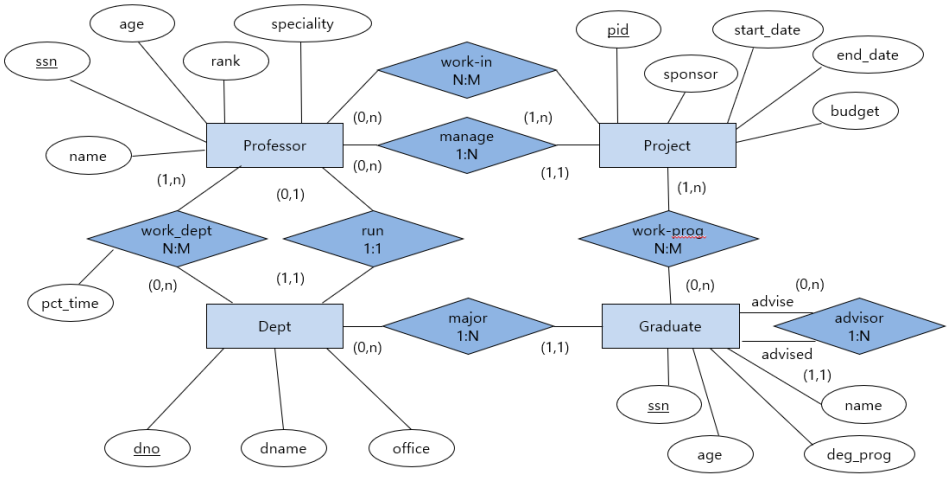
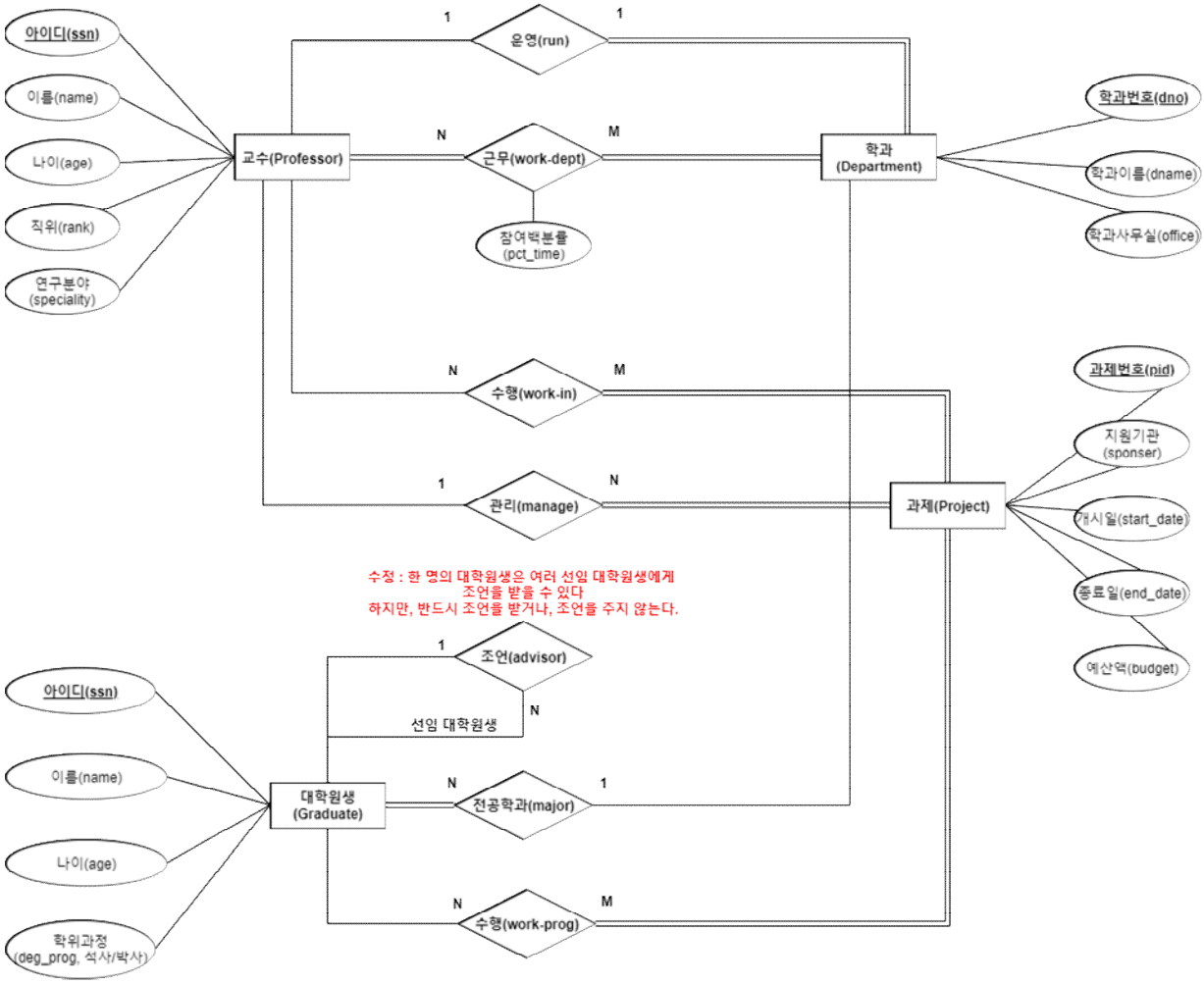




개념적 설계 단계에서 최종 완성된 위의 E-R 다이어그램에 따라, 논리적 설계를 수행한다. 위의 E-R 다이어그램은 관계에 필수 참여인지 선택 참여인지 나오지 않았으므로, 이전에 만들어둔 E-R 다이어그램을 같이 사용하여 관계를 분석한다.

1. 모든 개체를 릴레이션으로 변환

우선, 모든 개체를 릴레이션으로 변환해야 한다. Professor, Project, Dept, Graduate에 대하여 각 개체의 속성을 포함하여 릴레이션으로 변환한다. 여기서, 각 객체가 기지고 있는 속성이 분해 가능 한 복합 속성인 경우 복합 속성을 구성하고 있는 단순 속성만 릴레이션의 속성으로 변환한다.

ssn	name	age	rank	speciality
1	PF_Kim	30	조교수	운영체제
2	PF_Jin	40	부교수	임베디드
3	PF_Park	50	정교수	네트워크
4	PF_Lee	60	명예교수	그래픽스
5	PF_Roh	70	명예교수	데이터베이스

표 1) Professor 릴레이션

pid	sponser	start_date	end_date	budget
1	교육부	2020-01-01	2020-01-30	1000000
2	국방부	2020-02-01	2020-02-29	2000000
3	국세청	2020-03-01	2020-03-30	3000000
4	고용노동부	2020-04-01	2020-04-30	4000000
5	기상청	2020-05-01	2020-05-30	5000000

표 2) Project 릴레이션

dno	dname	office
1	컴퓨터공학과	100호
2	기계과	101호
3	전자과	102호
4	정보통신학과	103호
5	정보시스템과	104호

표 3) Dept 릴레이션

ssn	name	age	deg_prog
1	GD_Kim	26	석사
2	GD_Jin	27	석사
3	GD_Park	28	석사
4	GD_Lee	29	박사
5	GD_Roh	30	박사

표 4) Graduate 릴레이션

2. Professor, Project의 관계

그 다음, 각 개체 간의 관계에 대하여 릴레이션으로 변환하거나, 외래키를 추가하는 방식을 사용한다.

