

1. 논리 데이터저장소 확인하기

학습목표

- 업무 분석가, 데이터베이스 엔지니어가 작성한 논리 데이터저장소 설계 내역에서 정의된 데이터의 유형을 확인하고 식별할 수 있다.
- 논리 데이터저장소 설계 내역에서 데이터의 논리적 단위와 데이터 간의 관계를 확인할 수 있다.
- 논리 데이터저장소 설계 내역에서 데이터 또는 데이터간의 제약조건과 이들 간의 관계를 식별할 수 있다.

1. 데이터 모델링 정의

기업의 정보 구조를 실체(Entity)와 관계(Relation)를 중심으로 명확하고 체계적으로 표현하여 문서화하는 기법을 말한다.

2. 데이터 모델링 목적

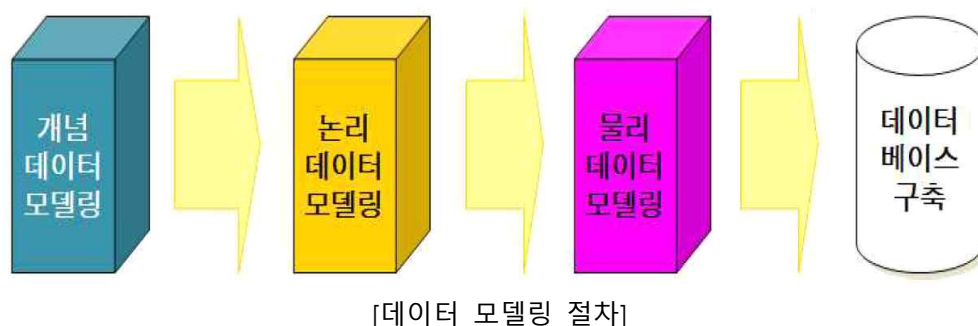
- (1) 연관 조직의 정보요구에 대한 정확한 이해를 할 수 있다.
- (2) 사용자, 설계자, 개발자 간에 효율적인 의사소통 수단을 제공한다.
- (3) 데이터 체계 구축을 통한 고품질 S/W와 유지보수 비용의 감소효과를 기대할 수 있다.
- (4) 신규 또는 개선 시스템의 개발 기초를 제공한다.

3. 데이터 모델링 특성

- (1) 데이터 중심 분석을 통한 업무 흐름 파악이 용이하다.
- (2) 데이터 무결성을 보장할 수 있다
- (3) 데이터의 공유를 통한 중복을 제거하고 일관성 있는 정보를 제공받을 수 있다.

4. 데이터 모델링 절차

데이터 모델링은 개념 모델링, 논리 모델링, 물리 모델링을 통해 데이터베이스에 구축되는 일련의 절차를 거쳐 진행된다.



개념 데이터 모델링	전사의 정보요건을 표현한 상위수준의 모델로서, (1) 주요 엔티티 타입, 기본 속성, 관계, 주요 업무기능 등을 포함한다. (2) 모든 업무 영역을 포함하고, 주제 영역에 포함되는 중심 엔티티 타입 간의 관계를 파악하여 주요 업무 규칙을 정의한다. (3) 논리 데이터 모델의 기초가 된다.
논리 데이터 모델링	개념모델로부터 업무영역의 업무 데이터 및 규칙을 구체적으로 표현한 모델로서, (1) 모든 업무용 엔티티 타입, 속성, 관계, 프로세스 등을 포함한다. (2) 모든 업무 데이터를 정규화(Normalization) 하여 모델링한다. (3) 모든 업무 규칙과 관계를 완전하고 정확하게 표현한다. (4) 성능 혹은 기타 제약 사항과는 독립적인 모델로서, 특정 DBMS로부터 독립적이라 할 수 있다.
물리 데이터 모델링	설계단계에서 시스템의 설계적 및 정보 요건을 정확하고 완전하게 표현한 모델로서, (1) 데이터베이스 생성을 위한 물리 구조로 변환한다. (2) 시스템 설계 요건 반영을 위한 아래와 같은 오브젝트를 추가한다. (가) 설계용 엔티티 타입 (나) 설계용 속성 (3) 설계와 성능을 고려한 조정을 수행한다. (가) 적용 DBMS 특성 고려 (나) 엔티티 타입의 분리 또는 통합 검토 (다) 반정규화(Denormalization) (라) 관계의 해제 (4) 적용 DBMS에 적합한 성능조정을 수행한다. (가) 인덱스 추가 및 조정 (나) 테이블 스페이스 조정 (다) 인덱스 스페이스 조정

5. 데이터 모델링 특성

- (1) 논리적 데이터 모델링 시 요구사항을 충분히 수집하지 않으면 다음 단계의 요구사항 변경에 따른 많은 비용이 발생한다.
- (2) 모든 이해당사자들과 의사소통의 보조 자료로서 E-R 모델을 활용한다.
- (3) 논리적 모델은 H/W나 S/W에 독립적이다.

6. 이상(Anomaly)

- (1) 정의
잘못된 모델링으로 인해 데이터 처리 시 여러 가지 부작용이 발생하는 경우로 삭제 이상, 삽입 이상, 갱신 이상 등이 있다.
- (2) 유형

삭제 이상	한 행을 삭제함으로써 유지해야 될 정보까지도 삭제되는 연쇄 삭제 현상이 일어나게 되어 정보 손실이 발생하게 되는 현상
-------	---

2001020205_16v4 데이터 입출력 구현

[예시]

학생 (학번, 이름)

과목 (과목번호, 과목이름, 교수)

수강 (학번, 과목번호, 점수, 학년)

<수강 테이블 예시>

학번	과목번호	성적	학년
100	C413	A	4
100	E412	A	4
200	C123	B	3
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	4
400	C324	A	4
400	C413	B	4
400	E412	C	4
500	C312	B	2

- 수강 테이블은 학생들의 수강 과목과 성적 정보를 확인할 수 있는 테이블이다.
- 또한 학생들의 학년 정보도 함께 저장하고 있다.
- 한 학생은 여러 과목을 등록할 수 있기 때문에 각 행을 유일하게 식별하기 위해서는 학번 값 하나로
는 되지 못하므로 학번과 과목번호를 함께 사용하여야 한다. 즉, 이 테이블의 기본키는 (학번, 과목번호)
복합키가 필요하다.
- 학번이 200인 학생이 과목 'C123'의 등록을 취소하는 경우 학번 200인 행에서 과목번호 'C123'을 삭제
해야 하는데 과목번호는 기본키 값의 일부이기 때문에 과목 번호만 삭제하지 못하므로 행 전체를
삭제해야 한다.

