Ch01 데이터 모델링의 이해

1-1 데이터 모델의 이해

> 모델링의 정의

- 복잡한 "현실세계"를 단순화시켜 표현하는 것
- 사물 또는 사건에 관한 양상(Aspect)이나 관점(Perspective)을 연관된 사람이나 그룹을 위하여 명확하게 하는 것
- 현실 세계의 추상화된 반영

> 모델링의 특징

- 추상화(모형화, 가설적)는 현실세계를 일정한 형식에 맞추어 표현을 한다는 의미
- 단순화는 복잡한 현실세계를 약속된 규약에 의해 제한된 표기법이나 언어로 표현
- 명확화는 누구나 이해하기 쉽게 하기 위해 대상에 대한 애매모호함을 제거하고 정확(正確)하게 현상을 기술

> 모델링의 세 가지 관점

1) 데이터 관점 :업무가 어떤 데이터와 관련이 있는지 또는 데이터간의 관계는 무엇인지 모델링하는 방법

2) 프로세스 관점 : 업무가 실제하고 있는 일은 무엇인지 또는 무엇을 해야하는지 모델링하는 방법

3) 상관 관점 : 업무가 처리하는 일의 방법에 따라 데이터는 어떻게 영향을 받고 있는지 모델링하는 방법

> 데이터 모델링의 정의

- 정보 시스템을 구축하기 위한 **데이터 관점**의 업무 분석 기법
- 현실세계의 데이터에 대해 **약속된 표기법에 의해 표현**되는 과정
- 데이터베이스를 구축하기 위한 **분석/설계**의 과정

> 데이터 모델이 제공하는 기능

- 시스템을 현재 또는 원하는 모습으로 **가시화**
- 시스템의 구조와 행동을 **명세화**
- 시스템을 구축하는 **구조화된 틀**을 제공
- 시스템을 구축하는 과정에서 결정한 것을 **문서화**
- 다양한 영역에 집중하기 위해 다른 영역의 세부 사항은 숨기는 **다양한 관점**을 제공
- 특정 목표에 따라 **구체화**된 상세 수준의 표현방법을 제공

> 데이터 모델링의 중요성 및 유의점

중요성	설명
파급효과	시스템 구축 작업에서 다른 어떤 설계 과정보다 설계가 중요함
간결한표현	데이터 모델은 구축할 시스템의 정보 요구사항과 한계를 가장 명확하고 간결하게 표현할 수 있는 도구
데이터 품질	데이터의 중복, 비유연성, 비일관성이 발생할 수 있음

> 데이터 모델링의 3 단계 진행

1) 개념적 데이터 모델링
추상화 수준이 높고 업무 중심적이고 **포괄적**인 수준의 모델링 진행. 전사적 데이터 모델링, EA 수립 시 많이 사용
2) 논리적 데이터 모델링
시스템으로 구축하고자 하는 업무에 대해 **Key, 속성, 관계** 등을 정확하게 표현, 재 사용성이 높음
3) 물리적 데이터 모델링
실제로 데이터베이스에 이식할 수 있도록 성능, 저장 등 **물리적**인 성격을 고려하여 설계

> 프로젝트 생명주기(Life Cycle)에서 데이터 모델링

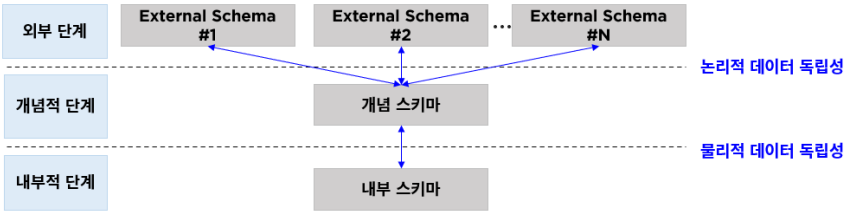
- 프로젝트 생명 주기는
  정보전략계획 -> 분석 -> 설계 -> 개발 -> 테스트 -> 전환/이행 단계가 있음
- 정보전략계획/분석 단계 : 개념적 데이터 모델링
- 분석 단계 : 논리적 데이터 모델링
- 설계 단계 : 물리적 데이터 모델링

> 데이터 독립성의 필요성

- 지속적으로 증가하는 유지보수 비용을 절감하고 데이터 복잡도를 낮추며 중복된 데이터를 줄이기 위한 목적이 있음
- 끊임없이 요구되는 사용자 요구사항에 대해 화면과 데이터베이스 간에 서로 독립성을 유지하기 위한 목적으로 데이터 독립성 개념이 출현
- 유지보수 비용 증가, 데이터 중복성 증가, 데이터 복잡도 증가, 요구사항 대응저하로 인해 데이터 독립성 필요

  * 데이터 독립성 확보
  - 각 View 의 독립성을 유지하고 계층별 View 에 영향을 주지 않고 변경이 가능
  - 단계별 Schema 에 따라 데이터 정의어(DDL)와 데이터 조작어(DML)가 다름을 제공

> 데이터베이스 3 단계 구조



1) 외부스키마 (사용자 관점)

- View 단계 여러 개의 사용자 관점으로 구성, 즉 개개 사용자 단계로서 개개 사용자가 보는 개인적 DB 스키마
- DB 의 개개 사용자나 응용프로그래머가 접근하는 DB 정의

2) 개념스키마 (통합 관점)

- 개념 단계 하나의 개념적 스키마로 구성 모든 사용자 관점을 통합한 조직 전체의 DB 를 기술하는 것
- 모든 응용시스템들이나 사용자들이 필요로 하는 데이터를 통합한 조직 전체의 DB 를 기술한 것으로 DB 에 저장되는 데이터와 그들간의 관계를 표현하는 스키마

3) 내부스키마 (물리적 관점)

- 내부 단계, 내부 스키마로 구성, DB 가 물리적으로 저장된 형식
- 물리적 장치에서 데이터가 실제적으로 저장되는 방법을 표현하는 스키마

> 데이터베이스 3 단계 구조에서의 데이터 독립성 2 가지

1) 논리적 독립성

- 개념 스키마가 변경되어도 외부 스키마에는 영향을 미치지 않도록 지원하는 것
- 논리적 구조가 변경되어도 응용 프로그램에 영향 없음

2) 물리적 독립성

- 내부 스키마가 변경되어도 외부/개념 스키마는 영향을 받지 않도록 지원하는 것
- 저장 장치의 구조 변경은 응용프로그램과 개념 스키마에 영향 없음

> 사상 (매핑) 2 가지

1) 외부적/개념적 사상 (논리적 사상)

- 외부적 뷰와 개념적 뷰의 상호 호환성을 정의함
- 외부화면이나 사용자에게 인터페이스하기 위한 스키마 구조는 전체가 통합된 개념적 스키마와 연결된다는 것

2) 개념적/내부적 사상 (물리적 사상)

- 개념적 뷰와 저장된 데이터베이스의 상호관련성 정의

- 통합된 개념적 스키마 구조와 물리적으로 저장된 구조의 물리적인 테이블 스페이오와 연결되는 구조

> 데이터 모델링의 세 가지 요소

- 업무가 관여하는 어떤 것(Things)
- 어떤 것이 가지는 성격(Attributes)
- 업무가 관여하는 어떤 것 간의 관계(Relationships)

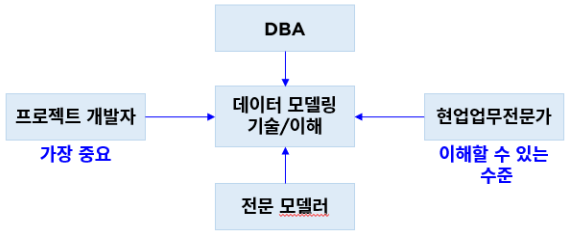
> 데이터 모델링 용어

개념	복수/집합 개념 타입/클래스	개별/단수 개념 어커런스/인스턴스
어떤 것 (Thing)	엔터티 타입(Entity Type)	엔터티(Entity)
	엔터티(Entity)	인스턴스(Instance) 어커런스(Occurrence)
어떤 것 간의 연관 (Association between Things)	관계(Relationship)	패어링(Pairing)
어떤 것의 성격 (Characteristic of a Thing)	속성(Attribute)	속성값(Attribute Value)

> 데이터 모델링 작업순서

1. 엔터티를 그린다.
2. 엔터티를 적절하게 배치한다.
3. 엔터티간 관계를 설정한다.
4. 관계명을 기술한다.
5. 관계의 참여도를 기술한다.
6. 관계의 필수 여부를 기술한다

> 데이터 모델링의 이해관계자



> 좋은 데이터 모델의 요소

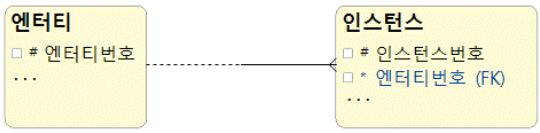
요소	설명
완전성	업무에 필요한 데이터가 모두 정의 되어야함
중복 배제	동일한 사실은 한번만 저장 해야함
업무규칙	데이터 모델 분석만으로도 비즈니스 로직이 이해되어야 함
데이터 재사용	데이터 통합성과 독립성 고려해야함
의사소통	데이터 모델을 보고 이해 당사자들끼리 의사소통이 이루어져야 함
통합성	동일한 데이터는 유일하게 정의해서 다른 영역에서 참조해야 함

1-2 엔터티

> 엔터티의 개념

- 엔터티는 사람, 장소, 물건, 사건, 개념 등의 명사에 해당한다.
- 엔터티는 업무상 관리가 필요한 관심사에 해당한다.
- 엔터티는 저장이 되기 위한 어떤 것(Thing)이다.

> 엔터티와 인스턴스



엔터티는 인스턴스의 집합

❖엔터티는 인스턴스의 집합이라고 할 수 있다.

> 엔터티 특징

- 반드시 해당 업무에서 필요하고 관리하고자 하는 정보이어야 한다.(예. 환자, 토익의 응시횟수, ...)
- 유일한 식별자에 의해 식별이 가능해야 한다.
- 영속적으로 존재하는 인스턴스의 집합 이어야 한다.('한개'가 아니라 '두 개 이상')
- 엔터티는 업무 프로세스에 의해 이용되어야 한다.
- 엔터티는 반드시 속성이 있어야 한다.
- 엔터티는 다른 엔터티와 최소 한 개 이상의 관계가 있어야 한다.

