**가. 소프트웨어 설계(20문제)**

1. 요구사항 확인
   1. [요구사항 확인 정리(1)](https://1d1cblog.tistory.com/141)
   2. [요구사항 확인 정리(2)](https://1d1cblog.tistory.com/142)
2. 화면 설계
   1. [화면 설계 정리(1)](https://1d1cblog.tistory.com/143)
   2. [화면 설계 정리(2)](https://1d1cblog.tistory.com/144)
3. 애플리케이션 설계
   1. [애플리케이션 설계 정리(1)](https://1d1cblog.tistory.com/145)
   2. [애플리케이션 설계 정리(2)](https://1d1cblog.tistory.com/146)
4. 인터페이스 설계
   1. [인터페이스 설계 정리(1)](https://1d1cblog.tistory.com/152)
   2. [인터페이스 설계 정리(2)](https://1d1cblog.tistory.com/153)

[1과목 예상 키워드 정리](https://1d1cblog.tistory.com/184)

**나. 소프트웨어 개발(20문제)**

1. 데이터 입출력 구현
   1. [데이터 입출력 구현 정리](https://1d1cblog.tistory.com/154)
2. 통합 구현
   1. [통합 구현 정리](https://1d1cblog.tistory.com/155)
3. 제품 소프트웨어 패키징
   1. [제품 소프트웨어 패키징 정리(1)](https://1d1cblog.tistory.com/157)
   2. [제품 소프트웨어 패키징 정리(2)](https://1d1cblog.tistory.com/158)
4. 애플리케이션 테스트 관리
   1. [애플리케이션 테스트 관리 정리(1)](https://1d1cblog.tistory.com/159)
   2. [애플리케이션 테스트 관리 정리(2)](https://1d1cblog.tistory.com/160)
   3. [애플리케이션 테스트 관리 정리(3)](https://1d1cblog.tistory.com/161)
5. 인터페이스 구현
   1. [인터페이스 구현 정리(1)](https://1d1cblog.tistory.com/162)
   2. [인터페이스 구현 정리(2)](https://1d1cblog.tistory.com/163)

[2과목 예상 키워드 정리](https://1d1cblog.tistory.com/185)

**다. 데이터베이스 구축(20문제)**

1. 논리 데이터베이스 설계
   1. [논리 데이터베이스 설계 정리(1)](https://1d1cblog.tistory.com/89)
   2. [논리 데이터베이스 설계 정리(2)](https://1d1cblog.tistory.com/90)
   3. [논리 데이터베이스 설계 정리(3)](https://1d1cblog.tistory.com/93)
   4. [논리 데이터베이스 설계 정리(4)](https://1d1cblog.tistory.com/94)
2. 물리 데이터베이스 설계
   1. [물리 데이터베이스 설계 정리(1)](https://1d1cblog.tistory.com/100)
   2. [물리 데이터베이스 설계 정리(2)](https://1d1cblog.tistory.com/101)
   3. [물리 데이터베이스 설계 정리(3)](https://1d1cblog.tistory.com/102)
   4. [물리 데이터베이스 설계 정리(4)](https://1d1cblog.tistory.com/106)
3. SQL 응용
   1. [SQL 응용 정리(1)](https://1d1cblog.tistory.com/95)
   2. [SQL 응용 정리(2)](https://1d1cblog.tistory.com/97)
4. SQL 활용
   1. [SQL 활용 정리(1)](https://1d1cblog.tistory.com/109)
   2. [SQL 활용 정리(2)](https://1d1cblog.tistory.com/111)
5. 데이터 전환

(책에 필요하면 첨가하겠음)

**라. 프로그래밍 언어 활용(20문제)**

1. 서버 프로그램 구현
   1. [서버 프로그램 구현 정리](https://1d1cblog.tistory.com/139)
2. 프로그래밍 언어 활용

(책에 필요하면 첨가하겠음)

1. 응용 SW 기초 기술 활용
   1. [응용 SW 기초 기술 활용 정리(1)](https://1d1cblog.tistory.com/112)
   2. [응용 SW 기초 기술 활용 정리(2)](https://1d1cblog.tistory.com/135)
   3. [응용 SW 기초 기술 활용 정리(3)](https://1d1cblog.tistory.com/137)
   4. [응용 SW 기초 기술 활용 정리(4)](https://1d1cblog.tistory.com/138)

**마. 정보시스템 구축 관리(20문제)**

1. 소프트웨어 개발 방법론 활용
   1. [소프트웨어 개발 방법론 활용 정리(1)](https://1d1cblog.tistory.com/164)
   2. [소프트웨어 개발 방법론 활용 정리(2)](https://1d1cblog.tistory.com/165)
2. IT 프로젝트 정보 시스템 구축 관리
   1. [IT 프로젝트 정보 시스템 구축 관리 정리(1)](https://1d1cblog.tistory.com/166)
   2. [IT 프로젝트 정보 시스템 구축 관리 정리(2)](https://1d1cblog.tistory.com/167)
   3. [IT 프로젝트 정보 시스템 구축 관리 정리(3)](https://1d1cblog.tistory.com/168)
3. 소프트웨어 개발 보안 구축
   1. [소프트웨어 개발 보안 구축 정리(1)](https://1d1cblog.tistory.com/170)
   2. [소프트웨어 개발 보안 구축 정리(2)](https://1d1cblog.tistory.com/171)
4. 시스템 보안 구축
   1. [시스템 보안 구축 정리](https://1d1cblog.tistory.com/172)

2020 정보처리기사 필기

## [1.1 요구사항 확인(1)](http://127.0.0.1:5503/141)

### 소프트웨어 생명 주기

**소프트웨어 생명 주기(Life Cycle)**

- 소프트웨어 개발 방법론의 바탕이 되어 소프트웨어를 개발하기 위해 정의하고 운용 유지보수 등의 과정을 각 단계별로 나눈 것

- 스프트웨어 개발 단계와 각 단계별 주요 활동 및 활동의 결과를 산출물로 표현

- 소프트웨어 생명 주기를 표현하는 형태를 소프트웨어 생명 주기 모형, 소프트웨어 프로세스 모형, 소프트웨어 공학 패러다임이라고 함

- 특정 모형을 선택하여 사용하거나 개별적인 모형을 사용할 수 있음

**폭포수 모형**

- 폭포수가 거슬러 올라갈 수 없듯이 이전 단계를 확실히 마무리하고 다음 단계로 진행하는 개발 방법론

- 소프트웨어 공학에서 가장 오래되고 폭넓게 사용된 생명 주기 모형

- 한 단계가 끝나야 다음 단계로 넘어갈 수 있는 선형 순차적 모형

- 매뉴얼을 작성해야 함

- 단계를 끝내고 다음 단계로 가기 위해서는 결과물이 명확히 나와야함

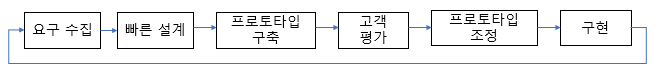
https://blog.kakaocdn.net/dn/cYHa5g/btqC9xAlfqH/mR6ekiDjKPhOBIBtLpszM1/img.png

**프로토타입 모형**

- 요구사항을 정확히 파악하기 위해 실제 개발될 소프트웨어에 대한 시제품을 만들어 최종 결과물을 예측하는 모형

- 폭포수 모델의 단점을 보완하기 위해 만들어진 모형

- 사용자와 시스템 사이 인터페이스에 중점을 두어 개발



**나선형 모형**

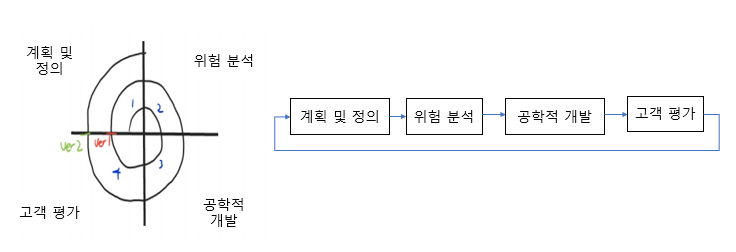
- 폭포수 모형과 프로토타입 모형의 장점에 위험 분석 기능을 추가한 모형

- 나선을 따라 돌듯이 여러 번의 개발 과정을 거쳐 점진적으로 완벽한 최종 소프트웨어를 개발

- 소프트웨어를 개발하면서 발생할 수 있는 위험을 관리하고 최소화하는 것이 목적

- 누락되거나 추가된 요구사항을 첨가할 수 있음

- 정밀하고 유지보수 과정이 필요 없음



**애자일 모형**

- 고객의 요구사항 변화에 유연하게 대응할 수 있도록 일정한 주기를 반복하면서 진행하는 모형

- 좋은 것을 빠르고 낭비 없게 만들기 위해 고객과의 소통에 초점을 맞춘 모든 방법론을 통칭

- 스프린트 또는 이터레이션이라고 불리는 짧은 개발 주기를 반복

- 반복되는 주기마다 결과물에 해단 평가와 요구 수용

- 요구사항에 우선순위를 부여하여 개발 진행

- 애자일 모형을 기반으로 하는 모형에는 스크럼, XP, 칸반, Lean, 크리스탈, ASD, FDD, DSDM 등이 있음

### 스크럼 기법

**스크럼의 개요**

- 팀이 중심이 되어 개발의 효율성을 높임

- 팀원 스스로가 팀을 구성하고 개발 작업에 대한 모든 것을 스스로 해결할 수 있어야 함

**스크럼의 구성 요소**

- 제품 책임자

    -> 개발될 제품에 대한 이해도가 높고 요구사항을 책임지고 의사 결정할 사람

    -> 개발 의뢰자나 사용자가 담당

    -> 이해관계자들의 의견을 종합하여 제품에 대한 요구사항을 작성

    -> 백로그를 작성

    -> 팀원들은 백로그에 스토리는 추가할 수 있지만 우선순위를 지정하는 것은 제품 책임자임

    -> 테스트를 수행하면서 주기적으로 요구사항의 우선순위 갱신

- 스크럼 마스터

    -> 팀이 잘 수행할 수 있도록 객관적인 시각에서 조언을 해주는 가이드 역할

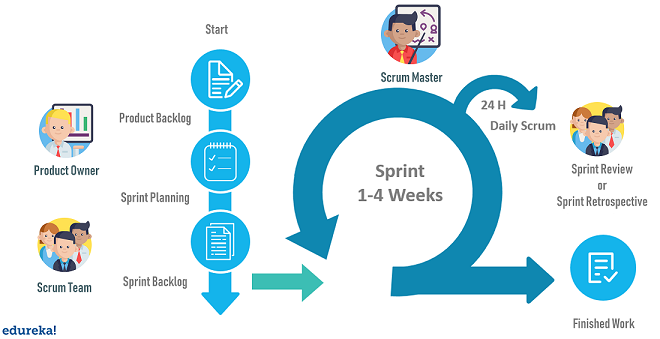
    -> 일일 스크럼 회의를 주관하여 진행 사항을 점검하고 개발과정에서 발생된 장애 요소를 공론화하여 처리

- 개발팀

    -> 개발자 외에도 디자이너, 테스터 등 제품 개발을 위해 참여하는 모든 사람

    -> 보통 최대 인원은 7~8명이 적당

**스크럼 개발 프로세스**

출처 : https://www.edureka.co/blog/agile-scrum-tutorial/

- 제품 백로그(Product Backlog)

    -> 개발에 필요한 요구사항을 우선순위에 따라 나열한 목록

    -> 새롭게 도출되는 요구사항으로 인해 지속적 업데이트

    -> 작성된 사용자 스토리를 기반으로 릴리즈 계획을 수립

- 스프린트 계획 회의(Sprint Planning Meeting)

    -> 이번 스프린트에서 수행할 작업을 대상으로 단기 일정 수립

    -> 처리할 요구사항을 개발자들이 나눠서 작업할 수 있도록 태스크라는 작업 단위로 나눠 개발자 별로 수행할 작업 목록인 스프린트 백로그(Sprint Backlog) 작성

- 스프린트(Sprint)

    -> 실제 개발 작업을 진행하는 과정

    -> 스프린트 백로그에 작성된 태스크를 대상으로 작업 시간이나 양을 추정한 후 개발 담당자에게 할당

    -> 태스크를 할당할 때는 개발자가 원하는 태스트를 직접 선별하여 담당할 수 있도록 하는 것이 좋음

    -> 할당된 태스크는 할 일, 진행 중, 완료의 상태를 가짐

- 일일 스크럼 회의(Daliy Scrum Meeting)

    -> 모든 팀원이 매일 약속된 시간에 짧은 시간동안 진행 상황을 점검

    -> 스크럼 마스터는 발견된 장애 요소를 해결할 수 있도록 도와줌

    -> 남은 작업 시간은 소멸 차트에 표시

- 스프린트 검토 회의(Spring Review)

    -> 부분 또는 완성 제품이 요구사항에 잘 부합되는지 사용자가 포함된 참석자 앞에서 테스트 수행

- 스프린트 회고(Sprint Retrospective)

    -> 스프린트가 끝나고 정해놓은 규칙을 잘 준수했는지, 개선할 점은 없는지 등을 점검하고 수행

### XP(eXtreme Programming) 기법

**XP의 개요**

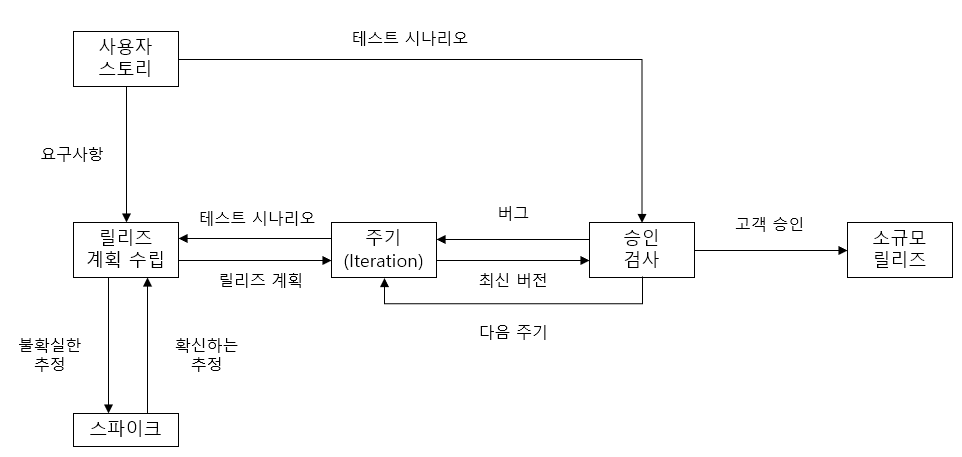
- 수시로 발생하는 고객의 요구사항에 유연하게 대응하기 위해 고객의 참여와 개발 과정의 반복을 극대화하여 개발 생산성을 향상시키는 기법

- 짧고 반복적인 개발주기, 단순한 설계, 고객의 적극적인 참여를 통해 빠르게 개발하는 것이 목적

- 릴리즈의 기간을 짧게 반복하면서 요구사항 반영에 대한 가시성을 높임

- XP의 5가지 핵심 가치 : 의사소통, 단순성, 용기, 존중, 피드백

**XP 개발 프로세스**



- 사용자 스토리

    -> 고객의 요구사항을 간단한 시나리오로 표현

- 릴리즈 계획 수립

    -> 몇 개의 스토리가 적용되어 부분적으로 기능이 완료된 제품을 제공하는 것에 대한 계획 수립

- 스파이크

    -> 요구사항의 신뢰성을 높이고 기술 문제에 대한 위험을 감소시키기 위해 별도로 만드는 프로그램

- 이터레이션

    -> 하나의 릴리즈를 더 세분화하여 한 단위

- 승인 검사

    -> 하나의 이터레이션 안에서 계획된 릴리즈 단위의 부분 완료 제품이 구현되면 수행하는 테스트

    -> 사용자 스토리 작성 시 함께 기재한 테스트 사항에 대해 고객이 직접 수행

- 소규모 릴리즈

    -> 고객의 반응을 기능별로 확인하고 고객의 요구사항에 유연하게 대응

    -> 진행된 이터레이션이 모두 완료되면 고객에 의한 최종 테스트 수행 후 최종 결과물을 고객에게 전달

**XP의 주요 실천 방법**

- Pair Programming : 다른 사람과 함께 프로그래밍 수행

- Test-Driven Development : 실제 코드 작성 전 테스트 케이스를 먼저 작성하여 무엇을 해야할지 파악

- Whole Team : 개발에 참여하는 모든 구성원은 각기 역할이 있어 책임을 다해야 함

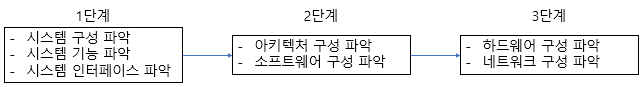
- Continuous Intergration : 모듈 단위로 나눠 개발한 코드는 하나의 작업이 마무리되면 지속적으로 통합

- Design Improvement / Refactoring : 프로그램 기능의 변경 없이 시스템을 재구성

- Small Release : 릴리즈 기간을 짧게 하여 고객의 요구 변화에 신속하게 대응

### 현행 시스템 파악

**현행 시스템 파악 절차**



- 1단계

    -> 시스템 구성 파악 : 조직의 업무를 담당하는 기간 업무와 이를 지원하는 업무로 구분하여 나타낸 구성을 파악

    -> 시스템 기능 파악 : 현재 제공하는 기능들을 주요, 하부, 세부 기능으로 구분하여 계층형으로 표시

    -> 시스템 인터페이스 파악 : 주고받는 데이터의 종류, 형식, 프로토콜, 연계 유형, 주기 등을 명시

- 2단계

    -> 아키텍쳐 구성 파악 : 어떠한 기술 요소들이 사용되는지 최상위 수준에서 계층별로 표현한 구성 파악

    -> 소프트웨어 구성 파악 : 업무 처리를 위해 설치되어 있는 소프트웨어의 제품명, 용도, 라이선스 적용 방식 등을 명시

- 3단계

    -> 하드웨어 구성 파악 : 단위 업무 시스템들이 운용되는 서버의 주요 사양과 수량 및 이중화 적용 여부 명시

    -> 네트워크 구성 파악 : 서버의 위치, 서버 간의 네트워크 연결 방식을 네트워크 구성도로 작성

## [1.1 요구사항 확인(2)](http://127.0.0.1:5503/142)

2020. 4. 6. 21:34

[20-20정보처리기사필기정리](http://127.0.0.1:5503/tag/20-20%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 개발 기술 환경 파악

**개발 기술 환경의 정의**

- 개발하고자 하는 소프트웨어와 관련된 O/S, DBMS, Middle Ware 등을 선정할 때 고려해야 할 사항을 기술하고 오픈 소스 사용 시 주의해야 할 내용을 제시

**운영체제(Operating System)**

- 컴퓨터 시스템 자원을 효율적으로 관리하여 사용자가 컴퓨터를 편리하고 사용할 수 있도록 환경을 제공하는 소프트웨어

- 요구사항 식별 시 고려사항 : 가용성, 성능, 기술 지원, 주변 기기, 구축 비용

**데이터베이스 관리 시스템(DBMS)**

- 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해주고 관리해주는 소프트웨어

- 요구사항 식별 시 고려사항 : 가용성, 성능, 기술 지원, 상호 호환성, 구축 비용

**웹 애플리케이션 서버(WAS)**

- 사용자의 요구에 따라 변하는 동적인 컨텐츠를 처리하기 위해 사용되는 미들웨어

- 요구사항 식별 시 고려사항 : 가용성, 성능, 기술 지원, 구축 비용

**오픈 소스(Open Source)**

- 누구나 제한없이 사용할 수 있도록 소스 코드를 공개한 것

- 라이선스를 만족하는 소프트웨어

- 요구사항 식별 시 고려사항 : 라이선스의 종류, 사용자의 수, 기술의 지속 가능성

### 요구사항 정의

**요구사항의 개념과 특징**

- 소프트웨어가 어떤 문제를 해결하기 위해 제공하는 서비스에 대한 설명과 정상적으로 운영되는데 필요한 제약조건 등

- 요구사항이 제대로 정의되어야 이를 토대로 이후 과정의 목표와 계획의 수립이 가능

**요구사항의 유형**

- 기술하는 내용에 따른 분류

    -> 기능 요구사항 : 시스템이 무엇을 하고 어떤 기능을 하는지에 대한 사항

    -> 비기능 요구사항 : 품질이나 제약사항에 대한 사항

- 기술 관점과 대상의 범위에 따른 분류

    -> 사용자 요구사항 : 사용자의 관점에서 본 시스템이 제공해야 할 사항

    -> 시스템 요구사항 : 개발자의 관점에서 본 시스템 전체가 사용자와 다른 시스템에 제공해야 할 사항

**요구사항 개발 프로세스**

https://blog.kakaocdn.net/dn/wknni/btqDaVOy0ON/A8Gdqxrb45XjHdfOY4nKck/img.png

- 요구사항 도출

    -> 요구사항이 어디에 있고 어떻게 수집할지 식별하고 이해하는 과정

    -> 인터뷰, 설문, 브레인스토밍, 워크샵, 프로토타이핑, 유스 케이스 등

    -> 브레인스토밍 : 3인 이상이 자유롭게 아이디어를 산출해 내는 방법

    -> 유스케이스 : 사용자의 요구사항을 기능 단위로 표현

- 요구사항 분석

    -> 개발 대상에 대한 사용자의 요구사항 중 명확하지 않거나 모호하여 이해되지 않는 부분을 발견하고 걸러내는 과정

- 요구사항 명세

    -> 요구사항을 분석한 후 승인될 수 있도록 문서화하는 과정

    -> 소프트웨어 요구사항 명세서 : 소프트웨어가 반드시 제공해야 하는 기능, 특징, 제약조건을 명시

- 요구사항 확인

    -> 개발 자원을 요구사항에 할당하기 전에 요구사항 명세서가 정확하게 작성되었는지 검토하는 과정

### 요구사항 분석 기법

**요구사항 분류**

- 요구사항을 명확히 확인할 수 있도록 정해진 기준으로 분류

    -> 기능/비기능 요구사항 분류

    -> 제품/과정으로 분류

    -> 우선순위로 분류

**개념 모델링**

- 요구사항의 현실 세계의 상황을 단순화하여 개념적으로 표현하는 과정

- 실세계 문제에 대한 모델링은 요구사항 분석의 핵심

- 개체와 개체 간의 관계 및 종속성을 반영

- 개념 모델은 다양하게 표현

- 주로 UML을 사용

**요구사항 할당**

- 요구사항을 만족시키기 위한 구성 요소를 식별

**요구사항 협상**

- 요구사항이 서로 충돌될 경우 해결하는 과정

- 적절한 기준점을 찾아 합의하는게 좋음

- 서로 충돌하는 경우 우선순위를 부여하면 해결에 도움이 됨

**정형 분석**

- 구문과 의미를 같은 정형화된 언어를 이용해 요구사항을 수학적 기호로 표현하고 분석하는 과정

### 요구사항 확인 기법

**요구사항 검토**

- 문서화된 요구사항을 훑어보면서 확인하는 과정

- 시스템 정의서, 시스템 사양서, 소프트웨어 요구사항 명세서 등을 완성한 시점에 이루어짐

**프로토타이핑**

- 초기 도출된 요구사항을 토대로 프로토타입을 만든 후 개발이 진행되는 동안 도출되는 요구사항을 반영하면서 지속적으로 프로토타입을 재작성하는 과정

- 장점

    -> 빠르고 반복되는 제작을 할 수 있어 발전된 결과물을 얻을 수 있음

    -> 최종 시스템을 완성하기 전에 추가 또는 변경된 요구사항에 대한 피드백 가능

- 단점

    -> 사용자의 관심이 핵심에서 벗어나 프로토타입 제작에만 집중될 수 있음

    -> 지속적이고 반복적인 개선에 대한 비용이 부담될 수 있음

**모델 검증**

- 요구사항 분석 단계에서 개발된 모델이 요구사항을 충족하는지 검증하는 과정

**인수 테스트**

- 사용자가 실제로 사용될 환경에서 요구사항이 모두 충족되는지 사용자 입장에서 확인하는 과정

### UML(Unified Modeling Language)

**UML의 개요**

- 시스템 개발 과정에서 시스템 개발자와 고객 또는 개발자 상호 간의 의사소통이 원활하게 이루어지도록 표준화한 객체지향 모델링 언어

- 객체지향 방법론의 장점을 통합하였으며 OMG(Object Management Group)에서 표준으로 지정

- 구성 요소 : 사물, 관계, 다이어그램

**사물**

- 모델을 구성하는 가장 중요한 기본 요소

- 다이어그램 안에서 관계가 형성될 수 있는 대상

- 구조 사물

    -> 시스템의 개념적, 물리적 요소 표현

    -> 클래스, 유스케이스, 컴포넌트, 노드 등

- 행동 사물

    -> 시간과 공간에 따른 요소들의 행위 표현

    -> 상호작용, 상태 머신 등

- 그룹 사물

    -> 요소들의 그룹으로 묶어서 표현

    -> 패키지

- 주해 사물

    -> 부가적인 설명이나 제약조건 등 표현

    -> 노트

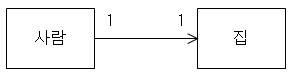
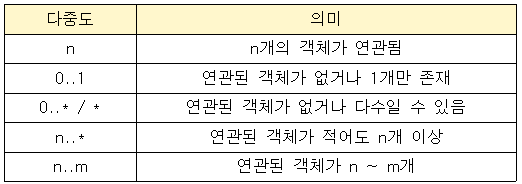
**관계**

- 사물과 사물 사이의 연관성 표현

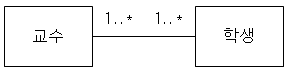
- 연관 관계

    -> 2개 이상의 사물이 서로 관련되어 있음을 표현

    -> 실선과 화살표로 연결하여 표현하지만 양방향 관계의 경우 화살표 없이 실선으로 연결하여 표현



사람과 집이 1:1 관계

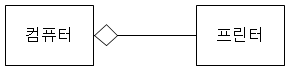


교수는 1명 이상의 학생을 가르치고 학생은 1명 이상의 교수에게 가르침을 받음

- 집합 관계

    -> 하나의 사물이 다른 사물에 포함되어 있는 관계

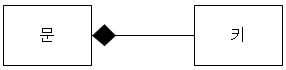
    -> 부분(포함되는 쪽)에서 전체(포함하는 쪽)로 속이 빈 마름모를 연결하여 표현

프린터는 컴퓨터에 연결해서 사용할 수 있음

- 포함 관계

    -> 집합 관계의 특수한 형태로 포함하는 사물의 변화가 포함되는 사물에게 영향을 미치는 관계

    -> 부분(포함되는 쪽)에서 전체(포함하는 쪽)로 속이 채워진 마름모를 연결하여 표현



문을 열 수 있는 키는 하나이고 해당 키로 다른 문은 열 수 없음

- 의존 관계

    -> 사물 사이에 연관은 있으나 필요에 의해서 서로에게 영향을 주는 짧은 시간 동안만 연관을 유지하는 관계

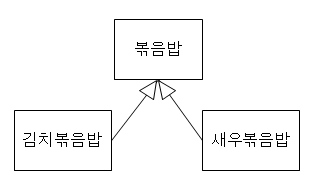
    -> 영향을 주는 사물이 영향을 받는 사물 쪽으로 점선 화살표 연결

출석률은 학점을 낼 때 영향을 미침

- 일반화 관계

    -> 하나의 사물이 다른 사물에 비해 일반적인지 구체적인지 표현

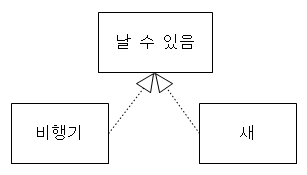
    -> 구체적인 사물에서 일반적인 사물 쪽으로 속이 빈 화살표를 연결



- 실체화 관계

    -> 사물이 할 수 있거나 해야 하는 기능으로 서로를 그룹화할 수 있는 관계

    -> 사물에서 기능 쪽으로 속이 빈 점선 화살표 연결



**다이어그램**

- 사물과 관계를 도형으로 표현

- 정적 모델링에서는 주로 구조적 다이어그램을 사용하고 동적 모델링에서는 주로 행위 다이어그램 사용

- 구조적 다이어그램

    -> 클래스 다이어그램 : 클래스, 클래스가 가지는 속성, 클래스 사이 관계 표현

    -> 객체 다이어그램 : 인스턴스를 특정 시점의 객체와 객체 사이의 관계로 표현

    -> 컴포넌트 다이어그램 : 구현 단계에서 사용되며 컴포넌트 간의 관계나 인터페이스를 표현

    -> 배치 다이어그램 : 구현단계에서 사용되며 결과물, 프로세스, 컴포넌트 등 물리적 요소들의 위치 표현

    -> 복합체 구조 다이어그램 : 복잡한 구조를 가지는 클래스 혹은 컴포넌트의 내부 구조 표현

    -> 패키지 다이어그램 : 유스케이스나 클래스 등의 모델 요소들을 그룹화한 패키지들의 관계 표현

- 행위 다이어그램

    -> 유스케이스 다이어그램 : 사용자의 요구를 분석하여 기능 모델링 작업에 사용됨

    -> 시퀀스 다이어그램 : 상호 작용하는 시스템이나 객체들이 주고받는 메시지 표현

    -> 커뮤니케이션 다이어그램 : 객체들이 주고받는 메시지를 표현할 뿐 아니라 객체들 간의 연관까지 표현

    -> 상태 다이어그램 : 하나의 객체가 자신이 속한 클래스의 상태 변화 혹은 다른 객체와의 상호 작용에 따라 어떻게 변화하는지 표현

    -> 활동 다이어그램 : 객체의 처리 로직이나 조건에 따른 처리의 흐름을 순서에 따라 표현

-> 상호작용 개요 다이어그램 : 상호작용 다이어그램 간의 제어 흐름 표현

    -> 타이밍 다이어그램 : 객체 상태 변화와 시간 제약을 명시적으로 표현

## [1.2 화면 설계(1)](http://127.0.0.1:5503/143)

2020. 4. 6. 22:42

### 사용자 인터페이스(User Interface)

**사용자 인터페이스의 개요**

- 사용자와 시스템 간의 상호작용을 원활하게 도와주는 장치나 소프트웨어

- 사용자 인터페에스의 3가지 분야

    -> 정보 제공과 전달을 위한 물리적 제어에 관한 분야

    -> 콘텐츠의 상세적인 표현과 전체적인 구성에 관한 분야

    -> 모든 사용자가 편리하고 간편하게 사용하도록 하는 기능에 관한 분야

**사용자 인터페이스의 구분**

- CLI(Command Line Interface) : 명령과 출력이 텍스트 형태로 이루어지는 인터페이스

- GUI(Graphic User Interface) : 아이콘이나 메뉴를 마우스로 선택하여 작업을 수행하는 인터페이스

- NUI(Natural User Interface) : 말이나 행동으로 조작하는 인터페이스

**사용자 인터페이스의 기본 원칙**

- 직관성, 유효성, 학습성, 유연성

**사용자 인터페이스의 설계 지침**

- 사용자 중심, 일관성, 단순성, 결과 예측 가능, 가시성, 표준화, 접근성, 명확성, 오류 발생 해결

### UI 표준 및 지침

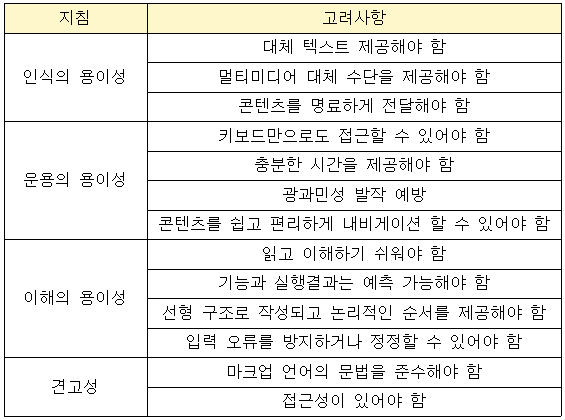
**UI 표준 및 지침의 개요**

- UI 표준과 지침을 토대로 기술의 중립성(표준), 보편적 표현 보장성(접근성), 기능의 호환성이 고려되었는지 확인

**한국형 웹 콘텐츠 접근성 지침(KWCAG)**

- 장애인과 비장애인이 동등하게 접근할 수 있는 웹 콘텐츠 제작의 방법 제시

- 웹 콘텐츠 접근성 지침 준수를 위한 고려사항



- 네비게이션 : 사용자가 사이트에서 원하는 정보를 빠르게 찾도록 도와주는 장치

**전자정보 웹 표준 준수 지침**

- 정부기관의 홈페이지 구축시 반영해야 할 최소한의 규약

- 전자정부 웹 표준 준수 지침 사항

    -> 내용의 문법 준수

    -> 내용과 표현의 분리

    -> 동작의 기술 중립성 보장

    -> 플러그인의 호환성

    -> 콘텐츠의 보편적 표현

    -> 운영체제에 독립적인 콘텐츠 제공

    -> 부가 기능의 호환성 확보

    -> 다양한 프로그램 제공

### UI 설계 도구

**UI 설계 도구**

- 사용자의 요구사항에 맞게 UI를 설계할 때 사용하는 도구

**와이어프레임**

- 기획 초기 단계에서 제작하는 것으로 페이지에 대한 대략적인 레이아웃이나 UI 요소 등에 대한 뼈대를 설계

- 와이어프레임 툴 : 손그림, 파워포인트, 키노트, 스케치, 일러스트, 포토샵 등

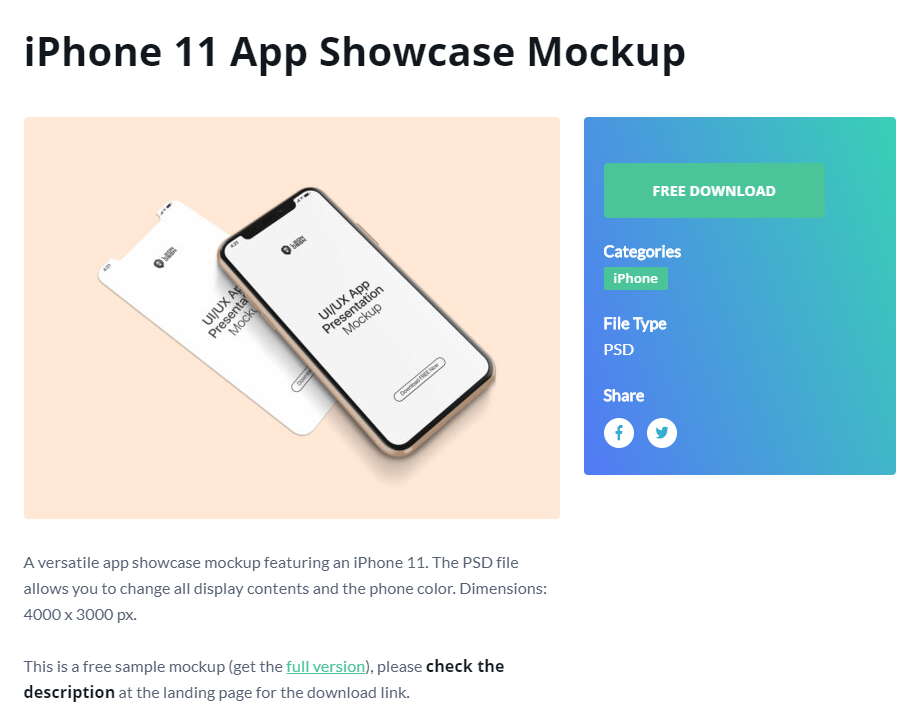


키노트

**목업**

- 와이어프레임보다 좀 더 실제 화면과 유사하게 만드는 정적인 형태의 모형

- 목업 툴 : 파워 목업, 발사믹 목업 등



출처 : https://www.mockupworld.co/free/category/iphone/

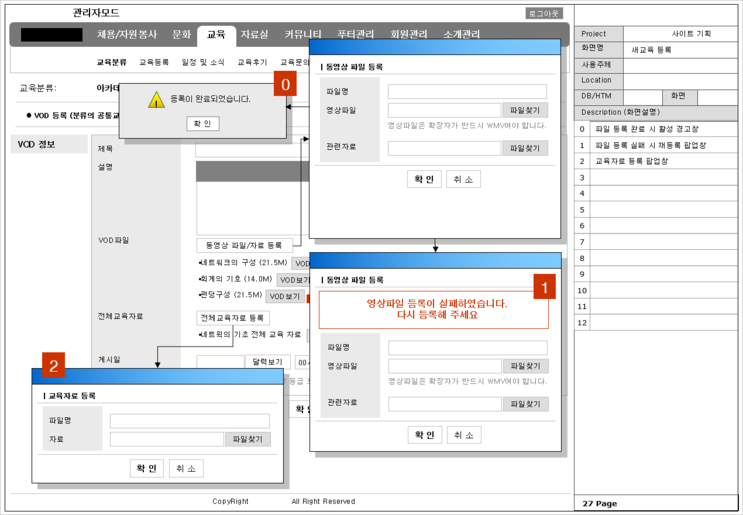
**스토리보드**

- 와이어프레임에 콘텐츠에 대한 설명이나 페이지 간 이동 흐름 등을 추가한 문서

- 디자이너와 개발자가 최종적으로 참고하는 작업 지침서

- 서비스 구축을 위한 모든 정보가 담겨 있어야 함

- 스토리보드 툴 : 파워포인트, 키노트, 스케치, Axure 등

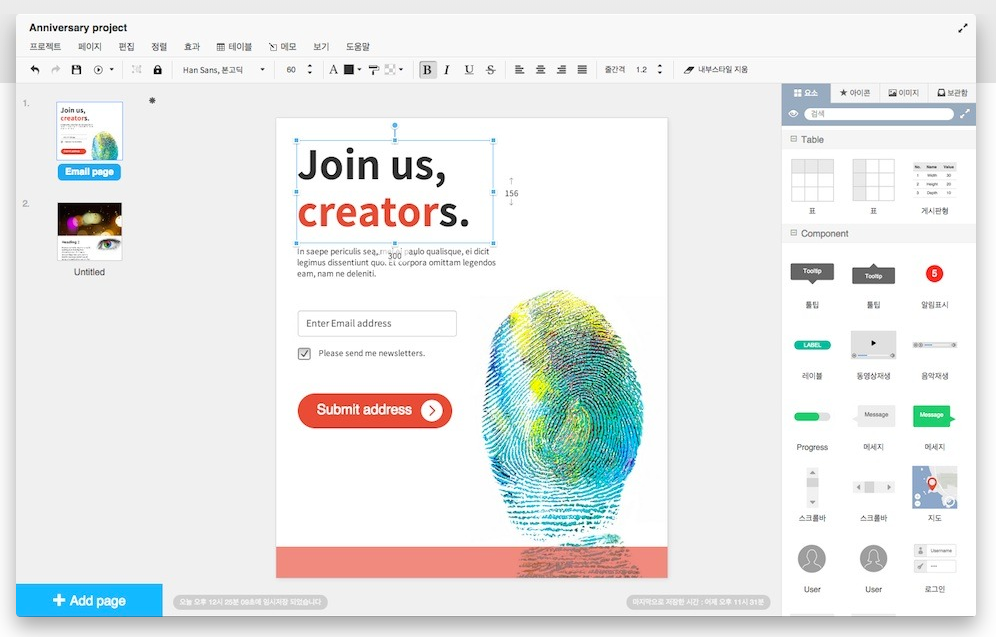
출처 : https://m.blog.naver.com/durandot/100205321229

**프로토타입**

- 와이어프레임이나 스토리보드 등에 인터랙션을 적용해 실제 구현된 것처럼 테스트가 가능한 동적인 형태의 모형

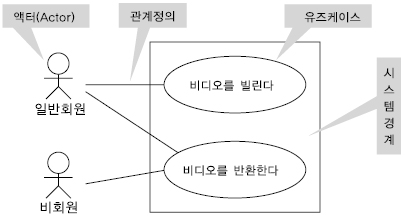
- 작성 방법에 따라 페이퍼/디지털 프로토타입으로 나눔

- 프로토타입 툴 : HTML/CSS, Axure, Flinto, 네이버 포로토나우, 카카오 오븐 등

출처 : https://ovenapp.io/

**유스케이스**

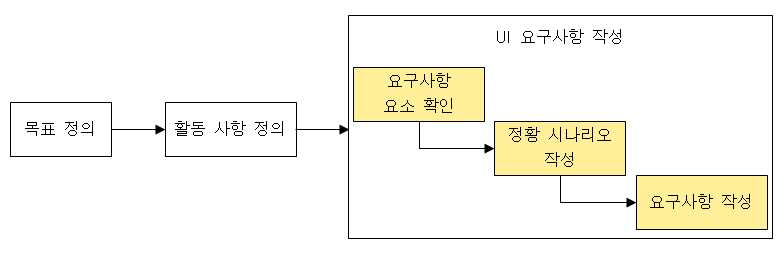
- 사용자 측면에서의 요구사항으로 사용자가 원하는 목표를 달성하기 위해 수행할 내용 기술

출처 : http://www.hanbit.co.kr/media/channel/view.html?cms\_code=CMS8900361225&cate\_cd=

### UI 요구사항 확인

**UI 요구사항 확인의 개요**

- 새로 개발할 시스템에 적용할 UI 관련 요구사항을 조사해서 작성하는 단계



**목표 정의**

- 사용자들을 대상으로 인터뷰를 하고 사용자들의 의견이 수렴된 비즈니스 요구사항을 정의

- 인터뷰 진행 시 유의 사항

    -> 사업적, 기술적 요구사항을 명확히 이해

    -> 가능한 개별적인 진행

    -> 한 시간을 넘기지 않는게 좋음

    -> 사용자 리서치 시작 전 해야 함

**활동 사항 정의**

- 조사한 요구사항을 토대로 앞으로 해야 할 활동 사항을 정의

- 기술의 발전 가능성을 파악하고 UI 디자인의 방향 제시

**UI 요구사항 작성**

- 여러 경로로 수집된 사용자의 요구사항을 검토하고 분석하여 UI 개발 목적에 맞게 작성해야 함

- 실 사용자 중심으로 작성

- 여러 사람의 인터뷰를 통해 다양한 의견을 수렴하여 작성

**요구사항 요소 확인**

- 파악된 요구사항 요소의 종류와 각각의 표현 방식 등을 검토

- 요구사항 요소 : 데이터 요구, 기능 요구, 제품/서비스의 품질, 제약 사항

**정황 시나리오 작성**

- 사용자의 요구사항을 도출하기 위해 작성

- 사용자가 목표를 달성하기 위해 수행하는 방법을 순차적으로 묘사

- 개발하는 서비스의 모습을 상상하는 첫번째 단계로 사용자 관점에서 시나리오를 작성해야 함

**요구사항 작성**

- 정황 시나리오를 토대로 작성

## [1.2 화면 설계(2)](http://127.0.0.1:5503/144)

2020. 4. 7. 17:21

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

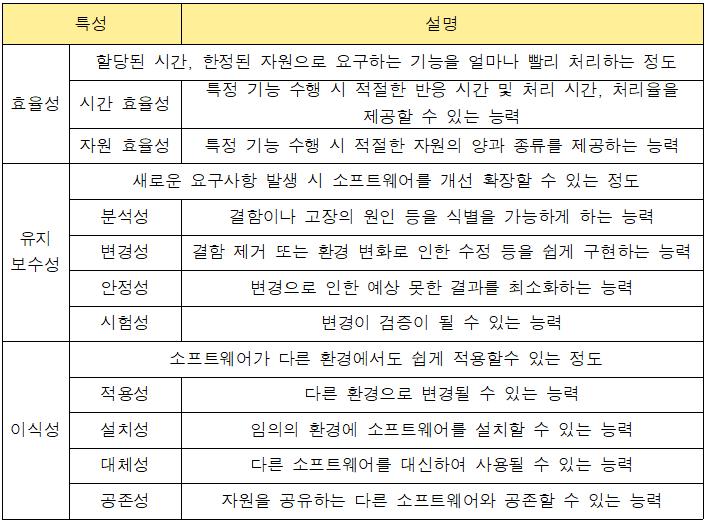
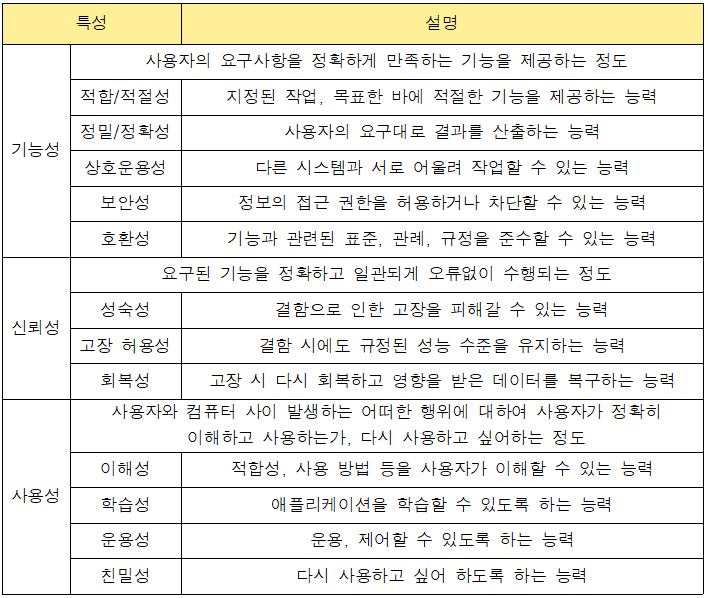
728x90

### 품질 요구사항

**품질 요구사항**

- 소프트웨어의 기능, 성능, 만족도 등 소프트웨어에 대한 요구사항이 얼마나 충족하는 가를 나타내는 것

- ISO/IEC 9126 : 국제 표준으로 소프트웨어 품질 특성과 평가를 위한 표준 지침



### UI 프로토타입 제작 및 검토

**UI 프로토타입의 개요**

- 사용자 요구사항을 기반으로 실제 동작하는 것처럼 만든 동적인 형태의 모형

- 테스트 가능

- 최대한 간단하게 만들어야 함

- 일부 핵심적인 기능을 제공하지만 최종 제품의 작동 방식을 이해시켜줄 기능은 반드시 포함되어야 함

- 실제 사용자를 대상으로 테스트해야 함

**UI 프로토타입의 장단점**

- 장점

    -> 사용자를 설득, 이해시키기 쉬움

    -> 요구사항을 점검하며 혼선은 예방하므로써 개발 시간을 줄일 수 있음

    -> 사전 오류 검출 가능

- 단점

    -> 프로토타입 제작으로 인해 작업 시간을 증가시킬 수 있음

    -> 필요 이상의 자원 소모 가능

    -> 부분적으로 작업 시 중요한 작업이 생략될 수 있음

**프로토타이핑의 종류**

- 페이퍼 프로토타입

    -> 아날로그 방법(스케치, 글, 그림) 등을 이용하여 직접 작성

    -> 제작 기간이 짧고, 제작 비용이 적을 경우, 업무 회의가 빠를 경우, 급하게 만들어야 하는 경우 사용

- 디지털 프로토타입

    -> 프로그램을 사용하여 작성

    -> 재사용이 필요하거나, 완성 제품과 비슷하게 만들어야 하거나, 숙련된 전문가가 있을 때 사용

**UI 프로토타입 계획 및 작성 시 고려사항**

- 계획 시 고려사항

    -> 일정은 아키텍처가 확정 ~ 프로젝트 실제 분석 작업 완료 사이에 진행해야 함

    -> 프로토타입을 통해서 발생하는 이슈를 모두 취합하여 해결 방법을 제시

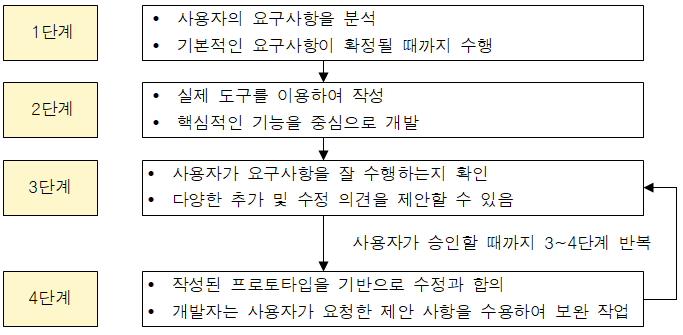
    -> 진행하면서 가장 많은 시간이 소요된 구간을 찾아 그 원인을 분석하여 해결 방법을 제시

