멀티미디어 CE 기기를 위한 빠른 질의 복구 기법

(Fast Query Recovery for Multimedia CE Devices)

진 희 규 † 이 기 용 †† 우 경 구 ††

(Heegyu Jin) (Ki Yong Lee) (Kyoung-Gu Woo)

요 약 멀티미디어 CE 기기(Multimedia Consumer Electronics Device)란 MP3 플레이어, PMP (Personal Media Player), 디지털 카메라 등과 같이 멀티미디어 데이타를 저장, 재생, 생성하는데 사용되는 기기를 말한다. 대부분의 멀티미디어 CE 기기는 기기에 저장된 멀티미디어를 검색하고, 그 결과를 브라우징하는 기능을 제공한다. 멀티미디어 CE 기기의 요구 사항 중의 하나는 사용자가 멀티미디어의 검색 결과를 브라우징하는 도중 기기의 전원이 꺼지더라도, 전원이 켜지면 원래의 브라우징 화면이 그대로 복구되어야 한다는 것이다. 기존의 방법은 이를 위해, (1) 검색 질의를 재수행하고, (2) 커서를 원래 브라우징하던 검색 결과의 레코드 위치까지 다시 이동시키는 방법을 사용해왔다. 그러나 이 방법은 검색 결과의 레코드 수가 많아질수록 많은 시간이 걸릴 수 있다. 본 논문에서는 멀티미디어의 검색 결과를 브라우징하는 도중, 기기의 전원이 꺼졌다 켜지더라도 원래의 브라우징 화면을 즉각적으로 복구할 수 있도록 하는 방법을 제안한다. 제안하는 방법은 검색 질의에 대한 질의 수행 계획의 일부 정보를 저장함으로써, 원래의 브라우징 화면을 즉각적으로 복구할 수 있도록 해준다. 실험 결과를 통해, 제안하는 방법은 검색 결과의 개수나 커서의 위치에 관계 없이 항상 일정한 성능을 제공함을 보인다.

키워드 : 질의 복구, 멀티미디어 검색

Abstract Multimedia consumer electronics (CE) devices, such as MP3 players, PMPs, and digital cameras, are electronic equipments used to record, play or create multimedia data. Most multimedia CE devices provide uses with the ability to search multimedia stored in the device and browse the search results. One of the unique requirements in multimedia CE devices is that the search results displayed in the screen must be restored quickly when the device powers off and later back on. For this purpose, the existing methods (1) re-execute the original search query, and (2) move the cursor to the original position in the search results. However, this approach may be inefficient when the number of records in the result set is large. In this paper, we propose an efficient method for multimedia CE devices that can quickly restore the search results displayed in the screen when the device powers off and later back on. The proposed method can retrieve the original search results in the screen quickly by saving and loading some information about the query evaluation plan. Though the performance evaluation, we show that the proposed method provides excellent performance regardless of the number of records in the result set or the original cursor position.

Key words : Query Recovery, Multimedia search

† 정 회 원 : 삼성전자 S/W 연구소 선임연구원
heegyu.jin@samsung.com
†† 정 회 원 : 삼성전자 S/W 연구소 책임연구원
kiyong.lee@gmail.com
kg.woo@samsung.com

논문접수 : 2007년 10월 24일
심사완료 : 2008년 2월 14일

정보과학회논문지: 데이타베이스 제35권 제3호(2008.6)

1. 서 론

소비자 가전(Consumer Electronics, CE) 기기란 통신, 오락, 정보 검색 등을 위해 다수의 소비자들에 의해 사용되는 전자 기기를 의미한다. 그 중에서 멀티미디어 CE 기기란 MP3 플레이어, PMP(Personal Media Player), 디지털 카메라, 캠코더 등과 같이 멀티미디어 데이타를 저장, 재생, 생성하는 데 사용되는 기기를 가리킨다. 대부분의 멀티미디어 CE 기기는 기기 내에 저장된 멀티미디어 데이타를 검색하고, 그 결과를 브라우징하는 기능을 제공한다. 멀티미디어 CE 기기들은 멀티미디어를 효율적으로 검색하기 위해 주로 메타데이타 기반의 검색 기법을 사용한다. 메타데이타 기반의 검색 기법이란 멀티미디어에 대한 메타데이타를 통하여 멀티미디어를 검색하는 기법이다. 예를 들어, MP3 플레이어의 경우 MP3 파일로부터 ID3 태그와 같이 가수, 앨범명, 장르, 길이 등을 포함하는 메타데이타를 추출한 후, 이 메타데이타를 통하여 MP3 파일을 검색한다.

최근 멀티미디어 CE 기기의 저장 용량이 증대됨에 따라, 멀티미디어 CE 기기에서도 내장형 데이타베이스(embedded database)를 사용하는 경우가 점차 늘고 있다. 이 경우 사용자는 데이타베이스에 질의를 던져 멀티미디어를 검색하게 된다. 그림 1은 MP3 플레이어에서 Artist가 "Beatles"인 MP3 파일을 제목으로 정렬된 순서에 따라 검색하는 예제이다. 그림 1에서 MP3 파일에 대한 메타데이타는 MP3Files라는 테이블에 저장되어 있다. 사용자가 메뉴에서 Artist를 선택한 후 Artist들의 목록이 나타나는 화면에서 "Beatles"를 선택하면, 데이타베이스는 그림 1의 SQL과 같은 질의를 수행하고 검색 결과에 대한 커서를 생성한다. 커서는 검색 결과 집합 내의 어떤 한 레코드를 가리키는 포인터를 말한다[1]. 그림 1은 Artist가 "Beatles"인 MP3 파일의 검색 결과가 500개일 때, 사용자가 커서를 이동하여 이 중 10번째에서 19번째에 해당하는 레코드를 보고 있는 경우를 나타낸다. 일반적으로 멀티미디어 CE 기기에서 화면의 크기는 제한되어 있으므로, 사용자는 커서를 위아래로 이동시켜 전체 검색 결과를 브라우징하게 된다.

일반적으로 멀티미디어 CE 기기는 전원이 꺼지고 켜지는 일이 빈번하게 발생한다. 이로 인해 멀티미디어 CE 기기에서 특징적으로 요구되는 사항은 다음과 같다.

- 기기의 전원이 꺼졌다가 다시 켜지는 경우, 사용자가 브라우징하고 있던 원래의 검색 화면이 그대로 복구되어야 한다.
- 기기의 응답 시간이 매우 빨라야 한다. 즉, 사용자가 기기의 멈춤 현상을 가능한 느끼지 않도록 해야 한다.

예를 들어, 그림 1에서 사용자가 500개의 검색 결과 중 10번째부터 19번째의 레코드를 브라우징하고 있었다면, 기기의 전원이 꺼졌다가 다시 켜지더라도 해당 화면이 그대로 복원되어야 한다. 또한, 사용자가 기기의 멈춤 현상을 느끼지 못하도록 화면의 복원이 가능한 빨리 처리되어야 한다. 이러한 요구 사항은 멀티미디어 CE 기기에서 흔히 요구되는 기능이다.

