



비매품

# 실력완성!! 데이터베이스

(V1.1)  
2019.02

아이리포 실력완성(올패스)  
김미경 정보관리기술사

# 데이터베이스

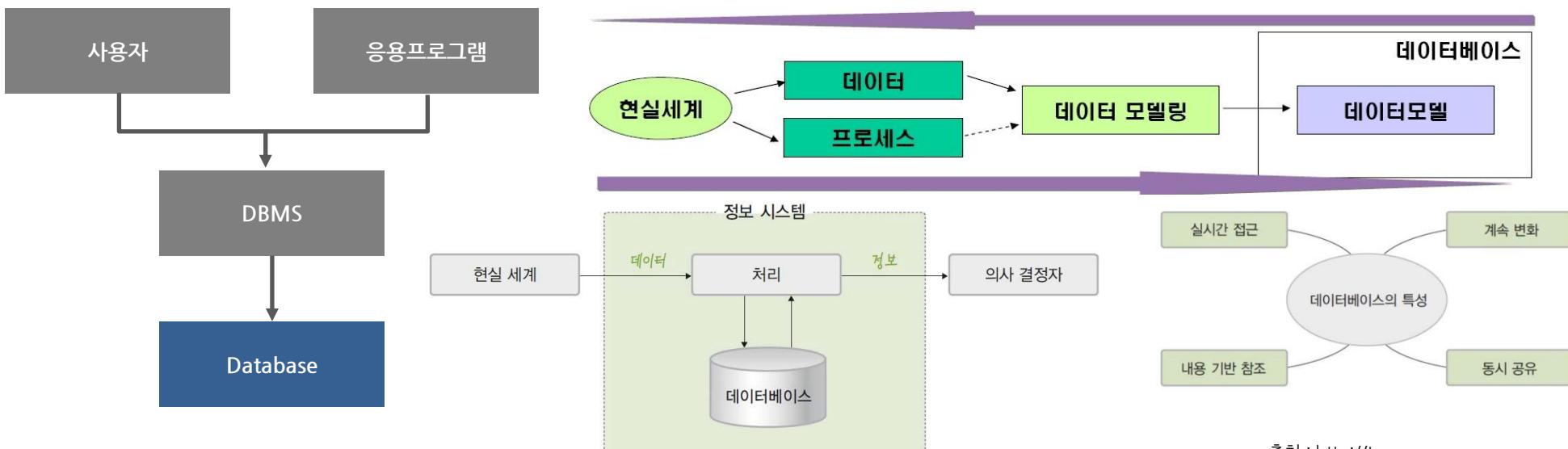
통저운공

## 1. 특정 조직의 업무를 수행하는 데 필요한 상호 관련된 데이터들의 모임, 데이터베이스의 개론

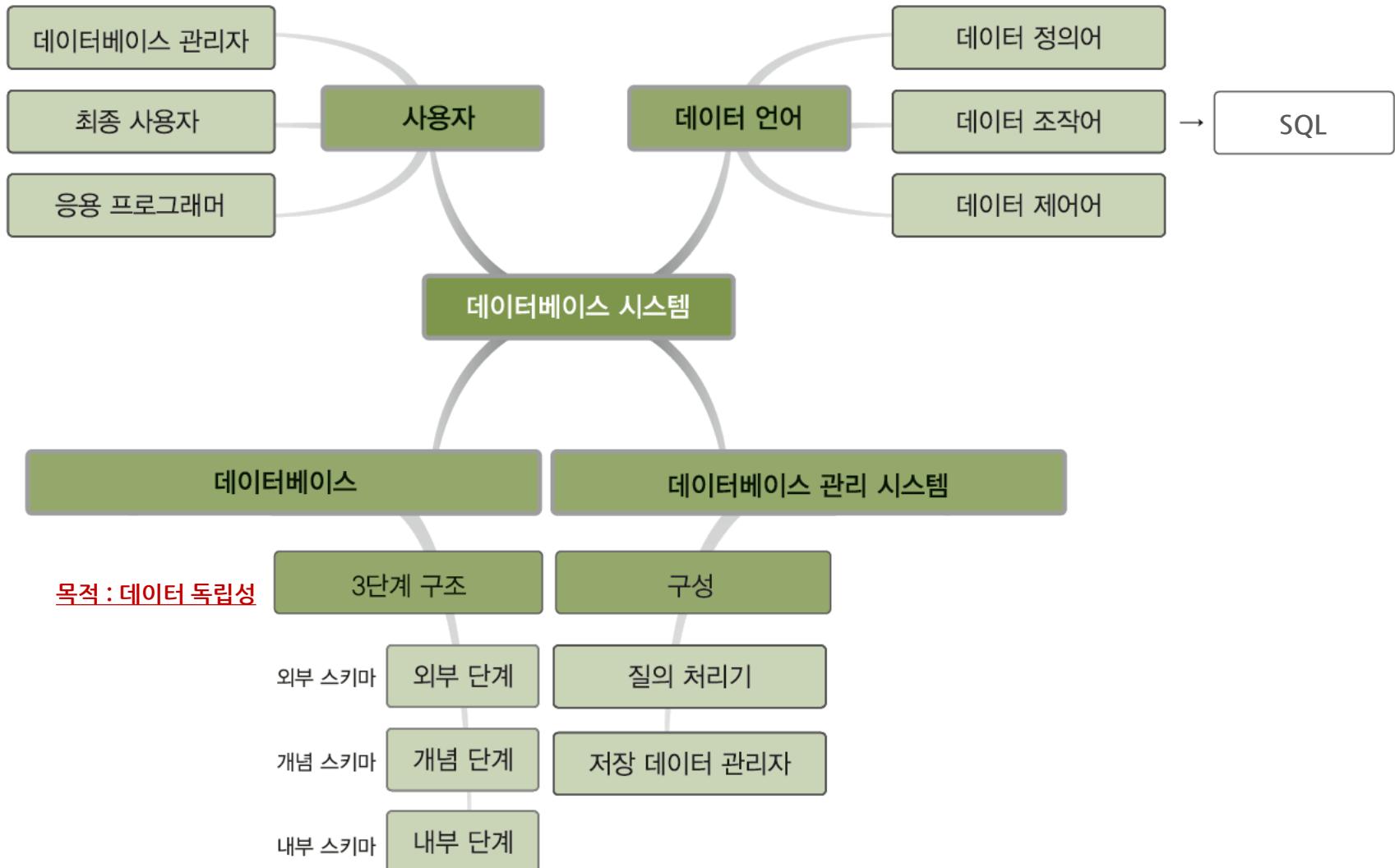
- 여러 사람에 의해 공유되어 사용될 목적으로 통합하여 관리되는 **데이터의 집합**
- 몇 개의 자료 파일을 조직적으로 통합하여 자료 항목의 중복을 없애고 자료를 구조화하여 기억시켜 놓은 자료의 집합체

## 2. 데이터베이스의 구성

구성요소	설명
<b>통</b> 합 데이터(integrated data)	<ul style="list-style-type: none"><li>중복을 배제하나, 경우에 따라 불가피하게 중복을 허용하는 데이터</li><li>의도적 중복은 항상 파악하여 관리할 수 있음</li></ul>
<b>저장</b> 데이터(stored data)	<ul style="list-style-type: none"><li>컴퓨터의 저장매체에 저장하여 관리하는 데이터</li></ul>
<b>운영</b> 데이터(operation data)	<ul style="list-style-type: none"><li>단순한 데이터의 집합이 아니라 그 조직의 기능을 수행하는 데 없어서는 안 될 필수의 데이터</li></ul>
<b>공용</b> 데이터(shared data)	<ul style="list-style-type: none"><li>조직의 여러 사용자와 여러 응용시스템들이 서로 다른 목적으로 데이터를 공동으로 이용할 수 있음</li></ul>



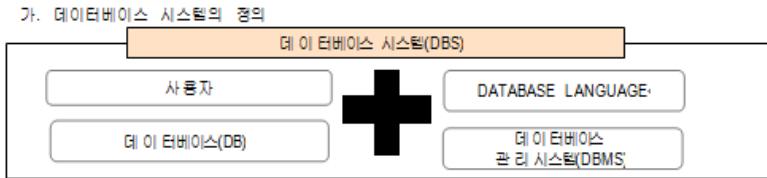
# DBMS(Database Management System)



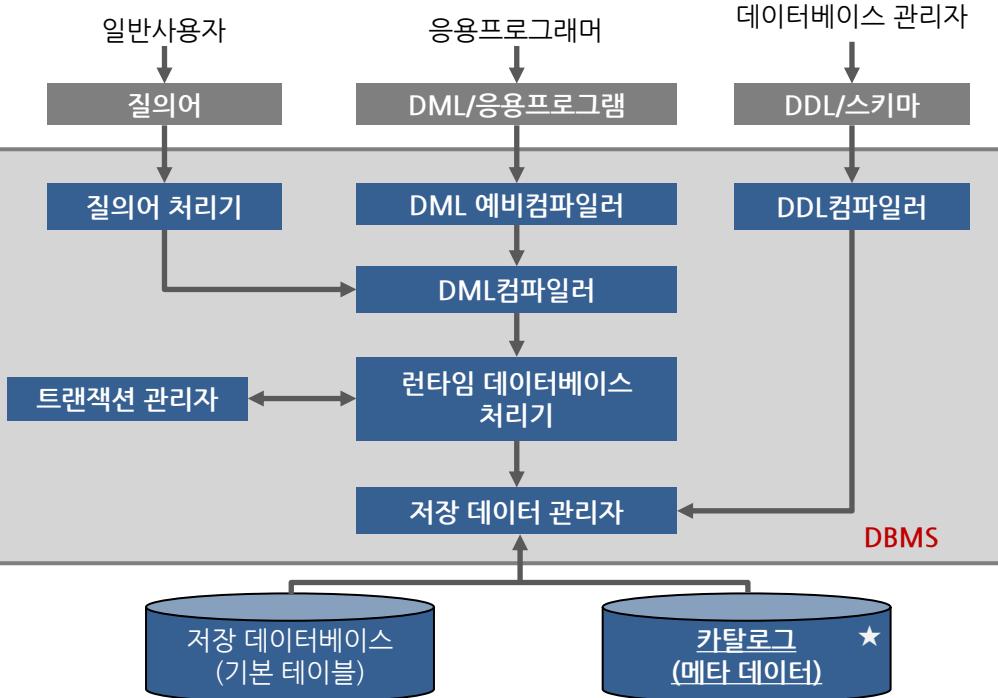
# DBMS(Database Management System)

## 1. DBMS의 개요

- 조직의 데이터베이스를 생성·유지·사용하는 과정을 제어해 주는 **응용 소프트웨어**
- 사용자와 데이터베이스 사이에 위치하여 데이터베이스를 관리하고 사용자의 요구에 따라 데이터베이스에 대한 연산을 수행해 정보를 생성해 주는 **소프트웨어**



## 2. DBMS의 구성요소



구성요소	설명
DDL 컴파일러 (DDL compiler)	<ul style="list-style-type: none"> <li>DDL로 명세된 schema를 내부 형태로 변환하여 catalog에 저장</li> <li>메타 데이터베이스(meta-database): 데이터의 데이터</li> </ul>
질의어 처리기 (query processor)	<ul style="list-style-type: none"> <li>질의문을 parsing, analysis, compile하여 DB를 접근하기 위한 object code를 생성</li> </ul>
예비 컴파일러 (precompiler)	<ul style="list-style-type: none"> <li>응용 프로그램에 삽입된 DML(DSL)을 추출하고 그 자리에 procedure call로 대체</li> <li>추출된 DML은 DML 컴파일러로 전달</li> <li>수정된 응용 프로그램은 host 프로그램 컴파일러로 전달</li> </ul>
DML 컴파일러 (DML compiler)	<ul style="list-style-type: none"> <li>DML 명령어를 object code로 변환</li> </ul>
런타임 데이터베이스 처리기 (runtime database processor)	<ul style="list-style-type: none"> <li>실행 시간에 데이터베이스를 접근</li> <li>DB 연산(operations)을 저장 데이터 관리자(stored data manager)를 통해 수행</li> </ul>
트랜잭션 관리자 (transaction manager)	<ul style="list-style-type: none"> <li>트랜잭션(transaction) 단위로 작업을 수행</li> <li>DB 접근 과정에서 무결성(integrity)과 권한(authorization) 제어</li> <li>병행 제어(concurrency control)와 회복(recovery) 작업</li> </ul>
저장 데이터 관리자 (stored data manager)	<ul style="list-style-type: none"> <li>디스크에 있는 사용자 DB나 카탈로그 접근을 제어</li> <li>기본 OS module(file manager, disk manager)을 이용</li> </ul>

# DBMS(Database Management System)

## 3. DBMS의 유형

유형	설명	종류
계층형 데이터베이스	<ul style="list-style-type: none"><li>데이터의 관계를 트리 구조로 정의하고, 부모, 자식 형태를 갖는 구조</li><li>상위에 레코드가 복수의 하위 레코드를 갖는 구조</li><li>단점 : 데이터의 중복</li></ul>	
네트워크형 데이터베이스	<ul style="list-style-type: none"><li>계층형 데이터의 데이터 중복 문제를 해결</li><li>레코드간의 다양한 관계를 그물처럼 갖는 구조</li><li>단점 : 복잡한 구조 때문에 추후에 구조를 변경한다면 많은 어려움 존재</li></ul>	
관계형 데이터 베이스	<ul style="list-style-type: none"><li>행(Column), 열(Record)로 구성된 Table 간의 관계를 나타낼 때 사용</li><li>이렇게 표현된 데이터를 SQL(Structured Query Language)을 사용하여 데이터 관리 및 접근</li></ul>	오라클 MySQL 등
NoSQL	<ul style="list-style-type: none"><li>관계형 데이터베이스보다 덜 제한적인 일관성 모델을 이용</li><li>키(key)와 값(value)형태로 저장</li><li>키를 사용해 데이터 관리 및 접근</li></ul>	카산드라 몽고유

# 시스템 카탈로그 / 데이터 사전(Data Dictionary)

## 1. 시스템 카탈로그의 개요

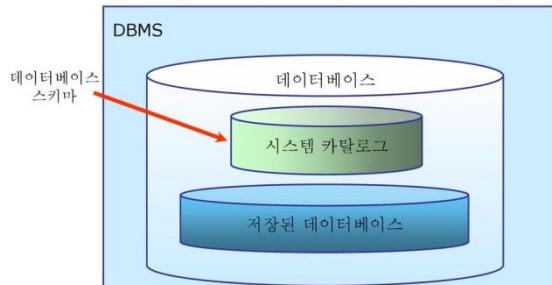
- 데이터베이스의 개체들에 대한 정의를 담고 있는 **메타데이터들로 구성된 데이터베이스 내의 인스턴스**
- 기본 테이블, 뷰 테이블, 동의어(synonym), 값 범위, 인덱스, 사용자, 사용자 그룹 등등과 같은 데이터베이스의 개체들이 저장
- DBMS는 특정 데이터베이스의 구조를 파악하기 위하여 데이터의 타입이나 포맷과 같은 정보를 데이터베이스 카탈로그에서 획득

## 2. 카탈로그의 특징

- 카탈로그 자체도 시스템 테이블로 구성되어 있어 일반 이용자도 **SQL을 이용하여 내용을 검색**해볼 수 있다.
- INSERT, DELETE, UPDATE문으로 **카탈로그를 갱신하는 것은 허용되지 않는다.**
- 데이터베이스 **시스템에 따라 상이한 구조**를 갖는다.
- 카탈로그는 **DBMS가 스스로 생성하고 유지**한다.
- 카탈로그의 갱신 : 사용자가 SQL문을 실행시켜 기본 테이블, 뷰, 인덱스 등에 변화를 주면 **시스템이 자동으로 갱신**한다.
- 분산 시스템에서의 카탈로그 : 보통의 릴레이션, 인덱스, 사용자 등에 정보를 포함할 뿐 아니라 위치 투명성 및 중복 투명성을 제공하기 위해 필요한 모든 제어 정보를 가져야 한다.

## 3. 시스템 카탈로그의 종류

- SYSTABLES** : 기본 테이블 및 뷰 테이블의 정보를 저장하는 테이블
- SYSOLUMNS** : 모든 테이블에 대한 정보를 열(속성) 중심으로 저장하는 테이블
- SYSVIEW** : 뷰에 대한 정보를 저장하는 테이블
- SYSTABAUTH** : 테이블에 설정된 권한 사항들을 저장하는 테이블
- SYSCOLAUTH** : 각 속성에 설정된 권한 사항들을 저장하는 테이블



[그림1. 3] 시스템 카탈로그와 저장된 데이터베이스

구성요소	내용	특징
스키마 구조	- 스키마의 테이블명, 인덱스명, 컬럼명, 뷰, 참조관계에 대해 내용	- 기본적인 데이터사전의 기능
감사/추적	- 데이터베이스에서 작업을 수행한 이력, 트랜잭션정보, 세션정보 등	- 전단 및 최적화 활용
사용자 권한	- 데이터베이스 오브젝트에 대해 접근하기 위한 접근, 입력, 수정, 삭제 등에 대한 권한 정보	- 데이터베이스 역할 기반접근방법인 RBAC 이 저장
질의 최적화기	- Optimizer 가 최적화된 경로를 찾기 위해 통계정보를 생성하여 저장	- SQL 문장이 실행될 때 Execution Plan 설정시 참조
컴파일러	- 고수준의 질의와 데이터 조작어 명령들을 저수준의 파일 접근 명령	- 스키마 접근을 위한 준비단계 정보

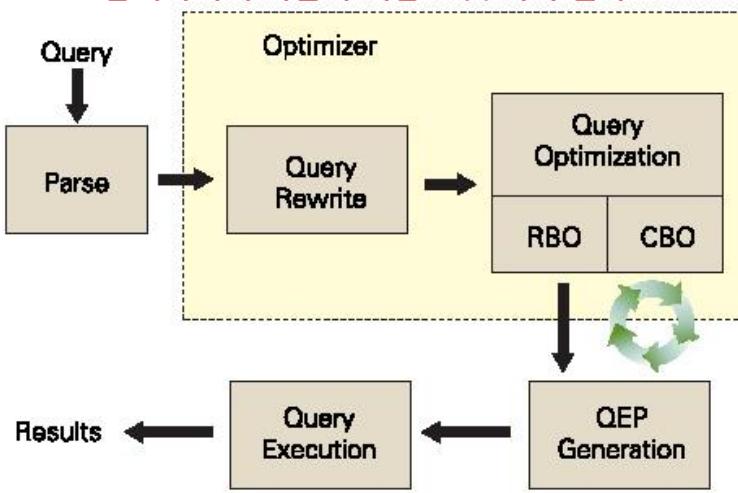
# 옵티マイ저

RBO, CBO

## 1. Query의 최적의 실행 방법을 결정해주는 DBMS의 핵심엔진. DB옵티マイ저의 개요

- 사용자가 요청한 SQL을 효율적으로 처리하기 위해 **최적의 실행 경로를 찾아주는 DBMS의 핵심 엔진**
- 구조화된 질의언어(SQL)로 사용자가 원하는 결과집합을 정의하면 이를 얻는데 필요한 처리절차(프로세서)는 DBMS에 내장된 옵티マイ저가 자동으로 생성해줌
- 핵심 기능 :** 실행 계획 탐색(Search Space Enumeration), 비용 산정(Cost Estimation, 최소 실행 비용 드는 경로를 제시)

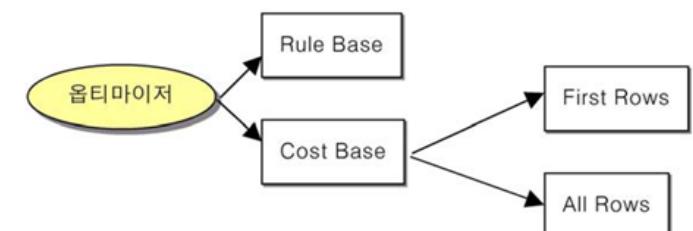
## 2. DB 옵티マイ저 역할의 개념도 및 처리 절차



<그림 1> 질의 처리 단계와 옵티マイ저의 역할

## 3. 옵티マイ저의 분류(RBO, CBO)

종류	설명
<b>RBO (Rule Based Optimizer)</b>	인덱스구조나 비교연산자에 따른 <b>순위부여를 기준으로 최적의 경로설정</b> 을 하는 옵티マイ저
<b>CBO (Cost Based Optimizer)</b>	처리방법들에 대한 비용을 산정해보고 그 중에서 <b>가장 적은 비용이 들어가는 처리방법들을 선택</b> 하는 옵티マイ저 (통계적)



# 옵티マイ저 - RBO, CBO

## 1. 규칙 기반 옵티マイ저, RBO(Rule Based Optimizer)의 개요

- 옵티マイ저로서 미리 정해진 **우선 순위 규칙에 따라 접근 경로(Access Path)**를 결정하는 옵티マイ저 유형
- 잘못된 우선 순위의 규칙이 적용되더라도 예측이 가능하며, 안정적이고 플랜의 제어가 용이

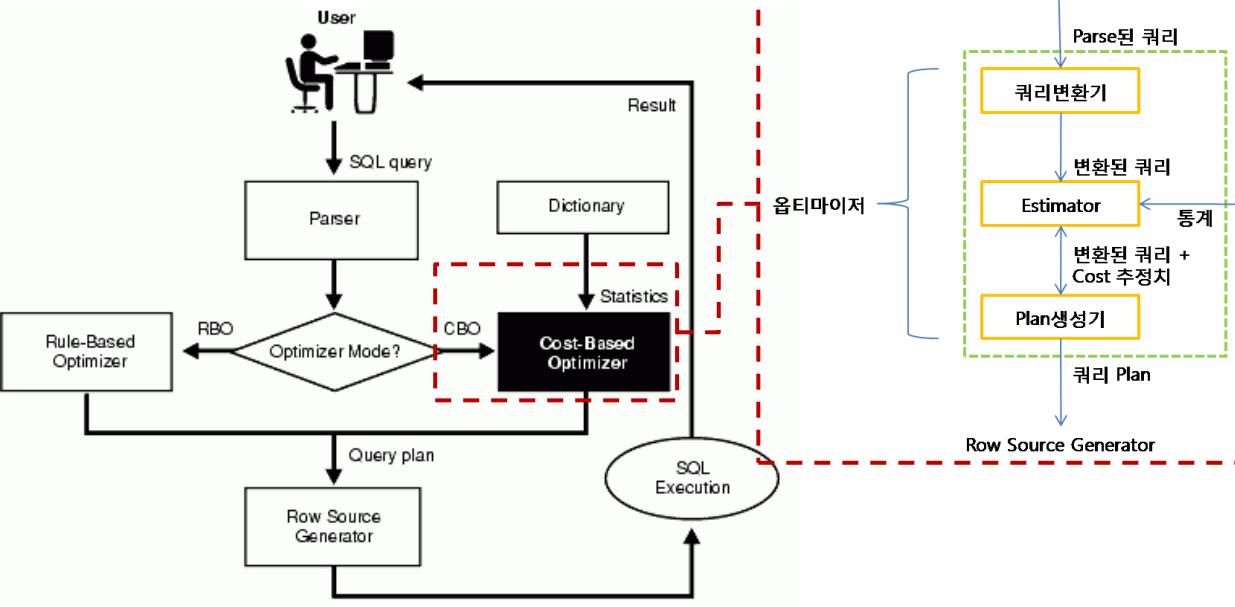
## 2. 비용기반 옵티マイ저, CBO(Cost Based Optimizer)의 개요

- 통계정보에 따른 비용을 계산**해 가장 **최소한의 비용이 소모되는 접근 경로(Access Path)**를 결정하는 옵티マイ저 유형
- 통계 데이터 : I/O 비용뿐만 아니라 CPU 연산 비용 및 메모리 비용을 포함함

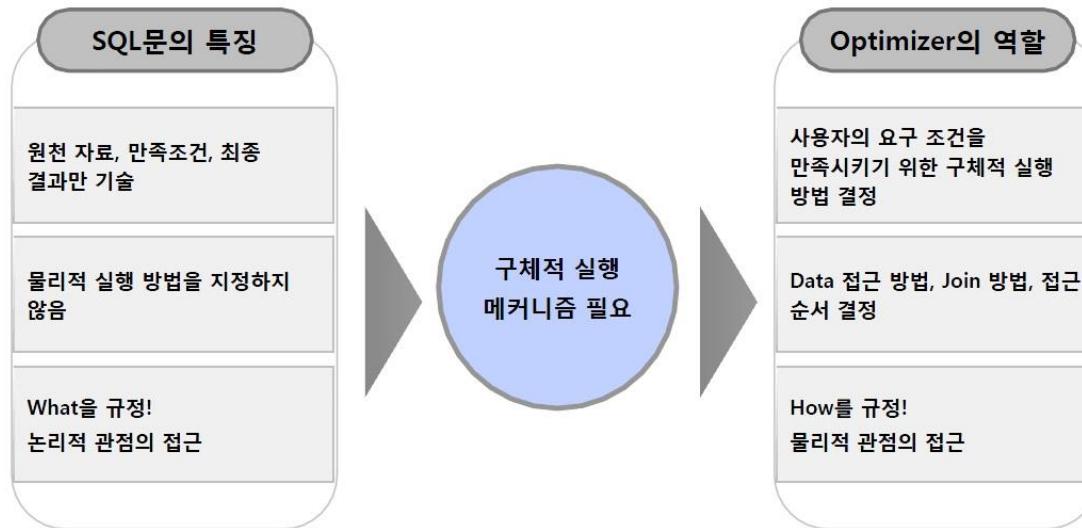
## 3. CBO의 장/단점

- 장점 : 통계정보와 I/O와 CPU 비용을 계산하여 실행계획을 예측
- 단점 : 원하는 경로로 유도하기 어려운 단점 (딕셔너리 정보의 정확성에 따라 성능이 상이함)

## 3. CBO 개념도



# SQL과 DB 옵티마이저



# 데이터베이스 3단계 구조

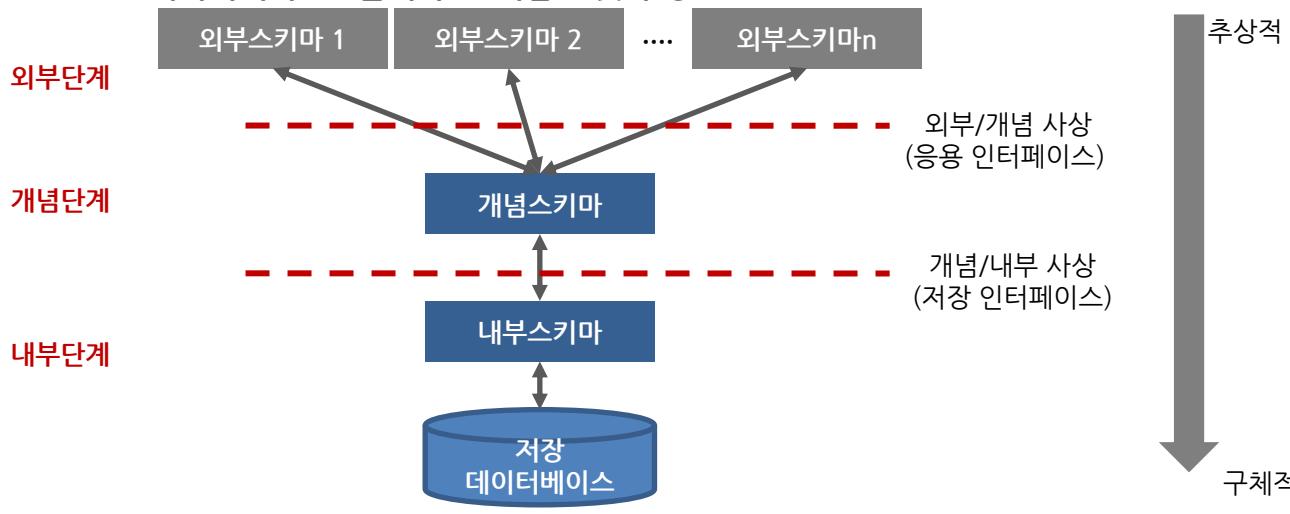
데이터베이스를 3단계 구조로 나누고 단계별로 스키마를 유지하며 스키마 사이의 대응 관계를 정의하는 궁극적인 목적  
→ 데이터 독립성의 실현

외개내

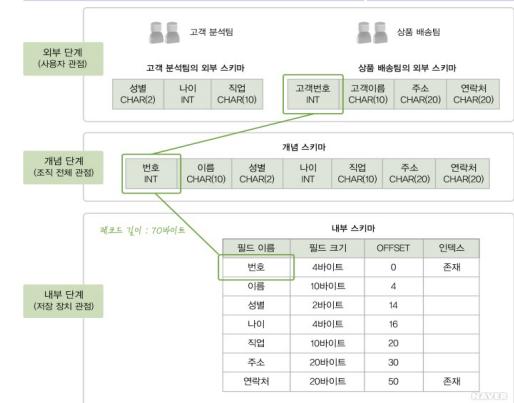
## 1. 데이터 독립성 보장을 위한 3단계 데이터베이스 구조(3-Level Database Architecture)의 개요

- 데이터베이스를 **관점(View)**에 따라 **3개의 계층으로 분리**하여 데이터베이스 사용자에게 내부적으로 **복잡한 데이터베이스 구조를 단순화**시킨 관점을 제공하는 사상
- 데이터베이스의 관리 층면**에서 데이터베이스를 외부/개념/내부 단계의 3단계로 구별하여 구조와 그 관계를 기술

## 2. 데이터베이스 3단계 구조 개념도 및 구성요소



단계	관점
외부단계 (External Level)	사용자관점
개념단계 (Conceptual Level)	현실세계 추상화
내부단계 (Internal Level)	물리적 저장장치 관점



스키마	내용	관리 정보
<b>외부 스키마 (External Schema)</b>	<ul style="list-style-type: none"> <li>데이터베이스의 각 <b>사용자나 응용 프로그래머가 접근하는 데이터베이스를 정의</b>한 것</li> <li>개인 및 특정 응용 프로그램에 제한된, 전체 데이터베이스의 한 논리적 부분</li> <li>서브스키마(sub schema)</li> </ul>	해당 응용 프로그램이나 사용자에 관련된 개체와 관계 정보
<b>개념 스키마 (Conceptual Schema)</b>	<ul style="list-style-type: none"> <li><b>전체 기관 입장에서 데이터베이스를 정의</b>한 것</li> <li>모든 응용시스템들이나 사용자들이 필요로 하는 데이터를 통합한, 조직 전체의 데이터 베이스를 기술한 것</li> <li>하나의 데이터베이스 <b>시스템에는, 하나의 개념 스키마만 존재</b></li> <li>모든 데이터 객체에 대한 정보와 효율적 관리를 위한 필수 정보도 포함(접근 권한, 보안 정책, 무결성 규칙)</li> </ul>	모든 데이터 객체 정보 (개체, 관계 및 제약 조건)
<b>내부 스키마 (Internal Schema)</b>	<ul style="list-style-type: none"> <li><b>저장장치(storage) 입장에서 데이터베이스 전체 저장 방법을 명세</b>한 것</li> <li>개념 스키마에 대한 <b>저장 구조를 정의</b></li> <li>내부레코드의 형식, 인덱스유무, 데이터 표현 방법 기술</li> </ul>	<ul style="list-style-type: none"> <li>저장 데이터 항목의 표현 방법</li> <li>내부 레코드의 물리적 순서 등</li> </ul>

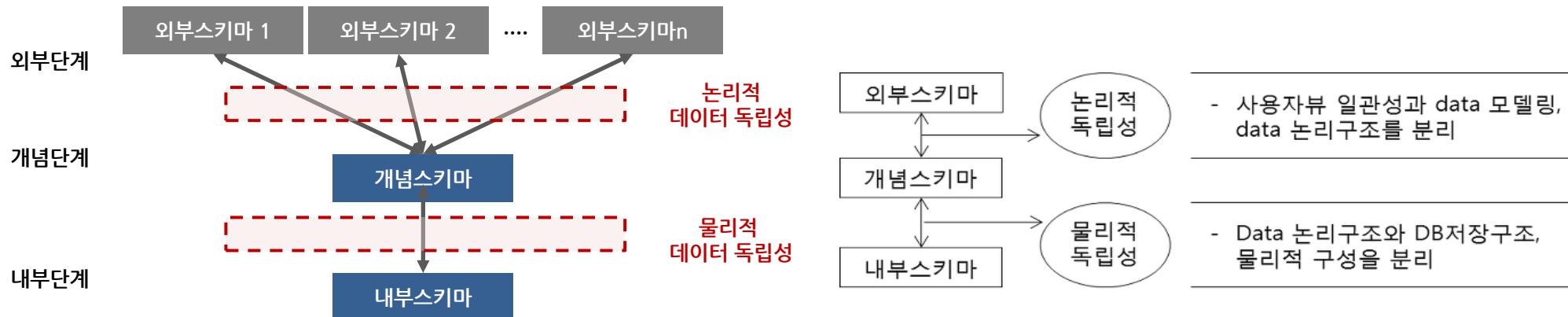
# 데이터 독립성(data independency)

논리적, 물리적

## 1. 계층구조 통한 논리적 독립성 확보, 데이터 독립성 개요

- 하위단계의 데이터구조가 변경되어도 상위단계에 영향을 미치지 않는 속성
- 데이터의 저장구조(Storage Structure)와 접근기법(Access Technique)으로부터 응용을 분리

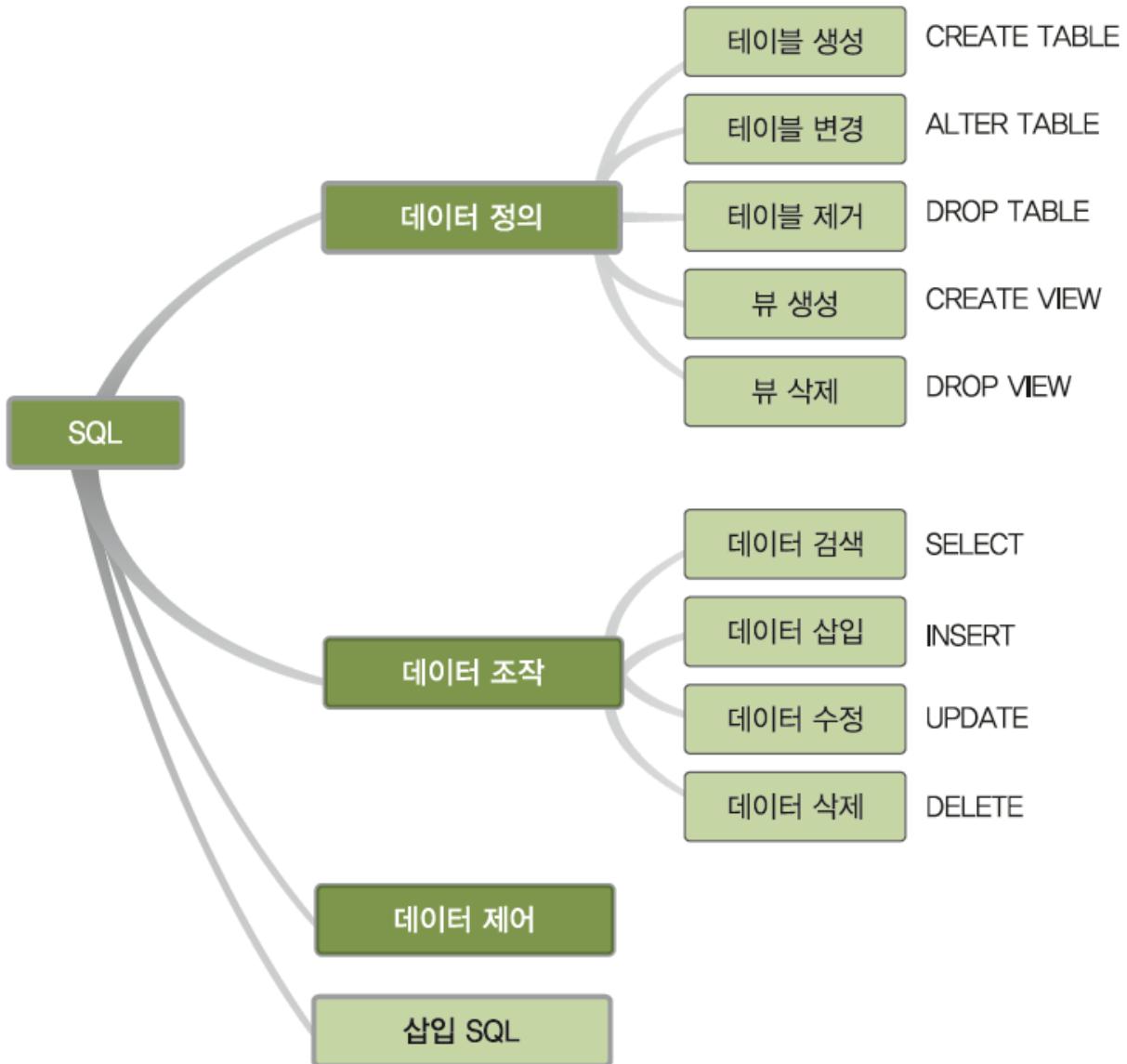
## 2. 데이터 독립성 개념도 및 종류



독립성	내용	목적
<b>논리적 독립성</b>	<ul style="list-style-type: none"> <li>데이터베이스의 논리적 구조를 변경시키더라도 기존 응용 프로그램에 영향을 주지 않는 것(<b>응용 프로그램과 자료구조를 독립시키는 것</b>)</li> <li>데이터베이스 관리시스템이 하나의 논리적 데이터 구조를 가지고 많은 응용 프로그램이 제각각 요구하는 다양한 형태의 논리적 구조로 사상(Mapping)시켜 줄 수 있어야 함</li> </ul>	<ul style="list-style-type: none"> <li>사용자 특성에 맞는 변경 가능</li> <li>통합구조 변경 가능</li> </ul>
<b>물리적 독립성</b>	<ul style="list-style-type: none"> <li><b>응용 프로그램과 논리적 구조에 영향 주지 않고, 데이터베이스의 물리적 구조를 변경</b></li> <li>하나의 논리적 구조로부터 여러 가지 상이한 물리적 구조를 지원할 수 있는 사상능력이 있어야 한다는 것</li> <li>시스템 성능(performance)을 향상시키기 위하여 필요</li> </ul>	<ul style="list-style-type: none"> <li>물리적 구조 변경 없이 개념구조 변경 가능</li> <li>개념구조 영향 없이 물리적인 구조 변경 가능</li> </ul>

- **스키마(schema)** : 데이터베이스에 저장되는 데이터 구조와 제약조건을 정의한 것
- **인스턴스(instance)** : 스키마에 따라 데이터베이스에 실제로 저장된 값

# SQL





# SQL (Structured Query Language)

DDL DML DCL

## 1. 데이터를 관리하기 위한 언어, SQL(Structured Query Language) 개요

- RDBMS에서 자료의 검색과 관리, 데이터베이스 스키마 생성과 수정, 데이터베이스 객체 접근 조정 관리등  
**데이터를 관리하기 위해 설계된** 특수 목적의 프로그래밍 언어 (사용자와 데이터베이스 관리 시스템 간의 통신 수단)

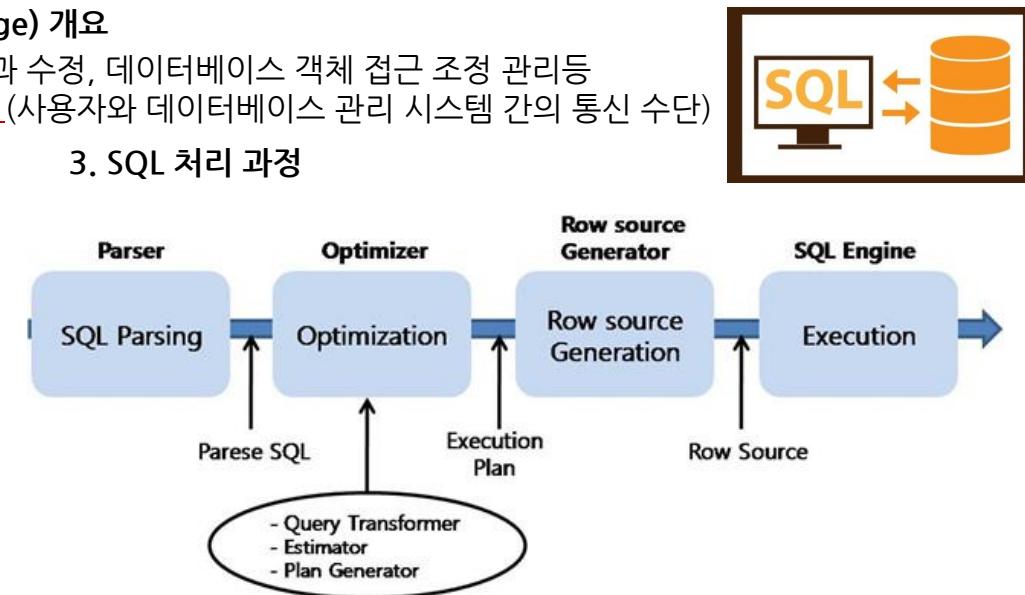
## 2. SQL 종류



## [참고자료]

파싱종류	설명
소프트파싱 (Soft Parsing)	SQL과 <b>실행계획을 캐시에서 찾아</b> 곧바로 실행단계로 넘어가는 경우 (예: PreparedStatement)
하드파싱 (Hard Parsing)	SQL과 실행계획을 캐시에서 찾지 못해 <b>최적화 과정을 거치고</b> 나서 실행단계로 넘어가는 경우(예: Statement)

## 3. SQL 처리 과정



엔진	역할
Parser	SQL 문장을 이루는 개별 구성요소를 분석하고 파싱해서 파싱 트리(내부적인 구조체)를 만든다. 이 과정에서 사용자 SQL에 문법적 오류가 없는지 (→ Syntax 체크), 의미상 오류가 없는지(→ Semantic 체크) 확인한다.
Optimizer	파싱된 SQL을 좀 더 일반적이고 표준적인 형태로 변환한다.
	오브젝트 및 시스템 통계정보를 이용해 쿼리 수행 각 단계의 선택도, 카디널리티, 비용을 계산하고, 궁극적으로는 실행계획 전체에 대한 총 비용을 계산해 낸다.
	하나의 쿼리를 수행하는 데 있어, 후보군이 될만한 실행계획들을 생성해낸다.
Row-Source Generator	옵티마이저가 생성한 실행계획을 SQL 엔진이 실제 실행할 수 있는 코드(또는 프로시저) 형태로 포맷팅한다.
SQL Engine	SQL을 실행한다.

# SQL (Structured Query Language)

## 4. 사용 방법에 따른 SQL 분류

### Static SQL

- String 형 변수에 쿼리 문장을 담지 않고 **코드 사이에 직접 기술**하는 SQL
- 하드 파싱
- Java 의 경우 statememt 구문 이용 하는 방식
- SQL Injection 위험 있음**

#### 1) STATIC SQL

```
SELECT E.EMPNO,
       E.ENAME,
       E.JOB,
       E.SAL,
       D.LOC
  FROM EMP E,
       DEPT D
 WHERE E.DEPTNO = D.DEPTNO
   AND E.JOB    = NVL(:P_JOB, E.JOB)
   AND E.SAL    BETWEEN NVL(:P_SAL_FROM, 0)
                     AND NVL(:P_SAL_TO, 1000000)
   AND D.LOC    = NVL(:P_SAL_LOC, D.LOC);
```

### Dynamic SQL

- String 변수에 담아서 처리하는 SQL문 즉 변수를 사용 하므로서 쿼리문을 동적으로 바꿀수 있으며 **런타임시에 사용자가 SQL에 대해서 입력(전체 입력 및 변경)하는 SQL**
- Runtime 시점에 사용자의 입력 값에 따라 동적으로 SQL 를 생성하여 실행 하는 방식의 SQL Binding 기법
- 자바에서 PreparedStatement 이용하는 방식

#### 2) DYNAMIC SQL

```
SELECT E.EMPNO,
       E.ENAME,
       E.JOB,
       E.SAL,
       D.LOC
  FROM EMP E,
       DEPT D
 WHERE E.DEPTNO = D.DEPTNO

 IF(:P_JOB IS NOT NULL) THEN
   AND E.JOB    = :P_JOB
 END IF;

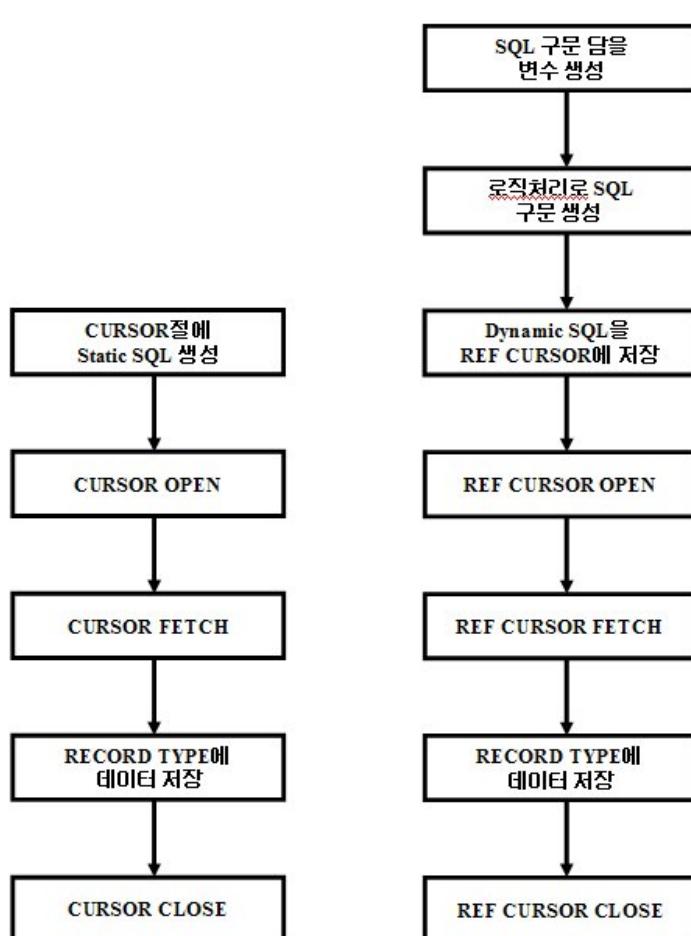
 IF(:P_SAL_FROM IS NOT NULL AND :P_SAL_TO IS NOT NULL) THEN
   AND E.SAL    BETWEEN :P_SAL_FROM
                     AND :P_SAL_TO
 END IF;

 IF(D.LOC IS NOT NULL) THEN
   AND D.LOC    = :P_LOC
 END IF;
```

특징	설명
문자열 변수 활용	<ul style="list-style-type: none"><li>SQL 구문을 Direct Invoke 하지 않고 String 등 문자열 변수를 이용해 동적으로 생성하여 실행</li></ul>
Runtime 시 구문 확정	<ul style="list-style-type: none"><li>Precompile 시점에 구문이 확정되지 않아 Syntax 체크 불가함</li></ul>
바인드 변수 활용 가능	<ul style="list-style-type: none"><li>Static SQL 과 동일하게 바인드 변수 활용 시 소프트 파싱 활용으로 라이브러리 캐시 효율 향상 가능</li></ul>

# SQL (Structured Query Language)

## 4. Static SQL 과 Dynamic SQL 처리 방식 및 두 SQL 방식의 비교



구분	Static SQL	Dynamic SQL
SQL 구성	<ul style="list-style-type: none"> <li>SQL구문을 CURSOR 선언하여 정적 처리</li> </ul>	<ul style="list-style-type: none"> <li>SQL 구문을 String형 변수에 담아 동적 처리</li> </ul>
개발 패턴	<ul style="list-style-type: none"> <li>Static SQL은 SQL 구문이 변경되지 않아야 하기 때문에 일반적인 개발 패턴은 Static SQL을 CURSOR 절에 선언한 뒤 이를 BEGIN END 절 사이에서 Looping 구조로 데이터 처리</li> </ul>	<ul style="list-style-type: none"> <li>Dynamic SQL은 SQL 구문이 변경 가능하므로 NVL()처리 불필요.</li> </ul>
컬럼 구성	<ul style="list-style-type: none"> <li>Static SQL은 정적이기 때문에 컬럼 및 Where절 변경 불가</li> </ul>	<ul style="list-style-type: none"> <li>Dynamic SQL은 구문을 변수에 담아서 DBMS를 콜하기 때문에 변수나 컬럼등 모든 SQL을 로직으로 처리하여 자유롭게 SQL구문 사용</li> </ul>
실행 계획	<ul style="list-style-type: none"> <li>Optimizer는 NVL()처리가 되어있는 조건을 처리하기 위해 IS NULL, IS NOT NULL로 나누어 실행계획 수립, 만약 6개 조건이 있다면 12개의 CONCATENATION으로 실행계획이 분할되어 실행 계획 수립을 위한 하드파싱 시간 장시간 소요</li> </ul>	<ul style="list-style-type: none"> <li>Optimizer는 NVL()처리된 WHERE 조건이 없기 때문에 실행계획을 꺼낼 필요가 없고 그만큼 순수 액세스 패스에 대하여 실행계획을 수립하기 때문에 하드파싱 시간 최적화</li> </ul>
장점	<ul style="list-style-type: none"> <li>Dynamic에 비해 실행속도 우수</li> <li>SQL문에 대하여 개발 시에 사전 검사가 가능</li> </ul>	<ul style="list-style-type: none"> <li>Application 내 각 SQL 구문에 대해 가장 최근 시점의 통계정보를 근거로 한 access plan 보유</li> <li>SQL 구문은 개발시가 아닌, 실행시에 확정되므로, 보다 다양하고 유연한 application 개발이 가능</li> </ul>
단점	<ul style="list-style-type: none"> <li>개발 시에 SQL 구문 정의 Precompile, bind과정 필요</li> </ul>	<ul style="list-style-type: none"> <li>Static에 비해 처리속도 느림</li> <li>실행전에 SQL 구문의 type, syntax, privilege checking이 불가능</li> </ul>

# Dynamic SQL(동적 SQL)

## 1. 동적 SQL의 장점

유형	장점	설명
개발 생산성 측면	개발 생산성 증가	<ul style="list-style-type: none"><li>SQL 구문 개수가 많은 경우 각각 Case 별 SQL 구문 개발 없이 Dynamic SQL문 구성으로 개발 생산성 증가</li></ul>
	유연한 응용프로그램 구현	<ul style="list-style-type: none"><li>다양한 유형 처리가 가능한 유연성 있는 프로그램 작성 가능</li></ul>
SQL 실행 측면	최근 통계정보 기반 Access Plan	<ul style="list-style-type: none"><li>Application 내 각 SQL 구문에 대해 가장 최근 시점의 통계정보를 근거로 한 access plan 보유</li></ul>
	하드파싱 시간 최적화	<ul style="list-style-type: none"><li>NVL() 미사용에 따른 실행계획 분리 필요 없음</li><li>순수 access path에 대한 실행계획 수립</li></ul>

## 2. 동적 SQL의 단점

유형	단점	설명
기능/개발 측면	개발 복잡도/난이도 증가	<ul style="list-style-type: none"><li>SQL 실행시점 분기에 따른 유연한 개발 능력 필요</li></ul>
	개발 공수 증가	<ul style="list-style-type: none"><li>단순 SQL Query 개발에 추가하여 변수 처리 로직 필요</li></ul>
	로직 이해도	<ul style="list-style-type: none"><li>Static 대비 바인드 변수 사용으로 Query 이해 난이도 높음</li></ul>
성능/보안 측면	성능 저연 이슈	<ul style="list-style-type: none"><li>어플리케이션 Cursor 캐싱 기능 비작동 가능하여 성능 문제 발생 가능함 (Loop 내 반복 Query 수행 등)</li></ul>
	SQL Injection 등 보안 취약	<ul style="list-style-type: none"><li>실행시점에 의도하지 않는 변수 삽입 공격 가능</li></ul>
	PreCompile 검증제약	<ul style="list-style-type: none"><li>사전 Compile 없이 실행시점에 SQL 구문 분석 처리로 Static SQL 대비 안전성 낮음 (구분분석/오브젝트/권한 등 사전검증 제한)</li></ul>

## 3. 동적 SQL의 단점 해결방안 설명

유형	해결방안	설명
기능/개발 측면	개발 표준화, 표준 F/W 활용	<ul style="list-style-type: none"><li>동적 SQL 활용 개발 표준 준수</li><li>빠르고 안전한 동적 SQL 활용 가능한 프레임워크 적용</li></ul>
	SQL 주석표기 (코딩 표준)	<ul style="list-style-type: none"><li>SQL 및 바인딩 변수 설명 이해가 가능하도록 주석표기 표준화</li></ul>
성능/보안 측면	HW 성능 확보/튜닝	<ul style="list-style-type: none"><li>Query 성능 최적화 (튜닝)</li><li>HW 사용자 원 검토 및 처리 능력 확보</li></ul>
	입력 값 검증 개발 품질 검증 도구 활용	<ul style="list-style-type: none"><li>특정 바인딩 변수 입력 제한 및 필터링 Query/Logic 오류 등 Dynamic SQL 쿼리 검증이 가능한 개발 지원 도구 활용</li></ul>

# 데이터 정의 언어 (DDL, Data Definition Language)

- DB 스키마 객체에 대한 생성, 변경, 제거 등 구조를 변경하는 언어. 데이터 정의 언어 (DDL, Data Definition Language)의 간단한 예제
- 데이터베이스를 정의하거나 수정할 목적으로 사용하는 언어

## 2. DDL 문 종류

명령어	기능 설명	문법
<b>CREATE</b>	새로운 데이터베이스 객체 생성	<pre><b>CREATE TABLE</b> table_name ({{col_name data_type [NOT NULL] [DEFAULT],}+} [PRIMARY KEY (column_list),] {{[UNIQUE(column_list),]}*} {{[FOREIGN KEY(column_list) REFERENCES table_name[(column_list)]} [ON DELETE option] [ON UPDATE option], ]}*} [CONSTRAINT constraint_name] [CHECK(conditional_expression)]);</pre>
<b>ALTER</b>	존재하는 데이터베이스 객체 변경	<b>ALTER TABLE</b> table_name [ADD  RENAME MODIFY DROP ] column_name data_type;
<b>DROP</b>	존재하는 데이터베이스 객체 제거	<b>DROP TABLE</b> table_name {RESTRICT CASCADE}
<b>TRUNCATE</b>	테이블에서 <u>데이터를 삭제</u>	<b>TRUNCATE TABLE</b> test_table
<b>RENAME</b>	테이블의 이름을 변경	<b>RENAME</b> old_table_name <b>TO</b> new_table_name

# 데이터 조작언어 (DML, Data Manipulation Language)

## 1. 스키마의 데이터 조작 및 질의를 위한 언어, DML 개요

- 데이터베이스 내부 스키마에 데이터를 입력(Insert), 수정(Update), 삭제(Delete)하거나 조회(Select)하기 위한 언어 또는 명령어
- 사용자가 DBMS로 하여금 원하는 Data를 처리하게 명세하는 언어

## 2. DML 구분

- 절차적 데이터 조작어(procedural DML): 사용자가 어떤(**what**) 데이터를 원하고, 그 데이터를 얻기 위해 어떻게(**how**) 처리해야 하는지도 설명
- 비절차적 데이터 조작어(nonprocedural DML): 사용자가 어떤(**what**) 데이터를 원하는지만 설명

## 3. DML 문 종류

명령어	기능 설명	문법
<b>INSERT</b>	데이터 입력 구문	INSERT INTO 테이블명 ( 컬럼명1, 컬럼명2 ) VALUES ( 입력값1, 입력값2 );
<b>UPDATE</b>	데이터 수정 구문	UPDATE 테이블명 SET 컬럼명1 = 입력값1, 컬럼명2 = 입력값2 WHERE 조건절
<b>DELETE</b>	데이터 삭제 구문	DELETE FROM 테이블명 WHERE 조건절
<b>SELECT</b>	데이터 조회 구문	SELECT 컬럼명1, 컬럼명2 FROM 테이블명 WHERE 조건절
<b>MERGE</b>	입력/수정/삭제의 동시 작업 구문	MERGE INTO 테이블명1 USING 테이블명A ON ( 조인조건 ) WHEN MATCHED THEN UPDATE SET 컬럼명1 = 컬럼명A DELETE WHERE 조건절 WHEN NOT MATCHED THEN INSERT (컬럼명1, 컬럼명2) VALUES (컬럼명A, 컬럼명B) ;

# 데이터 제어 언어 (DCL, Data Control Language)

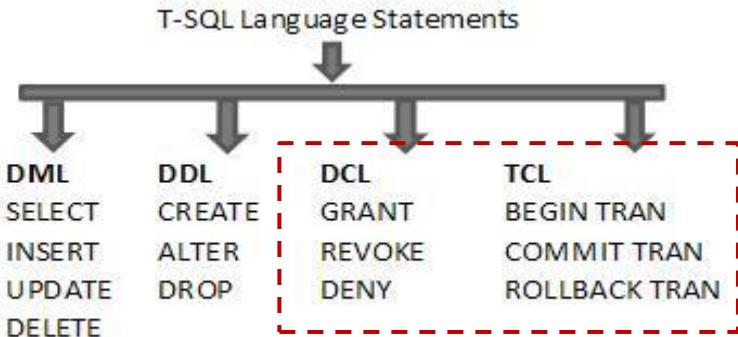
## 1. 데이터보안, 무결성, 회복, 병행수행 제어 등을 명세하는 언어, DCL의 개요

- 객체 권한 부여등의 제어어 (GRANT 등) **데이터의 보안, 무결성, 데이터 회복, 병행 수행 제어 등을 정의하는 데 사용하는 언어**

## 2. 목적

- 무결성 : 정확하고 유효한 데이터만 유지
- 보안 : 허가받지 않은 사용자의 데이터 접근 차단, 허가된 사용자에 대한 권한 부여
- 회복 : 장애가 발생해도 데이터 일관성 유지
- 동시성 제어 : 동시 공유 지원

## 3. DCL 문 종류



명령어	기능 설명	문법
COMMIT	트랜잭션 완료	- COMMIT 명령어는 이처럼 INSERT 문장, UPDATE 문장, DELETE 문장을 사용한 후에 이런 변경 작업이 완료되었음을 데이터베이스에 알려 주기 위해 사용 - Commit;
ROLLBACK	변경 사항을 취소	ROLLBACK;
SAVEPOINT	현 시점에서 SAVEPOINT까지 트랜잭션의 일부만 롤백	SAVEPOINT SVPT1;
GRANT	시스템권한 또는 객체에 대한 권한 부여	GRANT [권한명] ON [대상 객체] TO [계정] GRANT select, insert, delete, update ON student TO userA
REVOKE	시스템권한 또는 객체에 대한 권한 회수	REVOKE [권한명] ON [대상 객체] FROM [계정]; REVOKE select, insert, delete, update ON student FROM userA;

### [데이터 무결성]

- 데이터 모델링 과정에서 정의된 일련의 규칙에 따라 데이터가 생성, 수정, 삭제 될 수 있도록 프로그램이나 데이터베이스 기능을 이용하는데, 그 결과로 권한이 부여된 사용자에 의해 야기될 수 있는 의미적 에러를 방지하고, 데이터베이스 내의 데이터가 현실 세계의 올바른 데이터를 갖도록 보장하는 것

# DDL, DCL, DML

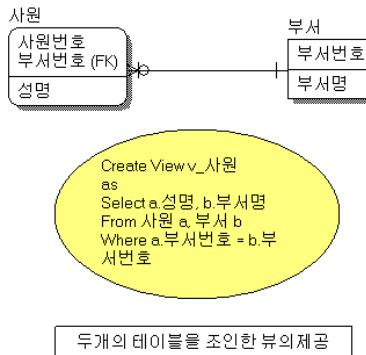
DML Data Manipulation Language	DDL Data Definition Language	TCL Transaction Control Language	DCL Data Control Language
데이터 조작언어는 데이터 내에 데이터를 입력, 수정, 삭제 할 수 있는 SQL 언어	데이터 베이스 내에서 객체를 생성하고 변경하고 삭제하기 위해서 쓰이는 언어	DML을 통해 변경된 데이터가 데이터베이스에 저정되기 위해 사용되는 언어  Commit이나 Rollback 하지 않으면 데이터 수정 작업 취소됨.	권한을 부여할 때 쓰이는 언어
<b>종류</b> 1. Insert 2. Update 3. Delete 4. Merge 5. Multiple-Insert  DDL의 TRUNCATE 와의 차이점 중요	<b>종류</b> 1. Create : 객체 생성 2. Alter : 객체 구조 변경 3. Drop : 객체 삭제 4. Rename : 이름변경 5. Comment : 주석달기 6. Truncate : 테이블 모든 행 삭제  DML의 DELETE 와의 차이점 중요	<b>종류</b> 1. Commit 2. RollBack 3. Savepoint : ANSI 표준이 아니어서 다른 DB에서 없을 수 있다.	<b>종류</b> 1. Grant 2. Revoke
Auto Commit이 아님	Auto Commit		Auto Commit

# 뷰(View)

## 1. 가상적으로 만든 테이블, 뷰의 개요.

- 이미 존재하는 하나 혹은 그 이상의 테이블에서 원하는 데이터만 가져올 수 있도록 미리 원하는 컬럼만 모아 가상적으로 만든 테이블
- 사용자에게 특정 테이블의 컬럼을 감추어 직접적인 접근제어 가능
- 특정 테이블을 미리 조인한 뷰를 제공하여, 사용자는 간단한 SQL문을 사용.

## 2. 뷰의 개념도 및 사용을 위한 쿼리문



구분	내용	비고
생성	CREATE VIEW viewTitles AS SELECT title_id, price FROM pub..titles	CREATE
조회	SELECT * FROM viewTitles	SELECT
수정	ALTER VIEW viewTitles AS SELECT title, price FROM pub..titles	ALTER
삭제	DROP VIEW view이름	DROP

## 3. 뷰의 종류

구분	설명
시스템 뷰(System View)	테이블의 제약, 색인등에 대한 정보 열람 가능.
인덱스된 뷰(Indexed View)	실제 데이터를 가져와서 직접 실시간 연산을 하고 처리하는 시간이 늦어지므로 성능향상을 높이고자 할 때 사용됨.
분할된 뷰(Partitioned View)	여러대의 SQL서버에 테이블을 나누어 저장. 한대의 SQL서버로 처리할수 없는 대량의 데이터일 때 사용하여 성능향상을 시킴.

# Sub-Query (부질의, 서브쿼리)

## 1. 하나의 SQL문 안에 포함되어 있는 또 다른 SQL, 서브 쿼리

- 메인 쿼리에 포함되는 종속적인 관계의 쿼리로 하나의 SQL문 안에 포함되어 있는 또 다른 SQL문

## 2. 서브 쿼리의 종류

구분	종류	설명
위치별	Nested Subquery	<ul style="list-style-type: none"> <li><u>Where절에 위치</u>하고, 가장 먼저 개발됨</li> </ul>
	Inline View	<ul style="list-style-type: none"> <li><u>FROM절에 위치</u>하고, SQL 내 절차성 효과</li> </ul>
	Scalar Subquery	<ul style="list-style-type: none"> <li><u>SELECT절에 위치</u>하고, 가장 최근에 개발됨</li> </ul>
반환데이터 형태별	단일행 서브쿼리	<ul style="list-style-type: none"> <li>서브쿼리의 <u>실행결과가 항상 1건 이하인</u> 서브쿼리. 단일행 비교연산자 (=, &lt;, &lt;=, &gt;, &gt;=, &lt;&gt;)와 함께 사용된다.</li> </ul>
	다중행 서브쿼리	<ul style="list-style-type: none"> <li>서브쿼리의 <u>실행결과가 여러 건인</u> 서브쿼리. 다중행 비교연산자 (IN, ALL, ANY, SOME, EXISTS)와 함께 사용된다</li> </ul>
	다중칼럼 서브쿼리	<ul style="list-style-type: none"> <li><u>서브쿼리의 실행결과로 여러 칼럼을 반환한다</u>. 메인쿼리의 조건절에 여러 칼럼을 동시에 비교할 수 있다. 서브쿼리와 메인쿼리에서 비교하고자 하는 칼럼개수와 위치가 동일함</li> </ul>
동작방식별	비연관 서브쿼리	<ul style="list-style-type: none"> <li>서브쿼리가 메인쿼리 칼럼을 가지고 있지 않는 형태의 서브쿼리. 메인쿼리에 값(서브쿼리가 실행된 결과)을 제공하기 위한 목적으로 주로 사용됨</li> </ul>
	연관 서브쿼리	<ul style="list-style-type: none"> <li>서브쿼리가 메인쿼리의 칼럼을 사용하는 형태의 서브쿼리. 메인쿼리가 먼저 수행되어 읽혀진 데이터를 서브쿼리에서 조건이 맞는지 확인하고자 할 때 사용된다.</li> </ul>

### [Nested Subquery / 단일행 서브쿼리]

```
SELECT PLAYER_NAME 선수명, POSITION 포지션, BACK_NO 백넘버
FROM PLAYER_T
WHERE TEAM_ID = (SELECT TEAM_ID
    FROM PLAYER_T
    WHERE PLAYER_NAME = '김남일')
ORDER BY PLAYER_NAME;
```

### [Inline View / 다중행 서브쿼리]

```
SELECT EMPNO
FROM (SELECT EMPNO, ENAME FROM EMP ORDER BY MGR);
```

# Join 연산

Join 유형 구분하기 (특정 짓지 않는다면 모두 다 쓴다!!)

