DB02

Data models

Database의 구조를 설명하는 개념의 집합

- Database의 구조/특성을 잘 표현하는 것
- Data types, constraint, relations 등이 포함된다.

Data abstraction을 얻는 방법 중 하나

- Data model을 통해 Database나 Data를 대강 확인할 수 있다.

Database의 Design 과정은 다음과 같다.

1. Application에 사용할 **Database의 구조를 결정**한다.
2. 해당 구조를 **Data model로 표현**한다.

Data model은 크게 다음 3가지로 나뉜다.

1. Physical data model

Data가 어떻게 구성되고 **Storage media(Harddisk, ssd)에 실제로 어떻게 저장이 될 지에 대한 자세한 설명**

- Physically stored 방식에 대한 설명
- **Low-level data model**

Provide record-related information (Example)

- **Type (타입):** 레코드가 저장되는 파일의 구조나 타입

- **Index (인덱스):** 데이터 검색 속도를 높이기 위해 인덱스(색인)를 사용할지, 사용한다면 어떤 종류의 인덱스를 사용할지를 정의
- **Access path (접근 경로):** 특정 레코드를 찾기 위해 **실제로 어떤 경로와 방법**을 사용할지

2. Conceptual data model

사람의 입장에서 Data가 어떻게 표현되는지를 설명하는 방법

- 사람이 Data를 이해하는 방식과 유사하다.
- **High-level data model**

EX) Entity-Relationship model

3. Representational data model

Physical data model과 Conceptual data model의 사이에 위치하는 Data model

- Physical data model에 비해 사용자가 이해하기 쉽다.
- Data가 Storage media에 어떻게 구성되는지를 나타내기도 한다.
- **Implementation data models**라고도 한다.

DBMS에서 가장 통상적으로 사용되는 방법이다.

EX) Relational model, Network model, Hierarchical model

Database Schema (or meta-data)

Data model에 의해 표현된 Database structure

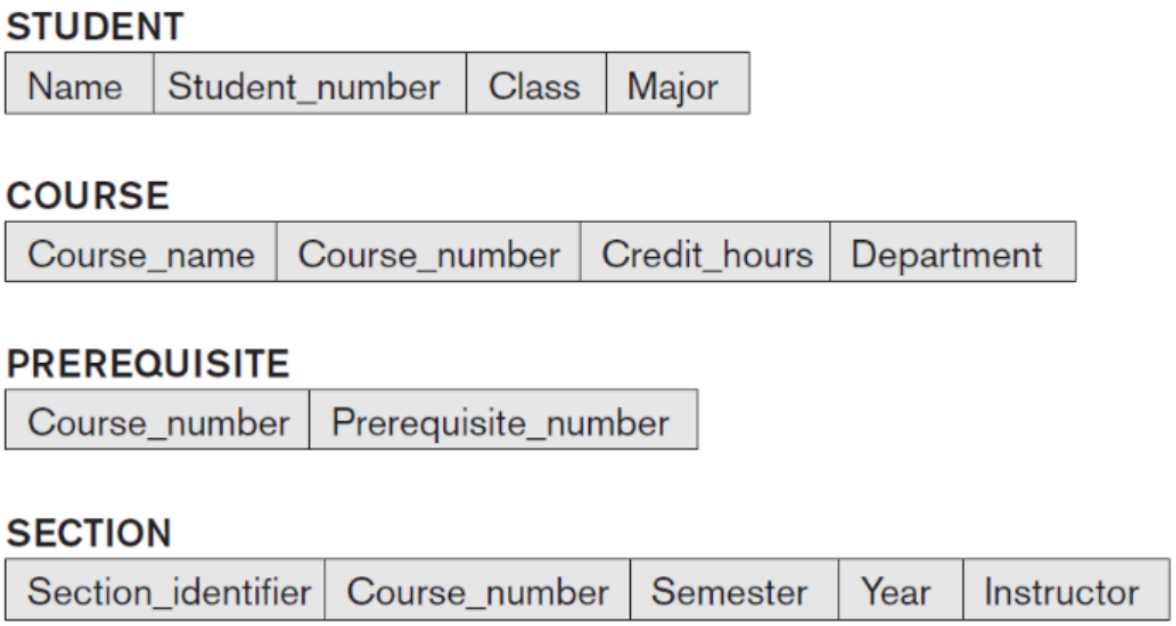
- **Intense**라고도 불린다.

Database designer에 의해 결정된다.

Schema자체가 바뀌는 경우는 거의 없다.

- 바뀌는 부분은 Data의 값이다.