기존의 방법은 기기의 전원이 꺼졌다가 다시 켜지는 경우 해당 화면을 다시 복구하기 위해, (1) 검색 질의를 다시 수행하고, (2) 커서를 원래 브라우징하던 검색 결과의 레코드 위치까지 다시 이동시키는 방법을 사용해

Table	MP3Files = (Id, Title, Artist, Album, Genre, Length)
Query	DECLARE c CURSOR FOR SELECT Id, Title, Artist FROM MP3Files WHERE Artist = "Beatles" ORDER BY Title

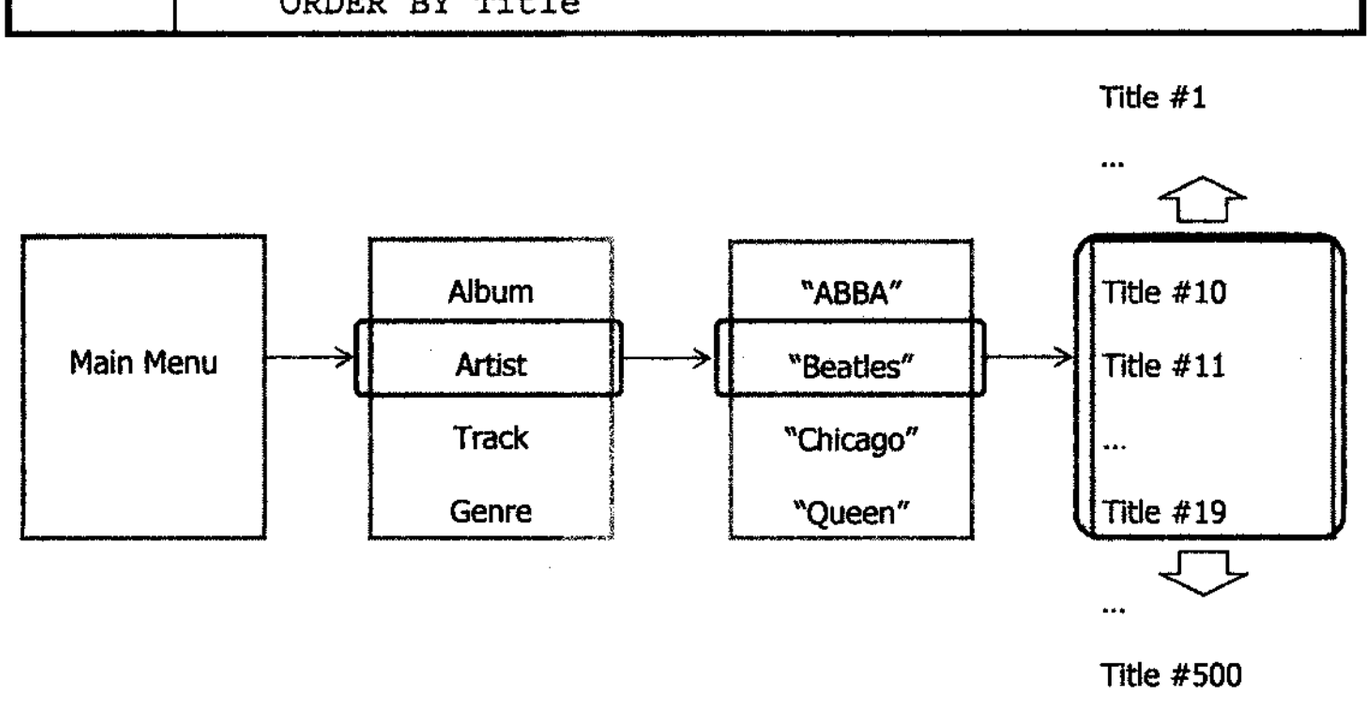


그림 1 MP3 플레이어에서의 멀티미디어 검색 및 브라우징의 예

왔다. 그러나 이러한 방법은 검색 결과의 크기가 매우 크거나 커서가 검색 결과의 매우 뒤 부분에 위치하는 경우, 커서를 원래 위치까지 이동시키는 데 시간이 많이 걸릴 수 있다. 이렇게 화면을 복구하는 시간이 길어지게 되면 사용자에게 큰 불편을 끼치게 된다. 본 논문에서는 사용자가 데이타베이스 질의에 대한 검색 결과를 브라우징하던 도중 시스템이 꺼졌다가 다시 복구된 경우, 원래 브라우징하던 검색 결과 화면을 즉각적으로 복구하는 방법을 제안한다. 제안하는 방법은 검색 질의에 대한 질의 수행 계획(Query Execution Plan, QEP)의 일부 정보를 저장함으로써, 추후에 이 정보를 사용하여 커서의 원래 위치 및 검색 화면을 복구할 수 있도록 한다. 따라서 제안하는 방법은 질의를 재수행하고 커서를 레코드 단위로 하나씩 이동하여 원래의 위치로 이동시키는 작업을 수행하지 않고서도, 거의 즉각적으로 원래의 브라우징 화면을 다시 복구할 수 있다. 사용자는 복구된 화면에서 커서를 위아래로 이동시켜 다른 검색 결과를 다시 자유롭게 브라우징할 수 있다. 또한 제안하는 방법은 검색 결과의 크기나 커서의 원래 위치에 상관없이 항상 일정한 시간 내의 응답 시간을 보장한다.

본 논문의 구성은 다음과 같다. 2장에서는 데이타베이스에서의 질의 처리 방법을 간략히 설명하고, 관련 연구를 기술한다. 3장에서는 본 연구의 동기를 언급하고, 4장에서 제안하는 방법을 자세히 기술한다. 5장에서는 실험 평가 결과를 보여주고, 6장에서는 결론을 맺는다.

2. 예비 지식 및 관련 연구

본 장에서는 데이타베이스에서의 일반적인 질의 처리 방식에 대해 간략히 알아보고, 본 연구의 관련 연구를 기술한다.

2.1 질의 처리 방식

일반적으로 데이타베이스에서 질의 수행 계획은 그림 2와 같이 질의 수행에 필요한 연산자들을 노드로 하는 트리로 표현된다. 질의 수행 계획을 수행하여 질의 결과를 얻어내는 질의 수행 단계는 크게 다음의 두 가지 방식을 사용한다[2].

- 실체화(Materialization) 방식: 질의 수행 계획에 포함된 각 노드의 계산 결과를 임시 테이블에 저장하고, 이를 사용하여 상위 레벨 노드들을 계산하는 방법이다. 예를 들어 그림 2의 질의 수행 계획을 실체화 방식을 사용하여 계산하는 경우, 먼저 MusicTable 테이블에서 Album = “Beatles”인 레코드를 찾아서 임시 테이블에 저장한다. 그 후 이 임시 테이블과 ContentTable 테이블을 조인하여 최종 결과를 얻는다. 이 방식은 질의 수행 과정에서 임시 테이블을 생성하기 때문에, 임시 테이블을 디스크에 쓰는 비용이 발생한다. 또한 임시 테이블의 결과를 완전히 저장하기 전까지 그를 입력으로 하는 연산자의 계산을 수행할 수 없다. 따라서, 질의 수행 계획에서 루트 노드 이하의 모든 연산자들의 계산이 완전히 끝나기 전까지는 질의의 최종 결과를 하나도 얻지 못한다는 단점이 있다. 따라서 질의의 응답 시간이 길어지게 된다.
- 파이프라이닝(Pipelining) 방식: 파이프라이닝 방식은 질의 수행 계획 내의 여러 연산자들을 파이프라인으로 연결하여, 중간 계산 결과를 임시 테이블로 저장하지 않고 파이프라인으로 연결된 다음 연산자로 바로 전달하는 방식이다. 파이프라인으로 연결된 각 연산자들은 자신의 입력으로 연결된 연산자로부터 결과 레코드를 받으면, 자신의 연산을 수행하여 결과 레코드를 생성하고, 이를 자신이 입력이 되는 연산자로 바로 전달한다. 예를 들어 그림 2의 질의 수행 계획을 파이프라이닝 방식을 사용하여 계산하는 경우, MusicTable 테이블을 스캔하여 Album = “Beatle”인 레코드가 찾아지는 대로 이를 임시 테이블에 저장하지 않고 바로 상위 조인 연산자로 전달한다. 조인 연산자는 레코드를 전달받는 대로 이 레코드와 ContentTable과의 조인을 수행하여 그 결과를 임시 테이블에 저장하지 않고 바로 상위 프로젝션 연산자로 전달한다. 프로젝션 연산자는 전달받은 레코드에 대해 바로 프로젝션 연산을 수행하여 최종 질의 결과를 생성한다. 파이프라이닝 방식은 질의 수행 과정에서 임시 테이블을 생성하지 않기 때문에, 임시 테이블을 디스크에 쓰는 비용이 없으며 임시 테이블을 저장하는데 필요한 별도의 메모리가 필요 없다는 장점이 있다. 또한, 파이프라이닝 방식은 질의 수행 계획에서 루트 노드 아래의 모든 연산자들의 계산이 완전히 끝나지 않은 상태에서도 질의의 최종 결과를 부분적으로 반환할 수 있기 때문에, 질의에 대한 응답 시간이 빨라진다는 장점이 있다. 따라서 이 방법은 응답 시간이 빨라야 하고, 디스크의 쓰기 비용이 크며, 메모리의 양이 제한된 모

