



Chapter 04

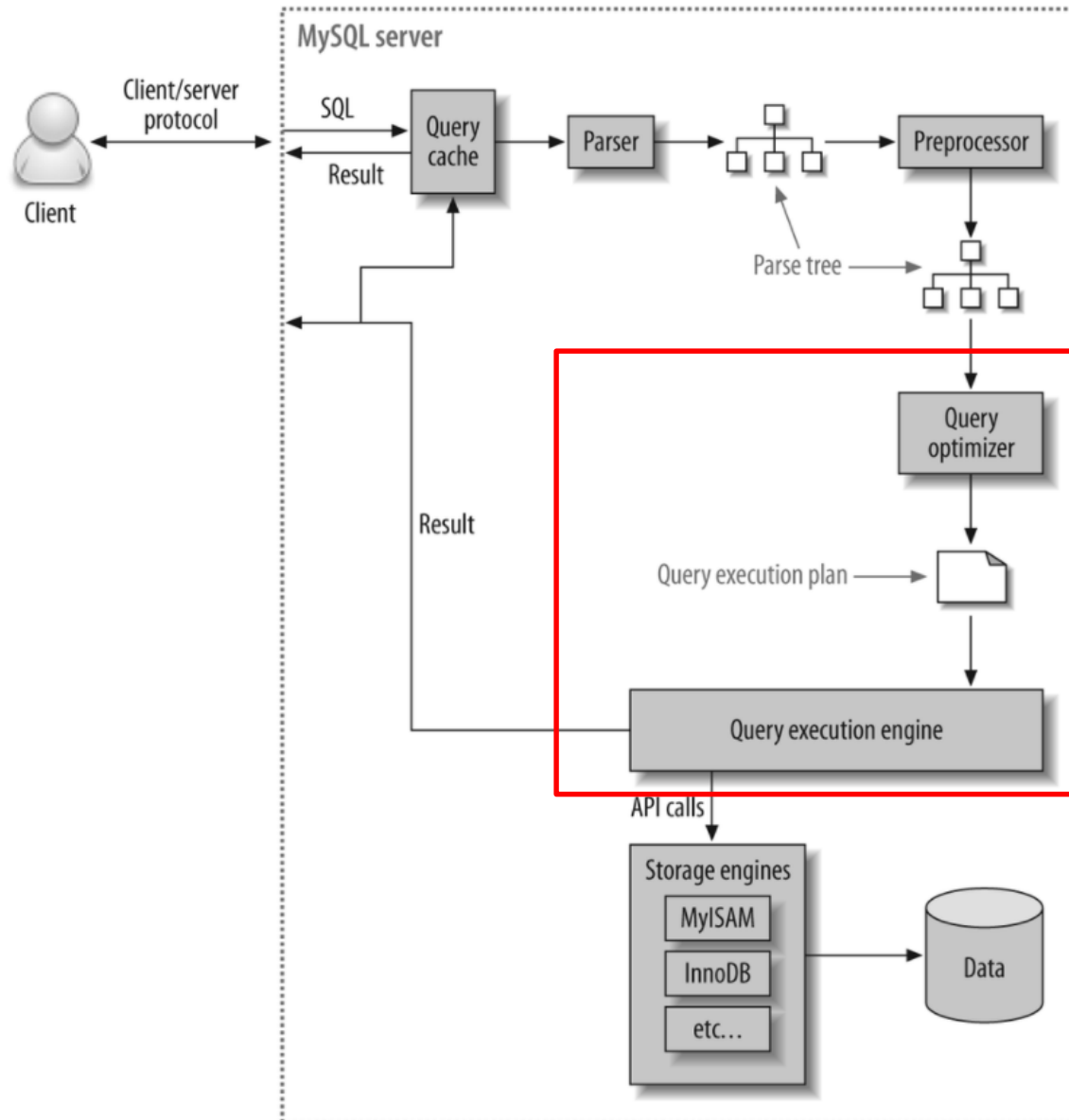
SQL 고급

02. 부속질의

1. 스칼라 부속질의 – SELECT 부속질의
2. 인라인 뷰 – FROM 부속질의
3. 중첩질의 – WHERE 부속질의



* 참고 > 쿼리 실행 계획 확인하기






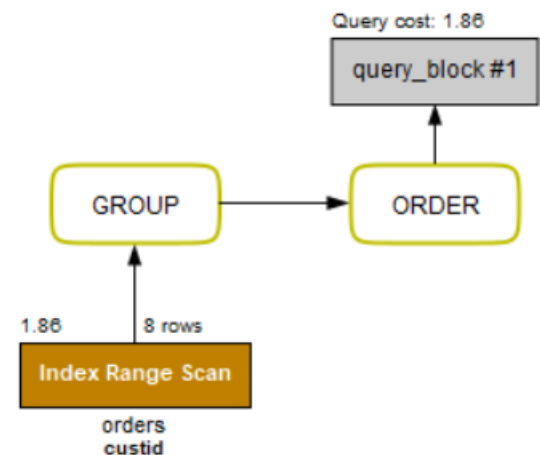
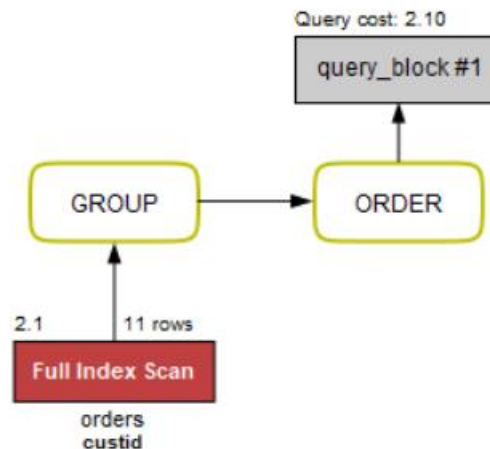
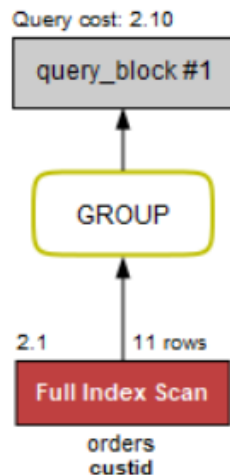
* 참고 > 쿼리 실행 계획 확인하기

- DBMS의 내부 핵심엔진인 쿼리 옵티마이저 : SQL을 처리하는 최저비용의 경로를 생성
- 쿼리를 수행할 때 생성한 최적의 처리경로를 실행
- 쿼리 앞에 explain 입력

```
select custid, count(*) from orders group by custid;
```

Result Grid Filter Rows: Export: Wrap Cell Content:												
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	orders	NULL	index	custid	custid	5	NULL	11	100.00	Using index

- Workbench에서 visual 실행계획 확인 (sql문 작성 -> control+alt+X,   )



부속질의

❖ 부속질의(subquery)란?

- 하나의 SQL 문 안에 다른 SQL 문이 중첩된(nested) 질의
- 다른 테이블에서 가져온 데이터로 현재 테이블에 있는 정보를 찾거나 가공할 때 사용
- 보통 데이터가 대량일 때 데이터를 모두 합쳐서 연산하는 조인보다 필요한 데이터만 찾아서 공급해주는 부속질의가 성능이 더 좋음
- 주질의(main query, 외부질의)와 부속질의(sub query, 내부질의)로 구성됨

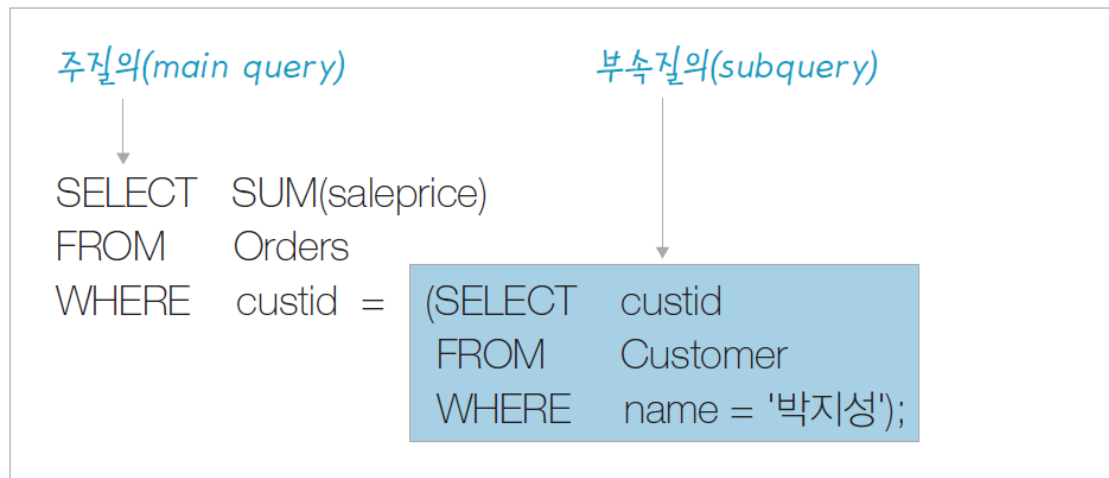


그림 4-2 부속질의

부속질의(서브쿼리, subquery)

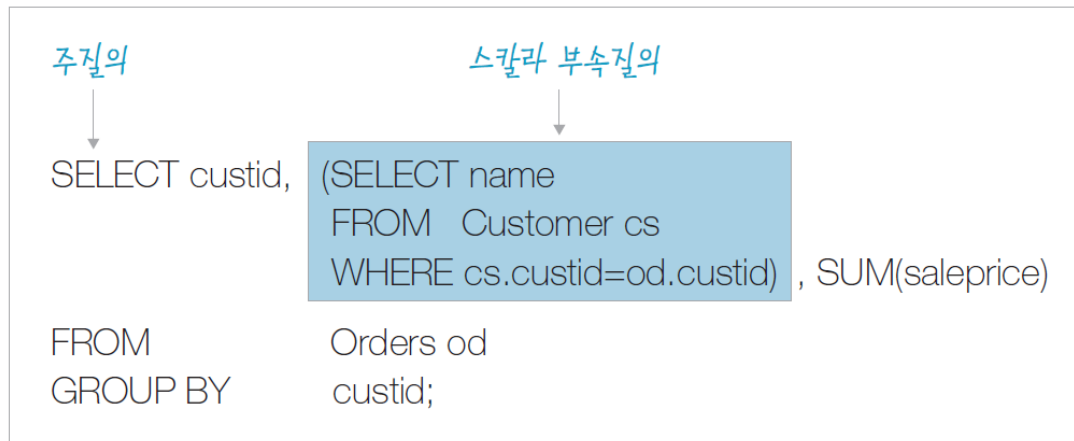
표 4-6 부속질의의 종류

명칭	위치	영문 및 동의어	설명
스칼라 부속질의	SELECT 절	scalar subquery	SELECT 절에서 함수처럼 사용되며 단일 값, 한 행만을 반환하기 때문에 스칼라 부속질의라고 함.
인라인 뷰	FROM 절	inline view, table subquery	FROM 절에서 결과를 뷰(view) 형태로 반환하기 때문에 인라인 뷰라고 함.
중첩 부속질의	WHERE 절	nested subquery, predicate subquery	WHERE 절에 술어와 같이 사용되며 결과를 한정시키기 위해 사용됨. 상관 혹은 비상관 형태.

