Machine Translated by Google

13장: 데이터 저장 구조

데이터베이스 시스템 개념, 7 판.

©Silberschatz, Korth 및 Sudarshan 재사용
조건은 www.db-book.com 참조

파일 구성

§ 데이터베이스는 파일 모음으로 저장됩니다. 각 파일은 일련의 기록. 레코드는 일련의 필드입니다. § 하나의 접근법

- 레코드 크기가 고정되어 있다고 가정
- 각 파일에는 특정 유형의 레코드만 있습니다.
- 다른 파일은 다른 관계에 사용됩니다.

이 경우는 구현하기 가장 쉽습니다. 나중에 가변 길이 레코드를 고려할 것입니다.

§ 레코드가 디스크 블록보다 작다고 가정합니다.

```
type instructor = record
                    ID varchar (5);
                    name varchar(20);
                    dept_name varchar (20);
                    salary numeric (8,2);
                  end
```

Machine Translated by Google

고정 길이 레코드

§ 간단한 접근 방식: • 각 레

코드의 바이트 n부터 시작하여 레코드 i를 저장합니 * (i – 1), 여기서 n은 다음의 크기입니다.
다.

• 레코드 액세스는 간단하지만 레코드가 디스크 블록을 교차할 수 있음

§ 수정: 레코드가 블록 경계를 넘지 않도록 합니다.

§ 디스크 블록은 스토리지 할당 및 검색을 위한 논리적 단위입니다. • 일반적으로 4~16KB

§ 더 작은 블록: 디스크에서 더 많은 전송

§ 더 큰 블록: 부분적으로 채워진 블록으로 인해 낭비되는 더 많은 공간

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

©Silberschatz, Korth 및 Sudarshan

고정 길이 레코드

§ 기록 삭제 i: 대안:

• 이동 레코드 i + 1, . . ., n에서 i까지, . . . , n – 1

• 레코드 n을 i로 이동

• 레코드를 이동하지 않고 자유 목록의 모든 자유 레코드를 연결합니다.

레코드 3 삭제됨

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

©Silberschatz, Korth 및 Sudarshan

Machine Translated by Google

고정 길이 레코드

§ 기록 삭제 i: 대안:

- 이동 레코드 i + 1, . . ., n에서 i까지, . . . , n – 1
- 레코드 n을 i로 이동
- 레코드를 이동하지 않고 자유 목록의 모든 자유 레코드를 연결합니다.

레코드 3이 삭제되고 레코드 11로 교체됨

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

©Silberschatz, Korth 및 Sudarshan

고정 길이 레코드

§ 기록 삭제 i: 대안:

- 이동 레코드 i + 1, . . ., n에서 i까지, . . . , n – 1
- 레코드 n을 i로 이동
- 레코드를 이동하지 않고 자유 목록의 모든 자유 레코드를 연결합니다.

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

©Silberschatz, Korth 및 Sudarshan

Machine Translated by Google

가변 길이 레코드

§ 가변 길이 레코드는 여러 가지 방법으로 데이터베이스 시스템에서 발생합니다.

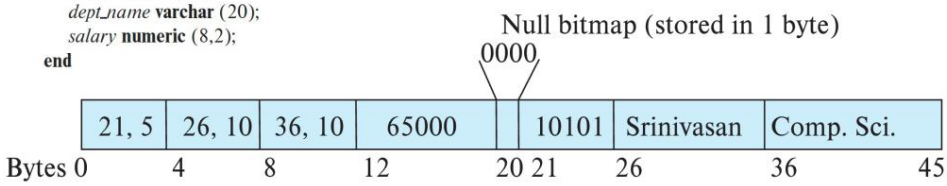
- 파일에 여러 레코드 유형 저장.
- 문자열 (varchar) 과 같은 하나 이상의 필드에 가변 길이를 허용하는 레코드 유형
- 반복 필드를 허용하는 레코드 유형(일부 이전 데이터에서 사용됨) 모델).

§ 속성은 순서대로 저장됩니다.

