2️⃣

SQL_이론

1. DB 개요

DB	운영데이터의 집합
DBMS	db 관리 sw
DB system	db, dbms, 사용자 모두 포함
data language	query language(ex)sql)

- database administrator(DBA) : db관리자 → 데이터베이스 설치, 관리, 설계
- data modeling(모형) : db 디자인, 설계
- database tuning & monitoring
    - tuning : 가장 마지막에 db성능 높이기
        1. schema tuning : db구조 결과(schema) 뜯어고침
        2. index tuning : index 설치 x → 성능 낮춤 → index(데이터 빨리 찾게 해줌)
        3. query tuning : 잘못된 지리 고침
    - monitoring : 성능살피기,db사용하여 데이터 사용하는 기본단위
- DBMS (database management system)
    - 정의 : db를 프로그램해주는 시스템 소프트웨어 (ex) Linux, DB engine)
    - 특징 예시 : IBM DB2(세계 최초 main frame), Oracle(DB2와 호환적이다)

      [기능-정의(definition)]
        - create database and table : db와 table만든다
        - create supporting structures(indexes) : schema 구조 만든다
        - read database data : db에 데이터를 읽는다

      [기능-조작(manipulation)]
        - modify(insert, update, delete) database data : db 데이터 수정
        - maintain db structures : db구조 유지

      [기능 -제어(control)]
        - enforce rules : 무결성 규칙(ex)성적(0~100점) → 200점은 잘못된 값)
        - control concurrency : 동시성 제어
        - provide security : 안전 제공
        - integrity (정확성)
        - perform backup and recovery : 백업과 리커버리 수행
- database applications
    - application : 컴퓨터 프로그램
        - create and process forms : 화면을 만들고 진행
        - process user queries : 사용자 쿼리를 진행
        - create and process reports : 출력을 만들고 진행
        - execute application logic : 계산 실행
        - control application : 컴퓨터 프로그램 통제
- DB(database)

- 정의
  - 조직의 모든 응용 프로그램이 공유하기 위해 통합 저장한 운영 데이터
    - 응용 : 인사관리, 회계관리, 계정관리, 수강신청
    - 공유 : 동시 공유(concurrent sharing) ↔ 순차 공유(sequential sharing)
    - 통합 : 모든 데이터 파일 통합 → 중복 제거
    - 저장 : 디스크에 저장
    - 운영 데이터 : 조직의 운영에 필수적으로 필요한 데이터
  - self-describing collection of data
    - data와 schema(meta-data) 모두 보유(여러 응용 통합하여 묶음)
    - db구조 정보(schema)도 db에 존재(schema=meta-data)
    - db자체가 자기자신의 구조를 안다.
- 특징
  - db의 목적은 변경된 정보를 추적하여 볼 수 있도록 하는 것
  - 데이터는 tables(행과 열로 구성)에 저장
  - 테이블의 각 행은 데이터를 occurence(발생)이나 instance에 대해 저장
  - db는 데이터와 관계모델(relational)을 저장 → E-R model의 개체-개체 관계
  - 테이블 이름은 모두 대문자로 쓰임
  - 칼럼 이름은 첫번째 글자가 대문자고 나머지는 소문자
  - data=기록된 사실이나 수치, information=데이터에서의 지식
  - 데이터베이스는 데이터를 기록하기도 하고 데이터에 대해 정보를 생산할 수 있도록함
- Database System의 구성요소
  - user(=end user) → database application → dbms → database

    ex) database application = ERP(인사, 회계 관리),CRM(고객 관리), OLTP(비행기표 예약)

    dbms = create(select, insert, delete, update), process, administer
  - with sql
    - user(=end user) → database application → sql → dbms → database
- DB 성능 단위
  - db처리 단위 : transaction
- contents
  - tables of user data : user데이터에 대한 table
  - metadata : schema 정보
  - indexes : b-tree는 p값으로 record값 찾음

    (logmn : m은 차수/index노드 자식 수, n은 데이터블록수)

    → overhead : 부담 → disk 많이 차지 (data space < index space)
  - stored procedures : 데이터 빠르게 실행
  - triggers : 자동 실행 program / 어떤 조건 만족하도록 → 데이터 연기할 때 쓰임
  - security data
  - backup/recovery data : data fail했을 때→ log = transaction이 변경할 것 기록
- microsoft access : dbms + application generator
  - singer user / terminal table aksemfa
  - microsoft access < microsoft sql server < ibm oracle corp. db2 oracle
    - 파워/특징, 사용의 어려움

    - user
        - casual user(end user) : canned application, interactive sql 이용
        - application programmer : esql/c < odbc (호환성, 인식성 더 good_) - 동일한 기능의 program
        - dba : db관리, 모든 권한 가짐
        - db designer : ER modeling 수행
    - db 변천과정
        - rdb 관계 db (oracle, db2) → oodb 객체지향 db (object 지원) → ordb 객체관계 db (table, object 모두 지원) → bigdatabase(nosql → not only sql, 대용량 분산 db)
- the relational db model
    - 모든 최근 dbms는 이것에 기반함
    - ibm engineer E.F Codd가 1990년대에 창조
    - relational algebra라는 수학이 기반

2. SQL 개요

- 데이터 언어
    - db를 정의, 조작, 제어하기 위해 사용
    - 종류
        - DDL : 데이터 정의어, 스키마 정의
        - DML : 데이터 조작어, 데이터 검색, 삽입, 삭제, 갱신
        - DCL : 데이터 제어어, 회복, 병행제어, 무결성, 보안관리
        - 관계 대수 기반 : dbms 내부용, 질의 최적화 등 수행에 사용
        - 관계 해석 기반 : 투플 해석식, 도메인 해석식
        - 관계 매핑 기반 : sql → ibm db2 (sql,sequel을 ansi(nist), iso로 표준화)
    - sql 언어 문장
        - DDL : 스키마(table)
            - create table, alter table, drop table → alter(구조 변경, 칼럼 추가/변경)
            - create view, drop view → 사용의 편의성(꼭 필요/virtual table), 보안성(physical table위에 virtual table)
            - create index, drop index → 장점)검색 빨라짐(b_tree), 단점)storage overhead/삽입, 삭제 느려짐 → roof mode balancing
        - DML
            - select from where 검색
            - insert into 삽입
            - delete from 삭제
            - update set 갱신
        - DCL
            - 주로 데이터 제어용, dba나 응용 프로그래머가 사용
            - 보안, 권한 부여 grant to, revoke from
            - 동시성 제어 lock, unlock ↔ dead lock
            - 회복 commit (다 끝났을 때 영구적으로 db에 저장), rollback (transaction(지금까지의 update) 취소 → nothing으로)
- SQL history
    - sql은 전체 특징지어있는 programming language가 아니다

