

〈레포트〉

설계 과제

Part 1

강의명: 데이터베이스시스템

담당교수: 강현철 교수님

작성자: 신성섭

학번: 20226XX2

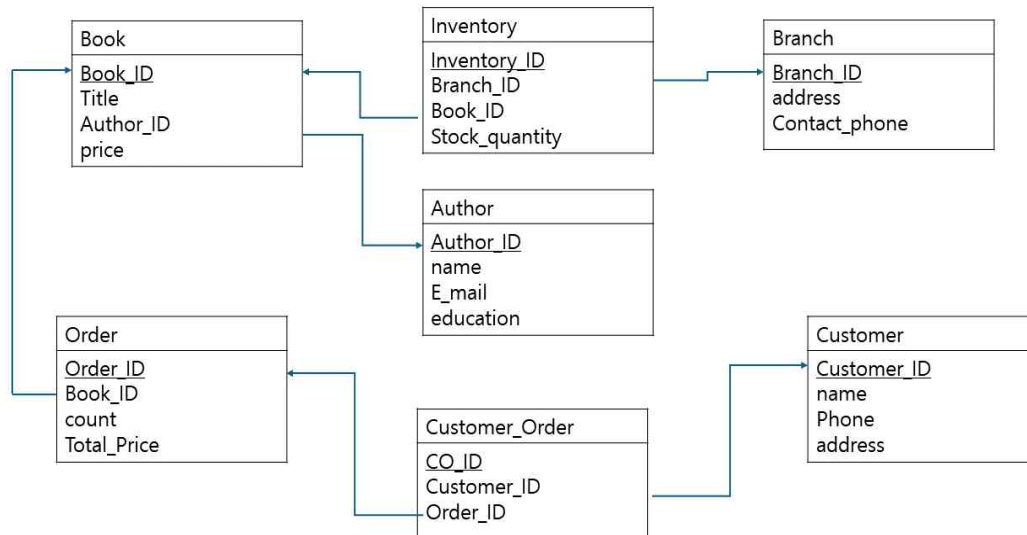
학과: 소프트웨어학부

1. 데이터 및 테스트 시나리오

- 설계 응용 분야

온라인 서점 운영을 위한 데이터 베이스

- 테이블 스키마 다이어그램



- 테이블 데이터

* BookTable

Book_ID (5)	Title(15)	Author_ID(5)	price(5)
10001	Secret Garden	99999	35000
10002	Lost Horizon	99998	22000
10003	Starry Night	99997	18000
10004	New year	99996	45000
10005	Happy Bear	99995	15000
20001	Scarry World	99994	35000
20002	Rain Rain Rain	99993	10000
20003	Big Boom	99992	30000
20004	Boo king	99991	15000

* Inventory

Inventory_ID (5)	Branch_ID(5)	BOOK_ID(5)	stock_quantity(5)
50001	30001	10001	100
50002	30002	10001	150
50003	30001	10002	1000
50004	30003	10003	50
50005	30004	10004	1500
50006	30002	10004	1000
50007	30003	10001	600
50008	30004	20001	300
50009	30005	20003	300
50010	30006	20002	1200
50011	30007	20002	400

- 테스트 시나리오

* block의 size가 100bytes 라고 가정한다.

Book Table의 한 레코드 크기는 5+15+5+5=30 이므로 $\text{int}(100/30) = 3$, 즉 blocking factor 가 3이 된다. 블록 단위 I/O을 이용하여 올바른 결과를 도출하는 지 확인하기 위해서 여러 가지 상황에서 테스트를 진행 해본다.

1) 테이블 생성

create table book

book_id char(5), title char(15), author_id char(5), price char(5)

을 통해 book.txt가 올바르게 형성되는 지, 메타데이터가 DB에 잘 저장되는 지를 확인한다.

2) 삽입

book의 데이터를 하나씩 매번 입력 하면서 올바르게 삽입되는 지를 확인한다.

삽입하는 초기 데이터들은 위의 book 테이블 표와 같다.

book의 튜플이 삭제된 상황에서 삭제된 튜플의 위치에 올바르게 삽입되는 지를 확인한다.

삭제된 위치에 튜플에 올바르게 삽입되었는지 확인을 쉽게 하기 위해 삽입하는 데이터는 다음으로 한다.

