**데이터베이스 프로젝트**

**병원 DB 구축**

컴퓨터공학과

2014313366 홍기원

제출일 : 2016/06/21

**1. DB 요구사항 작성**

**1.1 병원 DB 요구사항**

병원의 Database는 다음과 같이 구성된다. 병원은 **부서**들로 구성된다. **부서**는 *부서번호*, *부서명*, *부서의 전화번호*를 가진다. **부서**는 *부서번호*를 통해 식별된다. 부서의 *전화번호*는 여러 개일 수 있다. *부서명*이 같은 **부서**들은 없다. **부서**는 **진료부**와 **간호부**로 구분된다. 모든 **부서**는 반드시 **진료부** 혹은 **간호부** 둘 중 하나에 속해야 하며, **진료부**와 **간호부**에 동시에 속할 수 없다. **진료부**는 추가로 *평균 연봉*이라는 속성을 가진다.

**직원**은 *사원번호*, *이름*, *주민번호*, *나이*, *전화번호*, *연봉*, *근무연수*를 가진다. **직원**은 *사원번호*를 통해 식별된다. *주민번호*가 같은 **직원**은 없다. 같은 *이름*과 *나이*를 가지는 **직원**들이 있을 수 있다. 각 **직원**의 *전화번호*는 여러 개일 수 있다. **직원**은 **간호사**와 **의사**로 구분된다. 모든 **직원**은 **간호사**이거나 **의사**이며, **간호사**나 **의사** 둘 중 하나에만 속한다. **간호사**의 *연봉*은 *근무연수*에 의해 계산된다 (연봉 = 2680 + 근무연수\*100). **의사**는 추가로 *전공*을 가지며, *전공*은 **의사**가 속한 **부서**의 *이름*과 같아야 한다. (*전공*을 *부서의 이름*으로 표시) **의사**의 *연봉*은 **의사**가 속한 **부서**의 *평균 연봉*에서 +-10% 내외여야 한다.

모든 **간호사**는 **간호부**에 속하는 **부서**에서 ***근무한다***. **간호부**에 속하는 **부서**에는 최소 한 명의 **간호사**가 ***근무해야 한다***. **간호사** 중 한 명은 반드시 속한 **부서**의 책임자가 되어 **부서**를 ***관리한다***.

모든 **의사**는 **진료부**에 속하는 **부서**에서 ***근무한다***. **진료부**에 속하는 **부서**에는 최소 한 명의 **의사**가 ***근무해야 한다***. **의사** 중 한 명은 반드시 속한 **부서**의 책임자가 되어 **부서**를 ***관리한다***.

**의사**는 **전공의**를 ***교육한다***. **전공의**는 **인턴**과 **레지던트**로 구분된다. 모든 **전공의**는 반드시 **인턴**이나 **레지던트**에 속해야 하며, **인턴**과 **레지던트**에 동시에 속할 수 없다. **전공의**는 *이름*, *나이*, *근무연수*를 가진다. 한 **의사**가 교육하는 **전공의**들 중에는 *이름*이 같거나 *나이*가 같은 **전공의**들이 있지만 *이름*과 *나이* 모두 같은 **전공의**는 없다. 서로 다른 **의사**가 교육하는 **전공의**들 중에는 *이름*과 *나이* 모두 같은 **전공의**가 있을 수 있다. **전공의**는 이*름*, *나이*, 교육하는 **의사**의 *사원번호*로 식별된다.

한 **의사**는 최대 5명의 **인턴**을 ***교육할 수 있고***, 한 **인턴**은 오직 단 한 명의 **의사**로부터만 ***교육받는다***. 모든 **의사**가 **인턴**을 ***교육해야 하는*** 것은 아니다. **인턴**의 *근무연수*는 1년을 넘을 수 없다.

한 **의사**는 최대 3명의 **레지던트**를 ***교육할 수 있고***, 한 **레지던트**는 오직 단 한 명의 **의사**로부터만 ***교육받는다.*** 모든 **의사**가 **레지던트**를 ***교육해야 하는*** 것은 아니다. **레지던트**는 추가로 *전공*을 가진다. **레지던트**의 *근무연수*는 4년을 넘을 수 없다.

**직원**의 *근무연수*는 그 **직원**의 *나이*보다 클 수 없다.

**환자**는 *환자번호*, *이름*, *주민번호*, *나이*, *전화번호*를 가진다. **환자**는 *환자번호*를 통해 식별된다. *주민번호*가 같은 **환자**들은 없다. 같은 *이름*과 *나이*를 가지는 **환자**들은 있을 수 있다. 각 **환자**에게는 담당**의사**와 **간호사**가 ***배정된다***. 한 **간호사**는 최대 5명의 **환자**를 ***담당하고***, 한 **환자**는 최대 3명의 **간호사**를 ***배정받는다***. 한 **환자**에게는 반드시 한 명 이상의 **간호사**가 ***배정되어야*** 하지만, 모든 **간호사**가 **환자**를 ***담당하는*** 것은 아니다. 한 **의사**는 최대 3명의 **환자**를 ***진료하고***, 한 **환자**는 오직 단 한 명의 담당**의사**를 ***배정받는다***. 모든 **환자**에게는 반드시 한 명의 담당 **의사**가 ***배정되어야*** 하지만, 모든 **의사**가 **환자**를 ***진료해야*** 하는 것은 아니다. **환자**가 담당 **의사**에게 ***진료받을*** 때, *진료번호* (환자가 받은 진료의 번호수), *진료명*, *진료날짜*가 저장된다. *진료번호*는 **환자**마다 같을 수 있다. (A라는 환자의 진료번호 1, B라는 환자의 진료번호 1) 하지만 한 **환자**가 받은 **진료**들의 *진료번호*는 모두 다르다.

**병동**은 *병동번호*, *병동이름*, *병상의 수* (입원할 수 있는 최대 환자 수)를 가진다. **병동**은 *병동번호*를 통해 식별된다. *병동이름*이 같은 **병동**은 없다. **환자**는 **병동**에 ***입원한다.*** **환자**가 ***입원한*** **병동**은 오직 한 개이지만, **병동**에 ***입원한*** **환자**는 여러 명 일 수 있다. 각 **병동**에 ***입원한*** **환자** 수는 그 **병동**의 *병상의 수*를 넘을 수 없다. 모든 **환자**가 **병동**에 ***입원하는*** 것은 아니며, 모든 **병동**에 **환자**가 ***입원하고*** 있어야 하는 것도 아니다. **병동**에 ***입원한*** **환자**는 *입원날짜*와 *퇴원날짜*를 가진다. *입원날짜*는 반드시 *퇴원날짜*보다 이전이거나 같아야 한다. 모든 **병동**은 반드시 **간호부**에 속한 단 하나의 **부서**로부터 ***관리 받아야 한다***. **간호부**에 속한 **부서**는 여러 개의 **병동**을 ***관리하지만***, ***관리하는*** **병동**이 없을 수도 있다. **환자**를 ***담당하는*** **간호사**는 반드시 그 **환자**가 ***입원한*** **병동**을 ***관리하는*** **부서**에 속해있어야 한다.

**1.2 참고 자료**

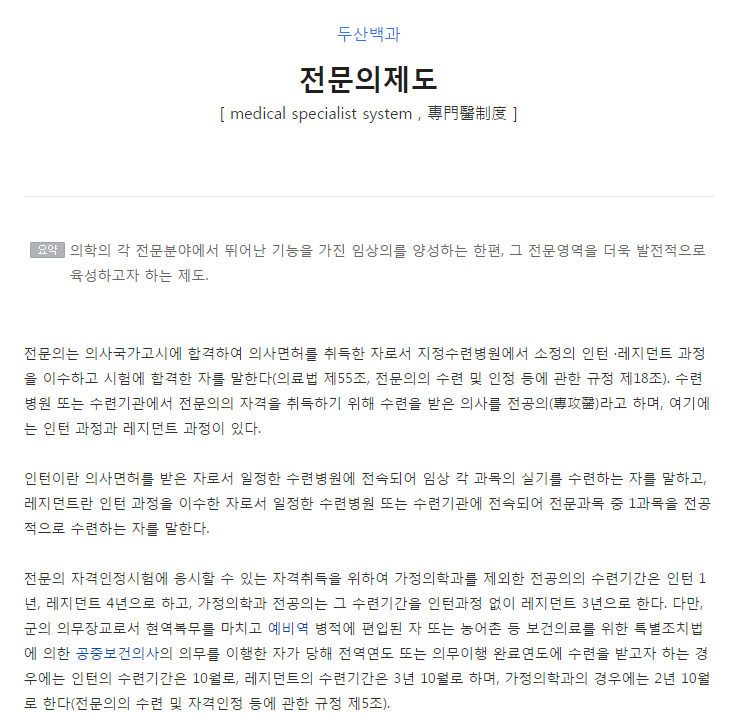
**A) 한양대학교 서울 병원의 조직도**

병원의 데이터베이스를 작성하기 위해서는 의사와 간호사가 어디에 속해있는지, 병원의 부서들에는 어떠한 종류가 있는지를 알아야 했다. 이를 위해 한양대학교 서울 병원의 조직도를 참고하였다. 조직도를 통해 크게 진료부와 간호부로 부서를 나누었고, 병동에 대한 관리를 간호부에서 한다는 것을 알게 되어(병동 1팀, 병동 2팀 등) 데이터베이스 작성에 반영하였다.



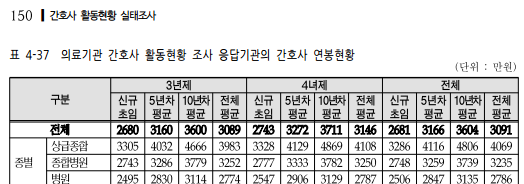
**B) 인터넷 백과사전**

의사들에 대한 정보를 검색해보다가 의사(지도전문의) 아래에서 교육을 받는 인턴과 레지던트(전문의제도)에 대해서 알게 되었다. 이에 대한 정보를 데이터베이스를 작성하는데 참고하여 인턴과 레지던트를 추가하였다. 인턴과 레지던트의 기간은 가정의학과를 제외하고 인턴 1년, 레지던트 4년이므로 인턴과 레지던트의 최대 근무 기간을 각각 1년과 4년으로 설정하였다. 또한 인턴과 레지던트를 합쳐서 전공의라고 부른다는 것을 검색을 통해 알게 되어, 전공의를 인턴과 레지던트로 구분하는 방식을 사용하여 데이터베이스를 작성하였다.



**C) 한국보건산업진흥원 – 간호사 활동현황 실태조사**

간호사의 급여에 대해서 찾아보기 위해 인터넷 검색을 수행한 결과, 한국보건산업진흥원의 간호사 활동현황 실태조사에서 간호사의 봉급에 대해서 찾아볼 수 있었다.



이 실태조사에서 간호사의 초봉이 2681만원, 5년차 평균이 3166만원, 10년차 평균이 3604만원으로 년당 약 100만원씩 봉급이 상승한다는 것을 알 수 있었다. 이를 통해 간호사의 봉급을 초봉 2680 만원 + 100\* 근무연수로 계산하는 값으로 설정하였다. 그리고 이 봉급과 근무연수를 간호사의 속성에 추가하여 derived attribute를 만들었다.

**D) 한국직업정보 시스템 – 의사의 전공별 평균연봉**

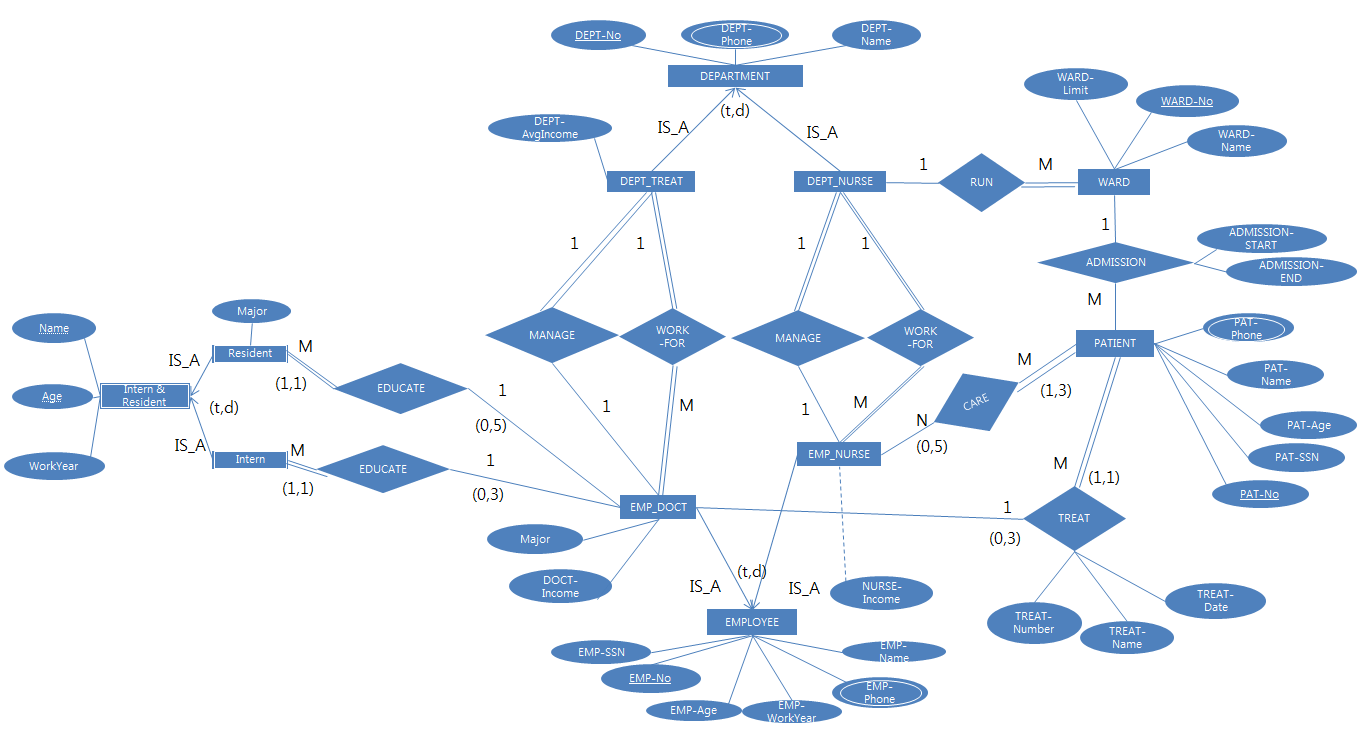
의사의 급여에 대해서 찾아보기 위해 인터넷 검색을 수행한 결과, 한국직업정보 시스템에서 각 전공별 평균 연봉을 찾을 수 있었다. 간호사와는 달리, 의사는 근무연수에 따른 평균 연봉에 대한 자료를 찾을 수 없었다. 따라서 전공별 평균 연봉을 각 부서에 저장하고, 그 부서에 속한 의사의 연봉이 그 평균 연봉의 +-10% 내외에 있어야 한다고 설정하였다.



평균 연봉을 찾지 못한 과에 대해서는 임의로 평균연봉을 부여하였다.