> 엔터티의 분류

1) 유무형에 따른 분류

유형	사원, 물품, 강사	물리적인 형태가 있고 안정적이며 지속적으로 활용되는 엔터티로 업무로부터 엔터티를 구분하기가 가장 용이하다
개념	조직, 보험상품	물리적인 형태는 존재하지 않고 관리해야 할 개념적 정보로 구분이 되는 엔터티
사건	주문, 청구, 미납	업무를 수행함에 따라 발생되는 엔터티로서 비교적 발생량이 많으며 각종 통계자료에 이용될 수 있다

2) 발생시점에 따른 분류

기본	사원,부서,고객,상품,자재	업무에 원래 존재하는 정보로서 다른 엔터티와 관계에 의해 생성되지 않고 독립적으로 생성이 가능하고 자신은 타 엔터티의 부모의 역할을 하게 된다

중심	계약,사고, 예금원장, 청구, 주문,매출	본엔터티로부터 발생되고 그 업무에 있어서 중심적인 역할을 한다. 데이터의 양이 많이 발생되고 다른 엔터티와의 관계를 통해 많은 행위엔터티를 생성
행위	주문목록, 사원변경이력	두 개 이상의 부모엔터티로부터 발생되고 자주 내용이 바뀌거나 데이터량이 증가 상세 설계단계나 프로세스와 상관모델링을 진행하면서 도출

> 엔터티의 명명

- 가능하면 현업업무에서 사용하는 용어를 사용한다.
- 가능하면 약어를 사용하지 않는다.
- 단수 명사를 사용한다.
- 모든 엔터티에서 유일하게 이름이 부여되어야 한다.
- 엔터티 생성 의미대로 이름을 부여한다.

1-3 속성

> 속성의 개념

- **업무에서 필요로** 한다.
- 의미상 **더 이상 분리되지 않는다.**
- 엔터티를 설명하고 인스턴스의 구성요소가 된다

> 엔터티, 인스턴스, 속성, 속성값의 관계

- 한 개의 엔터티는 두 개 이상의 인스턴스의 집합 이어야 한다.
- 한 개의 엔터티는 두 개 이상의 속성을 갖는다.
- 한 개의 속성은 한 개의 속성값을 갖는다.

> 속성의 분류

1) 특성에 따른 분류

- 속성은 업무분석을 통해 바로 정의한 속성을 **기본속성(Basic Attribute)**
- 원래 업무상 존재하지는 않지만 설계를 하면서 도출해내는 속성을 **설계속성(Designed Attribute)**
- 다른 속성으로부터 계산이나 변형이 되어 생성되는 속성을 **파생속성(Derived Attribute)**이라고 한다.

2) 엔터티 구성방식에 따른 분류

- 엔터티를 식별할 수 있는 속성을 PK(Primary Key)속성, 다른 엔터티와의 관계에서 포함된 속성을 FK(Foreign Key)속성, 엔터티에 포함되어 있고 PK, FK 에 포함되지 않은 속성을 일반속성이라 한다.

> 도메인

- 각 속성은 가질 수 있는 값의 범위가 있는데 이를 그 속성의 도메인(Domain)이라 한다
- 학생이라는 엔터티가 있을 때 학점이라는 속성의 도메인은 0.0 에서 4.0 사이의 실수 값이며 주소라는 속성은 길이가 20 자리 이내인 문자열로 정의
- 각 속성은 도메인 이외의 값을 갖지 못한다

> 속성의 명명

- 해당업무에서 사용하는 이름을 부여 한다.
- 서술 식 속성 명은 사용하지 않는다.
- 약어 사용은 가급적 제한한다.
- 전체 데이터모델에서 유일성 확보하는 것이 좋다.

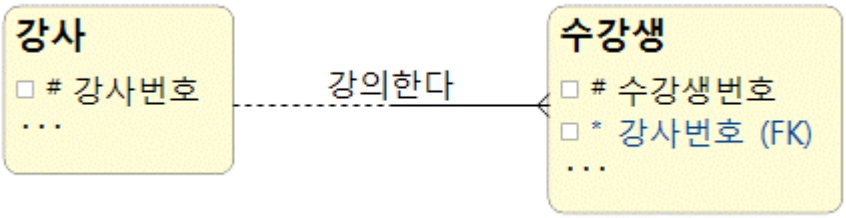
1-4 관계

> 관계의 정의

- 사전적으로 정의하면 상호 연관성이 있는 상태이다
- "엔터티의 인스턴스 사이의 논리적인 연관성으로서 존재의 형태 로서나 행위로서 서로에게 연관성이 부여된 상태" 라고 할 수 있다.

> 관계의 패어링

- 관계는 엔터티 안에 인스턴스가 개별적으로 관계를 가지는 것(패어링)이고 이것의 집합을 관계로 표현한다는 것이다.
- 개별 인스턴스가 각각 다른 종류의 관계를 가지고 있다면 두 엔터티 사이에 두 개 이상의 관계가 형성될 수 있다.
- 각각의 엔터티의 인스턴스들은 자신이 관련된 인스턴스들과 관계의 어커런스로 참여하는 형태를 관계 패어링(Relationship Paring)이라 한다



* 인스턴스 각각은 자신의 연관성을 가지고 있을 수 있음
* 이것을 집합하여 "강의한다"라는 관계를 도출

> 관계의 분류

1) 존재의 의한 관계

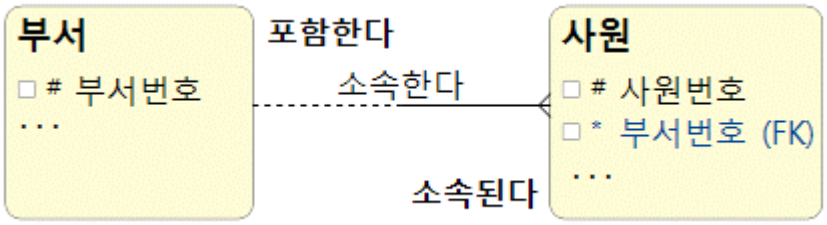
- "소속된다'라는 의미는 행위에 따른 이벤트에 의해 발생되는 의미가 아니고 그냥 사원이 부서에 소속되어 있기 때문에 나타나는 즉 존재의 형태에 의해 관계가 형성되어 있는 것이다.

2) 행위에 의한 관계

- 주문 엔터티의 주문번호는 고객이 '주문한다'라는 행위에 의해 발생되었기 때문에 두 엔터티 사이의 관계는 행위에 의한 관계가 되는 것이다

> 관계의 표기법 (관계명)

1. 엔터티가 관계에 참여하는 형태를 지칭한다.
2. 각각의 관계는 두 개의 관계명을 가지고 있다.
3. 또한 각각의 관계명에 의해 두 가지의 관점으로 표현될 수 있다



- 엔터티에서 관계가 시작되는 편을 관계시작점(The Beginning)이라고 부르고 받는 편을 관계끝점(The End)이라고 부른다.
- 관계 시작점과 끝점 모두 관계이름을 가져야 하며 참여자의 관점에 따라 관계이름이 능동적(Active)이거나 수동적(Passive)으로 명명된다.

> 관계의 명명 규칙

- **애매한 동사를 피한다.** 예를 들면 '관계된다', '관련이 있다', '이다', '한다' 등은 구체적이지 않아 어떤 행위가 있는지 또는 두 참여자간 어떤 상태가 존재하는지 파악할 수 없다.
- **현재형으로 표현한다.** 예를 들면 '수강을 신청했다', '강의를 할 것이다'라는 식으로 표현해서는 안된다. '수강 신청한다', '강의를 한다'로 표현해야 한다.

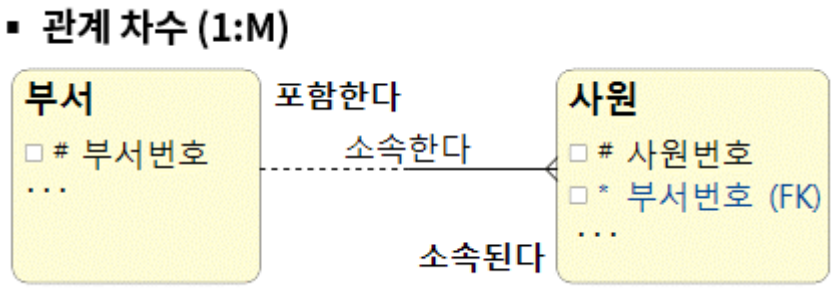
> 관계의 표기법 (관계 차수)

1) 1 : 1 관계 차수

- 관계에 참여하는 각각의 엔터티는 관계를 맺는 다른 엔터티에 대해 단지 하나의 관계만을 가지고 있다.

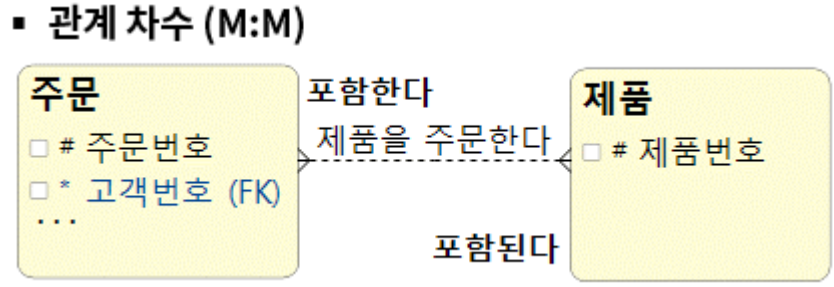
2) 1:M 관계 차수

- 한 명의 사원은 한 부서에 소속되고 한 부서에는 여러 사원을 포함한다.



3) M : M 관계 차수

- 관계에 참여하는 각각의 엔터티는 관계를 맺는 다른 엔터티에 대해 하나나 그 이상의 수와 관계를 가지고 있다



> 관계의 표기법 – 관계선택사양 (Optionality)

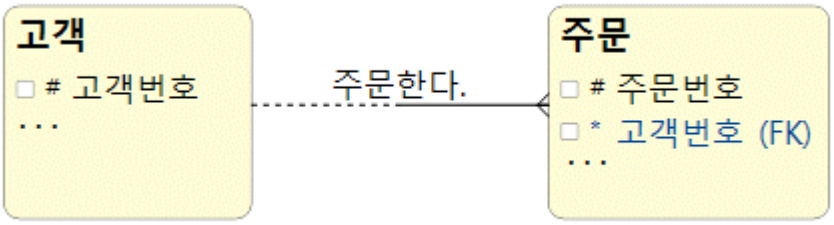
- "반드시 지하철의 문이 닫혀야만 지하철은 출발한다."
- 지하철출발과 지하철문닫힘은 **필수(Mandatory)적으로** 연결 관계가 있는 것이다. 이와 같은 것이 데이터 모델의 관계에서는 **필수참여관계(Mandatory)**가 된다.
- 지하철의 출발을 알리는 안내방송은 지하철의 출발과 상관없이 방송해도 아무런 문제가 발생하지 않는다. 즉 안내방송시스템이 고장이 나도 지하철운행에는 별로 영향을 주지 않는다
- 지하철의 출발과 지하철방송과는 정보로서 관련은 있지만 서로가 필수적인(Mandatory) 관계는 아닌 **선택적인 관계(Optional)**가 되는 것이다. 이와 같은 것이 데이터 모델 관계에서는 **선택참여관계(Optional)**가 된다

> 관계 정의 시 체크 사항

- 두 개의 엔터티 사이에 관심있는 연관 규칙이 존재하는가?
- 두 개의 엔터티 사이에 정보의 조합이 발생되는가?
- 업무기술서, 장표에 관계연결에 대한 규칙이 서술되어 있는가?
- 업무기술서, 장표에 관계연결을 가능하게 하는 동사(Verb)가 있는가?