Schema diagram: Schema를 **그림(diagram)으로 시각화**



Instances

특정 Database 안에 특정 시간에 저장되어져 있는 **Data의 값**

- 특정 시간이 중요한 이유는 Data의 값이 짧은 시간마다 바뀌기도 하기 때문이다.

Database state, snapshor, occurence, extense라고도 불린다.

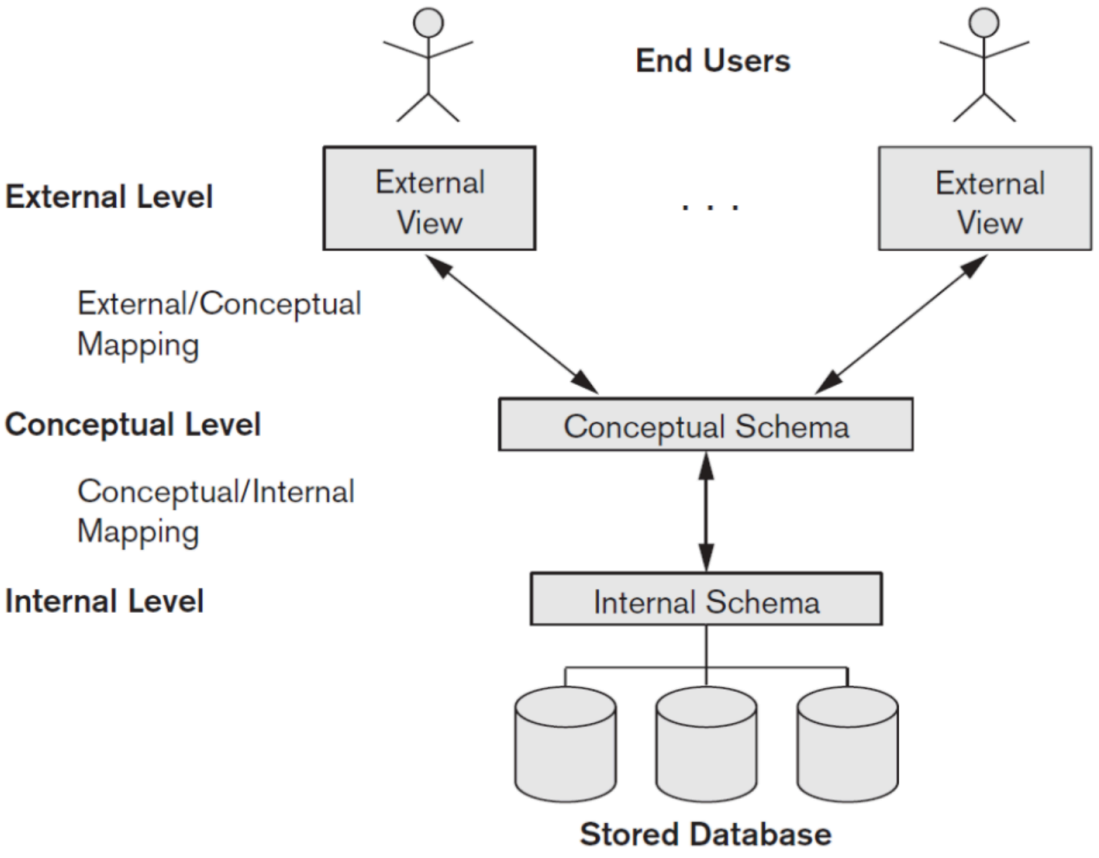
Database내의 Data의 Insertion, update, deletaion에 의해 자주 바뀐다.

Three-Schema Architecture

3가지의 Database schema를 갖는 구조를 통해 Data independence를 보장하는 것

- Application과 Database가 서로 독립이 되도록 보장한다.

Database designer는 **(1) 어떤 Data를 Database에 저장**할 지를 결정하고, **(2) Database의 Structure, schema를 결정**한다.



- **External view**: 전체 Data 중 사용자가 보는 **일부** Database
  - Database의 일부에 대한 Schema이다.
  - Program은 External view를 기준으로 만들어진다.
  - Conceptual model / Representational data model

    - User는 External view대로 Database가 저장되어 있다고 생각한다.
- **Conceptual Schema**: 사람이 이해할 수 있는 수준에서 표현하지 위해 사용하는 Schema
    - Internal view가 너무 Low-level이기 때문에 사람이 이해할 수 있도록 돕는다.
    - 전체 Database에 대한 Schema이다.
    - External view의 model과 동일해야 한다.
    - Conceptual model / Representational data model
        - DBMS가 보통 Representational data model을 사용하기 때문에 Representational data model을 사용하는 경향이 있다.
- **Internal Schema**: Physical model에 의해 Storage media에 Data가 어떻게 구체적으로 저장되어야 할 지를 표현한 것
    - 전체 Database에 대한 Schema이다.
    - Physical data model에 의해 표현된다.
    - External view, Conceptual schema와 **Data model이 반드시 달라야 한다.**

Independence는 어떻게 유지?