Book_ID (5)	Title(15)	Author_ID(5)	price(5)
77777	Good news	99999	12300
77776	Sad world	99998	15000

3) 삭제

delete from book

where book_id=20001

와 같은 명령어를 실행하여 튜플이 올바르게 삭제되는 지를 확인한다.

삭제하는 튜플은 다음과 같다.

Book_ID (5)	Title(15)	Author_ID(5)	price(5)
10001	Secret Garden	99999	35000
10004	New year	99996	45000
20001	Scarry World	99994	35000

4) 검색

Book 테이블의 튜플이 모두 차 있는 상태에서

select *

from book

where True

를 호출하여 검색 결과를 확인한다.

테이블이 가득 찬 상태에서

select book_id, price

from book

where book_id=20001

를 호출하여 1개의 튜플을 검색을 올바르게 수행하는 지를 확인한다.

튜플이 삭제된 후 삭제한 튜플을 검색하는 질의를 테스트 해본다.

select *

from book

where book_id=10004

를 호출하여 파일에서 book_id가 10004인 튜플을 찾지 못하고 빈 테이블을 반환하면 성공이다.

위의 상황뿐 아니라 더 다양한 상황들을 개발 하면서 테스트 해본다.

테이블의 메타데이터가 올바르게 저장되는 지는 workBench에서 직접 확인한다.

삽입, 삭제가 올바르게 수행되었는지 검증은 메모장(레코드들이 저장되어 있는 파일)을 켜서 직접 확인해본다.

검색의 경우는 콘솔창에 나오는 정보가 테이블에 저장된 정보(메모장안 데이터)와 일치하는 지를 확인한다.

2. 설계

- 프로그램 전반적 흐름

프로그램을 시작하면 기능을 선택하는 메뉴창을 유저한테 보여준다. 유저가 선택할 수 있는 기능은 1. 관계 테이블 생성(create table), 2. 튜플 삽입(insert record) 3. 튜플 삭제(delete tuple), 4. 튜플 검색(select tuple), 5. 프로그램 종료(exit program) 이다. 각각의 기능에 해당하는 작업을 수행 한 후 5번 프로그램 종료를 입력 받을 때까지 위의 과정을 반복한다.

- JAVA 프로그램에서 구현해야 할 기능들

- 1) MySQL 사용을 위해 JDBC 연결 & JDBC close 하는 기능
- 2) 유저한테 기능 선택 메뉴창 보여주기 & 선택된 기능 호출 하는 기능
- 3) 관계 테이블 정보를 입력 받은 후, 입력받은 테이블 이름의 txt 파일을 생성하는 기능
- 4) 입력 받은 테이블 정보를 분석하여, sql의 insert문을 이용해 metaData 스키마들에 적절한 정보를 저장하는 기능
- 5) 유저한테서 입력 받은 각각의 요구 상황(삽입, 삭제, 검색)에 대하여, 입력 받은 정보를 분석하고, sql의 select문을 이용해 필요한 metaData들을 가져오는 기능.
- 6) 가져온 메타 정보를 이용해 블록단위 I/O을 이용한 레코드 삽입, 삭제, 검색 각각을 수행하는 기능
- 7) 검색의 경우 가져온 레코드들을 튜플 단위, 컬럼 단위로 추출하고 유저가 요구한 질의의 결과셋을 구성한 후 콘솔에 출력하는 기능.
- 8) 프로그램을 종료하는 기능.

- 테이블 저장 방법

각 테이블 당 하나의 txt 파일에 저장한다. 한 테이블의 모든 레코드들은 하나의 텍스트 파일에 저장한다.