학번	과목번호	성적	학년
200	C123 => null (X)	B	3

- 학번이 200인 행이 삭제되면 이 학생이 3학년이라는 정보까지도 함께 삭제된다. 왜냐하면 수강 테이블이
학생의 학년 정보를 가지고 있는 유일한 테이블이기 때문이다. 그러나 학년 정보를 삭제하는 것은
원하지 않는다면 이러한 현상을 삭제 이상이라고 한다.

삽입 이상	어떤 데이터를 삽입하려고 할 때 불필요하고 원하지 않는 데이터도 함께 삽입해야만 되고 그렇지 않으면 삽입이 되지 않는 현상
-------	---

- 학번이 600, 이름이 홍길동, 학년이 2학년인 학생이 편입을 했다면 이 사실을 학생 테이블에 학번과
이름을 삽입하고, 학년 정보를 넣기 위해 수강 테이블에 학번과 학년 정보를 삽입하여야 한다. 그러나
이 학생이 최소한 하나의 과목을 등록하지 않는 한 이 삽입을 성공할 수 없다. 왜냐하면 수강 테이블

2001020205_16v4 데이터 입출력 구현

에서는 학번과 과목번호가 기본키이므로 과목 번호 값이 없으면 제약조건 위배가 발생한다.

- 아직 수강한 과목이 없는 상태에서 학번이 600인 학생의 정보를 수강 테이블에 삽입하고 싶다면 임시 과목 번호를 함께 삽입해야 하므로 결국 불필요한 데이터를 삽입해야 하는 현상이 발생하게 되며 이를 삽입 이상이라 한다.

갱신 이상	일부 데이터 값만을 갱신시킴으로써 정보의 모순이 생기는 현상
-------	-----------------------------------

- 학번이 400인 학생의 학년을 4에서 3으로 변경시킨다고 하자. 이 변경을 위해서는 이 테이블에서 학번이 400인 행 4개에 대해 학년의 값을 모두 갱신시켜야 한다. 만일 일부 행만 변경시킨다면 학번 400인 학생의 학년은 3학년과 4학년, 즉 두가지 값을 갖게 되어 일관성이 없게 된다. 이와 같이 중복된 데이터들 중에서 일부 데이터 값만 갱신함으로 인해 모순이 발생되며 이를 갱신 이상이라 한다.

(3) 이상이 발생하는 원인

- 여러 가지 상이한 종류의 정보를 하나의 테이블로 표현하려 하기 때문이다. 즉, 컬럼들 간에 존재하는 여러 가지 데이터 종속 관계를 무리하게 하나의 테이블로 표현하려는 데서 이런 이상들이 발생하게 된다.
- 해결 방법은 컬럼들 간의 종속성을 분석해서 하나의 테이블에는 기본적으로 하나의 종속성이 표현되도록 분해해야 한다. 이러한 분해 과정을 정규화라 한다.

7. 함수 종속(FD : Functional Dependency)

(1) 정의

- 어떤 테이블에 컬럼 X와 컬럼 Y가 있을 경우 컬럼 X의 값이 컬럼 Y의 오직 하나의 값과 연관되어 있다면 Y는 X에 함수 종속이라 하고, $X \rightarrow Y$ 로 표기한다.
- 또한 이때 X를 결정자라고 하고 Y를 종속자라고도 한다.

<p>[예시1] 학생 (학번, 이름, 학년, 학과)</p> <p>학생 테이블에 학번, 이름, 학년, 학과 컬럼이 있다. 어떤 학생의 학번이 정해지면 그 학번에 대응하는 이름, 학년, 학과는 오직 하나의 값만 가지게 된다. 그러므로 이름, 학년, 학과 컬럼은 학번 컬럼에 함수 종속이라고 말할 수 있다.</p> <p>이름 기호로 표현하면, 다음과 같다.</p>	<p>학번 \rightarrow 이름</p> <p>학번 \rightarrow 학년</p> <p>학번 \rightarrow 학과</p> <p>또는</p> <p>학번 \rightarrow (이름, 학년, 학과)</p>
--	---

2001020205_16v4 데이터 입출력 구현

(2) 함수 종속 다이어그램

- 한 테이블에서 컬럼들 간의 복잡한 함수 종속 관계를 쉽게 이해하기 위해 도식화하여 표현한다.

[예시2] 수강 (학번, 과목번호, 학년, 성적)



<수강 테이블의 함수 종속 다이어그램>

(학번, 과목번호) → 성적
학번 → 학년

(3) 완전 함수 종속과 부분 함수 종속

- 성적 컬럼은 기본키인 (학번, 과목번호)에 함수 종속이며, 학년 컬럼은 학번 컬럼에만 함수 종속이 되어 있다. 그러나 학번 컬럼이 기본키인 (학번, 과목번호) 복합 컬럼에 포함되어 있으므로 당연히 학년 컬럼이 (학번, 과목번호) 컬럼에도 함수 종속되어 있다고 말할 수 있다.

(학번, 과목번호) → 학년

- 이러한 경우 성적 컬럼은 기본키인 (학번, 과목번호) 복합 컬럼에 완전 함수 종속이라고 하며, 학년 컬럼은 기본키인 (학번, 과목번호) 복합 컬럼에 부분 함수 종속이라고 한다.

