11 데이터베이스 구축

Section 1. 데이터베이스 개념

1. 데이터베이스 개념

(1) 데이터와 정보

- 데이터
  - 현실 세계에서 단순히 관찰하거나 측정하여 수집한 사실이나 값
- 정보
  - 데이터를 의사 결정에 유용하게 활용할 수 있도록 처리하여 체계적으로 정리한 결과물

(2) 데이터베이스

- 특정 조직의 업무를 수행하는 데 필요한 상호 관련된 데이터들의 모임
- 여러 사람들이 공유하고 사용할 목적으로 통합 관리되는 데이터들의 모임

(3) 데이터베이스의 정의

- 통합 데이터(Integrated Data)
  - 검색의 효율성을 위해 중복이 최소화 된 데이터의 모임
- 저장 데이터(Stored Data)
  - 컴퓨터가 접근 가능한 저장 매체에 저장된 데이터
- 운영 데이터(Operational Data)
  - 조직의 목적을 위해 존재 가치가 확실하고 반드시 필요한 데이터
- 공유 데이터(Shared Data)
  - 여러 응용 프로그램들이 공동으로 사용하는 데이터

(4) 데이터베이스의 특징

- 실시간 접근성(Real Time Accessibility)
  - 사용자의 질의에 대하여 즉시 처리하여 응답
- 계속적인 변화(Continuous Evolution)
  - 삽입, 삭제, 갱신을 통하여 항상 최근의 정확한 데이터를 유지
- 동시 공유(Concurrent Sharing)
  - 여러 사용자가 동시에 원하는 데이터를 공유
- 내용에 의한 참조(Content Reference)
  - 데이터베이스에 있는 데이터를 주소가 아닌 내용에 따라 참조
- 데이터의 독립성(Independence)
  - 논리적 독립성 : 데이터의 논리적 구조를 변경시키더라도, 응용 프로그램은 변경되지 않음
  - 물리적 독립성 : 새로운 물리적 구조를 도입하더라도, 응용 프로그램에는 영향을 주지 않음

⑸ 데이터 언어

① DDL(Data Definition Language : 데이터 정의어)

- DB의 구조, 데이터 형식, 접근 방식 등 DB의 구축과 변경 목적으로 사용하는 언어
- 데이터베이스의 논리적, 물리적 구조를 정의 및 변경
- 스키마(Schema)에 사용되는 제약 조건을 정의

② DML(Data Manipulation Language : 데이터 조작어)

- 데이터 처리를 위한 응용 프로그램과 데이터베이스 관리 시스템 간의 인터페이스를 위한 언어
- 데이터의 검색, 삽입, 삭제, 갱신 연산 등을 포함한 집합

③ DCL(Data Control Language : 데이터 제어어)

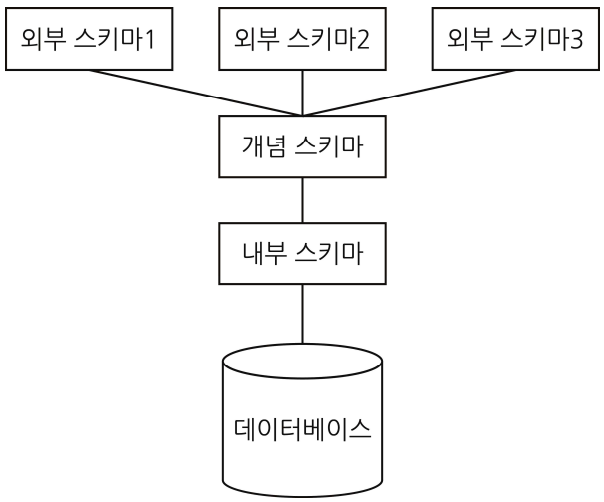
- 보안 및 권한 제어, 무결성, 회복, 병행 제어를 위한 언어

⑹ 스키마(schema)

① 스키마의 정의

- 데이터베이스의 구조와 제약조건에 관해 전반적인 명세를 기술한 것
- 개체의 특성을 나타내는 속성(Attribute)과 속성들의 집합으로 이루어진 개체(Entity), 개체 사이에 존재하는
- 관계(Relation)에 대한 정의와 이들이 유지해야 할 제약조건들을 기술한 것
- 스키마는 데이터 사전(Data Dictionary)에 저장

② 3계층 스키마



- 외부 스키마 (external schema) - 사용자 뷰
  - 데이터베이스의 논리적 구조 정의, 사용자 뷰
  - 전체 데이터베이스의 한 논리적 부분으로 서브스키마라고도 한다.
  - 하나의 데이터베이스에는 여러 개의 외부스키마가 존재 가능
  - 하나의 외부스키마를 여러 개의 응용프로그램이나 사용자가 공용 가능
  - 질의어(SQL)을 이용하거나 C, JAVA 등의 언어를 사용하여 DB에 접근

- 개념 스키마 (Conceptual schema) - 전체적인 뷰
  - 데이터베이스의 전체적인 논리적 구조
  - 모든 응용 프로그램이나 사용자들이 필요로 하는 데이터를 종합한 조직 전체의 데이터베이스로 하나만 존재
  - 데이터베이스 파일에 저장되는 데이터의 형태를 나타내는 것으로, 단순히 스키마(Schema)라고 하면 개념 스키마를 의미
  - 데이터의 접근 권한, 보안 정책, 무결성 규칙에 대한 명세를 정의
- 내부 스키마 (Internal schema) - 저장 스키마
  - 물리적 저장장치의 입장에서 본 데이터베이스 구조
  - 실제로 데이터베이스에 저장될 레코드의 물리적인 구조를 정의하고, 저장 데이터 항목의 표현방법, 내부 레코드의 물리적 순서 등을 나타낸다.

2. 데이터베이스 관리 시스템 (Database Management System)

⑴ DBMS의 정의

- 데이터베이스를 조작하는 별도의 소프트웨어
- DBMS를 통해 데이터베이스를 관리하여 응용 프로그램들이 데이터베이스를 공유하고, 사용할 수 있는 환경을 제공
- 데이터베이스를 구축하는 틀을 제공하고, 효율적으로 데이터를 검색하고 저장하는 기능을 제공
- 응용 프로그램들이 데이터베이스에 접근할 수 있는 인터페이스를 제공하고, 장애에 대한 복구 기능, 사용자 권한에 따른 보안성 유지 기능 등을 제공

⑵ DBMS의 기능

- 데이터 정의
  - 데이터에 대한 형식, 구조, 제약조건들을 명세하는 기능
  - 데이터베이스에 대한 정의 및 설명은 카탈로그나 사전의 형태로 저장
- 데이터 조작
  - 특정한 데이터를 검색하기 위한 질의, 데이터베이스의 갱신, 보고서 생성 기능
- 데이터 제어
  - 데이터 무결성(integrity)
  - 보안(Security)/권한(Authority) 검사
  - 동시성 제어(Concurrency control)
- 데이터 공유
  - 여러 사용자와 프로그램이 데이터베이스에 동시에 접근하도록 하는 기능
- 데이터 보호
  - 하드웨어나 소프트웨어의 오동작 또는 권한이 없는 악의적인 접근으로부터 시스템을 보호
- 데이터 구축
  - DBMS가 관리하는 기억 장치에 데이터를 저장하는 기능
- 유지보수
  - 시간이 지남에 따라 변화하는 요구사항을 반영할 수 있도록 하는 기능

⑶ DBMS의 장점/단점

- 장점
  - 데이터 중복 최소화
  - 데이터 독립성 확보
  - 데이터를 동시 공유
  - 데이터 보안이 향상
  - 데이터 무결성을 유지
  - 장애 발생 시 회복이 가능
- 단점
  - 비용이 많이 든다.
  - 백업과 회복 방법이 복잡

⑷ DBMS의 종류

- 계층형(Hierarchical DataBase)
  - 데이터 간의 관계가 트리 형태의 구조
  - 데이터를 세그먼트(레코드) 단위로 관리하며 세그먼트 간 계층을 트리구조로 관리
  - 구조가 간단하고 구현, 수정, 검색이 쉽지만 부모 자식 간에 N:N(다 대 다) 관계 처리가 불가능하고, 구조 변경이 어렵다.
- 네트워크형(Network DataBase)
  - 계층형 데이터베이스의 단점을 보완하여 데이터 간 N:N(다대다) 구성이 가능한 망 형 모델
  - 계층 구조에 링크를 추가하여 유연성과 접근성을 높였다.
  - 구조가 복잡해 유지보수가 어렵다.
- 관계형(Relational DataBase)
  - 키(key)와 값(value)으로 이루어진 데이터들을 행(row)과 열(Column)로 구성된 테이블 구조로 단순화시킨 모델
  - SQL(Structured Query Language) 를 사용하여 데이터를 처리
- 객체 지향형(Object-Oriented DataBase)
  - 객체지향 프로그래밍 개념에 기반하여 만든 데이터베이스 모델
  - 정보를 객체의 형태로 표현
  - 객체지향 프로그래밍 개념 (클래스, 상속 등)을 사용할 수 있다.
  - 비정형 데이터들을 데이터베이스화 할 수 있도록 하기 위해 만들어진 모델
- 객체 관계형(Object-Relational DataBase)
  - 관계형 데이터베이스에 객체 지향 개념을 도입하여 만든 데이터베이스 모델
  - 객체지향 개념을 지원하는 표준 SQL을 사용할 수 있고, 데이터 타입도 관계형 데이터베이스보다 더 다양하게 추가
- NoSQL
  - Not Only SQL의 줄임말로 SQL뿐만 아니라 다양한 특성을 지원한다는 의미
  - 데이터 간에 관계를 정의하지 않는 데이터베이스 모델로 기존의 RDBMS의 복잡도와 용량의 한계를 극복하기 위한 목적으로 만들어졌다.
  - 비정형 데이터 처리에 유리하지만 스키마 변경이 불가능해 데이터값에 문제가 발생하면 감지가 어렵다.
- NewSQL
  - RDBMS의 SQL과 NoSQL의 장점을 결합하여 관계형 모델
  - 트랜잭션 지원 및 확장성과 고 가용성을 모두 만족시키려는 목적에서 만들어진 데이터베이스 모델

Section 2. 데이터베이스 설계

1. 데이터베이스 설계 개요

(1) 데이터베이스 설계 정의

- 요구조건에서부터 데이터베이스 구조를 도출해 내는 과정
- 데이터들을 효과적으로 관리하기 위하여 데이터베이스의 구조를 조직화하는 작업

(2) 데이터베이스 설계 목적

- 이해관계자의 데이터 관점 요구사항에 대한 정확한 이해 및 추상화
- 데이터를 중심으로 한 이해관계자 간의 원활한 의사소통 수단

(3) 데이터베이스 설계 시 고려사항

- 제약조건
  - 저장된 데이터 값이 만족해야 될 주어진 조건
- 데이터베이스 무결성
  - 데이터의 삽입, 삭제, 갱신 연산이 수행된 뒤에도 데이터 값은 제약조건을 만족해야 하는 조건
- 일관성
  - 저장된 두 데이터 값 또는 특정 질의에 대한 응답들에 모순성 없이 일치하는 특성
- 회복
  - 시스템에 장애가 발생했을 때 장애 발생 직전의 일관된 데이터 상태로 돌아가는 기법
- 보안
  - 불법적인 데이터의 변경이나 손실 또는 노출에 대한 보호
- 효율성
  - 응답 시간의 단축, 저장공간의 최적화, 시스템 생산성이 포함
- 데이터베이스 확장성
  - 시스템 운영에 영향을 주지 않으면서 새로운 데이터를 계속적으로 추가 가능한 기법

2. 데이터베이스 설계 단계

(1) 요구조건 분석

- 데이터베이스의 사용자, 사용목적, 사용범위, 제약조건 등에 대한 내용을 정리하고 명세서를 작성
- 트랜잭션 유형, 트랜잭션 실행빈도와 같은 동적 DB 처리 요구조건 정의

(2) 개념적 설계

- 현실세계를 데이터관점으로 추상화 단계
- DBMS 에 독립적으로 설계
- 데이터베이스의 개념적 스키마 구성(E-R 다이어그램)
- 트랜잭션 모델링 및 정의

(3) 논리적 설계

- 자료를 컴퓨터가 이해할 수 있도록 특정 DBMS의 논리적 자료 구조로 변환하는 과정
- 특정 데이터모델(계층형, 관계형, 객체지향형 등)을 적용한 설계
- 데이터베이스의 논리적 스키마 생성
- 관계형 데이터베이스인 경우 이 단계에서 테이블을 설계하고, 정규화 과정
- 트랜잭션 인터페이스 설계

(4) 물리적 설계

- 특정 DBMS의 물리적 구조와 내부적인 저장구조, 분산형태, 데이터타입의 특징, 인덱스의 특징 등을 구체화하는 설계단계
- 레코드 집중의 분석 및 설계
- 오브젝트, 접근방법, 트랜잭션분석, 인덱스, 뷰, 데이터베이스 용량설계 등을 수행
- 데이터베이스의 물리적 스키마 생성
- 트랜잭션 세부 설계

(5) 구현

- 목표 DBMS의 DDL로 기술된 명령문을 컴파일하고, 실행시켜 데이터베이스 스키마 생성

Section 3. 데이터 모델링

1. 데이터모델 개념

(1) 데이터모델 개념

- 현실세계의 요소를 인간과 컴퓨터가 이해할 수 있는 정보로 표현한 것
- 데이터의 관계, 접근과 그 흐름에 필요한 처리 과정에 관한 추상화된 모형
- 현실 세계의 정보들을 컴퓨터에 표현하기 위해서 단순화, 추상화 하여 체계적으로 표현한 개념적 모형

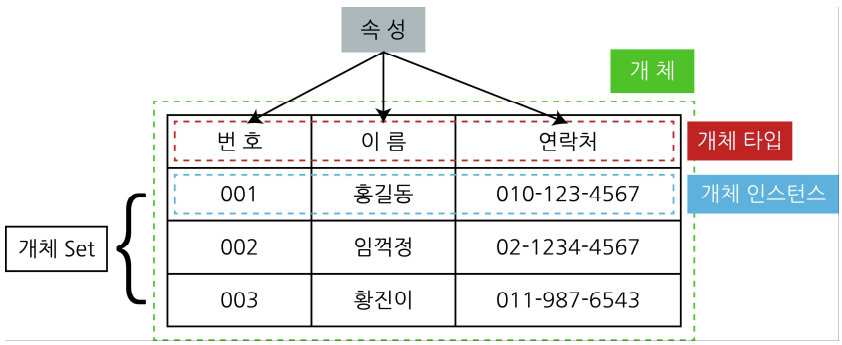
(2) 데이터모델 종류

- 계층형 모델(Hierarchical Data Model)
- 망형 데이터 모델(네트워크형 데이터 모델)
- 관계형 데이터 모델(Relational Data Model)
- 객체 지향형 데이터 모델(Object Oriented Data Model)

(3) 데이터모델 구분

- 개념적 데이터 모델
  - 현실세계에 대한 인식을 추상적인 개념으로 표현
  - 인간이 이해할 수 있는 정보 구조로 표현
  - 대표적으로 개체-관계(E-R) 모델
- 논리적 데이터 모델
  - 개념 데이터 모델링의 개념 구조를 컴퓨터가 이해할 수 있도록 변환한 구조
  - 필드, 데이터타입 등으로 개념적 모델 구현
  - 관계 모델, 계층 모델, 네트워크 모델 등으로 구분
- 물리적 모델
  - 데이터가 저장되는 방법을 표현
  - 레코드형식, 레코드 순서, 접근경로, 저장방법에 관한 정의

(4) 데이터모델 구조



- 개체(Entity)
  - 데이터베이스에 데이터로 표현하려고 하는 현실 세계의 대상체
  - 저장할 만한 가치가 있는 중요 데이터를 가지고 있는 사람이나 사물 등
- 개체 타입 (Entity type)
  - 개체를 구성하고 있는 속성들의 집합

- 개체 인스턴스 (Entity instance)
  - 데이터베이스에 저장되는 구체적인 객체
- 개체 세트 (Entity set)
  - 개체 인스턴스의 집합
- 속성(Attribute)
  - 데이터의 가장 작은 논리적 단위
  - 개체가 가지고 있는 고유한 특성
- 관계(Relation)
  - 개체와 개체가 맺고 있는 의미 있는 연관성

⑸ 데이터모델 표시해야 할 요소

- 구조(Structure)
  - 데이터베이스에 표현될 대상으로서의 개체 타입과 개체 타입들간의 관계
  - 데이터 구조 및 정적 성질
- 연산(Operation)
  - 데이터베이스에 저장될 실제 데이터를 처리하는 방법
- 제약조건(Constraint)
  - 저장될 수 있는 데이터의 논리적인 제약조건

2. 개체-관계 모델 (Entity Relation Model)

⑴ 개체-관계 모델 개념

- 데이터베이스에 대한 요구 사항을 그래픽적으로 표현하는 방법
- 피터 첸이 제안한 개념적 데이터 모델로써 현실 세계를 개체와 개체 간의 관계를 이용하여 개념적 구조로 표현
- 요구사항으로부터 얻어낸 정보들을 개체(Entity), 속성(Attribute), 관계(Relation)로 기술하는 데이터 모델
- 특정 DBMS 및 하드웨어에 독립적으로 데이터베이스의 구조를 나타낼 수 있다.
- 산출물로 개체-관계 다이어그램(Entity-Relationship Diagram)이 만들어진다.

