전북대
소중해유(You)

데이터베이스 기초:
SQL과 NoSQL 비교, 기초적 사용

전북대학교 JEONBUK NATIONAL UNIVERSITY
전북대학교 SOFTWARE중심대학사업단
SW중심대학
과학기술정보통신부
IITP 정보통신기획평가원

목차

데이터베이스란?

데이터 (Data)?

- 사실, 숫자, 문자, 기호 등의 원시적이고 가공되지 않은 자료
- 데이터는 다양한 형태로 존재할 수 있으며, 컴퓨터 시스템에서 처리되고 저장 됨
- 데이터는 구조화된 형태(예: 데이터베이스의 테이블)와 비구조화된 형태(예: 텍스트 문서, 이미지) 모두에서 발견될 수 있음

정보 (Information)?

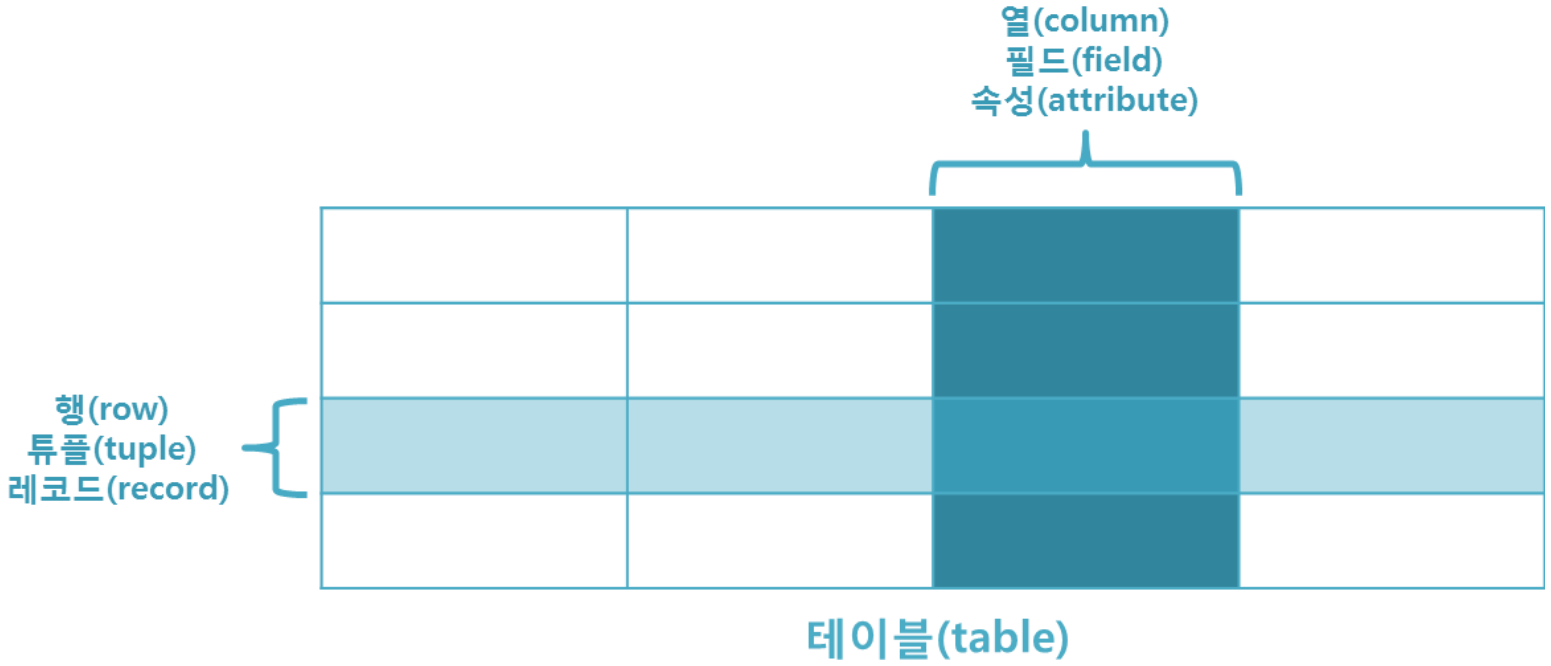
- 데이터를 가공하고 분석하여 특정 의미나 목적을 가진 유의미한 내용으로 변환된 형태
- 데이터에 맥락을 부여하거나 관계를 설명하여 특정 질문에 대한 답을 제공할 수 있는 형태로 데이터를 변환한 결과 [예시: "지난 7일간 평균 기온은 25°C" (단순한 온도 데이터를 종합하여 평균 기온이라는 정보를 도출)]

데이터베이스 (Database) ?