    - sql은 processing db data와 metadata를 만들어내는 데이터 부속어(프로그램 안에)
- cape codd retaul sales data extraction (example db_
    - table 3개만 사용 : retail_order(주문정보), order_item(세부항목), sku_data(상품)

```
#상품 테이블에서 부서와 고객정보 추출
select department, buyer
from sku_data;

select buyer, department
from sku_data;

#buyer 중복x
#distinct(sorting해서 중복찾음-sorthing overhead 존재 -> 시간 많이 걸림)
select distinct buyer, department
from sku_data;

#상품 테이블에 모든 정보 조회
select *
from sku_data;

#상품 테이블에서 부서가 Water Sports인 정보 출력
select *
from sku_data
where department='Water Sports';

#상품 테이블에서 부서가 Climbing인 설명서와 고객 출력
select sku_description, buyer
from sku_data
where department='Climbing';

#세부항목 테이블에서 주문번호로 오름차순 -> 가격으로 오름차순
select *
from order_item
order by orderNumber, Price;

select *
from order_item
order by price desc, ordernumber asc;

select *
from sku_data
where department='Water Sports' and buyer='Nancy Meyers';

select *
from sku_Data
where department='Camping' or department='Climbing';

#where in=and=및
select *
from sku_data
where buyer in ('Nancy Meyer','Cindy Lo','Jerry Martin'); #subquery

select *
from sku_data
where buyer not in ('Nancy Meyers','Cindy Lo','Jerry Martin');

select *
```

```
from order_item
where ExtendedPrice between 100 and 200; #100이상 200이하

select *
from order_item
where ExtendedPrice >= 100 and ExtendedPrice <= 200; #100이상 200이하

select *
from sku_data
where buyer like 'Pete%'; #고객의 앞의 이름이 Pete로 시작하는 고객의 모든 정보 조회

select *
from sku_Data
where sku_description like '%Tent%'; #Tent가 들어가는 이름의 설명서의 모든 정보 조회

select *
from sku_data
where sku like '%2__';  #뒤에서 세번째자리가 2인 상품번호의 모든 정보 조회
```

```
##count, sum, avg, min, max
select sum(ExtendedPrice) as Order3000Sum #변수명지정
from order_item
where OrderNumber=3000;  #주문번호가 3000인 세부항목에서 가격의 합 출력

select count(*) as numberofrows
from order_item; #세부항목의 수 출력

select count (distinct department) as DeptCount
from sku_data;  #부서의 중복되지 않게 갯수 출력

select quantity * price as ep, ExtendedPrice
from order_item  #양 곱하기 가격 두 변수에 저장 및 출력

select distinct rtrim (buyer) + ' in ' rtrim(department) as sponsor
from sku_data;   #buyer와 department 칼럼 합침 (rtrim : 오른쪽 공백 제거)

select department, buyer, count(*) as dept_buyer_SKU_count
from sku_data
group by department, buyer; #부서와 고객별로 그룹화한다음 고객의 수 출력
```

- SQL문 순서
  1. select (statement)
  2. from (table)
  3. where
  4. group by (having) (suboption)
  5. order by (sorting)

```
select department, count(*) as dept_SKU_Count
from sku_data
where sku <> 302000
group by department
order by dept_SKU_Count; #지정한 칼럼명으로 오름차순 정렬

select department, count(*) as dept_SKU_Count
from sku_data
where sku <> 302000
```

```
group by department
having count(*)>1
order by Dept_SKU_Count;  #부서별로 사원수가 1보다 큰 부서를 추출하여 상품번호가 302000이 아닌 부서 추출

select sum(extendedPrice) as revenue
from order_item
where sku in (select sku from sku_data where department='Water Sports');
#세부항목 테이블에서 이름이 Water Sports인 부서와 같은 상품번호인 상품번호들의 가격합 추출

select buyer, extendedPrice
from sku_data, order_item
where sku_data.sku=order_item.sku;  #join table (상품번호로 join)

select buyer, sum(extendedPrice) as BuyerRevenue
from sku_data, order_item
where sku_data.sku=order_item.sku #join
group by buyer
order by buyerrevenue desc;

select buyer, extendedPrice, OrderMonth
from sku_data, order_item, retail_order
where sku_data.sku=order_item.sku and order_item.OrderNumber=retail_order.OrderNumber;
```

- subqueries vs joins

  → joins가 상당히 복잡(여러 table에서 원하는 column 가져올 수 있음)

  → subquery 가장 바깥 from절에 table에서만 결과 가져올 수 있음

  → correlated subquery : join에서는 표현 x

3. 관계 모델

- 데이터 모델 (data model)
  - 실세계를 capture하는 도구 (db 설계 도구)
  - 종류
    - 개념적 데이터 모델
      - E-R 모델 : 사용자 요구사항 text로 받아들임/ 명사가 개체
      - semantic network, semantic object model(entity 하나의 object)
    - 논리적 데이터 모델 : detail한 level
      - 관계 데이터 모델 : relation(table) - 실제로 집합(set) >> DB2, Oracle
      - 계층 데이터 모델 : tree 기반 >> IMS
      - 네트워크 데이터 모델 : graph 기반
  - 용도 (db 설계)
    - 개념 데이터 모델 : db 개념적 설계 단계에서 사용 (E-R model 사용)
    - 논리 데이터 모델 : db 논리적 설계 단계에서 사용 (논리 data model 사용)
    - dbms : 모델 구현
  - 예시
    - Order_item(OrderNumber, SKU)

      → OrderNumber은 unique하지 않으므로 단독으로 기본키로 사용할 수 없다.