> 관계 읽기

- 기준(Source) 엔터티를 한 개(One) 또는 각(Each)으로 읽는다.
- 대상(Target) 엔터티의 관계 참여도 즉 개수(하나, 하나 이상)를 읽는다.
- 관계선택사양과 관계 명을 읽는다.



각각의/하나의	기준 엔터티	관계 차수	대상엔터티	필수/선택	관계명
각각의	고객은	여러 개의	주문을	때때로	주문한다
각각의	주문은	하나의	고객을	반드시	가진다.

1-5 식별자

> 식별자의 개념

- 엔터티는 인스턴스들의 집합이라고 하였다. 여러 개의 집합체를 담고 있는 하나의 통에서 각각을 구분할 수 있는 논리적인 이름이 있어야 한다. 이 구분자를 식별자(Identifier)라고 한다.
- Entity 의 각 Instance 를 개별적으로 식별하기 위해 사용되는 Relationship 또는 Attribute 들의 조합

> 식별자의 특징

- 주식별자에 의해 엔터티내에 모든 인스턴스들이 유일하게 구분되어야 한다.
- 주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 한다.
- 지정된 주식별자의 값은 자주 변하지 않는 것이어야 한다.
- 주식별자가 지정이 되면 반드시 값이 들어와야 한다.

특징	내용
유일성	주식별자에 의해 엔터티내에 모든 인스턴스들을 유일하게 구분함
최소성	주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 함
불변성	주식별자가 한 번 특정 엔터티에 지정되면 그 식별자의 값은 변하지 않아야 함
존재성	주식별자가 지정되면 반드시 데이터 값이 존재 (Null 안됨)

> 식별자 분류

분류	식별자	설명
대표성여부	**주식별자**	엔터티 내에서 각 행을 구분할 수 있는 구분자이며, 타 엔터티와 참조관계를 연결할 수 있는 식별자 (ex. 사원번호, 고객번호)
	보조식별자	엔터티 내에서 각 행을 구분할 수 있는 구분자이나 대표성을 가지지 못해 참조관계 연결을 못함(ex. 주민등록번호)
스스로생성여부	**내부식별자**	엔터티 내부에서 스스로 만들어지는 식별자(ex. 고객번호)
	외부식별자	타 엔터티와의 관계를 통해 타 엔터티로부터 받아오는 식별자(ex. 주문엔터티의 고객번호)
속성의 수	**단일식별자**	하나의 속성으로 구성된 식별자(ex. 고객엔터티의 고객번호)
	복합식별자	둘 이상의 속성으로 구성된 식별자(ex. 주문상세엔터티의 주문번호+상세순번)
대체여부	**본질식별자**	업무에 의해 만들어지는 식별자(ex. 고객번호)

	인조식별자	업무적으로 만들어지지는 않지만 원조식별자가 복잡한 구성을 가지고 있기 때문에 인위적으로 만든 식별자(ex. 주문엔터티의 주문번호(고객번호+주문번호+순번))

> 식별자 도출 기준

- 해당 업무에서 자주 이용되는 속성을 주식별자로 지정한다.
- 명칭, 내역 등과 같이 이름으로 기술되는 것들은 가능하면 주식별자로 지정하지 않는다.
- 복합으로 주식별자로 구성할 경우 너무 많은 속성이 포함되지 않도록 한다**.**

> 식별자 관계와 비식별자 관계의 결정

- 외부식별자(Foreign Identifier)는 자기 자신의 엔터티에서 필요한 속성이 아니라 다른 엔터티와의 관계를 통해 자식 쪽에 엔터티에 생성되는 속성을 **외부식별자**라 하며 데이터베이스 생성 시에 **Foreign Key** 역할을 한다.
- 자식엔터티에서 부모엔터티로부터 받은 외부식별자를 자신의 주식별자로 이용할 것인지(식별자 관계) 또는 **부모와 연결이 되는 속성으로서만 이용할 것인지를 결정(비식별자 관계)**해야 한다.

> 식별자 관계

- 자식엔터티의 주식별자로 부모의 주식별자가 상속이 되는 경우를 식별자 관계(Identifying Relationship)라고 지칭한다.
- 부모로부터 받은 식별자를 자식엔터티의 주식별자로 이용하는 경우는 Null 값이 오면 안되므로 반드시 부모엔터티가 생성되어야 자기 자신의 엔터티가 생성되는 경우이다

> 비식별자 관계

- 부모엔터티로부터 속성을 받았지만 자식엔터티의 주식별자로 사용하지 않고 일반적인 속성으로만 사용하는 경우가 있다. 이와 같은 경우를 비식별자 관계(Non-Identifying Relationship)라고 하며 다음의 네 가지 경우에 비식별자 관계에 의한 외부속성을 생성한다.
- 자식엔터티에서 받은 속성이 반드시 필수가 아니어도 무방하기 때문에 부모 없는 자식이 생성될 수 있는 경우이다.
- 자식엔터티에 주식별자로 사용하여도 되지만 자식엔터티에서 별도의 주식별자를 생성하는 것이 더 유리하다고 판단될 때 비식별자 관계에 의한 외부식별자로 표현한다.

> 식별자 관계로만 설정할 경우의 문제점

- 지속적으로 식별자 관계를 연결한 데이터 모델의 PK 속성의 수는 데이터 모델의 흐름이 길어질수록 증가할 수 밖에 없는 구조를 가지게 된다.
- 개발자가 개발할 때 당연히 데이터 모델을 참조하면서 엔터티와 관계를 이용하여 개발해야 하는데 생성된 엔터티 스키마 정보만을 보고 개발하는 경우가 많다
- 조인에 참여하는 주식별자속성의 수가 많을 경우 정확하게 조인관계를 설정하지 않고 즉, 누락하여 개발하는 경우가 간혹 발견되기도 한다
- 식별자 관계만으로 연결된 데이터 모델의 특징은 주식별자 속성이 지속적으로 증가할 수 밖에 없는 구조로서 개발의 복잡성과 오류가능성을 유발시킬 수 있는 요인이 될 수 있다는 사실을 기억해야 한다.

Ch02 데이터 모델과 성능

2-1 성능 데이터 모델링의 개요

> 성능 데이터 모델링의 정의

- 성능 데이터 모델링이란 데이터베이스 성능향상을 목적으로 설계단계의 데이터 모델링 때부터 정규화, 반정규화, 테이블통합, 테이블분할, 조인구조, PK, FK 등 여러 가지 성능과 관련된 사항이 데이터 모델링에 반영될 수 있도록 하는 것으로 정의할 수 있다.

>성능 데이터 모델링 수행시점

- 성능 향상을 위한 비용은 프로젝트 수행 중에 있어서 사전에 할수록 비용이 들지 않는다.
- 분석/설계 단계에서 데이터 모델에 성능을 고려한 데이터 모델링을 수행할 경우 성능 저하에 따른 재 업무(Rework) 비용을 최소화 할 수 있는 기회를 가지게 된다.
- 분석/설계 단계에서 데이터베이스 처리 성능을 향상시킬 수 있는 방법을 주도면밀하게 고려해야 한다.

> 성능 데이터 모델링 고려사항

- 데이터 모델링을 할 때 **정규화**를 정확하게 수행한다.
- 데이터베이스 **용량 산정을** 수행한다.
- 데이터베이스에 발생되는 **트랜잭션의 유형을 파악한다**.
- 용량과 트랜잭션의 유형에 따라 **반 정규화를 수행**한다.
- **이력 모델의 조정, PK/FK 조정, 슈퍼 타입/서브타입 조정** 등을 수행한다.
- 성능관점에서 **데이터 모델을 검증**한다.

2-2 정규화와 성능

- 정규화를 수행한다는 것은 데이터를 결정하는 결정자에 의해 함수적 종속을 가지고 있는 일반 속성을 의존자로 하여 입력/수정/삭제 이상현상을 제거하는 것이다.
- 데이터의 중복 속성을 제거하고 결정자에 의해 동일한 의미의 일반 속성이 하나의 테이블로 집약되므로 한 테이블의 데이터 용량이 최소화되는 효과가 있다.
- 정규화된 테이블은 데이터를 처리할 때 속도가 빨라질 수도 있고 느려질 수도 있는 특성이 있다.

정규화된 데이터 모델은 **조회 시에는 처리 조건에 따라 성능이 향상 혹은 저하** 된다. 정규화된 데이터 모델은 **입력/수정/삭제 시 무조건 성능이 향상된다**.

> 정규화 용어

용어	설명
정규화	함수적 종속성(FD: Functional Dependency) 등과 같은 이론에 근거하여 관계형 데이터베이스 테이블의 삽입·삭제·갱신 이상(Anomaly) 현상 발생을 최소화하기 위해 좀 더 작은 단위의 테이블로 설계하는 과정. 즉, 데이터 모델을 정규형에 맞도록 고치는 과정
정규형	정규화 규정. 정규화 결과에 의해 도출된 데이터 모델이 갖춰야 할 특성
함수적 종속성	테이블의 특정 컬럼 A 의 값을 알면 다른 컬럼 B 값을 알 수 있을 때, 컬럼 B 는 컬럼 A 에 함수적 종속성이 있다고 함(ex. 고객명은 고객주민등록번호에 함수적 종속성이 있음)
결정자	함수적 종속성 설명에서, 컬럼 A 를 결정자라고 함
다치종속	결정자 컬럼 A 에 의해 컬럼 B 의 값을 다수 개 알 수 있을 때, 컬럼 B 는 컬럼 A 에 다치종속 되었다고 함 (ex. 학번을 알면 해당 학생의 다수 개 수강과목을 알 수 있을 때, 수강과목은 학번에 다치종속관계임)

> 정규화 효과 및 장점

- 상호 종속성이 강한 데이터 요소들을 분리, 독립된 개념(엔티티, 테이블)으로 정의하게 됨에 따라 High Cohesion & Loose Coupling 원칙에 충실해지며 이로 인해 **유연성**이 극대화 됨
- 개념이 좀 더 세분화됨에 따라 해당 개념에 대한 **재활용** 가능성이 높아짐(일반적으로 각종 참조 모델은 정규형을 만족하고 있음)
- Non-key 데이터 요소가 한번 만 표현됨에 따라 **중복이 최소화** 됨(데이터 품질확보, 저장공간 절약, DML 성능)

> 정규화 이론

정규형	특징
제 1 정규형	● 모든 속성은 원자 값을 가져야 함 ● 다중 값을 가질 수 있는 속성은 분리되어야 함
제 2 정규형	● 제 1 정규형을 만족하고 모든 Non-key 컬럼은 기본 키 전체에 종속되어야 함 ● 기본 키에 종속적이지 않거나 기본 키 일부 컬럼(들)에만 종속적인 컬럼은 분리되어야 함

제 3 정규형	● 제 2 정규형을 만족하고 일반속성들간에도 종속관계가 존재하지 않아야 함 ● 일반속성들 간 종속관계가 존재하는 것들은 분리되어야 함

> 정규화와 성능

- 정규화를 수행해서 조인이 발생하게 되더라도 효율적인 **인덱스 사용을 통해 조인 연산을 수행**하면 성능 상 단점은 거의 없다.
- 정규화를 수행하여 소량의 테이블이 생성된다면 **소량의 테이블을 먼저 읽어 조인 연산을 수행하면 되므로 성능 상 유리**할 수 있다.
- 정규화가 제대로 되지 않으면 동일한 종류의 속성을 여러 개 가지고 있어서 과다한 인덱스가 만들어 질 수 있는데 **정규화를 한다면 하나의 인덱스**만 만들어도 된다.