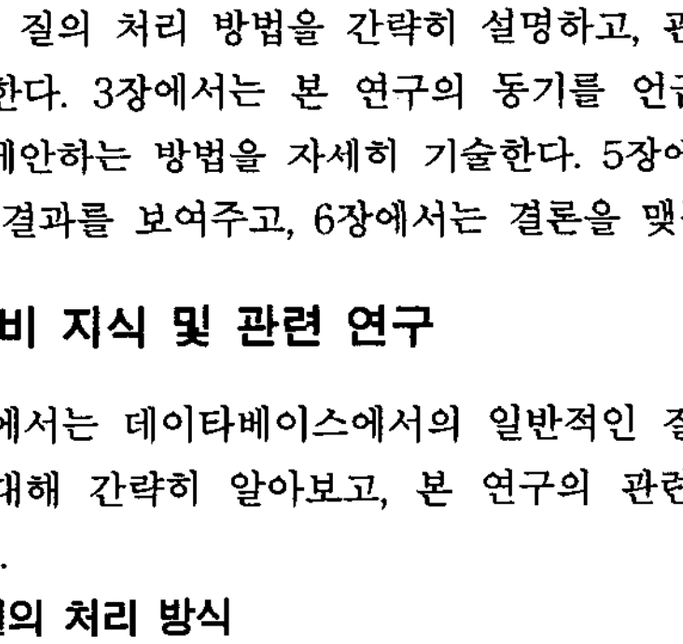


그림 2 질의 수행 계획의 예

바일 디바이스에서 많이 쓰이는 방법이다[3-5]. 하지만 정렬 연산자 등과 같이 질의 수행 계획에 포함된 모든 연산자가 항상 파이프라이닝 방식으로 처리될 수 있는 것은 아니다.

2.2 순환자(iterator) 모델

2.1절에서 설명한 파이프라이닝 방식은 대부분 요구 기반(demand driven) 방식으로 구현된다. 요구 기반 방식의 파이프라이닝은 질의 결과를 얻기 위해 질의 수행 계획의 루트 노드 연산자에게 반복적으로 요청을 보낸다. 요청을 받은 루트 노드 연산자는 다시 그의 하위 레벨 연산자에게 입력 레코드를 얻기 위한 요청을 보낸다. 요청을 받은 하위 레벨 연산자는 다시 그의 하위 레벨 연산자에게 입력 레코드를 얻기 위한 요청을 보낸다. 질의 수행 계획의 단말 노드에 해당하는 연산자들은 요청을 받으면 테이블 또는 인덱스로부터 레코드를 하나씩 읽어 그의 상위 레벨의 연산자에게 반환한다. 입력으로 연결된 하위 레벨 연산자로부터 레코드를 전달받은 상위 레벨 연산자는, 자신의 연산을 수행하여 결과 레코드를 생성한 뒤 이를 자신의 상위 레벨 연산자에게 반환한다. 이러한 방법을 통해 질의 수행 계획의 루트 연산자는 최종 결과 레코드를 하나씩 반환한다. 루트 연산자에게 전달될 입력 레코드가 더 이상 존재하지 않으면 질의 수행은 종료된다.

요구 기반 파이프라이닝 방식에서 질의 수행 계획의 각 연산자는 '순환자(iterator)'라는 이름으로 불린다. 일반적으로 각 순환자는 *Open()*, *Next()*, *Close()*의 세 가지 함수를 제공한다[6]. *Open()* 함수는 순환자에게 결과 레코드를 반환할 준비를 알린다. 그 후 순환자에 대해 *Next()* 함수를 호출할 때마다, 순환자는 다음 결과 레코드를 생성하여 이를 자신의 상위 레벨 순환자에게 반환한다. *Close()* 함수는 순환자에게 결과 레코드의 반환에 대한 요청이 끝났음을 알린다. 각 순환자는 *Next()*가 호출될 때마다 다음 결과 레코드를 올바르게 반환하기 위해서, 필요한 상태 정보들을 유지한다. 예를 들어 테이블의 레코드를 하나씩 읽어 반환하는 순환자의 경우, 자신이 최종적으로 반환한 레코드의 위치 정보를 유지함으로써 *Next()* 호출 시 그 다음 레코드를 올바르게 반환하도록 한다.

질의 수행 계획의 단말 노드에 해당하는 순환자로는 테이블 순환자와 인덱스 순환자가 있다. 테이블 순환자는 *Next()* 함수가 호출될 때마다 테이블의 레코드를 순차적으로 하나씩 읽어서 반환하는 순환자이며, 인덱스 순환자는 *Next()* 함수가 호출될 때마다 인덱스의 순서대로 레코드를 순차적으로 하나씩 읽어서 반환하는 순환자이다. B+ 트리를 사용하는 인덱스 순환자의 경우, 인덱스 순환자는 B+ 트리의 단말 노드를 순차적으로 순환하며 각 노드에서 가리키는 레코드를 순차적으로 반환한다. 이렇게 인덱스 순환자를 사용하는 경우 인덱스의 키 값으로 정렬된 순서에 따라 레코드를 얻을 수 있으므로, 인덱스 순환자는 파이프라이닝에서 정렬된 결과를 사용하고자 할 때 효과적으로 사용될 수 있다.

2.3 관련 연구

낮은 CPU 성능과 제한된 메모리를 가지고 있지만, 빠른 응답 시간을 제공해야 하는 모바일 디바이스에서는 주로 파이프라이닝 방식을 사용하여 질의를 처리한다. [3]은 모바일 디바이스 환경에서 가능한 파이프라이닝 방식으로 질의를 처리하는 방법에 대해 논의하였다. 해당 논문은 실체화가 필요한 연산자를 가능한 없애고, 파이프라이닝 방식으로 처리가 가능한 연산자만을 포함하는 질의 수행 계획을 생성하는 방법을 사용한다. 본 논문에서는 검색 질의는 기본적으로 파이프라이닝 방식으로 처리된다고 가정한다.

본 논문에서 다루는 문제인, 시스템이 다운되었다가 다시 복구된 경우 사용자가 브라우징하던 원래의 검색 화면을 다시 빠르게 복구하는 방법에 대한 연구는 거의 없었다. [7]에서는 많은 리소스와 시간이 드는 복잡한 질의를 수행할 때, 해당 질의의 수행을 일시적으로 중단시켰다가 추후에 다시 수행하는 방법에 대해 논의하였다. 해당 논문은 질의의 수행을 일시 중단시키기 전에 질의 수행에 대한 일부 정보들을 저장하고, 이를 이용하여 추후에 다시 질의를 이어 수행한다는 점에서 본 논문과 기본 개념이 유사하다. 그러나 본 논문에서는 복잡한 질의의 수행을 일시 정지하고 다시 수행하는 문제를 다루는 것이 아니라, 사용자가 브라우징하던 원래의 검색 화면을 다시 빠르게 복구하는 데 초점을 맞추고 있다. 더욱이 본 논문에서 다루는 문제는 사용자가 이동시킨 커서의 위치까지 고려해야 한다는 점에서 [7]과는 다르다. InfoSpace Inc.에서 개발한 데이타베이스의 기능 중에는 Automatic Query State Recovery라 하여, 질의의 상태를 저장하여 질의의 다음 결과를 빠르게 얻는 기능이 언급되어 있다[8]. 그러나 해당 기능에 대한 구체적인 내용이 포함된 문서는 찾아보기 어려웠다.

