데이터베이스론	국가 전산 7급	2017년 8월 26일

♤ 합격선/최종합격인원(75.83점/27명) - 채용예정인원 26명 ♤

1. 데이터베이스 관리 시스템(DBMS)의 역할에 대한 설명으로 옳지 않은 것은? [2017년 국가 7급]

① 데이터 조작어(DML)로 스키마의 구조를 기술하여 시스템 카탈로그(혹은 데이터 사전)에 저장한 후 필요할 때 활용한다.

② 질의어 처리기는 질의문을 파싱하고 분석해서 효율적인 데이터베이스 접근코드를 생성한다.

③ 트랜잭션 관리자는 무결성 제약조건 검사, 사용자의 권한검사, 병행제어, 회복 등의 작업을 수행한다.

④ 저장 데이터 관리자는 디스크에 저장되어 있는 사용자 데이터베이스와 시스템 카탈로그의 접근을 책임진다.

♤ **데이터베이스 관리 시스템(DBMS) 역할**

- **데이터 조작어(DML)**로 **스키마의 구조를 기술**하여 시스템 카탈로그(혹은 데이터 사전)에 저장한 후 필요할 때 활용한다.**(×)**
  → **스키마**는 **DDL로 기술**하고
  → **스키마**는 **DDL 컴파일러**에 의해 **번역**되어 데이터 사전에 저장된다.

- **DDL 컴파일러(DDL 처리기)**는 DBA가 명세한 데이터베이스 스키마 정의를 메타 데이터로 처리하여 시스템 카탈로그에 저장한다.
- **질의어 처리기**는 터미널을 통해 일반사용자가 제출한 고급 질의문을 처리한다.
- **질의어 처리기**는 질의문을 분석해서 파싱하고 효율적인 데이터베이스 접근코드를 생성한다.
- **질의어 처리기**가 생성한 코드는 **런타임 데이터베이스 처리기**에 보내진다.
- **DML 예비 컴파일러**는 응용프로그램 내에 삽입된 데이터 조작어(DML)를 추출하고 프로시저 호출로 대체시킨다. → 추출된 DML 명령어는 DML 컴파일러에게 넘긴다.
- **DML 컴파일러**는 DML 명령어를 목적코드로 변환한다,
- **트랜잭션 관리자**는 무결성 제약조건 검사, 사용자의 권한검사, 병행제어, 회복 등의 작업을 수행한다.
- **저장 데이터관리자**는 디스크에 저장되어 있는 사용자 데이터베이스와 시스템 카탈로그의 접근을 책임진다.
- **저장 데이터관리자**는 디스크에 저장된 데이터베이스를 운영체제의 모듈(파일관리자, 디스크관리자)을 이용하여 물리적 접근을 제어한다.
- **런타임 데이터베이스 처리기**는 실행시간에 데이터베이스에 접근하여 관리한다.

정답 : ①

2. 관계형 데이터베이스에 대한 설명으로 옳지 않은 것만을 모두 고른 것은? [2017년 국가 7급]

ㄱ. 기본키 속성이 복합속성인 경우 그 속성의 일부 요소 속성에서 널(NULL) 값을 가질 수 있다.

ㄴ. 슈퍼키는 후보키가 되기 위한 필요충분조건이다.

ㄷ. 릴레이션 R이 릴레이션 S를 참조하는 경우 R의 외래키가 S의 기본키가 아닌 후보키 중 하나를 참조해야 한다.

ㄹ. 테이블에 튜플 삽입 시 엔터티 무결성 혹은 키 제약조건, 도메인 제약조건, 참조 무결성 제약조건이 위배될 수 있다.

---

① ㄱ, ㄴ ② ㄷ, ㄹ

③ ㄱ, ㄴ, ㄷ ④ ㄱ, ㄷ, ㄹ

♤ **관계형 데이터베이스**

ㄱ. 기본키 속성이 복합속성인 경우 그 속성의 일부 요소 속성에서 널(NULL) 값을 가질 수 있다.**(×)**

→ 기본키 속성은 널(NULL) 값을 가질 수 없다.

→ "기본키 속성이 **복합속성**인"라 했는데 "기본키 속성이 **복합키**인"가 정확한 표현이다.

ㄴ. 슈퍼키는 후보키가 되기 위한 필요충분조건이다.**(×)**

→ 슈퍼키는 후보키가 되기 위한 필요조건이다.

→ 더 줄일 수 없는 슈퍼키는 후보키이다.