8. 정규화(Normalization)

(1) 정의

논리 모델링 단계에서 수행하며, 중복성을 최소화하고 정보의 일관성을 보장하기 위한 작업

(2) 목적

- (가) 데이터 중복 배제로 데이터 관리 편의성 및 자료 저장 공간의 최소화
- (나) 데이터 모형 단순화
- (다) 데이터 구조의 안정성 및 무결성 유지
- (라) 속성의 배열상태 검증
- (마) 엔티티와 속성의 누락 여부 검증 수단
- (바) 자료검색과 추출의 효율성을 추구

(3) 특징

- (가) 어떠한 관계구조가 바람직한 것인지, 바람직하지 못한 관계를 어떻게 분해하여야 하는지에 관한 구체적인 판단기준을 제공
- (나) 정규화된 데이터 모델은 정확성, 일치성, 단순성, 비중복성, 안정성 보장

2001020205_16v4 데이터 입출력 구현

(4) 유형

제1정규화	1) 반복되는 속성이나 Group 속성 제거 2) 새로운 실체와 1:N의 관계 추가 3) 모든 속성은 반드시 하나의 값을 가져야 함(반복형태가 있어서는 안됨)
제2정규화	1) 주식별자에 완전하게 종속되지 않는 속성 제거 2) 불완전 함수적 종속(Non Fully Dependency) 제거 3) 모든 속성은 반드시 UID전부에 종속되어야 함(UID일부에만 종속되어서는 안됨)
제3정규화	1) 비식별자에 종속되는 속성 제거 2) 주식별자에 이행종속(Transitive Dependency) 되는 속성 제거 3) UID가 아닌 모든 속성 간에는 서로 종속될 수 없음(속성간 종속성 배제)
제4정규화	1) 실제로 거의 고려되지 않는 정규화 2) 주식별자에 다가종속(Multi-Valued Dependency)되는 속성을 두가지 이상 두지 않음 3) 2차 정규화된 테이블은 다대다 관계를 가질 수 없음 4) 어떠한 관계구조가 바람직한 것인지, 바람직하지 못한 관계를 어떻게 분해하여야 하는지에 관한 구체적인 판단기준을 제공

(5) 제1정규형(1NF : First Normal Form)

어떤 테이블에 속한 모든 값들이 원자 값으로 되어 있다면 제1정규형에 속한다.

비정규 릴레이션			
학번	이름	과목명	성적
100	King	Oracle, Java	90, 85
101	Abel	Java, Linux	80, 95

→

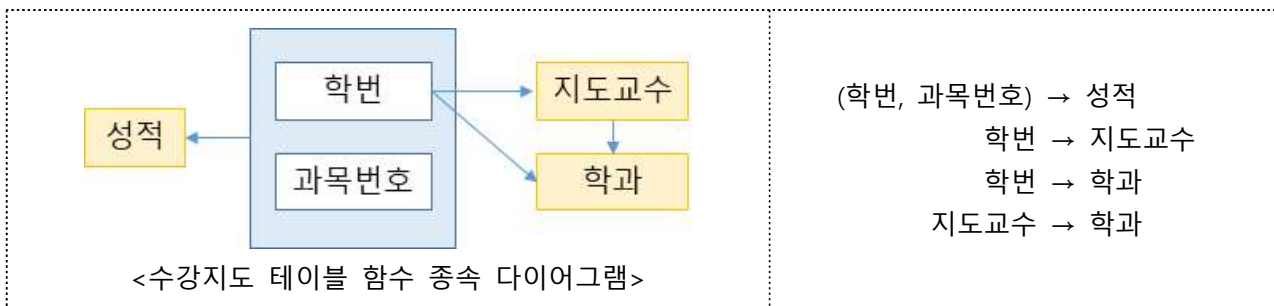
원자값으로 구성된 1NF 테이블			
학번	이름	과목명	성적
100	King	Oracle	90
101	Abel	Java	80
101	Abel	Linux	95
100	King	Java	85

- 대부분의 테이블은 제1정규형에 속한다. 그러나 제1정규형에만 속해 있는 테이블은 여러 가지 이유에서 바람직하지 않은 구조가 될 수 있다.

2001020205_16v4 데이터 입출력 구현

<수강지도 테이블>

학번	지도교수	학과	과목번호	성적
100	P1	컴퓨터	C413	A
100	P1	컴퓨터	E412	A
200	P2	전기	C123	B
300	P3	컴퓨터	C312	A
300	P3	컴퓨터	C324	C
300	P3	컴퓨터	C413	A
400	P1	컴퓨터	C312	A
400	P1	컴퓨터	C324	A
400	P1	컴퓨터	C413	B
400	P1	컴퓨터	E412	C



- 학번과 과목번호의 조합으로 학생이 수강한 과목의 성적을 식별할 수 있다.
- 한 학생은 한 지도교수를 가질 수 있고, 한 학과에만 속한다.
- 또한 각 지도교수는 한 학과에만 속한다.
- 수강지도 테이블의 저장된 값들을 보면 이 테이블은 지도교수와 학과 컬럼에 중복된 데이터가 많은 것을 볼 수 있다. 이로 인해 여러 가지 이상 현상이 발생할 수 있다.

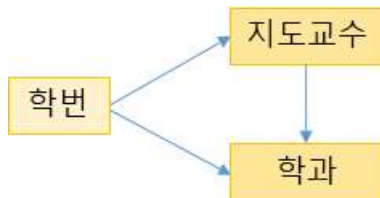
삽입 이상	- 예를 들어, 학번이 500인 학생의 지도교수가 P4라는 사실만 삽입할 수 없다. 왜냐하면 학번이 500인 학생이 어떤 교과목을 등록하지 않으면 기본 키에 속하는 과목번호가 널이 되므로 제약조건 위배로 인해 작업이 수행되지 않기 때문이다.
삭제 이상	- 만일 학번이 200인 학생이 교과목 C123의 등록을 취소하면 교과목(기본키)만 삭제가 되지 않으므로 학번이 200인 학생의 행 전체를 삭제해야 한다. 이렇게 되면 이 학생의 지도교수가 P2라는 정보까지도 손실되므로 삭제 이상이 발생된다.
갱신 이상	- 학번이 400인 학생의 지도교수가 P1에서 P3로 변경되었다면, 4개의 행에 대해 지도교수 데이터 값을 P3으로 변경해 주어야 한다. 만일 일부만 변경되면 400번 학생의 지도교수가 P1과 P3, 즉 둘이 되어 데이터의 일관성이 떨어지게 되는 갱신 이상이 발생된다.

