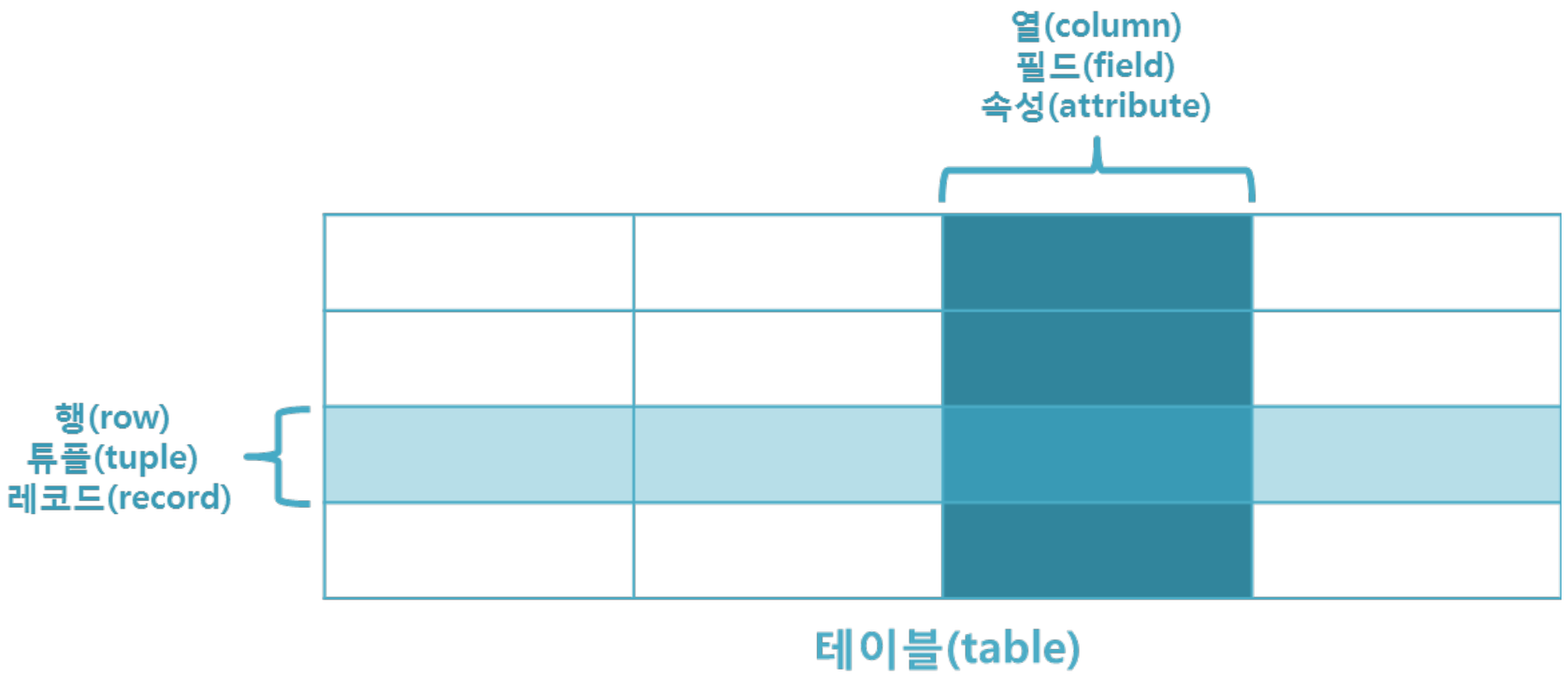
SQL

관계형 데이터 베이스란

- 테이블(table)로 이루어져 있으며, 이 테이블은 키(key)와 값(value)의 관계를 나타냄
- 데이터의 종속성을 관계(relationship)로 표현하는 것이 관계형 데이터베이스의 특징