      → OrderNumber + SKU : 복합키
    - SKU_DATA (SKU : 기본키이면서 외래키)

        - Order_item에 SKU_DATA 합칠 수 있는가 ? → No. SKU_DATA에 OrderNumber 칼럼이 없기 때문에 NULL값을 가지므로 → data lost
- 관계 데이터 모델
    - <S,O,C>로 구성
    - S(Structure) : 구조 → relation, 열(attribute), 행(tuple)
        - tuple의 집합, 카디션 프로덕트의 subset (카디션 프로덕트 : 두 개 테이블의 조합 가능한 모든 행의 쌍)
    - O(Operation) : 연산
        - 관계 대수 (relational algebra)
            - 집합 연산 : 합집합, 교집합, 차집합, 카디션 프로덕트
            - 순수 연산 : selection, projection, join, division
        - 관계 해석 (relational calculus)
            - 튜플 관계 해석
            - 도메인 관계 해석
    - C (Constraints) : 제약 조건
        - 구조적 제약조건
            - 개체 무결성 : primary key(기본키)에 대한 제약, NULL값 가질 수 없음 (NULL값이면 기본키를 통해 tuple 못 찾음)
            - 참조 무결성 : foreign key(외래키)에 대한 제약, FK는 자신이 기본키로 설정된 테이블의 기본키의 값만을 갖음
        - 의미적 제약조건 : 도메인 무결성 (칼럼값 범위 등등)

Important Relational Model Terms

- entity : user가 추적하고 싶은 식별할 수 있는 어떠한 것(data의 집합)이다.
- relation : 2차원의 table
    - 정의

      1. tuple의 집합
      2. domain의 cartesian product의 subset

      (domain : 특정 attribute가 가질 수 있는 값의 집합)

      3. relation scheme + relation instance
          - relation scheme = relation name + attribute name의 set
          - relation instance : 튜플의 집합
          - intension + extension (내포 + 외연)
    - 특성

      1. 모든 tuple은 상이하다 → tuple의 유일성
      2. 동일한 tuple은 존재할 수 없다. (중복 tuple)
      2. 모든 attribute(열)는 유일한 이름을 가진다.
      3. 각 열(column)은 동일한 종류의 값(entity)을 가진다. (각 열은 주어진 domain값 가짐)
      4. 각 열은 원자값(atomic value)을 가진다. → attribute의 원자성 (한 entity안에는 single value여야함)
      5. tuple 순서는 무관, column 순서도 무관 → tuple과 attribute의 무순서성
    - 행 : tuple(relation), record(file)
    - 열 : attribute(relation), field(file)

Key : relation에서 행들을 식별하는데 사용되는 칼럼들

- 성질 : 유일성(uniqueness), 최소성(mininality)
- composite key (복합키) : 두개 이상의 칼럼을 key로 정의하는 것

- super key(슈퍼키) : 테이블에서 각 행을 유일하게 식별할 수 있는 하나 또는 그 이상의 속성들의 집합 (유일성이란 하나의 키로 특정 행을 바로 찾아낼수 있는 고유한 데이터 속성)
- candidate key(후보키) : 테이블에서 각 행을 유일하게 식별할 수 있는 최소한의 속성들의 집합 (유일성과 최소성을 동시에 만족해야함)
- primary key(기본키) : 후보키들 중에서 하나를 선택한 키로 최소성과 유일성을 만족하는 속성 (null 값 X, 중복 X)
  1. relation에서는 오직 한개의 primary key가 있다.
  2. primary key는 composite key가 될 수 있다.
  3. 이상적인 primary key는 짧고, 숫자이고 변하지 않는 것이다.

  → unique, 가능하면 길이가 더 짧은 것, 잘 변하지 않는 것(Never changing), 문자열보단 숫자
- foreign key(외래키) : 외부 릴레이션의 기본 키 애트리뷰트를 참조하는 키이다.

  → 하나의 칼럼이거나 복합키일 수 있다.
- surrogate key(대체키) : 기본키에 적절한 번호가 없을 때, sequence, identity (system이 선정한 번호)로 대체 → 인공적인 칼럼
  1. DBMS가 제공
  2. primary key의 성질과 같음
  3. 사용자에게 의미있는 인공적인 값을 가지고 있음
  4. 폼이나 리포트에서는 대부분 숨겨져있음
- secondary key(보조키) : 기본 키로 선택되지 않은 후보 키 → index 관점 용어

	유일성	최소성
후보키(CK)	o	o
기본키(PK)	o	o
대체키(AK)	o	o
외래키(FK)	x	x
수퍼키(SK)	o	x

- referential integrity constraint(참조 무결성 제약조건) : 두 릴레이션의 연관된 튜플들 사이의 일관성을 유지하는데 사용 두 조건 중 하나만 만족해도 제약조건 만족
  1. 외래 키의 값은 참조 relation의 어떤 tuple의 primary key와 같다.
  2. 외래 키가 자신을 포함하고 있는 relation의 primary key를 구성하고 있지 않으면NULL값을 가진다.
- 차수(degree) : attribute(또는 domain)의 수 = 열의 수
- 카디널리티(cardinality) : tuple의 수 = 행의 수

종류	행	열
table	row	column
relation	tuple	attribute
file	record	field

- functional dependency (함수 종속성)
  - 하나의 attribute기 두번째 attribute를 결정할 수 있을 때 발생
  - X → Y (X : 결정자(determinant), Y : 종속자(dependent))
  - 예시) Student ID → StudentName

    StudentID → (DormName, DormRoom, Fee) : StudentID가 각각을 결정짓는다
  - 함수 종속성은 방정식(equations)에 기반을 둠 → 방정식은 아니다 !
    - Extended Price = Quantity X UnitPrice
    - (Quantity, UnitPrice) → ExtendedPrice : Quantity와 UnitPrice가 동시에 ExtendedPrice를 결정짓는다 → 복합 결정자 (Composite determinant)
    - ObjectColor → Weight

- ObjectColor → Shape
- ObjectColor → (Weight, Shape)
- Composite determinant(복합 결정자) : 함수 종속성의 결정자가 하나 이상의 attribute로 이루어져 있는 것
- 만약 a→(b,c) 이면, a→b 이고 a→c이다. 만약 (a,b)→c이면, a→b, a→c 둘다 아니다