§ 고정 크기(오프셋, 길이)로 표현되는 가변 길이 속성
모든 고정 길이 속성 뒤에 저장된 실제 데이터

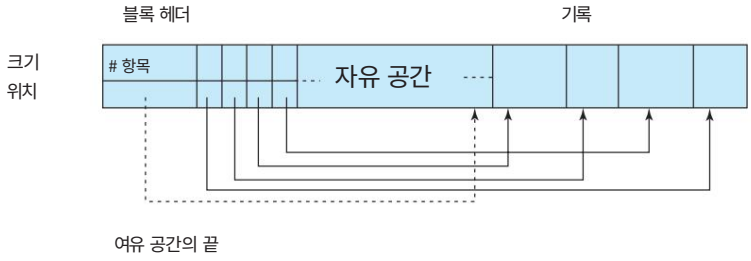
§ null 값 비트맵으로 표현되는 null 값

```
type instructor = record
        ID varchar (5);
        name varchar(20);
        dept_name varchar (20);
        salary numeric (8,2);
    end
```



 ©Silberschatz, Korth 및 Sudarshan

가변 길이 레코드: 슬롯 페이지 구조



§ 슬롯 페이지(즉, 블록) 헤더에는 다음이 포함됩니다.

- 레코드 항목 수
- 블록의 여유 공간 끝
- 각 레코드의 위치 및 크기

§ 레코드가 삭제되면 해당 레코드가 차지하는 공간이 해제되고 해당 항목이
삭제됨으로 설정됩니다(예를 들어 크기는 -1로 설정됨).

§ 레코드 사이에 빈 공간 없이 연속적으로 유지하기 위해 페이지 내에서 레코드를 이동할 수 있습니다. 헤더의 항목을 업데이트해야 합니다.

§ 포인터는 레코드를 직접 가리키면 안 됩니다. 대신 포인터는 레코드를 가리켜야 합니다.
헤더의 레코드 항목에.

 ©Silberschatz, Korth 및 Sudarshan

Machine Translated by Google

큰 개체 저장

§ 예: BLOB/CLOB 유형(이진 대형 객체/문자 대형 객체) § 레코드는 페이지보다 작아야 합니다.

§ 대안:

- 파일 시스템에 파일로 저장 • 데이터베이스에
  서 관리하는 파일로 저장 • 조각으로 나누어 별도의 관계
  로 여러 튜플에 저장
  - § PostgreSQL 토스트

파일의 기록 구성

§ 힙 – 공간이 있는 파일의 아무 곳에나 레코드를 배치할 수 있습니다 .

각 레코드의 검색 키

§ 다중 테이블 클러스터링 파일 조직 레코드 에서 여러 다른
관계는 동일한 파일에 저장될 수 있습니다.

- 동기: 관련 레코드를 동일한 블록에 저장하여 I/O 최소화

§ B+-트리 파일 구성 • 삽입/삭제가 있

는 경우에도 정렬된 스토리지 • 이에 대한 자세한 내용은 14장에

서 참조하십시오. § 해싱 – 검색 키에서 계

산된 해시 함수; 결과는
레코드를 배치해야 하는 파일의 블록

- 이에 대한 자세한 내용은 14장에서

Machine Translated by Google

힙 파일 구성

§ 여유 공간이 있는 파일의 아무 곳에나 레코드를 배치할 수 있습니다. § 레코드는 일반적으로 일단 할당되면 이동하지 않습니다. § 파일 내에서 여유 공간을 효율적으로 찾을 수 있어야 하는 것이 중요 합니다 . 각 항목은 바이트당 몇 비트이며

무료 블록

- 아래 예에서 블록당 3비트, 값을 8로 나눈 값은 사용 가능한 블록의 비율을 나타냅니다.

4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

§ 예, 블록 크기가 512바이트이고 파티션이 8개이면 각 파티션의 크기는 파티션은 512/8 = 64바이트입니다.

§ 첫 번째 항목은 블록에서 공간의 최소 4/8(즉, 4*64바이트)이 사용 가능함을 나타냅니다 .

- 두 번째 수준의 여유 공간 맵을 가질 수 있습니다.
- 아래 예에서 각 항목은 첫 번째 수준의 여유 공간 맵의 4개 항목 중 최대값을 저장합니다.

