(09/26)

**데이터베이스**

**4주차**

분반 : 01

이름 : 김민석

학번 : 20195124

학과 : 소프트웨어융합 대학

**순서**

**과제 1**

1. 하단 재명명 연산
2. 스트링 연산
3. 데이터 출력의 순서화
4. UNION, UNION ALL (합집합)
5. INTERSECT (교집합)
6. EXCEPT (차집합)
7. 카티전 곱
8. INNER JOIN (WHERE 절 사용)
9. INNER JOIN (INNER JOIN과 ON 사용)
10. OUTER JOIN (LEFT)
11. OUTER JOIN (RIGHT)
12. 집성함수 (AVG, MIN, MAX, SUM, COUNT) – WHERE 절 추가
13. GROUP BY, HAVING 절
14. IN & NOT IN
15. SOME & ALL
16. EXISTS & NOT EXISTS
17. FROM 절의 부질의

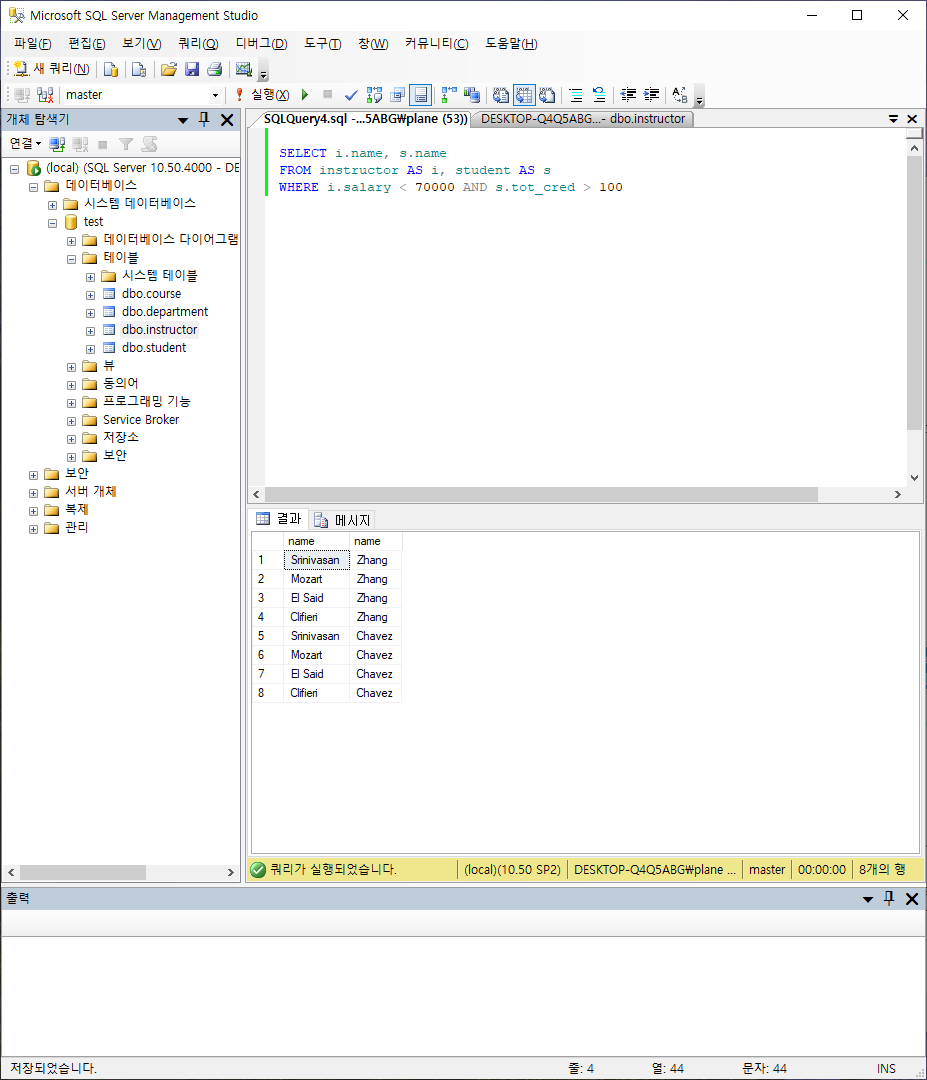
**과제 2**

1. section 테이블에서 모든 과목의 코드(c\_id) 중복 없이 검색
2. student & takes 테이블에서 ‘Zhang’ 학생이 들은 과목의 코드(c\_id)와 성적(grade) 검색
3. instructor & teaches 테이블에서 ‘Comp.Sci.’ 학과 소속의 강사의 수업 중 2009년에 수업한 과목의 코드(c\_id)를 검색
4. takes & course 테이블에서 각 학생이 들은 과목에 대해 학생(s\_id)별로 취득한 총 학점수(누적된 학점수)를 구하기

**과제 1**

* **하단 재명명 연산**

**[쿼리 결과]**

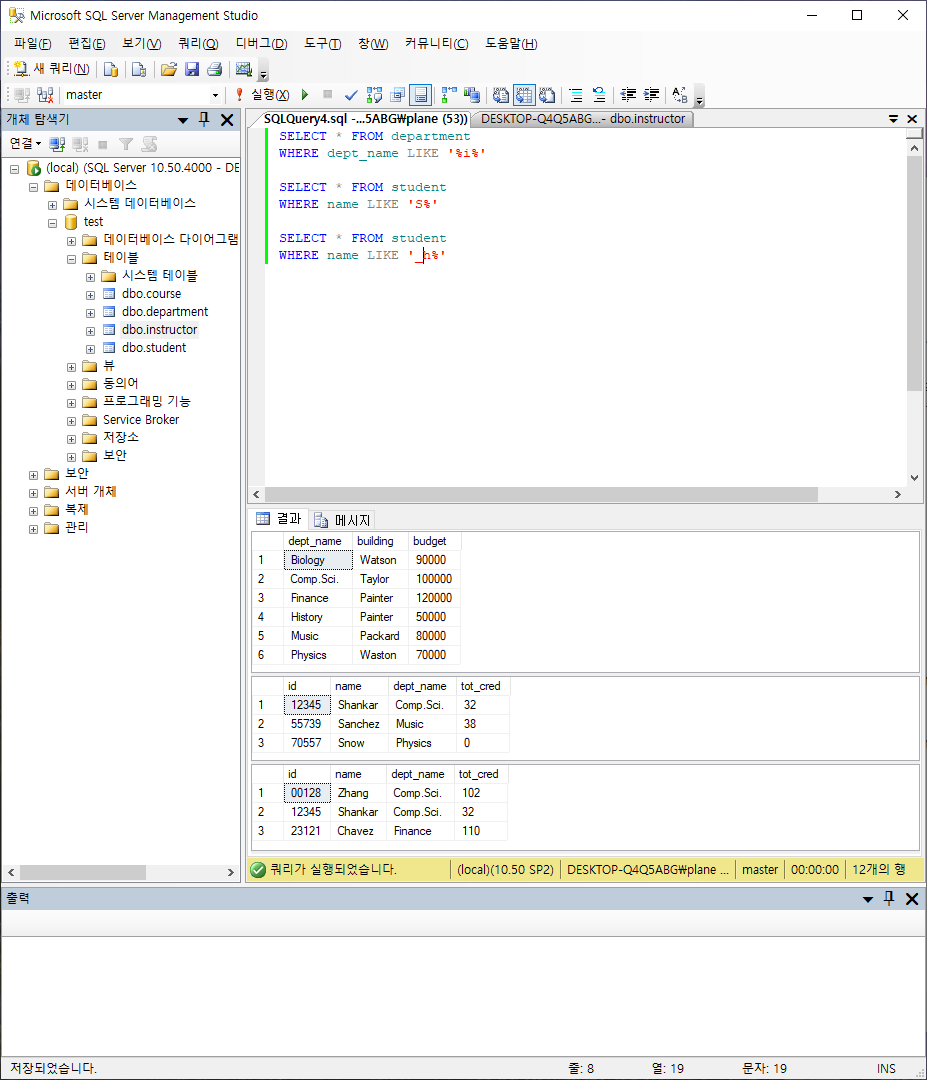


**[설명]**

instructor 와 student 테이블로부터 이름 두개를 가져오는데 둘의 name 필드가 같으므로 AS 명령어로 instructor 와 student를 줄임과 동시에 구별을 해줌으로써 FROM 명령어로 instructor는 i 로, student는 s 로 이름을 선언해서 연산을 수행한다.

