2019-2

CSE4095-1

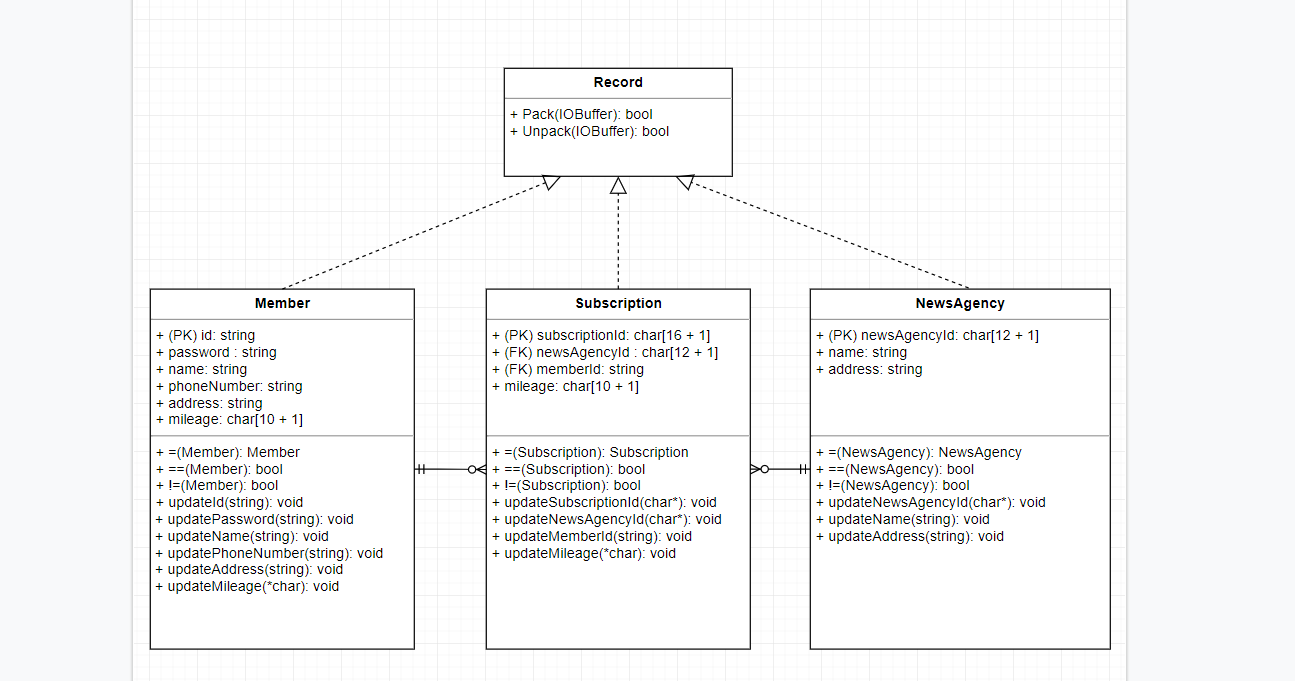
**File Processing**

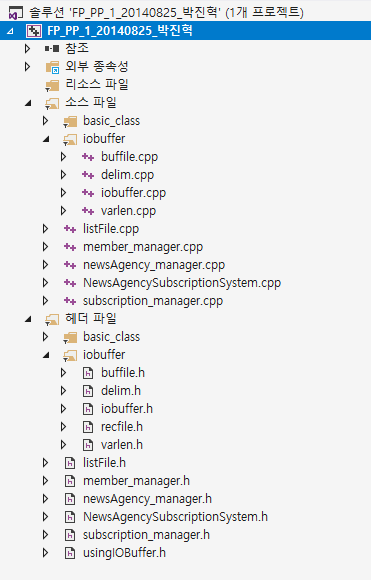
**Programming Project #1**

20140825

박진혁

**INDEX**

1. 프로젝트 진행 중 필요한 가정
2. 구현 클래스의 다이어그램(UML specification)
3. 프로젝트 구성 파일 및 자료구조
   1. basic\_class
   2. Iobuffer
   3. class\_manager
   4. 그 외 파일
4. 프로젝트 요구 사항
   1. “1.1 Basic class”의 구현
   2. “1.2 Adding methods to basic classes”의 구현
   3. “1.3 Using IOBuffer”의 구현
   4. “1.4 Deleting and Updating Records”의 구현
5. 연습문제 (6장 21~25번)
6. **프로젝트 진행 중 필요한 가정**
   1. 레코드의 필드들은 ‘|’로 구분하여 기록한다.