⑵ 개체 (Entity)

- 현실 세계에서 꼭 필요한 사람이나 사물과 같이 구별되는 모든 것
- 데이터로써 DB에 저장할 가치가 있는 중요한 사람, 사물, 개념, 사건 등
- ER 다이어그램에서 개체는 사각형으로 표현

⑶ 애트리뷰트, 속성(Attribute)

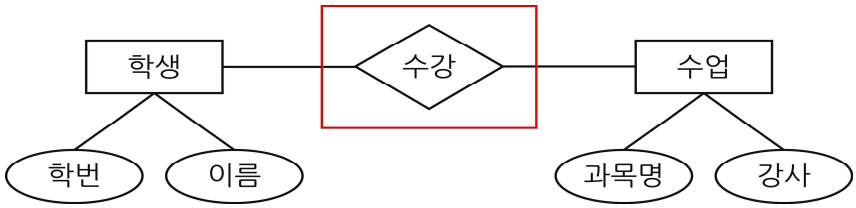
- 개체나 관계가 가지고 있는 고유의 특성
- DB에 저장할 데이터의 가장 작은 논리적 단위
- ER 다이어그램에서 속성은 기본적으로 원으로 표현, 키 속성은 원에 밑줄 표현, 다중값은 두 개의 원으로 표현, 유도 속성은 원을 점선으로 표현

- 속성의 유형

속성 유형	설명
단일 값 속성	- 값을 하나만 가질 수 있는 속성 (ex. 이름, 학번 등)
다중 값 속성	- 값을 여러 개 가질 수 있는 속성 (ex. 취미 등)
단순 속성	- 의미를 더는 분해할 수 없는 속성 (ex. 성별 등)
복합 속성	- 의미를 분해할 수 있는 속성 (ex. 주소, 생년월일 등)
유도 속성	- 기존의 다른 속성의 값에서 유도되어 결정되는 속성 (ex. 주민번호와 성별)
널 속성	- 아직 결정되지 않은 존재 하지 않는 값
키 속성	- 각 개체를 식별하는데 사용하는 속성

(4) 관계 (Relationship)

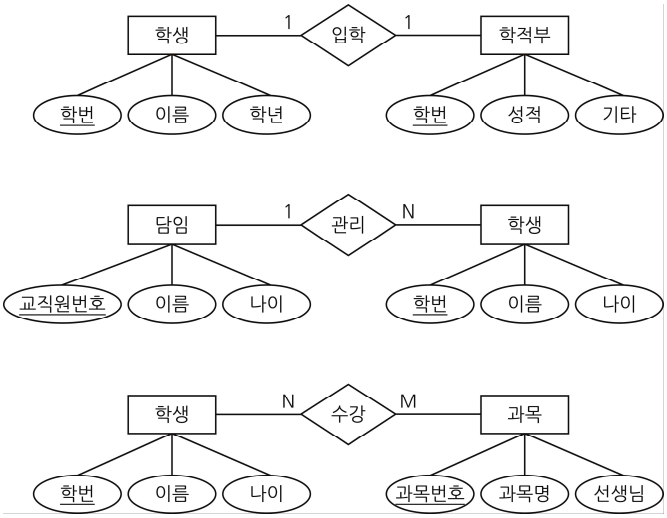
- 서로 다른 개체가 맺고 있는 의미 있는 연관성
- 개체 사이의 대응 관계
- ER 다이어그램에서 개체는 마름모로 표현



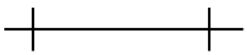
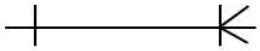
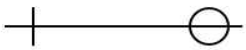
- 개체 간 대응 관계의 종류

종류	설명
1 : 1	- 개체 집합 A의 원소가 개체 집합 B의 원소 1개와 대응
1 : N	- 개체 집합 A의 각 원소는 개체 집합 B의 원소 여러 개와 대응할 수 있고, 개체 집합 B의 각 원소는 개체 집합 A의 원소 1개와 대응
N : M	- 개체 집합 A의 각 원소는 개체 집합 B의 원소 여러 개와 대응할 수 있고, 개체 집합 B의 각 원소도 개체 집합 A의 원소 여러 개와 대응

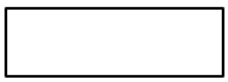
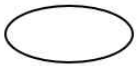
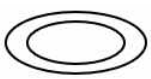
- 개체 간 대응 관계 표현



- 새발 표기법

종류	설명
	- 1:1 관계
	- 1:N 관계
	- N:M 관계
	- 관계가 있을 수도 없을 수도 있음

(5) E-R 다이어그램 기호

기호	기호 이름	설명
	사각형	- 개체(Entity)
	마름모	- 관계(Relationship)
	타원	- 속성(Attribute)
	밑줄 타원	- 기본키 속성
	이중 타원	- 복합속성
	선 링크	- 개체와 속성 연결

3. 데이터 모델의 품질 기준

기준항목	설명
정확성	데이터 모델이 표기법에 따라 정확하게 표현되었고, 업무영역 또는 요구사항이 정확하게 반영되었음을 의미함
완전성	데이터 모델의 구성 요소를 정의하는데 있어서 누락을 최소화하고, 요구사항 및 업무영역 반영에 있어서 누락이 없음을 의미함
준거성	제반 준수 요건들이 누락 없이 정확하게 준수되었음을 의미함
최신성	데이터 모델이 현행 시스템의 최신 상태를 반영하고 있고, 이슈사항들이 지체없이 반영되고 있음을 의미
일관성	여러 영역에서 공통 사용되는 데이터 요소가 전사 수준에서 한 번만 정의되고 이를 여러 다른 영역에서 참조·활용되면서, 모델 표현상의 일관성을 유지하고 있음을 의미함
활용성	작성된 모델과 그 설명 내용이 이해관계자에게 의미를 충분하게 전달할 수 있으면서, 업무 변화 시에 설계 변경이 최소화되도록 유연하게 설계되어 있음을 의미

Section 4. 논리 데이터베이스 설계

1. 논리적 데이터 모델링

⑴ 논리적 모델링

- 개념적 설계에서 추출된 실체와 속성들의 관계를 구조적으로 설계하는 단계
- 개념모델로부터 업무 데이터 및 규칙을 구체적으로 표현한 모델
- 모든 업무용 개체와 속성, 관계, 프로세스 등이 포함됨
- 모든 데이터를 정규화(Normalizaton)하여 모델링
- 성능, 제약사항과는 독립적인 모델로, 특정 DBMS로부터 독립적이다.
- 데이터 간의 관계를 어떻게 표현하느냐에 따라 관계 모델, 계층 모델, 네트워크 모델로 구분

⑵ 논리적 데이터 모델 종류

- 관계형 데이터 모델
- 계층형 데이터 모델
- 네트워크(망)형 데이터 모델
- 객체지향 데이터 모델

2. 데이터베이스 정규화(Normalization)

⑴ 정규화의 개념

- 관계형 데이터베이스의 설계에서 중복을 최소화하게 데이터를 구조화
- 중복된 데이터를 허용하지 않음으로써 무결성(Integrity)를 유지할 수 있으며, DB의 저장 용량 역시 줄일 수 있다.

⑵ 정규화의 목적

- 데이터의 중복을 최소화
- 정보의 무손실 : 정보가 사라지지 않아야 함
- 독립적인 관계는 별개의 릴레이션으로 표현
- 정보의 검색을 보다 용이하게 함
- 이상현상 최소화

⑶ 정규화의 장/단점

- 장점
  - 데이터 중복의 최소화
  - 저장 공간의 효율적 사용
  - 릴레이션에서 발생 가능한 이상 현상 제거
- 단점
  - 처리 명령의 복잡
  - 실행 속도 저하
  - 분리된 두 릴레이션간 참조 무결성 유지를 위한 노력 필요
  - 분리된 여러 개의 테이블에서 정보를 취합하기 위한 JOIN 연산이 필요

⑷ 이상(Anomaly) 현상

- 데이터 중복으로 인해 릴레이션 조작 시 예상하지 못한 곤란한 현상이 발생
- 이상은 속성들 간에 존재하는 여러 종류의 종속 관계를 하나의 릴레이션에 표현할 때 발생
- 이상의 종류
  - 삽입 이상 : 데이터를 삽입할 때 불필요한 데이터가 함께 삽입되는 현상
  - 삭제 이상 : 한 튜플을 삭제할 때 연쇄 삭제 현상으로 인해 정보 손실
  - 갱신 이상 : 튜플의 속성값을 갱신할 때 일부 튜플의 정보만 갱신되어 정보에 모순이 생기는 현상

⑸ 함수적 종속 (Functional Dependency)

① 함수적 종속의 개념

- 어떤 릴레이션 R이 있을때 X와 Y를 각각 속성의 부분집합이라고 가정했을 때
  - X의 값을 알면 Y의 값을 바로 식별할 수 있고, X의 값에 Y의 값이 달라질 때, Y는 X에 함수적 종속이라고 함
  - 이를 기호로 표현하면 X→Y

② 함수적 종속 관계

- 완전 함수적 종속 (Full Functional Dependency)
  - 종속자가 기본키에만 종속되며, 기본키가 여러 속성으로 구성되어 있을경우 기본키를 구성하는 모든 속성이 포함된 기본키의 부분집합에 종속된 경우

[기본키가 하나일 때 완전 함수 종속]

회원번호	이름	나이	거주지역
M001	이흥직	42	원주
M002	김명원	40	서울
M003	이창훈	12	부산
M004	이다은	10	경기

[기본키가 두 개일 때 완전 함수 종속]

회원번호	**주문번호**	강의명	진도율
M001	O_001	정보처리 기사	80%
M002	O_002	컴퓨터일반	60%
M003	O_003	C언어	90%
M004	O_004	파이썬	70%

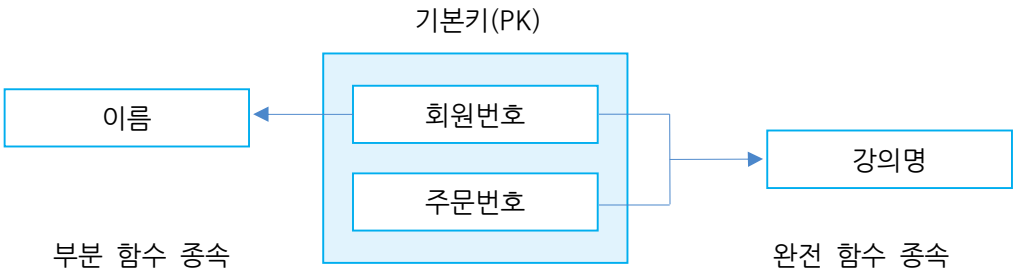
- 부분 함수적 종속 (Partial Functional Dependency)

  - 기본키가 여러 속성으로 구성되어 있을 경우 기본키를 구성하는 속성 중 일부만 종속되는 경우

  [기본키가 두개일 때 부분 함수 종속]

회원번호	**주문번호**	강의명	진도율	이름
M001	O_001	정보처리 기사	80%	이흥직
M002	O_002	컴퓨터일반	60%	김명원
M003	O_003	C언어	90%	이창훈
M004	O_004	파이썬	70%	이다은

  - 위의 릴레이션에서 강의명과 진도율은 기본키(회원번호+주문번호)에 종속적이지만, 이름은 회원번호에만 종속된다. 이때, 부분 함수적 종속이 발생한다.



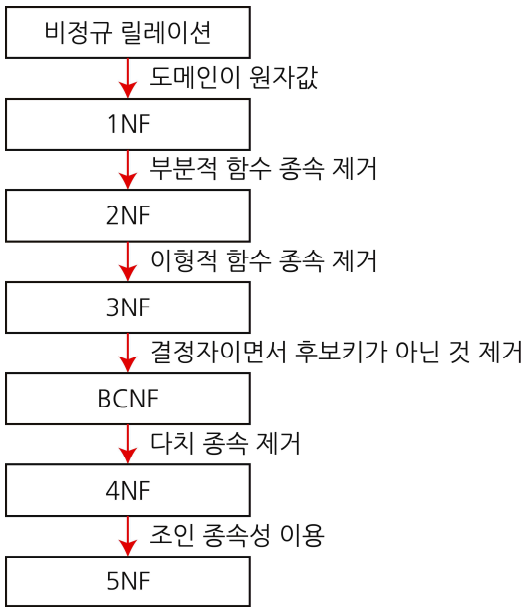
- 이행적 함수 종속 (Transitive Functional Dependecy)

  - X→Y, Y→Z 이란 종속 관계가 있을 경우, X→Z가 성립 되는 경우

회원번호	주민번호	이름
M001	800330	이흥직
M002	820320	김명원
M003	100501	이창훈
M004	120425	이다은

  - 위의 릴레이션에서 회원번호를 알면 주민번호를 알 수 있고, 주민번호를 알면 이름을 알 수 있다. 이때, 이행적 함수 종속이 발생한다.

(6) 정규화 과정



① 제 1정규형(1NF)

- 1NF 만족 조건
  - 어떤 Relation에 속한 모든 Domain이 원자값(atomic value)만으로 되어 있다.
- 조건 만족 처리
  - 고객번호와 이름을 하나의 릴레이션으로 분리하고, 여행지를 다른 릴레이션을 만들어, 속성이 원자값만으로 구성되게 한다.

[비정규 릴레이션]

고객번호	이름	여행지
M001	이흥직	서울, 원주, 수원
M002	이경직	수원
M003	김명원	원주, 제천

[1NF 만족 릴레이션]

고객번호	이름
M001	이흥직
M002	이경직
M003	김명원

고객번호	이름
M001	서울
M001	원주
M001	수원
M002	수원
M003	원주
M003	제천

② 제 2정규형(2NF)

- 2NF 만족 조건
  - 부분 함수적 종속을 모두 제거하여 완전 함수적 종속으로 만든다.
- 조건 만족 처리
  - 주문번호와 회원번호가 기본키일 때, 기본키를 가지고 주문금액을 알 수 있지만, 이름은 회원번호만 알아도 알 수 있다.
  - 주문번호와 회원번호를 모두 알아야 확인할 수 있는 주문금액을 하나의 릴레이션으로 구성하고, 회원번호와 이름을 가지고 다른 하나의 릴레이션을 구성한다.

[비정규 릴레이션]

주문번호	회원번호	주문금액	이름
O001	M001	10,000	이흥직
O002	M001	12,000	이흥직
O003	M002	8,000	이경직

[2NF 만족 릴레이션]

주문번호	회원번호	주문금액
O001	M001	10,000
O002	M001	12,000
O003	M002	8,000

회원번호	이름
M001	이흥직
M002	이경직

③ 제 3정규형(3NF)

- 3NF 만족 조건
  - 이행적 함수 종속을 없앤다.
- 조건 만족 처리

[비정규 릴레이션]

학번	주민번호	이름
O001	11111	이흥직
O002	22222	이경직
O003	33333	김명원

[3NF 만족 릴레이션]

학번	주민번호
O001	11111
O002	22222
O003	33333

주민번호	이름
11111	이흥직
22222	이경직
33333	김명원

④ 보이스/코드(BCNF) 정규형

- 결정자 중 후보키가 아닌 것들을 제거

⑤ 제 4정규형(4NF)

- 다치 종속을 제거

⑥ 제 5정규형(5NF)

- 조인 종속 제거

Section 5. 물리 데이터베이스 설계

1. 물리 데이터베이스 설계

⑴ 물리 데이터베이스 설계 과정

- 사용자 DBMS 결정
- 데이터 타입 크기 결정
- 데이터 용량, 설계 및 업무 프로세스 분석
- 역정규화(반정규화)
- 인덱스 정의
- 데이터베이스 생성

⑵ 물리 데이터베이스 설계 특징

- 논리적인 설계의 데이터 구조를 보조 기억 장치상의 파일(물리적인 데이터 모델)로 사상한다.
- 예상 빈도를 포함하여 데이터베이스 질의와 트랜잭션들을 분석한다.
- 데이터에 대한 효율적인 접근을 제공하기 위하여 저장 구조와 접근 방법을 고려한다.
- 특정 DBMS의 특성을 고려하여 진행된다.
- 질의를 효율적으로 지원하기 위해서 인덱스 구조를 적절히 사용한다.

⑶ 물리 데이터베이스 설계시 고려사항

- 반응시간
  - 응답시간을 최소화 해야 한다.
- 트랜잭션 처리량
  - 얼마나 많은 트랜잭션을 동시에 발생시킬 수 있는지 검토한다.
- 공간활용
  - 데이터가 저장될 공간을 효율적으로 배치한다.