3. 연구 동기

MP3 플레이어, PMP, 캠코더와 같은 대부분의 멀티미디어 CE 기기는 제한된 크기의 화면만을 제공한다. 따라서 대부분의 멀티미디어 CE 기기에서는 멀티미디어에 대한 검색 결과를 한꺼번에 한 화면에 다 보여주는 대신, 사용자가 커서를 통해 화면을 위아래로 스크롤하면서 나머지 검색 결과를 브라우징하는 인터페이스를 제공한다. 이러한 인터페이스에서는 검색 질의에 대한 모든 결과를 얻지 않고서도 일부의 결과만으로 화면을

구성하는 것이 가능하다. 이러한 경우 멀티미디어 CE 기기는 파이프라이닝 방식으로 질의를 수행하여 빠른 응답 시간을 제공할 수 있다. 사용자가 나머지 검색 결과를 얻기 위해 커서를 이동하면, 그 때마다 필요한 개수만큼의 레코드를 추가로 얻어온다.

한편, 멀티미디어 CE 기기는 일반 PC와는 달리 전원이 꺼지고 켜지는 일이 빈번하게 발생한다. 이것은 멀티미디어 CE 기기의 고유한 특성이다. 이로 인해 멀티미디어 CE 기기에서는 기기의 전원이 꺼졌다가 다시 켜지는 경우에도, 사용자가 보고 있던 원래의 브라우징 화면이 그대로 복구되어야 하는 요구 사항이 존재한다. 예를 들어 1장의 그림 1에서, 사용자가 총 500개의 검색 결과 중 커서를 이동하여 491번째부터 500번째의 결과 레코드를 브라우징하고 있었을 경우, 기기의 전원이 꺼졌다가 다시 켜진 경우에도 해당 화면이 그대로 다시 복원되어야 한다. 이러한 경우에는 파이프라이닝 기법으로 질의를 처리하더라도 커서를 다시 해당 위치까지 이동시키는데 시간이 많이 걸릴 수 있게 된다. 이것은 커서를 다시 원래의 위치까지 이동시키기 위해서는 해당 위치까지의 레코드 개수만큼 커서의 이동이 발생해야 하기 때문이다. 최악의 경우 응답 시간은 검색 결과의 크기에 비례하여 커지게 된다. 이것은 기기의 사용성에 치명적인 피해를 주게 된다.

이러한 문제를 해결하기 위한 가장 쉬운 해결책 중의 하나는, 기기의 전원이 꺼지기 전에 화면에 출력되고 있는 레코드들을 따로 저장해 놓는 방식이다. 기기의 전원이 다시 들어오면, 저장된 레코드를 사용하여 화면을 다시 디스플레이한다. 하지만 이러한 방법도 사용자가 화면 외의 나머지 검색 결과를 브라우징하려고 하면, 화면 외의 레코드를 얻어오기 위해 질의를 재수행하고, 커서를 해당 위치로 이동시키는 작업이 필요하다.

본 논문에서 제안하는 방법은 필요할 때 마다 질의 수행 계획의 일부 정보를 저장함으로써, 이를 사용하여 원래의 브라우징 화면을 빠르게 복구할 수 있도록 한다. 저장되는 정보에는 질의 수행 계획에 포함된 일부 순환자에 대한 내부 상태(local state)가 포함된다. 기기의 전원이 다시 켜지면 제안하는 방법은 이 정보를 사용하여 질의 수행 계획에 포함된 모든 순환자의 내부 상태를 저장 시점의 상태로 복구한다. 그 후, 제안하는 방법은 복구된 질의 수행 계획을 사용하여 필요한 수만큼의 결과 레코드를 얻어 다시 원래의 화면을 복구한다. 제안하는 방법은 검색 결과에 포함된 레코드의 수 또는 커서의 원래 위치에 관계 없이 항상 일정 수준 이하의 응답 시간을 보장할 수 있다. 또한 원래의 질의 수행 상태를 복구하는 것이기 때문에, 사용자가 화면 외의 나머지 검색 결과도 커서를 이동하여 자유롭게 브라우징할 수 있다. 다음 장에서 제안하는 방법을 자세히 설명한다.

4. 제안 방법

4.1 기본 개념

2장에서 언급한 바와 같이 멀티미디어 CE 기기에서는 빠른 응답 속도를 보장하기 위해 많은 수의 검색 질의를 파이프라이닝 방식으로 처리한다. 본 논문에서는 검색 질의는 기본적으로 파이프라이닝 방식으로 처리된다고 가정한다. 그러나 본 논문에서 제안하는 방식은 질의 수행 계획에 포함된 연산자의 일부가 실체화 방식으로 수행되는 경우에도 적용될 수 있다.

파이프라이닝 방식으로 처리되는 질의에 대해, 어느 한 순간의 질의 수행 상태는 질의 수행 계획에 포함된 각 순환자들의 내부 상태로 정의될 수 있다. 2.2절에서 설명한 바와 같이 각 순환자들은 *Next()* 함수가 호출되었을 때 다음 결과 레코드를 반환하기 위하여 자신의 내부 상태를 유지하고 있음을 상기하라.

정의 1. 순환자의 *내부 상태*란 *Next()* 함수 호출 시 다음 결과 레코드를 올바르게 반환하기 위해 해당 순환자가 내부적으로 유지하는 정보의 어느 한 순간의 값을 말한다.

정의 2. 요구 기반 방식의 파이프라이닝으로 수행되는 질의에 대해 *질의 수행 상태*란, 질의 수행 계획에 포함된 순환자들의 내부 상태의 집합이다.

본 논문에서는 검색 질의의 질의 수행 상태를 저장함으로써, 기기의 전원이 꺼졌다가 다시 켜진 경우에도 원래의 질의 수행 상태를 빠르게 복구할 수 있도록 한다. 질의 수행 상태는 사용자가 전원 버튼을 누를 때나, 커서의 이동에 따라 검색 화면이 바뀔 때, 혹은 주기적인 방식으로 저장될 수 있다. 기기의 전원이 다시 켜지면 이 정보를 사용하여 질의 수행 계획 및 질의 수행 상태를 원래 상태로 복원하고, 복원된 질의 수행 계획을 사용하여 화면을 구성했던 결과 레코드들을 바로 다시 얻어온다. 따라서, 질의를 재수행하거나 커서를 원래 위치로 다시 이동시키는 작업을 하지 않고서도 질의 수행 상태를 저장한 시점에 화면을 구성했던 결과 레코드들을 즉각적으로 얻어올 수 있다. 제안하는 방법은 질의 상태 저장 단계와 질의 상태 복구 단계의 2단계로 구성된다.

- 질의 상태 저장 단계

  질의 상태 저장 단계는 기기의 전원이 꺼지기 직전이나, 커서의 이동에 따라 검색 화면이 바뀌는 시점, 또는 주기적으로 수행될 수 있다. 질의 상태 저장 요청이 발생하면, 질의 수행 계획에 포함된 순환자들 중 단말 노드에 해당하는 순환자들의 내부 정보를 디스크에 저장한다. 단말 노드에 해당하는 순환자들은 테

이블 또는 인덱스로부터 현재 몇 번째 레코드까지 읽어 들였는가에 대한 정보를 내부 정보로 유지한다.

- 질의 상태 복구 단계

  질의 상태 복구 단계는 기기의 전원이 켜져서 다시 원래의 브라우징 화면을 복구할 때 수행된다. 질의 상태 복구 단계에서는 저장된 질의 수행 상태 정보를 사용하여 질의 수행 계획과 그의 각 단말 노드에 해당하는 순환자들의 내부 상태를 복구한다. 그 후, 복원된 질의 수행 계획의 루트 연산자에 대해 *Next()* 함수를 호출하여 화면을 구성하는데 필요한 수만큼의 결과 레코드를 얻어낸다. 단말 노드에 해당하는 순환자 외에 다른 순환자들의 내부 상태는 *Next()* 함수의 수행 결과로 자동적으로 원래 상태로 복구된다.

4.2 질의 상태 저장 단계

질의 상태 저장 단계에서는 질의 수행 계획의 단말 노드에 해당하는 순환자들의 내부 정보를 저장한다. 제안하는 방법은 질의 수행 계획에 포함된 모든 순환자들의 내부 정보를 저장하는 대신 단말 노드에 해당하는 순환자들의 내부 정보만을 저장함으로써 질의 상태 저장의 비용을 크게 줄인다. 질의 수행 계획의 단말 노드로는 테이블 순환자와 인덱스 순환자 두 종류의 순환자만이 올 수 있다.