   2. 입력 받는 레코드에는 \*나 공백문자가 포함되지 않는다.
   3. 입력 받는 레코드 중, 고정길이 필드에 대해서는 길이에 맞는 input이 들어온다.
   4. 입력 받는 레코드 중, 각 레코드 타입의 id에 해당하는 필드는 기존의 레코드들과 중복된 값이 입력되지 않는다.
7. **구현 클래스의 다이어그램(UML specification)**
8. **프로젝트 구성 파일 및 자료구조**



* 1. **basic\_class :** 프로젝트 내 별도의 ‘basic\_class’ 디렉토리를 생성해 관리
     1. **record.h**
        + Member, NewsAgency, Subscription이 구현하게 될 추상 클래스인 Record를 선언하고, 그 멤버 변수로는 가상 함수로서의 Pack과 Unpack을 선언하여 하위 클래스에서 두 함수를 필수적으로 구현할 수 있도록 하였다.
     2. **member.cpp과 member.h** 
        + member.h에서 Member 클래스를 선언하고, 그에 속하는 멤버 변수들과 메소드들을 선언하였다. friend 함수인 operator <<과 operator >> 역시 헤더 파일에 선언하였다. member.cpp에서 해당 함수들을 실제로 구현하였다.
     3. **newsAgency.cpp과 newsAgency.h**
        + newsAgency.h에서 NewsAgency 클래스를 선언하고, 그에 속하는 멤버 변수들과 메소드들을 선언하였다. friend 함수인 operator <<과 operator >> 역시 헤더 파일에 선언하였고, newsAgency.cpp에서 해당 함수들을 실제로 구현하였다.
     4. **subscription.cpp과 subscription.h**
        + subscription.h에서 Subscription 클래스를 선언하고, 그에 속하는 멤버 변수들과 메소드들을 선언하였다. friend 함수인 operator <<과 operator >> 역시 헤더 파일에 선언하였고, subscription.cpp에서 해당 함수들을 실제로 구현하였다.
  2. **Iobuffer :** 프로젝트 내 별도의 ‘iobuffer’ 디렉토리를 생성해 관리
     1. **iobuffer.cpp과 iobuffer.h** 
        + IOBuffer 클래스가 구현되어 있으며, 가상 메소드들을 포함하여 후에 IOBuffer 클래스를 상속받을 클래스에서 구현해야 할 메소드들에 대해 명세하고 있다.
     2. **varlen.cpp과 varlen.h**
        + IOBuffer 클래스를 상속받는 VariableLengthBuffer를 구현하였다. 구현된 클래스(VariableLengthBuffer)는 레코드의 가변 길이 필드 데이터들을 저장하는 데 사용된다.
     3. **delim.cpp과 delim.h** 
        + VariableLengthBuffer 클래스를 상속받아 DelimFieldBuffer를 구현하였다. DelimFieldBuffer는 Delim이라는 문자를 멤버 변수로 가지는데, 이 문자가 ‘구분자’로서의 역할을 하여 필드들을 구분짓는데 사용한다.
     4. **buffile.cpp과 buffile.h** 
        + IOBuffer와 fstream을 멤버 변수로 갖는 Buffile 클래스를 구현하였다. 이 클래스의 객체는 파일의 내용을 버퍼로 옮기거나(Read), 버퍼의 내용을 파일로 옮기는 작업(Write) 모두를 수행할 수 있다.
     5. **recfile.h**
        + BufferFile 클래스를 상속받는 템플릿 클래스인 RecordFile 클래스를 구현하였다. 클래스 객체는 템플릿 자료형으로 입력되는 RecType에 따라 해당 레코드와 BufferFile 클래스를 연결해준다. 구현된 메소드로는 파일의 내용을 버퍼로 읽어들여 레코드에 이를 Unpack하는 Read와 레코드의 내용을 버퍼에 Pack한 뒤 이를 파일에 작성하는 Write가 있다.
  3. **class\_manager**
     1. **member\_manager.cpp과 member\_manager.h** : Member 타입의 레코드들과 관련하여 이를 관리하는 함수들을 구현.