- 데이터를 체계적으로 저장하고 관리하는 집합체
- 여러 사용자나 애플리케이션이 데이터를 효율적으로 저장, 검색, 업데이트, 삭제할 수 있도록 설계된 구조를 가지고 있음.
- 일반적으로 데이터베이스는 데이터를 논리적이고 구조화된 방식으로 저장하며, 이를 통해 데이터 간의 관계를 유지하고 관리할 수 있음.

데이터베이스(Database)의 구조

- 데이터베이스는 보통 테이블, 열, 행으로 구성
- 각 테이블은 특정한 데이터 유형(예: 고객 정보, 제품 정보 등)을 저장
- 열은 특정 속성(예: 이름, 나이, 가격 등)을 나타내고, 행은 하나의 데이터를 나타냄.



데이터베이스(Database)의 종류

- 데이터베이스는 크게 **관계형 데이터베이스(Relational Database, RDB)**와 **비관계형 데이터베이스(NoSQL Database)**로 구분
- 관계형 데이터베이스는 테이블 간의 관계를 정의
- 비관계형 데이터베이스는 문서, 키-값, 그래프 등 다양한 데이터 구조를 지원

데이터베이스란?

데이터베이스(Database)의 특징

1. 데이터의 **중복성**을 최소화 할 수 있음
2. 데이터의 **일관성**을 유지할 수 있음 (Consistency)
3. 데이터의 **무결성**을 유지할 수 있음 (Integrity)
4. 데이터의 **독립성**을 유지할 수 있음
5. 데이터의 **공유성**을 최대화 할 수 있음
6. 데이터의 **보안성**을 보장할 수 있음
7. 데이터를 **표준화**하여 관리할 수 있음

데이터베이스 관리 시스템 (Database Management System, DBMS)

- 데이터베이스를 생성하고 관리하는 소프트웨어 시스템
- 데이터베이스의 사용자와 데이터베이스 자체 간의 인터페이스 역할을 하며, 데이터를 정의, 저장, 검색, 업데이트하는 작업을 지원
- DBMS를 사용하면 사용자는 데이터를 더 효율적이고 일관되게 관리할 수 있다는 장점을 지님.
- **기능**
  - **자료 정의 및 저장 관리 기능**: 디스크 기억 장치에 DB 공간을 할당하여 물리적인 저장 구조를 만들고 이에 접근
  - **질의 처리 및 transaction 기능**: DB에 저장되어 있는 데이터를 처리하기 위해 사용자가 입력한 요구 사항들을 번역/실행 (SQL)

대표적인 DBMS

DBMS	제작사	작동 운영체제	기타
MySQL	Oracle	Unix, Linux, Windows, Mac	오픈 소스(무료), 상용
MariaDB	MariaDB	Unix, Linux, Windows	오픈 소스(무료), MySQL 초기 개발자들이 독립해서 만듦
PostgreSQL	PostgreSQL	Unix, Linux, Windows, Mac	오픈 소스(무료)
Oracle	Oracle	Unix, Linux, Windows	상용 시장 점유율 1위
SQL Server	Microsoft	Windows	주로 중/대형급 시장에서 사용
DB2	IBM	Unix, Linux, Windows	메인프레임 시장 점유율 1위
Access	Microsoft	Windows	PC용
SQLite	SQLite	Android, iOS	모바일 전용, 오픈 소스(무료)

DB의 종류 및 발전 과정

- 1960년대: 계층형 DB / 네트워크 형 DB
- 1970년대: 관계형 DB
- 1980년대: 객체 지향형 DB
- 1990년대: 객체 관계형 DB
- 2000년대 이후: NoSQL

관계형 데이터베이스? (RDB/ SQL)

- **관계형 데이터베이스 (Relational Database, RDB)**
  - 데이터는 정해진 데이터 스키마(=structure)를 따라 데이터베이스 테이블에 저장
  - 데이터는 관계를 통해서 연결된 여러 개의 **테이블**에 분산
- **스키마 (Schema)**
  - 데이터베이스에서 데이터를 조직화하고 구조화하는 방식, 즉 데이터의 구조, 형식, 제약 조건 등을 정의하는 설계도
  - 데이터베이스의 논리적인 설계를 나타내며, 테이블, 필드, 데이터 유형, 관계, 인덱스, 뷰, 제약 조건 등 데이터베이스의 구조적 요소를 정의
- **SQL (Structured Query Language)**
  - SQL은 Structured Query Language의 약자로 **관계형 데이터베이스**와 상호작용하는데 사용하는 쿼리 언어
  - SQL을 사용하면 RDBMS에서 데이터를 저장, 수정, 삭제 및 검색 할 수 있음

