



RDBMS - Modeling

목 차

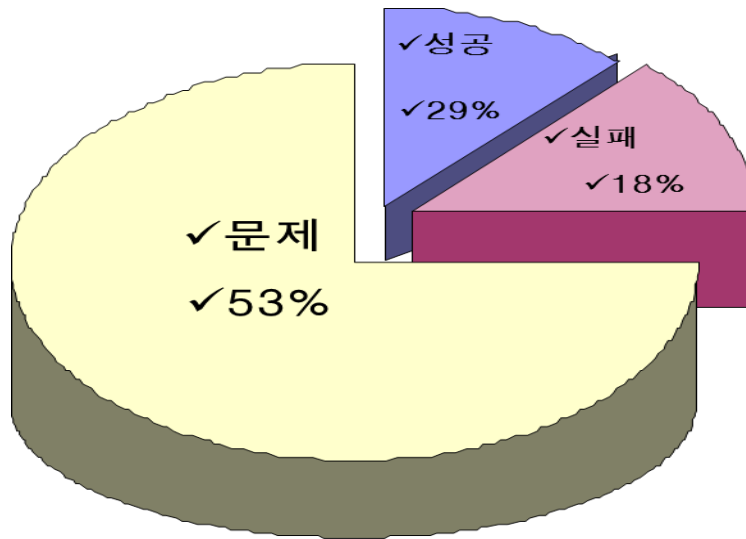
1. 요구사항 분석
2. 개념적 데이터 모델링
3. 상세 개념적 데이터 모델링
4. 논리적 데이터 모델링
5. 물리적 설계

1. 요구사항 분석

- 요구사항 분석
- 데이터의 표준화

요구사항 분석 - 정보 요구사항의 중요성

- ▶ 잘못된 정보 요구사항으로 분석, 설계된 시스템을 개발한다면 사용자의 요구 사항을 만족하지 못하게 될 것이며 추가적인 비용과 시간을 재투자해야 하는 문제가 발생하게 됨