        + **memberSerach** : 파라미터로 string 타입의 id를 받아 id와 일치하는 Member 객체를 탐색하며, 이는 .dat 파일의 레코드들에 대한 순차탐색으로 이루어진다. 일치하는 Member 객체가 없다면 관련 메시지를 띄우고, 있다면 연산자 ‘<<’를 통해 Member 객체 내용을 출력한다.
        + **memberInsert** : 콘솔 입력을 통해, id를 입력 받은 뒤, 이와 일치하는 Member가 없을 경우에만, 추가적으로 정보를 입력받아 새로운 Member 객체를 만든 뒤, 이를 saveMemberRecord 함수의 파라미터로 넣어주어 .dat 파일에 삽입한다.
        + **memberUpdate** : 콘솔 입력을 통해, id를 입력 받은 뒤, 이와 일치하는 Member가 있을 경우에만, 기존의 Member 레코드를 삭제한 뒤, 추가적으로 정보를 입력 받는다. 이를 통해 새로운 Member 객체를 만들고 saveMemberRecord 함수의 파라미터로 넣어주어 .dat 파일에 삽입한다.
        + **memberDelete** : 파라미터로 string 타입의 id를 받아, 이와 일치하는 Member 객체를 찾아, 이를 makeDummyMember 함수를 통해 dummy record로 만든 뒤, 같은 위치에 다시 삽입해준다. 만약 recaddr 위치 상 앞에 있는 레코드가 dummy record, 즉 삭제된 데이터라면 앞의 레코드와 현재 레코드를 합쳐 하나의 dummy record로 바꿔 입력한다. 이 때, 참조무결성을 위해, 해당 id와 일치하는 값을 memberId로 갖는 Subscription 객체를 같이 삭제한다.
        + **getMemberRecordLen** : Member 타입의 m을 파라미터로 받아, 해당 레코드가 실제로 .dat 파일에 Write 되게 될 때에 얼마만큼의 공간을 차지하게 될지를 계산하여 그 값을 반환한다.
        + **makeDummyMember** : Member 타입의 m과 만들어야 하는 레코드의 길이를 파라미터로 받아, 그 길이에 맞는 레코드가 .dat 파일에 쓰여지도록, Member의 가변 길이 필드 값을, 삭제 데이터임을 나타내는 ‘\*’로 채운다.
        + **saveMemberRecord** : Member 타입의 newMember 변수를 파라미터로 받은 뒤, 전체 DB(.dat 파일)을 순차 탐색하면서, newMember의 레코드가 들어갈 자리를 찾아 저장한다. 탐색 중 dummy record를 만나게 되고, 또 해당 dummy record의 길이가 또다른 최소 단위의 dummy record와 newMember로 분리될 수 있다면, 해당 위치에 레코드를 작성하고, 아니라면 DB의 마지막 레코드 다음 자리에 레코드를 기록한다.
     2. **newsAgency\_manager.cpp과 newsAgency\_manager.h :** NewsAgency 타입의 레코드들과 관련하여 이를 관리하는 함수들을 구현.
        + **newsAgencySerach** : 파라미터로 string 타입의 id를 받아 id와 일치하는 NewsAgency 객체를 탐색하며, 이는 .dat 파일의 레코드들에 대한 순차탐색으로 이루어진다. 일치하는 NewsAgency 객체가 없다면 관련 메시지를 띄우고, 있다면 연산자 ‘<<’를 통해 NewsAgency 객체 내용을 출력한다.
        + **newsAgencyInsert** : 콘솔 입력을 통해, id를 입력 받은 뒤, 이와 일치하는 NewsAgency가 없을 경우에만, 추가적으로 정보를 입력받아 새로운 NewsAgency 객체를 만든 뒤, 이를 saveNewsAgencyRecord 함수의 파라미터로 넣어주어 .dat 파일에 삽입한다.
        + **newsAgencyUpdate** : 콘솔 입력을 통해, id를 입력 받은 뒤, 이와 일치하는 NewsAgency가 있을 경우에만, 기존의 NewsAgency 레코드를 삭제한 뒤, 추가적으로 정보를 입력 받는다. 이를 통해 새로운 NewsAgency 객체를 만들고 saveNewsAgencyRecord 함수의 파라미터로 넣어주어 .dat 파일에 삽입한다.