1. 스칼라 부속질의 – SELECT 부속질의

❖ 스칼라 부속질의(scalar subquery)란?

- **SELECT 절에서 사용되는 부속질의로**, 부속질의의 결과 값을 **단일 행, 단일 열의 스칼라 값**으로 반환함
- 결과값이 다중 행이거나 다중 열이라면 DBMS 는 그 중 어떤 행, 어떤 열을 출력해야 하는지 알 수 없어 에러를 출력함, 결과가 없는 경우에는 NULL값을 출력함.
- 원칙적으로 스칼라 값이 들어갈 수 있는 모든 곳에 사용 가능하며, 일반적으로 SELECT 문과 UPDATE SET 절에 사용됨
- 주질의와 부속질의와의 관계는 상관/비상관 모두 가능함



Scalar Subquery

One Row, One Column

Zero Result → NULL Expression

One Row → Okay!!

Two or More → Error

그림 4-3 스칼라 부속질의

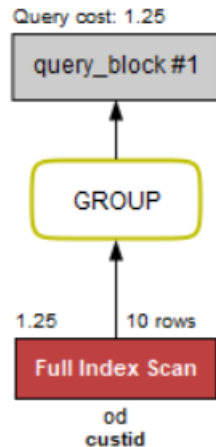
* 스칼라 값이란 벡터 값에 대응되는 말로 단일 값을 의미함

1. 스칼라 부속질의 - SELECT 부속질의

질의 4-12 마당서점의 고객별 판매액을 보이시오(고객이름과 고객별 판매액을 출력).

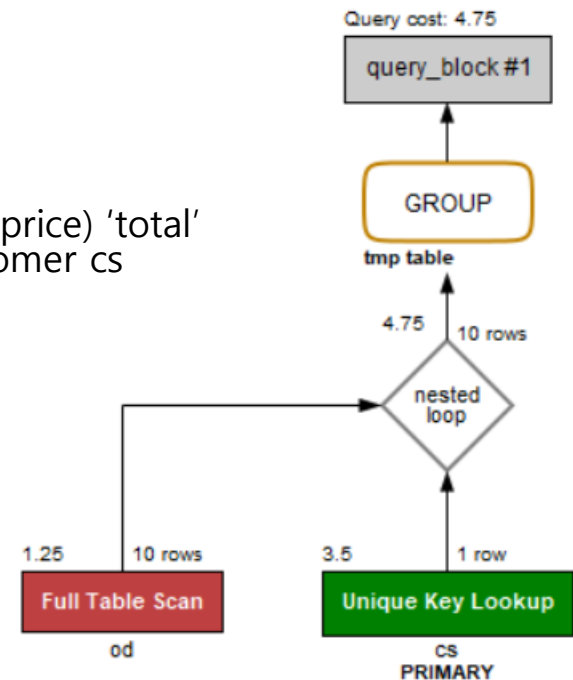
```
SELECT      ( SELECT  name
              FROM    Customer cs
              WHERE   cs.custid=od.custid ) 'name', SUM(saleprice) 'total'
FROM        Orders od
GROUP BY    od.custid;
```

name	total
박지성	39000
김연아	15000
장미란	31000
추신수	33000



(A blue arrow points from the 'GROUP BY' clause in the first query to this second query)

```
SELECT      name, SUM(saleprice) 'total'
FROM        Orders od , Customer cs
WHERE       cs.custid=od.custid
GROUP BY    od.custid;
```



1. 스칼라 부속질의 - SELECT 부속질의

```
SELECT    custid,  
          SUM(saleprice) 'total'  
FROM      Orders od  
GROUP BY custid ;
```



custid	total
1	39000
2	15000
3	31000
4	33000

```
SELECT    name  
FROM      Customer cs  
WHERE     cs.custid = od.custid
```

Custid 1의 이름은?



```
SELECT    (SELECT    name  
            FROM      Customer cs  
            WHERE     cs.custid = od.custid) name,  
          SUM(saleprice) 'total'  
FROM      Orders od  
GROUP BY od.custid ;
```



name	total
박지성	39000
김연아	15000
장미란	31000
추신수	33000

그림 4-4 마당서점의 고객별 판매액

1. 스칼라 부속질의 – SELECT 부속질의

다음 실습을 위해 새로운 필드 도서이름(bname)을 추가해보자.

```
ALTER TABLE Orders ADD bname VARCHAR(40);
```

새로운 필드에는 NULL 값이 저장되어 있다.

bookid가 1,2,3,...10인 경우 각각의 도서이름을 수정해보자.

질의 4-12 Orders 테이블에 각 주문에 맞는 도서이름을 입력하시오.

```
UPDATE   Orders
SET       bookname = ( SELECT bookname
                        FROM Book
                        WHERE Book.bookid=Orders.bookid );
```

orderid	custid	bookid	saleprice	orderdate	bname
1	1	1	6000	2014-07-01	축구의 역사
2	1	3	21000	2014-07-03	축구의 이해
3	2	5	8000	2014-07-03	피겨 교본
4	3	6	6000	2014-07-04	역도 단계별기술
5	4	7	20000	2014-07-05	야구의 추억
6	1	2	12000	2014-07-07	축구하는 여자
7	4	8	13000	2014-07-07	야구를 부탁해
8	3	10	12000	2014-07-08	Olympic Champions
9	2	10	7000	2014-07-09	Olympic Champions
10	3	8	13000	2014-07-10	야구를 부탁해

2. 인라인 뷰- FROM 부속질의

❖ 인라인 뷰(inline view)란?

- FROM 절에서 사용되는 부속질의
- 테이블 이름 대신 **인라인 뷰** 부속질의를 사용하면 보통의 **테이블과 같은 형태로** 사용할 수 있음
- 부속질의 결과 반환되는 데이터는 **다중 행, 다중 열**이어도 상관없음
- 다만 가상의 테이블인 뷰 형태로 제공되어 **상관 부속질의로** 사용될 수는 없음

질의 4-14 고객번호가 2 이하인 고객의 판매액을 보이시오(고객이름과 고객별 판매액 출력).

```
SELECT  cs.name, SUM(od.saleprice) 'total'
FROM    (SELECT custid, name
         FROM    Customer
         WHERE   custid <= 2) cs,
         Orders  od
WHERE   cs.custid=od.custid
GROUP BY cs.name;
```

name	total
박지성	39000
김연아	15000

2. 인라인 뷰- FROM 부속질의

실행과정 - cs 테이블을 계산해서 가상의 테이블(뷰)를 만들고 난다음 od 테이블과 조인을 한다.
나머지는 일반적인 SQL문의 처리 순서와 같다.

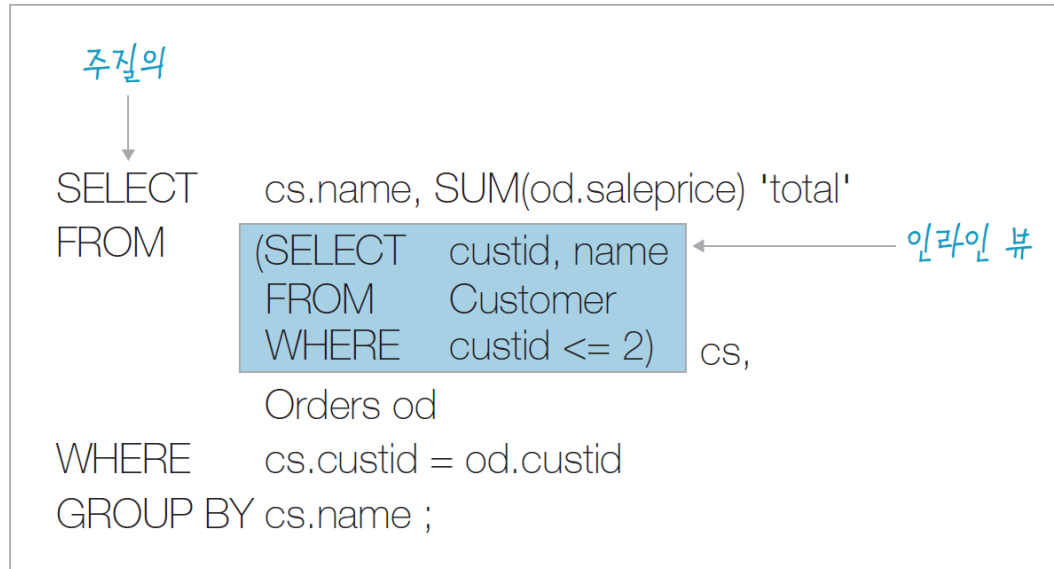


그림 4-5 인라인 뷰

아래 SQL과 비교해보자.

```
SELECT  cs.name, SUM(od.saleprice) 'total'
FROM    customer cs,
        Orders  od
WHERE    cs.custid=od.custid and od.custid <=2

GROUP BY cs.name;
```

3. 중첩질의 - WHERE 부속질의

- 중첩질의(nested subquery) : WHERE 절에서 사용되는 부속질의
- WHERE 절은 보통 데이터를 선택하는 조건 혹은 술어(predicate)와 같이 사용됨
→ 중첩질을 술어 부속질의(predicate subquery)라고도 함

표 4-7 중첩질의 연산자의 종류

술어	연산자	반환 행	반환 열	상관
비교	=, >, <, >=, <=, < >	단일	단일	가능
집합	IN, NOT IN	다중	다중	가능
한정(quantified)	ALL, SOME(ANY)	다중	단일	가능
존재	EXISTS, NOT EXISTS	다중	다중	필수

3. 중첩질의 - WHERE 부속질의

❖ 비교 연산자

부속질의가 반드시 **단일 행, 단일 열을 반환**해야 하며, 아닐 경우 질의를 처리할 수 없음

- ❖ 처리과정 : 주질의의 대상 열 값과 부속질의의 결과 값을 비교 연산자에 적용하여 참이면 주질의의 해당열을 출력한다.