> 함수적 종속성에 근거한 정규화 수행 필요

- 함수의 종속성(Functional Dependency)은 데이터들이 어떤 기준값에 의해 종속되는 현상을 지칭하는 것이다.
- 이 때 기준값을 결정자(Determinant)라 하고 종속되는 값을 종속자(Dependent)라고 한다.

> 반정규화의 정의

- 정규화된 엔터티, 속성, 관계에 대해 시스템의 성능향상과 개발(Development)과 운영(Maintenance)의 단순화를 위해 **중복, 통합, 분리** 등을 수행하는 데이터 모델링의 기법을 의미
- 협의의 반정규화는 데이터를 중복하여 성능을 향상시키기 위한 기법이라고 정의할 수 있고 좀 더 넓은 의미의 반정규화는 성능을 향상시키기 위해 정규화된 데이터 모델에서 중복, 통합, 분리 등을 수행하는 모든 과정을 의미
- 데이터 무결성이 깨질 수 있는 위험을 무릅쓰고 데이터를 중복하여 반정규화를 적용하는 이유는 데이터를 조회할 때 디스크 I/O 량이 많아서 성능이 저하되거나 경로가 너무 멀어 조인으로 인한 성능저하가 예상되거나 칼럼을 계산하여 읽을 때 성능이 저하될 것이 예상되는 경우 반정규화를 수행하게 된다.

> 반정규화 절차

- 반정규화도 하나의 난이도 높은 데이터 모델링의 실무기술이다.
- 반정규화에 대한 필요성이 결정이 되면 칼럼의 반정규화 뿐만 아니라 테이블의 반정규화와 관계의 반정규화를 종합적으로 고려하여 적용

- 반정규화를 막연하게 중복을 유도하는 것만을 수행하기 보다는 성능을 향상시킬 수 있는 다른 방법들을 고려하고 그 이후에 반정규화를 적용하도록 해야 한다

> 반정규화의 기법 – 테이블 정규화

기법 분류	기법	내용
테이블 병합	**1:1 관계 테이블 병합**	1:1 관계를 통합하여 성능향상
	1:M 관계 테이블 병합	1:M 관계를 통합하여 성능향상
	슈퍼/서브타입 테이블 병합	슈퍼/서브 관계를 통합하여 성능 향상
테이블 분할	**수직분할**	컬럼 단위의 테이블을 디스크 I/O 분산처리를 하기 위해 테이블을 1:1 로 분리하여 성능향상
	수평분할	로우 단위로 집중 발생되는 트랜잭션을 분석하여 디스크 I/O 및 데이터 접근 효율을 높여 성능 향상
테이블 추가	**중복 테이블 추가**	다른 업무이거나, 서버거 다른 경우 테이블 구조를 중복하여 원격조인을 제거하여 성능을 향상
	통계테이블 추가	SUM, AVG 등을 미리 수행하여 계산해둠으로써 조회 시 성능을 향상
	이력테이블 추가	이력 테이블 중에서 마스터 테이블에 존재하는 레코드를 중복하여 이력테이블에 존재하는 방법은 반정규화 유형
	부분 테이블 추가	하나의 테이블의 전체 칼럼 중 자주 이용하는데 자주 이용하는 집중화된 칼럼들이 있을 때 디스크 I/O 를 줄이기 위해 해당 칼럼들을 모아놓은 별도의 반정규화된 테이블을 생성

2-4 대량 데이터에 따른 성능

> 대량 데이터 발생에 따른 테이블 분할 개요

- 대량의 데이터가 존재하는 테이블에 많은 트랜잭션이 발생하여 성능이 저하되는 테이블 구조에 대해 수평/수직 분할 설계를 통해 성능 저하를 예방할 수 있음
- 테이블의 데이터는 Block 단위로 디스크에 저장된다.
  - 칼럼이 많아지게 되면 하나의 로우를 저장 시 물리적인 디스크에 여러 블록에 데이터가 저장될 가능성이 높아짐
  - 즉 하나의 행을 읽더라도 여러 개의 블록을 읽어야함
  - 자연스레 해당 SQL 문의 Block I/O 가 많아짐

- 대용량 테이블에서 발생할 수 있는 현상

현상	설명
로우체이닝	로우 길이가 길어서 데이터 블록에 모두 저장되지 않고 두개 이상의 블록에 걸쳐 하나의 로우가 저장되어 있는 형태
로우 마이그레이션	데이터 블록에서 수정이 발생하면 수정된 데이터를 해당 데이터 블록에서 저장하지 못하고 다른 블록의 빈 공간을 찾아 저장하는 방식

  - 로우체이닝과 로우마이그레이션이 발생하여 많은 블록에 데이터가 저장되면 데이터 조회 시 절대적인 Block I/O 의 횟수가 많아지게 된다.
  - Block I/O 의 횟수가 많아지면 Disk I/O 를 할 가능성도 높아진다.
  - Disk I/O 를 하게 되는 경우 성능이 급격히 저하 된다

> 대용량 테이블 성능 향상 방안

- 수직분할
- 수평분할
  - HASH PARTITION 적용
  - HASH PARTITION 은 지정된 HASH 조건에 따라 해싱 알고리즘이 적용되어 테이블이 분리됨
  - 설계자는 테이블에 데이터가 정확하게 어떻게 들어갔는지 알 수 없음
  - 성능향상을 위해 사용하며 데이터 보관 주기에 따ㅍ쉽게 삭제하는 기능은 제공될 수 없다.
- 수평/수직분할의 절차
  - 데이터 모델링을 완성한다.
  - 데이터베이스 용량 산정을 한다.
  - 대량 데이터가 처리되는 테이블에 대해서 트랜잭션 처리 패턴을 분석한다.
  - 칼럼 단위로 집중화된 처리가 발생하는지, 로우 단위로 집중화된 처리가 발생되는지 분석하여 집중화된 단위로 테이블을 분리하는 것을 검토한다.

2-5 데이터베이스 구조와 성능

> 슈퍼 타입 / 서브타입 모델

- 업무를 구성하는 데이터의 특징을 분석하여 공통점/차이점을 고려하여 효과적으로 표현할 수 있음
- 공통의 부분을 슈퍼타입으로 모델링하고 공통으로 부터 상속받아 다른 엔터티와 차이가 있는 속성에 대해서는 별도의 서브엔터티로 구분

- 업무의 모습을 정확하게 표현하면서 물리적인 데이터 모델로 변환을 할 때 선택의 폭을 넓힐 수 있는 장점이 있음

> 슈퍼 타입 / 서브타입 모델의 변환의 중요성

- 트랜잭션은 항상 일괄로 처리하는데 테이블은 개별로 유지되어 Union 연산에 의해 성능이 저하될 수 있다.
- 트랜잭션은 항상 서브타입 개별로 처리하는데 테이블은 하나로 통합되어 있어 불필요하게 많은 양의 데이터가 집약되어 있어 성능이 저하되는 경우가 있다.
- 트랜잭션은 항상 슈퍼+서브 타입을 공통으로 처리하는데 개별로 유지되어 있거나 하나의 테이블로 집약되어 있어 성능이 저하되는 경우가 있다.

> 슈퍼/서브타입 데이터 모델의 변환 기술

- 개별로 발생되는 트랜잭션에 대해서는 개별 테이블로 구성
  - 슈퍼타입과 서브타입 각각에 대해 독립적으로 트랜잭션이 발생이 되면 슈퍼타입에도 꼭 필요한 속성만을 가지게 하고 서브타입에도 꼭 필요한 속성 및 자신이 타입에 맞는 데이터만 가지게 하기 위해서 모두 분리하여 1:1 관계를 갖도록 한다.

- 전체를 하나로 묶어 트랜잭션이 발생할 때는 하나의 테이블로 구성
  - 테이블을 개별로 분리하면 불필요한 조인을 유발하거나 불필요한 UNION ALL 과 같은 SQL 구문이 작성되어 성능이 저하

> PK / FK 칼럼 순서와 성능

- PK / FK 칼럼 순서와 성능 개요
  - 테이블에 발생되는 트랜잭션 조회 패턴에 따라 PK/FK 칼럼의 순서를 조정해야 함
  - 성능저하 현상이 많은 부분이 PK 가 여러 개의 속성으로 구성된 복합식별자 일 때 PK 순서에 대해 별로 고려하지 않고 데이터 모델링을 한 경우에 해당
  - 물리적인 데이터 모델링 단계에서는 스스로 생성된 PK 순서 이외에 다른 엔터티로부터 상속받아 발생되는 PK 순서까지 항상 주의하여 표시

- PK 가 복합키일 경우 칼럼 순서가 성능에 영향을 미치는 이유
  - 인덱스 선두 칼럼에 대한 조건이 들어와야 한다. (가능한한 '=' 조건으로)
  - 인덱스 선두 칼럼에 대한 조건이 들어오지 않을 경우 인덱스 전체를 읽거나 테이블 전체를 읽게 됨

- PK 순서의 중요성
  - 데이터 모델 관계에 의해 상속받은 FK 속성들은 SQL WHERE 절에서 조인으로 이용되는 경우가 많으므로, FK 인덱스를 생성해야 성능이 좋음
  - FULL TABLE SCAN 이 발생하지 않도록 순서 구성

2-6 분산 데이터베이스와 성능

> 분산 데이터베이스의 개요

- 여러 곳으로 분산되어 있는 데이터베이스를 하나의 가상 시스템으로 사용할 수 있도록 한 데이터베이스
- 논리적으로 동일한 시스템에 속하지만, 컴퓨터 네트워크를 통해 물리적으로 분산되어 있는 데이터들의 모임. 물리적 Site 분산, 논리적으로 사용자 통합·공유

> 분산 데이터베이스의 투명성

- 분할 투명성 (단편화)
- 위치 투명성
- 지역 사상 투명성
- 중복 투명성
- 장애 투명성
- 병행 투명성

> 분산 데이터 베이스의 장 단점

장점	단점
● 지역자치성, 점증적 시스템 용량 확장 ● 신뢰성과 가용성 ● 효용성과 융통성	● 소프트웨어 개발 비용 ● 오류의 잠재성 증대 ● 처리 비용의 증대 ● 설계, 관리의 복잡성과 비용 ● 불규칙한 응답 속도

● 빠른 응답 속도와 통신비용 절감 ● 데이터의 가용성과 신뢰성 증가 ● 시스템 규모의 적절한 조절 ● 각 지역 사용자의 요구 수용 증대	● 통제의 어려움 ● 데이터 무결성에 대한 위협

> 분산 데이터베이스의 적용기법

- 테이블 위치 분산
- 테이블 분할 분산
- 테이블 복제 분산
- 테이블 요약 분산

Ch03 SQL 기본

3-1 관계형 데이터베이스 개요

> 데이터 베이스

- 넓은 의미에서의 데이터베이스는 일상적인 정보들을 모아 놓은 것 자체를 의미한다.
- 일반적으로 데이터베이스라고 말할 때는 특정 기업이나 조직 또는 개인이 필요에 의해(ex: 부가가치가 발생하는) 데이터를 일정한 형태로 저장해 놓은 것을 의미한다.
- 사용자들은 보다 효율적인 데이터의 관리 뿐만 아니라 예기치 못한 사건으로 인한 데이터의 손상을 피하고, 필요시 필요한 데이터를 복구하기 위한 강력한 기능의 소프트웨어를 필요로 하게 되었고 이러한 기본적인 요구사항을 만족시켜주는 시스템을 DBMS(Database Management System)라고 한다.