        + **newsAgencyDelete** : 파라미터로 string 타입의 id를 받아, 이와 일치하는 NewsAgency 객체를 찾아, 이를 makeDummyNewsAgency 함수를 통해 dummy record로 만든 뒤, 같은 위치에 다시 삽입해준다. 만약 recaddr 위치 상 앞에 있는 레코드가 dummy record, 즉 삭제된 데이터라면 앞의 레코드와 현재 레코드를 합쳐 하나의 dummy record로 바꿔 입력한다. 이 때, 참조 무결성을 위해, 해당 id와 일치하는 값을 newsAgencyId로 갖는 Subscription 객체를 같이 삭제한다.
        + **getNewsAgencyRecordLen** : NewsAgency 타입의 n을 파라미터로 받아, 해당 레코드가 실제로 .dat 파일에 Write 되게 될 때에 얼마만큼의 공간을 차지하게 될지를 계산하여 그 값을 반환한다.
        + **makeDummyNewsAgency** : NewsAgency 타입의 n과 만들어야 하는 레코드의 길이를 파라미터로 받아, 그 길이에 맞는 레코드가 .dat 파일에 쓰여지도록, NewsAgency의 가변 길이 필드 값을, 삭제 데이터임을 나타내는 ‘\*’로 채운다.
        + **saveNewsAgencyRecord** : NewsAgency 타입의 newNewsAgency 변수를 파라미터로 받은 뒤, 전체 DB(.dat 파일)을 순차 탐색하면서, newNewsAgency의 레코드가 들어갈 자리를 찾아 저장한다. 탐색 중 dummy record를 만나게 되고, 또 해당 dummy record의 길이가 또다른 최소 단위의 dummy record와 newNewsAgency로 분리될 수 있다면, 해당 위치에 레코드를 작성하고, 아니라면 DB의 마지막 레코드 다음 자리에 레코드를 기록한다.
     3. **subscription.cpp과 subscription.h** : Subscription 타입의 레코드들과 관련하여 이를 관리하는 함수들을 구현
        + **subscriptionSerach** : 파라미터로 string 타입의 id를 받아 id와 일치하는 Subscription 객체를 탐색하며, 이는 .dat 파일의 레코드들에 대한 순차탐색으로 이루어진다. 일치하는 Subscription 객체가 없다면 관련 메시지를 띄우고, 있다면 연산자 ‘<<’를 통해 Subscription 객체 내용을 출력한다.
        + **subscriptionInsert** : 콘솔 입력을 통해, id를 입력 받은 뒤, 이와 일치하는 Subscription이 없을 경우에만, 추가적으로 정보를 입력받아 새로운 Subscription 객체를 만든 뒤, 이를 saveSubscriptionRecord 함수의 파라미터로 넣어주어 .dat 파일에 삽입한다.
        + **subscriptionUpdate** : 콘솔 입력을 통해, id를 입력 받은 뒤, 이와 일치하는 Subscription이 있을 경우에만, 기존의 Subscription 레코드를 삭제한 뒤, 추가적으로 정보를 입력 받는다. 이를 통해 새로운 Subscription 객체를 만들고 saveSubscriptionRecord 함수의 파라미터로 넣어주어 .dat 파일에 삽입한다.
        + **subscriptionDelete** : 파라미터로 string 타입의 id를 받아, 이와 일치하는 Subscription 객체를 찾아, 이를 makeDummySubscription 함수를 통해 dummy record로 만든 뒤, 같은 위치에 다시 삽입해준다. 만약 recaddr 위치 상 앞에 있는 레코드가 dummy record, 즉 삭제된 데이터라면 앞의 레코드와 현재 레코드를 합쳐 하나의 dummy record로 바꿔 입력한다.
        + **getSubscriptionRecordLen** : Subscription 타입의 s를파라미터로 받아, 해당 레코드가 실제로 .dat 파일에 Write 되게 될 때에 얼마만큼의 공간을 차지하게 될지를 계산하여 그 값을 반환한다.
        + **makeDummySubscription** : Subscription 타입의 s와 만들어야 하는 레코드의 길이를 파라미터로 받아, 그 길이에 맞는 레코드가 .dat 파일에 쓰여지도록, Subscription의 가변 길이 필드 값을, 삭제 데이터임을 나타내는 ‘\*’로 채운다.