4	7	2	6

§ 여유 공간 맵이 주기적으로 디스크에 기록됨, 일부 항목에 대해 잘못된(오래된) 값이 있어도 괜찮음(감지 및 수정됨)

©Silberschatz, Korth 및 Sudarshan

순차적 파일 구성

§ 전체 파일의 순차적 처리가 필요한 애플리케이션에 적합

§ 파일의 레코드는 검색 키로 정렬됩니다.

10101 스리니바산	비교 과학.	65000	
12121 우	재원	90000	
15151 모차르트	음악	40000	
22222 아인슈타인	물리학의	95000	
32343 엘 사이드	역사 물리	60000	
33456 골드	학 Comp.	87000	
45565 카츠	과학.	75000	
58583 한정자	역사 금융	62000	
76543 싱		80000	
76766 크릭	생물학 비	72000	
83821 브란트	교. 과학.	92000	
98345 김	선택된. 공학	80000	

©Silberschatz, Korth 및 Sudarshan

Machine Translated by Google

순차 파일 구성(계속)

§ 삭제 – 포인터 체인 사용 § 삽입

– 여유 공간이 있으면 레코드를 삽입할 위치를 찾습니다. 여유 공간이 없으면

- 거기에 삽입하고 오버플로 블록 에 레
- 코드를 삽입합니다.
- 두 경우 모두 포인터 체인을 업데이트해야 합니다.

§ 순차적인 순서를 복원하기 위해 수시로 파일을 재구성해야 함

10101	스리니바산	비교 과학.	65000	
12121	우	재원	90000	
15151	모차르트	음악	40000	
22222	아인슈타인	물리학의	95000	
32343	엘 사이드	역사 물리	60000	
33456	금	학 Comp.	87000	
45565	카츠	과학.	75000	
58583	예선	역사 금융	62000	
76543	싱		80000	
76766	근육 경련	생물학 비	72000	
83821	브란트	교. 과학.	92000	
98345	김	선택된. 공학	80000	
32222 준		음악	48000	

다중 테이블 클러스터링 파일 구성

다중 테이블 클러스터링 파일 구성을 사용하여 하나의 파일에 여러 관계 저장

부서

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

강사

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

학과 및 강사의 다중 테이블 클러스터링

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

Machine Translated by Google

다중 테이블 클러스터링 파일 구성(계속)

§ 학과 강사 관련 문의 및 단일 학과 및 강사 관련 문의에 적합

§ 부서만 관련된 쿼리에 적합하지 않음

§ 가변 크기 레코드 결과

§ 특정 관계의 레코드를 연결하기 위해 포인터 체인을 추가할 수 있습니다.

§ 다중 테이블 클러스터링은 Oracle 데이터베이스 시스템에서 지원됩니다.

©Silberschatz, Korth 및 Sudarshan

파티셔닝

§ 테이블 분할: 관계의 레코드는 별도로 저장되는 더 작은 관계로 분할될 수 있습니다. § 예:
트랜잭션 관계는 transaction_2018,
transaction_2019 등으로 분할될 수 있습니다.

§ 트랜잭션에 작성된 쿼리는 모든 파티션의 레코드에 액세스해야 합니다.

- 쿼리에 연도=2019와 같은 선택 항목이 없는 경우에만
  하나의 파티션이 필요함

§ 파티셔닝

- 여유 공간 관리와 같은 일부 작업의 비용 절감 • 서로 다른 스토리지 장치에 서로 다른 파티션을 저장할 수 있습니다.
  - § 예: SSD의 현재 연도에 대한 트랜잭션 파티션, 이전 연도에 대한
    자기 디스크에

©Silberschatz, Korth 및 Sudarshan

Machine Translated by Google

데이터 사전 저장 Data Dictionary Storage

데이터 사전 (시스템 카탈로그 라고도 함)은 메타데이터를 저장합니다 .
즉, 다음과 같은 데이터에 대한 데이터

§ 관계에 대한 정보
- 관계의 이름
- 각 관계의 속성 이름, 유형 및 길이 • 보기의 이름 및 정의
- 무결성 제약

§ 암호를 포함한 사용자 및 계정 정보 § 통계 및 설명 데이터
- 각 관계의 튜플 수 § 물리적 파일 구성 정보 • 관계가 저장되는 방법(순차/해시/…) • 관계의 물리적 위치