→ 필요조건은 참이 되기 위해서 반드시 충족되어야 하는 조건이다.(사람은 동물이다)

→ 필요충분조건은 동치인 경우이다. 슈퍼키와 후보키는 동치가 아니다.

ㄷ. 릴레이션 R이 릴레이션 S를 참조하는 경우 R의 외래키가 S의 기본키가 아닌 후보키 중 하나를 참조해야 한다.**(×)**

→ 릴레이션 R의 외래키는 참조하는 릴레이션 S에서는 기본키 이어야 한다.

ㄹ. 테이블에 튜플 삽입 시 엔터티 무결성 혹은 키 제약조건, 도메인 제약조건, 참조 무결성 제약조건이 위배될 수 있다.(○)

→ 테이블에 튜플 삽입 시, 각종 제약조건에 위배될 수 있다.

→ 제약조건에 위배되는 튜플은 테이블에 삽입되지 않을 뿐이다.

→ 이 부분에 대해서 질문을 많이 받았다. 제약조건에 위배되는 튜플이 어떻게 삽입되는지?

→ 문제에서 "제약조건이 위배될 수 있다"라고 하였지, 삽입된다고 한 것은 아니다.

→ 참으로, 묘하게 문제를 출제한 것이다. 더 이상 할 말이 없다.

정답 : ③

3. 테이블의 튜플 데이터를 파일 내에 저장하기 위한 구조는 힙(heap) 파일구조, 순차(sequential) 파일구조, 해시(hash) 파일구조로 구분될 수 있다. 일반적으로 테이블에 가장 빈번하게 이루어지는 연산에 의해 사용될 파일구조가 결정된다. 다음에 주어진 각각의 연산에 대하여 가장 효율적인 파일구조를 바르게 연결한 것은? [2017년 국가 7급]

ㄱ. 속성 값의 범위 구간을 주고 이에 해당하는 튜플 검색(range search)
ㄴ. 빈번한 튜플 삽입
ㄷ. 특정 속성 값과 일치하는 튜플 검색

	㉠	㉡	㉢
①	순차	해시	힙
②	순차	힙	해시
③	힙	순차	해시
④	해시	힙	순차

♤ **효율적인 파일구조**

ㄱ. 속성 값의 범위 구간을 주고 이에 해당하는 튜플 검색(range search) – **순차**
→ 이유는, 속성 값의 범위 구간을 주고 순차 탐색하므로
→ 순차구조는 키를 기준으로 데이터가 정렬되어 있다.(범위 검색에 적절)

ㄴ. 빈번한 튜플 삽입 – **힙**
→ 이유는, 파일 안에 빈 공간이 있으면 튜플은 어디든지 삽입 가능하다.
→ 참고로, 힙 파일구조와 자료구조 힙은 같은 개념이 아니다.

ㄷ. 특정 속성 값과 일치하는 튜플 검색 – **해시**
→ 이유는, 해시구조에서 특정 값을 탐색하는 시간복잡도는 O(1)이다.

엔트리 순차파일	• 엔트리 순차파일(entry-sequence file)은 비순서파일이다. • 엔트리 순차파일은 레코드가 시스템에 입력되는 순서로 저장된다. • 엔트리 순차파일은 **힙(heap)** 또는 **파일(pile)**이라 한다. • 일반적으로, 새로 입력되는 레코드는 파일 끝에 저장된다. • 만약, 파일 중간에 빈 공간이 있으면 저장된다. • 레코드 검색은 모든 레코드들을 순차적으로 접근해야 한다. • 힙 파일은 좋은 성능 유지를 위해서는 주기적으로 재조직할 필요가 있다.
키 순차파일	• 키 순차파일(key-sequential file)은 레코드 키 순서로 저장된 파일이다. • 키 순차파일은 엔트리 순차파일이 키 순서로 가공된 구조이다. • 일반적으로, 순차파일이라고 하면 키 순차파일을 의미한다.

정답 : ②

4. 부서와 사원 테이블을 생성하는 SQL 문장을 수행한 후 튜플 삽입으로 두 테이블의 상태가 다음과 같을 때, 테이블 연산 수행에 대한 설명으로 옳지 않은 것은? [2017년 국가 7급]

```
CREATE TABLE 부서 (
  부서번호 INT NOT NULL,
  부서명   VARCHAR(20),
  PRIMARY KEY(부서번호));

CREATE TABLE 사원 (
  사번     INT NOT NULL,
  이름     VARCHAR(20),
  부서번호 INT,
  PRIMARY KEY(사번),
  FOREIGN KEY(부서번호) REFERENCES 부서(부서번호));
```

부서	
부서번호	부서명
1	자재부
2	영업부

사원		
사번	이름	부서번호
11	홍길동	1
12	이순신	2

① 부서 테이블에서 (2, '영업부') 튜플을 삭제한다면 참조 무결성 제약조건을 위배한다.
② 사원 테이블에 (13, '강감찬', 'A1') 튜플을 삽입한다면 도메인 무결성 제약조건을 위배한다.
③ 사원 테이블에 (14, '김유신', 0) 튜플을 삽입한다면 참조 무결성 제약조건을 위배한다.
④ 부서 테이블에 (1, '연구부') 튜플을 삽입한다면 참조 무결성 제약조건을 위배한다.