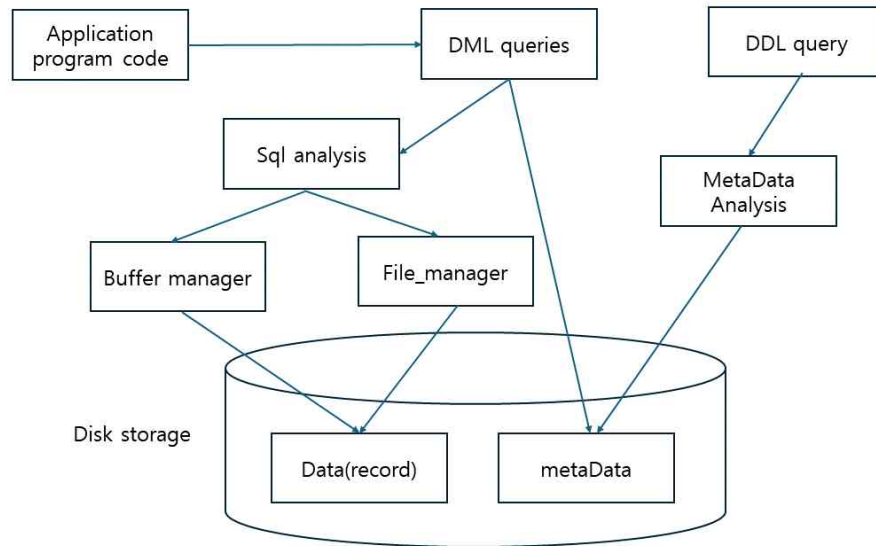
txt 파일의 맨 앞 부분에는 record의 size만큼을 차지하는 header가 들어가 있다. header에는 삭제된 한 record의 위치(바이트 값)을 가리키는 정보가 저장되어있다. 삭제된 record의 위치에는 position x 형식으로 x에 다른 삭제된 record의 위치(바이트 값)이 들어가 있고, 다른 것을 가리킬게 없는 삭제된 record의 경우 position -1 (NULL)값이 들어가 있다.

파일명은 테이블이름.txt 파일로 하고 편의상 파일의 저장위치는 소스코드와 동일한 위치로 한다.

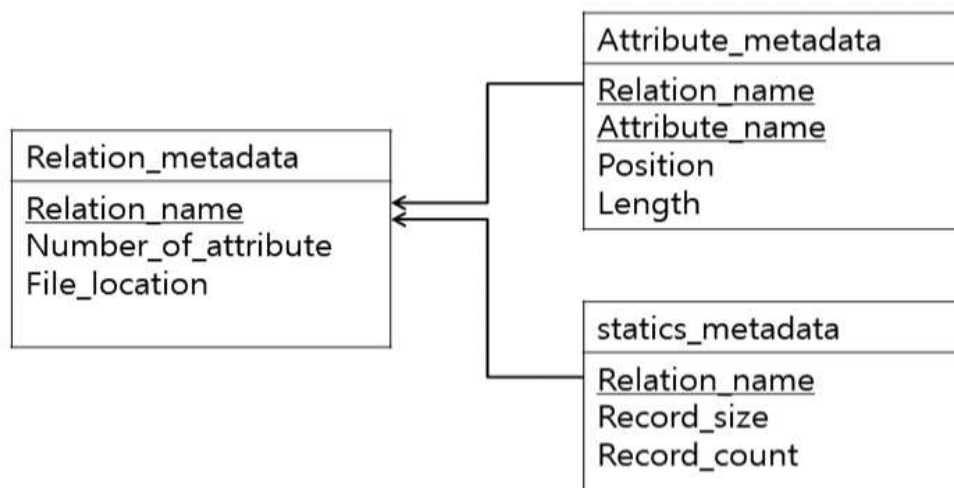
-블록단위 I/O 구현을 위해 필요한 클래스,

Java 코드에서 블록단위 I/O을 구현하기 위해 RandomAccessFile 클래스와 file.seek(), file.read(new byte[size]), file.writeBytes() 등을 이용한다.

- 시스템 모듈 구성 블록 다이어그램



- 메타 데이터 관계형 데이터베이스 스키마



-메타데이터 저장 용 테이블 SQL create table 문

*relation_metadata 테이블

```
CREATE TABLE `relation_metadata` (  
  `Relation_name` VARCHAR(30) NOT NULL,  
  `Number_of_attribute` INT ,  
  `File_location` VARCHAR(50),  
  PRIMARY KEY (`Relation_name`));
```