먼저 교수와 과제 간의 관계를 보면,

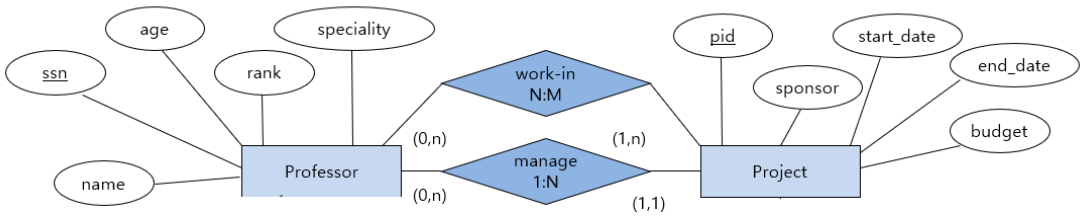


그림 3) 교수와 과제의 관계 1

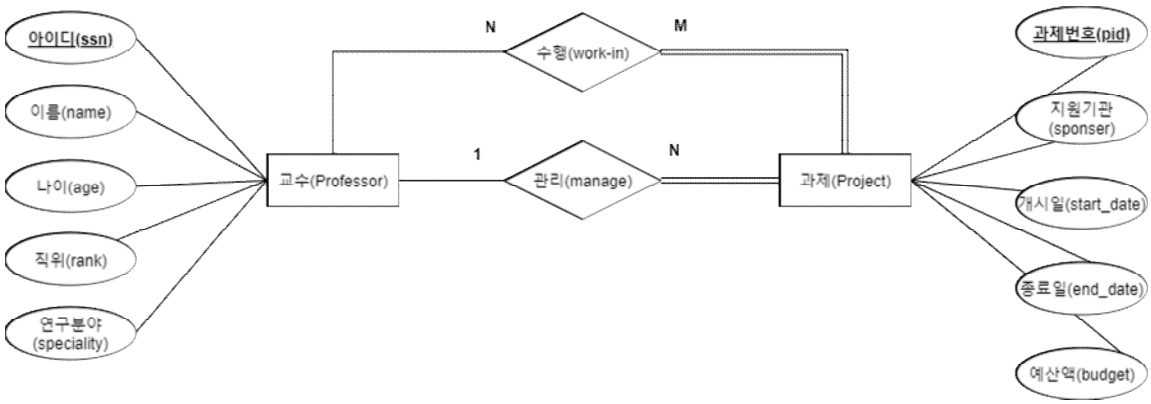


그림 4) 교수와 과제의 관계 2

수행(work_in)의 관계에 대하여 교수와 과제는 다대다(N:M)의 관계이다. 다대다 관계는 해당 관계를 릴레이션으로 변환해야 한다. 따라서, work_in이라는 릴레이션이 만들어지며, work_in 릴레이션의 기본키는 해당 관계를 맺고 있는 각 릴레이션들의 기본키를 참조하는 외래키들로 기본키를 구성하거나, 따로 기본키를 만든 후 각 릴레이션들의 기본키를 참조하는 외래키를 추가한다. 여기서는, 수행번호(win_id)라는 기본키를 만든 후 각 릴레이션의 기본키를 참조하는 외래키를 추가하였다.

win_id (수행번호,기본키)	ssn (수행교수, 외래키)	pid (수행과제, 외래키)

표 5) 수행(work_in) 릴레이션 스키마

관리(manage)의 관계에 대하여 교수와 과제는 일대다(1:N)의 관계이다. 일대다 관계의 경우 일반적으로 릴레이션으로 변환하지 않고 외래키를 추가한다. 약한 개체가 참여하는 일대다

관계는 릴레이션으로 변환하지 않고 약한 개체 측에 강한 개체의 기본 키를 참조하는 외래키를 추가하고, 해당 외래키와 약한 개체가 가지고 있던 키를 포함하여 약한 개체의 기본 키로 지정한다.

여기서 생각해야 할 것이 과제 개체가 상위 개체에 종속적인 약한 개체 타입인가? 즉, 교수 객체 없이 독자적으로 존재 할 수 없는 개체인가? 수행 관계의 관점에서 교수는 여러 과제를 수행 할 수 있고, 과제는 교수에 의해 수행되어야만 한다. 하지만, 비행기와 비행기 좌석의 존재라는 관계를 예로 들어 비행기가 없다면 비행기 좌석은 존재 할 수 없는 것처럼, 과제는 교수에 종속적인 관계는 아니다. 교수 개체가 과제 개체의 존재 여부를 결정하지는 않는다. 단지, 수행이라는 관계의 관점에서 과제는 수행에 필수적으로 참여하고, 교수는 수행에 선택적으로 참여한다.

따라서, 일반적인 일대다 관계이므로 외래키로 표현한다. 일대다의 관계에서 N 측의 릴레이션에 1 측의 릴레이션의 기본 키를 참조하는 외래키를 추가하여야 한다.

교수 한 명이 여러 과제를 관리 할 수 있지만, 하나의 과제는 한 명의 교수에 의해서만 관리된다. 만약, 교수 릴레이션에 과제 릴레이션의 기본 키를 참조하는 외래키를 추가하게 된다면, 관계 데이터 모델의 다중 값 속성을 허용하지 않는 규칙을 위반하게 되므로, 반드시 과제 릴레이션에 교수 릴레이션의 기본키를 참조하는 외래키(연구책임자)를 추가하여야 한다.

이에 따라, 과제 릴레이션을 다음과 같이 수정한다.

pid	sponser	start_date	end_date	budget	ssn (연구책임자. 외래키)
1	교육부	2020-01-01	2020-01-30	1000000	1
2	국방부	2020-02-01	2020-02-29	2000000	1
3	국세청	2020-03-01	2020-03-30	3000000	2
4	고용노동부	2020-04-01	2020-04-30	4000000	3
5	기상청	2020-05-01	2020-05-30	5000000	4