- 작성 시 고려사항

    -> 프로젝트의 상황을 감안해서 프로토타입의 범위를 정해야 함

    -> 완성된 프로토타입이 실제 개발에 참조될 수 있는 지 확인

**UI 프로토타입 제작 단계**

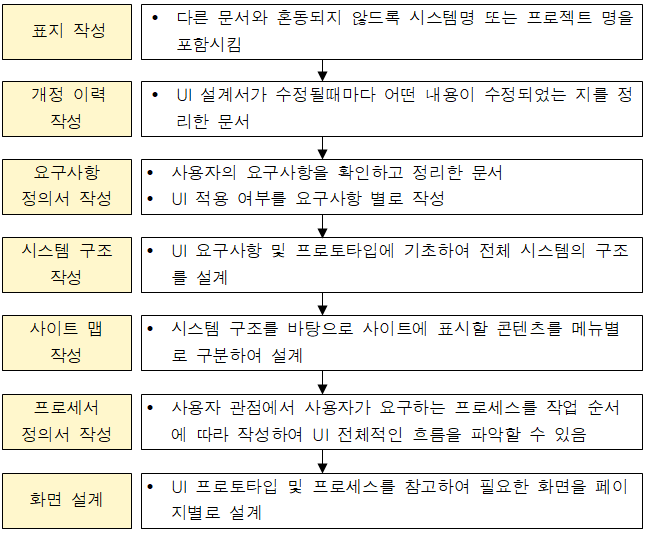


### UI 설계서 작성

**UI 설계서의 개요**

- 사용자의 요구사항을 바탕으로 UI 설계를 구체화하여 작성하는 문서

**UI 설계서 작성 순서**



### UI 상세 설계

**UI 시나리오 문서의 개요**

- UI 설계서를 바탕으로 실제 설계 및 구현을 위해 모든 화면에 대한 자세한 설계를 진행

- 시나리오를 작성해야 함

- 사용자 인터페이스의 기능 구조, 대표 화면, 화면 간 상호작용의 흐름, 다양한 상황에서의 예외 처리 등을 문서로 정리

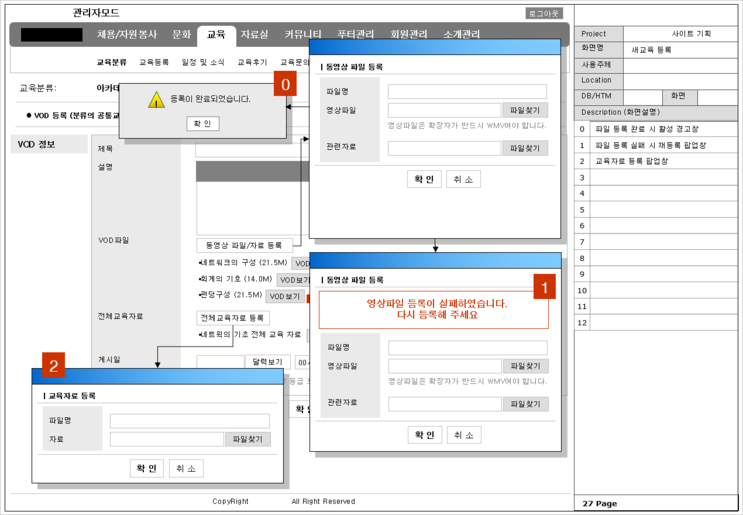
**UI 시나리오 문서 작성 원칙**

- 개발자가 전체 UI 기능과 작동 방식을 이해할 수 있도록 구체적으로 작성

- UI 요소와 인터랙션을 일반 규칙으로 정의

- 인터랙션의 흐름을 정의하고 인터랙션의 순서, 분기, 조건, 반복 등을 명시

- 예외 상황에 대비한 다양한 케이스를 정의

출처 : https://m.blog.naver.com/durandot/100205321229

**UI 시나리오 문서 작성을 위한 일반 규칙**

- 주요 키의 위치와 기능

    -> 모든 화면에 공통적으로 배치되는 주요 키의 위치와 기능을 설명

    -> 여러 화면 간 일관성 보장

- 공통 UI 요소

    -> UI 요소를 언제 어떤 형태로 사용할 지 정의

    -> 사용자의 조작에 대한 반응하는지에 대한 흐름을 설명

- 기본 스크린 레이아웃

    -> 모든 화면에 공통적으로 나타나는 요소들에 대한 위치와 속성을 정의

- 기본 인터랙션 규칙

    -> 터치 제스처 등에 공통적으로 사용되는 조작 방법과 화면 전환 효과 등을 기술

- 공통 단위 태스크 흐름

    -> 많은 기능들에 공통적으로 사용되는 삭제, 검색, 매너 모드 상태 등에 대한 인터랙션 흐름 설명

- 케이스 문서

    -> 다양한 상황에서 공통적으로 적용되는 시스템의 동작을 정의한 문서

**UI 시나리오 문서의 요건**

- 완전성 : 누락되지 않도록 상세히 기술

- 일관성 : 서비스의 목표, 요구사항, UI 스타일이 모두 일관성을 유지해야 함

- 이해성 : 누구나 쉽게 이해할 수 있도록 설명

- 가독성 : 표준화된 템플릿을 활용하여 읽기 쉽도록 해야 함

- 수정 용이성 : 시나리오의 수정, 개선이 쉬워야 함

- 추적 용이성 : 변경 사항이 언제 어떻게 왜 발생했는지 쉽게 추적할 수 있어야 함

### HCI / UX / 감성공학

**HCI(Human Computer Interaction or Interface)**

- 사람이 시스템을 보다 편리하고 안전하게 사용할 수 있도록 개발, 연구하는 학문

- 최종 목표는 시스템을 사용하는데 있어 최적의 사용자 경험을 만드는 것

**UX(User Experience)**

- 사용자가 시스템이나 서비스를 이용하면서 느끼고 생각하게 되는 총제적인 경험

**감성 공학**

- 제품이나 작업환경을 사용자의 감성에 맞도록 설계 제작하는 기술

## [1.3 애플리케이션 설계(1)](http://127.0.0.1:5503/145)

2020. 4. 8. 17:36

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 소프트웨어 아키텍처

**소프트웨어 아키텍처의 설계**

- 소프트웨어의 골격이 되는 기본 구조

- 소프트웨어를 구성하는 요소들 간의 관계를 표현하는 시스템의 구조 또는 구조체

- 좋은 품질을 유지하면서 사용자의 비기능적 요구사항으로 나타난 제약을 반영하고, 기능적 요구사항을 구현하는 방법을 찾는 해결 과정

- 애플리케이션의 분할 방법과 분할된 모듈에 할당될 기능, 모듈 간의 인터페이스 등을 결정

**모듈화**

- 소프트웨어의 성능을 향상하거나 시스템의 수정 및 재사용, 유지 관리 등이 용이하도록 시스템의 시스템의 기능들을 모듈 단위로 나누는 것

- 모듈의 크기와 개수는 반비례관계 개수와 통합 비용은 비례 관계

**추상화**

- 문제의 전체를 설계 후 세분화하여 구체화하는 과정

- 완전한 시스템을 구축하기 전에 그 시스템과 유사한 모델을 만들어 여러 가지 요인들을 테스트할 수 있음

- 최소 비용으로 실제 상황에 대처할 수 있고 시스템의 구조 및 구성을 대략적으로 파악할 수 있음

- 추상화 유형

    -> 과정 추상화 : 전반적인 흐름만 파악

    -> 데이터 추상화 : 데이터의 세부사항은 정의하지 않고 구조를 대표할 수 있는 표현으로 대체

    -> 제어 추상화 : 이벤트의 발생의 세부사항은 정의하지 않고 구조를 대표할 수 있는 표현으로 대체

**단계적 분해**

- 문제를 상위 중요 개념으로부터 하위의 개념으로 구체화하는 분할 기법

- 추상화의 반복으로 세분화

**정보 은닉**

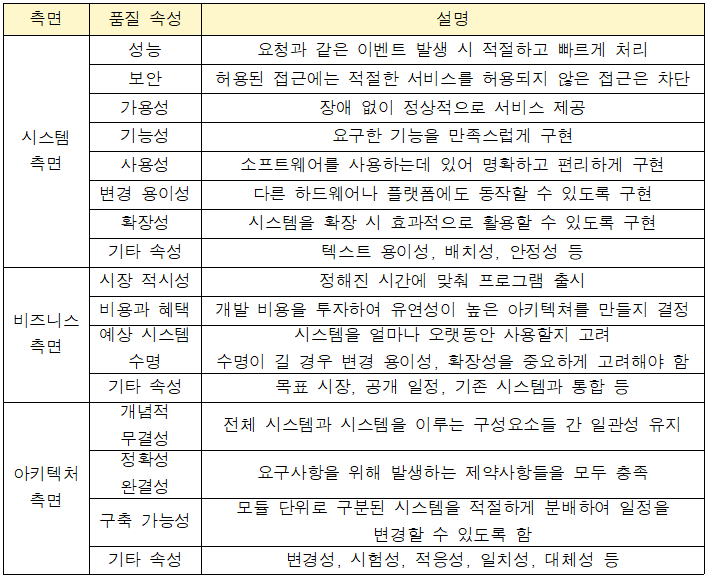
- 한 모듈 내부에 포함된 정보들을 감추어 다른 모듈이 접근하거나 변경하지 못하도록 하는 기법

- 다른 모듈과 커뮤니케이션을 할 때는 필요한 정보만 인터페이스를 통해 주고받음

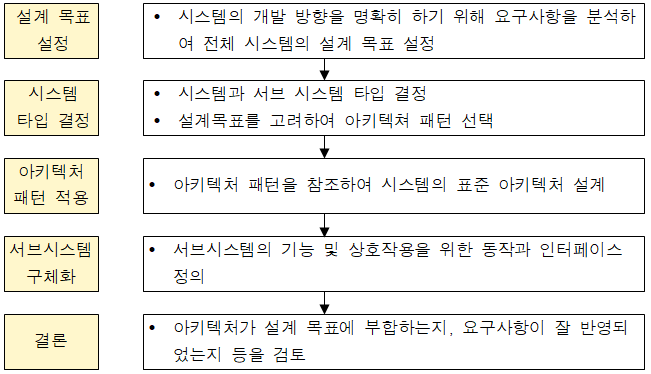
- 모듈을 독립적으로 수행하기 때문에 다른 모듈에 영향을 주지 않아 수정, 시험, 유지보수가 용이

**소프트웨어 아키텍처의 품질 속성**

- 소프트웨어 아키텍처가 이해 관계자들이 요구하는 수준의 품질을 유지하고 보장할 수 있게 설계되었는지를 확인하기 위해 품질 요소들을 구체화시켜 놓은 것



**소프트웨어 아키텍처의 설계 과정**



### 아키텍처 패턴

**아키텍처 패턴의 개요**

- 아키텍처를 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제

- 시스템의 구조를 구성하기 위한 기본적인 틀을 제공

- 서브시스템과 그 역할이 정의되어 있어 서브시스템 사이 관계와 규칙, 지침 등이 포함되어 있음

- 아키텍처 패턴의 장점

    -> 개발 시간을 단축시킴

    -> 고품질의 소프트웨어 생산 가능

    -> 검증된 구조로 작업을 하여 안정적인 개발 가능

    -> 시스템 구조를 이해하기 쉬워 개발에 참여하지 않아도 유지보수가 쉬움

**레이어 패턴**

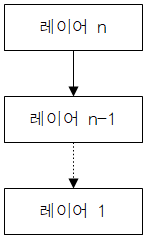
- 시스템을 계층으로 구분하여 구성

- 각각의 서브 시스템이 계층 구조를 이룸

- 상위 계층은 하위 계층에 대한 서비스 제공자가 되고 하위 계층은 상위 계층의 클라이언트가 됨

- 마주 보는 두 계층 사이에만 상호 작용이 이루어짐

- 특정 계층만을 교체해 시스템을 개선하는 것이 가능



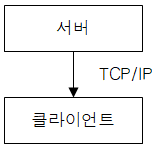
**클라이언트-서버 패턴**

- 하나의 서버와 다수의 클라이언트로 구성

- 사용자는 클라이언트를 통해 서버에 요청하고 응답을 받아 사용자에게 제공

- 서버는 클라이언트의 요청에 대비해 항상 대기 상태 유지

- 요청을 위하여 동기화되는 경우를 제외하고는 서로 독립적임



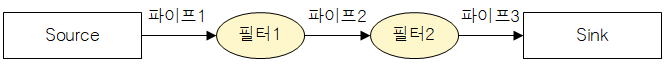
**파이프-필터 패턴**

- 데이터 스트림(데이터가 송수신되거나 처리되는 흐름) 절차의 각 단계를 필터 컴포넌트로 캡슐화하여 파이프를 통해 데이터를 전송

- 필터 컴포넌트는 재사용성이 좋고 추가가 쉬워 확장이 용이

- 필터 컴포넌트를 재배치하여 다양한 파이프라인 구축 가능

- 데이터 변환, 버퍼링, 동기화 등에 사용



**모델-뷰-컨트롤러 패턴**

- 서브시스템을 3개의 부분으로 구조화

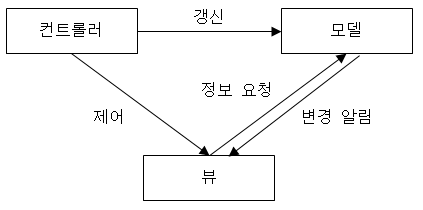
- 모델 : 서브시스템의 핵심 기능과 데이터 보관

- 뷰 : 사용자에게 정보 표시

- 컨트롤러 : 사용자로부터 받은 입력 처리

- 각 부분은 별도로 분리되어 있어 서로 영향을 받지 않고 독립적인 개발 작업 수행

- 여러 개의 뷰를 만들 수 있어 한 개의 모델에 여러 개의 뷰를 필요로 하는 대화형 애플리케이션에 적합



**기타 패턴**

- 마스터-슬레이브 패턴 : 마스터 컴포넌트에서 슬레이브 컴포넌트로 작업을 분할한 후 슬레이브 컴포넌트에서 처리된 결과물을 다시 돌려받는 방식

- 보로커 패턴 : 사용자가 원하는 서비스와 특성을 브로커 컴포넌트에 요청하면 브로커 컴포넌트가 요청에 맞는 컴포넌트를 연결

- 피어-투-피어 패턴 : 피어를 하나의 컴포넌트로 간주하여 각 피어는 클라이언트 또는 서버가 될 수도 있음

- 이벤트-버스 패턴 : 소스가 특정 채널에 이벤트 메시지를 발행하면 해당 채널을 구독한 리스너들이 메시지를 받아 이벤트를 처리하는 방식

- 블랙보드 패턴 : 모든 컴포넌트들이 공유 데이터 저장소와 블랙보드 컴포넌트에 접근이 가능하여 검색을 통해 블랙보드에서 원하는 데이터를 찾을 수 있음

- 인터프리터 패턴 : 프로그램 코드의 각 라인을 수행하는 방법을 지정하고 기호마다 클래스를 갖도록 구성

### 객체지향(Object-Oriented)

**객체지향의 개요**

- 현실 세계의 개체를 기계의 부품처럼 하나의 객체(Object)로 만들어 소프트웨어를 개발할 때 객체를 조립하여 작성할 수 있는 기법

- 구조적 기법의 문제점을 해결하기 위해 사용

- 소프트웨어의 재사용 및 확장이 용이하여 고품질의 소프트웨어를 빠르게 개발할 수 있어 유지보수가 쉬움

- 복잡한 구조를 단계적이고 계층적이게 표현

- 멀티미디어 데이터 및 병렬 처리 지원

- 구성 요소 : 객체, 클래스, 캡슐화, 상속, 다형성

**객체**

- 데이터와 데이터를 처리하는 함수를 묶어 캡슐화한 하나의 소프트웨어 모듈

- 데이터 : 객체가 가지고 있는 정보

- 함수 : 객체가 수행하는 기능으로 데이터를 처리하는 알고리즘

- 객체의 특성

    -> 객체는 독립적으로 식별 가능한 이름을 가지고 있음

    -> 객체의 상태는 시간에 따라 변함

    -> 객체 간의 상호 연관성에 의해 관계 형성

    -> 객체가 반응할 수 있는 메시지의 집합을 행위라고 하며 객체는 행위의 특징을 나타낼 수 있음

    -> 객체는 일정한 기억 장소를 가지고 있음

**클래스**

- 공통된 속성과 연산을 갖는 객체의 집합으로 객체의 일반적인 타입을 의미

- 각각의 객체들이 갖는 속성과 연산을 정의하고 있는 틀

- 인스턴스 : 클래스에 속한 각각의 객체

- 인스턴스화 : 클래스로부터 새로운 객체를 생성하는 것

- 슈퍼 클래스 : 특정 클래스의 부모(상위) 클래스

- 서브 클래스 : 특정 클래스의 자식(하위) 클래스

**캡슐화**

- 데이터와 함수를 하나로 묶은 것

- 인터페이스를 제외한 세부 내용이 은폐되어 외부에서 접근이 제한적이기 때문에 외부에서 변경하기 어려움

**상속**

- 이미 정의된 상위 클래스의 모든 속성과 연산을 하위 클래스가 물려받는 것

- 하위 클래스는 부모 클래스의 모든 속성과 연산을 다시 정의하지 않고 사용할 수 있음

- 하위 클래스는 상속받은 속성 외에 새로운 속성과 연산을 첨가하여 사용할 수 있음

- 객체와 클래스의 재사용을 높이는 중요한 개념

- 다중 상속 : 한 개의 클래스가 두 개 이상의 상위 클래스로부터 상속받는 것

**다형성**

- 메시지에 의해 객체가 연산을 수행하게 될 때 하나의 메시지에 대해 각각의 클래스가 가지고 있는 고유한 특성으로 응답할 수 있는 능력

- 객체지향의 오버로딩의 개념

- [C++ 오버로딩과 오버라이딩](https://1d1cblog.tistory.com/84)

[C++ 오버로딩과 오버라이딩](https://1d1cblog.tistory.com/84?category=834361" \t "_blank)

[오버 로딩(overloading)은 함수 다중 정의를 말합니다. 함수 다중 정의는 함수 하나가 여러 의미를 갖는 것을 말합니다. 이름은 같아도 함수를 구성하는 매개변수의 타입이나 개수가 달라지게 되면 다른 함수로 취..](https://1d1cblog.tistory.com/84?category=834361" \t "_blank)

## [1.3 애플리케이션 설계(2)](http://127.0.0.1:5503/146)

2020. 4. 8. 23:33

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 모듈

**모듈의 개요**

- 모듈화를 통해 분리된 시스템의 각 기능들

- 서브루틴, 서브시스템, 소프트웨어 내의 프로그램, 작업 단위 등과 같은 의미로 사용

- 단독으로 컴파일 가능하며 재사용 할 수 있음

- 각 모듈의 기능이 서로 독립적이고 모듈이 하나의 기능만을 수행하고 다른 모듈과의 과도한 상호작용을 배제함

- 독립성이 높을수록 모듈을 수정해도 다른 모듈에 영향이 없어 오류가 발생해도 쉽게 해결 가능

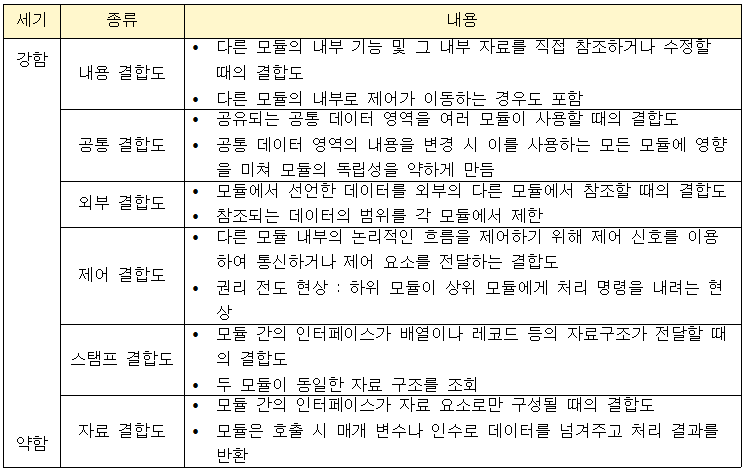
- 모듈의 독립성을 높이기 위해서는 결합도는 약하게, 응집도는 강하게 해야함

**결합도**

- 모듈 간에 상호 의존도 또는 모듈 사이의 연관 관계

- 결합도와 품질은 반비례 관계

- 결합도가 강하면 시스템 구현 및 유지보수 작업이 어려움

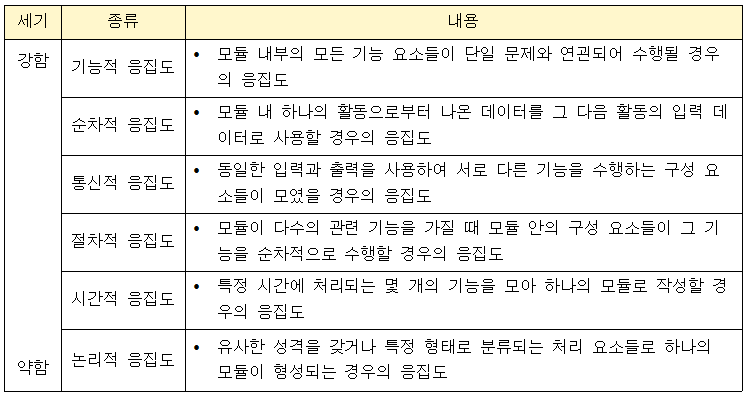


**응집도**

- 정보 은닉 개념을 확장한 것으로 모듈의 내부 요소들의 서로 관련되어 있는 정도

- 모듈이 독립적인 기능으로 정의되어 있는 정도

- 응집도와 품질은 비례 관계

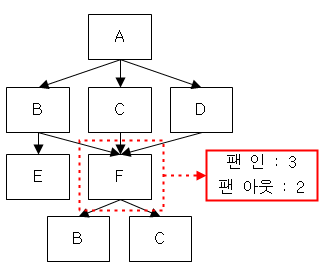


**+ 응집도 마지막 우연적 응집도 추가**

**팬 인/아웃**

- 팬 인 : 호출하는 모듈의 수

- 팬 아웃 : 호출되는 모듈의 수



### 공통 모듈

**공통 모듈의 개요**

- 여러 프로그램에서 공통적으로 사용할 수 있는 모듈

- 자주 사용되는 계산식이나 매번 필요한 사용자 인증 같은 기능들이 공통 모듈로 구성될 수 있음

- 공통 모듈의 명세 기법 : 정확성, 명확성, 완전성, 일관성, 추적성

**재사용**

- 비용과 시간을 절약하기 위해 이미 개발된 기능들을 파악하고 재구성하여 새로운 시스템 개발에 사용하기 적합하도록 최적화 시키는 작업

- 재사용되는 대상은 외부 모듈과의 결합도는 낮고 응집도는 높아야 함

**효과적인 모듈 설계 방안**

- 결합도는 줄이고 응집도는 높여 모듈의 독립성과 재사용성을 높임

- 하나의 입구와 하나의 출구를 가져야 함

### 코드

**코드의 개요**

- 컴퓨터를 이용하여 자료를 처리하는 과정에서 분류, 조합 및 집계를 용이하게 하고 특정 자료의 추출을 쉽게 하기 위해 사용하는 기호

- 코드의 기능 : 식별 기능, 분류 기능, 배열 기능

**코드의 종류**

- 순차 코드 : 일정 기준에 따라 최초의 자료부터 일련번호를 부여하는 방법

- 블록 코드 : 대상 항목에서 공통적인 것을을 블록으로 구분하고 블록 내에 일련번호를 부여하는 방법

- 10진 코드 : 대상 항목을 0~9까지 10진 분할하고 다시 각각에 대하여 10진 분할을 필요한 만큼 반복하는 방법

- 그룹 분류 코드 : 일정 기준에 따라 대분류, 중분류, 소분류 등으로 구분하고 그룹 안에서 일련번호를 부여하는 방법

- 연상 코드 : 항목의 명칭이나 약호와 관계있는 숫자, 문자, 기호를 이용하여 코드를 부여하는 방법

- 표의 숫자 코드 : 항목의 성질(길이, 넓이, 부피 등)의 물리적인 수치를 그대로 코드에 적용시키는 방법

- 합성 코드 : 하나의 코드로 수행하기 어려운 경우 2개 이상의 코드를 조합하여 적용시키는 방법

**코드 부여 체계**

- 코드(이름)만으로도 개체의 용도와 적용 범위를 알 수 있도록 코드를 부여하는 방식

- 시스템의 고유한 코드와 개체를 나타내는 코드 등이 정의되어야 함

### 디자인 패턴

**디자인 패턴의 개요**

- 각 모듈의 세분화된 역할이나 모듈들 간의 인터페이스와 같은 코드를 작성하는 수준의 세부적인 구현 방안을 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제

- 개발 과정에 문제가 발생 시 새로 해결책을 구상하기보다 문제에 해당하는 디자인 패턴을 참고하여 적용하는 것이 더욱 효율적임

- 재사용할 수 있는 기본형 코드들이 포함되어 있음

**생성 패턴**

- 객체의 생성과 참조 과정을 샘플화 하여 객체가 생성되거나 변경되어도 프로그램의 구조에 영향을 크게 받지 않도록 하여 프로그램의 유연성을 더해줌

- 추상 팩토리 : 구체적인 클래스에 의존하지 않고 인터페이스를 통해 서로 연관, 의존하는 객체들의 그룹으로 생성하여 추상적으로 표현

- 빌더 : 작게 분리된 인스턴스를 건축 하듯이 조합하여 객체 생성

- 팩토리 메소드 : 객체 생성을 서브 클래스에서 처리하도록 분리하여 캡슐화한 패턴

- 프로토타입 : 원본 객체를 복제하는 방법으로 객체를 생성하는 패턴

- 싱글톤 : 하나의 객체를 생성하면 생성된 객체를 어디서든 참조할 수 있지만 여러 프로세스가 동시에 참조할 수는 없음

**구조 패턴**

- 클래스나 객체들을 조합하여 더 큰 구조로 만들 수 있게 해주는 패턴

- 어댑터 : 호환성이 없는 클래스들의 인터페이스를 다른 클래스가 이용할 수 있도록 변환해주는 패턴

- 브리지 : 구현부에서 추상층을 분리하여 서로가 독립적으로 확장할 수 있도록 구성한 패턴

- 컴포지트 : 여러 객체를 가진 복합 객체와 단일 객체를 구분 없이 다루고자 할 때 사용하는 패턴

- 데코레이터 : 객체 간의 결합을 통해 능동적으로 기능들을 확장할 수 있는 패턴

- 퍼싸드 : 복잡한 서브 클래스들을 피해 더 상위에 인터페이스를 구상함으로써 서브 클래스의 기능을 간편하게 사용할 수 있도록 하는 패턴

- 플라이웨이트 : 인스턴스가 필요할 때마다 생성하는 것이 아닌 공유해서 사용함으로써 메모리를 절약하는 패턴

- 프록시 : 접근이 어려운 객체와 여기에 연결하려는 객체 사이에서 인터페이스 역할을 수행하는 패턴

**행위 패턴**

- 클래스나 객체들이 서로 상호작용하는 방법이나 책임 분배 방법을 정의한 패턴

- 책임 연쇄 : 요청을 처리할 수 있는 객체가 둘 이상 존재하여 한 객체가 처리하지 못하면 다음 객체로 넘어가는 형태의 패턴

- 커맨드 : 요청을 객체의 형태로 캡슐화하여 재이용하거나 취소할 수 있도록 요청에 필요한 정보를 저장하거나 로그에 남기는 패턴

- 인터프리터 : 언어에 문법 표현을 정의하는 패턴

- 반복자 : 자료 구조와 같이 접근이 잦은 객체에 대해 동일한 인터페이스를 사용하도록 하는 패턴

- 중재자 : 수많은 객체들 간의 복잡한 상호작용을 캡슐화하여 객체로 정의하는 패턴

- 메멘토 : 특정 시점에서의 객체 내부 상태를 객체화함으로써 이후 요청에 따라 객체를 해당 시점의 상태로 돌릴 수 있는 기능을 제공하는 패턴

- 옵서버 : 한 객체의 상태가 변화하면 객체에 상속되어 있는 다른 객체들에게 변화된 상태를 전달하는 패턴

- 상태 : 객체의 상태에 따라 동일한 동작을 다르게 처리해야 할 때 사용하는 패턴

- 전략 : 동일한 계열의 알고리즘들을 개별적으로 캡슐화하여 상호 교환할 수 있게 정의하는 패턴

- 템플릿 메소드 : 상위 클래스에서 골격을 정의하고 하위 클래스에서 처리를 구체화하는 구조의 패턴

- 방문자 : 각 클래스들의 데이터 구조에서 처리 기능을 분리하여 별도의 클래스로 구성하는 패턴

## [1.4 인터페이스 설계(1)](http://127.0.0.1:5503/152)

2020. 4. 11. 16:26

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 시스템 인터페이스 요구사항 분석

**시스템 인터페이스 요구사항 구성**

- 시스템 인터페이스는 독립적으로 떨어져 있는 시스템들끼리 서로 연동하여 상호작용하기 위한 접속 방법이나 규칙

- 시스템 인터페이스 요구사항은 개발을 목표로 하는 시스템과 외부 시스템을 연동하는데 필요한 필요한 시스템 인터페이스에 대한 요구사항을 기술한 것

- 시스템 요구사항 명세서 포함 요소 : 인터페이스 이름, 연계 대상 시스템, 연계 범위 및 내용, 연계 방식, 송신 데이터, 인터페이스 주기, 기타 고려사항 등

**시스템 인터페이스 요구사항 분석**

- 요구사항 명세서에서 요구사항을 기능적 요구사항과 비기능적 요구사항으로 분류하고 조직화하여 요구사항 명세를 구체화하고 이를 이해관계자에게 전달하는 일련의 과정

- 기능적 요구사항 : 시스템이 무엇을 하고 어떤 기능을 하는 가

- 비기능적 요구사항 : 시스템이나 프로젝트 개발 과정 등에서 지켜야 할 제약 사항

- 요구사항의 분해가 필요한 경우 세분화 할 수 있음

**시스템 인터페이스 요구사항 분석 절차**

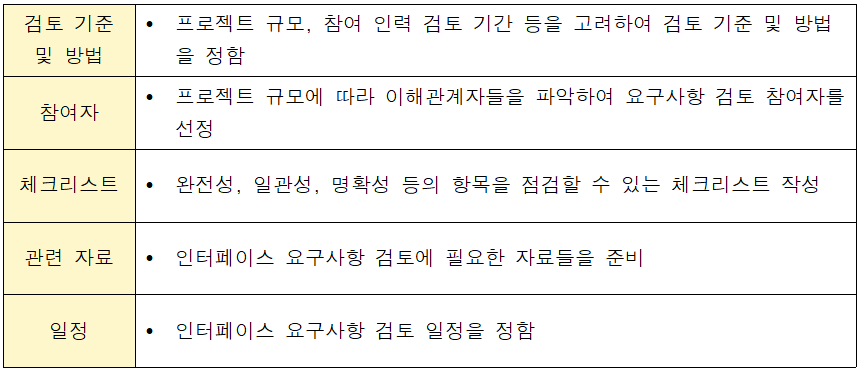


### 인터페이스 요구사항 검증

**요구사항 검증**

- 인터페이스의 설계 및 구현 전 사용자들의 요구사항이 요구사항 명세서에 정확하고 완전하게 기술되었는지 검토하고 개발 범위의 기준인 베이스라인을 설정하는 것

**인터페이스 요구사항 검토 계획 수립**



**인터페이스 요구사항 검토 및 오류 수정**

- 체크리스트의 항목에 따라 인터페이스 요구사항 명세서 검토

- 요구사항 검토 시 오류가 발견되면 이를 수정할 수 있도록 오류 목록과 시정 조치서 작성

- 시정 조치서를 작성할 경우 조치가 완료되었는지를 확인하여 조치가 완료되면 인터페이스 요구사항 검토 작업을 완료

**인터페이스 요구사항 베이스라인 설정**

- 검증된 인터페이스 요구사항은 주요 의사 결정자에게 공식적으로 승인을 받음

- 소프트웨어 설계 및 구현을 위해 요구사항 명세서의 베이스라인 설정

**요구사항 검증 방법**

- 요구사항 검토 : 요구사항 명세서의 결함 여부를 검토 담당자들이 수작업으로 분석하는 방법

    -> 동료검토 : 명세서 작성자가 직접 설명하는걸 동료들이 들으면서 결함을 발견하는 방법

    -> 워크스루 : 검토 회의 전 미리 명세서를 배포하여 사전 검토 후 짧은 회의를 통해 결함을 발견하는 방법

    -> 인스펙션 : 명세서 작성자를 제외한 다른 검토 전문가들이 명세서를 확인하면서 결함을 발견하는 방법

- 프로토타이핑 : 요구사항을 파악하기 위해 실제 개발될 소프트웨어에 대한 견본품을 만들어서 최종 결과물을 예측

- 테스트 설계 : 테스트 케이스를 생성하여 이후에 요구사항이 현실적으로 테스트 가능한지 검토

- CASE(Computer Aided Software Enginerring) 도구 활용 : 일관성 분석을 통해 요구사항 변경사항의 추적, 분석, 관리하고 표준 준수 여부를 확인

**인터페이스 요구사항 검증의 주요 항목**

- 완전성 : 모든 요구사항이 누락되지 않고 반영되었는가

- 일관성 : 요구사항이 모순되거나 충돌되는 점 없이 일관성을 유지하는가

- 명확성 : 모든 참여자가 요구사항을 명확하게 이해할 수 있는가

- 기능성 : 요구사항이 어떻게보다 무엇을에 중점을 두고 있는가

- 검증 가능성 : 요구사항이 사용자의 요구를 모두 만족하고 개발된 소프트웨어가 사용자의 요구 내용과 일치하는지를 검증할 수 있는가

- 추적 가능성 : 요구사항 명세서와 설계서를 추적할 수 있는가

- 변경 용이성 : 요구사항 명세서의 변경이 쉽도록 작성되었는가

### 인터페이스 시스템 식별

**개발 시스템 식별**

- 인터페이스 관련 자료들을 기반으로 개발하고자 하는 시스템의 상세 식별 정보를 정의하고 목록을 작성

- 시스템 아키텍쳐 : 시스템 내부에서 하위 시스템이 어떻게 상호작용하는지 파악할 수 있도록 구성이나 동작원리를 나타냄

- 유스케이스 : 사용자의 요구사항을 기능 단위로 표현

**내˙외부 시스템 식별**

- 인터페이스 관련 자로들을 기반으로 개발할 시스템과 연계할 시스템들의 상세 식별 정보를 정의하고 목록을 작성

**내˙외부 시스템 환경 및 관리 주체 식별**

- 연계할 시스템 접속에 필요한 IP, URL, Port 정보 등 시스템의 실제 운용 환경 및 하드웨어를 실제적으로 관리하는 담당자를 확인

**내˙외부 시스템 네트워크 연결 정보 식별**

- 내˙외부 시스템을 연계하는데 필요한 네트워크 연결 정보 확인

**인터페이스 식별**

- 인터페이스 요구사항 명세서와 인터페이스 요구사항 목록을 기반으로 개발할 시스템과 연계할 시스템 사이의 인터페이스를 식별하고 목록을 작성

**인터페이스 시스템 식별**

- 인터페이스별로 인터페이스에 참여하는 시스템들을 송신 시스템과 수신 시스템으로 구분하여 작성

## [1.4 인터페이스 설계(2)](http://127.0.0.1:5503/153)

2020. 4. 11. 16:53

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 송수신 데이터 식별

**식별 대상 데이터**

- 송수신 사이에 교환되는 데이터

- 규격화된 표준 형식에 따라 전송

- 인터페이스 표준 항목

    -> 송수신 시스템을 연계하는데 표준적으로 필요한 데이터

    -> 시스템 공통부 : 시스템 연동 시 필요한 공통 정보

    -> 거래 공통부 : 시스템이 연동된 후 송수신되는 데이터를 처리할 때 필요한 정보

- 송수신 데이터 항목 : 송수신 시스템이 업무를 수행하는 데 사용하는 데이터

- 공통 코드 : 시스템들에서 공통적으로 사용하는 코드

**정보 흐름 식별**

- 개발할 시스템과 내외부 시스템 사이에서 전송되는 정보들의 방향성 식별

**송수신 데이터 식별**

- 개발할 시스템과 연계할 시스템 사이의 정보 흐름과 데이터베이스 산출물을 기반으로 식별

- 인터페이스 표준 항목과 송수신 데이터 항목 식별

- 코드성 데이터 항목 식별

### 인터페이스 방법 명세화

**인터페이스 방법 명세화의 개념**

- 내외부 시스템이 연계하여 작동할 때 인터페이스별 송수신 방법, 손수신 데이터, 오류 식별 및 처리 방안에 대한 내용을 문서화해놓은 것

**시스템 연계 기술**

- 개발할 시스템과 내외부 시스템을 연계될 때 사용하는 기술

- DB Link : DB에서 제공하는 DB Link 객체 이용

- API/Open API : 송신 시스템의 DB에서 데이터를 읽어 와 제공하는 Application Programming Interface 프로그램

- 연계 솔루션 : EAI 서버와 송수신 시스템에 설치되는 클라이언트 이용

    -> EAI : 송수신 데이터를 식별하기 위해 송수신 처리 및 진행 현황을 모니터링하고 통제하는 시스템

- Socket : 서버에서 소켓을 생성하여 클라이언트의 통신 요청 시 클라이언트와 연결하여 통신하는 네트워크 기술

- Web Service : 웹 서비스에서 WSDL, UDDI, SOAP 프로토콜을 이용하여 연계하는 서비스

**인터페이스 통신 유형**

- 데이터를 송수신 하는 형태

- 단방향 : 시스템에서 거래 요청 후 응답 없음

- 동기 : 시스템에서 거래 요청 후 응답이 올 때까지 대기

- 비동기 : 시스템에서 거래를 요청 후 응답이 올때까지 다른 작업을 하면서 대기

**인터페이스 처리 유형**

- 송수신 데이터를 어떤 형태로 처리할 것인지에 대한 방식

- 실시간 방식, 지연 처리 방식, 배치 방식(대량 데이터 처리)

**인터페이스 발생 주기**

- 송수신 데이터가 전송되어 인터페이스가 사용되는 주기

**송수신 방법 명세화**

- 각각의 인터페이스에 대해 연계 방식, 통신 유형, 처리 유형, 발생 주기 등 송수신 방법을 정의하고 명세

**송수신 데이터 명세화**

- 인터페이스 시 필요한 송수신 데이터에 대한 명세 작성

**오류 식별 및 처리 방안 명세화**

- 인터페이스 시 발생할 수 있는 오류를 식별하고 오류 처리 방안에 대한 명세 작성

### 시스템 인터페이스 설계서 작성

**시스템 인터페이스 설계서의 개요**

- 시스템의 인터페이스 현황을 확인하기 위해 시스템이 갖는 인터페이스 목록과 상세 데이터 명세를 정의한 문서

**시스템 인터페이스 목록 작성**

- 업무 시스템과 내외부 시스템 간 데이터를 주고받는 경우에 사용하는 인터페이스에 대해 기술

**시스템 인터페이스 정의서 작성**

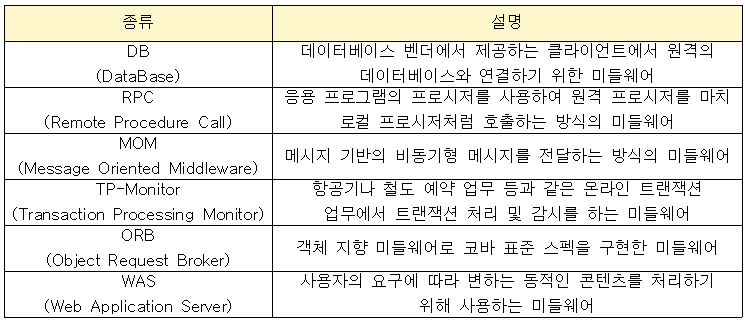
- 인터페이스별로 시스템 간의 연계를 위해 필요한 데이터 항목 및 구현 요건 등을 기술

### 미들웨어 솔루션 명세

**미들웨어의 개념 및 종류**

- 운영체제와 응용 프로그램 사이에서 운영체제가 제공하는 서비스 이외에 추가적인 서비스를 제공하는 소프트웨어

- 표준화된 인터페이스를 제공하여 시스템 간의 데이터 교환에 일관성을 보장



**미들웨어 솔루션 식별**

- 개발 및 운용 환경에 사용될 미들웨어 솔루션을 확인하고 목록을 작성

**미들웨어 솔루션 명세서 작성**

- 미들웨어 솔루션 목록의 미들웨어 솔루션별로 관련 정보들을 상세하게 기술

## [1과목 예상 키워드 정리](http://127.0.0.1:5503/184)

2020. 6. 4. 22:43

### 스크럼 개발 프로세스

스프린트 계획 회의 -> 스프린트 -> 일일 스크럼 회의 -> 스프린트 검토 회의 -> 스프린트 회고

### XP 핵심 가치

의사소통, 단순성, 용기, 존중, 피드백

### 요구사항 개발 프로세스

도출 -> 분석 -> 명세 -> 확인

### UML

* 사물 : 구조, 행동, 그룹, 주해
* 관계 : 집합(◇ㅡ), 포함(◆ㅡ), 의존(<·······), 일반화(◁ㅡ), 실체화(◁---)
* 다이어그램
  + 정적 모델링 -> 구조적 다이어그램 : 클래스, 객체, \*\***컴포넌트, 배치(구현단계)**\*\*, 복합체, 패키지
  + 동적 모델링 -> 행위 다이어그램 : 유스케이스, 시퀀스, 커뮤니케이션, 상태, 활동, 상호작용, 타이밍

### UI

* 와이어프레임 : 뼈대
* 목업 : 정적형태
* 스토리보드 : 와이어프레임 + 이동 흐름, 설명
* 프로토 타입 : 인터랙션, 테스트

### 품질 요구사항

* 기능성(적합성, 정밀성, 상호운용성, 보안성, 호환성)
* 신뢰성(성숙성, 고장 허용성, 회복성)
* 사용성(이해성, 학습성, 운용성, 친밀성)
* 효율성(시간 / 자원 효율성)
* 유지보수성(분석성, 변경성, 안정성, 시험성)
* 이식성(적용성, 설치성, 대체성, 공존성)

### 소프트웨어 아키텍쳐

* 모듈화 : 모듈의 개수와 크기는 반비례, 개수와 비용은 비례
* 추상화 : 세분화하여 구체화
  + 과정 추상화 : 전반적인 흐름
  + 데이터 추상화 : 데이터의 세부사항 정의 x
  + 제어 추상화 : 이벤트의 발생의 세부사항 정의 x

### 아키텍쳐 패턴

* 레이어 패턴
  + 계층식 마주보는 두 계층사이 상호작용
  + 상위 : 서비스 제공자 / 하위 : 클라이언트
* 클라이언트 서버
  + 하나의 서버 다수의 클라이언트
  + 요청을 위한 동기화를 제외하곤 독립적
* 파이프 필터
  + 데이터 스트름 절차의 각 단계를 캡슐화하여 파이프를 통해 전송
  + 데이터 변환, 버퍼링, 동기화
* MVC(모델 뷰 컨트롤러) 패턴
  + 모델 : 서브시스템의 핵심 기능과 데이터 보관
  + 뷰 : 사용자에게 정보 표시
  + 컨트롤러 : 사용자로부터 받은 입력 처리
  + 대화형 애플리케이션
* 마스터-슬레이브 : 작업 분할 후 다시 돌려받음
* 브로커 패턴 : 브로커 컴포넌트가 요청에 맞는 컴포넌트 연결
* 피어투피어 패턴 : 피어가 서버도 되고 클라이언트도 됨
* 이벤트 버스 : 구독한 리스너에게 메시지 받아 이벤트 처리
* 블랙보드 : 검색을 통해 블랙보드에서 데이터 찾음

### 모듈

* 결합도는 약하게 응집도는 강하게
* 결합도 : 모듈 간의 관계
  + 내용 : 내부 기능, 내부 자로 참조
  + 공통 : 공통 데이터 영역
  + 외부 : 외부의 다른 모듈에서 참조
  + 제어 : 제어 신호 이용하여 통신하거나 요소를 전달
  + 스탬프 : 자료구조가 전달됨
  + 자료 : 매개변수 데이터 넘겨줌
* 응집도 : 모듈 내부 요소가 관련된 정도
  + 기능적 : 단일 문제와 연관
  + 순차적 : 모듈에서 나온 데이터가 다음 모듈의 입력
  + 통신적 : 동일한 입력과 출력하여 서로 다른 기능
  + 절차적 : 다수 기능 시 기능을 순차적으로 수행
  + 시간적 : 특정 시간
  + 논리적 : 유사한 성격 혹은 특정 형태
  + 우연적 : 관련 없는 요소들로 구성

### 디자인 패턴

* 생성 패턴
  + 추상 팩토리 : 구체적인 클래스에 의존하지 않고 인터페이스를 통해 서로 연관, 의존하는 객체들의 그룹으로 생성하여 추상적으로 표현
  + 빌더 : 작게 분리된 인스턴스를 건축 하듯이 조합하여 객체 생성
  + 팩토리 메소드 : 객체 생성을 서브 클래스에서 처리하도록 분리하여 캡슐화한 패턴
  + 프로토타입 : 원본 객체를 복제하는 방법으로 객체를 생성하는 패턴
  + 싱글톤 : 하나의 객체를 생성하면 생성된 객체를 어디서든 참조할 수 있지만 여러 프로세스가 동시에 참조할 수는 없음
* 구조 패턴 : 클래스나 객체들을 조합하여 더 큰 구조로 만들 수 있게 해주는 패턴
  + 어댑터 : 호환성이 없는 클래스들의 인터페이스를 다른 클래스가 이용할 수 있도록 변환해주는 패턴
  + 브리지 : 구현부에서 추상층을 분리하여 서로가 독립적으로 확장할 수 있도록 구성한 패턴
  + 컴포지트 : 여러 객체를 가진 복합 객체와 단일 객체를 구분 없이 다루고자 할 때 사용하는 패턴
  + 데코레이터 : 객체 간의 결합을 통해 능동적으로 기능들을 확장할 수 있는 패턴
  + 퍼싸드 : 복잡한 서브 클래스들을 피해 더 상위에 인터페이스를 구상함으로써 서브 클래스의 기능을 간편하게 사용할 수 있도록 하는 패턴
  + 플라이웨이트 : 인스턴스가 필요할 때마다 생성하는 것이 아닌 공유해서 사용함으로써 메모리를 절약하는 패턴
  + 프록시 : 접근이 어려운 객체와 여기에 연결하려는 객체 사이에서 인터페이스 역할을 수행하는 패턴
* 행위 패턴 :클래스나 객체들이 서로 상호작용하는 방법이나 책임 분배 방법을 정의한 패턴
  + 책임 연쇄 : 요청을 처리할 수 있는 객체가 둘 이상 존재하여 한 객체가 처리하지 못하면 다음 객체로 넘어가는 형태의 패턴
  + 커맨드 : 요청을 객체의 형태로 캡슐화하여 재이용하거나 취소할 수 있도록 요청에 필요한 정보를 저장하거나 로그에 남기는 패턴
  + 인터프리터 : 언어에 문법 표현을 정의하는 패턴
  + 반복자 : 자료 구조와 같이 접근이 잦은 객체에 대해 동일한 인터페이스를 사용하도록 하는 패턴
  + 중재자 : 수많은 객체들 간의 복잡한 상호작용을 캡슐화하여 객체로 정의하는 패턴
  + 메멘토 : 특정 시점에서의 객체 내부 상태를 객체화함으로써 이후 요청에 따라 객체를 해당 시점의 상태로 돌릴 수 있는 기능을 제공하는 패턴
  + 옵서버 : 한 객체의 상태가 변화하면 객체에 상속되어 있는 다른 객체들에게 변화된 상태를 전달하는 패턴
  + 상태 : 객체의 상태에 따라 동일한 동작을 다르게 처리해야 할 때 사용하는 패턴
  + 전략 : 동일한 계열의 알고리즘들을 개별적으로 캡슐화하여 상호 교환할 수 있게 정의하는 패턴
  + 템플릿 메소드 : 상위 클래스에서 골격을 정의하고 하위 클래스에서 처리를 구체화하는 구조의 패턴
  + 방문자 : 각 클래스들의 데이터 구조에서 처리 기능을 분리하여 별도의 클래스로 구성하는 패턴

### 요구사항 검증 방법

* 동료 검토 : 직접 설명하는걸 동료들이 들으면서 결함 발견
* 워크 스루 : 미리 배포하여 사전 검토 후 짧은 회의를 통해 발견
* 인스펙션 : 검토 전문가들이 발견
* CASE : 일관성 분석을 통해 표준 준수 여부 확인

### 시스템 연계 기술

* DB Link, API/Open API, EAI(송수신 처리 현황 모니터링, 통제), Socket, Web Service

### 미들웨어

* RPC : 프로시저를 사용하여 원격 프로시저를 로컬 프로시저처럼 호출
* MOM : 메시지 기반의 비동기형 메시지 전달
* TP-Monitor : 트랜잭션 처리 및 감시
* ORB : 객체 지향 미들웨어
* WAS : 동적인 콘텐츠 처리

## [2.1 데이터 입출력 구현](http://127.0.0.1:5503/154)

2020. 4. 13. 23:27

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

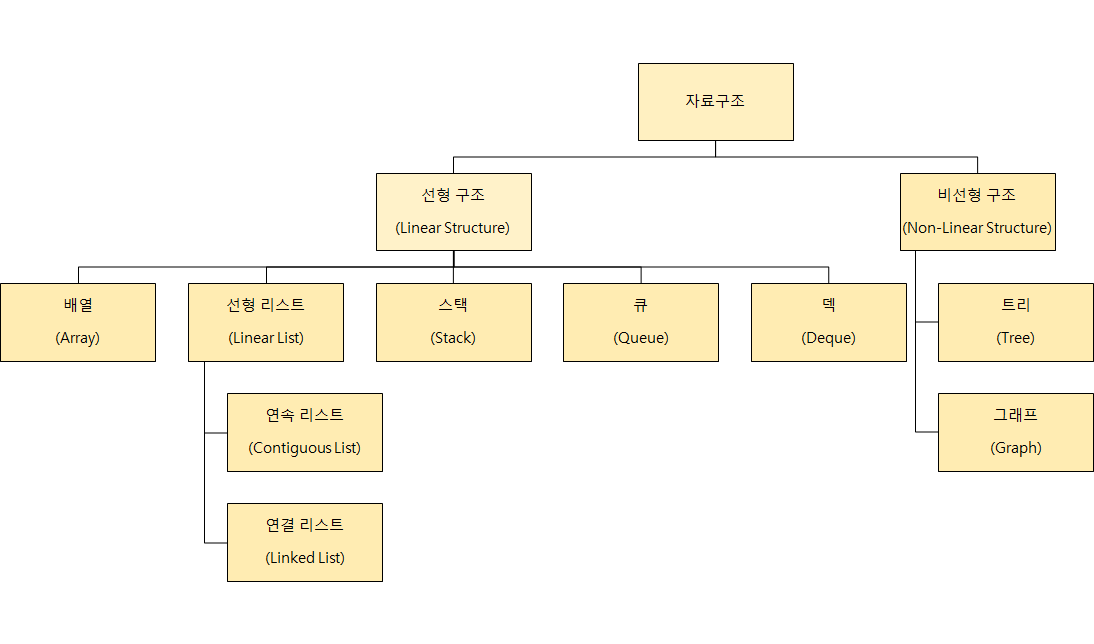
728x90

### 자료구조

**자료 구조의 정의**

- 프로그램에서 사용하기 위한 자료를 기억장치의 공간 내에 저장하는 방법과 자료 간의 관계, 처리 방법 등을 저장공간의 효율성 및 실행 간의 신속성을 높이기 위한 연구 분석하는 것

