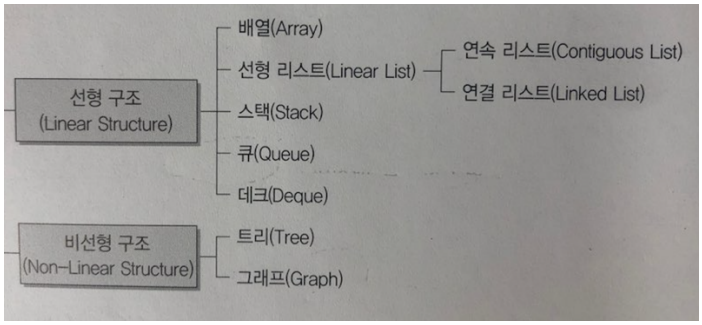
Session034 자료 구조

1 자료 구조의 정의

1.1 정의

- 자료의 표현과 그것과 관련된 연산
- 자료들을 조직하고 구조화하는 것
- 필요한 모든 연산 처리 가능
- 자료 구조에 따라 실행시간이 다름

2 자료 구조의 분류



3 배열(Array)

3.1 같은 크기로 나열되어 순서를 갖고 있는 집합

- 정적인 자료구조, 기억장소 추가 어렵, 데이터 삭제 시 기억장소는 빈 공간 메모리 낭비
- 첨자를 이용 데이터 접근
- 반복적인 데이터 처리 작업
- 동일한 이름 변수 사용
- N차원 배열

4 선형 리스트(Linear List)

4.1 일정한 순서에 의해 나열된 자료 구조

- 배열을 이용하는 연속 리스트(Contiguous List)
- 포인터를 이용하는 연결 리스트(Linked List)

4.2 연속 리스트(Contiguous List)

- 연속되는 기억장소에 저장
- 이용효율 밀도 1로 가장 좋다
- 중간 데이터 삽입을 위해 연속된 빈 공간 필요, 삽입 삭제 시 자료 이동 필요

4.3 연결 리스트(Linked List)

- 임의의 기억공간에 기억, 자료 항복 순서 따라 노드의 포인터 부분 이용

- 삽입 삭제 작업 용이
- 이용 효율이 좋지 않다
- 포인터 찾는 시간 접근속도가 느리다
- 중간 노드 연결이 끊어지면 다음 노드 찾기 힘들다

5 스택(Stack)

5.1 한쪽 끝으로만 자료 삽입, 삭제 작업

- LIFO
- 오버플로, 언더플로 발생

6 큐(Queue)

6.1 한쪽은 삽입, 한쪽은 삭제

- FIFO
- 두 개의 포인터(시작과 끝)
- 프런트(F, Front) 포인터 : 가장 먼저 삽입된
- 리어(R, Rear) 포인터 : 가장 마지막 삽입된

7 트리(Tree)

7.1 정정(Node)와 선분(Branch)을 이용, 사이클을 이루지 않도록 그래프 형태

- 기억공간 Node, 노드와 노드 연결선 Link
- 계보, 조직도 표현하기 적합
- 디그리(Degree, 차수) : 각 노드에서 뻗어 나온 가지 수
- 단말 노드(Terminal Node) = 잎 노드(Leaf Node) : 자식이 없는 노드, 디그리 0
- 자식 노드(Son Node) : 다음 레벨의 노드들
- 부모 노드(Parent Node) : 이전 레벨의 노드들
- 형제 노드(Brother Node, Sibling) : 동일한 부모가는 노드들
- 트리의 디그리 : 노드들의 디그리 중 가장 많은 수

Session035 데이터저장소/데이터베이스/DBMS

1 데이터저장소

1.1 데이터들을 논리적 구조로 조직화

- 논리 데이터저장소, 물리 데이터저장소 구분
- 논리 데이터저장소 : 데이터 간의 연관성, 제약조건을 논리적 구조로 조직화
- 물리 데이터저장소 : 데이터와 구조를 물리적 특성 고려, 저장장치에 저장
- 구축하는 과정은 동일

2 데이터베이스

2.1 상호 관련된 데이터들의 모임

- 통합된 데이터(Integrated Data) : 중복 배제한 모임
- 저장된 데이터(Stored Data) : 저장 매체에 저장된 자료
- 운영 데이터(Operational Data) : 없어서는 안 될 자료
- 공용 데이터(Share Data) : 공동으로 소유 유지하는 자료