관계형 데이터베이스 용어

- **테이블 형태의 데이터 구조**

- **테이블의 특징**
  - Relation 혹은 Entity (개체)라고도 함
  - 열(column) / 행 (row)으로 구성

- **테이블의 구성 조건**
  - 테이블에 포함된 행들은 유일해야하며, 중복된 행이 존재하지 않아야 함
  - 테이블에 포함된 행들 간에는 순서가 존재하지 않음
  - 테이블을 구성하는 열들 간에는 순서가 존재하지 않음

- **필드의 특징**
  - 속성 (attribute) 혹은 열 (column)이라고도 함
  - 테이블의 열에 해당
  - 데이터 값을 기억하는 기억 단위
  - 차수(degree): 테이블에 포함된 필드의 개수

관계형 데이터베이스 용어

- **테이블 형태의 데이터 구조**

열 (Column)

- **정의**: 테이블 내의 특정 속성(attribute) 또는 필드를 나타내며, 각 열은 데이터의 한 가지 유형(예: 이름, 이메일, 날짜)을 정의하고, 모든 행이 이 열에 해당하는 데이터를 가짐
- **데이터 유형**: 각 열은 특정 데이터 유형/타입을 가짐 (VARCHAR, INT, DATE, DECIMAL 등)

행 (Row)

- **정의**: 테이블 내에서 하나의 레코드(record) 또는 튜플(tuple)을 나타내며, 각 행은 테이블의 모든 열에 대해 하나의 데이터를 가짐.
- 행은 테이블 내에서 데이터를 개별적으로 식별할 수 있는 요소로 활용

키 (Row)

- **정의**: 키는 테이블 내에서 데이터의 무결성을 유지하고, 테이블 간의 관계를 정의하기 위해 사용되는 열 또는 열의 조합.
- 키는 테이블의 각 행을 고유하게 식별할 수 있도록 도와 줌.