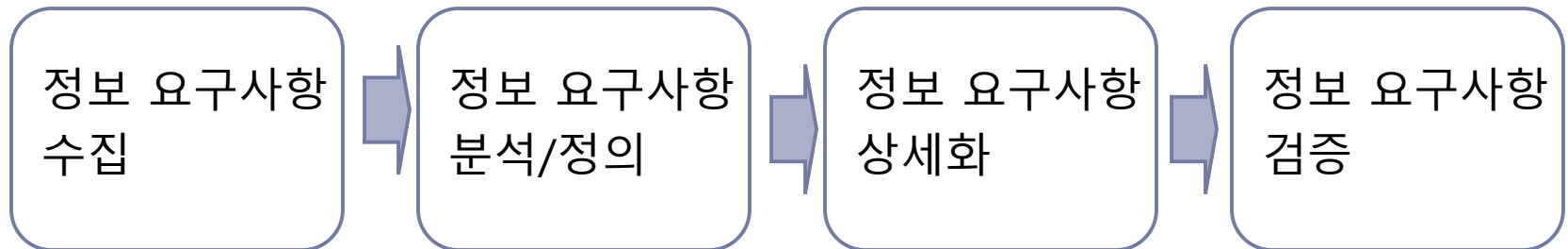


< 프로젝트 결과 >

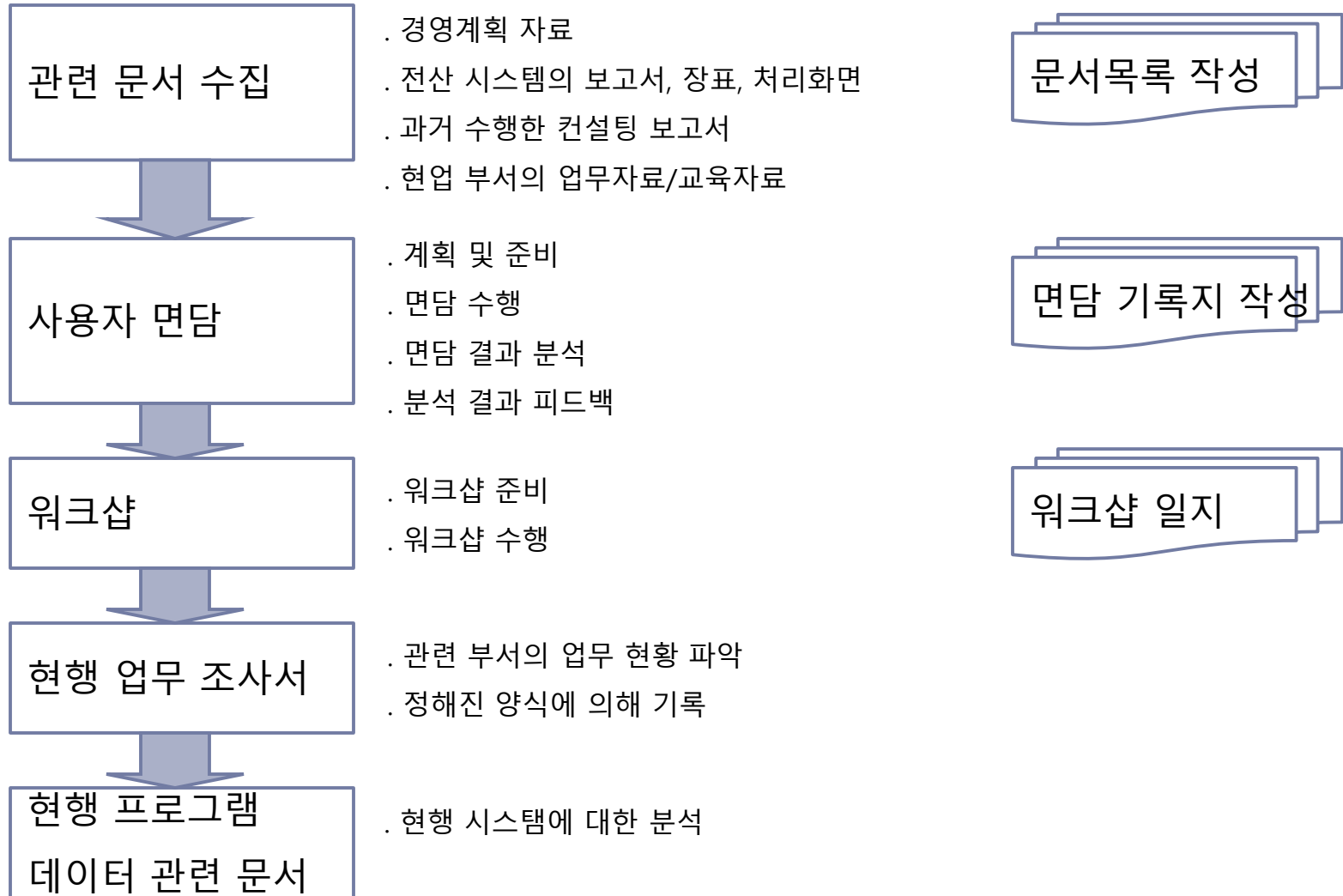
* 2004년 스탠디시 그룹 통계 참조 *

요구사항 분석 - 정보 요구사항 관리

- ▶ 기업 내에 흩어져 있는 많은 정보들을 사용자 관점에서 어떻게 수집해서 이를 체계적으로 분류하고 사용자가 필요로 하는 정보 요구사항을 도출하며 도출된 정보 요구사항이 누락되거나 추가 보완할 사항이 있는지 상관분석을 통해 검증하는 작업을 의미

