

핵심 요약 노트 | 이론편

정보 처리 기사 실기

# 목 차

1. 소프트웨어 구축 .....	1
2. 데이터베이스 .....	6
3. 운영체제 .....	9
4. 네트워크 .....	11
5. 정보보안 .....	14
6. 기타 용어 .....	16
7. SQL문 활용 .....	18



## ■ 소프트웨어 생명 주기 (Software Development Life Cycle, SDLC)

① 프로젝트 계획 ▶ ② 요구 분석 ▶ ③ 설계 ▶ ④ 구현 ▶ ⑤ 테스트 ▶ ⑥ 유지 보수

	선형 순차적 개발 / 고전적, 전통적 개발 모형 / Step-by-Step
	고객의 need 파악 위해 <b>전본/시제품</b> 을 통해 최종 결과 예측 인터페이스 중심 / 요구사항 변경 용이
	폭포수 + 프로토타입 + <b>위험 분석</b> 기능 추가 (위험 관리/최소화) 점진적 개발 과정 반복 / 정밀하며 유지보수 과정 필요 X ★ 계획 수립 → 위험 분석 → 개발 및 검증 → 고객 평가
	일정한 짧은 주기(Sprint 또는 Iteration) 반복하며 개발 진행 → 고객 요구사항에 유연한 대응 (고객 소통/상호작용 중심) Ex. XP(eXtreme Programming), Scrum, FDD (기능 중심), 린 (LEAN), DSDM (Dynamic System Development Method)

**하향식 설계 (Top-down):** **지향** (순차적) / 최상위 컴포넌트 설계 후 하위 기능 부여  
→ 테스트 초기부터 사용자에게 시스템 구조 제시 가능

**상향식 설계 (Bottom-up):** **지향** / 최하위 모듈 먼저 설계 후 이들을 결합하고 검사  
→ 인터페이스 구조 변경 시 상위 모듈도 같이 변경 필요하여 **기능 추가 어려움**

\* **지향**: 명백한 역할을 가지며 재사용되는 모든 단위 / 인터페이스 통해 접근 가능

※ **지향** (**지향**, XP)

- 고객의 요구사항을 유연하게 대응하기 위해 고객 참여와 신속한 개발 과정을 반복

- 5가지 핵심 가치: 용기 / 단순성 / 의사소통 / 피드백 / 존중

※ **피드백**: 시스템의 상태와 사용자의 지시에 대한 효과를 보여줘서 사용자가 명령에 대한 진행 상황과 표시된 내용을 해석할 수 있게 도와줌

- 기본 원리: 전체 팀 / 소규모 릴리즈 / 테스트 주도 개발 / 지속적인 통합 /

공동 소유권 (Collective ownership) / 짝(pair) 프로그래밍 /

디자인 개선 (리팩토링) / 애자일(Agile) 방법론 활용

상식적 원리 및 경험 추구, **개발 문서보단 소스코드에 중점** (문서화 X)

## ① 프로젝트 계획

### ▶ 하향식 비용 산정 기법

→ 개인적 / 주관적 판단 가능

- 전문가 감정 기법: 조직 내 두 명 이상의 전문가에게 비용 산정을 의뢰하는 기법

- **지향** 기법: 한 명의 조정자와 여러 전문가의 의견을 종합하여 산정

→ '전문가 감정 기법'의 주관적 편견 보완

### ▶ 상향식 비용 산정 기법

- 프로젝트 세부 작업 단위 별로 비용 정산 후 전체 비용을 산정하는 방법

[종류]

. **지향** (**지향**): 코드 라인 총 수 / 생산성 / 개발 참여 인원 등으로 계산

→ 낙관치(a), 비관치(b), 기대치(c)를 측정/예측하여 **비용 산정** =  $\frac{a+4c+b}{6}$

. 개발 단계별 인일 수 (Effort Per Task): LOC 기법 보완 / 생명 주기 각 단계별로 산정

### ▶ 수학적 비용 산정

① **지향** (**지향**): 보ehm (Boehm) 제안 / 원시코드 라인 수 기반

→ 비용 견적 강도 분석 및 비용 견적의 유연성이 높아 널리 통용됨

→ 같은 프로젝트라도 성격에 따라 비용이 다르게 산정

유형	조직형 (Organic)	중,소규모 SW용 / 5만 라인(50KDSI) 이하
	반분리형 (Semi-detached)	30만 라인 (300KDSI) 이하의 트랜잭션 처리 시스템
	내장형 (Embedded)	30만 라인 (300KDSI) 이상의 최대형 규모 SW 관리

② **지향**: SW 생명주기 전 과정에 사용될 **노력의 분포**를 이용한 비용 산정

Rayleigh Norden 곡선의 노력 분포도를 기초로 함

★ **지향**: Rayleigh-Norden 곡선 / Putnam 모형 기초로 개발된 자동화 추정 도구

③ **지향** (**지향**): SW 기능 증대 요인에 **가중치 부여** 후 합산하여 기능점수 산출

→ SW 기능 증대 요인: 자료 입력 (입력 양식) / 정보 출력 (출력 보고서) /

명령어 (사용자 질의수) / 데이터 파일 / 인터페이스

★ **지향**: FP 모형을 기반으로 하여 개발된 자동화 추정 도구

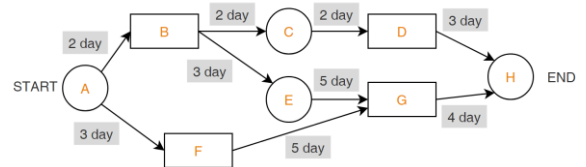
## ■ 개발 일정 산정

- WBS (Work Breakdown Structure): 프로젝트 목표 달성을 위한 활동과 업무를 세분화  
→ 전체 프로젝트를 분할 / 수행 업무 식별, 일정 및 비용

## ▶ 네트워크 차트

	. 프로젝트 작업 상호관계를 네트워크로 표현 . 원 노드(작업)와 간선(화살표)으로 구성 ( <b>불확실한 상황</b> ) → 간선에는 각 작업별 낙관치/기대치/비관치를 기재
	. 미국 Dupont 회사에서 화학공장 유지/관리 위해 개발 . 노드(작업) / 간선(작업 전후 의존 관계) / 박스(이정표) 구성 . 간선(화살표)의 흐름에 따라 작업 진행 ( <b>확실한 상황</b> )

[예시] **지향** 네트워크 - **임계 경로**: **지향**(A-**지향**-H) ← 경로상 가장 오래 걸리는 시간



▶ **지향**: 각 작업의 시작/종료 일정을 막대 바(Bar) 도표를 이용하여 표현  
시간선(Time-line) 차트 (수평 막대 길이 = 작업기간)  
작업 경로는 표현 불가 / 계획 변화에 대한 적응성이 낮음

## ② 요구사항 분석

■ **요구사항**: 어떠한 문제를 해결하기 위해 필요한 조건 및 제약사항을 요구  
소프트웨어 개발/유지 보수 과정에 필요한 기준과 근거 제공

## ▶ 요구사항의 유형

<b>지향</b> 요구사항	실제 시스템 수행에 필요한 기능 관련 요구사항 ex. 금융 시스템은 <b>조회/인출/입금/송금</b> 기능이 있어야 한다.
<b>지향</b> 요구사항	성능, 보안, 품질, 안정성 등 실제 수행에 보조적인 요구사항 Ex. 모든 화면이 3초 이내에 사용자에게 보여야 한다.

## ▶ 요구사항 개발 프로세스 ★순서 중요

① <b>도출/추출</b>	이해관계자들이 모여 요구사항 정의 (식별하고 이해하는 과정) Ex. 인터뷰, 설문, 브레인스토밍, 청취, 프로토타이핑, 유스케이스
② <b>지향</b>	사용자 요구사항에 타당성 조사 / 비용 및 일정에 대한 제약 설정 Ex. 관찰, 개발 모델링, 정형 분석, 요구사항 정의 문서화
③ <b>지향</b>	요구사항 체계적 분석 후 승인가능하도록 문서화
④ <b>지향/지향</b>	요구사항 명세서가 정확하고 완전하게 작성되었는지 검토

## ▶ 요구사항 분석 도구

<b>요구사항 분석</b>	. SADT: SoftTech사에서 개발 / 구조적 분석 및 설계분석 . SREM: 실시간 처리 SW 시스템에서 요구사항 명확한 기술 목적 . PSL/PSA: 문제 기술언어 및 요구사항 분석 보고서 출력 . TAGS: 시스템 공학 방법 응용에 대한 자동 접근 방법
<b>지향</b>	하향식 설계 방식 / 가시적, 총체적, 세부적 다이어그램으로 구성 기능과 자료의 의존 관계 동시 표현 / 이해 쉽고 유지보수 간단

## ▶ 구조적 분석 모델

- 데이터/**지향** (**지향**, **지향**):

프로세스 (Process) / 자료 흐름 (Flow) / 자료 저장소 (Data store) / 단말 (Terminator)

**지향** **지향** **지향** **지향**

→ 구조적 분석 기법에 이용 / 시간 흐름 명확한 표현 불가 / 버블(bubble) 차트

- **지향** (**지향**): 자료 흐름도에 기재된 모든 자료의 상세 정의/설명

=	정의	[ ]	택일 / 선택	( )	생략
+	구성	**	설명 / 주석	{ }	반복

- 소단위 명세서 / 개체 관계도 (ERD, Entity Relationship Diagram) / 상태 전이도



## ※ 객체지향 분석 모델

- . Booch (부치): 미시적, 거시적 개발 프로세스를 모두 사용 (클래스/객체 분석 및 식별)
- . : Use case를 사용 (사용자, 외부 시스템이 시스템과 상호작용)
- . Coad-Yourdon: E-R 다이어그램 사용 / 객체의 행위 모델링
- . : 분석과 설계 구분 없으며 고객 명세서 평가 후 설계 작업까지 연속 수행
- . : 가장 일반적으로 사용, 객체/동적/기능 모델로 구분
  - 모델링 → 객체 다이어그램 / 객체들 간의 관계 규정/정의
  - 모델링 → 상태 다이어그램 / 시스템 동적인 행위 기술
  - 모델링 → 자료 흐름도(DFD) / 다수의 프로세스들 간의 처리 과정 표현

## ▶ 요구사항 명세

명세	수학적 원리 / 정확하고 간결한 요구사항 표현 가능 어려운 표기법으로 사용자 이해 어려움 (VDM, Z, Petri-net, CSP)
명세	자연어, 그림 중심 / 쉬운 자연어 사용으로 의사소통 용이하나 작성자에 따라 모호한 내용으로 일관성 떨어짐 (FSM, Decision Table, E-R 모델, State Chart)

## ③ 소프트웨어 설계

### ■ 소프트웨어 설계 원리

- 분할과 정복**: 여러 개의 작은 서브시스템으로 나눠서 각각을 완성
- (): 시스템 기능을 모듈 단위로 분류하여 성능/재사용성 향상
  - 모듈 크기 ▲ → 모듈 개수 ▼ → 모듈간 통합비용 ▼ (but, 모듈 당 개발 비용 ▲)
  - 모듈 크기 ▼ → 모듈 개수 ▲ → 모듈간 통합비용 ▲
- (): 불필요한 부분은 생각하고 필요한 부분만 강조해 모델화
  - 문제의 포괄적인 개념을 설계 후 차례로 세분화하여 구체화 진행
  - ① **과정 추상화**: 자세한 수행 과정 정의의 X, 전반적인 흐름만 파악가능하게 설계
  - ② **데이터(자료) 추상화**: 데이터의 세부적 속성/용도 정의의 X, 데이터 구조를 표현
  - ③ **제어 추상화**: 이벤트 발생의 정확한 절차/방법 정의의 X, 대표 가능한 표현으로 대체
- 단계적 분해 (Stepwise refinement)**: 하향식 설계 전략 (by Niklaus Wirth)
  - 추상화의 반복에 의한 세분화 / 세부 내역은 가능한 뒤로 미루어 진행
- ():
  - 한 모듈 내부에 포함된 절차/자료 관련 정보가 숨겨져 다른 모듈의 접근/변경 불가
  - 모듈을 독립적으로 수행할 수 있어 요구사항에 따라 수정/시험/유지보수가 용이함

### ■ 아키텍처 패턴

	시스템을 계층으로 구분/구성하는 고전적 방식 (OSI 참조 모델)
	하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성 클라이언트와 서버는 요청/응답 제의 시 서로 독립적 * 컴포넌트(Component): 독립적 업무/기능 수행 위한 실행코드 기반 모듈
	데이터 스트림 절차의 각 단계를 필터 컴포넌트로 캡슐화 후 데이터 전송 / 재사용 및 확장 용이 / 필터 컴포넌트 재배치 가능 단방향으로 흐르며, 필터 이동 시 오버헤드 발생 변환, 버퍼링, 동기화 적용 (ex. UNIX 셸 - Shell)
	모델 (Model): 서브시스템의 핵심 기능 및 데이터 보관 뷰 (View): 사용자에게 정보 표시 컨트롤러 (Controller): 사용자로부터 받은 입력 처리 → 각 부분은 개별 컴포넌트로 분리되어 서로 영향 X → 하나의 모델 대상 다수 뷰 생성 ▶ 대화형 애플리케이션에 적합
Master-slave	마스터에서 슬레이브 컴포넌트로 작업 분할/부리/배포 후 슬레이브에서 처리된 결과물을 다시 돌려 받음 (병렬 컴퓨팅)
	컴포넌트와 사용자를 연결 (분산 환경 시스템)
Peer-to-peer	피어를 한 컴포넌트로 산정 후 각 피어는 클라이언트가 될 수도, 서버가 될 수도 있음 (펄티스레드 방식)
	소스가 특정 채널에 이벤트 메시지를 발행 시 해당 채널을 구독한 리스너들이 메시지를 받아 이벤트를 처리함
Blackboard	컴포넌트들이 검색을 통해 블랙보드에서 원하는 데이터 찾을
	특정 언어로 작성된 프로그램 코드를 해석하는 컴포넌트 설계

- () → 구성요소 : 사물, 관계, 다이어그램

- 고객/개발자 간 원활한 의사소통을 위해 표준화된 대표적 객체지향 모델링 언어
- Rumbaugh, Booch, Jacobson 등 객체지향 방법론의 장점 통합

- ※ : 클래스/컴포넌트가 구현해야하는 오퍼레이션 세트를 정의하는 모델 요소

1) 사물 (Things): 구조(개념, 물리적 요소) / 행동 / 그룹 / 주해(부가적 설명, 제약조건)

2) 관계 (Relationship)

관계 (Association)	2개 이상의 사물이 서로 관련
관계 (Aggregation)	하나의 사물이 다른 사물에 포함 (전체-부분 관계)
관계 (Composition)	집합 관계 내 한 사물의 변화가 다른 사물에게 영향
관계 (Generalization)	한 사물이 다른 사물에 비해 일반/구체적인지 표현 (한 클래스가 다른 클래스를 포함하는 상위 개념일 때)
관계 (Dependency)	사물 간 서로에게 영향을 주는 관계 (한 클래스가 다른 클래스의 기능을 사용할 때)
관계 (Realization)	한 객체가 다른 객체에게 오퍼레이션을 수행하도록 지정 / 서로를 그룹화할 수 있는 관계

## 3) 다이어그램 (Diagram)

구조, 정적 다이어그램 (클래스/컴포넌트)		클래스 사이의 관계 및 속성 표현
		인스턴스를 객체와 객체 사이의 관계로 표현
		구현 모델인 컴포넌트 간의 관계 표현
		물리적 요소(HW/SW)의 위치/구조 표현
		클래스 및 컴포넌트의 복합체 내부 구조 표현
행위, 동적 다이어그램 (유시커상활타상)		UML의 다양한 모델요소를 그룹화하여 묶음
		사용자의 요구를 분석 (사용자 관점) → 사용자(Actor) + 사용 사례 (Use Case)
		시스템/객체들이 주고받는 메시지 표현 → 구성항목: 액터* / 객체 / 생명선 / 메시지 제어 삼각형
		객체들이 주고받는 메시지와 객체 간의 연관관계까지 표현
		다른 객체와의 상호작용에 따라 상태가 어떻게 변화하는지 표현
		객체의 처리 로직 및 조건에 따른 처리의 흐름을 순서에 따라 표현
		객체 상태 변화와 시간 제약 명시적으로 표현
		상호작용 다이어그램 간 제어 흐름 표현