        + **saveSubscriptionRecord** : Subscription 타입의 newSubscription 변수를 파라미터로 받은 뒤, 전체 DB(.dat 파일)을 순차 탐색하면서, newSubscription의 레코드가 들어갈 자리를 찾아 저장한다. 탐색 중 dummy record를 만나게 되고, 또 해당 dummy record의 길이가 또다른 최소 단위의 dummy record와 newSubscription로 분리될 수 있다면, 해당 위치에 레코드를 작성하고, 아니라면 DB의 마지막 레코드 다음 자리에 레코드를 기록한다.
        + **subscriptionDeleteWithMemberId :** string 타입의 memberId를 파라미터로 받아, 순차 탐색을 통해 전체 Subscription 레코드 중 자신의 멤버변수 memberId가 파라미터 memberId와 일치하는 레코드를 찾아 삭제한다. 이 경우, DB 전체를 반드시 모두 탐색하게 하여 조건에 만족하는 모든 Subscription 레코드를 삭제한다.
        + **susbcriptionDeleteWithNewsAgencyId :** string 타입의 newsAgencyId를 파라미터로 받아, 순차 탐색을 통해 전체 Subscription 레코드 중 자신의 멤버변수 newsAgencyId가 파라미터 newsAgencyId와 일치하는 레코드를 찾아 삭제한다. 이 경우 역시, DB 전체를 반드시 모두 탐색하게 하여 조건에 만족하는 모든 Subscription 레코드를 삭제한다.
  4. **그 외 파일**
     1. **listFile.cpp과 listFile.h**
        + **generate[Member/NewsAgency/Subscription]List :** 무작위의 Member, NewsAgency, Subscription 레코드를 서로 중복되지 않도록 원하는 개수만큼 생성한 내용이 담긴 txt 파일을 작성한다.
        + **generateRandomNumber** : digit을 파라미터로 받아, 해당 digit 만큼의 자리수를 가지는 숫자를 문자화하여 string으로 반환한다.
        + **intToString** : integer와 digit을 파라미터로 받아, 주어진 integer를 digit 만큼의 길이를 갖는 string으로 변환하여 반환한다.
        + **<T>get[List/Data]FileName** : 템플릿 클래스 T에 따라 그에 맞는 list .txt/.dat 파일의 이름을 반환한다.
     2. **usingIOBuffer.h**
        + iobuffer 디렉토리 내부의 헤더 파일들을 한번에 include 할 수 있게 해주는 헤더 파일이다.
     3. **NewsAgencySubscriptionSystem.h** : NewsAgencySubscriptionSystem.cpp 파일에서 사용할 함수들을 선언하고, 관련 템플릿 함수들을 구현.
        + <**T>showData** : 템플릿 클래스 T에 맞는 .txt 파일을 읽어 들여 10개의 레코드를 콘솔에 출력한다.
        + **<T>dataTest** : 템플릿 클래스 T에 맞는 .txt 파일을 읽어 들여 파일 내 모든 레코드들을 .dat 파일을 만들어 저장한다. 이미 해당 .dat 파일이 있는경우, .dat파일을 새롭게 만든다.
     4. **NewsAgencySubscriptionSystem.cpp** : 실제 콘솔창에서 명령어를 입력 받아 작업을 수행하는 main 함수를 포함
        + **printMenu** : 콘솔창에 사용자가 선택할 수 있는 작업의 menu들을 띄운다
        + **executeSystem** : 사용자의 입력에 따른 작업들을 수행한다.

1. **프로젝트 요구 사항**
   1. **“1.1 Basic class”의 구현**
      1. ‘basic\_class’ 디렉토리 내부에 있는 파일들을 통해 Member, NewsAgency, Subscription 클래스들을 구현하고 그에 따른 함수들(복사 생성자, 대입 연산자, ==, !=, update메소드)도 모두 구현하였다.
      2. listFile.cpp과 listFile.h에 구현한 함수들을 통해 각 클래스 별 레코드들이 각각 1000개, 1000개, 10000개씩 담겨있는 txt 파일을 생성했다. 특히 1대 다 관계를 구현하기 위해, Subscription의 경우 실존하는 Member, NewsAgency 레코드들과 연결 지어질 수 있도록 레코드를 생성하였다.