관계형 데이터베이스는 테이블이 다른 테이블들과 관계를 맺고 모여있는 집합체

관계형 데이터베이스의 특징

- 데이터의 분류, 정렬, 탐색 속도가 빠름
- 신뢰성이 높고, 데이터의 무결성을 보장
- 기존에 작성된 스키마를 수정하기 어려움
- 데이터베이스의 부하를 분석하는 것이 어려움

NoSQL

Not Only SQL의 약자로 기존 RDBMs 형태의 관계형 데이터베이스가 아닌 다른 형태의 데이터 저장 기술을 의미하며, 수평적 확장성을 갖음

- 데이터 간의 관계를 정의하지 않음

RDBMS는 JOIN 연산을 수행할 수 있지만, NoSQL은 JOIN 연산이 불가능

- 대용량의 데이터를 저장할 수 있음

- 분산형 구조

여러 곳의 서버에 데이터를 분산 저장해 특정 서버에 장애가 발생했을 때도 데이터 유실 혹은 서비스 중지가 발생하지 않도록 함

- 고정되지 않은 테이블 스키마

구조에 대한 정의를 변경할 필요 없이 데이터베이스 레코드에 자유롭게 필드를 추가

*스키마는 **데이터베이스의 구조와 제약조건에 관한 전반적인 명세를 기술한 것**

데이터베이스를 구성하는 데이터 개체(Entity), 개체의 특성을 나타내는 속성(Attribute), 개체 사이에 존재하는 관계(Relationship) 및 데이터 조작 시 데이터 값들이 갖는 제약 조건 등에 관하여 기술

NoSQL 사용하는 이유

NoSQL 데이터베이스는 유연성과 확장성을 비롯해 고성능의 매우 기능적인 데이터베이스를 필요로 하는 모바일, 웹이나 게이밍과 같은 다양한 현대적인 애플리케이션에 적합

- 유연성:
  유연한 스키마를 제공 -> 보다 빠르고 반복적인 개발을 가능

- 확장성:
  고가의 강력한 서버를 추가하는 대신 분산형 하드웨어 클러스터를 이용해 확장하도록 설계

- 고성능:
  특정 데이터 모델(문서, 키 값, 그래프 등) 및 엑세스 패턴에 대해 최적화되어 관계형 데이터베이스를 통해 유사한 기능을 충족하려 할 때보다 뛰어난 성능을 가짐

- 고기능성:
  각 데이터 모델에 맞추어 특별히 구축된 뛰어난 기능의 API와 데이터 유형을 제공

NO SQL / RDBMS 비교

NO sql - 수평적 확장성 :
- 서버를 늘리기만 하면은 스케일이 늘어남. 그러면서 응답속도는 보장

RDBMS - 수직적 확장성 :
- ssd, cpu 업그레이드 하거나, 확장하는데 돈과 손이 많이 듦

NO sql - 쿼리 언어 없음 :
- 페이스북이나 인스타 같이 유저가 폭발적으로 많고 간단한 데이터를 다루면 채택

RDBMS - 쿼리 언어 :
- 표준적인 쿼리 언어가 있음. 만일 join을 많이 해야한다면 sql이 나음

NO sql - 자유로운 스키마 :
- 추가하고 싶은 어트리뷰트는 제약없이 넣음. 생각없이 만들면 지저분해질 수 있음. 최대한 key기준으로 쿼리를 작성하는게 요점.

RDBMS - 강한 스키마 :
- 테이블을 정의하고 나서, 여러 제약조건이 잇음. 타입이라던가 외래키 제약이라던가. 인덱스를 많이 걸게 됨. 만일 인덱스를 많이 체크해야한다면 rdbms가 좋음.