관계형 데이터베이스 용어

- 스키마 예시

```sql
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Birthdate DATE,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    UserID INT,
    OrderDate DATE,
    TotalAmount DECIMAL(10, 2),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

스키마 (Schema)

- 데이터베이스에서 데이터를 조직화하고 구조화하는 방식, 즉 데이터의 구조, 형식, 제약 조건 등을 정의하는 설계도
- 데이터베이스의 논리적인 설계를 나타내며, 테이블, 필드, 데이터 유형, 관계, 인덱스, 뷰, 제약 조건 등 데이터베이스의 구조적 요소를 정의

스키마의 중요성

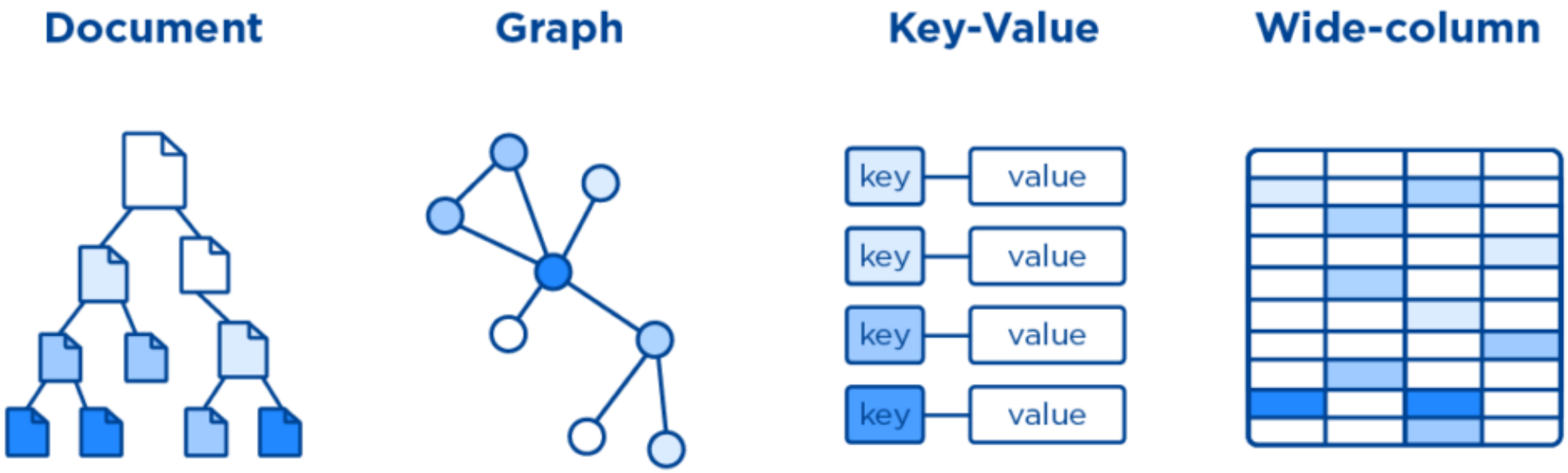
- **데이터 무결성**: 스키마는 데이터의 무결성을 유지하는 데 중요한 역할을 합니다. 예를 들어, NOT NULL 제약 조건을 사용하면 필수적인 데이터가 누락되지 않도록 보장할 수 있습니다.
- **데이터베이스 설계**: 스키마는 데이터베이스의 설계를 정의하므로, 잘 설계된 스키마는 데이터베이스의 성능과 확장성을 높일 수 있습니다.
- **데이터 관계 정의**: 스키마는 테이블 간의 관계를 정의하여, 복잡한 데이터 구조를 명확히 하고, 데이터 일관성을 유지합니다.
- **보안**: 스키마는 사용자 권한과 접근 제어를 설정하는 데도 사용됩니다. 특정 스키마에 대한 접근 권한을 제한하여

NoSQL?

- **NoSQL이란?**
  - NoSQL은 기본적으로 SQL (관계형 데이터베이스)와 반대되는 접근 방식을 따르기 때문에 지어진 이름
  - 스키마와 관계가 없음!!
  - NoSQL 세상에서는 레코드를 문서(documents)라고 정의 함

- **NoSQL의 주요 특징**
  - SQL에서는 정해진 스키마를 따르지 않으면 데이터를 추가할 수 없지만, **NoSQL에서는 다른 구조의 데이터를 같은 컬렉션(= SQL에서의 테이블)에 추가할 수 있음**
  - 문서는 JSON 데이터와 비슷한 형태를 가지고 있고 스키마에 대해 신경 쓸 필요가 없다..!!
  - 일반적으로 관련 데이터를 동일한 컬렉션에 넣어 저장 => 관계형 데이터베이스처럼 여러 테이블에 나누어 담지 않음.

NoSQL 데이터베이스 유형



NoSQL의 종류

- **key-value 타입 (Json 호환)**
  - 속성을 Key-Value의 쌍으로 나타내는 데이터를 배열의 형태로 저장
  - 여기서 Key는 속성 이름을 뜻하고, Value는 속성에 연결된 데이터 값을 의미
  - Redis, Dynamo 등이 대표적인 Key-Value 형식의 데이터베이스

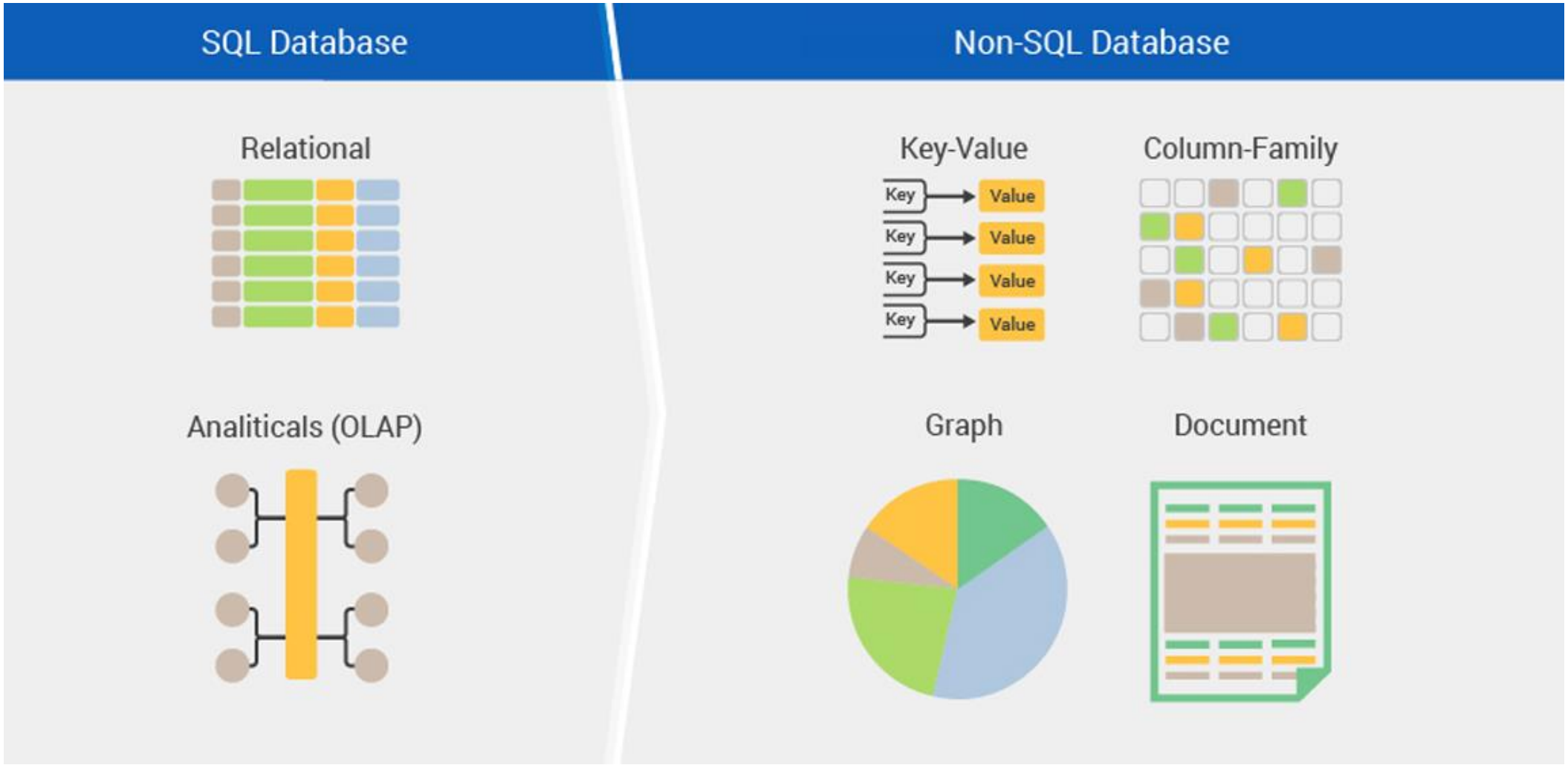
- **문서형(Document) 데이터베이스**
  - 데이터를 테이블이 아닌 **문서처럼 저장하는 데이터베이스**를 의미
  - 많은 문서형 데이터베이스에서 JSON과 유사한 형식의 데이터를 문서화하여 저장
  - 각각의 문서는 하나의 속성에 대한 데이터를 가지고 있고, 컬렉션이라고 하는 그룹으로 묶어서 관리
  - 대표적인 문서형 데이터베이스에는 **MongoDB**가 있음.

NoSQL의 종류

- **Wide-Column Store 데이터베이스**
  - 데이터베이스의 열(column)에 대한 데이터를 집중적으로 관리하는 데이터베이스