## ▶ UI 설계 (User Interface): , (사용자의 목적 달성), ,

- 설계 지침: 사용자 중심 / 일관성 / 단순성 / 결과 예측 / 가시성 / 표준화 / 접근성 / 명확성 / 오류 발생 해결

- (, , , , )  
텍스트      그래픽      말/행동      음성      사물과 사용자 상호작용

### - UI 설계 도구

	기획 초기 단계에 대략적인 레이아웃을 설계
	최종적인 산출문서 (와이어프레임-UI, 콘텐츠 구성, 프로세스 등)
	와이어프레임 / 스토리보드에 인터랙션 적용 실제 구현된 것처럼 테스트가 가능한 동적인 형태 모형 *인터랙션: UI를 통해 시스템을 사용하는 일련의 상호작용 (동적효과)
	실제 화면과 유사한 정적인 형태 모형
	사용자 측면 요구사항 및 목표를 다이어그램으로 표현



## ④ 소프트웨어 구현

## ■ 소프트웨어 개발 프레임워크

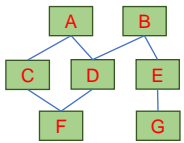
- 개발해야 할 애플리케이션 일부분이 이미 내장된 클래스 라이브러리에 구현
- 동일 로직 반복 최소화 / 재사용성 확대 / 생산성 및 유지보수성 향상
- 모듈화 / 재사용성 / 확장성 / 제어의 역할 (프레임워크가 애플리케이션 흐름 제어)

프레임워크 (Framework)	. JAVA 플랫폼을 위한 오픈 소스 경량형 프레임워크 . 동적 웹 사이트 개발 / 전자정부 표준 프레임워크
전자정부 프레임워크	. 우리나라 공공부문 정보화 사업 시 효율적인 정보 시스템 구축 지원 위해 필요한 기능/아키텍처를 제공
프레임워크 (Framework)	. MS에서 개발한 Windows 프로그램 개발 . 공통 언어 런타임(CLR)이라는 가상 머신 상에서 작동

## ※ 프레임워크

- 소프트웨어 간 인터페이스 (서로 다른 소프트웨어/서비스 간 상호 작용 용이)
- 운영체제 및 프로그래밍 언어의 기능을 프로그램에서 사용 가능하도록 구현
- 개발 비용 저감 / 중복 작업 최소화 / 유지 관리 용이 / 비즈니스 확장

- ▶ 프레임워크 : 자신을 사용하는 모듈의 수 → 팬인이 높으면 재사용 측면 설계 우수  
but, 단일 장애점 발생 가능성 높아 집중적인 관리/테스트 필요
- ▶ 프레임워크 : 자신이 사용하는 모듈의 수 → 불필요한 호출 가능성 (단순화 필요)



- Q. D의 팬인의 개수는?
- Q. D의 팬아웃의 개수는?

## ■ 응집도 (Cohesion)

- 개별 모듈이 독립적인 기능으로 정의되어 있는 정도 / 응집도 ▲ → 품질 ▲

## 높은 응집도

	모듈 내부의 모든 기능 요소가 단일 문제와 연관되어 수행
	모듈 내 출력 데이터를 다음 활동의 입력 데이터로 사용
	동일한 입/출력을 사용하여 서로 다른 기능을 수행
	모듈 내 구성 요소들이 다수 관련 기능을 순차적으로 수행
	특정 시간 내 처리되는 기능을 모아 하나의 모듈로 작성
	유사한 성격의 처리 요소들로 하나의 모듈이 형성
	각 구성 요소들이 서로 관련없는 요소로만 구성

## 낮은 응집도

## ■ 결합도 (Coupling) - 개별 모듈 간 상호 의존하는 정도 / 결합도 ▼ → 품질 ▲

## 높은 결합도

	한 모듈이 다른 모듈의 내부 기능 및 자료를 직접 참조/수정
	공유되는 공통 데이터를 여러 모듈이 사용 (전역 변수 참조)
	한 모듈에서 선언한 데이터를 외부의 다른 모듈에서 참조
	. 한 모듈이 다른 모듈의 상세한 처리 절차를 알고 있어 이를 통제하는 경우나 처리 기능이 두 모듈에 분리되어 설계 . 처리 대상 값뿐만 아니라 처리 방식의 제어 요소도 전달
	. 두 모듈이 동일한 자료 구조(배열, 오브젝트)를 조회 . 자료 구조 및 포맷 변화 시 조회하고 있는 모든 모듈에 영향
	모듈 간의 인터페이스가 자료 요소로만 구성

## 낮은 결합도

- ※ 모듈의 독립성 : 모듈 간 과도한 상호작용을 막고 하나의 독립적 기능만을 수행
- 높은 독립성 : 높은 응집도 / 낮은 결합도

## ■ 객체 지향 (Object-oriented) : 객체와 속성, 클래스와 멤버, 전체와 부분으로 나눠 분석

	고유 식별자 / 하나의 독립된 존재 / 일정한 기억장소 보유 상태(state) = 객체가 가질 수 있는 조건, 속성 값에 의해 정의 행위(연산, Method) = 객체가 반응할 수 있는 메시지 집합
	공통 속성과 연산(행위)을 갖는 객체들의 집합 / 데이터 추상화 단위 ※ 인스턴스 (Instance) : 클래스에 속한 각각의 객체 ※ Operation : 클래스의 동작 / 객체에 대해 적용될 메서드 정의
	데이터와 데이터 처리 함수를 하나로 묶음 세부 내용 은폐(정보 은닉) → 외부 접근 제한 결합도 낮음 / 재사용 용이 / 인터페이스 단순 / 오류 파급효과 낮음
	상위 클래스의 속성과 연산을 하위 클래스가 물려받는 것 ※ 다중 상속 : 단일 클래스가 두 개 이상의 상위 클래스로부터 상속
	하나의 메시지에 각 객체 별 고유 특성에 따라 여러 형태의 응답

- ( ) : 상위클래스로부터 상속받은 메서드를 하위클래스에서 재정의  
→ 단, 메서드 이름 / 매개변수 / 반환 타입은 동일해야 함

- ( ) : 메서드 이름은 동일하나 매개변수 개수 또는 타입을 다르게 지정

## ■ 객체지향 설계 5대 원칙 (SOLID)

- . 원칙 : 원칙  
→ 모든 클래스/객체는 하나의 책임만 / 완전한 캡슐화
- . 의 원칙 : 의 원칙  
→ 확장에는 Open하고, 수정에는 Close되어야 한다.
- . 원칙 : 원칙  
→ 상위 클래스의 행동 규약을 하위 클래스가 위반하면 안된다.
- . 원칙 : 원칙  
→ 클라이언트가 비사용 메서드에 의존하지 않아야 한다.
- . 원칙 : 원칙  
→ 의존 관계 수립 시 변화하기 어려운 것에 의존해야 한다.  
추상성이 높은 상위 클래스

## ■ 디자인 패턴 : GoF (Gang of Four) 처음 제안하여 구체화

- 서브시스템에 속하는 컴포넌트들과 그 관계를 설계하기 위한 참조 모델
- 객체 지향 프로그래밍 설계 시 자주 발생하는 문제에 대한 반복적 해결 방법
- c) 아키텍처 패턴 : 전체 시스템의 구조를 설계

## [ 생성 패턴 ]

	구체적인 클래스에 의존하지 않고, 서로 연관되거나 의존적인 객체들이 조합된 인터페이스 제공
	객체 생성 단계를 캡슐화/분리하여 객체를 조립하여 생성 → 동일한 객체 생성 절차에서 서로 다른 표현 결과를 제공
	상위클래스에서 객체 생성 인터페이스를 정의하지만, 인스턴스를 만드는 클래스는 서브 클래스에서 결정하도록 분리
	원본/원형 객체를 복제하는 방식으로 객체를 생성
	클래스에서 하나의 객체만 생성 가능하며, 해당 객체를 어디서든 참조할 수 있지만 여러 프로세스가 동시에 참조는 불가

## [ 구조 패턴 ]

	비호환 인터페이스에 호환성 부여하도록 변환
	구현부에서 추상층을 분리 후 각자 독립적으로 변형/확장 가능
	트리 구조로 부분/전체 계층 표현, 복합/단일 객체를 구분없이 사용
	상속 사용없이 객체 간 결합을 통해 객체 기능을 동적으로 추가/확장
	상위에 인터페이스 구성하여 서브클래스의 기능을 복잡하게 표현하지 않고 단순한 인터페이스로 구현
	인스턴스를 공유하여 메모리 절약 (클래스 경량화)
	접근이 힘든 객체를 연결하는 인터페이스 역할 (대리 객체 수행)





## [ 행위 패턴 ]

	처리가능한 객체가 둘 이상 존재하여 한 객체 내 처리 불가 시 다음 객체로 이관
	요청 명령어들을 추상/구체 클래스로 분리 후 단순화/캡슐화
	언어에 문법 표현 정의
	컬렉션 객체의 내부 구조를 숨기고 요소들을 순차적으로 접근 → 접근이 빈번한 객체에 대해 동일 인터페이스 사용 가능
	객체들간 복잡한 상호작용을 캡슐화하여 객체로 정의 후 중재
	객체를 이전의 특정 시점의 상태로 저장하고 복원 (캡슐화 유지)
	한 객체 상태 변화 시 상속되어 있는 객체들에 변화 전달
	객체의 상태에 따라 동일한 동작을 다르게 처리
	동일 계열 알고리즘을 개별적으로 캡슐화하여 상호 교환
	여러 클래스에서 공통 사용 메서드를 상위 클래스에서 정의하고, 하위 클래스마다 다르게 구현해야하는 세부 사항을 개별 구현
	각 클래스 데이터 구조로부터 처리/연산 기능을 분리하여 별도의 클래스를 만들고, 해당 클래스 메서드가 각 클래스를 돌아다니며 특정 작업을 수행 → 객체 구조 변경 X / 새로운 연산 기능만 추가

## ⑤ 소프트웨어 테스트

## ■ 애플리케이션 테스트 기본 원리

- 테스트는 기본적으로 결함이 존재함을 밝히는 것 (무결함을 증명할 수는 없음)  
→ 완벽한 테스트는 근원적으로 불가능 (무한 경로, 무한 입력 불가)
- 결함 집중 : 법칙 - 20%의 모듈에서 전체 결함 80% 발생
- 패러독스 : 동일한 테스트 케이스에 의한 반복 테스트는 새로운 버그 발견 X
- 의 궤변 : 결함이 없다 해도 사용자의 요구사항 미충족 시 품질 저하
- 의 법칙 : 지연되는 프로젝트에 입력 추가 투입 시 더 지연

## ■ 애플리케이션 테스트 분류

## ① 프로그램 실행 여부

테스트	프로그램 실행 X / 명세서, 소스 코드만 분석 Ex. 동료 검토, 워크 스루, 인스펙션, 코드 검사
테스트	프로그램 실행 후 오류 검사 ex. 화이트/블랙박스 테스트

## ② 테스트 기반 테스트

명세 기반	사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 구현하는지 확인 → ex. 동등 분할 / 경계값 분석 (블랙박스)
구조 기반	SW 내부 논리 흐름에 따라 테스트 케이스 작성/확인 ex. 구문 기반 / 결정 기반 / 조건 기반 (화이트박스)
경험 기반	테스터의 경험을 기반으로 수행 ex. 에러 추정, 체크리스트, 탐색적 테스트

## ③ 목적 기반 테스트

	시스템에 인위적 결함 부여 후 정상으로 회복되는 과정 확인
	외부 불법 침입으로부터 시스템을 보호할 수 있는지 확인
	과부하시 SW 정상 구동 여부 확인
성능 (Performance)	실시간 성능 및 전체적인 효율성 진단 (응답 시간, 업무 처리량)
구조 (Structure)	SW 내부 논리적 경로 및 소스 코드 복잡도 평가
	SW 내 변경 또는 수정된 코드에 새로운 결함이 없음 확인
	변경 및 기존 SW에 동일한 데이터 입력 후 결과 비교

## ④ 시각(관점) 기반 테스트

	개발자의 시각에서 제품의 생산 과정 테스트 Ex. 단위/통합/시스템 테스트
	사용자의 시각에서 생산된 제품의 결과 테스트 Ex. 인수 테스트 (알파 / 베타)

## ■ 테스트 ( Test)

- 모듈 안의 내용(작동) 직접 볼 수 있으며, 내부의 **논리적인** 모든 경로를 테스트
- 소스 코드의 모든 문장을 한 번 이상 수행 / 논리적 경로 점검 (선택, 반복 수행)
- 테스트 데이터 선택하기 위해 **검증 기준 커버리지(Coverage)** 정함

## [ 화이트박스 테스트 검증 기준 ]

커버리지 Coverage)	프로그램 내 모든 명령문을 적어도 한 번 수행
커버리지 Coverage)	프로그램 내 전체 결정의 분기가 적어도 한 번은 참/거짓 결과 수행
커버리지 Coverage)	결정 명령문 내의 각 개별 조건식이 적어도 한 번은 참/거짓 결과 수행
커버리지 	전체 조건식뿐만 아니라 개별 조건식도 참 한 번 이상, 거짓 한 번 이상 결과 수행
변경   커버리지 (MC/DC)	Modified Condition/Decision Coverage 각 개별조건식이 독립적으로 전체 조건식의 결과에 영향
커버리지 	결정 포인트 내 모든 개별조건식의 모든 가능한 논리적 조합을 고려하여 100% 커버리지 보장

## [ 화이트박스 테스트 종류 ]

검사 Testing)	. 대표적 화이트박스 테스트 기법 (동적 테스트) . 테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 해주는 테스트 기법 . 측정 결과는 실행 경로의 기초를 정의하는 지침으로 사용
검사 Testing)	. 조건 검사 : 프로그램 모듈 내 논리적 조건 테스트 . 루프 검사 : 프로그램 반복(Loop) 구조 테스트 . 자료 흐름 검사 : 변수의 정의와 변수 사용의 위치 테스트

## ■ 테스트 ( Test)

- 모듈 내부의 내용 알 수 없음 / 소프트웨어 인터페이스에서 실시되는 테스트
- SW 각 기능이 완전히 작동되는 것을 입증하는 테스트로 '**기능 테스트**' 라고 함

## [ 블랙박스 테스트 종류 ]

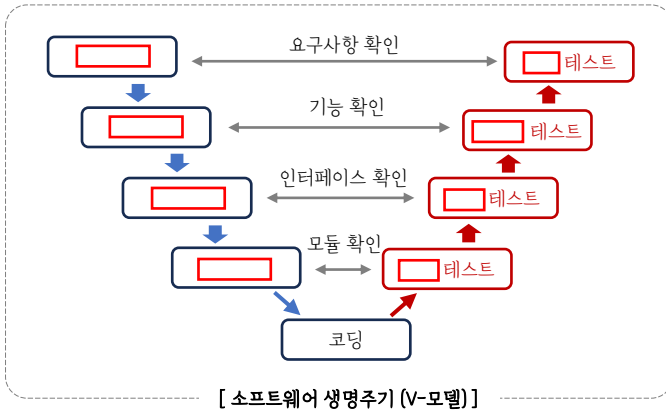
검사 	. 프로그램 입력 조건에 타당한 입력 자료와 타당하지 않은 입력 자료의 개수를 균등하게 진행
분석 	. 입력 조건의 경계값을 테스트 케이스로 선정 (ex. 범위 구간의 양끝 : ~이상/이하/초과/미만)
검사 	. 입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석 후 효용성이 높은 테스트 케이스 선정
검사 	. 과거의 경험이나 확인자의 감각으로 테스트 진행
검사 	. 여러 버전의 프로그램에 동일한 결과가 출력되는지 확인

## ■ 요구사항 검증 (Requirements Verification)

- 실제로 고객이 원하는 시스템을 제대로 정의했는지 점검하는 과정
- 시스템 개발 완료 후 문제 발생 시 막대한 재작업 비용 발생하기에 검증 중요 (실제 요구사항 반영 여부 / 문서 상 요구사항이 서로 상충되지 않는지 점검)
- ① 동료 검토 (Peer review) : 작성자가 내용 설명 후 동료들이 결함 검토
- ② : 요구사항 명세서 미리 배포 후 짧은 검토 회의 진행
- ③ 인스펙션 (Inspection) : 작성자 제외한 다른 전문가들이 결함 검토

## ※ 검증 체크리스트 : 기능성 (Functionality) / 완전성 (Completeness) /

일관성 (Consistency) / 명확성 (Unambiguity) / 검증 가능성 (Verifiability) / 추적 가능성 (Traceability) / 변경 용이성 (Easily Changeable)