* **스트링 연산**

**[쿼리 결과]**



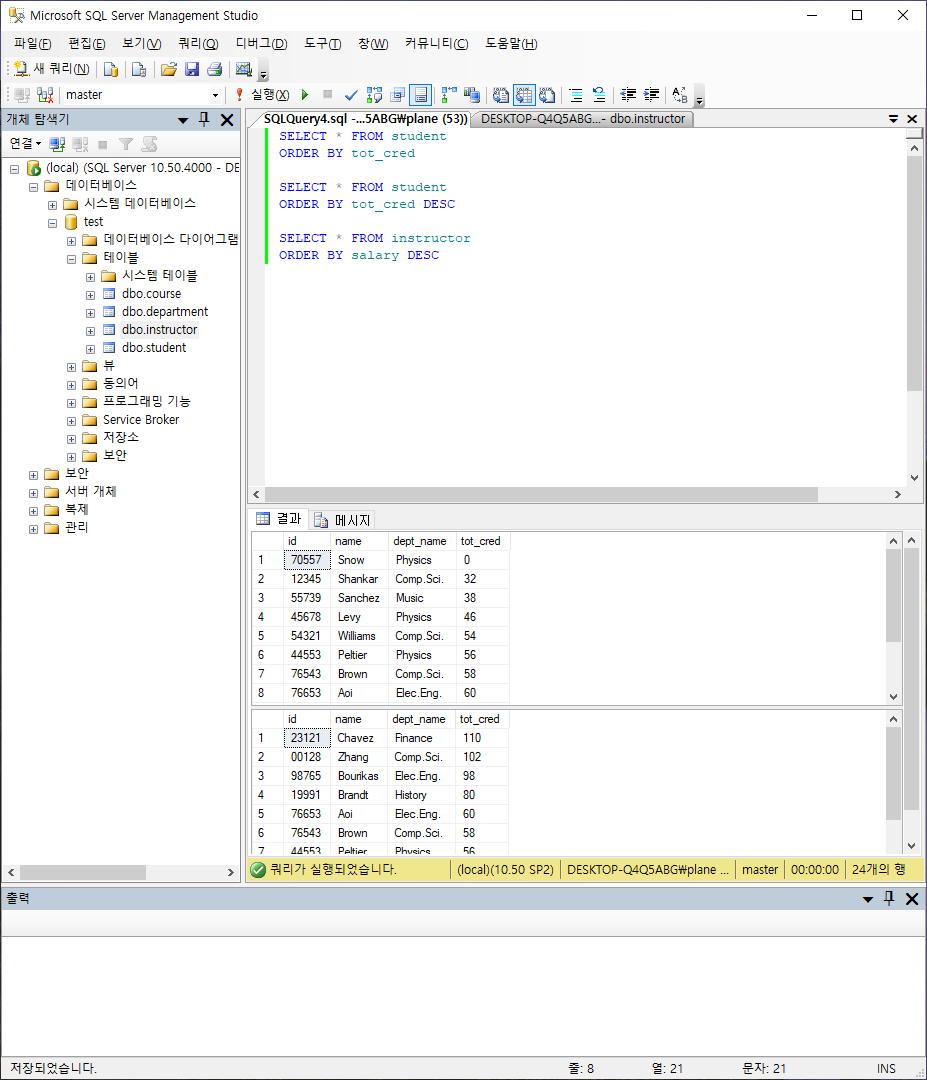
**[설명]**

SELECT FROM WHERE 문으로 특정 테이블에서 조건에 따라 값을 가져올 때 WHERE 절에서 LIKE ‘format’을 통해 원하는 값들을 정할 수 있는데, ‘ ‘ 사이에 찾고자 하는 값을 입력하고 % 나 \_ 를 쓰면 된다. 이때 %를 쓴 방향으로는 어떤 글자든, 몇 글자든 상관없이 모조리 가져오는 의미이고 \_는 어떤 글자든 상관없지만, 단 한 글자만 가져온다는 의미로 쓰인다.

Ex) ‘\_i%’ 의 결과로는 History, Finance 만 올 수 있다.

* **데이터 출력의 순서화**

**[쿼리 결과]**

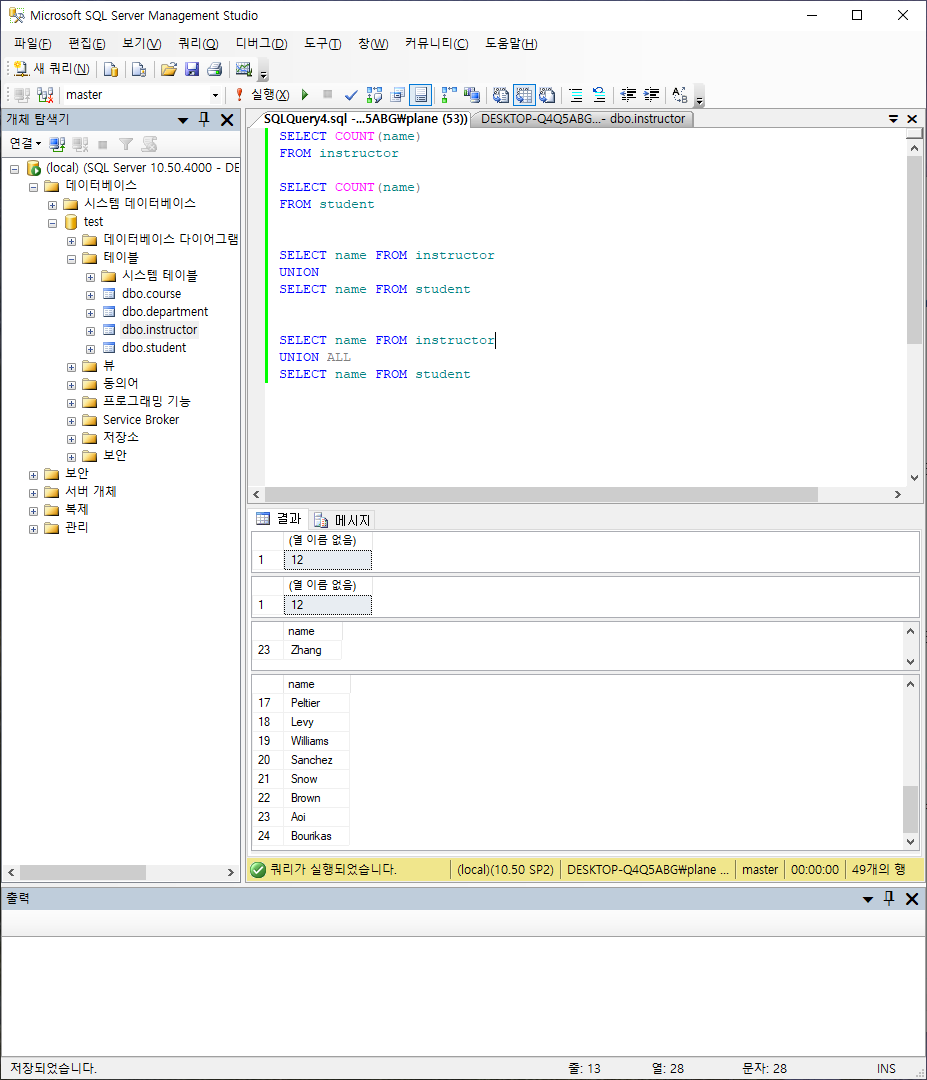


**[설명]**

데이터를 출력하는 데 있어서 특정 열들의 값을 정렬할 수 있는데 그때 쓰이는 명령어가 ORDER BY 명령어이다. 기본적으로 ORDER BY 는 오름차순으로 정렬을 해주며, ORDER BY 데이터 **DESC**라 쓰게 될 경우 내림차순으로 데이터를 정렬해서 나타내준다.