1. Internal schema가 변경되면? (EX. Data search 방법이 B-tree → Hash)
    - Internal schema의 Meta-data만 변경되면 된다.
2. Conceptual schema가 변경되면?
    - 수정된 Data를 External에서 실제로 사용하면 Program도 수정이 필요하다.
    - 수정된 Data를 External에서 실제로 사용하지 않는 경우 Internal과 Conceptual만 수정이 필요하다.
        - Meta-data를 수정한다.
    - Internal과 Conceptual은 항상 수정이 필요하다.

Internal level

전체 database의 **Physical storage structure**을 설명한 것

- **Physical data model**에 의해 표현된다.

Physical database는 Internal schema에 설명된 방식으로 저장된다.

Conceptual level

전체 database의 **logical structure**을 표현한 것

- Entity, data type, relations, operations 등

Data의 Physical storage structure를 숨겨 **Abstraction**을 얻는다.

High-level data model

- Conceptual data model
- Representational data model
    - DBMS가 이걸 사용해 보통 위 model을 주로 사용한다.

External or view level

전체 database의 **logical structure 중 일부분**을 표현한 것

- 사용자가 필요로 하는 부분만 보이고 나머진 숨긴다.

사용자는 **External Schema**에 기술된 방식대로 **Logical / Conceptual database를 참조**한다.

High-level data model

- Conceptual data model
- Representational data model
    - DBMS가 이걸 사용해 보통 위 model을 주로 사용한다.

Mapping 과정

1. User query는 External level schema를 위한 것이다.
2. **User query를 DBMS가 Conceptual level schema로 Mapping**한다.
    - External / Conceptual mapping
3. **DBMS가 Conceptual level schema를 Internal level schema로 Mapping**한다.
    - Conceptual / Internal mapping
4. Stored database에서 결과물을 얻는다.
5. (4)의 **결과물을 External view에 맞게 Mapping**한다.

Data Independence

한 Level의 Schema를 변경할 수 있는 것

Three-level schema에서 **한 레벨에서 Schema를 변경하더라도, 그 상위 레벨의 Schema나 응용 프로그램을 고칠 필요가 없는 것을 의미**한다.

DBMS의 가장 중요한 특징 중 하나로 두 가지 종류로 나뉜다.

- Database의 Schema가 변경되어도 Program은 변경될 필요가 없도록 보장한다.
- **Application program의 유지를 위한 Overhead를 줄인다.**

1. Logical data independence

Conceptual shcema를 변경하는 것

External schema는 변경할 필요가 없다.

- Conceptual schema에서 External schema와 관련없는 Data가 변경된 경우 수정할 필요가 없다.
- 이 경우, External schema가 변경되지 않기 때문에, **Program을 Rewrite할 필요가 없다**.

2. Physical data independence

Internal schema를 변경하는 것

Conceptual schema, external schema, application 모두 수정할 필요가 없다.

DBMS Languages

DBMS에 의해 제공하는 언어로 Database를 control/manage 한다.

Data Definiation Langauge (DDL)

- 원하는 Database에 대해 **External, conceptual schema를 정의**한다.

Data Manipulation Language (DML)

- 사용자가 data를 검색, 삽입, 삭제, 수정할 수 있도록 해준다.

Storage Definition Language (SDL)

- 원하는 Database에 대해 **Internal schema**를 정의한다.

View Definition Language (VDL)

- 사용자의 Veiw가 Conceptual schema로부터 **어떻게 만들어지는지 그 연결(매핑) 방법**을 정의한다.

DML의 종류

1. **Precedural DML**
    - 사용자의 Query를 **Step-by-Step / 구체적**으로 작성
    - **어떻게** Data를 가져오는지에 중점을 둔다.

2. **Non-precedural DML**
    - 사용자의 Query를 **Declaratively**하게 정의한다.
    - **어떤** Data를 가져오는지에 중점을 둔다.
    - 언어 개발자에게는 어렵고, 사용자에게는 편하다.

DBMS Interfaces

1. **Menu-based interfaces**
    - Language를 이용해 Raw query를 만드는 대신, Menu에서 제공하는 Sample에서 골라낸다.

2. **Graphical user interfaces**
    - Database schema의 Graphical representation을 조절하여 Database에 접근한다.

