**데이터베이스의 개념 및 특징**

1. 통합된 데이터(integrated data)

2. 저장된 데이터(stored data)

3. 운영 데이터(operational data)

4. 공용 데이터(shared data)

5. 실시간 접근(real time accessibility)

6. 계속적인 변화(continued change)

7. 동시 공유(concurrent sharing)

8. 내용에 따른 참조(reference by content)

**데이터베이스 언어**

1. 데이터 정의어 CREATE, ALTER, DROP

2. 데이터 조작어 SELECT, INSERT, DELETE, UPDATE

3. 데이터 제어어 GRANT, REVOKE

**데이터베이스 사용자**

1. 일반 사용자

2. 응용 프로그래머

3. SQL 사용자

4. 데이터베이스 관리자(DBA)

**DBMS의 기능**

1. 데이터 정의

2. 데이터 조작

3. 데이터 추출

4. 데이터 제어

**데이터 모델**

1. 계층 데이터 모델, 네트워크 데이터 모델 – 포인터 사용

2. 관계 데이터 모델 – 속성값 사용 (SQL Server, MySQL)

3. 객체 데이터 모델 – 객체식별자 사용

**3단계 데이터베이스 구조**

1. 외부 단계

2. 개념 단계

3. 내부 단계

4. 외부/개념 매핑

5. 개념/내부 매핑

**데이터 독립성**

논리적 데이터 독립성: 외부단계와 개념 단계 사이의 독립성으로, 개념 스키마가 변경되어도 외부 스키마에는 영향을 미치지 않도록 지원

물리적 데이터 독립성: 개념 단계와 내부 단계 사이의 독립성으로, 저장 장치 구조 변경과 같이 내부 스키마가 변경되어도 개념 스키마에 영향을 미치지 않도록 지원

**관계 데이터 모델**

**릴레이션 스키마와 인스턴스**

1. 릴레이션

2. 릴레이션 스키마

1) 속성: 릴레이션 스카마의 열

2) 도메인: 속성이 가질 수 있는 값의 집합

3) 차수: 속성의 개수

3. 릴레이션인스턴스

1) 투플: 릴레이션의 행

2) 카디날리티: 투플의 수

**무결성 제약 조건 첫번째**

1. **키**: 각 릴레이션의 투플을 유일하게 식별하는 장치이며 동시에 각 릴레이션 간의 관계를 말해 주는 연결고리이다.

2. **수퍼키**: 투플을 유일하게 식별할 수 있는 하나의 속성 혹은 속성의 집합을 말한다.

예) 고객번호, 주민번호 -> (주민번호), (주민번호, 이름), (주민번호, 이름, 주소) ~

3. **후보키**: 투플을 유일하게 식별할 수 있는 속성의 최소 집합이다(효율성)

예) ‘주민번호’ , ‘고객번호’만 후보키가 될 수 있다.

4. **기본키**: 여러 후보키 중 하나를 선정하여 대표로 삼는 키

5. **대리키**: 기본키에 보안이 필요하거나 여러 개의 속성으로 구성되어 복잡하거나. 마땅한 기본키가 없을때는 일련번호 같은 가상의 속성을 만들어 기본키로 삼는 경우

6. **대체키**: 기본키로 선정되지 않은 후보키

7. **외래키**: 다른 릴레이션의 기본키를 참조하는 속성, 다른 릴레이션의 기본키를 참조하여 관계 데이터 모델의 특징인 릴레이션 간의 관계를 표현한다.

**외래키의 특징**

1. 관계 데이터 모델의 릴레이션 간의 관계를 표현

2. 다른 릴레이션의 기본키를 참조하는 속성

3. 참조하고(외래키) 참조되는(기본키) 양쪽 릴레이션의 도메인은 서로 같아야 한다.

4. NULL 값과 중복값 등이 허용된다.

5. 자기 자신의 기본키를 참조하는 외래키도 가능하다.

6. 외래키가 기본키의 일부가 될 수 있다.