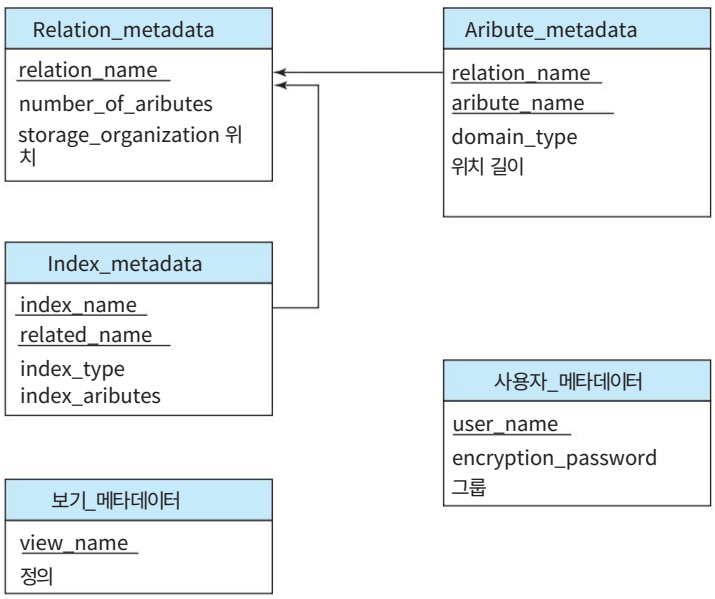
§ 인덱스에 대한 정보(14장)

시스템 메타데이터의 관계적 표현

§ 디스크의 관계형 표현

§ 특화된 데이터 구조
효율적인 메모리 액세스를 위해 설계됨

§ 데이터베이스 시스템이 관계에서 레코드를 검색해야 할 때마다 먼저 관계 메타데이터 관계를 참조하여 관계의 위치와 저장 조직을 찾은 다음 이 정보를 사용하여 레코드를 가져와야 합니다.

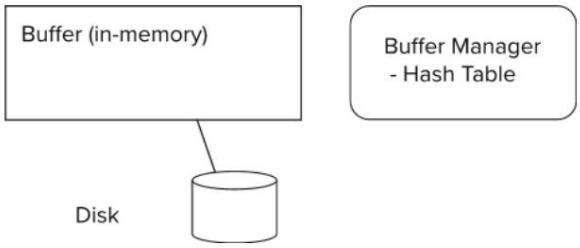


Machine Translated by Google

스토리지 액세스

§ 블록은 스토리지 할당 및 데이터 전송 단위입니다. § 데이터베이스 시스템은 디스크와 메모리 간의 블록 전송 수를 최소화하려고 합니다. 주 메모리에 가능한 한 많은 블록을 유지함으로써 디스크 액세스 수를 줄일 수 있습니다.

§ 버퍼 – 디스크 블록의 복사본을 저장하는 데 사용할 수 있는 주 메모리 부분. § 버퍼 관리자 – 메인 메모리에 버퍼 공간 할당을 담당하는 하위 시스템.



버퍼 관리자

§ 프로그램은 디스크에서 블록이 필요할 때 버퍼 관리자를 호출합니다.

- 블록이 이미 버퍼에 있는 경우 버퍼 관리자는 메인 메모리의 블록 주소를 반환합니다.
- 블록이 버퍼에 없으면 버퍼 관리자가 § 블록을 위해 버퍼에 공간을 할당합니다.
  - 필요한 경우 다른 블록을 교체(버리기)하여 새로운 블록을 위한 공간. • 수정된 경우에만 디스크에 다시 기록된 대체 블록
    가장 최근에 디스크에 기록/가져온 시간부터.

§ 디스크에서 버퍼로 블록을 읽고 메인 메모리에 있는 블록의 주소를 요청자에게 반환합니다.

