Software Architecture

Data-Centered (DC) Software Architectures

RDBMS로 생각하면 좋음

Repository Architecture Style

Eunmi Choi
Kookmin University

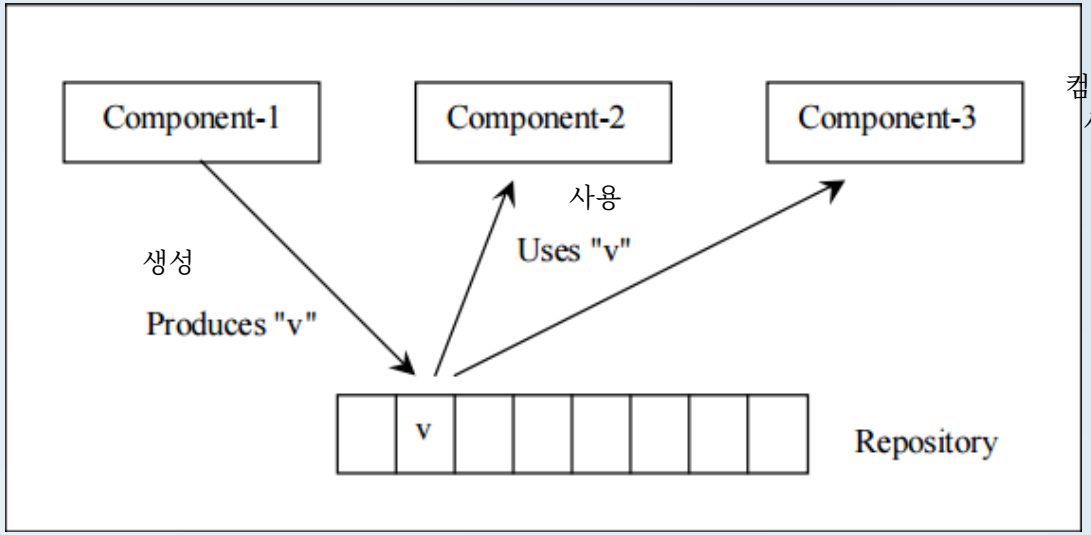
Types of Data-centered Architecture
- Repository
- Blackboard

Repository Architecture Style

저장소를 이용해 작업하는 구조 중 하나!
RDBMS를 쓴다! 도 그 예중 하나

Repository Architecture Style

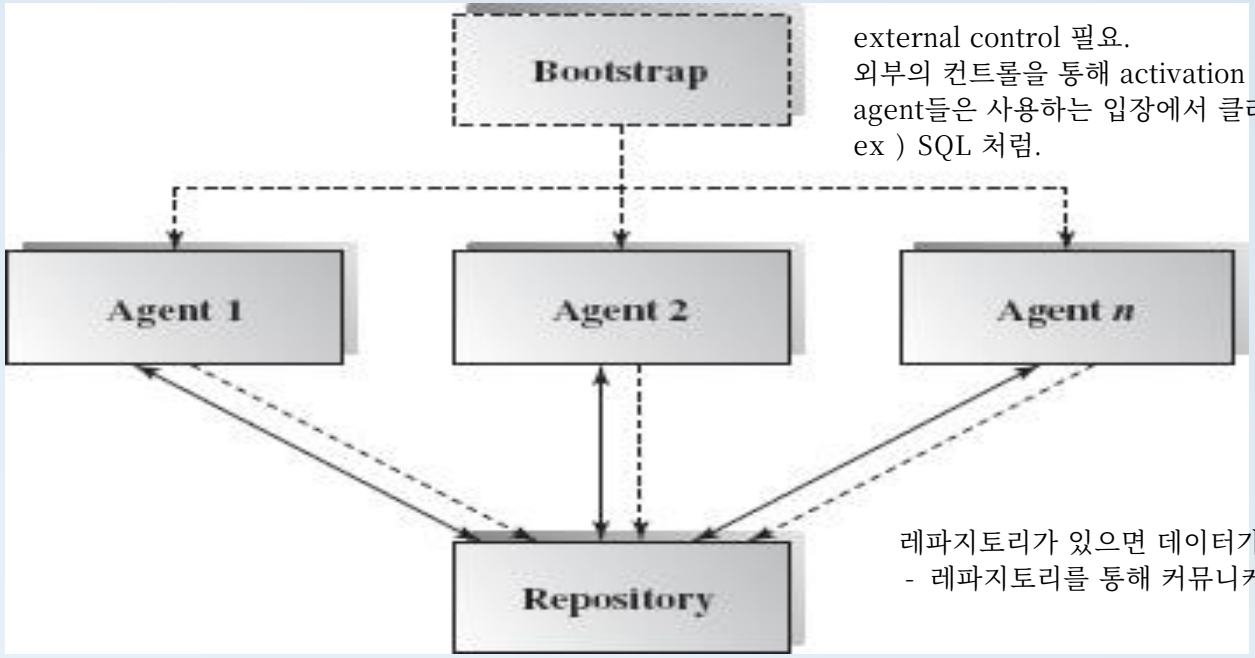
- Synopsys
  - Data-centered architecture that supports user interaction for data processing
    - as opposed to the batch sequential transaction processing discussed earlier
  - A software system made of **several software components** that **need to communicate**, that is **exchange** potentially large and evolving **data**, in order to meet the system requirements