**무결성 제약 조건 두번째**

1. 도메인 무결성 제약조건 : 릴레이션 내의 투플들이 각 속성의 도메인에 지정된 값만 가져야 한다는 조건

2. 개체 무결성 제약조건 : 기본키 제약이라고도 한다. 즉, 기본키는 NULL값을 가져서는 안되며 릴레이션 내에 오직 하나의 값만 존재해야 한다. 개체 무결성 제약조건은 참조 무결성 제약조건과 함께 관계 데이터 모델의 핵심 개념

1) 투플을 삽입하거나 수정할 때마다 검사

- 기본키 값이 유일한지

- NULL값이 아닌지

2) 삭제할 때는 즉시 수행

3. 참조 무결성 제약조건 : 외래키 제약이라고도 참조되는 부모릴레이션, 참조하는 자식릴레이션이라고 한다. 참조 무결성 제약조건은 자식 릴레이션의 외래키는 부모 릴레이션의 기본키와 도메인이 같아야 하며, 자식 릴레이션의 값이 변경될 때 부모 릴레이션의 제약을 받는 다는 것이다.

1) 삽입(자식 릴레이션에서)

1. 도메인 무결성 제약조건을 확인
2. 개체 무결성 제약조건 확인

2) 삭제(부모 릴레이션에서)

: 삽입과는 반대로 자식 릴레이션에서 투플이 삭제되는 경우 부모 릴레이션에는 아무런 영향을 주지 않으므로 바로 삭제할 수 있다.

그러나 부모 릴레이션에서 투플을 삭제되는 경우에는 문제가 발생 -> 참조하고 있는 다른 릴레이션이 없는지 찾는다.

1. 즉시 작업을 중지
2. 자식 릴레이션의 관련 투플을 삭제
3. 초기에 설정된 다른 어떤 값으로 변경
4. NULL 값을 설정

**관계대수**

릴레이션에서 데이터를 추출하는 데 사용되는 언어

**관계대수 연산자**

순수 관계연산 : 셀렉션, 프로젝션, 조인, 디비전, 개명

1) 셀렉션 : 투플을 추출하기 위한 연산

2) 프로젝션 : 릴레이션의 속성을 추출하기 위한 연산

3) 조인 : 두 릴레이션의 공통 속성을 기준으로 속성값이 같은 투플을 수평으로 결합하는 연산

조인을 수행하기 위해서는 두 릴레이션의 조인에 참여하는 속성이 서로 동일한 모메인으로 구성

* 세타조인 : 두 릴레이션의 속성값을 비교하여 조건에 맞는 투플을 반환
* 동등조인 : 속성값이 같은 투플만을 반환
* 자연조인 : 속성 이름이 같은 투플만을 대상으로 하되 중복 속성을 제거하고 반환
* 세미조인 : 두 릴레이션을 조인한 후 한 릴레이션의 투플만을 반환
* 외부조인 : 조인에 필패한 투플을 모두 보여 주되, 조인에 실패한 값들을 NULL 값으로 채워서 반환

1\_ 왼쪽 외부조인

2\_ 완전 외부조인

3\_ 오른쪽 외부조인

일반 집합연산 : 합집합, 교집합, 차집합

카티션 프로덕트: 두 릴레이션을 연결시켜 하나로 합칠 때 사용(속성과 도메인이 같을 필요X)

**SQL 기초**

1. SELECT

문제1: 모든 도서의 이름과 가격을 검색하시오.

**SELECT bookname, price**

**FROM Book;**

문제2: 도서 테이블에 있는 모든 출판사를 검색하시오.(중복제거)

**SELECT DISTINCT publisher**

**FROM Book;**

2. WHERE\_조건검색

1) 비교 : =, >, <, <=, >=

2) 범위 : BETWEEN

3) 집합 : IN, NOT IN

4) 패턴 : LIKE

5) NULL : IS NULL, IS NOT NULL

(가끔식 투플값이 비어있는데 NULL조건으로 안나오면 = ‘ ‘ 으로 찾아보기)

6) 복합조건 : AND, OR, NOT

문제1: 출판사가 ‘굿스포츠’ 혹은 ‘대한미디어’인 도서를 검색하시오.

**SELECT \***

**FROM Book**

**WHERE publisher IN (‘굿스포츠’ , ‘대한미디어’);**

3. 검색 결과의 정렬

문제1: 도서를 이름순으로 검색하시오.

**SELECT \***

**FROM Book**

**ORDER BY bookname;**

문제2: 도서를 가격순으로 검색하고, 가격이 같으면 이름순으로 검색하시오.