	SKU	SKU_Description	Department	Buyer
1	100100	Std. Scuba Tank, Yellow	Water Sports	Pete Hansen
2	100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen
3	101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
4	101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers
5	201000	Half-dome Tent	Camping	Cindy Lo
6	202000	Half-dome Tent Footprint	Camping	Cindy Lo
7	301000	Light Fly Climbing Harness	Climbing	Jerry Martin
8	302000	Locking carabiner, Oval	Climbing	Jerry Martin

** SKU, SKU_Description : 후보키(unique), 기본키 → 다른 모든 속성 결정 가능

** SKU → (SKU_Description, Department, Buyer)

Buyer → (Department)

	OrderNumber	SKU	Quantity	Price	ExtendedPrice
1	1000	201000	1	300.00	300.00
2	1000	202000	1	130.00	130.00
3	2000	101100	4	50.00	200.00
4	2000	101200	2	50.00	100.00
5	3000	100200	1	300.00	300.00
6	3000	101100	2	50.00	100.00
7	3000	101200	1	50.00	50.00

** OrderNumber, SKU : 복합키, Quantity * Price = ExtendedPrice

** (OrderNumber, SKU) → (Quantity, Price, ExtendedPrice)

(Quantity, Price) → (ExtendedPrice)

- 결정자는 relation에서 유일하고 relation에서 다른 모든 열을 결정하는 경우에만 해당
- 단순히 열에서 고유한 값을 갖는 것만으로는 모든 함수 종속성의 결정자를 찾을 수는 없다.

이상현상

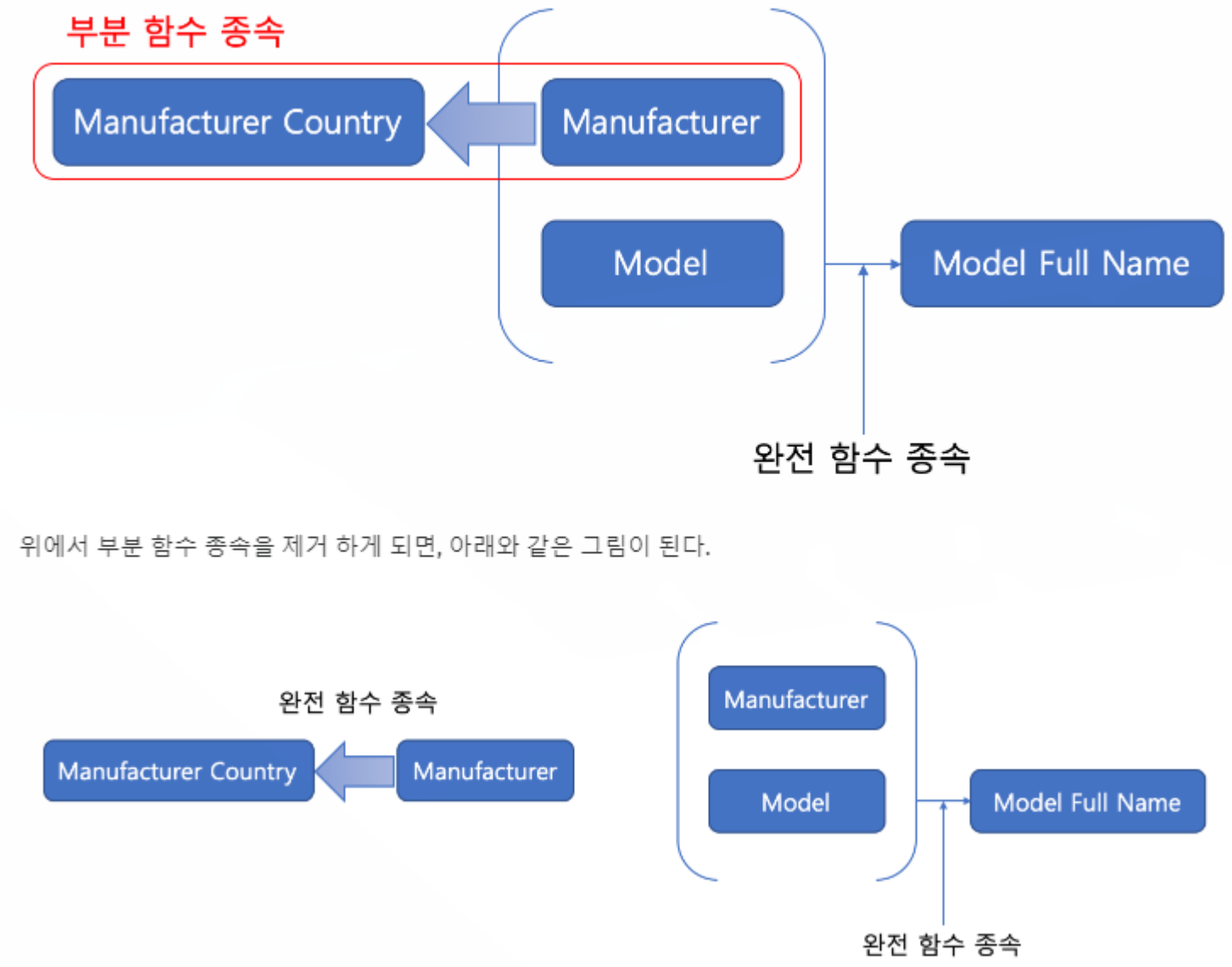
- deletion anomaly : 삭제 이상 현상은 데이터 삭제로 인해 다른 중요한 데이터가 의도하지 않게 손실되는 상황
- update anomaly : 업데이트 이상은 단일 데이터 값을 업데이트할 때 여러 데이터 행을 업데이트
- insertion anomaly : 삽입 이상은 데이터가 부족하여 관계에 새 튜플을 삽입할 수 없는 경우

→ relation은 정규형에 의해 분류된다.

source of Anomaly (이상현상)	normal forms (정규형)	design principles
functional dependencies	1NF, 2NF, 3NF, BCNF	BCNF
multivalued dependencies	4NF	4NF
data constraints and oddities	5NF, DK/NF	DK/NF

Normal Forms

- 1NF
    1. 어떤 Relation에 속한 모든 Domain이 원자값(atomic value)만으로 되어 있다.
    2. 모든 attribute에 반복되는 그룹(repeating group)이 나타나지 않는다.
    3. 기본 키를 사용하여 관련 데이터의 각 집합을 고유하게 식별할 수 있어야 한다.
- 2NF : 키가 아닌 모든 속성이 기본 키에 완전히 기능적으로 종속되는 경우 관계 (부분 함수 종속 제거)