> 관계형 데이터 베이스

- 관계형 데이터베이스는 정규화를 통한 합리적인 테이블 모델링을 통해 이상(ANOMALY) 현상을 제거하고 데이터 중복을 피할 수 있으며, 동시성 관리, 병행 제어를 통해 많은 사용자들이 동시에 데이터를 공유 및 조작할 수 있는 기능을 제공
- 관계형 데이터베이스는 메타 데이터를 총괄 관리할 수 있기 때문에 데이터의 성격, 속성 또는 표현 방법 등을 체계화할 수 있고, 데이터 표준화를 통한 데이터 품질을 확보할 수 있는 장점을 가지고 있음
- DBMS 는 인증된 사용자만이 참조할 수 있도록 보안 기능을 제공하고 있다. 테이블 생성 시에 사용할 수 있는 다양한 제약조건을 이용하여 사용자가 실수로 조건에 위배되는 데이터를 입력 한다든지, 관계를 연결하는 중요 데이터를 삭제하는 것을 방지하여 데이터 무결성(Integrity)을 보장
- DBMS 는 시스템의 갑작스런 장애로부터 사용자가 입력, 수정, 삭제하던 데이터가 제대로 반영될 수 있도록 보장해주는 기능과, 시스템 다운, 재해 등의 상황에서도 데이터를 회복/복구할 수 있는 기능을 제공

> SQL 문의 종류

종류	명령어
데이터 조작어 **(DML : Data Manipulation Language)**	**SELECT** **INSERT, UPDATE,DELETE**
데이터 정의어 **(DDL : Data Definition Language)**	**CREATE, ALTER, DROP, RENAME**
데이터 제어어	**GRANT, REVOKE**

(DCL : Data Control Language)	
트랜잭션 제어어 **(TCL : Transaction Control Language)**	**COMMIT,ROLLBACK**

3-2 DDL 데이터 정의어

- 데이터 유형은 데이터베이스의 테이블에 특정 자료를 입력할 때, 그 자료를 받아들일 공간을 자료의 유형별로 나누는 기준
- 선언한 유형이 아닌 다른 종류의 데이터가 들어오려고 하면 데이터베이스는 에러를 발생시킴
- 주의사항
  - 테이블 명은 단수형 권고
  - 테이블 명은 다른 테이블과 중복되면 안됨
  - 한 테이블내에서 컬럼 명이 중복되면 안됨
  - 테이블 생성 문 끝은 ';'로 끝나야 함
  - 데이터 유형은 반드시 지정 해야함
  - 테이블 명과 칼럼 명은 반드시 문자로 시작해야함
  - A-Z, a-z, 0-9, _, $, # 문자만 허용

> 제약 조건

- 제약조건(CONSTRAINT)이란 사용자가 원하는 조건의 데이터만 유지하기 생성함
- 데이터의 무결성을 유지하기 위한 데이터베이스의 보편적인 방법으로 테이블의 특정 칼럼에 설정하는 제약

> 제약 조건의 종류

타입	설명
기본 키(Primary Key)	테이블에 저장된 행을 고유하게 식별하기 위함 하나에 테이블에 단 하나의 기본 키만 정의 가능 기본 키 생성 시 DBMS 는 자동으로 UNIQUE 인덱스를 생성 기본 키 컬럼에는 NULL 입력 불가
고유 키(Unique Key)	테이블에 저장된 행 데이터를 고유하게 식별하기 위해 생성 NULL 은 입력 가능
NOT NULL	NULL 값의 입력을 금지 필수적으로 값이 들어가야하는 칼럼이 됨
CHECK	입력할 수 있는 값 종류 및 범위를 제한한다.
외래 키(Foreign Key)	다른 테이블의 기본 키를 외래 키로 지정하는 경우 생성함(참조무결성제약조건)

> DDL 데이터 정의어 종류

- CREATE
- ALTER
- RENAME
- TRUNCATE
- DROP

3-3 DML 데이터 조작어

- INSERT
- UPDATE
- DELETE
- SELECT
- SELECT DISTINCT : **칼럼 값 기준 중복을 제거한 유일한 값 만을 출력함**
  - **SELECT DISTINCT** A.ISSUE_INSTI_NM **FROM** SQLD.TB_CERTI A**;**
  - **SELECT * : 모든 컬럼이 조회됨**
- **ALIAS 지정 : AS 를 이용하여 컬럼의 이름을 지정할 수 있다.**

> 합성 연산자를 이용한 문자열 연결

```
SELECT
    A.CERTI_NM || '(' || A.CERTI_CD || ')' || '-' || A.ISSUE_INSTI_NM  AS CERTI_INFO
FROM SQLD.TB_CERTI A;

" || " 연산자를 이용하여 문자열을 연결 할 수 있다.
```

3-4 TCL

- 트랜잭션은 데이터베이스의 논리적 연산단위이다.
- 하나의 트랜잭션에는 하나 이상의 SQL 문장이 포함된다. 트랜잭션은 분할할 수 없는 최소의 단위이다. 그렇기 때문에 전부 적용하거나 전부 취소한다. 즉, TRANSACTION 은 ALL OR NOTHING 의 개념

특성	설명
원자성	트랜잭션에서 정의된 연산들은 모두 성공적으로 끝나거나 모두 실패해야 한다.
일관성	트랜잭션이 실행되기 전의 데이터베이스의 내용이 잘못되어 있지 않다면 실행된 이후에서 데이터베이스의 내용에 잘못이 있으면 안된다.
고립성	트랜잭션이 실행되는 도중에 다른 트랜잭션의 영향을 받아 잘못된 결과를 만들어서는 안된다.
지속성	트랜잭션이 성공적으로 수행되면 그 트랜잭션이 갱신한 데이터베이스의 내용은 영구적으로 저장된다

> COMMIT

- 입력한 자료나 수정한 자료에 대해서 또는 삭제한 자료에 대해서 전혀 문제가 없다고 판단되었을 경우 COMMIT 명령어를 통해서 트랜잭션을 완료할 수 있다.
- COMMIT 이나 ROLLBACK 이전의 데이터 상태
  - 단지 메모리 BUFFER 에만 영향을 받았기 때문에 데이터의 변경 이전 상태로 복구 가능하다.
  - 현재 사용자는 SELECT 문장으로 결과를 확인 가능하다.
  - 다른 사용자는 현재 사용자가 수행한 명령의 결과를 볼 수 없다.
  - 변경된 행은 잠금(LOCKING)이 설정되어서 다른 사용자가 변경할 수 없다.
- COMMIT 이후의 상태
  - 데이터에 대한 변경 사항이 데이터베이스에 반영된다.
  - 이전 데이터는 영원히 잃어버리게 된다.
  - 모든 사용자는 결과를 볼 수 있다.
  - 관련된 행에 대한 잠금(LOCKING)이 풀리고, 다른 사용자들이 행을 조작할 수 있게 된다.

> ROLLBACK

- 테이블 내 입력한 데이터나, 수정한 데이터, 삭제한 데이터에 대하여 COMMIT 이전에는 변경 사항을 취소할 수 있는데 데이터베이스에서는 롤백(ROLLBACK) 기능을 사용한다.
- 롤백(ROLLBACK)은 데이터 변경 사항이 취소되어 데이터의 이전 상태로 복구되며, 관련된 행에 대한 잠금(LOCKING)이 풀리고 다른 사용자들이 데이터 변경을 할 수 있게 된다.

> SAVE POINT

- 저장점(SAVEPOINT)을 정의하면 롤백(ROLLBACK)할 때 트랜잭션에 포함된 전체 작업을 롤백 하는 것이 아니라 현 시점에서 SAVEPOINT 까지 트랜잭션의 일부만 롤백 할 수 있다.
- 복잡한 대규모 트랜잭션에서 에러가 발생했을 때 SAVEPOINT 까지의 트랜잭션만 롤백하고 실패한 부분에 대해서만 다시 실행

> COMMIT 및 ROLLBACK 처리와는 상관없이 트랜잭션 처리가 일어나는 상황

- CREATE, ALTER, DROP, RENAME, TRUNCATE TABLE 등 DDL 문장을 실행하면 그 전후 시점에 자동으로 커밋 된다.
- DML 문장 이후에 커밋 없이 DDL 문장이 실행되면 DDL 수행 전에 자동으로 커밋 된다.
- 데이터베이스를 정상적으로 접속을 종료하면 자동으로 트랜잭션이 커밋 된다.
- 애플리케이션의 이상 종료로 데이터베이스와의 접속이 단절되었을 때는 트랜잭션이 자동으로 롤백 된다.

3-5 WHERE 조건절

- 사용자들은 자신이 원하는 자료만을 검색하기 위해서 SQL 문장에 WHERE 절을 이용하여 자료들에 대하여 제한할 수 있다. WHERE 절은 FROM 절 다음에 위치하며, 조건식은 아래 내용으로 구성된다
- 연산자의 우선순위

우선순위	설명
1	() 괄호
2	NOT 연산자
3	비교연산자, SQL 비교연산자
4	AND
5	OR

> BETWEEN, IN , LIK, IS NULL 의 사용

- BETWEEN : A 와 B 사이의 값
- IN : 리스트에 있는 값 중 하나라도 있으면 된다.
- LIKE : 비교문자열의 형태와 일치하면 된다.
  - 와일드카드
    - % : 0 개 이상의 어떤 문자를 의미한다.
    - — : 1 개인 단일 문자를 의미한다.
- IS NULL : 값이 NULL 이면 된다.

3-6 함수

- SELECT, WHERE, ORDER BY 절에 사용 가능하다.
- 각 행(Row)들에 대해 개별적으로 작용하여 데이터 값들을 조작하고, 각각의 행에 대한 조작 결과를 리턴 한다.

- 여러 인자(Argument)를 입력해도 단 하나의 결과만 리턴 한다.
- 함수의 인자(Arguments)로 상수, 변수, 표현식이 사용 가능하고, 하나의 인수를 가지는 경우도 있지만 여러 개의 인수를 가질 수도 있다.
- 특별한 경우가 아니면 함수의 인자(Arguments)로 함수를 사용하는 함수의 중첩이 가능하다.