컴포넌트는 레파지토리를 중심으로 서로 커뮤니케이션을 함.

Repository Architecture Style

- Structure
  - The software component agents of the data store control the computation and flow of logic of the system.
  - Different clients may have different interfaces and different data access privileges



external control 필요.
외부의 컨트롤을 통해 activation 하는 구조여야 함.
agent들은 사용하는 입장에서 클라이언트. 접근하는 인터페이스를 맞출 필요도 있음.
ex) SQL 처럼.

레파지토리가 있으면 데이터가 여기에 들어감
- 레파지토리를 통해 커뮤니케이션 하는 형태가 중요.

Repository Architecture Style

- Defines a **model of communication** for software components based on the **use** of **a shared repository**

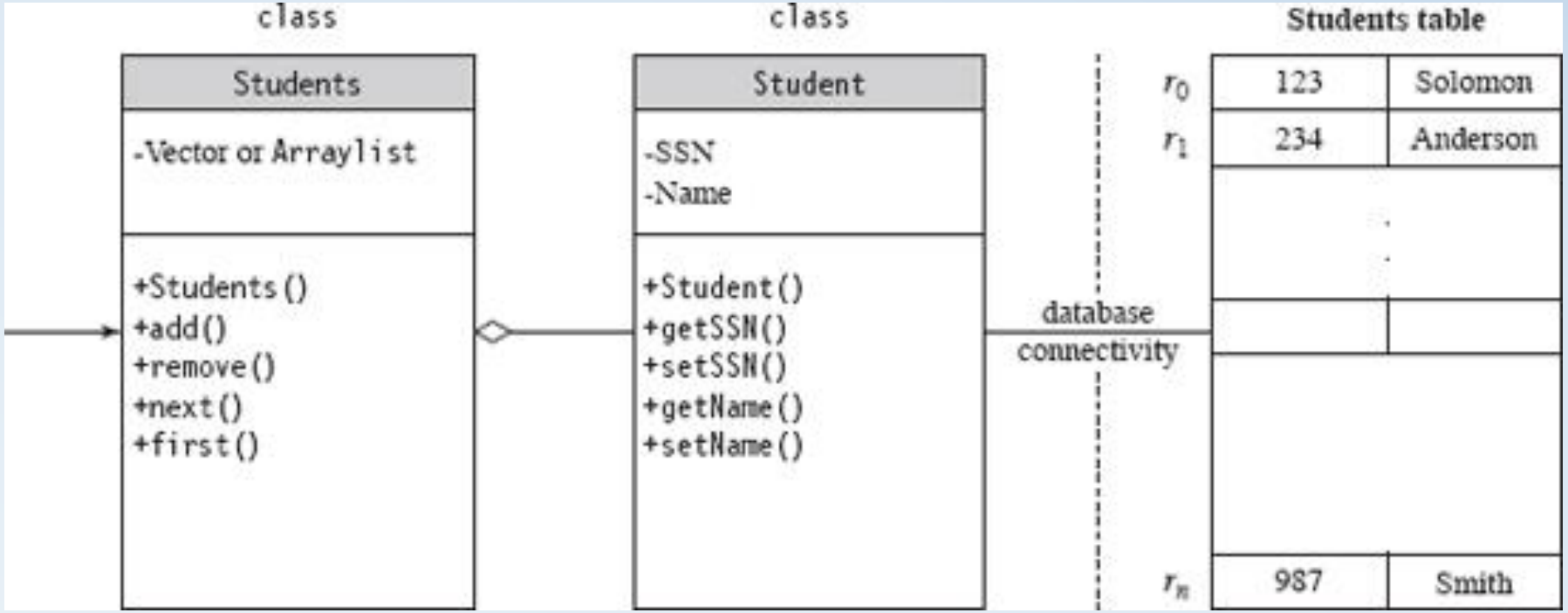
  레파지토리를 중심으로 공유되는 저장소가 있고 다른 컴포넌트들이 접근하고 커뮤니케이션 함.