질의 4-15 평균 주문금액 이하의 주문에 대해서 주문번호와 금액을 보이시오.

```
SELECT orderid, saleprice
FROM    Orders
WHERE   saleprice <= (SELECT AVG(saleprice)
                      FROM Orders);
```

orderid	saleprice
1	6000
3	8000
4	6000
9	7000

질의 4-16 각 고객의 평균 주문금액보다 큰 금액의 주문 내역에 대해서 주문번호, 고객번호, 금액을 보이시오.

```
SELECT orderid, custid, saleprice
FROM    Orders od
WHERE   saleprice > (SELECT AVG(saleprice)
                      FROM Orders so
                      WHERE od.custid=so.custid);
```

orderid	custid	saleprice
2	1	21000
3	2	8000
5	4	20000
8	3	12000
10	3	13000

3. 중첩질의 - WHERE 부속질의

❖ IN, NOT IN

- IN 연산자는 주질의 속성 값이 부속질의에서 제공한 결과 집합에 있는지 확인하는 역할을 함
- IN 연산자는 부속질의의 결과 **다중 행을 가질 수 있음**
- 주질의는 WHERE 절에 사용되는 속성 값을 부속질의의 결과 집합과 비교해 하나라도 있으면 참이 됨
- NOT IN은 이와 반대로 값이 존재하지 않으면 참이 됨

질의 4-17 대한민국에 거주하는 고객에게 판매한 도서의 총판매액을 구하시오.

```
SELECT SUM(saleprice) 'total'
FROM Orders
WHERE custid IN (SELECT custid
                  FROM Customer
                  WHERE address LIKE '%대한민국%');
```

total
46000

* 다중열 반환하는 경우

```
SELECT 컬럼, 컬럼 ...
FROM 테이블
WHERE (컬럼1, 컬럼2, ...) IN (SELECT 컬럼1, 컬럼2, ...
                              FROM 테이블);
```

```
SELECT *
FROM EMP
WHERE (EMPNO, JOB) IN (SELECT EMPNO, JOB
                       FROM EMP
                       WHERE DEPTNO = 20);
```

3. 중첩질의 – WHERE 부속질의

❖ ALL, SOME(ANY)

- ALL은 모두, SOME(ANY)은 어떠한(최소한 하나라도)이라는 의미
- 구문 구조

```
scalar_expression { 비교연산자 ( =, <>, !=, >, >=, !=, <, <=, != ) }  
                  { ALL | SOME | ANY } (부속질의)
```

질의 4-18 3번 고객이 주문한 도서의 최고 금액보다 더 비싼 도서를 구입한 주문의 주문번호와 금액을 보이시오.

```
SELECT orderid, saleprice  
FROM    Orders  
WHERE   saleprice > ALL (SELECT saleprice  
                        FROM    Orders  
                        WHERE custid='3');
```

orderid	saleprice
2	21000
5	20000

3. 중첩질의 - WHERE 부속질의

❖ EXISTS, NOT EXISTS

- 데이터의 존재 유무를 확인하는 연산자
- 주질의에서 부속질의로 제공된 속성의 값을 가지고 부속질의에 조건을 만족하여 값이 존재하면 참이 되고, 주질의는 해당 행의 데이터를 출력함
- NOT EXISTS의 경우 이와 반대로 동작함
- 구문 구조

WHERE [NOT] EXISTS (부속질의)

질의 4-19 EXISTS 연산자로 대한민국에 거주하는 고객에게 판매한 도서의 총 판매액을 구하시오.

```
SELECT SUM(saleprice) 'total'
FROM Orders od
WHERE EXISTS (SELECT *
              FROM Customer cs
              WHERE address LIKE '%대한민국%' AND cs.custid=od.custid);
```

total
46000

답인지 설명하시오.

```
(1) SELECT      custid, (SELECT      address
                        FROM        Customer cs
                        WHERE       cs.custid = od.custid) "address",
                        SUM(saleprice) "total"
```

```
FROM      Orders od
GROUP BY  od.custid;
```

```
(2) SELECT      cs.name, s
FROM            (SELECT  custid, AVG(saleprice) s
FROM            Orders
GROUP BY custid) od, Customer cs
WHERE           cs.custid = od.custid;
```

```
(3) SELECT      SUM(saleprice) "total"
FROM            Orders od
WHERE EXISTS    (SELECT      *
FROM            Customer cs
WHERE           custid <= 3 AND cs.custid = od.custid);
```

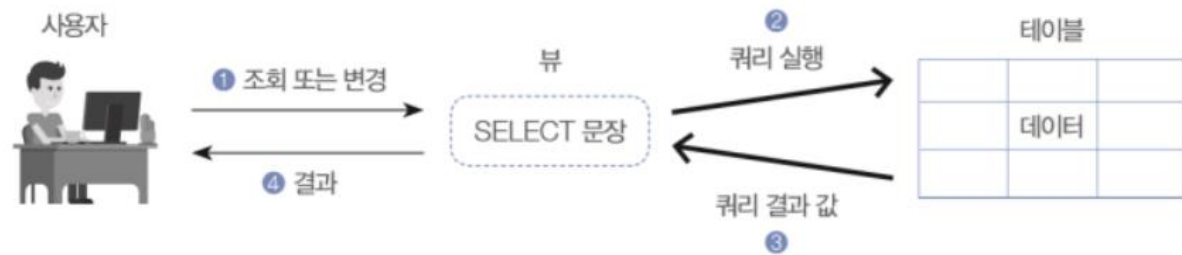
03. 뷰

1. 뷰의 생성
2. 뷰의 수정
3. 뷰의 삭제



뷰

- 뷰(view) : 하나 이상의 테이블을 합하여 만든 가상의 테이블
테이블처럼 행과 열을 가지고 있지만, 실제로 데이터를 저장하고 있지는 않다.



뷰의 작동 방식

이미지 : 이것이 MySQL이다.

■ 뷰의 장점

- **편리성 및 재사용성** : 자주 사용되는 복잡한 질의를 뷰로 미리 정의해 놓을 수 있음
→ 복잡한 질의를 간단히 작성, 복잡한 쿼리를 단순화해서 사용할 수 있음
- **보안성** : 사용자별로 필요한 데이터만 선별하여 보여줄 수 있고, 중요한 질의의 경우 질의 내용을 암호화할 수 있음, 특정 사용자에게 테이블 전체가 아닌 필요한 필드만 보여줄 수 있음
→ 개인정보(주민번호)나 급여, 건강 같은 민감한 정보를 제외한 테이블을 만들어 사용
- **독립성** : 미리 정의된 뷰를 일반 테이블처럼 사용할 수 있기 때문에 편리하고, 사용자가 필요한 정보만 요구에 맞게 가공하여 뷰로 만들어 쓸 수 있음
→ 원본 테이블의 구조가 변해도 응용에 영향을 주지 않도록 하는 논리적 독립성 제공

뷰

■ 뷰의 특징

- 원본 데이터 값에 따라 같이 변함
- 독립적인 인덱스 생성이 어려움, 뷰 자신만의 인덱스를 가질 수 없음
- 삽입, 삭제, 갱신 연산에 많은 제약이 따름
- 한 번 정의된 뷰는 변경이 불가능함

■ 뷰의 기본문법

```
CREATE VIEW 뷰이름 [(열이름 [ ,...n ])]
```

```
AS
```

```
SELECT {[필드명]}
```

```
FROM 테이블이름...
```

뷰

뷰 Vorders

orderid	custid	name	bookid	bookname	saleprice	orderdate
1	1	박지성	1	축구의 역사	6000	2014-07-01
2	1	박지성	3	축구의 이해	21000	2014-07-03
3	2	김연아	5	피겨 교본	8000	2014-07-03
4	3	장미란	6	역도 단계별기술	6000	2014-07-04

뷰 생성문

```
CREATE VIEW Vorders
AS SELECT orderid, O.custid, name, O.bookid, bookname, saleprice, orderdate
FROM Customer C, Orders O, Book B
WHERE C.custid=O.custid and B.bookid=O.bookid;
```