2001020205_16v4 데이터 입출력 구현

- 이러한 문제가 발생하는 원인은 수강지도 테이블에서 기본키가 아닌 컬럼들이 기본키에 완전 함수 종속되지 못하고 부분 함수 종속이 되기 때문이다. 따라서 이러한 문제들에 대한 해결은 수강지도 테이블을 두 개의 테이블로 분할하여 부분 함수 종속을 제거하는 것이다.
- 부분 함수 종속을 제거하기 위해 수강지도 테이블을 지도 테이블과 수강 테이블로 재구성하여 제1정규형에서 일어나는 문제를 해결하였다.
- 한 테이블에 부분 함수 종속이 존재한다는 것은 기본키로 식별되는 개체와 무관한 컬럼이 있다는 것이다. 즉, 두 가지 종류의 상이한 정보를 하나의 테이블에 혼합시켜 놓은 것이므로 정규화를 거쳐야 이상 현상을 막을 수 있다.

[테이블1] 지도 (학번, 지도교수, 학과)

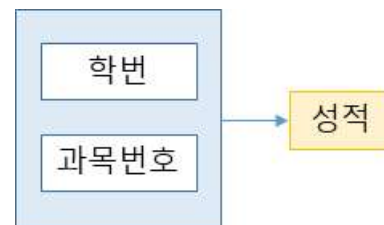
학번 → 지도교수
학번 → 학과
지도교수 → 학과



<지도 테이블 함수 종속 다이어그램>

[테이블2] 수강 (학번(FK), 과목번호, 성적)

(학번, 과목번호) → 성적



<수강 테이블 함수 종속 다이어그램>

(6) 제2정규형(2NF)

어떤 테이블이 1NF이고 기본키에 속하지 않은 컬럼 모두가 기본키에 완전 함수 종속이면 제2정규형에 속한다.

<지도 테이블>

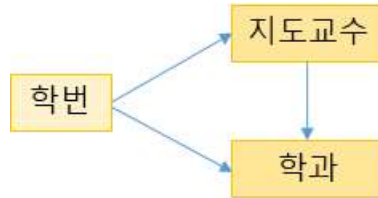
학번	지도교수	학과
100	P1	컴퓨터
200	P2	전기
300	P3	컴퓨터
400	P1	컴퓨터

<수강 테이블>

학번(FK)	과목번호	성적
100	C413	A
100	E412	A
200	C123	B
300	C312	A
300	C324	C
300	C413	A
400	C312	A
400	C324	A
400	C413	B
400	E412	C

2001020205_16v4 데이터 입출력 구현

- 지도 테이블과 수강 테이블은 모두 2NF에 속한다.
- 지도 테이블과 수강 테이블의 함수 종속 다이어그램을 살펴보면, 수강 테이블은 문제가 없지만 지도 테이블은 복잡한 상호 함수 종속 관계를 나타내고 있다.



<지도 테이블 함수 종속 다이어그램>

- 학과는 학번에 완전 함수 종속이면서 지도교수를 통해 이행적 함수 종속이 되고 있다. 즉, 학번은 지도교수를 결정하고 이 지도교수는 학과를 결정한다.

학번 → 지도교수, 지도교수 → 학과 =====> 학번 → 학과

- 학과는 학번에 이행적 함수 종속이라고 한다. 이러한 이행적 함수 종속을 가진 지도 테이블에 나타나는 이상 현상은 다음과 같다.

<지도 테이블>

학번	지도교수	학과
100	P1	컴퓨터
200	P2	전기
300	P3	컴퓨터
400	P1	컴퓨터

삽입 이상	- 지도 테이블에 어떤 지도교수가 어떤 특정 학과에 속한다는 사실만을 삽입하려고 할 때 삽입을 할 수 없다. 왜냐하면 지도교수의 지도를 받는 학생 정보가 없다면 기본키가 널이 되기 때문에 삽입 이상이 발생된다.
삭제 이상	- 학번이 300인 학생의 행이 삭제되는 경우 지도교수 P3이 컴퓨터 학과에 속한다는 정보도 손실이 되는 삭제 이상이 발생된다.
갱신 이상	- 지도교수 P1의 학과가 컴퓨터에서 전자로 바뀐다면 학번 100과 400에 있는 학과의 값을 모두 변경시켜야 한다. 그렇지 않으면 지도교수 P1은 두 개의 상이한 학과 값을 갖게 되어 데이터 모순이 발생된다.

- 이러한 이상은 결국 두 개의 상이한 정보를 하나의 테이블로 혼합해서 표현하려고 하여 발생된다.
- 이행적 함수 종속은 복수의 관계를 하나의 테이블로 표현하는 데서 나타나는 것이다. 그러므로 해결 방법은 이행적 함수 종속을 제거하여 두 개의 테이블로 분해하는 것이다.

[테이블1] 학생지도 (학번, 지도교수(FK))



<학생지도 테이블 함수 종속 다이어그램>

[테이블2] 지도교수학과 (지도교수, 학과)



<지도교수 테이블 함수 종속 다이어그램>

2001020205_16v4 데이터 입출력 구현

(7) 제3정규형(3NF)

어떤 테이블이 2NF이고 기본키에 속하지 않은 컬럼 모두가 기본키에 이행적 함수 종속이 아니면 제3정규형에 속한다.

<학생지도 테이블>

학번	지도교수(FK)
100	P1
200	P2
300	P3
400	P1

<지도교수학과 테이블>

지도교수	학과
P1	컴퓨터
P2	전기
P3	컴퓨터

- 학생지도 테이블과 지도교수학과 테이블은 3NF이다.

(8) 보이시/코드 정규형(Boyce/Codd Normal Form(BCNF), 강한 제3정규형)

어떤 테이블의 모든 결정자가 후보키(기본키)이면 보이시/코드 정규형(BCNF)에 속한다.
BCNF에 속하는 테이블은 모두 제3정규형에 속하지만 반대는 성립하지 않는다.
그런 의미에서 BCNF는 제3정규형보다 강력하므로 "강한 제3정규형"이라고도 한다.

- 앞에서 보았던 테이블 중 수강지도 테이블, 지도 테이블은 3NF도 아니고 BCNF도 아니다.

<수강지도 테이블>	<지도 테이블>
수강지도 (학번, 지도교수, 학과, 과목번호, 성적) (학번, 과목번호) → 성적 학번 → 지도교수 학번 → 학과 지도교수 → 학과 (학번, 과목번호), 학번, 지도교수가 결정자이지만 (학번, 과목번호)만 기본키이므로 BCNF가 아니다.	지도 (학번, 지도교수, 학과) 학번 → 지도교수 학번 → 학과 지도교수 → 학과 학번, 지도교수가 결정자이지만 학번만 기본키이므로 BCNF가 아니다.