\* 참고자료 : <https://stackoverflow.com/questions/406294/left-join-vs-left-outer-join-in-sql-server>

## 1. 둘 이상의 테이블에 전부 존재하는 데이터만 조회, 조인연산 개요

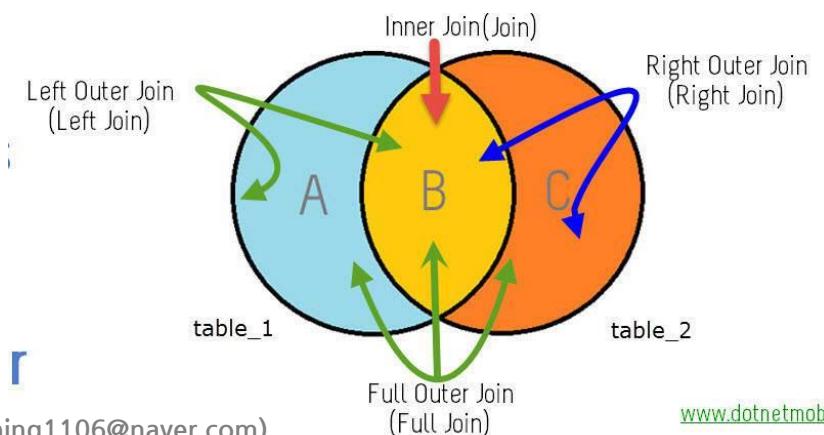
- **하나의 SQL 명령문에 의해 여러 테이블에 저장된 데이터를 한번에 조회**할 수 있는 기능.
- 둘 이상의 테이블 간의 논리적 관계를 기준으로 데이터를 검색하여 결과 집합을 만드는 기능

## 2. Join연산의 유형

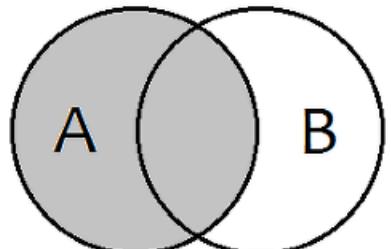
- **논리적 Join** : Join의 종류/유형
- **물리적 Join** : Join의 방식/수행 알고리즘

유형	종류	설명
물리적 조인 (Hint 사용)	Nested Loop Join	<ul style="list-style-type: none"> <li>• <b>선행 테이블</b>의 처리범위를 하나씩 액세스하면서 그 <b>추출된 값으로 연결할 테이블을 조인</b>하는 방식</li> </ul>
	Sort Merge Join	<ul style="list-style-type: none"> <li>• <b>양쪽 테이블</b>의 처리범위를 <b>각자</b> 액세스하여 <b>정렬</b>한 결과를 차례로 scan하면서 연결고리의 조건으로 merge해 가는 방식</li> </ul>
	Hash (Match) Join	<ul style="list-style-type: none"> <li>• 해시값을 이용하여 테이블을 조인하는 방식</li> </ul>
논리적 조인 (쿼리 사용)	Inner Join	<ul style="list-style-type: none"> <li>• 둘 이상의 테이블에 전부 존재하는 데이터만 조회하는 방식 (교집합)</li> <li>• 종류 : 동등조인, 자연조인, 비동등조인, 셀프조인</li> </ul>
	Outer Join	<ul style="list-style-type: none"> <li>• 상대 릴레이션에서 대응되는 튜플을 갖지 못하거나, 조인 애트리뷰트에 <b>널 값이 들어 있는 튜플들을 다 루기 위해서 확장된 조인 연산</b></li> <li>• 종류 : Left Outer Join, right Outer Join, Full Outer Join</li> </ul>

Table	Join Type	Table	Statement	What we use	Visualization
A	Join	B	A Inner Join B	A Inner Join B	
			A Left Outer Join B	A Left Join B	
			A Full Outer Join B	A Full Join B	
			A Right Outer Join B	A Right Join B	
			A Cross Outer Join B	A Cross Join B	
			A Natural Join B	A Natural Join B	

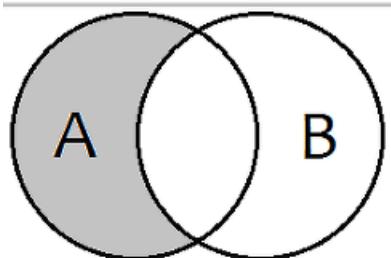


# Join 연산

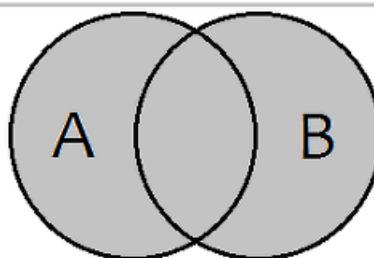


```
SELECT *
FROM TableA a
LEFT JOIN TableB b
ON a.Key = b.Key
```

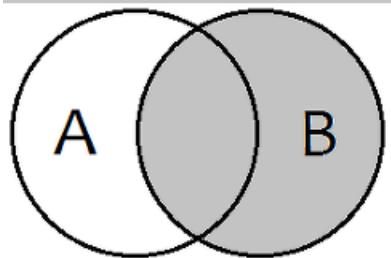
## SQL JOINS



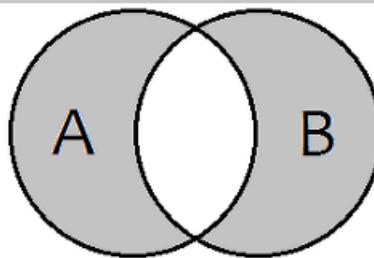
```
SELECT *
FROM TableA a
LEFT JOIN TableB b
ON a.Key = b.Key
WHERE b.Key IS NULL
```



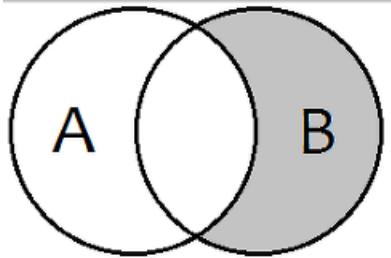
```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
ON a.Key = b.Key
```



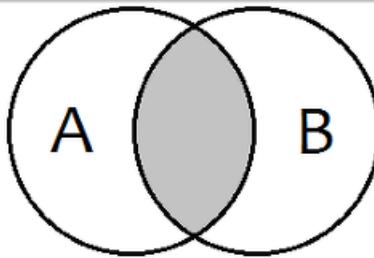
```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
ON a.Key = b.Key
```



```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
ON a.Key = b.Key
WHERE a.Key IS NULL
OR b.Key IS NULL
```



```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
ON a.Key = b.Key
WHERE a.Key IS NULL
```

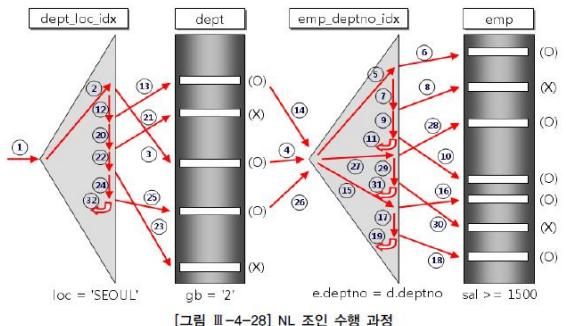


```
SELECT *
FROM TableA a
INNER JOIN TableB b
ON a.Key = b.Key
```

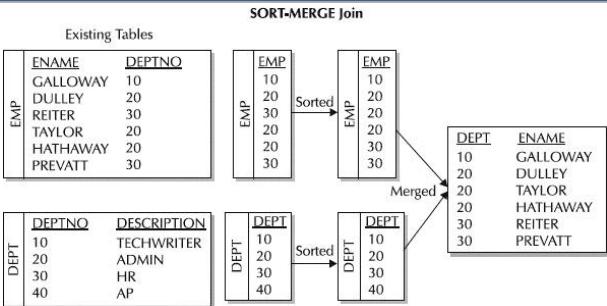
# [Join 연산] Join 방식(물리적 조인, 알고리즘)

NL, SM, H

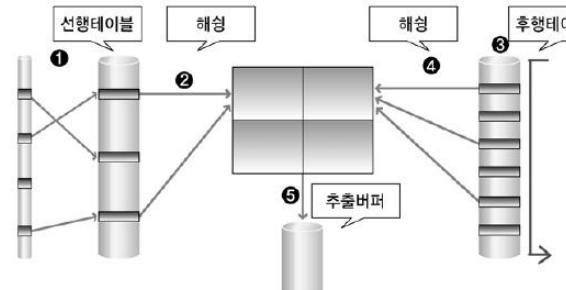
## 중첩반복(Nested Loops) 조인



## 정렬병합(Sort Merge) 조인



## 해시매치(Hash Match) 조인, 해시(Hash) 조인



[그림 II-3-14] Hash Join

- 선행테이블(드라이빙 테이블)의 처리범위를 하나씩 액세스하면서 그 추출된 값으로 연결할 테이블(후행 테이블)을 조인하는 방식
- 실행속도 = 선행 테이블 사이즈 \* 후행 테이블 접근횟수
- 쿼리  
select /\*+ use\_nl(b, a) \*/ a.dname, b.ename  
from emp b, dept a  
where a.loc = 'NEW YORK'  
and b.deptno = a.deptno
- Driving table(선행테이블) : 먼저 조회하는 테이블
- Driven Table (후행테이블) : 나중에 조회하는 테이블

- 조인의 대상범위가 넓을 때 발생하는 랜덤 액세스를 줄이기 위한 경우나 연결고리에 마땅한 인덱스가 존재하지 않을 때 해결하기 위한 대안
- 쿼리  
select /\*+ use\_merge(a, b) \*/ a.dname, b.empno, b.ename  
from depta,emp b  
where a.deptno = b.deptno  
and b.sal > 1000 ;

- 해싱 함수(Hashing Function) 기법을 활용하여 조인을 수행하는 방식(해싱 함수는 직접적인 연결을 담당하는 것이 아니라 연결될 대상을 특정 지역(partition)에 모아두는 역할만을 담당)
- 쿼리  
select /\*+ use\_hash(a, b) \*/ a.dname, b.empno, b.ename  
from dept a, emp b  
where a.deptno = b.deptno  
and a.deptno between 10 and 20;

### Nested Loop JOIN

- 부분범위 처리에 적합
- 소량 데이터 처리에 적합
- 온라인 프로그램에 적합
- 일반적으로 흔하게 발생
- 온라인 쿼리 약 80% 분포
- 테이블 접근 순서가 중요
- 조인절에 인덱스 없을 때 발생
- 순차적 접근

### Sort Merge JOIN

- 전체범위 처리에 적합
- 대량 데이터 처리에 적합
- 사용하지 않는 것이 최선
- Sort 회피 가능 테이블 선정
- Sort 부하가 가장 큰 관계
- Sort + Merge 접근

### Hash JOIN

- 전체범위 처리에 적합
- 대량 데이터 처리에 적합
- 배치 프로그램에 적합
- 대량 접계 작업 때 발생
- 배치 쿼리 약 50% 분포
- 작은 테이블로 해쉬 구성
- Memory 크기에 영향
- Hash 함수 이용한 접근

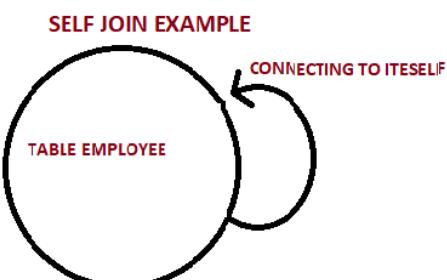
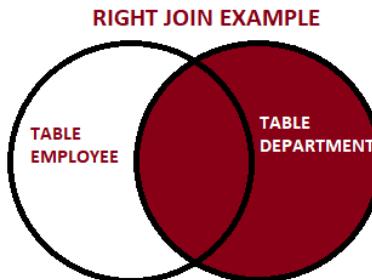
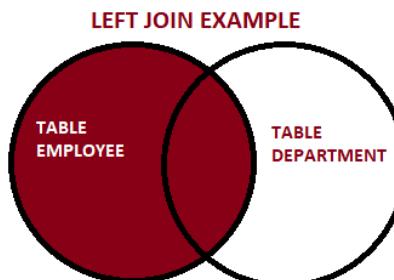
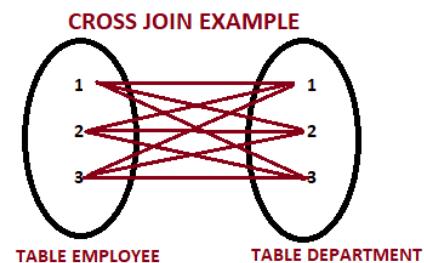
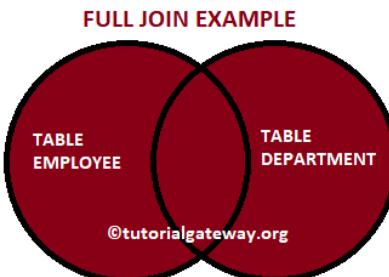
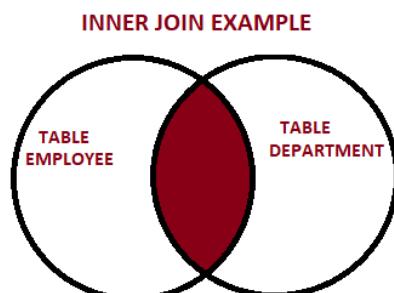
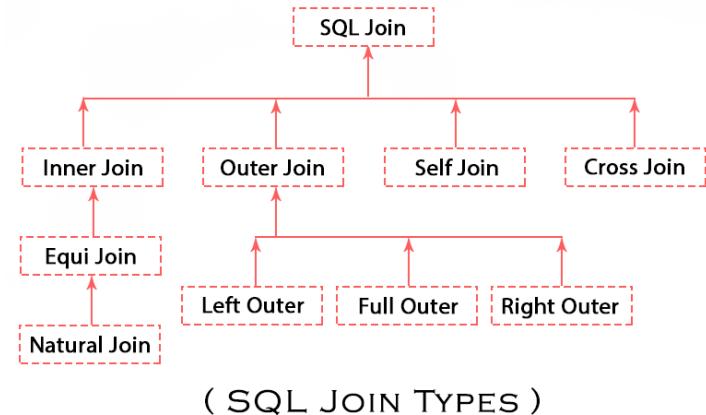
#### \* 참고자료 :

- <http://denodo1.tistory.com/299>
- <https://logicalread.com/oracle-11g-sort-merge-joins-mc02/>
- <http://www.gurubee.net/lecture/2388>

# [Join 연산] Join 종류(논리적 조인)

\* 참고자료 : <https://way2tutorial.com/>

구분	설명
Inner Join	조인 할 테이블들간에 조인 Key값이 상호 일치하는 Row를 ResultSet으로 생성하는 연산
Outer Join	조인 Key값에 대응되는 Row가 없는 경우 제거하지 않고 Null Tuple로 추출하는 연산
Natural Join (자연조인)	- 두 릴레이션의 공통된 속성을 매개체로 두 릴레이션의 정보를 관계로 묶어내는 연산 - 중복된 공통된 속성 중 하나는 제거됨
Equal Join (동일조인)	- 두 릴레이션의 공통된 속성을 매개체로 두 릴레이션의 정보를 관계로 묶어내는 연산 - 중복되는 공통의 속성은 모두 표시됨
Cross Join	- 연결되는 모든 테이블들의 모든 행들의 모든 조합을 결과에 포함 - 연결된 테이블의 Cartesian product를 반환함
Semi Join	- R과 S를 자연 조인한 결과에 R의 애트리뷰트로 프로젝트

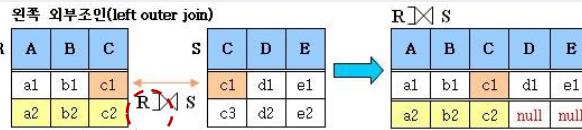
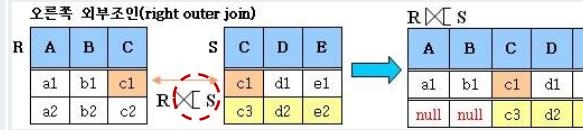
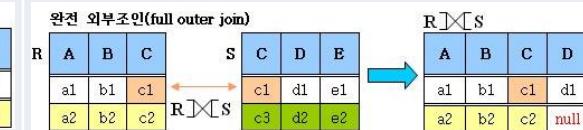
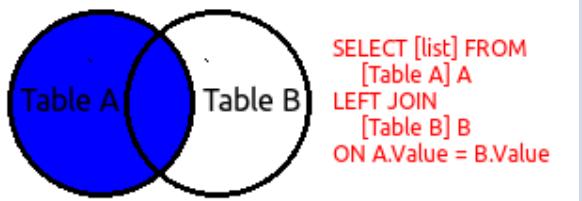
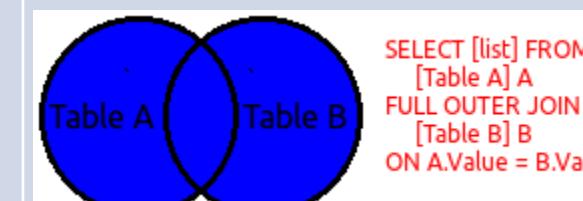
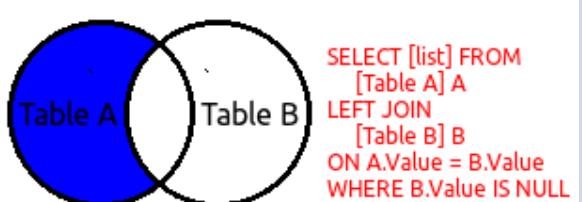
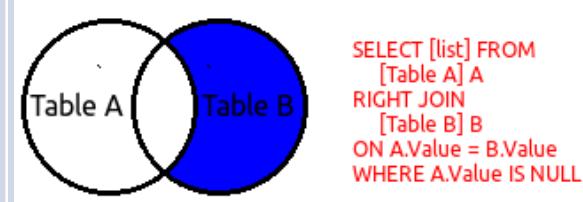
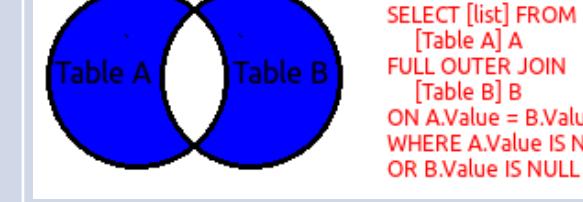


# [Join 연산] Outer Join

## 1. 외부조인(Outer Join)의 개념

- 상대 릴레이션에서 대응되는 투플을 갖지 못하거나, 조인 애트리뷰트에 널 값이 들어 있는 투플들을 다루기 위해서 확장된 조인 연산.

## 2. 외부조인의 종류

왼쪽 외부 조인 (Left Outer Join)	오른쪽 외부 조인 (Right Outer Join)	완전 외부 조인 (Full Outer Join)
<ul style="list-style-type: none"> <li>자연 조인과 다르게 <u>왼쪽의 릴레이션을 기준</u>으로 결과 릴레이션을 생성함</li> <li>자연조인의 경우에는 연관관계가 있는 투플만을 결과 릴레이션을 나타내는 반면, 왼쪽외부조인의 경우는 <u>왼쪽 릴레이션의 투플들을 모두 나타낸 후 대응되는 값이 없다면 NULL로 대체</u></li> </ul> <p>왼쪽 외부조인(left outer join)  </p>	<ul style="list-style-type: none"> <li>왼쪽 외부 조인과는 반대로 <u>오른쪽 릴레이션을 기준으로 결과 릴레이션을 생성</u></li> </ul> <p>오른쪽 외부조인(right outer join)  </p>	<ul style="list-style-type: none"> <li><u>양쪽의 릴레이션 모두를 기준으로 결과 릴레이션을 생성</u>함.</li> <li>대응되는 값이 없다면 양쪽 다 NULL 값을 삽입</li> </ul> <p>완전 외부조인(full outer join)  </p>
 <p>SELECT [list] FROM [Table A] A LEFT JOIN [Table B] B ON A.Value = B.Value</p>	 <p>SELECT [list] FROM [Table A] A RIGHT JOIN [Table B] B ON A.Value = B.Value</p>	 <p>SELECT [list] FROM [Table A] A FULL OUTER JOIN [Table B] B ON A.Value = B.Value</p>
 <p>SELECT [list] FROM [Table A] A LEFT JOIN [Table B] B ON A.Value = B.Value WHERE B.Value IS NULL</p>	 <p>SELECT [list] FROM [Table A] A RIGHT JOIN [Table B] B ON A.Value = B.Value WHERE A.Value IS NULL</p>	 <p>SELECT [list] FROM [Table A] A FULL OUTER JOIN [Table B] B ON A.Value = B.Value WHERE A.Value IS NULL OR B.Value IS NULL</p>

# DB Hint

## 1. SQL 실행 계획을 제어하는 도구, Hint의 개요

- 데이터베이스의 실행계획을 주도하는 **옵티マイ저에게 원하는 실행 계획으로 유도**하도록 사용하는 도구

## 2. hint의 특징

구분	설명
실행 계획 제어	<ul style="list-style-type: none"><li>새로운 인덱스를 생성하는 경우 존재</li><li>SQL에 힌트를 설정하여 <b>옵티マイ저에게 올바른 실행 계획을 생성하도록 유도</b>할 수 있음</li></ul>
에러 미발생	<ul style="list-style-type: none"><li>힌트는 잘못 작성하여도 해당 <b>SQL은 에러를 발생시키지 않음(다)</b>, 문법 오류 제외)</li></ul>
옵티マイ저에 의한 취사선택 가능	<ul style="list-style-type: none"><li>힌트의 문법이 올바르더라도 힌트는 옵티マイ저에 의해 버려질 수도 있고 선택되어질 수도 있다. (<b>힌트의 취사선택</b>)</li></ul>

## 3. Hint 문법

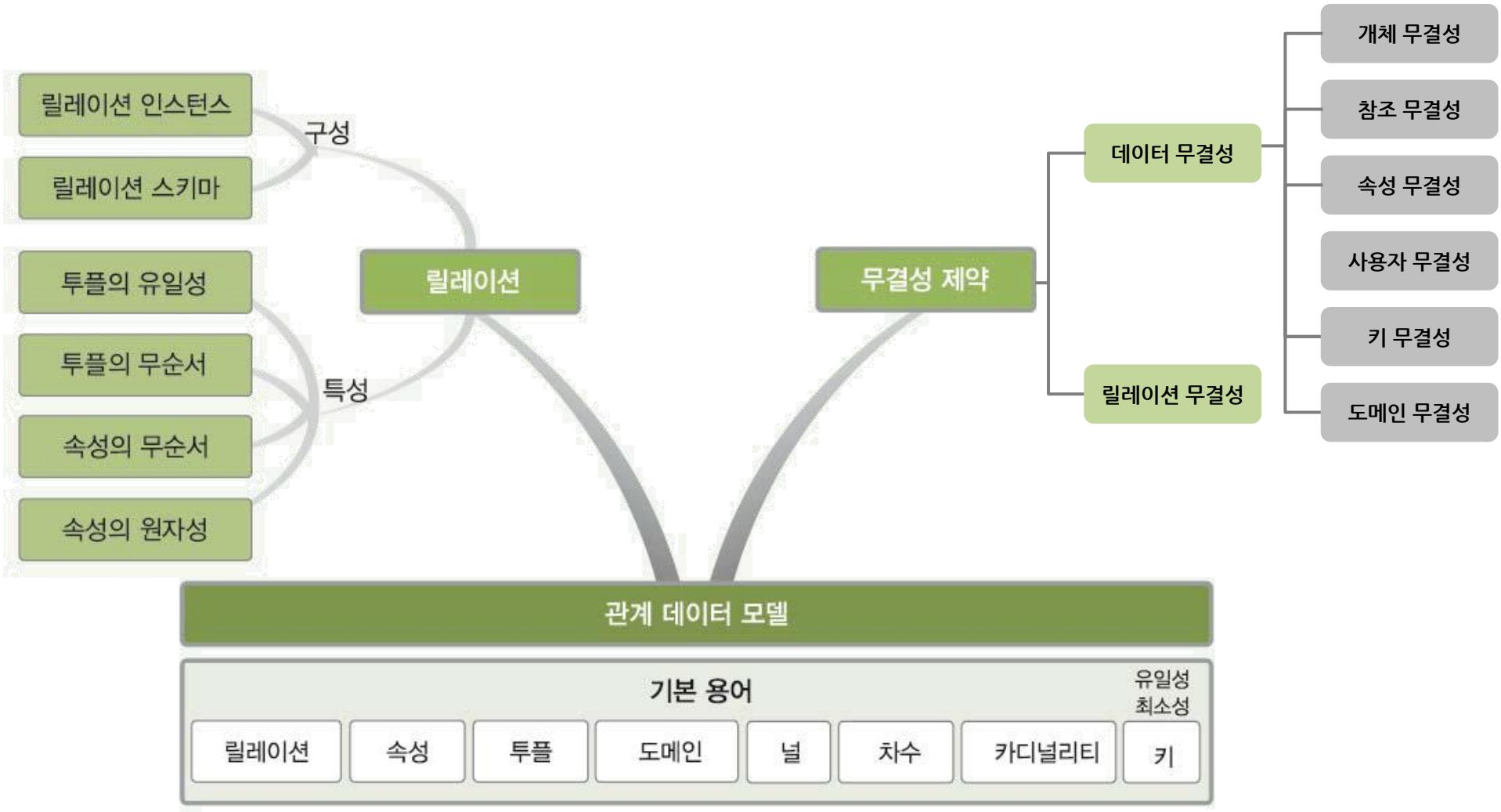
- 기본 문법 : **L\*+ hint L\***

• 힌트를 포함하는 주석은 SELECT, UPDATE, DELETE, INSERT 키워드 다음에만 사용 가능  
• 다수의 Hint 적용 가능

## 4. Hint 종류

구분		설명
최적화 목표(GOAL) 제어힌트	ALL_ROWS	<ul style="list-style-type: none"><li>쿼리의 전체 결과를 모두 수행 하는 것에 대한 최적화를 목표로 최저비용의 실행계획을 수립하도록 유도</li></ul>
	FIRST_ROWS	<ul style="list-style-type: none"><li>최적 응답시간을 목표로 최저 비용의 실행 계획을 수립하도록 유도</li></ul>
	RULE	<ul style="list-style-type: none"><li>규칙기준 옵티マイ저를 이용하여 최적화를 요구.</li></ul>
	CHOOSE	<ul style="list-style-type: none"><li>엑세스 하는 테이블의 통계정보 유무에 따라 규칙기준 또는 비용기준을 적용하여 최적화수행</li></ul>
조인순서 조정을 위한 힌트	ORDERED	<ul style="list-style-type: none"><li>FROM절에 기술한 순서대로 조인을 하도록 유도(LEADING 힌트와 함께 쓰면 LEADING 힌트는 무시)</li></ul>
	LEADING	<ul style="list-style-type: none"><li>FROM절의 테이블 순서와 상관없이 힌트에 나열된 조인순서를 제어</li></ul>
조인방법 선택용 힌트	USE_NL / NO_USE_NL	<ul style="list-style-type: none"><li>NESTED LOOP 조인을 유도하는 힌트 / NESTED LOOP 조인을 제외한 방식으로 유도</li></ul>
	USE_NL_WITH_INDEX	<ul style="list-style-type: none"><li>NESTED LOOP 조인에서 외측루프의 처리주관 인덱스를 지정할때 사용</li></ul>
	USE_HASH/NO_USE_HASH	<ul style="list-style-type: none"><li>해쉬조인 방식으로 수행 되도록 유도. / 해쉬조이 방식을 제외한 다른 조인방식으로 유도</li></ul>
병렬처리 관련힌트	PARALLEL / NOPARALLEL	<ul style="list-style-type: none"><li>엑세스 할때와 DML 처리할때 SQL의 병렬처리를 유도하는 힌트 / 병렬처리 사용하지 않도록 윤</li></ul>
엑세스수단 선택을 위한 힌트	FULL	<ul style="list-style-type: none"><li>힌트내에 정의된 테이블을 풀스캔 방식으로 유도</li></ul>
	INDEX	<ul style="list-style-type: none"><li>인덱스 범위 스캔에 의한 테이블 엑세스를 유도</li></ul>

# 관계 데이터 모델



# Key

유일, 최소, 대표, 불변, 존재

슈퍼, 후보, 기본, 대체

## 1. 투플(tuple)을 유일하게 식별할 수 있는 애트리뷰트(속성)의 집합, 데이터베이스 키의 개요

- 여러 개의 집합체를 담고 있는 하나의 엔티티타입에서 각각의 엔티티를 구분할 수 있는 결정자

## 2. Key의 특성

특성	내용	사례
<b>유일성</b>	• Key 값으로 Relation내에 Tuple들을 구분 가능(Unique / Not Null)	• 주민등록번호 주식별자가 모든 사람을 고유하게 식별할 수 있음
<b>최소성</b>	• 유일성을 지니는 최소한의 속성만을 포함	• 이미 주민번호하나만으로 유일성을 확보할 수 있는데 구분코드+주민번호 PK구성은 최소성을 위배함
<b>대표성</b>	• 해당 Relation을 대표할 수 있는 속성	• 주민등록번호
<b>불변성</b>	• 식별자가 한번 특정 엔티티를 지정되면 그 식별자는 변하지 않아야 함	• 주민등록번호가 변한다는 의미는 이전기록이 말소가 되고 새로운 기록이 발생되는 개념임
<b>존재성</b>	• 식별자가 지정되면 반드시 데이터값이 존재	• 주민등록번호 없는 대한민국 국민은 있을 수 없음

## 3. Key(식별자)의 종류



그림 5-8 키의 관계

- 기본키 선정절차는 결정자 도출 후 슈퍼키(유일성 만족) 선정, 후보키(유일성, 최소성 만족)들 중 대표성을 갖는 기본키를 도출하여 선정

특성	내용
<b>슈퍼키(super key)</b>	• 유일성을 만족하는 속성 또는 속성들의 집합
<b>후보키(candidate key)</b>	• 유일성과 최소성을 만족하는 속성 또는 속성들의 집합
<b>기본키(primary key)</b>	• 후보키 중에서 기본적으로 사용하기 위해 선택한 키
<b>대체키(alternate key)</b>	• 기본키로 선택되지 못한 후보키

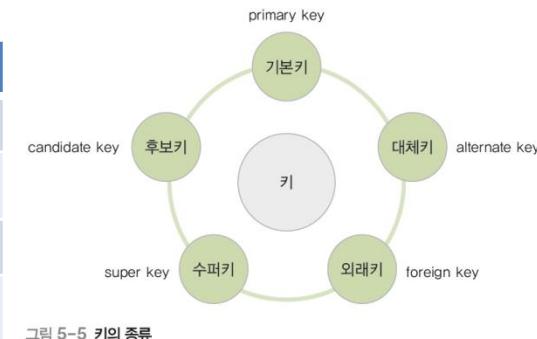
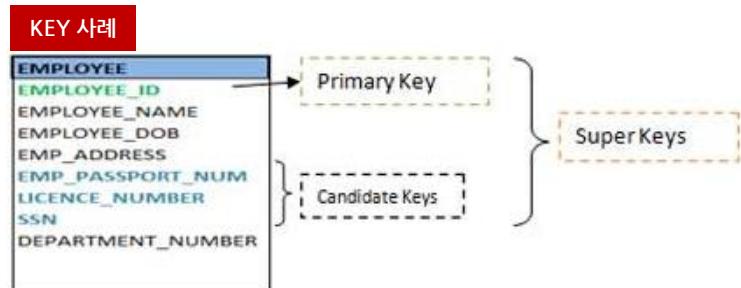


그림 5-5 키의 종류



#### 4. Key의 제약

제약유형	설명	구현형태
<b>본질적 제약</b>	<ul style="list-style-type: none"> <li>데이터 모델의 구조적인 특성으로 인한 제약</li> <li>반드시 주 키(primary key)가 있어야 하고 테이블의 각 셀이 단일 값을 가짐 (1차 정규화의 의미)</li> </ul>	Primary Key, Unique Key
<b>내재적 제약</b>	<ul style="list-style-type: none"> <li>데이터의 의미를 정확히 표현하고 오류를 방지</li> <li>데이터베이스의 스키마에 지정하는 제약</li> <li>영역 제약, 참조 무결성 제약</li> </ul>	Foreign Key, Check, Default ,Not null
<b>명시적 제약</b>	<ul style="list-style-type: none"> <li>프로그램에 명시하거나 사용자의 수작업으로 생성.</li> </ul>	Programmatically

#### 5. 본질적 제약을 위한 Key의 특성 및 종류

제약유형	설명	구현형태
키의 특성	유일성(Uniqueness)	기본키로서 각 튜플을 유일하게 구별
	최소성(minimality)	최소한의 속성을 기본키로 사용하여 유일성 보장
키의 종류	후보키(candidate key)	항상 키의 특성인 유일성과 최소성 모두를 만족하는 키
	기본키(primary key)	후보키 중에서 하나를 대표로 선정하여 사용
	대체키(Alternate key)	기본키 외의 후보키들을 기본키의 대체키로 정의

#### 6. 내재적 제약의 제약 유형 및 형태

제약유형	설명	구현형태
참조 무결성 제약	<ul style="list-style-type: none"> <li>두 관계의 Tuple 간의 일관성을 유지하기 위하여 명시하는 제약,</li> <li>외래키(Foreign Key) 또는 트리거 사용</li> </ul>	Foreign Key (삭제 규칙: restrict, cascade, nullify, default)
범위(Domain) 제약	<ul style="list-style-type: none"> <li>컬럼(속성)형식의 제약.</li> <li>잘못된 입력방지</li> </ul>	체크(Check), 디폴트(Default), 널값미허용(NOT NULL), 룰(Rule)

# Primary Key

## 1. 데이터의 개체 무결성을 보장하기 위한 PK(Primary Key)의 개념

- 여러 개의 집합체를 담고 있는 하나의 엔티티 타입에서 각각의 엔티티를 구분할 수 있는 결정자
- 특징 : 유일성(UNIQUE), Not Null, 최소성

Primary Key

## 2. Primary Key 사용 시의 장점

항목	세부설명
참조무결성 오류방지	<ul style="list-style-type: none"> <li>데이터 베이스에 대한 PK/FK관계를 설정 가능.</li> <li>참조 무결성을 깨뜨리는 무결성 오류 방지.</li> </ul>
식별자역할	<ul style="list-style-type: none"> <li>유일성 보장 (+NULL 허용불가).</li> </ul>
역공학 용이	<ul style="list-style-type: none"> <li>테이블에 한개의 PK만 허용, DB 역공학 시 데이터 모델 생성 및 구분용이</li> </ul>
옵티마이져 실행계획수립	<ul style="list-style-type: none"> <li>Primary Key의 유일한 식별자 역할을 하는 PK는 옵티마이져 실행계획수립 시 활용</li> </ul>

## 3. Primary Key 설정 방법

### Table 생성시 설정

```
CREATE TABLE table_name (
    id_col INT PRIMARY KEY,
    col2 CHARACTER VARYING(20),
    ...
)
```

### Table 정보 수정 시 설정

```
ALTER TABLE <table identifier>
ADD [ CONSTRAINT <constraint identifier> ]
PRIMARY KEY (<column expression>, <column expression>... )
```

Table : Employee	
Employee_ID	Employee_Name
1	Jhon
2	Alex
3	James
4	Roy
5	Kay

# Primary Key

## 4. Primary key와 Unique Index의 특성비교

항목	Primary Key	Unique Index
개념	논리적 개념(식별자)	물리적 개념
물리적구현	PK Constraint	Create Index
공통점	유일성 보장	유일성 보장
참조 무결성	PK/FK에 의해 지정 가능	지정 불가능
테이블당 개수	1개만 가능	여러 개 가능
인덱스 생성	Unique Index 생성	Unique Index 생성
역공학 적용시	PK 인식 가능	PK 인식 불가능
Null 허용	허용 안됨	허용됨

## [참고] 기본키(Primary Key)와 외래키(Foregin Key)



Item	Primary Key	Foreign Key
Consist of One or More Columns	Yes	Yes
Duplicate Values Allowed	No	Yes
Null Values Allowed	No	Yes
Uniquely Identify Rows In a Table	Yes	Maybe
Number allowed per table	One	One or More
Indexed	Automatically Indexed	No Index Automatically created

= primary key

= foreign key

- 참고 : <https://www.essentialsql.com/what-is-the-difference-between-a-primary-key-and-a-foreign-key/>



# Feigen Key

참조무결성

## 1. 참조무결성을 보장하는 외래키(foreign key)의 개요

- 다른 릴레이션의 기본키를 참조하는 속성 또는 속성들의 집합
- 관계형 DBMS에서 외래키는 한 테이블 내의 필드 또는 필드의 결합으로서 반드시 다른 테이블의 주키와 대응되거나 또는 널값을 가지는 역할을 하는 키

## 2. 외래키의 사용목적

Table : Employee

Employee_ID	Employee_Name
1	Jhon
2	Alex
3	James
4	Roy
5	Kay

Table : Salary

Employee_ID	Ref	Year	Month	Salary
1	2012	April		30000
1	2012	May		31000
1	2012	June		32000
2	2012	April		40000
2	2012	May		41000
2	2012	June		42000

## 3. 외래키의 제약조건

- 두 테이블간의 관계를 선언함으로써 **데이터의 무결성을 보장해주는 역할**
- 외래키 관계를 설정하게 되면 하나의 테이블이 또 다른 테이블에 의존하는 형태가 됨.
- 외래키 테이블에 데이터를 입력할 경우 기준테이블을 참조해서 데이터가 입력되므로 기준테이블에 이미 데이터가 존재해야 값이 반영됨
- 외래키 조하는 기준테이블의 열은 반드시 **Primary key, Unique 제약조건이 설정되어 있어야 외래키 제약조건을 설정할 수 있음**

# Feigen Key

## 4. Feigen Key 설정방법

Table 생성시 설정

```
CREATE TABLE table_name (
    id INTEGER PRIMARY KEY,
    col2 CHARACTER VARYING(20),
    col3 INTEGER,
    ...
    CONSTRAINT col3_fk FOREIGN KEY(col3)
        REFERENCES other_table ( UNIQUE(key_col) ON DELETE CASCADE,
        ... )
```

Table 정보 수정 시  
설정

```
ALTER TABLE <table identifier>
ADD [ CONSTRAINT <constraint identifier> ]
    FOREIGN KEY ( <column expression> {, <column expression>}... )
        REFERENCES <table identifier> [ ( <column expression> {, <column expression>}... ) ]
        [ ON UPDATE <referential action> ]
        [ ON DELETE <referential action> ]
```

## 5. 참조무결성 제약조건을 만족시키기위해 DBMS 가 제공하는 옵션

구분	설명	예시
<u>제한(restrict)</u>	위배를 야기한 연산을 단순히 거절	-참조된 외부키를 가진 ROW 삭제 시 참조무결성 제약조건에 위배되어 삭제연산을 거절
<u>연쇄 (Cascade)</u>	참조되는 릴레이션에서 튜플을 삭제하고 참조하는 릴레이션에서 이 튜플을 참조하는 튜플을 함께 삭제	-참조된 외부키를 가진 ROW 삭제 시 참조하는 관련 테이블의 ROW도 동시에 삭제
<u>널값(null)</u>	참조되는 릴레이션에서 튜플을 삭제하고 참조하는 릴레이션에서 이 튜플을 참조하는 널값을 삽입	-참조된 외부키를 가진 ROW 삭제 시 참조하는 관련 테이블의 ROW의 외부키 값도 Null로 설정
<u>디폴트값(default)</u>	널값을 넣는 대신 디폴트값을 넣는다는 것을 제외하고 널값의 옵션과 같음	널값의 옵션 예에 널값대신 디폴트값 삽입

# 채번

테이블, Max+1, 시퀀스 Obj

## 1. 데이터베이스의 순차적 효율성을 관리하는 채번 방식

- 식별자인 Primary Key를 사용하는 방식보다 **순차방식으로 번호를 증대시키는 일련번호 형식의 식별자를 사용**하는 방식

## 2. 채번 방식

방법	처리방법	장점	단점
① 채번 테이블	채번 테이블 이용	<ul style="list-style-type: none"><li>DUP 에러 없음</li><li>일련번호 체계 가능</li></ul>	<ul style="list-style-type: none"><li>LOCK 유발</li><li>성능 저하</li><li>관리 항목 증가</li></ul>
② 테이블에 최대값 적용	INSERT INTO 테이블(일련번호, COL2, COL3 ...) SELECT DECODE(MAX(번호), NULL, 1, MAX(번호)+1) 일련번호 FROM 트랜잭션 WHERE ROWNUM=1;	<ul style="list-style-type: none"><li>관리 항목 증가 없음</li><li>빠른 성능</li><li>일련번호 체계 가능</li></ul>	<ul style="list-style-type: none"><li>이론적 DUP 에러 가능</li></ul>
③ 시퀀스 오브젝트 이용	create sequence 테이블_seq increment by 1 start with 1 maxvalue 1000000000 cache 30;	<ul style="list-style-type: none"><li>빠른 성능</li><li>DUP 에러 없음</li><li>LOCK 없음</li></ul>	<ul style="list-style-type: none"><li>관리 항목 증가</li><li>일련번호 체계 불가능</li></ul>

# 인덱스(Index)

인덱스 내부 메커니즘 : B-Tree

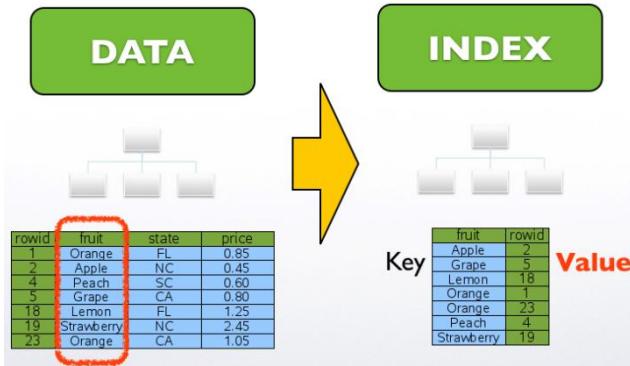
조회성능 향상

## 1. 데이터 검색속도 향상을 위한 데이터베이스 Object, 인덱스의 개요.

- 어떤 파일의 레코드들에 대한 효율적 접근을 위해 <레코드 키 값, 레코드 주소(포인터)> 쌍을 체계적으로 수집하여 관리하는 보조 데이터 구조
- 검색 연산을 최적화하기 위해 탐색 키 값과 해당 데이터의 위치 정보(Pointer)로 구성되어 있는 데이터 베이스 Object.

## 2. 인덱스의 특징

특징	내용
성능 향상	<ul style="list-style-type: none"> <li>데이터베이스 테이블에 접근하는 트랜잭션의 성능 향상이 목적이임</li> <li><b>조회(검색) 성능향상이 주목적, 입력/수정/삭제성능은 더 저하됨</b></li> </ul>
독립성	<ul style="list-style-type: none"> <li>테이블에 저장구조와 별도로 인덱스만 저장할 수 있음</li> </ul>
알고리즘	<ul style="list-style-type: none"> <li>Tree 구조, Hash 함수 등 적용, 알고리즘을 적용하여 생성</li> </ul>
Trade-Off	<ul style="list-style-type: none"> <li>조회 vs 입력/수정/삭제 <b>성능을 고려하여 인덱스 생성</b> 요구</li> </ul>



## 3. 인덱스 장/단점

장점	단점
<ul style="list-style-type: none"> <li>조회 성능향상효과</li> <li>일반적으로 테이블의 일부부만 활용하는 쿼리가 대부분이기 때문에 FTS(Full Table Scan)에 비해 물리적인 디스크I/O가 적음.</li> <li>INDEX를 이용하여 별도의 정렬없이 인덱스 순서대로 결과 추출 가능함.</li> </ul>	<ul style="list-style-type: none"> <li>입력/삭제/갱신 시에 성능저하</li> <li>부정형의 질의문에는 사용되지 않음.(!=, notexist)</li> <li>NULL 정보를 저장하지 않음</li> <li>비트맵 인덱스에 비하면 저장 공간을 많이 차지함.</li> <li>빈번한 DML이 일어나면 트리의 밸런스가 부분적으로 저하되는 단점</li> </ul>

# 인덱스(Index)

인덱스 유형

정적/동적 인덱스

### 3. 구조측면에서의 인덱스의 유형

기준	인덱스 구조	설명
형태	트리 기반 인덱스	<ul style="list-style-type: none"> <li>인덱스(RDBMS는 대부분 B-tree)</li> </ul>
	해쉬 기반 인덱스	<ul style="list-style-type: none"> <li>Hash 테이블을 이용하여 데이터 검색</li> <li>=, &lt;=, =&gt; 연산자만 사용 가능</li> </ul>
	비트맵 인덱스	<ul style="list-style-type: none"> <li>컴퓨터에서 사용하는 최소 단위인 비트를 이용하여 컬럼값을 저장하고 이를 이용하여 ROWID를 자동으로 생성하는 인덱스</li> </ul>
목적	함수기반 인덱스	<ul style="list-style-type: none"> <li>사용자 정의 함수 결과를 인덱스로 사용</li> <li>데이터 타입이 상이한 컬럼 간 사용</li> </ul>
	조인 인덱스	<ul style="list-style-type: none"> <li>DW에서 조인 쿼리의 처리가 가능</li> </ul>
	도메인 인덱스	<ul style="list-style-type: none"> <li>사용자 정의의 인덱스 타입을 사용</li> <li>텍스트, 카테고리 인덱스 등</li> </ul>
구조	정적 인덱스	<ul style="list-style-type: none"> <li>데이터 파일에 레코드가 삽입되거나 삭제됨에 따라 인덱스의 내용은 변하지만 인덱스 구조 자체는 변경되지 않게 하는 인덱싱 기법</li> </ul>
	동적 인덱스	<ul style="list-style-type: none"> <li>인덱스와 데이터 파일을 블록으로 구성하고, 각 블록에는 나중에 레코드가 삽입될 것을 감안하여 빈 공간을 미리 준비해두는 인덱싱 기법</li> </ul>

### 4. 활용목적 관점의 인덱스의 유형

구분	종류	주요내용
논리적 인덱스	Clustered Index	<ul style="list-style-type: none"> <li>데이터의 레코드 순서가 인덱스 페이지의 정렬 순서와 동일하거나 비슷하게 만들어진 인덱스.</li> </ul>
	Non-Clustered Index	<ul style="list-style-type: none"> <li>데이터 레코드의 물리적 순서가 인덱스의 엔트리 순서와 상관 없이 저장되도록 구성된 인덱스 (일반적으로 칼럼 1개 인덱싱)</li> </ul>
	Dense Index	<ul style="list-style-type: none"> <li>데이터 레코드 하나에 대해 하나의 인덱스 엔트리가 만들어지는 인덱스</li> </ul>
	Sparse Index	<ul style="list-style-type: none"> <li>데이터 파일의 레코드 그룹, 또는 데이터 블록에 하나의 엔트리가 만들어지는 인덱스</li> </ul>
	Unique Index	<ul style="list-style-type: none"> <li>인덱스 컬럼에 대해 중복된 값을 허용하지 않음</li> <li>테이블에서 기본키와 고유키 제약조건을 정의하면 인덱스가 묵시적으로 생성</li> </ul>
	Non-Unique Index	<ul style="list-style-type: none"> <li>인덱스 컬럼에 대해 중복된 값을 허용</li> <li>인덱스 키는 연관된 테이블의 여러 행을 가리킬 수 있음</li> </ul>
	Single Index	<ul style="list-style-type: none"> <li>하나의 컬럼으로 구성된 인덱스</li> </ul>
	Composite Index	<ul style="list-style-type: none"> <li>여러 개의 컬럼 조합으로 생성된 인덱스</li> <li>조합 인덱스는 최대 32개의 컬럼까지 구성 가능</li> </ul>
	Function-based index	<ul style="list-style-type: none"> <li>함수나 표현식의 계산 값으로 인덱스 생성 (Cost-based에서만 사용 가능)</li> <li>함수의 리턴 값은 DETERMINISTICS로 선언되어야 함</li> </ul>
물리적 인덱스	Partition Index(분할 인덱스)	<ul style="list-style-type: none"> <li>큰 테이블의 인덱스 엔트리를 저장하는데 사용</li> </ul>
	Bitmap Index(비트맵 인덱스)	<ul style="list-style-type: none"> <li>컴퓨터에서 사용하는 최소단위인 비트를 이용하여 컬럼값을 저장하고 이를 이용하여 ROWID를 자동으로 생성하는 인덱스의 한 방법</li> </ul>
	Reverse Index	<ul style="list-style-type: none"> <li>키 값을 뒤에서부터 역순으로 뒤집은 B*Tree 인덱스 (증가하는 값에 대한 인덱스를 고르게 분산하기 위한 인덱스)</li> </ul>
	Descending Index	<ul style="list-style-type: none"> <li>큰 값부터 작은 값으로 내림차순으로 정렬된 데이터를 위한 인덱스</li> </ul>

# 인덱스(Index)

## 5. 인덱스 선정 절차

절차	설명
액세스 유형	- 해당 테이블의 <b>액세스 유형 조사</b> (접근 프로그램, 주요 쿼리 집중 조사)
<b>분포도 분석</b>	- 대상 컬럼의 선정 및 <b>분포도 분석</b> (적정 분포도 : 10~15%)
액세스 경로	- 반복적으로 수행되는 최적의 액세스 경로를 탈수 있게 최적화
클러스터링 검토	- 물리적으로 <b>연속된 순서에 배치되어야 하는지 검토</b>
조합/순서 결정	- 인덱스 컬럼의 조합 및 순서의 결정
준비	- 시험 생성 및 테스트
수정	- 수정이 필요한 애플리케이션의 전수 조사 및 수정
일괄 적용	- 수정된 데이터베이스와 어플리케이션의 일괄 적용

## 6. 인덱스 선정 기준

- 분포도가 좋은 컬럼은 단독으로 생성하여 활용도 향상
- 자주 조합되어 사용하는 경우 결합 인덱스 생성
- 각종 액세스 경우의 수를 만족하도록 인덱스간 역할 분담
- 가능한 수정이 빈번하지 않은 컬럼
- 기본 키 및 외부키 (조인의 연결고리가 되는 컬럼)
- 결합 인덱스의 컬럼 순서 선정에 주의

## 7. 인덱스 선정 지침

구분	선택 지침	세부 내용
시스템 측면	<b>테이블 크기</b>	<ul style="list-style-type: none"> <li>일정크기 이상의 테이블에 인덱스 생성(6블럭이상)</li> </ul>
	<b>카디널리티</b>	<ul style="list-style-type: none"> <li>인덱스컬럼의 카디널리티가 3000건 미만인 경우</li> <li>무조건 분포도가 10% 이내인 경우 적용 지양</li> </ul>
	<b>기본키/ 외래키</b>	<ul style="list-style-type: none"> <li>조인의 연결고리가 되므로 인덱스 생성</li> </ul>
	<b>비트맵 인덱스</b>	<ul style="list-style-type: none"> <li>분포도가 나쁘면서 자주 사용된다면 BITMAP 인덱스 고려</li> </ul>
운영 측면	<b>조회전용</b>	<ul style="list-style-type: none"> <li>가능한 수정이 빈번하지 않은 컬럼</li> </ul>
	<b>ACCESS PATH</b>	<ul style="list-style-type: none"> <li>조사된 액세스 유형을 토대로 선정기준 마련</li> </ul>
	<b>중복성 회피</b>	<ul style="list-style-type: none"> <li>하나의 컬럼이 여러 인덱스에 포함되지 않게 적용</li> </ul>
	<b>부분 범위처리</b>	<ul style="list-style-type: none"> <li>부분 범위 처리 전용일 경우 분포도가 나쁘더라도 선정</li> </ul>

# 인덱스(Index)

## 8. 인덱스 사용 시 주의사항

주의사항	설명
인덱스 조건을 가공 금지	TRIM, LPAD와 같은 함수로 가공하면 인덱스 사용 불가
정의된 인덱스 컬럼의 WHERE 사용	WHERE에서 비교가 발생해야 인덱스의 정확한 사용 가능
통계정보의 생성 및 유지보수	CBO에서 정확한 인덱스 선택을 위해 통계정보를 최신화 하여야 함
NOT을 사용하지 않음	NOT을 사용하게 되면 인덱스 사용 불가
BETWEEN, IN의 사용	<, >을 사용하기보다는 BETWEEN, IN(A, B, …) 을 사용하여 인덱스 사용
인덱스의 개수는 4~5개가 적당	일반적으로 하나의 테이블에 4~5개가 적정, 데이터 변경이 없는 읽기 전용 테이블은 더 많은 인덱스 사용 가능

## 9. 인덱스 생성을 위한 DDL

[Table 생성시]

```
CREATE TABLE test
(
    Column
    INDEX <Index name> ( column 1, column 2 )
    OR
    UNIQUE INDEX <Index name> ( column )
```

[기 생성된 Table에 생성]

```
CREATE INDEX <Index name>
ON <Table name> ( column 1, column 2, ... );
```

또는

```
ALTER TABLE <Table name>
ADD INDEX <Index name> ( column 1, ... );
```

[인덱스 정의서]

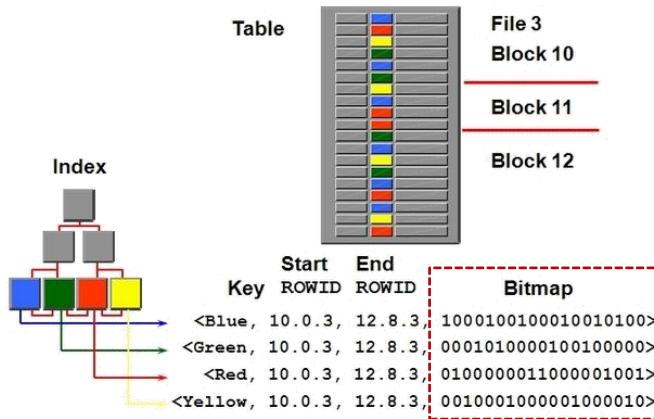
엔티티명	테이블명	인덱스명	컬럼명	타입	인덱스 스페이스	인덱스 유형	정렬	구분
부서	DEPT	I_DEPT01	DEPTNO	NUMBER(2)	ISTEST01	UNIQUE	ASC	PK INDEX
사원	EMP	I_EMP01	EMPNO	VARCHAR2(6)	ISTEST01	UNIQUE	ASC	PK INDEX
		I_EMP02	EMPNO	VARCHAR2(6)	ISTEST01	NON UNIQUE	DESC	INDEX
			HIREDATE	VARCHAR2(8)				

# Bitmap Index

## 1. 비트를 이용한 인덱스, 비트맵 인덱스의 개요

- 컴퓨터에서 사용하는 최소 단위인 **비트를 이용하여 컬럼 값을 저장**하고 이를 이용하여 ROWID를 자동으로 생성하는 인덱스
- 비트를 직접 관리하기 때문에 저장공간이 크게 감소하고 비트 별로 각종 연산을 수행함으로써 기존의 인덱스가 해결할 수 없었던 많은 문제를 해결 (**분포도가 낮은 컬럼에 적용할 때 효과적**)

## 2. 비트맵 인덱스의 구조와 구성요소



구분	설명
Entry Header	- 컬럼 수와 Lock 정보 저장
Key Value	- 각 키 컬럼의 길이와 값의 쌍을 저장 (인덱스 컬럼의 Unique한 값)
Start ROWID	- 시작 RowID 지정
End ROWID	- 종료 RowID 지정
Bitmap	- Bit문자(0,1)로 2차원 배열 구성, 해당하는 키 값일 시 1, 아니면 0

## 3. Bitmap Index와 B-Tree Index의 비교

구분	B-Tree 인덱스	비트맵 인덱스								
구조특징	- Root Block, Branched Block, Leaf Block으로 구성되어 Branched Block의 균형을 유지하는 트리 구조	- 전체 로우의 인덱스 컬럼 값을 0과 1을 이용하여 저장								
인덱스 노드 구조	<table border="1"> <tr> <td>Entry Header</td> <td>Key 값의 쌍</td> <td>ROWID</td> </tr> </table>	Entry Header	Key 값의 쌍	ROWID	<table border="1"> <tr> <td>Entry Header</td> <td>Key 값의 쌍</td> <td>시작 ROWID</td> <td>종료 ROWID</td> <td>Bitmap Segment</td> </tr> </table>	Entry Header	Key 값의 쌍	시작 ROWID	종료 ROWID	Bitmap Segment
Entry Header	Key 값의 쌍	ROWID								
Entry Header	Key 값의 쌍	시작 ROWID	종료 ROWID	Bitmap Segment						
사용 환경	- OLTP	- DW, Data Mart 등								
검색 속도	- 소량의 데이터를 검색하는데 유리	- 대량의 데이터를 읽을 때 유리								
분포도	- 데이터분포도가 높은 컬럼에 유리	- 데이터 분포도가 낮은 컬럼에 유리								
장점	- 입력, 수정, 삭제가 용이함	- 비트 연산으로 OR 연산, NULL값 비교 등에 가능함								
단점	- 스캔 범위가 넓을 때 랜덤 I/O 발생	- 전체 인덱스 조정의 부하로 입력, 수정, 삭제가 어려움								
생성문구	= CREATE INDEX [인덱스명]ON 테이블명(컬럼명)	- CREATE BITMAP INDEX [인덱스명] ON 테이블명(컬럼명)								

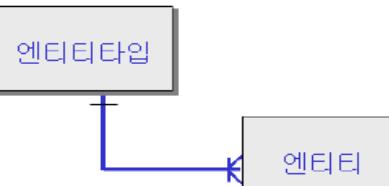
# [데이터 모델링] 엔티티(Entity)

## 1. 엔티티 타입의 개요

- 업무에 필요한 정보를 저장하고 관리하기 위한 것으로 영속적으로 존재하는 단위.
- Entity Type: 개체들을 동일한 유형별로 분류한 단위 (= Entity 집합)

## 2. 엔티티 타입과 표현

엔티티타입-엔티티 E/R



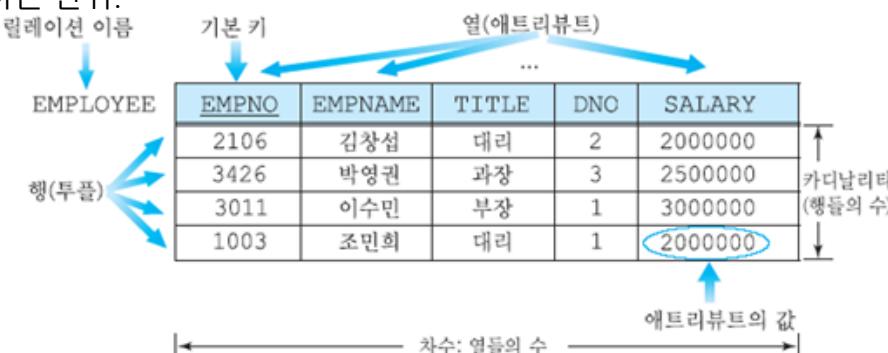
엔티티 타입은 엔티티들의 집합

엔티티타입-엔티티의 예

엔티티 타입	엔티티
강의실	101호
	102호
과목	J2EE
	DATA MODELING
강사	이강사
	김강사

## 3. Entity 유형

분류구분	종류	설명
유/무형	유형 (Tangible)	- 물리적인 형태가 있고 안정적이며 지속적으로 활용되는 엔티티타입 - 업무에서 엔티티타입을 구분하기가 가장 용이함 예) 사원, 물품, 강사 등
	개념 (Conceptual)	- 물리적인 형태는 존재하지 않고 관리해야 할 개념적 정보 예) 조직, 상품, 장소 등
	사건 (Event)	- 업무수행 중 발생하는 엔티티타입 - 비교적 발생 양이 많으며 각종 통계자료에 이용 가능 예) 주문, 청구, 미납, 계좌 등
발생시점	기본 (Fundamental)	- 업무에 원래 존재하는 정보 / - 독립적으로 생성 - 다른 엔티티타입의 부모역할을 함
	중심 (Main)	- 기본 엔티티타입에서 발생 - 데이터 양이 많고 다른 엔티티타입과의 관계를 통해 많은 행위 엔티티타입을 생성함
	행위 (Active)	- 두 개 이상의 부모 엔티티타입에서 발생 - 내용이 자주 바뀌거나 데이터 양이 증가 - 분석초기단계에는 잘 나타나지 않으며 상세설계단계나 프로세스와 상관 모델링을 진행하면서 도출 될 수 있음



구성요소	설명
속성	• 각 개체만의 고유한 특성이나 상태
개체타입	• 개체를 고유의 이름과 속성들로 정의한 것
인스턴스	• 속성이 실제 값을 가짐으로써 실체화된 개체
개체 집합	• 개체 타입에 대한 개체 인스턴스들을 모아놓은 것을 개체 집합

# 데이터모델링의 관계(Relationship)

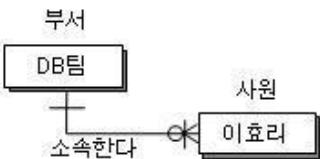
## 1. 두 개의 엔티티타입 사이의 논리적인 연결, 관계의 개요

- 엔티티와 엔티티가 존재의 형태나 행위로서 서로에게 영향을 주며 **논리적으로 관련성이 도출되는 상태**이며 보다 가시적인 업무의 흐름분석 및 데이터 무결성 보장을 위한 관계
- 두 개의 엔티티 타입 사이의 논리적인 관계, 즉 엔티티와 엔티티가 존재하는 형태로써 혹은 행위로써 서로에게 영향을 주는 상태
- 엔터티 간 존재의 형태나 행위로서 서로에게 영향을 주는 것
- 목적 : 업무흐름을 표현, Key의 상속관계, 참조무결성

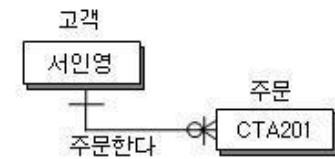
## 2. 관계의 형태

- 존재 : 정보의 흐름이 정적인 상태 (소속한다)
- 행위 : 정보의 흐름이 동적인 상태 (주문한다)

존재에 의한 관계



행위에 의한 관계



기호	의미	설명
	일대일 관계	하나의 개체가 하나의 개체에 대응
	일대다 관계	하나의 개체가 여러 개체에 대응
	다대일 관계	여러 개체가 하나의 개체에 대응
	다대다 관계	여러 개체가 여러 개체에 대응



# 데이터모델링의 관계(Relationship)

정상, 자기참조, 병렬, 슈퍼/서브, 주/비

## 3. 관계의 종류

종류	ERD	설명
정상적인 관계 (Normal Relationship)		<ul style="list-style-type: none"> <li>엔티티타입과 엔티티타입이 독립적으로 분리되어 있으면서 한 개의 관계만 상호간 존재하는 형태의 관계</li> </ul>
자기참조관계 (Self Relationship, Recursive Relationship)	 	<ul style="list-style-type: none"> <li>하나의 엔티티타입내에서 엔티티와 엔티티가 관계를 맺고 있는 형태</li> <li>부서, 부품, 메뉴 등과 같이 계층구조 형태를 표현할 때 유용한 관계형식</li> <li>예) 컴퓨터는 본체, 마우스, 키보드, 모니터로 조립되며, 본체는 다시 메모리, 디스크, CPU 등으로 조립</li> </ul>
병렬관계 (Parallel Relationship)		<ul style="list-style-type: none"> <li>엔티티타입과 엔티티타입이 독립적으로 분리되어 있으면서 두 개 이상의 관계가 상호간 존재하는 형태의 관계</li> </ul>
슈퍼타입 서브타입관계 (Super-type Sub-type Relationship)	<p><b>배타적관계</b> 비슷한 엔티티로부터 수퍼타입이 발생</p> <p><b>포함관계</b> 하나의 엔티티로부터 서브타입이 발생</p>	<ul style="list-style-type: none"> <li>공통의 속성을 가지는 수퍼 타입과 공통부분을 제외하고 두 개 이상의 엔티티 타입에 속성이 상호간의 차이가 있을 때 별도의 서브타입으로 존재할 수 있음.</li> <li>이때 수퍼타입과 서브타입이 1:1 관계를 가지게 되는데 이것이 수퍼타입과 서브 타입의 관계형식임.</li> <li>이때 서브타입을 구분할 수 있는 구분형식에 따라 수퍼타입의 특정 엔티티가 반드시 하나의 서브타입에만 속해야 하는 배타적관계(Exclusive Relationship)와 수퍼타입의 특정 엔티티가 두 개 이상의 서브타입에 포함될 수 있는 포함관계(Inclusive Relationship)로 구분할 수 있음</li> </ul>
주식별자/비식별자 관계 (Identifying/Non-Identifying Relationship)	<p><b>주식별자</b></p> <p><b>비식별자 관계</b></p>	<ul style="list-style-type: none"> <li>부모엔티티타입의 주식별자가 자식 엔티티타입의 주식별자로 상속되는 주식별자 관계와 부모 엔티티타입이 주식별자가 자식 엔티티타입의 일반 속성으로 상속되는 비식별자 관계로 구분</li> </ul>

# [데이터모델링의 관계] 식별자/비식별자 관계

유일성, 최소성, 불변성

## 1. 식별자(identifier)의 개요

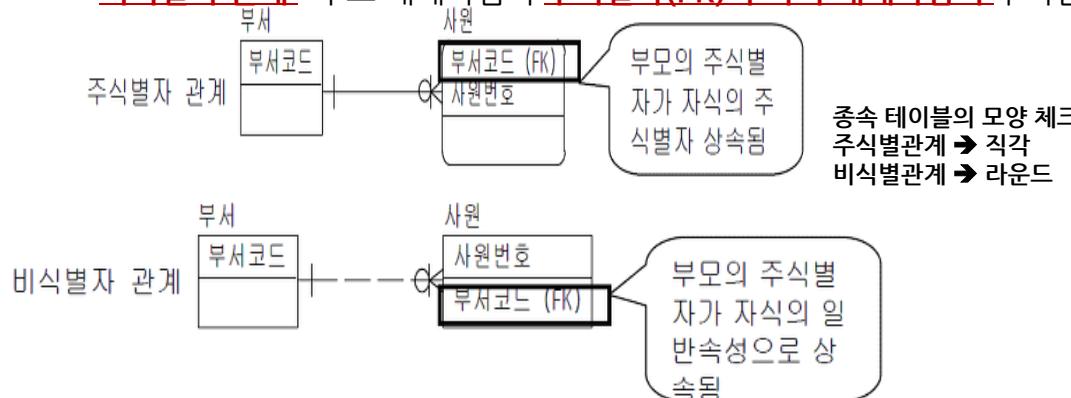
- 하나의 엔터티 내에서 각각의 인스턴스를 유일(unique)하게 구분해 낼 수 있는 속성 또는 속성 그룹

## 2. 식별자의 특징

- 유일성** : 주식별자에 의해 엔터티 내에 모든 인스턴스들을 유일하게 구분함
- 최소성** : 주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 함
- 불변성** : 주식별자가 한번 특정 엔터티에 지정되면 그 식별자에 값은 변하지 않아야 함
- 존재성** : 주식별자가 지정되면 반드시 값이 존재(Null 안됨)

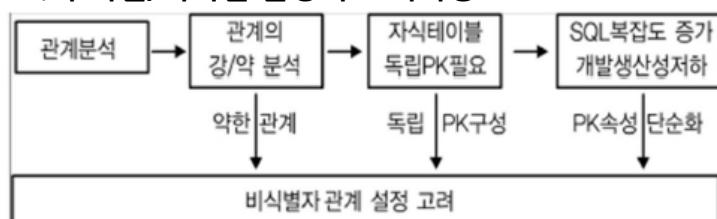
## 3. 식별자의 유형

- 주식별자 관계** : 부모 개체타입의 **주식별자(PK)**가 자식 개체타입의 **주식별자로 상속**되는 관계
- 비식별자 관계** : 부모 개체타입의 **주식별자(PK)**가 자식 개체타입의 주식별자가 아닌 **일반 속성으로 상속**되는 관계



항목	식별자관계	비식별자관계
목적	- 강한 연결관계 표현	- 약한 연결관계 표현
자식주식별자 영향	- 자식 주식별자의 구성에 포함	- 자식 일반 속성에 포함
표기법	- 실선 표현	- 점선 표현
연결고려사항	<ul style="list-style-type: none"><li>- 반드시 부모 엔터티의 증속</li><li>- 자식 주식별자 구성에 부모 주식별자 포함 필요</li><li>- 상속받은 주식별자 속성을 타 엔터티에 이전 필요</li><li>- PK 속성 단순화</li></ul>	<ul style="list-style-type: none"><li>- 약한 증속관계</li><li>- 자식 주식별자 구성을 독립적으로 구성</li><li>- 자식 주식별자 구성에 부모 주식별자 부분 필요</li><li>- 상속받은 주식별자 속성을 타 엔터티에 차단 필요</li><li>- 부모쪽의 관계 참여가 선택관계</li></ul>

## 3. 주식별/비식별 설정 시 고려사항



# [데이터모델링의 관계] 자기참조 관계

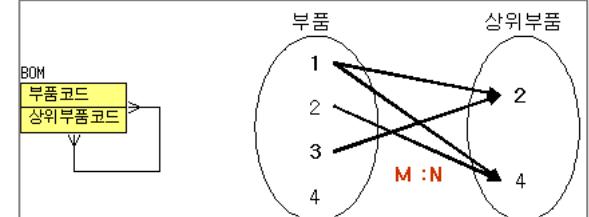
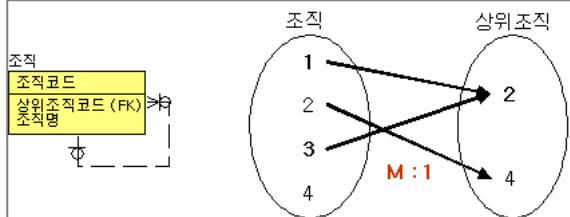
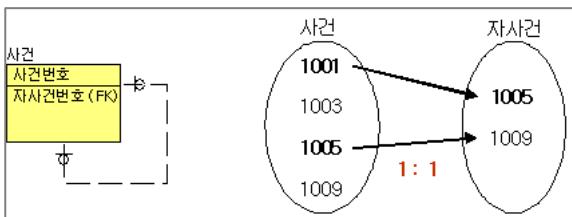
1:1, 1:M, M:N

## 1. 계층구조를 이용한, 자기 참조관계

- 동일한 Entity 상호간에 계층적으로 데이터가 구성되는 관계로 동일 엔티티 타입내에서 **자기가 자기의 인스턴스를 참조**하는 구조

## 2. 자기참조관계의 관계유형에 따른 종류

구분	설명
1:1 자기참조관계	<ul style="list-style-type: none"> <li>사건번호 하나에 자사건번호가 한 개만 올 수 있는 경우</li> </ul>
1:M 자기참조관계	<ul style="list-style-type: none"> <li>상위 조직 하나에 여러개의 하위조직이 올 수 있고 다시 하위 조직 하나는 그 하위 조직여러 개를 거느리는 경우</li> </ul>
M:N 자기참조관계	<ul style="list-style-type: none"> <li>부품 1은 상위부품으로 2도 구성할 수 있고 4도 구성할 수 있음</li> <li>이럴 경우 관계의 표현이 M:N으로 표현되어 모델링 됨</li> </ul>



## 3. M:N 자기참조 모형의 관계해소

### M:N 자기참조관계(Recursive Relationship)의 해소



하나의 엔티티타입내에서 엔티티와 엔티티가 관계를 1:N 맺고 있는 형태의 관계

\* M:N 관계는 모델링으로 표현 불가  
→ 물리적으로 관계를 해소해주어야 함

## 4. 자기참조 관계 쿼리 샘플

```

SELECT level, mntpcode, mntpname, mntpseqn
FROM mrfimntp /* 자기참조테이블 */
START WITH mntpcode = #setmtpcd# /* 출발점 */
CONNECT BY PRIOR mntpcode = mntpupcd /* 연결되는 부분 */
ORDER SIBLINGS BY mntpseqn /* 자기참조내의 정렬기준 */

```

\* ANSI-SQL 구문으로 모두 다 사용 가능

# [데이터모델링의 관계] 슈퍼타입/서브타입

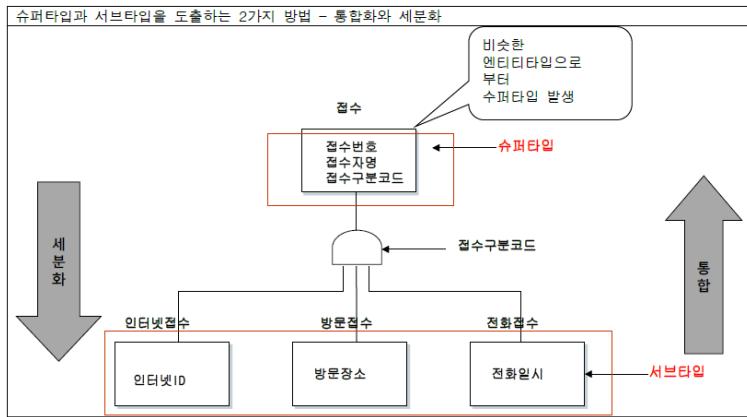
베타. 포함

Identity, Rollup, RollDown

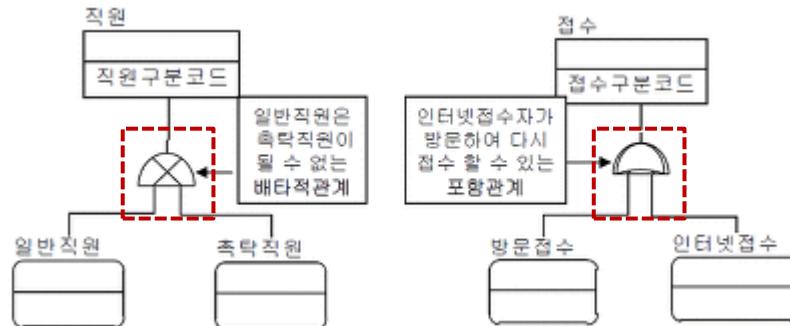
## 1. 데이터 모델의 상속성 표현, 슈퍼타입과 서브타입

- 엔티티 작성 시 대부분의 속성이 비슷하고 일부만 다른 여러 엔티티들을 하나로 묶어 통합하는 경우 수행하는 확장형 데이터모델링  
(객체지향설계의 상속관계와 유사)

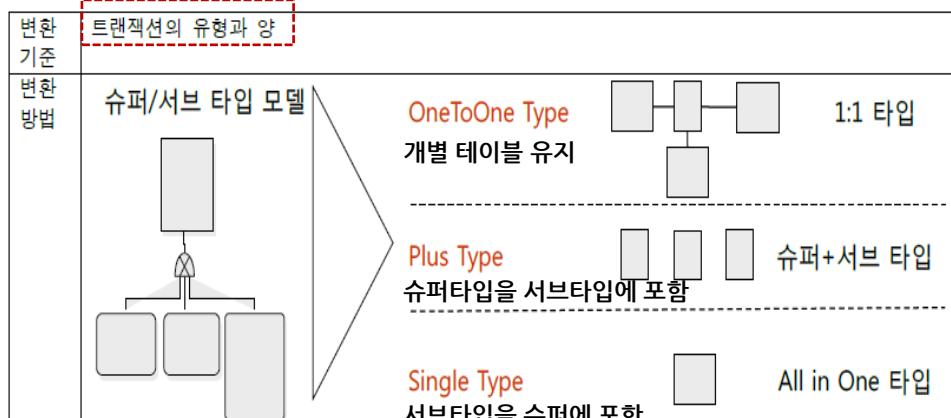
## 2. 슈퍼타입과 서브타입 데이터 모델링 기법



## 3. 슈퍼타입과 서브타입에 대한 데이터 모델 (베타/포함)



## 4. 슈퍼타입과 서브타입의 물리적인 데이터 모델링 기법



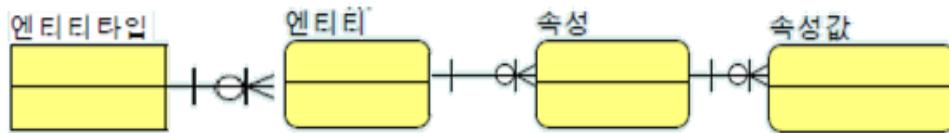
구분	OneToOne Type (Identity)	Plus Type (Rollup)	Single Type (Rollup)
특징	개별 테이블 유지	슈퍼+서브타입 테이블	하나의 테이블
확장성	우수함	보통	나쁨
조인성능	나쁨	나쁨	우수함
I/O량 성능	좋음	좋음	나쁨
관리용이성	좋지 않음	좋지 않음	좋음(1개)
트랜잭션 유형에 따른 성택 방법	개별 테이블로 접근이 많은 경우 선택	슈퍼+서브 형식으로 데이터를 처리하는 경우 선택	전체를 일괄적으로 처리하는 경우 선택

# [데이터 모델링] 속성 (attribute)

## 1. 데이터베이스의 속성의 개요

- 업무에 필요한 엔터티에서 관리하고자 하는 의미상 더 이상 분리되지 않는 최소의 데이터 단위

## 2. 속성과 엔티티 타입간의 관계



## 3. 속성의 유형

- 기본 속성 : 업무 분석을 통해 바로 정의한 속성 / 이름, 주소
- 설계 속성 : 업무상 미존재, 설계하면서 도출해내는 속성 / 코드
- 파생 속성 : 다른 속성으로부터 계산이나 변형을 통해 생성되는 속성 / 급여총액, 평균값

## 4. Entity 구성방식에 따른 분류

- PK속성
- FK속성
- 일반속성

속성(어트리뷰트) = 열 = 필드					
사원번호	이름	직급	주민번호	급여	부서명
101	김사랑	사원	831212-2212112	250	인사부
102	한예슬	대리	771227-2323123	300	영업부
103	오지호	과장	720224-1013112	500	영업부
104	이병현	부장	710509-1934142	600	인사부

속성의 개수 = 차수 = 6

# [데이터 모델링] 도메인

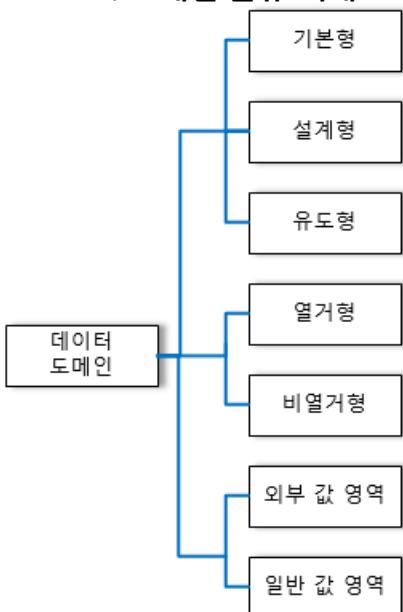
## 1. 개체의 속성이 가질 수 있는 값들의 집합(범위), 도메인의 개요

- 릴레이션에 저장되는 데이터 값들이 본래 의도했던 값들만 저장되고 관리되도록 하기 위해 릴레이션 내의 **속성에 대한 데이터타입과 크기, 제약 사항을 지정**(속성에 정의된 조건을 만족시키는 값의 범위)
- DBMS에서 도메인 : 컬럼에 대한 데이터 타입과 길이를 의미(도메인이 동일하다는 것은 데이터 속성과 길이가 같다는 것을 의미함)
- 속성이 일관된 규칙에 따라 데이터 타입과 크기가 부여 됨으로 모델의 관리가 용이
- 정의하는 작업도 중요하지만 **도메인이 변경, 추가시에 따른 일관성 유지도 필요**

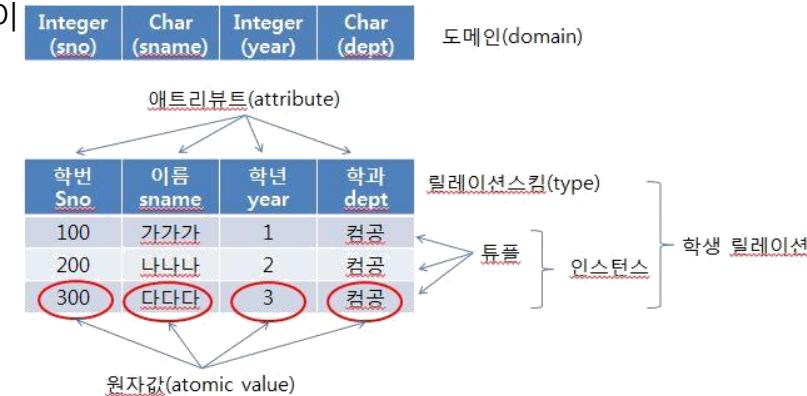
## 2. 도메인의 정의에 대한 예

```
CREATE DOMAIN DEMPNO INTEGER
CREATE DOMAIN DEMPNAME VARCHAR(30)
CREATE DOMAIN DSALARY INTEGER
```

## 3. 도메인 분류 체계



분류	설명
기본형	<ul style="list-style-type: none"> <li>업무로부터 직접 도출된 가장 기본적인 데이터 도메인</li> </ul>
설계형	<ul style="list-style-type: none"> <li>코드 또는 그 외 특정한 체계로 생성되도록 값을 고안할 필요가 있는 데이터 도메인/업무의 유형을 단순화하거나, 유일성을 보장하기 위한 방편으로 활용</li> </ul>
유도형	<ul style="list-style-type: none"> <li>이미 존재하는 다른 도메인의 계산에 의해 생성되는 데이터 도메인으로 계산 공식 등의 파생 알고리즘이 존재</li> </ul>
열거형	<ul style="list-style-type: none"> <li>구성하는 값의 원소를 모두 나열할 수 있는 데이터 도메인</li> <li>예로는 코드 집합, 표준 분류, 분류등의 형태를 포함합니다</li> </ul>
비열거형	<ul style="list-style-type: none"> <li>구성하는 값의 원소를 범위로서 규정하며, 일일이 나열하는 것이 불가능한 데이터 도메인</li> <li>예로는 수치의 간격, 문자열의 특성, 비트맵 등의 형태</li> </ul>
외부 값 영역	<ul style="list-style-type: none"> <li>국내외 타 기관이나 업체에서 정의한 값 영역을 변경하지 않고 그대로 차용한 경우</li> </ul>
일반 값 영역	<ul style="list-style-type: none"> <li>외부 값 영역이 아닌 일반적인 경우</li> </ul>





# [데이터 모델링] 카디널리티(Cardinality, 기수, 대응수)

## 1. 두개의 엔티티 탑간 관계에서 참여자의 수를 표현, 카디널리티의 개요

- 두 개의 엔티티 탑간 사이의 논리적인 관계, 즉 엔티티와 엔티티가 존재하는 형태로써 혹은 행위로써 서로에게 영향을 주는 상태
- 특정 액세스 단계를 거치고 나서 출력될 것으로 예상되는 결과 건수
- 카디널리티 = 총 로우수 \* 선택도 = num\_rows/num\_distinct
- Distinct Value = 10 이면 선택도 = 0.1이고, 총 로우 수가 1,000이라면 카디널리티 100이 됨.