베이스 릴레이션 Customer, Orders, Book

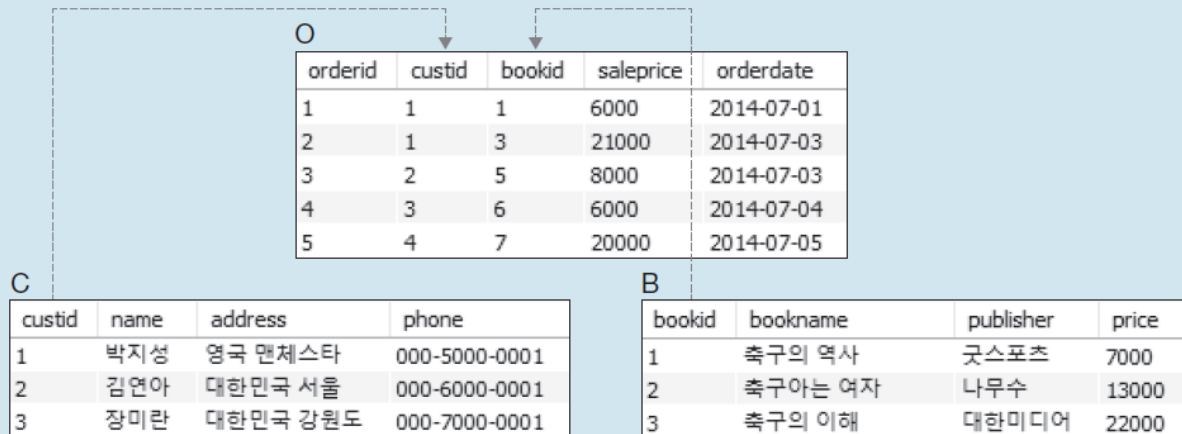


그림 4-6 뷰

1. 뷰의 생성

- Book 테이블에서 '축구'라는 문구가 포함된 자료만 보여주는 뷰

```
SELECT    *  
FROM      Book  
WHERE     bookname LIKE '%축구%';
```

- 위 SELECT 문을 이용해 작성한 뷰 정의문

```
CREATE VIEW vw_Book AS  
SELECT    *  
FROM      Book  
WHERE     bookname LIKE '%축구%';
```

1. 뷰의 생성 - 단일 테이블의 필요한 필드만 조회하는 뷰

질의 4-20 주소에 '대한민국'을 포함하는 고객들로 구성된 뷰를 만들고 조회하시오. 뷰의 이름은 vw_Customer로 설정하시오.

```
CREATE VIEW    vw_Customer    AS
  SELECT      *
  FROM        Customer
  WHERE       address LIKE '%대한민국%';
```

<결과 확인>

```
SELECT      *
FROM        vw_Customer;
```

custid	name	address	phone
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-7000-0001
5	박세리	대한민국 대전	NULL

1. 뷰의 생성 - 여러 테이블의 필요한 필드를 조회하는 뷰

질의 4-21 Orders 테이블에 고객이름과 도서이름을 바로 확인할 수 있는 뷰를 생성한 후, '김연아' 고객이 구입한 도서의 주문번호, 도서이름, 주문액을 보이시오.

```
CREATE VIEW vw_Orders (orderid, custid, name, bookid, bookname, saleprice, orderdate)
AS SELECT      od.orderid, od.custid, cs.name,
               od.bookid, bk.bookname, od.saleprice, od.orderdate
FROM          Orders od, Customer cs, Book bk
WHERE         od.custid =cs.custid AND od.bookid =bk.bookid;
```

<결과 확인>

```
SELECT      orderid, bookname, saleprice
FROM        vw_Orders
WHERE       name='김연아';
```

orderid	bookname	saleprice
3	피겨 교본	8000
9	Olympic Champions	7000

2. 뷰 대체 / 뷰 수정

- ❖ 기존에 생성했던 뷰를 다시 새로운 뷰로 대체한다는 뜻임
- 뷰는 한번 생성하면 변경이 불가능하기 때문에 새로운 뷰로 대체하는 것으로 뷰에 설정한 필드를 대체한다.

```
CREATE OR REPLACE VIEW 뷰이름 [(열이름 [ ,...n ])]  
AS SELECT 문
```

질의 4-22 [질의 4-20]에서 생성한 뷰 vw_Customer는 주소가 대한민국인 고객을 보여준다. 이 뷰를 영국을 주소로 가진 고객으로 변경하시오. phone 속성은 필요 없으므로 포함시키지 마시오.

```
CREATE OR REPLACE VIEW vw_Customer (custid, name, address)  
AS SELECT custid, name, address  
FROM Customer  
WHERE address LIKE '%영국%';
```

<결과 확인>

```
SELECT *  
FROM vw_Customer;
```

custid	name	address
1	박지성	영국 맨체스타

2. 뷰 대체 / 뷰 수정

- 만약 아래와 같이 뷰를 대체하게 되면, [field_name_2]라는 이름을 가진 필드가 [new_field_name]이라는 이름으로 대체됩니다.

```
CREATE OR REPLACE VIEW [view_name] AS  
SELECT [field_name_1], [field_name_2] AS [new_field_name]  
FROM [table_name];
```

- 뷰 수정 - ALTER

```
ALTER VIEW 뷰이름 AS  
SELECT {[필드명]}  
FROM 테이블이름
```

3. 뷰의 삭제

❖ 기본 문법

```
DROP VIEW 뷰이름 [ ,...n ];
```

질의 4-23 앞서 생성한 뷰 vw_Customer를 삭제하시오.

```
DROP VIEW vw_Customer;
```

<결과 확인>

```
SELECT *  
FROM vw_Customer;
```

Message	Duration / Fetch
Error Code: 1146. Table 'madang.vw_customer' doesn't exist	0.000 sec

* 뷰의 내용 확인

❖ 기본 문법

DESCRIBE 뷰이름;

SHOW CREATE VIEW 뷰이름;

```
156 • create or replace view v_customer  
157   as select * from customer;  
158 • describe v_customer;
```

Result Grid | Filter Rows: | Export:

	Field	Type	Null	Key	Default	Extra
▶	custid	int	NO		NULL	
	name	varchar(40)	YES		NULL	
	address	varchar(50)	YES		NULL	
	phone	varchar(20)	YES		NULL	

```
159 • show create view v_customer;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	View	Create View	character_set_client	collation_connection
▶	v_customer	CREATE ALGORITHM=UNDEFINED DEFINER=`r...` utf8mb4	utf8mb4	utf8mb4_0900_ai_ci

Form Editor | Navigate: 1 / 1

View:

v_customer