♤ **무결성 제약조건**

④ 부서 테이블에 (1, '연구부') 튜플을 삽입한다면 **참조** 무결성 제약조건을 위배한다.**(×)**
→ 부서 테이블에 (1, '연구부') 튜플을 삽입한다면 **개체** 무결성 제약조건을 위배한다.

부서	
부서번호	부서명
1	자재부
2	영업부
1	연구부

// 개체 무결성 제약
- 기본키는 널 값(Null Value)을 가질 수 없다.
- 기본키는 유일성과 최소성을 가져야 한다.
- (1, '연구부') 튜플을 삽입한다면, 유일성이 만족되지 않는다.

정답 : ④

5. 인덱싱(indexing) 기법에 대한 설명으로 옳은 것은? [2017년 국가 7급]

① 순서(ordered) 인덱싱은 키 값을 이용하여 찾고자 하는 튜플이 저장된 물리적 주소를 계산해내는 기법이다.

② 차수가 p인 B트리의 모든 노드는 최소한 $\lceil \frac{p}{2} \rceil$ 개의 트리 포인터를 갖는다.

③ B트리와 B+트리의 노드 크기가 동일한 경우, B+트리의 내부노드 차수(degree)가 B트리 노드 차수보다 더 크다.

④ 100명이 저장된 학생 테이블의 '성별' 속성에 대하여 비트맵 인덱스를 만들 때, 남학생의 비트맵 인덱스 크기는 남학생 수에 영향을 받는다.

♤ 인덱싱 기법

① 순서(ordered) 인덱싱은 **키 값**을 이용하여 찾고자 하는 튜플이 저장된 물리적 주소를 계산해내는 기법이다.**(×)**

→ 순서 인덱스는 검색 키가 정렬된 순서로 저장된다.

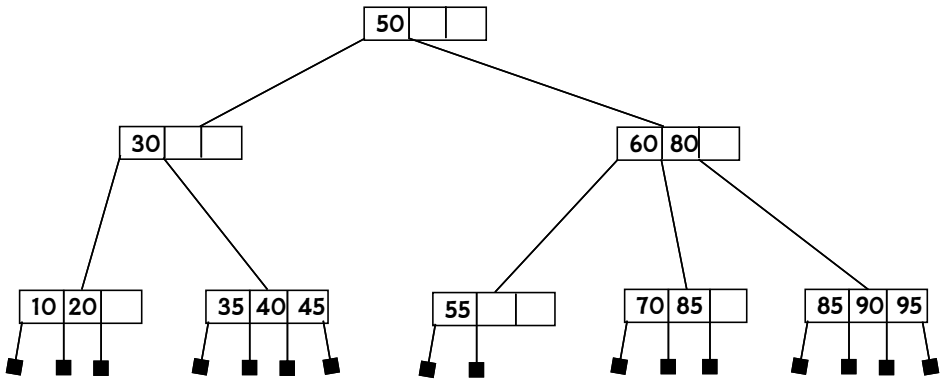
→ **키 값**을 이용하여 찾고자 하는 튜플이 저장된 주소를 계산하는 기법은 **해싱**이다.

② 차수가 p인 B트리의 **모든 노드**는 최소한 $\lceil \frac{p}{2} \rceil$ 개의 트리 포인터를 갖는다.**(×)**

→ 차수가 4일 때, 각 노드의 차수는 2 또는 3 또는 4가 될 수 있다.

→ **근노드**와 **외부노드**를 제외한 노드가 최소한 $\lceil \frac{p}{2} \rceil$ 개의 트리 포인터를 갖는다.