**자료 구조의 분류**



- 배열

    -> 동일한 자료형의 데이터들이 같은 크기로 나열되어 순서를 갖고 있는 집합

    -> 첨자를 이용하여 데이터에 접근

    -> 첨자의 개수에 따라 n차원 배열이라 부름

- 선형 리스트

    -> 일정한 순서에 의해 나열된 자료 구조

    -> 연속 리스트

        \* 배열을 이용한 선형 리스트

        \* 중간에 데이터를 삽입하기 위해 연속된 빈 공간이 있어야 하며 삽입, 삭제 시 자료의 이동 필요

    -> 연결 리스트

        \* 자료 항목의 순서에 따라 노드의 포인터 부분을 이용하여 서로 연결시킨 자료 구조

        \* 연결을 위한 포인터를 찾는 시간이 필요해 접근 속도가 느림

        \* 노드의 삽입 삭제 작업이 용이

        \* 노드 부분 때문에 연속 리스트에 비해 기억 공간의 효율이 좋지 않음

https://blog.kakaocdn.net/dn/bJe01h/btqDof0wQtR/JIiMXyTZaPqHiFlWg6WIX1/img.png노드

- 스택

    -> 리스트의 한쪽으로 자료의 삽입, 삭제가 이루어짐

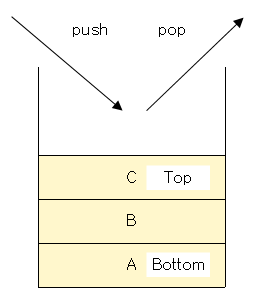
    -> LIFO(Last In First Out)의 구조를 가지고 있음

    -> 오버플로우(overflow) : 기억 공간이 모두 차있는 상태에서 데이터를 삽입하면 일어나는 현상

    -> 언더플로우(underflow) : 기억 공간이 비어있는 상태에서 데이터를 삭제하면 일어나는 현상

    -> Top : 스택에 가장 마지막으로 삽입된 자료의 위치

    -> Bottom : 스택의 가장 바닥

스택

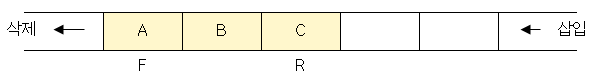
- 큐

    -> 리스트의 한쪽에서는 삽입 다른 한쪽에서는 삭제가 이루어짐

    -> FIFO(First In First Out)의 구조를 가지고 있음

    -> F(Front) : 먼저 삽입된 자료의 기억 공간을 가르키는 포인터

    -> R(Rear) : 마지막에 삽입된 자료의 기억 공간을 가르키는 포인터



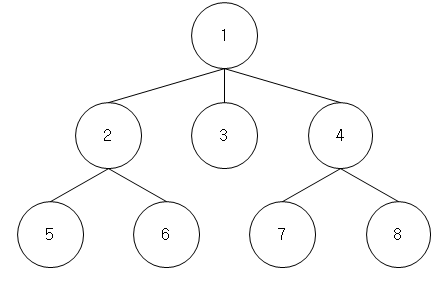
- 트리

    -> 노드와 가지를 이용하여 사이클이 없이 구성한 그래프의 특수 형태

    -> 디그리 : 노드에서 뻗어나온 가지의 개수

    -> 단말노드 / 잎 노드 : 자식이 없는 노드

    -> 트리의 디그리 : 노드들의 디그리 중 가장 많은 수



### 데이터저장소 / 데이터베이스 / DBMS

**데이터저장소의 개요**

- 소프트웨어 개발 과정에서 다루어야 할 데이터들을 논리적인 구조로 조직화하거나 물리적인 공간에 구현한 것

- 논리 데이터저장소는 데이터 및 데이터 간의 연간성, 제약 조건을 식별하여 논리적인 구조로 조직화한 것

- 물리 데이터저장소는 논리 데이터저장소에 저장된 데이터와 구조들을 하드웨어적인 저장장치에 저장한 것

**데이터베이스의 정의**

- 특정 조직의 업무를 수행하는데 필요한 데이터들의 모임

- 통합된 데이터 : 자료의 중복을 최소화

- 저장된 데이터 : 컴퓨터가 접근할 수 있는 저장 매체에 저장

- 운영 데이터 : 조직의 고유한 업무를 수행하는데 필요

- 공용 데이터 : 여러 시스템이 공동으로 소유하고 유지함

**DBMS(DataBase Management System)**

- 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해주고 데이터베이스를 관리하는 소프트웨어

- 기존의 파일 시스템이 가지는 데이터의 종속성과 종복성 문제를 해결하기 위해 제안된 시스템

- DBMS의 기능

    -> 정의 기능 : 데이터베이스에 저장될 데이터의 타입과 구조에 대해 명시하는 기능

    -> 조작 기능 : 데이터를 검색, 갱신, 삽입, 삭제 등 처리하기 위해 사용자와 데이터베이스 간 인터페이스 수단을 제공하는 기능

    -> 제어 기능

        \* 데이터의 무결성이 유지되도록 제어

        \* 사용자에게 허가된 데이터만 접근하도록 보안을 유지하고 권한을 검사

        \* 여러 사용자가 동시에 접근하여 데이터를 처리할 때 정확성을 유지하도록 병행 제어

**DBMS의 장단점**

- 장점

    -> 데이터 독립성, 일관성, 무결성 유지

    -> 보안 유지

    -> 데이터 실시간 처리, 통합 관리, 표준화 가능

- 단점

    -> 전문가 부족

    -> 전산화 비용 증가

    -> 파일의 백업과 회복이 어려움

    -> 시스템이 복잡함

### 데이터 입출력

**데이터 입출력의 개요**

- 소프트웨어의 기능을 구현하기 위해 데이터베이스에 데이터를 입력, 출력하는 작업

**SQL**

- 국제 표준 데이터베이스 언어

- 데이터 정의어, 조작어, 제어어로 구분됨

**데이터 접속(Data Mapping)**

- 프로그래밍 코드와 데이터베이스의 데이터를 연결하는 것

**트랜잭션**

- 하나의 논리적 기능을 수행하기 위한 작업의 단위 또는 한꺼번에 수행돼야 할 일련의 연산

- TCL(Transaction Control Language) : 트랜잭션을 제어하기 위해 사용되는 명령어

    -> COMMIT : 트랜잭션 처리가 비정상적으로 종료되어 트랜잭션이 수행한 변경 내용을 데이터베이스에 반영

    -> ROLLBACK : 트랜잭션 처리가 비정상적으로 종료되어 데이터베이스의 일관성이 깨졌을 때 트랜잭션이 행한 모든 변경 작업을 취소하고 이전 상태로 되돌림

    -> SAVEPOINT(CHECKPOINT) : 트랜잭션 내에 ROLLBACK 할 위치인 저장점을 지정

### 절차형 SQL

**절차형 SQL의 개요**

- 프로그래밍 언어와 같이 연속적인 실행이나 분기, 반복 등의 제어가 가능한 SQL

- 단일 SQL문장으로 처리가 어려운 연속적인 작업을 처리하는데 적합

- BEGIN ~ END 형식의 블록 구조로 되어 있어 기능별 모듈화가 가능

## [2.2 통합 구현](http://127.0.0.1:5503/155)

2020. 4. 14. 00:28

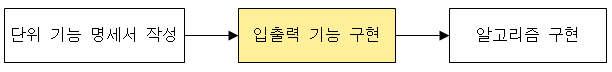
728x90

### 단위 모듈 구현

**단위 모듈의 개요**

- 소프트웨어 구현에 필요한 여러 동작 중 한 가지 동작을 수행하는 기능을 모듈로 구현한 것

- 사용자 또는 다른 모듈로부터 값을 전달받아 시작되는 작은 프로그램

단위 모듈 구현 순서

**단위 기능 명세서 작성**

- 단위 기능을 명세화한 문서

- 복잡한 시스템을 단순하게 구현하기 위한 추상화 작업이 필요

- 대형 시스템을 분해하여 단위 기능별로 구분하고 각 기능들로 계층적으로 구성하는 구조화 과정을 거침

**입출력 기능 구현**

- 단위 기능 명세서에서 정의한 데이터 형식에 따라 입출력 기능을 위한 알고리즘 및 데이터 구현

- 모듈 간 연동 또는 통신을 위한 데이터 구현

- IPC(Inter Process Communication) : 모듈 간 통신을 구현하기 위해 사용되는 프로그래밍 인터페이스 집합

    -> 공유 메모리 : 다수의 프로세스가 공유 가능한 메모리를 구성하여 통신 수행

    -> 소켓 : 네트워크 소켓을 이용하여 네트워크를 경유하는 통신 수행

    -> 세마포어 : 공유 자원에 대한 접근 제어를 통해 통신 수행

    -> 파이프 : 선입선출의 형태로 구성된 메모리를 여러 프로세스가 공유하여 통신 수행

    -> 메시지 큐잉 : 메시지가 발생하면 이를 전달하는 형태로 통신 수행

**알고리즘 구현**

- 입출력 데이터를 바당으로 단위 기능별 요구 사항들을 구현 가능 언어를 이용하여 모듈로 구현

### 단위 모듈 테스트

**단위 모듈 테스트의 개요**

- 모듈이 정해진 기능을 정확히 수행하는지 검증

- 단위 테스트라고도 하며 화이트박스 테스트와 블랙박스 테스트 기법 사용

- 시스템 수준의 오류는 발견할 수 없음

**테스트 케이스**

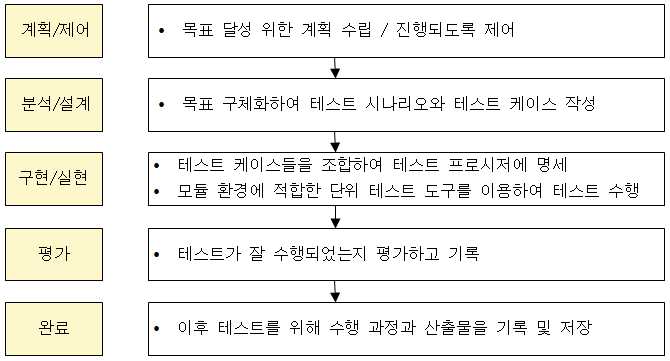
- 구현된 소프트웨어가 요구사항을 정확히 준수했는지 확인하기 위한 테스트 항목에 대한 명세서로 명세 기반 테스트의 설계 산출물에 해당

- 입력 데이터, 테스트 조건, 예상 결과 등을 모아 테스트 케이스를 만듦

- 테스트 케이스의 구성 요소 : 식별자, 테스트 항목, 입력 / 출력 명세, 환경 설정, 특수 절차 요구, 의존성 기술

**테스트 프로세스**

- 테스트를 위해 수행하는 작업이 테스트의 목적과 조건을 달성할 수 있도록 도와주는 과정



### 개발 지원 도구

**통합 개발 환경(IDE)**

- 개발에 필요한 편집기, 컴파일러 디버거 등의 다양한 툴을 하나의 인터페이스로 통합하여 제공

- Eclipse, Visual Studio, Xcode, Android Studio, IDEA 등

**빌드 도구**

- 소스 코드 파일들을 컴퓨터에서 실행할 수 있는 제품 소프트웨어로 변환하는 과정 또는 결과물

- 소스 코드를 소프트웨어로 변환하는 과정에 필요한 전처리, 컴파일 등의 작업을 수행

- Ant : 자바 프로젝트의 공식적인 빌드 도구

- Maven : Ant의 대안으로 의존성을 설정하여 라이브러리 관리

- Gradle : 안드로이드 스튜디오의 공식 빌드 도구

**협업 도구**

- 개발에 참여하는 사람들이 서로 다른 작업 환경에서 프로젝트를 수행할 수 있도록 도와주는 도구

- 협업 소프트웨어, 그룹웨어라고도 함

- 협업 도구의 종류

    -> 프로젝트 및 일정 관리 : 구글 캘린더, 분더리스트, 트렐, 지라, 플로우 등

    -> 정보 공유 및 커뮤니케이션 : 슬랙, 잔디, 태스크 월드 등

    -> 디자인 : 스케치, 제플린 등

    -> 아이디어 공유 : 에버노트 등

    -> API 문서화 : 스웨거 등

    -> Git 웹 호스팅 서비스 : 깃허브 등

## [2.3 제품 소프트웨어 패키징(1)](http://127.0.0.1:5503/157)

2020. 4. 14. 23:00

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 소프트웨어 패키징

**소프트웨어 패키징의 개요**

- 실행 파일을 묶어 배포용 설치 파일을 만듦

- 사용자 중심으로 진행

- 모듈화 하여 일반 배포 형태로 패키징

**패키징 고려사항**

- 사용자의 운영체제, CPU, 메모리 등에 필요한 최소 환경 정의

- UI는 시각적인 자료와 함께 매뉴얼과 일치시켜 패키징

- 소프트웨어는 하드웨어와 함께 관리될 수 있도록 Managed Service 형태로 제공

**패키징 작업 순서**

https://blog.kakaocdn.net/dn/tvV6d/btqDqi3YpSP/8e5aU5etuwmdBh5RIcYmEK/img.png

- 배포는 온라인 또는 오프라인으로 배포

### 릴리즈 노트 작성

**릴리즈 노트의 개요**

- 개발 과정에서 정의된 릴리즈 정보를 고객에게 공유하기 위한 문서

- 테스트 진행 방법에 대한 결과가 소프트웨어 사양에 대한 개발팀의 정확한 준수 여부 파악

- 소프트웨어의 버전 관리 및 릴리즈 정보를 체계적으로 관리

- 소프트웨어 초기 배포, 출시 후 개선 사항을 적용한 추가 배포 시 제공

**릴리즈 노트 초기 버전 작성 시 고려사항**

- 정확하고 완전한 정보를 기반으로 개발팀에서 직접 현재 시제로 작성

- 신규 코드, 빌드 등의 이력이 정확하게 관리되어 변경 또는 개선된 항목에 대한 이력 정보들도 작성

**릴리즈 노트 추가 버전 작성 시 고려사항**

- 테스트 과정에서 베타 버전이 출시되거나 긴급 버그 수정, 업그레이드, 사용자 요청 등의 특수한 상황의 경우 작성

- 긴급 버그 수정 시 수정하는 경우 릴리즈 버전을 출시하고 그 번호를 포함한 모든 내용을 수정된 내용을 담음

- 요구사항에 의해 추가 혹은 수정된 경우 자체 기능 향상과는 다른 별도의 릴리즈 버전으로 출시하고 작성

**릴리즈 노트 작성 순서**

https://blog.kakaocdn.net/dn/cOQGhQ/btqDpgFu9Ja/CmbkYk3dfto2Ay7T1k9k0k/img.png

### 디지털 저작권 관리

**저작권의 개요**

- 창작자가 가지는 배타적 독점적 권리로 타인의 침해를 받지 않을 고유한 권한

- 컴퓨터 프로그램처럼 복제하기 쉬운 저작물에 대해 저작권을 보호하는 방법을 저작권 보호 기술이라 함

**디지털 저작권 관리(Digital Right Management)의 개요**

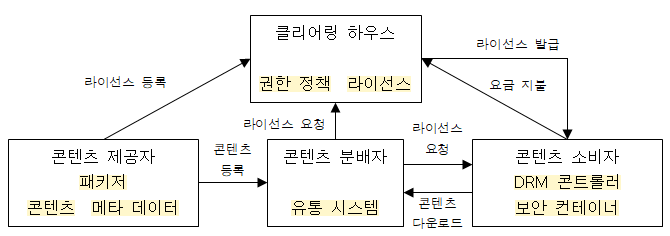
- 저작권자가 배포한 디지털 콘텐츠가 저작권자가 의도한 용도로만 사용되도록 생성, 유통, 이용까지 전 과정에 걸쳐 사용되는 디지털 콘텐츠 관리 및 보호 기술

- 크기가 작은 경우 사용자가 콘텐츠를 요청하는 시점에 실시간 패키징 수행

- 크기가 큰 경우 미리 패키징을 수행 후 배포

- 종량제 방식을 적용한 소프트웨어의 경우 서비스의 실제 사용량을 측정하여 이용한 만큼 이용 부과

**디지털 저작권 관리의 흐름도**



- 클리어링 하우스 : 저작권에 대한 사용 권한, 라이선스 발급, 사용량에 따른 결제 관리 등 수행

- 콘텐츠 제공자 : 콘텐츠를 제공하는 저작권자

- 패키저 : 콘텐츠를 메타 데이터와 함께 배포 가능한 형태로 묶어 암호화는 프로그램

- 콘텐츠 분배자 : 암호화된 콘텐츠를 유통

- 콘텐츠 소비자 : 콘텐츠를 구매해서 사용

- DRM 컨트롤러 : 배포된 콘텐츠의 이용 권한을 통제하는 프로그램

- 보안 컨테이너 : 콘텐츠 원본을 안전하게 유통하기 위한 전자적 보안 장치

**디지털 저작권 관리의 기술 요소**

- 암호화, 키 관리, 암호화 파일 생성, 식별 기술, 저작권 표현, 정책 관리, 크랙 방지, 인증

### 소프트웨어 설치 매뉴얼 작성

**소프트웨어 설치 매뉴얼의 개요**

- 개발 초기에서부터 적용된 기준이나 사용자가 소프트웨어를 설치하는 과정에 필요한 내용을 기록한 문서

- 설치 시작부터 완료까지의 과정을 순서대로 설명

**서문**

- 문서 이력, 설치 매뉴얼의 주석, 설치 도구의 구성, 설치 환경 체크 항목 기술

- 설치 매뉴얼의 주석 : 주의 사항과 참고 사항 기술

- 설치 환경 체크 항목 : 사용자 환경, 응용 프로그램, 업그레이드 버전, 백업 폴더 확인

**기본 사항**

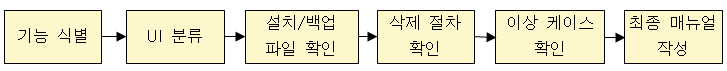
- 소프트웨어 개요, 설치 관련 파일, 설치 아이콘, 프로그램 삭제, 관련 추가 정보 설명

**설치 매뉴얼 작성**

- 사용자가 설치 과정을 이해하기 쉽게 설치 화면을 누락 없이 캡처하여 순서대로 설명

- 설치 화면 및 UI, 설치 이상 메시지, 설치 완료 및 결과, 설치 시 점검 사항, Network 환경 및 보안, 고객 지원 방법, FAQ, 준수 정보 & 제한 보증에 대해 기술

**설치 매뉴얼 작성 순서**



### 소프트웨어 사용자 매뉴얼 작성

**소프트웨어 사용자 매뉴얼의 개요**

- 사용자가 소프트웨어를 사용하는 과정에서 필요한 내용을 기록한 문서

**서문**

- 문서 이력, 사용자 매뉴얼의 주석, 기록 보관 내용 기술

- 기록 보관 내용 : 소프트웨어를 사용하면서 필요한 기술 지원이나 추가 정보를 얻기 위한 소프트웨어 등록 정보 기술

**기본 사항**

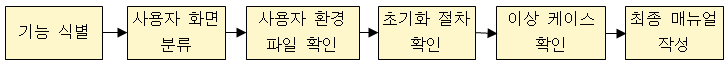
- 소프트웨어 개요, 사용 환경, 관리, 모델 버전별 특징, 기능 및 인터페이스의 특징, 구동 환경 설명

**사용자 매뉴얼 작성**

- 사용자가 사용방법을 이해하기 쉽도록 작성

- 사용자 화면 및 UI, 주요 기능 분류, 응용 프로그램 및 설정, 장치 연동, Network 환경, Profile 안내, 고객 지원 방법, 준수 정보 및 제한 보증에 대해 기술

**사용자 매뉴얼 작성 순서**



## [2.3 제품 소프트웨어 패키징(2)](http://127.0.0.1:5503/158)

2020. 4. 15. 01:22

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 소프트웨어 버전 등록

**소프트웨어 패키징 형상 관리**

- 형상관리는 소프트웨어의 변경 사항을 관리하기 위한 활동

**형상 관리의 중요성**

- 지속적으로 변경사항을 체계적으로 관리 및 추적할 수 있음

- 발견된 버그나 수정 사항을 추적

- 무절제한 변경 방지

**형상 관리 기능**

- 형상 식별 : 대상에 이름과 관리 번호를 부여하고 계층 구조로 구분하여 수정 및 추적이 용이하도록 하는 작업

- 버전 제어 : 소프트웨어 유지 보수 과정에서 생성된 다른 버전의 형상 항목을 관리하고 특정 절차와 도구를 결합하는 작업

- 형상 통제 : 식별된 형상 항목에 대한 변경 요구를 검토하여 현재의 기준선이 잘 반영될 수 있도록 하는 작업

- 형상 감사 : 기준선의 무결성을 평가하기 위해 확인, 검증, 검열 과정을 통해 공식적으로 승인하는 작업

- 형상 기록 : 형상의 식별, 통제, 감사 작업의 결과를 기록, 관리하고 보고서를 작성하는 작업

**소프트웨어 버전 등록 관련 주요 용어**

- 저장소(Repository) : 형상에 대한 정보들이 저장되어 있는 곳

- 가져오기 : 아무것도 없는 저장소에 처음으로 파일 복사

- 체크아웃 : 저장소에서 소스 파일, 버전 관리를 위한 파일을 받아옴

- 체크인 : 체크아웃으로 받아온 파일을 수정 후 저장소에 새로운 버전으로 갱신

- 커밋 : 체크인 수행 시 이전에 갱신된 내용이 있는 경우 충돌을 알리고 diff 도구를 이용해 수정한 후 갱신

- 동기화 : 저장소에 있는 최신 버전을 동기화

### 소프트웨어 버전 관리 도구

**공유 폴더 방식**

- 버전 관리 자료가 로컬 컴퓨터의 공유 폴더에 저장되어 관리

**클라이언트/서버 방식**

- 버전 관리 자료가 서버에 저장되어 관리

- 서버의 자료를 자신의 PC로 복사하여 작업 후 변경 내용을 서버에 반영

- 모든 버전 관리는 서버에서 수행

**분산 저장소 방식**

- 버전 관리 자료가 하나의 원격 저장소와 분산된 PC의 로컬 저장소에 함께 저장되어 관리

- 원격 저장소의 자료를 자신의 로컬 저장소로 복사하여 작업 후 변경 내용을 로컬 저장소에서 버전 관리 후 이를 원격 저장소에 반영

**Subversion(SVN)**

- 아파치 소프트웨어 재단에서 2000년에 발표

- 클라이언트/서버 방식

- 모든 작업은 trunk 디렉토리에서 추가 작업은 branches 디렉토리 안에 별도의 디렉토리를 만들어 작업 후 trunk 디렉토리와 병합

- 커밋 시 커밋의 버전인 리버전이 1씩 증가

- 서버는 주로 유닉스에서 사용

- 명령어

    -> add : 새로운 파일이나 디렉토리를 관리 대상으로 지정

    -> commit : add한 소스파일을 서버의 소스파일에 적용

    -> update : 서버의 최신 commit 이력을 클라이언트 소스에 적용

    -> checkout : 서버에서 버전 관리 정보와 소스 파일을 받아옴

    -> import : 아무것도 없는 서버의 저장소에 맨 처음 소스 파일을 저장

    -> export : 버전 관리 정보 빼고 소스 파일만 서버에서 받아옴

    -> info : 지정된 파일에 대한 정보를 표시

    -> diff : 지정된 파일이나 경로에 대해 이전 리버전과의 차이를 표시

    -> merge : 다른 디렉토리에서 작업된 버전 관리 내역을 기본 개발 작업과 병행

**Git**

- 리누스 토발즈가 2005년에 개발

- 분산 저장소 방식

- 버전 관리가 지역 저장소에서 진행되어 버전 관리가 신속하게 처리되고, 원격 저장소나 네트워크에 문제가 있어도 작업 가능

- 브랜치를 이용하여 기본 버전 관리 틀에 영향을 주지않으면서 다양항 형태의 테스팅 가능

- 파일의 변화를 스냅샷으로 저장하고 이전 스냅샷의 포인터를 가져 버전의 흐름 파악 가능

- 명령어

    -> add : 작업 내역을 스테이징 영역에 추가하여 버전 관리 대상으로 지정

    -> commit : 작업 내역을 지역 저장소에 저장

    -> branch : 새로운 브런치 생성 / 삭제

    -> checkout : 지정한 브런치로 이동

    -> merge : 두 브랜치 병합

    -> init : 지역 저장소 생성

    -> remote add : 원격 저장소에 연결

    -> push : 로컬 저장소의 변경 내용을 원격 저장소에 반영

    -> fetch : 원격 저장소의 변경 이력만 지역 저장소에 반영

    -> clone : 원격 저장소의 전체 내용을 지역 저장소로 복제

    -> fork : 지정한 원격 저장소의 내용을 자신의 원격 저장소로 복제

### 빌드 자동화 도구

**빌드 자동화 도구의 개념**

- 소스 코드를 컴파일한 후 여러 개의 모듈로 묶어 실행 파일로 만드는 과정을 포함하여 테스트 및 배포를 자동화하는 도구

**Jenkins**

- Java 기반의 오픈소스

- 서블릿 컨테이너에서 실행되는 서버 기반 도구

- 형상 관리 도구와 연동 가능

- Web GUI 제공으로 사용이 쉬움

- 여러 대의 컴퓨터를 이용한 분산 빌드나 테스트 가능

**Gradle**

- Groovy를 기반으로 한 오픈 소스 형태의 자동화 도구

- 안드로이드 앱 개발 환경에 사용

- Java, C/C++, Python 등의 언어도 빌드 가능

- Groovy를 사용해서 만든 DSL을 스크립트 언어로 사용

- 실행할 처리 명령들을 모아 태스크로 만든 후 태스크 단위로 실행

- 이전의 태스크를 재사용하거나 다른 시스템의 태스크를 공유하여 빌드의 속도를 향상시킬 수 있음

## [2.4 애플리케이션 테스트 관리(1)](http://127.0.0.1:5503/159)

2020. 4. 15. 18:19

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 애플리케이션 테스트

**애플리케이션 테스트의 개념**

- 애플리케이션에 잠재된 결함을 찾아내는 과정

- 확인(Validation) : 개발된 소프트웨어가 요구사항을 만족시키는지 사용자의 입장에서 확인

- 검증(Verification) : 기능을 제대로 수행하고 명세서에 맞게 만들었는지 개발자의 입장에서 점검

- 테스트 전 개발한 소프트웨어의 유형을 분류하고 특성을 정리해서 중점적으로 테스트할 사항을 정리

**애플리케이션 테스트의 필요성**

- 미리 오류를 발견하고 새로운 오류의 유입 예방

- 사용자의 요구사항에 만족하는지 테스트해 제품의 신뢰도 향상

**애플리케이션 테스트의 기본 원리**

- 잠재적인 결함을 줄일 수 있지만 소프트웨어 자체 결함이 없다곤 할 수 없음

- 결함은 특정 모듈에 집중되어 있어 애플리케이션의 20%에 해당하는 코드에서 80%의 결함이 발견된다고 하여 파레토 법칙을 적용하기도 함

- 살충제 패러독스 현상을 방지하기 위해 테스트 케이스를 지속적으로 보완 및 개선

- 테스트를 정황에 따라 다르게 진행

- 결함을 모두 제거해도 사용자의 요구사항을 만족할 수 없으면 안 됨

- 작은 부분에서 시작해서 점점 확대하며 진행

### 애플리케이션 테스트의 분류

**프로그램 실행 여부**

- 정적 테스트

    -> 프로그램을 실행하지 않고 소스코드나 명세서를 분석하여 테스트

    -> 개발 초기에 결함을 발견할 수 있어 비용이 절감

    ->워크 스루, 인스펙션, 코드 검사 등

- 동적 테스트

    -> 프로그램을 실행하여 테스트

    -> 개발의 모든 단계에서 진행

    -> 블랙박스 테스트, 화이트 박스 테스트

**테스트 기반**

- 명세 기반 테스트

    -> 사용자의 요구사항을 테스트 케이스로 만들어 구현하고 있는지 확인하여 테스트

    -> 동등 분할, 경계 값 분석

- 구조 기반 테스트

    -> 소프트웨어 내부 논리 흐름에 따라 테스트 케이스를 만들어 테스트

    -> 구문 기반, 결정 기반, 조건 기반, 결정 기반 등

- 경험 기반 테스트

    -> 테스터의 경험을 기반으로 테스트

    -> 요구사항에 대한 명세가 부족하거나 시간의 제약이 있는 경우

    -> 에러 추정, 체크 리스트, 탐색적 테스팅

**시각**

- 확인 테스트

    -> 사용자의 시각에서 결과를 테스트

    -> 요구사항을 만족하면서 정상적으로 동작이 되는지 테스트

- 검증 테스트

    -> 개발자의 시각에서 과정을 테스트

    -> 명세서에 맞게 완성되었는지 테스트

**목적에 따른 테스트**

- 회복 테스트 : 결함을 주고 잘 복구되는지 테스트

- 안전 테스트 : 시스템 보호 도구가 볼법적인 침입으로부터 보호할 수 있는지 테스트

- 강도 테스트 : 과부하 시 정상적으로 실행되는지 테스트

- 성능 테스트 : 응답 시간, 처리량 등을 테스트

- 구조 테스트 : 내부의 논리적인 경로, 소스 코드 복잡도 등을 평가

- 회귀 테스트 : 변경 혹은 수정에 따른 새로운 결함이 없는지를 테스트

- 병행 테스트 : 기존의 소프트웨어와 변경된 소프트웨어에 동일한 데이터를 입력하여 결과를 비교하는 테스트

### 동적 테스트

**화이트박스 테스트**

- 모듈의 원시 코드를 오픈하여 논리적인 모든 경로를 한번 이상 실행하면서 테스트하여 테스트 케이스를 설계

- 테스트 과정의 초기에 진행

- 설계된 절차에 초점을 둔 구조적 테스트

- 모듈 안의 동작을 직접 관찰

**화이트박스 테스트의 종류**

- 기초 경로 검사

    -> 테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 해주는 테스트 기법

    -> 테스트 측정 결과를 통해 실행 경로의 기초를 정의

- 제어 구조 검사

    -> 조건 검사 : 프로그램 내의 논리적 조건을 테스트

    -> 루프 검사 : 프로그램 내의 반복 구조에 초점을 맞춰 테스트

    -> 데이터 흐름 검사 : 프로그램 내의 변수의 정의와 사용의 위치에 초점을 맞춰 테스트

**화이트박스 테스트 검증 기준**

- 문장 검증 기준 : 모든 구문이 한 번 이상 수행되도록 설계

- 분기 검증 기준 : 모든 조건문이 한 번 이상 수행되도록 설계

- 조건 검증 기준 : 모든 조건문에 대해 참/거짓인 경우가 한번 이상 수행되도록 설계

- 분기/조건 기준 : 모든 조건문과 조건문에 포함된 개별 조건식의 결과가 참/거짓인 경우가 한번 이상 수행되도록 설계

**블랙박스 테스트**

- 소프트웨어가 수행할 특정 기능을 알기 위해 기능이 완전히 작동되는 것을 입증하는 기능 테스트

- 테스트 과정의 후반부에 진행

- 사용자의 요구사항 명세를 보면서 구현된 기능을 테스트

- 소프트웨어 인터페이스에서 실시

**블랙박스 테스트의 종류**

- 동치(동등) 분할 검사 : 입력 자료에 초점을 맞춰 테스트 케이스를 만들고 검사

- 경계값 분석 : 입력 조건의 경계값을 테스트 케이스로 선정하여 검사

- 원인-효과 그래프 검사 : 입력 데이터 간의 관계과 출력의 영향을 미치는 상황을 분석 후 효용성이 높은 테스트 케이스를 선정하여 검사

- 오류 예측 검사 : 과거 경험이나 확인자의 감각으로 테스트

- 비교 검사 : 여러 프로그램에 동일한 테스트 자료를 제공하여 동일한 출력이 나오는지 확인하는 검사

## [2.4 애플리케이션 테스트 관리(2)](http://127.0.0.1:5503/160)

2020. 4. 15. 19:53

728x90

### 개발 단계에 따른 애플리케이션 테스트

**단위 테스트**

- 코딩 직후 모듈이나 컴포넌트에 초점을 맞춰 테스트

- 인터페이스, 외부적 I/O, 자료 구조 등을 검사

- 사용자의 요구사항을 기반으로 한 기능성 테스트를 최우선으로 수행

- 구조 기반 테스트 : 화이트 박스 테스트를 시행하여 제어 흐름이나 조건 결정을 목적으로 함

- 명세 기반 테스트 : 블랙 박스 테스트를 시행하여 동등 분할이나 경계값 분석을 목적으로 함

**통합 테스트**

- 단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성하는 과정에서 테스트

- 모듈 간 또는 통합된 컴포넌트 간 상호 작용 오류 검사

- 비점진적 통합 방식

    -> 모든 모듈이 미리 결합되어 있는 프로그램 전체를 테스트

    -> 빅뱅 통합 테스트 방식

    -> 오류 발견 및 장애 위치 파악이 어려움

- 점진적 통합 방식

    -> 모듈 단위로 단계적으로 통합하면서 테스트

    -> 하향식 / 상향식 / 혼합식 테스트 방식

    -> 오류 수정이 용이하고 인터페이스 관련 오류를 완전히 테스트할 수 있음

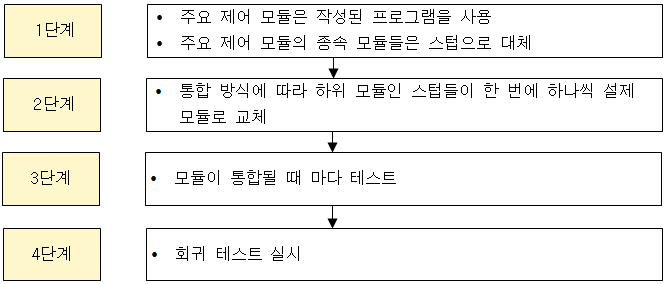
- 하향식 통합 테스트

    -> 상위 모듈에서 하위 모듈 방향으로 통합하면서 테스트

    -> 깊이 우선 통합법이나 넓이 우선 통합법 사용

    -> 상위 모듈에선 테스트 케이스 사용이 어려움

    -> 스텁 : 상위 모듈은 있지만 하위 모듈이 없는 경우 하위 모듈 대체

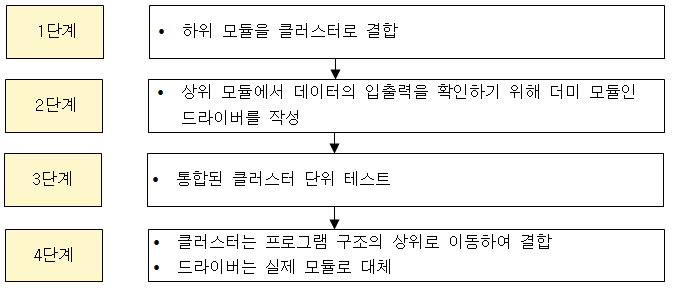


- 상향식 통합 테스트

    -> 하위 모듈에서 상위 모듈 방향으로 통합하면서 테스트

    -> 하나의 주요 제어 모듈과 종속 모듈의 그룹인 클러스터가 필요

    -> 드라이버 : 상위 모듈 없이 하위 모듈이 있는 경우 하위 모듈 구동



- 혼합식 통합 테스트

    -> 하위 수준에서는 상향식 통합 상위 수준에서는 하향식 통합을 사용하여 최적의 테스트를 지원

    -> 샌드위치 통합 테스트

- 회귀 테스팅

    -> 이미 테스트된 프로그램의 테스팅을 반복

    -> 통합 테스트로 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인

**시스템 테스트**

- 개발된 소프트웨어가 원하는 환경에서 수행되는지 테스트

- 실제 환경과 유사하게 만든 테스트 환경에서 진행

- 기능적 요구사항 : 명세서 기반의 블랙박스 테스트

- 비기능적 요구사항 : 구조적 요소에 대한 화이트박스 테스트

**인수 테스트**

- 개발한 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두고 테스트

- 사용자가 직접 테스트

- 사용자 인수 테스트 : 사용자가 시스템 사용의 적절성 여부 확인

- 운영상의 인수 테스트 : 시스템 관리자가 시스템 인수 시 수행

- 계약 인수 테스트 : 계약상의 조건을 준수하는지 확인

- 규정 인수 테스트 : 규정에 맞게 개발되었는지 확인

- 알파 테스트 : 개발된 환경에서 사용자가 개발자 앞에서 수행

- 베타 테스트 : 사용자의 환경에서 사용자가 직접 테스트 수행

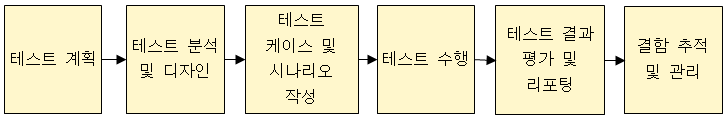
### 애플리케이션 테스트 프로세스

**애플리케이션 테스트 프로세스**

- 개발된 소프트웨어가 제대로 만들어 졌는지 테스트하는 절차

- 테스트를 마치면 테스트 계획서, 케이스, 시나리오, 결과서가 산출

- 에러는 빨리 발견될수록 좋음



- 테스트 계획 : 프로젝트 계획서 및 요구 명세서를 기반으로 테스트 목표를 정의하고 테스트 대상 및 범위 결정

- 테스트 분석 및 디자인 : 테스트의 목적과 원칙을 검토하고 사용자의 요구사항 분석

- 테스트 케이스 및 시나리오 작성 : 테스트 케이스를 작성, 검토 및 확인 후 시나리오 작성

- 테스트 수행 : 테스트 환경 구축 후 테스트 수행

- 테스트 결과 평가 및 리포팅 : 테스트 결과를 분석하여 테스트 결과 작성

- 결함 추적 및 관리 : 테스트 수행 후 결함이 어디에서 발생했고 어떤 결함인지 추적하고 관리

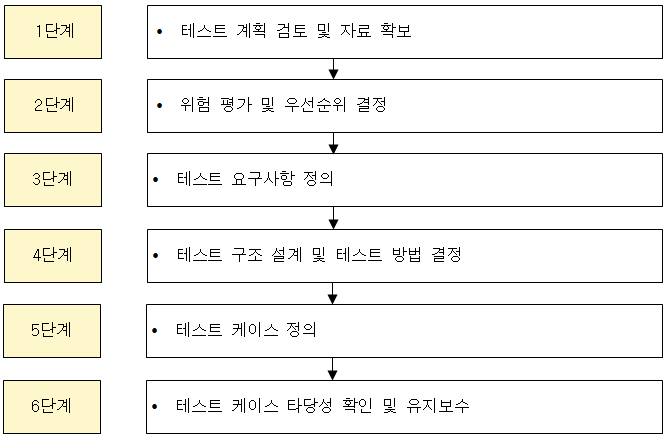
### 테스트 케이스 / 시나리오 / 오라클

**테스트 케이스**

- 사용자의 요구사항이 준수되었는지 확인하기 위해 테스트 항목에 대한 명세서

- 명세 기반 테스트의 설계 산출물

- 테스트 케이스 작성 순서



**테스트 시나리오**

- 테스트 케이스를 적용하는 구체적인 절차를 명세한 문서

**테스트 오라클**

- 테스트 결과가 올바른지 판단하기 위해 정의된 참 값을 대입하여 비교

- 테스트 오라클의 특징

    -> 제한된 검증 : 모든 테스트 케이스에는 적용 불가

    -> 수학적 기법 : 수학적 기법을 통해 테스트 오라클 값을 구할 수 있음

    -> 자동화 기능 : 테스트 대상에 대한 실행, 결과 비교 등을 자동화할 수 있음

- 테스트 오라클의 종류

    -> 참 오라클 : 모든 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공

    -> 샘플링 오라클 : 특정 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공

    -> 추정 오라클 : 특정 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하고 나머지 값에 대해서는 추정으로 처리

    -> 일관성 검사 오라클 : 변경 시 테스트 케이스 수행 전과 후의 결과 값이 동일한지 확인

### [필기 정리](https://1d1cblog.tistory.com/87)

## [2.4 애플리케이션 테스트 관리(3)](http://127.0.0.1:5503/161)

2020. 4. 15. 22:39

728x90

### 테스트 자동화 도구

**테스트 자동화의 개념**

- 반복적인 테스트 절차를 스크립트 형태로 구현하는 자동화 도구를 적용하여 쉽고 효율적으로 테스트 수행

**테스트 자동화 도구의 장단점**

- 장점

    -> 반복적인 작업을 자동화해 인력 및 시간 절감

    -> 향상된 테스트 품질 보장

    -> 사용자의 요구사항 등을 일관성 있게 검증

    -> 테스트 결과에 대한 객관적인 평가 기준 제공

    -> 테스트 결과를 다양한 표시 형태로 제공

    -> UI가 없는 서비스도 정밀 테스트 가능

- 단점

    -> 사용방법에 대한 교육 및 학습 필요

    -> 자동화 도구를 프로세스 단계별로 적용하기 위한 시간, 비용, 노력이 필요

**테스트 자동화 수행 시 고려사항**

- 모든 과정이 아닌 그때그때 맞는 적절한 도구를 선택

- 자동화 도구를 고려하여 프로젝트 일정 계획

- 프로젝트 초기에 테스트 엔지니어 투입 시기 계획

**테스트 자동화 도구의 유형**

- 정적 분석 도구 : 프로그램을 실행하지 않고 소스코드를 통해 결함을 발견

- 테스트 실행 도구 : 스크립트 언어를 사용하여 테스트를 실행

- 성능 테스트 도구 : 가상의 사용자를 만들어 테스트를 수행

- 테스트 통제 도구 : 테스트 계획 및 관리, 수행, 결함 관리 등을 수행

- 테스트 하네스 도구

    -> 테스트가 실행될 환경을 시뮬레이션하여 컴포넌트 및 모듈이 정상적으로 테스트되도록 함

    -> 구성요소 : 테스트 드라이버, 테스트 스텁, 테스트 슈트, 테스트 케이스, 테스트 스크립트, 목 오브젝트

**테스트 수행 단계별 테스트 자동화 도구**

- 테스트 계획 단계 : 요구사항 관리 도구

- 테스트 분석 및 설계 단계 : 테스트 케이스 생성 도구

- 테스트 수행 단계 : 테스트 자동화 / 정적 분석 / 동적 분석 / 성능 테스트 / 모니터링 도구

- 테스트 관리 단계 : 커버리지 분석 / 형상 관리 / 결함 추적 및 관리 도구

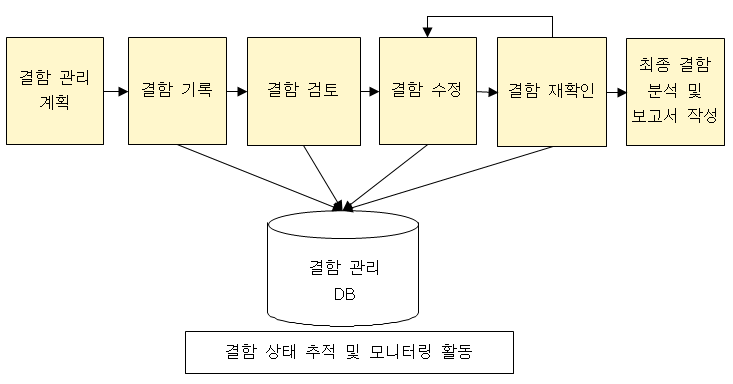
### 결함 관리

**결함의 정의**

- 소프트웨어가 개발자가 설계한 것과 다르게 동작하거나 다른 결과가 발생하는 것

**결함 관리 프로세스**

- 애플리케이션 테스트에서 발견된 결함을 처리



**결함 상태 추적**

- 테스트에서 발견된 결함은 지속적으로 상태 변화를 추적하고 관리해야 함

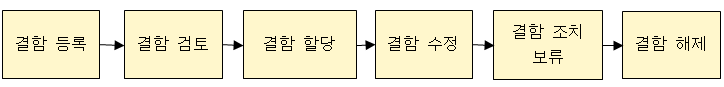
- 결함 분포 : 특정 속성에 해당하는 결함 수 측정

- 결함 추세 : 시간에 따른 결함 수의 추이 분석

- 결함 에이징 : 결함 상태로 지속되는 시간 측정

**결함 추적 순서**

- 결함이 발견되고 해결될 때까지의 과정



**결함 분류**

- 시스템 결함 : 주로 애플리케이션이나 데이터베이스 처리에서 발생된 결함

- 기능 결함 : 애플리케이션의 기획, 설계, 업무 시나리오 등의 단계에서 유입된 결함

- GUI 결함 : 화면 설계에서 발생된 결함

- 문서 결함 : 기획자, 사용자, 개발자 간 의사소통 및 기록이 원활하지 않아 발생된 결함

**결함 심각도**

- 결함이 전체 시스템에 미치는 치명도를 High, Medium, Low로 나눔

**결함 우선순위**

- 발견된 결함 처리에 대한 신속성을 나타내는 척도

- 결함의 중요도와 심각도에 따라 설정되고 수정 여부 결정

- Critical, High, Medium, Low 또는 즉시 해결, 주의 요망, 대기, 개선 권고 등으로 분류

**결함 관리 도구**

- Mantis : 소프트웨어 설계 시 단위 별 작업 내용을 기록할 수 있어 결함 및 이슈 관리, 추적 도구

- Trac : 결함 추적 및 통합 관리 도구

- Redmine : 프로젝트 관리 및 결함 추적 도구

- Bugzilla : 결함을 지속적으로 관리하고 심각도와 우선순위를 지정할 수 있는 도구

### 애플리케이션 성능 분석

**애플리케이션 성능**

- 사용자가 요구한 기능을 최소한의 자원을 사용하여 최대한 많은 기능을 신속하게 처리하는 정도

- 측정 지표 : 처리량, 응답 시간, 경과 시간, 자원 사용률

**성능 테스트 도구**

- 애플리케이션의 성능을 테스트 하기 위해 부하나 스트레스를 가해 성능 측정 지표를 점검하는 도구

- JMeter : 다양한 프로토콜을 지원하는 부하 테스트 도구

- LoadUI : 사용자의 편리성이 강화된 부하 테스트 도구

- OpenSTA : HTTP, HTTPS 프로토콜에 대한 부하 테스트 및 생산품 모니터링 도구

**시스템 모니터링 도구**

- 애플리케이션 실행 중 시스템 자원의 사용량을 확인하고 분석하는 도구

- 성능 저하의 원인 / 시스템 부하량 / 사용자 분석과 같은 시스템을 안정적으로 운영할 수 있는 기능 제공

- Scouter, Zabbix

**애플리케이션 성능 저하 원인 분석**

- 애플리케이션을 DB에 연결하기 위해 커넥션 객체를 생성하거나 쿼리를 실행하는 애플리케이션 로직에서 자주 발생

### 애플리케이션 성능 개선

**소스코드 최적화**

- 나쁜 코드를 배제하고 클린 코드로 작성

- 클린 코드 작성 원칙 : 가독성, 단순성, 의존성 배제, 중복성 최소화, 추상화

**소스 코드 최적화 유형**

- 클래스 분할 배치, 느슨한 결합, 코딩 형식 준수, 좋은 이름 사용, 적절한 주석문 사용

## [2.5 인터페이스 구현(1)](http://127.0.0.1:5503/162)

2020. 4. 16. 01:01

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 모듈 간 공통 기능 및 데이터 인터페이스 확인

**모듈 간 공통 기능 및 데이터 인터페이스의 개요**

- 공통 기능 : 모듈에 공통적으로 제공되는 기능

- 데이터 인터페이스 : 모듈 간 교환되는 데이터가 저장될 파라미터

- 인터페이스 설계서에서 정의한 모듈의 기능을 기반으로 확인

**인터페이스 설계서**

- 교환 데이터 및 관련 업무, 송수신 시스템 등에 대한 내용을 정리한 문서

- 일반적인 인터페이스 설계서

    -> 인터페이스 목록, 상세 데이터 명세, 기능의 세부 정보를 정의한 문서

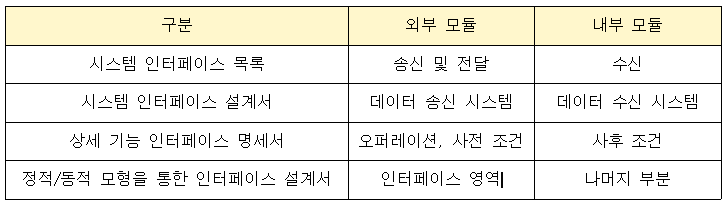
    -> 시스템 인터페이스 설계서 : 시스템 인터페이스 목록과 상세 데이터 명세를 정의