```
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `v_customer` AS select  
`customer`.`custid` AS `custid`,`customer`.`name` AS `name`,`customer`.`address` AS `address`,`customer`.`phone` AS  
`phone` from `customer`
```

Result 16 x

! Read Only

Result
Grid

Form
Editor

* 뷰를 이용해 테이블의 내용 수정

❖ 수정

update v_customer set phone = '000-0000' where custid=4;

	custid	name	address	phone
▶	1	박지성	영국 맨체스터	000-5000-0001
	2	김연아	대한민국 서울	000-6000-0001
	3	장미란	대한민국 강원도	000-7000-0001
	4	추신수	미국 클리블랜드	000-0000
	5	박세리	대한민국 대전	NULL

❖ 집계함수의 결과를 컬럼으로 가진 경우 등은 수정 불가

❖ 수정 가능 여부 확인은 ?

❖ SELECT * FROM information_schema.views

where table_schema = 'madang' and table_name = 'v_customer';

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	VIEW_DEFINITION	CHECK_OPTION	IS_UPDATABLE	DEFINER
def	madang	v_customer	select 'madang`.`customer`.`custid` AS `cust...	NONE	YES	root@localhost

* 뷰를 이용한 테이블 내용 추가

❖ 입력

- ❖ 뷰에 일부 필드만 있는 경우 해당 필드에 not null 제약조건이 있다면 뷰를 통해 입력하는 것이 불가능
- ❖ 뷰를 만들 때 지정한 조건에 벗어난 값을 가진 데이터를 입력 못하게 하려면 ? (나이가 20 이상인 사람만 가지고 있는 뷰에 나이가 10 인 사람의 자료는 입력불가)

-create or replace 뷰이름 as select * from customer where age >= 20
with check option


- alter view 뷰이름 as select * from user where age >= 20
with check option

* 복합뷰(2개이상의 테이블을 이용해 만든 뷰)를 통한 데이터 입력이나 수정은 불가능함

* 뷰가 설정되어 있는 테이블 삭제

- CREATE VIEW v_usertbl as select ...;
- CHECK TABLE v_usertbl;

Result Grid


Filter Rows:





Export:


	Table	Op	Msg_type	Msg_text
▶	madang.v_user	check	status	OK

- DROP TABLE if exists usertbl ;

Result Grid		Filter Rows: <input type="text"/>	Export: 	Wrap Cell Content: 
	Table	Op	Msg_type	Msg_text
▶	madang.v_user	check	Error	View 'madang.v_user' references invalid table(s...
	madang.v_user	check	error	Corrupt

06. 다음에 해당하는 뷰를 작성하시오. 데이터베이스는 마당서점 데이터베이스를 이용한다.

- (1) 판매가격이 20,000원 이상인 도서의 도서번호, 도서이름, 고객이름, 출판사, 판매가격을 보여주는 highorders 뷰를 생성하시오.
- (2) 생성한 뷰를 이용하여 판매된 도서의 이름과 고객의 이름을 출력하는 SQL 문을 작성하시오.
- (3) highorders 뷰를 변경하고자 한다. 판매가격 속성을 삭제하는 명령을 수행하시오. 삭제 후 (2)번 SQL 문을 다시 수행하시오.

04. 인덱스

