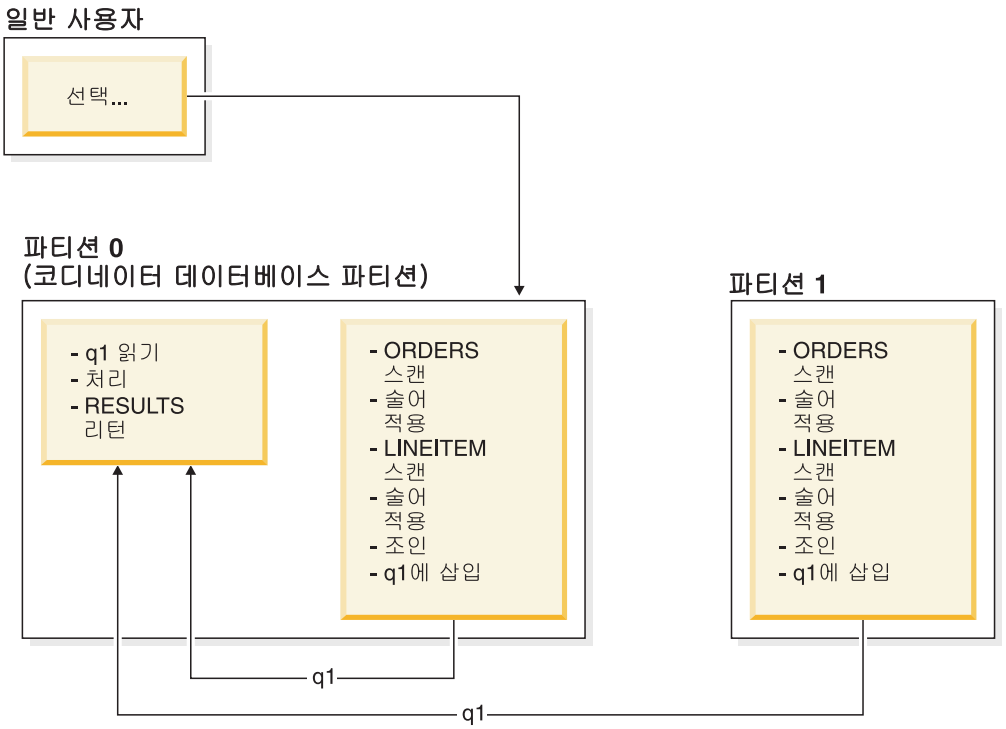


그림 27. 공동 배치 조인 예

LINEITEM 및 ORDERS 테이블은 모두 ORDERKEY 컬럼에서 파티션됩니다. 각 데이터베이스 파티션에서 로컬로 조인을 수행합니다. 이 예에서 Join 술어는 `orders.orderkey = lineitem.orderkey`로 가정합니다.

복제된 구체화된 쿼리 테이블(MQT)로 공동 배치 조인의 가능성이 증가합니다.

브로드캐스트 외부 테이블 조인

브로드캐스트 외부 테이블 조인은 조인된 테이블 간 동등 Join 술어가 없는 경우 사용할 수 있는 병렬 조인 전략을 나타냅니다. 또한 가장 비용 효율적인 조인 방법으로 입증된 다른 상황에서 사용할 수 있습니다. 예를 들어, 하나의 매우 큰 테이블이 있고 하나의 매우 작은 테이블이 있으며 Join 술어 컬럼에서 이러한 테이블이 분할되지 않은 경우 브로드캐스트 외부 테이블 조인이 발생할 수 있습니다. 두 테이블을 분할하는 대신 작은 테이블을 큰 테이블로 브로드캐스트하는 것이 저렴할 수 있습니다. 261 페이지의

지의 그림 28에서 예를 제공합니다.

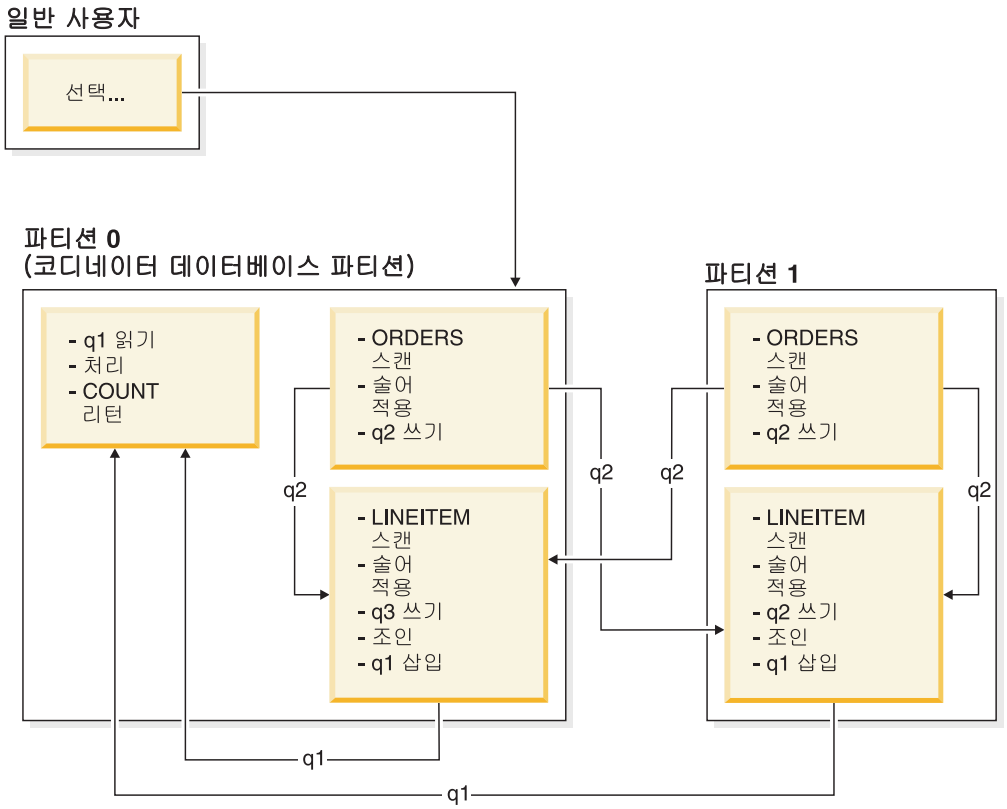
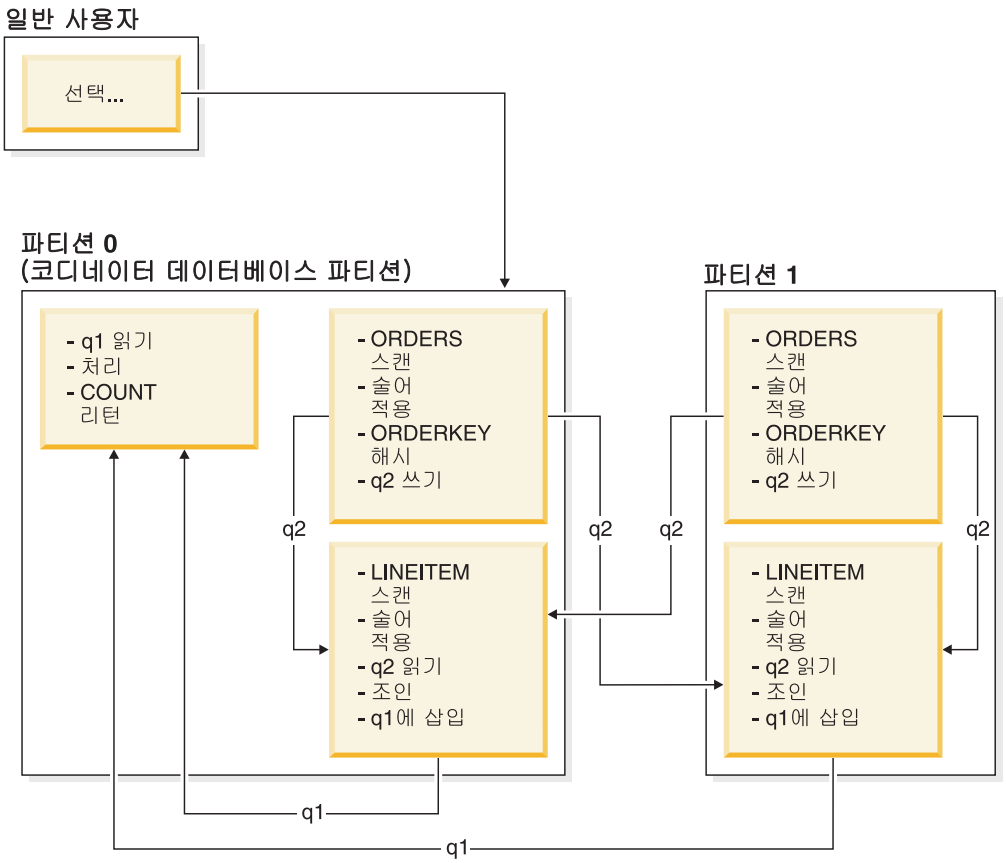


그림 28. 브로드캐스트 외부 테이블 조인 예

ORDERS 테이블을 LINEITEM 테이블이 있는 모든 데이터베이스 파티션으로 보냅니다. 테이블 큐 q2를 내부 테이블의 모든 데이터베이스 파티션으로 브로드캐스트합니다.

방향지정 외부 테이블 조인

방향지정 외부 테이블 조인 전략에서 외부 테이블의 각 행은 내부 테이블의 분할 속성에 따라 내부 테이블의 한 부분으로 보냅니다. 이 데이터베이스 파티션에서 조인이 발생합니다. 262 페이지의 그림 29에서 예를 제공합니다.



LINEITEM 테이블은 ORDERKEY 컬럼에서 파티션됩니다. ORDERS 테이블은 다른 컬럼에서 파티션됩니다. ORDERS 테이블을 해시하고 LINEITEM 테이블의 올바른 데이터베이스 파티션으로 보냅니다. 이 예에서 Join 술어는 `orders.orderkey = lineitem.orderkey`로 가정합니다.

그림 29. 방향지정 외부 테이블 조인 예

방향지정 내부 테이블 및 외부 테이블 조인

방향지정 내부 테이블 및 외부 테이블 조인 전략에서 조인 컬럼의 값에 따라 모든 외부 및 내부 테이블의 행을 데이터베이스 파티션 세트로 방향지정합니다. 이러한 데이터베이스 파티션에서 조인이 발생합니다. 263 페이지의 그림 30에서 예를 제공합니다.

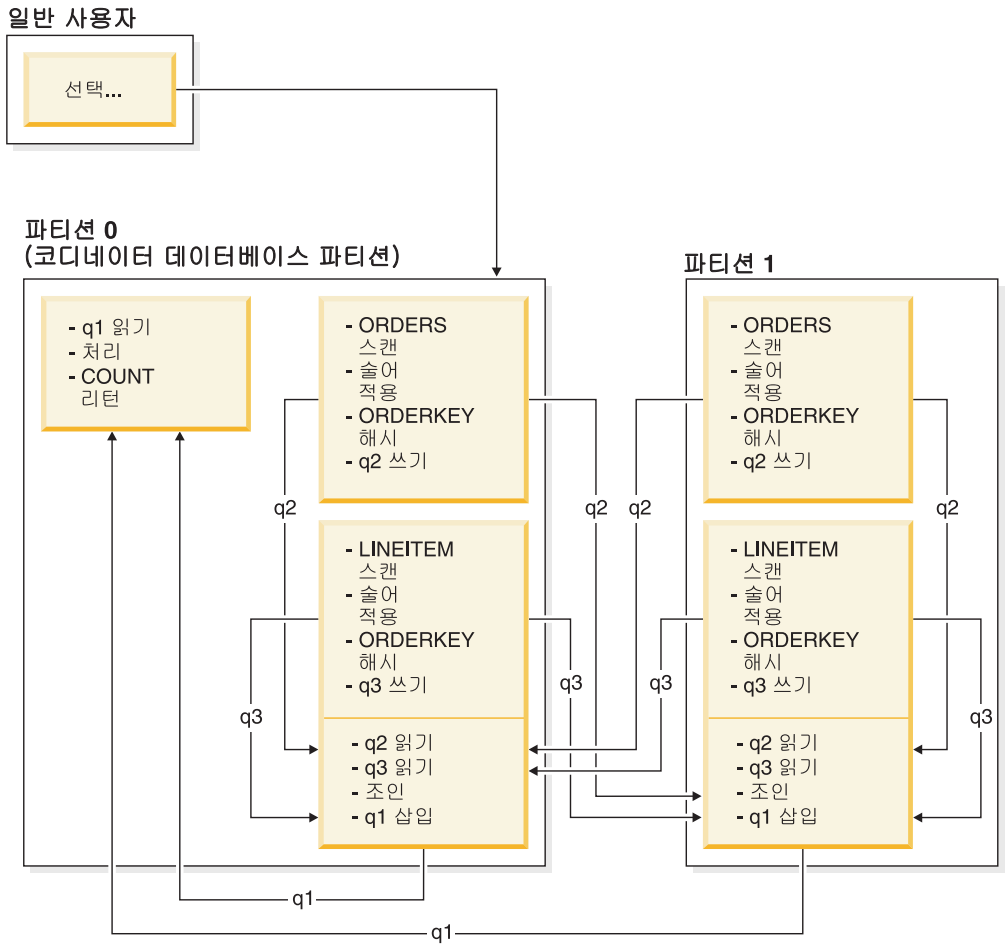
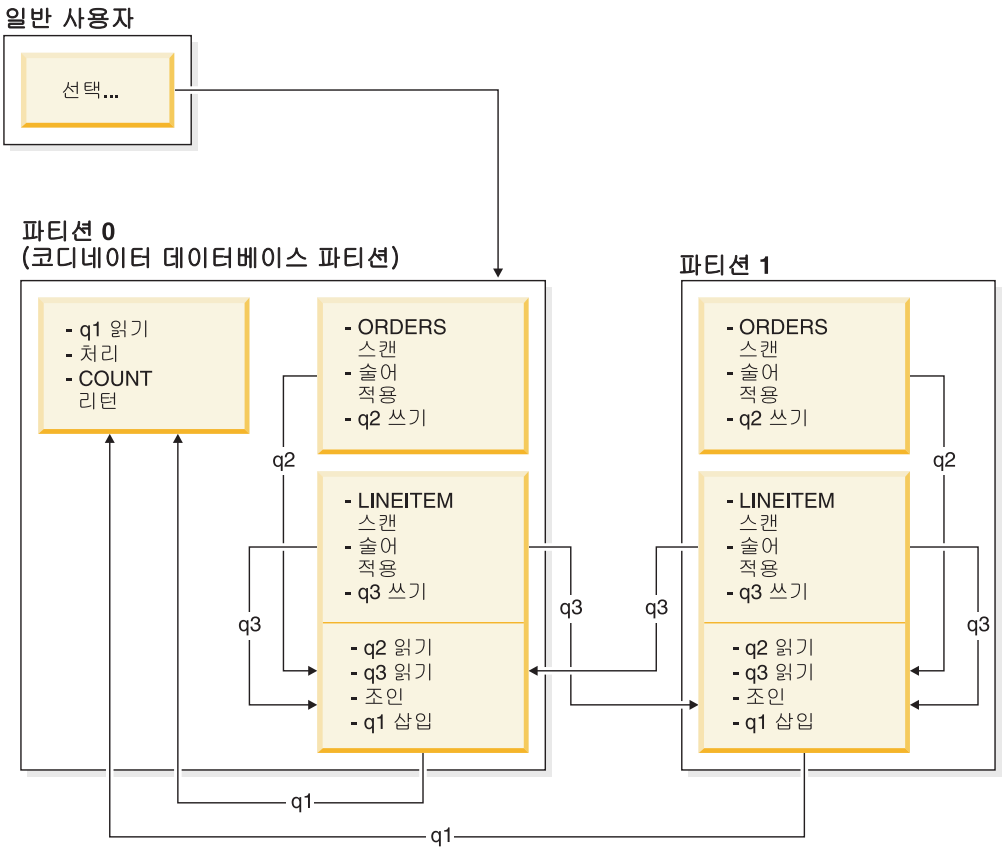


그림 30. 방향지정 내부 테이블 및 외부 테이블 조인 예

어느 테이블도 ORDERKEY 컬럼에서 파티션되지 않습니다. 두 테이블을 모두 해시하고 조인된 새 데이터베이스 파티션으로 보냅니다. 테이블 큐 q2 및 q3를 모두 방향지정합니다. 이 예에서 Join 술어는 `orders.orderkey = lineitem.orderkey`로 가정합니다.

브로드캐스트 내부 테이블 조인

브로드캐스트 내부 테이블 조인 전략에서 내부 테이블을 외부 테이블의 모든 데이터베이스 파티션으로 브로드캐스트합니다. 264 페이지의 그림 31에서 예를 제공합니다.

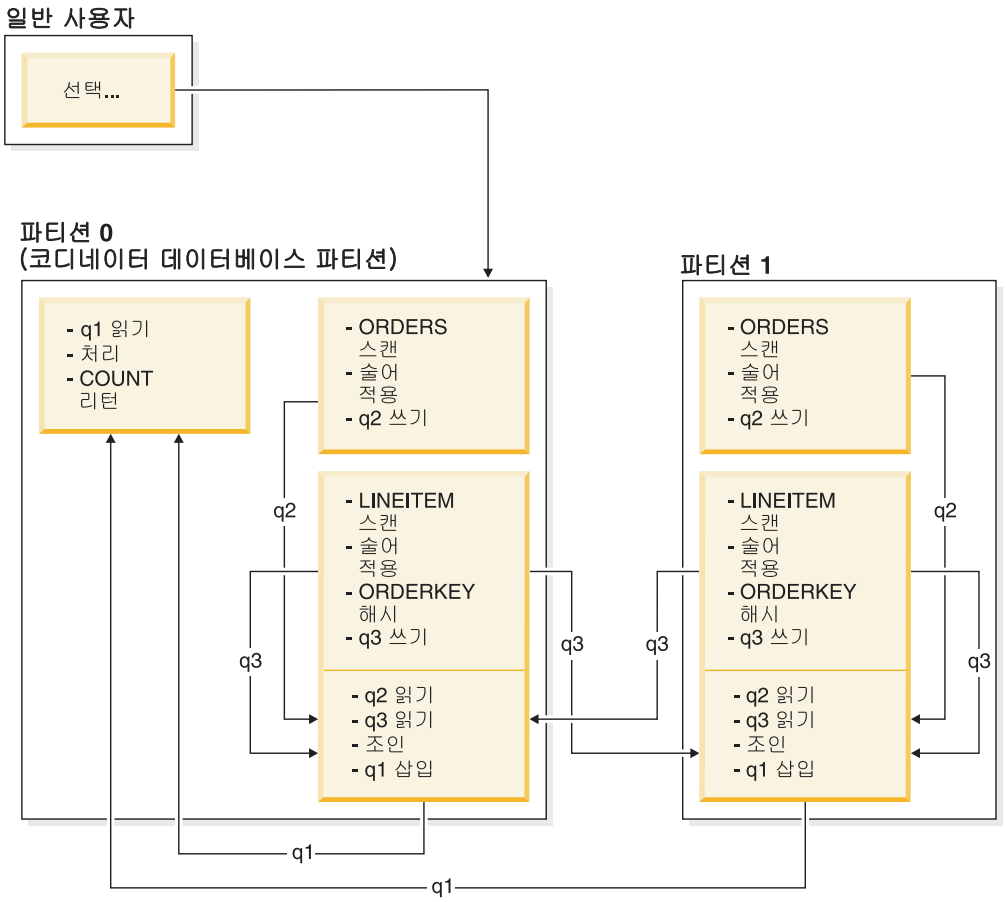


LINEITEM 테이블을 ORDERS 테이블이 있는 모든 데이터베이스 파티션으로 보냅니다. 테이블 큐 q3를 외부 테이블의 모든 데이터베이스 파티션으로 브로드캐스트합니다.

그림 31. 브로드캐스트 내부 테이블 조인 예

방향지정 내부 테이블 조인

방향지정 내부 테이블 조인 전략에서 내부 테이블의 각 행은 외부 테이블의 분할 속성에 따라 외부 테이블에 있는 하나의 데이터베이스 파티션으로 보냅니다. 이 데이터베이스 파티션에서 조인이 발생합니다. 265 페이지의 그림 32에서 예를 제공합니다.



ORDERS 테이블은 ORDERKEY 컬럼에서 파티션됩니다. LINEITEM 테이블은 다른 컬럼에서 파티션됩니다. LINEITEM 테이블을 해시하고 ORDERS 테이블의 올바른 데이터베이스 파티션으로 보냅니다. 이 예에서 Join 술어는 `orders.orderkey = lineitem.orderkey`로 가정합니다.

그림 32. 방향지정 내부 테이블 조인 예

쿼리 최적화에 대한 정렬 및 그룹화 영향

옵티마이저가 액세스 플랜을 선택할 때 데이터 정렬의 성능 영향을 고려합니다. 인덱스가 페치된 행의 요청된 순서를 충족시키지 않는 경우 정렬이 발생합니다. 옵티마이저가 정렬이 인덱스 스캔보다 비용이 저렴하다고 판별한 경우에도 정렬이 발생할 수 있습니다.

옵티마이저는 다음 중 한 방법으로 정렬된 데이터를 처리합니다.

- 쿼리가 실행될 때 정렬의 결과를 파이프합니다.
- 데이터베이스 관리 프로그램이 정렬을 내부적으로 처리할 수 있도록 합니다.

파이프 및 비파이프 정렬

단일 순차 전달에서 마지막으로 정렬된 데이터 목록을 읽을 수 있는 경우 결과는 *파이프*될 수 있습니다. 파이프는 정렬의 결과를 전달하는 비파이프 방식보다 빠릅니다. 옵티마이저는 가능할 때마다 정렬의 결과를 파이프하도록 선택합니다.

정렬의 파이프 여부에 관계없이 정렬 시간은 정렬할 행 수, 키 크기 및 행 너비와 같은 여러 가지 요인에 따라 다릅니다. 정렬할 행이 정렬 힙에서 사용 가능한 스페이스보다 많은 경우 여러 정렬 전달이 수행됩니다. 각 전달 중 전체 행 세트의 서브세트를 정렬합니다. 각 전달은 버퍼 풀의 임시 테이블에 저장됩니다. 버퍼 풀에 충분한 스페이스가 없는 경우 이 임시 테이블의 페이지를 디스크에 기록할 수 있습니다. 모든 정렬 전달이 완료되면 이와 같이 정렬된 서브세트를 정렬된 단일 행 세트로 병합합니다. 정렬이 파이프된 경우 행이 병합될 때 행을 직접 관계형 데이터 서비스(RDS)로 전달합니다(RDS는 데이터베이스의 내용에 액세스하거나 조작하는 요청을 처리하는 DB2 구성요소임).

그룹 및 정렬 푸시다운 연산자

옵티마이저가 RDS에서 DMS(Data Management Services)로 정렬 또는 집계 연산을 푸시다운하도록 선택할 수 있는 경우가 있습니다(DMS는 데이터베이스에서 테이블 및 테이블 데이터의 작성, 제거, 유지보수 및 액세스를 제어하는 DB2 구성요소임). 이러한 연산을 푸시다운하면 DMS가 정렬 또는 집계 루틴에 직접 데이터를 전달할 수 있도록 지원하므로 성능이 향상됩니다. 이 푸시다운을 사용하지 않으면 DMS는 먼저 이 데이터를 RDS로 전달한 후 정렬 또는 집계 루틴과 인터페이스합니다. 예를 들어, 다음의 쿼리는 이러한 유형의 최적화를 사용할 경우 유리합니다.

```
select workdept, avg(salary) as avg_dept_salary
  from employee
  group by workdept
```

정렬의 그룹 연산

정렬이 GROUP BY 연산에 필요한 순서를 작성하면 옵티마이저는 정렬을 수행하는 동안 일부 또는 모든 GROUP BY 집계를 수행할 수 있습니다. 이 사항은 각 그룹의 행 수가 큰 경우 유리합니다. 정렬 중 일부 그룹화를 수행하면 정렬을 디스크로 spill할 필요가 거의 없는 경우에도 훨씬 유리합니다.

정렬 중 집계하려면 올바른 결과가 리턴되도록 다음의 세 가지 집계 단계 중 하나 이상이 필요합니다.

- 첫 번째 집계 단계인 *부분 집계*는 정렬 힙이 가득 찰 때까지 집계 값을 계산합니다. 부분 집계 중 비집계 데이터를 수용하여 부분 집계를 작성합니다. 정렬 힙이 가득 차면 현재 정렬 힙에서 계산된 모든 부분 집계를 포함하여 나머지 데이터가 디스크로 spill됩니다. 정렬 힙이 재설정된 후 새 집계를 시작합니다.
- 두 번째 집계 단계인 *중간 집계*는 spill된 모든 정렬 실행을 수용하여 그룹화 키에서 추가로 집계합니다. 그룹화 키 컬럼은 분산 키 컬럼의 서브세트이므로 집계를 완료할

수 없습니다. 중간 집계는 기존의 부분 집계를 사용하여 새 부분 집계를 작성합니다. 이 단계는 항상 발생하지는 않습니다. 파티션 내 및 파티션 간 병렬 처리에서 모두 사용됩니다. 파티션 내 병렬 처리에서 전역 그룹화 키가 사용 가능한 경우 그룹화가 완료됩니다. 파티션 간 병렬 처리에서 그룹화 키가 데이터베이스 파티션들로 그룹을 나누는 분산 키의 서브세트이므로 집계를 완료하려면 재분산이 필요합니다. 집계를 완료하기 위해 단일 에이전트로 줄이기 전 각 에이전트가 spill된 정렬 실행 병합을 완료하면 파티션 내 병렬 처리에서 비슷한 경우가 존재합니다.

- 집계의 마지막 단계인 *최종 집계*는 모든 부분 집계를 사용하고 최종 집계를 작성합니다. 이 단계는 항상 GROUP BY 연산자에서 발생합니다. 정렬이 spill되지 않는다고 보장할 수 없으므로 정렬이 전체 집계를 수행할 수 없습니다. 전체 집계는 비집계 데이터에 들어 있으며 최종 집계를 작성합니다. 이 집계 방법은 보통 이미 올바른 순서의 데이터를 그룹화하는 데 사용됩니다.

최적화 전략

파티션 내 병렬 처리의 최적화 전략

SQL문이 컴파일될 때 병렬 처리 수준이 지정된 경우 옵티마이저는 단일 데이터베이스 파티션에서 쿼리를 병렬식으로 실행할 액세스 플랜을 선택할 수 있습니다.

런타임에 쿼리를 실행하기 위해 서브에이전트라고 하는 여러 데이터베이스 에이전트가 작성됩니다. 서브에이전트 수는 SQL문 컴파일 시 지정된 병렬 처리 수준 이하입니다.

액세스 플랜을 병렬 처리하기 위해 옵티마이저는 액세스 플랜을 각 서브에이전트가 실행하는 부분과 코디네이팅 에이전트가 실행하는 부분으로 나눕니다. 서브에이전트는 테이블 큐를 통해 데이터를 코디네이팅 에이전트 또는 기타 서브에이전트로 전달합니다. 파티션된 데이터베이스 환경에서 서브에이전트는 다른 데이터베이스 파티션의 서브에이전트에서 테이블 큐를 통해 데이터를 보내거나 받을 수 있습니다.

파티션 내 병렬 스캔 전략

관계형 스캔 및 인덱스 스캔은 동일한 테이블 또는 인덱스에서 병렬식으로 수행할 수 있습니다. 병렬 관계형 스캔의 경우 테이블은 서브에이전트에 지정된 페이지 또는 행 범위로 나눕니다. 서브에이전트는 지정된 범위를 스캔하고 현재 범위에서 작업을 완료하면 다른 범위가 지정됩니다.

병렬 인덱스 스캔의 경우 인덱스 키 값 및 키 값의 인덱스 항목 수에 따라 인덱스를 레코드 범위로 나눕니다. 서브에이전트에 레코드 범위를 지정하여 병렬 인덱스 스캔을 병렬 테이블 스캔과 같이 진행합니다. 서브에이전트가 현재 범위에서 작업을 완료하면 새 범위가 지정됩니다.

옵티마이저는 스캔 단위(페이지 또는 행) 및 스캔 세분화를 판별합니다.

병렬 스캔은 서브에이전트들로 균일하게 작업을 분산하는 기능을 제공합니다. 병렬 스캔의 목표는 서브에이전트들 간 로드 균형을 맞추고 모두 동일하게 사용 중인 상태를 유지하기 위한 것입니다. 사용 중인 서브에이전트 수가 사용 가능한 프로세서 수와 동일하며 디스크가 입출력 요청으로 과부하되지 않은 경우 머신 자원을 효과적으로 사용 중입니다.

다른 액세스 플랜 전략으로 쿼리 실행 시 데이터 불균형이 발생할 수 있습니다. 옵티마이저는 서브에이전트들 간 데이터 균형을 유지하는 병렬 전략을 선택합니다.

파티션 내 병렬 정렬 전략

옵티마이저는 다음의 병렬 정렬 전략 중 하나를 선택할 수 있습니다.

- 라운드 로빈 정렬

  이 정렬은 *재분산 정렬*이라고도 합니다. 이 방법은 공유 메모리를 사용하여 모든 서브에이전트에 가능하면 균일하게 데이터를 효율적으로 재분산합니다. 라운드 로빈 알고리즘을 사용하여 균일한 분산을 제공합니다. 먼저 각 서브에이전트의 개별 정렬을 작성합니다. 삽입 단계 중 라운드 로빈 방식으로 각 개별 정렬에 서브에이전트를 삽입하면 데이터가 더욱 균일하게 분산됩니다.
- 파티션된 정렬

  각 서브에이전트마다 정렬을 작성한다는 점에서 라운드 로빈 정렬과 비슷합니다. 서브에이전트는 정렬 컬럼에 해시 함수를 적용하여 행을 삽입해야 하는 정렬을 판별합니다. 예를 들어, 병합 조인의 내부 및 외부 테이블이 파티션된 정렬인 경우 서브에이전트는 병합 조인을 사용하여 해당 테이블 부분을 조인하고 병렬식으로 실행할 수 있습니다.
- 복제된 정렬

  이 정렬은 각 서브에이전트에 모든 정렬 출력이 필요한 경우 사용됩니다. 하나의 정렬이 작성되고 행이 정렬에 삽입될 때 서브에이전트가 동기화됩니다. 정렬이 완료되면 각 서브에이전트는 전체 정렬을 읽습니다. 행 수가 작으면 이 정렬을 사용하여 데이터 스트림의 재조정을 유지할 수 있습니다.
- 공유 정렬

  이 정렬은 서브에이전트가 라운드 로빈 정렬과 비슷한 방식으로 서브에이전트들에 데이터를 분산하기 위해 정렬된 결과에서 병렬 스캔을 연다는 점을 제외하고 복제된 정렬과 동일합니다.

파티션 내 병렬 임시 테이블

서브에이전트들은 협력하여 행을 동일한 테이블에 삽입함으로써 임시 테이블을 작성할 수 있습니다. 이 테이블을 *공유 임시 테이블*이라고 합니다. 서브에이전트는 데이터 스

트림을 복제하거나 분할할 것인지 여부에 따라 공유 임시 테이블에서 개인용 스캔 또는 병렬 스캔을 열 수 있습니다.

파티션 내 병렬 집계 전략

집계 연산은 서브에이전트가 병렬식으로 수행할 수 있습니다. 집계 연산의 경우 그룹화 컬럼에서 데이터를 정렬해야 합니다. 서브에이전트가 그룹화 컬럼 값 세트의 모든 행을 받을 것으로 보장할 수 있는 경우 전체 집계를 수행할 수 있습니다. 이전 파티션된 정렬로 인해 그룹화 컬럼에서 스트림이 이미 분할된 경우 이러한 상태가 발생할 수 있습니다.

그렇지 않으면 서브에이전트는 부분 집계를 수행하고 다른 전략을 사용하여 집계를 완료할 수 있습니다. 이러한 일부 전략은 다음과 같습니다.

- 병합 테이블 큐를 통해 코디네이터 에이전트로 부분적으로 집계된 데이터를 보내십시오. 코디네이터 에이전트가 집계를 완료합니다.
- 파티션된 정렬로 부분적으로 집계된 데이터를 삽입하십시오. 정렬은 그룹화 컬럼에서 분할되며 그룹화 컬럼 세트의 모든 행이 하나의 정렬 파티션에 포함됨을 확신합니다.
- 처리 균형 맞추기 위해 스트림을 복제해야 하는 경우 부분적으로 집계된 데이터를 복제된 정렬로 삽입할 수 있습니다. 각 서브에이전트는 복제된 정렬을 사용하여 집계를 완료하고 집계 결과의 동일한 사본을 수신합니다.

파티션 내 병렬 조인 전략

조인 연산은 서브에이전트가 병렬식으로 수행할 수 있습니다. 병렬 조인 전략은 데이터 스트림의 특성에 따라 판별됩니다.

조인의 내부 및 외부 테이블에서 데이터 스트림을 파티셔닝하거나 복제하여 조인을 병렬 처리할 수 있습니다. 예를 들어, 병렬 스캔을 위해 외부 스트림을 파티션하고 각 서브에이전트가 내부 스트림을 다시 독립적으로 평가하는 경우 중첩된 루프 조인이 병렬 처리될 수 있습니다. 내부 및 외부 스트림이 파티션된 정렬을 위해 가치 파티션된 경우 병합 조인을 병렬 처리할 수 있습니다.

MDC 테이블의 최적화 전략

MDC(다차원 클러스터링) 테이블을 작성하는 경우 옵티마이저는 추가 최적화 전략을 적용할 수 있으므로 여러 쿼리의 성능이 향상될 수 있습니다. 이러한 전략은 주로 블록 인덱스의 향상된 효율성에 기초하지만 여러 차원에서 클러스터링하는 장점으로 인해 데이터 검색 속도도 빨라집니다.

MDC 테이블 최적화 전략은 파티션 내 병렬 처리 및 파티션 간 병렬 처리의 성능 이점도 활용할 수 있습니다. 다음과 같은 MDC 테이블의 특정 이점을 참조하십시오.

- 차원 블록 인덱스 찾아보기는 테이블의 필수 분할 영역을 식별하고 필수 블록만 신속하게 스캔할 수 있습니다.

- 블록 인덱스는 레코드 ID(RID) 인덱스보다 작으므로 찾아보기 속도가 빠릅니다.
- 인덱스 AND 및 OR 작업을 블록 레벨에서 수행하고 RID와 결합할 수 있습니다.
- 데이터를 Extent에서 클러스터링할 수 있으므로 검색 속도가 빨라집니다.
- 롤아웃을 사용할 수 있는 경우 행을 신속하게 삭제할 수 있습니다.

REGION 및 MONTH 컬럼에 정의된 차원을 사용하여 SALES라고 하는 MDC 테이블에 대한 다음의 간단한 예를 참조하십시오.

```
select * from sales
  where month = 'March' and region = 'SE'
```

이 쿼리에서 옵티마이저는 차원 블록 인덱스 찾아보기를 수행하여 March와 SE 영역이 발생한 블록을 찾습니다. 그런 다음 이러한 블록만 스캔하여 결과 세트를 신속하게 페치합니다.

롤아웃 삭제

조건이 롤아웃을 사용하여 삭제를 허용하는 경우 이와 같이 MDC 테이블에서 행을 삭제하는 효율적인 방법이 사용됩니다. 필수 조건은 다음과 같습니다.

- DELETE문은 위치 지정된 DELETE가 아닌 검색된 DELETE입니다(명령문이 WHERE CURRENT OF절을 사용하지 않음).
- WHERE절이 없거나(모든 행을 삭제함) WHERE절의 조건만 차원이 적용됩니다.
- 테이블은 DATA CAPTURE CHANGES절을 사용하여 정의되어 있지 않습니다.
- 테이블은 참조 무결성 관계에서 상위가 아닙니다.
- 테이블에 ON DELETE 트리거가 정의되어 있지 않습니다.
- 테이블은 즉시 새로 고친 MQT에서 사용되지 않습니다.
- 외부 키가 테이블의 차원 컬럼에 대한 서브세트인 경우 연쇄 삭제 조작을 롤아웃할 수 있습니다.
- 트리거링 SQL 조작(CREATE TRIGGER문의 OLD TABLE AS절에 지정됨) 이전에 영향을 받은 행 세트를 식별하는 임시 테이블에 대해 실행되는 SELECT문에 DELETE문을 표시할 수 없습니다.

롤아웃 삭제 시, 삭제된 레코드는 로그되지 않습니다. 그 대신 레코드가 있는 페이지는 페이지의 부분들을 다시 형식화하여 비어 있게 됩니다. 다시 형식화된 부분의 변경사항은 로그되지만 레코드 자체는 로그되지 않습니다.

디폴트 동작 *즉시 정리 롤아웃*은 삭제 시 RID 인덱스를 정리합니다.
DB2_MDC_ROLLOUT 레지스트리 변수를 IMMEDIATE로 설정하거나 SET CURRENT MDC ROLLOUT MODE문에서 IMMEDIATE를 지정하여 이 모드를 지정할 수도 있습니다. 표준 삭제 조작과 비교하여 인덱스 갱신의 로깅에는 변경이 없으

므로 성능 향상은 여기에 포함된 RID 인덱스의 수에 따라 다릅니다. 총 시간 및 로그 스페이스의 퍼센트에 따라 RID 인덱스의 수가 적으면 향상 수준이 높습니다.

절약된 로그 스페이스의 양은 다음 공식을 사용하여 추정할 수 있습니다.

```
S + 38*N - 50*P
```

여기서 *N*은 삭제된 레코드의 수이고 *S*는 널(NULL) 표시기 및 VARCHAR 길이와 같은 오버헤드를 포함하여 삭제된 레코드의 전체 크기이고 *P*는 삭제된 레코드가 들어 있는 블록의 페이지 수입니다. 이 수치는 실제 로그 데이터의 감소를 나타냅니다. 롤백을 위해 예약된 스페이스가 절약되었으므로 필요한 활성 로그 스페이스에서 절약되는 양은 해당 값의 두 배입니다.

또는 *지연된 정리 롤아웃*을 사용하여 트랜잭션이 커미트된 후 RID 인덱스를 갱신할 수 있습니다. **DB2_MDC_ROLLOUT** 레지스트리 변수를 DEFER로 설정하거나 SET CURRENT MDC ROLLOUT MODE문에 DEFERRED를 지정하여 이 모드를 지정할 수도 있습니다. 지연된 롤아웃에서 RID 인덱스는 삭제가 커미트된 후 백그라운드에서 비동기로 정리됩니다. 이 롤아웃 방법으로 인해 삭제가 매우 큰 경우 또는 많은 RID 인덱스가 테이블에 존재하는 경우 삭제 시간이 상당히 빨라질 수 있습니다. 지연된 인덱스 정리 중에는 인덱스가 병렬식으로 정리되지만 즉시 인덱스 정리에서는 인덱스의 각 행이 하나씩 정리되므로 전체 정리 조작의 속도가 증가합니다. 또한 비동기 인덱스 정리가 인덱스 키 대신 인덱스 페이지의 인덱스 갱신을 로그하므로 DELETE문의 트랜잭션 로그 스페이스 요구사항이 상당히 줄어듭니다.

주: 지연된 정리 롤아웃에는 데이터베이스 힙에서 가져오는 추가 메모리 자원이 필요합니다. 데이터베이스 관리 프로그램이 필요한 메모리 구조를 할당할 수 없는 경우 지연된 정리 롤아웃이 실패하고 관리 통지 로그에 메시지가 기록됩니다.

지연된 정리 롤아웃을 사용하는 시기

삭제 성능이 가장 중요한 요인이며 테이블에 RID 인덱스가 정의된 경우 지연된 정리 롤아웃을 사용하십시오. 인덱스 정리 이전에 롤아웃된 블록의 인덱스 기반 스캔을 수행할 경우 롤아웃된 데이터의 양에 따라 다소 성능 저하가 있습니다. 즉시 인덱스 정리와 지연된 인덱스 정리 중에서 결정할 때 다음의 사항도 고려해야 합니다.

- 삭제 조작의 크기

  삭제가 매우 큰 경우 지연된 정리 롤아웃을 선택하십시오. 여러 개의 작은 MDC 테이블에서 차원 DELETE문을 자주 발행하는 경우 비동기로 인덱스 오브젝트를 정리하는 오버헤드가 삭제 조작 중 시간이 절약되는 장점보다 클 수 있습니다.

- 인덱스의 수 및 유형

  테이블에 행 레벨 처리가 필요한 여러 RID 인덱스가 있는 경우 지연된 정리 롤아웃을 사용하십시오.

- 블록 사용 가능성

  삭제 조작으로 해제된 블록 스페이스를 DELETE문이 커미트된 후 즉시 사용할 수 있도록 하려면 즉시 정리 롤아웃을 사용하십시오.
- 로그 스페이스

  로그 스페이스가 제한된 경우 삭제 규모가 크면 지연된 정리 롤아웃을 사용하십시오.
- 메모리 제한조건

  지연된 정리 롤아웃은 지연된 정리가 보류 중인 모든 테이블에서 추가 데이터베이스 힙 스페이스를 사용합니다.

삭제 중 롤아웃 동작을 사용 불가능하게 하려면 **DB2_MDC_ROLLOUT** 레지스트리 변수를 OFF로 설정하거나 SET CURRENT MDC ROLLOUT MODE문에서 NONE을 지정하십시오.

파티션된 테이블에 대한 최적화 전략

데이터 파티션 제거는 쿼리 술어를 기반으로 테이블에서 데이터 파티션의 서브세트에만 액세스하여 쿼리에 응답해야 함을 결정하는 데이터베이스 서버의 기능을 참조합니다. 데이터 파티션 제거는 특히 파티션된 테이블에 대해 결정 지원 쿼리를 실행할 때 유용합니다.

파티션된 테이블은 테이블의 하나 이상의 테이블 파티션 키 컬럼의 값에 따라 테이블 데이터가 데이터 파티션 또는 범위라고 하는 여러 스토리지 오브젝트로 나누어지는 데이터 조직 스킴을 사용합니다. 테이블의 데이터는 CREATE TABLE문의 PARTITION BY 절에서 제공된 스펙을 기반으로 여러 스토리지 오브젝트로 파티션됩니다. 이러한 스토리지 오브젝트는 서로 다른 테이블 스페이스, 동일한 테이블 스페이스 또는 둘의 조합에 있을 수 있습니다.

다음 예에서는 데이터 파티션 제거의 성능 이점을 보여줍니다.

```
create table custlist(
  subsdate date, province char(2), accountid int)
  partition by range(subsdate) (
    starting from '1/1/1990' in ts1,
    starting from '1/1/1991' in ts1,
    starting from '1/1/1992' in ts1,
    starting from '1/1/1993' in ts2,
    starting from '1/1/1994' in ts2,
    starting from '1/1/1995' in ts2,
    starting from '1/1/1996' in ts3,
    starting from '1/1/1997' in ts3,
    starting from '1/1/1998' in ts3,
    starting from '1/1/1999' in ts4,
    starting from '1/1/2000' in ts4,
    starting from '1/1/2001'
    ending '12/31/2001' in ts4)
```

2000년의 고객 정보에만 관심이 있다고 가정해봅니다.

```
select * from custlist
  where subsdate between '1/1/2000' and '12/31/2000'
```

그림 33에 표시된 대로, 데이터베이스 서버가 테이블 스페이스 TS4에서 하나의 데이터 파티션에만 액세스하여 이 쿼리를 해석해야 함을 결정합니다.

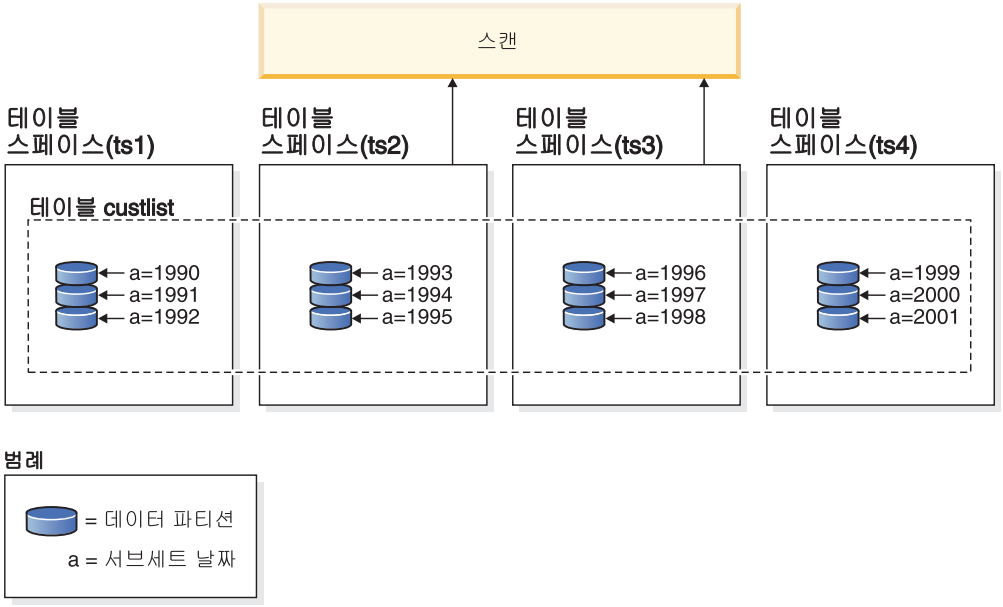


그림 33. 데이터 파티션 제거의 성능 이점

데이터 파티션 제거의 또 다른 예는 다음 스킴을 기반으로 합니다.

```
create table multi (
  sale_date date, region char(2))
  partition by (sale_date) (
    starting '01/01/2005'
    ending '12/31/2005'
    every 1 month)

create index sx on multi(sale_date)

create index rx on multi(region)
```

다음 쿼리를 발행한다고 가정해봅니다.

```
select * from multi
  where sale_date between '6/1/2005'
    and '7/31/2005' and region = 'NW'
```

테이블 파티셔닝 없이, 하나의 가능한 플랜은 인덱스 ANDing입니다. Index ANDing은 다음 태스크를 수행합니다.

- 각 인덱스에서 모든 관련 인덱스 항목을 읽습니다.
- 행 ID(RID) 두 세트를 모두 저장합니다.

- RID를 일치하여 두 인덱스에서 발생하는 항목을 판별합니다.
- RID를 사용하여 행을 페치합니다.

그림 34에서 설명된 대로, 테이블 파티셔닝을 통해 인덱스를 읽어 REGION 및 SALE_DATE 모두에 대한 일치 항목을 찾음으로써 일치하는 행을 신속하게 검색할 수 있습니다.

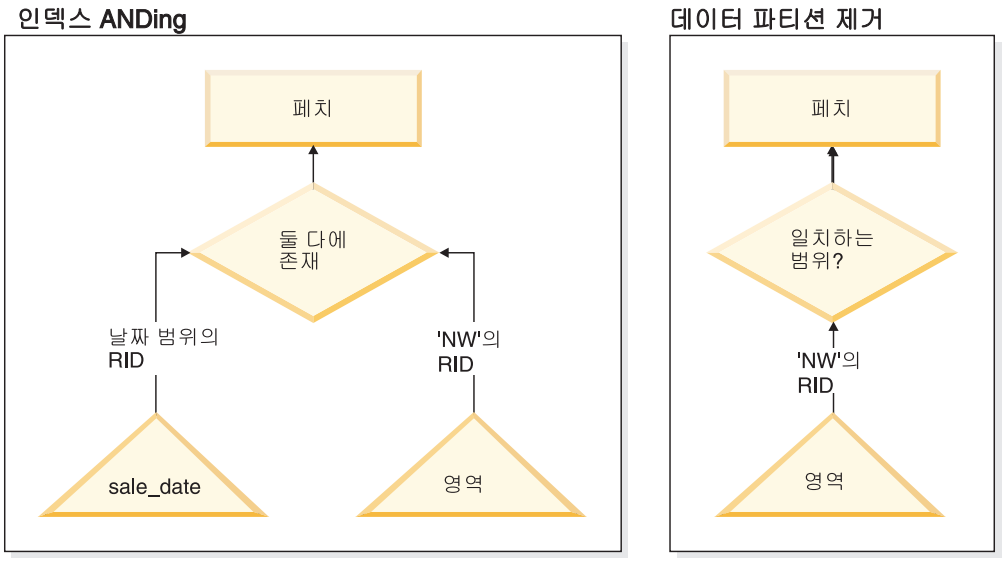


그림 34. 테이블 파티셔닝 및 인덱스 ANDing에 대한 옵티마이저 의사 결정 경로

DB2 Explain

Explain 기능을 사용하여 쿼리 옵티마이저에 의해 선택된 데이터 파티션 제거 플랜을 결정할 수도 있습니다. 『DP Elim Predicates』 정보에서는 다음 쿼리를 해석하기 위해 스캔되는 데이터 파티션을 보여줍니다.

```
   select * from custlist
     where subsdate between '12/31/1999' and '1/1/2001'

Arguments:
---------
DPESTFLG: (Number of data partitions accessed are Estimated)
    FALSE
DPLSTPRT: (List of data partitions accessed)
    9-11
DPNUMPRT: (Number of data partitions accessed)
   3

DP Elim Predicates:
------------------
Range 1)
  Stop  Predicate: (Q1.A <= '01/01/2001')
 Start Predicate: ('12/31/1999' <= Q1.A)
```

```
Objects Used in Access Plan:
 --------------------------

  Schema: MRSRINI
  Name:    CUSTLIST
  Type:    Data Partitioned Table
   Time of creation:      2005-11-30-14.21.33.857039
   Last statistics update:    2005-11-30-14.21.34.339392
    Number of columns:      3
    Number of rows:      100000
   Width of rows:      19
   Number of buffer pool pages:    1200
    Number of data partitions:    12
    Distinct row values:      No
    Tablespace name:      <VARIOUS>
```

복수 컬럼 지원

데이터 파티션 제거는 복수 컬럼이 테이블 파티션 키로 사용될 경우 작동합니다. 예를 들어, 다음과 같습니다.

```
create table sales (
  year int, month int)
 partition by range(year, month) (
    starting from (2001,1)
    ending at (2001,3) in ts1,
   ending at (2001,6) in ts2,
   ending at (2001,9) in ts3,
   ending at (2001,12) in ts4,
   ending at (2002,3) in ts5,
   ending at (2002,6) in ts6,
   ending at (2002,9) in ts7,
   ending at (2002,12) in ts8)

select * from sales where year = 2001 and month < 8
```

쿼리 옵티마이저에서 TS1, TS2 및 TS3의 데이터 파티션에만 액세스하여 이 쿼리를 해석해야 한다고 추론합니다.

주: 복수 컬럼이 테이블 파티션 키를 구성하는 경우, 테이블 파티션 키에 사용되는 비선행 컬럼이 독립적이지 않기 때문에 복합 키의 선행 컬럼에 술어가 있는 경우에만 데이터 파티션 제거가 가능합니다.

다중 범위 지원

다중 범위가 있는 데이터 파티션(즉, 함께 OR되는 데이터 파티션)에서 데이터 파티션 제거가 가능합니다. 위의 예에서 작성된 SALES 테이블을 사용하여 다음 쿼리를 실행하십시오.

```
select * from sales
  where (year = 2001 and month <= 3)
    or (year = 2002 and month >= 10)
```

데이터베이스 서버는 2001년의 첫 번째 분기 및 2002년의 마지막 분기의 데이터에만 액세스합니다.

생성된 컬럼

생성된 컬럼을 테이블 파티션 키로 사용할 수 있습니다. 예를 들어, 다음과 같습니다.

```
create table sales (
  a int, b int generated always as (a / 5))
  in ts1,ts2,ts3,ts4,ts5,ts6,ts7,ts8,ts9,ts10
  partition by range(b) (
    starting from (0)
    ending at (1000) every (50))
```

이 경우, 생성된 컬럼의 술어가 데이터 파티션 제거에 사용됩니다. 또한 컬럼 생성에 사용된 표현식이 단순할 경우, 데이터베이스 서버에서 소스 컬럼의 술어를 생성된 컬럼의 술어로 변환하여 생성된 컬럼에서 데이터 파티션 제거를 가능하게 합니다. 예를 들어, 다음과 같습니다.

```
select * from sales where a > 35
```

데이터베이스 서버가 a(a > 35)에서 b(b > 7)에 추가 술어를 생성하여 데이터 파티션 제거를 허용합니다.

Join 술어

Join 술어가 테이블 액세스 등급으로 밀려 내려가는 경우 데이터 파티션 제거에서 Join 술어가 사용될 수도 있습니다. Join 술어는 중첩된 루프 조인(NLJN)의 내부 조인에서 테이블 액세스 등급으로만 밀려 내려갑니다.

다음 테이블을 고려해보겠습니다.

```
create table t1 (a int, b int)
  partition by range(a,b) (
    starting from (1,1)
    ending (1,10) in ts1,
    ending (1,20) in ts2,
    ending (2,10) in ts3,
    ending (2,20) in ts4,
    ending (3,10) in ts5,
    ending (3,20) in ts6,
    ending (4,10) in ts7,
    ending (4,20) in ts8)

create table t2 (a int, b int)
```

다음 두 개의 술어가 사용됩니다.

```
P1: T1.A = T2.A
P2: T1.B > 15
```

이 예에서는 조인의 외부 값을 알 수 없기 때문에 컴파일 시간에 액세스되는 정확한 데이터 파티션을 판별할 수 없습니다. 이 경우 및 호스트 변수 또는 매개변수 표시문자가 사용되는 경우, 필요한 값이 바인드될 때 런타임에서 데이터 파티션 제거가 발생합니다.

런타임 동안 T1이 NLJN의 내부인 경우, 술어를 기반으로 T2.A의 모든 외부 값에 대해 데이터 파티션 제거가 동적으로 발생합니다. 런타임 동안, 액세스될 테이블 스페이스 TS6에 데이터 파티션을 규정하는 외부 값 T2.A = 3에 대해 술어 T1.A = 3 및 T1.B > 15가 적용됩니다.

테이블 T1 및 T2의 컬럼 A에 다음 값이 있다고 가정해봅니다.

외부 테이블 **T2:** 컬럼 **A**	내부 테이블 **T1:** 컬럼 **A**	내부 테이블 **T1:** 컬럼 **B**	내부 테이블 **T1:** 데이터 파티션 위치
2	3	20	TS6
3	2	10	TS3
3	2	18	TS4
	3	15	TS6
	1	40	TS3

중첩된 루프 조인을 수행하기 위해 (내부 테이블에 대한 테이블 스캔 가정), 데이터베이스 관리 프로그램은 다음 단계를 수행합니다.

1. T2에서 첫 번째 행을 읽습니다. A의 값은 2입니다.
2. T2.A 값(2)을 Join 술어 T1.A = T2.A의 컬럼 T2.A에 바인드합니다. 술어가 T1.A = 2가 됩니다.
3. 술어 T1.A = 2 및 T1.B > 15를 사용하여 데이터 파티션 제거를 적용합니다. 이렇게 하면 테이블 스페이스 TS4에 데이터 파티션이 규정됩니다.
4. T1.A = 2 및 T1.B > 15를 적용한 후, 행이 발견될 때까지 테이블 T1의 테이블 스페이스 TS4에서 데이터 파티션을 스캔합니다. 발견된 첫 번째 규정 행은 T1의 행 3입니다.
5. 일치하는 행을 조인합니다.
6. 다음 일치(T1.A = 2 및 T1.B > 15)가 발견될 때까지 테이블 T1의 테이블 스페이스 TS4에서 데이터 파티션을 스캔합니다. 행이 더 이상 발견되지 않습니다.
7. T2의 모든 행이 처리될 때까지 다음 행 T2에 대해 1-6단계를 반복합니다(A 값을 3으로 교체).

XML 데이터에 대한 인덱스

파티션되지 않은 XML 인덱스는 파티션된 테이블의 기타 관계형 인덱스가 유지보수되는 것과 동일한 방법으로 테이블 삽입, 갱신 및 삭제 조작 동안 데이터베이스 관리 프로그램에 의해 유지보수됩니다. 파티션된 테이블에서 XML 데이터에 대한 파티션되지 않은 인덱스는 파티션되지 않은 테이블에서 XML 데이터에 대한 인덱스와 동일한 방

법으로 사용되어 쿼리 처리 속도를 높입니다. 쿼리 술어를 사용하여 파티션된 테이블에서 데이터 파티션의 서브세트에만 액세스하여 쿼리에 응답해야 함을 결정할 수 있습니다.

XML 컬럼에 대한 인덱스와 데이터 파티션 제거는 함께 작동하여 쿼리 성능을 향상시킬 수 있습니다. 다음 파티션된 테이블을 고려해보십시오.

```
create table employee (a int, b xml, c xml)
  index in tbspx
  partition by (a) (
    starting 0 ending 10,
    ending 20,
    ending 30,
    ending 40)
```

Now consider the following query:

```
select * from employee
  where a > 21
  and xmlexist('$doc/Person/Name/First[.="Eric"]'
    passing "EMPLOYEE"."B" as "doc")
```

옵티마이저는 술어 a > 21을 기반으로 처음 두 파티션을 즉시 제거할 수 있습니다. 쿼리 플랜에서 옵티마이저에 의해 컬럼 B의 XML 데이터에 대한 파티션되지 않은 인덱스가 선택된 경우, XML 데이터에 대한 인덱스를 사용한 인덱스 스캔에서 옵티마이저의 데이터 파티션 제거 결과를 이용할 수 있고 관계형 데이터 파티션 제거 술어에 의해 제거되지 않은 파티션에 속하는 결과만 리턴할 수 있습니다.

구체화된 쿼리 테이블을 사용한 쿼리 최적화 향상

구체화된 쿼리 테이블(MQT)은 복잡한 쿼리의 응답 시간을 향상시키는 강력한 방법입니다.

다음과 같은 연산 중 하나 이상이 필요할 수도 있는 쿼리의 경우에 특히 해당되는 사항입니다.

- 하나 이상의 차원에 대한 집계 데이터
- 테이블 그룹에 대한 조인 및 집계 데이터
- 자주 액세스하는 데이터 서브세트 즉, 『자주 사용하는』 수평 또는 수직 데이터베이스 파티션의 데이터
- 파티션된 데이터베이스 환경에서 테이블 또는 테이블 일부분의 재파티션된 데이터

MQT에 대한 지식은 SQL 및 XQuery 컴파일러로 통합되었습니다. 컴파일러에서 쿼리는 단계를 재작성하고 옵티마이저는 쿼리를 MQT와 대조하여 기본 테이블에 액세스하는 쿼리에서 MQT 대체 여부를 판별합니다. MQT를 사용하는 경우 Explain 기능은 선택된 MQT에 대한 정보를 제공할 수 있습니다. 이 경우 사용자는 리라우트된 MQT가 아닌 기본 테이블에 대한 액세스 특권이 필요합니다.

MQT가 여러 면에서 일반 테이블과 비슷하게 작동하므로 테이블 스페이스 정의 및 인덱스를 사용하고 runstats 유틸리티를 호출하여 데이터 액세스를 최적화하는 동일한 지침이 MQT에 적용됩니다.

MQT 기능에 대한 이해를 돕기 위해 다음 예에서는 다차원 분석 쿼리가 MQT를 활용하는 방법을 보여줍니다. 컴퓨터 세트 및 신용 카드 계좌 세트가 있는 데이터베이스 웨어하우스를 생각해 보십시오. 웨어하우스는 신용 카드로 발생한 거래 세트를 기록합니다. 각 거래에는 함께 구입한 물품 세트가 있습니다. 이 스키마는 거래 물품이 포함된 하나의 대형 테이블과 구매 거래를 식별하는 다른 하나의 대형 테이블로 두 개의 대형 테이블이 있으므로 다중 스타 스키마로 분류됩니다.

세 개의 계층 차원인 제품, 위치 및 시간이 거래에 대해 설명합니다. 제품 계층 구조는 제품 그룹 및 제품 라인을 표시하는 두 개의 정규 테이블에 저장됩니다. 계층 구조의 위치는 시, 도 및 국가, 영역 또는 지역 정보를 포함하고 있으며 단일 비정규 테이블에 저장됩니다. 시간 계층 구조는 일, 월 및 년 정보를 포함하고 있으며 단일 날짜 필드에 인코드되어 있습니다. 날짜 차원은 내장 함수를 사용하여 거래의 날짜 필드에서 추출합니다. 이 스키마의 다른 테이블은 고객에 대한 계좌 정보와 고객 정보를 표시합니다.

다음의 계층 구조 각 레벨에서 판매에 대한 MQT가 작성됩니다.

- 제품
- 위치
- 년, 월 및 일로 구성된 시간

여러 쿼리는 이 저장된 집계 데이터로 충족될 수 있습니다. 다음 예에서는 제품 그룹 및 라인 차원을 따라, 시, 도 및 국가, 영역 또는 지역 차원을 따라, 시간 차원을 따라 판매 데이터 합계 및 수를 계산하는 MQT를 작성하는 방법을 보여줍니다. 또한 GROUP BY절에 있는 여러 개의 다른 컬럼이 들어 있습니다.

```
create table dba.pg_salessum
  as (
    select l.id as prodline, pg.id as pgroup,
      loc.country, loc.state, loc.city,
      l.name as linename, pg.name as pgname,
      year(pdate) as year, month(pdate) as month,
      t.status,
      sum(ti.amount) as amount,
      count(*) as count
    from cube.transitem as ti, cube.trans as t,
      cube.loc as loc, cube.pgroup as pg, cube.prodline as l
    where
      ti.transid = t.id and
      ti.pgid = pg.id and
      pg.lineid = l.id and
      t.locid = loc.id and
      year(pdate) > 1990
    group by l.id, pg.id, loc.country, loc.state, loc.city,
      year(pdate), month(pdate), t.status, l.name, pg.name
```

```
  )
data initially deferred refresh deferred;

refresh table dba.pg_salessum;
```

이렇게 사전 계산된 합계를 활용할 수 있는 쿼리는 다음과 같습니다.

- 월 및 제품 그룹에 따른 판매액
- 1990년 이후의 전체 판매액
- 1995년 또는 1996년의 판매액
- 특정 제품 그룹 또는 제품 라인의 판매액 합계
- 1995년 및 1996년 특정 제품 그룹 또는 제품 라인의 판매액 합계
- 특정 국가, 영역 또는 지역의 판매액 합계

이러한 쿼리에 대한 정확한 응답이 MQT에 포함되어 있지 않지만 응답의 일부분이 이미 계산되었으므로 MQT에 의한 응답 계산 비용은 대형 기본 테이블 사용으로 인한 비용보다 상당히 작을 수 있습니다. MQT는 비용이 많이 드는 기본 데이터의 조인, 정렬 및 집계 필요성을 줄일 수 있습니다.

다음 샘플 쿼리는 MQT 예에서 이미 계산된 결과를 사용할 수 있으므로 성능이 상당히 향상됩니다.

첫 번째 쿼리는 1995년 및 1996년의 총 판매액을 리턴합니다.

```
set current refresh age=any

select year(pdate) as year, sum(ti.amount) as amount
  from cube.transitem as ti, cube.trans as t,
    cube.loc as loc, cube.pgroup as pg, cube.prodline as l
  where
    ti.transid = t.id and
    ti.pgid = pg.id and
    pg.lineid = l.id and
    t.locid = loc.id and
    year(pdate) in (1995, 1996)
  group by year(pdate);
```

두 번째 쿼리는 1995년 및 1996년에 제품 그룹별 전체 판매액을 리턴합니다.

```
set current refresh age=any

select pg.id as "PRODUCT GROUP", sum(ti.amount) as amount
  from cube.transitem as ti, cube.trans as t,
    cube.loc as loc, cube.pgroup as pg, cube.prodline as l
  where
    ti.transid = t.id and
    ti.pgid = pg.id and
    pg.lineid = l.id and
    t.locid = loc.id and
    year(pdate) in (1995, 1996)
  group by pg.id;
```

기본 테이블이 크면 MQT 사용 시 응답 시간이 잠재적으로 향상될 가능성이 커집니다. MQT는 쿼리들 간 작업 중복을 효과적으로 제거할 수 있습니다. 계산은 MQT 빌드 시 한 번만 수행되며 MQT를 새로 고칠 때마다 한 번씩 수행되고 여러 쿼리를 실행하는 동안 내용을 재사용할 수 있습니다.

Explain 기능

DB2 Explain 기능은 옵티마이저가 SQL 또는 XQuery문에서 선택하는 액세스 플랜에 대한 자세한 정보를 제공합니다.

이 정보는 액세스 플랜을 선택하는 데 사용되는 결정 기준에 대해 설명하며 성능 향상을 위해 명령문 또는 인스턴스 구성을 조정하는 데 유용할 수 있습니다. 특히 Explain 정보는 다음과 같은 경우에 유용합니다.

- 사용자의 쿼리를 충족시키기 위해 데이터베이스 관리 프로그램이 테이블 및 인덱스에 액세스하는 방법을 알 수 있습니다.
- 성능 조정 조치를 평가할 수 있습니다. 명령문을 변경하거나 구성을 변경한 후 새 Explain 정보를 조사하여 조치의 성능 영향을 판별할 수 있습니다.

캡처되는 정보는 다음과 같습니다.

- 쿼리를 처리하는 데 사용된 조작 시퀀스
- 비용 정보
- 술어 및 각 술어의 선택 빈도 추정
- Explain 정보 캡처 시 SQL 또는 XQuery문에 참조된 모든 오브젝트의 통계
- SQL 또는 XQuery문을 다시 최적화하는 데 사용된 호스트 변수, 매개변수 표시문자 또는 특수 레지스터의 값

특정 설명 가능한 명령문에서 선택한 액세스 플랜에 관한 정보를 캡처하고 이 정보를 Explain 테이블에 기록하는 EXPLAIN문을 발행하여 Explain 기능을 호출합니다. EXPLAIN문을 발행하기 전에 Explain 테이블을 작성해야 합니다. Explain 기능의 동작을 제어하는 CURRENT EXPLAIN MODE 또는 CURRENT EXPLAIN SNAPSHOT 특수 레지스터를 설정할 수도 있습니다.

Explain 유틸리티를 사용하는 데 필요한 특권 및 권한은 EXPLAIN문에 대한 설명을 참조하십시오. EXPLAIN 권한은 Explain 정보에 액세스해야 하지만 데이터베이스에 저장된 데이터에는 액세스할 필요가 없는 개인에게 부여할 수 있습니다. 이 권한은 데이터베이스 관리자 권한의 서브세트이며 테이블에 저장된 데이터에 액세스할 수 있는 고유 권한이 없습니다.

Explain 정보를 표시하려면 명령행 도구 또는 Visual Explain을 사용할 수 있습니다. 사용하는 도구는 Explain 기능의 동작을 제어하는 특수 레지스터를 설정하는 방법을 판별합니다. 예를 들어, Visual Explain만 사용할 경우 스냅샷 캡처 정보만 필요합니다.

Explain 테이블에 대해 명령행 유틸리티 중 하나를 사용하거나 사용자 정의 SQL 또는 XQuery문을 사용하여 자세한 분석을 수행할 경우 모든 Explain 정보를 캡처해야 합니다.

Explain 기능을 사용하여 SQL문 조정

Explain 기능은 SQL문을 실행하기 위해 쿼리 옵티마이저가 선택한 쿼리 액세스 플랜을 표시하는 데 사용됩니다.

여기에는 플랜 연산자, 해당 인수, 실행 순서 및 비용과 같은, SQL문을 실행하는 데 사용되는 관계형 연산에 대한 광범위한 세부사항이 포함됩니다. 쿼리 액세스 플랜은 쿼리 성능에서 가장 중요한 인수 중 하나이므로, 쿼리 성능 문제를 진단할 때 Explain 기능 출력을 이해하는 것이 중요합니다.

Explain 정보는 일반적으로 다음을 수행하는 데 사용됩니다.

- 응용프로그램 성능이 변경된 이유 이해
- 성능 조정 노력 평가

성능 변경 분석

쿼리 성능의 변경에 대한 이유를 쉽게 이해하려면, 다음 단계를 사용하여 『전과 후』 Explain 정보를 얻으십시오.

1. 변경을 수행하기 전 쿼리에 대한 Explain 정보를 캡처하고 결과 Explain 테이블을 저장하십시오. 또는, db2exfmt 유틸리티의 출력을 저장할 수 있습니다. 그러나 Explain 테이블에 Explain 정보를 포함하면, SQL을 사용하여 쿼리하기가 쉬워지고 보다 복잡한 분석이 용이하게 됩니다. 또한 관계형 DBMS에 데이터를 포함하는 데서 얻을 수 있는 모든 분명한 유지보수 이점이 제공됩니다. db2exfmt 도구는 언제든지 실행할 수 있습니다.
2. 이 정보를 보기 위해 Visual Explain에 액세스할 수 없는 경우 현재 카탈로그 통계를 저장하거나 인쇄하십시오. db2look 명령을 사용하여 이 태스크 수행을 도울 수도 있습니다. DB2 버전 9.7에서는 Explain 테이블이 채워질 때 Explain 스냅샷을 수집할 수 있습니다. Explain 스냅샷에는 명령문이 설명될 당시의 모든 관련 통계가 포함됩니다. db2exfmt 유틸리티는 스냅샷에 포함된 통계를 자동으로 형식화합니다. 이는 쿼리 최적화에 사용된 통계가 아직 시스템 카탈로그 테이블에 없거나 명령문이 설명된 시간과 명령문이 시스템 카탈로그에서 검색된 시간 사이에 변경되었을 수도 있기 때문에, 자동 또는 실시간 통계 수집을 사용할 경우에 특히 중요합니다.
3. CREATE TABLE, CREATE VIEW, CREATE INDEX 및 CREATE TABLESPACE에 대한 명령문을 포함하여 DDL(Data Definition Language) 명령문을 저장하거나 인쇄하십시오. db2look 명령은 이 태스크 또한 수행합니다.

이러한 방식으로 수집하는 정보는 향후 분석에서 참조할 수 있습니다. 동적 SQL문의 경우, 응용프로그램을 처음으로 실행할 때 이 정보를 수집할 수 있습니다. 정적 SQL문의 경우, 바인드 시간에 이 정보를 수집할 수도 있습니다. 특히 DB2 릴리스 또는 새 서비스 레벨의 설치와 같은 주요 시스템 변경 전, 또는 데이터 재분배 및 데이터베이스 파티션 추가 또는 삭제와 같은 중요한 구성 변경 전에 이 정보를 수집하는 것이 중요합니다. 이는 이러한 유형의 시스템 변경이 액세스 플랜을 반대로 변경시킬 수도 있기 때문입니다. 액세스 플랜 회귀는 드물게 발생하지만, 이 정보를 사용 가능하게 하면 성능 회귀를 더 빨리 해결하는 데 도움이 됩니다. 성능 변경을 분석하려면 분석을 시작할 때 수집한 환경 및 쿼리에 대한 정보와 이전에 수집한 정보를 비교하십시오.

간단한 예로, 분석에서 인덱스가 액세스 플랜의 일부로 더 이상 사용되지 않음이 표시될 수 있습니다. Visual Explain 또는 db2exfmt에 의해 표시된 카탈로그 통계 정보를 사용하여, 인덱스 레벨의 수 (NLEVELS 컬럼)가 현재 쿼리가 데이터베이스에 처음 바운드되었을 때보다 상당히 더 높아진 것을 확인할 수도 있습니다. 다음 조치 중 하나를 선택하여 수행하십시오.

- 인덱스 재구성
- 테이블 및 인덱스의 새 통계 수집
- 쿼리 리바인드 시 Explain 정보 수집

이러한 조치 중 하나를 수행한 후에 액세스 플랜을 다시 검사하십시오. 인덱스가 다시 한 번 사용될 경우, 쿼리 성능이 더 이상 문제가 되지 않을 수도 있습니다. 인덱스가 여전히 사용되지 않을 경우, 또는 성능이 여전히 문제가 되는 경우, 두 번째 조치를 시도하고 결과를 검사하십시오. 문제가 해결될 때까지 이러한 단계를 반복하십시오.

성능 조정 노력 평가

구성 매개변수 조정, 컨테이너 추가, 또는 새로운 카탈로그 통계 수집 등, 쿼리 성능을 향상시키는 데 도움이 되는 다양한 조치를 취할 수 있습니다.

이러한 영역 중 하나에서 변경을 수행한 후, Explain 기능을 사용하여 선택한 액세스 플랜에 변경이 미친 영향을 판별할 수 있습니다(있는 경우). 예를 들어, 인덱스 지침을 기반으로 구체화된 쿼리 테이블(MQT) 또는 인덱스를 추가할 경우, 인덱스 또는 구체화된 쿼리 테이블이 실제로 예상한 대로 사용되는지 여부를 판별하는 데 Explain 데이터가 도움이 될 수 있습니다.

Explain 출력에서 선택된 액세스 플랜 및 상대적 비용을 판별할 수 있는 정보를 제공하지만, 쿼리에 대한 성능 향상을 정확하게 측정하는 유일한 방법은 벤치마크 테스트 기술을 사용하는 것입니다.

Explain 정보 캡처 지침

SQL 또는 XQuery문이 컴파일될 때 요청에 따라 Explain 데이터를 캡처할 수 있습니다.

런타임에 증분식 바인드 SQL 또는 XQuery문이 컴파일되는 경우 데이터는 바인드 시가 아닌 런타임에 Explain 테이블에 배치됩니다. 이러한 명령문의 경우 삽입된 Explain 테이블 규정자 및 권한 부여 ID는 패키지를 실행 중인 사용자가 아닌 패키지 소유자의 것입니다.

Explain 정보는 SQL 또는 XQuery문이 컴파일될 때에만 캡처됩니다. 초기 컴파일 이후 환경 변경 시 필요하거나 Explain 기능이 활성화되면 동적 쿼리 명령문이 재컴파일됩니다. 동일한 쿼리 명령문의 동일한 PREPARE문을 발행하는 경우 이 명령문이 준비 또는 실행될 때마다 쿼리가 컴파일되고 Explain 데이터가 캡처됩니다.

REOPT ONCE 또는 ALWAYS 바인드 옵션을 사용하여 패키지를 바인드하는 경우 호스트 변수, 매개변수 표시문자, 전역 변수 또는 특수 레지스터를 포함한 SQL 또는 XQuery문이 컴파일되고 이러한 변수의 실제 값(알려진 경우) 또는 디폴트 추정(컴파일 시 값이 알려지지 않은 경우)을 사용하여 액세스 경로를 작성합니다.

REOPT ONCE 옵션이 사용된 경우 지정된 SQL 또는 XQuery문을 패키지 캐시의 동일한 명령문과 일치시키려고 시도합니다. 이미 다시 최적화되고 캐시된 쿼리 명령문의 값을 사용하여 지정된 쿼리 명령문을 다시 최적화합니다. 사용자에게 필수 액세스 특권이 있는 경우 Explain 테이블에는 새로 다시 최적화된 액세스 플랜과 다시 최적화하는 데 사용된 값이 포함됩니다.

다중 파티션 데이터베이스 시스템에서는 명령문이 REOPT ONCE를 사용하여 원래 컴파일되고 다시 최적화된 동일한 데이터베이스 파티션에 Explain되어 있어야 합니다. 그렇지 않으면 오류가 리턴됩니다.

Explain 테이블에서 정보 캡처

- 정적 또는 증분식 바인드 SQL 및 XQuery문

  BIND 또는 PREP 명령에 EXPLAIN ALL 또는 EXPLAIN YES 옵션을 지정하거나 소스 프로그램에 정적 EXPLAIN문을 포함시키십시오.

- 동적 SQL 및 XQuery문

  다음과 같은 경우에 Explain 테이블 정보가 캡처됩니다.

  – CURRENT EXPLAIN MODE 특수 레지스터가 다음과 같이 설정됩니다.
    - YES: SQL 및 XQuery 컴파일러가 Explain 데이터를 캡처하고 쿼리 명령문을 실행합니다.
    - EXPLAIN: SQL 및 XQuery 컴파일러가 Explain 데이터를 캡처하지만 쿼리 명령문을 실행하지 않습니다.
    - RECOMMEND INDEXES: SQL 및 XQuery 컴파일러가 Explain 데이터를 캡처하며 권장 인덱스가 ADVISE_INDEX 테이블에 있지만 쿼리 명령문이 실행되지 않습니다.

- EVALUATE INDEXES: SQL 및 XQuery 컴파일러가 평가를 위해 사용자가 ADVISE_INDEX 테이블에 배치한 인덱스를 사용합니다. 이 모드에서는 이러한 가상 인덱스가 사용 가능한 것처럼 모든 동적문을 Explain합니다. 그런 다음 쿼리 컴파일러는 가상 인덱스가 명령문 성능을 향상시키는 경우 가상 인덱스를 사용하도록 선택합니다. 그렇지 않으면 인덱스는 무시됩니다. 제안된 인덱스가 유용한지 확인하려면 EXPLAIN 결과를 검토하십시오.
- REOPT: 호스트 변수, 매개변수 표시문자, 전역 변수 또는 특수 레지스터의 실제 값이 사용 가능한 경우 쿼리 컴파일러가 실행 시 명령문을 다시 최적화하는 중 정적 또는 동적 SQL 또는 XQuery문에 대한 Explain 데이터를 캡처합니다.

- BIND 또는 PREP 명령에 EXPLAIN ALL 옵션이 지정되었으면 CURRENT EXPLAIN MODE 특수 레지스터가 NO로 설정된 경우에도 쿼리 컴파일러는 런타임에 동적 SQL 및 XQuery문에 대한 Explain 데이터를 캡처합니다.

Explain 스냅샷 정보 캡처

Explain 스냅샷이 요청된 경우 Explain 정보는 Visual Explain에 필요한 형식으로 EXPLAIN_STATEMENT 테이블의 SNAPSHOT 컬럼에 저장됩니다. 이 형식은 다른 응용프로그램에서 사용할 수 없습니다. 데이터 오브젝트 및 데이터 연산자에 대한 정보를 포함하여 Explain 스냅샷에 대한 추가 정보를 Visual Explain 자체에서 사용할 수 있습니다.

다음과 같이 SQL 또는 XQuery문이 컴파일되고 Explain 데이터가 요청되었을 때 Explain 스냅샷 데이터가 캡처됩니다.

- 정적 또는 증분식 바인드 SQL 및 XQuery문

  BIND 또는 PREP 명령에 EXPLSNAP ALL 또는 EXPLSNAP YES절이 지정되었거나 FOR SNAPSHOT 또는 WITH SNAPSHOT절을 사용하는 정적 EXPLAIN 문이 소스 프로그램에 포함된 경우 Explain 스냅샷이 캡처됩니다.

- 동적 SQL 및 XQuery문

  다음과 같은 경우에 Explain 스냅샷이 캡처됩니다.

  – FOR SNAPSHOT 또는 WITH SNAPSHOT절을 사용하여 EXPLAIN문을 발행하십시오. FOR SNAPSHOT절의 경우 Explain 스냅샷 정보만 캡처되며 WITH SNAPSHOT절의 경우 모든 Explain 정보가 캡처됩니다.
  – CURRENT EXPLAIN SNAPSHOT 특수 레지스터가 다음과 같이 설정됩니다.
    - YES: SQL 및 XQuery 컴파일러가 Explain 스냅샷 데이터를 캡처하여 쿼리 명령문을 실행합니다.
    - EXPLAIN: SQL 및 XQuery 컴파일러가 Explain 스냅샷 데이터를 캡처하지만 쿼리 명령문을 실행하지 않습니다.

– BIND 또는 PREP 명령에 EXPLSNAP ALL 옵션을 지정하십시오. 쿼리 컴파일러는 CURRENT EXPLAIN SNAPSHOT 특수 레지스터가 NO로 설정된 경우에도 런타임에 Explain 스냅샷을 캡처합니다.

Explain 정보 사용 지침

Explain 정보를 사용하면 응용프로그램의 성능이 변경된 이유를 알 수 있으며 성능 조정의 작용력을 평가할 수 있습니다.

성능 변경 분석

쿼리 성능의 변경 이유를 알려면 다음 단계를 수행하여 얻을 수 있는 『사전 및 사후』 Explain 정보가 필요합니다.

1. 변경하기 전에 쿼리에 대한 Explain 정보를 캡처하고 결과 Explain 테이블을 저장하십시오. 또는 db2exfmt Explain 도구의 출력을 저장하십시오.
2. 이 정보를 보기 위해 Visual Explain에 액세스할 수 없는 경우 현재 카탈로그 통계를 저장하거나 인쇄하십시오. db2look 제작 도구를 사용하면 이 태스크를 수행하는 데 도움이 됩니다.
3. CREATE TABLE, CREATE VIEW, CREATE INDEX 또는 CREATE TABLESPACE를 포함한 DDL(Data Definition Language)문을 저장하거나 인쇄하십시오.

이러한 방식으로 수집하는 정보는 향후 분석에서 참조할 수 있습니다. 동적 SQL 또는 XQuery문의 경우 처음으로 응용프로그램을 실행할 때 이 정보를 수집할 수 있습니다. 정적 SQL 및 XQuery문의 경우 바인드 시에 이 정보를 수집할 수 있습니다. 성능 변경을 분석하려면 수집한 정보를 이전에 수집된 이 참조 정보와 비교하십시오.

예를 들어, 분석은 액세스 경로 판별 시 인덱스를 더 이상 사용하지 않음을 표시할 수 있습니다. Visual Explain에서 카탈로그 통계 정보를 사용하는 경우 쿼리가 처음 데이터베이스에 바인드될 때보다 인덱스 레벨(NLEVELS 컬럼)이 충분히 높아졌음을 확인할 수 있습니다. 다음 조치 중 하나를 선택하여 수행하십시오.

- 인덱스 재구성
- 테이블 및 인덱스의 새 통계 수집
- 쿼리 리바인드 시 Explain 정보 수집

이러한 조치 중 하나를 수행한 후에 액세스 플랜을 다시 검사하십시오. 인덱스를 사용중인 경우 쿼리 성능은 더 이상 문제가 되지 않을 수도 있습니다. 인덱스를 계속 사용하지 않거나 성능이 계속 문제가 되는 경우 이 목록에서 다른 조치를 선택하고 결과를 조사하십시오. 문제가 해결될 때까지 이러한 단계를 반복하십시오.

성능 조정 노력 평가

구성 매개변수 갱신, 컨테이너 추가, 새 카탈로그 통계 수집 등과 같이 쿼리 성능을 향상시키는 데 유용한 여러 조치를 수행할 수 있습니다.

이러한 영역에서 변경한 후 Explain 기능을 사용하여 변경이 선택된 액세스 플랜에 미치는 영향(있는 경우)을 판별하십시오. 예를 들어, 인덱스 지침에 따라 인덱스 또는 구체화된 쿼리 테이블(MQT)을 추가하는 경우 Explain 데이터는 인덱스 또는 MQT이 예상대로 사용되고 있는지 판별하는 데 유용합니다.

Explain 출력을 통해 선택된 액세스 플랜 및 해당 상대 비용을 판별할 수 있지만 특정 쿼리의 성능 향상을 정확하게 측정하려면 벤치마크 테스팅 기술을 사용해야 합니다.

Explain 정보 분석 지침

Explain 정보는 주로 쿼리 명령문의 액세스 경로를 분석할 때 사용합니다. Explain 데이터 분석이 쿼리 및 환경을 조정할 때 여러 가지 방식으로 도움이 될 수 있습니다.

다음과 같은 유형의 분석을 고려하십시오.

- 인덱스 용도

  적절한 인덱스는 성능에 상당한 이점이 될 수 있습니다. Explain 출력을 사용하면 특정 쿼리 세트를 지원하기 위해 작성한 인덱스가 사용 중인지 여부를 판별할 수 있습니다. 다음 영역에서 인덱스 사용을 확인하십시오.

  – Join 술어
  – 로컬 술어
  – GROUP BY절
  – ORDER BY절
  – WHERE XMLEXISTS절
  – 선택 목록

  Explain 기능을 사용하여 다른 인덱스를 사용하거나 인덱스를 사용하지 않는 것이 좋을지 평가할 수도 있습니다. 새 인덱스를 작성한 후 RUNSTATS 명령을 사용하여 해당 인덱스의 통계를 수집한 후 쿼리를 재컴파일하십시오. 시간이 지남에 따라 인덱스 스캔 대신 테이블 스캔을 사용 중임을 알 수 있습니다(Explain 데이터를 통해). 따라서 테이블 데이터의 클러스터링이 변경될 수 있습니다. 이전에 사용 중인 인덱스가 현재 낮은 클러스터 비율을 나타내는 경우 다음을 수행할 수 있습니다.

  – 인덱스에 따라 데이터를 클러스터링하도록 테이블 재구성
  – RUNSTATS 명령을 사용하여 인덱스 및 테이블에 대한 통계 수집
  – 쿼리 재컴파일

테이블 재구성이 액세스 플랜을 향상시켰는지 판별하려면 재컴파일된 쿼리의 Explain 출력을 조사하십시오.

- 액세스 유형

  Explain 출력을 분석하고 실행 중인 응용프로그램의 유형에 맞게 최적화되지 않은 데이터 액세스 유형을 찾으십시오. 예를 들어, 다음과 같습니다.

  – 온라인 트랜잭션 처리(OLTP) 쿼리

    OLTP 응용프로그램은 키 컬럼에 대해 등호 술어로 규정된 일부 행만 리턴하는 경향이 있으므로 범위 구분 술어를 사용한 인덱스 스캔을 수행할 수 있는 좋은 예입니다. OLTP 쿼리가 테이블 스캔을 사용 중인 경우 Explain 데이터를 분석하여 인덱스 스캔을 사용하지 않는 이유를 판별할 수 있습니다.

  – 찾아보기 전용 쿼리

    『찾아보기』 유형 쿼리의 검색 기준은 매우 모호하므로 규정 행이 많아질 수 있습니다. 사용자가 보통 출력 데이터의 일부 화면만 보는 경우 일부 결과가 리턴되기 전에 전체 응답 세트를 계산하지 않도록 지정할 수도 있습니다. 이 경우 사용자의 목표는 데이터의 처음 일부 화면만이 아닌 전체 쿼리의 자원 사용량을 최소화하려고 시도하는 옵티마이저의 기본 작동 원칙과 다릅니다.

    예를 들어, Explain 출력이 병합 스캔 조인 및 정렬 연산자가 모두 액세스 플랜에 사용되었음을 표시하는 경우 응용프로그램에 행이 리턴되기 전에 전체 응답 세트가 임시 테이블에 구체화됩니다. 이 경우 SELECT문에 OPTIMIZE FOR절을 사용하여 액세스 플랜을 변경하려고 할 수 있습니다. 이 옵션을 지정하면 옵티마이저는 응용프로그램에 첫 번째 행을 리턴하기 전에 임시 테이블에서 전체 응답 세트를 작성하지 않는 액세스 플랜을 선택하려고 할 수 있습니다.

- 조인 방법

  쿼리가 두 테이블을 조인하는 경우 사용 중인 조인 유형을 점검하십시오. 의사 결정 지원 쿼리의 조인과 같은 추가 행이 관련된 조인은 보통 해시 조인 또는 병합 조인을 사용하면 신속하게 실행됩니다. OLTP 쿼리의 조인과 같이 일부 행만 관련된 조인은 보통 중첩된 루프 조인을 사용하면 신속하게 실행됩니다. 그러나 어느 경우에나 일반 조인의 작동 방식을 변경할 수 있는 이해가 가능한 상황이 있을 수 있습니다(예: 로컬 술어 또는 인덱스 사용).

액세스 플랜을 사용하여 REFRESH TABLE 및 SET INTEGRITY문의 성능 문제점 자체 진단

REFRESH TABLE 또는 SET INTEGRITY문에 대해 Explain 유틸리티를 호출하면 이러한 명령문의 성능 문제점을 자체 진단하는 데 사용될 수 있는 액세스 플랜을 생성할 수 있습니다. 이는 구체화된 쿼리 테이블(MQT)을 더 잘 유지하는 데 도움이 될 수 있습니다.

REFRESH TABLE 또는 SET INTEGRITY문에 대한 액세스 플랜을 얻으려면 다음 방법 중 하나를 사용하십시오.

- EXPLAIN문에서 EXPLAIN PLAN FOR REFRESH TABLE 또는 EXPLAIN PLAN FOR SET INTEGRITY 옵션을 사용하십시오.
- REFRESH TABLE 또는 SET INTEGRITY문을 발행하기 전에 CURRENT EXPLAIN MODE 특수 레지스터를 EXPLAIN으로 설정한 다음 이후에 CURRENT EXPLAIN MODE 특수 레지스터를 NO로 설정하십시오.

제한사항

- REFRESH TABLE 및 SET INTEGRITY문은 재최적화 자격이 없으므로, 이러한 두 명령문에 REOPT Explain 모드(또는 Explain 스냅샷)를 적용할 수 없습니다.
- 지정된 설명 가능한 명령문이 재최적화됨을 표시하는 EXPLAIN문의 WITH REOPT ONCE 절을 REFRESH TABLE 및 SET INTEGRITY문에 적용할 수 없습니다.

시나리오

이 시나리오는 EXPLAIN 및 REFRESH TABLE문에서 액세스 플랜을 생성 및 사용하여 성능 문제점의 원인을 자체 진단할 수 있는 방법을 보여줍니다.

1. 테이블을 작성하고 채우십시오. 예를 들어, 다음과 같습니다.

```
create table t (
  i1 int not null,
  i2 int not null,
  primary key (i1)
);

insert into t values (1,1), (2,1), (3,2), (4,2);

create table mqt as (
  select i2, count(*) as cnt from t group by i2
)
data initially deferred
refresh deferred;
```

2. 다음과 같이 EXPLAIN 및 REFRESH TABLE문을 발행하십시오.

```
explain plan for refresh table mqt;
```

   다음과 같이 SET CURRENT EXPLAIN MODE 특수 레지스터에서 EXPLAIN 모드를 설정하는 것으로 이 단계를 교체할 수 있습니다.

```
set current explain mode explain;
refresh table mqt;
set current explain mode no;
```

3. db2exfmt 명령을 사용하여 Explain 테이블의 컨텐츠를 형식화하고 액세스 플랜을 얻으십시오. 이 도구는 인스턴스 sqllib 디렉토리의 misc 서브디렉토리에 있습니다.

```
db2exfmt –d dbname -o refresh.exp -1
```

4. 액세스 플랜을 분석하여 성능 문제점의 원인을 판별하십시오. 위의 예에서, T가 대형 테이블인 경우, 테이블 스캔에 비용이 매우 많이 듭니다. 인덱스를 정리하면 쿼리의 성능이 향상될 수도 있습니다.

Explain 정보 수집 및 분석 도구

DB2 데이터베이스 서버에는 옵티마이저가 SQL 또는 XQuery문에서 선택하는 액세스 플랜에 관한 자세한 정보를 제공하는 종합 Explain 기능이 있습니다.

Explain 데이터를 저장하는 테이블은 지원되는 모든 플랫폼에서 액세스할 수 있으며 정적, 동적 SQL 및 XQuery문에 대한 정보를 포함하고 있습니다. Explain 정보를 캡처, 표시 및 분석하는 데 필요한 유연성을 제공하는 여러 도구를 사용할 수 있습니다.

액세스 플랜을 자세히 분석하는 데 사용할 수 있는 세부 쿼리 옵티마이저 정보는 실제 액세스 플랜 자체와 다른 별도의 Explain 테이블에 저장됩니다. 다음 중 하나 이상의 방법을 사용하여 Explain 테이블에서 정보를 얻으십시오.

- db2exfmt 도구를 사용하여 형식화된 출력으로 Explain 정보를 표시하십시오.
- Explain 테이블에 대해 직접 쿼리를 작성하십시오. 직접 쿼리를 작성하면 출력을 쉽게 조작하거나 여러 쿼리들을 비교하거나 시간 경과에 따른 동일한 쿼리 실행들을 비교할 수 있습니다.

db2expln 도구를 사용하여 정적 SQL 또는 XQuery문의 하나 이상의 패키지에 사용할 수 있는 액세스 플랜 정보를 보십시오. 이 유틸리티는 선택한 액세스 플랜의 실제 구현을 표시합니다. 옵티마이저 정보를 표시하지 않습니다. db2expln 도구는 생성된 액세스 플랜을 조사하여 런타임에 발생하는 조작에 대해 비교적 간략한 문자 그대로의 개요를 제공합니다.

명령행 Explain 도구는 `sqllib` 디렉토리의 `misc` 서브디렉토리에서 찾을 수 있습니다.

다음 표는 DB2 Explain 기능에서 사용 가능한 여러 가지 도구를 요약합니다. 이 표를 사용하여 사용자의 환경과 필요에 가장 적합한 도구를 선택하십시오.

표 53. Explain 기능 도구

원하는 특성	Explain 테이블	db2expln	db2exfmt
텍스트 출력		예	예
『퀵 앤 더티』 정적 SQL 및 XQuery 분석		예	
정적 SQL 및 XQuery 지원	예	예	예
동적 SQL 및 XQuery 지원	예	예	예
CLI 응용프로그램 지원	예		예
DRDA® 응용프로그램 리퀘스터에 사용 가능	예		
세부 옵티마이저 정보	예		예
다중 명령문 분석에 적합	예	예	예
응용프로그램에서 정보에 액세스 가능	예		

Explain 시간에 적용되는 카탈로그 통계 표시

Explain 기능은 명령문 Explain 시 적용되는 통계를 캡처합니다. 이러한 통계는 시스템 카탈로그에 저장된 통계와 다를 수 있으며 특히 실시간 통계 수집이 사용 가능한 경우에 다릅니다. Explain 테이블이 채워지지만 Explain 스냅샷이 작성되지 않은 경우 일부 통계만 EXPLAIN_OBJECT 테이블에 기록됩니다.

Explain 중인 명령문과 관련된 모든 카탈로그 통계를 캡처하려면 Explain 테이블이 채워지는 동시에 Explain 스냅샷을 작성한 후 SYSPROC.EXPLAIN_FORMAT_STATS 스칼라 함수를 사용하여 스냅샷에서 카탈로그 통계를 형식화하십시오.

db2exfmt 도구를 사용하여 Explain 정보를 형식화하며 Explain 스냅샷이 수집된 경우 도구는 SYSPROC.EXPLAIN_FORMAT_STATS 함수를 자동으로 사용하여 카탈로그 통계를 표시합니다.

Explain 테이블 및 Explain 정보 구성

모든 Explain 정보는 Explain 인스턴스 개념에 따라 구성합니다. *Explain 인스턴스*는 하나 이상의 SQL 또는 XQuery문에 대한 하나의 Explain 기능 호출을 나타냅니다. 하나의 Explain 인스턴스에 캡처된 Explain 정보에는 컴파일 중인 SQL 또는 XQuery문을 충족시키기 위해 선택한 액세스 플랜과 컴파일 환경이 있습니다.

예를 들어, Explain 인스턴스는 다음 중 하나로 구성될 수 있습니다.

- 정적 쿼리 명령문에 대해 하나의 패키지에 있는 모든 적격 SQL 또는 XQuery문. SQL문(XML 데이터를 쿼리하는 명령문 포함)의 경우 CALL, 복합 SQL(동적), DELETE, INSERT, MERGE, REFRESH TABLE, SELECT, SET INTEGRITY, SELECT INTO, UPDATE, VALUES 및 VALUES INTO문에 대한 Explain 정보를 캡처할 수 있습니다. XQuery문의 경우 XQUERY db2-fn:xmlcolumn 및 XQUERY db2-fn:sqlquery문에 대한 Explain 정보를 얻을 수 있습니다.

  주: REFRESH TABLE 및 SET INTEGRITY문은 동적으로만 컴파일됩니다.
- 증분식 바인드 SQL문에 대한 하나의 특정 SQL문
- 동적 SQL문에 대한 하나의 특정 SQL문
- 각 EXPLAIN문(동적 또는 정적)

특정 설명 가능한 명령문에서 선택한 액세스 플랜에 관한 정보를 캡처하고 이 정보를 Explain 테이블에 기록하는 EXPLAIN문을 발행하여 Explain 기능을 호출합니다. EXPLAIN문을 발행하기 전에 Explain 테이블을 작성해야 합니다. 작성하려면 `sqllib` 서브디렉토리의 `misc` 서브디렉토리에서 찾을 수 있는 `EXPLAIN.DDL` 스크립트를 실행하십시오.

SYSPROC.SYSINSTALLOBJECTS 프로시저를 사용하여 Explain 테이블을 작성, 삭제 및 유효성 확인할 수도 있습니다. 특정 스키마 및 테이블 스페이스에서 테이블을 작성할 수 있습니다. EXPLAIN.DDL 파일에서 예를 찾을 수 있습니다.

Explain 테이블은 여러 사용자에게 공통적으로 해당될 수 있습니다. 한 사용자에 대한 테이블을 정의한 후 각 추가 사용자에 대해 정의된 테이블을 지정하는 별명을 작성할 수 있습니다. 또는 SYSTOOLS 스키마에서 Explain 테이블을 정의할 수 있습니다. 동적 SQL 또는 XQuery문의 사용자 세션 ID나 정적 SQL 또는 XQuery문의 명령문 권한 부여 ID에서 다른 Explain 테이블 또는 별명을 찾을 수 없는 경우 Explain 기능은 SYSTOOLS 스키마를 디폴트로 사용합니다. 일반 Explain 테이블을 공유하는 각 사용자는 이러한 테이블에 대해 INSERT 특권을 보유하고 있어야 합니다.

표 54. Explain 테이블 요약

테이블 이름	설명
ADVISE_INDEX	권장 인덱스에 대한 정보를 저장합니다. 테이블을 쿼리 컴파일러, db2advis 명령 또는 사용자로 채울 수 있습니다. 이 테이블은 다음과 같은 경우에 사용합니다. • 권장 인덱스 가져오기 • 제안된 인덱스에 대한 입력에 따른 인덱스 평가
ADVISE_INSTANCE	시작 시간을 포함하여 db2advis 실행에 대한 정보가 있습니다. db2advis의 각 실행에 해당하는 하나의 행이 있습니다.
ADVISE_MQT	권장되는 각각의 구체화된 쿼리 테이블(MQT), 각 MQT의 컬럼 통계(예: COLSTATS(XML 형식), NUMROWS 등)를 정의하는 쿼리 및 각 MQT의 세부 통계를 확보하기 위한 샘플링 쿼리가 있습니다.
ADVISE_PARTITION	db2advis에서 생성 및 평가하는 가상 데이터베이스 파티션을 저장합니다.
ADVISE_TABLE	MQT, 다차원 클러스터링 테이블(MDC) 및 데이터베이스 파티셔닝의 최종 디자인 어드바이저 권장사항을 사용하여 테이블 작성에 대한 DDL(Data Definition Language)을 저장합니다.
ADVISE_WORKLOAD	테이블의 각 행은 워크로드의 SQL 또는 XQuery문을 표시합니다. db2advis 명령은 이 테이블을 사용하여 워크로드 정보를 수집 및 저장합니다.
EXPLAIN_ARGUMENT	각 개별 연산자(있는 경우)의 고유 특성에 대한 정보가 있습니다.
EXPLAIN_DIAGNOSTIC	EXPLAIN_STATEMENT 테이블에 있는 Explain문의 특정 인스턴스에 대해 작성되는 각 진단 메시지의 항목이 있습니다.
EXPLAIN_DIAGNOSTIC_DATA	EXPLAIN_DIAGNOSTIC 테이블에 기록된 특정 진단 메시지의 메시지 토큰이 있습니다. 메시지 토큰은 메시지를 생성한 SQL문의 실행에 대한 추가 정보를 제공합니다.

표 54. Explain 테이블 요약 (계속)

테이블 이름	설명
EXPLAIN_INSTANCE	모든 Explain 정보에 대한 기본 제어 테이블입니다. Explain 테이블의 각 행은 이 테이블에 있는 하나의 고유 행에 명시적으로 링크됩니다. Explain 중인 SQL 또는 XQuery문의 소스에 대한 기본 정보와 환경 정보를 이 테이블에 보존합니다.
EXPLAIN_OBJECT	SQL 또는 XQuery문을 충족시키기 위해 생성된 액세스 플랜에 필요한 데이터 오브젝트를 식별합니다.
EXPLAIN_OPERATOR	쿼리 컴파일러가 SQL 또는 XQuery문을 충족시키기 위해 필요한 모든 연산자가 있습니다.
EXPLAIN_PREDICATE	특정 연산자가 적용하는 술어를 식별합니다.
EXPLAIN_STATEMENT	여러 가지 Explain 정보 레벨에 맞게 존재하는 SQL 또는 XQuery문의 텍스트가 들어 있습니다. 사용자가 입력한 원래의 SQL 또는 XQuery문은 옵티마이저가 액세스 플랜을 선택하기 위해 사용하는 버전으로 이 테이블에 저장됩니다. Explain 스냅샷이 요청된 경우 쿼리 옵티마이저가 선택한 액세스 플랜을 설명하는 추가 Explain 정보가 기록됩니다. 이 정보는 Visual Explain에 필요한 형식으로 EXPLAIN_STATEMENT 테이블의 SNAPSHOT 컬럼에 저장됩니다. 이 형식은 다른 응용프로그램에서 사용할 수 없습니다.
EXPLAIN_STREAM	개별 연산자와 데이터 오브젝트 간 입력 및 출력 데이터 스트림을 나타냅니다. 데이터 오브젝트 자체는 EXPLAIN_OBJECT 테이블에 표시됩니다. 데이터 스트림에 관련된 연산자는 EXPLAIN_OPERATOR 테이블에 표시됩니다.

데이터 오브젝트에 대한 Explain 정보:

단일 액세스 플랜은 하나 이상의 데이터 오브젝트를 사용하여 SQL 또는 XQuery문을 충족시킬 수 있습니다.

오브젝트 통계

Explain 기능은 다음과 같이 각 오브젝트에 대한 정보를 기록합니다.

- 작성 시간
- 마지막으로 오브젝트에 대한 통계를 수집한 시간
- 오브젝트의 데이터 정렬 여부(테이블 또는 인덱스 오브젝트만)
- 오브젝트의 컬럼 수(테이블 또는 인덱스 오브젝트만)
- 오브젝트의 추정 행 수(테이블 또는 인덱스 오브젝트만)
- 버퍼 풀에서 오브젝트가 차지하는 페이지 수
- 오브젝트가 저장되어 있는 지정된 테이블 스페이스에 대한 단일 무작위 입출력의 전체 추정 오버헤드(밀리초)

- 지정된 테이블 스페이스에서 4-KB 페이지를 읽는 추정 전송률(밀리초)
- 프리페치 및 Extent 크기(4-KB 페이지 단위)
- 해당 인덱스를 갖는 데이터 클러스터링의 등급
- 이 오브젝트의 인덱스에서 사용하는 리프 페이지 수와 트리의 레벨 수
- 이 오브젝트에 대한 인덱스에 있는 구별된 전체 키 값 수
- 테이블의 전체 오버플로우 레코드 수

데이터 연산자에 대한 Explain 정보:

단일 액세스 플랜은 SQL 또는 XQuery문을 충족시키기 위해 데이터에 대해 여러 연산을 수행한 후 사용자에게 결과를 다시 제공할 수 있습니다. 쿼리 컴파일러는 테이블 스캔, 인덱스 스캔, 중첩된 루프 조인 또는 group-by 연산자와 같은 필요한 연산을 판별합니다.

액세스 플랜에서 사용되는 각 연산자에 대한 정보를 표시하는 동시에 Explain 출력은 액세스 플랜의 누적 효과도 표시합니다.

계산된 비용 정보

연산자에 대한 다음의 추정 누적 비용이 기록됩니다. 이러한 비용은 선택한 액세스 플랜에 대해 정보가 캡처된 연산자까지 해당됩니다.

- 전체 비용(timeron 단위)
- 페이지 입출력 수
- 처리 명령어 수
- 필수 초기 오버헤드를 포함하여 첫 번째 행을 페치하는 비용(timeron 단위)
- 통신 비용(프레임 단위)

*timeron*은 새로운 상대적 수치 단위입니다. 데이터베이스 사용에 따라 변경되는 통계와 같은 내부 값에 기초하여 옵티마이저가 timeron 값을 판별합니다. 따라서 SQL 또는 XQuery문의 timeron 값은 timeron 단위의 계산된 비용이 판별될 때마다 동일하게 유지된다고 보장할 수 없습니다.

연산자 등록 정보

각 연산자의 등록 정보에 대해 설명하는 다음 정보는 Explain 기능으로 기록됩니다.

- 액세스한 테이블 세트
- 액세스한 컬럼 세트
- 데이터가 정렬된 컬럼(옵티마이저가 후속 연산자에서 이 순서를 사용할 수 있다고 판별한 경우)
- 적용된 술어 세트

- 리턴되는 추정 행 수(카디널리티)

인스턴스에 대한 Explain 정보:

Explain 인스턴스 정보는 EXPLAIN_INSTANCE 테이블에 저장됩니다. 인스턴스의 각 쿼리 명령문에 대한 추가 특정 정보는 EXPLAIN_STATEMENT 테이블에 저장됩니다.

Explain 인스턴스 식별

다음 정보는 특정 Explain 인스턴스를 식별하고 특정 명령문에 대한 정보를 Explain 기능의 특정 호출과 연관시킬 때 유용합니다.

- Explain 정보를 요청한 사용자
- Explain 요청이 시작된 시간
- Explain문이 들어 있는 패키지의 이름
- Explain문이 들어 있는 패키지의 SQL 스키마
- 명령문이 들어 있는 패키지의 버전
- 스냅샷 정보가 수집되었는지 여부

환경 설정

쿼리 컴파일러가 쿼리를 최적화한 데이터베이스 관리 프로그램 환경에 대한 정보가 캡처됩니다. 환경 정보에는 다음과 같은 내용이 포함됩니다.

- DB2 제품의 버전 및 릴리스 번호
- 쿼리를 컴파일한 병렬 처리 수준

  CURRENT DEGREE 특수 레지스터, DEGREE 바인드 옵션, SET RUNTIME DEGREE 명령 및 **dft_degree** 데이터베이스 구성 매개변수는 특정 쿼리를 컴파일하는 병렬 처리 수준을 판별합니다.
- 동적 또는 정적 명령문인지 여부
- 쿼리를 컴파일하는 데 사용된 쿼리 최적화 클래스
- 쿼리 컴파일 시 발생하는 커서의 행 블로킹 유형
- 쿼리가 실행되는 분리 수준
- 쿼리 컴파일 시 다양한 구성 매개변수의 값. Explain 스냅샷을 작성할 때 다음 매개변수의 값이 기록됩니다.
  - 정렬 힙 크기(**sortheap**)
  - 평균 활성 응용프로그램 수(**avg_appls**)
  - 데이터베이스 힙(**dbheap**)
  - 잠금 목록의 최대 스토리지(**locklist**)

- 에스컬레이션하기 전 잠금 목록의 최대 퍼센트(**maxlocks**)
- CPU 속도(**cpuspeed**)
- 통신 대역폭(**comm_bandwidth**)

명령문 식별

각 Explain 인스턴스마다 여러 명령문이 Explain되었을 수 있습니다. Explain 인스턴스를 고유하게 식별하는 정보 외에 다음 정보는 개별 쿼리 명령문을 식별하는 데 유용합니다.

- 명령문 유형: SELECT, DELETE, INSERT, UPDATE, 위치 지정된 DELETE, 위치 지정된 UPDATE 또는 SET INTEGRITY
- SYSCAT.STATEMENTS 카탈로그 뷰에 기록된 명령문을 발행하는 패키지의 명령문 및 섹션 번호

EXPLAIN_STATEMENT 테이블의 QUERYTAG 및 QUERYNO 필드에는 Explain 프로세스의 일부분으로 설정되는 ID가 들어 있습니다. EXPLAIN MODE 또는 EXPLAIN SNAPSHOT을 사용 중이며 명령행 처리기(CLP) 또는 호출 레벨 인터페이스(CLI) 세션 중 동적 Explain문이 제출된 경우 QUERYTAG 값은 각각 『CLP』 또는 『CLI』로 설정됩니다. 이 경우 QUERYNO 값은 각 명령문마다 하나 이상씩 증분하는 숫자를 디폴트로 사용합니다. CLP 또는 CLI에 속하지 않았거나 EXPLAIN문을 사용하지 않는 기타 모든 동적 Explain문의 경우 QUERYTAG 값은 공백으로 설정되며 QUERYNO는 항상 1입니다.

비용 추정

각 Explain문마다 옵티마이저는 선택된 액세스 플랜을 실행하는 상대적 비용 추정을 기록합니다. 이 비용은 *timeron*이라고 하는 새로운 상대적 수치 단위로 표시합니다. 다음과 같은 이유로 인해 경과 시간 추정은 제공되지 않습니다.

- 쿼리 옵티마이저가 경과 시간을 추정하지 않고 자원 사용량만 추정합니다.
- 옵티마이저가 경과 시간에 영향을 줄 수 있는 모든 요인을 모델링하지 않습니다. 액세스 플랜의 효율성에 영향을 주지 않는 요인을 무시합니다. 시스템 워크로드, 자원 경쟁 수, 병렬 처리 및 입출력 수, 사용자에게 행을 리턴하는 비용 및 클라이언트와 서버 간 통신 시간을 포함한 여러 가지 런타임 요인이 경과 시간에 영향을 줍니다.

명령문 텍스트

각 Explain문마다 두 가지 버전의 명령문 텍스트가 기록됩니다. 한 가지 버전은 쿼리 컴파일러가 응용프로그램으로부터 받는 코드입니다. 다른 버전은 쿼리의 내부(컴파일러) 표현에서 역변환됩니다. 이 변환은 기타 쿼리 명령문과 비슷해 보이지만 올바른 쿼리 언어 구문을 따르지 않고 내부 표현의 실제 경쟁을 전체적으로 반영하지도 않습니다. 이 변환은 옵티마이저가 액세스 플랜을 선택한 컨텍스트를 이해할 수 있도록 제공되는 것

입니다. 컴파일러가 최적화를 적절하게 수행하도록 쿼리를 다시 쓴 방식을 이해하려면 사용자 작성 명령문 텍스트를 쿼리 명령문의 내부 표현과 비교하십시오. 다시 쓴 명령문은 트리거 또는 제한조건과 같이 명령문에 영향을 주는 기타 요인도 표시합니다. 이 『최적화된』 텍스트에 사용된 일부 키워드는 다음과 같습니다.

$C*n* 파생 컬럼의 이름입니다. 여기서 *n*은 정수값을 표시합니다.

$CONSTRAINT$
이 태그는 컴파일 중 원래의 명령문에 추가된 제한조건을 식별하며 $WITH_CONTEXT$ 접두부와 연결되어 표시됩니다.

$DERIVED.T*n*
파생된 테이블의 이름입니다. 여기서 *n*은 정수값을 표시합니다.

$INTERNAL_FUNC$
이 태그는 Explain된 쿼리에 대해 컴파일러가 사용하지만 일반적으로 사용할 수 없는 함수가 있음을 표시합니다.

$INTERNAL_PRED$
이 태그는 Explain된 쿼리의 컴파일 중 추가되었지만 일반적으로 사용할 수 없는 술어가 있음을 표시합니다. 내부 술어는 컴파일러가 트리거 또는 제한조건으로 인해 원래의 명령문에 추가된 컨텍스트를 충족시키는 데 사용합니다.

$INTERNAL_XPATH$
이 태그는 단일 어노테이션 XPath 패턴을 입력 매개변수로 사용하고 해당 패턴과 일치하는 하나 이상의 컬럼이 있는 테이블을 리턴하는 내부 테이블 함수가 있음을 표시합니다.

RID
이 태그는 특정 행의 행 ID(RID) 컬럼을 표시합니다.

$TRIGGER$
이 태그는 컴파일 중 원래의 명령문에 추가된 트리거를 식별하며 $WITH_CONTEXT$ 접두부와 연결되어 표시됩니다.

$WITH_CONTEXT$(...)
이 접두부는 원래의 쿼리 명령문에 트리거 또는 제한조건이 추가되었을 때 텍스트의 처음에 표시됩니다. 명령문의 컴파일 및 분석에 영향을 주는 트리거 또는 제한조건의 이름 목록이 이 접두부 뒤에 표시됩니다.

SQL 및 XQuery Explain 도구

db2expln 명령은 SQL 또는 XQuery문에 대해 선택한 액세스 플랜에 대해 설명합니다.

이 도구를 사용하면 Explain 데이터가 캡처되었을 때 선택한 액세스 플랜에 대한 빠른 설명을 볼 수 있습니다. 정적 SQL 및 XQuery문의 경우 db2expln은 시스템 카탈로그에 저장된 패키지를 조사합니다. 동적 SQL 및 XQuery문의 경우 db2expln은 쿼리 캐시에서 해당 섹션을 조사합니다.

Explain 도구는 인스턴스 sqllib 디렉토리의 bin 서브디렉토리에 있습니다. db2expln이 현재 디렉토리에 없는 경우 PATH 환경 변수에 표시되는 디렉토리에 있어야 합니다.

db2expln 명령은 db2expln.bnd, db2exsrv.bnd 및 db2exdyn.bnd 파일을 사용하여 처음 데이터베이스에 액세스할 때 데이터베이스에 바인드합니다.

db2expln 출력 설명:

db2expln 명령의 Explain 출력에는 패키지 정보와 각 패키지에 대한 섹션 정보가 모두 포함되어 있습니다.

- 패키지 정보에는 바인드 조작의 날짜와 관련 바인드 옵션이 포함됩니다.
- 섹션 정보에는 섹션 번호와 Explain 중인 SQL 또는 XQuery문이 포함됩니다.

SQL 또는 XQuery문에 대해 선택한 액세스 플랜과 관련된 Explain 출력은 섹션 정보 아래에 표시됩니다.

액세스 플랜 또는 섹션의 단계는 데이터베이스 관리 프로그램이 실행하는 순서대로 표시됩니다. 각 주요 단계는 왼쪽을 맞춘 표제로 표시되며 해당 단계에 대한 정보를 그 아래에 들여씁니다. 액세스 플랜에 대한 Explain 출력의 왼쪽 여백에 들여쓰기 막대가 표시됩니다. 이러한 막대는 각 조작의 범위도 표시합니다. 가장 오른쪽에 있는 최저 레벨로 들여쓴 조작은 이전 들여쓰기 레벨에 표시된 조작보다 먼저 처리합니다.

선택된 액세스 플랜은 명령문 집중기가 사용되는 경우 원래 SQL문의 나중 버전인 *효과적인 SQL*문에 기초하며, 명령문 집중기가 사용되지 않는 경우 출력에 표시된 XQuery문에 기초합니다. 컴파일러의 쿼리 재작성 구성요소는 SQL 또는 XQuery문을 동등하지만 효율적인 형식으로 변환할 수 있으므로 Explain 출력에 표시되는 액세스 플랜은 예상과 완전히 다를 수 있습니다. Explain 테이블, SET CURRENT EXPLAIN MODE문 및 Visual Explain을 포함한 Explain 기능은 쿼리의 내부 표현을 역변환하여 작성된 유사한 SQL 또는 XQuery문 형식으로 최적화에 사용된 실제 SQL 또는 XQuery문을 나타냅니다.

db2expln의 출력을 Explain 기능의 출력과 비교할 때 연산자 ID 옵션(-opids)이 매우 유용할 수 있습니다. db2expln이 Explain 기능에서 새 연산자를 처리하기 시작할 때마다 연산자 ID 번호가 Explain 플랜의 왼쪽에 인쇄됩니다. 연산자 ID를 사용하면 액세스 플랜의 다른 표현에서 단계를 비교할 수 있습니다. Explain 기능 출력의 연산자와 db2expln에서 표시하는 연산이 항상 일대일로 대응되지는 않는다는 점을 참고하십시오.

테이블 액세스 정보:

db2expln 출력의 명령문은 액세스하는 테이블의 이름 및 유형을 제공합니다.

일반 테이블에 대한 정보에는 다음과 같은 테이블 액세스 명령문 중 하나가 포함됩니다.

```
Access Table Name = schema.name  ID = ts,n
Access Hierarchy Table Name = schema.name  ID = ts,n
Access Materialized Query Table Name = schema.name  ID = ts,n
```

각 항목에 대한 설명은 다음과 같습니다.

- *schema.name*은 액세스 중인 테이블의 완전한 이름입니다.
- ID는 테이블에 대한 SYSCAT.TABLES 카탈로그 뷰 항목의 해당 TABLESPACEID 및 TABLEID입니다.

임시 테이블에 대한 정보에는 다음의 테이블 액세스 명령문 중 하나가 포함됩니다.

```
Access Temp Table  ID = tn
Access Global Temp Table  ID = ts,tn
```

여기서 ID는 테이블에 대한 SYSCAT.TABLES 카탈로그 뷰 항목의 해당 TABLESPACEID(*ts*)이거나 db2expln에서 액세스하는 해당 ID(*tn*)입니다.

테이블 액세스 명령문 뒤에는 액세스에 대해 추가로 설명하는 다음의 추가 명령문이 제공됩니다.

- 컬럼 수
- 블록 액세스
- 병렬 스캔
- 스캔 방향
- 행 액세스
- 잠금 의도
- 술어
- 기타

컬럼 수 명령문

다음 명령문은 테이블의 각 행에서 사용 중인 컬럼 수를 표시합니다.

```
#Columns = n
```

블록 액세스 명령문

다음 명령문은 테이블에 하나 이상의 차원 블록 인덱스가 정의되어 있음을 표시합니다.

```
Clustered by Dimension for Block Index Access
```

이 명령문이 표시되지 않은 경우 ORGANIZE BY DIMENSIONS절을 사용하지 않고 테이블이 작성되었습니다.

병렬 스캔 명령문

다음 명령문은 데이터베이스 관리 프로그램이 여러 서브에이전트를 사용하여 테이블을 병렬식으로 읽을 것임을 표시합니다.

```
Parallel Scan
```

이 명령문이 표시되지 않은 경우 한 에이전트(또는 서브에이전트)만 테이블을 읽습니다.

스캔 방향 명령문

다음 명령문은 데이터베이스 관리 프로그램이 역순으로 행을 읽을 것임을 표시합니다.

```
Scan Direction = Reverse
```

이 명령문이 표시되지 않은 경우 스캔 방향은 정방향(디폴트)입니다.

행 액세스 명령문

다음 명령문 중 하나는 테이블의 규정 행에 액세스하는 방식을 표시합니다.

- Relation Scan 명령문은 테이블에서 규정 행을 순차적으로 스캔 중임을 표시합니다.
  - 다음 명령문은 데이터 프리페치를 수행하지 않음을 표시합니다.

    ```
    Relation Scan
    |  Prefetch: None
    ```

  - 다음 명령문은 옵티마이저가 프리페치할 페이지 수를 판별했음을 표시합니다.

    ```
    Relation Scan
    |  Prefetch: n Pages
    ```

  - 다음 명령문은 데이터를 프리페치해야 함을 표시합니다.

    ```
    Relation Scan
    |  Prefetch: Eligible
    ```

- 다음 명령문은 인덱스를 통해 규정 행을 식별 및 액세스 중임을 표시합니다.

  ```
  Index Scan:  Name = schema.name  ID = xx
  | Index type
  | Index Columns:
  ```

  각 항목에 대한 설명은 다음과 같습니다.

  - *schema.name*은 스캔 중인 인덱스의 완전한 이름입니다.
  - ID는 SYSCAT.INDEXES 카탈로그 뷰의 해당 IID 컬럼입니다.
  - 인덱스 유형은 다음 중 하나입니다.

```
Regular index (not clustered)
Regular index (clustered)
Dimension block index
Composite dimension block index
Index over XML data
```

그 뒤에는 인덱스의 각 컬럼에 해당하는 하나의 출력 행이 표시됩니다. 이 정보의 유효한 형식은 다음과 같습니다.

```
n: column_name (Ascending)
n: column_name (Descending)
n: column_name (Include Column)
```

인덱스 스캔의 유형을 명확하게 지정하기 위해 다음 명령문이 제공됩니다.

– 인덱스의 범위 구분 술어는 다음 명령문에 표시되어 있습니다.

```
#Key Columns = n
|  Start Key: xxxxx
|  Stop Key: xxxxx
```

여기서 *xxxxx*는 다음 중 하나입니다.