- 3NF : 2NF에 있는 관계이고 primary key를 제외한 결정자가 없는 경우 관계 (이행 함수 종속 제거)
- BCNF(3.5 정규형) : 모든 결정자가 후보키인 관계 (후보키가 아닌 결정자 제거)
- 4NF : 다중 값 종속성이 없는 경우 관계 (다치 종속(MVD) 제거)
- 5NF : 4NF에 있고 조인 종속성을 포함하지 않는 경우 (조인 종속(JD) 제거)
- DK/NF : domain과 key에 대한 제약조건을 모두 반영

#-# 정규화의 핵심 #-#

- 키가 아닌 결정자를 제거
    - 결정자가 키의 일부 → 부분함수 종속
    - 이행 종속 (a→b→c : 여기서 a만 키)
- Kroenke : 하나의 relation이 하나의 주제만 포함하도록 하여라

#-# 정규화예시 #-#

- (SID, Club, Name, Cost, AmountPaid)

-CK : (SID, Club)

-FD : (SID, Club) → AmountPaid

Club → Cost

SID → NAme

-결과 relations

(SID, Club, AmountPaid), (Club, Cost), (SID, Name)

- SKU_DATA(SKU, SKU_Description, Department, DeptBudgetCode, Buyer)

-CK : SKU, SKU_Description

-PK : SKU

-FD : SKU → (SKU_Description, Department, DeptBudgetCode, Buyer)

SKU_Description → (SKU, Department, DeptBudgetCode, Buyer)

Buyer → (Department, DeptBudgetCode)

Department → DeptBudgetCod

-결과 relations

(SKU, SKU_Description, Buyer), (Buyer, Department, DeptBudgetCode), (Department, DeptBudgetcod)

- multivalued dependency(다치 종속성) : 두개의 독립된 애트리뷰트가 1:N 관계로 대응하는 관계로, ' ↠ '화살표로 종속성을 표시
  - entity 분리해도 문제가 사라지지만 불필요하게 회원번호와 이름 애트리뷰트가 중복되는 값이 만들어진다는 단점이 생김
  - 다치 종속성의 결정자는 primary key가 절대 될 수 없다.
  - 다치 종속성은 분리된 relation에 있다면 문제되지 않는다

4. ER모델 - kroenke_dbplle

- Data Model : database design을 위한 계획이나 청사진(도면을 인화한)
  - 데이터 모델은 데이터베이스 디자인보다 더 일반적이고 추상적이다.
  - data model(E-R model)은 DB구조변경에 용이하지만 database design(관계 model)은 schema 단계에서의 구조 변경이 어려움
- E-R Model
  - Entity-Relationship model은 개념적 schema를 생성할 수 있는 개념의 집합
  - ER 모델은 요구사항으로부터 얻어낸 정보들을 개체(Entity), 애트리뷰트(Attribute), 관계성(Relation)으로 기술하는 데이터 모델
- Entities : 사람, 과목 등 식별가능한 어떤 것 그리고 사용자가 DB에 저장하고자하는 주체
  - entity class(set) : 동일한 entity type을 갖는 entity occurence의 집합
  - entity occurence(instance) : 특정 entity의 발생 → 100, kim, 50K
  - entity안에는 대부분 많은 entity의 instance가 있다.
  - entity type : schema : employee
- relationship : entity와 entity와의 관계
  - relationship type : schema : employee와 project간의 N:M 관계
  - relationship occurrence : 100번 employee가 p1 project에 참가, 개체 인스턴스들의 연관
  - relationship set : 모든 직원에 대해 모든 project 참가 정보 모은 것, 개체 클래스들의 연관
  - 종류
    - 참가 개체수(degree)에 따라 → 차수 (관계모델의 차수 , attribute의 개수와는 다름)

    → Unary 단항, 1차

    → Binary 이항

    → Ternary 삼항

    → N-ary N-항, N-차
    - Cardinality에 따라 (관계모델의 cardinality와는 다름)