*attribute_metadata 테이블

```
CREATE TABLE `attribute_metadata` (  
  `Relation_name` VARCHAR(30) NOT NULL,  
  `Attribute_name` VARCHAR(50) NOT NULL,  
  `Positional` INT,  
  `Length` INT,  
  PRIMARY KEY (`Relation_name`, `Attribute_name`),  
  CONSTRAINT `Relation_name`  
    FOREIGN KEY (`Relation_name`)  
    REFERENCES `relation_metadata` (`Relation_name`)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE);
```

*statics_metadata 테이블

```
CREATE TABLE `statics_metadata` (  
  `Relation_name` VARCHAR(30) NOT NULL,  
  `Record_size` INT ,  
  `Record_count` INT ZEROFILL ,  
  PRIMARY KEY (`Relation_name`),  
  CONSTRAINT `static_foreignkey`  
    FOREIGN KEY (`Relation_name`)  
    REFERENCES `relation_metadata` (`Relation_name`)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE);
```

- 관계 DB 시스템의 기능별 사용자 인터페이스

* **파란색 글자**가 콘솔에 나오면 사용자가 **붉은 글자** 형식으로 입력

1) 테이블 생성 인터페이스

메뉴창에서 1(관계 테이블 생성(create table)) 을 입력한다.

create table 이 콘솔에 나오면 사용자가 생성하기를 원하는 테이블 명을 입력하고 엔터를 친다.

그 후 생성하기 원하는 컬럼들을 입력하세요.(컬러명 char(n) 형식 입력, 여러 컬럼 입력 희망시 , 로 구분) 이 콘솔에 뜨면 사용자가 테이블의 컬럼들을 입력 해준다.

사용 예시)

create table **book** (엔터 입력)

생성하기 원하는 컬럼들을 입력하세요.(컬러명 char(n) 형식 입력, 여러 컬럼 입력 희망시 , 로 구분)

book_id char(5), title char(15), author_id char(5), price char(5) (엔터 입력)

2) 튜플 삽입

메뉴창에서 2(튜플 삽입)을 입력한다.

insert into 가 콘솔에 나오면 사용자가 튜플을 삽입하기를 원하는 테이블 명을 입력하고 엔터를 친다.

그 후 values 가 콘솔에 뜨면 사용자가 레코드들의 모든 속성을 ,로 구분하여 입력하고 엔터를 친다.

사용 예시)

insert into **book** (엔터 입력)

values 10004, New year, 99996, 45000 (엔터 입력)

3) 튜플 삭제

메뉴창에서 3(튜플 삭제)를 입력한다.

delete from 이 콘솔에 나오면 삭제하는 튜플이 속한 테이블명을 입력하고 엔터를 친다.

그 후 콘솔에 where 이 나오면 튜플의 조건을 입력하고 엔터를 친다. 이때 튜플의 조건은 컬럼 명=값 형식으로 입력한다.

사용 예시)

delete from **book** (엔터 입력)

where book_id=20001 (엔터 입력)

4) 튜플 검색

메뉴창에서 4(튜플 검색)을 입력한다.

select 가 콘솔에 나오면 검색하려는 속성들을 ,로 구분해서 입력하고 엔터를 친다. 모든 속성을 검색하기를 원하면 *를 입력한다.

from이 콘솔에 나오면 검색하는 테이블의 이름을 입력하고 엔터를 친다.

where이 콘솔에 나오면 검색 조건을 입력한다. 이때 튜플의 검색조건은 컬럼명=값 형식으로 입력한다.

만약 모든 튜플을 검색하기를 원하면 그냥 true 를 입력한다.

시스템이 검색 조건에 맞는 튜플들을 찾은 후 콘솔에 표 형식으로 논리적 view를 출력한다.

사용 예시)

select * (엔터 입력)

from book (엔터 입력)

where book_id=20001 (엔터 입력)

출력 예시 결과)