- `Start of Index`
- `End of Index`
- `Inclusive Value:` 또는 `Exclusive Value:`

  포함 키 값이 인덱스 스캔에 포함됩니다. 제외 키 값은 스캔에 포함되지 않습니다. 키의 값은 키의 각 부분에 대한 다음의 항목 중 하나로 판별됩니다.

  ```
  n: 'string'
  n: nnn
  n: yyyy-mm-dd
  n: hh:mm:ss
  n: yyyy-mm-dd hh:mm:ss.uuuuuu
  n: NULL
  n: ?
  ```

  리터럴 문자열의 처음 20자만 표시됩니다. 문자열이 20자보다 긴 경우 문자열 끝에 생략 부호(...)로 표시됩니다. 일부 키는 섹션이 실행되기 전에는 판별할 수 없습니다. 이러한 키는 물음표(?)가 표시됩니다. 표시됩니다.

– `Index-Only Access`

  인덱스 키에서 필요한 모든 컬럼을 확보할 수 있는 경우 이 명령문이 표시되고 테이블 데이터에 액세스할 수 없습니다.

– 다음 명령문은 인덱스 페이지의 프리페치가 수행되지 않음을 표시합니다.

```
Index Prefetch: None
```

– 다음 명령문은 인덱스 페이지를 프리페치해야 함을 표시합니다.

```
Index Prefetch: Eligible
```

– 다음 명령문은 데이터 페이지의 프리페치가 수행되지 않음을 표시합니다.

```
Data Prefetch: None
```

– 다음 명령문은 데이터 페이지를 프리페치해야 함을 표시합니다.

```
Data Prefetch: Eligible
```

– 인덱스 항목을 규정하도록 인덱스 관리 프로그램으로 패스할 수 있는 술어가 있는 경우 다음 명령문을 사용하여 이러한 술어의 수를 표시합니다.

```
Sargable Index Predicate(s)
|  #Predicates = n
```

- 액세스 플랜에서 초기에 준비된 행 ID(RID)를 통해 규정 행에 액세스하는 경우 다음 명령문을 통해 표시합니다.

```
Fetch Direct Using Row IDs
```

  테이블에 하나 이상의 블록 인덱스가 정의된 경우 블록 또는 행 ID를 사용하여 행에 액세스할 수 있습니다. 다음 명령문으로 표시됩니다.

```
Fetch Direct Using Block or Row IOs
```

잠금 의도 명령문

테이블에 액세스할 때마다 테이블 및 행 레벨에서 획득하는 잠금 유형은 다음 명령문을 사용하여 표시합니다.

```
Lock Intents
|  Table: xxxx
|  Row  : xxxx
```

테이블 잠금에 사용할 수 있는 값은 다음과 같습니다.

- Exclusive
- Intent Exclusive
- Intent None
- Intent Share
- Share
- Share Intent Exclusive
- Super Exclusive
- Update

행 잠금에 사용할 수 있는 값은 다음과 같습니다.

- Exclusive
- Next Key Weak Exclusive
- None
- Share
- Update

술어 명령문

액세스 플랜에서 사용되는 술어에 대한 정보를 제공하는 세 가지 유형의 명령문이 있습니다.

- 다음 명령문은 블로킹된 인덱스에서 검색하는 각 데이터 블록에 대해 평가되는 술어의 수를 표시합니다.

  ```
  Block Predicates(s)
  |  #Predicates = n
  ```

- 다음 명령문은 데이터에 액세스하는 중 평가되는 술어의 수를 표시합니다. 이 수에는 집계 또는 정렬과 같은 푸시다운 연산은 포함되지 않습니다.

  ```
  Sargable Predicate(s)
  |  #Predicates = n
  ```

- 다음 명령문은 데이터가 리턴된 후 평가되는 술어의 수를 표시합니다.

  ```
  Residual Predicate(s)
  |  #Predicates = n
  ```

이러한 명령문에 표시된 술어의 수는 쿼리 명령문에 제공된 술어의 수를 반영하지 않을 수도 있습니다. 이유는 다음과 같습니다.

- 동일한 쿼리에서 술어가 여러 번 적용될 수 있습니다.
- 쿼리 최적화 프로세스 중 내재된 술어의 추가로 술어가 변환 및 확장될 수 있습니다.
- 쿼리 최적화 프로세스 중 술어가 변환되고 적은 수의 술어로 압축될 수 있습니다.

기타 테이블 명령문

- 다음 명령문은 한 행에만 액세스함을 표시합니다.

  ```
  Single Record
  ```

- 다음 명령문은 테이블 액세스에 사용되는 분리 수준이 명령문의 분리 수준과 다를 때 표시됩니다.

  ```
  Isolation Level: xxxx
  ```

  가능한 이유는 여러 가지입니다. 예를 들어, 다음과 같습니다.

  - 반복 읽기(RR) 분리 수준에 바인되는 패키지는 특정 참조 무결성 제한조건에 영향을 줍니다. 이러한 제한조건을 점검하기 위한 상위 테이블 액세스는 이 테이블에서 불필요한 잠금 보유를 방지하기 위해 커서 안정성(CS) 분리 수준으로 낮아집니다.
  - 커미트되지 않은 읽기(UR) 분리 수준에 바인드된 패키지에는 DELETE문이 포함됩니다. 삭제 조작을 위한 테이블의 액세스는 CS로 업그레이드됩니다.

- 다음 명령문은 충분한 **sortheap** 메모리가 사용 가능한 경우 임시 테이블에서 읽은 일부 또는 모든 행이 버퍼 풀 외부에 캐시됨을 표시합니다.

  ```
  Keep Rows In Private Memory
  ```

- 다음 명령문은 테이블에 일시적 카디널리티 속성이 설정되어 있음을 표시합니다.

  ```
  Volatile Cardinality
  ```

임시 테이블 정보:

임시 테이블은 액세스 플랜 실행 중 작업 테이블로 사용됩니다. 일반적으로 액세스 플랜에서 초기에 서브쿼리를 평가해야 하는 경우 또는 중간 결과가 사용 가능한 메모리에 비해 너무 큰 경우 임시 테이블을 사용합니다.

임시 테이블이 필요한 경우 다음 명령문 중 하나가 db2expln 명령 출력에 표시됩니다.

```
Insert Into Temp Table  ID = tn     --> ordinary temporary table
Insert Into Shared Temp Table  ID = tn     --> ordinary temporary table will be created
                                               by multiple subagents in parallel
Insert Into Sorted Temp Table  ID = tn     --> sorted temporary table
Insert Into Sorted Shared Temp Table  ID = tn     --> sorted temporary table will be created
                                                      by multiple subagents in parallel

Insert Into Global Temp Table  ID = ts,tn     --> declared global temporary table
Insert Into Shared Global Temp Table  ID = ts,tn     --> declared global temporary table
                                                          will be created by multiple subagents
                                                          in parallel
Insert Into Sorted Global Temp Table  ID = ts,tn     --> sorted declared global temporary table
Insert Into Sorted Shared Global Temp Table  ID = ts,tn     --> sorted declared global temporary
                                                                table will be created by
                                                                multiple subagents in parallel
```

ID는 임시 테이블을 참조할 때 편의 상 db2expln에서 지정하는 ID입니다. 이 ID 앞에는 문자 ‘t’가 붙어서 테이블이 임시 테이블임을 표시합니다.

이러한 각 명령문 뒤에는 다음과 같은 내용이 표시됩니다.

```
#Columns = n
```

임시 테이블에 삽입되는 각 행에 있는 컬럼 수를 표시합니다.

정렬된 임시 테이블

다음과 같은 연산에서 정렬된 임시 테이블이 나타날 수 있습니다.

- ORDER BY
- DISTINCT
- GROUP BY
- 병합 조인
- ‘= ANY’ 서브쿼리
- ‘<> ALL’ 서브쿼리
- INTERSECT 또는 EXCEPT
- UNION(ALL 키워드를 사용하지 않음)

정렬된 임시 테이블과 연관된 명령문 수가 db2expln 명령 출력에 표시될 수 있습니다.

- 다음 명령문은 정렬에서 사용되는 키 컬럼의 수를 표시합니다.

  ```
  #Sort Key Columns = n
  ```

정렬 키의 각 컬럼마다 다음의 라인 중 하나가 표시됩니다.

```
Key n: column_name (Ascending)
Key n: column_name (Descending)
Key n: (Ascending)
Key n: (Descending)
```

- 다음 명령문은 런타임에 최적의 정렬 힙을 할당할 수 있도록 추정 행 수 및 행 *크기*를 제공합니다.

```
Sortheap Allocation Parameters:
|  #Rows     = n
|  Row Width = n
```

- 다음 명령문은 정렬된 결과의 첫 번째 행만 필요한 경우 표시됩니다.

```
Sort Limited To Estimated Row Count
```

- 대칭형 멀티프로세서(SMP) 환경에서 수행되는 정렬의 경우 수행할 정렬의 유형은 다음 명령문 중 하나로 표시됩니다.

```
Use Partitioned Sort
Use Shared Sort
Use Replicated Sort
Use Round-Robin Sort
```

- 다음 명령문은 정렬된 결과가 정렬 힙에 남아 있는지 여부를 표시합니다.

```
Piped
Not Piped
```

  파이프 정렬이 표시된 경우 데이터베이스 관리 프로그램은 정렬된 출력을 다른 임시 테이블에 배치하지 않고 메모리에 보존합니다.

- 다음 명령문은 정렬 조작 중 중복 값이 제거됨을 표시합니다.

```
Duplicate Elimination
```

- 정렬 조작 중 집계를 수행 중인 경우 다음 명령문 중 하나가 표시됩니다.

```
Partial Aggregation
Intermediate Aggregation
Buffered Partial Aggregation
Buffered Intermediate Aggregation
```

임시 테이블 완료

테이블 액세스 범위 내에서 임시 테이블이 작성될 때마다 완료 명령문이 표시됩니다. 이 명령문은 다음 중 하나입니다.

```
Temp Table Completion  ID = tn
Shared Temp Table Completion  ID = tn
Sorted Temp Table Completion  ID = tn
Sorted Shared Temp Table Completion  ID = tn
```

테이블 함수

테이블 함수는 명령문에 테이블의 형식으로 데이터를 리턴하는 사용자 정의 함수(UDF)입니다. 테이블 함수는 함수의 속성에 대해 자세히 설명하는 다음 명령문으로 표시됩니다. 특정 이름은 호출되는 테이블 함수를 고유하게 식별합니다.

```
Access User Defined Table Function
|  Name = schema.funcname
|  Specific Name = specificname
|  SQL Access Level = accesslevel
|  Language = lang
|  Parameter Style = parmstyle
|  Fenced                       Not Deterministic
|  Called on NULL Input         Disallow Parallel
|  Not Federated                Not Threadsafe
```

조인 정보:

db2expln 명령의 출력에는 Explain문의 조인에 대한 정보가 포함될 수 있습니다.

조인이 수행될 때마다 다음 명령문 중 하나가 표시됩니다.

```
Hash Join
Merge Join
Nested Loop Join
```

왼쪽 외부 조인은 다음 명령문 중 하나로 표시됩니다.

```
Left Outer Hash Join
Left Outer Merge Join
Left Outer Nested Loop Join
```

병합 또는 중첩된 루프 조인의 경우 조인의 외부 테이블은 이전 액세스 명령문에 참조된 테이블입니다(출력에 표시됨). 조인의 내부 테이블은 조인 명령문의 범위 내에 포함된 액세스 명령문에 참조된 테이블입니다. 해시 조인의 경우 액세스 명령문은 반대가 됩니다. 외부 테이블이 조인 범위 내에 포함되고 내부 테이블이 조인 이전에 표시됩니다.

해시 또는 병합 조인의 경우 다음의 추가 명령문이 표시될 수 있습니다.

- `Early Out: Single Match Per Outer Row`

  조인이 내부 테이블의 행이 외부 테이블의 현재 행과 일치하는지 판별해야 하는 경우가 있습니다.

- ```
  Residual Predicate(s)
  |  #Predicates = n
  ```

  조인이 완료된 후 술어를 적용할 수 있습니다. 이 명령문은 적용되는 술어의 수를 표시합니다.

해시 조인의 경우 다음의 추가 명령문이 표시될 수 있습니다.

- `Process Hash Table For Join`
  ```

해시 테이블은 내부 테이블에서 빌드됩니다. 내부 테이블에 액세스하는 중 해시 테이블 빌드가 술어로 푸시다운된 경우 이 명령문이 표시됩니다.

- `Process Probe Table For Hash Join`

  외부 테이블에 액세스하는 중에 프로브 테이블을 빌드하여 조인 성능을 향상시킬 수 있습니다. 외부 테이블에 액세스하는 중에 프로브 테이블이 빌드된 경우 이 명령문이 표시됩니다.

- `Estimated Build Size: n`

  이 명령문은 해시 테이블을 빌드할 때 필요한 추정 바이트 수를 표시합니다.

- `Estimated Probe Size: n`

  이 명령문은 프로브 테이블을 빌드할 때 필요한 추정 바이트 수를 표시합니다.

중첩된 루프 조인의 경우 다음 명령문이 조인 명령문 바로 뒤에 표시될 수 있습니다.

```
Piped Inner
```

이 명령문은 조인의 내부 테이블이 다른 일련의 연산으로 인한 결과임을 표시합니다. 이 테이블을 *복합 내부*라고도 합니다.

조인에 여러 개의 테이블이 관련된 경우 Explain 단계는 맨 위부터 맨 아래로 읽어야 합니다. 예를 들어, Explain 출력에 다음의 플로우가 있다고 가정해봅니다.

```
Access ..... W
Join
|  Access ..... X
Join
|  Access ..... Y
Join
|  Access ..... Z
```

실행 단계는 다음과 같습니다.

1. W 테이블에서 규정 행을 가져오십시오.
2. W의 행을 X 테이블의 다음 행과 조인하고 결과 P1을 호출하십시오(부분 조인 결과 번호 1의 경우).
3. P1을 Y 테이블의 다음 행과 조인하여 P2를 작성하십시오.
4. P2를 Z 테이블의 다음 행과 조인하여 하나의 전체 결과 행을 작성하십시오.
5. Z에 추가 행이 있는 경우 4단계로 이동하십시오.
6. Y에 추가 행이 있는 경우 3단계로 이동하십시오.
7. X에 추가 행이 있는 경우 2단계로 이동하십시오.
8. W에 추가 행이 있는 경우 1단계로 이동하십시오.

데이터 스트림 정보:

액세스 플랜에서는 하나의 일련의 조작에서 다른 일련의 조작으로 데이터 작성 및 플로우를 제어해야 합니다. 데이터 스트림 개념을 통해 액세스 플랜 내의 조작 그룹을 한 단위로 제어할 수 있습니다.

데이터 스트림의 시작은 db2expln 출력에서 다음 명령문으로 표시됩니다.

```
Data Stream n
```

여기서 *n*은 참조하기 쉽게 db2expln에서 지정한 고유 ID입니다.

데이터 스트림의 끝은 다음으로 표시합니다.

```
End of Data Stream n
```

이러한 명령문 사이의 모든 연산은 동일한 데이터 스트림의 일부분으로 간주됩니다.

데이터 스트림에는 여러 특성이 있으며 하나 이상의 명령문이 초기 데이터 스트림 명령문을 따라 이러한 특성에 대해 설명할 수 있습니다.

- 데이터 스트림의 조작이 액세스 플랜에서 이전에 생성된 값에 따라 다른 경우 데이터 스트림은 다음과 같이 표시됩니다.

  ```
  Correlated
  ```

- 정렬된 임시 테이블과 비슷하게 다음 명령문은 데이터 스트림의 결과가 메모리에 보존되는지 여부를 표시합니다.

  ```
  Piped
  Not Piped
  ```

  실행 시 메모리가 부족한 경우 디스크에 파이프 데이터 스트림이 기록될 수 있습니다. 액세스 플랜은 두 가지 가능성에 모두 대비합니다.

- 다음 명령문은 이 데이터 스트림에서 단일 레코드만 필요함을 표시합니다.

  ```
  Single Record
  ```

데이터 스트림에 액세스할 때 다음 명령문이 출력에 표시됩니다.

```
Access Data Stream n
```

삽입, 갱신 및 삭제 정보:

INSERT, UPDATE 또는 DELETE문의 Explain 텍스트는 특별한 설명이 필요하지 않습니다.

db2expln 출력에서 이러한 SQL 연산의 명령문 텍스트는 다음과 같습니다.

```
Insert: Table Name = schema.name ID = ts,n
Update: Table Name = schema.name ID = ts,n
Delete: Table Name = schema.name ID = ts,n
Insert: Hierarchy Table Name = schema.name ID = ts,n
Update: Hierarchy Table Name = schema.name ID = ts,n
Delete: Hierarchy Table Name = schema.name ID = ts,n
Insert: Materialized Query Table = schema.name ID = ts,n
```

```
Update: Materialized Query Table = schema.name ID = ts,n
Delete: Materialized Query Table = schema.name ID = ts,n
Insert: Global Temporary Table ID = ts, tn
Update: Global Temporary Table ID = ts, tn
Delete: Global Temporary Table ID = ts, tn
```

블록 및 행 ID 준비 정보:

일부 액세스 플랜의 경우 테이블에 액세스하기 전에 규정 행 및 블록 ID를 정렬하고 중복을 제거하거나(Index ORing의 경우) 액세스 중인 모든 인덱스에 표시되는 ID를 판별하는(Index ANDing) 기술을 사용하면 효율적입니다.

Explain 출력에 표시되는 ID 준비 정보의 세 가지 기본 용도가 있습니다.

- 다음 명령문 중 하나는 Index ORing을 사용하여 규정 ID 목록을 준비했음을 표시합니다.

  ```
  Index ORing Preparation
  Block Index ORing Preparation
  ```

  *Index ORing*은 여러 인덱스에 액세스하여 인덱스에 표시되는 구별 ID를 포함한 결과를 결합하는 기술을 나타냅니다. OR 키워드로 술어를 연결하거나 IN 술어가 있는 경우 옵티마이저는 Index ORing으로 간주합니다.
- 다음 명령문 중 하나는 리스트 프리페치 중 입력 데이터를 사용할 준비가 되었음을 표시합니다.

  ```
  List Prefetch Preparation
  Block List Prefetch RID Preparation
  ```
- *Index ANDing*은 여러 인덱스에 액세스하여 액세스한 모든 인덱스에 표시되는 ID를 포함한 결과를 결합하는 기술을 나타냅니다. Index ANDing은 다음 중 한 명령문으로 시작됩니다.

  ```
  Index ANDing
  Block Index ANDing
  ```

  옵티마이저가 결과 세트의 *크기*를 추정한 경우 다음 명령문을 사용하여 추정 값이 표시됩니다.

  ```
  Optimizer Estimate of Set Size: n
  ```

  Index ANDing은 연산 프로세스 ID를 필터하고 비트 필터 기술을 사용하여 액세스한 모든 인덱스에 표시되는 ID를 판별합니다. 다음 명령문은 Index ANDing을 위해 ID가 처리되었음을 표시합니다.

  ```
  Index ANDing Bitmap Build Using Row IDs
  Index ANDing Bitmap Probe Using Row IDs
  Index ANDing Bitmap Build and Probe Using Row IDs
  Block Index ANDing Bitmap Build Using Block IDs
  Block Index ANDing Bitmap Build and Probe Using Block IDs
  Block Index ANDing Bitmap Build and Probe Using Row IDs
  ```

```
Block Index ANDing Bitmap Probe Using Block IDs and Build Using Row IDs
Block Index ANDing Bitmap Probe Using Block IDs
Block Index ANDing Bitmap Probe Using Row IDs
```

옵티마이저가 비트맵에 대한 결과 세트의 크기를 추정한 경우 다음 명령문을 사용하여 추정 값이 표시됩니다.

```
Optimizer Estimate of Set Size: n
```

모든 유형의 ID 준비를 위해 리스트 프리페치를 수행할 수 있는 경우 다음 명령문을 사용하여 표시됩니다.

```
Prefetch: Enabled
```

집계 정보:

집계는 SQL문에 술어로 표시된 기준을 충족시키는 행에서 수행됩니다.

집계 함수가 실행되면 다음 명령문 중 하나가 db2expln 출력에 표시됩니다.

```
Aggregation
Predicate Aggregation
Partial Aggregation
Partial Predicate Aggregation
Intermediate Aggregation
Intermediate Predicate Aggregation
Final Aggregation
Final Predicate Aggregation
```

Predicate aggregation은 데이터에 액세스할 때 집계 연산이 술어로 처리되었음을 의미합니다.

집계 명령문 뒤에는 수행된 집계 함수의 유형을 식별하는 다른 명령문이 표시됩니다.

```
Group By
Column Function(s)
Single Record
```

특정 컬럼 함수가 원래의 SQL문에서 파생될 수 있습니다. 인덱스에서 단일 레코드가 페치되어 MIN 또는 MAX 연산을 충족시킵니다.

Predicate aggregation이 수행된 경우 집계 완료 조작과 이에 해당하는 출력이 표시됩니다.

```
Aggregation Completion
Partial Aggregation Completion
Intermediate Aggregation Completion
Final Aggregation Completion
```

병렬 처리 정보:

SQL문을 병렬식으로 실행하려면(파티션 내 또는 파티션 간 병렬 처리 사용) 일부 특수 액세스 플랜 연산이 필요합니다.

- 파티션 내 병렬 플랜을 실행하는 경우 플랜의 부분들이 여러 서브에이전트를 사용하여 동시에 실행됩니다. 이러한 서브에이전트의 작성은 db2expln 명령의 출력에 명령문으로 표시됩니다.

  ```
  Process Using n Subagents
  ```

- 파티션 간 병렬 플랜을 실행하는 경우 섹션이 여러 서브섹션으로 구분됩니다. 각 서브섹션을 하나 이상의 데이터베이스 파티션으로 보내어 실행합니다. 중요한 서브섹션은 *코디네이터 서브섹션*입니다. 코디네이터 서브섹션은 모든 플랜에서 첫 번째 서브섹션입니다. 처음 제어 권한을 획득하고 다른 서브섹션을 분산시키고 호출 응용프로그램으로 결과를 리턴할 책임이 있습니다.
  - 서브섹션 분배는 다음 명령문으로 표시됩니다.

    ```
    Distribute Subsection #n
    ```

  - 다음 명령문은 컬럼의 값에 따라 데이터베이스 파티션 그룹의 데이터베이스 파티션으로 서브섹션이 전송됨을 표시합니다.

    ```
    Directed by Hash
    |  #Columns = n
    |  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
    ```

  - 다음 명령문은 서브섹션이 사전 판별된 데이터베이스 파티션으로 전송됨을 표시합니다(명령문이 DBPARTITIONNUM() 스칼라 함수를 사용하는 경우 일반적인 상황).

    ```
    Directed by Node Number
    ```

  - 다음 명령문은 데이터베이스 파티션 그룹의 사전 판별된 데이터베이스 파티션 번호에 해당하는 데이터베이스 파티션으로 서브섹션이 전송됨을 표시합니다(명령문이 HASHEDVALUE 스칼라 함수를 사용하는 경우 일반적인 상황).

    ```
    Directed by Partition Number
    |  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
    ```

  - 다음 명령문은 응용프로그램의 커서에 현재 행을 제공한 데이터베이스 파티션으로 서브섹션이 전송됨을 표시합니다.

    ```
    Directed by Position
    ```

  - 다음 명령문은 명령문 컴파일 시 판별된 하나의 데이터베이스 파티션만 서브섹션을 받게 됨을 표시합니다.

    ```
    Directed to Single Node
    |  Node Number = n
    ```

  - 다음 명령문 중 하나는 코디네이터 데이터베이스 파티션에서 서브섹션이 실행됨을 표시합니다.

    ```
    Directed to Application Coordinator Node
    Directed to Local Coordinator Node
    ```

  - 다음 명령문은 나열된 모든 데이터베이스 파티션으로 서브섹션이 전송됨을 표시합니다.

```
Broadcast to Node List
|  Nodes = n1, n2, n3, ...
```

– 다음 명령문은 명령문 실행 시 판별된 하나의 데이터베이스 파티션만 서브섹션을 받게 됨을 표시합니다.

```
Directed to Any Node
```

- 테이블 큐를 사용하여 파티션된 데이터베이스 환경의 서브섹션들 간 또는 대칭형 멀티프로세서(SMP) 환경의 서브에이전트들 간에 데이터를 이동합니다.
  – 다음 명령문은 테이블 큐에 데이터를 삽입 중임을 표시합니다.

```
Insert Into Synchronous Table Queue  ID = qn
Insert Into Asynchronous Table Queue  ID = qn
Insert Into Synchronous Local Table Queue  ID = qn
Insert Into Asynchronous Local Table Queue  ID = qn
```

  – 데이터베이스 파티션 테이블 큐의 경우 테이블 큐에 삽입된 행의 대상은 다음 명령문 중 하나로 설명됩니다.

    각 행은 코디네이터 데이터베이스 파티션으로 전송됩니다.

```
Broadcast to Coordinator Node
```

    각 행은 지정된 서브섹션을 실행 중인 모든 데이터베이스 파티션으로 전송됩니다.

```
Broadcast to All Nodes of Subsection n
```

    각 행은 행의 값에 따라 다른 데이터베이스 파티션으로 전송됩니다.

```
Hash to Specific Node
```

    각 행은 명령문 실행 중 판별된 데이터베이스 파티션으로 전송됩니다.

```
Send to Specific Node
```

    각 행은 무작위로 판별된 데이터베이스 파티션으로 전송됩니다.

```
Send to Random Node
```

  – 데이터베이스 파티션 테이블 큐가 일부 행을 임시 테이블로 오버플로우해야 하는 경우가 있습니다. 이러한 가능성은 다음 명령문으로 식별됩니다.

```
Rows Can Overflow to Temporary Table
```

  – 테이블 큐에 행을 삽입하기 위한 푸시다운 연산이 포함된 테이블 액세스 이후 즉시 전송할 수 없는 행을 처리하는 "완료" 명령문이 있습니다. 이 경우 다음 라인 중 하나가 표시됩니다.

```
Insert Into Synchronous Table Queue Completion  ID = qn
Insert Into Asynchronous Table Queue Completion  ID = qn
Insert Into Synchronous Local Table Queue Completion  ID = qn
Insert Into Asynchronous Local Table Queue Completion  ID = qn
```

  – 다음 명령문은 테이블 큐에서 데이터를 검색 중임을 표시합니다.

```
Access Table Queue  ID = qn
Access Local Table Queue  ID = qn
```

이러한 명령문 뒤에는 항상 검색 중인 컬럼 수가 표시됩니다.

```
#Columns = n
```

– 테이블 큐가 수신 측에서 행을 정렬하는 경우 다음 명령문 중 하나가 표시됩니다.

```
Output Sorted
Output Sorted and Unique
```

이러한 명령문 뒤에는 정렬 조작에 사용 중인 키의 수가 표시됩니다.

```
#Key Columns = n
```

정렬 키의 각 컬럼마다 다음 명령문 중 하나가 표시됩니다.

```
Key n: (Ascending)
Key n: (Descending)
```

– 테이블 큐의 수신 측에 있는 행에 술어가 적용되는 경우 다음 명령문으로 표시됩니다.

```
Residual Predicate(s)
|  #Predicates = n
```

- 파티션된 데이터베이스 환경의 일부 서브섹션은 서브섹션의 시작 부분까지 명시적으로 루프백하며 다음 명령문으로 표시됩니다.

```
Jump Back to Start of Subsection
```

페더레이티드 쿼리 정보:

페더레이티드 데이터베이스에서 SQL문을 실행하려면 다른 데이터 소스에서 명령문 부분들을 수행할 수 있는 기능이 필요합니다.

db2expln 명령의 다음과 같은 출력은 데이터 소스를 읽을 것임을 표시합니다.

```
Ship Distributed Subquery #n
|   #Columns = n
```

분산 서브쿼리에서 리턴되는 데이터에 술어가 적용되는 경우 적용되는 술어 수는 다음 명령문으로 표시됩니다.

```
Residual Predicate(s)
|   #Predicates = n
```

데이터 소스에서 발생하는 삽입, 갱신 또는 삭제 조작은 다음 명령문 중 하나로 표시됩니다.

```
Ship Distributed Insert #n
Ship Distributed Update #n
Ship Distributed Delete #n
```

데이터 소스에서 테이블을 명시적으로 잠근 경우 다음 명령문이 표시됩니다.

```
Ship Distributed Lock Table #n
```

데이터 소스에 대한 DDL(Data Definition Language)문은 두 부분으로 분할됩니다. 데이터 소스에서 호출되는 부분은 다음 명령문으로 표시됩니다.

```
Ship Distributed DDL Statement #n
```

페더레이티드 서버가 파티션된 데이터베이스인 경우 DDL문의 일부분을 카탈로그 데이터베이스 파티션에서 실행해야 합니다. 다음 명령문으로 표시됩니다.

```
Distributed DDL Statement #n Completion
```

각 분산 부속 명령문에 대한 세부사항은 개별적으로 표시됩니다.

- 서브쿼리의 데이터 소스는 다음 명령문 중 하나로 표시됩니다.

  ```
  Server: server_name (type, version)
  Server: server_name (type)
  Server: server_name
  ```

- 데이터 소스가 관계형인 경우 부속 명령문의 SQL은 다음과 같이 표시됩니다.

  ```
  SQL Statement:
       statement
  ```

  비관계형 데이터 소스는 다음과 같이 표시됩니다.

  ```
  Non-Relational Data Source
  ```

- 부속 명령문에 참조된 별칭은 다음과 같이 나열됩니다.

  ```
  Nicknames Referenced:
     schema.nickname ID = n
  ```

  데이터 소스가 관계형인 경우 별칭의 기본 테이블은 다음과 같이 표시됩니다.

  ```
  Base = baseschema.basetable
  ```

  데이터 소스가 비관계형인 경우 별칭의 소스 파일은 다음과 같이 표시됩니다.

  ```
  Source File = filename
  ```

- 부속 명령문을 실행하기 전에 페더레이티드 서버에서 데이터 소스로 값이 전달된 경우 값의 수는 다음 명령문으로 표시됩니다.

  ```
  #Input Columns: n
  ```

- 부속 명령문이 실행된 후 데이터 소스에서 페더레이티드 서버로 값이 전달된 경우 값의 수는 다음 명령문으로 표시됩니다.

  ```
  #Output Columns: n
  ```

기타 Explain 정보:

db2expln 명령의 출력에는 쉽게 분류할 수 없는 유용한 추가 정보가 들어 있습니다.

- DDL(Data Definition Language)문의 섹션은 다음 명령문을 사용하여 출력에 표시됩니다.

  ```
  DDL Statement
  ```

DDL문에 추가 Explain 출력이 제공되지 않습니다.

- CURRENT EXPLAIN SNAPSHOT과 같이 갱신 가능한 특수 레지스터와 관련된 SET문의 섹션은 다음 명령문을 사용하여 출력에 표시됩니다.

  ```
  SET Statement
  ```

  SET문에 추가 Explain 출력이 제공되지 않습니다.

- SQL문에 DISTINCT절이 포함된 경우 다음 명령문이 출력에 표시될 수 있습니다.

  ```
  Distinct Filter  #Columns = n
  ```

  여기서 *n*은 구별 행을 확보하는 데 관련된 컬럼 수입니다. 구별 행 값을 검색하려면 먼저 행을 정렬하여 중복을 제거해야 합니다. 다음의 경우와 같이 데이터베이스 관리 프로그램이 중복을 명시적으로 제거할 필요가 없는 경우 이 명령문이 표시되지 않습니다.

  – 고유 인덱스가 존재하며 인덱스 키의 모든 컬럼이 DISTINCT 연산의 일부분입니다.

  – 정렬 중 중복을 제거할 수 있습니다.

- 다음 연산이 특정 레코드 ID에 따라 다른 경우 다음 명령문이 표시됩니다.

  ```
  Positioned Operation
  ```

  위치 지정된 연산이 페더레이티드 데이터 소스에 대한 연산인 경우 명령문은 다음과 같습니다.

  ```
  Distributed Positioned Operation
  ```

  이 명령문은 WHERE CURRENT OF 구문을 사용하는 SQL문의 경우에 표시됩니다.

- 결과에 적용되어야 하지만 다른 연산의 일부분으로 적용할 수 없는 술어가 있는 경우 다음 명령문이 표시됩니다.

  ```
  Residual Predicate Application
  |  #Predicates = n
  ```

- SQL문에 UNION 연산자가 포함된 경우 다음 명령문이 표시됩니다.

  ```
  UNION
  ```

- 후속 연산에서 사용할 행 값을 작성하기 위해서만 액세스 플랜에 연산이 있는 경우 다음 명령문이 표시됩니다.

  ```
  Table Constructor
  |  n-Row(s)
  ```

  테이블 컨스트럭터는 한 세트의 값을 후속 연산에 전달되는 일련의 행으로 변환하는 데 사용할 수 있습니다. 그 다음 행에 대해 테이블 컨스트럭터가 프롬프트되는 경우 다음 명령문이 표시됩니다.

  ```
  Access Table Constructor
  ```

- 특정 조건에서만 처리되는 연산이 있는 경우 다음 명령문이 표시됩니다.

```
Conditional Evaluation
|  Condition #n:
|  #Predicates = n
|  Action #n:
```

  조건부 평가를 사용하여 CASE문과 같은 활동이나 참조 무결성 제한조건 또는 트리거와 같은 내부 메커니즘을 구현합니다. 조치가 표시되지 않은 경우 조건이 참이면 데이터 조작 연산만 처리됩니다.

- 액세스 플랜에서 ALL, ANY 또는 EXISTS 서브쿼리가 처리되는 경우 다음 명령문 중 하나가 표시됩니다.

```
ANY/ALL Subquery
EXISTS Subquery
EXISTS SINGLE Subquery
```

- 특정 갱신 또는 삭제 조작 이전에 테이블에서 특정 행의 위치를 설정해야 합니다. 다음 명령문으로 표시됩니다.

```
Establish Row Position
```

- 롤아웃 최적화를 위해 규정되는 다차원 클러스터링 테이블에서 삭제 조작을 위해 다음 명령문 중 하나가 표시됩니다.

```
CELL DELETE with deferred cleanup
CELL DELETE with immediate cleanup
```

- 응용프로그램으로 행을 리턴하는 경우 다음 명령문이 표시됩니다.

```
Return Data to Application
|  #Columns = n
```

  테이블 액세스로 조작이 푸시다운된 경우 완료 단계 명령문이 출력에 표시됩니다.

```
Return Data Completion
```

- 스토어드 프로시저가 호출되는 경우 다음 명령문이 표시됩니다.

```
Call Stored Procedure
|   Name = schema.funcname
|   Specific Name = specificname
|   SQL Access Level = accesslevel
|   Language = lang
|   Parameter Style = parmstyle
|   Expected Result Sets = n
|   Fenced                      Not Deterministic
|   Called on NULL Input        Disallow Parallel
|   Not Federated               Not Threadsafe
```

- 하나 이상의 대형 오브젝트(LOB) 로케이터가 해제되는 경우 다음 명령문이 표시됩니다.

```
Free LOB Locators
```

쿼리 액세스 플랜 최적화

명령문 집중기가 컴파일 오버헤드를 줄임

명령문 집중기는 유사하지만 동일하지는 않은 SQL문이 동일한 액세스 플랜을 공유할 수 있도록 데이터베이스 서버에서 동적 SQL문을 수정합니다.

온라인 트랜잭션 처리(OLTP)에서는, 단순 명령문이 다른 리터럴 값을 사용하여 반복적으로 생성될 수도 있습니다. 이러한 워크로드에서 명령문 재컴파일 비용이 상당한 오버헤드를 추가시킬 수 있습니다. 명령문 집중기는 리터럴 값에 상관없이 컴파일된 명령문이 재사용될 수 있도록 하여 이러한 오버헤드를 예방합니다.

명령문 집중기는 디폴트로 사용되지 않습니다. **stmt_conc** 데이터베이스 구성 매개변수를 LITERALS로 설정하여 데이터베이스의 모든 동적문에 대해 사용 가능하게 할 수 있습니다. 처음 100,000개의 리터럴만 교체됩니다. 나머지 리터럴은 리터럴로 남아 있습니다.

명령문 집중기는 수신 동적 SQL문을 수정하여 성능을 개선합니다. 명령문 집중기에 적합한 워크로드에서 수신 SQL문의 수정과 관련된 오버헤드는 이미 패키지 캐시에 있는 명령문을 재사용하여 인식되는 절약과 비교할 때 중요하지 않습니다.

명령문 집중의 결과로 동적문이 수정되는 경우, 원래 명령문과 수정된 명령문이 둘 다 Explain 출력에 표시됩니다. 이벤트 모니터 논리적 모니터 요소와 MON_GET_ACTIVITY_DETAILS 테이블 함수의 출력은 명령문 집중기가 원래 명령문 텍스트를 수정한 경우 원래 명령문을 표시합니다. 다른 모니터 인터페이스는 수정된 명령문 텍스트만 표시합니다.

다음 예제를 고려하십시오. 여기서 **stmt_conc** 데이터베이스 구성 매개변수는 LITERALS로 설정되었으며 다음 두 명령문이 실행됩니다.

```
select firstnme, lastname from employee where empno='000020'
select firstnme, lastname from employee where empno='000070'
```

이 명령문은 패키지 캐시에서 동일한 항목을 공유하며 해당 항목은 다음 명령문을 사용합니다.

```
select firstnme, lastname from employee where empno=:L0000000000
```

데이터 서버는 원래 명령문에 사용된 리터럴에 따라 `:L0000000000`의 값('000020' 또는 '000070')을 제공합니다.

명령문 집중은 명령문 텍스트를 변경하므로 액세스 플랜 선택에 영향을 미칩니다. 패키지 캐시의 유사한 명령문에 유사한 플랜이 있는 경우 명령문 집중기를 사용해야 합니다. 명령문에 다른 리터럴 값으로 인해 상당히 다른 액세스 플랜이 생성되는 경우 해당 명령문의 경우에는 명령문 집중기를 사용 가능하게 하지 않아야 합니다.

액세스 플랜 재사용

패키지의 정적 SQL문에 대해 선택된 액세스 플랜이 몇 개의 바인드 또는 리바인드 조작에서 기존 액세스 플랜과 동일하거나 아주 유사한 상태를 유지하도록 요청할 수 있습니다.

액세스 플랜은 중요한 플랜 변경사항이 명시적 승인 없이 발생하지 않도록 할 수 있습니다. 이는 쿼리가 잠재적 액세스 플랜 향상 기능을 통해 이득을 보지 못함을 의미할 수 있지만 액세스 플랜 재사용이 제공하는 제어는 준비가 되었을 때 해당 향상 기능을 테스트하고 구현할 수 있는 능력을 제공합니다. 그 때까지는 안정적이면서 예측 가능한 성능을 위해 기존 액세스 플랜을 계속 사용할 수 있습니다.

ALTER PACKAGE문을 통해 또는 BIND, REBIND나 PRECOMPILE 명령에서 APREUSE 옵션을 사용하여 액세스 플랜을 사용할 수 있게 하십시오. 액세스 플랜을 재사용하게 되는 패키지는 SYSCAT.PACKAGES 카탈로그 뷰의 APREUSE 컬럼에 Y 값을 갖습니다.

ALTER_ROUTINE_PACKAGE 프로시저는 컴파일된 SQL 오브젝트(예: SQL 프로시저)에 액세스 플랜 재사용을 사용할 수 있게 하는 편리한 방법입니다. 그러나 오브젝트가 리바인드되기 전에 삭제되므로 컴파일된 오브젝트 유효성 다시 확인 중에는 액세스 플랜을 재사용할 수 없습니다. 이 경우, APREUSE는 다음 번에 패키지를 바인드하거나 리바인드할 때만 적용됩니다.

액세스 플랜은 스키마 및 컴파일 환경을 최소로 변경하는 경우 가장 효과적입니다. 대대적으로 변경하는 경우, 이전 액세스 플랜을 재작성하지 못할 수도 있습니다. 이러한 대대적인 변경의 예로는 액세스 플랜에서 사용되고 있는 인덱스 삭제 또는 다른 최적화 레벨에서 SQL문 재컴파일을 들 수 있습니다. 또한 쿼리 컴파일러의 명령문 분석을 대대적으로 변경하면 이전 액세스 플랜을 더 이상 사용하지 못할 수 있습니다.

액세스 플랜 재사용을 최적화 지침과 조합할 수 있습니다. 명령문 레벨 지침은 지침이 적용되는 정적 SQL문에 대한 액세스 플랜 재사용에 우선합니다. 지정된 일반 최적화 지침과 충돌하지 않으면 명령문 레벨 지침이 없는 정적 명령문에 대한 액세스 플랜을 재사용할 수 있습니다. 지침이 비어 있는 명령문 프로파일을 사용하여 특정 명령문에 액세스 플랜 재사용을 사용할 수 없게 하고 패키지의 기타 정적 명령문에는 플랜 재사용을 사용할 수 있게 할 수 있습니다.

주: 버전 9.7 이전 릴리스에서 작성한 패키지의 액세스 플랜은 재사용할 수 없습니다.

액세스 플랜을 재사용할 수 없는 경우, 컴파일은 계속되지만 액세스 플랜 재사용 시도가 실패한 이유를 표시하는 이유 코드와 함께 경고(SQL20516W)가 리턴됩니다. Explain 기능을 통해 사용 가능한 진단 메시지에 추가 정보가 제공되는 경우도 있습니다.

최적화 클래스

SQL 또는 XQuery문을 컴파일할 때, 옵티마이저가 해당 명령문에 대한 가장 효율적인 액세스 플랜을 선택하는 방법을 판별하는 최적화 클래스를 지정할 수 있습니다.

최적화 클래스는 쿼리의 컴파일 동안 고려되는 최적화 전략의 수와 유형에서 서로 다릅니다. 최적화 기술을 개별적으로 지정하여 쿼리에 대한 런타임 성능을 향상시킬 수 있지만, 지정하는 최적화 기술이 많을수록 쿼리 컴파일에 필요한 시간 및 시스템 자원이 더 많아집니다.

SQL 또는 XQuery문을 컴파일 할 때 다음 최적화 클래스 중 하나를 지정할 수 있습니다.

0 이 클래스는 액세스 플랜을 생성할 때 옵티마이저가 최소 최적화를 사용하도록 지시하고, 다음 특성을 갖습니다.

- 자주 사용되는 값 통계는 옵티마이저에서 고려되지 않습니다.
- 기본 쿼리 재작성 규칙만 적용됩니다.
- 그리디(Greedy) 조인 열거가 사용됩니다.
- 중첩된 루프 조인 및 인덱스 스캔 액세스 메소드만 사용 가능합니다.
- 생성된 액세스 메소드에서 리스트 프리페치가 사용되지 않습니다.
- 스타 조인 전략이 고려되지 않습니다.

이 클래스는 가능한 최저 쿼리 컴파일 오버헤드가 요구되는 환경에서만 사용되어야 합니다. 쿼리 최적화 클래스 0은 잘 인덱스화된 테이블에 액세스하는 매우 단순한 동적 SQL 또는 XQuery문으로 전적으로 구성된 응용프로그램에 적합합니다.

1 이 최적화 클래스는 다음 특성을 갖습니다.

- 자주 사용되는 값 통계는 옵티마이저에서 고려되지 않습니다.
- 쿼리 재작성 규칙의 서브세트만 적용됩니다.
- 그리디(Greedy) 조인 열거가 사용됩니다.
- 생성된 액세스 메소드에서 리스트 프리페치가 사용되지 않습니다.

최적화 클래스 1은 병합 스캔 조인과 테이블 스캔도 사용 가능하다는 점을 제외하고는 클래스 0과 비슷합니다.

2 이 클래스는 옵티마이저가 클래스 3 이상보다 훨씬 더 낮은 복합 쿼리에 대한 컴파일 비용을 유지하면서, 클래스 1보다 훨씬 더 높은 최적화 등급을 사용하도록 지시합니다. 이 최적화 클래스는 다음 특성을 갖습니다.

- 자주 사용되는 값 및 Quantile 통계를 포함하여 모든 사용 가능한 통계가 사용됩니다.
- 매우 드문 경우에만 적용 가능한 계산 집약적 규칙을 제외하고, 모든 쿼리 재작성 규칙(구체화된 쿼리 테이블 경로지정 포함)이 적용됩니다.

- 그리디(Greedy) 조인 열거가 사용됩니다.
- 리스트 프리페치 및 구체화된 쿼리 테이블 경로지정을 포함한 광범위한 액세스 메소드가 고려됩니다.
- 적용 가능한 경우, 스타 조인 전략이 고려됩니다.

최적화 클래스 2는 동적 프로그래밍 조인 열거 대신 그리디(Greedy) 조인 열거를 사용한다는 점을 제외하고는 클래스 5와 비슷합니다. 이 클래스는 복합 쿼리에 대해 더 적은 대체를 고려하고, 그에 따라 클래스 3 이상보다 더 적은 컴파일 시간을 소비하는 그리디(Greedy) 조인 열거 알고리즘을 사용하는 모든 클래스의 최대 최적화를 포함합니다. 클래스 2는 결정 지원 또는 온라인 분석 처리(OLAP) 환경의 매우 복잡한(복합) 쿼리에 권장됩니다. 이러한 환경에서는, 특정 쿼리가 정확히 동일한 방식으로 반복되지 않을 수 있으므로, 다음 쿼리 발생까지 액세스 플랜이 캐시에 남아있지 않을 수 있습니다.

3 이 클래스는 중간 정도의 최적화 양을 나타내며, z/OS용 DB2의 쿼리 최적화 특성과 가장 가깝게 일치합니다. 이 최적화 클래스는 다음 특성을 갖습니다.

- 사용 가능한 경우, 자주 사용되는 값 통계가 사용됩니다.
- 서브쿼리와 조인 간 변환을 포함하여 대부분의 쿼리 재작성 규칙이 적용됩니다.
- 다음과 함께 동적 조인 열거가 사용됩니다.
  - 복합 내부 테이블의 제한된 사용
  - 조회 테이블을 수반하는 스타 스키마에 대한 카테시안 곱의 제한된 사용
- 리스트 프리페치, 인덱스 ANDing 및 스타 조인을 포함한 광범위한 액세스 메소드가 고려됩니다.

이 클래스는 폭 넓은 범위의 응용프로그램에 적합하고, 4개 이상의 조인이 있는 쿼리에 대한 액세스 플랜을 향상시킵니다.

5 이 클래스는 액세스 플랜을 생성할 때 옵티마이저가 상당한 양의 최적화를 사용하도록 지시하고, 다음 특성을 갖습니다.

- 자주 사용되는 값 및 Quantile 통계를 포함하여 모든 사용 가능한 통계가 사용됩니다.
- 매우 드문 경우에만 적용 가능한 계산 집약적 규칙을 제외하고, 모든 쿼리 재작성 규칙(구체화된 쿼리 테이블 경로지정 포함)이 적용됩니다.
- 다음과 함께 동적 조인 열거가 사용됩니다.
  - 복합 내부 테이블의 제한된 사용
  - 조회 테이블을 수반하는 스타 스키마에 대한 카테시안 곱의 제한된 사용
- 리스트 프리페치, 인덱스 ANDing 및 구체화된 쿼리 테이블 경로지정을 포함한 광범위한 액세스 메소드가 고려됩니다.

최적화 클래스 5(디폴트)는 트랜잭션 처리와 복합 쿼리가 있는 혼합 환경에 대해 아주 훌륭한 선택입니다. 이 최적화 클래스는 가장 유용한 쿼리 변환 및 기타 쿼리 최적화 기술을 효율적으로 적용하도록 설계되었습니다.

옵티마이저에서 복합 동적 SQL 또는 XQuery문에 대한 추가 자원 및 처리 시간이 보장되지 않음을 감지할 경우, 최적화가 감소됩니다. 감소의 정도는 머신 크기 및 술어의 수에 따라 달라집니다. 쿼리 최적화의 양을 감소시킬 때, 옵티마이저는 일반적으로 적용되는 모든 쿼리 재작성 규칙을 계속 적용합니다. 그러나 그리디(Greedy) 조인 열거를 사용하고, 고려되는 액세스 플랜 조합의 수를 줄입니다.

7 이 클래스는 액세스 플랜을 생성할 때 옵티마이저가 상당한 양의 최적화를 사용하도록 지시합니다. 이 클래스는, 이 경우에 옵티마이저가 복합 동적 SQL 또는 XQuery문에 대한 쿼리 최적화 양을 줄이는 것을 고려하지 않는다는 점을 제외하고는 최적화 클래스 5와 비슷합니다.

9 이 클래스는 옵티마이저가 사용 가능한 모든 최적화 기술을 사용하도록 지시합니다. 여기에는 다음이 포함됩니다.

- 모든 사용 가능한 통계
- 모든 쿼리 재작성 규칙
- 카테시안 곱 및 무제한 복합 내부를 포함한, 조인 열거에 대한 모든 가능성
- 모든 액세스 메소드

이 클래스는 옵티마이저에 의해 고려되는 가능한 액세스 플랜의 수를 증가시킵니다. 이 클래스를 사용하여 보다 포괄적인 최적화가 대형 테이블을 사용하는 매우 복합적이고 매우 오랫동안 실행되는 쿼리에 대해 더 나은 액세스 플랜을 생성하는지 여부를 판별할 수도 있습니다. Explain 및 성능 측정을 사용하여 더 나은 플랜이 실제로 발견되었는지 검증하십시오.

최적화 클래스 선택:

최적화 클래스를 설정하면 최적화 기술을 명시적으로 지정할 수 있는 이점이 있습니다.

이 사항은 특히 다음과 같은 경우에 적용됩니다.

- 매우 작은 데이터베이스 또는 매우 간단한 동적 쿼리 관리
- 컴파일 시 데이터베이스 서버에 대한 메모리 제한사항 수용
- 예를 들어, 명령문 준비 중 쿼리 컴파일 시간 단축

대부분의 명령문은 기본 최적화 클래스 5를 사용하여 합리적인 자원량으로 충분하게 최적화할 수 있습니다. 쿼리 컴파일 시간 및 자원 사용량은 주로 쿼리의 복잡도 특히 조인 및 서브쿼리 수의 영향을 받습니다. 그러나 컴파일 시간 및 자원 사용량은 수행되는 최적화 양의 영향도 받습니다.

쿼리 최적화 클래스 1, 2, 3, 5 및 7은 모두 일반용으로 적합합니다. 쿼리 컴파일 시간을 추가로 줄여야 하는 경우에만 클래스 0을 참조하십시오. SQL 및 XQuery문이 매우 간단해집니다.

추가 정보: 장기 실행 쿼리를 분석하려면 db2batch로 쿼리를 실행하여 쿼리를 컴파일 및 실행하는 데 사용한 시간을 판별하십시오. 컴파일 시간을 초과한 경우 최적화 클래스를 줄이십시오. 실행 시간이 문제가 되는 경우 최적화 클래스를 높이십시오.

최적화 클래스를 선택할 때 다음의 일반 지침을 고려하십시오.

- 디폴트 쿼리 최적화 클래스 5를 사용하여 시작하십시오.
- 디폴트가 아닌 클래스를 선택할 때 먼저 클래스 1, 2 또는 3을 시도하십시오. 클래스 0, 1 및 2는 그리디(Greedy) 조인 열거 알고리즘을 사용합니다.
- 동일한 컬럼에 여러 Join 술어가 있는 테이블이 많은 경우 및 컴파일 시간이 관심 대상인 경우 최적화 클래스 1 또는 2를 사용하십시오.
- 1초 미만의 매우 짧은 런타임을 사용하는 쿼리의 경우 낮은 최적화 클래스(0 또는 1)를 사용하십시오. 이러한 쿼리는 다음과 같은 경향이 있습니다.
  - 단일 테이블 또는 일부 테이블에만 액세스합니다.
  - 단일 행 또는 일부 행만 페치합니다.
  - 완전한 고유 인덱스를 사용합니다.
  - 온라인 트랜잭션 처리(OLTP)에 관여합니다.
- 런타임이 30초를 초과하는 쿼리의 경우 높은 최적화 클래스(3, 5 또는 7)를 사용하십시오.
- 클래스 3 이상은 여러 개의 추가 대체 플랜을 고려하는 동적 프로그래밍 조인 열거 알고리즘을 사용하며 특히 테이블 수가 증가할 때 클래스 0, 1 또는 2보다 컴파일 시간이 상당히 증가할 수 있습니다.
- 쿼리에 대한 특정 최적화 요구사항이 있는 경우에만 최적화 클래스 9를 사용하십시오.

복합 쿼리에는 최상의 액세스 플랜을 선택하기 위해 다른 최적화 양이 필요합니다. 다음이 포함된 쿼리의 경우 높은 최적화 클래스를 사용하십시오.

- 대형 테이블에 액세스
- 많은 뷰
- 많은 술어
- 많은 서브쿼리
- 많은 조인
- UNION 또는 INTERSECT와 같은 많은 집합 연산자
- 많은 규정 행

- GROUP BY 및 HAVING 연산
- 중첩된 테이블 표현식

완전히 정규화된 데이터베이스에 대한 월말 보고 쿼리 또는 결정 지원 쿼리는 최소한 디폴트 쿼리 최적화 클래스가 사용되어야 하는 복합 쿼리의 좋은 예입니다.

쿼리 생성 프로그램에서 생성한 SQL 및 XQuery문의 경우 높은 쿼리 최적화 클래스를 사용하십시오. 쿼리 생성 프로그램이 많으면 비효율적인 쿼리가 작성됩니다. 올바로 작성되지 않은 쿼리의 경우 좋은 액세스 플랜을 선택하려면 추가적인 최적화가 필요합니다. 쿼리 최적화 클래스 2 이상을 사용하면 이러한 쿼리가 향상됩니다.

SAP 응용프로그램의 경우 항상 최적화 클래스 5를 사용하십시오. 이 최적화 클래스는 **DB2_REDUCED_OPTIMIZATION** 레지스트리 변수 설정 등 많은 DB2 기능이 SAP에 대해 최적화될 수 있게 해 줍니다.

페더레이티드 데이터베이스에서 최적화 클래스는 리모트 옵티마이저에 적용되지 않습니다.

최적화 클래스 설정:

최적화 레벨을 지정할 때, 쿼리에서 정적 또는 동적 SQL 및 XQuery문을 사용하는지 및 동일한 동적 쿼리가 반복적으로 실행되는지 여부를 고려하십시오.

정적 SQL 및 XQuery문의 경우, 쿼리 컴파일 시간과 자원이 한 번만 소비되고, 결과 플랜은 여러 번 사용될 수 있습니다. 일반적으로, 정적 SQL 및 XQuery문은 항상 디폴트 쿼리 최적화 클래스(5)를 사용해야 합니다. 동적문은 런타임에서 바인드되고 실행되기 때문에, 동적문에 대한 추가 최적화의 오버헤드가 전체 성능을 향상시키는지 여부를 고려하십시오. 그러나 동일한 동적 SQL 또는 XQuery문이 반복적으로 실행되는 경우, 선택된 액세스 플랜이 캐시됩니다. 이러한 문은 정적 SQL 및 XQuery문과 동일한 최적화 레벨을 사용할 수 있습니다.

쿼리가 추가 최적화의 이점을 얻는지 여부가 확실하지 않거나 컴파일 시간 및 자원 소비가 염려되는 경우, 벤치마크 테스팅을 고려하십시오.

쿼리 최적화 클래스를 지정하려면 다음 단계를 따르십시오.

1. 성능 인수를 분석하십시오.
   - 동적 쿼리문의 경우, 테스트에서 명령문의 평균 런타임을 비교해야 합니다. 다음 공식을 사용하여 평균 런타임을 추정하십시오.

     ```
     compilation time + sum of execution times for all iterations
     ------------------------------------------------------------
                        number of iterations
     ```

     반복의 수는 컴파일될 때마다 명령문이 실행되어야 하는 횟수를 나타냅니다.

주: 최초 컴파일 후, 환경 변경 시 필요할 때마다 동적 SQL 및 XQuery문이 재컴파일됩니다. 명령문이 캐시된 후 환경이 변경되지 않을 경우, 후속 PREPARE 문에서 캐시된 명령문을 재사용합니다.

- 정적 SQL 및 XQuery문의 경우, 명령문 런타임을 비교하십시오.

  정적 SQL 및 XQuery문의 컴파일 시간에 관심이 있을 수도 있지만 정적문에 대한 총 컴파일 및 실행 시간은 의미 있는 컨텍스트로 평가하기가 어렵습니다. 총 시간을 비교해도 정적문이 바인드될 때마다 여러 번 실행될 수 있다는 사실과 해당 명령문이 일반적으로 런타임 동안 바인드되지 않는다는 사실이 인식되지 않습니다.

2. 최적화 클래스를 지정하십시오.
   - 동적 SQL 및 XQuery문은 CURRENT QUERY OPTIMIZATION 특수 레지스터에 의해 지정된 최적화 클래스를 사용합니다. 예를 들어, 다음 명령문은 최적화 클래스를 1로 설정합니다.

     ```
     SET CURRENT QUERY OPTIMIZATION = 1
     ```

     동적 SQL 또는 XQuery문이 항상 동일한 최적화 클래스를 사용하도록 하려면 응용프로그램 프로그램에 SET문을 포함하십시오.

     CURRENT QUERY OPTIMIZATION 특수 레지스터가 설정되지 않은 경우, 디폴트 쿼리 최적화 클래스를 사용하여 동적문이 바인드됩니다. 동적 및 정적 쿼리 둘 다에 대한 디폴트값은 디폴트값이 5인 **dft_queryopt** 데이터베이스 구성 매개변수의 값에 의해 판별됩니다. 특수 레지스터와 바인드 옵션에 대한 디폴트 값 또한 **dft_queryopt** 데이터베이스 구성 매개변수에서 읽힙니다.
   - 정적 SQL 및 XQuery문은 PREP 및 BIND 명령에서 지정된 최적화 클래스를 사용합니다. SYSCAT.PACKAGES 카탈로그 뷰의 QUERYOPT 컬럼은 패키지를 바인드하는 데 사용되는 최적화 클래스를 기록합니다. 패키지가 내재적으로, 또는 REBIND PACKAGE 명령을 사용하여 리바인드될 경우, 이와 동일한 최적화 클래스가 정적문에 사용됩니다. 이러한 정적 SQL 및 XQuery문에 대한 최적화 클래스를 변경하려면 BIND 명령을 사용하십시오. 최적화 클래스를 지정하지 않을 경우, 데이터 서버는 **dft_queryopt** 데이터베이스 구성 매개변수에 의해 지정된 대로, 디폴트 최적화 클래스를 사용합니다.

다른 조정 옵션이 승인할 수 있는 결과를 작성하지 않을 경우 최적화 프로파일 사용

우수 사례 권장사항을 따랐지만 여전히 최적 성능에 못 미친다고 생각되면, DB2 옵티마이저에 명시적 최적화 지침을 제공할 수 있습니다.

이러한 최적화 지침은 최적화 프로파일이라고 하는 XML 문서에 포함되어 있습니다. 프로파일은 SQL문 및 연관된 최적화 지침을 정의합니다.

최적화 프로파일을 광범위하게 사용할 경우, 유지하는 데 많은 노력이 필요합니다. 더 중요하게는, 최적화 프로파일만을 사용해서 기존 SQL문에 대한 성능을 향상시킬 수 있습니다. 우수 사례를 일관되게 따르면 향후 쿼리를 포함하여 모든 쿼리에 대한 쿼리 성능 안정성을 얻는 데 도움이 될 수 있습니다.

최적화 프로파일 및 지침

최적화 프로파일은 하나 이상의 SQL문에 대한 최적화 지침을 포함할 수 있는 XML 문서입니다. 각 SQL문과 연관 최적화 지침 간의 대응은 SQL문을 명백하게 식별하는 데 필요한 SQL 텍스트 및 기타 정보를 사용하여 성립됩니다.

DB2 옵티마이저는 업계에서 가장 복잡한 비용 기반 옵티마이저 중 하나입니다. 그러나 드물게 옵티마이저에서 최적에 미치지 않는 실행 플랜을 선택할 수도 있습니다. 데이터베이스 친숙 DBA로서 db2advis, runstats 및 db2expln과 같은 유틸리티 및 최적화 클래스 설정을 사용하면 더 나은 데이터베이스 성능을 얻도록 옵티마이저를 조정하는 데 도움이 될 수 있습니다. 모든 조정 옵션이 사용된 후 예상한 결과를 수신하지 못할 경우, DB2 옵티마이저에 명시적 최적화 지침을 제공할 수 있습니다.

예를 들어, 데이터베이스 통계를 갱신하고 기타 모든 조정 단계를 수행한 후에도 옵티마이저가 여전히 다음 서브쿼리에서 SUPPLIERS 테이블에 액세스하기 위해 I_SUPPKEY 인덱스를 선택하지 않았다고 가정하십시오.

```
SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
FROM PARTS P, SUPPLIERS S, PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY
  AND S.S_SUPPKEY = PS.PS_SUPPKEY
  AND P.P_SIZE = 39
  AND P.P_TYPE = 'BRASS'
  AND S.S_NATION = 'MOROCCO'
  AND S.S_NATION IN ('MOROCCO', 'SPAIN')
  AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
                          FROM PARTSUPP PS1, SUPPLIERS S1
                          WHERE P.P_PARTKEY = PS1.PS_PARTKEY
                            AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
                            AND S1.S_NATION = S.S_NATION))
```

이 경우, 명시적 최적화 지침을 사용하여 옵티마이저에 영향을 줄 수 있습니다. 예를 들어, 다음과 같습니다.

```
<OPTGUIDELINES><IXSCAN TABLE="S" INDEX="I_SUPPKEY"/></OPTGUIDELINES>
```

최적화 지침은 단순 XML 스펙을 사용하여 지정됩니다. OPTGUIDELINES 요소의 각 요소는 DB2 옵티마이저에 의해 최적화 지침으로 해석됩니다. 이 예에서는 하나의 최적화 지침 요소가 있습니다. IXSCAN 요소는 옵티마이저가 인덱스 액세스를 사용하도록 요청합니다. IXSCAN 요소의 TABLE 속성은 목표 테이블 참조를 표시하고(테이블 참조의 표시 이름 사용) INDEX 속성은 인덱스를 지정합니다.

다음 예는 이전 쿼리를 기반으로 하며, 최적화 프로파일을 사용하여 DB2 옵티마이저로 최적화 지침을 전달할 수 있는 방법을 나타냅니다.

```
<?xml version="1.0" encoding="UTF-8">

<OPTPROFILE VERSION="9.1.0.0">
<STMTPROFILE ID="Guidelines for TPCD Q9">
  <STMTKEY SCHEMA="TPCD">
```

```
    SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
    FROM PARTS P, SUPPLIERS S, PARTSUPP PS
    WHERE P_PARTKEY = PS.PS_PARTKEY
      AND S.S_SUPPKEY = PS.PS_SUPPKEY
      AND P.P_SIZE = 39
      AND P.P_TYPE = 'BRASS'
      AND S.S_NATION = 'MOROCCO'
      AND S.S_NATION IN ('MOROCCO', 'SPAIN')
      AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
                              FROM PARTSUPP PS1, SUPPLIERS S1
                              WHERE P.P_PARTKEY = PS1.PS_PARTKEY
                                AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
                                AND S1.S_NATION = S.S_NATION))
  </STMTKEY>
  <OPTGUIDELINES><IXSCAN TABLE="S" INDEX="I_SUPPKEY"/></OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

각 STMTPROFILE 요소는 하나의 응용프로그램 명령문에 대한 최적화 지침 세트를 제공합니다. 목표 명령문은 STMTKEY 하위 요소에 의해 식별됩니다. 그런 다음 최적화 프로파일에 스키마 규정 이름이 제공되고 데이터베이스에 삽입됩니다. BIND 또는 PRECOMPILE 명령에서 이 이름을 지정하면 최적화 프로파일이 명령문에 대해 실행됩니다.

최적화 프로파일은 응용프로그램 또는 데이터베이스 구성 변경 없이 옵티마이저에 최적화 지침이 제공될 수 있도록 해 줍니다. 간단히 단순 XML 문서를 구성하고, 이를 데이터베이스에 삽입한 후 BIND 또는 PRECOMPILE 명령에서 최적화 프로파일의 이름을 지정합니다. 옵티마이저가 최적화 지침을 자동으로 해당 명령문에 일치시킵니다.

최적화 지침은 포괄적일 필요는 없지만, 필요한 실행 플랜을 목표로 해야 합니다. DB2 옵티마이저는 여전히 기존 비용 기반 방법을 사용하여 기타 가능한 액세스 플랜을 고려합니다. 특정 테이블 참조를 목표로 하는 최적화 지침이 일반 최적화 설정을 겹쳐쓸 수 없습니다. 예를 들어, 테이블 A와 B 간의 병합 조인을 지정하는 최적화 지침은 최적화 클래스 0에서 유효하지 않습니다.

옵티마이저는 유효하지 않거나 적용 가능하지 않은 최적화 지침을 무시합니다. 최적화 지침이 무시될 경우, 실행 플랜이 작성되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다. 그러면 EXPLAIN문을 사용하여 최적화 지침 처리에 대한 자세한 진단 정보를 얻을 수 있습니다.

최적화 프로파일:

최적화 프로파일 분석:

최적화 프로파일은 프로파일이 적용되는 동안 실행되는 모든 데이터 처리 언어(DML) 명령문에 적용되는 전역 지침을 포함할 수 있으며, 패키지의 개별 DML 명령문에 적용되는 특정 지침을 포함할 수 있습니다.

예:

- 현재 최적화 프로파일이 활성인 동안 명령문이 처리될 때마다 옵티마이저에서 구체화된 쿼리 테이블(MQT) Test.SumSales 및 Test.AvgSales를 참조하도록 요청하는 전역 최적화 지침을 쓸 수 있습니다.

- 옵티마이저에서 지정된 명령문을 발견할 때마다 SUPPLIERS 테이블에 액세스하는 데 I_SUPPKEY 인덱스가 사용되도록 요청하는 명령문 레벨 최적화 지침을 쓸 수 있습니다.

최적화 프로파일은 이러한 두 가지 유형의 지침을 지정할 수 있는 두 개의 주요 섹션을 포함합니다. 전역 최적화 지침 섹션은 하나의 OPTGUIDELINES 요소를 포함할 수 있고, 명령문 프로파일 섹션은 임의의 수의 STMTPROFILE 요소를 포함할 수 있습니다. 최적화 프로파일은 메타데이터 및 처리 지시문을 포함하는 OPTPROFILE 요소도 포함해야 합니다.

다음 코드는 DB2 버전 9.1에 대한 유효한 최적화 프로파일의 예로, 하나의 STMTPROFILE 요소가 있는 명령문 프로파일 섹션과 전역 최적화 지침 섹션을 포함합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.1.0.0">

  <!--
       Global optimization guidelines section.
       Optional but at most one.
  -->
  <OPTGUIDELINES>
    <MQT NAME="Test.AvgSales"/>
    <MQT NAME="Test.SumSales"/>
  </OPTGUIDELINES>

  <!--
       Statement profile section.
       Zero or more.
  -->
  <STMTPROFILE ID="Guidelines for TPCD Q9">
    <STMTKEY SCHEMA="TPCD">
      <![CDATA[SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE,
S.S_COMMENT FROM PARTS P, SUPPLIERS S, PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY
AND P.P_SIZE = 39 AND P.P_TYPE = 'BRASS'
AND S.S_NATION = 'MOROCCO' AND S.S_NATION IN ('MOROCCO', 'SPAIN')
AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
FROM PARTSUPP PS1, SUPPLIERS S1
WHERE P.P_PARTKEY  = PS1.PS_PARTKEY AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
AND S1.S_NATION  = S.S_NATION)]]>
    </STMTKEY>
    <OPTGUIDELINES>
      <IXSCAN TABID="Q1" INDEX="I_SUPPKEY"/>
    </OPTGUIDELINES>
  </STMTPROFILE>

</OPTPROFILE>
```

OPTPROFILE 요소

최적화 프로파일은 OPTPROFILE 요소로 시작합니다. 위의 예에서, 이 요소는 최적화 프로파일 버전이 9.1임을 지정하는 VERSION 속성으로 구성됩니다.

전역 최적화 지침 섹션

전역 최적화 지침은 최적화 프로파일이 적용되는 모든 명령문에 적용됩니다. 전역 최적화 지침 섹션은 전역 OPTGUIDELINES 요소에 의해 표시됩니다. 위의 예에서, 이 섹션은 최적화 프로파일이 적용되는 명령문을 처리할 때 MQTs Test.AvgSales 및 Test.SumSales가 고려되어야 함을 지정하는 단일 전역 최적화 지침을 포함합니다.

명령문 프로파일 섹션

명령문 프로파일은 특정 명령문에 적용되는 최적화 지침을 정의합니다. 최적화 프로파일에는 0개 이상의 명령문 프로파일이 있을 수 있습니다. 명령문 프로파일 섹션은 STMTPROFILE 요소에 의해 표시됩니다. 위의 예에서, 이 섹션은 최적화 프로파일이 적용되는 특정 명령문에 대한 지침을 포함합니다.

각 명령문 프로파일은 각각 STMTKEY와 OPTGUIDELINES로 표시되는, 명령문 키와 명령문 레벨 최적화 지침을 포함합니다.

- 명령문 키는 명령문 레벨 최적화 지침이 적용되는 명령문을 식별합니다. 이 예에서 STMTKEY 요소는 명령문을 명백하게 식별하는 데 필요한 기타 정보와 원래 명령문 텍스트를 포함합니다. 명령문 키를 사용하여 옵티마이저는 명령문 프로파일과 해당 명령문을 일치시킵니다. 이 관계는 응용프로그램을 수정할 필요 없이 명령문에 대한 최적화 지침을 제공할 수 있도록 해 줍니다.
- 명령문 프로파일의 명령문 레벨 최적화 지침 섹션은 OPTGUIDELINES 요소에 의해 표시됩니다. 이 섹션은 명령문에서 테이블을 조인하거나 액세스하기 위한 메소드를 지정하는 하나 이상의 액세스 또는 조인 요청으로 구성됩니다. 명령문 프로파일에서 명령문 키를 일치시킨 후, 옵티마이저는 명령문을 최적화할 때 연관된 명령문 레벨 최적화 지침을 참조합니다. 이 예에는 중첩된 하위 선택에서 참조되는 SUPPLIERS 테이블이 I_SUPPKEY라는 이름의 인덱스를 사용하도록 지정하는 하나의 액세스 요청을 포함되어 있습니다.

최적화 프로파일 작성:

최적화 프로파일은 하나 이상의 데이터 처리 언어(DML) 명령문에 대한 최적화 지침을 포함하는 XML 문서입니다.

최적화 프로파일은 많은 조합의 지침을 포함할 수 있으므로, 다음 정보는 최적화 프로파일 작성에 대해 일반적인 단계만 지정합니다.

최적화 프로파일을 작성하려면 다음을 수행하십시오.

1. XML 편집기를 시작하십시오. 가능한 경우, 스키마 유효성 확인 성능이 있는 것을 사용하십시오. 옵티마이저는 XML 유효성 확인을 수행하지 않습니다. 최적화 프로파일은 현재 최적화 프로파일 스키마에 따라 유효해야 합니다.

2. 적절한 이름을 사용하여 새 XML 문서를 작성하십시오. 적용할 명령문의 범위를 설명하는 이름을 제공할 수도 있습니다. 예를 들어, 다음과 같습니다.
   ```
   inventory_db.xml
   ```
3. 문서에 XML 선언을 추가하십시오. 인코딩 형식을 지정하지 않을 경우, UTF-8이 가정됩니다. 가능한 경우, UTF-16 인코딩을 사용하여 문서를 저장하십시오. 이 인코딩을 처리할 때 데이터 서버가 보다 효과적입니다.
   ```
   <?xml version="1.0" encoding="UTF-16"?>
   ```
4. 문서에 최적화 프로파일 섹션을 추가하십시오.
   ```
   <OPTPROFILE VERSION="9.1.0.0">
   </OPTPROFILE>
   ```
5. OPTPROFILE 요소 내에서, 상황에 맞게 전역 또는 명령문 레벨 최적화 지침을 작성하고 파일을 저장하십시오.

최적화 프로파일을 사용하도록 데이터 서버 구성:

최적화 프로파일이 작성되고 현재 최적화 프로파일 스키마(COPS)에 대해 해당 컨텐츠가 유효성 확인된 후, 컨텐츠는 고유 스키마 규정 이름과 연결되어 SYSTOOLS.OPT_PROFILE 테이블에 저장되어야 합니다.

최적화 프로파일을 사용하도록 데이터 서버를 구성하려면 다음을 수행하십시오.

1. 최적화 프로파일 테이블을 작성하십시오. 테이블의 각 행은 하나의 최적화 프로파일을 포함할 수 있습니다. SCHEMA 및 NAME 컬럼은 최적화 프로파일을 식별하고, PROFILE 컬럼은 최적화 프로파일의 텍스트를 포함합니다.
2. 선택사항: 데이터베이스 보안 요구사항을 충족하는 테이블에 권한 또는 특권을 부여할 수 있습니다. 이것은 옵티마이저가 테이블을 읽는 기능에 아무런 영향을 미치지 않습니다.
3. 사용하려는 최적화 프로파일을 테이블에 삽입하십시오.

옵티마이저에서 사용할 최적화 프로파일 지정:

OPTPROFILE 바인드 옵션을 사용하여 패키지 레벨에서 최적화 프로파일이 사용되도록 지정하거나, CURRENT OPTIMIZATION PROFILE 특수 레지스터를 사용하여 명령문 레벨에서 최적화 프로파일이 사용되도록 지정합니다.

이 특수 레지스터는 최적화를 위해 동적으로 준비되는 명령문에 의해 사용되는 최적화 프로파일의 규정된 이름을 포함합니다. CLI 응용프로그램의 경우, CURRENTOPTIMIZATIONPROFILE 클라이언트 구성 옵션을 사용하여 각 연결에 대해 이 특수 레지스터를 설정할 수 있습니다.

OPTPROFILE 바인드 옵션 설정은 또한 CURRENT OPTIMIZATION PROFILE 특수 레지스터에 대한 디폴트 최적화 프로파일을 지정합니다. 디폴트에 대한 우선 순서는 다음과 같습니다.

- 다른 설정에 상관없이, OPTPROFILE 바인드 옵션이 모든 정적문에 적용됩니다.
- 동적문의 경우, 최저에서 최고의 우선 순서로 다음에 의해 CURRENT OPTIMIZATION PROFILE 특수 레지스터가 판별됩니다.
  - OPTPROFILE 바인드 옵션
  - CURRENTOPTIMIZATIONPROFILE 클라이언트 구성 옵션
  - 응용프로그램에서 가장 최근 SET CURRENT OPTIMIZATION PROFILE문

응용프로그램 내 최적화 프로파일 설정:

SET CURRENT OPTIMIZATION PROFILE문을 사용하여 응용프로그램에서 동적 문에 대한 현재 최적화 프로파일의 설정을 제어할 수 있습니다.

명령문에서 제공하는 최적화 프로파일 이름은 스키마 규정 이름이어야 합니다. 스키마 이름을 제공하지 않을 경우, CURRENT SCHEMA 특수 레지스터의 값이 내재된 스키마 규정자로 사용됩니다.

지정하는 최적화 프로파일은 다른 SET CURRENT OPTIMIZATION PROFILE문이 발견될 때까지 모든 연속 동적문에 적용됩니다. 정적 동적문은 이 설정이 평가되기 전에 선행 처리되어 패키지되기 때문에 영향을 받지 않습니다.

응용프로그램 내에서 최적화 프로파일을 설정하려면 다음을 수행하십시오.

- 응용프로그램 내 임의의 위치에서 SET CURRENT OPTIMIZATION PROFILE문을 사용하십시오. 예를 들어, 다음 시퀀스에서 마지막 명령문은 JON.SALES 최적화 프로파일에 따라 최적화됩니다.

```
EXEC SQL  SET CURRENT OPTIMIZATION PROFILE = 'NEWTON.INVENTDB';

/* The following statements are both optimized with 'NEWTON.INVENTDB' */
EXEC SQL PREPARE stmt FROM SELECT ... ;
EXEC SQL EXECUTE stmt;

EXEC SQL EXECUTE IMMEDIATE SELECT ... ;

EXEC SQL SET CURRENT OPTIMIZATION PROFILE = 'JON.SALES';

/* This statement is optimized with 'JON.SALES' */
EXEC SQL EXECUTE IMMEDIATE SELECT ... ;
```

- 옵티마이저가 응용프로그램의 실행이 시작되었을 때 적용된 디폴트 최적화 프로파일을 사용하도록 하려는 경우, 널(NULL) 값을 지정하십시오. 예를 들어, 다음과 같습니다.

```
EXEC SQL SET CURRENT OPTIMIZATION PROFILE = NULL;
```

- 옵티마이저가 최적화 프로파일을 사용하지 않도록 하려면 비어 있는 문자열을 지정하십시오. 예를 들어, 다음과 같습니다.

```
EXEC SQL SET CURRENT OPTIMIZATION PROFILE = '';
```

- 콜 레벨 인터페이스(CLI) 응용프로그램을 사용 중인 경우, 구성 지원 프로그램 또는 UPDATE CLI CONFIGURATION 명령을 사용하여 CURRENTOPTIMIZATIONPROFILE 매개변수를 db2cli.ini 파일에 추가할 수 있습니다. 예를 들어, 다음과 같습니다.

  ```
  update cli cfg for section sanfran using currentoptimizationprofile jon.sales
  ```

  이 결과, db2cli.ini 파일에 다음 항목이 생깁니다.

  ```
  [SANFRAN]
  CURRENTOPTIMIZATIONPROFILE=JON.SALES
  ```

  주: 응용프로그램의 모든 SET CURRENT OPTIMIZATION PROFILE문은 이 설정을 겹쳐씁니다.

패키지에 최적화 프로파일 바인드:

BIND 또는 **PRECOMPILE** 명령을 사용하여 패키지를 준비할 때, OPTPROFILE 옵션을 사용하여 패키지에 대한 최적화 프로파일을 지정할 수 있습니다.

이 방법은 정적문에 최적화 프로파일을 적용하는 유일한 방법이고, 지정된 프로파일은 패키지의 모든 정적문에 적용됩니다. 이런 식으로 지정되는 최적화 프로파일은 또한 패키지 내 동적문에 대해 사용되는 디폴트 최적화 프로파일입니다.

API(예를 들어, sqlaprep) 또는 명령행 처리기(CLP)를 사용하여 SQLJ 또는 Embedded SQL로 최적화 프로파일을 바인드할 수 있습니다.
예를 들어, 다음 코드는 CLP에서 인벤토리 응용프로그램에 인벤토리 데이터베이스 최적화 프로파일을 바인드하는 방법을 보여줍니다.

```
db2 prep inventapp.sqc bindfile optprofile newton.inventdb
db2 bind inventapp.bnd
db2 connect reset
db2 terminate
xlc -I$HOME/sqllib/include -c inventapp.c -o inventapp.o
xlc -o inventapp inventapp.o -ldb2 -L$HOME/sqllib/lib
```

최적화 프로파일에 대한 스키마 이름을 지정하지 않을 경우, QUALIFIER 옵션이 내재된 규정자로 사용됩니다.

최적화 프로파일 수정:

문서를 편집하고, 현재 최적화 프로파일 스키마(COPS)에 대해 유효성을 확인하고, SYSTOOLS.OPT_PROFILE 테이블의 원래 문서를 새 버전으로 교체하여 최적화 프로파일을 수정할 수 있습니다.

최적화 프로파일은 참조될 때 메모리에서 컴파일 및 캐시되므로 이러한 참조 또한 제거해야 합니다. FLUSH OPTIMIZATION PROFILE CACHE문을 사용하여 최적화 프

로파일 캐시에서 이전 프로파일을 제거하고 이전 프로파일을 사용하여 준비된 동적 플랜 캐시에 있는 명령문을 무효화하십시오(*논리적 무효화*). 최적화 프로파일을 수정하려면 다음을 수행하십시오.

1. 필요한 변경을 적용하여 최적화 프로파일을 편집하고 XML을 유효성 확인하십시오.
2. 새 프로파일로 SYSTOOLS.OPT_PROFILE 테이블을 갱신하십시오.
3. 최적화 프로파일 캐시를 플러시하기 위한 트리거를 작성하지 않은 경우, FLUSH OPTIMIZATION PROFILE CACHE문을 발행하여 최적화 프로파일 캐시에 포함되어 있을 수 있는 모든 버전의 최적화 프로파일을 제거하십시오.

   주: 최적화 프로파일 캐시를 플러시하면, 이전 최적화 프로파일을 사용하여 준비된 동적문도 동적 플랜 캐시에서 무효화됩니다.

이후에 최적화 프로파일을 참조하면 옵티마이저가 새 프로파일을 읽고 이를 최적화 프로파일 캐시에 다시 로드하게 됩니다. 또한 이전 최적화 프로파일로 준비된 명령문의 논리적 무효화로 인해 해당 명령문에 대해 수행된 모든 호출이 새 최적화 프로파일로 준비되고 동적 플랜 캐시에 다시 캐시됩니다.

최적화 프로파일 삭제:

더 이상 필요하지 않은 최적화 프로파일을 SYSTOOLS.OPT_PROFILE 테이블에서 삭제하여 제거할 수 있습니다. 최적화 프로파일은 참조될 때 메모리에서 컴파일 및 캐시되므로, 원래 프로파일이 이미 사용된 경우 삭제된 최적화 프로파일을 최적화 프로파일 캐시에서도 플러시해야 합니다.

최적화 프로파일을 삭제하려면 다음을 수행하십시오.

1. SYSTOOLS.OPT_PROFILE 테이블에서 최적화 프로파일을 삭제하십시오. 예를 들어, 다음과 같습니다.

   ```
   delete from systools.opt_profile
     where schema = 'NEWTON' and name = 'INVENTDB'
   ```

2. 최적화 프로파일 캐시를 플러시하기 위한 트리거를 작성하지 않은 경우, FLUSH OPTIMIZATION PROFILE CACHE문을 발행하여 최적화 프로파일 캐시에 포함되어 있을 수 있는 모든 버전의 최적화 프로파일을 제거하십시오.

   주: 최적화 프로파일 캐시를 플러시하면, 이전 최적화 프로파일을 사용하여 준비된 동적문도 동적 플랜 캐시에서 무효화됩니다.

이후에 최적화 프로파일을 참조하면 옵티마이저가 이유 코드 13과 함께 SQL0437W를 리턴하게 됩니다.

최적화 지침:

최적화 지침의 유형:

DB2 옵티마이저는 두 단계, 쿼리 재작성 최적화 단계 및 플랜 최적화 단계로 명령문을 처리합니다.

최적화된 명령문은 *쿼리 재작성 최적화 단계*에서 판별됩니다. 이 단계에서는 원래 명령문을 플랜 최적화 단계에서 더 쉽게 최적화할 수 있는 의미적으로 동등한 명령문으로 변환합니다. *플랜 최적화 단계*에서는 여러 대안을 열거하고 실행 비용 추정을 최소화하는 대안을 선택하여 *최적화된 명령문*에 대한 최적 액세스 메소드, 조인 메소드 및 조인 순서를 판별합니다.

두 최적화 단계 동안 고려되는 쿼리 변환, 액세스 메소드, 조인 메소드, 조인 순서 및 기타 최적화 대안은 CURRENT QUERY OPTIMIZATION 특수 레지스터, REOPT 바인드 옵션 및 **DB2_REDUCED_OPTIMIZATION** 레지스트리 변수와 같은 다양한 DB2 매개변수에 의해 제어됩니다. 최적화 대안 세트를 *검색 스페이스*라고 합니다.

다음 유형의 명령문 최적화 지침이 지원됩니다.

- *일반 최적화 지침*: 일반 최적화 매개변수의 설정에 영향을 주기 위해 사용할 수 있는 지침으로, 검색 스페이스에 영향을 줄 수 있기 때문에 첫 번째로 적용됩니다.
- *쿼리 재작성 지침*: 쿼리 재작성 최적화 단계 동안 고려되는 변환에 영향을 주기 위해 사용할 수 있는 지침으로, 플랜 최적화 단계 동안 최적화되는 명령문에 영향을 줄 수 있기 때문에 다음으로 적용됩니다.
- *플랜 최적화 지침*: 플랜 최적화 단계 동안 고려되는 액세스 메소드, 조인 메소드 및 조인 순서에 영향을 주기 위해 사용할 수 있는 지침으로, 마지막으로 적용됩니다.

일반 최적화 지침:

일반 최적화 지침은 일반 최적화 매개변수를 설정하는 데 사용할 수 있습니다.

이러한 각 지침에는 명령문 레벨 범위가 있습니다.

쿼리 재작성 최적화 지침:

쿼리 재작성 지침은 원래 명령문을 의미적으로 동등한, 최적화된 명령문으로 변환하는 쿼리 재작성 최적화 단계 동안 고려되는 변환에 영향을 주기 위해 사용할 수 있습니다.

최적화된 명령문에 대한 최적 실행 플랜은 플랜 최적화 단계 동안 판별됩니다. 따라서, 쿼리 재작성 최적화 지침이 플랜 최적화 지침의 적용 가능성에 영향을 줄 수 있습니다.

각 쿼리 재작성 최적화 지침은 옵티마이저의 쿼리 변환 규칙 중 하나에 해당합니다. 쿼리 재작성 최적화 지침에 의해 다음 쿼리 변환 규칙이 영향을 받을 수 있습니다.

- IN-LIST-to-join
- Subquery-to-join
- NOT-EXISTS-subquery-to-antijoin
- NOT-IN-subquery-to-antijoin

쿼리 재작성 최적화 지침은 항상 적용 가능하지는 않습니다. 쿼리 재작성 규칙은 한 번에 하나씩 적용됩니다. 따라서, 후속 규칙 전에 적용된 쿼리 재작성 규칙이 해당 규칙과 연관된 쿼리 재작성 최적화 지침에 영향을 줄 수 있습니다. 환경 구성이 일부 재작성 규칙의 동작에 영향을 줄 수 있고, 다시 이 동작은 특정 규칙에 대한 쿼리 재작성 최적화 지침의 적용 가능성에 영향을 줍니다.

매번 동일한 결과를 얻기 위해 쿼리 재작성 규칙은 규칙이 실행되기 전에 적용되는 특정 조건을 갖습니다. 쿼리 재작성 구성요소가 쿼리에 규칙을 적용하려고 시도할 때 규칙과 연관된 조건이 충족되지 않을 경우, 해당 규칙에 대한 쿼리 재작성 최적화 지침이 무시됩니다. 쿼리 재작성 최적화 지침이 적용 가능하지 않고, 지침이 사용 가능 지침인 경우, 이유 코드 13과 함께 SQL0437W가 리턴됩니다. 쿼리 재작성 최적화 지침이 적용 가능하지 않고, 지침이 사용 불가능 지침인 경우, 메시지가 리턴되지 않습니다. 이 경우 규칙이 사용 불가능한 것처럼 처리되기 때문에 쿼리 재작성 규칙이 적용되지 않습니다.

쿼리 재작성 최적화 지침은 두 가지 범주, 명령문 레벨 지침과 술어 레벨 지침으로 구분될 수 있습니다. 모든 쿼리 재작성 최적화 지침은 명령문 레벨 범주를 지원합니다. INLIST2JOIN만 술어 레벨 범주를 지원합니다. 명령문 레벨 쿼리 재작성 최적화 지침은 전체 쿼리에 적용됩니다. 술어 레벨 쿼리 재작성 최적화 지침은 특정 술어에만 적용됩니다. 명령문 레벨과 술어 레벨 쿼리 재작성 최적화 지침이 모두 지정된 경우, 특정 술어에 대해 술어 레벨 지침이 명령문 레벨 지침을 겹쳐씁니다.

각 쿼리 재작성 최적화 지침은 최적화 지침 스키마에서 해당 재작성 요청 요소에 의해 나타납니다.

다음 예는 INLIST2JOIN 재작성 요청 요소가 나타내는 대로 IN-LIST-to-join 쿼리 재작성 최적화 지침을 설명합니다.

```
SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
FROM "Tpcd".PARTS P, "Tpcd".SUPPLIERS S, "Tpcd".PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY
  AND S.S_SUPPKEY = PS.PS_SUPPKEY
  AND P_SIZE IN (35, 36, 39, 40)
  AND S.S_NATION IN ('INDIA', 'SPAIN')
  AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
                          FROM "Tpcd".PARTSUPP PS1, "Tpcd".SUPPLIERS S1
                          WHERE P.P_PARTKEY = PS1.PS_PARTKEY
                            AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
                            AND S1.S_NATION = S.S_NATION)
ORDER BY S.S_NAME
<OPTGUIDELINES><INLIST2JOIN TABLE='P'/></OPTGUIDELINES>;
```

이 특정 쿼리 재작성 최적화 지침은 술어 P_SIZE IN (35, 36, 39, 40)의 상수 목록을 테이블 표현식으로 변환해야 함을 지정합니다. 그러면 이 테이블 표현식이 기본 subselect의 PARTS 테이블에 대한 인덱스된 중첩된 루프 조인 액세스를 구동할 수 있습니다. TABLE 속성은 이 술어가 적용되는 테이블 참조를 표시하여 목표 IN-LIST 술

어를 식별하는 데 사용됩니다. 식별된 테이블 참조에 대한 IN-LIST 술어가 여러 개인 경우, INLIST2JOIN 재작성 요청 요소는 앰비규어스로 간주되어 무시됩니다.

이러한 경우, COLUMN 속성을 추가하여 목표 IN-LIST 술어를 추가로 규정할 수 있습니다. 예를 들어, 다음과 같습니다.

```
SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
FROM "Tpcd".PARTS P, "Tpcd".SUPPLIERS S, "Tpcd".PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY
  AND S.S_SUPPKEY = PS.PS_SUPPKEY
  AND P_SIZE IN (35, 36, 39, 40)
  AND P_TYPE IN ('BRASS', 'COPPER')
  AND S.S_NATION IN ('INDIA', 'SPAIN')
  AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
                          FROM "Tpcd".PARTSUPP PS1, "Tpcd".SUPPLIERS S1
                          WHERE P.P_PARTKEY = PS1.PS_PARTKEY
                            AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
                            AND S1.S_NATION = S.S_NATION)
ORDER BY S.S_NAME
<OPTGUIDELINES><INLIST2JOIN TABLE='P' COLUMN='P_SIZE'/></OPTGUIDELINES>;
```

INLIST2JOIN 요소의 TABLE 속성은 기본 subselect의 PARTS 테이블 참조를 식별합니다. COLUMN 속성은 P_SIZE 컬럼의 IN-LIST 술어를 목표로 식별하는 데 사용됩니다. 일반적으로, COLUMN 속성 값은 목표 IN-LIST 술어에서 참조된 컬럼의 규정되지 않은 이름을 포함할 수 있습니다. COLUMN 속성이 TABLE 속성 없이 제공되는 경우, 쿼리 재작성 최적화 지침은 유효하지 않은 것으로 간주되어 무시됩니다.

OPTION 속성을 사용하여 특정 쿼리 재작성 최적화 지침을 사용하거나 사용할 수 없도록 할 수 있습니다. 다음 예에서는 OPTION 속성이 DISABLE로 설정되어 있기 때문에 술어 P_SIZE IN (35, 36, 39, 40)의 상수 목록이 테이블 표현식으로 변환되지 않습니다. OPTION 속성의 디폴트값은 ENABLE입니다. ENABLE 및 DISABLE을 대문자로 지정해야 합니다.

```
<OPTGUIDELINES>
  <INLIST2JOIN TABLE='P' COLUMN='P_SIZE' OPTION='DISABLE'/>
</OPTGUIDELINES>
```

다음 예에서는 INLIST2JOIN 재작성 요청 요소에 TABLE 속성이 없습니다. 옵티마이저는 이를 명령문의 모든 IN-LIST 술어에 대해 IN-LIST-to-join 쿼리 변환을 사용할 수 없도록 하기 위한 요청으로 해석합니다.

```
<OPTGUIDELINES><INLIST2JOIN OPTION='DISABLE'/></OPTGUIDELINES>
```

다음 예는 SUBQ2JOIN 재작성 요청 요소가 나타내는 대로 subquery-to-join 쿼리 재작성 최적화 지침을 설명합니다. subquery-to-join 변환은 서브쿼리를 동등한 테이블 표현식으로 변환합니다. 변환은 EXISTS, IN, =SOME, =ANY, <>SOME 또는 <>ANY가 정량화하는 서브쿼리 술어에 적용됩니다. subquery-to-join 쿼리 재작성 최적화 지침

은 서브쿼리가 병합됨을 보장하지 않습니다. 이 쿼리 재작성 최적화 지침은 특정 서브쿼리를 목표로 할 수 없습니다. 명령문 레벨에서만 변환을 사용 가능하게 하거나 사용 불가능하게 할 수 있습니다.

```
SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
FROM "Tpcd".PARTS P, "Tpcd".SUPPLIERS S, "Tpcd".PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY
  AND S.S_SUPPKEY = PS.PS_SUPPKEY
  AND P_SIZE IN (35, 36, 39, 40)
  AND P_TYPE = 'BRASS'
  AND S.S_NATION IN ('INDIA', 'SPAIN')
  AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
                          FROM "Tpcd".PARTSUPP PS1, "Tpcd".SUPPLIERS S1
                          WHERE P.P_PARTKEY = PS1.PS_PARTKEY
                            AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
                            AND S1.S_NATION = S.S_NATION)
ORDER BY S.S_NAME
<OPTGUIDELINES><SUBQ2JOIN OPTION='DISABLE'/></OPTGUIDELINES>;
```

다음 예는 NOTEX2AJ 재작성 요청 요소가 나타내는 대로 NOT-EXISTS-to-anti-join 쿼리 재작성 최적화 지침을 설명합니다. NOT-EXISTS-to-anti-join 변환은 anti-join 시맨틱을 사용하여 다른 테이블에 조인되는 테이블 표현식으로 서브쿼리를 변환합니다(일치하지 않는 행만 리턴됨). NOT-EXISTS-to-anti-join 쿼리 재작성 최적화 지침은 NOT EXISTS가 정량화하는 서브쿼리 술어에 적용됩니다. NOT-EXISTS-to-anti-join 쿼리 재작성 최적화 지침은 서브쿼리가 병합됨을 보장하지 않습니다. 이 쿼리 재작성 최적화 지침은 특정 서브쿼리를 목표로 할 수 없습니다. 명령문 레벨에서만 변환을 사용 가능하게 하거나 사용 불가능하게 할 수 있습니다.

```
SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
FROM "Tpcd".PARTS P, "Tpcd".SUPPLIERS S, "Tpcd".PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY
  AND S.S_SUPPKEY = PS.PS_SUPPKEY
  AND P_SIZE IN (35, 36, 39, 40)
  AND P_TYPE = 'BRASS'
  AND S.S_NATION IN ('INDIA', 'SPAIN')
  AND NOT EXISTS (SELECT 1
                  FROM "Tpcd".SUPPLIERS S1
                  WHERE S1.S_SUPPKEY = PS.PS_SUPPKEY)
ORDER BY S.S_NAME
<OPTGUIDELINES><NOTEX2AJ OPTION='ENABLE'/></OPTGUIDELINES>;
```

주: 명령문 레벨에서 쿼리 변환 규칙 지원은 명령문의 특정 파트에 규칙이 적용됨을 보장하지 않습니다. 쿼리 변환이 발생할지 여부를 판별하는 데 일반적인 기준이 사용됩니다. 예를 들어, 쿼리 블록에 NOT EXISTS 술어가 여러 개인 경우 옵티마이저는 이를 anti-joins로 변환하는 것을 고려하지 않습니다. 명령문 레벨에서 쿼리 변환을 명시적으로 사용하면 이 동작이 변경되지 않습니다.

다음 예는 NOTIN2AJ 재작성 요청 요소가 나타내는 대로 NOT-IN-to-anti-join 쿼리 재작성 최적화 지침을 설명합니다. NOT-IN-to-anti-join 변환은 anti-join 시맨틱을 사용하여 다른 테이블에 조인되는 테이블 표현식으로 서브쿼리를 변환합니다(일치하지 않

는 행만 리턴됨). NOT-IN-to-anti-join 쿼리 재작성 최적화 지침은 NOT IN이 정량화하는 서브쿼리 술어에 적용됩니다. NOT-IN-to-anti-join 쿼리 재작성 최적화 지침은 서브쿼리가 병합됨을 보장하지 않습니다. 이 쿼리 재작성 최적화 지침은 특정 서브쿼리를 목표로 할 수 없습니다. 명령문 레벨에서만 변환을 사용 가능하게 하거나 사용 불가능하게 할 수 있습니다.

```
SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
FROM "Tpcd".PARTS P, "Tpcd".SUPPLIERS S, "Tpcd".PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY
  AND S.S_SUPPKEY = PS.PS_SUPPKEY
  AND P_SIZE IN (35, 36, 39, 40)
  AND P_TYPE = 'BRASS'
  AND S.S_NATION IN ('INDIA', 'SPAIN')
  AND PS.PS_SUPPKEY NOT IN (SELECT S1.S_SUPPKEY
                            FROM "Tpcd".SUPPLIERS S1
                            WHERE S1.S_NATION = 'CANADA')
ORDER BY S.S_NAME
<OPTGUIDELINES><NOTIN2AJ OPTION='ENABLE'/></OPTGUIDELINES>
```

기타 쿼리 재작성 변환 컨텍스트에서 명령문에 적용된다고 고려할 경우 특정 쿼리 재작성 최적화 지침을 적용하지 못할 수도 있습니다. 즉, 변환을 사용할 수 있게 하기 위한 지침 요청을 적용할 수 없으면 경고가 리턴됩니다. 예를 들어, 다른 쿼리 변환에 의해 쿼리에서 제거되는 술어를 목표로 하는 INLIST2JOIN 재작성 사용 요청 요소를 적용할 수 없습니다. 뿐만 아니라 쿼리 재작성 최적화 지침을 적용하면 기타 쿼리 재작성 변환 규칙의 적용성이 변경될 수 있습니다. 예를 들어, IN-LIST를 테이블 표현식으로 변환하려는 요청은 다른 IN-LIST가 테이블 표현식으로 변환되지 못하게 할 수 있습니다. 그 이유는 옵티마이저가 쿼리 블록당 하나의 IN-LIST-to-join 변환만 적용하기 때문입니다.

플랜 최적화 지침:

플랜 최적화 지침은 명령문에 대한 실행 플랜의 액세스 메소드, 조인 메소드, 조인 순서 및 기타 세부사항이 판별되는 비용 기반 최적화 단계 동안 적용됩니다.

플랜 최적화 지침은 실행 플랜의 모든 측면을 지정할 필요가 없습니다. 지정되지 않은 실행 플랜의 측면은 비용 기반 방식으로 옵티마이저에 의해 판별됩니다.

플랜 최적화 지침에는 두 가지 범주가 있습니다.

- accessRequest – 액세스 요청은 명령문의 테이블 참조를 충족시키기 위한 액세스 메소드를 지정합니다.
- joinRequest – 조인 요청은 조인 조작을 수행하기 위한 메소드 및 시퀀스를 지정합니다. 조인 요청은 액세스 또는 기타 조인 요청으로 구성됩니다.

액세스 요청 최적화 지침은 테이블 스캔, 인덱스 스캔 및 리스트 프리페치와 같은 옵티마이저의 데이터 액세스 메소드에 해당합니다. 조인 요청 지침은 중첩된 루프 조인, 해

시 조인 및 병합 조인과 같은 옵티마이저의 조인 메소드에 해당합니다. 각 액세스 요청 및 조인 요청은 명령문 최적화 지침 스키마의 해당 액세스 요청 요소 및 조인 요청 요소로 나타냅니다.

다음 예는 IXSCAN 액세스 요청 요소가 나타내는 대로 인덱스 스캔 액세스 요청을 설명합니다. 이 특정 요청은 옵티마이저가 I_SUPPKEY 인덱스를 사용하여 명령문의 기본 subselect에서 SUPPLIERS 테이블에 액세스하도록 지정합니다. 선택적 INDEX 속성은 원하는 인덱스를 식별합니다. TABLE 속성은 액세스 요청이 적용되는 테이블 참조를 식별합니다. TABLE 속성은 표시 이름(이 예에서는 상관 이름 S임)을 사용하여 목표 테이블 참조를 식별해야 합니다.

```
SQL statement:

   select s.s_name, s.s_address, s.s_phone, s.s_comment
     from "Tpcd".parts, "Tpcd".suppliers s, "Tpcd".partsupp ps
     where p_partkey = ps.ps_partkey and
       s.s_suppkey = ps.ps_suppkey and
       p_size = 39 and
       p_type = 'BRASS' and
       s.s_nation in ('MOROCCO', 'SPAIN') and
       ps.ps_supplycost = (select min(ps1.ps_supplycost)
                             from "Tpcd".partsupp ps1, "Tpcd".suppliers s1
                             where "Tpcd".parts.p_partkey = ps1.ps_partkey and
                             s1.s_suppkey = ps1.ps_suppkey and
                             s1.s_nation = s.s_nation)
     order by s.s_name

Optimization guideline:

   <OPTGUIDELINES>
     <IXSCAN TABLE='S' INDEX='I_SUPPKEY'/>
   </OPTGUIDELINES>
```

다음 인덱스 스캔 액세스 요청 요소는 옵티마이저가 명령문의 기본 subselect에서 PARTS 테이블에 대한 인덱스 액세스를 사용하도록 지정합니다. INDEX 속성이 없기 때문에 옵티마이저는 비용 기반 방식으로 인덱스를 선택합니다. 연관된 상관 이름이 없기 때문에 TABLE 속성은 규정된 테이블 이름을 사용하여 목표 테이블 참조를 나타냅니다.

```
   <OPTGUIDELINES>
     <IXSCAN TABLE='"Tpcd".PARTS'/>
   </OPTGUIDELINES>
```

다음 리스트 프리페치 액세스 요청은 LPREFETCH 액세스 요청 요소로 나타냅니다. 이 특정 요청은 옵티마이저가 I_SNATION 인덱스를 사용하여 명령문의 중첩된 subselect에서 SUPPLIERS 테이블에 액세스하도록 지정합니다. 중첩된 subselect에서 SUPPLIERS 테이블 참조를 식별하는 표시 이름이 있기 때문에 TABLE 속성은 상관 이름 S1을 사용합니다.

```
   <OPTGUIDELINES>
     <LPREFETCH TABLE='S1' INDEX='I_SNATION'/>
   </OPTGUIDELINES>
```

다음 인덱스 스캔 액세스 요청 요소는 옵티마이저가 I_SNAME 인덱스를 사용하여 기본 subselect의 SUPPLIERS 테이블에 액세스하도록 지정합니다. FIRST 속성은 이 테이블이 해당 FROM절에 대해 선택된 조인 시퀀스로 액세스하는 첫 번째 테이블이 되도록 지정합니다. 모든 액세스 또는 조인 요청에 FIRST 속성을 추가할 수 있습니다. 그러나 동일한 FROM절의 테이블을 참조하는 FIRST 속성을 가진 액세스 또는 조인 요청은 최대 하나일 수 있습니다.

```
SQL statement:

   select s.s_name, s.s_address, s.s_phone, s.s_comment
     from "Tpcd".parts, "Tpcd".suppliers s, "Tpcd".partsupp ps
     where p_partkey = ps.ps_partkey
       s.s_suppkey = ps.ps_suppkey and
       p_size = 39 and
       p_type = 'BRASS' and
       s.s_nation in ('MOROCCO', 'SPAIN') and
       ps.ps_supplycost = (select min(ps1.ps_supplycost)
                             from "Tpcd".partsupp ps1, "Tpcd".suppliers s1
                             where "Tpcd".parts.p_partkey = ps1.ps_partkey and
                               s1.s_suppkey = ps1.ps_suppkey and
                               s1.s_nation = s.s_nation)
     order by s.s_name
     optimize for 1 row

Optimization guidelines:

   <OPTGUIDELINES>
     <IXSCAN TABLE='S' INDEX='I_SNAME' FIRST='TRUE'/>
   </OPTGUIDELINES>
```

다음 예는 다중 액세스 요청이 단일 명령문 최적화 지침에서 전달되는 방법을 설명합니다. TBSCAN 액세스 요청 요소는 테이블 스캔 액세스 요청을 나타냅니다. 이 특정 요청은 전체 테이블 스캔을 사용하여 중첩된 subselect의 SUPPLIERS 테이블에 액세스하도록 지정합니다. LPREFETCH 액세스 요청 요소는 옵티마이저가 중첩된 subselect의 SUPPLIERS 테이블에 대한 리스트 프리페치 인덱스 액세스 중에 I_SUPPKEY 인덱스를 사용하도록 지정합니다.

```
   <OPTGUIDELINES>
     <TBSCAN TABLE='S1'/>
     <LPREFETCH TABLE='S' INDEX='I_SUPPKEY'/>
   </OPTGUIDELINES>
```

다음 예는 NLJOIN 조인 요청 요소가 나타내는 대로 중첩된 루프 조인 요청을 설명합니다. 일반적으로 조인 요청 요소에는 두 개의 하위 요소가 포함됩니다. 첫 번째 하위 요소는 조인 연산의 원하는 외부 입력을 나타내며, 두 번째 하위 요소는 조인 연산의 원하는 내부 입력을 나타냅니다. 하위 요소는 액세스 요청, 기타 조인 요청 또는 액세스 요청과 조인 요청의 조합일 수 있습니다. 이 예에서, 첫 번째 IXSCAN 액세스 요청 요소는 기본 subselect의 PARTS 테이블이 조인 연산의 외부 테이블이 되도록 지정합니다. 또한 인덱스 스캔을 사용하여 PARTS 테이블 액세스가 수행되도록 지정합니

다. 두 번째 IXSCAN 액세스 요청 요소는 기본 subselect의 PARTSUPP 테이블이 조인 연산의 내부 테이블이 되도록 지정합니다. 또한 인덱스 스캔을 사용하여 테이블에 액세스하도록 지정합니다.

```
<OPTGUIDELINES>
  <NLJOIN>
    <IXSCAN TABLE='"Tpcd".Parts'/>
    <IXSCAN TABLE="PS"/>
  </NLJOIN>
</OPTGUIDELINES>
```

다음 예는 HSJOIN 조인 요청 요소가 나타내는 대로 해시 조인 요청을 설명합니다. ACCESS 액세스 요청 요소는 중첩된 subselect의 SUPPLIERS 테이블이 조인 연산의 외부 테이블이 되도록 지정합니다. 이 액세스 요청 요소는 조인 순서 지정이 주요 목적인 경우 유용합니다. IXSCAN 액세스 요청 요소는 중첩된 subselect의 PARTSUPP 테이블이 조인 연산의 내부 테이블이 되고, 옵티마이저가 해당 테이블에 액세스하기 위해 인덱스 스캔을 선택하도록 지정합니다.

```
<OPTGUIDELINES>
  <HSJOIN>
    <ACCESS TABLE='S1'/>
    <IXSCAN TABLE='PS1'/>
  </HSJOIN>
</OPTGUIDELINES>
```

다음 예는 조인 요청을 중첩하여 대형 조인 요청을 구성하는 방법을 설명합니다. 이 예는 MSJOIN 조인 요청 요소가 나타내는 대로 병합 조인 요청을 포함합니다. 조인 연산의 외부 입력은 NLJOIN 조인 요청 요소가 나타내는 대로 기본 subselect의 PARTS 및 PARTSUPP 테이블의 결과입니다. 조인 연산의 내부 입력은 IXSCAN 액세스 요청 요소가 나타내는 대로 기본 subselect의 SUPPLIERS 테이블입니다.

```
<OPTGUIDELINES>
  <MSJOIN>
    <NLJOIN>
      <IXSCAN TABLE='"Tpcd".Parts'/>
      <IXSCAN TABLE="PS"/>
    </NLJOIN>
    <IXSCAN TABLE='S'/>
  </MSJOIN>
</OPTGUIDELINES>
```

조인 요청이 올바르면, 직접 또는 간접으로 내부에 중첩되는 모든 액세스 요청 요소는 최적화된 명령문의 동일한 FROM절의 테이블을 참조해야 합니다.

명령문 레벨 최적화 지침 작성:

명령문 프로파일의 명령문 레벨 최적화 지침 섹션은 명령문에서 테이블 액세스 또는 조인을 위한 방법을 지정하는 하나 이상의 액세스 또는 조인 요청으로 구성됩니다.

기타 모든 조정 옵션을 사용하십시오. 예를 들어, 다음과 같습니다.

1. 데이터 분산 통계가 runstats 유틸리티를 통해 최근에 갱신되었는지 확인하십시오.
2. 데이터 서버가 워크로드에 대한 적절한 최적화 클래스 설정을 사용하여 실행되고 있는지 확인하십시오.
3. 옵티마이저에 쿼리에서 참조되는 테이블에 액세스하기 위한 적절한 인덱스가 있는지 확인하십시오.

명령문 레벨 최적화 지침을 작성하려면 다음을 수행하십시오.

1. 명령문 레벨 지침을 삽입하려는 최적화 프로파일을 작성하십시오.
2. 명령문에 대해 Explain 기능을 실행하여 최적화 지침이 도움이 되는지 여부를 판별하십시오. 그런 것 같으면 계속 진행하십시오.
3. 다음과 유사한 쿼리를 실행하여 *원래* 명령문을 가져오십시오.

```
select statement_text
  from explain_statement
  where explain_level = '0' and
    explain_requester = 'SIMMEN' and
    explain_time      = '2003-09-08-16.01.04.108161' and
    source_name       = 'SQLC2E03' and
    source_version    = '' and
    queryno           = 1
```

4. 최적화 프로파일을 편집하고 명령문 텍스트를 명령문 키에 삽입하여 명령문 프로파일을 작성하십시오. 예를 들어, 다음과 같습니다.

```
<STMTPROFILE ID="Guidelines for TPCD Q9">
   <STMTKEY SCHEMA="TPCD"><![CDATA[SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE,
     S.S_COMMENT
     FROM PARTS P, SUPPLIERS S, PARTSUPP PS
     WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY
     AND P.P_SIZE  = 39 AND P.P_TYPE  = 'BRASS' AND S.S_NATION
     = 'MOROCCO' AND
     PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
     FROM PARTSUPP PS1, SUPPLIERS S1
     WHERE P.P_PARTKEY  = PS1.PS_PARTKEY AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
     AND S1.S_NATION  = S.S_NATION)]]>
   </STMTKEY>
</STMTPROFILE>
```

5. 명령문 키 뒤에 명령문 레벨 최적화 지침을 삽입하십시오. 표시 이름을 사용하여 액세스 및 조인 요청에서 참조되는 오브젝트를 식별하십시오. 다음은 조인 요청의 예입니다.

```
<OPTGUIDELINES>
  <HSJOIN>
    <TBSCAN TABLE='PS1'/>
    <IXSCAN TABLE='S1'
      INDEX='I1'/>
  </HSJOIN>
</OPTGUIDELINES>
```

6. 파일의 유효성을 확인하고 저장하십시오.

예상한 결과가 얻어지지 않을 경우, 상황에 맞게 지침을 변경하거나 추가 지침을 작성하고 최적화 프로파일을 갱신하십시오.

최적화 지침의 테이블 참조 구성:

용어 *테이블 참조*는 SQL문 또는 뷰 정의에서 테이블, 뷰, 테이블 표현식 또는 별명을 의미하는 데 사용됩니다. 최적화 지침은 최적화된 명령문의 테이블 참조와 연관된 고유 상관 이름 또는 원래 명령문의 해당 표시 이름을 사용하여 테이블 참조를 식별할 수 있습니다.

표시 이름의 시퀀스인 확장 이름은 뷰에 임베드된 테이블 참조를 고유하게 식별하는 데 도움이 됩니다. 전체 명령문의 컨텍스트 내에서 고유하지 않은 확장 이름 또는 표시 이름을 식별하는 최적화 지침은 앰비규어스로 간주되며 적용되지 않습니다. 또한 둘 이상의 최적화 지침이 동일한 테이블 참조를 식별할 경우, 해당 테이블 참조를 식별하는 모든 최적화 지침이 충돌하는 것으로 간주되며 적용되지 않습니다. 쿼리 변환의 가능성으로 인해 최적화 동안 표시 이름 또는 확장 이름이 계속 존재할지 보장할 수 없습니다. 따라서, 해당 테이블 참조를 식별하는 지침이 무시됩니다.

원래 명령문의 표시 이름을 사용하여 테이블 참조 식별

테이블의 표시 이름을 사용하여 테이블 참조가 식별됩니다. 표시 이름은 테이블이 SQL 문에서 규정되는 것과 동일한 방법으로 지정됩니다.

SQL ID를 지정하기 위한 규칙이 최적화 지침의 TABLE 속성 값에도 적용됩니다. TABLE 속성 값이 명령문의 각 표시 이름과 비교됩니다. 이 DB2 릴리스에서는 단일 일치만 허용됩니다. TABLE 속성 값이 스키마 규정된 경우, 동등한 규정된 표시 테이블 이름과 일치됩니다. TABLE 속성 값이 규정되지 않은 경우, 동등한 상관 이름 또는 표시 테이블 이름과 일치됩니다. 따라서 TABLE 속성 값이 명령문에 대해 적용되는 디폴트 스키마에 의해 내재적으로 규정된 것으로 간주됩니다. 이러한 개념은 다음 예에서 설명됩니다. 명령문이 디폴트 스키마 Tpcd를 사용하여 최적화된다고 가정해봅니다.

```
select s_name, s_address, s_phone, s_comment
  from parts, suppliers, partsupp ps
  where p_partkey = ps.ps_partkey and
    s.s_suppkey = ps.ps_suppkey and
    p_size = 39 and
    p_type = 'BRASS'
```

명령문에서 테이블 참조를 식별하는 TABLE 속성 값에는 '"Tpcd".PARTS', 'PARTS', 'Parts'가 포함됩니다(ID가 구분되지 않기 때문에 대문자로 변환됨). 명령문에서 테이블 참조를 식별하지 못하는 TABLE 속성 값에는 '"Tpcd2".SUPPLIERS', 'PARTSUPP'(표시 이름이 아님) 및 'Tpcd.PARTS'가 포함됩니다(ID Tpcd가 구분되어야 함. 그렇지 않을 경우 대문자로 변환됨).

표시 이름은 뷰, SQL 함수, 트리거 또는 원래 명령의 테이블 참조를 목표로 정하는 데 사용될 수 있습니다.

원래 명령문의 표시 이름을 사용하여 뷰의 테이블 참조 식별

다음 예에 표시된 대로 최적화 지침은 확장 구문을 사용하여 뷰에 임베드된 테이블 참조를 식별할 수 있습니다.

```
create view "Rick".v1 as
  (select * from employee a where salary > 50000)

create view "Gustavo".v2 as
  (select * from "Rick".v1
     where deptno in ('52', '53', '54')

select * from "Gustavo".v2 a
  where v2.hire_date > '01/01/2004'

<OPTGUIDELINES>
  <IXSCAN TABLE='A/"Rick".V1/A'/>
</OPTGUIDELINES>
```

IXSCAN 액세스 요청 요소는 ″Gustavo″.V2 및 ″Rick″.V1 뷰에 임베드된(embedded) EMPLOYEE 테이블 참조에 인덱스 스캔이 사용되도록 지정합니다. 뷰의 테이블 참조를 식별하기 위한 확장 구문은 슬래시 문자로 분리된 일련의 표시 이름입니다. TABLE 속성 A/″Rick″.V1/A 값은 확장 구문을 설명합니다. 시퀀스의 마지막 표시 이름(A)은 최적화 지침의 목표인 테이블 참조를 식별합니다. 시퀀스의 첫 번째 표시 이름(A)은 원래 명령문에서 직접 참조되는 뷰를 식별합니다. 중간에 있는 표시 이름(″Rick″.V1)은 직접 뷰 참조에서 목표 테이블 참조로의 경로를 따라 뷰 참조에 속합니다. 이전 섹션에 설명된 최적화 지침의 표시 이름을 참조하는 규칙은 확장 구문의 각 단계에 적용됩니다.

뷰의 EMPLOYEE 테이블 참조의 표시 이름이 직접적으로 또는 간접적으로 명령문이 참조하는 모든 테이블에 관하여 고유하다면 확장 이름 구문은 필요하지 않습니다.

확장 구문은 SQL 함수, 트리거 또는 원래 명령의 테이블 참조를 목표로 정하는 데 사용될 수 있습니다.

최적화된 명령문의 상관 이름을 사용하여 테이블 참조 식별

최적화 지침은 최적화된 명령문의 테이블 참조와 연관된 고유 상관 이름을 사용하여 테이블 참조를 식별할 수도 있습니다. 최적화된 명령문은 쿼리 재작성 최적화 단계 동안 판별된, 원래 명령문의 의미적으로 동등한 버전입니다. 최적화된 명령문은 Explain 테이블에서 검색할 수 있습니다. 최적화 지침의 TABID 속성은 최적화된 명령문에서 테이블 참조를 식별하는 데 사용됩니다.예를 들어, 다음과 같습니다.

Original statement:

```
select s.s_name, s.s_address, s.s_phone, s.s_comment
  from sm_tpcd.parts p, sm_tpcd.suppliers s, sm_tpcd.partsupp ps
  where p_partkey = ps.ps_partkey and
    s.s_suppkey = ps.ps_suppkey and
    p.p_size = 39 and
```

```
    p.p_type = 'BRASS' and
    s.s_nation in ('MOROCCO', 'SPAIN') and
    ps.ps_supplycost = (select min(ps1.ps_supplycost)
                        from sm_tpcd.partsupp ps1, sm_tpcd.suppliers s1
                        where p.p_partkey = ps1.ps_partkey and
                          s1.s_suppkey = ps1.ps_suppkey and
                          s1.s_nation = s.s_nation)

<OPTGUIDELINES>
  <HSJOIN>
    <TBSCAN TABLE='S1'/>
    <IXSCAN TABID='Q2'/>
  </HSJOIN>
</OPTGUIDELINES>
```

Optimized statement:

```
select q6.s_name as "S_NAME", q6.s_address as "S_ADDRESS",
    q6.s_phone as "S_PHONE", q6.s_comment as "S_COMMENT"
  from (select min(q4.$c0)
          from (select q2.ps_supplycost
                  from sm_tpcd.suppliers as q1, sm_tpcd.partsupp as q2
                  where q1.s_nation = 'MOROCCO' and
                    q1.s_suppkey = q2.ps_suppkey and
                    q7.p_partkey = q2.ps_partkey
                  ) as q3
          ) as q4, sm_tpcd.partsupp as q5, sm_tpcd.suppliers as q6,
              sm_tpcd.parts as q7
  where p_size = 39 and
    q5.ps_supplycost = q4.$c0 and
    q6.s_nation in ('MOROCCO', 'SPAIN') and
    q7.p_type = 'BRASS' and
    q6.s_suppkey = q5.ps_suppkey and
    q7.p_partkey = q5.ps_partkey
```

이 최적화 지침은 해시 조인 요청을 표시합니다(중첩된 subselect의 SUPPLIERS 테이블은 TBSCAN 액세스 요청 요소가 지정한 대로 외부 테이블이고 중첩된 subselect의 PARTSUPP 테이블은 IXSCAN 액세스 요청 요소가 지정한 대로 내부 테이블임). TBSCAN 액세스 요청 요소는 TABLE 속성을 사용하여 원래 명령문의 해당 표시 이름을 사용하는 SUPPLIERS 테이블 참조를 식별합니다. 반면에, IXSCAN 액세스 요청 요소는 TABID 속성을 사용하여 최적화된 명령문의 해당 테이블 참조와 연관된 고유 상관 이름을 사용하는 PARTSUPP 테이블 참조를 식별합니다.

단일 최적화 지침이 TABLE 및 TABID 속성을 모두 지정할 경우, 이러한 속성은 동일한 테이블 참조를 식별해야 합니다. 그렇지 않으면 최적화 지침이 무시됩니다.

주: 현재 새 릴리스의 DB2 제품으로 업그레이드할 경우 최적화된 명령문의 상관 이름이 정적일 것이라는 보장이 없습니다.

앰비규어스 테이블 참조

여러 표시 이름 또는 확장 이름과 일치할 경우 최적화 지침이 유효하지 않은 것으로 간주되거나 적용되지 않습니다. 예를 들어, 다음과 같습니다.