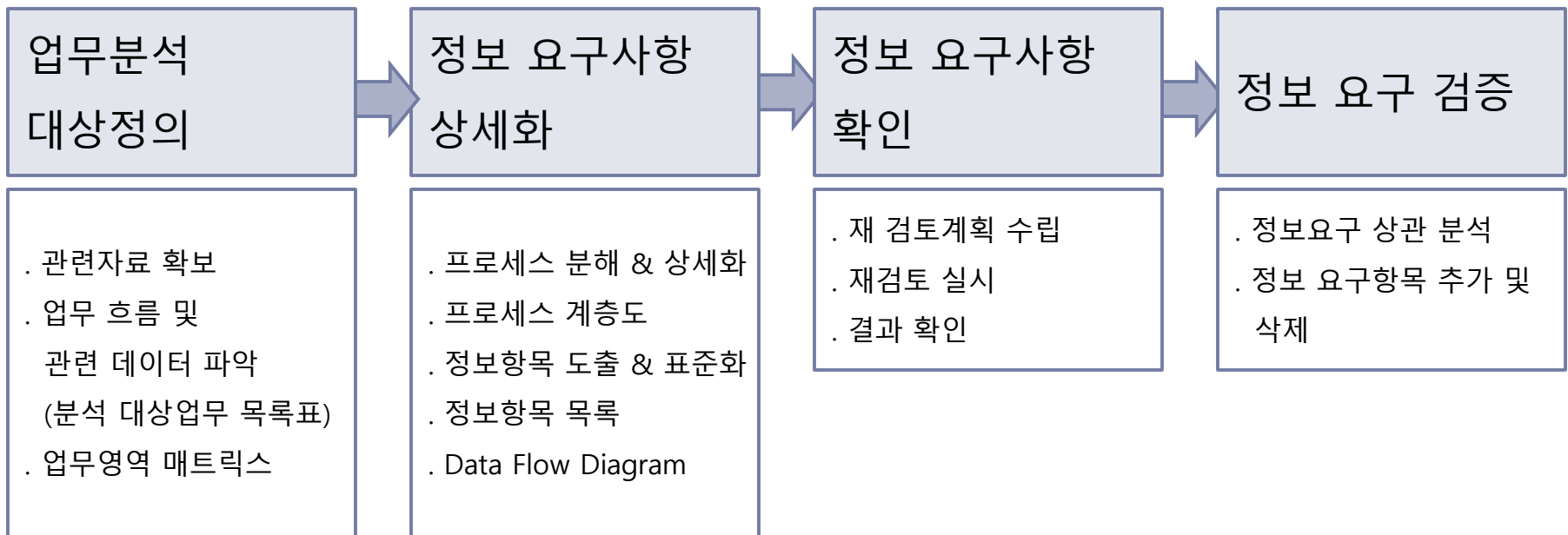


요구사항 분석 - 정보 요구사항 수집 절차



요구사항 분석 - 정보 요구사항 분석

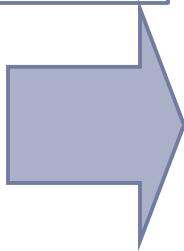
- ▶ 사용자로부터 수집한 정보 요구사항을 바탕으로 **업무 현황을 분석**하고 이를 근간으로 관련 업무 및 시스템의 문서를 조사, 수집 및 파악함으로써 **현행 업무 및 현행 시스템에 대한 분석 대상을 정의**



요구사항 분석 - 정보 요구사항 상세화

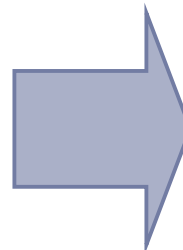
프로세스 분해/상세화

- . 프로세스의 분해 및 도출
- . 프로세스 계층도 작성
- . 프로세스 정의서 작성



정보항목 추출 및 표준화

- . 정보 항목의 추출
- . 정보 항목에 대한 표준화
- . 정보 항목 목록 정의



정보 항목 통합성 여부 결정

- . 정보 항목의 분류
- . 동음이의어, 이음동의어
- . 최종 정보항목 정의

요구사항 분석 - 정보항목의 도출

- ▶ 1. 프로세스 분해 및 상세화에서 추출된 기본 프로세스별로 등록, 조회, 변경, 삭제 기능을 구분하여 기술
- ▶ 2. 기능별로 구분된 프로세스별로 정보요구사항 정의서 및 업무조사서 상의 내용을 파악하여 관리하고자 하는 정보 항목을 추출함
- ▶ 3. 이 때 명사형으로 표현된 단어들이 정보 항목의 대상이 될 수 있음.
- ▶ 4. 추출된 정보 항목은 명명규칙을 준수해야 하며 업무용어를 그대로 사용하는 것이 좋음
- ▶ 5. 추출된 정보 항목을 그룹핑하여 정보 항목 군을 구분하고 목록을 작성

요구사항 분석 - 정보항목의 도출

DATA 항목 일람표

입출력명	사원정보 현황	작성일자	1992-08-31	작성자	주종면
------	---------	------	------------	-----	-----

번호	Data 항목	항 목 설 명	Data유형	자리 수	기타설명
1	사원번호	사원 구분을 위한 번호	숫자	7	
2	사원명	Last Name	문자	50	
3	성	First Name	문자	50	
4	사용자 계정	시스템 접속 시 계정	문자	8	8문자 범위
5	입사일자	사원의 입사일자	날자		YYYY/MM/DD
6	상관번호	팀장의 사원번호	숫자	7	사원 중 한명
7	직무	해당 직무	문자	25	
8	부서명	근무 부서명	문자	25	
9	급여	급여액	숫자	9(9).99	
10	커미션	제품 판매의 커미션	숫자	9(2).2	10 ~ 20%