* **UNION, UNION ALL (합집합)**

**[쿼리 결과]**

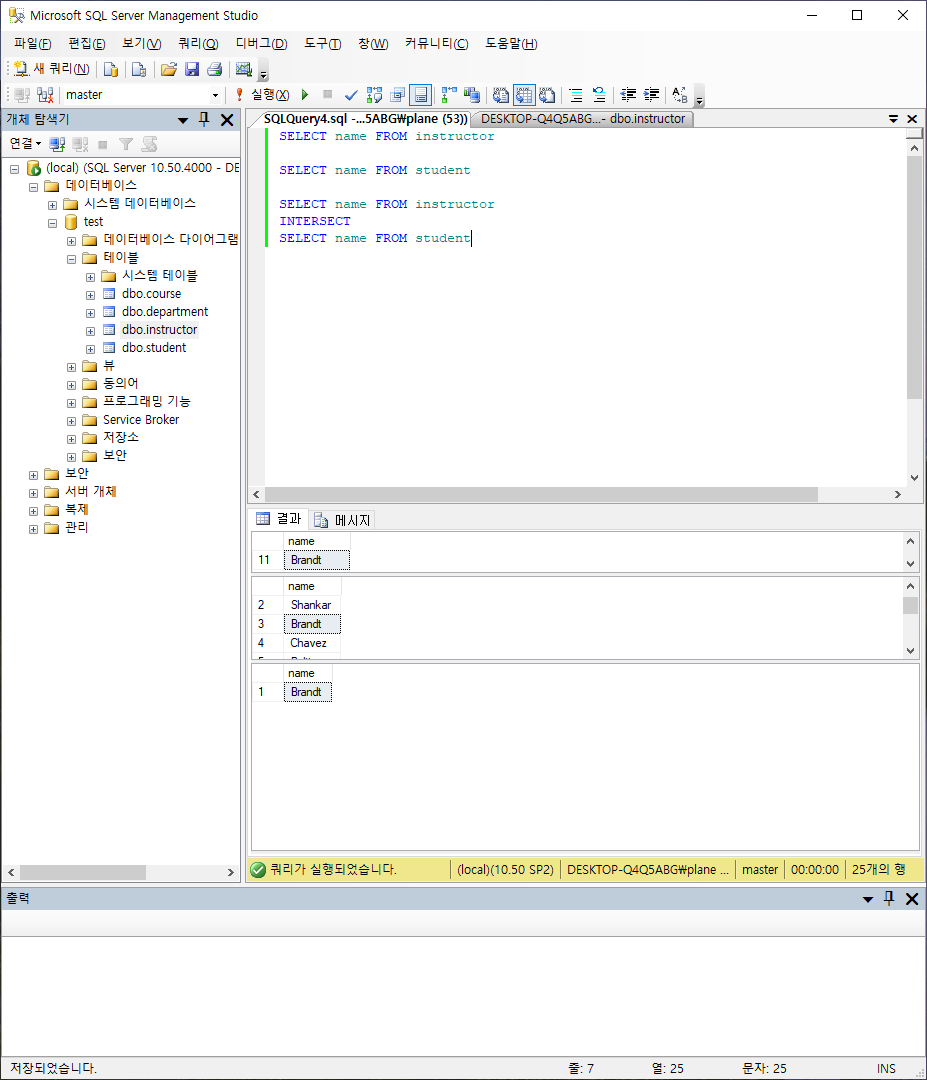


**[설명]**

UNION 명령어는 이름 그대로 합집합이며 두 데이터들의 값을 합집합을 해서 보여준다. 사진에 보이는 것처럼 두 데이터의 개수는 12개씩이지만 UNION을 쓰면 24 기본적으로 원하는 데이터의 중복을 제거하여 23개의 개수라고 나타내준다. 이때 UNION ALL로 쓰면 중복을 모두 나타내서 합집합의 결과는 12+12 = 24개라는 것을 알 수 있다.

* **INTERSECT (교집합)**

**[쿼리 결과]**

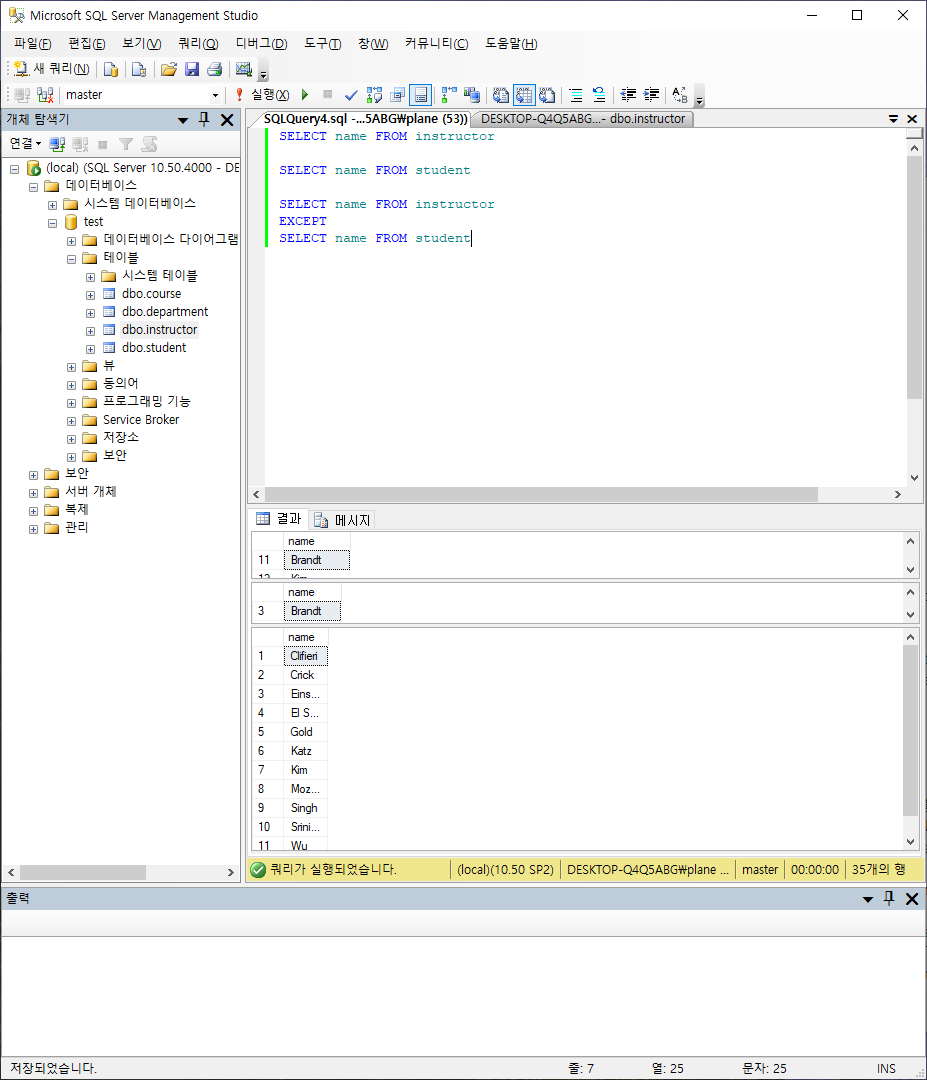


**[설명]**

INTERSECT 명령어는 이름에 맞게 데이터들의 교집합을 의미한다. 쿼리에 보이는 두 테이블을 교집합하여 name 값이 같은 것을 보여주는데, 둘다 있는 name은 ‘Brandt’ 뿐이므로 INTERSECT의 결과로도 ‘Brandt’ 하나만 나온 것을 볼 수 있다.

* **EXCEPT (차집합)**

**[쿼리 결과]**



**[설명]**

EXCEPT 명령어는 차집합을 의미하는데, 두 대상 중 첫번째로 온(쿼리상에선 instructor의 name) 데이터에서 두번째로 온(쿼리상에선 student의 name) 값을 뺀 나머지 데이터들을 보여준다.

즉,

SELCET name FROM instructor

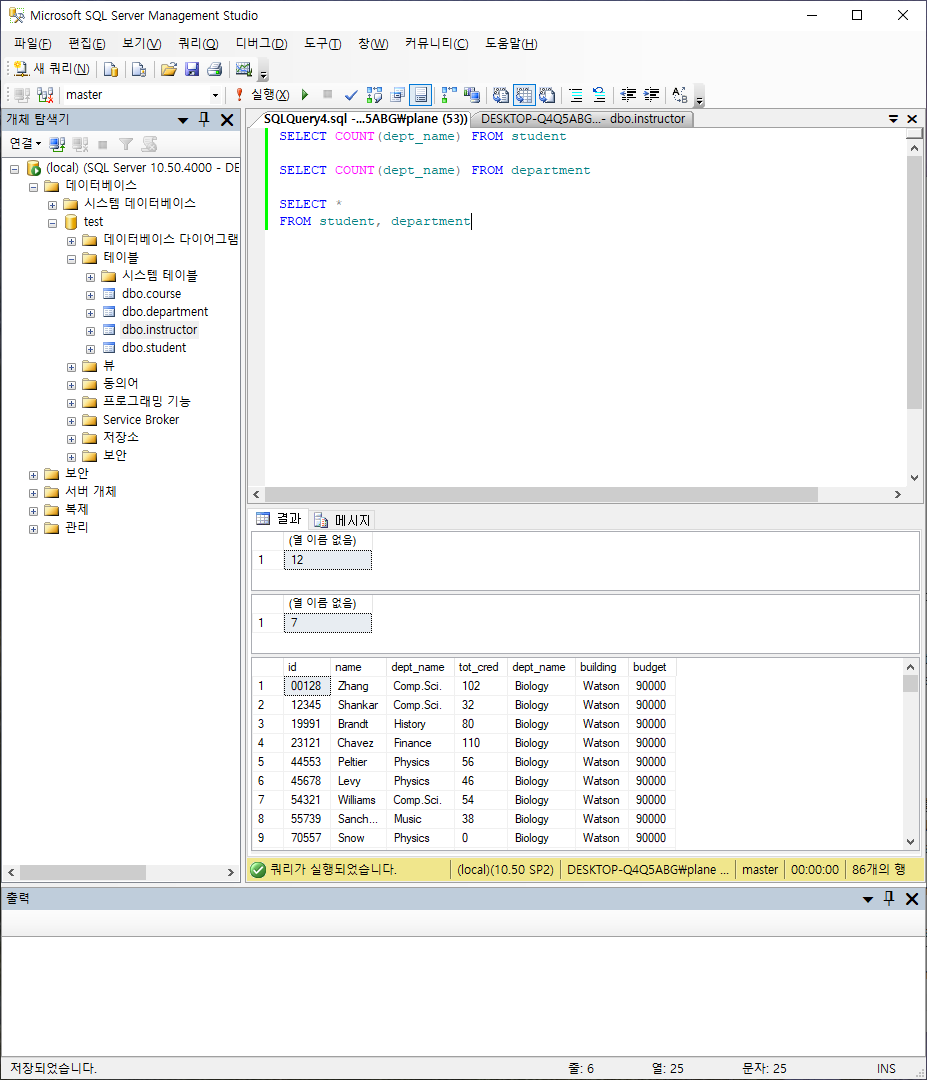
EXCEPT

SELECT name FROM student

문에서 결과는 SELECT name FROM instructor에서 student 테이블의 name 값과 겹치는 ‘Brandt’ 만 빼고 보이게 된다.