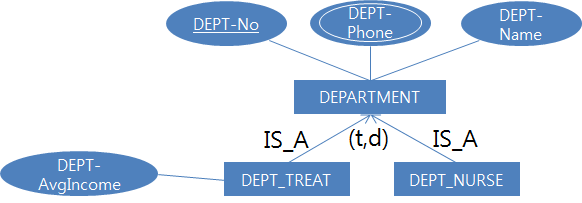
**2. ERD 작성**

**2-1) 전체 ERD**

****

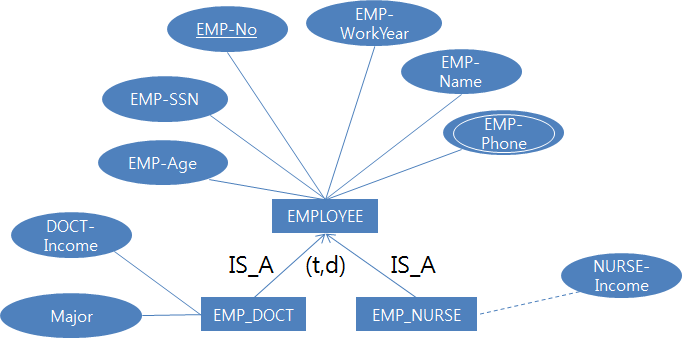
**2-2) Entities**

a) DEPARTMENT



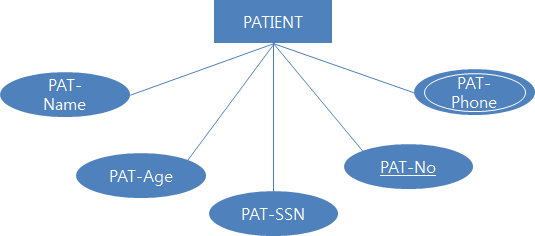
병원의 부서를 나타내는 DEPARTMENT이다. 부서들은 PK인 부서 번호, 부서 이름, multi-value 인 부서 전화번호를 가진다. sub class로는 진료부와 간호부가 있는데, 이들과 DEPARTMENT의 관계는 (total, disjoint)이다. 진료부는 추가로 평균 연봉이라는 속성을 가진다.

b) EMPLOYEE



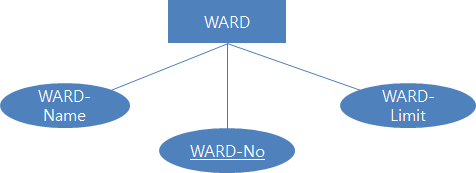
직원을 나타내는 EMPLOYREE이다. 직원은 PK인 사원번호와 주민번호, 이름, 나이, 근무연수, multi-value인 전화 번호를 가진다. sub class로는 의사와 간호사가 있으며, 이들과 EMPLOYEE의 관계는 (total, disjoint)이다. 의사는 추가로 의사연봉과 전공을 가지며, 간호사는 추가로 derived-value인 간호사 연봉을 가진다. 간호사의 연봉은 근무연수에 의해서 결정된다. 의사와 간호사 두 연봉을 나눈 이유는 의사는 주로 전공(과)에 의해서 연봉이 결정되고 간호사는 주로 근무연수에 의해 연봉이 결정되므로 두 값이 다른 성질을 가지고 있다고 생각하였기 때문이다.

c) PATIENT



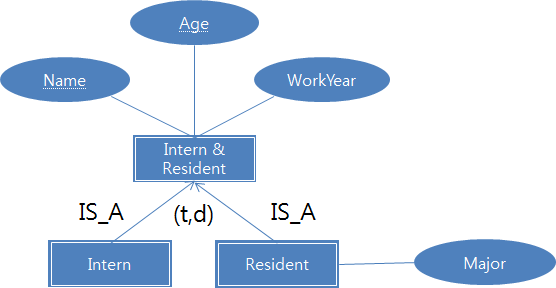
환자를 나타내는 PATIENT이다. 환자는 PK인 환자번호와 주민번호, 이름, 나이, multi-value인 전화 번호를 가진다.

d) WARD



병동을 나타내는 WARD이다. 병동은 PK인 병동번호와 병동이름, 병동의 최대 인원수를 가진다.

e) Intern & Resident

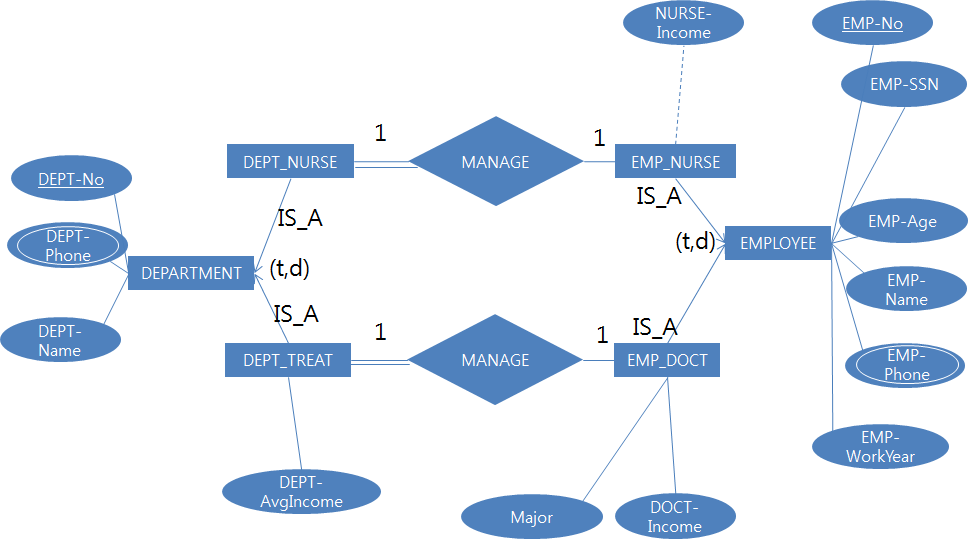


전공의를 나타내는 Intern & Resident이다. 전공의는 weak entity로서, partial key인 {이름, 나이}와 근무연수를 가진다. sub class로는 인턴과 레지던트가 있으며 이들과 Intern & Resident의 관계는 (total, disjoint)이다. 레지던트는 추가로 전공이라는 속성을 가진다.

Owner entity type은 의사(EMP\_DOCT)로서, EDUCATE라는 관계를 맺는다. Intern & Resident의 키는 partial key인 {이름, 나이}에 owner의 키인 사원번호를 합쳐서 {이름, 나이, 교육하는 의사의 사원번호} 가 된다.

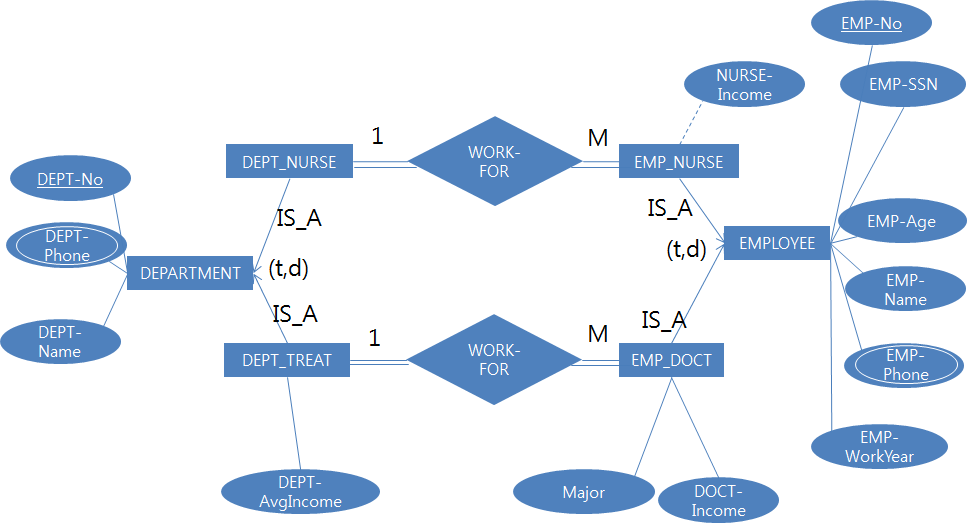
**2-3) Relationships**

a) MANAGE



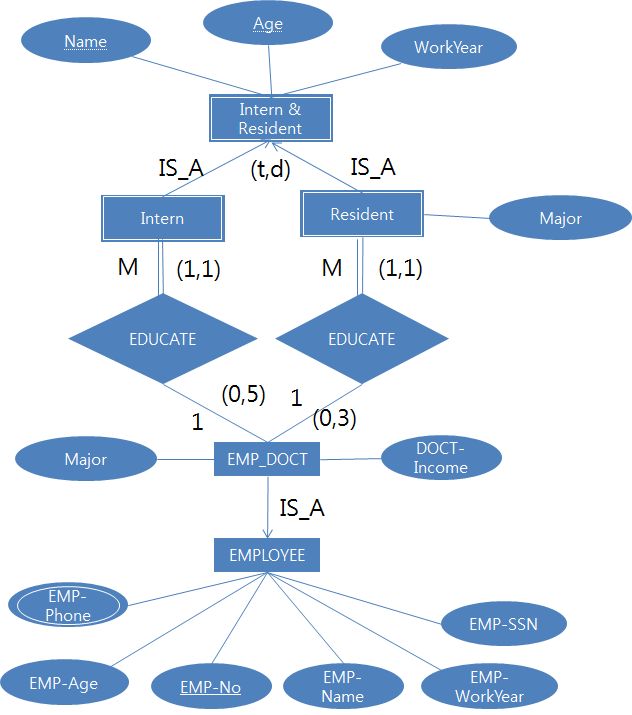
MANAGE는 직원(EMP\_NURSE, EMP\_DOCT)과 부서(DEPT\_TREAT, DEPT\_NURSE) 사이의 관계이다. MANAGE는 직원이 부서를 관리한다는 것을 나타낸다. 모든 부서에 관리자가 필요하지만 모든 직원이 관리자가 되지는 않기 때문에 total/partial 이다. 또한 부서의 관리자는 한 명이고, 한 직원은 한 부서의 관리자만 될 수 있기 때문에 1:1 이다.

b) WORK-FOR



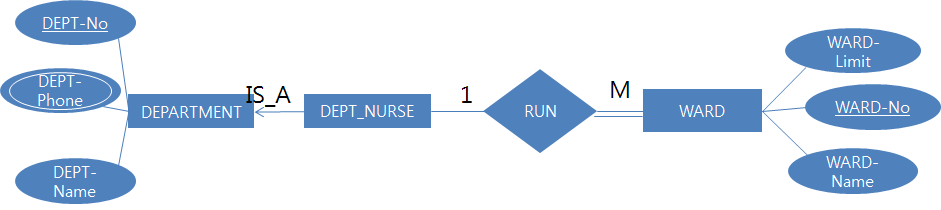
WORK-FOR는 직원(EMP\_NURSE, EMP\_DOCT)과 부서(DEPT\_TREAT, DEPT\_NURSE) 사이의 관계이다. WORK-FOR는 직원이 부서에서 근무한다는 것을 나타낸다. 모든 부서에 직원이 적어도 한 명은 근무해야 하고, 모든 직원은 부서에 근무해야 하기 때문에 total/total이다. 또한 한 부서에는 여러 직원이 있지만 한 직원은 한 부서에만 근무하므로 1:M이다.

c) EDUCATE



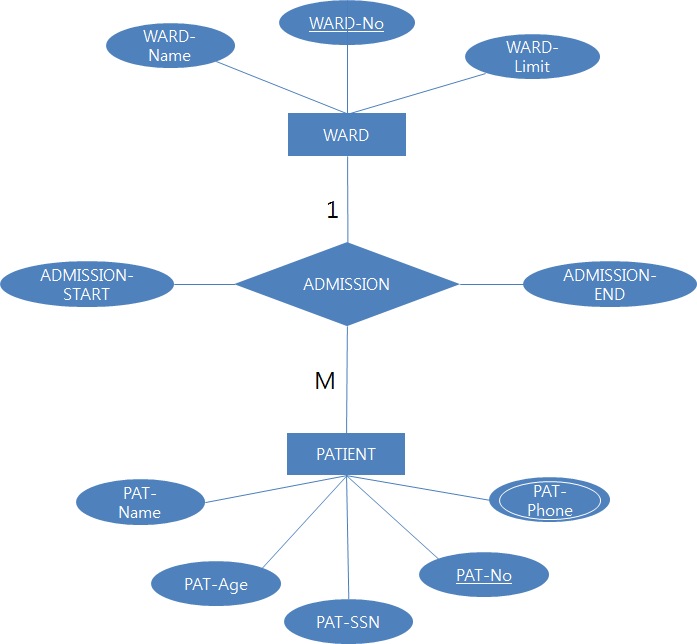
EDUCATE는 전공의(Intern, Resident)와 의사(EMP\_DOCT)의 관계이다. EDUCATE는 의사가 전공의를 교육한다는 것을 나타낸다. 모든 전공의는 의사에게 교육받아야 하지만 모든 의사가 전공의를 교육하지는 않기 때문에 total/partial이다. (Intern & Resident는 weak entity type이므로 owner인 EMP\_DOCT와의 관계에서 반드시 total이 되어야 한다.) 또한 전공의는 오직 한명의 의사로부터 교육받지만 의사는 최소 0명부터 최대 3명이나 5명을 교육할 수 있으므로 M:1 관계이다.

d) RUN



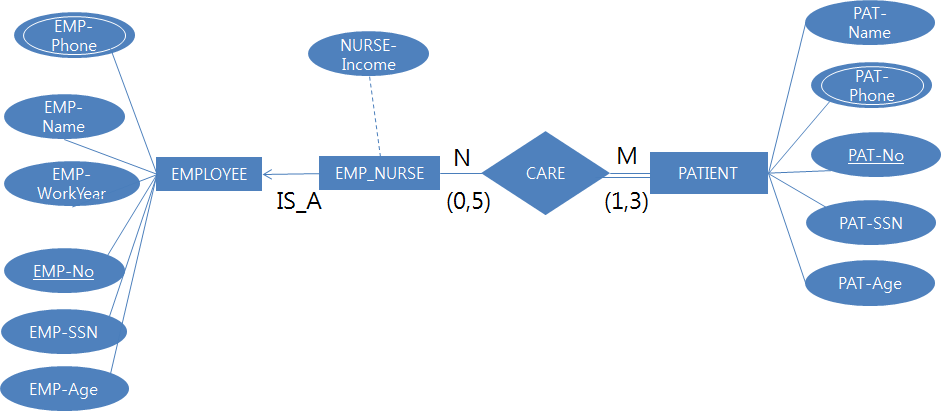
RUN은 간호부(DEPT\_NURSE)와 병동(WARD) 사이의 관계이다. RUN은 간호부가 병동을 운영한다는 것을 나타낸다. 모든 간호부가 병동을 운영하지는 않지만 모든 병동은 간호부로부터 관리 받아야 하기 때문에 partial/total이다. 또한 하나의 간호부는 여러 병동을 운영할 수 있지만 하나의 병동은 반드시 하나의 간호부로부터 관리 받아야 하기 때문에 1:M이다.

e) ADMISSION



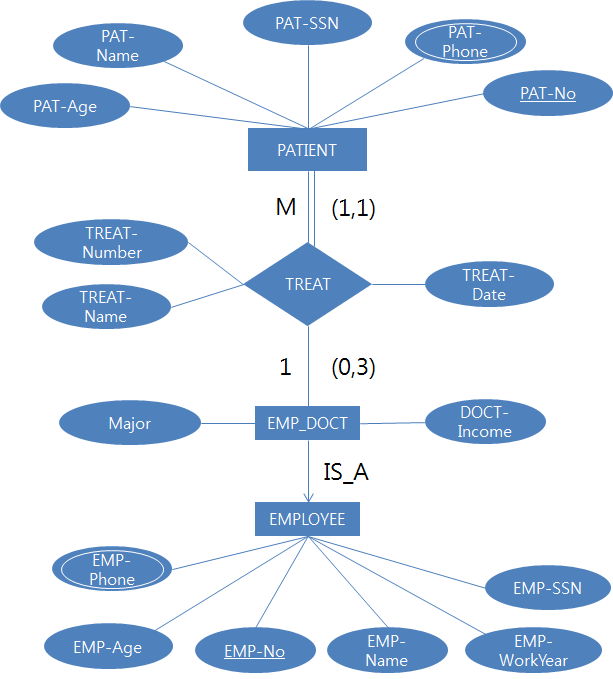
ADMISSION은 병동(WARD)와 환자(PATIENT) 사이의 관계이다. ADMISSION은 환자가 병동에 입원하는 것을 나타낸다. 모든 병동에 환자가 입원해야 하는 것은 아니고, 모든 환자가 병동에 입원해야 하는 것도 아니기 때문에 partial/partial 관계이다. 또한 한 병동은 여러 명의 환자를 수용할 수 있지만 한 환자는 한 병동에만 입원해야 하기 때문에 1:M이다.

f) CARE



CARE는 간호사(EMP\_NURSE)와 환자(PATIENT) 사이의 관계이다. CARE는 간호사가 환자를 담당하는 것을 나타낸다. 모든 간호사가 환자를 담당해야 하는 것은 아니지만 모든 환자에게는 간호사가 배정 되어야 하기 때문에 partial/total이다. 또한 한 명의 간호사가 여러 명의 환자를 담당할 수 있고 한 명의 환자는 여러 명의 간호사를 배정 받을 수 있기 때문에 N:M이다.

g) TREAT



TREAT는 환자(PATIENT)와 의사(EMP\_DOCT) 사이의 관계이다. TREAT는 환자가 의사로부터 진료받는 것을 나타낸다. 모든 의사가 환자를 진료해야 하는 것은 아니지만 모든 환자는 의사로부터 진료 받아야 하기 때문에 partial/total이다. 또한 한 의사는 여러 명의 환자를 진료할 수 있지만 한 환자는 오직 단 한 명의 의사로부터 진료받기 때문에 1:M 이다.

**3. Relational Schema 작성**

**3-1) Entities**

**a) DEPARTMENT**

DEPARTMENT는 DEPT\_TREAT와 DEPT\_NURSE로 나누어지는데, 이 관계가 total/disjoint 이므로 ERD의 IS\_A를 relational로 바꾸는 옵션 중 2번째인 option B를 사용하였다. 따라서 super class인 DEPARTMENT의 relation은 작성할 필요가 없고, 그 sub class인 DEPT\_TREAT와 DEPT\_NURSE의 relation만 만든다.

a-1) DEPT\_TREAT

DEPT\_TREAT (DEPT-No, DEPT-Name, DEPT-AvgIncome, Manager-No)

**Primary Key** : {DEPT-No}

**Foreign Key** : { Manager-No }

**Functional Dependencies**:

DEPT-No -> {DEPT-Name, DEPT-AvgIncome, Manager-No}

DEPT-Name -> {DEPT-No, DEPT-AvgIncome, Manager-No}

DEPT\_TREAT의 attribute 중 하나인 DEPT-Phone (전화번호)는 multi-value이므로 새로운 relation을 만들어준다. 따라서 DEPT-TREAT의 relation에는 포함되지 않는다.