Machine Translated by Google

버퍼 관리자

§ 버퍼 교체 전략 (자세한 내용은 추후 공개!) § 고정된 블록: 디스크에 다시 쓸 수 없는 메모리 블록 • 블록에서 데이터 읽기/쓰기 전에 고정 완료 • 읽기/쓰기가 완료되면 고정 해제 완료 • 다중 동시 핀 /고정 해제 작업 가능

§ 핀 수 유지, 버퍼 블록은 핀 수 = 0인 경우에만 제거될 수 있습니다.

§ 버퍼에 대한 공유 및 배타적 잠금

• 동시 작업이 이동/재구성될 때 페이지 내용을 읽는 것을 방지하고 한 번에 하나의 이동/재구성만 보장하는 데 필요합니다.

• 독자는 공유 잠금을 얻습니다. 블록 업데이트에는 배타적 잠금이 필요합니다. • 잠금 규칙: § 한 번에 하나의 프로세스만 배타적 잠금을 얻을 수 있습니다. § 공유 잠금은 배타적 잠금과 동시에 사용할 수 없습니다.

버퍼 교체 정책

§ 대부분의 운영 체제는 가장 최근에 사용되지 않은 블록 (LRU) 을 대체합니다. 전략)

• LRU 이면의 아이디어 – 블록 참조의 과거 패턴을 미래 참조의 예측자로 사용 • LRU는 일부 쿼리에 적합하지 않을 수 있습니다.

§ 쿼리에는 잘 정의된 액세스 패턴(예: 순차 스캔)이 있으며 데이터베이스 시스템은 사용자 쿼리의 정보를 사용하여 향후 참조를 예측할 수 있습니다 .

§ LRU에 대한 잘못된 액세스 패턴의 예: 중첩 루프에 의해 2개의 관계 r과 s의 조인을 계산할 때

```
r의 각 튜플 tr do 각 튜플 s
  의 ts do 튜플 tr과 ts가 일치
    하면 …
```

Machine Translated by Google

버퍼 교체 정책(계속) Buffer Replacement Policies (Cont.)

§ 즉시 던지기 전략 – 블록이 차지하는 공간을 즉시 비웁니다.
해당 블록의 최종 튜플이 처리됨 에 따라

§ 가장 최근에 사용된(MRU) 전략 - 시스템이 블록을 고정해야 합니다.
현재 처리 중입니다. 해당 블록의 마지막 튜플이 처리된 후 블록이 고정 해제되
고 가장 최근에 사용된 블록이 됩니다.

§ 버퍼 관리자는 확률에 대한 통계 정보를 사용할 수 있습니다.
요청이 특정 관계를 참조한다는 것 • 예: 데이터 사전

에 자주 액세스합니다. 휴리스틱: 유지
주 메모리 버퍼의 데이터 사전 블록

§ 운영 체제 또는 버퍼 관리자는 쓰기를 재정렬할 수 있습니다.

• 시스템 충돌 시 디스크의 데이터 구조가 손상 될 수 있습니다 .

• 쓰기 순서를 신중하게 지정하면 이러한 많은 문제를 피할 수 있습니다.

디스크 블록 액세스 최적화(계속) Optimization of Disk-Block Access (Cont.)

§ 버퍼 관리자는 복구 목적으로 블록의 강제 출력을 지원합니다(자세한 내용은 19 장 참조) .

휘발성 RAM 또는 플래시 버퍼를 즉시

• 쓰기 순서를 재정렬하여 디스크 암 이동을 최소화할 수 있습니다.

§ 로그 디스크 - 블록 업데이트의 순차적 로그를 작성하는 전용 디스크

• 비휘발성 RAM 과 똑같이 사용됨 검색이 필요

하지 않기 때문에 로그 디스크에 쓰기가 매우 빠름 저널링 파일 시스템이

NV -RAM 또는 로그 디스크에 순서대로 데이터 쓰기

• 저널링 없이 재정렬: 파일 시스템 데이터 손상 위험

Machine Translated by Google

컬럼 지향 스토리지

§ 열 표현 이라고도 함 § 관계의 각 속성을 개별적으로 저장 § 예

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

열 표현

§ 혜택:

- 일부 속성에만 액세스하는 경우 IO 감소
- CPU 캐시 성능 향상
- 압축 향상
- 최신 CPU 아키텍처의 벡터 처리

§ 단점

- 열 표현에서 튜플 재구성 비용
- 튜플 삭제 및 업데이트 비용
- 압축 해제 비용

§ 결정 지원에 보다 효율적인 것으로 밝혀진 컬럼 표현
행 중심 표현

§ 트랜잭션 처리에 선호되는 전통적인 행 중심 표현 § 일부 데이터베이스는 두 가지 표현을 모두 지원합니다.