정의 3. 요구 기반 방식의 파이프라이닝으로 수행되는 질의 수행 계획에서, *테이블 순환자*란 테이블의 레코드를 순차적으로 읽어 반환하는 순환자를, *인덱스 순환자*란 인덱스가 가리키는 순서대로 레코드를 순차적으로 읽어 반환하는 순환자를 말한다.

질의 수행 계획에서 단말 노드 외에 다른 순환자들에 대한 내부 상태는, 해당 순환자에 대해 *Next()* 함수를 호출하면 단말 노드 순환자의 내부 상태로부터 자동으로 복구된다. 위 두 종류의 순환자에 대해 본 논문에서는 다음과 같은 내부 상태를 저장한다.

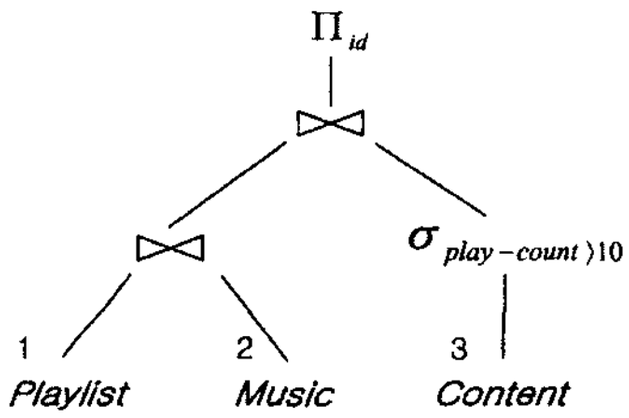
테이블 순환자는 현재 테이블 순환자가 가리키고 있는 레코드의 RID를 유지하다가, *Next()* 함수가 호출되면 현재 가리키고 있는 레코드를 반환한 뒤, 테이블에서 그 다음 레코드를 찾아 그의 RID를 저장한다. 이와 유사하게 인덱스 순환자는 현재 인덱스 순환자가 가리키고 있는 레코드의 RID 및 해당 레코드를 가져오는데 사용한 인덱스의 키 값을 유지하다가, *Next()* 함수가 호출되면 현재 가리키고 있는 레코드를 반환한 뒤, 인덱스에서 그 다음 레코드를 찾아 그의 RID와 그를 가리키는데 사용된 인덱스의 키 값을 저장한다. 이렇게 단말 순환자들에 대해 저장된 정보는 추후 질의 상태 복구 단계에서 각 순환자들이 가리키고 있던 레코드에 대한 정보를 다시 복구하는데 사용된다.

순환자	저장 정보
테이블 순환자	현재 가리키고 있는 레코드 ID (RID)
인덱스 순환자	현재 레코드를 가리키는데 사용된 인덱스의 키 값 현재 가리키고 있는 레코드 ID (RID)

그림 3 단말 순환자에 대해 저장되는 정보

질의 상태 저장 요청이 들어오면, 제안하는 방법은 주어진 질의 수행 계획을 탐색하여 단말 순환자들의 내부 상태를 저장한다. 이 때, 질의 수행 계획을 탐색하는 데는 깊이 우선 탐색(depth-first search) 또는 너비 우선 탐색(breadth-first search) 등과 같은 일반적인 탐색 방법을 사용할 수 있다. 그림 4는 깊이 우선 탐색 방법을 사용하여 질의 상태를 저장하는 예를 보여준다. 그림 4(a)의 질의 수행 계획에서 단말 순환자 위에 표시된 숫자는 방문 순서를 의미한다. 제안하는 방법은 그림 4(a)에 포함된 3개의 단말 순환자에 대해 그림 4(b)와 같은 내부 정보를 저장한다.

그러나 2.1절에서 언급한 바와 같이, 질의 수행 계획 내의 모든 연산자들을 항상 파이프라이닝 방식으로 수행할 수 있는 것은 아니다. 즉, 질의 수행 계획 내의 연산자 중 일부는 실체화 방식으로 수행해야 하는 경우가 있다. 이러한 연산자의 대표적인 경우는 정렬 연산자이다. 예를 들어, 그림 5와 같이 Order-by와 같은 정렬 연산자를 포함하는 경우는 질의를 파이프라이닝 방식으로 수행할 수 없다. 이 경우 만약 Artist를 키 값으로 하여 정렬된 순서를 유지하는 인덱스가 존재한다면, 정렬 연산자를 이 인덱스에 대한 인덱스 순환자로 대체하



(a)

No.	순환자	저장 정보
1	Playlist	현재 Playlist 내 레코드 RID
2	Music	현재 Music 내 레코드 RID
3	Content	현재 Content 내 레코드 RID

(b)

그림 4 질의 상태 저장의 예

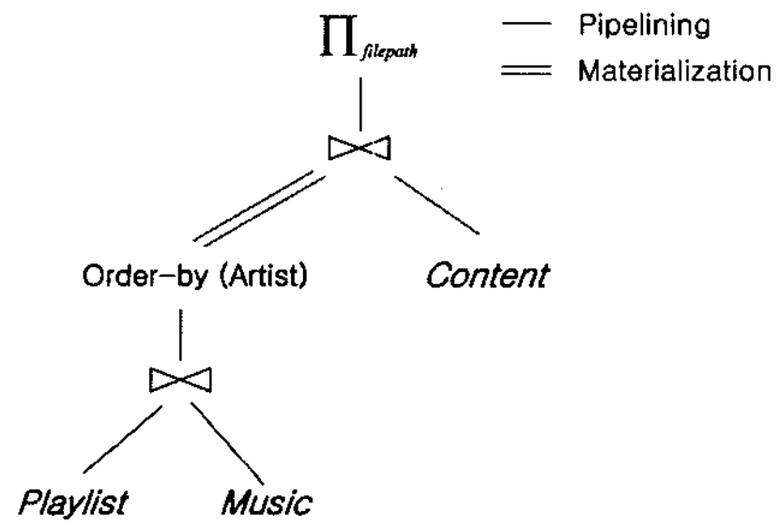
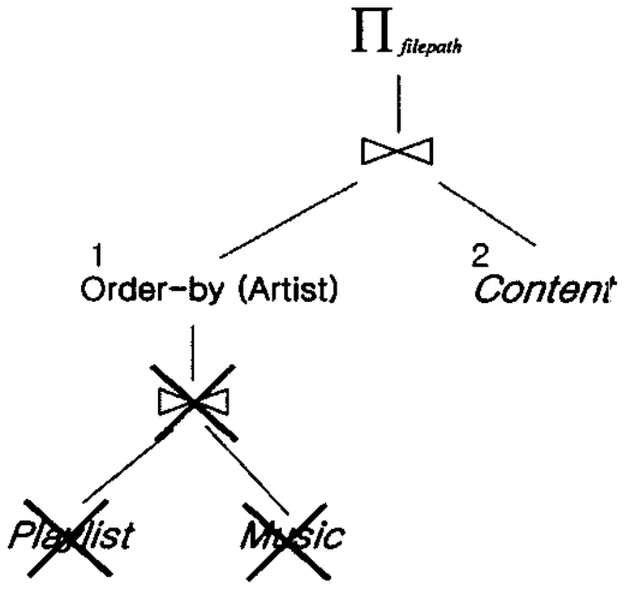


그림 5 정렬 연산자가 포함된 질의 수행 계획의 예

여 파이프라이닝 방식으로 질의를 수행하는 것이 가능하다. 하지만 이러한 인덱스도 존재하지 않을 경우에는 실체화 방법을 사용할 수 밖에 없다. 따라서, 이렇게 질의 수행 계획에 실체화 방식으로 수행되어야 하는 연산자가 포함되어 있을 경우에는 별도의 처리 방법이 필요하다.