    -> 상세 기능별 인터페이스 명세서 : 기능의 세부 인터페이스 정보 정의

- 정적/도형 모형을 통한 인터페이스 설계서

    -> 시스템의 구성요소를 다이어그램으로 표현하여 만든 문서

**인터페이스 설계서 별 모듈 기능 확인**



### 모듈 연계를 위한 인터페이스 기능 식별

**모듈 연계의 개요**- 모듈 간 데이터 교환을 위해 관계를 설정

- EAI(Enterprise Application Intergration)

    -> 기업 내 정보 전달, 연계, 통합 등 상호 연동이 가능하게 해주는 솔루션

    -> Point-to-Point

        \* 애플리케이션끼리 1:1로 연결

        \* 변경 및 재사용이 어려움

    -> Hub & Spoke

        \* 단일 접점인 허브 시스템을 통해 데이터를 전송하는 중앙 집중형 방식

        \* 확장 및 유지보수 용이

        \* 허브 장애 시 전체 시스템에 영향

    -> Message Bus(ESB 방식)

        \* 애플리케이션 사이 미들웨어를 두어 처리

        \* 확장성이 뛰어나고 대용량 처리 가능

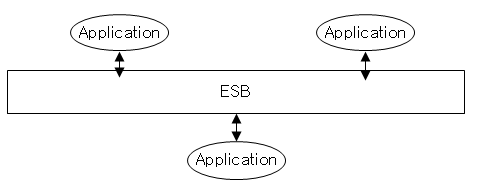
    -> Hybrid

        \* Hub & Spoke와 Message Bus의 혼합 방식

        \* 그룹 내에선 Hub & Spoke 방식을 그룹 간에는 Message Bus 방식 이용

        \* 데이터 병목 현상 최소화

- ESB(Enterprise Service Bus)



    -> 애플리케이션 간 표준 기반 인터페이스를 제공하는 솔루션

    ->애플리케이션보다는 서비스 중신의 통합을 지향

    -> 애플리케이션과의 결합도를 약하게 유지

    -> 관리 및 보안 유지가 쉽고 높은 수준의 품질 지원

**모듈 간 연계 기능 식별**

- 모듈 간 공통 기능 및 데이터 인터페이스를 기반으로 모듈과 연계된 기능을 시나리오 형태로 구체화하여 식별

- 인터페이스 기능을 식별하는 데 사용

**모듈 간 인터페이스 기능 식별**

- 식별된 모듈 간 기능을 검토하여 인터페이스 동작에 필요한 기능을 식별

- 해당 업무에 대한 시나리오를 통해 내부 모듈과 관련된 인터페이스 기능 식별

- 외부 및 인터페이스 모듈 간 동작하는 기능을 통해 인터페이스 기능 식별

### 모듈 간 인터페이스 데이터 표준 확인

**인터페이스 데이터 표준의 개요**

- 모듈 간 인터페이스에 사용되는 데이터의 형식을 표준화

**데이터 인터페이스 확인**

- 데이터 표준을 위해 식별된 데이터 인터페이스에서 입출력 값의 의미와 데이터의 특성 등을 구체적으로 확인

**인터페이스 기능 확인**

- 데이터 표준을 위해 식별된 인터페이스 기능을 기반으로 인터페이스 기능 구현을 위해 필요한 데이터 항목 확인

**인터페이스 데이터 표준 확인**

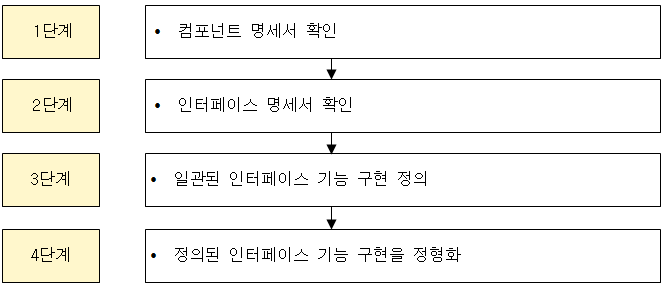
- 데이터 인터페이스에서 확인된 데이터 표준과 인터페이스 기능을 통해 확인된 데이터 항목을 검토하여 최종적으로 데이터 표준 확인

### 인터페이스 기능 구현 정의

**인터페이스 기능 구현 정의에 대한 개요**

- 인터페이스를 실제로 구현하기 위해 인터페이스 기능에 대한 구현 방법을 기능별로 기술

- 인터페이스 기능 구현 정의 순서



**모듈 세부 설계서**

- 모듈의 구성 요소와 세부적인 동작 등을 정의한 설계서

- 컴포넌트 명세서 : 컴포넌트의 개요 및 내부 클래스의 동작, 인터페이스를 통해 외부와 통신하는 명세 등을 정의

- 인터페이스 명세서 : 컴포넌트 명세서의 항목 중 인터페이스 클래스의 세부 조건 및 기능 등을 정의

**모듈 세부 설계서 확인**

- 모듈의 컴포넌트 명세서와 인터페이스 명세서를 기반으로 인터페이스에 필요한 기능 확인

**인터페이스 기능 구현 정의**

- 인터페이스의 기능, 데이터 표준, 모듈 세부 설계서를 기반으로 일관성 있고 정형화된 인터페이스 기능 구현에 대해 정의

### 인터페이스 구현

**인터페이스 구현**

- 송수신 시스템 간의 데이터 교환 및 처리를 실현해주는 작업

- 정의된 인터페이스 기능 구현을 기반으로 인터페이스 구현 방법을 분석하고 분석한 인터페이스 구현 정의를 기반으로 구현

**데이터 통신을 이용한 인터페이스 구현**

- 애플리케이션 영역에서 인터페이스 형식에 맞춘 데이터 포맷을 인터페이스 대상으로 전송하고 이른 수신 측에서 파싱 하여 해석하는 방식

- JSON, XML 형식 사용

**인터페이스 엔티티를 이용한 인터페이스 구현**

- 인터페이스가 필요한 시스템 사이에 별도의 인터페이스 엔티티를 두어 상호 연계하는 방식

- 인터페이스 테이블 활용

## [2.5 인터페이스 구현(2)](http://127.0.0.1:5503/163)

2020. 4. 16. 01:26

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 인터페이스 예외 처리

**인터페이스 예외 처리의 개요**

- 구현된 인터페이스가 동작하는 과정에서 기능상 예외 상황이 발생했을 때 처리하는 절차

**데이터 통신을 이용한 인터페이스 예외 처리**

- 인터페이스 객체를 이용해 구현한 인터페이스 동작이 실패할 경우를 대비

- 송수신 시 발생할 수 있는 예외 케이스를 정의하고 예외 처리 방법을 기술

- 시스템 환경, 송수신 데이터, 프로그램 자체 원인 등의 원인으로 예외 상황 발생

**인터페이스 엔티티를 이용한 인터페이스 예외 처리**

- 엔티티에 인터페이스의 실패 상황과 원인을 기록

- 조치를 취할 수 있도록 사용자와 관리자에게 알려주는 방식으로 예외 처리

### 인터페이스 보안

**인터페이스 보안의 개요**

- 충분한 보안을 갖추지 않으면 시스템 전체에 악영향을 주는 취약점이 될 수 있음

**인터페이스 보안 취약점 분석**

- 인터페이스 기능이 수행되는 각 구간들의 구현 현황을 확인 후 어떤 취약점이 있는지 확인

- 송수신 영역의 구현 기술 및 특징을 구체적으로 확인

- 확인된 인터페이스 기능을 기반으로 영역별로 발생할 수 있는 취약점을 시나리오 형태로 작성

**인터페이스 보안 기능 적용**

- 분석한 인터페이스 기능과 취약점을 기반으로 보안 기능 적용

- 네트워크 영역 : 송수신간 스니핑 등을 이용한 데이터 탈취 및 변조 위험을 방지하기 위해 네트워크 트래픽에 대한 암호화 설정

- 애플리케이션 영역 : 소프트웨어 개발 보안 가이드를 참조하여 코드 상의 취약점을 보완

- 데이터베이스 영역 : 접근 권한과 데이터베이스 동작 객체의 취약점에 보안 기능 적용

### 연계 테스트

**연계 테스트의 개요**

- 구축된 연계 시스템과 구성 요소가 정상적으로 동작하는지 확인

**연계 테스트 케이스 작성**

- 연계 시스템 간의 데이터 및 프로세스 흐름을 분석하여 필요한 테스트 항목을 도출

- 송수신 연계 응용 프로그램의 단위 테스트 케이스와 연계 테스트 케이스를 각각 작성

**연계 테스트 환경 구축**

- 테스트의 환경을 송수신 기관과의 협의를 통해 결정하고 구축

**연계 테스트 수행**

- 연계 응용 프로그램을 실행하여 연계 테스트 케이스의 시험 항목 및 처리 절차 등을 실제로 진행

**연계 테스트 수행 결과 검증**

- 연계 테스트 케이스의 시험 항목 및 처리 절차를 수행한 결과가 예상 결과와 동일한지 확인

- 연계 서버에서 적용하는 모니터링 현황 확인

- 시스템에서 기록하는 로그 확인

- 테이블 또는 파일을 열어 데이터를 확인

### 인터페이스 구현 검증

**인터페이스 구현 검증의 개요**

- 인터페이스가 정상적으로 잘 작동하는지 확인하는 것

**인터페이스 구현 검증 도구**

- xUnit : Java, C++, .Net 등 다양한 언어를 지원

- STAF : 서비스 호출 및 컴포넌트 재사용 등 다양한 환경을 지원

- FitNesse : 웹 기반 테스트케이스 설계, 진행, 결과 확인 등을 지원

- NTAF : FitNess의 협업 기능과 STAF의 재사용 및 확장성을 통합한 NHN의 프레임워크

- Selenium : 다양한 브라우저 및 개발 언어 지원

- watir : Ruby를 사용

**인터페이스 구현 감시 도구**

- APM을 사용하여 감시 가능

- 애플리케이션 성능 관리 도구를 통해 데이터베이스와 웹 애플리케이션의 다양한 정보를 조회하고 분석할 수 있음

- 스카우터(Scouter), 제니퍼(Jennifer) 등

**인터페이스 구현 검증 도구 및 감시 도구 선택**

- 인터페이스 명세서의 세부 기능을 참조하여 검증 도구와 감시 도구의 요건을 분석

- 분석 후 시장 및 솔루션 조사를 통해 적절한 도구 선택

**인터페이스 구현 검증 확인**

- 외부 시스템과 연계 모듈 동작 상태 확인

- 예상되는 결과값과 실제 검증 값이 동일한지 비교

**인터페이스 구현 감시 확인**

- 외부 시스템과 연결 모듈이 서비스를 제공하는 동안 정상적으로 동작하는지 확인

### 인터페이스 오류 확인 및 처리 보고서 작성

**인터페이스 오류 확인 및 처리 보고서의 개요**

- 인터페이스 오류 발생 시 오류사항을 확인하고 오류 처리 보고서를 작성하여 관리 조직에 보고

**인터페이스 오류 발생 즉시 확인**

- 화면에 오류 메시지를 표시하고 자동으로 SMS나 이메일을 발생하는 것으로 즉시 오류 발생 확인

**주기적인 인터페이스 오류 발생 확인**

- 시스템 로그나 인터페이스 오류 관련 테이블 등을 통해 주기적으로 오류 발생 여부 확인

**인터페이스 오류 처리 보고서 작성**

- 인터페이스 작동 시 발생하는 오류의 발생 및 종료 시점, 원인 및 증상, 처리사항 등을 정리한 문서

- 보고 시기를 최초 발생 시, 오류 처리 경과 시, 완료 시로 나누어 작성

## [2과목 예상 키워드 정리](http://127.0.0.1:5503/185)

2020. 6. 4. 23:13

728x90

**이 포스팅은 주관적으로 중요하다고 생각한 키워드를 정리한 글입니다.**

### 자료구조

* 배열 : 첨자로 접근
* 선형 리스트
  + 연속 리스트 : 배열을 이용, 삽입 / 삭제 시 자료 이동 용이
  + 연결 리스트 : 포인터 이용, 접근 속도 느리고 기억 공간 효율이 좋지 않음
* 스택 : LIFO
* 큐 : FIFO
* 트리
  + 노드와 가지를 이용하여 구성
  + 트리의 디그리 : 노드들 디그리 중 가장 많은 수

### 데이터베이스의 정의

* 통합된 데이터 : 중복 최소화
* 저장된 데이터 : 저장 매체 저장
* 운영 데이터 : 고유한 업무 수행
* 공용 데이터 : 여러 시스템 공동 소유

### DBMS 기능

* 정의 : 타입 및 구조 명시하는 기능 (DDL)
* 조작 : 검색, 갱신, 삭제, 삽입하는 기능 (DML)
* 제어 : 무결성 유지, 보안 유지, 권한 검사, 병행 제어 (DCL)

### 트랜잭션

* 하나의 논리적 기능 수행 작업 단위
* 한꺼번에 수행될 일련의 연산
* TCL : COMMIT, ROLLBACK, SAVEPOINT

### 테스트 케이스

* 테스트 항목에 대한 명세서
* 입력 데이터, 테스트 조건, 예상 결과를 모아 만듬
* 계획/제어 -> 분석/설계 -> 구현/실현 -> 평가 -> 완료

### 빌드 도구

* 소스 코드를 소프트웨어로 변환
* Ant : 자바 빌드
* Maven : Ant 대안 의존성 라이브러리 관리
* Gradle : 안드로이드 스튜디오

### 디지털 저작권 관리

* 클리어링 하우스 : 사용 권한, 라이선스 발급, 사용량에 따른 결제 관리 등 수행
* 콘텐츠 제공자 : 저작권자
* 패키저 : 암호화 프로그램
* 콘텐츠 분배자 : 암호화 콘텐츠 유통
* 콘텐츠 소비자 : 콘텐츠 사용
* DRM 콘트롤러 : 콘텐츠 이용 권한 통제 프로그램
* 보안 컨테이너 : 콘텐츠 원본 전자적 보안 장치

### 소프트웨어 버전 등록

* Import(빈 저장소에 파일 복사) -> checkout(저장소에서 소스 파일 및 버전 관리 파일 받음) -> commit(갱신) -> update(저장소의 최신 버전 동기화) -> diff

### 버전 관리 도구

* 공유 폴더 방식
* 클라이언트 / 서버 방식
  + 서버의 자료를 복사하여 작업 후 서버에 반영
  + 서버에서 버전 관리
  + SVN : trunk 에서 작업 후 추가 작업은 branches 디렉토리 안에 작업 후 trunk와 병합, 리버전
* 분산 저장소 방식
  + 하나의 원격 저장소와 분산된 PC의 로컬 저장소에 함께 저장되어 관리
  + Git : 브랜치를 이용하여 다양한 형태의 테스트, 스냅샷

### 애플리케이션 테스트

* 확인(Validation) : 사용자 입장에서 요구사항을 만족하는지
* 검증(Verification) : 개발자 입장에서 명세서에 맞게 만들어 졌는지
* pareto 법칙 : 애플리케이션의 20%의 코드에서 80% 결함이 발견
* 정적 테스트 : 프로그램 실행 없이 소스코드나 명세서 분석
* 동적 테스트 : 프로그램 실행
* 화이트박스 테스트 : 원시 코드를 오픈하여 모든 경로를 실행하면서 테스트
  + 조건 검사, 루프 검사, 데이터 흐름 검사
  + 문장 검증 기준, 분기 검증 기준, 조건 검증 기준, 분기/조건 검증 기준
* 블랙박스 테스트 : 특정 기능이 작동되는 것을 입증하는 테스트
  + 동등 분할 검사, 경계값 분석, 원인-효과 그래프 검사, 오류 예측 검사, 비교 검사
* 단위 테스트 -> 통합 테스트 -> 시스템 테스트 -> 인수 테스트
* 통합 테스트 : 모듈 간 상호 작용 오류 검사
  + 비점진적 : 미리 결합되어 있는 것을 테스트, 빅뱅 통합
  + 점진적 : 모듈 단위로 통합하면서 테스트, 하향식, 상향식, 혼합식
    - 하향식 : 상위 모듈에서 하위 모듈 방향, 스텁
    - 상향식 : 하위 모듈에서 상위 모듈 방향, 드라이버
    - 혼합식 : 샌드위치 통합 테스트
* 인수 테스트 : 사용자의 요구사항을 만족하는지 테스트
  + 알파테스트 : 개발된 환경에서 개발자 앞에서
  + 베타테스트 : 사용자 환경에서 사용자가 직접

### 테스트 오라클

* 정의된 참 값을 대입하여 비교
* 제한된 검증, 수학적 기법, 자동화 기능
* 참 오라클, 샘플링 오라클, 추정 오라클, 일관성 검사 오라클

### 모듈 연계

* EAI : 기업 내 상호 연동이 가능하게 해주는 솔루션
  + point to point, Hub & Spoke, Message Bus, Hybrid
* ESB : 애플리케이션 간 표준 기반 인터페이스 제공

### 인터페이스 구현 검증 도구

* xUnit : java, C++, .Net
* STAF : 서비스 호출 및 컴포넌트 재사용
* FitNesse : 웹 기반
* NTAF : FitNess + STAF, NHN의 프레임워크
* Selenium : 다양한 브라우저 및 개발 언어
* watir : Ruby

## [3.1 논리 데이터베이스 설계(1)](http://127.0.0.1:5503/89)

2020. 2. 12. 23:27

[개체](http://127.0.0.1:5503/tag/%EA%B0%9C%EC%B2%B4), [관계](http://127.0.0.1:5503/tag/%EA%B4%80%EA%B3%84), [논리데이터베이스](http://127.0.0.1:5503/tag/%EB%85%BC%EB%A6%AC%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4), [속성](http://127.0.0.1:5503/tag/%EC%86%8D%EC%84%B1), [정보처리기사](http://127.0.0.1:5503/tag/%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [정보처리기사필기](http://127.0.0.1:5503/tag/%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [정처기](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0), [정처기필기](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0)

728x90

### 데이터베이스 설계

**데이터베이스 설계의 개념**

- 사용자의 요구를 분석하여 그것에 맞게 설계하고 특정 DBMS로 데이터베이스를 구현하여 사용자들이 사용하는 것

**데이터베이스 설계 순서**

- 요구 조건 분석 → 개념적 설계 → 논리적 설계 → 물리적 설계 → 구현

- 개념적 설계 : 개념 스키마, E-R 모델, 트랜잭션 모델링

- 논리적 설계 : 논리 스키마 설계 /  트랜잭션 인터페이스 설계 / 관계형 DB - Table, 계층형 DB - Tree, 망형 DB - Graph

- 물리적 설계 : 컴퓨터에 저장

**데이터베이스 설계 시 고려사항**

- 무결성(=정확성), 일관성, 회복, 보안, 효율성, 데이터베이스 확장

### 데이터 모델의 개념

**데이터 모델의 정의**

- 현실 세계의 정보들을 컴퓨터에 표현하기 위해서 단순화, 추상화하여 체계적으로 표현한 개념적 모형

**데이터 모델의 구성 요소**

- 개체(Entity), 속성(Attribute), 관계(Relation)

**데이터 모델의 종류**

- 개념적 데이터 모델, 논리적 데이터 모델, 물리적 데이터 모델

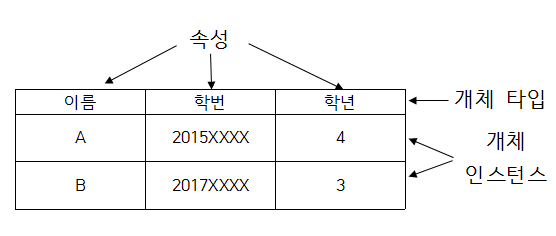
**데이터 모델에 표시할 요소**

- 구조(Structure) : 개체 타입들 간의 관계, 데이터 구조 및 정적 성질 표현

- 연산(Operation) : 저장된 데이터를 처리하는 작업에 대한 명세, DB를 조작하는 기본 도구

- 제약 조건(Constraint) :  데이터의 논리적인 제약 조건

### 데이터 모델의 구성 요소 - 개체, 속성, 관계



**개체의 정의**

- 데이터베이스의 표현하려는 정보

**개체의 특징**

- 유형, 무형의 정보로서 서로 연관된 몇 개의 속성으로 이루어짐

- 유일한 식별자에 의해 식별이 가능

- 개체(튜플)의 수를 카디널리티라고 함

- 개체 인스턴스 : 개체를 구성하고 있는 속성들이 값을 가져 하나의 개체를 나타내는 것. 개체 어커런스라고도 함

- 자료 흐름도(DFD)를 통해 업무 분석을 했을 경우는 자료 저장소(Data Store)를 이용함.

**속성의 정의**

- DB를 구성하는 가장 작은 논리적인 단위

- 파일에서의 데이터 항목 또는 데이터 필드에 해당

**속성의 특징**

- 개체의 특성을 기술함

- 속성의 수를 차수 혹은 디그리(Degree)라고 함

**속성의 종류**

- 기본 속성 : 업무 분석을 통해 정의한 속성. 가장 많고 일반적 ex) 제품명

- 설계 속성 : 업무상 존재하지 않지만 설계 과정에서 도출해내는 속성 ex) 제품 코드

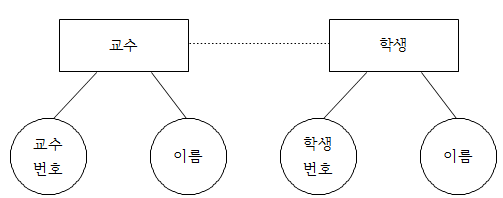
- 파생 속성 : 다른 속성으로부터 계산되거나 파생된 속성 ex) 제품 판매량, 판매 수익

**속성의 분류**

- 기본키 속성(Primary Key Attribute) : 개체를 식별할 수 있는 속성 ex) 학번

- 외래키 속성(Foreign Key Attribute) : 다른 개체와의 관계에서 포함된 속성

- 일반 속성 : 개체에 포함되어 있지만 기본키, 외래키가 아닌 속성

개체 간의 관계는 점선, 속성 간의 관계는 실선으로 표현

**관계의 정의**

- 개체 간의 논리적인 연결

**관계의 형태**

- 1:1, 1:N, N:M 3가지 관계가 있음

**식별 / 비식별 관계**

- 식별 관계 : A,B 개체 간의 관계에서 A 개체의 기본키가 B 개체의 외래키면서 동시에 기본인 것

- 비식별 관계 : A,B 개체 간의 관계에서 A 개체의 기본키가 B 개체의 외래키이지만 기본키는 아닌 것

- 한 개체의 기본키를 다른 개체가 기본키로 사용하면 식별, 아니면 비식별

## [3.1 논리 데이터베이스 설계(2)](http://127.0.0.1:5503/90)

2020. 2. 15. 13:02

[2020정보처리기사](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정처기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0), [E-R모델](http://127.0.0.1:5503/tag/E-R%EB%AA%A8%EB%8D%B8), [관계형데이터베이스](http://127.0.0.1:5503/tag/%EA%B4%80%EA%B3%84%ED%98%95%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4), [식별자](http://127.0.0.1:5503/tag/%EC%8B%9D%EB%B3%84%EC%9E%90)

728x90

### 식별자

**식별자의 정의**

- 하나의 개체 내에서 각각의 인스턴스(구체적인 데이터 값)를 유일하게 구분 지을수 있는 것

**식별자의 분류**

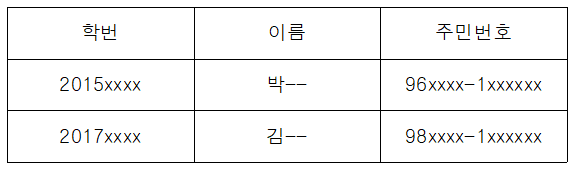
- 대표성 여부 : 개체를 유일하게 식별할 수 있음

    -> 주 식별자

        \* 개체를 대표하는 유일한 식별자 ex) 학번

        \* 주 식별자의 특징 : 유일성, 최소성, 불변성, 존재성

    -> 보조 식별자 : 주 식별자를 대신하여 개체를 식별할 수 있는 것 ex) 주민번호



- 스스로 생성 여부

    -> 내부 식별자 : 개체 내에서 스스로 만들어지는 식별자

    -> 외부 식별자 : 다른 개체와의 관계에서 만들어지는 식별자

- 단일 속성 여부

    -> 단일 식별자 : 주 식별자가 한 가지 속성으로 구성된 식별자

    -> 복합 식별자 : 주 식별자가 두 개 이상의 속성으로 구성된 식별자

- 대체 여부

    -> 원조 식별자(=본질 식별자) : 업무에 의해 만들어지는 가공되지 않은 원래의 식별자

    -> 대리 식별자(=대리 식별자) : 주 식별자의 속성이 두 개 이상인 경우 속성들을 하나의 속성으로 묶어 사용하는 식별자

### E-R(개체-관계) 모델

**E-R 모델의 개요**

- E-R 모델은 개념적 데이터 모델의 가장 대표적인 것

데이터베이스 설계 순서

- 요구 조건 분석 → 개념적 설계 → 논리적 설계 → 물리적 설계 → 구현

- 개념적 설계 : 개념 스키마, E-R 모델, 트랜잭션 모델링

- 피터첸에 의해 제안되어 기본적인 구성 요소가 적립

- 데이터를 개체, 관계, 속성으로 묘사

**E-R 다이어그램**

**-** E-R 모델의 기본 아이디어를 쉽게 기호를 사용하여 시각적으로 표현한 것

- 표기법에는 피터 첸 표기법, 정보 공학 표기법 등이 있다.

**피터 첸 표기법**

- 사각형 : 개체 타입

- 마름모 : 관계 타입

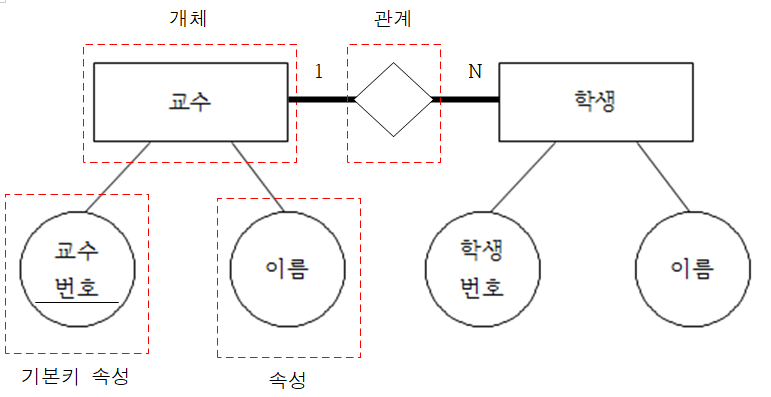
- 타원 : 속성 타입

- 이중 타원 : 복합 속성

- 밑줄 타원 : 기본키 속성

- 복수 타원 : 복합 속성

- 관계 : 1:1, 1:N, N:M 같은 관계에 대한 대응 수

교수와 학생이 1:N 관계

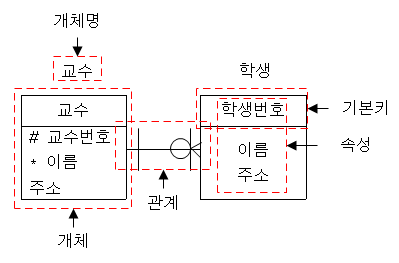
**정보 공학 표기법**

- 관계 표시 기호

    -> | : 1

    -> O : 0

    -> < : N

교수와 학생이 1:N 관계

### 관계형 데이터베이스의 구조

**관계형 데이터베이스**

- 개체, 속성, 관계를 모두 표로 표현

- 간결하고 보기 편하며 다른 데이터베이스로의 변환이 용이

**관계형 데이터베이스의 구조**

- 튜플

    -> 릴레이션을 구성하는 각각의 행

    -> 튜플의 수를 카디널리티(Cardinality)라고 함

- 속성

    -> 데이터베이스를 구성하는 가장 작은 논리적인 단위

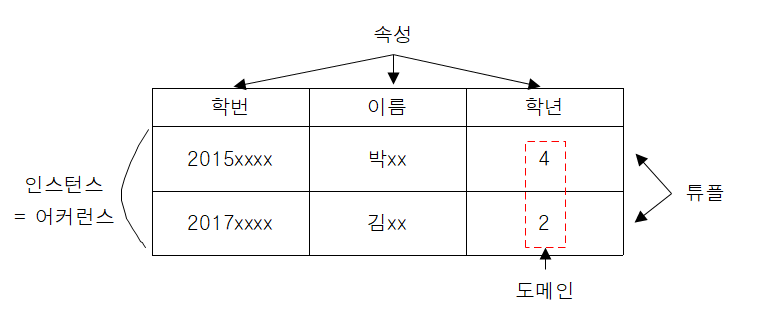
    -> 개체의 특성을 기술

    -> 속성의 수를 디그리(Degree) 또는 차수라고 함

- 도메인

    -> 속성이 취할 수 있는 값의 범위

    -> ex) 학년의 도메인 : 1~4



**릴레이션의 특징**

- 튜플

    -> 똑같은 튜플이 있을 수 없음. 모두 상이하다

    -> 튜플에는 순서가 없음

    -> 튜플의 삽입, 삭제로 인해 시간에 따라 변함

- 속성

    -> 속성의 명칭은 유일해야 하지만 속성을 구성하는 값은 다를 수 있음

    -> 속성에는 순서가 없음

    -> 속성들은 원자 값(Atomic-Value)를 가짐

## [3.1 논리 데이터베이스 설계(3)](http://127.0.0.1:5503/93)

2020. 2. 15. 14:00

[2020정보처리기사](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정처기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0), [KEY](http://127.0.0.1:5503/tag/KEY), [관계대수](http://127.0.0.1:5503/tag/%EA%B4%80%EA%B3%84%EB%8C%80%EC%88%98), [관계해석](http://127.0.0.1:5503/tag/%EA%B4%80%EA%B3%84%ED%95%B4%EC%84%9D), [무결성](http://127.0.0.1:5503/tag/%EB%AC%B4%EA%B2%B0%EC%84%B1)

728x90

### 관계형 데이터베이스의 제약 조건 - Key

**Key의 개념**

- key는 데이터베이스에서 조건에 맞는 튜플을 찾거나 정렬할 때 튜플을 서로 구분할 수 있는 기준이 되는 속성

**Key의 종류**

- 후보키

    -> 기본키로 사용할 수 있는 속성

    -> 유일성과 최소성의 성질을 만족

- 기본키

    ->후보 키 중에서 선정된 Main Key로 중복된 값을 가질 수 없음

    ->후보 키의 부분집합

    -> NULL 값을 가질 수 없음(=개체 무결성)

    -> NULL 값 : 정보의 부재를 나타내기 위해 사용하는 값. 0의 값이 아님

- 대체키

    ->후보 키가 둘 이상일 때 기본키를 제외한 나머지 후보키

- 슈퍼키

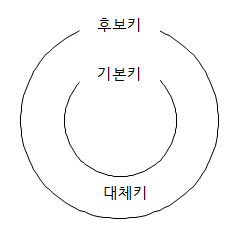
    -> 한 가지 속성일 땐 Key가 될 수 없지만 여러 속성이 뭉쳐서 Key의 속성을 가짐

    -> 유일성의 성질을 만족

- 외래키

    -> 다른 릴레이션의 기본 키를 참조한 것

    ->외래 키의 값은 참조한 릴레이션의 기본키 값과 동일해야 함(=참조 무결성)

후보키, 기본키, 대체키의 관계

### 관계형 데이터베이스의 제약조건 - 무결성

**무결성의 개념**

- 무결성은 데이터베이스의 저장된 데이터 값과 그것이 표현하는 실제 값이 일치하는 정확성을 의미

**무결성의 종류**

- 개체 무결성 : 기본키를 구성하는 어떤 속성도 NULL 값이나 중복 값이면 안됨

- 도메인 무결성 : 속성의 값이 도메인에 속한 값이어야 함

- 참조 무결성 : 외래 키의 값은 참조한 릴레이션의 기본키 값과 동일해야 함

- 사용자 정의 무결성 : 속성 값들은 사용자가 정의한 제약 조건에 만족해야 함

**무결성 강화**

- 애플리케이션

    -> 무결성 조건을 검증하는 코드를 데이터를 조작하는 프로그램 내에 추가

    -> 사용자 정의 같은 복잡한 무결성 조건의 구현이 가능

- 데이터베이스 트리거

    -> 트리거 : 데이터베이스 시스템에 이벤트가 발생할 때마다 자동으로 수행되는 절차형 SQL

- 제약 조건

    -> 데이터베이스 제약 조건을 설정하여 무결성을 유지

### 관계 대수 및 관계 해석

**관계 데이터 언어**

- 관계 대수 : 관계형 데이터베이스에서 원하는 정보와 그 정보를 검색하기 위해 유도하는 것을 기술하는 절차적 언어

- 관계 해석 : 관계 데이터의 연산을 표현하는 비절차적 언어

**관계 대수의 연산자**

- Select

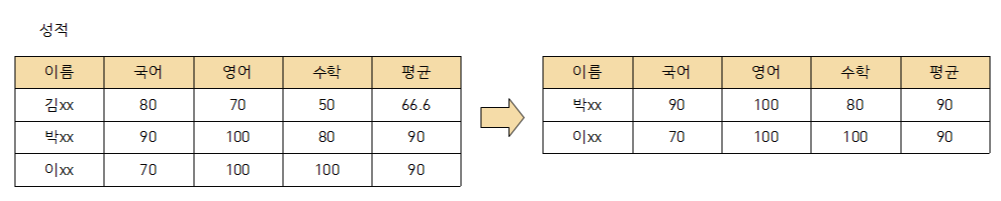
    -> 조건을 만족하는 튜플을 구하여 새로운 릴레이션을 만드는 연산

    -> 수평 연산이라고 함

    -> 연산자의 기호는 시그마(σ)를 사용

    -> 표기 형식 : σ<조건>(R) (여기서 R은 릴레이션을 의미)

    -> ex) σ평균>80 (성적) : 성적 릴레이션에서 평균값이 80 이상인 튜플을 구함



- Project

    -> 속성만 추출하여 새로운 릴레이션을 만드는 연산

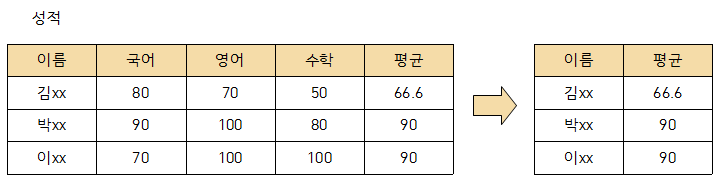
    -> 수직 연산이라고 함

    -> 연산 결과에 중복이 발생하면 제거

    -> 연산자의 기호는 파이(π)를 사용

    -> 표기 형식 : π<조건>(R)

    -> ex) π이름, 평균(성적) : 성적 릴레이션에서 이름, 평균 속성을 추출



- Join

    -> 두 릴레이션을 합쳐서 새로운 릴레이션을 만드는 연산

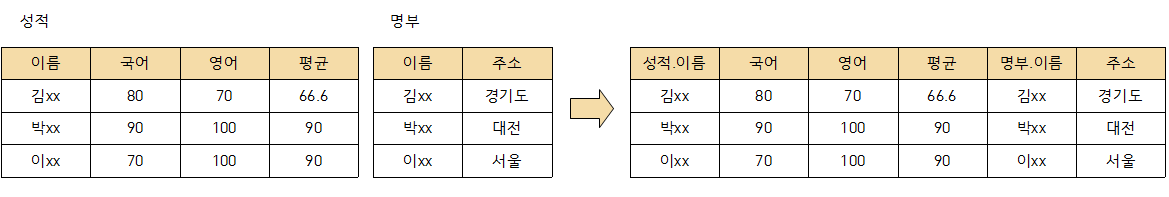
    -> Join의 결과로 만들어진 릴레이션의 차수는 두 차수의 합

    -> Join의 결과는 교차곱(Cartesian Product)을 수행 후 Select 한 것과 같음

    -> 연산자의 기호는 ⋈를 사용

    -> 표기 형식 : R⋈키 속성 r=키 속성 sS

    -> ex) 성적⋈이름=이름 명부 : 성적 릴레이션과 명부 릴레이션을 이름 속성을 기준으로 합침



    -> 자연 조인 : Join의 조건이 같을 때 동일한 속성이 두 번 나타나 중복된 속성을 제거하여 한번만 표기하는 방법

    -> 자연 조인이 성립되려면 두 릴레이션의 속성명과 도메인이 같아야 함

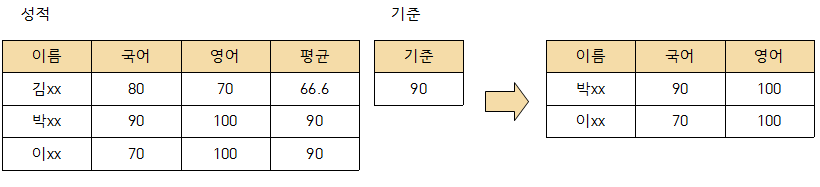
- Division

    -> R ⊃ S인 두 릴레이션이 있을 때, R의 속성이 S의 속성의 값을 모두 가진 튜플에서 S가 가진 속성을 제외한 속성만을 구하는 연산

    -> 연산자의 기호는 ÷를 사용

    -> 표기 형식 : R [속성 r ÷ 속성 s] S

    -> ex) 성적 [평균 ÷ 기준] 기준



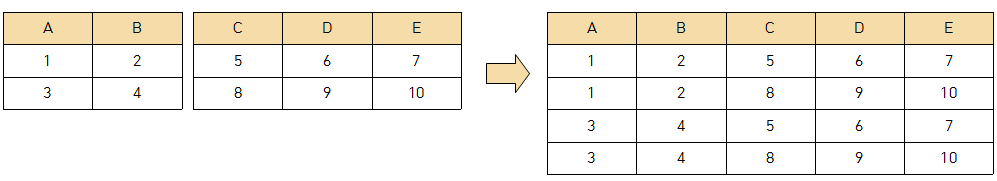
- 일반 집합 연산자

    -> 합집합(∪) : 두 릴레이션의 튜플의 합집합. 중복되는 튜플은 제거

    -> 교집합(∩) : 두 필레이션의 튜플의 교집합.

    -> 차집합(-) : 두 릴레이션의 튜플의 차집합.

    -> 교차곱(×) : 두 릴레이션의 튜플들의 순서쌍. 차수(Degree)는 서로 더한 값 카디널리티(Cardinality)는 서로 곱한 값

교차곱

## [3.1 논리 데이터베이스 설계(4)](http://127.0.0.1:5503/94)

2020. 2. 18. 16:28

[2020정보처리기사](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정처기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0), [반정규화](http://127.0.0.1:5503/tag/%EB%B0%98%EC%A0%95%EA%B7%9C%ED%99%94), [시스템카탈로그](http://127.0.0.1:5503/tag/%EC%8B%9C%EC%8A%A4%ED%85%9C%EC%B9%B4%ED%83%88%EB%A1%9C%EA%B7%B8), [정규화](http://127.0.0.1:5503/tag/%EC%A0%95%EA%B7%9C%ED%99%94), [정처기필기](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0)

728x90

### 정규화(Normalization)

**정규화의 개념**

- 관계형 데이터베이스에서 정확성을 더욱 유지하기 위해 스키마를 쪼개는 과정

- 데이터베이스의 논리적 설계 단계에서 수행

**정규화의 목적**

- 데이터 구조의 안정성 및 무결성을 유지

- 이상의 발생을 방지 및 자료 저장 공간의 최소화

**이상(Anomaly)**

- 사용자의 의도와는 상관없이 데이터가 삽입, 삭제, 갱신되는 현상

**정규화 과정**

- 1NF -> 2NF -> 3NF -> BCNF -> 4NF -> 5NF

- 1NF(제1 정규형)

    -> 릴레이션에 속한 모든 값들이 원자 값으로만 구성

- 2NF(제2 정규형)

    -> 기본키가 아닌 모든 속성이 기본키에 대하여 완전 함수적 종속을 만족

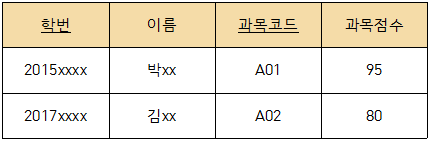
    -> 완전 함수적 종속 : 기본키에 의해서 속성이 결정

    -> 부분 함수적 종속 : 기본키의 일부에 의해 속성이 결정

    -> 아래와 같이 학번과 과목 코드가 기본키인 릴레이션이 있을 때

    -> 과목 점수는 기본키(학번, 과목 코드)를 가지고 알 수 있음 = 완전 함수적 종속

    -> 이름은 기본키의 일부(학번)를 가지고 알수 있음 = 부분 함수적 종속



- 3NF(제3 정규형)

    -> 기본키가 아닌 모든 속성이 기본키에 대해 이행적 종속을 만족하지 않음

    -> 이행적 종속 : A → B, B → C 일 때 A → C를 만족하는 관계

- BCNF(Boyce-Codd정규형)

    -> 결정자가 모두 후보키

- 4NF(제4 정규형)

    -> 릴레이션에 다치 종속이 성립하는 경우 모든 속성이 함수적 종속 관계를 만족

- 5NF(제5 정규형)

    -> 모든 조인 종속이 후보키를 통해서만 성립

정규화 과정

### 반정규화(Denormalization)

**반정규화의 개념**

- 정규화된 데이터를 다시 통합, 중복, 분리하는 과정으로 의도적으로 정규화 원칙을 위해

- 과도한 정규화로 성능이 떨어졌을 때 실행

**반정규화의 종류**

- 테이블 통합

    -> 하나의 테이블로 합쳐 사용하는 것이 성능 향상에 도움이 될 경우 수행

    -> Not NULL, Default, Check 등의 제약조건을 설계하기 어려움

- 테이블 분할

    -> 테이블을 수평 또는 수직으로 분할

- 중복 테이블 추가

    -> 여러 테이블에서 데이터를 추출해서 사용해야 하거나 다른 서버에 저장된 테이블을 이용해야 하는 경우 수행

- 중복 속성 추가

    -> 조인해서 데이터를 처리할 때 데이터를 조회하는 경로를 단축하기 위해 자주 사용하는 속성을 하나 더 추가

### 시스템 카탈로그

**시스템 카탈로그의 의미**

- 시스템 그 자체에 관련이 있는 다양한 객체에 관한 정보를 포함하는 시스템 데이터베이스

- 데이터 사전(Data Dictionary)이라고도 함

**시스템 카탈로그 저장 정보**

- 시스템 카탈로그에 저장되는 정보를 메타 데이터라고 함

- 메타 데이터 : 데이터에 대한 설명. 메타 데이터가 모이면 데이터 사전이 됨

- 메타 데이터의 유형

    -> 데이터 베이스 객체 정보 : Table, Index, View 등의 구조 및 통계 정보

    -> 사용자 정보

    -> 테이블의 무결성 제약 조건 정보

    -> 함수, 프로시저, 트리거 등에 대한 정보

**시스템 카탈로그의 특징**

- 시스템 테이블로 구성되어 있어 SQL문으로 검색해 볼 수 있음

- DML(INSERT, DELETE, UPDATE) 문으로 갱신이 불가능

## [3.2 물리 데이터베이스 설계(1)](http://127.0.0.1:5503/100)

2020. 2. 25. 16:54

[2020정보처리기사](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정처기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0), [crud](http://127.0.0.1:5503/tag/crud), [물리데이터베이스설계](http://127.0.0.1:5503/tag/%EB%AC%BC%EB%A6%AC%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4%EC%84%A4%EA%B3%84), [정처기필기](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0), [트랜잭션](http://127.0.0.1:5503/tag/%ED%8A%B8%EB%9E%9C%EC%9E%AD%EC%85%98)

728x90

### 사전 조사 분석

**물리 데이터베이스 설계**

- 논리적 구조로 표현된 논리적 데이터베이스를 디스크 등의 물리적 저장장치에 저장할 수 있는 물리적 구조의 데이터로 변화하는 과정

데이터베이스 설계 순서

- 요구 조건 분석 → 개념적 설계 → 논리적 설계 → 물리적 설계 → 구현

- 물리적 설계 단계에서 수행해야 할 것

    -> 저장 레코드의 양식 설계

    -> 레코드 집중의 분석 및 설계

    -> 접근 경로 설계

- 물리적 데이터베이스 구조는 여러 가지 타입의 저장 레코드 집합이라는 면에서 단순한 파일과 다름

- 물리적 설계 옵션

    -> 반응시간 : 트랜잭션 수행을 요구한 시점부터 처리 결과를 얻을 때까지의 경과 시간

    -> 공간 활용도 : 데이터베이스 파일과 액세스 경로 구조에 의해 사용되는 저장공간의 양

    -> 트랜잭션 처리량 : 단위시간 동안 데이터베이스 시스템에 의해 처리될 수 있는 트랜잭션의 평균 개수

**데이터 명명 규칙 파악**

- 물리 데이터 모델에 적용해야하는 규칙

- 데이터 표준화 및 논리 데이터베이스 설계의 결과물들을 통해 파악

- 물리-논리 데이터베이스 설계 적용되는 명명규칙은 일관성 유지

- 명명 규칙을 파악하기 위해선 도메인과 데이터 사전에 대한 지식 필요

**시스템 자원 파악**

- 데이터베이스 설치에 영향을 미칠 수 있는 물리적인 요소

- 하드웨어 자원, 운영체제 및 DBMS의 버전, DMBS 파라미터 정보 등으로 구분

**데이터베이스 관리 요소 파악**

- 데이터베이스 운영과 관련된 관리요소

- 데이터베이스 관리 요소를 파악한 후 이를 기반으로 시스템 조사 분석서를 작성

- 시스템 조사 분석서를 기반으로 다음과 같은 요소들을 파악

    -> 데이터베이스 구조 : 데이터베이스 구조에 따라 문제 발생 시 대응 방법이 다름

    -> 이중화 구성 : 문제 발생에 대비하여 동일한 데이터베이스를 복제하여 관리

    -> 분산 데이터베이스 : 물리적인 피해에 데이터 유실을 최소화할 수 있고 장애로 인한 데이터 유실 복구에 효과적

    -> 접근 제어 / 통제 : 접근 가능한 사용자의 권한 남용으로 인한 정보 유출 및 변조가 빈번하게 발생

    -> DB암호화 : 데이터 암호화, 암호 키에 대한 인증 등을 통해 데이터 유출 시 데이터의 복호화를 어렵게 함

### 데이터베이스 저장 공간 설계

**테이블**

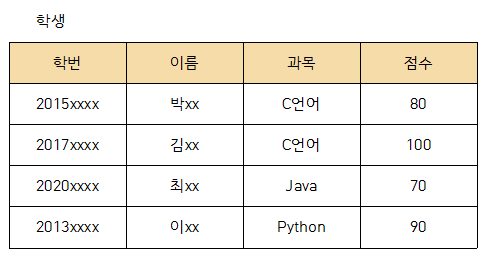
- 데이터베이스의 가장 기본적인 객체로써 행(Row)과 열(Column)으로 구성됨

- 논리 설계 단계의 개체(Entity)에 대응하는 객체

- 종류에는 일반 테이블, 클러스터 인덱스 테이블, 파티셔닝 테이블, 외부 테이블, 임시 테이블 등이 있음

**일반 테이블**

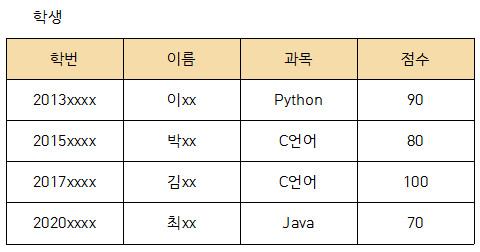
- 현재 사용되는 대부분의 DBMS(DataBase Management System)에서 표준 테이블로 사용되는 테이블



**클러스터드 인덱스 테이블**

- 기본키나 인덱스키의 순서에 따라 데이터가 저장되는 테이블

- 일반적인 인덱스를 사용하는 테이블에 비해 접근 경로가 단축

기본키인 학번을 기준으로 정렬

**파티셔닝 테이블**

- 대용량의 테이블을 작은 논리적인 단위인 파티션으로 나눈 테이블

- 대용량의 데이터를 효과적으로 관리할 수 있지만 파티션 키를 잘못 구성하면 성능 저하 등 역효과를 초래할 수 있음

- 방식에 따라 범위 분할, 해시 분할, 조합 분할 등으로 나뉨

    -> 범위 분할 : 지정한 열의 값을 기준으로 분할

    -> 해시 분할 : 해시 함수를 적용한 결과 값에 따라 데이터를 분할

    -> 조합 분할 : 범위 분할로 분할한 다음 해시 함수를 적용하여 다시 분할



**외부 테이블**

- 데이터베이스에서 일반 테이블처럼 이용할 수 있는 외부 파일로 데이터베이스 내에 객체로 존재

- 데이터 쉐어하우스, ETL(Extraction Transformation Loading, 추출 변환 적재) 등의 작업에 사용

**임시 테이블**

- 트랜잭션이나 세션별로 데이터를 저장하고 처리할 수 있는 테이블

- 절차적인 처리를 위해 임시로 사용

- 저장된 데이터는 트랜잭션이 종료되면 삭제

**칼럼**

- 테이블의 열을 구성하는 요소로 데이터 타입과 길이 등으로 정의

- 데이터 타입은 데이터의 일관성 유지를 위해 사용되는 가장 기본적인 것

- 도메인을 정의한 경우 도메인에 따라 데이터의 타입과 길이가 정의

- 두 칼럼을 비교하는 연산에서 두 칼럼의 데이터 타입이나 길이가 다르면 DBMS 내부적으로 데이터 타입 변화 후 비교 연산 수행

- 참조 관계인 칼럼들은 데이터 타입과 길이가 일치해야 함

**테이블스페이스**

- 테이블이 저장되는 논리적인 영역으로 하나의 테이블스페이스에 하나 또는 그 이상의 테이블을 저장할 수 있음

- 테이블을 저장하면 논리적으론 테이블스페이스에 물리적으론 연관된 데이터 파일에 저장됨

- 테이블스페이스를 테이블, 테이블스페이스, 데이터 파일로 나눠 관리하면 논리적 구성이 물리적 구성에 종속되지 않아 투명성이 보장됨

    -> 투명성 : 사실의 존재 여부를 염두에 두지 않아도 되는 성질

- 데이터베이스에 저장되는 내용에 따라 테이블, 인덱스, 임시 등의 용도로 구분하여 설계

### 트랜잭션 및 CRUD 분석

**트랜잭션의 정의**

- 데이터베이스의 상태를 변환시키는 하나의 논리적인 기능을 수행하기 위한 작업 단위 또는 한꺼번에 수행되어야 할 일련의 연산들을 의미

- 데이터베이스 시스템에서 병행 제어 및 회복 작업 시 처리되는 작업의 논리적인 단위

- 사용자가 시스템에 대한 서비스 요구 시 시스템이 응답하기 위한 상태 변환 과정의 작업 단위

**트랜잭션의 특성**

- 원자성(Atomicity)