- Using a data *repository* for communication
  - The *repository* is **known and accessible** by **all the software components** of the system and represents the only means of communication for components

- When a component produces some **information that is of interest for other components**, it **stores** it **in the *shared repository***

Repository Architecture Style

- Example: student data management



<RDBMS>

Repository Architecture: Example

- Example: in java

```
public class Student implements Serializable {
  String SSN;
  String Name;
  public Student() {
      SSN = "";
      Name = "";
  }
  public Student(String ssn, String name) {
      SSN = ssn;
      Name = name;
  }
public void setSSN(String ssn) {
   SSN = ssn;
}
public String getSSN() {
    return SSN;
}
public void setName(String name) {
    Name=name;
}
public String getName() {
    return Name; }
}
```

Repository Architecture: Example

```
. . .

try {


Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"
);

 Connection connection =
DriverManager.getConnection(   "jdbc:odbc:stu
dents");


} catch(Exception e){…}
```

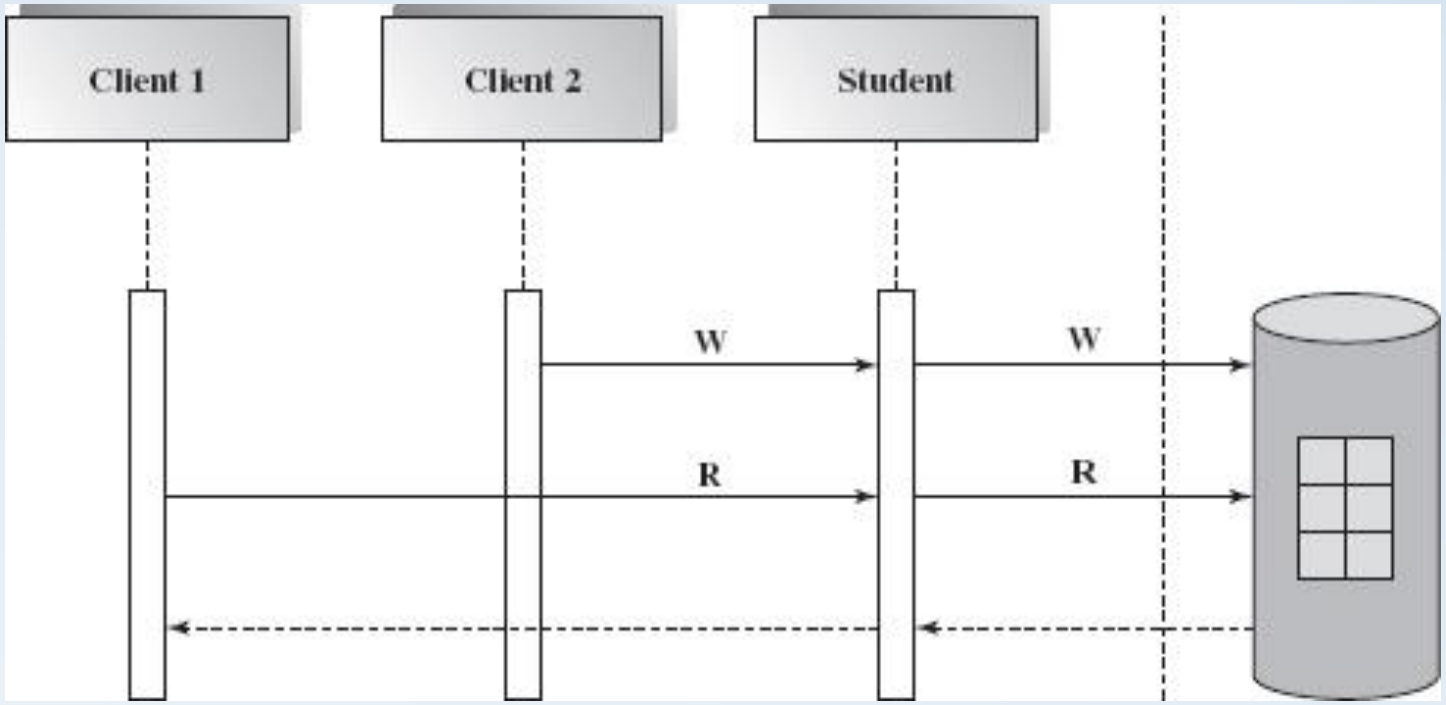
```
. . .

ArrayList studentList = new ArrayList();

Statement statement =
connection.createStatement();


ResultSet results =                    여러개를 관리. 쿼리를 던지고
statement.executeQuery("SELECT * FROM students");

Student student = new Student();


while (results.next()) {        그에 대한 값을 뉴 오브젝트에 넣어줌.

 student.setSsn(results.getSsn(1));

 student.setName(results.getName(2));

 studentList.add(student);

}
```