  - 각 열에는 key-value 형식으로 데이터가 저장되고, 컬럼 패밀리(column families)라고 하는 열의 집합체 단위로 데이터를 처리
  - 하나의 행에 많은 열을 포함할 수 있어, 유연성이 높으며, 이에, 규모가 큰 데이터 분석에 주로 사용됨.
  - 대표적인 wide-column 데이터베이스: Cassandra, Hbase

- **그래프(Graph) 데이터베이스**
  - 자료구조의 그래프와 비슷한 형식으로 데이터 간의 관계를 구성하는 데이터베이스
  - 노드(nodes)에 속성별(entities)로 데이터를 저장하며, 각 노드간 관계는 선(edge)으로 표현
  - 대표적인 그래프 데이터베이스: Neo4J, InfiniteGraph

SQL 및 NoSQL


SQL Database
Non-SQL Database
Relational
Key-Value
Key
Value
Key
Value
Key
Value
Key
Value
Column-Family
Analiticals (OLAP)
Graph
Document

SQL 및 NoSQL

	관계형 데이터베이스 (SQL)	NoSQL
데이터 저장	• SQL을 이용해 테이블에 저장하므로 미리 작성된 스키마를 기반으로 정해진 형식에 맞게 데이터를 저장	• Key-Value, Document, wide-column, graph 등 의 방식으로 데이터를 저장
스키마	• SQL을 사용하려면 고정된 형식의 스키마가 필요하기 때문에 처리하려는 데이터 속성별로 열(column)에 대한 정보를 미리 정해 두어야 함.	• 행을 추가할 때 즉시 새로운 열을 추가할 수 있고 개별 속성에 대해 모든 열에 대한 데이터를 반드시 입력하지 않아도 된다는 장점을 지님. • 동적으로 스키마의 형태를 관리할 수 있음
쿼리	• 테이블의 형식과 테이블간의 관계에 맞춰 데이터를 요청해야 하므로 **SQL과 같은 구조화된 쿼리 언어**를 사용	• 데이터 그룹 자체를 조회하는 것에 초점을 두고 있어 **구조화 되지 않은 쿼리 언어로도 데이터 요청**이 가능
확장성	• 높은 메모리, CPU를 사용하는 수직적 확장을 하기에 하드웨어의 성능을 많이 이용하므로 높은 비용 발생 • 여러 서버에 거쳐서 데이터베이스의 관계를 정의할 수 있지만 매우 복잡하고 시간이 많이 소요 될 수 있음	• NoSQL로 구성된 데이터베이스는 수평적으로 확장하기에, NoSQL DB를 위한 서버를 추가적으로 구축하면 많은 트래픽을 편리하게 처리할 수 있다는 장점을 지님 • 저렴한 범용 하드웨어나 클라우드 기반의 인스턴스에 NoSQL DB를 호스팅할 수 있어서 상대적으로 비용이 저렴

SQL 및 NoSQL

RDB vs NoSQL

	RDB (SQL)	NoSQL
데이터 저장 모델	table	json document / key-value / 그래프 등
개발 목적	데이터 중복 감소	애자일 / 확장가능성 / 수정가능성
예시	Oracle, MySQL, PostgreSQL 등	MongoDB, DynamoDB 등
Schema	엄격한 데이터 구조	유연한 데이터 구조
⭐장점⭐	- 명확한 데이터구조 보장 - 데이터 중복 없이 한 번만 저장 (무결성) - 데이터 중복이 없어서 **데이터 update 용이**	- 유연하고 자유로운 데이터 구조 - 새로운 필드 추가 자유로움 - 수평적 확장(scale out) 용이
⭐단점⭐	- 시스템이 커지면 Join문이 많은 복잡한 query가 필요 - 성능 향상을 위해 수직적 확장(Scale up)만 가능하여 비용이 큼 - 데이터 구조가 유연하지 못함	- 데이터 중복 발생 가능 **- 중복 데이터가 많기 때문에 데이터 변경 시 모든 컬렉션에서 수정이 필요함** - 명확한 데이터구조 보장 X