    -> all or nothing

    -> 데이터베이스에 반영되도록 완전히 완료 아니면 전혀 반영되지 않도록 복구되어야 함

    -> 어느 하나라도 오류가 발생하면 트랜잭션 전부가 취소되어야 함

- 일관성(Consistency)

    -> 문법을 일괄적으로 맞춰야 함

    -> 트랜잭션을 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 변환

    -> 시스템이 가지고 있는 고정 요소는 트랜잭션 수행 전과 후가 상태가 같아야 함

- 독립성(Isolation)

    -> 하나의 트랜잭션 연산 중에는 다른 트랜잭션이 관여하면 안 됨

    ->수행 중인 트랜잭션은 완전히 완료될 때까지 다른 트랜잭션에서 수행 결과를 참조할 수 없음

- 지속성(Durability)

    -> 성공적으로 완료된 트랜잭션의 결과는 영구적으로 유지, 반영되어야 함

**CRUD 분석**

- 데이터베이스의 테이블에 변화를 주는 트랜잭션 연산 중 생성(Create), 읽기(Read), 갱신(Update), 삭제(Delete)의 연산에 대해 CRUD 매트릭스를 작성하여 분석하는 것

- 테이블에 발생되는 트랜잭션의 주기별 발생 횟수를 파악하고 연관된 테이블들을 분석하면 테이블에 저장되는 데이터의 양을 유추할 수 있음

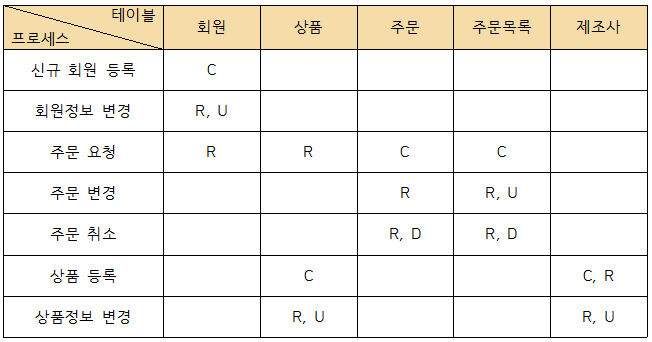
**CRUD 매트릭스**

- 2차원 표로 행에는 프로세스 열에는 테이블을 행과 열이 만나는 위치에는 프로세스가 테이블에 발생시키는 변화를 표시하는 업무 프로세스와 데이터 간 상관 분석표

- CRUD 매트릭스를 통해 프로세스의 트랜잭션이 테이블에 수행하는 작업을 검증

- CRUD 매트릭스의 각 셀에는 C, R, U, D가 들어가고 복수의 작업 시에는 우선순위는 C > D > U > R를 적용

- CRUD 매트릭스가 완성되면 C, R, U, D 중 어느 것도 적히지 않는 행이나 열, C나 R이 없는 행을 확인하여 불필요하거나 누락된 테이블 또는 프로세스를 찾음

온라인 쇼핑몰 CRUD 매트릭스

**트랜잭션 분석**

- CRUD 매트릭스를 기반으로 테이블에 발생하는 트랜잭션 양을 분석하고 테이블에 저장되는 데이터의 양을 유추하고 이를 근거로 DB용량을 산정, DB 구조를 최적화하는 것

**트랜잭션 분석서**

- 단위 프로세스와 CRUD 매트릭스를 이용하여 작성

- 구성 요소에는 단위 프로세스, CRUD 연산, 테이블 명, 칼럼명, 테이블 참조 횟수, 트랜잭션 수, 발생 주기 등

주문요청 프로세스에 대한 트랜잭션 분석서

## [3.2 물리 데이터베이스 설계(2)](http://127.0.0.1:5503/101)

2020. 2. 26. 21:34

[2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정처기필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0), [뷰](http://127.0.0.1:5503/tag/%EB%B7%B0), [인덱스](http://127.0.0.1:5503/tag/%EC%9D%B8%EB%8D%B1%EC%8A%A4), [정보처리기사필기정리](http://127.0.0.1:5503/tag/%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [정처기필기정리](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [클러스터](http://127.0.0.1:5503/tag/%ED%81%B4%EB%9F%AC%EC%8A%A4%ED%84%B0), [파티션](http://127.0.0.1:5503/tag/%ED%8C%8C%ED%8B%B0%EC%85%98)

728x90

### 인덱스 설계

**인덱스의 개념**

- 데이터 레코드를 빠르게 접근하기 위해 키값, 포인터 쌍으로 구성되는 데이터 구조

- 책의 목차와 유사

- 데이터가 저장된 물리적 구조와 밀접한 관계가 있음

- 파일의 레코드에 대한 액세스를 빠르게 수행할 수 있음

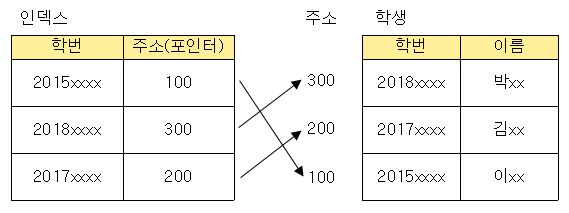
- 인덱스가 없으면 특정한 값을 찾기 위해 모든 데이터 페이지를 확인하는 TABLE SCAN이 발생

    -> TABLE SCAN : 데이터가 나올 때까지 모든 레코드를 순차적으로 읽는 것

- 레코드의 삽입과 삭제가 수시로 일어나는 경우에는 인덱스의 개수를 최소로 하는 것이 효율적

- 클러스터드 인덱스 : 인덱스 키의 순서에 따라 데이터가 정렬되어 저장되는 방식

- 넌클러스터드 인덱스 : 인덱스의 키 값만 정렬되어 있을 뿐 실제 데이터는 정렬되지 않는 방식



**트리 기반 인덱스**

- 인덱스를 저장하는 블록들이 트리 구조를 이루고 있는 것으로 상용 DBMS에서는 트리 구조 기반의 B+ 트리 인덱스를 주로 활용

- B 트리 인덱스

    -> 일반적으로 사용하는 인덱스 방식

    -> 루트 노드에서 하위 노드로 키값의 크기를 비교하면서 데이터를 검색

    -> 모든 리프 노드의 레벨은 같음

- B+ 트리 인덱스

    -> 단말 노드가 아닌 노드로 구성된 인덱스 세트와 단말 노드로만 구성된 순차 세트로 구분

    -> 인덱스 세트에 있는 노드들은 단말 노드에 있는 키 값을 찾아갈 수 있는 경로로만 제공

    -> 순차 세트에 있는 단말 노드가 해당 데이터 레코드의 주소를 가리킴

    -> 인덱스 세트에 있는 모든 키 값이 단말 노드에 다시 나타나므로 단말 노드 만을 이용한 순차 처리 가능

**비트맵 인덱스**

- 인덱스 칼럼의 데이터를 Bit 값인 0 또는 1로 변환하여 인덱스 키로 사용하는 방법

- 키 값을 포함하는 로우(Row)의 주소를 제공

- 데이터가 Bit로 구성되어 있어 효율적인 논리 연산이 가능하고 저장공간이 작음

**함수 기반 인덱스**

- 칼럼의 값 대신 칼럼의 특정 함수나 수식을 적용하여 산출된 값을 사용

- B+ 트리 인덱스 또는 비트맵 인덱스를 생성하여 사용

- 데이터를 입력하거나 수정할 때 함수를 적용하기 때문에 부하가 발생할 수 있음

- 사용자 정의 함수를 사용했을 경우 시스템 함수보다 부하가 더 크다

- 대소문자, 띄어쓰기 등에 상관없이 조회할 때 유용하게 사용

**비트맵 조인 인덱스**

- 다수의 조인된 객체로 구성된 인덱스

**도메인 인덱스**

- 개발자가 필요한 인덱스를 직접 만들어 사용하는 것으로 확장형 인덱스라고도 함

**인덱스 설계 순서**

- 인덱스의 대상 테이블이나 칼럼 등을 선정 → 인덱스의 효율성을 검토하여 인덱스 최적화 수행 →  인덱스 정의서 작성

**인덱스 테이블 선정 기준**

- MULTI BLOCK READ 수에 따라 판단

    -> MULTI BLOCK READ : 테이블 액세스 시 메모리에 한 번에 읽어 들일 수 있는 블록의 수

- 랜덤 액세스가 빈번한 테이블

- 특정 범위나 특정 순서로 데이터 조회가 필요한 테이블

- 다른 테이블과 순차적 조인이 발생되는 테이블

**인덱스 서계 시 고려사항**

- 새로 추가되는 인덱스는 기존 액세스 경로에 영향을 미칠 수 있음

- 인덱스를 지나치게 만들면 오버헤드 발생

- 넓은 범위를 인덱스로 처리하면 많은 오버헤드 발생

- 인덱스를 만들면 추가적인 저장공간 필요

- 인덱스와 테이블 데이터의 저장 공간이 분리되도록 설계

### 뷰 설계

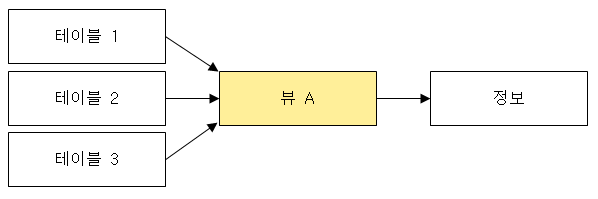
**뷰의 개요**

- 사용자에게 접근이 허용된 자료만을 제한적으로 보여주기 위해 하나 이상의 기본 테이블로부터 유도된 이름을 가지는 가상 테이블

- 물리적으로 존재하지는 않지만 사용자에게는 있는 것처럼 간주됨

- 데이터 보정 작업, 처리 과정 시험 등 임시적인 작업을 위한 용도로 활용

- 조인문의 최소화로 사용자 편의성을 최대화함



**뷰의 특징**

- 기본 테이블과 같은 형태의 구조를 사용하고 조작도 기본 테이블과 거의 같음

- 가상테이블이기 때문에 물리적으로 구현되어 있지 않음

- 데이터의 논리적 독립성을 제공할 수 있음

- 필요한 데이터만 뷰로 정의해서 처리할 수 있기 때문에 관리가 용이하고 명령문이 간단해짐

- 뷰를 통해서만 데이터에 접근하게 되면 뷰에 나타나지 않는 데이터를 안전하게 보호하는 효율적인 기법으로 사용할 수 있음

- 뷰가 정의된 기본 테이블이나 뷰를 삭제 시 그 테이블이나 뷰를 기초로 정의된 다른 뷰도 자동으로 삭제

**뷰의 장단점**

- 장점

    -> 논리적 데이터 독립성 제공

    -> 동일 데이터에 대해 동시에 여러 사용자의 상이한 요구를 지원

    -> 사용자의 데이터 관리가 용이

    -> 접근 제어를 통한 자동 보안 제공

- 단점

    -> 독립적인 인덱스를 가질 수 없음

    -> 뷰의 정의 변경 불가

    -> 뷰로 구성된 내용에 대해 INSERT, DELETE, UPDATE 연산에 제약이 따름

**뷰 설계 순서**

- 대상 테이블 선정 → 대상 칼럼 선정 → 정의서 작성

**뷰 설계 시 고려사항**

- 테이블 구조가 단순화될 수 있도록 반복적으로 조인을 설정하여 사용하거나 동일한 조건절을 사용하는 테이블을 뷰로 생성

- 동일한 테이블이라도 업무에 따라 테이블을 이용하는 부분이 달라질 수 있으므로 사용할 데이터를 다양한 관점에서 제시

- 데이터의 보안을 유지하며 설계

### 클러스터 설계

**클러스터의 개요**

- 데이터 저장 시 데이터 액세스 효율을 향상시키기 위해 동일한 성격의 데이터를 데이터 블록에 저장하는 물리적 저장 방법

- 클러스터링키로 지정된 칼럼 값의 순서대로 저장되고 여러 개의 테이블이 하나의 클러스터에 저장

**클러스터의 특징**

- 데이터 조회 속도는 향상시키지만 데이터 입력 수정 삭제에 대한 성능은 저하시킴

- 데이터의 분포도가 넓을수록 유리

- 대용량을 처리하는 트랜잭션은 전체 테이블을 스캔하는 일이 자주 발생하므로 클러스터링을 지양

- 파티셔닝 된 테이블에는 적용할 수 없음

**클러스터 대상 테이블**

- 분포도가 넓은 테이블

- 대량의 범위를 자주 조회하는 테이블

- 입력, 수정, 삭제가 자주 발생하지 않는 테이블

- 자주 조인되어 사용되는 테이블

- ORDER BY, GROUP BY, UNION이 빈번한 테이블

DML - SELECT

ORDER BY : 특정 속성을 기준으로 정렬하여 검색

GROUP BY : 특정 속성을 기준으로 그룹화하여 검색할 때 사용. 그룹 함수와 같이 사용

UNION : 두 SELECT 문의 결과를 통합하고 중복된 행은 한 번만 출력

### 파티션 설계

**파티션의 개요**

- 대용량의 테이블이나 인덱스를 작은 논리적 단위인 파티션으로 나누는 것

- 대용량 DB의 경우 테이블들을 작은 단위로 나눠 분산시키면 성능 저하를 방지하고 데이터 관리가 용이함

- 데이터 처리는 테이블 단위, 데이터 저장은 파티션 별로 수행

**파티션의 장단점**

- 장점

    -> 데이터 접근 시 액세스 범위를 줄여 쿼리 성능 향상

    -> 데이터가 분산되어 저장되므로 디스크 성능 향상

    -> 파티션별로 백업 및 복구를 수행하므로 속도 향상

    -> 시스템 장애 시 데이터 손상 정도 최소화

    -> 데이터 가용성 향상

    -> 파티션 단위로 입출력 분산

- 단점

    -> 하나의 테이블을 세분화하여 관리하기 때문에 세심한 관리가 요구됨

    -> 테이블 간 조인에 대한 비용이 증가

    -> 용량이 작은 테이블에 파티셔닝을 수행하면 성능이 저하됨

**파티션의 종류**

- 범위 분할 : 지정한 열의 값을 기준으로 분할

- 해시 분할 : 해시 함수를 적용한 결과 값에 따라 데이터를 분할

- 조합 분할 : 범위 분할로 분할한 다음 해시 함수를 적용하여 다시 분할

**파티션 키 선정 시 고려사항**

- 파티션 키는 테이블 접근 유형에 따라 파티셔닝이 이루어지도록 선정

- 데이터 관리의 용이성을 위해 이력성 데이터는 파티션 생성 주기와 소멸 주기를 일치시켜야 함

- 매일 생성되는 날짜 칼럼, 백업의 기준이 되는 날짜 칼럼, 파티션 간 이동이 없는 칼럼, I/O 병목을 줄일 수 있는 데이터 분포가 양호한 칼럼 등을 파티션 키로 선정

**인덱스 파티션**

- 파티션 된 테이블의 데이터를 관리하기 위해 인덱스를 나눈 것

- 파티션된 테이블의 종속 여부에 따른 구분

    -> Local Partitioned Index : 테이블 파티션과 인덱스 파티션이 1:1 대응되도록 파티셔닝 함

    -> Global Partitioned Index : 테이블 파티션과 인덱스 파티션이 독립적으로 구성되도록 파티셔닝 함

    -> Local Partitioned Index가 Global 보다 관리하기 용이함

- 인덱스 파티션 키 칼럼의 위치에 따른 구분

    -> Prefixed Partitioned Index : 인덱스 파티션 키와 인덱스 첫 번째 칼럼이 같음

    -> Non-Prefixed Partitioned Index : 인덱스 파티션키와 인덱스 첫 번째 칼럼이 다름

## [3.2 물리 데이터베이스 설계(3)](http://127.0.0.1:5503/102)

2020. 2. 27. 00:46

[2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [2020정처기필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0), [데이터베이스용량설계](http://127.0.0.1:5503/tag/%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4%EC%9A%A9%EB%9F%89%EC%84%A4%EA%B3%84), [데이터베이스이중화](http://127.0.0.1:5503/tag/%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4%EC%9D%B4%EC%A4%91%ED%99%94), [분산데이터베이스](http://127.0.0.1:5503/tag/%EB%B6%84%EC%82%B0%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4), [정보처리기사필기정리](http://127.0.0.1:5503/tag/%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [정처기필기정리](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [클러스터링](http://127.0.0.1:5503/tag/%ED%81%B4%EB%9F%AC%EC%8A%A4%ED%84%B0%EB%A7%81)

728x90

### 데이터베이스 용량 설계

**데이터베이스 용량 설계**

- 데이터가 테이블에 저장될 공간을 정의

- 테이블에 저장할 데이터양, 인덱스, 클러스터 등이 차지하는 공간 등을 예측하여 반영

**데이터베이스 용량 설계의 목적**

- 디스크의 저장공간을 효과적으로 사용하고 확장성 및 가용성을 높임

- 디스크의 입출력 부하를 분산시키고 채널의 병목현상 최소화

- 디스크에 대한 입출력 경합이 최소화되도록 설계함으로써 데이터 접근성이 향상

- 데이터베이스에 생성되는 오브젝트의 익스텐트 발생을 최소화하여 성능을 향상

- 테이블과 인덱스에 적합한 저장 옵션을 지정

- 데이터 접근성을 향상시키는 설계 방법

    -> 테이블의 테이블스페이스와 인덱스의 테이블스페이스를 분리하여 구성

    -> 테이블스페이스와 임시 테이블스페이스를 분리하여 구성

    -> 테이블을 마스터 테이블과 트랜잭션 테이블로 분류

**데이터베이스 용량 분석 절차**

- 데이터 예상 건수, 로우 길이, 보존 기간, 증가율 등 기초 자료를 수집하여 용량 분석

- 분석된 자료를 바탕으로 DBMS에 이용될 테이블, 인덱스 등 오브젝트별 용량을 산정

- 테이블과 인덱스의 테이블스페이스 용량을 산정

- 데이터베이스에 저장될 모든 데이터 용량과 데이터베이스 설치 및 관리를 위한 시스템 용량을 합해 디스크 용량 산정

### 분산 데이터베이스 설계

**분산 데이터베이스 정의**

- 논리적인 하나의 시스템이지만 물리적으로는 네트워크로 연결된 여러 개의 컴퓨터 사이트에 분산되어 있는 데이터베이스

- 데이터베이스를 네트워크를 이용해 나눠놓음

**분산 데이터베이스의 구성 요소**

- 분산 처리기 : 지리적으로 분산되어 있는 컴퓨터 시스템

- 분산 데이터베이스 : 지리적으로 분산되어 있는 데이터베이스

- 통신 네트워크 : 분산 처리기들을 네트워크로 연결하여 하나의 시스템처럼 동작할 수 있도록 하는 통신 네트워크

**분산 데이터베이스 설계 시 고려사항**

- 작업 부하의 노드별 분산 정책

- 지역의 자치성 보장 정책

- 데이터의 일관성 정책

- 사이트나 회선의 고장으로부터의 회복 기능

- 통신 네트워크를 통한 원격 접속 기능

**분산 데이터베이스의 목표**

- 위치 투명성 : 접근하려는 데이터베이스의 실제 위치를 알 필요 없이 논리적인 명칭만으로 접근 가능

- 중복 투명성 : 동일 데이터가 여러 곳에 중복되어 있어도 사용자는 하나의 데이터만 존재하는 것처럼 사용 가능

- 병행 투명성 : 분산 데이터베이스와 관련된 다수의 트랜잭션들이 동시에 실현되더라도 그 트랜잭션의 결과는 영향을 받지 않음

- 장애 투명성 : 트랜잭션, DBMS, 네트워크, 컴퓨터 등 장애에도 트랜잭션을 정확하게 처리

- 투명성 : 사실 존재 여부를 염두에 두지 않아도 되는 성질

**분산 데이터베이스의 장점**

- 장점

    -> 지역 자치성이 높음

    -> 자료의 공유성 향상

    -> 분산 제어 가능

    -> 시스템 성능 향상

    -> 중앙 컴퓨터의 장애가 전체 시스템에 영향을 끼치지 않음

    -> 효용성과 융통성이 높음

    -> 신뢰성 및 가용성이 좋음

    -> 점진적 시스템 용량 확장 용이

- 단점

    -> DBMS가 수행할 기능이 복잡

    -> DB 설계가 어려움

    -> 소프트웨어 개발 비용 증가

    -> 처리 비용 증가

    -> 잠재적 오류 증가

**분산 데이터베이스 설계**

- 애플리케이션이나 사용자가 분산되어 저장된 데이터에 접근하는 것이 목적

- 전역 관계망을 논리적 측면에서 소규모 단위로 분할, 분할된 결과를 복수의 노드에 할당

- 분산 설계 방법에는 테이블 위치 분산, 분할, 할당이 있음

**테이블 위치 분산**

- 데이터베이스의 테이블을 각기 다른 서버에 분산시켜 배치

- 테이블의 구조를 변경시키지 않고 다른 데이터베이스의 테이블과 중복되지 않게 배치

**분할**

- 테이블의 데이터를 분할하여 분산

- 분할 규칙

    -> 완전성 : 전체 데이터를 대상으로 분할해야 함

    -> 재구성 : 분할된 데이터는 관계 연산을 활용하여 본래의 데이터로 재구성이 가능해야 함

    -> 상호 중첩 배제 : 분할된 데이터는 서로 다른 분할의 항목에 속하지 않아야 함

- 분할 방법

    -> 수평 분할 : 행 단위로 분할

    -> 수직 분할 : 속성 단위로 분할



**할당**

- 동일한 분할을 여러 개의 서버에 생성하는 분산 방법

- 비중복 할당 방식

    -> 최적의 노드를 선택해서 분산 데이터베이스의 단일 노드에서만 분할이 존재

- 중복 할당 방식

    -> 동일한 테이블을 다른 서버에 복제하는 방식

### 데이터베이스 이중화 / 서버 클러스터링

**데이터베이스 이중화**

- 시스템 오류로 인한 데이터베이스 서비스 중단이나 물리적 손상 발생 시 이를 복구하기 위해 동일한 데이터베이스를 복제하여 관리하는 것

- 하나 이상의 데이터베이스가 항상 같은 상태를 유지

- 데이터베이스가 문제가 생기면 즉시 해결 가능

- 사용자가 작업을 수행하면 이중화 시스템에 연결된 다른 데이터베이스도 동일하게 적용

- 애플리케이션을 여러 개의 데이터베이스로 분산시켜 처리해 데이터베이스의 부하를 감소

**데이터베이스 이중화의 분류**

- 내용 전달 방식에 따른 분류

- Eager 기법 : 트랜잭션 수행 중 데이터 변경 발생 시 이중화된 모든 데이터베이스에 즉시 전달하여 변경 내용이 즉시 적용

- Lazy 기법 : 트랜잭션 수행이 종료되면 변경 사실을 새로운 트랜잭션에 작성하여 각 데이터베이스에 전달하여 데이터베이스마다 새로운 트랜잭션이 수행되는 것으로 간주

**데이터베이스 이중화 구성 방법**

- 활동-대기 방법

    -> 한 DB가 서비스 시 다른 DB는 대기

    -> 활성 DB에 장애 발생 시 대기 중이었던 DB가 모든 서비스를 대신 수행

    -> 구성 방법 및 관리가 쉬워 많은 기업에서 사용

- 활동-활동 방법

    -> 두 개의 DB가 서로 다른 서비스를 제공

    -> 한쪽 DB에 장애 발생 시 다른 DB가 서비스를 제공

    -> 두 DB가 모두 처리를 해 처리율이 높지만 구성 방법 및 설정이 복잡

**클러스터링**

- 두 대 이상의 서버를 하나의 서버처럼 운영하는 기술

- 서버 이중화 및 공유 스트로지를 사용하여 서버의 고가용성을 제공

    -> 고가용성 : 시스템을 오랜 시간 동안 계속해서 정상적으로 운영이 가능한 성질

- 병렬 처리 클러스터링 : 처리율을 높이기 위해 하나의 작업을 여러 개의 서버에서 분산하여 처리

- 고가용성 클러스터링 : 하나의 서버에 장애가 발생 시 다른 서버가 받아서 처리하여 서비스 중단을 방지하는 방식

### 데이터베이스 보안 / 암호화

**데이터베이스 보안의 개요**

- 데이터베이스의 일부분 또는 전체에 권한이 없는 사용자가 액세스 하는 것을 금지하기 위해 사용되는 기술

**암호화 / 복호화**

- 암호화는 데이터를 보낼 때 송신자가 지정한 수신자 외는 그 내용을 알 수 없도록 평문을 암호문으로 변환

- 암호화 과정 : 암호화되지 않은 평문을 정보 보호를 위해 암호문으로 바꿈

- 복호화 과정 : 암호문을 원래의 평문으로 바꿈

**개인키 / 공개키 암호 방식**

- 암호화 방식의 키와 복호화 방식의 키가 같을 때

    -> 개인키 / 비밀키 / 대칭키 암호 방식

    -> 종류 : 전위 기법, 대수 기법, 합성 기법(DES)

- 암호화 방식의 키와 복호화 방식의 키가 다를 때

    -> 공개키 / 비대칭키

    -> RSA 기법

## [3.2 물리 데이터베이스 설계(4)](http://127.0.0.1:5503/106)

2020. 3. 2. 23:56

[2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [2020정처기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0), [2020정처기필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0), [2020정처기필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [데이터베이스](http://127.0.0.1:5503/tag/%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4)

728x90

### 데이터베이스 보안 - 접근통제

**접근통제**

- 데이터가 저장된 객체와 사용하려는 주체 사이의 정보 흐름을 제한

- 데이터에 대해 통제를 함으로써 자원의 불법적인 접근 및 파괴를 예방

- 임의 접근통제(Discretionary Access Control)

    -> 데이터에 접근하는 사용자의 신원에 따라 접근 권한 부여

    -> 통제 권한이 주체에 있어 접근통제 권한을 주체가 지정하고 제어

    -> 객체를 생성한 사용자가 객체에 대한 모든 권한을 부여받고 다른 사용자에게 허가 가능

    -> SQL 명령어로는 GRANT, REVOKE가 있음

- 강제 접근통제(Mandatory Access Control)

    -> 주체와 객체의 등급을 비교하여 접근 권한을 부여

    -> 제 3자가 접근통제 권한 지정

    -> 객체별로 보안 등급을 부여할 수 있으며 사용자별로 인가 등급을 부여할 수 있음

    -> 주체의 보안 등급이 자신의 보안등급보다 높으면 읽기, 수정, 등록이 모두 불가하고 동등하면 읽기, 수정, 등록이 모두 가능하며 낮으면 읽기만 가능

**접근통제 정책**

- 어떤 주체(Who)가 언제(When) 어디서(Where) 어떤 객체(What)에게 어떤 행위(How)에 대한 허용 여부를 정의하는 것

- 신분 기반 정책

    -> 주체나 그룹의 신분에 근거하여 객체의 접근을 제한

    -> IBP(Individual-Based Policy) : 최소 권한 정책, 단일 주체에게 하나의 객체에 대한 허가를 부여

    -> GBP(Group-Based Policy) : 복수 주체에 대한 허가를 부여

- 규칙 기반 정책

    -> 주체가 갖는 권한에 근거하여 객체의 접근을 제한

    -> MLP(Multi-Level Policy) : 사용자 및 객체별로 지정된 기밀 분류에 따른 정책

    -> CBP(Compartment-Based Policy) : 집단별로 지정된 기밀 허가에 따른 정책

- 역할 기반 정책

    -> 주체의 신분이 아니라 주체가 맡은 역할에 근거하여 객체의 접근을 제한

**접근통제 매커니즘**

- 정의된 접근통제 정책을 구현하는 기술적인 방법

- 접근통제 목록 : 객체를 기준으로 특정 객체에 대해 어떤 주체가 어떤 행위를 할 수 있는지 기록한 목록

- 능력 리스트 : 주체를 기준으로 주체에게 허가된 자원 및 권한을 기록한 목록

- 보안 등급 : 주체나 객체에 부여된 보안 속성의 집합

- 패스워드 : 주체가 자신임을 증명할 때 사용하는 인증 방법

- 암호화

암호화 / 복호화

- 암호화는 데이터를 보낼 때 송신자가 지정한 수신자 외는 그 내용을 알 수 없도록 평문을 암호문으로 변환

- 암호화 과정 : 암호화되지 않은 평문을 정보 보호를 위해 암호문으로 바꿈

- 복호화 과정 : 암호문을 원래의 평문으로 바꿈

**접근통제 보안 모델**

- 보안 정책을 구현하기 위한 정형화된 모델

- 기밀성 모델

    -> 군사적인 목적으로 개발된 최초의 수학적 모델로 기밀성 보장이 최우선

- 무결성 모델

    -> 기밀성 모델에서 발생하는 불법적인 정보 변경을 방지하기 위해 무결성을 기반으로 개발된 모델

    -> 데이터 일관성 유지에 중점을 두어 개발

- 접근통제 모델

    -> 접근통제 매커니즘을 보안 모델로 발전시킨 것

    -> 접근통제 모델 : 임의적인 접근 통제를 관리하기 위한 모델로 행은 주체 열은 객체를 의미

    -> R : 읽기 권한

    -> W : 쓰기 권한

    -> ALL : 모든 권한

접근통제 행렬

**접근통제 조건**

- 접근통제 매커니즘의 취약점을 보안하기 위해 접근 통제 정책에 부가하여 적용할 수 있는 조건

- 값 종속 통제 : 객체에 저장된 값에 따라 다르게 접근통제를 허용해야 하는 경우

- 다중 사용자 통제 : 지정된 객체에 다수의 사용자가 동시에 접근을 요구하는 경우

- 콘텍스트 기반 통제 : 특정 시간, 네트워크 주소, 접근 경로, 인증 경로 등에 근거하여 접근을 제어하는 방식

**감사 추적**

- 사용자나 애플리케이션의 데이터베이스에 접근하여 수행한 모든 활동을 기록하는 기능

- 오류가 발생한 데이터베이스를 복구하거나 부적절한 데이터 조작을 파악하기 위해 사용

### 데이터베이스 백업

**데이터베이스 백업**

- 전산 장비의 장애에 대비하여 데이터베이스에 저장된 데이터를 보호하고 복구하기 위한 작업

**데이터베이스 장애 유형**

- 사용자 실수 : 사용자의 실수로 인한 오류

- 미디어 장애 : 하드웨어 장애나 데이터가 파손된 경우

- 구문 장애 : 프로그램 오류나 사용 공간의 부족으로 발생하는 장애

- 사용자 프로세스 장애 : 프로그램이 비정상적으로 종료되거나 네트워크 이상으로 세션이 종료되어 발생하는 장애

- 인스턴스 장애 : 하드웨어 장애, 정전, 시스템 파일 파손 등 비정상적 요인으로 메모리나 데이터베이스 서버의 프로세스가 중단된 경우

**로그 파일**

- 데이터베이스의 처리 내용이나 이용 상황 등 상태 변화를 시간의 흐름에 따라 기록한 파일

- 로그 파일을 기반으로 과거 상태로 복귀(UNDO)시키거나 현재 상태로 재생(REDO)시켜 데이터베이스 상태를 일관성 있게 유지

**데이터베이스 복구 알고리즘**

- 동기적/비동기적 갱신에 따라 분류

- NO-UNDO/REDO : 비동기적으로 갱신한 경우

- UNDO/NO-REDO : 동기적으로 갱신한 경우

- UNDO/REDO : 동기/비동기적으로 갱신한 경우

- NO-UNDO/NO-REDO : 동기적으로 저장 매체에 기록하지만 데이터베이스와는 다른 영역에 기록한 경우

**백업 종류**

- 물리 백업 : 데이터베이스 파일을 백업하는 방법

- 논리 백업 : 데이터베이스 내의 논리적 객체들을 백업하는 방법

### 스토리지

**스토리지의 개요**

- 단일 디스크로 처리할 수 없는 대용량의 데이터를 저장하기 위해 서버와 저장장치를 연결하는 기술

**DAS(Direct Attached Storage)**

- 서버와 저장장치를 전용 케이블로 직접 연결하는 방식

- 서버에서 저장장치를 관리

- 저장장치를 직접 연결하므로 속도가 빠르고 설치 및 운영이 쉬움

- 다른 서버에서 스토리지에 접근하여 사용 불가

**NAS(Network Attached Storage)**

- 서버와 저장장치를 네트워크를 통해 연결하는 방식

- 별도의 파일 관리 기능이 있는 NAS Storage가 내장된 저장장치를 직접 관리

- DAS에 비해 확장성 및 유연성이 좋음

- 서버들이 자유롭게 스토리지에 접근하여 파일 공유

**SAN(Storage Area Network)**

- DAS의 빠른 처리와 NAS의 파일 공유 장점을 혼합한 방식

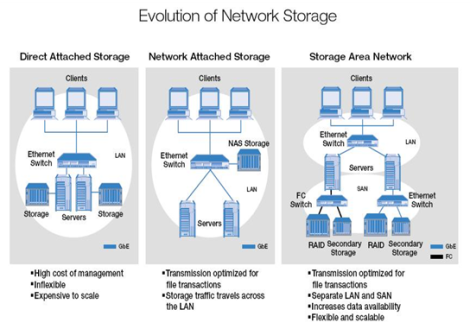
- 서버와 저장장치를 연결하는 전용 네트워크를 별도로 구성

- 파이버 채널(FC) 스위치를 이용하여 네트워크를 구성

    -> 파이버 채널 : 장치 간 데이터 전송 속도를 기가바이트로 높이기 위한 네트워크 기술

- 서버나 저장장치를 광케이블로 연결하므로 처리 속도가 빠름

- 서버들이 저장장치 및 파일을 자유롭게 공유

DAS, NAS, SAN 비교

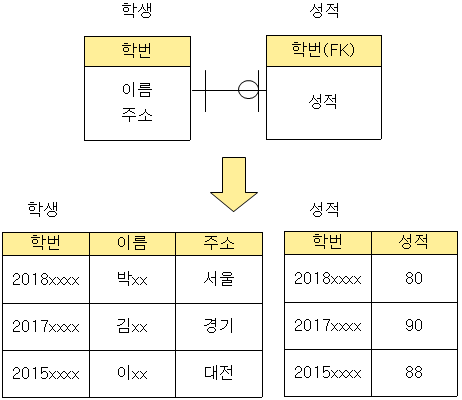
### 논리 데이터 모델의 물리 데이터 모델 변환

**테이블**

- 데이터를 저장하는 데이터베이스의 가장 기본적인 오브젝트

**엔티티를 테이블로 변환**

- 논리 데이터 모델에서 정의된 엔티티를 물리 데이터 모델의 테이블로 변환

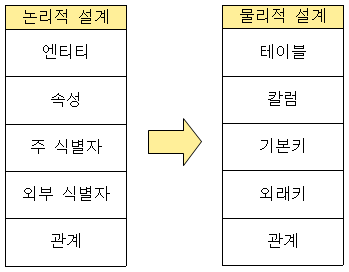


- 테이블과 엔티티 명칭은 동일하게 하는 것을 권고

- 테이블은 소스코드 가독성을 위해 영문명을 사용

- 표준화된 용어 사용 지향

- 변환 규칙



**슈퍼타입/서브타입을 테이블로 변환**

- 슈퍼타입과 서브타입은 논리 데이터 모델에서 이용되는 형태이므로 물리 데이터 모델을 설계할 때는 테이블로 변환

- 슈퍼타입 기준 테이블 변환

    -> 서브타입을 슈퍼타입에 통합하여 하나의 테이블로 만듦

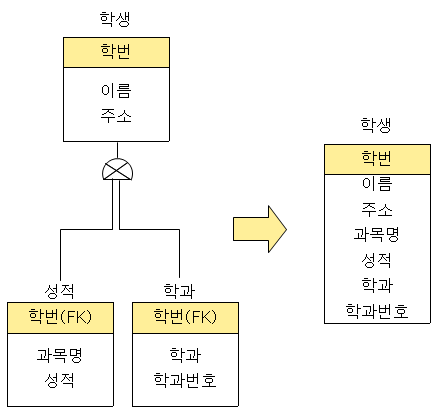
    -> 데이터의 액세스가 상대적 용이

    -> 뷰를 이용하여 각각의 서브타입만을 액세스 하거나 수정할 수 있음

    -> SQL 문장 구성이 단순

    -> 디스크 저장 공간이 증가

    -> 인덱스의 효율이 떨어짐



- 서브타입 기준 테이블 변환

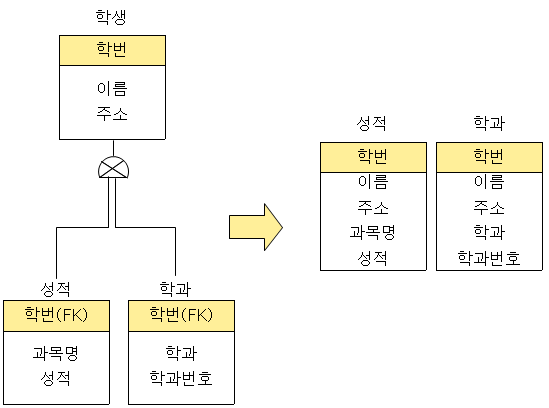
    -> 슈퍼타입의 속성들을 각각의 서브타입에 추가하여 서브타입들을 개별적인 테이블로 만듦

    -> 각 서브타입 속성들의 선택 사양이 명확한 경우 유리

    -> 여러 개의 테이블로 통합하므로 테이블당 크기가 감소하여 전체 테이블 스캔 시 유리

    -> 수행 속도 감사

    -> 복잡한 처리를 하는 SQL의 통합이 어려움

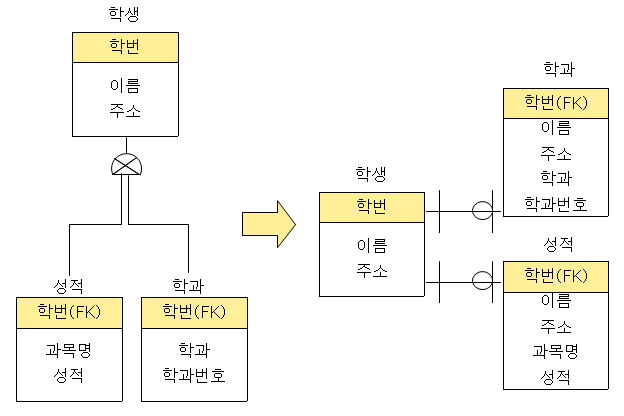


- 개별타입 기준 테이블 변환

    -> 슈퍼타입과 서브타입들을 각각의 개별적인 테이블로 변환

    -> 슈퍼타입과 서브타입 테이블 사이에는 각각 1:1 관계가 형성

    -> 저장공간이 상대적으로 작음



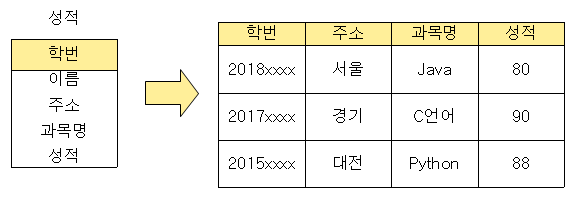
**속성을 칼럼으로 변환**

- 논리 데이터 모델에서 정의한 속성을 물리 데이터 모델의 컬럼으로 변환

- 일반 속성 변환

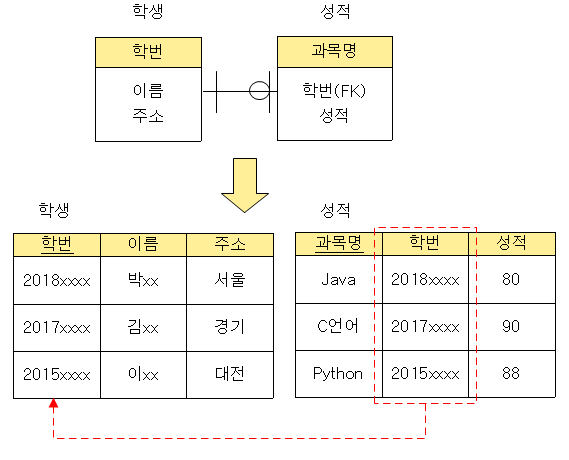
     -> 엔티티의 속성을 테이블의 각각의 컬럼으로 변환

     -> 칼럼명은 SQL 예약어 사용을 피하고 가능한 한 짧게 지정



**관계를 외래키로 변환**

- 논리 데이터 모델에서 정의된 관계는 기본키와 이를 참조하는 기본키로 변환



**관리 목적의 테이블/칼럼 추가**

- 논리 데이터 모델에는 존재하지 않는 테이블이나 칼럼을 데이터베이스 관리 혹은 이용하는 프로그래밍의 수행 속도를 향상시키기 위해 물리 데이터 모델에 추가할 수 있음

**데이터 타입 선택**

- 정의된 논리적인 데이터 타입을 물리적인 DBMS의 물리적 특성과 성능을 고려하여 최적의 데이터 타입과 데이터의 최대 길이를 선택

- Oracle에서 자주 사용되는 데이터 유형

    -> CHAR : 고정길이 문자열

    -> VARCHAR : 가변 길이 문자열

    -> NUMBER : 38 자릿수의 숫자 저장 가능

    -> DATE : 날짜 저장

## [3.3 SQL 응용(1)](http://127.0.0.1:5503/95)

2020. 3. 2. 23:57

[2020정보처리기사](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [alter](http://127.0.0.1:5503/tag/alter), [Create](http://127.0.0.1:5503/tag/Create), [DCL](http://127.0.0.1:5503/tag/DCL), [DDL](http://127.0.0.1:5503/tag/DDL), [DML](http://127.0.0.1:5503/tag/DML), [DROP](http://127.0.0.1:5503/tag/DROP), [SQL](http://127.0.0.1:5503/tag/SQL), [정처기필기](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0)

728x90

### SQL 개념

**SQL의 개요**

- 국제 표준 데이터베이스 언어이며 많은 관계형 데이터베이스(RDB)를 지원하는 언어로 채택

**SQL의 분류**

- DDL(Data Define Language, 데이터 정의어)

    -> 스키마, 도메인, 테이블, 뷰, 인덱스를 정의, 변경, 삭제할 때 사용하는 언어

    -> CREATE : 스키마, 도메인, 테이블, 뷰, 인덱스를 정의

    -> ALTER : 테이블에 대한 정의를 변경

    -> DROP : 스키마, 도메인, 테이블, 뷰, 인덱스를 삭제

- DML(Data Manipulation Language, 데이터 조작어)

    -> 사용자가 저장된 데이터를 실질적으로 처리하는 데 사용

    -> SELECT : 테이블에서 조건에 맞는 튜플 검색

    -> INSERT : 테이블에 새로운 튜플 삽입

    -> DELETE : 테이블에서 조건에 맞는 튜플 삭제

    -> UPDATE : 테이블에서 조건에 맞는 튜플의 내용 변경

- DCL(Data Control Language, 데이터 제어어)

    -> 데이터의 보안, 무결성, 회복, 병행 수행 제어 등을 정의하는 데 사용하는 언어

    -> COMMIT : 명령에 의해 수행된 결과를 실제 물리적 디스크로 저장하고 데이터베이스 조작 작업이 정상적으로 완료되었음을 알려줌

    -> ROLLBACK : 데이터베이스 조작 작업이 비정상적으로 종료되었을 때 원래의 상태로 복구

    -> GRANT : 데이터베이스 사용자에게 사용 권한을 부여

### DDL

**DDL의 개념**

- DDL(데이터 정의어)는 DB 구조, 데이터 형식, 접근 방식 등 DB를 구축하거나 수정할 목적으로 사용하는 언어

- DDL의 종류에는 CREATE, ALTER, DROP이 있음

**CREATE**

- CREATE SCHEM

    -> 스키마(데이터베이스 구조와 제약 조건에 관한 전반적인 명세를 기술한 것)를 정의

CREATE SCHEMA 스키마명 AUTHORIZATION 사용자\_id;

     -> 소유권자의 사용자 ID가 '김이박'이고 스키마 '성적'을 정의하는 SQL문

CREATE SCHEMA 성적 AUTHORIZATION 김이박;

- CREATE DOMAIN

     -> 도메인을 정의

     -> 대괄호는 생략 가능하다는 의미

CREATE DOMAIN 도메인명 [AS] 데이터\_타입

[DEFAULT 기본값]

[CONSTRAINT 제약조견명 CHECK (범위값)];

     -> 성별을 '남' 또는 '여'와 같이 정해진 1개 문자로 표현되는 도메인 GENDER를 정의하는 SQL문

CREATE DOMAIN [AS] CHAR(1)

DEFAULT '남'

CONSTRAINT VALID-GENDER CHECK(VALUE IN('남','여');

- CREATE TABLE

     -> 테이블을 정의

CREATE TABLE 테이블명

(속성명 데이터\_타입 [DEFAULT 기본값][NOT NULL], ...

[, PRIMARAY KEY(기본키\_속성명), ...)]

[, UNIQUE(대체키\_속성명), ...)]

[, FOREIGN KEY(외래키\_속성명, ...)]

REFERENCES 참조테이블(기본키\_속성명, ...)]

[ON DELETE 옵션]

[ON UPDATE 옵션]

[, CONSTRAINT 제약조건명][CHECK (조건식)]);

     -> '이름', '학번', '전공', '성별', '생년월일'로 구성된 학생 테이블을 정의하는 SQL문

     -> 제약조건

         \* '이름'은 NULL 일 수 없음

         \* '학번'은 기본키

         \* '전공'은 학과 테이블의 '학과 코드'를 참조하는 외래키로 사용

         \* 학과 테이블에서 삭제가 일어나면 관련된 튜플들의 전공 값을 NULL로 만듦

         \* 학과 테이블에서 '학과 코드'가 변경되면 전공 값도 같은 값으로 변경

         \* '생년월일'은 1980-01-01 이후의 데이터

         \* 제약 조건의 이름은 '생년월일 제약'

         \* 각 속성의 데이터 타입은 적당하게 지정

         \* '성별'은 도메인 GENDER 사용

CREATE TABLE 학생

(이름 VARCHAR(15) NOT NULL,

학번 CHAR(8)

전공 CHAR(5)

성별 GENDER

생년월일 DATE

PRIMARY KEY(학번)

FOREIGN KEY(전공) REFERENCES 학과(학과코드)

ON DELETE SET NULL

ON UPDATE CASCADE

CONSTRAINT 생년월일제약 CHECK(생년월일>='1980-01-01'));

- CREATE VIEW

     -> 뷰(하나 이상의 기본 테이블로부터 유도되는 이름을 갖는 가상 테이블)를 정의

     -> SELECT문의 결과로써 뷰를 생성

CREATE VIEW 뷰명[(속성명[, 속성명, ...])]