> 단일 행 함수의 종류

종류	설명
문자 형 함수	문자를 입력하면 문자나 숫자값을 반환한다. (LOWER, UPPER, SUBSTR, LENGTH, LTRIM, RTRIM, TRIM, ASCII)
숫자 형 함수	숫자를 입력하면 숫자값을 반환한다. (ABS, MOD, ROUND, TRUNC, SIGN, CHR, CEIL, FLOOR, EXP, LOG, LN, POWER, SIN, COS, TAN)
날짜 형 함수	DATE 타입의 값을 연산한다. (SYSDATE, EXTRACT, TO_NUMBER)
변환 형 함수	문자, 숫자, 날짜형의 값의 데이터 타입을 변환한다. (TO_NUMBER, TO_CHAR, TO_DATE, CONVERT)
NULL 관련 함수	NULL 을 처리하기 위한 함수 (NVL, NULLIF, COALESE)

3-7 GROUP BY, HAVING 절

> 집계 함수

- 여러 행들의 그룹이 모여서 그룹당 단 하나의 결과를 돌려주는 함수이다.
- GROUP BY 절은 행들을 소 그룹화 한다.
- SELECT 절, HAVING 절, ORDER BY 절에 사용할 수 있다.

> 집계함수의 종류

항목	설명
COUNT(*)	NULL 값을 포함한 행의 수를 출력
COUNT(표현식)	표현식의 값이 NULL 아닌 행의 수를 출력
SUM(표현식)	표현식이 NULL 값인 것을 제외한 합계를 출력
AVG(표현식)	표현식이 NULL 값인 것을 제외한 평균을 출력

MAX(표현식)	표현식이 NULL 값인 것을 제외한 최대값을 출력
MIN(표현식)	표현식이 NULL 값인 것을 제외한 최소값을 출력
STDDEV(표현식)	표현식이 NULL 값인 것을 제외한 표준편차를 출력
VARIAN(표현식)	표현식이 NULL 값인 것을 제외한 분산을 출력

> GROUP BY 절

- GROUP BY 절을 통해 소그룹 별 기준을 정한 후, SELECT 절에 집계 함수를 사용한다.
- 집계 함수의 통계 정보는 NULL 값을 가진 행을 제외하고 수행한다.
- GROUP BY 절에서는 SELECT 절과는 달리 ALIAS 명을 사용할 수 없다.
- 집계 함수는 WHERE 절에는 올 수 없다. (집계 함수를 사용할 수 있는 GROUP BY 절보다 WHERE 절이 먼저 수행된다)
- WHERE 절은 전체 데이터를 GROUP 으로 나누기 전에 행들을 미리 제거시킨다.
- HAVING 절은 GROUP BY 절의 기준 항목이나 소그룹의 집계 함수를 이용한 조건을 표시할 수 있다.
- GROUP BY 절에 의한 소 그룹별로 만들어진 집계 데이터 중, HAVING 절에서 제한 조건을 두어 조건을 만족하는 내용만 출력한다.
- HAVING 절은 일반적으로 GROUP BY 절 뒤에 위치한다.

> HAVING 절

- WHERE 절에서는 집계 함수를 쓸 수 없다.
- 집계된 결과 집합을 기준으로 특정 조건을 주고 싶은 경우 HAVING 절을 이용하면 된다.
- HAVING 절은 WHERE 절과 비슷하지만 그룹을 나타내는 결과 집합의 행에 조건이 적용된다는 점에서 차이가 있다.

3-8 ORDER BY 정렬

- ORDER BY 절은 SQL 문장으로 조회된 데이터들을 다양한 목적에 맞게 특정 칼럼을 기준으로 정렬하여 출력하는데 사용
- ORDER BY 절에 칼럼(Column)명 대신에 SELECT 절에서 사용한 ALIAS 명이나 칼럼 순서를 나타내는 정수도 사용 가능
- 별도로 정렬 방식을 지정하지 않으면 기본적으로 오름차순이 적용되며, SQL 문장의 제일 마지막에 위치
- 숫자 형 데이터 타입은 오름차순으로 정렬했을 경우에 가장 작은 값부터 출력, 날짜 형 데이터 타입은 오름차순으로 정렬했을 경우 날짜 값이 가장 빠른 값이 먼저 출력
- NULL 값을 가장 큰 값으로 간주하여 오름차순으로 정렬했을 경우에는 가장 마지막에, 내림차순으로 정렬했을 경우에는 가장 먼저 위치

- ORDERBY – 널 포함 정렬
  - 오라클은 널 값이 가장 크다고 인식한다
  - 내림차순의 결과 널인 행이 가장 위에 위치한다.

- ORDERBY – SELECT 절에 존재하지 않는 컬럼으로 정렬
  - SELECT 절에 기재하지 않은 칼럼을 기준으로 ORDER BY 해도 정상적으로 실행됨
  - SELECT 절에 CERTI_NM 칼럼은 존재하지 않음
  - ORDER BY 절에 CERTI_NM 칼럼으로 ORDER BY 함 정상 수행됨

- ORDERBY – HAVING 절의 결과를 정렬

3-9 조인

- 두 개 이상의 테이블들을 연결 또는 결합하여 데이터를 출력하는 것을 조인이라고 하며 일반적으로 사용 되는 SQL 문의 상당수가 조인으로 이루어져 있다.
- 일반적인 경우 PRIMARY KEY 와 FOREIGN KEY 의 값 연관에 의해 조인이 이루어지며 PK, FK 관계와는 별도로 일반 칼럼 끼리 조인이 이루어지는 경우도 있다

Ch04 SQL 활용

4-1 표준 조인

> 일반 집합 연산자와 SQL 의 비교

일반 집합 연산자	SQL 문	설명
UNION	UNION	UNION 연산은 수학적 합집합을 제공하기 위해, 공통 교집합의 중복을 없애기 위한 사전 작업으로 시스템에 부하를 주는 정렬 작업이 발생한다
INTERSECTION	INTERSECT	INTERSECTION 은 수학의 교집합으로써 두 집합의 공통 집합을 추출한다
DIFFERENCE	EXCEPT(ORACLE MINUS)	DIFFERENCE 는 수학의 차집합으로써 첫 번째 집합에서 두 번째 집합과의 공통 집합을 제외한 부분이다.
PRODUCT	CROSS JOIN	PRODUCT 의 경우는 CROSS(ANIS/ISO 표준) PRODUCT 라고 불리는 곱집합으로 JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합을 말한다

> 순수 관계 연산자와 SQL 비교

일반 집합 연산자	SQL 문	설명
SELECT 연산	WHERE 절로 구현	SELECT 연산은 SQL 문장에서는 WHERE 절 기능으로 구현이 되었다.
PROJECT 연산	SELECT 절로 구현	PROJECT 연산은 SQL 문장에서는 SELECT 절의 칼럼 선택 기능으로 구현되었다
(NATURAL) JOIN	다양한 JOIN 기능으로 구현	JOIN 연산은 WHERE 절의 INNER JOIN 조건과 함께 FROM 절의 NATURAL JOIN, INNER JOIN, OUTER JOIN, USING 조건절, ON 조건절 등으로 가장 다양하게 발전하였다.
DIVIDE	사용되지 않음	

> 조인의 형태

일반 집합 연산자	설명
INNER JOIN	INNER JOIN 은 OUTER(외부) JOIN 과 대비하여 내부 JOIN 이라고 하며 JOIN 조건에서 동일한 값이 있는 행만 반환
NATURAL JOIN	NATURAL JOIN 은 두 테이블 간의 동일한 이름을 갖는 모든 칼럼들에 대해 EQUI(=) JOIN 을 수행
USING 조건절	NATURAL JOIN 에서는 모든 일치되는 칼럼들에 대해 JOIN 이 이루어지지만, FROM 절의 USING 조건절을 이용하면 같은 이름을 가진 칼럼들 중에서 원하는 칼럼에 대해서만 선택적으로 EQUI JOIN 을 할 수가 있음
ON 조건절	JOIN 서술부(ON 조건절)와 비 JOIN 서술부(WHERE 조건 절)를 분리하여 이해가 쉬우며, 칼럼 명이 다르더라도 JOIN 조건을 사용할 수 있는 장점이 있음
CROSS JOIN	CROSS JOIN 은 E.F.CODD 박사가 언급한 일반 집합 연산자의 PRODUCT 의 개념으로 테이블 간 JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합을 말함
OUTER JOIN	INNER(내부) JOIN 과 대비하여 OUTER(외부) JOIN 이라고 불리며 JOIN 조건에서 동일한 값이 없는 행도 반환할 때 사용할 수 있음

> OUTER JOIN

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

4-2 집합 연산자

> 집합 연산자의 종류

종류	설명
UNION	여러 개의 SQL 문의 결과에 대한 합집합 중복된 행은 한개의 행으로 출력됨
UNION ALL	여러 개의 SQL 문의 결과에 대한 합집합 중복된 행도 그대로 결과로 표시한다
INTERSECT	여러 개의 SQL 문의 대한 교집합 중복된 행은 하나로 표시한다
EXCEPT	위의 SQL 문의 집합에서 아래의 SQL 문의 집합을 뺀 결과를 표시한다

> 계층 형 질의와 SELF 조인

- 테이블에 계층 형 데이터가 존재하는 경우 데이터를 조회하기 위해서 계층 형 질의(Hierarchical Query)를 사용
- 계층 형 데이터란 동일 테이블에 계층적으로 상위와 하위 데이터가 포함된 데이터를 말한다.

> 오라클의 계층형 SQL

구분	설명
SELECT	• 조회하고자 하는 칼럼을 지정한다.
FROM TABLE	• 대상 테이블을 지정한다.
WHERE	• 모든 전개를 수행한 후에 지정된 조건을 만족하는 데이터만 추출한다.(필터링)
START WITH 조건	• 계층 구조 전개의 시작 위치를 지정하는 구문이다. 즉, 루트 데이터를 지정한다.
CONNECT BY [NOCYCLE] [PRIOR] A AND B	• CONNECT BY절은 다음에 전개될 자식 데이터를 지정하는 구문이다. • PRIOR 자식 = 부모 형태를 사용하면 계층구조에서 자식 데이터에서 부모 데이터(자식 → 부모) 방향으로 전개하는 순방향 전개를 한다. • PRIOR 부모 = 자식 형태를 사용하면 반대로 부모 데이터에서 자식 데이터(부모 → 자식) 방향으로 전개하는 역방향 전개를 한다. • NOCYCLE를 추가하면 사이클이 발생한 이후의 데이터는 전개하지 않는다.
ORDER SIBLINGS BY 칼럼	• 형제 노드(동일 LEVEL) 사이에서 정렬을 수행한다.

4-4 서브쿼리

> 서브쿼리란

- 서브 쿼리(Subquery)란 하나의 SQL 문안에 포함되어 있는 또 다른 SQL 문을 말한다.
- 조인은 조인에 참여하는 모든 테이블이 대등한 관계에 있기 때문에 조인에 참여하는 모든 테이블의 칼럼을 어느 위치에서라도 자유롭게 사용할 수 있다. 그러나 서브 쿼리는 메인 쿼리의 칼럼을 모두 사용할 수 있지만 메인 쿼리는 서브 쿼리의 칼럼을 사용할 수 없다.

