

Report

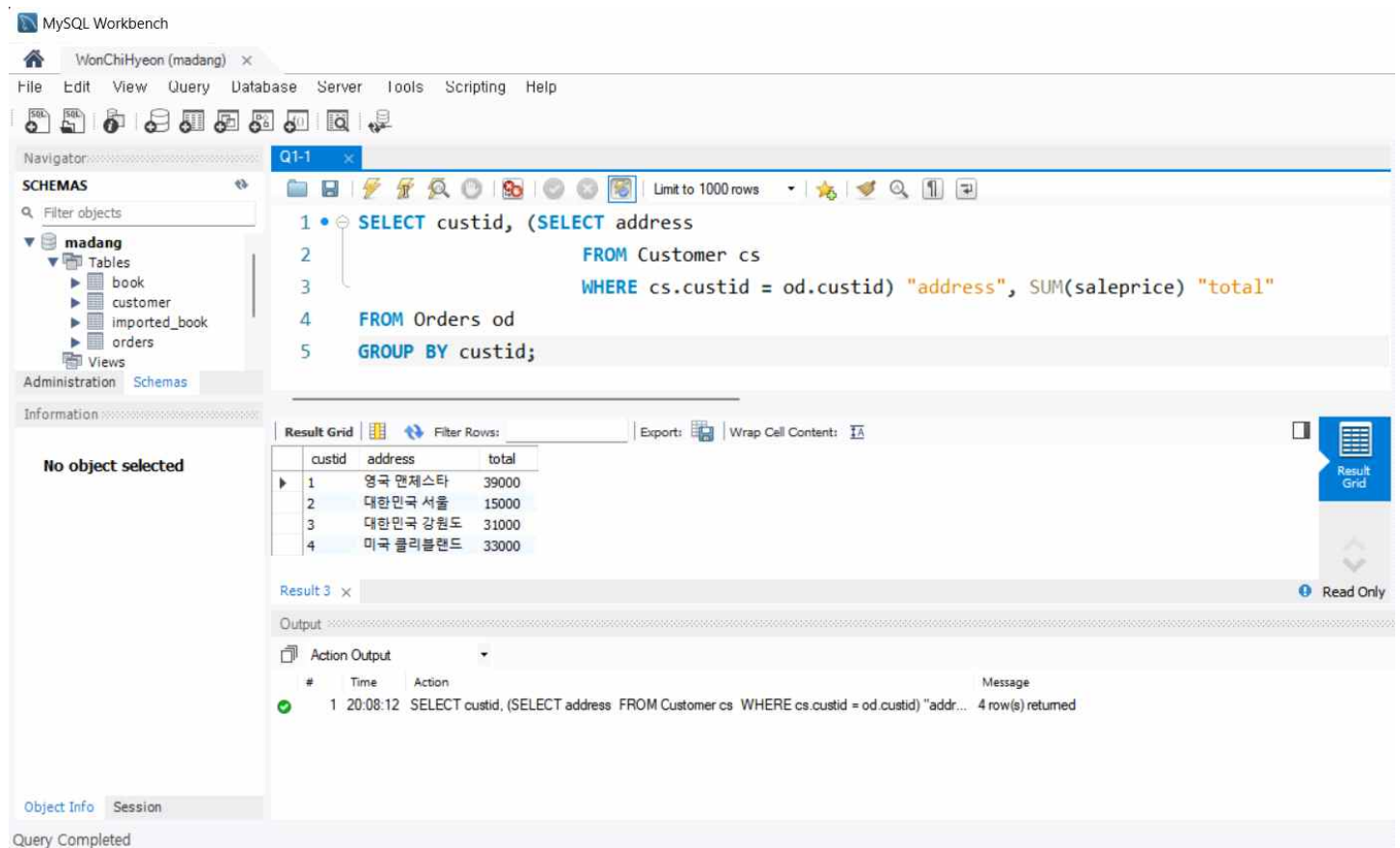
과제1 : 4장 SQL 고급에서 다뤄진 내장함수, 부속 질의 형태, 인덱스, DBMS의 물리적 구조 등과 관련된 문제



과 목	데이터베이스응용_캡스톤디자인
담당교수	한문석 교수님
이 름	원 치 현
학 번	20191486
학 과	컴퓨터공학과
제출일	2023.11.09

1. 부속질의에 관한 다음 SQL 문에서 부속질의의 종류와 무엇을 검색하는지 답하시오.
데이터베이스는 madang 데이터베이스를 이용한다.
부속질의는 SELECT, FROM, WHERE 절에 각각 포함되어 있다.
(직접 실습한 결과화면 캡처하여 삽입하고 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;



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'madang' database schema with tables 'book', 'customer', 'imported_book', and 'orders'. The main editor window contains the following SQL query:

```
1 SELECT custid, (SELECT address
2 FROM Customer cs
3 WHERE cs.custid = od.custid) "address", SUM(saleprice) "total"
4 FROM Orders od
5 GROUP BY custid;
```

The 'Result Grid' at the bottom shows the query results:

	custid	address	total
1	영국 맨체스터		39000
2	대한민국 서울		15000
3	대한민국 강원도		31000
4	미국 콜라볼랜드		33000

The 'Output' pane at the bottom shows the execution message: 'SELECT custid, (SELECT address FROM Customer cs WHERE cs.custid = od.custid) "address", SUM(saleprice) "total" FROM Orders od GROUP BY custid; 4 row(s) returned'.

SQL 문장에서 부속질의는 스칼라 부속질의입니다. Customer 테이블에서 od.custid와 일치하는 address를 한 다음Orders 테이블에서 custid별로 address와 saleprice의 합계를 검색합니다. 검색 결과는 custid, address, total 속성으로 구성된 테이블입니다.

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

The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL query:

```
1 • SELECT cs.name, s  
2 FROM (SELECT custid, avg(saleprice) s  
3 FROM Orders GROUP BY custid) od, Customer cs  
4 WHERE cs.custid = od.custid;
```

The Results window displays the query results in a table with two columns: name and s.