## 2. 데이터베이스 카디널리티 주요유형

유형	관계 구성도	설명
1:1 관계	<p>카디널리티(Cardinality) - 1:1(One To One) 한개의 구매신청서에 대해 한 개의 구매주문을 신청하고 한개의 구매주문에는 한 개의 구매신청내용을 작성한다.</p> <pre>     graph LR       subgraph CustomerOrder [구매신청]         CO[구매신청]         CO -- "신청한다" --&gt; CO         CO -- "작성한다" --&gt; CO       end       subgraph CustomerOrderDetail [구매주문]         COD[구매주문]         COD -- "작성한다" --&gt; COD       end   </pre> <p>각 엔티티의 어느 입장에서 반드시 단 하나씩 관계를 가질 때 성립</p>	- 각 엔티티는 상호간에 하나만의 관계를 형성
1:M 관계	<p>카디널리티(Cardinality) - 1:M(One To Many) 한 명의 사원은 부서에 소속되고 한 부서에는 여러 사원을 포함한다.</p> <pre>     graph LR       subgraph Department [부서]         D[부서]         D -- "포함한다" --&gt; D         D -- "소속된다" --&gt; D       end       subgraph Employee [사원]         E[사원]         E -- "포함한다" --&gt; E         E -- "소속된다" --&gt; E       end   </pre> <p>업무 설계 시, 가장 흔하게 발생하는 관계의 형태로 부모와 자식의 관계처럼 계층적인 구조</p>	- 각 엔티티는 관계를 맺는 다른 엔티티에 대해 하나 또는 다수 관계 형성 (반대방향은 단지 하나만의 관계)
M:N 관계	<p>카디널리티(Cardinality) - M:M(Many To Many) 하나의 주문서에는 여러 개의 제품을 포함할 수 있고 한 제품은 여러 개의 주문서에 의해 신청될 수 있다.</p> <pre>     graph LR       subgraph Order [주문]         O[주문]         O -- "포함한다" --&gt; O         O -- "소속된다" --&gt; O       end       subgraph Product [제품]         P[제품]         P -- "포함한다" --&gt; P         P -- "소속된다" --&gt; P       end   </pre> <p>양쪽 엔티티 모두 1:M 관계가 성립</p>	- 각 엔티티는 관계를 맺는 다른 엔티티에 대해 하나 또는 다수 관계를 가짐 (반대방향도 동일)

# [데이터 모델링] 카디널리티(Cardinality, 기수, 대응수)

## 3. 카디널리티 표기법

카디널리티	최소	최대	표기법
One	1	1	
Many	> 1	> 1	
Zero or one	0	1	
One or many	1	>1	
Zero or many	0	>1	

# 데이터 무결성

## [무결성 제약 조건]

- 데이터 무결성 제약조건 vs 릴레이션 무결성 제약조건
- 물리적 무결성 vs 의미적 무결성

### 1. 데이터의 일관성 보장을 위한, 데이터 무결성 (Data Integrity)의 개요

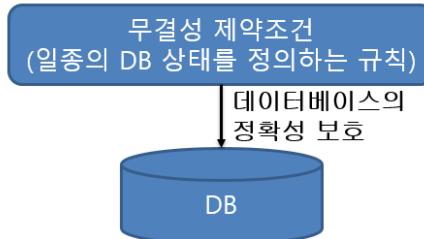
- 사용자의 목적이나 의도와 다른 데이터의 오류방지, 정확성, 유효성, 일관성, 신뢰성을 위해 무효 간섭으로부터 데이터를 보호하는 개념
- 데이터베이스의 일관성을 보장하기 위해 일관된 데이터베이스 상태를 정의하는 규칙들을 목시적 또는 명시적으로 정의하는 조건

### 2. 무결성 제약조건의 유형

유형	내용	명령어
<u>물리적 무결성 제약조건 (physical integrity constraint)</u>	<ul style="list-style-type: none"> <li>데이터 모델 내부에서 정의되고, 주로 데이터의 구조나 이에 적용되는 연산의 물리적 특성을 제약하는 조건</li> </ul>	개체 무결성, 참조 무결성, 도메인 무결성, 사용자 정의 무결성
<u>의미적 무결성 제약 조건 (semantic integrity constraint)</u>	<ul style="list-style-type: none"> <li>데이터베이스 설계자가 외부 스키마를 정의할 때 해당 데이터베이스가 실세계를 정확히 반영하도록 하기 위해서 데이터베이스 안에 의미에 관한 정보를 규정하도록 하는 제약조건</li> </ul>	참조 무결성, 개체 무결성, 관계 무결성

- 데이터베이스가 갱신될 때 DBMS가 자동적으로 일관성 조건을 검사하므로 제약조건을 응용 프로그램에서 나타낼 필요가 없음

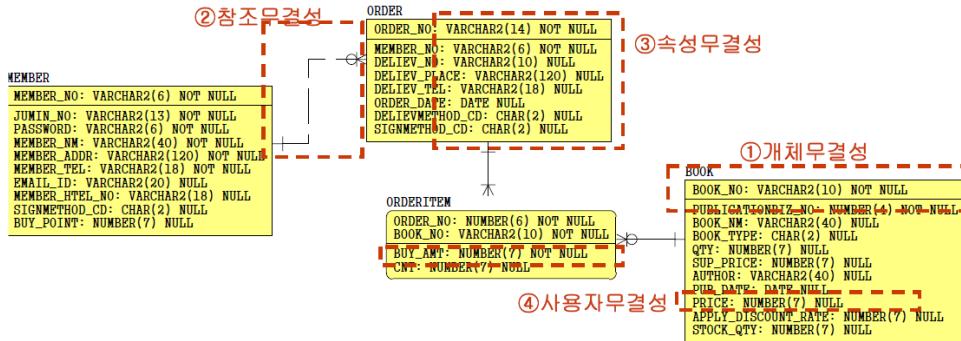
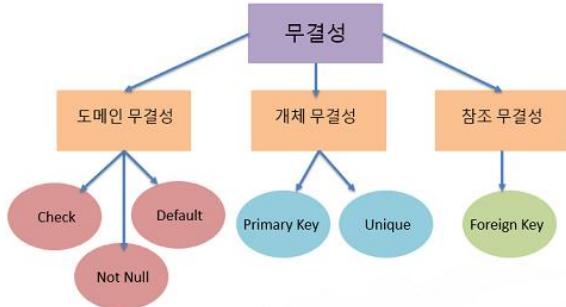
\* 참고자료 : SQL Implementation & Administration(state college)



# 데이터 무결성

개참속사카도

## 3. 데이터 무결성의 개념도 및 종류



종류	내용	명령어
<b>개체(엔티티) 무결성</b>	• 한 엔티티는 <u>중복과 누락이 있을 수가 없음</u> . 즉, 동일한 PK를 가질 수 없거나, PK의 속성이 Null을 허용할 수 없음	Primary Key, Unique Index, PK is not null
<b>참조 무결성</b>	• 외래 키가 <u>참조하는 다른 개체의 기본 키에 해당하는 값이 기본 키 값이나 Null</u> 이어야 함	Foreign Key
<b>속성 무결성</b>	• 속성의 값은 기본값, Null 여부, 도메인 (데이터 타입, 길이)가 <u>지정된 규칙을 준수</u> 하여 존재해야 함(정의된 값만 입력)	Character, Date, LONG, VARCHAR2, NUMBER
<b>사용자(의미) 무결성</b>	• <u>사용자의 의미적 요구사항을 준수</u> 해야함 (업무적 의미 값만 허용)	Trigger, User <u>Defined Data Type</u>
<b>키 무결성</b>	• 한 릴레이션에 같은 키 값을 가진 튜플들이 허용 안됨( <u>중복 없음, 누락 존재</u> )	Unique
<b>도메인 무결성</b>	• 특정 속성 <u>값이 미리 정의된 도메인 범위에 속해야 함</u> (허용된 범위 값 입력)	CHECK, Default

# 데이터 무결성

## 4. 데이터 무결성 유지 방법의 종류

### 가. 선언적 방법(DDL을 이용)

유형	내용
<u>Primary key</u>	<ul style="list-style-type: none"> <li>지정된 컬럼들이 유일성이 위배되는 일이 없음을 보장</li> <li>Primary Key는 Null 값이 될 수 없음</li> </ul>
<u>Unique</u>	<ul style="list-style-type: none"> <li>다중의 보조키 개념을 지원함</li> <li>Primary Key 와 마찬가지로 지정된 컬럼들의 유일성이 위배되지 않음을 보장</li> <li>Unique는 Null을 허용</li> </ul>
<u>Foreign Key</u>	<ul style="list-style-type: none"> <li>테이블간의 논리적 관계가 유지됨을 보장함</li> <li>Foreign Key 값은 반드시 참조하려는 테이블의 Primary Key값으로 나타나야 함</li> <li>Foreign Key 값은 Null 값을 가질 수 있음</li> <li>Cascaded Option : Master가 삭제 시 레코드가 함께 삭제</li> <li>Nullified Option : Master 삭제 시 해당 값을 Null로 셋팅</li> <li>Restricted Option : Foreign Key가 존재하면 Master 레코드를 삭제 할 수 없음</li> </ul>
<u>Data Type</u>	<ul style="list-style-type: none"> <li>데이터의 형을 제한함으로써 데이터 무결성을 유지함</li> </ul>
<u>Check</u>	<ul style="list-style-type: none"> <li>데이터를 추가할 때마다 SQL 서버가 해당 값이 해당 컬럼들에 지정된 Check 제약을 위배하는지를 검사함으로써 데이터 무결성을 유지</li> </ul>
<u>Default</u>	<ul style="list-style-type: none"> <li>특정 컬럼에 대해 명시적으로 값을 입력하지 않은 경우에 SQL 서버가 자동적으로 지정된 값을 삽입할 수 있도록 함으로써 데이터 무결성을 유지</li> <li>INSERT 또는 UPDATE에서 DEFAULT 키워드를 사용할 수 있음</li> </ul>

### 나. 절차적 방법 제약조건 (DML 이용하여 명세)

요소	설명
<u>Trigger</u>	<ul style="list-style-type: none"> <li>테이블의 내용을 변경하려는 특정 사건 (DB 연산)에 대해서 DBMS가 미리 정의된 일련의 행동(DB 연산)들을 수행하는 메커니즘</li> <li>DBMS 서버에 의해 자동적으로 호출됨</li> <li>데이터에 대한 변경을 시도 할 때마다 자동적으로 호출됨 (데이터의 변경 전 상태와 변경 후의 상태를 사용)</li> <li>트랜잭션의 철회 (rollback)와 같은 동작을 수행할 수 있음</li> <li>저장 프로시저의 특별한 형태로서 SQL의 모든 기능을 이용할 수 있음</li> <li>참조 무결성을 위해 사용될 수도 있음. 참조 무결성이 위배되는 경우에 원하는 동작을 하도록 트리거를 구성하면 됨</li> </ul>
<u>Stored procedure</u>	<ul style="list-style-type: none"> <li>SQL과 SPL(Stored Procedure Language)언어를 조합하여 만든 프로시저</li> <li>DB 엔진의 각 인스턴스에 컴파일 된 형태로 저장</li> <li>저장 프로시저를 사용하여 데이터에 대한 무결성 유지</li> </ul>
<u>application</u>	<ul style="list-style-type: none"> <li>비즈니스 로직을 갖고 있는 응용 프로그램 코드에 업무 규칙을 강제로 시행하여 데이터 무결성 확보</li> </ul>

### [데이터 무결성 유지 방법 간 비교]

구분	선언적 데이터 무결성 구현	절차적 데이터 무결성 구현
개념	DBMS 기능으로 무결성 구현	애플리케이션에서 무결성 구현
구현방법	DDL문으로 구현	DML 문으로 구현
무결성 점검	DBMS에 의해 수행됨	프로그래머에 의해 수행됨
장점	절차적 데이터 무결성보다 오류 발생 가능성 낮음	여러 번 반복해서 사용하는 경우 편리성 높음
단점	시스템 성능에 영향을 미침	오류 발생 확률 높음
사례	DDL(CREATE, ALTER), PK, FK, Unique, Check, Data type, default	Trigger, Stored Procedure, application

# 데이터 무결성 - 참조 무결성

## 1. 릴레이션 간에 적용되는 제약조건, 참조무결성의 개요

- 외래키를 통해 릴레이션은 **참조할 수 없는 외래키 값을 가질 수 없도록 함**으로써 두 테이블 간의 데이터 무결성을 유지하는 성질

## 2. 참조무결성의 종류

참조무결성의 종류		
1. 입력 참조무결성	2. 수정 참조무결성	3. 삭제 참조무결성
<ul style="list-style-type: none"><li>의존(DEPENDENT)</li><li>자동(AUTOMATIC)</li><li>기본(DEFAULT)</li><li>지정(CUSTOMIZED)</li><li>NULL</li><li>미 지정</li></ul>	<ul style="list-style-type: none"><li>제한(RESTRICT)</li><li>연쇄(CASCADE)</li></ul>	<ul style="list-style-type: none"><li>제한(RESTRICT)</li><li>연쇄(CASCADE)</li><li>기본(DEFAULT)</li><li>지정(CUSTOMIZED)</li><li>NULL</li><li>미 지정</li></ul>



## 3. 참조무결성 제약조건의 옵션

입력 참조 무결성	설명
DEPENDENT	참조되는 테이블에 PK가 존재할 때만 입력 허용
AUTOMATIC	참조되는 테이블에 PK가 없는 경우 PK를 생성 후 입력
DEFAULT	참조되는 테이블에 PK가 없는 경우 지정된 기본값으로 입력
CUSTOMIZED	특정한 조건이 만족할 때만 입력 허용
NULL	참조되는 테이블에 PK가 없는 경우 외부키를 Null 값으로 처리
No EFFECT	조건 없이 입력을 허용

삭제/수정 참조 무결성	설명
RESTRICT	참조하는 테이블에 PK가 없는 경우 삭제/수정 허용
CASCADE	참조되는 테이블과 참조하는 테이블의 외부키를 연쇄적 삭제/수정
DEFAULT	참조되는 테이블의 수정을 항상 허용하고, 참조하는 테이블의 외부키를 지정된 기본값 바꿈
CUSTOMIZED	특정한 조건이 만족할 때만 수정/삭제 허용
NULL	참조되는 테이블의 수정을 항상 허용하고, 참조하는 테이블의 외부키를 NULL 값으로 수정
No Effect	조건 없이 삭제/수정 허용

# 데이터 무결성 - 도메인 무결성

## 1. 속성에 대한 무결성, 도메인 무결성의 개요

- 특정 속성의 값이, 그 **속성이 정의된 도메인에 속한 값이어야 한다**는 규정

## 2. 도메인의 개념

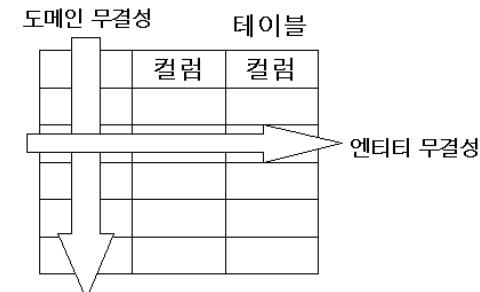
- 하나의 어트리뷰트가 취할 수 있는 같은 타입의 원자(atomic)값들의 집합
- 실제 어트리뷰트 값이 나타날 때 그 값의 합법 여부를 시스템이 검사하는 데에도 이용

## 3. 도메인의 정의에 대한 예

```
CREATE DOMAIN DEMPNO INTEGER
```

```
CREATE DOMAIN DEMPNAME VARCHAR(30)
```

```
CREATE DOMAIN DSALARY INTEGER
```



## 4. 도메인 무결성과 키 무결성의 비교

구분	도메인	키	
	도메인 무결성 제약조건	개체 무결성 제약조건	참조 무결성 제약조건
제약 대상	속성	튜플	속성 & 튜플
같은 용어	도메인 제약	기본키 제약	외래키 제약
해당되는 키	-	기본키	외래키
NULL 허용 여부	허용	불가	허용
릴레이션 내 제약조건의 개수	속성의 개수와 동일	1개	0 ~ 여러 개
기타	튜플 삽입, 수정 시 제약 사항 우선 확인	튜플 삽입, 수정 시 제약 사항 우선 확인	<ul style="list-style-type: none"> <li>- 튜플 삽입, 수정 시 제약 사항 우선 확인</li> <li>- 부모 릴레이션의 튜플 수정, 삭제 시 제약사항 우선 확인</li> </ul>

\* 참고자료 : [https://m.blog.naver.com/hj\\_veronica/220569168323](https://m.blog.naver.com/hj_veronica/220569168323)

# 데이터 무결성 - 릴레이션 무결성

## 1. 릴레이션 무결성의 개요

- 어느 한 튜플이 릴레이션에 삽입될 수 있는가 또는 한 릴레이션과 다른 릴레이션의 튜플들의 관계의 정확성과 정밀성을 유지하는 기법
- 릴레이션을 조작하는 과정에서 의미적 관계를 명세, 삽입/삭제/갱신과 같은 연산을 수행하기 전과 후에 대한 상태에 대한 정확성 보장하는 기법

## 2. 릴레이션 무결성의 유형 (변환기준, 범위기준, 시점기준)

구분	항목	설명
변환 기준	상태제약 (State Constraint)	- DB가 일관성 있는 상태가 되기 위한 조건의 명세 - 각 DB 상태가 모두 만족해야 되는 정적 제약
	과도제약 (Transition Constraint)	- DB의 한 상태에서 다른 상태로 변환되는 과정에 적용되는 규정 - DB 상태의 변환 직전과 직후와 비교가 관련되는 동적 제약
범위 기준	집합제약 (Set Constraint)	- 어떤 튜플 집합전체에 관련된 제약 - Entity 기준까지 확장된 적용
	튜플제약 (Tuple Constraint)	- 처리되고 있는 튜플에만 적용되는 제약 - 대상 Record에만 해당되는 적용
시점 기준	즉시제약 (Immediate Constraint)	- 삽입이나 삭제, 갱신 연산이 수행된 즉시 적용되는 제약 - 트랜잭션 내의 단일 Manipulation 기준 적용
	지연제약 (Deferred Constraint)	- 트랜잭션이 완전히 수행된 뒤에 적용되는 제약 - 업무의 일관성 보장을 위한 트랜잭션 기준 적용

- 동적/정적기준, 적용범위, 적용시점에 따른 릴레이션 무결성 제약

- 릴레이션 무결성 규칙 유지를 위한 다양한 사례 존재

※ 상태(State) : 어느 특정시점에 DB에 저장되어 있는 데이터의 값

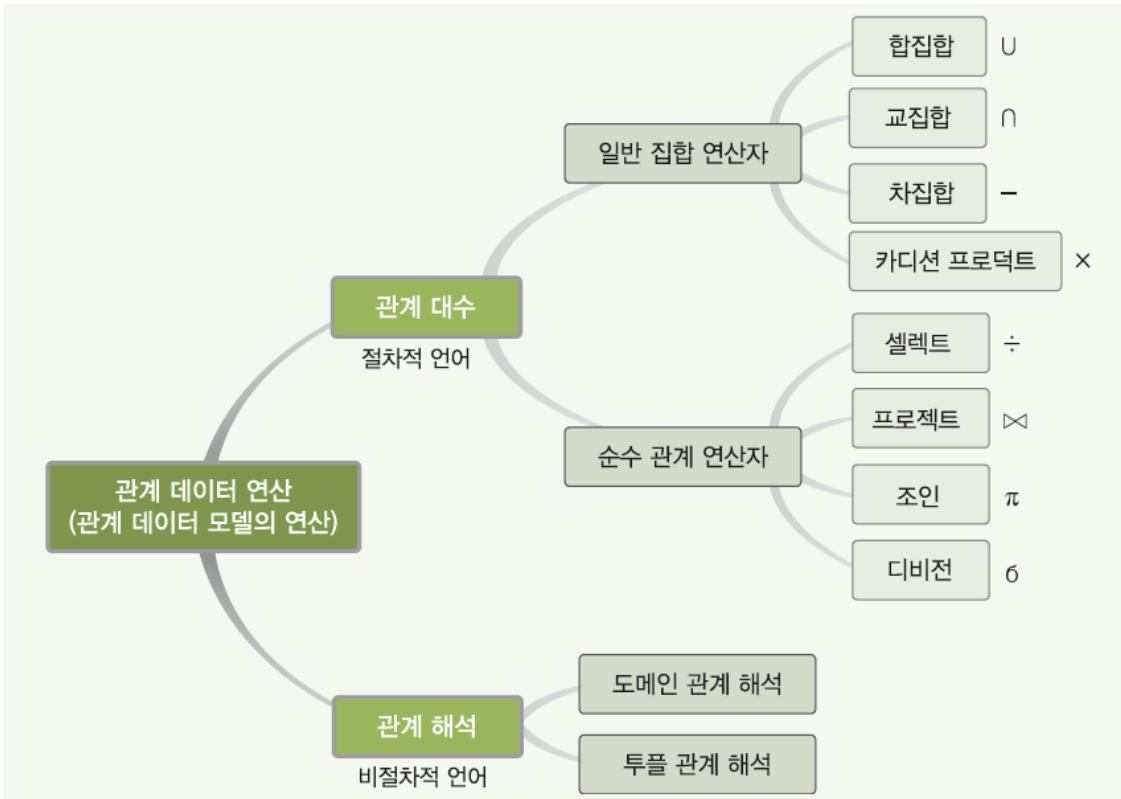
## 3. 릴레이션 무결성 구현기법

구분	항목	설명
변환 기준	상태제약 (State Constraint)	WHEN INSERT CHILD.FAMILY_NAME: CHECK(PARENT.FAMILY_NM=CHILD.FAMILY_NM) - CHILD.FMLY_NM은 항상 PARENT.FMLY_NM 동일
	과도제약 (Transition Constraint)	WHEN UPDATE EMP.SAL: CHECK(EMP.NEW SAL > EMP.OLD SAL); - EMP의 NEW SAL은 OLD SAL보다 크다는 제약
범위 기준	집합제약 (Set Constraint)	AFTER UPDATING EMP.SAL: CHECK(AVG(EMP.SAL) ≤ 300); - EMP 투플 전체의 SAL의 AVG에 적용
	튜플제약 (Tuple Constraint)	AFTER UPDATING EMP.SAL: CHECK(SAL ≤ 500); - EMP 개별 SAL의 대소비교
시점 기준	즉시제약 (Immediate Constraint)	AFTER UPDATING EMP.SEX: CHECK(SEX='M' OR SEX='F'); - SEX를 M 또는 F로 갱신했는지 즉시 확인
	지연제약 (Deferred Constraint)	WHEN COMMIT: CHECK(SUM(ACCOUNT.BALANCE)=SUMMARY.TOTAL); - SUM(ACCOUNT.BALANCE)와 SUMMARY.TOTAL 투플사이의 관계규정 - 다수 투플과 개별 투플간 비교를 위해 일시적 규정 위반이 가능하여, 검사시기를 WHEN COMMIT으로 표현

# 관계 데이터 연산



그림 6-2 관계 데이터 연산의 종류



# 관계대수

일반집합/순수관계

## 1. 관계 데이터 모델에서 지원되는 정형적 언어, 관계대수의 개요

- 관계형 데이터베이스에서 원하는 정보와 그 정보를 검색하기 위해서 어떻게 유도하는가를 기술하는 절차적인 언어
- 원하는 결과를 얻기 위해 릴레이션의 처리 과정을 순서대로 기술하는 언어

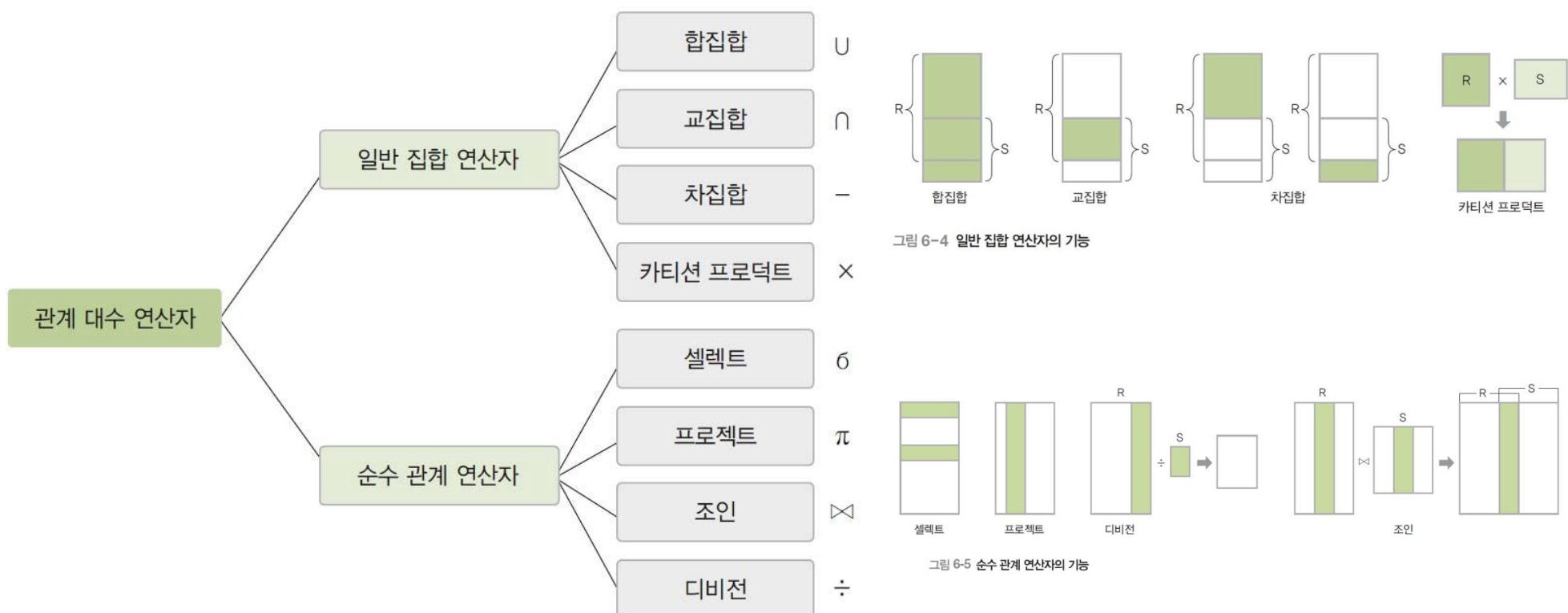


그림 6-3 관계 대수 연산자의 종류

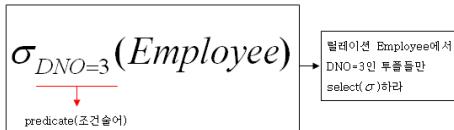
# 관계대수

## 2. 관계대수 연산자의 종류와 표기법

분류	연산자	표기법	단항구분	설명
기본 (근원 연산자)	셀렉션(Selection)	σ	단항	릴레이션에서 주어진 조건에 만족하는 <b>튜플(row)을 선택</b> 하는 연산자
	프로젝션(Projection)	π	단항	주어진 릴레이션에서 어트리뷰트 리스트에 제시된 <b>어트리뷰트만</b> 을 추출하는 연산자(컬럼)
	합집합(Union)	U	이항	두개의 릴레이션을 합하여 하나의 릴레이션을 반환
	차집합(Difference)	—	이항	첫번째 릴레이션에는 속하고 두번째 릴레이션에는 속하지 않는 튜플을 반환
	카티션 곱(Cartesian Product)	×	이항	두 릴레이션을 연결시켜 하나로 합칠때 사용 결과 릴레이션은 첫번째 릴레이션의 오른쪽에 두번째 릴레이션의 모든 튜플을 순서대로 배열하여 반환
복합 연산자	교집합(Intersection)	∩	이항	R과 S <b>모두에 속한(공통) 튜플</b> 들로 이루어진 릴레이션
	세타 조인(Theta Join)	⋈	이항	주어진 조인 조건을 만족하는 두 릴레이션의 모든 튜플을 연결하여 생성된 새로운 튜플로 결과 릴레이션을 구성
	동등 조인(Equijoin)	⋈	이항	<b>두 릴레이션을 조합</b> 하여 결과 릴레이션을 구성
	자연 조인(Natural Join)	*	이항	동등 조인 결과로 얻어진 불필요한 중복되는 어트리뷰트를 한 개 제외한 조인
	세미 조인(Semijoin)	⋈	이항	조인 속성으로 <b>프로젝션 연산을 수행한 릴레이션을 이용하는 조인</b>
	디비전(Division)	÷	이항	릴레이션 속성값의 집합으로 연산을 수행

# 관계대수

select(or selection)



<Employee>

EmpNo	EmpName	Title	Manager	Salary	DNO
2106	나대로	대리	1003	2500000	2
3426	아무개	과장	4377	3000000	1
3011	박문수	부장	4377	4000000	3
1003	갑을병	과장	4377	3000000	2
3427	나숙녀	사원	3011	1500000	3
1365	김개동	사원	3426	1500000	1
4377	이미인	이사	$\wedge$	5000000	2

select할 투플

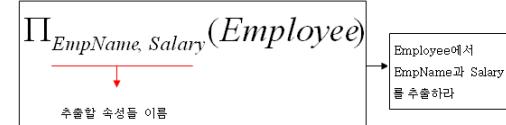
$\sigma_{DNO=3}(Employee)$



<Result>

EmpNo	EmpName	Title	Manager	Salary	DNO
3011	박문수	부장	4377	4000000	3
3427	나숙녀	사원	3011	1500000	3

project(or projection)



<Employee>

EmpNo	EmpName	Title	Manager	Salary	DNO
2106	나대로	대리	1003	2500000	2
3426	아무개	과장	4377	3000000	1
3011	박문수	부장	4377	4000000	3
1003	갑을병	과장	4377	3000000	2
3427	나숙녀	사원	3011	1500000	3
1365	김개동	사원	3426	1500000	1
4377	이미인	이사	$\wedge$	5000000	2

$\Pi_{EmpName, Salary}(Employee)$



<Result>

EmpName	Salary
나대로	2500000
아무개	3000000
박문수	4000000
갑을병	3000000
나숙녀	1500000
김개동	1500000
이미인	5000000

- Q : "사원번호가 2106인 사원의 이름과 직급을 구하라"

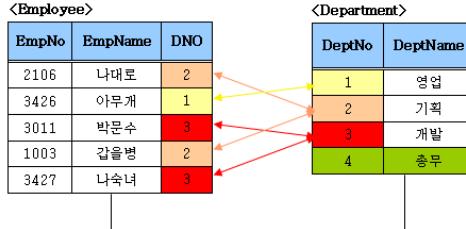
$$\Pi_{EmpName, Title} (\sigma_{EmpNo = 2106} (Employee))$$

김미정 강의자료집(mayching1100@naver.com)

# 관계대수

## 자연조인(natural join)

- Employee 릴레이션과 Department 릴레이션을 동등조인(equi join)하라



EmpNo	EmpName	DNO	DeptNo	DeptName
2106	나대로	2	2	기획
3426	아무개	1	1	영업
3011	박문수	3	3	개발
1003	갑을병	2	2	기획
3427	나숙녀	3	3	개발

EmpNo	EmpName	DNO	DeptNo	DeptName
2106	나대로	2	2	기획
3426	아무개	1	1	영업
3011	박문수	3	3	개발
1003	갑을병	2	2	기획
3427	나숙녀	3	3	개발

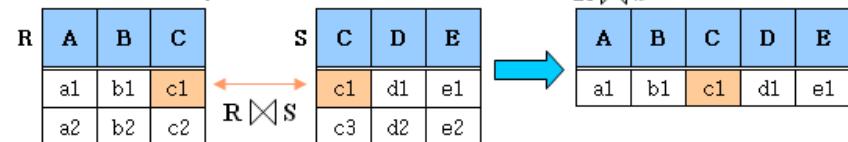
natural join(중복속성 중 하나 제거)

Employee	Department

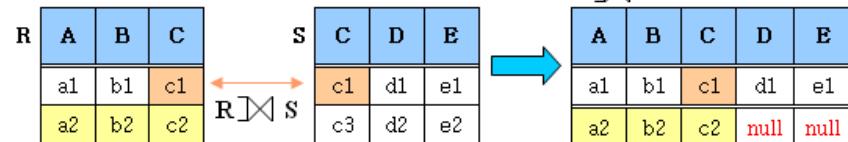
EmpNo	EmpName	DNO	DeptName
2106	나대로	2	기획
3426	아무개	1	영업
3011	박문수	3	개발
1003	갑을병	2	기획
3427	나숙녀	3	개발

## 외부조인(outer join)

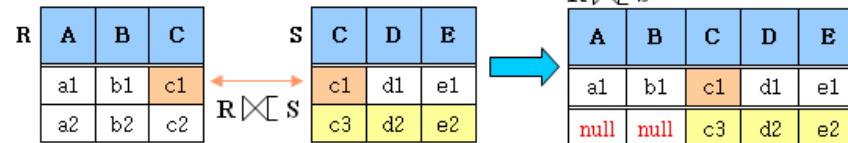
### 자연조인(natural join)



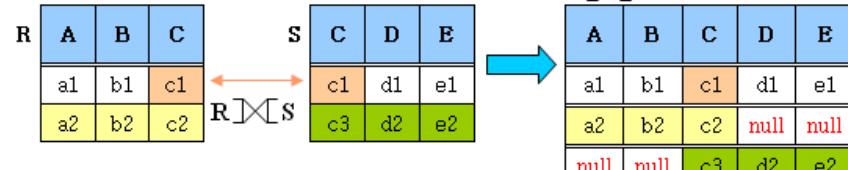
### 왼쪽 외부조인(left outer join)



### 오른쪽 외부조인(right outer join)



### 완전 외부조인(full outer join)



# 관계대수

## 합집합(union)

- Query "나숙녀가 속한 부서이거나 기획부서의 부서번호를 검색하라."
- Query 분석 (나숙녀가 속한 부서번호)  $\cup$  (기획 부서번호)

<Employee>

EmpNo	EmpName	Title	Manager	Salary	DNO
2106	나대로	대리	1003	2500000	2
3426	아무개	과장	4377	3000000	1
3011	박문수	부장	4377	4000000	3
1003	갑을병	과장	4377	3000000	2
3427	나숙녀	사원	3011	1500000	3
1365	김개동	사원	3426	1500000	1
4377	이미인	이사	^	5000000	2



$\Pi_{DNO} (\sigma_{EmpName = "나숙녀"} (Employee))$



Result1

DNO
3

## 차집합(set difference)

- Query "소속된 직원이 한명도 없는 부서의 부서번호를 검색하라"
- Query 분석 :(부서테이블의 부서번호 전체) - (사원테이블의 부서번호 전체)

<Department>

DeptNo	DeptName	Floor
1	영업	8
2	기획	10
3	개발	9
4	총무	7



$\Pi_{DeptNo} (Department)$



Result1

DeptNo
1
2
3
4

# 관계대수

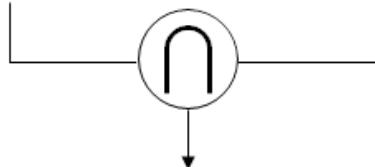
교집합 연산(set intersection)

- 대출과 계좌를 모두 가지고 있는 모든 고객을 찾아라

$$\Pi_{customer-name}(depositor) \cap \Pi_{customer-name}(borrower)$$

customer-name
Hayes
Johnson
Johnson
Jones
Lindsay
Smith
Turner

customer-name
Adams
Curry
Hayes
Jackson
Jones
Smith
Williams



customer-name
Hayes
Jones
Smith

나누기(division)

R ÷ S

A#	B#	C#
a1	b1	c1
a1	b2	c1
a1	b4	c6
a1	b5	c3
a2	b2	c2
a2	b4	c3
a2	b6	c5
a3	b3	c6
a4	b1	c1
a2	b2	c3
a1	b3	c1

S

C#
c6

R ÷ S

A#	B#
a1	b4
a3	b3

A#	B#
a1	b5
a2	b2
a2	b4

A#
a4
a2
a1

C#

C#
c3
c2
c3

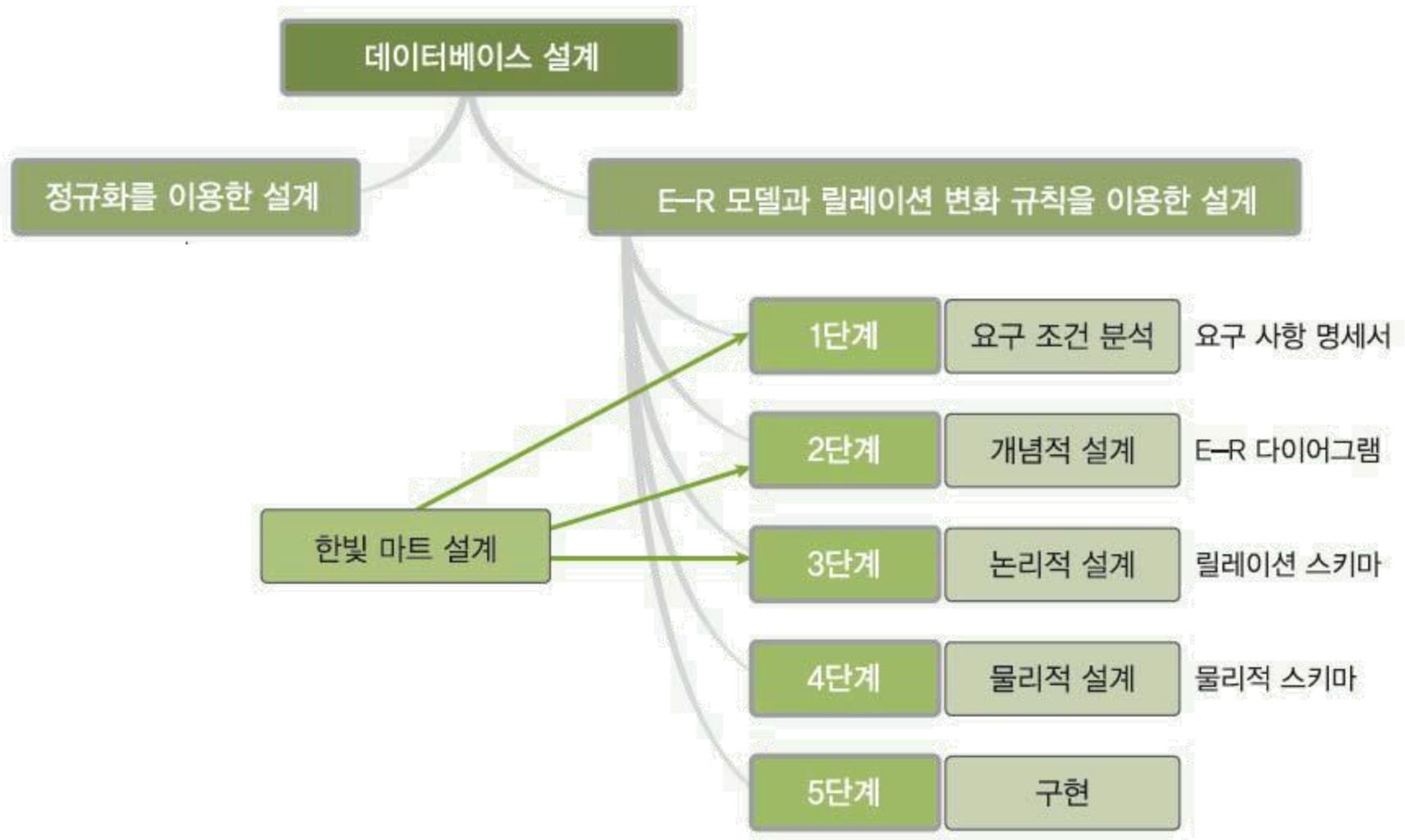
B# C#

B#	C#
b1	c1
b2	c3
b3	c1

B#

B#
b1
b2
b3

# 데이터베이스 설계



# 데이터베이스 분석, 설계, 구축 프로세스

(데이터 모델링 적용 절차)

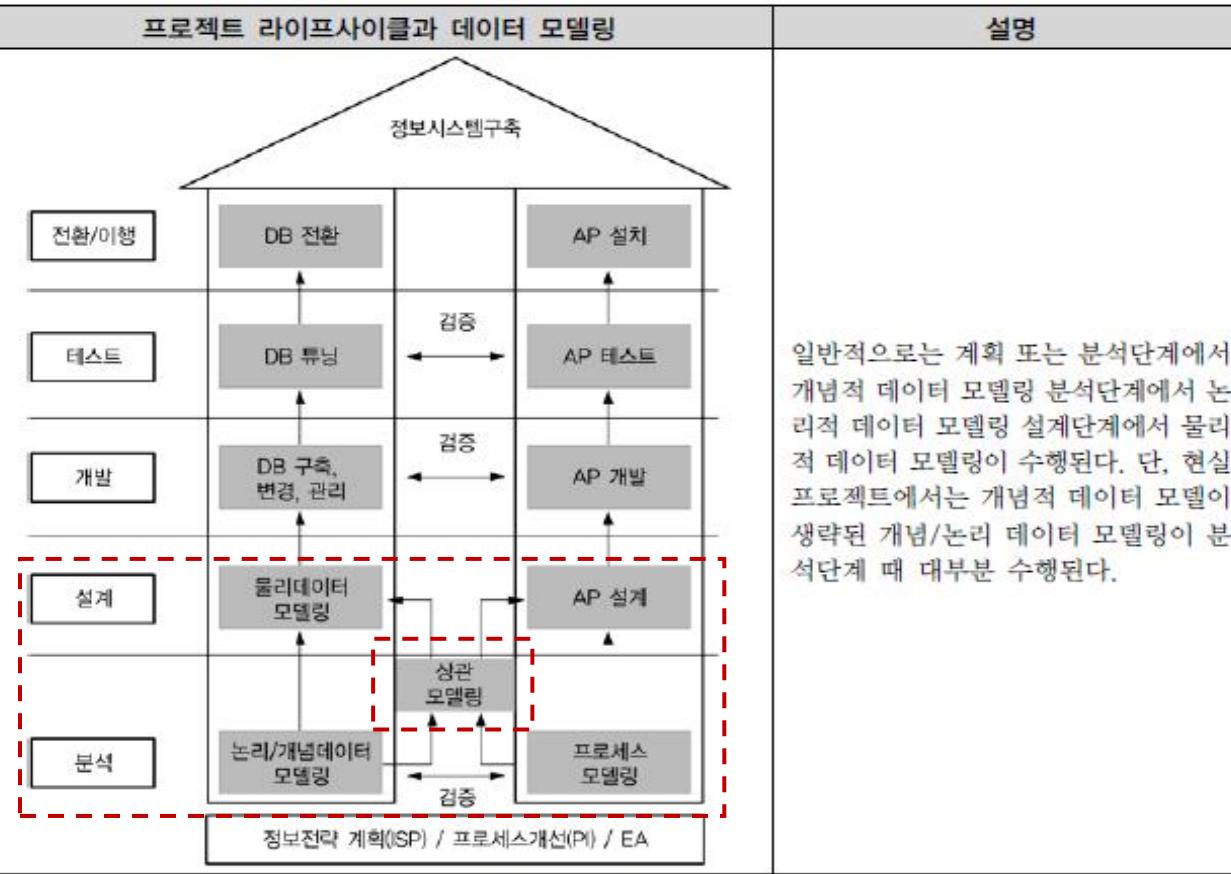
상관모델링

연산, 구조, 제약

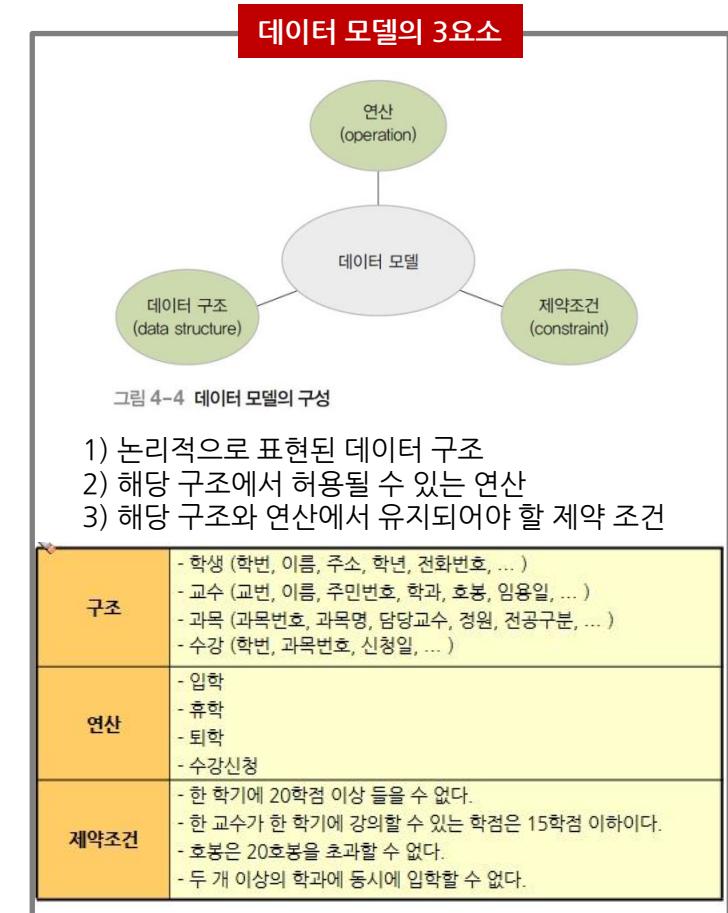
## 1. 사용자의 요구조건에서부터 데이터베이스구조를 도출해내는 DB구축 프로세스의 개요

- 요구수집 및 분석, DB 설계, 구현 및 테스트의 데이터베이스를 효율적으로 관리 할수 있도록 구축하는 일련의과정

## 2. 데이터베이스 구축 프로세스 개념도



출처 : <http://www.dbguide.net/db/db?cmd=view&boardUid=148404&boardConfigUid=9&categoryUid=216&boardIdx=132&boardStep=1>



# 데이터베이스 설계단계 과정

DB설계과정	단계별 내용	결과물
<b>핵심단계</b> <div style="border: 2px dashed red; padding: 5px;"> <p>요구조건 분석 단계</p> <p>데이터 및 처리 요구조건</p> </div> <p>↓</p> <p>개념적 설계 단계</p> <p>DBMS독립적 개념스키마 설계, 트랜잭션 모델링</p> <p>↓</p> <p>논리적 설계 단계</p> <p>목표DBMS에 맞는 스키마 설계, 트랜잭션 인터페이스 설계</p> <p>↓</p> <p>물리적 설계 단계</p> <p>목표DBMS에 맞는 물리적 구조 설계, 트랜잭션 세부 설계</p> <p>↓</p> <p>구현 단계</p> <p>목표DBMS DDL로 스키마 작성, 트랜잭션(응용프로그램)작성</p>	<ul style="list-style-type: none"> <li>데이터베이스에 대한 잠재적 사용자를 식별하고 그 <b>사용자가 원하는 데이터베이스의 용도를 파악</b></li> <li>정보요구조건, 처리요구조건 등을 수집, 분석 명세 -개체, attribute, 관계성, 제약조건 등 정적구조 요구</li> <li>트랜잭션 유형/빈도 등 동적구조 요구</li> <li>경영목표, 정책, 규정 등 범기관적 요구</li> </ul> <ul style="list-style-type: none"> <li>속성들로 기술된 <b>개체타입(entity type)</b>과 이 개체 타입들 간의 관계를 이용해 현실 <b>세계를 표현</b></li> <li>개체-관계 모델 (entity-relationship model)</li> </ul> <ul style="list-style-type: none"> <li>개념적 구조로부터 <b>특정 목표DBMS가 처리할 수 있는 스키마</b>를 생성</li> <li>레코드 타입에 기초를 둔 논리적 개념을 이용하여 데이터 필드로 기술된 데이터 타입과 이 데이터 타입들간의 관계를 이용하여 데이터 모델링</li> <li>데이터 모델링 정규화</li> </ul> <ul style="list-style-type: none"> <li>논리 데이터 모델을 <b>특정 DBMS의 특성 및 성능을 고려하여 효율적이고 구현 가능한 물리적인 스키마</b>를 생성</li> <li>H/W 및 운영체제특성 반영, 저장 레코드 양식 설계</li> <li>레코드 집중화 및 분산, 접근경로 설계</li> <li>응답시간, 저장공간 효율성, 트랜잭션처리, 반정규화</li> </ul> <ul style="list-style-type: none"> <li>데이터베이스 스키마 생성</li> <li>공백 데이터베이스 파일 생성</li> <li>데이터 전환/적재/트랜잭션 테스트</li> </ul>	요구조건 명세
		개념ER모델, ERD
		논리스키마, 상세ER모델
		물리 테이블
		DB구축

# 데이터 모델링

## 1. 현실세계의 데이터를 표현하기 위한 추상화 방법, 데이터 모델링의 개요

- 정보화 시스템을 구축하기 위해, 어떤 데이터가 존재하는지 또는 업무가 필요로 하는 정보는 무엇인지를 분석하는 방법
- 현실세계의 업무 프로세스를 이해하기 쉬운 형태로 추상화하고, 이를 데이터베이스의 데이터 구조로 표현하기 위한 설계과정
- 정보화 시스템을 구축하기 위해, 어떤 데이터가 존재하는지 또는 업무가 필요로 하는 정보는 무엇인지를 분석하는 방법

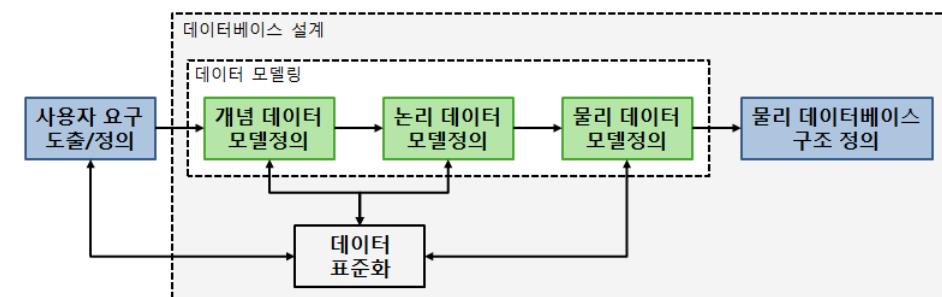
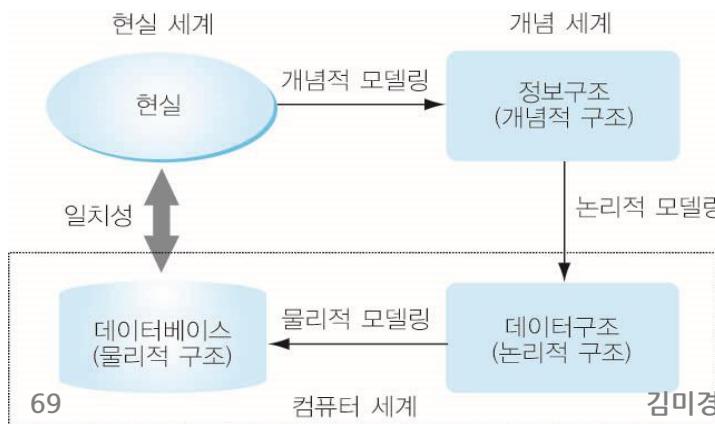
## 2. 데이터모델링의 3가지 관점

관점	설명
데이터 관점	• 업무가 어떤 데이터와 관련이 있는지, 데이터간의 관계는 무엇인지에 대해 모델링하는 방법
프로세스 관점	• 업무를 통해 어떤 일을 처리하는지, 무엇을 해야 하는지를 모델링하는 방법
데이터와 프로세스의 상관 관점	• 업무에서 일을 처리하는 방법에 따라 데이터가 어떻게 영향을 받는지 모델링하는 방법

## 3. 데이터 모델링 3요소

3요소	내용	데이터모델 예
Ithings	업무가 관여하는 어떤 것(THINGS)	
Relationship	업무가 관여하는 어떤 것 간(THINGS)의 관계	
Attributes	어떤 것(THINGS)이 가지는 성격	<p>유화부인 아들 사랑하고 지혜롭고 아름다움</p> <p>주·종 키크고 잘생기고 돌표의식 강하고 항상 친지 침</p> <p>모자 관계</p>

## 4. 데이터 모델링의 개념도

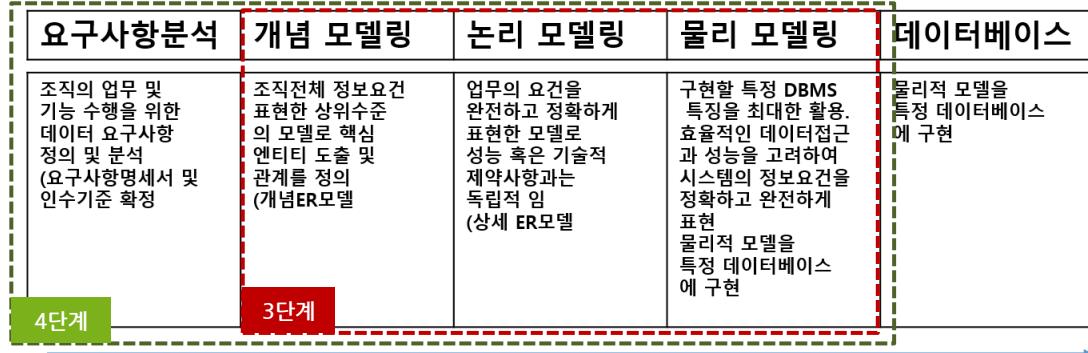


# 데이터 모델링

개념물

## 5. 데이터 모델링의 단계

### 추상화/단순화



### [데이터 모델링 수준]

- 3단계 : 개념+논리+물리
- 4단계 : 요구분석 + 3단계

### 구체화/세분화

단계	설명	산출물
요구분석	<ul style="list-style-type: none"> <li>기본 프로세스별 정보항목 표준화</li> <li>요구사항 기술적 사항 확정</li> <li>요구사항 검증</li> <li>DRB(Data model Review Board)수행</li> </ul>	<ul style="list-style-type: none"> <li>부정확한 요구사항에 대한 <b>애매모호성 제거</b></li> <li>업무처리규정의 각 항목은 엔티티(Entity)관계도와 1:1로 상응</li> </ul>
<b>개념 데이터 모델링 (Conceptual Data Modeling)</b>	<ul style="list-style-type: none"> <li>추상화 수준이 높고 <b>업무중심적이며 포괄적인 수준의 모델링</b> 진행</li> <li><b>현실 세계의 중요 데이터를 추출하여 개념 세계로 옮기는 작업</b></li> <li>전사적 데이터 모델링 (EA 수립시 많이 사용함)</li> </ul>	<ul style="list-style-type: none"> <li>핵심 엔티티 추출</li> <li>속성 및 관계 정의</li> <li>ERD 작성</li> </ul>
<b>논리 데이터 모델링 (Logical Data Modeling)</b>	<ul style="list-style-type: none"> <li>개념 세계의 <b>데이터를 데이터베이스에 저장하는 구조로 표현하는 작업</b></li> <li>시스템으로 구축하고자 하는 업무에 대해 Key, 속성, 관계 등을 정확히 표</li> </ul>	<ul style="list-style-type: none"> <li>식별자 확정</li> <li>재사용성 확보</li> <li><b>정규화 수행</b></li> </ul>
<b>물리 데이터 모델링 (Physical Data Modeling)</b>	<ul style="list-style-type: none"> <li><b>실제로 데이터베이스에 이식할 수 있도록</b> 성능, 물리적 성격을 고려하여 설계</li> <li>DBMS의 특성 및 종류, 구현환경을 감안한 스키마 도출</li> </ul>	<ul style="list-style-type: none"> <li>컬럼의 데이터 타입과 크기 정의</li> <li>제약조건 정의</li> <li>인덱스 정의</li> <li>반정규화</li> </ul>
데이터베이스	<ul style="list-style-type: none"> <li>물리적 모델을 특정 DBMS에 구현</li> </ul>	<ul style="list-style-type: none"> <li>데이터베이스</li> </ul>



# 데이터 모델링

## 5. 데이터 모델링 시 고려사항

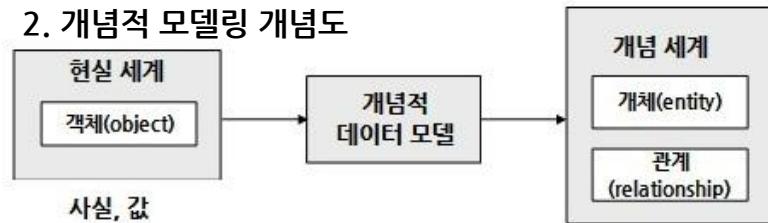
- 최대한 객관화
- 불확실한 업무를 확실하게 정의.(→ 불확실할 경우 ‘연결함정’ 발생)
- 단순한 설계 (다중 사용자 환경, 사용자 및 개발자 등의 관계자와의 통신 프로토콜)
- 성능 고려.
- 관계형 데이터베이스의 특성을 반영
- 모델링의 접근 전략과 사고의 객관화.
- 업무의 순서 중요

# [데이터 모델링] 개념모델링

## 1. 실세계의 대상을 추상적 개념인 개체와 관계로 식별하는 과정, 개념적 데이터 모델링의 개요

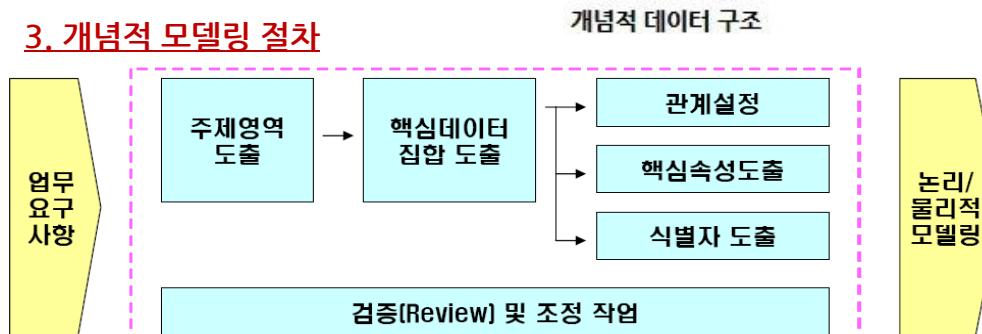
- 해당 조직의 업무요건을 충족하기 위해서 **주제영역과 핵심 데이터 집합간의 관계를 정의**하는 상위 수준의 개략적 데이터 설계 작업
- 특징 : 특정 DB에 상관 없는 독립적인 스키마를 기술 (DB독립적), 업무 중심적**

## 2. 개념적 모델링 개념도



모델	설명	사례
개체-관계 모델	<ul style="list-style-type: none"> <li>개체-관계 모델은 피터 첸(Peter Chen)이 1976년에 제안한 것으로, <b>현실 세계를 개체(entity)와 개체 간의 관계(relationship)</b>를 이용해 <b>개념적 구조로 표현하는 방법</b></li> </ul>	<ul style="list-style-type: none"> <li>요구사항을 설계도로 그리는 과정</li> </ul>
개체-관계 다이어그램	<ul style="list-style-type: none"> <li>개체-관계 모델을 이용해 현실 세계를 개념적으로 모델링 한 결과물을 그림으로 표현한 것</li> </ul>	<ul style="list-style-type: none"> <li>E-R 다이어그램</li> </ul>

## 3. 개념적 모델링 절차



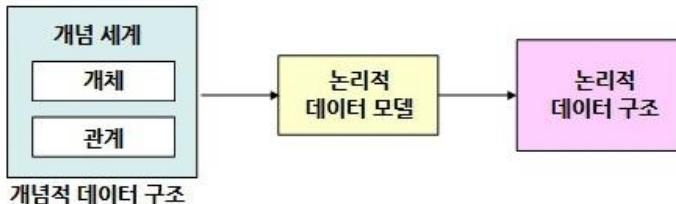
절차	상세내용	기법 및 종류
<b>주제영역 도출</b>	- 하위주제 영역 또는 데이터 집합들로 구성 - 업무 기능과 대응	상향식, 하향식, Inside-out, 혼합식
<b>핵심데이터 집합 도출 (Entity)</b>	- 데이터의 보관 단위로서 주제영역에서 중심이 되는 데이터 집합을 정의	독립중심, 의존중심, 의존특성, 의존연관데이터
<b>관계설정 (Cardinality)</b>	- 업무적 연관성에 따라 개체간 갖는 relationship 설정	1:1, 1:N, M:N, 순환관계
<b>핵심속성 정의 (Attribute)</b>	- 데이터 집합의 특성을 나타내는 항목	원자단위검증, 유일값 유무판단, 관리수준 상세화
<b>식별자 정의 (Identifier)</b>	- 데이터 집합을 유일하게 식별해주는 속성(PK로 구현)	PK, CK, AK, FK로 구분

# [데이터 모델링] 논리적 데이터 모델링

## 1. 업무 중심이면서 데이터 관점의 모델링, 논리적 데이터 모델링

- 업무의 모습을 모델링 표기법으로 형상화 하여 **사람이 이해하기 쉽게 표현**하는 절차
- 논리적 데이터 모델은 완전히 논리적이기 때문에 **특정 물리적 저장구조에 의한 구현을 의미하지는 않음**
- 논리적 데이터 모델로 변화 : DBMS 독립적인 개념적 스키마(개념적 구조)를 관계/계층/네트워크/객체지향모델로 변환하는 것
- 트랜잭션 인터페이스 설계 : 트랜잭션의 전체적인 골격(skeleton)을 개발하고 인터페이스를 정의
- 스키마의 평가 및 정제 : 정량적 정보와 성능 평가 기준에 따라 평가 및 정제