- 그러나 수강 테이블, 학생지도 테이블, 지도교수학과 테이블은 3NF이면서 BCNF에 속한다.

<수강 테이블>	<학생지도 테이블>	<지도교수학과 테이블>
수강 (학번(FK), 과목번호, 성적) (학번, 과목번호) → 성적	학생지도 (학번, 지도교수(FK)) 학번 → 지도교수	지도교수학과 (지도교수, 학과) 지도교수 → 학과
위 세 테이블은 결정자가 모두 기본키이므로 BCNF에 속한다.		

- 3NF이지만 BCNF가 아닌 테이블인 경우 여러 이상 현상이 발생할 수 있다.
- 3NF이지만 BCNF가 아닌 테이블은 다음과 같다.
- 수강과목 테이블은 한 학생은 여러 과목을 수강할 수 있으므로 학번 컬럼 하나로는 기본키가 될 수

2001020205_16v4 데이터 입출력 구현

없다. 그러므로 (학번, 과목) 또는 (학번, 교수)가 기본키가 될 수 있다.

- (학번, 과목)을 기본키로 결정하였다.
- 또한 교수는 한 과목만 담당할 수 있고, 하나의 과목은 여러 교수가 담당할 수 있다.

수강과목 (학번, 과목, 교수)

(학번, 과목) → 교수

교수 → 과목

```
graph LR; subgraph Table [수강과목]; direction TB; A[학번]; B[과목]; end; C[교수]; A --> C; C --> B;
```

<수강과목 테이블 함수 종속 다이어그램>



<수강과목 테이블>

<u>학번</u>	<u>과목</u>	교수
100	프로그래밍	P1
100	데이터베이스	P2
200	프로그래밍	P1
200	데이터베이스	P3
300	데이터베이스	P3
300	프로그래밍	P4

- 수강과목 테이블은 1NF, 2NF, 3NF이나 BCNF는 아니다. 왜냐하면 (학번, 과목)과 교수 컬럼이 결정자이지만 교수 컬럼이 기본키로 취급되고 있지 않기 때문이다.
- 수강과목 테이블에서 발생하는 이상 현상은 다음과 같다.

삽입 이상	- 교수 P5가 데이터베이스를 담당하게 되었을 때 이 사실만을 삽입할 수 없다. 적어도 수강 학생이 한 사람 있어야 가능하다.
삭제 이상	- 학번이 100인 학생이 데이터베이스 과목을 취소하여 테이블로부터 행이 삭제된다면 교수 P2가 데이터베이스 과목을 담당하고 있다는 정보마저 없어지게 되는 삭제 이상이 발생된다.
갱신 이상	- 교수 P1의 담당과목이 프로그래밍에서 데이터베이스로 변경된다면 P1에 해당되는 모든 행을 변경해 주어야 한다. 그렇지 않으면 교수는 한 과목만 담당하기로 되어 있으나 사실과 달리 P1이 두 과목을 담당한 것이 되어 데이터 모순이 발생한다.

- 이와 같은 이상 현상의 원인은 교수 컬럼이 결정자이지만 기본키로 취급되고 있지 않기 때문이다.
- 문제 해결을 위해 다음과 같이 수강교수 테이블과 과목교수 테이블로 분해하였다.

<p>[테이블1] 수강교수 (학번, 교수(FK))</p> 	<p>[테이블2] 과목교수 (교수, 과목)</p>  <p><과목교수 테이블 함수 종속 다이어그램></p>
--	---

- 수강교수 테이블과 과목교수 테이블은 제3정규형에서 일어날 수 있는 이상 현상을 제거한 BCNF에 속한다.

2001020205_16v4 데이터 입출력 구현

(8) 제4정규형(4NF)

어떤 테이블이 BCNF(강한 제3정규형)에 속하고 모든 다치 종속(MVD)이 함수 종속이면 제4정규형에 속한다.

<개설과목 테이블>

과목	교수	교재
프로그래밍	P1	T1
프로그래밍	P1	T2
프로그래밍	P2	T1
프로그래밍	P2	T2
데이터베이스	P3	T3
데이터베이스	P3	T4
데이터베이스	P3	T5

- 개설과목 테이블은 다음과 같은 갱신 이상이 발생할 수 있다.
- 데이터베이스 과목을 새로운 교수 P4가 담당하게 되었다는 정보를 추가하려 한다면, 교재 (T3, T4, T5)에 대해 모두 3개의 새로운 행을 삽입해야만 하는 문제가 생긴다.
- 그럼에도 개설과목 테이블은 BCNF에 속한다. 왜냐하면 이 테이블의 기본키는 (과목, 교수, 교재)이므로 결정자가 모두 기본키인 BCNF에 속한다.
- BCNF인데도 갱신 이상 문제가 발생하는 이유는 교수와 교재 컬럼이 서로 무관한데 하나의 테이블로 묶어서 표현한데 원인이 있다.
- 개설과목 테이블에는 함수 종속은 없으나 다치 종속은 존재한다.
- **다치 종속이란?**

A, B, C 3개의 컬럼을 가진 테이블 R에서 A 값에 대응되는 B의 값 집합이 A값에만 종속되고 C값에는 독립적이면 B는 A에 다치 종속이라 하고 $A \twoheadrightarrow B$ 로 표기한다.

다치 종속은 컬럼이 적어도 3개 이상인 테이블에서만 존재할 수 있다.
또 테이블 R(A, B, C)에서 다치 종속 $A \twoheadrightarrow B$ 가 성립하면 $A \twoheadrightarrow C$ 도 동시에 성립한다.
그래서 $A \twoheadrightarrow B \mid C$ 로 한꺼번에 표현하기도 한다.

[테이블] 개설과목
과목 \rightarrow 교수
과목 \rightarrow 교재
또는
과목 \rightarrow 교수 | 교재

- 즉, 과목 \rightarrow 교수에서 과목 "프로그래밍"은 교수 컬럼 값의 집합 (P1, P2)와 대응하고 과목 \rightarrow 교재에서 과목 "프로그래밍"은 교재 컬럼 값의 집합 (T1, T2)와 대응한다.
- 또한 "교수" 컬럼과 "교재" 컬럼은 사실상 서로 독립적이다.
- 그러므로 개설과목 테이블에는 과목 \rightarrow 교수 | 교재 다치종속이 존재한다.
- 모든 함수 종속은 전부 다치 종속에 속한다. 즉, 함수 종속은 종속 값 집합의 원소 수가 언제나 하나인 특수한 다치 종속인 것이다.
- 개설과목 테이블에서 일어나는 문제는 함수 종속이 아닌 다치 종속을 갖고 있기 때문에 발생된다. 다치 종속을 가진 테이블의 문제를 해결하기 위해 분해 작업을 해야 한다.