⑷ 논리 - 물리 데이터 모델 변환

- Entity를 Table 로 변환
- 속성을 컬럼으로 변환
- Primary UID를 Primary Key 로 변환
- Secondary(Alternate) UID를 Unique Key 로 변환
- Relationship을 Foreign Key 로 변환
- Business Constraints를 Check Constraints 로 변환

⑸ 데이터베이스 암호화 방식

- API방식
  - 데이터베이스 솔루션 외부의 어플리케이션에서 데이터의 암/복화화를 수행
- Plug-in 방식
  - 데이터베이스 서버에 제품을 설치 → 암/복호화 수행
- TDE(Transparent Data Encryption)방식
  - DBMS 암호화 기능을 이용하여 데이터 파일 저장시 암호화
  - 파일에 저장된 내용을 메모리로 가져올 때 DBMS에 의해 복호화

- 파일암호화 방식
  - 데이터뿐만 아니라 비정형 데이터 암호화 적용가능
- 하드웨어방식
  - 별도의 하드웨어 장비를 외부에 설치

2. 반정규화

⑴ 반정규화의 개념

- 데이터베이스 정규화 이후, 성능향상과 개발 편의성 등 정규화 기법에 위배되는 수행 기법
- 정규화된 엔티티, 속성, 관계를 시스템의 성능 향상과 개발 운영의 단순화를 위해 중복, 통합, 분리 등을 수행하는 데이터 모델링 기법

⑵ 반정규화시 고려사항

- 데이터의 중복이 발생하여 데이터 수정시 무결성이 깨질 수 있다.
- 읽기 속도는 향상되지만, 삽입/삭제/수정 속도는 느려짐
- 저장공간의 효율이 떨어짐
- 테이블이 크고 복잡해져 유지보수가 어려움
- 과도한 반정규화는 오히려 성능을 저하시킴
- 반정규화를 위해서는 사전에 데이터의 일관성과 무결성을 우선으로 할지, 데이터베이스의 성능과 단순화를 우선으로 할지를 결정해야 함

⑶ 반정규화의 적용순서

- 반정규화의 대상을 조사한다.
  - 자주 사용하는 테이블에 접근하는 프로세스의 수가 많고 항상 일정 범위만을 조회하는 경우
  - 테이블에 대량의 데이터가 있고, 대량의 데이터 범위를 자주 처리하는 경우
  - 통계 정보를 필요로 할 때
  - 지나치게 많은 조인이 걸려 있을 때
- 다른 방법으로 유도한다.
  - 성능을 고려한 뷰를 생성하고, 뷰를 통해 접근하게 함으로 성능저하 위험 예방
  - 인덱스나 클러스터링을 통한 성능 향상
  - 파티셔닝 고려
  - 어플리케이션의 로직을 변경하여 성능 향상
- 반정규화 수행

⑷ 반정규화의 유형

구분	유형	설명
테이블 분할	수평분할	- 레코드 단위로 분할
	수직분할	- 컬럼 단위로 분할
테이블 중복	통계 테이블 추가	- DW, OLAP 데이터 용
	진행 테이블 추가	- 업무 프로세스 상태
컬럼기반 분할	조회 빈도 기반	- 고빈도 컬럼 분리
	크기 기반 분할	- 일정 용량 컬럼 분리
컬럼 중복	중복 컬럼 추가	- 자주 조회되는 컬럼 추가
	파생 컬럼 추가	- 연산 결과 별도 저장

3. 테이블 저장 사이징

⑴ 데이터베이스 용량 분석

- 정확한 데이터 용량을 산정하여 디스크 사용의 효율을 높인다.
- 업무량이 집중되어 있는 디스크를 분리, 설계하여 디스크에 대한 입출력 부하를 분산
- 데이터양이 많고 데이터의 증가량이 많을수록 디스크에 대한 입출력 분산이 철저하게 고려되어야 성능을 향상
- 여러 프로세스가 동시에 접근할 때 발생하는 디스크 입출력 경합을 최소화하여 데이터의 접근 성능을 향상

⑵ 테이블 사이징

- 초기 크기 계산
  - 테이블의 보관 주기를 확정하며 초기건수, 증가건수, 최대건수를 정의
  - 테이블의 컬럼의 Length를 모두 더한다.
  - 물리적인 공간의 확보가 가능 하다면 테이블의 초기크기는 “최대건수 * 컬럼 Lenght 총합”
- 확장 크기 계산
  - 초기 크기 작업 후 테이블의 데이터 증가에 따라 달라짐

⑶ 테이블 용량 계산

- 총 블록 헤드 계산
- 블록당 가능한 데이터 영역 계산
- 평균 Row의 전체 길이 계산
- 총 평균 Row 크기 계산
- 데이터 블록 내의 평균 Row 수 계산
- 테이블에 요구하는 블록과 바이트 수를 계산

⑷ 인덱스 용량 계산

- 총 블록 헤드 계산
- 블록당 가능한 데이터 영역 계산
- 결합된 열 길이 계산
- 인덱스 값 크기의 전체 평균 계산

4. 데이터베이스 이중화

(1) 데이터베이스 이중화 구성

- 장애발생시 데이터베이스를 보호하기 위한 방법으로 동일한 데이터베이스를 중복시켜 동시에 갱신하여 관리하는 방법
- 서버와 네트워크, 프로그램 등의 정보 시스템이 지속적으로 정상 운영이 가능한 고가용성(HA, High Availability) 서버로 구성하는 것

(2) 데이터베이스 이중화의 목적

- 장애 또는 재해시 빠른 서비스 재개를 위함
- 원활한 서비스의 성능을 보장하기 위함

(3) 데이터베이스 이중화의 분류

- Eager 기법
  - 트랜잭션 수행 중에 발생한 변경은 발생 즉시 모든 이중화서버로 전달되어 연쇄적으로 변경 내용이 반영
- Lazy 기법
  - 트랜잭션의 수행이 완전히 완료된 후에 그 변경 사실에 대한 새로운 트랜잭션을 작성하여 각 노드에게 전달하는 기법

(4) 데이터베이스 이중화의 종류

① Active-Active

- 다중화된 장비가 모두 가동되는 방식
- 두 대를 모두 사용하기 때문에 처리율이 높지만, 구성이 복잡함
- 사용자 세션관리와 부하에 대한 분산처리에 대해 고려

② Active-Standby

- 두 대 중 하나는 가동이 되고, 하나는 장애 상황의 경우를 대비해서 준비 상태로 대기
- 장애가 발생하여 Active장비가 죽게되면 Standby장비가 Active상태가 되어 서비스에 문제가 없도록 처리
- Active-Standby 타입

Hot Standby	- Standby 장비가 가동되었을 때 즉시 사용가능
Warm Standy	- Standby 장비가 가동되었을 때 설정에 대한 준비가 필요함
Cold Standby	- Standby 장비를 평소에는 정지시켜두며 필요에 따라서 직접 켜서 구성을 함

5. 데이터베이스 백업

(1) 데이터베이스 백업 개념

- 정전, 사이버 공격 및 다른 중단 사태에 대비하여 복구를 진행할 수 있도록 데이터를 주기적으로 복사하는 것을 의미
- 백업 : 자료들(datas)을 복사, 보관
- 복원 : 손상된 데이터베이스를 복구

(2) 백업 방식

- 전체 백업(Full Backup)
  - 선택된 폴더의 DATA를 모두 백업하는 방식
- 증분 백업(Incremental Backup)
  - Full 백업 이후 변경/추가된 Data만 백업하는 방식
- 차등 백업(Differential Backup)
  - Full 백업 이후 변경/추가된 Data를 모두 포함하여 백업
- 실시간 백업(RealTime Backup)
  - 즉각적으로 모든 변경사항을 분리된 스토리지 디바이스에 복사
- 트랜잭션 로그 백업(Transaction log Backup)
  - 데이터베이스에서 실행되는 모든 SQL문을 기록한 로그
  - REDO(다시 실행), UNDO(원상태로 복구) 로 복원
  - CHECK POINT : 설정한 지점 이전 까지는 트랜잭션이 성공적으로 수행이 돼서 disk에 확실히 저장된 상태

(3) 복구시간목표/복구시점 목표

- 복구 시간 목표(RTO)
  - 서비스 중단 시점과 서비스 복원 시점 간에 허용되는 최대 지연 시간
  - 서비스를 사용할 수 없는 상태로 허용되는 기간
- 복구 시점 목표(RPO)
  - 마지막 데이터 복구 시점 이후 허용되는 최대 시간
  - 마지막 복구 시점과 서비스 중단 시점 사이에 허용되는 데이터 손실량

Section 6. 데이터베이스 물리속성 설계

1. 파티셔닝

(1) 파티셔닝 개념

- 데이터베이스를 여러 부분으로 분할하는 것
- 데이터가 너무 커져서, 조회하는 시간이 길어질 때 또는 관리 용이성, 성능, 가용성 등의 향상을 이유로 분할

(2) 파티셔닝의 장점

- 가용성(Availability)
  - 물리적인 Partitioning으로 인해 전체 데이터의 훼손 가능성이 줄어들고 데이터 가용성이 향상
- 관리용이성(Manageability)
  - 각 분할 영역(partition별로)을 독립적으로 백업하고 복구
- 성능(Performance)
  - 특정 DML과 Query의 성능을 향상

(3) 파티셔닝의 단점

- Table간의 Join에 대한 비용이 증가
- Table과 Index를 별도로 파티션 할수는 없다

(4) 파티셔닝의 종류

- 수평 분할(horizontal partitioning)
  - 하나의 테이블의 각 행들을 분할
  - 스키마를 복제한 후 샤드키를 기준으로 데이터를 나눈다.
- 수직 분할(vertical partitioning)
  - 테이블의 일부를 컬럼을 기준으로 분할
  - 자주 사용하는 컬럼 등을 분리시켜 성능을 향상
  - 하나의 테이블을 2개 이상으로 분리하는 작업

(5) 분할 기준

① 범위 분할 (range partitioning)

- Partition Key 의 연속된 범위로 파티션을 정의
- 파티션 키 위주로 검색이 자주 실행될 경우 유용
- 예) 월별, 분기별 등

② 목록 분할 (list partitioning)

- 특정 Partition 에 저장 될 Data 에 대한 명시적 제어
- 많은 SQL 에서 해당 Column 의 조건이 많이 들어오는 경우 유용
- ex) [한국, 일본, 중국 → 아시아] [노르웨이, 스웨덴, 핀란드 → 북유럽]

③ 해시 분할 (hash partitioning)

- 파티션 키 값에 해시 함수를 적용하고, 거기서 반환된 값으로 파티션 매핑
- 데이터가 모든 파티션에 고르게 분산되도록 DBMS가 관리
- 병렬처리 시 성능효과 극대화

④ 라운드 로빈 분할 (round robin partitioning)

- Data를 균일하게 분배해서 저장하는 방식

⑤ 합성 분할 (composite partitioning)

- 위의 기술들을 복합적으로 사용하는 방법
- ex) 범위 분할 후 분할 된 데이터를 해시 분할하는 등

(6) 파티션 생성

```
-- 범위 분할 파티션
CREATE TABLE TB_USER (
    id INT,
    year INT
)
PARTITION BY RANGE (year) (
    PARTITION U1 VALUES LESS THAN (2000),
    PARTITION U2 VALUES LESS THAN (2010),
    PARTITION U3 VALUES LESS THAN (2020)
);

-- 목록 분할 파티션
CREATE TABLE TB_STUDENT (
    id INT,
    grade INT
)
PARTITION BY LIST (grade) (
    PARTITION high_grade VALUES IN (1, 2, 3),
    PARTITION low_grade VALUES IN (4, 5, 6)
);
```

2. 클러스터 설계

(1) 클러스터의 개념

- 디스크로부터 데이터를 읽어오는 시간을 줄이기 위해서 조인이나 자주 사용되는 테이블의 데이터를 디스크의 같은 위치에 저장시키는 방법
- 데이터 저장시 데이터 액세스 효율을 향상시키기 위해 동일한 성격의 데이터를 동일한 데이터 블록에 저장하는 물리적 저장 방법

(2) 클러스터의 특징

- 그룹된 데이터들이 같은 데이터 Block 에 저장되기 때문에 디스크 I/O를 줄여준다
- 클러스터된 테이블 사이에 조인이 발생할 경우 처리 시간이 단축
- 클러스터는 데이터 조회 성능을 향상 시키지만 데이터 저장, 수정, 삭제나 Full Scan 시 성능 저하

- 클러스터는 데이터의 분포도가 넓을수록 유리함
- 파티셔닝된 테이블에는 클러스터링 불가
- 클러스터링된 테이블에 클러스터드 인덱스를 생성하면 접근 성능 향상

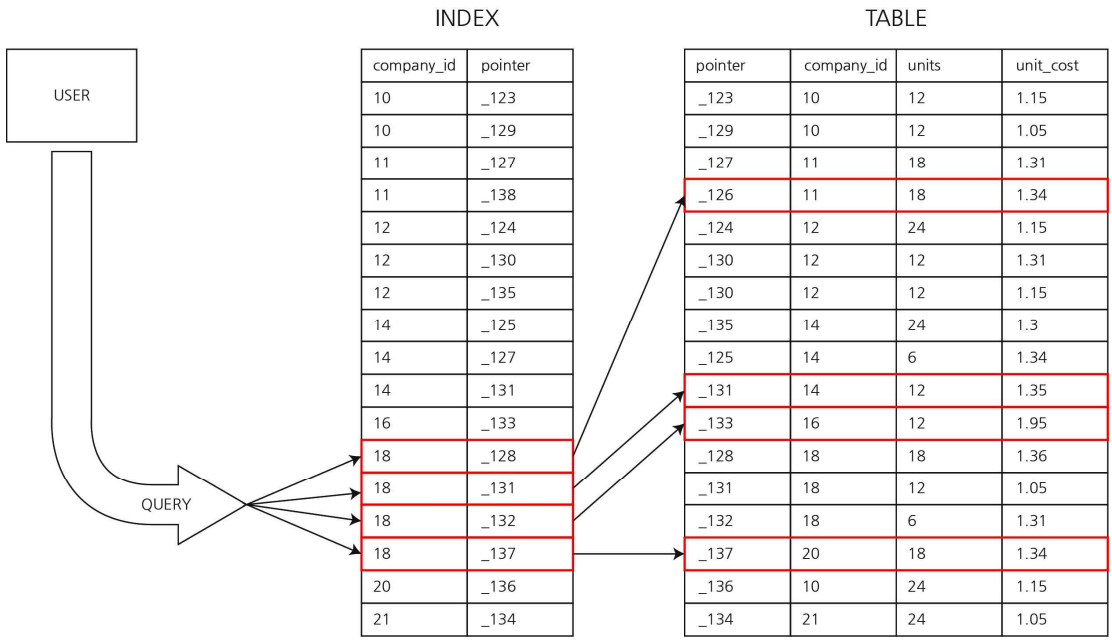
(3) 클러스터 대상 테이블

- 분포도가 넓은 테이블
- 대량의 범위를 자주 조회하는 테이블
- 입력, 수정, 삭제가 자주 발생하지 않는 테이블
- 자주 JOIN 되어 사용되는 테이블
- ORDER BY, GROUP BY, UNION 이 빈번한 테이블

3. 인덱스(Index)

(1) 인덱스의 개념

- 추가적인 저장 공간을 활용하여 데이터베이스 테이블의 검색 속도를 향상시키기 위한 자료구조
- 책의 맨 앞 또는 맨 뒤에 색인을 추가하는데, 데이터베이스의 index는 책의 색인과 같다.
- 테이블의 모든 데이터를 검색하면 시간이 오래 걸리기 때문에 데이터와 데이터의 위치를 포함한 자료구조를 생성하여 빠르게 조회



(2) 인덱스를 사용하는 이유

- 규모가 작지 않은 테이블에서 데이터를 빠르게 추출
- 조건 검색 절의 효율성
- 정렬 절의 효율성
- MIN, MAX의 효율적인 처리
- JOIN을 실행할 때 따른 테이블에서 열을 추출

⑶ 인덱스의 종류

- 클러스터 인덱스
  - 테이블당 1개만 허용되며, 해당 컬럼을 기준으로 테이블이 물리적으로 정렬
  - 데이터는 기본적으로 오름차순으로 정렬을 진행
  - 기본 키를 설정하면 자동으로 클러스터드 인덱스가 적용
  - 인덱스 자체의 리프 페이지가 곧 데이터
  - 데이터 입력, 수정, 삭제 시 항상 정렬 상태를 유지
  - 비 클러스형 인덱스보다 검색 속도는 빠르나, 데이터의 입력, 수정, 삭제시에는 느림
- 논클러스터 인덱스
  - 테이블당 약 240개의 인덱스 생성 가능
  - 레코드의 원본은 정렬되지 않고, 인덱스 페이지만 정렬
  - 인덱스 자체의 리프 페이지는 데이터가 아니라 데이터가 위치하는 포인터(RID)이기 때문에 클러스터형보다 검색 속도는 더 느리지만 데이터의 입력, 수정, 삭제는 더 빠름
  - 인덱스를 생성할 때 데이터 페이지는 그냥 둔 상태에서 별도의 인덱스 페이지를 따로 만들기 때문에 용량을 더 차지함
- 밀집 인덱스
  - 데이터 레코드 각각에 대해 하나의 인덱스가 만들어짐
- 희소 인덱스
  - 레코드 그룹 또는 데이터 블록에 대해 하나의 인덱스가 만들어짐

⑷ 인덱스의 구조

- 트리 기반 인덱스
  - 대부분의 상용 DBMS에서는 트리 구조를 기반으로 하는 B+ 트리 인덱스를 주로 사용
  - B+ 트리는 루트에서 리프노드까지 모든 경로의 깊이가 같은 밸런스 트리 형태
  - 다른 인덱스에 비해 대용량 처리의 데이터 삽입과 삭제에 좋은 성능 유지
- 비트맵 인덱스
  - 비트를 이용하여 컬럼값을 저장하고 이를 이용하여 주소를 자동으로 생성하는 인덱스
  - 주어진 키 값을 포함하고 있는 로우에 대한 주소 정보를 제공하는 것
- 함수 기반 인덱스
  - 함수(function)나 수식(expression)으 로 계산된 결과에 대해 B+ 트리 인덱스나 Bit-Map Index를 생성, 사용할 수 있는 기능을 제공
  - 사용되는 함수는 산술식(Arithmetic expression), PL/SQL Function, SQL Function, Package, C callout
- 비트맵 조인 인덱스
  - 비트맵 조인 인덱스의 물리적인 구조는 비트맵 인덱스와 완전히 동일
  - 인덱스의 구성 칼 럼이 베이스 테이블의 칼럼 값이 아니라 조인된 테이블의 칼럼 값이라는 차이
- 도메인 인덱스
  - 개발자가 자신이 원하는 인덱스 타입을 생성할 수 있게 함으로써 인덱스 시스템에 많은 확장 가능

(5) 인덱스 컬럼의 선정

- 분포도가 좋은(10%~15%) 컬럼
- 자주 조합되어 사용되는 경우 결합 인덱스를 생성
- 가능한 수정이 빈번하지 않은 컬럼을 선정
- 한 컬럼이 여러 인덱스 포함되지 않도록 설계
- 기본키 및 외부키가 되는 컬럼을 선정

(6) 인덱스 생성시 고려사항

- 새로 추가된 인덱스는 기존 엑세스 경로에 영향을 미칠 수 있다.
- 지나치게 많은 인덱스는 오버헤드를 발생시킨다.
- 넓은 범위를 인덱스로 처리시 많은 오버헤드를 발생시킨다.
- 옵티마이저를 위한 통계 데이터를 주기적으로 갱신한다.
- 인덱스를 위한 추가적인 저장공간이 필요

(7) 인덱스 생성/삭제

① 인덱스 생성

```
--문법
CREATE INDEX [인덱스명] ON [테이블명](컬럼1, 컬럼2, 컬럼3.....)

--예제
CREATE INDEX USER_INDEX ON TB_USER(NAME,HP);
```

② 인덱스 삭제

```
--문법
DROP INDEX 인덱스 명;

--예제
DROP INDEX USER_INDEX;
```

4. 뷰(View)

⑴ 뷰의 개념

- 하나 이상의 기본 테이블로부터 유도된, 이름을 가지는 가상 테이블
- 뷰는 실제로 데이터를 저장하고 있지 않으며, 논리적으로만 존재
- 데이터베이스 사용자는 실제로 데이터가 존재하는 테이블과 동일하게 뷰를 조작할 수 있다.