**SELECT \***

**FROM Book**

**ORDER BY price, bookname;**

문제3: 도서를 가격의 내림차순으로 검색하시오. 만약 가격이 같다면 출판사의 오름차순으로 출력하시오.

**SELECT \***

**FROM Book**

**ORDER BY price desc, publisher asc;**

4. 집계함수와 GROUP BY 검색

1) 집계 함수 : AVG, MIN, MAX, COUNT

문제1: 고객이 주문한 도서의 총판매액을 구하시오. (별칭)

**SELECT SUM(saleprice) AS 총매출**

**FROM Orders;**

문제2: 김연아 고객이 주문한 도서의 총판매액을 구하시오.

**SELECT SUM(saleprice) AS 총판매액**

**FROM Orders**

**WHERE custid = 2;**

문제3: 마당서점의 도서 판매 건수를 구하시오.

**SELECT COUNT(\*)**

**FROM Orders;**

문제4: 가격 8000원 이상인 도서를 구매한 고객에 대해여 고객별 주문 도서의 총수량을 구하시오. 단, 2권 이상 구매한 고객만 구하시오.

**SELECT custid, COUNT(\*) AS 도서수량**

**FROM Orders**

**WHERE saleprice >= 8000**

**GROUP BY custid**

**HAVING BY COUNT(\*) > 2;**

두개 이상 테이블에서 SQL 질의

: 박지성 고객이 주문한 도서의 총구매액을 알고 싶으면 어떻게 해야 할까? 총구매액은 saleprice 값을 합치면 되지만 문제가 있다. 구매액 saleprice는 Orders 테이블에 있고 고객의 이름은 Customer 테이블에 있다. Orders 테이블에는 고객의 이름 ‘박지성’이 없는 것이다. 그렇다면 어떻게 총구매액을 계산할 수 있을까?

앞에서는 하나의 테이블을 이용한 SQL 질의를 배웠다. 이제부터는 두개 이상의 테이블을 이용한 SQL 질의에 대해 알아본다. SQL은 여러 개의 테이블을 이용하여 질의하는 두 가지 방법을 제공한다. **조인(JOIN)과 부속질의(SUBQUERY)**이다. 두 가지 모두 여러 개의 테이블을 동시에 다루지만 방법은 약간 다르다.

Customer와 Orders 테이블의 합체 결과 투플의 개수는 고객이 5명이고 주문이 10건이므로

(5x10) 50 이다. Customer 테이블의 1번 고객 박지성은 Orders 테이블의 열 개 행 모두에 대해

결합하고, 나머지 고객도 각각 Orders 테이블의 열 개 행 모두에 대해 결합한다.(이처럼 조건이 없는 테이블 간의 조인을 카티션 프로덕트라고 한다.) 그렇지만 결과는 논리에 맞지 않는다.

**박지성의 custid가 1이므로 Orders 테이블에서 custid가 1인 것만 선택하면 된다.**

이것은 SQL문의 WHERE 절에 두 테이블의 연결 조건을 추가함으로써 쉽게 해결할 수 있다.

1. 조인

문제1: 고객과 고객의 주문에 관한 데이터를 모두 보이시오.

**SELECT \***

**FROM Customer, Orders**

**WHERE Customer.custid = Orders.custid;**

**ORDER BY Customer.custid;** //위 결과를 고객별로 보기 위해 ORDER BY로 정렬

**여러 개의 테이블을 연결하여 하나의 테이블을 만드는 과정을 조인이라고 한다.** 특히 앞의 SQL 문처럼 동등조건에 의하여 테이블을 조인하는 것을 동등조인이라고 한다. 조인이라고 하면 대부분 동등조인을 말한다.

**조인을 사용하면 두 개 이상의 테이블에서 일부 데이터만 얻을 수 있다.** 예를 들어 고객의 이름과 고객이 주문한 도서의 가격을 알고 싶다면 다음과 같이 SQL문을 작성한다.

문제2: 고객의 이름과 고객이 주문한 도서의 판매가격을 검색하시오.

**SELECT name, saleprice**

**FROM Customer, Orders**

**WHERE Customer.custid = Orders.custid**

문제3: 고객별로 주문한 모든 도서의 총판매액을 구하고, 고객별로 정렬하시오.