→ 1:1 일대일 → marry 관계

→ 1:N 일대다 → PM 관계

→ N:M 다대다 → emp-proj, bill of material

- 의미에 따라

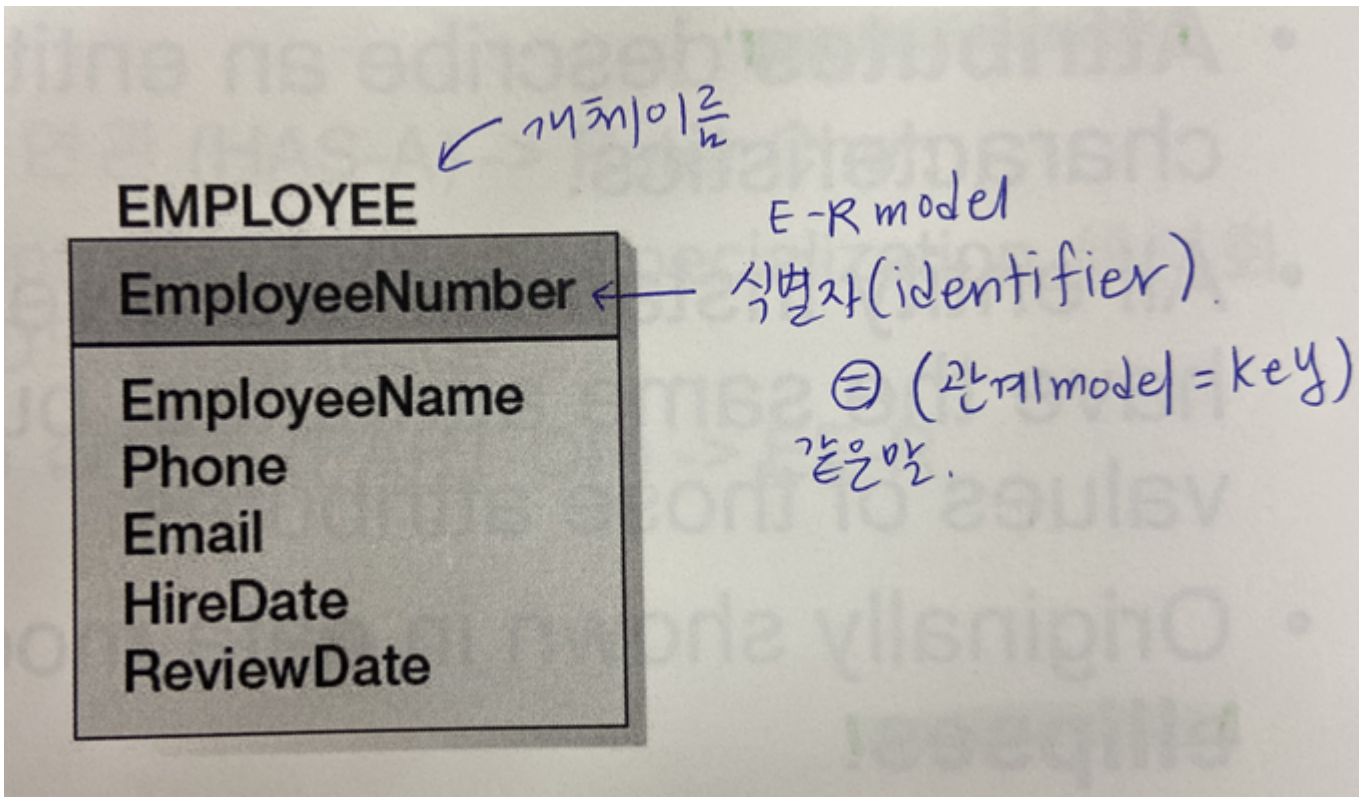
→ association 연관

→ generalization 일반화, specialization 세분화

→ aggregation 집단화

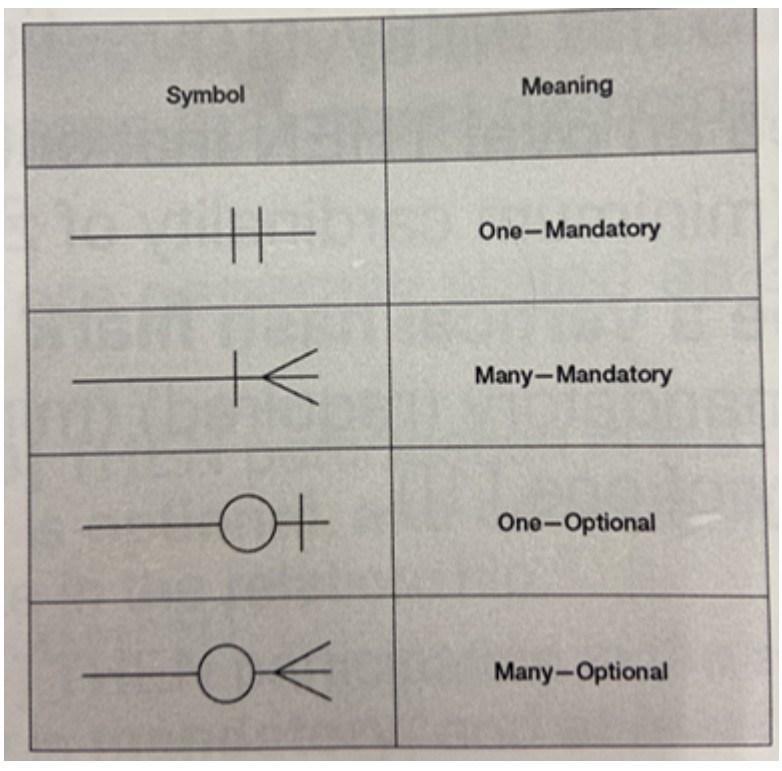
- 원래 E-R model에는 relationship은 속성을 가지고 있어야한다.
- 관계 class는 두개나 더 많은 개체 클라스를 포함할 수 있다.
- degree of the relationship = the number of entity classess
  - 두 개체는 두 차수의 binary relationship
  - 세 개체는 세 차수의 ternary relationship
- HAS-A relationships : 각각의 개체 인스턴스는 다른 개체 인스턴스와 관계를 가지고 있다.

- Attributes(속성)
  - 개체들의 특징을 설명
  - 주어진 개체 집합의 모든 개체 인스턴스들은 같은 속성을 가지고 있지만 이 속성들의 값은 다르다.
  - data model에서는 타원으로 보여짐
  - 오늘날에는 속성은 직사각형으로 표시



- Identifier
  - 이름, 식별할 수 있는, 개체 인스턴스의 속성
  - 개체 인스턴스의 식별자는 개체 속성의 하나 또는 하나 이상으로 구성되어있다.
  - 복합적인 식별자들은 두개나 더 많은 속성들로 구성되어있다.
  - 데이터 모델에서의 식별자는 데이터베이스 디자인에서는 key로 바뀐다.
    - 개체들은 식별자를 가지고 있다
    - 테이블들은 키를 가지고 있다.
- Entity & Tables
  - 개체와 테이블의 차이점은 개체들의 관계는 외래키 없이도 표현할 수 있다는 것
  - 개체들의 존재와 그들간의 관계가 불확실할 때도 early design process를 개체들과 쉽게 할 수 있음

- Cardinality
  - ‘count’ : 이것은 숫자로 표현됨
  - Maximum Cardinality : 개체 인스턴스의 수의 최대값이고 관계에 참여할 수 있다.
    - One-to-One [1:1]
    - One-to-Many [1:N]
      - One에 있는 relationship을 부모 개체 또는 부모라고 한다. Many에 있는 relationship은 자식 개체 또는 자식이라고 한다.
    - Many-to-Many [N:M]
  - Minimum Cardinality : 개체 인스턴스의 수의 최솟값이고 관계에 참여해야만 한다.
    - zero나 one으로 진술됨
    - zero[0]이면 (최소 0개 매칭) 개체에 의한 관계의 참여는 선택적(optional)하다. 그리고 개체 인스턴스가 관계에 참여하면 안된다. 이 선택적인 참여는 선택적인 개체 옆에 타원을 그린다.
    - one[1]이면 개체에 의한 관계의 참여는 의무적(mandatory)이고 적어도 1개의 개체 인스턴스가 관계에 참여하여야 한다. 이 의무적인 참여는 수직 해시마크를 요구 개체 옆에 그린다.
  - Crow’s Foot version