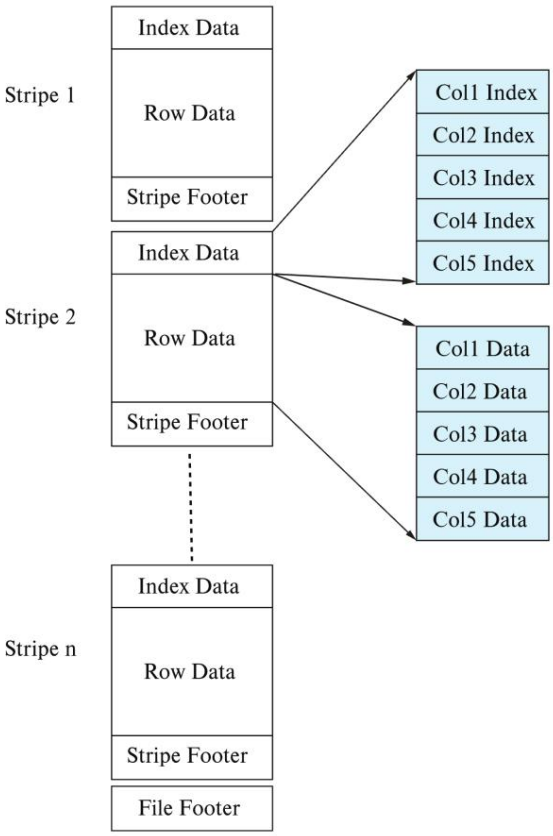
- 하이브리드 행/열 저장소 라고 함

Machine Translated by Google

열 형식 파일 표현

§ ORC(최적화된 행 Columnar) 및 Parquet: 파일 내부에 열 형식 저장소가 있는 파일 형식 § 빅 데이터 처리 애플리케이션에 매우 인기 있음

§ 오른쪽에 표시된 Orc 파일 형식:

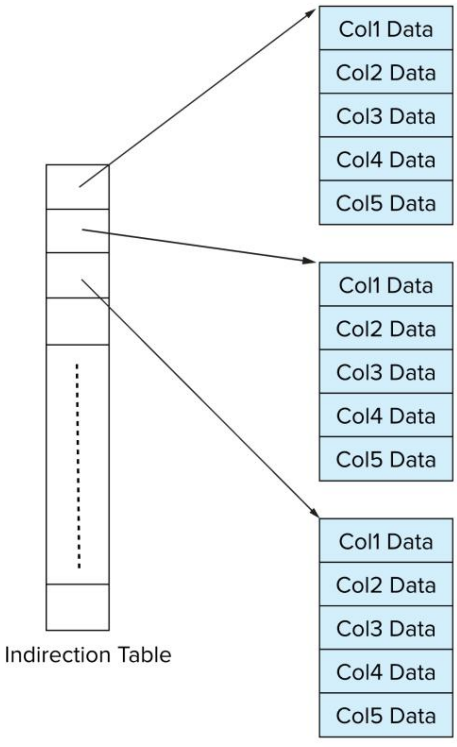


©Silberschatz, Korth 및 Sudarshan

메인 메모리 데이터베이스의 스토리지 조직

§ 레코드를 직접 저장할 수 있습니다. 버퍼 관리자가 없는 메모리

§ 의사 결정 지원 애플리케이션을 위해 메모리 내에서 열 기반 스토리지를 사용할 수 있습니다. • 압축을 통해 메모리 요구 사항이 줄어듭니다.



©Silberschatz, Korth 및 Sudarshan

Machine Translated by Google

13장 끝 Chapter 13

©Silberschatz, Korth 및 Sudarshan