```
create view v1 as
  (select * from employee
     where salary > (select avg(salary) from employee)

select * from v1
  where deptno in ('M62', 'M63')

<OPTGUIDE>
  <IXSCAN TABLE='V1/EMPLOYEE'/>
</OPTGUIDE>
```

표시 이름 EMPLOYEE가 뷰 V1의 정의 내에서 고유하지 않으므로 옵티마이저는 IXSCAN 액세스 요청을 앰비규어스로 간주합니다.

모호성을 제거하려면 뷰를 다시 써 고유 상관 이름을 사용하거나 TABID 속성을 사용할 수 있습니다. TABID 속성에 의해 식별되는 테이블 참조는 최적화된 명령문의 모든 상관 이름이 고유하기 때문에 모호하지 않습니다.

충돌 최적화 지침

여러 최적화 지침이 동일한 테이블 참조를 식별할 수 없습니다. 예를 들어, 다음과 같습니다.

```
<OPTGUIDELINES>
  <IXSCAN TABLE='"Tpcd".PARTS' INDEX='I_PTYPE'/>
  <IXSCAN TABLE='"Tpcd".PARTS' INDEX='I_SIZE'/>
</OPTGUIDELINES>
```

각 IXSCAN 요소는 기본 subselect의 ″Tpcd″.PARTS 테이블을 참조합니다.

둘 이상의 지침이 동일한 테이블을 참조할 경우, 첫 번째 지침만 적용되고 기타 모든 지침은 무시되며 오류가 리턴됩니다.

쿼리당 술어 레벨에서 하나의 INLIST2JOIN 쿼리 재작성 요청 요소만 사용할 수 있습니다. 다음 예는 술어 레벨에서 두 개의 IN-LIST 술어를 사용할 수 있는 지원되지 않는 쿼리 재작성 최적화 지침을 설명합니다. 두 지침 모두 무시되며, 경고가 리턴됩니다.

```
<OPTGUIDELINES>
  <INLIST2JOIN TABLE='P' COLUMN='P_SIZE'/>
 <INLIST2JOIN TABLE='P' COLUMN='P_TYPE'/>
</OPTGUIDELINES>
```

최적화 지침이 사용되었는지 검증:

옵티마이저는 최적화 프로파일에 지정되어 있는 최적화 지침을 따르기 위해 모든 시도를 하지만, 유효하지 않거나 적용 가능하지 않은 지침을 거부할 수 있습니다.

Explain 기능을 사용할 수 있으려면 우선 Explain 테이블이 존재해야 합니다. Explain 테이블을 작성하기 위한 DDL(Data Definition Language)은 `sqllib` 디렉토리의 `misc` 서브디렉토리에서 찾을 수 있는 `EXPLAIN.DDL`에 포함되어 있습니다.

유효한 최적화 지침이 사용되었는지 검증하려면 다음을 수행하십시오.

1. 지침이 적용되는 명령문에 대해 EXPLAIN문을 발행하십시오. 최적화 프로파일을 사용하여 최적화 지침에 명령문에 적용되는 경우, 최적화 프로파일 이름이 EXPLAIN_ARGUMENT 테이블에서 RETURN 연산자 인수로 나타납니다. 그리고 최적화 지침에 현재 명령문과 일치하는 명령문 프로파일 또는 SQL 임베디드 최적화 지침이 포함된 경우, 명령문 프로파일의 이름이 RETURN 연산자 인수로 나타납니다. 이러한 두 새 인수 값의 유형은 OPT_PROF와 STMTPROF입니다.
2. Explain문의 결과를 검사하십시오. Explain 테이블에 대한 다음 쿼리는 EXPLAIN_REQUESTER, EXPLAIN_TIME, SOURCE_NAME, SOURCE_VERSION 및 QUERYNO의 특정 조합에 대한 명령문 프로파일 이름과 최적화 프로파일 이름을 리턴하도록 수정될 수 있습니다.

```
SELECT VARCHAR(B.ARGUMENT_TYPE,   9) as TYPE,
     VARCHAR(B.ARGUMENT_VALUE, 24) as VALUE

FROM   EXPLAIN_STATEMENT A, EXPLAIN_ARGUMENT B

WHERE  A.EXPLAIN_REQUESTER = 'SIMMEN'
  AND  A.EXPLAIN_TIME      = '2003-09-08-16.01.04.108161'
  AND  A.SOURCE_NAME       = 'SQLC2E03'
  AND  A.SOURCE_VERSION    = ''
  AND  A.QUERYNO           = 1

  AND  A.EXPLAIN_REQUESTER = B.EXPLAIN_REQUESTER
  AND  A.EXPLAIN_TIME      = B.EXPLAIN_TIME
  AND  A.SOURCE_NAME       = B.SOURCE_NAME
  AND  A.SOURCE_SCHEMA     = B.SOURCE_SCHEMA
  AND  A.SOURCE_VERSION    = B.SOURCE_VERSION
  AND  A.EXPLAIN_LEVEL     = B.EXPLAIN_LEVEL
  AND  A.STMTNO            = B.STMTNO
  AND  A.SECTNO            = B.SECTNO

  AND  A.EXPLAIN_LEVEL     = 'P'

  AND  (B.ARGUMENT_TYPE  = 'OPT_PROF' OR ARGUMENT_TYPE = 'STMTPROF')
  AND  B.OPERATOR_ID = 1
```

최적화 지침이 사용 중이고 Explain문이 최적화 지침의 STMTKEY 요소에 포함되어 있는 명령문과 일치할 경우, 이전 예와 유사한 쿼리가 다음 출력과 유사한 출력을 생성합니다. STMTPROF 인수의 값은 STMTPROFILE 요소의 ID 속성과 동일합니다.

```
TYPE      VALUE
--------- -------------------------
OPT_PROF  NEWTON.PROFILE1
STMTPROF  Guidelines for TPCD Q9
```

최적화 프로파일 및 지침에 대한 XML 스키마:

현재 최적화 프로파일 스키마:

주어진 DB2 릴리스에 대한 유효한 최적화 프로파일 컨텐츠는 현재 최적화 프로파일 스키마(COPS)로 알려진 XML 스키마에 의해 설명됩니다. 최적화 프로파일은 Linux, UNIX 및 Windows 서버용 DB2 Database에만 적용됩니다.

다음 목록은 DB2 제품의 현재 릴리스에 대한 COPS를 나타냅니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" version="1.0">
<!--*******************************************************************************************-->
<!-- IBM Confidential                                                                        -->
<!-- Licensed Materials - Property of IBM                                                    -->
<!-- (C) Copyright International Business Machines Corporation 2009. All rights reserved.    -->
<!-- U.S. Government Users Restricted Rights; Use, duplication or disclosure restricted by   -->
<!-- GSA ADP Schedule Contract with IBM Corp.                                                -->
<!--*******************************************************************************************-->
<!--*******************************************************************************************-->
<!-- Definition of the current optimization profile schema for V9.7.0.0                      -->
<!--                                                                                         -->
<!-- An optimization profile is composed of the following sections:                          -->
<!--                                                                                         -->
<!-- + A global optimization guidelines section (at most one) which defines optimization     -->
<!-- guidelines affecting any statement for which the optimization profile is in effect.     -->
<!--                                                                                         -->
<!-- + Zero or more statement profile sections, each of which defines optimization           -->
<!-- guidelines for a particular statement for which the optimization profile                -->
<!-- is in effect.                                                                           -->
<!--                                                                                         -->
<!-- The VERSION attribute indicates the version of this optimization profile                -->
<!-- schema.                                                                                 -->
<!--*******************************************************************************************-->
<xs:element name="OPTPROFILE">
   <xs:complexType>
      <xs:sequence>
         <!-- Global optimization guidelines section. At most one can be specified. -->
         <xs:element name="OPTGUIDELINES" type="globalOptimizationGuidelinesType" minOccurs="0"/>
         <!-- Statement profile section. Zero or more can be specified -->
         <xs:element name="STMTPROFILE" type="statementProfileType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <!-- Version attribute is currently optional -->
      <xs:attribute name="VERSION" use="optional">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="9.7.0.0"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
</xs:element>

<!--*******************************************************************************************-->
<!-- Global optimization guidelines supported in this version:                               -->
<!-- + MQTOptimizationChoices elements influence the MQTs considered by the optimizer.       -->
<!-- + computationalPartitionGroupOptimizationsChoices elements can affect repartitioning    -->
<!-- optimizations involving nicknames.                                                      -->
<!-- + General requests affect the search space which defines the alternative query          -->
<!-- transformations, access methods, join methods, join orders, and other optimizations,   -->
<!-- considered by the compiler and optimizer.                                               -->
<!-- + MQT enforcement requests specify semantically matchable MQTs whose usage in access    -->
<!-- plans should be enforced regardless of cost estimates.                                  -->
<!-- *******************************************************************************************-->
<xs:complexType name="globalOptimizationGuidelinesType">
   <xs:sequence>
      <xs:group ref="MQTOptimizationChoices" />
      <xs:group ref="computationalPartitionGroupOptimizationChoices" />
      <xs:group ref="generalRequest"/>
      <xs:group ref="mqtEnforcementRequest" />
   </xs:sequence>
</xs:complexType>
<!-- *******************************************************************************************-->
<!-- Elements for affecting materialized query table (MQT) optimization.                     -->
<!--                                                                                         -->
<!-- + MQTOPT - can be used to disable materialized query table (MQT) optimization.          -->
<!-- If disabled, the optimizer will not consider MQTs to optimize the statement.            -->
```

```
<!--                                                                                    -->
<!-- + MQT - multiple of these can be specified. Each specifies an MQT that should be  -->
<!-- considered for optimizing the statement. Only specified MQTs will be considered.  -->
<!--                                                                                    -->
<!--************************************************************************************-->
<xs:group name="MQTOptimizationChoices">
   <xs:choice>
      <xs:element name="MQTOPT" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="OPTION" type="optionType" use="optional"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="MQT" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
            <xs:attribute name="NAME" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
   </xs:choice>
</xs:group>
<!-- ************************************************************************************-->
<!-- Elements for affecting computational partition group (CPG) optimization.          -->
<!--                                                                                    -->
<!-- + PARTOPT - can be used disable the computational partition group (CPG) optimization   -->
<!-- which is used to dynamically redistributes inputs to join, aggregation,           -->
<!-- and union operations when those inputs are results of remote queries.             -->
<!--                                                                                    -->
<!-- + PART - Define the partition groups to be used in CPG optimizations.             -->
<!--                                                                                    -->
<!-- ************************************************************************************-->
<xs:group name="computationalPartitionGroupOptimizationChoices">
   <xs:choice>
      <xs:element name="PARTOPT" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="OPTION" type="optionType" use="optional"/>
      </xs:complexType>
      </xs:element>
      <xs:element name="PART" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="NAME" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
   </xs:choice>
</xs:group>
<!-- ************************************************************************************-->
<!-- Definition of a statement profile.                                                 -->
<!-- Comprised of a statement key and optimization guidelines.                          -->
<!-- The statement key specifies semantic information used to identify the statement to      -->
<!-- which optimization guidelines apply. The optional ID attribute provides the statement   -->
<!-- profile with a name for use in EXPLAIN output.                                     -->
<!-- ************************************************************************************-->
<xs:complexType name="statementProfileType">
   <xs:sequence>
      <!-- Statement key element -->
      <xs:element name="STMTKEY" type="statementKeyType"/>
      <xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
   </xs:sequence>
   <!-- ID attribute.Used in explain output to indicate the statement profile was used. -->
   <xs:attribute name="ID" type="xs:string" use="optional"/>
</xs:complexType>
<!--************************************************************************************-->
<!-- Definition of the statement key. The statement key provides semantic information used  -->
<!-- to identify the statement to which the optimization guidelines apply.              -->
<!-- The statement key is comprised of:                                                 -->
<!-- + statement text (as written in the application)                                   -->
<!-- + default schema (for resolving unqualified table names in the statement)          -->
<!-- + function path (for resolving unqualified types and functions in the statement)       -->
<!-- The statement text is provided as element data whereas the default schema and function -->
<!-- path are provided via the SCHEMA and FUNCPATH elements, respectively.              -->
<!--************************************************************************************-->
<xs:complexType name="statementKeyType" mixed="true">
   <xs:attribute name="SCHEMA" type="xs:string" use="optional"/>
   <xs:attribute name="FUNCPATH" type="xs:string" use="optional"/>
</xs:complexType>

<!--************************************************************************************-->
<!--                                                                                    -->
```

```
<!-- Optimization guideline elements can be chosen from general requests, rewrite          -->
<!-- requests access requests, or join requests.                                           -->
<!--                                                                                       -->
<!-- General requests affect the search space which defines the alternative query          -->
<!-- transformations, access methods, join methods, join orders, and other optimizations,  -->
<!-- considered by the optimizer.                                                          -->
<!--                                                                                       -->
<!-- Rewrite requests affect the query transformations used in determining the optimized   -->
<!-- statement.                                                                            -->
<!--                                                                                       -->
<!-- Access requests affect the access methods considered by the cost-based optimizer,     -->
<!-- and join requests affect the join methods and join order used in the execution plan.  -->
<!--                                                                                       -->
<!-- MQT enforcement requests specify semantically matchable MQTs whose usage in access    -->
<!-- plans should be enforced regardless of cost estimates.                                -->
<!--                                                                                       -->
<!--*****************************************************************************************-->
<xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
<xs:complexType name="optGuidelinesType">
   <xs:sequence>
      <xs:group ref="generalRequest" minOccurs="0" maxOccurs="1"/>
      <xs:choice maxOccurs="unbounded">
         <xs:group ref="rewriteRequest" />
         <xs:group ref="accessRequest"/>
         <xs:group ref="joinRequest"/>
         <xs:group ref="mqtEnforcementRequest"/>
      </xs:choice>
   </xs:sequence>
</xs:complexType>
<!--***************************************************************************************** -->
<!-- Choices of general request elements.                                                   -->
<!-- REOPT can be used to override the setting of the REOPT bind option.                    -->
<!-- DPFXMLMOVEMENT can be used to affect the optimizer's plan when moving XML documents   -->
<!-- between database partitions. The value can be NONE, REFERENCE or COMBINATION. The     -->
<!-- default value is NONE.                                                                 -->
<!--***************************************************************************************** -->
<xs:group name="generalRequest">
   <xs:sequence>
      <xs:element name="REOPT" type="reoptType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="DEGREE" type="degreeType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="QRYOPT" type="qryoptType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="RTS" type="rtsType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="DPFXMLMOVEMENT" type="dpfXMLMovementType" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
</xs:group>
<!--*****************************************************************************************-->
<!-- Choices of rewrite request elements.                                                -->
<!--*****************************************************************************************-->
<xs:group name="rewriteRequest">
   <xs:sequence>
      <xs:element name="INLIST2JOIN" type="inListToJoinType" minOccurs="0"/>
      <xs:element name="SUBQ2JOIN" type="subqueryToJoinType" minOccurs="0"/>
      <xs:element name="NOTEX2AJ" type="notExistsToAntiJoinType" minOccurs="0"/>
      <xs:element name="NOTIN2AJ" type="notInToAntiJoinType" minOccurs="0"/>
   </xs:sequence>
</xs:group>
<!--***************************************************************************************** -->
<!-- Choices for access request elements.                                                   -->
<!-- TBSCAN - table scan access request element                                             -->
<!-- IXSCAN - index scan access request element                                             -->
<!-- LPREFETCH - list prefetch access request element                                       -->
<!-- IXAND - index ANDing access request element                                            -->
<!-- IXOR - index ORing access request element                                              -->
<!-- XISCAN - xml index access request element                                              -->
<!-- XANDOR - XANDOR access request element                                                 -->
<!-- ACCESS - indicates the optimizer should choose the access method for the table         -->
<!--***************************************************************************************** -->
<xs:group name="accessRequest">
   <xs:choice>
      <xs:element name="TBSCAN" type="tableScanType"/>
      <xs:element name="IXSCAN" type="indexScanType"/>
      <xs:element name="LPREFETCH" type="listPrefetchType"/>
      <xs:element name="IXAND" type="indexAndingType"/>
      <xs:element name="IXOR" type="indexOringType"/>
      <xs:element name="XISCAN" type="indexScanType"/>
      <xs:element name="XANDOR" type="XANDORType"/>
```

```
      <xs:element name="ACCESS" type="anyAccessType"/>
   </xs:choice>
</xs:group>
<!--**************************************************************************** -->
<!-- Choices for join request elements.                                         -->
<!-- NLJOIN - nested-loops join request element                                 -->
<!-- MSJOIN - sort-merge join request element                                   -->
<!-- HSJOIN - hash join request element                                         -->
<!-- JOIN - indicates that the optimizer is to choose the join method.          -->
<!--**************************************************************************** -->
<xs:group name="joinRequest">
   <xs:choice>
      <xs:element name="NLJOIN" type="nestedLoopJoinType"/>
      <xs:element name="HSJOIN" type="hashJoinType"/>
      <xs:element name="MSJOIN" type="mergeJoinType"/>
      <xs:element name="JOIN" type="anyJoinType"/>
   </xs:choice>
</xs:group>
<!--****************************************************************************-->
<!-- MQT enforcement request element.                                         -->
<!-- MQTENFORCE - This element can be used to specify semantically matchable MQTs whose  -->
<!--   usage in access plans should be enforced regardless of Optimizer cost estimates.  -->
<!--   MQTs can be specified either directly with the NAME attribute or generally using  -->
<!--   the TYPE attribute.                                                      -->
<!--   Only the first valid attribute found is used and all subsequent ones are ignored. -->
<!--   Since this element can be specified multiple times, more than one MQT can be      -->
<!--   enforced at a time.                                                      -->
<!--   Note however, that if there is a conflict when matching two enforced MQTs to the  -->
<!--   same data source (base-table or derived) an MQT will be picked based on existing  -->
<!--   tie-breaking rules, i.e., either heuristic or cost-based.                -->
<!--   Finally, this request overrides any other MQT optimization options specified in   -->
<!--   a profile, i.e., enforcement will take place even if MQTOPT is set to DISABLE or  -->
<!--   if the specified MQT or MQTs do not exist in the eligibility list specified by    -->
<!--   any MQT elements.                                                        -->
<!--****************************************************************************-->
<xs:group name="mqtEnforcementRequest">
   <xs:sequence>
      <xs:element name="MQTENFORCE" type="mqtEnforcementType" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:group>
<!--**************************************************************************** -->
<!-- REOPT general request element. Can override REOPT setting at the package, db,           -->
<!-- dbm level.                                                                 -->
<!--**************************************************************************** -->
<xs:complexType name="reoptType">
   <xs:attribute name="VALUE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ONCE"/>
            <xs:enumeration value="ALWAYS"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
<!--****************************************************************************-->
<!-- RTS general request element to enable, disable or provide a time budget for             -->
<!-- real-time statistics collection.                                            -->
<!-- OPTION attribute allows enabling or disabling real-time statistics.         -->
<!-- TIME attribute provides a time budget in milliseconds for real-time statistics collection.-->
<!--****************************************************************************-->
<xs:complexType name="rtsType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="TIME" type="xs:nonNegativeInteger" use="optional"/>
</xs:complexType>
<!--****************************************************************************-->
<!-- Definition of an "IN list to join" rewrite request                         -->
<!-- OPTION attribute allows enabling or disabling the alternative.             -->
<!-- TABLE attribute allows request to target IN list predicates applied to a   -->
<!-- specific table reference. COLUMN attribute allows request to target a specific IN list   -->
<!-- predicate.                                                                 -->
<!--****************************************************************************-->
<xs:complexType name="inListToJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="TABLE" type="xs:string" use="optional"/>
   <xs:attribute name="COLUMN" type="xs:string" use="optional"/>
</xs:complexType>
```

```
<!--*****************************************************************************************-->
<!-- Definition of a "subquery to join" rewrite request                                      -->
<!-- The OPTION attribute allows enabling or disabling the alternative.                     -->
<!--*****************************************************************************************-->
<xs:complexType name="subqueryToJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of a "not exists to anti-join" rewrite request                              -->
<!-- The OPTION attribute allows enabling or disabling the alternative.                     -->
<!--*****************************************************************************************-->
<xs:complexType name="notExistsToAntiJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of a "not IN to anti-join" rewrite request                                  -->
<!-- The OPTION attribute allows enabling or disabling the alternative.                     -->
<!--*****************************************************************************************-->
<xs:complexType name="notInToAntiJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Effectively the superclass from which all access request elements inherit.             -->
<!-- This type currently defines TABLE and TABID attributes, which can be used to tie an    -->
<!-- access request to a table reference in the query.                                      -->
<!-- The TABLE attribute value is used to identify a table reference using identifiers      -->
<!-- in the original SQL statement. The TABID attribute value is used to identify a table   -->
<!-- referece using the unique correlation name provided via the                            -->
<!-- optimized statement. If both the TABLE and TABID attributes are specified, the TABID   -->
<!-- field is ignored. The FIRST attribute indicates that the access should be the first    -->
<!-- access in the join sequence for the FROM clause.                                       -->
<!-- The SHARING attribute indicates that the access should be visible to other concurrent  -->
<!-- similar accesses that may therefore share bufferpool pages. The WRAPPING attribute     -->
<!-- indicates that the access should be allowed to perform wrapping, thereby allowing it to -->
<!-- start in the middle for better sharing with other concurrent accesses. The THROTTLE    -->
<!-- attribute indicates that the access should be allowed to be throttled if this may      -->
<!-- benefit other concurrent accesses. The SHARESPEED attribute is used to indicate whether -->
<!-- the access should be considered fast or slow for better grouping of concurrent accesses. -->
<!--***************************************************************************************** -->
<xs:complexType name="accessType" abstract="true">
   <xs:attribute name="TABLE" type="xs:string" use="optional"/>
   <xs:attribute name="TABID" type="xs:string" use="optional"/>
   <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
   <xs:attribute name="SHARING" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="WRAPPING" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="THROTTLE" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="SHARESPEED" type="shareSpeed" use="optional"/>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of an table scan access request method.                                     -->
<!--*****************************************************************************************-->
<xs:complexType name="tableScanType">
   <xs:complexContent>
      <xs:extension base="accessType"/>
   </xs:complexContent>
</xs:complexType>
<!-- *****************************************************************************************-->
<!-- Definition of an index scan access request element. The index name is optional.       -->
<!--***************************************************************************************** -->
<xs:complexType name="indexScanType">
   <xs:complexContent>
      <xs:extension base="accessType">
         <xs:attribute name="INDEX" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of a list prefetch access request element. The index name is optional.     -->
<!--***************************************************************************************** -->
<xs:complexType name="listPrefetchType">
   <xs:complexContent>
      <xs:extension base="accessType">
         <xs:attribute name="INDEX" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

```
<!--*****************************************************************************-->
<!-- Definition of an extended access element which will be used by IXAND and ACCESS      -->
<!-- requests.                                                                            -->
<!-- A single index scan be specified via the INDEX attribute. Multiple indexes           -->
<!-- can be specified via INDEX elements. The index element specification supersedes the  -->
<!-- attribute specification. If a single index is specified, the optimizer will use the  -->
<!-- index as the first index of the index ANDing access method and will choose addi-     -->
<!-- tional indexes using cost. If multiple indexes are specified the optimizer will      -->
<!-- use exactly those indexes in the specified order. If no indexes are specified        -->
<!-- via either the INDEX attribute or INDEX elements, then the optimizer will choose     -->
<!-- all indexes based upon cost.                                                         -->
<!-- Extension for XML support:                                                           -->
<!-- TYPE: Optional attribute. The allowed value is XMLINDEX. When the type is not        -->
<!-- specified, the optimizer makes a cost based decision.                                -->
<!-- ALLINDEXES: Optional attribute. The allowed value is TRUE. The default               -->
<!-- value is FALSE.                                                                      -->
<!--***************************************************************************** -->
<xs:complexType name="extendedAccessType">
   <xs:complexContent>
      <xs:extension base="accessType">
         <xs:sequence minOccurs="0">
            <xs:element name="INDEX" type="indexType" minOccurs="2" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="INDEX" type="xs:string" use="optional"/>
         <xs:attribute name="TYPE" type="xs:string" use="optional" fixed="XMLINDEX"/>
         <xs:attribute name="ALLINDEXES" type="boolType" use="optional" fixed="TRUE"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************-->
<!-- Definition of an index ANDing access request element.                        -->
<!-- Extension for XML support:                                                   -->
<!-- All attributes and elements in extendedAccessType are included.              -->
<!-- Note that ALLINDEXES is a valid option only if TYPE is XMLINDEX.             -->
<!--***************************************************************************** -->
<xs:complexType name="indexAndingType">
   <xs:complexContent>
         <xs:extension base="extendedAccessType"/>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************-->
<!-- Definition of an INDEX element method. Index set is optional. If specified,  -->
<!-- at least 2 are required.                                                     -->
<!--***************************************************************************** -->
<xs:complexType name="indexType">
   <xs:attribute name="IXNAME" type="xs:string" use="optional"/>
</xs:complexType>
<!--*****************************************************************************-->
<!-- Definition of an XANDOR access request method.                               -->
<!--*****************************************************************************-->
<xs:complexType name="XANDORType">
   <xs:complexContent>
      <xs:extension base="accessType"/>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************-->
<!-- Use for derived table access or other cases where the access method is not of -->
<!-- consequence.                                                                 -->
<!-- Extension for XML support:                                                   -->
<!-- All attributes and elements in extendedAccessType are included.              -->
<!-- Note that INDEX attribute/elements and ALLINDEXES are valid options only if TYPE -->
<!-- is XMLINDEX.                                                                 -->
<!--*****************************************************************************-->
<xs:complexType name="anyAccessType">
   <xs:complexContent>
      <xs:extension base="extendedAccessType"/>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************-->
<!-- Definition of an index ORing access                                          -->
<!-- Cannot specify more details (e.g indexes). Optimizer will choose the details based -->
<!-- upon cost.                                                                   -->
<!--***************************************************************************** -->
<xs:complexType name="indexOringType">
   <xs:complexContent>
      <xs:extension base="accessType"/>
```

```
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Effectively the super class from which join request elements inherit.                   -->
<!-- This type currently defines join element inputs and also the FIRST attribute.           -->
<!-- A join request must have exactly two nested sub-elements. The sub-elements can be       -->
<!-- either an access request or another join request. The first sub-element represents      -->
<!-- outer table of the join operation while the second element represents the inner         -->
<!-- table. The FIRST attribute indicates that the join result should be the first join      -->
<!-- relative to other tables in the same FROM clause.                                       -->
<!--***************************************************************************************** -->
<xs:complexType name="joinType" abstract="true">
   <xs:choice minOccurs="2" maxOccurs="2">
      <xs:group ref="accessRequest"/>
      <xs:group ref="joinRequest"/>
   </xs:choice>
   <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
</xs:complexType>
<!--***************************************************************************************** -->
<!-- Definition of nested loop join access request. Subclass of joinType.                    -->
<!-- Does not add any elements or attributes.                                                -->
<!--***************************************************************************************** -->
<xs:complexType name="nestedLoopJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of merge join access request. Subclass of joinType.                          -->
<!-- Does not add any elements or attributes.                                                -->
<!--***************************************************************************************** -->
<xs:complexType name="mergeJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
<!--***************************************************************************************** -->
<!-- Definition of hash join access request. Subclass of joinType.                           -->
<!-- Does not add any elements or attributes.                                                -->
<!--***************************************************************************************** -->
<xs:complexType name="hashJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
<!--***************************************************************************************** -->
<!-- Any join is a subclass of binary join. Does not extend it in any way.                   -->
<!-- Does not add any elements or attributes.                                                -->
<!--***************************************************************************************** -->
<xs:complexType name="anyJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- The MQTENFORCE element can be specified with one of two attributes:                     -->
<!--   NAME: Specify the MQT name directly as a value to this attribute.                     -->
<!--   TYPE: Specify the type of the MQTs that should be enforced with this attribute.       -->
<!--   Note that only the value of the first valid attribute found will be used. All         -->
<!--   subsequent attributes will be ignored.                                                -->
<!--*****************************************************************************************-->
<xs:complexType name="mqtEnforcementType">
   <xs:attribute name="NAME" type="xs:string"/>
   <xs:attribute name="TYPE" type="mqtEnforcementTypeType"/>
</xs:complexType>
<!--*******************************************************************************************-->
<!-- Allowable values for the TYPE attribute of an MQTENFORCE element:                       -->
<!--   NORMAL: Enforce usage of all semantically matchable MQTs, except replicated MQTs.     -->
<!--   REPLICATED: Enforce usage of all semantically matchable replicated MQTs only.         -->
<!--   ALL: Enforce usage of all semantically matchable MQTs.                                -->
<!--***************************************************************************************** -->
<xs:simpleType name="mqtEnforcementTypeType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="NORMAL"/>
      <xs:enumeration value="REPLICATED"/>
      <xs:enumeration value="ALL"/>
```

```
   </xs:restriction>
</xs:simpleType>
<!--*****************************************************************************************-->
<!-- Allowable values for a boolean attribute.                                               -->
<!--***************************************************************************************** -->
<xs:simpleType name="boolType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="TRUE"/>
      <xs:enumeration value="FALSE"/>
   </xs:restriction>
</xs:simpleType>
<!--*****************************************************************************************-->
<!-- Allowable values for an OPTION attribute.                                               -->
<!--***************************************************************************************** -->
<xs:simpleType name="optionType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="ENABLE"/>
      <xs:enumeration value="DISABLE"/>
   </xs:restriction>
</xs:simpleType>
<!--*****************************************************************************************-->
<!-- Allowable values for a SHARESPEED attribute.                                            -->
<!--***************************************************************************************** -->
<xs:simpleType name="shareSpeed">
   <xs:restriction base="xs:string">
      <xs:enumeration value="FAST"/>
      <xs:enumeration value="SLOW"/>
   </xs:restriction>
</xs:simpleType>
<!--*****************************************************************************************-->
<!-- Definition of the qryopt type: the only values allowed are 0, 1, 2, 3, 5, 7 and 9       -->
<!--*****************************************************************************************-->
<xs:complexType name="qryoptType">
   <xs:attribute name="VALUE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="0"/>
            <xs:enumeration value="1"/>
            <xs:enumeration value="2"/>
            <xs:enumeration value="3"/>
            <xs:enumeration value="5"/>
            <xs:enumeration value="7"/>
            <xs:enumeration value="9"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of the degree type: any number between 1 and 32767 or the strings ANY or -1  -->
<!--***************************************************************************************** -->
<xs:simpleType name="intStringType">
   <xs:union>
      <xs:simpleType>
         <xs:restriction base="xs:integer">
            <xs:minInclusive value="1"></xs:minInclusive>
            <xs:maxInclusive value="32767"></xs:maxInclusive>
         </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ANY"/>
            <xs:enumeration value="-1"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:union>
</xs:simpleType>

<xs:complexType name="degreeType">
        <xs:attribute name="VALUE" type="intStringType"></xs:attribute>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of DPF XML movement types                                                    -->
<!--***************************************************************************************** -->
<xs:complexType name="dpfXMLMovementType">
    <xs:attribute name="VALUE" use="required">
     <xs:simpleType>
```

```
      <xs:restriction base="xs:string">
        <xs:enumeration value="REFERENCE"/>
        <xs:enumeration value="COMBINATION"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
 </xs:complexType>

</xs:schema>
```

OPTPROFILE 요소에 대한 XML 스키마:

OPTPROFILE 요소는 최적화 프로파일의 루트입니다.

이 요소는 다음과 같이 정의됩니다.

XML Schema

```
<xs:element name="OPTPROFILE">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="OPTGUIDELINES" type="globalOptimizationGuidelinesType"
           minOccurs="0"/>
         <xs:element name="STMTPROFILE" type="statementProfileType" minOccurs="0"
           maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="VERSION" use="optional">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="9.7.0.0"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
</xs:element>
```

설명

선택적 OPTGUIDELINES 하위 요소는 최적화 프로파일에 대한 전역 최적화 지침을 정의합니다. 각 STMTPROFILE 하위 요소는 명령문 프로파일을 정의합니다. VERSION 속성은 특정 최적화 프로파일이 작성되고 유효성 확인된 현재 최적화 프로파일 스키마를 식별합니다.

전역 OPTGUIDELINES 요소에 대한 XML 스키마:

OPTGUIDELINES 요소는 최적화 프로파일에 대한 전역 최적화 지침을 정의합니다.

이것은 복합 유형 globalOptimizationGuidelinesType으로 정의됩니다.

XML Schema

```
<xs:complexType name="globalOptimizationGuidelinesType">
   <xs:sequence>
      <xs:group ref="MQTOptimizationChoices"/>
      <xs:group ref="computationalPartitionGroupOptimizationChoices"/>
      <xs:group ref="generalRequest"/>
      <xs:group ref="mqtEnforcementRequest"/>
   </xs:sequence>
</xs:complexType>
```

설명

전역 최적화 지침은 computationalPartitionGroupOptimizationChoices 그룹 요소, MQTOptimizationChoices 그룹 요소 또는 REOPT 일반 요청 요소를 사용하여 정의될 수 있습니다.

- MQTOptimizationChoices 그룹 요소는 MQT 대체에 영향을 주는 데 사용될 수 있습니다.
- computationalPartitionGroupOptimizationChoices 그룹 요소는 리모트 데이터 소스에서 데이터 읽기의 동적 재분산과 관련된 전산 파티션 그룹 최적화에 영향을 주는 데 사용될 수 있습니다. 이 요소는 파티션된 페더레이티드 데이터베이스 구성에만 적용됩니다.
- REOPT 일반 요청 요소는 변수를 참조하는 명령문에 대해 최적화가 발생하는 시간에 영향을 주는 데 사용될 수 있습니다.
- MQT 강제 요청은 액세스 플랜에서의 사용을 비용 추정에 관계없이 강제해야 하는 구체화된 일치 가능한 쿼리 테이블(MQT)을 시맨틱으로 지정합니다.

MQT 최적화 선택사항:

MQTOptimizationChoices 그룹은 구체화된 쿼리 테이블(MQT) 최적화에 영향을 주는 데 사용될 수 있는 요소 세트를 정의합니다. 특히, MQT 대체의 고려사항을 사용/사용하지 않도록 하거나 옵티마이저에 의해 고려될 전체 MQT 세트를 지정하는 데 이러한 요소를 사용할 수 있습니다.

XML Schema

```
<xs:group name="MQTOptimizationChoices">
   <xs:choice>
      <xs:element name="MQTOPT" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="OPTION" type="optionType" use="optional"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="MQT" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
            <xs:attribute name="NAME" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
   </xs:choice>
</xs:group>
```

설명

MQTOPT 요소는 MQT 최적화의 고려사항을 사용하거나 사용하지 않도록 하는 데 사용됩니다. OPTION 속성은 값 ENABLE(디폴트) 또는 DISABLE을 취할 수 있습니다.

MQT 요소의 NAME 속성은 옵티마이저에 의해 고려될 MQT를 식별합니다. NAME 속성에서 MQT에 대한 참조를 구성하기 위한 규칙은 표시 테이블 이름에 대한 참조를

구성하기 위한 규칙과 동일합니다. 하나 이상의 MQT 요소가 지정된 경우, 이러한 MQT만 옵티마이저에 의해 고려됩니다. 하나 이상의 지정된 MQT를 사용하여 MQT 대체를 수행하려는 결정은 비용 기반으로 수행됩니다.

예

다음 예에서는 MQT 최적화를 사용하지 않도록 하는 방법을 보여줍니다.

```
<OPTGUIDELINES>
   <MQTOPT OPTION='DISABLE'/>
</OPTGUIDELINES>
```

다음 예제는 MQT 최적화를 Tpcd.PARTSMQT 테이블 및 COLLEGE.STUDENTS 테이블로 제한하는 방법을 보여줍니다.

```
<OPTGUIDELINES>
   <MQT NAME='Tpcd.PARTSMQT'/>
   <MQT NAME='COLLEGE.STUDENTS'/>
<OPTGUIDELINES>
```

전산 파티션 그룹 최적화 선택사항:

computationalPartitionGroupOptimizationChoices 그룹은 전산 파티션 그룹 최적화에 영향을 주는 데 사용될 수 있는 요소 세트를 정의합니다. 특히, 전산 그룹 최적화를 사용/사용하지 않도록 하거나 전산 파티션 그룹 최적화에 사용될 파티션 그룹을 지정하는 데 이러한 요소를 사용할 수 있습니다.

XML Schema

```
<xs:group name="computationalPartitionGroupOptimizationChoices">
   <xs:choice>
      <xs:element name="PARTOPT" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="OPTION" type="optionType" use="optional"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="PART" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="NAME" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
   </xs:choice>
</xs:group>
```

설명

PARTOPT 요소는 전산 파티션 그룹 최적화의 고려사항을 사용하거나 사용하지 않도록 하는 데 사용됩니다. OPTION 속성은 값 ENABLE(디폴트) 또는 DISABLE을 취할 수 있습니다.

PART 요소를 사용하여 전산 파티션 그룹 최적화에 사용될 파티션 그룹을 지정할 수 있습니다. NAME 속성은 기존 파티션 그룹을 식별해야 합니다. 지정된 파티션 그룹을 사용하여 동적 재분산을 수행하려는 결정은 비용 기반으로 수행됩니다.

예

다음 예에서는 전산 파티션 그룹 최적화를 사용하지 않도록 하는 방법을 보여줍니다.

```
<OPTGUIDELINES>
   <PARTOPT OPTION='DISABLE'/>
</OPTGUIDELINES>
```

다음 예에서는 전산 파티션 그룹 최적화에서 WORKPART 파티션 그룹을 사용하도록 지정하는 방법을 보여줍니다.

```
<OPTGUIDELINES>
   <MQT NAME='Tpcd.PARTSMQT'/>
   <PART NAME='WORKPART'/>
<OPTGUIDELINES>
```

REOPT 전역 최적화 지침:

REOPT 전역 최적화 지침은 변수(호스트 변수, 매개변수 표시문자, 전역 변수 또는 특수 레지스터)를 포함하는 데이터 처리 언어(DML) 명령문에 대해 최적화가 발생하는 시기를 제어합니다.

REOPT 요소의 설명 및 구문은 전역 최적화 지침과 명령문 레벨 최적화 지침 모두에서 동일합니다. 자세한 정보는 『REOPT 요청』을 참조하십시오.

STMTPROFILE 요소에 대한 XML 스키마:

STMTPROFILE 요소는 최적화 프로파일 내 명령문 프로파일을 정의합니다.

이것은 복합 유형 statementProfileType에 의해 정의됩니다.

```
XML Schema

<xs:complexType name="statementProfileType">
   <xs:sequence>
      <xs:element name="STMTKEY" type="statementKeyType"/>
      <xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
   </xs:sequence>
   <xs:attribute name="ID" type="xs:string" use="optional"/>
</xs:complexType>
```

설명

명령문 프로파일은 특정 명령문에 대한 최적화 지침을 지정하고, 다음 부분을 포함합니다.

- 명령문 키

  최적화 프로파일은 응용프로그램에서 둘 이상의 명령문에 적용될 수 있습니다. 명령문 키를 사용하여 옵티마이저는 각 명령문 프로파일을 응용프로그램의 해당 명령문에 자동으로 일치시킵니다. 이를 통해 응용프로그램을 편집하지 않고 명령문에 대한 최적화 지침을 제공할 수 있습니다. 명령문 키는 명령문의 텍스트(응용프로그램에서

작성된)뿐 아니라 올바른 명령문을 명백하게 식별하는 데 필요한 기타 정보도 포함합니다. STMTKEY 하위 요소는 명령문 키를 나타냅니다.

- 명령문 레벨 최적화 지침

  명령문 프로파일의 이 부분은 명령문 키에 의해 식별되는 명령문에 대해 적용되는 최적화 지침을 지정합니다. 자세한 정보는 『명령문 레벨 OPTGUIDELINES 요소에 대한 XML 스키마』를 참조하십시오.

- 명령문 프로파일 이름

  특정 명령문 프로파일을 식별하기 위해 진단 출력에 표시되는 사용자 지정 이름입니다.

STMTKEY 요소에 대한 XML 스키마:

STMTKEY 요소는 옵티마이저가 명령문 프로파일을 응용프로그램의 해당 명령문에 일치시킬 수 있도록 해 줍니다.

이것은 복합 유형 statementKeyType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="statementKeyType" mixed="true">
   <xs:attribute name="SCHEMA" type="xs:string" use="optional"/>
   <xs:attribute name="FUNCPATH" type="xs:string" use="optional"/>
</xs:complexType>
</xs:schema>
```

설명

선택적 SCHEMA 속성을 사용하여 명령문 키의 디폴트 스키마 부분을 지정할 수 있습니다.

선택적 FUNCPATH 속성을 사용하여 명령문 키의 함수 경로 부분을 지정할 수 있습니다. 다중 경로는 쉼표로 구분되어야 하고, 지정된 함수 경로는 컴파일 키에 지정된 함수 경로와 정확히 일치해야 합니다.

예

다음 예에서는 디폴트 스키마가 'COLLEGE'이고 함수 경로가 'SYSIBM,SYSFUN,SYSPROC,DAVE'인 특정 명령문과 연관된 명령문 키 정의를 보여줍니다.

```
<STMTKEY SCHEMA='COLLEGE' FUNCPATH='SYSIBM,SYSFUN,SYSPROC,DAVE'>
   <![CDATA[select * from orders" where foo(orderkey) > 20]]>
</STMTKEY>
```

명령문 텍스트에 특수 XML 문자 '>'가 포함되어 있기 때문에 CDATA 태깅(<![CDATA[로 시작하고]]>로 끝남)이 필요합니다.

명령문 키 및 컴파일 키 일치:

명령문 키는 명령문 레벨 최적화 지침이 적용되는 응용프로그램 명령문을 식별하는 데 사용됩니다.

SQL문이 컴파일될 때 다양한 요인이 컴파일러에 의해 명령문이 의미적으로 해석되는 방법에 영향을 미칩니다. SQL문과 SQL 컴파일러 매개변수의 설정이 함께 컴파일 키를 구성합니다. 명령문 키의 각 부분은 컴파일 키의 일부 부분에 해당합니다.

명령문 키는 다음 부분으로 구성됩니다.

- 명령문 텍스트: 응용프로그램에서 작성된 명령문의 텍스트
- 디폴트 스키마: 규정되지 않은 테이블 이름에 대한 내재된 규정자로 사용되는 스키마 이름. 이 부분은 선택적이지만, 명령문에 규정되지 않은 테이블 이름이 있는 경우 제공되어야 합니다.
- 함수 경로: 규정되지 않은 함수와 데이터 유형 참조를 확인할 때 사용되는 함수 경로. 이 부분은 선택적이지만, 명령문에 규정되지 않은 사용자 정의 함수(UDF) 또는 사용자 정의 유형이 있는 경우 제공되어야 합니다.

데이터 서버에서 SQL문을 컴파일하고 활성 최적화 프로파일을 찾으면, 최적화 프로파일의 각 명령문 키를 현재 컴파일 키와 일치시키려고 시도합니다. 명령문 키의 각 지정된 부분이 컴파일 키의 해당 부분과 일치할 경우 명령문 키와 컴파일 키가 일치한다고 합니다. 명령문 키의 일부가 지정되지 않은 경우, 생략된 부분은 디폴트로 일치하는 것으로 간주됩니다. 명령문 키의 각 지정되지 않은 부분은 컴파일 키의 해당 부분과 일치하는 와일드 카드로 처리됩니다.

데이터 서버에서 현재 컴파일 키와 일치하는 명령문 키를 찾은 후에는 검색을 중지합니다. 명령문 키가 현재 컴파일 키와 일치하는 명령문 프로파일이 여러 개인 경우, 첫 번째 해당 명령문 프로파일(문서 순서 기반)만 사용됩니다.

명령문 레벨 OPTGUIDELINES 요소에 대한 XML 스키마:

명령문 프로파일의 OPTGUIDELINES 요소는 연관된 명령문 키에 의해 식별되는 명령문에 대해 적용되는 최적화 지침을 정의합니다. 이것은 유형 optGuidelinesType에 의해 정의됩니다.

XML Schema

```
<xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
<xs:complexType name="optGuidelinesType">
   <xs:sequence>
      <xs:group ref="general request" minOccurs="0" maxOccurs="1"/>
      <xs:choice maxOccurs="unbounded">
         <xs:group ref="rewriteRequest"/>
         <xs:group ref="accessRequest"/>
         <xs:group ref="joinRequest"/>
```

```
            <xs:group ref="mqtEnforcementRequest"/>
          </xs:choice>
      </xs:sequence>
    </xs:complexType>
```

설명

optGuidelinesType 그룹은 OPTGUIDELINES 요소의 유효한 하위 요소 세트를 정의합니다. 각 하위 요소는 DB2 옵티마이저에 의해 최적화 지침으로 해석됩니다. 하위 요소는 일반 요청 요소, 재작성 요청 요소, 액세스 요청 요소 또는 조인 요청 요소로 구분될 수 있습니다.

- *일반 요청 요소*는 옵티마이저의 검색 스페이스를 변경하는 데 사용할 수 있는 일반 최적화 지침을 지정하는 데 사용됩니다.
- *재작성 요청 요소*는 최적화된 명령문이 판별될 때 적용되는 쿼리 변환에 영향을 주는 데 사용할 수 있는 쿼리 재작성 최적화 지침을 지정하는 데 사용됩니다.
- *액세스 요청 요소* 및 *조인 요청 요소*는 최적화된 명령문에 대한 실행 플랜에서 사용되는 액세스 메소드, 조인 메소드 및 조인 순서에 영향을 주는 데 사용할 수 있는 플랜 최적화 지침입니다.
- *MQT 강제 요청 요소*는 액세스 플랜에서의 사용을 비용 추정에 관계없이 강제해야 하는 구체화된 쿼리 테이블(MQT)을 시맨틱으로 지정합니다.

주: 명령문 프로파일에서 지정된 최적화 지침은 최적화 프로파일의 전역 섹션에서 지정된 최적화 지침보다 우선합니다.

일반 최적화 지침에 대한 XML 스키마:

generalRequest 그룹은 최적화 프로세스의 특정 단계에 국한되지 않은 지침을 정의하고, 옵티마이저의 검색 스페이스를 변경하는 데 사용될 수 있습니다.

```
<!--****************************************************************************** --> ₩
<!-- Choices of general request elements.                                         --> ₩
<!-- REOPT can be used to override the setting of the REOPT bind option.          --> ₩
<!-- DPFXMLMOVEMENT can be used to affect the optimizer's plan when moving XML documents  --> ₩
<!-- between database partitions. The allowed value can be REFERENCE or COMBINATION. The  --> ₩
<!-- default value is NONE.                                                       --> ₩
<!--****************************************************************************** --> ₩
<xs:group name="generalRequest">
   <xs:sequence>
      <xs:element name="REOPT" type="reoptType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="DEGREE" type="degreeType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="QRYOPT" type="qryoptType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="RTS" type="rtsType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="DPFXMLMOVEMENT" type="dpfXMLMovementType" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
</xs:group>
```

설명

일반 요청 요소는 최적화 검색 스페이스에 영향을 주는 일반 최적화 지침을 정의하는 데 사용될 수 있고, 따라서 재작성 및 비용 기반 최적화 지침의 적용 가능성에 영향을 줄 수 있습니다.

DEGREE 요청:

DEGREE 일반 요청 요소를 사용하여 DEGREE 바인드 옵션 설정, **dft_degree** 데이터베이스 구성 매개변수 값 또는 이전 SET CURRENT DEGREE문의 결과를 겹쳐쓸 수 있습니다.

DEGREE 일반 요청 요소는 인스턴스가 파티션 내 병렬 처리를 위해 구성된 경우에만 고려됩니다. 그렇지 않은 경우, 경고가 리턴됩니다. 이것은 복합 유형 degreeType에 의해 정의됩니다.

XML Schema

```
<xs:simpleType name="intStringType">
   <xs:union>
      <xs:simpleType>
         <xs:restriction base="xs:integer">
            <xs:minInclusive value="1"></xs:minInclusive>
            <xs:maxInclusive value="32767"></xs:maxInclusive>
         </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ANY"/>
            <xs:enumeration value="-1"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:union>
</xs:simpleType>

<xs:complexType name="degreeType">
   <xs:attribute name="VALUE"
      type="intStringType"></xs:attribute>
</xs:complexType>
```

설명

DEGREE 일반 요청 요소에는 DEGREE 옵션의 설정을 지정하는 필수 VALUE 속성이 있습니다. 이 속성은 1에서 32 767의 정수 값 또는 문자열 값 -1 또는 ANY를 취할 수 있습니다. 값 -1(또는 ANY)은 병렬 처리의 등급이 데이터 서버에 의해 결정되도록 지정합니다. 값 1은 쿼리에서 파티션 내 병렬 처리를 사용하지 않도록 지정합니다.

DPFXMLMOVEMENT 요청:

DPFXMLMOVEMENT 일반 요청 요소를 파티션된 데이터베이스 환경에서 사용하여 XML 유형의 컬럼을 이동하거나 해당 컬럼에 대한 참조만 데이터베이스 파티션 간에 이동하는 플랜을 선택하는 옵티마이저의 결정을 겹쳐쓸 수 있습니다. 이는 복합 유형 dpfXMLMovementType에 의해 정의됩니다.

```
<xs:complexType name="dpfXMLMovementType">
   <xs:attribute name="VALUE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string"
            <xs:enumeration value="REFERENCE"/>
```

```
            <xs:enumeration value="COMBINATION"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
```

설명

파티션된 데이터베이스 환경에서는 명령문 실행 중에 데이터베이스 파티션 간에 데이터를 이동해야 하는 경우가 있습니다. XML 컬럼의 경우, 옵티마이저는 해당 컬럼에 포함된 실제 문서 또는 원래 데이터베이스 파티션의 소스 문서에 대한 참조만 이동하도록 선택할 수 있습니다.

DPFXMLMOVEMENT 일반 요청 요소에는 REFERENCE 또는 COMBINATION 값을 갖는 VALUE 속성이 있습니다. XML 컬럼이 포함된 행을 한 데이터베이스 파티션에서 다른 데이터베이스 파티션으로 이동해야 하는 경우, 다음을 수행합니다.

- REFERENCE는 XML 문서에 대한 참조가 테이블 큐(TQ) 운영자를 통해 이동되도록 지정합니다. 문서 자체는 소스 데이터베이스 파티션에 남아 있습니다.
- COMBINATION은 일부 XML 문서가 이동되도록 지정하고, 나머지 XML 문서에 대한 참조만 TQ 운영자를 통해 이동되도록 지정합니다.

문서 또는 해당 문서에 대한 참조만 이동되는지 여부에 대한 결정은 쿼리가 실행될 때 우세한 조건에 따라 다릅니다. DPFXMLMOVEMENT 일반 요청 요소가 지정되지 않은 경우, 옵티마이저는 성능을 최대화하기 위해 비용 기반 결정을 내립니다.

QRYOPT 요청:

QRYOPT 일반 요청을 사용하여 이전 SET CURRENT QUERYOPT OPTIMIZATION 명령문의 결과, **dft_queryopt** 데이터베이스 구성 매개변수의 값 또는 QUERYOPT 바인드 옵션의 설정을 겹쳐쓸 수 있습니다. 이것은 복합 유형 qryoptType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="qryoptType">
   <xs:attribute name="VALUE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="0"/>
            <xs:enumeration value="1"/>
            <xs:enumeration value="2"/>
            <xs:enumeration value="3"/>
            <xs:enumeration value="5"/>
            <xs:enumeration value="7"/>
            <xs:enumeration value="9"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
```

설명

QRYOPT 일반 요청 요소에는 QUERYOPT 옵션의 설정을 지정하는 필수 VALUE 속성이 있습니다. 이 속성은 0, 1, 2, 3, 5, 7 또는 9 값 중 하나를 취할 수 있습니다. 이러한 값이 나타내는 내용에 대한 자세한 정보는 『최적화 클래스』를 참조하십시오.

REOPT 요청:

REOPT 일반 요청을 사용하여 매개변수 표시문자 또는 호스트 변수를 포함하는 명령문의 최적화에 영향을 주는 REOPT 바인드 옵션의 설정을 겹쳐쓸 수 있습니다. 이것은 복합 유형 reoptType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="reoptType">
   <xs:attribute name="VALUE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ONCE"/>
            <xs:enumeration value="ALWAYS"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
```

설명

REOPT 일반 요청 요소에는 REOPT 옵션의 설정을 지정하는 필수 VALUE 속성이 있습니다. 이 속성은 ONCE 또는 ALWAYS 값을 취할 수 있습니다. ONCE는 첫 번째 세트의 호스트 변수 또는 매개변수 표시문자 값에 대해 명령문을 최적화해야 하도록 지정합니다. ALWAYS는 각 세트의 호스트 변수 또는 매개변수 표시문자 값에 대해 명령문을 최적화해야 하도록 지정합니다.

RTS 요청:

RTS 일반 요청 요소를 사용하여 실시간 통계 콜렉션을 사용하거나 사용하지 않도록 할 수 있습니다. 또한 실시간 통계 콜렉션에 걸리는 시간 양을 제한하는 데 사용할 수도 있습니다.

특정 쿼리 또는 워크로드의 경우, 명령문 컴파일 시간의 추가 오버헤드를 피할 수 있도록 실시간 통계 콜렉션을 제한하는 것이 좋을 수도 있습니다. RTS 일반 요청 요소는 복합 유형 rtsType에 의해 정의됩니다.

```
<!--****************************************************************************************--> ₩
<!-- RTS general request element to enable, disable or provide a time budget for             --> ₩
<!-- real-time statistics collection.                                                        --> ₩
<!-- OPTION attribute allows enabling or disabling real-time statistics.                     --> ₩
<!-- TIME attribute provides a time budget in milliseconds for real-time statistics collection.--> ₩
<!--**************************************************************************************** --> ₩
<xs:complexType name="rtsType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="TIME" type="xs:nonNegativeInteger" use="optional"/>
</xs:complexType>
```

설명

RTS 일반 요청 요소에는 두 가지 선택적 속성이 있습니다.

- OPTION 속성은 실시간 통계 콜렉션을 사용하거나 사용하지 않도록 하는 데 사용됩니다. 이 속성은 ENABLE(디폴트) 또는 DISABLE 값을 취할 수 있습니다.
- TIME 속성은 명령문 컴파일 시간에 실시간 통계 콜렉션에 명령문당 소요되는 최대 시간(밀리초)을 지정합니다.

OPTION 속성에 대해 ENABLE이 지정된 경우, 자동 통계 콜렉션 및 실시간 통계가 해당 구성 매개변수를 통해 사용되어야 합니다. 그렇지 않으면, 최적화 지침이 적용되지 않고, 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

쿼리 재작성 최적화 지침에 대한 XML 스키마:

rewriteRequest 그룹은 최적화 프로세스의 쿼리 재작성 단계에 영향을 주는 지침을 정의합니다.

```
XML Schema

   <xs:group name="rewriteRequest">
      <xs:sequence>
         <xs:element name="INLIST2JOIN" type="inListToJoinType" minOccurs="0"/>
         <xs:element name="SUBQ2JOIN"   type="subqueryToJoinType" minOccurs="0"/>
         <xs:element name="NOTEX2AJ"    type="notExistsToAntiJoinType" minOccurs="0"/>
         <xs:element name="NOTIN2AJ"    type="notInToAntiJoinType" minOccurs="0"/>
      </xs:sequence>
   </xs:group>
```

설명

명령문 레벨과 술어 레벨 최적화 지침 둘 다를 지정하는 데 INLIST2JOIN 요소가 사용된 경우, 술어 레벨 지침이 명령문 레벨 지침을 겹쳐씁니다.

IN-LIST-to-join 쿼리 재작성 요청:

INLIST2JOIN 쿼리 재작성 요청 요소는 IN-LIST predicate-to-join 재작성 변환을 사용하거나 사용하지 않도록 하는 데 사용될 수 있습니다. 이것은 명령문 레벨 최적화 지침 또는 술어 레벨 최적화 지침으로 지정될 수 있습니다. 후자의 경우, 쿼리당 하나의 지침만 사용할 수 있습니다. INLIST2JOIN 요청 요소는 복합 유형 inListToJoinType에 의해 정의됩니다.

```
XML Schema

   <xs:complexType name="inListToJoinType">
      <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
      <xs:attribute name="TABLE"  type="xs:string"  use="optional"/>
      <xs:attribute name="COLUMN" type="xs:string"  use="optional"/>
   </xs:complexType>
```

설명

INLIST2JOIN 쿼리 재작성 요청 요소에는 세 가지 선택적 속성이 있으며 하위 요소는 없습니다. OPTION 속성은 값 ENABLE(디폴트) 또는 DISABLE을 취할 수 있습니

다. TABLE 및 COLUMN 속성은 IN-LIST 술어를 지정하는 데 사용됩니다. 이러한 속성이 지정되지 않거나 빈 문자열 (『』) 값으로 지정된 경우 지침이 명령문 레벨 지침으로 처리됩니다. 이러한 속성 중 하나 또는 둘 다가 지정된 경우에는 술어 레벨 지침으로 처리됩니다. TABLE 속성이 지정되지 않거나 빈 문자열 값으로 지정되었지만 COLUMN 속성이 지정된 경우, 최적화 지침이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

NOT-EXISTS-to-anti-join 쿼리 재작성 요청:

NOTEX2AJ 쿼리 재작성 요청 요소는 NOT-EXISTS 재작성 변환을 사용하거나 사용하지 않도록 하는 데 사용될 수 있습니다. 이것은 명령문 최적화 지침으로만 지정될 수 있습니다. NOTEX2AJ 요청 요소는 복합 유형 notExistsToAntiJoinType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="notExistsToAntiJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
```

설명

NOTEX2AJ 쿼리 재작성 요청 요소에는 한 가지 선택적 속성이 있으며 하위 요소는 없습니다. OPTION 속성은 값 ENABLE(디폴트) 또는 DISABLE을 취할 수 있습니다.

NOT-IN-to-anti-join 쿼리 재작성 요청:

NOTIN2AJ 쿼리 재작성 요청 요소는 NOT-IN predicate-to-anti-join 재작성 변환을 사용하거나 사용하지 않도록 하는 데 사용될 수 있습니다. 이것은 명령문 최적화 지침으로만 지정될 수 있습니다. NOTIN2AJ 요청 요소는 복합 유형 notInToAntiJoinType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="notInToAntiJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
```

설명

NOTIN2AJ 쿼리 재작성 요청 요소에는 한 가지 선택적 속성이 있으며 하위 요소는 없습니다. OPTION 속성은 값 ENABLE(디폴트) 또는 DISABLE을 취할 수 있습니다.

Subquery-to-join 쿼리 재작성 요청:

SUBQ2JOIN 쿼리 재작성 요청 요소는 subquery-to-join 재작성 변환을 사용하거나 사용하지 않도록 하는 데 사용될 수 있습니다. 이것은 명령문 최적화 지침으로만 지정될 수 있습니다. SUBQ2JOIN 요청 요소는 복합 유형 subqueryToJoinType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="subqueryToJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
```

설명

SUBQ2JOIN 쿼리 재작성 요청 요소에는 한 가지 선택적 속성이 있으며 하위 요소는 없습니다. OPTION 속성은 값 ENABLE(디폴트) 또는 DISABLE을 취할 수 있습니다.

플랜 최적화 지침에 대한 XML 스키마:

플랜 최적화 지침은 액세스 요청 또는 조인 요청으로 구성될 수 있습니다.

- *액세스 요청*은 테이블 참조에 대한 액세스 메소드를 지정합니다.
- *조인 요청*은 조인 조작을 수행하기 위한 메소드 및 시퀀스를 지정합니다. 조인 요청은 기타 조인 또는 액세스 요청으로 구성됩니다.

대부분의 사용 가능한 액세스 요청은 테이블 스캔, 인덱스 스캔 및 리스트 프리페치와 같은 옵티마이저의 데이터 액세스 메소드에 해당합니다. 대부분의 사용 가능한 조인 요청은 중첩된 루프 조인, 해시 조인 및 병합 조인과 같은 옵티마이저의 조인 메소드에 해당합니다. 각 액세스 요청 또는 조인 요청 요소를 사용하여 플랜 최적화에 영향을 줄 수 있습니다.

액세스 요청:

accessRequest 그룹은 유효한 액세스 요청 요소 세트를 정의합니다. 액세스 요청은 테이블 참조에 대한 액세스 메소드를 지정합니다.

XML Schema

```
<xs:group name="accessRequest">
   <xs:choice>
      <xs:element name="TBSCAN" type="tableScanType"/>
      <xs:element name="IXSCAN" type="indexScanType"/>
      <xs:element name="LPREFETCH" type="listPrefetchType"/>
      <xs:element name="IXAND" type="indexAndingType"/>
      <xs:element name="IXOR" type="indexOringType"/>
      <xs:element name="XISCAN" type="indexScanType"/>
      <xs:element name="XANDOR" type="XANDORType"/>
      <xs:element name="ACCESS" type="anyAccessType"/>
   </xs:choice>
</xs:group>
```

설명

- TBSCAN, IXSCAN, LPREFETCH, IXAND, IXOR, XISCAN 및 XANDOR

이러한 요소는 DB2 데이터 액세스 메소드에 해당하고, 명령문에서 참조되는 로컬 테이블에만 적용될 수 있습니다. 별칭(리모트 테이블)이나 파생된 테이블(하위 선택의 결과)은 참조할 수 없습니다.

- ACCESS

  옵티마이저가 액세스 메소드를 선택하도록 하는 이 요소는 액세스 메소드가 아닌 조인 순서가 최대 관심사인 경우 사용될 수 있습니다. 목표 테이블 참조가 파생된 테이블인 경우 ACCESS 요소를 사용해야 합니다. XML 쿼리의 경우, 이 요소를 TYPE = XMLINDEX 속성과 함께 사용하여 옵티마이저가 XML 인덱스 액세스 플랜을 선택하도록 지정할 수도 있습니다.

액세스 유형:

TBSCAN, IXSCAN, LPREFETCH, IXAND, IXOR, XISCAN, XANDOR 및 ACCESS 요소의 일반 측면은 추상 유형 accessType에 의해 정의됩니다.

XML Schema

```
   <xs:complexType name="accessType" abstract="true">
      <xs:attribute name="TABLE" type="xs:string" use="optional"/>
      <xs:attribute name="TABID" type="xs:string" use="optional"/>
      <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
      <xs:attribute name="SHARING" type="optionType" use="optional" default="ENABLE"/>
<xs:attribute name="WRAPPING" type="optionType" use="optional" default="ENABLE"/>
<xs:attribute name="THROTTLE" type="optionType" use="optional"/>
      <xs:attribute name="SHARESPEED" type="shareSpeed" use="optional"/>
   </xs:complexType>

   <xs:complexType name="extendedAccessType">
      <xs:complexContent>
         <xs:extension base="accessType">
            <xs:sequence minOccurs="0">
               <xs:element name="INDEX" type="indexType" minOccurs="2"
                 maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="INDEX" type="xs:string" use="optional"/>
            <xs:attribute name="TYPE" type="xs:string" use="optional" fixed="XMLINDEX"/>
<xs:attribute name="ALLINDEXES" type="boolType" use="optional" fixed="TRUE"/>
</xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

설명

모든 액세스 요청 요소는 복합 유형 accessType을 확장합니다. 이러한 각 요소는 TABLE 또는 TABID 속성을 사용하여 목표 테이블 참조를 지정해야 합니다. 액세스 요청 요소에서 적절한 테이블 참조를 구성하는 방법에 대한 정보는 『최적화 지침의 테이블 참조 구성』을 참조하십시오.

액세스 요청 요소는 선택적 FIRST 속성도 지정할 수 있습니다. FIRST 속성이 지정된 경우, 참 값을 포함해야 합니다. 액세스 요청 요소에 FIRST 속성을 추가하는 것은 실행 플랜에서 해당 FROM 절의 조인 시퀀스에서 지정된 테이블을 첫 번째 테이블로 포함해야 한다는 것을 표시합니다. FROM절당 하나의 액세스 또는 조인 요청만이 FIRST 속성을 지정할 수 있습니다. 동일한 FROM절의 테이블을 목표로 하는 여러 액세스 또

는 조인 요청이 FIRST 속성을 지정할 경우, 최초 요청을 제외한 모든 요청이 무시되고 경고(이유 코드 13과 함께 SQL0437W)가 리턴됩니다.

새 옵티마이저 지침을 사용하여 컴파일러의 스캔 공유 결정에 영향을 줄 수 있습니다. 컴파일러에서 공유 스캔, 랩핑 스캔 또는 조절을 허용한 경우, 적절한 지침을 지정하면 공유 스캔, 랩핑 스캔 또는 조절을 예방할 수 있습니다. 공유 스캔은 스캔 공유에 참여 중인 다른 스캔에서 볼 수 있고 해당 스캔은 해당 정보를 기반으로 특정 결정을 할 수 있습니다. 랩핑 스캔은 테이블의 임의 지점에서 시작되어 이미 버퍼 풀에 있는 페이지를 이용할 수 있습니다. 조절된 스캔은 전체 공유 레벨을 증가시키기 위해 지연됩니다.

유효한 optionType 값(SHARING, WRAPPING 및 THROTTLE 속성의 경우)은 DISABLE과 ENABLE(디폴트)입니다. 컴파일러가 사용할 수 없도록 선택하는 경우에는 SHARING과 WRAPPING을 사용할 수 없습니다. 이러한 경우에는 ENABLE 사용이 효과가 없습니다. THROTTLE은 사용 가능하거나 사용 불가능할 수 있습니다. 유효한 SHARESPEED 값(컴파일러의 스캔 속도 추정을 겹쳐쓰기 위한)은 FAST와 SLOW입니다. 디폴트는 컴파일러에서 추정을 기반으로 값을 판별하도록 허용하는 것입니다.

TYPE 속성에 지원되는 유일한 값은 XMLINDEX입니다(옵티마이저가 IXAND, IXOR, XANDOR 또는 XISCAN과 같은 XML 인덱스 액세스 메소드 중 하나를 사용하여 테이블에 액세스해야 함을 표시함). 이 속성이 지정되지 않은 경우, 옵티마이저는 지정된 테이블에 대한 액세스 플랜을 선택할 때 비용 기반 결정을 내립니다.

선택적 INDEX 속성을 사용하여 인덱스 이름을 지정할 수 있습니다.

선택적 INDEX 요소를 사용하여 두 개 이상의 인덱스 이름을 인덱스 요소로 지정할 수 있습니다. INDEX 속성과 INDEX 요소가 둘 다 지정된 경우, INDEX 속성은 무시됩니다.

TYPE 속성 값이 XMLINDEX인 경우에만 선택적 ALLINDEXES 속성(유일하게 지원되는 값은 TRUE임)을 지정할 수 있습니다. ALLINDEXES 속성이 지정된 경우, 옵티마이저는 비용에 관계없이 적용 가능한 모든 관계형 인덱스 및 XML 데이터에 대한 인덱스를 사용하여 지정된 테이블에 액세스해야 합니다.

임의 액세스 요청:

ACCESS 액세스 요청 요소는 옵티마이저가 비용을 기반으로 테이블에 액세스하기 위한 적절한 메소드를 선택하도록 지정하는 데 사용될 수 있으며, 파생된 테이블을 참조할 때 사용되어야 합니다. 파생된 테이블은 다른 하위 선택의 결과입니다. 이 액세스 요청 요소는 복합 유형 anyAccessType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="anyAccessType">
```

```
    <xs:complexContent>
       <xs:extension base="extendedAccessType"/>
    </xs:complexContent>
 </xs:complexType>
```

설명

복합 유형 anyAccessType은 추상 유형 extendedAccessType의 단순 확장입니다. 새 요소 또는 속성이 추가되지 않습니다.

TYPE 속성(유일하게 지원되는 값은 XMLINDEX임)은 옵티마이저가 IXAND, IXOR, XANDOR 또는 XISCAN과 같은 XML 인덱스 액세스 메소드 중 하나를 사용하여 테이블에 액세스해야 함을 표시합니다. 이 속성이 지정되지 않은 경우, 옵티마이저는 지정된 테이블에 대한 액세스 플랜을 선택할 때 비용 기반 결정을 내립니다.

TYPE 속성 값이 XMLINDEX인 경우에만 선택적 INDEX 속성을 사용하여 인덱스 이름을 지정할 수 있습니다. 이 속성이 지정되지 않은 경우, 옵티마이저는 다음 플랜 중 하나를 선택할 수 있습니다.

- XML 데이터에 대한 지정된 인덱스를 사용하는 XISCAN 플랜
- XML 데이터에 대한 지정된 인덱스가 XANDOR의 인덱스 중 하나인 XANDOR 플랜(옵티마이저는 XANDOR 플랜의 XML 데이터에 대한 적용 가능한 모든 인덱스를 사용함)
- 지정된 인덱스가 IXAND의 선행 인덱스인 IXAND 플랜(옵티마이저는 비용 기반 방식으로 IXAND 플랜에 인덱스를 더 추가함)
- 비용 기반 IXOR 플랜

TYPE 속성 값이 XMLINDEX인 경우에만 선택적 INDEX 요소를 사용하여 두 개 이상의 인덱스 이름을 인덱스 요소로 지정할 수 있습니다. 이 요소가 지정된 경우, 옵티마이저는 다음 플랜 중 하나를 선택할 수 있습니다.

- XML 데이터에 대한 지정된 인덱스가 XANDOR에 나타나는 XANDOR 플랜(옵티마이저는 XANDOR 플랜의 XML 데이터에 대한 적용 가능한 모든 인덱스를 사용함)
- 지정된 인덱스가 지정된 순서로 IXAND의 인덱스인 IXAND 플랜
- 비용 기반 IXOR 플랜

INDEX 속성과 INDEX 요소가 둘 다 지정된 경우, INDEX 속성은 무시됩니다.

TYPE 속성 값이 XMLINDEX인 경우에만 선택적 ALLINDEXES 속성(유일하게 지원되는 값은 TRUE임)을 지정할 수 있습니다. 이 속성이 지정된 경우, 옵티마이저는 비용에 관계없이 적용 가능한 모든 관계형 인덱스 및 XML 데이터에 대한 인덱스를 사용하여 지정된 테이블에 액세스해야 합니다. 옵티마이저는 다음 플랜 중 하나를 선택합니다.

- XML 데이터에 대한 적용 가능한 모든 인덱스가 XANDOR 연산자에 나타나는 XANDOR 플랜
- 적용 가능한 모든 관계형 인덱스 및 XML 데이터에 대한 인덱스가 IXAND 연산자에 나타나는 IXAND 플랜
- IXOR 플랜
- 단일 인덱스만 테이블에 정의되고 해당 인덱스 유형이 XML인 경우 XISCAN 플랜

예

다음 지침은 액세스 요청 예입니다.

```
<OPTGUIDELINES>
  <HSJOIN>
    <ACCESS TABLE='S1'/>
    <IXSCAN TABLE='PS1'/>
  </HSJOIN>
</OPTGUIDELINES>
```

다음 예는 SECURITY 테이블에 대한 일부 XML 인덱스 액세스를 사용해야 함을 지정하는 ACCESS 지침을 표시합니다. 옵티마이저는 임의의 XML 인덱스 플랜 (예: XISCAN, IXAND, XANDOR 또는 IXOR 플랜)을 선택할 수 있습니다.

```
SELECT * FROM security
  WHERE XMLEXISTS('$SDOC/Security/SecurityInformation/
    StockInformation[Industry= "OfficeSupplies"]')

<OPTGUIDELINES>
  <ACCESS TABLE='SECURITY' TYPE='XMLINDEX'/>
</OPTGUIDELINES>
```

다음 예는 SECURITY 테이블에 대한 가능한 모든 인덱스 액세스를 사용해야 함을 지정하는 ACCESS 지침을 표시합니다. 메소드 중 하나는 옵티마이저에게 맡기는 것입니다. 두 개의 XML 인덱스(SEC_INDUSTRY 및 SEC_SYMBOL)가 두 개의 XML 술어와 일치한다고 가정하십시오. 옵티마이저는 비용 기반 방식을 사용하여 XANDOR 또는 IXAND 액세스 메소드를 선택합니다.

```
SELECT * FROM security
  WHERE XMLEXISTS('$SDOC/Security/SecurityInformation/
      StockInformation[Industry= "Software"]') AND
    XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')

<OPTGUIDELINES>
  <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' ALLINDEXES='TRUE'/>
</OPTGUIDELINES>
```

다음 예는 최소한 SEC_INDUSTRY XML 인덱스를 사용하여 SECURITY 테이블에 액세스해야 함을 ACCESS 지침을 표시합니다. 옵티마이저는 비용 기반 방식으로 다음 액세스 플랜 중 하나를 선택합니다.

- SEC_INDUSTRY XML 인덱스를 사용하는 XISCAN 플랜

- IXAND의 첫 번째 레그로 SEC_INDUSTRY 인덱스가 있는 IXAND 플랜. 옵티마이저는 비용 기반 분석을 따라 IXAND 플랜의 관계형 또는 XML 인덱스를 추가로 얼마든지 사용할 수 있습니다. 예를 들어, TRANS_DATE 컬럼에서 관계형 인덱스를 사용할 수 있는 경우 옵티마이저가 유용하다고 간주하는 경우에는 해당 인덱스가 IXAND의 추가적인 레그로 나타날 수 있습니다.
- SEC_INDUSTRY 인덱스 및 기타 적용 가능한 XML 인덱스를 사용하는 XANDOR 플랜

```
SELECT * FROM security
  WHERE trans_date = CURRENT DATE AND
    XMLEXISTS('$SDOC/Security/SecurityInformation/
      StockInformation[Industry= "Software"]') AND
    XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')

<OPTGUIDELINES>
  <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' INDEX='SEC_INDUSTRY'/>
</OPTGUIDELINES>
```

인덱스 ANDing 액세스 요청:

IXAND 액세스 요청 요소를 사용하여 옵티마이저가 인덱스 ANDing 데이터 액세스 메소드를 사용해 로컬 테이블에 액세스하도록 지정할 수 있습니다. 이것은 복합 유형 indexAndingType에 의해 정의됩니다.

```
XML Schema

   <xs:complexType name="indexAndingType">
      <xs:complexContent>
         <xs:extension base="extendedAccessType">
      </xs:complexContent>
   </xs:complexType>
```

설명

복합 유형 indexAndingType은 extendedAccessType의 단순 확장입니다.
extendedAccessType 유형은 선택적 INDEX 속성, 선택적 INDEX 하위 요소, 선택적 TYPE 속성 및 선택적 ALLINDEXES 속성을 추가하여 추상 유형 accessType을 확장합니다. INDEX 속성을 사용하여 인덱스 ANDing 조작에서 사용할 첫 번째 인덱스를 지정할 수 있습니다. INDEX 속성이 사용되는 경우, 옵티마이저가 비용 기반 방식으로 추가 인덱스 및 액세스 시퀀스를 선택합니다. INDEX 하위 요소를 사용하여 인덱스 및 액세스 시퀀스의 정확한 세트를 지정할 수 있습니다. INDEX 하위 요소가 나타나는 순서는 개별 인덱스 스캔이 수행되어야 하는 순서를 표시합니다. INDEX 하위 요소의 스펙은 INDEX 속성의 스펙보다 우선합니다.

- 인덱스가 지정되지 않은 경우, 옵티마이저가 비용 기반 방식으로 인덱스와 액세스 시퀀스를 모두 선택합니다.
- 속성 또는 하위 요소를 사용하여 인덱스가 지정된 경우, TABLE 또는 TABID 속성에 의해 식별되는 테이블에서 이러한 인덱스가 정의되어야 합니다.

- 인덱스가 테이블에 정의되지 않은 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

TYPE 속성(유일하게 지원되는 값은 XMLINDEX임)은 옵티마이저가 XML 데이터에 대한 하나 이상의 인덱스를 사용하여 테이블에 액세스해야 함을 표시합니다.

TYPE 속성 값이 XMLINDEX인 경우에만 선택적 INDEX 속성을 사용하여 XML 인덱스 이름을 지정할 수 있습니다. TYPE 속성 스펙에 관계없이 선택적 INDEX 속성에 관계형 인덱스를 지정할 수 있습니다. 옵티마이저는 지정된 인덱스를 IXAND 플랜의 선행 인덱스로 사용합니다. 옵티마이저는 비용 기반 방식으로 IXAND 플랜에 인덱스를 더 추가합니다.

TYPE 속성 값이 XMLINDEX인 경우에만 선택적 INDEX 요소를 사용하여 XML 데이터에 대한 두 개 이상의 인덱스 이름을 인덱스 요소로 지정할 수 있습니다. TYPE 속성 스펙에 관계없이 선택적 INDEX 요소에 관계형 인덱스를 지정할 수 있습니다. 옵티마이저는 지정된 인덱스를 지정된 순서로 IXAND 플랜의 인덱스로 사용합니다.

TYPE 속성이 없는 경우, INDEX 속성 및 INDEX 요소는 관계형 인덱스에 여전히 유효합니다.

INDEX 속성과 INDEX 요소가 둘 다 지정된 경우, INDEX 속성은 무시됩니다.

TYPE 속성 값이 XMLINDEX인 경우에만 선택적 ALLINDEXES 속성(유일하게 지원되는 값은 TRUE임)을 지정할 수 있습니다. 이 속성이 지정된 경우, 옵티마이저는 비용에 관계없이 IXAND 플랜의 적용 가능한 모든 관계형 인덱스 및 XML 데이터에 대한 인덱스를 사용하여 지정된 테이블에 액세스해야 합니다.

TYPE 속성은 지정되지만 INDEX 속성, INDEX 요소 및 ALLINDEXES 속성도 지정되지 않은 경우, 옵티마이저는 최소 하나의 XML 데이터에 대한 인덱스가 있는 IXAND 플랜을 선택합니다. 플랜의 기타 인덱스는 관계형 인덱스 또는 XML 데이터에 대한 인덱스일 수 있습니다. 옵티마이저는 비용 기반 방식으로 인덱스 순서와 선택을 결정합니다.

인덱스 ANDing 요청에서 블록 인덱스가 레코드 인덱스 앞에 나타나야 합니다. 이 요구사항이 충족되지 않을 경우, 이유 코드 13과 함께 SQL0437W가 리턴됩니다. 인덱스 ANDing 액세스 메소드에서는 최소한 하나의 술어가 각 인덱스에 대해 인덱스화될 수 있어야 합니다. 필요한 술어가 존재하지 않아 인덱스 ANDing이 적합하지 않을 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다. 인덱스 ANDing 데이터 액세스 메소드가 명령문에 대해 적용되는 검색 스페이스에 없는 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

다음 예는 인덱스 ANDing 액세스 요청을 설명합니다.

```
SQL statement:

   select s.s_name, s.s_address, s.s_phone, s.s_comment
     from "Tpcd".parts, "Tpcd".suppliers s, "Tpcd".partsupp ps
     where p_partkey = ps.ps_partkey and
       s.s_suppkey = ps.ps_suppkey and
       p_size = 39 and
       p_type = 'BRASS' and
       s.s_nation in ('MOROCCO', 'SPAIN') and
       ps.ps_supplycost = (select min(ps1.ps_supplycost)
                             from "Tpcd".partsupp ps1, "Tpcd".suppliers s1
                             where "Tpcd".parts.p_partkey = ps1.ps_partkey and
                               s1.s_suppkey = ps1.ps_suppkey and
                               s1.s_nation = s.s_nation)
     order by s.s_name
     optimize for 1 row

Optimization guideline:

   <OPTGUIDELINES>
     <IXAND TABLE='"Tpcd".PARTS' FIRST='TRUE'>
       <INDEX IXNAME='ISIZE'/>
       <INDEX IXNAME='ITYPE'/>
     </IXAND>
   </OPTGUIDELINES>
```

인덱스 ANDing 요청은 기본 subselect의 PARTS 테이블이 인덱스 ANDing 데이터 액세스 메소드를 사용하여 충족되도록 지정합니다. 첫 번째 인덱스 스캔은 ISIZE 인덱스를 사용하고 두 번째 인덱스 스캔은 ITYPE 인덱스를 사용합니다. INDEX 요소의 IXNAME 속성이 인덱스를 지정합니다. FIRST 속성 설정은 PARTS 테이블이 동일한 FROM절에 SUPPLIERS, PARTSUPP 및 파생된 테이블을 가진 조인 시퀀스의 첫 번째 테이블이 되도록 지정합니다.

인덱스 ORing 액세스 요청:

IXOR 액세스 요청 요소를 사용하여 옵티마이저가 인덱스 ORing 데이터 액세스 메소드를 사용해 로컬 테이블에 액세스하도록 지정할 수 있습니다. 이것은 복합 유형 indexOringType에 의해 정의됩니다.

```
XML Schema

   <xs:complexType name="indexOringType">
      <xs:complexContent>
         <xs:extension base="accessType"/>
      </xs:complexContent>
   </xs:complexType>
```

설명

복합 유형 indexOringType은 추상 유형 accessType의 단순 확장입니다. 새 요소 또는 속성이 추가되지 않습니다. 인덱스 ORing 액세스 메소드가 명령문에 대해 적용되는 검색 스페이스에 없는 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다. 옵티마이저는 비용 기반 방식으로 인덱스 ORing 조작에서

사용되는 인덱스 및 술어를 선택합니다. 인덱스 ORing 액세스 메소드에서는 최소한 하나의 IN 술어가 인덱스화 될 수 있어야 하거나 용어가 있는 술어가 인덱스화되어 논리적 OR 조작에 의해 연결될 수 있어야 합니다. 필요한 술어 또는 인덱스가 존재하지 않아 인덱스 ORing이 적합하지 않을 경우, 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

다음 예는 인덱스 ORing 액세스 요청을 설명합니다.

```
SQL statement:

   select s.s_name, s.s_address, s.s_phone, s.s_comment
     from "Tpcd".parts, "Tpcd".suppliers s, "Tpcd".partsupp ps
     where p_partkey = ps.ps_partkey and
       s.s_suppkey = ps.ps_suppkey and
       p_size = 39 and
       p_type = 'BRASS' and
       s.s_nation in ('MOROCCO', 'SPAIN') and
       ps.ps_supplycost = (select min(ps1.ps_supplycost)
                             from "Tpcd".partsupp ps1, "Tpcd".suppliers s1
                             where "Tpcd".parts.p_partkey = ps1.ps_partkey and
                               s1.s_suppkey = ps1.ps_suppkey and
                               s1.s_nation = s.s_nation)
     order by s.s_name
     optimize for 1 row

Optimization guideline:

   <OPTGUIDELINES>
     <IXOR TABLE='S'/>
   </OPTGUIDELINES>
```

이 인덱스 ORing 액세스 요청은 기본 subselect에서 참조되는 SUPPLIERS 테이블에 액세스하는 데 인덱스 ORing 데이터 액세스 메소드가 사용되도록 지정합니다. 옵티마이저는 비용 기반 방식으로 인덱스 ORing 연산에 대한 해당 술어 및 인덱스를 선택합니다.

인덱스 스캔 액세스 요청:

IXSCAN 액세스 요청 요소를 사용하여 옵티마이저가 인덱스 스캔을 사용해 로컬 테이블에 액세스하도록 지정할 수 있습니다. 이것은 복합 유형 indexScanType에 의해 정의됩니다.

```
XML Schema

   <xs:complexType name="indexScanType">
      <xs:complexContent>
         <xs:extension base="accessType">
            <xs:attribute name="INDEX" type="xs:string" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

설명

복합 유형 indexScanType은 선택적 INDEX 속성을 추가하여 추상 accessType을 확장합니다. INDEX 속성은 테이블에 액세스하는 데 사용될 인덱스의 이름을 지정합니다.

- 인덱스 스캔 액세스 메소드가 명령문에 대해 적용되는 검색 스페이스에 없는 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.
- INDEX 속성이 지정된 경우, TABLE 또는 TABID 속성에 의해 식별되는 테이블에서 정의된 인덱스를 식별해야 합니다. 인덱스가 존재하지 않는 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.
- INDEX 속성이 지정되지 않은 경우, 옵티마이저가 비용 기반 방식으로 인덱스를 선택합니다. 인덱스가 목표 테이블에 정의되지 않은 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

다음 지침은 인덱스 스캔 액세스 요청 예입니다.

```
<OPTGUIDELINES>
  <IXSCAN TABLE='S' INDEX='I_SUPPKEY'/>
</OPTGUIDELINES>
```

리스트 프리페치 액세스 요청:

LPREFETCH 액세스 요청 요소를 사용하여 옵티마이저가 리스트 프리페치 인덱스 스캔을 사용해 로컬 테이블에 액세스하도록 지정할 수 있습니다. 이것은 복합 유형 listPrefetchType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="listPrefetchType">
   <xs:complexContent>
      <xs:extension base="accessType">
         <xs:attribute name="INDEX" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

설명

복합 유형 listPrefetchType은 선택적 INDEX 속성을 추가하여 추상 유형 accessType을 확장합니다. INDEX 속성은 테이블에 액세스하는 데 사용될 인덱스의 이름을 지정합니다.

- 리스트 프리페치 액세스 메소드가 명령문에 대해 적용되는 검색 스페이스에 없는 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.
- 리스트 프리페치 액세스 메소드에서는 최소한 하나의 술어가 인덱스화될 수 있어야 합니다. 필요한 술어가 존재하지 않아 리스트 프리페치 메소드가 적합하지 않을 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

- INDEX 속성이 지정된 경우, TABLE 또는 TABID 속성에 의해 지정되는 테이블에서 정의된 인덱스를 식별해야 합니다. 인덱스가 존재하지 않는 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.
- INDEX 속성이 지정되지 않은 경우, 옵티마이저가 비용 기반 방식으로 인덱스를 선택합니다. 인덱스가 목표 테이블에 정의되지 않은 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

다음 지침은 리스트 프리페치 액세스 요청 예입니다.

```
<OPTGUIDELINES>
  <LPREFETCH TABLE='S1' INDEX='I_SNATION'/>
</OPTGUIDELINES>
```

테이블 스캔 액세스 요청:

TBSCAN 액세스 요청 요소를 사용하여 옵티마이저가 순차 테이블 스캔을 사용해 로컬 테이블에 액세스하도록 지정할 수 있습니다. 이것은 복합 유형 tableScanType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="tableScanType">
   <xs:complexContent>
      <xs:extension base="accessType"/>
   </xs:complexContent>
</xs:complexType>
```

설명

복합 유형 tableScanType은 추상 유형 accessType의 단순 확장입니다. 새 요소 또는 속성이 추가되지 않습니다. 테이블 스캔 액세스 메소드가 명령문에 대해 적용되는 검색 스페이스에 없는 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

다음 지침은 테이블 스캔 액세스 요청 예입니다.

```
<OPTGUIDELINES>
  <TBSCAN TABLE='S1'/>
  <LPREFETCH TABLE='S' INDEX='I_SUPPKEY'/>
</OPTGUIDELINES>
```

XML 인덱스 ANDing 및 ORing 액세스 요청:

XANDOR 액세스 요청 요소를 사용하여 옵티마이저가 XML 데이터에 대한 XANDORed 인덱스 스캔을 여러 개 사용하여 로컬 테이블에 액세스하도록 지정할 수 있습니다. 이는 복합 유형 XANDORType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="XANDORType">
```

```
      <xs:complexContent>
         <xs:extension base="accessType"/>
      </xs:complexContent>
   </xs:complexType>
```

설명

복합 유형 XANDORType은 추상 유형 accessType의 단순 확장입니다. 새 요소 또는 속성이 추가되지 않습니다.

예

다음 쿼리를 고려하십시오.

```
SELECT * FROM security
  WHERE trans_date = CURRENT DATE AND
    XMLEXISTS('$SDOC/Security/SecurityInformation/
      StockInformation[Industry = "Software"]') AND
    XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
```

다음 XANDOR 지침은 적용 가능한 모든 XML 인덱스에 대해 XANDOR 연산을 사용하여 SECURITY 테이블에 액세스해야 함을 지정합니다. 관계형 인덱스는 XANDOR 연산자와 함께 사용할 수 없으므로 SECURITY 테이블의 관계형 인덱스는 고려하지 않습니다.

```
<OPTGUIDELINES>
  <XANDOR TABLE='SECURITY'/>
</OPTGUIDELINES>
```

XML 인덱스 스캔 액세스 요청:

XISCAN 액세스 요청 요소를 사용하여 옵티마이저가 XML 데이터에 대한 인덱스 스캔을 사용하여 로컬 테이블에 액세스하도록 지정할 수 있습니다. 이것은 복합 유형 indexScanType에 의해 정의됩니다.

XML Schema

```
   <xs:complexType name="indexScanType">
      <xs:complexContent>
         <xs:extension base="accessType"/>
            <xs:attribute name="INDEX" type="xs:string" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

설명

복합 유형 indexScanType은 선택적 INDEX 속성을 추가하여 추상 accessType을 확장합니다. INDEX 속성은 테이블에 액세스하는 데 사용될 XML 데이터에 대한 인덱스의 이름을 지정합니다.

- XML 데이터에 대한 인덱스 스캔 액세스 메소드가 명령문에 적용되는 검색 스페이스에 없는 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.
- INDEX 속성이 지정된 경우, TABLE 또는 TABID 속성에 의해 식별된 테이블에 정의된 XML 데이터에 대한 인덱스를 식별해야 합니다. 인덱스가 존재하지 않는 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.
- INDEX 속성이 지정되지 않은 경우, 옵티마이저는 비용 기반 방식으로 XML 데이터에 대한 인덱스를 선택합니다. XML 데이터에 대한 인덱스가 목표 테이블에 정의되지 않은 경우, 액세스 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

예

다음 쿼리를 고려하십시오.

```
SELECT * FROM security
  WHERE XMLEXISTS('$SDOC/Security/SecurityInformation/
    StockInformation[Industry = "OfficeSupplies"]')
```

다음 XISCAN 지침은 SEC_INDUSTRY라는 XML 인덱스를 사용하여 SECURITY 테이블에 액세스해야 함을 지정합니다.

```
<OPTGUIDELINES>
  <XISCAN TABLE='SECURITY' INDEX='SEC_INDUSTRY'/>
</OPTGUIDELINES>
```

조인 요청:

joinRequest 그룹은 유효한 조인 요청 요소 세트를 정의합니다. 조인 요청은 두 개의 테이블을 조인하기 위한 메소드를 지정합니다.

XML Schema

```
<xs:group name="joinRequest">
   <xs:choice>
      <xs:element name="NLJOIN" type="nestedLoopJoinType"/>
      <xs:element name="HSJOIN" type="hashJoinType"/>
      <xs:element name="MSJOIN" type="mergeJoinType"/>
      <xs:element name="JOIN" type="anyJoinType"/>
   </xs:choice>
</xs:group>
```

설명

- NLJOIN, MSJOIN 및 HSJOIN

  이러한 요소는 각각 중첩된 루프, 병합 및 해시 조인 메소드에 해당합니다.

- JOIN

옵티마이저가 조인 메소드를 선택하도록 하는 이 요소는 조인 순서가 최대 관심사가 아닌 경우에 사용될 수 있습니다.

모든 조인 요청 요소는 조인 조작의 입력 테이블을 나타내는 두 개의 하위 요소를 포함합니다. 조인 요청은 선택적 FIRST 속성도 지정할 수 있습니다.

다음 지침은 조인 요청 예입니다.

```
<OPTGUIDELINES>
  <HSJOIN>
    <ACCESS TABLE='S1'/>
    <IXSCAN TABLE='PS1'/>
  </HSJOIN>
</OPTGUIDELINES>
```

중첩 순서는 궁극적으로 조인 순서를 판별합니다. 다음 예는 소형 조인 요청에서 대형 조인 요청을 구성하는 방법을 설명합니다.

```
<OPTGUIDELINES>
  <MSJOIN>
    <NLJOIN>
      <IXSCAN TABLE='"Tpcd".Parts'/>
      <IXSCAN TABLE="PS"/>
    </NLJOIN>
    <IXSCAN TABLE='S'/>
  </MSJOIN>
</OPTGUIDELINES>
```

조인 유형:

모든 조인 요청 요소의 일반 측면은 추상 유형 joinType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="joinType" abstract="true">
   <xs:choice minOccurs="2" maxOccurs="2">
      <xs:group ref="accessRequest"/>
      <xs:group ref="joinRequest"/>
   </xs:choice>
   <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
</xs:complexType>
```

설명

복합 유형 joinType을 확장하는 조인 요청 요소는 정확히 두 개의 하위 요소를 포함해야 합니다. 하위 요소는 accessRequest 그룹에서 선택된 액세스 요청 요소이거나 joinRequest 그룹에서 선택된 조인 요청 요소일 수 있습니다. 조인 요청에 표시되는 첫 번째 하위 요소는 조인 조작의 외부 테이블을 지정하고, 두 번째 요소는 내부 테이블을 지정합니다.

FIRST 속성이 지정된 경우, 참 값을 포함해야 합니다. 조인 요청 요소에 FIRST 속성을 추가하는 것은 실행 플랜에서 조인 요청이 목표로 하는 테이블이 해당 FROM절의 조인 시퀀스에서 가장 외부 테이블이어야 한다는 것을 표시합니다. FROM절당 하나의

액세스 또는 조인 요청만이 FIRST 속성을 지정할 수 있습니다. 동일한 FROM절의 테이블을 목표로 하는 여러 액세스 또는 조인 요청이 FIRST 속성을 지정할 경우, 최초 요청을 제외한 모든 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

임의 조인 요청:

JOIN 조인 요청 요소를 사용하여 옵티마이저가 특정 순서로 두 개의 테이블을 조인하기 위한 적절한 메소드를 선택하도록 지정할 수 있습니다.

테이블은 액세스 요청 하위 요소에 의해 지정된 대로 로컬 또는 파생된 테이블이거나, 조인 요청 하위 요소에 의해 지정된 대로 조인 조작의 결과일 수 있습니다. 파생된 테이블은 다른 하위 선택의 결과입니다. 이 조인 요청 요소는 복합 유형 anyJoinType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="anyJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
```

설명

복합 유형 anyJoinType은 추상 유형 joinType의 단순 확장입니다. 새 요소 또는 속성이 추가되지 않습니다.

다음 예는 JOIN 조인 요청을 사용하여 테이블 세트에 대해 특정 조인 순서를 강제 실행하는 방법을 설명합니다.

SQL statement:

```
select s.s_name, s.s_address, s.s_phone, s.s_comment
   from "Tpcd".parts, "Tpcd".suppliers s, "Tpcd".partsupp ps
   where p_partkey = ps.ps_partkey and
     s.s_suppkey = ps.ps_suppkey and
     p_size = 39 and
     p_type = 'BRASS' and
     s.s_nation in ('MOROCCO', 'SPAIN') and
     ps.ps_supplycost = (select min(ps1.ps_supplycost)
                           from "Tpcd".partsupp ps1, "Tpcd".suppliers s1
                           where "Tpcd".parts.p_partkey = ps1.ps_partkey and
                             s1.s_suppkey = ps1.ps_suppkey and
                             s1.s_nation = s.s_nation)
   order by s.s_name
```

Optimization guideline:

```
<OPTGUIDELINES>
  <JOIN>
    <JOIN>
      <ACCESS TABLE='Tpcd".PARTS'/>
      <ACCESS TABLE='S'/>
```

```
    </JOIN>
    <ACCESS TABLE='PS'>
  </JOIN>
</OPTGUIDELINES>
```

JOIN 조인 요청 요소는 기본 subselect의 PARTS 테이블이 SUPPLIERS 테이블과 함께 조인되고 이 결과가 PARTSUPP 테이블에 조인되도록 지정합니다. 옵티마이저는 이 특정 조인 시퀀스의 조인 메소드를 비용 기반 방식으로 선택합니다.

해시 조인 요청:

HSJOIN 조인 요청 요소를 사용하여 옵티마이저가 해시 조인 메소드를 사용하여 두 개의 테이블을 조인하도록 지정할 수 있습니다.

테이블은 액세스 요청 하위 요소에 의해 지정된 대로 로컬 또는 파생된 테이블이거나, 조인 요청 하위 요소에 의해 지정된 대로 조인 조작의 결과일 수 있습니다. 파생된 테이블은 다른 하위 선택의 결과입니다. 이 조인 요청 요소는 복합 유형 hashJoinType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="hashJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
```

설명

복합 유형 hashJoinType은 추상 유형 joinType의 단순 확장입니다. 새 요소 또는 속성이 추가되지 않습니다. 해시 조인 메소드가 명령문에 대해 적용되는 검색 스페이스에 없는 경우, 조인 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

다음 지침은 해시 조인 요청 예입니다.

```
<OPTGUIDELINES>
  <HSJOIN>
    <ACCESS TABLE='S1'/>
    <IXSCAN TABLE='PS1'/>
  </HSJOIN>
</OPTGUIDELINES>
```

병합 조인 요청:

MSJOIN 조인 요청을 사용하여 옵티마이저가 병합 조인 메소드를 사용해 두 개의 테이블을 조인하도록 지정할 수 있습니다.

테이블은 액세스 요청 하위 요소에 의해 지정된 대로 로컬 또는 파생된 테이블이거나, 조인 요청 하위 요소에 의해 지정된 대로 조인 조작의 결과일 수 있습니다. 파생된 테이블은 다른 하위 선택의 결과입니다. 이 조인 요청 요소는 복합 유형 mergeJoinType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="mergeJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
```

설명

복합 유형 mergeJoinType은 추상 유형 joinType의 단순 확장입니다. 새 요소 또는 속성이 추가되지 않습니다. 병합 조인 메소드가 명령문에 대해 적용되는 검색 스페이스에 없는 경우, 조인 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

다음 지침은 병합 조인 요청 예입니다.

```
<OPTGUIDELINES>
  <MSJOIN>
    <NLJOIN>
      <IXSCAN TABLE='"Tpcd".Parts'/>
      <IXSCAN TABLE="PS"/>
    </NLJOIN>
    <IXSCAN TABLE='S'/>
  </MSJOIN>
</OPTGUIDELINES>
```

중첩된 루프 조인 요청:

NLJOIN 조인 요청 요소를 사용하여 옵티마이저가 중첩된 루프 조인 메소드를 사용하여 두 개의 테이블을 조인하도록 지정할 수 있습니다.

테이블은 액세스 요청 하위 요소에 의해 지정된 대로 로컬 또는 파생된 테이블이거나, 조인 요청 하위 요소에 의해 지정된 대로 조인 조작의 결과일 수 있습니다. 파생된 테이블은 다른 하위 선택의 결과입니다. 이 조인 요청 요소는 복합 유형 nestedLoopJoinType에 의해 정의됩니다.

XML Schema

```
<xs:complexType name="nestedLoopJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
```

설명

복합 유형 nestedLoopJoinType은 추상 유형 joinType의 단순 확장입니다. 새 요소 또는 속성이 추가되지 않습니다. 중첩된 루프 조인 메소드가 명령문에 대해 적용되는 검색 스페이스에 없는 경우, 조인 요청이 무시되고 이유 코드 13과 함께 SQL0437W가 리턴됩니다.

다음 지침은 중첩된 루프 조인 요청 예입니다.

```
<OPTGUIDELINES>
  <NLJOIN>
    <IXSCAN TABLE='"Tpcd".Parts'/>
    <IXSCAN TABLE="PS"/>
  </NLJOIN>
</OPTGUIDELINES>
```

SYSTOOLS.OPT_PROFILE 테이블:

SYSTOOLS.OPT_PROFILE 테이블은 모든 최적화 프로파일을 포함합니다.

이 테이블을 작성하는 데에는 두 가지 방법이 있습니다.

- SYSINSTALLOBJECTS 프로시저 호출:

  ```
  db2 "call sysinstallobjects('opt_profiles', 'c', '', '')"
  ```

- CREATE TABLE문 발행:

  ```
  create table systools.opt_profile (
    schema  varchar(128) not null,
    name    varchar(128) not null,
    profile blob (2m)    not null,
    primary key (schema, name)
  )
  ```

SYSTOOLS.OPT_PROFILE 테이블의 컬럼은 다음과 같이 정의됩니다.

SCHEMA

최적화 프로파일에 대한 스키마 이름을 지정합니다. 이 이름은 최대 30개 영숫자 또는 밑줄 문자를 포함할 수 있지만, 표시된 바와 같이 VARCHAR(128)로 정의하십시오.

NAME

최적화 프로파일에 대한 기본 이름을 지정합니다. 이 이름은 최대 128개 영숫자 또는 밑줄 문자를 포함할 수 있습니다.

PROFILE

최적화 프로파일을 정의하는 XML 문서를 지정합니다.

최적화 프로파일 캐시를 플러시하도록 트리거:

최적화 프로파일 캐시는 SYSTOOLS.OPT_PROFILE 테이블의 항목이 갱신되거나 삭제될 때마다 자동으로 플러시됩니다.

프로파일 캐시의 자동 플러시가 발생하려면 먼저 다음 SQL 프로시저 및 트리거가 작성되어야 합니다.

```
CREATE PROCEDURE SYSTOOLS.OPT_FLUSH_CACHE( IN SCHEMA VARCHAR(128),
                                           IN NAME VARCHAR(128) )
LANGUAGE SQL
MODIFIES SQL DATA
BEGIN ATOMIC
 -- FLUSH stmt (33) + quoted schema (130) + dot (1) + quoted name (130) = 294
 DECLARE FSTMT VARCHAR(294) DEFAULT 'FLUSH OPTIMIZATION PROFILE CACHE '; --

 IF NAME IS NOT NULL THEN
   IF SCHEMA IS NOT NULL THEN
     SET FSTMT = FSTMT || '"' || SCHEMA || '".'; --
   END IF; --

   SET FSTMT = FSTMT || '"' || NAME || '"'; --

   EXECUTE IMMEDIATE FSTMT; --
 END IF; --
END;

CREATE TRIGGER SYSTOOLS.OPT_PROFILE_UTRIG AFTER UPDATE ON SYSTOOLS.OPT_PROFILE
REFERENCING OLD AS O
FOR EACH ROW
  CALL SYSTOOLS.OPT_FLUSH_CACHE( O.SCHEMA, O.NAME );

CREATE TRIGGER SYSTOOLS.OPT_PROFILE_DTRIG AFTER DELETE ON SYSTOOLS.OPT_PROFILE
REFERENCING OLD AS O
FOR EACH ROW
  CALL SYSTOOLS.OPT_FLUSH_CACHE( O.SCHEMA, O.NAME );
```

SYSTOOLS.OPT_PROFILE 테이블 관리:

최적화 프로파일은 고유 스키마 규정 이름과 연관되어 SYSTOOLS.OPT_PROFILE 테이블에 저장되어야 합니다. LOAD, IMPORT 및 EXPORT 명령을 사용하여 해당 테이블의 파일을 관리할 수 있습니다.

예를 들어, DB2 클라이언트에서 IMPORT 명령을 사용하여 SYSTOOLS.OPT_PROFILE 테이블에 데이터를 삽입하거나 갱신할 수 있습니다. EXPORT 명령을 사용하여 SYSTOOLS.OPT_PROFILE 테이블의 프로파일을 파일에 복사할 수 있습니다.

다음 예에서는 3개의 새 프로파일을 SYSTOOLS.OPT_PROFILE 테이블에 삽입하는 방법을 보여줍니다. 파일이 현재 디렉토리에 있다고 가정해봅니다.

1. 별도의 라인에서 각 프로파일에 대한 스키마, 이름 및 파일 이름이 있는 입력 파일 (예: `profiledata`)을 작성하십시오.

   ```
   "ROBERT","PROF1","ROBERT.PROF1.xml"
   "ROBERT","PROF2","ROBERT.PROF2.xml"
   "DAVID", "PROF1","DAVID.PROF1.xml"
   ```

2. IMPORT 명령을 실행하십시오.

   ```
   import from profiledata of del
     modified by lobsinfile
     insert into systools.opt_profile
   ```

기존 행을 갱신하려면 IMPORT명령에서 INSERT_UPDATE 옵션을 사용하십시오.

```
import from profiledata of del
  modified by lobsinfile
  insert_update into systools.opt_profile
```

프로파일 길이가 32 700바이트 미만이라고 가정하고 ROBERT.PROF1 프로파일을 ROBERT.PROF1.xml에 복사하려면 EXPORT 명령을 사용하십시오.

```
export to robert.prof1.xml of del
  select profile from systools.opt_profile
    where schema='ROBERT' and name='PROF1'
```

32 700바이트가 넘는 데이터를 익스포트하는 방법을 포함한 자세한 정보는 『EXPORT 명령』을 참조하십시오.

쿼리 최적화에 대한 데이터베이스 파티션 그룹 영향

파티션된 데이터베이스 환경에서 옵티마이저는 쿼리에 가장 적합한 액세스 플랜을 판별하면 테이블 공동 배치를 인식하고 사용합니다.

조인 쿼리에서 테이블을 자주 사용하는 경우 조인되는 각 테이블의 행을 동일한 데이터베이스 파티션에 배치하는 방식으로 테이블들을 여러 데이터베이스 파티션들로 분산해야 합니다. 조인 연산 중 두 개의 조인된 테이블의 데이터 공동 배치는 한 데이터베이스 파티션에서 다른 데이터베이스 파티션으로 데이터가 이동되는 것을 예방합니다. 두 테이블을 동일한 데이터베이스 파티션 그룹에 배치하여 데이터를 공동 배치하십시오.

테이블의 *크기*에 따라 데이터를 추가 데이터베이스 파티션에 분산시키면 쿼리를 실행하는 추정 시간이 줄어듭니다. 테이블 수, 테이블 *크기*, 테이블에서 데이터 위치 및 쿼리 유형(예: 조인 필요 여부)이 모두 쿼리 비용에 영향을 줍니다.

고급 통계 기능을 포함하여 정확한 카탈로그 통계 수집

정확한 데이터베이스 통계는 쿼리 최적화에 매우 중요합니다. 쿼리 성능에 중요한 모든 테이블에서 runstats 조작을 정기적으로 수행하십시오.

응용프로그램이 시스템 카탈로그 테이블을 직접 쿼리하는 경우 및 DDL(Data Definition Language) 명령문 실행 결과로 생긴 활동과 같이 상당한 카탈로그 갱신 활동이 있는 경우, 시스템 카탈로그 테이블에서 통계를 수집하려 할 수도 있습니다. 자동 통계 콜렉션을 사용하면 DB2 데이터 서버에서 자동으로 runstats 조작을 수행하도록 할 수 있습니다. 실시간 통계 콜렉션을 사용하면 DB2 데이터 서버에서 쿼리가 최적화되기 직전에 통계를 수집하여 훨씬 더 시기 적절한 통계를 제공하도록 할 수 있습니다.

RUNSTATS 명령을 사용하여 수동으로 통계를 수집할 경우, 최소한 다음 옵션을 사용해야 합니다.

```
RUNSTATS ON TABLE DB2USER.DAILY_SALES
  WITH DISTRIBUTION AND SAMPLED DETAILED INDEXES ALL
```

분산 통계는 옵티마이저가 데이터 비대칭을 인식하게 해 줍니다. 자세한 인덱스 통계는 특정 인덱스를 사용하여 테이블에 액세스할 때 데이터 페이지를 페치하는 데 필요한 입출력에 대한 추가 세부사항을 제공합니다. 자세한 인덱스 통계를 수집하면 대형 테이블의 경우 상당한 처리 시간 및 메모리가 소비됩니다. SAMPLED 옵션은 거의 동일한 정밀도로 자세한 인덱스 통계를 제공하지만 CPU 및 메모리의 일부를 필요로 합니다. 이러한 디폴트값은 테이블에 대해 통계 프로파일이 제공되지 않은 경우 자동 통계에서도 사용됩니다.

쿼리 성능을 향상시키려면 컬럼 그룹 통계 또는 LIKE 통계와 같은 보다 고급 통계 수집 또는 통계 뷰 작성을 고려하십시오.

컬럼 그룹 통계

쿼리에 두 개의 테이블을 조인하는 둘 이상의 Join 술어가 있는 경우, DB2 옵티마이저는 쿼리 실행을 위한 플랜을 선택하기 전에 각 술어가 얼마나 선택적인가를 계산합니다.

예를 들어, 다양한 색상, 탄성 및 품질의 원자재에서 제품을 생산하는 제조업자가 있다고 가정하십시오. 완제품은 사용된 원자재와 색상 및 탄성이 동일합니다. 제조업체는 다음 쿼리를 발행합니다.

```
SELECT PRODUCT.NAME, RAWMATERIAL.QUALITY
  FROM PRODUCT, RAWMATERIAL
  WHERE
    PRODUCT.COLOR = RAWMATERIAL.COLOR AND
    PRODUCT.ELASTICITY = RAWMATERIAL.ELASTICITY
```

이 쿼리는 모든 제품의 이름 및 원자재 품질을 리턴합니다. 두 개의 Join 술어가 있습니다.

```
PRODUCT.COLOR = RAWMATERIAL.COLOR
PRODUCT.ELASTICITY = RAWMATERIAL.ELASTICITY
```

옵티마이저는 두 개의 술어가 독립적이라고 가정합니다. 즉, 탄성의 모든 변형이 각 색상에 대해 발생합니다. 그런 다음 다른 색상의 수와 탄성 레벨의 수를 기반으로 각 테이블에 대한 카탈로그 통계 정보를 사용하여 술어 쌍의 전체 선택 빈도를 추정합니다. 이 추정을 기반으로, 예를 들어, 병합 조인보다는 중첩된 루프 조인을 선택하거나, 또는 그 반대로 선택할 수도 있습니다.

그러나 이러한 두 술어는 독립적이지 않을 수도 있습니다. 예를 들어, 매우 탄성이 있는 자재를 몇 가지 색상으로만 사용 가능할 수도 있고, 매우 탄성이 없는 자재를 탄성이 있는 자재와 다른 몇 가지 다른 색상으로만 사용 가능할 수도 있습니다. 이 경우, 술어의 조합된 선택 빈도는 더 적은 행을 제거하고 쿼리는 더 많은 행을 리턴합니다. 이 정보 없이, 옵티마이저는 더 이상 최상의 플랜을 선택하지 않을 수도 있습니다.

PRODUCT.COLOR 및 PRODUCT.ELASTICITY에서 컬럼 그룹 통계를 수집하려면 다음 RUNSTATS 명령을 발행하십시오.

```
RUNSTATS ON TABLE PRODUCT ON COLUMNS ((COLOR, ELASTICITY))
```

옵티마이저는 이러한 통계를 사용하여 상관의 경우를 감지하고 상관된 술어의 조합된 선택 빈도를 동적으로 조정하므로, 조인 *크기* 및 비용에 대한 보다 정확한 추정을 얻습니다.

쿼리에서 GROUP BY 또는 DISTINCT와 같은 키워드를 사용하여 데이터를 그룹화할 경우에도 컬럼 그룹 통계는 옵티마이저가 다른 그룹화의 수를 계산할 수 있게 해 줍니다.

다음 쿼리를 고려해보십시오.

```
SELECT DEPTNO, YEARS, AVG(SALARY)
  FROM EMPLOYEE
  GROUP BY DEPTNO, MGR, YEAR_HIRED
```

인덱스 또는 컬럼 그룹 통계 없이, 옵티마이저는 DEPTNO, MGR 및 YEAR_HIRED에서 고유 값 수를 산출하여 그룹화 수(및 이 경우 리턴되는 행의 수)를 추정합니다. 이 추정에서는 그룹화 키 컬럼이 독립적이라고 가정합니다. 그러나 각 관리자가 정확히 하나의 부서를 담당할 경우 이 가정은 잘못된 것일 수 있습니다. 또한 각 부서에서 매년 직원을 고용했을 가능성이 없습니다. 따라서, DEPTNO, MGR 및 YEAR_HIRED에서 고유 값 수를 산출한 것이 고유 그룹의 실제 수보다 과대 평가된 것일 수 있습니다.

DEPTNO, MGR 및 YEAR_HIRED에서 수집된 컬럼 그룹 통계는 옵티마이저에 이전 쿼리에 대한 정확한 고유 그룹화 수를 제공합니다.

```
RUNSTATS ON TABLE EMPLOYEE ON COLUMNS ((DEPTNO, MGR, YEAR_HIRED))
```

JOIN 술어 상관과 더불어, 옵티마이저는 다음과 같은, 단순 등호 술어가 있는 상관을 관리합니다.

```
DEPTNO = 'Sales' AND  MGR = 'John'
```

이 예에서, EMPLOYEE 테이블에 있는 DEPTNO 컬럼의 술어는 YEAR 컬럼의 술어와 관계가 없을 가능성이 큽니다. 그러나 각 부서는 일반적으로 한 번에 한 명의 관리자에 의해 관리되므로 DEPTNO와 MGR의 술어는 확실히 독립적이지 않습니다. 옵티마이저는 컬럼에 대한 통계 정보를 사용하여 고유 값의 조합된 수를 판별한 다음 카디널리티(cardinality) 추정을 조정하여 컬럼 간 상관을 설명합니다.

단순 등호 술어 상관

Join 술어 상관 이외에 옵티마이저는 COL = 유형의 단순 등호 술어를 사용하여 상관을 관리합니다.

예를 들어, 각각 MAKE(즉, 제조업체), MODEL, YEAR, COLOR 및 STYLE(예: 세단, 스테이션 왜건 또는 SUV)이 있는 여러 가지 유형의 자동차로 구성된 테이블을 가

정하십시오. 거의 모든 제조업체가 매년 모델 및 스타일마다 동일한 표준 색상을 생산하므로 COLOR의 술어는 MAKE, MODEL, STYLE 또는 YEAR의 술어와 독립적일 가능성이 높습니다. 그러나 단일 자동차 제조업체만 특정 이름의 모델을 생산하므로 MAKE 및 MODEL에 기초한 술어는 독립적이지 않습니다. 둘 이상의 자동차 제조업체가 사용하는 동일한 모델 이름은 거의 존재할 가능성이 없습니다.

두 컬럼 MAKE 및 MODEL에 대한 인덱스가 존재하거나 컬럼 그룹 통계가 수집되는 경우 옵티마이저는 인덱스 또는 컬럼에 대한 통계 정보를 사용하여 결합된 구별 값 수를 판별하고 이러한 두 컬럼 간 상관의 선택 빈도 또는 카디널리티(cardinality)를 조정합니다. 술어가 로컬 등호 술어인 경우 옵티마이저는 조정할 고유 인덱스가 필요하지 않습니다.

통계 뷰

DB2 비용 기반 옵티마이저는 액세스 플랜 연산자에 의해 처리된 카디널리티(cardinality) 또는 행 수의 추정을 사용하여 해당 연산자의 비용을 정확하게 산정합니다. 이 카디널리티(cardinality) 추정은 옵티마이저의 비용 모델에 대한 하나의 가장 중요한 입력이고, 해당 정밀도는 runstats 유틸리티가 데이터베이스에서 수집한 통계에 따라 크게 달라집니다.

표현식 관련 비교 (예: `price > MSRP + Dealer_markup`), 여러 테이블에 걸쳐 있는 관계(예: `product.name = 'Alloy wheels' and product.key = sales.product_key`) 또는 단순 비교 연산 및 독립 속성 관련 술어 이외의 모든 경우와 같은 보다 복잡한 관계를 나타내려면 보다 복잡한 통계가 필요합니다. 통계 뷰는 뷰에 의해 참조된 기본 테이블이 아니라 뷰에 의해 리턴된 결과 세트에서 통계가 수집되기 때문에 이러한 유형의 복잡한 관계를 나타낼 수 있습니다.

쿼리가 컴파일될 때 옵티마이저는 사용 가능한 통계 뷰에 쿼리를 일치시킵니다. 옵티마이저는 중간 결과 세트에 대한 카디널리티(cardinality) 추정을 계산할 때 뷰의 통계를 사용하여 더 나은 추정을 계산합니다.

옵티마이저가 통계 뷰를 사용하기 위해 쿼리에서 직접 통계 뷰를 참조할 필요는 없습니다. 옵티마이저는 구체화된 쿼리 테이블(MQT)에 사용된 것과 동일한 일치 메커니즘을 사용하여 쿼리를 통계 뷰에 일치시킵니다. 이 점에서, 통계 뷰는 영구적으로 저장되지 않고, 디스크 스페이스를 사용하지 않으며, 유지보수할 필요가 없다는 점을 제외하고는 MQT와 매우 유사합니다.

통계 뷰는 뷰를 처음으로 작성한 후 ALTER VIEW문을 사용하여 최적화에 사용 가능하게 함으로써 작성됩니다. 그런 다음 RUNSTATS 명령이 통계 뷰에 대해 실행되고, 뷰에 대한 통계로 시스템 카탈로그 테이블을 채웁니다. 예를 들어, 스타 스키마에서 사실 테이블과 시간 차원 테이블 간 조인을 나타내는 통계 뷰를 작성하려면 다음을 수행하십시오.

```
CREATE VIEW SV_TIME_FACT AS (
  SELECT T.* FROM TIME T, SALES S
    WHERE T.TIME_KEY = S.TIME_KEY)

ALTER VIEW SV_TIME_FACT ENABLE QUERY OPTIMIZATION

RUNSTATS ON TABLE DB2DBA.SV_TIME_FACT WITH DISTRIBUTION
```

이 통계 뷰는 카디널리티(cardinality) 추정을 향상시키고, 그 결과 다음과 같은 쿼리에 대한 액세스 플랜 및 쿼리 성능을 향상시키는 데 사용될 수 있습니다.