이 저장소에 다른 종류의 클라이언트가 wr을 하는데 엔티티 빈에 껍데기를 씌워 스토어, 리드를 할 수 있음.

Repository Architecture: Example

5~5:45

- Example: student data management
  - The clients can access the same data with command line interface, GUI interface, program interface, Remote Procedure Call interfaces (RPC), or object-oriented Remote Method Invocation (RMI).

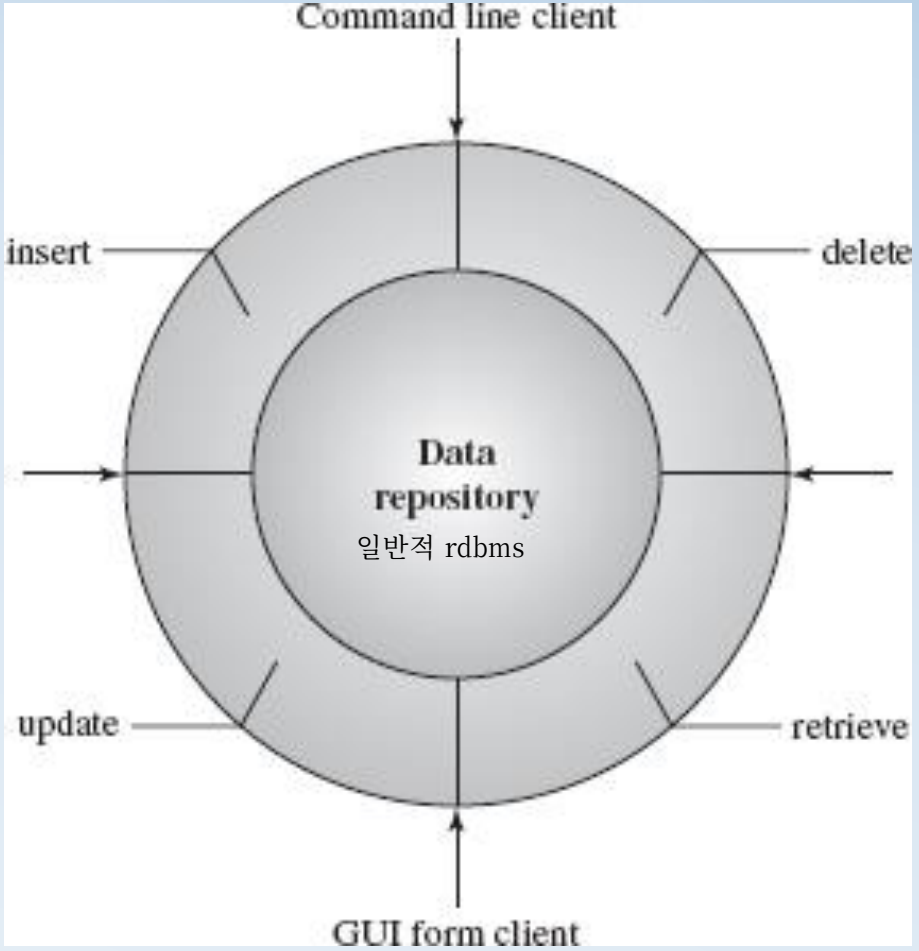


어떤 프로세스는 읽기도 하고
어떤 프로세스는 쓰기도 하고 둘 다 하기도 함

컨텐츠를 담을 수 있는 엔티티 클래스

Repository Architecture: Example

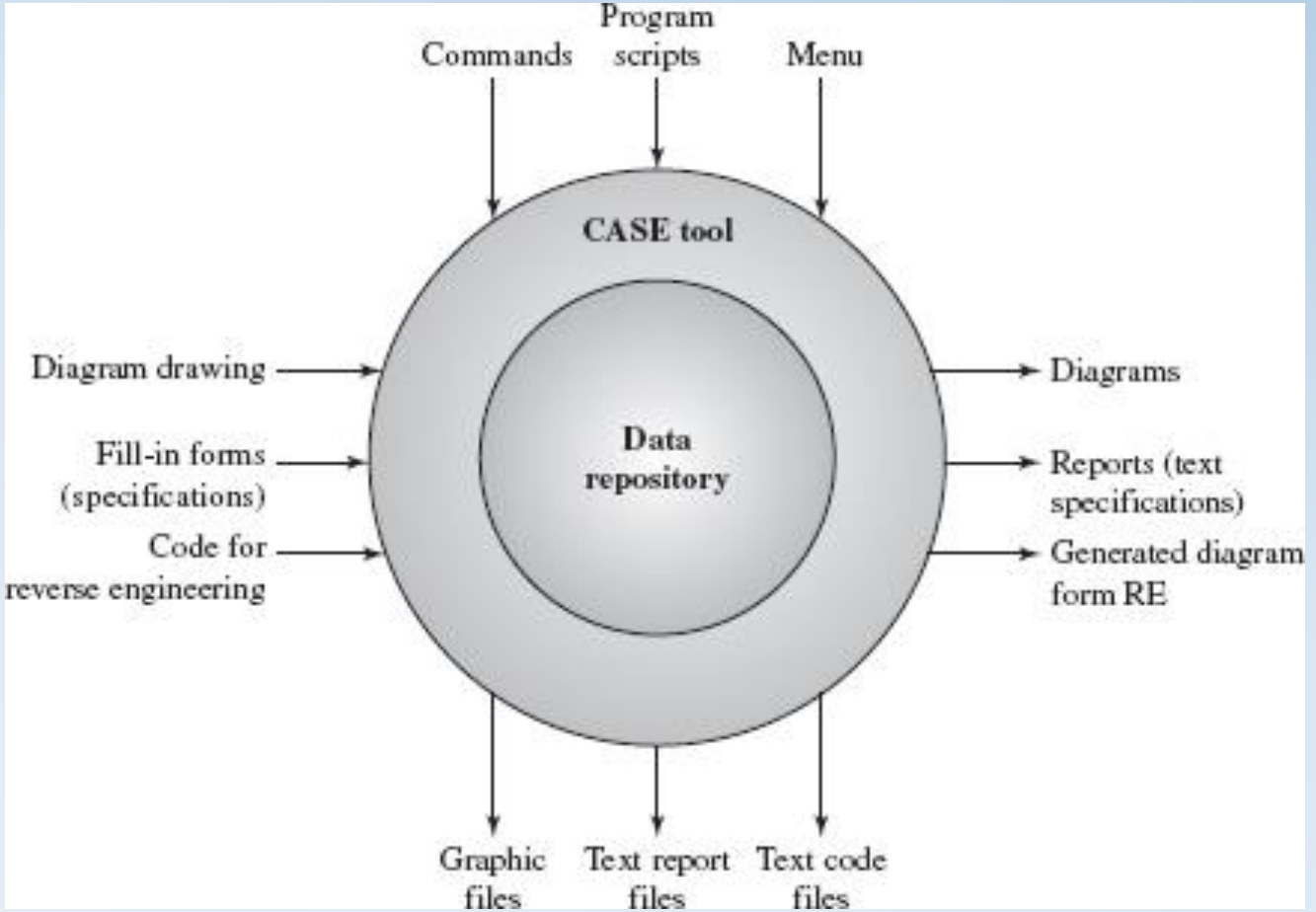
- Application Examples
  - relational database management system