name	s
박지성	13000.0000
김연아	7500.0000
장미란	10333.3333
추신수	16500.0000

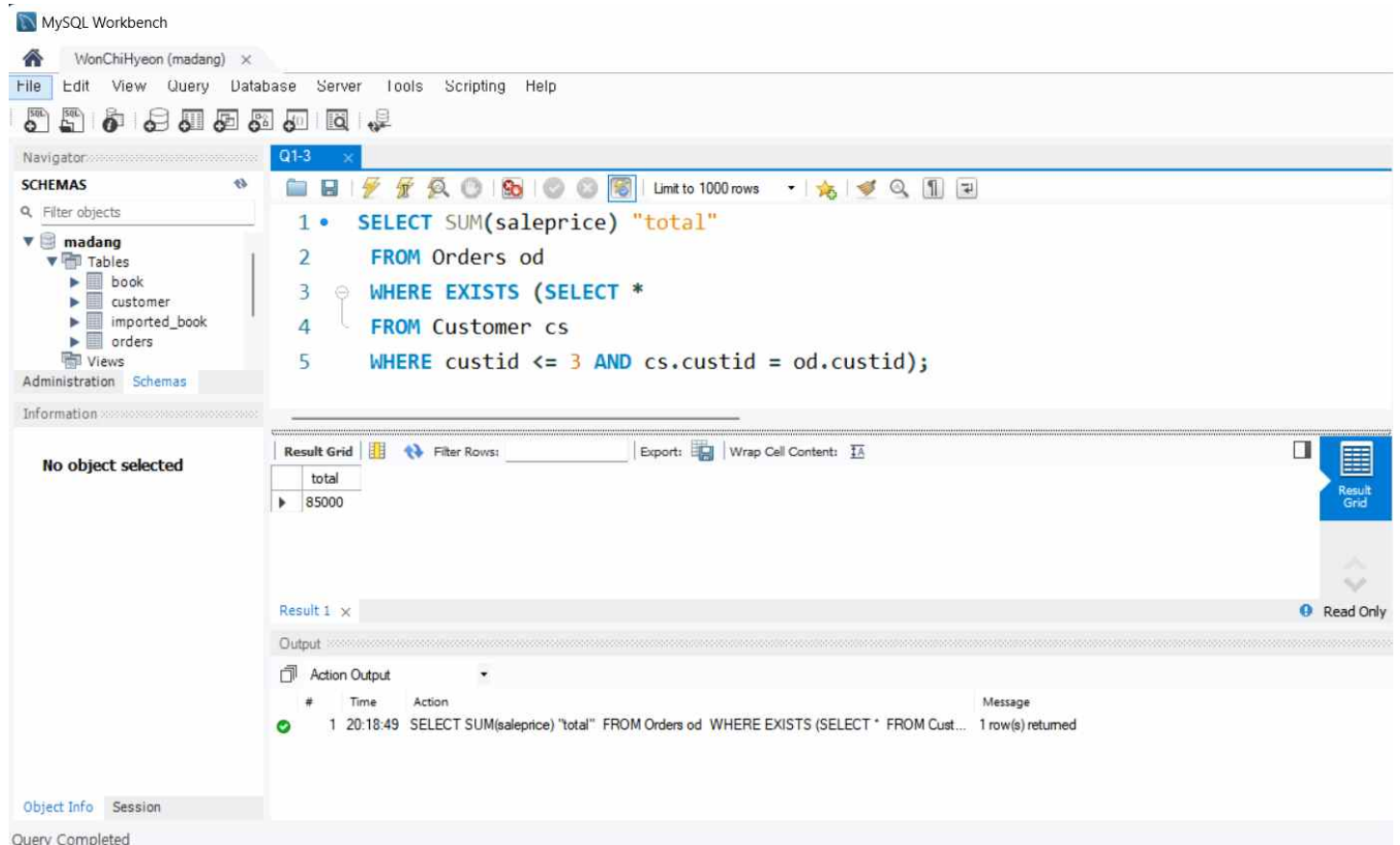
The Action Output window shows the execution details:

#	Time	Action	Message
1	20:14:17	SELECT cs.name, s FROM (SELECT custid, avg(saleprice) s FROM Orders GROUP BY cus...	4 row(s) returned

Query Completed

SQL 문장에서 부속질의는 인라인 뷰(FROM) 부속질의입니다. 이 부속질의는 Orders 테이블에서 custid별로 saleprice의 평균을 계산합니다. 그 후 Customer 테이블과 Orders 테이블을 custid 열을 기준으로 조인하고 결과는 cs.name과 s 열로 구성된 테이블입니다.

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



SQL 문장에서 부속질의는 중첩질의-WHERE 부속질의입니다. 부속질의는 Customer 테이블에서 custid가 3 이하인 고객의 custid와 Orders 테이블의 custid를 비교합니다. 그 후 Orders 테이블에서 custid별로 saleprice의 합계를 검색합니다. 그 결과는 total 열로 구성된 테이블입니다.

2. 다음에 해당하는 뷰를 작성하시오. 데이터베이스는 마당서점 데이터베이스를 이용한다. (직접 실습한 결과화면 캡처하여 삽입하고 설명하시오.)

(1) 판매가격이 20,000원인 도서의 도서번호, 도서이름, 고객이름, 출판사, 판매가격을 보여주는 highorders 뷰를 생성하시오.

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the 'madang' database with tables 'book', 'customer', 'imported_book', and 'orders', and a view 'highorders'. The 'highorders' view is selected, showing its columns: bookid, bookname, name, publisher, and saleprice. The main editor shows the SQL script for creating and querying the view. The 'Result Grid' shows the output of the query, displaying two rows of data. The 'Output' pane shows the execution log.

```
1 • CREATE VIEW highorders (bookid,bookname,name,publisher,saleprice)
2 AS SELECT od.bookid,bk.bookname,cs.name,bk.publisher,od.saleprice
3 FROM Orders od, Customer cs, Book bk
4 WHERE cs.custid=od.custid AND od.bookid=bk.bookid AND saleprice >= 20000;
5
6 • SELECT * FROM madang.highorders;
```

bookid	bookname	name	publisher	saleprice
3	죽구의 이해	박지성	대한미디어	21000
7	아구의 추억	추신수	이상미디어	20000

highorders 1 x

Output

Action Output

#	Time	Action	Message
1	20:28:11	CREATE VIEW highorders (bookid,bookname,name,publisher,saleprice) AS SELECT od boo...	0 row(s) affected
2	20:28:11	SELECT * FROM madang.highorders LIMIT 0, 1000	2 row(s) returned