⑵ 뷰의 특징

- 뷰는 기본테이블로부터 유도된 테이블이기 때문에 기본 테이블과 같은 형태의 구조를 사용하며, 조작도 기본 테이블과 거의 같다.
- 뷰는 가상 테이블이기 때문에 물리적으로 구현되어 있지 않다.
- 데이터의 논리적 독립성을 제공할 수 있다.
- 필요한 데이터만 뷰로 정의해서 처리할 수 있기 때문에 관리가 용이하고 명령문이 간단해진다.
- 뷰를 통해서만 데이터에 접근하게 하면 뷰에 나타나지 않는 데이터를 안전하게 보호한다.
- 데이터의 삽입, 삭제, 갱신에는 제한이 있다.
- 뷰가 정의된 기본 테이블이나 뷰를 삭제하면 그 테이블이나 뷰를 기초로 정의된 다른 뷰도 자동으로 삭제된다.
- CREATE 로 생성하고, DROP 으로 삭제한다. (ALTER로 변경이 불가능하다)

⑶ 뷰의 장/단점

- 장점
  - 논리적 데이터 독립성을 제공한다.
  - 동일 데이터에 대해 동시에 여러 사용자의 상이한 응용이나 요구를 지원해 준다.
  - 사용자의 데이터관리를 간단하게 해준다.
  - 접근 제어를 통한 자동 보안이 제공된다.
- 단점
  - 독립적인 인덱스를 가질 수 없다.
  - ALTER VIEW문을 사용할 수 없다.
  - 뷰로 구성된 내용에 대한 삽입, 삭제, 갱신, 연산에 제약이 따른다.

⑷ 뷰 생성/삭제

① 뷰 생성

```
--문법
CREATE VIEW [view_name]AS
SELECT [field_name_1], [field_name_2]
FROM [table_name]
WHERE [조건];

--예제
CREATE OR REPLACE VIEW USER_VIEW AS
SELECT
    id, user
FROM TB_USER
WHERE name like '이%'
;
```

② 뷰 삭제

```
--문법
DROP VIEW 뷰 명;

--예제
DROP VIEW USER_VIEW;
```

5. 시스템 카탈로그

(1) 시스템 카탈로그

- 데이터베이스 관리자의 도구로, 데이터베이스에 저장되어 있는 모든 데이터 개체들에 대한 정의나 명세에 대한 정보가 수록되어 있는 시스템 테이블
- 시스템 카탈로그를 데이터 사전(Data Dictionary) 라고도 한다.
- 시스템 카탈로그에는 DDL의 결과로 구성되는 기본 릴레이션, 인덱스, 뷰, 사용자, 접근 권한 등의 데이터베이스 구조 및 통계 정보가 저장
- 시스템 카탈로그에 저장된 내용을 메타데이터라고 한다.
- 시스템 카탈로그는 사용자와 데이터베이스 관리 시스템의 접근이 가능하다.

(2) 시스템 카탈로그의 내용

- 릴레이션 관련 정보
  - 릴레이션의 이름
  - 릴레이션이 저장된 파일 이름과 파일 구조
  - 릴레이션의 속성들에 대한 속성 이름과 타입(또는 도메인)
  - 릴레이션에 대해 정의된 각 인덱스의 이름
  - 릴레이션에 대해 정의된 무결성 제약 조건
- 인덱스 관련 정보
  - 인덱스의 이름
  - 인덱스의 구조
  - 인덱스 키에 대한 정보
- 뷰 관련 정보
  - 뷰의 이름
  - 뷰의 정의
  - 뷰의 소유자
- 통계 관련 정보
  - 릴레이션 카디널리티(Cardinality) : 각 릴레이션에 저장된 레코드 수
  - 인덱스 카디널리티 : 각 인덱스에 저장된 레코드의 수
  - 인덱스의 높이 : 각 트리 인덱스에 대한 레벨
  - 인덱스의 범위 : 각 인덱스에 대한 최소 키 값과 최대 키 값
- 사용자 관련 정보
  - 사용자의 계정 정보
  - 사용자의 권한 정보

(3) 시스템 카탈로그의 특징

- 시스템 카탈로그 자체도 시스템 테이블로 구성되어 있어 사용자가 SQL문을 이용하여 내용을 검색해 볼 수 있다.
- 시스템 카탈로그는 데이터베이스 관리 시스템에 의해 생성되고 유지된다.
- 사용자가 시스템 카탈로그를 직접 갱신하는 것은 허용되지 않는다.

⑷ 시스템 카탈로그의 종류

- SYSTABLES
  - 기본 테이블 및 뷰 테이블의 정보를 저장하는 테이블
- SYSCOLUMNS
  - 모든 테이블에 대한 정보를 열(속성) 중심으로 저장하는 테이블
- SYSVIEW
  - 뷰에 대한 정보를 저장하는 테이블
- SYSTABAUTH
  - 테이블에 설정된 권한 사항들을 저장하는 테이블
- SYSCOLAUTH
  - 각 속성에 설정된 권한 사항들을 저장하는 테이블

Section 7. 관계 데이터베이스 모델

1. 관계 데이터 모델

(1) 관계 데이터 모델 개념

- 데이터의 논리적 구조가 릴레이션, 즉 테이블 형태의 평면 파일로 표현되는 데이터 모델
- 테이블 형식을 이용하여 데이터를 정의하고 설명한 모델

(2) 관계 데이터 릴레이션의 구조

〈학생 릴레이션〉

- 속성
  - 릴레이션의 각 열을 속성 또는 Attribute라고 한다.
  - 데이터를 구성하는 가장 작은 논리적인 단위
  - 개체의 특성을 기술
  - 속성의 수 = 디그리(Degree) = 차수
- 튜플(Tuple)
  - 릴레이션의 행을 Tuple 튜플이라고 한다.
  - 속성들의 모임으로 구성된다.
  - 튜플의 수 = 카디널리티(Cardinality) = 기수
- 도메인
  - 하나의 속성이 가질 수 있는 값의 범위
  - 성별은 남, 여를 가질 수 있고, 학년은 1~4를 가질 수 있다.
  - 속성 값에 도메인을 정해두면, 정해진 값 외에 다른 값이 삽입 될 수 없으므로 무결성이 유지됨
- 차수 (Degree)
  - 하나의 릴레이션에서 속성의 전체 개수
- 카디널리티 (Cardinality)
  - 하나의 릴레이션에서 튜플의 전체 개수

(3) 릴레이션

- 데이터들을 2차원 테이블의 구조로 저장한 것
- 릴레이션의 구성
  - 릴레이션 스키마 : 릴레이션 이름과 릴레이션에 포함된 모든 속성의 이름으로 정의하는 릴레이션의 논리적인 구조
  - 릴레이션 인스턴스 : 릴레이션 스키마에 실제로 저장된 데이터의 집합

- 릴레이션의 특징
  - 튜플의 유일성 : 릴레이션 안에는 똑같은 튜플이 존재할 수 없음
  - 튜플의 무순서성 : 튜플 사이에는 순서가 없음
  - 속성의 무순서성 : 속성 사이에는 순서가 없음
  - 속성의 원자성 : 속성은 더 이상 분해할 수 없는 원자값만 가진다.
  - 튜플들의 삽입, 갱신, 삭제작업이 실시간으로 일어나므로 릴레이션은 수시로 변한다.

2. 관계데이터 언어(관계대수, 관계해석)

(1) 관계 대수의 개념

- 원하는 데이터를 얻기 위해, 데이터를 어떻게 찾는지에 대한 처리 과정을 명시하는 절차적인 언어
- 질의에 대한 해를 구하기 위해 수행해야 할 연산의 순서를 명시한다.
- 릴레이션 조작을 위한 연산의 집합으로 피연산자와 결과가 모두 릴레이션이다.
- 일반 집합 연산과 순수 관계 연산으로 구분
- 기본적으로 관계해석과 관계대수는 관계 데이터베이스를 처리하는 기능과 능력 면에서 동일

(2) 순수 관계 연산자

<<학생>>

학번	이름	학년	학과	성적
1001	김철수	2	컴퓨터	95
1002	김영희	3	컴퓨터	90
1003	이흥직	1	환경	75
1004	홍길동	2	건축	80
1005	김명민	1	건축	85
1006	소찬휘	3	전자	100

<<수강과목>>

학번	수강과목	학점
1001	국어	3
1001	영어	2
1002	국어	3
1002	수학	5
1003	국어	3
1005	국어	3

① SELECT

- 릴레이션에서 주어진 조건을 만족하는 튜플을 선택하는 연산자
- 기호 : σ(시그마)
- 표기법 : σ<조건>(R)
- 조건에서는 =, ≠, <, ≤, >, ≥ 등의 기호를 사용한 비교 연산이 허용,
  AND(∧), OR(∨), NOT(¬) 등의 논리 연산자를 사용
- 예시1 : σ 성적 > 90 (학생)

학번	이름	학년	학과	성적
1001	김철수	2	컴퓨터	95
1006	소찬휘	3	전자	100

- 예시2 : σ 성적 >= 90 ^ 학과 = '컴퓨터' (학생)

학번	이름	학년	학과	성적
1001	김철수	2	컴퓨터	95
1002	김영희	3	컴퓨터	90

② PROJECT

- 주어진 릴레이션에서 속성 리스트에 제시된 속성 값만을 추출하는 연산자
- 기호 : π(파이)
- 표기법 : π<리스트>(R)
- 예시1 : π 학번, 성적 (학생)

학번	성적
1001	95
1002	90
1003	75
1004	80
1005	85
1006	100

- 예시2 : π 학번, 이름, 성적 (σ 성적 >= 90 (학생))

학번	이름	성적
1001	김철수	95
1002	김영희	90
1006	소찬휘	100

③ JOIN

- join두 개의 릴레이션으로부터 연관된 튜플들을 결합하는 연산자
- 기호 : □(보타이)
- 표기법 : R□<조건>S
- 예시1 : (학생) □ 학번=학번 (수강과목)

학번	이름	학년	학과	성적	수강과목	학점
1001	김철수	2	컴퓨터	95	국어	3
1001	김철수	2	컴퓨터	95	영어	2
1002	김영희	3	컴퓨터	90	국어	3
1002	김영희	3	컴퓨터	90	수학	5
1003	이흥직	1	환경	75	국어	3
1005	김명민	1	건축	85	국어	3

④ DIVISION

- 릴레이션 S의 모든 튜플과 관련이 있는 릴레이션 R의 튜플들을 반환
- 기호 : ÷(나누기)
- 표기법 : R÷S

<<R>>

A	B
a1	b1
a1	b2
a1	b3
a2	b1
a2	b3

<<S1>>

B
b1

<<S2>>

B
b1
b2

- 예시1 : (R) ÷ (S1)

A
a1
a2

- 예시2 : (R) ÷ (S2)

A
a1

(3) 일반 집합 연산자

<<A과목>>

학번	이름
1001	김철수
1002	김영희
1003	이흥직

<<B과목>>

학번	이름
1001	김철수
1004	김명민
1005	소찬휘

① 합집합(Union)

- 두 릴레이션에 존재하는 튜플의 합집합을 구하되, 결과로 생성된 릴레이션에서 중복되는 튜플은 제거
- 표기법 : ∪
- 예시1 : A과목 ∪ B과목

학번	이름
1001	김철수
1002	김영희
1003	이흥직
1004	김명민
1005	소찬휘

② 교집합 (Intersection)

- 두 릴레이션에 존재하는 튜플의 교집합을 구하는 연산
- 표기법 : ∩
- 예시1 : A과목 ∩ B과목

학번	이름
1001	김철수

③ 차집합 (Difference)

- 두 릴레이션에 존재하는 튜플의 차집합을 구하는 연산
- 표기법 : -
- 예시1 : A과목 - B과목

학번	이름
1002	김영희
1003	이흥직

④ 교차곱 (Cartesian Product)

- 두 릴레이션에 있는 튜플들의 순서쌍을 구하는 연산
- 표기법 : X

<<학생>>

학번	이름
1001	김철수
1002	김영희

<<과목>>

번호	과목
S01	국어
S02	영어

- 예시1 : 학생 X 과목

학생.학번	학생.이름	과목.번호	과목.과목
1001	김철수	S01	국어
1001	김철수	S02	영어
1002	김영희	S01	국어
1002	김영희	S02	영어

(4) 관계해석

- 관계 데이터 모델의 제안자인 코드(E. F. Codd)가 수학의 Predicate Calculus(술어 해석)에 기반을 두고 관계 데이터베이스를 위해 제안
- 관계해석은 관계 데이터의 연산을 표현하는 방법으로, 원하는 정보를 정의할 때 계산 수식을 사용
- 관계해석은 원하는 정보가 무엇이라는 것만 정의하는 비절차적 특성
- 튜플 관계해석과 도메인 관계해석이 있다.
- 질의어로 표현한다.
- 관계해석과 관계대수는 관계 데이터베이스를 처리하는 기능과 능력 면에서 동등하다.
- 연산자

구분	기호	설명
연산자	∨	- OR 연산
	∧	- AND 연산
	¬	- NOT 연산
정량자	∀	- 모든 가능한 튜플 “For All”
	∃	- 어떤 튜플 하나라도 존재

Section 8. 키와 무결성 제약조건

1. 컬럼

(1) 컬럼의 개념

- 릴레이션에서 정보를 나타내는 최소 단위로, 각 열의 상태나 특성을 나타내는 항목을 말한다.
- 릴레이션에에서 특정한 단순 자료형의 일련의 데이터값과 릴레이션에서의 각 열을 말한다.
- 컬럼은 열이 어떻게 구성되어야 할 지에 대한 구조를 제공한다.
- 관계형 데이터베이스 용어에서 컬럼과 같은 의미로 사용되는 것은 속성(attribute)이다.

(2) 속성의 특징

- 시스템을 구축하려는 업무 프로세스에 필요한 정보로 구성되어야 한다.
- 하나의 속성은 하나의 속성값만을 가진다. (여러 개의 속성값이 들어가면 별도의 테이블로 분리)

(3) 속성의 분류

- 기본속성
  - 업무로부터 추출한 모든 속성
- 설계속성
  - 코드성 데이터, 릴레이션 식별용 일련번호
- 파생속성
  - 다른 속성에 영향을 받아 발생하는 속성
  - 계산값, 합계, 재고 등

(4) 세부 의미에 따른 분류

- 단순 속성(Simple Attribute)
  - 나이, 성별같이 다른 속성들로 구성될 수 없는 단순한 속성
- 복합 속성(Composite Attribute)
  - 주소와 같이 시, 구, 동처럼 여러 세부 속성들로 구성될 수 있는 속성

(5) 구성방식 따른 분류

- PK(Primary Key) 속성
  - 릴레이션에서 튜플을 유일하게 구분할 수 있는 속성
- FK(Foreign Key) 속성
  - 다른 릴레이션과의 관계에서 참조하고 있는 속성
- 일반 속성
  - 릴레이션에 포함된 속성 중, PK와 FK가 아닌 속성

(6) 도메인

- 속성이 가질 수 있는 값의 범위

(7) 속성명 부여 원칙

- 해당 업무에서 사용하는 이름을 부여한다.
- 서술식 속성명은 사용하지 않는다.
- 약어사용은 가급적 제한한다.
- 전체 데이터 모델에서 유일성을 확보하는게 좋다.

2. 키 종류

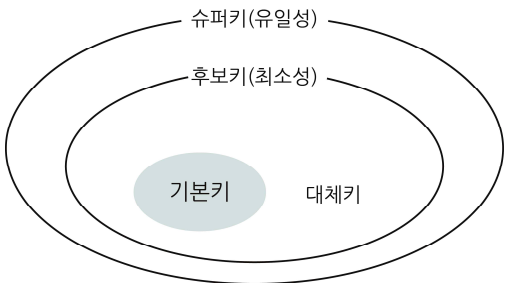
(1) 키(Key)의 개념

- 데이터베이스에서 조건에 만족하는 튜플을 찾거나 순서대로 정렬할 때 다른 튜플들과 구별할 수 있는 유일한 기준이 되는 컬럼
- 아래 학생 릴레이션에서 학생 튜플을 구별할 수 있는 키는 학번 컬럼이다.