```
SELECT SUM(S.PRICE)
  FROM SALES S, TIME T, PRODUCT P
  WHERE
    T.TIME_KEY = S.TIME_KEY AND
    T.YEAR_MON = 200712 AND
    P.PROD_KEY = S.PROD_KEY AND
    P.PROD_DESC = 'Power drill'
```

통계 뷰 없으면, 옵티마이저는 특정 시간 차원 YEAR_MON 값에 해당하는 모든 사실 테이블 TIME_KEY 값이 사실 테이블 내에서 균일하게 발생한다고 가정합니다. 그러나 12월에 판매액이 특히 현저하여, 다른 달 동안보다 Sales 트랜잭션이 더 많은 결과가 생길 수도 있습니다.

자동 통계 콜렉션은 현재 통계 뷰에서 사용 가능하지 않습니다. 통계 뷰에 의해 참조되는 기본 테이블이 크게 갱신될 때마다 해당 뷰에서 통계를 수집하십시오.

통계 뷰 사용

쿼리를 최적화하는 데 해당 통계를 사용할 수 있으려면 먼저 최적화에 뷰를 사용 가능해야 합니다. 최적화에 사용 가능한 뷰를 *통계 뷰*라고 합니다.

통계 뷰가 아닌 뷰는 최적화에 사용 가능하지 않으며 *일반 뷰*라고 합니다. 처음 작성 시 뷰는 최적화에 사용 가능하지 않습니다. ALTER VIEW문을 사용하여 최적화에 뷰를 사용 가능하게 하십시오. 이 태스크를 수행하는 데 필요한 특권 및 권한에 대해서는 ALTER VIEW문의 설명을 참조하십시오. 뷰에 대해 runstats 유틸리티를 사용하는 데 필요한 특권 및 권한에 대해서는 RUNSTATS 명령의 설명을 참조하십시오.

다음 조건 중 하나에 해당되는 경우 최적화에 뷰를 사용할 수 없습니다.

- 뷰가 직접 또는 간접적으로 구체화된 쿼리 테이블(MQT)을 참조합니다(MQT 또는 통계 뷰가 통계 뷰를 참조할 수 있음).
- 뷰가 작동하지 않습니다.
- 뷰가 유형이 지정된 뷰입니다.
- 동일한 ALTER VIEW문에 또 다른 뷰 변경 요청이 있습니다.

최적화를 사용 가능하게 하기 위해 변경되고 있는 뷰의 정의에 다음 항목이 포함된 경우, 경고가 리턴되고 옵티마이저가 뷰의 통계를 이용하지 않습니다.

- aggregation 또는 distinct 연산
- union, except 또는 intersect 연산
- 스칼라 집계 (OLAP) 함수

1. 최적화에 뷰를 사용 가능하게 하십시오.

   ALTER VIEW문에서 ENABLE OPTIMIZATION 절을 사용하여 최적화에 뷰를 사용 가능하게 할 수 있습니다. 최적화에 사용 가능하게 된 뷰는 이후에 DISABLE OPTIMIZATION 절을 사용하여 최적화에 사용 불가능하게 할 수 있습니다. 예를 들어, 최적화에 MYVIEW를 사용 가능하게 하려면 다음을 입력하십시오.

   ```
   alter view myview enable query optimization
   ```

2. RUNSTATS 명령을 호출하십시오. 예를 들어, MYVIEW의 통계를 수집하려면 다음을 입력하십시오.

   ```
   runstats on table db2dba.myview
   ```

   분산 통계를 포함하여 뷰 통계를 수집하면서 행의 10퍼센트의 행 레벨 샘플링을 사용하려면 다음을 입력하십시오.

   ```
   runstats on table db2dba.myview with distribution tablesample bernoulli (10)
   ```

   분산 통계를 포함하여 뷰 통계를 수집하면서 페이지의 10퍼센트의 페이지 레벨 샘플링을 사용하려면 다음을 입력하십시오.

   ```
   runstats on table db2dba.myview with distribution tablesample system (10)
   ```

3. 선택사항: 뷰 정의의 영향을 받는 쿼리가 정적 SQL 패키지의 일부인 경우, 해당 패키지를 리바인드하여 새 통계의 결과로 생긴 액세스 플랜에 대한 변경을 이용하십시오.

최적화와 관련된 뷰 통계

CARD 및 COLCARD와 같이, 통계 뷰를 정의하는 쿼리의 데이터 분산에 특성을 부여하는 통계만 쿼리 최적화 동안 고려됩니다.

옵티마이저에서 사용하도록 뷰 레코드와 연관된 다음 통계를 수집할 수 있습니다.

- 테이블 통계(SYSCAT.TABLES, SYSSTAT.TABLES)
  - CARD - 뷰 결과에서 행 수
- 컬럼 통계(SYSCAT.COLUMNS, SYSSTAT.COLUMNS)
  - COLCARD - 뷰 결과에서 컬럼의 고유 값 수
  - AVGCOLLEN - 뷰 결과에서 컬럼의 평균 길이
  - HIGH2KEY - 뷰 결과에서 두 번째로 높은 컬럼 값
  - LOW2KEY - 뷰 결과에서 두 번째로 낮은 컬럼 값
  - NUMNULLS - 뷰 결과에서 컬럼의 널(NULL) 값 수
  - SUB_COUNT - 뷰 결과에서 컬럼의 평균 하위 요소 수

  - SUB_DELIM_LENGTH - 하위 요소를 분리하는 각 분리문자의 평균 길이
- 컬럼 분포 통계(SYSCAT.COLDIST, SYSSTAT.COLDIST)
  - DISTCOUNT - COLVALUE 통계보다 작거나 같은 고유 Quantile 값 수
  - SEQNO - 테이블에서 행을 고유하게 식별하는 데 도움이 되는 시퀀스 번호의 빈도 순위
  - COLVALUE - 빈도 또는 Quantile 통계가 수집되는 대상 데이터 값
  - VALCOUNT - 뷰 컬럼에서 데이터 값이 발생하는 빈도, 또는 Quantile의 경우 데이터 값(COLVALUE)보다 작거나 같은 값 수

데이터 분산을 설명하지 않는 통계(예: NPAGES 및 FPAGES)를 수집할 수 있지만 옵티마이저에서 무시됩니다.

시나리오: 통계 뷰를 사용하여 카디낼리티(cardinality) 추정 향상

데이터 웨어하우스에서 사실 테이블 정보는 종종 매우 동적으로 변경되는 반면에 차원 테이블 데이터는 정적입니다. 즉, 차원 속성 데이터는 긍정적으로나 부정적으로 사실 테이블 속성 데이터와 상관될 수 있습니다.

현재 옵티마이저에 사용 가능한 일반 기본 테이블 통계에서는 테이블 간 관계를 식별할 수 없습니다. MQT 및 통계 뷰의 컬럼 및 테이블 분산 통계를 사용하여 옵티마이저에서 이러한 유형의 카디낼리티(cardinality) 추정 오류를 정정하는 데 필요한 정보를 제공할 수 있습니다.

매년 7월 동안 판매된 골프채의 연간 판매 수익을 계산하는 다음 쿼리를 고려해보십시오.

```
select  sum(f.sales_price), d2.year
  from  product d1, period d2, daily_sales f
  where  d1.prodkey = f.prodkey
    and  d2.perkey = f.perkey
    and  d1.item_desc = 'golf club'
    and  d2.month = 'JUL'
  group by  d2.year
```

옵티마이저에서 PRODUCT 및 DAILY_SALESA 포함 세미조인이 가장 선별적인지, 또는 PERIOD 및 DAILY_SALES 포함 세미조인이 가장 선별적인지를 판별할 수 있다면 스타 조인 쿼리 실행 플랜이 이 쿼리에 대한 훌륭한 선택이 될 수 있습니다. 효과적인 스타 조인 플랜을 생성하려면 옵티마이저에서 인덱스 ANDing 조작의 외부 레그에 대한 가장 선별적인 세미조인을 선택할 수 있어야 합니다.

데이터 웨어하우스에는 종종 저장 셸프(shelf)에 더 이상 있지 않은 제품에 대한 레코드가 포함됩니다. 이로 인해 조인 후 PRODUCT 컬럼의 분산이 조인 전 분산과 크게 다르게 표시될 수 있습니다. 더 나은 정보의 부족으로, 옵티마이저에서 기본 테이블 통계만을 기반으로 하여 로컬 술어의 선택 빈도를 판별하기 때문에, 옵티마이저가 술어 `item_desc = 'golf club'`의 선택 빈도에 대해 지나치게 낙관적이 될 수 있습니다.

예를 들어, 골프채가 과거 생산된 제품의 1%를 나타내지만 현재 판매액의 20%를 차지할 경우, 조인 후 *item_desc*의 분산을 설명하는 통계가 없기 때문에 옵티마이저에서 `item_desc = 'golf club'`의 선택 빈도를 과대 평가할 수 있습니다. 그리고 모든 12개월의 판매액이 균일할 경우, 술어 `month = 'JUL'`의 선택 빈도는 약 8%가 되고, 따라서 술어 `item_desc = 'golf club'`의 선택 빈도 추정 오류가 옵티마이저에서 PRODUCT와 DAILY_SALES 간에 스타 조인 플랜 인덱스 ANDing 조작의 외부 레그에 비해 겉보기에 더 선별적인 세미조인을 수행하도록 잘못 야기합니다.

다음 예에서는 통계 뷰를 설정하여 이러한 유형의 문제를 해결하는 방법에 대한 단계별 설명을 제공합니다.

STORE, CUSTOMER, PRODUCT, PROMOTION 및 PERIOD가 차원 테이블이고, DAILY_SALES가 사실 테이블인, 일반 데이터 웨어하우스의 데이터베이스를 고려해보겠습니다. 다음 테이블에서는 이러한 테이블에 대한 정의를 제공합니다.

표 55. STORE(63 행)

컬럼	**storekey**	**store_number**	**city**	**state**	**district**	**...**
속성	integer not null primary key	char(2)	char(20)	char(5)	char(14)	...

표 56. CUSTOMER(1 000 000행)

컬럼	**custkey**	**name**	**address**	**age**	**gender**	**...**
속성	integer not null primary key	char(30)	char(40)	smallint	char(1)	...

표 57. PRODUCT(19 450행)

컬럼	**prodkey**	**category**	**item_desc**	**price**	**cost**	**...**
속성	integer not null primary key	integer	char(30)	decimal(11)	decimal(11)	...

표 58. PROMOTION(35행)

컬럼	**promokey**	**promotype**	**promodesc**	**promovalue**	**...**
속성	integer not null primary key	integer	char(30)	decimal(5)	...

표 59. PERIOD(2922행)

컬럼	**perkey**	**calendar_date**	**month**	**period**	**year**	**...**
속성	integer not null primary key	date	char(3)	smallint	smallint	...

표 60. DAILY_SALES(754 069 426행)

컬럼	storekey	custkey	prodkey	promokey	perkey	sales_price	...
속성	integer	integer	integer	integer	integer	decimal(11)	...

회사 관리자가 재방문 시 할인을 제공할 경우 소비자가 제품을 다시 구매할지 여부를 판별하고자 한다고 가정해봅니다. 또한 이 연구가 전국적으로 18개의 위치가 있는 상점 '01'에 대해서만 수행된다고 가정합니다. 표 61에서는 사용 가능한 다른 범주의 판촉에 대한 정보를 보여줍니다.

표 61. PROMOTION(35행)

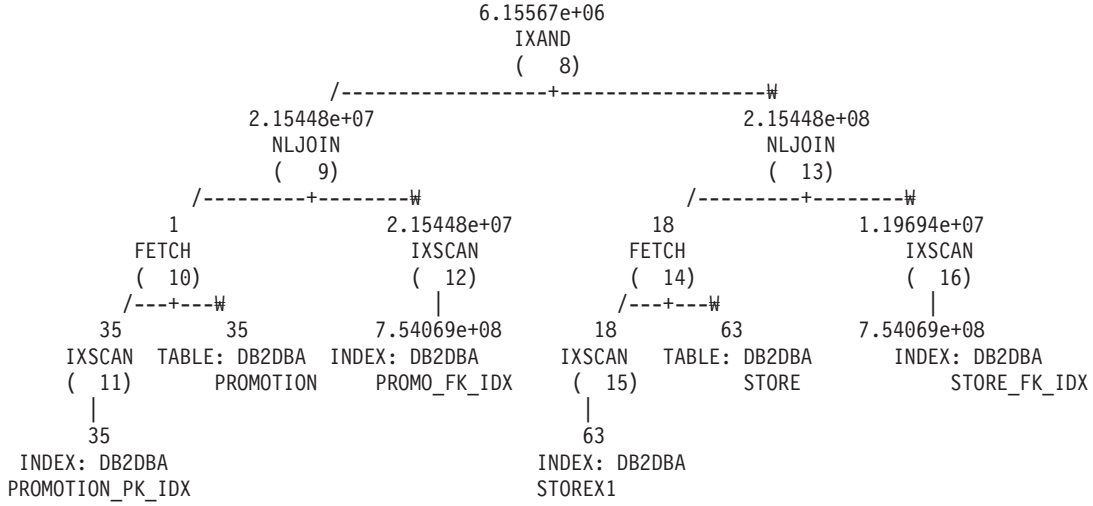
promotype	promodesc	COUNT (promotype)	percentage of total
1	재방문 고객	1	2.86%
2	쿠폰	15	42.86%
3	광고	5	14.29%
4	관리자 특별 할인	3	8.57%
5	재고 과잉 품목	4	11.43%
6	매대 디스플레이	7	20.00%

이 표에는 재방문한 고객에 대한 할인이 제공된 35가지 판촉 중 2.86%만 나타낸다고 표시됩니다.

다음 쿼리는 12 889 514의 계수를 리턴합니다.

```
select  count(*)
  from  store d1, promotion d2, daily_sales f
  where  d1.storekey = f.storekey
    and  d2.promokey = f.promokey
    and  d1.store_number = '01'
    and  d2.promotype = 1
```

이 쿼리는 옵티마이저에 의해 생성된 다음 플랜에 따라 실행됩니다. 이 다이어그램의 각 노드에서 첫 번째 행은 카디널리티(cardinality) 추정이고, 두 번째 행은 연산자 유형이며, 세 번째 행(괄호의 숫자)은 연산자 ID입니다.



중첩된 루프 조인(숫자 9)에서 옵티마이저는 판매된 제품의 약 2.86%가 할인된 가격에 같은 제품을 구입하려고 돌아 온 고객으로 인한 것이라고 추정합니다(2.15448e+07 ÷ 7.54069e+08 ≈ 0.0286). 이 값은 PROMOTION 테이블을 DAILY_SALES 테이블과 조인하기 전과 후에 동일합니다. 표 62에서는 조인 전후의 카디널리티(cardinality) 추정 및 해당 퍼센트를 요약합니다.

표 62. DAILY_SALES와의 조인 전후 카디널리티(cardinality) 추정

	조인 전		조인 후	
술어	계수	규정된 행의 퍼센트	계수	규정된 행의 퍼센트
store_number = '01'	18	28.57%	2.15448e+08	28.57%
promotype = 1	1	2.86%	2.15448e+07	2.86%

promotype = 1의 가능성이 store_number = '01'의 가능성보다 작기 때문에, 옵티마이저에서 스타 조인 플랜 인덱스 ANDing 조작의 외부 레그로 PROMOTION과 DAILY_SALES 간 세미조인을 선택합니다. 이를 통해 판촉 유형 1을 사용하여 약 6 155 670개 제품이 판매된 것으로 추정됩니다. — 인수 2.09에 의해 제공된 올바르지 않은 카디널리티(cardinality) 추정 (12 889 514 ÷ 6 155 670 ≈ 2.09).

무엇이 옵티마이저에서 두 술어를 충족하는 실제 레코드 수의 절반만 추정하도록 합니까? 상점 '01'은 모든 상점의 약 28.57%를 나타냅니다. 다른 상점이 상점 '01'보다 더 많은 판매액을 갖는다면 어떨까요(28.57% 미만)? 또는 상점 '01'에서 실제로 대부분의 제품을 판매했다면 어떨까요(28.57% 초과)? 마찬가지로, 표 62에 표시된 판촉 유형 1을 사용하여 판매된 제품 2.86%가 잘못 나온 것일 수 있습니다. DAILY_SALES의 실제 퍼센트는 추정된 퍼센트와 매우 다른 수치일 수 있습니다.

우리는 통계 뷰를 사용하여 옵티마이저에서 해당 추정을 정정하도록 도울 수 있습니다. 우선, 앞의 쿼리의 각 세미조인을 나타내는 2개의 통계 뷰를 작성해야 합니다. 첫 번째 통계 뷰는 모든 일별 판매액에 대한 상점의 분산을 제공합니다. 두 번째 통계 뷰는 모든 일별 판매액에 대한 판촉 유형의 분산을 나타냅니다. 각 통계 뷰는 특정 상점 번호 또는 판촉 유형에 대한 분산 정보를 제공할 수 있습니다. 이 예에서는 10% 샘플 비율을 사용하여 해당 뷰에 대해 DAILY_SALES에서 레코드를 검색하고 전역 임시 테이블에 저장합니다. 그런 다음 이러한 테이블을 쿼리하여 필요한 통계를 수집해 두 개의 통계 뷰를 갱신합니다.

1. DAILY_SALES와 STORE의 조인을 나타내는 뷰를 작성하십시오.

```
create view sv_store_dailysales as
  (select s.*
    from store s, daily_sales ds
    where s.storekey = ds.storekey)
```

2. DAILY_SALES와 PROMOTION의 조인을 나타내는 뷰를 작성하십시오.

```
create view sv_promotion_dailysales as
  (select p.*
    from promotion.p, daily_sales ds
    where p.promokey = ds.promokey)
```

3. 쿼리 최적화에 뷰를 사용하여 뷰를 통계 뷰로 만드십시오.

```
alter view sv_store_dailysales enable query optimization
alter view sv_promotion_dailysales enable query optimization
```

4. RUNSTATS 명령을 실행하여 뷰에 대한 통계를 수집하십시오.

```
runstats on table db2dba.sv_store_dailysales with distribution
runstats on table db2dba.sv_promotion_dailysales with distribution
```

5. 다시 최적화될 수 있도록 쿼리를 다시 실행하십시오. 재최적화 시, 옵티마이저는 SV_STORE_DAILYSALES 및 SV_PROMOTION_DAILYSALES를 쿼리와 일치시키고, 뷰 통계를 사용하여 사실 테이블과 차원 테이블 간 세미조인의 카디널리티(cardinality) 추정을 조정하여 이러한 통계 없이 선택된 세미조인의 원래 순서를 반대로 되게 합니다. 새 플랜은 다음과 같습니다.

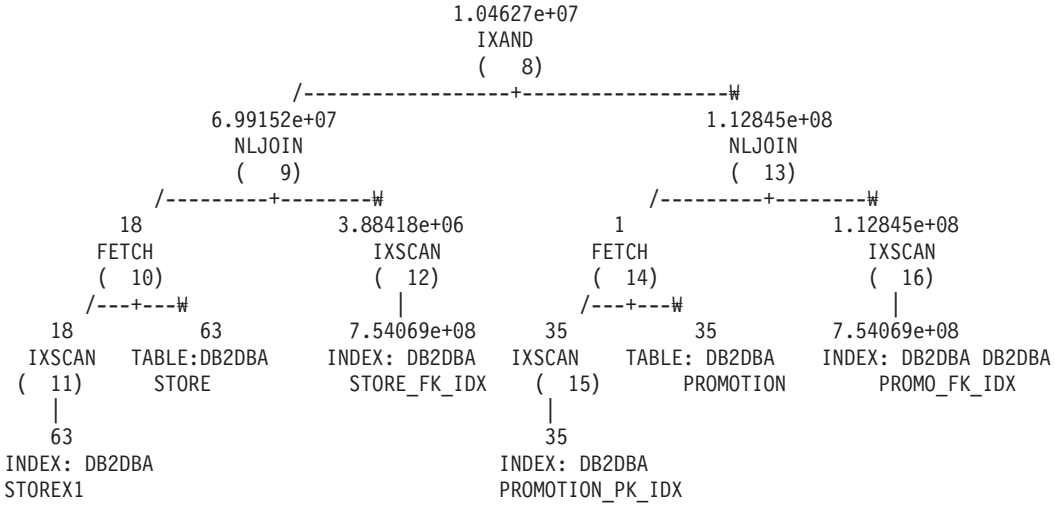


표 63에서는 각 세미조인에 대한 조인 전후의 카디널리티(cardinality) 조정 및 해당 퍼센트(필터링 효과)를 요약합니다.

표 63. DAILY_SALES와의 조인 전후 카디널리티(cardinality) 추정

	조인 전		조인 후(*통계 뷰 없음*)		조인 후 (*통계 뷰 있음*)	
술어	계수	규정된 행의 퍼센트	계수	규정된 행의 퍼센트	계수	규정된 행의 퍼센트
store_number = '01'	18	28.57%	2.15448e+08	28.57%	6.99152e+07	9.27%
promotype = 1	1	2.86%	2.15448e+07	2.86%	1.12845e+08	14.96%

이번에는, 인덱스 ANDing 플랜의 외부 레그에서 STORE와 DAILY_SALES 간 세미조인이 수행됩니다. 이는 두 통계 뷰에서 본질적으로 옵티마이저에 술어 `store_number = '01'`이 `promotype = 1`보다 더 많은 행을 필터링한다고 알려주기 때문입니다. 이 때, 옵티마이저는 약 10 462 700개 제품이 판매된 것으로 추정합니다. 이 추정은 인수 1.23에 의해 제공되며(12 889 514 ÷ 10 462 700 ≈ 1.23), 통계 뷰 없는 추정(395 페이지의 표 62)에 비해 크게 향상된 것입니다.

카탈로그 통계

쿼리 컴파일러가 쿼리 계획을 최적화할 때 결정은 데이터베이스 테이블, 인덱스 및 통계 뷰의 크기에 대한 통계 정보의 영향을 많이 받습니다. 이 정보는 시스템 카탈로그 테이블에 저장됩니다.

또한 옵티마이저는 테이블, 인덱스 및 통계 뷰의 특정 컬럼에서 데이터 분산에 대한 정보를 사용합니다(이러한 컬럼을 사용하여 행을 선택하거나 테이블을 조인하는 경우). 옵티마이저는 이 정보를 사용하여 각 쿼리의 대체 액세스 플랜 비용을 추정합니다.

인덱스의 클러스터 비율, 인덱스의 리프 페이지 수, 원래의 페이지를 오버플로우한 테이블 행 수 및 테이블에서 가득 찬 페이지 및 비어 있는 페이지 수에 대한 통계 정보도 수집할 수 있습니다. 이 정보를 사용하여 테이블 또는 인덱스를 재구성하는 시기를 결정할 수 있습니다.

파티션된 데이터베이스 환경의 테이블 통계는 유틸리티를 실행 중인 데이터베이스 파티션에 상주하는 테이블의 일부분 또는 테이블이 들어 있는 데이터베이스 파티션의 첫 번째 데이터베이스 파티션에 대해서만 수집됩니다. 통계 뷰에 대한 정보는 모든 데이터베이스 파티션에 대해 수집됩니다.

runstats 유틸리티로 갱신된 통계

카탈로그 통계는 RUNSTATS 명령을 발행하거나 ADMIN_CMD 프로시저를 호출하거나 db2Runstats API를 호출하여 시작할 수 있는 runstats 유틸리티로 갱신됩니다. 갱신은 수동 또는 자동으로 시작할 수 있습니다.

선언된 임시 테이블에 대한 통계는 시스템 카탈로그에 저장되지 않고 선언된 임시 테이블에 대한 카탈로그 정보를 표시하는 메모리 구조에 저장됩니다. 선언된 임시 테이블에서 runstats를 수행할 수 있습니다(유용한 경우도 있음).

runstats 유틸리티는 테이블 및 인덱스에 대해 다음 정보를 수집합니다.

- 행이 들어 있는 페이지 수
- 사용 중인 페이지 수
- 테이블의 행 수(*카디널리티(cardinality)*)
- 오버플로우된 행 수
- MDC(다차원 클러스터링) 테이블의 경우 데이터가 들어 있는 블록 수
- 파티션된 테이블의 경우 단일 데이터 파티션에서 데이터 클러스터링 수준
- 옵티마이저가 데이터가 균등하게 분산되지 않고 컬럼에 상당한 중복 값이 있는 테이블 및 통계 뷰의 효율적인 액세스 플랜을 추정하는 데 사용하는 데이터 분산 통계
- 옵티마이저가 인덱스를 통해 테이블 데이터에 액세스할 때 효율성을 판별하기 위해 사용하는 세부 인덱스 통계

- LIKE 술어의 하위 요소 통계. 특히 문자열에서 패턴 검색(예: `LIKE %disk%`)도 옵티마이저가 사용합니다.

runstats 유틸리티는 테이블의 각 데이터 파티션에 대해 다음 통계를 수집합니다. 이 통계는 파티션을 인식해야 하는지 여부를 판별하는 데만 사용됩니다.

- 행이 들어 있는 페이지 수
- 사용 중인 페이지 수
- 테이블의 행 수(카디낼리티(cardinality))
- 오버플로우된 행 수
- MDC 테이블의 경우, 데이터를 포함하는 블록 수

다음과 같은 경우에 분산 통계를 수집하지 않습니다.

- **num_freqvalues** 및 **num_quantiles** 데이터베이스 구성 매개변수가 0으로 설정된 경우
- 각 데이터 값이 고유한 경우와 같이 데이터 분산이 알려진 경우
- 컬럼에 LONG, LOB 또는 구조화된 데이터 유형이 포함된 경우
- 하위 테이블에 있는 행 유형의 경우(테이블 레벨 통계 NPAGES, FPAGES 및 OVERFLOW는 수집되지 않음)
- Quantile 분산이 요청되었지만 컬럼에 널(NULL)이 아닌 하나의 값만 있는 경우
- 확장 인덱스 또는 선언된 임시 테이블의 경우

runstats 유틸리티는 테이블 또는 통계 뷰의 각 컬럼과 인덱스 키의 첫 번째 컬럼에 대해 다음 정보를 수집합니다.

- 컬럼의 카디낼리티(cardinality)
- 컬럼의 평균 길이(데이터베이스 메모리 또는 임시 테이블에 컬럼을 저장할 때 필요한 평균 공간(바이트))
- 컬럼에서 두 번째로 높은 값
- 컬럼에서 두 번째로 낮은 값
- 컬럼의 널(NULL) 값 수

대형 오브젝트(LOB) 또는 LONG 데이터 유형이 있는 컬럼의 경우 runstats 유틸리티는 컬럼의 평균 길이 및 컬럼의 널(NULL) 값 수만 수집합니다. LOB 데이터가 데이터 페이지에 인라인으로 위치한 경우를 제외하고 컬럼의 평균 길이는 데이터 디스크립터의 길이를 나타냅니다. 디스크에 컬럼을 저장하는 데 필요한 평균 공간량은 이 통계의 값과 다를 수 있습니다.

runstats 유틸리티는 각 XML 컬럼에 대해 다음 정보를 수집합니다.

- NULL XML 문서 수
- 널(NULL)이 아닌 XML 문서 수

- 구별 경로 수
- 각 구별 경로의 노드 수 합계
- 각 구별 경로의 문서 수 합계
- 노드 수가 가장 큰 *k*쌍의 (경로, 노드 수)
- 문서 수가 가장 큰 *k*쌍의 (경로, 문서 수)
- 노드 수가 가장 큰 *k* 트리플의 (경로, 값, 노드 수)
- 문서 수가 가장 큰 *k* 트리플의 (경로, 값, 문서 수)
- 텍스트 또는 속성 값으로 이어지는 각 구별 경로의 경우
  - 이 경로가 사용할 수 있는 구별 값 수
  - 최고값
  - 최저값
  - 텍스트 또는 속성 노드 수
  - 텍스트 또는 속성 노드가 들어 있는 문서 수

XML 컬럼의 각 행에 XML 문서를 저장합니다. 경로 또는 경로-값 쌍의 노드 수는 경로 또는 경로-값 쌍이 접근할 수 있는 노드 수를 나타냅니다. 경로 또는 경로-값 쌍의 문서 수는 해당 경로 또는 경로-값 쌍이 들어 있는 문서 수를 나타냅니다.

runstats 유틸리티는 컬럼 그룹에 대해 다음 정보를 수집합니다.

- 컬럼 그룹에 대한 시간소인 기반 이름
- 컬럼 그룹의 카디널리티(cardinality)

runstats 유틸리티는 인덱스에 대해 다음 정보를 수집합니다.

- 인덱스 항목 수(*인덱스 카디널리티(cardinality)*)
- 리프 페이지 수
- 인덱스 레벨 수
- 인덱스에 테이블 데이터를 클러스터링하는 수준
- 데이터 파티션과 관련된 인덱스 키의 클러스터링 수준
- 인덱스 키 순서의 디스크에 위치한 리프 페이지와 인덱스가 차지하는 페이지 범위의 페이지 수 비율
- 인덱스의 첫 번째 컬럼에 있는 구별 값 수
- 인덱스의 처음 2, 3 및 4컬럼에 있는 구별 값 수
- 인덱스의 모든 컬럼에 있는 구별 값 수
- 인덱스 키 순서의 디스크에 위치한 리프 페이지 수(사이에 큰 갭이 거의 없음)
- 포함 컬럼 없이, 평균 리프 키 *크기*
- 포함 컬럼과 함께, 평균 리프 키 *크기*

- 모든 레코드 ID(RID)가 삭제된 것으로 표시된 페이지 수
- 모든 RID가 삭제된 것으로 표시되지는 않은 페이지에서 삭제된 것으로 표시된 RID 수

세부 인덱스 통계를 요청하는 경우 인덱스에 대한 테이블 데이터의 클러스터링 수준과 여러 버퍼 *크기*의 페이지 페치 추정에 대한 추가 정보를 수집합니다.

파티션된 인덱스의 경우 이 통계는 단일 인덱스 파티션을 나타냅니다. 인덱스의 첫 번째 컬럼, 인덱스의 처음 두 개, 세 개 및 네 개의 컬럼, 인덱스의 모든 컬럼에 있는 구별 값은 예외입니다. 인덱스 파티션마다의 통계도 인덱스 파티션이 인식되어야 하는지 여부를 판별하는 목적으로 수집됩니다.

카탈로그 통계 테이블

데이터베이스 테이블, 인덱스 및 통계 뷰의 *크기*에 대한 통계 정보가 시스템 카탈로그 테이블에 저장됩니다.

다음 표는 이 통계 정보에 대한 간단한 설명과 저장 위치를 나타냅니다.

- 『테이블』 컬럼은 RUNSTATS 명령의 FOR INDEXES 또는 AND INDEXES 옵션이 지정되지 않은 경우 특정 통계의 수집 여부를 표시합니다.
- 『인덱스』 컬럼은 FOR INDEXES 또는 AND INDEXES 옵션이 지정된 경우 특정 통계의 수집 여부를 표시합니다.

일부 통계는 테이블에서만 제공하고 일부는 인덱스에서만 제공하며 일부는 테이블 및 인데스 모두 제공할 수 있습니다.

- 표 1. 테이블 통계(SYSCAT.TABLES 및 SYSSTAT.TABLES)
- 표 2. 컬럼 통계(SYSCAT.COLUMNS 및 SYSSTAT.COLUMNS)
- 표 3. 다중 컬럼 통계(SYSCAT.COLGROUPS 및 SYSSTAT.COLGROUPS)
- 표 4. 다중 컬럼 분포 통계(SYSCAT.COLGROUPDIST 및 SYSSTAT.COLGROUPDIST)
- 표 5. 다중 컬럼 분포 통계(SYSCAT.COLGROUPDISTCOUNTS 및 SYSSTAT.COLGROUPDISTCOUNTS)
- 표 6. 인덱스 통계(SYSCAT.INDEXES 및 SYSSTAT.INDEXES)
- 표 7. 컬럼 분포 통계(SYSCAT.COLDIST 및 SYSSTAT.COLDIST)

표 4. 다중 컬럼 분포 통계(SYSCAT.COLGROUPDIST 및 SYSSTAT.COLGROUPDIST)와 표 5. 다중 컬럼 분포 통계 (SYSCAT.COLGROUPDISTCOUNTS 및 SYSSTAT.COLGROUPDISTCOUNTS)에 나열된 다중 컬럼 분포 통계는 runstats 유틸리티로 수집되지 않습니다. 수동으로 갱신할 수 없습니다.

표 64. 테이블 통계(SYSCAT.TABLES 및 SYSSTAT.TABLES)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스
FPAGES	테이블에서 사용 중인 페이지 수	Yes	Yes
NPAGES	행이 들어 있는 페이지 수	Yes	Yes
OVERFLOW	오버플로우된 행 수	Yes	No
CARD	테이블의 행 수(카디널리티(cardinality))	Yes	Yes(주 1 참조)
ACTIVE_BLOCKS	MDC 테이블의 경우 차지한 전체 블록 수	Yes	No

주:

1. 테이블에 정의된 인덱스가 없으며 인덱스에 대한 통계를 요청하는 경우 CARD 통계가 갱신되지 않습니다. 이전 CARD 통계가 보유됩니다.

표 65. 컬럼 통계(SYSCAT.COLUMNS 및 SYSSTAT.COLUMNS)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스
COLCARD	컬럼 카디널리티 (cardinality)	Yes	Yes(주 1 참조)
AVGCOLLEN	컬럼의 평균 길이	Yes	Yes(주 1 참조)
HIGH2KEY	컬럼의 두 번째 최고값	Yes	Yes(주 1 참조)
LOW2KEY	컬럼의 두 번째 최저값	Yes	Yes(주 1 참조)
NUMNULLS	컬럼의 널(NULL) 값 수	Yes	Yes(주 1 참조)
SUB_COUNT	평균 하위 요소 수	Yes	No(주 2 참조)
SUB_DELIM_LENGTH	하위 요소를 분리하는 각 분리문자의 평균 길이	Yes	No(주 2 참조)

주:

1. 인덱스 키의 첫 번째 컬럼에 대한 컬럼 통계가 수집됩니다.
2. 이러한 통계는 공백으로 구분된 일련의 하위 필드 또는 하위 요소가 있는 컬럼의 데이터에 대한 정보를 제공합니다. SUB_COUNT 및 SUB_DELIM_LENGTH 통계는 코드 페이지 속성이 1바이트 문자 세트(SBCS), FOR BIT DATA 또는 UTF-8인 유형 CHAR 및 VARCHAR의 컬럼일 경우에만 수집됩니다.

표 66. 다중 컬럼 통계(SYSCAT.COLGROUPS 및 SYSSTAT.COLGROUPS)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스
COLGROUPCARD	컬럼 그룹의 카디널리티 (cardinality)	Yes	No

표 67. 다중 컬럼 분포 통계(SYSCAT.COLGROUPDIST 및 SYSSTAT.COLGROUPDIST)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스
TYPE	F = 빈도 값 Q = Quantile 값	Yes	No
ORDINAL	그룹에 있는 컬럼의 서수	Yes	No
SEQNO	*n*번째 TYPE 값을 표시하는 시퀀스 번호 *n*	Yes	No
COLVALUE	문자 리터럴 또는 널(NULL) 값인 데이터 값	Yes	No

표 68. 다중 컬럼 분포 통계(SYSCAT.COLGROUPDISTCOUNTS 및 SYSSTAT.COLGROUPDISTCOUNTS)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스
TYPE	F = 빈도 값 Q = Quantile 값	Yes	No
SEQNO	*n*번째 TYPE 값을 표시하는 시퀀스 번호 *n*	Yes	No
VALCOUNT	TYPE = F인 경우 VALCOUNT는 이 SEQNO의 컬럼 그룹에 대한 COLVALUE의 어커런스 수입니다. TYPE = Q인 경우 VALCOUNT는 값이 이 SEQNO의 컬럼 그룹에 대한 COLVALUE 이하인 행 수입니다.	Yes	No
DISTCOUNT	TYPE = Q인 경우 이 컬럼에는 이 SEQNO의 컬럼 그룹에 대한 COLVALUE 이하인 구별 값 수가 들어 있습니다. 사용 불가능한 경우 널(NULL)입니다.	Yes	No

표 69. 인덱스 통계(SYSCAT.INDEXES 및 SYSSTAT.INDEXES)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스
NLEAF	인덱스 리프 페이지 수	No	Yes
NLEVELS	인덱스 레벨 수	No	Yes
CLUSTERRATIO	테이블 데이터의 클러스터링 수준	No	Yes(주 2 참조)

표 69. 인덱스 통계(SYSCAT.INDEXES 및 SYSSTAT.INDEXES) (계속)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스
CLUSTERFACTOR	클러스터링 세부 수준	No	Detailed(주 1, 2 참조)
DENSITY	인덱스가 차지하는 페이지 범위의 페이지 수에 대한 SEQUENTIAL_PAGES 비율(퍼센트)	No	Yes
FIRSTKEYCARD	인덱스의 첫 번째 컬럼에 있는 구별 값 수	No	Yes
FIRST2KEYCARD	인덱스의 처음 두 컬럼에 있는 구별 값 수	No	Yes
FIRST3KEYCARD	인덱스의 처음 세 컬럼에 있는 구별 값 수	No	Yes
FIRST4KEYCARD	인덱스의 처음 네 컬럼에 있는 구별 값 수	No	Yes
FULLKEYCARD	모든 레코드 ID(RID)가 삭제된 것으로 표시된 인덱스의 키 값을 제외하고 인덱스의 모든 컬럼에 있는 구별 값 수	No	Yes
PAGE_FETCH_PAIRS	여러 가지 버퍼 크기에 대한 페이지 페치 추정	No	Detailed(주 1, 2 참조)
AVGPARTITION_CLUSTERRATIO	단일 데이터 파티션의 데이터 클러스터링 수준	No	Yes(주 2 참조)
AVGPARTITION_CLUSTERFACTOR	단일 데이터 파티션에서 클러스터링 수준에 대한 세부 측정	No	Detailed(주 1, 2 참조)
AVGPARTITION_PAGE_FETCH_PAIRS	단일 데이터 파티션에 따라 생성된 여러 가지 버퍼 크기에 대한 페이지 페치 추정	No	Detailed(주 1, 2 참조)
DATAPARTITION_CLUSTERFACTOR	인덱스 스캔 중 데이터 파티션 참조 수	No(주 6 참조)	Yes(주 6 참조)
SEQUENTIAL_PAGES	인덱스 키 순서의 디스크에 위치한 리프 페이지 수(사이에 큰 갭이 거의 없음)	No	Yes
AVERAGE_SEQUENCE_PAGES	순서대로 액세스 가능한 평균 인덱스 페이지 수. 프리페처가 순서대로 감지할 수 있는 인덱스 페이지 수입니다.	No	Yes

표 69. 인덱스 통계(SYSCAT.INDEXES 및 SYSSTAT.INDEXES) (계속)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스
AVERAGE_RANDOM_PAGES	순차 페이지 액세스 간 평균 무작위 인덱스 페이지 수	No	Yes
AVERAGE_SEQUENCE_GAP	시퀀스 간 갭	No	Yes
AVERAGE_SEQUENCE_FETCH_PAGES	순서대로 액세스 가능한 평균 테이블 페이지 수. 프리페처가 인덱스를 사용하여 테이블 행을 페치할 때 순서대로 감지할 수 있는 테이블 페이지 수입니다.	No	Yes(주 4 참조)
AVERAGE_RANDOM_FETCH_PAGES	인덱스를 사용하여 테이블 행을 페치할 때 순차 페이지 액세스 간 평균 무작위 테이블 페이지 수	No	Yes(주 4 참조)
AVERAGE_SEQUENCE_FETCH_GAP	인덱스를 사용하여 테이블 행을 페치할 때 시퀀스 간 갭	No	Yes(주 4 참조)
NUMRIDS	삭제된 RID를 포함한 인덱스의 RID 수	No	Yes
NUMRIDS_DELETED	모든 RID가 삭제된 것으로 표시된 리프 페이지의 RID를 제외하고 삭제된 것으로 표시된 인덱스의 전체 RID 수	No	Yes
NUM_EMPTY_LEAFS	모든 RID가 삭제된 것으로 표시된 전체 리프 페이지 수	No	Yes
INDCARD	인덱스 항목 수(인덱스 카디널리티(cardinality))	No	Yes

표 69. 인덱스 통계(SYSCAT.INDEXES 및 SYSSTAT.INDEXES) (계속)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스

주:

1. 세부 인덱스 통계는 RUNSTATS 명령에서 DETAILED절을 지정하여 수집합니다.
2. 테이블이 충분한 크기(약 25페이지 이상)가 아닌 경우 DETAILED절을 사용하여 CLUSTERFACTOR 및 PAGE_FETCH_PAIRS를 수집하지 않습니다. 이 경우 CLUSTERRATIO는 -1(수집하지 않음)입니다. 테이블이 상대적으로 작은 경우 runstats 유틸리티는 CLUSTERRATIO만 수집합니다. CLUSTERFACTOR 및 PAGE_FETCH_PAIRS는 수집하지 않습니다. DETAILED절이 지정되지 않은 경우 CLUSTERRATIO만 수집합니다.
3. 이 통계는 해당 테이블에 속한 인덱스가 있는 파일의 페이지 수 퍼센트를 측정합니다. 하나의 인덱스만 정의된 테이블의 경우 DENSITY는 100이어야 합니다. 옵티마이저에서는 DENSITY를 사용하여 인덱스 페이지가 프리페치된 경우에 읽을 수 있는 다른 인덱스의 관련이 없는 페이지 수를 추정합니다.
4. 이 통계는 테이블이 DMS 테이블 스페이스에 있는 경우 계산할 수 없습니다.
5. 명령 호출 시 통계 수집이 지정된 경우에도 인덱스 로드 또는 작성 조작 중 프리페치 통계를 수집하지 않습니다. 프리페치 통계는 **seqdetect** 데이터베이스 구성 매개변수가 NO로 설정된 경우에도 수집되지 않습니다.
6. 테이블의 runstats 옵션이 『No』인 경우 테이블 통계 수집 시 통계를 수집하지 않습니다. 인덱스의 runstats 옵션이 『Yes』인 경우 RUNSTATS 명령을 INDEXES 옵션과 함께 사용하는 경우 통계를 수집합니다.

표 70. 컬럼 분포 통계(SYSCAT.COLDIST 및 SYSSTAT.COLDIST)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스
DISTCOUNT	TYPE = Q인 경우 DISTCOUNT는 COLVALUE 통계 이하인 구별 값 수입니다.	Distribution(주 2 참조)	No
TYPE	행이 자주 사용되는 값 또는 Quantile 통계를 제공하는지 여부를 나타내는 표시기	Distribution	No
SEQNO	테이블에서 행을 고유하게 식별하는 데 유용한 시퀀스 번호의 빈도 순위	Distribution	No
COLVALUE	빈도 또는 Quantile 통계를 수집하는 데이터 값	Distribution	No
VALCOUNT	컬럼에서 데이터 값이 나타나는 빈도. Quantile의 경우 데이터 값 (COLVALUE) 이하인 값의 수입니다.	Distribution	No

표 70. 컬럼 분포 통계(SYSCAT.COLDIST 및 SYSSTAT.COLDIST) (계속)

통계	설명	RUNSTATS 옵션	
		테이블	인덱스
주: 1. 컬럼 분포 통계는 RUNSTATS 명령에서 WITH DISTRIBUTION절을 지정하여 수집합니다. 분포 통계는 컬럼 값이 균일하지 않은 경우 수집할 수 없습니다. 2. DISTCOUNT는 인덱스의 첫 번째 키 컬럼인 경우에만 수집합니다.			

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 ON으로 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 사용

되지 않는 경우 비동기 통계 콜렉션이 두 시간 간격으로 발생합니다. 이것은 일부 응용 프로그램에 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.

- 스로틀된 runstats 유틸리티를 사용하여 비동기 통계 콜렉션이 수행됩니다. 스로틀링은 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 경쟁 및 오버헤드가 방지됩니다. 통계 캐시의 통계는 후속 SQL 컴파일 요청에 사용 가능합니다.
- 테이블 당 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 충족

한다 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 제작이 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 제작이 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 테이블에는 다양한 값의 CURRENT EXPLAIN MODE 특수 레지스터에 따른 동작이 요약되어 있습니다.

표 71. 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 크기를 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 ON으로 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 파

티션된 데이터베이스 환경, 특정 페더레이티드 데이터베이스 환경 또는 파티션 내 병렬 처리가 사용되는 환경에서는 사용 가능하지 않습니다.

자동 통계 콜렉션을 사용하려면 다음을 수행하십시오.

1. **auto_maint**, **auto_tbl_maint** 및 **auto_runstats** 데이터베이스 구성 매개변수를 ON으로 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 ON으로 설정하십시오. **auto_maint** 매개변수는 **auto_stats_prof** 및 **auto_prof_upd**의 상위입니다.
3. 선택사항: 실시간 통계 수집을 사용하려면 **auto_stmt_stats** 구성 매개변수를 ON으로 설정하십시오. 쿼리를 최적화해야 할 때마다 명령문 컴파일 시간에 테이블 통계가 자동으로 수집됩니다. **auto_maint** 매개변수는 **auto_stmt_stats**의 상위입니다.

통계 프로파일을 사용하여 통계 수집:

runstats 유틸리티는 특정 테이블에 대해 수집할 통계의 유형 (예: 테이블 통계, 인덱스 통계 또는 분산 통계)을 지정하는 통계 프로파일을 등록하고 사용하기 위한 옵션을 제공합니다. 이 기능은 사용자가 runstats 옵션을 저장하여 나중에 편리하게 사용할 수 있도록 함으로써 통계 콜렉션을 간편하게 해 줍니다.

동시에 프로파일을 등록하고 통계를 수집하려면 프로파일 설정 옵션과 함께 RUNSTATS 명령을 실행하십시오. 프로파일만 등록하려면 프로파일만 설정 옵션과 함께 RUNSTATS 명령을 실행하십시오. 이미 등록된 프로파일을 사용하여 통계를 수집하려면 프로파일 사용 옵션과 함께 RUNSTATS 명령을 실행하십시오.

특정 테이블에 대한 통계 프로파일에서 현재 지정되어 있는 옵션을 확인하려면 SYSCAT.TABLES 카탈로그 뷰를 쿼리하십시오. 예를 들어, 다음과 같습니다.

```
select statistics_profile from syscat.tables where tabname = 'EMPLOYEE'
```

자동 통계 프로파일링

DB2 자동 통계 프로파일링 기능을 사용하여 통계 프로파일을 자동으로 생성할 수도 있습니다. 이 기능을 사용할 경우, 데이터베이스 활동에 대한 정보가 수집되고 쿼리 피드백 웨어하우스에 저장됩니다. 그런 다음 이 데이터를 기반으로 통계 프로파일이 생성됩니다. 이 기능을 사용하면 특정 워크로드에 어떤 통계가 관련되는지에 대한 불확실성을 줄일 수 있습니다.

자동 통계 프로파일링은 자동으로 생성된 통계 프로파일에 포함된 정보를 기반으로 통계 유지보수 조작을 스케줄하는 자동 통계 콜렉션과 함께 사용할 수 있습니다.

자동 통계 프로파일링을 사용하려면 적절한 데이터베이스 구성 매개변수를 설정하여 자동 테이블 유지보수가 이미 사용 가능하게 설정되었는지 확인하십시오. 자세한 정보는

『auto_maint - 자동 유지보수 구성 매개변수』를 참조하십시오. **auto_stats_prof** 구성 매개변수는 쿼리 피드백 데이터의 콜렉션을 활성화하고, **auto_prof_upd** 구성 매개변수는 자동 통계 콜렉션에서 사용하기 위한 통계 프로파일의 생성을 활성화합니다.

자동 통계 프로파일 생성은 파티션된 데이터베이스 환경, 특정 페더레이티드 데이터베이스 환경 및 파티션 내 병렬 처리가 사용되는 경우 지원되지 않습니다.

자동 통계 프로파일링은 많은 술어가 있고, 대형 조인을 사용하거나 확장 그룹화를 지정하는 대규모 복합 쿼리를 실행하는 시스템에 가장 적합합니다. 주로 트랜잭션 워크로드가 있는 시스템에는 그리 적합하지 않습니다.

런타임 모니터링의 성능 오버헤드를 쉽게 견딜 수 있는 개발 환경에서, **auto_stats_prof** 및 **auto_prof_upd** 구성 매개변수를 ON으로 설정하십시오. 테스트 시스템에서 실제 데이터 및 쿼리를 사용할 경우, 추가 모니터링 오버헤드를 초래하지 않고 쿼리에 이익이 될 수 있는 프로덕션 시스템으로 해당 통계 프로파일이 전송될 수 있습니다.

프로덕션 환경에서, 특정 쿼리 세트에서 성능 문제가 감지될 경우(잘못된 통계의 원인이 될 수 있는 문제), **auto_stats_prof** 구성 매개변수를 ON으로 설정하고 일정 기간 동안 목표 워크로드를 실행할 수 있습니다. 자동 통계 프로파일링은 쿼리 피드백을 분석하고 SYSTOOLS.OPT_FEEDBACK_RANKING 테이블에 권장사항을 작성합니다. 이러한 권장사항을 점검하고 수동으로 통계 프로파일을 적절하게 조정할 수 있습니다. DB2 서버에서 이러한 권장사항을 기반으로 통계 프로파일을 자동으로 갱신하도록 하려면 **auto_stats_prof**를 사용할 때 **auto_prof_upd**를 사용하십시오.

쿼리 피드백 웨어하우스 작성

자동 통계 프로파일링에 필요한 쿼리 피드백 웨어하우스는 SYSTOOLS 스키마에서 5개의 테이블로 구성됩니다. 이러한 테이블은 통계 콜렉션을 위한 권장사항뿐 아니라 쿼리 실행 중에 발견되는 술어에 대한 정보를 저장합니다. 5개의 테이블은 다음과 같습니다.

- OPT_FEEDBACK_PREDICATE
- OPT_FEEDBACK_PREDICATE_COLUMN
- OPT_FEEDBACK_QUERY
- OPT_FEEDBACK_RANKING
- OPT_FEEDBACK_RANKING_COLUMN

쿼리 피드백 웨어하우스를 작성하려면 SYSINSTALLOBJECTS 프로시저를 사용하십시오. SYSTOOLS 스키마에서 오브젝트를 작성하거나 삭제하는 데 사용되는 이 프로시저에 대한 자세한 정보는 『SYSINSTALLOBJECTS』를 참조하십시오.

자동 통계 콜렉션 및 프로파일링에서 사용되는 스토리지:

자동 통계 콜렉션 및 재구성 기능에서는 사용자의 데이터베이스에 속한 테이블에 작업 데이터를 저장합니다. 이러한 테이블은 SYSTOOLSPACE 테이블 스페이스에서 작성됩니다.

SYSTOOLSPACE는 데이터베이스가 활성화될 때 디폴트 옵션을 사용하여 자동으로 작성됩니다. 이러한 테이블에 대한 스토리지 요구사항은 데이터베이스의 테이블 수에 비례하고 테이블당 약 1KB로 추정될 수 있습니다. 이것이 사용자의 데이터베이스에서 상당한 크기를 차지할 경우, 테이블 스페이스를 삭제한 후 재작성하여 스토리지를 적절히 할당할 수도 있습니다. 테이블 스페이스에서 자동 유지보수 및 상태 모니터링 테이블이 자동으로 재작성된다 하더라도 해당 테이블에서 캡처되었던 실행기록은 테이블 스페이스를 삭제할 때 손실됩니다.

자동 통계 콜렉션 활동 로깅:

통계 로그는 특정 데이터베이스에 대해 발생한 모든 통계 콜렉션 활동(수동과 자동 모두)의 레코드입니다.

통계 로그의 디폴트 이름은 db2optstats.*번호*.log입니다. 이것은 $DIAGPATH/events 디렉토리에 상주합니다. 통계 로그는 회전 로그입니다. 로그 동작은 **DB2_OPTSTATS_LOG** 레지스트리 변수에 의해 제어됩니다.

통계 로그는 SYSPROC.PD_GET_DIAG_HIST 테이블 함수를 사용하여 쿼리하거나 직접 볼 수 있습니다. 이 테이블 함수는 시간 소인, DB2 인스턴스 이름, 데이터베이스 이름, 프로세스 ID, 프로세스 이름 및 스레드 ID와 같은 로그된 이벤트에 대한 표준 정보를 포함하는 많은 컬럼을 리턴합니다. 이 로그는 여러 로깅 기능에서 사용할 일반 컬럼도 포함합니다. 다음 표에서는 통계 로그에서 이러한 일반 컬럼이 사용되는 방법을 설명합니다.

표 72. 통계 로그 파일의 일반 컬럼

컬럼 이름	데이터 유형	설명
OBJTYPE	VARCHAR(64)	이벤트가 적용되는 오브젝트의 유형입니다. 통계 로깅에서, 이것은 수집될 통계의 유형입니다. OBJTYPE은 프로세스가 시작되거나 중지될 때 통계 콜렉션 백그라운드 프로세스를 참조할 수 있습니다. 또한 샘플링 테스트, 초기 샘플링 및 테이블 평가 등 자동 통계 콜렉션에 의해 수행되는 활동을 참조할 수도 있습니다. 통계 콜렉션 활동에 대해 가능한 값은 다음과 같습니다. **TABLE STATS** 테이블 통계가 수집됩니다. **INDEX STATS** 인덱스 통계가 수집됩니다. **TABLE AND INDEX STATS** 테이블 통계와 인덱스 통계가 모두 수집됩니다. 자동 통계 콜렉션에 대해 가능한 값은 다음과 같습니다. **EVALUATION** 자동 통계 백그라운드 콜렉션 프로세스에서 평가 단계를 시작합니다. 이 단계 동안에는 갱신된 통계가 필요한지 여부를 판별하기 위해 테이블이 점검되고, 필요한 경우 통계가 수집됩니다. **INITIAL SAMPLING** 샘플링을 사용하여 테이블에 대한 통계가 수집됩니다. 샘플링된 통계는 시스템 카탈로그에 저장됩니다. 이를 통해 통계가 없는 테이블에 대해 자동 통계 콜렉션이 신속히 진행될 수 있습니다. 후속 조작에서는 샘플링 없이 통계를 수집합니다. 초기 샘플링은 자동 통계 콜렉션의 평간 단계 동안 수행됩니다. **SAMPLING TEST** 샘플링을 사용하여 테이블에 대한 통계가 수집됩니다. 샘플링된 통계가 시스템 카탈로그에 저장되지 않습니다. 샘플링된 통계가 현재 카탈로그 통계와 비교되어 이 테이블에 대한 전체 통계를 수집해야 하는지 여부 및 시기를 판별됩니다. 샘플링은 자동 통계 콜렉션의 평간 단계 동안 수행됩니다. **STATS DAEMON** 통계 디먼은 실시간 통계 처리에 의해 제출된 요청을 처리하는 데 사용됩니다. 이 오브젝트 유형은 백그라운드 프로세스가 시작되거나 중지될 때 로그됩니다.
OBJNAME	VARCHAR(255)	이벤트가 적용되는 오브젝트의 이름입니다(사용 가능한 경우). 통계 로깅에서, 이것은 테이블 또는 인덱스 이름입니다. OBJTYPE이 STATS DAEMON 또는 EVALUATION인 경우, OBJNAME은 데이터베이스 이름이고 OBJNAME_QUALIFIER는 NULL입니다.
OBJNAME_QUALIFIER	VARCHAR(255)	통계 로깅에서, 이것은 테이블 또는 인덱스의 스키마입니다.

표 72. 통계 로그 파일의 일반 컬럼 (계속)

컬럼 이름	데이터 유형	설명
EVENTTYPE	VARCHAR(24)	이벤트 유형은 이 이벤트와 연관된 조치입니다. 통계 로깅에 대해 가능한 값은 다음과 같습니다. **COLLECT** 이 조치는 통계 콜렉션 조작에 대해 로그됩니다. **START** 이 조치는 실시간 통계 백그라운드 프로세스 (OBJTYPE = STATS DAEMON) 또는 자동 통계 콜렉션 평가 단계 (OBJTYPE = EVALUATION)가 시작될 때 로그됩니다. **STOP** 이 조치는 실시간 통계 백그라운드 프로세스 (OBJTYPE = STATS DAEMON) 또는 자동 통계 콜렉션 평가 단계 (OBJTYPE = EVALUATION)가 중지될 때 로그됩니다. **ACCESS** 이 조치는 통계 콜렉션 목적으로 테이블에 대한 액세스 시도가 이루어질 때 로그됩니다. 이 이벤트 유형은 오브젝트가 사용 불가능할 때 실패한 액세스 시도를 로그하는 데 사용됩니다.
FIRST_EVENTQUALIFIERTYPE	VARCHAR(64)	첫 번째 이벤트 규정자의 유형입니다. 이벤트 규정자는 이벤트에 의해 영향을 받는 내용을 설명하는 데 사용됩니다. 통계 로깅에서, 첫 번째 이벤트 규정자는 이벤트가 발생한 시간에 대한 시간소인입니다. 첫 번째 이벤트 규정자 유형의 경우, 값은 AT입니다.
FIRST_EVENTQUALIFIER	CLOB(16k)	이벤트에 대한 첫 번째 규정자입니다. 통계 로깅에서, 첫 번째 이벤트 규정자는 통계 이벤트가 발생한 시간에 대한 시간소인입니다. 통계 이벤트의 시간소인은 TIMESTAMP 컬럼에서 나타난 대로, 로그 레코드의 시간소인과 다를 수도 있습니다.
SECOND_EVENTQUALIFIERTYPE	VARCHAR(64)	두 번째 이벤트 규정자의 유형입니다. 통계 로깅에서, 이 값은 BY 또는 NULL일 수 있습니다. 이 필드는 다른 이벤트 유형에서는 사용되지 않습니다.

표 72. 통계 로그 파일의 일반 컬럼 (계속)

컬럼 이름	데이터 유형	설명
SECOND_EVENTQUALIFIER	CLOB(16k)	이벤트에 대한 두 번째 규정자입니다. 통계 로깅에서, 이것은 COLLECT 이벤트 유형에 대해 통계가 수집된 방법을 나타냅니다. 가능한 값은 다음과 같습니다. **User** LOAD, REDISTRIBUTE 또는 RUNSTATS 명령을 사용하거나 CREATE INDEX문을 발행하여 DB2 사용자에 의해 통계 콜렉션이 수행되었습니다. **Synchronous** DB2 서버에 의해 SQL문 컴파일 시간에 통계 콜렉션이 수행되었습니다. 통계는 시스템 카탈로그가 아닌 통계 캐시에 저장됩니다. **Synchronous sampled** DB2 서버에 의해 SQL문 컴파일 시간에 샘플링을 사용하여 통계 콜렉션이 수행되었습니다. 통계는 시스템 카탈로그가 아닌 통계 캐시에 저장됩니다. **Fabricate** 데이터 및 인덱스 관리 프로그램에 대해 유지보수되는 정보를 사용하여 SQL문 컴파일 시간에 통계가 제작되었습니다. 통계는 시스템 카탈로그가 아닌 통계 캐시에 저장됩니다. **Fabricate partial** 데이터 및 인덱스 관리 프로그램에 대해 유지보수되는 정보를 사용하여 SQL문 컴파일 시간에 일부 통계만 제작되었습니다. 특히, 특정 컬럼에 대한 HIGH2KEY 및 LOW2KEY 값만 제작되었습니다. 통계는 시스템 카탈로그가 아닌 통계 캐시에 저장됩니다. **Asynchronous** DB2 백그라운드 프로세스에 의해 통계가 수집되고 시스템 카탈로그에 저장됩니다. 이 필드는 다른 이벤트 유형에서는 사용되지 않습니다.
THIRD_EVENTQUALIFIERTYPE	VARCHAR(64)	세 번째 이벤트 규정자의 유형입니다. 통계 로깅에서, 이 값은 DUE TO 또는 NULL일 수 있습니다.

표 72. 통계 로그 파일의 일반 컬럼 (계속)

컬럼 이름	데이터 유형	설명
THIRD_EVENTQUALIFIER	CLOB(16k)	이벤트에 대한 세 번째 규정자입니다. 통계 로깅에서, 이것은 통계 활동이 완료될 수 없는 이유를 나타냅니다. 가능한 값은 다음과 같습니다. **Timeout** 동기 통계 콜렉션이 시간 예산을 초과했습니다. **Error** 오류로 인해 통계 활동에 실패했습니다. **RUNSTATS error** RUNSTATS 오류로 인해 동기 통계 콜렉션에 실패했습니다. 일부 오류의 경우, 통계가 수집될 수 없더라도 SQL문 컴파일이 완료되었을 수도 있습니다. 예를 들어, 통계를 수집할 메모리가 충분하지 않은 경우, SQL문 컴파일이 계속됩니다. **Object unavailable** 액세스할 수 없어 데이터베이스 오브젝트에 대해 통계를 수집하지 못했습니다. 일부 가능한 원인은 다음과 같습니다. • 오브젝트가 강한 독점(Z) 모드로 잠김 • 오브젝트가 상주하는 테이블 스페이스를 사용할 수 없음 • 테이블 인덱스를 재작성해야 함 **Conflict** 다른 응용프로그램에서 이미 동기 통계를 수집하고 있어 동기 통계 수집이 수행되지 않았습니다. FULLREC 컬럼 또는 db2diag.log 파일에서 오류 세부사항을 확인하십시오.
EVENTSTATE	VARCHAR(255)	이벤트의 결과로서 조치 또는 오브젝트의 상태입니다. 통계 로깅에서, 이것은 통계 조작의 상태를 표시합니다. 가능한 값은 다음과 같습니다. • Start • Success • Failure

예

이 예에서는 쿼리에서 PD_GET_DIAG_HIST를 호출하여 현재 시간소인 이전 최대 1년까지의 이벤트에 대한 통계 로그 레코드를 리턴합니다.

```
select pid, tid,
    substr(eventtype, 1, 10),
    substr(objtype, 1, 30) as objtype,
    substr(objname_qualifier, 1, 20) as objschema,
    substr(objname, 1, 10) as objname,
    substr(first_eventqualifier, 1, 26) as event1,
    substr(second_eventqualifiertype, 1, 2) as event2_type,
    substr(second_eventqualifier, 1, 20) as event2,
    substr(third_eventqualifiertype, 1, 6) as event3_type,
    substr(third_eventqualifier, 1, 15) as event3,
    substr(eventstate, 1, 20) as eventstate
  from table(sysproc.pd_get_diag_hist
```

```
  ('optstats', 'EX', 'NONE',
    current_timestamp - 1 year, cast(null as timestamp))) as sl
order by timestamp(varchar(substr(first_eventqualifier, 1, 26), 26));
```

결과는 통계 이벤트의 시간을 나타내는 FIRST_EVENTQUALIFIER 컬럼에 저장된 시간소인에 따라 정렬됩니다.

```
PID   TID          EVENTTYPE OBJTYPE              OBJSCHEMA OBJNAME    EVENT1                  EVENT2_ EVENT2              EVENT3_ EVENT3   EVENTSTATE
                                                                                                TYPE                        TYPE
----- ------------ --------- -------------------- --------- --------- ----------------------- ------- ------------------- ------- -------- ----------
28399          1082145120 START     STATS DAEMON               -               PROD_DB    2007-07-09-18.37.40.398905 -       -                   -       -        success
28389        183182027104 COLLECT   TABLE AND INDEX STATS      DB2USER         DISTRICT   2007-07-09-18.37.43.261222 BY      Synchronous         -       -        start
28389        183182027104 COLLECT   TABLE AND INDEX STATS      DB2USER         DISTRICT   2007-07-09-18.37.43.407447 BY      Synchronous         -       -        success
28399          1082145120 COLLECT   TABLE AND INDEX STATS      DB2USER         CUSTOMER   2007-07-09-18.37.43.471614 BY      Asynchronous        -       -        start
28399          1082145120 COLLECT   TABLE AND INDEX STATS      DB2USER         CUSTOMER   2007-07-09-18.37.43.524496 BY      Asynchronous        -       -        success
28399          1082145120 STOP      STATS DAEMON               -               PROD_DB    2007-07-09-18.37.43.526212 -       -                   -       -        success
28389        183278496096 COLLECT   TABLE STATS                DB2USER         ORDER_LINE 2007-07-09-18.37.48.676524 BY      Synchronous sampled -       -        start
28389        183278496096 COLLECT   TABLE STATS                DB2USER         ORDER_LINE 2007-07-09-18.37.53.677546 BY      Synchronous sampled DUE TO  Timeout   failure
28389          1772561034 START     EVALUATION                 -               PROD_DB    2007-07-10-12.36.11.092739 -       -                   -       -        success
28389          8231991291 COLLECT   TABLE AND INDEX STATS      DB2USER         DISTRICT   2007-07-10-12.36.30.737603 BY      Asynchronous        -       -        start
28389          8231991291 COLLECT   TABLE AND INDEX STATS      DB2USER         DISTRICT   2007-07-10-12.36.34.029756 BY      Asynchronous        -       -        success
28389          1772561034 STOP      EVALUATION                 -               PROD_DB    2007-07-10-12.36.39.685188 -       -                   -       -        success
```

대형 통계 로그에 대한 쿼리 성능 향상:

통계 로그 파일이 대형인 경우, 로그 레코드를 테이블에 복사하고, 인덱스를 작성한 다음 통계를 수집하여 쿼리 성능을 향상시킬 수 있습니다.

프로시저

1. 로그 레코드에 대한 적절한 컬럼이 있는 테이블을 작성하십시오.

```
create table db2user.stats_log (
  pid         bigint,
  tid         bigint,
  timestamp   timestamp,
  dbname      varchar(128),
  retcode     integer,
  eventtype   varchar(24),
  objtype     varchar(30),
  objschema   varchar(20),
  objname     varchar(30),
  event1_type varchar(20),
  event1      timestamp,
  event2_type varchar(20),
  event2      varchar(40),
  event3_type varchar(20),
  event3      varchar(40),
  eventstate  varchar(20))
```

2. SYSPROC.PD_GET_DIAG_HIST에 대해 쿼리의 커서를 선언하십시오.

```
declare c1 cursor for
  select pid, tid, timestamp, dbname, retcode, eventtype,
      substr(objtype, 1, 30) as objtype,
      substr(objname_qualifier, 1, 20) as objschema,
      substr(objname, 1, 30) as objname,
      substr(first_eventqualifiertype, 1, 20),
      substr(first_eventqualifier, 1, 26),
      substr(second_eventqualifiertype, 1, 20),
      substr(second_eventqualifier, 1, 40),
      substr(third_eventqualifiertype, 1, 20),
      substr(third_eventqualifier, 1, 40),
      substr(eventstate, 1, 20)
```

```
from table (sysproc.pd_get_diag_hist
  ('optstats', 'EX', 'NONE',
    current_timestamp - 1 year, cast(null as timestamp ))) as sl
```

3. 통계 로그 레코드를 테이블로 로드하십시오.

```
load from c1 of cursor replace into db2user.stats_log
```

4. 인덱스를 작성한 다음 테이블에서 통계를 수집하십시오.

```
create index sl_ix1 on db2user.stats_log(eventtype, event1);
create index sl_ix2 on db2user.stats_log(objtype, event1);
create index sl_ix3 on db2user.stats_log(objname);

runstats on table db2user.stats_log
  with distribution and sampled detailed indexes all;
```

통계 수집 및 갱신 지침

runstats 유틸리티는 테이블, 인덱스 및 통계 뷰에 대한 통계를 수집하여 옵티마이저에 액세스 플랜 선택에 필요한 정확한 정보를 제공합니다.

다음과 같은 경우 runstats 유틸리티를 사용하여 통계를 수집하십시오.

- 테이블로 데이터가 로드되었으며 적합한 인덱스가 작성된 후
- 테이블에서 새 인덱스를 작성한 후
- reorg 유틸리티를 사용하여 테이블을 재구성한 후
- 갱신, 삽입 또는 삭제 조작을 통해 테이블 및 인덱스가 많이 수정된 후
- 성능이 중요한 응용프로그램을 바인드하기 전
- 현재 및 이전 통계를 비교할 경우
- 프리페치 값이 변경된 경우
- REDISTRIBUTE DATABASE PARTITION GROUP 명령을 실행한 후
- XML 컬럼이 있는 경우. runstats를 사용하여 XML 컬럼에 대해서만 통계를 수집하는 경우 로드 조작 또는 이전 runstats 조작 중 수집된 비XML 컬럼에 대한 기존 통계를 보유합니다. 일부 XML 컬럼에 대한 통계가 이전에 수집된 경우 현재 runstats 조작에 이러한 컬럼이 포함되지 않았으면 이러한 통계를 교체하거나 삭제합니다.

runstats 성능을 향상시키고 통계를 저장하는 데 사용되는 디스크 스페이스를 절약하려면 데이터 분산 통계를 수집해야 하는 컬럼만 지정하십시오.

runstats를 실행한 후 응용프로그램을 리바인드해야 합니다. 새 통계를 사용할 수 있는 경우 쿼리 옵티마이저는 다른 액세스 플랜을 선택할 수도 있습니다.

전체 통계 세트를 한 번에 수집할 수 없는 경우 오브젝트의 서브세트에서 runstats 유틸리티를 사용하십시오. 이러한 오브젝트에 대해 진행 중인 활동으로 인해 불일치가 발생한 경우 쿼리 최적화 중 경고 메시지(SQL0437W, 이유 코드 6)가 리턴됩니다. 이러한 상태가 발생하면 runstats를 다시 사용하여 분산 통계를 갱신하십시오.

인덱스 통계를 해당 테이블과 동기화하려면 테이블 및 인덱스 통계를 동시에 수집하십시오. 통계가 마지막으로 수집된 후 테이블이 광범위하게 수정된 경우 해당 테이블의 인덱스 통계만 갱신하면 두 통계 세트가 서로 동기화되지 않습니다.

프로덕션 시스템에서 runstats 유틸리티를 사용하면 워크로드 성능에 부정적인 영향을 줄 수 있습니다. 유틸리티는 이제 높은 수준의 데이터베이스 활동 중 runstats 실행의 성능 영향을 제한하는 데 사용할 수 있는 자동제한 옵션을 지원합니다.

파티션된 데이터베이스 환경에서 테이블의 통계를 수집할 때 runstats는 유틸리티를 실행하는 데이터베이스 파티션에서만 작동합니다. 이 데이터베이스 파티션의 결과는 다른 데이터베이스 파티션으로 외삽(extrapolate)됩니다. 이 데이터베이스 파티션이 테이블의 필수 분할 영역을 포함하지 않을 경우, 필요한 데이터를 포함하는 데이터베이스 파티션 그룹의 첫 번째 데이터베이스 파티션으로 요청이 보내집니다.

통계 뷰의 통계는 뷰에서 참조하는 기본 테이블이 포함된 모든 데이터베이스 파티션에서 수집합니다.

runstats의 효율성 및 통계의 유용성을 개선하려면 다음 추가 정보를 고려하십시오.

- 테이블을 조인하는 데 사용되는 컬럼 또는 WHERE, GROUP BY절이나 쿼리의 비슷한 절에 참조된 컬럼의 경우에만 통계를 수집하십시오. 컬럼이 인덱스화된 경우 RUNSTATS 명령에서 ONLY ON KEY COLUMNS절을 사용하여 이러한 컬럼을 지정할 수 있습니다.
- 특정 테이블 및 컬럼의 **num_freqvalues** 및 **num_quantiles** 데이터베이스 구성 매개변수 값을 사용자 정의하십시오.
- SAMPLE DETAILED절을 사용하여 세부 인덱스 통계를 수집하십시오. 이 경우 세부 인덱스 통계를 위해 수행되는 백그라운드 계산량이 줄어듭니다. SAMPLE DETAILED절은 통계를 수집하는 데 필요한 시간을 줄이고 대부분의 경우 적합한 정밀도를 작성합니다.
- 채운 테이블의 인덱스를 작성하는 경우 COLLECT STATISTICS절을 사용하여 인덱스 작성 시 통계를 작성하십시오.
- 상당한 수의 테이블 행을 추가하거나 제거하는 경우 또는 통계를 수집할 컬럼의 데이터가 갱신된 경우 runstats를 다시 사용하여 통계를 갱신하십시오.
- runstats는 단일 데이터베이스 파티션에서만 통계를 수집하므로 모든 데이터베이스 파티션에 일관적으로 데이터가 분산되지 않은 경우 통계의 정확성이 떨어집니다. 데이터 분산이 왜곡된 것으로 의심되는 경우 runstats 유틸리티를 사용하기 전에 REDISTRIBUTE DATABASE PARTITION GROUP 명령을 사용하여 데이터베이스 파티션으로 데이터를 다시 분산시키십시오.

카탈로그 통계 수집:

runstats 유틸리티를 사용하여 테이블, 인덱스 및 통계 뷰에서 카탈로그 통계를 수집하십시오. 쿼리 옵티마이저는 이 정보를 사용하여 쿼리에 대한 최상의 액세스 플랜을 선택합니다.

이 유틸리티를 사용하는 데 필요한 특권 및 권한에 대해서는 RUNSTATS 명령의 설명을 참조하십시오. 카탈로그 통계를 수집하려면 다음을 수행하십시오.

1. 통계 정보를 수집하려는 테이블, 인덱스 또는 통계 뷰를 데이터베이스에 연결하십시오.
2. DB2 명령행에서 적절한 옵션과 함께 RUNSTATS 명령을 실행하십시오. 이러한 옵션을 통해 테이블, 인덱스 또는 통계 뷰에 대해 실행되는 쿼리에 대해 수집되는 통계를 사용자 정의할 수 있습니다.
3. runstats 조작이 완료되면, COMMIT문을 발행하여 잠금을 해제하십시오.
4. 통계 정보를 갱신한 테이블, 인덱스 또는 통계 뷰에 액세스하는 패키지를 리바인드하십시오.

주:

1. runstats 유틸리티는 별칭의 사용을 지원하지 않습니다. 쿼리가 페더레이티드 데이터베이스에 액세스할 경우, runstats 유틸리티를 사용하여 모든 데이터베이스의 테이블에 대한 통계를 갱신한 후, 리모트 테이블에 액세스하는 별칭을 삭제하고 재작성하여 옵티마이저에 대해 새 통계를 사용 가능하게 만드십시오.
2. 파티션된 데이터베이스 환경에서 테이블의 통계를 수집할 때 runstats는 유틸리티를 실행하는 데이터베이스 파티션에서만 작동합니다. 이 데이터베이스 파티션의 결과는 다른 데이터베이스 파티션으로 외삽(extrapolate)됩니다. 이 데이터베이스 파티션이 테이블의 필수 분할 영역을 포함하지 않을 경우, 필요한 데이터를 포함하는 데이터베이스 파티션 그룹의 첫 번째 데이터베이스 파티션으로 요청이 보내집니다.

   통계 뷰의 통계는 뷰에서 참조하는 기본 테이블이 포함된 모든 데이터베이스 파티션에서 수집합니다.

테이블 데이터의 샘플에 대한 통계 수집:

테이블 통계는 쿼리 옵티마이저에서 쿼리를 위한 최상의 액세스 플랜을 선택하는 데 사용되므로 통계가 최신 상태로 유지되는 것이 중요합니다. 데이터베이스의 *크기*가 계속해서 증가하면서 효율적인 통계 콜렉션이 점점 더 어려운 과제가 되고 있습니다.

효과적인 방법은 테이블 데이터의 무작위 샘플에서 통계를 수집하는 것입니다. 입출력 바운드 또는 프로세서 바운드 시스템의 경우, 성능 이점이 엄청납니다.

DB2 제품을 통해 통계 콜렉션에 대한 데이터를 효과적으로 샘플링하고, 잠재적으로는 높은 등급의 정밀도를 유지하면서 runstats 유틸리티의 성능을 *크게* 향상시킬 수 있습니다.

두 가지 샘플링 방법, 행 레벨 샘플링과 페이지 레벨 샘플링이 사용 가능합니다. 이러한 샘플링 방법에 대한 설명은 『쿼리의 데이터 샘플링』을 참조하십시오.

선택된 각 페이지에 대해 하나의 입출력 조작만 필요하므로 페이지 레벨 샘플링의 성능은 매우 좋습니다. 행 레벨 샘플링에서는 전체 테이블 스캔으로 모든 테이블 페이지가 검색되므로 입출력 비용이 줄어들지 않습니다. 그러나 행 레벨 샘플링은 통계 콜렉션이 프로세서 집약적이기 때문에 입출력 양이 줄어들지 않는다 하더라도 상당한 성능 향상을 제공합니다.

행 레벨 샘플링은 데이터 값이 고도로 클러스터된 환경에서 페이지 레벨 샘플링보다 더 나은 샘플을 제공합니다. 페이지 레벨 샘플링에 비해 행 레벨 샘플 세트는 각 데이터 페이지에서 *P* 퍼센트의 행을 포함하므로 데이터를 더 잘 반영할 수 있습니다. 페이지 레벨 샘플링에서는 *P* 퍼센트의 페이지의 모든 행이 샘플 세트에 포함됩니다. 행이 테이블에 무작위로 분산된 경우, 행 샘플링된 통계의 정밀도는 페이지 샘플링된 통계의 정밀도와 유사합니다.

이전 샘플이 다시 생성되는 REPEATABLE 옵션이 사용된 경우가 아니면, RUNSTATS 명령의 반복된 호출을 통해 각 샘플이 무작위로 생성됩니다. 이 옵션은 데이터가 상수로 유지되는 테이블에 대해 일관성 있는 통계가 필요한 경우에 유용할 수 있습니다.

하위 요소 통계:

패턴의 마지막이 아닌 위치에 % 와일드 카드 문자를 사용하여 LIKE 술어를 지정할 경우, 하위 요소 구조에 대한 기본 정보를 수집해야 합니다.

와일드 카드 LIKE 술어(예를 들어, `SELECT...FROM DOCUMENTS WHERE KEYWORDS LIKE '%simulation%'`)뿐 아니라, 컬럼 및 쿼리도 하위 요소 통계의 이점을 얻으려면 특정 기준에 부합해야 합니다.

테이블 컬럼은 공백으로 구분된 하위 요소 또는 하위 필드를 포함해야 합니다. 예를 들어, 4행 테이블 DOCUMENTS는 텍스트 검색 용도로 관련 키워드 목록이 있는 KEYWORDS 컬럼을 포함합니다. KEYWORDS의 값은 다음과 같습니다.

```
'database simulation analytical business intelligence'
'simulation model fruit fly reproduction temperature'
'forestry spruce soil erosion rainfall'
'forest temperature soil precipitation fire'
```

이 예에서, 각 컬럼 값은 5개의 하위 요소로 구성되고, 각각은 하나의 공백에 의해 다른 하위 요소와 구분된 단어(키워드)입니다.

쿼리는 WHERE 절에서 이러한 컬럼을 참조해야 합니다.

옵티마이저는 항상 각 술어에 일치하는 행 수를 추정합니다. 이 와일드 카드 LIKE 술어의 경우, 옵티마이저는 일치되는 컬럼에 함께 병합된 일련의 요소가 포함되어 있다고 가정하고, 선행 및 후행 % 문자를 제외한 문자열의 길이를 기반으로 각 요소의 길이를

추정합니다. 하위 요소 통계를 수집할 경우, 옵티마이저는 분리문자 및 각 하위 요소의 길이에 대한 정보를 갖습니다. 이러한 추가 정보를 사용하여 술어에 일치되는 행 수를 보다 정확하게 추정할 수 있습니다.

하위 요소 통계를 수집하려면 LIKE STATISTICS 옵션과 함께 RUNSTATS 명령을 실행하십시오.

하위 요소에 대한 runstats 통계:

runstats 유틸리티는 LIKE STATISTICS절을 지정하는 경우 코드 페이지 속성이 1바이트 문자 세트(SBCS), FOR BIT DATA 또는 UTF-8인 CHAR 및 VARCHAR 유형의 컬럼에 대한 통계를 수집합니다.

SUB_COUNT
평균 하위 요소 수

SUB_DELIM_LENGTH
하위 요소를 분리하는 각 분리문자의 평균 길이. 이 컨텍스트의 분리문자는 하나 이상의 연속 공백 문자입니다.

DB2_LIKE_VARCHAR 레지스트리 변수는 옵티마이저가 `<column > like '%<character-string>%'` 형식의 술어를 처리하는 방식에 영향을 줍니다. 이 레지스트리 변수에 대한 자세한 정보는 『쿼리 컴파일러 변수』를 참조하십시오.

하위 요소 통계의 값을 조사하려면 SYSCAT.COLUMNS 카탈로그 뷰를 쿼리하십시오. 예를 들어, 다음과 같습니다.

```
select substr(colname, 1, 16), sub_count, sub_delim_length
  from syscat.columns where tabname = 'DOCUMENTS'
```

LIKE STATISTICS절을 사용하는 경우 runstats 유틸리티를 완료하려면 시간이 오래 걸릴 수 있습니다. 이 옵션을 고려 중인 경우 이 추가 오버헤드에 대한 쿼리 성능의 향상을 평가하십시오.

카탈로그 통계 수동 갱신을 위한 일반 규칙:

카탈로그 통계를 갱신할 때 가장 중요한 일반 규칙은 다양한 통계의 유효한 값, 범위 및 형식이 해당 통계의 뷰에 저장되는지 확인하는 것입니다.

다양한 통계들 간의 관계 일관성을 보존하는 것도 중요합니다. 예를 들어, SYSSTAT.COLUMNS의 COLCARD는 SYSSTAT.TABLES의 CARD보다 작아야 합니다(컬럼의 구별 값 수는 테이블의 행 수보다 클 수 없음). COLCARD를 100에서 25로 줄이고 CARD를 200에서 50으로 줄인다고 가정하십시오. 먼저 SYSSTAT.TABLES를 갱신하는 경우 CARD는 COLCARD 미만이므로 오류가 리턴됩니다.

그러나 충돌을 감지하기 어렵고 오류가 리턴되지 않는 경우가 있습니다. 특히 영향을 받은 통계가 다른 카탈로그 테이블에 저장되는 경우가 있습니다.

카탈로그 통계를 갱신하기 전에 최소한 다음을 확인하십시오.

- 숫자 통계가 -1이거나 0 이상인지 확인하십시오.
- 퍼센트를 표시하는 숫자 통계(예: SYSSTAT.INDEXES의 CLUSTERRATIO)가 0 - 100의 범위에 속하는지 확인하십시오.

테이블 작성 시 카탈로그 통계는 -1로 설정되어 테이블에 통계가 없음을 표시합니다. 통계를 수집할 때까지 DB2 서버는 SQL 또는 XQuery문 컴파일 및 최적화를 위해 디폴트값을 사용합니다. 새 값이 디폴트값과 일치하지 않는 경우 테이블 또는 인덱스 통계 갱신이 실패할 수 있습니다. 따라서 테이블을 작성한 후 테이블 또는 인덱스의 통계를 갱신하려고 하기 전에 runstats 유틸리티를 사용하는 것이 좋습니다.

주:

1. 행 유형의 경우 하위 테이블의 테이블 레벨 통계 NPAGES, FPAGES 및 OVERFLOW를 갱신할 수 없습니다.
2. 파티션 레벨 테이블 및 인덱스 통계는 갱신할 수 없습니다.

컬럼 통계 수동 갱신을 위한 규칙:

SYSSTAT.COLUMNS 카탈로그 뷰에서 통계를 갱신할 때 따라야 하는 특정 지침이 있습니다.

- HIGH2KEY 또는 LOW2KEY 값을 수동으로 갱신할 때 다음을 확인하십시오.
  - 값이 해당 사용자 컬럼의 데이터 유형에 유효한지 확인하십시오.
  - 값의 길이는 33 또는 대상 컬럼 데이터 유형의 최대 길이 중 작은 값이어야 하며 추가 인용 부호를 포함하지 않은 값입니다(포함할 경우 문자열 길이가 68로 증가할 수 있음). 즉, HIGH2KEY 또는 LOW2KEY 값을 판별할 때 해당 사용자 컬럼에서 값의 처음 33자만 고려합니다.
  - 값은 비용 계산 시와 마찬가지로 UPDATE문의 SET절과 함께 사용할 수 있는 방식으로 저장됩니다. 문자열의 경우 문자열의 처음과 끝에 작은따옴표가 추가되며 문자열에 이미 있는 모든 인용 부호마다 인용 부호가 추가됨을 의미합니다. 사용자 컬럼 값 및 HIGH2KEY 또는 LOW2KEY에서 해당 값의 예는 표 73을 참조하십시오.

표 73. 데이터 유형별 HIGH2KEY 및 LOW2KEY 값

사용자 컬럼의 데이터 유형	사용자 데이터	해당 **HIGH2KEY** 또는 **LOW2KEY** 값
INTEGER	-12	-12
CHAR	abc	‘abc’
CHAR	ab’c	‘ab’’c’

  - 해당 컬럼에서 구별 값 수가 세 개를 초과할 때마다 HIGH2KEY가 LOW2KEY 보다 큽니다.
- 컬럼의 카디널리티(SYSSTAT.COLUMNS의 COLCARD)는 해당 테이블 또는 통계 뷰의 카디널리티(SYSSTAT.TABLES의 CARD)보다 클 수 없습니다.
- 컬럼의 널(NULL) 값 수(SYSSTAT.COLUMNS의 NUMNULLS)는 해당 테이블 또는 통계 뷰의 카디널리티(SYSSTAT.TABLES의 CARD)보다 클 수 없습니다.
- LONG 또는 대형 오브젝트(LOB) 데이터 유형으로 정의된 컬럼의 경우 통계가 지원되지 않습니다.

테이블 및 별칭 통계 수동 갱신을 위한 규칙:

SYSSTAT.TABLES 카탈로그 뷰에서 통계를 갱신할 때 따라야 하는 특정 지침이 있습니다.

- SYSSTAT.TABLES에서 갱신할 수 있는 통계 값은 CARD, FPAGES, NPAGES 및 OVERFLOW 뿐이며 MDC(다차원 클러스터링) 테이블의 경우 ACTIVE_BLOCKS입니다.
- CARD는 SYSSTAT.COLUMNS의 해당 테이블에 대한 모든 COLCARD 값 이상이어야 합니다.
- CARD는 NPAGES보다 커야 합니다.
- FPAGES는 NPAGES보다 커야 합니다.
- NPAGES는 임의의 인덱스에서 PAGE_FETCH_PAIRS 컬럼에 있는 임의의 『페치』 값 이하여야 합니다(이 통계가 인덱스와 관련된 경우).
- CARD는 임의의 인덱스에서 PAGE_FETCH_PAIRS 컬럼에 있는 임의의 『페치』 값 이하가 될 수 없습니다(이 통계가 인덱스와 관련된 경우).

페더레이티드 데이터베이스 시스템에서 리모트 뷰를 통해 별칭의 통계를 수동으로 갱신할 때에는 주의하십시오. 별칭이 리턴할 행 수와 같은 통계 정보는 이 리모트 뷰를 평가하는 실제 비용을 반영하지 않을 수도 있으므로 DB2 옵티마이저가 혼동할 수 있습니다. 그러나 리모트 뷰가 통계 갱신을 활용할 수 있는 경우가 있습니다. SELECT 목록에 컬럼 함수를 적용하지 않고 단일 기본 테이블에 정의된 리모트 뷰가 이러한 경우에 해당합니다. 복합 뷰의 경우 각 쿼리를 조정하는 복합 조정 프로세스가 필요할 수 있습니다. DB2 옵티마이저가 이러한 뷰의 비용을 정확하게 도출하는 방법을 알 수 있도록 별칭을 통해 로컬 뷰를 작성하십시오.

세부 인덱스 통계

DETAILED 옵션을 사용하는 인덱스의 runstats 조작은 버퍼 풀 크기에 따라 옵티마이저가 필요한 데이터 페이지 패치 수를 추정할 수 있도록 통계 정보를 수집합니다. 이 추가 정보는 옵티마이저가 인덱스를 통해 테이블에 액세스하는 비용을 적절하게 추정하는 데 유용하게 사용됩니다.

세부 통계는 다른 버퍼 풀 *크기*에서 전체 인덱스 스캔을 수행하는 경우 테이블의 데이터 페이지에 액세스하는 데 필요한 실제 입출력 수에 대한 간단한 정보를 제공합니다. runstats 유틸리티가 인덱스 페이지를 스캔할 때 다른 버퍼 *크기*를 모델링하고 페이지 결함이 발생하는 빈도를 추정합니다. 예를 들어, 하나의 버퍼 페이지만 사용 가능한 경우 인덱스에서 참조하는 각각의 새 페이지로 인해 페이지 결함이 발생합니다. 최악의 경우 각 행이 다른 페이지를 참조하여 인덱스화된 테이블의 행 수와 동일한 입출력 수가 발생할 수도 있습니다. 또는 다른 극단적인 상황으로 버퍼 *크기*가 전체 테이블을 보유할 수 있을 정도로 큰 경우(최대 버퍼 *크기*에 따라 다름) 모든 테이블 페이지를 한 번에 읽습니다. 따라서 실제 입출력 수는 버퍼 *크기*의 증가하지 않는 단순한 기능입니다.

통계 정보는 인덱스 순서에 대한 테이블 행 클러스터링 수준의 세부 추정도 제공합니다. 클러스터링이 적게 발생하면 인덱스를 통해 테이블 행에 액세스하는 데 필요한 입출력이 증가합니다. 옵티마이저는 인덱스를 통해 테이블에 액세스하는 비용을 추정할 때 버퍼 *크기* 및 클러스터링 수준을 모두 고려합니다.

다음과 같은 경우에 세부 인덱스 통계를 수집하십시오.

- 쿼리가 인덱스에 없는 컬럼을 참조하는 경우
- 테이블에 클러스터링 수준이 다양한 여러 개의 비클러스터 인덱스가 있는 경우
- 키 값들 간 클러스터링 수준이 균일하지 않은 경우
- 균일하지 않은 방식으로 인덱스 값이 갱신된 경우

사전 지식이 없거나 다양한 버퍼 *크기*에서 인덱스 스캔을 강제 실행한 후의 실제 입출력 결과를 모니터하지 않으면 이러한 조건을 식별하기 어렵습니다. 이러한 조건의 존재 여부를 판별하는 가장 저렴한 방법은 인덱스에 대한 세부 통계를 수집 및 조사하고 결과 PAGE_FETCH_PAIRS가 비선형이면 보유하는 것입니다.

세부 인덱스 통계를 수집하는 경우 runstats 조작을 완료하는 데 시간이 오래 걸리고 메모리 및 처리 시간이 추가로 필요합니다. 예를 들어, SAMPLED DETAILED 옵션은 2MB의 통계 힙이 필요합니다. 이 메모리 요구사항의 **stat_heap_sz** 데이터베이스 구성 매개변수 설정에 추가로 488 4KB 페이지를 할당하십시오. 힙이 너무 작은 경우 runstats 유틸리티는 통계를 수집하기 전에 오류를 리턴합니다.

테이블의 *크기*가 충분(약 25페이지 이상)하지 않으면 CLUSTERFACTOR 및 PAGE_FETCH_PAIRS를 수집하지 않습니다. 이 경우 CLUSTERFACTOR는 0 - 1의 값이고 CLUSTERRATIO는 -1(수집하지 않음)입니다. 테이블이 상대적으로 작은 경우 runstats 유틸리티는 값이 0 - 100인 CLUSTERRATIO만 수집합니다.
CLUSTERFACTOR 및 PAGE_FETCH_PAIRS는 수집하지 않습니다. DETAILED절이 지정되지 않은 경우 CLUSTERRATIO만 수집합니다.

인덱스 통계 수집:

인덱스 통계를 수집하여 옵티마이저에서 특정 인덱스를 사용하여 쿼리를 해석해야 하는지 여부를 결정하도록 돕습니다.

다음 예는 인덱스 CUSTIDX1 및 CUSTIDX2와 함께 CUSTOMERS 테이블을 포함하는 SALES라는 이름의 데이터베이스를 기반으로 합니다.

runstats 유틸리티를 사용하는 데 필요한 특권 및 권한에 대해서는 RUNSTATS 명령의 설명을 참조하십시오.

인덱스에 대한 자세한 통계를 수집하려면 다음을 수행하십시오.

1. SALES 데이터베이스에 연결하십시오.
2. 사용자의 요구사항에 따라 DB2 명령행에서 다음 명령 중 하나를 실행하십시오.
   - CUSTIDX1 및 CUSTIDX2 둘 다에 대한 자세한 통계를 수집하려면 다음을 수행하십시오.

     ```
     runstats on table sales.customers and detailed indexes all
     ```

   - 두 인덱스에 대한 자세한 통계를 수집하되, 각 인덱스 항목에 대한 자세한 계산 대신 샘플링을 사용하려면 다음을 수행하십시오.

     ```
     runstats on table sales.customers and sampled detailed indexes all
     ```

     SAMPLED DETAILED 옵션은 2MB의 통계 힙을 필요로 합니다. 이 메모리 요구사항의 **stat_heap_sz** 데이터베이스 구성 매개변수 설정에 추가로 488 4KB 페이지를 할당하십시오. 힙이 너무 작은 경우 runstats 유틸리티는 통계를 수집하기 전에 오류를 리턴합니다.

   - 인덱스와 테이블 통계가 일관되도록 테이블에 대한 분산 통계뿐 아니라 샘플링된 인덱스에 대한 자세한 통계를 수집하려면 다음을 수행하십시오.

     ```
     runstats on table sales.customers
       with distribution on key columns
         and sampled detailed indexes all
     ```

인덱스 통계 수동 갱신을 위한 규칙:

SYSSTAT.INDEXES 카탈로그 뷰에서 통계를 갱신할 때 따라야 하는 특정 지침이 있습니다.

- 다음 규칙은 PAGE_FETCH_PAIRS에 적용됩니다.
  - PAGE_FETCH_PAIRS 통계의 개별 값은 10자리를 초과할 수 없으며 최대 정수 값(2 147 483 647)보다 작아야 합니다.
  - PAGE_FETCH_PAIRS 통계의 개별 값은 공백 문자로 구분해야 합니다.
  - CLUSTERFACTOR가 0보다 큰 경우 항상 유효한 PAGE_FETCH_PAIRS 통계가 있어야 합니다.
  - 단일 PAGE_FETCH_PAIRS 통계에 정확히 11쌍이 있어야 합니다.

- PAGE_FETCH_PAIRS 통계의 버퍼 *크기* 값(각 쌍의 첫 번째 값)은 오름차순으로 표시되어야 합니다.
- PAGE_FETCH_PAIRS 통계의 버퍼 *크기* 값은 32비트 운영 체제의 경우 MIN(NPAGES, 524 287)보다 클 수 없으며 64비트 운영 체제의 경우 MIN(NPAGES, 2 147 483 647)보다 클 수 없습니다. 여기서 NPAGES(SYSSTAT.TABLES에 저장됨)는 해당 테이블의 페이지 수입니다.
- PAGE_FETCH_PAIRS 통계의 페이지 페치 값(각 쌍의 두 번째 값)은 내림차순으로 표시되어야 하며 개별 값은 NPAGES보다 작거나 해당 테이블의 CARD보다 클 수 없습니다.
- 두 개의 연속 쌍에서 버퍼 *크기* 값이 동일한 경우 두 쌍의 페이지 페치 값도 동일해야 합니다.

유효한 PAGE_FETCH_PAIRS 통계 예는 다음과 같습니다.

```
PAGE_FETCH_PAIRS =
  '100 380 120 360 140 340 160 330 180 320 200 310 220 305 240 300
   260 300 280 300 300 300'
```

각 항목은 다음과 같습니다.

```
NPAGES = 300
CARD = 10000
CLUSTERRATIO = -1
CLUSTERFACTOR = 0.9
```

- 다음 규칙이 CLUSTERRATIO 및 CLUSTERFACTOR에 적용됩니다.
  - CLUSTERRATIO의 유효한 값은 -1이거나 0 - 100의 값입니다.
  - CLUSTERFACTOR의 유효한 값은 -1이거나 0 - 1의 값입니다.
  - CLUSTERRATIO 및 CLUSTERFACTOR 값 중 최소한 하나는 항상 -1이어야 합니다.
  - CLUSTERFACTOR가 양수인 경우 유효한 PAGE_FETCH_PAIRS 값이 동반되어야 합니다.
- 관계형 인덱스의 경우 FIRSTKEYCARD, FIRST2KEYCARD, FIRST3KEYCARD, FIRST4KEYCARD, FULLKEYCARD 및 INDCARD에 다음 규칙이 적용됩니다.
  - 단일 컬럼 인덱스의 경우 FIRSTKEYCARD는 FULLKEYCARD와 동일해야 합니다.
  - FIRSTKEYCARD는 해당 컬럼의 SYSSTAT.COLUMNS.COLCARD와 동일해야 합니다.
  - 이러한 인덱스 통계가 관련이 없는 경우 -1로 설정하십시오. 예를 들어, 세 개의 컬럼만 있는 인덱스가 있는 경우 FIRST4KEYCARD를 -1로 설정하십시오.
  - 복수 컬럼 인덱스의 경우 모든 통계가 관련이 있으면 통계들 간 관계는 다음과 같아야 합니다.

```
FIRSTKEYCARD <= FIRST2KEYCARD <= FIRST3KEYCARD <= FIRST4KEYCARD
  <= FULLKEYCARD <= INDCARD == CARD
```

- XML 데이터를 통한 인덱스의 경우 FIRSTKEYCARD, FIRST2KEYCARD, FIRST3KEYCARD, FIRST4KEYCARD, FULLKEYCARD 및 INDCARD 간 관계는 다음과 같아야 합니다.

  ```
  FIRSTKEYCARD <= FIRST2KEYCARD <= FIRST3KEYCARD <= FIRST4KEYCARD
    <= FULLKEYCARD <= INDCARD
  ```

- 다음 규칙은 SEQUENTIAL_PAGES 및 DENSITY에 적용됩니다.
  - SEQUENTIAL_PAGES의 유효한 값은 -1이거나 0 - NLEAF의 값입니다.
  - DENSITY의 유효한 값은 -1이거나 0 - 100의 값입니다.

분산 통계

두 가지 유형의 데이터 분산 통계 즉, 자주 사용되는 값 통계와 Quantile 통계를 수집할 수 있습니다.

- *자주 사용되는 값 통계*는 컬럼 및 중복 수가 가장 큰 데이터 값, 중복 수가 두 번째로 높은 값 등 **num_freqvalues** 데이터베이스 구성 매개변수 값이 지정한 레벨까지 정보를 제공합니다. 자주 사용되는 값 통계 수집을 사용하지 않으려면 **num_freqvalues**를 0으로 설정하십시오. 특정 테이블, 통계 뷰 또는 컬럼의 경우 RUNSTATS 명령에서 NUM_FREQVALUES절을 사용할 수도 있습니다.
- *Quantile 통계*는 데이터 값을 다른 값과 관련하여 분산시키는 방법에 대한 정보를 제공합니다. *K*-quantile이라고 하는 이러한 통계는 최소한 *K* 값 이하의 *V* 값을 나타냅니다. 값을 오름차순으로 정렬하여 *K*-quantile을 계산할 수 있습니다. *K*-quantile 값은 범위의 최소값에서 *K* 번째 자리의 값입니다.

  컬럼 데이터 값을 그룹화해야 하는『섹션』(Quantile) 수를 지정하려면 **num_quantiles** 데이터베이스 구성 매개변수를 2 - 32 767의 값으로 설정하십시오. 디폴트값은 20이며 등호, 미만 또는 초과 술어의 경우 플러스 마이너스 2.5%의 최대 옵티마이저 추정 오차와 BETWEEN 술어의 경우 플러스 마이너스 5%의 최대 오차가 가능합니다. Quantile 통계 수집을 사용하지 않으려면 **num_quantiles**를 0 또는 1로 설정하십시오.

  특정 테이블, 통계 뷰 또는 컬럼의 경우 **num_quantiles**를 설정할 수 있습니다.

주: 큰 **num_freqvalues** 및 **num_quantiles** 값이 사용되는 경우 runstats 유틸리티는 추가 처리 자원 및 메모리(**stat_heap_sz** 데이터베이스 구성 매개변수가 지정함)를 사용합니다.

분산 통계 수집 시기

테이블 또는 통계 뷰의 분산 통계가 유용할지 여부를 결정하려면 먼저 다음을 판별하십시오.

- 응용프로그램의 쿼리가 호스트 변수를 사용하는지 여부.

  분산 통계는 호스트 변수를 사용하지 않는 동적 및 정적 쿼리의 경우에 가장 유용합니다. 옵티마이저는 호스트 변수가 있는 쿼리를 평가할 때 분산 통계를 제한적으로 사용할 수 있도록 합니다.
- 컬럼의 데이터가 균일하게 분산되는지 여부.

  테이블에서 최소한 하나의 컬럼에 매우 『균일하지 않은』 데이터 분산이 있으며 이 컬럼이 등호 또는 범위 술어 즉, 다음과 같은 절에 자주 나타나는 경우 분산 통계를 작성하십시오.

  ```
  where c1 = key;
  where c1 in (key1, key2, key3);
  where (c1 = key1) or (c1 = key2) or (c1 = key3);
  where c1 <= key;
  where c1 between key1 and key2;
  ```

두 가지 유형의 균일하지 않은 데이터 분산이 발생할 수 있으며 동시에 발생할 수도 있습니다.

- 데이터는 최고와 최저 데이터 값 사이에 균일하게 분산하지 않고 거의 클러스터링될 수 있습니다. 데이터를 (5,10) 범위에서 클러스터링하는 다음 컬럼을 참조하십시오.

  ```
    0.0
    5.1
    6.3
    7.1
    8.2
    8.4
    8.5
    9.1
   93.6
  100.0
  ```

  Quantile 통계는 옵티마이저가 이러한 유형의 데이터 분산을 처리할 때 유용합니다.

  쿼리는 컬럼 데이터가 균일하게 분산되지 않았는지 판별할 때 도움이 될 수 있습니다. 예를 들어, 다음과 같습니다.

  ```
  select c1, count(*) as occurrences
    from t1
  group by c1
  order by occurrences desc
  ```
- 중복 데이터 값이 자주 발생할 수 있습니다. 다음과 같은 빈도로 데이터가 분산되는 컬럼을 참조하십시오.

표 74. 컬럼의 데이터 값 빈도

데이터 값	빈도
20	5
30	10
40	10

표 74. 컬럼의 데이터 값 빈도 (계속)

데이터 값	빈도
50	25
60	25
70	20
80	5

자주 사용되는 값 및 Quantile 통계는 모두 옵티마이저가 여러 중복 값을 처리하는 데 유용합니다.

인덱스 통계만 수집하는 시기

다음과 같은 경우 인덱스 데이터에만 근거한 통계를 수집할 수도 있습니다.

- runstats 유틸리티가 실행된 후 새 인덱스가 작성되었으며 테이블 데이터에서 통계를 다시 수집하지 않을 경우
- 인덱스의 첫 번째 컬럼에 영향을 주는 데이터를 여러 번 변경한 경우

지정할 통계 정밀도 레벨

num_quantiles 및 **num_freqvalues** 데이터베이스 구성 매개변수를 사용하여 분산 통계 저장 시 사용할 정밀도를 지정하십시오. 테이블 또는 컬럼의 통계를 수집할 때 해당 RUNSTATS 명령 옵션을 사용하여 정밀도를 지정할 수도 있습니다. 이러한 값을 높게 설정할수록 runstats 유틸리티가 분산 통계를 작성 및 갱신할 때 사용하는 정밀도가 높아집니다. 그러나 정밀도가 높으면 runstats 조작을 수행할 때와 카탈로그 테이블에 추가 데이터를 저장할 때 자원이 많이 필요합니다.

대부분의 데이터베이스에서 **num_freqvalues** 데이터베이스 구성 매개변수의 값으로 10 - 100을 지정하십시오. 가장 자주 사용되는 값의 빈도를 비교할 때 자주 사용되는 값 통계를 나머지 값의 빈도가 서로 대략 동일하거나 무시해도 상관없는 방식으로 작성해야 합니다. 이러한 통계는 여러 번 발생하는 데이터 값의 경우에만 수집되므로 데이터베이스 관리 프로그램은 이 수보다 적게 수집할 수 있습니다. Quantile 통계만 수집해야 하는 경우 **num_freqvalues**의 값을 0으로 설정하십시오.

Quantile 수를 지정하려면 **num_quantiles** 데이터베이스 구성 매개변수를 20 - 50의 값으로 설정하십시오.

- 먼저 모든 범위의 쿼리에서 행 수를 추정할 때 퍼센트 *P*로 승인 가능한 최대 오차를 판별하십시오.
- Quantile 수는 BETWEEN 술어의 경우 약 100/*P*와 기타 범위 술어 유형(<, <=, > 또는 >=)의 경우 50/*P*여야 합니다.

예를 들어, 25 Quantile은 최대 추정 오차가 BETWEEN 술어의 경우 4%이고 『>』 술어의 경우 2%입니다. 일반적으로 최소한 10 Quantile을 지정하십시오. 매우 균일하지 않은 데이터의 경우에만 50 이상의 Quantile이 필요합니다. 자주 사용되는 값 통계만

필요한 경우 **num_quantiles**를 0으로 설정하십시오. 이 매개변수를 1로 설정한 경우 전체 값 범위가 한 Quantile에 속하므로 Quantile 통계를 수집하지 않습니다.

옵티마이저의 분산 통계 사용:

옵티마이저는 다른 쿼리 액세스 플랜의 비용을 적절하게 추정할 수 있도록 분산 통계를 사용합니다.

최저값과 최고값 사이의 값 분산에 대한 추가 정보가 없으면 옵티마이저는 데이터 값이 균일하게 분산된 것으로 가정합니다. 데이터 값이 서로 매우 다르거나 범위의 일부분에서 클러스터링되었거나 여러 중복 값이 포함된 경우 옵티마이저는 차선의 액세스 플랜을 선택합니다.

다음 예를 참조하십시오. 가장 저렴한 액세스 플랜을 선택하기 위해 옵티마이저는 등호 또는 범위 술어를 충족시키는 컬럼 값이 있는 행 수를 추정해야 합니다. 추정이 정확할수록 옵티마이저가 최적의 액세스 플랜을 선택할 가능성이 높아집니다. 다음 쿼리를 참조하십시오.

```
select c1, c2
  from table1
  where c1 = 'NEW YORK'
  and c2 <= 10
```

C1 및 C2 컬럼에 모두 인덱스가 있다고 가정하십시오. 사용할 수 있는 하나의 액세스 플랜은 C1의 인덱스를 사용하여 `C1 = 'NEW YORK'`인 모든 행을 검색한 후 검색한 각 행에 대해 `C2 <= 10`인지 점검하는 것입니다. 대체 플랜은 C2의 인덱스를 사용하여 `C2 <= 10`인 모든 행을 검색한 후 검색한 각 행에 대해 `C1 = 'NEW YORK'`인지 점검하는 것입니다. 쿼리를 실행하는 기본 비용은 보통 행을 검색하는 비용이므로 최상의 플랜은 검색이 가장 적게 필요한 플랜입니다. 이 플랜을 선택한 경우 각 술어를 충족시키는 행 수를 추정해야 합니다.

분산 통계를 사용할 수 없지만 테이블 또는 통계 뷰에서 runstats 유틸리티가 사용된 경우 컬럼에서 두 번째로 높은 데이터 값(HIGH2KEY), 두 번째로 낮은 데이터 값(LOW2KEY), 구별 값 수(COLCARD) 및 행 수(CARD)만 옵티마이저가 사용할 수 있는 정보입니다. 컬럼의 데이터 값에 동일한 빈도가 있으며 LOW2KEY와 HIGH2KEY 사이에 데이터 값을 균일하게 분산시킨다는 가정 하에 등호 또는 범위 술어를 충족시키는 행 수를 추정합니다. 특히 등호 술어(`C1 = KEY`)를 충족시키는 행 수는 CARD/COLCARD로 추정하고 범위 술어를 충족시키는 행 수(`C1 BETWEEN KEY1 AND KEY2`)는 다음과 같이 추정합니다.

```
   KEY2 - KEY1
------------------  x CARD
HIGH2KEY - LOW2KEY
```

이러한 추정은 컬럼에서 데이터 값의 실제 분산이 적당하게 균일한 경우에만 정확합니다. 분산 통계가 사용 불가능하며 데이터 값 빈도가 매우 다양하거나 데이터 값이 균일하지 않게 분산된 경우 크기 정도에 따라 추정을 종료할 수 있으며 옵티마이저는 차선의 액세스 플랜을 선택할 수 있습니다.

분산 통계가 사용 가능한 경우 자주 사용되는 값 통계를 사용하여 등호 술어를 충족시키는 행 수를 추정하고 자주 사용되는 값 통계와 Quantile 통계를 모두 사용하여 범위 술어를 충족시키는 행 수를 추정하면 이러한 오류의 발생 가능성을 상당히 줄일 수 있습니다.

특정 컬럼에 대한 분산 통계 수집:

효과적인 runstats 조작과 후속 쿼리-플랜 분석을 위해, 쿼리가 WHERE, GROUP BY 및 유사한 절에서 참조하는 해당 컬럼에서만 분산 통계를 수집하십시오. 또한 조합된 컬럼 그룹에서 카디널리티(cardinality) 통계를 수집할 수도 있습니다. 옵티마이저는 이러한 정보를 사용하여 그룹의 컬럼을 참조하는 쿼리에 대한 선택 빈도를 추정할 때 컬럼 상관을 감지합니다.

다음 예는 인덱스 CUSTIDX1 및 CUSTIDX2와 함께 CUSTOMERS 테이블을 포함하는 SALES라는 이름의 데이터베이스를 기반으로 합니다.

runstats 유틸리티를 사용하는 데 필요한 특권 및 권한에 대해서는 RUNSTATS 명령의 설명을 참조하십시오.

파티션된 데이터베이스 환경에서 테이블의 통계를 수집할 때 runstats는 유틸리티를 실행하는 데이터베이스 파티션에서만 작동합니다. 이 데이터베이스 파티션의 결과는 다른 데이터베이스 파티션으로 외삽(extrapolate)됩니다. 이 데이터베이스 파티션이 테이블의 필수 분할 영역을 포함하지 않을 경우, 필요한 데이터를 포함하는 데이터베이스 파티션 그룹의 첫 번째 데이터베이스 파티션으로 요청이 보내집니다.

특정 컬럼에 대한 통계를 수집하려면 다음을 수행하십시오.

1. SALES 데이터베이스에 연결하십시오.
2. 사용자의 요구사항에 따라 DB2 명령행에서 다음 명령 중 하나를 실행하십시오.
   - 컬럼 ZIP 및 YTDTOTAL에서 분산 통계를 수집하려면 다음을 실행하십시오.

     ```
     runstats on table sales.customers
       with distribution on columns (zip, ytdtotal)
     ```

   - 동일한 컬럼에서 분산 통계를 수집하되 다른 분산 옵션을 사용하려면 다음을 실행하십시오.

     ```
     runstats on table sales.customers
       with distribution on columns (
         zip, ytdtotal num_freqvalues 50 num_quantiles 75)
     ```

- CUSTIDX1 및 CUSTIDX2에서 인덱스화된 컬럼에서 분산 통계를 수집하려면 다음을 실행하십시오.

```
runstats on table sales.customer
  on key columns
```

- REGION 및 TERRITORY를 포함하는 컬럼 그룹과 컬럼 ZIP 및 YTDTOTAL에 대한 통계를 수집하려면 다음을 실행하십시오.

```
runstats on table sales.customers
  on columns (zip, (region, territory), ytdtotal)
```

- 비 XML 컬럼에 대한 통계가 STATISTICS 옵션과 함께 LOAD 명령을 사용하여 이전에 수집되었다고 가정해봅니다. XML 컬럼 MISCINFO에 대한 통계를 수집하려면 다음을 실행하십시오.

```
runstats on table sales.customers
  on columns (miscinfo)
```

- 비 XML 컬럼에 대한 통계만 수집하려면 다음을 실행하십시오.

```
runstats on table sales.customers
  excluding xml columns
```

  EXCLUDING XML COLUMNS 절은 XML 컬럼을 지정하는 다른 모든 절보다 우선합니다.

분산 통계 사용의 확장 예:

분산 통계는 데이터가 균일하게 분산되지 않았으며 중복이 많은 경우 옵티마이저가 쿼리 액세스 플랜을 빌드하는 데 유용한 테이블 데이터의 빈도 및 분산에 대한 정보를 제공합니다.

다음 예는 옵티마이저가 분산 통계를 사용하는 방식을 이해하는 데 도움이 됩니다.

자주 사용되는 값 통계 예

C1 = KEY 형식의 등호 술어가 있는 쿼리를 참조하십시오. 자주 사용되는 값 통계가 사용 가능한 경우 옵티마이저는 다음과 같이 이러한 통계를 사용하여 적절한 액세스 플랜을 선택할 수 있습니다.

- KEY가 *N*개의 가장 자주 사용되는 값 중 하나인 경우 옵티마이저는 카탈로그에 저장된 KEY의 빈도를 사용합니다.
- KEY가 *N*개의 가장 자주 사용되는 값 중 하나가 아닌 경우 옵티마이저는(COLCARD - *N*) 자주 사용되지 않는 값에 균일한 분산이 있다는 가정 하에 술어를 충족시키는 행 수를 추정합니다. 즉, 행 수는 다음 공식 (1)을 사용하여 추정합니다.

```
CARD - NUM_FREQ_ROWS
--------------------
    COLCARD - N
```

여기서 CARD는 테이블의 행 수이고 COLCARD는 컬럼의 카디널리티(cardinality)이며 NUM_FREQ_ROWS는 *N*의 값 중 가장 자주 사용되는 값과 동일한 값을 가지고 있는 전체 행 수 입니다.

예를 들어, 값이 다음의 빈도를 나타내는 C1 컬럼을 참조하십시오.

데이터 값	빈도
1	2
2	3
3	40
4	4
5	1

테이블의 행 수는 50이고 컬럼 카디널리티(cardinality)는 5입니다. 정확히 40행이 C1 = 3 술어를 충족시킵니다. 데이터가 균일하게 분산된 것으로 가정하면 옵티마이저는 오차가 -75%이며 50/5 = 10으로 술어를 충족시키는 행 수를 추정합니다. 그러나 가장 자주 사용되는 값에만 근거한 자주 사용되는 값 통계(즉, *N* = 1)가 사용 가능한 경우 행 수는 오차 없이 40으로 추정됩니다.

두 개의 행이 C1 = 1 술어를 충족시키는 다른 예를 참조하십시오. 자주 사용되는 값 통계를 사용하지 않으면 술어를 충족시키는 행 수는 10으로 추정되며 오차는 400%입니다.

```
계산된 행 수 - 실제 행 수
--------------------------  X 100
        실제 행 수

10 - 2
------  X 100 = 400%
   2
```

옵티마이저는 자주 사용되는 값 통계(*N* = 1)를 사용하여 위에서 다음과 같이 제공된 공식 (1)로 이 값을 포함한 행 수를 추정합니다.

```
(50 - 40)
--------- = 3
 (5 - 1)
```

오차는 크기 정도에 따라 줄어듭니다.

```
 3 - 2
 ----- = 50%
   2
```

Quantile 통계 예

다음 Quantile 통계 설명에서는 『K-quantile』이라는 용어를 사용합니다. 컬럼의 *K-quantile*은 가장 작은 데이터 값 *V*이며 최소한 *K*행에 *V* 이하의 데이터 값이 있습니

다. K-quantile을 계산하려면 컬럼 값을 오름차순으로 정렬하십시오. *K*-quantile은 정렬된 컬럼에서 *K* 번째 행의 데이터 값입니다.

Quantile 통계가 사용 가능한 경우 옵티마이저는 범위 술어를 충족시키는 행 수를 적절하게 추정할 수 있습니다. 다음 예를 참조하십시오. 다음 값이 있는 C1 컬럼을 참조하십시오.

```
  0.0
  5.1
  6.3
  7.1
  8.2
  8.4
  8.5
  9.1
 93.6
100.0
```

다음과 같이 *K* = 1, 4, 7 및 10의 경우에 *K*-quantile을 사용할 수 있다고 가정하십시오.

K	***K*-quantile**
1	0.0
4	7.1
7	8.5
10	100.0

- 정확히 7행이 `C <= 8.5` 술어를 충족시킵니다. 균일한 데이터 분산을 가정할 때 다음 공식 (2)는

  ```
     KEY2 - KEY1
  ------------------  X CARD
  HIGH2KEY - LOW2KEY
  ```

  KEY1 대신 LOW2KEY와 함께 사용하면 다음과 같이 술어를 충족시키는 행 수를 추정합니다.

  ```
   8.5 - 5.1
  ----------  X 10 ≈ 0
  93.6 - 5.1
  ```

  여기서 ≈는 『대략 같음』을 의미합니다. 이 추정의 오차는 약 -100%입니다.

  Quantile 통계가 사용 가능한 경우 옵티마이저는 8.5(Quantile 중 하나의 최고값)에 해당하는 *K*의 값으로 이 술어를 충족시키는 행 수 7을 추정합니다. 이 경우 오차는 0으로 줄어듭니다.
- 정확히 8행이 `C <= 10` 술어를 충족시킵니다. 옵티마이저가 균일한 데이터 분산을 가정하고 공식 (2)를 사용하는 경우 술어를 충족시키는 행 수는 1로 추정하고 오차는 -87.5%입니다.

이전 예와 달리 값 10은 저장된 *K*-quantile 중 하나가 아닙니다. 그러나 옵티마이저는 Quantile을 사용하여 r_1 + r_2로 술어를 충족시키는 행 수를 추정합니다. 여기서 r_1은 C <= 8.5 술어를 충족시키는 행 수이고 r_2는 C > 8.5 AND C <= 10 술어를 충족시키는 행 수입니다. 위의 예에 따르면 r_1 = 7입니다. r_2를 추정하기 위해 옵티마이저는 선형 보간법을 사용합니다.

```
       10 - 8.5
r_2 ≈ ----------  X (값이 > 8.5이고 <= 100.0인 행 수)
       100 - 8.5

       10 - 8.5
r_2 ≈ ----------  X (10 - 7)
       100 - 8.5

       1.5
r_2 ≈ -----   X (3)
       91.5

r_2 ≈ 0
```

최종 추정은 r_1 + r_2 ≈ 7이고 오차는 겨우 -12.5%입니다.

실제 데이터 값이 5 - 10의 범위에서 『클러스터링』되므로 Quantile은 이러한 예에서 추정의 정확도를 향상시키지만 표준 추정 공식은 데이터 값이 0 - 100에 균일하게 분산되는 것으로 가정합니다.

여러 데이터 값의 빈도가 상당한 차이가 있을 때에도 Quantile을 사용하면 정확성이 개선됩니다. 다음 빈도의 데이터 값이 있는 컬럼을 참조하십시오.

데이터 값	빈도
20	5
30	5
40	15
50	50
60	15
70	5
80	5

K = 5, 25, 75, 95 및 100의 경우에 *K*-quantile을 사용할 수 있다고 가정하십시오.

K	***K*-quantile**
5	20
25	40
75	50
95	70
100	80

또한 세 개의 자주 사용되는 값에 따라 자주 사용되는 값 통계가 사용 가능하다고 가정하십시오.

정확히 10행이 C BETWEEN 20 AND 30 술어를 충족시킵니다. 균일한 데이터 분산을 가정하고 공식 (2)를 사용할 때 술어를 충족시키는 행 수는 다음과 같이 추정됩니다.

```
30 - 20
-------  X 100 = 25
70 - 30
```

오차는 150%입니다.

자주 사용되는 값 통계 및 Quantile 통계를 사용하는 경우 술어를 충족시키는 행 수는 r_1 + r_2로 추정됩니다. 여기서 r_1은 (C = 20) 술어를 충족시키는 행 수이고 r_2는 C > 20 AND C <= 30 술어를 충족시키는 행 수입니다. 공식 (1)을 사용하면 r_1은 다음과 같이 추정됩니다.

```
100 - 80
-------- = 5
  7 - 3
```

선형 보간법을 사용하면 r_2는 다음과 같이 추정됩니다.