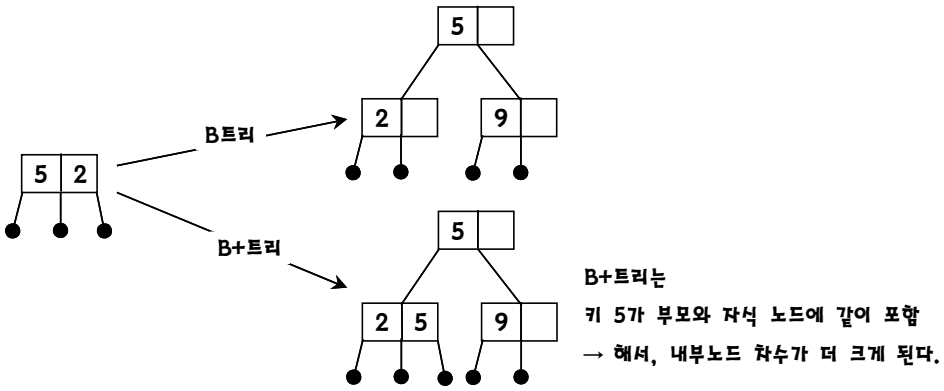
// 차수가 4인 B 트리



• 예를 들어, 차수가 100인 B 트리에서 근노드는 차수가 2가 될 수 있다.

③ B트리와 B+트리의 노드 크기가 동일한 경우, B+트리의 **내부노드 차수**(degree)가 B트리 노드 차수보다 더 크다.(○)

→ 키 : **5 2 9**를 차례로 삽입하는 경우



④ 100명이 저장된 학생 테이블의 '성별' 속성에 대하여 비트맵 인덱스를 만들 때, 남학생의 비트맵 인덱스 크기는 남학생 수에 영향을 받는다.(×)

→ 남학생 수에 영향 없이 비트맵 인덱스 크기는 100이 된다.(전체 자료 수에 영향)

// 비트맵 인덱스

- 인덱스 값을 0과 1로 변환하여 저장한다.(비트맵)
- 0과 1로 표현하므로 저장 공간을 절약할 수 있고, 수행 속도도 향상된다.
- 비트 단위로 표현하므로 압축 알고리즘으로 활용할 수 있다.

- 성별에 대한 비트맵 인덱스(학생이 5명인 경우)

학번	이름	성별
1	홍재연	남
2	홍하은	여
3	김영미	여
4	홍연재	남
5	김하은	여

남 : 10010
여 : 01101

- 성별과 무관하게 비트맵 인덱스 크기는 5가 된다.(학생이 5명인 경우)
- 학생 수가 100명이면, 비트맵 인덱스 크기는 100이 된다.
- 비트가 1로 설정돼 있으면, 상응하는 레코드가 해당 키 값을 포함하고 있음을 의미한다.

정답 : ③

6. 두 트랜잭션 T1과 T2가 다음과 같은 트랜잭션 스케줄로 실행될 때 발생하는 문제는? (단, 데이터 항목 X와 Y의 초깃값은 각각 200과 300이고, read(X)와 write(X)는 각각 트랜잭션이 데이터 항목 X를 읽고 쓰는 연산이다) [2017년 국가 7급]

T1	T2
read(X); X = X - 20;	
	read(X); X = X + 50;
write(X); read(Y);	
	write(X);
Y = Y - 20; write(Y);	

	read(Y);
	Y = Y + 30;
	write(Y);

① 갱신 손실 문제(lost update problem)
② 오손 읽기 문제(dirty read problem)
③ 부정확한 요약 문제(incorrect summary problem)
④ 반복할 수 없는 읽기 문제(unrepeatable read problem)

♤ 갱신 손실 문제(lost update problem)

T1	T2	초깃값 : X = 200, Y = 300
read(X); X = X - 20;		X = X - 20; X = 200 - 20 = 180
	read(X); X = X + 50;	X = X + 50; X = 200 + 50 = 250

write(X);
read(Y);
		write(X); // X = 180
		read(Y); // Y = 300

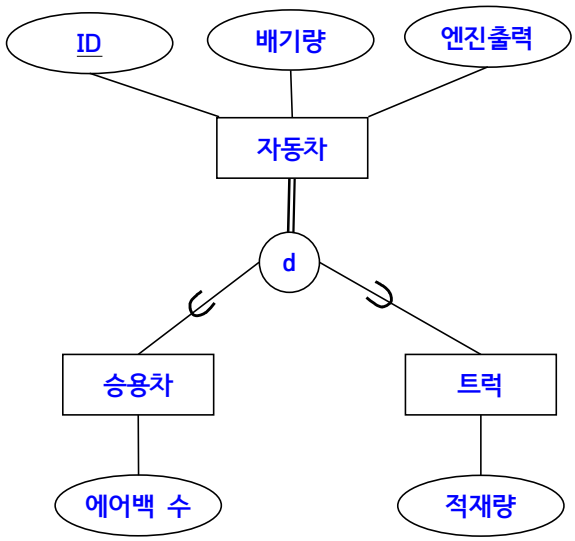
	write(X);	write(X); // X = 250(갱신손실)
Y = Y - 20; write(Y);		Y = Y - 20 = 300 - 20 = 280 write(Y); // Y = 280

	read(Y);
	Y = Y + 30;
	write(Y);
		read(Y); // Y = 280
		Y = Y + 30 = 280 + 30 = 310
		write(Y); // Y = 310