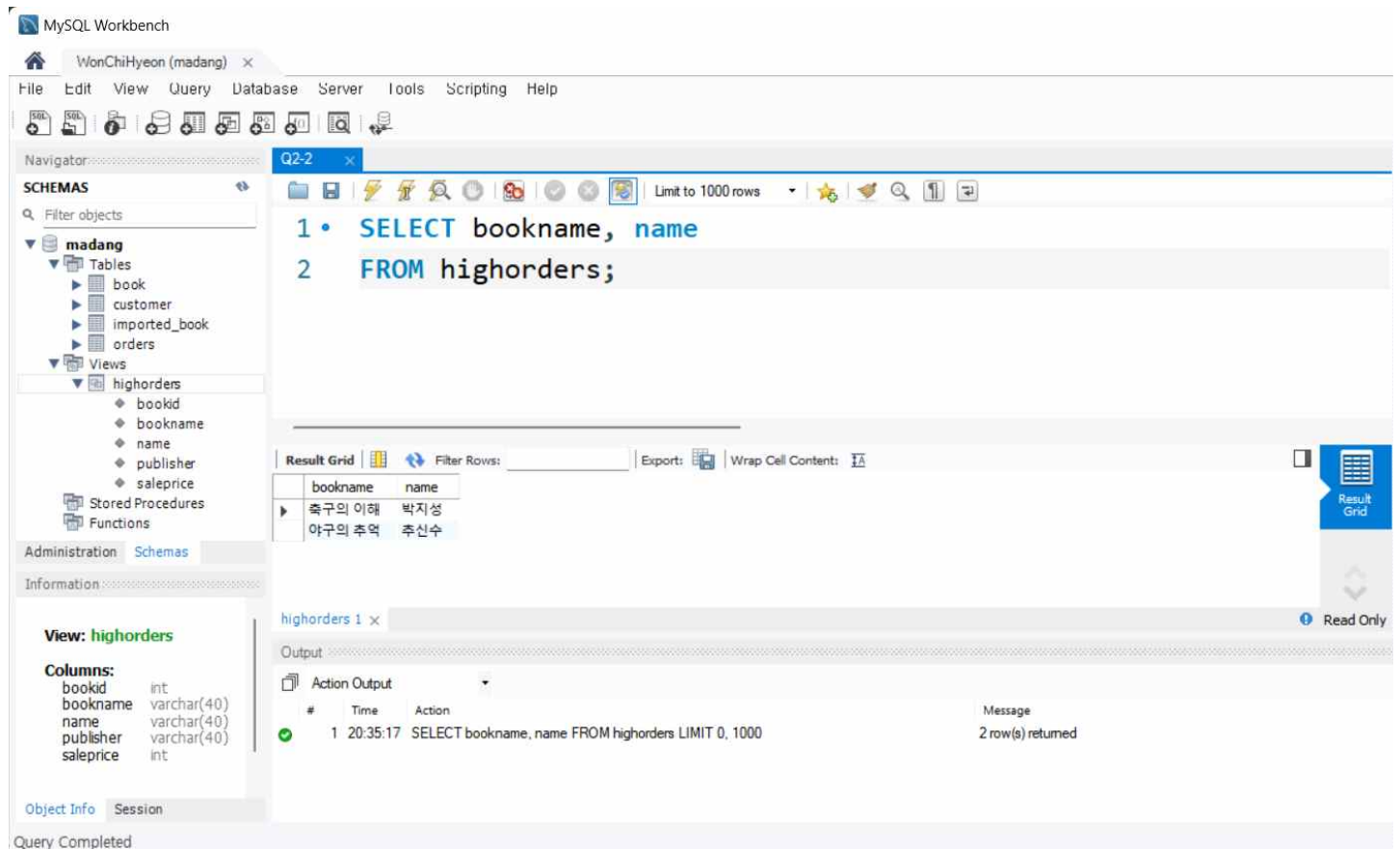
SQL script saved to 'C:\Users\ylose\Desktop\과제1 sql\Q2-1.sql'

```
CREATE VIEW highorders (bookid,bookname,name,publisher,saleprice)
AS SELECT od.bookid,bk.bookname,cs.name,bk.publisher,od.saleprice
FROM Orders od, Customer cs, Book bk
WHERE cs.custid=od.custid AND od.bookid=bk.bookid AND saleprice >= 20000;
```

다음 쿼리를 실행한 후에 SELECT * FROM madang.highorders; 문을 사용하여 highorders 뷰를 출력해 보았습니다. 이 뷰는 bookid, bookname, name, publisher, saleprice 열로 구성됩니다.

SQL문의 실행과정은 다음과 같습니다. 처음에 Orders, Customer, Book 테이블에서 각각의 레코드를 가져옵니다. Orders 테이블의 custid와 Customer 테이블의 custid가 일치하는 레코드를 선택합니다. 선택된 레코드 중, Orders 테이블의 bookid와 Book 테이블의 bookid가 일치하는 레코드를 선택하고 추가로 saleprice가 20,000원 이상인 레코드를 선택합니다. 선택된 레코드의 bookid, bookname, name, publisher, saleprice 정보를 가지고 highorders 뷰를 생성합니다.

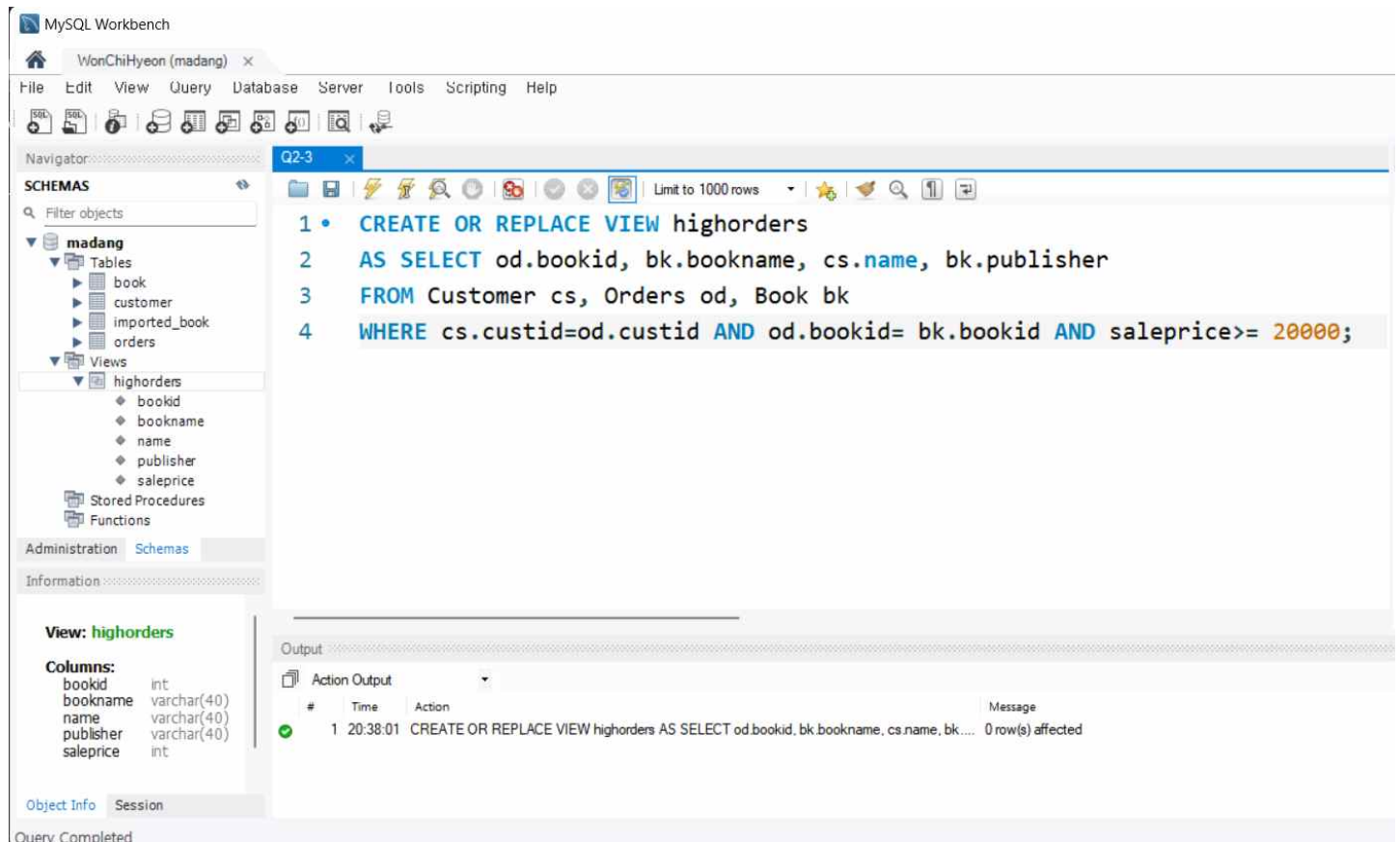
2) 생성한 뷰를 이용하여 판매된 도서의 이름과 고객의 이름을 출력하는 SQL 문을 작성하시오.