* **카티전 곱**

**[쿼리 결과]**



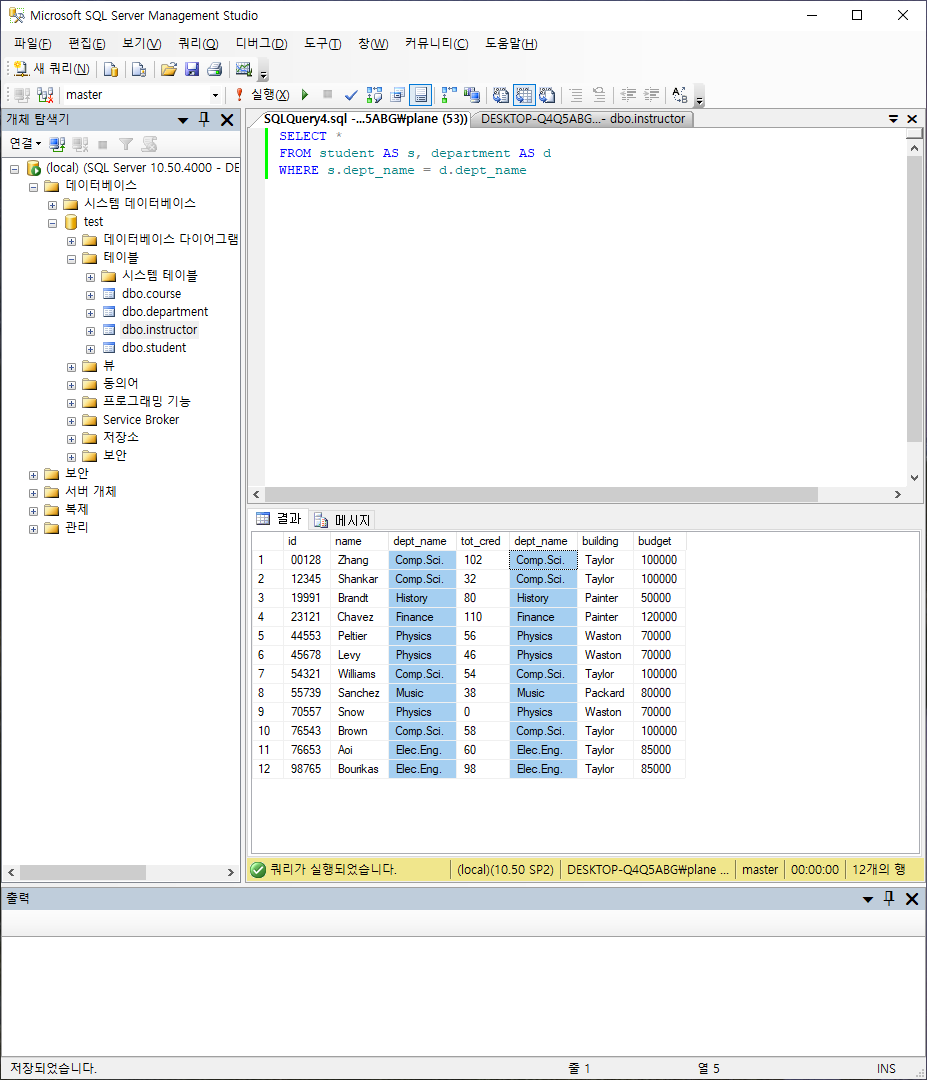
**[설명]**

카티전 곱은 FROM 절을 통해 여러 테이블에서 동시에 값을 가져올 때 발생하는데 쿼리문에서처럼 student와 department 테이블에서 값을 모두 가져오게 되면 둘의 값들이 서로 매칭되어 두 테이블의 개수를 곱한 만큼의 데이터가 생기게 된다.

즉, SELECT \* FROM student, department를 하게 되면 12(student table) \* 7(department table) = 84 해서 위의 쿼리 결과처럼 나오게 된다.

* **INNER JOIN (WHERE 절 사용)**

**[쿼리 결과]**

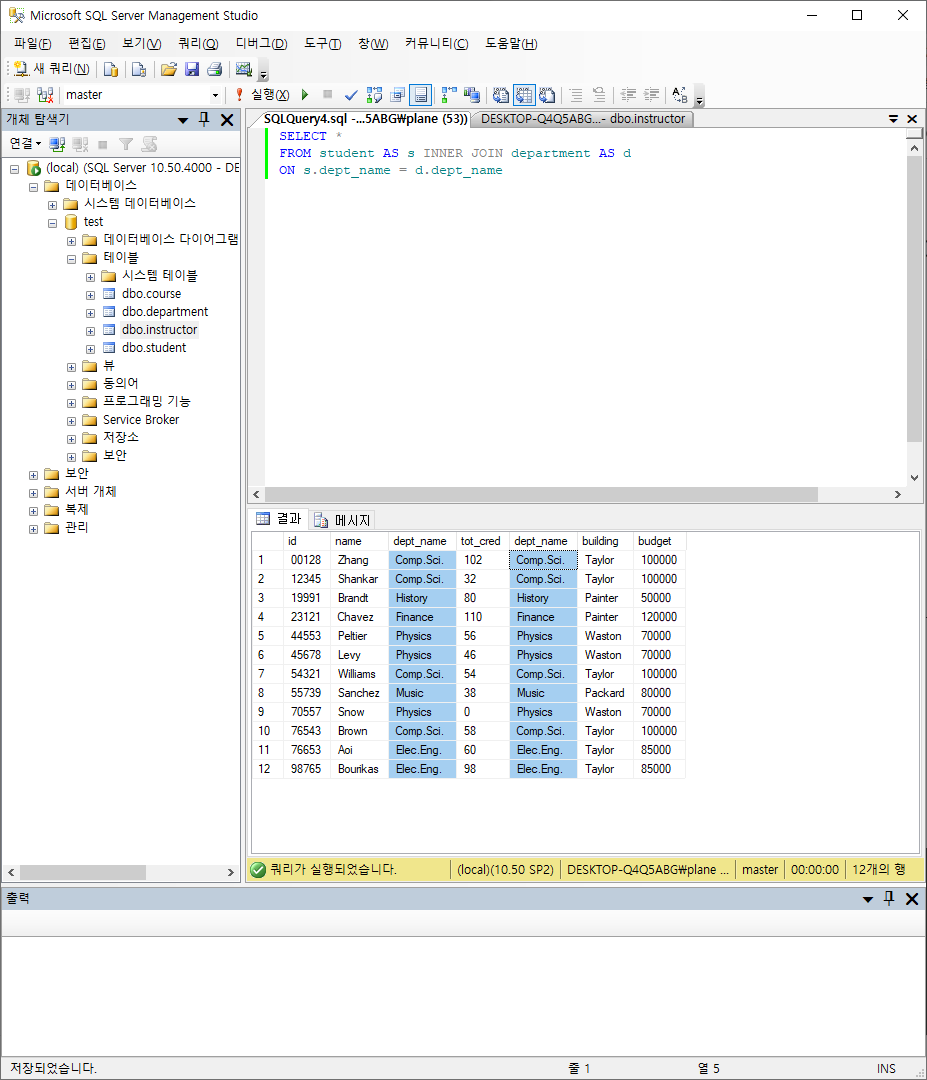


**[설명]**

INNER JOIN 은 서로 매칭되는 것만 엮어 조회할 수 있는 명령어이다. 이때 INNER JOIN 명령어를 안쓰고 WHERE 절로 할 수 있는데, INNER JOIN 이 서로 매칭되는 것만 엮어서 조회할 수 있으므로 특정 데이터가 여러 테이블에서 같은 경우만을 WHERE 절로 불러옴으로써 INNER JOIN 명령어처럼 할 수 있다.