2001020205_16v4 데이터 입출력 구현

- Fagin의 정리

테이블 R(A, B, C)에서 MVD $A \twoheadrightarrow B \mid C$ 가 존재하면 R1(A, B)와 R2(A, C)로 무손실 분해될 수 있다.

- 이처럼 BCNF인 테이블에 다치 종속이 존재하는 경우 다치 종속을 제거하면 제4정규형에 속한다.
- 개설과목 테이블의 다치 종속을 제거하기 위해 분해한 결과는 다음과 같다.

[테이블1] 과목교수		[테이블2] 과목교재	
과목	교수	과목	교재
프로그래밍	P1	프로그래밍	T1
프로그래밍	P2	프로그래밍	T2
데이터베이스	P3	데이터베이스	T3
		데이터베이스	T4
		데이터베이스	T5

- 과목교수 테이블과 과목교재 테이블은 4NF에 속한다.

(9) 제5정규형(5NF)

어떤 테이블에 존재하는 모든 조인 종속(JD)이 기본키를 통해서만 성립된다면 제5정규형에 속한다.

[테이블] 학생 (학번, 이름, 학과, 학년)	
<p>[ex1] 학생학번 (학번, 이름, 학과) 학생학년(학번, 학년)</p> <p>↓</p> <p>기본키 학번 컬럼을 통해 조인하면 학생 테이블과 동일함.</p>	<p>[ex2] 학번이름 (학번, 이름) 학번학년 (학번, 학년) 학번학과 (학번, 학과)</p> <p>↓</p> <p>기본키 학번 컬럼을 통해 조인하면 학생 테이블과 동일함.</p>

- 학생 테이블은 [ex1]과 [ex2]의 조인 종속이 존재한다. 그러므로 위 경우와 같이 여러 가지 방법의 무손실 분해가 가능하다. 그러므로 학생 테이블은 5NF에 속한다.
- 5NF에 속하는 학생 테이블을 조인 종속을 기준으로 분해를 할 것인지는 여러 장/단점이 있을 수 있으므로 정해진 것은 없다.

(10) 정규화 수준이 높을수록 장단점

장점	1) 유연한 데이터 구축이 가능 2) 데이터의 정확성 높아짐
단점	1) 물리적 접근이 복잡 2) 길이가 짧은 데이터 생성으로 과도한 조인 발생

9. 정규화 과정 정리



2. 물리 데이터저장소 확인하기

학습목표

- 논리 데이터저장소 설계를 바탕으로 응용소프트웨어가 사용하는 데이터저장소의 특성을 반영한 물리 데이터저장소 설계를 수행할 수 있다.
- 논리 데이터저장소 설계를 바탕으로 목표 시스템의 데이터 특성을 반영하여 최적화된 물리 데이터저장소를 설계할 수 있다.
- 물리 데이터저장소 설계에 따라 데이터저장소에 실제 데이터가 저장될 물리적 공간을 구성할 수 있다.

1. 반정규화(Denormalization)

(1) 정의

정규화에 충실하여 모델링을 수행하면 종속성, 활용성은 향상되나 수행속도가 증가하는 경우가 발생하여 이를 극복하기 위해 성능에 중점을 두어 정규화하는 방법

(2) 특징

- (가) 데이터 모델링 규칙에 얽매이지 않고 수행한다.
- (나) 시스템이 물리적으로 구현되었을 때 성능향상을 목적으로 한다.

(3) 사용 시기

- (가) 정규화에 충실하였으나 수행속도에 문제가 있는 경우
- (나) 다량의 범위를 자주 처리해야 하는 경우
- (다) 특정범위의 데이터만 자주 처리하는 경우
- (라) 처리범위를 줄이지 않고는 수행속도를 개선할 수 없는 경우
- (마) 요약 자료만 주로 요구되는 경우
- (바) 추가된 테이블의 처리를 위한 오버헤드를 고려하여 결정
- (사) 인덱스의 조정이나 부분범위처리로 유도하고, 클러스터링을 이용하여 해결할 수 있는지를 철저히 검토 후 결정

(4) 반정규화 유형

중복 테이블 추가	<p>(1) 용도</p> <ul style="list-style-type: none"> (가) 다량의 범위를 자주 처리하는 경우 (나) 특정 범위의 데이터만 자주 처리되는 경우 (다) 처리범위를 줄이지 않고는 수행속도를 개선할 수 없는 경우 <p>(2) 방법</p> <ul style="list-style-type: none"> (가) 집계 테이블의 추가 활용하고자 하는 집계정보를 위한 테이블을 추가하고, 각 원본테이블에 트리거를 등록시켜 생성하여 활용하는데, 이때 트리거의 오버헤드에 유의해야 한다. (나) 진행 테이블의 추가 이력관리 등의 목적으로 사용되며 활용도가 좋아지도록 기본키를 적절히
-----------	---