제안하는 방법은 질의 수행 계획 내에 실체화 방법으로 처리되는 연산자가 있을 경우, 해당 연산자에 대해 생성된 임시 테이블과 해당 연산자에 대한 순환자가 가리키고 있던 임시 테이블 내의 레코드 RID를 함께 저장한다. 이 때 임시 테이블은 이미 디스크에 쓰여진 내용이므로 이를 다시 저장할 필요는 없으며, 이미 생성된 임시 테이블을 유지만 하면 된다. 따라서, 제안하는 방법은 임시 테이블을 저장하기 위한 별도의 비용을 발생시키지 않는다. 이렇게 임시 테이블이 저장되는 경우에는, 해당 연산자의 모든 결과 레코드가 저장되는 것이므로 추후에 해당 연산자의 하위 레벨 연산자들은 다시 수행될 필요가 없다. 따라서, 결과 레코드를 포함하는 임시 테이블이 저장된 실체화 연산자가 있을 경우에는 그의 하위 레벨 연산자들에 대해서 내부 상태를 저장하거나 복구하지 않는다. 그림 7은 실체화 방식으로 처리되어야 하는 정렬 연산자가 포함된 그림 5의 질의 수행 계획에 대해 질의 상태를 저장하는 예를 보여준다. 그림



(a)

No.	순환자	저장 정보
1	Order-by	정렬 결과 정렬 결과 내 레코드 RID
2	Content	현재 Content 내 레코드 RID

(b)

그림 6 실체화 방식을 사용하는 연산자가 포함되어 있을 경우의 질의 상태 저장

```
Procedure SaveQueryState(S, QEP)
Input S: query string
      QEP: query evaluation plan

1: Save S to disk;
2: Let r = the root node of QEP;
3: SaveIteratorState(r);

End procedure
```

```
Procedure SaveIteratorState(N)
Input N: a node of QEP

1: if (N is a leaf node of QEP) then
2:     if (N is an index iterator) then
3:         Save (current RID, current key value) of N to disk;
4:     else if (N is a table iterator) then
5:         Save (current RID) of N to disk;
6: else if (N is a materialized node of QEP) then
7:         Save (temporary relation, current RID) of N to disk;
8: else
9:     for each child node n of N do
10:        SaveIteratorState(n);

End procedure
```

그림 7 질의 상태 저장 프로시저

6(a)에서 Order-by 연산자는 실체화 방식으로 처리되며, 그의 결과는 임시 테이블에 저장된다. 따라서, 그의 하위 레벨의 연산자들에 대해서는 내부 상태를 저장하지 않는다. 대신 Order-by 연산자에 대해서는 그의 결과를 담고 있는 임시 테이블과 해당 연산자에 대한 순환자가 가리키고 있던 임시 테이블 내의 레코드 RID를 저장한다. 그림 6(b)는 이러한 방법으로 저장된 질의 상태 정보를 나타낸다.

그림 7은 지금까지 설명한 질의 상태 저장 방법을 나타내는 프로시저이다. 그림 7의 SaveQueryState() 함수는 질의 상태 외에 질의 문자열도 같이 저장한다. 이 질의 문자열의 사용에 대해서는 4.3절에서 설명한다.

4.3 질의 상태 복구 단계

질의 상태 복구 단계에서는 질의 수행 계획을 재구성한 뒤, 저장된 질의 상태 정보를 이용하여 질의 수행 계획에 포함된 각 단말 순환자들의 내부 상태를 원래 상태로 복구한다. 제안하는 방법의 질의 상태 복구 단계에서는 저장된 질의 문자열을 사용하여 질의 최적화기

(optimizer)를 통해 질의 수행 계획을 재구성한다. 본 논문에서는 질의 상태가 저장된 시점과 질의 상태가 복구되는 시점의 데이타베이스 간에는 변화가 없다고 가정한다. 데이타베이스에 변화가 없는 경우 동일한 질의에 대해서는 항상 동일한 질의 수행 계획이 생성된다[9,10]. 따라서, 본 논문에서는 질의 상태 저장 단계에서 질의 수행 계획 자체를 저장하는 대신 질의 문자열만을 저장하고, 저장된 질의 문자열을 사용하여 원래의 질의 수행 계획을 재구성한다.

이렇게 질의 수행 계획이 재구성되면, 질의 상태 저장 단계에서와 동일한 탐색 방법을 사용하여 질의 수행 계획을 탐색하면서, 질의 상태가 저장된 단말 순환자들의 내부 상태를 원래 상태대로 복구한다. 질의 수행 계획에서 단말 순환자와 임시 결과가 저장된 실체화 연산자를 위한 순환자의 내부 상태가 복구되면, 질의 수행 계획 내의 다른 순환자들의 상태는 각 순환자에 대한 *Next()* 함수의 호출을 통해 다시 원래 상태로 복구될 수 있다. 그림 8은 질의 상태 복구 방법을 나타내는 프로시저이다.

이렇게 질의 수행 계획 및 각 순환자의 내부 상태가 원래대로 복구되면, 이로부터 한 화면을 구성하기 위해 필요한 수만큼의 레코드를 다시 얻어와서 원래의 검색 화면을 복구한다.

```
Procedure RestoreQueryState(S, I)
Input S: query string
      I: saved query state information

1: Construct QEP from S;
2: Let r = the root node of QEP;
3: RestoreIteratorState(r, I);

End procedure
```

```
Procedure RestoreIteratorState(N, I)
Input N: a node of QEP
      I: saved query state information

1: if (N is a leaf node of QEP) then
2:     if (N is an index iterator) then
3:         Restore (current RID, current key value) of N from I;
4:     else if (N is a table iterator) then
5:         Restore (current RID) of N from I;
6: else if (N is a materialized node of QEP) then
7:         Restore (temporary relation, current RID) of N from I;
8: else
9:     for each child node n of N do
10:        RestoreIteratorState(n, I);

End procedure
```

그림 8 질의 상태 복구 프로시저

보조 정리 1. RestoreQueryState()는 질의 수행 상태를 올바르게 복구한다.

증명. 그림 7의 SaveQueryState()에 의해 저장된 질의 수행 상태를 $P_1 = <p_1, p_2, \cdots, p_N>$이라 하자. N은 질의 수행 계획의 단말 노드의 개수이고, $p_i(1 \le i \le N)$는 질의 수행 계획에 포함된 각 단말 노드의 내부 상태를 나타낸다. RestoreQueryState()에 의해 복구되는 질의 수행 상태를 $P_2 = <q_1, q_2, \cdots, q_N>$라 하면, RestoreQueryState()는 SaveQueryState()와 동일한 순서대로 질의 수행 계획을 탐색하며 동일한 조건으로 노드의 내부 상태를 복구하므로, $q_i = p_i(1 \le i \le N)$가 된다. 따라서 $P_2 = P_1$이므로 증명이 완료된다.

4.4 고려 사항

제안하는 방법은 질의 상태를 저장할 때마다 질의 수행 계획을 탐색하여 일부 순환자의 내부 상태를 디스크에 써야 하는 비용이 발생한다. 이는 특히 플래시 메모리를 저장 장치로 사용하는 멀티미디어 CE 기기에서 많은 비용을 야기할 수 있다. 그러나 제안하는 방법에서 질의 상태 저장 단계는 검색 결과 레코드를 하나씩 얻을 때마다 매번 수행될 필요는 없으며, 다음의 경우에만 수행되면 된다.

- 커서의 이동에 따라 검색 화면이 바뀌는 시점
- 사용자가 전원 버튼을 눌러 기기의 전원이 꺼지기 직전

위와 같은 경우는 실제 그리 빈번하게 발생하는 일이 아니므로 제안하는 방법은 비교적 적은 비용으로 수행될 수 있다. 또한, 제안하는 방법은 질의 상태를 저장할 때마다 모든 순환자의 내부 상태를 디스크에 쓰는 대신, 단말 순환자와 실체화 방식으로 수행되는 순환자에 대한 내부 상태만을 저장하므로 질의 상태 저장 단계의 비용을 더욱 줄일 수 있다. 또한 질의 상태를 저장할 때마다 매번 질의 문자열을 저장하는 대신, 최초 1회만 질의 문자열을 저장하도록 하여 질의 상태 저장 비용을 더욱 줄일 수 있다.