표 6) 수정 된 Project 릴레이션

2. Professor, Dept의 관계

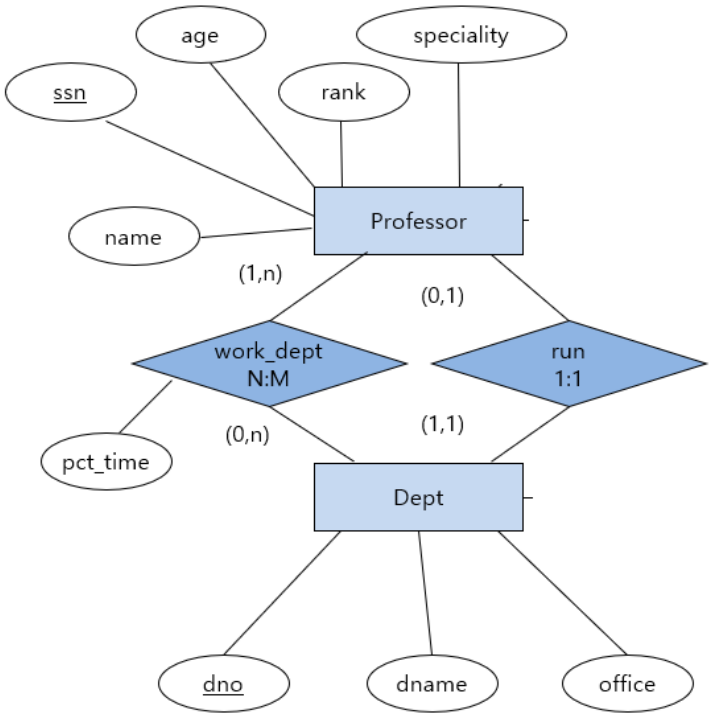


그림 5) 교수와 학과의 관계 1

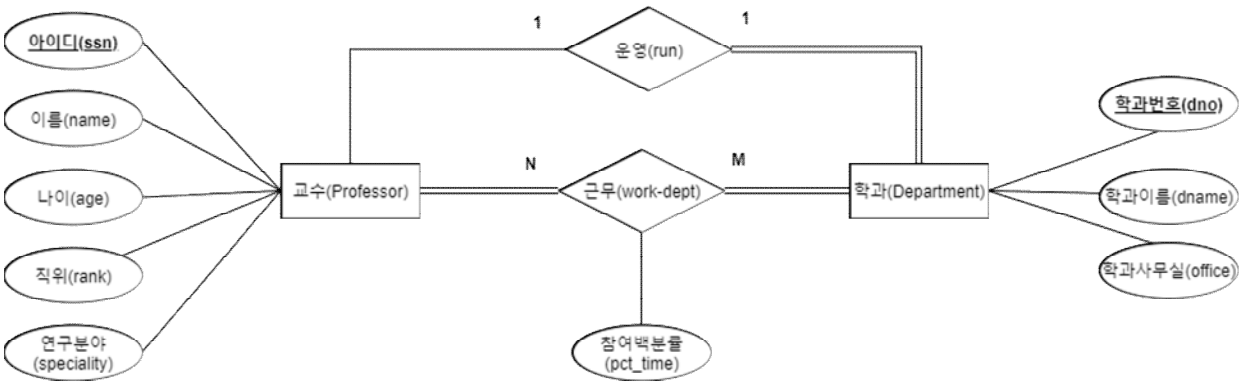


그림 6) 교수와 학과의 관계 2

그 다음, 교수와 학과의 관계를 보면, 근무(work_dept)의 관계에서 다대다(N:M)의 관계이다. 다대다 관계는 릴레이션으로 변환해야 하므로, work_dept라는 릴레이션이 만들어지며, work_dept 릴레이션의 기본키는 해당 관계를 맺고 있는 각 릴레이션들의 기본키를 참조하는 외래키들로 기본키를 구성하거나, 따로 기본키를 만든 후 각 릴레이션들의 기본키를 참조하는 외래키를 추가한다. 여기서는, 근무번호(wdpt_id)라는 기본키를 만든 후 각 릴레이션의 기본키를 참조하는 외래키를 추가하였다.

근무 관계의 속성인 참여백분율(pct_time)은 근무 릴레이션의 속성으로 그대로 변환한다.

wdpt_id (근무번호, 기본키)	ssn (근무교수,외래키)	dno (근무학과.외래키)	pct_time (참여백분율)

표 7) 근무(work_dept) 릴레이션 스키마

운영(run)의 관계에서 교수와 학과는 일대일(1:1)의 관계이다. 일반적인 일대일 관계는 릴레이션으로 변환하지 않고, 외래키로만 표현한다. 데이터의 중복을 피하려면 개체가 관계에 참여하는 특성에 따라 차이가 있다.

1. 일반적인 일대일 관계는 외래키를 서로 주고받는다. (불필요한 데이터 중복 발생 가능)
2. 일대일 관계에 필수적으로 참여하는 개체의 릴레이션만 외래키를 받는다.
3. 모든 개체가 일대일 관계에 필수적으로 참여하면 하나의 릴레이션으로 표현한다.

위의 세부규칙에 따라, 교수는 운영에 선택 참여, 학과는 운영에 필수 참여하므로 관계에 필수 참여하는 학과 릴레이션 측에 교수 릴레이션의 기본 키를 참조하는 외래키를 추가한다. 이에 따라, 수정된 학과 릴레이션은 다음과 같다.

dno	dname	office	ssn (학과장, 외래키)
1	컴퓨터공학과	100호	1
2	기계과	101호	2
3	전자과	102호	3
4	정보통신학과	103호	4
5	정보시스템과	104호	5

표 8) 수정 된 Dept 릴레이션

3. Graduate, Project의 관계

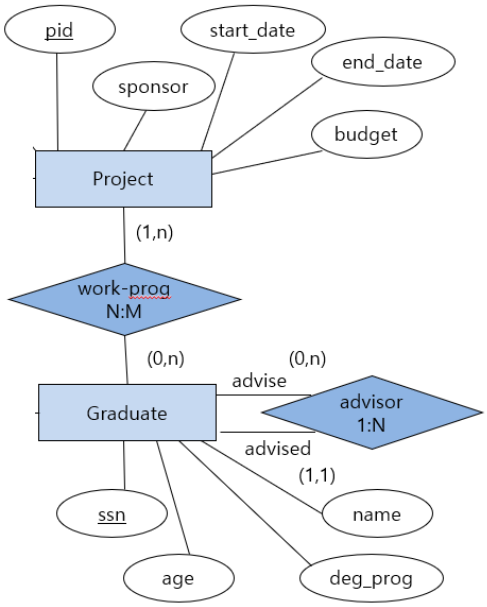


그림 7) 대학원생과 과제의 관계 1

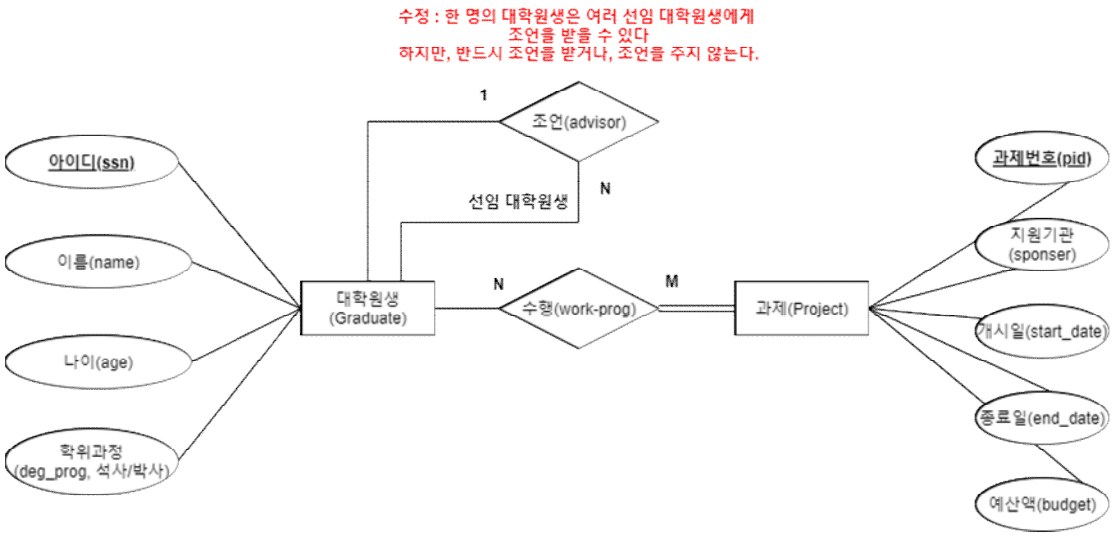


그림 8) 대학원생과 과제의 관계 2

먼저, 대학원생과 과제의 수행(work_prog)관계를 보면 다대다(N:M) 관계이다. 다대다 관계는 해당 관계를 릴레이션으로 변환해야 한다. 따라서, work_prog라는 릴레이션이 만들어지며, work_prog 릴레이션의 기본키는 해당 관계를 맺고 있는 각 릴레이션들의 기본키를 참조하는 외래키들로 기본 키를 구성하거나, 따로 기본 키를 만든 후 각 릴레이션들의 기본 키를 참조하는 외래키를 추가한다. 여기서는, 수행번호(wpr_id)라는 기본 키를 만든 후 각 릴레이

션의 기본키를 참조하는 외래키를 추가하였다.

wpr_id (수행번호, 기본키)	ssn (수행 대학원생, 외래키)	pid (수행과제, 외래키)

표 9) 수행(work_prog) 릴레이션 스키마

그 다음, 대학원생과 선임 대학원생 사이에 조언(advisor)이라는 관계가 존재한다. 조언 관계는 순환 관계로서 대학원생 릴레이션 내에서 한 대학원생 개체가 자기 자신을 제외한 다른 대학원생 개체를 참조함을 의미한다. 또한, 조언 관계는 일대다 관계로서 릴레이션으로 변환하지 않고 외래 키로 표현한다. 따라서, 대학원생 개체가 다른 대학원생 개체를 참조 할 수 있도록 대학원생 릴레이션의 기본 키를 참조하는 외래 키를 추가한다.

여기서 주의해야 할 것이, 관계 데이터 모델에서 다중 값 속성을 허용하지 않는다는 규칙이 있다. 만약, 대학원생 릴레이션에 선임 대학원생을 참조하는 외래키를 추가하게 된다면, 한 명의 대학원생은 여러 선임 대학원생에게 조언을 받을 수 있으므로 다중 값 규칙을 위반하게 된다. 따라서, 선임 대학원생을 참조(멘토)하지 않고, 선임 대학원생 측에서 조언을 주는 대학원생(멘티)을 참조하는 외래키로 추가해야 한다.