Book_ID	Title	Author_ID	price
20001	Scarry World	99994	35000

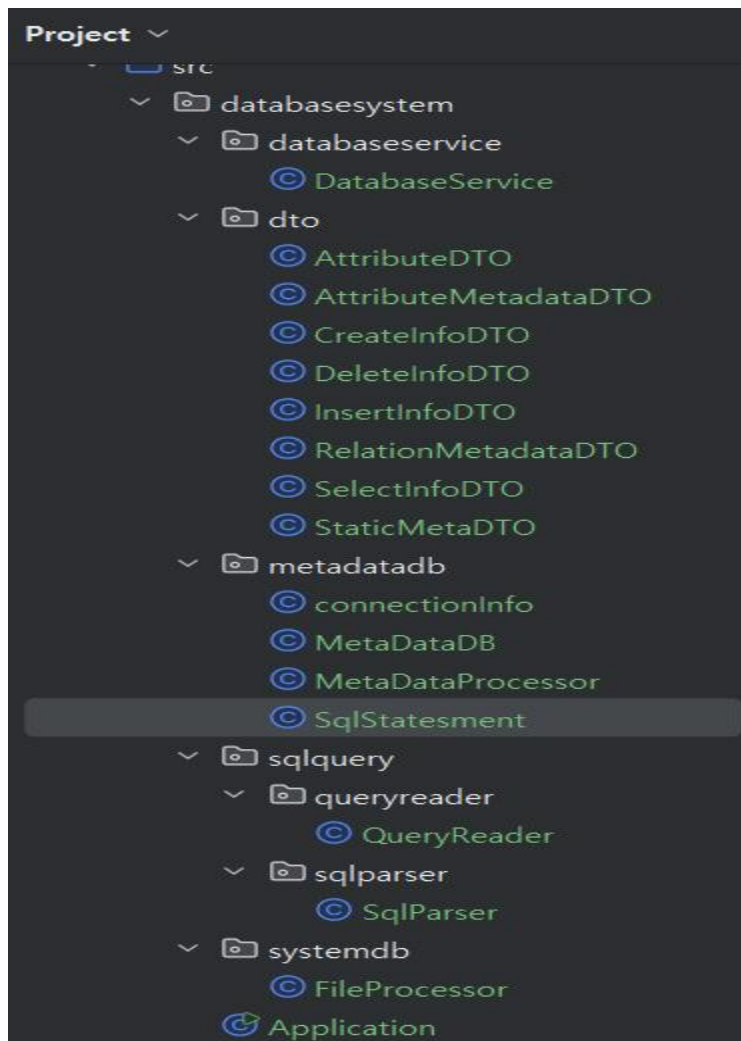
3절 구현

1. 2절 설계 계획 중 변경 사항

기존에는 삭제된 레코드의 위치를 표기 할 때 position x 형식으로 저장하려고 하였으나, 이와 같은 방식으로 저장할 경우 테이블의 데이터의 형태와 겹칠 가능성이 있다.

그래서 레코드의 위치를 표기 할 때 L\$pos x 의 형식으로 저장하는 걸로 변경하였다.

2. 디렉토리 클래스 구조



객체지향원리를 잘 살리기 위해서 기능별로 클래스를 분리하였다. 크게 DBService, DTO, meataDataDB, SQLQuery, systemDB, Application으로 분류 할 수 있다.

3. 기능별 주요 구현 로직.

4절 테스트 및 정확성 검증

1. 테이블 생성

이름과 속성들이 다른 두 테이블을 생성해 보면서 일반성을 가지는 지 확인해본다.

* 초기 메타데이터 DB 상태

	Relation_name	Number_of_attribute	File_location
*	NULL	NULL	NULL

	Relation_name	Record_size	Record_count
*	NULL	NULL	NULL

	Relation_name	Attribute_name	Positional	Length
*	NULL	NULL	NULL	NULL

1) book Table 생성 test

기능 선택:1

```
create table book
```

생성하기 원하는 컬럼들을 입력하세요.(컬럼명 char(n) 형식 입력, 여러 컬럼 입력 희망시 , 로 구분)
`book_id char(5), title char(15), author_id char(5), price char(5)`

실행 후 메타데이터 DB 상태

-relation_metadata

	Relation_name	Number_of_attribute	File_location
▶	book	4	book.txt

테이블 이름이 book이고 컬럼의 개수가 4개, book.txt 파일 경로를 가지는 것이 올바르게 저장되었다.

-statics_metadata

	Relation_name	Record_size	Record_count
▶	book	30	0000000000

book 레코드의 총 길이(5+5+5+15=30)이 올바르게 계산되어져 저장되었다.