제안하는 방법은 파이프라이닝 방식의 질의 수행에 기반을 두고 있다. 하지만, 4.3절에서 설명한 바와 같이 제안하는 방법은 실체화 방식으로 수행되는 연산자를 포함하는 질의에 대해서도 적용이 가능하다. 질의 수행 계획에 실체화 방식으로 처리되어야 하는 연산자가 존재하는 경우, 제안하는 방법은 해당 연산자에 대한 중간 결과를 담고 있는 임시 테이블을 유지해야 하는 오버헤드를 가진다. 그러나 임시 테이블은 이미 디스크에 쓰여진 내용이기 때문에 이를 유지하기 위한 추가적인 쓰기 비용은 발생하지 않는다.

제안하는 방법에서 질의 상태 저장 단계 수행 도중 기기의 전원이 꺼질 수도 있다. 이 경우, 일부 노드에 대한 내부 상태는 디스크에 저장 되고 일부 노드의 내부 상태는 디스크에 저장되지 못할 수도 있다. 이렇게 질의 상태가 완전하게 저장되지 못하면 이후 질의 상태 복구 단계에서 질의 상태가 올바르지 못한 상태로 복구될 수 있다. 예를 들어, 일부 노드는 최신의 상태로 복구되고 일부 노드는 올바르지 않은 상태로 복구되면 잘못된 검색 결과 레코드가 반환될 수 있다. 따라서 이와 같은 경우를 막기 위해 질의 수행 상태를 디스크에 쓰

고 난 후, 질의 수행 상태가 모두 올바르게 쓰여졌음을 보장하기 위해 엔드 마크(end mark)나 체크섬(checksum)과 같은 정보를 부가적으로 저장할 수 있다. 질의 상태 복구 단계에서는 이와 같은 정보를 확인한 후, 질의 수행 상태가 올바르게 저장되었음이 확인되면 저장된 정보를 사용하여 질의 상태를 복구하고, 그렇지 못하면 기존의 방법을 사용한다.

5. 성능 평가

본 장에서는 실제 상용으로 사용되는 멀티미디어 CE 기기 환경에서, 본 논문에서 제안하는 질의 상태 저장 방법의 비용을 측정하고, 제안하는 질의 상태 복구 방법과 기존의 방법을 비교한 실험 결과를 기술한다. 실험은 200MHz의 ARM-core CPU와 32MB의 DRAM을 장착한 상용 MP3 플레이어에서 수행되었다. 해당 MP3 플레이어는 Linux 기반 플랫폼을 사용하고 있으며, 8GB 플래시 메모리를 저장 매체로 사용한다. 그림 9는 질의 상태 저장과 복구 성능을 측정하기 위해 실험에 사용한 테이블 스키마 및 질의를 나타내고, 그림 10은 그림 9의 질의에 대한 질의 수행 계획을 나타낸다.

Content 테이블에는 총 10,000개의 레코드가 삽입되었으며, 그림 9의 질의를 수행한 결과 총 1,000개의 결과 레코드가 검색되었다. 실험에서는 총 1,000개의 결과 레코드들 중, 기기의 전원이 꺼지기 전에 커서가 100번째 레코드에 위치해 있었을 때부터 1,000번째 레코드에 위치해 있었을 때까지, 커서의 위치를 100개 레코드 단위로 변경해가면서 제안하는 방법의 성능을 측정하였다. 그림 11은 제안하는 방법을 사용하여 질의 상태를 저장하고, 복구하는 데 걸리는 시간을 측정한 결과이다.

Table	Content (id, content-name, filepath, play-count) Playlist (id, playlist-name)
Query	SELECT content-name FROM Content, Playlist WHERE Content.id = Playlist.id AND play-count > 0

그림 9 실험에 사용된 테이블 스키마 및 질의

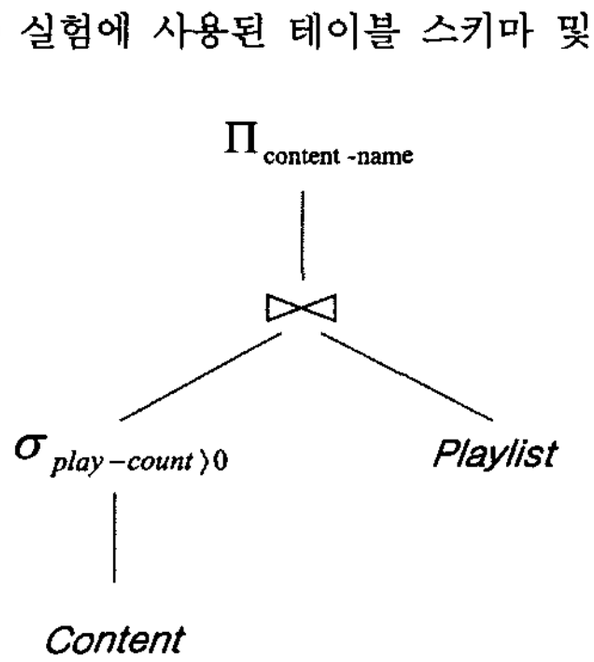


그림 10 그림 9의 질의에 대한 질의 수행 계획

4.2절에서 설명한 바와 같이, 제안하는 방법의 질의 상태 저장 단계는 기기의 전원이 꺼질 때나 커서 이동에 따라 브라우징 화면이 변경될 때, 혹은 주기적으로 수행될 수 있다. 따라서 질의 상태 저장 단계는 어플리케이션의 성능에 영향을 미치므로 가능한 적은 비용으로 처리되어야 한다. 그림 9의 질의와 같이 MP3 플레이어에서 흔히 쓰이는 2개의 테이블을 조인하는 질의의 경우, 질의 상태 저장 단계에 걸리는 시간은 대략 2ms로 매우 적은 시간이 소요됨을 알 수 있다.

그림 12는 기기의 전원이 꺼졌다가 다시 켜진 경우, 기존의 방법과 제안하는 방법을 사용하여 화면을 복구하는데 필요한 검색 결과 레코드를 다시 가져오는데 걸린 시간을 비교한 결과이다. 기존의 방법은 질의 수행 계획을 재생성한 뒤, 원래의 검색 화면을 구성했던 레코드가 나타날 때까지 커서를 이동시키는 작업을 수행한다. 따라서, 기존의 방법은 커서의 위치가 검색 결과의 뒤에 있을수록 커서의 위치에 비례하여 성능이 저하된다. 반면에 제안하는 방법은 커서의 원래 위치에 관계없이 항상 일정한 성능을 유지함을 알 수 있다. 그림 12에서 제안하는 질의 상태 복구 방법은 커서의 원래 위치에 관계없이 약 3ms이내로 항상 일정한 성능이 유지되고 있지만, 기존의 방법은 0.2초에서 약 2초까지의 시간이 소요된다. 이러한 성능 차이는 검색 결과에 포함된 레코드의 수가 많을수록 더욱 커지게 된다. 따라서 제안하는 방법은 전원이 빈번하게 꺼졌다 켜지는 멀티미디