DEPT\_TREAT는 EMP\_DOCT와 1:1인 total/partial 관계(MANAGE)를 가지고 있으므로, EMP\_DOCT의 PK인 EMP-No를 total쪽인 DEPT\_TREAT의 FK인 Manager-No 로 가져온다.

같은 부서 이름을 가지는 부서들이 없으므로, DEPT-Name는 키가 되어 DEPT-TREAT의 다른 attribute들인 {Dept-No, Dept-Income, Manager-No}의 값을 결정할 수 있다.

PK인 {DEPT-No}는 다른 attribute인 {DEPT-Name, DEPT-AvgIncome, Manager-No} 를 결정한다. 이후로 PK가 다른 attribute를 결정한다는 것은 당연하므로 설명하지 않도록 한다.

a-2) DEPT\_TREAT-Phone

DEPT\_TREAT -Phone (DEPT-No, Phone)

**Primary Key** : {DEPT-No, Phone}

**Foreign Key** : {DEPT-No}

**Functional Dependencies**:

{DEPT-No, Phone} -> {DEPT-No, Phone}

DEPT-Phone은 DEPT\_TREAT의 multi-value attribute이므로 따로 relational table을 만들어 주었다. FK인 DEPT-No는 DEPT\_TREAT의 PK인 DEPT-No를 참조한다.

a-3) DEPT\_NURSE

DEPT\_NURSE (DEPT-No, DEPT-Name, Manager-No)

**Primary Key** : {DEPT-No}

**Foreign Key** : {Manager-No }

**Functional Dependencies**:

DEPT-No -> {DEPT-Name, Manager-No}

DEPT-Name -> {DEPT-No, Manager-No}

DEPT\_NURSE의 attribute 중 하나인 DEPT-Phone (전화번호)는 multi-value이므로 새로운 relation을 만들어준다. 따라서 DEPT-NURSE의 relation에는 포함되지 않는다.

DEPT\_NURSE는 EMP\_NURSE와 1:1인 total/partial 관계(MANAGE)를 가지고 있으므로, EMP\_NURSE의 PK인 EMP-No를 total쪽인 DEPT\_NURSE의 FK인 Manager-No로 가져온다.

같은 부서 이름을 가지는 부서들이 없으므로, DEPT-Name는 키가 되어 DEPT-NURSE의 다른 attribute들인 {Dept-No, Manager-No}의 값을 결정할 수 있다.

a-4) DEPT\_NURSE-Phone

DEPT\_ NURSE -Phone (DEPT-No, Phone)

**Primary Key** : {DEPT-No, Phone}

**Foreign Key** : {DEPT-No}

**Functional Dependencies**:

{DEPT-No, Phone} -> {DEPT-No, Phone}

DEPT-Phone은 DEPT\_ NURSE 의 multi-value attribute이므로 따로 relational table을 만들어 주었다. FK인 DEPT-No는 DEPT\_NURSE 의 PK인 DEPT-No를 참조한다.

**b) EMPLOYEE**

EMPLOYEE 는 EMP\_DOCT와 EMP\_NURSE로 나누어지는데, 이 관계가 total/disjoint 이므로 ERD의 IS\_A를 relational로 바꾸는 옵션 중 2번째인 option B를 사용하였다. 따라서 super class인 EMPLOYEE 의 relation은 작성할 필요가 없고, 그 sub class인 EMP\_DOCT와 EMP\_NURSE의 relation만 만든다.

b-1) EMP\_DOCT

EMP\_DOCT (EMP-No, EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, Major, DEPT-No)

**Primary Key** : {EMP-No}

**Foreign Key** : {DEPT-No }

**Functional Dependencies**:

{EMP-No} -> {EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, Major, DEPT-No}

{EMP-SSN} -> {EMP-No, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, Major, DEPT-No}

{Major} -> {DEPT-No}

{DEPT-No} -> {Major}

EMP\_DOCT의 attribute 중 하나인 EMP-Phone (전화번호)는 multi-value이므로 새로운 relation을 만들어준다. 따라서 EMP\_DOCT 의 relation에는 포함되지 않는다.

EMP\_DOCT는 DEPT\_TREAT와 M:1인 total/total 관계(WORK-FOR)를 가지고 있으므로, DEPT\_TREAT의 PK인 DEPT-No를 M쪽인 EMP\_DOCT의 FK인 DEPT-No로 가져온다.

같은 EMP-SSN (주민번호)를 가지는 의사들이 없으므로, EMP-SSN은 키로서 EMP\_DOCT의 다른 attribute들인 {EMP-No, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, Major, DEPT-No}의 값들을 결정한다.

의사의 전공은 의사가 속한 부서의 이름과 같아야 하고, 부서의 이름이 같은 부서들은 없으므로(부서의 이름이 키이므로) 의사의 전공은 의사가 속한 부서의 번호를 결정하게 된다. (역도 성립)

b-2) EMP\_DOCT-Phone

EMP\_DOCT-Phone (EMP-No, Phone)

**Primary Key** : {EMP-No, Phone}

**Foreign Key** : {EMP-No}

**Functional Dependencies**:

{EMP-No, Phone} -> {EMP-No, Phone}

EMP-Phone은 EMP\_DOCT 의 multi-value attribute이므로 따로 relational table을 만들어 주었다. FK인 EMP-No는 EMP\_DOCT 의 PK인 EMP-No를 참조한다.

b-3) EMP\_NURSE

EMP\_NURSE (EMP-No, EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, NURSE-Income, DEPT-No)

**Primary Key** : {EMP-No}

**Foreign Key** : {DEPT-No }

**Functional Dependencies**:

{EMP-No} -> {EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, NURSE-Income, DEPT-No}

{EMP-SSN} -> {EMP-No, EMP-name, EMP-age, EMP-WorkYear, NURSE-Income, DEPT-No}

{EMP-WorkYear} -> {NURSE-Income}

{NURSE-Income} -> {EMP-WorkYear}

EMP\_ NURSE의 attribute 중 하나인 EMP-Phone (전화번호)는 multi-value이므로 새로운 relation을 만들어준다. 따라서 EMP\_ NURSE의 relation에는 포함되지 않는다.

EMP\_ NURSE는 DEPT\_ NURSE와 M:1인 total/total 관계(WORK-FOR)를 가지고 있으므로, DEPT\_ NURSE의 PK인 DEPT-No를 M쪽인 EMP\_ NURSE의 FK인 DEPT-No로 가져온다.

같은 EMP-SSN (주민번호)를 가지는 간호사들이 없으므로, EMP-SSN은 키로서 EMP\_ NURSE의 다른 attribute들인 {EMP-No, EMP-name, EMP-age, EMP-WorkYear, NURSE-Income, DEPT-No}의 값들을 결정한다.

NURSE-Income은 EMP-WorkYear에 의해 결정되는 derived attribute이므로 (NURSE-Income = 2680 + EMP-WorkYear \* 100), functional dependency를 가진다. 반대로 EMP-WorkYear을 통해 NURSE-Income을 구할 수도 있으므로 역도 성립한다.

b-4) EMP\_ NURSE-Phone

EMP\_ NURSE-Phone (EMP-No, Phone)

**Primary Key** : {EMP-No, Phone}

**Foreign Key** : {EMP-No}

**Functional Dependencies**:

{EMP-No, Phone} -> {EMP-No, Phone}

EMP-Phone은 EMP\_ NURSE의 multi-value attribute이므로 따로 relational table을 만들어 주었다. FK인 EMP-No는 EMP\_ NURSE의 PK인 EMP-No를 참조한다.

c) WARD

WARD (WARD-No, WARD-Name, WARD-Limit, DEPT-No)

**Primary Key** : {WARD-No}

**Foreign Key** : {DEPT-No}

**Functional Dependencies**:

{WARD-No} -> {WARD-Name, WARD-Limit, DEPT-No}

{WARD- Name} -> {WARD-No, WARD-Limit, DEPT-No}

WARD (병동)은 DEPT\_NURSE와 M:1 (total, partial) 관계 (RUN)을 가지므로, DEPT\_NURSE의 PK인 DEPT-No를 WARD의 FK로 가져와 참조한다.

병동 이름이 같은 병동들은 없으므로, WARD-Name는 키로서 다른 attribute들인 {WARD-No, WARD-Limit, DEPT-No} 을 결정한다.

d) PATIENT

PATIENT (PAT-No, PAT-SSN, PAT-Age, PAT-Name, DOCT-No)

**Primary Key** : {PAT-No}

**Foreign Key** : {DOCT-No}

**Functional Dependencies**:

{PAT-No} -> {PAT-SSN, PAT-Age, PAT-Name, DOCT-No}

{PAT- SSN} -> {PAT-No, PAT-Age, PAT-Name, DOCT-No}

PATIENT의 attribute 중 하나인 PATIENT-Phone (전화번호)는 multi-value이므로 새로운 relation을 만들어준다. 따라서 PATIENT의 relation에는 포함되지 않는다.

PATIENT는 EMP\_DOCT와 M:1 (total, partial) 관계 (TREAT)를 가지므로 EMP\_DOCT의 PK인 EMP-No를 PATIENT의 FK인 DOCT-No로 가져와서 참조한다.

주민번호가 같은 환자들은 없으므로, SSN은 키로서 다른 attribute들인 {PAT-No, PAT-Age, PAT-Name}를 결정한다.

d-2) PATIENT-Phone

PATIENT -Phone (PAT-No, Phone)

**Primary Key** : {PAT-No, Phone}

**Foreign Key** : {PAT-No}

**Functional Dependencies**:

{PAT-No, Phone} -> {PAT-No, Phone}

PAT-Phone은 PATIENT의 multi-value attribute이므로 따로 relational table을 만들어 주었다. FK인 PAT-No는 PATIENT의 PK인 PAT-No를 참조한다.

e) Intern & Resident

Intern & Resident는 Intern과 Resident로 나누어지는데, 이 관계가 total/disjoint 이므로 ERD의 IS\_A를 relational로 바꾸는 옵션 중 2번째인 option B를 사용하였다. 따라서 super class인 Intern & Resident 의 relation은 작성할 필요가 없고, 그 sub class인 Intern와 Resident의 relation만 만든다.

e-1) Intern

Intern (Name, Age, DOCT-No, WorkYear)

**Primary Key** : {Name, Age, DOCT-No}

**Foreign Key** : {DOCT-No}

**Functional Dependencies**:

{Name, Age, DOCT-No} -> {WorkYear}

Intern은 weak entity type이므로 primary key는 Intern의 partial key와 owner type인 EMP\_DOCT의 primary key의 합이다. 따라서 EMP\_DOCT의 PK인 EMP-No를 Intern의 FK인 DOCT-No로 가져와 참조한다. Intern의 PK는 partial key {Name, Age}와 FK인 {DOCT-No}를 합친 {Name, Age, DOCT-No} 가 된다.

e-2) Resident

Resident (Name, Age, DOCT-No, WorkYear, Major)

**Primary Key** : {Name, Age, DOCT-No}

**Foreign Key** : {DOCT-No}

**Functional Dependencies**:

{Name, Age, DOCT-No} -> {WorkYear, Major}

Intern은 weak entity type이므로 primary key는 Intern의 partial key와 owner type인 EMP\_DOCT의 primary key의 합이다. 따라서 EMP\_DOCT의 PK인 EMP-No를 Intern의 FK인 DOCT-No로 가져와 참조한다. Intern의 PK는 partial key {Name, Age}와 FK인 {DOCT-No}를 합친 {Name, Age, DOCT-No} 가 된다.

**3-2) Relationships**

a) ADMISSION

ADMISSION (PAT-No, WARD-No, ADMISSION-START, ADMISSION-END)

**Primary Key** : {PAT-No}

**Foreign Key** : {PAT-No}, {WARD-No}

**Functional Dependencies**:

{PAT-No} -> {WARD-No, ADMISSION-START, ADMISSION-END}

PATIENT와 WARD의 관계인 ADMISSION은 M:1 (partial, partial) 이므로 새로운 relation으로 만든다. PATIENT와 WARD의 PK를 참조하여 FK로 가져온다. ADMISSION의 PK는 M쪽인 PATIENT의 PK에서 가져온 FK로, 각 환자가 오직 하나의 병동에만 입원하기 때문에 키가 될 수 있다.

FK인 PAT-No는 WARD-No를 결정하지만, PAT-No가 이미 PK이므로 functional dependency로 따로 명시하지 않는다.

b) CARE

CARE (NURSE-No, PAT-No)

**Primary Key** : {NURSE-No, PAT-No}

**Foreign Key** : {NURSE-No}, {PAT-No}

**Functional Dependencies**:

{NURSE-No, PAT-No} -> {NURSE-No, PAT-No}

EMP\_NURSE와 PATIENT의 관계인 CARE는 N:M (partial, total)이므로 새로운 relation으로 만든다. EMP\_NURSE와 PATIENT의 PK를 참조하여 FK로 가져오고, 그 둘을 합쳐서 CARE의 PK로 한다. 이는 N:M 관계이므로 어느 한 FK가 키가 될 수 없기 때문이다.

c) TREAT

TREAT (PAT-No, TREAT-Number, DOCT-No, TREAT-Name, TREAT-Date)

**Primary Key** : {PAT-No, TREAT-Number}

**Foreign Key** : {PAT-No}, {DOCT-No}

**Functional Dependencies**:

{PAT-No, TREAT-Number} -> {DOCT-No, TREAT-Name, TREAT-Date}

{PAT-No} -> {DOCT-No}

EMP\_DOCT와 PATIENT의 관계인 TREAT는 1:M (partial, total)이므로 새로운 relation으로 만들 필요는 없지만, 진료가 여러 번 행해지고 그 진료를 구분하는 데에는 진료번호가 필요하므로 따로 relation으로 만든다.

EMP\_DOCT에서 PK인 EMP-No를, PATIENT에서 PK인 PAT-No를 각각 DOCT-No와 PAT-No로 가져와 참조한다. TREAT의 TREAT-Number는 환자당 진료 번호이므로 환자들을 구분 할 수 없어 키가 되지 못하고 PATIENT에서 가져온 FK PAT-No와 TREAT-Number를 합쳐서 PK로 한다. EMP\_DOCT에서 가져온 FK DOCT-No는 PK에 속하지 않는데, 이는 의사(EMP\_DOCT)와 환자(PATIENT)의 관계가 1:M이므로 PAT-No가 DOCT-No를 결정하기 때문이다.

PAT-No가 DOCT-No를 결정하므로, (1:M 관계) functional dependency로 나타낸다.

**4. Normalization**

Normalization 단계에서는 모든 relation들이 BCNF가 되는 것을 목적으로 한다. 또한 이를 통해 데이터베이스의 중복과 이로 인한 이상을 없애는 것을 목적으로 한다. 이상은 아래와 같은 종류들이 있다.

Redundancy: 같은 정보가 불필요하게 반복해서 나타난다.

Insert Anomaly: 새로운 tuple을 삽입할 수 없다.

Delete Anomaly: 삭제할 때, 원하지 않았던 정보들이 같이 지워질 수 있다.

Update Anomaly: 정보가 업데이트 될 때, 여러 번 업데이트 해야 한다.

아래는 문제가 있는 relation들이다. 각각의 relation이 왜 문제가 있는지, 어떤 문제가 있는지, 어떻게 수정하는지 설명하고, 마지막에 모든 relation을 확인해보도록 하겠다.

**1) First Normal Form (1NF)**

이미 모든 relation 안의 tuple들이 PK에 의해 구분되고, 각각의 attribute 들이 atomic value를 가지도록 multi-valued attribute들을 다른 relation으로 분리하였으므로, 모든 relation은 1NF이다.

**2) Second Normal Form (2NF)**

**1. TREAT**

TREAT (PAT-No, TREAT-Number, DOCT-No, TREAT-Name, TREAT-Date)

**Primary Key** : {PAT-No, TREAT-Number}

**Foreign Key** : {PAT-No}, {DOCT-No}

**Functional Dependencies**:

{PAT-No, TREAT-Number} -> {DOCT-No, TREAT-Name, TREAT-Date}

{PAT-No} -> {DOCT-No}

TREAT relation에서 PK의 일부인 PAT-No가 키의 일부가 아닌 DOCT-No를 결정하므로, 이는 2NF가 아니라고 볼 수 있다. 이 경우, redundancy의 문제가 발생한다. 진료(TREAT)는 여러 번 반복되는데, 이때마다 동일한 환자번호에 대해서 동일한 사원(의사)번호가 나타난다.

Insert Anomaly는 발생하지 않는다. TREAT는 환자가 의사로부터 받은 진료를 나타내기 때문에 새로운 tuple이 TREAT에 삽입될 때 (즉, 환자가 의사로부터 진료를 받은 것이 저장될 때) 환자에 대한 정보인 PAT-No와 진료에 대한 정보인 TREAT-Number는 항상 존재할 것이다.

Delete Anomaly는 발생하지 않는다. tuple의 정보가 지워진다는 것은 어느 특정한 진료의 기록이 삭제된다는 것인데, 이 경우 진료의 이름이나 번호, 날짜가 지워진다는 것은 당연하다. 환자번호나 의사번호는 PATIENT와 EMP\_DOCT table에 저장되어 있으므로 tuple이 삭제되어도 문제가 없다.