```
  30 - 20
  -------  X (값이 > 20이고 <= 40인 행 수)
  40 - 20

  30 - 20
= -------  X (25 - 5)
  40 - 20

= 10
```

최종 추정 15가 산출되고 오차는 3배로 줄어듭니다.

분산 통계 수동 갱신을 위한 규칙:

SYSSTAT.COLDIST 카탈로그 뷰에서 통계를 갱신할 때 따라야 하는 특정 지침이 있습니다.

- 자주 사용되는 값 통계:
  - SEQNO의 값이 증가하는 경우 VALCOUNT 값은 변경되지 않거나 감소해야 합니다.
  - COLVALUE 값의 수는 SYSSTAT.COLUMNS.COLCARD에 저장된 컬럼의 구별 값 수 이하여야 합니다.
  - VALCOUNT에 있는 값의 합계는 SYSSTAT.TABLES.CARD에 저장된 컬럼의 행 수 이하여야 합니다.
  - 대부분의 경우 COLVALUE 값은 각각 SYSSTAT.COLUMNS의 HIGH2KEY 및 LOW2KEY에 저장된 컬럼의 두 번째 최고 데이터 값과 두 번째 최저 데이터

값 사이에 속해야 합니다. HIGH2KEY보다 큰 하나의 자주 사용되는 값과 LOW2KEY보다 작은 하나의 자주 사용되는 값이 있을 수 있습니다.

- Quantile 통계:
  - COLVALUE 값은 SEQNO의 값이 증가하는 경우 변경되지 않거나 감소해야 합니다.
  - VALCOUNT 값은 SEQNO의 값이 증가하는 경우 증가해야 합니다.
  - 최대 COLVALUE 값은 VALCOUNT에서 컬럼의 행 수와 동일한 해당 항목이 있어야 합니다.
  - 대부분의 경우 COLVALUE 값은 각각 SYSSTAT.COLUMNS의 HIGH2KEY 및 LOW2KEY에 저장된 컬럼의 두 번째 최고 데이터 값과 두 번째 최저 데이터 값 사이에 속해야 합니다.

*R*행의 C1 컬럼에 분산 통계를 사용할 수 있으며 동일한 상대적 비율의 데이터 값이 있지만 (*F* x *R*)행을 사용하는 컬럼과 일치하도록 통계를 수정한다고 가정하십시오. 자주 사용되는 값 또는 Quantile 통계의 범위를 *F*배씩 확장하려면 각 VALCOUNT 항목에 *F*를 곱하십시오.

사용자 정의 함수(UDF) 통계

사용자 정의 함수(UDF)에 대한 통계 정보를 작성하려면 SYSSTAT.ROUTINES 카탈로그 뷰를 편집하십시오.

runstats 유틸리티는 UDF에 대한 통계를 수집하지 않습니다. UDF 통계가 사용 가능한 경우 옵티마이저는 다양한 액세스 플랜의 비용을 추정할 때 이 통계를 사용할 수 있습니다. 통계가 사용 불가능한 경우 옵티마이저는 간단한 UDF를 가정하는 디폴트값을 사용합니다.

표 75에서는 성능을 향상시키기 위한 추정을 제공할 수 있는 카탈로그 뷰 컬럼을 나타냅니다. SYSSTAT.ROUTINES(SYSCAT.ROUTINES가 아님)의 컬럼 값만 사용자가 수정할 수 있다는 점을 참고하십시오.

표 75. 함수 통계(SYSCAT.ROUTINES 및 SYSSTAT.ROUTINES)

통계	설명
IOS_PER_INVOC	함수를 호출할 때마다 실행되는 읽기 또는 쓰기 요청 수 추정
INSTS_PER_INVOC	함수를 호출할 때마다 실행되는 머신 명령어 수 추정
IOS_PER_ARGBYTE	입력 인수 유형마다 실행되는 읽기 또는 쓰기 요청 수 추정
INSTS_PER_ARGBYTE	입력 인수 유형마다 실행되는 머신 명령어 수 추정
PERCENT_ARGBYTES	함수가 실제로 처리할 입력 인수 바이트의 평균 퍼센트 추정

표 75. 함수 통계(SYSCAT.ROUTINES 및 SYSSTAT.ROUTINES) (계속)

통계	설명
INITIAL_IOS	처음 또는 마지막으로 함수를 호출하면 실행되는 읽기 또는 쓰기 요청 수 추정
INITIAL_INSTS	처음 또는 마지막으로 함수를 호출하면 실행되는 머신 명령어 수 추정
CARDINALITY	테이블 함수가 생성하는 행 수 추정

예를 들어, EU_SHOE의 경우 미국 신발 *크기*를 동등한 유럽 신발 *크기*로 변환하는 UDF가 있습니다. 이 UDF에서는 다음과 같이 SYSSTAT.ROUTINES에 통계 컬럼의 값을 설정할 수 있습니다.

- INSTS_PER_INVOC. 다음을 수행하는 데 필요한 머신 명령어 수 추정으로 설정하십시오.
  - EU_SHOE 호출
  - 출력 문자열 *초기화*
  - 결과 리턴
- INSTS_PER_ARGBYTE. 입력 문자열을 유럽 신발 *크기*로 변환하는 데 필요한 머신 명령어 수 추정으로 설정하십시오.
- PERCENT_ARGBYTES. 전체 입력 문자열을 변환하도록 표시하는 100으로 설정하십시오.
- INITIAL_INSTS, IOS_PER_INVOC, IOS_PER_ARGBYTE 및 INITIAL_IOS. 이 UDF는 계산만 수행하므로 각각 0으로 설정됩니다.

PERCENT_ARGBYTES는 항상 전체 입력 문자열을 처리하는 것은 아닌 함수에서 사용합니다. 예를 들어, 두 인수를 입력 받아 두 번째 인수에서 첫 번째 인수의 값이 처음으로 어커런스 된 위치를 리턴하는 UDF인 LOCATE를 참조하십시오. 첫 번째 인수의 길이가 두 번째 인수에 비해 무시해도 될 정도로 충분히 작으며 평균 75%의 두 번째 인수를 검색한다고 가정하십시오. 이 정보와 다음 가정에 따르면
PERCENT_ARGBYTES는 75로 설정해야 합니다.

- 거의 항상 첫 번째 인수를 찾을 수 없으므로 전체 두 번째 인수를 검색합니다.
- 첫 번째 인수는 두 번째 인수의 어디에나 동등하게 표시될 수 있으므로 첫 번째 인수를 찾으면 두 번째 인수의 절반을 검색합니다(평균적으로).

INITIAL_INSTS 또는 INITIAL_IOS를 사용하면 처음 또는 마지막으로 함수를 호출할 때 수행되는 머신 명령어나 읽기 또는 쓰기 요청 수 추정을 기록할 수 있습니다. 이 수는 예를 들어, 스크래치 패드 영역을 설정하는 비용을 나타낼 수 있습니다.

UDF에서 사용하는 입출력 및 명령어에 대한 정보를 보려면 프로그래밍 언어 컴파일러 또는 운영 체제에 사용 가능한 모니터링 도구에서 제공하는 출력을 사용하십시오.

모델링 및 가정 계획의 카탈로그 통계

계획 용도로 시스템 카탈로그에서 특정 통계 정보 변경에 따른 데이터베이스 성능 영향을 관찰할 수 있습니다.

선택된 시스템 카탈로그 통계를 갱신하는 기능을 사용하면 다음의 작업이 가능합니다.

- 프로덕션 시스템 통계를 사용하여 개발 시스템에서 쿼리 성능 모델링
- 『가정』 쿼리 성능 분석 수행

프로덕션 시스템에서 통계를 수동으로 갱신하지 마십시오. 그렇지 않으면 옵티마이저가 동적 SQL 또는 XQuery문이 포함된 프로덕션 쿼리에 가장 적합한 액세스 플랜을 선택할 수 없습니다.

테이블 및 인덱스와 해당 구성요소에 대한 통계를 수정하려면 데이터베이스에 대한 명시적 DBADM 권한이 있어야 합니다. DATAACCESS 권한을 보유하는 사용자는 SYSSTAT 스키마에 정의된 뷰에 대해 UPDATE문을 실행하여 이러한 통계 컬럼의 값을 변경할 수 있습니다.

DATAACCESS 권한이 없는 사용자는 CONTROL 특권을 가지고 있는 오브젝트에 대한 통계가 들어 있는 행만 볼 수 있습니다. DATAACCESS 권한이 없는 경우 각 데이터베이스 오브젝트에 대해 다음 특권이 있으면 개별 데이터베이스 오브젝트에 대한 통계를 변경할 수 있습니다.

- 테이블에 대한 명시적 CONTROL 특권. 이러한 테이블에 대한 컬럼 및 인덱스의 통계를 갱신할 수도 있습니다.
- 페더레이티드 데이터베이스 시스템의 별칭에 대한 명시적 CONTROL 특권. 이 별칭에 대한 컬럼 및 인덱스의 통계를 갱신할 수도 있습니다. 이러한 갱신은 로컬 메타데이터에만 영향을 주며(데이터 소스 테이블 통계는 변경되지 않음) DB2 옵티마이저에서 생성한 전역 액세스 전략에만 영향을 줍니다.
- 사용자 정의 함수(UDF)의 소유권

다음 코드는 EMPLOYEE 테이블의 통계 갱신 예입니다.

```
update sysstat.tables
  set
    card = 10000,
    npages = 1000,
    fpages = 1000,
    overflow = 2
  where tabschema = 'MELNYK'
    and tabname = 'EMPLOYEE'
```

카탈로그 통계를 수동으로 갱신할 때에는 주의해야 합니다. 임의 변경은 연속 쿼리 성능을 심각하게 저하할 수 있습니다. 다음 방법을 사용하여 개발 시스템에 대한 통계를 일관성 있는 상태로 되돌릴 수 있습니다.

- 수동으로 변경한 작업 단위를 롤백하십시오(작업 단위가 아직 커미트되지 않은 경우).

- runstats 유틸리티를 사용하여 카탈로그 통계를 새로 고치십시오.
- 카탈로그 통계를 갱신하여 통계가 수집되지 않았음을 지정하십시오. 예를 들어, NPAGES 컬럼 값을 -1로 설정하면 이 통계가 수집되지 않았음을 표시합니다.
- 변경한 사항을 실행 취소하십시오. 이 방법은 변경하기 전에 db2look 명령을 사용하여 통계를 캡처한 경우에만 사용할 수 있습니다.

일부 값 또는 값 조합이 유효하지 않은 것으로 판별된 경우 옵티마이저는 디폴트값을 사용하고 경고를 리턴합니다. 그러나 대부분의 유효성 확인은 통계가 갱신될 때 수행되므로 이러한 상황은 거의 발생하지 않습니다.

모델링 프로덕션 데이터베이스에 대한 통계:

개발 시스템에 프로덕션 시스템의 데이터 서브세트를 포함시켜야 하는 경우가 있습니다. 그러나 개발 시스템에서 선택한 액세스 플랜이 프로덕션 시스템에서 선택하는 액세스 플랜과 반드시 동일하지는 않습니다.

개발 시스템의 카탈로그 통계 및 구성을 프로덕션 시스템의 카탈로그 통계 및 구성과 일치하도록 갱신해야 하는 경우가 있습니다.

가상 갱신 모드(-m 옵션 지정)의 db2look 명령을 사용하면 개발 및 프로덕션 데이터베이스의 카탈로그 통계를 일치시키는 데 필요한 데이터 처리 언어(DML) 명령문을 생성할 수 있습니다.

db2look이 개발 시스템에 대해 작성하는 UPDATE문을 실행한 후 이 시스템을 사용하여 프로덕션 시스템에서 생성 중인 액세스 플랜의 유효성을 확인할 수 있습니다. 옵티마이저는 테이블 스페이스의 구성을 사용하여 입출력 비용을 추정하므로 개발 시스템의 테이블 스페이스는 프로덕션 시스템의 테이블 스페이스와 동일한 유형(SMS 또는 DMS)이어야 하며 컨테이너 수가 동일해야 합니다.

카탈로그 통계 수동 갱신 방지

DB2 데이터 서버는 SYSSTAT 스키마의 뷰에 대해 UPDATE문을 실행하여 카탈로그 통계 수동 갱신을 지원합니다.

이 기능은 쿼리 액세스 플랜을 검사하기 위해 테스트 시스템에서 프로덕션 데이터베이스를 가상 갱신할 때 유용할 수 있습니다. db2look 유틸리티는 다른 시스템에서의 재생을 위해 SYSSTAT 스키마의 뷰에 대해 DDL 및 UPDATE문을 캡처하는 데 매우 유용합니다.

특정 쿼리 액세스 플랜을 강제 실행하기 위해 올바르지 않은 통계를 수동으로 제공함으로써 쿼리 옵티마이저에 영향을 주지 않도록 하십시오. 이 실행을 통해 일부 쿼리에 대한 성능이 향상될 수도 있지만, 다른 쿼리에 대한 성능 저하로 이어질 수도 있습니

다. 이 접근 방법에 의존하기 전에 최적화 지침 및 프로파일 사용과 같은 기타 조정 옵션을 고려하십시오. 이 방법이 필요하게 될 경우, 리스토어되어야 할 경우를 생각하여 원래 통계를 반드시 기록하십시오.

runstats 영향 최소화

runstats 성능을 향상시키기 위해 사용할 수 있는 몇 가지 접근 방법이 있습니다.

이 유틸리티가 성능에 미치는 영향을 최소화하려면 다음을 수행하십시오.

- COLUMNS 절을 사용하여 통계가 수집되어야 하는 컬럼을 제한하십시오. 많은 컬럼이 쿼리 워크로드에서 술어에 의해 참조되지 않으므로, 통계를 필요로 하지 않습니다.
- 데이터가 균일하게 분산되는 경향이 있는 경우 분산 통계가 수집되는 컬럼을 제한하십시오. 분산 통계를 수집하려면 기본 컬럼 통계를 수집하는 것보다 더 많은 CPU 및 메모리가 필요합니다. 그러나 컬럼의 값이 균일하게 분산되어 있는지 여부를 판별하려면 기존 통계를 갖거나 데이터를 쿼리해야 합니다. 이 방법에서는 또한 테이블이 수정되면서 데이터가 여전히 균일하게 분산된다는 것을 가정합니다.
- 페이지 또는 행 레벨 샘플링을 사용하여 (TABLESAMPLE SYSTEM 또는 BERNOULLI 절을 지정하여) 처리된 행 및 페이지 수를 제한하십시오. TABLESAMPLE SYSTEM(10)을 지정하여 10% 페이지 레벨 샘플부터 시작하십시오. 통계의 정밀도 및 액세스 플랜의 변경으로 인해 시스템 성능이 저하되었는지 여부를 점검하십시오. 저하된 경우, TABLESAMPLE BERNOULLI(10)를 지정하여 10% 행 레벨 샘플을 대신 시도하십시오. 통계의 정밀도가 충분하지 않은 경우, 샘플링 양을 늘리십시오. RUNSTATS 페이지 또는 행 레벨 샘플링을 사용할 경우, 조인되는 테이블에 대해 동일한 샘플링 비율을 사용하십시오. 이것은 조인 컬럼 통계가 동일한 레벨의 정밀도를 갖도록 보장하기 위해 중요합니다.
- CREATE INDEX문에서 COLLECT STATISTICS 옵션을 지정하여 인덱스 작성 동안 인덱스 통계를 수집하십시오. 이 방법은 인덱스가 작성된 후 별도의 runstats 조작을 수행하는 것보다 더 빠릅니다. 또한 새 인덱스가 작성 직후에 생성된 통계를 갖도록 하여 옵티마이저가 인덱스 사용 비용을 정확하게 추정할 수 있게 해 줍니다.
- REPLACE 옵션과 함께 LOAD 명령을 실행할 때 통계를 수집하십시오. 이 방법은 로드 조작이 완료된 후 별도의 runstats 조작을 수행하는 것보다 더 빠릅니다. 또한 테이블이 데이터가 로드된 직후에 가장 최신 통계를 갖도록 하여 옵티마이저가 테이블 사용 비용을 정확하게 추정할 수 있게 해 줍니다.

파티션된 데이터베이스 환경에서, runstats 유틸리티는 단일 데이터베이스 파티션에서 통계를 수집합니다. RUNSTATS 명령이 테이블이 상주하는 데이터베이스 파티션에서 발행될 경우, 이 곳에서 통계가 수집됩니다. 그렇지 않을 경우, 테이블에 대한 데이터베이스 파티션 그룹의 첫 번째 데이터베이스 파티션에서 통계가 수집됩니다. 일관성 있는 통계를 위해, 조인된 테이블에 대한 통계가 동일한 데이터베이스 파티션에서 수집되도록 하십시오.

데이터 압축 및 성능

데이터 압축을 사용하여 디스크에서 읽거나 디스크에 써야 하는 데이터의 양을 줄일 수 있으며, 이에 따라 입출력 비용을 줄일 수 있습니다.

현재 두 가지 양식의 데이터 압축이 사용 가능합니다.

- *값 압축*은 값의 중복되는 항목을 제거하고, 하나의 사본만 저장하며, 저장된 값에 대한 모든 참조의 위치 추적을 유지하는 것을 뜻합니다.
- *행 압축*은 행 내 복수 컬럼 값에 걸쳐 있는 반복되는 패턴을 더 짧은 기호 문자열로 교체하는 것을 뜻합니다. 행 압축 논리는 반복 및 중복 데이터에 대해 압축될 테이블을 스캔합니다. 압축 사전에는 데이터에 대한 짧은 숫자 키가 포함되고, 압축 행에서 이러한 키가 실제 데이터를 교체합니다.

DB2 버전 9.1 이전에는 오프라인 테이블 재구성을 수행하여 압축 사전을 수동으로 작성해야 했습니다. 버전 9.5 이후에는 데이터 압축에 사용 가능한 테이블에 대해 데이터 압축 사전이 자동으로 작성됩니다.

이 릴리스에서는 버전 9.5에서 영구 테이블에 대해 도입되었던 자율 행 압축이 모든 임시 테이블을 포함하도록 확장되었습니다. 임시 테이블에 대한 데이터 압축:

- 대형 및 복합 쿼리에 필요한 임시 디스크 스페이스의 양을 줄입니다.
- 쿼리 성능을 향상시킵니다.

임시 테이블에 대한 데이터 압축은 DB2 스토리지 최적화 기능에서 자동으로 사용됩니다.

행 압축에 적합한 각 임시 테이블에는 압축 사전을 작성할 2 - 3MB의 메모리가 추가로 필요합니다.

압축된 임시 테이블의 인덱스 및 인덱스 오브젝트 또한 압축하여 스토리지 비용을 줄일 수 있습니다. 이것은 특히 일반적으로 매우 큰 인덱스가 많이 있는 대형 온라인 트랜잭션 처리(OLTP) 및 데이터 웨어하우스 환경에서 유용합니다. 이러한 두 경우에, 인덱스 압축은 입출력 바운드 환경에는 상당한 성능 향상을 가져오고, CPU 바운드 환경에는 성능 감소를 거의 일으키지 않을 수 있습니다.

XML 컬럼이 있는 테이블에서 압축을 사용할 수 있는 경우, XDA 오브젝트에 저장된 XML 데이터도 압축됩니다. XML 데이터에 대한 별도의 압축 사전이 XDA 오브젝트에 저장됩니다. 이는 현재 버전의 DB2 제품에서 XML 컬럼이 추가된 테이블에도 적용됩니다. XDA 압축은 이 버전 이전에 XML 컬럼이 작성된 테이블의 경우에는 지원되지 않습니다. 이러한 테이블의 경우, 데이터 오브젝트만 압축됩니다.

로깅 오버헤드를 축소한 DML 성능 향상

데이터베이스 관리 프로그램은 모든 데이터베이스 변경사항을 기록하는 로그 파일을 유지보수합니다. 두 가지 로깅 전략 즉, 순환 로깅과 아카이브 로깅이 있습니다.

- *순환 로깅*을 사용할 경우 사용 가능한 파일이 가득 차면 로그 파일을 재사용합니다(초기 로그 파일부터 시작). 겹쳐쓴 로그 레코드는 복구할 수 없습니다.
- *아카이브 로깅*을 사용할 경우 로그 레코드로 가득 차면 로그 파일이 아카이브됩니다. 로그 보존을 통해 롤 포워드 복구가 가능합니다. 로그 파일에 기록된 데이터베이스의 변경사항(완료된 작업 단위 또는 트랜잭션)이 재해 복구 중 다시 적용될 수 있습니다.

일반 데이터 및 인덱스 페이지의 모든 변경사항은 로그 프로그램 프로세스에서 디스크에 기록하기 전에 로그 버퍼로 기록됩니다. SQL문 처리는 로그 데이터가 디스크에 기록될 때까지 기다려야 합니다.

- COMMIT할 때
- 해당 데이터 페이지가 디스크에 기록될 때까지. COMMIT문을 사용하여 트랜잭션이 완료되면 변경된 모든 데이터 및 인덱스 페이지를 디스크에 기록할 필요는 없는 미리 쓰기 로깅을 DB2 서버에서 사용합니다.
- 메타데이터가 변경될 때까지(주로 데이터 정의 언어 명령문 실행으로 발생함)
- 로그 버퍼가 가득 찰 때

데이터베이스 관리 프로그램은 이러한 방식으로 디스크에 로그 데이터를 기록하여 처리 대기 시간을 최소화합니다. 간단한 여러 개의 트랜잭션을 동시에 처리 중인 경우 대부분의 대기 시간은 로그 데이터가 디스크에 기록될 때까지 기다려야 하는 COMMIT문으로 인해 발생합니다. 따라서 로그 프로그램 프로세스는 소량의 로그 데이터를 디스크에 자주 기록합니다. 로그 입출력으로 인해 추가 대기 시간이 발생합니다. 응용프로그램 응답 시간을 로깅 대기 시간과 균형을 맞추려면 **mincommit** 데이터베이스 구성 매개변수를 1보다 큰 값으로 설정하십시오. 이 설정으로 일부 응용프로그램에 대한 COMMIT 대기 시간이 길어지지만 한 번의 조작으로 추가 로그 데이터를 기록할 수 있습니다.

쉐도우 페이징을 통해 대형 오브젝트(LOB) 및 LONG VARCHAR의 변경사항을 추적합니다. 로그 보유를 지정하지 않고 LOB 컬럼이 CREATE TABLE문의 NOT LOGGED절 없이 정의되지 않았으면 LOB 컬럼 변경사항이 로그되지 않습니다. LONG 또는 LOB 데이터 유형의 할당 페이지에 대한 변경사항은 일반 데이터 페이지와 같이 로그됩니다. 인라인 LOB 값은 VARCHAR 값이었던 것처럼 전체적으로 갱신, 삽입 또는 삭제 로깅에 참여합니다.

인라인 LOB가 성능을 향상시킴

일부 응용프로그램에서는 대형 오브젝트를(LOB) 광범위하게 사용합니다. 대부분 이러한 LOB는 그리 크지 않으며 대부분 몇 KB의 크기입니다. 이제 이러한 LOB 데이터를 LOB 스토리지 오브젝트 대신 데이터 페이지의 형식화된 행 내에 배치하여 LOB 데이터 액세스 성능을 향상시킬 수 있습니다.

이러한 LOB를 *인라인 LOB*라고 합니다. 이전에는 이러한 LOB의 처리로 응용프로그램의 병목 현상이 생길 수 있었습니다. 이 데이터를 페치, 삽입 또는 갱신하는 데 추가 입출력이 필요하지 않기 때문에 인라인 LOB는 LOB 데이터에 액세스하는 쿼리 성능을 향상시킵니다. 또한 인라인 LOB 데이터는 행 압축에 적합합니다.

이 기능은 ALTER TABLE문 또는 CREATE TABLE문에서 INLINE LENGTH 옵션을 통해 사용됩니다. INLINE LENGTH 옵션은 구조화된 유형, XML 유형 또는 LOB 컬럼에 적용됩니다. LOB 컬럼의 경우, 인라인 길이는 기본 테이블 행에 저장될 수 있는 LOB 값의 최대 바이트 *크기*(오버헤드에 대한 4바이트 포함)를 표시합니다.

또한 새 테이블 또는 기존 테이블의 모든 LOB 컬럼(LOB 컬럼이 추가되는 경우) 및 데이터베이스 업그레이드 시 LOB의 모든 기존 컬럼에 이 기능을 내재적으로 사용할 수 있습니다. 모든 LOB 컬럼에는 정의된 최대 크기를 기반으로 행 스페이스가 예약되어 있습니다. 각 LOB 컬럼에 대해 내재된 INLINE LENGTH 값은 자동으로 정의되며 명시적으로 지정된 것처럼 저장됩니다.

인라인으로 저장할 수 없는 LOB 값은 LOB 스토리지 오브젝트에 별도로 저장됩니다.

테이블에 인라인 LOB가 있는 컬럼이 있는 경우, 페이지에 맞는 행 수가 줄어들며 LOB가 아닌 데이터만 리턴하는 쿼리 성능이 저하될 수 있음에 유의하십시오. LOB 인라인은 대부분의 명령문이 하나 이상의 LOB 컬럼을 포함하는 워크로드에 유용합니다.

LOB 데이터가 반드시 로그되는 것은 아니지만 인라인 LOB는 항상 로그되므로 로깅 오버헤드가 증가할 수 있습니다.

제 4 장 성능 조정 전략 설정

디자인 어드바이저

DB2 디자인 어드바이저는 워크로드 성능을 상당히 향상시킬 수 있는 도구입니다. 복합 워크로드에 대해 작성할 인덱스, 구체화된 쿼리 테이블(MQT), 클러스터링 차원 또는 데이터베이스 파티션을 선택하는 태스크는 복잡해 보입니다. 디자인 어드바이저가 워크로드의 성능을 향상시키는 데 필요한 모든 오브젝트를 식별합니다.

워크로드의 SQL문 세트에 대해 디자인 어드바이저는 다음에 대한 권장사항을 생성합니다.

- 새 인덱스
- 새 클러스터링 인덱스
- 새 MQT
- MDC(다차원 클러스터링) 테이블로 변환
- 테이블 재분산

디자인 어드바이저는 이러한 일부 또는 전체 권장사항을 즉시 구현하거나 나중에 실행하도록 스케줄링할 수 있습니다.

db2advis 명령을 사용하여 디자인 어드바이저 유틸리티를 시작하십시오.

디자인 어드바이저를 사용하면 다음 태스크가 간단해집니다.

새 데이터베이스 계획 및 설정

데이터베이스를 설계할 때 디자인 어드바이저를 사용하여 인덱싱, MQT, MDC 테이블 또는 데이터베이스 파티셔닝에 맞게 테스트 환경에서 설계 대안을 생성하십시오.

파티션된 데이터베이스 환경에서 디자인 어드바이저를 사용하여 다음을 수행할 수 있습니다.

- 데이터베이스에 데이터를 로드하기 전에 적합한 데이터베이스 파티셔닝 전략을 판별할 수 있습니다.
- 단일 파티션 데이터베이스에서 다중 파티션 데이터베이스로 업그레이드를 보조할 수 있습니다.
- 다른 데이터베이스 제품에서 다중 파티션 DB2 데이터베이스로 이주를 보조할 수 있습니다.

워크로드 성능 조정

데이터베이스가 설정된 후 디자인 어드바이저를 사용하여 다음을 수행할 수 있습니다.

- 특정 명령문 또는 워크로드의 성능 향상
- 샘플 워크로드 성능을 게이지로 사용하여 일반 데이터베이스 성능 향상
- 예를 들어, 활동 모니터가 식별한 가장 자주 실행되는 쿼리 성능 향상
- 새 쿼리의 성능을 최적화하는 방법 판별
- 정렬에 집중된 워크로드의 공유 메모리 유틸리티 또는 정렬 힙 문제에 관해 Health Center 권장사항에 응답
- 워크로드에서 사용되지 않는 오브젝트 찾기

디자인 어드바이저 출력

디자인 어드바이저 출력은 디폴트로 표준 출력에 기록되며 ADVISE_* 테이블에 저장됩니다.

- 디자인 어드바이저가 실행될 때마다 ADVISE_INSTANCE 테이블은 새로운 한 행으로 갱신됩니다.
  - START_TIME 및 END_TIME 필드는 유틸리티의 시작 및 중지 시간을 표시합니다.
  - 유틸리티가 올바로 종료된 경우 STATUS 필드에는 COMPLETED 값이 있습니다.
  - MODE 필드는 db2advis 명령의 -m 옵션이 사용되었는지 여부를 표시합니다.
  - COMPRESSION 필드는 사용된 압축 유형을 표시합니다.
- MQT, MDC 테이블 또는 데이터베이스 파티셔닝 전략 권장사항이 충족된 경우 ADVISE_TABLE 테이블의 USE_TABLE 컬럼에는 Y 값이 있습니다.

  MQT 권장사항은 ADVISE_MQT 테이블, MDC 권장사항은 ADVISE_TABLE 테이블, 데이터베이스 파티셔닝 전략 권장사항은 ADVISE_PARTITION 테이블에서 찾을 수 있습니다. 이러한 테이블의 RUN_ID 컬럼에는 ADVISE_INSTANCE 테이블에 있는 행의 START_TIME 값에 해당하는 값이 있으며 이를 동일한 디자인 어드바이저 실행으로 링크합니다.

  MQT, MDC 또는 데이터베이스 파티셔닝 권장사항이 제공된 경우 관련 ALTER TABLE 스토어드 프로시저 호출이 ADVISE_TABLE 테이블의 ALTER_COMMAND 컬럼에 배치됩니다. ALTER TABLE 스토어드 프로시저 호출은 ALTOBJ 스토어드 프로시저의 테이블에 대한 제한사항으로 인해 완료되지 않습니다.

- 권장사항이 충족된 경우 ADVISE_INDEX 테이블의 USE_INDEX 컬럼에는 Y(인덱스 권장 또는 평가) 또는 R(기존 클러스터링 RID 인덱스를 클러스터링 해제하도록 권장됨) 값이 있습니다.
- ADVISE_MQT 테이블의 COLSTATS 컬럼에는 MQT에 대한 컬럼 통계가 있습니다. 이러한 통계는 다음과 같이 XML 구조로 표시됩니다.

```
<?xml version=₩"1.0₩" encoding=₩"USASCII₩"?>
<colstats>
    <column>
         <name>COLNAME1</name>
         <colcard>1000</colcard>
         <high2key>999</high2key>
         <low2key>2</low2key>
    </column>
    ....

    <column>
         <name>COLNAME100</name>
         <colcard>55000</colcard>
         <high2key>49999</high2key>
         <low2key>100</low2key>
    </column>
</colstats>
```

db2advis 명령에서 -o 옵션을 사용하여 디자인 어드바이저 권장사항을 파일에 저장할 수 있습니다. 저장된 디자인 어드바이저 출력은 다음 요소로 구성되어 있습니다.

- 새 인덱스, MQT, MDC 테이블 또는 데이터베이스 파티셔닝 전략과 연관된 CREATE문
- MQT에 대한 REFRESH문
- 새 오브젝트의 RUNSTATS 명령

이러한 출력 예는 다음과 같습니다.

```
--<?xml version="1.0"?>
--<design-advisor>
--<mqt>
--<identifier>
--<name>MQT612152202220000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<statementlist>3</statementlist>
--<benefit>1013562.481682</benefit>
--<overhead>1468328.200000</overhead>
--<diskspace>0.004906</diskspace>
--</mqt>
.....
--<index>
--<identifier>
--<name>IDX612152221400000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<table><identifier>
--<name>PART</name>
```

```
--<schema>TPCD </schema>
--</identifier></table>
--<statementlist>22</statementlist>
--<benefit>820160.000000</benefit>
--<overhead>0.000000</overhead>
--<diskspace>9.063500</diskspace>
--</index>
.....
--<statement>
--<statementnum>11</statementnum>
--<statementtext>
--
-- select
-- c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice,
-- sum(l_quantity) from tpcd.customer, tpcd.orders,
-- tpcd.lineitem where o_orderkey in( select
-- l_orderkey from tpcd.lineitem group by l_orderkey
-- having sum(l_quantity) > 300 ) and c_custkey
-- = o_custkey and o_orderkey = l_orderkey group by
-- c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
-- order by o_totalprice desc, o_orderdate fetch first
-- 100 rows only
--</statementtext>
--<objects>
--<identifier>
--<name>MQT612152202490000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>ORDERS</name>
--<schema>TPCD </schema>
--</identifier>
--<identifier>
--<name>CUSTOMER</name>
--<schema>TPCD </schema>
--</identifier>
--<identifier>
--<name>IDX612152235020000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>IDX612152235030000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>IDX612152211360000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--</objects>
--<benefit>2091459.000000</benefit>
--<frequency>1</frequency>
--</statement>
```

이 XML 구조에는 여러 컬럼이 있을 수 있습니다. 컬럼 카디낼리티(즉, 각 컬럼의 값 수)가 포함되어 있으며 선택적으로 HIGH2KEY 및 LOW2KEY 값입니다.

인덱스가 정의된 기본 테이블도 포함되어 있습니다. benefit 값을 사용하여 인덱스 및 MQT 순위를 지정할 수 있습니다. (benefit - overhead)를 사용하여 인덱스 순위를 지정하고 (benefit - 0.5 * overhead)를 사용하여 MQT 순위를 지정할 수도 있습니다.

인덱스 및 MQT 목록 뒤에는 SQL 텍스트, 명령문의 명령문 번호, 권장사항을 통해 추정되는 성능 향상(이점)과 명령문에서 사용한 테이블, 인덱스 및 MQT 목록이 들어 있는 명령문 목록입니다. SQL 텍스트에서 원래의 스페이스는 이 출력 예에서 보존되지만 쉽게 읽을 수 있도록 SQL 텍스트는 보통 80자의 주석행으로 분할됩니다.

기존 인덱스 또는 MQT가 워크로드를 실행하는 데 사용되는 경우 출력에 표시됩니다.

MDC 및 데이터베이스 파티셔닝 권장사항은 이 XML 출력 예제에 명시적으로 표시되지 않습니다.

일부 사소한 부분을 수정한 후 이 출력 파일을 CLP 스크립트로 실행하면 권장 오브젝트를 작성할 수 있습니다. 수정할 사항은 다음과 같습니다.

- 모든 RUNSTATS 명령을 수정되었거나 새로운 오브젝트에 대한 단일 RUNSTATS 호출로 결합
- 시스템 생성 ID 대신 쉽게 사용할 수 있는 오브젝트 이름 제공
- 즉시 구현하지 않을 오브젝트의 DDL(Data Definition Language) 제거 또는 주석처리

디자인 어드바이저 사용

db2advis 명령을 호출하여 디자인 어드바이저를 실행할 수 있습니다.

1. 워크로드를 정의하십시오. 『디자인 어드바이저에 대한 워크로드 정의』를 참조하십시오.
2. 이 워크로드에 대해 db2advis 명령을 실행하십시오.

   주: 데이터베이스의 통계가 최신이 아닌 경우, 생성된 권장사항의 신뢰성이 다소 떨어집니다.
3. db2advis의 출력을 해석하고 필요한 수정을 수행하십시오.
4. 상황에 맞게 디자인 어드바이저 권장사항을 구현하십시오.

디자인 어드바이저에 대한 워크로드 정의

디자인 어드바이저는 특정 워크로드를 분석할 때 워크로드에 포함된 명령문의 유형, 특정 명령문이 발생하는 빈도 및 워크로드를 실행하는 총 비용을 최소화하는 권장사항을 생성하기 위한 데이터베이스의 특성과 같은 인수를 고려합니다.

*워크로드*는 데이터베이스 관리 프로그램이 주어진 기간 동안 처리해야 하는 SQL문 세트입니다. 다음에 대해 디자인 어드바이저가 실행될 수 있습니다.

- db2advis 명령을 사용하여 인라인을 입력하는 단일 SQL문

- DB2 스냅샷에서 캡처된 동적 SQL문 세트
- 워크로드 파일에 들어 있는 SQL문 세트

새 워크로드 파일을 작성하거나 기존 워크로드 파일을 수정할 수 있습니다. 다음을 포함한 여러 소스에서 파일로 명령문을 임포트할 수 있습니다.

- 구분된 텍스트 파일
- 이벤트 모니터 테이블
- Query Patroller 실행기록 데이터 테이블(명령행에서 -qp 옵션을 사용)
- EXPLAIN_STATEMENT 테이블의 Explian문
- DB2 스냅샷을 통해 캡처된 최근 SQL문
- 워크로드 관리 프로그램 활동 테이블
- 워크로드 관리 프로그램 이벤트 모니터 테이블(명령행에서 -wlm 옵션을 사용)

SQL문을 워크로드 파일로 임포트한 후 명령문을 추가, 변경, 수정 또는 제거하고 빈도를 수정할 수 있습니다.

- 동적 SQL문에 대해 디자인 어드바이저를 실행하려면 다음을 수행하십시오.
  1. 다음 명령을 사용하여 데이터베이스 모니터를 재설정하십시오.

     ```
     db2 reset monitor for database database-name
     ```

  2. 데이터베이스에 대한 동적 SQL문이 실행될 수 있도록 적당한 시간 동안 대기하십시오.
  3. -g 옵션을 사용하여 db2advis 명령을 호출하십시오. 나중에 참조하기 위해 ADVISE_WORKLOAD 테이블에 동적 SQL문을 저장하려는 경우, -p 옵션도 사용하십시오.
- 워크로드 파일의 SQL문에 대해 디자인 어드바이저를 실행하려면 다음을 수행하십시오.
  1. 각 SQL문을 세미콜론으로 분리하여 워크로드 파일을 수동으로 작성하거나, 위에 나열된 하나 이상의 소스에서 SQL문을 임포트하십시오.
  2. 워크로드에서 명령문의 빈도를 설정하십시오. 워크로드 파일의 모든 명령문에는 디폴트로 빈도 1이 지정됩니다. SQL문의 빈도는 다른 명령문이 발생하는 횟수에 비해 워크로드 내에서 명령문이 발생하는 횟수를 나타냅니다. 예를 들어, 특정 SELECT문은 워크로드에서 100번 발생할 수도 있는 반면, 다른 SELECT문은 10번 발생합니다. 이러한 두 명령문의 상대적 빈도를 나타내려면, 첫 번째 SELECT문에 빈도 10을 지정할 수 있고, 두 번째 SELECT문은 빈도 1을 갖습니다. 다음 라인을 명령문 뒤에 삽입하여 워크로드에서 특정 명령문의 빈도 또는 중량을 수동으로 변경할 수 있습니다. `- - # SET FREQUENCY n`. 여기서 *n*은 명령문에 지정하려는 빈도 값입니다.
  3. 워크로드 파일의 이름이 뒤에 오는 -i 옵션을 사용하여 db2advis 명령을 호출하십시오.

- ADVISE_WORKLOAD 테이블에 포함된 워크로드에 대해 디자인 어드바이저를 실행하려면, 워크로드의 이름이 뒤에 오는 -w 옵션을 사용하여 db2advis 명령을 호출하십시오.

디자인 어드바이저를 사용하여 단일 파티션에서 다중 파티션 데이터베이스로 변환

디자인 어드바이저를 사용하면 단일 파티션 데이터베이스에서 다중 파티션 데이터베이스로 변환하는 데 도움이 될 수 있습니다.

이 태스크에 대한 정보

새 인덱스, 구체화된 쿼리 테이블(MQT) 및 다차원 클러스터링(MDC) 테이블에 대한 권장사항을 제공하는 것 외에도 디자인 어드바이저는 데이터 분배에 대한 권장사항을 제공할 수 있습니다.

프로시저

1. db2licm 명령을 사용하여 데이터베이스 파티셔닝 기능(DPF) 라이센스 키를 등록하십시오.
2. 다중 파티션 데이터베이스 파티션 그룹에서 적어도 하나의 테이블 스페이스를 작성하십시오.

   주: 디자인 어드바이저는 기존 테이블 스페이스로의 데이터 재분배만 권장할 수 있습니다.
3. db2advis 명령에서 파티셔닝 옵션을 지정하여 디자인 어드바이저를 실행하십시오.
4. 디자인 어드바이저에 의해 생성된 DDL문을 실행하기 전에 db2advis 출력 파일을 약간 수정하십시오. 디자인 어드바이저가 생성하는 DDL 스크립트를 실행할 수 있으려면 먼저 데이터베이스 파티셔닝이 설정되어야 하므로, 리턴되는 스크립트에서 권장사항이 주석으로 제공됩니다. 권장사항에 따라 테이블을 변환할지 여부는 사용자가 결정해야 합니다.

디자인 어드바이저 제한사항

인덱스, 구체화된 쿼리 테이블(MQT), 다차원 클러스터링(MDC) 테이블 및 데이터베이스 파티셔닝에 대한 디자인 어드바이저 권장사항과 연관된 특정 제한사항이 있습니다.

인덱스 권장사항에 대한 제한사항

- MQT에 권장되는 인덱스는 MQT 새로 고침 성능이 아니라 워크로드 성능을 향상시키기 위해 설계되었습니다.
- 클러스터링 RID 인덱스는 MDC 테이블에만 권장됩니다. 디자인 어드바이저는 테이블에 대한 MDC 구조를 생성하기 보다는 클러스터링 RID 인덱스를 옵션으로 포함합니다.

- 버전 9.7 디자인 어드바이저는 파티션된 테이블에서 파티션된 인덱스를 권장하지 않습니다. 모든 인덱스는 명시적 NOT PARTITIONED절과 함께 권장됩니다.

MQT 권장사항에 대한 제한사항

- 디자인 어드바이저는 증분 MQT를 권장하지 않습니다. 증분 MQT를 작성하려는 경우 스테이징 테이블의 선택사항을 통해 REFRESH IMMEDIATE MQT를 증분 MQT로 변환할 수 있습니다.
- MQT에 권장되는 인덱스는 MQT 새로 고침 성능이 아니라 워크로드 성능을 향상시키기 위해 설계되었습니다.
- 워크로드에 갱신, 삽입 또는 삭제 조작이 포함되지 않는 경우, 권장 REFRESH IMMEDIATE MQT 갱신의 성능 영향이 고려되지 않습니다.
- REFRESH IMMEDIATE MQT에 MQT 쿼리 정의의 GROUP BY 절에 있는 컬럼으로 구성된 내포된 고유 키에서 작성된 고유 인덱스가 포함되는 것이 좋습니다.

MDC 권장사항에 대한 제한사항

- 디자인 어드바이저에서 테이블에 대해 MDC를 고려하려면 먼저 기존 테이블이 충분한 데이터로 채워져야 합니다. 최소 20-30MB의 데이터가 권장됩니다. 12보다 작은 테이블은 고려되지 않습니다.
- 샘플링 옵션, -r이 db2advis 명령과 함께 사용되지 않는 한 새 MQT에 대한 MDC 권장사항은 고려되지 않습니다.
- 디자인 어드바이저는 입력된, 임시, 또는 페더레이티드 테이블에 대해 MDC를 권장하지 않습니다.
- db2advis 명령을 실행하는 동안 사용되는 샘플링 데이터를 위한 충분한 스토리지 스페이스(대형 테이블의 경우 테이블 데이터의 약 1%)가 사용 가능해야 합니다.
- 수집된 통계가 없는 테이블은 고려되지 않습니다.
- 디자인 어드바이저는 다중 컬럼 차원에 대해 권장하지 않습니다.

데이터베이스 파티셔닝 권장사항에 대한 제한사항

디자인 어드바이저는 DB2 Enterprise Server Edition에 대해서만 데이터베이스 파티셔닝을 권장할 수 있습니다.

추가적인 제한사항

디자인 어드바이저가 실행될 때 임시 시뮬레이션 카탈로그 테이블이 작성됩니다. 불완전한 실행으로 인해 이러한 테이블 중 일부가 삭제되지 않을 수 있습니다. 이 경우, 디자인 어드바이저를 사용, 유틸리티를 재시작하여 이러한 테이블을 삭제할 수 있습니다. 시뮬레이션 카탈로그 테이블을 제거하려면 -f 옵션과 -n 옵션을 모두 지정하십시오(-n의

경우, 불완전한 실행에 사용된 것과 동일한 사용자 이름을 지정). -f 옵션을 지정하지 않을 경우, 디자인 어드바이저에서 테이블을 제거하는 데 필요한 DROP문만 생성하고 실제로는 테이블을 제거하지 않습니다.

주: 버전 9.5에서는, -f 옵션이 디폴트입니다. 즉, MQT 선택과 함께 db2advis를 실행할 경우, 데이터베이스 관리 프로그램에서 스키마 이름과 동일한 사용자 ID를 사용하여 모든 로컬 시뮬레이션 카탈로그 테이블을 삭제합니다.

이러한 시뮬레이션 카탈로그 테이블을 저장하려면 카탈로그 데이터베이스 파티션에 별도의 테이블 스페이스를 작성하고, CREATE 또는 ALTER TABLESPACE문에서 DROPPED TABLE RECOVERY 옵션을 해제(OFF)로 설정해야 합니다. 이렇게 하면 보다 간편한 정리와 신속한 디자인 어드바이저 실행이 가능해집니다.

제 2 부 문제점 해결

훌륭한 문제점 분석의 첫 번째 단계는 문제점을 완벽히 설명하는 것입니다. 문제점 설명이 없으면 어디서부터 문제점의 원인을 조사해야 할지 알지 못합니다.

이 단계에서는 스스로에게 다음과 같은 기본적인 질문을 하게 됩니다.

- 증상은 어떤 것입니까?
- 어디서 문제점이 발생합니까?
- 언제 문제점이 발생합니까?
- 어떤 조건에서 문제점이 발생합니까?
- 문제점을 재현할 수 있습니까?

이러한 질문과 기타 질문에 답변하면 대부분의 문제점을 충분히 설명할 수 있게 되며, 문제점 해결 경로를 시작하는 최상의 방법입니다.

증상은 어떤 것입니까?

문제점을 설명하기 시작할 때 가장 분명한 질문은 "문제점은 어떤 것입니까?"입니다. 이는 직설적인 질문처럼 여겨질 수 있지만, 기타 여러 질문으로 세분화하여 문제점에 대한 보다 구체적인 그림을 작성할 수 있습니다. 이러한 질문은 다음과 같습니다.

- 누가 또는 무엇이 문제점을 보고합니까?
- 오류 코드 및 오류 메시지는 어떤 것입니까?
- 어떻게 실패합니까?(예: 루프, 정지, 중지, 성능 저하, 올바르지 않은 결과)
- 비즈니스에 어떤 영향을 줍니까?

어디서 문제점이 발생합니까?

문제점이 시작된 위치를 판별하는 것이 항상 쉬운 일은 아니지만, 문제점 해결의 가장 중요한 단계 중 하나입니다. 보고 구성요소와 실패 구성요소 간에는 많은 기술 계층이 존재할 수 있습니다. 네트워크, 디스크 및 드라이버는 문제점을 조사할 때 고려할 몇 가지 구성요소에 불과합니다.

- 문제점이 해당 플랫폼에서만 발생합니까, 아니면 다중 플랫폼에 공통적으로 발생합니까?
- 현재 환경 및 구성이 지원됩니까?
- 데이터베이스 서버 또는 리모트 서버에서 로컬로 응용프로그램을 실행 중입니까?
- 관련된 게이트웨이가 있습니까?
- 데이터베이스가 개별 디스크 또는 RAID 디스크 배열에 저장되어 있습니까?

이러한 유형의 질문은 문제점 계층을 분리하는 데 도움을 주며, 문제점 원인을 판별하는 데 필요합니다. 하나의 계층은 하나의 문제점만을 보고하기 때문에 항상 근본 원인이 여기에 있음을 의미하는 것은 아님을 기억하십시오.

문제점이 발생하는 위치를 식별하는 것의 일부는 문제점이 존재하는 환경을 이해하는 것입니다. 운영 체제, 버전, 모든 해당 소프트웨어 및 버전, 하드웨어 정보 등 문제점 환경을 완벽히 설명하는 데 항상 어느 정도의 시간을 할애해야 합니다. 함께 실행하지 않거나 함께 완벽하게 테스트되지 않은 소프트웨어 레벨을 찾아 많은 문제점을 설명할 수 있기 때문에 지원되는 구성인 환경에서 실행 중인지 확인하십시오.

언제 문제점이 발생합니까?

실패로 이끄는 이벤트의 세부 타임 라인을 개발하는 것은 문제점 분석에서 특히, 일회성 어커런스인 해당 경우에는 또 다른 필수 단계입니다. 역방향으로(오류가 보고된 시간(밀리초까지 가능한 정확히)에서 시작) 작업하면 이를 가장 쉽게 수행할 수 있으며, 사용 가능한 로그 및 정보를 통해 역방향으로 작업할 수 있습니다. 대개 진단 로그에서 발견하는 첫 번째 의심스런 이벤트까지 살펴보기만 하면 되며, 이를 수행하는 것이 항상 쉬운 것만은 아니지만 연습하면 됩니다. 각각 자체 진단 정보가 있는 여러 계층의 기술이 있는 경우 어디서 중지할지 아는 것이 특히 어렵습니다.

- 문제점이 낮이나 밤의 특정 시간에만 발생합니까?
- 얼마나 자주 발생합니까?
- 문제점이 보고되는 시간으로 이끄는 이벤트 시퀀스는 어떤 것입니까?
- 환경 변경(예: 기존 소프트웨어나 하드웨어 업그레이드 또는 새 소프트웨어나 하드웨어 설치) 이후에 문제점이 발생합니까?

이와 같은 질문에 응답하면 이벤트의 세부 타임 라인을 작성하는 데 도움이 되며, 조사할 참조 틀을 제공합니다.

어떤 조건에서 문제점이 발생합니까?

문제점 발생 당시 그 밖에 무엇이 실행 중이었는지 아는 것은 완벽한 문제점 설명에 중요합니다. 문제점이 특정 환경이나 특정 조건에서 발생하는 경우, 이는 문제점 원인의 주요 지표일 수 있습니다.

- 동일한 태스크를 수행할 때 문제점이 항상 발생합니까?
- 문제점을 표면화하려면 특정 이벤트 시퀀스가 발생해야 합니까?
- 다른 응용프로그램이 동시에 실패합니까?

이러한 유형의 질문에 답변하면 문제점이 발생하는 환경을 설명할 수 있으며, 종속성을 상관시킬 수 있습니다. 여러 개의 문제점이 거의 동일한 시간에 발생했을 수도 있기 때문에 문제점이 항상 관련되어 있음을 반드시 의미하는 것은 아님을 기억하십시오.

문제점을 재현할 수 있습니까?

문제점 설명 및 조사 관점에서 "이상적인" 문제점은 재현 가능한 문제점입니다. 재현 가능한 문제점의 경우 조사를 돕기 위해 사용할 수 있는 대형 도구 또는 프로시저 세트를 거의 항상 갖고 있습니다. 따라서 재현 가능한 문제점은 대개 디버그 및 해결이 쉽습니다.

그러나 재현 가능한 문제점의 단점이 있습니다. 문제점이 비즈니스에 중요한 영향을 주는 경우, 문제점 재발을 원하지 않습니다. 가능하면, 이 경우에는 테스트 또는 개발 환경에서 문제점을 재현하는 것이 대개 바람직합니다.

- 테스트 시스템에서 문제점을 재현할 수 있습니까?
- 동일한 유형의 문제점이 발생하는 사용자나 응용프로그램이 여러 개입니까?
- 단일 명령, 명령 세트 또는 특정 응용프로그램이나 독립형 응용프로그램을 실행하여 문제점을 재현할 수 있습니까?
- DB2 명령행에서 동등한 명령/쿼리를 입력하여 문제점을 재현할 수 있습니까?

일반적으로 문제점을 조사할 때 유연성과 제어가 탁월하기 때문에 테스트 또는 개발 환경에서 단일 부수 문제점을 재현하는 것이 대개 바람직합니다.

제 5 장 문제점 해결 도구

진단 데이터 수집, 형식화 또는 분석을 돕기 위해 사용 가능한 도구는 다음과 같습니다.

- db2dart

  db2dart 명령을 사용하여 데이터베이스 및 데이터베이스 내 오브젝트의 아키텍처 정확성을 검증할 수 있습니다. 그렇지 않으면 액세스할 수 없는 테이블에서 데이터를 추출하기 위해 데이터베이스 제어 파일의 컨텐츠를 표시하는 데도 사용할 수 있습니다.

- db2diag

  db2diag 도구는 db2diag 로그 파일에서 사용 가능한 정보 볼륨을 필터링하고 형식화합니다. db2diag 로그 파일의 레코드를 필터링하면 문제점 해결 시 필요한 레코드를 찾는 데 필요한 시간을 줄일 수 있습니다.

- db2greg

  db2greg 도구를 사용하여 전역 레지스트리를 보고 편집할 수 있습니다.

- db2level

  db2level 명령은 DB2 인스턴스의 버전 및 서비스 레벨(빌드 레벨 및 FixPack 번호)을 판별하는 데 도움을 줍니다.

- db2look

  다른 데이터베이스와 구조적으로 유사한 데이터베이스를 작성할 수 있으면 유리한 경우가 많습니다. 예를 들어, 프로덕션 시스템에서 새 응용프로그램 또는 복구 플랜을 테스트하지 않고 구조 및 데이터가 유사한 테스트 시스템을 작성한 후 프로덕션 시스템에 역효과를 주지 않고 테스트 시스템에 대해 테스트를 수행하는 것이 훨씬 타당합니다. db2look 도구를 사용하여 다른 데이터베이스에서 한 데이터베이스의 데이터베이스 오브젝트를 재생하는 데 필요한 필수 DDL문을 추출할 수 있습니다. 또한 이 도구는 한 데이터베이스에서 다른 데이터베이스로 통계를 복제하는 데 필요한 필수 SQL문은 물론 데이터베이스 구성, 데이터베이스 관리 프로그램 구성 및 레지스트리 변수를 복제하는 데 필요한 명령문을 생성할 수 있습니다.

- db2ls

  시스템에 DB2 제품 사본을 여러 개 설치할 수 있고 사용자가 선택한 경로에 DB2 제품과 기능을 설치할 수 있는 유연성으로 인해, 설치되는 제품과 설치 위치를 추적

하는 데 도움이 되는 도구가 필요합니다. 지원되는 Linux 및 UNIX 운영 체제에서 db2ls 명령은 시스템에 설치된 DB2 제품과 기능(DB2 버전 9 HTML 문서 포함)을 나열합니다.

- db2pd

  db2pd 도구는 DB2 메모리 세트에서 신속하면서도 즉각적으로 정보를 리턴할 수 있기 때문에 문제점 해결에 사용됩니다.

- db2support

  DB2 문제점에 대한 정보를 수집할 때 실행해야 하는 가장 중요한 DB2 유틸리티는 db2support입니다. db2support 유틸리티는 사용 가능한 모든 DB2 및 시스템 진단 정보를 자동으로 수집합니다. 이 유틸리티에는 문제점 상황에 대한 질문을 제기하는 선택적 대화식 "질문과 응답" 세션도 있습니다.

- 추적

  DB2에 반복적이면서 재현 가능한 문제점이 발생하는 경우, 추적을 사용하여 이에 대한 추가 정보를 캡처할 수 있는 경우가 있습니다. 일반적인 상황에서는 IBM Software Suppor에서 요구할 경우에만 추적을 사용해야 합니다. 추적 프로세스는 추적 기능 설정, 오류 재현, 데이터 수집을 수반합니다.

- 플랫폼별 도구(Windows) (Linux 및 UNIX)

  Windows, Linux 및 UNIX 운영 체제와 함께 제공되는 유용한 진단 도구를 사용하여 시스템에서 발생하고 있는 문제점의 원인을 식별하는 데 도움을 줄 수 있는 데이터를 수집하고 처리할 수 있습니다.

db2dart 도구 개요

db2dart 명령을 사용하여 데이터베이스 및 데이터베이스 내 오브젝트의 아키텍처 정확성을 검증할 수 있습니다. 그렇지 않으면 액세스할 수 없는 테이블에서 데이터를 추출하기 위해 데이터베이스 제어 파일의 컨텐츠를 표시하는 데도 사용할 수 있습니다.

가능한 옵션을 모두 표시하려면 매개변수 없이 db2dart 명령을 실행하십시오. 명령행에 명시적으로 지정되지 않은 경우 테이블 스페이스 ID와 같은 매개변수를 필요로 하는 일부 옵션을 입력하도록 프롬프트가 표시됩니다.

디폴트로 db2dart 유틸리티는 `databaseName.RPT` 이름을 가진 보고서 파일을 작성합니다. 단일 파티션 데이터베이스 파티션 환경의 경우, 파일은 현재 디렉토리에 작성됩니다. 다중 파티션 데이터베이스 파티션 환경의 경우, 파일은 진단 디렉토리의 서브디렉토리에 작성됩니다. 서브디렉토리는 `DART####`입니다(여기서 `####`은 데이터베이스 파티션 번호임).

db2dart 유틸리티는 디스크에서 직접 읽음으로써 데이터베이스의 데이터 및 메타데이터에 액세스합니다. 이 때문에 사용 중인 연결을 여전히 갖고 있는 데이터베이스에 대해 도구를 실행해서는 안됩니다. 연결이 있는 경우, 도구는 예를 들어 버퍼 풀의 페이지 또는 메모리의 제어 구조에 대해 알지 못하며 잘못된 오류를 결과로 보고할 수 있습니다. 마찬가지로, 응급 복구가 필요하거나 롤 포워드 복구를 완료하지 않은 데이터베이스에 대해 db2dart를 실행하는 경우 디스크의 데이터 불일치 특징으로 인해 유사한 불일치가 발생할 수 있습니다.

INSPECT 및 db2dart 비교

INSPECT 명령은 데이터베이스 아키텍처 무결성을 검사하고 데이터베이스 페이지의 페이지 일관성을 검사합니다. INSPECT 명령은 테이블 오브젝트 구조와 테이블 스페이스 구조가 올바른지 점검합니다. 교차 오브젝트 유효성 확인은 온라인 인덱스의 데이터 일관성 검사를 수행합니다. db2dart 명령은 데이터베이스의 아키텍처 정확성을 조사하고 발생한 오류를 보고합니다.

INSPECT 명령은 데이터베이스, 테이블 스페이스 및 테이블을 점검할 수 있다는 측면에서 db2dart 명령과 유사합니다. 두 명령 간의 중요한 차이는 데이터베이스를 비활성화한 후에 db2dart를 실행해야 하는 반면, INSPECT는 데이터 연결을 필요로 하고 동시 사용 중인 데이터베이스 연결이 있는 동안에 실행할 수 있다는 것입니다.

데이터베이스를 비활성화하지 않으면, db2dart가 신뢰할 수 없는 결과를 생성합니다.

다음 표는 db2dart 및 INSPECT 명령이 수행하는 테스트 간의 차이점을 나열합니다.

표 76. 테이블 스페이스에 대한 db2dart 및 INSPECT 기능 비교

수행되는 테스트	db2dart	INSPECT
SMS 테이블 스페이스		
테이블 스페이스 파일 점검	YES	NO
내부 페이지 헤더 필드 컨텐츠 유효성 확인	YES	YES
DMS 테이블 스페이스		
두 개 이상의 오브젝트가 가리키는 Extent Map 점검	YES	NO
모든 Extent Map 페이지의 일관성 비트 오류 점검	NO	YES
모든 스페이스 맵 페이지의 일관성 비트 오류 점검	NO	YES
내부 페이지 헤더 필드 컨텐츠 유효성 확인	YES	YES
Extent Map이 테이블 스페이스 맵과 일치하는지 검증	YES	NO

표 77. 데이터 오브젝트에 대한 db2dart 및 INSPECT 기능 비교

수행되는 테스트	db2dart	INSPECT
데이터 오브젝트의 일관성 비트 오류 점검	YES	YES
특수 제어 행 컨텐츠 점검	YES	NO
가변 길이 컬럼의 길이 및 위치 점검	YES	NO
테이블 행의 LONG VARCHAR, LONG VARGRAPHIC 및 대형 오브젝트(LOB) 디스크립터 점검	YES	NO
전체 요약 페이지 수, 사용된 페이지 수 및 여유 공간 비율 점검	NO	YES
내부 페이지 헤더 필드 컨텐츠 유효성 확인	YES	YES
각 행 레코드 유형 및 길이 검증	YES	YES
행이 겹치지 않는지 검증	YES	YES

표 78. 인덱스 오브젝트에 대한 db2dart 및 INSPECT 기능 비교

수행되는 테스트	db2dart	INSPECT
일관성 비트 오류 점검	YES	YES
인덱스 키의 위치와 길이 및 오버랩핑이 있는지 여부 점검	YES	YES
인덱스의 키 순서지정 점검	YES	NO
전체 요약 페이지 수 및 사용된 페이지 수 판별	NO	YES
내부 페이지 헤더 필드 컨텐츠 유효성 확인	YES	YES
고유 키의 고유성 검증	YES	NO
주어진 인덱스 항목에 대해 데이터 행이 존재하는지 점검	NO	YES
데이터 값에 대한 각 키 검증	NO	YES

표 79. 블록 맵 오브젝트에 대한 db2dart 및 INSPECT 기능 비교

수행되는 테스트	db2dart	INSPECT
일관성 비트 오류 점검	YES	YES
전체 요약 페이지 수 및 사용된 페이지 수 판별	NO	YES
내부 페이지 헤더 필드 컨텐츠 유효성 확인	YES	YES

표 80. Long 필드 및 LOB 오브젝트에 대한 db2dart 및 INSPECT 기능 비교

수행되는 테스트	db2dart	INSPECT
할당 구조 점검	YES	YES

표 80. Long 필드 및 LOB 오브젝트에 대한 db2dart 및 INSPECT 기능 비교 (계속)

수행되는 테스트	db2dart	INSPECT
전체 요약 페이지 수 및 사용된 페이지 수(LOB 오브젝트의 경우만) 판별	NO	YES

또한 db2dart 명령을 사용하여 다음 조치를 수행할 수 있습니다.

- 데이터 페이지 형식화 및 덤프
- 인덱스 페이지 형식화 및 덤프
- 데이터 행을 컬럼 식별자가 있는 ASCII로 형식화
- 인덱스를 유효하지 않음으로 표시

INSPECT 명령을 사용하여 해당 조치를 수행할 수는 없습니다.

db2diag 도구를 사용하여 db2diag 로그 파일 분석

데이터베이스 및 시스템 관리자가 사용하도록 하기 위한 기본 로그 파일은 관리 통지 로그입니다. db2diag 로그 파일은 문제점 해결 지원 목적으로 DB2 지원부에서 사용하도록 하기 위한 것입니다.

관리 통지 로그 메시지도 표준화된 메시지 형식을 사용하여 db2diag 로그 파일에 로그됩니다.

db2diag 도구는 db2diag 로그 파일에서 사용 가능한 정보 볼륨을 필터링하고 형식화합니다. db2diag 로그 파일 레코드를 필터링하면 문제점 해결 시 필요한 레코드를 찾는 데 필요한 시간을 줄일 수 있습니다.

예 1: 데이터베이스 이름별로 db2diag 로그 파일 필터링

인스턴스에 몇 개의 데이터베이스가 있고 "SAMPLE" 데이터베이스에 속한 해당 메시지만 보려면, 다음과 같이 db2diag 로그 파일을 필터링할 수 있습니다.

```
db2diag -g db=SAMPLE
```

따라서 다음과 같이 "DB: SAMPLE"을 포함한 db2diag 로그 파일 레코드만 보게 됩니다.

```
2006-02-15-19.31.36.114000-300 E21432H406         LEVEL: Error
PID     : 940                  TID  : 660         PROC : db2syscs.exe
INSTANCE: DB2                  NODE : 000         DB   : SAMPLE
APPHDL  : 0-1056               APPID: *LOCAL.DB2.060216003103
FUNCTION: DB2 UDB, base sys utilities, sqleDatabaseQuiesce, probe:2
MESSAGE : ADM7507W  Database quiesce request has completed successfully.
```

예 2: 프로세스 ID별로 db2diag 로그 파일 필터링

다음 명령을 사용하여 프로세스 ID(PID)가 2200인 파티션 0, 1, 2 또는 3에서 실행 중인 프로세스가 생성한 심각한 오류 메시지를 모두 표시할 수 있습니다.

```
db2diag -g level=Severe,pid=2200 -n 0,1,2,3
```

이 명령은 db2diag -l severe -pid 2200 -n 0,1,2,3을 포함하여 두 가지 다른 방법으로 작성되었을 수 있음에 유의하십시오. 또한 -g 옵션은 대소문자 구분 검색을 지정하므로 여기서 ″Severe″는 작동하지만 ″severe″가 사용되는 경우에는 실패함에 유의해야 합니다. 이러한 명령은 다음과 같이 이러한 요구사항을 충족시키는 db2diag 로그 파일 레코드를 검색합니다.

```
2006-02-13-14.34.36.027000-300 I18366H421         LEVEL: Severe
PID     : 2200                 TID  : 660         PROC : db2syscs.exe
INSTANCE: DB2                  NODE : 000         DB   : SAMPLE
APPHDL  : 0-1433               APPID: *LOCAL.DB2.060213193043
FUNCTION: DB2 UDB, data management, sqldPoolCreate, probe:273
RETCODE : ZRC=0x8002003C=-2147352516=SQLB_BAD_CONTAINER_PATH
          "Bad container path"
```

예 3: db2diag 도구 출력 포맷팅

다음 명령은 2006년 1월 1일 이후에 발생하는, 파티션 0, 1 또는 2에서 로그된 심각한 오류와 심각하지 않은 오류가 포함된 레코드를 모두 필터링합니다. 시간소인, 파티션 번호 및 레벨은 첫 번째 라인에, PID, TID 및 인스턴스 이름은 두 번째 라인에 나타나고 그 후에 오류 메시지가 오는 일치된 레코드를 출력합니다.

```
db2diag -time 2006-01-01 -node "0,1,2" -level "Severe, Error" | db2diag -fmt
"Time: %{ts}
Partition: %node Message Level: %{level} ₩nPid: %{pid}  Tid: %{tid}
Instance: %{instance}₩nMessage: @{msg}₩n"
```

생성된 출력 예는 다음과 같습니다.

```
Time: 2006-02-15-19.31.36.099000 Partition: 000 Message Level: Error
Pid: 940 Tid:940 Instance: DB2
Message: ADM7506W  Database quiesce has been requested.
```

자세한 정보를 보려면 다음 명령을 실행하십시오.

- db2diag -help - 사용 가능한 모든 옵션에 대한 간단한 설명을 제공합니다.
- db2diag -h brief - 예 없이 모든 옵션에 대한 설명을 제공합니다.
- db2diag -h notes - 사용법 참고 및 제한사항을 제공합니다.
- db2diag -h examples - 시작할 작은 예 세트를 제공합니다.
- db2diag -h tutorial - 사용 가능한 모든 옵션에 대한 예를 제공합니다.
- db2diag -h all - 가장 완벽한 옵션 목록을 제공합니다.

예 4: 다른 기능의 메시지 필터링

다음 예는 데이터베이스 관리 프로그램 내 특정 기능(또는 모든 기능)의 메시지만 표시하는 방법을 나타냅니다. 지원되는 기능은 다음과 같습니다.

- ALL - 모든 기능의 레코드를 리턴합니다.
- MAIN - DB2 일반 진단 로그(예: db2diag 로그 파일 및 관리 통지 로그)의 레코드를 리턴합니다.
- OPSTATS - 옵티마이저 통계와 관련된 레코드를 리턴합니다.

MAIN 기능의 메시지를 읽으려면 다음 명령을 실행하십시오.

```
db2diag -facility MAIN
```

OPSTATS 기능의 메시지를 표시하고 레벨이 Severe인 레코드를 필터링하려면 다음 명령을 실행하십시오.

```
db2diag -fac OPSTATS -level Severe
```

사용 가능한 모든 기능의 메시지를 표시하고 instance=harmistr 및 level=Error인 레코드를 필터링하려면 다음 명령을 실행하십시오.

```
db2diag -fac all -g instance=harmistr,level=Error
```

레벨이 Error이고 특정 형식으로 Timestamp 및 PID 필드를 출력하는 OPSTATS 기능의 메시지를 모두 표시하려면 다음 명령을 실행하십시오.

```
db2diag -fac opstats -level Error -fmt " Time :%{ts} Pid :%{ts}"
```

db2greg를 사용하여 전역 레지스트리 표시 및 변경(UNIX)

UNIX 및 Linux 플랫폼에서는 db2greg 명령을 사용하여 전역 레지스트리를 볼 수 있습니다.

DB2 버전 9.7 이상에서는 DB2 전역 프로파일 레지스트리가 <DB2DIR>/default.env 텍스트 파일에 기록되지 않습니다. 이제 현재 DB2 설치와 관련된 DB2 프로파일 설정을 등록하는 데 전역 레지스트리 파일(global.reg)이 사용됩니다.

전역 레지스트리는 UNIX 및 Linux 플랫폼에만 존재합니다.

- 루트 설치의 경우, 전역 레지스트리 파일은 /var/db2/global.reg(HP-UX에서는 /var/opt/db2/global.reg)에 있습니다.
- 루트가 아닌 설치의 경우, 전역 레지스트리 파일은 $HOME/sqllib/global.reg에 있습니다(여기서 $HOME은 루트가 아닌 사용자의 홈 디렉토리임).

전역 레지스트리는 다음 세 개의 서로 다른 레코드 유형으로 이루어집니다.

- "서비스": 서비스 레코드에는 제품 레벨에 대한 정보(예: 버전 및 설치 경로)가 포함됩니다.
- "인스턴스": 인스턴스 레코드에는 인스턴스 레벨에서의 정보(예: 인스턴스 이름, 인스턴스 경로, 버전 및 "start-at-boot" 플래그)가 포함됩니다.

- "변수": 변수 레코드에는 변수 레벨에서의 정보(예: 변수 이름, 변수 값)가 포함됩니다.
- 주석.

db2greg 도구를 사용하여 전역 레지스트리를 볼 수 있습니다. 이 도구는 sqllib/bin에 있으며, bin 아래의 install 디렉토리에도 있습니다(루트로 로그인한 경우).

db2greg 도구를 사용하여 전역 레지스트리를 편집할 수 있습니다. 루트 설치의 전역 레지스트리를 편집하려면 루트 권한이 필요합니다.

IBM Software Support에서 요청한 경우에만 db2greg 도구를 사용해야 합니다.

제품 버전 및 서비스 레벨 식별

db2level 명령은 DB2 인스턴스의 버전 및 서비스 레벨(빌드 레벨 및 FixPack 번호)을 판별하는 데 도움을 줍니다.

DB2 인스턴스가 최신 서비스 레벨인지 여부를 판별하려면, db2level 출력을 DB2 지원부 웹 사이트(www.ibm.com/support/docview.wss?rs=71&uid=swg27007053)의 FixPack 다운로드 페이지에 있는 정보와 비교하십시오.

Windows 시스템에서 db2level 명령을 실행할 경우 일반적인 결과는 다음과 같습니다.

```
DB21085I  Instance "DB2" uses "32" bits and DB2 code release "SQL09010" with
level identifier "01010107".
Informational tokens are "DB2 v9.1.0.189", "n060119", "", and Fix Pack "0".
Product is installed at "c:¥SQLLIB" with DB2 Copy Name "db2build".
```

네 개의 정보용 토큰 조합은 DB2 인스턴스의 정확한 서비스 레벨을 고유하게 식별합니다. 지원을 받기 위해 IBM Software Support에 문의하는 경우 이 정보가 필요합니다.

JDBC 또는 SQLJ 응용프로그램의 경우, IBM DB2 드라이버를 SQLJ 및 JDBC에 사용하는 경우 db2jcc 유틸리티를 실행하여 드라이버 레벨을 판별할 수 있습니다.

```
db2jcc -version

IBM DB2 JDBC Driver Architecture 2.3.63
```

db2look을 사용하여 데이터베이스 가상 갱신

다른 데이터베이스와 구조적으로 유사한 데이터베이스를 작성할 수 있으면 유리한 경우가 많습니다. 예를 들어, 프로덕션 시스템에서 새 응용프로그램 또는 복구 플랜을 테스트하지 않고 대신에, 구조 및 데이터가 유사한 테스트 시스템을 작성한 후 테스트를 수행하는 것이 훨씬 타당합니다.

이런 방식으로 프로덕션 시스템은 테스트 성능 저하 또는 잘못된 응용프로그램의 우발적인 데이터 파괴로 인해 영향을 받지 않습니다. 또한 문제점(예: 유효하지 않은 결과, 성능 문제점 등)을 조사할 때 프로덕션 시스템과 동일한 테스트 시스템에서 문제점을 디버그하는 것이 훨씬 쉬울 수 있습니다.

db2look 도구를 사용하여 다른 데이터베이스에서 한 데이터베이스의 데이터베이스 오브젝트를 재생하는 데 필요한 필수 DDL문을 추출할 수 있습니다. 또한 이 도구는 한 데이터베이스에서 다른 데이터베이스로 통계를 복제하는 데 필요한 필수 SQL문은 물론 데이터베이스 구성, 데이터베이스 관리 프로그램 구성 및 레지스트리 변수를 복제하는 데 필요한 명령문을 생성할 수 있습니다. 새 데이터베이스가 원래 데이터베이스와 똑같은 데이터 세트를 포함하지 않지만 두 시스템에 대해 동일한 액세스 플랜을 선택하기를 원할 수 있기 때문에 이는 중요합니다.

db2look 도구는 *DB2 Command Reference*에 자세히 설명되어 있지만 매개변수 없이 도구를 실행하면 옵션 목록을 볼 수 있습니다. -h 옵션을 사용하여 자세한 사용법을 표시할 수 있습니다.

db2look을 사용하여 데이터베이스의 테이블 가상 갱신

데이터베이스의 테이블에 대한 DDL을 추출하려면 -e 옵션을 사용하십시오. 예를 들어, 첫 번째 데이터베이스의 모든 오브젝트가 새 데이터베이스에 작성되도록 SAMPLE2라는 SAMPLE 데이터베이스 사본을 작성하십시오.

```
C:₩>db2 create database sample2
DB20000I The CREATE DATABASE command completed successfully.
C:₩>db2look -d sample -e > sample.ddl
-- USER is:
-- Creating DDL for table(s)
-- Binding package automatically ...
-- Bind is successful
-- Binding package automatically ...
-- Bind is successful
```

주: 생성될 사용자 정의 스페이스, 데이터베이스 파티션 그룹 및 버퍼 풀에 대한 DDL도 원하면, 위 명령문에서 -e 다음에 -l 플래그를 추가하십시오. 디폴트 데이터베이스 파티션 그룹, 버퍼 풀 및 테이블 스페이스는 추출되지 않습니다. 그 이유는 디폴트로 모든 데이터베이스에 이미 존재하기 때문입니다. 이를 가상 갱신하려면, 수동으로 직접 변경해야 합니다.

텍스트 편집기에서 `sample.ddl` 파일을 불러 오십시오. 이 파일의 DDL을 새 데이터베이스에 대해 실행하려고 하기 때문에 CONNECT TO SAMPLE문을 CONNECT TO SAMPLE2로 변경해야 합니다. -l 옵션을 사용한 경우, 해당 경로도 가리키도록 테이블 스페이스 명령과 연관된 경로를 변경해야 합니다. 나머지 파일 컨텐츠를 살펴보십시오. 샘플 데이터베이스의 모든 사용자 테이블에 대해 CREATE TABLE, ALTER TABLE 및 CREATE INDEX문이 표시되어야 합니다.

```
...
-----------------------------------------------
-- DDL Statements for table "DB2"."ORG"
-----------------------------------------------

 CREATE TABLE "DB2"."ORG"  (
    "DEPTNUMB" SMALLINT NOT NULL ,
    "DEPTNAME" VARCHAR(14) ,
    "MANAGER" SMALLINT ,
    "DIVISION" VARCHAR(10) ,
    "LOCATION" VARCHAR(13) )
   IN "USERSPACE1" ;
...
```

연결 명령문을 변경했으면 다음과 같이 명령문을 실행하십시오.

```
C:₩>db2 -tvf sample.ddl > sample2.out
```

sample2.out 출력 파일을 살펴보십시오. 모든 것이 정상적으로 실행되었어야 합니다. 오류가 발생한 경우, 오류 메시지는 문제점이 무엇인지 나타내야 합니다. 해당 문제점을 수정하고 명령문을 다시 실행하십시오.

출력에서 알 수 있듯이 모든 사용자 테이블에 대한 DDL이 익스포트됩니다. 이는 디폴트 동작이지만 포함된 테이블에 대한 보다 특정한 사용 가능한 기타 옵션이 있습니다. 예를 들어, STAFF 및 ORG 테이블만 포함하려면 -t 옵션을 사용하십시오.

```
C:₩>db2look -d sample -e -t staff org > staff_org.ddl
```

DB2 스키마가 있는 테이블만 포함하려면 -z 옵션을 사용하십시오.

```
C:₩>db2look -d sample -e -z db2 > db2.ddl
```

테이블 통계 가상 갱신

테스트 데이터베이스의 의도가 성능 테스트를 수행하거나 성능 문제점을 디버그하는 것이면, 두 데이터베이스 모두에 대해 생성된 액세스 플랜이 동일해야 합니다. 옵티마이저는 통계, 구성 매개변수, 레지스트리 변수 및 환경 변수를 기반으로 액세스 플랜을 생성합니다. 이런 것들이 두 시스템 간에 동일하면, 액세스 플랜이 동일할 가능성이 아주 높습니다.

두 데이터베이스 모두 똑같은 데이터를 로드했고 RUNSTATS의 동일한 옵션이 둘 다에서 수행되는 경우, 통계는 동일해야 합니다. 그러나 데이터베이스에 서로 다른 데이터가 포함되어 있거나 테스트 데이터베이스에서 데이터 서브세트만 사용되는 경우에는 통계가 아주 다를 가능성이 있습니다. 이러한 경우, db2look을 사용하여 프로덕션 데이터베이스에서 통계를 수집하고 데이터베이스에 이를 배치할 수 있습니다. 갱신 가능한 카탈로그 테이블의 SYSSTAT 세트에 대해 UPDATE문을 작성하는 것은 물론 모든 테이블에 대해 RUNSTATS 명령을 실행하면 이를 수행할 수 있습니다.

통계 명령문 작성 옵션은 -m입니다. SAMPLE/SAMPLE2 예로 돌아가서, SAMPLE로부터 통계를 수집하고 SAMPLE2에 추가하십시오.

```
C:₩>db2look -d sample -m > stats.dml
-- USER is:
-- Running db2look in mimic mode
```

위와 같이 CONNECT TO SAMPLE문이 CONNECT TO SAMPLE2로 변경되도록 출력 파일을 편집해야 합니다. 다시 나머지 파일을 살펴 RUNSTATS 및 UPDATE문의 일부에 포함된 내용을 확인하십시오.

```
...
-- Mimic table ORG
RUNSTATS ON TABLE "DB2"."ORG" ;

UPDATE SYSSTAT.INDEXES
SET NLEAF=-1,
    NLEVELS=-1,
    FIRSTKEYCARD=-1,
    FIRST2KEYCARD=-1,
    FIRST3KEYCARD=-1,
    FIRST4KEYCARD=-1,
    FULLKEYCARD=-1,
    CLUSTERFACTOR=-1,
    CLUSTERRATIO=-1,
    SEQUENTIAL_PAGES=-1,
    PAGE_FETCH_PAIRS='',
    DENSITY=-1,
    AVERAGE_SEQUENCE_GAP=-1,
    AVERAGE_SEQUENCE_FETCH_GAP=-1,
    AVERAGE_SEQUENCE_PAGES=-1,
    AVERAGE_SEQUENCE_FETCH_PAGES=-1,
    AVERAGE_RANDOM_PAGES=-1,
    AVERAGE_RANDOM_FETCH_PAGES=-1,
    NUMRIDS=-1,
    NUMRIDS_DELETED=-1,
    NUM_EMPTY_LEAFS=-1
WHERE TABNAME = 'ORG' AND TABSCHEMA = 'DB2   ';
...
```

DDL을 추출하는 -e 옵션의 경우처럼 -t 및 -z 옵션을 사용하여 테이블 세트를 지정할 수 있습니다.

구성 매개변수 및 환경 변수 추출

옵티마이저는 통계, 구성 매개변수, 레지스트리 변수 및 환경 변수를 기반으로 플랜을 선택합니다. 통계의 경우처럼 db2look을 사용하여 필요한 구성 갱신 및 설정 명령문을 생성할 수 있습니다. 이는 -f 옵션을 사용하여 수행됩니다. 예를 들어, 다음과 같습니다.

```
c:₩>db2look -d sample -f>config.txt
-- USER is: DB2INST1
-- Binding package automatically ...
-- Bind is successful
-- Binding package automatically ...
-- Bind is successful
```

config.txt는 다음과 유사한 출력을 포함합니다.

```
-- This CLP file was created using DB2LOOK Version 9.1
-- Timestamp: 2/16/2006 7:15:17 PM
-- Database Name: SAMPLE
-- Database Manager Version: DB2/NT Version 9.1.0
-- Database Codepage: 1252
-- Database Collating Sequence is: UNIQUE


CONNECT TO SAMPLE;

------------------------------------------------------
-- Database and Database Manager configuration parameters
------------------------------------------------------

UPDATE DBM CFG USING cpuspeed 2.991513e-007;
UPDATE DBM CFG USING intra_parallel NO;
UPDATE DBM CFG USING comm_bandwidth 100.000000;
UPDATE DBM CFG USING federated NO;

...

---------------------------------
-- Environment Variables settings
---------------------------------


COMMIT WORK;

CONNECT RESET;
```

주: DB2 컴파일러에 영향을 주는 해당 매개변수 및 변수만 포함됩니다. 컴파일러에 영향을 주는 레지스트리 변수가 디폴트값으로 설정된 경우에는 "환경 변수 설정"에 표시되지 않습니다.

시스템에 설치된 DB2 데이터베이스 제품 나열(Linux 및 UNIX)

지원되는 Linux 및 UNIX 운영 체제에서 db2ls 명령으로 DB2 버전 9.7 HTML 문서를 포함한 시스템에 설치된 DB2데이터베이스 제품 및 기능을 나열합니다.

시작하기 전에

db2ls 명령에 대한 기호 링크를 /usr/local/bin 디렉토리에 사용할 수 있도록 최소한 하나의 DB2 버전 9(또는 이상) 데이터베이스 제품이 루트 사용자에 의해 이미 설치되어 있어야 합니다.

이 태스크에 대한 정보

시스템에 DB2 데이터베이스 제품 사본을 여러 개 설치할 수 있고 사용자가 선택한 경로에 DB2 데이터베이스 제품과 기능을 설치할 수 있는 유연성으로 인해, 설치되는 제품과 설치 위치를 추적하는 데 도움이 되는 도구가 필요합니다. 지원되는 Linux 및 UNIX 운영 체제에서 db2ls 명령으로 DB2 HTML 문서를 포함한 시스템에 설치된 DB2 제품 및 기능을 나열합니다.

db2ls 명령은 설치 미디어 및 시스템의 DB2 설치 사본에서 찾을 수 있습니다. db2ls 명령은 둘 중 한 위치에서 실행할 수 있습니다. db2ls 명령은 IBM Data Server Driver Package를 제외한 모든 제품의 설치 미디어에서 실행할 수 있습니다.

db2ls 명령을 사용하여 다음을 나열할 수 있습니다.

- 여기에서 DB2 데이터베이스 제품이 시스템에 설치되고 DB2 데이터베이스 제품 레벨을 나열
- 특정 설치 경로의 모든 또는 특정한 DB2 데이터베이스 제품 및 기능

제한사항

db2ls 명령이 나열하는 출력은 다음과 같이 사용되는 ID에 따라 다릅니다.

- db2ls 명령이 루트 권한으로 실행되면 루트 DB2 설치만 쿼리됩니다.
- db2ls 명령이 비루트 ID로 실행되면 루트 DB2 설치 및 일치하는 비루트 ID가 소유한 비루트 설치가 쿼리됩니다. 다른 비루트 ID가 소유한 DB2 설치는 쿼리되지 않습니다.

db2ls 명령은 DB2 데이터베이스 제품을 쿼리할 수 있는 유일한 방법입니다. pkginfo, rpm, SMIT 또는 swlist와 같은 Linux 또는 UNIX 운영 체제 원시(native) 유틸리티를 사용하여 DB2 데이터베이스 제품을 쿼리할 수 *없습니다*. DB2 설치와 쿼리 및 인터페이스에 사용했던 원시(native) 설치 유틸리티를 포함한 기존의 스크립트는 변경해야 합니다.

Windows 운영 체제에 db2ls 명령을 사용할 수 *없습니다*.

프로시저

- DB2 데이터베이스 제품을 시스템에 설치하는 경로와 DB2 데이터베이스 제품을 나열하려면 다음을 입력하십시오.

  ```
  db2ls
  ```

  이 명령은 시스템에 설치된 각 DB2 데이터베이스 제품에 대한 다음과 같은 정보를 나열합니다.

  – 설치 경로

  – 레벨

- FixPack
- 특별 설치 번호. 이 컬럼은 IBM DB2 지원에 사용됩니다.
- 설치 날짜. 이 컬럼에는 DB2 데이터베이스 제품이 마지막으로 수정된 날짜가 표시됩니다.
- 설치 프로그램 UID. 이 컬럼에는 DB2 데이터베이스 제품이 설치된 UID가 표시됩니다.

• DB2 데이터베이스 제품 또는 기능에 대한 정보를 특정 설치 경로에 나열하려면 다음과 같이 **q** 매개변수를 지정해야 합니다.

```
db2ls -q -p -b baseInstallDirectory
```

각 항목에 대한 설명은 다음과 같습니다.

- **q**는 제품 또는 기능을 쿼리하는 중임을 지정합니다. 이 매개변수는 필수입니다. DB2 버전 8 제품이 쿼리되면 공백 값이 리턴됩니다.
- **p**는 목록에 기능 보다 제품이 표시되도록 지정합니다.
- **b**는 제품 또는 기능의 설치 디렉토리를 지정합니다. 이 매개변수는 설치 디렉토리의 명령을 실행하는 경우가 아니면 필수입니다.

결과

제공된 매개변수에 따라 명령으로 다음 정보를 나열합니다.

• 설치 경로. 이것은 한 번만 지정되며 각 기능마다 지정되지는 않습니다.

• 다음 정보가 표시됩니다.
- 설치된 기능의 응답 파일 ID 또는 **p** 옵션이 지정된 경우 설치된 제품의 응답 파일(예: `ENTERPRISE_SERVER_EDITION`)
- 기능 이름 또는 **p** 옵션이 지정된 경우 제품 이름
- 제품 버전, 릴리스, 수정 레벨, FixPack 레벨(VRMF)(예: `9.5.0.0`)
- 적용할 수 있는 경우 FixPack. 예를 들어 FixPack 1이 설치되면 표시된 값은 1입니다. 여기에는 FixPack 1a와 같은 임시 FixPack이 포함됩니다.

• 제품의 VRMF 정보가 일치하지 않으면 경고 메시지가 출력 목록의 끝에 표시됩니다. 메시지에 적용할 FixPack이 제시됩니다.

db2pd 명령을 사용하여 모니터링 및 문제점 해결

db2pd 명령은 DB2 메모리 세트에서 신속하면서도 즉각적으로 정보를 리턴할 수 있기 때문에 문제점 해결에 사용됩니다.

개요

이 도구는 래치를 획득하거나 엔진 자원을 사용하지 않고 정보를 수집합니다. 그러므로 db2pd가 정보를 수집하는 동안 변경되고 있는 정보를 검색하는 것이 가능하며 예측됩니다. 따라서 데이터가 완전히 정확하지는 않을 수도 있습니다. 변경되는 메모리 포인터가 발생하면, db2pd가 비정상적으로 종료되지 않도록 신호 핸들러가 사용됩니다. 이로 인해 출력에 ″데이터 구조 변경으로 명령이 강제로 종료됨″과 같은 메시지가 표시될 수 있습니다. 그럼에도 불구하고 이 도구는 문제점 해결에 유용할 수 있습니다. 래칭 없이 정보를 수집할 경우의 두 가지 이점은 신속한 검색 및 엔진 자원 경합이 없다는 점입니다.

특정 SQLCODE, ZRC 코드 또는 ECF 코드가 발생할 때 데이터베이스 관리 시스템에 대한 정보를 캡처하려면, db2pdcfg -catch 명령을 사용하여 이를 수행할 수 있습니다. 오류가 발생한 경우, db2cos(콜아웃 스크립트)가 시작됩니다. db2cos 스크립트를 동적으로 변경하여 db2pd 명령, 운영 체제 명령 또는 문제점을 해결하는 데 필요한 기타 명령을 실행할 수 있습니다. 템플리트 db2cos 스크립트 파일은 UNIX 및 Linux의 경우 `sqllib/bin`에 있습니다. Windows 운영 체제의 경우, db2cos는 `$DB2PATH₩bin` 디렉토리에 있습니다.

새 노드를 추가하는 경우, db2pd -addnode 명령을 선택적 `oldviewapps` 및 `detail` 매개변수와 함께 사용하여 노드를 추가하고 있는 데이터베이스 파티션 서버에서 조작 진행을 모니터하여 자세한 정보를 얻을 수 있습니다.

현재 사용 중이거나 어떤한 이유로 비활성화된 이벤트 모니터 목록이 필요하면, db2pd -gfw 명령을 실행하십시오. 또한 이 명령은 각 고속 기록기 EDU에 대해 이벤트 모니터가 데이터를 기록하는 목표에 대한 정보와 통계를 리턴합니다.

예

다음은 db2pd 명령을 사용하여 문제점 해결을 촉진할 수 있는 예 콜렉션입니다.

- 예 1: 잠금 대기 진단
- 예 2: **-wlocks** 매개변수를 사용하여 대기 중인 잠금 모두 캡처
- 예 3: **-apinfo** 매개변수를 사용하여 잠금 소유자 및 잠금 대기자에 대한 자세한 런타임 정보 캡처
- 예 4: 잠금 문제점 고려 시 콜라웃 스크립트 사용
- 예 5: 응용프로그램을 동적 SQL문에 맵핑
- 예 6: 메모리 사용 모니터링
- 예 7: 테이블 스페이스를 다 사용하고 있는 응용프로그램 식별
- 예 8: 복구 모니터링
- 예 9: 트랜잭션이 사용 중인 자원 양 판별

- 예 10: 로그 사용 모니터링
- 예 11: Sysplex 목록 보기
- 예 12: 스택 추적 작성
- 예 13: 데이터베이스 파티션에 대한 메모리 통계 보기

예 1: 잠금 대기 진단

db2pd -db *databasename* -locks -transactions -applications -dynamic을 실행하는 경우, 결과는 다음과 유사합니다.

```
Locks:
Address            TranHdl Lockname                   Type Mode Sts Owner Dur HldCnt Att    ReleaseFlg
0x07800000202E5238 3       00020002000000040000000052 Row  ..X  G   3     1   0      0x0000 0x40000000
0x07800000202E4668 2       00020002000000040000000052 Row  ..X  W*  2     1   0      0x0000 0x40000000
```

-db 데이터베이스 이름 옵션을 사용하여 지정한 데이터베이스의 경우 첫 번째 결과로 해당 데이터베이스에 대한 잠금이 표시됩니다. 이 결과는 TranHdl 2가 TranHdl 3에서 보유한 잠금에 대해 대기 중임을 표시합니다.

```
Transactions:
Address            AppHandl [nod-index] TranHdl Locks State Tflag      Tflag2     Firstlsn       Lastlsn        LogSpace SpaceReserved TID            AxRegCnt GXID
0x0780000020251B80 11       [000-00011] 2       4     READ  0x00000000 0x00000000 0x000000000000 0x000000000000 0        0             0x0000000000B7 1        0
0x0780000020252900 12       [000-00012] 3       4     WRITE 0x00000000 0x00000000 0x000000FA000C 0x000000FA000C 113      154           0x0000000000B8 1        0
```

TranHdl 2는 AppHandl 11과 연관되고 TranHdl 3은 AppHandl 12와 연관됨을 알 수 있습니다.

```
Applications:
Address            AppHandl [nod-index] NumAgents CoorPid Status        C-AnchID C-StmtUID L-AnchID L-StmtUID Appid

0x07800000006879E0 12       [000-00012] 1         1073336 UOW-Waiting   0        0         17       1         *LOCAL.burford.060303225602
0x0780000000685E80 11       [000-00011] 1         1040570 UOW-Executing 17       1         94       1         *LOCAL.burford.060303225601
```

AppHandl 12는 동적문 17, 1을 마지막으로 실행했음을 알 수 있습니다. ApplHandl 11은 동적문 17, 1을 현재 실행 중이며 명령문 94, 1을 마지막으로 실행했습니다.

```
Dynamic SQL Statements:
Address            AnchID StmtUID NumEnv NumVar NumRef NumExe Text
0x07800000209FD800 17     1       1      1      2      2      update pdtest set c1 = 5
0x07800000209FCCC0 94     1       1      1      2      2      set lock mode to wait 1
```

텍스트 컬럼이 잠금 시간종료와 연관된 SQL문을 표시함을 알 수 있습니다.

예 2: **-wlocks** 매개변수를 사용하여 대기 중인 잠금 모두 캡처

db2pd -wlocks -db pdtest를 실행하는 경우, 다음과 유사한 결과가 생성됩니다. 첫 번째 응용프로그램(AppHandl 47)은 테이블에서 삽입을 수행 중이고 두 번째 응용프로그램(AppHandl 46)은 해당 테이블에서 선택을 수행 중임을 표시합니다.

```
venus@boson:/home/venus =>db2pd -wlocks -db pdtest

Database Partition 0 -- Database PDTEST -- Active -- Up 0 days 00:01:22

Locks being waited on :
AppHandl [nod-index] TranHdl    Lockname                   Type    Mode Conv Sts CoorEDU    AppName  AuthID   AppID
47       [000-00047] 8          00020004000000000840000652 Row     ..X       G   5160       db2bp    VENUS    *LOCAL.venus.071207213730
46       [000-00046] 2          00020004000000000840000652 Row     .NS       W   5913       db2bp    VENUS    *LOCAL.venus.071207213658
```

예 3: **-apinfo** 매개변수를 사용하여 잠금 소유자 및 잠금 대기자에 대한 자세한 런타임 정보 캡처

예 2와 동일한 조건에서 다음 샘플 출력이 생성되었습니다.

```
venus@boson:/home/venus =>db2pd -apinfo 47 -db pdtest

Database Partition 0 -- Database PDTEST -- Active -- Up 0 days 00:01:30

Application :
  Address :                0x0780000001676480
  AppHandl [nod-index] :   47       [000-00047]
  Application PID :        876558
  Application Node Name :  boson
  IP Address:              n/a
  Connection Start Time :  (1197063450)Fri Dec  7 16:37:30 2007
  Client User ID :         venus
  System Auth ID :         VENUS
  Coordinator EDU ID :     5160
  Coordinator Partition :  0
  Number of Agents :       1
  Locks timeout value :    4294967294 seconds
  Locks Escalation :       No
  Workload ID :            1
  Workload Occurrence ID : 2
  Trusted Context :        n/a
  Connection Trust Type :  non trusted
  Role Inherited :         n/a
  Application Status :     UOW-Waiting
  Application Name :       db2bp
  Application ID :         *LOCAL.venus.071207213730

  ClientUserID :           n/a
  ClientWrkstnName :       n/a
  ClientApplName :         n/a
  ClientAccntng :          n/a

  List of inactive statements of current UOW :
    UOW-ID :          2
    Activity ID :     1
    Package Schema :  NULLID
    Package Name :    SQLC2G13
    Package Version :
    Section Number :  203
    SQL Type :        Dynamic
    Isolation :       CS
    Statement Type :  DML, Insert/Update/Delete
    Statement :       insert into pdtest values 99


venus@boson:/home/venus =>db2pd -apinfo 46 -db pdtest

Database Partition 0 -- Database PDTEST -- Active -- Up 0 days 00:01:39

Application :
  Address :                0x0780000000D77A60
  AppHandl [nod-index] :   46       [000-00046]
  Application PID :        881102
  Application Node Name :  boson
  IP Address:              n/a
  Connection Start Time :  (1197063418)Fri Dec  7 16:36:58 2007
  Client User ID :         venus
```

```
  System Auth ID :          VENUS
  Coordinator EDU ID :      5913
  Coordinator Partition :   0
  Number of Agents :        1
  Locks timeout value :     4294967294 seconds
  Locks Escalation :        No
  Workload ID :             1
  Workload Occurrence ID :  1
  Trusted Context :         n/a
  Connection Trust Type :   non trusted
  Role Inherited :          n/a
  Application Status :      Lock-wait
  Application Name :        db2bp
  Application ID :          *LOCAL.venus.071207213658

  ClientUserID :            n/a
  ClientWrkstnName :        n/a
  ClientApplName :          n/a
  ClientAccntng :           n/a

  List of active statements :
   *UOW-ID :           3
    Activity ID :      1
    Package Schema :   NULLID
    Package Name :     SQLC2G13
    Package Version :
    Section Number :   201
    SQL Type :         Dynamic
    Isolation :        CS
    Statement Type :   DML, Select (blockable)
    Statement :        select * from pdtest
```

예 4: 잠금 문제점 고려 시 콜라웃 스크립트 사용

콜아웃 스크립트를 사용하려면 db2cos 출력 파일을 찾으십시오. 데이터베이스 관리 프로그램 구성 매개변수 **diagpath**가 파일 위치를 제어합니다. 출력 파일의 컨텐츠는 db2cos 스크립트 파일에 입력하는 명령에 따라 다릅니다. db2cos 스크립트 파일에 db2pd -db sample -locks 명령이 포함된 경우 제공된 출력 예는 다음과 같습니다.

```
Lock Timeout Caught
Thu Feb 17 01:40:04 EST 2006
Instance DB2
Datbase: SAMPLE
Partition Number: 0
PID: 940
TID: 2136
Function: sqlplnfd
Component: lock manager
Probe: 999
Timestamp: 2006-02-17-01.40.04.106000
AppID: *LOCAL.DB2...
AppHdl:
...
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:06:53
Locks:
Address    TranHdl Lockname                   Type Mode Sts Owner Dur HldCnt Att Rlse
0x402C6B30 3       000200030000000040000000052 Row  ..X  W*  3     1   0      0   0x40
```

출력에서 W*는 시간종료가 발생한 잠금을 표시합니다. 이 경우, 잠금 대기가 발생했습니다. 잠금이 상위 모드로 변환되는 경우에는 잠금 시간종료도 발생할 수 있습니다. 이는 출력에 C*로 표시됩니다.

결과를 트랜잭션, 응용프로그램, 에이전트 또는 db2cos 파일에 기타 db2pd 명령이 제공한 출력이 있는 SQL문에 맵핑할 수 있습니다. 출력 범위를 좁히거나 기타 명령을 사용하여 필요한 정보를 수집할 수 있습니다. 예를 들어, db2pd **-locks wait** 매개변수를 사용하여 활성화된 대기 상태인 잠금만 인쇄할 수 있습니다. 또한 **-app** 및 **-agent** 매개변수를 사용할 수 있습니다.

예 5: 응용프로그램을 동적 SQL문에 맵핑

db2pd -applications -dynamic 명령은 동적 SQL문에 대한 현재 및 마지막 앵커 ID와 명령문 고유 ID를 보고합니다. 따라서 응용프로그램에서 동적 SQL문으로의 직접 맵핑이 가능합니다.

```
Applications:
Address            AppHandl [nod-index] NumAgents  CoorPid  Status
0x00000002006D2120 780      [000-00780] 1          10615    UOW-Executing

C-AnchID C-StmtUID  L-AnchID L-StmtUID  Appid
163      1          110      1          *LOCAL.burford.050202200412

Dynamic SQL Statements:
Address            AnchID StmtUID NumEnv NumVar NumRef NumExe Text
0x0000000220A02760 163    1       2      2      2      1      CREATE VIEW MYVIEW
0x0000000220A0B460 110    1       2      2      2      1      CREATE VIEW YOURVIEW
```

예 6: 메모리 사용 모니터링

다음 샘플 출력에 표시된 대로 메모리 사용을 이해하려면 db2pd -memblock 명령이 유용할 수 있습니다.

```
All memory blocks in DBMS set.

Address            PoolID   PoolName   BlockAge   Size(Bytes) I LOC   File
0x0780000000740068 62       resynch    2          112         1 1746  1583816485
0x0780000000725688 62       resynch    1          108864      1 127   1599127346
0x07800000001F4348 57       ostrack    6          5160048     1 3047  698130716
0x07800000001B5608 57       ostrack    5          240048      1 3034  698130716
0x07800000001A0068 57       ostrack    1          80          1 2970  698130716
0x07800000001A00E8 57       ostrack    2          240         1 2983  698130716
0x07800000001A0208 57       ostrack    3          80          1 2999  698130716
0x07800000001A0288 57       ostrack    4          80          1 3009  698130716
0x0780000000700068 70       apmh       1          360         1 1024  3878879032
0x07800000007001E8 70       apmh       2          48          1 914   1937674139
0x0780000000700248 70       apmh       3          32          1 1000  1937674139
...
```

이 다음에는 정렬된 '풀별' 출력이 옵니다.

```
Memory blocks sorted by size for ostrack pool:
PoolID     PoolName   TotalSize(Bytes)     TotalCount LOC   File
57         ostrack    5160048              1          3047  698130716
57         ostrack    240048               1          3034  698130716
57         ostrack    240                  1          2983  698130716
57         ostrack    80                   1          2999  698130716
57         ostrack    80                   1          2970  698130716
```

```
57          ostrack    80                    1           3009  698130716
Total size for ostrack pool: 5400576 bytes

Memory blocks sorted by size for apmh pool:
PoolID      PoolName   TotalSize(Bytes)      TotalCount LOC   File
70          apmh       40200                 2          121   2986298236
70          apmh       10016                 1          308   1586829889
70          apmh       6096                  2          4014  1312473490
70          apmh       2516                  1          294   1586829889
70          apmh       496                   1          2192  1953793439
70          apmh       360                   1          1024  3878879032
70          apmh       176                   1          1608  1953793439
70          apmh       152                   1          2623  1583816485
70          apmh       48                    1          914   1937674139
70          apmh       32                    1          1000  1937674139
Total size for apmh pool: 60092 bytes
...
```

출력의 최종 섹션은 전체 메모리 세트에 대해 메모리 소비자를 정렬합니다.

```
All memory consumers in DBMS memory set:
PoolID      PoolName   TotalSize(Bytes)      %Bytes TotalCount %Count LOC   File
57          ostrack    5160048               71.90  1          0.07   3047  698130716
50          sqlch      778496                10.85  1          0.07   202   2576467555
50          sqlch      271784                3.79   1          0.07   260   2576467555
57          ostrack    240048                3.34   1          0.07   3034  698130716
50          sqlch      144464                2.01   1          0.07   217   2576467555
62          resynch    108864                1.52   1          0.07   127   1599127346
72          eduah      108048                1.51   1          0.07   174   4210081592
69          krcbh      73640                 1.03   5          0.36   547   4210081592
50          sqlch      43752                 0.61   1          0.07   274   2576467555
70          apmh       40200                 0.56   2          0.14   121   2986298236
69          krcbh      32992                 0.46   1          0.07   838   698130716
50          sqlch      31000                 0.43   31         2.20   633   3966224537
50          sqlch      25456                 0.35   31         2.20   930   3966224537
52          kerh       15376                 0.21   1          0.07   157   1193352763
50          sqlch      14697                 0.20   1          0.07   345   2576467555
...
```

UNIX 및 Linux 운영 체제에서는 전용 메모리에 대한 메모리 블록도 보고할 수 있습니다. 예를 들어, db2pd -memb pid=159770을 실행하는 경우 다음과 유사한 결과가 생성됩니다.

```
All memory blocks in Private set.

Address            PoolID     PoolName     BlockAge   Size(Bytes) I LOC   File
0x0000000110469068 88         private      1          2488        1 172   4283993058
0x0000000110469A48 88         private      2          1608        1 172   4283993058
0x000000011046A0A8 88         private      3          4928        1 172   4283993058
0x000000011046B408 88         private      4          7336        1 172   4283993058
0x000000011046D0C8 88         private      5          32          1 172   4283993058
0x000000011046D108 88         private      6          6728        1 172   4283993058
0x000000011046EB68 88         private      7          168         1 172   4283993058
0x000000011046EC28 88         private      8          24          1 172   4283993058
0x000000011046EC68 88         private      9          408         1 172   4283993058
0x000000011046EE28 88         private      10         1072        1 172   4283993058
0x000000011046F288 88         private      11         3464        1 172   4283993058
0x0000000110470028 88         private      12         80          1 172   4283993058
0x00000001104700A8 88         private      13         480         1 1534  862348285
0x00000001104702A8 88         private      14         480         1 1939  862348285
0x0000000110499FA8 88         private      80         65551       1 1779  4231792244
Total set size: 94847 bytes

Memory blocks sorted by size:
PoolID      PoolName   TotalSize(Bytes)      TotalCount LOC   File
88          private    65551                 1          1779  4231792244
```

```
88          private    28336                12          172   4283993058
88          private    480                  1           1939  862348285
88          private    480                  1           1534  862348285
Total set size: 94847 bytes
```

예 7: 테이블 스페이스를 다 사용하고 있는 응용프로그램 식별

db2pd -tcbstats를 사용하여 테이블에 대한 삽입 수를 식별할 수 있습니다. 다음은 TEMP1이라는 사용자 정의 전역 임시 테이블에 대한 샘플 정보입니다.

```
TCB Table Information:
Address            TbspaceID TableID PartID MasterTbs MasterTab TableName  SchemaNm ObjClass DataSize  LfSize LobSize XMLSize
0x0780000020B62AB0 3         2       n/a    3         2         TEMP1      SESSION  Temp     966       0      0       0

TCB Table Stats:
Address            TableName   Scans   UDI   PgReorgs   NoChgUpdts   Reads   FscrUpdates Inserts   Updates   Deletes   OvFlReads OvFlCrtes
0x0780000020B62AB0 TEMP1       0       0     0          0            0       0           43968     0         0         0         0
```

db2pd -tablespaces 명령을 사용하여 테이블 스페이스 3에 대한 정보를 얻을 수 있습니다. 샘플 출력은 다음과 같습니다.

```
Tablespace 3 Configuration:
Address            Type Content PageSz ExtentSz Auto Prefetch BufID BufIDDisk FSC NumCntrs MaxStripe  LastConsecPg Name
0x0780000020B1B5A0 DMS  UsrTmp  4096   32       Yes  32       1     1         On  1        0          31           TEMPSPACE2

Tablespace 3 Statistics:
Address            TotalPgs   UsablePgs  UsedPgs    PndFreePgs FreePgs    HWM        State      MinRecTime NQuiescers
0x0780000020B1B5A0 5000       4960       1088       0          3872       1088       0x00000000 0          0

Tablespace 3 Autoresize Statistics:
Address            AS  AR  InitSize    IncSize     IIP MaxSize     LastResize      LRF
0x0780000020B1B5A0 No  No  0           0           No  0           None            No

Containers:
Address            ContainNum Type    TotalPgs   UseablePgs StripeSet  Container
0x0780000020B1DCC0 0          File    5000       4960       0          /home/db2inst1/tempspace2a
```

FreePgs 컬럼은 스페이스를 채우고 있음을 표시합니다. 사용 가능한 페이지 값이 줄어들면 사용 가능한 스페이스도 줄어듭니다. 또한 FreePgs 값에 UsedPgs 값을 더하면 UsablePgs 값과 같음에 유의하십시오.

이를 알았으면 db2pd -db sample -dyn을 실행하여 TEMP1 테이블을 사용 중인 동적 SQL문을 식별할 수 있습니다.

```
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:13:06

Dynamic Cache:
Current Memory Used           1022197
Total Heap Size               1271398
Cache Overflow Flag           0
Number of References          237
Number of Statement Inserts   32
Number of Statement Deletes   13
Number of Variation Inserts   21
Number of Statements          19

Dynamic SQL Statements:
Address            AnchID StmtUID  NumEnv  NumVar  NumRef  NumExe  Text
0x0000000220A08C40 78     1        2       2       3       2       declare global temporary table temp1 (c1 char(6)) not logged
0x0000000220A8D960 253    1        1       1       24      24      insert into session.temp1 values('TEST')
```

마지막으로, db2pd -db sample -app를 실행하여 이전 출력의 정보를 응용프로그램 출력에 맵핑하여 응용프로그램을 식별할 수 있습니다.

```
Applications:
Address            AppHandl [nod-index] NumAgents  CoorPid Status
0x0000000200661840 501      [000-00501] 1          11246   UOW-Waiting

C-AnchID C-StmtUID  L-AnchID L-StmtUID  Appid
0        0          253      1          *LOCAL.db2inst1.050202160426
```

동적 SQL문을 식별한 앵커 ID(AnchID) 값을 사용하여 연관된 응용프로그램을 식별할 수 있습니다. 결과는 마지막 앵커 ID(L-AnchID) 값이 앵커 ID(AnchID) 값과 동일함을 표시합니다. 다음 번에 db2pd를 실행할 때 db2pd 실행 결과를 사용하십시오.

db2pd -agent 출력은 응용프로그램이 읽은 행 수를 Rowsread 컬럼에, 기록한 행 수를 Rowswrtn 컬럼에 표시합니다. 다음 샘플 출력에 표시된 대로 이러한 값을 통해 응용프로그램이 완료한 것과 응용프로그램이 여전히 완료해야 하는 것을 알 수 있습니다.

```
Address            AppHandl [nod-index] AgentPid  Priority  Type DBName
0x0000000200698080 501      [000-00501] 11246     0         Coord SAMPLE

State       ClientPid  Userid   ClientNm Rowsread   Rowswrtn   LkTmOt
Inst-Active 26377      db2inst1 db2bp    22         9588       NotSet
```

db2pd -agent 명령 실행으로 생기는 AppHandl 및 AgentPid 값을 db2pd -app 명령 실행으로 생기는 해당 AppHandl 및 CoorPiid 값에 맵핑할 수 있습니다.

내부 임시 테이블이 테이블 스페이스를 채우고 있다고 예측되는 경우에는 단계가 약간 다릅니다. 그러나 db2pd -tcbstats를 사용하여 삽입이 많은 테이블을 식별하십시오. 다음은 내재된 임시 테이블에 대한 샘플 정보입니다.

```
TCB Table Information:
Address            TbspaceID TableID PartID MasterTbs MasterTab TableName          SchemaNm ObjClass    DataSize  ...
0x0780000020CC0D30 1         2       n/a    1         2         TEMP (00001,00002) <30>     <JMC Temp   2470      ...
0x0780000020CC14B0 1         3       n/a    1         3         TEMP (00001,00003) <31>     <JMC Temp   2367      ...
0x0780000020CC21B0 1         4       n/a    1         4         TEMP (00001,00004) <30>     <JMC Temp   1872      ...

TCB Table Stats:
Address            TableName          Scans   UDI   PgReorgs   NoChgUpdts Reads   FscrUpdates Inserts ...
0x0780000020CC0D30 TEMP (00001,00002) 0       0     0          0          0       0           43219   ...
0x0780000020CC14B0 TEMP (00001,00003) 0       0     0          0          0       0           42485   ...
0x0780000020CC21B0 TEMP (00001,00004) 0       0     0          0          0       0           0       ...
```

이 예에서는 TEMP (TbspaceID, TableID) 이름 지정 규칙을 가진 테이블에 대한 많은 삽입이 있습니다. 이는 내재된 임시 테이블입니다. SchemaNm 컬럼 값에서는 AppHandl 값 이름 지정 규칙이 SchemaNm 값과 병합되므로 작업 수행 중인 응용프로그램을 식별할 수 있습니다.

그런 다음, 해당 정보를 db2pd -tablespaces 출력에 맵핑하여 테이블 스페이스 1에 사용한 스페이스를 확인할 수 있습니다. 다음 출력의 테이블 스페이스 통계에서 UsedPgs 값과 UsablePgs 값 간의 관계에 주의하십시오.

```
Tablespace Configuration:
Address            Id    Type Content PageSz ExtentSz Auto Prefetch BufID BufIDDisk FSC NumCntrs MaxStripe LastConsecPg Name
0x07800000203FB5A0 1     SMS  SysTmp  4096   32       Yes  320      1     1         On  10       0         31           TEMPSPACE1

Tablespace Statistics:
Address            Id    TotalPgs   UsablePgs  UsedPgs    PndFreePgs FreePgs    HWM    State      MinRecTime NQuiescers
0x07800000203FB5A0 1     6516       6516       6516       0          0          0      0x00000000 0          0

Tablespace Autoresize Statistics:
Address            Id    AS  AR  InitSize    IncSize     IIP MaxSize     LastResize  LRF
0x07800000203FB5A0 1     No  No  0           0           No  0           None        No

Containers:
...
```

-tcbstats 출력에서 응용프로그램 핸들 30 및 31을 보았으므로 db2pd -app 명령을 사용하여 이를 식별할 수 있습니다.

```
Applications:
Address            AppHandl [nod-index] NumAgents  CoorPid    Status         C-AnchID C-StmtUID  L-AnchID   L-StmtUID  Appid
0x07800000006FB880 31       [000-00031] 1          4784182    UOW-Waiting    0        0          107        1          *LOCAL.db2inst1.051215214142
0x07800000006F9CE0 30       [000-00030] 1          8966270    UOW-Executing  107      1          107        1          *LOCAL.db2inst1.051215214013
```

마지막으로, 이전 출력의 정보를 db2pd -dyn 명령을 실행하여 확보한 동적 SQL 출력에 맵핑할 수 있습니다.

```
Dynamic SQL Statements:
Address             AnchID StmtUID    NumEnv     NumVar     NumRef     NumExe     Text
0x0780000020B296C0 107    1          1          1          43         43         select c1, c2 from test group by c1,c2
```

예 8: 복구 모니터링

db2pd -recovery 명령을 실행하는 경우, 다음 샘플 출력에 표시된 대로 출력은 복구가 진행 중임을 검증하는 데 사용할 수 있는 몇 개의 카운터를 표시합니다. Current Log 및 Current LSN 값은 로그 위치 지정을 제공합니다. CompletedWork 값은 지금까지 완료된 바이트 수입니다.

```
Recovery:
Recovery Status     0x00000401
Current Log         S0000005.LOG
Current LSN         000002551BEA
Job Type            ROLLFORWARD RECOVERY
Job ID              7
Job Start Time      (1107380474) Wed Feb  2 16:41:14 2005
Job Description     Database Rollforward Recovery
Invoker Type        User
Total Phases        2
Current Phase       1

Progress:
Address            PhaseNum Description StartTime                CompletedWork  TotalWork
0x0000000200667160 1        Forward     Wed Feb  2 16:41:14 2005 2268098 bytes  Unknown
0x0000000200667258 2        Backward    NotStarted                0 bytes        Unknown
```

예 9: 트랜잭션이 사용 중인 자원 양 판별

db2pd -transactions 명령을 실행하는 경우, 다음 샘플 출력에 표시된 대로 출력은 잠금 수, 첫 번째 로그 시퀀스 번호(LSN), 마지막 LSN, 사용한 로그 스페이스, 예약된 스페이스를 표시합니다. 이는 트랜잭션 동작을 이해하는 데 유용할 수 있습니다.

```
Transactions:
Address            AppHandl [nod-index] TranHdl  Locks  State  Tflag
0x0000000022026D980 797      [000-00797] 2        108    WRITE  0x00000000
0x0000000022026E600 806      [000-00806] 3        157    WRITE  0x00000000
0x0000000022026F280 807      [000-00807] 4        90     WRITE  0x00000000

Tflag2     Firstlsn       Lastlsn        LogSpace   SpaceReserved
0x00000000 0x000001072262 0x0000010B2C8C 4518       95450
0x00000000 0x000001057574 0x0000010B3340 6576       139670
0x00000000 0x00000107CF0C 0x0000010B2FDE 3762       79266

TID            AxRegCnt   GXID
0x000000000451 1          0
0x0000000003E0 1          0
0x000000000472 1          0
```

예 10: 로그 사용 모니터링

db2pd -logs 명령은 데이터베이스에 대한 로그 사용 모니터링에 유용합니다. 다음 샘플 출력에 표시된 대로 Pages Written 값을 사용하여 로그 사용이 증가하고 있는지 여부를 판별할 수 있습니다.

```
Logs:
Current Log Number            2
Pages Written                 846
Method 1 Archive Status       Success
Method 1 Next Log to Archive  2
Method 1 First Failure        n/a
Method 2 Archive Status       Success
Method 2 Next Log to Archive  2
Method 2 First Failure        n/a

Address            StartLSN       State      Size  Pages  Filename
0x0000000023001BF58 0x000001B58000 0x00000000 1000  1000   S0000002.LOG
0x0000000023001BE98 0x000001F40000 0x00000000 1000  1000   S0000003.LOG
0x0000000230008F58 0x000002328000 0x00000000 1000  1000   S0000004.LOG
```

이 출력을 사용하여 다음 두 가지 유형의 문제점을 식별할 수 있습니다.

- 가장 최근의 로그 아카이브가 실패하면, Archive Status가 Failure 값으로 설정됩니다. 진행 중인 아카이브 실패가 있고 로그가 전혀 아카이브되지 않는다면, Archive Status가 First Failure 값으로 설정됩니다.
- 로그 아카이브가 아주 느린 속도로 진행되는 경우, Next Log to Archive 값이 Current Log Number 값보다 작습니다. 아카이브 속도가 아주 느린 경우, 사용 중인 로그 스페이스가 부족할 수 있으며 이로 인해 데이터 변경이 데이터베이스에서 발생하지 못할 수 있습니다.

예 11: Sysplex 목록 보기

db2pd -sysplex 명령이 다음 샘플 출력을 표시하지 않는 경우, sysplex 목록을 보고하는 다른 유일한 방법은 DB2 추적을 사용하는 것입니다.

```
Sysplex List:
Alias:         HOST
Location Name: HOST1
Count:         1

IP Address      Port       Priority   Connections Status     PRDID
 1.2.34.56      400        1          0           0
```

예 12: 스택 추적 작성

Windows 운영 체제의 경우 db2pd -stack all 명령을, 또는 UNIX 운영 체제의 경우 -stack 명령을 사용하여 현재 데이터베이스 파티션의 모든 프로세스에 대한 스택 추적을 생성할 수 있습니다. 프로세스 또는 스레드가 루프 또는 정지 중이라고 예측되는 경우에는 이 명령을 반복적으로 사용할 수 있습니다.

다음 예에 표시된 대로 db2pd -stack *eduid* 명령을 실행하여 특정 엔진 디스패치 가능 단위(EDU)에 대한 현재 모든 스택을 확보할 수 있습니다.

```
Attempting to dump stack trace for eduid 137.
See current DIAGPATH for trapfile.
```

예를 들어, 모든 DB2 프로세스에 대한 호출 스택을 원할 경우 Windows 운영 체제에서는 db2pd -stack all 명령을 사용하십시오.

```
Attempting to dump all stack traces for instance.
See current DIAGPATH for trapfiles.
```

실제 노드가 여러 개인 파티션된 데이터베이스 환경을 사용 중인 경우, db2_all ″; db2pd -stack all″ 명령을 사용하여 모든 파티션에서 정보를 얻을 수 있습니다. 그러나 파티션이 동일한 시스템에서 모두 논리적 파티션인 경우, 보다 빠른 방법은 db2pd -alldbp -stacks를 사용하는 것입니다.

예 13: 데이터베이스 파티션에 대한 메모리 통계 보기

db2pd -dbptnmem 명령은 DB2 서버가 현재 사용 중인 메모리 양과 상위 레벨에서 해당 메모리를 사용 중인 서버 영역을 표시합니다.

다음은 AIX 시스템에서 db2pd -dbptnmem 실행으로 인한 출력 예입니다.

```
Database Partition Memory Controller Statistics

Controller Automatic: Y
Memory Limit:   122931408KB
Current usage:     651008KB
HWM usage:         651008KB
Cached memory:     231296KB
```

이러한 데이터 필드 및 컬럼에 대한 설명은 다음과 같습니다.

Controller Automatic
: 메모리 제어기 설정을 표시합니다. **instance_memory** 구성 매개변수가 AUTOMATIC으로 설정된 경우 ″Y″ 값을 표시합니다. 이는 데이터베이스 관리 프로그램이 메모리 소비 상한을 자동으로 판별함을 의미합니다.

Memory Limit
: 소비할 수 있는 DB2 서버 메모리의 상한입니다. **instance_memory** 구성 매개변수 값입니다.

Current usage
: 서버가 현재 소비하고 있는 메모리의 양입니다.

HWM usage
: db2start 명령이 실행될 때 데이터베이스 파티션이 활성화된 이후로 소비된 상위 워터 마크(water mark)(HWM) 또는 최대 메모리 사용량입니다.

Cached memory
: 현재 사용되고 있지 않지만 향후 메모리 요청을 위해 성능상의 이유로 캐시된 현재 사용 양입니다.

다음은 AIX 운영 체제에서 db2pd -dbptnmem 실행으로 인한 샘플 출력의 연속입니다.

```
Individual Memory Consumers:
Name           Mem Used (KB)   HWM Used (KB)   Cached (KB)
============================================================
APPL-DBONE       160000          160000        159616
DBMS-name         38528           38528          3776
FMP_RESOURCES     22528           22528             0
PRIVATE           13120           13120           740
FCM_RESOURCES     10048           10048             0
LCL-p606416         128             128             0
DB-DBONE         406656          406656         67200
```

DB2 서버 내 등록된 모든 메모리『소비자』가 소비 중인 전체 메모리 양과 함께 나열됩니다. 컬럼 설명은 다음과 같습니다.

Name 메모리 소비자의 짧은 식별 이름. 예를 들면, 다음과 같습니다.

APPL-*dbname*
dbname 데이터베이스용으로 소비된 응용프로그램 메모리

DBMS-*name*
전역 데이터베이스 관리 프로그램 메모리 요구사항

FMP_RESOURCES
db2fmps와 통신하는 데 필요한 메모리

PRIVATE
기타 전용 메모리 요구사항

FCM_RESOURCES
FCM(Fast Communication Manager) 자원

LCL-*pid*
로컬 응용프로그램과 통신하는 데 사용된 메모리 세그먼트

DB-*dbname*
dbname 데이터베이스용으로 소비된 데이터베이스 메모리

Mem Used (KB)
소비자에게 현재 할당된 메모리 양

HWM Used (KB)
소비자가 사용한 메모리의 상위 워터 마크(HWM) 또는 최대 메모리

Cached (KB)
Mem Used (KB) 중에서, 현재 사용되고 있지 않지만 향후 메모리 할당에 즉시 사용할 수 있는 메모리 양

db2support 명령을 사용하여 환경 정보 수집

DB2 문제점에 대한 정보를 수집할 때 실행해야 하는 가장 중요한 DB2 유틸리티는 db2support입니다. db2support 유틸리티는 사용 가능한 모든 DB2 및 시스템 진단 정보를 자동으로 수집합니다. 이 유틸리티에는 문제점 상황에 대한 질문을 제기하는 선택적 대화식 "질문과 응답" 세션도 있습니다.

db2support 유틸리티를 사용하면 GET DATABASE CONFIGURATION FOR *database-name* 또는 LIST TABLESPACES SHOW DETAIL과 같은 명령을 수동으로 입력하지 않아도 되기 때문에 가능한 사용자 오류를 방지할 수 있습니다. 또한 실행할 명령 또는 수집할 파일에 대한 지시사항이 필요하지 않으므로 데이터 수집에 소요되는 시간이 줄어듭니다.

- db2support -h 명령을 실행하여 전체 명령 옵션 목록을 표시하십시오.
- 해당 db2support 명령을 사용하여 데이터를 수집하십시오.

  SYSADM 권한을 가진 사용자(예: 인스턴스 소유자)가 db2support 유틸리티를 실행해야 유틸리티가 오류 없이 모든 필수 정보를 수집할 수 있습니다. SYSADM 권한이 없는 사용자가 db2support를 실행하는 경우, 유틸리티가 QUERY CLIENT 또는 LIST ACTIVE DATABASES와 같은 명령을 실행할 때 SQL 오류(예: SQL1092N)가 발생할 수 있습니다.

  IBM Software Support에 정보를 제공하기 위해 db2support 유틸리티를 사용 중인 경우, 시스템에 문제점이 발생하는 동안 db2support 명령을 실행하십시오. 이런 방법으로 이 도구는 적시의 정보(예: 운영 체제 성능 세부사항)를 수집합니다. 문제점이 발생한 당시에 유틸리티를 실행할 수 없는 경우, 일부 FODC(First Occurrence Data Capture) 진단 파일이 자동으로 생성되기 때문에 문제점이 중지된 후에 db2support 명령을 실행할 수 있습니다.

  일반적으로 다음 기본 호출은 문제점을 디버그하는 데 필요한 대부분의 정보를 수집하기에 충분합니다(-c 옵션을 사용하는 경우, 유틸리티는 데이터베이스 연결을 설정함에 유의).