**SELECT name, SUM(saleprice)**

**FROM Customer, Orders**

**WHERE Customer.custid = Orders.custid**

**GROUP BY Customer.name** //3개의 박지성이 하나로 합쳐져서 판매총액을 구함.

**ORDER BY Customer.name;**

세 개 이상의 테이블을 조인할 수도 있다. 예를 들어 각 고객의 이름과 고객이 주문한 도서의 이름을 구하고 싶다고 하자. 결과를 얻으려면 세 개의 테이블을 한꺼번에 조인해야 한다.

고객의 이름은 **Customer** 테이블에 있고, 고객의 주문 내역은 **Orders** 테이블에 있으며, **도서의 이름**은 **Book** 테이블에 있기 때문이다. Customer 테이블과 Orders 테이블은 Customer의 기본키인 custid와 Orders의 외래키인 custid로 연결한다. 또 Orders 테이블과 Book 테이블은 Orders의 외래키인 bookid인 Book의 기본키인 bookid로 연결한다.

문제4: 고객의 이름과 고객이 주문한 도서의 이름을 구하시오.

**SELECT Customer.name, book.bookname**

**FROM Customer, Orders, Book**

**WHERE Customer.custid = Orders.custid AND Orders.bookid = Book.bookid;**

문제5: 가격 20,000원인 도서를 주문한 고객의 이름과 도서의 이름을 구하시오.

**SELECT Customer.name , Book.bookname**

**FROM Customer, Orders, Book**

**WHERECustomer.custid=Orders.custidANDOrders.bookid=Book.bookidANDBook.price=20000;**

조인 연산의 특별한 경우로 외부조인outer join이 있다. 고객의 이름과 고객의 주문한 도서의 가격을 구하는 **동등조인의 예에서 도서를 주문하지 않은 고객 ‘박세리’는 결과에 포함되지 않는다.** 만약 도서의 가격을 구하려면 어떻게 해야 할까? 방법은 외부조인을 사용하는 것이다.

문제6: 도서를 구매하지 않은 고객을 포함하여 고객의 이름과 고객이 주문한 도서의 판매가격을 구하시오.

**SELECT Customer.name, saleprice**

**FROM Cutomer LEFT OUTER JOIN Orders**

**ON Customer.custid = Orders.cutid;**

2. 부속질의

SQL 문 내에 또 다른 SQL 문을 작성해 보자. “가격이 가장 비싼 도서의 이름은 무엇인가?”라는 질문에 대한 답을 구한다고 생각해 보자. 가장 비싼 도서의 가격은 다음과 같이 구할 수 있으며, 답은 35000원이다.

**SELECT MAX(price)**

**FROM Book;**

만약 가장 비싼 도서의 가격을 알고 있다면 다음과 같이 가격이 35,000원인 도서의 이름을 바로 검색하면 된다.

**SELECT bookname**

**FROM Book**

**WHERE price=35000;**

위 두 질의를 하나의 질의로 작성할 수 있을까? 가능하다. 두 번째 질의의 35000 값 위치에 첫 번째 질의를 대치하면 된다.

**SELECT bookname**

**FROM Book**

**WHERE price=(SELECT MAX(price)**

**FROM Book);**

문제1: 도서를 구매한 적이 있는 고객의 이름을 검색하시오.

**SELECT name**

**FROM Customer**

**WHERE custid IN (SELECT custid FROM Orders);**

문제2: ‘대한미디어’에서 출판한 도서를 구매한 고객의 이름을 보이시오.

SELECT name

FROM Customer

WHERE custid IN (SELECT custid

FROM Orders

WHERE bookid IN (SELECT bookid

FROM Book

WHERE publisher=’대한미디어’));

부속질의와 조인은 여러 테이블을 하나의 SQL 문에서 다루는 점은 같이잠 차이점이 있다. 부속질의는 SELECT 문에 나오는 결과 속성을 FROM 절의 테이블에서만 얻을 수 있고, 조인은 조인한 모든 테이블에서 결과 속성을 얻을 수 있다. 조인은 부속질의가 할 수 있는 모든 것을 할 수 있다. 그러나 부속질의를 조인을 이용해서 작성해 보면 부속질의만의 편리함을 알 수 있다. 한 개의 테이블에서만 결과를 얻는 여러 테이블 질의는 조인보다 부속질의로 작성하는 것이 훨씬 편하다.