## ■ 개발단계에 따른 애플리케이션 테스트

→ 소프트웨어를 이루는 기본 단위 (독립적 기능)

① **테스트 (test)**: 최소 단위(모듈/컴포넌트) 기반 테스트

주로 구조 기반 테스트 진행 / 기능성 테스트 최우선

② **테스트 (test)**: 인터페이스 간 시스템이 정상 실행되는지 확인

- 단위 테스트 후 모듈을 통합하는 과정에서 발생하는 오류 및 결함을 찾는 테스트 기반

<b>상위 모듈에서 하위 모듈 방향으로 통합</b> ( )	. 상위 모듈에서 하위 모듈 방향으로 통합 . 깊이 우선(Depth first) 통합법 / 넓이 우선(Breadth first) 통합법 . 초기부터 사용자에게 시스템 구조 보여줌 . : 모듈의 기능을 단순히 수행하는 도구 (시험용 모듈)
<b>하위 모듈에서 상위 모듈 방향으로 통합</b> ( )	. 하위 모듈에서 상위 모듈 방향으로 통합 . 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 클러스터(Cluster)와 사용 / 스텝(Stub) 미사용

③ **테스트 (test)**: 개발된 SW의 컴퓨터 시스템 내 작동여부 점검

실제 사용 환경과 유사한 테스트 환경 ▶ 기능적 및 비기능적 테스트 구분  
블랙박스 화이트박스

④ **테스트 (test)**: 사용자의 요구사항 충족 여부 확인

- 테스트: 통제된 환경에서 사용자가 개발자와 함께 확인

- 테스트: 통제되지 않은 환경에서 개발자 없이 여러 명의 사용자가 검증

## ■ 테스트 관련 기타 용어

<b>테스트 (Test)</b>	. 테스트 케이스 적용/동작 순서에 따라 여러 테스트 케이스를 묶은 집합
<b>테스트 (Test)</b>	. 테스트 결과의 참/거짓 판단 위해 사전에 정의된 참 값을 대입
<b>오라클 (Oracle)</b>	. 모든 테스트 케이스의 입력값에 기대 결과 제공
<b>오라클 (Oracle)</b>	. 특정 테스트 케이스 입력값에 기대 결과 제공
<b>오라클 (Oracle)</b>	. 특정 테스트 케이스 입력값에 기대 결과 제공 + 나머지 입력값에 대해선 추정 결과 제공
<b>오라클 (Oracle)</b>	. 테스트 케이스의 수행 전/후의 결과값 동일 여부 확인

## ■ 테스트 하네스 (Test Harness)

<b>테스트 (Test)</b>	시험 대상의 하위 모듈 호출 / 모듈 테스트 수행 후의 결과 도출 → 삼각식 통합 테스트에서 사용
<b>테스트 (Test)</b>	제어 모듈이 호출하는 하위 모듈의 역할 단순 수행 → 하향식 테스트에 사용
<b>테스트 (Test)</b>	시스템에 사용되는 테스트 케이스의 집합 (컴포넌트 / 모듈)
<b>테스트 (Test)</b>	사용자의 요구사항 준수 여부 확인 위해 설계된 테스트 항목 명세서 (입력값, 실행 조건, 기대 결과 등)
<b>테스트 (Test)</b>	자동화된 테스트 실행 절차에 대한 명세서
<b>( )</b>	사용자의 행위 조건부 입력 시 계획된 행위를 수행하는 객체

## ⑥ 소프트웨어 유지 보수

### ■ 애플리케이션 성능 개선

- 성능 분석 지표: 처리량 / 응답 시간 / 경과 시간 / 자원 사용률

- 소스코드 최적화를 통한 코드 스멜 제거 후 품질 개선

※ ( ) : 소스코드 내 존재하는 잠재적 문제점

예시) 중복 코드 / 긴 메서드 / 큰 클래스 / 클래스 동시 수정

→ 스파게티 코드 : 소스코드 로직이 복잡하게 얽혀있는 구조

→ 외계인 코드 : 오래되어 유지보수 작업이 어려운 코드

▶ 클린 코드 작성 원칙 : 가독성 / 단순성 / 의존성 배제 / 중복성 최소화 / 추상화

### ▶ 소스 코드 품질 분석 도구

<b>분석 도구 (Analysis)</b>	프로그램 실행 없이 코딩 표준/스타일/결함 등을 분석 → PMD, Checkstyle, SonarQube, Cppcheck, Ccm, Cobertura
<b>분석 도구 (Analysis)</b>	프로그램 실행하여 코드 내 메모리 누수 및 스레드 결함 발견 → Avalanche, Valgrind, Valance

### ■ 소프트웨어 관리 (SCM: Software Management)

- 개발 과정에서 SW 변경사항을 관리하기 위한 일련의 활동 / 개발 전체 단계 적용

- 중요성: 변경사항 추적/통제, 무절제한 변경 방지, 개발 진행 이력 확인

- 형상 관리 역할: 배포본 관리 용이 / 불필요한 소스 수정 제한 / 여러 개발자 동시 개발

형상 ( )	관리 대상에 이름/번호 부여 후 계층 구조로 구분 → 수정/추적 용이
형상 ( )	식별된 형상 항목에 대한 변경 요구 검토 (기준선 반영될 수 있게)
형상 ( )	기준선(Base line)의 무결성 평가 위해 확인/검증/검열 과정 진행
형상 ( )	형상 식별/통제/감사 작업 결과를 기록/관리하고 보고서 작성

※ 제품 SW의 형상 관리 역할: 배포본 관리 용유 / 소스 수정 제한

동일 프로젝트에 대해 여러 개발자 참여 후 동시 개발 가능

### ■ 소프트웨어 버전 관리 도구

공유 폴더	클라이언트/서버	분산 저장소
공유 폴더에 복사	서버에서 버전 일괄 관리	원격 → 지역 저장소
RCS, SCCS, PVCS, QVCS	CVS, SVN, CVSNT, CMVC	Git, GNU arch, DCVS, Bitkeeper, Bazaar

### ※ 형상관리 도구

( )	서버/클라이언트 구성, 다수의 인원이 동시 버전 관리 가능
( )	CVS 개선 툴 / 모든 개발은 trunk 디렉터리에서 수행 Commit 수행 시 revision 1씩 증가
( )	서버(원격) 저장소와 개발자(지역) 저장소가 독립적 → Commit 실수 발생해도 서버에 영향 없음 (분산된 P2P 모델)

### ■ ISO 12207 - 소프트웨어 관련 생명주기

<b>생명주기</b>	획득, 공급, 개발, 운영, 유지보수 프로세스
<b>생명주기</b>	문서화, 형상관리, 품질보증, 검증, 확인, 합동검토, 감사
<b>생명주기</b>	관리, 기반구조, 개선, 교육훈련

■ ( ) : SW 개발 조직의 업무 능력 평가 모델

1) 초기	프로세스 X	작업자 능력에 따라 성공 여부 결정
2) 관리	규칙화 프로세스	특정한 프로젝트 내의 프로젝트 정의/수행
3) 정의	표준화 프로세스	조직의 표준 프로세스 활용 업무 수행
4) 정량적 관리	예측 가능 프로세스	프로젝트 정량적 관리/통제
5) 최적화	지속 개선 프로세스	프로세스 역량 향상 위한 지속적인 개선



## ■ 데이터베이스 (DBMS, DataBase Management System)

- 특정 조직 내 필요한 데이터들의 모임, 공용으로 소유/유지/이용하는 공용 데이터

## ■ 스키마 (Schema)

<b>스키마</b> (서브 스키마)	. 사용자 관점의 스키마 → 하나의 DB에 여러 개의 외부 스키마 존재 . 사용자, 프로그램마다 다양한 형태의 논리적 구조로 존재
<b>스키마</b>	. 사용자와 DB 관리자 관점의 스키마 / DB의 전체적인 논리적 구조 . 일반적으로 하나의 DB에는 하나의 개념 스키마가 존재 → 데이터 개체/관계/제약조건/접근권한/무결성 규칙 명세
<b>스키마</b>	. DB 설계자/개발자 관점의 스키마 . 개념 스키마를 <b>물리적 저장장치</b> 에 구현하는 방법을 정의 → 물리적 구조 / 내부 레코드의 물리적 순서

## ■ 데이터 독립성

- 논리적 독립성: 응용 프로그램 영향없이 데이터베이스 논리 구조 변경
- 물리적 독립성: 응용 프로그램 및 논리 구조 영향없이 데이터의 물리적 구조 변경

## ■ 데이터 언어

<b>(Data Language)</b>	데이터베이스 구조/제약 조건 정의
<b>(Data ~)</b>	데이터 처리/조작에 사용되는 언어
<b>(Data ~)</b>	데이터의 보안, 권한, 무결성, 권한 검사, 병행 제어를 위한 언어

## ■ 데이터베이스 설계 순서 ← ★순서 외우기

<b>요구조건 분석</b>	데이터베이스의 사용 용도 / 요구사항 / 요구 조건 명세서 작성
<b>모델</b>	. 현실 세계의 인식을 추상적 개념 세계로 표현 . 독립적인 개념 스키마 모델링 / <b>트랜잭션 모델링</b> / E-R 모델
<b>모델</b>	. 개념적 모델을 컴퓨터가 처리할 수 있는 구조로 표현 (관계 모델) . 종속적인 논리 스키마 설계 / <b>트랜잭션 인터페이스 설계</b> / 정규화 . 논리적 데이터 베이스 구조로 Mapping / 스키마의 평가 및 정제
<b>모델</b>	. 데이터의 실제 저장 방법 및 접근 경로 표현 (레코드 형식/순서) . 목표 DBMS 종속적인 물리적 구조 데이터로 변환 / <b>반정규화</b> . 오브젝트, 접근방법, 인덱스, 뷰 용량 설계 / <b>트랜잭션 작성</b> → <b>저장 레코드의 형식/순서/접근 경로 설계</b>
<b>구현</b>	DBMS에서 SQL로 작성한 명령문 실행 후 데이터베이스 실제 생성

※ 데이터 모델링: 현실 내 복잡한 데이터 구조를 단순/추상화하여 체계적으로 표현

## ■ 데이터 모델 구성 요소

- **( )**: 논리적 개체 간 관계, 데이터 구조, 정적 성질 표현
- **( )**: 실제 데이터를 처리하는 작업 명세로 데이터베이스 조작 도구
- **조건 ( )**: DB에 저장될 수 있는 실제 데이터의 논리적 제약 조건

## ■ 개체-관계 모델 (E-R Model / Entity-Relation Model)

<b>( )</b>	독립적이고 구별 가능한 모든 것 . 개체 타입: 개체를 구성하는 속성들의 집합 . 개체 인스턴스: 실제 값을 가지면서 실체화된 개체 . 개체 세트: 개체 인스턴스들의 집합
<b>( )</b>	개체가 가진 고유의 특성 / 가장 작은 논리적 단위 (원)
<b>( )</b>	두 개체 간의 의미 있는 연관성 및 연결 구조 (마름모) → 일대일 (1:1) / 일대다 (1:N) / 다대다 (N:M)

## ■ E-R 다이어그램 표기법

	<b>집합</b>		<b>속성</b>
	<b>집합</b>		<b>연결</b>

※ 점선: **( )** 연결

## ■ 관계형 데이터베이스 용어

- 릴레이션(Relation)이라는 표(Table)로 표현
- **( )** = 열 = 필드 / **( )** = 행 = 레코드
- **( )**: 속성의 개수 / **( )**: 튜플의 개수
- **( )**: 하나의 속성에서 취할 수 있는 원자값들의 집합 (ex. 성별: 남/여)  
Atomic value
- **릴레이션 ( )**: 릴레이션에 실제로 저장된 데이터 집합

## ※ 릴레이션 특징

- 튜플과 속성은 **( )**하며 순서는 무의미함
- 튜플은 삽입/삭제 등에 의해 계속 변함 / 튜플은 서로 상이한 값을 가짐
- 속성의 값은 분해 불가 (원자성) / 동일할 수 있음
- 튜플을 식별하기 위해 속성(필드)의 일부를 Key로 설정
- 속성은 Null 값을 가질 수 **( )**, 기본키에 해당되는 속성은 Null 값을 가질 수 **( )**

## ■ 키 (Key)

<b>( )</b>	. 기본키로 사용 가능한 속성 / 모든 릴레이션에는 후보키 존재 . 모든 튜플에 대해 <b>유일성/최소성 만족시켜야 함</b> ※ <b>유일성</b> : 하나의 키 값으로 하나의 튜플 유일하게 식별 ※ <b>최소성</b> : 모든 레코드 식별하는데 최소한의 속성으로만 구성
<b>( )</b>	. 후보키 중에서 선택되어, <b>중복된 값과 NULL 값 가질 수 없음</b> . 유일성/최소성 만족하며 튜플 식별하기 위해 반드시 필요한 키
<b>( )</b>	. 후보키가 둘 이상일 때 기본키를 제외한 나머지 후보키
<b>( )</b>	. 한 릴레이션 내 속성들의 집합으로 구성된 키 . 모든 튜플에 대해 <b>유일성은 만족하지만, 최소성은 만족 불가</b>
<b>( )</b>	. 다른 릴레이션의 기본키를 참조하는 속성 or 속성들의 집합 . 참조되는 릴레이션 기본키와 대응되어 릴레이션 간 참조 관계

## ■ 무결성 (Integrity)

<b>( )</b> 무결성	. <b>기본키를 구성하는 어떤 속성도 NULL/중복 값 가질 수 없음</b> . 기본키의 속성 값이 NULL값이 아닌 원자 값을 갖는 성질
<b>( )</b> 무결성	. 릴레이션 내 튜플들이 각 속성의 <b>도메인</b> 에 지점된 값만 가짐
<b>( )</b> 무결성	. <b>외래키</b> 는 NULL 또는 참조 릴레이션의 기본키 값과 동일 . 릴레이션은 참조할 수 없는 외래키 값을 가질 수 없음
사용자 정의 무결성	. 속성 값들의 사용자가 정의한 제약 조건에 만족
데이터 무결성 강화	. 데이터 특성에 맞는 적절한 무결성을 정의하고 강화 . 강화 방법: 제약조건, 어플리케이션, 데이터베이스 트리거

## ■ 접근 통제 기술 (AC, Access Control)

<b>( )</b> AC	. <b>사용자의 신원/신분에 따라</b> 접근 권한 부여 . 데이터 소유자가 접근 통제 권한 지정/제어 . 객체 생성자가 모든 권한 갖고, 다른 사용자에게 허가 . SQL 명령어: GRANT / REVOKE
<b>( )</b> AC	. <b>주체와 객체의 등급을 비교 후 시스템이</b> 접근 권한 부여 . DB 객체별로 보안 등급 부여 및 사용자별로 인가 등급 부여 . 자신보다 보안 높은 객체에 읽기/수정/등록 불가하나 등급 같은 객체에는 모두 가능, 자신보다 낮은 객체에는 읽기 가능
<b>( )</b> AC	. <b>사용자의 역할에 따라</b> 접근 권한 부여 ( <b>중앙관리자</b> 가 지정) . 다중 프로그래밍에 최적화

## ■ ( )

- 기본 테이블에 기반을 둔 이름을 가지는 **가상 테이블** (기본 테이블과 동일 형태)
- 저장장치 내 가상으로, **논리적으로 존재** (실존하는 **물리적 존재 X**)
- 기본 테이블 및 뷰 삭제 시 해당 테이블/뷰를 기초로 정의된 다른 뷰도 자동 삭제
- ▷ 장점: **논리적 데이터 독립성 제공** / 데이터 자동 보안 제공 / 데이터 관리 용이
- ▷ 단점: **독립적 인덱스 보유 불가** / ALTER 변경 불가 / 삽입, 삭제, 갱신, 연산 제약





## ■ [ ]

- 이상(Anomaly) 현상이 발생하지 않도록 **중복성/중속성을 최소화**하기 위한 작업
- **논리적 설계 단계**에서 수행하며, 속성 수가 적은 테이블로 분할되어 관리가 용이해짐
- 데이터 구조 안정성 최대화 / 데이터 삽입 시 릴레이션 재구성 필요 최소화

※ 이상현상 종류 - [ ] 이상: 데이터 삽입 시 불필요한 데이터가 함께 삽입  
 [ ] 이상: 튜플 삭제 시 필요한 데이터도 함께 삭제  
 [ ] 이상: 일부만 수정되어 데이터 불일치 → 정보 모순 발생