* **INNER JOIN (INNER JOIN과 ON 사용)**

**[쿼리 결과]**

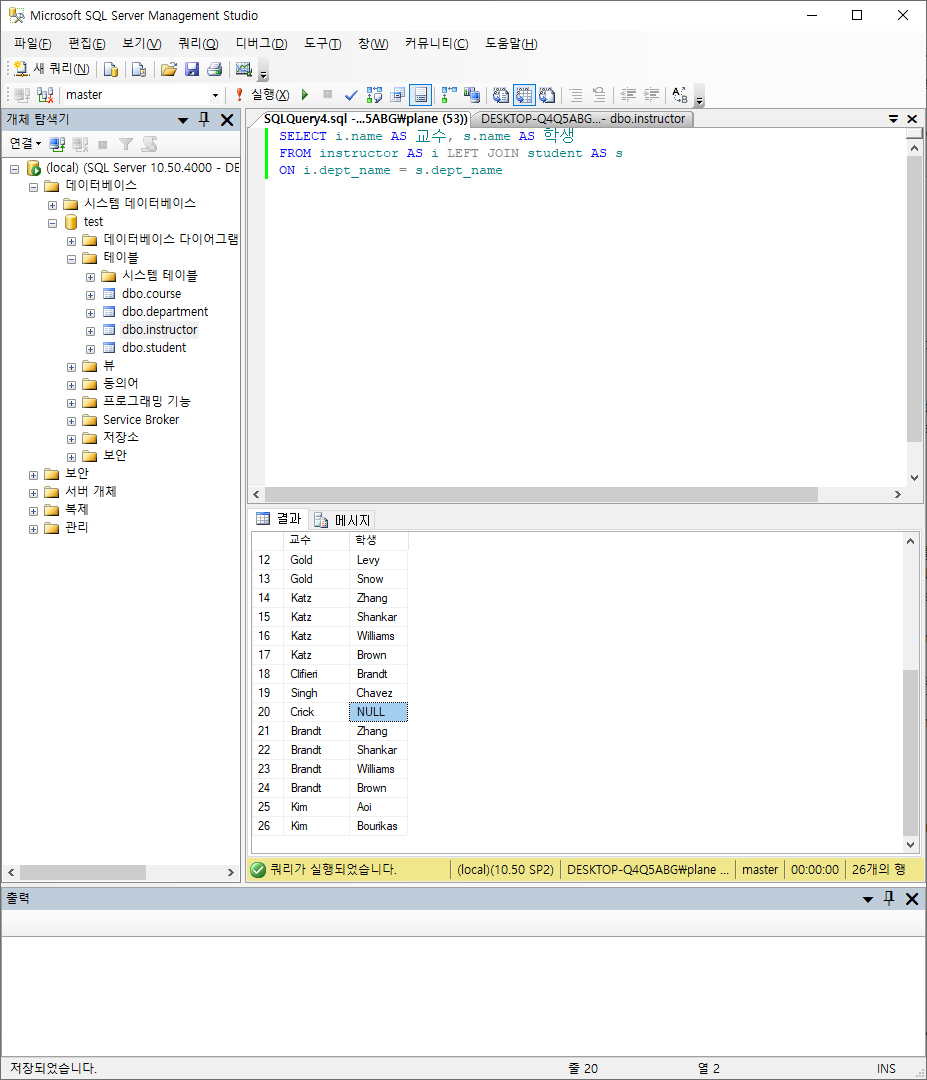


**[설명]**

INNER JOIN 명령어를 사용한다고 명시함으로써 뒤에 오는 ON 명령어에서 WHERE 절처럼 조건을 달아 두 테이블의 이름이 같을 때 INNER JOIN 한 결과를 보여준다.

* **OUTER JOIN (LEFT)**

**[쿼리 결과]**

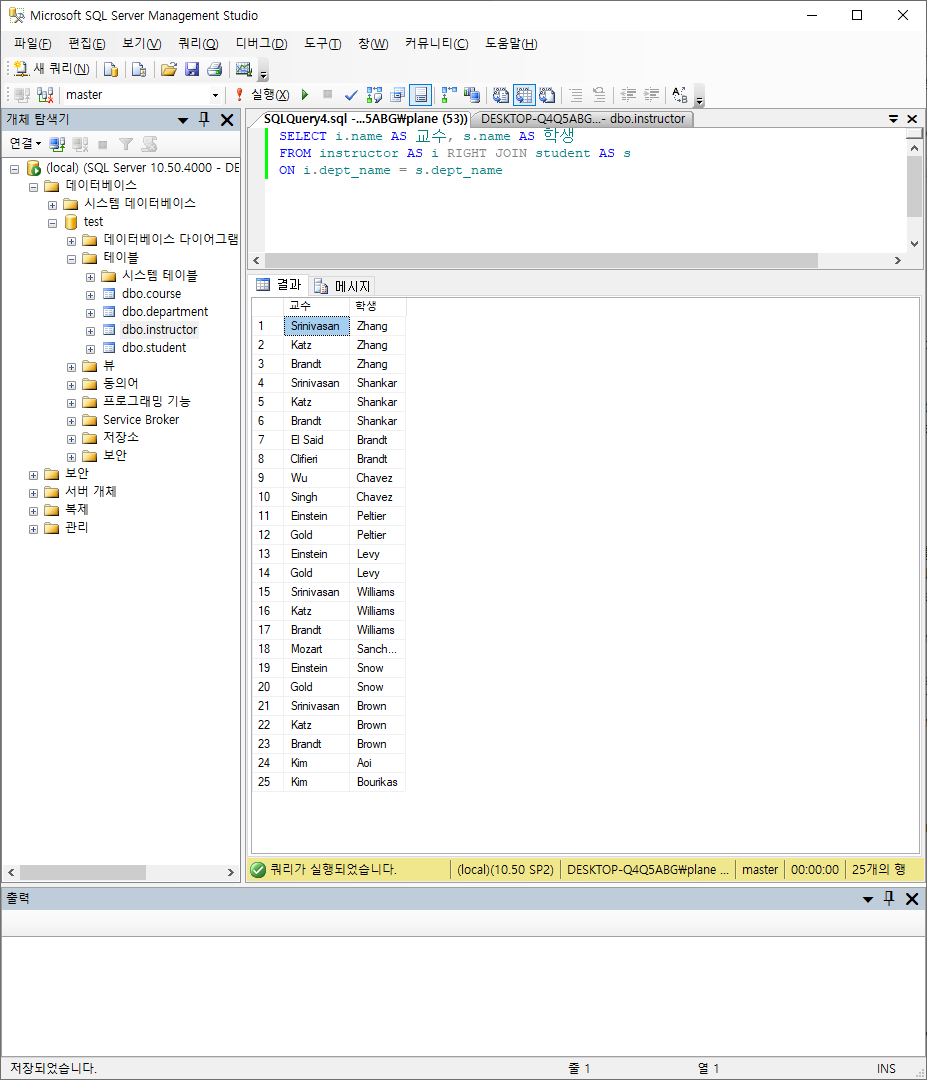


**[설명]**

OUTER JOIN 은 매칭되는 것 뿐만 아니라 매칭되지 않는 것도 함께 조회한다. 그리고 LEFT JOIN 과 RIGHT JOIN 은 조회할 테이블 방향이다. 따라서 LEFT JOIN을 쓰게 되면 현재 쿼리에서 좌측(instructor)에 맞춰서 모든 교수에 대한 데이터를 조회해서 ON 조건에 맞는 값들을 보여주는데, 이때 LEFT 즉, instructor 테이블에 초점을 둬서 두번째 테이블인 student 에서 넘어온 값이 NULL 인 경우도 생기게 된다.

* **OUTER JOIN (RIGHT)**

**[쿼리 결과]**

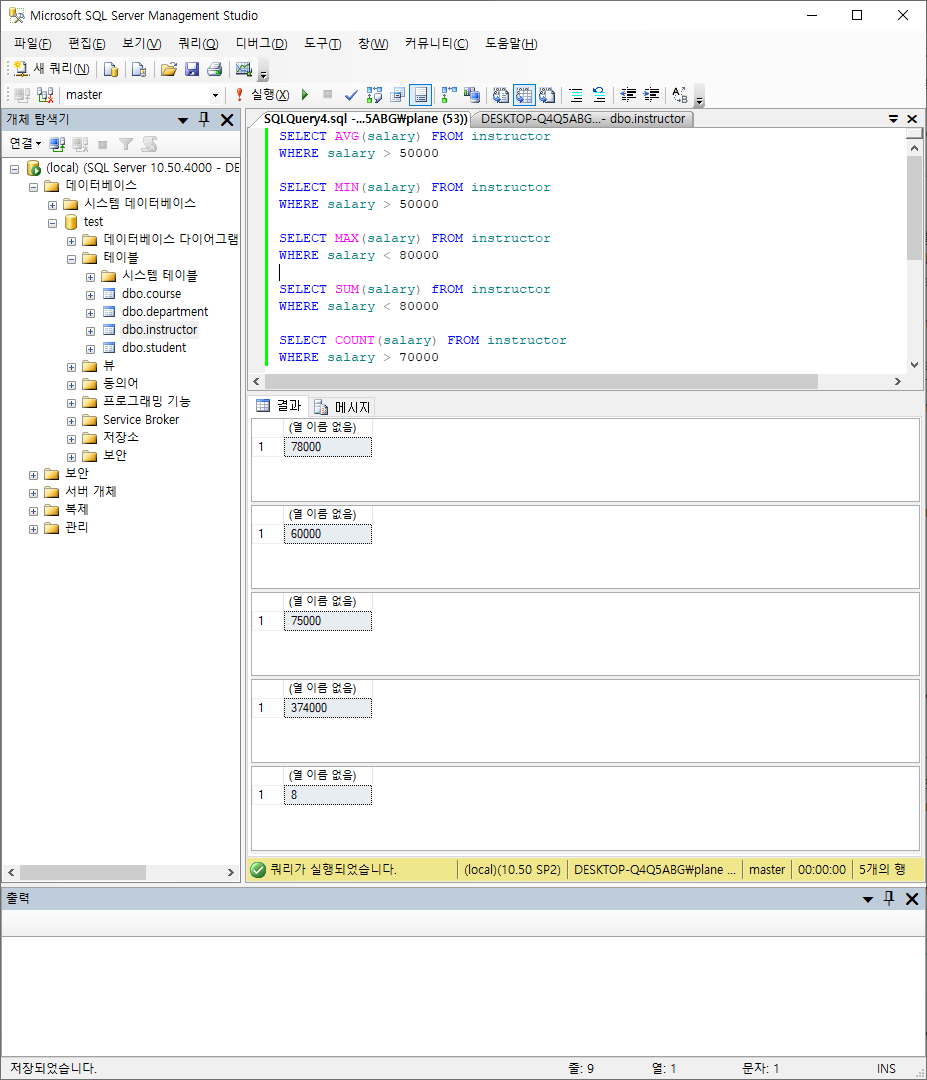


**[설명]**

이번엔 위와 반대로 RIGHT JOIN을 한 결과인데, 위에선 LEFT(instructor)에 초점을 뒀다면 이번엔 RIGHT(student)에 초점을 뒀기 때문에 아까와 같이 student 쪽에선 NULL 이 보이지 않는다. 대신 아까 student의 값이 NULL 이였던 Crick 교수의 정보는 보이지 않는다.