<<학생>>

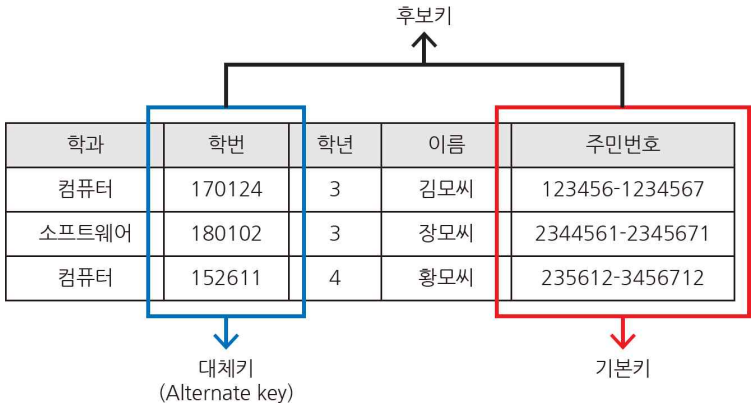
학번	주민번호	이름	학년	학과	성적
1001	111	김철수	2	컴퓨터	95
1002	222	김영희	3	컴퓨터	90
1003	333	이흥직	1	환경	75

(2) 키(Key)의 종류



① 후보키 (Candidate Key)

- 릴레이션을 구성하는 속성들 중에서 튜플을 유일하게 식별할 수 있는 속성들의 부분집합
- 모든 릴레이션은 반드시 하나 이상의 후보키를 가져야 한다.
- 튜플에 대한 유일성과 최소성을 만족시켜야 한다.



② 기본키 (Primary Key)

- 후보키 중에서 선택한 주키(Main Key)
- 한 릴레이션에서 특정 튜플을 유일하게 구별할 수 있는 속성
- Null 값을 가질 수 없다. (개체 무결성)
- 기본키로 정의된 속성에는 동일한 값이 중복되어 저장될 수 없다. (개체 무결성)

③ 대체키 (Alternate Key)

- 후보키가 둘 이상일 때 기본키를 제외한 나머지 후보키
- 보조키라고도 한다.

④ 슈퍼키 (Super Key)

- 한 릴레이션 내에 있는 속성들의 집합으로 구성된 키
- 튜플에 대한 유일성은 만족하지만, 최소성은 만족시키지 못한다.

⑤ 외래키 (Foreign Key)

- 관계(Relation)를 맺고 있는 릴레이션 R1, R2에서 릴레이션 R1이 참조하고 있는 릴레이션 R2의 기본키와 같은 R1 릴레이션의 속성
- 참조되는 릴레이션의 기본키와 대응되어 릴레이션 간에 참조 관계를 표현하는데 중요한 도구로 사용
- 외래키로 지정되면 참조 테이블의 기본키에 없는 값은 입력할 수 없다. (참조 무결성 조건)

〈학생 Table〉

학번	주민등록번호	성명
A11	111-123	홍길동
A12	112-789	김철수
A13	119-753	박영희
A14	115-951	홍길동

〈수강 Table〉

학번	과목명
A11	영어
A12	수학
A13	영어
A14	수학

3. 데이터베이스 무결성

(1) 데이터베이스 무결성 개념

- 데이터의 정확성, 일관성, 유효성이 유지되는 것
- 데이터의 무결성을 유지하는 것은 데이터베이스 관리시스템 (DBMS)의 중요한 기능
- 데이터에 적용되는 연산에 제한을 두어 데이터의 무결성을 유지

(2) 데이터베이스 무결성 종류

① 개체 무결성 (Entity integrity)

- 모든 릴레이션은 기본 키(primary key)를 가져야 한다
- 기본키는 중복되지 않은 고유한 값을 가져야한다.
- 릴레이션의 기본키는 NULL 값을 허용하지 않는다.

② 참조 무결성 (Referential integrity)

- 외래키 값은 NULL이거나 참조하는 릴레이션의 기본키 값과 동일해야 한다.
- 각 릴레이션은 참조할 수 없는 외래키 값을 가질 수 없다.
- 참조 무결성 제약조건

종류	설명
제한(Restricted)	- 문제가 되는 연산을 거절
연쇄(Cascade)	- 참조되는 릴레이션에서 튜플을 삭제하고 참조하는 릴레이션에서 이 튜플을 참조하는 튜플도 함께 삭제
널값(Nullify)	- 참조되는 릴레이션에서 튜플을 삭제하고 참조하는 릴레이션에서 이 튜플을 참조하는 튜플들의 외래키에 NULL 등록
기본값(Default)	- Null을 넣는 대신에 디폴트 값을 등록

③ 도메인 무결성 (Domain integrity)

- 속성들의 값은 정의된 도메인에 속한 값이어야 한다.
- 성별이라는 컬럼에는 '남', '여'를 제외한 데이터는 제한되어야 한다.

④ 고유 무결성(Unique integrity)

- 릴레이션의 특정 속성에 대해 각 튜플이 갖는 속성 값들이 서로 달라야 한다.

⑤ 키 무결성(Key integrity)

- 하나의 릴레이션에는 적어도 하나의 키가 존재해야 한다.

⑥ 릴레이션 무결성(Relation Integrity)

- 릴레이 션을 조작하는 과정에서의 의미적 관계(Semantic Relationship)을 명세한다.

Section 9. 물리데이터 모델 품질 검토

1. CRUD 분석

(1) CRUD의 개념

- 데이터 처리 기능인 Create(생성), Read(읽기), Update(갱신), Delete(삭제)를 묶어서 표현한 말이다.
- 시스템 구축 시 프로세스와 DB에 저장되는 데이터 사이의 의존관계를 표현하는 표
- 시스템을 구축하기 위해 해당 업무에 어떤 데이터가 존재하는지 무엇이 영향을 받는지 분석
- 데이터베이스에 영향을 주는 생성, 읽기, 갱신, 삭제 연산으로 프로세스와 테이블 간에 매트릭스를 만들어서 트랜잭션을 분석

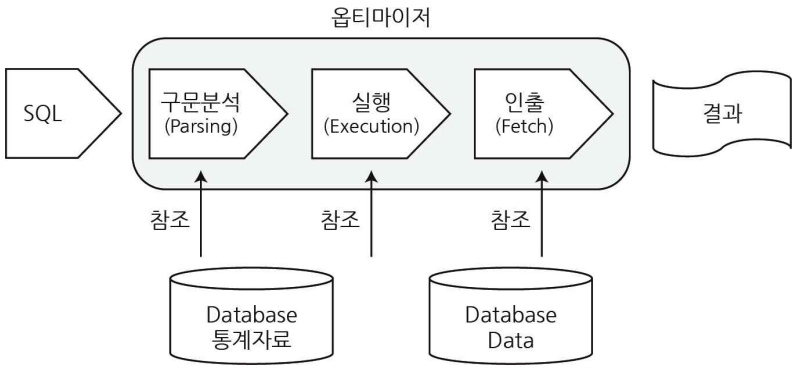
테이블 / 업무	회원	상품	주문	재고
회원가입	C			
로그인	RU			
상품보기		RU		
상품주문		R	C	U
주문취소		R	D	U

(2) CRUD의 필요성

- 모델링 작업검증
  - 분석 단계의 데이터 모델과 업무 프로세스 모델에 대한 작업 검증
- 중요 산출물
  - 시스템 구축 단계에서 필요하며, 산출물의 역할을 함
- 테스트 시 사용
  - 테스트 자료로 사용되며, 각 테스트 단위별 데이터의 변화를 확인
- 인터페이스 현황 파악
  - 전체 업무의 인터페이스 파악

2. 옵티마이저

⑴ SQL 처리 흐름



① 구문분석 단계

- 사용자가 요청한 SQL문이 데이터베이스에서 사용된 적이 있는지 공유 풀 영역을 검색하여 확인
- 이미 사용했다면 구문분석 작업을 하지 않고, 처음 사용되었다면 구문분석 작업을 수행
- SQL 문이 문법에 따라 정상적으로 작성되었는지 분석하고, SQL내에 포함된 테이블, 뷰 등이 데이터베이스에 존재하는지 확인

② 실행 단계

- SQL 문에서 사용된 데이터가 버퍼캐시 영역에 존재하는지 검색
- 데이터버퍼 캐시영역에 존재한다면, 테이블의 해당 데이터 파일로부터 테이블을 읽지 않고 캐시영역의 데이터를 그대로 추출
- 존재하지 않는다면 정의된 테이블의 해당 데이터 파일로부터 테이블을 읽어서 데이터버퍼 캐시영역에 저장

③ 추출 단계

- 실행단계가 끝나면 서버 프로세스는 데이터버퍼 캐시영역에서 관련 테이블 데이터를 읽어서 사용자가 요청한 클라이언트로 전송
- SELECT문을 실행하는 경우에만 추출단계가 실행되고, UPDATE, INSERT, DELETE문 실행 시는 추출단계는 실행되지 않는다.

⑵ 옵티마이저 개념

- 사용자가 질의한 SQL 문에 대해 최적의 실행 방법을 결정하는 역할을 수행
- 실행계획을 결정한다.
- 옵티마이저의 구분

㉠ 규칙기반 옵티마이저 (Rule Based Optimizer)
- 규칙(우선순위)를 가지고 실행 계획을 생성
- 인덱스 유무, 연산자, 객체 등을 참조하여 우선순위를 부여

㉡ 비용기반 옵티마이저 (Cost Based Optimizer)
- SQL문을 처리하는데 필요한 비용이 가장 적은 실행계획을 선택하는 방식
- 소요시간 이나 자원 사용량을 가지고 실행계획 생성
- 테이블, 인덱스, 컬럼 등의 다양한 객체 통계정보와 시스템 통계정보 활용

3. SQL 성능 튜닝

⑴ 튜닝의 개념

- SQL 문을 최적화하여 빠른 시간내에 원하는 결과값을 얻기 위한 작업
- 주어진 H/W 환경을 통해 처리량과 응답속도를 개선하기 위해 수행
- 쿼리문의 응답시간, 해당 쿼리가 실행될 때 사용된 CPU, 메모리 자원 사용등을 측정하여 최적의 실행 쿼리를 만들어준다

⑵ 튜닝의 접근 방법

- 부하의 감소
  - 동일한 부하를 보다 효율적인 방법을 수행
- 부하의 조정
  - 부하 정도에 따라 업무를 조정하는 방법
- 부하의 병렬 수행
  - 부하가 많이 걸리는 부분을 병렬로 처리하는 방법

⑶ 튜닝 영역

- 데이터베이스 설계튜닝
  - 데이터베이스 설계 단계에서 성능을 고려하여 설계
- 데이터베이스 환경
  - 성능을 고려하여 메모리나 블록 크기 등을 지정
- SQL 문장 튜닝
  - 성능을 고려하여 SQL 문장을 작성

⑷ 튜닝절차

- 시스템 현황 분석
- 문제점 탐지 및 원인 분석
- 목표 설정
- 튜닝 시행
- 결과 분석

⑸ SQL 성능 최적화를 위한 유틸리티

- SQL Trace
- TKPROF(Trace Kernel PROFile)
- EXPLAIN PLAN

⑹ Row Migration / Row Chaining

- Row Migration
  - 데이터가 수정되면서 데이터가 더 커져서 기존 block에 못 들어가는 경우
  - 다른 블록에 데이터를 넣고, 기존 블록 위치에는 링크를 남긴다.
- Row Chaining
  - 컬럼이 너무 길어서 DB BLOCK 사이즈보다 길어진 경우 블록 두개에 이어서 한 Row가 저장

Section 10. 분산 데이터베이스

1. 분산 데이터베이스

⑴ 분산 데이터베이스(Distribute Database)의 정의

- 여러 곳으로 분산되어있는 데이터베이스를 하나의 가상 시스템으로 사용할 수 있도록 한 데이터베이스
- 논리적으로 동일한 시스템에 속하지만, 컴퓨터 네트워크를 통해 물리적으로 분산되어 있는 데이터들의 모임.
- 데이터베이스를 연결하는 빠른 네트워크 환경을 이용하여 데이터베이스를 여러 지역 여러 노드로 위치시켜 사용성/성능 등을 극대화 시킨 데이터베이스

⑵ 분산 데이터베이스 구성요소

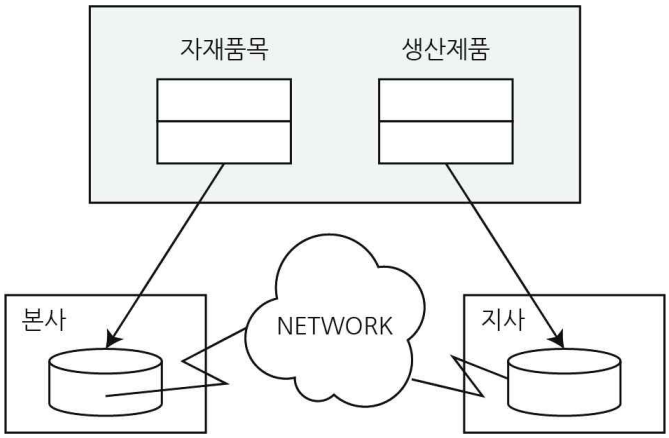
- 분산 처리기
  - 자체적으로 처리 능력을 가지며, 지리적으로 분산되어 있는 컴퓨터 시스템
- 분산 데이터베이스
  - 지리적으로 분산되어 있는 데이터베이스로서 해당 지역의 특성에 맞게 데이터베이스가 구성
- 통신 네트워크
  - 분산처리기들을 통신망으로 연결하여 논리적으로 하나의 시스템처럼 작동할 수 있도록 하는 통신 네트워크

⑶ 분산 데이터베이스 장/단점

- 장점
  - 지역 자치성, 점증적 시스템 용량 확장
  - 신뢰성과 가용성 높음
  - 효용성과 융통성 높음
  - 빠른 응답속도와 통신비용 절감
  - 시스템 규모의 적절한 조절
  - 각 지역 사용자의 요구 수용 증대
- 단점
  - 소프트웨어 개발 비용 증가
  - 오류의 잠재성 증대
  - 처리 비용의 증대
  - 설계, 관리의 복잡성과 비용 증대
  - 불규칙한 응답 속도
  - 통제의 어려움
  - 데이터 무결성에 대한 위협

⑷ 분산 데이터베이스의 적용 기법

- 테이블 위치 분산
  - 설계된 테이블의 위치를 각각 다르게 위치시키는 것
  - 테이블의 구조가 변하지 않고, 데이터베이스에 중복되어 생성되지 않는다.



- 테이블 분할(Fragmentation) 분산
  - 각각의 테이블을 쪼개어 분산하는 방법
  - 종류

수평분할 (Horizontal Fragmentation)	- 테이블을 특정 칼럼의 값을 기준으로 로우(Row)를 분리 - 컬럼은 분리되지 않음
수직분할 (Vertical Fragmentation)	- 테이블 칼럼을 기준으로 컬럼(Column)을 분리

- 테이블 복제(Replication) 분산
  - 동일한 테이블을 다른 지역이나 서버에서 동시에 생성하여 관리하는 유형
  - 종류

부분복제 (Segment Replication)	- 마스터 데이터베이스에서 테이블의 일부의 내용만 다른 지역이나 서버에 복제
광역복제 (Broadcast Replication)	- 마스터 데이터베이스의 테이블의 내용을 각 지역이나 서버에 복제

- 테이블 요약(Summarization) 분산
  - 지역간에 또는 서버 간에 데이터가 비슷하지만 서로 다른 유형으로 존재
  - 종류

분석요약 (Rollup Replication)	- 각 지역별로 존재하는 요약정보를 마스터에 통합하여 다시 전체에 대해서 요약정보를 산출하는 분산방법
통합요약 (Consolidation Replication)	- 각 지역별로 존재하는 다른 내용의 정보를 마스터에 통합하여 다시 전체에 대해서 요약정보를 산출하는 분산방법

(5) 투명성 조건

- 위치 투명성 (Location Transparency)
  - 액세스하려는 데이터베이스의 실제 위치를 알 필요없이 단지 데이터베이스의 논리적인 명칭만으로 액세스할 수 있음
- 분할 투명성 (Division Transparency)
  - 하나의 논리적 테이블이 여러 단편으로 분할되어 각 단편의 사본이 여러 위치에 저장
- 지역사상 투명성 (Local Mapping transparency)
  - 지역DBMS와 물리적 DB사이의 Mapping 보장. 각 지역시스템 이름과 무관한 이름 사용 가능
- 중복 투명성 (Replication Transparency)
  - 동일 데이터가 여러 곳에 중복되어 있더라도 사용자는 마치 하나의 데이터만 존재하는 것처럼 사용하고, 시스템은 자동으로 여러 자료에 대한 작업을 수행함
- 병행 투명성 (Concurrency Transparency)
  - 분산 데이터베이스와 관련된 다수의 트랜잭션들이 동시에 실현되더라도 그 트랜잭션의 결과는 영향을 받지 않음
- 장애 투명성 (Failure Transparency)
  - 트랜잭션, DBMS, 네트워크, 컴퓨터 장애에도 불구하고 트랜잭션을 정확하게 처리함

(6) CAP 이론

① 개념

- 어떤 분산환경에서도 일관성(C), 가용성(A), 분단 허용성(P) 세 가지 속성 중, 두 가지만 가질 수 있다는 것
- 3가지 모두 만족할 수는 없다.

② 특징의 의미

- 일관성(Consistency)
  - 모든 노드들은 같은 시간에 동일한 항목에 대하여 같은 내용의 데이터를 사용자에게 보여준다.
- 가용성(Availability)
  - 모든 사용자들이 읽기 및 쓰기가 가능해야 하며, 몇몇 노드의 장애 시에도 다른 노드에 영향을 미치면 안된다.
- 분할내성(Partition tolerance)
  - 메시지 전달이 실패하거나 시스템 일부가 망가져도 시스템이 계속 동작할 수 있어야 한다.