이에 따라, 수정된 대학원생 릴레이션은 다음과 같다.

ssn	name	age	deg_prog	ssn (멘티, 외래키)
1	GD_Kim	26	석사	NULL
2	GD_Jin	27	석사	1
3	GD_Park	28	석사	NULL
4	GD_Lee	29	박사	3
5	GD_Roh	30	박사	NULL

표 10) 수정 된 Graduate 릴레이션

4. Graduate, Dept의 관계

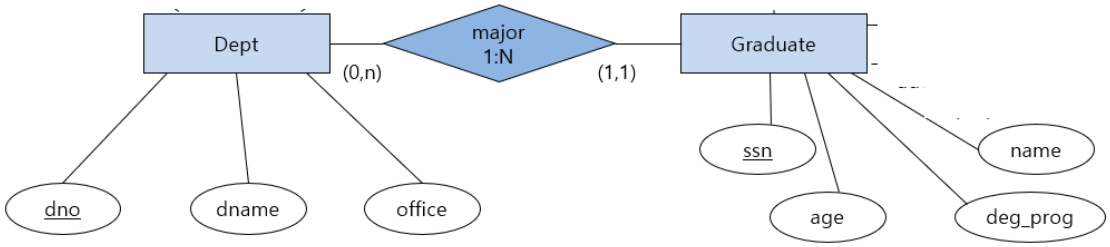


그림 9) 대학원생과 학과의 관계 1

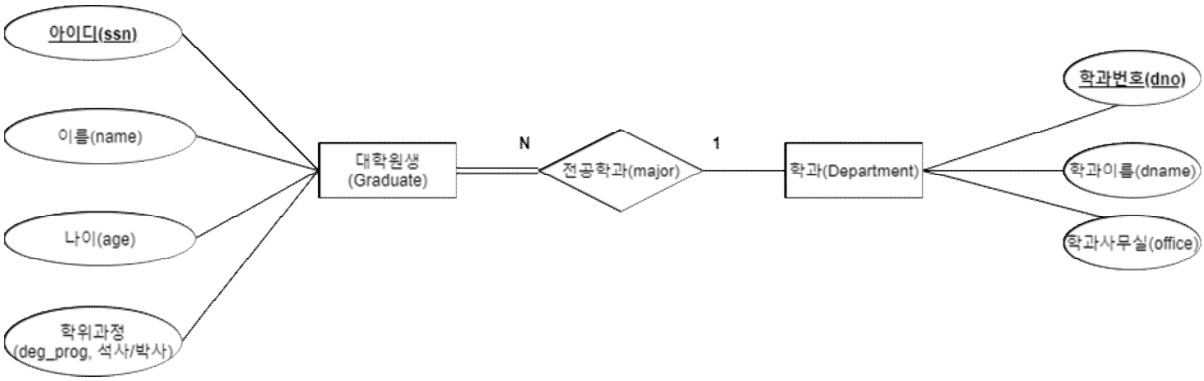


그림 10) 대학원생과 학과의 관계 2

대학원생과 학과의 관계를 보면 전공학과(major)의 관계로서, 일대다(1:N)의 관계이다. 여기서, 한 명의 대학원생은 하나의 학과만 전공학과를 가질 수 있으므로, 대학원생 측이 N, 학과 측이 1 이 된다. 대학원생은 전공학과에 필수적으로 참여하고, 학과는 전공학과에 선택적으로 참여한다. 즉, 대학원생이 전공학과로 선택하지 않은 학과가 존재 할 수 있다. 일대다 관계의 경우 일반적으로 릴레이션으로 변환하지 않고 외래키를 추가한다.

대학원생 개체가 학과 개체의 존재 여부를 결정하지 않는 일반적인 일대다 관계이므로 외래키로 표현한다. 일대다의 관계에서 N 측의 릴레이션에 1 측의 릴레이션의 기본 키를 참조하는 외래키를 추가하여야 한다.

이에 따라, 대학원생 릴레이션을 다음과 같이 수정한다.

ssn	name	age	deg_prog	ssn (멘티, 외래키)	dno (전공학과, 외래키)
1	GD_Kim	26	석사	NULL	1
2	GD_Jin	27	석사	1	1
3	GD_Park	28	석사	NULL	2
4	GD_Lee	29	박사	3	3
5	GD_Roh	30	박사	NULL	4

표 11) 다시 수정 된 Graduate 릴레이션

5. 논리적 설계 단계의 최종 릴레이션 스키마

최종적으로 완성된 한빛대학 학사관리 데이터베이스 스키마는 다음과 같다.

아이디 (ssn, 기본키)	이름 (name)	나이 (age)	직위 (rank)	연구분야 (speciality)

표 12) 교수(Professor) 릴레이션 스키마

학과번호 (dno, 기본키)	학과명 (dname)	학과 사무실 (office)	학과장 (ssn, 외래키)

표 13) 학과(Dept) 릴레이션 스키마

아이디 (ssn, 기본키)	이름 (name)	나이 (age)	학위과정 (deg_prog)	멘티 (ssn, 외래키)	전공학과 (dno, 외래키)

표 14) 대학원생(Graduate) 릴레이션
스키마

과제번호 (pid, 기본키)	지원기관 (sponser)	개시일 (start_date)	종료일 (end_date)	예산액 (budget)	연구책임자 (ssn, 외래키)

표 15) 과제(Project) 릴레이션 스키마

근무번호 (wdpt_id, 기본키)	근무교수 (ssn, 외래키)	근무학과 (dno, 외래키)	참여백분율 (pct_time)

표 16) 교수와 학과 간의 근무(work_dept) 릴레이션 스키마

수행번호 (win_id, 기본키)	수행교수 (ssn, 외래키)	수행과제 (pid, 외래키)

표 17) 교수와 과제 간의 수행(work_in) 릴레이션
스키마

수행번호 (wpr_id, 기본키)	수행 대학원생 (ssn, 외래키)	수행과제 (pid, 외래키)

표 18) 대학원생과 과제 간의 수행(work_prog) 릴레이션 스키마

6. 최종 릴레이션 스키마의 테이블 명세서

각 릴레이션 속성의 데이터 타입, 길이, NULL 값 허용 여부, 기본값, 제약조건을 정한다. 기본 키에 해당하는 아이디 혹은 학과번호, 과제번호 등에 대하여 적절한 크기의 데이터 타입을 선택하여야 저장공간의 낭비가 없을 것이다. 대학 학사관리 데이터베이스로서 정말 큰 대학이라면 32767 범위를 넘어서는 SMALLINT로 타입을 정하기에는 부족할 지도 모르지만, 적당한 크기의 일반적인 대학이라고 가정하고 SMALLINT로 교수 아이디, 대학원생 아이디, 학과번호, 과제번호, 근무번호, 수행번호를 할당한다. 나이의 경우, TINYINT로 저장한다면 127살까지 표현 할 수 있다. 일반적으로 현재 127살이상 생존하는 경우는 거의 없으므로, TINYINT를 사용한다.

테이블 명	교수(Professor)					
속성이름	데이터 타입	NULL 허용	기본값	기본키	외래키	제약조건
아이디(ssn)	SMALLINT	N		PK		자동 증가, 삭제 혹은 수정 시 참조하는 테이블에도 적용
이름(name)	VARCHAR(10)	N				
나이(age)	TINYINT	N				0 이상
직위(rank)	ENUM	N				전임강사, 조교수,부교수,정교수,명예교수,연구교수,산학협력교수,시간강사,겸임교수,임상교수,초빙교수,석좌교수만 허용
연구분야(speciality)	VARCHAR(10)	N				

표 19) 교수(Professor) 릴레이션 스키마의 테이블 명세서

교수의 아이디는 새로운 데이터를 입력 시마다 자동 증가되도록 한다. 또한, 교수의 아이디를 다른 릴레이션에서 외래키로 참조하므로 삭제 혹은 수정 시에도 변경 내용이 참조되는 릴레이션에도 적용되도록 한다.
교수의 직위는 데이터베이스 사용자가 직위 목록에 대해서만 선택하여 입력 혹은 수정 할 수 있도록 열거형으로 처리한다.