* **집성함수 (AVG, MIN, MAX, SUM, COUNT) – WHERE 절 추가**

**[쿼리 결과]**



**[설명]**

* **AVG**

WHERE 절에 맞는 조건에 해당하는 값들을 instructor에서 가져와서 ‘평균’을 보여주는 명령어다.

* **MIN**

WHERE 절에 맞는 조건에 해당하는 값들을 instructor에서 가져와서 그 중 ‘최솟값’을 보여주는 명령어이다.

* **MAX**

WHERE 절에 맞는 조건에 해당하는 값들을 instructor에서 가져와서 그 중 ‘최댓값’을 보여주는 명령어이다.

* **SUM**

WHERE 절에 맞는 조건에 해당하는 값들을 instructor에서 가져와서 모두 합한 ‘총합’을 보여주는 명령어이다.

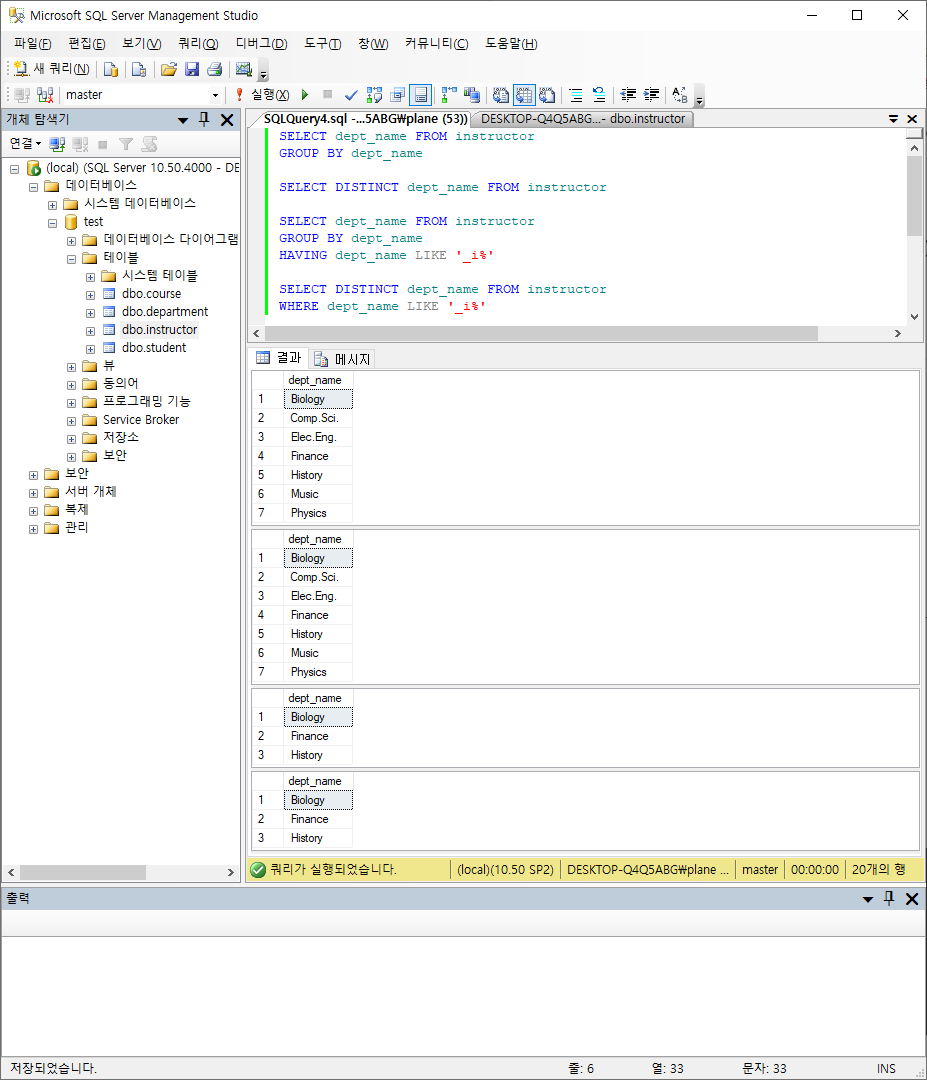
* **COUNT**

WHERE 절에 맞는 조건에 해당하는 값들을 instructor에서 가져온 모든 결과의 ‘총 개수’를 보여주는 명령어이다.

또한, 집성함수로 나온 값들의 열은 모두 이름이 없으므로 집성함수 뒤에 AS (이름) 을 추가로 입력하여 보여질 결과에서 집성함수의 값들의 열 이름을 정해줄 수 있다.

* **GROUP BY, HAVING 절**

**[쿼리 결과]**



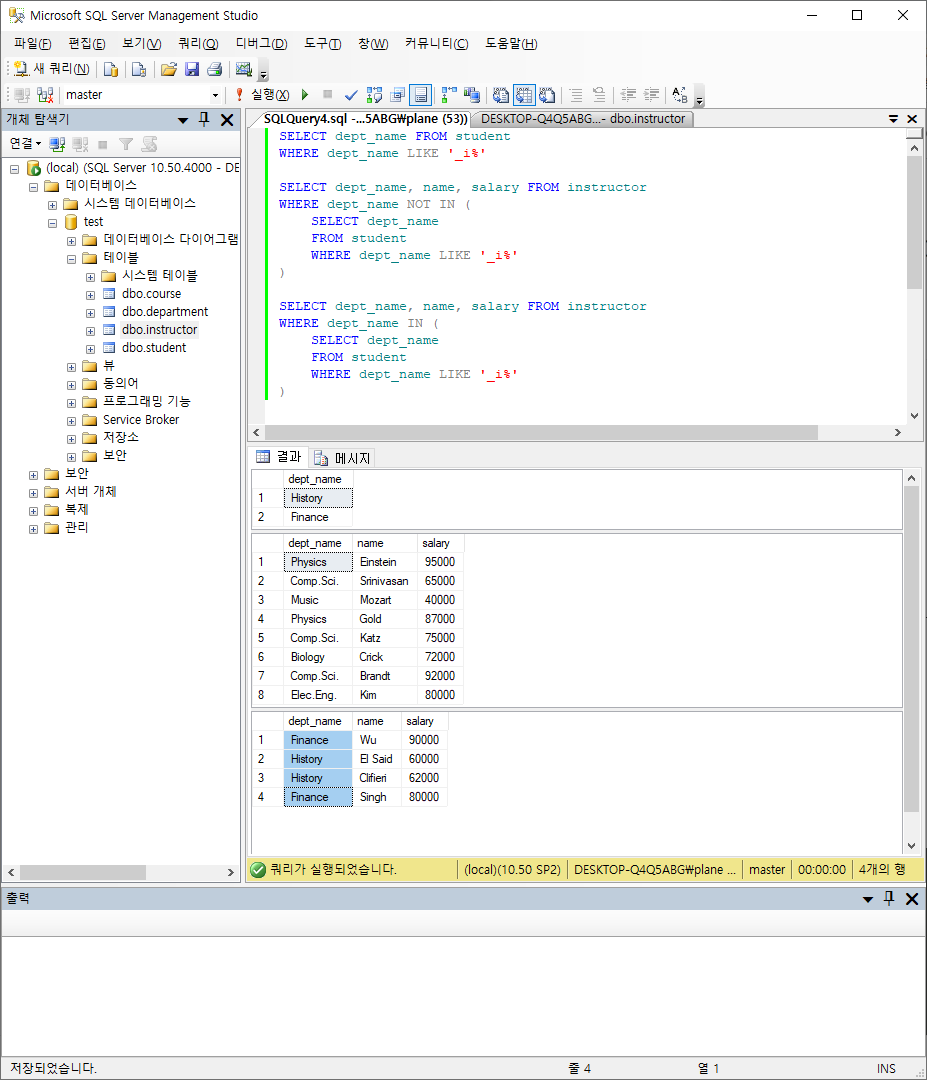
**[설명]**

데이터의 중복을 없애서 결과를 보여주는 방법이 두가지 있는데 그 중 하나가 GROUP BY이다. GROUP BY (원하는 필드)를 하게 되면 특정 필드를 중심으로 원하는 데이터 값들을 보여주는데, 이때 중복되는 값들을 자동으로 안보여주기 때문이다.