데이터의 표준화 - 필요성 및 기대효과

▶ 데이터 표준화의 필요성

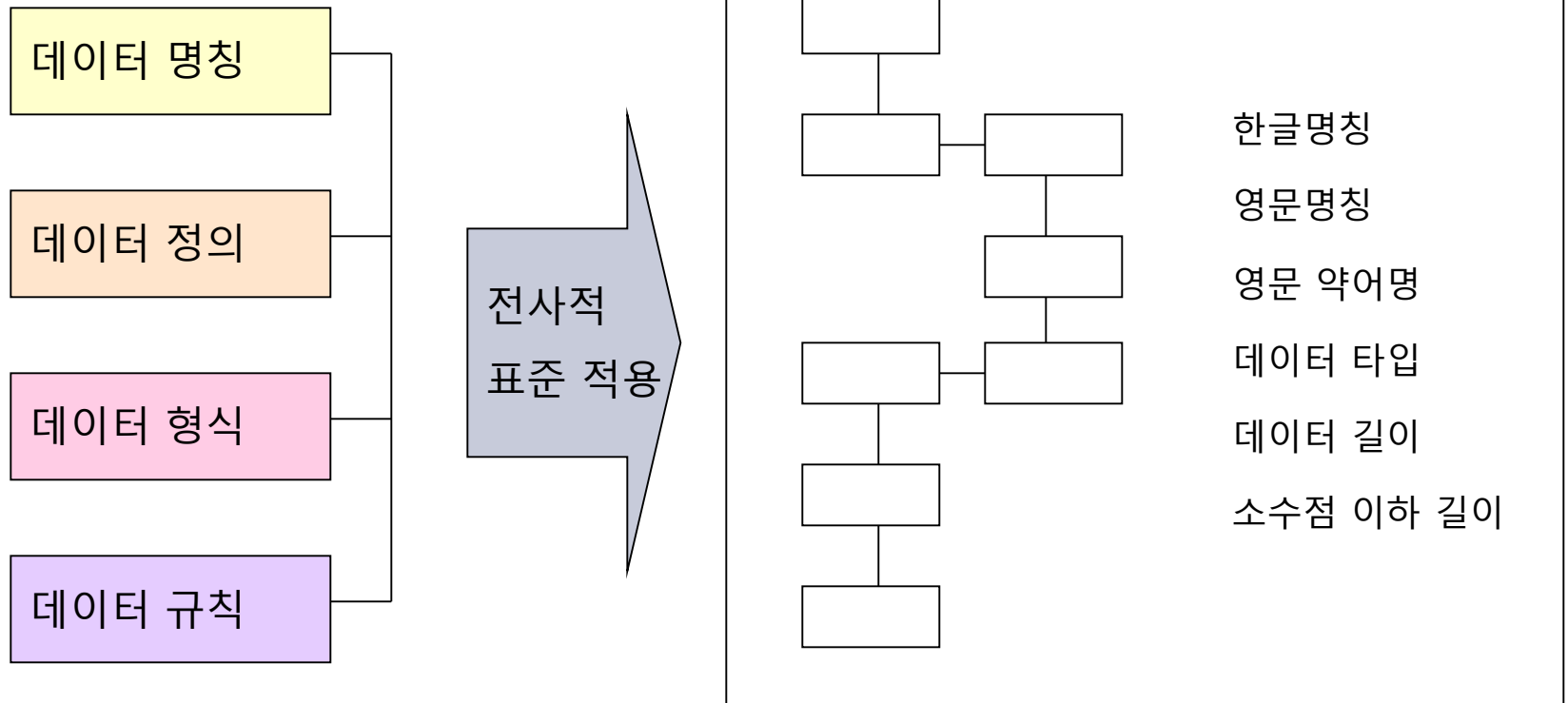
- 1) 데이터의 중복 및 데이터의 불일치 발생
- 2) 데이터에 대한 의미 파악 지연으로 정보 제공의 적시성 결여
- 3) 데이터 통합의 어려움
- 4) 정보 시스템의 변경 및 유지보수의 곤란

▶ 기대효과

- 1) 표준화로 인한 명확한 의사소통 가능
- 2) 데이터 소재파악에 소요되는 시간절약
- 3) 데이터 품질의 향상
- 4) 데이터 변환, 정제 비용의 감소

데이터의 표준화 - 개념

- ▶ 시스템별로 산재되어 있는 데이터 정보 요소에 대한 명칭, 정의, 형식, 규칙에 대한 원칙을 정의하여 이를 전사적으로 적용하는 것을 의미



데이터의 표준화 - 기본 원칙

표준화 원칙	예시
당사에서 사용하고 있는 관용화된 용어를 우선 적용	사원(종업원)
영문명 결정시 발음식은 최대한 지양하며 정상적인 영어 단어로 표현해야 함	사원 : SAWON (X) 사원 : EMPLOYEE (O)
한글명, 영문명 부여 시 특수문자 사용과 띄어쓰기는 금지	
기관명은 해당 기관에서 사용하고 있는 약어를 따름	정통부(정보통신부)
한글명에 대한 영문명은 복수 개를 둘 수 있음	
영문명에 대한 한글명은 복수 개를 허용하지 않음	