AS SELECT문;

     -> 고객 테이블에서 '주소'가 '포천시'인 고객들의 '이름'과 '전화번호'를 '포천 고객'이라는 뷰로 정의하는 SQL문

CREATE VIEW 포천고객(이름, 전화번호)

AS SELECT 이름,전화번호

FROM 고객

WHERE 주소 = '포천시';

- CREATE INDEX

     -> 인덱스(검색 시간을 단축시키기 위해 만든 보조적인 데이터 구조)를 정의

     -> ASC : 오름차순 정렬, 생략 시 기본 값 / DESC : 내림차순 정렬

     -> CLUSTER : 사용 시 인덱스를 클러스터드 인덱스로 지정

     -> 클러스터드 인덱스 : 인덱스 키의 순서에 따라 데이터가 정렬되어 저장되는 방식

CREATE [UNIQUE] INDEX 인덱스명

ON 테이블명(속성명 [ASC|DESC][,속성명 [ASC|DESC]])

[CLUSTER];

     -> 고객 테이블에서 UNIQUE 한 특성을 갖는 '고객번호'속성에 대해 내림차순으로 정렬하여 '고객번호\_idx'라는 이름으로 인덱스를 정의하는 SQL문

CREATE UNIQUE INDEX 고객번호\_idx

ON 고객(고객번호 DESC);

**ALTER**

- ALTER TABLE

     -> 테이블에 대한 정의를 변경

ALTER TABLE 테이블명 ADD 속성명 데이터\_타입 [DEFAULT '기본값'];

ALTER TABLE 테이블명 ALTER 속성명 [SET DEFAULT '기본값'];

ALTER TABLE 테이블명 DROP COLUMN 속성명 [CASCADE];

     -> 학생 테이블에 최대 3 문자로 구성되는 '학년' 속성을 추가하는 SQL문

ALTER TABLE 학생 ADD 학년 VARCHAR(3);

     -> 학생 테이블의 학년 필드의 데이터 타입과 크기를 최대 10글자로 하고 NULL값이 입력되지 않게 하는 SQL문

ALTER TABLE 학생 ALTER 학년 VARCHAR(10) NOT NULL;

**DROP**

- 스키마, 도메인, 기본 테이블, 튜 테이블, 인덱스, 제약 조건 등을 제거하는 명령문

DROP SCHEMA 스키마명 [CASCADE | RESTRICTED];

DROP DOMAIN 도메인명 [CASCADE | RESTRICTED];

DROP TABLE 테이블명 [CASCADE | RESTRICTED];

DROP VIEW 뷰명 [CASCADE | RESTRICTED];

DROP INDEX 인덱스명 [CASCADE | RESTRICTED];

DROP CONSTRAINT 제약조건명

- CASCADE : 제거할 때 참조 관계에 있는 테이블의 데이터도 연쇄 삭제

- RESTRICTED : 제거할 때 참조하고 있는 테이블이 있다면 삭제를 취소

### DCL

**DCL의 개념**

- DCL(데이터 제어어)는 데이터의 보안, 무결성, 회복, 병행 제어 등을 정의하는 데 사용하는 언어

**GRANT**

- 데이터베이스 관리자가 사용자에게 권한 부여

**REVOKE**

- 데이터베이스 관리자가 사용자에게 권한 취소

**COMMIT**

- 트랜잭션이 성공적으로 끝난 후 변경된 내용을 데이터베이스에 반영

**ROLLBACK**

- COMMIT 되지 않은 변경된 내용을 취소하고 데이터베이스를 이전 상태로 되돌리는 명령

**SAVEPOINT**

- 트랜잭션 내에 ROLLBACK 할 위치인 저장점을 저장

## [3.3 SQL 응용(2)](http://127.0.0.1:5503/97)

2020. 3. 2. 23:57

[2020정보처리기사](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [select](http://127.0.0.1:5503/tag/select), [정처기필기](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0)

728x90

### DML

**DML의 개념**

- DML(데이터 조작어)는 데이터베이스 사용자가 응용 프로그램이나 질의어를 통해 저장된 데이터를 관리하는 데 사용하는 언어

**INSERT**

- 테이블에 새로운 튜플을 삽입

INSERT INTO 테이블명([속성명1, 속성명2, ...])

VALUES (데이터1, 데이터2, ...)

- 사원 테이블에 (이름 - 김이박, 부서 - 개발)을 삽입하는 SQL문

INSERT INTO 사원(이름, 부서) VALUE(김이박, 개발);

- 사원 테이블에 있는 개발의 모든 튜플을 개발 부원(이름, 생일) 테이블에 삽입하는 SQL문

INSERT INTO 개발부원(이름, 주소)

SELECT 이름, 주소

FROM 사원

WHERE 부서 = "개발"

**DELETE**

- 테이블의 튜플 중 특정 튜플을 삭제

DELETE

FROM 테이블명

[WHERE 조건]

- 사원 테이블에서 부서가 개발인 튜플을 삭제하는 SQL문

DELETE FROM 사원 WHERE 부서 = "개발";

**UPDATE**

- 테이블의 튜플 중 특정 튜플의 내용을 변경

UPDATE 테이블명

SET 속성명 = 데이터[, 속성명=데이터, ...]

[WHERE 조건];

- 사원 테이블에서 김이박의 부서를 IT로 변경하는 SQL문

UPDATE 사원

SET 부서 = "IT"

WHERE 이름 = "김이박";

**SELECT**

- 테이블에서 튜플을 검색

SELECT [PREDICATE] [테이블명.]속성명 [AS 별칭][, [테이블명.]속성명, ...]

FROM 테이블명[, 테이블명, ...]

[, WINDOW 함수 OVER (PARTITION BY 속성명1, 속성명2, ... ORDER BY 속성명3, 속성명4, ...) [AS 별칭]]

[WHERE 조건]

[GROUP BY 속성명, 속섬영, ..]

[HAVING 조건]

[ORDER BY 속성명 [ASC | DESC]];

- ORDER BY : 특정 속성을 기준으로 정렬하여 검색

    -> ASC : 오름차순

    -> DESC : 내림차순

- WINDOW 함수 : GROUP BY 절을 이용하지 않고 속성의 값을 집계할 함수를 기술

    -> PARTITION BY : WINDOW 함수가 적용될 범위로 사용할 속성을 지정

    -> ORDER BY : PARTITION 안에서 정렬 기준으로 사용할 속성을 지정

    -> ROW\_NUMBER() : 각 레코드에 대한 일련번호 반환

    -> RANK() : 순위를 반환하되 공동 순위를 반영

    -> DENSE\_RANK() : 순위를 반환하되 공동 순위를 반영하지 않음

- GROUP BY : 특정 속성을 기준으로 그룹화하여 검색할 때 사용. 그룹 함수와 같이 사용

    -> COUNT(속성명) : 그룹별 튜플 수를 구하는 함수

    -> SUM(속성명) : 그룹별 합계를 구하는 함수

    -> AVG(속성명) : 그룹별 평균을 구하는 함수

    -> MAX(속성명) : 그룹별 최대값을 구하는 함수

    -> MIN(속성명) : 그룹별 최소값을 구하는 함수

    -> ROLLUP(속성명, 속성명, ...) : 인수로 주어진 속성을 대상으로 그룹별 소계를 구하는 함수

    -> CUBE(속성명, 속성명, ...) : 인수로 주어진 속성을 대상으로 모든 조합의 그룹별 소계를 구하는 함수

- HAVING : GROUP BY와 함께 사용하여 그룹에 조건을 지정

**SELECT 예제**

예제

- 기본 검색

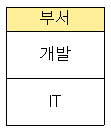
    -> 사원 테이블에 있는 모든 튜플을 검색하는 SQL문

SELETE \* FROM 사원;



-> 사원 테이블에서 부서를 중복없이 검색하는 SQL문

SELECT DISTINCT 부서 FROM 사원;



- 조건 지정 검색

    -> 사원 테이블에서 부서가 개발인 튜플을 검색하는 SQL문

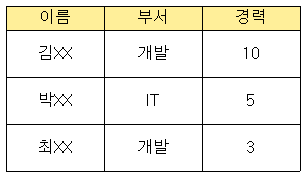
SELECT \* FROM 사원 WHERE 부서 = "개발";



    -> 사원 테이블에서 부서가 개발 혹은 IT인 튜플을 검색하는 SQL문

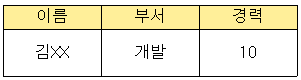
SELECT \* FROM 사원 WHERE 부서 = "개발" OR 부서 = "IT";

SELECT \* FROM 사원 WHERE 부서 IN("개발","IT");



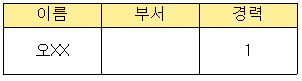
    -> 사원 테이블에서 성이 김인 튜플을 검색하는 SQL문

SELECT \* FROM 사원 WHERE 이름 LIKE "김%";



    -> 사원 테이블에서 부서가 NULL인 튜플을 검색하는 SQL문

SELECT \* FROM 사원 WHERE 부서 IS NULL;



- 정렬 검색 : ORDER BY를 이용한 검색

    -> 사원 테이블에서 이름을 오름차순으로 정렬하여 튜플을 검색하는 SQL문

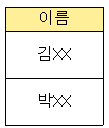
SELECT \* FROM 사원 ORDER BY 이름 ASC;



- 하위 질의 : 조건절에 다시 SELECT문을 넣어 그 결과를 조건으로 검색

    -> 취미 테이블에서 취미활동이 축구인 사원 이름의 튜플을 검색하는 SQL문

SELECT 이름 FROM 사원 WHERE 이름 = (SELECT 이름 FROM 취미 WHERE 취미활동 = "축구");



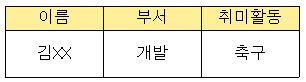
- 복수 테이블 : 여러 테이블을 대상으로 검색

    -> 경력이 10년 이상인 사원의 이름, 부서, 취미활동을 검색하는 SQL문

SELECT 사원.이름, 사원.부서

FROM 사원, 취미

WHERE 사원.경력 >= 10 AND 사원.이름 = 취미.이름;



**SELECT 예제 2**

예제 2

- WINDOW 함수 이용

    -> 사원 테이블에서 부서 별로 경력에 대한 일련번호를 구하여 짬순이라는 이름을 붙이는 SQL문

SELECT 부서, 경력

ROW\_NUMBER() OVER (PARTITION BY 부서 ORDER BY 경력 DESC) AS 짬순

FROM 사원;



- GROUP BY : 그룹 지정 검색

    -> 사원 테이블에서 부서별 경력의 평균을 구하는 SQL문

SELECT 부서, AVG(경력) AS 평균

FROM 사원

GROUP BY 부서;



- 집합 연산자를 이용한 통합 질의

SELECT 속성명1, 속성명2, ...

FROM 테이블명

UNION | UNION ALL | INTERSECT | EXCEPT

SELECT 속성명1, 속성명2, ...

FROM 테이블명

[ORDER BY 속성명 [ASC | DESC]];

- UNION : 두 SELECT 문의 결과를 통합하고 중복된 행은 한 번만 출력

- UNION ALL : 두 SELECT 문의 결과를 통합하고 중복된 행도 그대로 출력

- INTERSECT : 두 SELECT 문의 결과 중 공통된 행만 출력

- EXCEPT : 첫번째 SELECT 문의 결과에서 두 번째 SELECT 문의 결과를 제외한 행을 출력

## [3.4 SQL 활용(1)](http://127.0.0.1:5503/109)

2020. 3. 4. 18:36

[2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [2020정처기필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0), [2020정처기필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [사용자정의함수](http://127.0.0.1:5503/tag/%EC%82%AC%EC%9A%A9%EC%9E%90%EC%A0%95%EC%9D%98%ED%95%A8%EC%88%98), [정처기필기정리](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [트리거](http://127.0.0.1:5503/tag/%ED%8A%B8%EB%A6%AC%EA%B1%B0), [프로시저](http://127.0.0.1:5503/tag/%ED%94%84%EB%A1%9C%EC%8B%9C%EC%A0%80)

728x90

### 프로시저

**프로시저의 개요**

- 절차형 SQL을 활용하여 특정 기능을 수행하는 일종의 트랜잭션 언어

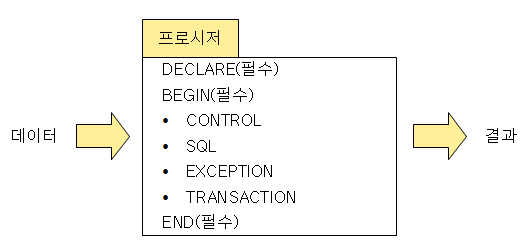
- 호출을 통해 실행되어 미리 저장해놓은 SQL 작업 수행

- 여러 프로그램에서 호출하여 사용 가능

- 데이터베이스에 저장되어 수행되기 때문에 스토어드 프로시저라고도 함

- 시스템의 일일 마감 작업, 일괄 작업 등에 사용

**프로시저의 구성**



- DECLARE : 프로시저의 명칭, 변수, 인수, 데이터 타입을 정의하는 선언부

- BEGIN / END : 프로시저의 시작과 종료를 의미

- CONTROL : 조건문 또는 반복문이 삽입되어 순차적으로 처리

- SQL : DML, DCL이 삽입되어 데이터 관리를 위한 작업(조회, 추가, 수정, 삭제) 수행

- EXCPETION : BEGIN ~ END 안의 구문 실행 시 예외 처리

- TRANSACTION : 수행된 데이터 작업들을 DB에 저장할지 취소할지 결정

**프로시저 생성**

CREATE [OR REPLACE] PROCEDURE 프로시저명(파라미터)

[지역번수 선언]

BEGIN

프로시저 BODY;

END;

- OR REPLACE : 동일한 프로시저 이름이 이미 존재하는 경우 기존의 프로시저를 대체

- 파라미터

    -> IN : 호출 프로그램이 프로시저에게 값을 전달할 때 사용

    -> OUT : 프로시저가 호출 프로그램에게 값을 전달할 때 사용

    -> INOUT : 호출 프로그램이 프로시저에게 값을 전달하고, 프로시저 실행 후 호출 프로그램에게 값을 반환할 때 지정

    -> 매개변수명 : 호출 프로그램으로부터 전달받은 값을 저장할 변수의 이름을 지정

    -> 자료형 : 변수의 자료형을 지정

- 프로시저 BODY

    -> 프로시저의 코드를 기록

    -> BEGIN과 END 사이에 적어도 하나의 SQL 문이 존재해야 함

- 사원번호를 입력받아 해당 사원의 지급방식을 S로 변경하는 프로시저

CREATE OR REPLACE PROCEDURE emp\_change\_s(i\_사원번호 IN INT)

IS

BEGIN

UPDATE 급여 SET 지급방식 = 'S' WHERE 사원번호 = i\_사원번호;

EXCEPTION

WHEN PROGRAM\_ERROR THEN

ROLLBACK;

COMMIT;

END;

**프로시저 실행**

EXECUTE 프로시저명;

EXEC 프로시저명;

CALL 프로시저명;

- 위 3가지 명령어 중 하나를 사용

- 위에 정의한 emp\_change\_s 프로시저를 사원번호 32를 인수로 하여 실행

EXECUTE emp\_change\_s(32);

EXEC emp\_change\_s(32);

CALL emp\_change\_s(32);

**프로시저 제거**

DROP PROCEDURE 프로시저명;

- 위에 정의한 emp\_change\_s 프로시저를 제거

DROP PROCEDURE emp\_change\_s;

### 트리거

**트리거의 개요**

- 데이터베이스 시스템에서 데이터의 삽입 갱신 삭제 등의 이벤트가 발생할 때마다 관련 작업이 자동으로 수행되는 절차형 SQL

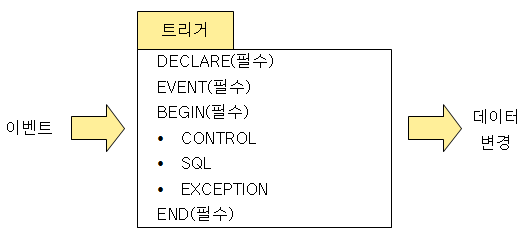
- 데이터베이스에 저장

- 데이터 변경 및 무결성 유지 로그 메시지 출력 등의 목적으로 사용

- DCL을 사용할 수 없으며, DCL이 포함된 프로시저나 함수를 호출하는 경우에도 오류 발생

- 트리거에 오류가 있는 경우 트리거가 처리하는 데이터에도 영향을 미침

**트리거의 구성**



- DECLARE : 트리거의 명칭, 변수 및 상수, 데이터 타입을 정의

- EVENT : 트리거가 실행되는 조건

- BEGIN / END : 트리거의 시작과 끝

- CONTROL : 조건문 또는 반복문이 삽입되어 순차적으로 처리

- SQL : DML문이 삽입되어 데이터 관리를 위한 작업(조회, 추가, 수정, 삭제) 수행

- EXCEPTION : BEGIN ~ END 안의 구문에서 예외가 발생 시 처리

**트리거의 생성**

CREATE [OR REPLACE] TRIGGER 트리거명 [동작시기 옵션][동작 옵션] ON 테이블명

REFERENCING [NEW | OLD] AS 테이블명

FOR EACH ROW

[WHEN 조건식]

BEGIN

트리거 BODY;

END;

- 동작 시기 옵션 : 트리거가 실행될 때를 지정

    -> AFTER : 테이블이 변경된 후

    -> BEFORE : 테이블이 변경되기 전

- 동작 옵션 : 트리거가 실행되게 할 작업의 종류를 지정

    -> INSERT : 새로운 튜플을 삽입할 때

    -> DELETE : 튜플을 삭제할 때

    -> UPDATE : 튜플을 수정할 때

- NEW | OLD : 트리거가 적용될 테이블의 별칭을 지정

    -> NEW : 추가되거나 수정에 참여할 튜플들의 집합(테이블)을 의미

    -> OLD : 수정되거나 삭제 전 대상이 되는 튜플들의 집합(테이블)을 의미

- FOR EACH ROW : 각 튜플마다 트리거를 적용한다는 의미

- WHEN 조건식 : 트리거를 적용할 튜플의 조건을 지정

- 학생 테이블에 새로운 튜플이 삽입될 때 튜플에 학년 정보가 누락되었으면 학년 필드에 신입생을 치환하는 트리거를 학년 정보\_tri라는 이름으로 정의

CREATE TRIGGER 학생정보\_tri BEFORE INSERT ON 학생

REFERENCING NEW AS new\_table

FOR EACH ROW

WHEN (new\_table.학년 IS NULL)

BEGIN

:new\_table.학년 := '신입생';

END;

**트리거의 제거**

DROP TRIGGER 트리거명;

- 위에 정의한 학생정보\_tri를 제거

DROP TRIGGER 학생정보\_tri;

### 사용자 정의 함수

**사용자 정의 함수의 개요**

- 프로시저와 유사하게 SQL을 사용하여 일련의 작업을 연속적으로 처리하여 종료 시 처리 결과를 단일 값으로 반환하는 절차형 SQL

- 데이터베이스에 저장되어 DML문의 호출에 의해 실행

- 예약어 RETURN을 통해 값이 반환되기 때문에 출력 파라미터가 없음

- 테이블 조작은 할 수 없고 SELECT를 통해 검색만 할 수 있음

- 프로시저를 호출하여 사용할 수 없음

**사용자 정의 함수의 구성**



- 프로시저의 구성과 유사

- RETURN : 호출 프로그램에 반환할 값이나 변수를 정의

**사용자 정의 함수 생성**

CREATE [OR REPLACE] FUNCTION 사용자 정의 함수명(파라미터)

[지역변수 선언]

BEGIN

사용자 정의 함수 BODY;

RETURN 반환;

END;

- 프로시저와 유사하며 파라미터에의 구성요소는 IN, 매개변수명, 자료형이 있음

- RETURN 반환값 : 반환할 값이나 반환할 값이 저장된 변수를 호출 프로그램으로 돌려줌

- i\_성별코드를 입력받아 1이면 남자, 2면 여자를 반환하는 사용자 정의 함수를 Get\_S\_성별로 정의

CREATE FUNCTION Get\_S\_성별(i\_성별코드 IN INT)

RETURN VARCHAR2

IS

BEGIN

IF (i\_성별코드 = 1) THEN

RETURN '남자';

ELSE

RETURN '여자';

END IF;

END;

**사용자 정의 함수 실행**

SELECT 사용자 정의 함수명 FROM 테이블명;

INSERT INTO 테이블명(속성명) VALUES (사용자 정의 함수명);

DELETE FROM 테이블명 WHERE 속성명 = 사용자 정의 함수명;

UPDATE 테이블명 SET 속성명 = 사용자 정의 함수명;

**사용자 정의 함수 제거**

DROP FUNCTION 사용자 정의 함수명;

- 위에 정의한 Get\_S\_성별을 제거

DROP FUNCTION Get\_S\_성별;

## [3.4 SQL 활용(2)](http://127.0.0.1:5503/111)

2020. 3. 4. 20:06

[2020정보처리기사](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [2020정처기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0), [2020정처기필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [DBMS접속기술](http://127.0.0.1:5503/tag/DBMS%EC%A0%91%EC%86%8D%EA%B8%B0%EC%88%A0), [ORM](http://127.0.0.1:5503/tag/ORM), [SQL테스트](http://127.0.0.1:5503/tag/SQL%ED%85%8C%EC%8A%A4%ED%8A%B8), [정처기필기](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0), [쿼리성능최적화](http://127.0.0.1:5503/tag/%EC%BF%BC%EB%A6%AC%EC%84%B1%EB%8A%A5%EC%B5%9C%EC%A0%81%ED%99%94)

728x90

### DBMS 접속 기술

**DBMS 접속 기술의 개요**

- 사용자가 데이터를 접속하기 위해 응용 시스템을 이용하여 DBMS에 접근하는 것

- 응용 시스템은 사용자로부터 매개 변수를 전달받아 SQL을 실행하고 DBMS로부터 전달받은 결과를 사용자에게 전달

- 웹 응용프로그램은 웹 응용 시스템을 통해 DBMS에 접근

- 웹 응용 시스템은 웹 서버와 웹 애플리케이션 서버(WAS)로 구성

    -> 사용자 ↔ 웹 서버 ↔ WAS ↔ DBMS

**DBMS 접속 기술**

- DBMS에 접근하기 위해 사용하는 API 또는 프레임워크를 의미

    -> API(Application Programming Interface) : 응용 프로그램 개발 시 운영 체제나 DBMS 등을 이용할 수 있도록 규칙 등에 대해 정의해 놓은 인터페이스

    -> 프레임워크 : 소프트웨어에서는 특정 기능을 수행하기 위해 필요한 클래스나 인터페이스 등을 모아둔 집합체

- JDBC(Java DataBase Connectivity)

    -> Java 언어

    -> 썬 마이크로시스템에서 출시

    -> Java SE에 포함되어 있고 JDBC 클래스는 java.sql, javax.sql에 포함

    -> 접속하려는 DBMS에 대한 드라이버 필요

- ODBC(Open DataBase Connectivity)

    -> 개발 언어와 상관 없음

    -> 마이크로소프트에서 출시

    -> MS-Access, DBase, DB2, Excel, Text 등 다양한 데이터베이스에 접근 가능

- MyBatis

    -> JDBC 코드를 단순화하여 사용할 수 있는 SQL Mapping 기반 오픈소스 접속 프레임워크

    -> SQL 문장을 분리하여 XML 파일을 만들고 Mapping을 통해 SQL을 실행

    -> SQL을 거의 그대로 사용할 수 있어 국내 환경에 적합

**동적 SQL**

- 개발 언어에 삽입되는 SQL 코드를 문자열 변수에 넣어 처리하는 것

- 조건에 따라 SQL 구문을 동적으로 변경하여 처리 가능

- NVL 함수를 사용할 필요가 없음

- 응용 프로그램 수행 시 SQL이 변형될 수 있어 프리컴파일할 때 구문 분석, 접근 권한 확인 등을 할 수 없음

### SQL 테스트

**SQL 테스트의 개요**

- SQL이 작성 의도에 맞게 원하는 기능을 수행하는지 검증하는 과정

- 단문 SQL은 코드를 직접 실행한 후 결과를 확인

- 절차형 SQL은 테스트 전에 생성을 통해 구문 오류나 참조 오류의 존재 여부 확인

- 정상적으로 생성된 절차형 SQL은 디버깅을 통해 로직을 검증하고 결과를 통해 최종 확인

**단문 SQL 테스트**

- DDL, DML, DCL이 포함되어 있는 SQL과 TCL을 직접 실행하여 테스트

- DESCRIBE 명령어를 이용하면 DDL로 작성된 테이블이나 뷰의 속성, 자료형, 옵션들을 확인할 수 있음

- DCL로 설정된 사용자 권한은 사용자 권한 정보가 저장된 테이블을 SELECT문으로 조회하거나 SHOW 명령어로 확인할 수 있음

    -> Oracle : SELECT \* FROM DBA\_ROLE\_PRIVES WHERE GRANTEE = 사용자;

    -> MySQL : SHOW GRANTS FOR 사용자@호스트;

**절차형 SQL 테스트**

- 프로시저, 사용자 정의 함수, 트리거 등의 절차형 SQL은 디버깅을 통해 기능의 적합성 여부 검증, 실행을 통해 결과를 확인하는 테스트를 수행

- SHOW 명령어를 통해 오류 내용을 확인

    -> SHOW ERRORS;

- 데이터베이스에 변화를 줄 수 있는 SQL문은 주석 처리 후 출력문을 이용하여 결과 확인

    -> Oracle : DBMS\_OUTPUT.ENABLE; / DBMS\_OUTPUT.PUT\_LINE(데이터);

    -> MySQL : SELECT 데이터;

### ORM

**ORM(Object-Relational Mapping)의 개요**

- 객체지향 프로그래밍의 객체와 관계형 데이터베이스의 데이터를 연결하는 기술

- 객체지향 프로그래밍에서 사용할 수 있는 가상의 객체지향 데이터베이스를 만들어 프로그래밍 코드와 데이터 연결

- 프로그래밍 코드 또는 데이터베이스와 독립적이므로 재사용 및 유지보수가 용이

- SQL 코드를 직접 사용하지 않기 때문에 직관적이고 간단하게 데이터 조작 가능

**ORM 프레임워크**

- ORM을 구현하기 위한 구조와 구현을 위해 필요한 여러 기능들을 제공하는 소프트웨어

- JAVA : JPA, Hibernate, EclipseLink, DataNucleus, Ebean 등

- C++ : ODB, QxOrm 등

- Python : Django, SQLAlchemy, Storm 등

- iOS : DatabaseObjects, Core Data 등

- .NET : NHibernate, DatabaseObjects, Dapper 등

- PHP : Doctrine, Propel, RedBean 등

**ORM의 한계**

- 프레임워크가 자동으로 작성하기 때문에 의도대로 작성되었는지 확인할 필요가 있음

- 객체지향적 사용을 고려, 설계한 데이터베이스가 아닌 경우 프로젝트가 크고 복잡할수록 ORM 기술을 적용하기 어려움

- 기존의 기업들은 ORM 고려하지 않은 데이터베이스를 사용하고 있어 ORM에 적합하게 변환하려면 많은 시간과 노력이 필요

### 쿼리 성능 최적화

**쿼리 성능 치적화의 개요**

- 데이터 입출력 애플리케이션의 성능 향상을 위해 SQL 코드를 최적화

- 최적화 전 APM을 사용하여 최적화할 쿼리 선정

    -> APM(Application Performance Management/Monitoring) : 애플리케이션의 성능 관리를 위해 다양한 모니터링 기능을 제공하는 도구

- RBO(Rule Based Optimizer) : 규칙 기반 옵티마이저

- CBO(Cost Based Optimizer) : 비용 기반 옵티마이저

RBO, CBO 비교

**실행 계획**

- DBMS의 옵티마이저가 수립한 SQL 코드의 실행 절차와 방법을 의미

- EXPLAIN 명령어를 통해 확인

- 그래픽이나 텍스트로 표현

- 요구사항을 처리하기 위한 연산 순서가 적혀있고 연산에는 조인, 테이블 검색, 필터, 정렬 등이 있음

**쿼리 성능 최적화**

- 실행 계획에 표시된 연산 순서, 조인 방식, 테이블 조회 방법 등을 참고하여 SQL문이 더 빠르고 효율적으로 작동하도록 코드와 인덱스를 재구성

- SQL 코드 재구성

    -> WHERE 절을 추가하여 일부 레코드만 조회

    -> WHERE 절에 연산자 사용 자제

    -> 특정 데이터 확인 시 IN보다 EXISTS 사용

    -> 힌트를 활용하여 실행 계획의 액세스 경로 및 조인 순서 변경

- 인덱스 재구성

    -> SQL 코드에서 조회되는 속성과 조건들을 고려하여 인덱스 구성

    -> 인덱스를 추가하거나 기존 인덱스의 열 순서 변경

    -> 단일 인덱스로 쓰기나 수정 없이 읽기로만 사용되는 경우 IOT(Index-Organized Table)로 구성

## [3.5 데이터 전환](http://127.0.0.1:5503/111)

### 데이터 전환

## [4.1 서버 프로그램 구현](http://127.0.0.1:5503/139)

2020. 3. 28. 23:35

[2020정보처리기사](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정처기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0), [2020정처기필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0), [배치프로그램](http://127.0.0.1:5503/tag/%EB%B0%B0%EC%B9%98%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%A8), [보안소프트웨어](http://127.0.0.1:5503/tag/%EB%B3%B4%EC%95%88%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4), [서버](http://127.0.0.1:5503/tag/%EC%84%9C%EB%B2%84), [패키지소프트웨어](http://127.0.0.1:5503/tag/%ED%8C%A8%ED%82%A4%EC%A7%80%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4)

728x90

### 개발 환경 구축

**개발 환경 구축 개요**

- 응용 소프트웨어 개발을 위해 개발 프로젝트를 이해하고 S/W 및 H/W 장비를 구축하는 것

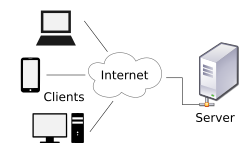
- 응용 소프트웨어가 운영될 환경과 유사한 구조로 구축

- 개발 프로젝트의 분석 단계의 산출물을 바탕으로 개발에 필요한 S/W와 H/W를 선정

- S/W와 H/W의 성능, 편의성, 라이선스 등 비즈니스 환경에 적합한 제품들을 최종적으로 결정하여 구축

**H/W 환경**

- 사용자의 인터페이스를 역할을 하는 클라이언트와 클라이언트와 통신하여 서비스하는 서버로 구성

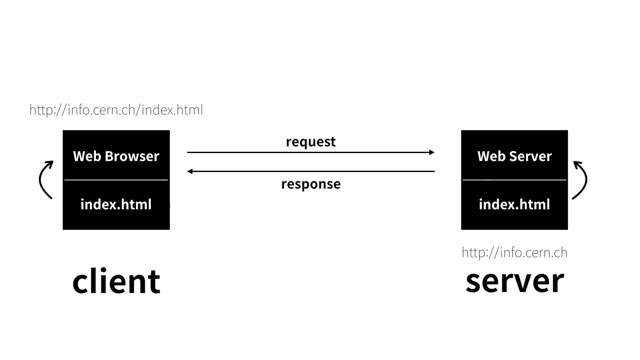
출처 : 위키백과

- 웹 서버

    -> 클라이언트로부터 직접 요청을 받아 처리하는 서버

    -> 저용량의 정적 파일(HTML, CSS 등)을 제공

    -> ex) Apache HTTP Server, Microsoft Internet Information Service 등

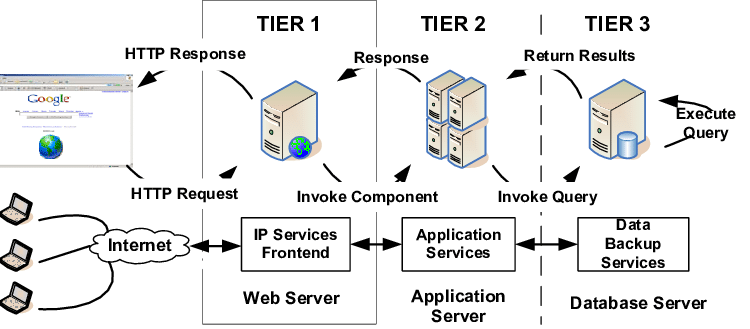
출처 : 생활코딩(https://opentutorials.org/course/3084/18890)

- 웹 애플리케이션 서버(WAS)

    -> 사용자에게 동적 서비스를 제공하기 위해 웹 서버로부터 요청을 받아 데이터 가공 작업을 수행

    -> 웹 서버와 데이터베이스 서버 또는 웹 서버와 파일 서버 사이에서 인터페이스 역할 수행

    -> ex) Apache Tomcat, IBM WebSphere 등

출처 : Research Gate(https://www.researchgate.net/figure/A-Typical-3-Tier-Server-Architecture-Tier-1-Web-Server-Tier-2-Application-Server-Tier\_fig1\_221147997)

- 데이터베이스 서버

    -> 데이터베이스와 이를 관리하는 DBMS를 운영하는 서버

    -> ex) MySQL Server, Oracle Server 등

- 파일 서버

    -> 데이터베이스에 저장하기에는 비효율적이거나 서비스 제공 목적으로 유지하는 파일들을 저장하는 서버

    -> ex) AWS S3 등

**S/W 환경**

- 클라이언트 서버 운영을 위한 시스템 S/W와 개발에 사용되는 개발 S/W로 구성

- 시스템 S/W

    -> O/S, 웹 서버, WAS 운용을 위한 서버 프로그램

    -> DBMS

- 개발 S/W

    -> 요구사항 관리 도구 : 요구사항 수집과 분석, 추적 등을 편리하게 도와주는 S/W

    -> 설계 모델링 도구 : UML을 지원하며 개발의 전 과정에서 설계 및 모델링을 도와주는 S/W

    -> 구현 도구 : 개발 언어를 통해 애플리케이션의 실제 구현을 지원하는 S/W

    -> 빌드 도구 : 구현 도구를 통해 작성된 소스의 빌드 및 배포, 라이브러리 관리를 지원하는 S/W

    -> 테스트 도구 : 모듈들이 요구사항에 적합하게 구현되었는지 테스트하는 S/W

    -> 형상 관리 도구 : 버전 관리를 하여 품질 향상을 지원하는 S/W

### 서버 개발

**서버 개발의 개요**

- 웹 애플리케이션의 로직을 구현할 서버 프로그램을 제작하여 WAS에 탑재하는 것

- 서버 프로그램을 개발할 수 있도록 지원하는 프레임워크가 있음

**서버 개발 프레임워크**

- 서버 프로그램 개발 시 다양한 설정을 손쉽게 할 수 있도록 클래스나 인터페이스를 제공하는 S/W

- 뷰-모델-컨트롤러(MVC) 패턴을 기반으로 개발됨

- Spring

    -> Java 기반 프레임워크

    -> 전자정부 표준 프레임워크의 기반 기술로 사용

- Node.js

    -> Javascript 기반 프레임워크

    -> 실시간으로 입출력이 빈번한 애플리케이션에 적합

- Django

    -> Python 기반 프레임워크

    -> 컴포넌트 재사용 및 플러그인화를 기반으로 신속한 개발을 지원

- Codeigniter

    -> PHP 기반 프레임워크

    -> 인터페이스가 간편하며 서버 자원을 적게 사용

- Ruby on Rails

    -> Ruby 기반 프레임워크

    -> 테스트를 위한 웹 서버 지원

    -> 데이터베이스 작업을 단순화, 자동화시켜 신속한 개발 가능

- 프레임워크의 특성 : 모듈화, 재사용성, 확장성, 제어의 역 흐름

**서버 프로그램 구현**

- 응용 S/W와 동일하게 모듈 및 공통 모듈 개발 후 통합하는 방식으로 구현

- 모듈 개발 시 다른 모듈과의 과도한 상호작용을 배제함으로써 특정 모듈의 수정이 다른 모듈에게 영향을 미치지 않아야 함

- 모듈의 독립성은 결합도와 응집도에 의해 측정

- 공통 모듈은 여러 프로그램에서 재사용할 수 있는 모듈

### 보안 및 API

**S/W 개발 보안의 개요**

- 개발 과정에서 발생할 수 있는 보안 취약점을 최소화하여 보안 위협으로부터 안전한 S/W를 개발하기 위한 보안 활동

- 데이터의 기밀성, 무결성, 가용성을 유지해야 함

- S/W 개발 보안 가이드를 참고하여 점검해야 할 보안 항목을 점검해야 함

**S/W 개발 보안 점검 항목**

- 세션 통제

    -> 서버와 클라이언트의 연결 간 발생하는 정보를 관리

    -> 불충분한 세션 관리 또는 잘못된 세션에 의한 정보 노출

- 입력 데이터 검증 및 표현

    -> 입력 데이터에 대한 유효성 검증 체계를 갖추고 실패 시 이를 처리할 수 있도록 코딩하는 것

    -> SQL 삽입, 경로 조작 및 자원 삽입, 크로스 사이트 스크립팅

- 보안 기능

    -> 인증, 접근제어, 기밀성, 암호화 등의 기능

    -> 적절한 인증 없는 중요 기능 허용, 부적절한 인가

- 시간 및 상태

    -> 병렬 처리 시스템이나 다수의 프로세스가 동작하는 환경에서 시간과 실행 상태를 관리하여 시스템이 원활히 동작되도록 하는 것

    -> 검사 시점과 사용 시점 경쟁 조건, 무한 루프, 재귀 함수

- 에러 처리

    -> S/W 실행 중 발생할 수 있는 오류들을 사전에 정의하여 에러로 인해 발생할 수 있는 문제들을 예방

    -> 오류 메시지를 통한 정보 노출, 오류 상황 대응 부재

- 코드 오류

    -> 형 변환, 자원의 반환 등을 고려하여 코딩하는 것

    -> 널 포인터 역참조, 부적절한 자원 해제

- 캡슐화

    -> 데이터와 데이터를 처리하는 함수를 하나의 객체로 묶어 코딩하는 것

    -> 잘못된 세션에 의한 정보 노출, 제거되지 않고 남은 디버그 코드

- API 오용

    -> API를 잘못사용하거나 보안의 취약한 API를 사용하지 않도록 고려하여 코딩하는 것

    -> DNS lookup에 의존한 보안 결정

**API(Application Programming Interface)**

- 응용 프로그램 개발 시 운영체제나 프로그래밍 언어 등에 있는 라이브러리를 이용할 수 있도록 규칙 등을 정의해 놓은 인터페이스

### 배치 프로그램

**배치 프로그램의 개요**

- 사용자와의 상호 작용 없이 여러 작업들을 미리 정해진 작업을 일괄적으로 처리하는 것

- 배치 프로그램의 필수 요소 : 대용량 데이터, 자동화, 견고성, 안정성, 신뢰성, 성능

- 정기 배치 : 정해진 기간에 정기적으로 수행

- 이벤트성 배치 : 설정한 특정 조건이 충족될 때 수행

- On-Demand 배치 : 사용자 요청 시 수행

**배치 스케줄러**

- 일괄 처리 작업이 설정된 주기에 맞춰 자동으로 수행되도록 지원해주는 도구

- 잡 스케줄러라고도 함

- 스프링 배치

    -> Spring Source 사와 Accenture 사가 공동 개발한 오픈소스 프레임워크

    -> 스프링 프레임워크의 특성을 그대로 가져와 스프링의 기능을 모두 사용할 수 있음

    -> 데이터베이스나 파일의 데이터를 교환하는데 필요한 컴포넌트를 제공

    -> 로그 관리, 추적, 트랜잭션 관리, 작업 처리 통계, 작업 재시작 등의 다양한 기능 제공

    -> 구성요소 : Job, Job Launcher, Step, Job Repository

- Quatz

    -> 스프링 프레임워크로 개발되는 응용 프로그램들의 일괄 처리를 위한 다양한 기능을 제공하는 오픈소스 라이브러리

    -> 수행할 작업과 수행 시간을 관리하는 요소들을 분리하여 일괄 처리 작업에 유연성을 제공

    -> 구성요소 : Scheduler, Job, JobDetail, Trigger

### 패키지 소프트웨어

**패키지 소프트웨어의 개요**

- 기업에서 일반적으로 사용하는 여러 기능을 통합하여 제공하는 S/W

- 기업에서는 패키지 소프트웨어를 구입하여 기업 환경에 적합하게 커스터마이징하여 사용

**패키지 소프트웨어의 특징**

- 요구사항을 분석하여 업무 특성에 맞게 전용으로 개발되는 S/W와 비교하여 안정성, 라이선스, 생산성 등의 차이가 있음

- 전문 업체에 의해 품질이 검증되었고 국제 산업계 표준을 준수하고 있어 안정적인 이용 가능

- S/W에 대한 라이선스가 판매자에게 있어 시스템 구축 후 기능 추가 및 코드 재사용 등에 제약이 있음

- 개발 조직을 갖추어야 할 필요성이 없어 비용을 절감할 수 있음

- 이미 개발된 S/W를 사용하기 때문에 프로젝트 기간 단축

## [4.3 응용 SW 기초 기술 활용(1)](http://127.0.0.1:5503/112)

2020. 3. 6. 17:39

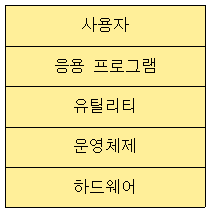
[2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [2020정처기필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0), [2020정처기필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [운영체제](http://127.0.0.1:5503/tag/%EC%9A%B4%EC%98%81%EC%B2%B4%EC%A0%9C), [저장장치관리](http://127.0.0.1:5503/tag/%EC%A0%80%EC%9E%A5%EC%9E%A5%EC%B9%98%EA%B4%80%EB%A6%AC), [주기억장치할당기법](http://127.0.0.1:5503/tag/%EC%A3%BC%EA%B8%B0%EC%96%B5%EC%9E%A5%EC%B9%98%ED%95%A0%EB%8B%B9%EA%B8%B0%EB%B2%95)

728x90

### 운영체제의 개념

**운영체제의 정의**

- 컴퓨터 시스템의 자원들을 효율적으로 관리하며, 사용자가 컴퓨터를 편리하고 효과적으로 사용할 수 있도록 환경을 제공하는 여러 프로그램의 모임



**운영체제의 목적**

- 처리 능력 향상, 사용 가능도 향상, 신뢰도 향상, 반환 시간 단축 등

- 운영체제의 성능을 평가하는 기준

    -> 처리능력 : 일정 시간 내에 시스템이 처리하는 일의 양

    -> 반환 시간(≒응답 시간) : 시스템에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸리는 시간

    -> 사용 가능도 : 시스템을 사용할 필요가 있을 때 즉시 사용 가능한 정도

    -> 신뢰도 : 주어진 문제를 정확하게 해결하는 정도

**운영체제의 기능**

- 프로세서(처리기 / Processor / CPU), 기억장치(주기억, 보조기억), 입 출력 장치, 파일 및 정보 등의 자원 관리

- 자원을 효율적으로 관리하기 위해 스케쥴링 기능 제공

- 사용자와 시스템 간 편리한 인터페이스 제공

- 각종 하드웨어와 네트워크를 관리 제어

- 데이터 관리, 데이터 및 자원의 공유 기능 제공

- 시스템의 오류 검사 및 복구

- 자원 보호 기능 제공

- 입 출력에 대한 보조 기능 제공

- 가상 계산기 기능 제공

**운영체제의 주요 자원 관리**

- 프로세스 관리 : 프로세스 스케쥴링 및 동기화 관리

- 기억장치 관리 : 프로세스에게 메모리 할당 및 회수 관리

- 주변장치 관리 : 입 출력장치 스케쥴링 및 전반적인 관리

- 파일 관리 : 파일의 생성과 삭제, 변경, 유지 등의 관리

**운영체제의 종류**

- Windows, UNIX, LINUX, MacOS, MS-DOS 등

### Windows

**Windows의 개요**

- 마이크로소프트에서 개발한 운영체제

**Windows 시스템의 특징**

- 그래픽 사용자 인터페이스(GUI, Graphci User Interface) : 키보드로 명령어를 수행하지 않고 마우스로 아이콘이나 메뉴를 선택하여 모든 작업을 수행

- 선점형 멀티태스킹 : 동시에 여러 개의 프로그램을 실행하는 멀티태스킹을 하면서 운영체제가 각 작업의 CPU 이용 시간을 제어하여 응용 프로그램 실행 중 문제가 발생하면 해당 프로그램을 강제 종료시키고 모든 시스템 자원을 반환

- Pnp(Plug and Play) : 하드웨어를 설치할 때 해당 하드웨어를 사용하는데 필요한 시스템 환경을 운영체제가 자동으로 구성해주는 기능

- OLE(Object Linking and Embedding) : 다른 응용 프로그램에서 작성된 문자나 그림 등의 개체 현재 작성 중인 문서에 자유롭게 연결하거나 삽입하여 편집할 수 있는 기능

- Single User 시스템 : 컴퓨터 한대를 한 사람이 독점 사용

### UNIX / LINUX / MacOS

**UNIX의 개요 및 특징**

- 1960년대 AT&T 벨(Bell) 연구소, MIT, General Electric이 공동 개발한 운영체제

- 시분할 시스템을 위해 설계된 대화식 운영체제

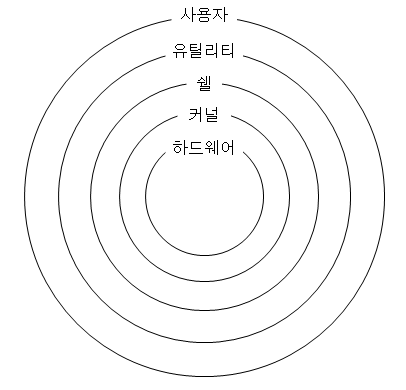
- 소스코드가 개방형 시스템로 구성되어 있음

- 대부분 C언어로 작성되어 이식성이 높으며 장치, 프로세스 간 호환성이 높음

- 다중 사용자(Multi-User) 및 다중 작업(Multi-Tasking)을 지원

- 트리 구조의 파일 시스템을 가짐

**UNIX 시스템의 구성**



- 커널(Kernel)

    -> 컴퓨터가 부팅될 때 주기억장치에 적재된 후 상주하면서 실행됨

    -> 하드웨어를 보호하고 프로그램과 하드웨어 간의 인터페이스 역할을 담당

    -> 프로세스 관리, 기억장치 관리, 파일 관리, 입출력 관리, 프로세스 간 통신, 데이터 전송 및 변환 등 여러 가지 기능 수행

- 쉘(Shell)

    -> 명령어를 인식하여 수행하는 명령어 해석기

    -> 시스템과 사용자 간의 인터페이스 담당

    -> DOS의 COMMAND.COM과 같은 기능 수행

    -> 주기억장치에 상주하지 않고 명령어가 포함된 파일 형태로 존재

    -> 보조기억장치에서 교체 처리 가능

    -> 파이프라인 기능 지원

    -> 파이프라인 : 둘 이상의 명령을 함께 묶어 처리한 결과를 다른 명령의 입력으로 전환하는 기능

    -> 입출력 재지정을 통해 입력과 출력의 방향을 변경할 수 있음

    -> 공용 Shell이나 사용자가 만들 Shell을 사용할 수 있음

- 유틸리티(Utility Program)

    -> 사용자가 작성한 외부 프로그램을 처리

    -> DOS에서의 외부 명령어에 해당

    -> 에디터, 컴파일러, 인터프리터, 디버거 등

**LINUX의 개요 및 특징**

- 1991년 리누스 토발즈가 UNIX를 기반으로 개발한 운영체제

- 프로그램 소스 코드가 무료로 공개되어 있어 사용자가 원하는 기능을 추할 수 있고 다양한 플랫폼에 설치하여 사용이 가능하여 재배포가 가능

- UNIX와 완벽하게 호환

- 대부분의 특징이 UNIX와 동일

**MacOS의 개요 및 특징**

- 1980년대 애플사가 UNIX를 기반으로 개발한 운영체제

- 애플사에서 생산하는 제품에서만 사용 가능

- 드라이버 설치 및 install / uninstall 과정이 단순

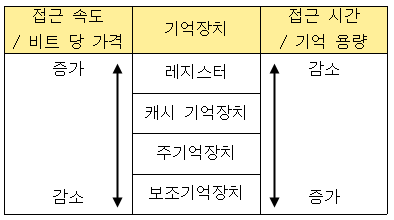
### 저장장치 관리의 개요

**저장장치 계층 구조의 특징**

- 주기억장치는 각기 자신의 주소를 가지는 워드 또는 바이트들로 구성되어 주소를 이용하여 접근

- 보조기억장치에 있는 프로그램이나 데이터는 CPU가 직접 액세스 할 수 없음

- 보조기억장치에 있는 데이터는 주기억장치에 적재된 후 CPU에 의해 액세스



**기억장치의 관리 전략의 개요**

- 3치 : Fetch(반입), 배치(Placement), 재배치(Replacement) 전략

**반입(Fetch) 전략**

- 보조기억장치에 보관중인 데이터를 언제 주기억장치에 적재할 것인지를 결정하는 전략

- 요구 반입 : 실행중인 프로그램이 특정 프로그램이나 데이터 등의 참조를 요구할 때 적재

- 예상 반입 : 실행중인 프로그램에 의해 참조될 프로그램이나 데이터를 미리 예상하여 적재

**배치(Placement) 전략**

- 새로 반입되는 데이터를 주기억장치의 어디에 위치시킬 것인지를 결정하는 전략

- 최초 적합(First Fit) : 배치가 가능한 크기의 빈 영역 중에서 첫 번째 분할 영역에 배치

- 최적 적합(Best Fit) : 배치가 가능한 크기의 빈 영역 중에서 단편화를 가장 적게 남기는 분할 영역에 배치

- 최악 적합(Worst Fit) : 배치가 가능한 크기의 빈 영역중에서 단편화를 가장 많이 남기는 분할 영역에 배치

- 단편화

    -> 내부 단편화 : 배치 후 남은 공간

    -> 외부 단편화 : 배치를 못해 빈 공간으로 남아있는 공간



**재배치(Replacement) 전략**

- 주기억장치의 모든 영역이 이미 사용중인사용 중인 상태에서 새로운 프로그램이나 데이터가 배치하려고 할 때, 이미 사용 중인 영역에서 어느 영역을 교체할 것인지를 결정하는 전략

- FIFO, OPT, LRU, LFU, NUR, SCR 등

### 주기억장치 할당 기법

**주기억장치 할당의 개념**

- 프로그램이나 데이터를 실행시키기 위해 주기억장치에 어떻게 할당할 것인지에 대한 내용

- 연속 할당 기법

    -> 프로그램을 주 기억장치에 연속으로 할당하는 기법

    -> 단일 분할 할당 기법 : 오버레이, 스와핑

    -> 다중 분할 기법 : 고정 분할 할당 기법, 동적 분할 할당 기법

- 분산 할당 기법

    -> 프로그램을 특정 단위의 조각으로 나누어 주기억장치 내에 분산하여 할당하는 기법

    -> 페이징 기법, 세그먼테이션 기법

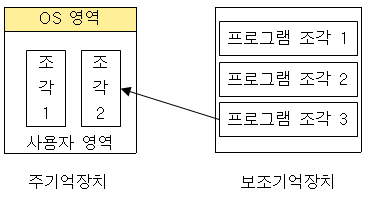
**단일 분할 할당 기법**

- 주기억장치를 운영체제 영역과 사용자 영역으로 나누어 한순간에는 오직 한 명의 사용자만이 주기억장치의 사용자 영역을 사용하는 기법

- 오버레이 기법

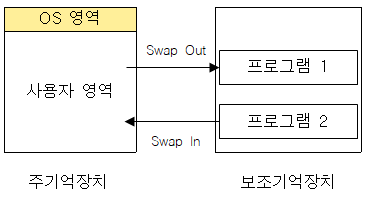
    -> 주기억장치보다 큰 사용자 프로그램을 실행하기 위한 기법

    -> 보조기억장치에 저장된 하나의 프로그램을 여러개의 조각으로 분할한 후 필요한 조각을 차례로 주기억장치에 적재하여 프로그램을 실행



- 스와핑 기법

    -> 하나의 프로그램 전체를 주기억장치에 할당하여 사용하다 필요에 따라 다른 프로그램과 교체하는 기법



**다중 분할 할당 기법**

- 고정 분할 할당 기법 : 프로그램에 할당하기 전에 운영체제가 주기억장치의 사용자 영역을 여러 개의 고정된 크기로 분할하고 준비상태 큐에서 준비 중인 프로그램을 각 영역에서 할당하여 수행하는 기법