## ■ 정규화 과정

제 [ ] 정규형	모든 도메인(Domain)이 <b>워치 값만</b> 으로 되어 있음
제 [ ] 정규형	기본키가 아닌 속성이 기본키에 대한 완전 함수적 종속 만족 <b>부분적 함수 종속을 제거</b> 한 정규형
제 [ ] 정규형	기본키가 아닌 모든 속성이 기본키에 대해 <b>이행적 함수 종속 관계</b> 를 만족하지 않아야 함 ( $X \rightarrow Y, Y \rightarrow Z$ 이면 $X \rightarrow Z$ )
[ ] (보이스/코드)	릴레이션 R에서 <b>모든 결정자가 후보키</b> 인 정규형
제 [ ] 정규형	릴레이션 R에 <b>다치 종속</b> 이 성립하는 경우 R의 모든 속성이 A에 함수적 종속 관계를 만족하는 정규형
제 [ ] 정규형	릴레이션 R의 모든 조인 종속이 R의 후보키를 통해서만 성립

※ [ ] : 정규화 과정 진행 후, **시스템 성능 향상** 및 운영 편의성을 위해 의도적으로 데이터 중복/통합/분리를 허용(**무결성 허용**)하여 정규화 원칙을 위반

## ■ 함수적 종속

[ ] 함수적 종속	[ ] Functional Dependency / 종속자가 기본키에만 종속
[ ] 함수적 종속	[ ] Functional Dependency / 기본키가 여러 속성으로 구성되어 있을 때 기본키를 구성하는 속성 중 일부만 종속
[ ] 함수 종속	[ ] Functional Dependency / $X \rightarrow Y, Y \rightarrow Z$ 이면 $X \rightarrow Z$

## ■ 관계 대수 - 원하는 정보의 검색 과정을 정의하는 **절차적 언어**

### ① 순수 관계 연산자

[ ]	Select (선택)	조건을 만족하는 튜플들의 부분 집합 (수평 연산)
[ ]	Project (추출)	속성들의 부분 집합, 중복 제거 (수직 연산)
[ ]	Join (조인)	두 개의 릴레이션을 하나로 합쳐 새로운 릴레이션 형성
[ ]	Division (나누기)	A의 속성이 B의 속성 값을 모두 가진 튜플에서 ( $A \supset B$ ) B가 가진 속성을 제외한 나머지 속성들만 추출

- [ ] **조인**: 두 릴레이션 속성 값을 비교 후 조건을 만족하는 튜플만 반환
- [ ] **조인**: 조건이 정확하게 '=' 등호로 일치하는 결과를 반환
- [ ] **조인**: 동등 조인의 결과에서 중복된 속성을 제거한 결과를 반환

### ② 일반 집합 연산자

[ ]	Union (합집합)	두 릴레이션의 합 추출 / 중복은 제거
[ ]	Intersection (교집합)	두 릴레이션의 중복되는 값만 추출
[ ]	Difference (차집합)	A 릴레이션에서 B 릴레이션 간 중복되지 않는 값을 추출
[ ]	Cartesian Product (교차곱)	두 릴레이션의 가능한 모든 튜플의 집합 → 속성/컬럼끼리 더하기 / 튜플끼리는 곱하기

## ■ 관계 해석 - 원하는 정보가 무엇이라는 것만 정의하는 **비절차적 언어**

논리 연산자	[ ]	OR	원자식 간 "또는" 관계로 연결
	[ ]	AND	원자식 간 "그리고" 관계로 연결
	[ ]	NOT	원자식에 대한 부정
정량자	[ ]	전칭 정량자	모든 가능한 튜플 "For All"
	[ ]	존재 정량자	어떤 튜플 하나라도 존재 "There Exists"

## ■ [ ] [ ]

- 데이터베이스 상태를 변환시키는 **하나의 논리적 기능을 수행하기 위한 작업 단위**

[ ] [ ]	트랜잭션 연산이 정상적으로 수행되거나 (Commit), 아니면 어떠한 연산도 수행되지 않아야 함 (Rollback)
[ ] [ ]	시스템의 고정 요소는 트랜잭션 수행 전/후로 동일해야 함
[ ] [ ]	개별 트랜잭션은 다른 트랜잭션의 간섭 및 영향 받지 않아야 함
[ ] [ ]	완료된 트랜잭션 결과는 영구적으로 기록되어야 함

- [ ] : 트랜잭션 정상 종료 후 변경된 내용을 DB에 반영하는 명령어
- [ ] : 트랜잭션 비정상 종료 후 모든 변경 작업을 취소하고 이전 상태로 원복
- [ ] : 데이터베이스 비정상 종료 시 트랜잭션 시작(start)과 완료(commit)에 대한 기록이 있는 트랜잭션들의 작업을 재작업 (이전 값을 이후 값으로 변경)
- [ ] : 데이터베이스 비정상 종료 시 시작은 있지만 완료 기록이 없는 트랜잭션들이 작업한 내용을 모두 취소 (이후 값을 이전 값으로 변경)

## ■ 데이터 회복 기법

- [ ] : 트랜잭션 실행 상태에서 변경되는 내용을 바로 데이터베이스에 적용  
→ 변경되는 모든 내용을 로그(Log)에 기록하여 장애 시 해당 로그 토대로 복원  
→ Redo / Undo 모두 수행
- [ ] : 트랜잭션 수행 후 부분 완료될 때까지는 데이터베이스에 바로 적용하지 않고, 지연시킨 후 부분 완료 시 로그(Log)의 내용을 토대로 저장  
→ 트랜잭션이 실패할 경우 Undo 수행 없이 Redo 만 수행
- [ ] : 트랜잭션 수행 중간에 검사시점 (Checkpoint) 지정하여  
검사 시점까지 부분 수행 후 완료된 내용을 중간중간 데이터베이스에 저장
- [ ] **기법**: 로그 미사용 / 데이터베이스를 동일 크기의 페이지로 분할 후  
각 페이지마다 복사하여 그림자 페이지 보관  
→ 변경 내용은 원본 페이지에만 적용하여 장애 발생 시 해당 페이지 사용/회복

## ■ [ ] 단위 [ ] : 로킹의 대상이 되는 객체의 크기

- 로킹: 데이터베이스 병행 제어 위해 트랜잭션(transaction)이 접근하고자 하는 데이터를 잠가(lock) **다른 트랜잭션이 접근하지 못하도록 하는 병행 제어 기법**
- 대상: 데이터베이스 / 파일 / 레코드
- [ ] 로킹 단위 ▼ → 로크의 수 ▲ → 로킹 오버헤드 ▲ → 병행성(공유도) ▲
- [ ] 로킹 단위 ▲ → 로크의 수 ▼ → 로킹 오버헤드 ▼ → 병행성(공유도) ▼

## ※ 병행제어 기법 종류

- [ ] **기법**: 일관성과 무결성을 유지하기 위한 트랜잭션의 순차적 진행을 보장 (병행 수행 트랜잭션들 간 동일 데이터 접근 차단)
- [ ] **검증**: 일단 트랜잭션을 수행하고, 트랜잭션 종료 시 검증을 수행
- [ ] **기법**: 타임 스탬프를 부여해 보여진 시간에 따라 트랜잭션 수행
- [ ] **제어** [ ] : 타임스탬프를 비교해 직렬가능성이 보장되는 적절한 버전을 선택해 접근하도록 함

## ■ 데이터베이스

- 논리적으로는 하나의 시스템에 존재하나, 물리적으로는 연결된 다수의 컴퓨터에 분산되어 있는 데이터 베이스
- 구성요소: 분산 처리기 (분산된 지역 컴퓨터) / 분산 데이터 베이스 / 통신 네트워크
- 목표: 위치 투명성 (Location) / 중복 투명성 (Replication) / 병행 투명성 (Concurrency) 분할 투명성 (Division) / 장애 투명성 (Failure)  
→ 사용자 입장에서 데이터베이스의 위치/중복/병행/분할/장애 여부 인식할 필요 X



### ■ 데이터베이스 관련 용어

<input type="text"/>	데이터베이스 테이블 검색 속도 향상을 위한 저장 위치 자료 → 데이터의 빠른 조회를 위한 위치 정보(포인터) 포함
<input type="text"/>	자주 사용하는 테이블 데이터를 동일 위치에 저장하여 데이터 접근 효율 향상
데이터베이스 <input type="text"/>	서비스 장애에 대비하여 데이터베이스 중복(복제)하여 관리 . Eager 기법: 모든 이중화 데이터 즉시 업데이트 . Lazy 기법: 변경 사항을 새로운 트랜잭션으로 각 노드에 전달
<input type="text"/>	대용량 데이터베이스를 여러 섹션으로 분할 (조회 속도 향상) . 범위 분할: 파티션 키 값을 범위로 분할 . 해시 분할: 해시 함수 적용 후 반환된 값으로 파티션 분할 . 합성 분할: 범위 분할 후 해시 함수 적용하여 다시 분할
<input type="text"/> 목표 (RTO)	<input type="text"/> / 서비스 중단 후 복원까지 최대 시간 → 서비스를 사용할 수 없는 최대 허용 시간
<input type="text"/> 목표 (RPO)	<input type="text"/> / 마지막 데이터 복구 후 허용되는 최대 시간 → 허용 가능한 최대 데이터 손실량 결정

### ■ 알고리즘

- 선형 검색: 하나씩 순서대로 비교하며 원하는 결과값을 찾아내는 검색
- 이진 검색: 검색 수행 전에 데이터 집합이 정렬되어야 함

### ■ 알고리즘 시간 복잡도

복잡도	알고리즘	설명
$O(1)$	해시 함수	자료 크기 무관 일정한 속도
$O(\log_2 N)$	이진 탐색	로그형 복잡도
$O(n)$	순차 탐색	선형 복잡도: 입력 자료를 하나씩 처리 (정비례)
$O(N \log_2 N)$	힙 / 합병(병합) 정렬	선형 로그형 복잡도
$O(N^2)$	선택 / 버블 / 삽입 퀵 정렬	대부분의 경우 힙/합병 보다 복잡도가 큼 $N^2 > N \log_2 N$ ( $N > 2$ )

### ■ 알고리즘 설계 기법

Divide and Conquer	문제를 최소 단위로 나누어 풀고, 각각을 다시 병합
Dynamic Programming	더 작은 문제의 연장선으로 여기고 과거에 구한 해를 활용
Greedy	현 시점에서 가장 최적의 방법을 선택
Backtracking	모든 조합을 시도하여 문제의 답을 선택 → 문제의 부모 노드로 되돌아간 후 다른 자손 노드 검색

### ■ 정렬 알고리즘

<input type="text"/>	이미 정렬된 파일에 새로운 레코드를 순서에 맞게 삽입하여 정렬
<input type="text"/>	N개의 레코드 중 최소값을 찾아 첫번째에 배치, 나머지 N-1개 레코드 중 최소값 찾아 두번째 배치, 이를 반복하여 정렬
<input type="text"/>	인접한 두 개의 레코드 키 값 비교 후 크기에 따라 레코드 위치 교환
<input type="text"/>	자료 이동 최소화 후 하나의 파일을 부분적으로 나눠가면서 정렬
<input type="text"/> 합병	이미 정렬된 2개의 파일을 한 개의 파일로 합병하여 정렬
<input type="text"/>	입력 파일을 매개변수 값으로 서버파일 구성 후 각 서버파일을 삽입 정렬 방식으로 순서 배열하는 과정을 반복
<input type="text"/>	완전 이진 트리를 이용한 정렬

### ■ 정렬 방법

#### ① 선택 정렬 (Selection sort)

- 최소값을 찾아 첫번째 숫자와 위치 교환, 이후 정렬된 값을 제외한 최소 숫자를 정렬되지 않은 숫자 중 첫번째 숫자와 다시 위치 교환, 이를 반복

시작:  9 6 7 3 5 →  →   
→  →  3 5 6 7 9

#### ② 삽입 정렬 (Insert sort)

- 두번째 숫자와 첫번째 숫자 비교하여 크기 순으로 정렬, 이후 3번째 숫자를 앞서 정렬된 숫자들 사이에 크기 순에 맞게 재정렬, 이를 반복

시작:  9 6 7 3 5 →  →   
→  →  3 5 6 7 9

#### ③ 버블 정렬 (Bubble sort)

- 왼쪽부터 인접한 두 숫자 간 크기 비교하여 위치 교환,  
→ 1 Pass 마다 정렬되는 숫자들 중 가장 큰 숫자를 오른쪽으로 밀어서 배치

시작:  9 6 7 3 5 →  →   
→  →  3 5 6 7 9



## ■ 운영체제 종류

	마이크로소프트사에서 개발한 운영체제
	미국 AT&T 벨연구소에서 개발한 운영체제 → 커널(Kernel) / 셸(Shell)로 구성
	리눅스 토발즈가 유닉스를 기반으로 개발한 운영체제
	애플이 유닉스 기반으로 개발한 운영체제
	구글이 리눅스 커널 기반으로 개발한 개방형 모바일 운영체제

## ※ UNIX 구성요소

	. 하드웨어 보호 / 프로그램 및 하드웨어 간 인터페이스 역할 . 프로세스 관리, 기억 장치 관리, 파일 관리, 입출력 관리, 데이터 전송 및 변환, 셸 프로그램 실행을 위한 프로세스 및 메모리 관리
	. 사용자의 명령 인식/해석 후 커널로 전달, 명령을 수행 (반복적인 명령 프로그램을 만드는 프로그래밍 기능 제공) . 시스템과 사용자 간 인터페이스 역할 . 초기화 파일을 이용해 사용자 환경 설정

※ UNIX Shell 환경 변수 : set, env, printenv, setenv

## ■ 주기억장치

	휘발성 (전원 off 시 data 없어짐) / 실행 중인 프로그램 저장
	속도 집적도 전력소모 재충전 용도
	SRAM (정적) 빠름 낮음 많음 불필요 캐시
	DRAM (동적) 느림 높음 적음 필요 주기억
	- 전력공급 없어도 내용 사라지지 않음 (비휘발성) - CMOS SETUP으로 값 수정 가능 - H/W와 S/W의 중간인 일종의 펌웨어 (Firmware) - 업데이트만으로 시스템 성능 향상 - 부팅 시 가장 먼저 동작하여 자체 진단 (POST) 실행 - BIOS에는 날짜, 전원관리, 부팅순서, 기본글꼴, 칩셋, 시스템 암호 (Windows 암호 X), PnP (Plug and Play), 하드디스크 타입, 안티바이러스 등의 정보 포함
	Mask ROM 제조 과정에서 저장된 ROM, 임의 수정 불가
	PROM 한번만 기록 가능, 이후는 읽기만 가능
	EPROM 자외선 신호를 이용한 ROM
	EEPROM 전기 신호를 이용한 ROM (플래쉬 메모리) - BIOS ROM으로 활용 및 업데이트 가능

## ■ 기타 메모리

	CPU(중앙처리장치) 내부에서 처리할 명령어나 연산의 중간 값 등을 일시적으로 저장하는 휘발성 메모리 (메모리 중 가장 빠른 속도)
	메모리 SRAM을 이용하여 CPU(빠름)와 주기억장치(느림) 사이의 속도 차이 해결 (버퍼 메모리 일종) → 캐시 적중률이 높을수록 시스템 처리 속도 증가
	메모리 보조기억장치를 주기억장치처럼 사용하는 메모리 (기억용량 확대) 큰 프로그램 실행 가능 / 매핑-mapping 방식 필요
	메모리 MP3나 디지털카메라 등에서 사용되는 일종의 EEPROM (비휘발성) (ex. USB / SD card / NAND Flash / SATA Memory)
	메모리 데이터 일시적으로 저장해 두 장치 간 데이터 전송 속도 차이 해결
	메모리 저장된 데이터의 일부를 이용하여 기억장치에 접근 후 데이터 확인 내용 참조 (매핑-mapping 방식) / 주소가 아닌 내용을 참조

★ H/W 처리 속도: 레지스터 > > 주기억 장치 > 보조기억장치

## ■ 기억장치 관리 전략

반입 전략 (Fetch)	요구 반입	특정 프로그램/데이터의 참조를 '요구'할 때 적재
	예상 반입	미래의 참조가 '예상'되는 데이터를 미리 적재
배치 전략 (Placement)	적합 (Fit)	사용 가능한 '첫 번째' 분할 영역에 데이터 배치
	적합 (Fit)	단편화를 '최소화'하는 분할 영역에 데이터 배치
	적합 (Fit)	단편화를 '최대화'하는 분할 영역에 데이터 배치
교체 전략 (Replacement)	새로운 데이터를 배치하고자 할 때 기존 사용 중 데이터를 교체 ex. FIFO, OPT, LRU, LFU, NUR ...	