  ```
  db2support <output path> -d <database name> -c
  ```

  출력은 편리하게 수집되고 압축된 ZIP 아카이브(db2support.zip)로 저장되므로 모든 시스템에서 쉽게 전송 및 추출할 수 있습니다.

db2support가 캡처하는 정보 유형은 명령 호출 방법, 데이터베이스 관리 프로그램이 시작되었는지 여부, 데이터베이스 연결이 가능한지 여부에 따라 다릅니다.

db2support 유틸리티는 모든 조건에서 다음 정보를 수집합니다.

- db2diag 로그 파일
- 모든 트랩 파일

- 잠금 목록 파일
- 덤프 파일
- 여러 시스템 관련 파일
- 여러 시스템 명령의 출력
- db2cli.ini

상황에 따라 db2support 유틸리티는 다음 정보도 수집할 수 있습니다.

- 사용 중인 로그 파일
- 버퍼 풀 및 테이블 스페이스(SQLSPCS.1 및 SQLSPCS.2) 제어 파일(-d 옵션 사용)
- db2dump 디렉토리 컨텐츠
- 확장 시스템 정보(-s 옵션 사용)
- 데이터베이스 구성 설정(-d 옵션 사용)
- 데이터베이스 관리 프로그램 구성 설정 파일
- 로그 파일 헤더 파일(-d 옵션 사용)
- 복구 실행기록 파일(-d 옵션 사용)

HTML 보고서(db2support.html)에는 항상 다음 정보가 포함됩니다.

- 문제점 레코드(PMR) 번호(-n이 지정된 경우)
- 운영 체제 및 레벨(예: AIX 5.1)
- DB2 릴리스 정보
- 32비트 환경인지 64비트 환경인지 여부에 대한 표시
- DB2 설치 경로 정보
- db2nodes.cfg의 컨텐츠
- CPU 및 디스크 수와 메모리 양
- 인스턴스의 데이터베이스 목록
- 레지스트리 정보 및 환경(PATH 및 LIBPATH 포함)
- UNIX용 현재 파일 시스템 및 노드에 대한 디스크 여유 공간
- Java™ SDK 레벨
- 데이터베이스 관리 프로그램 구성
- 데이터베이스 복구 실행기록 파일 목록
- sqllib 디렉토리의 ls -lR 출력(또는 Windows 동등값)
- LIST NODE DIRECTORY 명령 결과
- LIST ADMIN NODE DIRECTORY 명령 결과
- LIST DCS DIRECTORY 명령 결과
- LIST DCS APPLICATIONS EXTENDED 명령 결과

- 설치된 모든 소프트웨어 목록

-s 옵션이 지정된 경우 db2support.html 파일에 다음 정보가 표시됩니다.

- 자세한 디스크 정보(파티션 레이아웃, LVM 정보 등)
- 자세한 네트워크 정보
- 커널 통계
- 펌웨어 버전
- 기타 운영 체제별 명령

DB2가 시작된 경우 db2support.html 파일에는 다음 추가 정보가 포함됩니다.

- 클라이언트 연결 상태
- 데이터베이스 및 데이터베이스 관리 프로그램 구성(데이터베이스 구성에는 -d 옵션이 필요함)
- CLI 구성
- 메모리 풀 정보(*크기* 및 소모량). -d 옵션을 사용하는 경우에는 전체 데이터가 수집됩니다.
- LIST ACTIVE DATABASES 명령 결과
- LIST DCS APPLICATIONS 명령 결과

-c 옵션을 지정하고 데이터베이스 연결이 설정된 경우 db2support.html 파일에는 다음 정보가 포함됩니다.

- 사용자 테이블 수
- 데이터베이스 데이터의 대략적인 *크기*
- 데이터베이스 스냅샷
- 응용프로그램 스냅샷
- 버퍼 풀 정보
- LIST APPLICATIONS 명령 결과
- LIST COMMAND OPTIONS 명령 결과
- LIST DATABASE DIRECTORY 명령 결과
- LIST INDOUBT TRANSACTIONS 명령 결과
- LIST DATABASE PARTITION GROUPS 명령 결과
- LIST DBPARTITIONNUMS 명령 결과
- LIST ODBC DATA SOURCES 명령 결과
- LIST PACKAGES/TABLES 명령 결과
- LIST TABLESPACE CONTAINERS 명령 결과
- LIST TABLESPACES 명령 결과

- LIST DRDA IN DOUBT TRANSACTIONS 명령 결과
- DB2 워크로드 관리 프로그램 정보

db2support.zip 파일 컨텐츠 예

db2support.zip 파일 컨텐츠 예의 경우, 실행된 명령은 다음과 같습니다.

```
db2support . -d sample -c -f -st "select * from staff"
```

db2support.zip 파일 추출의 경우, 수집된 파일과 디렉토리는 다음과 같습니다.

- DB2CONFIG/ - 구성 정보(예: 데이터베이스, 데이터베이스 관리 프로그램, BP, CLI 및 Java 개발자 킷)
- DB2DUMP/ - ~/sqllib/db2dump 디렉토리 컨텐츠
- DB2MISC/ - sqllib 디렉토리 목록
- DB2SNAP/ - DB2 명령의 출력(예: db2set, LIST TABLES, LIST INDOUBT TRANSACTIONS 및 LIST APPLICATIONS)
- EVENTS/ - db2optstats.#.log 파일에 로그된 DB2 이벤트
- STMM/ - ~/sqllib/db2dump/stmmlog 디렉토리 컨텐츠(STMM 로그 파일)
- db2supp_opt.zip - 옵티마이저 문제점에 대한 진단 정보
- db2supp_system.zip - 운영 체제 정보
- db2support.html - HTML 섹션으로 형식화된 진단 정보

옵티마이저에 대한 정보는 db2supp_opt.zip 파일에 있습니다. 이 파일을 추출하면 다음 디렉토리가 나타납니다.

- OPTIMIZER/ - 옵티마이저 문제점에 대한 진단 정보
- OPTIMIZER/optimizer.log - 파일에는 모든 활동의 로그가 포함됨
- OPTIMIZER/CATALOGS - 다음 서브디렉토리에 LOB가 있는 모든 카탈로그
  - FUNCTIONS
  - ROUTINES
  - SEQUENCES
  - TABLES
  - VIEWS
- OPTIMIZER/DB2DUMP - db2serv 출력(serv.* 및 serv2.* 출력 파일)

시스템 정보는 db2supp_system.zip 파일에 있습니다. 이 파일을 추출하면 다음 파일과 디렉토리가 나타납니다.

- DB2CONFIG/ - db2cli.ini(~/sqllib/cfg의 파일)
- DB2DUMP/ - 트랩 및 진단 로그 파일과 FODC 디렉토리
- DB2MISC/ - 특히, DB2SYSTM 파일(2진)

- OSCONFIG/ - 서로 다른 운영 체제 정보 파일(예: 특히 netstat, services, vfs, ulimit 및 hosts)
- OSSNAP/ - 운영 체제 스냅샷(예: 특히 iostat, netstat, uptime, vmstat 및 ps_elf)
- SQLDBDIR/ - 중요한 버퍼 풀 메타 파일(~/sqllib/sqldbdir)
- SQLGWDIR/ - DCS 디렉토리(~/sqllib/sqlgwdir의 파일)
- SQLNODIR/ - 노드 디렉토리(~/sqllib/sqlnodir의 파일)
- SPMLOG/ - ~/sqllib/spmlog의 파일
- report.log - 모든 콜렉션 활동의 로그

기본 추적 진단

DB2에 반복적이면서 재현 가능한 문제점이 발생하는 경우, 추적을 사용하여 이에 대한 추가 정보를 캡처할 수 있는 경우가 있습니다. 일반적인 상황에서는 IBM Software Suppor에서 요구할 경우에만 추적을 사용해야 합니다. 추적 프로세스는 추적 기능 설정, 오류 재현, 데이터 수집을 수반합니다.

추적으로 수집된 정보의 양은 급속도로 증가합니다. 추적을 실시하면, 오류 상황만 캡처하고 가능하면 기타 활동은 피하십시오. 추적을 실시하면, 가장 작은 시나리오를 사용하여 문제점을 재현하십시오.

추적 수집은 DB2 인스턴스의 성능을 저하시키는 경우가 많습니다. 성능 저하 정도는 성능 유형과 추적 정보를 수집하는 데 사용되는 자원 수에 따라 다릅니다.

IBM Software Support는 추적을 요청할 때 다음 정보를 제공해야 합니다.

- 단순한 단계별 프로시저
- 각 추적을 실시할 위치에 대한 설명
- 추적할 사항에 대한 설명
- 추적을 요청하는 이유에 대한 설명
- 백아웃 프로시저(예: 추적을 모두 사용 불가능하게 하는 방법)

확보할 추적과 관련하여 IBM Software Support의 자문을 받아야 하지만, 특정 추적을 확보해야 하는 시기와 관련하여 몇 가지 일반적인 지침은 다음과 같습니다.

- 설치 중에 문제점이 발생하고 문제점의 원인을 판별하기에 디폴트 설치 로그가 부족한 경우, 설치 추적이 적합합니다.
- GUI(Graphical User Interface) 중 하나에서 문제점이 발생하고 DB2 명령 창에서 명시적 명령을 통해 수행하면 동일한 조치가 성공하는 경우, 제어 센터 추적이 적합합니다. 이는 제어 센터에서 시작할 수 있는 도구를 사용해서만 문제점을 캡처함에 유의하십시오.

- 문제점이 CLI 응용프로그램에서 나타나고 응용프로그램 밖에서는 문제점을 재현할 수 없는 경우, CLI 추적이 적합합니다.
- 문제점이 JDBC 응용프로그램에서 나타나고 응용프로그램 밖에서는 문제점을 재현할 수 없는 경우, JDBC 추적이 적합합니다.
- 문제점이 DRDA 계층에서 통신하는 정보와 직접 관련되어 있는 경우, DRDA 추적이 적합합니다.
- 추적을 실행할 수 있는 기타 모든 상황의 경우, DB2 추적이 적합할 가능성이 높습니다.

추적 정보가 오류 진단 시 항상 유용한 것은 아닙니다. 예를 들어, 다음 상황에서는 오류 조건을 캡처하지 못할 수 있습니다.

- 지정한 추적 버퍼 크기가 전체 추적 이벤트 세트를 수용할 만큼 크지 않았으며, 추적이 파일에 쓰기를 중지했거나 래핑된 경우 유용한 추적 정보가 유실되었습니다.
- 추적된 시나리오가 오류 상황을 재현하지 않았습니다.
- 오류 상황이 재현되었지만 문제점 발생 위치에 관한 가정이 올바르지 않았습니다. 예를 들어, 추적은 클라이언트 워크스테이션에서 수집된 반면, 실제 오류는 서버에서 발생했습니다.

DB2 추적

db2trc를 사용하여 DB2 추적 확보

db2trc 명령은 DB2와 함께 제공되는 추적 기능을 제어합니다. 추적 기능은 DB2 조작에 대한 정보를 기록하고 이 정보를 읽을 수 있는 양식으로 형식화합니다.

추적 실행 중에는 오버헤드가 추가되므로 추적 기능을 사용하면 시스템 성능에 영향을 줄 수 있음을 기억하십시오.

일반적으로, IBM Software Support 및 개발 팀에서는 문제점 해결을 위해 DB2 추적을 사용합니다. 조사 중인 문제점에 대한 정보를 얻기 위해 추적을 실행할 수 있지만 DB2 소스 코드에 대한 지식이 없으면 사용은 다소 제한됩니다.

그럼에도 불구하고 추적 파일 확보 요청에 대비하여 올바로 추적을 작동하는 방법과 추적 파일 덤프 방법을 알아두는 것이 중요합니다.

주: db2trc를 사용하려면 SYSADM, SYSCTRL 또는 SYSMAINT 권한 중 하나가 필요합니다.

사용 가능한 옵션에 대한 일반적인 개념을 얻으려면, 매개변수 없이 db2trc 명령을 실행하십시오.

```
C:₩>db2trc
Usage: db2trc (chg|clr|dmp|flw|fmt|inf|off|on) options
```

특정 db2trc 명령 매개변수에 대한 자세한 정보를 보려면 -u 옵션을 사용하십시오. 예를 들어, 추적 작동에 대한 자세한 정보를 보려면 다음 명령을 실행하십시오.

```
db2trc on -u
```

이렇게 하면 DB2 추적을 작동할 때 지정할 수 있는 모든 추가 옵션("기능"으로 레이블됨)에 대한 정보가 제공됩니다.

추적을 작동할 때 가장 중요한 옵션은 -L입니다. 이 옵션은 추적 중인 정보를 저장하는 데 사용될 메모리 버퍼의 *크기*를 지정합니다. 바이트 또는 MB 단위로 버퍼 *크기*를 지정할 수 있습니다. (MB를 지정하려면 값 뒤에 "M" 또는 "m"을 추가하십시오.) 추적 버퍼 *크기*는 2MB여야 합니다. 이 요구사항을 충족시키지 못하는 *크기*를 지정하면, 버퍼 *크기*가 2의 거듭제곱에 가장 가깝게 자동으로 반내림됩니다.

버퍼가 너무 작으면, 정보가 유실될 수 있습니다. 디폴트로, 버퍼가 가득차면 가장 최근의 추적 정보만 보존됩니다. 버퍼가 너무 크면, IBM Software Support로 파일을 보내기 어려울 수 있습니다.

비교적 짧은 조작(예: 데이터베이스 연결)을 추적하는 경우, *크기*가 약 8MB면 일반적으로 충분합니다.

```
C:₩> db2trc on -l 8M
Trace is turned on
```

그러나 대형 조작을 추적하거나 많은 작업이 동시에 진행되는 경우에는 대형 추적 버퍼가 필요할 수 있습니다.

대부분의 플랫폼에서 추적은 언제든지 설정할 수 있으며 앞서 설명한 대로 작동합니다. 그러나 알아 두어야 할 다음과 같은 특정 상황이 있습니다.

1. 다중 데이터베이스 파티션 시스템에서는 논리적 데이터베이스 파티션과 반대로 각 실제 데이터베이스 파티션에 대해 추적을 실행해야 합니다.
2. HP-UX, Linux 및 Solaris 플랫폼에서는 인스턴스가 시작된 후에 추적이 작동 해제된 경우 지정된 *크기*에 관계 없이 다음 번에 추적이 시작될 때 아주 작은 버퍼가 사용됩니다. 예를 들어, 어제 db2trc on -l 8m을 사용하여 추적을 작동하고 추적을 수집한 후 추적을 작동 해제했습니다(db2trc off). 오늘, 인스턴스를 줄인 후 재시작하지 않고 메모리 버퍼를 32MB로 설정하여 추적을 실행하려고 합니다(db2trc on -l 32m). 이 경우 추적은 작은 버퍼만 가져옴을 알 수 있습니다. 이러한 플랫폼에서 추적을 효과적으로 실행하려면, 필요한 버퍼 *크기*로 인스턴스를 시작하기 전에 추적을 작동하고 나중에 필요에 따라 버퍼를 『지우십시오』.

DB2 추적 파일 덤프

ON 옵션을 사용하여 추적 기능을 사용할 수 있게 한 후에는 인스턴스가 완료한 모든 후속 작업이 추적됩니다.

추적이 실행되는 동안 clr 옵션을 사용하여 추적 버퍼를 지울 수 있습니다. 추적 버퍼의 모든 기존 정보가 제거됩니다.

```
C:₩>db2trc clr
Trace has been cleared
```

추적 조작이 완료되었으면, dmp 옵션 다음에 추적 파일 이름을 사용하여 디스크에 메모리 버퍼를 덤프하십시오. 예를 들어, 다음과 같습니다.

```
C:₩>db2trc dmp trace.dmp
Trace has been dumped to file
```

디스크에 추적 버퍼를 덤프한 후에 추적 기능이 계속 실행됩니다. 추적을 작동 해제하려면, 다음과 같이 OFF 옵션을 사용하십시오.

```
C:₩>db2trc off
Trace is turned off
```

DB2 추적 파일 포맷팅

db2trc dmp 명령이 작성한 덤프 파일은 2진 형식으로 되어 있으며 읽을 수 없습니다. 추적 파일을 읽을 수 있는지 검증하려면 2진 추적 파일을 형식화하여 순서 제어를 표시하고 형식화된 출력을 널(NULL) 디바이스로 보내십시오.

다음 예는 이 태스크를 수행하기 위한 명령을 표시합니다.

```
db2trc flw example.trc nul
```

여기서 example.trc는 dmp 옵션을 사용하여 생성된 2진 파일입니다.

이 명령의 출력은 파일을 읽는 데 문제점이 있는지 여부와 추적이 래핑되었는지 여부를 명시적으로 알려줍니다.

이 시점에서 덤프 파일을 IBM Software Support로 보낼 수 있습니다. 그러면 IBM Software Support에서 DB2 서비스를 기반으로 파일을 형식화합니다. 그러나 덤프 파일을 보내기 전에 파일을 ASCII로 형식화할 것인지 묻는 경우가 있습니다. 이는 flw 및 fmt 옵션을 통해 수행됩니다. 작성할 ASCII 파일의 이름과 함께 2진 덤프 파일의 이름을 제공해야 합니다.

```
C:₩>db2trc flw trace.dmp trace.flw
C:₩Temp>db2trc flw trace.dmp trace.flw
Total number of trace records     : 18854
Trace truncated                   : NO
Trace wrapped                     : NO
Number of trace records formatted : 1513 (pid: 2196 tid 2148 node: -1)
Number of trace records formatted : 100 (pid: 1568 tid 1304 node: 0)
...

C:₩>db2trc fmt trace.dmp trace.fmt
C:₩Temp>db2trc fmt trace.dmp trace.fmt
Trace truncated                   : NO
```

```
Trace wrapped                      : NO
Total number of trace records      : 18854
Number of trace records formatted : 18854
```

출력이 "Trace wrapped"가 "YES"임을 표시하는 경우, 이는 추적 기간 동안 수집된 모든 정보를 포함할 수 있을 만큼 추적 버퍼가 크지 않았음을 의미합니다. 래핑된 추적은 상황에 따라 적절할 수 있습니다. 최신 정보(-i 옵션을 지정하지 않으면 이는 유지보수되는 디폴트 정보임)에 관심이 있으면, 추적 파일의 내용이 충분할 수 있습니다. 그러나 추적 기간 처음에 발생한 것에 관심이 있거나 발생한 모든 것에 관심이 있으면, 대형 추적 버퍼를 사용하여 조작을 재실행할 수 있습니다.

2진 파일을 읽을 수 있는 텍스트 파일로 포맷팅하는 경우 사용 가능한 옵션이 있습니다. 예를 들어, `db2trc fmt -xml trace.dmp trace.fmt`를 사용하여 2진 데이터를 변환하고 결과를 XML 구문 분석 가능한 형식으로 출력할 수 있습니다. 추가 옵션은 추적 명령(db2trc)에 대한 자세한 설명에 표시되어 있습니다.

알아야 할 다른 사항은 Linux 및 UNIX 운영 체제에서는 DB2가 심각한 오류로 인해 인스턴스를 종료하는 경우 추적 버퍼를 디스크에 자동으로 덤프한다는 것입니다. 따라서 인스턴스가 비정상적으로 종료될 때 추적을 사용할 수 있으면 진단 디렉토리에 파일이 작성되며 파일 이름은 `db2trdmp.###`입니다(여기서 ###는 데이터베이스 파티션 번호임). Windows 플랫폼에서는 이러한 상황이 발생하지 않습니다. 해당 상황에서는 추적을 수동으로 덤프해야 합니다.

요약하면, 다음은 db2trc 명령의 일반 시퀀스 예입니다.

```
db2trc on -l 8M
db2trc clr
<Execute problem recreation commands>
db2trc dump db2trc.dmp
db2trc off
db2trc flw db2trc.dmp <filename>.flw
db2trc fmt db2trc.dmp <filename>.fmt
db2trc fmt -c db2trc.dmp <filename>.fmtc
```

DRDA 추적 파일

DRDA 추적을 분석하기 전에 DRDA가 통신 구조 및 데이터 정의에 대한 개방형 표준임을 이해해야 합니다. 예를 들어, DRDA는 전송을 위한 데이터 구성 방법과 해당 정보 통신 발생 방법에 대한 규칙 세트를 포함합니다.

이러한 규칙이 정의된 참조 매뉴얼은 다음과 같습니다.

- DRDA V3 Vol. 1: Distributed Relational Database Architecture™
- DRDA V3 Vol. 2: Formatted Data Object Content Architecture
- DRDA V3 Vol. 3: Distributed Data Management Architecture

이러한 매뉴얼의 PDF 버전은 www.opengroup.org에서 제공됩니다.

db2drdat 유틸리티는 DRDA 응용프로그램 리퀘스터(AR)와 DB2 DRDA 응용프로그램 서버(AS)간(예: DB2 Connect와 호스트 또는 Power Systems® 서버 데이터베이스 서버간)에 교환되는 데이터를 기록합니다.

추적 유틸리티

db2drdat 유틸리티는 DB2 Connect 서버(IBM Data Server Client 대신)와 IBM 메인프레임 데이터베이스 서버 간에 교환되는 데이터를 기록합니다.

데이터베이스 관리자(또는 응용프로그램 개발자)로서 데이터 작업이 어떻게 진행되는지를 알고 있으면 특정 문제점의 원인을 판별하는데 도움이 됩니다. IBM 메인프레임 데이터베이스 서버의 CONNECT TO 데이터베이스 명령문을 발행했지만 명령이 실패하고 실패 리턴 코드를 받았다고 가정해 봅시다. 어떤 정보가 IBM 메인프레임 데이터베이스 서버 관리 시스템에 전달되었는지 정확히 알고 있다면 리턴 코드 정보가 일반적이라 해도 실패의 원인을 판별할 수 있습니다. 실패의 대부분은 단순히 사용자 오류에 의한 것입니다.

db2drdat의 출력에 DB2 Connect 워크스테이션과 IBM 메인프레임 데이터베이스 서버 관리 시스템 간에 교환된 데이터 스트림이 나열됩니다. IBM 메인프레임 데이터베이스 서버에 전송된 데이터에는 SEND BUFFER 레이블이 작성되고 IBM 메인프레임 데이터베이스 서버에서 수신된 데이터에는 RECEIVE BUFFER 레이블이 작성됩니다.

버퍼 받기에 SQLCA 정보가 포함되는 경우 이 데이터의 형식화된 해석과 레이블이 작성된 SQLCQ가 뒤에 옵니다. SQLCA의 SQLCODE 필드는 IBM 메인프레임 데이터베이스 서버에서 리턴된 *맵핑되지* 않은 값입니다. 버퍼 보내기 및 받기는 파일 내에 가장 이전 버퍼에서부터 가장 최근 버퍼까지 배열됩니다. 각 버퍼에는 다음이 포함됩니다.

- 프로세스 ID
- SEND BUFFER, RECEIVE BUFFER 또는 SQLCA 레이블. 버퍼의 첫 번째 DDM 명령 또는 오브젝트는 DSS TYPE으로 레이블이 작성됩니다.

버퍼 보내기 및 받기에 남아있는 데이터는 다섯 개의 컬럼으로 나뉘며 다음으로 구성됩니다.

- 바이트 수
- 컬럼 2 및 3은 ASCII 또는 EBCDIC 두 시스템 간에 교환된 DRDA 데이터 스트림을 나타냅니다.
- 컬럼 2 및 3의 ASCII 표시
- 컬럼 2 및 3의 EBCDIC 표시

추적 결과

db2drdat 유틸리티는 다음 정보를 *tracefile*에 작성합니다.

- -r

  - DRDA 응답/오브젝트 유형
  - 버퍼 받기
- -s
  - DRDA 요청 유형
  - 버퍼 보내기
- -c
  - SQLCA
- TCP/IP 오류 정보
  - 함수 리턴 코드 받기
  - 심각도
  - 사용된 프로토콜
  - 사용된 API
  - 함수
  - 오류 번호

주:

1. 종료 코드의 영(0) 값은 명령이 성공적으로 완료되었음을 표시하고 0이 아닌 값은 그렇지 않음을 표시합니다.
2. 리턴된 필드는 사용된 API에 따라 달라집니다.
3. 리턴된 필드는 API가 같더라도 DB2 Connect이 실행 중인 플랫폼에 따라 달라집니다.
4. db2drdat 명령이 이미 존재하는 파일에 출력을 전송하면 이전 파일은 파일에 대한 사용 권한이 이전 파일을 지우도록 허용하는 경우 지워집니다.

추적 결과 파일 분석

다음 정보는 db2drdat 추적에 캡처됩니다.

- 클라이언트 응용프로그램의 PID(Process ID)
- DCS(Database Connection Service) 디렉토리에 카탈로그된 RDB_NAME
- DB2 Connect CCSID
- IBM 메인프레임 데이터베이스 서버 CCSID
- DB2 Connect 시스템이 통신하는 IBM 메인프레임 데이터베이스 서버 관리 시스템.

첫 번째 버퍼에 IBM 메인프레임 데이터베이스 서버 관리 시스템에 전송된 EXCSAT(Exchange Server Attributes)와 ACCRDB(Access RDB)가 있습니다. 첫 번째 버퍼가 CONNECT TO 데이터베이스 명령의 결과로 이 명령을 전송합니다. 다음 버퍼에 IBM 메인프레임 데이터베이스 서버 관리 시스템에서 DB2 Connect이 수신한 응답

이 있습니다. 여기에는 EXCSATRD(Exchange Server Attributes Reply Data)와 ACCRDBRM(Access RDB Reply Message)이 있습니다.

EXCSAT

EXCSAT 명령에는 SRVNAM(서버 이름) 오브젝트에서 지정된 클라이언트의 워크스테이션 이름이 포함되어 있으며 DDM 스펙에 따라 코드 포인트가 X'116D'입니다. EXCSAT 명령은 첫 번째 버퍼에 있습니다. EXCSAT 명령 내에서 값 X'9481A292'(CCSID 500에 코드됨)는 X'116D'가 제거되면 *마스크*로 변환됩니다.

EXCSAT 명령에는 또한 EXTNAM(외부 이름) 오브젝트가 포함되어 있으며 이것은 IBM 메인프레임 데이터베이스 관리 시스템의 진단 정보에 있는 경우가 많습니다. 이 오브젝트는 20바이트의 응용프로그램 ID와 그 뒤의 8바이트 프로세스 ID 또는 4바이트의 프로세스 ID와 4바이트의 스레드 ID로 구성됩니다. 이것은 코드 포인트 X'115E'로 표시되며 이 예에서 값은 뒤에 000C50CC가 오고 공백으로 패드된 db2bp입니다. Linux 또는 UNIX IBM Data Server Client에서 이 값은 ps 명령과 관련될 수 있으며 이 명령은 사용 중인 프로세스에 대한 프로세스 상태 정보를 표준 출력에 리턴합니다.

ACCRDB

ACCRDB 명령에는 RDBNAM 오브젝트에 RDB_NAME이 포함되어 있으며 코드 포인트가 X'2110'입니다. ACCRDB 명령은 첫 번째 버퍼의 EXCSAT 명령을 따릅니다. ACCRDB 명령에서 값 X'E2E3D3C5C3F1'은 X'2110'이 제거되면 STLEC1로 전환됩니다. 이 값은 DCS 디렉토리의 목표 데이터베이스 이름 필드와 일치합니다.

어카운팅 문자열에는 코드 포인트 X'2104'가 있습니다.

DB2 Connect 워크스테이션에 구성된 코드 세트는 ACCRDB 명령에서 코드 포인트가 X'119C'인 CCSID 오브젝트 CCSIDSBC(단일 바이트 문자의 CCSID)를 찾아서 표시됩니다. 이 예에서 CCSIDSBC는 X'0333'이며 이것은 819입니다.

코드 포인트가 각각 X'119D' 및 X'119E'인 추가 오브젝트 CCSIDDBC(이중 바이트 문자의 CCSID) 및 CCSIDMBC(혼합 바이트 문자의 CCSID)는 ACCRDB 명령에도 표시됩니다. 이 예에서 CCSIDDBC는 X'04B0'이며 1200이고 CCSIDMBC는 X'0333'이며 819입니다.

EXCSATRD 및 ACCRDBRM

CCSID 값은 두 번째 버퍼 내의 ACCRDBRM(Access RDB Reply Message)의 IBM 메인프레임 데이터베이스 서버에서도 리턴됩니다. 이 버퍼에는 ACCRDBRM이 뒤에 오는 EXCSATRD가 있습니다. 예제 출력 파일에는 IBM 메인프레임 데이터베이스 서버 시스템의 두 개의 CCSID 값이 있습니다. 값은 1208(단일 바이트 및 혼합 바이트 문자의 경우)과 1200(이중 바이트 문자의 경우)입니다.

DB2 Connect이 IBM 메인프레임 데이터베이스 서버에서 되돌아오는 코드 페이지를 인식하지 못하는 경우 SQLCODE -332가 소스 및 목표 코드 페이지와 함께 사용자에게 리턴됩니다. IBM 메인프레임 데이터베이스 서버가 DB2 Connect에서 전송되는 코드 세트를 인식하지 못하는 경우 VALNSPRM(DDM 코드 포인트가 X'1252'인 지원되지 않는 매개변수 값)을 리턴하여 사용자에게 SQLCODE -332로 변환됩니다.

ACCRDBRM에는 매개변수 PRDID(코드 포인트가 X'112E'인 제품 특정 ID)도 있습니다. 이 값은 X'C4E2D5F0F8F0F1F5'로 EBCDIC에서 DSN08015입니다. 표준에 따라 DSN은 z/OS용 DB2입니다. 버전 번호도 표시됩니다. ARI는 DB2 Server for VSE & VM이고 SQL은 DB2 데이터베이스 또는 DB2 Connect이며 QSQ는 IBM i용 DB2입니다.

추적 결과 파일 샘플

다음 그림은 DB2 Connect 워크스테이션과 호스트 또는 시스템 i 데이터베이스 서버 간에 교환된 일부 DRDA 데이터 스트림을 표시하는 샘플 출력을 나타냅니다. 사용자 측면에서는 `CONNECT TO` 데이터베이스 명령이 명령행 처리기(CLP)를 사용하여 발행되었습니다.

500 페이지의 그림 35에서는 DB2 Connect Enterprise Edition 버전 9.1 및 TCP/IP 연결을 통한 z/OS용 DB2 버전 8을 사용합니다.

```
1 data DB2 UDB DRDA Communication Manager sqljcSend fnc (3.3.54.5.0.100)
 pid 807116 tid 1 cpid -1 node 0 sec 0 nsec 0 probe 100
 bytes 16

 Data1  (PD_TYPE_UINT,8) unsigned integer:
 233

2 data DB2 UDB DRDA Communication Manager sqljcSend fnc (3.3.54.5.0.1177)
 pid 807116 tid 1 cpid -1 node 0 sec 0 nsec 19532 probe 1177
 bytes 250

 SEND BUFFER(AR):

           EXCSAT RQSDSS                     (ASCII)           (EBCDIC)
         0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  00C3D041000100BD 1041007F115E8482   ...A.....A...^..  .C}........".;db
  0010  F282974040404040 4040404040404040   ...@@@@@@@@@@@@@  2bp
  0020  4040F0F0F0C3F5F0 C3C3F0F0F0000000   @@..............    000C50CC000...
  0030  0000000000000000 0000000000000000   ................  ................
  0040  0000000000000000 000000000060F0F0   ............`..  ............-00
  0050  F0F1A2A495404040 4040404040404040   .....@@@@@@@@@@@  01sun
  0060  4040404040404040 4040404040404040   @@@@@@@@@@@@@@@@
  0070  C4C5C3E5F8404040 F0A2A49540404040   .....@@@....@@@@  DECV8   0sun
  0080  4040404040404040 4000181404140300   @@@@@@@@@.......           .......
  0090  0724070008147400 05240F0008144000   .$....t..$....@.  .............. .
  00A0  08000E1147D8C4C2 F261C1C9E7F6F400   ....G....a......  .....QDB2/AIX64.
  00B0  08116D9481A29200 0C115AE2D8D3F0F9   ..m.......Z.....  .._mask...]SQL09
  00C0  F0F0F0                              ...               000

           ACCSEC RQSDSS                     (ASCII)           (EBCDIC)
         0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  0026D00100020020 106D000611A20003   .&..... .m......  ..}......_...s..
  0010  00162110E2E3D3C5 C3F1404040404040   ..!.......@@@@@@  ....STLEC1
  0020  404040404040                        @@@@@@

3 data DB2 UDB DRDA Communication Manager sqljcReceive fnc (3.3.54.3.0.100)
 pid 807116 tid 1 cpid -1 node 0 sec 0 nsec 110546200 probe 100
 bytes 12

 Data1  (PD_TYPE_UINT,4) unsigned integer:
 105

4 data DB2 UDB DRDA Communication Manager sqljcReceive fnc (3.3.54.3.0.1178)
 pid 807116 tid 1 cpid -1 node 0 sec 0 nsec 110549755 probe 1178
 bytes 122

 RECEIVE BUFFER(AR):

            EXCSATRD OBJDSS                  (ASCII)           (EBCDIC)
          0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  0059D04300010053 1443000F115EE5F8   .Y.C...S.C...^..  ..}.........;V8
  0010  F1C14BE2E3D3C5C3 F100181404140300   ..K.............  1A.STLEC1.......
  0020  0724070007147400 05240F0007144000   .$....t..$....@.  .............. .
  0030  0700081147D8C4C2 F20014116DE2E3D3   ....G.......m...  .....QDB2..._STL
  0040  C5C3F14040404040 4040404040000C11   ...@@@@@@@@@@...  EC1          ...
  0050  5AC4E2D5F0F8F0F1 F5                Z........         ]DSN08015

            ACCSECRD OBJDSS                  (ASCII)           (EBCDIC)
          0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  0010D0030002000A 14AC000611A20003   ................  ..}.........s..

5 data DB2 UDB DRDA Communication Manager sqljcSend fnc (3.3.54.5.0.100)
 pid 807116 tid 1 cpid -1 node 0 sec 0 nsec 110656806 probe 100
 bytes 16

 Data1  (PD_TYPE_UINT,8) unsigned integer:
 233
```

그림 35. 추적 출력의 예(TCP/IP 연결)

```
6 data DB2 UDB DRDA Communication Manager sqljcSend fnc (3.3.54.5.0.1177)
 pid 807116 tid 1 cpid -1 node 0 sec 0 nsec 110659711 probe 1177
 bytes 250

 SEND BUFFER(AR):

           SECCHK RQSDSS                    (ASCII)           (EBCDIC)
         0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  003CD04100010036 106E000611A20003   .<.A...6.n......  ..}......>...s..
  0010  00162110E2E3D3C5 C3F1404040404040   ..!.......@@@@@@  ....STLEC1
  0020  404040404040000C 11A1D9858799F485   @@@@@.........        ....Regr4e
  0030  A599000A11A09585 A6A39695           ............      vr....newton

           ACCRDB RQSDSS                    (ASCII)           (EBCDIC)
         0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  00ADD001000200A7 20010006210F2407   ........ ...!.$.  ..}....x........
  0010  00172135C7F9F1C1 F0C4F3C14BD7C1F8   ..!5........K...  ....G91A0D3A.PA8
  0020  F806030221064600 162110E2E3D3C5C3   ....!.F..!......  8.........STLEC
  0030  F140404040404040 4040404040000C11   .@@@@@@@@@@@@...  1           ...
  0040  2EE2D8D3F0F9F0F0 F0000D002FD8E3C4   ............/...  .SQL09000....QTD
  0050  E2D8D3C1E2C30016 00350006119C0333   .........5.....3  SQLASC.........
  0060  0006119D04B00006 119E0333003C2104   ...........3.

7 data DB2 UDB DRDA Communication Manager sqljcReceive fnc (3.3.54.3.0.100)
 pid 807116 tid 1 cpid -1 node 0 sec 0 nsec 259908001 probe 100
 bytes 12

 Data1  (PD_TYPE_UINT,4) unsigned integer:
 176

8 data DB2 UDB DRDA Communication Manager sqljcReceive fnc (3.3.54.3.0.1178)
 pid 807116 tid 1 cpid -1 node 0 sec 0 nsec 259911584 probe 1178
 bytes 193

 RECEIVE BUFFER(AR):

              SECCHKRM RPYDSS                 (ASCII)           (EBCDIC)
            0 1 2 3 4 5 6 7  8 9 A B C D E F  0123456789ABCDEF  0123456789ABCDEF
  0000  0015D0420001000F 1219000611490000  ...B.........I..  ..}.............
  0010  000511A400                         .....             ...u.

              ACCRDBRM RPYDSS                 (ASCII)           (EBCDIC)
            0 1 2 3 4 5 6 7  8 9 A B C D E F  0123456789ABCDEF  0123456789ABCDEF
  0000  009BD00200020095 2201000611490000  ........"....I..  ..}....n........
  0010  000D002FD8E3C4E2 D8D3F3F7F0000C11  .../............  ....QTDSQL370...
  0020  2EC4E2D5F0F8F0F1 F500160035000611  ............5...  .DSN08015.......
  0030  9C04B80006119E04 B80006119D04B000  ................  ................
  0040  0C11A0D5C5E6E3D6 D540400006212524  .........@@..!%$  ...NEWTON  .....
  0050  34001E244E000624 4C00010014244D00  4..$N..$L....$M.  ....+...<.....(.
  0060  06244FFFFF000A11 E8091E768301BE00  .$O........v....  ..!.....Y...c...
  0070  2221030000000005 68B3B8C7F9F1C1F0  "!......h.......  ...........G91A0
  0080  C4F3C1D7C1F8F840 4040400603022106  .......@@@@...!.  D3APA88    .....
  0090  46000A11E8091E76 831389            F......v...       ....Y...c.i

9 data DB2 UDB DRDA Communication Manager sqljcSend fnc (3.3.54.5.0.100)
 pid 807116 tid 1 cpid -1 node 0 sec 2 nsec 364420503 probe 100
 bytes 16

 Data1  (PD_TYPE_UINT,8) unsigned integer:
 10
```

그림 36. 계속되는 추적 출력의 예(TCP/IP 연결)

```
10 data DB2 UDB DRDA Communication Manager sqljcSend fnc (3.3.54.5.0.1177)
 pid 807116 tid 1 cpid -1 node 0 sec 2 nsec 364440751 probe 1177
 bytes 27

 SEND BUFFER(AR):

           RDBCMM RQSDSS                    (ASCII)          (EBCDIC)
         0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  000AD00100010004 200E              ........ .       ..}.......

11 data DB2 UDB DRDA Communication Manager sqljcReceive fnc (3.3.54.3.0.100)
 pid 807116 tid 1 cpid -1 node 0 sec 2 nsec 475009631 probe 100
 bytes 12

 Data1  (PD_TYPE_UINT,4) unsigned integer:
 54

12 data DB2 UDB DRDA Communication Manager sqljcReceive fnc (3.3.54.3.0.1178)
 pid 807116 tid 1 cpid -1 node 0 sec 2 nsec 475014579 probe 1178
 bytes 71

 RECEIVE BUFFER(AR):

            ENDUOWRM RPYDSS                  (ASCII)          (EBCDIC)
          0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  002BD05200010025 220C000611490004   .+.R...%"....I..  ..}.............
  0010  00162110E2E3D3C5 C3F1404040404040   ..!.......@@@@@@  ....STLEC1
  0020  4040404040400005 211501             @@@@@@..!..             .....

            SQLCARD OBJDSS                   (ASCII)          (EBCDIC)
          0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  000BD00300010005 2408FF             ........$..       ..}........

13 data DB2 UDB DRDA Communication Manager sqljcSend fnc (3.3.54.5.0.100)
 pid 807116 tid 1 cpid -1 node 0 sec 5 nsec 721710319 probe 100
 bytes 16

 Data1  (PD_TYPE_UINT,8) unsigned integer:
 126

14 data DB2 UDB DRDA Communication Manager sqljcSend fnc (3.3.54.5.0.1177)
 pid 807116 tid 1 cpid -1 node 0 sec 5 nsec 721727276 probe 1177
 bytes 143

 SEND BUFFER(AR):

            EXCSQLIMM RQSDSS                 (ASCII)          (EBCDIC)
          0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  0053D0510001004D 200A00442113E2E3   .S.Q...M ..D!...  ..}....(......ST
  0010  D3C5C3F140404040 4040404040404040   ....@@@@@@@@@@@@  LEC1
  0020  D5E4D3D3C9C44040 4040404040404040   ......@@@@@@@@@@  NULLID
  0030  4040E2D8D3C3F2C6 F0C1404040404040   @@........@@@@@@    SQLC2F0A
  0040  4040404041414141 41484C5600CB0005   @@@@AAAAAAHLV....      ......<.....
  0050  2105F1                              !..               ..1

            SQLSTT OBJDSS                    (ASCII)          (EBCDIC)
          0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  002BD00300010025 2414000000001B64   .+.....%$......d  ..}.............
  0010  656C657465206672 6F6D206464637375   elete from ddcsu  .%......?_......
  0020  73312E6D79746162 6C65FF             s1.mytable.       ..._`./.%..

15 data DB2 UDB DRDA Communication Manager sqljcReceive fnc (3.3.54.3.0.100)
 pid 807116 tid 1 cpid -1 node 0 sec 5 nsec 832901261 probe 100
 bytes 12

 Data1  (PD_TYPE_UINT,4) unsigned integer:
 102
```

그림 37. 계속되는 추적 출력의 예(TCP/IP 연결)

```
16 data DB2 UDB DRDA Communication Manager sqljcReceive fnc (3.3.54.3.0.1178)
 pid 807116 tid 1 cpid -1 node 0 sec 5 nsec 832906528 probe 1178
 bytes 119

 RECEIVE BUFFER(AR):

           SQLCARD OBJDSS                   (ASCII)           (EBCDIC)
         0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  0066D00300010060 240800FFFFFF3434   .f.....`$.....44  ..}....-........
  0010  3237303444534E58 4F544C2000FFFFFE   2704DSNXOTL ....  ......+.!.<.....
  0020  0C00000000000000 00FFFFFFFF000000   ................  ................
  0030  0000000000572020 2057202020202020   .....W   W       ................
  0040  001053544C454331 2020202020202020   ..STLEC1          ....<...........
  0050  2020000F44444353 5553312E4D595441     ..DDCSUS1.MYTA  ............(...
  0060  424C450000FF                        BLE...            .<....

17 data DB2 UDB DRDA Communication Manager sqljcSend fnc (3.3.54.5.0.100)
 pid 807116 tid 1 cpid -1 node 0 sec 5 nsec 833156953 probe 100
 bytes 16

 Data1  (PD_TYPE_UINT,8) unsigned integer:
 10

18 data DB2 UDB DRDA Communication Manager sqljcSend fnc (3.3.54.5.0.1177)
 pid 807116 tid 1 cpid -1 node 0 sec 5 nsec 833159843 probe 1177
 bytes 27

 SEND BUFFER(AR):

           RDBRLLBCK RQSDSS                 (ASCII)           (EBCDIC)
         0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  000AD00100010004 200F               ........ .        ..}.......

19 data DB2 UDB DRDA Communication Manager sqljcReceive fnc (3.3.54.3.0.100)
 pid 807116 tid 1 cpid -1 node 0 sec 5 nsec 943302832 probe 100
 bytes 12

 Data1  (PD_TYPE_UINT,4) unsigned integer:
 54

20 data DB2 UDB DRDA Communication Manager sqljcReceive fnc (3.3.54.3.0.1178)
 pid 807116 tid 1 cpid -1 node 0 sec 5 nsec 943306288 probe 1178
 bytes 71

 RECEIVE BUFFER(AR):

           ENDUOWRM RPYDSS                  (ASCII)           (EBCDIC)
         0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  002BD05200010025 220C000611490004   .+.R...%"....I..  ..}.............
  0010  00162110E2E3D3C5 C3F1404040404040   ..!.......@@@@@@  ....STLEC1
  0020  4040404040400005 211502             @@@@@@..!..             .....

           SQLCARD OBJDSS                   (ASCII)           (EBCDIC)
         0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  000BD00300010005 2408FF             ........$..       ..}........
```

그림 38. 계속되는 추적 출력의 예(TCP/IP 연결)

DRDA 추적에 대한 연속 버퍼 정보

추가 정보에 대한 연속 버퍼 보내기 및 받기를 분석할 수 있습니다. 다음 요청에는 커미트가 포함됩니다. commit 명령은 IBM 메인프레임 데이터베이스 서버 관리 시스템이 현재 작업 단위을 커미트하도록 지시합니다. 네 번째 버퍼가 커미트 또는 롤백의 결과

로 IBM 메인프레임 데이터베이스 서버 데이터베이스 관리 시스템에서 수신됩니다. 여기에는 ENDUOWRM(End Unit of Work Reply Message)이 포함되고 이것은 현재 작업 단위가 종료되었음을 표시합니다.

이 예에서 추적 항목 12에 널(NULL) SQLCA가 포함되어 있고 이것은 뒤에 X'FF'가 오는 DDM 코드 포인트 X'2408'로 표시됩니다. 널(NULL) SQLCA(X'2408FF')는 성공(SQLCODE 0)을 나타냅니다.

500 페이지의 그림 35에는 추적 항목 16에 오류 SQLCA가 포함된 버퍼 받기의 예가 표시됩니다.

제어 센터 추적

제어 센터에서 문제점 추적을 시도하기 전에 DB2 명령 프롬프트에서 명시적 명령을 통해 동등한 조치가 수행되는 경우 동일한 문제점이 발생하지 않는지 먼저 확인하는 것이 바람직합니다.

제어 센터(또는 제어 센터에서 시작할 수 있는 다른 GUI 도구 중 하나)에서 태스크를 수행하는 경우 도구가 사용할 명령에 정확한 구문을 제공하는 "명령 표시" 단추가 자주 표시됩니다. 정확한 해당 명령이 DB2 명령 프롬프트에서 성공하지만 GUI 도구에서 실행될 경우에는 실패하면, 제어 센터 추적을 확보하는 것이 적절합니다.

제어 센터에서만 재현 가능한 문제점 추적을 확보하려면 다음과 같이 제어 센터를 시작하십시오.

```
db2cc -tf filename
```

이는 제어 센터 추적을 작동하고 추적 출력을 지정된 파일에 저장합니다. 추적 파일은 Windows에서는 <DB2 install path>₩sqllib₩tools, UNIX 및 Linux에서는 /home/<userid>/sqllib/tools에 저장됩니다.

주: 추적을 사용 가능하게 하여 제어 센터가 시작되었으면, 가능한 적은 단계를 사용하여 문제점을 재현하십시오. 도구에서 불필요하거나 관련되지 않은 항목을 누르지 마십시오. 문제점을 재현했으면, 제어 센터(및 문제점을 재현하기 위해 연 기타 GUI 도구)를 닫으십시오.

분석을 위해 결과 추적 파일을 IBM Software Support로 보내야 합니다.

JDBC 추적 파일

Linux, UNIX 및 Windows용 DB2 JDBC 유형 2 드라이버를 사용하는 응용프로그램 추적 확보

이 태스크는 Linux, UNIX 및 Windows용 DB2 JDBC 유형 2 드라이버를 사용하는 응용프로그램 추적 확보 방법을 설명합니다.

이 추적 유형은 다음 위치에서 문제점이 발생하는 상황에 적용 가능합니다.

- Linux, UNIX 및 Windows용 DB2 JDBC 유형 2 드라이버(DB2 JDBC 유형 2 드라이버)를 사용하는 JDBC 응용프로그램
- DB2 JDBC 스토어드 프로시저

주: 응용프로그램 동작에 영향을 줄 수 있는 db2cli.ini 파일에 추가할 수 있는 키워드가 많이 있습니다. 이러한 키워드는 응용프로그램 문제점을 해결하거나 응용프로그램 문제점의 원인이 될 수 있습니다. CLI 문서에서 다루지 않는 일부 키워드도 있습니다. DB2 지원부에서만 이를 사용할 수 있습니다. 문서화되지 않은 키워드가 db2cli.ini 파일에 있는 경우, DB2 지원부에서 권장한 키워드일 가능성이 있습니다. 내부적으로 DB2 JDBC 유형 2 드라이버는 데이터베이스 액세스를 위해 DB2 CLI 드라이버를 사용합니다. 예를 들어, Java getConnection() 메소드는 DB2 JDBC Type 2 드라이버에 의해 DB2 CLI SQLConnect() 함수에 내부적으로 맵핑됩니다. 따라서 Java 개발자는 DB2 CLI 추적이 DB2 JDBC 추적에 유용한 보완물임을 알 수 있습니다.

1. 추적 파일 경로를 작성하십시오. 모든 사용자가 기록할 수 있는 경로를 작성하는 것이 중요합니다.

   예를 들어, Windows의 경우:

   ```
   mkdir c:₩temp₩trace
   ```

   Linux 및 UNIX의 경우:

   ```
   mkdir /tmp/trace
   chmod 777 /tmp/trace
   ```

2. CLI 구성 키워드를 갱신하십시오. 이를 수행하는 두 가지 방법이 있습니다.
   - db2cli.ini 파일을 수동으로 편집하십시오. 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』 주제를 참조하십시오.
     a. 일반 텍스트 편집기에서 db2cli.ini 파일을 여십시오.
     b. 파일에 다음 섹션을 추가하십시오(COMMON 섹션이 이미 존재하는 경우에는 변수만 추가).

        ```
        [COMMON]
        JDBCTrace=1
        JDBCTracePathName=<path>
        JDBCTraceFlush=1
        ```

        여기서 <path>는 예를 들어 Windows에서는 C:₩temp₩trace, Linux 또는 UNIX 운영 체제에서는 /tmp/trace입니다.

c. 파일 끝에 최소 하나의 공백 라인을 사용하여 파일을 저장하십시오(이렇게 하면 일부 구문 분석 오류가 방지됨).

- UPDATE CLI CFG 명령을 사용하여 db2cli.ini 파일을 갱신하십시오. 다음 명령을 실행하십시오.

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING JDBCTrace 1
db2 UPDATE CLI CFG FOR SECTION COMMON USING JDBCTracePathName <path>
```

여기서 <path>는 예를 들어 Windows에서는 C:¥temp¥trace, Linux 또는 UNIX 운영 체제에서는 /tmp/trace입니다.

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING JDBCTraceFlush 1
```

추적 기능을 사용하여 응용프로그램 문제점을 진단하는 경우, 추적 기능은 응용프로그램 성능에 영향을 주며 테스트 응용프로그램 뿐만 아니라 모든 응용프로그램에 영향을 준다는 점을 기억하십시오. 따라서 문제점을 식별한 후에는 추적 기능을 해제하는 것이 중요합니다.

3. 다음 명령을 실행하여 올바른 키워드가 설정되어 선택되고 있는지 검증하십시오.

```
db2 GET CLI CFG FOR SECTION COMMON
```

4. 응용프로그램을 재시작하십시오.

응용프로그램이 시작될 때에만 db2cli.ini 파일을 읽으므로 변경사항을 적용하려면 응용프로그램을 재시작해야 합니다.

JDBC 스토어드 프로시저를 추적하는 경우, 이는 DB2 인스턴스 재시작을 의미합니다.

5. 오류를 캡처하십시오. 오류가 생성될 때까지 응용프로그램을 실행한 후 응용프로그램을 종료하십시오. 가능하다면, 추적 당시에 실행 중인 JDBC 응용프로그램만 문제점 재현과 관련되도록 상황을 축소하십시오. 이렇게 하면 추적 파일이 훨씬 명확해집니다.

6. JDBC 추적을 사용할 수 없게 하십시오.

db2cli.ini의 [COMMON] 섹션에서 JDBCTrace=0 키워드를 수동으로 설정하거나 다음 명령을 실행하십시오.

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING Trace 0
db2 UPDATE CLI CFG FOR SECTION COMMON USING JDBCTrace 0
```

7. 실행 및 추적 중인 응용프로그램을 재시작하십시오.

8. 추적 파일을 수집하십시오.

JDBC 추적 파일은 JDBCTracePathName 키워드에 지정된 경로에 기록됩니다. 생성된 파일 이름은 모두 .trc 확장자로 끝납니다. 문제점 재현 시에 추적 경로에 생성된 모든 파일이 필요합니다.

DB2 Universal JDBC 드라이버를 사용하는 응용프로그램 추적 확보

이 태스크는 DB2 Universal JDBC 드라이버를 사용하는 응용프로그램 추적 확보 방법을 설명합니다.

DB2 Universal JDBC 드라이버를 사용 중인 SQLJ 또는 JDBC 응용프로그램이 있으면, 몇 가지 다른 방법으로 JDBC 추적을 사용할 수 있습니다.

- DataSource 인터페이스를 사용하여 데이터 소스에 연결하는 경우, DataSource.setTraceLevel() 및 DataSource.setTraceFile() 메소드를 사용하여 추적을 사용할 수 있게 하십시오.
- DriverManager 인터페이스를 사용하여 데이터 소스에 연결하는 경우, 추적을 사용하는 가장 쉬운 방법은 연결을 확보하기 전에 DriverManager에서 logWriter를 설정하는 것입니다.

  예를 들어, 다음과 같습니다.

  ```
  DriverManager.setLogWriter(new PrintWriter(new FileOutputStream("trace.txt")));
  ```

- DriverManager 인터페이스를 사용 중인 경우, 드라이버를 로드할 때 URL의 일부로 traceFile 및 traceLevel 등록 정보를 지정할 수도 있습니다.

  예를 들어, 다음과 같습니다.

  ```
  String databaseURL =
  "jdbc:db2://hal:50000/sample:traceFile=c:/temp/foobar.txt;" ;
  ```

CLI 추적 파일

CLI 추적은 DB2 CLI 드라이버에 액세스하는 응용프로그램에 대한 정보를 캡처합니다. CLI 추적은 DB2 CLI 드라이버 내부 작업에 대한 아주 미미한 정보를 제공합니다.

이 추적 유형은 다음 위치에서 문제점이 발생하는 상황에 적용 가능합니다.

- CLI 응용프로그램
- ODBC 응용프로그램(ODBC 응용프로그램이 DB2 CLI 인터페이스를 사용하여 DB2에 액세스하기 때문)
- DB2 CLI 스토어드 프로시저
- JDBC 응용프로그램 및 스토어드 프로시저

ODBC 응용프로그램 진단 시 ODBC 추적 또는 DB2 CLI 추적을 사용하여 문제점을 판별하는 것이 가장 쉽습니다. ODBC 드라이버 관리자를 사용 중인 경우에는 ODBC 추적 기능을 제공할 가능성이 높습니다. 드라이버 관리 프로그램 문서를 참조하여 ODBC 추적 사용 방법을 판별하십시오. DB2 CLI 추적은 DB2에 특정하며 일반 ODBC 추적보다 자세한 정보를 포함하는 경우가 많습니다. 두 추적 모두 일반적으로 상당히 유

사하며, 해당 호출에 대한 리턴 코드와 매개변수를 포함하여 응용프로그램으로부터의 모든 CLI 호출에 대한 진입점과 종료점을 나열합니다.

Linux, UNIX 및 Windows용 DB2 JDBC 유형 2 드라이버(DB2 JDBC 유형 2 드라이버)는 DB2 CLI 드라이버에 의존하여 데이터베이스에 액세스합니다. 따라서 Java 개발자는 응용프로그램이 여러 소프트웨어 계층을 통해 데이터베이스와 상호작용하는 방법에 대한 추가 정보를 얻기 위해 DB2 CLI 추적도 사용하려고 할 수 있습니다. DB2 JDBC 및 DB2 CLI 추적 옵션(둘 다 db2cli.ini 파일에 설정됨)은 서로 독립적입니다.

CLI 추적 확보

CLI 추적을 작동하려면 CLI 구성 키워드 세트를 사용할 수 있게 해야 합니다.

시작하기 전에

주: 응용프로그램 동작에 영향을 줄 수 있는 db2cli.ini 파일에 추가할 수 있는 키워드가 많이 있습니다. 이러한 키워드는 응용프로그램 문제점을 해결하거나 응용프로그램 문제점의 원인이 될 수 있습니다. CLI 문서에서 다루지 않는 일부 키워드도 있습니다. IBM Software Support에서만 이를 사용할 수 있습니다. 문서화되지 않은 키워드가 db2cli.ini 파일에 있는 경우, IBM Software Support에서 권장한 키워드일 가능성이 있습니다.

이 태스크에 대한 정보

추적 기능을 사용하여 응용프로그램 문제점을 진단하는 경우, 추적 기능은 응용프로그램 성능에 영향을 주며 테스트 응용프로그램뿐만 아니라 모든 응용프로그램에 영향을 준다는 점을 기억하십시오. 따라서 문제점을 식별한 후에는 추적 기능을 해제하는 것이 중요합니다.

프로시저

CLI 추적을 확보하려면 다음을 수행하십시오.

1. 추적 파일 경로를 작성하십시오.

   모든 사용자가 기록할 수 있는 경로를 작성하는 것이 중요합니다. 예를 들어, Windows 운영 체제의 경우:

   ```
   mkdir c:¥temp¥trace
   ```

   Linux 및 UNIX 운영 체제의 경우:

   ```
   mkdir /tmp/trace
   chmod 777 /tmp/trace
   ```

2. CLI 구성 키워드를 갱신하십시오.

   db2cli.ini 파일을 수동으로 편집하거나 UPDATE CLI CFG 명령을 사용하여 이를 수행할 수 있습니다.

- db2cli.ini 파일을 수동으로 편집하려면 다음을 수행하십시오.
  a. 일반 텍스트 편집기에서 db2cli.ini 파일을 여십시오. 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』 주제를 참조하십시오.
  b. 파일에 다음 섹션을 추가하십시오(또는 COMMON 섹션이 이미 존재하는 경우에는 변수만 추가).

     ```
     [COMMON]
     Trace=1
     TracePathName=path
     TraceComm=1
     TraceFlush=1
     TraceTimeStamp=1
     ```

     여기서 *path*는 예를 들어 Windows에서는 C:₩temp₩trace, Linux 및 UNIX에서는 /tmp/trace입니다.
  c. 파일 끝에 최소 하나의 공백 라인을 사용하여 파일을 저장하십시오(이렇게 하면 일부 구문 분석 오류가 방지됨).
- UPDATE CLI CFG 명령을 사용하여 CLI 구성 키워드를 갱신하려면 다음 명령을 실행하십시오.

  ```
  db2 UPDATE CLI CFG FOR SECTION COMMON USING Trace 1
  db2 UPDATE CLI CFG FOR SECTION COMMON USING TracePathName path
  db2 UPDATE CLI CFG FOR SECTION COMMON USING TraceComm 1
  db2 UPDATE CLI CFG FOR SECTION COMMON USING TraceFlush 1
  db2 UPDATE CLI CFG FOR SECTION COMMON USING TraceTimeStamp 3
  ```

  여기서 *path*는 예를 들어 Windows에서는 C:₩temp₩trace, Linux 및 UNIX에서는 /tmp/trace입니다.

3. db2cli.ini 구성을 확인하십시오.

   다음 명령을 실행하여 올바른 키워드가 설정되어 선택되고 있는지 검증하십시오.

   ```
   db2 GET CLI CFG FOR SECTION COMMON
   ```

4. 응용프로그램을 재시작하십시오.

   응용프로그램 시작 시에만 db2cli.ini 파일을 읽으므로 변경사항을 적용하려면 응용프로그램을 재시작해야 합니다.

   CLI 스토어드 프로시저를 추적하는 경우, 이는 DB2 인스턴스 재시작을 의미합니다.

5. 오류를 캡처하십시오.

오류가 생성될 때까지 응용프로그램을 실행한 후 응용프로그램을 종료하십시오. 문제점 재현과 관련된 응용프로그램만 추적 당시에 실행되도록 상황을 축소할 수 있으면, 추적 분석이 훨씬 명확해집니다.

6. CLI 추적을 사용할 수 없게 하십시오.

   db2cli.ini의 [COMMON] 섹션에서 수동으로 **Trace** 키워드를 0으로 설정하거나 다음 명령을 실행하십시오.

   ```
   db2 UPDATE CLI CFG FOR SECTION COMMON USING Trace 0
   ```

7. (선택사항) 실행 및 추적 중인 응용프로그램을 재시작하십시오.

결과

CLI 추적 파일은 **TracePathName** 키워드에 지정된 경로에 기록됩니다. 파일 이름 형식은 p*pid*t*tid*.cli입니다. 여기서 *pid*는 운영 체제에서 지정한 프로세스 ID이고 *tid*는 응용프로그램 프로세스가 생성한 각 스레드에 대한 숫자 카운터(0에서 시작)입니다(예: p1234t1.cli). IBM Software Support와 공동으로 문제점을 진단 중인 경우, IBM Software Support에서는 추적 경로에 생성된 모든 파일을 필요로 합니다.

CLI 추적 파일에서 입력 및 출력 매개변수 해석

일반 함수의 경우와 마찬가지로 DB2 CLI 함수에는 입력 및 출력 매개변수가 있습니다. DB2 CLI 추적에서 이러한 입력 및 출력 매개변수를 볼 수 있으며, 각 응용프로그램이 특정 CLI API를 호출하는 방법에 대한 세부사항을 제공합니다. CLI 추적에 표시된 대로 CLI 함수의 입력 및 출력 매개변수를 문서의 CLI 참조 섹션의 해당 CLI 함수 정의와 비교할 수 있습니다.

CLI 추적 파일 스니펫은 다음과 같습니다.

```
SQLConnect( hDbc=0:1, szDSN="sample", cbDSN=-3, szUID="",
            cbUID=-3, szAuthStr="", cbAuthStr=-3 )
    ---> Time elapsed - +6.960000E-004 seconds

SQLRETURN   SQLConnect       (
            SQLHDBC           ConnectionHandle,  /* hdbc */
            SQLCHAR      *FAR ServerName,        /* szDSN */
            SQLSMALLINT       NameLength1,       /* cbDSN */
            SQLCHAR      *FAR UserName,          /* szUID */
            SQLSMALLINT       NameLength2,       /* cbUID */
            SQLCHAR      *FAR Authentication,    /* szAuthStr */
            SQLSMALLINT       NameLength3);      /* cbAuthStr */
```

CLI 함수 초기 호출은 입력 매개변수 및 입력 매개변수에 지정되는 값(해당될 경우)을 표시합니다.

CLI 함수는 리턴되면 결과 출력 매개변수를 표시합니다. 예를 들면, 다음과 같습니다.

```
SQLAllocStmt( phStmt=1:1 )
    <--- SQL_SUCCESS   Time elapsed - +4.444000E-003 seconds
```

이 경우, CLI 함수 SQLAllocStmt()는 값이 ″1:1″(연결 핸들 1, 명령문 핸들 1)인 출력 매개변수 phStmt를 리턴합니다.

CLI 추적에서 동적 SQL 분석

DB2 CLI 추적은 SQLPrepare() 및 SQLBindParameter()의 매개변수 표시문자 사용 및 선언을 통해 동적 SQL이 수행되는 방법도 표시합니다. 이를 통해 런타임 시 수행될 SQL문을 판별할 수 있습니다.

다음 추적 항목은 SQL문 준비를 표시합니다(물음표(?) 또는 콜론 다음의 이름(:*name*)은 매개변수 표시문자를 표시함).

```
 SQLPrepare( hStmt=1:1, pszSqlStr=
   "select * from employee where empno = ?",
   cbSqlStr=-3 )
    ---> Time elapsed - +1.648000E-003 seconds
( StmtOut="select * from employee where empno = ?" )
SQLPrepare( )
  <--- SQL_SUCCESS   Time elapsed - +5.929000E-003 seconds
```

다음 추적 항목은 매개변수 표시문자 바인딩을 최대 길이가 7인 CHAR로 표시합니다.

```
SQLBindParameter( hStmt=1:1, iPar=1, fParamType=SQL_PARAM_INPUT,
fCType=SQL_C_CHAR, fSQLType=SQL_CHAR, cbColDef=7, ibScale=0,
 rgbValue=&00854f28, cbValueMax=7, pcbValue=&00858534 )
    ---> Time elapsed - +1.348000E-003 seconds
SQLBindParameter( )
    <--- SQL_SUCCESS   Time elapsed - +7.607000E-003 seconds
```

이제 동적 SQL문이 실행됩니다. rbgValue=″000010″은 런타임 시 매개변수 표시문자를 대체한 값을 표시합니다.

```
SQLExecute( hStmt=1:1 )
    ---> Time elapsed - +1.317000E-003 seconds
( iPar=1, fCType=SQL_C_CHAR, rgbValue="000010" - X"303030303130",
 pcbValue=6, piIndicatorPtr=6 )
    sqlccsend( ulBytes - 384 )
    sqlccsend( Handle - 14437216 )
    sqlccsend( ) - rc - 0, time elapsed - +1.915000E-003
    sqlccrecv( )
    sqlccrecv( ulBytes - 1053 ) - rc - 0, time elapsed - +8.808000E-003
SQLExecute( )
    <--- SQL_SUCCESS   Time elapsed - +2.213300E-002 seconds
```

CLI 추적에서 시간 제어 정보 해석

DB2 CLI 추적에서 시간 제어 정보를 수집하는 몇 가지 방법이 있습니다. 기본적으로, 마지막 CLI API 호출이 특정 스레드에서 이루어졌기 때문에 CLI 추적은 응용프로그램에서 소비한 시간을 캡처합니다.

DB2에서 소비한 시간은 물론 클라이언트와 서버 간의 네트워크 시간도 포함합니다. 예를 들어, 다음과 같습니다.

```
SQLAllocStmt( hDbc=0:1, phStmt=&0012ee48 )
    ---> Time elapsed - +3.964187E+000 seconds
```

(마지막 CLI API가 호출되었기 때문에 이 시간 값은 응용프로그램에서 소비한 시간을 표시함)

```
SQLAllocStmt( phStmt=1:1 )
    <--- SQL_SUCCESS   Time elapsed - +4.444000E-003 seconds
```

(함수가 완료되었기 때문에 이 시간 값은 네트워크 시간을 포함하여 DB2에서 소비한 시간을 표시함)

시간 제어 정보를 캡처하는 다른 방법은 CLI 키워드 TraceTimeStamp를 사용하는 것입니다. 이 키워드는 모든 호출 및 DB2 CLI API 호출 결과에 대해 시간소인을 생성합니다. 이 키워드에는 4개의 표시 옵션(시간소인 정보 없음, 프로세서 틱 및 ISO 시간소인, 프로세서 틱 또는 ISO 시간소인)이 있습니다.

이는 시간 제어 관련 문제점(예: CLI0125E - 함수 시퀀스 오류)에 대한 작업을 할 때 아주 유용할 수 있습니다. 또한 멀티스레드 응용프로그램에 대한 작업을 할 때 먼저 발생한 이벤트를 판별하려고 시도할 때 도움이 될 수 있습니다.

CLI 추적에서 알 수 없는 값 해석

DB2 CLI 함수가 ″알 수 없는 값″을 CLI 추적의 입력 매개변수 값으로 리턴할 수 있습니다.

DB2 CLI 드라이버가 해당 입력 매개변수에 특정한 어떤 것을 찾고 있지만 응용프로그램이 다른 값을 제공하는 경우 이런 상황이 발생할 수 있습니다. 예를 들어, 오래된 CLI 함수 정의를 따르고 있거나 사용되지 않는 CLI 함수를 사용하는 경우 이런 상황이 발생할 수 있습니다.

또한 CLI 함수 호출이 ″옵션 값 변경됨″ 또는 ″키세트 구문 분석기 리턴 코드″를 리턴함을 볼 수 있습니다. 이는 커서가 일부 특정 이유로 정적 커서로 다운그레이드되고 있는 경우처럼 메시지를 표시하는 키세트 커서의 결과입니다.

```
SQLExecDirect( hStmt=1:1, pszSqlStr="select * from org", cbSqlStr=-3 )
    ---> Time elapsed - +5.000000E-002 seconds
( StmtOut="select * from org" )
( COMMIT=0 )
( StmtOut=" SELECT A.TABSCHEMA, ..... )
( StmtOut=" SELECT A.TABSCHEMA, ..... )
( Keyset Parser Return Code=1100 )

SQLExecDirect( )
    <--- SQL_SUCCESS_WITH_INFO   Time elapsed - +1.06E+001 seconds
```

위 CLI 추적에서, 키세트 구문 분석기는 리턴 코드 1100(테이블에 대한 고유 인덱스 또는 기본 키가 없으므로 키세트 커서를 작성할 수 없음을 표시)을 표시했습니다. 이러

한 리턴 코드는 구체화되지 않으므로 리턴 코드의 의미에 대한 추가 정보를 얻으려면 이 시점에서는 IBM Software Support에 문의해야 합니다.

SQLError 또는 SQLDiagRec 호출은 커서 유형이 변경되었음을 표시합니다. 그러면 응용프로그램이 커서 유형 및 동시성을 쿼리하여 변경된 속성을 판별해야 합니다.

멀티스레드 CLI 추적 결과 해석

CLI 추적은 멀티스레드 응용프로그램을 추적할 수 있습니다. 멀티스레드 응용프로그램을 추적하는 최상의 방법은 CLI 키워드 TracePathName을 사용하는 것입니다. 이렇게 하면 p<pid>t<tid>.cli라는 추적 파일이 생성됩니다(여기서 <tid>는 응용프로그램의 실제 스레드 ID임).

실제 스레드 ID를 알아야 하는 경우, CLI 추적 헤더에서 이 정보를 볼 수 있습니다.

```
[ Process: 3500, Thread: 728 ]
[ Date & Time:          02/17/2006 04:28:02.238015 ]
[ Product:              QDB2/NT DB2 v9.1.0.190 ]
...
```

또한 CLI 키워드 TraceFileName을 사용하여 멀티스레드 응용프로그램을 한 파일에 추적할 수 있습니다. 한 스레드의 특정 API를 다른 스레드의 다른 API와 동시에 실행할 수 있기 때문에(추적 검토 시 약간의 혼동을 일으킬 가능성이 있음) 이 메소드는 사용자가 선택한 하나의 파일을 생성하지만 읽기 귀찮을 수 있습니다.

특정 API가 실행된 시간을 살펴 정확한 이벤트 시퀀스를 판별할 수 있으므로 일반적으로 TraceTimeStamp 작동을 권장합니다. 이는 하나의 스레드로 인해 다른 스레드에서 문제점이 발생한 경우 문제점(예: CLI0125E - 함수 시퀀스 오류)을 조사하는 데 아주 유용할 수 있습니다.

플랫폼별 도구

진단 도구(Windows)

Windows 시스템에서 유용한 세 개의 진단 도구를 설명합니다.

Windows 운영 체제에 사용 가능한 진단 도구는 다음과 같습니다.

이벤트 뷰어, 성능 모니터 및 기타 관리 도구

관리 도구 폴더는 이벤트 로그 액세스와 성능 정보 액세스를 포함하여 다양한 진단 정보를 제공합니다.

작업 관리자

작업 관리자는 Windows 서버에서 실행 중인 프로세스를 메모리 사용에 대한 세부사항과 함께 모두 표시합니다. 실행 중인 DB2 프로세스를 찾고 성능 문제

점을 진단하려면 이 도구를 사용하십시오. 이 도구를 사용하면 프로세스의 메모리 사용, 메모리 한계, 사용된 스와퍼 스페이스 및 메모리 누수를 판별할 수 있습니다.

작업 관리자를 열려면, Ctrl+Alt+Delete를 누르고 사용 가능한 옵션에서 **작업 관리자**를 누르십시오.

Dr. Watson

Dr. Watson 유틸리티는 GPF(General Protection Fault)의 경우 호출됩니다. 문제점 진단에 도움이 될 수 있는 데이터를 로그하고 이 정보를 파일에 저장합니다. 명령행에서 `drwatson`을 입력하여 이 유틸리티를 시작해야 합니다.

진단 도구(Linux 및 UNIX)

이 섹션에서는 Linux 및 UNIX 플랫폼에서 문제점 해결 및 성능 모니터링을 위한 일부 필수 명령을 설명합니다.

이러한 명령 중 하나에 대한 자세한 내용을 보려면 명령행에서 앞에 ″man″을 붙이십시오. 시스템에서 발생하고 있는 문제점의 원인을 식별하는 데 도움을 줄 수 있는 데이터를 수집하고 처리하려면 이러한 명령을 사용하십시오. 데이터가 수집되면, 문제점에 친숙한 누군가가 조사하거나 요청이 있을 경우 IBM Software Support에 제공할 수 있습니다.

문제점 해결 명령(AIX)

DB2 문제점 해결에 유용한 AIX 시스템 명령은 다음과 같습니다.

errpt errpt 명령은 하드웨어 오류 및 네트워크 장애와 같은 시스템 오류를 보고합니다.

- 오류당 한 라인을 표시하는 개요를 보려면 errpt를 사용하십시오.
- 각 오류에 대해 한 페이지를 표시하는 자세한 보기는 errpt -a를 사용하십시오.
- 오류 번호가 ″1581762B″인 오류의 경우, errpt -a -j 1581762B를 사용하십시오.
- 과거에 페이징 스페이스가 부족했는지 여부를 확인하려면 errpt | grep SYSVMM을 사용하십시오.
- 토큰링 카드 또는 디스크 문제점이 있는지 여부를 확인하려면 ″disk″ 및 ″tr0″ 구문에 대한 errpt 출력을 확인하십시오.

lsps lsps -a 명령은 페이징 공간 사용 방법을 모니터하고 표시합니다.

lsattr 이 명령은 여러 운영 체제 매개변수를 표시합니다. 예를 들어, 데이터베이스 파티션에서 실제 메모리 양을 확인하려면 다음 명령을 사용하십시오.

```
lsattr -l sys0 -E
```

xmperf

Motif를 사용하는 AIX 시스템의 경우, 이 명령은 시스템 관련 성능 데이터를 수집하고 표시하는 그래픽 모니터를 시작합니다. 모니터는 각 데이터베이스 파티션에 대한 3차원 다이어그램을 하나의 창에 표시하며, 상위 레벨 모니터링에 유용합니다. 그러나 활동이 저조한 경우 이 모니터의 출력은 제한된 값입니다.

spmon

시스템 파티셔닝을 PSSP(Parallel System Support Program)의 일부로 사용 중인 경우, 모든 워크스테이션에서 SP 스위치가 실행 중인지 여부를 확인해야 합니다. 모든 데이터베이스 파티션의 상태를 보려면, CWS(Control Workstation)에서 다음 명령 중 하나를 사용하십시오.

- ASCII 출력의 경우 spmon -d
- 그래픽 사용자 인터페이스의 경우 spmon -g

또는 데이터베이스 파티션 워크스테이션에서 netstat -i 명령을 사용하여 스위치가 작동 중지 상태인지 여부를 확인하십시오. 스위치가 작동 중지 상태이면 데이터베이스 파티션 이름 옆에 별표(*)가 있습니다. 예:

```
css0* 65520 <Link>0.0.0.0.0.0
```

스위치가 작동 중이면 별표가 표시되지 않습니다.

문제점 해결 명령(Linux 및 UNIX)

다음 시스템 명령은 별도로 명시하지 않으면 AIX를 포함하여 모든 Linux 및 UNIX 시스템 명령에 해당합니다.

df df 명령을 사용하여 파일 시스템이 가득 찼는지 여부를 확인할 수 있습니다.

- 모든 파일 시스템(마운트된 파일 시스템 포함)에서 여유 공간 양을 확인하려면 df를 사용하십시오.
- ″dev″가 포함된 이름을 가진 모든 파일 시스템에서 여유 공간 양을 확인하려면 df | grep dev를 사용하십시오.
- 홈 파일 시스템에서 여유 공간 양을 확인하려면 df /home을 사용하십시오.
- 파일 시스템 ″tmp″에서 여유 공간 양을 확인하려면 df /tmp를 사용하십시오.
- 시스템에 여유 공간이 충분한지 여부를 확인하려면 df /usr , df /var , df /tmp 및 df /home 명령의 출력을 확인하십시오.

truss 이 명령은 하나 이상의 프로세스에서 시스템 호출을 추적하는 데 유용합니다.

pstack

Solaris 2.5.1 이상에 사용할 수 있으며, /usr/proc/bin/pstack 명령은 스택 역추적 정보를 표시합니다. /usr/proc/bin 디렉토리는 일시중단된 것으로 보이는 프로세스 디버깅을 위한 기타 도구를 포함합니다.

성능 모니터링 도구

시스템 성능 모니터링에 사용 가능한 도구는 다음과 같습니다.

vmstat

이 명령은 뭔가 일시중단되었거나 오랜 시간이 걸리는지 여부를 판별하는 데 유용합니다. 페이지 인(pi) 및 페이지 아웃(po) 컬럼에 있는 페이징 비율을 모니터할 수 있습니다. 기타 중요한 컬럼은 할당된 가상 스토리지(avm) 및 여유 가상 스토리지(fre)의 양입니다.

iostat 이 명령은 I/O 활동을 모니터링하는 데 유용합니다. 읽기 및 쓰기 비율을 사용하여 특정 SQL 조작(시스템에서 유일한 활동인 경우)에 필요한 시간 양을 추정할 수 있습니다.

netstat

이 명령을 사용하여 각 데이터베이스 파티션의 네트워크 트래픽 및 발생한 오류 패킷 수를 알 수 있습니다. 네트워크 문제점을 분리하는 데 유용합니다.

시스템 파일

Solaris 운영 체제에 사용할 수 있으며 `/etc/system` 파일은 커널 구성 한계에 대한 정의(예: 시스템에서 동시에 허용되는 최대 사용자 수, 사용자당 최대 프로세스 수 및 자원 크기와 수에 대한 프로세스간 통신(IPC) 한계)를 포함합니다. 이러한 한계는 Solaris 운영 체제 시스템에서 DB2 성능에 영향을 주기 때문에 중요합니다.

제 6 장 DB2 데이터베이스 문제점 해결

일반적으로 문제점 해결 프로세스에는 문제점을 분리 및 식별한 후 해결책을 찾아야 합니다. 이 섹션에서는 DB2 제품의 특정 기능과 관련된 문제점 해결 정보를 제공합니다.

일반적인 문제점이 식별되면, 찾은 내용이 점검목록 양식으로 이 섹션에 추가됩니다. 점검목록을 통해 해결책을 찾을 수 없으면, 추가 진단 데이터를 수집하여 직접 분석하거나 분석을 위해 IBM Software Support에 데이터를 제출할 수 있습니다.

다음 질문을 통해 적절한 문제점 해결 태스크를 수행할 수 있습니다.

1. 알려진 FixPack을 모두 적용했습니까? 그렇지 않다면, *DB2 Server 설치*의 『FixPack 적용』을 고려하십시오.
2. 다음을 수행할 때 문제점이 발생합니까?
   - DB2 데이터베이스 서버 또는 클라이언트 설치. 그렇다면, 이 책의 『설치 문제점에 대한 데이터 수집』 주제를 참조하십시오.
   - 인스턴스 또는 DB2 Administration Server(DAS) 작성, 삭제, 갱신 또는 업그레이드. 그렇다면, 이 책의 『DAS 및 인스턴스 관리 문제점에 대한 데이터 수집』 주제를 참조하십시오.
   - EXPORT, IMPORT, LOAD 또는 db2move 명령을 사용하여 데이터 이동. 그렇다면, 이 책의 『데이터 이동 문제점에 대한 데이터 수집』 주제를 참조하십시오.

문제점이 이러한 범주 중 하나에 해당하지 않는 경우에는 IBM Software Support에 문의하는 경우 기본 진단 데이터가 여전히 필요할 수 있습니다.

DB2에 대한 데이터 수집

단순히 증상을 해결하여 문제점을 해결할 수 없는 경우가 있습니다. 이러한 경우, 진단 데이터를 수집해야 합니다. 수집해야 하는 진단 데이터와 해당 데이터를 수집하는 소스는 조사 중인 문제점 유형에 따라 다릅니다. 다음 단계는 IBM Software Support에 문제점을 제출할 때 일반적으로 제공해야 하는 기본 정보 세트 수집 방법을 나타냅니다.

가장 완벽한 출력을 확보하려면, 인스턴스 소유자가 db2support 유틸리티를 호출해야 합니다.

기본 진단 정보 세트를 압축된 파일 아카이브로 수집하려면, db2support 명령을 입력하십시오.

```
db2support <output_directory> -s -d <database_name> -c
```

-s를 사용하면 사용된 하드웨어 및 운영 체제에 대한 시스템 세부사항이 제공됩니다. -d를 사용하면 지정된 데이터베이스에 대한 세부사항이 제공됩니다. -c를 사용하면 지정된 데이터베이스에 연결 시도가 허용됩니다.
출력은 편리하게 수집되고 압축된 ZIP 아카이브(db2support.zip)로 저장되므로 모든 시스템에서 쉽게 전송 및 추출할 수 있습니다.

특정 증상 또는 제품의 특정 파트 문제점의 경우, 추가 데이터를 수집해야 합니다. 문제점별 "데이터 수집" 문서를 참조하십시오.

그 다음에는 다음 태스크를 수행할 수 있습니다.

- 데이터 분석
- IBM Software Support에 데이터 제출

데이터 이동 문제점에 대한 데이터 수집

데이터 이동 명령을 수행하는 동안 문제점이 발생하고 문제점의 원인을 판별할 수 없는 경우, 직접 또는 IBM Software Support에서 문제점을 진단 및 해결하는 데 사용할 수 있는 진단 데이터를 수집하십시오.

발생하는 상황에 맞게 다음 목록의 데이터 콜렉션 지시사항을 따르십시오.

- db2move 명령과 관련된 문제점에 대한 데이터를 수집하려면, 명령을 실행한 디렉토리로 이동하십시오. 명령에 지정한 조치에 따라 다음 파일을 찾으십시오.
  - COPY 조치의 경우, COPY.*timestamp*.ERR 및 COPYSCHEMA.*timestamp*.MSG 파일을 찾으십시오. LOAD_ONLY 또는 DDL_AND_LOAD 모드도 지정한 경우, LOADTABLE.*timestamp*.MSG 파일도 찾으십시오.
  - EXPORT 조치의 경우, EXPORT.out 파일을 찾으십시오.
  - IMPORT 조치의 경우, IMPORT.out 파일을 찾으십시오.
  - LOAD 조치의 경우, LOAD.out 파일을 찾으십시오.
- EXPORT, IMPORT 또는 LOAD 명령과 관련된 문제점에 대한 데이터를 수집하려면, 명령에 MESSAGES 매개변수가 포함되었는지 여부를 판별하십시오. 포함되었다면, 출력 파일을 수집하십시오. 별도로 지정하지 않으면 이러한 유틸리티는 현재 디렉토리 및 디폴트 드라이브를 대상으로 사용합니다.
- REDISTRIBUTE 명령과 관련된 문제점에 대한 데이터를 수집하려면, *"databasename.database_partition_groupname. timestamp"* (Linux 및 UNIX의 경우) 및 *"databasename. database_partition_groupname.date.time"*(Windows의 경우) 파일을 찾으십시오. 이 파일은 각각 *$HOME*/sqllib/db2dump 디렉토리 또는 $DB2PATH₩sqllib₩redist(여기서 *$HOME*은 인스턴스 소유자이 홈 디렉토리임)에 있습니다.

DAS 및 인스턴스 관리 문제점에 대한 데이터 수집

DAS(DB2 Administration Server) 또는 인스턴스 관리를 수행하는 동안 문제점이 발생하고 문제점의 원인을 판별할 수 없는 경우, 직접 또는 IBM Software Support에서 문제점을 진단 및 해결하는 데 사용할 수 있는 진단 데이터를 수집하십시오.

다음 단계는 문제점을 재현할 수 있고 Linux 또는 UNIX에서 DB2를 사용 중인 상황에만 해당됩니다.

DAS 또는 OR 관리 문제점에 대한 데이터를 수집하려면 다음을 수행하십시오.

1. 추적 또는 디버그 모드를 사용 가능하게 한 상태에서 실패한 명령을 반복하십시오. 명령 예:

   ```
   db2setup -t trace.out
   dascrt -u DASUSER -d
   dasdrop -d
   dasmigr -d
   dasupdt -d
   db2icrt -d INSTNAME
   db2idrop INSTNAME -d
   db2iupgrade -d INSTNAME
   db2iupdt -d INSTNAME
   ```

2. 진단 파일을 찾으십시오. 두 개 이상의 파일이 있을 수 있으므로 시간소인을 비교하여 모든 해당 파일을 확보하고 있는지 확인하십시오.

   출력은 디폴트로 /tmp 디렉토리에 있습니다.

   파일 이름 예: dascrt.log, dasdrop.log , dasupdt.log , db2icrt.log.PID, db2idrop.log.PID, db2iupgrade.log.PID 및 db2iupdt.log.PID(여기서 PID는 프로세스 ID임).

3. IBM Software Support에 진단 파일을 제공하십시오.

db2start 또는 START DATABASE MANAGER 명령 실패가 문제점인 경우, insthome/sqllib/log 디렉토리(여기서 insthome은 인스턴스 소유자의 홈 디렉토리임)에서 db2start.*timestamp*.log 파일을 찾으십시오. 마찬가지로, db2stop 또는 STOP DATABASE MANAGER 명령 실패가 문제점인 경우에는 db2stop.*timestamp*.log 파일을 찾으십시오. 데이터베이스 관리 프로그램이 **start_stop_time** 데이터베이스 관리 프로그램 구성 매개변수에 지정된 시간 내에 명령에 응답하지 못한 경우에만 이러한 파일을 찾을 수 있습니다.

DB2에 대한 데이터 분석

데이터를 수집한 후에는 해당 데이터가 특정 문제점을 해결하는 데 어떻게 도움을 줄 수 있는지 판별해야 합니다. 분석 유형은 조사 중인 문제점 유형과 수집한 데이터에 따라 다릅니다. 다음 단계는 기본 DB2 진단 데이터 조사를 시작하는 방법을 나타냅니다.

진단 데이터를 분석하려면 다음 조치를 취하십시오.

- 데이터의 여러 조각이 서로 어떻게 관련되는지 명확히 이해하십시오. 예를 들어, 데이터가 두 개 이상의 시스템에 걸쳐 있는 경우 어떤 데이터 조각이 어떤 소스에서 오는지 알 수 있도록 잘 구성된 데이터를 보존하십시오.
- 시간소인을 검사하여 진단 데이터의 각 조각이 문제점 시간 제어와 관련되는지 확인하십시오. 소스가 서로 다른 데이터는 시간소인 형식이 다를 수 있음에 유의하십시오. 서로 다른 이벤트가 발생한 시기를 알 수 있도록 각 시간소인 형식의 서로 다른 요소의 시퀀스를 이해하십시오.
- 문제점에 대한 정보를 포함할 가능성이 가장 높은 데이터 소스를 판별하고 거기서 분석을 시작하십시오. 예를 들어, 문제점이 설치와 관련된 경우 일반 제품 또는 운영체제 로그 파일에서 시작하지 말고 설치 로그 파일(있는 경우)에서 분석을 시작하십시오.
- 지정된 분석 방법은 각 데이터 소스에 고유하지만, 대부분의 추적 및 로그 파일에 적용 가능한 하나의 추가 정보는 문제점이 발생하는 데이터의 위치를 식별하는 것으로 시작하는 것입니다. 해당 위치를 식별한 후에는 데이터 전반에서 시간을 뒤로 돌려 작업하여 문제점의 근본 원인을 해결할 수 있습니다.
- 작동 중인 환경과 작동하고 있지 않은 환경의 비교 데이터가 있는 문제점을 조사 중인 경우, 각 환경에 대한 운영 제체 및 제품 구성 세부사항을 비교하는 것으로 시작하십시오.

잠금 문제점 진단 및 해결

잠금 문제점을 해결하려면, SQL 쿼리 성능 저하 또는 쿼리 완료 실패를 일으키는 잠금 이벤트 유형 및 관련된 SQL문 또는 명령문을 진단하는 것으로 시작해야 합니다. 여기서는 잠금 문제점 유형 진단 시 도움이 되는 단계와 잠금 문제점 해결을 돕기 위해 취할 수 있는 단계를 제공합니다.

개요

잠금으로 인해 응용프로그램이 태스크를 완료하지 못하거나 SQL 쿼리 성능이 저하되는 경우 잠금 문제점은 적절한 진단입니다. 따라서 이상적인 목표는 데이터베이스 시스템에서 잠금 시간종료 또는 교착 상태를 갖지 않는 것입니다(둘 다 응용프로그램이 태스크를 완료하지 못하게 됨).

잠금 대기는 일반적인 예측 이벤트이지만, 잠금을 기다리는 데 소비한 시간이 길어지면 잠금 대기로 인해 SQL 쿼리 성능과 응용프로그램 완료가 느려질 수 있습니다. 잠금 대기 지속 시간이 지나치게 길면 응용프로그램이 태스크를 완료하지 못하게 되는 잠금 시간종료를 초래할 위험이 있습니다.

잠금 에스컬레이션은 잠금 시간종료를 일으키는 경우 잠금 문제점으로 간주됩니다. 이상적인 목표는 잠금 에스컬레이션을 갖지 않는 것이지만, 역효과가 발생하지 않는다면 적은 수의 잠금은 허용할 수 있습니다.

잠금 대기, 잠금 시간종료 및 교착 상태 잠금 이벤트를 항상 모니터하는 것이 좋습니다 (일반적으로 잠금 대기의 경우 워크로드 레벨에서, 잠금 시간종료 및 교착 상태의 경우 데이터베이스 레벨에서).

발생하는 잠금 문제점 유형 진단 및 해결책은 정보 수집 및 진단 표시기 검색으로 시작합니다. 다음 조치는 이 프로세스를 안내합니다.

정보 수집

일반적으로, 시스템이 처리 지연 및 성능 저하를 포함할 수 있는 비정상 동작을 보이는지 객관적으로 평가할 수 있으려면 시스템의 일반적인 동작(기준선)을 설명하는 정보가 있어야 합니다. 그런 다음, 예측된 비정상 동작 관찰 내용과 기준선 간에 비교를 할 수 있습니다. 정기적인 운영 모니터링 태스크를 스케줄링하여 기준선 데이터를 수집하는 것은 문제점 해결 프로세스의 주요 구성요소입니다. 시스템의 기준선 운영 설정에 대한 자세한 정보는 13 페이지의 『시스템 성능 운영 모니터링』을 참조하십시오.

SQL 쿼리 성능 저하 또는 쿼리 완료 실패의 이유가 되는 잠금 문제점 유형을 확인하려면 관련된 잠금 이벤트 유형, 이 잠금을 요청하거나 보류 중인 응용프로그램, 이 이벤트 동안 응용프로그램이 수행하던 작업, 현저한 성능 저하에 관련된 SQL문 또는 명령문을 식별하는 데 도움이 되는 정보를 수집해야 합니다.

잠금 이벤트 모니터 작성, 테이블 함수 사용 또는 db2pd 명령 사용을 통해 이러한 유형의 정보를 수집할 수 있습니다. 잠금 이벤트 모니터가 수집하는 정보를 다음 세 개의 기본 범주로 구분할 수 있습니다.

- 해당 잠금에 대한 정보
- 이 잠금 및 현재 활동을 요청하는 응용프로그램에 대한 정보. 교착 상태의 경우, 이는 희생(victim)이라는 명령문에 대한 정보입니다.
- 잠금 및 현재 활동을 소유한 응용프로그램에 대한 정보. 교착 상태의 경우, 이는 구성원(participant)이라는 명령문에 대한 정보입니다.

잠금 대기, 잠금 시간종료, 교착 상태 잠금 이벤트 모니터 방법에 대한 지시사항은 *데이터베이스 모니터링 안내서 및 참조서*의 『잠금 이벤트 모니터링』을 참조하십시오.

진단 표시기 검색

잠금 이벤트 모니터, 테이블 함수 또는 db2pd 명령 실행을 통해 잠금 문제점의 특징을 분리하는 데 도움을 줄 수 있는 정보를 수집할 수 있습니다. 특히, 다음 주제에는 발생하고 있는 특정 유형의 잠금 문제점을 진단 및 확인하는 데 도움이 되는 진단적 지시 정보가 포함되어 있습니다.

- 대기 시간이 길고 잠금 시간종료가 발생하지 않는다면, 잠금 대기 문제점일 가능성이 있습니다. 확인하려면 잠금 대기 문제점 진단을 참조하십시오.
- 기준 수 이상으로 교착 상태 수가 늘어나고 있으면, 교착 상태 문제점일 가능성이 있습니다. 확인하려면 교착 상태 문제점 진단을 참조하십시오.
- 잠금 시간종료 수가 늘어나고 **locktimeout** 데이터베이스 구성 매개변수가 0이 아닌 시간 값으로 설정된 경우, 잠금 시간종료 문제점일 가능성이 있습니다. 확인(잠금 대기 문제점도 고려)하려면 잠금 시간종료 문제점 진단을 참조하십시오.
- 일반적인 잠금 대기 수보다 많고 잠금 이벤트 모니터가 잠금 에스컬레이션이 발생하고 있음을 표시(Yes)하는 경우, 잠금 에스컬레이션 문제점일 가능성이 있습니다. 확인하려면 잠금 에스컬레이션 문제점 진단을 참조하십시오.

잠금 대기 문제점 진단

트랜잭션이 다른 트랜잭션이 이미 보유한 자원에 대한 잠금을 확보하려고 하면 잠금 대기가 발생합니다. 잠금 대기 시간의 지속기간이 연장되면, SQL 쿼리 실행 속도가 저하됩니다. 오랜 또는 예기치 않은 잠금 대기 시간이 발생하고 잠금 시간종료가 없으면 잠금 대기 문제점일 가능성이 있습니다.

시작하기 전에

일반적으로, 시스템이 처리 지연 및 성능 저하를 포함할 수 있는 비정상 동작을 보이는지 객관적으로 평가할 수 있으려면 시스템의 일반적인 동작(기준선)을 설명하는 정보가 있어야 합니다. 그런 다음, 예측된 비정상 동작 관찰 내용과 기준선 간에 비교를 할 수 있습니다. 정기적인 운영 모니터링 태스크를 스케줄링하여 기준선 데이터를 수집하는 것은 문제점 해결 프로세스의 주요 구성요소입니다. 시스템의 기준선 운영 설정에 대한 자세한 정보는 13 페이지의 『시스템 성능 운영 모니터링』을 참조하십시오.

잠금 대기 잠금 이벤트 모니터 방법에 대한 지시사항은 *데이터베이스 모니터링 안내서 및 참조서*의 『잠금 이벤트 모니터링』을 참조하십시오.

프로시저

진단 하나의 트랜잭션(하나 이상의 SQL문으로 작성됨)이 다른 트랜잭션이 보유한 잠금과 모드가 충돌하는 잠금을 획득하려고 시도하면 잠금 대기가 발생합니다. 지나친 잠금 대기 시간은 종종 부족한 응답 시간을 나타내므로 잠금 대기 시간을 모니터하는 것이 중요합니다. 단일 트랜잭션의 잠금 대기 시간은 일반적으로 아주 짧으며 정규화된 측정은 처리하기 쉽기 때문에 잠금 대기 시간 양은 천 개의 트랜잭션으로 최적으로 정규화됩니다.

각각의 진단을 확인하려고 시도할 때 고려해야 하는 잠금 대기 품질은 서로 다릅니다. 다음은 서로 다른 세 개의 잠금 대기 품질 목록과 최적의 진단 방입니다.

- 긴 개별 잠금 대기

- 서비스 클래스 및 워크로드로부터 최대 잠금 대기 시간을 점검하십시오. 워크로드에 잠금 이벤트 모니터를 설정하여 해당 값을 확보하십시오.

- 잠금 대기 시간은 길지만 짧은 개별 잠금 대기
  - 일반적으로 잠금 호위의 결과입니다. db2pd -locks wait 명령을 사용하여 대기 체인을 발견하십시오.
- 대기 중인 잠금 유형
  - 이를 검사하면 문제점을 판별하는 데 도움이 될 수 있습니다. 잠금 대기 중인 에이전트를 찾아 잠금 유형에 대한 정보를 가져오십시오. 잠금 유형 정보를 사용하여 명백한 무언가가 발생되고 있는지 여부를 판별하십시오. 예를 들어, 패키지 잠금은 BIND/REBIND 명령 또는 DDL이 해당 패키지의 사용자와 충돌함을 표시할 수 있습니다. 내부 c(카탈로그 캐시) 잠금은 DDL이 명령문 컴파일과 충돌함을 표시할 수 있습니다.

표시 징후

다음 잠금 대기 표시 징후를 찾으십시오.

- 잠금 대기 수가 증가함(**lock_waits** 모니터 요소 값 증가)
- 잠금 대기 중인 활성 에이전트 비율이 높음(예를 들어, 전체 활성 에이전트의 20% 이상). 이 정보를 얻는 방법에 대한 정보는 다음 섹션(『모니터할 사항』)을 참조하십시오.
- 데이터베이스 또는 워크로드 레벨에서 캡처한 잠금 대기 기간 값(**lock_wait_time** 모니터 요소)이 증가함

모니터할 사항

기타 많은 DB2 모니터 데이터 유형과 달리, 잠금 정보는 매우 일시적입니다. 전체로 실행되는 **lock_wait_time**을 제외하고, 대부분의 다른 잠금 정보는 잠금이 해제되면 사라집니다. 따라서, 잠금 및 잠금 대기 이벤트 데이터는 전개도를 더 잘 이해할 수 있도록 일정 기간 동안 주기적으로 수집된 경우에 가장 유용합니다.

잠금 대기 중인 활성 에이전트에 대한 정보를 수집하려면 WLM_GET_SERVICE_CLASS_AGENTS_V97 테이블 함수를 사용하십시오. 잠금 대기 중인 에이전트는 다음 속성-값 쌍을 가진 에이전트로 표시됩니다.

- EVENT_OBJECT = LOCK
- EVENT_TYPE = ACQUIRE

또한 응용프로그램 스냅샷, 잠금 관리 뷰 또는 db2pd -wlocks 명령의 잠금 대기 옵션을 사용하여 잠금 대기 중인 활성 에이전트에 대한 정보를 얻을 수 있습니다.

주요 표시기 모니터링 요소는 다음과 같습니다.

- **lock_waits** 값이 증가함
- **lock_wait_time** 값이 김

여기 나열된 표시 징후 중 하나 이상을 관찰한 경우, 잠금 대기 문제점일 가능성이 있습니다. 『다음에 수행할 일』 섹션의 링크를 따라 이 문제점을 해결하십시오.

다음 단계

잠금 대기로 인해 문제점이 발생하고 있음을 진단한 후에는 문제점을 해결하기 위한 단계를 수행하십시오(『잠금 대기 문제점 해결』).

잠금 대기 문제점 해결

잠금 대기 문제점을 진단한 후, 다음 단계는 응용프로그램이 너무 오래 잠금을 기다려야 하기 때문에 발생하는 문제점을 해결하려고 시도하는 것입니다. 여기 제공된 지침은 잠금 대기 문제점을 해결하는 데 도움을 주고 향후 이러한 사건이 발생하지 않도록 예방할 수 있습니다.

시작하기 전에

520 페이지의 『잠금 문제점 진단 및 해결』에 설명된 잠금 문제점에 대한 필수 진단 단계를 수행하여 잠금 대기 문제점이 발생하고 있음을 확인하십시오.

이 태스크에 대한 정보

여기 제공된 지침은 발생하고 있는 잠금 대기 문제점을 해결하는 데 도움을 주고 향후 이러한 사건이 발생하지 않도록 예방할 수 있습니다.

프로시저

다음 단계를 사용하여 허용할 수 없는 잠금 대기 문제점의 원인을 진단하고 교정 방법을 적용하십시오.

1. 에이전트가 잠금을 대기하는 데 오랜 시간을 소비하고 있는 모든 테이블에 대한 관리 통지 로그에서 정보를 얻으십시오.
2. 관리 통지 로그의 정보를 사용하여 잠금 대기 문제점 해결 방법을 결정하십시오. 잠금 경합 및 잠금 대기 시간을 줄이는 데 도움이 되는 많은 지침이 있습니다. 다음 옵션을 고려하십시오.
   - 가능한 경우, 매우 긴 트랜잭션 및 WITH HOLD 커서를 피하십시오. 잠금이 길게 보유될수록 다른 응용프로그램과의 경쟁이 발생할 가능성이 더 커집니다. 이는 높은 분리 수준을 사용 중인 경우에만 문제가 됩니다.
   - 가능한 빨리 다음 조치를 커미트하는 것이 우수 사례입니다.
     – 쓰기 조치(예: 삭제, 삽입 및 갱신)

  - DDL(Data Definition Language)문(예: ALTER, CREATE 및 DROP문)
  - BIND 및 REBIND 명령
- ALTER 또는 DROP DDL문을 실행한 후, SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS 프로시저를 실행하여 데이터 오브젝트의 유효성을 다시 확인하고 db2rbind 명령을 실행하여 패키지를 리바인드하십시오.
- 특히 반복 읽기(RR) 분리 수준 하에서, 필요한 것보다 더 큰 결과 세트를 페치하지 않도록 하십시오. 행을 더 많이 건드릴 수록 잠금이 더 많이 보유되고 그 밖의 누군가가 보유하는 잠금에 부딪히게 될 가능성이 더 커집니다. 실질적으로, 이것은 종종 더 많은 행을 다시 가져와 응용프로그램에서 필터링하기보다는 SELECT문의 WHERE 절로 행 선택 기준을 푸시다운하는 것을 의미합니다. 예를 들어, 다음과 같습니다.

```
exec sql declare curs for
  select c1,c2 from t
  where c1 not null;
exec sql open curs;
do {
  exec sql fetch curs
    into :c1, :c2;
} while( P(c1) != someVar );

==>

exec sql declare curs for
  select c1,c2 from t
  where c1 not null
  and myUdfP(c1) = :someVar;
exec sql open curs;
exec sql fetch curs
    into :c1, :c2;
```

- 필요한 것보다 더 높은 분리 수준을 사용하지 않도록 하십시오. 응용프로그램에서 결과 세트 무결성을 보존하기 위해 반복 읽기가 필요할 수도 있지만, 이는 잠금 보유 및 잠재적 잠금 충돌의 견지에서 추가 비용을 초래합니다.
- 응용프로그램의 비즈니스 논리에 적합한 경우, **DB2_EVALUNCOMMITTED**, **DB2_SKIPDELETED** 및 **DB2_SKIPINSERTED** 레지스트리 변수를 통한 잠금 동작 수정을 고려하십시오. 이러한 레지스트리 변수는 DB2 데이터베이스 관리 프로그램이 일부 상황에서 잠금 사용을 지연시키거나 피할 수 있게 하므로, 경쟁이 줄어들고 잠재적으로 처리량이 향상됩니다.
- 가능한 경우 잠금 에스컬레이션을 제거하십시오.

다음 단계

응용프로그램을 재실행하여 관리 통지 로그에서 잠금 관련 항목을 검사하거나 해당 워크로드, 연결, 서비스 서브클래스, 작업 단위(UOW) 및 활동 레벨에 대한 잠금 대기 및 잠금 대기 시간 메트릭을 검사하여 잠금 문제점이 제거되었는지 확인하십시오.

교착 상태 문제점 진단

두 응용프로그램이 상대방이 필요로 하는 데이터를 잠근 경우 교착 상태가 작성되어 교착 상태 검출기의 개입이 없으면 어느 응용프로그램도 계속 실행할 수 없는 상황이 발생합니다. 교착 상태는 교착 상태 발견을 기다리는 동안 구성원(participant) 트랜잭션의 속도를 저하시키며, 희생(victim) 트랜잭션을 롤백하여 시스템 자원을 낭비하고 전체 프로세스 중에 추가 시스템 작업 및 트랜잭션 로그 액세스를 초래합니다. 기준 수 이상으로 교착 상태 수가 늘어나고 트랜잭션이 재실행되고 있으면 교착 상태 문제점일 가능성이 있습니다.

시작하기 전에

일반적으로, 관찰된 교착 상태는 비정상으로 간주됩니다. 시스템이 처리 지연 및 성능 저하를 포함할 수 있는 비정상 동작을 보이는지 객관적으로 평가할 수 있으려면 시스템의 일반적인 동작(기준선)을 설명하는 정보가 있어야 합니다. 그런 다음, 예측된 비정상 동작 관찰 내용과 기준선 간에 비교를 할 수 있습니다. 정기적인 운영 모니터링 태스크를 스케줄링하여 기준선 데이터를 수집하는 것은 문제점 해결 프로세스의 주요 구성요소입니다. 시스템의 기준선 운영 설정에 대한 자세한 정보는 13 페이지의 『시스템 성능 운영 모니터링』을 참조하십시오.

교착 상태 잠금 이벤트 모니터 방법에 대한 지시사항은 *데이터베이스 모니터링 안내서 및 참조서*의 『잠금 이벤트 모니터링』을 참조하십시오.

프로시저

진단 두 응용프로그램이 상대방이 필요로 하는 데이터를 잠근 경우 교착 상태가 작성되어 교착 상태 검출기의 개입이 없으면 어느 응용프로그램도 계속 실행할 수 없는 상황이 발생합니다. 희생(victim) 응용프로그램은 시스템이 이전 교착 상태 트랜잭션을 자동으로 롤백한 후에 처음부터 트랜잭션을 재실행해야 합니다. 이러한 발생 비율을 모니터링하면 많은 교착 상태가 DBA에서 인식하지 않은 상태에서 시스템에 상당한 추가 로드를 주는 경우를 방지하는 데 도움이 됩니다.

표시 징후

다음 교착 상태 표시 징후를 찾으십시오.

- 하나 이상의 응용프로그램이 때때로 트랜잭션을 재실행함
- 관리 통지 로그의 교착 상태 메시지 항목
- **deadlocks** 모니터 요소에 대해 표시되는 교착 상태 수 증가
- **int_deadlock_rollbacks** 모니터 요소에 대해 표시되는 롤백 수 증가
- 에이전트가 로그 레코드를 디스크로 플러시하기를 기다리는 데 소비하는 시간 양(**log_disk_wait_time** 모니터 요소에 대해 표시됨) 증가

모니터할 사항

교착 상태의 비용은 다양하지만, 롤백된 트랜잭션의 길이에 정비례합니다. 그래도 교착 상태는 일반적으로 문제점을 표시합니다.

본질적으로 교착 상태 이벤트를 발견하기 위한 세 가지 접근 방법이 있습니다.

1. 잠금 이벤트 모니터를 설정하고 데이터베이스 전반에서 발생하는 모든 교착 상태 이벤트에 대한 세부사항을 캡처하도록 **mon_deadlock** 데이터베이스 구성 매개변수 설정
2. 관리 통지 로그에서 교착 상태 메시지 및 이에 수반되는 기본 정보 모니터
3. 테이블 함수를 통해 표시기 모니터 요소 모니터

대부분의 사용자는 첫 번째 접근 방법을 채택합니다. 주요 표시기 모니터 요소를 모니터링하여 교착 상태가 발생하는 시기를 발견한 후 사용자는 이벤트 모니터가 수집한 정보를 검사하여 자세한 정보를 얻을 수 있습니다.

주요 표시기 모니터링 요소는 다음과 같습니다.

- **deadlocks** 값이 0이 아님
- **int_deadlock_rollbacks**가 교착 상태 이벤트로 인한 롤백 수 증가를 표시
- **log_disk_wait_time**이 에이전트가 로그 레코드를 디스크로 플러시하기를 기다리는 데 소비하는 시간 양 증가를 표시

여기 나열된 표시 징후 중 하나 이상을 관찰한 경우, 교착 상태 문제점일 가능성이 있습니다. 『다음에 수행할 일』 섹션의 링크를 따라 이 문제점을 해결하십시오.

다음 단계

교착 상태로 인해 문제점이 발생하고 있음을 진단한 후에는 문제점을 해결하기 위한 단계를 수행하십시오(『교착 상태 문제점 해결』).

교착 상태 문제점 해결

교착 상태 문제점을 진단한 후, 다음 단계는 동시에 실행 중인 두 개의 응용프로그램(다른 응용프로그램이 필요로 하는 자원을 각각 잠금) 간에 발생하는 교착 상태 문제점을 해결하려고 시도하는 것입니다. 여기 제공된 지침은 발생하고 있는 교착 상태 문제점을 해결하는 데 도움을 주고 향후 이러한 사건이 발생하지 않도록 예방할 수 있습니다.

시작하기 전에

520 페이지의 『잠금 문제점 진단 및 해결』에 설명된 잠금 문제점에 대한 필수 진단 단계를 수행하여 교착 상태 문제점이 발생하고 있음을 확인하십시오.

이 태스크에 대한 정보

여기 제공된 지침은 발생하고 있는 교착 상태 문제점을 해결하는 데 도움을 주고 향후 이러한 사건이 발생하지 않도록 예방할 수 있습니다.

프로시저

다음 단계를 사용하여 허용할 수 없는 교착 상태 문제점의 원인을 진단하고 교정 방법을 적용하십시오.

1. 에이전트에 교착 상태가 발생하고 있는 모든 테이블에 대한 관리 통지 로그 또는 잠금 이벤트 모니터에서 정보를 얻으십시오.
2. 관리 통지 로그의 정보를 사용하여 교착 상태 문제점 해결 방법을 결정하십시오. 잠금 경합 및 잠금 대기 시간을 줄이는 데 도움이 되는 많은 지침이 있습니다. 다음 옵션을 고려하십시오.
   - 각 응용프로그램 연결은 잠금 대기를 방지하기 위해 자체 행 세트를 처리해야 합니다.
   - 때로는 모든 응용프로그램이 동일한 순서로 공통 데이터에 액세스하는지(예를 들어, 테이블 A, 테이블 B, 테이블 C 등의 순으로 행 액세스(및 잠금)를 수행함을 의미) 확인하여 교착 상태 빈도를 줄일 수 있습니다. 두 응용프로그램이 다른 순서로 동일한 오브젝트에 대해 호환되지 않는 잠금을 사용할 경우, 교착 상태에 이를 위험이 훨씬 더 큽니다.
   - 둘 다 트랜잭션을 롤백하기 때문에 잠금 시간종료가 교착 상태보다 훨씬 낫지는 않지만, 교착 상태 수를 최소화해야 하는 경우 관련된 잠재적 교착 상태를 발견하기 전에 잠금 시간종료가 일반적으로 발생함을 확인하여 교착 상태 수를 최소화할 수 있습니다. 이를 수행하려면 **locktimeout** 데이터베이스 구성 매개변수 값(초 단위)을 **dlchktime** 데이터베이스 구성 매개변수 값(밀리초 단위)보다 훨씬 낮게 설정하십시오. 그렇지 않으면, **locktimeout**이 **dlchktime** 간격보다 긴 경우 교착 상태 검출기는 교착 상태 상황이 시작된 직후 작동하여 잠금 시간종료가 발생하기 전에 교착 상태를 발견할 수 있습니다.
   - 가능하면 동시 DDL 조작을 피하십시오. 예를 들어, 테이블 자체 외에 테이블 인덱스, 기본 키, 점검 제한조건 등에 대한 행을 삭제해야 할 수도 있으므로 DROP TABLE문은 많은 카탈로그 갱신을 초래할 수 있습니다. 기타 DDL 조작이 오브젝트를 삭제하거나 작성하는 경우, 잠금 충돌과 이따금씩 교착 상태가 발생할 수 있습니다.
   - 가능한 빨리 다음 조치를 커미트하는 것이 우수 사례입니다.
     - 쓰기 조치(예: 삭제, 삽입 및 갱신)
     - DDL(Data Definition Language)문(예: ALTER, CREATE 및 DROP)

– BIND 및 REBIND 명령

3. 교착 상태 검출기는 다음 상황을 알 수 없어 해결할 수 없으므로 응용프로그램 설계는 이를 예방해야 합니다. 응용프로그램(특히, 멀티스레드 응용프로그램)은 DB2 잠금 대기 및 세마포어와 같은 비DB2 자원에 대한 대기와 관련한 교착 상태에 빠질 수 있습니다. 예를 들어, 연결 A는 연결 B가 보유한 잠금을 기다리고 B는 A가 보유한 세마포어를 기다릴 수 있습니다.

다음 단계

응용프로그램을 재실행하여 관리 통지 로그에서 잠금 관련 항목을 검사하여 잠금 문제점이 제거되었는지 확인하십시오.

잠금 시간종료 문제점 진단

자원 잠금을 기다리고 있는 트랜잭션이 **locktimeout** 데이터베이스 구성 매개변수가 지정한 대기 시간 값을 초과할 만큼 오래 기다리는 경우 잠금 시간종료가 발생합니다. 이는 SQL 쿼리 성능 저하를 초래하는 시간을 소비합니다. 잠금 시간종료 수가 늘어나고 **locktimeout** 데이터베이스 구성 매개변수가 0이 아닌 시간 값으로 설정된 경우, 잠금 시간종료 문제점일 가능성이 있습니다.

시작하기 전에

일반적으로, 시스템이 처리 지연 및 성능 저하를 포함할 수 있는 비정상 동작을 보이는지 객관적으로 평가할 수 있으려면 시스템의 일반적인 동작(기준선)을 설명하는 정보가 있어야 합니다. 그런 다음, 예측된 비정상 동작 관찰 내용과 기준선 간에 비교를 할 수 있습니다. 정기적인 운영 모니터링 태스크를 스케줄링하여 기준선 데이터를 수집하는 것은 문제점 해결 프로세스의 주요 구성요소입니다. 시스템의 기준선 운영 설정에 대한 자세한 정보는 13 페이지의 『시스템 성능 운영 모니터링』을 참조하십시오.

잠금 시간종료 잠금 이벤트 모니터 방법에 대한 지시사항은 *데이터베이스 모니터링 안내서 및 참조서*의 『잠금 이벤트 모니터링』을 참조하십시오.

프로시저

진단 때때로, 잠금 대기 상황으로 인해 트랜잭션이 롤백되는 잠금 시간종료가 발생합니다. 잠금 대기가 잠금 시간종료가 될 때까지의 기간은 **locktimeout** 데이터베이스 구성 매개변수로 지정됩니다. 과도한 수의 잠금 시간종료는 교착 상태만큼 시스템에 방해가 될 수 있습니다. 대부분의 프로덕션 제품에서는 교착 상태가 비교적 드물지만, 잠금 시간종료는 더 흔할 수 있습니다. 응용프로그램에서는 일반적으로 이들을 비슷한 방법으로 처리해야 합니다(트랜잭션을 처음부터 재실행). 이러한 발생 비율을 모니터링하면 많은 잠금 시간종료가 DBA에서 인식하지 않은 상태에서 시스템에 상당한 추가적인 부하를 주는 것을 방지하는 데 도움이 됩니다.

표시 징후

다음 잠금 시간종료 표시 징후를 찾으십시오.

- 응용프로그램이 자주 트랜잭션을 재실행함
- **lock_timeouts** 모니터 요소 값이 증가함
- 관리 통지 로그의 잠금 시간종료 메시지 항목

모니터할 사항

비교적 일시적인 잠금 이벤트의 특징 때문에 잠금 이벤트 데이터는 전개도를 더 잘 이해할 수 있도록 일정 기간 동안 주기적으로 수집된 경우에 가장 유용합니다.

관리 통지 로그에서 잠금 시간종료 메시지를 모니터할 수 있습니다.

이벤트 모니터를 작성하여 워크로드 또는 데이터베이스에 대한 잠금 시간종료 데이터를 캡처하십시오.

주요 표시기 모니터링 요소는 다음과 같습니다.

- **lock_timeouts** 값이 증가함
- **int_rollbacks** 값이 증가함

여기 나열된 표시 징후 중 하나 이상을 관찰한 경우, 잠금 시간종료 문제점일 가능성이 있습니다. 『다음에 수행할 일』 섹션의 링크를 따라 이 문제점을 해결하십시오.

다음 단계

잠금 시간종료로 인해 문제점이 발생하고 있음을 진단한 후에는 문제점을 해결하기 위한 단계를 수행하십시오(『잠금 시간종료 문제점 해결』).

잠금 시간종료 문제점 해결

잠금 시간종료 문제점을 진단한 후, 다음 단계는 잠금 시간종료 기간이 경과될 때까지 응용프로그램이 잠금을 기다리기 때문에 발생하는 문제점을 해결하려고 시도하는 것입니다. 여기 제공된 지침은 발생하고 있는 잠금 시간종료 문제점을 해결하는 데 도움을 주고 향후 이러한 사건이 발생하지 않도록 예방할 수 있습니다.

시작하기 전에

520 페이지의 『잠금 문제점 진단 및 해결』에 설명된 잠금 문제점에 대한 필수 진단 단계를 수행하여 잠금 시간종료 문제점이 발생하고 있음을 확인하십시오.

이 태스크에 대한 정보

여기 제공된 지침은 발생하고 있는 잠금 시간종료 문제점을 해결하는 데 도움을 주고 향후 이러한 사건이 발생하지 않도록 예방할 수 있습니다.

프로시저

다음 단계를 사용하여 허용할 수 없는 잠금 시간종료 문제점의 원인을 진단하고 교정 방법을 적용하십시오.

1. 에이전트에 잠금 시간종료가 발생하고 있는 모든 테이블에 대한 관리 통지 로그 또는 잠금 이벤트 모니터에서 정보를 얻으십시오.
2. 관리 통지 로그의 정보를 사용하여 잠금 시간종료 문제점 해결 방법을 결정하십시오. 잠금 경합 및 잠금 대기 시간을 줄이는 데 도움이 되는 지침(잠금 시간종료 수를 줄일 수 있음)이 많이 있습니다. 다음 옵션을 고려하십시오.
   - **locktimeout** 데이터베이스 구성 매개변수를 사용자의 데이터베이스 환경에 맞는 초 수로 조정하십시오.
   - 가능한 경우, 매우 긴 트랜잭션 및 WITH HOLD 커서를 피하십시오. 잠금이 길게 보유될수록 다른 응용프로그램과의 경쟁이 발생할 가능성이 더 커집니다.
   - 가능한 빨리 다음 조치를 커미트하는 것이 우수 사례입니다.
     - 쓰기 조치(예: 삭제, 삽입 및 갱신)
     - DDL(Data Definition Language)문(예: ALTER, CREATE 및 DROP)
     - BIND 및 REBIND 명령
   - 특히 반복 읽기(RR) 분리 수준 하에서, 필요한 것보다 더 큰 결과 세트를 페치하지 않도록 하십시오. 행을 더 많이 건드릴 수록 잠금이 더 많이 보유되고 그 밖의 누군가가 보유하는 잠금에 부딪히게 될 가능성이 더 커집니다. 실질적으로, 이것은 종종 더 많은 행을 다시 가져와 응용프로그램에서 필터링하기보다는 SELECT문의 WHERE 절로 행 선택 기준을 푸시다운하는 것을 의미합니다. 예를 들어, 다음과 같습니다.

     ```
     exec sql declare curs for
       select c1,c2 from t
       where c1 not null;
     exec sql open curs;
     do {
       exec sql fetch curs
         into :c1, :c2;
     } while( P(c1) != someVar );

     ==>

     exec sql declare curs for
       select c1,c2 from t
       where c1 not null
       and myUdfP(c1) = :someVar;
     exec sql open curs;
     exec sql fetch curs
         into :c1, :c2;
     ```

   - 필요한 것보다 더 높은 분리 수준을 사용하지 않도록 하십시오. 응용프로그램에서 결과 세트 무결성을 보존하기 위해 반복 읽기가 필요할 수도 있지만, 이는 잠금 보유 및 잠재적 잠금 충돌의 견지에서 추가 비용을 초래합니다.