- T1이 변경한 x값을 데이터베이스에 기록하기 전에 T2가 x값을 읽는다.
  → x의 최종값은 틀리게 된다.(특정 트랜잭션의 연산 결과가 분실)
  → T1이 갱신한 x의 결과를 잃어버리게 된다. x는 잘못된 값을 가진다.**(갱신 손실)**
- 트랜잭션들이 하나의 데이터를 동시에 갱신할 때 발생하는 문제이다.
  → 어떤 트랜잭션이 갱신한 내용을 다른 트랜잭션이 덮어 쓴 경우(갱신 무효)

정답 : ①

7. ISA 관계를 슈퍼타입(혹은 상위클래스)과 서브타입(혹은 하위클래스)으로 표현한 EER(Enhanced E-R) 다이어그램의 한 예를 다음 그림에서 보여주고 있다. 이 그림에서 슈퍼타입은 전체 세분화(total specialization)되어 있고, 서브타입들은 서로 분리(disjoint)되어 있다. 이 EER 다이어그램에 대한 설명으로 옳지 않은 것은? (단, 밑줄은 기본키를 의미한다)



① 트럭 엔터티의 속성의 개수는 4개이다.
② 위 EER 다이어그램에서는 승용차도 아니고 트럭도 아닌 자동차(예를 들면 버스)는 존재할 수 없다.
③ 자동차 엔터티를 테이블로 표현하지 않고, ID, 배기량, 엔진 출력 속성을 포함한 승용차와 트럭 테이블로 표현할 수 있다.
④ 승용차와 트럭이 중복(overlap)되었다면 자동차 엔터티는 테이블로 변환할 수 없다.

♤ ISA 관계를 슈퍼타입으로 표현한 EER(Enhanced E-R) 다이어그램

④ 승용차와 트럭이 중복(overlap)되었다면 자동차 엔터티는 테이블로 변환할 수 없다.(×)
↓다음처럼 테이블을 만들 수 있다.

방법 1	**자동차**(ID, 배기량, 엔진출력) **승용차**(ID, 에어백수) **트럭**(ID, 적재량)
방법 2	**승용차**(ID, 배기량, 엔진출력, 에어백수) **트럭**(ID, 배기량, 엔진출력, 적재량) ← 트럭 속성 개수는 4개
방법 3	**자동차**(ID, 배기량, 엔진출력, 차종, 에어백수, 적재량) ↳ 차종 속성을 추가할 수 있다.

// 특수화 계층구조를 테이블로 표현(3가지 표현 방법)

① 슈퍼클래스에 대해 하나의 테이블과 각 서브클래스에 각각 하나씩의 테이블을 생성한다.
자동차=(ID, 배기량, 엔진출력)
승용차=(ID, 에어백수)
트럭=(ID, 적재량)

② 각 서브클래스가 슈퍼클래스의 속성을 상속받아, 각 서브클래스에 대해 각각 하나의 테이블을 생성한다.
승용차=(ID, 배기량, 엔진출력, 에어백수)
트럭=(ID, 배기량, 엔진출력, 적재량)

③ 모든 서브클래스의 속성을 포함하여 슈퍼클래스에 대해 하나의 테이블을 생성한다.
자동차=(ID, 배기량, 엔진출력, 차종, 에어백수, 적재량)
↳ 차종 속성을 추가할 수 있다.

// 전체참여와 부분참여

- 전체참여는 개체 집합의 모든 개체가 관계에 참여한다.
- 전체참여는 EER 다이어그램에서 이중선으로 표시한다.
- 부분참여는 개체 집합의 모든 개체 중 일부만 참여한다.
- 부분참여는 EER 다이어그램에서 실선으로 표시한다.
- 전체참여는 (최솟값, 최댓값)으로 표현할 때 최솟값이 1이상으로 모두 참여 한다는 뜻이고, 부분참여는 최솟값이 0 이상이다.