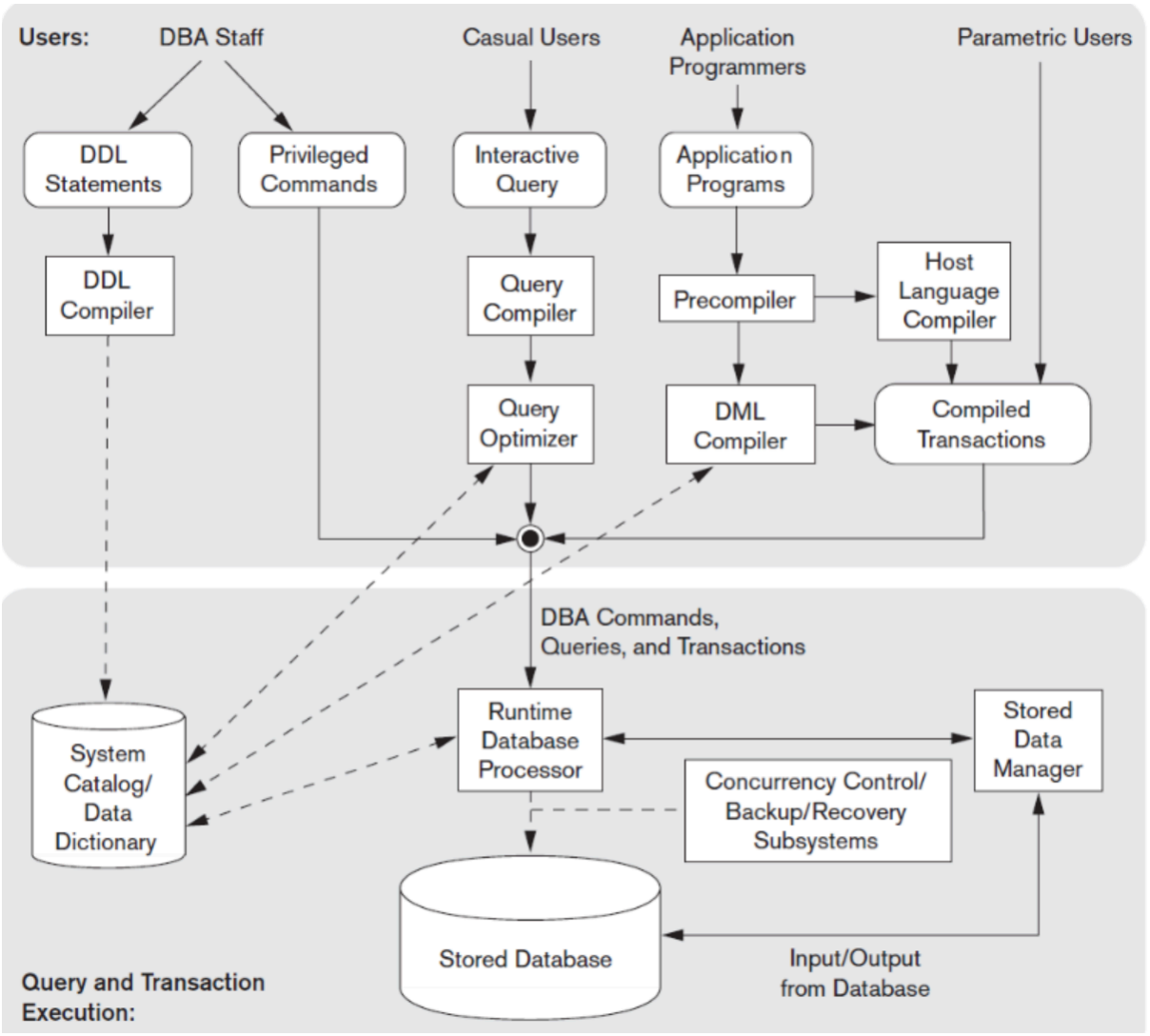
3. **Forms-based interfaces**
    - 기본 틀을 주고 그 안을 채워 넣는 방식으로 Database에 접근

4. **Natural language interfaces**

- Natural language style로 Database에 접근

5. **Interfaces for DBAs**
    - DBMS Management 환경을 제공

DBMS Component Modules



Stored data manager: Storage에 저장된 모든 data를 관리한다.

- Meta-data와 User data 모두 관리한다.

DDL Compiler: DDL의 Schema definition을 처리한다.

- DBMS의 System catalof에 **schema-related meta data를 저장**한다.
- Conceptual schema / Internal schema에 대응되는 각각의 Meta-data 모두 System catalog에 저장된다.