-attribute_metadata

	Relation_name	Attribute_name	Positional	Length
▶	book	author_id	2	5
	book	book_id	0	5
	book	price	3	5
	book	title	1	15

book테이블에 있는 각 컬럼의 이름, 각각의 위치, 고정길이 컬럼의 길이가 입력 한 것과 동일하다.

2) inventory Table 생성 test

기능 선택:1

```
create table inventory
```

생성하기 원하는 컬럼들을 입력하세요.(컬럼명 char(n) 형식 입력, 여러 컬럼 입력 희망시 , 로 구분)

```
inventory_id char(5), branch_id char(5), book_id char(5), stock_quantity char(5)
```

-relation_metadata

	Relation_name	Number_of_attribute	File_location
▶	book	4	book.txt
	inventory	4	inventory.txt

inventory 테이블이 가지는 특성의 개수와 경로가 올바르게 저장되었다.

-statics_metadata

	Relation_name	Record_size	Record_count
▶	book	30	0000000000
	inventory	20	0000000000

record의 길이(5+5+5+5=20)이 올바르게 저장되었다.

-attribute_metadata

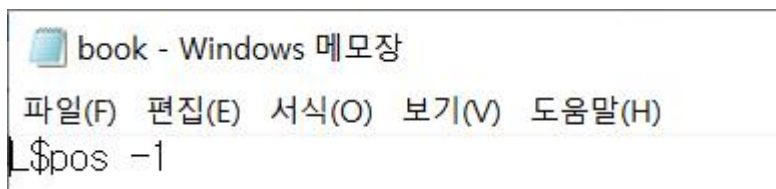
	Relation_name	Attribute_name	Positional	Length
▶	book	author_id	2	5
	book	book_id	0	5
	book	price	3	5
	book	title	1	15
	inventory	book_id	2	5
	inventory	branch_id	1	5
	inventory	inventory_id	0	5
	inventory	stock_quantity	3	5

inventory의 각 컬럼의 이름과 위치, 길이가 입력한거와 동일하게 저장되었다.

- create문을 통한 DB 파일 생성

src	2024-04-08 오후 7:58	파일 폴더	
.gitignore	2024-04-08 오후 7:45	텍스트 문서	1KB
book	2024-04-10 오후 8:24	텍스트 문서	1KB
databaseSystem1.iml	2024-04-08 오후 8:24	IML 파일	1KB
inventory	2024-04-10 오후 8:36	텍스트 문서	1KB

각 파일과 헤더가 올바르게 생성되었다.

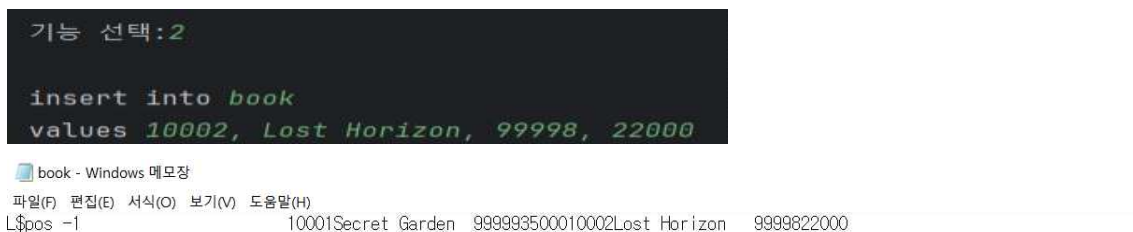


두 테스트를 통해 올바르게 데이터베이스 파일이 생성되고, 메타데이터들이 DB에 저장되는 것을 확인 할 수 있다.

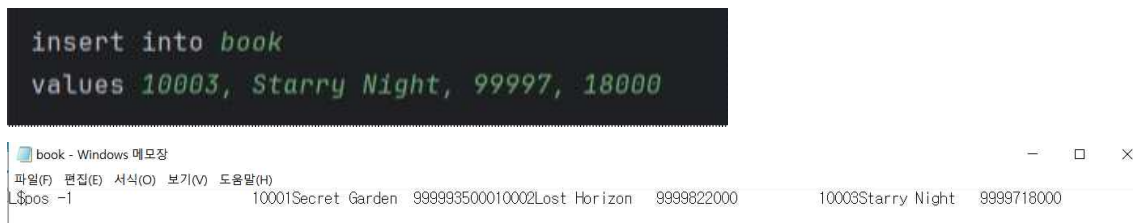
2. 삽입

1) 헤더의 포인터가 null(L\$pos -1) 인 경우

파일의 마지막에 새로운 레코드를 삽입해야 한다. 다음 두 테스트를 통해 올바르게 블록 I/O를 통한 삽입이 수행되고 있음을 알 수 있다.