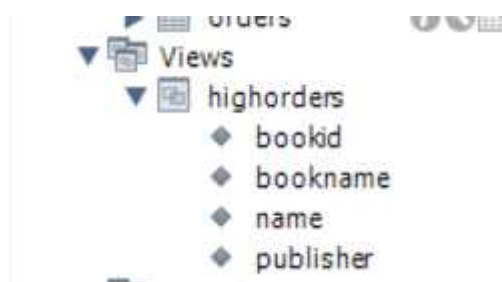
SELECT bookname, name
FROM highorders;

앞서 생성한 highorders 뷰에서 bookname과 name 레코드의 데이터를 select문을 사용하여 판매된 도서의 이름과 고객의 이름을 출력하는 SQL 문을 작성하였습니다.

(3) highorders 뷰를 변경하고자 한다. 판매가격 속성을 삭제하는 명령을 수행하시오. 삭제 후 (2)번 SQL 문을 다시 수행하시오.



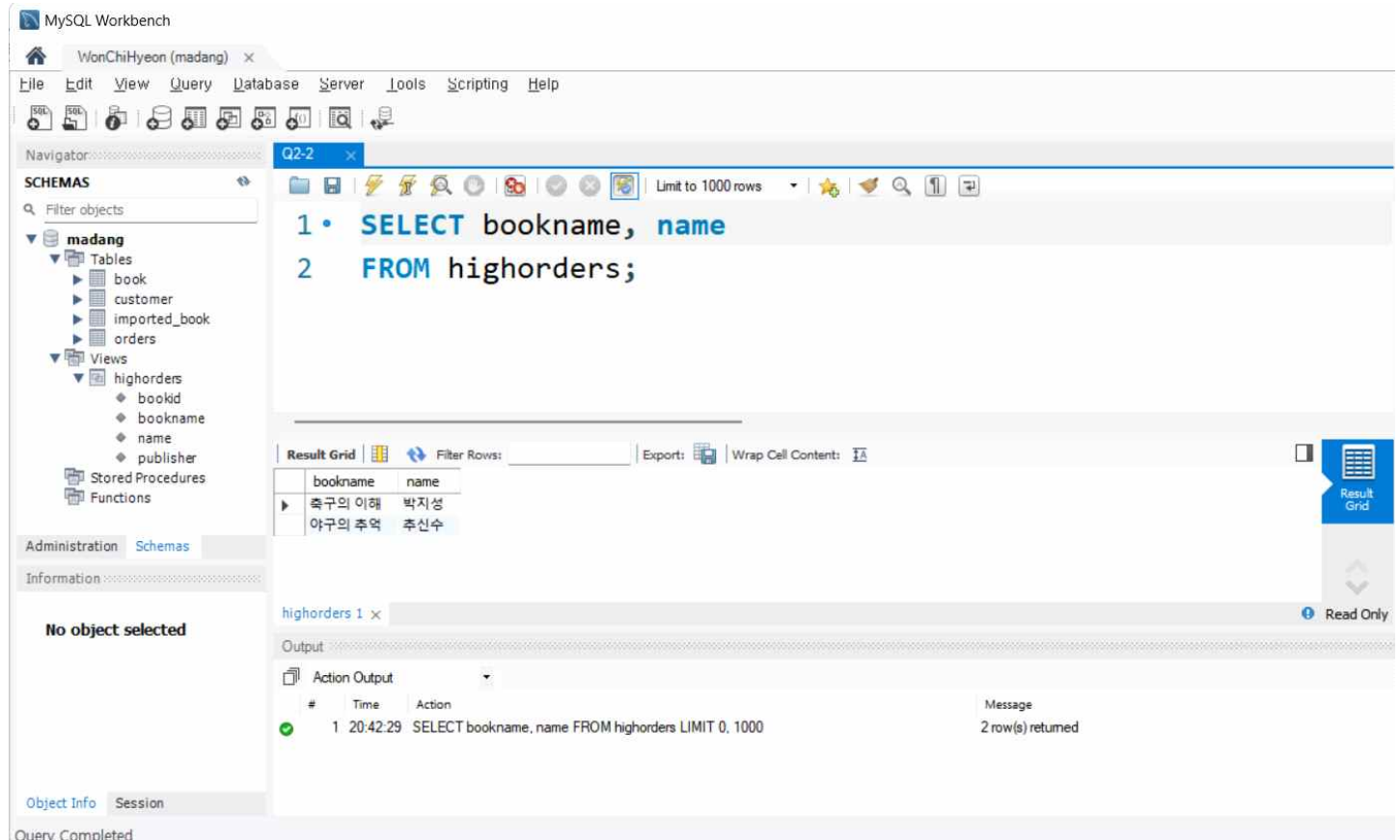
```
CREATE OR REPLACE VIEW highorders
AS SELECT od.bookid, bk.bookname, cs.name, bk.publisher
FROM Customer cs, Orders od, Book bk
WHERE cs.custid=od.custid AND od.bookid= bk.bookid AND saleprice>= 20000;
```



다음 SQL문을 실행하면 highorders 뷰의 레코드 중 판매가격 속성(saleprice)가 사라지고 bookid, bookname, name, publisher 레코드만 남게됩니다.