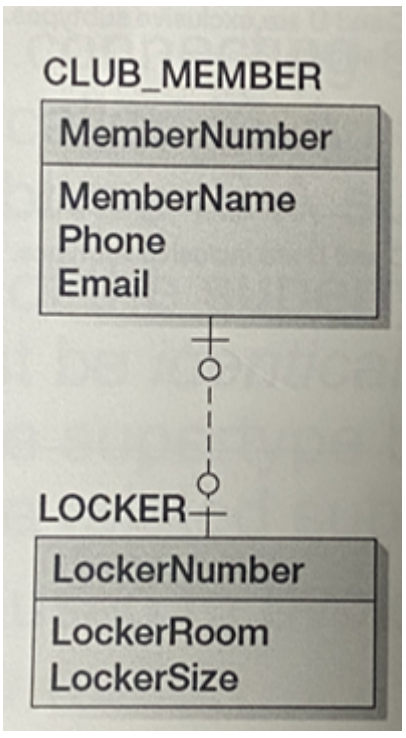


  - ID-dependent entity : 자신의 식별자가 다른 개체의 식별자를 포함하는 개체이다.
  - ID-dependent entity의 부모 최소 커디널리는 항상 1이다.
- Weak entity(약한 개체)
  - 개체의 존재여부가 다른 개체의 존재여부에 달려있을 때
  - 모든 ID-Dependent 개체들은 weak하다.
  - ID-Dependent가 아닌 약간 개체들 또한 존재한다.
  - ID-Dependent → weak (o), weak → ID-Dependent (x)
  - 약한 자식 개체의 식별자의 부모 식별자는 나타나지 않음
- Subtype Entities
  - 기존 entity type에서 중복되는 attribute와 특징이 되는 attribute를 나눠 entity type을 나누는 것
  - supertype entities : Subtype의 일반화된 버전, 상위집단
  - subtype entities : 하위집단
  - 각각의 subtype들은 supertype의 attribute들과 relation들을 상속받는다.
  - super type은 subtype을 나타내는 식별가능한(discriminator) 속성을 가지고 있다.
  - subtype이 exclusive(배타)한 경우 : 하나의 supertype instance는 단 하나의 서브타입과 관계(일대일 관계)가 존재, subtype 부분집합 간에 공통 부분을 갖지 않는 subtype

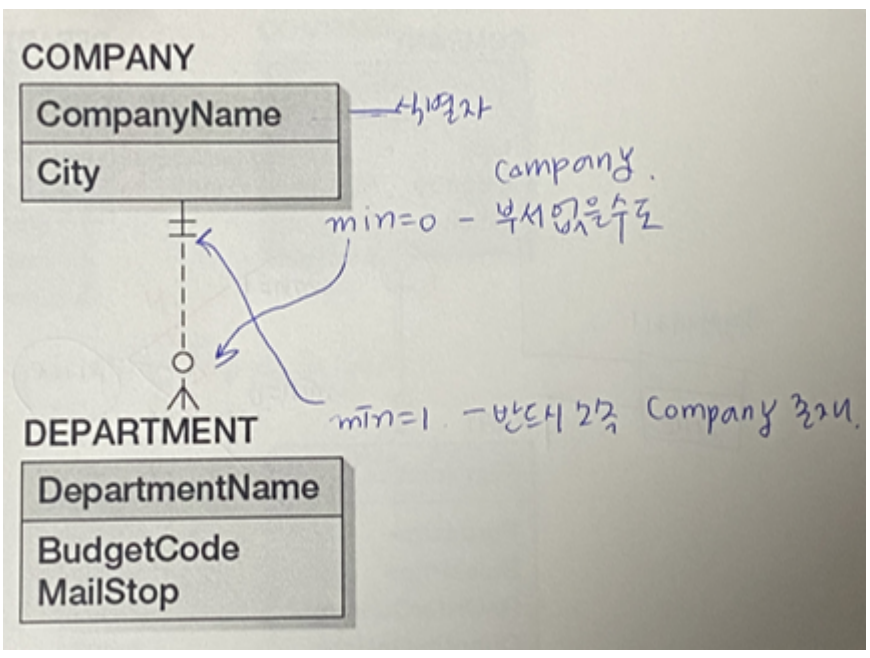
- subtype이 inclusive(중복)한 경우 : 하나의 supertype이 하나나 더 많은 subtype과의 관계 존재
- IS-A relationships : subtype is a supertype
  - supertype의 식별자와 subtype의 모든 것들은 반드시 동일해야한다.
  - subtypes는 부적절한 널값을 피하기 위해 사용