현재 블록에는 60bytes가 사용중 이므로 위의 레코드는 블록에 삽입 될 수 있다. ID가 10001인 레코드 바로 다음에 ID가 10002인 레코드가 올바르게 저장되었다.



블록의 크기가 100bytes 이고 각 레코드의 크기가 30bytes이므로 첫번째 블록에는 더 이상 레코드를 삽입 할 수 없다. 따라서 새로운 두번째 블록에 레코드를 삽입해야 하는 데, 실행 결과를 보면 22000과 10003 사이에 빈공간(10bytes)가 있는 것을 통해 올바르게 수행되었음을 확인 할 수 있다.

2) 헤더의 포인터가 삭제된 공간을 가르키는 경우

3) 검색한 테이블이 존재 하지 않는 경우

```
insert into order
values (10000,10001,1,35000)

order 테이블은 생성되어 있지 않습니다.
```

3.삭제

1)헤더와 동일한 블록에 있는 레코드 삭제

- 삭제 전 파일

book - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

L\$pos -1	10001	Secret Garden	999993500010002	Lost Horizon	9999822000	1
-----------	-------	---------------	-----------------	--------------	------------	---

-삭제 명령어 수행

```
기능 선택:3

delete from book
where book_id=10001
```

-삭제 후 파일

book - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

L\$pos 30	L\$pos -1	10002	Lost Horizon	9999822000	1
-----------	-----------	-------	--------------	------------	---

book_id가 10001인 레코드가 올바르게 파일에서 제거되었고, 삭제된 레코드 위치에는 삭제 전 헤더의 값이, 새로운 헤더값으로는 삭제된 파일의 offset이 올바르게 저장되었다.

2) 헤더와 다른 블록에 있는 레코드 삭제

- 삭제 전 파일

book - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

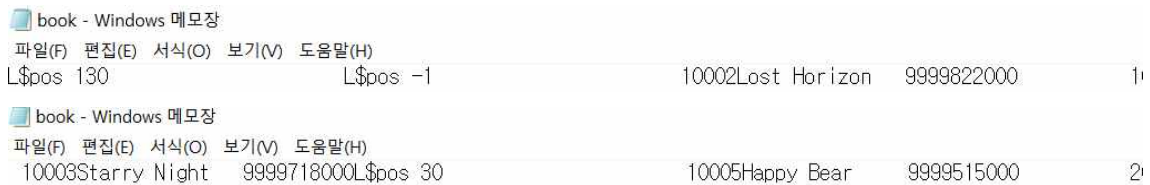
L\$pos 30	L\$pos -1	10002	Lost Horizon	9999822000	1
-----------	-----------	-------	--------------	------------	---

book - Windows 메모장

-삭제 명령어 수행

```
delete from book
where book_id=10004
```

-삭제 후 파일

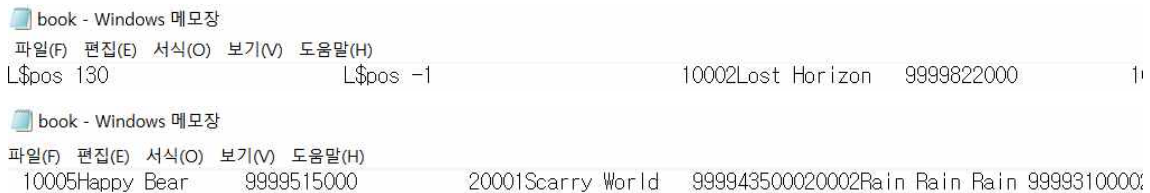


L\$pos	130	L\$pos	-1	10002	Lost Horizon	9999822000	1
L\$pos	30			10003	Starry Night	9999718000	
				10005	Happy Bear	9999515000	2

book_id가 10004인 레코드가 파일에서 올바르게 삭제 되었다. 삭제된 위치에는 이전의 헤더값인 L\$pos 30이 올바르게 저장되었고, 헤더에는 삭제된 레코드의 offset인 130이 올바르게 저장되었다.