## ■ 페이지 분할 기법

### 1) 기법 ()

- 가상기억장치에 보관된 프로그램과 주기억장치의 영역을 동일 크기로 분할 후 나눠진 프로그램(페이지)을 동일하게 나눠진 주기억장치의 영역(페이지 프레임)에 배치하여 실행하는 기법 (내부 단편화 발생 가능성 존재)

▶ 작은 페이지 크기 : 페이지 단편화 감소 / 페이지 이동시간 감소 / 워킹 셋 유지  
맵 테이블 크기 커지며 매핑 속도 감소 / 입출력 시간 증가

▶ 큰 페이지 크기 : 맵 테이블 크기 작아지며 매핑 속도 증가 / 입출력 시간 감소  
페이지 단편화 증가 / 페이지 이동시간 증가 / 불필요 데이터 적재

### 2) 기법 ()

- 가상기억장치에 보관된 프로그램을 다양한 크기의 논리적 단위로 분할 후 (Segment) 주기억 장치에 배치 및 실행 (외부 단편화 발생 가능성 존재)

※ Paging 기법의 경우, 프로그램을 동일 일정 크기로 분할함

## ■ 페이지 교체 알고리즘

	. 앞으로 가장 오랫동안 사용하지 않을 페이지를 교체 . 페이지 부재 횟수가 가장 적게 발생 / 효율적 교체 알고리즘
	. 가장 먼저 들어와 가장 오래 있었던 페이지를 교체
	. 최근에 가장 오랫동안 사용하지 않은(오래 전에 사용된) 페이지를 교체
	. 사용 빈도가 가장 적은 페이지를 교체 / 활발한 페이지는 교체 X

## [기출 문제] 20년도 4회 필기

4개의 페이지를 수용할 수 있는 주기억장치가 있으며, 초기에는 모두 비어 있다고 가정한다. 다음의 순서로 페이지 참조가 발생할 때, FIFO 페이지 교체 알고리즘을 사용할 경우 페이지 결함 발생 횟수는?

▶ 페이지 참조 순서 : 1, 2, 3, 1, 2, 4, 5, 1

참조 페이지	1	2	3	1	2	4	5	1
페이지 프레임 (4개)	1	1	1	①	1	1	5	5
		2	2	2	②	2	2	1
			3	3	3	3	3	3
						4	4	4
부재 발생								

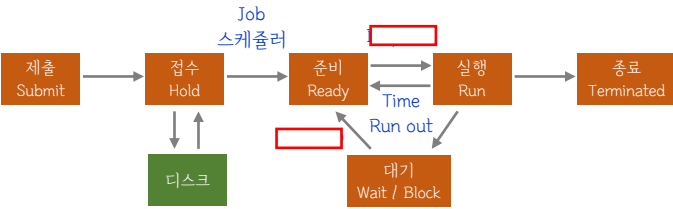
4개의 페이지가 다 찰 때까지 진행  
(앞서 중복 숫자가 있으면 넘어감 = 부재 미발생)

전체 페이지가 다 차서 새로운 참조 값을 기존 페이지에 교체해야 하는 상황  
이때, FIFO 기준에 따라 교체된지 오래된 페이지부터 신규 참조 값을 대체



### ■ 프로세스 (Process)

- 프로세서(CPU)에 의해 처리되는 사용자/시스템 프로그램
- 프로세스 제어 블록 (PCB, Process Control Block)에 프로세스 식별자, 상태 등의 중요 정보를 저장 / 각 프로세스 실행 시 고유 PCB 생성, 작업 완료 후 PCB 제거



- ※ **준비 상태**: 준비 상태에서 대기 중인 프로세스가 실행 상태로 전이되는 과정
- ※ **입출력 작업**: 입출력 작업이 완료되어 프로세스가 대기 상태에서 준비 상태로 전이
- ※ Context Switching: 이전 프로세스의 상태 레지스터 내용 보관하고 (문맥 교환) 다음 프로세스의 레지스터를 적재하는 과정

### ■ **프로세스 내 작업 단위 / 프로그램 단위**

- 단위: 하나의 프로세스에 하나의 스레드 / 다중: 하나의 프로세스에 복수의 스레드
- 동일 프로세스 환경 내 서로 다른 독립적 다중 스레드 수행 가능

**[장점]** 한 프로세스 내 복수의 스레드 생성하여 작업 병행성 향상

응답 시간 단축 / 기억 장소 낭비 최소화 / 프로세스 간 통신 향상

운영체제 성능 및 프로그램 처리 효율 향상 / 접근 가능한 기억장치 사용

<b>수준</b>	. 사용자가 만든 라이브러리를 이용한 스레드 운용 . 속도 빠르나 구현 어려움
<b>수준</b>	. 운영체제의 커널에 의한 스레드 운용 . 구현 쉬우나 속도가 느림

### ■ 스케줄링: 여러 프로세스의 처리 순서를 결정하는 기법

- 1) **우선순위**: 운영체제가 실행 중인 프로세서로부터 CPU를 강제 빼앗음  
→ 프로세서 별 CPU 처리 시간에 순서 조정으로 효율적 운영 가능 / 높은 오버헤드

<b>우선순위</b>	. 우선순위 없이 시간 단위로 CPU 할당 (공평한 자원 분배)
<b>비선점형 SJF</b>	. 비선점형 SJF 기법을 선점 스케줄링 형식으로 변경 . 잔여 실행 시간이 가장 짧은 프로세스에 CPU 우선 할당
<b>그룹 스케줄링</b>	. 프로세스를 그룹화 후 각 그룹마다 다른 준비 상태 큐를 사용 . 하위 준비 상태의 큐 실행 간 상위 준비 상태 큐 유입 시 상위 프로세서에 CPU 우선 할당
<b>FCFS</b>	. 프로세스 생성 시 가장 높은 우선 순위 준비 큐에 등록되며 등록된 프로세스는 FCFS 순서로 CPU를 할당받아 실행 . 단계가 내려갈수록 시간 할당량(Time Quantum)이 증가

- 2) **라운드 로빈**: 기존 프로세서가 점유하고 있는 CPU를 빼앗을 수 없음  
→ 낮은 처리율 / 적은 오버헤드

<b>FIFO</b>	. 프로세서 도착 순서에 따라 처리 순서 결정 (=FIFO)
<b>라운드 로빈</b>	. 준비 상태의 프로세서 중 실행 시간이 가장 짧은 것부터 처리
<b>우선순위</b>	. 우선 순위를 계산하여 프로세스의 처리 순서 결정 → 우선 순위 = (대기시간 + 서비스 시간) / 서비스 시간 . 실행 시간이 긴 프로세서가 불리한 SJF 기법을 보완

- **교착 상태**: 둘 이상의 프로세스들이 서로 점유하고 있는 자원을 요구하며 무한정 대기하고 있는 상태

### [교착 상태 필요 충분 조건]

<b>상호 배타성</b>	한 번에 한 개의 프로세스만이 공유 자원을 사용
<b>점유 및 점유 가능성</b>	최소한 하나의 자원을 점유하면서 다른 프로세스에 할당된 자원을 추가로 점유하기 위해 대기하는 프로세스 존재
<b>비점성</b> (Non-preemption)	다른 프로세스에 할당된 자원은 끝날때까지 강제로 선점 불가
<b>대기</b> (Wait)	프로세스들이 원형으로 구성되어 있어 앞/뒤 프로세스 자원을 요구

### [교착 상태 해결 방법]

<b>예방</b>	. 교착발생 4가지 조건 중 하나 이상을 사전에 제거하여 예방
<b>회피</b>	. 발생된 교착상태를 적절히 피해가는 방법 ※ 은행원 알고리즘 (Banker's Algorithm): 은행에서 모든 고객의 요구가 충족되도록 현금을 할당하는 데서 유래
<b>탐색</b>	. 교착상태에 있는 프로세스 및 자원을 발견 / 점검
<b>회복</b>	. 교착상태 유발 프로세스 종료 / 프로세스 내 할당된 자원 선점






### ※ 기억장치 관련 용어

<b>교체</b>	. 프로세스 처리 시간보다 페이지 교체 시간이 더 많아지는 상태 . 전체 시스템 성능 저하 / 다중 프로그래밍 다발 시 스래싱 발생 <b>[방지 방법]</b> 다중 프로그래밍 적정 수준 유지 / 일부 프로세스 중단 페이지 부재 빈도 조절 / 워킹 셋 유지
<b>시간 구성성</b>	. 프로세스 실행 간 주기억장치 참조 시 일부 페이지만 집중 참조 . 시간 구성성: 한 페이지를 일정 시간 동안 집중적으로 액세스 → 반복/순환(Loop), 스택(Stack), 부 프로그램(Sub Routine)
<b>공간 구성성</b>	. 공간 구성성: 프로그램 실행 시 인접한 페이지들도 집중 참조 → 배열순회(Array Traversal), 순차적 코드 실행
<b>일정 시간 동안 자주 참조하는 페이지들을 주기억장치에 상주시켜 페이지 부재 및 교체 현상을 최소화</b>	→ 기억장치 사용 안정화
<b>페이지 부재</b>	. 프로세스 실행 중 필요한 페이지가 주기억장치에 누락(부재)



## ■ 정보 전송 방식

- 단방향: 한쪽 방향으로만 전송 가능 (TV, 라디오)
- 반이중: 한쪽에서 송신하면 다른 쪽에서는 수신만 가능 (무전기)
- 전이중: 동시에 송/수신 가능 (전화기)

<b>스타형/성형 중앙 집중형</b>		중앙노드와 1:1 (P2P) 연결 / 고장 발견, 유지 보수, 확장이 쉬운 중앙노드의 제어장치가 통신망의 처리능력, 신뢰성 결정
<b>버스형</b>		한 개의 회선에 여러 단말 장치 연결, 회선 양 끝 종단장치 필요 단말 장치 하나가 고장나도 전체 영향 없기에 신뢰성 높음 설치/제거 용이, <b>기밀성 낮고 통신 회선의 제한이 있음</b>
<b>링/루프 원/환형</b>		인접한 단말기를 서로 연결 / 양방향 전송 가능 통신망 하나가 고장 시 전체 통신망 마비 <b>단말 장치 추가/제거, 기밀보호 어려움</b>
<b>트리/계층 분산형</b>		나뭇가지 모양으로 계층적 연결 / 분산처리 시스템 구성 방식 <b>확장이 많으면 트래픽 (통신량)이 과중됨</b>
<b>망/매쉬 그물/완전형</b>		모든 지점의 단말기를 서로 연결 / 빠른 응답 시간, 높은 연결성 통신량이 많을 경우 유리함 (공중 데이터 통신망) 회선 장애 발생 시 다른 경로로 전송 / 높은 보안성 및 안정성 <b>단말 장치 추가/제거 어려움</b> (낮은 확장성)

## ■ 동기 / 비동기 전송

구분	동기식	비동기식
전송 단위	<input type="text"/> 단위	<input type="text"/> 단위
동기 제어 방식	클럭 동기	Start bit, Stop bit
통신 속도	<input type="text"/>	<input type="text"/>
회로 복잡도	<input type="text"/>	<input type="text"/>
휴지 시간	블록 간 휴지 시간 없음 → <input type="text"/> 전송 효율	문자 간 휴지 시간 존재 가능 → <input type="text"/> 전송 효율
예시	데이터 통신망, 전화 교환망	편지, 이메일
전송 방식	문자 동기 방식 (SYN, STX, ETX) 비트 동기 방식	-

### ※ ()

- 자료를 일정한 크기(Cell)로 정한 후 순서대로 전송 / 멀티미디어 서비스 적합
- Cell : 고정 길이 패킷 크기 **53 Byte** (페이로드 48 Byte + 헤더 5 Byte)
- 단순한 처리 방식 / 고속망 적합 / 실시간 및 비실시간 서비스 제공 가능
- 고정 비트레이트(CBR) 및 가변 비트레이트(VBR) 모두 처리 가능

## ■ IPv4 / IPv6 특징

<input type="text"/>	<input type="text"/>
10.97.135.183 (0~255 = 256개 = 8 Bit)	2544::D6::4B::13F:2C4F 0000~FFFF (각 자리를 <b>·콜론</b> 으로 구분)
10진수 (0~9)	16진수 (0~9 + A~F)
8 Bit x 4개 = <input type="text"/> Bit	16 Bit x 8개 = <input type="text"/> Bit
유니, 멀티, <b>브로드캐스트</b>	유니, 멀티, <b>애니캐스트</b>

※ IPv6는 주소의 확장성/용통성/연동성이 높으며, IPv4 대비 전송 속도가 빠름,  
인증성/기밀성, 데이터 무결성으로 보안 문제 해결 가능  
패킷 크기 제한 없으며, 등급/서비스별 패킷 구분 가능 → 품질 보장 용이

### ※ 캐스팅 전송 방식

유니캐스트	일 대 일 통신 (단일 송신 ↔ 단일 수신)
멀티캐스트	일 대 다 통신 (단일 송신 ↔ 다중 수신)
브로드캐스트	호스트가 속해 있는 네트워크 전체를 대상으로 패킷을 전송
애니캐스트	일 대 일 통신 (수신 가능한 노드 중 가장 가까운 한 노드와 통신)

- () : 정보 교환을 위해 무엇을, 어떻게, 언제 통신할 것인지  
(구문, 의미, 순서)를 정한 **통신 규약**

[기능] **흐름 제어**: 패킷 흐름(전송량/속도)을 조정하여 송수신 안정성 유지

**동기화**: 송/수신이 같은 상태를 유지

**오류 제어**: 전송 중 발생한 오류를 검출, 복원/정정

### ※ 흐름제어 방식

<input type="text"/>	. 수신 측 확인 후 다음 패킷 전송 / 한 번에 하나의 패킷만 전송
<input type="text"/>	. 수신 측 확인 없이 사전에 정해진 패킷 수만큼 연속적으로 전송 . 한 번에 여러 패킷을 동시에 전송 가능 . 긍정 수신 시 윈도우 크기 증가 / 부정 신호 시 윈도우 크기 감소
<input type="text"/>	. 정상 패킷 전송 시 혼잡 윈도우 크기를 패킷마다 1씩 증가시키나 혼잡 현상 발생 시 혼잡 윈도우 크기를 1로 줄임
<input type="text"/>	. 네트워크 내 패킷의 지연이 너무 높아 트래픽이 붕괴되지 않도록 패킷의 흐름을 제어하는 트래픽 제어

### ※ 전송 오류 제어 방식

- 오류 수정 ()  
: 수신측에서 **스스로 오류 검출 후 수정**하는 방식 (송신측에 별도 통보 X)  
ex. 해밍 코드 / 상송 코드
- 오류 수정 ()  
: 오류 발생 시 **송신측에 재전송 요구**  
ex. 패리티 검사 / CRC (Cyclic Redundancy Check) / 블록 합

### ※ 오류 검출 방식

<input type="text"/>	. 수신측에서 오류 검출 후 자동 수정 / <b>1비트 오류 수정</b> 가능 . 검출 가능 최대 오류 수 = 해밍 거리 - 1
<input type="text"/>	. 순차적/한계값 디코딩 사용 / 여러 비트의 오류 수정 가능
<input type="text"/>	. 7~8개 비트로 구성된 데이터 블록 끝에 특정 패리티 비트 (짝수/홀수)를 추가하여 오류 검출
<input type="text"/>	. 다항식을 통해 산출된 값으로 오류 검사 (집단 오류 해결) . 데이터 뒤 오류 검출코드 FCS (Frame Check Sequence) 추가
<input type="text"/>	. 짝수개 비트 오류 검출 불가한 패리티 검사를 개선한 방법 . 데이터 블록의 수평/수직 패리티 비트 추가

- : 다양한 플랫폼의 특정 사용자 데이터에 접근하기 위해 비밀번호를 제공하지 않고 제 3자 클라이언트가 사용자의 **접근 권한을 위임** 받을 수 있는 표준 프로토콜 (사용자 인증에 사용되는 표준 인증 방법 / 공개 API로 구현)

- () : 실행 프로세스 간 통신을 가능하게 하는 기술  
- 종류 : 파이프, 메시지 큐, 공유 메모리, 세마포어, 소켓

- : 기지국/엑세스 포인트와 같은 네트워크 장치가 필요하지 않고, 멀티 홉 라우팅 기능에 의해 무선 인터페이스가 가지는 통신 거리상의 제약 극복  
→ 긴급 구조, 긴급 회의, 전쟁터에서의 군사 네트워크에 활용