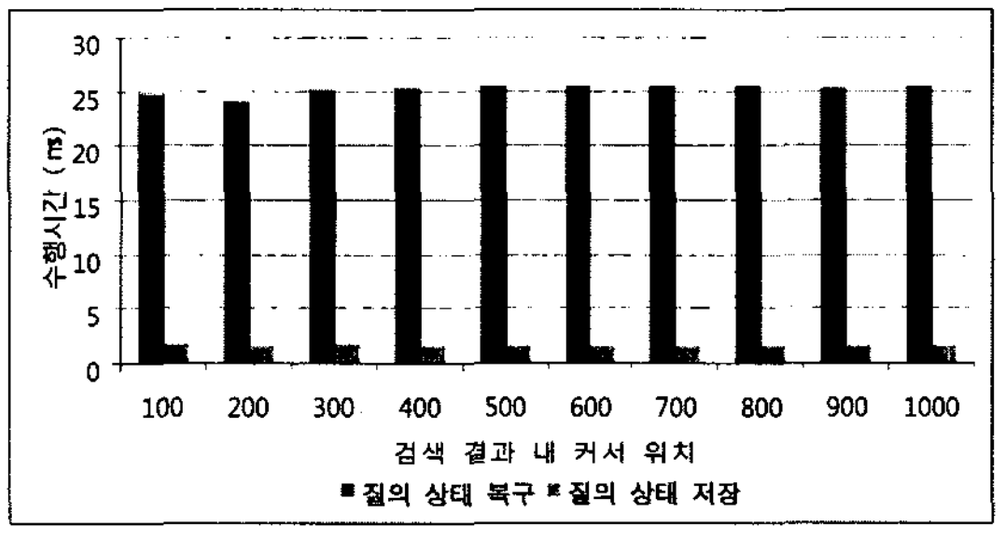


그림 11 제안하는 방법의 수행 시간

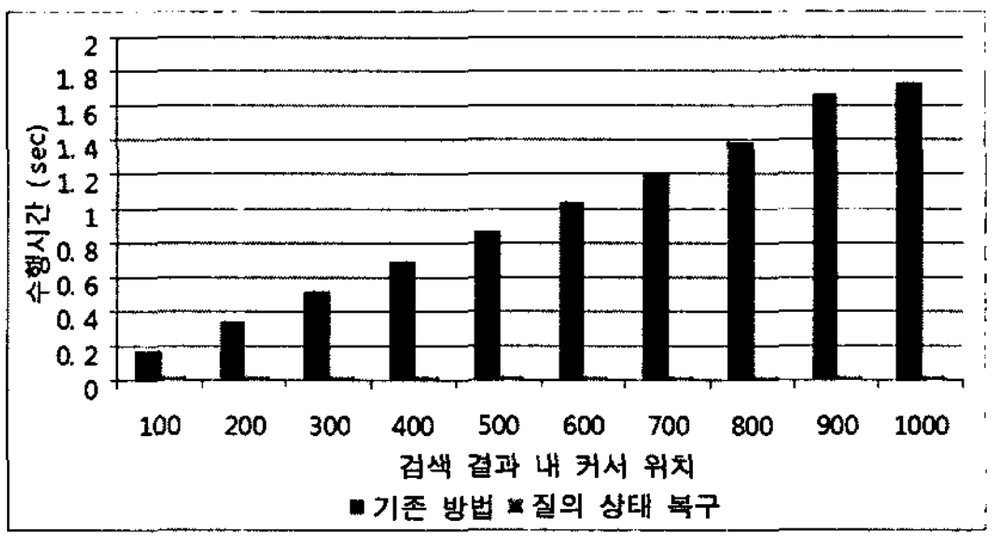


그림 12 기존 방법과 질의 상태 복구 방법의 성능 비교 결과

어 CE 기기 환경에서 실제 기기의 응답 시간을 줄이기 위해 매우 효과적으로 사용될 수 있음을 알 수 있다.

6. 결 론

멀티미디어 CE 기기는 전원이 꺼지고 켜지는 일이 빈번하게 발생한다. 이러한 환경에서는 기기의 전원이 꺼졌다가 다시 켜졌을 때, 사용자가 브라우징하던 검색 화면을 다시 원래대로 복구해야 하는 새로운 요구 사항이 존재한다. 특히, 사용자에 대한 응답 시간이 중요한 멀티미디어 CE 기기의 특성 상 검색 화면을 빠르게 복구하는 것은 매우 중요한 일이다. 이러한 요구 사항을 위해 기존의 방법은 질의를 재수행하고, 커서의 위치를 전원이 꺼지기 전에 사용자가 브라우징하던 검색 결과의 위치로 다시 이동시키는 방법을 사용해왔다. 하지만 이러한 방법은 검색 결과의 크기와 커서의 위치에 따라 성능이 악화되는 문제점을 가지고 있다. 본 논문에서는 원래의 검색 화면을 빠르게 복구하기 위하여, 질의 수행 계획의 일부 정보를 디스크에 저장하고 이를 사용하여 화면 구성에 필요한 검색 결과 레코드들을 빠르게 얻는 방법을 제안한다. 제안하는 방법은 커서를 원래 위치로 이동시키는 작업을 하지 않고도 원래 화면을 구성하던 레코드들을 즉시 얻을 수 있기 때문에, 검색 화면을 매우 빠르게 복구할 수 있다. 또한 제안하는 방법은 검색 결과의 크기나 커서의 원래 위치에 관계없이 항상 일정한 성능을 유지한다. 실제 MP3 플레이어 환경에서의 실험을 통해, 제안하는 방법은 기존 방법에 비해 수 십 배 이상의 성능을 낼 수 있음을 보였다.

참 고 문 헌

[1] A. Eisenberg, J. Melton, "SQL Standardization: The Next Steps," ACM SIGMOD Record, 29(1), pp. 63-67, 2000.

[2] Silberschatz A., Korth H., Sudarshan, S. "Database System Concepts," 4th edition, McGrawHill, 2001.

[3] Pucheral P., Bouganim L., Valduriez P., Bobineau C., "PicoDBMS: Scaling down Database Techniques for the Smartcard," Very Large Data Bases Journal (VLDBJ), 10(2-3), 2001.

[4] Anciaux N., Bouganim L., Pucheral P., "Memory Requirements for Query Execution in Highly Constrained Devices," Int. Conf. on Very Large Data Bases (VLDB), 2003.

[5] N. Anciaux, L. Bouganim, P. Pucheral, "On Finding a Memory Lower Bound for Query Evaluation in Lightweight Devices," Technical Report, PRiSM, 2003.

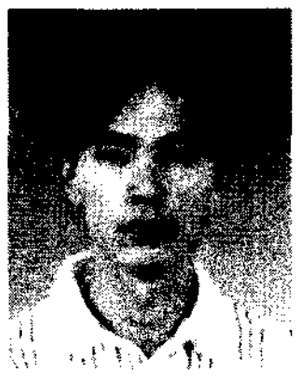
[6] G. Graefe, "Query Evaluation Techniques for Large Databases," ACM Computing Surveys, 25(2), 1993.

[7] B. Chandramouli, Christopher N. Bond, Shivnath B., J. Yang, "Query Suspend and Resume," ACM SIGMOD Conference, 2007.

[8] Infospace Inc., INSP Securities Registration Statement (S-1/A) Business, http:// sec.edgar-online.com/1999/10/04/17/0001032210-99-001396/Section8.asp

[9] R. Cole and G. Graefe, "Optimization of Dynamic Query Evaluation Plans," ACM SIGMOD Conference, 1994.

[10] S. Madden, M. Shah, J.M.Hellerstein, V. Raman, "Continuously Adaptive Continuous Queries," ACM SIGMOD Conference, 2000.



진 희 규

2002년 2월 부산대학교 컴퓨터공학과 학사. 2005년 2월 부산대학교 컴퓨터공학과 석사. 2005년 2월~현재 삼성전자 기술총괄 소프트웨어연구소 선임연구원. 관심분야는 시공간 DB, Embedded DB

이 기 용

1998년 2월 한국과학기술원 전산학과 학사. 2000년 2월 한국과학기술원 전산학과 석사. 2006년 2월 한국과학기술원 전자전산학과 전산학전공 박사. 2006년 3월~2008년 2월 삼성전자 기술총괄 소프트웨어연구소 책임연구원. 2008년 3월~현재 한국과학기술원 전산학과 연구조교수. 관심분야는 Data warehouse, OLAP, Embedded DB

우 경 구

1991년~1996년 서울대학교 컴퓨터공학과 학사. 1997년~1998년 한국과학기술원 전산학과 석사. 1998년~2004년 한국과학기술원 전산학과 전산학전공 박사. 2003년~현재 삼성전자 기술총괄 소프트웨어연구소 책임연구원. 기술적으로는 데이타 인덱싱 및 검색, 클러스터링 등의 데이타 마이닝 기술, Software Engineering 등에 관심이 많으며, 다양한 Problem Solving 과정 및 Team Building을 좋아한다.