실행과정은 다음과 같습니다. Customer, Orders, Book 테이블에서 각각의 레코드를 가져옵니다. Customer 테이블의 custid와 Orders 테이블의 custid가 일치하는 레코드를 선택합니다. Orders 테이블의 bookid와 Book 테이블의 bookid가 일치하는 레코드를 선택합니다. 선택된 레코드 중, saleprice가 20,000원 이상인 레코드를 선택합니다. 선택된 레코드의 bookid, bookname, name, publisher 정보를 가지고 highorders 뷰를 생성합니다.

삭제 후 (2)번 SQL 문을 다시 수행하시오.



앞서 판매가격 속성을 삭제하고 난 후의 수정된 highorders 뷰에서 bookname과 name 레코드의 데이터를 select문을 사용하여 판매된 도서의 이름과 고객의 이름을 출력하는 SQL 문을 작성하였습니다.

결과는 Q2-2의 결과와 같습니다. 뷰의 수정은 CREATE 구문에 OR REPLACE가 합쳐진다는 것을 알 수 있었습니다.

3. 다음 데이터를 순서대로 삽입할 때 B-트리를 단계별로 구축하는 과정을 순서대로 그림으로 보이시오. (단, B-트리의 Order(최대 차수)는 3이라고 가정한다.)

키값들의 삽입 순서: 8, 5, 1, 7, 3, 12, 9, 6

이 문제에서는 B-트리의 최대 차수가 3이므로, 한 노드에 최대 2개의 데이터가 들어갈 수 있습니다. 다음은 주어진 키값들을 순서대로 삽입하여 B-트리를 구축하는 과정입니다

키값들의 삽입 순서: 8, 5, 1, 7, 3, 12, 9, 6

그림과 간단한 설명으로 B-트리를 구축하는 과정을 설명드리겠습니다
루트 노드를 생성합니다. 루트 노드는 처음에는 빈 상태입니다.



8을 루트 노드에 삽입합니다.

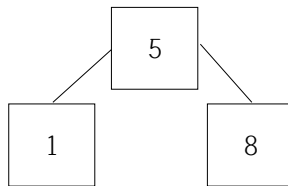
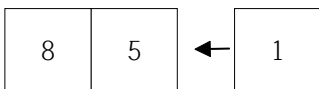


5를 삽입합니다.

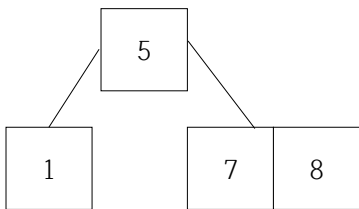


1을 삽입합니다.

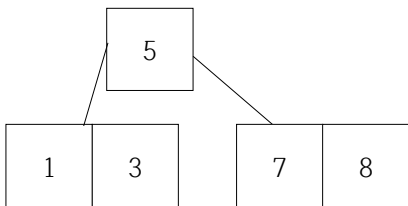
이 때 오버플로우가 발생하고 높이가 1인 이진트리로 분할되게 됩니다.



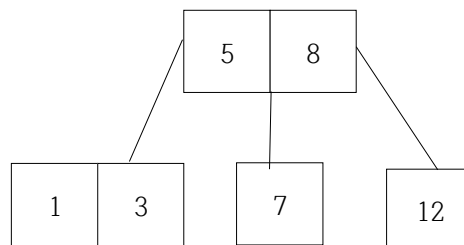
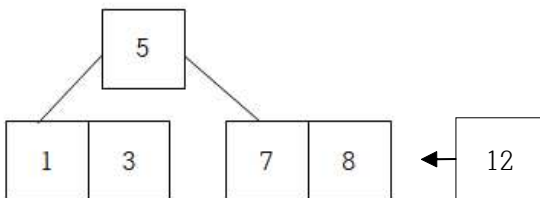
7을 삽입합니다.



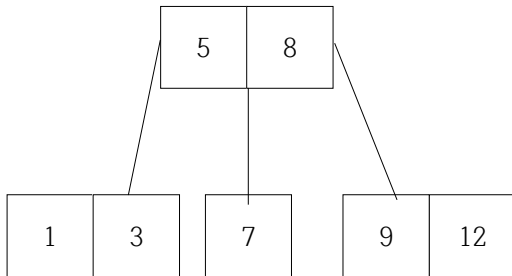
3을 삽입합니다.



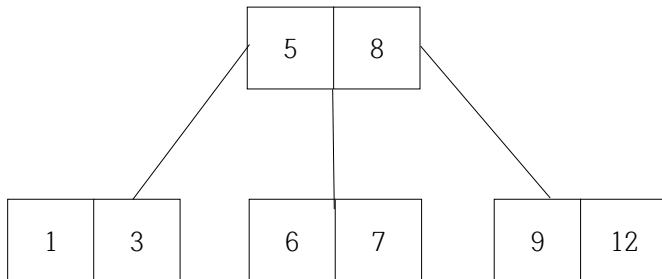
12를 삽입합니다. 이때 오버플로우가 발생합니다.



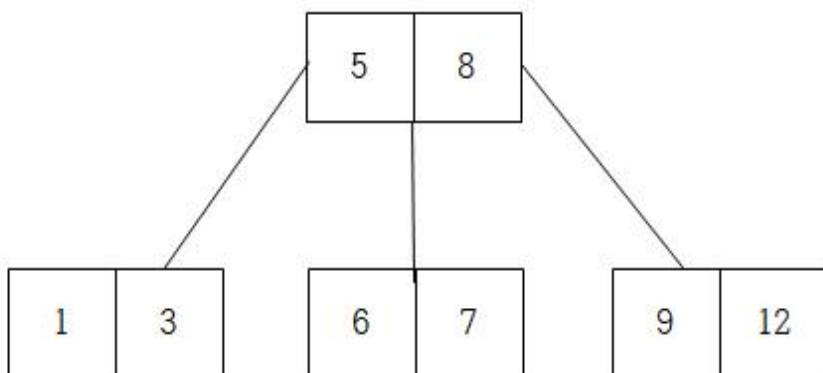
9를 삽입합니다.



마지막으로 6을 삽입합니다.