Update Anomaly는 발생하지 않는다. PAT-No나 DOCT-No가 바뀌어도 환자와 의사에 대한 정보는 PATIENT와 EMP\_DOCT table에 저장되어 있으므로 따로 update해주어야 할 것이 없다. TREAT-Number나 TREAT-Name, TREAT-Date가 update 된다면 그 tuple에 대해서만 해주면 되며, 다른 tuple에 update할 필요가 없다.

2NF를 위반하고 있는 FD는 {PAT-No} -> {DOCT-No}이다. 따라서 이들에 대해서 따로 테이블을 만들고, PAT-No에 의해 결정되는 DOCT-No는 이 relation에서 제거한다.

relation을 나누어 보면 아래와 같다.

TREAT (PAT-No, TREAT-Number, TREAT-Name, TREAT-Date)

**Primary Key** : {PAT-No, TREAT-Number}

**Foreign Key** : {PAT-No}

**Functional Dependencies**:

{PAT-No, TREAT-Number} -> {TREAT-Name, TREAT-Date}

PAT-DOCTOR (PAT-No, DOCT-No)

**Primary Key** : {PAT-No }

**Foreign Key** : {PAT-No, DOCT-No}

**Functional Dependencies**:

{PAT-No} -> {DOCT-No}

만들어진 relation을 보면 PATIENT relation의 일부라는 것을 알 수 있으며, 따라서 삭제한다. TREAT relation에는 DOCT-No가 필요 없었는데 불필요하게 들어갔었다는 것을 알 수 있다.

이외의 relation들은 이미 모두 2NF이다.

**3) Third Normal Form (3NF)**

**1. EMP\_DOCT**

EMP\_DOCT (EMP-No, EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, Major, DEPT-No)

**Primary Key** : {EMP-No}

**Foreign Key** : {DEPT-No }

**Functional Dependencies**:

{EMP-No} -> {EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, Major, DEPT-No}

{EMP-SSN} -> {EMP-No, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, Major, DEPT-No}

{Major} -> {DEPT-No}

{DEPT-No} -> {Major}

{Major} -> {DEPT-No}와 {DEPT-No} -> {Major}을 보면 {Major}과 {DEPT-No}는 super key도 아니고 prime attribute도 아니므로 3NF가 아님을 알 수 있다.

이 경우, redundancy의 문제가 발생한다. 동일한 전공을 가지는 모든 의사들이 동일한 부서 번호를 가지고, 동일한 부서 번호를 가지는 모든 의사들이 동일한 전공을 가지므로 정보가 반복된다.

Insert Anomaly는 발생하지 않는다. 새로운 의사에 대한 정보가 삽입될 때, 모든 의사들은 부서에 속해있으므로 DEPT-No 값은 항상 존재한다. 따라서 Major 값도 항상 존재하게 된다.

Delete Anomaly는 발생하지 않는다. tuple의 정보가 지워진다는 것은 어느 특정한 의사의 기록이 삭제된다는 것인데, 이 경우 그 의사의 사원번호, 전공이나 부서번호에 대한 정보가 지워진다는 것은 당연하다. DEPT-No의 경우 정보가 지워져도 DEPT\_DOCT table에 정보가 저장되어 있으므로 상관없다.

Update Anomaly는 발생하지 않는다. 전공이나 부서 번호에 대한 정보가 바뀌더라도 그 tuple에서만 update해주면 되며, 다른 tuple의 경우에는 관계없기 때문에 (각각의 의사마다의 값이므로) update해줄 필요가 없다.

3NF를 위반하고 있는 FD는 {Major} -> {DEPT-No}와 {DEPT-No} -> {Major}이다. 이들에 대해서 따로 테이블을 만들고 DEPT-No가 결정하는 Major는 이 relation에서 제거한다. (DEPT-No는 FK이므로 Major을 제거)

3NF로 바꾸기 위해 relation을 나누어 보면 아래와 같다.

EMP\_DOCT (EMP-No, EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, DEPT-No)

**Primary Key** : {EMP-No}

**Foreign Key** : {DEPT-No }

**Functional Dependencies**:

{EMP-No} -> {EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, DEPT-No}

{EMP-SSN} -> {EMP-No, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, DEPT-No}

DEPT-Major (DEPT-No, Major)

**Primary Key** : {DEPT-No}

**Foreign Key** : {DEPT-No }

**Functional Dependencies**:

{DEPT-No} -> {Major}

새로 만들어진 이 relation을 보면, Major는 부서의 이름과 같으므로 DEPT\_DOCT relation의 일부라는 것을 알 수 있다. 따라서 제거한다. EMP\_DOCT에는 Major가 필요 없었는데 불필요하게 들어갔다는 것을 알 수 있다.

**2. EMP\_NURSE**

EMP\_NURSE (EMP-No, EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, NURSE-Income, DEPT-No)

**Primary Key** : {EMP-No}

**Foreign Key** : {DEPT-No }

**Functional Dependencies**:

{EMP-No} -> {EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, NURSE-Income, DEPT-No}

{EMP-SSN} -> {EMP-No, EMP-name, EMP-age, EMP-WorkYear, NURSE-Income, DEPT-No}

{EMP-WorkYear} -> {NURSE-Income}

{NURSE-Income} -> {EMP-WorkYear}

{EMP-WorkYear} -> {NURSE-Income}와 {NURSE-Income} -> {EMP-WorkYear}을 보면 {EMP-WorkYear}와 {NURSE-Income}은 super key도 아니고 prime attribute도 아니므로 3NF가 아님을 알 수 있다.

이 경우, redundancy의 문제가 발생한다. 동일한 EMP-WorkYear을 가지는 모든 간호사들에 대해서 NURSE-Income은 동일하고, 동일한 NURSE-Income을 가지는 모든 간호사들에 대해서 EMP-WorkYear가 동일하기 때문에 정보가 반복된다.

Insert Anomaly는 발생하지 않는다. 새로운 간호사에 대한 정보가 삽입될 때, 그 간호사의 사원번호에 대한 정보는 당연히 존재할 것이므로 insert하는데에 문제가 없다.

Delete Anomaly는 발생하지 않는다. tuple의 정보가 지워진다는 것은 어느 특정한 간호사의 기록이 삭제된다는 것인데, 이 경우 그 간호사의 사원번호, 근무연수나 연봉이 지워지는 것은 당연하다. DEPT-No의 경우 정보가 지워져도 DEPT\_DOCT table에 정보가 저장되어 있으므로 상관없다.

Update Anomaly는 발생하지 않는다. 근무연수나 연봉에 대한 정보가 바뀌더라도 그 tuple에서만 update해주면 되며, 다른 tuple의 경우에는 관계없기 때문에 (각각의 간호사마다의 값이므로) update해줄 필요가 없다.

3NF를 위반하고 있는 FD는 {EMP-WorkYear} -> {NURSE-Income}와 {NURSE-Income} -> {EMP-WorkYear}이다. 이들에 대해서 따로 테이블을 만들고 EMP-WorkYear가 결정하는 NURSE-Income은 이 relation에서 제거한다. (EMP\_WorkYear가 NURSE-Income을 결정하는 게 더 명확하므로 NURSE-Income을 제거)

3NF가 되도록 relation을 나누어 보면 아래와 같다.

EMP\_NURSE (EMP-No, EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, DEPT-No)

**Primary Key** : {EMP-No}

**Foreign Key** : {DEPT-No }, {EMP-WorkYear}

**Functional Dependencies**:

{EMP-No} -> {EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, DEPT-No}

{EMP-SSN} -> {EMP-No, EMP-name, EMP-age, EMP-WorkYear, DEPT-No}

NURSE-Income (NURSE-WorkYear, Income)

**Primary Key** : {NURSE-WorkYear}

**Foreign Key** :

**Functional Dependencies**:

{NURSE-WorkYear} -> Income

이외에 모든 relation은 이미 모두 3NF이다.

**4) Boyce Codd Normal Form (BCNF)**

모든 relation은 이미 모두 BCNF이다. (FD에서 좌측이 모두 키이다)

**5) 확인**

DEPT\_TREAT (DEPT-No, DEPT-Name, DEPT-AvgIncome, Manager-No)

**Functional Dependencies**:

DEPT-No -> {DEPT-Name, DEPT-AvgIncome, Manager-No}

DEPT-Name -> {DEPT-No, DEPT-AvgIncome, Manager-No}

DEPT-No와 DEPT-Name는 모두 Super-key이므로 BCNF를 만족한다.

DEPT\_TREAT -Phone (DEPT-No, Phone)

**Functional Dependencies**:

{DEPT-No, Phone} -> {DEPT-No, Phone}

{DEPT-No, Phone}는 Super-key이므로 BCNF를 만족한다.

DEPT\_NURSE (DEPT-No, DEPT-Name, Manager-No)

**Functional Dependencies**:

DEPT-No -> {DEPT-Name, Manager-No}

DEPT-Name -> {DEPT-No, Manager-No}

DEPT-No와 DEPT-Name은 모두 Super-key이므로 BCNF를 만족한다.

DEPT\_ NURSE -Phone (DEPT-No, Phone)

**Functional Dependencies**:

{DEPT-No, Phone} -> {DEPT-No, Phone}

{DEPT-No, Phone}는 Super-Key이므로 BCNF를 만족한다.

EMP\_DOCT-Phone (EMP-No, Phone)

**Functional Dependencies**:

{EMP-No, Phone} -> {EMP-No, Phone}

{EMP-No, Phone}는 Super-Key이므로 BCNF를 만족한다.

EMP\_ NURSE-Phone (EMP-No, Phone)

**Functional Dependencies**:

{EMP-No, Phone} -> {EMP-No, Phone}

{EMP-No, Phone}는 Super-Key이므로 BCNF를 만족한다.

WARD (WARD-No, WARD-Name, WARD-Limit, DEPT-No)

**Functional Dependencies**:

{WARD-No} -> {WARD-Name, WARD-Limit, DEPT-No}

{WARD- Name} -> {WARD-No, WARD-Limit, DEPT-No}

WARD-No, WARD-Name 모두 Super-Key이므로 BCNF를 만족한다.

PATIENT (PAT-No, PAT-SSN, PAT-Age, PAT-Name, DOCT-No)

**Functional Dependencies**:

{PAT-No} -> {PAT-SSN, PAT-Age, PAT-Name, DOCT-No}

{PAT- SSN} -> {PAT-No, PAT-Age, PAT-Name, DOCT-No}

PAT-No, PAT-SSN 모두 Super-Key이므로 BCNF를 만족한다.

PATIENT -Phone (PAT-No, Phone)

**Functional Dependencies**:

{PAT-No, Phone} -> {PAT-No, Phone}

{PAT-No, Phone}는 Super-key이므로 BCNF를 만족한다.

Intern (Name, Age, DOCT-No, WorkYear)

**Functional Dependencies**:

{Name, Age, DOCT-No} -> {WorkYear}

{Name, Age, DOCT-No}는 Super-key이므로 BCNF를 만족한다.

Resident (Name, Age, DOCT-No, WorkYear, Major)

**Functional Dependencies**:

{Name, Age, DOCT-No} -> {WorkYear, Major}

{Name, Age, Doct-No}는 Super-key이므로 BCNF를 만족한다.

ADMISSION (PAT-No, WARD-No, ADMISSION-START, ADMISSION-END)

**Functional Dependencies**:

{PAT-No} -> {WARD-No, ADMISSION-START, ADMISSION-END}

PAT-No는 Super-key이므로 BCNF를 만족한다.

CARE (NURSE-No, PAT-No)

**Functional Dependencies**:

{NURSE-No, PAT-No} -> {NURSE-No, PAT-No}

{NURSE-No, PAT-No}는 Super-key이므로 BCNF를 만족한다.

TREAT (PAT-No, TREAT-Number, TREAT-Name, TREAT-Date)

**Functional Dependencies**:

{PAT-No, TREAT-Number} -> {TREAT-Name, TREAT-Date}

{PAT-No, TREAT-Number}은 Super-key이므로 BCNF를 만족한다.

EMP\_DOCT (EMP-No, EMP-SSN, EMP-Name, EMP-Age, EMP-WorkYear, DOCT-Income, DEPT-No)

**Functional Dependencies**:

{EMP-No} -> {EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, DEPT-No}

{EMP-SSN} -> {EMP-No, EMP-name, EMP-age, EMP-WorkYear, DOCT-Income, DEPT-No}

EMP-No와 EMP-SSN은 모두 Super-Key이므로 BCNF를 만족한다.

EMP\_NURSE (EMP-No, EMP-SSN, EMP-Name, EMP-Age, EMP-WorkYear, DEPT-No)

**Functional Dependencies**:

{EMP-No} -> {EMP-SSN, EMP-name, EMP-age, EMP-WorkYear, DEPT-No}

{EMP-SSN} -> {EMP-No, EMP-name, EMP-age, EMP-WorkYear, DEPT-No}

EMP-No와 EMP-SSN은 모두 Super-Key이므로 BCNF를 만족한다.

NURSE-Income (NURSE-WorkYear, Income)

**Functional Dependencies**:

{NURSE-WorkYear} -> Income

NURSE-WorkYear은 Super-Key이므로 BCNF를 만족한다.

**5. SQL DDL 명령문 작성**

콘솔 창에 명령어를 하나하나 치기는 힘들기 때문에, Notepad++ 라는 프로그램에 미리 명령어를 짜고 MySQL에서 source 명령어를 사용해 불러오는 방식을 이용하였다.

**1) DATABASE 생성**

****

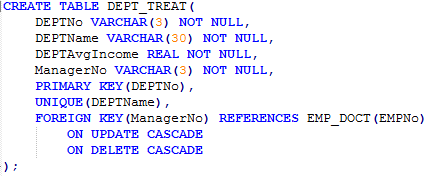
create database 명령어를 이용해 데이터베이스를 만들고 use 명령어를 이용해 사용할 데이터베이스를 지정하였다.

**2) BASE TABLE 생성**

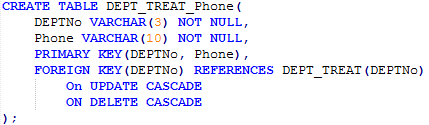
바로 FK 제약조건이 있는 table을 작성하면 오류가 나기에 (참조되는 table이 미리 만들어져 있어야 한다) set foreign\_key\_checks 명령어를 이용해 잠시 FK 제약조건의 검사를 멈추고 table을 작성한 뒤, 나중에 다시 검사를 하도록 했다.



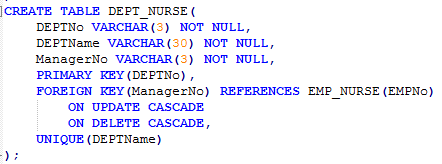
**a) DEPT\_TREAT**



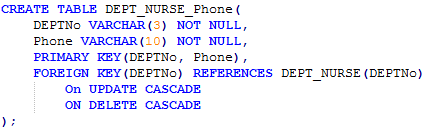
**b) DEPT\_TREAT\_Phone**



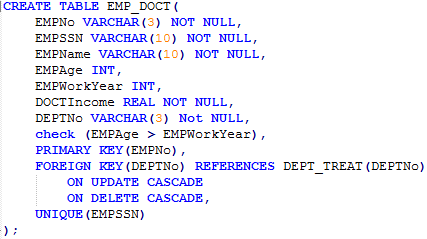
**c) DEPT\_NURSE**



**d) DEPT\_NURSE\_Phone**

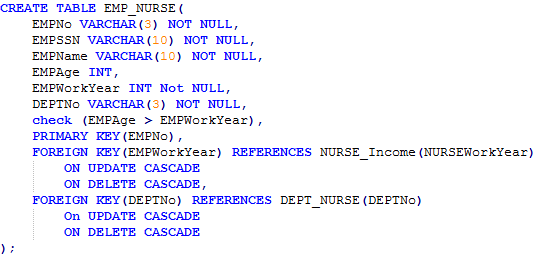


**e) EMP\_DOCT**



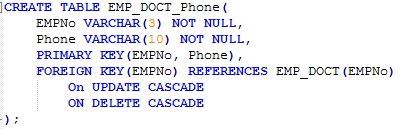
사원의 근무연수는 사원의 나이를 당연히 넘을 수 없으므로, 이를 확인하게 하였다.

**f) EMP\_NURSE**

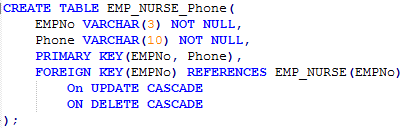


사원의 근무연수는 사원의 나이를 당연히 넘을 수 없으므로, 이를 확인하게 하였다.