### ■ IEEE 802 LAN 표준안

802.2	논리 링크 제어 (LLC)	802.6	도시형 통신망 (MAN)
802.3	CSMA/CD	802.8	Fiber Optic LANS
802.4	Token Bus	802.9	종합 음성/데이터 네트워크
802.5	Token Ring	802.11	무선 LAN





## ■ OSI 참조 모델 (7계층)

		TCP/IP 4계층
<b>7계층</b> (Application)	응용 프로그램이 OSI 환경에 접속 가능한 서비스	< 응용 계층 > Telnet / FTP HTTP / POP SMTP DHCP / SNMP DNS
<b>6계층</b> (Presentation)	응용-세션 간 코드/데이터 변환, 데이터 암호화/압축	
<b>5계층</b> (Session)	컴퓨터 간 세션/연결을 생성/유지/종료하여 적절한 통신 상태 유지	
<b>4계층</b> (Transport)	종단(END-TO-END) 간 투명한 데이터 전송/전달 주소 설정 / 다중화 / 오류 제어 / 흐름 제어 수행	< 전송 계층 > TCP / UDP
<b>3계층</b> (Network)	개방 시스템 간 네트워크 연결을 관리 / 데이터 교환 경로 설정(Routing), 트래픽 제어, 패킷 정보 전송	< 인터넷 계층 > IP / ICMP ARP / RARP
<b>2계층</b> (Data Link)	송/수신 간 속도 차이 해결 위한 흐름 제어 기능 프레임 동기화 / 오류 제어 기능 / 노드 간 프레임 전송 → 물리적 주소 (MAC) 결정	< 네트워크 액세스 > HDLC / PPP LLC
<b>1계층</b> (Physical)	전송에 필요한 기계적, 전기적, 기능적 특성을 정의 데이터를 0과 1의 전기적 신호로 변환	

■ **프로토콜** :

- 대표적 데이터링크 프로토콜 / 방식
- BSC (프레임에 전송 제어 문자 삽입) / LAP-B (패킷교환망) / LAP-D (ISDN에서 사용)
- PPP (전화회선을 이용한 PC 간 연결) / LLC (LAN 프로토콜)

[HDLC 링크 구성] 불균형 (1:다 구성) / 균형(1:1 구성)

\*주국 : 데이터 전송 명령 수행 / \*종국 : 명령에 응답 / \*혼합국 : 명령과 응답 모두 처리

[HDLC 전송 모드]

<b>모드 : NRM</b> (Mode)	. 주국과 종국의 관계 (불균형) → 종국에서 데이터 전송 시 주국 허락 필요
<b>모드 : ABM</b> (Mode)	. 대등한 혼합국 간의 관계 (균형) → 양쪽에서 명령과 응답 모두 전송 가능
<b>모드 : ARM</b> (Mode)	. 불균형 모드 / 종국이 주국의 허락 없이 데이터 전송 및 송신 가능하나 제어 기능은 주국만 허용

[HDLC 프레임 종류]

<b>프레임</b> (제어부가 0으로 시작)	. 사용자 데이터 및 일부 제어 정보의 전달 ① Seq(송신용 순서번호) / ② Next(응답용 순서번호) / ③ P/F(주국 컴퓨터가 종국 컴퓨터의 데이터 전송 허용) 구성
<b>프레임</b> (제어부가 1으로 시작)	. 확인 응답, 데이터링크의 흐름 제어, 오류 제어 용도 . 전송 목적이 아니라 Seq 값 필요 없음 (Next만 존재) . 2비트로 구성되어 4가지 Type(0~3)으로 구성
<b>프레임</b> (제어부가 1로 시작)	. 순서가 없는 프레임 / 링크의 동작 모드 설정 및 관리 . 2비트 Type + 3비트 Modifier = 총 5비트 구성

## ■ 네트워크 계층 프로토콜

	. Protocol / 전송할 데이터에 주소 지정하고 경로를 설정 . 비연결형 데이터그램 방식 / 신뢰성 보장 X / 패킷을 분할 및 병합 . 헤더 체크섬(header checksum) 제공 / 데이터 체크섬은 제공 X
	. Protocol / 헤더 8Byte . IP와 함께 통신 간 오류 처리와 전송 경로 변경 등 제어 메시지 관리
	. Protocol . 호스트 IP주소를 네트워크 접속 장비의 물리적 주소(MAC)로 바꿈
	. Protocol . ARP 반대로 물리적 주소(Mac Address)를 IP주소로 변환하는 기능
	. Protocol . 멀티캐스트 지원하는 호스트/라우터 간 멀티캐스트 그룹 유지
	. Protocol . 최소 Hop count (최단거리) 경로로 라우팅하는 프로토콜
	. (네트워크 주소 변환) . 사설 네트워크에 속한 IP를 공인 IP 주소로 변환하는 기술 → 1개의 정식 IP 주소에서 다량의 가상 사설 IP 주소를 할당/연결

## ■ 전송 계층 프로토콜

	. Transmission Control Protocol / 양방향 서비스 제공 . 순서 제어/오류 제어/흐름 제어 기능 → 높은 신뢰성 . 프로토콜 헤더는 기본적으로 20~60 Byte . 패킷 단위의 스트림 위주 전달
	. User Datagram Protocol / 비연결형 서비스 제공 . TCP 대비 단순한 헤더 구조로 오버헤드 적고, 전송속도 빠름 (제어 X) . 실시간 전송 유리 / 신뢰성보다는 속도가 중요한 네트워크에 활용
	. Real-time Control Protocol / RTP 패킷의 전송 품질 제어 . 세션에 참여한 각 참여자들에게 주기적으로 제어 정보 전송

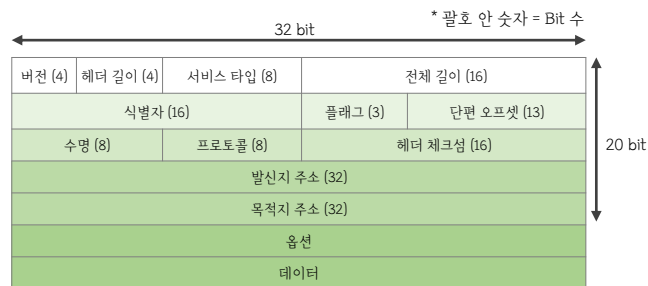
## ■ 응용 계층 프로토콜

※ 괄호 안 숫자 : 포트 번호

	(23) 다른 컴퓨터 접속 후 원격 서비스 제공 / 가상의 터미널 기능 수행
	(21) File Transfer Protocol / 원격 파일 전송 프로토콜 (컴퓨터-인터넷)
	(80) Hyper Text Transfer Protocol / WWW 내 HTML 문서 송수신 위한 프로토콜 / GET과 POST 메서드를 통해 메시지 주고 받음
	(25) Simple Mail Transfer Protocol / 전자 우편 교환 서비스
	Domain Name System / 도메인 네임을 IP 주소로 매핑하는 시스템
	Simple Network Management Protocol TCP/IP 네트워크 관리 프로토콜 (네트워크 기기 정보 전송 규약)
	IP주소 및 TCP/IP 프로토콜의 기본 설정을 클라이언트에게 자동 제공

## ■ IP 헤더 구조

버전 (Version)	사용 중인 프로토콜 버전 (ex. IPv4 / IPv6)
헤더 길이 (Header Length)	IP 헤더의 길이를 바이트 단위로 표현 (20 ~ 60 Byte)
서비스 유형 (Service Type)	요구 서비스 품질 지정 (패킷의 전송 우선순위 제공)
전체 패킷 길이 (Packet Length)	IP 헤더 + 데이터 포함된 전체 IP 패킷 길이
식별자 (Identification)	전송 패킷을 식별하기 위한 부여된 일련번호 (16 Bit)
플래그 (Flags, DF/MF)	단편화(분할) 상태를 표현
단편 오프셋 (Fragmentation Offset)	단편화(분할)된 패킷들의 순서 지정
수명 (Time to live, TTL)	패킷이 네트워크를 통과할 수 있는 남은 라우터의 최대 개수
프로토콜 타입 (Protocol)	데이터에 포함되는 상위 프로토콜 종류 표현 Ex. 1 = ICMP / 2 = IGMP / 6 = TCP / 17 = UDP
헤더 체크섬 (Header Checksum)	헤더 필드의 오류 발생 유무 검사
발신지/목적지 주소 (Source / Destination IP Address)	패킷을 보낸/받는 IP 주소



<input type="checkbox"/>	. <input type="text"/> Protocol . Cisco사에서 개발 / UDP 사용
<input type="checkbox"/>	. <input type="text"/> Protocol . MS사 개발 / 일 대 일 통신 지원
<input type="checkbox"/>	. <input type="text"/> Protocol . L2FP와 PPTP의 기능 결합 / 인터넷 내 두 지점 간 가상의 터널을 만들어 통신을 안전하게 전송

(2) 전체 호스트는  $2^6 = 64$ 개, 이 중 네트워크 주소와 브로드캐스트 주소를 제외하면  $\boxed{62}$ 개



## ■ 정보 보안 3대 요소

- : 시스템 내 정보 자원은 인가된 사용자에게만 접근 허용
- : 오직 인가된 사용자가 시스템 내 정보 수정 가능
- : 인가된 사용자는 권한 범위 내에서 언제든지 자원 접근 가능  
+ 인증 (사용자의 신분 확인) / 부인방지 (부인할 수 없도록 송·수신 증거 제공)

## ■ AAA (Triple-A)

- (□) : 사용자가 네트워크 접속 전에 시스템에서 사용자 신원 확인
- (□) : 검증된 사용자에게 사용가능한 접근 권한 확인
- (□) : 사용자의 자원 사용에 대한 정보 수집 (과금, 감사, 보고서)

## ■ 양방향 암호화 방식

	□ / □ / □	□ / □
특징	동일한 키로 데이터를 암호화/복호화 블록 (Block) 2 bit 이상 연산 스트림 (Stream) 1 bit 씩 연산	암호화 키는 DB 사용자에게 공개 복호화 키는 비밀키로 관리자만
종류	DES, AES, SEED, ARIA, IDEA, LEA	RC4, LFSR 소인수분해 : RSA, Rabin 이산대수 : Diffie-Hellman, DSA 타원곡선 : ECC
키 개수	N(N-1)/2 개	2N 개
장점	알고리즘 단순 암호/복호화 속도 빠름	키 분배 용이 관리해야 할 키의 수 적음
단점	관리해야 할 키의 수 많음	알고리즘 복잡 암호/복호화 속도 느림

## ※ 양방향 암호화 종류

□	IBM에서 개발, 미국 NBS에서 국가 표준으로 발표 블록 크기 64비트 / 키 길이 56비트 / 16라운드 (페이스펠 구조)
□	미국 표준 기술 연구소 (NIST) 발표 (키 길이 128/192/256 비트) DES 대체 목적 / 레인달(Rijndael) 기반 암호화 / SNP 암호 방식
□	국내 개발 개인키 (128 및 256 비트 사용)
□	국가보안기술연구소(NSRI) 주도 개발 (128/192/256 비트)
□	스위스 개발 PES 개선한 암호키 (64비트 블록 / 128비트 키)
□	국내 NSRI 개발 (128비트의 데이터 블록을 암호화)
□	미국 NSA 개발 / 64비트 입출력, 80비트의 키, 32라운드 (전화기)
□	Ron Rivest가 설계한 스트림 암호화 (옥텟 단위 기반)
□	선형 연산을 통한 다음 상태 생성 / 스트림 기반 난수 생성 활용
□	MIT 공개키 암호화 / 소인수분해 어려운 큰 소수 숫자 활용
□	Rabin 개발 (소인수분해) / RSA보다 빠름
□	두 사용자가 사전에 어떤 비밀 교환 없이도 공통키를 교환 가능
□	미국 표준 디지털 서명 알고리즘
□	타원곡선 이론에 기반한 공개 키 암호 방식

## ■ 단방향 암호화 방식 - □ 함수 (□)

- □ 알고리즘 (□ 함수)로 불리며, □ 함수로 변환된 값/키를 □ 값/키라고 함
- 임의의 길이의 입력 데이터를 받아 고정된 길이의 □ 값을 변환한다. (단방향 함수)
- 종류: SHA 시리즈 / HAVAL / MD4 / MD5 / N-NASH / SNEFRU

중첩법 (폴딩법)	레코드 키를 여러 부분으로 나누고, 나눈 부분의 각 숫자를 더하거나 XOR 한 값을 홈 주소로 사용하는 방식
제산법	레코드 키로 해시표의 크기보다 큰 수 중에서 가장 작은 소수로 나눈 나머지를 홈 주소로 삼는 방식
기수변환법	키 숫자의 진수를 다른 진수로 변환시켜 주소 크기를 초과한 높은 자릿수는 절단하고, 이를 다시 주소 범위에 맞게 조정하는 방식
숫자분석법	키 값을 이루는 숫자의 분포를 분석하여 비교적 고른 자리를 필요한 만큼 택해서 홈 주소로 삼는 방식

기타 : 제곱법, 무작위법

## ■ 서비스 공격 유형

□ 공격 (□)	대량의 데이터를 한 곳의 서버에 집중적으로 전송하여 표적이 되는 서버의 정상적인 기능을 방해
□ 공격 (□)	여러 대의 장비에서 한 곳의 서버에 분산 서비스 공격 수행 종류 : Trinoo, Tribe Flood Network, Stacheldraht
□	Ping 명령을 전송할 때 허용범위 이상의 ICMP 패킷을 전송하여 대상 시스템의 네트워크를 마비
□	과도한 ICMP 메시지에 의한 응답 과다로 시스템 에러 유발
□	SYN 패킷 신호만 전송하여 각 서버의 가용 사용자 수를 선점 후 다른 사용자의 서버 접근을 차단
□	다량의 UDP 패킷을 전송하여 네트워크 자원을 고갈
□ (□)	IP / ICMP 특성을 악용하여 엄청난 양의 데이터를 한 사이트에 집중적으로 보내 네트워크를 붕괴 상태로 만들
□	Fragment number 값을 변형하여 수신측에서 패킷 조립 시 오류로 인한 과부하로 시스템 다운을 유도
□	송/수신 IP 주소를 모두 타겟 IP주소로 하여 자기 자신에게 무한히 응답하게 하는 공격
□	악의적 사용자가 지인/유명인 사칭하여 로그온한 사용자의 계정 정보 및 신용 정보 탈취
□	위조된 매체 접근 제어(MAC) 주소를 지속 보내 스위치 MAC 주소를 혼란시켜 더미 허브 (Dummy Hub)처럼 작동

## ■ 정보 보안 솔루션

□ (□)	내부 → 외부로 나가는 패킷은 그대로 통과 외부 → 내부로 유입되는 패킷은 인증된 패킷만 통과
□ (□)	일반 방화벽은 탐지 불가한 SQL 삽입 공격, XSS 등 웹 기반 공격을 방어하는 목적으로 웹 서버에 특화된 방화벽
□ (□)	Intrusion Detection System 시스템의 비정상적인 사용/오용/남용 등을 실시간 탐지
□ (□)	Intrusion Prevention System / 방화벽 + 침입 탐지 시스템 비정상적 트래픽/패킷을 능동적으로 차단하고 격리
□ (□)	Data Loss Prevention / 내부 정보의 외부 유출을 방지 사내 직원이 사용하는 시스템의 모든 정보 탐지/통제
□ (□)	공공 네트워크 암호화 기술을 이용하여 사용자가 마치 자신의 전용 회선을 사용하는 것처럼 하게 함
□ (□)	네트워크에 접속하는 내부 PC의 MAC 주소를 IP 관리 시스템에 등록한 후 일관된 보안 관리 기능을 제공
□ (□)	상기 보안 솔루션에서 발생한 로그 기록을 통합하여 관리 → 종합적인 보안 관리 체계 수립 가능
□	Security Information & Event Management 보안 경로의 실시간 분석 제공 로그 및 이벤트 통합 관리 / 빅데이터 기반 보안 솔루션
□	Software Defined Perimeter / '블랙 클라우드'라 불림 GIG 네트워크 우선권에 따라 DISA에서 수행한 작업에서 발전
□	외부에서 유입된 프로그램이 보호된 영역에서 동작해 시스템이 부정하게 조작되는 것을 막는 보안 형태
□	Fraud Detection System / 금융거래의 이상 거래 탐지 및 차단
□	Tamper Proofing / SW의 위·변조 시 SW 오작동시켜 악용 방지 해시함수, 펌거 프린트, 워터마킹 등 보안 요소 생성 후 보호
□	독립적 보안 구역을 따로 두어 중요한 정보를 보호하는 하드웨어 기반의 보안 기술