// EER 다이어그램에서 표현

- 하나의 수퍼클래스와 여러 개의 서브클래스들을 특수화 고리(specialization circle)로 연결
- EER 다이어그램에서 원(○)을 특수화 고리라 한다.
- EER 다이어그램에서 원(○) 안에 d는 서브타입들은 서로 분리(disjoint)되어 있음을 의미
- 서브클래스는 컵 모양(∪)의 상속 기호로 수퍼클래스와 연결

// IS-A 관계 / HAS-A 관계

- IS-A 관계는 "~는 ~이다."가 성립되는 상속관계이고, (학생은 사람이다)
- HAS-A 관계는 "~는 ~가진다."가 성립되는 포함관계이다. (군인은 총을 가진다)
- HAS-A 관계는 상속관계를 이용하여 포함관계를 구현할 수도 있다.

// IS-A 관계

```
public class 사람
{
  String 이름;
  int 나이;
  int 성별;
}
public class 학생 extends 사람      //학생클래스가 사람클래스를 상속
{
  int 학번;
  int 학과;
}
```

- "학생은 사람이다". 라는 관계를 위 코드처럼 표현했을 때 IS-A 관계라고 한다.
- IS-A 관계는 아래로 갈수록 특성이 많아진다.

// HAS-A 관계

```
public class 총
{
  String 총이름;
  int 총알수;
}
public class 군인
{
   총 소총;              //군인클래스 안에 총클래스(객체멤버)를 가진다.
}
```

- "군인은 총을 가진다."의 구조를 HAS-A 관계라고 한다.
- HAS-A는 상속관계가 아닌 포함관계로 표현할 수 있다.(객체멤버에 의한 포함관계)
- 주어진 예는 HAS-A 관계를 상속관계가 아닌 포함관계로 표현한 것이다.

정답 : ④

8. 데이터베이스 관리 시스템의 캐시 관리 방식에 대한 설명으로 옳지 않은 것은?

① no-steal 방식에서는 회복과정 중 UNDO와 REDO 연산의 수행이 모두 필요하다.
② 갱신된 페이지의 수가 많고 크기가 큰 경우 no-steal 방식에 비해 steal 방식이 필요한 캐시 버퍼 크기를 줄일 수 있다.
③ 완료된(committed) 트랜잭션에서 갱신된 페이지가 다른 트랜잭션들에서도 빈번히 갱신되는 경우, force 방식에 비해 no-force 방식이 디스크로부터 캐시로 그 페이지를 다시 읽는 횟수를 줄일 수 있다.
④ force 방식은 트랜잭션이 완료되기 전에 그 트랜잭션이 갱신한 모든 페이지들을 디스크에 저장하게 한다.

♤ **데이터베이스 관리 시스템의 캐시 관리 방식**

· **no-steal** 방식에서는 회복과정 중 UNDO와 REDO 연산의 수행이 **모두 필요**하다.**(×)**
→ **no-steal** 방식은 자료가 갱신되어도 트랜잭션 종료 전에는 디스크에 기록하지 않는다.
→ 해서, **UNDO** 과정은 수행할 필요가 없고 **REDO 과정만 수행**한다.(지연갱신 기법)

Steal	· 트랜잭션이 **완료되기 전**에 갱신된 페이지를 언제든지 디스크에 **기록**할 수 있는 정책 · DBMS의 버퍼관리자가 다른 트랜잭션을 위한 버퍼가 필요로 할 때 적용한다. · 미완료된 트랜잭션이 갱신한 페이지를 훔치는 것처럼 디스크에 기록한다. · 장점: 로그버퍼 공간을 줄일 수 있다.
No-Steal	· 트랜잭션이 완료되기 전까지는 갱신된 페이지를 **버퍼에 유지**하는 정책 · 트랜잭션이 완료되기 전까지는 갱신된 페이지를 디스크에 기록하지 않는다.

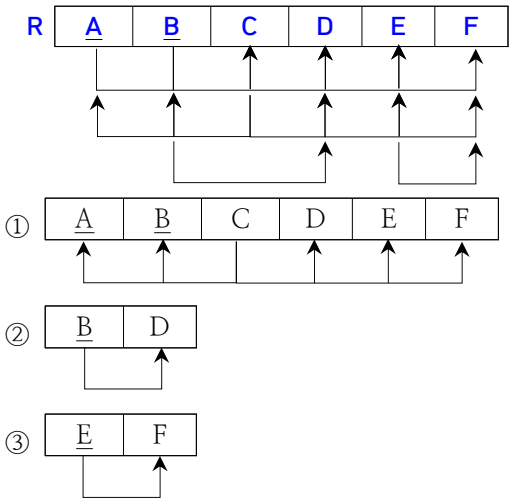
· **No-Steal** 정책은 **undo**가 불필요한 매력을 가지고 있다.(디스크에 기록한 것이 없으므로)
· 하지만, **No-Steal** 정책은 매우 큰 메모리 버퍼가 필요하다는 문제점을 가지고 있다.
· **Steal** 정책은 필연적으로 **undo** 복구를 수반한다.
· 이유는, **Steal** 정책은 수정된 페이지가 언제든지 디스크에 기록될 수 있으므로
· 대부분의 DBMS 채택하는 버퍼 관리 정책은 **Steal** 정책이다.