## 2. 논리적 모델링 개념도



모델	설명
관계 데이터 모델	<ul style="list-style-type: none"> <li>데이터베이스의 논리적 구조가 <b>2차원 테이블 형태</b></li> </ul>
계층 데이터 모델	<ul style="list-style-type: none"> <li>데이터베이스의 논리적 구조가 <b>트리 형태</b></li> <li>루트 역할을 하는 개체가 존재하고 사이클이 존재하지 않음</li> </ul>
네트워크 데이터 모델	<ul style="list-style-type: none"> <li>데이터베이스의 논리적 구조가 <b>그래프 형태</b></li> <li>개체 간에는 일대다(1:n) 관계만 허용</li> <li>구조가 복잡하고 데이터의 삽입·삭제·수정·검색이 쉽지 않음</li> </ul>

## 3. 논리적 모델링 절차



[그림 2-1] 논리 데이터모델링체계도

순서	Task	내용
순서 구분없이 수행	Entity type 도출	기본, 중심, 행위 Entity type 도출
	관계 도출	Entity type 간의 관계 도출
	식별자 도출	PK< FK, UK, AK 등에 대한 정의
	속성 도출	용어사전, 도메인 정의, 속성의 규칙(기본값, 체크 값 등) 정의
	세부사항 도출	용어사전, 도메인 정의, 속성의 규칙(기본값, 체크 값 등) 정의
	정규화	1~3차 정규화, BCNF, 4~5차 정규화 적용
	통합/분할	Entity Type의 성격에 따라 통합, 분할 수행
단계별	데이터 모델링 검증	Entity type, 속성, 관계 등에 대한 적합성 검증

# [데이터 모델링] 논리적 모델링

## 4. 논리적 모델링의 주요 Task

주요 Task		내용
개체 상세화	속성 상세화	<ul style="list-style-type: none"> <li>개념 데이터 모델링에서 추출된 핵심 속성 외에 필요한 모든 속성 도출           <ul style="list-style-type: none"> <li>최소 단위 까지 분할(분할 및 통합의 기준은 업무의 요구사항에 준함)</li> <li>하나의 값만(Single Value)를 가지는지 검증</li> <li>추출 속성(Derived Attribute) 아닌지 검증</li> </ul> </li> </ul>
	식별자 확정	<ul style="list-style-type: none"> <li>엔티티를 유일하게 결정짓는 식별자를 확정함           <ul style="list-style-type: none"> <li>각 인스턴스를 유일하게 식별</li> <li>NULL이 될 수 없음</li> <li>자주 변경되지 않는 것이어야 함</li> </ul> </li> <li>인조 식별자를 사용하는 경우에도 그 개체의 의미상의 주어가 되는 본질 식별자 구분</li> </ul>
	정규화	<ul style="list-style-type: none"> <li>논리적 데이터 모델을 일관성 있고 중복을 제거하여 데이터 무결성을 유지하기 위한 바람직한 자료 구조로 만들기 위한 과정</li> <li>대체로 적절하고 일관성을 유지하면서 중복이 없는 논리 데이터 모델 구축을 위해 3차 정규형이 사용됨</li> </ul>
	M:M 관계 해소	<ul style="list-style-type: none"> <li>M:M 관계는 교차엔티티(Interaction Entity)를 도출하여 1:M 관계로 해소</li> </ul>
	이력관리 결정	<ul style="list-style-type: none"> <li>이력으로 관리할 대상 엔티티 결정</li> <li>이력관리 형태를 결정</li> </ul>

## 5. 논리적 모델링 전환 과정

단계	주요개념
Relation (테이블)전환	<ul style="list-style-type: none"> <li>개념 모델링에서의 Entity를 유일성을 보장하는 기본 키를 지정하여 실제 데이터가 저장될 논리적 Relation으로 전환</li> </ul>
Relation(관계)	<ul style="list-style-type: none"> <li>Entity간에 연결고리를 1:1 또는 1:M 형태로 전환</li> </ul>
정규화 수행	<ul style="list-style-type: none"> <li>데이터 중복 저장, 이상 현상 방지를 위해 1NF ~ BCNF 정규형까지 모두 만족하는지를 검사하고, 오류 발생 시 이전 단계를 재 검토</li> </ul>
사용자 트랜잭션 검증	<ul style="list-style-type: none"> <li>도출된 논리적 데이터 모델이 사용자가 원하는 트랜잭션을 모두 만족시키는지를 확인함</li> </ul>
ERD 검증	<ul style="list-style-type: none"> <li>수정 및 보안해야 될 사항에 대해서 ERD에 재 반영</li> </ul>
무결성 제약 조건	<ul style="list-style-type: none"> <li>관계형 DB에서 무결성을 만족하기 위한 제약조건을 설정</li> </ul>

[개념적 모델과 논리적 모델의 속성 맵핑]



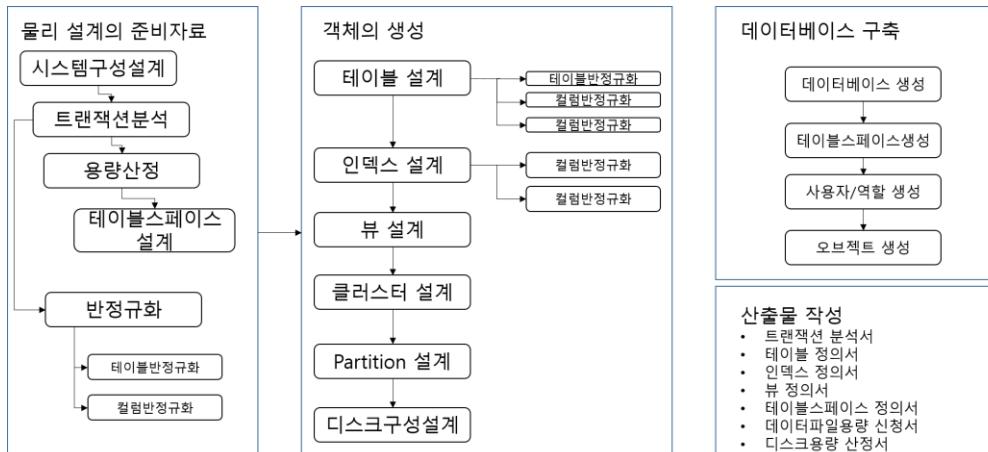
# [데이터 모델링] 물리적 모델링

## 1. 성능을 고려한 모델링, 물리적 데이터 모델링

- 논리 데이터 모델을 **특정 DBMS의 특성에 맞게 성능을 고려**하여 저장, **물리적인 스키마**를 만드는 일련의 과정
- 작성된 논리적 모델을 실제 컴퓨터의 저장 장치에 저장하기 위한 물리적 구조를 정의하고 구현하는 과정
- 논리 모델인 ERD를 물리 모델링 ERD로(테이블 관계도)로 전환
- 특징 : 성능향상 중요, 시스템환경 종속적

## 2. 물리 모델로의 변환 주요 TASK

단계	과정	고려사항
일괄전환	Entity별 Table로의 전환	<ul style="list-style-type: none"> <li>Sub Type 설계 방안</li> </ul>
	식별자의 Primary Key 정의	<ul style="list-style-type: none"> <li>Artificial Key 검토, PK 컬럼 순서 검토</li> </ul>
	속성의 컬럼 전환	<ul style="list-style-type: none"> <li>영문 컬럼명 매핑데이터 타입길이 결정정의컬럼의 순서</li> </ul>
	관계의 컬럼 전환	<ul style="list-style-type: none"> <li>참조무결성 규칙 및 구현방향 결정</li> </ul>
구조조정	수퍼타입/서브타입 모델 전환	<ul style="list-style-type: none"> <li>트랜잭션의 성격에 따라 전체 통합부분 통합개별 유저에 대한 의사 결정을 통해 데이터 모델 조정</li> </ul>
성능향상	성능을 고려한 <b>반정규화</b>	<ul style="list-style-type: none"> <li>SQL 활용 능력의 미흡으로 인한 빈번한 반정규화는 배제하도록 신중하게 검토</li> </ul>

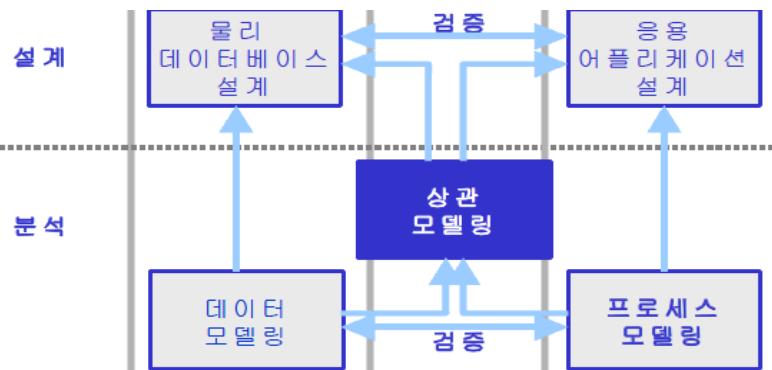


# DB 상관 모델링(CRUD - Matrix)

## 1. 프로세스와 데이터간의 상관관계 표현을 위한 Matrix, 상관(Interaction) 모델링의 개요

- 시스템 개발 시 **프로세스(또는 메소드, 클래스)와 DB에 저장되는 데이터 사이의 Dependency**를 나타내기 위한 Matrix

## 2. CRUD Matrix 작성 개념도 및 내용



CRUD	내용
C (Create)	하나의 업무 기능이 <u>데이터를 생성</u> 하는 관계
R (Read)	하나의 업무 기능이 업무 수행의 목적을 달성하기 위해서 <u>데이터를 참조</u> 하는 관계
U (Update)	하나의 업무를 수행하는 과정에서 <u>데이터가 수정/갱신</u> 되는 관계
D (Delete)	하나의 업무를 수행하는 과정에서 <u>데이터가 삭제</u> 되는 관계

## 3. CRUD Matrix 점검방법

- 모든 Entity Type은 반드시 자신과 관련된 단위프로세스가 존재해야 함
- 불필요한 Entity Type 도출, 적절한 단위 프로세스가 도출되지 않은 경우
- 단위 프로세스의 CRUD가 아직 충분히 정의되지 않은 경우

엔티티타입 단위 프로세스	고객	주문	주문목록	제품	
신규고객을 등록한다.	C				
제품주문을 신청한다.	R	C	C	R	
주문량을 변경한다.		R	U		
주문을 취소한다.		D	D		
고객정보를 조회한다.	R				

※ 엔티티타입에 조회(R)는 존재하는데 데이터를 생성하는 단위 프로세스가 없다.

엔티티타입 단위 프로세스	고객	주문	사원	주문목록	제품	
신규고객을 등록한다.	C					
주문을 신청한다.	R	C				
주문량을 변경한다.		R				
주문을 취소한다.		D				
제품을 등록한다.			C		R	
고객정보를 조회한다.	R			U	D	
					C	

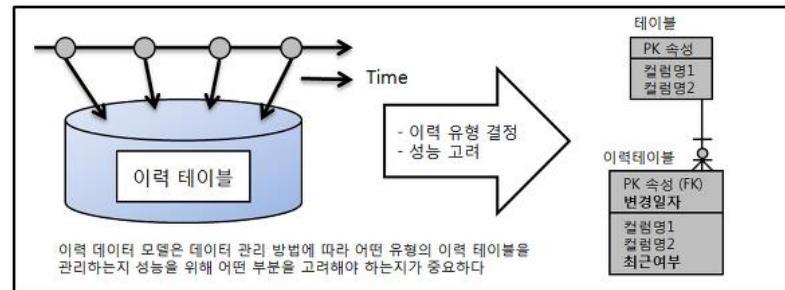
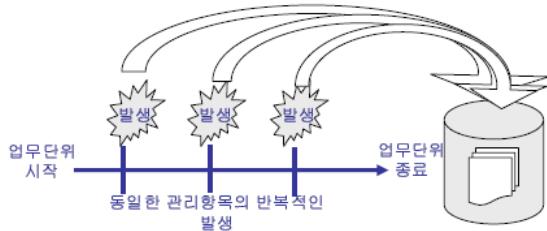
※ 엔티티타입에 발생되는 CRUD가 존재하지 않음

# 이력데이터 모델링

시점/선분      로우/컬럼      변경/발생/진행      스냅샷

## 1. 변경이력을 정보로서 관리, 이력 데이터 모델링의 정의

- 시간의 변화에 따른 데이터의 **변경이력을 정보로서 관리**하기 위하여 시간을 나타내주는 애트리뷰트를 첨가하여 관리하는 데이터 모델링 기법



- 시간에 따라 발생**하며, **동일한 컬럼의 유형에 발생**
- 시간에 따라 반복적으로 발생하기 때문에 **대량의 데이터가 발생**할 가능성이 높음
- 성능에 영향**을 주는 경우가 많음
- 이력은 시간에 따라 **발생->변경->진행이력의 형식**으로 구분

## 2. 이력의 유형

### 1) 컬럼과 로우 단위에 따른 유형

이력 이름	개념도	내용
유형1) 로우 단위 이력	테이블 PK 속성 컬럼명 + 이력테이블 PK 속성 (FK) 발생일자 컬럼명	- 이력 테이블에는 마스터 테이블의 변경된 내용이 하나의 로우에 전체적으로 기록되는 형태(일반적인 이력)
유형2) 컬럼 단위 이력	테이블 PK 속성 컬럼명1 컬럼명2 + 이력테이블 PK 속성 (FK) 변경컬럼코드 변경일자 코드 001: 컬럼명1 변경컬럼코드 변경일자 002: 컬럼명2 변경컬럼코드 변경일자	- 이력 테이블에는 마스터 테이블의 변경된 내용이 하나의 로우에 하나의 컬럼 변경된 내용이 기록된 형태

### 2) 이력데이터 발생 방법에 따른 유형

이력 이름	개념도	내용
유형1) 변경이력	테이블 PK 속성 컬럼명1 컬럼명2 + 이력테이블 PK 속성 발생일자 컬럼명1 컬럼명2 변경 컬럼 이력관리	- 마스터 테이블의 컬럼이 변경되면 이력을 관리하는 형태
유형2) 발생이력	이력테이블 PK 속성 발생일자	- 마스터 테이블의 PK를 포함하여 전체에 대해 인스턴스 생성 - 엄격하게 구분하면 이력 형식이 아닌 인스턴스 생성이라고도 구분할 수 있음
유형3) 진행이력	이력테이블 PK 속성 일련번호 상태코드 컬럼명 ...	- 업무진행 상태에 따라 업무의 상태정보를 관리하는 관리 - 상태정보가 계속 영향을 미치는 형태

# 이력데이터 모델링

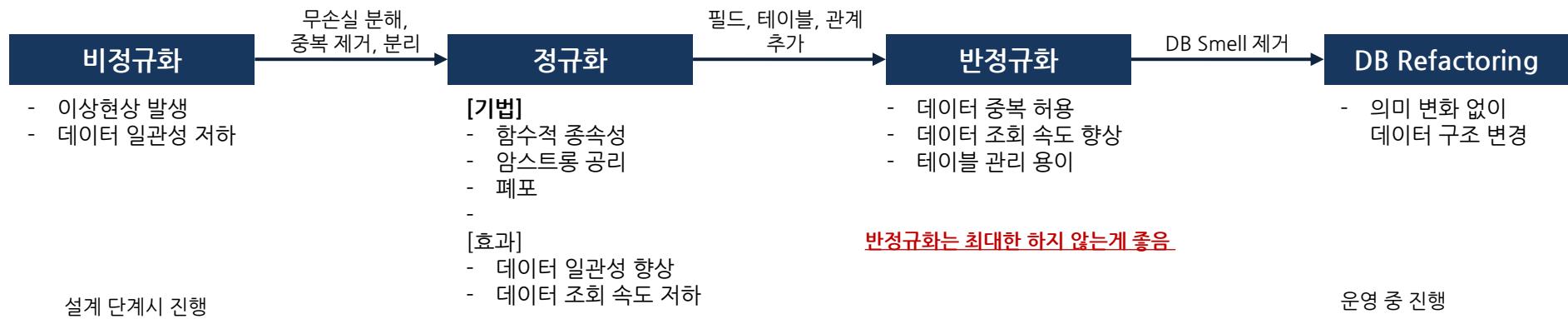
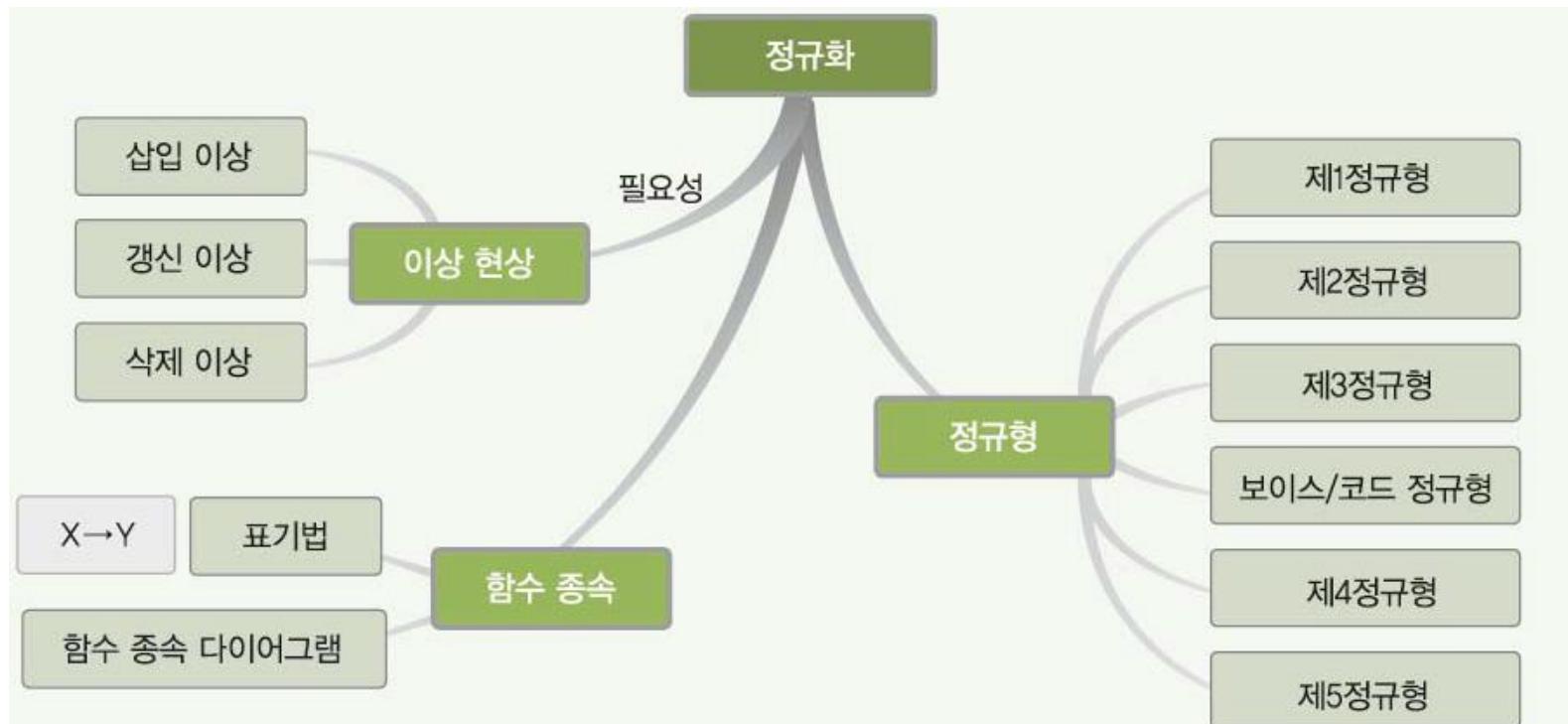
## 3) 테이블 구분에 따른 유형

이력 이름	개념도	내용	항목	시점이력	선분이력
유형1) 내부스냅샷 이력	이력테이블 PK 속성 발생일자	<ul style="list-style-type: none"> <li>- 별도의 테이블 없이 시간에 따라 자기 자신의 테이블에 데이터가 발생되는 구조임</li> <li>- 이력 테이블이 마스터이므로 관계를 통해 PK구조도 모두 상속됨</li> <li>- 특징 : 일부 속성값이 변경되어도 전체 속성값 생성</li> </ul>	개념	<ul style="list-style-type: none"> <li>- 이력의 시점만 관리하는 방식</li> </ul>	<ul style="list-style-type: none"> <li>- 시작시점과 종료시점을 함께 관리하는 방식</li> <li>- 개체의 상태가 지속된 유효기간을 관리</li> </ul>
유형2) 1:M 스냅샷 전체 이력	테이블 PK 속성	이력테이블 PK 속성(FK) 발생일자	개념도	<p>1355# 1352# 1350# 10:10 11:20</p>	<p>1355# 1352# 1350# 09:00 10:10 11:20</p>
유형3) 1:M 스냅샷 과거 이력	테이블 PK 속성	이력테이블 PK 속성(FK) 발생일자	장점	<ul style="list-style-type: none"> <li>- 이력 엔티티의 개체 무결성을 보장</li> <li>- DML이 비교적 쉬움</li> </ul>	<ul style="list-style-type: none"> <li>- 쿼리가 간단</li> <li>- 특정시점 조회시 성능상 유리</li> </ul>
유형4) 1:M 스냅샷 군집 전체 이력	테이블 PK 속성 컬럼명1 컬럼명2	이력테이블1 PK 속성(FK) 발생일자 컬럼명1  이력테이블2 PK 속성(FK) 발생일자 컬럼명1	단점	<ul style="list-style-type: none"> <li>- 특정 시점 데이터를 조회시 쿼리 복잡</li> <li>- 조회시 성능상의 문제 발생</li> </ul>	<ul style="list-style-type: none"> <li>- 시작시점과 종료시점을 같이 변경하는 불편함</li> <li>- DML에 대한 부하</li> <li>- 개체 무결성을 완벽히 보장하기 어려움</li> </ul>
유형5) 1:M 스냅샷 군집 과거 이력	테이블 PK 속성 컬럼명1 컬럼명2	이력테이블1 PK 속성(FK) 발생일자 컬럼명1  이력테이블2 PK 속성(FK) 발생일자 컬럼명1	사례	<p>通话 #通话_ID +국가 ... 환율변동이력 #발생시간 +환율 ... &lt; 그림 3 &gt;</p> <ul style="list-style-type: none"> <li>- 점형태의 SQL</li> <li>- Select 환율 from 환율변동이력 where 발생시간 = (select max(발생시간) from 환율변동이력 where 발생시간 &lt;= '20150901090000')</li> </ul>	<p>通话 #通话_ID +국가 ... 환율변동이력 #시작시간 +종료시간 +환율 ... &lt; 그림 3 &gt;</p> <ul style="list-style-type: none"> <li>- 선분형태의 SQL</li> <li>- Select 환율 from 환율변동이력 where '20150901090000' between 시작시간 and 종료시간</li> </ul>

# 정규화

## [정규화 계산문제 풀이 과정]

- 이상현상 제시 → 함수적 종속성을 이용 대상 정규화 해야 하는 이유 → 암스트롱 공리/폐포를 이용해 키 도출 → 정규화 수행





# 이상현상 (anomaly)

삽입, 갱신, 삭제

## 1. 이상(anomaly) 현상

- 불필요한 데이터 중복으로 인해 릴레이션에 대한 데이터 삽입·수정·삭제 연산을 수행할 때 발생할 수 있는 부작용

## 2. 이상현상의 유형

삽입이상				
• 릴레이션에 새 데이터를 삽입하려면 불필요한 데이터도 함께 삽입해야 하는 문제				
<b>[예시]</b>				
<ul style="list-style-type: none"> <li>아직 이벤트에 참여하지 않은 아이디가 “melon”이고, 이름이 “성원용”, 등급이 “gold”인 신규 고객의 데이터는 이벤트참여 릴레이션에 삽입할 수 없음</li> <li>삽입하려면 실제로 참여하지 않은 임시 이벤트 번호를 삽입해야 함</li> </ul>				
<p>그림 9-3 이벤트참여 릴레이션의 삽입 이상</p>				

갱신이상				
• 릴레이션의 중복된 투플들 중 일부만 수정하여 데이터가 불일치하게 되는 모순이 발생하는 문제				
<b>[예시]</b>				
<ul style="list-style-type: none"> <li>아이디가 “apple”인 고객의 등급이 “gold”에서 “vip”로 변경되었는데, 일부 투플에 대해서만 등급이 수정된다면 “apple” 고객이 서로 다른 등급을 가지는 모순이 발생</li> </ul>				
<p>그림 9-4 이벤트참여 릴레이션의 갱신 이상</p>				

삭제이상				
• 릴레이션에서 투플을 삭제하면 꼭 필요한 데이터까지 손실되는 연쇄 삭제 현상이 발생하는 문제				
<b>[예시]</b>				
<ul style="list-style-type: none"> <li>아이디가 “orange”인 고객이 이벤트 참여를 취소해 관련 투플을 삭제하게 되면 이벤트 참여와 관련이 없는 고객아이디, 고객이름, 등급 데이터까지 손실됨</li> </ul>				
<p>그림 9-5 이벤트참여 릴레이션의 삭제 이상</p>				

## 정규화를 통해 이상현상 제거

## 1. Anomaly 현상을 해결하기 위한 무손실 분해의 원리, 정규화의 개요

- 관계형 데이터 모델에서 **데이터의 중복성을 제거**하여 **이상 현상을 방지**하고 **데이터의 일관성과 정확성을 유지하기 위한 과정**
- 속성(Attribute)들간의 종속성(Dependency)을 분석하여 기본적으로 하나의 종속성이 하나의 릴레이션(Relation)으로 표현되도록 분해해 나가는 과정
- 이상 현상(Anomaly)을 야기하는 Attribute 간의 종속 관계를 제거하기 위해 Relation을 작은 Relation으로 무손실 분해하는 과정
- 함수 종속성을 이용해 릴레이션을 연관성이 있는 속성들로만 구성되도록 분해해서 이상 현상이 발생하지 않는 바람직한 릴레이션으로 만들어 가는 과정**

## 2. 정규화의 원리 (원칙)

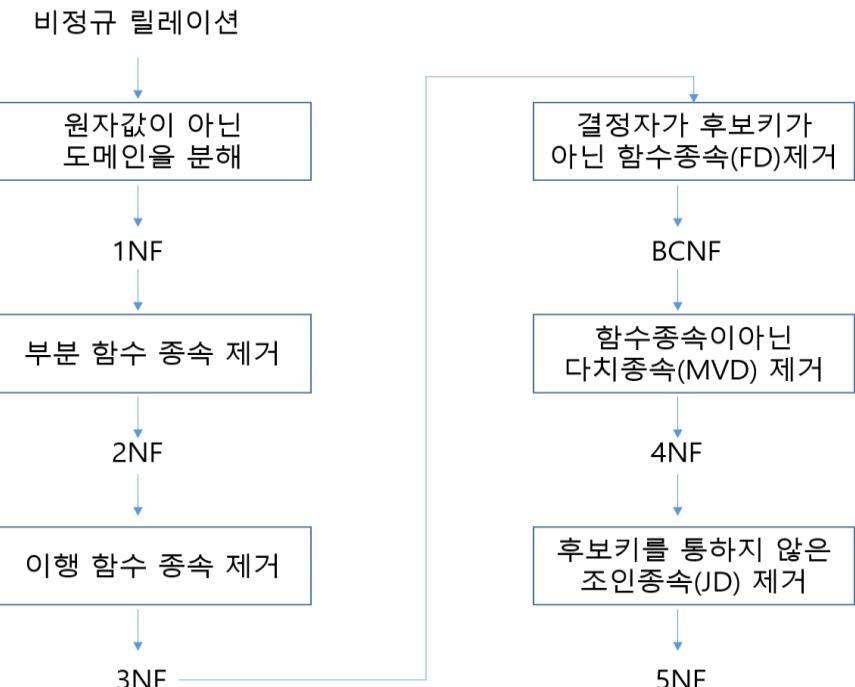
- 정보의 **무손실 분해** (nonloss representation of information)
- 최소의 데이터 중복성** (minimal data redundancy)
- 분리**의 원칙 (principle of separation)

### [정규형(NF; Normal Form)]

- 릴레이션이 정규화된 정도
- 예) 1차 정규형은 2차 정규화를 진행해야 함



일반적으로 정규화를 수행해야 데이터처리의 성능이 향상되며  
데이터의 조회처리 트랜잭션시에 성능저하가 나타날 수 있음



# 함수적 종속성

완전, 부분, 이행, 결정자

## 1. 정규화의 기반이론, 함수적 종속성

- 릴레이션 내의 모든 투플을 대상으로 하나의 X 값에 대한 Y 값이 항상 하나(X가 Y를 함수적으로 결정한다)
- 함수 종속성을 이용하여, 릴레이션을 연관성이 있는 속성들로만 구성되도록 분해하여 이상 현상이 발생하지 않는 바람직한 릴레이션 만들어 나감

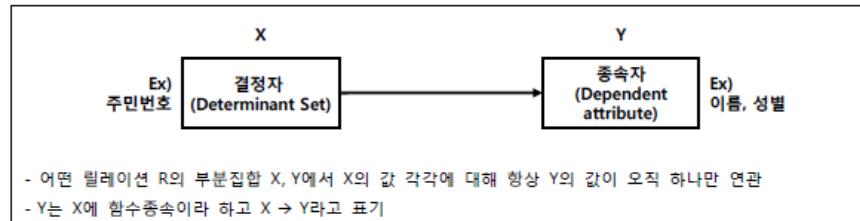


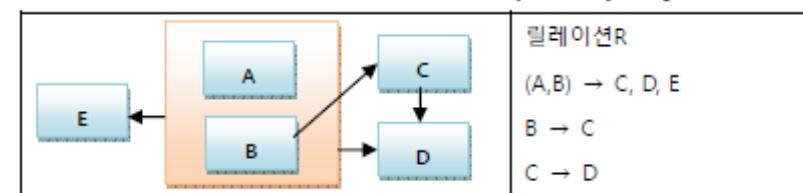
그림 9-8 고객 릴레이션에 존재하는 함수 종속 관계

## 2. 함수종속성 종류 : FDD(FD Diagram)을 통해 복잡한 함수 종속 관계를 표현하고 암스트롱 공리를 통해 함수종속의 성질을 유도

### 1) 함수적 종속성

종류	설명	정규화
완전함수종속성	• $X' \subset X$ 이고 $X' \rightarrow Y$ 를 만족하는 애트리뷰트 $X'$ 이 존재하지 않음	-
부분함수종속성	• $X' \subset X$ 이고 $X' \rightarrow Y$ 를 만족하는 애트리뷰트 $X'$ 이 존재함	2NF
이행함수종속성	• 두 함수 종속성이 $X \rightarrow Y$ 이고 $Y \rightarrow Z$ 일 때, $X \rightarrow Z$ 가 성립	3NF
결정자함수종속성	• 함수적 종속이 되는 결정자가 후보키가 아닌 경우, 즉 $X \rightarrow Y$ 일 때 $X$ 가 후보키가 아님	BCNF

함수종속다이어그램 (FDD: Functional Dependency Diagram)



### 2) 다중화로 인한 종속 관계

다중값 종속성	한 관계에 둘 이상의 독립적 다중값 속성 존재	4NF
조인 종속성	관계 중에서 둘로 나눌 때는 원래의 관계를 회복할 수 없으나 세트 이상으로 분리시 원래의 관계를 회복하는 관계	5NF

## 3. 함수 종속 관계 판단 시 유의 사항

- 속성 자체의 특성과 의미를 기반으로 함수 종속성을 판단해야 함
- 일반적으로 기본키와 후보기는 릴레이션의 다른 모든 속성들을 함수적으로 결정함
- 기본키나 후보기가 아니어도 다른 속성 값을 유일하게 결정하는 속성은 함수 종속 관계에서 결정자가 될 수 있음

# 함수적 종속성

(1NF)

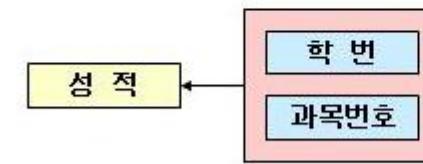


(1NF)

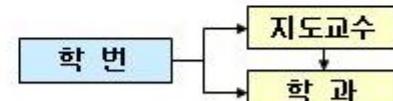


부분함수종속 제거

(2NF)



{학번, 과목번호} → 성적



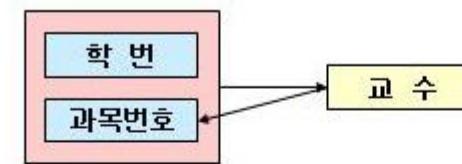
학번 → 지도교수  
학번 → 학과  
지도교수 → 학과

(2NF)



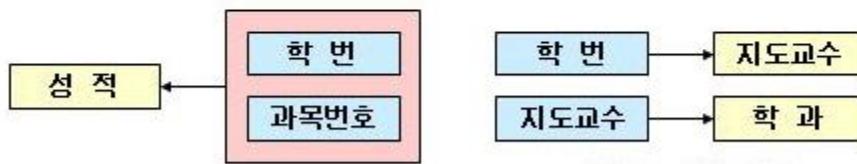
미행적 함수종속 제거

(3NF)



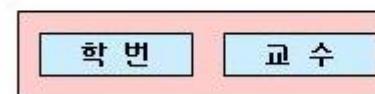
결정자가 후보키가 아닌 함수종속 제거

(3NF)

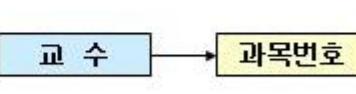


학번 → 지도교수  
지도교수 → 학과

(BCNF)



기본키: 학번, 교수  
외래키: 교수  
참조: 교수과목번호



기본키: 교수

# 암스트롱 공리

재증이연분가

## 1. 함수적 종속성 추론, 암스트롱 공리

- 릴레이션 R에 대해 X, Y, Z라는 애트리뷰트의 집합이 주어졌을 경우 여러 가지 **함수종속의 성질을 유도해 낼 수 있는 추론 규칙**

## 2. 암스트롱의 추론 규칙

구분	규칙	내용
기본	<b>재귀</b>	Y가 X의 부분집합이면 $X \rightarrow Y$ 이다
	<b>증가</b>	$X \rightarrow Y$ 이면, $XZ \rightarrow YZ$ 이다
	<b>이행</b>	$X \rightarrow Y$ 이고, $Y \rightarrow Z$ 이면, $X \rightarrow Z$ 이다.
부가	<b>연합</b>	$X \rightarrow Y$ 이고, $Y \rightarrow Z$ 이면, $X \rightarrow YZ$ 이다
	<b>분해</b>	$X \rightarrow YZ$ 이면, $X \rightarrow Y$ 이고, $X \rightarrow Z$ 이다.
	<b>가이행</b>	$X \rightarrow Y$ 이고, $YW \rightarrow Z$ 이면, $XW \rightarrow Z$ 이다

\* **공리** : 어떤 이론체계에서 가장 기초적인 근거가 되는 명제(命題)

지식이 참된 것이 되기 위해서는 근거가 필요하나 근거를 소급해 보면 더 이상 증명하기가 곤란한 명제

# 폐포(closure)

## 1. 함수적 종속성의 추론규칙, F의 폐포(closure)의 개요

- FD의 집합
- F로부터 추론할 수 있는 모든 가능한 함수적 종속성들의 집합 (표기법 : F+)
- 키(K)는 자신의 폐포가 모든 애트리뷰트를 포함하는 애트리뷰트 집합을 의미, K+ = {ALL}

나를 포함한,  
나를 레퍼런스하는 모든 녀석들의 집합

## 2. F 하에서 속성 집합 X의 폐포(closure of X under F): X+

- 함수적 종속성 집합 F를 사용하여 X에 의해 함수적으로 결정되는 모든 애트리뷰트의 집합

## 3. F하의 X의 폐포 X+를 구하는 알고리즘 및 예제

```
X+ := X;
X repeat
  oldX+ := X+;
  for each functional dependency Y→Z in F do
    if Y⊆X+ then X+ := X+UZ;
  until (oldX+ = X+);
```

F {SSN→ENAME, PNUMBER→{PNAME, PLOCATION}, {SSN, PNUMBER}→HOURS}

F의 폐포

SSN+ = {SSN, ENAME}

PNUMBER+ = {PNUMBER, PNAME, PLOCATION}

SSN, PNUMBER+ = {SSN, PNUMBER, ENAME, PNAME, PLOCATION, HOURS}

릴레이션 R(A,B,C,D,E,F,G) 의 함수 종속성 집합 FD={AB→CF, A→G, BC→E C-D} 일 때 키를 구하라.

AB+ = {ABCFGED}  
 A+ = {AG}  
 BC+ = {BCDE}  
 C+ = {CD}

- AB+ 폐포 과정중에 조합이 되어 도출이 가능한 것도 전부 포함시켜야 함  
 예 ) AB+ 내부에 조합으로 BC 가 포함되어 BCDE가 포함이 됨

# [정규화] 1차 정규화 (1NF)

## 1. 반복되는 속성 제거, 1차 정규화

- 엔티티에서 하나의 속성이 복수개의 값을 갖도록 설계되어 있을 때 하나의 속성이 단일 값(atomic value)을 갖도록 설계를 변경하는 과정
- 모든 엔티티 타입의 속성에는 하나의 속성값만을 가지고 있어야 하며 반복되는 속성의 집단은 별도의 엔티티 타입으로 분리 함 ( 전 제조건은 결정자에 의존하는 의존자의 반복성을 나타냄 )
- 정규화만 되어 있으면 1차정규화(1NF)라고 선언 각 속성에 값이 반복 집단이 없는 원자값(Atomic Value)으로만 구성되어 있어야 한다(**릴레이션의 속성값이 반복 집단이 없는 즉, 더 이상 분해될 수 없는 원자값으로만 구성**)
- 완전함수종속** : 레이션 R의 어떤 애트리뷰트 Y가 다른 복합 애트리뷰트 X에 함수종속이면서, X의 어떤 진부분 집합에도 함수 종속이 아닐 때 Y는 X에 완전 함수 종속이라고 함

### ● 제1정규화 과정의 개념



완전함수종속 ( 성적은 {학번, 과목번호}에 완전함수종속 )

## 1차 정규화의 응용

주문

주문번호
주문일자
배송요청일자
제품번호
제품명
제품단가

1차 정규화 대상

주문번호
주문일자 배송요청일자

1차 정규화 대상

주문제품

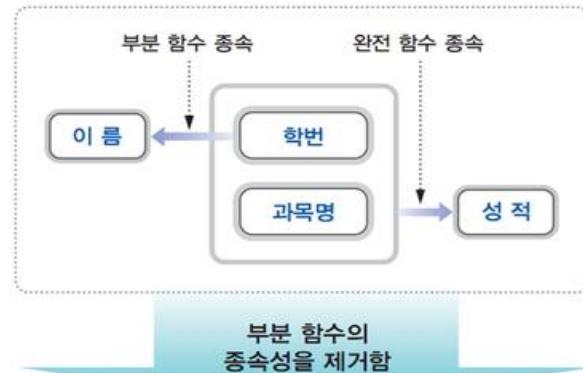
주문번호 (FK)
제품번호
제품명
제품단가

1차 정규화 대상

# [정규화] 2차 정규화

## 1. 부분함수 종속성 제거, 2차 정규화

- 주식별자가 아닌 속성을 중에서 주식별자 전체가 아닌 일부 속성에 종속된 속성을 찾아 제거하는 과정
- 어떤 릴레이션 R이 제1정규형이고, 키에 속하지 않는 속성 모두가 키에 완전 함수 종속이 되도록 해야 함



수 강			
학 번	과 목 명	성 적	이 름
100	전자계산기구조	92	김사랑
101	데이터베이스	82	오지호
100	운영체제	90	김사랑
101	데이터 통신	76	오지호
102	운영체제	82	이선균

부분 함수의 종속성을 제거함

수 강 1

학 생

학 번

이 름

학 번

과 목 명

성 적

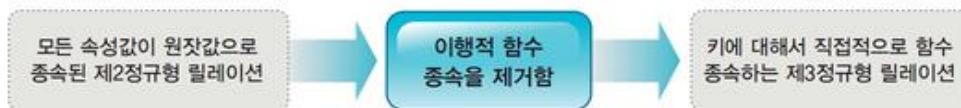
학 번	이 름
100	김사랑
101	오지호
102	이선균

학 번	과 목 명	성 적
100	전자계산기구조	92
101	데이터베이스	82
100	운영체제	90
101	데이터 통신	76
102	운영체제	82

# [정규화] 3차 정규화

## 1. 이행함수 종속성 제거, 3차 정규화

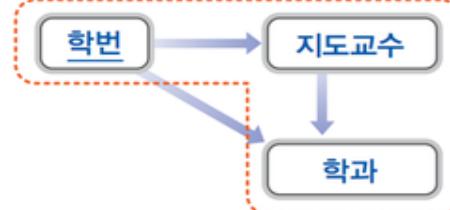
- 주식별자가 아닌 속성들 중에서 종속관계에 있는 속성을 찾아 제거하는 과정



### 지도

학번	지도교수	학과
100	이문세	컴퓨터 소프트웨어학과
101	김연아	멀티미디어학과
102	이문세	컴퓨터 소프트웨어학과
103	강승범	경영 정보학과
104	이문세	컴퓨터 소프트웨어학과
105	김연아	멀티미디어학과

이행적 함수 종속



94 page

예

학번 → 지도교수 ∧ 지도교수 → 학과 → 학번 → 학과

이행적 함수 종속을 제거함



### 지도(이행적 함수 종속 릴레이션)

학번	지도교수	학과
100	이문세	컴퓨터 소프트웨어학과
101	김연아	멀티미디어학과
102	이문세	컴퓨터 소프트웨어학과
103	강승범	경영 정보학과
104	이문세	컴퓨터 소프트웨어학과
105	김연아	멀티미디어학과

이행적 함수 종속을 제거함

### 지도1(이행적 함수 종속 제거한 3NF)

학번	지도교수
100	이문세
101	김연아
102	이문세
103	강승범
104	이문세
105	김연아

### 교수(이행적 함수 종속 제거한 3NF)

지도교수	학과
이문세	컴퓨터 소프트웨어학과
김연아	멀티미디어학과
강승범	경영 정보학과

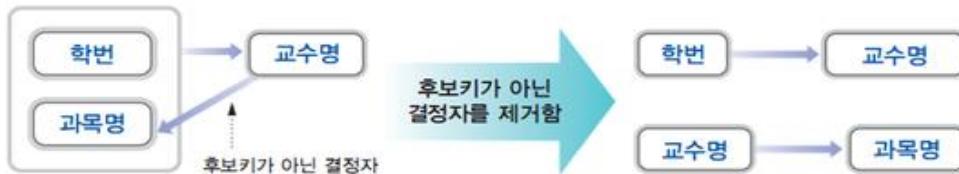
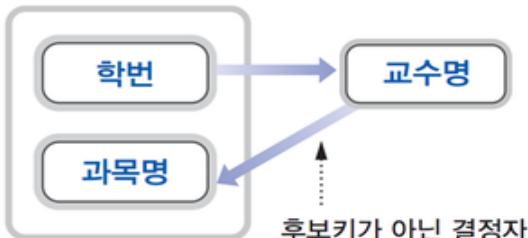
# [정규화]보이스/코드 정규화(BCNF)

## 1. 결정자함수 종속성의 제거, 보이스코드 정규화의 개요

- 복잡한 식별자 관계에 의해 발생하는 문제를 해결하기 위해서 제3정규형을 보완한 것
- 릴레이션 R이 제3정규형을 만족하고, 모든 결정자가 후보키일 경우 수행

수 강

학 번	과목명	교수명
100	전자계산기구조	이문세
100	데이터베이스	김연아
100	운영체제	강승범
101	데이터베이스	김연아
101	운영체제	전혜영



수강(결정자가 키가 아닌 속성인 릴레이션)

학 번	과목명	교수명
100	전자계산기구조	이문세
100	데이터베이스	김연아
100	운영체제	강승범
101	데이터베이스	김연아
101	운영체제	전혜영

후보키가 아닌 결정자를 제거함

지도교수(모든 결정자가 후보키인 BCNF 릴레이션)

학 번	교수명
100	이문세
100	김연아
100	강승범
101	김연아
101	전혜영

교수(모든 결정자가 후보키인 BCNF 릴레이션)

교수명	과목명
이문세	전자계산기구조
김연아	데이터베이스
강승범	운영체제
전혜영	운영체제

# [정규화]4차 정규화

전제 조건

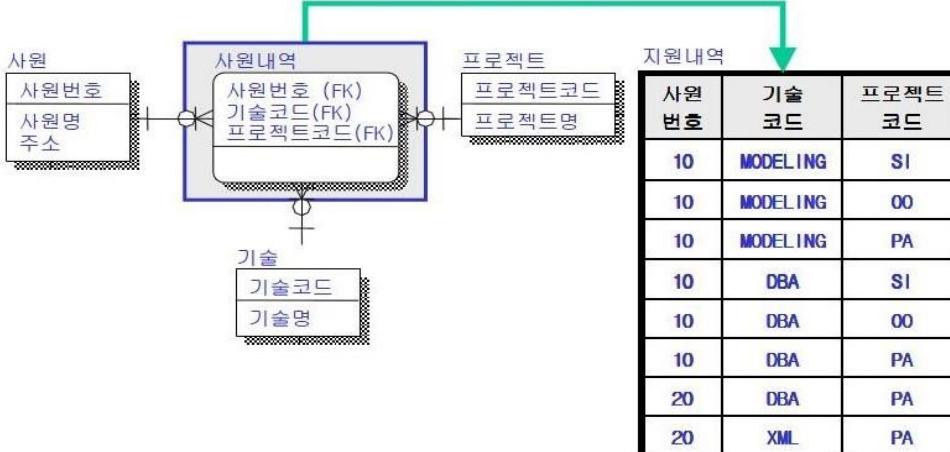
## 1. 다치종속을 제거, 4차 정규화의 개요

- 하나의 릴레이션에 두 개 이상의 다치 종속(MVD; Multi-Valued Dependency)이 발생, 이를 제거하는 과정

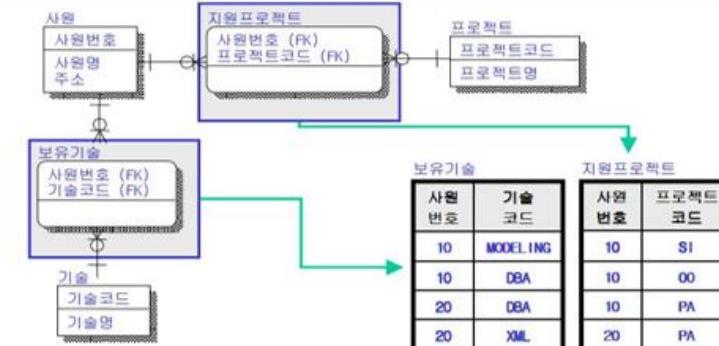
## 2. 4차 정규화의 전제조건

- PK에서 발생함
- 독립된 속성간의 의미적인 연관성이 있어야 함
- A,B,C가 있을 때 A-B, A-C는 연관성이 있으나 B-C는 연관성이 없는데도 불구하고 하나의 테이블에 PK로 사용이 될 때 발생**이 됨

4차 정규화 - 대상



발생원인	해결방법
어떤 종속적 관계가 하나의 테이블 안에 두 개 이상으로 존재	상호관계가 있는 속성끼리 분리시킴(4차 정규화) 수행



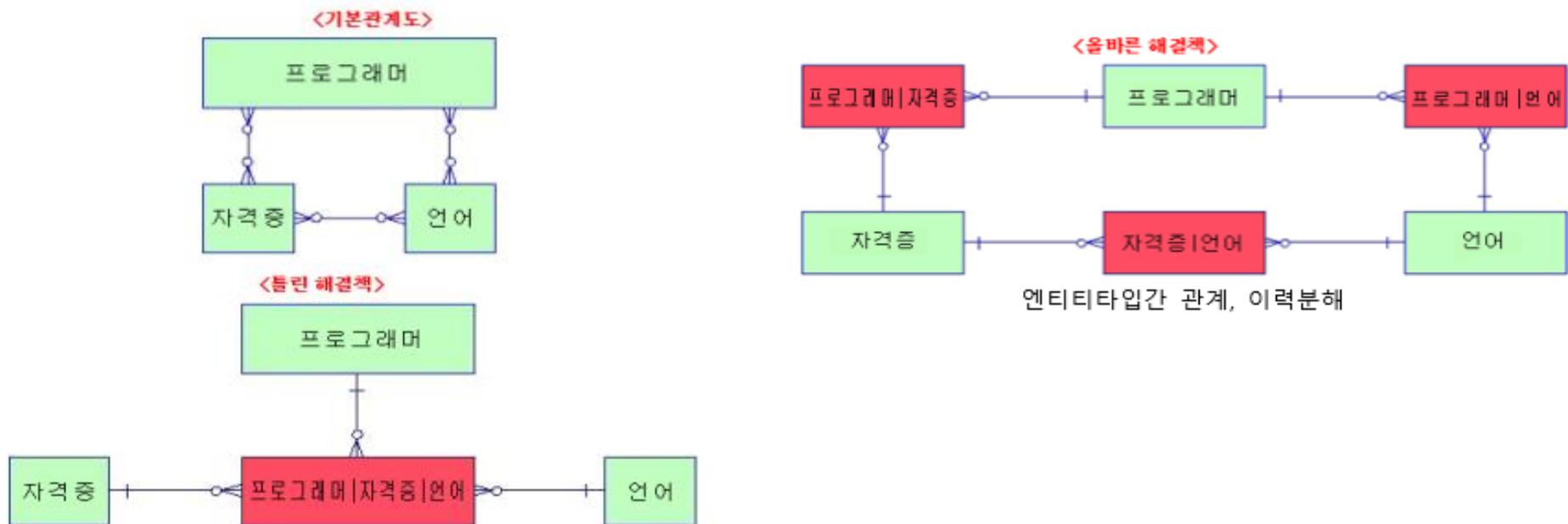
# [정규화]5차 정규화

전체 조건

## 1. 조인종속성 제거, 5차 정규화의 개요

### 2. 5차 정규화의 전제조건

- PK에서 발생함
- 독립된 속성간의 조인 연관성이 있어야 함
- 관계 엔터티(Associative Relation)
- A,B,C가 있을 때 A-B, A-C, B-C는 연관성이 있으나, A-B-C는 연관성이 없는데도 불구하고 하나의 테이블에 PK로 사용이 될 때 발생이 됨





# 연결함정 (Connection Trap)

Fan, Chasm

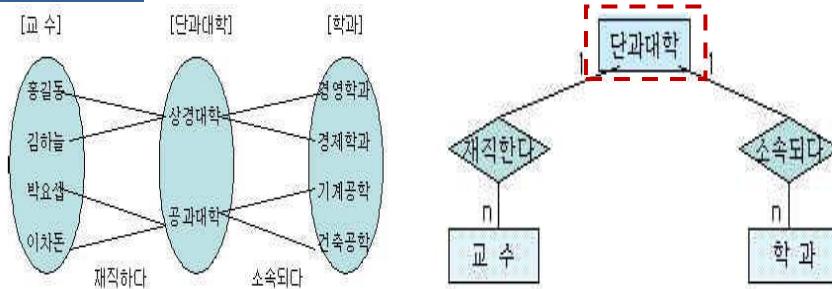
## 1. 엔터티간 잘못된 관계설정, 연결함정

- ER모델 설계 중에서 외형상, 모든 튜플의 관계가 연결된 것 같지만 실제 정확한 관계가 설정되어 있지 않아 정보를 못찾아 가는 경우가 발생되는 관계 (비즈니스 분석을 명확하게 하지 못해서 발생)

### 부채꼴 함정 (Fan Trap)

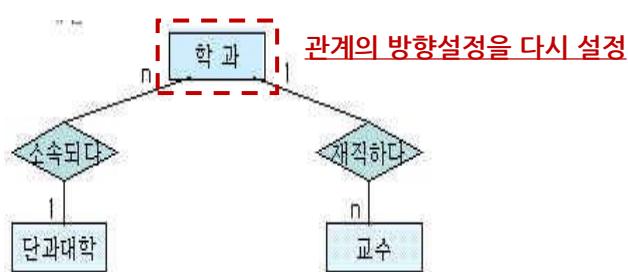
- 관계성이 모호한 경우
- 개체 집합(Entity Set) 사이에 관계성 집합(Relation Set)이 정의되어 있기는 하지만 관계성 예시가 모호한 경우

#### [사례]



- 어느 교수가 어느 학과에 재직하고 있는지 알고자 할 경우 알 수 있는 방법이 없음

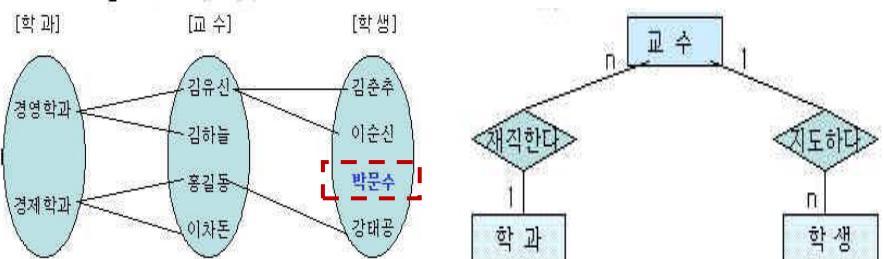
#### [해결방법]



### 균열 함정 (Chasm Trap)

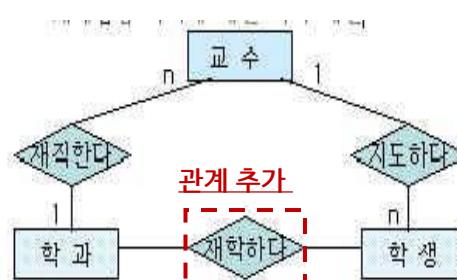
- 관계성이 존재하지 않는 경우 (일부 관계의 누락)
- 개체 집합(Entity Set) 사이에 관계성 집합(Relation Set)이 정의되어 있기는 하지만 일부 개체집합과 개체집합 사이에 관계성이 존재하지 않는 경우

#### [사례]



- 학생이 지도교수를 할당 받지 못한, 즉, 복학생인 경우 어느 학과에 속하는지 알 수 있는 방법이 없음

#### [해결방법]



# 반정규화 (De-normalization)

조회성능 향상

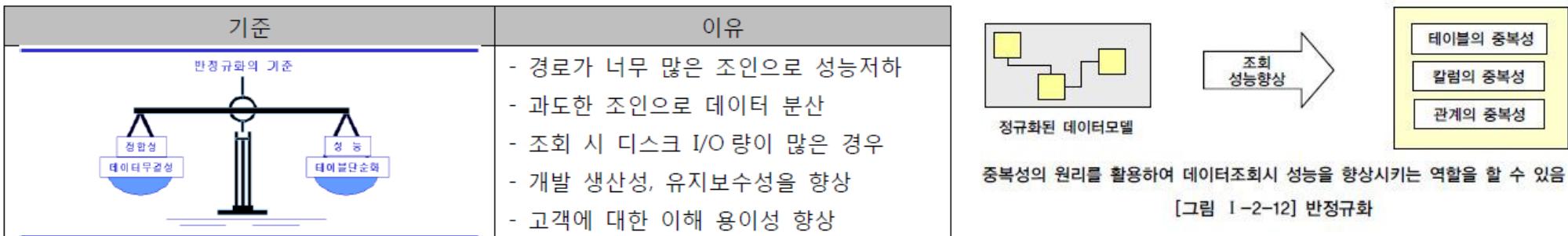
절차

테이블, 컬럼, 관계

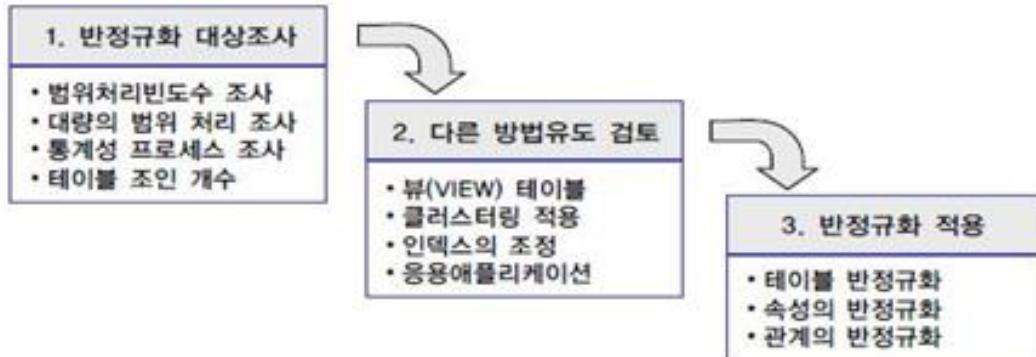
## 1. 조회 성능 향상을 위한 데이터 중복의 허용, 반정규화 (De-normalization)

- 적정 수준의 정규화 이후, 질의 성능 향상 및 개발/운영의 단순화를 위해 일부의 데이터에 중복을 허용하는 정규화의 역작업 (Trade Off)

## 2. 반정규화 기준 및 이유



## 3. 반정규화 절차



대분류	구분
<b>테이블 반정규화</b>	테이블 병합 분할(수평,수직) 추가(중복,통계,이력,부분 테이블 추가)
<b>컬럼의 반정규화</b>	중복컬럼 추가 파생컬럼 추가 이력테이블컬럼 추가 PK에 의한 컬럼 추가 응용시스템 오동작을 위한 컬럼 추가
<b>관계의 반정규화</b>	중복관계 추가

- 반정규화(역정규화, denormalization) : 이전에 정규화된 데이터베이스에서 성능을 개선하기 위해 정규화 역작업
- 비정규화(unnormalized form) : 정규화 이전의 상태

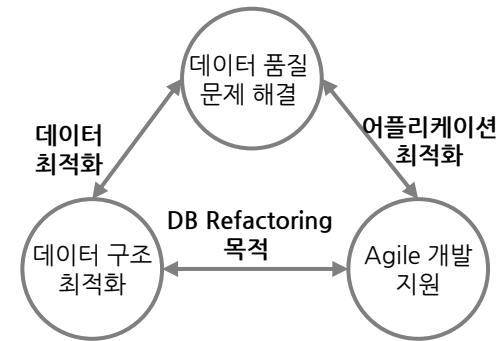
# DB Refactoring

## 1. DB 최적화를 위한 스키마 변경, DB Refactoring의 개요

- 데이터베이스의 의미(정보)에 대한 아무런 변화 없이 디자인을 개선하는 작업
- DB의 복잡한 설계 문제를 개선하여 어플리케이션 연동의 효율성을 제공하기 위해 수행
- DB Refactoring은 동일 정보에 대한 유지, Code Refactoring은 기능에 대한 유지가 주요 차이

## 2. 대표적인 DB Smell 종류

구분	상세설명
Multi-purpose column	<ul style="list-style-type: none"><li><u>단일 컬럼이</u> 의미론적으로 <u>다양한 목적으로 사용</u>하고 있을 경우</li><li>다목적 컬럼은 해당 컬럼만으로는 사용자에게 올바른 정보를 주기가 어렵고 부가적인 컬럼을 확인해야 함</li></ul>
Multi-purpose table	<ul style="list-style-type: none"><li><u>단일 테이블이 여러 유형의 엔티티를 저장</u>하는 사용하고 있다면 설계 결함 오류가 있음</li><li>예를 들어, 고객 테이블이 고객 정보가 아닌, 기업정보에 대한 내역까지 추가적으로 저장하고 있다면 하나의 목적으로 사용된다고 볼 수 없음</li></ul>
Redundant data	<ul style="list-style-type: none"><li><u>중복데이터는</u> DB Smell의 가장 대표적인 사례</li><li>여러 테이블, 여러 컬럼에 중복된 데이터는 데이터의 동기화, 데이터 품질에 심각한 영향을 줌</li><li>데이터베이스의 일관성을 저해하며, 사용자에게 잘못된 정보를 줄 수 있음</li></ul>
Tables with many Columns	<ul style="list-style-type: none"><li><u>너무 많은 컬럼을 가지고 있는 테이블</u>은 검색, 삽입, 삭제 속도에 심각한 영향을 미치며 Big Table을 양산할 수 있음</li><li>컬럼에 대한 의미를 분석하여 정규화시키며 테이블 분할을 고려함</li></ul>
Tables with many rows	<ul style="list-style-type: none"><li>대용량 테이블은 보통 수백만, <u>수천만 건의 데이터를 가지고 있는 테이블</u></li><li>대형 테이블은 보통 정보계 테이블(Fact Table)에 많이 존재하는데, 성능지표에 근거하여 테이블의 수직, 수평 분할을 고려함</li></ul>
"Smart" columns	<ul style="list-style-type: none"><li>A “smart column” is one in which different positions within the data represent different concepts</li><li>예, 주민등록번호 (앞자리는 생년월일, 뒷자리는 성별, 주소 등등)</li><li><u>스마트 컬럼은 하나의 컬럼에 대해 다기능적 목적을 수행</u>할 수 있으나, DB 구조를 복잡하게 하며 어플리케이션의 복잡도를 증가시킴</li></ul>
Fear of change	<ul style="list-style-type: none"><li>데이터베이스를 변화해야 하는 필요성은 이해당사자(DBA, Modeler, Application개발자 등)들이 모두 느끼고 있지만, 변화에 대해 소극적인 태도와 변화에 대한 두려움이 가장 나쁜 냄새임</li></ul>



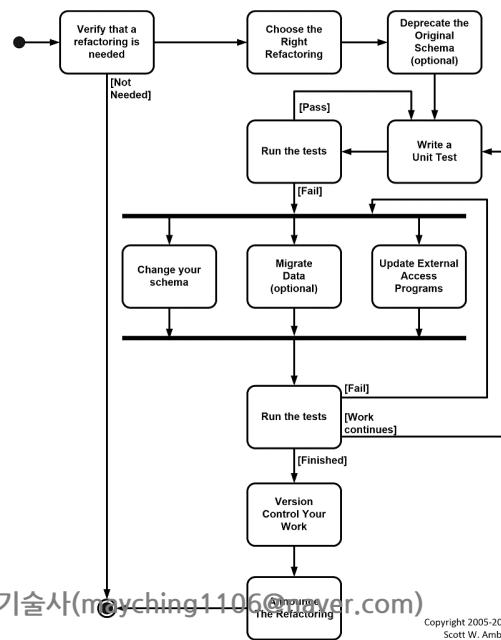
# DB Refactoring

## 3. DB Refactoring 기법

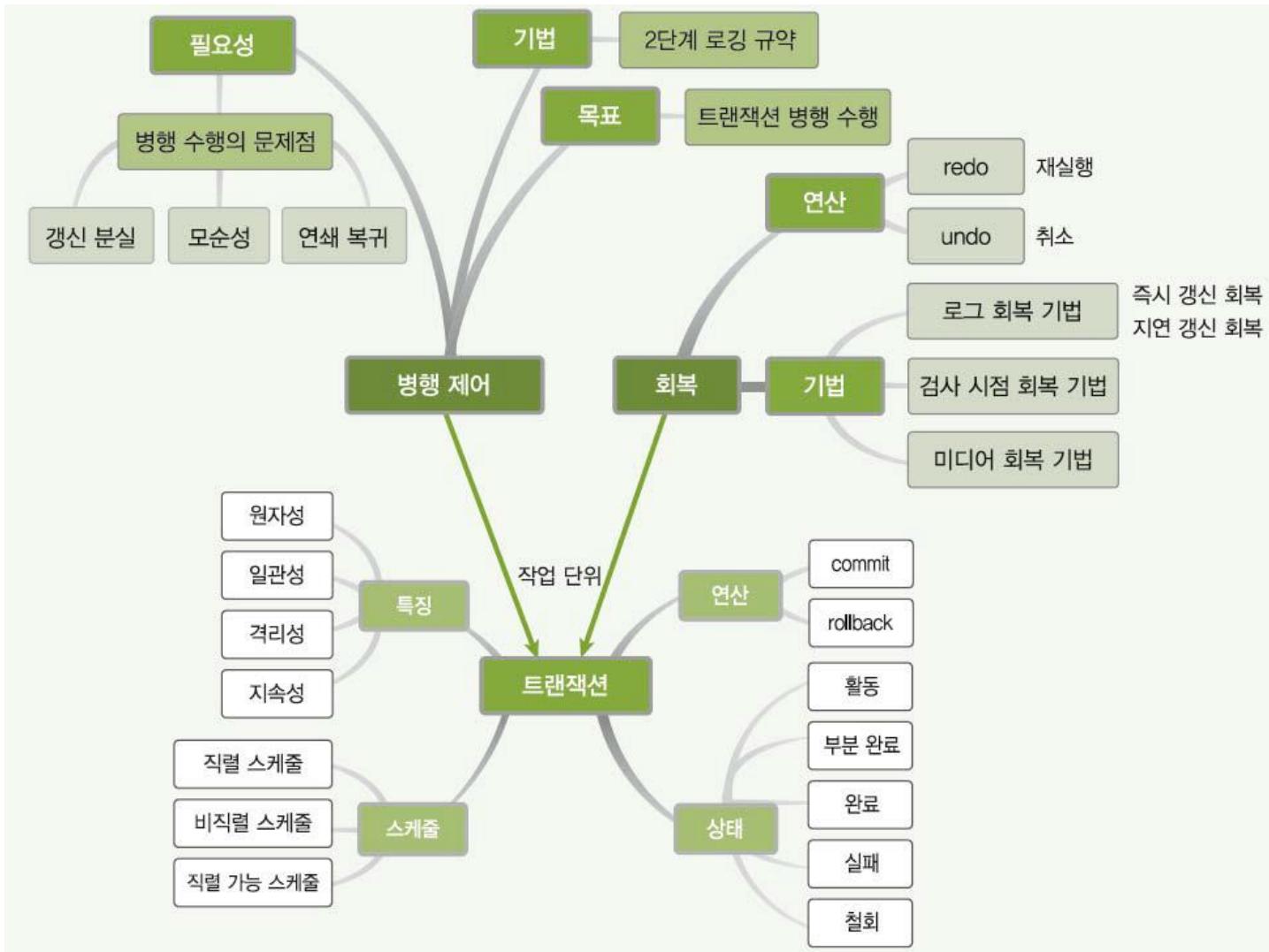
유형	설명	주요 기법
구조 리팩토링	-Structual Refactorings : 데이터베이스 스키마의 테이블 구조를 변경함.	- Merge : table, column -Drop : table, view
데이터 품질 리팩토링	-Data Quality Refactoring : 데이터베이스에 저장되어 있는 값의 일관성 및 사용성 개선	- Standard codes -Default 값이용
참조 무결성 리팩토링	-Referential integrity : 참조된 컬럼에 대한 무결성 보장을 위한 제약조건 추가	- 외래키 추가, 계산 런에 트리거를 적용, Constraint 적용
아키텍처 리팩토링	-Architectural Refactoring : 외부 프로그램이 데이터베이스와 상호 작용하는 전반적인 방법 개선	Encapsulate Table With View, Migrate Method to Database
기능 리팩토링	Method Refactoring : 저장 프로시저의 품질, 저장 기능, 트리거를 개선 변경	Consolidate Conditional Expression
변환	Non-refactoring transformation : 새로운 요소를 추가, 수정하여 데이터베이스 스키마 변경	Insert Data, Introduce New Column

## 4. DB Refactoring 절차

- DB Smell에 의한 DB Refactoring 필요요건 확인
- 가장 적합한 Refactoring 유형 및 기법 선택
- 기존 스키마에 대한 분석
- 단위 테스트 작성
- 데이터베이스 스키마 수정
- 원본 데이터에 대한 마이그레이션 수행
- 외부 액세스 프로그램 업데이트
- 데이터 마이그레이션 스크립트 업데이트
- 회귀 테스트 수행
- 리팩토링 완료에 대한 공식화
- 버전 작업 수행



# 트랜잭션



# 트랜잭션 (Transaction)

ACID

## 1. 데이터베이스 논리적인 작업 단위, 트랜잭션

- 한 번에 수행되어야 할 데이터베이스의 일련의 Read와 Write 연산을 수행하는 **논리적인 작업 단위**

## 2. 트랜잭션의 특징 (ACID)



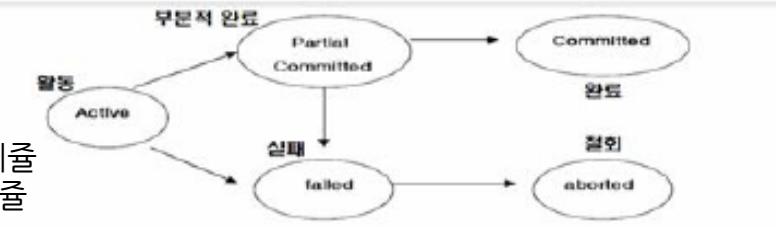
ACID	설명	유형
<b>Atomicity(원자성)</b>	분해가 불가능한 최소 단위, <u>all or nothing</u>	커밋/롤백
<b>Consistency(일관성)</b>	트랜잭션이 실행 성공 후 항상 <u>모순 없는 상태</u>	무결성 제어기
<b>Isolation(고립성)</b>	<u>연산의 중간 결과를 다른 트랜잭션 접근 불가</u>	병행 제어 관리자
<b>Durability(영속성)</b>	완료된 트랜잭션의 결과는 <u>영구(속)적으로 데이터베이스에 저장됨</u>	회복기법

## 3. 트랜잭션 상태

- Active : 초기 상태, 트랜잭션이 실행을 시작하였거나 실행 중인 상태
- 부분적 완료 : 마지막 명령문이 실행된 후 Commit 직전 상태
- 실패 : 정상적인 실행이 더 이상 진행될 수 없는 상태
- 철회(Aborted) : 트랜잭션 실행이 실패하여 취소되고 트랜잭션 시작전 상태로 환원된 상태 (Rollback)
- 완료(committed) : 트랜잭션이 성공적으로 완료된 후 Commit 연산 수행한 상태