2. 트랜잭션

(1) 트랜잭션의 개념

- 데이터베이스의 상태를 변환시키는 하나의 논리적인 기능을 수행하는 작업 단위
- 한꺼번에 모두 수행되어야 할 연산

(2) 트랜잭션의 성질

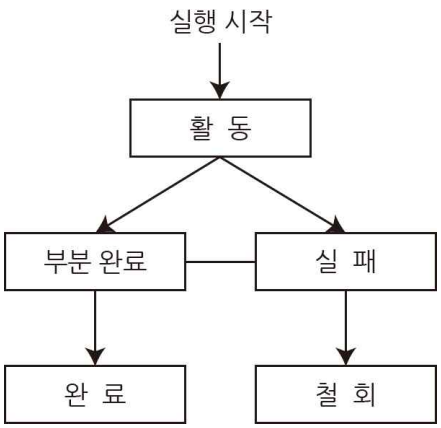
- 원자성 (Atomicity)
  - 트랜잭션의 연산은 데이터베이스에 모두 반영되든지 아니면 전혀 반영되지 않아야 한다.
  - 트랜잭션 내의 모든 명령은 반드시 완벽히 수행되어야 하며, 모두가 완벽히 수행되지 않고 하나라도 오류가 발생하면 트랜잭션 전부가 취소되어야 한다.
  - Commit과 Rollback 명령어에 의해 보장 받는다.
- 일관성 (Consistency)

- 트랜잭션이 그 실행을 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 변환한다.
- 시스템이 가지고 있는 고정요소는 트랜잭션 수행 전과 트랜잭션 수행 완료 후의 상태가 같아야 한다.

• 독립성,격리성 (Isolation)
- 둘 이상의 트랜잭션이 동시에 병행 실행되는 경우 어느 하나의 트랜잭션 실행중에 다른 트랜잭션의 연산이 끼어들 수 없다.
- 수행중인 트랜잭션은 완전히 완료될 때까지 다른 트랜잭션에서 수행 결과를 참조할 수 없다.

• 영속성 (Durablility)
- 성공적으로 완료된 트랜잭션의 결과는 시스템이 고장나더라도 영구적으로 반영되어야 한다.

(3) 트랜잭션의 상태



• 활동(Active)
- 트랜잭션이 실행중인 상태

• 실패(Failed)
- 트랜잭션 실행에 오류가 발생하여 중단된 상태

• 철회(Aborted)
- 트랜잭션이 비정상적으로 종료되어 Rollback 연산을 수행한 상태

• 부분 완료(Partially Committed)
- 트랜잭션의 마지막 연산까지 실행했지만, Commit 연산이 실행되기 직전의 상태

• 완료(Committed)
- 트랜잭션이 성공적으로 종료되어 Commit 연산을 실행한 후의 상태

12 SQL 활용

Section 1. 기본 SQL 작성

1. SQL (Structured Query Language)

⑴ SQL의 개념

- 데이터베이스 시스템에서 자료를 처리하는 용도로 사용되는 구조적 데이터 질의 언어
- 관계형 데이터베이스 관리 시스템(RDBMS)의 데이터를 관리하기 위해 설계된 특수 목적의 프로그래밍 언어

⑵ SQL의 특징

- 영어 문장과 비슷한 구문을 갖추고 있어 쉽게 배우고 사용할 수 있는 언어
- SQL은 데이터 연산에 대한 처리가 절차적으로 진행되지 않고, 데이터의 집합 단위로 처리
- 표준 SQL 문법이 존재해 DBMS 종류(Oracle, MSSQL, MySQL 등)에 얽매이지 않고 사용
- SQL은 기본적으로 대소문자를 구분하지 않는다.
- 한 줄 주석은 '--'를 앞에 붙여 사용하고, 여러 줄 주석은 /* */로 감싸준다.

⑶ SQL 문법의 종류

① Data Definition Language (DDL) - 데이터 정의어

- 데이터가 저장되는 테이블이나 각종 개체들을 정의하는데 사용되는 명령
- CREATE, ALTER, DROP, RENAME, TRUNCATE

② Data Manipulation Language (DML) - 데이터 조작어

- 데이터베이스 내의 데이터를 조작(추출, 생성, 수정, 삭제) 하는 명령
- SELECT, INSERT, UPDATE, DELETE

③ Data Control Language (DCL) - 데이터 제어어

- 데이터베이스에 접근하고, 객체들을 사용하도록 권한을 주고 회수하는 명령
- GRANT, REVOKE

④ Transaction Control Language (TCL) - 트랜잭션 제어어

- 논리적인 작업의 단위를 묶어 이에 의해 조작된 결과를 작업단위로별로 제어하는 명령어
- COMMIT, ROLLBACK, SAVEPOINT

2. DDL (Data Definition Language) - 데이터 정의어

(1) 데이터 정의어의 개념

- 데이터가 저장되는 테이블이나 각종 개체들을 정의하는데 사용되는 언어
- 데이터를 담는 그릇을 정의하는 언어

(2) 객체 유형

- 스키마 (Schema)
  - DBMS 특성과 구현 환경을 감안한 데이터 구조
- 도메인 (Domain)
  - 속성의 데이터 타입과 크기, 제약조건 등을 지정
  - 속성이 가질 수 있는 값의 범위
- 테이블 (Table)
  - 데이터의 저장 공간
- 뷰 (View)
  - 하나 이상의 물리적인 테이블에서 유도되는 가상의 논리 테이블
- 인덱스 (Index)
  - 검색을 빠르게 하기 위한 데이터 구조

(3) 조작방법

① CREATE

- 생성 명령어
- 데이터베이스 오브젝트 생성
- 데이터베이스 생성

```
// db라는 이름의 데이터베이스 생성
CREATE DATABASE db;
```

- 테이블 생성

```
// 회원 테이블 생성
CREATE TABLE 회원 (
    USER_NO INT(11) NOT NULL AUTO_INCREMENT,
    NAME VARCHAR(50) NOT NULL,
    AGE TINYINT(4) DEFAULT '0',
    PRIMARY KEY (USER_NO)
);
```

- 테이블 생성시 제약 조건

㉠ PRIMARY KEY
- 테이블의 기본키를 정의
- 기본으로 NOT NULL, UNIQUE 제약이 포함

㉡ FOREIGN KEY
- 외래키를 정의
- 참조 대상을 테이블이름(열이름)으로 명시
- 참조 무결성 위배 상황 발생 시 처리 방법으로 옵션 지정 가능
- 예) FOREIGN KEY (user_id) REFERENCES user(id) ON DELETE CASCADE;

㉢ UNIQUE
- 테이블 내에서 열은 유일한 값을 가져야 함
- 테이블 내에서 동일한 값을 가져서는 안 되는 항목에 지정함
- 예) USER_ID VARCHAR(10) UNIQUE NOT NULL,

㉣ NOT NULL
- 테이블 내에서 관련 열의 값은 NULL일 수 없음
- 필수 입력 항목에 대해 제약 조건으로 설정
- 예) USER_ID VARCHAR(10) NOT NULL,

㉤ CHECK
- 개발자가 정의하는 제약 조건
- 상황에 따라 다양한 조건 설정 가능
- 예) CONSTRAINT user_jumin CHECK(LENGTH(jumin)=13),

- INDEX 생성

```
// 회원 테이블에 search_name 이름의 인덱스 생성
CREATE INDEX search_name
ON 회원 ( name );
```

- VIEW 생성

```
// 회원 테이블에서 이름과 나이만 가져오는 VIEW 생성
CREATE VIEW v_user
AS
    SELECT name, age FROM 회원
;
```

② ALTER

- 변경 명령어
- 데이터베이스 오브젝트 변경
- 속성 추가

```
// 회원 테이블에 ADDR 속성 추가
ALTER TABLE 회원 ADD ADDR VARCHAR(200) null;
```

- 속성 변경

```
// 회원 테이블에 AGE 속성 INT 로 변경
ALTER TABLE 회원 MODIFY AGE INT(11);
```

- 속성 삭제

```
// 회원 테이블에 AGE 속성 삭제
ALTER TABLE 회원 DROP COLUMN AGE;
```

③ DROP

- 삭제 명령어
- 데이터베이스 오브젝트 삭제
- 테이블 삭제

```
// 회원 테이블 삭제
DROP TABLE 회원;
```

④ TRUNCATE

- 데이터 삭제 명령어
- 데이터베이스 오브젝트 내용 삭제 (비우기)
- 내용 삭제

```
// 회원 테이블 내용 삭제
TRUNCATE [TABLE] 회원;
```

3. DML (Data Manipulation Language) - 데이터 조작어

(1) 데이터 조작어의 개념

- 데이터베이스에 대해 데이터 검색, 등록, 삭제, 수정을 위한, 데이터베이스 언어
- 관계형 데이터베이스에 대해 검색 및 업데이트 등의 데이터 조작을 위해 사용

(2) INSERT

- 테이블의 데이터를 추가
- 컬럼의 순서와 값의 순서가 같아야 해당 컬럼으로 데이터가 추가된다.

```
// 회원 데이터 추가
INSERT INTO 회원 ( NAME, AGE )
VALUES ( '이흥직', '42' );

// 여러 행을 검색하여 추가
INSERT INTO 회원
SELECT NAME, AGE FROM 회원2
```

(3) SELECT

- 하나 또는 그 이상의 테이블에서 데이터를 추출하는 SQL의 데이터 조작 언어
- 사용 함수
  - WHERE : 조건에 맞는 데이터를 지정
  - GROUP BY : 특정 속성을 그룹으로 만들어서 집계 함수를 사용
  - HAVING : GROUP BY 절에 정의된 조건
  - ORDER BY : 반환되는 행의 순서를 지정
- 논리연산자
  - NOT, AND, OR
- 사용 예

<<사원정보>>

사번	부서	이름	직책	입사일	나이
0001	개발팀	이흥직	차장	2003	42
0002	영업팀	홍길동	과장	2005	38
0003	개발팀	김길	대리	(NULL)	32
0004	경영지원팀	김영희	차장	2002	43
0005	영업팀	이철수	대리	2012	28
0006	디자인팀	이창훈	사원	2010	26
0007	디자인팀	소찬휘	과장	2007	40
0008	기획팀	김명민	차장	2001	45

- 모든 직원 정보 조회

```
SELECT * FROM 사원정보;
```

- 개발팀 직원 정보 조회(사번, 이름, 직책 만)

```
SELECT 사번, 이름, 직책 FROM 사원정보
WHERE 부서 = '개발팀';
```

- 개발팀 이나 디자인팀 조회

```
SELECT * FROM 사원정보
WHERE 부서 = '개발팀'
OR 부서 = '디자인팀';
```

- 개발팀 이나 디자인팀 이면서 이씨만 조회

```
SELECT * FROM 사원정보
WHERE ( 부서 = '개발팀' OR 부서 = '디자인팀' )
AND 이름 LIKE '이%';
```

- 개발팀 이나 디자인팀 이면서 이씨로 시작하고 이름이 두 글자만 조회

```
SELECT * FROM 사원정보
WHERE ( 부서 = '개발팀' OR 부서 = '디자인팀' )
AND 이름 LIKE '이_';
```

- 개발팀 이면서 입사일이 NULL인 직원 조회

```
SELECT * FROM 사원정보
WHERE 부서 = '개발팀'
AND 입사일 IS NULL;
```

- 개발팀 이면서 입사일이 NULL이 아닌 직원 조회

```
SELECT * FROM 사원정보
WHERE 부서 = '개발팀'
AND 입사일 IS NOT NULL;
```

- 입사일이 2010년 이후 직원조회

```
SELECT * FROM 사원정보
WHERE 입사일 >= '2010';
```

- 입사일이 2010년부터 2020년까지 출력

```
SELECT * FROM 사원정보
WHERE 입사일 BETWEEN 2010 AND 2020;
```

- 사원정보에서 부서만 출력

```
SELECT 부서 FROM 사원정보;
```

- 사원정보에서 부서만 중복없이 출력

```
SELECT DISTINCT 부서 FROM 사원정보;
```

- 부서별 인원수 출력

```
SELECT 부서, COUNT(*) AS 인원수 FROM 사원정보
GROUP BY 부서;
```

- 부서별 인원수 출력 (부서별 인원 2명 이상)

```
SELECT 부서, COUNT(*) AS 인원수 FROM 사원정보
GROUP BY 부서
HAVING COUNT(*) >= 2;
```

- 부서별 인원수와 나이 합산

```
SELECT 부서, COUNT(*) AS 인원수, SUM(나이) AS 나이
FROM 사원정보
GROUP BY 부서;
```

- 모든 직원 정보 조회 나이별 내림차순

```
SELECT * FROM 사원정보
ORDER BY 나이 DESC;
```

(4) UPDATE

- 테이블에 있는 데이터를 갱신
- 셀단위로 갱신이 가능
- WHERE 절을 생략했을 경우 모든 행이 갱신
- 사용 예

- 모든 직원의 나이를 1살 증가

```
UPDATE 사원정보
SET
   나이 = 나이 + 1
```

- 개발팀의 부서명을 개발지원팀으로 변경

```
UPDATE 사원정보
SET
   부서 = '개발지원팀'
WHERE 부서 = '개발팀'
```

(5) DELETE

- 테이블에 있는 일부 데이터를 직접 삭제
- WHERE 절을 생략했을 경우 모든 행이 삭제
- 사용 예

- 개발팀 직원만 삭제

```
DELETE FROM 사원정보
WHERE 부서 = '개발팀'
```

- 디자인팀 '이창훈'만 삭제

```
DELETE FROM 사원정보
WHERE 부서 = '디자인팀'
AND 이름 = '이창훈'
```

4. DCL (Data Control Language) - 데이터 제어어

(1) 데이터 제어어의 개념

- 데이터베이스에 접근하거나 객체에 권한을 주는 등의 역할을 하는 언어
- 사용자에게 권한 생성 혹은 권한 삭제 같은 명령어

(2) GRANT

- 데이터베이스 사용자에게 권한을 부여하는 명령
- 기본형태 : GRANT [권한] ON [객체명] TO [사용자 계정] [WITH GRANT OPTION];
- WITH GRANT OPTION : 지정된 권한을 다른 유저에게 부여할 수 있도록 함
- 사용 예
  - HUNGJIK 사용자에게 사원정보 SELECT 권한 부여

```
GRANT SELECT ON 사원정보 TO HUNGJIK;
```

  - HUNGJIK 사용자에게 사원정보 SELECT 권한 부여하고 다른 사용자에게 부여 할 수 있도록

```
GRANT SELECT ON 사원정보 TO HUNGJIK WITH GRANT OPTION;
```

(3) REVOKE

- 데이터베이스 사용자에게서 권한을 회수하는 명령
- 기본형태 : REVOKE [권한] ON [객체명] FROM [사용자 계정] [CASCADE]
- CASCADE : 사용자가 부여한 모든 사용자 권한을 같이 회수
- 사용 예
  - HUNGJIK 사용자에게 사원정보 SELECT 권한 회수

```
REVOKE SELECT ON 사원정보 FROM HUNGJIK;
```

(4) 사용 예

```
DBA : GRANT SELECT ON STUDENT TO U1 WITH GRANT OPTION
U1 : GRANT SELECT ON STUDENT TO U2
DBA : REVOKE SELECT ON STUDENT FROM U1 CASCADE
```

- DBA 가 U1 사용자에게 SELECT 권한을 주면서 U1 사용자에게 다른 계정에 대한 SELECT 권한을 부여할 수 있도록 WITH GRANT OPTION 추가
- U1 사용자가 U2 사용자에게 SELECT 권한 부여
- DBA 가 U1 사용자에게 부여했던 SELECT 권한을 회수하며, U1 사용자가 부여했던 모든 사용자에 대한 권한도 함께 회수

5. TCL (Transaction Control Language) - 트랜잭션 제어어

(1) 트랜잭션 제어어의 개념

- 트랜잭션을 제어하는 명령인 COMMIT과 ROLLBACK만을 따로 분리해서 TCL이라고 표현

(2) 트랜잭션 제어어 종류

- COMMIT
  - 트랜잭션 처리가 정상적으로 종료되어 트랜잭션이 수행한 변경 내용을 데이터베이스에 반영하는 연산
  - 내용을 변경한 트랜잭션이 완료되면 그 트랜잭션에 의해 데이터베이스는 새롭게 일관된 상태로 변경되며, 이 상태는 시스템 오류가 발생하더라도 취소되지 않는다.
- ROLLBACK
  - 데이터 변경 사항이 취소되어 데이터의 이전 상태로 복구
  - 관련된 행에 대한 잠금이 풀리고, 다른 사용자들이 행을 조작할 수 있게 됨
- SAVEPOINT
  - 저장점을 정의하면 롤백할 때 트랜잭션에 포함된 전체 작업을 롤백하는 것이 아니라 현 시점에서 SAVEPOINT 까지 트랜잭션의 일부만 롤백

Section 2. SELECT 쿼리 활용

1. 집합 연산자

(1) 집합 연산자

- 여러 개의 질의의 결과를 연결하여 하나로 결합하는 방식
- 두 개 이상의 테이블에서 조인을 사용하지 않고 연관된 데이터를 조회하는 방법
- 집합 연산자를 사용하기 위해서는 SELECT 절의 컬럼 수가 동일하고, 동일 위치에 존재하는 컬럼의 데이터 타입이 상호 호환 가능해야 한다.