> 서브 쿼리 사용시 주의점

- 서브쿼리를 괄호로 감싸서 사용한다.
- 서브 쿼리는 단일 행(Single Row) 또는 복수 행(Multiple Row) 비교 연산자와 함께 사용 가능하다.
- 단일 행 비교 연산자는 서브 쿼리의 결과가 반드시 1 건 이하이어야 하고 복수 행 비교 연산자는 서브 쿼리의 결과 건수와 상관 없다.
- 서브쿼리에서는 ORDER BY 를 사용하지 못한다. ORDER BY 절은 SELECT 절에서 오직 한 개만 올 수 있기 때문에 ORDER BY 절은 메인 쿼리의 마지막 문장에 위치해야 한다.

> 서브 쿼리가 사용 가능한 위치

- SELECT 절 - FROM 절 - WHERE 절 - HAVING 절 - ORDER BY 절
- INSERT 문의 VALUES 절 - UPDATE 문의 SET 절

> 동작 방식에 따른 서브쿼리 분류

- 비 연관 서브쿼리
  - 서브 쿼리가 메인 쿼리의 칼럼을 가지고 있지 않은 형태의 서브 쿼리임
  - 메인 쿼리에 값을 제공하기 위한 목적으로 주로 사용
- 연관 서브쿼리
  - 서브 쿼리가 메인 쿼리의 값을 가지고 있는 형태의 서브쿼리이다.
  - 일반적으로 메인 쿼리가 먼저 수행되어 읽혀진 데이터를 서브쿼리에서 조건이 맞는지 확인하고자 할 때 주로 사용한다.

> 반환 형태에 따른 서브 쿼리

- **단일 행 서브 쿼리**
  - 서브 쿼리의 실행 결과가 항상 1 건 이하인 서브쿼리를 의미한다.
  - 항상 비교 연산자와 함께 사용된다.
  - (=, <, <=, >, >=, <>)

- **다중 행 서브 쿼리**
  - 서브 쿼리의 실행 결과가 여러 건인 서브쿼리를 의미한다.
  - 다중 행 서브 쿼리는 다중 행 비교 연산자와 함께 사용된다.
  - (IN, ALL, ANY, SOME, EXISTS)

- **다중 칼럼 서브 쿼리**
  - 서브 쿼리의 실행 결과로 여러 칼럼을 반환한다.
  - 메인 쿼리의 조건 절에 여러 칼럼을 동시에 비교 할 수 있다.
  - 서브 쿼리와 메인 쿼리의 칼럼 수와 칼럼 순서가 동일해야 한다

- EXIST 문 서브쿼리
- 스칼라 서브쿼리
- 인라인 뷰 서브쿼리
- HAVING 절에서 서브 쿼리
- UPDATE 문에 사용되는 서브 쿼리

- INSERT 문에 사용되는 서브 쿼리

> 뷰 사용의 장점

장점	설명
독립성	테이블 구조가 변경되어도 뷰를 사용하는 응용프로그램은 변경하지 않아도 된다.
편리성	복잡한 질의를 뷰로 생성함으로써 관련 질의를 단순하게 작성할 수 있다. 또한 해당 형태의 SQL 문을 자주 사용할 때 뷰를 이용하면 편리하게 사용할 수 있다
보안성	직원의 급여 정보와 같이 숨기고 싶은 정보가 존재한다면, 뷰를 생성 할 때 해당 칼럼을 빼고 생성함으로써 사용자에게 정보를 감출 수 있다

4-5 그룹 함수

> 그룹 함수란?

- 그룹 함수를 이용하여 특정 집합의 소계, 중계, 합계, 총 합계를 구할 수 있다.
- 즉 이러한 합계를 계산하기 위해서 기존에 들어갔던 다양한 노력들이 그룹 함수를 이용하여 간단하게 처리 할 수 있게 되었다

> 그룹 함수의 종류

- **ROLLUP**
  - 소 그룹간의 소계를 계산하는 기능
  - ROLLUP 함수 내에 인자로 지정된 GRUPING 칼럼은 SUBTOTAL 을 생성하는데 사용된다.
  - GROUPING 칼럼의 수가 N 이라고 했을 때 N+1 의 SUBTOTAL 이 생성된다.
  - ROLLUP 함수 내의 인자의 순서가 바뀌면 결과도 바뀌게 된다.(ROLLUP 은 계층 구조임)
- **CUBE**
  - 다차원적인 소계를 계산하는 기능
  - 결합 가능한 모든 값에 대하여 다차원 집계를 생성
  - CUBE 함수 내에 칼럼이 N 개라면 2 의 N 승만큼의 SUBTOTAL 이 생성됨
  - 시스템에 많은 부담을 주기때문에 사용상 주의가 필요함
- **GROUPING SETS**
  - **특정 항목에 대한 소계를 계산하는 기능**

4-6 윈도우 함수

>윈도우 함수 개요

- 행과 행간의 관계에서 다양한 연산 처리를 할 수 있는 함수가 윈도우 함수이다.
- 분석함수로도 알려져 있다. (ANSI 표준은 윈도우 함수이다.)
- 윈도우함수는 일반 함수와 달리 중첩하여 호출 될 수는 없다.

> 윈도우 함수의 종류

- **순위 관련함수**
  - RANK
  - DENSE_RANK
  - ROW_NUMBER
- **집계관련함수**
  - SUM
  - MAX
  - MIN

- AVG
- COUNT

- **행순서관련함수**
  - FIRST_VALUE
  - LAST_VALUE
  - LAG
  - LEAD
- **그룹내 비율관련 함수**
  - CUME_DIST
  - PERCENT_RANK
  - NTILE
  - RATIO_TO_REPORT

4-7 DCL

> DCL 이란?

- 유저를 생성하고 권한을 제어할 수 있는 명령어
- 데이터의 보호와 보안을 위해서 유저와 권한을 관리 해야함

> 오라클에서 제공하는 유저들

- **SCOTT**
  - 테스트용 샘플 유저
- **SYS**
  - 테스트용 샘플 유저
- **SYSTEM**
  - SYSTEM 데이터베이스의 모든 시스템 권한을 부여 받은 유저(SYS 바로 밑)

> 유저 생성과 시스템 권한 부여

- 유저 생성후, 접속을 위한 권한 부여 필요,
- 모든 DDL 문장 수행을 위한 권한 부여 필요
- 개별로 부여 하기 보다는 ROLE 을 통한 권한 부여

> OBJECT 에 대한 권한 부여

- 오브젝트 권한과 오브젝트와의 관계
- 개별 오브젝트에 대한 작업을 위해서는 오브젝트 권한 부여 필요

> ROLE 을 이용한 권한 부여

- 유저를 생성하면 다양한 많은 권한들을 부여 해야함
- DBA 는 ROLE 을 생성하고 ROLE 에 각종 권한을 부여한 후 해당 ROLE 을 다른 유저에게 부여
- ROLE 에 포함된 권한들이 필요한 유저에게 빠르게 권한을 부여할 수 있음

4-8 절차형 SQL

> 절차형 SQL 이란?

- 일반적인 개발언어처럼 SQL 문도 절차지향적인 프로그램 작성이 가능하도록 절차 형 SQL 을 제공한다.
- 절차 형 SQL 을 사용하면 SQL 문의 연속적인 실행이나 조건에 따른 분기 처리를 수행하는 모듈을 생성할 수 있다.
- 오라클 기준 이러한 절차 형 모듈의 종류는 사용자정의함수, 프로시저, 트리거가 있다.
- 오라클 기준 이러한 절차 형 모듈을 PL/SQL 이라고 부른다.

> PL / SQL 개요

- PL/SQL 은 Block 구조로 되어 있고 Block 내에는 SQL 문, IF, LOOP 등이 존재함
- PL/SQL 을 이용해서 다양한 모듈을 개발 가능
  - **Block 구조로 되어있으며 각 기능별로 모듈화가 가능**
  - 변수/상수 선언 및 IF/LOOP 문 등의 사용이 가능
  - DBMS 에러나 사용자 에러 정의를 할 수 있음
  - PL/SQL 은 오라클에 내장 시킬수 있으므로 어떠한 오라클 서버로도 이식이 가능
  - PL/SQL 은 여러 SQL 문장을 Block 으로 묶고 한번에 Block 전부를 서버로 보내기때문에 네트워크 패킷 수를 감소 시킴

> PL/SQL BLOCK 구조

구조 명	필수/선택	설명
DECLARE(선언 부)	필수	• BEGIN~END에서 사용할 변수나 인수에 대한 정의 및 데이터 타입 선언
BEGIN(실행 부)	필수	• 개발자가 처리하고자 하는 SQL문과 필요한 LOGIC (비교 문, 제어 문 등)이 정의되는 실행 부
EXCEPTION(예외 처리 부)	선택	• BEGIN~END에서 실행되는 SQL문에 발생된 에러를 처리하는 에러 처리 부
END	필수	

> 사용자 정의 함수란?

- 사용자 정의 함수는 프로시저처럼 SQL 문을 IF/LOOP 등의 LOGIC 와 함께 데이터베이스에 저장해 놓은 명령문의 집합이다.

- SUM, AVG, NVL 의 함수처럼 호출해서 사용할 수 있다.
- 프로시저와 차이점은 반드시 한 건을 되돌려 줘야 한다는 것이다.

> 트리거란?

- 특정한 테이블에 INSERT, UPDATE, DELETE 를 수행할 때 DBMS 내에서 자동으로 동작하도록 작성된 프로그램이다.
- 즉 사용자가 직접 호출하는 것이 아니고 DBMS 가 자동적으로 수행한다.
- 트리거는 테이블과 뷰, DB 작업을 대상으로 정의 할수 있으며 전체 트랜잭션 작업에 대해 발생되는 트리거와 각행에 대해 발생되는 트리거가 있다.

> 프로시저와 트리거의 차이점

프로시저	트리거
CREATE PROCEDURE 문법 사용	CREATE TRIGGER 문법 사용
EXECUTE/EXEC 명령어로 실행	생성 후 자동으로 실행
내부에서 COMMIT, ROLLBACK 실행가능	내부에서 COMMIT, ROLLBACK 실행 안됨

Ch05 SQL 최적화 기본원리

5-1 옵티마이저와 실행계획

> 옵티마이저란?

- 사용자가 질의한 SQL 문에 대한 최적의 실행방법을 결정하는 역할을 수행한다. 이러한 최적의 실행방법을 실행 계획이라고 한다.
- 사용자의 요구사항을 만족하는 결과를 추출할 수 있는 다양한 실행 방법이 존재함
- 다양한 실행 방법들 중에서 최적의 실행방법을 결정하는 것이 옵티마이저의 역할이다.