⭐사용⭐	**- 데이터 구조가 변경될 여지가 없이 명확**한 경우 **- 데이터 update가 잦은** 시스템 (중복 데이터가 없으므로 변경에 유리)	- 정확한 데이터 구조가 정해지지 않은 경우 - Update가 자주 이루어지지 않는 경우 (조회가 많은 경우) - 데이터 양이 매우 많은 경우 (scale out 가능)

관계형 데이터베이스를 사용해야 하는 경우?

- **데이터베이스의 ACID 성질을 준수해야 하는 경우**
  - ACID는 Atomicity(원자성), Consistency(일관성), Isolation(격리성), Durability(지속성)
  - SQL을 사용하면 데이터베이스와 상호 작용하는 방식을 정확하게 규정할 수 있기 때문에, 데이터베이스에서 데이터를 처리할 때 발생할 수 있는 예외적인 상황을 줄이고, 데이터베이스의 무결성을 보호할 수 있음
  - 전자 상거래를 비롯한 모든 금융 서비스를 위한 소프트웨어 개발 에서는 반드시 데이터베이스의 ACID 권장

- **소프트웨어에 사용되는 데이터가 구조적이고 일관적인 경우**
  - 소프트웨어(프로젝트)의 규모가 많은 서버를 필요로 하지 않고 일관된 데이터를 사용하는 경우, 관계형 데이터베이스 권장
  - NoSQL은 **다양한 데이터 유형**과 **높은 트래픽**을 지원하도록 설계 된 것!!!

NoSQL을 사용해야 하는 경우?

- **데이터의 구조가 거의 또는 전혀 없는 대용량의 데이터를 저장하는 경우**
  - 대부분의 NoSQL 데이터베이스는 저장할 수 있는 데이터의 유형에 제한이 없다는 장점!!
  - 필요에 따라, 언제든지 데이터의 새 유형을 추가할 수 있음.
  - 소프트웨어 개발에 정형화 되지 않은 많은 양의 데이터가 필요한 경우, NoSQL을 적용하는 것이 더 효율적!

- **빠르게 서비스를 구축하는 과정에서 데이터 구조를 자주 업데이트 하는 경우**
  - NoSQL 데이터베이스의 경우 스키마를 미리 준비할 필요가 없기 때문에 빠르게 개발하는 과정에 매우 유리
  - 시장에 빠르게 프로토타입을 출시해야 하는 경우 등 시간이 중요한 경우!
  - 또한 소프트웨어 버전별로 많은 다운타임(데이터베이스 서버를 오프라인으로 전환하여 데이터 처리를 진행하는 작업 시간) 없이 데이터 구조를 자주 업데이트 해야하는 경우나 스키마를 매번 수정해야 하는 관계형 데이터베이스 보다 NoSQL 기반의 비관계형 데이터베이스를 사용하는 게 더 적합

관계형 데이터베이스 (SQL)

- **데이터베이스 및 테이블 생성 (create)**
  - CREATE DATABASE Library;: Library라는 이름의 새로운 데이터베이스를 생성합니다.
  - USE Library;: Library 데이터베이스를 사용하겠다는 선언입니다. 이후 명령들은 이 데이터베이스에서 실행됩니다.
  - CREATE TABLE Books;: Books라는 테이블을 생성합니다.
  - BookID INT PRIMARY KEY AUTO_INCREMENT;: BookID는 각 책을 고유하게 식별하는 기본 키입니다. AUTO_INCREMENT는 새로운 레코드가 추가될 때마다 BookID가 자동으로 증가하도록 합니다.
  - Title VARCHAR(100);, Author VARCHAR(100);, PublishedYear INT;, Genre VARCHAR(50);: 각 열의 이름과 데이터 유형을 정의합니다. VARCHAR(100)는 최대 100자의 문자열을 저장할 수 있다는 의미입니다.