(2) 집합 연산자 종류

- UNION
  - 여러 개의 SQL 문의 결과에 대한 합집합으로 결과에서 모든 중복된 행은 하나의 행으로 출력
- UNION ALL
  - 여러 개의 SQL 문의 결과에 대한 합집합으로 중복된 행도 그대로 결과로 표시
- INTERSECT
  - 여러 개의 SQL 문의 결과에 대한 교집합으로 중복된 행은 하나의 행으로 출력
- EXCEPT (MINUS)
  - 앞의 SQL 문의 결과에서 뒤의 SQL 문의 결과에 대한 차집합으로 중복된 행은 하나의 행으로 출력

(3) 집합 연산자 사용 예

<<수강정보>>

순번	과목명	이름
0001	국어	이흥직
0002	영어	이흥직
0003	수학	이흥직
0004	도덕	이흥직
0005	국어	홍길동
0006	수학	홍길동
0007	사회	홍길동
0008	과학	홍길동

- UNION

```
SELECT 과목명 FROM 수강정보 WHERE 이름 = '이흥직'
UNION
SELECT 과목명 FROM 수강정보 WHERE 이름 = '홍길동'
```

<<결과값>>

과목명
국어
영어
수학
도덕
사회
과학

- UNION ALL

```
SELECT 과목명 FROM 수강정보 WHERE 이름 = '이흥직'
UNION ALL
SELECT 과목명 FROM 수강정보 WHERE 이름 = '홍길동'
```

<<결과값>>

과목명
국어
영어
수학
도덕
국어
수학
사회
과학

- INTERSECT

```
SELECT 과목명 FROM 수강정보 WHERE 이름 = '이흥직'
INTERSECT
SELECT 과목명 FROM 수강정보 WHERE 이름 = '홍길동'
```

<<결과값>>

과목명
국어
수학

- MINUS

```
SELECT 과목명 FROM 수강정보 WHERE 이름 = '이흥직'
MINUS
SELECT 과목명 FROM 수강정보 WHERE 이름 = '홍길동'
```

<<결과값>>

과목명
영어
도덕

2. JOIN

(1) JOIN 의 개념

- 두 개 이상의 테이블을 결합하여 데이터를 검색하는 방법
- 조인(JOIN) 연산자를 사용해 관련 있는 컬럼 기준으로 행을 합쳐주는 연산
- Primary Key 혹은 Foreign Key로 두 테이블을 연결

(2) JOIN 의 종류

- 내부 조인 (INNER JOIN)
  - 교차 조인 (CROSS JOIN - CARTESIN JOIN)
  - 등가/동등/동일 조인(EQUI JOIN)
  - 비등가 조인(NON-EQUI JOIN)
  - 자연 조인 (NATURAL JOIN)
- 외부 조인 (OUTER JOIN)
  - 완전 외부 조인 (FULL OUTER JOIN)
  - 왼쪽 (LEFT OUTER)
  - 오른쪽 (RIGHT OUTER)
- 셀프 조인 (SELF JOIN)

(3) 내부 조인 (INNER JOIN)

- 두 테이블에 존재하는 데이터 중에 공통된 데이터만 추출

<<회원정보>>

회원번호	이름
0001	이흥직
0002	홍길동
0003	이창훈
0004	정광호

<<주문>>

주문번호	회원번호	상품
O_01	0001	상품1
O_02	0001	상품2
O_03	0003	상품1
O_04	0004	상품2

- 사용 예

```
SELECT * FROM 회원정보 INNER JOIN 주문
ON 회원정보.회원번호 = 주문.회원번호
```

<<결과>>

회원번호	이름	주문번호	회원번호	상품
0001	이흥직	O_01	0001	상품1
0001	이흥직	O_02	0001	상품2
0003	이창훈	O_03	0003	상품1
0004	정광호	O_04	0004	상품2

(4) 외부 조인 (OUTER JOIN)

- LEFT OUTER JOIN
  - 왼쪽 테이블을 기준으로 오른쪽 테이블을 JOIN
  - 사용 예

```
SELECT * FROM 회원정보 LEFT OUTER JOIN 주문
ON 회원정보.회원번호 = 주문.회원번호
```

<<결과>>

회원번호	이름	주문번호	회원번호	상품
0001	이흥직	O_01	0001	상품1
0001	이흥직	O_02	0001	상품2
0003	이창훈	O_03	0003	상품1
0004	정광호	O_04	0004	상품2
0002	홍길동	(NULL)	(NULL)	(NULL)

- RIGHT OUTER JOIN
  - 오른쪽 테이블을 기준으로 왼쪽 테이블을 JOIN
- FULL OUTER JOIN
  - 왼쪽, 오른쪽 데이터를 모두 JOIN

3. 서브쿼리

(1) 서브쿼리의 개념

- SELECT 문 안에 다시 SELECT 문이 기술된 형태의 쿼리

(2) 서브쿼리의 종류

① 스칼라 서브쿼리 (Scalar Subqueries)

- SELECT 절 안에 서브쿼리가 들어있는 형태
- 반드시 단일행이거나, SUM/COUNT 등의 집계함수를 거친 단일값이 리턴되어야 한다.
- 사용 예

<<회원정보>>

회원번호	이름
0001	이흥직
0002	홍길동
0003	이창훈
0004	정광호

<<주문>>

주문번호	회원번호	상품
O_01	0001	상품1
O_02	0001	상품2
O_03	0003	상품1
O_04	0004	상품2

```
SELECT
    이름,
    (SELECT 주문번호 FROM 주문 WHERE 주문.회원번호 = 회원정보.회원번호) as 주문번호
FROM 회원정보
WHERE 회원번호 = '0003';
```

<<결과>>

이름	주문번호
이창훈	O_03

② 인라인뷰 서브쿼리 (Inline Views Subqueries)

- FROM 절 안에 서브쿼리 존재
- 서브쿼리의 결과는 반드시 하나의 테이블로 리턴되어야 한다.
- 사용 예

```
SELECT
    회원번호, 이름
FROM (
    SELECT * FROM 회원정보
    WHERE 이름 like '이%'
) TB_TEMP;
```

<<결과>>

회원번호	이름
001	이흥직
003	이창훈

③ 중첩 서브쿼리 (Nested Subqueries)

- WHERE 절 안에 서브쿼리 존재
- 단일행 서브쿼리 연산자 : >, >=, <, <=, = 등
- 다중행 서브쿼리 연산자 : IN, ANY, SOME, ALL, EXISTS
- IN 사용 예

 - 테이블의 모든 컬럼 값을 직접 비교한다.

<<회원정보>>

회원번호	이름
0001	이흥직
0002	홍길동
0003	이창훈
0004	정광호

<<주문>>

주문번호	회원번호	상품
O_01	0001	상품1
O_02	0001	상품2
O_03	0003	상품1
O_04	0004	상품2

```
SELECT
    회원번호, 이름
FROM 회원정보
WHERE 회원번호 IN (
    SELECT 회원번호 FROM 주문
) ;
```

<<결과>>

회원번호	이름
001	이흥직
003	이창훈
004	정광호

- EXISTS 사용 예

 - 해당 값이 TRUE/FALSE 인지만을 비교한다.

 - 대량의 데이터 조회 시 IN 보다 빠른 성능을 가진다.

<<학생>>

학번	이름	학년	학과	주소
1000	김철수	1	전산	서울
2000	고영준	1	전기	경기
3000	유진호	2	전자	경기
4000	김영진	2	전산	경기
5000	정현영	3	전자	서울

<<성적>>

학번	과목번호	과목이름	학점	점수
1000	A100	자료구조	A	91
2000	A200	DB	A+	99
3000	A100	자료구조	B+	88
3000	A200	DB	B	85
4000	A200	DB	A	94
4000	A300	운영체제	B+	89
5000	A300	운영체제	B	88

```
SELECT 과목이름
FROM 성적
WHERE EXISTS (
    SELECT
        학번
    FROM 학생
    WHERE 학생.학번=성적.학번
    AND 학생.학과 IN ('전산', '전기')
    AND 학생.주소='경기'
);
```

<<결과>>

과목이름
DB
DB
운영체제

Section 3. 그룹함수와 윈도우 함수

1. 그룹함수와 집계함수

(1) 그룹함수

- 테이블의 전체 행을 지정한 컬럼 값에 따라 그룹화하여 그룹별로 결과를 출력하는 함수

종류	설명
ROLLUP	- 그룹별 중간 집계값을 생성한다.
CUBE	- 결합 가능한 값에 대한 다차원 집계를 생성한다.
GROUPING SETS	- 개별 집계를 구한다.

(2) 집계함수

- 여러 행 또는 전체 행으로부터 하나의 결과값을 반환하는 함수

종류	설명
COUNT	- 행의 개수를 반환
SUM	- 특정 컬럼의 합계를 반환
AVG	- 특정 컬럼의 평균을 반환
MAX	- 특정 컬럼에서 최대값을 반환
MIN	- 특정 컬럼에서 최솟값을 반환
STDDEV	- 특정 컬럼간의 표준편차를 반환
VARIAN	- 특정 컬럼간의 분산을 계산하여 반환

2. 그룹함수 사용

<<성적표>>

학번	이름	학기	과목	점수
0001	이흥직	1	영어	80
0001	이흥직	1	수학	75
0002	김보연	1	영어	90
0002	김보연	1	수학	70
0001	이흥직	2	영어	95
0001	이흥직	2	수학	70
0002	김보연	2	영어	85
0002	김보연	2	수학	75

(1) GROUP BY

① 학번별로 총점과 평균 구하기

```
SELECT
    학번,
    SUM(점수) AS 총점,
    AVG(점수) AS 평균
FROM 성적표
GROUP BY 학번
```

<<결과>>

학번	총점	평균
0001	320	80
0002	320	80

② 학번/학기별로 총점과 평균 구하기

```
SELECT
    학번,
    학기,
    SUM(점수) AS 총점,
    AVG(점수) AS 평균
FROM 성적표
GROUP BY 학번, 학기
```

<<결과>>

학번	학기	총점	평균
0001	1	155	77.5
0001	2	165	82.5
0002	1	160	80
0002	2	160	80

(2) ROLLUP

① 학번/학기별로 총점과 평균 구하기

```
SELECT
    학번,
    학기,
    SUM(점수) AS 총점,
    AVG(점수) AS 평균
FROM 성적표
GROUP BY ROLLUP(학번, 학기)
```

<<결과>>

학번	학기	총점	평균
0001	1	155	77.5
0001	2	165	82.5
001	(NULL)	320	80
0002	1	160	80
0002	2	160	80
002	(NULL)	320	80
(NULL)	(NULL)	640	80

(3) CUBE

- ROLLUP에서는 단지 가능한 Subtotal만을 생성하였지만, CUBE는 결합 가능한 모든 값에 대하여 다차원 집계를 생성한다.
- 그룹핑 컬럼이 가질 수 있는 모든 경우의 수에 대하여 소계(SUBTOTAL)과 총계(GRAND TOTAL)을 생성한다.
- 시스템에 많은 부담을 주기 때문에 데이터 양이 많다면 사용을 지양한다.

```
SELECT
    학번,
    학기,
    SUM(점수) AS 총점,
    AVG(점수) AS 평균
FROM 성적표
GROUP BY CUBE(학번, 학기)
```

(4) GROUPING SETS

- ROLLUP과 CUBE와 달리 계층 구조가 나타나지 않으며 따라서 인자의 순서가 달라도 결과는 똑같다.
- GROUPING SETS 함수는 괄호로 묶은 집합별로도 집계를 구할 수 있다.

```
SELECT
    학번,
    학기,
    SUM(점수) AS 총점,
    AVG(점수) AS 평균
FROM 성적표
GROUP BY GROUPING SETS(학번, 학기)
```

3. 윈도우 함수

(1) 윈도우 함수 개념

- 그룹 함수들에 대해서 데이터 처리를 간단하게 하기 위한 함수이다.
- 윈도우 함수는 중첩해서 사용할 수 없다.
- 윈도우 함수는 서브 쿼리에서 사용할 수 있다.
- 윈도우 함수는 'OVER' 구문이 필수로 포함되어야 한다.
- 윈도우 함수를 OLAP 함수라고도 한다.

(2) 윈도우 함수 분류

분류	종류
순위 함수	- 그룹 내의 순위를 계산하는 함수 - RANK, DENSE_RANK, ROW_NUMBER
집계 함수	- 그룹 내의 값을 집계하는 함수 - SUM, MAX, MIN, AVG, COUNT
순서 함수	- 그룹 내의 행의 순서를 구하는 함수 - FIRST_VALUE, LAST_VALUE, LAG, LEAD
비율 함수	- 그룹 내의 비율을 구하는 함수 - CUME_DIST, PERCENT_RANK, NTILE, RATIO_TO_REPORT

⑶ 윈도우 함수 문법

```
SELECT
    WINDOW_FUNCTION (ARGUMENTS)
    OVER (
        [PARTITION BY 컬럼]
        [ORDER BY 절]
        [WINDOWING 절]
    )
FROM 테이블명;
```

- WINDOW_FUNCTION
  - 함수명
- ARGUMENTS (인수)
  - 함수에 따라 0 ~ N개의 인수가 지정될 수 있다.
- PARTITION BY 절
  - 전체 집합을 기준에 의해 소그룹으로 나눌 수 있다.
- ORDER BY 절
  - 어떤 항목에 대해 순위를 지정할 지 ORDER BY 절을 기술한다.
- WINDOWING 절
  - 함수의 대상이 되는 행 기준의 범위를 지정할 수 있다.

⑷ 그룹 내 순위(RANK) 관련 함수

<<학생>>

학번	학과	이름	점수
1000	전산	철수	70
2000	전기	영준	85
3000	전자	진호	95
4000	전산	영진	100
5000	전자	현영	85

- RANK
  - ORDER BY를 포함한 쿼리문에서 특정 컬럼에 대한 순위를 구하는 함수
- DENSE_RANK
  - RANK 함수와 비슷하지만, 동일한 순위를 하나의 건수로 취급하는 함수
- ROW_NUMBER
  - RANK나 DENSE_RANK 함수가 동일한 값에 대해서는 동일한 순위를 부여하는데 반해, 동일한 값이라도 고유한 순위를 부여한다.

- 사용 예

```
SELECT
    학번,
    이름,
    점수,
    RANK() OVER(ORDER BY 점수 DESC) AS '순위1',
    DENSE_RANK() OVER(ORDER BY 점수 DESC) AS '순위2',
    ROW_NUMBER OVER(ORDER BY 점수 DESC) AS '순위3'
FROM 학생;
```

<<결과>>

학번	이름	점수	순위1	순위2	순위3
4000	영진	100	1	1	1
3000	진호	95	2	2	2
5000	현영	85	3	3	3
2000	영준	85	3	3	4
1000	철수	70	5	4	5

(5) 그룹 내 행 순서 관련 함수

- FIRST_VALUE
  - 파티션별 윈도우에서 가장 먼저 나온 값을 구한다.
  - MIN 함수를 활용하여 같은 결과를 얻을 수 있다.
- LAST_VALUE
  - 파티션별 윈도우에서 가장 나중에 나온 값을 구한다.
  - MAX 함수를 활용하여 같은 결과를 얻을 수 있다.
- LAG
  - 파티션별 윈도우에서 이전 몇 번째 행의 값을 가져올 수 있다.
  - 두 번째 인자는 몇 번째 앞의 행을 가져올지 결정하고, 3번째 인자는 값이 없을 때 다른 값으로 변경한다.
- LEAD
  - 파티션별 윈도우에서 이후 몇 번째 행의 값을 가져올 수 있다.
- FIRST_VALUE, LAST_VALUE 사용 예

```
SELECT
    학번,
    이름,
    점수,
    FIRST_VALUE(이름) OVER (ORDER BY 점수 DESC ) AS F_VAL,
    LAST_VALUE(이름) OVER (ORDER BY 점수 DESC ) AS L_VAL,
FROM 학생;
```

<<결과>>

학번	이름	점수	F_VAL	L_VAL
1000	영진	100	영진	철수
5000	진호	95	영진	철수
5000	현영	85	영진	철수
2000	영준	85	영진	철수
1000	철수	70	영진	철수

- LAG, LEAD 사용예

```
SELECT
    학번,
    이름,
    점수,
    LAG(이름) OVER (ORDER BY 점수 DESC ) AS LAG,
    LEAD(이름) OVER (ORDER BY 점수 DESC ) AS LEAD
FROM 학생;
```

<<결과>>

학번	이름	점수	LAG	LEAD
1000	영진	100	(NULL)	진호
5000	진호	95	영진	현영
5000	현영	85	진호	영준
2000	영준	85	현영	철수
1000	철수	70	영준	(NULL)

⑹ 그룹 내 비율 관련 함수

- CUME_DIST
  - 파티션별 윈도우의 전체 건수에서 현재 행보다 작거나 같은 건수에 대한 누적백분율을 구한다.
- PERCENT_RANK
  - 파티션별 윈도우에서 제일 먼저 나오는 것으로 0으로, 제일 늦게 나오는 것을 1로 하여, 값이 아닌 행의 순서별 백분율을 구한다.
- NTILE
  - 파티션별 전체 건수를 인수 값으로 N 등분한 결과를 구할 수 있다.
  - NTILE을 활용하면 등분하여 순위를 매길 수 있다
  - NTILE(3) : 상/중/하
  - NTILE(2) : 상/하
- RATIO_TO_REPORT
  - 파티션 내 전체 SUM(컬럼) 값에 대한 행별 컬럼 값의 백분율을 소수점으로 구할 수 있다.
- 사용 예

```
SELECT
    학번,
    이름,
    점수,
    ROUND(RATIO_TO_REPORT(점수) OVER(), 2) AS R_R
FROM 학생;
```

Section 4. 절차형 SQL

1. 저장 프로시저 (Stored Procedure)

(1) 저장 프로시저의 개념

- 일련의 쿼리를 마치 하나의 함수처럼 실행하기 위한 쿼리의 집합
- 데이터베이스에 대한 일련의 작업을 정리한 절차를 관계형 데이터베이스 관리 시스템에 저장한 모듈
- 리턴값이 없거나, 하나 또는 여러 개의 리턴값을 가질 수 있다.