테이블 명	학과(Dept)					
속성이름	데이터 타입	NULL 허용	기본값	기본키	외래키	제약조건
학과번호(dno)	SMALLINT	N		PK		자동 증가, 삭제 혹은 수정 시 참조하는 테이블에도 적용
학과이름(dname)	VARCHAR(10)	N				UNIQUE
학과사무실(office)	VARCHAR(20)	Y				UNIQUE
학과장(ssn)	SMALLINT	Y			FK	UNIQUE

표 20) 학과(Dept) 릴레이션 스키마의 테이블 명세서

학과번호는 새로운 데이터 입력 시마다 자동 증가되도록 한다.
또한, 학과번호를 다른 릴레이션에서 외래키로 참조하므로, 삭제 혹은 수정 시에도 변경 내용이 참조되는 릴레이션에도 적용되도록 한다.
학과 사무실의 경우, 새로운 학과가 생긴다면 사무실 배정이 안되는 상황이 발생 할 수 있으므로, NULL 값을 허용한다. 여러 학과가 한 사무실을 같이 쓰지 않으므로, 중복 데이터는 입력 할 수 없다.
각 학과마다 그 학과를 운영하는 학과장이 존재해야만 한다. 하지만, 특수한 사정에 의하여 학과 교수회에 의하여 학과장이 아직 선출되지 않은 상황이 발생한다면, NULL 값을 허용함으로서 나중에 학과장이 선출되었을 시 입력할 수 있도록 한다. 학과장은 각 학과마다 한 명씩 할당되며, 중복을 허용하지 않는다. 한 교수가 여러 학과의 학과장을 맡을 수 없다.

테이블 명	대학원생(Graduate)					
속성이름	데이터 타입	NULL 허용	기본값	기본키	외래키	제약조건
아이디(ssn)	SMALLINT	N		PK		자동 증가, 삭제 혹은 수정 시 참조하는 테이블에도 적용
이름(name)	VARCHAR(10)	N				
나이(age)	TINYINT	N				0 이상
학위과정(deg_prog)	ENUM	N				석사/박사만 가능
멘티(ssn)	SMALLINT	Y			FK	
전공학과(dno)	SMALLINT	N			FK	

표 21) 대학원생(Graduate) 릴레이션 스키마의 테이블 명세서

대학원생의 아이디는 새로운 데이터 입력 시마다 자동 증가되도록 한다. 또한, 대학원생의 아이디는 다른 릴레이션 및 멘티로서 외래키로 참조하므로 삭제 혹은 수정 시에도 변경사항이 적용되도록 한다.
학위과정은 데이터베이스 사용자가 '석사' 혹은 '박사'만 선택 할 수 있도록 열거형으로 처리한다.
대학원생은 반드시 조언을 주거나 받을 필요는 없으므로 멘티 항목은 NULL 값을 허용한다.
어떠한 한 명의 대학원생은 여러 선임 대학원생에게 조언을 받을 수 있으므로, 멘티(선임 대학원생 측에서 조언을 해주는 대학원생) 항목은 중복을 허용한다.
대학원생은 반드시 전공학과를 가져야 하므로 전공학과 항목은 NULL 값을 허용하지 않는다.
또한, 전공학과는 어떠한 한 학과에 대하여 여러 대학원생을 수용하므로, 중복을 허용한다.

테이블 명	과제(Project)					
속성이름	데이터 타입	NULL 허용	기본값	기본키	외래키	제약조건
과제번호(pid)	SMALLINT	N		PK		자동 증가, 삭제 혹은 수정 시 참조하는 테이블에도 적용
지원기관(sponser)	VARCHAR(20)	Y				
개시일(start_date)	DATE	Y				
종료일(end_date)	DATE	Y				
예산액(budget)	INT	Y				0 이상
연구책임자(ssn)	SMALLINT	N			FK	

표 22) 과제(Project) 릴레이션 스키마의 테이블 명세서

과제번호는 새로운 데이터 입력 시 마다 자동 증가되도록 한다. 또한, 과제번호는 다른 릴레이션에서 외래키로 참조하므로, 삭제 혹은 수정 시에도 변경 내용이 참조되는 릴레이션에도 적용되도록 한다.
과제의 지원기관, 개시일, 종료일, 예산액은 아직 미정일 수 있으므로, NULL 값을 허용한다.
예산액은 일반적인 대학에서 21억 이상 ~ 경 단위까지 가는 경우는 없으므로, 자료형은 INT 형으로 정한다.
과제는 한 교수에 의해서만 관리된다. 하지만, 한 교수는 여러 과제를 관리 할 수 있다. 따라서, 연구책임자 항목은 중복을 허용한다.

테이블 명	근무(work_dept)					
속성이름	데이터 타입	NULL 허용	기본값	기본키	외래키	제약조건
근무번호(wdpt_id)	SMALLINT	N		PK		자동 증가
근무교수(ssn)	SMALLINT	N			FK	
근무학과(dno)	SMALLINT	N			FK	
참여백분율(pct_time)	FLOAT	Y	0.0			자동 계산. 0.0~1.0

표 23) 근무(work_dept) 릴레이션 스키마의 테이블 명세서

근무번호는 새로운 데이터 입력 시마다 자동 증가되도록 한다.
근무(work_dept) 릴레이션에서 교수 혹은 학과 하나라도 NULL 값을 허용하게 된다면, 근무 관계가 성립하지 않으므로 NULL 값을 허용하지 않는다.
요구사항에 한 교수가 여러 학과에 근무 할 수 있다고 했으므로, 근무교수 항목과 근무학과 항목은 중복을 허용한다.
참여백분율은 교수가 근무하는 학과마다 **(해당 학과에 근무하는 교수의 수 / 근무(work_dept)에 참여하는 전체 교수의 수)**의 비율에 따라 근무 릴레이션에 삽입, 삭제, 수정 작업 발생 시 자동 계산하여 근무 릴레이션의 모든 레코드에 대하여 학과별로 갱신한다. 따라서, 참여백분율은 0.0 ~ 1.0의 범위를 가진다.

테이블 명	수행(work_in)					
속성이름	데이터 타입	NULL 허용	기본값	기본키	외래키	제약조건
수행번호(win_id)	SMALLINT	N		PK		자동 증가
수행교수(ssn)	SMALLINT	N			FK	
수행과제(pid)	SMALLINT	N			FK	

표 24) 수행(work_in) 릴레이션 스키마의 테이블 명세서

교수와 과제 간의 관계인 수행(work_in) 릴레이션에서의 수행번호(win_id)는 새로운 데이터 입력 시마다 자동 증가시킨다.
수행(work_in) 관계에서 수행교수, 수행과제 하나라도 NULL 값을 허용하면 수행 관계가 성립하지 않으므로 NULL 값을 허용하지 않는다
과제는 한 사람 이상의 교수에 의해 수행되므로, 수행교수와 수행과제 항목은 중복을 허용한다.

테이블 명	수행(work_prog)					
속성이름	데이터 타입	NULL 허용	기본값	기본키	외래키	제약조건
수행번호(wpr_id)	SMALLINT	N		PK		자동 증가
수행 대학원생(ssn)	SMALLINT	N			FK	
수행과제(pid)	SMALLINT	N			FK	

표 25) 수행(work_prog) 릴레이션 스키마의 테이블 명세서

대학원생과 과제 간의 관계인 수행(work_prog) 릴레이션에서의 수행번호(wpr_id)는 새로운 데이터 입력 시마다 자동 증가시킨다.
수행(work_prog) 관계에서 수행 대학원생, 수행과제 하나라도 NULL값을 허용하면 수행 관계

가 성립하지 않으므로 NULL 값을 허용하지 않는다.
한 과제는 한 명 이상의 대학원생에 의해 수행되므로, 수행 대학원생과 수행과제 항목은 중복을 허용한다.