>비용기반 옵티마이저

- SQL 문을 처리하는데 비용이 가장 적게 드는 실행계획을 선택하는 방식이다. 비용이란 SQL 문을 처리하는데 예상되는 시간 또는 자원 의미한다.
- 테이블, 인덱스 등의 통계 정보와 시스템 통계정보를 이용하여 최적의 실행계획을 도출한다.
- 인덱스를 사용하는 비용이 전체 테이블 스캔 비용보다 크다고 판단되면 테이블 풀 스캔을 유도 하게 된다.

>옵티마이저의 구성요소

- **구성요소**
  - **질의 변환기**
    - 사용자가 작성한 SQL 문을 처리하기에 보다 용이한 형태로 변환
  - **비용 예측기**
    - 대안 계획 생성시에 의해서 생성된 대안 계획의 비용을 예측하는 모듈
    - 대안 계획의 정확한 비용을 측정하기 위해서 연산의 중간 집합의 크기 및 결과 집합의 크기, 분포도 등의 예측을 함, 보다 나은 예측을 위해서 정확한 통계 정보가 필요함
  - **대안계획생성기**
    - **동일한 결과를 생성하는 다양한 대안 계획을 생성하는 모듈**
    - **대안 계획은 연산의 적용 순서, 연산방법변경, 조인 순서 변경 등을 통해서 생성**
    - **동일한 결과를 생성하는 가능한 모든 대안 계획을 생성해야 보다 나은 최적화를 수행할 수 있음**

5-2 인덱스 기본

> 인덱스란?

- 인덱스는 원하는 데이터를 쉽게 찾을 수 있도록 돕는 책의 찾아보기와 유사한 개념이다.
- 검색조건에 부합하는 데이터를 효과적으로(빠르게) 검색할 수 있도록 돕는다.
- 한 테이블은 0 개~N 개의 인덱스를 가질 수 있다.
- 한 테이블에 과도하게 많은 인덱스가 존재하면 INSERT, UPDATE, DELETE 와 같은 DML 작업 시 부하가 발생한다

> 트리기반 인덱스 란?

- DBMS 에서 널리 사용되는 가장 일반적인 인덱스이다.
- 루트 블록, 브랜치 블록, 리프 블록으로 구성된다.
- 가장 상위에 존재하는 블록이 루트 블록이고 브랜치 블록은 분기를 목적으로 하는 블록이다.
- 리프 블록은 트리의 가장 아래 단계에 존재하는 블록이다.
- 리프 블록은 인덱스를 구성하는 칼럼의 데이터와 해당 데이터를 가지고 있는 행의 위치를 가리키는 레코드 식별자 인 ROWID 로 구성되어 있다.

> 인덱스 구조 상세

- 루프와 브랜치 블록에 있는 각 레코드는 하위 블록에 대한 주소 값을 갖는다. 키 값은 하위 블록에 저장된 키 값의 범위를 나타낸다.
- LMC 가 가리키는 주소로 찾아간 블록에는 키 값을 가진 첫번째 레코드보다 작거나 같은 레코드가 저장돼 있다.
- 리프 블록에 저장된 각 레코드는 키 값 순으로 정렬돼 있을 뿐만 아니라 테이블 레코드를 가리키는 주소값 즉 Rowid 를 갖는다.
- 인덱스 키 값이 같으면 Rowid 순으로 정렬된다.
- 인덱스를 스캔하는 이유는 검색조건을 만족하는 소량의 데이터를 빨리 찾고 거기서 Rowid 를 얻기 위해서이다

> 풀 테이블 스캔과 인덱스 스캔

스캔유캔	설명
풀테이블 스캔	● 테이블에 존재하는 모든 데이터를 읽어가면서 조건에 맞으면 결과로 추출하고 조건에 맞지 않으면 버리는 방식 ● HIGH WATER MARK 는 테이블에 데이터가 쓰여졌던 블록 상의 최상위 위치로써 테이블 풀 스캔 시는 HWM 까지의 블록에 있는 ● 모든 데이터를 읽어야 하기 때문에 시간이 오래 걸릴 수 있다. ● 풀 테이블 스캔으로 읽은 블록은 재 사용성이 낮다고 보고 메모리 버퍼 캐시에서 금방 제거될 수 있도록 관리한다. ● 옵티마이저가 풀 테이블 스캔을 선택하는 경우 ■ SQL 문에 조건이 존재하지 않는 경우 ■ SQL 문의 조건을 기준으로 사용 가능한 인덱스가 없는 경우 ■ 옵티마이저의 판단으로 풀 테이블 스캔이 유리하다고 판단하는 경우 ■ 전체 테이블 스캔을 하도록 강제로 힌트를 지정한 경우
인덱스 스캔	● 인덱스 스캔은 인덱스를 구성하는 칼럼의 값을 기반으로 데이터를 추출하는 액세스 기법 ● 인덱스 리프 블록은 인덱스를 구성하는 칼럼과 ROWID 로 구성 ● 인덱스의 리프 블록을 읽으면 인덱스 구성 칼럼의 값과 ROWID 를 알 수 있음 ● 즉 인덱스를 읽어서 대상 ROWID 를 찾으면 해당 ROWID 로 다시 테이블을 찾아 가야함 ● 하지만 SQL 문에서 필요로 하는 칼럼이 모두 인덱스 구성칼럼이라면 테이블을 찾아갈 필요 없음 ● 일반적으로 인덱스 스캔을 통해 데이터를 추출하면 해당 결과는 인덱스의 칼럼의 순서로 정렬된 상태로 반환됨

> 인덱스 스캔 유형

- **인덱스 범위 스캔**
  - 인덱스를 이용하여 한건 이상의 데이터를 추출하는 방식
  - 인덱스 스캔으로 특정 범위를 스캔하면서 대상 레코드를 하나하나 리턴하는 방식임
- **인덱스 유일 스캔**
  - 인덱스를 사용하여 단 하나의 데이터를 추출하는 방식
  - 유일인덱스는 중복 레코드를 허용하지 않음

- 유일인덱스는 반드시 '='조건으로 조회 해야 됨(그렇게 할 수 밖에 없음)

- **인덱스 전체 스캔**
  - 인덱스를 처음부터 끝까지 전체를 읽으면서 조건에 맞는 데이터를 추출함
  - 데이터를 추출 시 리프 블록에 있는 ROWID 로 테이블의 레코드를 찾아가서 조건에 부합하는지 판단하고
  - 조건에 부합되면 해당 행을 리턴 함
- **인덱스 스킵 스캔**
  - 인덱스 선두 컬럼이 조건절에 없어도 인덱스를 활용하는 스캔 방식이다.
  - 조건절에 빠진 인덱스 선두 컬럼(성별)의 Distinct Value 의 개수가 적고, 후행 컬럼(연봉)의 Distinct Value 의 개수가 많을 때 유용
  - Index Skip Scan 은 루트 또는 브랜치에서 읽은 컬럼 값 정보를 이용해 조건절에 부합하는 레코드를 포함할 가능성이 있는 리프 블록만 액세스 한다.
- **인덱스 고속 전체 스캔**
  - Index Fast Full Scan 은 물리적으로 디스크에 저장된 순서대로 인덱스 리프 블록들을 Multi Block I/O 방식으로 읽어 들인다. 또한 병렬 인덱스 스캔도 가능하다.
- 인덱스 역순 범위 스캔
  - 인덱스 리프 블록은 Doubly Linked List 방식으로 저장되어 있음
  - 즉 이 성질을 이용하여 인덱스를 역순으로(거꾸로) 읽을 수 있음
  - 인덱스를 뒤에서부터 앞쪽으로 스캔하기 때문에 내림차순으로 정렬된 결과 집합을 얻을 수 있다. (스캔 순서를 제외 하고는 Range Scan 과 동일함)

> 테이블 스캔 VS 인덱스 스캔

풀 테이블 스캔	인덱스 스캔
항상 이용 가능	인덱스가 존재해야만 이용가능
한번에 여러 개의 BLOCK을 읽음	한번에 한 개에 블록만을읽음
많은 데이터를 조회 시 성능 상 유리	극히 일부분의 데이터를 조회 시 유리
Table Random Access부하 없음	Table Random Access에 의한 부하가 발생됨
읽었던 블록을 반복해서 읽는 경우 없음	읽었던 블록을 반복해서 읽는 비효율 발생(논리적인 블록 I/O의 개수도 많아짐)

5-2 조인 수행 원리

> 조인이란?

- 조인이란 두개 이상의테이블을 하나의 집합으로 만드는 연산이다.
- SQL 문의 FROM 절에 두개 이상의 테이블 혹은 집합이 존재할 경우 조인이 수행된다.

- 조인은 3 개 이상의 테이블을 조인한다고 하더라도 특정 시점에 2 개의 테이블 단위로 조인이 된다.
- A, B, C 집합을 조인한다면 A, B 조인 후 해당 결과 집합을 C 와 조인 하는 방식이다.
- 각각의 조인 단계에서는 서로 다른 조인 기법이 사용될 수 있다.
- 즉 A, B 조인 시에는 NL 조인을 수행하고 A, B 조인의 결과와 C 를 조인 시에는 해시 조인이 수행될 수 있다.

> NL 조인

NL 조인은 작은 집합이 Driving 되어야 하고, Inner 테이블의 인덱스 스캔이 매우 중요하다.

- RANDOM 액세스 위주(인덱스구성이 완벽 해도 대량 데이터 조인 시 불리)
- 한 레코드 씩 순차 진행(부분 범위 처리를 유도해야 효율적 수행)
- DRIVING 테이블 처리 범위에 의해 전체 성능이 결정됨
- 인덱스 유무, 인덱스 구성에 크게 영향 받음
- 소량의 데이터를 처리하거나 부분범위처리가 가능한 OLTP 환경에 적합

> 소트 머지 조인

정렬 작업을 생략할 수 있는 인덱스가 존재하는 경우 사용

- 실시간 인덱스 생성 : 양쪽 집합을 정렬한 다음에는 NL 조인과 같은 오퍼레이션
- 인덱스 유무에 영향을 받지 않음 : 미리 정렬된 인덱스가 있으면 좀 더 빠르게 수행할 수는 있음
- 양쪽 집합을 개별적으로 읽고 나서 조인 : 조인 컬럼에 인덱스가 없는 상황에서 두 테이블을 독립적으로 읽어 조인 대상 집합을 줄일 수 있을 때 아주 유리
- 스캔(Scan) 위주의 액세스 방식 : 양쪽 소스 집합에서 정렬 대상 레코드를 찾는 작업은 인덱스를 이용 Random 액세스 방식으로 처리될 수 있음

> 해시 조인

작은 집합을 Build Input 으로 하고 큰집합을 probe input 으로 하는 것이 중요

- 대량의 데이터 처리가 필요하고 쿼리 수행 시간이 오래 걸리는 대용량 테이블을 조인할 때(배치 프로그램, DW, OLAP 성 쿼리) 사용
- NL 조인처럼 Random 액세스 부하 없음
- 소트 머지 조인처럼 정렬 부하 없음
- 해시 테이블을 생성하는 비용에 따라서 Build Input 이 (Hash Area 에 담을 수 있을 정도로 충분히) 작을 때라야 효과적