## 4. 트랜잭션 처리를 위한 DBMS 구성요소 및 관련 기법

- 스케줄러 (병행수행 제어 관리자) / Locking, TimeStamp
  - 직렬 스케줄 : 각 트랜잭션의 명령들이 끼어들기 불가, 연속적으로 실행되는 스케줄
  - 직렬화 스케줄 : 병행수행을 보장하면서 직렬 스케줄과 동일한 결과를 얻는 스케줄
- 트랜잭션 관리자 : 트랜잭션 완료와 철회 / Transaction begin(end), 커밋, 롤백
- 로그관리 : 데이터베이스 변화(Changes)를 디스크에 로그로 기록 / 백업파일, 복제, 로그
- 회복 관리자 : 로그를 검토하여 데이터베이스를 일관성 있는 상태로 재저장 / 즉시갱신, 지연갱신



# 트랜잭션 (Transaction)

\*참고자료 : <https://gregorio78.tistory.com/307?category=696728>

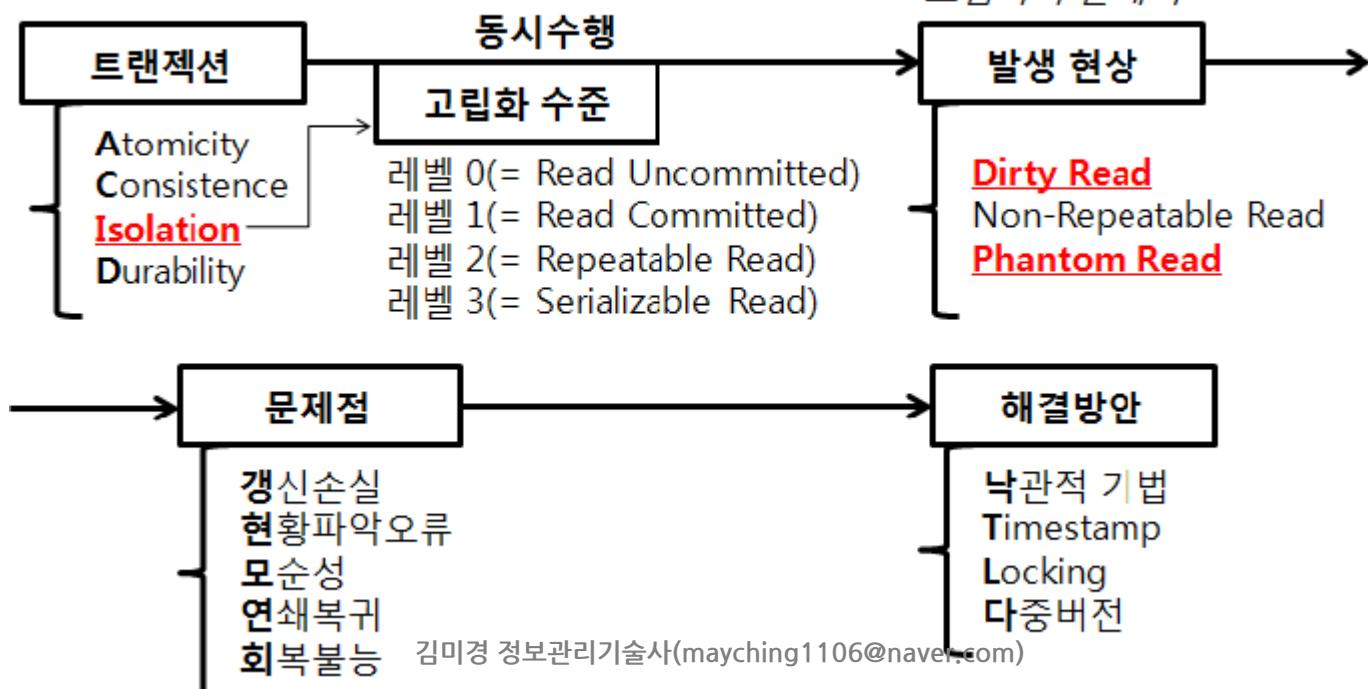
## 7. 트랜잭션 스케줄

- 트랜잭션에 포함되어 있는 연산들을 수행하는 순서를 지정하는 알고리즘

스케줄 유형	설명
직렬 스케줄	<ul style="list-style-type: none"><li>인터리빙 방식을 이용하지 않고 각 트랜잭션별로 <b>연산들을 순차적으로 실행</b>시키는 것</li><li>다른 트랜잭션의 방해를 받지 않고 독립적으로 수행</li></ul>
비직렬 스케줄	<ul style="list-style-type: none"><li>인터리빙 방식을 이용하여 <b>트랜잭션들을 병행해서 수행</b>시키는 것</li><li>하나의 트랜잭션이 완료되기 전에 다른 트랜잭션의 연산이 실행</li><li><b>LDIC 발생 가능</b></li></ul>
직렬 가능 스케줄	<ul style="list-style-type: none"><li>직렬 스케줄과 같이 정확한 결과를 생성하는 비직렬 스케줄</li><li>직렬 가능 스케줄은 인터리빙 방식을 이용하여 여러 트랜잭션을 병행</li><li>수행하면서도, 정확한 결과를 얻을 수 있음 .</li></ul>

## 6. RDBMS의 동시성제어의 원인, 현상, 해결방안 제시

낮은 단계 트랜잭션  
고립화 수준에서



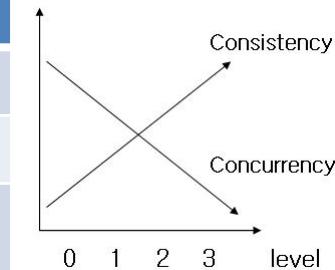
# Isolation Level

## 1. ACID 속성을 보장하기 위한, Isolation Level의 개요

- 트랜잭션 실행 중 중간 연산 결과가 다른 트랜잭션으로 접근 불가하도록 하는 고립성을 유지하기 위한 데이터 접근 허용하는 수준

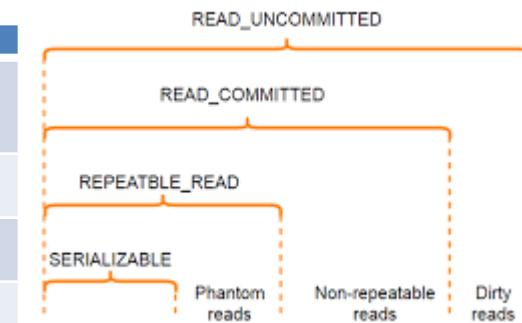
## 2. 직렬성 위반 내용

내용	설명
부정판독 (Dirty Read)	<ul style="list-style-type: none"> <li>트랜잭션 T1이 특정 행(ROW)의 갱신을 수행하고 난 후, T2가 그 행을 검색한 뒤 T1이 취소된다면 트랜잭션 T2는 더 이상 존재하지 않는, 그리고 결코 존재하지 않았던 행을 본 것이 됨</li> </ul>
비반복판독 (Nonrepeatable Read)	<ul style="list-style-type: none"> <li>트랜잭션 T1이 한 행을 검색하고 T2가 그 행을 갱신한 뒤 T1이 동일한 행을 다시 검색한다고 가정하면, 트랜잭션 T1은 “동일” 한 행을 두 번 검색한 것이지만 두 개의 다른 값을 보게 됨</li> </ul>
가상 판독 (Phantom Read)	<ul style="list-style-type: none"> <li>트랜잭션 T1이 특정 조건을 만족하는 모든 행을 검색한다고 하자. 그런 다음 트랜잭션 T2는 동일한 조건을 만족하는 새로운 행을 삽입한다고 할 때 트랜잭션 T1이 검색 요구를 반복한다면, 이전에는 존재하지 않았던 한 행을 보게 됨</li> </ul>



## 3. 트랜잭션 Isolation Level

고립화 수준	설명	직렬성 위반
Read Uncommitted	<ul style="list-style-type: none"> <li>트랜잭션에서 처리중인, 아직 COMMIT되지 않은 데이터를 다른 트랜잭션이 읽는 것을 허용</li> </ul>	<ul style="list-style-type: none"> <li>Dirty Read</li> <li>Nonrepeatable Read</li> <li>Phantom Read</li> </ul>
Read Committed	<ul style="list-style-type: none"> <li>트랜잭션이 COMMIT되어 확정된 데이터만 읽는 것을 허용</li> </ul>	<ul style="list-style-type: none"> <li>Nonrepeatable Read</li> <li>Phantom Read</li> </ul>
Repeatable Read	<ul style="list-style-type: none"> <li>선행 트랜잭션이 읽은 데이터는 트랜잭션이 종료될 때까지 후행 트랜잭션이 갱신, 삭제하는 것 불허(삽입 가능)</li> </ul>	<ul style="list-style-type: none"> <li>Phantom Read</li> </ul>
Serializable	<ul style="list-style-type: none"> <li>선행 트랜잭션이 읽은 데이터를 후행 트랜잭션이 갱신하거나 삭제하지 못할 뿐 아니라, 중간에 새로운 레코드를 삽입하는 것도 막아줌</li> </ul>	<ul style="list-style-type: none"> <li>모두 발생불가</li> </ul>



## 4. 트랜잭션 격리성 수준과 비일관성 현상

격리성 수준	Dirty Read	Non-Repeatable Read	Phantom Read
Read Uncommitted	가능	가능	가능
Read Committed	불가능	가능	가능
Repeatable Read	불가능	불가능	가능
Serializable Read	불가능	불가능	불가능

- 다중 트랜잭션 환경에서 DBMS가 제공하는 기능을 이용해 동시성을 제어하려면 트랜잭션 시작 전에 명시적으로 Set Transaction 명령어를 수행하여야함
- 예) set transaction isolation level read serializable;

# Phantom Conflict

## 1. 가상 투플에 의한 충돌, Phantom Conflict의 개요

- 두 개 이상의 트랜잭션이 실제 데이터베이스에 저장되어 있는 투플(Tuple)이 아니라 **데이터베이스에 삽입되려고 가상의 투플**, 즉 팬텀 투플(Phantom Tuple)에 의해 **트랜잭션이 충돌되어 일관성 및 무결성이 보장되지 않는 현상**

영향	설명
읽기 수행 시	- 다른 트랜잭션의 삽입 결과로 처음 읽을 때 없었던 행이 다음에 읽을 시 나타날 수 있음
삭제 수행 시	- 다른 트랜잭션의 삭제 동작으로 인해 트랜잭션이 첫 번째 행 범위를 읽을 때 있었던 행이 재수행시 사라질 수 있음

## 2. Phantom Conflict의 발생의 예시 (T1, T2를 포함하는 스케줄S)

T1	T2
SELECT SUM(Sal) FROM PROFESSOR WHERE Dept = 'COMP ENG'	INSERT INTO PROFESSOR(Pno, Pname, Dept, Sal) VALUES('P123', 'LEE', 'COMP ENG', 200)

- DB에 저장되어 있는 투플의 견지에서 볼 때 스케줄S의 T1, T2는 서로 공통으로 접근하려는 투플은 없음
- 그러나 실제로 트랜잭션을 T1 → T2순으로 실행하는 것과 T2 → T1순으로 실행한 결과는 서로 상이
- 충돌이 없는 트랜잭션들의 실행 결과가 순서에 따라 상이한 것은 논리적으로 맞지 않음
- 실제로 DB에 저장되어 있는 투플이 아니라 가상의 팬텀 투플에 의해 트랜잭션이 충돌하기 때문

## 3. Phantom Conflict의 해결방법

해결방법	설명
Locking 단위 확대	- Locking의 단위를 투플이 아니라 릴레이션으로 확대 - T1은 릴레이션 PROFESSOR에 S-lock(공용 lock), T2는 X-lock(전용 lock)으로 실제 데이터에 대한 충돌발생 - 로킹단위가 커짐으로 병행성 감소
Index Locking	- 트랜잭션이 한 릴레이션에 투플을 삽입하려면 그 릴레이션을 기초로 만들어진 모든 인덱스를 갱신해야 함. 이점에 착안하여 Locking 규약을 인덱스에 사용 - 릴레이션에 Index가 만들어져 있는 이점을 이용하여 Phantom Conflict를 이 인덱스 버킷(index bucket)에 대한 충돌로 해결

## 4. 팬텀 충돌 현상을 방지하기 위한 트랜잭션의 isolation 수준

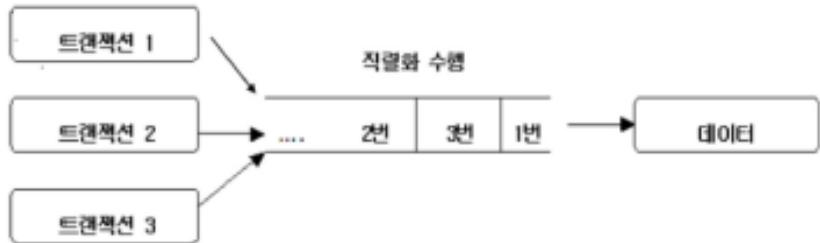
- Phantom Conflict를 방지하기 위하여 isolation Level은 Serializable 단계로 이상으로 설정

# 동시성 제어 기법

LDIC | 2PL | 타임스탬프 | 낙관적제어기법

## 1. 트랜잭션의 직렬성 보장, 동시성 제어

- 다중 사용자 환경을 지원하는 데이터 베이스 시스템에서 **여러 트랜잭션들이 성공적으로 동시에 실행될 수 있도록 직렬화 수행 보장 기법(데이터 무결성 및 일관성 보장)**
- 다중 사용자 환경에서 둘 이상의 트랜잭션이 동시에 수행될 때, 일관성을 해치지 않도록 **트랜잭션의 데이터 접근 제어**



## 2. 동시성 제어를 하지 않은 경우 발생하는 문제점 (LDIC)

문제점	설명
Lost Update (갱신 손실)	<ul style="list-style-type: none"> <li><b>이전 트랜잭션 값을</b> 트랜잭션 종료전에 <b>나중 트랜잭션이 덮어쓰는 현상</b></li> </ul>
Dirty Read(현황파악오류)	<ul style="list-style-type: none"> <li><b>트랜잭션의 중간 수행결과를 다른 트랜잭션이 참조</b> 함으로써 발생하는 오류</li> <li>다른 트랜잭션이 변경 중인 데이터를 읽었는데, 그 트랜잭션이 최종 롤백됨으로써 현재 트랜잭션이 비일관성 (inconsistency) 상태에 놓이는 것</li> </ul>
Inconsistency(모순성)	<ul style="list-style-type: none"> <li>두 트랙잭션이 동시에 실행할 때 <b>DB가 일관성이 없는 상태</b>로 남는 문제</li> </ul>
Cascading Rollback(연쇄복귀)	<ul style="list-style-type: none"> <li>특정 트랜잭션이 처리취소시, 다른 트랜잭션이 처리한 <b>부분 취소 불가능</b></li> </ul>

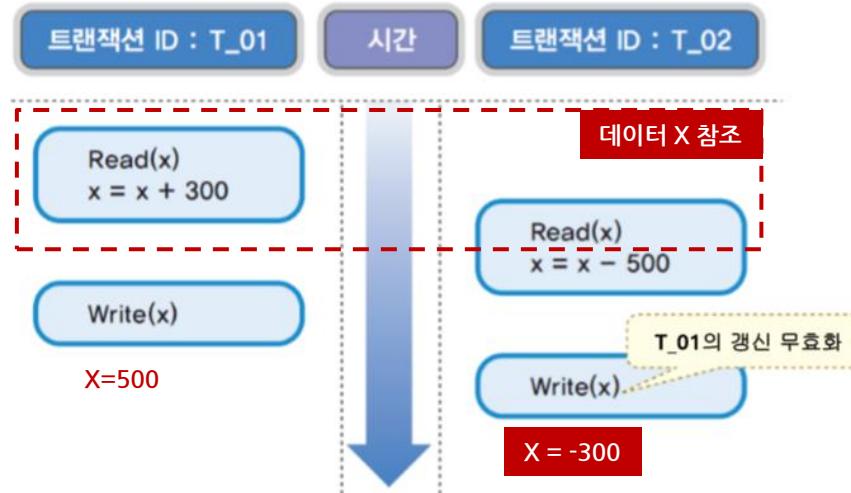
## 3. 동시성 제어 기법

기법	설명
2PL	<ul style="list-style-type: none"> <li>트랜잭션이 사용하는 자원에 대하여 상호 배제 기능을 제공하여 <b>잠금을 설정한 트랜잭션이 해제할 때까지 자원을 독점적으로 사용</b>하는 기법</li> </ul>
Timestamp	<ul style="list-style-type: none"> <li>트랜잭션을 식별하기 위해 DBMS가 부여하는 유일한 식별자인 타임 스탬프를 지정하여 <b>트랜잭션간의 순서를 미리 선택, 동시성 제어</b>하는 기법</li> </ul>
낙관적 제어	<ul style="list-style-type: none"> <li>트랜잭션 수행 동안은 어떠한 검사도 하지 않고, <b>트랜잭션 종료 시 일괄적으로 직렬성 위반을 검사</b>하는 동시성 제어 기법</li> </ul>

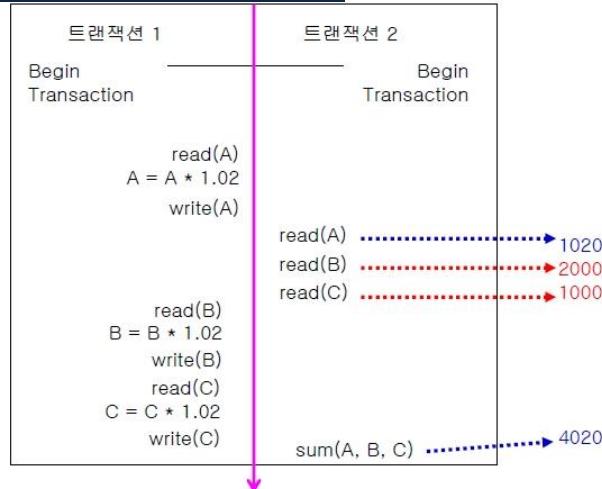
# 동시성 제어 기법 - 미적용시 문제점

## [갱신 손실 (Lost Update)]

X=200일 경우

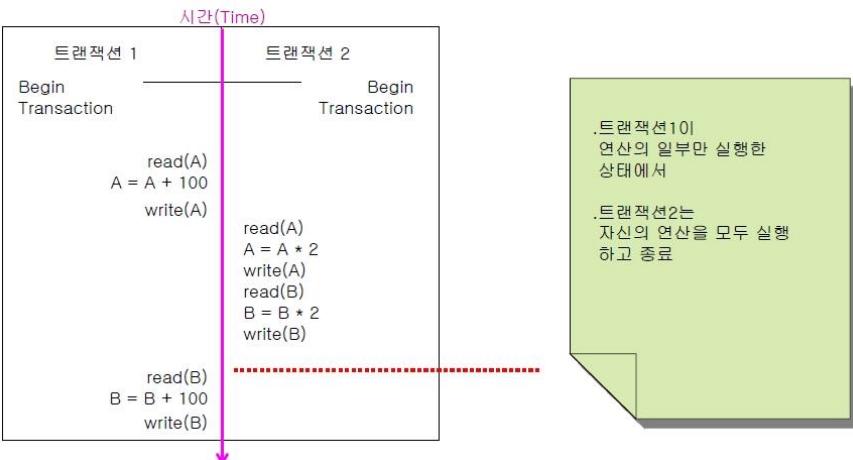


## 현황파악오류 (Dirty Read)

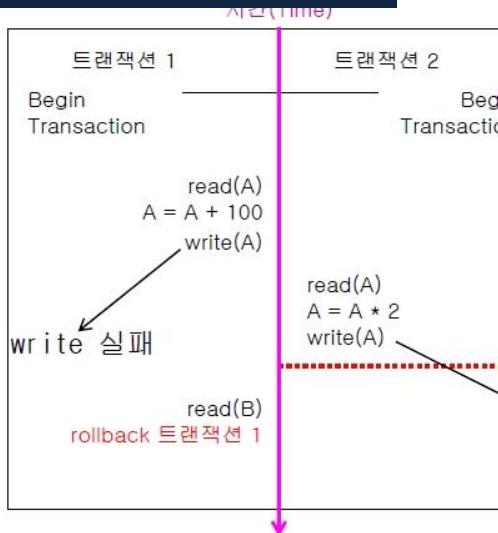


.A는 이자금 계산 후  
값을  
.B와 C는 계산 이전 값을  
읽어서  
.합이 40800이 아닌  
의미 없는 40200이 됨

## 모순성 (Inconsistency)



## 연쇄복귀 (Cascading Rollback)



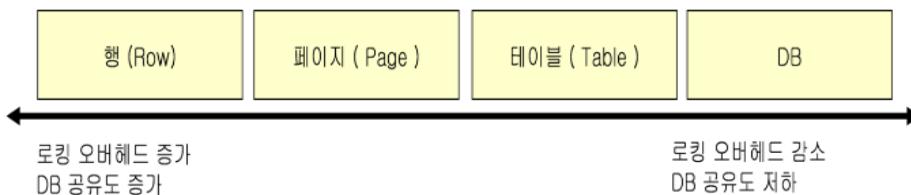
.트랜잭션1이  
복귀를 하지만  
.트랜잭션2는  
성공완료 이후이므로  
복귀를 하지 않음

# [동시성 제어 기법] Lock

## 1. 트랜잭션 동시성제어를 위한 잠금 기법의 개요

- 트랜잭션이 사용하는 자원(데이터 항목)에 대하여 **상호 배제(Mutual Exclusive)** 기능을 제공하는 기법

## 2. 잠금(Locking) 단위



## 3. 잠금(Locking)의 종류

종류	주요 개념
공유 Lock (Shared Lock)	<ul style="list-style-type: none"><li>공유 잠금한 트랜잭션은 데이터 항목에 대해 <b>읽기(read) 만 가능</b></li><li><b>다른 트랜잭션도 읽기(read) 만을 실행</b>할 수 있는 형태</li></ul>
전용 Lock (Exclusive Lock)	<ul style="list-style-type: none"><li>전용 잠금한 트랜잭션은 데이터 항목에 대해서 <b>읽기(read) 와 기록(write)가 모두 가능</b></li><li><b>다른 트랜잭션은 읽기(read)와 기록(write) 모두 할 수 없는 형태</b></li></ul>

## 3. 잠금(Locking) 규칙

조건	예시
- read(x) 연산을 실행하기 전에 S-lock(x)나 X-lock(x)를 실행	S-lock(A) Read(A)
- write(x) 연산을 실행하기 전에 X-lock(x)를 실행해야 함	Unlock(A) x-lock(A)
- read(x)나 write(x) 연산을 실행이 끝나면 unlock(x) 실행	read(B) B=B+A
- S-lock(x)나 X-lock(x)를 실행한 경우만 unlock(x) 실행가능	Write(B) Unlock(B)

# [동시성 제어 기법] 2PL (Phase Locking)

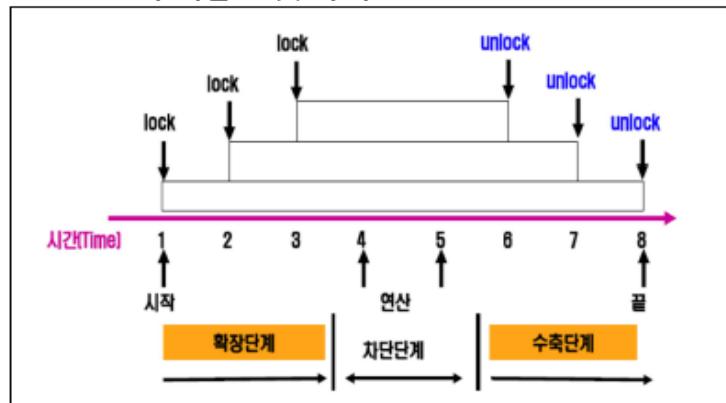
확장, 수축

로깅기법

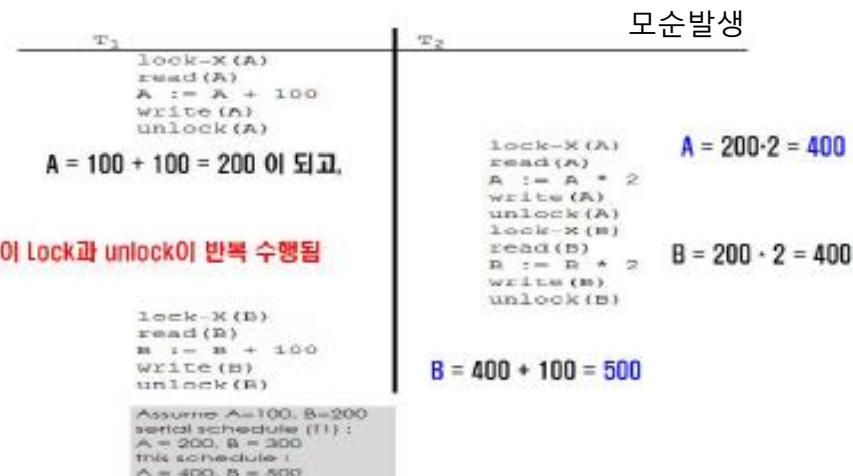
## 1. 직렬성 보장을 통해 동시성을 제어하는 2PL 기법

- 모든 트랜잭션들이 Lock과 Unlock 연산을 **확장단계와 수축단계로 구분**하여, 사용하는 자원(데이터 항목)에 대하여 상호 배제 기능을 제공하는 기법

## 2. 2PL의 개념도 및 사례



<b>확장단계</b>	트랜잭션은 lock만 수행가능하고, unlock은 수행 불가능 단계
<b>수축단계</b>	트랜잭션은 unlock만 수행가능하고, lock은 수행 불가능 단계



## 3. 2PL의 문제점 및 해결 방법

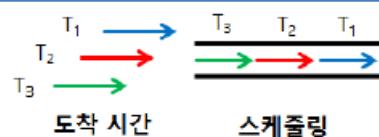
- 교착상태, Cascading rollback 발생 → 교착상태 예방과 탐지로 해결 함

## 4. 기본적인 2단계 로킹 기법의 문제점을 피하기 위한 로킹 기법

기법	설명
<u>Strict 2PLP</u> (Strict 2 Phase Locking Protocol)	<ul style="list-style-type: none"> <li>모든 <b>독점 lock(Lock-X)</b>는 그 Transaction이 완료될 때까지 unlock 하지 않고 그대로 유지</li> </ul>
<u>Rigorous 2PLP</u> (Rigorous 2 Phase Locking Protocol)	<ul style="list-style-type: none"> <li>모든 <b>lock</b>는 그 Transaction이 완료될 때까지 unlock 하지 않음.</li> </ul>
<u>Static (Conservative) 2PL</u>	<ul style="list-style-type: none"> <li><b>Transaction 수행 전부터 그 Transaction의 읽기 집합과 쓰기집합을 미리 선언</b>하여 그 Transaction이 접근하려는 모든 항목들에 Lock을 획득 (트랜잭션 전 미리 예상해서 Lock 획득하는 방식 → 실현 불가능한 방법)</li> </ul>

# [동시성 제어 기법] 타임스탬프 기법

## 1. 동시성 제어를 위한 타임스탬프 기법의 개요



- 트랜잭션을 식별하기 위해 DBMS가 부여하는 유일한 식별자인 **타임 스탬프를 지정하여 트랜잭션간의 순서를 미리 선택, 동시성 제어**하는 기법
- 타임스탬프 순서 기법에서는 **스케줄이 트랜잭션들의 타임스탬프의 순서에 해당하는 특정 직렬 순서와 동치**

## 2. Time Stamp의 종류

- 노리적 계수기 (counter)** : 트랜잭션이 발생할 때마다 **카운터를 하나씩 증가**시켜 타임스탬프 값으로 부여 (1씩 증가)
- 시스템 시계(system clock) 기법** : **시스템 클록**의 값을 타임스탬프 값으로 부여 함 (시스템 시간정보 활용)

## 3. 타임스탬프 순서 알고리즘

알고리즘	설명
<u>read_TS(X)</u>	<ul style="list-style-type: none"> <li>항목 X의 읽기 타임스탬프(read timestamp)로서 항목 X를 성공적으로 읽은 트랜잭션들의 타임스탬프 중 가장 큰 값이다. 즉, <math>\text{read\_TS}(X) = \text{TS}(T)</math>이고 여기서 T는 X를 성공적으로 읽은 가장 최근의(youngest) 트랜잭션</li> </ul>
<u>write_TS(X)</u>	<ul style="list-style-type: none"> <li>항목 X의 쓰기 타임스탬프(write timestamp)로서 항목 X를 성공적으로 기록한 트랜잭션들의 타임스탬프 중 가장 큰 값이다. 즉, <math>\text{write\_TS}(X) = \text{TS}(T)</math>이고 여기서 T는 X를 성공적으로 기록한 가장 최근의 트랜잭션</li> </ul>

## 4. 기본적 타임스탬프 순서(Basic Timestamp Ordering)

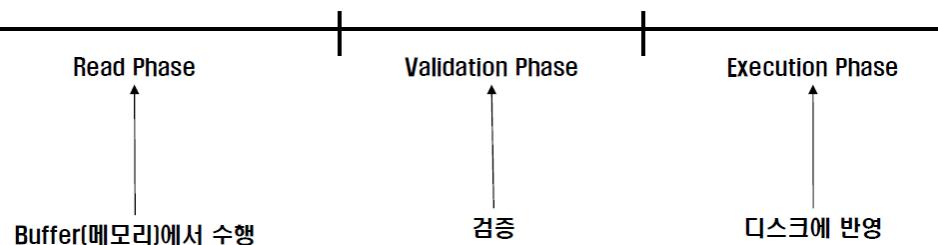
케이스	설명
1) $\text{read\_TS}(X) > \text{TS}(T)$ 또는 $\text{write\_TS}(X) > \text{TS}(T)$	<ul style="list-style-type: none"> <li>T를 철회하고 복귀시키고 그 연산을 거부(reject)</li> </ul>
2) 1)의 조건이 발생하지 않으면	<ul style="list-style-type: none"> <li>T는 <math>\text{write\_item}(X)</math> 연산을 수행하고 <math>\text{write\_TS}(X)</math>를 <math>\text{TS}(T)</math>로 설정</li> </ul>
3) $\text{write\_TS}(X) > \text{TS}(T)$	<ul style="list-style-type: none"> <li>T를 철회하고 복귀시키고 그 연산을 거부</li> </ul>
4) $\text{write\_TS}(X) \leq \text{TS}(T)$	<ul style="list-style-type: none"> <li>T의 <math>\text{read\_item}(X)</math> 연산을 수행하고 <math>\text{read\_TS}(X)</math>를 <math>\text{TS}(T)</math>와 현재의 <math>\text{read\_TS}(X)</math> 중 큰 값으로 설정</li> </ul>

# [동시성 제어 기법] 낙관적 병행 검증(제어)기법

## 1. 낙관적(Validation) 검증기법

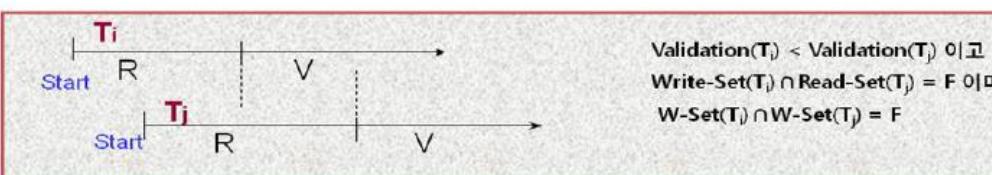
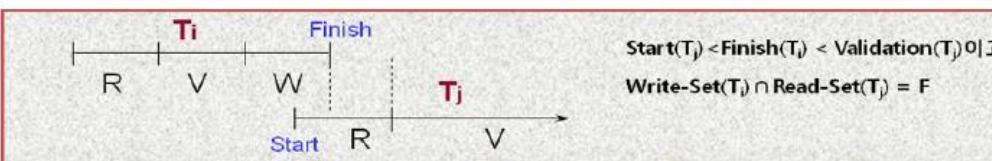
- 트랜잭션 수행 동안은 어떠한 검사도 하지 않고, **트랜잭션 종료 시 일괄적으로 검사**하는 기법
- 트랜잭션을 실행하는 동안 모든 갱신은 지역 사본에만 반영**되고 트랜잭션 종료 시 확인 단계를 통해 직렬 가능성에 위반되지 않으면 실행하고 위반되면 복귀하는 기법

## 2. 낙관적 병행 제어기법의 처리단계



처리단계	설명
판독단계 (Read phase; R)	- 트랜잭션의 모든 갱신은 사본에 대해서만 수행하고 실제 데이터베이스에 대해서는 수행하지 않음
확인단계 (Validation phase; V)	- 판독 단계에서 사본에 반영된 트랜잭션의 실행 결과를 데이터베이스에 반영 전 직렬 가능성의 위반여부를 확인
기록단계 (Write phase; W)	- 확인 단계를 통과하면 트랜잭션의 실행 결과를 데이터베이스에 반영 - 확인 단계를 실패하면 실행 결과는 취소되고 트랜잭션은 복귀

## 3. 확인 단계의 확인 검사 조건



- 3가지 조건 중 하나만 만족하면 트랜잭션들 간에 간섭이 없는 것으로 보고 확인을 성공

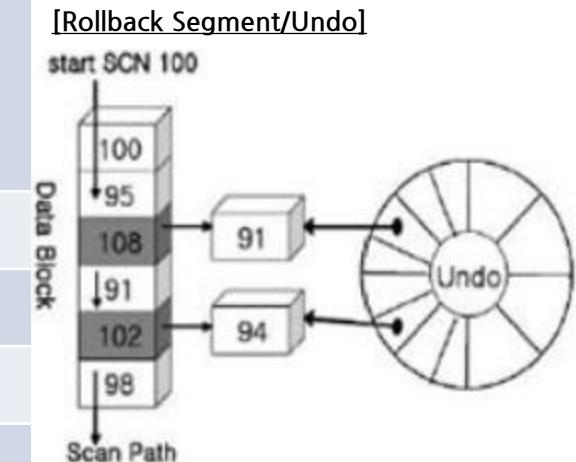
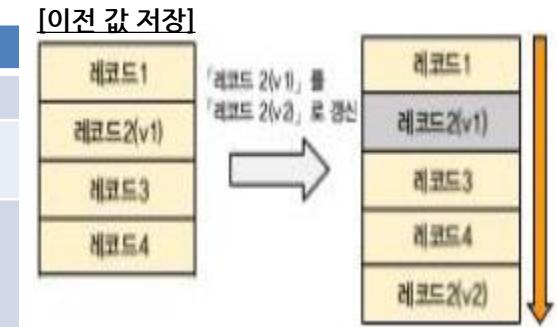
# [동시성 제어 기법] 다중버전 병행제어 기법

\*참고자료 : <https://gregorio78.tistory.com/301>

## 1. 다중버전 병행제어 기법(MultiVersion Concurrency Control)의 정의

- 하나의 데이터 아이템에 대해 여러 버전을 유지하는 기법**으로, 트랜잭션이 한 데이터 아이템을 접근하려 할 때, 그 트랜잭션의 타임스탬프와 접근하려는 데이터 아이템의 여러 버전의 타임스탬프를 비교하여, 현재 실행하고 있는 스케줄의 직렬가능성이 보장되는 적절한 버전을 선택하여 접근하도록 하는 기법
- 쓰기(Write) 세션이 읽기(Read) 세션을 블로킹(blocking)하지 않고, 읽기 세션이 쓰기 세션을 블로킹하지 않게 서로 다른 세션이 동일 데이터에 접근했을 때 각 세션마다 스냅샷 이미지를 보장해주는 메커니즘

특징	설명
배경	<ul style="list-style-type: none"> <li>기본적인 락킹(Locking) 매커니즘 문제(Unlock이 너무 일찍 수행되면 직렬가능성 보장하지 못함)</li> </ul>
구현	<ul style="list-style-type: none"> <li>트랜잭션의 시작 시간에 따른 데이터의 버전을 읽을 수 있게 함</li> <li>버전 데이터 관리 DBMS 마다 어떻게 구현하느냐에 따라 다르며, 로그파일 혹은 임시 테이블을 이용</li> </ul>
타임스탬프 (Timestamp)	<ul style="list-style-type: none"> <li><b>타임스탬프 순서에 기반하여 각 데이터 항목에 대한 버전 관리.</b></li> <li>각 데이터 항목 버전을 Qi라 할 때 3개의 필드 값 보유</li> <li>Content : 버전 Qi 값</li> <li>W-Timestamp(Qi) : 버전Qi 의 생성 타임스탬프</li> <li>R-Timestamp(Qi) : 버전Qi 의 판독 성공한 트랜잭션 중 Max 타임스탬프</li> <li>다중 버전 타임스탬프 기법</li> <li>Ti 가 Read(Q) 실행 <math>\rightarrow</math> Qi 반환</li> <li>Ti 가 Write(Q) 실행</li> <li>만약 TS(Ti) &lt; R-Timestamp(Qi) 이면 Ti 복구됨</li> <li>만약 TS(Ti) &lt; W-Timestamp(Qi) 이면 Qi 의 내용은 Overwritten</li> <li>그 외의 경우 Q 의 새 버전 생성</li> </ul>
일관성 보장	<ul style="list-style-type: none"> <li>문장수준 읽기 일관성</li> <li>트랜잭션 수준 읽기 일관성 지원하지 않음(Isolation Level 을 Serializable Read로 상향 조정 시 일관성 보장)</li> </ul>
동시성 향상	<ul style="list-style-type: none"> <li>잠금(Locking)을 필요로 하지 않기 때문에 속도향상, Dirty Read 방지</li> <li>데이터 항목 변경이 완료 시 까지 변경사항은 다른 사용자가 볼 수 없음</li> </ul>
스냅샷(Snapshot)	<ul style="list-style-type: none"> <li>기 연산의 직렬가능성을 유지하기 위해 이전 버전을 읽도록 허용.</li> <li>트랜잭션(Transaction) 당 스냅샷(Snapshot) 형태로 이전버전 유지.</li> </ul>
오버헤드	<ul style="list-style-type: none"> <li><b>Undo 블록 I/O, 버전생성과 캐싱(Cache)과 작업에 따른 오버헤드</b></li> </ul>



# 동시성 제어 기법 비교

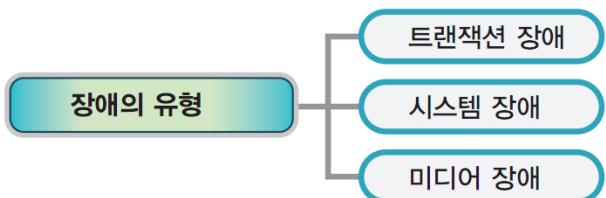
기법	장점	단점
2PL (Phase Locking)	<ul style="list-style-type: none"> <li>알고리즘이 간단하여 구현 용이</li> <li>연쇄 복구(Cascading rollback) 방지</li> </ul>	<ul style="list-style-type: none"> <li>Lock연산 지연에 따른 대기시간 증가</li> <li>교착 상태(Dead Lock) 발생 가능</li> </ul>
Timestamp	<ul style="list-style-type: none"> <li>Dead Lock 발생을 방지</li> <li>트랜잭션 처리시간의 빠름</li> <li>조회 업무 환경에 최적</li> </ul>	<ul style="list-style-type: none"> <li>Rollback 발생 가능성</li> <li>연쇄 복구(Cascading Rollback) 문제</li> </ul>
Validation	<ul style="list-style-type: none"> <li>병행 처리량 높음</li> <li>트랜잭션 대기 시간이 최소화</li> </ul>	<ul style="list-style-type: none"> <li>트랜잭션 철회 시 자원 낭비가 높음</li> <li>트랜잭션 증가하면 시스템의 부하 증가</li> </ul>
Multi-version	<ul style="list-style-type: none"> <li>데이터의 동시성과 일관성 향상</li> <li>대기 시간을 최소화 (읽기 실패 감소)</li> </ul>	<ul style="list-style-type: none"> <li>버전별 주가적인 저장 공간이 필요</li> <li>시스템 오버헤드 증가(Undo 블럭I/O 등)</li> </ul>

구분	Locking(2PL)	Validation	Timestamp
직렬성 (Serializability)	○ (Lock point 순서)	○ (Validation순서)	○ (타임스탬프순서)
교착상태 (Deadlock) 방지	X	○ (기다리지 않고 Rollback)	○ (기다리지 않고 Rollback)
기아(Starvation) 방지	○	X	X
회복/비연쇄 복구가능	X	○ (트랜잭션 완료 후 DB에 반영)	X

# 장애와 회복기법

## 1. 데이터베이스 장애 유형

유형	주요내용
트랜잭션 장애	<ul style="list-style-type: none"><li>트랜잭션간 상호 실행 순서/결과 등에 의한 오류</li><li>논리적 오류: 내부적인 오류로 트랜잭션을 완료할 수 없음.</li><li>시스템 오류: Deadlock 등의 오류 조건으로 활성 트랜잭션을 강제로 종료</li></ul>
시스템 장애	<ul style="list-style-type: none"><li><u>전원, 하드웨어, 소프트웨어 등의 고장</u></li><li><u>시스템 장애</u>로 인해 저장 내용이 영향 없도록 무결성 체크</li></ul>
디스크 장애	<ul style="list-style-type: none"><li><u>디스크 스토리지</u>의 일부 또는 전체가 <u>붕괴되는</u> 경우</li><li>가장 최근의 덤프와 로그를 이용, 덤프 이후에 완결된</li><li>트랜잭션을 재실행(REDO)</li></ul>
사용자 장애	<ul style="list-style-type: none"><li><u>사용자들의</u> 데이터베이스에 대한 이해 부족으로 발생</li><li>DBA 가 데이터베이스 관리를 하다가 <u>발생시키는 실수</u></li></ul>



# 장애와 회복기법

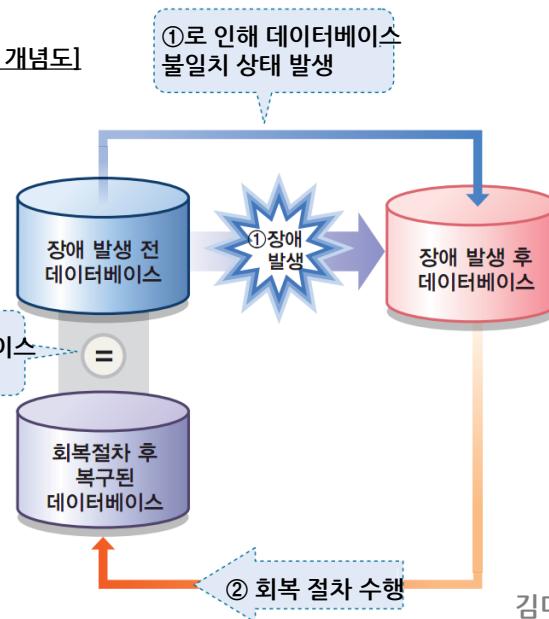
## 1. 데이터의 일관성과 무결성을 위한 장애 회복(Data Recovery)의 개념

- 데이터베이스 운영 도중 예기치 못한 **장애(Failure)**가 발생할 경우 데이터베이스를 장애 발생 이전의 일관된 상태로 복원

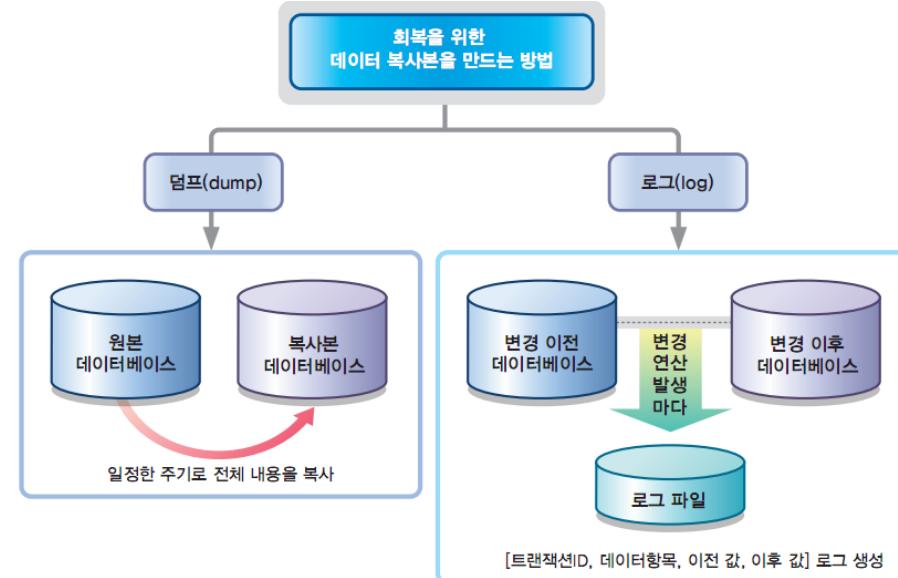
## 2. 데이터베이스 회복을 위한 주요 요소

구분	요소	개념
회복의 기본 원칙 <i>(중복)</i>	데이터	<ul style="list-style-type: none"> <li>데이터의 중복</li> </ul>
	Archive 또는 Dump	<ul style="list-style-type: none"> <li>다른 저장장치로 <b>자료의 복사</b> 및 덤프</li> </ul>
	Log 또는 journal	<ul style="list-style-type: none"> <li>데이터베이스 내용이 변경될 때마다 변경내용을 <b>로그파일에 저장</b></li> <li>갱신된 속성의 과거 값 / 갱신 값을 별도의 파일에 유지</li> <li>온라인로그(디스크), 보관로그(테이프)</li> </ul>
회복을 위한 조치	REDO (Forward Recovery)	<ul style="list-style-type: none"> <li>최근 변경된 내용을 로그파일에 기록하고, 장애발생시 로그파일을 읽어서 <b>재실행</b>하여 데이터베이스 내용을 복원</li> <li>Archive 사본 + log: commit 후의 상태</li> </ul>
	UNDO (Backward Recovery)	<ul style="list-style-type: none"> <li>장애발생시 모든 변경된 내용을 <b>최소</b>함으로 원래의 데이터베이스 상태로 복원</li> <li>Log + Backward 취소 연산: 해당 트랜잭션 수행이전 상태 (트랜잭션 오류가 많음)</li> </ul>
시스템	회복관리기능	<ul style="list-style-type: none"> <li>신뢰성 제공을 위한 DBMS 서브시스템</li> </ul>

[장애 회복 개념도]



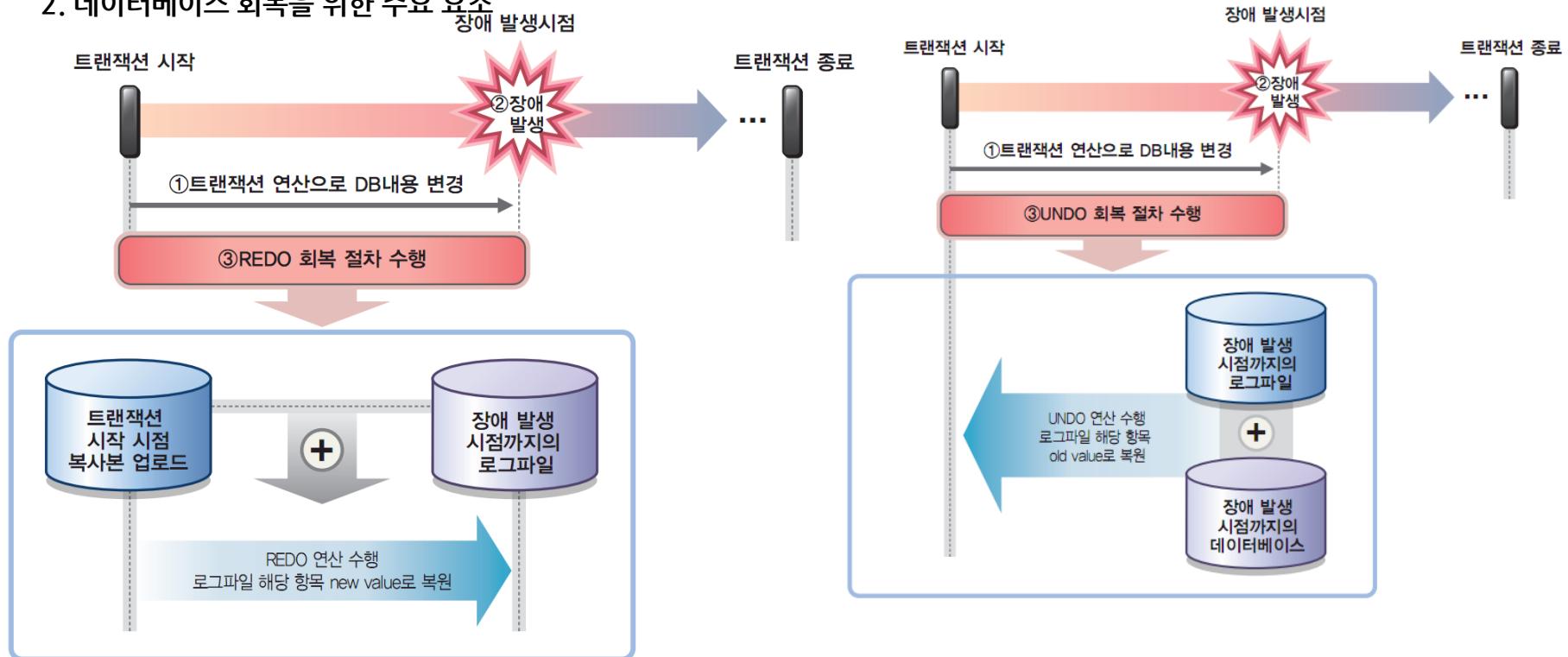
회복을 위한 데이터 복사본을 만드는 방법





# 장애와 회복기법

## 2. 데이터베이스 회복을 위한 주요 요소



## 3. 회복 기법 유형



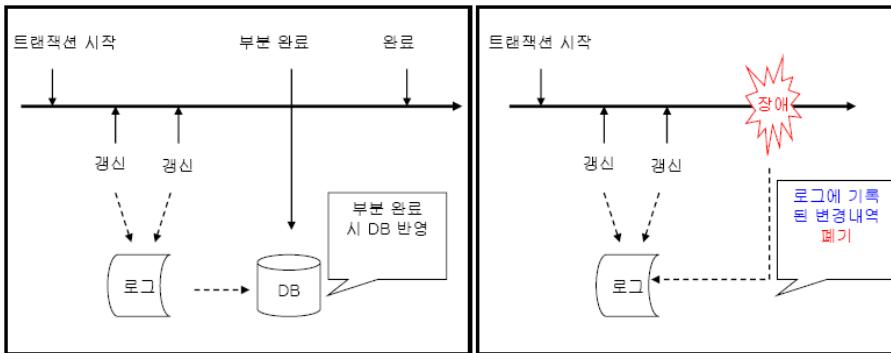
# [장애와 회복기법] 로그기반

즉시(Undo), 지연(Redo)

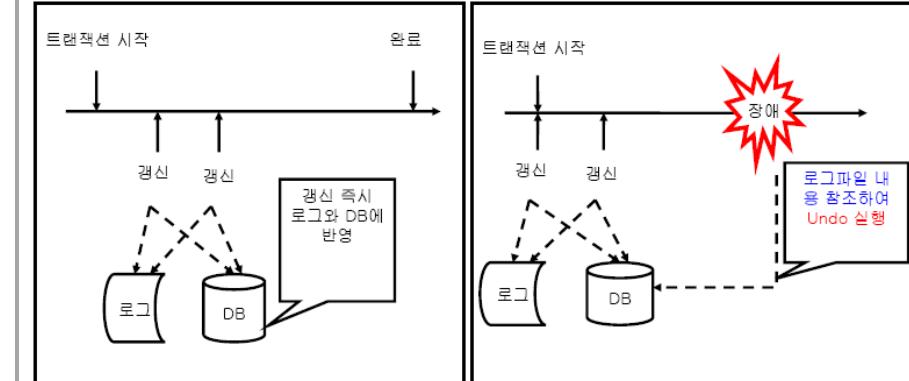
## 1. 데이터베이스 로그를 이용한 회복기법

- 데이터의 **변경이 발생할 때마다 생성되는 로그 파일을 이용**하는 방식
- 유형 : 지연 갱신, 즉시 갱신

지연 갱신



즉시 갱신



갱신	<ul style="list-style-type: none"> <li>트랜잭션 단위가 종료될 때까지 DB에 Write 연산을 지연시키고 동시에 <b>DB 변경내역을 Log에 보관</b>한 후 <b>트랜잭션이 완료되면 Log 를 이용하여 데이터베이스에 Write 연산을 수행</b></li> </ul>
회복	<ul style="list-style-type: none"> <li>트랜잭션이 종료된 상태이면 <b>회복</b> 시 Undo 없이 <b>Redo</b>만 실행함.</li> <li>트랜잭션이 종료가 안된 상태였으면 Log 정보는 무시함</li> </ul>

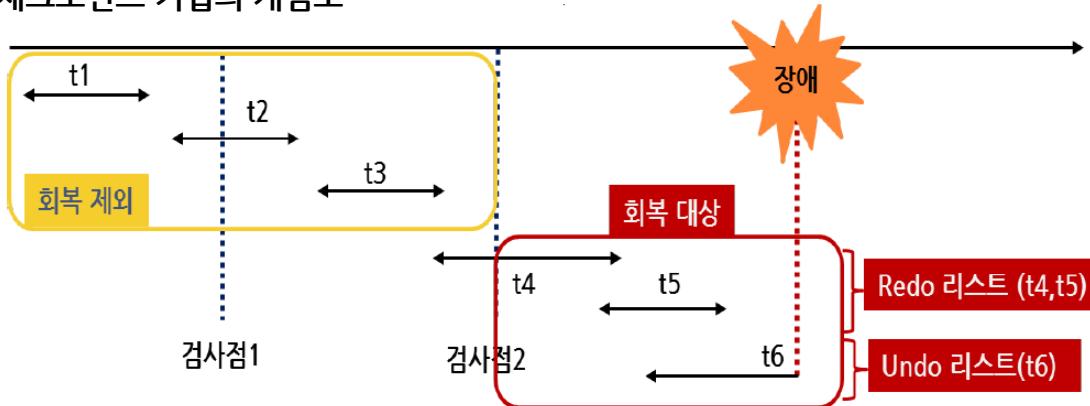
갱신	<ul style="list-style-type: none"> <li>트랜잭션 활동상태에서 <b>갱신결과를 DB 에 즉시 반영하고 (commit 전) log 기록</b></li> </ul>
회복	<ul style="list-style-type: none"> <li>트랜잭션 수행 도중 실패상태에 도달하여 트랜잭션을 철회할 경우에는 로그 파일에 저장된 내용을 참조하여 <b>Undo 연산 수행</b></li> </ul>

# [장애와 회복기법] 체크포인트

## 1. 로그파일과 검사점을 이용한 복구기법, 체크포인트 기법의 개요

- 로그는 그대로 기록 유지하면서, 회복 관리자가 정하는 일정한 시간 간격으로 검사시점을 생성하는 것

## 2. 체크포인트 기법의 개념도



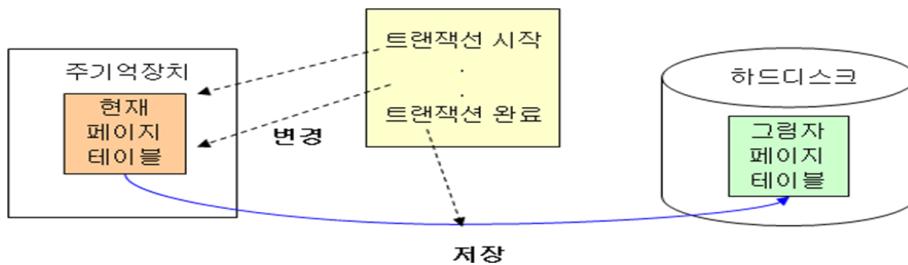
갱신	<ul style="list-style-type: none"> <li><u>검사점(Check Point)을 Log 파일에 기록하고 장애 발생시에 검사시점 이전에 처리된 트랜잭션은 회복작업에서 제외하고 검사시점 이후에 처리된 내용에 대해서만 회복작업을 수행하는 회복기법</u></li> </ul>
회복	<ul style="list-style-type: none"> <li>트랜잭션 수행 도중 문제점 발생 시, 로그 파일의 정보를 모두 검사하여, <u>Redo와 Undo 연산을 실행할 트랜잭션과 체크포인트 선정</u></li> <li>검사점의 Log 파일 기록을 이용하여 실행함</li> <li>장애 발생 시 검사점 이전에 처리된 트랜잭션들을 회복대상에서 제외</li> <li>검사점 이후에 처리된 트랜잭션에 대해서만 회복작업을 시행.</li> <li>새로 시작한 트랜잭션은 Undo 리스트</li> <li>Commit 된 트랜잭션은 Redo 리스트</li> <li>로그 역방향으로 Undo 실행 후 로그 방향으로 Redo 실행</li> </ul>

# [장애와 회복기법] 그림자페이지

## 1. 그림자 페이지(Shadow Page) 기법의 개요

- 로그를 이용하지 않고, 현재 페이지 테이블(current page table)과 그림자 페이지 테이블(shadow page table)을 유지하는 것

## 2. 그림자 페이지 기법 개념도

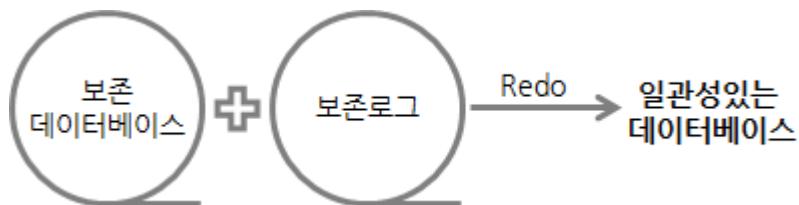


- 현재 페이지 테이블은 주기억장치, **그림자 페이지 테이블은 하드디스크에 저장**함.
- 데이터베이스 **트랜잭션의 시작시점에 현재 페이지 테이블의 내용과 동일한 그림자 페이지 테이블을 생성**함.
- 트랜잭션의 변경 **연산이 수행되면, 현재 페이지 테이블의 내용만 변경**하고 그림자 페이지 테이블의 내용은 변경하지 않음.
- 트랜잭션이 성공적으로 완료될 경우, 현재 페이지 테이블의 내용을 그림자 테이블의 내용으로 저장**함

복구방법	<ul style="list-style-type: none"> <li>수정된 데이터베이스 페이지를 반환하고, 현재의 페이지 테이블을 폐기한다.</li> <li><b>그림자 페이지 테이블을 현재 페이지 테이블로 설정</b>한다</li> </ul>
단점	<ul style="list-style-type: none"> <li>데이터 단편(data fragmentation)</li> <li>쓰레기 수집(garbage collection)</li> <li>병행트랜잭션(concurrent transaction) 지원이 곤란=&gt;병행수행환경에서는 로그와 검사시점 기법을 함께 사용</li> </ul>

# [장애와 회복기법] 디스크 장애 대비한 장애회복 기법

- 원리 : 데이터베이스 내용 전체를 주기적으로 다른 안전한 저장장치에 덤프시키는 것
- 디스크 붕괴시 최근 덤프를 이용하여 장애 발생 이전의 데이터로 상태 복원(Redo)



# [장애와 회복기법] ARIES 회복 알고리즘

- 장애가 발생하여 복구 처리 중에 다시 에러가 발생했을 시 대응하는 알고리즘

## 1. 빠른 데이터베이스 복구를 위한 알고리즘, ARIES 회복 알고리즘의 개요

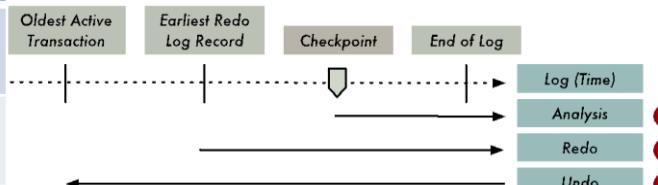
- REDO 중 역사 반복(Repeating history)**: 붕괴가 발생했을 때의 데이터베이스 상태를 복구하기 위하여 붕괴 발생 이전에 수행했던 모든 연산을 다시 한번 수행. 붕괴가 발생했을 때 완료되지 않은 상태였던 (진행 트랜잭션)은 UNDO됨
- UNDO 중 로깅**: UNDO를 할 때에도 로깅을 함으로써 회복을 수행하는 도중에 실패하여 회복을 다시 시작할 때에 이미 완료된 UNDO 연산은 반복하지 않음

## 2. ARIES 알고리즘의 원칙

원칙	설명
<u>Write-ahead logging</u>	<ul style="list-style-type: none"> <li>DB객체에 대한 어떠한 변화도 먼저 로그에 기록, 로그에 있는 그 어떤 기록도 DB객체의 변화가 disk에 기록되기 전에 stable storage에 기록</li> </ul>
<u>Repeating history during redo</u>	<ul style="list-style-type: none"> <li>crash 후 restart되었을 때 log의 모든 행위를 재실시하여 DB를 crash시점의 상태로 만든 후에 crash 시점에 active tx를 undo</li> </ul>
<u>Logging changes during undo</u>	<ul style="list-style-type: none"> <li>tx를 undoing할 때 그 변화를 log에 기록. restart가 재실시될 때 동일한 action이 반복되지 않도록 하기 위함</li> </ul>

## 3. ARIES 회복 알고리즘의 3단계

단계	내용
분석단계	<ul style="list-style-type: none"> <li>붕괴가 발생한 시점에 버퍼에 있는 수정된 페이지와 진행 트랜잭션을 파악, REDO가 시작되어야 하는 로그의 위치를 결정</li> <li>crash 시점의 buffer pool의 dirty pages와 active transaction 확인</li> </ul>
REDO단계	<ul style="list-style-type: none"> <li>로그의 유효한 상태에서부터 시작하여 모든 action을 재실시하고 database 상태를 crash시점으로 복구</li> <li>분석 단계에서 결정한 REDO 시작 위치의 로그로부터 로그가 끝날 때 까지 REDO를 수행, REDO 된 로그 레코드의 리스트를 관리하여 불 필요한 REDO 연산이 수행되지 않도록 함</li> </ul>
UNDO단계	<ul style="list-style-type: none"> <li>commit되지 않은 tx의 action은 undo하여 database에 committed된 tx의 action만 반영되도록 함</li> <li>로그를 역순으로 읽으면서 진행 트랜잭션의 연산을 역순으로 UNDO</li> </ul>

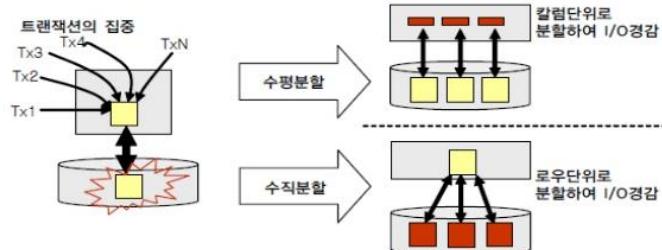


## 4. 회복을 위해 필요한 정보

\* 오손 : 더럽하고 손상함

구분	내용
로그	<ul style="list-style-type: none"> <li>페이지에 대한 갱신(write), 트랜잭션 완료(commit), 트랜잭션 철회(abort), 갱신에 대한 UNDO, 트랜잭션 종료(end) 시 기록, 각 로그 레코드마다 로그 순차번호(LSN)가 할당 (LSN: 디스크에 저장된 로그 레코드의 주소로서 단조 증가 - Log Sequence Number)</li> </ul>
트랜잭션 테이블	진행 트랜잭션에 대한 정보(트랜잭션 식별자, 트랜잭션 상태, 해당 트랜잭션의 가장 최근 로그레코드의 LSN)가 관리
오손* 페이지 테이블	버퍼에 있는 오손 페이지에 대한 정보(페이지 식별자, 해당 페이지에 대한 가장 최근 로그 레코드 LSN)가 관리

# 파티셔닝션



분할방법 종류

## 1. 분산 DB의 관리 전략 Partitioning

- 퍼포먼스(performance), 가용성(availability) 또는 정비용이성(maintainability)를 목적으로 논리적인 데이터 엘리먼트들을 다수의 엔티티(table)로 쪼개는 행위

## 2. 데이터의 분할 방법

분할방식	내용
수평 분할 (Horizontal)	<ul style="list-style-type: none"> <li>한 관계의 튜플을 분할, 둘 이상의 서로 다른 장소에 저장하는 것(<b>RECORD별 분할</b>)</li> <li>분할된 테이블들은 서로 <b>중복되는 레코드들이 없도록 분할</b></li> </ul>
수직 분할 (Vertical)	<ul style="list-style-type: none"> <li>한 관계의 속성을 분할하여 둘 이상의 서로 다른 장소에 저장하는 것(<b>FIELD별 분할</b>)</li> <li>전역 테이블을 구성하는 속성들을 <b>몇 개의 부분 집합으로 분할</b></li> <li><b>수직 분할에 의한 테이블들은 조인 연산에 의하여 재결합이 가능하여야 함</b></li> </ul>
혼합 (Hybrid)	- 수평 분할과 수직 분할을 혼합한 방법
Replication	- 동일한 데이터 사본을 둘 이상의 장소에 중복하여 저장하는 방법

## 3. 파티셔닝 종류

종류	내용	예시
Range partitioning	<ul style="list-style-type: none"> <li>연속적인 숫자나 날짜 등의 기준으로 Partitioning</li> <li>ex) 우편번호, 일별, 월별, 분기별 등 의 데이터에 적합</li> </ul>	
List partitioning	<ul style="list-style-type: none"> <li>분포도가 비슷하며, 많은 SQL에서 해당 컬럼의 조건이 많은 경우 유용</li> <li>ex) [한국, 일본, 중국 -&gt; 아시아] [노르웨이, 스웨덴, 핀란드 -&gt; 북유럽]</li> </ul>	
Hash partitioning	<ul style="list-style-type: none"> <li>Partition Key의 Hash값에 의한 Partitioning (균등한 데이터 분할 가능)</li> <li>파티션을 위한 범위가 없는 데이터에 적합</li> </ul>	
Composite partitioning	<ul style="list-style-type: none"> <li>큰 파티션에 대한 I/O 요청을 여러 partition으로 분산</li> <li>Range Partitioning 할 수 있는 Column이 있지만, Partitioning 결과 생성된 Partition이 너무 커서 효과적으로 관리할 수 없을 때 유용</li> </ul>	유형 : Range-list, Range-Hash

# 데이터베이스 샤딩 (Sharding)

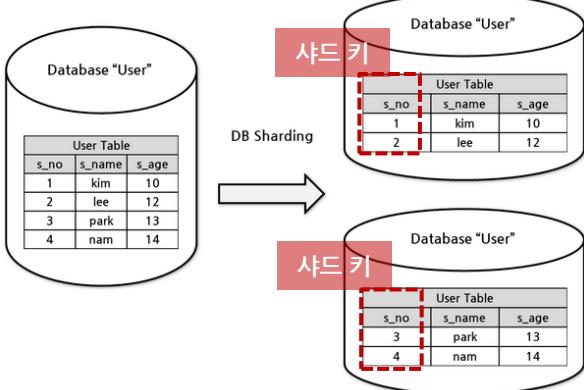
수평 분할

Scale-Up

## 1. 대량의 데이터 처리하기 위한 데이터베이스 파티셔닝 기술, 샤딩의 개요

- **물리적으로 다른 데이터베이스에** 데이터를 수평 분할 방식으로 분산 저장하고 조회하는 방법
- 대용량의 데이터를 처리하기 위해 **테이블을 수평 분할**하여 **데이터를 분산 저장하고 처리**하는 방법(샤드키를 기준으로 데이터를 나눔)
- DB 확장성을 위해 장비 업그레이드를 통해서 **Scale-Up을 해서 병목현상 해소**

## 2. 샤딩의 개념도



## 3. 샤딩의 데이터베이스 분할 방법

방법	설명	사례
Vertical Partitioning	<ul style="list-style-type: none"> <li>• 테이블 별로 서버를 분할하는 방식</li> </ul>	<ul style="list-style-type: none"> <li>• 사용자 프로필정보용 서버, 사용자 친구 리스트용 서버, 사용자가 만든 컨텐츠용 서버 등으로 분할하는 방식</li> </ul>
Range based Partitioning	<ul style="list-style-type: none"> <li>• 하나의 feature나 table이 점점 거대해지는 경우 서버를 분리하는 방식</li> </ul>	<ul style="list-style-type: none"> <li>• 사용자가 많은 경우 사용자의 지역정보를 이용하여 use별로 서버를 분리하거나, 일정 데이터라면 년도별로 분리, 거래정보라면 우편 번호를 이용하는 방식</li> </ul>
Key or Hash Based Partitioning	<ul style="list-style-type: none"> <li>• 엔티티를 해쉬 함수에 넣어서 나오는 값을 이용해서 서버를 정하는 방식</li> </ul>	<ul style="list-style-type: none"> <li>• 사용자ID가 숫자일 경우 나머지연산을 이용하는 방법</li> </ul>
Directory Based Partitioning	<ul style="list-style-type: none"> <li>• 파티셔닝메커니즘을 제공하는 추상화된 서비스를 만드는 것</li> </ul>	<ul style="list-style-type: none"> <li>• DB 와 Cache를 적절히 조합해서 만들 수 있다.</li> </ul>

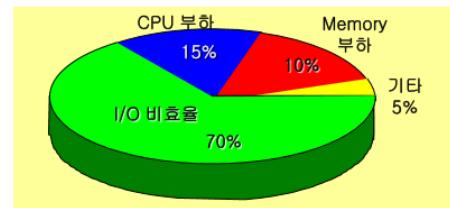
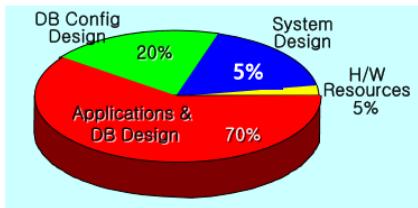
# DB성능 개선방안(Database Tuning)