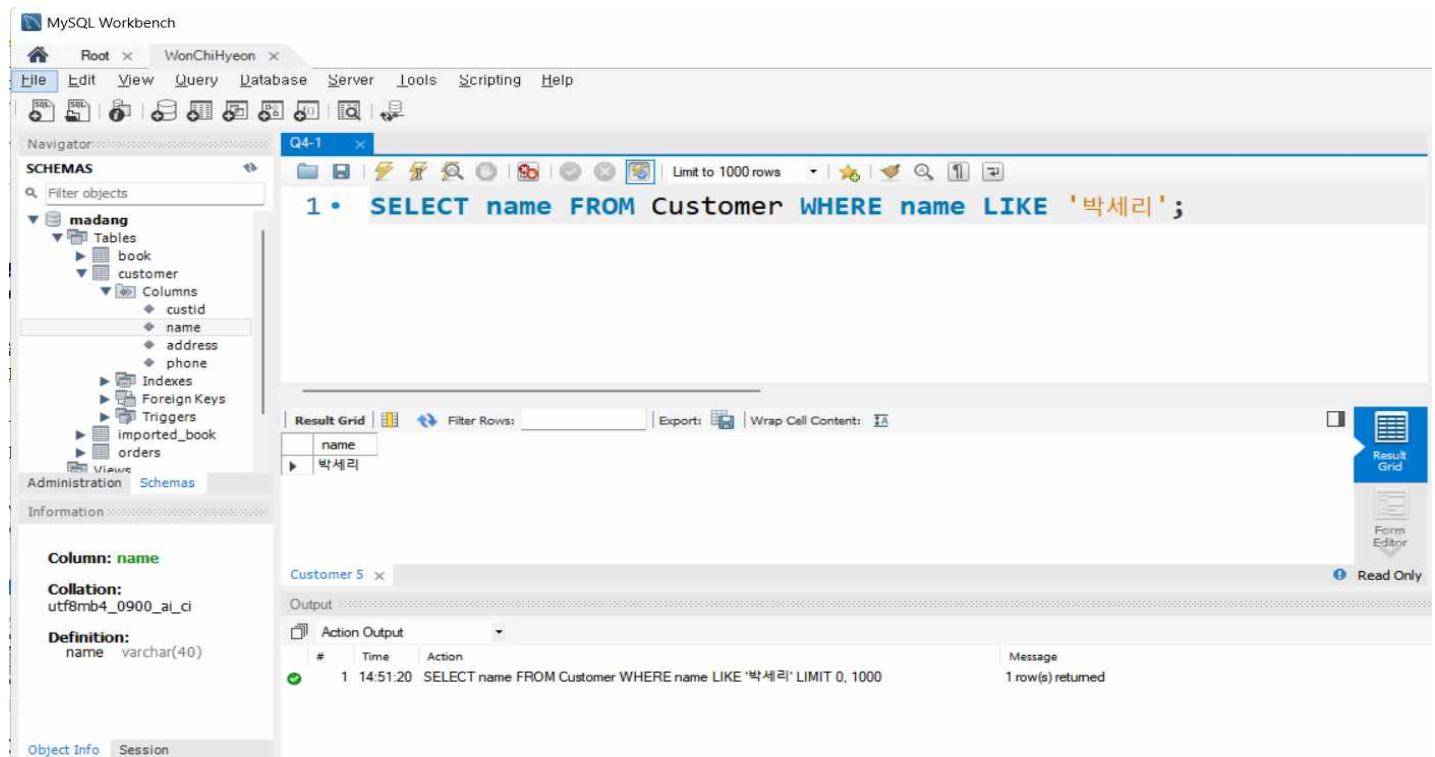
주어진 키값들을 순서대로 삽입하여 구축된 B-트리는 최종적으로 다음과 같습니다 :)



4. [마당서점 데이터베이스 인덱스] 마당서점 데이터베이스에서 다음 SQL 문을 수행하고 데이터베이스가 인덱스를 사용하는 과정을 확인해보고 실행화면을 문제별로 캡처하여 제출하시오.

(1) 다음 SQL 문을 각각 수행한 결과 캡처 화면을 삽입하시오.
EXPLAIN문 은 실행 계획 정보를 받을 때 테이블 형식으로 결과를 보여준다.

다음은 SELECT name FROM Customer WHERE name LIKE '박세리';
SQL문을 수행한 실행화면 캡처입니다.



이 SQL문을 실행하면 인덱스를 사용하는 과정은 다음과 같습니다.
데이터베이스는 Customer 테이블에서 name 컬럼의 값이 '박세리'와 일치하는 모든 행을 찾습니다.
데이터베이스는 name 컬럼에 인덱스가 있는지 확인합니다.
인덱스가 있다면, 데이터베이스는 인덱스를 사용하여 해당 행을 찾습니다.

EXPLAIN SELECT name FROM Customer WHERE name LIKE '박세리';

MySQL Workbench interface showing the execution of the query: `SELECT name FROM Customer WHERE name LIKE '박세리';`

The Visual Explain view displays the execution plan for the query. The plan shows a **Full Table Scan** on the **Customer** table, with a **Query cost: 0.75** and **5 rows** returned.