- 가변 분할 할당 기법 : 미리 주기억장치에 분할해 놓는 것이 아닌 프로그램을 주기억장치에 적재하면서 필요한 만큼의 크기로 영역을 분할

## [4.3 응용 SW 기초 기술 활용(2)](http://127.0.0.1:5503/135)

2020. 3. 25. 16:44

[2020정보처리기사](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [2020정처기필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [정보처리기사필기정리](http://127.0.0.1:5503/tag/%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC), [정처기필기정리](http://127.0.0.1:5503/tag/%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 가상 기억장치 구현 기법 / 페이지 교체 알고리즘

**가상 기억장치의 개요**

- 보조기억장치의 일부를 주기억장치처럼 사용하는 것

- 용량이 작은 주기억장치를 마치 큰 용량을 가진 것처럼 사용할 수 있음

- 프로그램을 여러 작은 블록 단위로 나누어서 가상 기억장치에 보관해 놓고, 프로그램 실행 시 요구되는 블록만 주기억장치에 불연속적으로 할당하여 처리

- 주기억장치의 이용률과 다중 프로그래밍의 효율 상승

- 가상기억장치에 저장된 프로그램을 실행하려면 가상 기억장치의 주소를 주기억장치의 주소로 바꾸는 주소 변환(Mapping) 작업이 필요

- 연속 할당 방식에서 발생할 수 있는 단편화를 해결할 수 있음

**페이징 기법**

- 가상기억장치의 보관되어 있는 프로그램과 주기억장치의 영역을 동일한 크기로 나눈 후 나눠진 프로그램을 동일하게 나눠진 주기억장치의 영역에 적재시켜 실행하는 기법

- 프로그램을 일정한 크기로 나눈 크기를 페이지라 하고 페이지 크기로 일정하게 나누어진 주기억장치의 단위를 페이지 프레임이라고 함

- 외부 단편화는 발생하지 않으나 내부 단편화는 발생할 수 있음

- Mapping 작업을 위해서 페이지 맵 테이블이 필요함

**세그먼트 기법**

- 가상기억장치에 보관되어 있는 프로그램을 다양한 크기의 논리적인 단위로 나눈 후 주기억장치에 적재시켜 실행시키는 방법

- 프로그램을 논리적인 크기로 나눈 단위를 세그먼트라고 함

- 내부 단편화는 발생하지 않으나 외부 단편화는 발생할 수 있음

- Mapping 작업을 위해서 세그먼트 맵 테이블이 필요함

- 세그먼트가 주기억장치에 적재될 때 다른 세그먼트에게 할당된 영역을 침범할 수 없으며 이를 위해 기억장치 보호키가 필요

가상 메모리

- 고정크기 분할 : 페이지

- 가변크기 분할 : 세그먼트

ex) 5명이 펜션에 놀러갈 때

- 페이지 : 고정된 크기인 2인실 방을 3개를 주면 1명의 자리가 남게 됨 -> 내부 단편화 발생

- 세그먼트 : 5인실 방을 만들어서 주면 6명의 인원이 놀러올 때 모두가 못 들어가게 됨 -> 외부 단편화 발생

가상 메모리

- 고정크기 분할 : 페이지

- 가변크기 분할 : 세그먼트

ex) 5명이 펜션에 놀러갈 때

- 페이지 : 고정된 크기인 2인실 방을 3개를 주면 1명의 자리가 남게 됨 -> 내부 단편화 발생

- 세그먼트 : 5인실 방을 만들어서 주면 6명의 인원이 놀러올 때 모두가 못 들어가게 됨 -> 외부 단편화 발생

**페이지 교체 알고리즘**

- 페이지 부재가 발생했을 때 가상 기억장치의 필요한 페이지를 주기억장치에 적재해야 하는데 이럴 경우 주기억장치의 모든 페이지 프레임이 사용 중이면 어떤 페이지 프레임을 선택하여 교체할 것인지 결정하는 기법

- OPT(OPTimal replacement, 최적 교체)

    -> 앞으로 가장 오랫동안 사용하지 않을 페이지를 교체(예상)

- FIFO(First In First Out)

    -> 페이지가 주기억장치에 적재될 때마다 그때의 시간을 기억시켜 가장 먼저 들어와서 오래 있었던 페이지를 교체

    -> 먼저 들어온 것이 먼저 나감



- LRU(Least Recently Used)

    -> 최근에 가장 오랫동안 사용하지 않은 페이지를 교체

    -> 페이지마다 Counter나 Stack을 두어 현시점에서 가장 오래전에 사용된 페이지 교체



- LFU(Least Frequently Used)

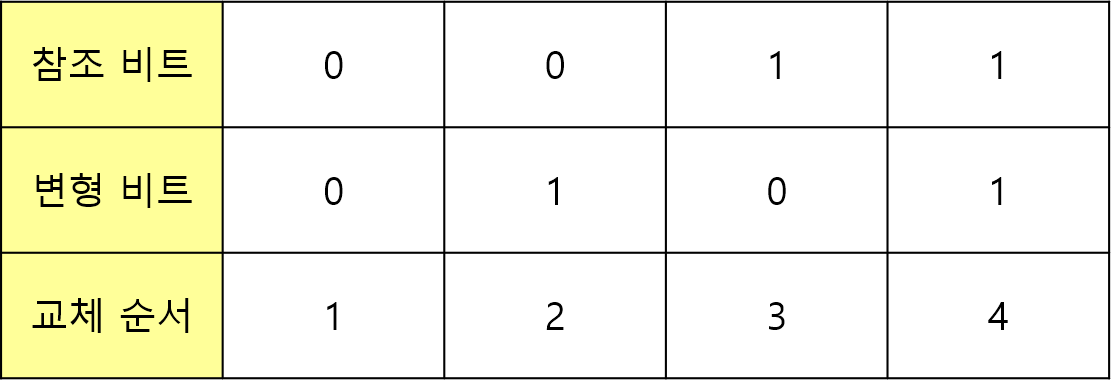
    -> 사용 빈도가 가장 적은 페이지를 교체

- NUR(Not Used Frequency)

    -> 최근에 사용하지 않은 페이지를 교체

    -> 최근 사용 여부를 확인하기 위해 페이지마다 참조 비트와 변형 비트를 사용

    -> 교체 우선순위



### 가상 기억장치 기타 관리 사항

**페이지 크기에 따른 특징**

- 페이지 크기가 작을 경우

    -> 단편화와 주기억장치로 이동하는 시간 감소

    -> 불필요한 내용이 적재될 확률이 낮아 워킹 셋이 효율적으로 유지

    -> Locality에 더욱 일치하여 기억장치에 효율 상승

    -> 페이지 맵 테이블 크기가 커지므로 매핑 속도가 늦어짐

    -> 디스크 접근 횟수가 많아져 전체적인 입출력 시간이 늘어남

- 페이지 크기가 클 경우

    -> 단편화와 주기억장치로 이동하는 시간이 증가

    -> 프로세스 수행에 불필요한 내용까지 적재될 수 있음

    -> 페이지 맵 테이블 크기가 작아지므로 매핑 속도가 빨라짐

    -> 디스크 접근 횟수가 줄어들어 전체적인 입출력 시간이 줄어듦

**Locality**

- 프로세스가 실행되는 동안 주기억장치를 참조할 때 일부 페이지만 집중적으로 참조하는 성질이 있다는 이론

- 스래싱을 방지하기 위한 워킹 셋 이론의 기반

- 프로세스가 집중적으로 사용하는 페이지를 알아내는 방법

- 시간 구역성

    -> 프로세스가 실행되면서 하나의 페이지를 일정 시간 동안 집중적으로 액세스 하는 현상

    -> 시간 구역성이 이루어지는 기억 장소 : 반복, 스택, 부프로그램, 1씩 증감, 집계에 사용되는 변수

- 공간 구역성

    -> 프로세스 실행 시 일정 위치의 페이지를 집중적으로 액세스하는 현상

    -> 공간 구역성이 이루어지는 기억 장소 : 배열 순회, 순차적 코드의 실행, 프로그래머들이 관련된 변수들을 서로 근처에 선언하여 할당되는 기억 장소

**워킹 셋**

- 프로세스가 일정 시간 동안 자주 참조하는 페이지들의 집합

- 자주 참조되는 워킹 셋을 주기억장치에 상주시켜 페이지 부재 및 페이지 교체 현상이 줄어들어 프로세스의 기억장치 사용이 안정됨

- 워킹 셋은 시간에 따라 변화

**페이지 부재 빈도 방식**

- 페이지 부재 빈도는 페이지 부재가 일어나는 횟수

- 페이지 부재율에 따라 주기억장치에 있는 페이지 프레임 수를 조정하여 적정 수준으로 유지하는 방식

- 운영체제는 프로세스 실행 초기에 임의의 페이지 프레임 할당 후 페이지 부재율에 따라 프레임을 할당하거나 회수

**프리 페이징**

- 처음의 과도한 페이지 부재를 방지하기 위해 필요할 것 같은 모든 페이지를 한꺼번에 페이지 프레임에 적재하는 기법

**스래싱**

- 프로세스의 처리 시간보다 페이지 교체에 소요되는 시간이 더 많아지는 현상

- 다중 프로그래밍 시스템이나 가상 기억장치를 사용하는 시스템에서 하나의 프로세스 수행 중 자주 페이지 부재가 발생하면서 나타나는 현상

- 전체 프로세스 선응이 저하됨

- 다중 프로그래밍의 정도가 높아짐에 따라 CPU의 이용률은 어느 특정 시점까지는 높아지지만 다중 프로그래밍의 정도가 더욱 커지면 스래싱이 나타나고 CPU의 이용률은 급격히 감소됨

- 스래싱 현상 방지 방법

    -> 다중 프로그래밍의 정도를 적정 수준으로 유지

    -> 페이지 부재 빈도를 조절하여 사용

    -> 워킹 셋 유지

    -> 부족한 자원 증설, 일부 프로세스 중단

    -> CPU 성능에 대한 자료의 지속적 관리 및 분석으로 임계치를 예상하여 운영

### 프로세스의 개요

**프로세스의 정의**

- 프로세서에 의해 처리되는 사용자 프로그램이나 시스템 프로그램을 의미

- 실행 중인 프로그램을 의미하며 작업 혹은 태스크라고도 함

- 프로세스의 또 다른 형태

    -> PCB를 가진 프로그램

    -> 실기억 장치에 저장된 프로그램

    -> 디스패치가 가능한 단위

    -> 프로시저(부 프로그램)가 활동 중인 것

    -> 비동기적 행위(다수의 프로세스가 서로 독립적으로 실행)를 일으키는 주체

    -> 지정된 결과를 얻기 위한 일련의 계통적 동작

    -> 목적 또는 결과에 따라 발생되는 사건들의 과정

    -> 운영체제가 관리하는 실행 단위

**PCB(Process Control Block)**

- 운영체제가 프로세스에 대한 중요한 정보를 저장해 놓은 곳

- 프로세스가 생성될 때마다 고유의 PCB를 생성하고 프로세스 완료 시 제거됨

- PCB에 저장되는 정보

    -> 프로세스 현재 상태

    -> 포인터

    -> 프로세스 고유 식별자

    -> 스케줄링 및 프로세스의 우선순위

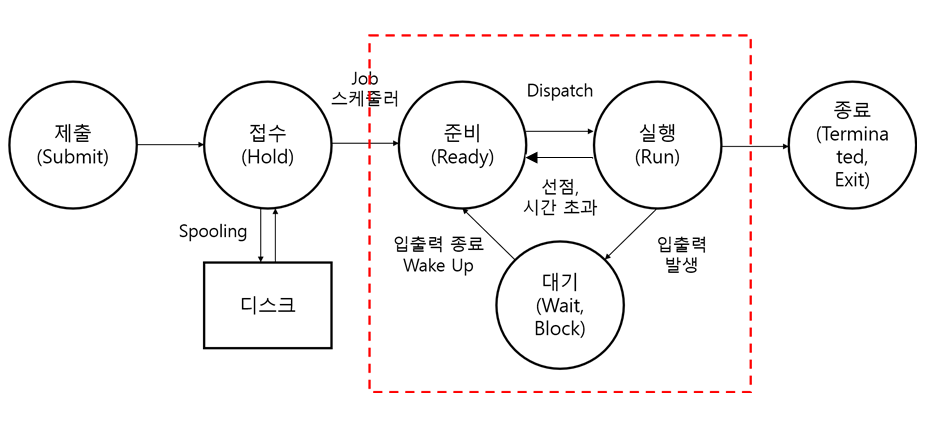
    -> CPU 레지스터 정보

    -> 주기억장치 관리 정보

    -> 입출력 상태 정보

    -> 계정 정보

**프로세스 상태 전이**



- 제출 : 작업을 처리하기 위해 사용자가 작업을 시스템에 제출한 상태

- 접수 : 제출된 작업이 스풀 공간인 디스크의 할당 위치에 저장한 상태

- 준비

    -> 프로세스가 프로세서를 할당받기 위해 기다리고 있는 상태

    -> 프로세스는 준비상태 큐에서 실행 준비

    -> Job 스케줄러에 의해 수행

- 실행

    -> 준비상태 큐에 있는 프로세스가 프로세서를 할당받아 실행되는 상태

    -> 프로세스 수행이 완료되기 전에 프로세스에게 주어진 할당 시간이 종료되면 프로세스는 준비 상태로 전이

    ->실행 중인 프로세스에 입출력 처리가 필요하면 실행중인 프로세스는 대기상태로 전이

    -> CPU 프로세스에 의해 수행

- 대기 : 프로세스에 입출력 처리가 필요하면 현재 실행중인 프로세스가 중단되고, 입출력 처리가 완료될 때까지 대기하고 있는 상태

- 종료 : 프로세스의 실행이 끝나고 프로세스 할당이 해제된 상태

**프로세스 상태 전이 관련 용어**

- Dispatch : 준비 상태에 대기하고 있는 프로세스 중 하나가 프로세스를 할당받아 실행 상태로 전이되는 과정

- Wake up : 입출력 작업이 완료되어 프로세스가 대기 상태에서 준비 상태로 전이되는 과정

- Spooling : 입출력 장치의 공유 및 상대적으로 느린 입출력장치의 처리 속도를 보완하기 위해 입출력할 데이터를 직접 장치에 보내지 않고 나중에 한꺼번에 입출력하기 위해 디스크에 저장하는 과정

- 교통량 제어기 : 프로세스의 상태에 대한 조사와 통보를 담당

**스레드(Thread)**

- 프로세스 내에서의 작업 단위로서 시스템의 여러 자원을 할당받아 실행하는 프로세스 단위

- 프로세스의 일부 특성을 가지고 있어 경량 프로세스라고도 함

- 동일 프로세스 환경에서 독립적인 다중 수행 가능

- 스레드의 분류

    -> 사용자 수준의 스레드 : 사용자가 만든 라이브러리를 사용하여 운용하기 때문에 속도는 빠르지만 구현이 어려움

    -> 커널 수준의 스레드 : 운영체제의 커널에 의해 스레드를 운영하기 때문에 속도는 느리지만 구현이 쉬움

- 스레드 사용의 장점

    -> 하나의 프로세스를 여러 개의 스레드로 생성하여 병행성을 증진할 수 있음

    -> H/W, O/S의 성능과 응용 프로그램의 처리율을 향상시킬 수 있음

    -> 응용프로그램의 응답 시간을 단축시킬 수 있음

    -> 프로세스들 간의 통신이 향상

    -> 공통적으로 접근 가능한 기억장치를 통해 효율적으로 통신

## [4.3 응용 SW 기초 기술 활용(3)](http://127.0.0.1:5503/137)

2020. 3. 26. 19:59

[2020정보처리기사](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC), [2020정보처리기사필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0), [2020정처기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0), [2020정처기필기](http://127.0.0.1:5503/tag/2020%EC%A0%95%EC%B2%98%EA%B8%B0%ED%95%84%EA%B8%B0), [정보처리기사필기](http://127.0.0.1:5503/tag/%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0)

728x90

### 스케줄링

**스케줄링의 개요**

- 프로세스가 생성되어 실행될 때 필요한 시스템의 여러 자원을 해당 프로세스에게 할당하는 작업

- 프로세스가 생성되어 완료될 때까지 여러 종류의 스케줄링 과정을 거침

- 그 종류에는 장기, 중기, 단기 스케줄링이 있음

- 문맥 교환

    -> 하나의 프로세스에서 다른 프로세스로 CPU가 할당되는 과정에서 발생되는 것

    -> 새로운 프로세스에 CPU를 할당하기 위해 현재 CPU가 할당된 프로세스의 상태 정보 저장

    -> 새로운 프로세스의 상태 정보를 설정한 후 CPU를 할당하여 실행하도록 하는 작업

**스케줄링의 목적**

- CPU나 자원을 효율적으로 사용하기 위한 정책

- 공정성 : 모든 프로세스에게 공정하게 할당

- 처리율(처리량) 증가 : 단위 시간당 프로세스를 처리하는 비율 혹은 양을 증가

- CPU 이용률 증가 : CPU의 낭비 시간을 줄이고 CPU가 순수 프로세스를 실행하는 데 사용되는 시간 비율 증가

- 우선순위 제도 : 우선순위가 높은 프로세스를 먼저 실행

- 오버헤드 최소화 : 어떤 처리를 하기 위해 들어가는 간접적인 처리시간이나 메모리를 최소화

- 응답 시간 최소화 : 작업을 지시하고 반응하기 시작하는 시간을 최소화

- 반환 시간 최소화 : 프로세스를 제출한 시간부터 실행이 완료될 때까지 걸리는 시간 최소화

- 대기 시간 최소화 : 프로세스가 준비상태 큐에서 대기하는 시간 최소화

- 균형 있는 자원의 사용

- 무한 연기 회피

**프로세스 스케줄링의 기법**

- 비선점 스케줄링

    -> 강제 종료 불가능

    -> 이미 할당된 CPU를 다른 프로세스가 강제로 빼앗아 사용할 수 없음

    -> 프로세스가 CPU를 할당받으면 완료될 때 까지 사용

    -> 프로세스 응답 시간 예측이 용이

    -> 일괄 처리 방식에 적합

    -> 중요한(짧은) 작업이 중요하지 않은(긴) 작업을 기다리는 경우가 발생할 수 있음

    -> FCFS, SJF, 우선순위, HRN, 기한부 등의 알고리즘

- 선점 알고리즘

    -> 강제 종료 가능

    -> 우선순위가 높은 다른 프로세스가 CPU를 강제로 빼앗아 사용할 수 있음

    -> 우선순위가 높은 프로세스를 빠르게 처리할 수 있음

    -> 빠른 응답 시간을 요구하는 대화식 시분할 시스템에 사용

    -> 많은 오버헤드를 초래함

    -> 선점이 가능하도록 일정 시간 배당에 대한 인터럽트용 타이머 클록이 필요

    -> Round Robin, SRT, 선점 우선순위, 다단계 큐, 다단계 피드백 큐 등의 알고리즘

### 환경변수

**환경변수의 개요**

- 시스템 소프트웨어의 동작에 영향을 미치는 동적인 값들의 모임

**Windows 환경 변수**

- Windows에서 환경 변수를 명령어나 스크립트에서 사용하기 위해서는 변수명 앞뒤에 %를 붙여야 함

**UNIX / LINUX 환경 변수**

- UNIX나 LINUX에서 환경 변수를 사용하기 위해서는 변수명 앞에 $를 붙여야 함

### 운영체제 기본 명령어

**운영체제 기본 명령어의 개요**

- CLI(Command Line Interface) : 키보드로 명령어를 입력하여 작업을 수행하는 인터페이스

- GUI(Graphic User Interface) : 마우스로 아이콘이나 메뉴를 선택하여 작업을 수행하는 인터페이스

**Windows & UNIX / LINUX 기본 명령어**



## [4.3 응용 SW 기초 기술 활용(4)](http://127.0.0.1:5503/138)

2020. 3. 26. 22:41

728x90

### 인터넷

**인터넷의 개요**

TCP/IP 프로토콜을 기반으로 전 세계 수많은 컴퓨터와 네트워크들이 연결된 광범위한 컴퓨터 통신망

- 미 국방성의 ARPANET에서 시작

- 유닉스 운영체제를 기반으로 함

- 인터넷에 연결된 컴퓨터는 고유한 IP 주소를 가짐

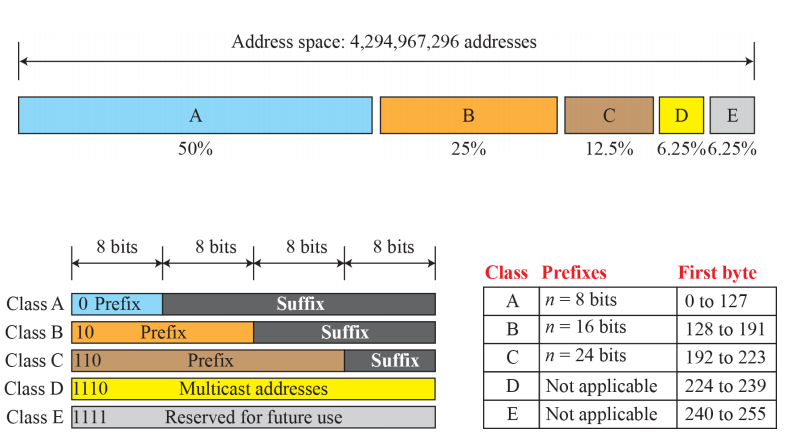
- 인터넷을 구성하기 위해서는 브리지, 라우터, 게이트웨이가 사용됨

- 백본 : 네트워크를 연결하여 중추적 역할을 하는 네트워크로 인터넷의 주가 되는 기간망을 일컫는 용어

**IP주소(Internet Protocol Address)**

- 인터넷에 연결된 컴퓨터를 구분하기 위한 고유한 주소

- 8비트 씩 4부분으로 구성됨(IPv4)

데이터 통신과 네트워킹 5판, 18강 강의자료

**서브네팅**

- 할당된 네트워크 주소를 다시 여러 개의 작은 네트워크로 나누어 사용

- IPv4의 주소 부족 문제를 해결하기 위한 방법

- 서브넷 마스크 : 네트워크 주소와 호스트 주소를 구분하기 위한 비트

**IPv6**

- IPv4의 주소 부족 문제를 해결하기 위해 개발됨

- 128비트의 긴 주소를 사용하여 주소 부족 문제를 해결할 수 있고, 자료 전송 속도가 빠름

- 인증성, 기밀성, 데이터 무결성의 지원으로 보안 문제 해결 가능

    -> 인증성 : 사용자의 식별과 접근 권한 검증

    -> 기밀성 : 시스템 내의 정보와 자료는 인가된 사용자에게만 접근 허용

    -> 무결성 : 시스템 내의 정보는 인가된 사용자만 수정 가능

- Traffic Class, Flow Label을 이용하여 등급별, 서비스별로 패킷을 구분할 수 있어 품질 보장이 용이

    -> Traffic Class : IPv6 패킷의 클래스나 우선순위를 나타내는 필드

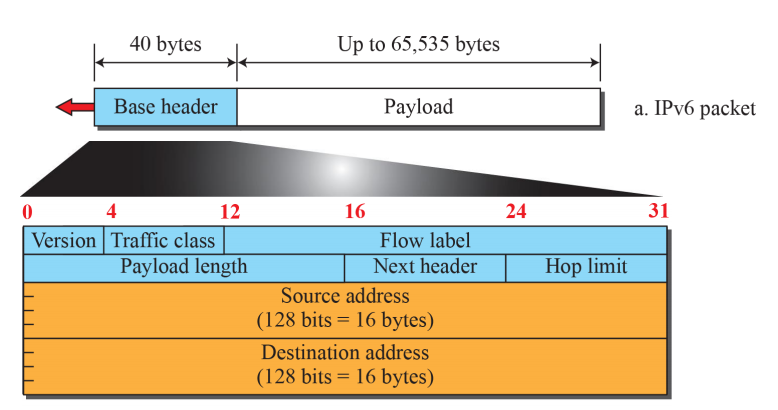
    -> Flow Label : 네트워크 상에서 패킷들의 흐름에 대한 특성을 나타내는 필드

- 주소 체계

    -> 유니캐스트 : 1:1 통신

    -> 멀티캐스트 : 1:N 통신

    -> 애니캐스트 : 1:1 통신, 유니캐스트와 다르게 송신자와 가장 가까이 있는 수신자 간의 통신

참고용) IPv6 데이터그램

**도메인 네임**

- IP 주소 사람이 이해하기 쉬운 문자 형태로 표현한 것

ex) http://210.89.164.90/(IP주소) -> www.naver.com (도메인 네임)

- 도메인 네임의 구성



- 도메인 네임을 IP 주소로 변환하는 역할을 하는 시스템을 DNS라고 하며 이런 역할을 하는 서버를 DNS 서버라고 함

### OSI 참조 모델

**OSI(Open System Interconnetion) 참조 모델의 개요**

- 다른 시스템 간의 원활한 통신을 위해 ISO(국제표준화기구)에서 제안한 통신 규약(프로토콜)

출처 : http://www.a24s.com/data/jeongbotongsinhakseub/jbts/contents/ch2-4.htm

- Application / Presentation / Session / Transport / Network / Datalink / Physical => 아파서티난다 피

**OSI 모델의 계층**

- 물리 계층

    -> 전송에 필요한 두 장치 간의 실제 접속과 절단 등 기계적, 전기적, 기능적, 절차적 특성에 대한 규칙을 정의

    -> 물리적 전송 매체와 신호 방식을 정의

    -> RS-232C, X.21 등의 표준

    -> 관련 장비 : 리피터, 허브

    -> 데이터 단위 : 비트

- 데이터 링크 계층

    -> 두 개의 인접 개방 시스템들 간의 신뢰성 있고 효율적인 정보 전송을 할 수 있도록 함

    -> 주요 기능 : 흐름 제어, 프레임 동기화, 오류 제어, 순서 제어

    -> HDLC, LAPB, LLC, MAC, LAPD, PPP 등의 표준

    -> 관련 장비 : 브리지

    -> 데이터 단위 : 프레임

- 네트워크 계층

    -> 개방 시스템들 간의 네트워크 연결을 관리하고 데이터의 교환 및 중계 기능

    -> 네트워크 연결 설정, 유지, 해제

    -> 경로 설정, 데이터 교환 및 중계, 트래픽 제어, 패킷 정보 전송 수행

    -> X.25, IP 등의 표준

    -> 관련 장비 : 라우터

    -> 데이터 단위 : 패킷

- 전송 계층

    -> 논리적 안정과 균일한 데이터 전송 서비스를 제공

    -> 종단 시스템 간의 투명한 데이터 전송을 가능하게 함

    -> 종단 시스템 간의 전송 연결 설정, 데이터 전송, 연결 해제 기능

    -> 주소 설정, 다중화, 오류 제어, 흐름 제어

    -> TCP, UDP 등의 표준

    -> 관련 장비 : 게이트웨이

    -> 데이터 단위 : 세그먼트

- 세션 계층

    -> 송수신 측 간의 관련성을 유지하고 대화 제어를 담당

    -> 동기점 : 송수신 측간의 대화 동기를 위해 전송하는 정보를 일정한 부분에 두어 정보의 수신 상태를 체크하는 포인트

    -> 데이터 단위 : 메시지

 - 표현 계층

    -> 데이터를 응용 계층, 세션 계층에 보내기 전에 계층에 맞게 변환

    -> 서로 다른 데이터 표현 형태를 갖는 시스템 간 상호 접속을 위해 필요한 계층

    -> 코드 변환, 데이터 암호화, 데이터 압축, 구문 검색, 정보 형식 변환, 문맥 관리 기능

    -> 데이터 단위 : 메시지

- 응용 계층

    -> 사용자가 OSI 환경에 접속할 수 있도록 서비스 제공

    -> 프로세스 간의 정보 교환, 전자 사서함, 가상 터미널 등의 서비스 제공

### 네트워크 관련 장비

**네트워크 인터페이스 카드(Network Interface Card)**

- 컴퓨터를 연결하는 장치로 정보 전송 시 정보가 케이블을 통해 전송될 수 있도록 정보 형태를 변경

- 이더넷 카드 혹은 네트워크 어댑터라고도 함

**허브(hub)**

- 가까운 거리의 컴퓨터를 연결하는 장치

- 각 회선을 통합적으로 관리하며 신호 증폭 기능을 하는 리피터의 역할도 포함

- 더미 허브

    -> 네트워크에 흐르는 모든 데이터를 단순히 연결만 함

    -> LAN이 보유한 대역폭을 나누어 제공

- 스위칭 허브

    -> 네트워크상에 흐르는 데이터의 유무 및 흐름을 제어하여 각각의 노드가 허브의 최대 대역폭을 사용할 수 있는 지능형 허브

**리피터(Repeater)**

- 전송되는 신호가 원래의 형태와 다르게 왜곡되거나 약해질 경우 원래의 신호 형태로 재생하여 다시 전송하는 역할

- 근접한 네트워크 사이에 신호를 전송

- 전송 거리의 연장 또는 배선의 자유도를 높이는 용도

**브리지(Bridge)**

- LAN과 LAN을 연결하거나 LAN안에서 컴퓨터 그룹을 연결

- 데이터 링크 계층 중 MAC 계층에서 사용되므로 MAC 브리지라고도 함

**스위치(Switch)**

- LAN과 LAN을 연결하여 훨씬 더 큰 LAN을 만드는 장치

- 포트마다 각기 다른 전송속도를 지원하도록 제어할 수 있음

- 수십 ~ 수백 개의 포트를 지원

**라우터(Router)**

- LAN과 LAN의 연결 기능에 데이터 전송의 최적 경로를 선택할 수 있는 기능을 추가

- 서로 다른 LAN이나 LAN과 WAN의 연결도 수행

- 접속 가능한 경로에 대한 정보를 Routing Table에 저장하여 보관

**게이트웨이(Gateway)**

- 전 계층의 프로토콜 구조가 다른 네트워크의 연결을 수행

- LAN에서 다른 네트워크에 데이터를 송수신하는 출입구 역할을 함

### 프로토콜의 개념

**프로토콜의 정의**

- 서로 다른 기기들 간의 데이터 교환을 원활하게 수행할 수 있도록 표준화시켜 놓은 통신 규약

**프로토콜의 기본 요소**

- 구문 : 전송하고자 하는 데이터의 형식, 부호화, 신호 레벨 등을 규정

- 의미 : 두 기기 간의 효율적이고 정확한 정보 전송을 위한 협조 사항과 오류 관리를 위한 제어 정보를 규정

- 시간 : 두 기기 간의 통신 속도, 메시지의 순서 제어 등을 규정

**프로토콜의 기능**

- 단편화와 재결합

    -> 단편화 : 송신 측에서 전송할 데이터를 전송에 알맞은 작은 크기의 블록으로 자르는 작업

    -> 재결합 : 수신 측에서 수신한 단편화된 데이터를 다시 모으는 작업

- 캡슐화

    -> 단편화 된 데이터에 주소, 오류 검출 코드, 프로토콜 제어 정보를 부가하는 것

- 흐름 제어

    -> 수신 측에서 송신 측의 데이터 전송 속도나 전송 량을 제어할 수 있는 기능

    -> 정지-대기 방식이나 슬라이딩 윈도우 방식을 이용

- 오류 제어

    -> 전송 중에 발생하는 오류를 검출하고 정정하여 데이터나 제어 정보의 파손에 대비하는 기능

- 동기화

    -> 송수신 측이 같은 상태를 유지하도록 타이밍을 맞추는 기능

- 순서 제어

    -> 전송되는 데이터 블록에 전송 순서를 부여하여 연결 위주의 데이터 전송 방식에 사용

    -> 흐름 제어 및 오류 제어를 용이하게 함

- 주소 지정

    -> 데이터가 목적지까지 정확하게 전송될 수 있도록 목적지 이름, 주소, 경로를 부여하는 기능

- 다중화

    -> 한 개의 통신 회선을 여러 가입자들이 동시에 사용하도록 하는 기능

- 경로 제어

    -> 송수신 측간의 송신 경로 중에서 최적의 패킷 교환 경로를 설정하는 기능

- 전송 서비스

    -> 전송하려는 데이터가 사용하도록 하는 별도의 부가 서비스

### TCP/IP

**TCP/IP의 개요**

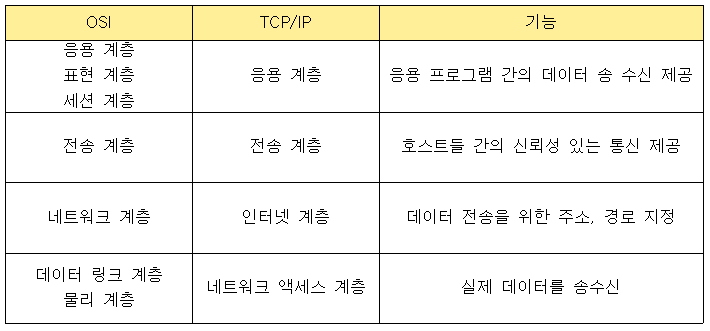
- 인터넷에 연결된 서로 다른 기종의 컴퓨터들이 데이터를 주고받을 수 있도록 하는 표준 프로토콜

- 1960년대 말 ARPA에서 개발하여 ARPANET에서 사용하기 시작

- UNIX의 기본 프로토콜로 사용되었다가 현재는 인터넷 범용 프로토콜로 사용

- TCP(Transmission Control Protocol)과 IP(Internet Protocol)이 결합

**TCP/IP의 구조**



**응용 계층의 주요 프로토콜**

- FTP : 원격 파일 전송 프로토콜

- SMTP : 전자 우편 교환 서비스

- TELNET :

    -> 원격 접속 서비스

    -> 가상 터미널 기능 수행

- SNMP :

    -> TCP/IP의 관리 프로토콜

    -> 네트워크 기기의 네트워크 정보를 네트워크 관리 시스템에 보내는 데 사용되는 프로토콜

- DNS : 도메인 네임을 IP주소로 매핑하는 시스템

- HTTP : WWW에서 HTML을 송수신하기 위한 표준 프로토콜

**전송 계층의 주요 프로토콜**

- TCP

    -> 양방향 연결형 서비스 제공

    -> 가상 회선 연결 형태의 서비스 제공

    -> 순서 제어, 오류 제어, 흐름 제어 기능을 함

    -> 스트림 위주의 패킷 단위 전달

- UDP

    -> 비연결형 서비스 제공

    -> 실시간 전송에 유리하며, 신뢰성보다는 속도가 중요시되는 네트워크에서 사용

- RTCP

    -> 패킷의 전송 품질을 제어하기 위한 제어 프로토콜

    -> 세션에 참여한 각 참여자들에게 주기적으로 제어 정보를 전송

    -> 데이터 전송을 모니터링하고 최소한의 제어와 인증 기능만을 제공

    -> 패킷은 항상 32비트의 경계로 끝남

**인터넷 계층의 주요 프로토콜**

- IP : 전송할 데이터에 주소를 지정하고 경로를 설정

- ICMP

    -> IP와 조합하여 통신 중에 발생하는 오류의 처리와 전송 경로 변경 등을 위한 제어 메시지를 관리

    -> 헤더는 8Byte로 구성

- IGMP : 멀티캐스트를 지원하는 호스트나 라우터 사이에서 멀티캐스트 그룹 유지를 위해 사용

- ARP : IP 주소를 MAC Address로 변환 (논리 주소 → 물리 주소)

- RARP : ARP의 반대로 MAC Address를 IP 주소로 변환 (물리 주소 → 논리 주소)

**네트워크 액세스 계층의 주요 프로토콜**

- IEEE 802 : LAN을 위한 표준 프로토콜

    -> IEEE 802.3(Ethernet) : CSMA/CD 방식의 LAN

    -> IEEE 802.4 : 토큰 박스

    -> IEEE 802.5 : 토큰링

    -> IEEE 802.11 : 무선 LAN

- HDLC : 비트 위주의 데이터 링크 제어 프로토콜

- X.25 : 패킷 교환망을 통한 DTE와 DCE 간의 인터페이스를 제공하는 프로토콜

    -> DTE(신호 단말 장치), DCE(신호 통신 장비)

- RS-232C : 공중전화 교환망을 통한 DTE와 DCE 간의 인터페이스를 제공하는 프로토콜

## [5.1 소프트웨어 개발 방법론 활용(1)](http://127.0.0.1:5503/164)

2020. 4. 16. 18:31

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 소프트웨어 개발 방법론

**소프트웨어 개발 방법론의 개요**

- 소프트웨어 개발, 유지보수에 필요한 수행 방법과 효율적으로 수행하려는 과정에서 필요한 기법 및 도구를 정리하여 표준화

**구조적 방법론**

- 정형화된 분석 절차에 따라 사용자 요구사항을 파악하여 문서화하는 처리 중심의 방법론

- Divide and Conquer 원리 적용

https://blog.kakaocdn.net/dn/bDJfwm/btqDuq8ArKY/SxVNAlkkhjhsEasI7Nw691/img.png

**정보공학 방법론**

- 정보 시스템 개발을 위해 계획, 분석, 설계, 구축에 정형화된 기법들을 상호 연관성 있게 통합 및 적용하는 자료 중심의 방법론

- 대규모 정보시스템 구축에 적합

https://blog.kakaocdn.net/dn/tKz2I/btqDtA4DU6w/b7hkt3ksSIG3XuOLhlRN00/img.png

**객체지향 방법론**

- 기계의 부품을 조립하듯이 객체들을 조립하여 소프트웨어를 구현하는 방법론

- 구조적 기법의 해결책으로 채택

https://blog.kakaocdn.net/dn/bFkLcZ/btqDvXScLG3/LCApxJv0HOkA98aKDNXShk/img.png

- [객체 지향](https://1d1cblog.tistory.com/145)

불러오는 중입니다...

**컴포넌트 기반(CBD) 방법론**

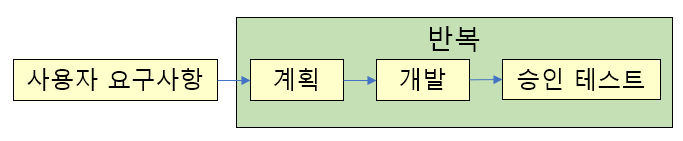
- 컴포넌트를 조합하여 새로운 애플리케이션을 만드는 방법론

- 컴포넌트의 재사용이 가능하여 시간, 노력, 비용을 절감하고 품질을 높임

https://blog.kakaocdn.net/dn/H8pn4/btqDwdUMh0o/MxHQTALefgVxSgF9FSETG0/img.png

**애자일 방법론**

- 고객의 요구사항 변화에 빠르고 유연하게 대응하도록 일정한 주기를 반복하면서 개발 과정을 진행하는 방법론



- [애자일, XP, 스크럼](https://1d1cblog.tistory.com/141)

[2020 정보처리기사 필기 - 1.1 요구사항 확인(1)](https://1d1cblog.tistory.com/141" \t "_blank)

[소프트웨어 생명 주기 소프트웨어 생명 주기(Life Cycle) - 소프트웨어 개발 방법론의 바탕이 되어 소프트웨어를 개발하기 위해 정의하고 운용 유지보수 등의 과정을 각 단계별로 나눈 것 - 스프트웨어 개발 단계..](https://1d1cblog.tistory.com/141" \t "_blank)

[1d1cblog.tistory.com](https://1d1cblog.tistory.com/141" \t "_blank)

**제품 계열 방법론**

- 특정 제품에 적용하고 싶은 공통된 기능을 정의하여 개발하는 방법론

- 임베디드 소프트웨어 개발에 적합

### 비용 산정 기법

**소프트웨어 비용 산정의 개요**

- 소프트웨어의 개발 규모를 소요되는 인원, 자원, 기간 등으로 확인하여 실행 가능한 계획을 수립하기 위해 필요한 비용을 산정하는 것

**소프트웨어 비용 결정 요소**

- 프로젝트 요소 : 제품 복잡도, 시스템 크기, 요구되는 신뢰도

- 자원 요소 : 인적 자원, 하드웨어 자원, 소프트웨어 자원

- 생산성 요소 : 개발자 능력, 개발 기간

**하향식 산정 기법**

- 과거의 유사한 경험을 바탕으로 전문 지식이 많은 개발자들이 참여한 회의를 통해 비용을 산정

- 전문가 감정 기법

    -> 조직 내 경험이 많은 두 명 이상의 전문가에게 비용 산정을 의뢰

    -> 진행했던 유사한 프로젝트와 진행할 새로운 프로젝트 간 새로운 요소가 있을 수 있고 경험이 없을 수 있음

- 델파이 기법

    -> 많은 전문가의 여러 의견을 종합하여 산정

    -> 한 명의 조정자와 여러 전문가로 구성되어 객관적임

**상향식 산정 기법**

- 프로젝트의 세부적인 작업 단위별로 비용을 산정 후 집계하여 전체 비용을 산정

- LOC(source Line Of Code) 기법

    -> 소프트웨어의 각 기능의 원시 코드 라인 수로 예측치를 구하고 비용을 산정하는 기법

    -> 노력(인월) = 개발 기간 x 투입 인원 = LOC / 1인당 월평균 생산 코드 라인 수

    -> 개발 비용 = 노력(인월) x 단위 비용(1인당 월평균 인건비)

    -> 개발 기간 = 노력(인월) / 투입 인원

    -> 생산성 = LOC / 노력(인월)

- 개발 단계별 인월수 기법

    -> 각 기능을 구현시키는 데 필요한 노력을 생명 주기의 각 단계별로 산정

### 수학적 산정 기법

**수학적 산정 기법의 개요**

- 상향식 선정 기법

- 개발 비용 산정의 자동화를 목표로 함

- 경험적 / 실험적 추정 모형이라고도 함

- 과거 유사한 프로젝트를 기반으로 공식을 유도

**COCOMO 모형**

- 보헴이 제안하였으며 LOC에 의한 비용 산정 기법

- 개발 유형 : 소프트웨어의 복잡도 또는 원시 프로그램의 규모에 따라 분류

    -> 조직형(Organinc Mode) : 기관 내부에서 개발된 중·소규모의 소프트웨어로 5만 라인 이하의 소프트웨어를 개발하는 유형

    -> 반분리형(Semi-Detached Mode) : 트랜잭션 처리 시스템이나 운영체제, 데이터베이스 관리 시스템 등 30만 라인 이하의 소프트웨어를 개발하는 유형

    -> 내장형(Embedded Mode) : 최대형 규모의 트랜잭션 처리 시스템이나 운영체제 등 30만 라인 이상의 소프트웨어를 개발하는 유형

- 모형의 종류 : 비용 산정 단계 및 적용 변수의 구체화 정도로 구분

    -> 기본형(Basic) : 소프트웨어의 크기와 개발 유형만을 이용하여 비용을 산정

    -> 중간형(Intermediate) : 기본형의 공식을 사용하거나 4가지 특성의 15가지 요인에 의해 비용을 산정

        \* 제품의 특성 : 요구되는 신뢰도, 데이터베이스 크기, 제품의 복잡도

        \* 컴퓨터의 특성 : 수행 시간의 제한, 기억 장소의 제한, 가상 기계의 안정성, 반환 시간

        \* 개발 요원의 특성 : 분석가의 능력, 개발 분야의 경험, 가상 기계의 경험, 프로그래머의 능력, 프로그래밍 언어의 경험

        \* 프로젝트 특성 : 소프트웨어 도구의 이용, 프로젝트 개발 일정, 최신 프로그래밍 기법의 이용

    -> 발전형(Detailed) : 개발 공정별로 보다 자세하고 정확하게 노력을 산출하여 비용을 산정

**Putnam 모형**

- 소프트웨어 생명 주기의 전 과정 동안에 사용될 노력의 분포를 가정해주는 모형

- 생명 주기 예측 모형이라고도 함

- 시간에 따라 함수로 표현되는 Rayleigh-Norden 곡선의 노력 분포도를 기초로 함

- 대형 프로젝트 노력 분포 산정에 이용

- SLIM : Rayleigh-Norder 곡선과 Putnam 예측 모델을 기초로 하여 개발된 자동화 측정 도구

**기능 점수(FP) 모형**

- 소프트웨어 기능을 증대시키는 요인별로 가중치를 부여하고 합산하여 총 기능 점수를 산출하며 총 기능 점수와 영향도를 이용하여 기능 점수를 구한 후 이를 이용해서 비용을 산정

- 알브레히트가 제안

## [5.2 IT 프로젝트 정보 시스템 구축 관리(1)](http://127.0.0.1:5503/166)

2020. 4. 17. 11:23

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 신기술 관련 용어

**네트워크 관련 신기술 용어**

- IoT(Internet of Things, 사물인터넷) : 정보 통신 기술을 기반으로 실세계와 가상 세계의 다양한 사물과 사람을 인터넷으로 소로 연결하여 진보된 서비스를 제공하기 위한 서비스 기반 기술

- M2M(Machine to Machine, 사물 통신) : 무선 통신을 이용한 기계와 기계 사이의 통신

- 모바일 컴퓨팅 : 휴대형 기기로 이동하면서 자유롭게 네트워크에 접속하여 업무를 처리할 수 있는 환경

- 클라우딩 컴퓨팅 : 각종 컴퓨터 자원을 중앙 컴퓨터에 두고 인터넷 기능을 갖는 단말기로 언제 어디서나 인터넷을 통해 컴퓨터 작업을 수행할 수 있는 환경

- 모바일 클라우딩 컴퓨팅 : 모바일 컴퓨팅과 클라우딩 컴퓨팅을 혼합하여 클라우드 서비스를 이용하여 모바일 기기로 클라우딩 컴퓨팅 인프라를 구성하여 여러 가지 정보와 자원을 공유하는 ICT 기술

- 인터클라우드 컴퓨팅 : 각기 다른 클라우드 서비스를 연동하거나 컴퓨팅 자원의 동적 할당이 가능하도록 여러 클라우드 서비스 제공자들이 제공하는 클라우드 서비스나 자원을 연결하는 기술

- 매시 네트워크 : 차세대 이통통신, 홈네트워킹, 공공 안전 등 특수 목적을 위한 새로운 방식의 네트워크 기술로 대규모 디바이스의 네트워크 생성에 최적화

- 와이선 : 장거리 무선 통신을 필요로 하는 사물인터넷 서비스를 위한 저전력 장거리 통신 기술

- NDN(Named Data Networking) : 콘텐츠 자체의 정보와 라우터 기능만으로 데이터 전송을 수행하는 기술

- NGN(Next Generation Network, 차세대 통신망) : 유선망 기반의 차세대 통신망으로 유선망뿐만 아니라 이동 사용자를 목표로 하며 이동 통신에서 제공하는 완전하 이동성 제공을 목표로 개발

- SDN(Software Defined Networking) : 네트워크를 컴퓨터처럼 모델링하여 여러 사용자가 각각의 소프트웨어들로 네트워킹을 가상화하여 제어하고 관리하는 네트워크

- NFC(Near Field Communication) : 고주파를 이용한 근거리 무선 통신 기술

- UWB(Ultra WideBand) : 짧은 거리에서 많은 양의 디지털 데이터를 낮은 전력으로 전송하기 위한 무선 기술로 무선 디지털 펄스라고도 함

- 피코넷 : 여러 개의 독립된 통신 장치가 블루투스 기술이나 UWB 통신 기술을 사용하여 통신망을 형성하는 무선 네트워크 기술

- WBAN(Wireless Body Area Network) : 웨어러블 또는 몸에 심는 형태의 센서나 기기를 무선으로 연결하는 개인 영역 네트워킹 기술

- GIS(Geographic Information System) : 지리적인 자료를 수집, 저장, 분석, 출력할 수 있는 컴퓨터 응용 시스템으로 위성을 이용해 모든 사물의 위치 정보를 제공해줌

- USN(Ubiquitous Sensor Network) : 각종 센서로 수집한 정보를 무선으로 수집할 수 있도록 구성하는 네트워크

- SON(Self Organizing Network) : 주변 상황에 맞추어 스스로 망을 구성하는 네트워크

- 애드 혹 네트워크 : 재난 현장과 같이 별도의 고정된 유선망을 구축할 수 없는 장소에서 모바일 호스트만을 이용하여 구성한 네트워크

- 네트워크 슬라이싱 : 여러 글로벌 이동통신 표준화 단체가 선정한 5G의 핵심기술 중 하나로 네트워크에서 하나의 물리적인 코어 네트워크 인프라를 독립된 다수의 가상 네트워크로 분리하여 각각의 네트워크를 통해 다양한 고객 맞춤형 서비스를 제공하는 것을 목적으로 하는 네트워크 기술