```sql
CREATE DATABASE Library;
USE Library;

CREATE TABLE Books (
    BookID INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(100),
    Author VARCHAR(100),
    PublishedYear INT,
    Genre VARCHAR(50)
);
```

관계형 데이터베이스 (SQL)

- **데이터 삽입 (INSERT)**
  - INSERT INTO Books (Title, Author, PublishedYear, Genre): Books 테이블의 Title, Author, PublishedYear, Genre 열에 데이터를 삽입합니다.
  - VALUES (...);: 삽입할 데이터의 값을 지정합니다. 여러 개의 레코드를 한 번에 삽입할 수 있습니다.
  - 예시로 세 개의 책 데이터를 Books 테이블에 추가합니다.

```sql
INSERT INTO Books (Title, Author, PublishedYear, Genre)
VALUES ('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Fiction'),
    ('To Kill a Mockingbird', 'Harper Lee', 1960, 'Fiction'),
    ('1984', 'George Orwell', 1949, 'Dystopian');
```

관계형 데이터베이스 (SQL)

- **데이터 조회 (SELECT)**
  - 모든 데이터 조회
    - SELECT * FROM Books;: Books 테이블의 모든 데이터를 조회합니다. *는 모든 열을 선택하라는 의미입니다.
  - 특정 데이터 조회
    - WHERE PublishedYear < 1950;: PublishedYear가 1950년보다 이전인 책만 조회합니다.

```
1 SELECT * FROM Books;
2 SELECT * FROM Books WHERE PublishedYear < 1950;
```

관계형 데이터베이스 (SQL)

- **데이터 업데이트 (UPDATE)**
  - UPDATE Books: Books 테이블의 데이터를 수정합니다.
  - SET Genre = 'Classic Fiction';: Genre 열의 값을 'Classic Fiction'으로 변경
  - WHERE Title = 'The Great Gatsby';: Title이 'The Great Gatsby'인 행만 업데이트합니다. 조건을 지정하지 않으면 테이블의 모든 행이 업데이트될 수 있으므로 주의가 필요

```sql
UPDATE Books
SET Genre = 'Classic Fiction'
WHERE Title = 'The Great Gatsby';
```

- **데이터 삭제 (DELETE)**
  - DELETE FROM Books: Books 테이블에서 데이터를 삭제합니다.
  - WHERE Title = '1984';: Title이 '1984'인 행을 삭제합니다. 조건을 지정하지 않으면 테이블의 모든 행이 삭제될 수 있으므로 주의해야 합니다.