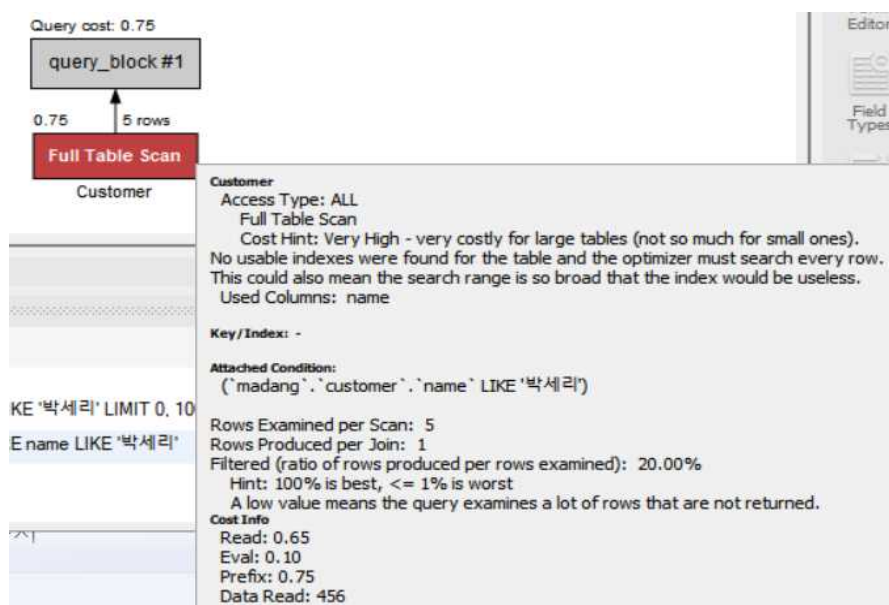
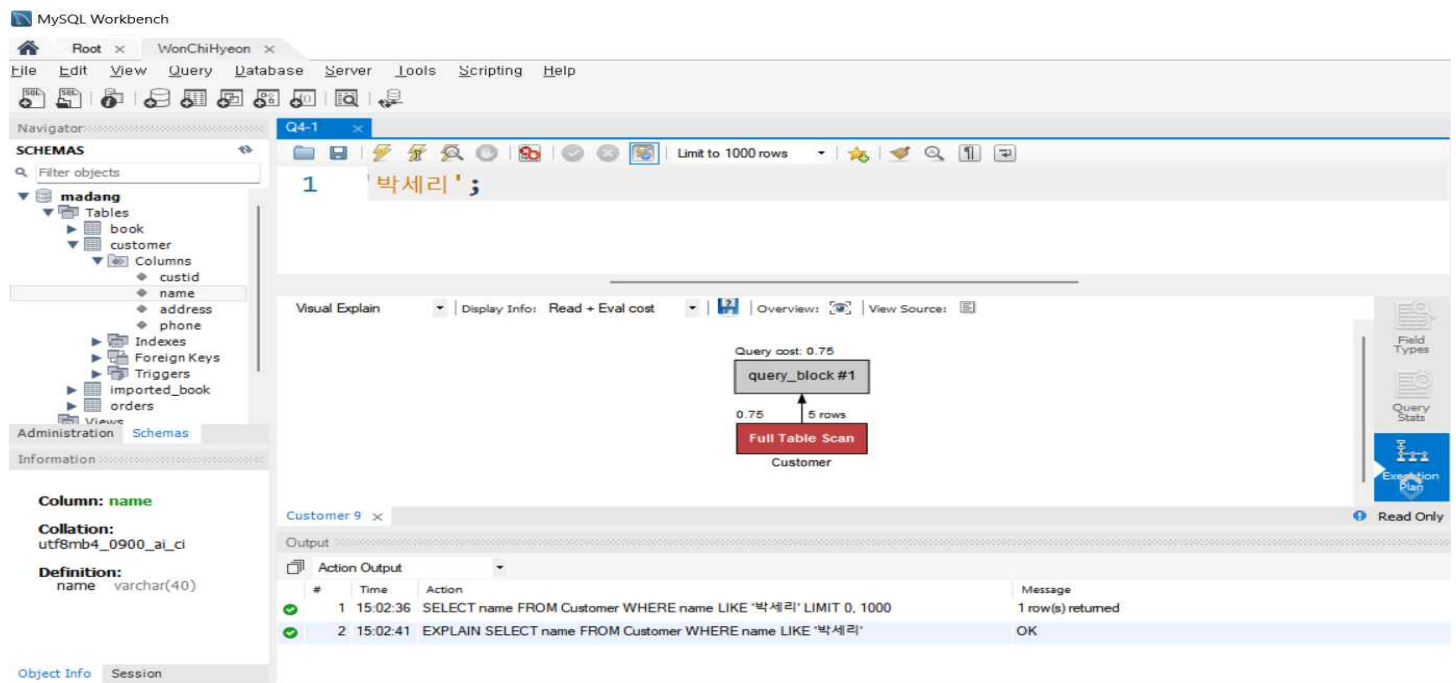
The Output tab shows the results of the query execution:

#	Time	Action	Message
1	15:20:09	SELECT name FROM Customer WHERE name LIKE '박세리' LIMIT 0, 1000	1 row(s) returned
2	15:20:14	EXPLAIN SELECT name FROM Customer WHERE name LIKE '박세리'	OK

이 SQL문을 실행하면 인덱스를 사용하는 과정은 4-1번 SQL을 실행시켰을 때의 과정과 같습니다.

다만 EXPLAIN을 사용하면 데이터베이스가 인덱스를 사용하여 쿼리를 처리하는지 여부를 확인할 수 있습니다. 결과 화면으로 id, select_type, table, partitions, type, possible_keys, key, key_len, ref, rows, filtered, Extra 정보를 테이블 Grid의 형태로 출력하는 것을 알 수 있습니다.

(2) (1)번 질의의 실행 계획(Execution Plan)에 대한 캡처 화면을 삽입하시오.
 실행 계획에 출력된 그림에 대한 의미를 설명하시오.
 MySQL Workbench에서 수행 방법: 질의를 수행한 후 실행 결과 창의 우측에 Execution Plan 키를 선택

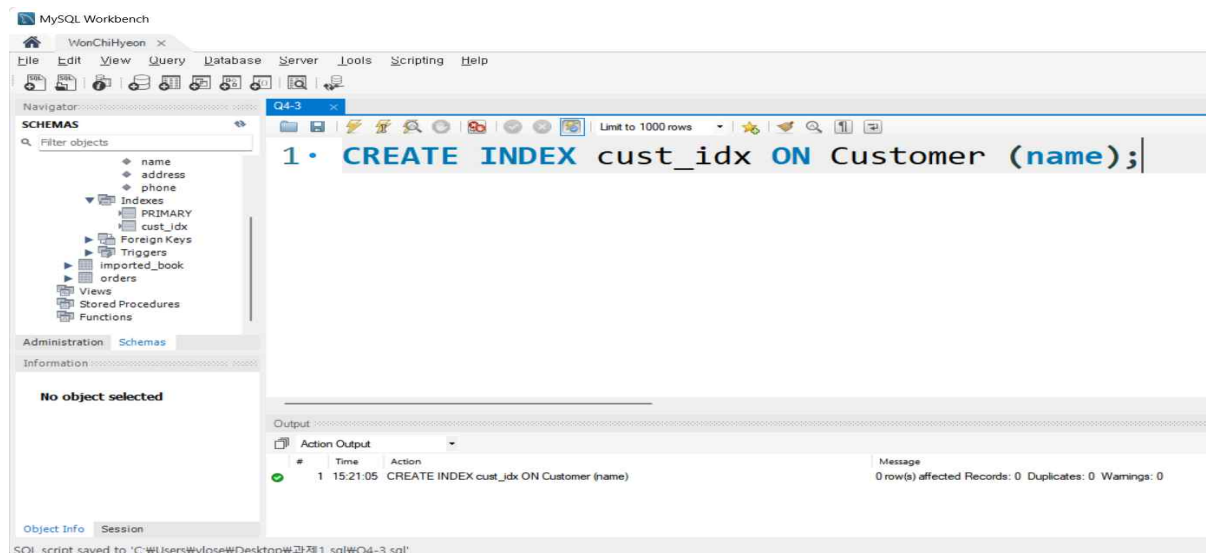


이것은 SQL 쿼리의 실행 계획에 대한 다이어그램입니다. 다이어그램은 세 개의 노드를 가진 트리 구조입니다. 쿼리를 실행하면 나오는 그림에서 각 연산의 비용(Query cost)을 알 수 있습니다. 쿼리를 실행할 때 사용한 인덱스(Indexes)와 테이블(Customer)을 알 수 있습니다.

(실행 계획에서 인덱스를 사용하는지 여부를 확인할 수 있습니다.)
 쿼리 비용, 반환된 행 수 및 수행된 작업을 보여줍니다. 이 경우, 쿼리 비용은 0.75이며, 5개의 행이 반환되며, Customer 테이블에서 전체 테이블 스캔이 수행됩니다.