Runtime database processor: User query에 맞게 database에 접근할 수 있도록 해준다.

Query compiler (Iterative SQL)

- User가 SQL로 Query를 작성
- High-level DBMS Query를 Parsing
- 사용자의 요구에 맞는 Runtime database processor를 호출

Precompiler: Application에서 DML Commands만을 골라낸다.

DML Compiler: SQL을 low-level function call sqeunce(Object code)로 바꿈

Previleged Commands: System의 상태 모니터링을 하기 위한 명령어

Java + SQL 프로그램 실행 (Precompiler 방식) 과정

1. 전체 Program이 Java와 SQL로 이루어져 있다고 가정하자.
2. Precompiler가 위 Program에서 SQL 부분만 뽑아낸다.
3. DML Compiler가 SQL을 function call의 sqeunce로 바꾼다.
    - JAVA API 형태
4. (3)의 결과로 SQL이 java function call이 되고 전체 Program은 SQL 없이 Java로만 이루어진다.
5. Java compiler가 Program을 compile하고 실행한다.

DDL을 이용한 스키마 정의 과정

1. Database designer가 DDL을 이용하여 Schema에 대한 정의를 한다.
2. DDL compiler가 Schema를 이해한다.
3. Meta-data 형태로 Schema를 저장한다.
4. DDL 컴파일러가 Stored data manager에게 이 meta-data를 system catalog에 저장해 달라고 요청한다.
5. Storage manager는 DDL을 이용하여 Meta-data를 System catalog에 저장한다.

사용자의 Query 처리 과정

1. 사용자의 Query를 Query compiler가 해석한다.
2. 해석한 후 최적의 실행 계획(Optimization)을 찾는다.
3. (2)의 최적의 실행 계획을 Runtime database processor에게 전달한다.
4. Runtime database processor는 실행 계획을 실제로 실행하며 Storeged manager를 통해 Database에 직접 접근한다.

Database System Utilities

Database Utilities: **DBA가 쉽게 Database system을 관리**할 수 있도록 해주는 것

1. Loading

기존 data file을 DBMS의 database에 load할 수 있도록 해주는 방법

- 자동으로 DBMS가 요구하는 Database 형식에 맞게 파일을 변환한다.
- 새로운 Database를 Build하는데 유용하다.

2. Backup

Database의 **Backup copy본**을 생성하는 것

- Database의 System failure에 대비한다.

3. Database storage reorganization

DBMS의 성능을 높이기 위해 **Database의 file을 다른 file 형식으로 변환**하는 것.

- **Internal schema의 구조를 바꾸는 방법**이다.
- index 방법을 바꾸는 등이 해당된다.

4. Performance monitoring

- **Database의 사용량을 추적하고 통계를 DBA에게 전달**한다.
- Reorganization이 필요한지 결정을 내리는 데에 도움을 준다.

Classification of DBMS

먼저 DBMS가 지원하는 **Data model에 따라 구분할 수 있다.**

1. Relational model (Representation model)

- Database가 Tables의 집합으로 표현된다.
- 각 Table은 records를 가진다.
- 최근에 많이 사용된다. Oracle, MSSQL이 사용하는 방법이다.

2. Network model

- Data를 Record type과 1:N set type으로 나타낸다.

3. Hierarchical model

- Data를 Hierarchical tree 구조로 나타낸다.

4. Object model

- Database를 Class의 집합으로 표현한다.
- Data는 Class에 속한 Object로 표현한다.
- Modeling feature가 풍부하여 Complex object를 다룰 수 있다.

두 번째로 **DBMS를 사용하는 User의 수**에 따라 나눌 수 있다.

- **Single user DBMS**
- **Multi-user DBMS**

세 번째로 **Sites의 수**에 따라 나눌 수 있다.

- **Sites: 물리적 위치, 컴퓨터의 수**
- **Centralized DBMS**: DBMS가 하나의 Machine에만 존재한다.
  - 모든 User가 하나의 Machine에 접속해야 한다.
  - Bottleenck이 되어 성능이 감소한다.
- **Distributed DBMS**: DBMS가 다른 Machine에 존재한다.
  - **Homogenous DBMS**: 모든 DBMS가 다 같은 Model을 가짐
  - **Heterogenous DBMS**: DBMS가 다른 Model을 가짐

마지막으로 **목적**에 따라 나눌 수 있다.

- **General purpose DBMS**: 특별한 목적없이 구현된 DBMS
- **Special purpose DBMS**: 특별한 목적으로 구현된 DBMS
  - Real-time DBMS