Repository Architecture: Example

case tool

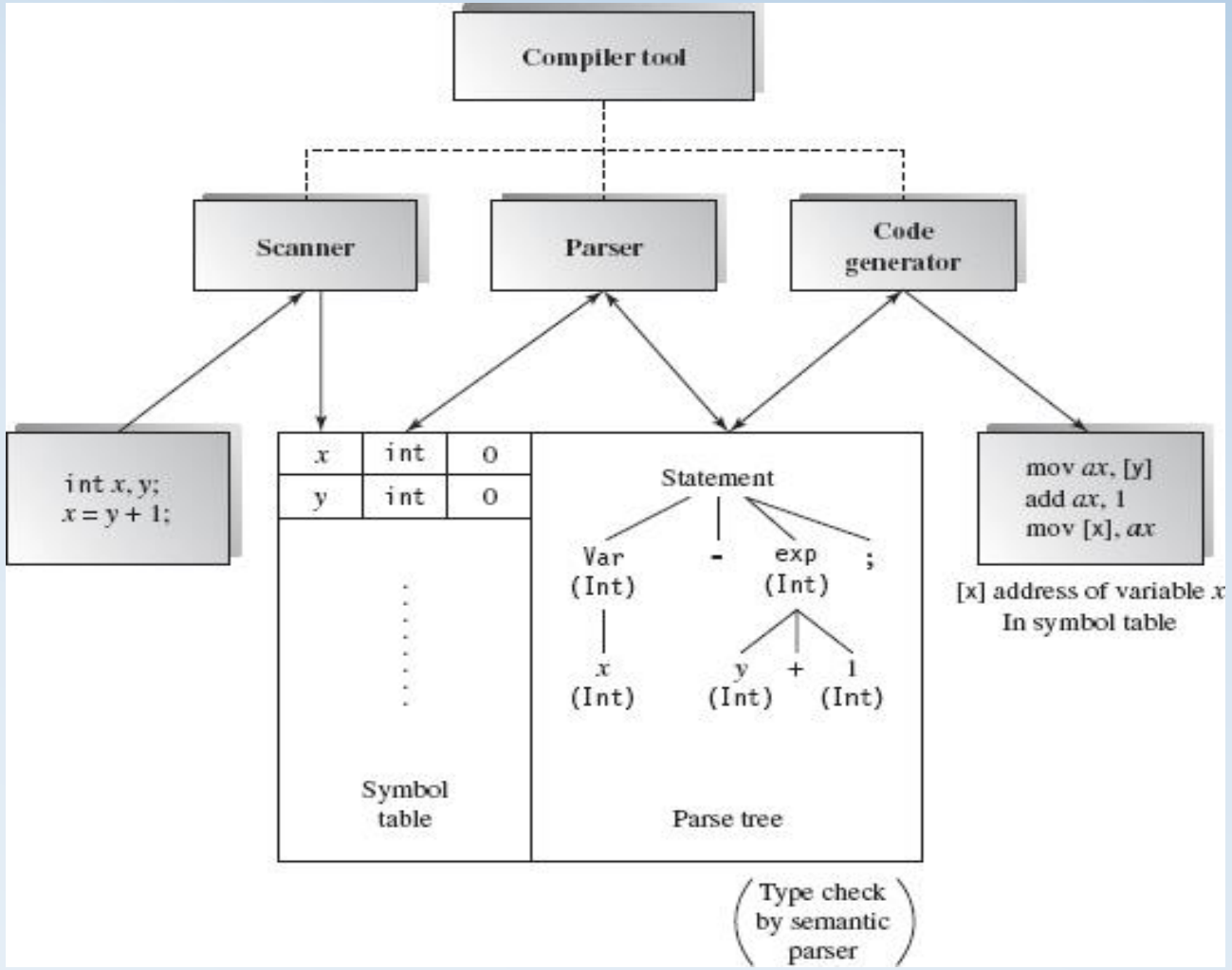
- Application Examples
  - Computer Aided Software Engineering (CASE) system



Repository Architecture: Example

- Application Examples
  - Compiler system

    심볼 테이블을 레파지토리로 생각해볼 수 있음.