TRL

## 1. 통합적 관점의 성능개선, DB 투닝

- 데이터베이스의 성능에 영향을 줄 수 있는 요소 및 자원의 최적화를 통해 **처리 속도를 보장하기 위한 개선 활동**

## 2. 데이터 성능저하 요인



H/W적인 문제보다 설계 및 개발의 S/W적 요인이 더 큼

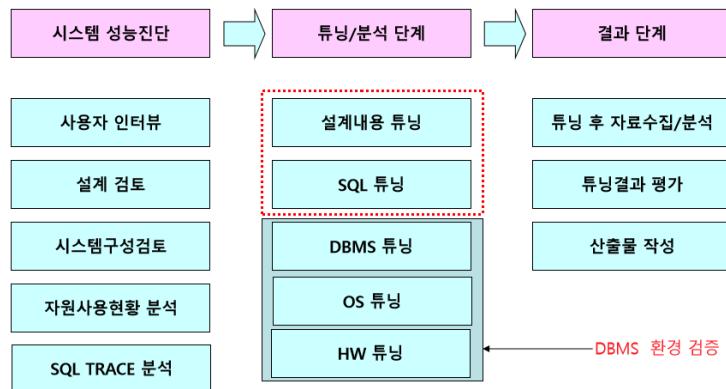
## 3. 데이터베이스 성능 개선 목표

목표	내용
처리능력 (Throughput)	<ul style="list-style-type: none"> <li>해당 작업을 수행하기 위해 소요되는 시간</li> <li>트랜잭션 수(작업량 기준)/시간</li> <li>전체적인 시스템 시각에서 측정/평가</li> </ul>
처리시간 (Throughput Time)	<ul style="list-style-type: none"> <li>작업이 완료되는데 소요되는 시간. 주로 대량 Batch 작업 성능 목표</li> <li>시간단축 요건: 병행 처리(Parallel Processing), 인덱스스캔&lt;Full스캔, Nest-Loop&lt;Hash조인, 병목 제거 위한 작업계획 수립, 대형테이블의 경우 파티션 생성, 대량작업 위한 SORT_AREA, HASH_AREA 메모리 확보 등</li> </ul>
응답시간 (Response Time)	<ul style="list-style-type: none"> <li>사용자가 키를 누른 때부터 시스템이 응답할 때까지의 소요 시간</li> <li>사용자가 느끼는 시스템 성능 척도로 OLTP 시스템의 성능 지표</li> <li>응답시간 향상 요건: 인덱스 이용(액세스 경로 단축), 부분 범위 처리, Sort-Merge/Hash &lt;Nest-Loop 조인, 잠김(Locking) 억제 위한 Sequence 오브젝트 이용</li> </ul>
로드시간(Load Time)	<ul style="list-style-type: none"> <li>정기/비정기적 데이터 로드작업(or 데이터 마이그레이션 등) 수행시간</li> <li>로드시간 단축 요건: 로그파일 사용하지 않는 Direct Load 사용, 병렬 로딩, DISK IO 경합 고려한 작업 분산, 인덱스가 많을 경우 인덱스 삭제&gt;로드&gt;재생성 처리, 파티션을 이용하여 작업 단순화</li> </ul>
반환시간 (Turn-around time)	<ul style="list-style-type: none"> <li>주어진 작업이 그 수행을 위해 시스템에 도착한 시점부터 완료 되어 그 작업의 출력이 사용자에게 제출되는 시점까지의 시간</li> </ul>

# DB성능 개선방안(Database Tuning)

모델, DBMS, SQL

## 4. 데이터베이스 성능 개선 프로세스 및 개선 항목



- Pro-Active : 설계 >환경> SQL
- Re-Active : SQL >환경 >설계

주요 요소	설명	사례
<b>설계관점 (모델링 최적화)</b>	- 데이터베이스 <b>설계 단계에서 성능을 고려</b> 하여 설계 - 데이터 모델링, 인덱스 설계 - 데이터파일, 테이블 스페이스 설계 - 데이터베이스 용량 산정	- 반정규화 - 분산파일배치
<b>DBMS 관점 (환경 최적화)</b>	- 성능을 고려하여 메모리나 블록 크기 등을 지정 - CPU, 메모리 I/O에 관한 관점	- Buffer - Cache 크기
<b>SQL 관점 (응용프로그램 측면)</b>	- <b>SQL 작성 시 성능 고려</b> - join, Indexing, SQL Execution Plan	- Hash / Join - 힌트

## 5. 데이터베이스 성능 개선 기법

구분	기법	내용
DB 설계	테이블의 분할/통합	- 논리적으로는 통합된 단일 테이블이지만 DBMS가 지원하는 파티션 기능 적용함으로써 액세스 효율화, DB I/O 최적화 - DBMS에 따라 파티션 기능 제약 시: 테이블의 수평분할 고려 / 액세스 패턴에 따라 단일 테이블을 1:1로 수직분할 고려
	식별자 지정	- 본질 식별자와 인조식별자의 선택에 따라 정보의 상속과 단절에 영향을 줄 수 있음
	효율적인 인덱스 전략	- 최소한의 인덱스로 최대의 효과를 얻을 수 있는 최적의 인덱스 구조 수립 ( <b>인덱스 분포도 15% 이하 유용 유도</b> , 이상이면 Full Scan)
	반정규화	- 변경이 적고, 성능이 요구되는 경우 <b>조인 최소화</b>
DBMS 투닝	CPU 투닝	- Peak Time 시에 60~70%의 사용량 유지 권고 / - CPU 증설 또는 CPU 과다 점유하는 프로세스를 찾아서 해결
	메모리 투닝	- DBMS의 사용 메모리의 최적화 필요 - 업무를 가능하게 하기 위한 충분한 메모리 확보
	I/O 투닝	- I/O를 분산시킬 수 있도록 데이터 파일의 재배치 / - RAID를 이용→ I/O가 많이 일어나는 것은 Raid 0/1에 배치
	네트워크 투닝	- Ping, Ftp를 이용하여 응답시간 분석
SQL	옵티マイ저 이해기반 SQL 작성	- RBO (Rule Based Optimizer): 통계정보가 없는 상태에서 미리 정해진 Rule에 따라 실행계획 수립 - CBO (Cost Based Optimizer): 통계정보로부터 모든 Access Path를 고려하여 실행계획 수립 - 옵티マイ저가 선택한 실행계획을 확인하고 최적화된 실행계획 수립이 이루어지도록 Factor 부여
	힌트 사용	- 옵티マイ저가 항상 최적화된 실행계획을 수립하는 것은 아니므로 힌트를 사용하여 원하는 실행계획으로 유도
	부분범위 처리	- 액세스하고도 결과를 리턴 할 수 있도록 하여 온라인 프로그램에서 응답시간(Response Time)을 최소화 할 수 있음
	인덱스 활용	- 인덱스가 있음에도 불구하고 SQL을 잘못 기술함으로써 무용지물로 만드는 오류 제거
	조인방식/ 조인순서	- 동일한 SQL문이라도 <b>조인방식과 조인순서에 따라 처리속도 차이 존재</b> SQL 실행계획 확인 후 조정 필요
	다중처리(Array Processing)	- 배치작업의 경우 한번의 DBMS 호출로 여러 건을 동시에 처리할 수 있는 다중처리 활용
	병렬쿼리(Parallel Query)	- 배치작업의 경우 하나의 SQL을 여러 개의 CPU가 병렬로 분할 처리하게 함으로써 처리속도 향상 가져옴
	Static SQL 사용	- 조건 절에 입력된 값을 먼저 Binding 한 후 실행계획을 수립하는 Dynamic SQL은 파싱 부하가 커지므로 입력 값을 <b>Binding 하기 전에 실행계획을 수립하는 Static SQL을 가급적 사용</b> 하도록 함

# SQL 부분처리 범위

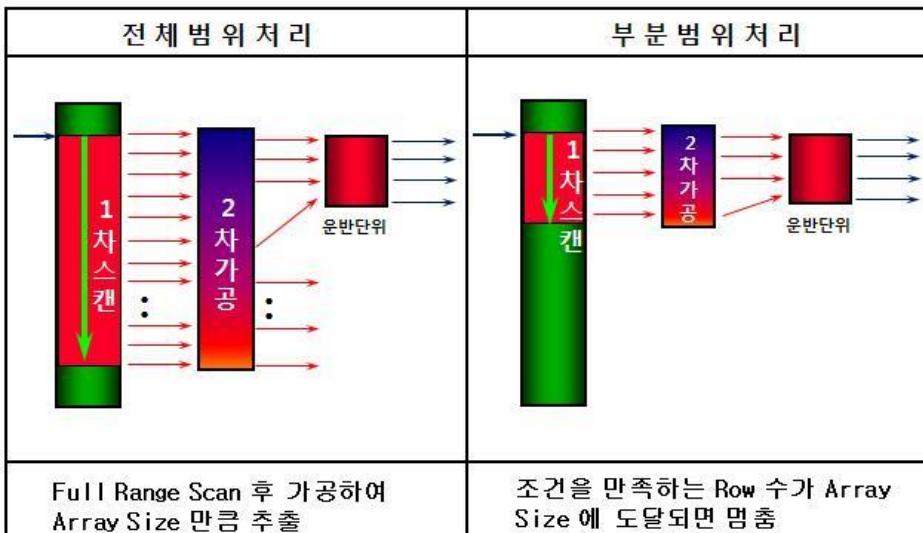
## 1. 넓은 범위의 데이터를 빠른 속도를 얻도록 하기 위한 방법, 부분처리 범위의 개요

- 조건을 만족하는 전체범위를 처리하는 것이 아니라 일단 **운반단위(Array Size)**까지만 처리하여 추출하는 처리방식

## 2. SQL 부분처리의 적용원칙

구분	설명
부분범위처리의 자격	논리적으로 전체범위를 읽어 <u>추가적인 가공을 하지 않고도 동일한 결과를 추출할 수 있다면</u> 자격이 있다.
부분범위처리를 할 수 없는 경우	- ORDER BY가 사용된 경우 - UNION, MINUS, INTERSECT를 사용한 경우
부분범위처리를 할 수 없는 경우의 대체	- ORDER BY -> INDEX를 이용하여 ORDER BY를 하지 않아도 되는 형태로 대체 - MINUS, INTERSECT -> EXISTS, NOT EXISTS, IN, NOT IN

## 3. SQL 부분범위처리의 개념도

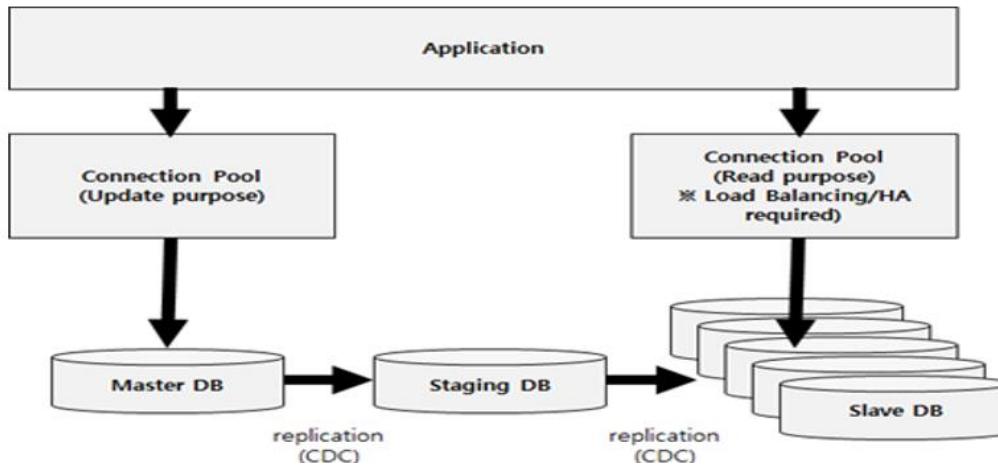


SQL 범위처리로의 유도 방법	내용
인덱스나 클러스터를 이용하여 SORT를 대체	SQL 구문에 order by가 있는 경우 인덱스 등을 이용하여 order by를 빼도 되는 형태로 변환한다.
인덱스만으로 엑세스해도 되는 구조	결과 컬럼을 얻어올 때 인덱스에서 모두 가져올 수 있는 항목인지를 살펴서 인덱스가 다시 테이블을 읽지 않아도 되는 형태로 사용한다.
MAX값을 얻어와야 할 경우 인덱스를 이용	보통 사용하는 MAX(seq)+1 형태를 버리고 역순 인덱스를 이용하여 next seq를 구하는 형태로 변경한다.
EXISTS를 활용	데이터의 존재여부를 체크하는 등의 로직을 수행해야 할 때 count()를 수행하는 것보다는 exists를 이용하여 존재여부를 파악한다.
ROWNUM을 활용	<p>rownum은 오라클의 pseudo column 부분적으로 필요한 데이터만을 얻어올 수 있는 구조로 정의하여 사용</p> <pre>select ..., ROWNUM from t where &lt;where clause&gt; group by &lt;columns&gt; having &lt;having clause&gt; order by &lt;columns&gt;;</pre> <p>실행순서</p> <ol style="list-style-type: none"> <li>1). FROM/WHERE 절을 처리한다.</li> <li>2). ROWNUM이 할당되고 FROM/WHERE 절에서 전달되는 각각의 출력 로우에 대해 증가한다.</li> <li>3). SELECT가 적용된다.</li> <li>4). GROUP BY 조건이 적용된다.</li> <li>5). HAVING 조건이 적용된다.</li> <li>6). ORDER BY 조건이 적용된다.</li> </ol>

# Query-Off Loading

## 1. 대용량 시스템을 위한 DB아키텍처, 쿼리 오프로딩의 개요

- DB의 트랜잭션에서 **Update 트랜잭션과 Read 트랜잭션을 분리**하여 DB의 처리량을 증가시켜 성능 향상을 위한 기법



구성		설명
구성요소	Master DB	<ul style="list-style-type: none"><li>- <b>Update 트랜잭션(Create/Delete/Update)</b> 만 수행</li></ul>
	Staging DB	<ul style="list-style-type: none"><li>- Slave DB로 <b>복제하기 위한 중간 경유지 역할</b></li><li>- MasterDB에서 바로 다수의 Slave DB로 복제해야 하면 성능 저하 유발 → 이를 방지하기 위해 필요</li></ul>
	Slave DB	<ul style="list-style-type: none"><li>- <b>Read 트랜잭션만 수행</b></li><li>- N개의 Slave DB로 구성</li><li>- Slave DB장애시 다른 Slave DB인스턴스에 접근할 수 있도록 HA 기능 제공</li></ul>
복제기술	CDC (Change Data Capture)	<ul style="list-style-type: none"><li>- Back Log를 이용하여 데이터 복제 → Source DB로부터 Back Log를 읽어서, 복제를 하고자 하는 Target DB에 replay하는 형식</li><li>- 대표적 제품 : Oracle의 Golden Gate, Quest의 Share Flex, 오픈소스 Galera [참고]</li><li>DBMS는 공통적으로 Create/Update/Delete와 같은 쓰기 관련 작업 수행시 데이터를 실제로 저장하기 전에 해당 작업에 대한 request를 Back Log에 저장</li></ul>

# 해싱(Hashing)

DB 해쉬, 보안 해쉬 두가지 존재 >> 개념은 유사  
BUT!! 문제에서 도메인 영역확인하고 접근하기(없다면 둘다 기제 할 것)

제산, 제곱, 중첩, 기수

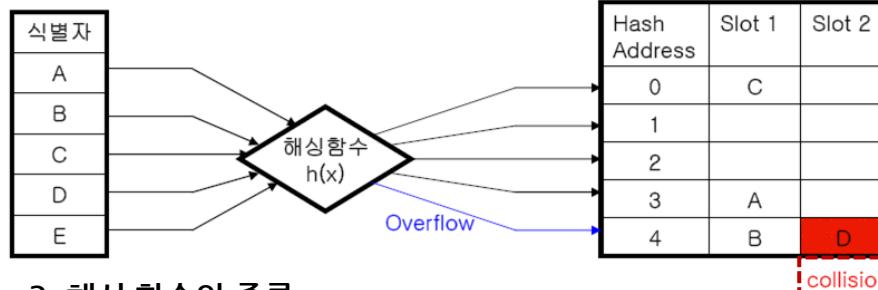
정적/동적 해싱

충돌회피

## 1. 주어진 속성값을 기초로 하여 원하는 목표 레코드를 직접 접근할 수 있게 하는 기법, 해싱의 개요

- 키 값에서 레코드가 저장되어 있는 주소를 직접 계산한 후 산출된 주소로 바로 접근이 가능하게 하는 방법
- 하나의 문자열을 원래의 것을 상징하는 더 짧은 길이의 값이나 키로 변환하는 기술

## 2. 해싱의 개념도 및 적용 기술 (해싱 알고리즘)



## 3. 해시 함수의 종류

기술 구분	내용
제산함수 (Division)	<ul style="list-style-type: none"> <li><b>나머지를 구하는 MOD 연산자 이용하여 주소 값을 취하는 방식</b></li> <li>나누고자 하는 값, 즉 제수는 해싱 테이블의 크기를 나타냄</li> <li>적재율은 70 ~ 80%가 적당</li> </ul>
제곱 함수 (Mid Square)	<ul style="list-style-type: none"> <li>키 값의 중간 N자리를 뽑아서 제곱한 후 상대 번지로 사용</li> <li><b>제곱한 결과를 주소 공간의 크기에 맞도록 조정</b></li> </ul>
중첩 함수 (Folding)	<ul style="list-style-type: none"> <li><b>키 값을 여러 방식으로 접어 값을 합산한 후 버킷(Bucket) 주소로 활용</b></li> </ul>
기수 변환 (Radix Conversion)	<ul style="list-style-type: none"> <li>주어진 키 값을 <b>특정 진법</b>으로 간주한 후 다른 진법으로 변환 값을 이용하는 기술임</li> </ul>
기타	<ul style="list-style-type: none"> <li>계수분석(Digit Analysis), 대수적 코딩(Algebraic Coding)</li> </ul>

종류	설명	개념도
정적해싱 (Static Hashing)	<ul style="list-style-type: none"> <li><b>버켓 주소의 집합을 고정시켜 처리하는 기법</b></li> <li>파일크기가 커짐에 따라 주기적 해싱구조 재구성</li> <li>Collision(충돌) 발생</li> <li><b>종류 : Mid-Square, Division, Folding</b></li> </ul>	
동적해싱 (Dynamic Hashing)	<ul style="list-style-type: none"> <li>데이터의 증감에 적응하기 위해 동적으로 해시 함수가 변경되도록 하는 기술(Overflow 발생 시 2배수 확장)</li> <li>데이터베이스가 커지면서 <b>버킷들을 쪼개거나 합침</b></li> </ul>	

\* Collision : 2개 이상의 레코드가 동일한 버켓 주소를 갖는 경우를 말하며, 삽입해야 할 레코드가 만월이 된 버켓에 해싱이 되면 오버플로우가 발생하게 됨.

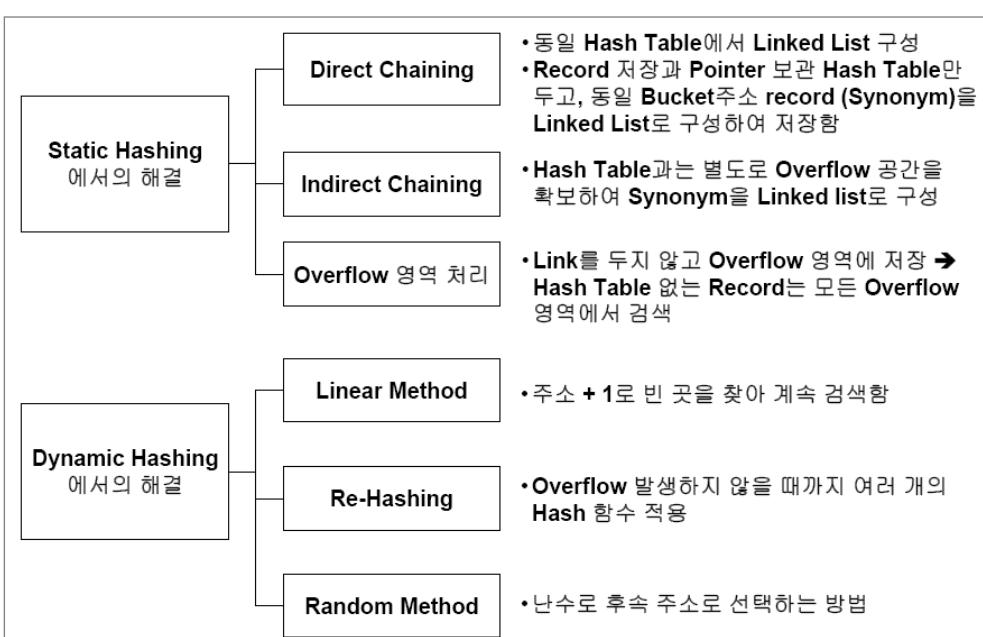
# 해싱(Hashing)

개방/폐쇄

선재무직간오

## 4. 충돌해결전략 및 방안

해결전략	주요 내용	주요방법
Bucket 해싱	<ul style="list-style-type: none"> <li>버켓 : 하나의 주소를 가지면서 하나 이상의 레코드를 저장할 수 있는 파일의 한 구역</li> <li>테이블 엔트리에 몇 개의 키 값이 들어가도록 공간을 만들어 놓음, 충돌시 동일 버켓에 쌓아 놓음</li> <li>버켓의 오버플로우 발생 가능, I/O증가로 인한 성능저하, 메모리 낭비</li> </ul>	
Open Addressing (개방주소방식)	<ul style="list-style-type: none"> <li>충돌이 발생할 경우 다음 가용 공간에 저장</li> <li>영역을 찾을 때까지 계속 수행하는 단순한 방법</li> <li>별도 포인터나 데이터 스트럭처 사용하지 않음</li> </ul>	<ul style="list-style-type: none"> <li>선형방식, 재해싱, 무작위처리법</li> </ul>
Closed Addressing (폐쇄주소방식)	<ul style="list-style-type: none"> <li>같은 해시 값을 가지는 레코드들을 리스트로 만들어 관리</li> <li>충돌을 쉽게 다룰 수 있음</li> <li>linked list를 통해 second clustering이라는 데이터 치우침 현상 방지</li> <li>데이터 영역이 동적으로 할당되므로 테이블의 크기에 관계없이 다양한 레코드 관리가 가능함</li> </ul>	<ul style="list-style-type: none"> <li>연결방식(직접/간접), 오버플로우 영역 처리 방식</li> </ul>



해결방법	설명
<b>선형방식 (linear method)</b>	<ul style="list-style-type: none"> <li>어떤 레코드의 키 K의 주소에 이미 다른 키가 저장되어 있을 경우 선형 방법은 키 K 레코드를 저장한 주소 대신 <b>후속 주소로 주소+1을 지정</b>.</li> <li>이 때 지정한 후속 주소가 빈자리이면 새로운 레코드를 저장하고 이미 다른 레코드가 저장되어 있으면 또 다시 주소+1을 지정하여 주소로 사용함.</li> </ul>
<b>재해싱</b>	<ul style="list-style-type: none"> <li><b>여러 개의 해싱함수를 적용</b>하는 방법.</li> <li>오버플로우가 발생하지 않을 때까지 여러 개의 해싱함수 적용</li> </ul>
<b>무작위처리법 (random method)</b>	<ul style="list-style-type: none"> <li>난수를 발생시켜 해시표의 주소를 후속 주소로 선택하는 방법으로 이미 <math>h_0(K)</math>의 자리에 다른 레코드가 저장되어 있으면 <b>후속 주소는 <math>h_1(K)=h_0(K)+r_1</math>이 됨</b></li> </ul>
<b>직접 연결방식 (Direct chaining)</b>	<ul style="list-style-type: none"> <li><b>동일한 해시표 내에서</b> synonym 들을 <b>linked list로 구성</b></li> <li>레코드 저장 부분과 포인터 보관 부분을 가진 버킷 단위로 구성된 해시 표만을 두고 비슷한 레코드들을 연결 리스트로 구성하여 저장하는 것.</li> </ul>
<b>간접 연결방식 (Indirect chaining)</b>	<ul style="list-style-type: none"> <li><b>해시표와는 별도로 오버플로우 공간을 확보</b>하여 synonym 들을 linked list로 구성</li> </ul>
<b>오버플로우 영역 처리 방법</b>	<ul style="list-style-type: none"> <li>연결기법과는 달리 각 버켓에 링크 부분을 두자 않고 <b>과잉 상태를 유발한 레코드들을 오버플로우 영역에 저장</b>하는 단순한 방법</li> <li>검색하려는 레코드가 해시표에 없으면 오버플로우 영역내 모든 레코드들을 찾아야 한다는 단점이 있음.</li> </ul>



# CAP Theorem(브루어의 정리)

완벽한 CP시스템

완벽한 AP시스템

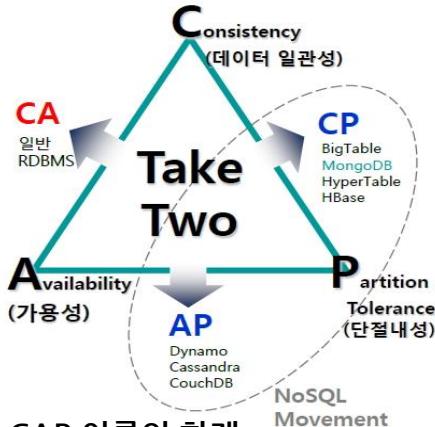
요구 사항에 따른 수준

일관성, 가용성, 단절내성

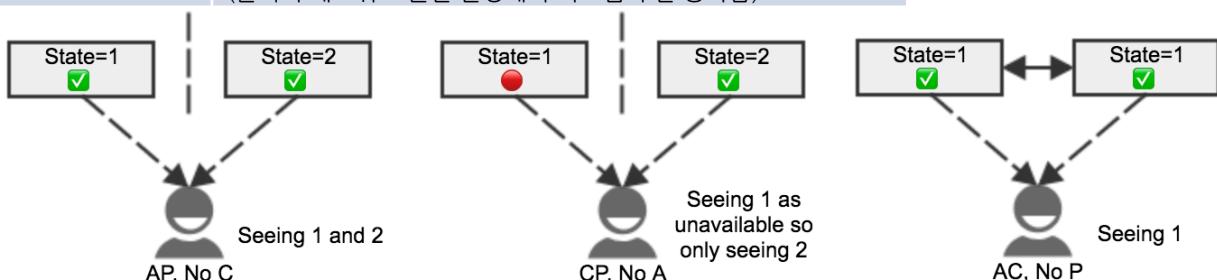
## 1., CAP Theorem의 개념

- 대용량 분산 시스템(데이터 저장소)는 데이터 일관성(Consistency), 가용성(Availability), 단절내성(Partition Tolerance)을 모두 만족시키는 것이 불가능하므로 두 가지만 전략적으로 선택해야 한다는 이론

## 2. CAP Theorem의 구성도와 항목



구분	항목
C onsistency	모든 노드들은 같은 시간에 같은 데이터를 보여줘야 함. (각각의 사용자가 항상 동일한 데이터를 조회함)
A vailability	몇몇 노드가 다운되어도 다른 노드들에게 영향을 주지 않아야 함. (모든 사용자가 항상 읽고 쓸 수 있음)
P artition Tolerance	일부 메시지를 손실하더라도 시스템은 정상 동작을 해야 함 (물리적 네트워크 분산 환경에서 시스템이 잘 동작함)



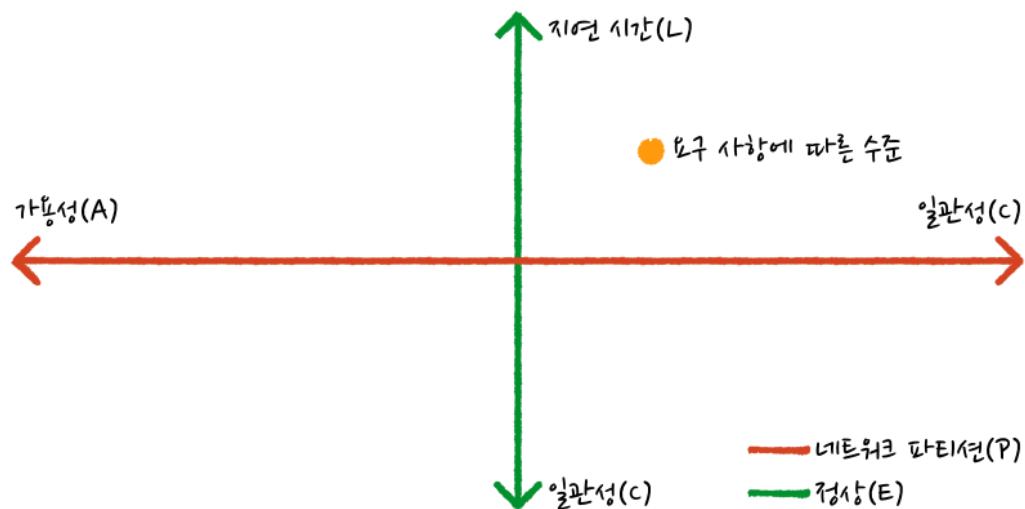
## 3. CAP 이론의 한계

- CAP 이론에 따르면 분산 시스템은 CP이거나 AP여야 함. 하지만 실제 시스템은 둘 중에 하나라고 명확히 구분지을 수 없으며, 완벽한 CP시스템이나 완벽한 AP시스템은 사실상 쓸모가 없음
- 완벽한 CP시스템** - 완벽한 일관성을 갖는 분산 시스템에서는 하나의 트랜잭션이 다른 모든 노드에 복제된 후에 완료될 수 있다. 이는 가용성 뿐만 아니라 성능의 희생을 필요로 한다. 이런 시스템은 하나의 노드라도 문제가 있으면 트랜잭션은 무조건 실패하고, 노드가 늘어날 수록 지연시간은 길어질 것이다.
- 완벽한 AP시스템** - 완벽한 가용성을 갖는 시스템은 모든 노드가 어떤 상황에서라도 응답할 수 있어야 한다. 하나의 노드가 네트워크 파티션으로 인해 고립되었다고 생각해보자. 이런 상황에서 고립된 노드가 갖고 있는 데이터는 쓸모가 없어지지만(일관성이 깨지므로) 어쨌든 응답한다면 완벽한 가용성을 갖게 된다. 운 나쁘게 이 노드와 연결된 사용자는 문제를 인지하지 못하고 계속해서 요청을 보낼 것이다.
  - 일관성과 가용성은 상충 관계에 있지만 둘 중에 반드시 하나만을 선택해야 하는 것은 아니다'
  - 완벽한 CP시스템과 완벽한 AP시스템 사이에는 수많은 가능성이 있다. 요구 사항에 따라 '다소 강한 일관성-다소 약한 가용성', '다소 약한 일관성-다소 강한 가용성'과 같이 일관성과 가용성의 수준을 선택해야 한다.

# PACELC 이론

## 1. CAP 이론의 한계를 극복하기 위한 이론, PACELC 이론의 개요

- CAP 이론이 네트워크 파티션 상황에서 일관성-가용성 축을 이용하여 시스템의 특성을 설명
- PACELC 이론: CAP 이론에 정상 상황이라는 새로운 축을 추가
- PACELC : P(네트워크 파티션)상황에서 A(가용성)과 C(일관성)의 상충 관계와 E(else, 정상)상황에서 L(지연 시간)과 C(일관성)의 상충 관계를 설명
- PACELC 이론에서는 장애 상황, 정상 상황에서 어떻게 동작하는지에 따라 시스템을 PC/EC, PC/EL, PA/EC, PA/EL로 나눔

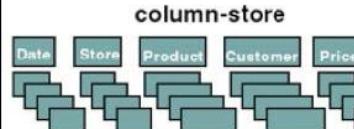
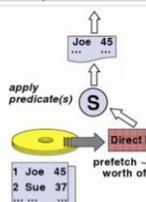
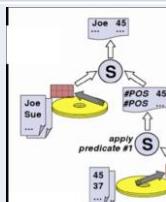


# 로우 기반 DBMS와 칼럼 기반 DBMS

## 1. 로우 기반 DBMS와 칼럼 기반 DBMS의 개요

로우 기반 DBMS (Row-Oriented DBMS)	데이터를 디스크에 저장하는데 있어서 행 기반 저장 방식, 즉 <u>레코드 단위로 데이터를 저장</u> 하는 DBMS
컬럼 기반 DBMS (Column-Oriented DBMS)	데이터를 디스크에 저장하는데 있어서 기존의 행 기반 저장 방식(레코드 단위의 저장)이 아니라 컬럼기반 저장 방식, 즉 <u>같은 필드의 값을 모아서 저장</u> 하는 DBMS

## 2. 로우 기반 DBMS와 칼럼 기반 DBMS의 비교

구분	ROW기반 DB	Column 기반 DB
구조도	<ul style="list-style-type: none"> <li>-<u>로우 단위로 데이터를 저장</u>함</li> <li>-하나의 디스크 페이지에 여러 레코드가 저장되는 구조</li> </ul> 	<ul style="list-style-type: none"> <li>-컬럼 단위로 데이터를 저장함</li> <li>-컬럼 별로 파일이 생성되고 디스크 페이지에는 <u>동일 컬럼 값들이 연속적으로 저장</u>되는 구조</li> </ul> 
적용 DBMS	레코드를 추가하거나 삭제하는데 적합	연관된 데이터들을 위주로 읽는데 유리
데이터조작시 단점	데이터 조회시 불필요한 컬럼값도 읽음	데이터 수정 시 여러군데를 수정하여야함
압축효율	레코드 단위의 압축은 상대적으로 효율이 낮음	동일한 도메인 컬럼 값이 연속되므로 압축 효율이 좋음
질의 형태	레코드 단위의 인덱스를 구성하여 랜덤 액세스에 유리	컬럼에 대한 범위 선택 질의에 유리
용도	<ul style="list-style-type: none"> <li>-실시간 온라인 처리(<b>OLTP 시스템</b>)</li> <li>-특정 레코드를 조회하는 질의에 유리</li> </ul>	<ul style="list-style-type: none"> <li>-읽기 중심의 대용량 <b>DW나 OLAP 시스템</b></li> <li>-특정 컬럼의 집계 값을 구하는 질의에 유리</li> </ul>
업무 성격	실시간 업데이트 형태의 거래	배치 형태의 대량의 트랜잭션 처리
조회 데이터	<ul style="list-style-type: none"> <li>-일부 데이터의 <u>전체 칼럼 추출</u></li> <li>-(SELECT * FROM ~)</li> </ul>	<ul style="list-style-type: none"> <li>-전체 데이터의 일부 컬럼의 집계 데이터 추출</li> <li>-(SELECT SUM(COL1) FROM ~)</li> </ul>
적용 DBMS	-일반적인 RDBMS (Oracle, DB2 등)	<ul style="list-style-type: none"> <li>-DW 전용 DB (Vertica 등)</li> <li>-Key-Value 타입의 NoSQL (HBase 등)</li> </ul>
데이터 중복	-중복 값이 적은 DB	-중복 값이 많은 DB
SQL 처리 과정	<pre>SELECT NAME, AGE WHERE AGE &gt; 40</pre> 	

# 병렬 데이터베이스

## 1. 병렬 데이터베이스의 개요

- 성능을 향상시키기 위해 데이터 로딩, 인덱스 생성, 쿼리 처리 등 동작을 병렬적으로 수행하는 데이터베이스
- 다수의 마이크로프로세서를 동시에 사용하여 데이터 처리를 고속으로 수행하는 데이터베이스
- 주로 멀티프로세서 구조를 통한 병렬 처리를 가리킴

## 2. 병렬데이터 베이스의 유형 (메모리와 디스크 공유 여부로 나뉨)

Shared Memory	Shared Disk	Shared Nothing
<ul style="list-style-type: none"><li>모든 프로세서는 자체 디스크를 소유하고 있음</li><li>메인 메모리 공통 영역(Global Shared Memory)을 Interconnection network을 통하여 접근 할 수 있음</li></ul> <p>(Tightly coupled)</p>	<ul style="list-style-type: none"><li>모든 프로세서는 자체 메모리 보유(다른 프로세서는 접근 못함)</li><li>모든 프로세서는 모든 디스크를 접근할 수 있음</li></ul> <p>(Loosely coupled)</p>	<ul style="list-style-type: none"><li>가장 보편적인 아키텍처</li><li>모든 프로세서는 자체 메모리/디스크를 소유, 고속의 네트워크와 스위치를 통한 통신 수행.</li></ul> <p>Interconnection Network</p>

## 3. 병렬데이터 분할 방법

분할방법	설명
구간 파티셔닝(Range Partitioning)	<ul style="list-style-type: none"><li>샤딩과 동일한 의미를 가지며 스키마를 다수의 복제본을 구성하고 각각의 샴드에 샴드키를 기준으로 데이터를 분리</li></ul>
해시 기반 파티셔닝(Hash-based partitioning)	<ul style="list-style-type: none"><li>엔티티를 해시함수에 넣어서 나오는 값을 이용해서 서버를 정하는 방식</li></ul>
라운드 로빈 기반 파티셔닝(Round-robin partitioning)	<ul style="list-style-type: none"><li>각 파티션에 메시지를 순환하여 균등하게 분배하는 방식</li></ul>

# 분산 데이터베이스

위복병분장지

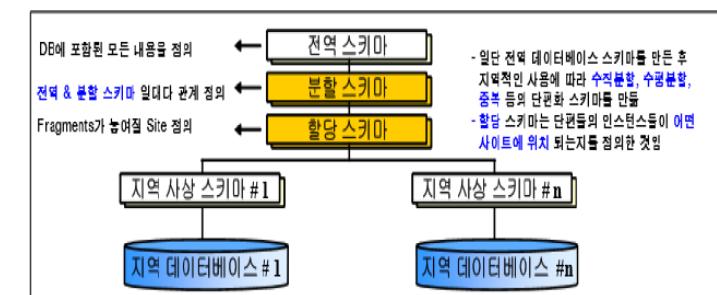
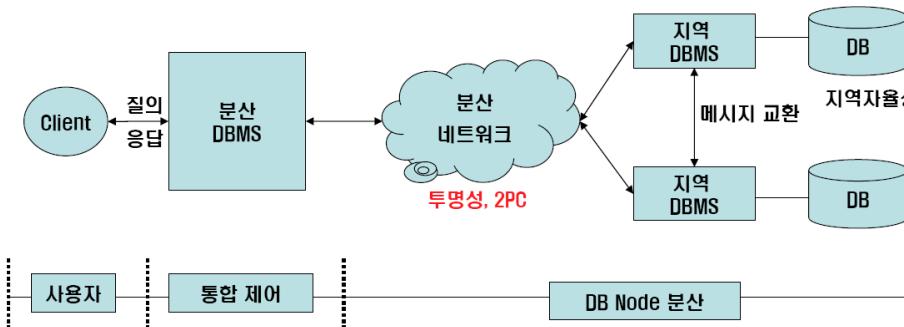
## 1. 신뢰성과 확장성을 제공하는 분산 데이터베이스의 개요

- 여러 곳으로 분산되어 있는 데이터베이스를 하나의 가상 시스템으로 사용할 수 있도록 한 데이터베이스
- 논리적으로 하나의 시스템으로 구현되어 있으나, 물리적으로 네트워크를 통하여 분산화된 형태로 관리되는 데이터베이스
- 데이터처리 지역화, 운영 및 관리의 지역화, 처리/부하분산 및 병렬처리, 데이터가용성 신뢰성 향상

## 2. 분산 데이터베이스의 투명성(Transparency) - 6가지

특성	주요개념
<b>위치 투명성</b>	사용자나 응용프로그램이 접근할 데이터의 물리적 위치를 알아야 할 필요가 없는 성질 이를 보장하기 위해 DBMS는 Distributed Data Dictionary Directory가 필요
<b>복제 무관성</b>	사용자가 응용프로그램이 접근할 데이터가 물리적으로 여러 곳에 복제되어 있는지 여부에 대해 알 필요가 없는 성질
<b>병행 무관성</b>	여러 사용자나 응용프로그램이 동시에 분산 데이터베이스에 대한 트랜잭션을 수행하는 경우에도 결과에 이상이 발생하지 않는 성질 (Locking, Time Stamp 기법 이용)
<b>분할 투명성</b>	사용자가 하나의 논리적 릴레이션이 여러 단편으로 분할되어 각 단편의 사본이 여러 Site에 저장되어 있음을 알 필요가 없는 성질, 성능향상, Fragmentation을 위한 설계 필요
<b>장애 무관성</b>	데이터베이스가 분산되어 있는 각 지역의 시스템이나 통신망에 이상이 생기더라도 데이터의 를 보존할 수 있는 성질 (2PC 활용)
<b>지역사상 투명성</b>	지역 DBMS 와 물리적 DB 사이의 Mapping보장, 각 지역 시스템 이름과 무관한 이름 사용 가능 / 점진적 확장 가능주요개념

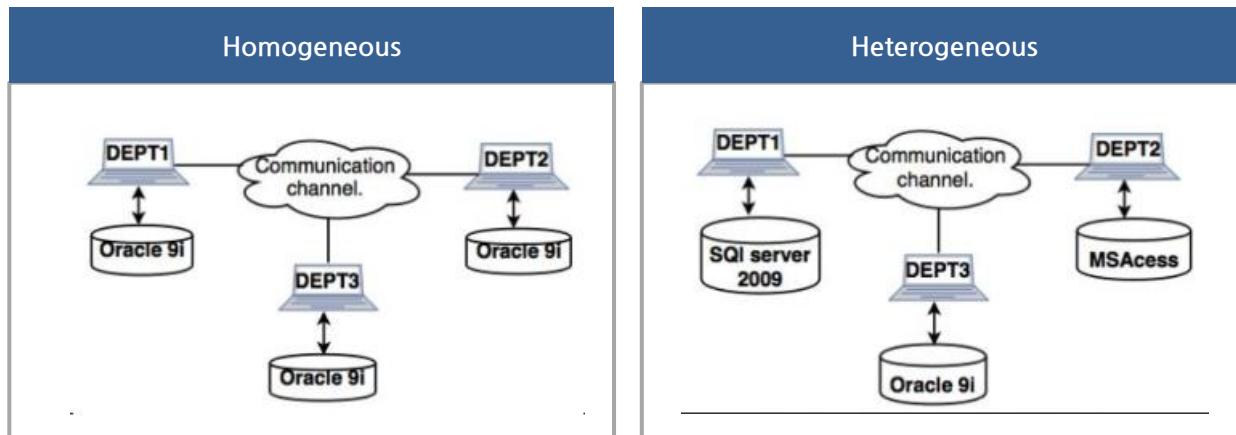
## 3. 분산 DB의 구조와 종류



종류	내용
<b>동질형</b>	-모든 지역에 동일한 DBMS를 사용하는 방식 -전역 Interface를 통해 완전 동질 투명성을 제공
<b>이질형</b>	-지역간 이기종 DBMS를 사용하는 경우 -이기종 데이터베이스와 연동하기 위해 별도의 게이트웨이가 필요 -기존 독립적 Site를 통합하는 Bottom up 경우 발생
<b>강 결합형</b>	-각 Site의 DB 전역 data를 참조하여 운영 -전역 스키마 분산DB: 전역스키마가 모든 통제 수행 -연합 분산 DB: 참조 데이터를 Import/Export하는 방식
<b>약 결합형</b>	-각 지역 Site DB가 필요한 데이터만을 상호 교환하는 방식 약 결합형

# 분산 데이터베이스

## 4. 분산데이터 베이스의 유형 (데이터베이스가 동일 여부)



## 5. 분산데이터 베이스에서의 데이터 분할 방법

분할방법	설명
수평분할 (Horizontal)	-한 릴레이션의 튜플을 분할, 둘 이상의 서로 다른 장소에 저장하는 것(RECORD별 분할)
수직 분할 (Vertical)	-한 릴레이션의 속성을 분할하여 둘 이상의 서로 다른 장소에 저장하는 것 (FIELD별 분할)
혼합(hybrid)	-수평 분할과 수직 분할을 혼합한 방법

## 6. 일반 DB와 분산DB 비교

구분	일반DB	분산 DB
통제	중앙통제	지역자치 및 전역 스키마에 의한 통제
데이터	<b>중복 배제, 일관성</b>	<b>데이터 분산, 중복 허용</b>
Data 중복성	데이터 공유를 통한 중복도 감소	중복성 및 일관성은 바람직
구조	통합DBMS 하나로 구성	업무 지역별 DBMS존재
트랜잭션	<b>동시성 제어</b>	<b>분산 동시성 제어 필요</b>
투명성	기능	자체 지원
보안통제	단일 DBA	지역적 DBA

# 병렬데이터베이스 VS 분산데이터 베이스

## 1. 병렬데이터베이스 VS 분산데이터 베이스

구분	병렬데이터베이스	분산데이터베이스
지리적 위치	<ul style="list-style-type: none"><li>노드는 지리적으로 동일한 위치</li></ul>	<ul style="list-style-type: none"><li>노드는 대개 지리적으로 다른 위치</li></ul>
실행속도	<ul style="list-style-type: none"><li>빠름</li></ul>	<ul style="list-style-type: none"><li>상대적으로 느림</li></ul>
오버헤드	<ul style="list-style-type: none"><li>적다</li></ul>	<ul style="list-style-type: none"><li>상대적으로 많음(여러 지역에 대한 관리가 복잡하고 비용도 많이 발생)</li></ul>
노드 유형	<ul style="list-style-type: none"><li>동일함(Homogeneous)</li></ul>	<ul style="list-style-type: none"><li>꼭 동일하지 않아도 됨(Homogeneous &amp; Heterogeneous)</li></ul>
Performance	<ul style="list-style-type: none"><li>Lower reliability &amp; availability</li><li>(장애가 발생하면 사용이 불가하여 신뢰성과 가용성이 낮음)</li></ul>	<ul style="list-style-type: none"><li>Higher reliability &amp; availability</li></ul>
확장범위	<ul style="list-style-type: none"><li>확장에 어려움 있음</li></ul>	<ul style="list-style-type: none"><li>확장이 용이</li></ul>
백업	<ul style="list-style-type: none"><li>한 개의 사이트에서만 백업</li></ul>	<ul style="list-style-type: none"><li>여러 개의 사이트에서 백업 가능</li></ul>
운영 비용	<ul style="list-style-type: none"><li>상대적으로 적음</li></ul>	<ul style="list-style-type: none"><li>상대적으로 높음</li></ul>
장점	<ul style="list-style-type: none"><li>고성능 온라인 트랜잭션 처리 가능</li><li>데이터베이스 처리 요구를 병렬적으로 처리할 수 있어서, 속도가 매우 빠름</li><li>클러스터 상의 한 노드의 장애 시에도 지속적인 서비스 가능</li></ul>	<ul style="list-style-type: none"><li>데이터가 여러 사이트에 복제됨으로 가용성이 매우 뛰어남 - 다양한 노드가 연동되어 부하가 분산되어 성능이 향상됨</li><li>다양한 수준의 투명성, 하드웨어, 운영 체제, 네트워크 및 위치 독립성을 갖춘 분산된 데이터 관리, 연속성을 제공함</li></ul>
단점	<ul style="list-style-type: none"><li>병렬 아키텍처에 대한 프로그래밍이 어려움</li><li>성능 향상을 위해 다양한 코드 변경이 수행 되어야 함</li><li>멀티코어 아키텍처 특성상 전력 사용율이 매우 높음(효율적인 냉각기술 필요)</li></ul>	<ul style="list-style-type: none"><li>DBA는 분산 데이터베이스 구성을 위한 추가적인 작업을 수행해야 하는 부담</li><li>인프라가 확장될 경우, 복잡성이 증가하고 추가비용이 발생함</li><li>중앙 뿐만 아니라 원격지 사이트에도 보안을 유지해야 함</li><li>분산 DB환경에서 무결성을 유지하기 위한 많은 네트워크 자원들이 필요 할 수 있음</li><li>분산 데이터베이스에 대한 구축 경험이 부족하고, 분산형 DB로 전환하는데 도움되는 도구나 방법론이 없음(표준 부재)</li></ul>



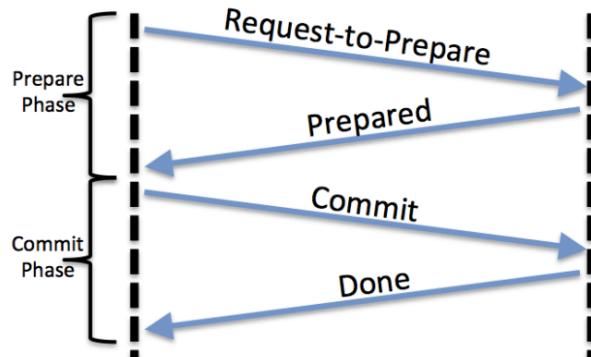
# 2PC(Two Phase Commit)

2단계(Prepare, commit)

## 1. 분산 데이터베이스 환경에서의 트랜잭션 처리를 위한 2PC

- 분산 데이터베이스 환경에서 원자성을 보장하기 위해 분산 트랜잭션에 포함되어 있는 모든 노드가 Commit하거나 Rollback하는 메커니즘
- 분산데이터베이스에서는 모든 지역의 데이터베이스에서 트랜잭션이 성공 완료되었음을 확인한 후에 트랜잭션의 처리가 완료되어야 함

## 2. 2PC의 개념도 및 주요 절차



2 Phase Commit



질의처리 단계	처리내용	참여자	주요 개념
Phase 1 ) <b>Prepare Phase</b>	<ul style="list-style-type: none"> <li>Global Coordinator (분산 트랜잭션 및 Global Commit을 시작하는 노드)가 분산 트랜잭션에 참여하는 노드들에 대하여 Prepare 하도록 요청하는 단계 [절차]           <ul style="list-style-type: none"> <li>한 노드에서 Commit 요구</li> <li>Coordinator가 Commit Point Site 결정</li> <li>Coordinator가 Prepare 메시지 전송, 원격노드는 Prepare 메시지에 응답 수행</li> </ul> </li> </ul>	<p>서버 (Server) 조정자 (Global Coordinator) 참여자 (Participant) 클라이언트 (Client)</p>	<p>원격 노드에서 요구하는 데이터를 가지고 있는 노드(조정자 또는 참여자)</p> <p>분산 트랜잭션에 참여하는 참여자 목록을 가지며 분산 트랜잭션 및 Global Commit을 시작하는 노드</p> <p>분산 트랜잭션에서 지역 트랜잭션을 수행하는 서버 (조정자의 존재를 알고 그 결정을 따름)</p> <p>분산 트랜잭션 응용을 실행하는 노드</p>
Phase 2) <b>Commit Phase</b>	<ul style="list-style-type: none"> <li>노드에 Commit / Rollback 명령을 보내는 단계 [절차]           <ul style="list-style-type: none"> <li>모두 Commit준비되었다는 메시지 받으면 Commit 명령 수행</li> <li>Coordinator가 노드들로부터 예리 보고 받으면 Rollback 수행</li> </ul> </li> </ul>		

# NoSQL

## 1. 분산 DBMS환경에서의 고확장성 및 고용량 데이터 처리, NoSQL의 개요

- 관계형 DB의 한계를 벗어나, Web2.0의 비정형 초고용량 데이터 처리를 위해 데이터의 읽기보다 쓰기에 중점을 둔, 수평적 확장이 가능하며 다수 서버들에 데이터 복제 및 분산 저장이 가능한 DBMS
- 특징 : Schema-less, 탄력성(Elasticity), 질의가능(Query), 캐싱(Caching)

## 2. NoSQL 특징

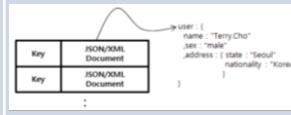
측면	구분	설명
기술적 측면	Schema-less (Document Based)	<ul style="list-style-type: none"> <li>데이터 모델링을 위한 고정된 데이터 스키마 없이 키(KEY)값을 이용해 다양한 형태의 데이터 저장과 접근이 가능한 기능을 이용.</li> <li>데이터를 저장하는 방식에는 크게 컬럼 기반(Column), 값(Value), 문서(Document), 그래프(Graph) 등 네 가지로 구분됨</li> </ul>
	탄력성 (Elasticity)	<ul style="list-style-type: none"> <li>시스템의 일부 장애에도 불구하고 시스템에 접근하는 클라이언트, 응용 시스템의 다른 타임이 없도록 하는 동시에 대용량 데이터의 생성/업데이트, 질의에 대응할 수 있도록 시스템 규모와 성능 확장이 용이하고 입출력의 부하 분산에도 용이한 구조를 갖춤</li> </ul>
	질의가능 (Query)	<ul style="list-style-type: none"> <li>수십 대에서 수천 대 규모로 구성된 시스템에서도 데이터의 특성에 맞게 효율적으로 데이터를 검색, 처리 할 수 있는 질의 언어, 관련 처리 기술, API를 제공</li> </ul>
	캐싱 (Caching)	<ul style="list-style-type: none"> <li>대규모의 질의에도 고성능 응답 속도를 제공할 수 있는 메모리 기반의 캐싱 기술의 적용이 매우 중요하고, 개발 및 운영에서도 투명하고 일관되게 적용 할 수 있는 구조</li> </ul>
RDBMS의 확장 측면	Update/Delete	<ul style="list-style-type: none"> <li>갱신과 삭제 없이 Insert 위주의 사용에 따른 가용성 보장</li> </ul>
	강한 Consistency 불필요	<ul style="list-style-type: none"> <li>Basically Available, Soft State, Eventual consistency 특성</li> </ul>
	노드의 추가/삭제, 데이터 분산에 유연	<ul style="list-style-type: none"> <li>우선 순위가 연관된(Associated) 요소를 큐에 추가</li> </ul>
	유연한 모델링	<ul style="list-style-type: none"> <li>Key-Value Pair, 계층형 데이터, 그래프 타입 데이터 등의 사용</li> </ul>

## 3. NoSQL의 데이터 모델별 분류

모델 분류	설명	종류
<u>Key/Value store</u>	<ul style="list-style-type: none"> <li>키 기반의 get, put, delete 기능제공</li> <li>메모리 기반에서 성능을 우선하는 시스템과 빅데이터를 저장·처리할 수 있는 방식으로 구분</li> </ul>	Redis, Riak, DynamoDB
<u>Column Family Data Store</u>	<ul style="list-style-type: none"> <li>테이블기반, 조인미지원, 컬럼기반으로 구글의 BigTable의 영향</li> </ul>	Hbase, Cassandra
<u>Document Store</u>	<ul style="list-style-type: none"> <li>JSON, XML 형태의 구조적 문서 저장, 조인 미 지원</li> </ul>	MongoDB, Couchbase
<u>Graph Database</u>	<ul style="list-style-type: none"> <li>그래프로 데이터를 표현하는DB</li> <li>시맨틱웹과 온톨로지라는 분야에서 활용되고 발전된 특화된 데이터베이스</li> </ul>	Neo4j, AllegroGraph

# NoSQL

## 4. NoSQL의 데이터 저장구조

구분	설명	개념도	종류																
Key/Value Store	<ul style="list-style-type: none"> <li>Key/Value Store란 <u>Unique한 Key에 하나의 Value를 가지고 있는 형태</u></li> <li>Put(Key,Value), Value := get(Key) 형태의 API로 접근.</li> </ul>	<table border="1"> <tr> <td>Key</td> <td>Value</td> </tr> <tr> <td>Key</td> <td>Value</td> </tr> </table> :	Key	Value	Key	Value	Redis												
Key	Value																		
Key	Value																		
Ordered Key/Value Store	<ul style="list-style-type: none"> <li>Key/Value Store의 확장된 형태로 Key/Value Store와 데이터 저장 방식은 동일하나, <u>데이터가 내부적으로 Key 순서로 Sorting되어 저장</u></li> </ul>	<table border="1"> <tr> <td>Key</td> <td>Column</td> <td>Column</td> <td>Column</td> </tr> <tr> <td></td> <td>Value</td> <td>Value</td> <td>Value</td> </tr> </table> <table border="1"> <tr> <td>Key</td> <td>Column</td> <td>Column</td> <td>Column</td> </tr> <tr> <td></td> <td>Value</td> <td>Value</td> <td>Value</td> </tr> </table> Sorted by Key :	Key	Column	Column	Column		Value	Value	Value	Key	Column	Column	Column		Value	Value	Value	Hbase, Cassandra
Key	Column	Column	Column																
	Value	Value	Value																
Key	Column	Column	Column																
	Value	Value	Value																
Document Key/Value Store	<ul style="list-style-type: none"> <li>Key/Value Store의 확장된 형태로, Key에 해당하는 Value 필드에 데이터를 저장하는 구조는 같으나, <u>저장되는 Value의 데이터 타입이 Document 타입 사용</u></li> <li>Document 타입은 MS 워드와 같은 문서를 이야기하는 것이 아니라 XML, JSON, YAML과 같이 구조화된 데이터 타입으로, 복잡한 계층 구조를 표현.</li> </ul>		MongoDB, Couchbase																

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

[Document 기반]

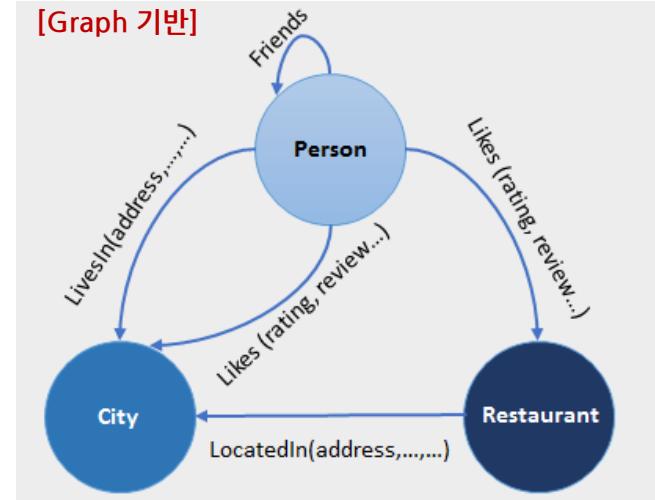
Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1
{ "prop1": data, "prop2": data, "prop3": data, "prop4": data }
Document 2
{ "prop1": data, "prop2": data, "prop3": data, "prop4": data }
Document 3
{ "prop1": data, "prop2": data, "prop3": data, "prop4": data }

[Column 기반]

Row Key	Column Name		
	Key	Key	Key
Value	Value	Value	Value
Column Name			
Key	Key	Key	Key
Value	Value	Value	Value

[Graph 기반]



## 5. NoSQL과 RDBMS의 비교

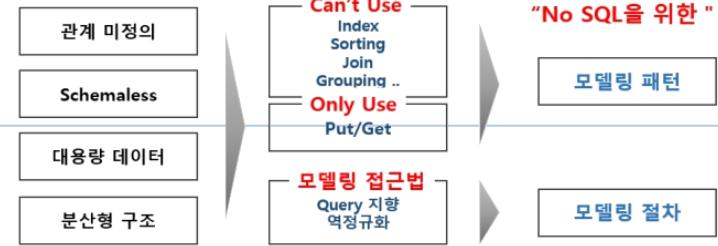
구 분	RDBMS	NoSQL
저장 데이터	기업의 제품판매 정보, 고객정보 등의 핵심 정보의 저장	중요하지 않으나 데이터 양이 많고 급격히 늘어나는 정보 저장
환경측면	일반적인 환경이나 분산 환경 등에 사용	클라우드 컴퓨팅처럼 수천, 수만대 서버의 분산 환경
CAP유형	CA 충족	CP, AP 충족
지원용량	테파 단위 규모	테파 단위 규모
인덱스	복잡(관계형 모델기반)	단순(key-value)기반
데이터접근	SQL	NoSQL(API이용)
특성	ACID	BASE
경합해소	MVCC(Multi Version Concurrency Control)	경합 데이터 분산화 유도(샤딩 및 파티셔닝)
조인 지원	가능 (다양한 기법)	효율적 조인 지원 불가능
장점	무결성, 정합성, 원자성, 독립성, 일관성	비용적인 측면과 확장성 기준
분산처리방법	오라클 RAC 등으로 분산 처리 방법	페타 바이트 수준의 대량의 데이터 처리
특징	고정된 스키마를 가지며 조인 등을 통하여 데이터를 검색함	단순한 키와 값의 쌍으로만 이루어져 인덱스와 데이터가 분리되어 별도로 운영됨
사용 예	오라클, 인포믹스, DB2, MySQL	구글의 Bigtable, 아마존의 Dynamo, 트위터의 Cassandra 등



# NoSQL 데이터 모델링 패턴

NoSQL 특징

시사점



## 1. NoSQL 데이터 모델링 패턴의 개요

- Key/Value 저장 구조에 Put/Get 밖에 없는 단순한 DBMS에 대해 다양한 형태의 Query를 지원하기 위한 테이블을 디자인하기 위한 가이드
- NoSQL은 Order By, Group By, Sorting을 지원하지 않으므로 NoSQL 내에서 데이터 모델링을 통해 이러한 기능들을 구현하는 방법에 대한 가이드 제공

## 2. NoSQL 데이터 모델링 패턴 - 기본적인 데이터 모델링 패턴

패턴	개념도	설명
Renormalization	<p>User Table Join 필요 ZipCode Table 2 tables</p> <p>User Table ZipCode Table User + Zipcode Table ← User + Zipcode 중복저장 2 + 1 tables</p>	<ul style="list-style-type: none"> <li>한 테이블에 Join할 여러 테이블데이터를 중복하여 저장하는 방식</li> <li>테이블간의 Join을 없앨 수 있음</li> <li>장점 : 성능 향상, 쿼리 로직의 복잡도가 낮아짐</li> <li>단점 : 데이터 일관성 문제 발생, 스토리지 용량 증가</li> </ul>
Aggregation	<p>File +fileid +directory +name Photofile +photoid +format moviefile +movieid +title +length</p> <p>Photofile +photoid +format moviefile +movieid +title +length</p> <p>File { type : "photo" 공통 필드 (field, directory, name) meta { format : "JPEG" } }  File { type : "movie" 공통 필드 (field, directory, name) meta { title : "Movie title", format : "mp4", length : "1:40" }</p>	<ul style="list-style-type: none"> <li>Key만 같다면 Row는 각기 다른 형태에도 상관없는 NoSQL 특징 활용, 하나의 테이블로 합칠 수 있음</li> <li>1:N의 복잡한 엔터티의 관계를 하나의 테이블로 바꿈</li> <li>Join 수를 줄여 성능 향상</li> </ul>
Application Side Join	<p>Application</p> <p>① V1=Get(Key) from Table1 ② V2=Get(Table1.FK) from Table2 ③ Return V1, V2</p> <p>Table1</p> <p>Table2</p>	<ul style="list-style-type: none"> <li>반드시 Join을 해야 하는 경우</li> <li>NoSQL이 아닌 application 단에서 로직으로 처리하는 방식</li> <li>Application에서 ①번 수행을 통해 Table2를 연결하는 FK 찾고 ②번 수행하여 원하는 결과값을 찾음</li> <li>조인이 필요한 수만큼 IO발생</li> <li>Denormalization에 비해 스토리지 절약 가능</li> </ul>

# NoSQL 데이터 모델링 패턴

## 2. NoSQL 데이터 모델링 패턴 - 확장된 데이터 모델링 패턴

패턴	개념도	설명														
Atomic aggregation	<pre> graph LR     subgraph Left [Client]         direction TB         A[User Address]         B[User Profile]         C[User ID]         A -- "① Create User" --&gt; C         B -- "① Create User" --&gt; C     end     subgraph Right [Client]         direction TB         D[User { User Address(), User Profile(), User Id() }]         D -- "① Create User" --&gt; D     end     style A fill:#e0f2e0     style B fill:#e0f2e0     style C fill:#e0f2e0     style D fill:#e0f2e0   </pre>	<ul style="list-style-type: none"> <li>여러 테이블에 대한 트랜잭션을 보장해야 할 때 사용하는 방식</li> <li>NoSQL도 한 테이블에는 atomic operation을 보장함을 활용</li> <li>트랜잭션이 보장되어야 하는 테이블을 하나의 테이블로 생성</li> <li>Aggregation:Join을 없애기 위해,</li> <li>atomic aggregation: 장애 · 에러로부터 트랜잭션 불일치 방지를 위해 사용</li> </ul>														
Index Table	<p>Table</p> <pre> graph TD     T1[Table] -- "Index 필요" --&gt; T2[Directory Index Table]     T1[fileID, filename, directory, filelocation]     T2[directory]     T2[fileid, fileID, fileID, fileID, fileID, fileid]   </pre>	<ul style="list-style-type: none"> <li>NoSQL은 Index가 없기 때문에, Index를 위한 별도의 Index table을 만들어서 사용</li> <li>개념도상의 Directory 컬럼에 Index 필요시 Directory를 Key로 하는 별도의 Index table을 생성하여 사용</li> </ul>														
Composite Key	<table border="1"> <thead> <tr> <th>Key</th> <th>value</th> </tr> </thead> <tbody> <tr> <td>windows:etc</td> <td>...</td> </tr> <tr> <td>windows:programfile</td> <td>...</td> </tr> <tr> <td>windows:system32</td> <td>...</td> </tr> <tr> <td>↓ windows:temp</td> <td>...</td> </tr> <tr> <td>msoffice:msword</td> <td>...</td> </tr> <tr> <td>↓ msoffice:powerpoint</td> <td>...</td> </tr> </tbody> </table>	Key	value	windows:etc	...	windows:programfile	...	windows:system32	...	↓ windows:temp	...	msoffice:msword	...	↓ msoffice:powerpoint	...	<ul style="list-style-type: none"> <li>NoSQL은 RDBMS처럼 복합키를 사용할 수 없으므로 Key를 정할 때 하나 이상의 필드를 Delimiter를 이용하여 사용</li> <li>Key 선정 시에는 분산되어 있는 전체 서버에 걸쳐 부하가 골고루 분산될 수 있는 Key를 선정하는 것이 좋음</li> </ul>
Key	value															
windows:etc	...															
windows:programfile	...															
windows:system32	...															
↓ windows:temp	...															
msoffice:msword	...															
↓ msoffice:powerpoint	...															
Inverted Search Index	<p>key</p> <pre> graph TD     T1[key, value]     T1[bcho.tisoty.com/nosql, Nosql, Cassandra, riak]     T1[bcho.tisoty.com/cloud, Amazon, azure, google]     T1[Facebook.com/group/serverside, Amazon, google, riak]     T1[Highscalability.com/bigdata, Nosql, riak]     T1[www.basho.com/riak, riak]      T2[key, value]     T2[Riak, bcho.tisoty.com/nosql, Facebook.com/group/serverside, Highscalability.com/bigdata, www.basho.com/riak]     T2[nosql, Highscalability.com/bigdata, bcho.tisoty.com/nosql]   </pre>	<ul style="list-style-type: none"> <li>Value의 내용을 key로, Key의 내용을 반대로 value로 하는 패턴</li> <li>검색엔진에서 많이 사용</li> <li>검색키워드를 Key로 해서 URL을 value값에 저장하여 검색을 빠르게 수행</li> </ul>														

# NoSQL 데이터 모델링 패턴

## 2. NoSQL 데이터 모델링 패턴 - 계층구조

패턴	개념도	설명																								
Tree Aggregation	<p>Blog Post content comments Mike : "... Sam : "... Eliot : "... Comments라는 Key의 Value 값으로 Tree구조로 이루어진 JSON String을 저장</p> <pre>{   "comments": [     { "by": "Mike", "message": "...", "replies": [       { "by": "Sam", "message": "...", "replies": [] }     ] },     { "by": "Eliot", "message": "...", "replies": [] }   ] }</pre>	<ul style="list-style-type: none"> <li>JSON이나 XML 등을 이용하여 Tree구조를 정의하고 Value에 Tree 구조 자체를 저장하는 방식</li> <li>Tree 자체가 크지 않고 변경이 많이 없는 경우에 적합</li> <li>“계층형 게시판의 답글 Tree구조”에 용이한 구조</li> </ul>																								
Adjacent Lists	<table border="1"> <thead> <tr> <th>Key(name)</th> <th>parent</th> <th>child</th> </tr> </thead> <tbody> <tr> <td>root</td> <td>Null</td> <td>windows,temp</td> </tr> <tr> <td>windows</td> <td>root</td> <td>system32</td> </tr> <tr> <td>temp</td> <td>root</td> <td>null</td> </tr> </tbody> </table>	Key(name)	parent	child	root	Null	windows,temp	windows	root	system32	temp	root	null	<ul style="list-style-type: none"> <li>Linked List를 사용하여 각 Tree노드에 Parent node와 child node들에 대한 포인터를 저장하는 방식</li> <li>IO가 많은 편으로 큰 Tree구조의 저장 및 Read에는 적합하지 않음</li> </ul>												
Key(name)	parent	child																								
root	Null	windows,temp																								
windows	root	system32																								
temp	root	null																								
Materialized Path	<table border="1"> <thead> <tr> <th>Key (materialized path)</th> <th>value</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>...</td> </tr> <tr> <td>A/B</td> <td>...</td> </tr> <tr> <td>A/C</td> <td>...</td> </tr> <tr> <td>A/C/D</td> <td>...</td> </tr> <tr> <td>A/D/E</td> <td>...</td> </tr> <tr> <td>A/C/F</td> <td>...</td> </tr> <tr> <td>A/C/F/G</td> <td>...</td> </tr> </tbody> </table>	Key (materialized path)	value	A	...	A/B	...	A/C	...	A/C/D	...	A/D/E	...	A/C/F	...	A/C/F/G	...	<ul style="list-style-type: none"> <li>Tree의 Root부터 현재 노드까지의 전체 경로를 Key로 저장하는 방법</li> <li>일반적인 Key-value, Ordered Key Value에서는 적용이 힘들며, Document DB(Regular Express 지원)에서는 효과적임</li> </ul>								
Key (materialized path)	value																									
A	...																									
A/B	...																									
A/C	...																									
A/C/D	...																									
A/D/E	...																									
A/C/F	...																									
A/C/F/G	...																									
Nested Sets	<table border="1"> <thead> <tr> <th>Index</th> <th>1</th> <th>2</th> <th>3</th> <th>4</th> <th>5</th> <th>6</th> <th>7</th> </tr> <tr> <th>Node</th> <td>A</td> <td>B</td> <td>C</td> <td>D</td> <td>E</td> <td>F</td> <td>G</td> </tr> </thead> <tbody> <tr> <th>Child node(start, end)</th> <td>2,7</td> <td>null</td> <td>4,6</td> <td>null</td> <td>null</td> <td>7,7</td> <td>null</td> </tr> </tbody> </table>	Index	1	2	3	4	5	6	7	Node	A	B	C	D	E	F	G	Child node(start, end)	2,7	null	4,6	null	null	7,7	null	<ul style="list-style-type: none"> <li>Node가 포함하는 모든 Child Node에 대한 범위 정보를 가지고 있음</li> <li>각 Node는 자신이 포함하는 모든 Sub Tree들이 포함된 Start와 End index를 저장</li> <li>매우 빠른 성능을 보장하나, Update에는 취약 (Index 재배열 필요)</li> </ul>
Index	1	2	3	4	5	6	7																			
Node	A	B	C	D	E	F	G																			
Child node(start, end)	2,7	null	4,6	null	null	7,7	null																			