이와 비슷하게 GROUP BY를 쓰지 않고도 똑같이 보여주는 방법은 SELECT 절에서 DISTINCT를 추가로 붙여줌으로써 중복을 직접 없앨 수 있다. 또한 SELECT DISTINCT 절을 사용하면 WHERE 절을 이용해 조건을 달 수 있다면, GROUP BY 명령어는 추가로 HAVING 명령어를 이용해 조건을 붙여 나타낼 수 있다.

* **IN & NOT IN**

**[쿼리 결과]**



**[설명]**

IN & NOT IN은 WHERE 절에서 추가로 더 세부적인 조건을 달 수 있는데 조건에 해당하는 데이터가 더 세부적인 범위 내에서 다루어져야 할 때 쓰인다.

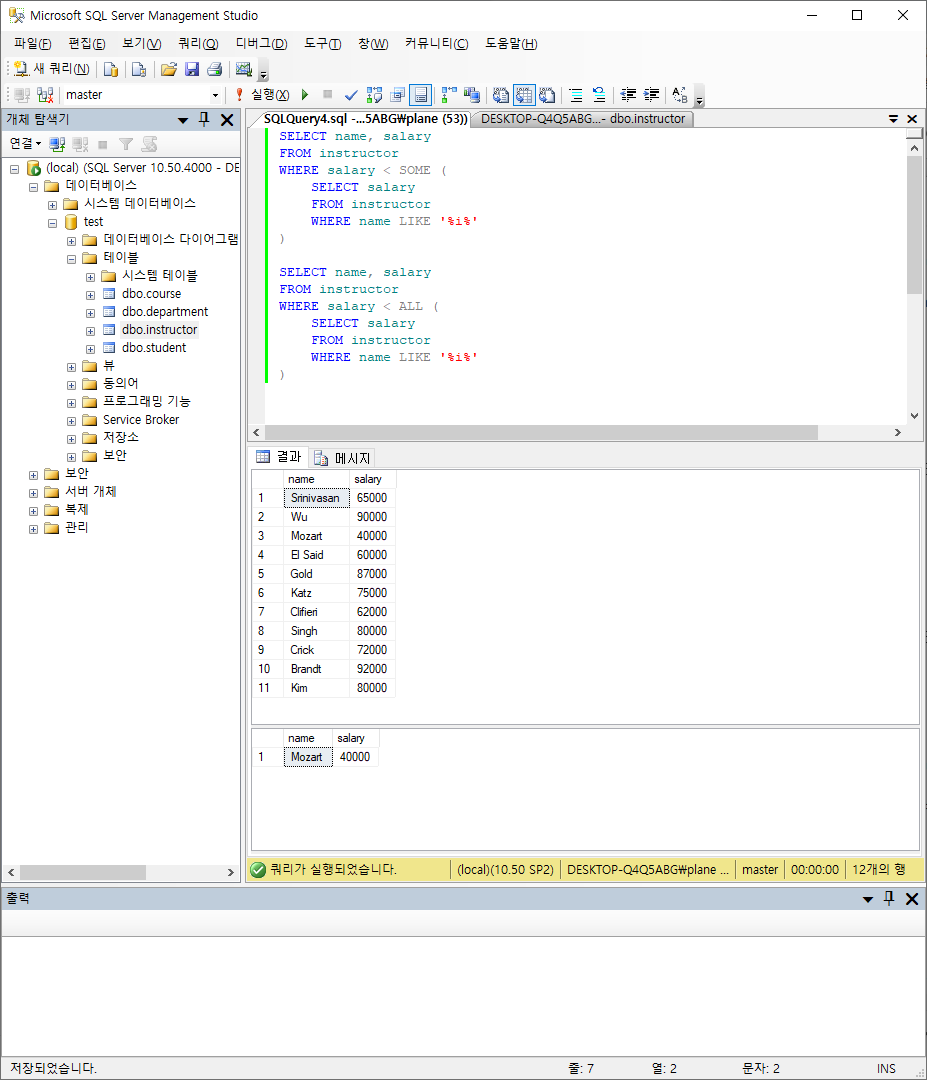
예를 들어 위에 쿼리에서

SELECT dept\_name FROM student WHERE dept\_name LIKE ‘\_i%’을 한 결과를 봤을 때, dept\_name 필드의 값이 History와 Finance만 나오게 된다.

그리고 밑에서 IN & NOT IN을 추가해서 instructor 테이블로부터 dept\_name, name, salary를 가져올 때 WHERE 절에서 IN & NOT IN으로 dept\_name이 History거나 Finance인 경우를 포함하지 않는 결과(두번째 결과 표)와 포함하는 결과(세번째 결과)처럼 나오게 된다.

* **SOME & ALL**

**[쿼리 결과]**



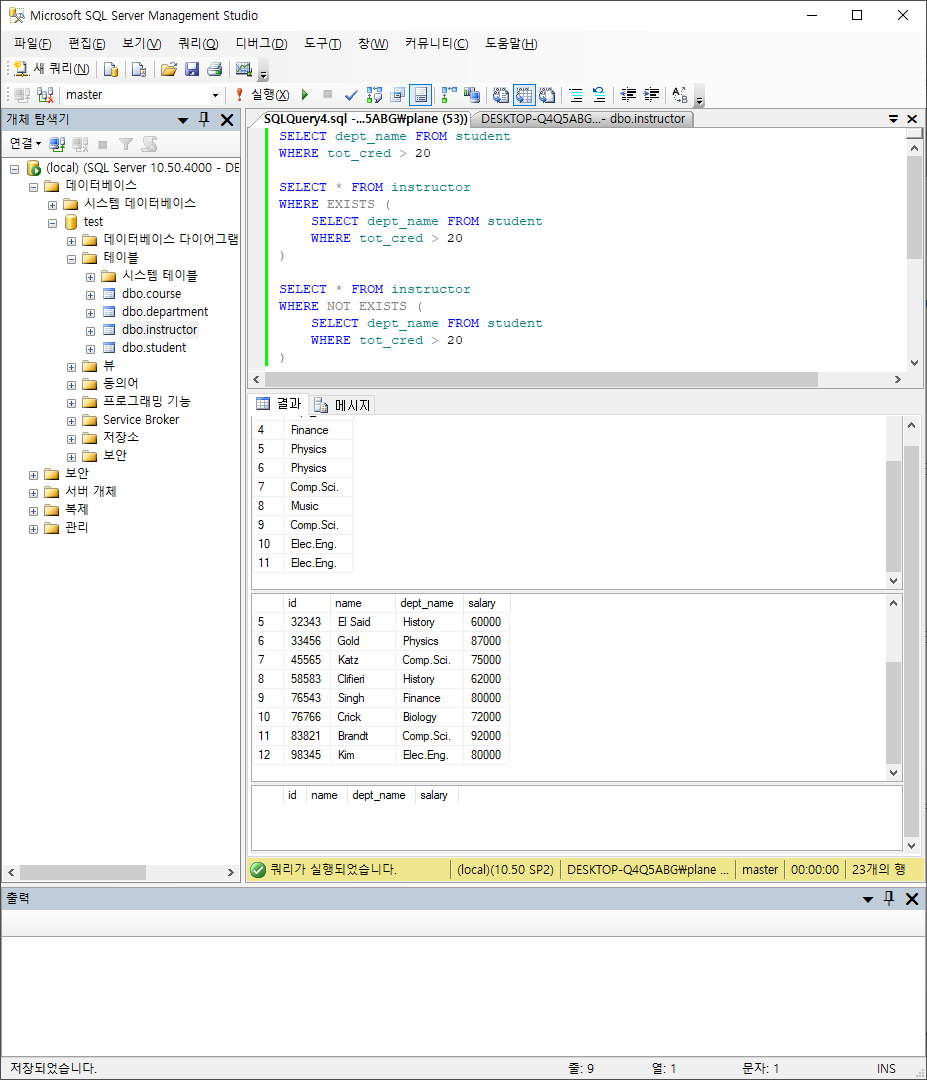
**[설명]**

위의 IN & NOT IN과 비슷하게 조건에 추가로 세부적인 조건을 달 수 있는데, 쿼리에서 salary < SOME ( )과 salary < ALL ( )의 차이를 보면 SOME의 경우 ( ) 부분에서 특정되는 salary 값들 중 어느 하나라도 WHERE 절의 조건을 만족하면 ‘참’이다. 또 ALL은 ( ) 부분에서 특정되는 salary 값들 모두다 WHERE 절의 조건을 만족해야만 값을 보여주는 결과를 알 수 있다.

위의 IN & NOT IN과 SOME & ALL의 경우 SELECT FROM WHERE 문에서 WHERE 절에 쓰이는 비교 대상은 같아야 한다(salary < SOME ( ) & salary < ALL( )를 했으므로 SOME과 ALL의 ( ) 안에서는 SELECT salary를 해야한다).