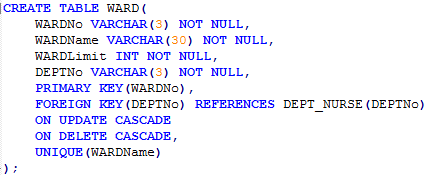
**g) EMP\_DOCT\_Phone**



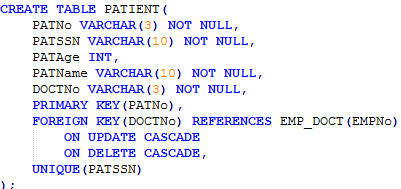
**h) EMP\_NURSE\_Phone**



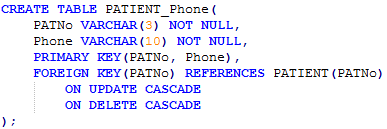
**i) WARD**



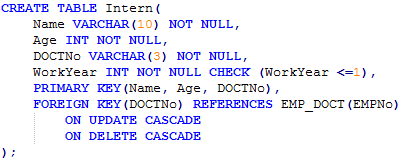
**j) PATIENT**



**k) PATIENT\_Phone**

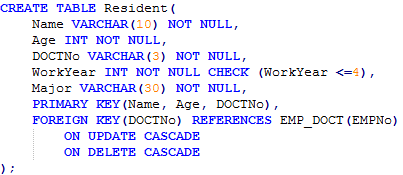


**l) Intern**



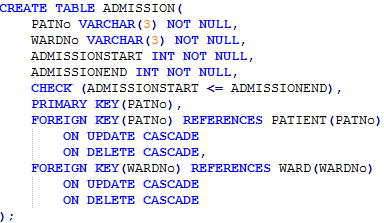
인턴의 근무연수는 1년 이하여야 하므로, 이를 확인하게 하였다.

**m) Resident**



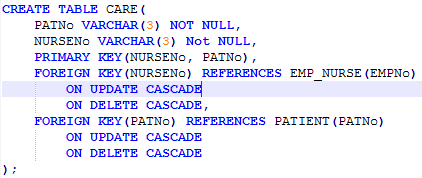
레지던트의 근무연수는 4년 이하여야 하므로, 이를 확인하게 하였다.

**n) ADMISSION**

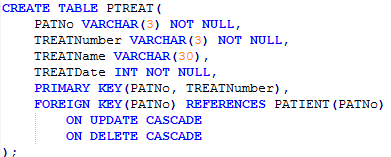


환자가 병동에 입원한 날짜가 퇴원한 날짜 이전이어야 하므로, 이를 확인하게 하였다.

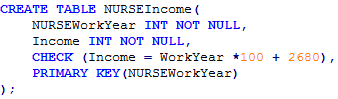
**o) CARE**



**p) PTREAT**



**q) NURSEIncome**



간호사의 연봉은 근무연수\*100 + 2680으로 계산되므로, 이를 확인하게 하였다.

**3) View 생성**

**1. DEPT\_TREATwithPhone**

DEPT\_TREAT에 속하는 부서들에 대한 정보와 그 부서들의 전화번호들을 한번에 볼 수 있도록 모아놓은 View이다.



**2.** **DEPT\_NURSEwithPhone**

DEPT\_NURSE에 속하는 부서들에 대한 정보와 그 부서들의 전화번호들을 한번에 볼 수 있도록 모아놓은 View이다.



**3. PATIENTatWARD**

환자들 중 입원한 환자들의 정보와 입원에 대한 정보(입원 병동, 입원 날짜, 퇴원 날짜)를 한번에 볼 수 있도록 모아놓은 View이다.



**4. DOCTwithIntern**

인턴에 대한 정보와 그 인턴을 교육하는 의사에 대한 정보를 한번에 볼 수 있도록 모아놓은 View이다.



**5. DOCTwithResident**

레지던트에 대한 정보와 그 레지던트를 교육하는 의사에 대한 정보를 한번에 볼 수 있도록 모아놓은 View이다.



**6. PATIENTwithPhone**

환자의 정보와 그 환자의 전화번호를 함께 보여주는 View이다.



**7. DOCTwithPhone**

의사의 정보와 그 의사의 전화번호를 함께 보여주는 View이다.



**8. NURSEwithPhoneAndIncome**

간호사의 정보와 그 간호사의 전화번호, 연봉을 함께 보여주는 View이다.



**9. DEPTRunWARD**

병동의 정보와 그 병동을 관리하는 간호부서의 정보를 함께 보여주는 View이다.



**10. NURSE\_CARE\_PATIENT**

환자의 정보와 그 환자를 간호하는 간호사의 정보를 한번에 볼 수 있게 모아놓은 View이다.



**11. MANAGERofDEPT\_NURSE**

간호사들 중 간호부서의 관리자들의 정보만을 보여주는 View이다.



**12.** **MANAGERofDEPT\_TREAT**

의사들 중 진료부서의 관리자들의 정보를 보여주는 View이다.



**13. DOCTwithTREAT\_PATIENT**

환자의 정보와 그 환자를 치료하는 의사의 정보, 치료의 정보 중 필요한 정보를 한번에 볼 수 있도록 모아놓은 View이다.



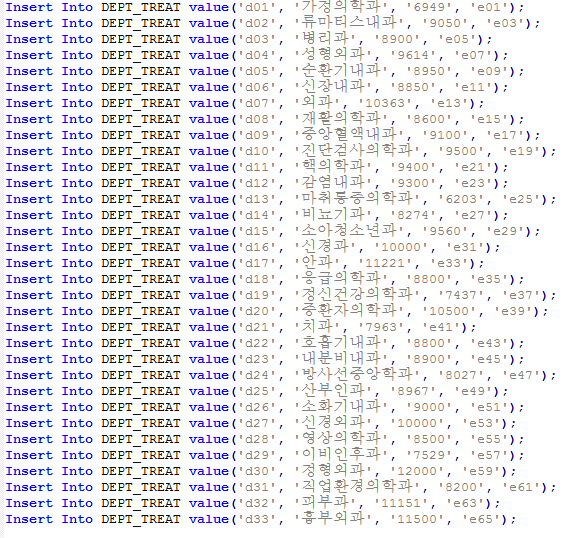
base table들을 만들고 view들을 모두 만든 후 source 명령어를 이용해 작성한 코드를 MySQL에서 실행하였다.



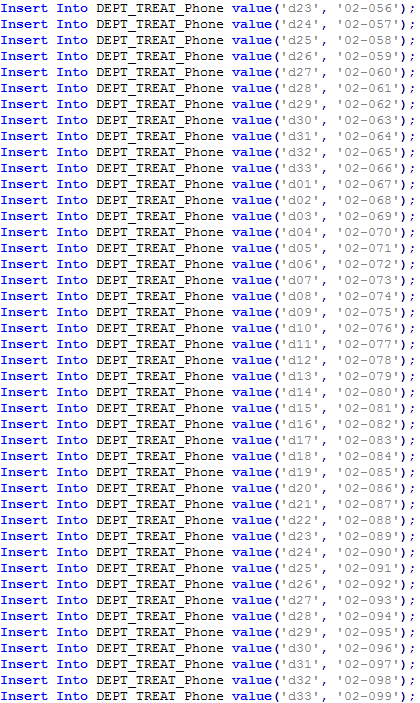
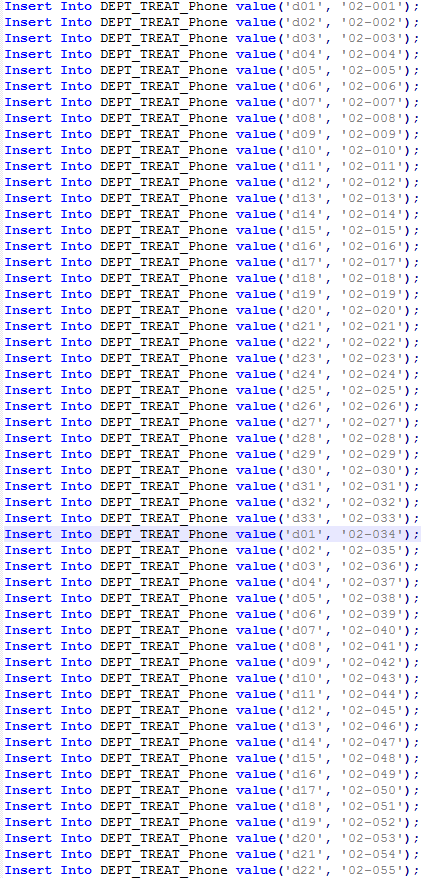
**6. SQL DML 명령문 작성**

**1) Tuple 삽입**

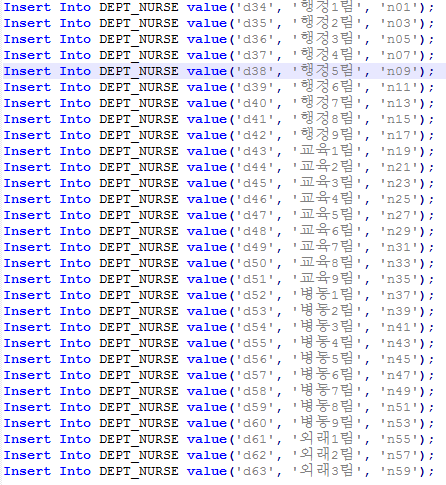
a) DEPT\_TREAT



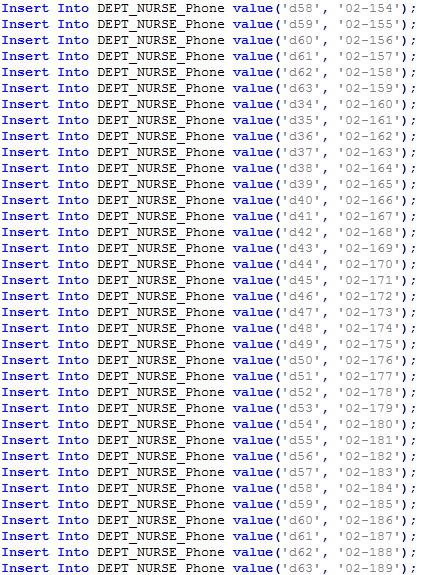
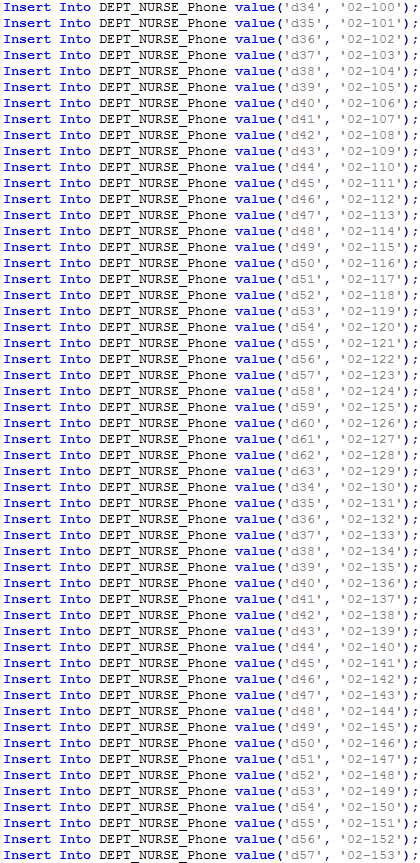
b) DEPT\_TREAT\_Phone



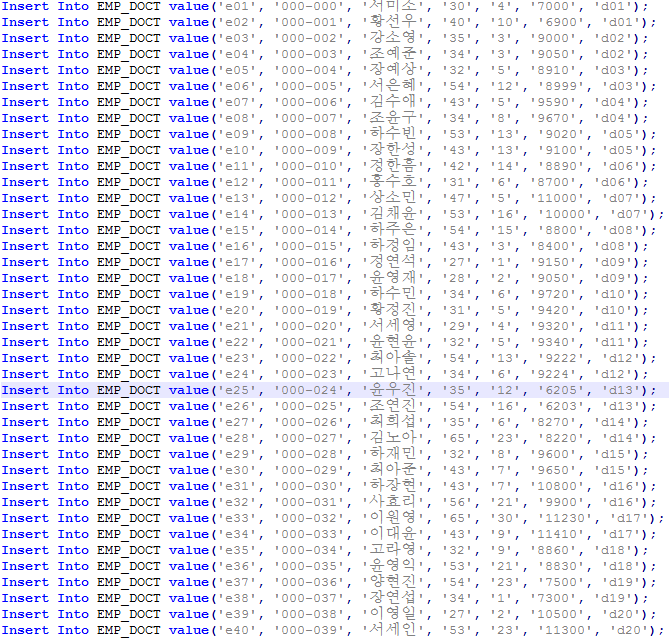
c) DEPT\_NURSE



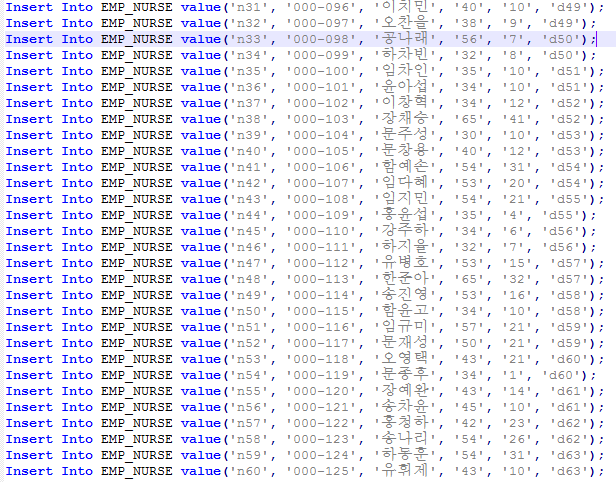
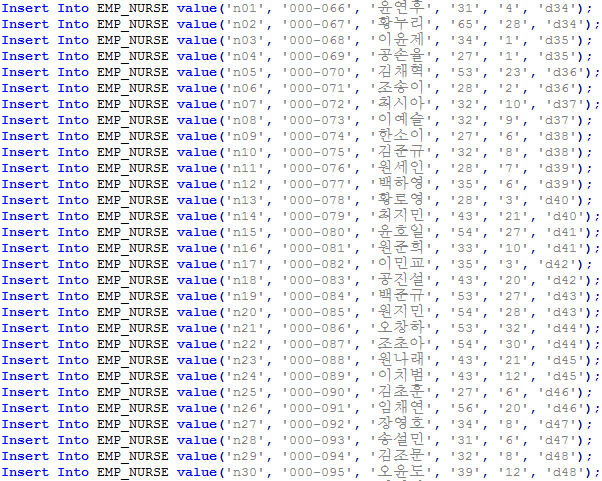
d) DEPT\_NURSE\_Phone



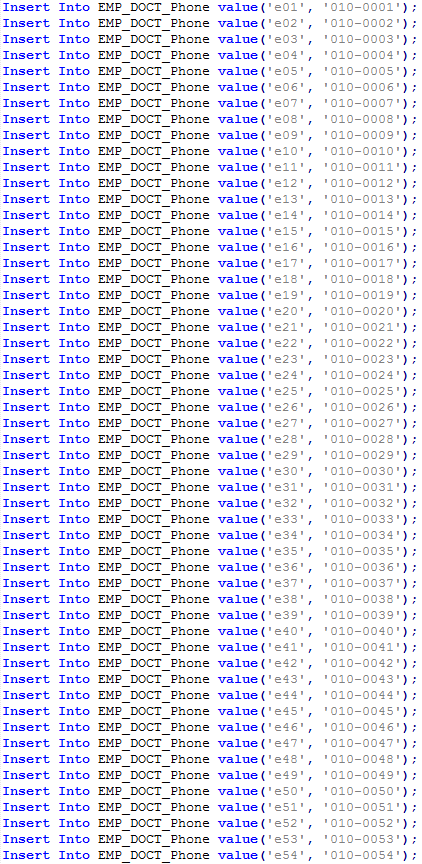
e) EMP\_DOCT



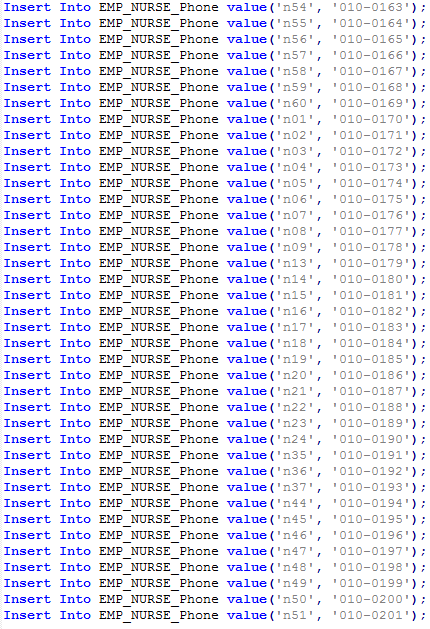
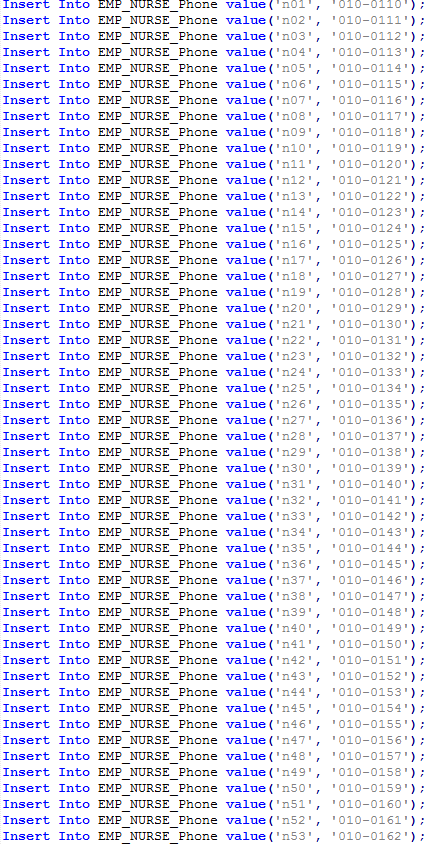
f) EMP\_NURSE