1. 데이터베이스의 물리적 저장
2. 인덱스와 B-tree
3. MySQL 인덱스
4. 인덱스의 생성
5. 인덱스의 재구성과 삭제



1. 데이터베이스의 물리적 저장

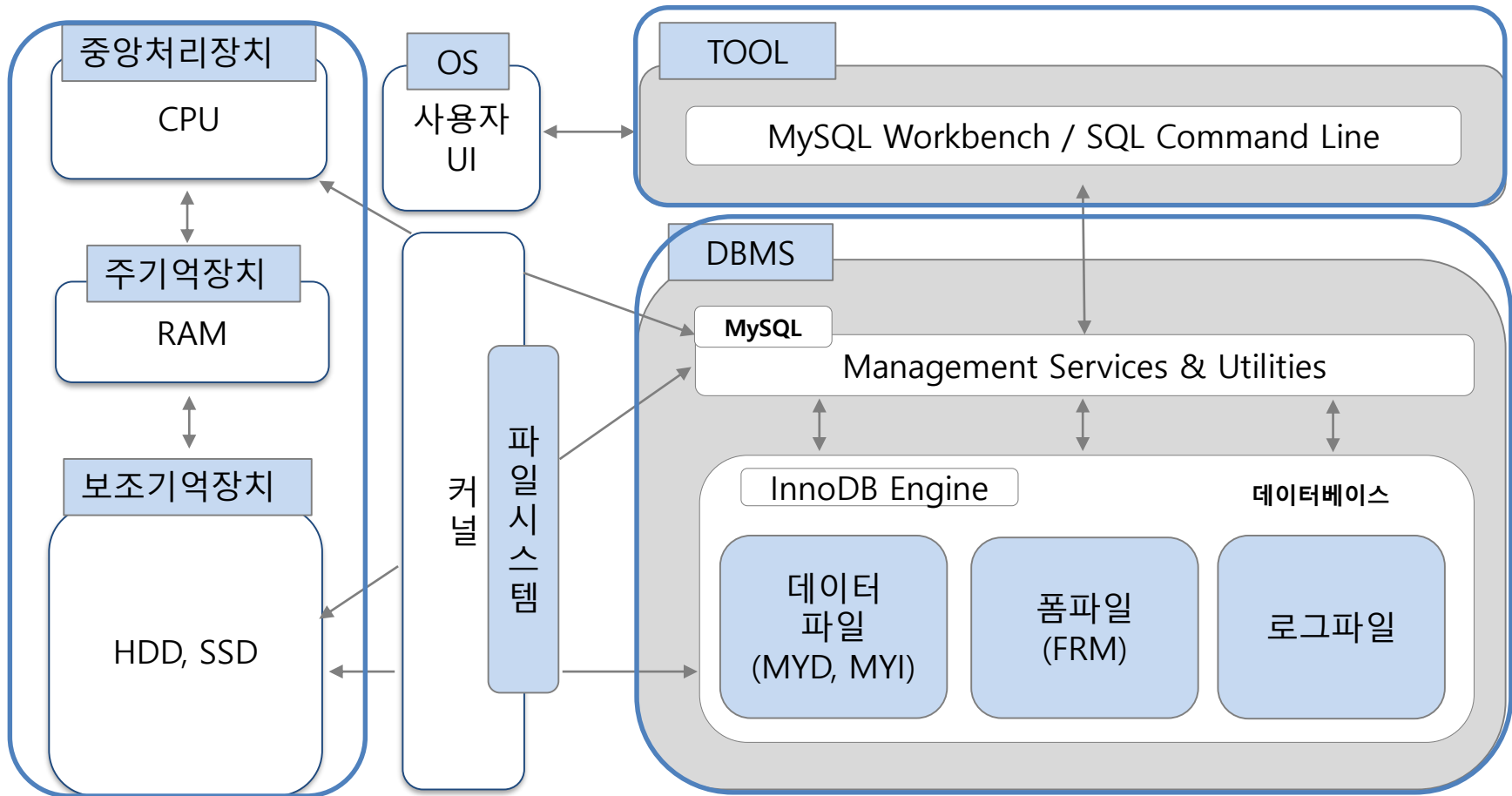
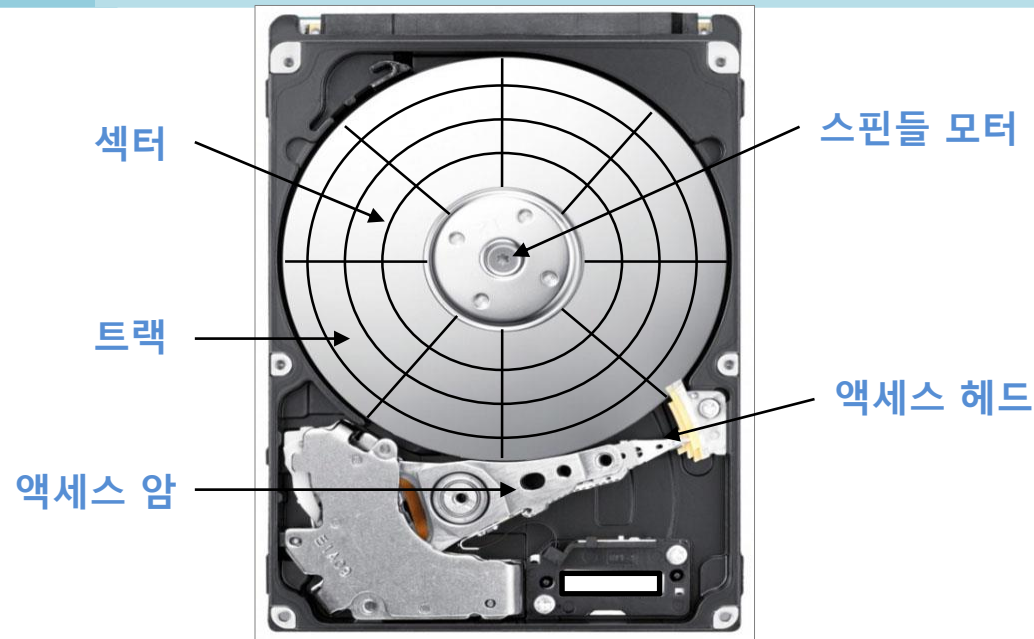


그림 4-7 DBMS와 데이터 파일

1. 데이터베이스의 물리적 저장

■ 실제 데이터가 저장되는 곳은 보조기억장치

- 하드디스크, SSD, USB 메모리 등



■ 가장 많이 사용되는 장치는 하드디스크

- 하드디스크는 원형의 플레이트(plate)로 구성되어 있고, 이 플레이트는 논리적으로 트랙으로 나뉘며 트랙은 다시 몇 개의 섹터로 나뉨
- 원형의 플레이트는 초당 빠른 속도로 회전하고, 회전하는 플레이트를 하드디스크의 액세스 암 (arm)과 헤더(header)가 접근하여 원하는 섹터에서 데이터를 가져옴
- 하드디스크에 저장된 데이터를 읽어 오는 데 걸리는 시간은 모터(motor)에 의해서 분당 회전 하는 속도(RPM, Revolutions Per Minute), 데이터를 읽을 때 액세스 암이 이동하는 시간 (latency time), 주기억장치로 읽어오는 시간(transfer time)에 영향을 받음

그림 4-8 하드디스크의 구조

1. 데이터베이스의 물리적 저장

❖ 액세스 시간(access time)

- 데이터의 저장 및 읽기에 많은 영향을 끼침

액세스 시간 = 탐색시간(seek time, 액세스 헤드를 트랙에 이동시키는 시간)

+ 회전지연시간(rotational latency time, 섹터가 액세스 헤드에 접근하는 시간)

+ 데이터 전송시간(data transfer time, 데이터를 주기억장치로 읽어오는 시간)

* 디스크는 주기억장치보다 1000배 이상 느리기 때문에 DBMS가 하드디스크에 데이터를 저장하고 읽어올때는 근본적인 속도 문제가 발생할 수밖에 없음

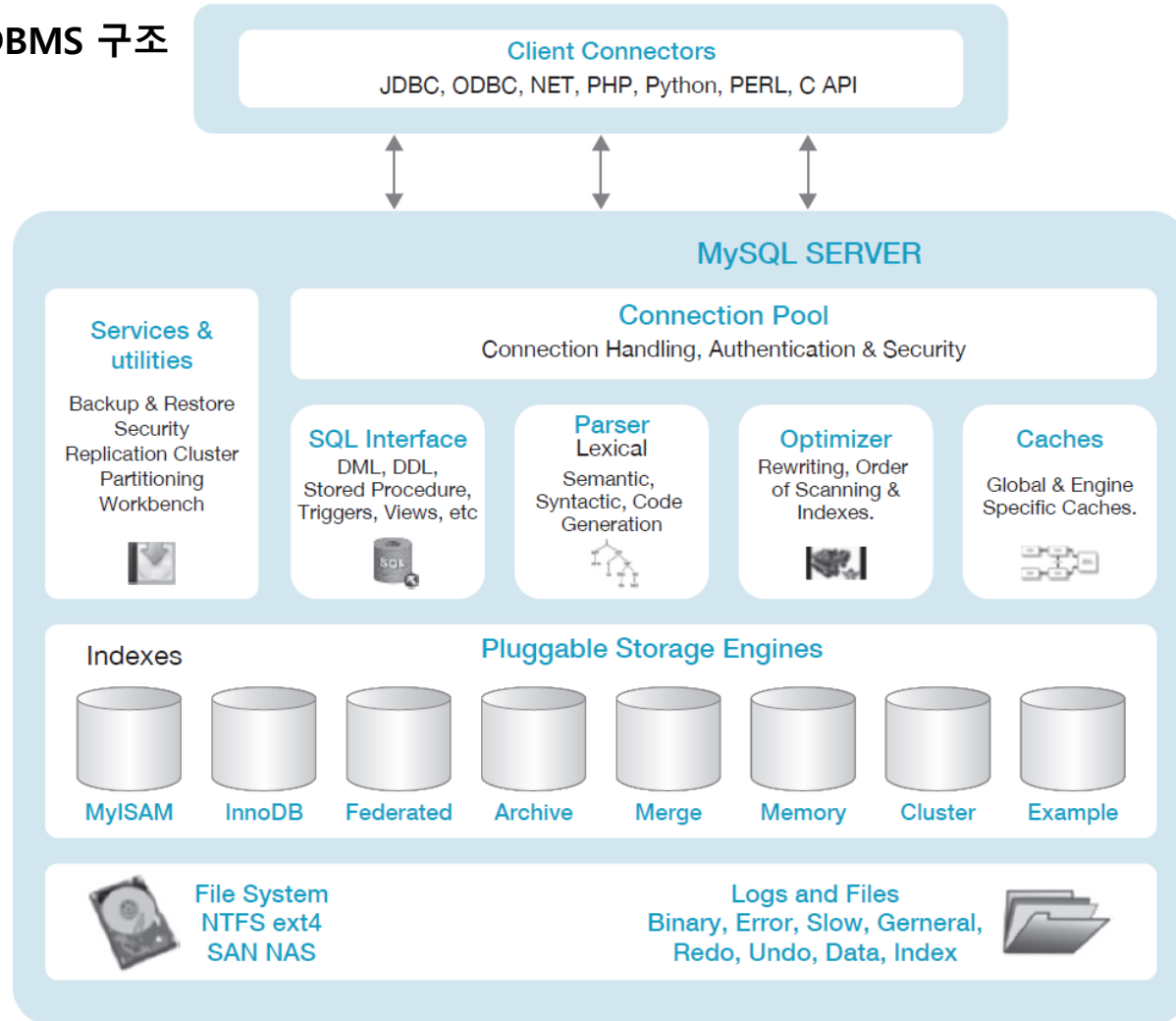
* 주기억장치에 DBMS가 사용하는 공간중 일부를 버퍼 풀(Buffer Pool Memory)로 만들어 사용하는 방법으로 속도 문제를 줄일 수 있음

- DB는 버퍼에 자주 사용하는 데이터를 저장해두고 Least Recently-Used 알고리즘을 이용하여 사용빈도가 높은 데이터를 저장하고 관리

- 데이터 검색시 버퍼 풀에 저장된 데이터를 우선 읽어 들인 후 작업을 진행함.

1. 데이터베이스의 물리적 저장

그림 4-9 MySQL의 DBMS 구조



* 데이터베이스 파일은 고유한 파일구조를 가지고
동시에 수많은 사용자가 사용해야 하므로 특별한 처리방법을 통해 관리된다.

1. 데이터베이스의 물리적 저장

표 4-8 MySQL InnoDB 엔진 데이터베이스의 파일

파일	설명
데이터 파일 (ibdata)	<ul style="list-style-type: none">• 사용자 데이터와 개체를 저장• 테이블과 인덱스로 구성• 확장자는 *.ibd
폼파일 (frm File)	<ul style="list-style-type: none">• 테이블에 대한 각종 정보와 테이블을 구성하는 필드, 데이터 타입에 대한 정보 저장• 데이터베이스 구조 등의 변경사항이 있을 때 자동으로 업데이트됨

* `SHOW VARIABLES LIKE 'datadir';` // 데이터베이스가 저장된 위치 확인

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
	Variable_name	Value	
▶	datadir	C:\ProgramData\MySQL\MySQL Server 8.0\Data	

2. 인덱스와 B-tree

- 인덱스(index, 색인) : 책 뒤의 찾아보기, 도서관의 색인카드나 사전과 같이 데이터를 쉽고 빠르게 찾을 수 있도록 만든 자료구조, 튜플의 키값에 대한 물리적 위치를 기록해둔 자료구조

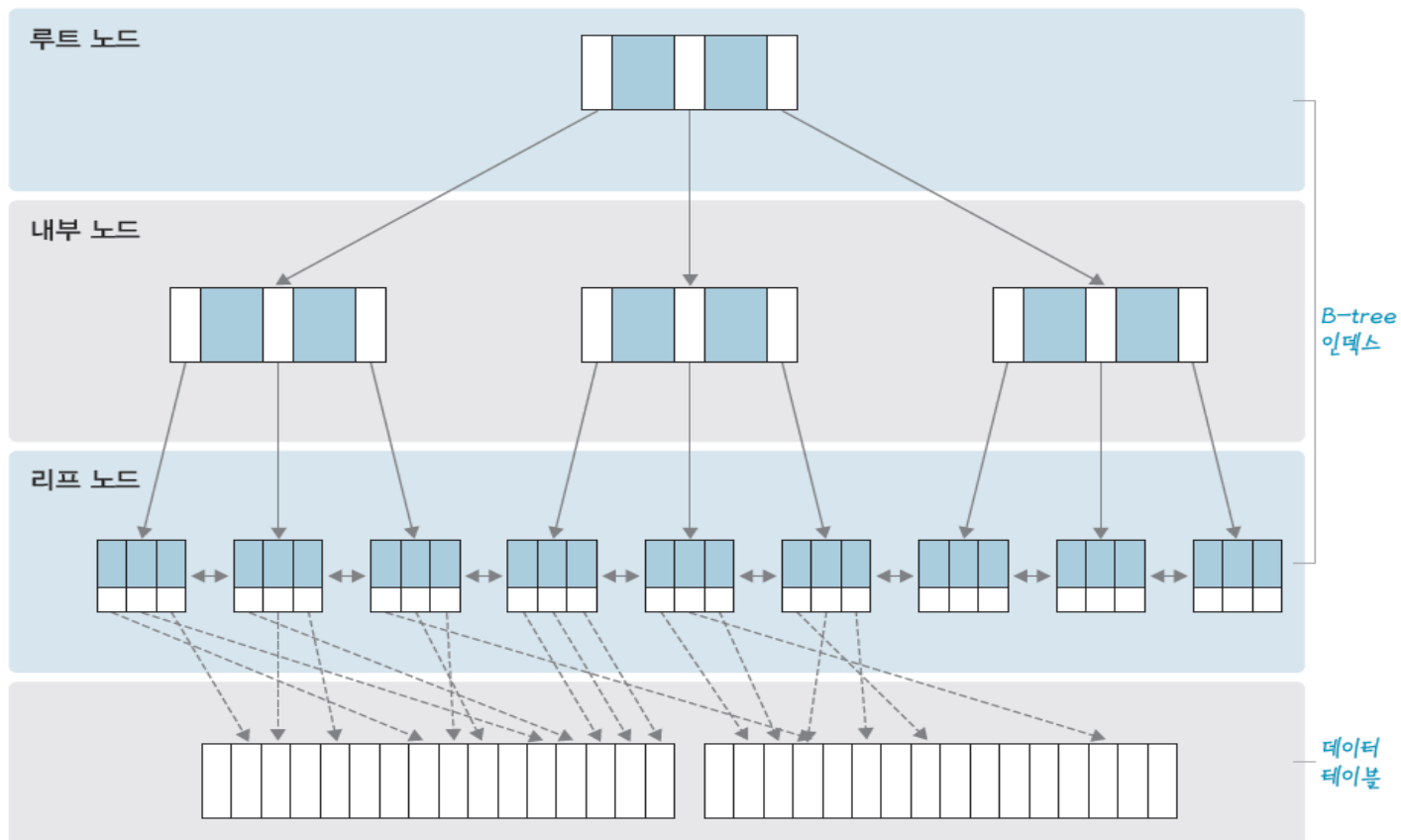


그림 4-10 B-tree(Balanced-tree)의 구조 - 리프노드가 모두 같은 레벨에 존재하는 균형트리