- 저전력 블루투스 기술 : 일반 블루투스와 동일한 주파수 대역을 사용하지만 연결되지 않은 상태에서는 절전 모드를 유지하는 기술

- 지능형 초연결망 : 스마트 시티, 스마트 스테이션 등 4차 산업혁명 시대를 맞아 새로운 변화에 따라 급격하게 증가흐는 데이터 트래픽을 효과적으로 수용하기 위해 시행되는 정부 주관 사업

**SW 관련 신기술 용어**

- 인공지능 : 인간의 두뇌와 같이 컴퓨터 스스로 추론, 학습, 판단 등 인간 지능적인 작업을 수행하는 시스템

- 뉴럴링크 : 테슬라의 CEO 일론 머스크가 사람의 뇌와 컴퓨터와 결합하는 기술을 개발하기 위해 설립한 회사

- 딥 러닝 : 인간의 두뇌를 모델로 만들어진 인공 신경망을 기반으로 하는 기계 학습 기술

- 전문가 시스템 : 특정 분야의 전문가가 수행하는 고도의 업무를 지원하기 위한 컴퓨터 응용 프로그램

- 증강현실 : 실제 촬영한 화면에 가상의 정보를 부가하여 보여주는 기술

- 블록체인 : P2P 네트워크를 이용하여 온라인 금융 거래 정보를 온라인 네트워크 참여자의 디지털 장비에 분산 저장하는 기술

- 분산 원장 기술 : 중앙 관리자나 중앙 데이터 저장소가 존재하지 않고 P2P 망내의 참여자들에게 모든 거래 목록이 분산 저장되어 거래가 발생할 때마다 지속적으로 갱신되는 디지털 원장

- 해시 : 임의의 길이의 입력 데이터나 메시지를 고정된 길이의 값이나 키로 변환

- 양자 암호키 분배 : 양자 통신을 위해 비밀키를 분배하여 관리하는 기술

- 프라이버시 강화 기술 : 개인 정보 위험 관리 기술

- 공통 평가 기준 : ISO 15408 표준으로 채택된 정보 보호 제품 평가 기준

- 개인정보 영향평가 제도 : 개인정보를 활용하는 새로운 정보 시스템의 도입 및 기존 정보시스템의 중요한 변경 시 시스템의 구축, 운영이 기업의 고객은 물론 국민의 사생활에 미칠 영향에 대해 미리 조사, 분석, 평가하는 제도

- 그레이웨어 : 소프트웨어를 제공하는 입장에서는 악의적이지 않은 유용한 소프트웨어 일지라도 사용자 입장에서는 유용할 수도 있고 악의적일 수도 있는 악성 코드나 공유 웨어

- 매시업 : 웹에서 제공하는 정보 및 서비스를 이용하여 새로운 소프트웨어나 서비스 데이터베이스 등을 만드는 기술

- 리치 인터넷 애플리케이션 : 플래스 애니메이션 기술과 웹 서버 애플리케이션 기술을 통합하여 기존 HTML보다 역동적인 웹페이지를 제공하는 플래시 웹페이지 제작 기술

- 시맨틱 웹 : 컴퓨터가 사람을 대신하여 정보를 읽고 이해하고 가공하여 새로운 정보를 만들어 낼 수 있도록 이해하기 쉬운 의미를 가진 차세대 지능형 웹

- 증발품 : 판매 계획 또는 배포 계획은 발표되었으나 실제로 고객에게는 판매되거나 배포하지 않고 있는 소프트웨어

- 오픈 그리드 서비스 아키텍처 : 애플리케이션 공유를 위한 웹 서비스를 그리드 상에서 제공하기 위해 만든 개방형 표준

- 서비스 지향 아키텍처 : 기업의 소프트웨어 인프라인 정보시스템을 공유와 재사용이 가능한 서비스 단위나 컴포넌트 중신으로 구축하는 정보기술 아키텍처

- 서비스형 소프트웨어(SaaS) : 소프트웨어의 여러 기능 중에서 사용자가 필요로 하는 서비스만 이용할 수 있도록 한 소프트웨어

- 소프트웨어 에스크로 : 소프트웨어 개발자의 지적재산권을 보호하고 사용자는 저렴한 비용으로 소프트웨어를 안정적으로 사용 및 유지보수받을 수 있도록 소스 프로그램과 기술 정보 등을 제3의 기관에 보관하는 것

- 복잡 이벤트 처리 : 실시간으로 발생하는 많은 사건들 중 의미가 있는 것만을 추출할 수 있도록 사건 발생 조건을 정의하는 데이터 처리 방법

- 디지털 트윈 : 현실속의 사물을 소프트웨어로 가상화한 모델

**HW 관련 신기술 용어**

- 고가용성 : 긴 시간동안 안정적인 서비스 운영을 위해 장애 즉시 다른 시스템으로 대체 가능한 환경을 구축하는 메커니즘

- 3D 프린팅 : 평면에 출력하는 것이 아닌 얇은 두께로 한층 한층 적재시켜 하나의 형태를 만들어내는 기술

- 4D 프린팅 : 특정 시간이나 환경 조건이 갖추어지면 스스로 형태를 변화시키거나 제조되는 자가 조립 기술이 적용된 제품을 3D 프린팅 하는 기술

- RAID : 여러 개의 하드디스크로 디스크 배열을 구성하여 파일을 구성하여 파일을 구성하고 있는 데이터 블록들을 서로 다른 디스크들에 분산 저장하는 기술

- 앤 스크린 : N개의 서로 다른 단말기에서 동일한 콘텐츠를 자유롭게 이용할 수 있는 서비스

- 컴패니언 스크린 : TV 방송 시청 시 방송 내용을 공유하며 추가적인 기능을 수행할 수 있는 디바이스

- 씬(Thin) 클라이언트 PC : 하드디스크나 주변 장치 없이 기본적인 메모리만 갖추고 서버와 네트워크로 운용되는 개인용 컴퓨터

- 멤스 : 초정밀 반도체 기술을 바탕으로 센서나 액추에이터 등 기계 구조를 다양한 기술로 미세 가공하여 전기기계적 동작을 할 수 있도록 한 초미세 장치

- 패블릿 : 태블릿 기능을 포함한 5인치 이상의 대화면 스마트폰

- 트러스트존 기술 : ARM에서 개발한 하나의 프로세서 내에 일반 애플리케이션을 처리하는 일반 구역과 보안이 필요한 애플리케이션을 처리하는 보안 구역으로 분할하여 관리하는 하드웨어 기반의 보안 기술

- 엠디스크 : 한 번의 기록만으로도 자료를 영구 보관할 수 있는 광 저장 장치

- 멤리스터 : 메모리와 레지스터의 합성어로 전류의 방향과 양 등 기존의 경험을 모두 기억하는 소자

**DB 관련 신기술 용어**

- 빅데이터 : 기존의 관리 방법이나 분석 체계로는 처리하기 어려운 막대한 양의 정형 / 비정형 데이터 집합

- 브로드 데이터 : 다양한 채널에서 소비자와 상호 작용을 통해 생산 되었거나 기업 마케팅에 있어 효율적이고 다양한 데이터, 이전에 사용하지 않거나 몰랐던 새로운 데이터나 기존 데이터에 새로운 가치가 더해진 데이터

- 메타 데이터 : 일련의 데이터를 정의하고 설명해주는 데이터

- 디지털 아카이빙 : 디지털 정보 자원을 장기적으로 보존하기 위한 작업

- 하둡 : 오픈 소스를 기반으로 한 분산 컴퓨팅 플랫폼

- 타조 : 우리나라가 주도적으로 개발 중인 하둡 기반의 분산 데이터 웨어하우스 프로젝트

- 데이터 다이어트 : 데이터를 삭제하는 것이 아닌 압축하고 중복된 정보를 배제하고 새로운 기분에 따라 나누어 저장하는 작업

## [5.2 IT 프로젝트 정보 시스템 구축 관리(2)](http://127.0.0.1:5503/167)

2020. 4. 17. 13:06

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 네트워크 구축

**네트워크 설치 구조**

- 정보를 전달하기 위해 통신 규약에 의해 연결한 통신 설비의 집합

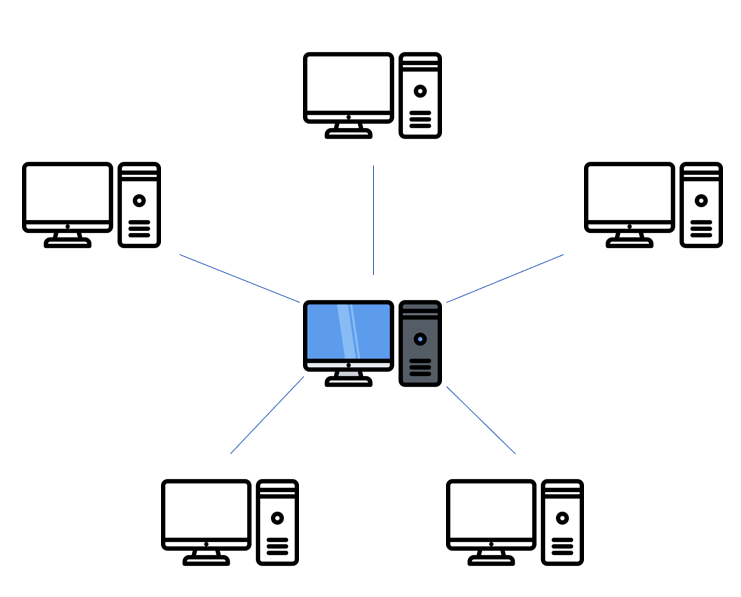
**성형(중앙 집중형)**

- 중앙 컴퓨터에 단말 장치들이 연결되는 구조

- Point-to-Point 방식으로 연결

- 중앙 집중식이므로 교환 노드의 수가 가장 적음

- 단말장치가 고장이나도 전체에 영향을 주지 않지만 중앙 장치가 고장이 나면 전체에 영향을 줌

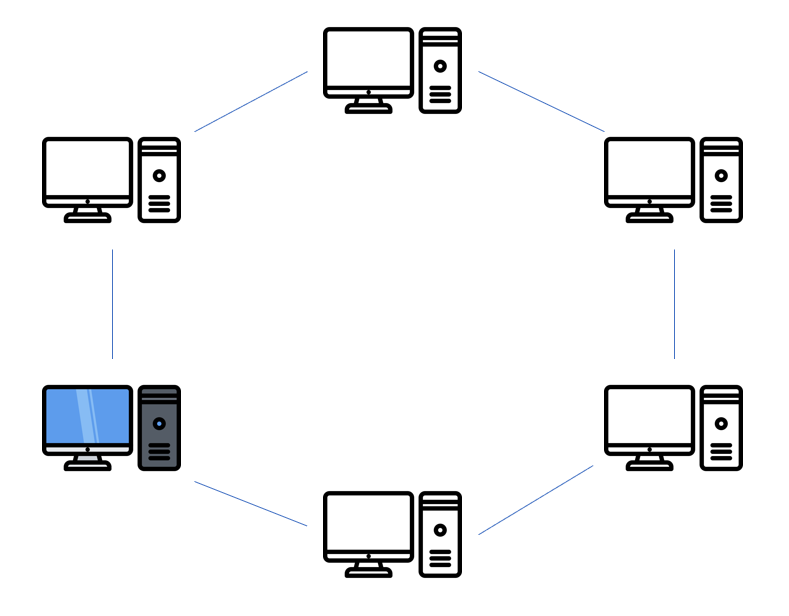


**링형(루프형)**

- 컴퓨터와 단말장치들을 서로 이웃하는 것끼리 Point-to-Point 방식으로 연결

- 데이트는 단방향 또는 양방향으로 전송 가능

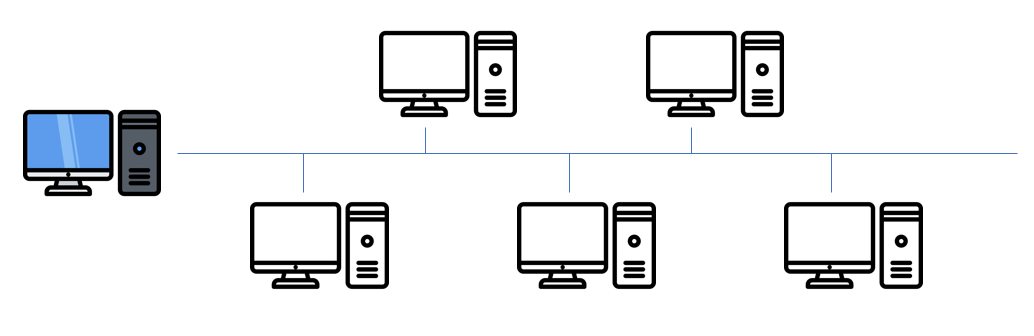
- 하나의 단말장치라도 고장이나면 전체에 영향을 줌



**버스형**

- 한 개의 통신 회선에 여러 대의 단말 장치가 연결되어 있는 형태

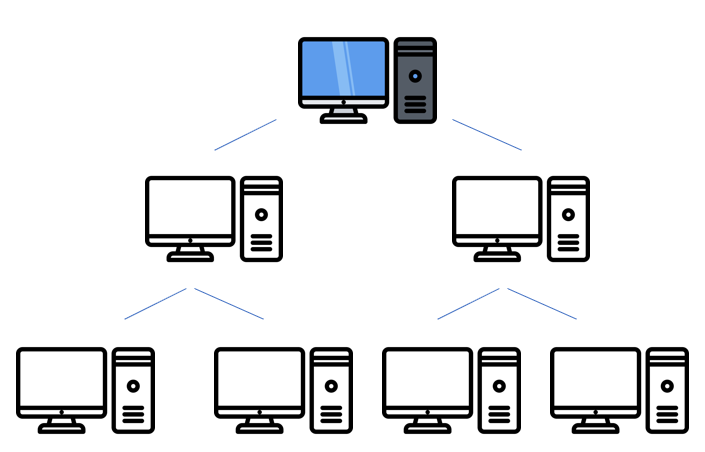
- 단말장치가 고장이나도 전체에 영향을 주지 않음



**계층형(분산형)**

- 중앙 컴퓨터와 일정 지역의 단말 장치까지는 하나의 회선으로 연결시키고 이웃하는 단말장치는 일정 지역 내 설치된 중간 단말장치로부터 다시 연결

- 분산처리 시스템을 구성하는 방식



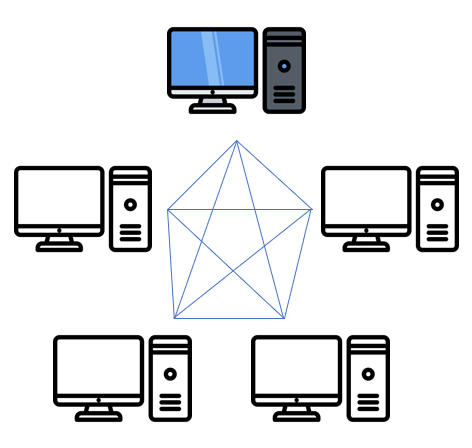
**망형(메쉬형)**

- 모든 지점의 컴퓨터와 단말장치를 서로 연결시킨 상태

- 많은 양의 통신을 필요로 하는 경우 사용

- 필요한 포트의 수 = 노드 수 - 1

- 필요한 회선의 수 = 노드 수(노드 수 -1) / 2

포트 수 = 4 / 회선 수 = 5x4/2 = 10

**네트워크 분류**

- 근거리 통신망(LAN)

    -> 비교적 가까운 거리에 있는 노드들을 연결하여 구성

    -> 주로 버스형이나 링형 구조 사용

- 광대역 통신망(WAN)

    -> 국가와 국가, 대륙과 대륙 등 멀리 떨어진 사이트들을 연결하여 구성

    -> 일정 지역은 LAN으로 연결하고 각 LAN을 연결하는 방식을 사용

### 스위치

**스위치의 정의**

- LAN과 LAN을 연결하여 훨씬 더 큰 LAN을 만드는 장치

**스위치의 분류**

- L2 스위치

    -> OSI 2계층에 속함

    -> 일반적으로 부르는 스위치를 말함

    -> MAC주소를 기반으로 프레임 전송

    -> 동일 네트워크 간 연결만 가능

- L3 스위치

    -> OSI 3계층에 속함

    -> L2 스위치에 라우터 기능이 추가

    -> IP 주소를 기반으로 패킷 전송

    -> 서로 다른 네트워크 연결 가능

- L4 스위치

    -> OSI 4계층에 속함

    -> L3 스위치에 로드밸런서(트래픽 분산 장치) 추가

    -> IP 주소 및 TCP/UDP를 기반으로 사용자들의 요구를 서버의 부하가 적은 곳에 배분하는 로드밸런싱 기능 제공

- L7 스위치

    -> OSI 7계층에 속함

    -> IP 주소, TCP/UDP 포트 ㅈ어보에 패킷 내용까지 참조하여 세밀하게 로드밸런싱함

**스위칭 방식**

- 스위치가 프레임을 전달하는 방식에 따라 나뉨

- Store and Forwarding : 데이터를 모두 받은 후 스위칭

- Cut-through : 데이터의 목적지 주소만 확인 후 바로 스위칭

- Fragment Free : 위의 두 방식의 장점만을 결합한 방식

**백본 스위치**

- 여러 네트워크들을 연결할 때 중추적 역할을 하는 네트워크인 백본에서 스위칭하는 장비

- 모든 패킷이 지나가는 네트워크 중심에 배치

- 주로 L3 스위치가 백본 스위치 역할을 함

### 경로 / 트래픽 제어

**경로 제어의 개요**

- 전송 경로 중 어느 한 경로에 데이터의 양이 집중되는 것을 피하면서 최저의 비용으로 최단 시간에 송신할 수 있는 경로인 최적 패킷 교환 경로를 결정

- 경로 제어표를 참조해서 라우터에 의해 수행

- 경로 제어 요소 : 성능 기준, 경로의 결정 시간 / 장소, 정보 발생지, 경로 정보의 갱신 시간

**경로 제어 프로토콜**

- 효율적인 경로 제어를 위해 네트워크를 제어하는 프로토콜

- IGP(내부 게이트웨이 프로토콜)

    -> 하나의 자율 시스템(AS) 내의 라우팅에 사용

    -> RIP : 현재 가장 널리 사용되며 소규모 네트워크에서 효율적인 방법

    -> OSPF : 대규모 네트워크에서 많이 사용되는 프로토콜

- EGP(외부 게이트웨이 프로토콜)

    -> 자율 시스템 / 게이트웨이 간 라우팅에 사용되는 프로토콜

- BGP

    -> 자율 시스템 간의 라우팅 프로토콜

    -> EGP의 단점을 보완하기 위해 만들어짐

    -> 초기 연결 시 라우팅 테이블을 교환하고 이후에는 변화된 정보만을 교환

**트래픽 제어의 개요**

- 네트워크의 보호, 성능 유지, 자원의 효율적인 이용을 위해 전송되는 패킷의 흐름, 양을 조절하는 기능

**흐름 제어**

- 네트워크 내의 원활한 흐름을 위해 송수신 사이 전송되는 패킷의 양이나 속도를 규제

- 송수신간 처리 속도 또는 버퍼 크기의 차이에 의해 생길 수 있는 버퍼 오버플로우를 방지

- 정지-대기 : 수신 측에서 확인 신호를 받아야 다음 패킷을 전송

- 슬라이딩 윈도우

    -> 확인 신호를 이용하여 송신 데이터의 양을 조절

    -> 수신 측의 확인 신호를 받지 않아도 정해진 패킷 수만큼 연속적으로 전송하는 방식

    -> 한 번에 여러 개의 패킷을 전송할 수 있음

**혼잡 제어**

- 네트워크 내의 패킷 수를 조절하여 네트워크의 오버플로우를 방지

**교착상태 방지**

- 교환기 내에 패킷들을 축적하는 기억 공간이 꽉 차있을 때 다음 패킷들을 이 기억 공간에 들어가기 위해 무한정 기다리는 현상을 방지

## [5.2 IT 프로젝트 정보 시스템 구축 관리(3)](http://127.0.0.1:5503/168)

2020. 4. 17. 13:07

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### 소프트웨어 개발 보안

**소프트웨어 개발 보안의 개요**

- 소프트웨어 개발 과정에서 발생할 수 있는 보안 취약점을 최소화하여 보안 위협으로부터 안전한 소프트웨어를 개발하기 보안 활동

- 데이터의 기밀성, 무결성, 가용성을 유지하는 것이 목표

**소프트웨어 개발 보안 관련 기관**

- 행정안전부(정책기관), 한국인터넷진흥원(전문기관), 행정기관(발주기관), 사업자(개발기관), 감리법인(보안 약점 진단)

### 소프트웨어 개발 직무별 보안 활동

**소프트웨어 개발 직무별 보안 활동**

- 프로젝트 관리자 : 보안 전략을 조직 구성원에게 전달하고 모니터링

- 요구사항 분석가 : 아키텍트가 고려해야 할 보안 관련 비즈니스 요구사항을 설명

- 아키텍트 : 보안 오류가 발생하지 않도록 보안 기술 문제를 충분히 이해하고 시스템에 사용되는 모든 리소스 정의 및 보안 요구사항 적용

- 설계자 : 특정 기술에 대해 보안 요구사항의 만족성 여부를 파악하고 소프트웨어에서 발견된 보안 위협에 대해 적절히 대응

- 구현 개발자 : 개발 환경에서 프로그램을 구현할 수 있도록 시큐어 코딩 표준을 준수하여 개발

- 테스트 분석가 : 개발 요구사항과 구현 결과를 반복적으로 확인

- 보안 감시자 : 개발 프로젝트의 전체 단계에서 활동하며 현재 상태와 보안을 보장

### Secure OS

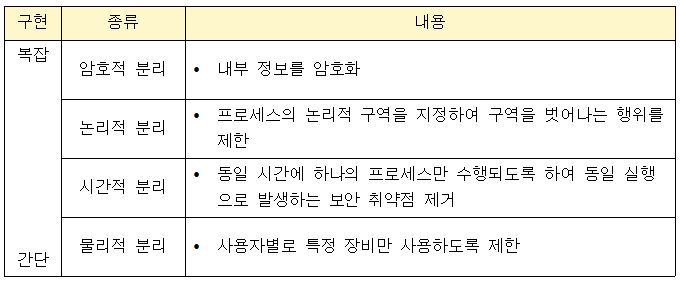
**Secure OS의 개요**

- 기존 운영체제에 내재된 보안 취약점을 해소하기 위해 보안 기능을 갖춘 커널을 이식한 운영체제

- TCB를 기반으로 참조모니터의 개념을 구현하고 집행

    -> 참조 모니터와 보안 커널의 특징 : 격리성, 검증 가능성, 완전성

- 보호 대상 : 메모리, 보조 기억장치 및 저장된 데이터, 하드웨어 장치, 자료 구조, 명령어, 각종 보호 메커니즘 등



**Secure OS의 보안 기능**

- 식별 및 인증, 임의적 접근통제, 강제적 접근 통제, 객체 재사용 보호, 완전한 조정, 신뢰 경로, 감사 및 감사기록 축소

### 회복 / 병행제어

**회복**

- 트랜잭션을 수행하는 도중 장애가 발생하여 데이터베이스가 손상되었을 때 복구하는 작업

- 장애의 유형 : 트랜잭션 장애, 시스템 장애, 미디어 장애

- 회복 관리기 : 트랜잭션이 실행이 완료되지 못하면 트랜잭션이 데이터베이스에 생성했던 모든 변화를 취소(Undo)시키고 이전의 원래 상태로 복구하는 역할을 담당

**병행 제어**

- 동시에 여러 개의 트랜잭션을 수행할 때 데이터베이스의 일관성을 유지할 수 있도록 트랜잭션 간 상호작용을 제어

- 병행 제어의 목적

    -> 데이터베이스의 공유 및 시스템의 활용도 최대화

    -> 데이터베이스 일관성 유지

    -> 응답 시간 최소화

**병행 수행의 문제점**

- 갱신 분실 : 두 개 이상의 트랜잭션이 같은 자료를 공유하여 갱신할 때 갱신 결과의 일부가 없어짐

- 비 완료 의존성 : 하나의 트랜잭션이 실패한 후 회복되기 전에 다른 트랜잭션이 실패한 갱신 결과를 참조

- 모순성 : 병행 수행될 때 원치 않는 자료를 이용하여 문제가 발생

- 연쇄 복귀 : 트랜잭션 중 하나에 문제가 생겨 ROLLBACK 하는 경우 다른 트랜잭션도 같이 ROLLBACK 됨

### 데이터 표준화

**데이터 표준화의 정의**

- 시스템을 구성하는 데이터 요소의 명칭, 정의, 형식, 규칙에 대한 원칙을 수립하고 정의

**데이터 표준**

- 데이터 모델이나 데이터베이스에서 정의할 수 있는 모든 오브젝트를 대상으로 수행

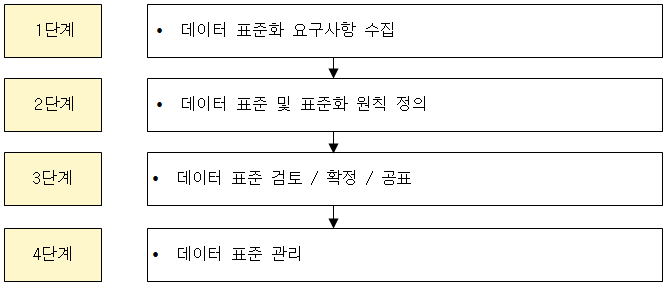
- 표준 단어 : 업무에서 사용하는 일정한 의미를 가진 최수 단위의 단어

- 표준 도메인 : 칼럼을 성질에 따라 그룹핑함

- 표준 코드 : 선택할 수 있는 값을 기준에 맞게 이미 정의된 코드값

- 표준 용어 : 표준 단어 / 도메인 / 코드를 바탕으로 표준 용어 구성

**데이터 표준화 절차**



**데이터 표준화의 대상**

- 데이터 명칭 : 데이터를 유일하게 구별, 의미 전달, 업무적 보편성을 갖는 이름을 가져야 함

- 데이터 정의 : 제3자의 입장에서도 쉽게 이해할 수 있도록 데이터가 의미하는 범위와 자격 요건을 규정

- 데이터 형식 : 데이터를 형식을 일관적으로 정의함으로써 데이터 입력 오류 및 통제 위험 등을 최소화

- 데이터 규칙 : 데이터 값을 사전에 지정해 데이터의 정합성 및 완전성 향상

**데이터 표준화의 기대효과**

- 데이터의 의미나 위치를 파악하고 의사소통하기 쉽다

- 데이터 유지보수 및 운용에 있어 여러 이점이 있음

## [5.3 소프트웨어 개발 보안 구축(1)](http://127.0.0.1:5503/170)

2020. 4. 17. 21:37

[2020정보처리기사필기정리](http://127.0.0.1:5503/tag/2020%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%EA%B8%B0%EC%82%AC%ED%95%84%EA%B8%B0%EC%A0%95%EB%A6%AC)

728x90

### Secure SDLC

**Secure SDLC의 개요**

- 보안상 안전한 소프트웨어를 개발하기 위해 SDLC(소프트웨어 개발 생명주기)에 보안 강화를 위한 프로세스를 포함한 것

- 유지보수 단계에서 보안 이슈를 해결하기 위해 소모되는 비용을 최소화하기 위함

- Secure Software 사의 CLASP, Microsoft 사의 SDL 등

**요구사항 분석 단계에서의 보안 활동**

- 보안 항목에 해당하는 요구사항을 식별하는 작업 수행

- 보안 수준을 보안 요소별로 등급을 구분하여 분류

- 보안 요소 : 기밀성, 무결성, 가용성, 인증, 부인 방지

**설계 단계에서의 보안 활동**

- 식별된 요구사항을 소프트웨어 설계서에 반영하고 보안 설계서 작성

- 네트워크, 서버, 물리적 보안, 개발 프로그램 등 환경에 대한 보안통제 기준을 수립하여 설계에 반영

**구현 단계에서의 보안 활동**

- 표준 코딩 정의서 및 소프트웨어 개발 보안 가이드를 준수하여 설계서에 따라 보안 요구 사항 구현

- 단위 테스트 실행

- 시큐어 코딩 : 구현 단계에서 발생할 수 있는 보안 취약점을 최소화하기 위해 보안 요소들을 고려하여 코딩

**테스트 단계에서의 보안 활동**

- 작성된 보안 설계서를 바탕으로 보안 사항들이 정확히 반영되고 동작되는지 점검

**유지보수 단계에서의 보안 활동**

- 이전 과정을 모두 수행했음에도 발생할 수 있는 보안사고들을 식별하고 발생 시 해결하고 보안 패치 실시

### 세션 통제

**세션 통제의 개요**

- 서버와 클라이언트의 연결인 세션 간의 연결로 인해 발생하는 정보를 관리

- 요구사항 분석 및 설계 단계에서 진단해야 하는 보안 점검 내용

**불충분한 세션 관리**

- 일정한 규칙이 존재하는 세션ID가 발급되거나 타임아웃이 너무 길게 설정되어 있는 경우 발생

- 세션 하이재킹(세션 정보를 가로채는 공격)을 통해 획득한 세션 ID로 접근할 수 있음

**잘못된 세션에 의한 정보 노출**

- 다중 스레드 환경에서 멤버 변수에 정보를 저장할 때 발생

- 변수의 범위를 제한하는 방법으로 방지 가능

- 싱글톤 패턴에서 발생하는 레이스컨디션으로 인해 동기화 오류가 발생하거나 멤버 변수의 정보가 노출될 수 있음

- 레이스컨디션 : 두 개 이상의 프로세스가 공용 자원을 획득하기 위해 경쟁하고 있는 상태

**세션 설계 시 고려사항**

- 로그아웃 요청 시 할당된 세션이 완전히 제거되도록 함

- 이전 세션이 종료되지 않으면 새로운 세션이 생성되지 못하도록 함

### 입력 데이터 검증 및 표현

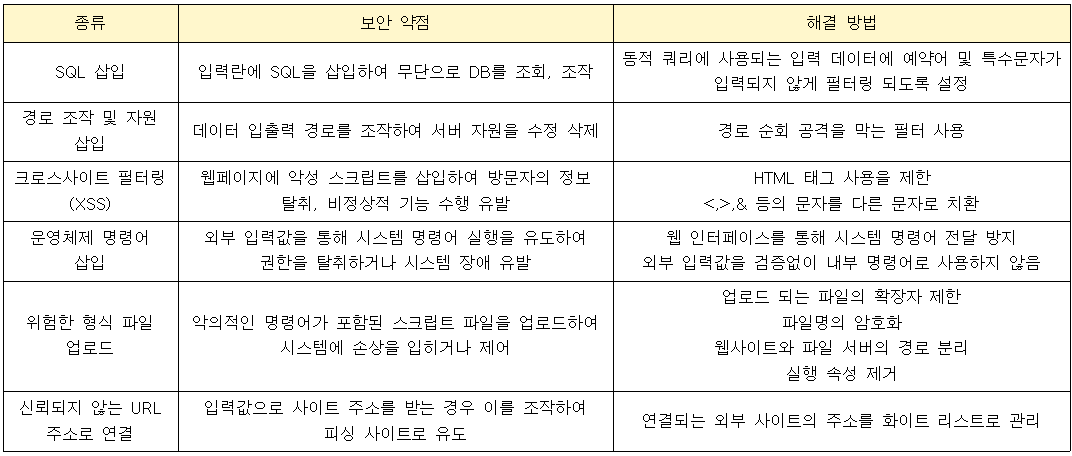
**입력 데이터 검증 및 표현의 개요**

- 입력 데이터로 인해 발생하는 문제들을 예방하기 위해 구현 단계에서 검증해야 하는 보안 점검 항목

- 개발 단계에서 유효성 검증 체계를 갖추고 검증되지 않은 데이터가 입력될 시 처리할 수 있도록 구현해야 함

- 일관된 언어셋을 사용하여 코딩

**입력 데이터 검증 및 표현의 보안 약점**

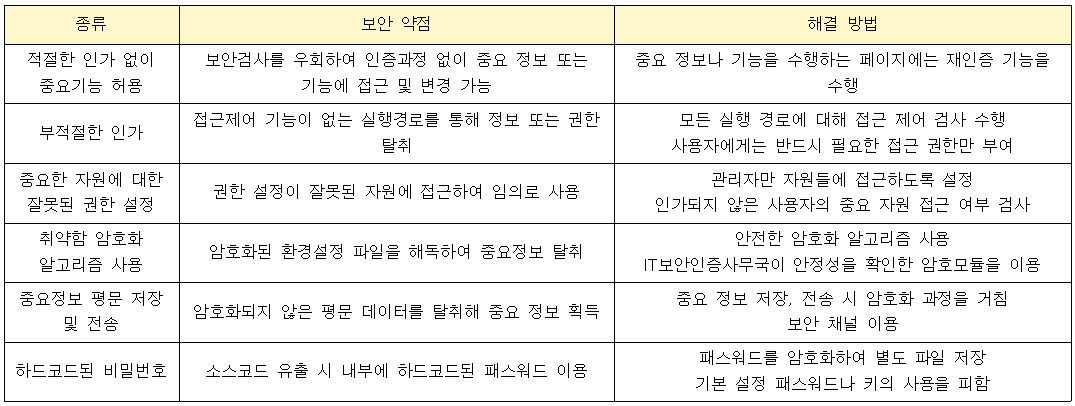


### 보안 기능

**보안 기능의 개요**

- 코딩하는 기능인 인증, 접근제어, 기밀성, 암호화들을 올바르게 구현하기 위해 구현 단계에서의 보안 점검 항목

**보안 기능의 보안 약점**



### 시간 및 상태

**시간 및 상태의 개요**

- 동시 수행을 지원하는 병렬 시스템이나 다수의 프로세스가 동작하는 환경에서 시간과 실행 상태를 관리하여 원활하게 동작되도록 하기 위한 보안 검증 항목

**TOCTOU 경쟁 조건**

- 검사 시점과 사용 시점을 고려하지 않고 발생하는 보안 약점

**종료되지 않은 반복문 또는 재귀 함수**

- 조건이나 논리 구조를 잘못 구성하여 종료할 수 없게 되는 경우 시스템 자원을 끊임없이 사용하여 자원고갈로 인한 서비스 또는 시스템 장애 발생

## [5.3 소프트웨어 개발 보안 구축(2)](http://127.0.0.1:5503/171)

2020. 4. 17. 23:23

728x90

### 에러 처리의 개요

**에러 처리의 개요**

- 소프트웨어 실행 중 발생할 수 있는 오류들을 사전에 정의하여 오류로 인해 발생할 수 있는 문제들을 예방하기 위한 보안 점검 항목

- 예외처리 구문을 통해 오류에 대한 사항 정의

**오류 메시지를 통한 정보 노출**

- 오류 발생으로 실행 환경, 사용자 정보, 디버깅 정보 등 중요 정보를 소프트웨어가 메시지로 외부에 노출하는 보안 약점

- 오류 발생 시 최대한 내부에서 처리하거나 메시지를 최소한의 내용으로 출력하여 정보 노출을 방지해야 함

**오류 상황 대응 부재**

- 소프트웨어의 오류에 대한 에러 처리를 하지 않았거나 미비로 인해 발생하는 보안 약점

**부적절한 예외처리**

- 함수의 반환 값 또는 오류들을 세분화하여 처리하지 않고 광범위하게 묶어서 한 번에 처리하거나 누락된 예외가 존재할 때 발생하는 보안 약점

### 코드 오류

**코드 오류의 개요**

- 소프트웨어 구현 단계에서 코딩 중 실수하기 쉬운 형 변환, 자원 반환 등 오류를 예방하기 위한 보안 점검 항목

**널 포인터 역참조**

- 널 포인터가 가리키는 메모리에 어떠한 값을 저장할 때 발생하는 보안 약점

- 오류로 인해 반환되는 널 값을 포인터로 참조하는 경우 발생

**부적절한 자원 해제**

- 자원을 반환하는 코드를 누락하거나 프로그램 오류로 할당된 자원을 반환하지 못했을 때 발생하는 보안 약점

- 유한한 시스템 자원이 계속 점유하고 있으면 자원 부족이 발생

**해제된 자원 사용**

- 이미 반환된 메모리를 참조하는 경우 발생하는 보안 약점

- 반한 된 메모리를 참조하는 경우 예상하지 못한 값 또는 코드를 수행하게 되어 의도하지 않은 결과가 발생됨

**초기화되지 않은 변수 사용**

- 변수 선언 후 값이 부여되지 않은 변수를 사용할 때 발생하는 보안 약점

### 캡슐화

**캡슐화의 개요**

- 정보 은닉이 필요한 중요한 데이터와 기능을 불충분하게 캡슐화하거나 잘못 사용함으로써 발생할 수 있는 문제를 예방하기 위한 보안 점검 항목

**제거되지 않고 남은 디버그 코드**

- 개발 중에 버그 수정이나 결과값을 확인을 위해 남겨둔 코드들로 인해 발생하는 보안 약점

**시스템 데이터 정보 노출**

- 시스템의 내부 정보를 시스템 메시지 등을 통해 외부로 출력하도록 구현했을 때 발생하는 보안 약점

**Public 메소드로부터 반환된 Private 배열**

- Private 배열을 Public 메소드에서 반환할 때 발생하는 보안 약점

**Private 배열에 Public 데이터 할당**

- Private 배열에 Public으로 선언된 데이터 또는 메소드의 파리 미터를 저장할 때 발생하는 보안 약점

### API 오용

**API 오용의 개요**

- 소프트웨어 구현 단계에서 API를 잘못 사용하거나 보안에 취약한 API를 사용하지 않도록 하는 보안 검증 항목

**DNS Lookup에 의존한 보안 결정**

- 도메인명에 의존하여 보안 결정을 내리는 경우 발생하는 보안 약점

- IP 주소를 직접 입력하여 접근하게 하여 방지 가능

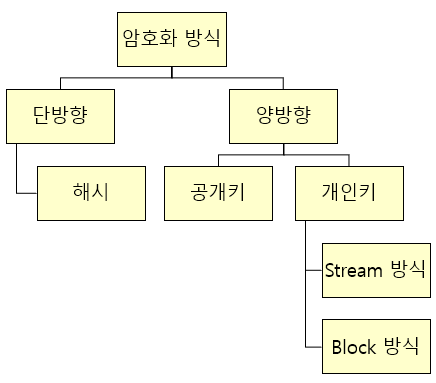
**취약한 API 사용**

- 보안 문제로 사용이 금지된 API를 사용하거나 잘못된 방식으로 API를 사용했을 때 발생하는 보안 약점

### 암호 알고리즘

**암호 알고리즘의 개요**

- 중요정보를 보호하기 위한 평문을 암호화된 문장으로 만드는 방법



**개인키 암호화 기법**

- 동일한 키로 데이터를 암호화하고 복호화함

- 대칭 암호 기법, 단일키 암호화 기법이라고도 함

- Stream 기법 : 평문과 동일한 길이의 스트림을 생성하여 비트 단위로 암호화

- Block 기법 : 한 번에 하나의 데이터 블록을 암호화

**공개키 암호화 기법**

- 데이터를 암호화하는 공개키는 데이터베이스 사용자에게 공개하고 복호화하는 비밀키는 관리자에게만 공개

- 비대칭 암호화 기법이라고도 함

- RSA기법 : 공개키와 비밀키는 메시지를 열고 잠그는 상수를 의미

**양방향 암호화 알고리즘 종류**

- SEED : 블록 크기는 128비트, 키의 길이에 따라 128, 256로 분류

- ARIA : 블록 크기는 128비트, 키의 길이에 따라 128, 192, 256로 분류

- DES : 블록 크기는 64비트, 키의 길이 56비트

- AES : 블록 크기는 128비트, 키의 길이에 따라 128, 192, 256로 분류

**해시(Hash)**

- 임의의 길이의 입력 데이터나 메시지를 고정된 길이의 값이나 키로 변환

- SHA 시리즈, MD5, N-NASH, SNEFRU 등

## [5.4 시스템 보안 구축](http://127.0.0.1:5503/172)

2020. 4. 19. 21:42

728x90

### 서비스 공격 유형

**서비스 거부 공격의 개념**

- 표적이 되는 서버의 자원을 고갈시킬 목적으로 다수의 공격자 또는 시스템에서 대량의 데이터를 한 곳의 서버를 집중적으로 전송함으로써 표적이 되는 서버의 정상적인 기능을 방해

**Ping of Death**

- Ping 명령 전송 시 패킷의 크기를 인터넷 프로토콜 허용 범위 이상으로 공격하여 공격 대상의 네트워크를 마비시키는 서비스 거부 방법

**SMURFING**

- IP나 ICMP의 특성을 악용하여 엄청난 양의 데이터를 한 사이트에 집중적으로 보냄으로써 네트워크를 불능 상태로 만드는 공격 방법

**SYN Flooding**

- 공격자가 가상의 클라이언트로 위장하여 3-way-handshake 과정을 의도적으로 중단시킴으로써 공격 대상지인 서버가 대기 상태에 놓여 정상적인 서비스를 수행하지 못하게 하는 공격 방법

**TearDrop**

- 데이터의 송수신 단계에서 전송되는 Fragment Offset 값을 변경시켜 패킷을 재조립할 때 오류로 인한 과부하를 발생시킴으로 시스템이 다운되도록 하는 공격 방법

**Land**

- 패킷 전송 시 송수신 IP 주소를 모두 공격 대상의 IP주소로 하여 공격 대상에게 전송하여 무한히 자신에게 응답을 수행하게 되는 공격 방법

**DDos(Distributed Denial of Service, 분산 서비스 거부) 공격**

- 여러 곳에 분산된 공격 지점에서 한 곳의 서버에 대해 공격을 수행

- 네트워크에서 취약점이 있는 호스트들을 탐색한 후 호스트들에게 분산 서비스 공격 툴을 설치하여 에이전트로 만든 후 공격에 이용

- 분산 서비스 공격 툴

    -> Trin00 : 초기 형태의 데몬으로 UDP Flooding 공격 수행

    -> TFN : UDP Flooding, TCP SYN Flood 공격, ICMP 응답 요청, 스머핑 공격 등 수행

    -> TFN2K : TFN의 확장판

    -> Stacheldraht : 이전의 툴들을 유지하면서 암호화된 통신을 수행하며 툴이 자동으로 업데이트되도록 설계

**네트워크 침해 공격 관련 용어**

- 스미싱 : 문자 메시지를 이용해 사용자의 개인 신용 정보를 빼내는 수법

- 스피어 피싱 : 일반적인 이메일로 위장한 메일을 지속적으로 발송하여 메일의 링크나 첨부된 파일을 클릭하게 유도하여 개인 정보를 탈취

- APT(지능형 지속 위협) : 조직적으로 특정 기업이나 조직 네트워크에 침투해 활동 거점을 마련한 뒤 때를 기다리면서 보안을 무력화시키고 정보를 수집한 다음 외부로 빼돌리는 형태의 공격

- 무작위 대입 공격 : 암호화된 문서의 암호키를 찾기 위해 무작위로 값을 대입하여 공격하는 방식

- 큐싱 : QR코드를 통해 악성 앱을 다운받게 하여 개인 정보를 탈취하는 공격 방식

- SQL 삽입 공격 : 웹사이트를 무차별적으로 공격하는 과정에서 취약한 사이트 발견 시 데이터를 조작하는 일련의 공격 방식

- 크로스 사이트 스크립 : 웹 페이지의 내용을 사용자 브라우저에 표현하기 위해 사용되는 스크립트의 취약점을 악용한 해킹 기법

**정보 보안 침해 공격 관련 용어**

- 좀비 PC : 악성코드에 감염되어 다른 프로그램이나 컴퓨터를 조종하도록 만들어진 컴퓨터

- C&C 서버 : 해커가 원격지에서 감염된 좀비 PC에 명령을 내리고 악성코드를 제어하기 위한 용도로 사용하는 서버

- 봇넷 : 악성 프로그램에 감염된 컴퓨터들이 네트워크로 연결된 형태

- 웜 : 네트워크를 통해 연속적으로 자신을 복사하여 시스템의 부하를 높여 시스템을 다운시키는 바이러스의 일종

- 제로 데이 공격 : 보안 취약점이 발견됐을 때 공표되기도 전에 해당 취약점을 통해 신속하게 이루어지는 보안 공격

- 키로거 공격 : 사용자의 키보드 움직임을 탐지하여 개인 정보를 몰래 빼가는 공격

- 랜섬웨어 : 사용자의 컴퓨터에 잡입해 파일을 암호화하여 사용자가 열지 못하게 하는 프로그램

- 백도어 : 액세스 편의를 위해 시스템 보안을 제거하여 만들어 놓은 비밀 통로를 통해 범죄에 악용되는 형태

- 트로이 목마 : 정상적인 기능을 하는 프로그램인 척 프로그램에 숨어 있다가 해당 프로그램이 동작될 때 활성화되어 부작용을 일으키는 형태

### 서버 인증

**보안 서버의 개념**

- 인터넷을 통해 개인정보를 암호화하여 송수신할 수 있는 기능을 갖춘 서버

- 서버에 SSL(Secure Socket Layer) 인증서를 설치하여 전송 정보를 암호화하여 송수신

- 서버에 암호화 응용 프로그램을 설치하고 전송 정보를 암호화하여 송수신

**인증의 개념**

- 다중 사용자 컴퓨터 / 네트워크 시스템에서 로그인을 요청한 사용자의 정보를 확인하고 접근 권한을 검증하는 보안 절차

- 지식 기반 인증

    -> 사용자가 기억하고 있는 정보를 기반으로 인증을 수행

    -> 고정된 패스워드, 패스 프레이즈, 아이핀

- 소유 기반 인증

    -> 사용자가 소유하고 있는 것을 기반으로 인증을 수행

    -> 신분증, 메모리 카드, 스마트 카드, OTP

- 생체 기반 인증

    -> 사용자의 고유한 생체 정보를 기반으로 인증을 수행

    -> 지문, 홍채/망막, 얼굴, 음성, 정맥

- 기타 인증 방법

    -> 행위 기반 인증 : 사용자의 행동 정보를 이용해 인증 수행

    -> 위치 기반 인증 : 인증을 시도하는 위치나 적절성 확인

### 보안 아키텍처 / 프레임워크

**보안 아키텍처**

- 정보 시스템의 무결성, 가용성, 기밀성을 확보하기 위해 보안 요소 및 보안 체계를 식별하고 이들 간의 관계를 정의한 구조

- ITU-T, X.805의 보안 표준을 기준으로 하여 보안 아키텍처 모델 구성

    -> 보안 계층 : 인프라 시스템, 응용 프로그램, 데이터, 단말기, 인터페이스

    -> 보안 영역 : 정보 시스템, 제어 시스템, 클라우드, 무선, 사물인터넷

    -> 보안 요소 : 인증, 접근 통제, 데이터 처리 보호, 암호화, 감사 추적, 위협 탐지

**보안 프레임워크**

- 안전한 정보 시스템 환경을 유지하고 보안 수준을 향상시키기 위한 체계

- ISO 27001 : 정보 보안 관리를 위한 국제 표준이며 가장 대표적인 보안 프레임워크

### 로그 분석

**로그의 개념**

- 시스템 사용에 대한 모든 내역을 기록하여 시스템 침해 사고 발생 시 해킹 흔적이나 공격 기법을 파악할 수 있음

**리눅스 로그**

- var/log 디렉토리에서 기록하고 관리

- syslogd 데몬은 etc/syslog.conf 파일을 읽어 로그 관련 파일들의 위치를 파악 후 작업 시작

- 커널 로그, 부팅 로그, 크론 로그, 시스템 로그, 보안 로그, FTP 로그, 메일 로그

**윈도우 로그**

- Windows 시스템에서 이벤트 로그 형식으로 시스템의 로그 확인

- 응용 프로그램, 보안, 시스템, Setup, Forwareded Event에 대한 로그 확인 가능

### 보안 솔루션

**보안 솔루션의 개념**

- 접근 통제, 침입 차단 등을 수행하여 외부로부터 불법적인 침입을 막는 기술 및 시스템

**방화벽**

- 기업이나 조직 내부의 네트워크와 인터넷 간에 전송되는 정보를 선별하여 수용, 거부, 수정하는 기능을 가진 침입 차단 시스템

**침입 탐지 시스템(IDS)**

- 컴퓨터 시스템의 비정상적인 행위를 실시간으로 탐지하는 시스템

- 문제 발생 시 모든 내외부 정보의 흐름을 실시간으로 차단하기 위해 해커 침입 패턴에 대한 추적과 유해 정보 감시가 필요

**침입 방지 시스템(IPS)**

- 방화벽과 침입 탐지 시스템을 결합

- 비정상적인 트래픽을 능동적으로 차단하고 격리하는 방어 조치를 취하는 보안 솔루션

**데이터 유출 방지(DLP)**

- 내부 정보의 외우 유출을 방지하는 보안 솔루션

- 내부 PC와 네트워크 상의 모든 정보를 검색하고 사용자 행위를 탐지, 통제해 외부로의 유출을 사전에 방지

**웹 방화벽**

- 일반 방화벽이 탐지하지 못하는 SQL 삽입 공격, XSS 등의 웹 기반 공격을 방어할 목적으로 만들어진 웹 서버에 특화된 방화벽

**VPN(가상 사설 통신망)**

- 인터넷 등 통신 사업자의 공중 네트워크와 암호화 기술을 이용하여 사용자가 마치 자신의 전용 회선을 사용하는 것처럼 해주는 보안 솔루션

**NAC**

- 네트워크에 접속하는 내부 PC의 MAC 주소를 IP 관리 시스템에 등록 후 일관된 보안 관리 기능을 제공하는 보안 솔루션

**ESM**

- 다양한 장비에서 발생하는 로그 및 보안 이벤트를 통합하여 관리하는 보안 솔루션