집합 연산

SQL 문의 결과는 테이블로 나타난다. 테이블은 투플의 집합이므로 테이블 간의 집합 연산을 이용하여 합집합, 차집합, 교집합을 구할 수 있다. SQL 에서 집합 연산은 합집합을 UNION, 차집합을 EXCEPT, 교집합을 INTERSECT로 나타낸다. 예들 들어, Customer 테이블에는 도서를 주문한 고객과 주문하지 않은 고객이 같이 있다. 도서를 주문하지 않은 고객의 이름은 어떻게 알 수 있을까?

{도서를 주문하지 않은 고객} = {모든 고객} – {도서를 주문한 고객}

1) 모든 고객의 이름은 Customer 테이블에서 다음과 같이 구할 수 있다.

**SELECT name**

**FROM Customer;**

2) 도서를 주문한 고객은 Orders 테이블에 기록이 있는 고객번호를 이용하여 Customer 테이블에서 구할 수 있다.

**SELECT name**

**FROM Customer**

**WHERE custid IN (SELECT custid FROM Orders);**

3) 도서를 주문한 고객은 Orders 테이블에 기록이 있는 고객번호를 이용하여 Customer 테이블에서 구할 수 있다.

**SELECT name**

**FROM Customer**

**WHERE custid IN (SELECT custid FROM Orders);**

4) 도서를 주문하지 않은 고객은 위의 두 SQL문을 이용하여 구한다.

**SELECT name**

**FROM Customer**

**EXCEPT**

**SELCT name**

**FROM Customer**

**WHERE custid IN (SELECT custid FROM Orders);**

EXISTS

원래 단어에서 의미하는 것과 같이 조건에 맞는 투플이 존재하면 결과에 포함시킨다. 즉 부속질의문의 어떤 행이 조건을 만족하면 참이다.

문제1: 주문이 있는 고객의 이름과 주소를 보이시오.

**SELCET name, address**

**FROM Customer cs**

**WHERE EXISTS (SELECT \***

**FROM Orders od**

**WHERE cs.custid = od.custid);**

**SQLD 공부**

1. 모델링의 특징

1) 추상화

2) 단순화

3) 명확화

2. 모델링의 3가지 관점

1) 데이터 관점

2) 프로세스 관점

3) 상관 관점

3. 데이터 모델링의 3단계 진행

1단계: 개념적구조

2단계: 논리적구조

3단계: 물리 구조(데이터베이스)

4. 데이터 독립성의 필요성

**데이터의 독립성은 하위 단계의 데이터 구조가 변경되더라도 상위 단계에는 영향을 미치지 않는 속성**

필요한 이유:

1) 유지보수 비용 증가

2) 데이터 중복성 증가

3) 데이터 복잡도 증가

4) 요구사항 대응 저하

5. 데이터베이스 3단계 구조

외부단계 : 외부 스키마

개념적 단계 : 개념 스키마

내부적 단계 : 내부 스키마

**외부 스키마**

1) 각각의 사용자가 보는 DB 스키마이다.

2) 개인 사용자 혹은 응용 프로그램 개발자가 접근하는 DB 스키마이다.

**개념 스키마**

1) 모든 사용자의 관점을 하나로 통합한 비즈니스 전체의 DB를 기술한 스키마이다.

2) 응용 프로그램 및 사용자들이 필요한 데이터를 통합한 전체 DB를 기술한 것으로, 실제 DB에

저장되는 데이터와 응용 프로그램 및 사용자들 간의 관계를 표현하는 스키마이다.

**내부 스키마**

1) DB가 물리적으로 저장된 형식을 표현한 스키마다.

2) 물리적 하드웨어 장치에 데이터가 실제로 저장되는 방법을 표현한 스키마이다.

6. 데이터베이스 3단계 구조에서의 2가지 데이터 독립성

1) 논리적 데이터 독립성

- 개념 스키마가 변경되어도 외부 스키마에는 영향을 미치지 않도록 지원

- 논리적 구조가 변경되어도 응용 프로그램에 영향을 미치지 않는다.