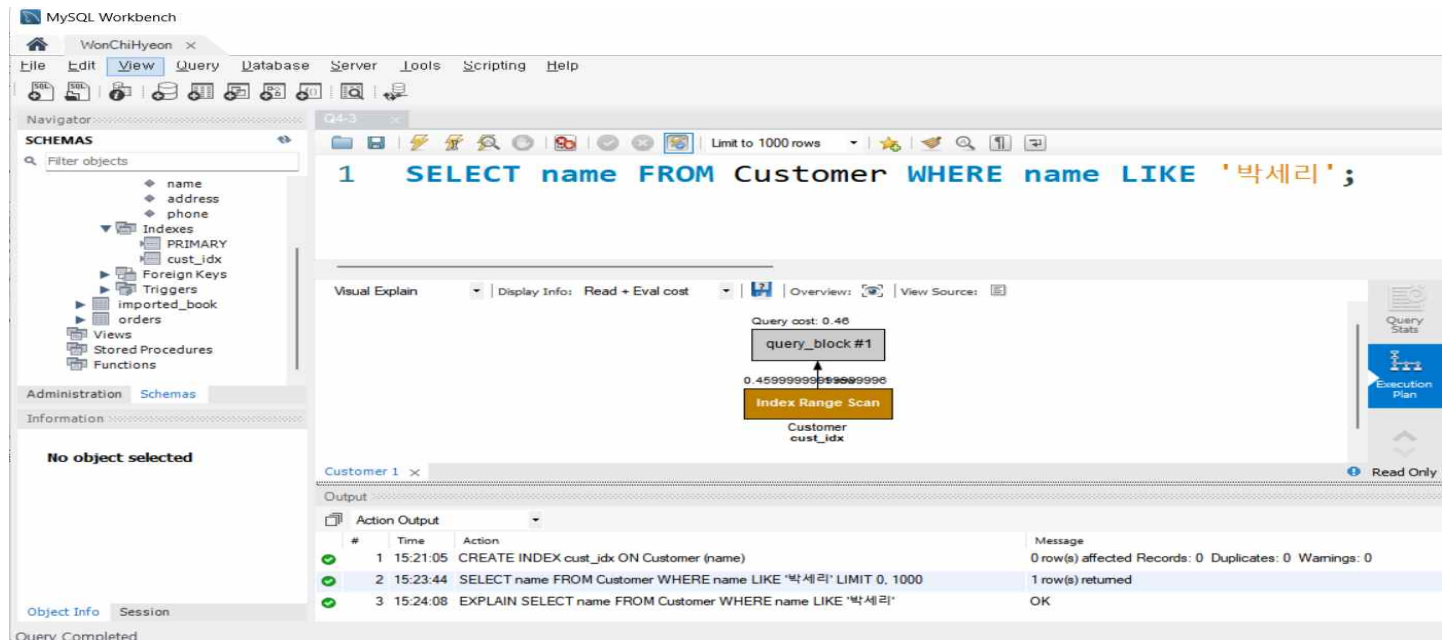
(3) Customer 테이블에 name으로 cust_idx라는 이름으로 인덱스를 생성하시오.
생성 후 (1)번의 첫 번째 SQL 문을 다시 수행하고 실행 계획에 대한 캡처 화면을 삽입하시오.

실행한 sql문 : CREATE INDEX cust_idx ON Customer (name);



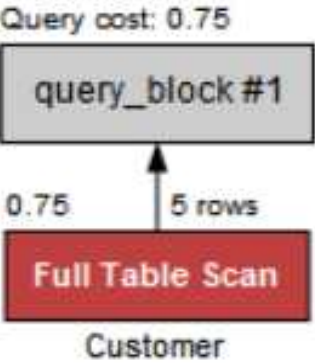
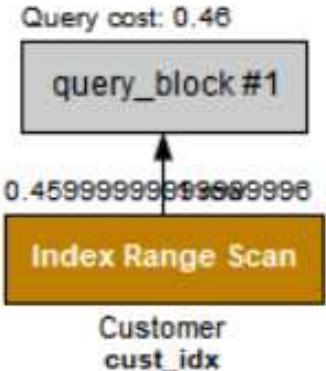
SQL문을 실행하면 우측 schemas창의 Indexes 밑에 cust_idx 이름의 인덱스가 생성됨을 확인할 수 있습니다.

실행한 (1)번의 첫 번째 SQL 문 : SELECT name FROM Customer WHERE name LIKE '박세리';



이 다이어그램은 SQL 쿼리의 실행 계획을 보여줍니다. 쿼리의 비용과 실행 단계가 표시됩니다. 쿼리의 비용은 0.48이고, 인덱스 범위 스캔의 비용은 0.45999999999999996입니다. 인덱스 범위 스캔은 고객 테이블(Customer)을 대상으로 하며, 인덱스는 cust_idx입니다.

(4) 같은 질의에 대한 두 가지 실행 계획을 비교하여 어떤 변화가 있는지 설명하시오.

첫 번째 실행 계획	두 번째 실행 계획
	
<p>쿼리 비용은 0.75이며, 5개의 행이 반환됩니다.</p> <p>인덱스 범위 스캔은 Customer 테이블에서 전체 테이블 스캔이 수행됩니다.</p>	<p>쿼리의 비용은 0.48이고, 인덱스 범위 스캔의 비용은 0.45999999999999996입니다.</p> <p>인덱스 범위 스캔은 고객 테이블(Customer)을 대상으로 하며, 인덱스는 cust_idx입니다.</p>

첫 번째 그림은 인덱스를 생성하고 사용하기 전의 쿼리를 실행한 결과이며
두 번째 그림은 cust_idx 인덱스를 생성하고 실행한 쿼리의 결과입니다.

두 그림의 쿼리 비용을 비교하면 첫 번째 그림의 쿼리 비용은 0.75이고
두 번째 그림의 쿼리 비용은 0.48이라는 것을 알 수 있습니다. 두 번째 그림의 쿼리 비용이 0.27 낮습니다.

또 인덱스 범위 스캔을 비교하면 첫 번째 그림은 Customer 테이블에서 전체 테이블을 스캔하고
두 번째 그림은 Customer 테이블에서 cust_idx 인덱스만을 스캔합니다.

두 그림의 차이의 결과를 정리하자면 인덱스를 사용한 두 번째 그림은 데이터베이스가 테이블의 모든 행을 검색하는 첫 번째 그림에 반해 모든 테이블을 검색하지 않고도 데이터를 찾을 수 있습니다.
즉, 인덱스를 사용하면 데이터베이스의 성능을 향상시키고, 쿼리(검색) 비용을 낮출 수 있는 장점이 있습니다.