2. 인덱스와 B-tree

- 데이터 검색시간을 단축하기위한 자료구조로 Rudolf Bayer가 고안함.
- 각 노드는 키값과 포인터를 가짐
- 키값은 오름차순으로 저장, 키 값의 왼쪽 포인터는 키 값보다 작은 값을 오른쪽 포인터는 키 값보다 큰 값을 가진 다음 노드를 가리킴
- 3을 찾는 과정

루트노드의 값 4와 3을 비교 => 왼쪽 포인터가 지시하는 노드로 이동

=> 2와 3을 비교 ==> 오른쪽 포인터가 지시하는 노드로 이동

=> 3과 3을 비교 ==> 같은 값을 발견하면 검색을 중지

- 새로운 노드를 삽입하는 과정

==> 루트노드에서 시작하여 값을

삽입될 위치를 찾아감

==> 저장할 공간이 없으면 새로운
분할하여 값을 이동시킨 후 삽입함

>>> 균형상태 유지

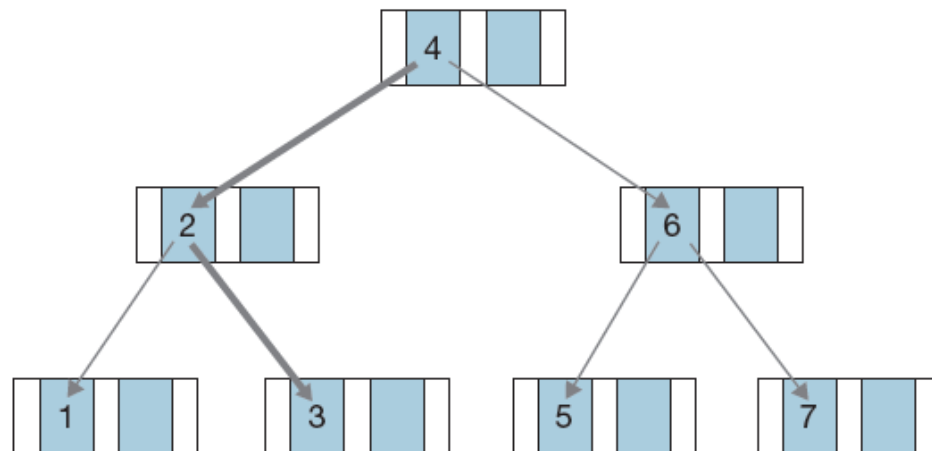


그림 4-11 B-tree에서 검색 예

2. 인덱스와 B-tree(Balanced-tree)

■ 인덱스의 특징

- 인덱스는 테이블에서 한 개 이상의 속성을 이용하여 생성함
- 빠른 검색과 함께 효율적인 레코드 접근이 가능함
- 순서대로 정렬된 속성과 데이터의 위치만 보유하므로 테이블보다 작은 공간을 차지함
- 저장된 값들은 테이블의 부분집합이 됨
- 일반적으로 B-tree 형태의 구조를 가짐
- 데이터의 수정, 삭제 등의 변경이 발생하면 인덱스의 재구성이 필요함

■ 인덱스의 단점

- 인덱스도 공간을 차지해서 데이터베이스 안에 추가적인 공간(대략 10%)이 필요함
- 처음에 인덱스를 만드는데 시간이 오래 걸릴 수 있음
- 데이터의 변경 작업이 자주 일어나면 오히려 성능이 더 나빠질 수도 있음

3. MySQL 인덱스

❖ MySQL 인덱스의 종류

표 4-9 MySQL 인덱스의 종류

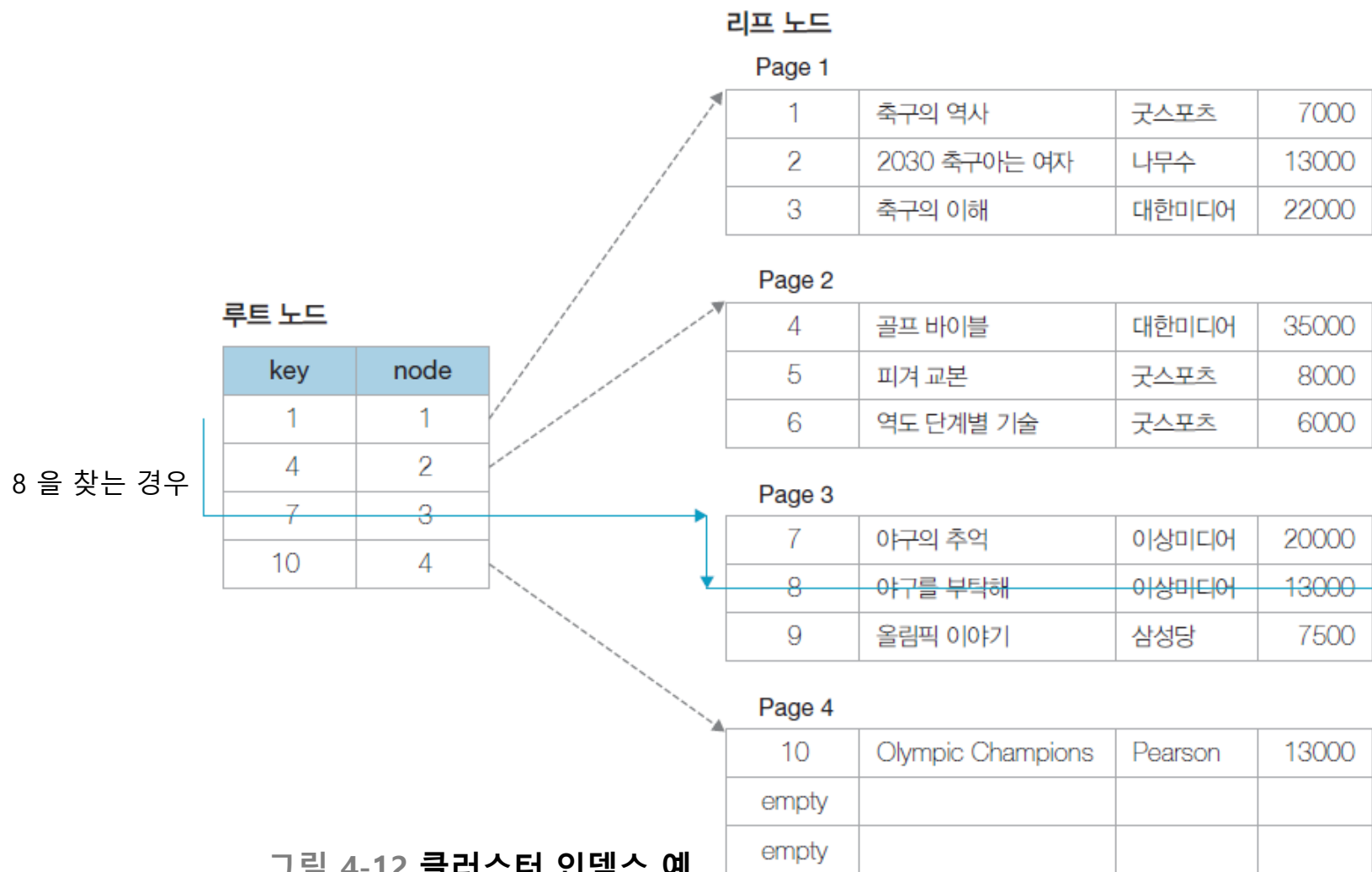
인덱스 명칭	설명 / 생성 예
클러스터 인덱스	<ul style="list-style-type: none">기본적인 인덱스로 테이블 생성 시 기본키를 지정하면 기본키에 대하여 클러스터 인덱스를 생성한다.기본키를 지정하지 않으면 먼저 나오는 UNIQUE 속성에 대하여 클러스터 인덱스를 생성한다.기본키나 UNIQUE 속성이 없는 테이블은 MySQL 이 자체 생성한 행번호 (Row ID)를 이용하여 클러스터 인덱스를 생성한다.
보조 인덱스	<ul style="list-style-type: none">클러스터 인덱스가 아닌 모든 인덱스는 보조 인덱스이며 보조 인덱스의 각 레코드는 보조 인덱스 속성과 기본키 속성 값을 갖고 있다.보조 인덱스를 검색하여 기본키 속성 값을 찾은 다음 클러스터 인덱스로 가서 해당 레코드를 찾는다.

* 쉽게 비교하면 클러스터 인덱스는 영어사전과 같은 방식이고 클러스터 인덱스로 지정한 열을 기준으로 자동 정렬된다.

* 보조 인덱스는 책의 뒤에 찾아보기가 있는 일반적인 책과 같고, 여러 개를 만들 수 있지만 자동 정렬되지는 않는다.

3. MySQL 인덱스

❖ 클러스터 인덱스



3. MySQL 인덱스

❖ MySQL 인덱스 B-tree

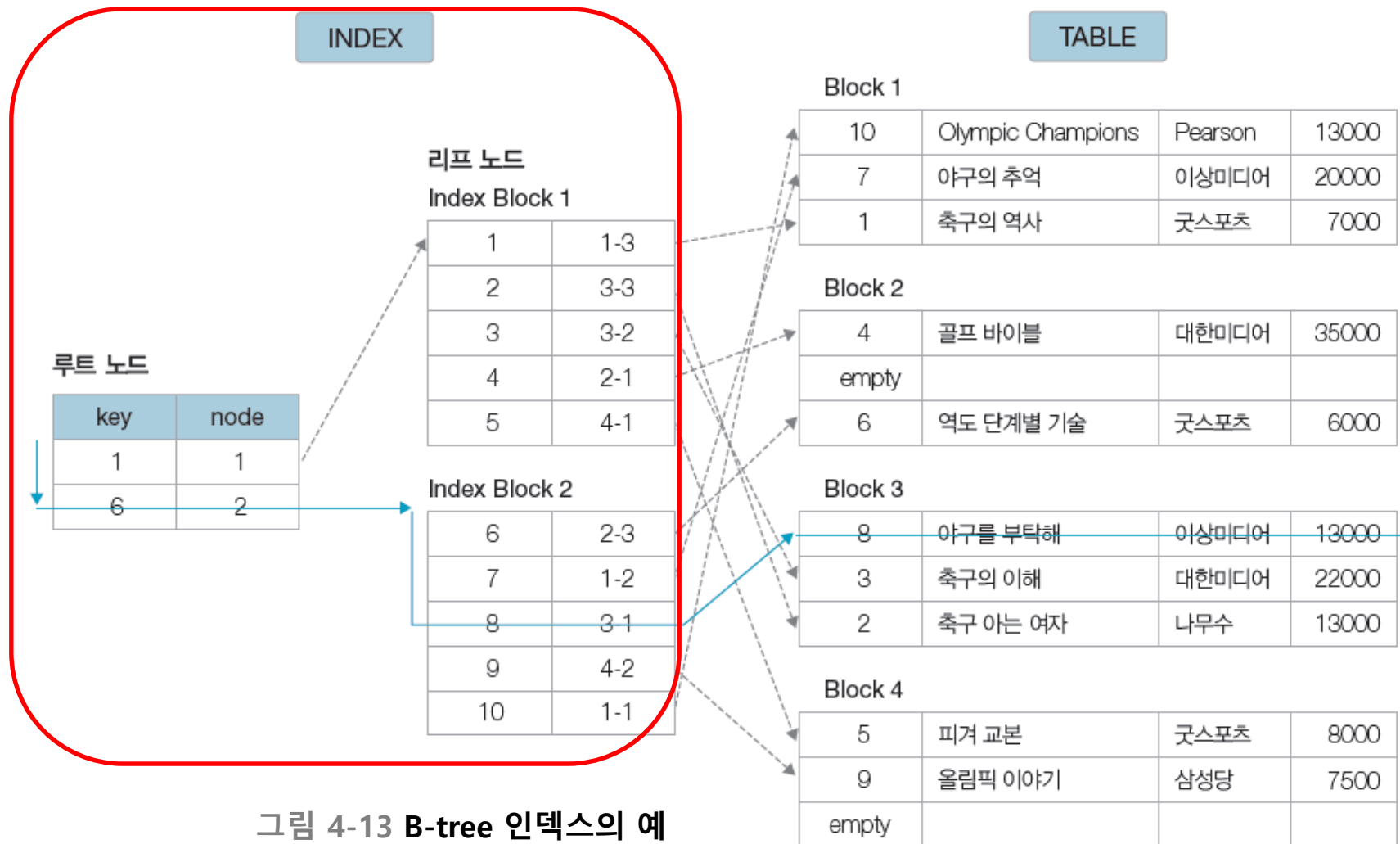
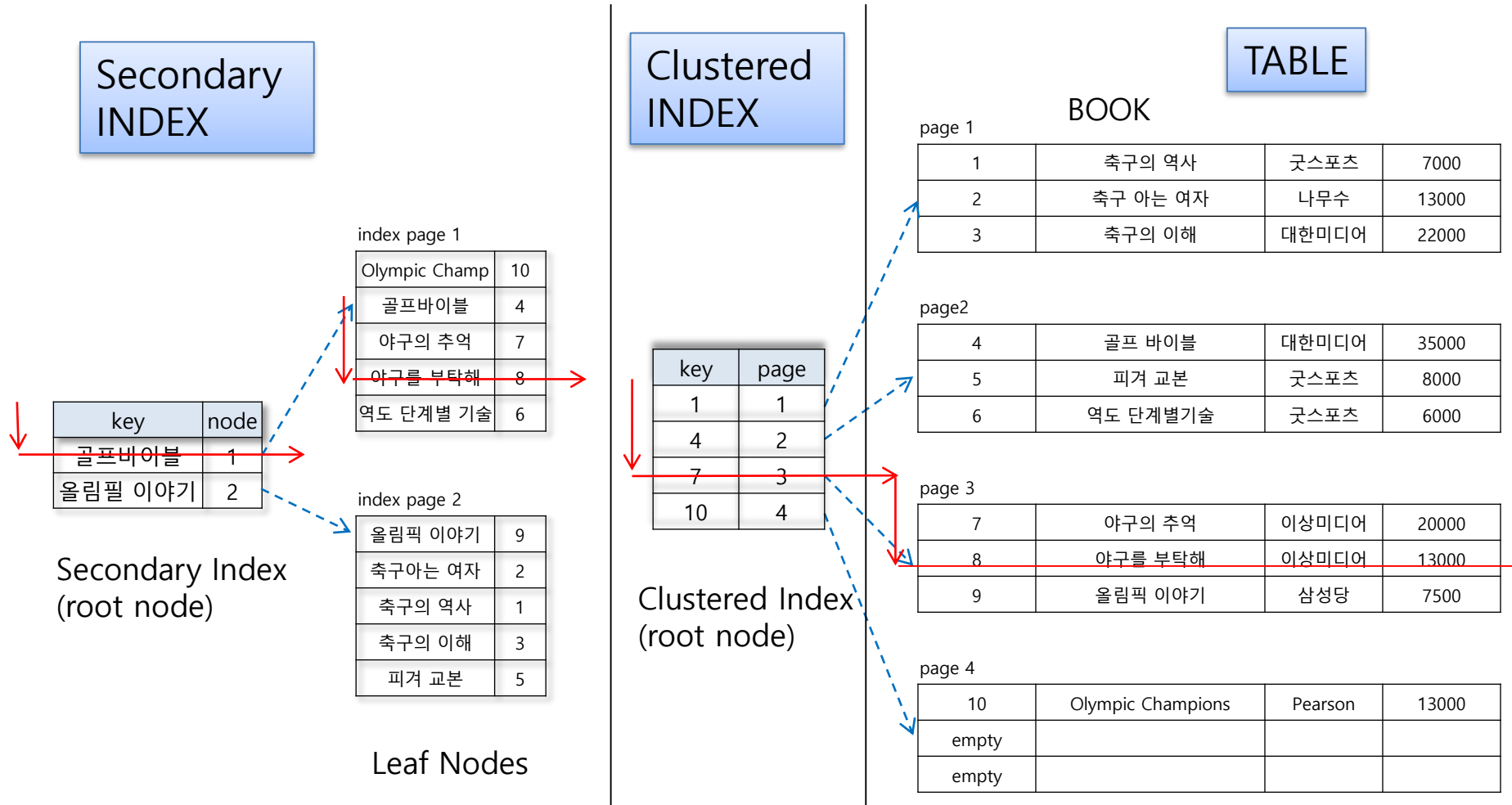


그림 4-13 B-tree 인덱스의 예

bookid 를 가지고 생성한 index이고 이렇게 생성된
인덱스의 리프노드는 실제 데이터값이 아니라 테이블상의 데이터 위치를 지정하는 rowid를 저장한다.

3. MySQL 인덱스



[그림 4-14] 클러스터 인덱스와 보조 인덱스를 동시에 사용하는 검색

4. 인덱스의 생성

❖ 인덱스 생성 시 고려사항

- 인덱스는 WHERE 절에 자주 사용되는 속성이어야 함
- 인덱스는 조인에 자주 사용되는 속성이어야 함
- 단일 테이블에 인덱스가 많으면 속도가 느려질 수 있음(테이블당 4~5개 정도 권장)
- 속성이 가공되는 경우 사용하지 않음
- 속성의 선택도가 낮을 때 유리함(속성의 모든 값이 다른 경우)

❖ 인덱스의 생성/조회 문법