2) 물리적 데이터 독립성

- 내부 스키마가 변경되어도 외부/개념 스키마는 영향을 받지 않도록 지원하는 것을 뜻한다.

- 저장 장치의 구조 변경은 응용 프로그램/개념 스키마에 영향을 미치지 않는다.

7. 좋은 데이터 모델의 요소

1) 완전성

2) 중복 배제

3) 업무 규칙

4) 데이터 재사용

5) 의사소통

6) 통합성

7. 엔터디

비즈니스 관점에서 IT 시스템을 통해 저장 및 관리해야 하는 집합적인 어떤 것이라고 한다.

엔터티: 상가

인스턴스: 스타벅스 강남점, 이디야

엔터티: 지하철역

인스턴스: 2호선 강남역, 2호선 역삼역

7-1. 엔터티의 특징

1) 비즈니스 요구 조건 만족을 위해 반드시 필요하고, 저장 및 관리하고자 하는 정보여야 한다.

2) 유일한 식별자에 의해 식별이 가능해야 한다. 즉, 집합 내에서 단 1건을 콕 짚어낼 수 있어야 한다.

3) 영속적으로 존재하는 인스턴스(2개이상)의 집합이어야 한다.

4) 엔터티는 비즈니스 프로세스에 의해 반드시 이용되어야 한다.

5) 엔터티는 반드시 속성을 가지고 있어야 한다.

6) 엔터티는 다른 엔터티와 최소 1개 이상의 관계가 있어야 한다.

7-2. 엔터티의 분류

**1. 유무형에 따른 분류**

1) 유형 : 실체가 존재하고 물리적인 형태가 있으며 안정적이고 지속적으로 활용되는 엔터티

- 사원, 물품, 강사

2) 개념 : 물리적인 형태가 존재하는 것은 아니지만 비즈니스적으로 관리해야 할 개념적 정보를

정보를 저장하는 엔터티이다.

- 조직, 보험상품

3) 사건 : 비즈니스를 수행함으로써 발생되는 엔터니이다. ; 유행/개념 엔터티에 비해 데이터 발생량이 많으며, 다양한 통계 자료에 이용될 수 있다.

- 주문, 청구, 미납

**2. 발생시점에 따른 분류**

1) 기본 :

- 비즈니스에서 스스로 태어난 존재에 대한 정보로서, 타 엔터티와 관계에 의해서 생성되는 것이 아닌 독립적으로 생성이 가능한 엔터티이다.

(사원, 부서, 고객, 상품, 자재)

2) 중심 :

- 기본 엔터티로부터 발생되며 비즈니스에 있어서 중심적인 역할을 하는 엔터티

- 데이터의 양이 많이 발생되고 타 엔터티와의 관계 속에서 많은 행위 엔터티를 도출시킨다.

(계약, 사고, 예금원장, 청구, 주문, 매출)

3) 행위 : 2개 이상의 부모 엔터티로부터 발생되는 엔터티이다. 다양하고 복잡한 비즈니스를 처리하는 과정에서 데이터양이 많아질 수 있다.

상세 설계 단계 혹은 프로세스와 상관 모델링을 진행하면서 도출된다.

(주문목록, 사원변경이력)

8. 속성

**속성은 고객 엔터티의 고객성별과 같이 비즈니스에 필요로 하며, 인스턴스에서 관리하고자 하는 의미상 더 이상 분리되지 않는 최소의 데이터 단위를 말합니다.**

속성은 엔터티에 대한 자세하고 구체적인 정보를 나타내며, 각각의 속성은 구체적인 값을 갖게됩니다. 예를 들면 지하철 엔터티의 **노선명**, **역명**과 같은 각각의 값을 대표하는 이름들을 **속성**이라고 하고 ‘경의중앙’(노선명), ‘신촌’(역명) 과 같이 각각의 이름에 대한 구체적인 값을 속성값이라고 합니다.

엔터티 – 인스턴스 – 속성 – 속성값

8-1. 속성의 표기법

# : 식별자임을 표시

\* : 필수 값임을 표시

ㅇ: 선택 값임을 표시

8-2. 속성의 특징

1) 엔터티와 마찬가지로 반드시 비즈니스에서 필요로 하고 IT 시스템에서 저장 및 관리하고자 하는 정보여야 한다.