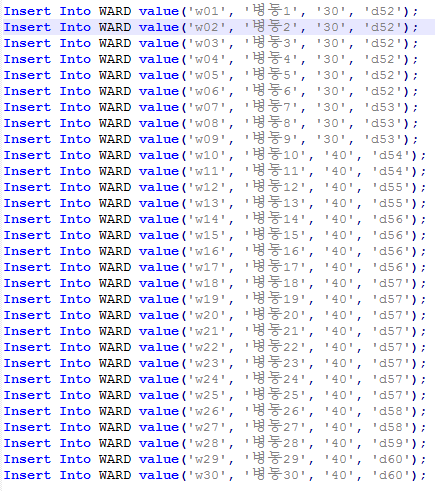
g) EMP\_DOCT\_Phone



h) EMP\_NURSE\_Phone



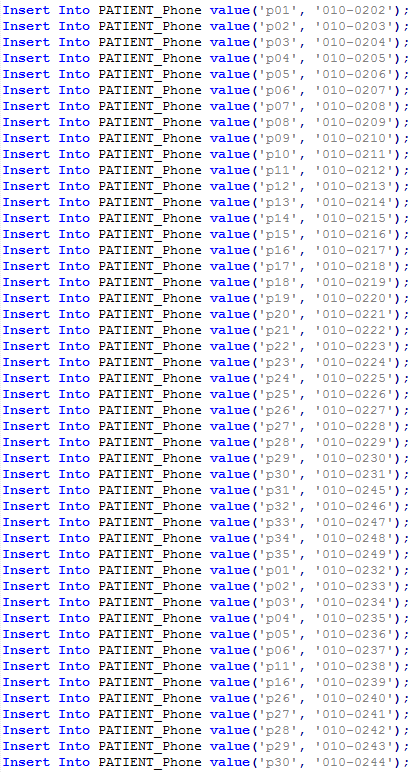
i) WARD



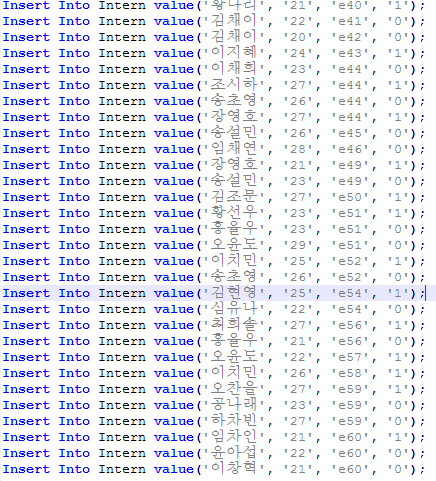
j) PATIENT



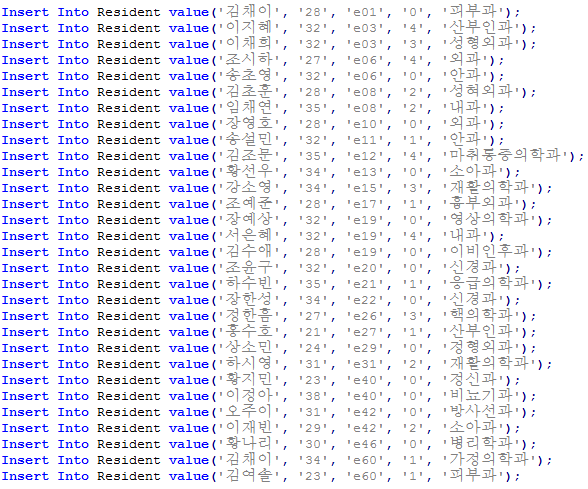
k) PATIENT\_Phone



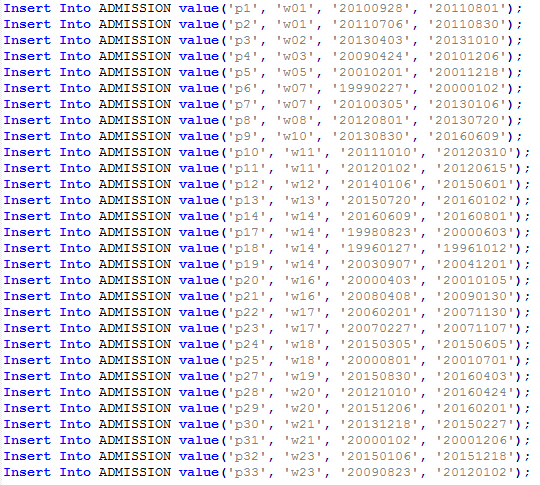
l) Intern



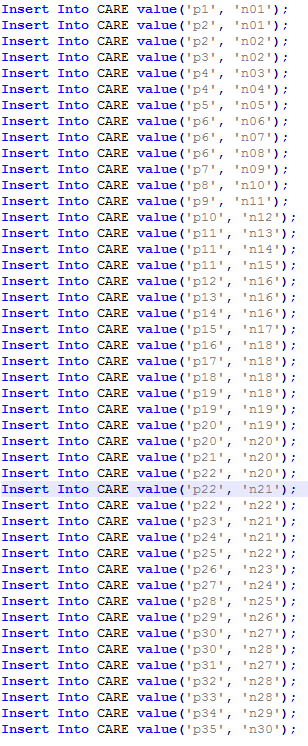
m) Resident



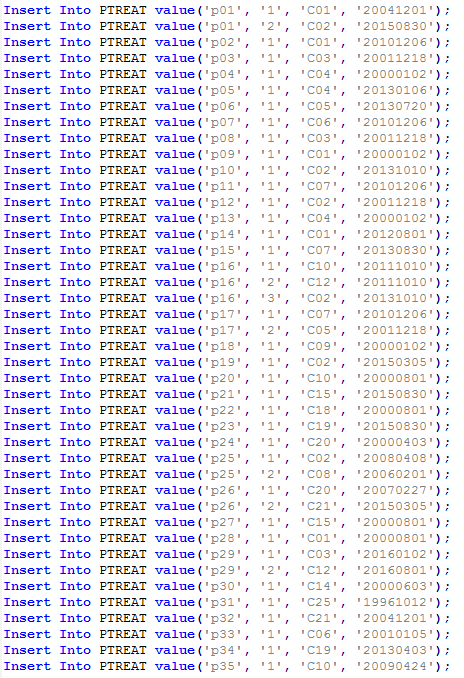
n) ADMISSION



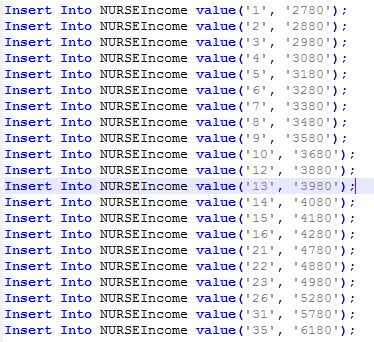
o) CARE



p) PTREAT

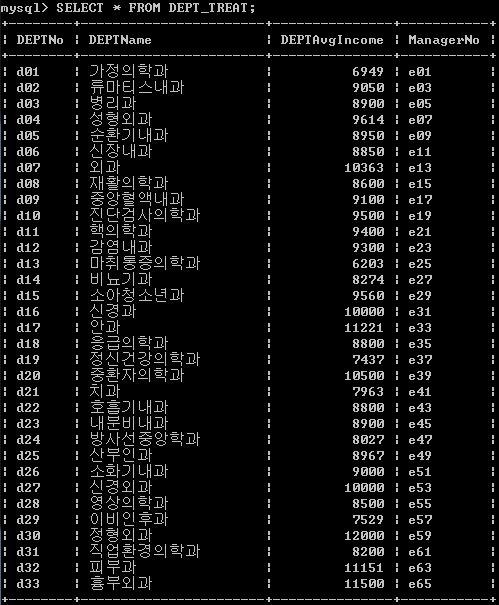


q) NURSEIncome

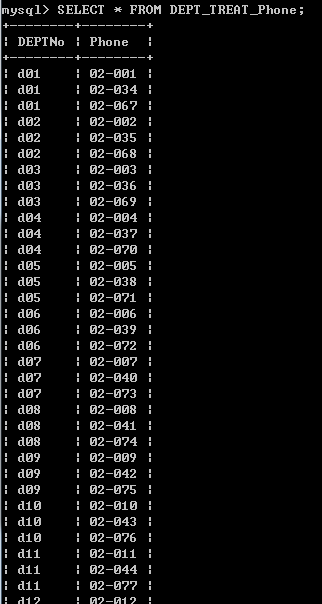


**2) Table 내용 출력**

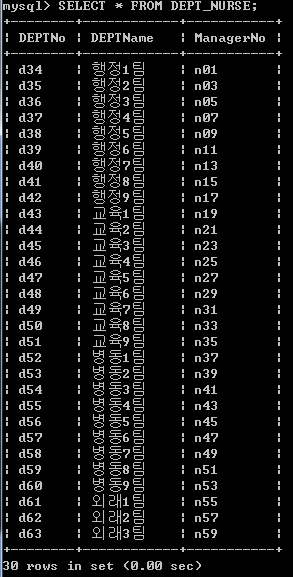
**a) DEPT\_TREAT**



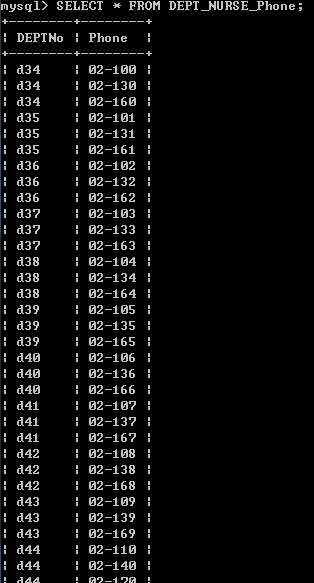
**b) DEPT\_TREAT\_Phone**



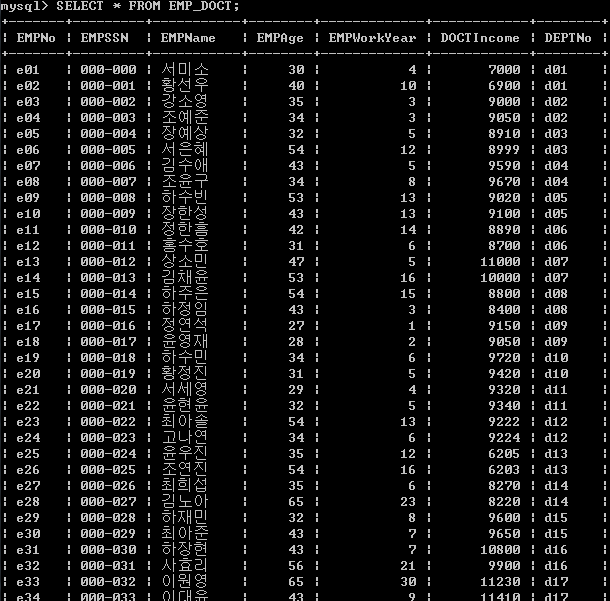
**c) DEPT\_NURSE**



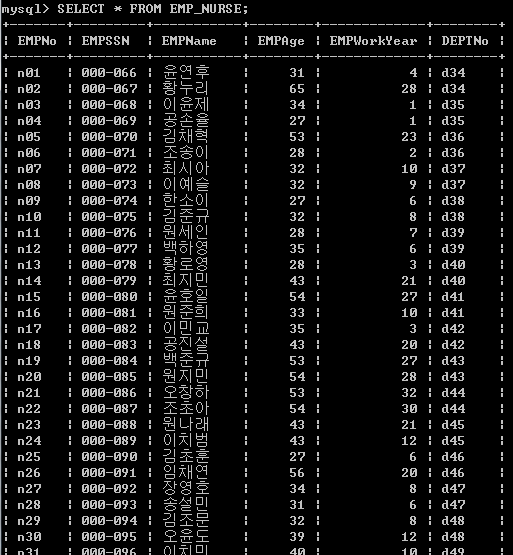
**d) DEPT\_NURSE\_Phone**



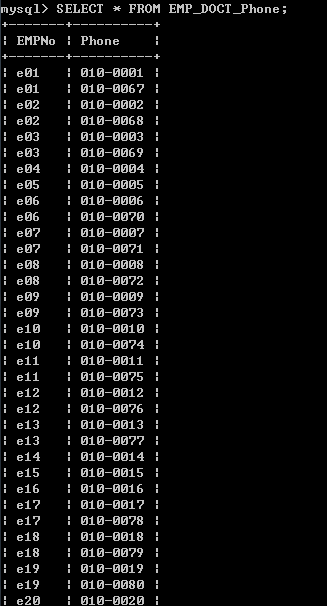
**e) EMP\_DOCT**



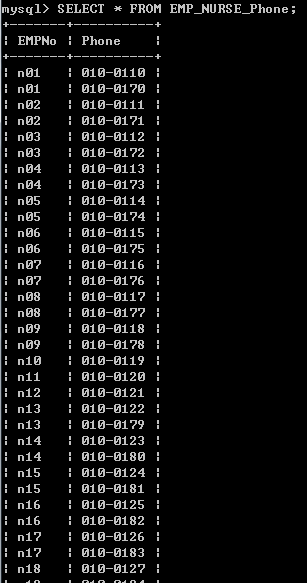
**f) EMP\_NURSE**



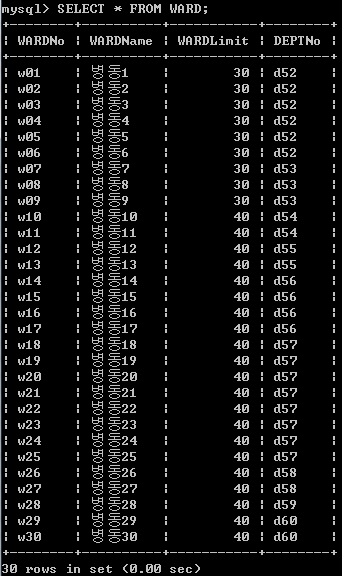
**g) EMP\_DOCT\_Phone**



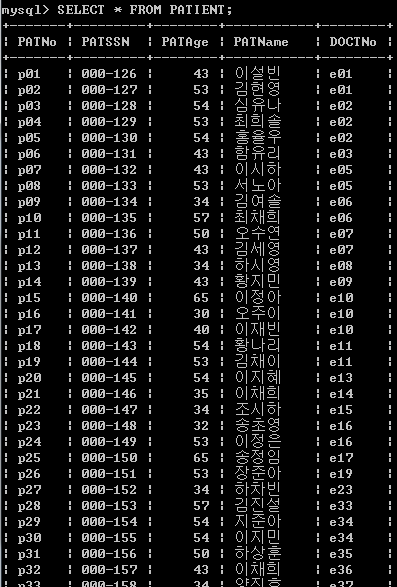
**h) EMP\_NURSE\_Phone**



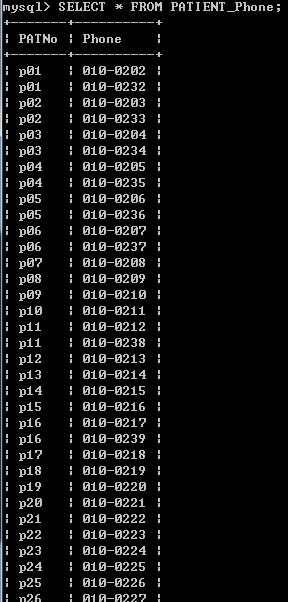
**i) WARD**



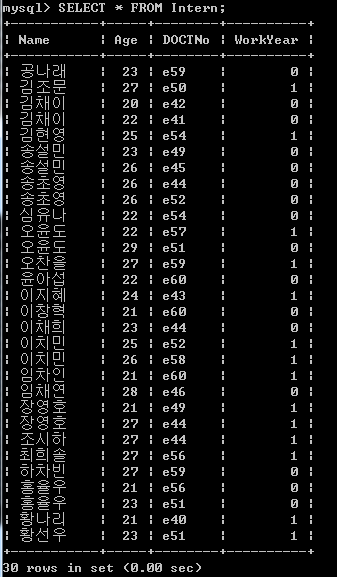
**j) PATIENT**



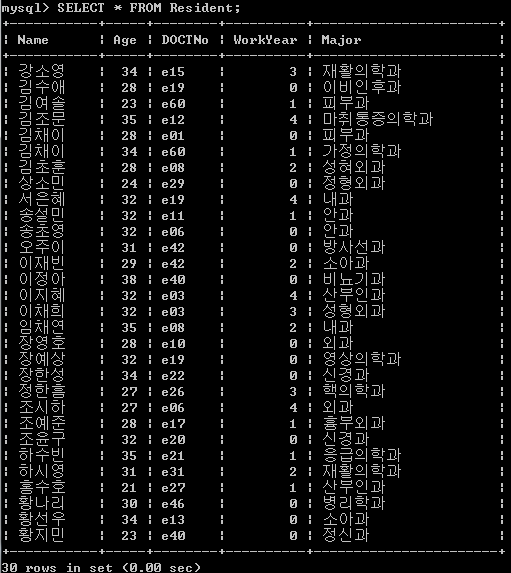
**k) PATIENT\_Phone**



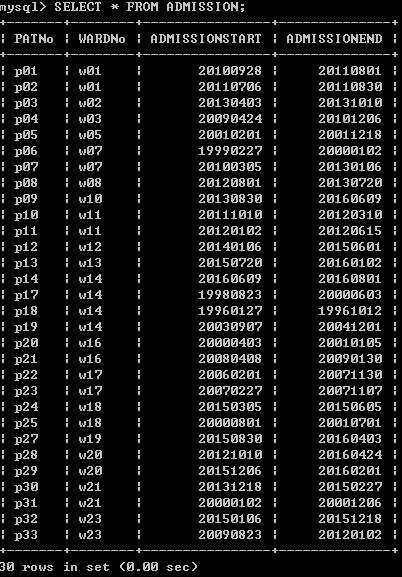
**l) Intern**



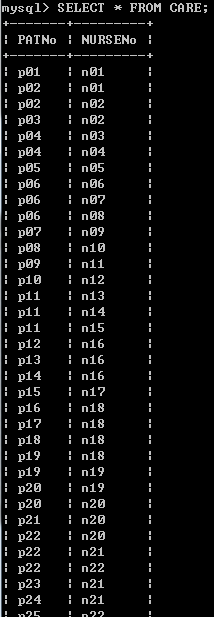
**m) Resident**



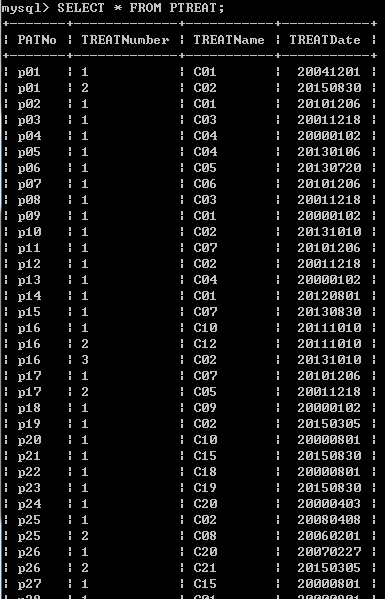
**n) ADMISSION**



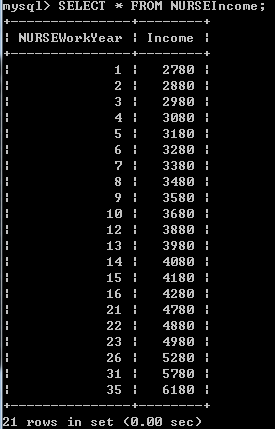
**o) CARE**



**p) PTREAT**

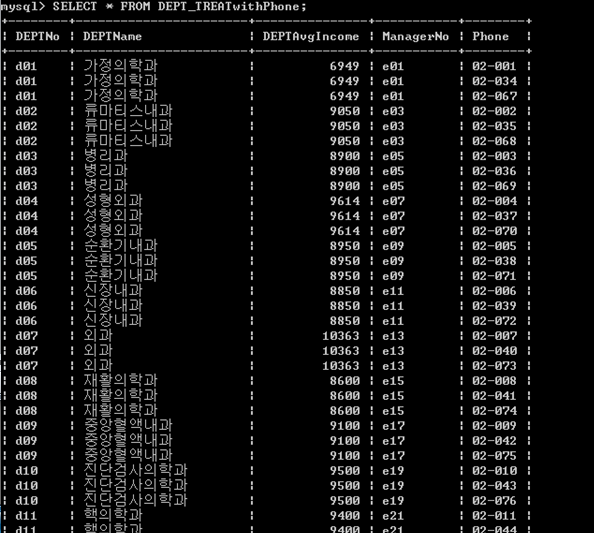


**q) NURSEIncome**

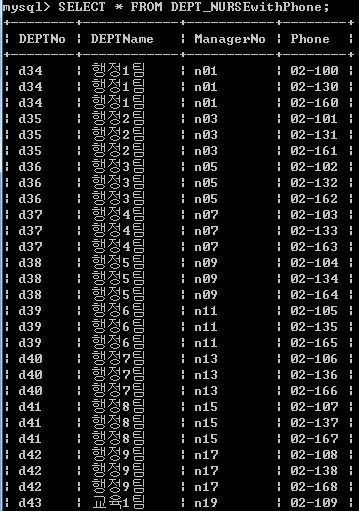


**3) View 내용 출력**

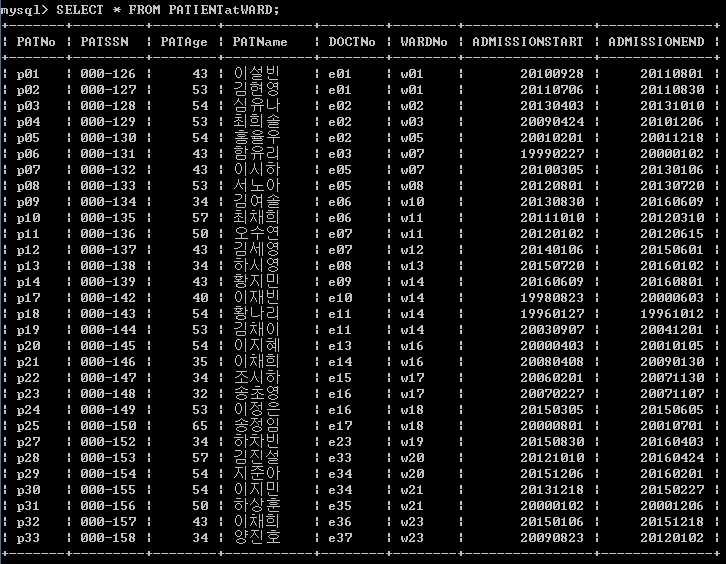
**1. DEPT\_TREATwithPhone**



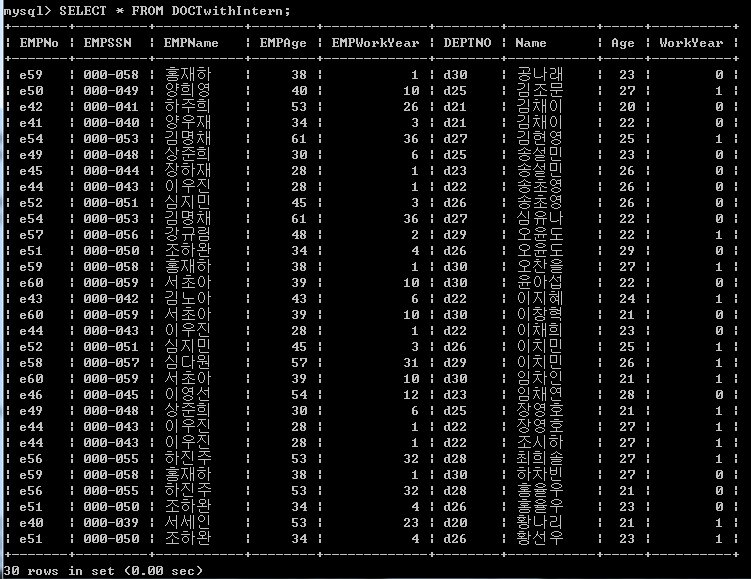