	<p>설정하여야 한다.</p> <p>(다) 특정 부분만을 포함하는 테이블 추가</p> <p>거대한 테이블의 특정 부분만을 사용하는 경우 자주 사용되는 부분으로 새로운 테이블 생성하여 활용한다.</p>
테이블 조합	<p>(1) 용도</p> <p>대부분 처리가 두 개 이상의 테이블에 대해 항상 같이 일어나는 경우에 활용한다.</p> <p>(2) 방법</p> <p>해당 테이블을 통합하여 설계한다.</p> <p>(3) 고려사항</p> <p>(가) 데이터 액세스가 보다 간편하지만 Row수가 증가하여 처리량이 증가하는 경우가 발생할 수 있으므로 이를 고려해야 한다.</p> <p>(나) 입력, 수정, 삭제 규칙이 복잡해질 수 있음에 유의해야 한다.</p> <p>(다) Not Null, Default, Check 등의 Constraint를 완벽히 설계하기 어려운 점이 있다.</p>
테이블 분할	<p>(1) 용도</p> <p>(가) 칼럼의 사용빈도의 차이가 많은 경우</p> <p>(나) 각각의 사용자가 각기 특정한 부분만 지속적으로 사용하는 경우</p> <p>(다) 상황에 따라 SUPER-TYPE을 모두 내려 SUB-TYPE 별로 분할하거나 SUPER-TYPE만은 따로 테이블을 생성하는 경우</p> <p>(2) 방법</p> <p>(가) 수직 분할</p> <p>칼럼별 사용빈도의 차이가 많은 경우 자주 사용되는 칼럼들과 그렇지 않은 칼럼으로 분류하여 테이블을 분할하는 방법이다.</p> <p>(나) 수평 분할</p> <p>특정 범위별 사용 빈도의 차이가 많은 경우 해당 범위 별로 테이블을 분할하는 방법이다.</p> <p>(3) 고려사항</p> <p>(가) 특정 칼럼 또는 범위를 사용하지 않는 경우 수행속도에 많은 영향이 있음을 고려해야 한다.</p> <p>(나) 기본키의 유일성 관리가 어려워진다.</p> <p>(다) 액세스 빈도나 처리할 데이터양이 적은 경우는 분할이 불필요함을 고려하여야 한다.</p> <p>(라) 분할된 테이블은 오히려 수행속도를 나쁘게 하기도 함에 유의하여야 한다.</p> <p>(마) 데이터 프로세싱 관점이 아니라 검색에 중점을 두어 결정하여야 한다.</p>
테이블 제거	<p>(1) 용도</p> <p>테이블 재정의나 칼럼의 중복화로 더 이상 액세스 되지 않는 테이블 발생할 경우</p>

	<p>(2) 방법 해당 테이블을 삭제한다.</p> <p>(3) 고려사항 (가) 관리 소홀로 인해, 누락 시 유지보수 단계에서 많이 발생하는 현상임을 고려해야 한다. (나) 유지보수 단계에서 초기 설계 시 예상하지 못했던 새로운 요구사항이 증가하게 되면 눈앞의 해결에만 급급하여 테이블의 추가나 변경이 함부로 일어나게 되어 시스템은 일관성과 통합성이 무너지게 된다는 사실을 간과해서는 안된다.</p>
컬럼의 중복화	<p>(1) 용도 (가) 자주 사용되는 칼럼이 다른 테이블에 분산되어 있어 상세한 조건에도 불구하고 액세스 범위를 줄이지 못하는 경우 (나) 대량 데이터에서 Row별 연산 결과를 얻고자 할 때 성능향상을 위한 파생(Derived) 칼럼을 추가할 경우 (다) 기본키의 형태가 적절하지 않거나 너무 많은 칼럼으로 구성된 경우 (라) 정규화 규칙에 얽매이지 않으면서 성능향상을 목적으로 한 반정규화를 통한 중복 데이터를 허용하는 경우</p> <p>(2) 방법 필요한 해당 테이블이나 칼럼을 추가한다.</p> <p>(3) 고려사항 (가) 테이블 중복과 칼럼의 중복을 고려한다. (나) 데이터 일관성 및 무결성에 유의해야 한다. (다) SQL Group Function을 이용하여 해결 가능한지 검토한다. (라) 저장 공간의 지나친 낭비를 고려해야 한다.</p>

2. 테이블 제약조건

(1) Primary key

- 기본키 제약조건, 테이블 당 한번 선언 가능
- Not null과 Unique의 성격을 모두 가지고 있는 제약조건

(2) Not Null

- Null값이 삽입되는 것을 막아주는 제약조건
- 컬럼 레벨의 문법으로만 정의 가능

(3) Unique

- 중복된 데이터가 삽입되는 것을 막아주는 제약조건

(4) Check

- 해당 컬럼이 만족해야하는 조건문을 직접 정의

(5) Foreign key

- 외래키 제약조건
- 자기 자신 테이블이나 다른 테이블의 특정 컬럼(PK, UK)을 참조하는 제약조건
- 옵션 : on delete cascade, on delete not null

예제1

[테이블1] emp_test1

empno	ename	job	sal	deptno
100	Kim	sa	300	10
101	Hong	st	250	20
102	Han	ad	400	30
103	Lim	mk	450	10
104	Jeon	ad	150	40
105	Park	ad	180	20

[테이블2] dept_test1

deptno	dname
10	Account
20	IT
30	Sales
40	Testing

// dept_test1 테이블 생성

```
SQL> create table dept_test1
      (deptno number(10) constraint d1_dno_pk primary key,
       dname varchar2(10));
```

// dept_test1 테이블에 4개의 행 삽입

```
SQL> insert into dept_test1 values (&dno, '&dna');
```

// emp_test1 테이블 생성 – FK 제약조건 정의 시 옵션 없음

```
SQL> create table emp_test1
      (empno number(10) constraint e1_eno_pk primary key,
       ename varchar2(10),
       job varchar2(10),
       sal number(10),
       deptno number(5) constraint e1_dno_fk references dept_test1(deptno));
```

// emp_test1 테이블에 6개의 행 삽입

```
SQL> insert into emp_test1 values (&eno, '&ena', '&job', &sal, &dno);
```

// dept_test1 테이블의 30번 부서 삭제 작업

```
SQL> delete from dept_test1 where deptno = 20;
```

=> FK 제약조건 위배 오류 발생!

FK 제약조건 정의 시 옵션 지정하지 않은 경우 자식보다 부모가 먼저 삭제될 수 없음.