Force	· 트랜잭션이 갱신한 모든 페이지를 트랜잭션 **완료 시점에 즉시 디스크에 반영**하는 정책
No-Force	· 트랜잭션이 갱신한 페이지를 트랜잭션 완료 시점에 **즉시 디스크에 반영하지 않는** 정책 · 여기서 주의할 것은 즉시 반영하지 않는다는 것이지 어떤 기록도 없다는 것은 아니다. · 특정 페이지가 여러 트랜잭션에 의해 자주 갱신되는 경우에 적용한다. · 장점 : 디스크 입출력 비용을 줄인다.

· **Force** 정책은 **redo**가 불필요(수정된 페이지가 이미 디스크의 데이터베이스에 반영되어서)
· 하지만, **Force** 정책을 적용해도 데이터베이스 백업 복구는 **redo**가 요구된다.
· **No-Force** 정책은 **redo**가 필요(수정된 페이지가 데이터베이스에 반영되지 않을 수 있기에)
· 대부분의 DBMS 채택하는 버퍼 관리 정책은 **No-Force** 정책이다.

정답 : ①

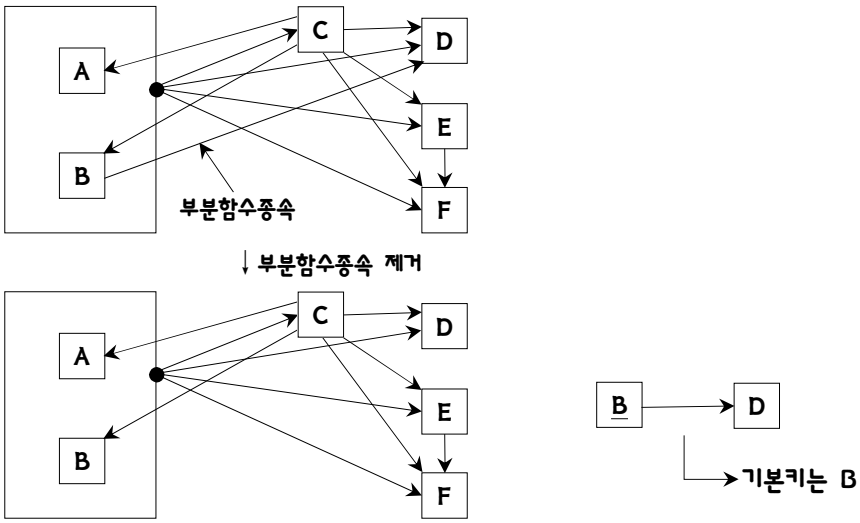
9. 다음 그림은 릴레이션 R과 그 함수적 종속성을 표현하고 있다. 속성 C는 릴레이션 R의 후보키이며, 이 릴레이션은 이미 제1정규화를 수행하였다. 이 릴레이션을 후보키까지 고려하여 제2정규화하였을 때, 분해된 릴레이션 중 기본키가 A, B가 아닌 릴레이션은? (단, 밑줄은 기본키를 의미한다) [2017년 국가 7급]



④ 제2정규화하더라도 변화 없음

♤ **정규화**

- 제2정규화는 부분함수종속을 제거한 것이다.



- 분해된 릴레이션 B→D에서 기본키는 B이다.(A, B가 아님)

정답 : ②

10. 병렬 데이터베이스 구조에 대한 설명으로 옳지 않은 것은? [2017년 국가 7급]

① 공유 메모리(shared memory) 구조는 비공유(shared nothing) 구조에 비해 프로세서 간 통신 속도가 빠르지만 프로세서 수의 확장성이 낮다.