2) 정규화 이론에 따라 속성이 속해 있는 엔터티의 주식별자에 함수적 종속성을 가져야 한다. *– 지하철역 엔터티의 식별자인 지하철역번호가 노선명과 역명을 결정한다.*

3) 하나의 속성에는 1개의 값만을 가진다. 하나의 속성에 여러 개의 값이 있는 다중 값일 경우 별도의 엔터티를 이용하여 분리한다. – *특정 지하철의 노선명과 역명은 각각 하나씩이다.*

8-3. 속성의 분류

1. **기본 속성**: 비즈니스 분석을 통해 도출된 속성을 기본 속성이라고 한다.

- 상가 엔터티의 상호명 속성은 기본 속성이라고 할 수 있다.

2. **설계 속성**: 비즈니스 분석을 통해 도출된 것은 아니지만 데이터 모델 설계를 하면서 도출하는 속성을 설계 속성이라고 한다.

- 지하철역 엔터티의 식별자인 지하철역번호가 노선명과 역명을 결정한다.

3. **파생 속성**: 다른 속성에 의해서 계산이나 변형이 되어 생성되는 속성을 파생 속성이라고 한다.

- 특정 지하철역의 노선명과 역명은 각각 하나씩이다.

8-3-1. 속성의 분류 – 엔터티 구성 방식에 따른 분류

1. PK: 엔터티에 단 하나의 인스턴스를 식별할 수 있는 속성을 PK속성이라고 한다.

2. FK: 타 엔터디와의 관계를 통해 포함된 속서을 FK속성이라고 한다.

3. 일반속성: 엔터티 내에 존재하면서 PK 혹은 FK 속성이 아닌 일반 속성이라고 한다.

|  |  |  |
| --- | --- | --- |
| 엔터티명 | 속성분류 | 속성명 |
| 지하철역 | PK 속성 | **지하철역번호** |
| 일반 속성 | 노선명  역명 |
| 지하철역승하차 | PK 속성 | **지하철역번호**  기준일자  시작시간  종료시간  승하차구분코드 |
| FK 속성 | **지하철역번호** |
| 일반 속성 | 승하차횟수 |

지하철역승하차 엔터티의 FK 속성은 지하철역번호 속성이다. **지하철역승하차 엔터티**는 지하철역 엔터티의 자식 엔터티이다. 부모 엔터티인 지하철역 엔터티의 PK 속성을 물려 받았으며 해당 속성은 FK 속성이 된다.

9. 도메인

각 속성은 값의 범위 및 유형이 있으며, 이를 속성의 도메인이라고 한다.

= **속성이 가질 수 있는 값의 범위를 도메인이라고 한다.**

10. 관계

관계는 엔터티끼리 상호 연관성이 있는 상태를 의미한다.

관계는 데이터 모델 내에 존재하는 엔터디 간 논리적인 연관성을 의히한다.

관계느 부서 엔터티와 사원 엔터티의 관계와 같이 존재의 의한 관계가 있고, 고객 엔터티와 주문 엔터티의 관계와 같이 행위에 의한 관계가 있다.

10-1. 관계의 페어링

관계는 엔터티 안에 인스턴스가 개별적으로 관계를 가지는 것(페어링)이고 이것이 집합을 관계로 표현한다는 것이다. 개별 인스턴스가 각각 다른 종류의 관계를 가지고 있다면 두 엔터티 사이에 2개 이상의 관계가 형성될 수 있다.

예) [강사] ------강의 한다-----🡪 [수강생] : ‘강의한다’라는 관계

10-2. 관계의 표기법

관계 표기 시에는 관계 차수 및 관계 선택사양을 명확하게 해야 한다.

1) **관계 차수**: 2개의 엔터티 간 관계에서 참여자의 수를 표현하는 것

**1:M , 1:1, M:M**

예) 주문은 여러 제품을 포함할 수 있고 제품도 여러 주문에 의해 주문될 수 있다.