## ※ [ ] 시스템 ([ ] System)

- 시스템의 비정상 사용/오용/남용 등을 실시간 탐지

. **오용 탐지**: 사전에 정립되어 입력해 둔 공격 패턴 감지 시 통보

. **이상 탐지**: 평균적인 상태 기준에서 비정상적 행위 또는 자원의 사용 감지 시 통보

### [종류]

[ ] ([ ]-based)	. 시스템 내부 감시/분석, 내부 변화를 실시간으로 감지하여 외부 침입자 작업 기록 및 추적 → OSSEC, md5deep, AIDE, Samhain
[ ] ([ ]-based)	. 외부 침입 감시/분석, 네트워크 트래픽 감시 → Snort, Zeek

- IDS 설치 가능 위치: 패킷 라우터로 들어가기 전 / 라우터 뒤 / 방화벽 뒤 / **DMZ\***

※ **DMZ**: 외부 인터넷에 서비스 제공하는 서버가 위치하는 네트워크

## ■ 정보 보안 프로토콜

[ ] [ ]	. 서로 연결된 컴퓨터 간 원격 명령실행 및 셸 서비스 수행 . 키를 통한 인증은 클라이언트의 공개키를 서버에 등록해야 함 . 전송 데이터는 암호화되며, 기본적으로 <b>22번 포트</b> 사용
[ ] [ ]	. 웹 브라우저 ↔ 서버 간 안전한 데이터 전송 목적 . https로 시작 / 443번 포트 사용
[ ] [ ] [ ]	. 인터넷 커뮤니케이션을 위한 개인 정보와 데이터 무결성을 제공하는 보안 프로토콜 / SSL의 개선 버전
[ ] [ ]	. IP 네트워크 계층을 안전하게 보호하기 위한 보안 프로토콜 → 전송 모드 (트래픽 경로 유출 가능) / 터널 모드 (패킷 전체) . AH(Authentication Header)와 ESP(Encapsulating Security Payload)라는 두 가지 보안 프로토콜을 사용
[ ] [ ]	. 웹상에서 네트워크 트래픽을 암호화 / 기존 HTTP의 보안 확장
[ ] [ ] [ ]	. 임시 키 무결성 프로토콜 / WEP 취약성을 보완하기 위해 암호 알고리즘의 입력 키 길이를 128비트로 늘림

## ■ 코드 오류

[ ] 오류 ([ ] error)	입력시 한 자리를 빼놓고 기록	ABCD → ABD_
[ ] 오류 ([ ] error)	임의의 한 자리를 잘못 기록	ABCD → ABFD
[ ] 오류 ([ ] error)	입력시 좌우 자리를 바꿔어 기록	ABCD → ACBD
[ ] 오류 ([ ] error)	전위 오류가 두 가지 이상 발생	ABCD → CABD
[ ] 오류 ([ ] error)	입력시 한 자리 추가로 기록	ABCD → ABCDE
[ ] 오류 ([ ] error)	다른 오류가 두 가지 이상 결합	ABCD → ACBDF

## ■ [ ] [ ]

- 정보보호 관리 체계: 정보 자산을 안전하게 보호하기 위한 보호 절차와 대책 확보 조직에 맞는 정보보호 정책을 수립하고 위험에 상시 대응가능한 보안 대책 통합 관리

## ■ [ ] 데이터 ([ ] Data)

- 특정 목적으로 수집된 데이터가 활용되지 않고 저장만 되어 있는 대량의 데이터

## ■ 정보 보안 침해 공격

해킹	시스템에 침입해 정보를 수정하거나 빼내는 행위
[ ]	시스템에 침입해 정보를 파괴하거나 변경하는 행위
[ ]	악성코드에 감염되어 다른 컴퓨터를 조종하는 행위
[ ] 서버	해커가 감염된 종피 PC에 명령을 내리고 악성코드를 제어하기 위한 용도로 사용하는 서버
[ ] [ ]	다른 컴퓨터의 취약점을 이용하여 스스로 전파하거나 메일로 전파되며 스스로를 증식 ( <b>독자적으로 실행 가능</b> )
[ ]	파일, 메모리 영역에 자신을 복제(기생)하는 악성 프로그램 ( <b>독자적으로 실행 불가</b> )
[ ]	정상적인 프로그램으로 가장하여 숨어 있는 바이러스 (복제 X)
[ ]	보안이 제거된 비밀통로로 무단 접근을 위한 통로(뒷문) ★탐지 방법: 무결성 검사, 로그 분석, SetID 파일, 열린 포트 검사
[ ]	내부 문서 파일 등을 암호화해 사용자가 열지 못하게 하고 이를 인질로 금전을 요구하는 데 사용되는 악성 프로그램
[ ]	지능형 지속 공격 / 특정 개인 및 조직을 타겟으로 침투 후 내부 서버 제어권 획득하여 무력화된 시스템 상의 데이터 수집
[ ] (큐싱)	QR코드를 통해 악성 앱 다운로드 유도하거나 설치
[ ]	Cross Site Scripting / 브라우저 스크립트 취약점을 악용 특정 링크 클릭 시 악성 스크립트 실행되어 개인 정보 탈취
[ ]	Cross-Site Request Forgery / 사용자가 공격자가 의도한 행위(수정, 삭제, 등록 등)를 특정 웹사이트에 요청하게 하는 공격
[ ] 공격	발견된 취약점의 존재를 공표하기 전에 해당 취약점으로 이용한 보안 공격 (해킹의 신속성)
[ ]	패킷을 엿보면서 계정 정보(ID/PW)를 가로채는 행위
[ ]	검증된 사람이 네트워크를 통해 데이터를 보낸 것처럼 데이터를 변조하여 접속을 시도하는 일종의 속임수 . IP Spoofing: 공격자가 자신의 IP 주소를 다른 주소로 위장 . ARP Spoofing: 공격자의 MAC 주소를 다른 컴퓨터 주소로 위장 . DNS Spoofing: DNS IP 주소를 중간에서 조작하여 위장
[ ]	서버에 접속하고 있는 클라이언트들의 <b>세션 정보를 가로채는 공격</b> <b>Reset 패킷을 통해 강제 종료시킨 후 재연결 시 침입자에게 연결</b>
[ ]	불특정 다수에게 메일 발송 후 가짜 위장 사이트로 유인한 후 금융기관 관련 개인 정보를 빼내는 행위
[ ]	문자 메시지(SMS) 통한 사용자 개인 신용 정보 탈취
[ ]	금융기관의 도메인 주소(DNS)를 중간에 가로채 사용자가 금융기관 사이트에 접속한 듯 착각하게 하여 개인정보 탈취
[ ]	사용자가 웹 URL 주소를 잘못 입력하는 실수를 악용
[ ]	CEO, 고위 경영진, 연예인 등 유명인사를 타겟으로 한 스피어 피싱
[ ]	키 입력 캐치 프로그램을 이용하여 개인정보를 빼내어 악용하는 행위
[ ]	할당된 메모리의 범위를 넘어선 위치에서 자료를 사용할 때 발생
[ ] 공격 ([ ] Attack)	임의의 SQL 문을 주입하고 실행되게 하여 데이터베이스가 비정상적인 동작을 하도록 조작하는 코드 삽입 공격
[ ] Attack	무작위 대입 공격 / 가능한 모든 문자열 조합으로 암호키 탈취
[ ] Attack	타겟이 주로 방문하는 웹 사이트 감염시킨 후 해당 타겟이 감염된 웹 사이트 방문할 때까지 기다리는 공격
[ ]	블루투스 연결 취약점을 이용한 공격
[ ]	공격자가 여러 가지의 경로로 수집한 사용자들의 로그인 인증 정보(Credential)를 다른 사이트의 계정 정보에 무작위 대입(Stuffing)
[ ]	타겟 기업을 침해하기 위해 보안이 더 취약한 협력사, 파트너 기업의 네트워크를 해킹
[ ]	컴퓨터 보안에 있어서 인간 상호 작용의 깊은 신뢰를 바탕으로 사람들을 속여 보안 절차를 깨뜨리기 위한 비기술적 침입 수단
[ ]	. 권한이 없는 사용자가 접근할 수 없는 영역에 접근하여 시스템을 제어하도록 설계된 <b>악성 소프트웨어의 모음</b> . 해킹에 사용되는 기능들을 제공하는 프로그램들의 모음





## ■ 웹 관련 용어

- 하이퍼링크**: 다른 문서/그림으로 이동할 수 있는 연결을 가지고 있는 텍스트
- 마크업 언어**: 하이퍼텍스트 및 웹 페이지를 만들 수 있는 마크업 언어
- 메타태그**: 인터넷 상에서 특정 웹 페이지의 위치를 나타내는데 사용되는 문자열
- 이메일 클라이언트**: 이메일에서 텍스트 외 이미지/비디오/오디오 등 여러 유형의 파일을 전송 가능

## ■ 웹 서비스

<b>WWW</b>	<b>월드 와이드 웹</b> 다른 컴퓨터 내 데이터나 서비스를 호출하기 위한 통신규약 프로토콜 XML 기반 메시지를 다른 컴퓨터 네트워크 상에서 교환
<b>XML</b>	<b>Extensible Markup Language</b> (웹 서비스 기술언어) 웹 서비스에서 제공하는 서비스에 대한 정보를 XML 기반으로 기술 (서비스 제공 장소, 서비스 메시지 포맷, 프로토콜 등)
<b>UDDI</b>	<b>Universal Description, Discovery and Integration</b> 전역 비즈니스 레지스트리 / 웹 서비스 관련 정보 공개 및 탐색 → 웹 서비스 제공자가 UDDI라는 온라인 저장소에 자신들의 서비스를 등록/저장하고, 소비자들은 그 저장소에 접근 후 원하는 서비스 탐색

## ■ 인터페이스 구현 기술

<b>HTML</b>	<b>HyperText Markup Language</b> HTML 문법의 비호환성과 SGML의 복잡성 해결하기 위해 개발
<b>JavaScript</b>	<b>JavaScript</b> , XML을 이용한 비동기식으로 웹 페이지의 일부 콘텐츠를만 Reload 해오는 방식 → HTML 보다 다양한 작업을 웹 페이지에서 구현 가능
<b>SOAP</b>	<b>Simple Object Access Protocol</b> 속성-값 쌍 (Attribute-Value Pairs) 으로 이뤄진 데이터 오브젝트를 전달하기 위해 사용하는 개방형 표준 포맷 → <b>SOAP</b> 에서 많이 사용 / XML을 대체하는 주요 데이터 포맷

## ■ 인터페이스 구현 검증 도구

<b>JUnit</b>	Java, C++, Net 등 다양한 언어 지원하는 단위 테스트 프레임워크
<b>JUnit4</b>	자바 프로그래밍 언어용 유닛 테스트 프레임워크 같은 테스트 코드를 여러 번 작성하지 않게 함
<b>TestNG</b>	서비스 호출, 컴포넌트 재사용 등 다양한 환경을 지원 크로스 플랫폼, 분산 소프트웨어 테스트 환경을 조성
<b>JUnit5</b>	웹 기반 테스트케이스 설계/실행/결과 확인 지원
<b>JUnit4</b>	STAF의 재사용 및 확장성 + Fitness의 협업 기능 통합 (현재는 폐기됨)
<b>JUnit4</b>	다양한 브라우저 및 개발 언어를 지원하는 웹 어플리케이션 테스트
<b>JUnit4</b>	Ruby 언어 기반 어플리케이션 테스트 프레임워크

## ■ 트리 순회 방법

<b>전위 순회</b> (Pre-order)	 A - <b>전위 순회</b> - J
<b>중위 순회</b> (In-order)	 D - <b>중위 순회</b> - J
<b>후위 순회</b> (Post-order)	 D - <b>후위 순회</b> - A

## ■ 클라우드 서비스

<b>클라우드 컴퓨팅</b>	<b>Cloud Computing</b> 물리적 자원(서버/네트워크/스토리지)을 가상화
<b>클라우드 서비스</b>	<b>Cloud Service</b> 응용프로그램 개발 시 필요한 플랫폼(OS/미들웨어/런타임) 제공
<b>클라우드 서비스</b>	<b>Cloud Service</b> 사용자에게 제공되는 소프트웨어(데이터/앱)를 가상화하여 제공
<b>클라우드 서비스</b>	<b>Cloud Service</b> 블록체인 기반 서비스 개발/관리의 편의성을 제공하는 클라우드
<b>클라우드 서비스</b>	<b>Cloud Service</b> 여러 모델의 클라우드 시스템 기반으로 보안 서비스를 제공
<b>클라우드 서비스</b>	<b>Cloud Service</b> 각종 업무용 운영체제/SW 등을 클라우드 방식으로 제공 (재택근무 활용)
<b>클라우드 서비스</b>	<b>Cloud Service</b> 사용자가 앱 개발 및 런칭에 관련한 하부 구조의 복잡한 빌드, 유지보수 없이 애플리케이션 기능을 개발, 실행, 관리할 수 있도록 플랫폼을 제공

## ■ RAID (Redundant Array of Independent Disks)

- 여러 개의 디스크를 한 개의 디스크처럼 관리하는 기술 (서버에 주로 사용)
- 데이터 안정성 높고 복구 용이함, 빠른 전송 속도
- Level값이 클수록 저장장치 높은 신뢰성/효율성**
  - ① **스트라이핑** - 데이터를 여러 개의 디스크에 분할하여 저장  
→ 하나라도 손상 시 데이터 복구 불가
  - ② **미러링** - 데이터를 2개의 디스크에 동일하게 저장  
→ 한 쪽 손상 시 다른 쪽 이용하여 복구 가능

<b>RAID 0</b>	스트라이핑 방식 / 중복 저장 X, <b>에러검출 X</b>
<b>RAID 1</b>	<b>미러링 방식</b> / 중복 저장 O, 높은 신뢰도
<b>RAID 2</b>	스트라이핑 방식 / <b>해밍코드</b> 활용 에러 검증
<b>RAID 3</b>	스트라이핑 방식 / <b>패리티(8bit)</b> 를 <b>에러 검증</b> , 바이트 단위로 데이터 저장
<b>RAID 4</b>	RAID 3과 동일하나, 데이터를 <b>블록 단위</b> 로 나눠 저장
<b>RAID 5</b>	스트라이핑 방식 / <b>패리티 블록</b> 을 <b>각 디스크마다 분산 저장</b>
<b>RAID 6</b>	스트라이핑 방식 / 패리티 블록을 <b>이중 구조</b> 로 구축

## ※ **클라우드 서비스**

- 기업 내 운영되는 플랫폼 및 애플리케이션 간의 정보전달, 연계, 통합 수행
- Point to Point, Hub&Spoke, Message Bus, Hybrid 형태로 구성

## ※ **클라우드 서비스**

- 입력 데이터를 프로세스가 처리하기 전에 미리 처리하여 프로세스 처리 시간을 줄여주는 프로그램

## ■ 디지털 저작권 관리 (Digital Rights Management)

- 콘텐츠 제공자/분배자/소비자, 패키지 (배포가능한 형태로 암호화하는 프로그램)
- 클리어링 하우스 (Clearing House): 사용 권한, 라이선스 발급, 결제 관리
- DRM 컨트롤러: 배포된 콘텐츠의 이용 권한을 통제하는 프로그램
- 보안 컨테이너: 콘텐츠 원본을 안전하게 유통하기 위한 전자적 보안 장치  
→ 기술 요소: **암호화 / 키 관리 / 식별 기술 / 저작권 표현 / 암호화 파일 생성 정책 관리 / 크랙 방지 / 인증**

## ■ UNIX / LINUX 기본 명령어

cat	파일 내용을 화면에 표시	fsck	파일 시스템 검사/보수
chdir	현재 사용할 디렉터리 위치 변경	getpid	자신의 프로세스 아이디 호출
chmod	파일의 보호 모드 설정	getppid	부모 프로세스 아이디 호출
chown	소유자 변경	ls	디렉터리 내 파일 목록 확인
cp	파일 복사	rm	파일 삭제
exec	새로운 프로세스 수행		
fork	새로운 프로세스 생성	wait	상위 프로세스가 하위 프로세스 종료 등의 event 기다림