(2) 저장 프로시저의 장/단점

- 장점
  - DB 보안 향상
  - 네트워크 소요 시간 절감
  - 절차적 기능 구현
  - 개발 업무 구분
- 단점
  - 낮은 처리 성능
  - 유지 보수 어려움

(3) 저장 프로시저의 구조

```
CREATE OR REPLACE PROCEDURE 프로시저명
( 변수1 IN 변수타입, 변수2 OUT 변수타입, 변수3 IN OUT 변수타입.... )
IS
    변수 처리부
BEGIN
    처리내용
EXCEPTION
    예외처리부
END;
```

2. 트리거

(1) 트리거의 개념

- 테이블에 대한 이벤트에 반응해 자동으로 실행되는 작업
- 특정 테이블에 INSERT, DELETE, UPDATE 같은 DML 문이 수행되었을 때, 데이터베이스에서 자동으로 동작하도록 작성된 프로그램

(2) 트리거의 유형

- 행 트리거
  - 테이블 안의 영향을 받은 행 각각에 대해 실행
  - 데이터 변화가 생길 때마다 실행
  - 변경 전 또는 변경 후의 레코드(rows)는 OLD, NEW라는 가상 줄 변수를 이용해 사용 가능
  - FOR EACH ROW 옵션 사용
- 문장 트리거
  - INSERT, UPDATE, DELETE 문에 대해 단 한번만 실행

(3) 트리거의 실행 시기

- BEFORE : 이벤트 전
- AFTER : 이벤트 후

(4) 트리거 생성 예

```
CREATE TRIGGER TRIGGER_GOODS_STOCK
AFTER INSERT ON TB_GOODS_STOCK FOR EACH ROW
BEGIN
    UPDATE TB_GOODS
    SET
        nStock = nStock + NEW.nStock
    WHERE idx = NEW.p_idx;
END
```

3. 사용자 정의 함수

(1) 사용자 정의 함수의 개념

- 프로시저와 사용자 정의 함수 모두 호출하게 되면 미리 정의 해놓은 기능을 수행하는 모듈
- 프로그램 로직을 도와주는 역할을 한다.
- 파라미터는 입력 파라미터만 가능하고, 리턴값이 하나이다.

(2) 사용자 정의 함수의 구조

```
CREATE OR REPLACE FUNCTION 함수명
( 매개변수1, 매개변수2, 매개변수3,..... )
RETURN 데이터 타입
IS
    변수 처리부
BEGIN
    처리내용
    RETURN 반환값;
EXCEPTION
    예외처리부
END;
```

13 병행제어와 데이터전환

Section 1. 병행제어와 회복

1. 병행제어

(1) 병행제어의 개념

- 여러 트랜잭션들이 동시에 실행되면서도 데이터베이스의 일관성을 유지할 수 있게 하는 기법
- 동시에 여러 개의 트랜잭션을 병행 수행할 때, 트랜잭션들이 DB의 일관성을 파괴하지 않도록 트랜잭션 간의 상호작용을 제어하는 것

(2) 병행제어의 목적

- 데이터베이스의 공유도 최대화
- 시스템 활용도 최대화
- 응답시간 최소화
- 단위 시간당 트랜잭션 처리건수 최대화
- DB 일관성 유지

(3) 병행제어의 문제점

① 갱신 분실 (Lost Update)

- 두 개 이상의 트랜잭션이 같은 자료를 공유하여 갱신할 때 갱신 결과의 일부가 없어지는 현상

데이터	트랜잭션 1	트랜잭션 2
1000	READ(1000)	
1000		READ(1000)
1000	ADD(1000)	
1000		ADD(2000)
2000	STORE(2000)	
3000		STORE(3000)

② 비완료 의존성 (Uncommitted Dependency)

- 하나의 트랜잭션 수행이 실패한 후 회복되기 전에 다른 트랜잭션이 실패한 갱신 결과를 참조하는 현상

데이터	트랜잭션 1	트랜잭션 2
1000	READ(1000)	
1000	ADD(1000)	
2000	STORE(2000)	READ(2000)
1000	장애로 ROLLBACK	

③ 모순성 (Inconsistency)

- 두 개의 트랜잭션이 병행수행될 때 원치 않는 자료를 이용함으로써 발생하는 문제
- 갱신 분실과 비슷해 보이지만 여러 데이터를 가져올 때 발생하는 문제

④ 연쇄 복귀 (Cascading Rollback)

- 병행수행되던 트랜잭션들 중 어느 하나에 문제가 생겨 Rollback하는 경우 다른 트랜잭션도 함께 Rollback되는 현상

데이터	트랜잭션 1	트랜잭션 2
1000	READ(1000)	
1000	ADD(1000)	
2000	STORE(2000)	
2000		READ(2000)
2000		ADD(2000)
4000		STORE(4000)
1000	장애로 ROLLBACK	

(4) 병행제어 기법

① 로킹(Locking)

- 트랜잭션이 어떤 데이터에 접근하고자 할 때 로킹 수행
- 로킹이 되어 있는 데이터에는 다른 트랜잭션이 접근할 수 없음
- 트랜잭션은 로킹이 된 데이터에 대해서만 연산 수행
- 로킹 단위: 필드, 레코드, 파일, 데이터베이스 모두 로킹 단위가 될 수 있음
- 로킹 단위에 따른 구분

구분	로크 수	병행성	오버헤드
로킹 단위가 크면	적어짐	낮아짐	감소
로킹 단위가 작으면	많아짐	높아짐	증가

② 2단계 로킹 규약(Two-Phase Locking Protocol)

- Lock과 Unlock이 동시에 이루어지면 일관성이 보장되지 않으므로 Lock만 가능한 단계와 Unlock만 가능한 단계를 구분
- 확장단계: 새로운 Lock은 가능하고 Unlock은 불가능하다.
- 축소단계: Unlock 은 가능하고 새로운 Lock은 불가능하다.
- 직렬가능성을 보장
- 교착상태가 발생할 수 있다.

③ 타임스탬프(Time Stamp)

- 데이터에 접근하는 시간을 미리 정하여서 정해진 시간(Time Stamp)의 순서대로 데이터에 접근하여 수행
- 직렬가능성을 보장
- 교착상태가 발생하지 않는다.
- 연쇄복귀(Cascading Rollback)를 초래할 수 있음

④ 낙관적 병행제어(Optimistic Concurrency Control)

- 트랜잭션 수행 동안은 어떠한 검사도 하지 않고, 트랜잭션 종료 시에 일괄적으로 검사
- 트랜잭션 수행 동안 그 트랜잭션을 위해 유지되는 데이터 항목들의 지역 사본에 대해서만 갱신
- 트랜잭션 종료 시에 동시성을 위한 트랜잭션 직렬화가 검증되면 일시에 DB로 반영

⑤ 다중 버전 병행제어(Multi-version, Concurrency Control)

- 여러 버전의 타임스탬프를 비교하여 스케줄상 직렬가능성이 보장되는 타임스탬프를 선택
- 충돌이 발생할 경우 복귀 수행. 연쇄 복귀 발생 가능성

2. 회복 (Database Recovery)

(1) 회복의 개념

- 트랜잭션들을 수행하는 도중 장애로 인해 손상 된 데이터베이스를 손상되기 이전의 정상적인 상태로 복구시키는 작업

(2) 장애의 유형

- 트랜잭션 장애
  - 트랜잭션의 실행 시 논리적인 오류로 발생할 수 있는 에러 상황
- 시스템 장애
  - H/W 시스템 자체에서 발생할 수 있는 에러 상황
- 미디어 장애
  - 디스크 자체의 손상으로 발생할 수 있는 에러 상황

(3) Undo와 Redo

- Undo
  - 트랜잭션 로그를 이용하여 오류와 관련된 모든 변경을 취소하여 복구 수행
- Redo
  - 트랜잭션 로그를 이용하여 오류가 발생한 트랜잭션을 재실행하여 복구 수행

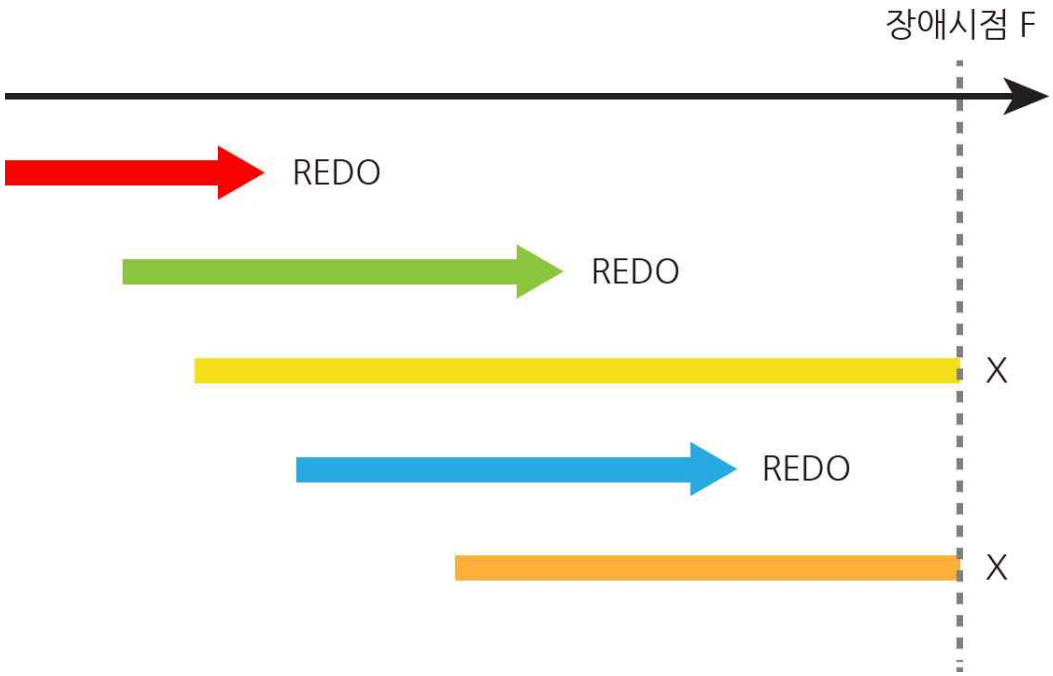
(4) 로그 파일

- 트랜잭션이 반영한 모든 데이터의 변경사항을 데이터베이스에 기록하기 전에 미리 기록해두는 별도의 파일
- 안전한 하드디스크에 저장되며 전원과 관계없이 기록이 존재
- 로그파일을 이용한 복구
  - 로그파일에 트랜잭션의 시작(START)와 종료(COMMIT)가 있는 경우 REDO 수행
  - 로그파일에 트랜잭션의 시작(START)는 있고 종료(COMMIT)가 없는 경우 UNDO 수행

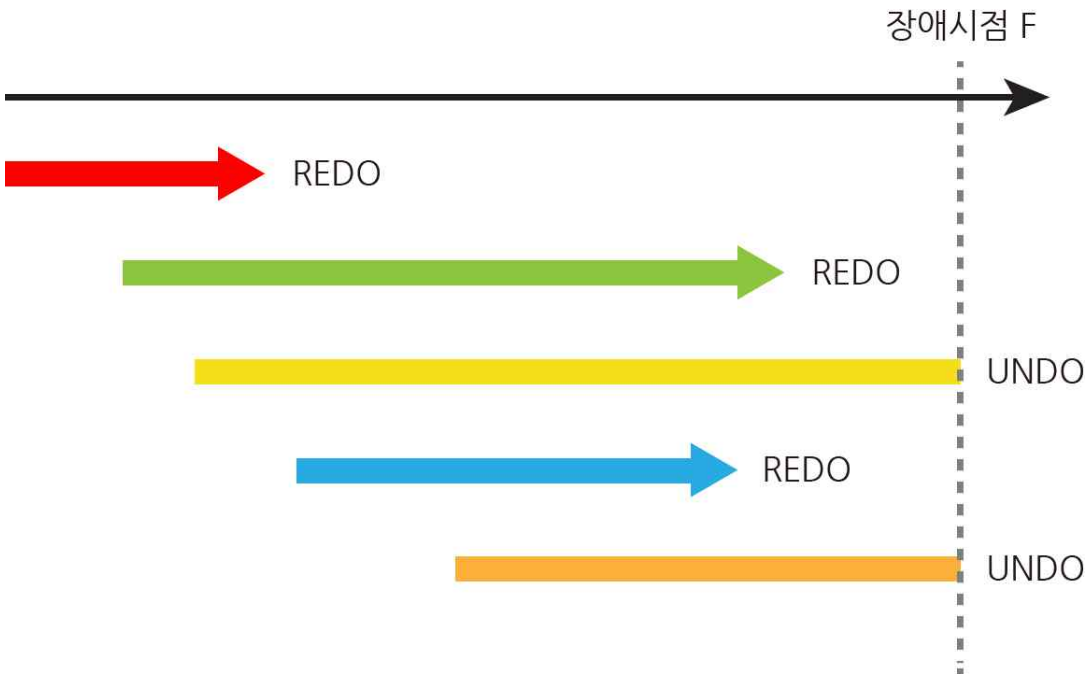
(5) 회복 기법

① 로그 기반 회복 기법

- 지연갱신 회복 기법(Deferred Update)
  - 트랜잭션의 부분 완료 상태에선 변경 내용을 로그 파일에만 저장
  - 커밋이 발생하기 전까진 데이터베이스에 기록하지 않음
  - 중간에 장애가 생기더라도 데이터베이스에 기록되지 않았으므로 UNDO가 필요 없음(미실행 된 로그 폐기)



- 즉시갱신 회복 기법(Immediate Update)
  - 트랜잭션 수행 도중에도 변경 내용을 즉시 데이터베이스에 기록
  - 커밋 발생 이전의 갱신은 원자성이 보장되지 않는 미완료 갱신이므로 장애 발생 시 UNDO 필요



② 검사점 회복 기법 (Checkpoint Recovery)

- 체크포인트(Checkpoint) 회복 기법
- 장애 발생 시 검사점(Checkpoint) 이전에 처리된 트랜잭션은 회복에서 제외하고 검사점 이후에 처리된 트랜잭션은 회복 작업 수행

③ 그림자 페이징 회복 기법 (Shadow Paging Recovery)

- 트랜잭션이 실행되는 메모리상의 Current Page Table과 하드디스크의 Shadow Page Table 이용
- 트랜잭션 시작시점에 Current Page Table과 동일한 Shadow Page Table 생성
- 트랜잭션이 성공적으로 완료될 경우 Shadow Page Table 삭제
- 트랜잭션이 실패할 경우 Shadow Page Table을 Current Page Table로 함

④ 미디어 회복 기법 (Media Recovery)

- 디스크와 같은 비휘발성 저장 장치가 손상되는 장애 발생을 대비한 회복 기법
- 데이터베이스 내용을 백업, 미러링, RAID등을 통해 별도의 물리적 저장장치에 덤프
- 미디어 장애 시 가장 최근 덤프로 복구하고 로그 파일을 참조해 덤프 이후의 작업 Redo

⑤ ARIES 회복 기법

- REDO 중 Repeating history
  - 붕괴가 발생했을 때의 데이터베이스 상태를 복구하기 위하여 붕괴 발생 이전에 수행했던 모든 연산을 다시 한 번 수행
- UNDO 중 Logging
  - UNDO를 할 때에도 로깅을 함으로써 회복을 수행하는 도중에 실패하여 회복을 다시 시작할 때에 이미 완료된 UNDO 연산은 반복하지 않음

Section 2. 데이터 전환

1. ETL(Extraction, Transformation, Loading)

(1) ETL 개념

- 다양한 소스 시스템(source system)으로부터 필요한 데이터를 추출(extract)하여 변환(transform) 작업을 거쳐 타깃 시스템(target system)으로 전송 및 로딩(loading)하는 모든 과정
- 데이터 웨어하우스(DW, Data Warehouse) 구축 시 데이터를 운영 시스템에서 추출하여 가공(변환, 정제)한 후 데이터 웨어하우스에 적재하는 모든 과정
- 데이터에 대해서 필터링, 정렬, 집계, 데이터 조인, 데이터 정리, 중복 제거 및 데이터 유효성 검사 등의 다양한 작업이 포함된다.

(2) ETL 기능

- Extraction (추출)
  - 하나 또는 그 이상의 데이터 원천들로 부터 데이터 획득
- Transformation (변환)
  - 데이터 클렌징, 형식 변환 및 표준화, 통합 또는 다수 애플리케이션에 내장된 비즈니스룰 적용 등
- Load (적재)
  - 변형 단계의 처리가 완료된 데이터를 특정 목표 시스템에 적재

(3) ETL 작업단계

- interface
  - 다양한 DBMS 및 스프레드시트 등 데이터 원천으로부터 데이터를 획득하기 위한 인터페이스 메커니즘 구현
- staging
  - 데이터 원천으로부터 트랜잭션 데이터 획득 작업 수행 후, 획득된 데이터를 스테이징 테이블에 저장
- profiling
  - 스테이징 테이블에서 데이터 특성을 식별하고 품질 측정
- cleansing
  - 다양한 규칙들을 활용해 프로파일링된 데이터의 보정 작업을 수행
- integration
  - 데이터 충돌을 해소하고, 클렌징된 데이터를 통합
- denormalizing
  - 운영 보고서 생성 및 데이터 웨어하우스 또는 데이터 마트에 대한 데이터 적재를 위해 데이터 비정규화 수행