3) Primary_Key 말고 다른 특성으로 삭제

- 삭제전 파일

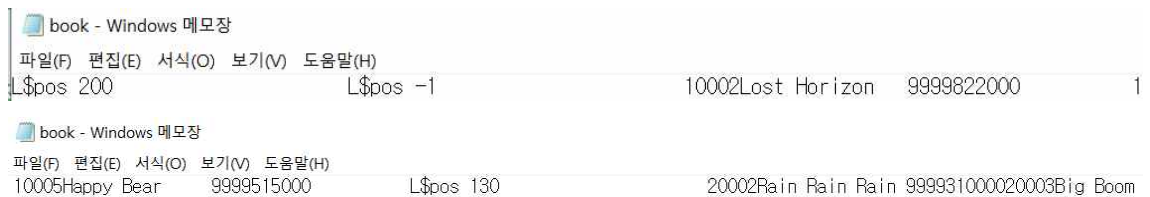


L\$pos	130	L\$pos	-1	10002	Lost Horizon	9999822000	1
				10005	Happy Bear	9999515000	
				20001	Scarry World	9999435000	2
				20002	Rain Rain Rain	9999310000	3

-삭제 명령어 수행

```
delete from book
where author_id=99994
```

-삭제 후 파일



L\$pos	200	L\$pos	-1	10002	Lost Horizon	9999822000	1
				10005	Happy Bear	9999515000	
				20002	Rain Rain Rain	9999310000	3
				20003	Big Boom	9999310000	3

author_id가 99994인 레코드가 올바르게 삭제되었고 삭제된 위치에는 삭제전 파일의 헤더값인 L\$pos 130이 저장되었다. 삭제된 레코드의 offset인 200이 올바르게 저장되었다.

4. 튜플 검색

1) 모든 레코드 검색

```
select *  
from book  
where True
```

book_id	title	author_id	price
10001	Secret Garden	99999	35000
10002	Lost Horizon	99998	22000
10003	Starry Night	99997	18000
10004	New Year	99996	45000
10005	Happy Bear	99995	15000
20001	Scarry World	99994	35000
20002	Rain Rain Rain	99993	10000
20003	Big Boom	99992	30000
20004	Boo king	99991	15000

1의 테이블과 동일하게 파일에 있는 모든 레코드를 올바르게 출력하였다.

2) 특정한 조건의 튜플 찾기

```
select *  
from book  
where book_id=20003
```

book_id	title	author_id	price
20003	Big Boom	99992	30000

book_id가 20003인 레코드를 올바르게 찾고 출력하였다.

3) 모든 컬럼 말고 특정 컬럼만 검색

```
select book_id, price  
from book  
where book_id=20001
```

book_id	price
20001	35000

book_id가 20001인 레코드의 book_id와 price만 올바르게 출력하였다.

4) 특정조건을 만족시키는 여러개의 레코드를 가지는 select 검색


```
select book_id, title, price
from book
where price=35000
```

book_id	title	price
10001	Secret Garden	35000
20001	Scarry World	35000

price가 35000인 레코드는 파일에 2개 뿐인데, 2개모두 올바르게 출력하였다.

5) 테이블에 없는 레코드 검색

기능 선택:4

```
select *
from book
where book_id=30000
```

book_id	title	author_id	price
---------	-------	-----------	-------

6) 레코드에서 삭제후 삭제한 레코드 검색

```
select *
from book
where book_id=10001
```

book_id	title	author_id	price
---------	-------	-----------	-------

```
select *
from book
where True
```

book_id	title	author_id	price
10002	Lost Horizon	99998	22000
10003	Starry Night	99997	18000
10005	Happy Bear	99995	15000
20002	Rain Rain Rain	99993	10000
20003	Big Boom	99992	30000
20004	Boo King	99991	15000

삭제 후 레코드를 검색 해 본 결과 삭제된 레코드들(id가 10001, 10004, 20001인 레코드)

은 정상적으로 검색되지 않았다.