## ■ [ ] (ISO 15504) → 6단계 / 소프트웨어 처리 개선 및 능력 평가 기준

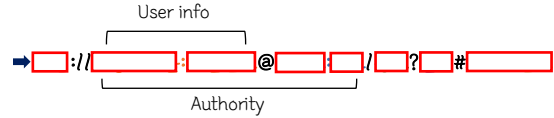
1) 불완전	프로세스가 구현되지 않거나 목적 달성 불가
2) 수행	목적은 달성하지만, 계획/축적 불가
3) 관리	프로세스 수행이 계획되고 관리됨
4) 확립	표준 프로세스를 이용하여 계획/관리
5) 예측 가능	표준 프로세스 능력에 대해 정량적 이해/성능 예측
6) 최적	정의된 프로세스와 표준 프로세스가 지속적으로 개선

## ■ IT 용어

[ ]	Stack 상 일정 주소 번지에 프로그래머가 유도하는 Canary를 심고, 스택이 붕괴/변조된 경우에 오버플로우 상태로 가정하여 Canary 체크 후 프로그램 실행을 비정상적으로 중단시키는 방법
[ ]	컨테이너 기술을 자동화하여 쉽게 사용하는 오픈소스 프로젝트 SW 컨테이너 안에 응용 프로그램 배치 자동화 역할
[ ]	JAVA에서 암호화/복호화 기능을 제공하는 컨테이너
[ ]	암호화 기법으로 문자열의 위치를 바꾸는 방법
[ ]	2015년 공개된 구글 브레인 팀의 기계 학습 오픈소스 SW Library
[ ]	일본/브라질에 상용 중인 디지털 TV 방송 기술
[ ]	위치 기반 소셜 네트워크 서비스
[ ]	국내 IT 서비스 경쟁력 강화 목적 개발된 개방형 클라우드 인프라 제어, 관리/실행/개발/서비스/운영 환경으로 구성
[ ] ( [ ] LAN)	물리적 배치와 상관없이 논리적으로 LAN 구성하는 기술 접속된 장비들의 성능 향상 / 보안성 증대 효과
[ ]	한 번의 로그인으로 다른 사이트 로그인도 허용하는 시스템
[ ]	. Message Queuing Telemetry Transport TCP/IP 기반 네트워크에서 발행-구독 기반의 메시징 프로토콜 푸시 기술 기반 경량 메시지 전송 프로토콜 (IBM 개발 주도)
[ ]	동일한 패스워드들을 다른 암호 값으로 저장되도록 덧붙이는 무작위의 값 → 같은 패스워드임에도 다른 결과 산출
[ ]	N개의 서로 다른 단말기에서 동일 콘텐츠를 자유롭게 이용
[ ]	. Address Space Layout Randomization 프로그램 실행마다 스택/힙/라이브러리 주소를 랜덤화하여 공격자로 하여금 메모리 상 주소 예측을 어렵게 함
[ ] [ ]	서로 다른 네트워크 대역에 있는 호스트를 상호 간에 통신할 수 있도록 해주는 네트워크 장비
[ ]	서버에 열린 포트 정보를 스캐닝하여 보안취약점을 찾는 도구
[ ]	크래커가 침입하여 백도어를 만들어 놓거나, 설정파일을 변경했을 때 분석하는 도구
[ ]	정보 기술을 활용하여 전력망 지능화, 고도화함으로써 고품질의 전력서비스를 제공하고 에너지 이용 효율을 극대화
[ ] (SOA)	구성 계층 : 표현 / 업무 프로세스 / 서비스 중간 / 애플리케이션 / 데이터 저장
[ ]	물리적 사물을 가상화하여(twin) 실제 자산의 특성 정보를 구현 → 자산 최적화 / 돌발 사고 최소화 / 생산성 증가
[ ]	웹에서 제공하는 정보/서비스를 이용하여 새로운 SW 제작
[ ] Network	차세대 이동통신, 홈네트워킹 등 특수 목적을 위한 새로운 방식의 네트워크 기술 / 대규모 네트워크 생성에 최적화
[ ]	여러 개의 독립된 통신장치가 블루투스 기술이나 UWB 통신 기술을 사용하여 통신망을 형성하는 무선 네트워크 기술

## ■ URL 구조

<https://ryan:2457@www.dream.com:443/index?id=5&page=1#bottom>



[ ]	사용할 프로토콜 명시 (리소스 요청 및 접근 방법) ex. http, https, ftp, mailto(이메일), rtsp(스트리밍) 등
사용자 정보	User info : 사용자 이름 + 비밀번호
[ ]	도메인명 또는 IP 주소
[ ]	Socket(소켓) 구분 역할 생략 시 http는 80, https는 443로 기본값 포트번호 설정
[ ]	호스트에서 제공하는 자원의 경로 (계층적 구조)
[ ]	Query parameter 또는 Query string으로 불림 key=value 형태, &로 질의 추가 (ex. ?directoryNo=0&currentPage=1)
[ ]	내부 북마크 기능 (웹문서 해시태그) / 해시태그로 이동을 원하는 요소의 id를 링크로 연결 시 스크롤 이동없이 해당 위치로 바로 이동 가능



## ■ SQL (Structured Query Language) 분류

### ① 데이터 정의어 (DDL: Data Definition Language) → 논리/물리적 데이터 구조 정의

<input type="text"/> (생성)	CREATE DOMAIN / SCHEMA / TABLE / VIEW / INDEX → 생성
<input type="text"/> (변경)	TABLE 이름 변경 → ALTER TABLE / 컬럼 추가
<input type="text"/> (삭제)	DROP DOMAIN / SCHEMA / TABLE / VIEW / INDEX → 삭제 ※ CASCADE : 참조하는 모든 개체 함께 제거 ※ RESTRICT : 제거할 요소를 다른 개체가 참조 시 제거 취소

### ② 데이터 조작어 (DML: Data Manipulation Language)

SELECT (검색)	SELECT <input type="text"/> 테이블명 [WHERE 조건];
INSERT (삽입)	INSERT <input type="text"/> 테이블명 VALUES 데이터;
DELETE (삭제)	DELETE <input type="text"/> 테이블명 [WHERE 조건];
UPDATE (변경)	UPDATE 테이블명 <input type="text"/> 속성명 = 데이터 [WHERE 조건];

#### ※ SELECT 구문 예시

SELECT [DISTINCT] 필드 이름 FROM 테이블명 [ <input type="text"/> 조건식] [ <input type="text"/> BY 필드이름] [ <input type="text"/> 그룹 조건식] [ <input type="text"/> BY 정렬]	SELECT 이름 FROM 학생 WHERE 수학>=80; GROUP BY 학년 HAVING COUNT(*)>=2 ORDER BY 나이 <input type="text"/> or 생략 ORDER BY 나이 <input type="text"/>	이름을 산출 / 학생 테이블에서 수학이 80 이상인 자료만 학년 필드를 그룹으로 묶어서 레코드 개수가 2이상인 그룹만 나이 오름차순 정렬 (기본) 나이 내림차순 정렬
---	---	---

※  : 중복 레코드 제거한 조회 결과 출력

OR	① 부서='경리' OR 부서='영업' ② <input type="text"/> ('경리', '영업')
AND	① 생일=#2001-1-1# AND 생일<=#2002-12-31# ② <input type="text"/> #2001-1-1# AND #2002-12-31#
만능문자	LIKE <input type="text"/> → 박으로 시작되는 모든 글자 LIKE <input type="text"/> → 은으로 끝나는 두 글자
빈 칸	IS NULL      부정문      NOT

### ③ 데이터 제어어 (DCL: Data Control Language)

→ 데이터 보안/복구/병행수행 제어무결성 유지

<input type="text"/>	수행된 결과를 실제 물리적 디스크로 저장 후 작업의 정상 완료 통보
<input type="text"/>	COMMIT 수행되지 않은 작업은 취소하고, 이전 상태로 원복 ※ SAVEPOINT : 트랜잭션 내 ROLLBACK할 저장 위치 지정
<input type="text"/>	데이터베이스 사용자에게 사용 권한 부여 → <input type="text"/> 권한 리스트 <input type="text"/> 개체 <input type="text"/> 사용자 [WITH GRANT OPTION] 부여받은 권한을 다른 사용자에게 재부여
<input type="text"/>	데이터베이스 사용자의 사용 권한 취소 → <input type="text"/> [GRANT OPTION FOR] 권한 리스트 <input type="text"/> 개체 <input type="text"/> 사용자 다른 사용자에게 권한 부여 가능한 권한을 취소

#### [실전 예제 - 1] 23년 3회 실기

다음 두 테이블에 대해 SQL문을 실행하였을 때, 나타나는 결과 작성 (속성명 + 값 출력)

A	B	A	B
2	X	1	Y
1	Y	2	Z
3	Z	4	X
[L]		[R]	

SELECT A FROM L  
UNION  
SELECT A FROM R  
ORDER BY A DESC;




\* UNION : 두 집합의 합 (중복은 제거) → 내림차순으로 정렬

#### [실전 예제 - 2] 23년 2회 실기

다음 처리 조건에 맞는 쿼리문 작성

- 학생 테이블에 학번이 1415245, 이름이 '라이언', 학년이 2, 과목이 '수학', 연락처가 '010-1234-5678'인 학생의 정보를 입력
- 인용 부호가 필요한 경우 경우 작은 따옴표(' ') 사용
- 명령문 마지막 세미콜론(;)은 생략 가능

#### [학생] 테이블

속성명	데이터 타입	비고
학번	INT	PRIMARY KEY
이름	VARCHAR(20)	
학년	INT	
과목	VARCHAR(20)	
연락처	VARCHAR(20)	

#### [실전 예제 - 3] 23년 1회 실기

[학생] 테이블에서 학생 이름이 '지영'인 튜플을 삭제하는 쿼리 작성

#### [학생] 테이블

순번	학년	이름
1	1	나희
2	2	지영
3	2	소울
4	3	이안




★ 속성인 [이름]에는 따옴표 붙이지 않기!

#### [실전 예제 - 4] 23년 1회 실기

다음 테이블과 SQL문 결과 데이터를 확인하여 처리한 SQL문 작성  
(과목별 그룹을 묶었을 때 과목 평균이 90이상인 과목에 대해서만 출력)

#### [성적] 테이블

순번	과목	점수
1	데이터베이스	89
2	데이터베이스	94
3	네트워크	45
4	소프트웨어	68
5	네트워크	95
6	소프트웨어	74

#### 결과

과목	최소점수	최대점수
데이터베이스	89	94

- WHERE 사용 금지
- SELECT절에 별칭을 사용하여 작성
- 반드시 GROUP BY와 HAVING을 사용
- 집계함수를 사용
- SQL문 마지막 세미콜론(;) 생략 가능

#### [실전 예제 - 5] 22년 3회 실기

학생(STUDENT) 테이블에 재료가 학생이 50명, 전기과 학생이 100명, 컴퓨터공학과 학생이 50명 있다고 할 때, 아래 1~3번 SQL문 실행 결과로 표시되는 튜플의 수 구하기  
(DEPT 필드는 학과를 의미)

1번. SELECT DEPT FROM STUDENT;

2번. SELECT DISTINCT DEPT FROM STUDENT;

3번. SELECT COUNT(DISTINCT DEPT) FROM STUDENT WHERE DEPT = '재료과';



## [실전 예제 - 6] 22년 3회 실기

다음과 같이 테이블을 정의하고 튜플을 삽입하였을 때 아래 1, 2번의 SQL문 결과 작성

```
CREATE TABLE 부서 (
  부서코드 INT PRIMARY KEY,
  부서명 VARCHAR(20)
);
CREATE TABLE 직원 (
  직원코드 INT PRIMARY KEY,
  부서코드 INT
  직원명 VARCHAR(20)
  FOREIGN KEY(부서코드) REFERENCES 부서(부서코드)
  ON DELETE CASCADE
);

INSERT INTO 부서 VALUES(10, '영업팀');
INSERT INTO 부서 VALUES(20, '개발팀');
INSERT INTO 부서 VALUES(30, '기획팀');

INSERT INTO 직원 VALUES(1000, 10, '오지은');
INSERT INTO 직원 VALUES(2000, 10, '황나영');
INSERT INTO 직원 VALUES(3000, 20, '김새희');
INSERT INTO 직원 VALUES(4000, 20, '이슬기');
INSERT INTO 직원 VALUES(5000, 20, '성예은');
INSERT INTO 직원 VALUES(6000, 30, '박서희');
INSERT INTO 직원 VALUES(7000, 30, '김지현');
```

[부서] 테이블

[직원] 테이블

## 1번. SQL문

```
SELECT DISTINCT COUNT(부서코드) FROM 직원
WHERE 부서코드 = 20;
```

## 2번. SQL문

```
DELETE FROM 부서 WHERE 부서코드 = 20;
SELECT DISTINCT COUNT(부서코드) FROM 직원
```

→ 직원 테이블 내 부서코드가 부서 테이블 내 부서코드와 외래키로 연결되어 있기에 직원 테이블 내 부서코드가 20인 튜플들 같이 삭제 (DELETE CASCADE)

## [실전 예제 - 7] 22년 2회 실기

아래는 <제품>(제품명, 단가, 제조사) 테이블을 대상으로 “C” 제조사에서 생산한 제품들의 ‘단가’ 보다 높은 ‘단가’를 가진 제품의 정보를 조회하는 SQL문이다. 이때, 괄호 ( ) 안에 알맞은 답을 넣어 SQL문 완성하기

```
SELECT 제품명, 단가, 제조사 FROM 제품
WHERE 단가 > ( ) (SELECT 단가 FROM 제품 WHERE 제조사 = "C");
```

①

②

## [실전 예제 - 8] 22년 2회 실기

다음 <TABLE>을 참조하여 SQL문을 실행했을 때 출력 결과 구하기  
(<TABLE> 내 ‘NULL’은 값이 없음을 의미)

INDEX	COL1	COL2
1	2	NULL
2	5	8
3	3	5
4	7	3
5	NULL	3

```
SELECT COUNT(COL2)
FROM TABLE
WHERE COL1 IN (2, 3) OR COL2 IN (3, 5)
```

## [실전 예제 - 9] 22년 1회 실기

다음 SQL문을 보고 SQL문의 Score 결과가 내림차순으로 정렬되도록 괄호 채우기

[성적] 테이블

	NAME	SCORE
1	PARK	95
2	KIM	90
3	LEE	60

```
SELECT NAME, SCORE FROM 성적
( ① ) BY ( ② ) ( ③ )
```

▶ ①:  / ②:  / ③:

## [실전 예제 - 10] 21년 3회 실기

다음은 테이블에서 조건값을 실행한 화면으로 이에 대한 알맞은 결과값 구하기

[A] 테이블

NAME
MARTIN
SCOTT
SMITH

[B] 테이블

RULE
S%
%T%

[CNT] 테이블

```
SELECT COUNT(*) CNT FROM A CROSS JOIN B
WHERE A.NAME LIKE B.RULE
```

CROSS JOIN 후 테이블  
(JOIN해서 나올 수 있는 모든 행의 조합)

## [실전 예제 - 11] 21년 2회 실기

‘이’씨 성을 가진 사람의 이름을 내림차순으로 출력하는 SQL문의 괄호 채우기

```
SELECT ... FROM ... WHERE 이름 LIKE ( A ) ORDER BY 이름 ( B )
```

▶ A:  / B:  (정답)

## [실전 예제 - 12] 21년 2회 실기

학생 테이블의 점수가 90점 이상인 학생의 과목평가를 ‘A’로 수정하는 SQL문 작성

```
( A ) 학생
( B )
과목 평가 = 'A'
WHERE 점수 >= 90
```

▶ A:   
B:

## [실전 예제 - 13] 21년 2회 실기

학생 정보와 학과 정보를 조인하여 결과값을 출력하는 SELECT문 작성

```
SELECT * FROM 학생정보 A JOIN 학과정보 B
( ① ) A.학과 = B.( ② )
```

▶ ①:   
②:

## [실전 예제 - 14] 21년 1회 실기

다음 <TABLE>을 참조하여 SQL문을 실행했을 때 출력 결과 구하기

EMPNO	SAL
100	1000
200	3000
300	1500

```
SELECT COUNT(*) FROM 급여
WHERE EMPNO > 100 AND SAL >= 3000 OR EMPNO = 200;
```

▶

이 자료는 대한민국 저작권법의 보호를 받습니다. 작성된 모든 내용의 권리는 작성자에게 있으며, 작성자의 동의 없는 사용이 금지됩니다. 본 자료의 일부 혹은 전체 내용을 무단으로 복제/배포하거나 2차적 저작물로 재편집하는 경우, 5년 이하의 징역 또는 5천만원 이하의 벌금과 민사상 손해배상을 청구합니다.