**2.** **DEPT\_NURSEwithPhone**



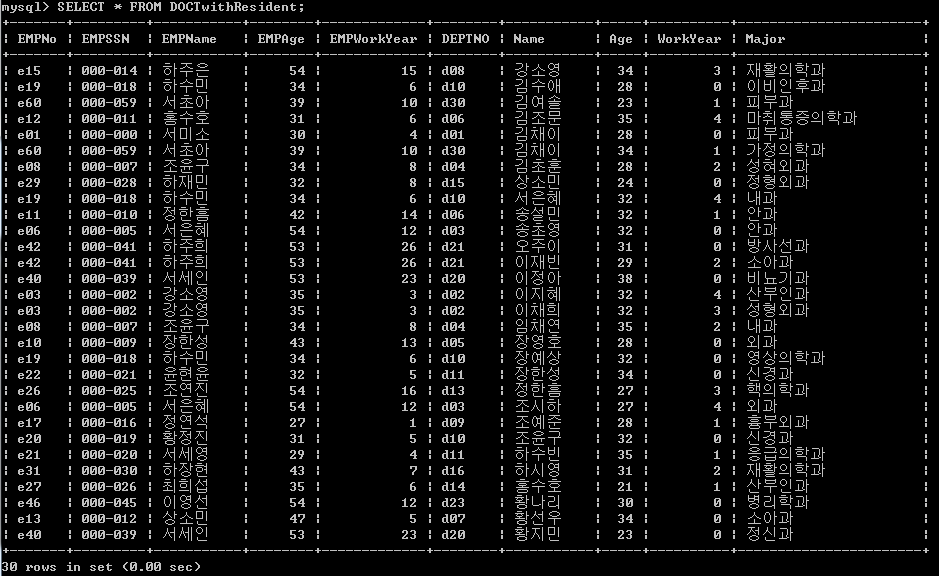
**3. PATIENTatWARD**



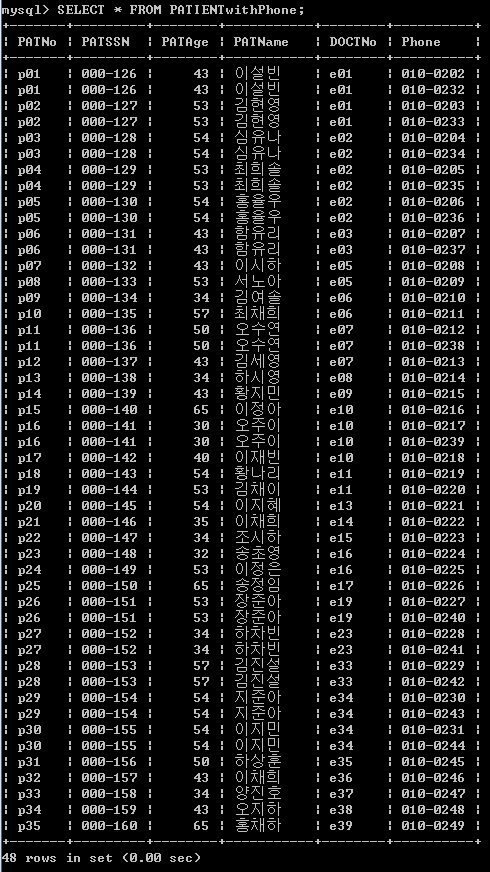
**4. DOCTwithIntern**



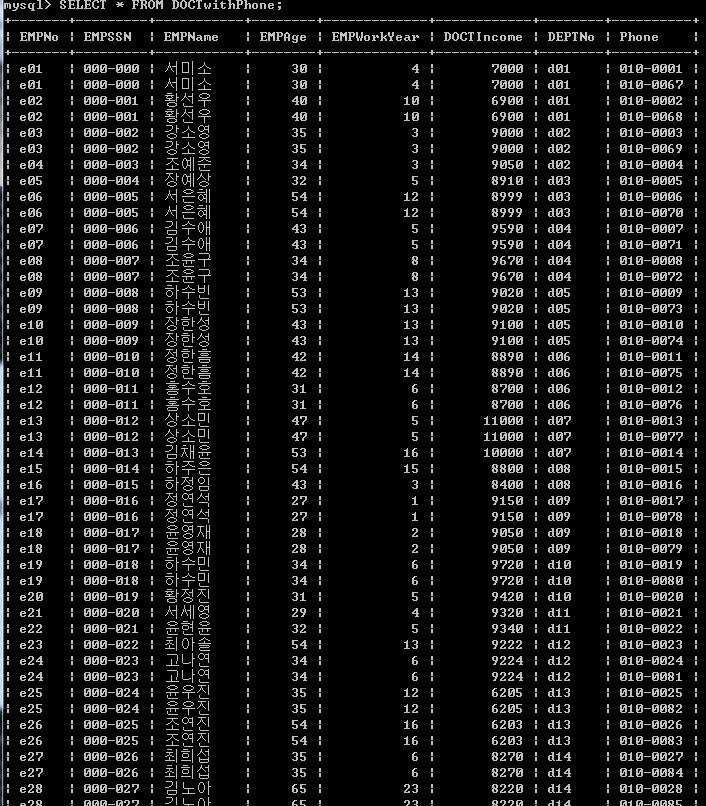
**5. DOCTwithResident**



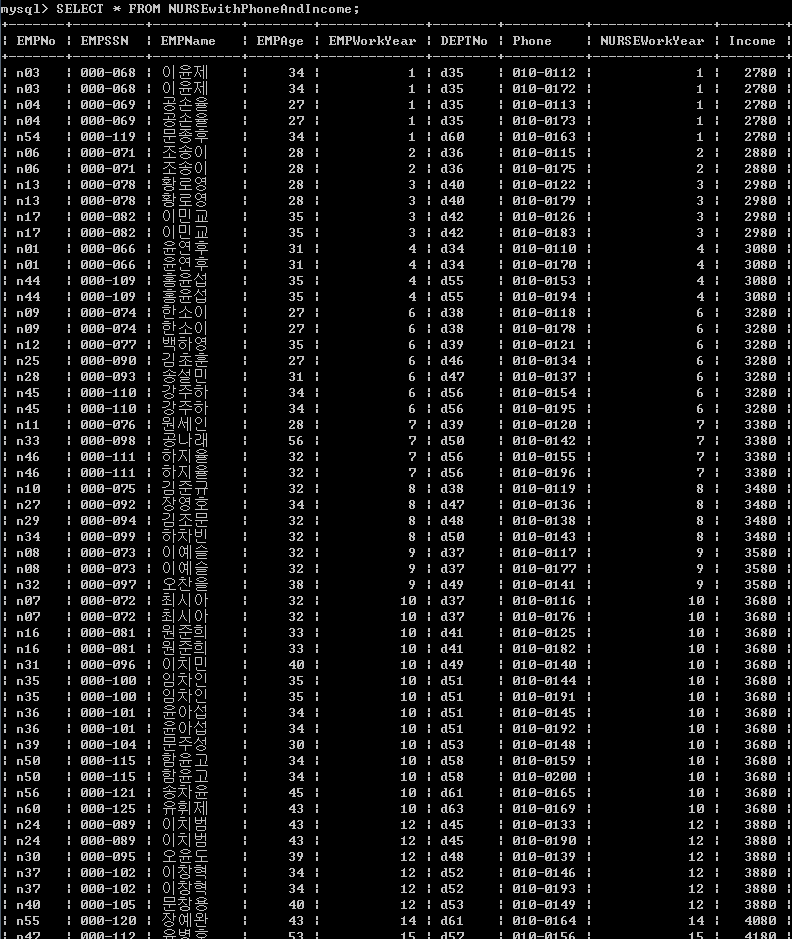
**6. PATIENTwithPhone**



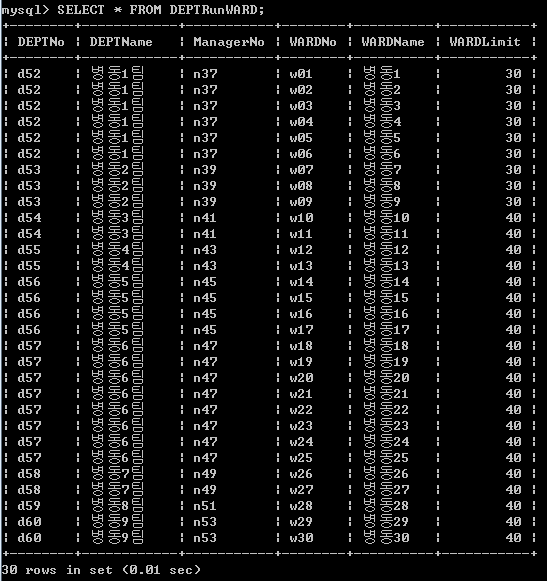
**7. DOCTwithPhone**



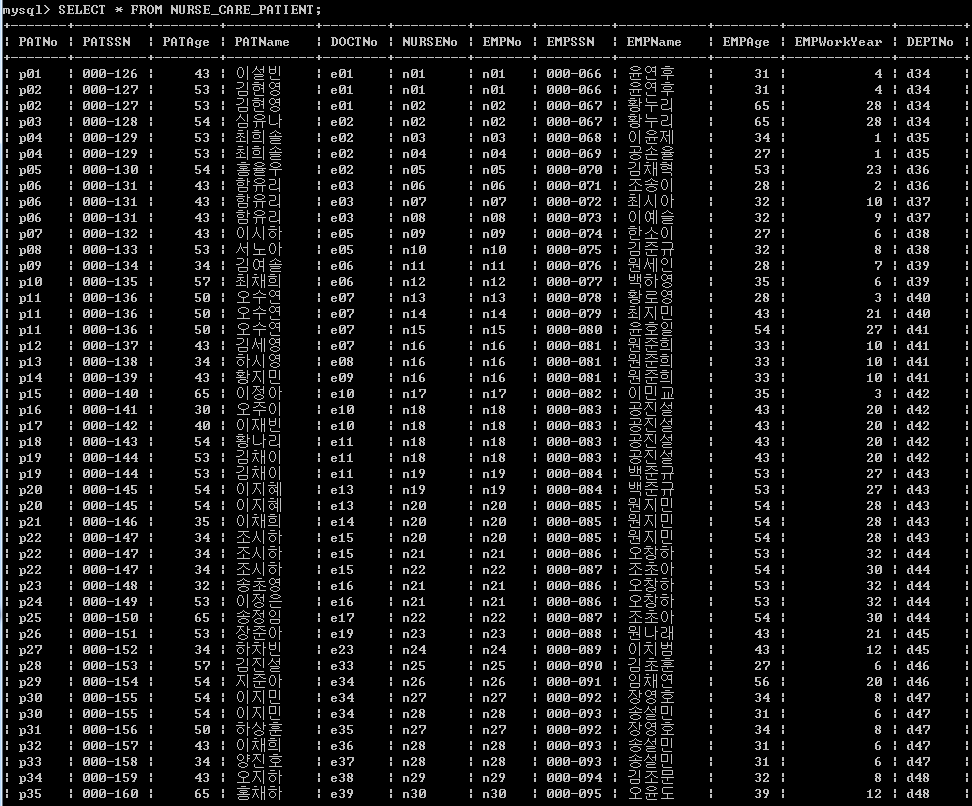
**8. NURSEwithPhoneAndIncome**



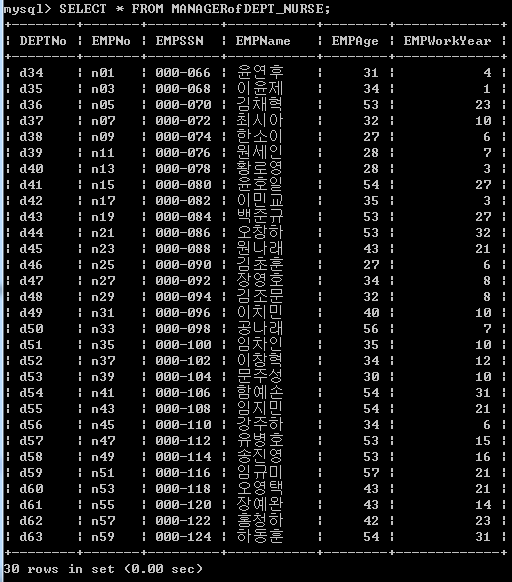
**9. DEPTRunWARD**



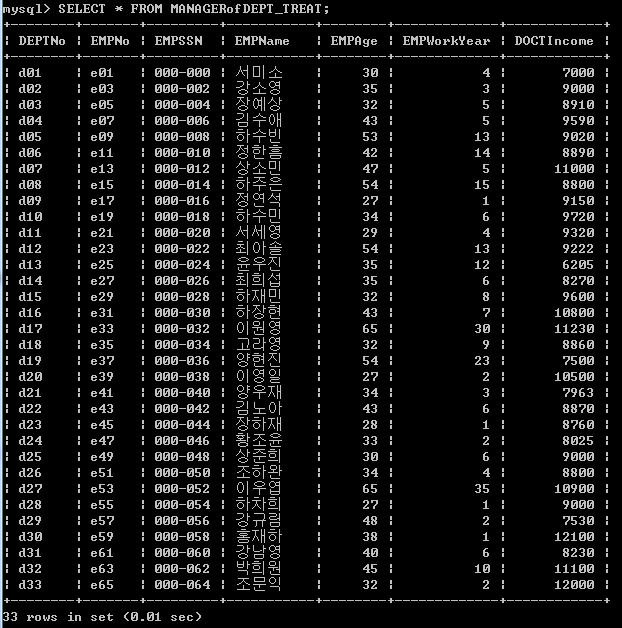
**10. NURSE\_CARE\_PATIENT**



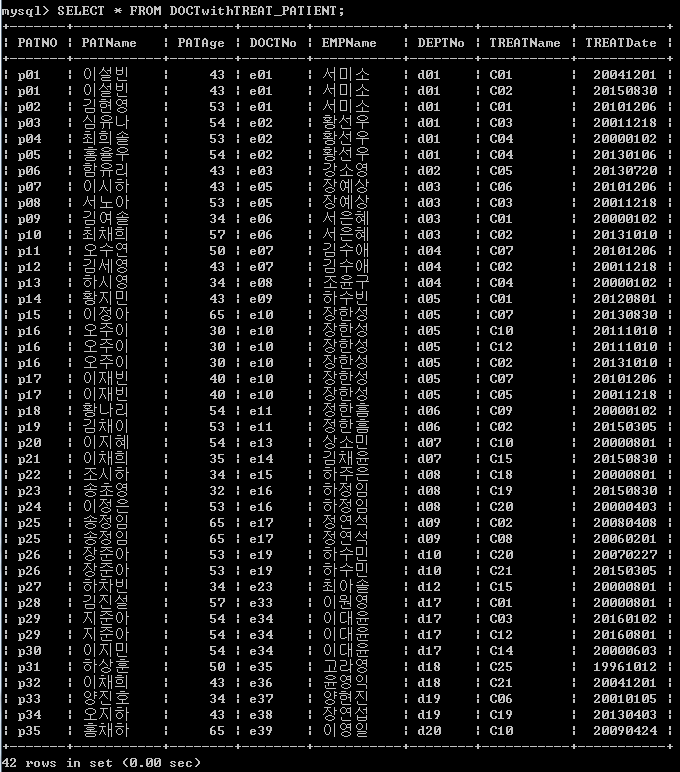
**11. MANAGERofDEPT\_NURSE**



**12.** **MANAGERofDEPT\_TREAT**

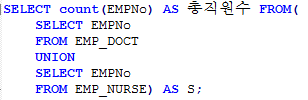


**13. DOCTwithTREAT\_PATIENT**



**4) Query Program**

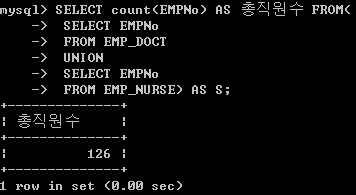
**1. 병원의 총 직원 수를 출력하라.**



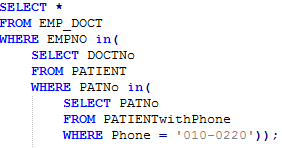
Nested query를 이용하여 총 직원 수를 구하였다. EMP\_DOCT table에서 EMPNo를 가져오고 EMP\_NURSE table에서 EMPNo를 가져와 UNION 한 뒤 EMPNo의 수를 세는 방식으로 총 직원 수를 구하였다. (EMP\_DOCT와 EMP\_NURSE는 UNION될 수 없다. column의 수가 다르다).

도중에 ‘Every derived table must have its own alias.’라는 에러를 만났는데, 이는 sub query가 별명이 필요하기 때문이라고 한다. 따라서 AS S를 붙여주었다.

결과:



**2. 전화번호가 ‘010-0220’인 환자를 진료하는 의사의 정보를 출력하라.**



Nested query를 사용하여 의사의 정보를 구했다. 우선 전화번호가 010-0220인 환자의 환자번호를 구해서 그 환자를 치료하는 의사번호를 구하고 그 의사번호를 가지는 의사의 정보를 출력하였다.

결과:

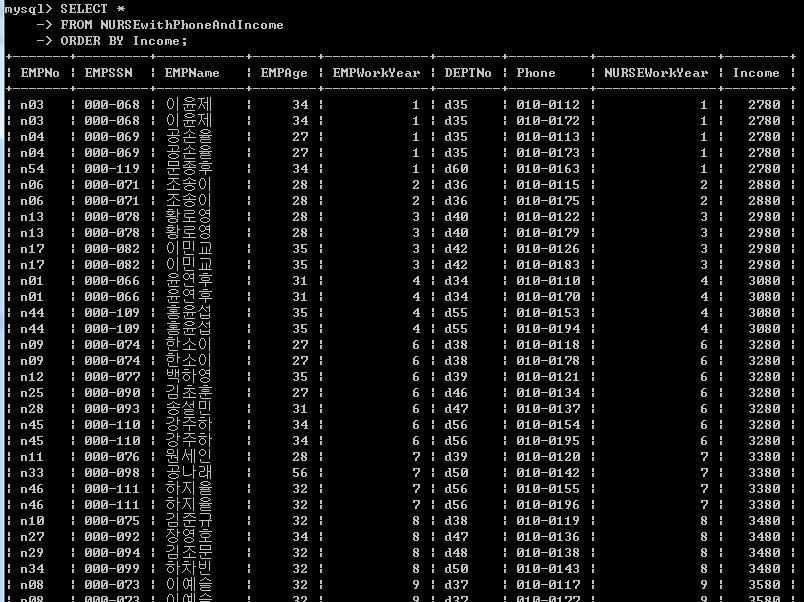


**3. 간호사의 정보를 간호사의 연봉을 기준으로 오름차순으로 정렬하여 출력하여라.**

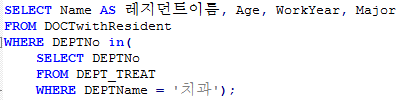


간호사의 정보와 간호사의 연봉이 함께 모여있는 View NURSEwithPhoneAndIncome을 사용하여 Income을 기준으로 오름차순 정렬하여 출력하였다.

결과:

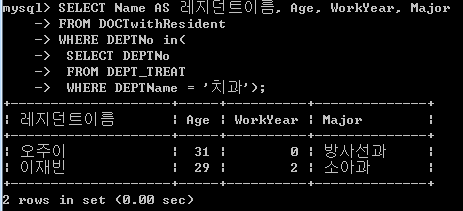


**4. ‘치과’에서 일하는 의사들이 교육하는 레지던트의 정보를 출력하라**

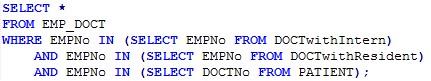


레지던트와 의사에 대한 정보가 모여있는 DOCTwithResident View를 이용하였다. 부서 이름이 치과인 부서의 부서번호를 구해서 DOCTwihResident에서 레지던트를 교육하는 의사들의 부서번호와 비교하여 레지던트들의 정보를 출력하였다.