# NoSQL 데이터 모델링 패턴

## 3. NoSQL 데이터 모델링 절차

No.	절차	절차 별 상세 설명
1	도메인 모델 파악	<ul style="list-style-type: none"><li>먼저 저장하고자 하는 도메인을 파악</li><li>데이터 개체 및 개체간의 관계를 파악하여 ERD 도식화</li><li>사례 설명 : 사례의 블로그 시스템은 사용자ID 기반으로 블로그의 분류(Catalog)를 가지로, 분류별로 글을 작성하고, 파일 첨부, 댓글이 가능함</li></ul>
2	쿼리결과 디자인	<ul style="list-style-type: none"><li>“도메인 모델”을 기반으로 애플리케이션에 의해서 쿼리 되는 결과값 결정</li><li>출력되는 형식을 기반으로 필요한 쿼리 정의</li></ul>
3	패턴을 이용한 데이터 모델링	<ul style="list-style-type: none"><li>Put/Get으로만 데이터를 가져올 수 형태로 데이터 모델을 정의함</li><li>NoSQL 모델링 패턴을 이용하여 테이블 정의 (Denormalization, Aggregation, Application Side Join 등)</li></ul>
4	기능 최적화	<ul style="list-style-type: none"><li>RDBMS의 Index와 같은 이 개념을 NoSQL에서 ‘Secondary Index’를 이용하여 기능의 최적화</li><li>디자인된 테이블에 Document Store, Ordered Key, Secondary Index 등 NoSQL의 특성을 반영</li></ul>
5	후보 NoSQL 을 선정 및 테스트	<ul style="list-style-type: none"><li>NoSQL에 대한 구조 및 특성을 분석한 후에 실제로 부하테스트, 안정성, 확장성 테스트를 거친 후에 가장 적절한 솔루션을 선택</li></ul>
6	데이터 모델 최적화 및 하드웨어 디자인	<ul style="list-style-type: none"><li>선정된 NoSQL을 기반으로 그에 적합한 데이터 모델 최적화 후 이에 맞는 어플리케이션 인터페이스 설계와 구동시킬 하드웨어 디자인을 실시</li></ul>

## 4. NoSQL 도입 시 고려사항

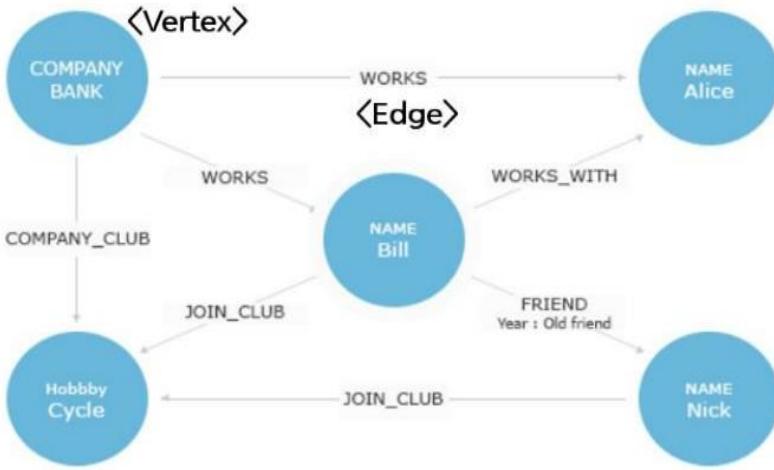
시기	고려사항	설명
도입	NoSQL의 필요성	<ul style="list-style-type: none"><li>현재 업무에 NoSQL이 적절한지 그리고 그 정도의 데이터 용량이 필요한지 체크</li><li>교육과 개발 비용이 높음</li></ul>
	적절한 제품군 선정	<ul style="list-style-type: none"><li>Not Only Sql로 그 범위가 넓고 제품군의 특성이 매우 다양함</li><li>각 NoSQL의 특성을 잘 파악하고 업무와 팀의 특성에 맞는 제품을 선정 필요</li></ul>
	PoC	<ul style="list-style-type: none"><li>반드시 PoC를 통해 NoSQL에 대한 테스트 권장</li></ul>
설계	데이터 모델링	<ul style="list-style-type: none"><li>RDBMS처럼 데이터 모델링을 하면 100% 성능 문제 발생</li><li>NoSQL 사상에 맞는 모델링 필요</li></ul>
	RDBMS와 적절한 혼합	<ul style="list-style-type: none"><li>인덱스성 데이터나 복잡한 쿼리가 필요한 데이터는 RDBMS에 저장하고 실제 데이터 등을 NoSQL에 저장함으로써 RDBMS와 NoSQL을 적절하게 혼합하여 설계</li><li>하나의 NoSQL 제품으로 전체 데이터를 저장하려 해서는 안됨</li></ul>
	하드웨어 설계 병행	<ul style="list-style-type: none"><li>NoSQL은 기본적으로 분산구조임</li><li>디스크 RAID 구성, 대역폭, 몇 개의 Disk 등 HW 설계와 병행</li></ul>
운영	운영 및 백업	<ul style="list-style-type: none"><li>전문적인 administrator 필요</li><li>시스템에 대한 체계적인 모니터링 방안 등의 운영 및 백업 체계 수립</li></ul>

# 그래프 데이터베이스(Graph Database)

## 1. 그래프 이론에 기반한 NoSQL DB, 그래프 데이터베이스의 개요

- 노드(엔티티)와 이들 간의 관계를 **그래프 데이터 모델로 저장하여 트랜잭션을 관리**하는 기능을 API로 제공하는 NoSQL 기반 데이터 베이스
- 기존의 관계형 데이터베이스와 달리 **노드(Node)와 엣지(Edge)로만 구성되어** 있어 연결된 데이터 저장과 유연한 구조의 데이터 모델을 표현할 수 있는 데이터베이스

## 2. 그래프 데이터베이스의 개념도 및 정보표현



주요구성	역할	상세설명
Vertex	Node	<ul style="list-style-type: none"> <li>객체, 실체 즉, 조회하고자 하는 실체를 의미</li> </ul>
Edge	Relationship	<ul style="list-style-type: none"> <li>Vertex를 연결하는 관계를 의미, (Vertex 간 연결을 통해 표현)</li> </ul>
Properties	Attribute	<ul style="list-style-type: none"> <li>Vertex를 설명하는 정보</li> </ul>

특징	설명
직관성제공	- 데이터를 표현하기 위해 그래프모델 사용
일관성보장	- FULL ACID 트랜잭션을 제공
안정성보장	- 커스텀 디스크와 네이티브 스토리지 엔진
가용성보장	- Master와 Slave 간의 클러스터링 기술 지원
확장성보장	- 도메인을 이용한 데이터 분할 저장
사용성보장	- REST interface, JSON 등을 통한 API 이용
그래프계산엔진	- 대규모 그래프(Node, Relations)에 대해 글로벌 그래프 계산(쿼리) 알고리즘을 실행할수 있는기술
API제공	- 그래프 모델을 관리하고 CRUD 처리 및 가시화 기능을 제공(REST)

Node Properties			Nodes that this edge connects			Edge Properties
\$node_id	Name	Age	\$edge_id	\$from_id	\$to_id	StartDate
{"type":"node","id":0}	John	30	{"type":"edge","id":0}	{"type":"node","id":0}	{"type":"node","id":1}	01/01/2013
{"type":"node","id":1}	Mary	28	{"type":"edge","id":1}	{"type":"node","id":1}	{"type":"node","id":2}	05/05/2010
{"type":"node","id":2}	Alice	25	{"type":"edge","id":2}	{"type":"node","id":2}	{"type":"node","id":0}	09/09/2016

Person Node Table    Friends Edge Table

- 그래프 모델로 Key/Value 데이터를 노드/관계로 구성하고 대규모 SNS 데이터 등을 질의

# 그래프 데이터베이스(Graph Database)

## 3. 그래프 데이터베이스의 주요기술

구분	기술	사례
관계와 속성	노드(Node)	<ul style="list-style-type: none"><li>Key에 해당하는 객체</li></ul>
	관계(relation)	<ul style="list-style-type: none"><li>Value에 해당하는 객체</li></ul>
	Incoming	<ul style="list-style-type: none"><li>자기 노드로 들어오는 관계</li></ul>
	Outcoming	<ul style="list-style-type: none"><li>다른 노드로 나가는 관계</li></ul>
SQL 기술	인덱싱	<ul style="list-style-type: none"><li>인덱싱된 노드를 통해 고속 탐색 지원</li></ul>
	트랜잭션	<ul style="list-style-type: none"><li>노드 변경 혹은 관계 추가 시, 트랜잭션 보장</li></ul>
배포 기술	OLTP	<ul style="list-style-type: none"><li>런타임시 응용 프로그램 쿼리를 처리, 요청, 응답 수행 하는 OLTP 속성을 보유한 SOR (레코드 시스템) 데이터베이스 제공</li></ul>
	ETL	<ul style="list-style-type: none"><li>주기적인 ETL(Extract, Transform, and Load) 작업을 위한 오프라인 쿼리 및 분석 기능</li><li>그래프 컴퓨팅 엔진으로 데이터를 이동하여 인메모리 프로세싱 처리</li></ul>

## 4. 그래프 데이터베이스의 종류

종류	활용
Neo4j	- 자바 기반 임베딩 방식이나 REST방식으로 제공
AllegroGraph	- RDF를 저장할 수 있는 그래프 DB
S2그래프DB	- 카카오에서 개발하여 오픈소스로 제공되는 SNS 분석용 그래프 DB
Amazon Neptune	- SPARQL 지원하며 대량으로 연결된 데이터세트를 효율적으로 탐색하는 쿼리 제공
JanusGraph/Titan	- 크고 작은 그래프를 최대 수천억 개의 정점과 엣지로 저장 및 트래버스 할 수 있도록 설계된 그래프 DB



# BASE

## 1. 가용성과 성능을 중시하는 특성을 중시하는 분산시스템의 속성, BASE의 개요

- ACID와 대조적으로 가용성과 성능을 중시하는 특성을 가진 분산 시스템의 특성

## 2. BASE속성

구분	내용
<u>B</u> asically <u>A</u> vailable	<ul style="list-style-type: none"> <li>• 가용성을 중시, Optimistic Locking 및 큐 사용</li> <li>• 다수의 실패에도 가용성을 보장, 다수의 스토리지에 복사본 저장</li> </ul>
<u>S</u> oft-State	<ul style="list-style-type: none"> <li>• 노드의 상태는 외부에서 전송된 정보를 통해 결정됨</li> <li>• 분산 노드 간 업데이트는 <u>데이터가 노드에 도달한 시점에 갱신</u></li> </ul>
<u>E</u> ventually Consistent	<ul style="list-style-type: none"> <li>• 일시적으로 데이터가 비일관적인 상태가 되어도 <u>일정시간 후에는 데이터가 일관성이 있는 상태</u>가 되는 성질</li> </ul>

## 3. BASE속성과 ACID속성의 비교

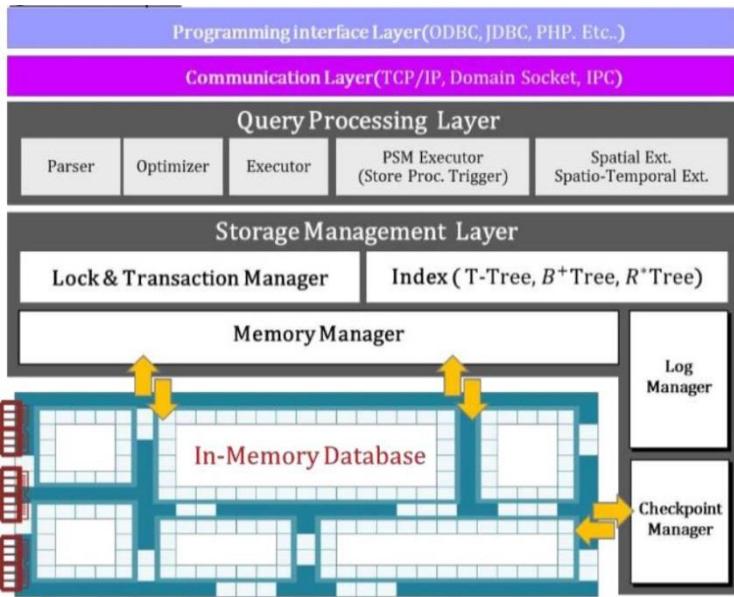
속성	BASE	ACID
적용분야	NOSQL	RDBMS
범위	<u>시스템 전체에 대한 특성</u>	<u>트랜잭션에 한정</u>
일관성측면	약한 일관성	강한 일관성
중점사항	<u>Availability</u>	<u>'Commit'에 집중</u>
시스템측면	성능에 초점	엄격한 데이터관리
효율성	쿼리디자인이 중요	테이블 디자인이 중요

# 인메모리 데이터베이스(In-memory Database)

## 1. 데이터 처리 속도 향상 위한 인메모리 데이터베이스(In-memory Database)의 개요

- 디스크 기반 DBMS가 제공하는 데이터 관리 관련 모든 기능을 메인 메모리를 이용하여 제공하는 데이터베이스 시스템
- 전통적으로 디스크에 저장하던 데이터베이스 대신 **데이터의 일부 또는 전부를 메인 메모리에서 관리하는 데이터베이스**

## 2. 인메모리 데이터베이스의 구성 및 주요 기술

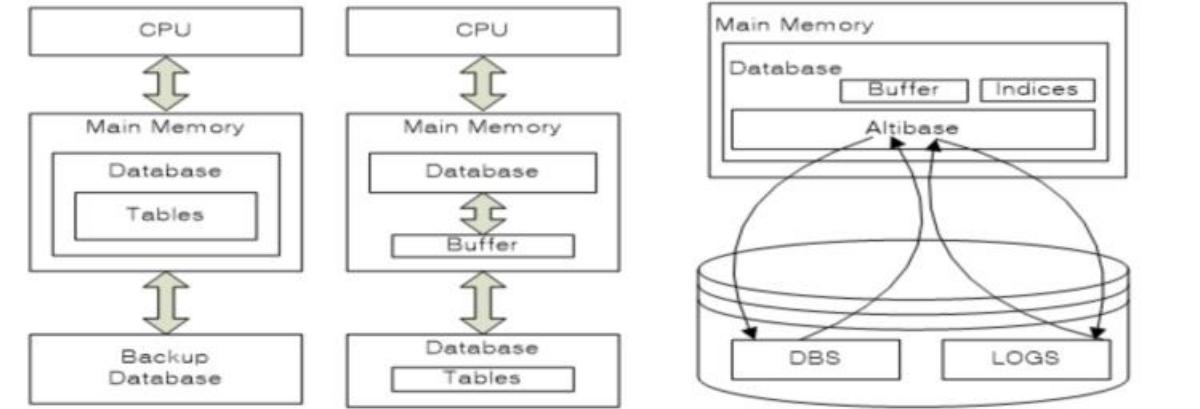


구분	기술	내용
인메모리 분산 DB 기술	키-값 기반의 데이터 모델	<ul style="list-style-type: none"> <li>관계형 DB와 달리 테이블과 컬럼의 사전 정의 없이도 키-값으로 런타임에 추가 될 수 있는 유연성 제공</li> </ul>
	비공유 구조(shared nothing) 파티셔닝	<ul style="list-style-type: none"> <li>테이블을 여러 파티션으로 겹치지 않게 분할 후 다수의 노드에 할당 관리하여 데이터 용량의 확장성 제공</li> </ul>
	데이터 분할 및 복제	<ul style="list-style-type: none"> <li>다수의 데이터 처리 요청을 클러스터 내의 노드들로 동시에 처리하여 트랜잭션의 처리량 증가</li> <li>장애 대응을 위한 데이터 복제 방식</li> </ul>
인메모리 데이터 분석 기술	듀얼 포맷 기술	<ul style="list-style-type: none"> <li>OLTP : 행 기반 저장 기술 적용</li> <li>OLAP : 열 기반 저장 기술 적용</li> </ul>
	다차원 표현 기술	<ul style="list-style-type: none"> <li>다차원 데이터 저장 구조로 데이터 중복 방지, 용량 절감</li> <li>OLAP 분석 및 연산 지원</li> </ul>
인메모리 저장/처리 기술	계층적 데이터 관리	<ul style="list-style-type: none"> <li>자주 사용되는 데이터들을 분류하여 계층별로 데이터 저장</li> <li>Hot Data : In-Memory 컬럼 저장 공간 적재</li> <li>Active Data : 플레이시 메모리 적재</li> <li>Cold Data : 디스크 적재</li> </ul>
	압축 저장 기술	<ul style="list-style-type: none"> <li>데이터 압축 저장으로 데이터의 스캔 및 필터 시, 작업량 축소로 처리 성능 향상</li> </ul>
	SIMD Vector 프로세싱	<ul style="list-style-type: none"> <li>한번의 CPU 명령어를 통해 여러 개의 컬럼 값들을 한 번에 처리</li> </ul>

# 인메모리 데이터베이스(In-memory Database)

## 3. 디스크 기반 DBMS와 인메모리 기반 DBMS의 비교

구분	디스크 기반 DB	인메모리 DB
데이터 저장 장치	디스크	메인 메모리
운영 목표	데이터의 안정적 운영	트랜잭션의 빠른 실행
동시성 제어	데이터 접근 트랜잭션 중심	인덱스에 대한 동시성 제어
Index 알고리즘	B-tree, B+-tree	Hashing, T-tree
회복 기법	Undo/Redo 로그 관리	하드웨어 적인 회복기법
데이터 용량	컬럼, 저장, 압축(디스크기반), 데이터 중복관리	컬럼별 저장 및 압축(메모리기반), 데이터 중복없음
정보의 반영 속도	ETL을 통한 배치작업 및 데이터마트 구성 및 캐쉬작업으로 인한 지연 발생	별도의 요약테이블 및 캐쉬작업 없이 인메모리데이터 처리엔진 통해 실시간 검색 가능
계산 속도	컬럼별 데이터 저장방식 + 캐쉬	컬럼별 데이터 저장방식 + 압축
유연성	제한적(데이터 모델 변경 취약)	유동적(데이터 구조 유동적 구성)
애플리케이션 플랫폼	분석용도로만사용가능(트랜잭션 용도로 사용불가)	기간계와 정보계를 하나로 합병가능, 대용량 빅데이터 분석에 용이



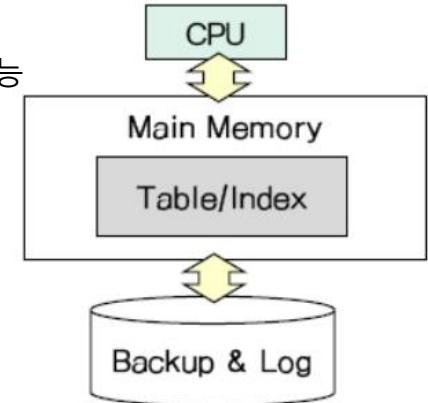
# MMDB(Main Memory Data Base)

## 1. MMDB(Main Memory Data Base)의 개요

- 데이터베이스 전체를 주기억장치에 상주시켜 데이터베이스 연산을 처리하는 고성능 DB
- 디스크의 접근 없이 직접 메모리 접근을 통해 Data 를 관리함으로써 고성능 Transaction 처리가 가능
- 메모리 상주 데이터베이스의 데이터검색을 위한 T-Tree 검색기법 사용

## 2. MMDB(Main Memory Data Base)의 개념도 및 기술요소

구분	기술요소	설명
인덱스 구조	T-트리 인덱스	<ul style="list-style-type: none"><li>B-트리 장점과 AVL-트리 장점을 결합 데이터 접근계산 최소화</li><li>회전연산비용을 메모리 구조 및 인덱스 구조변경 최소화</li></ul>
질의 최적화	메모리 기반	<ul style="list-style-type: none"><li>데이터 접근 비용 계산 평가비용이 디스크 기반에 비해 낮음</li></ul>
지속성 기술	백업DB	<ul style="list-style-type: none"><li>메모리 내부 데이터를 디스크형 백업용DB에 동기화</li></ul>



## 3. 기존 DB와 MMDB 비교

구분	디스크 기반 DB	MMDB
개념	<p>The diagram shows a vertical stack of three boxes: 'CPU' at the top, 'Main Memory' in the middle containing a 'Buffer' box, and 'Table / Index' at the bottom. Yellow double-headed arrows connect the CPU to Main Memory and Main Memory to Table / Index.</p> <ul style="list-style-type: none"><li>버퍼만 메인 메모리에, DB테이블은 디스크에 존재</li></ul>	<p>The diagram shows a vertical stack of three boxes: 'CPU' at the top, 'Main Memory' in the middle containing a 'Table/Index' box, and 'Backup &amp; Log' at the bottom. Yellow double-headed arrows connect the CPU to Main Memory and Main Memory to Backup &amp; Log.</p> <ul style="list-style-type: none"><li>메인 메모리 내에 DB테이블, 인덱스 등 존재</li></ul>
저장 장치	<ul style="list-style-type: none"><li>디스크</li></ul>	<ul style="list-style-type: none"><li>주기억장치 (메인 메모리)</li></ul>
Indexing	<ul style="list-style-type: none"><li>B-Tree, B+Tree</li></ul>	<ul style="list-style-type: none"><li>Hashing, T-Tree</li></ul>
설계방향	<ul style="list-style-type: none"><li>Disk 접근횟수 최소화</li><li>데이터 Clustering 향상</li></ul>	<ul style="list-style-type: none"><li>CPU처리 시간 최소화</li><li>메모리 공간 사용 최소화</li></ul>

# NewSQL

\* 참고자료 : NewSQL: Towards Next-Generation Scalable RDBMS for Online Transaction Processing (OLTP) for Big Data Management

## 1. NoSQL처럼 높은 확장성과 성능을 갖춘 RDB, NewSQL 의 개요

- 기존 RDB관점에서의 ACID(Atomicity, Consistency, Isolation, Durability)를 보장하며, NoSQL 시스템의 특징인 확장성과 유연성을 겸비한 DBMS (RDBMS 계열에서 NoSQL 특성을 흡수)
- NewSQL is a class of modern relational database management systems that seek to provide the same scalable performance of NoSQL systems for online transaction processing (OLTP) read-write workloads while still maintaining the ACID guarantees of a traditional database system

## 2. RDBMS vs NoSQL, NewSQL 비교

구분	Traditional RDBMS	NoSQL	NewSQL
SQL	Supported	Not supported	<u>Supported</u>
Machine dependency	Singe machine	Multi- machine/Distributed	Multi- machine/Distributed
<u>DBMS type</u>	<u>Relational</u>	<u>Non- relational</u>	<u>Relational</u>
Schema	Table	Key-value, column- store, document store	both
Storage	On disk + cache	On disk + cache	On disk + cache
Properties support	ACID	CAP through BASE	<u>ACID</u>
Horizontal scalability	Not supported	Supported	Supported
Query Complexity	Low	High	Very High
Security concern	Very high	Low	Low
Big volume	Less performance	Fully supported	Fully supported
OLTP	Not fully supported	supported	Fully supported
Cloud support	Not fully supported	supported	Fully supported

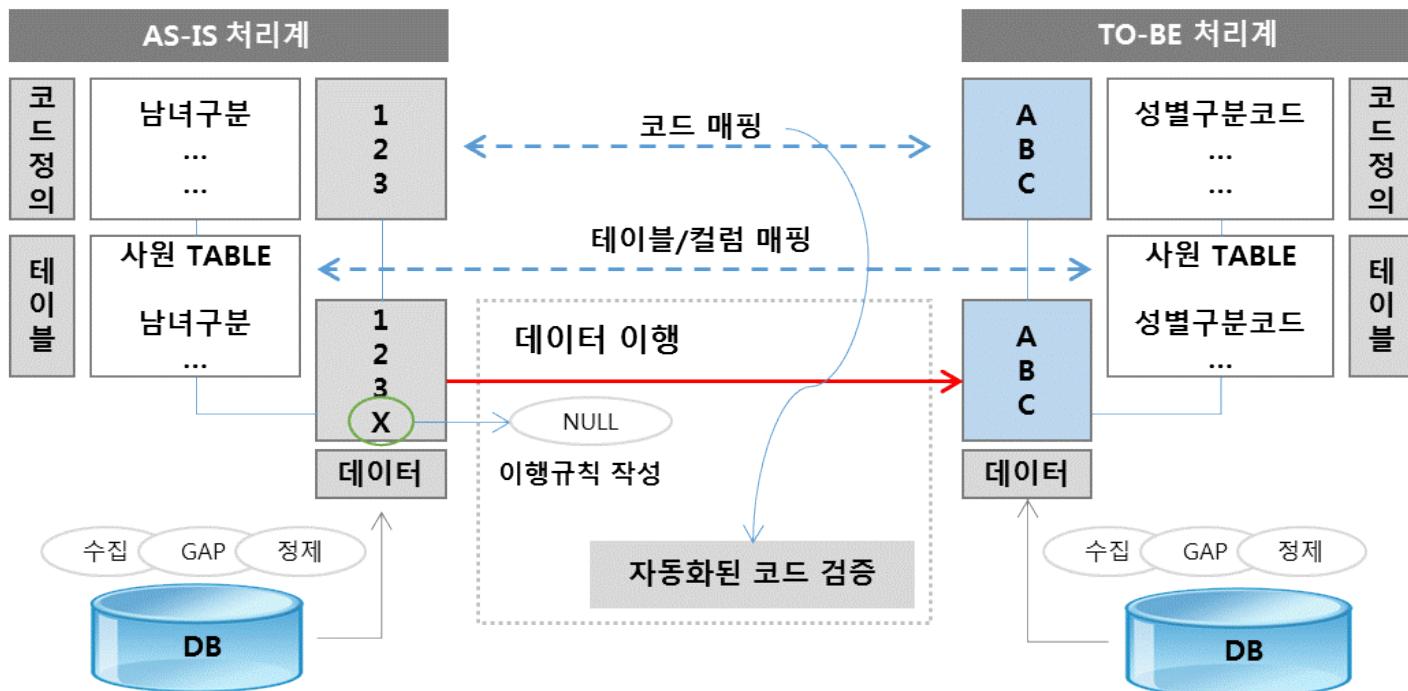
	Old SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL	Yes	No	Yes
ACID transactions	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Performance / big volume	No	Yes	Yes
Schema-less	No	Yes	No

# Data Migration (전환, 이관)

## 1. 시스템의 변화에 따른 데이터의 이관 작업, Data Migration의 개요

- 현 시스템의 데이터를 목표 시스템의 데이터 구조에 맞게 데이터를 매핑하는 규칙을 정의하고 추출, 변화하여 이관하는 활동
- 신규/통합 시스템 적용, HW/SW 성능향상 등을 위해 서로 다른 컴퓨터 시스템, 스토리지, 데이터베이스 사이에서 데이터를 이전하는 행위

## 2. Data Migration의 개념도

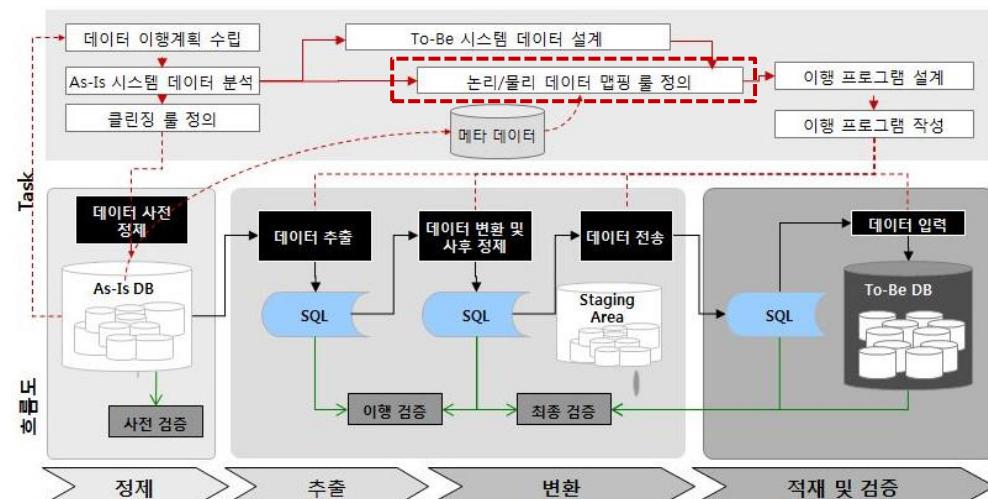
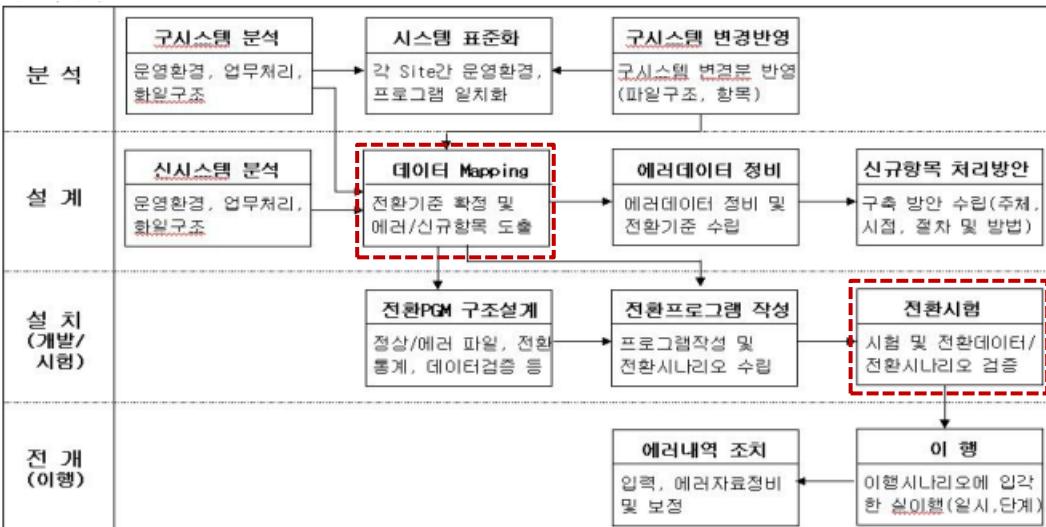


# Data Migration (전환, 이관)

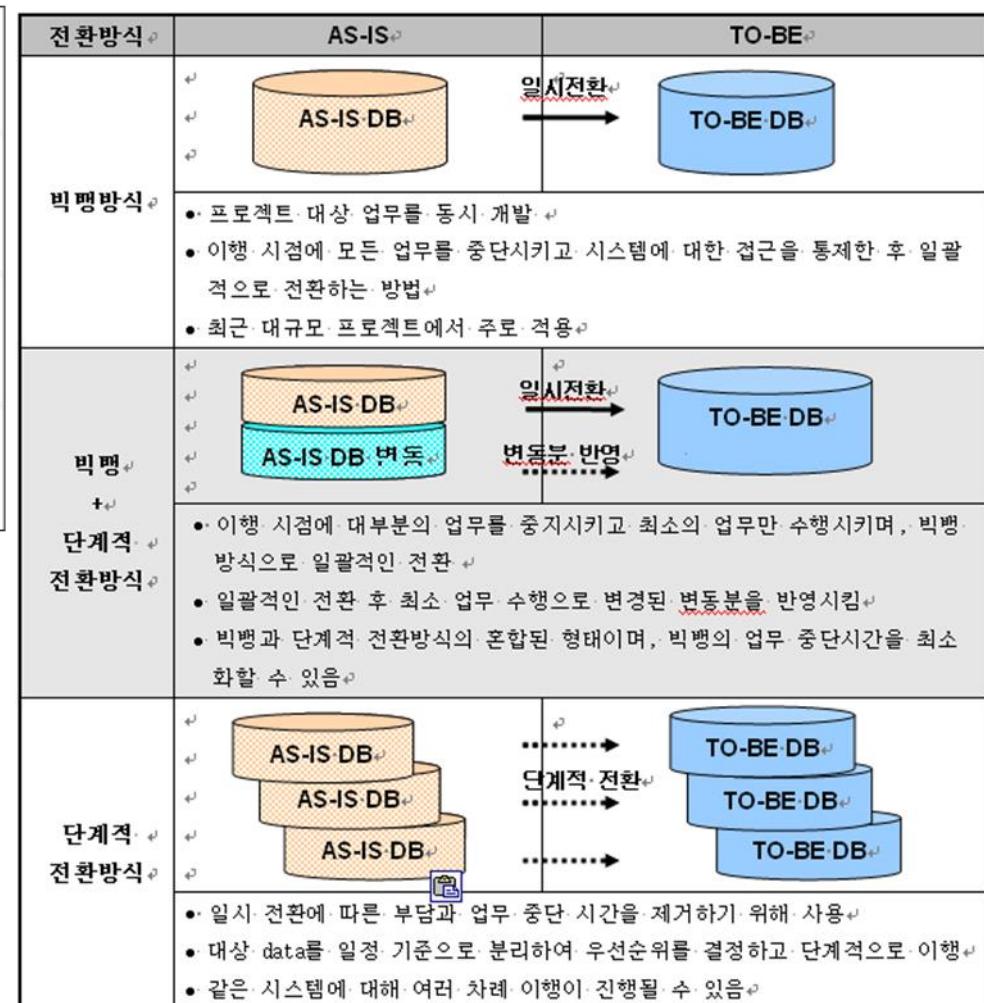
절차(Mapping, 전환 PG, 시험)

빅뱅, 단계적, 하이브리드

## 3. Data Migration 절차

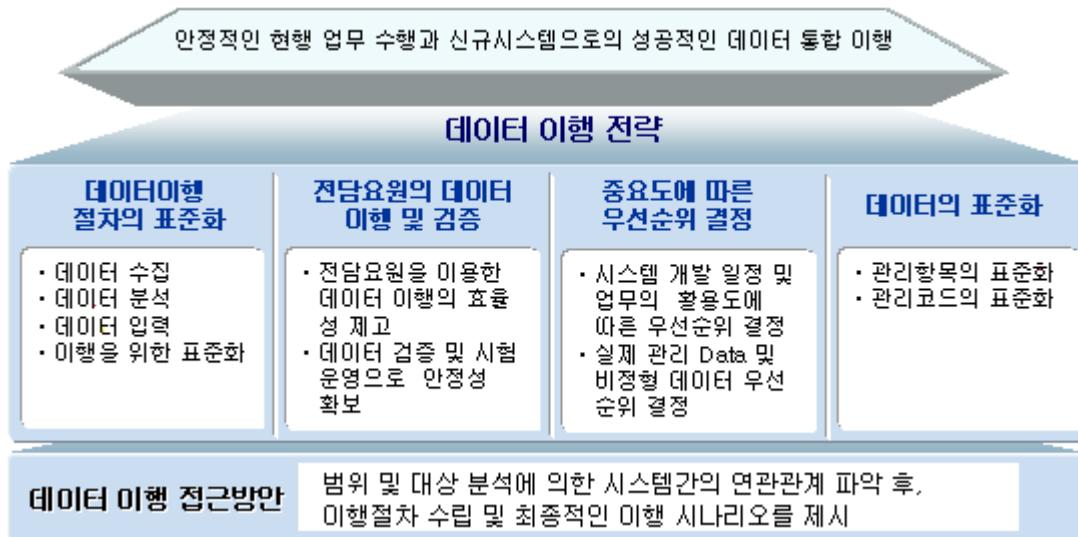


## 4. 데이터 전환방식



# Data Migration (전환, 이관)

## 5. 데이터 Migration 전략



# DMBOK (Data Management Body of Knowledge)

## 1. 데이터관리를 위한 국제 표준, DMBOK 의 개요

- 데이터 관리 분야의 best practices로 일반적으로 받아 들여지는 프로세스 및 지식 영역을 엮은 **데이터관리를 위한 국제 표준 지침서**
- 특정 방법 및 기술이 아닌, 표준 산업 관점의 데이터관리 기능 및 개념, Best Practice 제공

## 2. DMBOK의 프레임워크와 구성요소



기능	설명	특징
1) Data Governance	데이터 관리 계획 및 관리감독, 통제	데이터 관리 원칙 및 조직 기능
2) Data Architecture Management	전사 데이터 모델링, Value Chain 분석 관련 데이터 아키텍처	전사 아키텍처 관점 기능
3) Data Development	데이터 분석, 모델링, Database 디자인 및 구현	SDLC에 따른 활동
4) Database Operations Management	데이터 획득, 복구, 튜닝, 유지, 파기	Data Lifecycle
5) Data Security Management	데이터보안 표준, 분류, 관리, 인증, 감사	데이터보안 기능
6) Reference & Master Data Management	외부코드, 내부코드, 고객 데이터, 제품 데이터, Dimension 관리	데이터 일관성 보장
7) Data Warehousing & Business Intelligence Management	DW 및 BI를 위한 아키텍처, 구현, 기술훈련 및 지원, 모니터링 및 튜닝	데이터 분석 지원
8) Document & Content Management	데이터 획득 및 보관, 백업 및 복구, 컨텐츠 관리, 추출, 유지	컨텐츠 데이터 관리
9) Meta Data Management	메타 데이터 아키텍처, 통합, 통제, 제공	메타 데이터 관리 방안
10) Data Quality Management	데이터 품질 기준, 분석, 측정, 개선	데이터 품질 관리 가이드

## 3. DMBOK 데이터관리의 환경요소

DMBOK 7개 환경요소



기능	설명	범위
<b>1) Goals &amp; Principles</b>	각 기능들의 직접적인 비즈니스 목표와 기능의 성과를 가이드하는 기초적인 원칙들	Vision & Mission, Business Benefits, Strategic Goals, Specific Objectives, Guiding Principles
<b>2) Activities</b>	각 기능은 하위 수준의 활동으로 더욱 더 분해됨	Phases/Tasks/Steps, Dependencies, Sequence & Flow, Use Case Scenarios, Trigger events
<b>3) Deliverables</b>	각 기능의 산출물로서 생성되는 정보와 물리적 데이터베이스와 문서들	Inputs & Outputs, Information, Documents, Databases, Other Resources
<b>4) Roles &amp; Responsibilities</b>	기능을 수행하고 감독하는 비즈니스와 IT 역할들과 각 역할의 구체적인 책임들	Individual Roles, Organizational Roles, Business & IT Roles, Qualifications & Skills
<b>5) Practices &amp; Techniques</b>	프로세스들을 수행하고, 산출물을 생산하는 데 사용되는 공통적이고 대중적인 방법과 기술	Recognized Best Practices, Common Approaches, Alternative Techniques
<b>6) Technology</b>	기술(주로 소프트웨어 도구들), 표준, 프로토콜 등을 지원하는 카테고리	Tool Categories, Standards & Protocols, Selection Criteria, Learning Curves
<b>7) Organization &amp; Culture</b>	조직, 문화 관련 환경 요소	Critical Success Factors, Reporting Structures, Management Metrics, Values/Benefits/Expectations, Attitudes/Styles/Preferences, Rituals/Symbols, Heritage

# 데이터 표준화

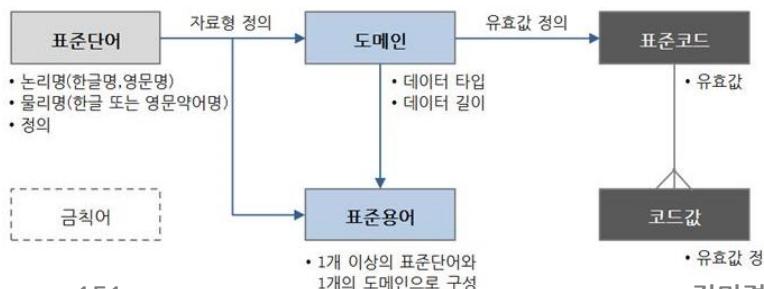
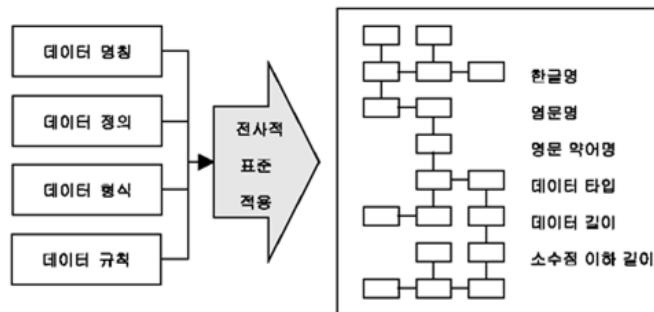
명칭, 정의, 형식, 규칙

## 1. 데이터 표준화 개념

- 시스템 별로 산재해 있는 데이터 정보 요소에 대한 **명칭, 정의, 형식, 규칙에 대한 원칙을 수립**하여 이를 전사적으로 **데이터의 품질을 향상시키려는 지속적인 활동**

## 2. 데이터 표준화 목적

목적	설명
의사소통의 혼란방지	동일한 데이터에 대해 동일한 명칭을 사용함으로 다양한 계층간에 명확하고 신속한 의사소통이 가능하게 함
<b>데이터의 정확성 및 품질 확보</b>	데이터 정의, 형식, 규칙을 표준에 맞게 적용함으로 데이터 사용 및 활용에 대한 오류를 방지하고 데이터의 품질을 향상
<b>개발 생산성 향상</b>	표준화된 데이터를 사용함에 따라 모델링 작업 시 단어 선택 시간을 단축시 키고 재사용 기회를 높여 개발 생산성을 향상시킨다
상호운용성 증대	전사 차원에서 데이터 표준을 관리하고 적용함에 따라 시스템간 인터페이스 연계가 용이하고 별도의 데이터 변환 또는 정제 작업을 수행하지 않아도 됨

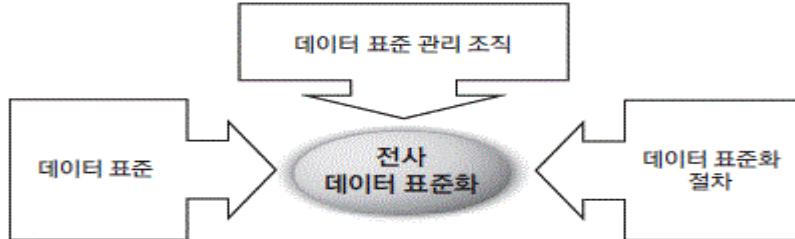


코드ID	코드	코드값
001	KOR	한국
001	USA	미국

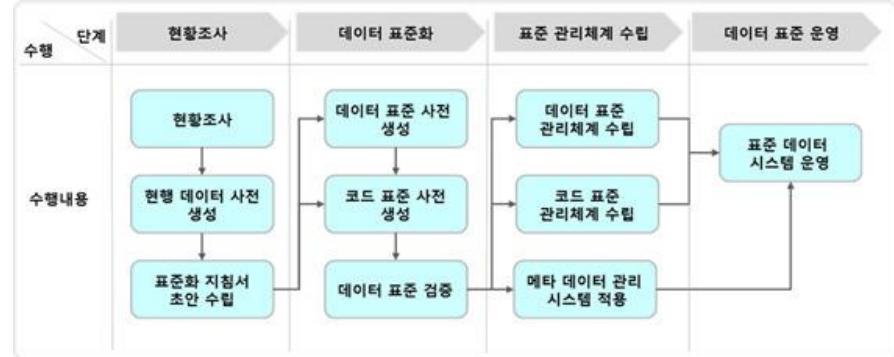
# 데이터 표준화

단어, 용어, 도메인, 표준코드

## 3. 데이터 표준화 구성요소

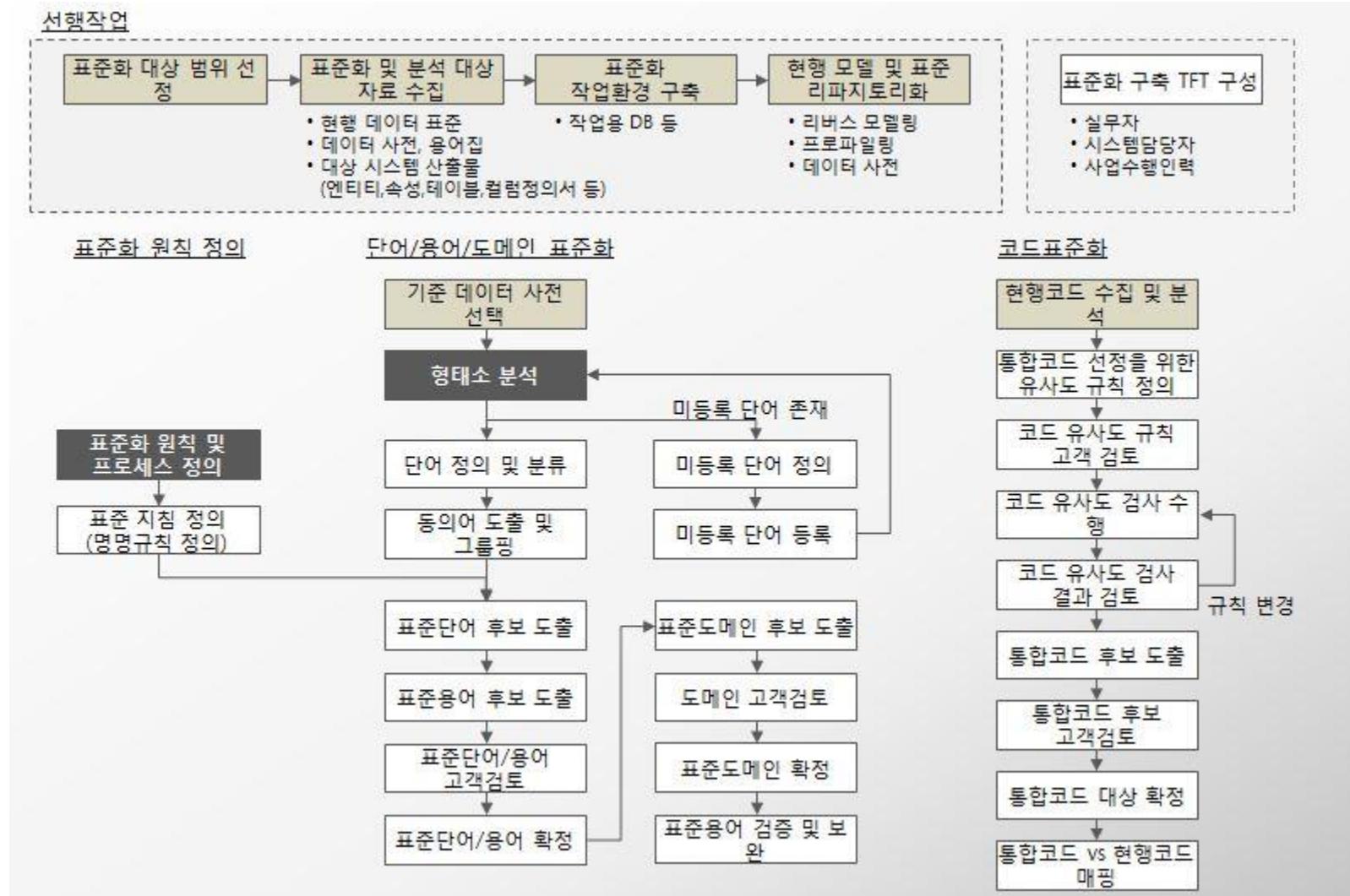


- 데이터 표준



데이터 표준 종류	설명	예시
표준 단어	<ul style="list-style-type: none"> <li><b>표준 용어를 구성하는 단어</b>에 대한 표준을 정의함으로써 용어에 대한 한글명과 영문명 및 영문 약어를 일관되게 정의할 수 있게 함</li> </ul>	단어: 고객 영문명: CUSTOMER 영문약어: CUST
표준 용어 (단어의 조합 + 도메인 적용)	<ul style="list-style-type: none"> <li>업무적으로 사용하는 용어에 대한 표준을 정의함으로써 용어 사용 및 적용에 대한 혼란을 방지하고 원활한 커뮤니케이션을 촉진. 표준 용어는 업무적 용어와 기술적 용어가 있음.</li> <li>단어와 도메인 표준을 먼저 수행 후 용어 표준을 수행 함(표준 용어는 단어의 조합에 도메인을 적용해야 하므로)</li> </ul>	고객명(단어의 조합) : 고객 + 명: CUST_NM VARCHAR(20)  '명' 을 VARCHAR(20) 도메인으로 지정
표준 도메인	<ul style="list-style-type: none"> <li>동일한 성질을 가진 칼럼의 <b>데이터 타입 및 데이터 길이</b>를 일관되게 관리</li> <li>칼럼에 대한 성질을 그룹핑 한 개념. 도메인은 크게는 문자형, 숫자형, 일자형, 시간 형으로 분류할 수 있고, 더 세부적으로는 명, 주소, ID(이상 문자형), 금액, 율, 수량(이상 숫자형) 등으로 분류할 수 있음</li> </ul>	명 : 사람의 이름을 나타내는 도메인으로 VARCHAR(20) 데이터 타입 지정
표준 코드	<ul style="list-style-type: none"> <li>코드는 도메인의 한 유형으로서 특정 도메인 값(코드값)이 이미 정의되어 있는 도메인.</li> <li>따라서 코드에 대한 표준은 다른 표준과는 달리 <b>데이터 값, 즉 코드값</b>까지 미리 정의해야 함.</li> </ul>	신호등색 : 노란색: C001 초록색: C002 빨간색: C002

# 데이터 표준화



# 데이터 품질

값, 구조, 프로세스

도정통량최

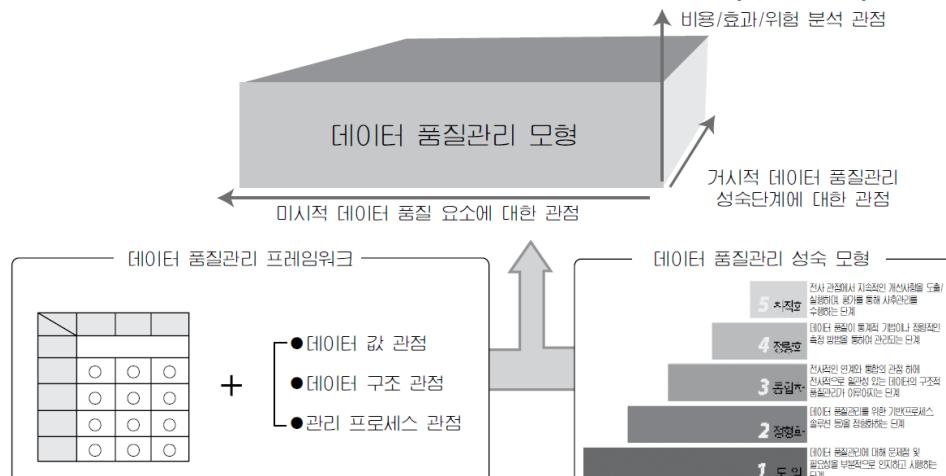
## 1. 실시간 기업의 자산 데이터의 신뢰성 확보를 위한 데이터 품질관리의 개요

- 조직의 목적 달성을 위해 관리되는 데이터가 조직 구성원, 고객 등 **데이터 이용자의 만족을 충족시킬 수 있는 수준**

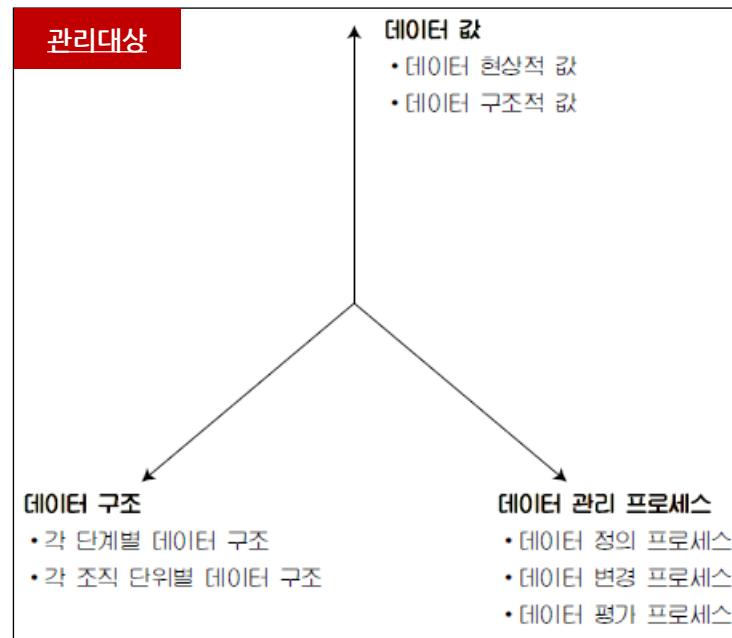
## 2. 데이터 품질 조건 (미국 데이터품질 법)

조건	설명
이용편리성(Utility)	제공데이터는 이용자들이 쉽게 구하고 이용 만족도가 높아야함
안전성 및 신뢰성(Integrity)	관리적, 기술적, 물리적 보안에 입각한 데이터베이스 보호 및 접근권한 관리를 통해 데이터의 안전성 및 신뢰성이 확보되어야 함
공익성(Objective)	데이터는 근거 및 출처가 분명하여야 하며, 근거 및 출처는 과학적 혹은 통계 기법에 의하여 증명될 수 있어야 함

## 3. 데이터 품질관리 모형의 3가지 관점과 관리대상(품질요소)



관점	내용
미시적	데이터 품질관리의 각 요소를 확인하고 요소별 데이터 품질 향상 방안을 도출
거시적	전사 조직 측면에서 데이터 관리의 성숙도 단계를 정의하고 각 조직의 성숙도를 측정하여 상위 단계로 발전하도록 유도
부가가치적	데이터 품질관리의 비용·효과·위험 모형을 개발하여 각 조직의 상황에 맞는 데이터 품질관리 방안을 제시



# 데이터 품질

값, 구조, 프로세스

## 3. 데이터 품질관리 프레임워크

조직	대상	데이터	데이터구조	데이터 관리 프로세스	
CIO 관점 (개괄적 관점)		데이터 관리정책 (모델관리정책, 품질관리정책(데이터 품질관리 규정, 문서관리규정))			
DA 관점 (개념적 관점)	표준데이터 (용어, 명칭, 도메인, 코드)	개념모델 참조모델		데이터 표준관리 요구사항관리	
Modeler 관점 (논리적 관점)	모델데이터 (데이터모델 데이터, DB구조 데이터)	논리모델 (개념모델 상세화)		데이터 모델, 흐름관리 (모델 형상관리, 원천 → 목표 데이터 과정 흐름 및 이력관리)	
DBA 관점 (물리적 관점)	관리데이터 (데이터 흐름 데이터, 정합성 검증데이터 등)	물리모델 데이터베이스		데이터베이스 관리 (백업, 보안, 튜닝, 재구성, 모니터링)	
User 관점 (운용적 관점)	업무데이터 (원천 데이터, 운영 데이터, 분석 데이터)	사용 View		데이터 활용관리	

## 4. 데이터 품질관리 프레임워크의 구성요소

품질관리 대상	설명	비고
데이터 값	<ul style="list-style-type: none"> <li>기관 및 기업에서 사용되어지는 전산화된 데이터 또는 전산화에 필요한 데이터 값</li> </ul>	<ul style="list-style-type: none"> <li>데이터 현상적 값</li> <li>데이터 구조적 값</li> </ul>
데이터 구조	<ul style="list-style-type: none"> <li>데이터가 담겨 있는 모양이나 틀</li> <li>데이터를 취급하는 관점에 따라 사용자 View(양식, 보고서, 화면, 장표 등), 모델(개괄, 개념, 참조, 논리, 물리), DB 파일 형태로 보임 (Data Architecture 관점에서 데이터 Schema)</li> </ul>	<ul style="list-style-type: none"> <li>각 단계별 데이터 구조</li> <li>각 조직 단위 별 데이터구조</li> </ul>
데이터 관리 프로세스	<ul style="list-style-type: none"> <li>데이터 및 데이터 구조의 품질을 안정적으로 유지 개선하기 위한 활동으로 절차, 조직, 인력 등을 포함</li> </ul>	<ul style="list-style-type: none"> <li>데이터 정의 프로세스</li> <li>데이터 변경 프로세스</li> <li>데이터 평가 프로세스</li> </ul>

조직	설명
CIO/EDA	<ul style="list-style-type: none"> <li>데이터 관리의 총괄, 데이터 관리 정책 및 지원 마련, 데이터 관리자 간 이슈사항 조정</li> </ul>
DA	<ul style="list-style-type: none"> <li>표준 개발 및 형상 관리, 검증 및 표준화 절차를 수립하고 운영</li> </ul>
Modeler	<ul style="list-style-type: none"> <li>해당 기능 영역의 데이터 요구사항 및 이슈 사항을 조정하고 통합</li> <li>해당 기능 영역의 비즈니스 요건을 토대로 데이터 모델링을 수행하고 데이터 표준을 확인하고 적용</li> </ul>
DBA	<ul style="list-style-type: none"> <li>DB 설계, DB와 데이터의 형상관리를 수행하고 DB의 모니터링, 튜닝, 보안관리를 수행</li> </ul>
User	<ul style="list-style-type: none"> <li>데이터 소스, 운영 데이터 및 분석 데이터를 활용</li> <li>데이터에 대한 추가 요건을 요청</li> </ul>

# 데이터 품질 - 관리 기법

클렌징(보정), 프로파일링(발견)

## 1. 데이터 Cleansing

- 불완전하고 **오류가 있는 데이터를 보정**하여 정제된 데이터를 만드는 과정 또는 방법

구성요소	상세설명	비고
데이터 변환	<ul style="list-style-type: none"><li>코드체계 변환: 다양한 형태의 코드값을 단일 형태로 변환</li><li>형식 재구성: 다양한 형식의 데이터 값을 단일 형식으로 변환</li><li>수학적 변환: 다양한 형식의 단위 값을 단일 단위 값으로 변환</li></ul>	<p>예: 성별 예: 일자 예: 화폐단위</p>
데이터 파싱	<ul style="list-style-type: none"><li>데이터 정제 규칙을 적용하기 위해 유의미한 취소 단위로 분할하는 작업</li></ul>	<p>예: 주민등록번호, 생년월일 등의 유효성 체크</p>
데이터 보강	<ul style="list-style-type: none"><li>변화, 파싱, 수정 표준화 등을 통해 추가 정보를 반영하는 작업</li></ul>	

## 2. 데이터 프로파일링

- 데이터 품질보증 목적을 위해 데이터베이스 내에 있는 **부정확한 것들을 발견하는 프로세스**
- 데이터의 새로운 용도 개발 혹은 다른 프로젝트 **데이터를 마이그레이션 할 때 유용**한 기술

구성요소	상세설명
컬럼 속성 분석	<ul style="list-style-type: none"><li>하나의 컬럼에 저장된 값 분석</li><li>룰 세트: 각 컬럼에 유효한 값이 무엇인지 정의한 각 컬럼의 속성 예) department_id 값은 숫자이며 100~999 사이의 값, 이때 010은 틀린 값임</li></ul>
구조 분석	<ul style="list-style-type: none"><li>테이블을 구성하는 컬럼간의 관계 분석</li></ul>
단순한 데이터 룰 분석	<ul style="list-style-type: none"><li>값의 결합이 가능한 비즈니스 객체의 여러 컬럼에 걸친 값을 분석하여 데이터를 정제</li></ul>
복잡한 데이터 룰 분석	<ul style="list-style-type: none"><li>여러 비즈니스 객체에 연관된 복잡한 데이터 룰을 분석</li></ul>
값 룰 분석	<ul style="list-style-type: none"><li>집계값을 통하여 부정확한 데이터의 존재는 찾는 것 예를 들어, 각 컬럼값의 발생 빈도를 보면 어떤 값의 빈도가 너무 낮거나 높은 것</li></ul>



# DQM3 (Data Quality Management Maturity Model)

기준, 프로세스, 성숙도

유효성, 활용성

## 1. 데이터 품질관리 성숙모형, DQM3

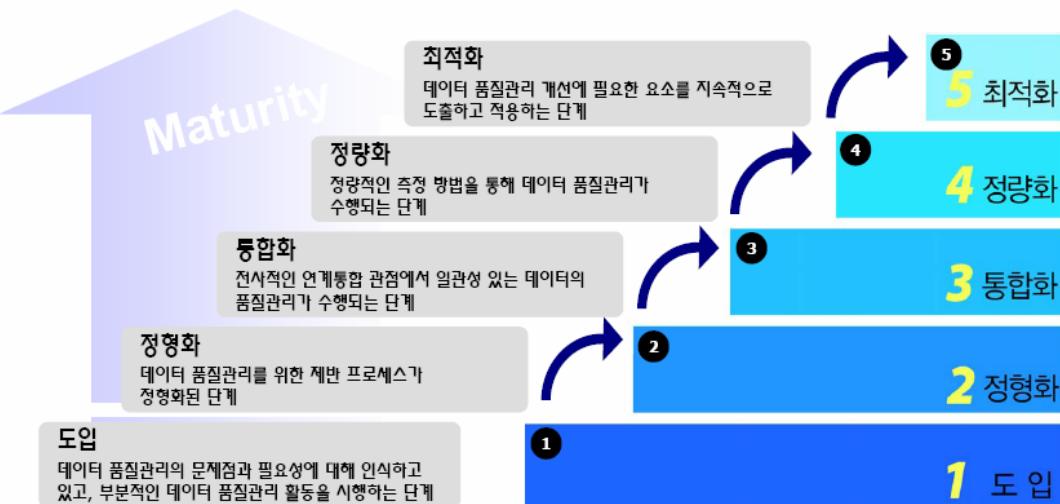
- 데이터 품질관리 수준을 진단하고 개선 과제 및 방안을 단계적, 체계적으로 제시하기 위해 개발된 데이터 품질관리 프로세스의 성숙도 모델 (우리나라가 세계 최초로 개발하여 운영하는 데이터 베이스 품질 인증 시스템)

## 2. 데이터 품질관리 성숙모형의 구성

- 품질기준**: 정확성, 일관성, 유용성, 접근성, 적시성, 보안성 측면으로 정의
- 품질관리 프로세스**: 요구사항관리, 데이터구조관리, 데이터흐름관리, 데이터베이스 관리, 데이터 활용관리, 데이터 표준관리, 데이터 오너십관리, 사용자 뷰 관리
- 품질관리 성숙수준**: '도입-정형화-통합화-정량화-최적화'의 5단계로 구분

## 3. 데이터 품질관리 성숙모형 단계 및 구성요소

### ▶ 데이터 품질관리 성숙 단계



### ▶ 데이터 품질진단과 데이터 품질관리 프로세스 정비를 융합

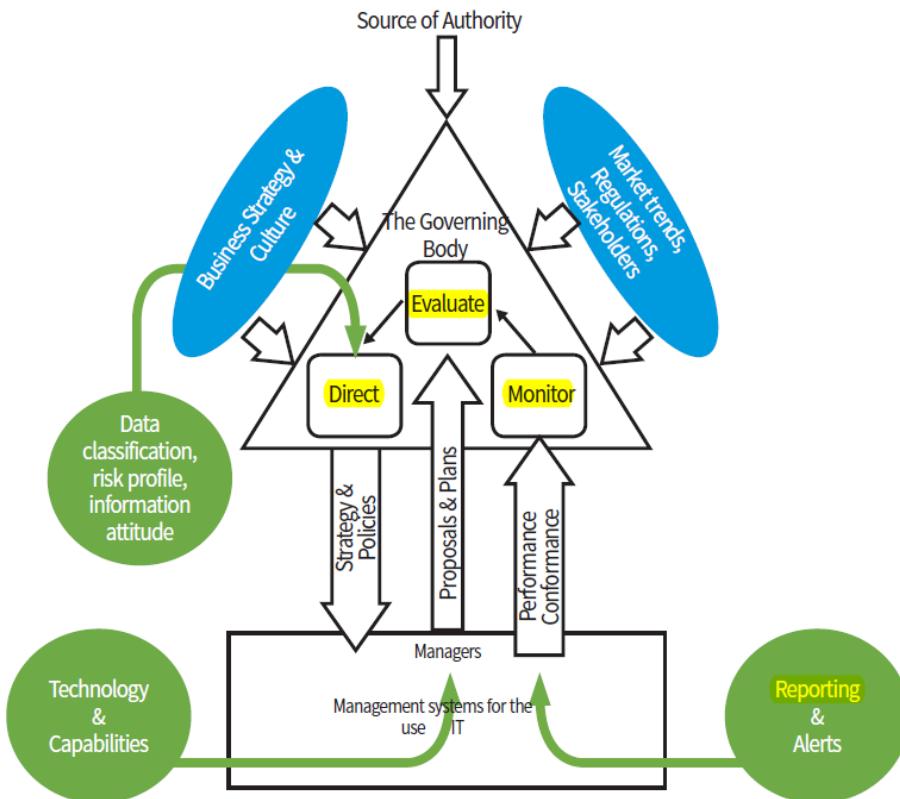


# 데이터 거버넌스

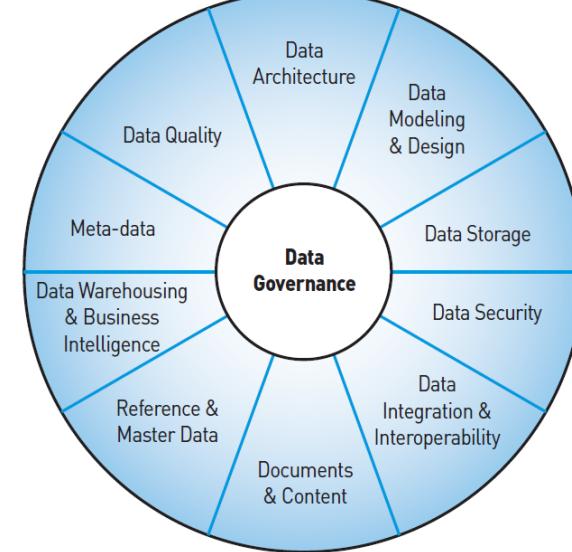
## 1. 전사의 데이터 통제, Data Governance 의 개요

- 전사의 데이터 정책, 지침, 표준, 전략, 방향 등에 근거하여 **기업의 목표달성을 위해 데이터에 대한 의사결정을 지원**하는 매커니즘
- 현재 데이터 아키텍처의 정책, 지침, 방향, 표준 등에 근거하여 데이터에 대한 변화, 진화를 결정하기 위한 의사결정을 지원
- 목적: 고품질 데이터의 확보와 관리 및 적극적인 활용을 통해 조직의 다양 한 가치 창출에 지속적으로 기여하는 것

## 2. 국제표준화기구의 데이터 거버넌스 개념



## 3. 데이터 거버넌스와 관련된 조직내 데이터 관리기능



- 데이터 **구조 관리**(data architecture management)
- 데이터 **모델링과 디자인**(data modeling and design)
- 데이터베이스 저장관리(database storage)
- 데이터 **보안관리**(data security management)
- 데이터 **통합과 상호운영** 관리 (data integration and interoperability)
- 콘텐츠** 관리(content management)
- 마스터 데이터** 관리(master data management)
- 비즈니스 **인텔리전스 관리**(business intelligence management)
- 메타 데이터** 관리(metadata management)
- 데이터 **품질** 관리(data quality management)

# DQC(DB Quality Certification)

데이터, 관리, 보안

## 1. DQC(DB Quality Certification)의 개요

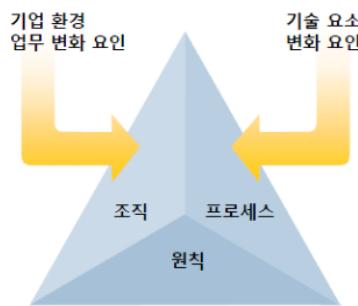
- 공공·민간에서 개발하여, 활용중인 정보시스템의 데이터 품질을 확보하기 위해 **데이터 자체품질과 데이터 관리체계의 품질, 보안체계를 심사 인증**

구분	주요내용	인증수준
<b>데이터 인증 (DQC-V : Database Quality Certification-Value)</b>	<ul style="list-style-type: none"><li>공공·민간에서 구축·활용 중인 데이터베이스를 대상으로 도메인, 업무규칙을 기준으로 데이터 자체에 대한 품질 영향요소 전반을 심사·심의하여 인증</li></ul>	<ul style="list-style-type: none"><li>데이터 정합율 기준</li><li>Platinum(99.977% 이상)</li><li>Gold(97.700% 이상)</li><li>Silver(95.510%)</li></ul>
<b>데이터 관리 인증 (DQC-M : Database Quality Certification- Management)</b>	<ul style="list-style-type: none"><li>행정 및 업무지원, 의사결정 및 정책지원, 지식의 활용 및 제공 등을 목적으로 운영되고 있는 정보 시스템의 데이터 관리 수준을 심사하여 인증</li></ul>	<ul style="list-style-type: none"><li>정확성, 일관성, 유용성, 접근성, 적시성, 보안성</li><li>레벨 1~5</li></ul>
<b>데이터 보안 인증 (DQC-S, Database Quality Certification-Security)</b>	<ul style="list-style-type: none"><li>접근제어, 암호화, 취약점분석, 작업결재 등 데이터베이스 보안에 대한 기술요소 전반을 심사하여 인증하는 것을 의미</li></ul>	<ul style="list-style-type: none"><li>DB접근, 암호화, 작업결재, 취약점 분석</li><li>레벨 1~4</li></ul>