Repository Architecture: Example

- Application Examples
  - Compiler system (Cont'd)
    - Every compiler system has
      - own reserved keyword table,
      - identifier symbol table,
      - constant table generated after lexical analysis, and
      - syntax and semantics trees generated by syntax and semantics analysis.
    - the data in memory are shared by all agents
    - the agents don't pass on data to each other directly.

Repository Architecture: Variants

- Variants
  - Virtual repository
    - built up on the top of multiple physical repositories.
    - it can also provide security management of authority privileges in terms of scope of data and types of manipulations for different users or groups.
  - Decentralized (distributed) repository
    - distributed database system
    - enterprise information system
    - all data are distributed over all sites linked by network
    - Data are replicated in order to improve reliability and local accessibility.
    - issues such as vertical or horizontal data partitions, synchronizations of duplicated data, and cost of data transmission on the network
    - collaboration in a distributed transaction is a complicated two-phase transaction commitment.

여러개의 레파지토리 위의 virtual 레파지토리도 있음.
서울 데이터 센터, 대전 데이터 센터가 있음. 대전 - 서울 데이터 센터를 합치는 하나의 버츄얼 레파지토리를 만든다면,
authority privileges : 버츄얼 레파지토리를 깨는 것. 서울만 들어갈 수 있는 권한만 있어도 대전으로 들어갈 수 있음
단점 : 뚫리는 경우 "접근권한"을 고려해야 함.

흩어져 있는, 분산된 데이터베이스 시스템(레파지토리) : 통상 큰 기업에서는 DB를 여러개 둠. availability를 최대화 시킬 때, 디비를 여러개 두는데, 이 때 수직 구조로
DB를 duplication을 함. 한 디비가 한 세트가 되도록 함. 수직으로 할지 수평으로 할지는 상황마다 다름, 조직의 니즈, 보안(높으면벌티컬), 관리에 따라 달라짐.
호리젠탈 : 똑같은 세트
벌티칼 : 세로로 자름. 여행이라 치면 호텔 따로 비행기 따로 이런식

가장 좋은건 distributed transaction을 함. 서로 다른 오퍼레이션을 마치 하나처럼 트랜즈엑션으로 묶어준다는 것! - - >
On(transaction commit) or nothing(rollback)이 장점! (예시 : 계좌이체). Atomic(아토믹) 구조로 만든다는 것.

Repository Architecture: Domains

- Applicable domains
  - Suitable for large, complex information systems where many software component clients need to access them in different ways
  - Requires data transactions to drive the control flow of computation

- Related Architecture
  - Layered, multi-tier, and MVC

Repository Architecture: Benefits & Limitations

- Benefits
  - Data integrity: easy to back up and restore
  - System scalability and reusability of agents: easy to add new software components because they do not have direct communication with each other
  - Reduces the overhead of transient data between software components
- Limitations:
  - Data store reliability and availability are important issues.
    - Centralized repository is vulnerable to failure compared to distributed repository with data replication.
  - High dependency between data structure of data store and its agents.
    - Changes in data structure have significant impacts on its agents.
    - Data evolution is more difficult and expensive.
  - Cost of moving data on network if data is distributed.

데이터 무결성 : 데이터에 결함이 생길 수가 없음.
데이터에 결함이 생기면 알아서 버리게 됨. CRC, hash, parity bit 등등을 넣어줌. 깨졌다면 로그를 따라가서 안깨진 것을 가져오게 되어 롤백됨.

저장소를 주변으로 에이전트들이 접근됨, 에이전트들을 필요에 따라 scale - out(확장성)할 수 있음. 재사용 할 수 있다.

transient : 중간에 진행되는 데이터.
가끔 시스템에 대한 상태 정보를 글로벌하게 스냅샷 찍는 경우가 있음. 중간과정의 데이터가 오버헤드를 만드는 경우가 있는데 이를 줄여줌. 저장된 데이터만 관리하기 때문에 간편하당.

데이터를 이런식으로 저장함!! 읽으심.

데이터를 사용하는 에이전트가 서로 연관되어 있음. 데이터 저장소 특성에 따라 에이전트가 읽어들여야 한다. 이런게 고려사항!.

데이터가흩어져 있는 경우 고려해야한다.