```sql
DELETE FROM Books WHERE Title = '1984';
```

NoSQL (MongoDB)

- **데이터베이스 및 컬렉션 생성 (CREATE**

```
use library; // 'library' 데이터베이스 사용 (없으면 생성됨)
```

  - use library;: library라는 이름의 데이터베이스를 선택합니다. 데이터베이스가 존재하지 않으면 자동으로 생성됩니다.
  - MongoDB에서는 컬렉션을 명시적으로 생성하지 않고, 데이터를 삽입할 때 컬렉션이 자동으로 생성됩니다.

- **데이터 삽입 (INSERT)**

```
db.books.insertMany([
    { "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "publishedYear": 1925, "genre": "Fiction" },
    { "title": "To Kill a Mockingbird", "author": "Harper Lee", "publishedYear": 1960, "genre": "Fiction" },
    { "title": "1984", "author": "George Orwell", "publishedYear": 1949, "genre": "Dystopian" }
]);
```

  - db.books.insertMany([...]);: books 컬렉션에 여러 문서를 한 번에 삽입합니다.
  - 문서(Document)는 MongoDB에서 데이터를 저장하는 기본 단위로, JSON 형식으로 표현됩니다.
  - 각 문서는 title, author, publishedYear, genre 필드를 가지고 있습니다.

NoSQL (MongoDB)

- **데이터 조회 (FIND)**

```
db.books.find();
db.books.find({ "publishedYear": { $lt: 1950 } });
```

  - db.books.find();: books 컬렉션의 모든 문서를 조회합니다.
  - db.books.find({ "publishedYear": { $lt: 1950 } });: publishedYear가 1950년보다 작은 문서만 조회합니다. MongoDB의 쿼리 언어는 JavaScript 객체 형식을 사용하며, $lt는 "less than"을 의미하는 연산자입니다.

- **데이터 업데이트 (UPDATE)**

```
db.books.updateOne(
    { "title": "The Great Gatsby" },
    { $set: { "genre": "Classic Fiction" } }
);
```

  - db.books.updateOne(...): books 컬렉션에서 조건에 맞는 첫 번째 문서를 업데이트합니다.
  - { "title": "The Great Gatsby" }: title이 'The Great Gatsby'인 문서를 찾습니다.
  - { $set: { "genre": "Classic Fiction" } }: genre 필드를 'Classic Fiction'으

NoSQL (MongoDB)

- **데이터 삭제 (DELETE)**
  - db.books.deleteOne(...): books 컬렉션에서 조건에 맞는 첫 번째 문서를 삭제합니다.
  - { "title": "1984" }: title이 '1984'인 문서를 찾고 삭제합니다.

```
db.books.deleteOne({ "title": "1984" });
```

SQL & NoSQL 사용 예제

- **SQL과 NoSQL의 주요 차이점**

- **SQL(MySQL)**
  - **정형화된 데이터**: 데이터가 정해진 스키마에 따라 테이블에 저장
  - **관계형 데이터 모델**: 테이블 간의 관계를 외래 키 등을 통해 정의
  - **고정된 스키마**: 테이블의 구조(열과 데이터 유형)가 사전에 정의되고, 변경이 까다로울 수 있음.
  - **복잡한 쿼리**: 트랜잭션 처리와 복잡한 JOIN 연산을 통해 데이터를 복잡하게 조작할 수 있음

- **NoSQL(MongoDB)**
  - **비정형 데이터**: JSON과 유사한 문서 형식으로 데이터를 저장하며, 각 문서는 서로 다른 구조를 가질 수 있음
  - **유연한 스키마**: 스키마가 유연하고, 문서에 새로운 필드를 쉽게 추가 가능
  - **확장성**: 수평적 확장이 용이하여 대규모 데이터 처리에 적합
  - **빠른 개발**: 스키마 설계에 대한 제약이 적어 빠르게 개발할 수 있으며, 대용량 데이터를 처리하는 데 유리

감사합니다.

- 본 온라인 콘텐츠는 2024년도 과학기술 정보통신부 및 정보통신기획평가원의 'SW중심대학사업' 지원을 받아 제작되었습니다.
- 본 결과물의 내용을 전재할 수 없으며, 인용(재사용)할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원이 지원한 'SW중심대학'의 결과물이라는 출처를 밝혀야 합니다.

전북대학교 JEONBUK NATIONAL UNIVERSITY
전북대학교 SOFTWARE중심대학사업단
SW중심대학
과학기술정보통신부
IITP 정보통신기획평가원