7. 물리적 설계 및 구현

최종 릴레이션 스키마의 테이블 명세서에 따라 한빛대학 학사관리 데이터베이스를 물리적 설계 및 구현한다.

```
CREATE TABLE Professor -- 교수 테이블
(
    아이디 SMALLINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    이름 VARCHAR(10) NOT NULL,
    나이 TINYINT NOT NULL,
    직위 ENUM('전임강사','조교수','부교수','정교수','명예교수',
    '연구교수','산학협력교수','시간강사','겸임교수','임상교수',
    '초빙교수','석좌교수') NOT NULL,
    연구분야 VARCHAR(10) NOT NULL,

    CHECK(나이 >= 0)
);
```

그림 11) 명세서에 따른 Professor 테이블 구현

```
CREATE TABLE Dept -- 학과 테이블
(
    학과번호 SMALLINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    학과이름 VARCHAR(10) NOT NULL UNIQUE,
    학과사무실 VARCHAR(20) UNIQUE,
    학과장 SMALLINT UNIQUE,

    FOREIGN KEY(학과장) REFERENCES Professor(아이디) ON DELETE CASCADE ON UPDATE CASCADE
);
```

그림 12) 명세서에 따른 Dept 테이블 구현

```sql
CREATE TABLE Graduate -- 대학원생 테이블
(
    아이디 SMALLINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    이름 VARCHAR(10) NOT NULL,
    나이 TINYINT NOT NULL,
    학위과정 ENUM('석사', '박사') NOT NULL,
    멘티 SMALLINT,
    전공학과 SMALLINT NOT NULL,

    CHECK(나이 >= 0),
    FOREIGN KEY(멘티) REFERENCES Graduate(아이디) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(전공학과) REFERENCES Dept(학과번호) ON DELETE CASCADE ON UPDATE CASCADE
);
```

그림 13) 명세서에 따른 Graduate 테이블 구현

```sql
CREATE TABLE Project -- 과제 테이블
(
    과제번호 SMALLINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    지원기관 VARCHAR(20),
    개시일 DATE,
    종료일 DATE,
    예산액 INT,
    연구책임자 SMALLINT NOT NULL,

    CHECK(예산액 >= 0),
    FOREIGN KEY(연구책임자) REFERENCES Professor(아이디) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE work_dept -- 근무 테이블 (교수와 학과 간의 관계)
(
    근무번호 SMALLINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    근무교수 SMALLINT NOT NULL,
    근무학과 SMALLINT NOT NULL,
    참여백분율 FLOAT DEFAULT 0.0,

    CHECK(참여백분율 >= 0.0 AND 참여백분율 <= 1.0),
    FOREIGN KEY(근무교수) REFERENCES Professor(아이디) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(근무학과) REFERENCES Dept(학과번호) ON DELETE CASCADE ON UPDATE CASCADE
);
```

그림 15) 명세서에 따른 work_dept 테이블 구현

```
CREATE TABLE work_in -- 수행 테이블 (교수와 과제 간의 관계)
(
    수행번호 SMALLINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    수행교수 SMALLINT NOT NULL,
    수행과제 SMALLINT NOT NULL,

    FOREIGN KEY(수행교수) REFERENCES Professor(아이디) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(수행과제) REFERENCES Project(과제번호) ON DELETE CASCADE ON UPDATE CASCADE
);
```

그림 16) 명세서에 따른 work_in 테이블 구현

```
CREATE TABLE work_prog-- 수행 테이블 (대학원생과 과제 간의 관계)
(
    수행번호 SMALLINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    수행대학원생 SMALLINT NOT NULL,
    수행과제 SMALLINT NOT NULL,

    FOREIGN KEY(수행대학원생) REFERENCES Graduate(아이디) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(수행과제) REFERENCES Project(과제번호) ON DELETE CASCADE ON UPDATE CASCADE
);
```

그림 17) 명세서에 따른 work_prog 테이블 구현

```
DROP PROCEDURE IF EXISTS update_pct_time;
DELIMITER //
CREATE PROCEDURE update_pct_time()
BEGIN
   DECLARE total_work_dept_pf_cnt SMALLINT; -- 근무 릴레이션의 전체 교수의 수
   SELECT COUNT(*) INTO total_work_dept_pf_cnt FROM work_dept; -- 전체 근무에 참여하는 교수의 수
   -- 각 근무학과별로 그룹 지어 각 근무학과별 근무에 참여하는 교수의 수 / 전체 근무에 참여하는 교수의 수로 참여백분율 설정 (내부
   UPDATE work_dept a
       INNER JOIN
       (SELECT
           COUNT(*) grouped_cnt,
           근무학과
       FROM
           work_dept
       GROUP BY 근무학과) b
       ON a.근무학과 = b.근무학과
   SET
   a.참여백분율 = (b.grouped_cnt / total_work_dept_pf_cnt);
END //
DELIMITER ;
```

그림 18) 근무(work_dept) 테이블에 대한 변경사항 발생 시 해당 테이블의 모든 레코드에 대하여 근무학과별 참여백분율을 갱신하는 프로시저 정의

근무(work_dept) 테이블에 대하여 삽입, 수정, 삭제 작업이 발생하면 이하 그림 19) ~ 그림

23) 까지 정의된 기존의 INSERT, UPDATE, DELETE 문을 대체하는 프로시저를 호출하여 수행한다. 이 프로시저들은 근무(work_dept) 테이블에 대하여 INSERT, UPDATE, DELETE 작업을 수행 후 내부적으로 **그림 18) 근무(work_dept) 테이블에 대한 변경사항 발생 시 해당 테이블의 모든 레코드에 대하여 근무학과별 참여백분율을 갱신하는 프로시저 정의**에 따라 만들어진 update_pct_time() 을 호출하여 근무(work_dept) 테이블의 모든 레코드에 대하여 근무학과별 참여백분율을 갱신한다.

```
DROP PROCEDURE IF EXISTS insert_work_dept;
DELIMITER $$
CREATE PROCEDURE insert_work_dept(IN 근무교수 SMALLINT , IN 근무학과 SMALLINT) -- work_dept에 대한 삽입 작업 수행
BEGIN
   INSERT INTO work_dept VALUES(NULL, 근무교수, 근무학과, NULL);
   CALL update_pct_time(); -- 각 근무학과별로 참여백분율 갱신
END$$
DELIMITER ;
```

그림 19) 근무(work_dept) 테이블에 대하여 기존의 INSERT 명령어를 대체하는 프로시저 정의

```
DROP PROCEDURE IF EXISTS update_work_dept;
DELIMITER $$
CREATE PROCEDURE update_work_dept(IN old_src_근무교수 SMALLINT , IN old_src_근무학과 SMALLINT, IN new_src_근무교수 SMALLINT,
IN new_src_근무학과 SMALLINT) -- work_dept에 대한 수정 작업 수행
BEGIN
   UPDATE work_dept SET 근무교수 = new_src_근무교수, 근무학과 = new_src_근무학과
   WHERE 근무교수 = old_src_근무교수 AND 근무학과 = old_src_근무학과;
   CALL update_pct_time(); -- 각 근무학과별로 참여백분율 갱신
END$$
DELIMITER ;
```

그림 20) 근무(work_dept) 테이블에 대하여 기존의 UPDATE 명령어를 대체하는 프로시저 정의 1

```
DROP PROCEDURE IF EXISTS update_work_dept_old_pk;
DELIMITER $$
CREATE PROCEDURE update_work_dept_old_pk(IN old_src_근무번호 SMALLINT ,
IN new_src_근무교수 SMALLINT, IN new_src_근무학과 SMALLINT) -- work_dept에 대한 수정 작업 수행
BEGIN
    UPDATE work_dept SET 근무교수 = new_src_근무교수, 근무학과 = new_src_근무학과
    WHERE 근무번호 = old_src_근무번호;
    CALL update_pct_time(); -- 각 근무학과별로 참여백분율 갱신
END$$
DELIMITER ;
```