CREATE TABLE, DROP TABLE 문법

테이블생성

```
—문법
CREATE TABLE 테이블명(
컬럼명 타입(크기) NOT NULL, ——널값이 들어갈 수 없음
컬럼명 타입 NULL DEFAULT(값), ——초기값 지정.
CONSTRAIN PK이름 PRIMARY KEY(컬럼명) —PK설정
)
```

```
—예제
CREATE TABLE MY_TABLE(
NO_EMP NVARCHAR(10)NOT NULL, -- NULL 값이 들어갈 수 없음
NM_KOR NVARCHAR(40)NOT NULL, -- NULL 값이 들어갈 수 없음.
AGE INT NULL DEFAULT (0), ——디폴트 = 0
TODAY DATETIME DEFAULT(GETDATE()), ——디폴트 현재일자.
CONSTRAINT PK_MY_TABLE PRIMARY KEY(NO_EMP)——PK : NO_EMP)
```

테이블 삭제

```
DROP TABLE 테이블명.
DROP TABLE MY_TABLE
```

SELECT WHERE 문법

테이블에 있는 데이터 조회

```
--My_Table로 부터 모든 칼럼 조회
SELECT * FROM My_Table
--My_Table의 No_Emp,Nm_Kor,Age 칼럼 조회
SELECT No_Emp,Nm_Kor,Age FROM My_Table
)
```

Where 조건문

```
--이름이 '홍길동'인사람 검색
SELECT * FROM My_Table WHERE Nm_Kor ='홍길동'

--나이가 25살인 사원의 한국이름과 나이 조회
SELECT Nm_Kor,Age FROM My_Table WHERE Age=25

--나이가 25살이 아닌 사원 조회
SELECT * FROM My_Table WHERE Age<>25

--사원번호가 '0315' 이고 나이가 25살보다 작거나 이름이 '홍길동'인 사원 조회
SELECT * FROM My_Table WHERE No_Emp = '0315' AND (Age<25  OR Nm_Kor = '홍길동')

--사원번호가 '0315' 이거나 나이가 25살 이상이면서 이름이 '홍길동'인 사원 조회
SELECT * FROM My_Table WHERE No_Emp = '0315' OR (Age>=25 AND Nm_Kor = '홍길동')
```

Like(~로 시작,포함,끝나는 단어)

```
--'김'으로 시작하는 사원 조회
SELECT * FROM My_Talbe WHERE Nm_Kor LIKE '김%'

--김이 들어가는 시작하는 사원 조회
SELECT * FROM My_Talbe WHERE Nm_Kor LIKE '%김%'

--김으로 끝나는 사원의 사원번호 조회
SELECT No_Emp FROM My_Talbe WHERE Nm_Kor LIKE '%김'
```

In(~이거나)

```
--나이가 20살,24살,26살인 사원 조회
SELECT * FROM My_Table WHERE Age IN(20,24,26)

사원번호가 '0000','0004','0008'이고 나이가 20살 24살 28살인 사원 조회
SELECT * FROM My_Table Where No_Emp IN(‘0000','0004','0008)AND Age IN(20,24,28)
```

Between(~부터~까지)

```
나이가 20살~25살까지의 사원조회
SELECT * FROM My_Table WHERE Age Between 20 AND 24

나이가 사원번호가 '0000'~'0010'까지이거나 나이가 30살~40살인 사원의 이름 조회
SELECT Age FROM My_Table WHERE (No_Emp BETWEEN '0000' AND '0010') OR (Age BETWEEN 30
AND 40)
```

UPDATE 문법

Update문은 굉장히 위험한 명령어이므로 꼭 트랜잭션 안에서만 사용해야 함
특히 Update문구에 실수로 Where 절을 쓰지 않고 실행을 하게 되면 테이블에 있는 칼럼 모두가 바뀜
=> Update에 하기 앞서서 Select문으로 자신이 바꿔야 할 데이터를 조회한 뒤 꼭 트랜잭션 안에서 Update문을 실행

```
UPDATE 테이블명 SET 칼럼명 = '내용'
UPDATE 테이블명 SET 칼럼명 = '내용' WHERE 조건문
)
```