3 DBMS(DataBase Management System)

3.1 데이터베이스를 관리해 주는 소프트웨어

- 데이터의 종속성, 중복성 문제를 해결
- 데이터베이스의 구성, 접근 방법, 유지관리에 대한 모든 책임
- 정의(Definition), 조작(Manipulation), 제어(Controll) 기능
  - ■ 정의 : 데이터의 형, 구조에 대한 정의, 이용 방식, 제약 조건 등을 명시
  - ■ 조작 : 검색, 갱신, 삽입, 삭제 등을 처리 위해 사용자와 데이터베이스 사이 인터페이스 제공
  - ■ 제어 : 무결성 유지, 허가된 데이터만 접근, 권한을 검사 가능 해야 함, 동시 접근하여도 항상 정확성 유지(병행제어 Concurrency Control)
  - ■

4 DBMS의 장단점

4.1 장점

- 논리적, 물리적 독립성 보장
- 중복 피함
- 자료 공동 이용
- 일관성 유지
- 무결성 유지
- 보안 유지
- 데이터 표준화 가능
- 통합하여 관리
- 최신의 데이터 유지
- 실시간 처리 가능

4.2 단점

- 전문가 부족
- 전산화 비용 증가
- 디스크의 집중적 액세스로 과부하(Overhead)
- 예비(Backup)와 회복(Recovery) 어렵
- 시스템 복잡

Session036 데이터 입출력

1 데이터 입출력 개요

1.1 데이터베이스에 데이터를 입력, 출력하는 작업

- 단순 입출력뿐만 아니라 데이터 조작하는 모든 행위, SQL 사용
- 코드 내 SQL코드 삽입, 객체와 데이터 연결하는 것을 데이터 접속(Data Mapping)
- 작업의 단위 또는 한꺼번에 모두 수행되어야 할 일련의 연산들을 트랙잭션(Transaction)

2 SQL(Structed Query Language)

2.1 관계형 데이터베이(RDB)를 지원하는 언어

- 관계대수와 관계해석 기초, 혼합 데이터 언어
- 질의어이지만 데이터 구조의 정의, 데이터 조작, 데이터 제어 기능 모두
- DDL(Data Define Language)
  - SCHEMA, DOMAIN, TABLE, VIEW, INDEX를 정의 변경 삭제
- DML(Data Manipulation Language)
  - 데이터를 실질적으로 처리
- DCL(Data Control Language)
  - 데이터의 보안, 무결성, 회복, 병행수행 제어

3 데이터 접속(Data Mapping)

3.1 프로그래밍 코드와 데이터를 연결, SQL Mapping, ORM

3.1.1 SQL Mapping : 코드 내에 SQL 직접 입력, JDBC, ODBC, MyBatis

3.1.2 ORM(Object-Relational Mapping) : 프로그래밍의 객체(Object)와 관계형(Relational) 데이터베이스의 데이터를 연결(Mapping), JPA, Hibernate, Django

4 트랜잭션(Transaction)

4.1 상태를 변화시키는 하나의 논리적 기능을 수행하기 위한 작업의 단위 또는 한꺼번에 모두 수행되어야 할 일련의 연산

4.2 트랜잭션 제어 명령어를 사용 TCL(Transaction Control Language), COMMIT, ROLLBACK, SAVEPOINT

- COMMIT : 변경 내용을 데이터베이스에 반영
- ROLLBACK 모든 변경 작업 취소, 이전 상태로
- SAVEPOINT(=CHECKPOINT) : ROLLBACK 할 위치 저장

Session037 절차형 SQL

1 절차형 SQL의 개요

1.1 프로그래밍 언어와 같이 연속적인 실행, 분기, 반복 등의 제어 가능한 SQL

- 단일 SQL 문장으로 처리하기 어려운 연속적인 작업들을 처리
- 다양한 기능을 수행하는 저장 모듈 생성
- DBMS에서 직접 실행, 입출력 패킷이 적은 편
- 블록 구조, 기능별 모듈화 가능
- 프로시저, 트리거, 사용자 정의 함수
  - ■ 프로시저 : 특정 기능 수행 일종의 트랜잭션 언어, 호출을 통해 실행
  - ■ 트리거 : 이벤트 발생할 때마다 관련 작업 자동으로 수행
  - ■ 사용자 정의 함수 : 프로시저와 유사, 종료 시 처리 결과를 단일값으로 반환

2 절차형 SQL의 테스트와 디버깅

2.1 디버깅을 통해 기능의 적합성 여부 검증, 실행을 통해 테스트 과정

- 생성을 통해 구문 오류(Syntax Error)나 참조 오류 존재 여부 확인
- 오류 및 경고 메시지 상세하지 않음, SHOW 명령어 로 확인
- 디버깅을 통해 로직 검증, 결과를 통해 최종 확인
- 디버깅은 주석처리로 출력을 이용하여 확인

3 쿼리 성능 최적화

3.1 성능 향상을 위해

- 성능 측정 도구 APM(Application Performance Management/Mornitoring)을 사용, 최적화 할 쿼리 선정
- 실행 계혹 검토, SQL 코드와 인덱스 재구성