```
CREATE [UNIQUE] INDEX [인덱스이름]  
ON 테이블이름 (컬럼 [ASC | DESC] [{, 컬럼 [ASC | DESC]} ...])[;]
```

SHOW INDEX FROM 테이블명

Result Grid Filter Rows: Export: Wrap Cell Content:											
	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
▶	customer	0	PRIMARY	1	custid	A	5	NULL	NULL		BTREE

4. 인덱스의 생성

질의 4-24 Book 테이블의 bookname 열을 대상으로 비 클러스터 인덱스 ix_Book을 생성하라.

```
CREATE INDEX ix_Book ON Book (bookname);
```

0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

질의 4-25 Book 테이블의 publisher, price 열을 대상으로 인덱스 ix_Book2를 생성하시오.

```
CREATE INDEX ix_Book2 ON Book(publisher, price);
```

0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
book	0	PRIMARY	1	bookid	A	10	NULL	NULL		BTREE
book	1	ix_Book	1	bookname	A	10	NULL	NULL	YES	BTREE
book	1	ix_Book2	1	publisher	A	6	NULL	NULL	YES	BTREE
book	1	ix_Book2	2	price	A	10	NULL	NULL	YES	BTREE

4. 인덱스의 생성

```
SELECT *  
FROM Book  
WHERE publisher='대한미디어' AND price >= 30000;
```

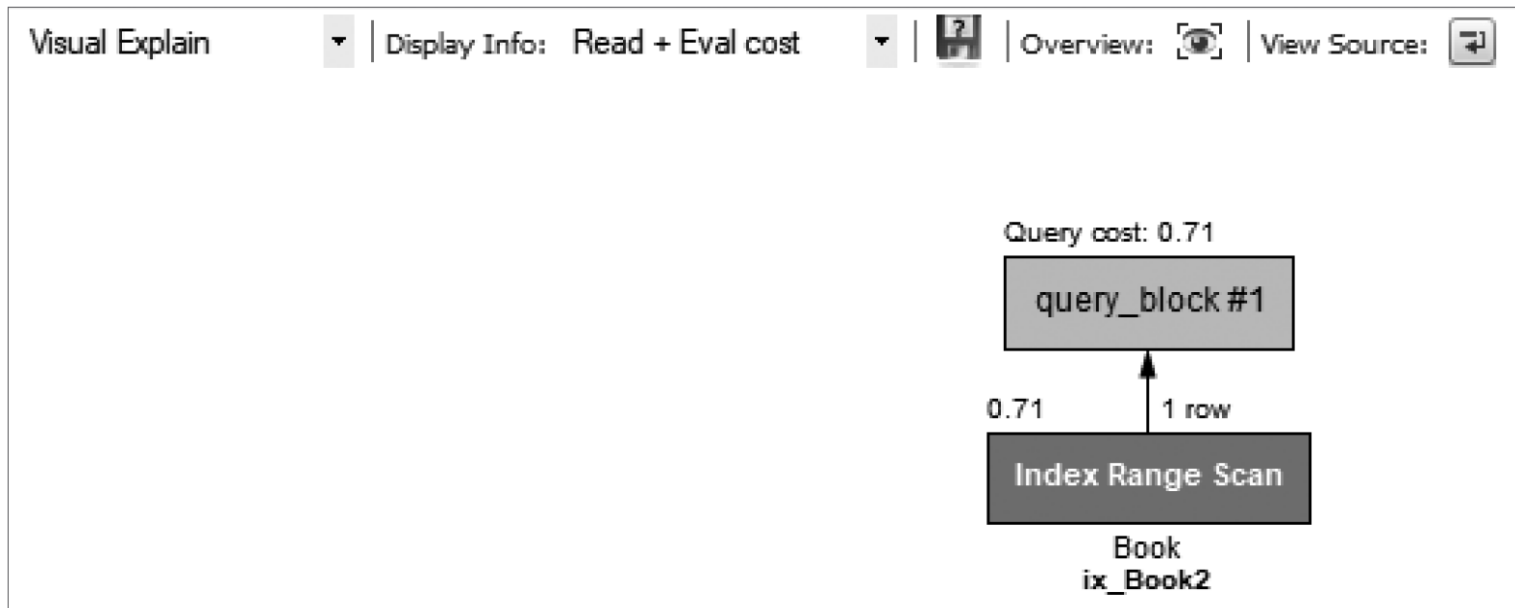


그림 4-15 실행 계획(Execution Plan)을 통한 인덱스 사용 확인

```
SHOW TABLE STATUS LIKE 'book'; // 테이블 상태 확인
```

	Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length	Max_data_length	Index_length
▶	book	InnoDB	10	Dynamic	19	862	16384	0	32768

5. 인덱스의 재구성과 삭제

- 인덱스의 재구성은 ANALYZE TABLE 명령을 사용함.
- 생성 문법

```
ANALYZE TABLE 테이블이름;
```

질의 4-26 Book 테이블의 인덱스를 최적화하시오.

```
ANALYZE TABLE Book;
```

Table	Op	Msg_type	Msg_text
madang.book	analyze	status	OK

- 삭제 문법

```
DROP INDEX 인덱스이름
```

질의 4-27 인덱스 ix_Book을 삭제하시오.

```
DROP INDEX ix_Book ON Book ;
```

0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

0.187 sec

연습문제

13. [마당서점 데이터베이스 인덱스] 마당서점 데이터베이스에서 다음 SQL 문을 수행하고 데이터베이스가 인덱스를 사용하는 과정을 확인하시오.

(1) 다음 SQL 문을 수행해본다.

```
SELECT name, address FROM Customer WHERE name LIKE '박세리';
```

(2) 실행 계획을 살펴본다. 실행 계획은 Workbench에서 [Query] → [Explain Current Statement]를 선택하면 표시된다.

(3) Customer 테이블에 name으로 인덱스를 생성하시오. 생성 후 (1)번의 SQL 문을 다시 수행하고 실행 계획을 살펴보시오.

(4) 같은 질의에 대한 두 가지 실행 계획을 비교해보시오.

(5) (3)번에서 생성한 인덱스를 삭제하시오.

요약

1. 내장 함수
2. 부속질의
3. 뷰
4. 인덱스
5. B-tree
6. MySQL 인덱스의 종류