Update 예제

```
--문법
SELECT * FROM 테이블명 WHERE 칼럼명 = '내용' --칼럼의 존재 확인

BEGIN TRAN --트랜잭션 시작
UPDATE 테이블명 SET Update할칼럼 = '바꿀문구' WHERE 칼럼명 ='기존문구' --홍길동을 이순신으로 바꿈
SELECT * FROM 테이블명 WHERE 칼럼명 = '바뀐문구'--나만 바뀐것을 확인할수 있고 아직 적용은 안되어있는 상태이다.
ROLLBACK TRAN -되돌리기 --Update 취소
COMMIT TRAN --트랜잭션 완료 --Update 적용
-------------------------------------------------------------------------------
--Example
SELECT * FROM My_Table WHERE Nm_Kor = '홍길동' --홍길동 칼럼의 존재 확인

BEGIN TRAN --트랜잭션 시작
UPDATE My_Table SET Nm_Kor = '이순신' WHERE Nm_Kor ='홍길동' --홍길동을 이순신으로 바꿈
SELECT * FROM My_Table WHERE NM_KOR = '이순신'--나만 바뀐것을 확인할수 있고 아직 적용은 안되어있는 상태이다.
ROLLBACK TRAN -되돌리기
COMMIT TRAN --트랜잭션 완료
```

INSERT 문법

```
INSERT INTO 테이블명(칼럼1,칼럼2,칼럼3,....) values(데이터1,데이터2,데이터3,.....)
INSERT INTO My_Table(no_emp,nm_kor,age) values('dz000','홍길동',20)
```

다른 테이블의 데이터를 Insert 하는 법

```
---일부 칼럼만 복사
INSERT INTO  복사 될 테이블명(칼럼1,칼럼2,컬럼3,....)
SELECT 칼럼1,칼럼2,칼럼3,.... from 복사 할 테이블명
```

```
--전체 복사
INSERT INTO  복사 될 테이블명
SELECT * from 복사 할 테이블명
```

예제(Example)

```
--일부 칼럼만 복사
INSERT INTO MyTable_01(NO_EMP,NM_KOR,AGE)
SELECT NO_EMP,NM_KOR,AGE from MyTable
```

```
--전체 복사
INSERT INTO MyTable_01
SELECT * from MyTable
```

DELETE 문법

```
BEGIN TRAN --트랜잭션 시작.
DELETE FROM 테이블명 WHERE 조건문.
SELECT * FROM 테이블명 WHERE 조건문 --나만 삭제된것을 확인할수 있고 아직 적용은 안되어있는 상태이다.
ROLLBACK TRAN -되돌리기 --Delete 취소
COMMIT TRAN --트랜잭션 완료 --Delete 적용
------------------------------------------------------------------------------
SELECT * FROM My_Table WHERE Nm_Kor = '홍길동' --홍길동 컬럼의 존재 확인

BEGIN TRAN --트랜잭션 시작
DELETE FROM My_Table WHERE Nm_Kor ='홍길동' --Km_Kor이 홍길동인 칼럼 삭제
SELECT * FROM My_Table WHERE NM_KOR = '홍길동'--나만 삭제된것을 확인할수 있고 아직 적용은.
                                              안되어있는 상태이다.
ROLLBACK TRAN -되돌리기.
COMMIT TRAN --트랜잭션 완료
```

JOIN문법

두개이상의 테이블이나 데이터베이스를 연결하여 데이터를 검색하는 방법

자신이 검색하고 싶은 컬럼이 다른 테이블에 있을 경우 주로 사용하며 여러 개의 테이블을 마치 하나의 테이블인 것처럼 활용하는 방법

- INNER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN
- CROSS JOIN
- SELF JOIN