그림 21) 근무(work_dept) 테이블에 대하여 기존의 UPDATE 명령어를 대체하는 프로시저 정의 2

```
DROP PROCEDURE IF EXISTS delete_work_dept;
DELIMITER $$
CREATE PROCEDURE delete_work_dept(IN src_근무교수 SMALLINT , IN src_근무학과 SMALLINT) -- work_dept에 대한 삭제 작업 수행
BEGIN
    DELETE FROM work_dept WHERE 근무교수 = src_근무교수 AND 근무학과 = src_근무학과;
    CALL update_pct_time(); -- 각 근무학과별로 참여백분율 갱신
END$$
DELIMITER ;
```

그림 22) 근무(work_dept) 테이블에 대하여 기존의 DELETE 명령어를 대체하는 프로시저 정의 1

```
DROP PROCEDURE IF EXISTS delete_work_dept_pk;
DELIMITER $$
CREATE PROCEDURE delete_work_dept_pk(IN src_근무번호 SMALLINT) -- work_dept에 대한 삭제 작업 수행
BEGIN
    DELETE FROM work_dept WHERE 근무번호 = src_근무번호;
    CALL update_pct_time(); -- 각 근무학과별로 참여백분율 갱신
END$$
DELIMITER ;
```

그림 23) 근무(work_dept) 테이블에 대하여 기존의 DELETE 명령어를 대체하는 프로시저 정의 2

데이터베이스의 사용자가 어떠한 방법으로 사용하여도 원하는 연산이 수행되도록, 기존의 레코드에 대하여 수정 시 단순히 PK인 근무번호와 새로운 데이터를 입력하거나 삭제 시 근무번호만 입력하여 연산을 수행 할 수도 있고, 수정 혹은 삭제 시 근무교수와 근무학과를 입력하여도 수행되도록 하였다.

```
INSERT INTO Professor VALUES (NULL, 'PF_Kim', 30, '조교수', '운영체제');
INSERT INTO Professor VALUES (NULL, 'PF_Jin', 40, '부교수', '임베디드');
INSERT INTO Professor VALUES (NULL, 'PF_Park', 50, '정교수', '네트워크');
INSERT INTO Professor VALUES (NULL, 'PF_Lee', 60, '명예교수', '그래픽스');
INSERT INTO Professor VALUES (NULL, 'PF_Roh', 70, '명예교수', '데이터베이스');
INSERT INTO Dept VALUES (NULL, '컴퓨터공학과', '100호', 1);
INSERT INTO Dept VALUES (NULL, '기계과', '101호', 2);
INSERT INTO Dept VALUES (NULL, '전자과', '102호', 3);
INSERT INTO Dept VALUES (NULL, '정보통신학과', '103호', 4);
INSERT INTO Dept VALUES (NULL, '정보시스템과', '104호', 5);
INSERT INTO Graduate VALUES (NULL, 'GD_Kim', 26, '석사', NULL, 1);
INSERT INTO Graduate VALUES (NULL, 'GD_Jin', 27, '석사', 1, 1);
INSERT INTO Graduate VALUES (NULL, 'GD_Park', 28, '석사', NULL, 2);
INSERT INTO Graduate VALUES (NULL, 'GD_Lee', 29, '박사', 3, 3);
INSERT INTO Graduate VALUES (NULL, 'GD_Roh', 30, '박사', NULL, 4);
INSERT INTO Project VALUES (NULL, '교육부', '2020-01-01', '2020-01-30', 1000000, 1);
INSERT INTO Project VALUES (NULL, '국방부', '2020-02-01', '2020-02-29', 2000000, 1);
INSERT INTO Project VALUES (NULL, '국세청', '2020-03-01', '2020-03-30', 3000000, 2);
INSERT INTO Project VALUES (NULL, '고용노동부', '2020-04-01', '2020-04-30', 4000000, 3);
INSERT INTO Project VALUES (NULL, '기상청', '2020-05-01', '2020-05-30', 5000000, 4);
```

그림 24) Professor, Dept, Graduate, Project 테이블 오류 검출을 위한 테스트 데이터 입력

그림 24) 오류 검출을 위한 테스트 데이터 입력 1에 따라 Professor, Dept, Graduate, Project 테이블에 입력을 실시한다. 해당 데이터는 제약사항을 위반하지 않도록, 모든 조건을 고려하여 만들었다.

데이터의 입력 순서는 논리적 설계의 최종 릴레이션 스키마에서 정의 한 순서와 같으며, 상세 내용은 다음과 같다.

- Professor : 아이디, 이름, 나이, 직위, 연구분야
- Dept : 학과번호, 학과명, 학과사무실, 학과장
- Graduate : 아이디, 이름, 나이, 학위과정, 멘티, 전공학과
- Project : 과제번호, 지원기관, 개시일, 종료일, 예산액, 연구책임자

	아이디	이름	나이	직위	연구분야
▶	1	PF_Kim	30	조교수	운영체제
	2	PF_Jin	40	부교수	임베디드
	3	PF_Park	50	정교수	네트워크
	4	PF_Lee	60	명예교수	그래픽스
	5	PF_Roh	70	명예교수	데이터베이스
*	NULL	NULL	NULL	NULL	NULL

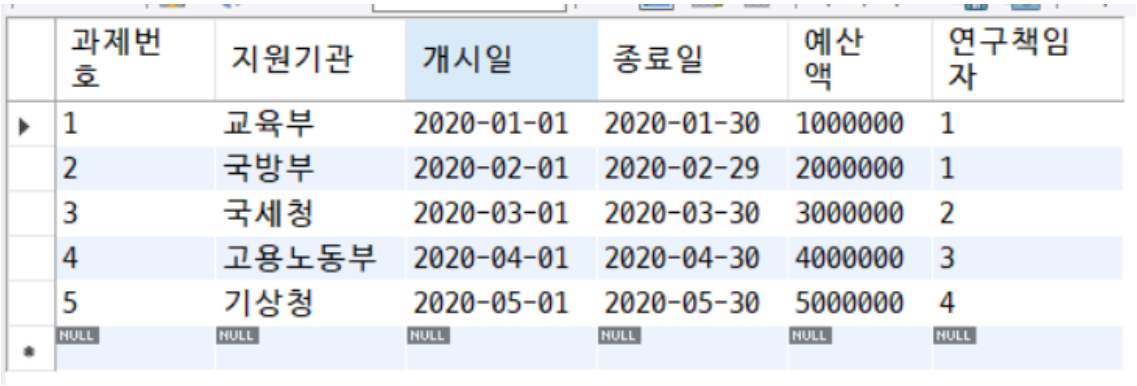
그림 25) Professor 테이블 삽입 결과

	학과번호	학과이름	학과사무실	학과장
▶	1	컴퓨터공학과	100호	1
	2	기계과	101호	2
	3	전자과	102호	3
	4	정보통신학과	103호	4
	5	정보시스템과	104호	5
*	NULL	NULL	NULL	NULL

그림 26) Dept 테이블 삽입 결과

	아이디	이름	나이	학위과정	멘티	전공학과
▶	1	GD_Kim	26	석사	NULL	1
	2	GD_Jin	27	석사	1	1
	3	GD_Park	28	석사	NULL	2
	4	GD_Lee	29	박사	3	3
	5	GD_Roh	30	박사	NULL	4
*	NULL	NULL	NULL	NULL	NULL	NULL

그림 27) Graduate 테이블 삽입 결과



과제번호	지원기관	개시일	종료일	예산액	연구책임자
1	교육부	2020-01-01	2020-01-30	1000000	1
2	국방부	2020-02-01	2020-02-29	2000000	1
3	국세청	2020-03-01	2020-03-30	3000000	2
4	고용노동부	2020-04-01	2020-04-30	4000000	3
5	기상청	2020-05-01	2020-05-30	5000000	4
NULL	NULL	NULL	NULL	NULL	NULL