예제2

[테이블1] emp_test2

empno	ename	job	sal	deptno
100	Kim	sa	300	10
101	Hong	st	250	20
102	Han	ad	400	30
103	Lim	mk	450	10
104	Jeon	ad	150	40
105	Park	ad	180	20

[테이블2] dept_test2

deptno	dname
10	Account
20	IT
30	Sales
40	Testing

// dept_test2 테이블 생성

```
SQL> create table dept_test2
      (deptno number(10) constraint d2_dno_pk primary key,
       dname varchar2(10));
```

// dept_test2 테이블에 4개의 행 삽입

```
SQL> insert into dept_test2 values (&dno, '&dna');
```

// emp_test2 테이블 생성 – FK 제약조건 정의 시 on delete set null 옵션 부여

```
SQL> create table emp_test2
      (empno number(10) constraint e2_eno_pk primary key,
       ename varchar2(10),
       job varchar2(10),
       sal number(10),
       deptno number(5) constraint e2_dno_fk references dept_test2(deptno)
        on delete set null);
```

// emp_test2 테이블에 6개의 행 삽입

```
SQL> insert into emp_test2 values (&eno, '&ena', '&job', &sal, &dno);
```

// dept_test2 테이블의 30번 부서 삭제 작업

```
SQL> delete from dept_test2 where deptno = 20;
```

```
SQL> select * from dept_test2;    => 20번 부서 삭제됨
```

```
SQL> select * from emp_test2;    => 20번 부서에 소속된 직원들의 부서 번호가 null로 바뀜.
```

예제3

[테이블1] emp_test3

empno	ename	job	sal	deptno
100	Kim	sa	300	10
101	Hong	st	250	20
102	Han	ad	400	30
103	Lim	mk	450	10
104	Jeon	ad	150	40
105	Park	ad	180	20

[테이블2] dept_test3

deptno	dname
10	Account
20	IT
30	Sales
40	Testing

// dept_test3 테이블 생성

```
SQL> create table dept_test3
      (deptno number(10) constraint d3_dno_pk primary key,
       dname varchar2(10));
```

// dept_test3 테이블에 4개의 행 삽입

```
SQL> insert into dept_test3 values (&dno, '&dna');
```

// emp_test3 테이블 생성 – FK 제약조건 정의 시 on delete cascade 옵션 부여

```
SQL> create table emp_test3
      (empno number(10) constraint e3_eno_pk primary key,
       ename varchar2(10),
       job varchar2(10),
       sal number(10),
       deptno number(5) constraint e3_dno_fk references dept_test3(deptno)
       on delete cascade);
```

// emp_test3 테이블에 6개의 행 삽입

```
SQL> insert into emp_test3 values (&eno, '&ena', '&job', &sal, &dno);
```

// dept_test3 테이블의 30번 부서 삭제 작업

```
SQL> delete from dept_test3 where deptno = 20;
```

```
SQL> select * from dept_test3;    => 20번 부서 삭제됨
```

```
SQL> select * from emp_test3;    => 20번 부서에 소속된 직원들도 연쇄적으로 삭제됨.
```

3. 인덱스 설계

(1) 인덱스 컬럼 선정

- Primary key와 Unique 제약조건이 선언된 컬럼은 자동으로 인덱스가 생성되어 있음.
- 조회 및 출력 조건으로 자주 사용되는 컬럼
- 가능한 한 수정이 빈번하게 발생하지 않는 컬럼
- 자주 조합되어 사용되는 컬럼은 결합 인덱스로 생성하여 활용한다.
- 결합 인덱스는 구성되는 컬럼 순서에 유의해야 한다.

(2) 인덱스 설계 시 고려사항

- 인덱스는 테이블과 마찬가지로 저장공간을 필요로 하므로 지나치게 많은 인덱스는 공간 낭비가 발생할 수 있다.
- 수정이 빈번하게 발생하는 컬럼의 인덱스는 오버헤드로 작용할 수 있다.
- 테이블 규모가 작은 경우 오히려 인덱스가 기존 액세스 경로에 영향을 미칠 수 있다.

4. 뷰 설계

(1) 뷰 속성

- REPLACE: 뷰가 이미 존재하는 경우 재생성
- FORCE: 기본 테이블(Base table)의 존재 여부에 관계없이 뷰 생성
- NOFORCE(default): 기본 테이블(Base table)이 존재할 때 만 뷰 생성
- WITH CHECK OPTION: Subquery 내의 조건을 만족하는 행만 뷰를 통한 변경 가능

예제1

[View] deptv30

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
114	Raphaely	11000	30
115	Khoo	3100	30
116	Baida	2900	30
117	Tobias	2800	30
118	Himuro	2600	30
119	Colmenares	2500	30

// deptv30 뷰 생성

```
SQL> create view deptv30
      as select employee_id, last_name, salary, departmet_id
      from employees
      where department_id = 30;
```

```
SQL> select * from deptv30;
```

// deptv30 뷰를 통한 update 작업

```
SQL> update deptv30
      set department_id = 40
      where employee_id = 115;
```

```
SQL> select * from deptv30;
```

```
SQL> rollback;
```

예제2

[View] deptv30_a

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
114	Raphaely	11000	30
115	Khoo	3100	30
116	Baida	2900	30
117	Tobias	2800	30
118	Himuro	2600	30
119	Colmenares	2500	30

// deptv30_a 뷰 생성 – with check option 옵션 부여

```
SQL> create view deptv30_a
      as select employee_id, last_name, salary, department_id
      from employees
      where department_id = 30 with check option;
```

```
SQL> select * from deptv30_a;
```

// deptv30_a 뷰를 통한 update 작업

```
SQL> update deptv30_a
      set department_id = 40
      where employee_id = 115; => with check option에 위배되어 오류 발생!
```

```
SQL> update deptv30_a
      set salary = 4800
      where employee_id = 115; => salary 값은 업데이트 가능함.
```

```
SQL> select * from deptv30_a;
```

```
SQL> commit;
```

2001020205_16v4 데이터 입출력 구현

- WITH READ ONLY: 뷰를 통한 DML 작업 불가, 읽기 전용 뷰 생성

예제			
[View] deptv30_b			
EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
114	Raphaely	11000	30
115	Khoo	3100	30
116	Baida	2900	30
117	Tobias	2800	30
118	Himuro	2600	30
119	Colmenares	2500	30


```
// deptv30_b 뷰 생성 - with read only 옵션 부여
SQL> create view deptv30_b
  as select employee_id, last_name, salary, department_id
  from employees
  where department_id = 30 with read only;

SQL> select * from deptv30_b    => 읽기는 가능함.

// deptv30_b를 통한 update 작업
SQL> update deptv30_b
  set department_id = 40
  where employee_id = 115;    => 읽기 전용이므로 변경 작업(DML) 안됨.
```

(2) 뷰 설계 시 고려사항

- 최종적으로는 테이블을 액세스하는 것이므로 사용에 따라 수행속도에 문제가 발생할 수 있다.