- 응용프로그램의 비즈니스 논리에 적합한 경우, **DB2_EVALUNCOMMITTED**, **DB2_SKIPDELETED** 및 **DB2_SKIPINSERTED** 레지스트리 변수를 통한 잠금 동작 수정을 고려하십시오. 이러한 레지스트리 변수는 DB2 데이터베이스 관리 프로그램이 일부 상황에서 잠금 사용을 지연시키거나 피할 수 있게 하므로, 경쟁이 줄어들고 잠재적으로 처리량이 향상됩니다.

다음 단계

응용프로그램을 재실행하여 관리 통지 로그에서 잠금 관련 항목을 검사하거나 해당 워크로드, 연결, 서비스 서브클래스, 작업 단위(UOW) 및 활동 레벨에 대한 잠금 대기 및 잠금 대기 시간 메트릭을 검사하여 잠금 문제점이 제거되었는지 확인하십시오.

잠금 에스컬레이션 문제점 진단

잠금(잠금 스페이스)에 할당된 메모리를 줄이기 위해 많은 행 레벨 잠금이 단일 메모리 절약 테이블 잠금으로 에스컬레이션되는 경우 잠금 에스컬레이션이 발생합니다. 이런 상황은 자동화되고 잠금에 할당된 메모리 스페이스를 절약하긴 하지만 허용할 수 없는 레벨로 동시성을 감소시킬 수 있습니다. 일반적인 잠금 대기 수보다 많고 관리 통지 로그 항목이 잠금 에스컬레이션이 발생하고 있음을 표시하는 경우, 잠금 에스컬레이션 문제점일 가능성이 있습니다.

시작하기 전에

일반적으로, 시스템이 처리 지연 및 성능 저하를 포함할 수 있는 비정상 동작을 보이는지 객관적으로 평가할 수 있으려면 시스템의 일반적인 동작(기준선)을 설명하는 정보가 있어야 합니다. 그런 다음, 예측된 비정상 동작 관찰 내용과 기준선 간에 비교를 할 수 있습니다. 정기적인 운영 모니터링 태스크를 스케줄링하여 기준선 데이터를 수집하는 것은 문제점 해결 프로세스의 주요 구성요소입니다. 시스템의 기준선 운영 설정에 대한 자세한 정보는 13 페이지의 『시스템 성능 운영 모니터링』을 참조하십시오.

프로시저

진단 다중 행 레벨 잠금으로부터 단일 테이블 레벨 잠금으로의 잠금 에스컬레이션이 발생하는 이유는 다음과 같습니다.

- 테이블에 대해 보유한 많은 행 레벨 잠금이 소비하는 전체 메모리 양이 잠금 저장을 위해 할당된 전체 메모리 비율을 초과합니다.
- 잠금 목록에 스페이스가 부족합니다. 잠금 목록 소모의 원인이 된 응용프로그램은 응용프로그램이 대부분의 잠금 보유자가 아니더라도 잠금 에스컬레이션 프로세스를 통해 잠금을 강제 실행합니다.

잠금 저장을 위해 할당된 전체 메모리의 임계값 비율(잠금 에스컬레이션이 발생하려면 응용프로그램이 이를 초과해야 함)은 **maxlocks** 데이터베이스 구성 매개변수를 통해 정의되며 잠금에 할당된 메모리는 **locklist** 데이터베이스 구성 매개변수를 통해 정의됩니다. 잘 구성된 데이터베이스에서는 잠금 에스컬레이션이

거의 발행하지 않습니다. 잠금 에스컬레이션이 허용할 수 없는 레벨로 동시성을 감소시킬 경우, 문제점을 분석하고 최상의 조치 과정을 결정해야 합니다.

자체 조정 메모리 관리자(STMM)가 그렇지 않으면 **locklist** 데이터베이스 구성 매개변수를 통해서만 할당되는 잠금 메모리를 관리하는 경우에는 메모리 스페이스 관점에서 볼 때 잠금 에스컬레이션은 큰 문제가 되지 않습니다. STMM은 여유 메모리 스페이스가 소모되면 잠금 메모리 스페이스를 자동으로 조정합니다.

표시 징후

다음 잠금 에스컬레이션 표시 징후를 찾으십시오.

- 관리 통지 로그의 잠금 에스컬레이션 메시지 항목

모니터할 사항

비교적 일시적인 잠금 이벤트의 특징 때문에 잠금 이벤트 데이터는 전개도를 더 잘 이해할 수 있도록 일정 기간 동안 주기적으로 수집된 경우에 가장 유용합니다.

다음 모니터링 요소를 점검하여 잠금 에스컬레이션이 SQL 쿼리 성능 저하에 기여하는 요인이 될 수 있다는 표시를 확인하십시오.

- **lock_escals**

여기 나열된 표시 징후 중 하나 이상을 관찰한 경우, 잠금 에스컬레이션 문제점일 가능성이 있습니다. 『다음에 수행할 일』 섹션의 링크를 따라 이 문제점을 해결하십시오.

다음 단계

잠금 에스컬레이션으로 인해 문제점이 발생하고 있음을 진단한 후에는 문제점을 해결하기 위한 단계를 수행하십시오(『잠금 에스컬레이션 문제점 해결』).

잠금 에스컬레이션 문제점 해결

잠금 에스컬레이션 문제점을 진단한 후, 다음 단계는 데이터베이스 관리 프로그램이 행 레벨에서 테이블 레벨로 잠금을 자동으로 에스컬레이션하기 때문에 발생하는 문제점을 해결하려고 시도하는 것입니다. 여기 제공된 지침은 발생하고 있는 잠금 에스컬레이션 문제점을 해결하는 데 도움을 주고 향후 이러한 사건이 발생하지 않도록 예방할 수 있습니다.

시작하기 전에

520 페이지의 『잠금 문제점 진단 및 해결』에 설명된 잠금 문제점에 대한 필수 진단 단계를 수행하여 잠금 에스컬레이션 문제점이 발생하고 있음을 확인하십시오.

이 태스크에 대한 정보

여기 제공된 지침은 발생하고 있는 잠금 에스컬레이션 문제점을 해결하는 데 도움을 주고 향후 이러한 사건이 발생하지 않도록 예방할 수 있습니다.

프로시저

목표는 잠금 에스컬레이션을 최소화하거나, 가능하면 이를 제거하는 것입니다. 잠금 처리를 위한 우수한 응용프로그램 설계와 데이터베이스 구성을 조합하면 잠금 에스컬레이션을 최소화거나 제거할 수 있습니다. 잠금 에스컬레이션은 동시성을 감소시키고 잠재적인 잠금 시간종료를 초래할 수 있으므로 잠금 에스컬레이션 해결은 중요한 태스크입니다. 관리 통지 로그에 기록된 메시지와 **lock_escals** 모니터 요소를 사용하여 잠금 에스컬레이션을 식별하고 정정할 수 있습니다.

먼저, 잠금 에스컬레이션 정보가 기록되고 있는지 확인하십시오. **notifylevel** 데이터베이스 관리 프로그램 구성 매개변수의 값을 디폴트인 3 또는 4로 설정하십시오. **notifylevel** 2에서는 오류 SQLCODE만 기록됩니다. 잠금 에스컬레이션이 **notifylevel** 3 또는 4에서 실패할 경우, 잠금 에스컬레이션이 실패한 테이블에 대한 정보 또한 기록됩니다. **notifylevel** 4에서, 현재 동적 SQL문을 실행 중인 경우 쿼리도 로그됩니다.

다음 단계를 사용하여 허용할 수 없는 잠금 에스컬레이션 문제점의 원인을 진단하고 교정 방법을 적용하십시오.

1. 잠금이 에스컬레이션된 모든 테이블에 대한 관리 통지 로그 및 관련된 응용프로그램으로부터 정보를 수집하십시오. 이 로그 파일에는 다음 정보가 포함되어 있습니다.
   - 현재 보유된 잠금의 수
   - 잠금 에스컬레이션이 완료되기 전에 필요한 잠금의 수
   - 에스컬레이션되는 각 테이블의 테이블 이름 및 테이블 ID
   - 현재 보유된 비테이블 잠금의 수
   - 에스컬레이션의 일부로 획득될 새 테이블 레벨 잠금. 일반적으로, S 또는 X 잠금이 획득됩니다.
   - 새 테이블 레벨 잠금의 획득과 연관된 내부 리턴 코드
2. 잠금 에스컬레이션에 관련된 응용프로그램에 대한 관리 통지 로그 정보를 사용하여 에스컬레이션 문제점 해결 방법을 결정하십시오. 다음 옵션을 고려하십시오.
   - **maxlocks** 또는 **locklist** 데이터베이스 구성 매개변수, 또는 둘 다를 점검하고 가능한 한 조정하십시오. 파티션된 데이터베이스 시스템에서는 모든 데이터베이스 파티션에서 이 변경을 수행하십시오. **locklist** 구성 매개변수 값이 현재 워크로드에 비해 너무 작을 수 있습니다. 다중 응용프로그램에 잠금 에스컬레이션이 발생하는 경우, 잠금 목록 크기를 늘려야 한다는 표시일 수 있습니다. 워크로드 증가 또는 새 응용프로그램 추가로 인해 잠금 목록이 너무 작아질 수 있습니다. 하나의 응용프로그램에만 잠금 에스컬레이션이 발생하는 경우, **maxlocks** 구성 매개변수를 조정하여 이를 해결할 수 있습니다. 그러나 **maxlocks**를 늘리는 동시에

locklist를 늘리는 것을 고려할 수 있습니다. 하나의 응용프로그램이 잠금 목록을 추가로 사용하도록 허용된 경우, 다른 모든 응용프로그램은 이제 잠금 목록에서 사용 가능한 남아 있는 잠금을 소모시킬 수 있으며 에스컬레이션이 발생할 수 있습니다.

- 응용프로그램 및 SQL문이 실행되는 분리 수준(예: RR, RS, CS 또는 UR)을 고려할 수 있습니다. COMMIT가 실행될 때까지 잠금이 보유되기 때문에 RR 및 RS 분리 수준은 더 많은 에스컬레이션을 초래하는 경향이 있습니다. CS 및 UR 분리 수준은 COMMIT가 실행될 때까지 잠금을 보유하지 않으므로 잠금 에스컬레이션이 발생할 가능성이 적습니다. 응용프로그램에서 용인할 수 있는 가능한 가장 낮은 분리 수준을 사용하십시오.
- 응용프로그램의 커미트 빈도를 늘리십시오(응용프로그램 요구 및 응용프로그램 설계가 이를 허용하는 경우). 커미트 빈도를 늘리면 주어진 시간에 보유되는 잠금 수가 줄어듭니다. 따라서 응용프로그램이 **maxlocks** 값(잠금 에스컬레이션을 트리거함)에 도달하지 못하게 하고 모든 응용프로그램이 잠금 목록을 소모시키지 못하게 할 수 있습니다.
- LOCK TABLE문을 사용하여 테이블 잠금을 획득하도록 응용프로그램을 수정할 수 있습니다. 이는 많은 응용프로그램과 사용자의 동시 액세스가 중요하지 않은 테이블의 경우 우수한 전략입니다. 응용프로그램이 이 응용프로그램 인스턴스에 대해 고유하게 이름 지정된 영구 작업 테이블(예: DGTT가 아님)을 사용하는 경우를 예로 들 수 있습니다. 응용프로그램이 보유한 잠금 수가 줄어들고 작업 테이블에서 액세스하는 행에서 더 이상 행 잠금을 획득 및 해제하지 않아도 되기 때문에 성능이 향상되므로 테이블 잠금 획득은 이 경우 우수한 전략입니다.

  응용프로그램에 작업 테이블이 없고 **locklist** 또는 **maxlocks** 구성 매개변수 값을 늘릴 수 없는 경우, 응용프로그램이 테이블 잠금을 획득하게 할 수 있습니다. 그러나 잠글 테이블을 선택할 때 주의해야 합니다. 많은 응용프로그램과 사용자가 액세스하는 테이블을 잠그면 응답 시간에 영향을 줄 수 있는 동시성 문제가 발생하고, 최악의 경우 응용프로그램에 잠금 시간종료가 발생할 수 있으므로 이러한 테이블은 피하십시오.

다음 단계

응용프로그램을 재실행하여 관리 통지 로그에서 잠금 관련 항목을 검사하여 잠금 문제점이 제거되었는지 확인하십시오.

지속된 트랩 복구

DB2 인스턴스는 발생한 트랩에 대해 FODC(First Occurrence Data Capture) 패키지를 준비합니다. 디폴트로, DB2 인스턴스는 트랩 탄력성에 맞게 구성되었습니다. 또한 DB2 인스턴스는 트랩 지속 가능 여부를 판별했습니다. 『지속 가능』은 트랩된 DB2 엔진 스레드는 일시중단되었고 DB2 인스턴스는 계속 실행됨을 의미합니다.

디폴트로, DB2 인스턴스는 **DB2RESILIENCE** 레지스트리 변수의 디폴트 설정을 기반으로 트랩 탄력성에 맞게 구성되었습니다.

프로시저

지속된 트랩 재구성

트랩은 트랩(DB2 프로그래밍 오류)이 발생할 때 데이터베이스 시스템에 미치는 영향을 최소화하기 위해 지속됩니다. 지속된 트랩의 결과 다음 진단이 발생합니다.

1. **diagpath** 데이터베이스 관리 프로그램 구성 매개변수를 사용하여 지정된 완전한 경로에 FODC 디렉토리가 작성됩니다.
2. ADM14013C 오류 메시지가 관리 통지 및 db2diag 로그 파일에 로그됩니다.

   주: 트랩이 지속될 수 없으면 ADM14011C가 로그되며, 인스턴스가 종료됩니다.
3. sqlcode -1224 오류가 응용프로그램에 리턴됩니다.
4. EDU 스레드가 일시중단됩니다(db2pd -edus 출력에서 관찰할 수 있음).

복구

지속된 트랩이 인스턴스 일반 조작을 방해하지 않을 것으로 예측하지만, 일시중단된 EDU 스레드는 일부 자원을 보유하므로 다음 단계를 따라 되도록 일찍 인스턴스를 중지했다가 재시작할 것을 권장합니다.

1. db2start 명령이 실행될 때 응급 복구를 위한 복구 창을 최소화하는 시간종료 기간 내에 COMMIT 또는 ROLLBACK을 실행하는 모든 활성 응용프로그램을 종료하려면, 다음 명령을 실행하십시오.

   ```
   db2 quiesce instance instance_name user user_name defer with timeout minutes
   ```
2. [선택사항] 1단계의 시간종료 기간 동안 COMMIT 또는 ROLLBACK을 수행하지 않은 응용프로그램과 시간종료 기간이 완료된 후에 데이터베이스에 액세스한 새 응용프로그램을 종료하려면, 다음 명령을 실행하십시오.

   ```
   db2 quiesce instance instance_name user user_name immediate
   ```
3. 다음 명령을 실행하여 인스턴스를 강제로 종료하고 EDU를 일시중단하십시오.

   ```
   db2_kill
   ```

   주: 인스턴스가 트랩을 지속한 경우 db2stop 명령 실행은 완료되지 않습니다.
4. 다음 명령 중 하나를 사용하여 DB2 인스턴스를 재시작하십시오.

   ```
   db2start
   ```

   또는

   ```
   START DATABASE MANAGER
   ```

진단 **diagpath** 데이터베이스 관리 프로그램 구성 매개변수에 지정된 FODC 디렉토리를 찾으십시오. 또한 관리 통지 또는 db2diag 로그 파일을 보고 FODC 디렉토리 위치를 확인할 수 있습니다. IBM Software Support에 FODC 정보를 전달하십시오.

관리 태스크 스케줄러 문제점 해결

이 점검목록은 관리 태스크 스케줄러에서 태스크를 실행하는 동안 발생하는 문제점을 해결하는 데 도움을 줄 수 있습니다.

프로시저

1. 예측한 대로 태스크가 실행되지 않는 경우, 수행해야 할 첫 번째 작업은 ADMIN_TASK_STATUS 관리 뷰에서 실행 상태 레코드를 찾는 것입니다.
   - 레코드가 있으면, 여러 값을 검사하십시오. 특히, STATUS, INVOCATION, SQLCODE, SQLSTATE, SQLERRMC 및 RC 컬럼에 주의하십시오. 값은 문제점의 근본 원인을 식별하는 경우가 많습니다.
   - 뷰에 실행 상태 레코드가 없는 경우, 태스크가 실행되지 않았습니다. 이에 대한 많은 설명이 있는데, 다음과 같습니다.
     - 관리 태스크 스케줄러를 사용할 수 없습니다. 관리 태스크 스케줄러를 사용할 수 없으면 태스크가 실행되지 않습니다. 스케줄러를 사용하려면 **DB2_ATS_ENABLE** 레지스트리 변수를 설정하십시오.
     - 태스크가 제거되었습니다. 누군가 태스크를 제거했습니다. ADMIN_TASK_LIST 관리 뷰를 쿼리하여 태스크가 존재하는지 확인하십시오.
     - 스케줄러가 태스크를 알지 못합니다. 관리 태스크 스케줄러는 사용 중인 각 데이터베이스에 5분마다 연결하여 새 태스크 및 갱신된 태스크를 찾습니다. 이 기간이 경과할 때까지 스케줄러는 태스크를 알지 못합니다. 최소 5분간 기다리십시오.
     - 데이터베이스가 비활성화되어 있습니다. 관리 태스크 스케줄러는 데이터베이스가 활성화되어야 태스크를 검색하거나 실행할 수 있습니다. 데이터베이스를 활성화하십시오.
     - 트랜잭션이 커미트되지 않았습니다. 관리 태스크 스케줄러는 커미트되지 않은 태스크를 무시합니다. 태스크를 추가, 갱신 또는 제거한 후에는 커미트하십시오.
     - 스케줄이 유효하지 않습니다. 태스크의 스케줄로 인해 태스크가 실행되지 않을 수 있습니다. 예를 들어, 태스크가 최대 호출 수에 이미 도달했을 수 있습니다. ADMIN_TASK_LIST 뷰에서 태스크의 스케줄을 검토하고 필요하면 스케줄을 갱신하십시오.

2. ADMIN_TASK_STATUS 관리 뷰를 참조하여 문제점의 원인을 판별할 수 없으면, DB2 진단 로그를 참조하십시오. 중요한 모든 오류는 db2diag 로그 파일에 로그됩니다. 정보용 이벤트 메시지도 태스크 실행 중에 관리 태스크 스케줄러 디먼에 의해 로그됩니다. 이러한 오류 메시지는 "관리 태스크 스케줄러" 구성요소에 의해 식별됩니다.

다음 단계

위 단계를 따랐는데도 여전히 문제점의 원인을 판별할 수 없으면, IBM Software Support에 문제점 관리 레코드(PMR)를 여십시오. 이러한 지시사항을 준수했음을 IBM Software Support에 알리고 수집한 진단 데이터를 보내십시오.

압축 문제점 해결

데이터 압축 사전이 자동으로 작성되지 않음

대형 테이블 또는 테이블에 대한 대형 XML 스토리지 오브젝트가 있지만 데이터 압축 사전이 작성되지 않았습니다. 예상한 대로 데이터 압축 사전 작성이 발생하지 않은 이유를 이해하려고 합니다. 이 정보는 테이블 오브젝트에 대한 압축 사전과 XML 스토리지 오브젝트에 대한 압축 사전 둘 다에 적용됩니다.

다음과 같은 상황일 수 있습니다.

- COMPRESS 속성이 YES로 설정된 테이블을 갖고 있습니다.
- 테이블이 일정 기간 동안 존재했으며 데이터가 추가 및 제거되었습니다.
- 테이블 *크기*가 임계값 *크기*에 근접합니다. 데이터 압축 사전이 자동으로 작성될 것으로 예상합니다.
- 임계값 *크기*를 넘어 테이블 *크기*를 증가시킬 것으로 예상되는 테이블 데이터 채우기 조작(예: INSERT, LOAD INSERT 또는 REDISTRIBUTE)을 실행했습니다.
- 데이터 압축 사전 자동 작성이 발생하지 않습니다. 데이터 압축 사전이 작성되어 테이블에 배치되지 않습니다. 해당 시점 이후에 테이블에 추가된 데이터에서 압축이 발생할 것으로 예상하지만 데이터가 압축 해제된 채로 남아 있습니다.
- XML 데이터의 경우, 데이터는 DB2 버전 9.7 스토리지 형식으로 되어 있습니다.

  테이블에 DB2 버전 9.5 이하를 사용하여 작성된 XML 컬럼이 포함된 경우 테이블의 XML 스토리지 오브젝트에 있는 데이터 압축은 지원되지 않습니다. 데이터 행 압축을 위해 이러한 테이블을 사용하는 경우, 테이블 오브젝트의 테이블 행 데이터만 압축됩니다. 삽입, 로드 또는 재구성 조작 중에 XML 스토리지 오브젝트를 압축할 수 없는 경우, XML 컬럼이 DB2 V9 또는 DB2 V9.5를 사용하여 작성된 경우에만, db2diag 로그 파일에 메시지가 기록됩니다.

데이터 압축이 발생하지 않는 이유는 무엇입니까?

압축 사전 자동 작성을 사용하기 위한 임계값 *크기*보다 데이터가 *크*더라도 점검되는 다른 조건이 있습니다. 조건은 사전을 작성할 수 있으려면 오브젝트에 충분한 데이터가 있어야 하며, ADM5591W 메시지가 이 요구사항을 알려줍니다. 또한 데이터에 대한 과거 활동이 데이터 삭제 또는 제거를 포함했을 수 있습니다. 데이터가 없는 오브젝트에 대형 섹션이 있을 수 있습니다. 이것이 오브젝트 *크기* 임계값을 충족시키거나 초과하는 대형 오브젝트(LOB)를 가질 수 있는 방법이긴 하지만, 사전 작성을 사용하기에는 오브젝트의 데이터가 부족할 수 있습니다.

오브젝트에 대해 활동이 많은 경우, 정기적으로 오브젝트를 재구성해야 합니다. XML 데이터의 경우, longlobdata 옵션을 사용하여 테이블을 재구성해야 합니다. 재구성하지 않는 경우, 오브젝트 *크기*가 커질 수 있지만 데이터로 산재하여 채워질 수 있습니다. 오브젝트 재구성은 분할된 데이터를 제거하고 오브젝트의 데이터를 압축합니다. 재구성 후 오브젝트는 *크기*는 작아지고 보다 조밀하게 채워집니다. 재구성된 오브젝트는 오브젝트의 데이터 양을 보다 정확하게 나타내며, 데이터 압축 사전 자동 작성을 사용하기 위한 임계값 *크기*보다 작을 수 있습니다.

오브젝트가 산재하여 채워진 경우, REORG TABLE 명령을 사용(XDA의 경우 LONGLOBDATA 옵션 사용)하여 테이블 재구성을 수행하여 사전을 작성할 수 있습니다. 디폴트로 KEEPDICTIONARY가 지정됩니다. 사전 작성을 강제하려면 RESETDICTIONARY를 지정할 수 있습니다.

테이블을 재구성해야 하는지 여부를 판별하려면 REORGCHK 명령을 사용하십시오.

데이터 행 압축에 테이블을 사용할 수 없는 경우 테이블에 대해 자동 사전 작성(ADC)이 발생하지 않습니다. ADM5594I 메시지는 데이터베이스에 ADC 처리를 사용할 수 없는 경우에 리턴되며 이유를 설명합니다.

테이블에 DB2 버전 9.5 이하를 사용하여 작성된 XML 컬럼이 포함된 경우, ADMIN_MOVE_TABLE 스토어드 프로시저를 사용하여 테이블을 업그레이드한 후 데이터 행 압축을 사용하십시오.

행 압축으로 임시 테이블의 디스크 스토리지 스페이스가 줄어들지 않음

스토리지 최적화 기능이 사용 허가되었음에도 임시 테이블의 디스크 스토리지 스페이스가 예상만큼 줄어들지 않는 알려진 상황이 발생할 수 있습니다.

증상

임시 테이블에서 행 압축을 사용해도 잠재적 디스크 스페이스 절약이 예측한 대로 실현되지 않습니다.

원인

- 동시에 실행 중이고 임시 테이블을 작성 중인 응용프로그램(각각 데이터베이스 관리 프로그램 메모리의 일부를 소비함) 수가 너무 많으면 대개 이런 상황이 발생합니다.

이로 인해 압축 사전을 작성하는 데 사용 가능한 메모리가 부족하게 됩니다. 이런 상황이 발생할 때 통지는 제공되지 않습니다.

- 행은 알고리즘에 따라 사전 기반 접근 방법을 사용하여 압축됩니다. 임시 테이블의 행이 디스크 스페이스를 상당히 절약할 수 있을 만큼 크면, 행이 압축됩니다. 임시 테이블의 작은 행은 압축되지 않으며, 디스크 스토리지 스페이스가 예상만큼 절약되지 않는 것은 이 때문입니다. 이런 상황이 발생할 때 통지는 제공되지 않습니다.

위험

행 *크기*가 임계값 미만인 임시 테이블에서 행 압축이 사용되지 않는 것을 제외하고 시스템에 위험은 없습니다. 사용 가능한 메모리가 여전히 극도로 제한되면 데이터베이스 관리 프로그램에 기타 역효과를 줄 수 있습니다.

데이터 복제 프로세스가 압축 행 이미지를 압축 해제할 수 없음

데이터 복제 솔루션이 압축 행 이미지가 있는 로그 레코드를 압축 해제할 수 없는 알려진 상황이 있습니다. 일시적(임시) 오류의 경우, 리턴된 SQL 코드는 오류 원인에 해당하는 반면 영구 오류는 일반적으로 SQL0204N 통지로 나타냅니다. 일시적 오류 상황에서만 나중에 로그 레코드의 행 이미지 압축 해제에 성공할 수 있습니다. db2ReadLog API는 로그 레코드를 압축 해제할 수 없더라도 다른 로그 레코드를 계속 처리합니다.

증상

압축 사용자 데이터가 포함된 로그 레코드를 읽는 동안 로그 판독기에 일시적 및 영구 오류가 발생할 수 있습니다. 압축 데이터(행 이미지)가 있는 로그 레코드를 읽는 동안 발생할 수 있는 두 가지 오류 클래스의 부분적인 목록 예는 다음과 같습니다.

일시적 오류:

- 테이블 스페이스 액세스가 허용되지 않음
- 테이블에 액세스할 수 없음(잠금 시간종료)
- (필수 사전 로드 및 저장을 위한) 메모리 부족

영구 오류:

- 테이블이 상주하는 테이블 스페이스가 존재하지 않음
- 로그 레코드가 속한 테이블 또는 테이블 파티션이 존재하지 않음
- 테이블 또는 테이블 파티션에 대해 사전이 존재하지 않음
- 로그 레코드에 테이블의 사전보다 이전 사전으로 압축된 행 이미지가 포함되어 있음

원인

복제 솔루션 또는 기타 로그 판독기가 데이터베이스 활동에 뒤져 있어 압축 사용자 데이터 로그 레코드를 읽는 중 오류를 수신할 수 있습니다(시나리오 1 참조). 읽고 있는 로그 레코드에 로그 읽기 시점에 테이블에서 사용 가능한 사전 보다 이전 압축 사전으로 압축된 압축 사용자 데이터가 포함된 경우 이러한 경우가 발생할 수 있습니다.

마찬가지로, 테이블이 삭제되면 테이블과 연관된 사전도 제거됩니다. 이 경우에는 테이블에 대한 압축 행 이미지를 압축 해제할 수 없습니다(시나리오 2 참조). 테이블이 삭제된 경우에도 압축되지 않은 행 이미지를 여전히 읽고 복제할 수 있으므로 이 제한사항은 압축 상태에 있지 않은 행 이미지에는 적용되지 않습니다.

임의의 한 테이블의 경우, 사용 중인 데이터 압축 사전과 실행기록 사전이 하나만 있을 수 있습니다.

시나리오 1:

t6 테이블에서는 압축을 사용할 수 있습니다. 복제 목적으로 테이블에 DATA CAPTURE CHANGES 속성을 사용할 수 있습니다. 데이터 복제 응용프로그램이 테이블을 복제하고 있고 로그 판독기가 압축 데이터(행 이미지)가 포함된 로그 레코드를 읽고 있습니다. REORG TABLE 명령(테이블의 사전이 재빌드되게 함)이 실행된 후, 클라이언트 로그 판독기는 LOAD 조작이 t6 테이블에서 수행되기 때문에 db2ReadLog API를 사용하여 첫 번째 INSERT문에 대한 첫 번째 로그 레코드를 읽고 있습니다.

압축 사전을 이미 포함하고 있고 DATA CAPTURE CHANGES 속성을 사용할 수 있는 t6 테이블에 대해 다음 명령문이 실행됩니다.

```
-> db2 alter table t6 data capture changes
-> db2 insert into t6 values (...)
-> db2 insert into t6 values (...)
```

데이터 압축 사전이 t6 테이블에 대해 이미 존재하기 때문에 ALTER 후 두 개의 INSERT가 압축됩니다(t6 테이블의 압축 사전을 사용). 이때, 로그 판독기는 아직 첫 번째 INSERT문에 도달하지 못했습니다.

다음 REORG TABLE 명령은 t6 테이블에 대해 새 압축 사전을 빌드하며, 현재 압축 사전은 실행기록 사전으로 보존되므로 로그 판독기는 현재 압축 사전 뒤에 하나의 사전을 작성합니다(그러나 실행기록 사전은 REORG 후 메모리에 로드되지 않음).

```
-> db2 reorg table t6 resetdictionary
```

로그 판독기가 INSERT문에 대해 INSERT 로그를 읽고 있기 때문에(이제 실행기록 사전을 메모리로 로드해야 함) t6 테이블은 LOAD 조작을 겪게 됩니다.

```
-> db2 load from data.del of del insert into table t6 allow no access
```

LOAD가 소스 테이블에서 실행되면, 지정된 ALLOW NO ACCESS 옵션으로 인해 t6 테이블은 Z 잠금됩니다. 로그 판독기는 INSERT 로그 레코드에서 찾은 행 이미지를 압축 해제하기 위해 실행기록 사전을 메모리로 로드해야 하지만, 사전을 페치하려면 IN 테이블 잠금이 필요합니다. 이 경우, 로그 판독기는 잠금을 획득하지 못합니다. 따라서 db2ReadLogFilterData 구조의 sqlcode 구성원이 SQL 코드 SQL2048N을 리턴합니다. 이는 일시적 오류에 해당합니다(즉, API가 다시 호출되면 로그 레코드를 압축 해제할 수 있음). 로그 판독기는 로그 레코드의 압축 행 이미지를 리턴하고 다음 로그 레코드를 계속 읽습니다.

시나리오 2:

t7 테이블에서는 DATA CAPTURE CHANGES 속성을 사용할 수 있습니다. 스토리지 비용을 줄이기 위해 테이블 압축을 사용할 수 있습니다. 데이터 복제 응용프로그램이 테이블을 복제하고 있지만, 로그 판독기가 소스 테이블 활동에 뒤져 있고 로그 판독기가 로그 레코드를 다시 읽기 전에 데이터 압축 사전이 이미 두 번 다시 빌드되었습니다.

DATA CAPTURE CHANGES 속성을 이미 사용할 수 있고, 테이블 압축을 사용할 수 있으며 새 사전이 빌드된 t7 테이블에 대해 다음 명령문이 실행됩니다.

```
-> db2 alter table t7 compress yes
-> db2 reorg table t7 resetdictionary
-> db2 insert into t7 values (...)
```

클라이언트 로그 판독기는 db2ReadLog API를 사용하여 아래 첫 번째 INSERT문에 대응하는 다음 로그를 읽으려고 합니다.

```
-> db2 insert into t7 values (...)
...
-> db2 reorg table t7 resetdictionary
-> db2 insert into t7 values (...)
...
-> db2 reorg table t7 resetdictionary
```

로그 판독기가 두 개 이상의 REORG RESETDICTIONARY 조작에 뒤졌기 때문에 db2ReadLog API는 이 경우 로그 레코드의 컨텐츠를 압축 해제할 수 없습니다. 로그 레코드의 행 이미지를 압축 해제하는 데 필요한 사전을 테이블에서 찾을 수 없습니다. 두 번째 REORG의 압축 사전과 마지막 REORG의 압축 사전만 테이블과 함께 저장됩니다. 그러나 db2ReadLog API는 오류로 실패하지 않습니다. 대신, 압축되지 않은 행 이미지가 사용자 버퍼에 리턴되며, 로그 레코드 앞에 db2ReadLogFilterData 구조가 붙어 sqlcode 구성원이 SQL 코드 SQL0204N을 리턴합니다. 이 코드는 영구 오류에 해당합니다(즉, 로그 레코드를 압축 해제할 수 없음).

환경

데이터 복제 솔루션이 db2ReadLog API를 사용하고 테이블에 대해 DATA CAPTURE CHANGES 속성이 설정된 플랫폼에서 이전 압축 사전 누락으로 인해 압축 로그 레코드를 압축 해제하지 못할 수 있습니다.

문제점 해결

사용자 조치:

일시적 오류의 경우, 읽기 요청을 재실행하고 정상적으로 로그를 읽을 수 있습니다. 예를 들어, 로그 레코드가 테이블 스페이스에 상주하는 테이블에 속하고 테이블 액세스가 허용되지 않는 경우에는 사전에 액세스하여 로그 레코드를 압축 해제할 수 없습니다(시나리오 1 참조). 나중에 테이블 스페이스를 사용할 수 있으며, 그 때 로그 읽기 요청을 재실행하면 로그 레코드를 정상적으로 압축 해제할 수 있습니다.

- 일시적 오류가 리턴되면(시나리오 1 참조), 해당 조치를 취할 수 있도록 오류 정보를 읽으십시오. 여기에는 테이블 조작이 완료되기를 기다리는 것이 포함될 수 있습니다(로그 레코드 다시 읽기 및 압축 해제에 성공할 수 있음).
- 영구 오류가 발생하면(시나리오 2 참조), 행 이미지를 압축하는 데 사용된 압축 사전을 더 이상 사용할 수 없으므로 로그 레코드의 행 이미지를 압축 해제할 수 없습니다. 이 경우, 복제 솔루션이 영향받는(목표) 테이블을 다시 초기화해야 합니다.

전역 변수 문제점 해결

일반적으로, 문제점을 겪고 있는 사용자가 전역 변수 READ 권한을 갖고 있으면 전역 변수와 관련한 응용프로그램 문제점 해결은 문제가 되지 않습니다. READ 권한이 있으면 VALUES(전역 변수 이름)문을 실행하여 전역 변수 값을 알 수 있습니다. 응용프로그램을 실행 중인 사용자에게 전역 변수 READ를 위한 액세스 권한이 없는 경우가 있습니다.

첫 번째 시나리오는 단순 솔루션을 갖는 전역 변수를 참조할 때 가능한 문제점을 설명합니다. 두 번째 시나리오는 해당 사용자에게 전역 변수 READ 권한이 부여되어야 하는 상황을 제공합니다.

시나리오 1

전역 변수 참조가 올바로 규정되어야 합니다. 동일한 이름을 갖는 변수와 PATH 레지스터 값에서 조기에 올바르지 않은 스키마가 발생하는 다른 스키마가 존재할 수 있습니다. 하나의 솔루션은 전역 변수 참조가 완전한지 확인하는 것입니다.

시나리오 2

응용프로그램 개발자(developerUser)는 자신만 읽기 액세스 권한을 갖는 일부 전역 변수의 값을 기반으로 아주 복잡한 일련의 프로시저, 뷰, 트리거 등을 작성합니다. 응용프로그램 최종 사용자(finalUser)는 로그인하여 developerUser가 작성한 환경을 사용하여 SQL 실행을 시작합니다. finalUser는 developerUser에게 자신이 볼 수 있도록 허용되어야 하는 데이터를 볼 수 없음을 컴플레인합니다. 이 문제점 해결의 일부로 developerUser는 자신의 권한 부여 ID를 finalUser의 권한 부여 ID로 변경하고 finalUser로 로그인한 후 finalUser와 동일한 SQL을 시도합니다. developerUser는 finalUser가 올바르고 문제점이 있음을 발견합니다.

developerUser는 finalUser가 자신이 보는 것과 동일한 전역 변수값을 보는지 여부를 판별해야 합니다. developerUser는 SET SESSION USER를 실행하여 finalUser가 보는 전역 변수값을 확인합니다. 이 문제점을 판별하고 해결하기 위한 제안 방법은 다음과 같습니다.

developerUser는 보안 관리자(secadmUser)에게 finalUser로서 SET SESSION USER를 사용할 수 있는 권한을 부여할 것을 요구합니다. 그런 다음, developerUser는 직접 로그인하고 SET SESSION AUTHORIZATION문을 사용하여 SESSION_USER 특수 레지스터를 finalUser의 것으로 설정합니다. 문제가 있는 SQL을 실행한 후에 다른 SET SESSION AUTHORIZATION문을 사용하여 developerUser로 다시 전환합니다. 이제 developerUser는 VALUES문을 실행하고 전역 변수의 실제 값을 볼 수 있습니다.

다음은 developerUser가 데이터베이스에서 취한 조치를 표시하는 샘플 SQL입니다.

```
#########################################################################
# developerUser connects to database and creates needed objects
#########################################################################

db2 "connect to sample user developerUser using xxxxxxxx"

db2 "create table security.users ₩
(userid varchar(10) not null primary key, ₩
firstname varchar(10), ₩
lastname varchar(10), ₩
authlevel int)"

db2 "insert into security.users values ('ZUBIRI', 'Adriana', 'Zubiri', 1)"
db2 "insert into security.users values ('SMITH', 'Mary', 'Smith', 2)"
db2 "insert into security.users values ('NEWTON', 'John', 'Newton', 3)"

db2 "create variable security.gv_user varchar(10) default (SESSION_USER)"
db2 "create variable security.authorization int default 0"

# Create a procedure that depends on a global variable
db2 "CREATE PROCEDURE SECURITY.GET_AUTHORIZATION() ₩
SPECIFIC GET_AUTHORIZATION ₩
RESULT SETS 1 ₩
LANGUAGE SQL ₩
   SELECT authlevel INTO security.authorization ₩
   FROM security.users ₩
   WHERE userid = security.gv_user"
```

```
db2 "grant all on variable security.authorization to public"
db2 "grant execute on procedure security.get_authorization to public"
db2 "terminate"

#########################################################################
# secadmUser grants setsessionuser
#########################################################################
db2 "connect to sample user secadmUser using xxxxxxxx"
db2 "grant setsessionuser on user finalUser to user developerUser"
db2 "terminate"

#########################################################################
# developerUser will debug the problem now
#########################################################################

echo "-------------------------------------------------------------"
echo " Connect as developerUser "
echo "-------------------------------------------------------------"
db2 "connect to sample user developerUser using xxxxxxxx"

echo "-------------------------------------------------------------"
echo " SET SESSION AUTHORIZATION = finalUser "
echo "-------------------------------------------------------------"
db2 "set session authorization = finalUser"

echo "--- TRY to get the value of gv_user as finalUser (we must not be able to)"
db2 "values(security.gv_user)"

echo "--- Now call the procedure---"
db2 "call security.get_authorization()"

echo "--- if it works it must return 3 ---"
db2 "values(security.authorization)"

echo "-------------------------------------------------------------"
echo " SET SESSION AUTHORIZATION = developerUser "
echo "-------------------------------------------------------------"

db2 "set session authorization = developerUser"

echo "--- See what the variable looks like ----"
db2 "values(security.gv_user)"

db2 "terminate"
```

고가용성 문제점 해결

AIX 6.1에서 DB2 버전 9.5 GA에 의해 Tivoli SA MP(System Automation for Multiplatforms) Base Component가 설치되지 않음

DB2 버전 9.5 GA 고가용성 기능에 포함된 IBM Tivoli® SA MP Base Component는 AIX 6.1 운영 체제를 지원하지 않습니다. AIX 6.1용 SA MP Base Component의 적절한 버전을 확보하려면 DB2 버전 9.5 FixPack 1 이상의 FixPack을 설치하십시오.

증상

AIX 6.1에 DB2 버전 9.5 GA 데이터베이스 제품을 설치하는 경우, 설치 프로그램은 사용자가 AIX 6.1을 사용 중임을 감지하고 SA MP Base Component를 설치하지 않습니다.

원인

DB2 버전 9.5 GA와 함께 번들로 제공되는 SA MP Base Component는 AIX 6.1을 지원하지 않습니다.

문제점 해결

AIX 6.1에 DB2 버전 9.5 FixPack 1 이상의 FixPack을 설치하면 SA MP Base Component가 정상적으로 설치됩니다.

불일치 문제점 해결

데이터 불일치 문제점 해결

데이터베이스에서 데이터 불일치가 존재하는 위치를 진단하는 것이 매우 중요합니다. 데이터 불일치를 판별하는 한 가지 방법은 INSPECT 명령의 출력을 사용하여 문제점이 존재하는 위치를 식별하는 것입니다. 불일치를 찾으면, 문제점 처리 방법을 결정해야 합니다.

데이터 일관성 문제점이 있음을 판별했다면, 선택은 다음 두 가지입니다.

- IBM Software Support에 문의하여 데이터 불일치 복구 지원을 받으십시오.
- 데이터 일관성 문제점이 있는 데이터베이스 오브젝트를 삭제(drop)한 후 재빌드하십시오.

INSPECT 명령의 변형 형태인 INSPECT CHECK를 사용하여 데이터 불일치가 명백한 데이터베이스, 테이블 스페이스 또는 테이블을 점검하십시오. INSPECT CHECK 명령 결과가 생성되면, db2inspf 명령을 사용하여 검사 결과를 형식화해야 합니다.

INSPECT 명령이 완료되지 않으면, IBM Software Support에 문의하십시오.

인덱스 데이터 불일치 문제점 해결

인덱스는 테이블의 올바른 데이터에 신속히 액세스할 수 있도록 정확해야 하며 그렇지 않으면 데이터베이스가 손상됩니다.

INSPECT 명령을 사용하면 교차 오브젝트 검사 절에서 INDEXDATA 옵션을 사용하여 인덱스 데이터 불일치에 대한 온라인 점검을 수행할 수 있습니다. INSPECT 명령을 사용하는 경우에는 인덱스 데이터 검사가 디폴트로 수행되지 않습니다. 명시적으로 요청해야 합니다.

INSPECT가 INDEXDATA 검사를 수행하는 동안 인덱스 데이터 불일치로 인해 오류가 발견되는 경우, SQL1141N 오류 메시지가 리턴됩니다. 이 오류 메시지가 리턴되는 동시에 진단 정보가 수집되고 db2diag 로그 파일에 덤프됩니다. 관리 통지 로그에 긴

급 메시지도 로그됩니다. db2diag 로그 파일 분석 도구(db2diag)를 사용하여 db2diag 로그 파일 컨텐츠를 필터링하고 형식화하십시오.

잠금 포함

INSPECT 명령을 INDEXDATA 옵션과 함께 사용하여 인덱스 데이터 불일치를 검사하는 동안에는 검사된 테이블만 IS 모드에서 잠깁니다.

INDEXDATA 옵션이 지정되면, 디폴트로 명시적으로 지정된 레벨 절 옵션의 값만 사용됩니다. 명시적으로 지정되지 않은 레벨 절 옵션의 경우, 디폴트 레벨(INDEX NORMAL 및 DATA NORMAL)은 NORMAL에서 NONE으로 겹쳐써집니다.

DB2 데이터베이스 시스템 설치 문제점 해결

DB2 데이터베이스 제품을 설치하는 동안 문제점이 발생하면, 시스템이 설치 요구사항을 충족하는지 확인하고 일반적인 설치 문제점 목록을 검토하십시오.

프로시저

DB2 데이터베이스 시스템 설치 문제점을 해결하려면 다음을 수행하십시오.

- 시스템이 설치 요구사항을 모두 충족하는지 확인하십시오.
- 라이센스 오류가 발생하면, 해당 라이센스를 적용했는지 확인하십시오.

  『지식 콜렉션: DB2 라이센스 문제점』 기술 노트(http://www.ibm.com/support/docview.wss?rs=71&uid=swg21322757)에 있는 자주 묻는 질문 목록을 검토하십시오.
- 해당 문서 및 DB2 기술 지원부 웹 사이트(www.ibm.com/software/data/db2/support/db2_9/troubleshoot.html)에 있는 설치 문제점 목록을 검토하십시오.

다음 단계

이러한 단계를 완료했지만 아직 문제점의 원인을 식별할 수 없는 경우, 진단 데이터 수집을 시작하여 자세한 정보를 얻으십시오.

설치 문제점에 대한 데이터 수집

설치 문제점이 발생하고 문제점의 원인을 판별할 수 없는 경우, 직접 또는 IBM Software Support에서 문제점을 진단 및 해결하는 데 사용할 수 있는 진단 데이터를 수집하십시오.

설치 문제점에 대한 데이터를 수집하려면 다음을 수행하십시오.

1. 선택사항: 추적을 사용 가능하게 한 상태에서 설치 시도를 반복하십시오. 예를 들어, 다음과 같습니다.

Linux 및 UNIX 운영 체제의 경우

```
db2setup -t /filepath/trace.out
```

Windows 운영 체제의 경우

```
setup -t ₩filepath₩trace.out
```

2. 설치 로그 파일을 찾으십시오.
   - Windows에서 디폴트 파일 이름은 "DB2-*ProductAbbreviation-DateTime*.log"입니다(예: DB2-ESE-Wed Jun 21 11_59_37 2006.log). 설치 로그의 디폴트 위치는 "My Documents"₩DB2LOG₩ 디렉토리입니다.
   - Linux 및 UNIX에서 디폴트 파일 이름은 db2setup.log, db2setup.his 및 db2setup.err입니다.

     추적(또는 디버그 모드)을 사용 가능하게 한 상태에서 문제점을 재현한 경우, dascrt.log, dasdrop.log, dasupdt.log, db2icrt.log.*PID*, db2idrop.log.*PID*, db2iupgrade.log.*PID* 및 db2iupdt.log.*PID*(여기서 *PID*는 프로세스 ID임)와 같은 추가 파일이 작성될 수 있습니다.

     이러한 모든 파일의 디폴트 위치는 /tmp 디렉토리입니다. 추적 파일(trace.out)의 경우, 지정하지 않으면 디폴트 디렉토리가 없으므로 추적 결과 파일이 작성된 폴더로 파일 경로를 지정해야 합니다.
3. 선택사항: IBM Software Support에 데이터를 제출할 경우, DB2에 대한 데이터도 수집하십시오. 추가 정보는 "DB2에 대한 데이터 수집"을 참조하십시오.

설치 문제점에 대한 데이터 분석

설치 문제점에 대한 진단 데이터를 수집한 후에 데이터를 분석하여 문제점의 원인을 판별할 수 있습니다. 이러한 단계는 선택적입니다. 문제점의 원인이 쉽게 판별되지 않으면, IBM Software Support에 데이터를 제출하십시오.

이러한 단계에서는 설치 문제점에 대한 데이터 수집에 설명된 파일을 확보했다고 가정합니다.

1. 해당 설치 로그 파일을 살펴보고 있는지 확인하십시오. 파일의 작성 날짜 또는 파일 이름에 포함된 시간소인을 점검하십시오(Windows 운영 체제의 경우).
2. 설치가 완료되었는지 여부를 판별하십시오.
   - Windows 운영 체제의 경우, 성공은 설치 로그 파일 맨 아래에 다음과 유사한 메시지로 표시됩니다.

     ```
     Property(C): INSTALL_RESULT = Setup Complete Successfully
     === Logging stopped: 6/21/2006  16:03:09 ===
     MSI (c) (34:38) [16:03:09:109]:
     Product: DB2 Enterprise Server Edition - DB2COPY1 -- Installation operation
     completed successfully.
     ```

- Linux 및 UNIX 운영 체제의 경우, 성공은 설치 로그 파일(디폴트로 db2setup.log로 이름 지정된 파일) 맨 아래에 메시지로 표시됩니다.

3. 선택사항: 오류가 발생했는지 여부를 판별하십시오. 설치가 완료되었지만 설치 프로세스 중에 오류 메시지를 수신한 경우, 설치 로그 파일에서 이러한 오류를 찾으십시오.
   - Windows 운영 체제의 경우, 대부분의 오류는 "ERROR:" 또는 "WARNING:"으로 시작합니다. 예를 들어, 다음과 같습니다.

     ```
     1: ERROR:An error occurred while running the command
     "D:WIBMWSQLLIBWbinWdb2.exe
     CREATE TOOLS CATALOG SYSTOOLS USE EXISTING DATABASE TOOLSDB FORCE" to
     initialize and/or migrate the DB2 tools catalog database.
     The return value is "4".

     1: WARNING:A minor error occurred while installing "DB2 Enterprise Server
     Edition - DB2COPY1" on this computer. Some features may not function
     correctly.
     ```

   - Linux 및 UNIX 운영 체제의 경우, Java가 오류(예: 예외 및 트랩 정보)를 리턴한 경우 디폴트 이름이 db2setup.err인 파일이 있습니다.

   설치 추적을 사용할 수 있게 한 경우, 설치 로그 파일에 추가 항목이 있으며 항목이 보다 자세합니다.

이 데이터를 분석해도 문제점을 해결할 수 없고 IBM Software Support와 유지보수 계약을 체결한 경우, 문제점 보고서를 열 수 있습니다. IBM Software Support에서는 사용자가 수집한 데이터 제출을 요구하며, 사용자가 수행한 분석에 대해서도 문의할 수 있습니다.

조사를 통해 문제점이 해결되지 않은 경우, IBM Software Support에 데이터를 제출하십시오.

알려진 문제점 및 솔루션

시스템 WPAR의 디폴트 경로에 루트가 아닌 사용자로 DB2 데이터베이스 제품 설치 시 오류(AIX)

AIX 6.1에서는 시스템 워크로드 파티션(WPAR)의 디폴트 설치 경로(/opt/IBM/db2/V9.7)에 루트가 아닌 사용자로 DB2 데이터베이스 제품을 설치하는 경우 여러 오류가 발생할 수 있습니다. 이러한 문제점을 방지하려면, WPAR에만 액세스할 수 있는 파일 시스템에 DB2 데이터베이스 제품을 설치하십시오.

증상

시스템 WPAR의 /usr 또는 /opt 디렉토리에 DB2 데이터베이스 제품을 설치하는 경우, 디렉토리 구성 방법에 따라 여러 오류가 발생할 수 있습니다. /usr 및 /opt 디렉

토리를 전역 환경과 공유(이 경우, /usr 및 /opt 디렉토리는 읽기 가능하지만 WPAR에서 쓰기 액세스는 불가능함)하거나 /usr 및 /opt 디렉토리 로컬 사본을 갖도록 시스템 WPAR을 구성할 수 있습니다.

첫 번째 시나리오에서는 DB2 데이터베이스 제품이 전역 환경의 디폴트 경로에 설치되는 경우 시스템 WPAR에 해당 설치가 표시됩니다. 외형적으로는 DB2가 WPAR에 설치되지만 DB2 인스턴스를 작성하려고 시도하면 DBI1288E The execution of the program db2icrt failed. This program failed because you do not have write permission on the directory or file /opt/IBM/db2/V9.7/profiles.reg,/opt/IBM/db2/V9.7/default.env 오류가 발생합니다.

두 번째 시나리오에서는 DB2 데이터베이스 제품이 전역 환경의 디폴트 경로에 설치되는 경우 WPAR이 /usr 및 /opt 디렉토리의 로컬 사본을 작성할 때 DB2 데이터베이스 제품 설치도 복사됩니다. 따라서 시스템 관리자가 데이터베이스 시스템을 사용하려고 시도하면 예기치 않은 문제점이 발생할 수 있습니다. DB2 데이터베이스 제품이 다른 시스템용으로 되어 있기 때문에 부정확한 정보가 복사될 수 있습니다. 예를 들어, 전역 환경에서 원래 작성된 DB2 인스턴스가 WPAR에 있는 것으로 나타납니다. 이로 인해 시스템 관리자가 시스템에 실제 설치된 인스턴스와 관련하여 혼동을 일으킬 수 있습니다.

원인

시스템 WPAR의 /usr 또는 /opt 디렉토리에 DB2 데이터베이스 제품을 설치하기 때문에 이러한 문제점이 발생합니다.

문제점 해결

전역 환경의 디폴트 경로에 DB2 데이터베이스 제품을 설치하지 마십시오.

WPAR에만 액세스할 수 있는 파일 시스템을 마운트하고 해당 파일 시스템에 DB2 데이터베이스 제품을 설치하십시오.

DB2 데이터베이스 제품의 베타 버전과 비베타 버전이 공존할 수 없음

DB2 사본이 하나 이상의 다른 DB2 데이터베이스 제품을 포함할 수 있지만 베타 제품과 비베타 제품을 둘 다 포함할 수는 없습니다. DB2 데이터베이스 제품의 베타 버전과 비베타 버전을 동일한 위치에 설치하지 마십시오.

이 제한사항은 DB2 데이터베이스 제품의 클라이언트 구성요소와 서버 구성요소 둘 다에 적용됩니다.

문제점 해결

비베타 버전을 설치하기 전에 DB2 버전 9.7 베타 버전을 설치 제거하거나 다른 설치 경로를 선택하십시오.

DB2 데이터베이스 제품 설치 시 서비스 이름 오류 해결

사용할 DB2 데이터베이스 제품 또는 DB2 정보 센터에 디폴트가 아닌 서비스 이름 또는 포트 번호를 선택하는 경우, 이미 사용 중인 값을 지정하지 않는지 확인하십시오.

증상

DB2 데이터베이스 제품 또는 *DB2 정보 센터*를 설치하려고 시도하면, DB2 설치 마법사가 "지정된 서비스 이름이 이미 사용 중입니다"를 나타내는 오류를 보고합니다.

원인

다음을 설치할 때 DB2 설치 마법사는 포트 번호와 서비스 이름을 선택하도록 프롬프트를 표시합니다.

- *DB2 정보 센터*
- 클라이언트로부터 TCP/IP 통신을 승인할 DB2 데이터베이스 제품
- 데이터베이스 파티션 서버의 역할을 할 DB2 데이터베이스 제품

디폴트값을 승인하지 않고 서비스 이름과 포트 번호를 선택하면 이 오류가 발생할 수 있습니다. 시스템의 services 파일에 이미 존재하는 서비스 이름을 선택하고 포트 번호만 변경하는 경우, 이 오류가 발생합니다.

문제점 해결

다음 조치 중 하나를 취하십시오.

- 디폴트값을 사용하십시오.
- services 파일에 둘 다 이미 있는 서비스 이름과 포트 번호를 사용하십시오.
- 사용되지 않는 서비스 이름과 사용되지 않는 포트 번호를 services 파일에 추가하십시오. DB2 설치 마법사에 이러한 값을 지정하십시오.

라이센스 문제점 해결

DB2 라이센스 준수 보고서 분석

DB2 기능의 라이센스 준수를 검증하려면 DB2 라이센스 준수 보고서를 분석하십시오. 라이센스 위반이 있으면, 해당 라이센스 키를 확보하거나 문제가 있는 DB2 데이터베이스 제품 또는 기능을 제거하여 해결할 수 있습니다.

시작하기 전에

다음 단계에서는 라이센스 센터 또는 db2licm 명령을 사용하여 DB2 라이센스 준수 보고서를 생성했다고 가정합니다.

프로시저

1. DB2 라이센스 준수 보고서가 포함된 파일을 여십시오.
2. 준수 보고서에서 각 DB2 기능의 상태를 검사하십시오. 보고서는 각 기능에 대해 다음 값 중 하나를 표시합니다.

   준수 위반이 발견되지 않았음을 표시합니다. 기능이 사용되었으며 올바로 사용 허가되었습니다.

   사용되지 않음
   이 특정 기능을 필요로 하는 활동을 수행하지 않았음을 표시합니다.

   위반 기능이 사용 허가되지 않았는데 사용되었음을 표시합니다.
3. 위반이 있는 경우, 라이센스 센터 또는 db2licm -l 명령을 사용하여 라이센스 정보를 보십시오.

   DB2 기능이 "라이센스가 없는 사용자" 상태로 나열된 경우, 해당 기능에 대한 라이센스를 확보해야 합니다. 라이센스 키 및 라이센스 키 등록 지시사항은 DB2 기능을 구입할 때 제공되는 활성화 CD에 있습니다.

   일부 DB2 기능은 소프트 중지 규정을 갖습니다. 즉, 위반 상태임에도 기능이 계속 작동하며 라이센스 키를 확보하여 적용할 시간을 제공합니다. 위반 상태에서는 기능이 작동하지 않는 하드 중지 규정을 갖는 기능도 있습니다.

   주: DB2 Workgroup Server Edition 및 DB2 Express Edition의 경우, SAMPLE 데이터베이스에는 구체화된 쿼리 테이블(MQT) 및 라이센스 위반을 일으키는 다차원 클러스터 테이블(MDC)이 포함됩니다. DB2 Enterprise Server Edition로 업그레이드해야만 이 위반을 제거할 수 있습니다.
4. 라이센스를 구입하는 대신 문제가 있는 오브젝트를 삭제하기로 결정하는 경우, 다음 명령을 사용하여 DB2 데이터베이스 제품에서 라이센스 위반을 일으키는 오브젝트 또는 설정을 판별하십시오.
   - DB2 Advanced Access Control 기능의 경우:

     레이블 기반 액세스 제어(LBAC)를 사용하는 테이블을 점검하십시오. DB2 사본의 모든 인스턴스에 있는 모든 데이터베이스에 대해 다음 명령을 실행하십시오.

     ```
     SELECT TABSCHEMA, TABNAME
     FROM SYSCAT.TABLES
     WHERE SECPOLICYID>0
     ```

   - DB2 Performance Optimization 기능의 경우:
     - 구체화된 쿼리 테이블이 있는지 여부를 점검하십시오. DB2 사본의 모든 인스턴스에 있는 모든 데이터베이스에 대해 다음 명령을 실행하십시오.

       ```
       SELECT CREATOR, NAME
       FROM SYSIBM.SYSTABLES WHERE TYPE='S'
       ```

– 다차원 클러스터 테이블이 있는지 여부를 점검하십시오. DB2 사본의 모든 인스턴스에 있는 모든 데이터베이스에 대해 다음 명령을 실행하십시오.

```
SELECT A.TABSCHEMA, A.TABNAME, A.INDNAME, A.INDSCHEMA
FROM SYSCAT.INDEXES A, SYSCAT.TABLES B
WHERE (A.TABNAME=B.TABNAME AND A.TABSCHEMA=B.TABSCHEMA)
AND A.INDEXTYPE='BLOK'
```

– 인스턴스가 쿼리 병렬 처리(*쿼리간 병렬 처리*라고도 함)를 사용하는지 여부를 점검하십시오. DB2 사본의 각 인스턴스에서 다음 명령을 한 번 실행하십시오.

```
SELECT NAME, VALUE
FROM SYSIBMADM.DBMCFG
WHERE NAME IN ('intra_parallel')
```

• DB2 Storage Optimization 기능의 경우:

테이블에 행 레벨 압축을 사용할 수 있는지 여부를 점검하십시오. DB2 사본의 모든 인스턴스에 있는 모든 데이터베이스에 대해 다음 명령을 실행하십시오.

```
SELECT TABSCHEMA, TABNAME
FROM SYSCAT.TABLES
WHERE COMPRESSION IN ('R', 'B')
```

다음 단계

위반을 해결한 경우(기능에 대한 라이센스를 확보하거나 위반 소스를 제거하여), 라이센스 센터에서 또는 다음 명령을 실행하여 라이센스 준수 보고서를 재설정할 수 있습니다.

```
db2licm -x
```

최적화 지침 및 프로파일 문제점 해결

최적화 지침(최적화 프로파일이 전달)에 대한 진단 지원은 EXPLAIN 테이블이 제공합니다.

옵티마이저가 최적화 지침을 적용하지 않으면 이유 코드 13과 함께 SQL0437W 경고를 수신합니다. 최적화 지침이 적용되지 않은 이유를 자세히 설명하는 진단 정보가 EXPLAIN 테이블에 추가됩니다. 옵티마이저 진단 출력을 수신하기 위한 다음 두 개의 EXPLAIN 테이블이 있습니다.

• EXPLAIN_DIAGNOSTIC - 이 테이블의 각 항목은 특정 명령문의 최적화에 속한 진단 메시지를 나타냅니다. 각 진단 메시지는 숫자 코드를 사용하여 나타냅니다.

• EXPLAIN_DIAGNOSTIC_DATA - 이 테이블의 각 항목은 EXPLAIN_DIAGNOSTIC 테이블의 특정 진단 메시지와 관련된 진단 데이터입니다.

진단 Explain 테이블을 작성하는 데 사용되는 DDL은 아래 555 페이지의 그림 39에 표시되어 있습니다.

다음 단계는 최적화 지침 사용 시 발생하는 문제점을 해결하는 데 도움을 줄 수 있습니다.

1. *문제점 해결 및 데이터베이스 성능 조정*의 『최적화 지침이 사용되었는지 검증』.
2. 내장 *관리 루틴 및 뷰*의 『EXPLAIN_GET_MSGS 테이블 함수』를 사용하여 전체 오류 메시지 조사.

이러한 단계를 완료했지만 문제점의 원인을 식별할 수 없는 경우, 진단 데이터 수집을 시작하고 IBM Software Support에 문의하십시오.

```
CREATE TABLE EXPLAIN_DIAGNOSTIC
      ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
        EXPLAIN_TIME      TIMESTAMP    NOT NULL,
        SOURCE_NAME       VARCHAR(128) NOT NULL,
        SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
        SOURCE_VERSION    VARCHAR(64)  NOT NULL,
        EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
        STMTNO            INTEGER      NOT NULL,
        SECTNO            INTEGER      NOT NULL,
        DIAGNOSTIC_ID     INTEGER      NOT NULL,
        CODE              INTEGER      NOT NULL,
        PRIMARY KEY (EXPLAIN_REQUESTER,
                     EXPLAIN_TIME,
                     SOURCE_NAME,
                     SOURCE_SCHEMA,
                     SOURCE_VERSION,
                     EXPLAIN_LEVEL,
                     STMTNO,
                     SECTNO,
                     DIAGNOSTIC_ID),
        FOREIGN KEY (EXPLAIN_REQUESTER,
                     EXPLAIN_TIME,
                     SOURCE_NAME,
                     SOURCE_SCHEMA,
                     SOURCE_VERSION,
                     EXPLAIN_LEVEL,
                     STMTNO,
                     SECTNO)
        REFERENCES EXPLAIN_STATEMENT ON DELETE CASCADE);

   CREATE TABLE EXPLAIN_DIAGNOSTIC_DATA
      ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
        EXPLAIN_TIME      TIMESTAMP    NOT NULL,
        SOURCE_NAME       VARCHAR(128) NOT NULL,
        SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
        SOURCE_VERSION    VARCHAR(64)  NOT NULL,
        EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
        STMTNO            INTEGER      NOT NULL,
        SECTNO            INTEGER      NOT NULL,
        DIAGNOSTIC_ID     INTEGER      NOT NULL,
        ORDINAL           INTEGER      NOT NULL,
        TOKEN             VARCHAR(1000),
        TOKEN_LONG        BLOB(3M) NOT LOGGED,
        FOREIGN KEY (EXPLAIN_REQUESTER,
                     EXPLAIN_TIME,
                     SOURCE_NAME,
                     SOURCE_SCHEMA,
                     SOURCE_VERSION,
                     EXPLAIN_LEVEL,
                     STMTNO,
                     SECTNO,
                     DIAGNOSTIC_ID)
        REFERENCES EXPLAIN_DIAGNOSTIC ON DELETE CASCADE);
```

주: EXPLAIN_REQUESTOR, EXPLAIN_TIME, SOURCE_NAME, SOURCE_SCHEMA, SOURCE_VERSION, EXPLAIN_LEVEL, STMTNO 및 SECTNO 컬럼은 EXPLAIN_STATEMENT 테이블에 대한 외부 키 및 EXPLAIN_DIAGNOSTIC와 EXPLAIN_DIAGNOSTIC_DATA 간의 상위-하위 (parent-child) 관계를 구성하기 위해 두 테이블 모두의 일부입니다.

그림 39. 진단 Explain 테이블 작성에 사용되는 DDL

이 DDL은 sqllib 디렉토리의 misc 서브디렉토리에 있는 EXPLAIN.DDL 파일에 포함되어 있습니다.

파티션된 데이터베이스 환경 문제점 해결

127.0.0.2와 관련된 FCM 문제점(Linux 및 UNIX)

파티션된 데이터베이스 환경에서는 /etc/hosts 파일에 127.0.0.2에 대한 항목이 있는 경우 FCM(Fast Communication Manager)에 문제점이 발생할 수 있습니다.

증상

상황에 따라 다양한 오류 메시지가 발생할 수 있습니다. 예를 들어, 데이터베이스를 작성할 때 SQL1229N "시스템 오류로 인해 현재 트랜잭션이 롤백되었습니다." 오류가 발생할 수 있습니다. SQLSTATE=40504

원인

/etc/hosts 파일에 IP 주소 127.0.0.2에 대한 항목이 있기 때문에 문제점이 발생합니다(여기서 127.0.0.2는 시스템의 완전한 호스트 이름에 맵핑됨). 예를 들어, 다음과 같습니다.

```
127.0.0.2 ServerA.ibm.com ServerA
```

여기서 "ServerA.ibm.com"은 완전한 호스트 이름입니다.

환경

문제점은 DB2 Database Partitioning Feature가 있는 DB2 Enterprise Server Edition로 제한됩니다.

문제점 해결

/etc/hosts 파일에서 항목을 제거하거나 주석으로 변환하십시오. 예를 들어, 다음과 같습니다.

```
# 127.0.0.2 ServerA.ibm.com ServerA
```

암호화된 파일 시스템에서 데이터베이스 파티션 작성(AIX)

AIX 6.1은 JFS2 파일 시스템 또는 파일 세트 암호화 기능을 지원합니다. DB2 데이터베이스 제품의 파티션된 데이터베이스 환경에는 이 기능이 지원되지 않습니다. AIX에서 EFS(암호화된 파일 시스템)를 사용하여 파티션된 데이터베이스 환경을 작성하려고 시도하면 SQL10004C 오류가 발생합니다.

증상

다중 파티션 데이터베이스 환경의 암호화된 파일 시스템에서 데이터베이스를 작성하려고 시도하면, SQL10004C ″데이터베이스 디렉토리에 액세스하는 중 입출력 오류가 발생했습니다.″ 오류를 수신합니다. SQLSTATE=58031

원인

지금은 AIX에서 암호화된 파일 시스템(EFS)를 사용하여 파티션된 데이터베이스 환경을 작성할 수 없습니다. 파티션된 데이터베이스 파티션은 rsh 또는 ssh를 사용하기 때문에 EFS의 키스토어가 유실되고 데이터베이스 파티션이 암호화된 파일 시스템에 저장된 데이터베이스 파일에 액세스할 수 없습니다.

문제점 진단

DB2 진단(db2diag) 로그 파일은 오류 메시지와 `OSERR : ENOATTR (112) "No attribute found"` 텍스트를 포함합니다.

문제점 해결

파티션된 데이터베이스 환경에서 데이터베이스를 작성하려면, 모든 시스템에 사용 가능한 파일 시스템이 있어야 하고 암호화된 파일 시스템이 아니어야 합니다.

스크립트 문제점 해결

데이터베이스 엔진에서 실행 중인 프로세스를 기반으로 하는 내부 도구 또는 스크립트가 있을 수 있습니다. 이제 모든 에이전트, 프리페처 및 페이지 클리너가 단일, 멀티스레드 프로세스에서 스레드로 간주되므로 이러한 도구 또는 스크립트가 더 이상 작동하지 않을 수 있습니다.

스레드 프로세스를 설명하려면 내부 도구 및 스크립트를 수정해야 합니다. 예를 들어, ps 명령을 시작하여 프로세스 이름을 나열한 후 특정 에이전트 프로세스에 대해 태스크를 수행하는 스크립트가 있을 수 있습니다. 스크립트를 재작성해야 합니다.

문제점 판별 데이터베이스 명령(db2pd)에는 에이전트 이름을 해당 스레드 ID와 함께 모두 나열하는 새 옵션인 -edu(『엔진 디스패치 가능 단위』의 줄임말)가 있습니다. db2pd -stack 명령은 스레드 엔진에 대한 작업을 계속하여 개별 EDU 스택을 덤프하거나 현재 노드에 대한 모든 EDU 스택을 덤프합니다.

스토리지 키 지원 문제점 해결

스토리지 보호 키(스레드 레벨의 하드웨어 키)는 DB2 엔진이 유효하지 않은 액세스 시도로부터 메모리를 보호하여 탄력성을 높이는 데 사용됩니다. 이 기능을 사용하는 동안 발생한 오류를 해결하려면 다음 섹션의 지시사항을 따르십시오.

레지스트리 변수 오류 진단

*데이터베이스 관리 개념 및 구성 참조서*의 『DB2_MEMORY_PROTECT』 레지스트리 변수 설정 시 유효하지 않은 값(DBI1301E) 오류가 리턴되었습니다. 이 오류가 발생하는 이유는 다음 중 하나입니다.

- 레지스트리 변수에 주어진 값이 유효하지 않습니다. 유효한 값에 대한 정보는 **DB2_MEMORY_PROTECT** 레지스트리 변수 사용법을 참조하십시오.
- 하드웨어 및 운영 체제가 스토리지 보호 키를 지원하지 않을 수 있으며 이 기능을 사용하지 못할 수 있습니다. 스토리지 보호 키는 POWER6™ 프로세서에서 사용 가능하며, AIX 5L™ 버전 5.3(기술 레벨 5300-06) 운영 체제부터 지원됩니다.

제 7 장 문제점 해결 DB2 Connect

DB2 Connect 환경은 여러 개의 소프트웨어, 하드웨어 및 통신 제품으로 구성되어 있습니다. 종결(오류의 위치)에 도달하기 위해 사용 가능한 데이터의 제거 및 순화 프로세스를 통해 문제점 해결에 가장 잘 접근할 수 있습니다.

관련 정보를 수집한 뒤 선택한 응용프로그램 주제를 토대로 참조 섹션으로 넘어가십시오.

진단 도구

문제가 발생하면 다음을 사용할 수 있습니다.

- 덤프 파일, 트랩 파일, 오류 로그, 통지 파일 및 경고 로그을 포함한 모든 진단 데이터는 다음과 같은 진단 데이터 디렉토리 경로(**diagpath**) 데이터베이스 관리 프로그램 구성 매개변수에 의해 지정된 경로에 있습니다.

  이 구성 매개변수의 값이 널(NULL)이면 진단 데이터가 다음 디렉토리 또는 폴더 중 하나에 작성됩니다.

  – Linux 및 UNIX 환경의 경우: `INSTHOME/sqllib/db2dump`, 여기에서 *INSTHOME*은 인스턴스의 홈 디렉토리입니다.

  – 지원되는 Windows 환경의 경우:

    - **DB2INSTPROF** 환경 변수가 설정되지 않으면 `x:₩SQLLIB₩DB2INSTANCE`가 사용됩니다. 여기에서 `x:₩SQLLIB`는 드라이브 참조 및 **DB2PATH** 레지스트리 변수에 지정된 디렉토리이며 **DB2INSTANCE**의 값에는 인스턴스의 이름이 있습니다.

      주: 디렉토리는 SQLLIB로 이름 지정될 필요는 없습니다.

    - **DB2INSTPROF** 환경 변수가 설정되면 `x:₩DB2INSTPROF₩DB2INSTANCE`가 사용됩니다. 여기에서 **DB2INSTPROF**는 인스턴스 프로파일 디렉토리의 이름이며 **DB2INSTANCE**는 인스턴스의 이름입니다. (디폴트로 Windows 32 비트 운영 체제의 **DB2INSTDEF** 값)

- Windows 운영 체제의 경우 이벤트 보기 프로그램을 사용해서 관리 통지 로그를 볼 수 있습니다.
- 사용 가능한 진단 도구에는 **db2trc**, **db2pd**, **db2support** 및 **db2diag**가 포함됩니다.
- Linux 및 UNIX 운영 체제의 경우 사용 중인 프로세스에 대한 프로세스 상태 정보를 표준 출력으로 리턴하는 **ps** 명령.

- UNIX 운영 체제의 경우 심각한 오류가 발생했을 때 현재 디렉토리에 작성된 코어 파일. 이 코어 파일에는 종료된 프로세스의 메모리 이미지가 들어 있으며 이 파일을 사용하여 오류를 일으킨 기능을 판별할 수 있습니다.

관련 정보 수집

문제점 해결은 문제점 범위를 좁히고 가능한 원인을 조사하는 것으로 구성됩니다. 문제점 해결은 관련 정보를 수집하고 알고 있는 사항, 수집된 데이터 및 제거할 수 있는 경로를 판별하는 것으로 시작됩니다. 다음 질문에 대해 간략하게 대답하십시오.

- 초기 연결에 성공했습니까?
- 하드웨어가 잘 작동하고 있습니까?
- 통신 경로가 작동 가능합니까?
- 통신 네트워크가 변경되어 이전 디렉토리 입력이 유효하지 않습니까?
- 데이터베이스가 시작되었습니까?
- 하나 이상의 클라이언트와 DB2 Connect 서버(게이트웨이) 사이, DB2 Connect 게이트웨이와 IBM 메인프레임 데이터베이스 서버 사이 또는 DB2 Connect Personal Edition과 IBM 메인프레임 데이터베이스 서버 사이의 통신이 고장났습니까?
- 메시지 내용과 메시지의 리턴된 토큰으로 무엇을 판별할 수 있습니까?
- 지금 db2trc, db2pd, 또는 db2support와 같은 진단 도구의 사용이 도움을 제공합니까?
- 기타 머신이 올바르게 작동하는 유사한 태스크를 수행하고 있습니까?
- 이것이 원격 태스크인 경우 로컬로 수행 시 성공적입니까?

초기 연결 실패

다음 질문을 검토하고 설치 단계를 따랐는지 확인하십시오.

1. *설치 처리가 성공적으로 완료되었습니까?*
   - 모든 전제조건 소프트웨어 제품을 사용할 수 있었습니까?
   - 메모리 및 디스트 스페이스가 적절했습니까?
   - 원격 클라이언트 지원이 설치되었습니까?
   - 통신 소프트웨어 설치가 오류 조건 없이 완료되었습니까?
2. *UNIX 운영 체제의 경우 제품 인스턴스가 작성되었습니까?*
   - 루트로서 사용자와 그룹을 인스턴스 소유자 및 sysadm 그룹이 되도록 작성했습니까?
3. *적용 가능한 경우 라이센스 정보가 성공적으로 처리되었습니까?*
   - UNIX 운영 체제의 경우 노드 잠금 파일을 편집하고 IBM이 제공한 암호를 입력했습니까?

4. *IBM 메인프레임 데이터베이스 서버와 워크스테이션 통신이 올바르게 구성되었습니까?*
   - 다음과 같은 세 가지 구성을 고려해야 합니다.
     a. IBM 메인프레임 데이터베이스 서버 구성이 서버에 대한 응용프로그램 요청자를 식별합니다. IBM 메인프레임 서버 데이터베이스 관리 시스템이 위치 네트워크 프로토콜 및 보안과 관련하여 요청자를 정의합니다.
     b. DB2 Connect 워크스테이션 구성이 서버에 대한 클라이언트 모집단 및 클라이언트에 대한 IBM 메인프레임을 정의합니다.
     c. 클라이언트 워크스테이션 구성에 워크스테이션 이름과 통신 프로토콜이 정의되어야 합니다.
   - 초기 연결이 되지 않는 문제점 분석에는 PU (실제 단위) 이름이 완전하고 올바른지를 검증하거나 올바른 포트 번호 및 호스트 이름이 TCP/IP 연결에 지정되었는지 검증하는 것이 포함됩니다.
   - IBM 메인프레임 서버 데이터베이스 관리자와 네트워크 관리자에는 문제점 진단에 사용할 수 있는 유틸리티가 있습니다.
5. *IBM 메인프레임 서버 데이터베이스를 사용하기 위해 IBM 메인프레임 서버 데이터베이스 관리 시스템에서 요구하는 권한 레벨이 있습니까?*
   - 사용자의 액세스 권한, 테이블 규정자 규칙, 예상 결과를 고려하십시오.
6. *IBM 메인프레임 데이터베이스 서버에 대해 SQL문을 발행하기 위해 CLP(Command Line Processor) 사용을 시도하는 경우 실패했습니까?*
   - CLP를 IBM 메인프레임 데이터베이스 서버에 바인드하는 절차를 따랐습니까?

초기 연결 후 문제점 발생

다음은 문제점의 범위를 좁히는데 도움이 되는 시작점으로서 제공되는 질문입니다.

1. *특수 또는 일반적이지 않은 운영 체제가 있습니까?*
   - 새 응용프로그램입니까?
   - 새 프로시저를 사용 중입니까?
   - 최근에 시스템에 영향을 미칠 수 있는 변경이 있습니까? 예를 들어 응용프로그램이나 시나리오가 마지막으로 성공적으로 실행된 이후 소프트웨어 제품이나 응용프로그램이 변경되었습니까?
   - 응용프로그램의 경우 어떤 API(Application Programming Interface)를 사용하여 프로그램을 작성하였습니까?
   - 소프트웨어 또는 통신 API를 사용하는 기타 응용프로그램이 사용자의 시스템에 실행되었습니까?

- 최근에 FixPack을 설치하였습니까? 문제점이 발생한 경우 사용자가 설치 이후 운영 체제에서 사용되거나 로드되지 않은 기능을 사용하려고 시도할 때 IBM의 최근 FixPack을 판별하고 해당 기능을 설치한 *다음* 로드하십시오.

2. *이 오류가 발생한 적이 있습니까?*
   - 이전 오류 조건에 대한 문서화된 분석이 있습니까?
   - 구성원은 누구였으며 그들이 가능한 조치 과정에 식견을 제공할 수 있습니까?
3. *네트워크에 대한 정보를 리턴하는 통신 소프트웨어 명령을 사용하여 탐색한 적이 있습니까?*
   - TCP/IP에는 TCP/IP 명령과 상주 프로그램을 사용하여 검색된 유용한 정보가 있습니다.
4. *유용한 SQLCA(SQL 통신 영역)에 리턴된 정보가 있습니까?*
   - 문제점 처리 프로시저에는 SQLCODE 및 SQLSTATE 필드의 컨텐츠를 조사하는 단계가 포함되어 있습니다.
   - SQLSTATE를 사용하여 응용프로그램 프로그래머는 데이터베이스 제품의 DB2 제품군에 일반적인 오류 클래스를 시험할 수 있습니다. 분산된 관계형 데이터베이스 네트워크의 이 필드는 일반 기본을 제공합니다.
5. *START DBM이 서버에 실행되었습니까?* 추가적으로 DB2COMM 환경 변수가 원격으로 서버에 액세스하는 클라이언트에 올바르게 설정되었는지 확인하십시오.
6. *같은 태스크를 수행하는 다른 머신이 서버에 성공적으로 연결할 수 있습니까?* 서버 연결을 시도하는 클라이언트 수가 최대에 도달했습니다. 다른 클라이언트가 서버에서 연결을 끊으면 이전에 연결할 수 없었던 클라이언트가 지금은 연결할 수 있습니까?
7. *머신에 주소 지정이 올바릅니까?* 머신이 네트워크에서 고유한지 확인하십시오.
8. *원격으로 연결할 때 클라이언트에게 올바른 권한이 부여되었습니까?* 인스턴스 연결에는 성공했으나 데이터베이스 또는 테이블 레벨에서 권한이 부여되지 않았습니다.
9. *원격 데이터베이스에 연결된 것이 맨 처음의 머신입니까?* 분산 환경에서 네트워크 사이의 라우터 또는 브릿지가 클라이언트와 서버 간의 통신을 블로킹합니다. 예를 들어 TCP/IP를 사용할 때 원격 호스트를 PING할 수 있는지 확인하십시오.

지원되지 않는 DDM 명령

Linux, UNIX 및 Windows용 DB2 버전 9.5는 DRDA 응용프로그램 서버(DRDA AS)의 역할을 할 때 DDM 명령(BNDCPY, BNDDPLY, DRPPKG 및 DSCRDBTBL)을 지원하지 않습니다.

증상

DRDA 응용프로그램 리퀘스터(DRDA AR)가 Linux, UNIX 및 Windows용 DB2 버전 9.5에 연결하여 다음 명령을 실행하면, 명령이 실패합니다.

표 81. 지원되지 않는 DDM 명령

DDM 명령	DDM 코드 포인트	설명
BNDCPY	X'2011'	기존 관계형 데이터베이스(RDB) 패키지 복사
BNDDPLY	X'2016'	기존 RDB 패키지 전개
DRPPKG	X'2007'	패키지 삭제(drop)
DSCRDBTBL	X'2012'	RDB 테이블 설명

또한 매개변수 방식(또는 컬럼 방식) 배열 입력을 위해 SQLDTA 디스크립터에서 사용되는 다음 코드 포인트도 지원되지 않습니다.

표 82. 지원되지 않는 FD:OCA 데이터 오브젝트

FD:OCA 데이터 오브젝트	DDM 코드 포인트	설명
FDOEXT	X'147B'	FD:OCA(Formatted Data Object Content Architecture) 데이터 Extent
FDOOFF	X'147D'	FD:OCA 데이터 오프셋

이 상황에서 가장 일반적인 오류 메시지는 SQL30020N("후속 명령 및 SQL문의 성공적인 실행에 영향을 주는 분산 프로토콜 오류로 인해 실행이 실패했습니다.")입니다.

원인

DDM(Distributed Data Management Architecture)은 DRDA 프로토콜의 일부입니다. Linux, UNIX 및 Windows용 DB2 버전 9.5에서 지원하는 모든 DRDA 레벨에 DDM 명령(BNDCPY, BNDDPLY, DRPPKG 및 DSCRDBTBL)이 존재하지만 DRDA 응용프로그램 서버(AS)는 이러한 DDM 명령을 지원하지 않습니다.

마찬가지로, Linux, UNIX 및 Windows DRDA 응용프로그램 서버(AS)용 DB2 버전 9.5는 **FDOEXT** 및 **FDOOFF** 코드 포인트를 지원하지 않습니다. 이러한 코드 포인트는 컬럼 방식 배열 입력 요청을 제출할 때 서버로 전송되는 SQLDTA 디스크립터에서 사용됩니다.

문제점 진단

DRDA 응용프로그램 서버(AS)에 DB2 추적을 확보하는 경우, 이러한 명령에 응답하여 `ERROR MSG = Parser: Command Not Supported`와 유사한 메시지가 표시됩니다.

문제점 해결

현재 BNDCPY 및 BNDDPLY DDM 명령에 지원되는 대안은 없습니다.

패키지를 삭제(drop)하려면 DROP PACKAGE SQL문을 사용하십시오. 예를 들어, Linux, UNIX 및 Windows DRDA 응용프로그램 서버(AS)용 DB2 버전 9.5에 연결

하고 EXECUTE IMMEDIATE 요청에서 DROP PACKAGE SQL문을 보내십시오. Linux, UNIX 및 Windows용 DB2 버전 9.5가 해당 요청을 처리합니다.

RDB 테이블을 설명하려면, DSCSQLSTT(SQL문 설명) 또는 PRPSQLSTT(SQL문 준비) DDL 명령 중 하나를 사용하십시오. 예를 들어, TAB1 테이블에 대한 설명을 원하면 `SELECT * FROM TAB1` 명령문을 설명하거나 준비하십시오.

주: DRDA AR이 PRPSQLSTT 명령을 실행하는 경우, TRUE 값을 사용하여 **RTNSQLDA** 인스턴스 변수도 지정해야 합니다. 그렇지 않으면, 서버가 SQLDA 응답 데이터(SQLDARD) 디스크립터를 리턴하지 않습니다.

FDOEXT 및 **FDOOFF** 코드 포인트 문제점을 방지하려면, 매개변수 방식(또는 컬럼 방식) 배열 입력 요청 대신 행 방식 배열 입력 요청을 사용하십시오.

일반 DB2 Connect 문제점

이 주제에는 DB2 Connect 사용 시 발생하는 연결 문제점의 가장 일반적인 증상이 나열되어 있습니다. 각각의 경우에 다음이 제공됩니다.

- 해당 메시지와 연관된 메시지 번호 및 리턴 코드(또는 프로토콜에 대한 특정 리턴 코드)의 조합. 각 메시지 및 리턴 코드 조합에는 별도의 표제가 있고 표제의 순서는 메시지 번호, 그 다음으로는 리턴 코드에 따라 정해집니다.
- 일반적으로 샘플 메시지 목록 형태의 증상.
- 가능한 오류 원인을 표시하는 제안된 솔루션. 어떤 경우 제안된 솔루션이 두 개 이상 제공됩니다.

SQL0965 또는 SQL0969

증상 SQL0965 및 SQL0969 메시지는 IBM i용 DB2, z/OS용 DB2 및 DB2 서버에서 여러 개의 다른 리턴 코드로 발행될 수 있습니다.

이 메시지 중 하나가 발행되면 메시지를 발행한 데이터베이스 서버 제품의 문서에서 원래 SQL 코드를 조사해야 합니다.

솔루션 IBM 메인프레임 데이터베이스에서 수신된 SQL 코드를 변환할 수 없습니다. 오류 코드를 기반으로 문제점을 정정한 다음 실패 명령을 다시 제출하십시오.

SQL5043N

증상 하나 이상의 통신 프로토콜 지원이 시작되지 않았습니다. 그러나 핵심 데이터베이스 관리 프로그램의 기능은 시작되었습니다.

DB2 Connect 서버에서 TCP/IP 프로토콜이 시작되지 않았습니다. 이전에 클라이언트 연결에 성공했습니다.

다음 예와 같이 `diaglevel = 4`인 경우 db2diag 로그 파일에 유사한 항목이 있습니다.

```
2001-05-30-14.09.55.321092   Instance:svtdbm5   Node:000
PID:10296(db2tcpcm)   Appid:none
common_communication  sqlcctcpconnmgr_child   Probe:46
DIA3205E Socket address "30090" configured in the TCP/IP
services file and
required by the TCP/IP server support is being used by another
process.
```

솔루션 이 경고는 원격 클라이언트의 서버로 활동하는 DB2 Connect에 하나 이상의 통신 프로토콜을 처리하는 문제가 있다는 신호를 보내는 증상입니다. 이 프로토콜은 TCP/IP 및 기타 프로토콜이 될 수 있으며 메시지는 일반적으로 DB2 Connect에 정의된 통신 프로토콜 중 하나가 올바르게 구성되지 않았음을 나타냅니다.

DB2COMM 프로파일 변수가 정의되지 않았거나 올바르게 정의되지 않은 것이 원인이 되는 경우가 종종 있습니다. 일반적으로 문제점은 데이터베이스 관리 프로그램 구성에 정의된 DB2COMM 변수와 이름이 일치하지 않아서 발생합니다. (예: svcename 또는 nname)

가능한 시나리오는 구성은 변경되지 않았지만 이전에 연결에 성공한 다음 SQL5043 오류 메시지를 받은 경우입니다. 이것은 원격 시스템의 연결이 어떤 이유로 비정상적으로 종료될 때 TCP/IP 프로토콜을 사용하여 발생할 수 있습니다. 이러한 문제점이 발생하면 클라이언트에게는 연결이 계속 존재하는 것으로 나타나고 아래에 표시된 명령을 발행하여 더 이상 개입하지 않고 연결을 복구하는 것이 가능해집니다.

DB2 Connect 서버에 연결하는 클라이언트 중 하나는 계속 TCP/IP 포트를 처리할 가능성이 많습니다. DB2 Connect 서버에 연결된 각 클라이언트에서 다음 명령을 입력하십시오.

```
db2 terminate
db2stop
```

SQL30020

증상 SQL30020N 후속 명령 및 SQL문의 성공적인 실행에 영향을 주는 분산 프로토콜 오류로 인해 실행이 실패했습니다.

솔루션 이 오류로 인해 서비스에 접속되어야 합니다. 서비스에 접속하기 전에 db2support 명령을 실행하십시오.

SQL30060

증상 SQL30060N *"<authorization-ID>"*에는 조작 *"<operation>"*을 수행할 수 있는 특권이 없습니다.

솔루션 DB2에 접속할 때 CDB(Communications Database) 테이블이 올바르게 갱신되지 않았습니다.

SQL30061

증상 올바르지 않은 IBM 메인프레임 데이터베이스 서버 위치에 연결 - 목표 데이터베이스를 찾을 수 없습니다.

솔루션 DCS 디렉토리 항목에 올바르지 않은 서버 데이터베이스 이름이 지정되었습니다. 이 문제점이 발생하면 SQLCODE -30061이 응용프로그램에 리턴됩니다.

DB2 노드, 데이터베이스 및 DCS 디렉토리 항목을 확인하십시오. DCS 디렉토리 항목의 목표 데이터베이스 이름 필드는 플랫폼 기반 데이터베이스의 이름과 일치해야 합니다. 예를 들어 DB2 데이터베이스의 경우 사용될 이름은 BSDS(Boot Strap Data Set) ″LOCATION=*locname*″ 필드에 에 사용된 이름과 같아야 하며 DDF(Distributed Data Facility)가 시작될 때 DSNL004I 메시지((LOCATION=*location*)에도 제공됩니다.

TCP/IP 노드의 올바른 명령은 다음과 같습니다.

```
db2 catalog tcpip node <node_name> remote <host_name_or_address>
            server <port_no_or_service_name>
db2 catalog dcs database <local_name> as <real_db_name>
db2 catalog database <local_name> as <alias> at <node node_name>
            authentication server
```

데이터베이스에 연결하려면 다음을 실행하십시오.

```
db2 connect to <alias> user <user_name> using <password>
```

리턴 코드가 79인 SQL30081N

증상

```
SQL30081N  A communication error has been detected.
Communication protocol
being used: "TCP/IP".  Communication API being used: "SOCKETS".
위치   where the error was detected: "".  Communication function
detecting the error:
"connect".  Protocol specific error code(s): "79", "*", "*".
SQLSTATE=08001
```

솔루션 이 오류는 원격 클라이언트가 DB2 Connect 서버 연결에 실패하는 경우에 발생합니다. 또한 DB2 Connect 서버에서 IBM 메인프레임 데이터베이스 서버로 연결할 때도 발생할 수 있습니다.

1. DB2COMM 프로파일 변수가 DB2 Connect 서버에 올바르지 않게 설정되었습니다. 이를 확인하십시오. 예를 들어 AIX에 DB2 Enterprise Server Edition을 실행할 때 명령 db2set db2comm=tcpip가 sqllib/db2profile에 표시되어야 합니다.
2. IBM Data Server Client 및 DB2 Connect 서버에서 TCP/IP 서비스 이름과 포트 번호 스펙이 일치하지 않습니다. 두 개의 머신 모두에서 TCP/IP services 파일의 항목을 검증하십시오.

3. DB2 Connect 서버에서 DB2가 시작되었는지 확인하십시오. 다음 명령을 사용하여 데이터베이스 관리 프로그램 구성 diaglevel을 4로 설정하십시오.

   ```
   db2 update dbm cfg using diaglevel 4
   ```

   DB2를 중지하고 다시 시작한 다음 db2diag 로그 파일을 조사하여 DB2 TCP/IP 통신이 시작되었는지 확인하십시오. 다음과 유사한 출력이 표시되어야 합니다.

   ```
   2001-02-03-12.41.04.861119   Instance:svtdbm2   Node:00
   PID:86496(db2sysc)   Appid:none
   common_communication  sqlcctcp_start_listen   Probe:80
   DIA3000I "TCPIP" protocol support was successfully started.
   ```

프로토콜 특정 오류 코드가 10032인 SQL30081N

증상

```
SQL30081N  A communication error has been detected.
Communication protocol
being used: "TCP/IP".  Communication API being used: "SOCKETS".
위치   where the error was detected: "9.21.85.159".  Communication
function detecting
the error: "send".  Protocol specific error code(s): "10032",
"*", "*".
SQLSTATE=08001
```

솔루션 이 오류 메시지는 TCP/IP 통신이 이미 실패한 머신에서 연결을 끊으려 할 때 수신됩니다. TCP/IP 서브시스템으로 문제점을 정정하십시오.

대부분의 머신에서는 머신의 TCP/IP를 다시 시작해서 문제점을 간단히 정정할 수 있습니다. 경우에 따라서는 전체 머신을 재사용해야 합니다.

SQL30082 RC=24 During CONNECT

증상 SQLCODE -30082 제공된 사용자 이름 및/또는 암호가 올바르지 않습니다.

솔루션 필요한 경우 CONNECT문에 올바른 암호가 입력되었는지 확인하십시오. 목표 서버 데이터베이스에 전송할 암호를 사용할 수 없습니다. 암호를 IBM Data Server Client에서 목표 서버 데이터베이스로 전송해야 합니다. 어떤 플랫폼 (예: AIX)에서는 CONNECT문으로 제공되는 경우에만 암호를 얻을 수 있습니다.

제 8 장 지식 기반 검색

알려진 문제점을 효과적으로 검색하는 방법

DB2 APAR, 백서, IBM Redbooks® 서적, 기술 노트, 매뉴얼 등 알려진 문제점을 설명하는 사용 가능한 자원이 많이 있습니다. 발생하는 문제점에 대한 솔루션이 이미 존재하는지 여부를 신속히 판별할 수 있도록 이러한 자원과 기타 자원을 효과적으로 검색하는 것이 중요합니다.

검색하기 전에 문제점 상황을 명확히 이해해야 합니다.

문제점 상황이 무엇인지 명확히 이해했으면 기존 솔루션을 찾는 기회를 증가시키기 위해 검색 키워드 목록을 작성해야 합니다. 몇 가지 추가 정보는 다음과 같습니다.

1. 검색에서 여러 개의 단어를 사용하십시오. 사용하는 검색 용어가 적절할수록 보다 나은 검색 결과를 얻을 수 있습니다.
2. 특정 결과로 시작한 후 필요하면 결과 범위를 넓히십시오. 예를 들어, 리턴된 결과가 너무 적으면 관련성이 덜한 검색 용어 중 일부를 제거하고 다시 시도하십시오. 또는 사용할 키워드를 모를 경우, 몇 개의 키워드를 사용하여 광범위한 검색을 수행하고 수신하는 결과 유형을 살펴본 후 키워드를 추가로 선택할 수 있습니다.
3. 특정 구문을 검색하는 것이 보다 효과적인 경우도 있습니다. 예를 들어, ″administration notification file″(인용 부호 사용)을 입력하면 입력한 것과 똑같은 순서로 정확한 구문을 포함하는 해당 문서만 얻게 됩니다(세 개의 해당 단어 조합을 포함하는 모든 문서와 대조적임).
4. 와일드 카드를 사용하십시오. 특정 SQL 오류가 발생하는 경우, ″SQL5005<wildcard>″를 검색하십시오(여기서 <wildcard>는 검색 중인 자원에 적합한 와일드 카드임). 이렇게 하면 단순히 ″SQL5005″ 또는 ″SQL5005c″를 검색한 경우보다 많은 결과를 리턴할 가능성이 있습니다.
5. 인스턴스가 비정상적으로 종료되고 트랩 파일이 생성되는 상황이 발생하면, 트랩 또는 코어 파일의 스택 역추적에서 처음 2 - 3개의 기능을 사용하여 알려진 문제점을 검색하십시오. 너무 많은 결과가 리턴되면, ″trap″, ″abend″ 또는 ″crash″ 키워드를 추가하십시오.
6. 운영 체제에 특정한 키워드(예: 신호 번호 또는 오류 번호 값)를 검색 중인 경우, 값이 아닌 상수 이름을 검색하십시오. 예를 들어, 오류 번호 27 대신 ″EFBIG″를 검색하십시오.

일반적으로 성공적인 검색 용어에는 다음이 포함되는 경우가 많습니다.

- 실행된 명령을 설명하는 단어

- 증상을 설명하는 단어
- 진단 토큰

문제점 해결 자원

DB2 제품을 사용하는 데 도움이 되는 광범위한 문제점 해결 정보를 사용할 수 있습니다.

DB2 문서

DB2 정보 센터 전반은 물론 DB2 라이브러리를 구성하는 PDF 서적에서도 문제점 해결 정보를 찾을 수 있습니다.

DB2 기술 지원부 웹 사이트

문제점이 발생했을 때 가능한 원인 및 솔루션을 찾으려면 DB2 기술 지원 웹 사이트를 참조하십시오. 기술 지원 사이트에는 최신 DB2 서적, 기술 노트, APAR(Authorized Program Analysis Report), FixPack 및 기타 자원으로의 링크가 포함되어 있습니다. 이 지식 기반을 검색하여 문제점에 대해 가능한 솔루션을 찾으십시오.

DB2 기술 지원부 웹 사이트(www.ibm.com/software/data/db2/support/db2_9/)에 액세스하십시오.

제 9 장 DB2 제품 수정사항 얻기

FixPack에는 코드 갱신 및 IBM이 제품 테스트 중에 그리고 고객이 제품을 사용할 때 발견한 문제점에 대한 수정사항이 포함됩니다. 최근 FixPack 찾는 방법 및 데이터베이스 환경에 수정사항을 적용하는 방법에 대한 설명이 있습니다.

수정사항 얻기

문제점을 해결하기 위해 제품 수정사항을 사용할 수 있습니다. 다음 단계를 따라 수정사항을 얻을 수 있습니다.

1. 각각 다음 웹 페이지에서 수정 목록을 보고 FixPack을 얻을 수 있습니다.
   - Linux, UNIX 및 Windows 지원용 DB2 9
   - Linux, UNIX 및 Windows용 DB2 버전별 수정사항
2. 필요한 FixPack을 판별하십시오. 일반적으로, 이미 알려져 정정된 소프트웨어 결함으로 인한 문제점 발생을 방지하기 위해 최신 FixPack 설치를 권장합니다.
3. FixPack을 다운로드하고 자기 압축 해제를 실행할 수 있는 패키지를 두 번 눌러 파일 압축을 해제하십시오. `SERVER/doc/your_language/readme.txt` 문서를 열고 제공된 DB2 정보 센터 링크를 따라 설치 지시사항을 확보하십시오.
4. 수정사항을 적용하십시오. 지시사항은 *DB2 Server 설치*의 『FixPack 적용』을 참조하십시오.

FixPack, 임시 FixPack 및 테스트 수정사항

APAR(Authorized Program Analysis Report)은 현재 변경되지 않은 IBM 프로그램 릴리스에서 예측되는 결함으로 인해 발생한 문제점에 대한 공식 보고서입니다. APAR은 IBM에서 테스트 중에 발견한 문제점은 물론 고객이 보고한 문제점도 설명합니다.

APAR에 설명된 문제점을 해결하는 수정된 DB2 코드는 FixPack, 임시 FixPack 및 테스트 수정사항으로 전달될 수 있습니다.

FixPack

FixPack은 APAR 수정사항 누적 콜렉션입니다. 특히, FixPack은 DB2 새 릴리스 간에 발생하는 APAR을 해결합니다. 특정 유지보수 레벨까지 이동할 수 있게 하기 위한 것입니다. FixPack의 특성은 다음과 같습니다.

- 누적됩니다. 특정 릴리스의 DB2에 대한 FixPack은 해당 릴리스에 대한 이전 FixPack 및 임시 FixPack에 제공된 모든 APAR 수정사항을 대체 또는 포함합니다.
- 지원되는 모든 운영 체제와 DB2 데이터베이스 제품에 사용할 수 있습니다.

- 많은 APAR을 포함합니다.
- DB2 기술 지원부 웹 사이트에서 발행되며, 일반적으로 Passport Advantage® 프로그램의 제품을 구입한 고객이 사용할 수 있습니다.
- IBM에서 충분한 테스트를 수행했습니다.
- 데이터베이스 제품 변경사항, FixPack 설치 및 제거 방법을 설명하는 문서를 수반합니다.

주: APAR 수정사항이 FixPack으로 제공되면 APAR 상태가 ″Open″에서 ″Closed as program error″로 변경됩니다. DB2 기술 지원부 웹 사이트에서 APAR 설명을 검토하여 개별 APAR의 상태를 판별할 수 있습니다.

임시 FixPack

임시 FixPack은 FixPack 간에 발생하는 중요한 APAR 수정사항 누적 콜렉션입니다. 임시 FixPack에 포함되려면 APAR이 보편적이거나 그렇지 않으면 중요하다고 간주되어야 합니다. DB2 기술 지원 팀이 전문가가 후보 APAR을 평가하고 승인합니다. 임시 FixPack의 특성은 다음과 같습니다.

- 누적됩니다. 특정 릴리스의 DB2에 대한 임시 FixPack은 해당 릴리스에 대한 이전 FixPack 및 임시 FixPack에 제공된 모든 APAR 수정사항을 대체 또는 포함합니다.
- 운영 체제 및 DB2 데이터베이스 제품 서브세트에 사용할 수 있습니다.
- 일반적으로 20 - 30개의 APAR을 포함합니다.
- DB2 기술 지원부 웹 사이트에서 발행되며, 일반적으로 Passport Advantage 프로그램의 제품을 구입한 고객이 사용할 수 있습니다.
- IBM에서 충분한 테스트를 수행했습니다.
- 임시 FixPack 설치 및 제거 방법을 설명하는 문서를 수반합니다.

임시 FixPack은 공개된 후 2년간 생산적으로 지원됩니다. FixPack 사이의 대략적인 중간점에서 사용 가능하며, 테스트 수정사항(동일한 테스트 레벨을 수신하거나 임시 FixPack과 동일한 지원 레벨을 갖지 않음)에 대한 선호 대안으로 사용됩니다.

테스트 수정사항

테스트 수정사항은 보고된 문제점에 응답하여 테스트를 위해 특정 고객에게 제공되는 임시 솔루션입니다. 테스트 수정사항을 ″특수 빌드″라고 하는 경우도 있으며 특성은 다음과 같습니다.

- 일반적으로 하나의 APAR을 포함합니다.
- DB2 지원부에서 얻을 수 있으며, 일반적으로 대중에게는 공개되지 않습니다.
- 제한된 IBM 테스트를 거칩니다.

- 테스트 수정사항 적용 방법에 대한 설명, 수정하는 APAR, 테스트 수정사항 제거 지시사항을 포함하여 최소 문서를 포함합니다.

  테스트 수정사항은 새로운 문제점이 발견되었고 문제점에 대한 일시적인 해결책이나 우회법이 없으며, 다음 FixPack이나 임시 FixPack이 제공될 때까지 기다릴 수 없는 상황에 제공됩니다. 예를 들어, 문제점이 비즈니스에 중요한 영향을 초래하는 경우 FixPack 또는 임시 FixPack에서 APAR이 해결될 때까지 상황을 완화하기 위해 테스트 수정사항이 제공될 수 있습니다.

문제점 없이 조작을 수행하려면 최신 FixPack에서 DB2 환경을 실행할 것을 권장합니다. 새 FixPack 사용 가능성에 대한 통지를 수신하려면, DB2 기술 지원부 웹 사이트(http://www.ibm.com/software/data/db2/support/db2_9/)의 ″My notifications″ 전자 우편 갱신에 가입하십시오.

DB2 수정사항 및 FixPack의 역할과 목적에 대해 자세히 알려면, support policy statement를 참조하십시오.

테스트 수정사항 적용

테스트 수정사항은 보고된 문제점에 응답하여 테스트를 위해 특정 고객에게 제공되는 임시 수정사항입니다. 모든 테스트 수정사항에는 Readme 파일이 수반됩니다. 테스트 수정사항 Readme 파일은 테스트 수정사항을 설치 및 설치 제거하기 위한 지시사항과 테스트 수정사항에 포함된 APAR(있는 경우)을 제공합니다.

각 테스트 수정사항마다 전제조건이 있습니다. 자세한 내용은 테스트 수정사항에 수반되는 Readme 파일을 참조하십시오.

다음 두 가지 유형의 테스트 수정사항이 있습니다.

- 개별 DB2 제품에 대한 테스트 수정사항. 이러한 테스트 수정사항을 기존 제품 설치에 적용하거나 기존 DB2 설치가 없는 전체 제품 설치를 수행하는 데 사용할 수 있습니다.
- 범용 테스트 수정사항(Linux 및 UNIX 전용). 범용 테스트 수정사항은 두 개 이상의 DB2 제품이 설치된 설치를 서비스합니다.

자국어가 설치된 경우, 별도의 자국어 테스트 수정사항도 필요할 수 있습니다. 설치된 DB2 제품과 테스트 수정사항 레벨이 동일한 경우에만 자국어 테스트 수정사항을 적용할 수 있습니다. 범용 테스트 수정사항을 적용하는 경우, DB2 제품을 갱신하려면 범용 테스트 수정사항과 자국어 테스트 수정사항을 둘 다 적용해야 합니다.

IBM Software Support에서 테스트 수정사항을 확보하고 테스트 수정사항 설치, 테스트 및 제거(필요한 경우)와 관련한 Readme 파일의 지시사항을 따르십시오.

다중 파티션 데이터베이스 파티션 환경에 테스트 수정사항을 설치하는 경우, 시스템이 오프라인 상태여야 하며 인스턴스에 참여하는 모든 컴퓨터를 동일한 테스트 수정사항 레벨로 업그레이드해야 합니다.

제 10 장 문제점 해결에 대한 자세한 내용

DB2 데이터베이스 제품으로 작업 중에 어떤 점에서 문제점이 발생할 수 있습니다. 이 문제점은 사용자가 데이터베이스에 『문제점이 발생했음』을 피드백하면 데이터베이스 관리 프로그램, 데이터베이스에 대해 실행되는 응용프로그램 또는 사용자에 의해 보고됩니다.

여기에 제시된 개념 및 도구는 데이터베이스 조작 시 실제 문제점 또는 인지되는 문제점을 소개하고 이러한 문제점에 대해 도움을 줍니다. 적시에 적절한 데이터 캡처의 중요성이 강조되므로 처음의 어커런스 데이터 캡처가 설명된 최초의 도구입니다. 데이터베이스의 조작에 대한 데이터를 캡처하기 위해 데이터베이스 관리 프로그램에 사용되는 기타 로그 및 파일이 운영 체제 진단 도구에 대한 언급과 함께 표시됩니다.

자세한 내용

다음 주제는 DB2 제품의 문제점을 효과적으로 해결하는 데 필요한 개념 정보를 획득하는 데 도움을 줄 수 있습니다.

- 문제점 해결 정보

  문제점 해결은 문제점을 해결하기 위한 체계적인 접근 방법입니다. 목적은 무언가가 예측한 대로 작동하지 않는 이유와 문제점 해결 방법을 판별하는 것입니다.

- 관리 통지 로그 파일 정보

  DB2 데이터베이스 관리 프로그램은 관리 통지 로그에 DB2 유틸리티(예: REORG 및 BACKUP)의 상태, 클라이언트 응용프로그램 오류, 서비스 클래스 변경사항, 라이센스 활동, 로그 파일 경로 및 스토리지 문제점, 모니터링 및 인덱싱 활동, 테이블스페이스 문제점을 기록합니다. 데이터베이스 관리자는 이 정보를 사용하여 문제점을 진단하고 데이터베이스를 조정하거나 데이터베이스를 모니터할 수 있습니다.

- DB2 진단(db2diag) 로그 파일 정보

  표준화된 메시지 형식을 사용하여 관리 통지 로그 메시지도 db2diag 로그 파일에 로그되므로 db2diag 로그 파일을 살펴보는 것은 데이터베이스에 발생하는 사항을 이해하는 탁월한 첫 번째 태스크입니다.

- 플랫폼별 오류 로그 정보

  문제점 분석을 돕기 위해 DB2 밖에서 사용 가능한 기타 많은 파일과 유틸리티가 있습니다. DB2 파일에서 정보를 사용하는 것 못지 않게 근본 원인을 판별하는 것이 중요한 경우가 많습니다.

- 메시지 정보

메시지에 대해 자세히 알면 오류나 문제점을 식별하고 적절한 복구 조치를 사용하여 문제점을 해결할 수 있습니다. 이 정보는 또한 메시지가 생성되고 기록되는 위치를 이해하는 데에도 사용할 수 있습니다.

- 내부 리턴 코드 정보

  두 가지 유형의 내부 리턴 코드(ZRC 값 및 ECF 값)가 있습니다. 내부 리턴 코드는 DB2 추적 결과 및 db2diag 로그 파일에 표시됩니다. ZRC 및 ECF 값은 일반적으로 음수이며 오류 조건을 나타내는 데 사용됩니다.

- 덤프 파일 정보

  덤프 파일은 문제점 진단 시 유용한 추가 정보(예: 내부 제어 블록)가 있는 오류가 발생할 때 작성됩니다. 덤프 파일에 기록된 모든 데이터 항목에는 문제점 판별을 돕기 위해 연관된 시간소인이 있습니다. 덤프 파일은 2진 형식으로 되어 있으며 DB2 고객 지원 담당자가 사용하도록 하기 위한 것입니다.

- 트랩 파일 정보

  DB2는 트랩, 세그먼트 위반 또는 예외로 인해 처리를 계속할 수 없는 경우 트랩 파일을 생성합니다. DB2가 수신하는 모든 신호 또는 예외는 트랩 파일에 기록됩니다. 또한 트랩 파일은 오류가 발생할 당시 실행 중이던 함수 시퀀스를 포함합니다. 이 시퀀스를 ″함수 호출 스택″ 또는 ″스택 추적″이라고 하는 경우도 있습니다. 트랩 파일은 신호 또는 예외가 발생했을 때 프로세스 상태에 대한 추가 정보도 포함합니다.

- FODC(First Occurrence Data Capture) 정보

  FODC(First Occurrence Data Capture)는 DB2 인스턴스에 대한 시나리오 기반 데이터를 캡처하는 데 사용되는 프로세스입니다. 특정 증상을 기반으로 DB2 사용자가 수동으로 FODC를 호출하거나 사전 판별된 시나리오 또는 증상이 발견된 경우 자동으로 호출할 수 있습니다. 이 정보를 사용할 경우, 진단 정보를 얻기 위해 오류를 재생해야 하는 필요성이 줄어듭니다.

- 콜아웃 스크립트(db2cos) 출력 파일 정보

  db2cos 스크립트는 데이터베이스 관리 프로그램이 심각한 상황, 트랩, 세그먼트 위반 또는 예외로 인해 처리를 계속할 수 없는 경우 디폴트로 호출됩니다.

- DB2 및 OS 진단 조합 정보

  메모리, 스왑 파일, CPU, 디스크 스토리지 및 기타 자원과 관련된 일부 문제점을 진단하려면 주어진 운영 체제가 이러한 자원을 관리하는 방법을 완전히 이해해야 합니다. 최소한 자원 관련 문제점을 정의하려면 존재하는 해당 자원의 양, 사용자당 존재할지도 모르는 자원 한계를 알아야 합니다.

관리 통지 로그

관리 통지 로그(*instance_name*.nfy)는 여러 개의 데이터베이스 관리 및 유지보수 활동에 대한 정보를 얻을 수 있는 저장소입니다. 데이터베이스 관리자는 이 정보를 사용하여 문제점을 진단하고 데이터베이스를 조정하거나 데이터베이스를 모니터할 수 있습니다.

DB2 데이터베이스 관리 프로그램은 UNIX 및 Linux 운영 체제 플랫폼에 다음과 같은 정보 유형을 관리 통지 로그에 작성합니다. Windows 운영 체제 플랫폼에는 이벤트 로그를 사용하여 관리 통지 이벤트를 기록합니다.

- DB2 유틸리티 상태(예: REORG 및 BACKUP)
- 클라이언트 응용프로그램 오류
- 서비스 클래스 변경
- 라이센스 부여 활동
- 파일 경로
- 스토리지 문제점
- 모니터링 활동
- 인덱싱 활동
- 테이블 스페이스 문제점

관리 통지 로그 메시지도 표준화된 메시지 형식을 사용하여 db2diag 로그 파일에 로그됩니다.

통지 메시지에는 제공된 SQLCODE를 보충하는 추가 정보가 있습니다.

관리 통지 로그 파일은 다음의 서로 다른 두 양식으로 존재할 수 있습니다.

단일 관리 통지 로그 파일

로그 파일의 *크기*가 무한정 커지는 사용 중인 하나의 관리 통지 로그 파일(*instance_name*.nfy)입니다. 이는 디폴트 폼이며, **diagsize** 데이터베이스 관리 프로그램 구성 매개변수 값이 0(이 매개변수의 디폴트값은 0임)일 때마다 존재합니다.

순환하는 관리 통지 로그 파일

diagpath 구성 매개변수가 정의한 위치에서 일련의 관리 로그 파일을 찾을 수 있지만 사용 중인 단일 로그 파일입니다(이름은 *instance_name*.*N*.nfy임. 여기서, *N*은 0에서 시작하여 연속적으로 증가하는 파일 이름 인덱스임). 로그 파일은 각각 제한된 *크기*에 도달할 때까지 커지며, 제한된 *크기*에 도달하면 로그 파일이 닫히고 증분된 파일 이름 인덱스(*instance_name*.*N+1*.nfy)를 사용하여 새 로그 파일이 작성되어 로깅을 위해 열립니다. **diagsize** 데이터베이스 관리 프로그램 구성 매개변수 값이 0이 아닐 때마다 존재합니다.

주: Windows 운영 체제 플랫폼에는 단일 관리 통지 로그 파일 및 회전 관리 통지 로그 파일을 모두 사용할 수 없습니다.

diagsize 데이터베이스 관리 프로그램 구성 매개변수를 적절히 설정하여 시스템에 존재하는 이 두 양식 중에서 선택할 수 있습니다.

구성

다음 데이터베이스 관리 프로그램 구성 매개변수를 설정하여 관리 로그 파일의 *크기*, 위치 및 기록되는 세부사항의 이벤트 유형과 레벨을 구성할 수 있습니다.

diagsize

diagsize 값은 채택할 관리 통지 로그 파일의 양식을 결정합니다. 값이 0이면, 단일 관리 통지 로그 파일을 채택합니다. 값이 0이 아니면, 순환하는 관리 통지 로그 파일을 채택하며, 이 0이 아닌 값은 모든 회전 진단 로그 파일 및 모든 순환하는 관리 통지 로그 파일의 전체 *크기*도 지정합니다. **diagsize** 매개변수의 새 값을 적용하려면 인스턴스를 재시작해야 합니다. 자세한 내용은 ″diagsize - 진단 로그 파일 *크기* 구성 매개변수″ 주제를 참조하십시오.

diagpath

diagpath 구성 매개변수가 정의한 위치의 관리 통지 로그 파일에 진단 정보가 기록되도록 지정할 수 있습니다. 자세한 내용은 ″diagpath - 진단 데이터 디렉토리 경로 구성 매개변수″ 주제를 참조하십시오.

notifylevel

관리 통지 로그 파일에 작성된 이벤트 유형과 세부사항 레벨은 **notifylevel** 구성 매개변수로 지정할 수 있습니다. 자세한 내용은 ″notifylevel - 통지 레벨 구성 매개변수″ 주제를 참조하십시오.

관리 통지 로그 파일 항목 해석

텍스트 편집기를 사용하여 문제점이 발생했다고 예측되는 시스템에서 관리 통지 로그 파일을 볼 수 있습니다. 기록된 가장 최근의 이벤트가 파일 맨 아래에 있습니다.

일반적으로 각 항목에는 다음 파트가 포함됩니다.

- 시간소인
- 오류를 보고하는 위치. 응용프로그램 ID를 사용하여 서버 및 클라이언트 로그에서 응용프로그램에 속한 항목을 비교할 수 있습니다.
- 오류를 설명하는 진단 메시지(일반적으로 ″DIA″ 또는 ″ADM″으로 시작).
- 사용 가능한 모든 지원 데이터(예: SQLCA 데이터 구조 및 추가 덤프 또는 트랩 파일의 위치를 가리키는 포인터).

다음 예는 샘플 로그 항목에 대한 헤더 정보를 식별된 로그의 모든 파트와 함께 표시합니다.

주: 모든 로그 항목에 이러한 파트가 모두 포함되는 것은 아닙니다.

```
2006-02-15-19.33.37.630000 1     Instance:DB2 2    Node:000 3
PID:940(db2syscs.exe) TID: 660 4    Appid:*LOCAL.DB2.020205091435 5
recovery manager 6   sqlpresr 7   Probe:1 8    Database:SAMPLE   9
ADM1530E 10   Crash recovery has been initiated. 11
```

범례:

1. 메시지의 시간소인
2. 메시지를 작성 중인 인스턴스의 이름
3. 다중 파티션 시스템의 경우, 메시지를 작성 중인 데이터베이스 파티션(파티션되지 않은 데이터베이스에서는 값이 ″000″임)
4. 메시지 생성을 담당하는 프로세스 ID(PID)(그 다음에 프로세스 이름과 스레드 ID(TID)가 차례로 옴)
5. 프로세스가 작업 중인 응용프로그램의 ID. 이 예에서, 메시지를 작성 중인 프로세스는 ID가 *LOCAL.DB2.020205091435인 응용프로그램을 대신하여 작업합니다.

   이 값은 **appl_id** 모니터 요소 데이터와 동일합니다. 이 값을 해석하는 방법에 대한 자세한 정보는 **appl_id** 모니터 요소에 대한 문서를 참조하십시오.

   특정 응용프로그램 ID에 대해 자세히 식별하려면, 다음 중 어느 하나를 수행하십시오.

   - DB2 서버에서 LIST APPLICATIONS 명령 또는 DB2 Connect 게이트웨이에서 LIST DCS APPLICATIONS를 사용하여 응용프로그램 ID 목록을 보십시오. 이 목록에서 오류가 발생하는 클라이언트에 대한 정보(예: 노드 이름 및 TCP/IP 주소)를 판별할 수 있습니다.
   - GET SNAPSHOT FOR APPLICATION 명령을 사용하여 응용프로그램 ID 목록을 보십시오.
6. 메시지를 작성 중인 DB2 구성요소. db2AdminMsgWrite API를 사용하여 사용자 응용프로그램이 작성한 메시지의 경우, 구성요소는 『사용자 응용프로그램』을 나타냅니다.
7. 메시지를 제공 중인 함수의 이름. 이 함수는 메시지를 작성 중인 DB2 구성요소에서 작동합니다. db2AdminMsgWrite API를 사용하여 사용자 응용프로그램이 작성한 메시지의 경우, 함수는 『USER 함수』를 나타냅니다.
8. 고유 내부 ID. 이 번호를 사용하여 DB2 고객 지원 및 개발 팀에서는 메시지를 보고한 DB2 소스 코드의 위치를 찾을 수 있습니다.
9. 오류가 발생한 데이터베이스
10. 사용 가능한 경우, 16진수 코드로 오류 번호 및 유형을 표시하는 메시지
11. 사용 가능한 경우, 로그된 이벤트를 설명하는 메시지 텍스트

관리 통지 로그 파일의 오류 캡처 레벨 설정

이 태스크는 관리 통지 로그 파일의 오류 캡처 레벨 설정 방법을 설명합니다.

DB2가 관리 통지 로그에 기록하는 정보는 **NOTIFYLEVEL** 설정으로 결정됩니다.

- 현재 설정을 점검하려면, GET DBM CFG 명령을 실행하십시오.

  다음 변수를 찾으십시오.

  ```
  Notify Level            (NOTIFYLEVEL) = 3
  ```

- 설정을 변경하려면, UPDATE DBM CFG 명령을 사용하십시오. 예를 들어, 다음과 같습니다.

  ```
  DB2 UPDATE DBM CFG USING NOTIFYLEVEL X
  ```

  여기서 *X*는 원하는 통지 레벨입니다.

DB2 진단(db2diag) 로그 파일

DB2 진단 db2diag 로그 파일은 문제점 해결 목적으로 주로 IBM Software Support에서 사용하도록 하기 위한 것입니다. 관리 통지 로그는 문제점 해결을 위해 주로 데이터베이스 및 시스템 관리자가 사용하도록 하기 위한 것입니다. 관리 통지 로그 메시지도 표준화된 메시지 형식을 사용하여 db2diag 로그 파일에 로그됩니다.