결과:

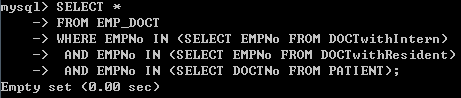


**5. 인턴도 교육하고 레지던트도 교육하고 환자도 진료하는 의사들의 정보를 출력하라.**

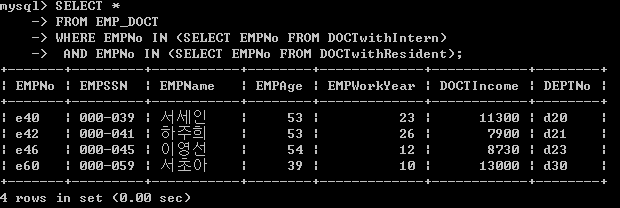


View DOCTwithIntern과 DOCTwithResident, table PATIENT로부터 인턴을 교육하거나 레지던트를 교육하거나 환자를 진료하는 의사의 사원번호를 얻을 수 있다. 이를 이용하여 모든 조건에 해당하는 의사들을 구하였다.

결과:



모든 조건을 만족하는 의사가 없어서 조건 중 환자를 진료한다는 조건을 빼고 다시 검색하여 보았다.

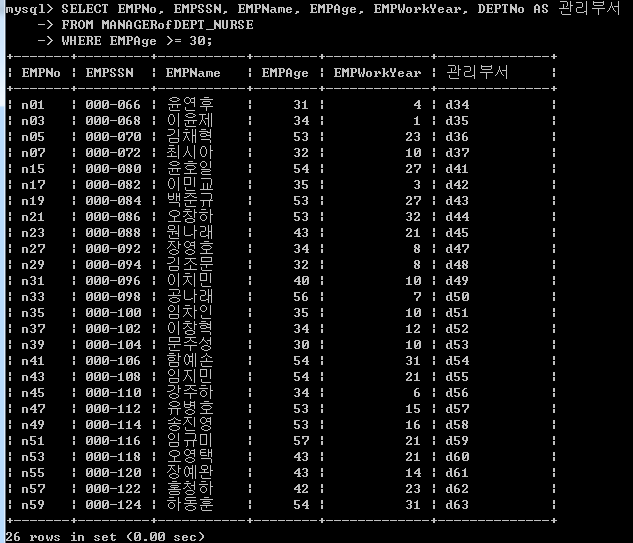


**6. 간호부서의 관리자 중 나이가 30세 이상인 간호사들의 정보를 출력하라**



간호부서의 관리자(간호사)에 대한 정보는 View MANAGERofDEPT\_NURSE에서 확인 할 수 있다. 이 View의 tuple중 EMPAge가 30 이상인 tuple에서 간호사의 정보만을 출력하였다.

결과:

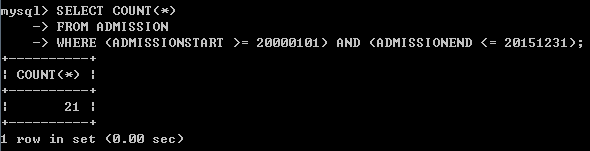


**7. 2000년 1월 1일 이후에 병원에 입원하고 2015년 12월 31일 이전에 퇴원하는 환자들의 수를 출력하여라.**

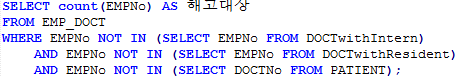


입원에 대한 tuple이 있는 ADMISSION table을 이용하였다. ADMISSION table에는 입원의 시작 날짜와 퇴원 날짜가 기록되므로, 조건에 맞는 tuple의 개수를 세어 출력하였다.

결과:

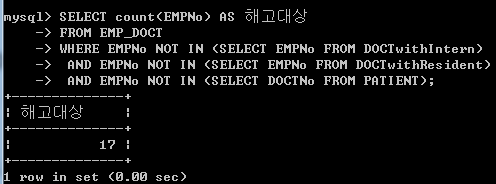


**8. 인턴도 레지던트도 교육하지 않고 환자를 진료하지도 않는 의사들의 수를 출력하라.**



View DOCTwithIntern과 DOCTwithResident, table PATIENT로부터 인턴을 교육하거나 레지던트를 교육하거나 환자를 진료하는 의사의 사원번호를 얻을 수 있다. 이를 이용하여 EMP\_DOCT table에서 위의 조건들에 해당하지 않는 의사들의 수만 출력하였다.

결과:



**9. 레지던트들의 전공별 숫자를 출력하라.**

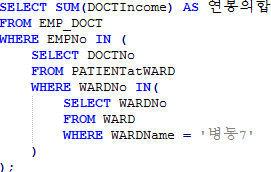


레지던트 table에서 major로 group을 만들고 각 group의 수를 count로 세어 출력하였다.

결과:

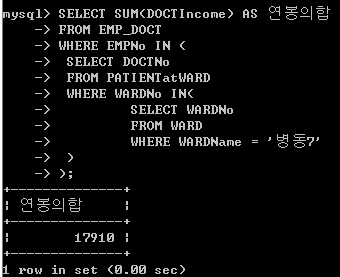


**10. ‘병동7’에 입원한 환자들을 진료하는 의사들의 연봉의 합을 출력하라.**

****

WARD table에서 병동7의 병동번호(병동이름도 키이므로 값이 1개)를 구한 뒤, PATIENTatWARD View를 이용하여 해당 병동에 입원한 환자들을 진료하는 의사의 의사번호를 구하였다. 의사번호를 구한 뒤에는 SUM을 이용하여 연봉의 합을 구해 출력하였다.

결과:

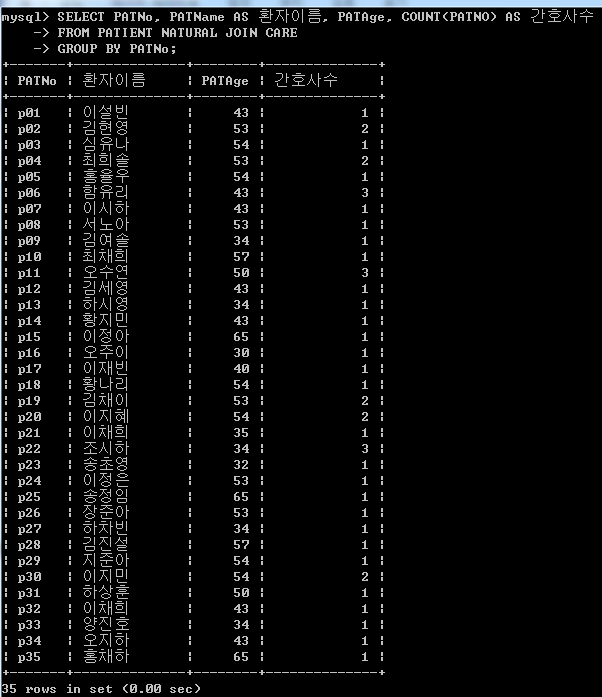


**11. 각 환자에 대해서, 환자의 정보와 함께 환자를 담당하는 간호사 수를 출력하라.**



환자를 담당하는 간호사들의 정보를 알기 위해서 PATIENT와 CARE를 natural join하고, 그 결과에 환자번호로 group하여 (중복되는 환자번호만큼 간호사들이 있을 것이기 때문에) 환자당 간호사 수를 구하였다.

결과:



소스코드는 icampus에 제출한 hospital.sql 파일.