Strong Entity Patterns

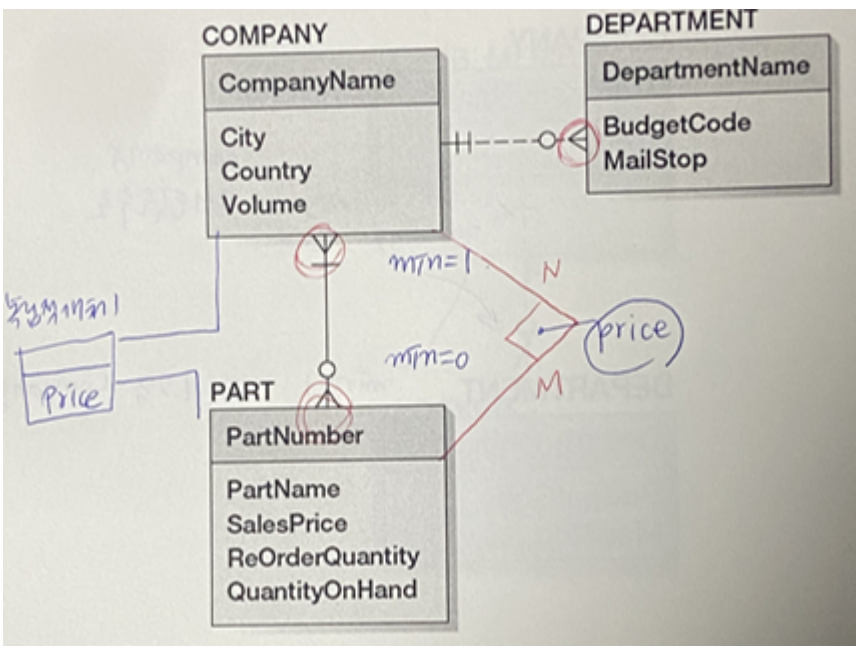
- 1:1



1:N

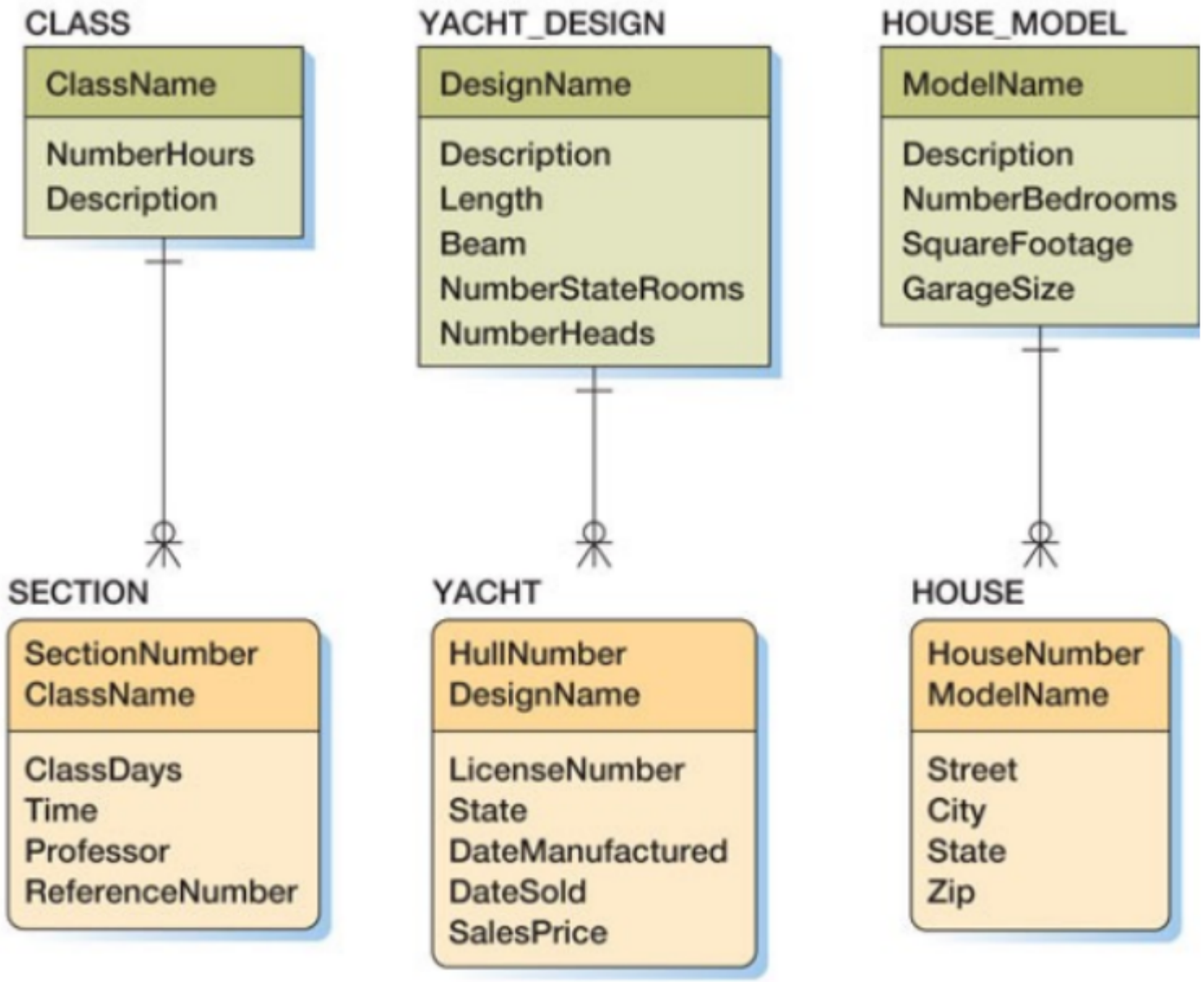


N:M



ID-Dependent Relationships : The archetype/instance pattern

- ID-dependent child 개체가 추상 또는 논리적 parent의 물리적 표현(instance)일 때 발생



→ SECTION, YACHT, HOUSE 는 약한 개체지만 ID 의존 개체 는 아니다.

- Recursive Relationships : 자기자신과의 관계를 갖는 개체가 존재

5. ER-model 관계-model 변환

- entity
  - entity → relation → 정규화 → relations
  - entity → one or more relation
- relationship
  - 1:1 : 한 relation의 PK를 다른 relation의 FK로 추가
  - 1:N : one-side의 PK를 many-side에 FK로 추가
  - N:M : 양쪽 relation의 PK를 새 relation에 FK로 추가
    - 이 FK의 조합이 새 relation의 PK(복합키)
- design for minimum cardinality
  - O-O : 부모 선택적, 자식 선택적
  - M-O : 부모 의무적, 자식 선택적
    - 모든 자식은 부모를 가지고 있다.
    - 실행은 부모가 없는 자식을 절대 만들기 않음
    - FK 칼럼은 NULL값을 가지지 않는다.
  - O-M : 부모 선택적, 자식 의무적
    - department 가 부모이고 employee가 자식이면 deparment 행은 관계가 자식 행에 의해 만들어질 때 만들어진다(trigger 요구됨)

        - 적어도 하나의 자식이 부모에 들어있다.
        - 새로운 employee를 자식에 넣을 수 있다
        - 각각의 부서에 한개의 emplyoyee는 무조건 있어야한다.
            - employee FK는 department에서 가장 최근 employee가 있다면 바꾸거나 삭제할 수 없다.
    - M-M : 부모 의무적, 자식 의무적
- action : 관계에 참여하는 최소한의 개체 수의 강화 액션
- cascading update : 부모 PK가 바뀌면 자식 FK에도 적용됨

  → 대체키는 절대 변하지 않는다. 이것은 cascading update가 필요하지 않다.
- cascading delete : 연관된 자식 행은 부모 행의 제거를 통해 삭제됨
    - 강한 개체는 일반적으로 cascade delete를 하지 않는다.
    - 약한 개체는 일반적으로 cascade delete를 함