2) 관계 선택사양(Optionality)

**필수 참여 관계** : 열차문이 완전히 닫혀야만 열차는 출발한다. 열차의 출발과 열차문의 완전한 닫힘은 필수적인 연관관계이다. 이러한 관계를 필수참여관계라고 한다.

**선택 참여 관계** :열차의 출발을 알리는 출발 안내 방송은 열차의 출발과는 상관없이 언제든지 방송할 수 있다. 출발 안내가 제대로 되지 않아도 출발에는 영향을 주지 않는다. 이렇게 열차의 출과과 출발 안내 방송은 정보 안내로서의 관계는 있지만 서로가 필수적인 상황은 아니다. 이러한 관계를 **선택참여관계**라고 한다.

예) 지하철역승하차 입장에서는 자신이 갖고 있는 지하철역의 승하차 정보가 어떤 지하철역인지 반드시 알아야 한다. 즉 지하철역번호가 반드시 존재해야 한다. -> **실선**

지하철역 입장에서는 여러 개 (까치발)의 지하철역승하차 정보를 가질 수 있고 단 1개의 지하철역승하차 정도도 갖지 않을 수 있다. -> **점선**

10-3. 관계읽기

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **각각의 혹은 하나의** | **기준 엔터티** | **관계 차수** | **대상 엔터티** | **필수/선택** | **관계명** |
| 각각의 | 고객은 | 여러 개의 | 주문을 | 때때로 | 주문한다 |
| 하나의 | 주문은 | 한 명의 | 고객을 | 항상 | 가진다 |

11. 식별자

엔터티는 인스턴스들의 집합이다. 이러한 집합에서 단 하나의 인스턴스를 구별해 낼 수 있는 논리적인 이름이 필요하며 이러한 구분자를 식별자라고 한다.

엔터티의 각 인스턴스를 개별적으로 식별하기 위해 사용되는 하나의 속성 혹은 속성들의 조합을 식별자라고 한다. **엔터티 내에서 하나의 행을 꼭 집어낼 때 사용하는 것**

식별자는 주식별자에 의해 엔터티 내의 모든 인스턴스들이 유일하게 구분되어야 하고, 주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 한다.

1) 유일성

- 사원 엔터티의 사원번호 속성은 주식별자이다. 사원번호는 모든 직원에게 대해 개인별로 고유하게 부여된다.

2) 최소성

- 사원번호만으로도 고유한 구조인데, 사원분류코드+사원번호 조합으로 식별자가 구성될 경우 부적절한 주식별자 구조이다.

3) 불변성

- 한번 정해진 사원번호의 값은 다른 값으로 변경되지 않아야 한다.

4) 존재성

- 사원번호가 없는 회사직원은 있을 수 없다.

11-1. 식별자 분류

1) 대표성 여부

2) 스스로 생성 여부

3) 속성의 수

4) 대체 여부

11-2. 식별자 도출 기준

1) 비즈니스에서 자주 이용되는 속성을 주식별자로 지정한다.

2) 명칭, 장소와 같이 이름으로 기술되는 속성은 가능하면 주식별자로 하지 않는다.

3) 주식별자를 복합식별자로 할 경우 지나치게 많은 속성이 포함되지 않도록 한다.

11-3. 식별자 관계와 비식별자 관계의 결정

**외부식별자**는 자기 자신의 엔터티에서 필요한 속성이 아니라 다른 엔터티와의 관계를 통해 자식쪽 엔터티에 생성되는 속성을 말하며, 데이터베이스 생성 시에 **외래키 역할**을 한다.

자식 엔터티에서 부모 엔터티로부터 받은 외부식별자를 (1)자신의 주식별자(식별자관계)로 이용할 것인지 또는 부모와 (2)연결이 되는 속성(비식별자관계)으로서만 이용할 것인지를 결정해야 한다. 식별자 관계를 사용할지 비식별자 관계를 사용할지는 업무의 특징, 자식 엔터티의 주식별자 구성, SQL작성 전략에 의해 결정된다.

자식 입장에서 외부식별자(FK)를 자신의 식별자로 사용하고 있는 것을 식별자 관계라고 한다. 다시 말해, 자식 엔터티의 주식별자로 부모 엔터티의 주식별자가 상속이 되는 경우를 식별자 관계라고 칭한다.

부모로부터 받은 식별자를 자식 엔터티의 주식별자로 이용하는 경우는 NULL이 허용되지 않으므로, 반드시 부모 엔터티가 생성되어야 자식 엔터티가 생성되는 경우입니다.

최종적으로 식별자 관계를 정리하자면,