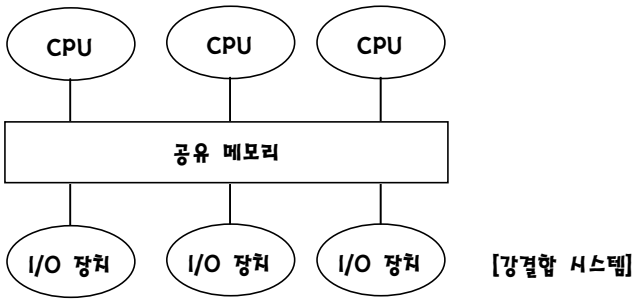
② 공유 디스크(shared disk) 구조는 공유 메모리 구조에 비해 메모리 버스의 병목현상과 프로세서 간 통신속도를 모두 줄인다.

③ 분산 가상메모리(distributed virtual memory) 구조는 논리적으로는 단일 공유 메모리 구조와 동일하게 사용할 수 있지만 물리적으로는 다중 분리 메모리(multiple disjoint memory) 구조로 구축될 수 있다.

④ 비공유 구조는 공유 디스크 구조에 비해 한 프로세서에서 다른 프로세서가 관리하는 비지역(non-local) 디스크 접근에 대한 비용이 높을 수 있다.

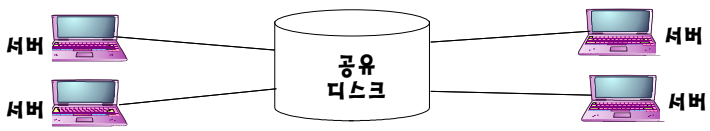
♤ **병렬 데이터베이스 구조**

// 공유 메모리(shared memory)



- 여러 프로세서 사이의 통신은 공유 메모리를 통해서 이루어진다.
- 프로세서들은 공유 메모리를 점유하기 위해 서로 경쟁하게 된다.
- 프로세서들과 메모리 사이에 통신량이 많아지면 지연시간이 길어 질 수 있다.
- 이런 단점은 '연결망 고속화, 캐시메모리, 기억장치 인터리빙' 등으로 보완할 수 있다.

// 공유 디스크(shared disk)



- 여러 개의 서버가 하나의 저장소를 공유한다.(저장소 병목현상 발생)
- 시스템에 있는 물리적 하드디스크로 다른 시스템에서 원격으로 접근할 수 있다.
- 공유 메모리 구조에 비해 병목현상과 프로세서 간 통신 속도가 **모두 줄어드는 것은 아니다.**

// 비공유(shared nothing) 구조

- shared nothing은 "아무것도 공유하지 않는다"라는 의미이다.
  → 네트워크 이외의 모두 자원을 분리하는 방식이다.
- 비공유 구조는 하나의 서버에 하나의 저장소가 딸려 있는 구조이다.
- 비공유 구조는 저장소가 병목현상이 되는 것을 방지한다.
  → 비공유 구조 세트(set)에 비례해서 처리율이 증가한다.
  → 병목현상은 여러 개의 서버가 하나의 저장소(디스크)를 공유할 때 발생될 수 있다.
- 비공유 구조는 구글(google)에서 그 유효성을 증명하였다.
  → 구글은 자사가 개발한 비공유 구조를 샤딩(sharding)이라고 부른다.

// 비공유 구조의 단점(예 : 도 단위로 "DB 서버 + 저장소" 세트를 갖춘 경우)

- 경기도 데이터를 가진 DB 서버는 경기도 데이터만 접근할 수 있다.
  → 강원도 데이터를 가진 DB 서버는 강원도 데이터만 접근할 수 있다.
  → 제주도 데이터를 가진 DB 서버는 제주도 데이터만 접근할 수 있다.
  → 즉, 다른 지역의 데이터에 접근하기 위해서는 별도의 비용이 필요하다.
- 만약, 도별 인구를 합산해서 우리나라 전체 인구를 합산해야 하는 경우
  → 각 세트로부터 도별 인구를 가져와서 집계하는 별도의 서버가 필요하다.

// 분산 가상메모리(distributed virtual memory) 구조

- 분산 가상메모리는 가상 메모리가 지리적으로 분산되어 있는 구조이다.
- 논리적으로는 단일 공유 메모리 구조와 동일하게 사용할 수 있다.
  → 사용자는 메모리가 지리적으로 분산되어 있어도 공유 메모리처럼 사용할 수 있다.
- 물리적으로는 다중 분리 메모리(multiple disjoint memory) 구조로 구축될 수 있다.
  → 메모리를 실제로 지리적으로 여러 곳에 배치할 수 있다는 것이다.
- **예** : 메모리를 한국, 미국, 영국, 중국 등에 배치할 수 있다는 것이다.

정답 : ②