INNER JOIN

쉽게말해 교집합, 기준 테이블과 Join 한 테이블의 중복된 값을 보여줌
결과값: A의 테이블과 B테이블이 모두 가지고 있는 데이터만 검색

```
--문법--
SELECT
테이블별칭.조회할칼럼,
테이블별칭.조회할칼럼.
FROM 기준테이블 별칭.
INNER JOIN 조인테이블 별칭 ON 기준테이블별칭.기준키 = 조인테이블별칭.기준키...
```

```
--예제--
SELECT
A.NAME, --A테이블의 NAME조회
B.AGE --B테이블의 AGE조회
FROM EX_TABLE A
INNER JOIN JOIN_TABLE B ON A.NO_EMP = B.NO_EMP AND A.DEPT = B.DEPT
```

LEFT OUTER JOIN

기준테이블의 값 + 테이블과 기준 테이블의 중복된 값
왼쪽 테이블을 기준으로 JOIN을 하겠다->결과값: A테이블의 모든 데이터와 A테이블과 B테이블의 중복되는 값

```
--문법--
SELECT
테이블별칭.조회할칼럼,
테이블별칭.조회할칼럼.
FROM 기준테이블 별칭.
LEFT OUTER JOIN 조인테이블 별칭 ON 기준테이블별칭.기준키 = 조인테이블별칭.기준키...
```

```
--예제--
SELECT
A.NAME, --A테이블의 NAME조회
B.AGE --B테이블의 AGE조회
FROM EX_TABLE A
LEFT OUTER JOIN JOIN_TABLE B ON A.NO_EMP = B.NO_EMP AND A.DEPT = B.DEPT
```

RIGTH OUTER JOIN

LEFT OUTER JOIN의 반대, 오른쪽 테이블을 기준으로 JOIN
결과값: B테이블의 모든 데이터와 A테이블과 B테이블의 중복되는 값이 검색

FULL OUTER JOIN

쉽게 말해 합집합,
A테이블이 가지고 있는 데이터 , B테이블이 가지고 있는 데이터 모두 검색

```
--문법--
SELECT
테이블별칭.조회할칼럼,
테이블별칭.조회할칼럼.
FROM 기준테이블 별칭.
FULL OUTER JOIN 조인테이블 별칭 ON 기준테이블별칭.기준키 = 조인테이블별칭.기준키…
```

```
--예제--
SELECT
A.NAME, --A테이블의 NAME조회.
B.AGE --B테이블의 AGE조회.
FROM EX_TABLE A
FULL OUTER JOIN JOIN_TABLE B ON A.NO_EMP = B.NO_EMP AND A.DEPT = B.DEPT
```

CROSS JOIN

모든 경우의 수를 전부 표현해주는 방식,
기준 테이블이 A일 경우 A의 데이터 한 ROW를 B테이블 전체와 JOIN 하는 방식 -> 결과값 : N * M

```
--문법(첫번째방식)--
SELECT
테이블별칭.조회할칼럼,
테이블별칭.조회할칼럼.
FROM 기준테이블 별칭.
CROSS JOIN 조인테이블 별칭.
```

```
--예제(첫번째방식)--
SELECT
A.NAME, --A테이블의 NAME조회
B.AGE --B테이블의 AGE조회
FROM EX_TABLE A
CROSS JOIN JOIN_TABLE B
```

```
==========================================================
--예제(두번째방식)--
SELECT
A.NAME, --A테이블의 NAME조회
B.AGE --B테이블의 AGE조회
FROM EX_TABLE A,JOIN_TABLE B
```

SELF JOIN

자기자신과 자기 자신을 조인한다는 의미
하나의 테이블을 여러번 복사해서 조인,자신이 가지고 있는 칼럼을 다양하게 변형시켜 활용할 경우에 자주 사용

```
--문법--
SELECT.
테이블별칭.조회할칼럼,
테이블별칭.조회할칼럼.
FROM 테이블 별칭,테이블 별칭2.
```

```
--예제--
SELECT
A.NAME, --A테이블의 NAME조회
B.AGE --B테이블의 AGE조회
FROM EX_TABLE A,EX_TABLE B
```