개요

db2diag 로그 파일에 DB2 진단 및 관리 통지 메시지가 둘 다 로그되므로 데이터베이스 운영에 대한 정보를 얻기 위해 db2diag 로그 파일을 먼저 조사하는 경우가 많습니다. 이러한 진단 로그 파일의 컨텐츠 해석에 대한 도움은 ″관련 링크″ 섹션에 나열된 주제에 제공됩니다. 문제점 해결을 시도해도 문제점을 해결할 수 없고 도움을 받아야 한다고 생각되면, IBM Software Support에 문의할 수 있습니다(자세한 내용은 ″IBM Software Support에 문의″ 주제 참조). IBM Software Support로 보내도록 요청될 관련 진단 정보를 수집할 때 기타 관련 로그, 스토리지 덤프 및 추적을 포함하는 기타 정보 소스 중에 db2diag 로그 파일을 포함시킬 수 있습니다.

db2diag 로그 파일은 다음 서로 다른 두 양식으로 존재할 수 있습니다.

단일 진단 로그 파일

로그 파일 *크기*가 무한정 커지는 사용 중인 하나의 진단 로그 파일입니다(이름은 db2diag.log임). 이는 디폴트 폼이며, **diagsize** 데이터베이스 관리 프로그램 구성 매개변수 값이 0(이 매개변수의 디폴트값은 0임)일 때마다 존재합니다.

회전 진단 로그 파일

diagpath 구성 매개변수가 정의한 위치에서 일련의 진단 로그 파일을 찾을 수 있긴 하지만 사용 중인 단일 로그 파일입니다(이름은 db2diag.*N*.log임. 여기서, *N*은 0에서 시작하여 연속적으로 증가하는 파일 이름 인덱스임). 로그 파일은 각각 제한된 *크기*에 도달할 때까지 커지며, 제한된 *크기*에 도달하면 로그 파일

이 닫히고 증분된 파일 이름 인덱스(db2diag.*N+1*.log)를 사용하여 새 로그 파일이 작성되어 로깅을 위해 열립니다. **diagsize** 데이터베이스 관리 프로그램 구성 매개변수 값이 0이 아닐 때마다 존재합니다.

diagsize 데이터베이스 관리 프로그램 구성 매개변수를 적절히 설정하여 시스템에 존재하는 이 두 양식 중에서 선택할 수 있습니다.

구성

다음 데이터베이스 관리 프로그램 구성 매개변수를 설정하여 db2diag 로그 파일의 크기, 위치 및 기록되는 진단 오류 유형을 구성할 수 있습니다.

diagsize

diagsize 값은 채택할 진단 로그 파일의 양식을 결정합니다. 값이 0이면, 단일 진단 로그 파일을 채택합니다. 값이 0이 아니면, 회전 진단 로그 파일을 채택하며, 이 0이 아닌 값은 모든 회전 진단 로그 파일 및 모든 회전 관리 통지 로그 파일의 전체 크기도 지정합니다. **diagsize** 매개변수의 새 값을 적용하려면 인스턴스를 재시작해야 합니다. 자세한 내용은 ″diagsize - 진단 로그 파일 크기 구성 매개변수″ 주제를 참조하십시오.

diagpath

diagpath 구성 매개변수가 정의한 위치의 db2diag 로그 파일에 진단 정보가 기록되도록 지정할 수 있습니다. 자세한 내용은 ″diagpath - 진단 데이터 디렉토리 경로 구성 매개변수″ 주제를 참조하십시오.

diaglevel

diaglevel 구성 매개변수를 사용하여 db2diag 로그 파일에 기록되는 진단 오류 유형을 지정할 수 있습니다. 자세한 내용은 ″diaglevel - 진단 오류 캡처 레벨 구성 매개변수″ 주제를 참조하십시오.

진단 로그 파일 항목 해석

db2diag 로그 파일을 필터링하고 형식화하려면 db2diag 로그 파일 분석 도구(db2diag)를 사용하십시오. 표준화된 메시지 형식을 사용하여 관리 통지 로그 메시지도 db2diag 로그 파일에 로그되므로 데이터베이스에 발생되는 사항을 이해하려면 db2diag 로그 파일을 먼저 살펴보는 것이 좋습니다.

db2diag를 사용하는 대신 텍스트 편집기를 사용하여 문제점이 발생했다고 예측되는 시스템에서 진단 로그 파일을 볼 수 있습니다. 기록된 가장 최근의 이벤트가 파일 맨 아래에 있습니다.

주: 관리 통지(*instance_name*.nfy) 및 진단(db2diag.log) 로그는 단일 로그 파일로 연속적으로 증가합니다. **diagsize** 데이터베이스 관리 프로그램 구성 매개변수가 0이 아닌

값으로 설정된 경우, 관리 통지 및 db2diag 로그 파일 모두 **diagsize** 구성 매개변수 값으로 판별된 제한된 전체 크기를 갖는 일련의 회전 로그 파일(*instance_name*.*N*.nfy 및 db2diag.*N*.log)이 됩니다.

다음 예는 샘플 로그 항목에 대한 헤더 정보를 식별된 로그의 모든 파트와 함께 표시합니다.

주: 모든 로그 항목에 이러한 파트가 모두 포함되는 것은 아닙니다. 처음 몇 개의 필드(TID 시간소인)와 FUNCTION만 모든 db2diag 로그 파일 레코드에 있습니다.

```
2007-05-18-14.20.46.973000-240 1  I27204F655 2  LEVEL: Info 3
PID : 3228 4  TID : 8796 5  PROC : db2syscs.exe 6
INSTANCE: DB2MPP 7  NODE : 002 8  DB : WIN3DB1 9
APPHDL : 0-51 10  APPID: 9.26.54.62.45837.070518182042 11
AUTHID : UDBADM 12
EDUID : 8796 13  EDUNAME: db2agntp 14  (WIN3DB1) 2
FUNCTION: 15  DB2 UDB, data management, sqldInitDBCB, probe:4820
DATA #1 : 16  String, 26 bytes
Setting ADC Threshold to:
DATA #2 : unsigned integer, 8 bytes
1048576
```

범례:

1. 메시지의 시간소인 및 시간대.

   주: db2diag 로그 파일의 시간소인에는 시간대가 포함되어 있습니다 (예: `2006-02-13-14.34.35.965000-300`). 여기서 "`-300`"은 세계 표준시(UTC, 이전에는 GMT라고 함)와 응용프로그램 서버(AS)의 분 단위 로컬 시간 간의 차이입니다. 따라서 -300은 UTC - 5시간을 나타냅니다(예: EST(Eastern Standard Time)).

2. 레코드 ID 필드. db2diag 로그 파일의 레코드 ID는 현재 메시지가 로그되고 있는 파일 오프셋(예: 『27204』) 및 DB2 진단 로그가 작성된 플랫폼의 메시지 길이(예: 『655』)를 지정합니다.

3. 오류 메시지와 연관된 진단 레벨(예: `Info`, `Warning`, `Error`, `Severe` 또는 `Event`)

4. 프로세스 ID

5. 스레드 ID

6. 프로세스 이름

7. 메시지를 작성 중인 인스턴스의 이름

8. 다중 파티션 시스템의 경우, 메시지를 작성 중인 데이터베이스 파티션(파티션되지 않은 데이터베이스에서는 값이 "000"임)

9. 데이터베이스 이름

10. 응용프로그램 핸들. 이 값은 db2pd 출력 및 잠금 덤프 파일에서 사용된 값과 일치합니다. 먼저 코디네이터 파티션 번호가 오고, 그 다음에 대시로 분리된 코디네이터 인덱스 번호가 옵니다.

11. 프로세스가 작업 중인 응용프로그램의 ID. 이 예에서, 메시지를 작성 중인 프로세스는 ID가 9.26.54.62.45837.070518182042인 응용프로그램을 대신하여 작업합니다.

TCP/IP 생성 응용프로그램 ID는 세 개 섹션으로 이루어져 있습니다.

1. **IP 주소**: 최대 8개의 16진수 문자로 표시되는 32비트 숫자로 표시됩니다.
2. **포트 번호**: 4개의 16진수 문자로 표시됩니다.
3. 이 응용프로그램 인스턴스의 **고유 ID**.

주: IP 주소 또는 포트 번호의 16진수 버전이 0 - 9로 시작하는 경우, G - P로 변경됩니다. 예를 들어, ″0″은 ″G″에 맵핑되고 ″1″은 ″H″에 맵핑됩니다. IP 주소(AC10150C.NA04.006D07064947)는 다음과 같이 해석됩니다. IP 주소는 AC10150C(172.16.21.12로 변환됨)로 남아 있습니다. 포트 번호는 NA04입니다. 첫 번째 문자는 ″N″(″7″에 맵핑됨)입니다. 따라서 16진수 양식의 포트 번호는 7A04(10진수 양식으로는 31236으로 변환됨)입니다.

이 값은 *appl_id* 모니터 요소 데이터와 동일합니다. 이 값을 해석하는 방법에 대한 자세한 정보는 *appl_id* 모니터 요소에 대한 문서를 참조하십시오.

특정 응용프로그램 ID에 대해 자세히 식별하려면, 다음 중 어느 하나를 수행하십시오.

- DB2 서버에서 LIST APPLICATIONS 명령 또는 DB2 Connect 게이트웨이에서 LIST DCS APPLICATIONS를 사용하여 응용프로그램 ID 목록을 보십시오. 이 목록에서 오류가 발생하는 클라이언트에 대한 정보(예: 데이터베이스 파티션 이름 및 TCP/IP 주소)를 판별할 수 있습니다.
- GET SNAPSHOT FOR APPLICATION 명령을 사용하여 응용프로그램 ID 목록을 보십시오.
- db2pd -applications -db <dbname> 명령을 사용하십시오.

12 권한 부여 ID

13 엔진 디스패치 가능 단위 ID

14 엔진 디스패치 가능 단위의 이름

15. 제품 이름(″DB2″), 구성요소 이름(『data management』), 메시지를 작성 중인 함수 이름(『sqlInitDBCB』) 및 함수 내 프로브 포인트(『4820』)

16. 피호출 함수가 리턴한 정보. 리턴된 데이터 필드가 여러 개일 수 있습니다.

샘플 db2diag 로그 파일 항목을 살펴보았으므로 가능한 모든 필드 목록은 다음과 같습니다.

```
<timestamp><timezone>       <recordID>         LEVEL: <level> (<source>)
PID     : <pid>             TID  : <tid>       PROC : <procName>
INSTANCE: <instance>        NODE : <node>      DB   : <database>
APPHDL  : <appHandle>       APPID: <appID>
AUTHID  : <authID>
EDUID   : <eduID>           EDUNAME: <engine dispatchable unit name>
FUNCTION: <prodName>, <compName>, <funcName>, probe:<probeNum>
MESSAGE : <messageID>  <msgText>
CALLED  : <prodName>, <compName>, <funcName>   OSERR: <errorName> (<errno>)
RETCODE : <type>=<retCode> <errorDesc>
ARG #N  : <typeTitle>, <typeName>, <size> bytes
... argument ...
DATA #N : <typeTitle>, <typeName>, <size> bytes
... data ...
```

예에서 아직 설명하지 않은 필드는 다음과 같습니다.

- <source>: 로그된 오류의 원점을 표시합니다(샘플 첫 번째 라인의 끝에서 찾을 수 있음). 사용할 수 있는 값은 다음과 같습니다.
  - origin - 메시지는 오류가 시작된 함수에 의해 로그됩니다(시작점).
  - OS - 오류가 운영 체제에 의해 생성되었습니다.
  - received - 다른 프로세스(클라이언트/서버)로부터 오류가 수신되었습니다.
  - sent - 다른 프로세스(클라이언트/서버)로 오류가 송신되었습니다.
- MESSAGE: 로그되는 메시지를 포함합니다. 다음으로 이루어져 있습니다.
  - <messageID> - 메시지 번호(예: ECF=0x9000004A 또는 DIA8604C)
  - <msgText> - 오류 설명

  CALLED 필드도 있는 경우, <msgText>는 FUNCTION 필드에 지정된 대로 메시지를 로깅하는 함수에서 CALLED 함수가 리턴한 오류의 영향입니다.
- CALLED: 오류를 리턴한 함수입니다. 다음으로 이루어져 있습니다.
  - <prodName> - 제품 이름: ″OS″, ″DB2″, ″DB2 Tools″ 또는 ″DB2 Common″
  - <compName> - 구성요소 이름(시스템 호출의 경우 '-')
  - <funcName> - 피호출 함수 이름
- OSERR: CALLED 시스템 호출이 리턴한 운영 체제 오류입니다(CALLED와 동일한 라인의 끝에서 찾을 수 있음). 다음으로 이루어져 있습니다.
  - <errorName> - 시스템 특정 오류 이름
  - <errno> - 운영 체제 오류 번호
- ARG: 이 섹션은 오류를 리턴한 함수 호출의 인수를 나열합니다. 다음으로 이루어져 있습니다.
  - <N> - ″피호출″ 함수 호출의 인수 위치
  - <typeTitle> - N번째 인수 typename과 연관된 레이블
  - <typeName> - 로그되는 인수 유형의 이름

  - <size> - 로그될 인수의 크기
- DATA: 로깅 함수가 덤프한 추가 데이터를 포함합니다. 다음으로 이루어져 있습니다.
  - <N> - 덤프되는 데이터 오브젝트의 순차 번호
  - <typeTitle> - 덤프되는 데이터의 레이블
  - <typeName> - 로그되는 데이터 필드 유형의 이름(예: PD_TYPE_UINT32, PD_TYPE_STRING)
  - <size> - 데이터 오브젝트의 크기

db2diag 로그 파일의 정보용 레코드 해석

db2diag 로그 파일의 첫 번째 메시지는 항상 정보용 레코드여야 합니다.

정보용 레코드 예는 다음과 같습니다.

```
2006-02-09-18.07.31.059000-300 I1H917            LEVEL: Event
PID     : 3140                 TID  : 2864       PROC : db2start.exe
INSTANCE: DB2                  NODE : 000
FUNCTION: DB2 UDB, RAS/PD component, _pdlogInt, probe:120
START   : New Diagnostic Log file
DATA #1 : Build Level, 124 bytes
Instance "DB2" uses "32" bits and DB2 code release "SQL09010"
with level identifier "01010107".
Informational tokens are "DB2 v9.1.0.190", "s060121", "", Fix Pack "0".
DATA #2 : System Info, 1564 bytes
System: WIN32_NT MYSRVR Service Pack 2 5.1 x86 Family 15, model 2, stepping 4
CPU: total:1 online:1 Cores per socket:1 Threading degree per core:1
Physical Memory(MB): total:1024 free:617 available:617
Virtual  Memory(MB): total:2462 free:2830
Swap     Memory(MB): total:1438 free:2213
Information in this record is only valid at the time when this file was created
(see this record's time stamp)
```

정보용 레코드는 모든 논리적 파티션에서 db2start의 출력입니다. 따라서 논리적 파티션 당 하나씩 여러 개의 정보용 레코드가 생성됩니다. 정보용 레코드에는 모든 파티션에서 다른 메모리 값이 포함되기 때문에 이 정보가 유용할 수 있습니다.

진단 로그 파일의 오류 캡처 레벨 설정

DB2 진단(db2diag) 로그 파일은 DB2에서 로그한 텍스트 정보를 포함하는 파일입니다. 이 정보는 문제점 해결에 사용되며 주로 IBM Software Support에서 사용하기 위한 것입니다.

db2diag 로그 파일에 기록되는 진단 오류 유형은 **diaglevel** 데이터베이스 관리 프로그램 구성 매개변수 설정에 의해 결정됩니다.

- 현재 설정을 점검하려면, GET DBM CFG 명령을 실행하십시오.

  다음 변수를 찾으십시오.

  ```
  Diagnostic error capture level              (DIAGLEVEL) = 3
  ```

- 값을 동적으로 변경하려면, UPDATE DBM CFG 명령을 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
```

예를 들어, 다음과 같습니다.

```
DB2 UPDATE DBM CFG USING DIAGLEVEL X
```

여기서 X는 원하는 통지 레벨입니다. 재현할 수 있는 문제점을 진단하고 있는 경우, IBM Software Support 담당자가 문제점 해결을 수행하는 동안 사용자가 **diaglevel** 4를 사용하도록 제안할 수 있습니다.

DB2 데이터베이스 및 OS 진단 조합

메모리, 스왑 파일, CPU, 디스크 스토리지 및 기타 자원과 관련된 일부 문제점을 진단하려면 주어진 운영 체제가 이러한 자원을 관리하는 방법을 완전히 이해해야 합니다. 최소한 자원 관련 문제점을 정의하려면 존재하는 해당 자원의 양, 사용자당 존재할지도 모르는 자원 한계를 알아야 합니다. (관련 한계는 일반적으로 DB2 인스턴스 소유자의 사용자 ID에 대한 것입니다.)

확보해야 하는 중요한 구성 정보 중 일부는 다음과 같습니다.

- 운영 체제 패치 레벨, 설치된 소프트웨어 및 업그레이드 실행기록
- CPU 수
- RAM 양
- 스왑 및 파일 캐시 설정
- 사용자 데이터 및 파일 자원 한계와 사용자당 프로세스 한계
- IPC 자원 한계(메시지 큐, 공유 메모리 세그먼트, 세마포어)
- 디스크 스토리지 유형
- 그 밖에 무엇이 시스템에 사용됩니까? DB2가 자원과 경쟁합니까?
- 어디서 인증이 발생합니까?

대부분의 플랫폼에는 자원 정보 검색을 위한 간단한 명령이 있습니다. 그러나 db2support 유틸리티가 이 데이터를 더 많이 수집하기 때문에 해당 정보를 수동으로 확보해야 하는 경우는 거의 없습니다. db2support(-s 및 -m 옵션이 지정된 경우)가 생성한 `detailed_system_info.html` 파일은 이 정보를 수집하는 데 사용되는 많은 운영 체제 명령에 대한 구문을 포함합니다.

다음 연습은 여러 DB2 진단 파일에서 시스템 구성 및 사용자 환경 정보를 찾는 데 도움을 주기 위한 것입니다. 첫 번째 연습은 db2support 유틸리티 실행과 관련된 단계에

친숙하게 합니다. 이어지는 연습에서는 사용자 환경 및 자원 한계를 이해하는 데 유용할 수 있는 DB2 생성 데이터를 제공하는 추가로 제공하는 트랩 파일을 다룹니다.

연습 1: db2support 명령 실행

1. db2start 명령을 사용하여 DB2 인스턴스를 시작하십시오.
2. 사용 가능한 SAMPLE 데이터베이스가 이미 있다고 가정하면 db2support의 출력을 저장하기 위한 디렉토리를 작성하십시오.
3. 해당 디렉토리로 변경하고 다음을 실행하십시오.

   ```
   db2support <directory> -d sample -s -m
   ```

4. 콘솔 출력, 특히 수집된 정보 유형을 검토하십시오.

   Windows에서 실행하는 경우 다음과 유사한 출력이 표시되어야 합니다.

   ```
   ...
   Collecting "System files"
        "db2cache.prf"
        "db2cos9402136.0"
        "db2cos9402840.0"
        "db2dbamr.prf"
        "db2diag.bak"
        "db2eventlog.000"
        "db2misc.prf"
        "db2nodes.cfg"
        "db2profile.bat"
        "db2systm"
        "db2tools.prf"
        "HealthRulesV82.reg"
        "db2dasdiag.log"
        ...
   Collecting "Detailed operating system and hardware information"
   Collecting "System resource info (disk, CPU, memory)"
   Collecting "Operating system and level"
   Collecting "JDK Level"
   Collecting "DB2 Release Info"
   Collecting "DB2 install path info"
   Collecting "Registry info"
   ...
   Creating final output archive
        "db2support.html"
        "db2_sqllib_directory.txt"
        "detailed_system_info.html"
        "db2supp_system.zip"
        "dbm_detailed.supp_cfg"
        "db2diag.log"
   db2support is now complete.
    An archive file has been produced: "db2support.zip"
   ```

5. 이제 웹 브라우저를 사용하여 detailed_system_info.html 파일을 보십시오. 각 시스템에서 다음 정보를 식별하십시오.
   - CPU 수
   - 운영 체제 레벨

- 사용자 환경
- 사용자 자원 한계(UNIX ulimit 명령)

연습 2: DB2 트랩 파일에서 환경 정보 찾기

1. DB2 인스턴스가 시작되는지 확인한 후 다음을 실행하십시오.

   ```
   db2pd -stack all
   ```

   호출 스택은 **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 정의한 대로 진단 디렉토리의 파일에 배치됩니다.
2. 트랩 파일 중 하나에서 다음을 찾으십시오.
   - DB2 코드 레벨
   - 맨 위 데이터 seg(이는 필요한 최대 개인용 주소 스페이스임)
   - Cur 데이터 *크기*(이는 최대 개인용 주소 스페이스 한계임)
   - Cur 코어 *크기*(이는 최대 코어 파일 한계임)
   - 신호 핸들러(이 정보가 모든 트랩 파일에는 나타나지 않을 수도 있음)
   - 환경 변수(이 정보가 모든 트랩 파일에는 나타나지 않을 수도 있음)
   - 맵 출력(로드된 라이브러리 표시)

Windows 트랩 파일 예(절단됨):

```
...
<DB2TrapFile version="1.0">
<Trap>
<Header>
DB2 build information: DB2 v9.1.0.190 s060121 SQL09010
timestamp: 2006-02-17-14.03.43.846000
uname: S:Windows
comment:
process id: 940
thread id: 3592
</Header>
<SystemInformation>
Number of Processors: 1
Processor Type: x86 Family 15 Model 2 Stepping 4
OS Version: Microsoft Windows XP, Service Pack 2 (5.1)
Current Build: 2600
</SystemInformation>
<MemoryInformation>
<Usage>
Physical Memory:    1023 total,     568 free.
Virtual Memory :    2047 total,    1882 free.
Paging File    :    2461 total,    2011 free.
Ext. Virtual   :       0 free.
</Usage>
</MemoryInformation>
<EnvironmentVariables>
<![CDATA[
[e] DB2PATH=C:₩Program Files₩IBM₩SQLLIB
[g] DB2_EXTSECURITY=YES
```

```
[g] DB2SYSTEM=MYSRVR
[g] DB2PATH=C:₩Program Files₩IBM₩SQLLIB
[g] DB2INSTDEF=DB2
[g] DB2ADMINSERVER=DB2DAS00
]]></EnvironmentVariables>
```

DB2 및 시스템 이벤트 또는 오류 상관

시스템 메시지 및 오류 로그는 무시되는 경우가 아주 많습니다. 문제점 정의 및 조사 초기 단계에서 하나의 단순 태스크를 수행하는 데 시간이 걸리는 경우 문제점을 해결하는 데 걸리는 시간을 몇 시간, 몇 일, 심지어 몇 주나 절약할 수 있습니다. 해당 태스크는 다른 로그의 항목을 비교하고 시간과 항목이 참조하는 자원과 시간 모두에서 관련된 것처럼 보이는 것을 적어두는 것입니다.

문제점 진단과 항상 관련되어 있는 것은 아니지만 많은 경우 시스템 로그에서 최상의 실마리를 즉시 사용할 수 있습니다. 보고된 시스템 문제점을 DB2 오류와 상관시킬 수 있는 경우, DB2 증상을 직접적으로 일으키는 문제점을 식별하는 경우가 많습니다. 명백한 예는 디스크 오류, 네트워크 오류 및 하드웨어 오류입니다. 그렇게 명백하지 않은 것은 다른 시스템에서 보고된 문제점(예: 연결 시간 또는 인증에 영향을 줄 수 있는 도메인 제어기)입니다.

특히, 아주 새로운 시스템에서 문제점이 보고되는 경우 안정성을 평가하기 위해 시스템 로그를 조사할 수 있습니다. 일반 응용프로그램에서 발생하는 중간 트랩은 기본 하드웨어 문제점이 있다는 징후일 수 있습니다.

시스템 로그가 제공하는 일부 기타 정보는 다음과 같습니다.

- 중요한 이벤트(예: 시스템을 재부트한 시기)
- 시스템에서 DB2 트랩(및 실패하는 기타 소프트웨어의 오류, 트랩 또는 예외) 연대기
- 커널 심각성, 파일 시스템 스페이스 부족 및 스왑 스페이스 부족 오류(시스템이 새 프로세스를 작성하거나 포크하지 못할 수 있음)

시스템 로그는 db2diag 로그 파일의 파손 항목을 근심의 원인으로 배제하는 데 도움을 줄 수 있습니다. 선행 오류가 없는 DB2 관리 통지 또는 DB2 진단 로그에서 파손 항목을 보게 되면, DB2 응급 복구가 시스템 종료의 결과일 가능성이 있습니다.

이 정보 상관 원칙은 모든 소스의 로그 및 식별 가능한 사용자 증상으로 확대됩니다. 예를 들어, 완벽히 이해할 수는 없더라도 다른 응용프로그램 로그의 상관 항목을 식별하고 설명하는 데 아주 유용할 수 있습니다.

이 정보 요약은 서버 및 문제점 당시에 발생하는 다양한 모든 이벤트에 대한 아주 완벽한 이해입니다.

db2cos(콜아웃 스크립트) 출력 파일

db2cos 스크립트는 데이터베이스 관리 프로그램이 심각한 상황, 트랩, 세그먼트 위반 또는 예외로 인해 처리를 계속할 수 없는 경우 디폴트로 호출됩니다. 각 디폴트 db2cos 스크립트는 db2pd 명령을 호출하여 래치되지 않은 방법으로 정보를 수집합니다.

db2cos 스크립트 이름은 db2cos_hang, db2cos_trap 등의 양식으로 되어 있습니다. 각 스크립트는 db2fodc 도구에서 호출되는 db2cos_hang을 제외하고 유사한 방법으로 작동합니다.

디폴트 db2cos 스크립트는 bin 디렉토리에 있습니다. UNIX 운영 체제에서 이 디렉토리는 읽기 전용입니다. db2cos 스크립트 파일을 adm 디렉토리로 복사하고 필요하면 해당 위치에서 파일을 수정할 수 있습니다. db2cos 스크립트가 adm 디렉토리에 있으면 실행됩니다. 그렇지 않으면, bin 디렉토리의 스크립트가 실행됩니다.

다중 파티션 구성에서 스크립트는 트랩을 발생시키는 파티션에서 트랩핑 에이전트에 대해서만 호출됩니다. 다른 파티션의 정보를 수집해야 하는 경우, db2cos 스크립트를 갱신하여 db2_all 명령을 사용하거나 모든 파티션이 동일한 시스템에 있는 경우에는 db2pd 명령에 -alldbpartitionnums 옵션을 지정하십시오.

db2cos 호출을 트리거하는 신호 유형도 db2pdcfg -cos 명령을 통해 구성할 수 있습니다. 디폴트 구성은 심각한 상황 또는 트랩이 발생할 때 db2cos 스크립트를 실행하는 것입니다. 그러나 생성된 신호는 디폴트로 db2cos 스크립트를 시작하지 않습니다.

심각한 상황, 트랩, 세그먼트 위반 또는 예외가 발생할 때 이벤트 순서는 다음과 같습니다.

1. 트랩 파일이 작성됨
2. 신호 핸들러가 호출됨
3. db2cos 스크립트가 호출됨(db2cos 설정 사용 가능 여부에 따라 다름)
4. 항목이 관리 통지 로그에 기록됨
5. 항목이 db2diag 로그 파일에 기록됨

db2cos 스크립트의 db2pd 명령이 수집한 디폴트 정보에는 운영 체제, 설치된 DB2 제품의 버전과 서비스 레벨, 데이터베이스 관리 프로그램 및 데이터베이스 구성은 물론 에이전트, 메모리 풀, 메모리 세트, 메모리 블록, 응용프로그램, 유틸리티, 트랜잭션, 버퍼 풀, 잠금, 트랜잭션 로그, 테이블 스페이스 및 컨테이너의 상태가 포함됩니다. 또한 동적, 정적 및 카탈로그 캐시, 테이블 및 인덱스 통계의 상태, 복구 상태는 물론 다시 최적화된 SQL문 및 활성 명령문 목록에 대한 정보도 제공합니다. 추가 정보를 수집해야 하는 경우, 추가 명령을 사용하여 db2cos 스크립트를 갱신하십시오.

디폴트 db2cos 스크립트가 호출되면, DIAGPATH 데이터베이스 관리 프로그램 구성 매개변수가 지정한 디렉토리에 출력 파일을 생성합니다. 파일 이름은

XXX.YYY.ZZZ.cos.txt입니다(여기서 XXX는 프로세스 ID(PID)이고 YYY는 스레드 ID(TID)이며, ZZZ는 데이터베이스 파티션 번호(또는 단일 파티션 데이터베이스의 경우 000)임). 다중 스레드가 트랩되는 경우, 각 스레드에 대해 db2cos 스크립트가 별도로 호출됩니다. PID 및 TID 조합이 두 번 이상 발생하는 경우, 데이터가 파일에 추가됩니다. 출력 반복을 구별할 수 있도록 시간소인이 있습니다.

db2cos 출력 파일에는 db2cos 스크립트에 지정된 명령에 따라 서로 다른 정보가 포함됩니다. 디폴트 스크립트가 변경되지 않으면, 다음과 유사한 항목이 표시됩니다(뒤에 자세한 db2pd 출력이 이어짐).

```
2005-10-14-10.56.21.523659
PID     : 782348               TID  : 1           PROC : db2cos
INSTANCE: db2inst1             NODE : 0           DB   : SAMPLE
APPHDL  :                      APPID: *LOCAL.db2inst1.051014155507
FUNCTION: oper system services, sqloEDUCodeTrapHandler, probe:999
EVENT   : Invoking /home/db2inst1/sqllib/bin/db2cos from
oper system services sqloEDUCodeTrapHandler
Trap Caught

Instance db2inst1 uses 64 bits and DB2 code release SQL09010
...
Operating System Information:

OSName:   AIX
NodeName: n1
Version:  5
Release:  2
Machine:  000966594C00

...
```

db2diag 로그 파일은 어커런스와 관련된 항목도 포함합니다. 예를 들어, 다음과 같습니다.

```
2005-10-14-10.42.17.149512-300 I19441A349         LEVEL: Event
PID     : 782348               TID  : 1           PROC : db2sysc
INSTANCE: db2inst1             NODE : 000
FUNCTION: DB2 UDB, trace services, pdInvokeCalloutScript, probe:10
START   : Invoking /home/db2inst1/sqllib/bin/db2cos from oper system
services sqloEDUCodeTrapHandler

2005-10-14-10.42.23.173872-300 I19791A310         LEVEL: Event
PID     : 782348               TID  : 1           PROC : db2sysc
INSTANCE: db2inst1             NODE : 000
FUNCTION: DB2 UDB, trace services, pdInvokeCalloutScript, probe:20
STOP    : Completed invoking /home/db2inst1/sqllib/bin/db2cos

2005-10-14-10.42.23.519227-300 E20102A509         LEVEL: Severe
PID     : 782348               TID  : 1           PROC : db2sysc
INSTANCE: db2inst1             NODE : 000
FUNCTION: DB2 UDB, oper system services, sqloEDUCodeTrapHandler, probe:10
MESSAGE : ADM0503C  An unexpected internal processing error has occurred.  ALL
          DB2 PROCESSES ASSOCIATED WITH THIS INSTANCE HAVE BEEN SHUTDOWN.
          Diagnostic information has been recorded.  Contact IBM Support for
          further assistance.

2005-10-14-10.42.23.520111-300 E20612A642         LEVEL: Severe
PID     : 782348               TID  : 1           PROC : db2sysc
INSTANCE: db2inst1             NODE : 000
```

```
FUNCTION: DB2 UDB, oper system services, sqloEDUCodeTrapHandler, probe:20
DATA #1 : Signal Number Recieved, 4 bytes
11
DATA #2 : Siginfo, 64 bytes
0x0FFFFFFFFFFFD5C0 : 0000 000B 0000 0000 0000 0009 0000 0000    ................
0x0FFFFFFFFFFFD5D0 : 0000 0000 0000 0000 0000 0000 0000 0000    ................
0x0FFFFFFFFFFFD5E0 : 0000 0000 0000 0000 0000 0000 0000 0000    ................
0x0FFFFFFFFFFFD5F0 : 0000 0000 0000 0000 0000 0000 0000 0000    ................
```

덤프 파일

덤프 파일은 문제점 진단 시 유용한 추가 정보(예: 내부 제어 블록)가 있는 오류가 발생할 때 작성됩니다. 덤프 파일에 기록된 모든 데이터 항목에는 문제점 판별을 돕기 위해 연관된 시간소인이 있습니다. 덤프 파일은 2진 형식으로 되어 있으며 IBM Software Support 담당자가 사용하도록 하기 위한 것입니다.

덤프 파일이 작성되거나 추가되면, db2diag 로그 파일에 기록된 데이터 유형과 시간을 표시하는 항목이 작성됩니다. 이러한 db2diag 로그 항목은 다음과 유사합니다.

```
2007-05-18-12.28.11.277956-240 I24861950A192 LEVEL: Severe
PID:1056930 TID:225448 NODE:000 Title: dynamic memory buffer
Dump File:/home/svtdbm5/sqllib/db2dump/1056930.225448.000.dump.bin
```

주: 파티션된 데이터베이스 환경의 경우, 파일 확장자는 파티션 번호를 식별합니다. 예를 들어, 다음 항목은 파티션 10에서 실행 중인 DB2 프로세스가 덤프 파일을 작성했음을 표시합니다.

```
Dump File: /home/db2/sqllib/db2dump/6881492.2.010.dump.bin
```

FODC(First Occurrence Data Capture) 정보

FODC(First Occurrence Data Capture)는 DB2 인스턴스에 대한 시나리오 기반 데이터를 캡처하는 데 사용되는 프로세스입니다. 특정 증상을 기반으로 DB2 사용자가 수동으로 FODC를 호출하거나 사전 판별된 시나리오 또는 증상이 발견된 경우 자동으로 호출할 수 있습니다. 이 정보를 사용할 경우, 진단 정보를 얻기 위해 오류를 재생해야 하는 필요성이 줄어듭니다.

FODC 정보는 다음 파일에 있습니다.

관리 통지 로그(*instance_name*.nfy)

- 운영 체제: 모두
- 디폴트 위치:
  - Linux 및 UNIX: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 지정한 디렉토리에 있습니다.
  - Windows: 이벤트 뷰어 도구 사용(**시작 > 제어판 > 관리 도구 > 이벤트 뷰어**)
- 인스턴스가 작성될 때 자동으로 작성됩니다.

- 중요한 이벤트가 발생하면, DB2가 관리 통지 로그에 정보를 기록합니다. 이 정보는 데이터베이스 및 시스템 관리자용입니다. 이 파일에 기록된 메시지 유형은 **notifylevel** 구성 매개변수로 판별됩니다.

  주: diagsize 데이터베이스 관리 프로그램 구성 매개변수가 0이 아닌 값으로 설정된 경우, 단일 관리 통지 로그 파일 동작(*instance_name*.nfy)은 회전 로그 동작(*instance_name.N*.nfy)으로 변경됩니다.

DB2 진단 로그(db2diag.log)

- 운영 체제: 모두
- 디폴트 위치: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 식별한 디렉토리에 있습니다.
- 인스턴스가 작성될 때 자동으로 작성됩니다.
- 이 텍스트 파일은 인스턴스에 발생한 오류 및 경고에 대한 진단 정보를 포함합니다. 이 정보는 문제점 해결에 사용되며 IBM Software Support의 기술자를 대상으로 합니다. 이 파일에 기록된 메시지 유형은 **diaglevel** 데이터베이스 관리 프로그램 구성 매개변수로 판별됩니다.

  주: diagsize 데이터베이스 관리 프로그램 구성 매개변수가 0이 아닌 값으로 설정된 경우, 단일 진단 로그 파일 동작(db2diag.log)은 회전 로그 동작(db2diag.*N*.log)으로 변경됩니다.

DB2 Administration Server(DAS) 진단 로그(db2dasdiag.log)

- 운영 체제: 모두
- 디폴트 위치:
  - Linux 및 UNIX: DASHOME/das/dump에 있습니다(여기서 DASHOME은 DAS 소유자의 홈 디렉토리임).
  - Windows: DAS 홈 디렉토리의 "dump" 폴더에 있습니다(예: C:¥Program Files¥IBM¥SQLLIB¥DB2DAS00¥dump).
- DAS가 작성될 때 자동으로 작성됩니다.
- 이 텍스트 파일은 DAS에 발생한 오류 및 경고에 대한 진단 정보를 포함합니다.

DB2 이벤트 로그(db2eventlog.xxx, 여기서 xxx는 데이터베이스 파티션 번호임)

- 운영 체제: 모두
- 디폴트 위치: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 지정한 디렉토리에 있습니다.
- 인스턴스가 작성될 때 자동으로 작성됩니다.
- DB2 이벤트 로그 파일은 데이터베이스 관리 프로그램에서 발생하는 인프라스트럭처 레벨 이벤트의 순환 로그입니다. 파일은 크기가 정해져 있으며, 인

스턴스가 실행될 때 로그되는 특정 이벤트에 대한 순환 버퍼의 역할을 합니다. 인스턴스를 중지할 때마다 이전 이벤트 로그가 추가되지 않고 교체됩니다. 인스턴스가 트랩되면, db2eventlog.XXX.crash 파일도 생성됩니다. 이러한 파일은 IBM Software Support에서 사용하도록 하기 위한 것입니다.

DB2 콜아웃 스크립트(db2cos) 출력 파일

- 운영 체제: 모두
- 디폴트 위치: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 지정한 디렉토리에 있습니다.
- db2cos 스크립트가 FODC 중단의 결과로 실행되는 경우, db2cos 출력 파일은 **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 지정한 위치에 작성된 FODC 디렉토리에 배치됩니다.
- 심각한 상황, 트랩 또는 세그먼트 위반이 발생하면 자동으로 작성됩니다. 또한 db2pdcfg 명령을 사용하여 지정한 대로 특정 문제점 시나리오 중에 작성될 수도 있습니다.
- 디폴트 db2cos 스크립트는 db2pd 명령을 호출하여 래치되지 않은 방법으로 정보를 수집합니다. db2cos 출력 파일의 컨텐츠는 db2cos 스크립트에 포함된 명령(예: 운영 체제 명령 및 기타 DB2 진단 도구)에 따라 다릅니다. db2cos 스크립트를 사용하여 실행되는 도구에 대한 자세한 내용을 보려면 텍스트 편집기에서 스크립트 파일을 여십시오.
- db2cos 스크립트는 `bin/` 디렉토리에 제공됩니다. UNIX인 경우, 이 디렉토리는 읽기 전용입니다. 이 스크립트의 자체 수정 가능한 버전을 작성하려면, db2cos 스크립트를 `adm/` 디렉토리에 복사하십시오. 이 버전의 스크립트를 얼마든지 수정할 수 있습니다. 스크립트가 `adm/` 디렉토리에 있으면, 해당 버전이 실행됩니다.그렇지 않으면, `bin/` 디렉토리의 디폴트 버전이 실행됩니다.

덤프 파일

- 운영 체제: 모두
- 디폴트 위치: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 지정한 디렉토리에 있습니다.
- 이러한 파일이 FODC 중단 중에 덤프되는 경우, 파일은 FODC 디렉토리에 배치됩니다.
- 특정 문제점 시나리오가 발생할 때 자동으로 작성됩니다.
- 일부 오류 조건의 경우, 실패한 프로세스 ID의 이름을 딴 2진 파일에 추가 정보가 로그됩니다. 이러한 파일은 IBM Software Support에서 사용하도록 하기 위한 것입니다.

트랩 파일

- 운영 체제: 모두

- 디폴트 위치: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 지정한 디렉토리에 있습니다.
- 이러한 파일이 FODC 중단 중에 덤프되는 경우, 파일은 FODC 디렉토리에 배치됩니다.
- 인스턴스가 비정상적으로 종료될 때 자동으로 작성됩니다. 또한 db2pd 명령을 사용하여 작성할 수 있습니다.
- 데이터베이스 관리 프로그램은 트랩, 세그먼트 위반 또는 예외로 인해 처리를 계속할 수 없는 경우 트랩 파일을 생성합니다.

코어 파일

- 운영 체제: Linux 및 UNIX
- 디폴트 위치: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 지정한 디렉토리에 있습니다.
- 이러한 파일이 FODC 중단 중에 덤프되는 경우, 파일은 FODC 디렉토리에 배치됩니다.
- DB2 인스턴스가 비정상적으로 종료되는 경우 운영 체제에 의해 작성됩니다.
- 그 중에서 코어 이미지는 문제점 설명에 필요할 수도 있는 DB2 메모리 할당을 대부분 또는 모두 포함합니다.

일반적인 중단 문제점을 기반으로 진단 정보 수집

인스턴스에 영향을 주는 중단이 발생하는 경우 패키지에 자동으로 진단 정보를 수집할 수 있습니다. 또한 패키지의 정보를 수동으로 작성할 수 있습니다.

DB2 인스턴스 및 데이터베이스에 대한 작업을 할 때 문제점이 발생하면, 문제점이 발생한 시간에 데이터를 수집해야 합니다. FODC(First Occurrence Data Collection)는 DB2 환경에서 문제점이 발생할 때 취한 조치를 설명하는 데 사용되는 용어입니다. db2pdcfg 도구를 사용하여 DB2FODC 레지스트리 변수 옵션 설정을 통해 중단 중에 수집되는 데이터를 제어할 수 있습니다. DB2FODC 레지스트리 변수 옵션을 변경하려면 db2pdcfg -fodc를 사용하십시오. 옵션은 FODC 상황의 데이터 캡처와 관련하여 데이터베이스 시스템 동작에 영향을 줍니다.

진단 정보 자동 콜렉션

데이터베이스 관리 프로그램은 자동 FODC(First Occurrence Data Capture)를 위해 db2fodc 명령을 호출합니다.

중단을 DB2 진단 로그 및 기타 문제점 해결 파일과 상관시키기 위해 진단 메시지가 관리 통지 파일과 db2diag 로그 파일 둘 다에 기록됩니다. 진단 메시지에는 FODC 디렉토리가 작성된 당시의 시간소인과 FODC 디렉토리 이름이 포함됩니다. FODC 패키지 설명 파일은 새 FODC 디렉토리에 배치됩니다.

표 83. 자동 FODC 유형 및 패키지

패키지	설명	호출 유형	실행되는 스크립트
FODC_Trap_*timestamp*	인스턴스 와이드 트랩이 발생했습니다.	자동	db2cos_trap(.bat)
FODC_Panic_*timestamp*	엔진이 모순을 발견하고 계속하지 않기로 결정했습니다.	자동	db2cos_trap(.bat)
FODC_BadPage_ *timestamp*	잘못된 페이지가 발견되었습니다.	자동	db2cos_datacorruption(.bat)
FODC_DBMarkedBad_ *timestamp*	오류로 인해 데이터베이스가 잘못된 것으로 표시되었습니다.	자동	db2cos(.bat)
FODC_IndexError_ *timestamp*_PID_EDUID_ Node#	EDU 와이드 인덱스 오류가 발생했습니다 (db2cos_indexerror_short(.bat) 및 / 또는 db2cos_indexerror_long(.bat)이 디렉토리에 덤프됨).	자동	N/A

진단 정보 수동 콜렉션

FODC(First Occurrence Data Collection) 명령(db2fodc)은 잠재적 정지에 대한 정보를 수집하는 데 사용되거나 심각한 성능 문제점이 있는 경우 사용됩니다. db2fodc 명령이 실행되면, 현재 진단 경로 아래에 새 디렉토리(`FODC_hang_timestamp`)가 자동으로 작성됩니다. db2cos_hang 스크립트가 실행됩니다. 이 스크립트는 수집되어 `FODC` 서브디렉토리에 배치될 데이터 콜렉션을 제어합니다. `FODC` 서브디렉토리의 존재는 db2fodc 명령이 실행되는 방법 또는 DB2 레지스트리 변수 구성에 따라 다릅니다.

표 84. 수동 FODC 유형 및 패키지

패키지	설명	호출 유형	실행되는 스크립트
FODC_Hang_ *timestamp*	사용자가 정지 문제점 해결(또는 심각한 성능)에 대한 데이터를 수집하기 위해 db2fodc -hang을 호출했습니다.	수동	db2cos_hang(.bat)
FODC_Perf_ *timestamp*	사용자가 성능 문제점 해결에 대한 데이터를 수집하기 위해 db2fodc -perf를 호출했습니다.	수동	db2cos_perf(.bat)

표 84. 수동 FODC 유형 및 패키지 (계속)

패키지	설명	호출 유형	실행되는 스크립트
FODC_IndexError_*timestamp*_PID_EDUID_Node#에 위치한 스크립트	사용자는 db2fodc -indexerror *FODC_IndexError_directory* [basic | full](디폴트는 basic임)을 호출하여 스크립트의 db2dart 명령을 호출할 수 있습니다. DPF에서는 db2_all "<<+node#< db2fodc -indexerror *FODC_IndexError_directory* [basic | full]"을 사용하십시오. node#는 *FODC_IndexError_directory* 디렉토리 이름의 마지막 번호입니다. db2fodc -indexerror를 db2_all 명령과 함께 사용하는 경우에는 절대 경로가 필요합니다.	수동	db2cos_indexerror_long(.bat), db2cos_indexerror_short(.bat)

진단 정보 자동 콜렉션 구성

취할 조치를 데이터베이스 관리 프로그램에 표시해야 데이터베이스 관리 프로그램이 조치를 자동으로 수행할 수 있습니다.

데이터베이스 조작 중에 오류 또는 경고가 발생하면 데이터베이스 관리 프로그램이 취할 조치를 표시하는 플래그가 설정됩니다. 수행되는 조치는 다음과 같습니다.

- db2diag 로그 파일에 스택 추적 생성(디폴트)
- 콜아웃 스크립트(db2cos) 실행(디폴트)
- 추적(db2trc) 명령 중지

FODC(First Occurrence Data Capture) 옵션 변경

문제점 판별 동작을 위한 DB2 데이터베이스 구성(db2pdcfg) 명령을 사용하여 FODC(First Occurrence Data Capture) 옵션을 변경하십시오. FODC 옵션은 db2pdcfg 도구를 사용하여 DB2FODC 레지스트리 변수에 설정됩니다. 옵션은 FODC 상황의 데이터 캡처와 관련하여 데이터베이스 시스템 동작에 영향을 줍니다.

FODC의 일부로 수집된 데이터 및 배치

인스턴스 내 중단 유형에 따라, FODC(First Occurrence Data Capture)의 결과 서브디렉토리가 작성되고 특정 컨텐츠가 수집됩니다. 파일 및 로그 콜렉션과 함께 일련의 서브디렉토리가 작성됩니다.

하나 이상의 다음 서브디렉토리가 FODC 디렉토리에 작성됩니다.

- DB2 구성 출력 및 파일을 포함하는 DB2CONFIG
- db2pd 출력 또는 출력 파일을 포함하는 DB2PD

- DB2 스냅샷을 포함하는 DB2SNAPS
- DB2 추적을 포함하는 DB2TRACE
- 운영 체제 구성 파일을 포함하는 OSCONFIG
- 운영 체제 모니터 정보를 포함하는 OSSNAPS
- 운영 체제 추적을 포함하는 OSTRACE

DB2FODC 구성 또는 db2fodc가 실행되는 모드에 따라 이러한 디렉토리가 항상 존재하지는 않을 수도 있습니다.

중단 유형에 따라 FODC 디렉토리 및 서브디렉토리에 다음 컨텐츠가 있습니다.

- 트랩 파일
- 중단의 일부로 데이터 캡처 중에 생성되고 다른 구성요소에 의해 완료된 모든 다른 2진 및 일반 텍스트 덤프 파일
- db2evlog의 이벤트 로그 파일
- 중단 시점에 추적이 설정되었던 경우 DB2 추적 덤프
- 코어 파일을 포함하는 디렉토리
- DB2FODC 로그 파일:
  - 하나의 "log" 파일만 수동 FODC. db2fodc_hang.log(정지의 경우) 또는 db2fodc_badpage.log(잘못된 페이지의 경우)에 사용됩니다.
- 데이터 손상 관련 정보
  - 프로세스 정보: ps(UNIX의 경우) 및 db2pd -edus 출력
  - db2support가 현재 수집한 추가 정보(선택사항):
    - errpt -a 출력(AIX의 경우)
    - UNIX 플랫폼의 경우 시스템 로그. 예를 들어, SunOS의 경우 `/var/adm/messages`, HP/UX의 경우 `/var/adm/syslog.log`입니다. 이는 이러한 파일을 수집할 수 있으면 완료됩니다(Linux에서는 루트 액세스 권한이 있어야 syslog 파일을 복사할 수 있음).

자동 FODC 데이터 생성

중단이 발생하고 자동 FODC(First Occurrence Data Capture)를 사용할 수 있는 경우, 증상을 기반으로 데이터가 수집됩니다. 수집되는 데이터는 중단을 진단하는 데 필요한 사항에 특정합니다.

"critical"로 정의된 메시지를 포함하여 하나의 메시지 또는 많은 메시지가 중단 원인을 표시하는 데 사용됩니다.

트랩 파일에 포함된 내용은 다음과 같습니다.

- 여유 가상 스토리지의 양
- 트랩이 발생한 당시에 제품의 구성 매개변수 및 레지스트리 변수와 연관된 값

- 트랩 시점에 DB2 제품이 사용한 추정 메모리 양
- 중단에 대한 컨텍스트를 제공하는 정보

원시 스택 덤프가 ASCII 트랩 파일에 포함될 수 있습니다.

데이터베이스 관리 프로그램 내 구성요소에 특정한 덤프 파일은 해당 FODC 패키지 디렉토리에 저장됩니다.

DB2 Query Patroller 및 FODC(First Occurrence Data Capture)

DB2 Query Patroller 문제점을 조사해야 한다고 판단될 경우, 겪고 있는 어려움의 가능한 원인에 대한 정보가 포함된 로그가 있습니다.

qpdiag.log

- 운영 체제: 모두
- 디폴트 위치: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 식별한 디렉토리에 있습니다.
- Query Patroller 시스템이 활성화될 때 자동으로 작성됩니다.
- Query Patroller에 대한 정보용 및 진단 레코드를 포함합니다. 이 정보는 문제점 해결에 사용되며 IBM Software Support에서 사용하도록 하기 위한 것입니다.

qpmigrate.log

- 운영 체제: 모두
- 디폴트 위치: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 식별한 디렉토리에 있습니다.
- qpmigrate 유틸리티에 의해 자동으로 작성됩니다. Query Patroller를 설치할 때 내재적으로(Query Patroller를 실행하기 위해 기존 데이터베이스를 지정하는 경우) 또는 설치 후 명시적으로 qpmigrate 명령을 실행할 수 있습니다.
- 한 버전에서 다른 버전으로 Query Patroller가 이주되는 경우 정보 및 오류 메시지를 캡처합니다. Query Patroller 관리자가 사용하도록 하기 위한 것입니다.

qpsetup.log

- 운영 체제: 모두
- 디폴트 위치: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 식별한 디렉토리에 있습니다.
- qpsetup 유틸리티에 의해 자동으로 작성됩니다. Query Patroller를 설치할 때 내재적으로(Query Patroller를 실행하기 위해 기존 데이터베이스를 지정하는 경우) 또는 설치 후 명시적으로 qpsetup 명령을 실행할 수 있습니다.

- qpsetup 유틸리티가 실행 중에 발생하는 정보 및 오류 메시지를 캡처합니다. Query Patroller 관리자가 사용하도록 하기 위한 것입니다.

qpuser.log

- 운영 체제: 모두
- 디폴트 위치: **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 식별한 디렉토리에 있습니다.
- Query Patroller 시스템이 활성화될 때 자동으로 작성됩니다.
- Query Patroller에 대한 정보 메시지(예: Query Patroller 중지 및 시작 시기를 표시하는 메시지)를 표시합니다. Query Patroller 관리자가 사용하도록 하기 위한 것입니다.

FODC(First Occurrence Data Capture)를 사용하는 모니터 및 감사 기능

모니터 또는 감사 기능 문제점을 조사해야 한다고 판단될 경우, 겪고 있는 어려움의 가능한 원인에 대한 정보가 포함된 로그가 있습니다.

DB2 감사 로그("db2audit.log")

- 운영 체제: 모두
- 디폴트 위치:
  - Windows: $DB2PATH¥*instance_name*¥security 디렉토리에 있습니다.
  - Linux 및 UNIX: $HOME¥sqllib¥security 디렉토리(여기서 $HOME은 인스턴스 소유자의 홈 디렉토리임)에 있습니다.
- db2audit 기능이 시작될 때 작성됩니다.
- 사전 정의된 일련의 데이터베이스 이벤트에 대해 DB2 감사 기능을 통해 생성된 감사 레코드를 포함합니다.

DB2 조정자 로그("*mylog.x*", 여기서 *x*는 조정자(governor)가 실행 중인 데이터베이스 파티션 번호임)

- 운영 체제: 모두
- 디폴트 위치:
  - Windows: $DB2PATH¥*instance_name*¥log 디렉토리에 있습니다.
  - Linux 및 UNIX: $HOME¥sqllib¥log 디렉토리(여기서 $HOME은 인스턴스 소유자의 홈 디렉토리임)에 있습니다.
- 조정자 유틸리티 사용 시 작성됩니다. 로그 파일 이름의 기본은 db2gov 명령에 지정됩니다.
- 조정자(governor) 디먼이 수행한 조치(예: 응용프로그램 강제 실행, 조정자 구성 파일 읽기, 유틸리티 시작 또는 종료)에 대한 정보는 물론 오류 및 경고도 기록합니다.

이벤트 모니터 파일(예: "00000000.evt")

- 운영 체제: 모두
- 디폴트 위치: 파일 이벤트 모니터를 작성할 때 모든 이벤트 레코드는 CREATE EVENT MONITOR문에 지정된 디렉토리에 기록됩니다.
- 이벤트가 발생할 때 이벤트에 의해 생성됩니다.
- 이벤트 모니터와 연관된 이벤트 레코드를 포함합니다.

FODC(First Occurrence Data Capture)를 사용하는 그래픽 도구

명령 편집기, Data Warehouse Center 또는 정보 카탈로그 센터 문제점을 조사해야 한다고 판단될 경우, 겪고 있는 어려움의 가능한 원인에 대한 정보가 포함된 로그가 있습니다.

명령 편집기 로그

- 운영 체제: 모두
- 디폴트 위치: 이 로그 파일의 이름과 위치는 DB2 도구 모음의 명령 편집기 페이지를 사용하여 지정됩니다. 경로를 지정하지 않은 경우, 로그는 Windows에서는 `$DB2PATH₩sqllib₩tools` 디렉토리, Linux 및 UNIX에서는 `$HOME/sqllib/tools` 디렉토리(여기서 HOME은 인스턴스 소유자의 홈 디렉토리임)에 저장됩니다.
- 명령 편집기에서 **명령 실행기록을 파일에 로그**를 선택한 후 파일 및 위치를 지정하면 작성됩니다.
- 명령 편집기로부터의 명령 및 명령문 실행기록을 포함합니다.

Data Warehouse Center IWH2LOGC.log 파일

- 운영 체제: 모두
- 디폴트 위치: VWS_LOGGING 환경 변수로 지정된 디렉토리에 있습니다. 디폴트 경로는 Windows에서는 `$DB2PATH₩sqllib₩logging` 디렉토리, Linux 및 UNIX에서는 `$HOME/sqllib/logging` 디렉토리(여기서 HOME은 인스턴스 소유자의 홈 디렉토리임)입니다.
- 로그 프로그램이 중지되면 Data Warehouse Center에서 자동으로 작성합니다.
- 로그 프로그램이 중지된 상황에서 전송할 수 없는 메시지(Data Warehouse Center 및 OLE 서버에 의해 기록됨)를 포함합니다. Data Warehouse Center의 로그 보기 프로그램 창을 사용하여 이 로그를 볼 수 있습니다.

Data Warehouse Center IWH2LOG.log 파일

- 운영 체제: 모두

- 디폴트 위치: VWS_LOGGING 환경 변수로 지정된 디렉토리에 있습니다. 디폴트 경로는 Windows에서는 $DB2PATH₩sqllib₩logging 디렉토리, Linux 및 UNIX에서는 $HOME/sqllib/logging 디렉토리(여기서 HOME은 인스턴스 소유자의 홈 디렉토리임)입니다.
- 자체적으로 시작할 수 없거나 추적이 활성화되면 Data Warehouse Center에서 자동으로 작성합니다.
- Data Warehouse Center 로그 프로그램이 자체적으로 시작될 수 없고 Data Warehouse Center 로그(IWH2LOGC.log)에 기록할 수 없는 상황에 대한 진단 정보를 포함합니다. Data Warehouse Center의 로그 보기 프로그램 창을 사용하여 이 로그를 볼 수 있습니다.

Data Warehouse Center IWH2SERV.log 파일

- 운영 체제: 모두
- 디폴트 위치: VWS_LOGGING 환경 변수로 지정된 디렉토리에 있습니다. 디폴트 경로는 Windows에서는 $DB2PATH₩sqllib₩logging 디렉토리, Linux 및 UNIX에서는 $HOME/sqllib/logging 디렉토리(여기서 HOME은 인스턴스 소유자의 홈 디렉토리임)입니다.
- Data Warehouse Center 서버 추적 기능을 통해 자동으로 작성됩니다.
- Data Warehouse Center 시작 메시지를 포함하며 서버 추적 기능을 통해 작성된 메시지를 로그합니다. Data Warehouse Center의 로그 보기 프로그램 창을 사용하여 이 로그를 볼 수 있습니다.

정보 카탈로그 센터 태그 파일 EXPORT 로그

- 운영 체제: 모두
- 디폴트 위치: 익스포트된 태그 파일 경로와 로그 파일 이름은 정보 카탈로그 센터의 익스포트 도구의 **옵션** 탭에 지정됩니다.
- 정보 카탈로그 센터의 익스포트 도구를 통해 생성됩니다.
- 태그 파일 익스포트 정보(예: 익스포트 프로세스가 시작 및 중지된 시간과 날짜)를 포함합니다. 익스포트 조작 중에 발생한 오류 메시지도 포함합니다.

정보 카탈로그 센터 태그 파일 IMPORT 로그

- 운영 체제: 모두
- 디폴트 위치: 임포트된 태그 파일 경로와 로그 파일 이름은 정보 카탈로그 센터의 임포트 도구에 지정됩니다.
- 정보 카탈로그 센터의 임포트 도구를 통해 생성됩니다.
- 태그 파일 임포트 실행기록 정보(예: 임포트 프로세스가 시작 및 중지된 시간과 날짜)를 포함합니다. 임포트 조작 중에 발생한 오류 메시지도 포함합니다.

내부 리턴 코드

두 가지 유형의 내부 리턴 코드(ZRC 값 및 ECF 값)가 있습니다. 이는 일반적으로 IBM Software Support에서 사용하기 위한 진단 도구에서만 볼 수 있는 리턴 코드입니다.

예를 들면, 내부 리턴 코드는 DB2 추적 결과 및 db2diag 로그 파일에 표시됩니다.

ZRC 및 ECF 값은 기본적으로 동일한 용도로 사용되지만 형식이 약간 다릅니다. 각 ZRC 값은 다음 특성을 갖습니다.

- 클래스 이름
- 구성요소
- 이유 코드
- 연관된 SQLCODE
- SQLCA 메시지 토큰
- 설명

그러나 ECF 값은 다음으로 이루어집니다.

- 세트 이름
- 제품 ID
- 구성요소
- 설명

ZRC 및 ECF 값은 일반적으로 음수이며 오류 조건을 나타내는 데 사용됩니다. ZRC 값은 값이 나타내는 오류 유형에 따라 그룹화됩니다. 이러한 그룹화를 ″클래스″라고 합니다. 예를 들어, 이름이 『SQLZ_RC_MEMHEP』로 시작하는 ZRC 값은 일반적으로 메모리 부족과 관련된 오류입니다. ECF 값은 마찬가지로 ″세트″로 그룹화됩니다.

ZRC 값을 포함하는 db2diag 로그 파일 항목 예는 다음과 같습니다.

```
2006-02-13-14.34.35.965000-300    I17502H435       LEVEL: Error
PID     : 940                TID  : 660    PROC : db2syscs.exe
INSTANCE: DB2                NODE : 000     DB   : SAMPLE
APPHDL  : 0-1433                APPID: *LOCAL.DB2.050120082811
FUNCTION: DB2 UDB, data protection, sqlpsize, probe:20
RETCODE : ZRC=0x860F000A=-2045837302=SQLO_FNEX "File not found."
          DIA8411C A file "" could not be found.
```

예를 들어, db2diag 명령을 사용하여 이 ZRC 값에 대한 전체 세부사항을 확보할 수 있습니다.

```
c:₩>db2diag -rc 0x860F000A

Input ZRC string '0x860F000A' parsed as 0x860F000A (-2045837302).

ZRC value to map: 0x860F000A (-2045837302)
        V7 Equivalent ZRC value: 0xFFFFE60A (-6646)
```

```
ZRC class :
        Critical Media Error (Class Index: 6)
Component:
        SQLO ; oper system services (Component Index: 15)
Reason Code:
        10 (0x000A)

Identifer:
        SQLO_FNEX
        SQLO_MOD_NOT_FOUND
Identifer (without component):
        SQLZ_RC_FNEX

Description:
        File not found.

Associated information:
        Sqlcode -980
SQL0980C  A disk error occurred.  Subsequent SQL statements cannot be
processed.

        Number of sqlca tokens : 0
        Diaglog message number: 8411
```

db2diag -rc -2045837302 또는 db2diag -rc SQLO_FNEX 명령을 실행하면 동일한 정보가 리턴됩니다.

ECF 리턴 코드에 대한 출력 예는 다음과 같습니다.

```
c:₩>db2diag -rc 0x90000076

Input ECF string '0x90000076' parsed as 0x90000076 (-1879048074).

ECF value to map: 0x90000076 (-1879048074)

ECF Set :
        setecf (Set index : 1)
Product :
        DB2 Common
Component:
        OSSe
Code:
        118 (0x0076)

Identifier:
        ECF_LIB_CANNOT_LOAD

Description:
        Cannot load the specified library
```

db2diag 명령 출력에서 가장 귀중한 문제점 해결 정보는 설명 및 연관된 정보(ZRC 리턴 코드의 경우만)입니다.

ZRC 또는 ECF 값 전체 목록을 보려면, 각각 db2diag -rc zrc 및 db2diag -rc ecf를 사용하십시오.

메시지 개요

사용자가 DB2가 설치된 운영 체제의 기능에 익숙하다고 가정합니다. 이 주제 항목에 있는 정보를 사용하여 오류나 문제점을 식별하고 적절한 복구 조치를 사용하여 문제점을 해결할 수 있습니다. 이 정보는 또한 메시지가 생성되고 기록되는 위치를 이해하는 데에도 사용할 수 있습니다.

메시지 구조

메시지 도움말은 메시지의 원인과 메시지에 응답할 때 취해야 조치에 대해 설명합니다.

메시지 ID는 세 개의 문자 메시지 접두부와 그 뒤의 네 개 또는 다섯 개의 숫자 메시지 번호, 그리고 그 뒤의 한 개의 문자 접미부로 구성됩니다. 예를 들어 *SQL1042C*입니다. 메시지 접두부 목록은 606 페이지의 『메시지 도움말 호출』 및 607 페이지의 『기타 DB2 메시지』의 내용을 참조하십시오. 한 개의 문자 접미부는 오류 메시지의 심각도를 나타냅니다.

일반적으로 메시지 ID가 *C*로 끝나면 심각한 메시지, *E*로 끝나면 긴급 메시지, *N*으로 끝나면 오류 메시지, *W*로 끝나면 경고 메시지 그리고 *I*로 끝나면 정보 메시지를 나타냅니다.

ADM 메시지 ID는 *C*로 끝나면 심각한 메시지, *E*로 끝나면 긴급 메시지, *W*로 끝나면 중요 메시지 그리고 *I*로 끝나면 정보 메시지를 나타냅니다.

SQL 메시지 ID는 *C*로 끝나면 심각한 시스템 오류, *N*로 끝나면 오류 메시지 그리고 *W*로 끝나면 경고 또는 정보 메시지를 나타냅니다.

일부 메시지에는 메시지 변수라고도 불리는 토큰이 포함되어 있습니다. DB2에서 토큰이 포함된 메시지가 생성되면 사용자가 오류 메시지의 원인을 진단할 수 있도록 각각의 토큰이 발생한 오류 조건에 대한 특정 값으로 교체됩니다. 예를 들어 DB2 메시지 SQL0107N 은 다음과 같습니다.

- 명령행 처리기에서 오류 발생 시:

  SQL0107N 이름 ″<name>″이 너무 깁니다. 최대 길이는 ″<length>″입니다.

- DB2 정보 센터에서 오류 발생 시:

  SQL0107N 이름 *name*이 너무 깁니다. 최대 길이는 *length*입니다.

이 메시지에는 두 개의 토큰 ″<name>″과 ″<length>″가 포함되어 있습니다. 런타임에 이 메시지가 생성되면 메시지 토큰이 오류를 일으킨 오브젝트의 실제 이름과 오브젝트 유형에 허용된 최대 길이로 각각 교체됩니다.

다음 예와 같이 토큰을 특정 오류 인스턴스에 적용할 수 없는 경우 *N 값이 대신 리턴됩니다.

```
SQL20416N 제공된 값("*N")을 보안 레이블로 변환 할 수
없습니다. 정책 ID가 1인 보안 정책의 레이블은 "8"자가
되어야 합니다. "0" 문자의 값이 깁니다. SQLSTATE=23523
```

메시지 도움말 호출

다음 DB2 메시지를 명령행 처리기에서 액세스할 수 있습니다.

접두부 설명

ADM 여러 DB2 구성요소에 의해 생성된 메시지. 이 메시지는 관리 통지 로그 파일에 작성되며 시스템 관리자에게 추가 정보를 제공하기 위한 것입니다.

AMI MQ 응용프로그램 메시지 인터페이스에 의해 생성된 메시지

ASN DB2 복제에 의해 생성된 메시지

CCA 구성 지원 프로그램에 의해 생성된 메시지

CLI 콜 레벨 인터페이스에 의해 생성된 메시지

DBA 데이터베이스 관리 도구에 의해 생성된 메시지

DBI 설치 및 구성에 의해 생성된 메시지

DBT 데이터베이스 도구에 의해 생성된 메시지

DB2 명령행 프로세서에 의해 생성된 메시지

DQP Query Patroller에 의해 생성된 메시지

EAS 임베디드(embedded) 응용프로그램 서버에 의해 생성된 메시지

EXP Explain 유틸리티에 의해 생성된 메시지

GSE DB2 Spatial Extender에 의해 생성된 메시지

LIC DB2 라이센스 관리자에 의해 생성된 메시지

MQL MQ 리스너에 의해 생성된 메시지

SAT Satellite 환경에 의해 생성된 메시지

SPM 동기점 관리 프로그램에 의해 생성된 메시지

SQL 경고 또는 오류 조건이 발견된 경우 데이터베이스 관리 프로그램에 의해 생성된 메시지

XMR XML 메타데이서 저장소에 의해 생성된 메시지.

메시지 도움말을 호출하려면 명령행 처리기를 열고 다음 명령을 입력하십시오.

```
? XXXnnnnn
```

여기에서 *XXX*는 유효한 메시지 접두부를 나타내고 *nnnnn*은 유효한 메시지 번호를 나타냅니다.

주어진 SQLSTATE 값과 연관된 메시지 텍스트는 다음을 발행하여 검색할 수 있습니다.

```
? nnnnn
```

또는 다음을 실행하는 경우,

```
? nn
```

여기에서 *nnnnn*은 다섯 자리 SQLSTATE(영숫자)이고 *nn*은 두 자리 SQLSTATE 클래스 모드(SQLSTATE 값의 첫 번째 두 자리 수)입니다.

주: db2 명령의 매개변수로 승인된 메시지 ID는 대소문자를 구분하지 않습니다. 또한 단일 문자 접미부는 선택적이며 무시됩니다.

따라서 다음 명령의 결과는 같습니다.

- ? SQL0000N
- ? sql0000
- ? SQL0000w

UNIX 기반 시스템의 명령행에 도움말을 호출하려면 다음을 입력하십시오.

```
db2 "? XXXnnnnn"
```

여기에서 *XXX*는 유효한 메시지 접두부를 나타내고
*nnnnn*은 유효한 메시지 번호를 나타냅니다.

메시지 텍스트가 화면에 너무 길면 Unix 기반 시스템 및 '추가'를 지원하는 다른 시스템의 경우 다음 명령을 사용하십시오.

```
db2 "? XXXnnnnn" | more
```

기타 DB2 메시지

일부 DB2 구성요소는 온라인으로 사용할 수 없거나 이 메뉴얼에 설명되어 있지 않은 메시지를 리턴합니다. 일부 메시지 접두부에는 다음이 포함되어 있습니다.

AUD DB2 감사 기능에 의해 생성된 메시지

DIA 여러 DB2 구성요소에 의해 생성된 진단 메시지. 이 메시지는 db2diag 로그 파일에 작성되며 오류 조사 시 사용자와 DB2 서비스 담당자에게 추가 정보를 제공하기 위한 것입니다.

GOV DB2 조정자 유틸리티에 의해 생성된 메시지

대부분의 경우 이 메시지는 경고 또는 오류의 원인을 판별할 수 있는 충분한 정보를 제공합니다. 메시지를 생성한 명령 또는 유틸리티에 대한 자세한 정보는 명령 또는 유틸리티가 문서화된 해당 매뉴얼을 참조하십시오.

기타 메시지 소스

시스템에 다른 프로그램이 실행 중인 경우 이 참조에 언급되지 않은 다른 접두부가 포함된 메시지가 수신됩니다.

이 메시지에 대한 정보는 해당 프로그램 제품에 사용 가능한 정보를 참조하십시오.

플랫폼별 오류 로그 정보

문제점 분석을 돕기 위해 DB2 밖에서 사용 가능한 기타 많은 파일과 유틸리티가 있습니다. DB2 파일에서 정보를 사용하는 것 못지 않게 근본 원인을 판별하는 것이 중요한 경우가 많습니다.

기타 파일 및 유틸리티는 다음 영역과 관련된 로그 및 추적에 포함된 정보에 액세스할 수 있게 합니다.

- 운영 체제
- 응용프로그램 및 다른 업체
- 하드웨어

운영 환경에 따라 여기 설명된 것 말고 더 많은 위치가 있을 수 있으므로 시스템에서 문제점을 디버깅할 때 조사해야 하는 모든 잠재적인 영역을 알아 두십시오.

운영 체제

모든 운영 체제마다 활동 및 실패를 추적하기 위한 자체 진단 파일 세트가 있습니다. 가장 일반적이면서 대개 가장 유용한 것은 오류 보고서 또는 이벤트 로그입니다. 이 정보를 수집할 수 있는 방법 목록은 다음과 같습니다.

- AIX: /usr/bin/errpt -a 명령
- Solaris: `/var/adm/messages*` 파일 또는 /usr/bin/dmesg 명령
- Linux: `/var/log/messages*` 파일 또는 /bin/dmesg 명령
- HP-UX: `/var/adm/syslog/syslog.log` 파일 또는 /usr/bin/dmesg 명령
- Windows: 시스템, 보안 및 응용프로그램 이벤트 로그 파일과 `windir₩drwtsn32.log` 파일(여기서 windir은 Windows 설치 디렉토리임)

각 운영 체제에 맞는 추적 및 디버깅 유틸리티는 항상 더 있습니다. 사용 가능한 추가 정보를 판별하려면 운영 체제 문서 및 지원 자료를 참조하십시오.

응용프로그램 및 다른 업체

각 응용프로그램마다 자체 로깅 및 진단 파일이 있어야 합니다. 이러한 파일은 DB2 정보 세트를 보완하여 잠재적인 문제점 영역에 대한 보다 정확한 그림을 제공합니다.

하드웨어

하드웨어 디바이스는 일반적으로 운영 체제 오류 로그에 정보를 로그합니다. 그러나 추가 정보가 필요한 경우도 있습니다. 해당 경우, 사용자의 환경에서 하드웨어 조각에 사용할 수 있는 하드웨어 진단 파일과 유틸리티를 식별해야 합니다. 이러한 경우의 예는 DB2가 잘못된 페이지 또는 일부 유형 손상을 보고하는 경우입니다. 일반적으로 이는 디스크 문제점으로 인해 보고되며, 이 경우 하드웨어 진단을 조사해야 합니다. 사용 가능한 추가 정보를 판별하려면 하드웨어 문서 및 지원 자료를 참조하십시오.

일부 정보(예: 하드웨어 로그의 정보)는 시간에 민감합니다. 오류가 발생하면, 가능한 빨리 관련 소스로부터 가능한 많은 정보를 수집하기 위해 모든 노력을 기울여야 합니다.

요약하면, 문제점을 완벽히 이해하고 평가하려면 DB2, 응용프로그램, 운영 체제 및 기본 하드웨어에서 사용 가능한 모든 정보를 수집해야 합니다. db2support 도구가 사용자가 필요로 하는 대부분의 DB2 및 운영 체제 정보 수집을 자동화하긴 하지만, 이 밖에 조사에 도움이 될 수 있는 정보도 여전히 알아야 합니다.

시스템 코어 파일(Linux 및 UNIX)

프로그램이 비정상적으로 종료되면, 시스템은 코어 파일을 작성하여 종료된 프로세스의 메모리 이미지를 저장합니다. 메모리 주소 위반, 잘못된 명령어, 버스 오류 및 사용자 생성 종료 플래그와 같은 오류로 인해 코어 파일이 덤프됩니다.

코어 파일의 이름은 ″core″이며, DB2FODC 레지스트리 변수의 값을 사용하여 별도로 구성되지 않으면 기본적으로 **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 지정한 디렉토리에 배치됩니다. 시스템 코어 파일은 DB2 트랩 파일과 다름에 유의하십시오.

시스템 코어 파일 정보에 액세스(Linux 및 UNIX)

dbx 시스템 명령은 어떤 함수로 인해 시스템 코어 파일이 작성되었는지 판별하는 데 도움을 줍니다. 이는 데이터베이스 관리 프로그램에 오류가 있는지 여부 또는 운영 체제나 응용프로그램 오류로 인해 문제점이 발생하는지 여부를 식별하는 데 도움을 주는 간단한 점검입니다.

- dbx 명령이 설치되어 있어야 합니다. 이 명령은 운영 체제별로 다릅니다. AIX 및 Solaris에서는 dbx를 사용하고 HP-UX에서는 xdb를 사용하며, Linux에서는 gdb를 사용하십시오.
- AIX에서는 chdev 명령 또는 smitty를 사용하여 전체 코어 옵션을 사용할 수 있게 했는지 확인하십시오.

다음 단계를 사용하여 코어 파일 덤프 발생의 원인이 된 함수를 판별할 수 있습니다.

1. UNIX 명령 프롬프트에서 다음 명령을 입력하십시오.

   ```
   dbx program_name core_filename
   ```

*program_name*은 비정상적으로 종료된 프로그램 이름이고 *core_filename*은 코어 파일 덤프가 포함된 파일 이름입니다. *core_filename* 매개변수는 선택적입니다. 이를 지정하지 않으면, 디폴트 이름 ″core″가 사용됩니다.

2. 코어 파일의 호출 스택을 검사하십시오. UNIX 명령 프롬프트에서 man dbx를 실행하면 이를 수행하는 방법에 대한 정보를 얻을 수 있습니다.
3. dbx 명령을 종료하려면, dbx 프롬프트에서 quit를 입력하십시오.

다음 예는 dbx 명령을 사용하여 ″main″이라는 프로그램의 코어 파일을 읽는 방법을 나타냅니다.

1. 명령 프롬프트에서 다음을 입력하십시오.

```
dbx main
```

2. 다음과 유사한 출력이 표시장치에 나타납니다.

```
dbx version 3.1 for AIX.
Type 'help' for help.
reading symbolic information ...
[using memory image in core]
segmentation.violation in freeSegments at line 136
136         (void) shmdt((void *) pcAdress[i]);
```

3. 코어 덤프를 일으킨 함수의 이름은 ″freeSegments″입니다. 프로그램 장애 지점의 경로를 표시하려면 dbx 프롬프트에서 where를 입력하십시오.

```
(dbx) where
freeSegments(numSegs = 2, iSetId = 0x2ff7f730, pcAddress = 0x2ff7f758, line
136
in "main.c"
main (0x1, 2ff7f7d4), line 96 in "main.c"
```

이 예에서는 freeSegments(main.c의 96라인에서 호출됨)의 136 라인에서 오류가 발생했습니다.

4. dbx 명령을 종료하려면, dbx 프롬프트에서 quit를 입력하십시오.

이벤트 로그에 액세스(Windows)

이 태스크는 Windows 이벤트 로그에 액세스하는 방법을 설명합니다.

Windows 이벤트 로그도 유용한 정보를 제공할 수 있습니다. DB2 파손 또는 기타 이해할 수 없는 시스템 자원 관련 오류가 발생할 경우 시스템 이벤트 로그가 가장 유용하긴 하지만, 다음 세 개의 이벤트 로그 유형을 모두 확보할 만한 가치가 있습니다.

- 시스템
- 응용프로그램
- 보안

Windows 이벤트 뷰어를 사용하여 이벤트 로그를 살펴보십시오. 뷰어를 여는 데 사용되는 방법은 사용 중인 Windows 운영 체제에 따라 다릅니다.

예를 들어, Windows XP에서 이벤트 뷰어를 열려면 **시작 —> 제어판**을 누르십시오. **관리 도구**를 선택한 후 **이벤트 뷰어**를 두 번 누르십시오.

이벤트 로그 익스포트(Windows)

이 태스크는 Windows 이벤트 로그 익스포트 방법을 설명합니다.

Windows 이벤트 뷰어에서 다음 두 가지 형식으로 이벤트 로그를 익스포트할 수 있습니다.

- 로그 파일 형식
- 텍스트 또는 쉼표로 구분된 형식

Windows 이벤트 뷰어에서 이벤트 로그를 익스포트하십시오.

- 로그 파일 형식(*.evt) 데이터를 이벤트 뷰어로(예를 들어, 다른 워크스테이션에서) 다시 로드할 수 있습니다. 뷰어를 사용하여 연대순으로 전환하고 특정 이벤트를 필터링하며, 앞으로 또는 뒤로 진행할 수 있으므로 이 형식은 작업하기 용이합니다.
- 대부분의 텍스트 편집기에서 텍스트(*.txt) 또는 쉼표로 구분된(*.csv) 형식 로그를 열 수 있습니다. 또한 이러한 로그는 시간소인과 관련하여 잠재적 문제점을 방지합니다. .evt 형식으로 이벤트 로그를 익스포트하는 경우, 시간소인은 세계 표준시로 되어 있으며 뷰어에서 워크스테이션의 로컬 시간으로 변환됩니다. 주의하지 않으면 시간대 차이로 인해 주요 이벤트를 간과할 수 있습니다. 텍스트 파일도 검색하기 용이합니다.

Dr. Watson 로그 파일에 액세스(Windows)

이 태스크는 Windows 시스템에서 Dr. Watson 로그 파일에 액세스하는 방법을 설명합니다.

Dr. Watson 로그(drwtsn32.log)는 시스템에서 발생한 모든 예외의 연대기입니다. Dr. Watson 로그가 전반적인 시스템 안정성을 평가할 때 유용하고 DB2 트랩 실행기록 문서로 유용하긴 하지만 DB2 트랩 파일이 Dr. Watson 로그보다 더 유용합니다.

Dr. Watson 로그 파일을 찾으십시오. 디폴트 경로는 <install_drive>:₩Documents and Settings ₩All Users₩Documents₩DrWatson입니다.

트랩 파일

DB2는 트랩, 세그먼트 위반 또는 예외로 인해 처리를 계속할 수 없는 경우 트랩 파일을 생성합니다.

DB2가 수신하는 모든 신호 또는 예외는 트랩 파일에 기록됩니다. 또한 트랩 파일은 오류가 발생할 당시 실행 중이던 함수 시퀀스를 포함합니다. 이 시퀀스를 "함수 호출 스택" 또는 "스택 추적"이라고 하는 경우도 있습니다. 트랩 파일은 신호 또는 예외가 발생했을 때 프로세스 상태에 대한 추가 정보도 포함합니다.

트랩 파일은 분리(fenced) 스레드 안전 루틴을 실행하는 동안 응용프로그램이 강제 실행되는 경우에도 생성됩니다. 프로세스가 종료되면 트랩이 발생합니다. 이는 치명적 오류는 아니며 걱정하지 않아도 됩니다.

파일은 **diagpath** 데이터베이스 관리 프로그램 구성 매개변수가 지정한 디렉토리에 있습니다.

모든 플랫폼에서 트랩 파일 이름은 프로세스 ID(PID)로 시작하며, 그 다음에 스레드 ID(TID)와 파티션 번호(단일 파티션 데이터베이스에서는 000)가 차례로 오고『.trap.txt』로 끝납니다.

인스턴스 파손을 보증하지 않지만 스택을 살펴보는 것이 유용한 특정 조건이 발생할 때 코드가 생성하는 진단 트랩도 있습니다. 해당 트랩 이름은 10진수 형식의 PID 다음에 파티션 번호(단일 파티션 데이터베이스에서는 0)가 옵니다.

예:

- `6881492.2.000.trap.txt`는 프로세스 ID(PID)가 6881492이고 스레드 ID(TID)가 2인 트랩 파일입니다.
- `6881492.2.010.trap.txt`는 프로세스 및 스레드가 파티션 10에서 실행 중인 트랩 파일입니다.

db2pd 명령을 -stack all 또는 -dump 옵션과 함께 사용하여 요구가 있을 때 트랩 파일을 생성할 수 있습니다. 일반적으로 IBM Software Support에서 요청할 때에만 이를 수행해야 합니다.

db2pd -stacks 또는 db2pd -dumps 명령을 사용하여 스택 추적 파일을 생성할 수 있습니다. 이러한 파일은 트랩 파일과 컨텐츠는 동일하지만 진단 목적으로만 생성됩니다. 이름은 `6881492.2.000.stack.txt`와 유사합니다.

트랩 파일 포맷팅(Windows)

db2xprt 명령을 사용하여 트랩 파일(*.TRP)을 형식화할 수 있습니다. 이 명령은 DB2 데이터베이스 2진 트랩 파일을 사람이 읽을 수 있는 ASCII 파일로 형식화합니다.

db2xprt 도구는 트랩 파일을 형식화하기 위해 DB2 기호 파일을 사용합니다. 이러한 .PDB 파일의 서브세트는 DB2 데이터베이스 제품에 포함되어 있습니다.

diagpath 데이터베이스 관리 프로그램 구성 매개변수가 지정한 디렉토리에 ″DB30882416.TRP″라는 트랩 파일이 생성된 경우, 다음과 같이 이 파일을 형식화할 수 있습니다.

```
db2xprt DB30882416.TRP DB30882416.FMT
```

제 11 장 IBM Software Support에 문의

IBM Software Support에 문의

IBM Software Support에서는 제품 결함에 대한 지원을 제공합니다.

IBM Software Support에 문의하기 전에 회사가 IBM 소프트웨어 유지보수 계약을 체결한 상태여야 하며, IBM에 문제점을 제출할 수 있는 권한이 사용자에게 있어야 합니다. 유지보수 계약 유형에 대한 정보는 *Software Support Handbook*의 『Premium Support』(techsupport.services.ibm.com/guides/services.html)를 참조하십시오.

IBM Software Support에 문제점을 문의하려면 다음 단계를 완료하십시오.

1. 문제점을 정의하고 백그라운드 정보를 수집한 후 문제점의 심각도를 판별하십시오. 도움말을 보려면 *Software Support Handbook*의 『Contacting IBM』(techsupport.services.ibm.com/guides/beforecontacting.html)을 참조하십시오.
2. 진단 정보를 수집하십시오.
3. 다음 방법 중 하나로 IBM Software Support에 문제점을 제출하십시오.
   - 온라인: IBM Software Support, Open Service Request 사이트(www.ibm.com/software/support/probsub.html)의 **ESR**(Electronic Service Request) 링크를 누르십시오.
   - 전화: 해당 국가/지역의 전화번호를 알려면 *Software Support Handbook*의 Contacts 페이지(techsupport.services.ibm.com/guides/contacts.html)로 이동하십시오.

제출하는 문제점이 소프트웨어 결함이나 문서 누락 또는 부정확성에 대한 문제일 경우, IBM Software Support에서는 APAR(Authorized Program Analysis Report)을 작성합니다. APAR은 문제점을 자세히 설명합니다. 가능하면 IBM Software Suppor는 APAR이 해결되거나 수정사항이 제공될 때까지 구현할 수 있는 일시적인 해결책을 제공합니다. IBM은 해결된 APAR을 IBM Software Support 웹 사이트에 매일 게시하므로 동일한 문제점을 경험하는 다른 사용자가 동일한 해결을 통해 혜택을 볼 수 있습니다.

IBM Software Support에 데이터 제출

FTP를 통해 또는 ESR(Electronic Service Request) 도구를 사용하여 IBM Software Support에 데이터를 제출할 수 있습니다. 여기서는 지시사항을 제공합니다.

단계에서는 사용자가 IBM Software Support에 문제점 관리 레코드(PMR)를 이미 열었다고 가정합니다.

다음 방법 중 하나를 사용하여 로그 파일, 구성 파일과 같은 진단 데이터를 IBM Software Support에 보낼 수 있습니다.

- FTP
- ESR(Electronic Service Request) 도구
- FTP를 통해 EcuRep(Enhanced Centralized Client Data Repository)에 파일을 제출하려면 다음을 수행하십시오.
  1. ZIP 또는 TAR 형식으로 수집한 데이터 파일을 패키지하고 문제점 관리 레코드(PRM) ID에 따라 패키지 이름을 지정하십시오.

     PMR과 올바로 연관되도록 xxxxx.bbb.ccc.yyy.yyy 이름 지정 규칙을 사용해야 합니다(여기서 xxxxx는 PMR 번호이고 bbb는 PMR의 분기 번호이며, ccc는 PRM의 지역 코드이고 yyy.yyy는 파일 이름임).
  2. FTP 유틸리티를 사용하여 ftp.emea.ibm.com 서버에 연결하십시오.
  3. 사용자 ID "anonymous"로 로그인하고 전자 우편 주소를 암호로 입력하십시오.
  4. `toibm` 디렉토리로 이동하십시오(예: cd toibm).
  5. 운영 체제별 서브디렉토리 중 하나로 이동하십시오. 예를 들어, 서브디렉토리는 `aix`, `linux`, `unix` 또는 `windows`입니다.
  6. 2진 모드로 변경하십시오. 예를 들어, 명령 프롬프트에서 bin을 입력하십시오.
  7. put 명령을 사용하여 서버에 파일을 배치하십시오. 다음 파일 이름 지정 규칙을 사용하여 파일 이름을 지정하고 서버에 파일을 배치하십시오. PMR이 갱신되어 `xxxx.bbb.ccc.yyy.yyy` 형식을 사용하여 파일이 저장된 위치를 나열합니다(xxx는 PMR 번호이고 bbb는 분기이며, ccc는 지역 코드이고 yyy.yyy는 `tar.Z` 또는 `xyz.zip`과 같은 파일 유형에 대한 설명임). FTP 서버로 파일을 보낼 수 있지만 파일을 갱신할 수는 없습니다. 나중에 파일을 변경해야 하는 경우, 새 파일 이름을 작성해야 합니다.
  8. quit 명령을 입력하십시오.
- ESR 도구를 사용하여 파일을 제출하려면 다음을 수행하십시오.
  1. ESR에 로그온하십시오.
  2. 시작 페이지에서 **보고서 번호 입력** 필드에 PMR 번호를 입력하고 **이동**을 누르십시오.
  3. **관련 파일 첨부** 필드로 아래로 스크롤하십시오.
  4. **찾아보기**를 눌러 로그, 추적 또는 IBM Software Support에 제출할 기타 진단 파일을 찾으십시오.
  5. **제출**을 누르십시오. 파일이 FTP를 통해 IBM Software Support로 전송되며, PMR과 연관됩니다.

EcuRep 서비스에 대한 자세한 정보는 IBM EMEA Centralized Customer Data Store Service를 참조하십시오.

ESR에 대한 자세한 정보는 ESR(Electronic Service Request) 도움말을 참조하십시오.

제 3 부 부록

부록 A. DB2 기술 정보 개요

DB2 기술 정보는 다음 도구 및 메소드를 통해 사용할 수 있습니다.

- DB2 정보 센터
  - 주제 항목(태스크, 개념 및 참조 항목)
  - DB2 도구에 대한 도움말
  - 샘플 프로그램
  - 자습서
- DB2 서적
  - PDF 파일(다운로드)
  - PDF 파일(DB2 PDF DVD)
  - 인쇄된 서적
- 명령행 도움말
  - 명령 도움말
  - 메시지 도움말

주: DB2 정보 센터의 주제는 PDF 또는 하드카피 서적보다 더 자주 갱신됩니다. 최신 정보를 보려면 사용 가능한 문서 갱신사항을 설치하거나 ibm.com에서 DB2 정보 센터를 참조하십시오.

ibm.com에서 추가 DB2 기술 정보(예: 기술 노트, 백서 및 IBM Redbooks 서적)를 온라인으로 액세스할 수 있습니다. 다음은 DB2 정보 관리 라이브러리 소프트웨어 사이트의 주소입니다. http://www.ibm.com/software/data/sw-library/

문서 피드백

DB2 문서에 대한 피드백을 환영합니다. DB2 문서를 향상시키는 방법에 대해서 제안사항이 있는 경우 db2docs@ca.ibm.com으로 전자 우편을 보내십시오. DB2 문서 팀에서는 고객의 모든 피드백을 읽지만 직접 응답할 수는 없습니다. 고객의 문제를 더 잘 이해할 수 있도록 가능한 한 구체적인 예를 제공해주십시오. 특정 주제 또는 도움말 파일에 대한 피드백을 보내실 경우, 제목 및 URL을 알려주십시오.

DB2 고객 지원에 문의할 때는 이 전자 우편 주소를 사용하지 마십시오. 문서에서 해결할 수 없는 DB2 기술 문제점이 있는 경우, 해당 지역의 IBM 서비스 센터에 도움을 요청하십시오.

DB2 기술 라이브러리(하드카피 또는 PDF 형식)

다음 표는 IBM Publications Center(www.ibm.com/shop/publications/order)에서 사용할 수 있는 DB2 라이브러리에 대한 설명입니다. PDF 형식의 영문 DB2 버전 9.7 매뉴얼 및 번역된 버전은 www.ibm.com/support/docview.wss?rs=71&uid=swg2700947에서 다운로드할 수 있습니다.

표에 인쇄할 수 있는 책으로 설명된 경우라도, 사용 국가 또는 지역에 따라 해당 책을 사용할 수 없을 수도 있습니다.

매뉴얼이 갱신될 때마다 문서 번호가 증가합니다. 다음 사항을 참조하여 읽고 있는 매뉴얼이 최신 버전인지 확인하십시오.

주: *DB2 정보 센터*는 PDF 또는 하드카피 서적보다 자주 갱신됩니다.

표 85. DB2 기술 정보

이름	문서 번호	인쇄 가능	마지막 갱신 날짜
관리 API 참조서	SA30-3958-00	예	2009년 8월
관리 루틴 및 뷰	SA30-3955-00	아니오	2009년 8월
Call Level Interface Guide and Reference, Volume 1	SC27-2437-00	예	2009년 8월
Call Level Interface Guide and Reference, Volume 2	SC27-2438-00	예	2009년 8월
명령어 참조서	SA30-3959-00	예	2009년 8월
데이터 이동 유틸리티 안내서 및 참조서	SA30-3969-00	예	2009년 8월
데이터 복구 및 고가용성 안내서 및 참조서	SA30-3970-00	예	2009년 8월
데이터베이스 관리 개념 및 구성 참조서	SA30-3951-00	예	2009년 8월
데이터베이스 모니터링 안내서 및 참조서	SA30-3953-00	예	2009년 8월
데이터베이스 보안 안내서	SA30-3971-00	예	2009년 8월
DB2 Text Search Guide	SC27-2459-00	예	2009년 8월
Developing ADO.NET and OLE DB Applications	SC27-2444-00	예	2009년 8월
Developing Embedded SQL Applications	SC27-2445-00	예	2009년 8월
Developing Java Applications	SC27-2446-00	예	2009년 8월
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-2447-00	아니오	2009년 8월

표 85. DB2 기술 정보 (계속)

이름	문서 번호	인쇄 가능	마지막 갱신 날짜
Developing User-defined Routines(SQL and External)	SC27-2448-00	예	2009년 8월
Getting Started with Database Application Development	GI11-9410-00	예	2009년 8월
Linux 및 Windows에서 DB2 설치 및 관리 시작하기	GA30-3960-00	예	2009년 8월
자국어 안내서	SA30-3972-00	예	2009년 8월
DB2 Server 설치	GA30-3962-00	예	2009년 8월
IBM Data Server Client 설치	GA30-3963-00	아니오	2009년 8월
Message Reference Volume 1	SC27-2450-00	아니오	2009년 8월
Message Reference Volume 2	SC27-2451-00	아니오	2009년 8월
Net Search Extender Administration and User's Guide	SC27-2469-00	아니오	2009년 8월
파티셔닝 및 클러스터링 안내서	SA30-3973-00	예	2009년 8월
pureXML Guide	SC27-2465-00	예	2009년 8월
Query Patroller 관리 및 사용자 안내서	SA30-3974-00	아니오	2009년 8월
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC27-2468-00	아니오	2009년 8월
SQL Procedural Languages: Application Enablement and Support	SC27-2470-00	예	2009년 8월
SQL 참조서, 볼륨 1	SA30-3956-00	예	2009년 8월
SQL 참조서, 볼륨 2	SA30-3957-00	예	2009년 8월
문제점 해결 및 데이터베이스 성능 조정	SA30-3952-00	예	2009년 8월
DB2 버전 9.7로 업그레이드	SA30-3961-00	예	2009년 8월
Visual Explain 자습서	SA30-3968-00	아니오	2009년 8월
DB2 버전 9.7의 새로운 내용	SA30-3967-00	예	2009년 8월
Workload Manager Guide and Reference	SC27-2464-00	예	2009년 8월

표 85. DB2 기술 정보 (계속)

이름	문서 번호	인쇄 가능	마지막 갱신 날짜
XQuery Reference	SC27-2466-00	아니오	2009년 8월

표 86. DB2 Connect 특정 기술 정보

이름	문서 번호	인쇄 가능	마지막 갱신 날짜
DB2 Connect Personal Edition 설치 및 구성	SA30-3965-00	예	2009년 8월
DB2 Connect Server 설치 및 구성	SA30-3966-00	예	2009년 8월
DB2 Connect 사용자 안내서	SA30-3964-00	예	2009년 8월

표 87. Information Integration 기술 정보

이름	문서 번호	인쇄 가능	마지막 갱신 날짜
Information Integration: Administration Guide for Federated Systems	SC19-1020-02	예	2009년 8월
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-04	예	2009년 8월
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-02	아니오	2009년 8월
Information Integration: SQL Replication Guide and Reference	SC19-1030-02	예	2009년 8월
Information Integration: Introduction to Replication and Event Publishing	GC19-1028-02	예	2009년 8월

인쇄된 DB2 서적 주문

인쇄된 DB2 서적이 필요한 경우, 대부분 온라인으로 구매할 수 있으나 모든 국가 또는 지역에서 가능한 것은 아닙니다. 언제든지 해당 지역의 IBM 담당자로부터 인쇄된 DB2 서적을 주문할 수 있습니다. *DB2 PDF 문서* DVD의 일부 소프트카피 서적은 인쇄할 수 없다는 점에 유의하십시오. 예를 들어, *DB2 메시지 참조서*의 볼륨은 인쇄된 서적으로 사용할 수 없습니다.

DB2 PDF 문서 DVD에서 사용할 수 있는 다수의 DB2 서적의 인쇄된 버전은 IBM에서 유료로 주문할 수 있습니다. 주문하는 위치에 따라 IBM Publications Center에서 온라인으로 서적을 주문할 수도 있습니다. 해당 국가 또는 지역에서 온라인 주문이

불가능하면, 언제든지 해당 지역의 IBM 담당자로부터 인쇄된 DB2 서적을 주문할 수 있습니다. DB2 PDF 문서 DVD의 모든 서적을 인쇄할 수는 없다는 점에 유의하십시오.

주: 가장 최신의 완전한 DB2 문서는 http://publib.boulder.ibm.com/infocenter/db2luw/v9r7의 DB2 정보 센터에서 유지보수됩니다.

인쇄된 DB2 서적을 주문하려면 다음을 수행하십시오.

- 해당 국가 또는 지역에서 인쇄된 DB2 서적을 온라인으로 주문할 수 있는지 여부를 확인하려면 http://www.ibm.com/shop/publications/order의 IBM Publications Center를 확인하십시오. 서적 주문 정보를 액세스하려면 국가/지역/언어를 선택한 다음 해당 위치에서 주문 지시사항을 따르십시오.
- 해당 지역의 IBM 담당자로부터 인쇄된 DB2 서적을 주문하려면 다음을 수행하십시오.
  1. 다음 웹 사이트 중 하나에서 해당 지역 담당자에 대한 문의처 정보를 찾으십시오.
     - www.ibm.com/planetwide에 있는 IBM 전세계 문의처 디렉토리
     - http://www.ibm.com/shop/publications/order의 IBM Publications 웹 사이트. 사용 지역의 해당 서적 홈 페이지에 액세스하려면 해당 국가, 지역 또는 언어를 선택해야 합니다. 이 페이지에서 "이 제품의 정보" 링크를 수행하십시오.
  2. 전화로 주문할 경우, 주문할 DB2 서적을 지정하십시오.
  3. 담당자에게 주문하려는 서적의 제목 및 문서 번호를 제공하십시오. 서적의 제목 및 문서 번호는 620 페이지의 『DB2 기술 라이브러리(하드카피 또는 PDF 형식)』를 참조하십시오.

명령행 처리기에서 SQL 상태 도움말 표시

DB2 제품은 SQL문의 결과로 나타나는 상태에 대한 SQLSTATE 값을 리턴합니다. SQLSTATE 도움말은 SQL 상태 및 SQL 상태 클래스 코드의 의미를 설명합니다.

SQL 상태 도움말을 시작하려면 명령행 처리기를 열고 다음을 입력하십시오.

```
? sqlstate or ? class code
```

여기서, *sqlstate*는 유효한 5자리 숫자로 된 SQL 상태이고 *class code*는 SQL 상태의 처음 2자리 숫자를 나타냅니다.
예를 들어, `? 08003`은 08003 SQL 상태에 대한 도움말을 표시하고, `? 08`은 08 클래스 코드에 대한 도움말을 표시합니다.

DB2 정보 센터의 다른 버전에 액세스

DB2 버전 9.7 주제에 대한 DB2 정보 센터 URL은 http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/입니다.

DB2 버전 9.5 주제에 대한 DB2 정보 센터 URL은 http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/입니다.

DB2 버전 9 주제에 대한 DB2 정보 센터 URL은 http://publib.boulder.ibm.com/infocenter/db2luw/v9/입니다.

DB2 버전 8 주제에 대한 버전 8 정보 센터 URL은 http://publib.boulder.ibm.com/infocenter/db2luw/v8/입니다.

DB2 정보 센터에서 원하는 언어로 항목 표시

DB2 정보 센터는 브라우저 환경 설정에 지정된 언어로 주제 항목을 표시합니다. 주제가 원하는 언어로 변환되지 않은 경우, DB2 정보 센터는 해당 주제 항목을 영어로 표시합니다.

- Internet Explorer 브라우저에서 원하는 언어로 항목을 표시하려면 다음을 수행하십시오.
  1. Internet Explorer에서 **도구 —> 인터넷 옵션 —> 언어** 단추를 누르십시오. 언어 환경 설정 창이 열립니다.
  2. 원하는 언어가 언어 목록의 첫 번째 항목으로 지정되었는지 확인하십시오.
     - 목록에 새 언어를 추가하려면 **추가...** 단추를 누르십시오.

       주: 언어를 추가하더라도 원하는 언어로 항목을 표시하는 데 필요한 글꼴이 컴퓨터에 설치되지는 않습니다.
     - 언어를 목록 맨위로 이동하려면, 언어를 선택한 후 해당 언어가 언어 목록의 첫 번째 항목이 될 때까지 **위로 이동** 단추를 누르십시오.
  3. 브라우저 캐시를 지운 후 페이지를 새로 고치면 원하는 언어로 DB2 정보 센터가 표시됩니다.
- Firefox 또는 Mozilla 브라우저에서 원하는 언어로 주제 항목을 표시하려면 다음을 수행하십시오.
  1. **도구 —> 설정 —> 내용** 대화 상자의 **언어** 섹션에서 단추를 선택하십시오. 환경 설정 창에 언어 패널이 표시됩니다.
  2. 원하는 언어가 언어 목록의 첫 번째 항목으로 지정되었는지 확인하십시오.
     - 목록에 새 언어를 추가하려면 언어 선택 창에서 원하는 언어를 선택한 다음 **추가...** 단추를 누르십시오.

– 언어를 목록 맨위로 이동하려면, 언어를 선택한 후 해당 언어가 언어 목록의 첫 번째 항목이 될 때까지 **위로 이동** 단추를 누르십시오.

3. 브라우저 캐시를 지운 후 페이지를 새로 고치면 원하는 언어로 DB2 정보 센터가 표시됩니다.

일부 브라우저 및 운영 체제 조합에서는 운영 체제의 국가별 설정을 선택한 로케일 및 언어로 변경해야 합니다.

컴퓨터 또는 인트라넷 서버에 설치된 DB2 정보 센터 갱신

로컬로 설치된 DB2 정보 센터는 주기적으로 갱신해야 합니다.

시작하기 전에

DB2 버전 9.7 정보 센터는 미리 설치된 상태여야 합니다. 자세한 내용은 *DB2 Server 설치*의 『DB2 설치 마법사를 사용하여 DB2 정보 센터 설치』 주제를 참조하십시오. 정보 센터 설치에 적용되는 모든 전제조건 및 제한사항은 정보 센터 갱신에도 적용됩니다.

이 태스크에 대한 정보

기존의 DB2 정보 센터는 자동 또는 수동으로 갱신할 수 있습니다.

- 자동 갱신 - 기존 정보 센터 기능 및 언어를 갱신합니다. 자동 갱신의 또 다른 이점으로는 갱신 동안 정보 센터를 사용할 수 없는 시간이 매우 짧다는 점입니다. 또한 자동 갱신은 주기적으로 실행되는 기타 일괄처리 작업의 일부로 실행되도록 설정할 수도 있습니다.
- 수동 갱신 - 갱신 프로세스 중에 기능이나 언어를 추가하려는 경우 사용하십시오. 예를 들어, 로컬 정보 센터는 기본적으로 영어와 프랑스어로 설치되어 있으며, 수동 갱신을 통해 기존 정보 센터의 기능 및 언어 갱신뿐만 아니라 독일어도 설치할 수 있습니다. 단, 수동 갱신을 수행하려면 정보 센터를 중지한 다음 갱신하고 재시작해야 합니다. 정보 센터는 갱신 프로세스 동안에는 사용할 수 없습니다.

프로시저

이 주제는 자동 갱신 프로세스에 대한 설명입니다. 수동 갱신에 대한 지시사항은 『컴퓨터 또는 인트라넷 서버에 설치된 DB2 정보 센터 수동 갱신』 주제를 참조하십시오.

컴퓨터 또는 인트라넷 서버에 설치된 DB2 정보 센터를 자동으로 갱신하려면 다음을 수행하십시오.

1. Linux 운영 체제의 경우
   a. 정보 센터가 설치된 경로를 찾아가십시오. DB2 정보 센터는 `/opt/ibm/db2ic/V9.7` 디렉토리에 디폴트로 설치됩니다.

b. 설치 디렉토리에서 doc/bin 디렉토리로 이동하십시오.

c. 다음과 같이 ic-update 스크립트를 실행하십시오.

```
ic-update
```

2. Windows 운영 체제의 경우

a. 명령 창을 여십시오.

b. 정보 센터가 설치된 경로를 찾아가십시오. DB2 정보 센터는 <Program Files>₩IBM₩DB2 Information Center₩Version 9.7 디렉토리에 디폴트로 설치됩니다. 여기서 <Program Files>는 프로그램 파일 디렉토리의 위치를 나타냅니다.

c. 설치 디렉토리에서 doc₩bin 디렉토리로 이동하십시오.

d. 다음과 같이 ic-update.bat 파일을 실행하십시오.

```
ic-update.bat
```

결과

DB2 정보 센터가 자동으로 재시작됩니다. 갱신사항이 사용 가능한 경우, 정보 센터에는 새로 갱신된 주제가 표시됩니다. 정보 센터 갱신을 사용할 수 없는 경우, 메시지가 로그에 추가됩니다. 로그 파일은 doc₩eclipse₩configuration 디렉토리에 있습니다. 이 로그 파일 이름은 임의로 생성된 번호입니다. 예: 1239053440785.log

컴퓨터 또는 인트라넷 서버에 설치된 DB2 정보 센터 수동 갱신

DB2 정보 센터를 로컬로 설치한 경우, IBM으로부터 문서 갱신사항을 받아 설치할 수 있습니다.

로컬로 설치된 DB2 정보 센터를 수동으로 갱신하려면 다음을 수행하십시오.

1. 컴퓨터에서 DB2 정보 센터를 중지한 후 독립형 모드에서 다시 시작하십시오. 독립형 모드에서 정보 센터를 실행하면 사용자의 네트워크와 연결된 다른 사용자는 정보 센터에 액세스할 수 없으므로 갱신사항을 적용할 수 있습니다. DB2 정보 센터의 워크스테이션 버전은 항상 독립형 모드에서 실행됩니다.
2. 사용 가능한 갱신사항을 확인하려면 갱신 기능을 사용하십시오. 설치해야 할 갱신사항이 있는 경우, 갱신 기능을 사용하여 이를 가져온 후 설치할 수 있습니다.

   주: 인터넷에 연결되지 않은 머신에 DB2 정보 센터 갱신사항을 설치해야 할 경우, 인터넷에 연결되고 DB2 정보 센터가 설치된 머신을 사용하여 갱신 사이트를 로컬 파일 시스템으로 미러하십시오. 네트워크 상에 문서 갱신사항을 설치하려는 사용자가 많을 경우에는 갱신 사이트를 로컬로 미러링하거나 갱신 사이트의 프록시를 작성하여 갱신을 수행하면 각 개인에게 필요한 시간을 줄일 수 있습니다.
   갱신 패키지가 사용 가능하면 갱신 기능을 사용하여 패키지를 가져오십시오. 그러나 갱신 기능은 독립형 모드에서만 사용할 수 있습니다.

3. 독립형 정보 센터를 중지한 후 컴퓨터에서 DB2 정보 센터를 재시작하십시오.

주: Windows 2008, Windows Vista 이상의 경우 이 절 다음에 나오는 명령은 관리자로 실행해야 합니다. 전체 관리자 권한으로 명령 프롬프트 또는 그래픽 도구를 열려면 단축 아이콘을 마우스 오른쪽 단추로 누른 후 **관리자로 실행**을 선택하십시오.

컴퓨터 또는 인트라넷 서버에 설치된 DB2 정보 센터를 갱신하려면 다음을 수행하십시오.

1. DB2 정보 센터를 중지하십시오.
   - Windows의 경우, **시작 → 제어판 → 관리 도구 → 서비스**를 누르십시오. 그런 다음 **DB2 Information Center** 서비스를 마우스 오른쪽 단추로 누른 후 **중지**를 선택하십시오.
   - Linux의 경우, 다음 명령을 입력하십시오.

     ```
     /etc/init.d/db2icdv97 stop
     ```
2. 독립형 모드에서 정보 센터를 시작하십시오.
   - Windows의 경우:
     a. 명령 창을 여십시오.
     b. 정보 센터가 설치된 경로를 찾아가십시오. DB2 정보 센터는 `<Program Files>₩IBM₩DB2 Information Center₩Version 9.7` 디렉토리에 디폴트로 설치됩니다. 여기서 `<Program Files>`는 프로그램 파일 디렉토리의 위치를 나타냅니다.
     c. 설치 디렉토리에서 `doc₩bin` 디렉토리로 이동하십시오.
     d. 다음과 같이 `help_start.bat` 파일을 실행하십시오.

        ```
        help_start.bat
        ```
   - Linux의 경우:
     a. 정보 센터가 설치된 경로를 찾아가십시오. DB2 정보 센터는 `/opt/ibm/db2ic/V9.7` 디렉토리에 디폴트로 설치됩니다.
     b. 설치 디렉토리에서 `doc/bin` 디렉토리로 이동하십시오.
     c. 다음과 같이 `help_start` 스크립트를 실행하십시오.

        ```
        help_start
        ```

   시스템의 기본 웹 브라우저가 열리고 독립형 정보 센터가 표시됩니다.
3. **갱신** 단추( )를 누르십시오. (JavaScript™가 브라우저에서 사용 가능해야 합니다.) 정보 센터의 오른쪽 패널에서 **갱신사항 찾기**를 누르십시오. 기존 문서의 갱신사항 목록이 표시됩니다.
4. 설치 프로세스를 시작하려면 설치할 선택란을 체크한 후 **갱신사항 설치**를 누르십시오.
5. 설치 프로세스가 완료되면 **완료**를 누르십시오.

6. 독립형 정보 센터를 중지하십시오.
   - Windows의 경우, 설치 디렉토리의 doc¥bin 디렉토리로 이동한 후 다음과 같이 help_end.bat 파일을 실행하십시오.

     ```
     help_end.bat
     ```

     주: help_end 일괄처리 파일에는 help_start 일괄처리 파일로 시작된 프로세스를 안전하게 중지하는 데 필요한 명령이 포함되어 있습니다. help_start.bat를 중지할 때 Ctrl+C 또는 다른 메소드를 사용하지 마십시오.
   - Linux의 경우, 설치 디렉토리의 doc/bin 디렉토리로 이동한 후 다음과 같이 help_end 스크립트를 실행하십시오.

     ```
     help_end
     ```

     주: help_end 스크립트에는 help_start 스크립트로 시작된 프로세스를 안전하게 중지하는 데 필요한 명령이 포함되어 있습니다. help_start 스크립트를 중지할 때 다른 메소드를 사용하지 마십시오.
7. DB2 정보 센터를 재시작하십시오.
   - Windows의 경우, **시작 → 제어판 → 관리 도구 → 서비스**를 누르십시오. 그런 다음 **DB2 Information Center** 서비스를 마우스 오른쪽 단추로 누른 후 **시작**을 선택하십시오.
   - Linux의 경우, 다음 명령을 입력하십시오.

     ```
     /etc/init.d/db2icdv97 start
     ```

갱신된 DB2 정보 센터에는 새로 갱신된 주제가 표시됩니다.

DB2 자습서

DB2 자습서는 DB2 제품의 여러가지 측면을 학습하는 데 유용합니다. 각 레슨은 단계별 지시사항을 제공합니다.

시작하기 전에

정보 센터(http://publib.boulder.ibm.com/infocenter/db2help/)에서 XHTML 버전의 자습서를 볼 수 있습니다.

일부 레슨에서는 샘플 데이터나 코드를 사용합니다. 특정 태스크에 필요한 전제조건 설명은 자습서를 참조하십시오.

DB2 자습서

자습서를 보려면 제목을 누르십시오.

『**pureXML®**』(*pureXML Guide*)

DB2 데이터베이스를 설정하여 XML 데이터를 저장하고 원시 XML 데이터 스토어로 기본 조작을 수행할 수 있습니다.

*Visual Explain 자습서*의 『**Visual Explain**』

더 나은 성능을 위해 Visual Explain을 사용하여 SQL문을 분석, 최적화 및 조정할 수 있습니다.

DB2 문제점 해결 정보

DB2 데이터베이스 제품 사용 시 발생하는 광범위한 문제점을 판별하고 해결하는 데 도움이 되는 정보를 사용할 수 있습니다.

DB2 문서

문제점 해결 정보는 *DB2 문제점 해결 안내서* 또는 *DB2 정보 센터*의 데이터베이스 기본 절을 참조하십시오. DB2 진단 도구 및 유틸리티를 사용하여 문제점을 찾아내고 식별하는 방법, 가장 일반적인 문제점에 대한 솔루션 및 DB2 데이터베이스 제품에서 발생할 수 있는 문제점을 해결하는 방법 등에 관한 정보가 있습니다.

DB2 기술 지원 웹 사이트

문제점이 발생한 경우 해당 원인 및 솔루션을 찾으려면 DB2 기술 지원 웹 사이트를 참조하십시오. 기술 지원 사이트에는 최신 DB2 서적, 기술 노트, APAR(Authorized Program Analysis Report 또는 버그 수정), FixPack 및 기타 자원에 대한 링크가 있습니다. 이러한 기술 자료를 검색하여 문제에 대해 사용 가능한 솔루션을 찾을 수 있습니다.

다음은 DB2 기술 지원 웹 사이트의 주소입니다. http://www.ibm.com/software/data/db2/support/db2_9/

이용약관

다음 조건에 따라 이 책을 사용할 수 있습니다.

개인적 사용: 모든 소유권 사항을 표시하는 경우에 한하여 귀하는 이 책을 개인적, 비상업적 용도로 복제할 수 있습니다. IBM의 명시적인 동의 없이는 이 책 또는 그 일부를 배포 또는 전시하거나 2차적 저작물을 만들 수 없습니다.

상업적 사용: 모든 소유권 사항을 표시하는 경우에 한하여 귀하는 이 책을 귀하 기업 집단 내에서만 복제, 배포 및 전시할 수 있습니다. 귀하는 IBM의 명시적 동의 없이 이 책의 2차적 저작물을 만들거나 이 책 또는 그 일부를 복제, 배포 또는 전시할 수 없습니다.

본 허가에서 명시적으로 부여된 경우를 제외하고, 이 책이나 이 책에 포함된 정보, 데이터, 소프트웨어 또는 기타 지적 재산권에 대한 어떠한 허가나 라이센스 또는 권한도 명시적 또는 묵시적으로 부여되지 않습니다.

IBM은 이 책의 사용이 IBM의 이익을 해친다고 판단되거나 위에서 언급된 지시사항이 준수되지 않는다고 판단하는 경우 언제든지 이 사이트에서 부여한 허가를 철회할 수 있습니다.

귀하는 미국 수출법 및 관련 규정을 포함하여 모든 적용 가능한 법률 및 규정을 철저히 준수하는 경우에만 본 정보를 다운로드, 송신 또는 재송신할 수 있습니다.

IBM은 이 책의 내용에 대해 어떠한 보증도 제공하지 않습니다. 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 (단 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증 없이 현 상태대로 제공합니다.

부록 B. 주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다. 비IBM 제품에 대한 정보는 이 책을 처음 발행할 때의 정보에 기초하고 있으며 변경될 수 있습니다.

IBM은 다른 국가에서 이 책에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 그러나 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

135-700
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터
전화번호: 080-023-8080

2바이트 문자 세트(DBCS) 정보에 관한 라이센스 문의는 한국 IBM 고객만족센터에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 Japan

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다. IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 (단, 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증없이 이 책을 『현상태대로』 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 변경된 사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통지 없이 언제든지 개선 및/또는 변경할 수 있습니다.

이 정보에서 언급되는 비IBM의 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

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

(i) 독자적으로 작성된 프로그램과 다른 프로그램(본 프로그램 포함) 간의 정보 교환 및 (ii) 교환된 정보의 상호 이용을 목적으로 본 프로그램에 관한 정보를 얻고자 하는 라이센스 사용자는 다음 주소로 문의하십시오.

135-700
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠. 주식회사
고객만족센터

이러한 정보는 해당 조건(예를 들면, 사용료 지불 등) 하에서 사용될 수 있습니다.

이 정보에 기술된 라이센스가 부여된 프로그램 및 프로그램에 대해 사용 가능한 모든 라이센스가 부여된 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

본 문서에 포함된 모든 성능 데이터는 제한된 환경에서 산출된 것입니다. 따라서 다른 운영 환경에서 얻어진 결과는 상당히 다를 수 있습니다. 일부 측정치는 개발 레벨 시스템에서 작성되었을 수 있으며, 따라서 이러한 측정치가 일반적으로 사용되고 있는 시스템에서도 동일하게 나타날 것이라고는 보증할 수 없습니다. 또한 일부 성능은 추정을 통해 추측되었을 수도 있으므로 실제 결과는 다를 수 있습니다. 이 책의 사용자는 해당 데이터를 본인의 특정 환경에서 검증해야 합니다.

비IBM 제품에 관한 정보는 해당 제품의 공급업체, 공개 자료 또는 다른 기타 범용 소스로부터 얻은 것입니다. IBM에서는 이러한 제품들을 테스트하지 않았으므로, 비IBM 제품과 관련된 성능의 정확성, 호환성 또는 기타 청구에 대해서는 확신할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

IBM이 제시하는 방향 또는 의도에 관한 모든 언급은 특별한 통지 없이 변경될 수 있습니다.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이들 예제에는 개념을 가능한 완벽하게 설명하기 위하여 개인, 회사, 상표 및 제품의 이름이 사용될 수 있습니다. 이들 이름은 모두 가공의 것이며 실제 기업의 이름 및 주소와 유사하더라도 이는 전적으로 우연입니다.

저작권 라이센스:

이 정보에는 여러 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원어로 된 샘플 응용프로그램이 들어 있습니다. 귀하는 이러한 샘플 프로그램의 작성 기준이 되는 운영 플랫폼의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 IBM에 추가 비용을 지불하지 않고 이들 샘플 프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다. 이러한 샘플 프로그램은 모든 조건하에서 완전히 테스트된 것은 아닙니다. 따라서 IBM은 이러한 프로그램의 신뢰성, 서비스 가능성 또는 기능을 보증하거나 진술하지 않습니다. 샘플 프로그램은 어떠한 보증없이 ″있는 그대로″ 제공됩니다. IBM은 샘플 프로그램의 사용으로 인해 발생하는 모든 손해에 대해 책임을 지지 않습니다.

이러한 샘플 프로그램 또는 파생 제품의 각 사본이나 일부에는 반드시 다음과 같은 저작권 표시가 포함되어야 합니다.

© (*귀하의 회사명*) (연도). 이 코드의 일부는 IBM Corp.의 샘플 프로그램에서 파생됩니다. © Copyright IBM Corp. *_enter* 연도_. All rights reserved.

상표

IBM, IBM 로고 및 ibm.com®은 여러 국가에 등록된 International Business Machines Corp.의 상표 또는 등록상표입니다. 기타 제품 및 서비스 이름은 IBM 또는 기타 회사의 상표입니다. 현재 IBM 상표 목록은 웹 “저작권 및 상표 정보”(www.ibm.com/legal/kr/copytrade.shtml)에 있습니다.

다음 용어는 기타 회사의 상표 또는 등록상표입니다.

- Linux는 미국 또는 기타 국가에서 사용되는 Linus Torvalds의 등록상표입니다.
- Java 및 모든 Java 기반 상표는 미국 또는 기타 국가에서 사용되는 Sun Microsystems, Inc.의 상표입니다.
- UNIX는 미국 또는 기타 국가에서 사용되는 The Open Group의 등록상표입니다.
- Intel®, Intel 로고, Intel Inside®, Intel Inside 로고, Intel® Centrino®, Intel Centrino 로고, Celeron®, Intel® Xeon®, Intel SpeedStep®, Itanium® 및 Pentium®은 미국 또는 기타 국가에서 사용되는 Intel Corporation의 상표 또는 등록상표입니다.
- Microsoft, Windows, Windows NT® 및 Windows 로고는 미국 또는 기타 국가에서 사용되는 Microsoft Corporation의 상표입니다.

기타 회사, 제품 및 서비스 이름은 해당 회사의 상표 또는 서비스표입니다.

색인

[가]

[나]

[다]

[라]

[마]

[바]

[사]

[아]

[자]

[차]

[카]

[타]

[파]

[하]

A

C

D

E

F

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

X

Z

[특수 문자]

SA30-3952-00

Spine information:

Linux, UNIX 및 Windows용 IBM DB2 9.7

문제점 해결 및 데이터베이스 성능 조정

IBM