# 데이터 거버넌스

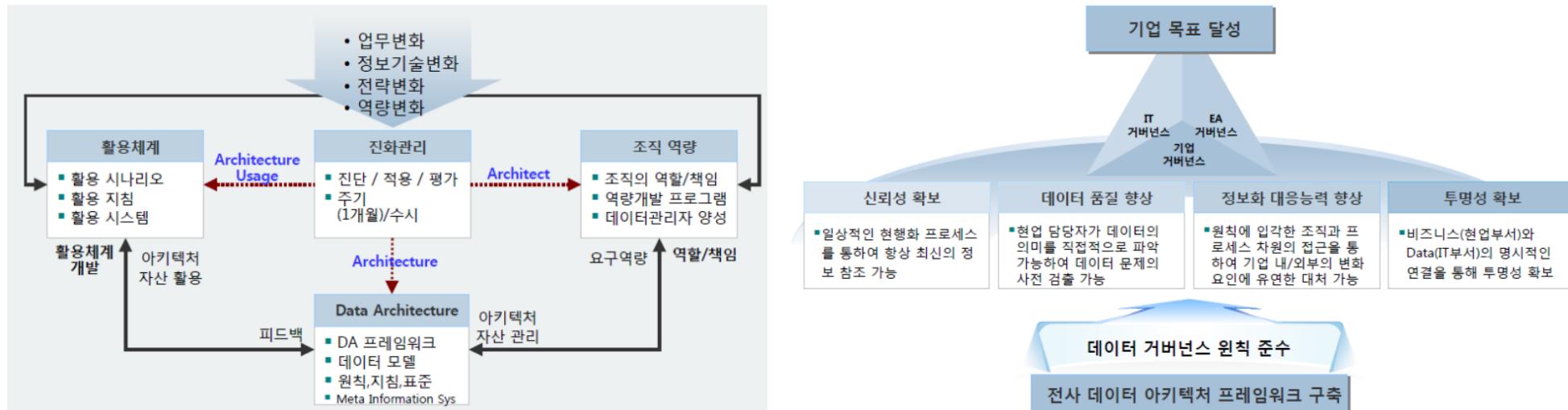
원칙, 조직, 절차

## 4. Data Governance의 구성 및 구성요소



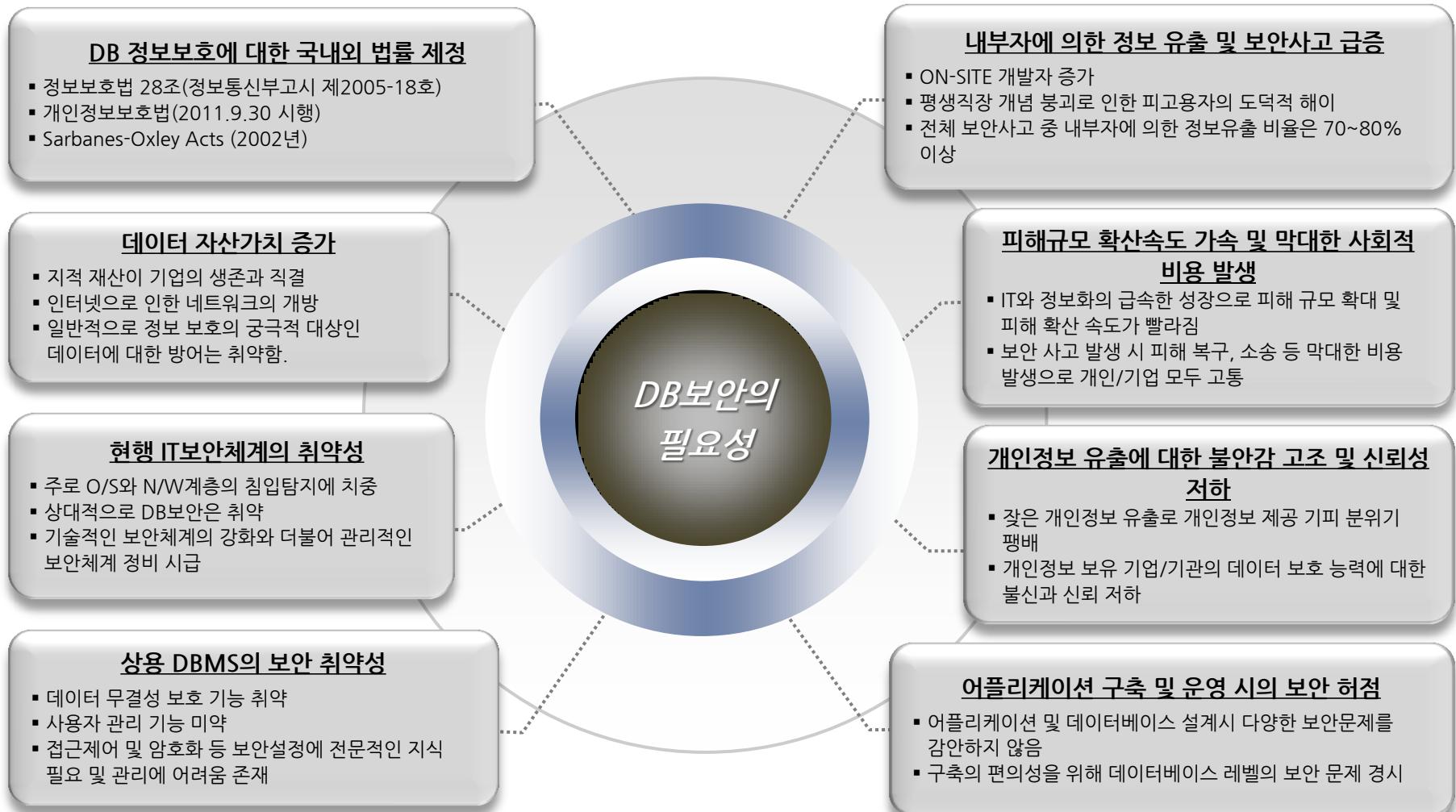
구성요소	주요내용
원칙 (Principle)	- 데이터를 유지 관리하기 위한 지침, 가이드라인 - 보완, 변경/유지관리, 품질기준
조직 (Organization)	- 데이터를 관리할 조직 체계(역할, 책임) - Data Architect, 데이터 관리자, 데이터베이스 관리자 조직역량
절차 (Process)	- 조직이 데이터 관리를 위해 수행하는 활동과 체계 - 활용 시스템, 작업절차, 측정활동, 감시활동

## 5. 데이터 거버넌스 구축을 위한 참조모델



# [보안] 데이터 베이스 보안

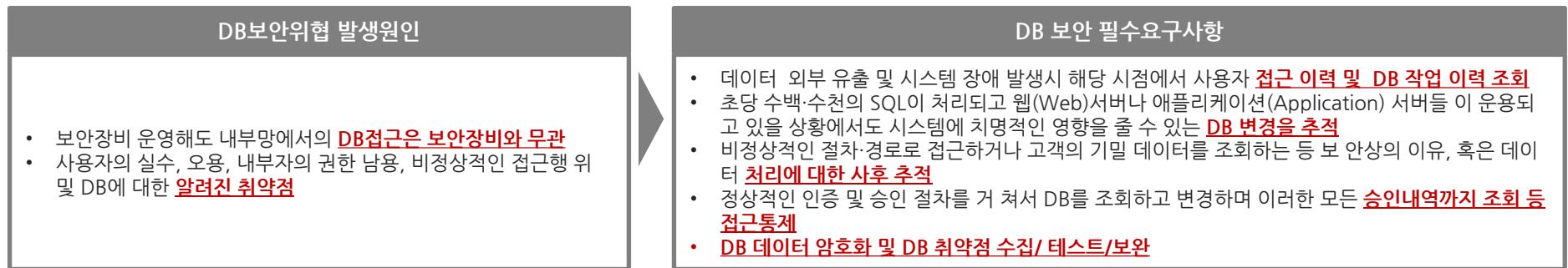
## 1. 데이터 베이스 보안의 필요성



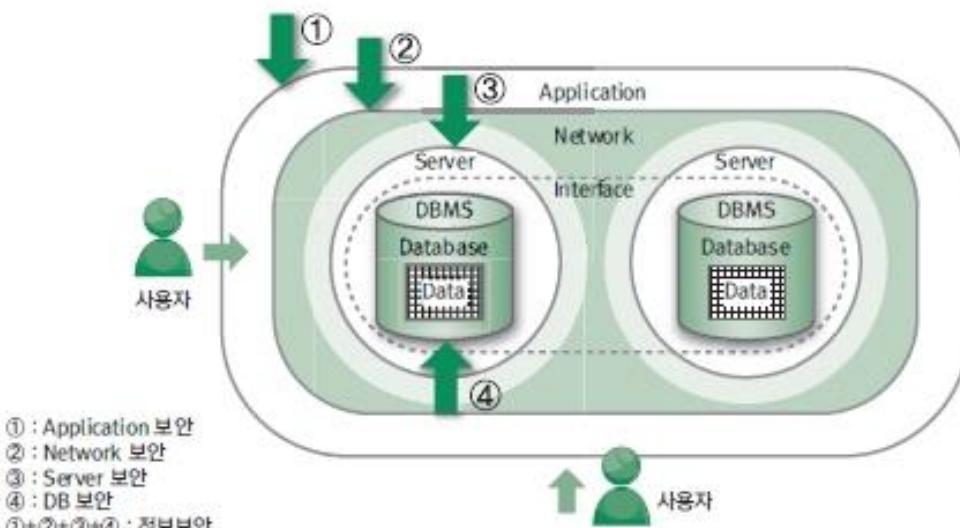
# [보안] 데이터 베이스 보안

## 1. 데이터 베이스 보안의 개요

- DB와 DB 내의 저장된 데이터의 비인가 변경, 파괴, 노출 및 비밀관성 등을 발생시키는 사건이나 위협에서 보호하는 조치 및 활동
- 저장된 데이터에 대한 인증, 기밀성, 무결성, 가용성 유지 위한 관리적, 물리적, 기술적 보안조치 및 활동
- DB 보안은 정보보안의 범주에서도 DB와 DB 내에 저장된 데이터의 보호에 초점



## 2. 정보보안과 DB와의 관계



## 3. 정보보안의 목적

- DB에 대한 위협요소를 식별하고 분석하여 이를 적절하게 통제함으로써 불확실한 이벤트의 발생 **위험을 감소시키고 수습 가능한 수준(Acceptable Level)으로 최소화**시키는 것
- DB에 저장된 데이터를 공개/노출, 변조/파괴/훼손, 지체/재난 등의 위협으로부터 보호하여 기밀성, 무결성, 가용성을 확보하는 것

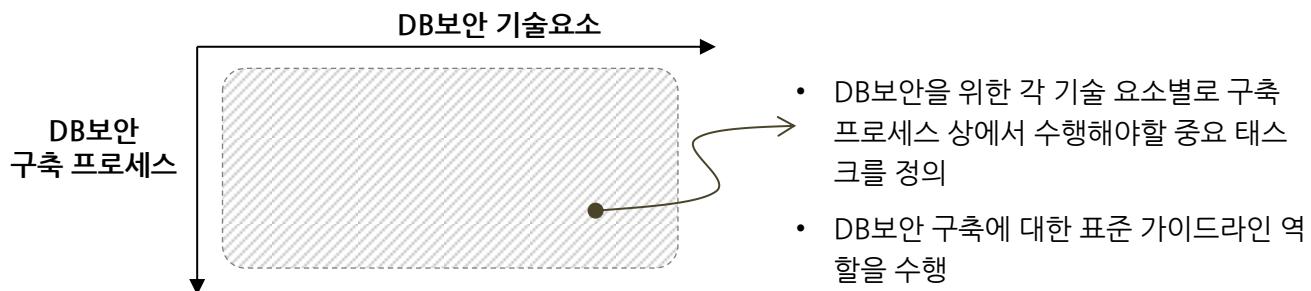
### [DB 보안대상 및 통제]

구분	항목
대상	데이터, DB 애플리케이션, 저장장치, DB 시스템, DB 서버, 관련 네트워크 링크 등
통제	기술적, 절차적, 관리적, 물리적 보안 통제

# [보안] 데이터 베이스 보안

## 4. DB 보안의 3대 요소

요소	설명
기밀성	<ul style="list-style-type: none"><li>데이터 분류 (접근 가능한 데이터 선별), 접근권한 분류(데이터 접근 자격 분류)</li><li>기밀성 제공 방법 : 접근제어, 암호화</li></ul>
무결성	<ul style="list-style-type: none"><li>접근제어(데이터 변경 권한 확인), 의미적 무결성(갱신된 데이터가 의미적으로 정확한지 검증)</li><li>하드웨어나 소프트웨어 고장시 연산적 무결성</li><li>동시 트랜잭션이 수행되는 동안 DB 논리적 일치성 보장</li></ul>
가용성	<ul style="list-style-type: none"><li>정당한 권한을 가진 사용자나 애플리케이션에 대해 원하는 데이터에 대한 원활한 접근을 제공하는 서비스를 지속 할 수 있도록 보장</li></ul>



# [보안] 데이터 베이스 보안 프레임워크

## 1. DB 보안 프레임워크의 개요

- 실제적이며 효용성 있는 구체적인 DB보안 가이드라인 제공, DB보안의 개선·보완 방향에 대한 기준선 역할
- DB보안 통제를 구축하는데 대한 시간, 비용, 전문인력 부재 등의 문제 해결을 지원
- DB보안을 구축하는데 필요한 효과적인 통제 수단과 이들의 구축절차 및 주요 태스크 등을 일목요연하게 정리하여 프레임워크로 제공

## 2. DB보안 기술 요소의 도출





# [보안] 데이터 베이스 보안 프레임워크

접근제어, 암호화, 작업결재, 취약점 분석

## 3. DB보안 기술 요소

기술요소	설명	비고
접근제어	<ul style="list-style-type: none"> <li>사용자가 DBMS에 로그인하거나 SQL 수행 시 기 정의된 보안규칙에 따라 권한 여부를 판단하여 통제하고, 로깅이 필요한 SQL에 대하여 SQL 수행(CRUD)과 관련된 정보를 저장(로깅)하여 부당한 조작 여부를 판단하는 보안통제 방법</li> <li>대체로 DBMS 자체가 이미 강력한 보안기능을 제공하고 있어서, 해킹보다는 DB에 대한 접근 권한을 가진 사용자가 권한을 남용하여 정보를 유출하거나 변조하는 것 막는데 주력</li> <li>접근제어 규칙을 보수적으로 적용할 수록 우회 가능성은 감소</li> <li>구성원들의 보안 의지와 의식 변화가 관건</li> </ul>	<ul style="list-style-type: none"> <li>개인정보보호법 및 동법 시행령, 개인정보의 안전성 확보조치 기준(행정안전부고시 제2011-제43호) 등에서 접근통제, 암호화, 접속기록 등에 대해 규정</li> <li>법규에 따른 암호화 대상 정보 - 고유식별정보, 비밀번호, 바이오정보 등</li> </ul>
암호화	<ul style="list-style-type: none"> <li>데이터를 암호화하여 저장하고, <b>권한이 있는 사람 혹은 서버마이 해당 데이터를 복호화</b> 할 수 있도록 하여 데이터를 보호하는 기술</li> <li>접근제어와 병행 사용하여 더 큰 효과를 얻을 수 있음</li> <li>업무 요구사항이나 목적, 데이터 중요도, 법적 요건 등에 의해 적용 결정</li> <li>암호화 이후 키관리나 복호화 등 통제 상태에 대한 지속적인 모니터링이 필요</li> <li><b>성능 이슈를 고려한 데이터 구조 설계와 관련 어플리케이션 개발 필요</b></li> </ul>	
작업결재	<ul style="list-style-type: none"> <li>보안상 중요한 데이터에 대한 접근 및 처리 등의 <b>DB작업에 대하여 사전 승인을 거치도록 하는 결재 정책을 정의하고, 기안-승인 프로세스를 거쳐 실행 권한을 부여 받은 후 DB작업이 이루어지도록 하는 것</b></li> <li>워크플로우 관점에서의 관리적 보안의 한 형태</li> <li>사전에 DB작업이 기안되어야 하므로 작업 중 실수를 최소화 할 수 있음</li> <li>DB작업 시간과 횟수 등을 사전 승인 받을 수 있어 감사 이력으로 적합</li> </ul>	
취약점분석	<ul style="list-style-type: none"> <li>데이터베이스 시스템 상의 취약점 존재 여부를 점검·추출하고 확인하여 해결책을 제시함으로써 그 취약점을 개선·제거하는 것</li> <li>사용자의 실수, 오용, 내부자의 권한 남용 및 데이터베이스에 대해 알려진 취약점들에 기인하는 외부 혹은 내부자로부터의 주요 기밀정보 유출에 대한 방어에 있어 매우 중요</li> <li>최신 취약점 정보의 주기적 업데이트가 중요. 취약점 검출은 솔루션을 이용한 자동화 가능</li> </ul>	

## 4. DB보안 구축 프로세스

구축 프로세스	설명	주요 산출물 예시
기획 단계	<ul style="list-style-type: none"> <li>DB보안을 구축하기 위한 기본적인 계획을 수립하고 DB보안 정책과 지침·가이드라인 등을 개발하는 단계</li> <li>보안 대상 데이터 식별·분류, DB보안을 위한 원칙·지침 등 정의, DB보안 담당 조직의 구성과 역할, 책임 정의, 보안통제 운영 및 관리절차 등을 정의하여 DB보안 정책과 지침·가이드라인 등에 반영</li> <li>DB보안 구축 계획 수립</li> </ul>	<ul style="list-style-type: none"> <li>DB보안 구축 계획</li> <li>DB보안 정책·지침·가이드라인 등</li> </ul>
설계 단계	<ul style="list-style-type: none"> <li>접근제어 규칙, 암호화 규칙, 작업결재 규칙 등과 같이 각 기술요소별로 구현을 위한 상세 규칙을 정의하고 시험계획을 수립하는 단계 (취약점 분석 계획 수립 포함)</li> </ul>	<ul style="list-style-type: none"> <li>(기술요소별)설계서</li> <li>(기술요소별)시험계획서</li> </ul>
구축 단계	<ul style="list-style-type: none"> <li>설계 단계의 수행 결과에 따라 실제로 각 기술요소를 구현하고 시험하는 단계</li> </ul>	<ul style="list-style-type: none"> <li>(기술요소별)시험결과서</li> <li>(기술요소별)구현내역서</li> </ul>
운영 단계	<ul style="list-style-type: none"> <li>구현된 기술요소들의 작동 상태 및 결과를 모니터링하고, 보완·개선 사항을 도출하여 시행하며, 이와 관련된 보안규칙을 유지관리하는 단계</li> <li>구현된 기술요소들을 계속적으로 유지하고 개선·보완하기 위한 기술적·관리적 보안이 주로 이루어짐</li> </ul>	<ul style="list-style-type: none"> <li>DB보안 점검 체크리스트 및 점검 결과</li> <li>(기술요소별)운영 보고서</li> </ul>

# [보안] 데이터 베이스 보안 프레임워크

## 3. DB 보안 프레임워크와 구성요소

DB보안 기술요소

DB보안  
구축 프로세스

	접근제어	암호화	작업결재	취약점 분석
기획	접근제어 규칙 정의	암호화 규칙 정의	DB 보안 정책수립	
설계	우회 방지	<ul style="list-style-type: none"> <li>• 원본 데이터 삭제</li> <li>• 제약 사항 유지</li> <li>• 암호화 키 관리</li> </ul>	작업결재 규칙 정의	취약점 분석 계획
구축			우회 방지	모의 해킹
		환경 보완		내부보안 감사
		보안 적용 시험		
운영		보안규칙관리		취약점 수집
		사용자 로그 관리		취약점 제거
		모니터링		
		운영 리뷰		

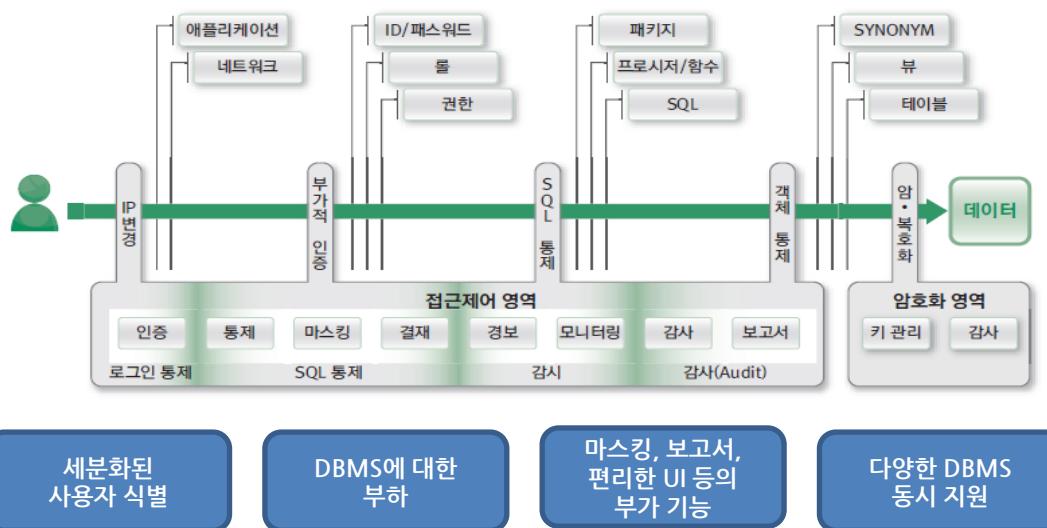
구성요소	설명
DB보안 기술요소	<ul style="list-style-type: none"> <li>• DB보안을 위한 최소한의 보안 통제 방법을 정의한 것</li> <li>• 각 요소들은 독자적으로 구축될 수도 있고, 둘 이상의 요소들이 복합적으로 구성될 수도 있음</li> <li>• 보다 견고한 DB보안을 위해서는 둘 이상의 요소들을 복합적으로 구성하고 체계적인 관리 노력을 기울여야 함</li> </ul>
DB보안 구축 프로세스	<ul style="list-style-type: none"> <li>• 각 기술요소들을 구축해 가는 절차적 단계를 구분한 것</li> <li>• 각 기술요소들을 구축하는데 있어서 단계적으로 수행해야 할 중요한 사항들의 선후관계를 식별하고 누락되지 않도록 체계적으로 수행해 나갈 수 있는 가이드라인의 의미를 지님</li> </ul>

# [보안] DB 접근제어

## 1. DB 접근제어의 개요

- 사용자가 DBMS에 로그인 하거나 SQL을 수행하려고 할 때 미리 정의된 보안규칙에 따라 권한 여부를 판단하여 통제하는 활동

## 2. DB 접근제어의 개념도



## 3. 접근제어 유형

접근제어 방법	특성
에이전트 방법	<ul style="list-style-type: none"><li>서버 자체에 접근제어 및 로깅 기능을 포함하는 에이전트 이식</li><li>DB에 직접 접근하는 전용 클라이언트를 포함해 모든 접근 루트를 제어 할 수 있는 가장 강력한 보안 방법</li><li>DB 서버에 트래픽이 발생, DB 서버의 성능 저하 우려, 시스템 정지의 리스크 내재</li></ul>
프록시 방법	<ul style="list-style-type: none"><li>DB 서버로 접속하는 모든 IP를 DB 보안서버(프록시서버)를 통하여 설정 변경</li><li>가장 강력한 접근제어 기능 제공</li><li>타깃 DBMS의 추가 가능해 대형 시스템 환경에 유리</li><li>보안서버(프록시서버)의 장애 발생 시에도 이중화 구성이 가능하므로 온라인 업무에 지장 없이 복구 가능</li></ul>
게이트웨이 방법	<ul style="list-style-type: none"><li>타깃 DB 서버와 클라이언트 네트워크 사이에 인라인 보안시스템 구성</li><li>서버나 클라이언트에 별도의 에이전트 설치나 설정 변경 불필요</li><li>규모가 크지 않고 DB 서버가 한 장소에 위치하고 온라인 업무의 비중이 그다지 높지 않은 시스템에 유리</li><li>보안서버 다운 시 모든 업무의 중단 우려가 있으며, 타깃 DB 서버의 규모에 따라 다수의 DB 보안서버 필요</li></ul>
인라인 방법	<ul style="list-style-type: none"><li>네트워크 선로 상의 패킷들을 TAP 방식과 패킷 미러링 방식을 통해 패킷을 분석 로깅하는 방법으로 사후 감사의 의미에 비중을 두는 보안 방식</li><li>서버와 클라이언트 사이에 어떠한 에이전트의 설치나 설정변경이 필요 없으며 네트워크에 부하 없이 시스템 구축 용이</li><li>데이터의 번조나 훼손으로 인한 무결성 유지 곤란</li></ul>
스니핑 방식	<ul style="list-style-type: none"><li>네트워크 선로 상의 패킷들을 TAP 방식과 패킷 미러링 방식을 통해 패킷을 분석 로깅하는 방법으로 사후 감사의 의미에 비중을 두는 보안 방식</li><li>서버와 클라이언트 사이에 어떠한 에이전트의 설치나 설정변경이 필요 없으며 네트워크에 부하 없이 시스템 구축 용이</li><li>데이터의 번조나 훼손으로 인한 무결성 유지 곤란</li></ul>

[출처] 2011년 DB백자[2011], 한국DB진흥원

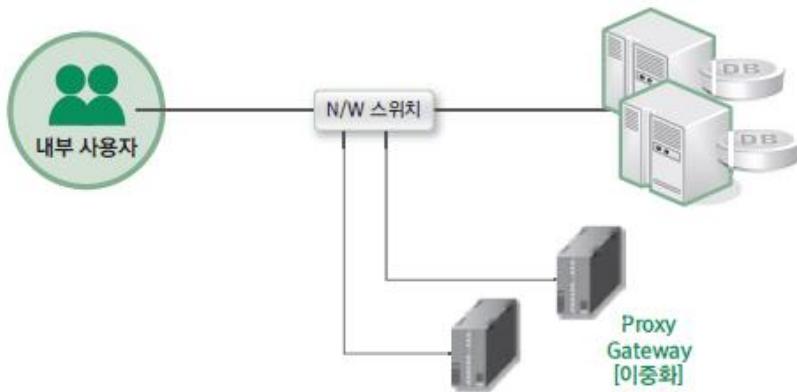
# [보안] DB 접근제어

## 4. DB 접근제어 유형 - Gateway 방식

- 내부 사용자의 개인정보 및 대외비 유출에 대한 통제에 용이
- 사용자가 보안서버를 통해 DB에 접근하도록 함
- 중앙 집중식 보안 기능

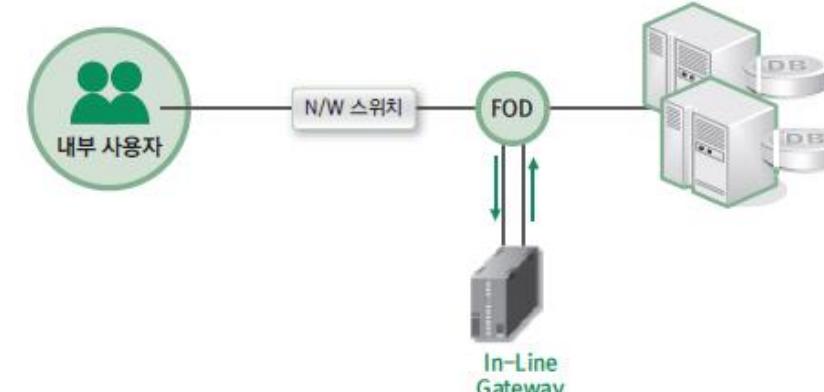
### Proxy Gateway 방식

- 사용자 단말기에 NAT Agent를 설치하여 Proxy Gateway 서버로 IP Forward
- 이중화 구성을 통하여 1개의 서버 장애 시에도 다른 서버를 통해서 보안기능 수행
- 보안 규칙에 대한 관리의 편의를 위해 보안 규칙이 복제(Replication) 되도록 함



### In-Line Gateway 방식

- FOD(Fail Over Device)를 이용하여 Packet 을 In-Line Gateway 서버로 전송하도록 설정
- In-Line Gateway 서버 장애 시에는 DB서버로 Packet 전송
- 사용자, DB 서버 환경 변화가 없는 투명한 설치가 가능함

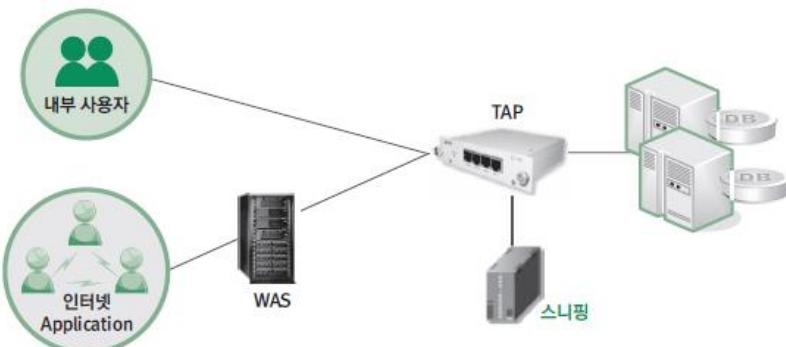


# [보안] DB 접근제어

## 4. DB 접근제어 유형

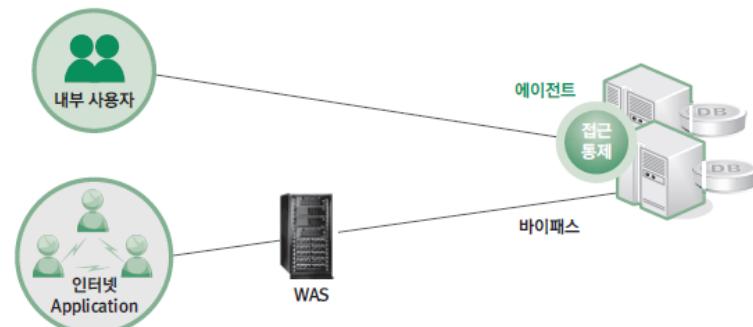
### Sniffing 방식

- 외부 사용자 및 내부 사용자가 DB로 보내는 모든 패킷을 로그서버에 저장하는 방식
- DB 서버에 미치는 영향이 없음
- 다수의 DBMS 관리에 용이
- TAP 을 이용하는 방식과, Switch Mirroring 기능을 이용하는 방식이 있음
- 고속의 Packet 처리 능력이 필요하며, 경우에 따라 Packet Loss 가 발생할 수 있음



### Agent 방식

- 우회 경로 완벽 차단
- Oracle의 BEO(Local Session) 완벽 처리
- telnet, SSH 모두 지원
- 감사 및 접근 제어 모두 제공**
- WAS 서버 등에서 오는 세션은 Bypass 설정을 하여, Agent 서버를 거치지 않고 직접 DB서버에 접속



# [보안] DB암호화

API, PlugIn,하이브리드

## 1. DB 암호화의 개요

- 데이터를 암호화하여 저장하고, **권한이 있는 사람 혹은 서버만이 해당 데이터를 복호화** 할 수 있도록 하여 데이터를 보호하는 기술

## 2. DB 암호화의 유형

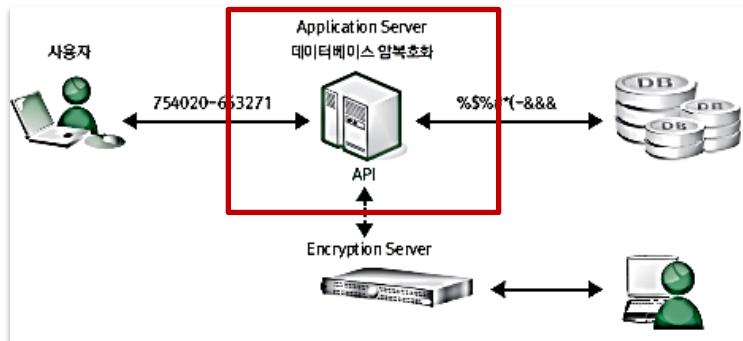
구분	설명
API 방식	<ul style="list-style-type: none"> <li>데이터 암/복호화를 수행하는 API를 웹애플리케이션 서버 단에 둠</li> <li>DB에 데이터를 저장하기 전에 웹 애플리케이션 서버 단에서 암호화 작업을 수행</li> <li>데이터 검색 시 DB에서 암호화된 데이터를 가져와 이를 복호화한 후 사용자에게 전송</li> </ul>
Filter(Plug-In)방식	<ul style="list-style-type: none"> <li>데이터 암/복호화를 수행하는 암호화 플러그인 모듈을 DB서버 단에 설치함으로써 암/복호화 작업이 DB서버 내에서 이루어지는 방식</li> </ul>
하이브리드방식	<ul style="list-style-type: none"> <li>API방식과 Filter방식을 결합하거나, Filter방식에 추가적으로 SQL문에 대한 최적화를 대행해 주는 어플라이언스 제공</li> </ul>
커널방식	<ul style="list-style-type: none"> <li>디스크에 있는 물리적 데이터파일과 작업 메모리 공간(SGA) 사이에 입출력을 담당하는 백그라운드 프로세스(서버, 프로세스, DBWR)들이 자동으로 암복호화를 담당</li> </ul>

## 3. DB 암호화 유형 비교

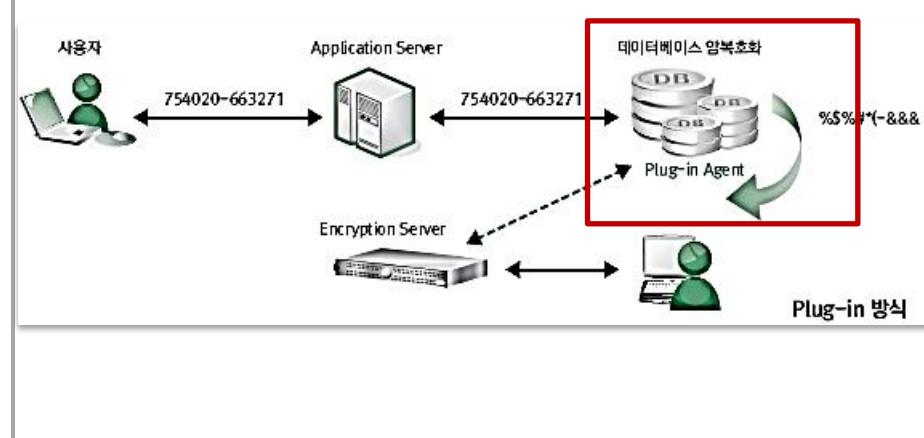
구분	API 방식	Filter(Plug-In) 방식	하이브리드방식
암호화/ 보안방식	별도의 API 개발 및 통합	DB내 설치, 연동	별도의 어플라이언스, DB내 에이전트
서버 성능 부하	<b>응용 서버에</b> 암/복호화, 정책 관리, 키관리의 부하 발생	<b>DB 서버에</b> 암/복호화, 정책 관리, 키관리의 부하 발생	<b>DB와 어플라이언스에서 부하 분산</b>
시스템 통합 용이성	Application 개발 통합 기간 필요	<b>Application 변경 불필요</b>	Application 변경 불필요
암호화 대상 변경 발생시	<b>Application 수정</b>	간단한 지정으로 가능	간단한 지정으로 가능
관리 편의성	App 변경 및 암호화 필드 변경에 따른 유지보수 필요	관리자용 GUI 이용, 다수 DB통합관리 가능. 편의성 높음	관리자용 GUI 이용, 다수 DB 통합 관리 가능. 편의성 높음

# [보안] DB암호화

API 방식



Plug-in 방식



어플라이언스



커널방식



# [보안] DB암호화

## 특징 및 장단점

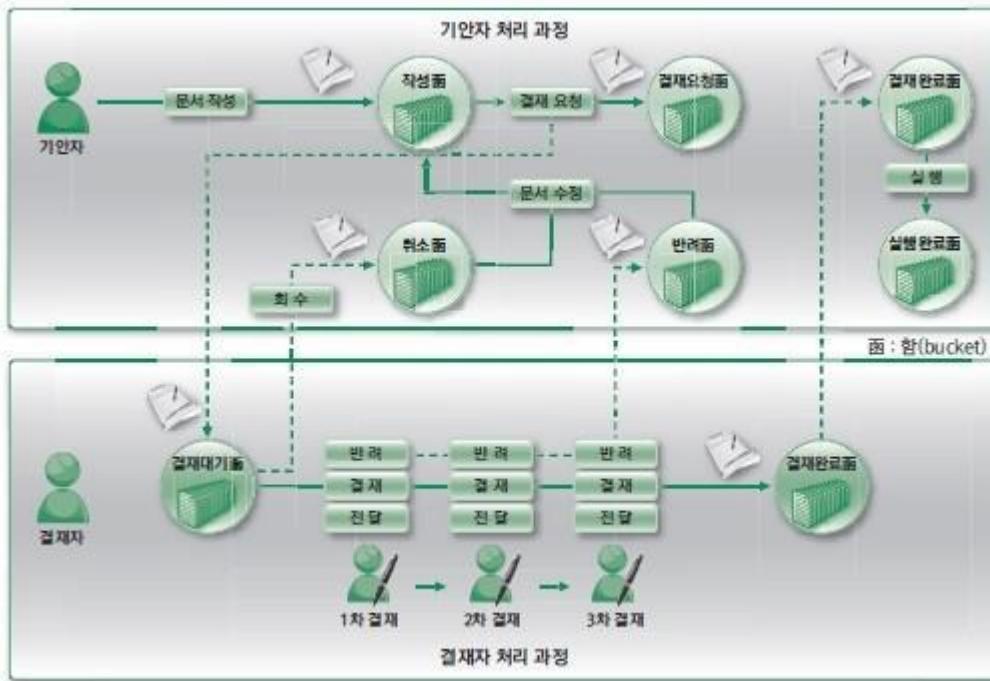
\* 출처 : 신시웨이

구분	암호화 기술	특징	장점	단점	보안인증
컬럼 단위 암호화	Plug-In 방식	<ul style="list-style-type: none"> <li>DBMS 서버에 암복호화 라이브러리를 설치 후 연동 (External Procedure Call / Java 함수 Call 방식)</li> <li>원래의 테이블과 동명의 뷰가 생성되고, 실제 테이블에 변경하기 위해 Instead of Trigger를 생성함</li> </ul>	<ul style="list-style-type: none"> <li>어플리케이션 수정이 적어 구축이 용이함</li> </ul>	<ul style="list-style-type: none"> <li>DBMS 서버에 부하가 있음</li> <li>SQL 수행 시 속도저하 발생하여 튜닝 필요</li> <li>일부 어플리케이션 변경 필요</li> <li>별도 인덱스(Index) 기능 필요</li> </ul>	국정원의 인증(CC, 암호모듈 검증제품)
	API방식	<ul style="list-style-type: none"> <li>AP 서버에 암복호화 라이브러리를 설치</li> <li>암호화 컬럼을 사용하는 모든 부분에 저장/변경 시 암호화 하고, 조회 시 복호화 하는 API 함수 호출하도록 PG변경</li> <li>DB서버에 별도로 설치되는 것이 없고, 뷰나 트리거 등이 생성되지 않음</li> </ul>	<ul style="list-style-type: none"> <li>DB 서버에 설치되는 것이 없어서 DB서버에 주는 부하가 없고, 속도가 빠름</li> </ul>	<ul style="list-style-type: none"> <li>API 적용을 위해 다수의 AP 프로그램 수정을 해야 하므로 장기간 소요</li> <li>프로그램 수정, 테스트를 위한 비용이 크게 발생</li> <li>별도의 인덱스(Index) 기능 필요</li> </ul>	
블록 단위 암호화	TDE 방식 (Transparent Data Encryption)	<ul style="list-style-type: none"> <li>DBMS 커널에서 암호화된 테이블스페이스를 생성하고, 암호화 대상 테이블을 해당 테이블스페이스로 이동</li> <li>DBMS 커널에서 DB의 블록 단위로 자동 암복호화 수행</li> </ul>	<ul style="list-style-type: none"> <li>암호화에 따른 어플리케이션 변경이 없고 성능이 우수함</li> </ul>	<ul style="list-style-type: none"> <li>DB에 직접 접속 시, 권한 없는 사용자도 암호화된 정보를 조회할 수 있어서 보안성이 떨어짐</li> <li>Oracle 11g로 업그레이드가 필요 하므로 부가적인 비용이 수반됨</li> </ul>	국정원 인증 제품이 없음
	File 암호화 방식	<ul style="list-style-type: none"> <li>암호화 파일을 사용하여 테이블스페이스를 생성하고, 암호화 대상 테이블을 해당 테이블스페이스로 이동</li> <li>OS 커널에서 DB의 블록 단위로 자동으로 암복호화 수행</li> </ul>	<ul style="list-style-type: none"> <li>모든 데이터 타입 지원</li> <li>DBMS 자체 인덱스를 모두 사용 가능</li> </ul>	<ul style="list-style-type: none"> <li>DB에 직접 접속 시, 권한 없는 사용자도 암호화된 정보를 조회할 수 있어서 보안성이 떨어짐</li> <li>일부 OS, Volume Manager의 경우 지원 불가</li> </ul>	

# [보안] 작업결재

## 1. SQL 수행과정에서 관리자의 승인을 획득하도록 하는 관리적 보안, DB 보안의 작업 결제의 개요

- DB사용자가 고의적은 Data 유출을 목적으로 DB에 접근하거나 또는, DB작업 시 실수로 인해 문제가 발생하는 것을 사전에 차단하기 위하여, **사전에 정의된 DB보안정책에 따른 내부결재를 거친 SQL만 실행하도록 하는 DB보안의 한 형태**



종류	설명
사전 결제	<ul style="list-style-type: none"><li>수행하고자 하는 SQL 수행 이전에 결제 받는 방식</li><li>SQL, DB 계정, 테이블, 컬럼 등의 다양한 단위로 결재 가능</li><li>작업 시간 지정도 가능</li></ul>
즉시 결제	<ul style="list-style-type: none"><li>SQL 수행 도중에 결재 받는 방식</li><li>관리자가 승인을 할 수 있는 상태에 있어야 함</li></ul>
사후 결제	<ul style="list-style-type: none"><li>SQL 수행 후 결재하는 방식</li></ul>

# B Tree

## 1. Balanced Tree를 통한 균등한 응답속도 보장을 위한 탐색 트리, B-Tree 개요

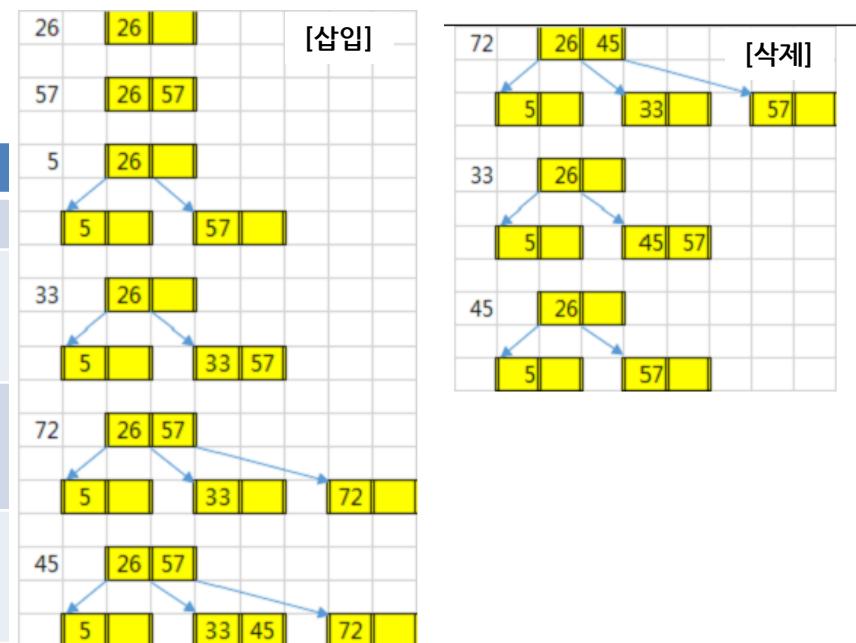
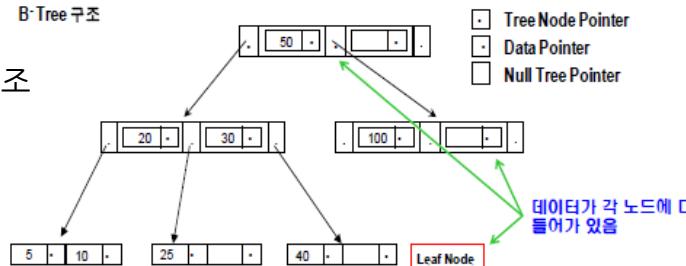
- 데이터를 정렬하여 탐색, 삽입, 삭제 및 순차 접근이 가능하도록 유지하는 트리형 자료구조

## 2. B tree의 특징

- 각 노드는 1/2 이상 채워져야 하며**, 모든 leaf node는 같은 level에 있음
- 탐색, 추가, 삭제는 **root node로부터 시작함 / 노드 내 값은 오름차순 유지**
- 공백이거나 높이가 1이상인 m-원 탐색 트리(m-way search tree)
- 루트와 리프(leaf)를 제외한 내부 노드는 최소[m/2], 최대 m 개의 서브트리를 가지며, 적어도 [m/2]-1개의 키 값을 가짐 -> 적어도 반 이상이 키 값으로 채워져 있어야 한다.
- 루트는 그 자체가 리프가 아닌 이상 적어도 두 개의 서브트리를 가짐**
- 모든 리프는 같은 레벨에 있음(**균형트리**)

## 3. B트리 삽입 알고리즘

구분	Case	설명
삽입	여유공간 있을 경우	<ul style="list-style-type: none"> <li>삽입할 key 값은 항상 Leaf 노드에 삽입</li> </ul>
	여유공간 없을 경우	<ul style="list-style-type: none"> <li>해당 Leaf 노드에 새로운 key 값을 삽입 했을 경우 가정</li> <li>중간 key 값을 중심으로 2개 노드로 분할(Split)</li> <li>중간 key 값을 분할된 노드에 대한 부모 노드에 삽입</li> <li>이 때, 부모 노드에서 Overflow 발생할 경우 분할 작업 반복</li> </ul>
삭제	삭제 key 값이 Leaf 노드인 경우	<ul style="list-style-type: none"> <li>삭제는 항상 Leaf 노드에서 이루어짐</li> <li>최소 key 개수가 <math>m / 2 - 1</math> 보다 작은 경우 Underflow 발생</li> <li>Underflow 발생 시, 재분배(Redistribution) 또는 합병(Merge) 수행</li> </ul>
	삭제 key 값이 Leaf 노드 아닌 경우	<ul style="list-style-type: none"> <li>후행 key 값과 자리를 바꾸어 Leaf 노드로 이동시킨 후 삭제</li> <li>최소 key 개수가 <math>m / 2 - 1</math> 보다 작은 경우 Underflow 발생</li> <li>Underflow 발생 시, 재분배(Redistribution) 또는 합병(Merge) 수행</li> </ul>

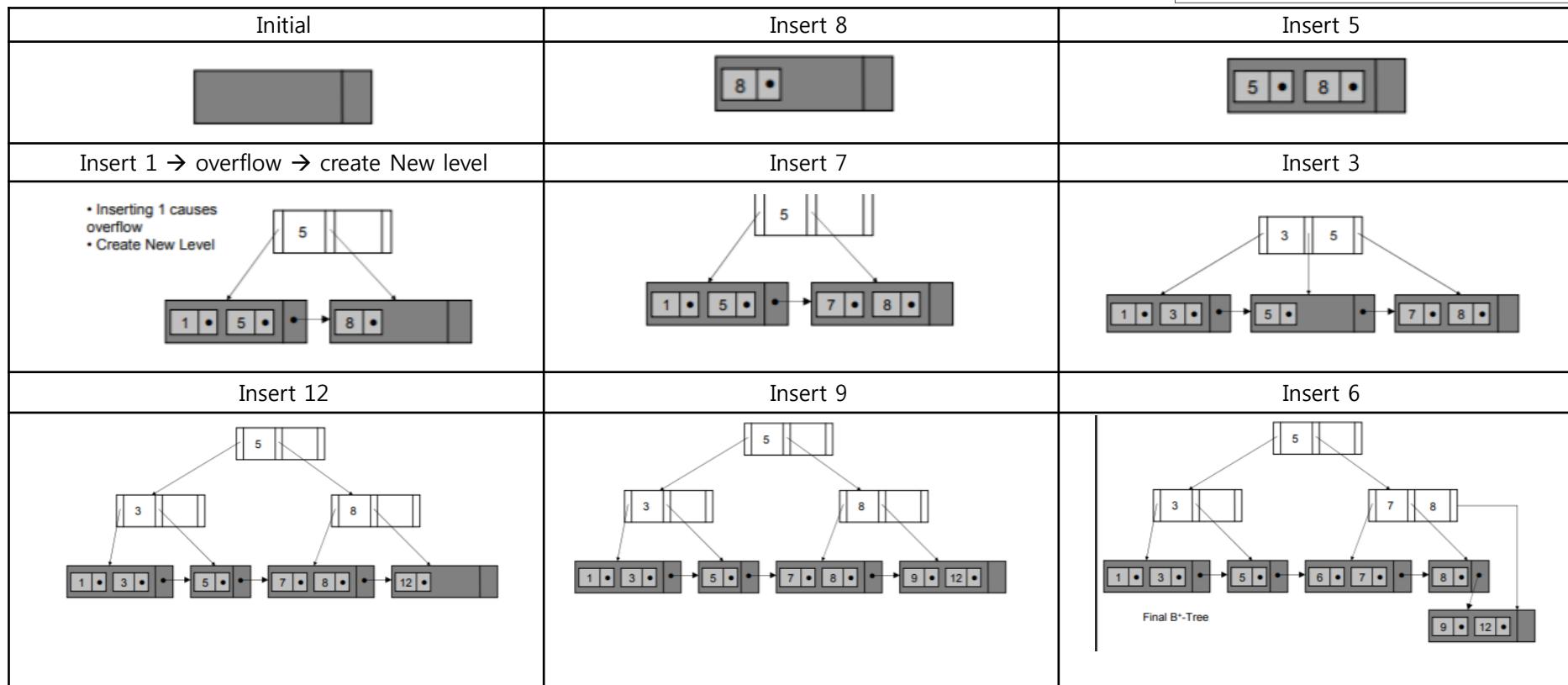
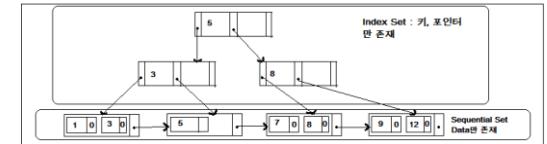


# B+ Tree

## 1. B+ Tree의 개요

- 키에 의해서 각각 식별되는 레코드의 효율적인 삽입, 검색과 삭제를 통해 정렬된 데이터를 표현하기 위한 트리자료구조
- Index Set(키, 포인터만 존재), Sequence Set (Data만 존재)  
→ 모든 레코드들이 트리의 가장 하위 레벨에 정렬되고, 오직 키들만이 내부 블록에 저장

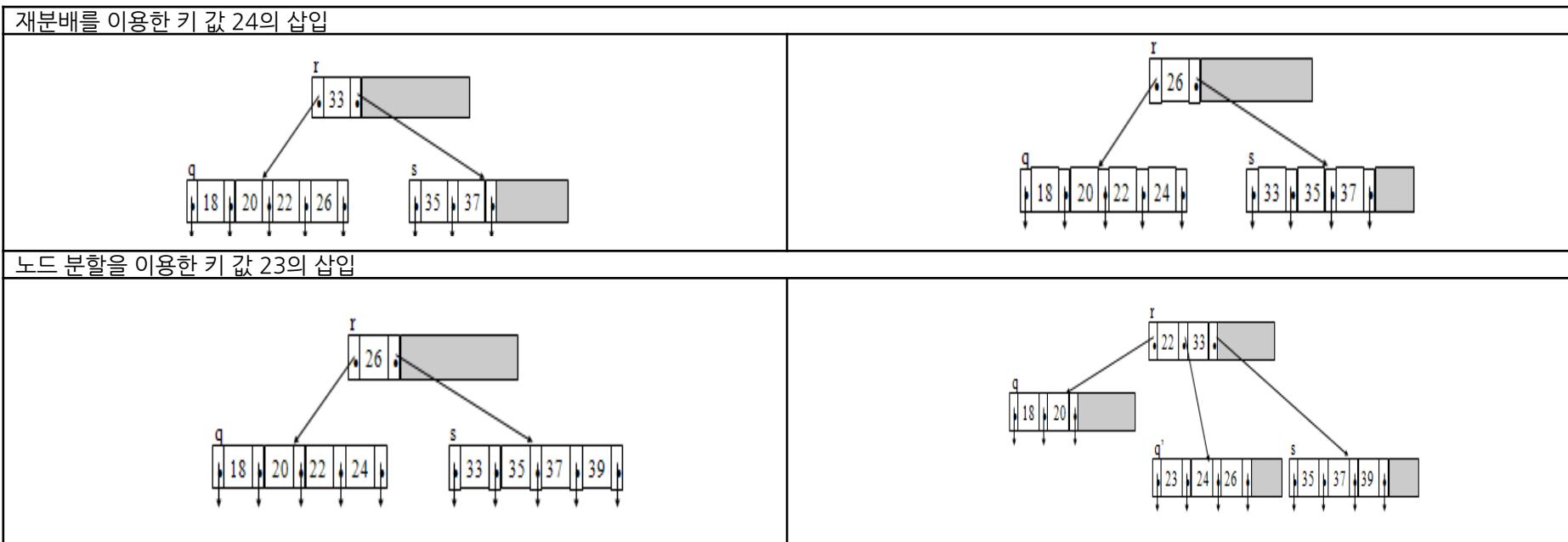
## 2. B+ Tree의 데이터 삽입 과정



# B\* Tree

## 1. B\* Tree의 개요

- B+트리와 구조는 같으나 1/2 분열 문제를 보완하기 위해 B+-tree를 약간 변형한 트리  
(공간활용도를 높이고 보조연산 횟수를 감소시킴, B+ Tree의 Fill Factor 조정 모델)
- Root Node를 제외하고는 2/3이상 노드가 채워져야 함



# AVL Tree

LL, LR, RL, RR

## 1. AVL(Adelson-Velskii, Landis)Tree 개념

- 각 노드에서 왼쪽 서브 트리 높이와 오른쪽 **서브 트리 높이 차이(Balance Factor)** 가 1이하인 이진탐색 트리
- 균형인자(Balance Factor) = 왼쪽트리 높이 - 오른쪽 트리 높이 = 모든 노드 균형인수가 ±1 이하인 AVL 트리

## 2. 균형 트리로 만드는 방법

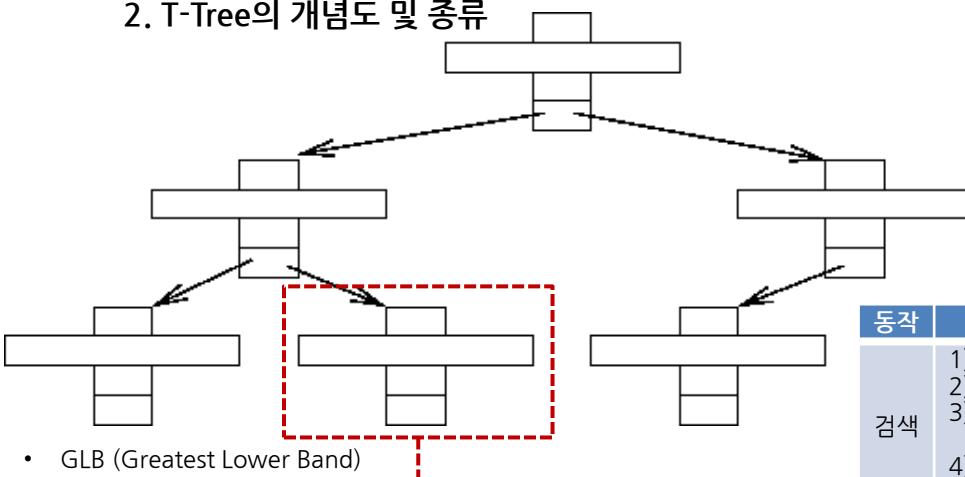
	LL 타입	LR 타입	RR 타입	RL 타입
LL 회전	가운데 노드를 Pivot 설정, <b>오른쪽 회전</b>			
LR 회전	마지막노드 Pivot, <b>왼쪽-오른쪽 회전</b>			
RR 회전	가운데 노드를 Pivot 설정, <b>왼쪽 회전</b>			
RL 회전	마지막노드를 Pivot, <b>오른쪽-왼쪽 회전</b>			

# T Tree

## 1. AVL-Tree와 B-Tree의 개선된 형태, T-tree의 개요

- AVL-Tree의 이진 탐색 특성 및 높이 균형과 B-Tree의 업데이트와 저장 효율(한 노드는 여러 개 데이터 가짐)을 물려 받은 메모리 기반 DBMS에서 선호되는 인덱스 자료구조 (목적 : Memory 구조의 최적화)
- AVL-Tree가 하나의 노드에 데이터 한 개만을 가지는 대신 T-Tree는 **하나의 노드가 n개의 데이터를 가질 수 있도록 개선한 구조임**

## 2. T-Tree의 개념도 및 종류



종류	설명
내부노드 (Internal Node)	• 오른쪽, 왼쪽 서브 트리를 가짐. 내부 노드에 포함될 수 있는 데이터의 개수는 T트리 생성할 때 결정
하프 리프노드 (Half-leaf Node)	• 방향과 상관 없이 한쪽 서브 트리만을 갖고 하나의 자식 포인터만 가짐
리프노드 (Leaf Node)	• 자식포인터가 하나도 없음

동작	설명
검색	<ol style="list-style-type: none"> <li>이진트리에서의 검색과 유사</li> <li>검색은 항상 트리의 루트부터 시작</li> <li>만약 검색하려는 값이 그 노드의 가장 작은 값보다 작으면 왼쪽 서브트리로 이동하여 해당 노드들을 계속해서 검색</li> <li>그렇지 않고 만약 검색 값이 그 노드의 가장 큰 값보다 크면 오른쪽 서브트리로 이동하여 계속 검색</li> <li>그렇지 않으면 현재 노드에서 검색</li> </ol>

동작	설명
삽입	<ol style="list-style-type: none"> <li>삽입할 위치를 검색</li> <li>만약 삽입할 노드가 발견되면, 해당 노드에서 삽입할 수 있는 여분의 방이 있는지를 검사하여 적당한 여분의 방이 있으면 삽입하고 종료.</li> <li>그렇지 않으면, 가장 작은 값을 제거하고 빈 공간을 만든 후 삽입하려는 값을 삽입하고, 삭제되었던 최소 값은 왼쪽 서브 트리에 삽입</li> <li>만약 왼쪽 서브트리가 없다면 신규 트리 생성 후 삽입</li> <li>서브 트리에도 빈 공간이 없다면 위의 삽입 과정을 반복</li> <li>만약 새로운 단말 노드가 첨가되면 단말 노드부터 루트까지의 경로에 대한 균형을 검사(LL, LR, RL, RR)</li> </ol>

동작	설명
삭제	<ol style="list-style-type: none"> <li>삭제할 값을 가지고 있는 노드 검색</li> <li>만약 삭제 연산이 언더플로우(삭제 후 노드의 값이 NULL만 있음)를 야기 시키지 않는다면 단순히 그 값을 삭제하고 종료</li> <li>그렇지 않고 만약 중간 노드(internal node)라면, 그 값을 삭제하고 왼쪽 서브 트리로부터 가장 큰 값을 가지고 옴</li> <li>만약 단말 노드(자식노드가 0) 또는 한쪽 단말 노드(자식노드가 1)가 있으면 아이템을 삭제한다.</li> <li>만약 노드가 한쪽 단말 노드이고 한 단말 노드와 병합시킬 수 있다면, 두 개의 노드를 한 개의 노드로 병합하고 다른 한 노드는 삭제</li> <li>그렇지 않다면 트리에 대한 재 균형 작업(LL, LR, RL, RR)</li> </ol>



# R Tree

공간

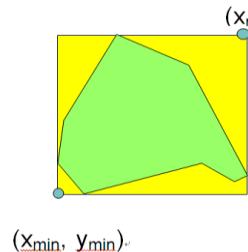
MPR

교차, 포함, 근접이웃

## 1. N차원 사각형 기반의 구조화된 자료 구조, R-Tree

- N차원의 공간 데이터를 효과적으로 저장하고 지리정보와 관련된 질의를 빠르게 수행 할 수 있는 자료구조

## 2. R-Tree의 구조(MBR: Minimum Bounding Rectangle)



- R-Tree 저장단위 : 공간을 최소경계사각형(MBR: Minimum Bounding Rectangle)들로 분할하여 저장
- MBB(Minimum Bounding Box) 라고도 함
- 내부의 다각형이 실제 데이터

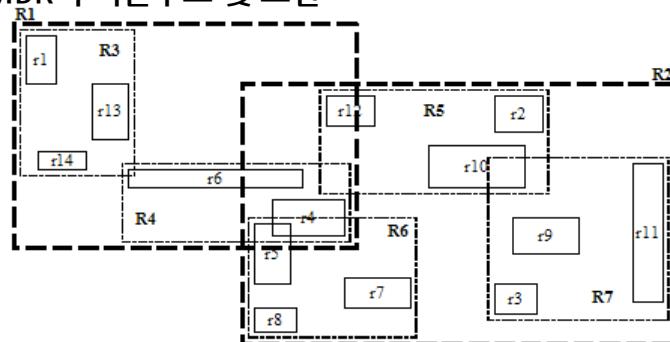
### [MBR의 종류]

- 객체 MBR ; 객체 자체를 나타냄
- 리프노드 MBR ; 몇 개의 객체 MBR의 그룹을 표현하는 MBR
- 중간 MBR ; 몇 개의 리프노드 그룹을 표현하는 MBR

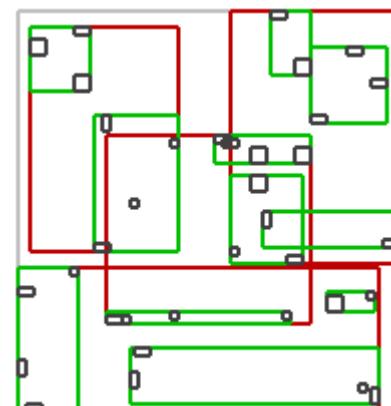
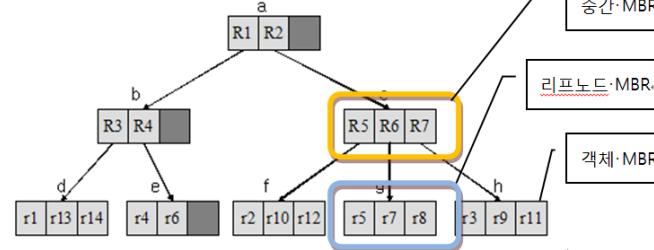
### [활용분야]

- GPS 차량/항법
- 이동객체 색인,
- 과거위치/궤적(Spatio-Temporal/Trajectory Bundle/Combined)
- 현재/미래(Time Parameterized R-Tree)

## 3. MBR의 기본구조 및 표현



### [R-Tree의 표현]



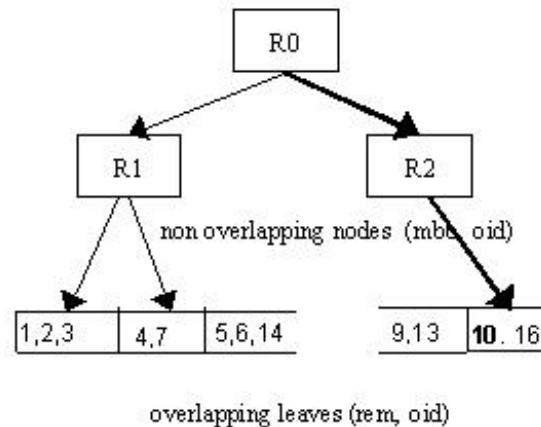
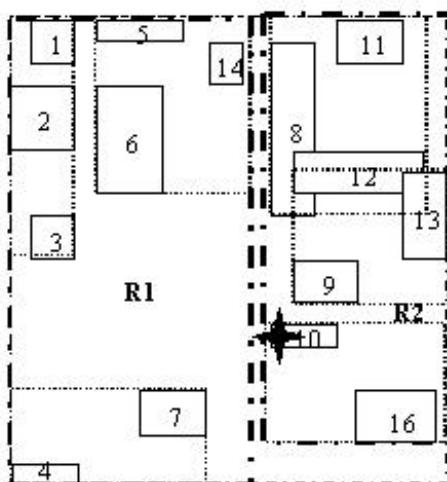
## 4. R-Tree의 탐색 알고리즘

탐색 알고리즘	설명
교차 질의(Intersection)	모든 노드들은 질의입력 MBR과 자식 노드들의 MBR을 비교하여 <u>교차되는 영역이 있는 자식 노드들에게만 질의를 전달</u>
포함 질의(Containment)	<u>교차되는 영역이 있는 자식 노드가 아닌</u> , 질의를 포함하는 자식 노드들만 검색
근접 이웃 질의(Nearest Neighbor)	점좌표와 거리를 입력 받아 특정 점좌표로부터 <u>가장 가까운 거리에 위치한 노드를 검색</u>

# R+ Tree

## 1. 겹침 관계를 제거한 R 트리 R+Tree의 개요

- R-Tree와 K-dimensional Tree의 중간형태로, 기본적인 구조와 연산은 R-Tree와 거의 동일하며, R+Tree는 내부 노드들의 중첩을 최소화하는 방법을 제시하여 삽입이나 삭제시 연산성능이 더 우수함
- 검색도중 다중 경로를 피함(Avoids multiple paths during searching)



saglio@enst.fr

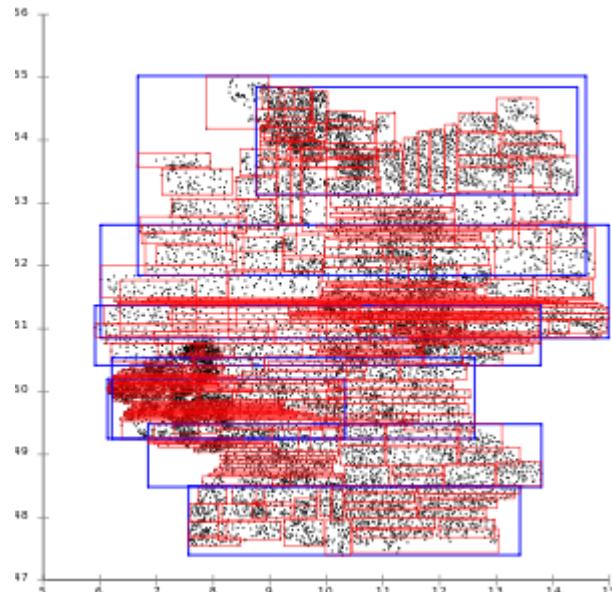
# R\* Tree

## 1. R-Tree 삽입/삭제 개선으로 성능 향상한, R\* Tree 개요

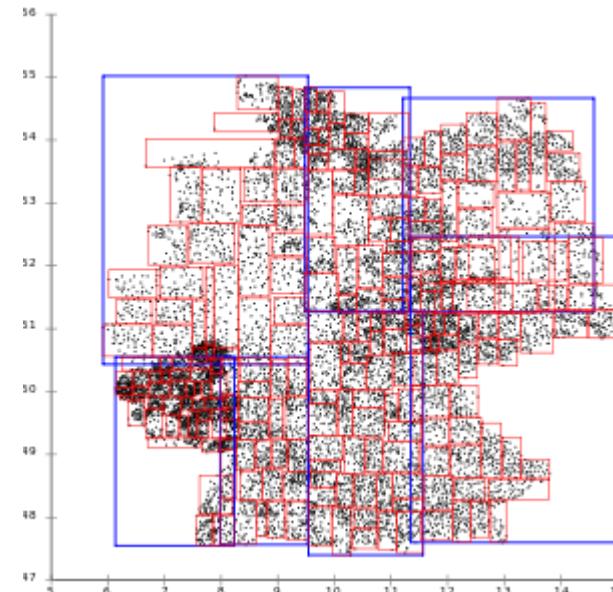
- R-Tree의 변형으로 기본적인 구조, 연산은 거의 동일, 삽입, 삭제 연산시 부모노드의 사각형을 효율적으로 확장 가능한 Tree
- 성능 개선 고려 요소 : 겹침감소, 면적감소, 디렉토리 사각형의 둘레길이 감소
- 검색속도는 개선되었지만, 업데이트 속도는 느려졌음

- R Tree : 겹침으로 인한 탐색 성능저하, 임의의 삽입순서로 인한 성능저하
- R+ Tree : 겹침이 없음, 공간사용 증가
- R\*-Tree : 새로운 분할방법은 겹침을 줄임, 강제 재삽입을 통해 노드 클러스터링 효과 증대

[R-Tree]



[R\*-Tree]





# 출처

1. 아이리포 카페 ([café.naver.com/itlf](http://cafe.naver.com/itlf))
2. 데이터베이스 시스템 (이석호)
3. 개념을 콕콕 잡아주는 데이터베이스
4. 데이터베이스 개론(한빛미디어)
5. <http://www.dbguide.net>
6. <http://www.dator.co.kr>



# ALL PASS

# THANK YOU

<http://cafe.naver.com/itlf/>  
<http://www.ilifo.co.kr/>

비매품