그림 28) Project 테이블 삽입 결과

```
/***
    교수와 학과 간의 근무 테이블은 참여백분율 자동 계산위해 사용자 정의 프로시저 호출로만 수행
    ---
    입력 : 근무교수 아이디, 근무학과 아이디만 입력(insert_work_dept)
    수정 : 변경하고자 하는 근무교수 아이디, 변경하고자 하는 근무학과 아이디, 새로운 근무교수 아이디, 새로운 근무학과 아이디 (update_work_dept)
           or 변경하고자 하는 근무번호, 새로운 근무학과 아이디, 새로운 근무교수 아이디(update_work_dept_old_pk)
    삭제 : 근무교수 아이디, 근무학과 아이디만 입력(delete_work_dept) or 근무번호만 입력(delete_work_dept_pk)
***/
CALL insert_work_dept(1, 1); -- 교수아이디 : 1, 학과 아이디 : 1 레코드 입력
CALL insert_work_dept(2, 1);
CALL insert_work_dept(3, 3);
CALL insert_work_dept(4, 3);
CALL insert_work_dept(5, 5);
```

그림 29) 근무(work_dept) 테이블 오류 검출을 위한 테스트 데이터 입력 1

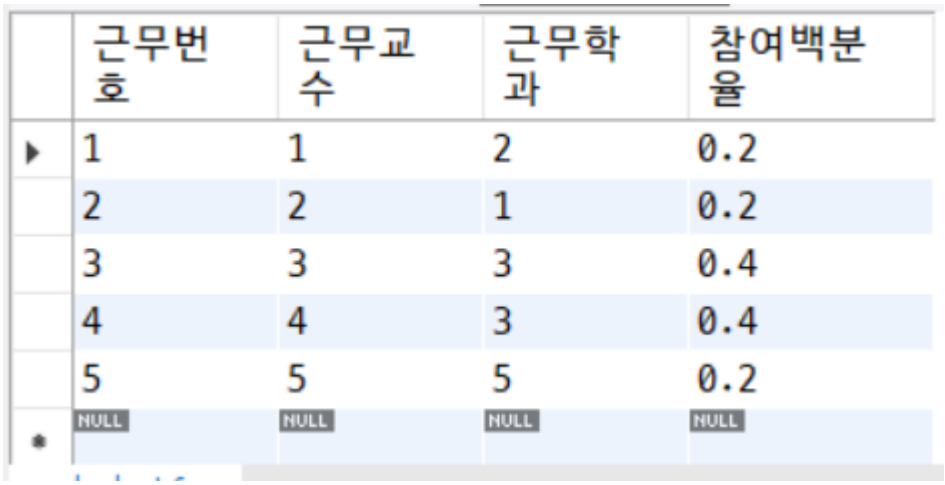
근무번호	근무교수	근무학과	참여백분율
1	1	1	0.4
2	2	1	0.4
3	3	3	0.4
4	4	3	0.4
5	5	5	0.2
NULL	NULL	NULL	NULL

그림 30) 근무(work_dept) 테이블 삽입 결과

근무(work_dept) 테이블에 데이터를 입력하기 위해 기존의 INSERT 문을 대체하는 프로시저를 호출하여 데이터를 입력한다. 이는, 내부적으로 근무(work_dept) 테이블의 참여백분율을 갱신하는 다른 프로시저를 호출한다. 이에 따라, 갱신된 근무(work_dept) 테이블의 결과는 **그림 30) 근무(work_dept) 테이블 삽입 결과**와 같다. 각 학과별로 근무에 참여하는 전체 교수에 대해 학과 별 교수 비율이 올바르게 나왔음을 알 수 있다.

```
CALL update_work_dept(1, 1, 1, 2); -- 기존 교수아이디 : 1, 학과 아이디 : 1 인 레코드를 교수아이디 : 1, 학과 아이디 : 2 로 변경
SELECT * FROM work_dept; -- 교수와 학과간의 근무테이블
```

그림 31) 근무(work_dept) 테이블 오류 검출을 위한 테스트 데이터 입력 2



	근무번호	근무교수	근무학과	참여백분율
▶	1	1	2	0.2
	2	2	1	0.2
	3	3	3	0.4
	4	4	3	0.4
	5	5	5	0.2
*	NULL	NULL	NULL	NULL

그림 32) 근무(work_dept) 테이블 변경 결과

기존의 UPDATE 문을 대체하는 프로시저를 호출하여 근무(work_dept) 테이블의 데이터가 새로운 데이터로 변경되었는지 확인한다. 이에 따른 결과는, **그림 32) 근무(work_dept) 테이블 변경 결과**와 같으며, 새로운 학과로 올바르게 변경되었고, 참여백분율도 이에 따라 모두 올바르게 갱신되었음을 확인할 수 있다.

```
CALL delete_Work_dept(1, 2); -- 교수아이디 : 1, 학과 아이디 : 2 인 레코드 삭제
SELECT * FROM work_dept; -- 교수와 학과간의 근무테이블
```

그림 33) 근무(work_dept) 테이블 오류 검출을 위한 테스트 데이터 입력 3

	근무번호	근무교수	근무학과	참여백분율
▶	2	2	1	0.25
	3	3	3	0.5
	4	4	3	0.5
	5	5	5	0.25
*	NULL	NULL	NULL	NULL

그림 34) 근무(work_dept) 테이블 삭제 결과

기존의 DELETE 문을 대체하는 프로시저를 호출하여 근무(work_dept) 테이블의 삭제를 원하는 레코드가 삭제 되었는지 확인한다. 이에 따른 결과는, **그림 34) 근무(work_dept) 테이블 삭제 결과**와 같으며, 참여백분율도 이에 따라 모두 올바르게 갱신되었음을 확인할 수 있다.

```
-- 과제는 한 사람 이상의 교수에 의해 수행되므로, 수행교수와 수행과제 항목은 중복을 허용
INSERT INTO work_in VALUES(NULL, 1, 1); -- 수행교수, 수행과제
INSERT INTO work_in VALUES(NULL, 1, 2); -- 수행교수, 수행과제
INSERT INTO work_in VALUES(NULL, 1, 3); -- 수행교수, 수행과제
INSERT INTO work_in VALUES(NULL, 1, 4); -- 수행교수, 수행과제
INSERT INTO work_in VALUES(NULL, 2, 5); -- 수행교수, 수행과제
SELECT * FROM work_in; -- 교수와 과제 간의 수행 테이블
```

그림 35) 수행(work_in) 테이블 오류 검출을 위한 테스트 데이터 입력

	수행번호	수행교수	수행과제
▶	1	1	1
	2	1	2
	3	1	3
	4	1	4
	5	2	5
*	NULL	NULL	NULL

그림 36) 수행(work_in) 테이블 삽입 결과

수행(work_in) 테이블에 대하여 데이터를 삽입하여 올바르게 입력되는지 확인한다. 이에 따른 결과는, **그림 36) 수행(work_in) 테이블 삽입 결과**와 같으며 이상 없이 입력되었음을 확인 할 수 있다.

```
-- 한 과제는 한 명 이상의 대학원생에 의해 수행되므로, 수행 대학원생과 수행과제 항목은 중복을 허용
INSERT INTO work_prog VALUES(NULL, 1, 1); -- 수행 대학원생, 수행과제
INSERT INTO work_prog VALUES(NULL, 1, 2); -- 수행 대학원생, 수행과제
INSERT INTO work_prog VALUES(NULL, 1, 3); -- 수행 대학원생, 수행과제
INSERT INTO work_prog VALUES(NULL, 2, 1); -- 수행 대학원생, 수행과제
INSERT INTO work_prog VALUES(NULL, 3, 5); -- 수행 대학원생, 수행과제
SELECT * FROM work_prog;
```

그림 37) 수행(work_prog) 테이블 오류 검출을 위한 테스트 데이터 입력

	수행번호	수행대학원생	수행과제
▶	1	1	1
	2	1	2
	3	1	3
	4	2	1
	5	3	5
*	NULL	NULL	NULL

그림 38) 수행(work_prog) 테이블 삽입 결과

수행(work_prog) 테이블에 대하여 데이터를 삽입하여 올바르게 입력되는지 확인한다. 이에 따른 결과는, **그림 38) 수행(work_prog) 테이블 삽입 결과**와 같으며 이상 없이 입력되었음을 확인 할 수 있다.