   2. **“1.2 Adding methods to basic classes”의 구현**
      1. ‘basic\_class’ 디렉토리 내부 파일들에 구현되어 있는 각 클래스에 friend 함수 ‘operator >>’, ‘operator <<’를 추가하여 적절한 형식으로 각 인스턴스들이 콘솔창에 출력될 수 있도록 하였다.
      2. NewsAgencySubscriptionSystem.h에 템플릿 함수인 showData를 구현하여, 각 basic class 타입에 맞는 showMember, showNewsAgency, showSubscription 프로그램이 콘솔창에서 잘 작동할 수 있도록 하였다.
   3. **“1.3 Using IOBuffer”의 구현**
      1. Record.h 파일에 추상 클래스 Record를 만들고, 해당 클래스의 멤버함수로서 Pack과 Unpack을 가상함수로 구현하였다. Member, NewsAgency, Subscription 각 클래스들은 Record 클래스를 상속받아 두 함수를 모두 overwriting 하여 각 클래스에 맞게 구현하였다.
      2. NewsAgencySubscriptionSystem.h에 템플릿 함수인 dataTest를 구현하여, 각 basic class 타입에 맞는 MemberTest, NewsAgencyTest, SubscriptionTest 프로그램이 콘솔창에서 잘 작동할 수 있도록 하였다.
   4. **“1.4 Deleting and Updating Records”의 구현**
      1. 각 클래스 별로 레코드들을 관리하기 위하여 MemberManager, NewsAgencyManager, SubscriptionManager 클래스를 만들고, 해당 클래스들에 각각 search, insert, update, delete의 기능을 하는 메소드들을 구현하였다.
      2. NewsAgencySubscriptionSystem.cpp의 executeSystem 함수 내에 각 manager 클래스 객체를 선언한 뒤, 각 객체들의 메소드들을 사용하였다.
      3. Member와 NewsAgency 레코드에 대한 삭제 시, 해당 레코드의 id와 같은 memberId 또는 newsAgencyId를 가지고 있는 모든 Subscription 레코드에 대해서도 레코드 삭제를 수행해 주어 참조 무결성을 유지하도록 했다.
2. **연습문제 (6장 21~25번)**
   1. 21번 : 삭제된 레코드에 대해 각 필드 값을 ‘\*’로 채워주어 삭제된 레코드임을 표시한다. 레코드 길이가 고정되어 있으므로, 순차 탐색 시 해당 레코드가 삭제되었음을 인지하고 다음 레코드로 이동해 탐색을 계속할 수 있다.
   2. 22번 : 레코드 순차 탐색 시, PK(ID)가 \*를 포함하고 있다면 해당 레코드를 삭제된 것으로 파악하고, 삭제된 레코드의 자리에 현재 레코드를 새롭게 작성할 수 있다.
   3. 23번 : 가변길이 레코드 파일을 삭제하게 되면, 각 필드 값을 ‘\*’로 채워주어 삭제된 레코드임을 표시한다. 이 경우 필드별로 구분자를 사용하여 구분자의 개수를 통해 필드의 개수를 세어, 삭제된 해당 가변길이 레코드의 다음 레코드를 탐색한다.
   4. 24번 : 레코드 순차 탐색 시, PK(ID)가 \*를 포함하고 있다면 해당 레코드를 삭제된 것으로 파악하고, 삭제된 레코드의 크기와 현재 레코드의 크기를 비교하여, 현재 레코드의 크기가 삭제된 레코드의 크기보다 작다면 해당 위치에 새롭게 레코드를 작성할 수 있다.
   5. 25번 : 기존의 레코드를 Delete 메소드를 통해 삭제한 뒤, 새로운 레코드의 정보를 갖는 새로운 객체를 만든다. 새롭게 만들어진 레코드의 길이와 최소한의 길이로 만들 수 있는 더미 레코드, 즉 아무런 정보가 없지만 삭제되었음을 알 수 있는 레코드의 길이의 합이 기존에 삭제된 레코드가 차지하던 길이보다 크지 않다면, 기존에 삭제된 레코드의 자리에 새롭게 만들어진 레코드를 작성하고, 빈 공간은 또다른 더미 레코드를 만들어 작성할 수 있다. 만약 더미레코드와 새 레코드의 길이의 합이 기존 삭제된 레코드의 길이보다 크다면, 해당 위치에는 새 레코드를 삽입할 수 없다.