* **EXISTS & NOT EXISTS**

**[쿼리 결과]**



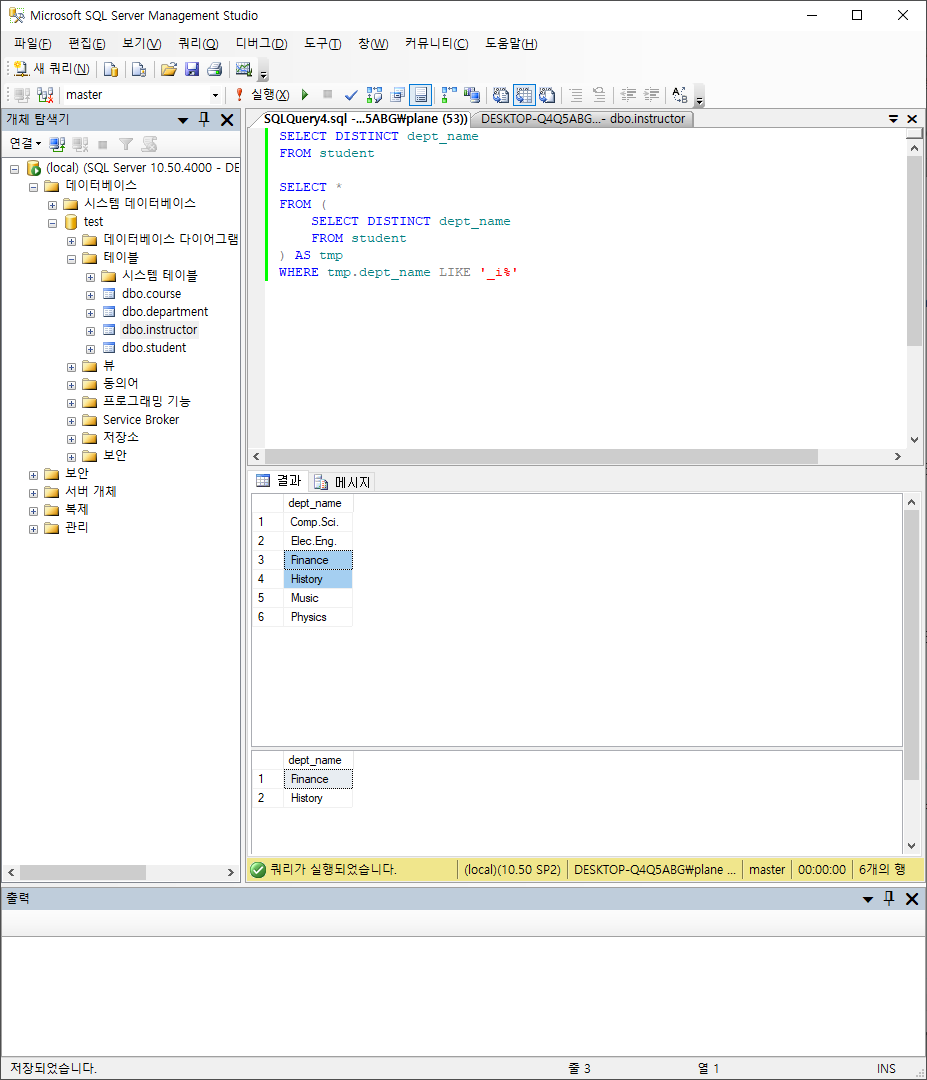
**[설명]**

EXISTS & NOT EXISTS 명령어는 조건 절에서 EXISTS ( )와 NOT EXISTS ( )로 쓰이는데, ( ) 안에 조건에 따라 ‘참(EXISTS) & 거짓(NOT EXISTS)인 데이터들을 모아주는 역할을 한다.

그래서 위의 쿼리에서 SELECT dept\_name FROM student WHERE tot\_cred > 20인 데이터들이 있을 때, WHERE EXISTS ( )를 통해 instructor table로부터 값을 가져왔을 때 위 SELECT FROM WHERE 문에 맞는 데이터들이 instructor table에 존재하므로 모든 값을 가져오는 결과를 보여주었고, 그 반대로 WHERE NOT EXISTS ( )에선 ( )에 해당하는 데이터들이 없는 것들을 가져오므로 instructor에서 어느 것도 값을 가져오지 않는 결과를 보여준다.

* **FROM 절의 부질의**

**[쿼리 결과]**



**[설명]**

FROM 절의 부질의는 위에 있던 IN & NOT IN, SOME & ALL, 그리고 EXISTS & NOT EXISTS에서 WHERE 절에 ( ) 를 통해 추가로 세부적인 정보를 다뤘던 것처럼 FROM 절에서도 할 수 있다는 것을 의미한다.

위의 쿼리에서처럼 SELECT DISTINCT dept\_name FROM student 문으로 student 테이블에서 중복되지 않는 dept\_name을 가져올 때의 결과들을 FROM ( )에 씀으로써 데이터 범위를 줄이고 그 줄인 범위를 AS tmp로 이름을 바꿔서 WHERE 조건에 맞는 값들을 모두(\*) 가져오는 결과들을 보여준다.

**과제 2**

1. **Section 테이블에서 모든 과목의 코드(c\_id) 중복 없이 검색**

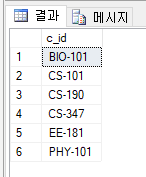
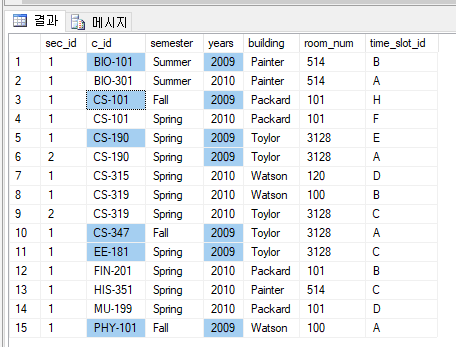
**[쿼리]**

SELECT DISTINCT c\_id

FROM section

WHERE years = '2009'

**[쿼리 결과]**



1. **Student & takes 테이블에서 ‘Zhang’ 학생이 들은 과목의 코드(c\_id)와 성적(grade) 검색**

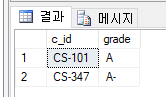
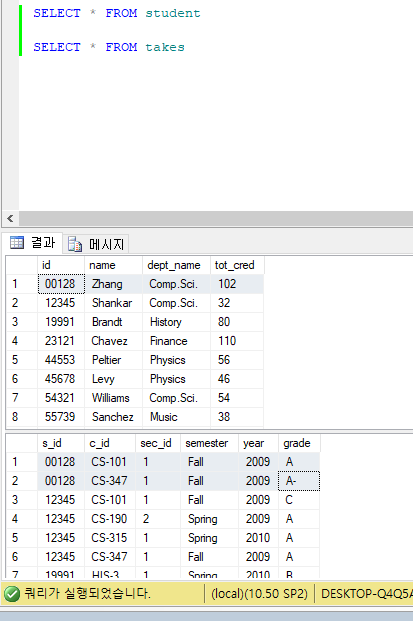
**[쿼리]**

SELECT c\_id, grade

FROM takes, student

WHERE s\_id = id AND name = 'Zhang'

**[쿼리 결과]**



1. **Instructor & teaches 테이블에서 ‘Comp.Sci.’ 학과 소속의 강사의 수업 중 2009년에 수업한 과목의 코드(c\_id)를 검색**

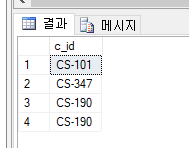
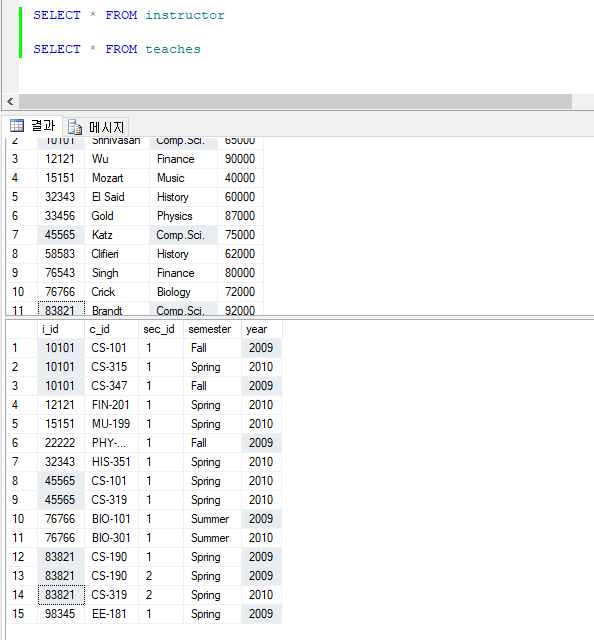
**[쿼리]**

SELECT c\_id

FROM instructor, teaches

WHERE dept\_name = 'Comp.Sci.' AND id = i\_id AND year = '2009'

**[쿼리 결과]**



1. **Takes & course 테이블에서 각 학생이 들은 과목에 대해 학생(s\_id)별로 취득한 총 학점수(누적된 학점수)를 구하기**

**[쿼리]**

SELECT s\_id, SUM(credits) AS 모든학점

FROM (

SELECT s\_id, credits

FROM takes, course

WHERE c\_id = id

) AS temp

GROUP BY temp.s\_id

**[쿼리 결과]**