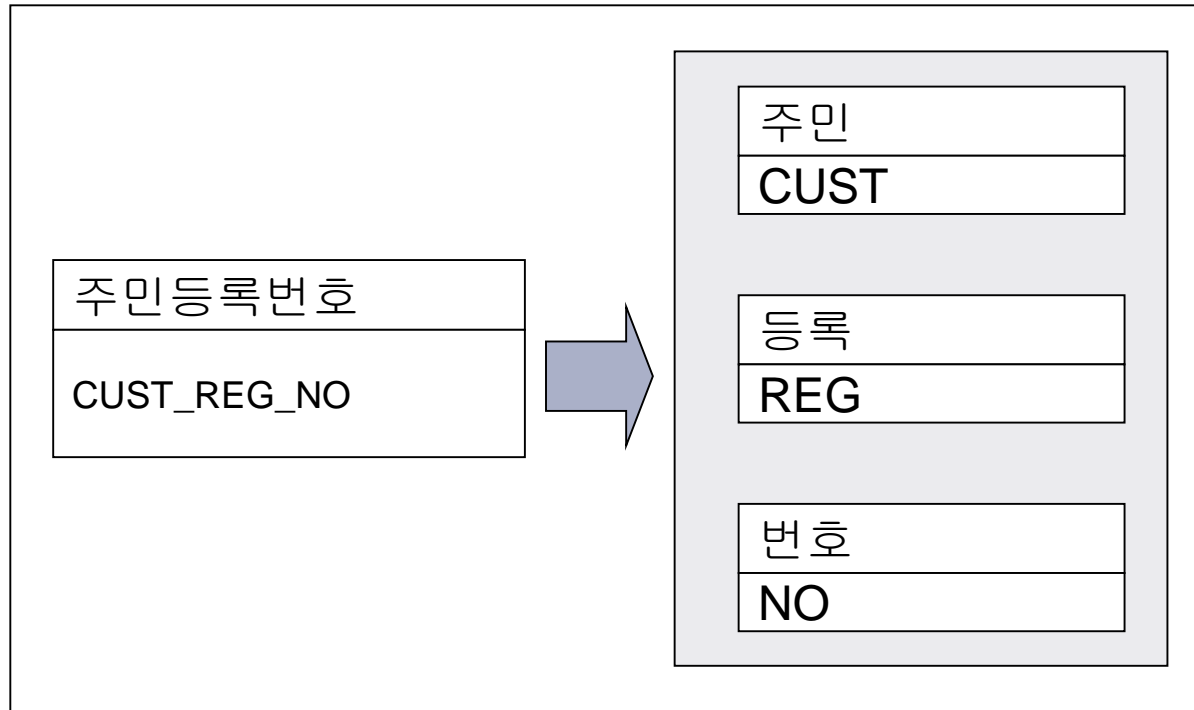
데이터의 표준화 - 데이터 표준 지침 작성

기술 내용	설 명
사용 문자	알파벳, 한글, 숫자 만 허용
영문 대소문자	표준화된 모든 용어는 대문자를 사용 (테이블명과 컬럼명은 대문자로 작성하되 SQL에서는 소문자)
한글명과 영문명 사용규칙	테이블명과 컬럼명은 영문명으로 통일 데이터 모델의 엔티티와 속성은 한글로 작성
명칭의 구조	테이블명은 [업무영역_주제어_수식어] 형태로 지정 (예) 영업관리 시스템의 직원정보 테이블 : s_emp_info 주문상세 테이블 : s_order_detal
허용 길이	표준 용어에 맞는 적절한 데이터 길이를 결정함(12문자 이내)
명칭 표준화에 대한 기준	유사한 단어가 복수 개 존재할 때 결정하는 기준 (예) 일련번호, 순번 : SEQ → NO, 번호 : ID

데이터의 표준화 - 표준 단어사전 정의(1/3)

▶ 단어의 분할

- 업무상 사용되는 단어를 최소 단위로 분할하는 것이 원칙
- 단, 영문명으로 분할될 수 있는지를 고려해 볼 것



데이터의 표준화 - 표준 단어사전 정의(2/3)

▶ 단어의 정련

- 이음동의어 : 한글명은 다르지만 같은 의미로 사용되는 단어

AS-IS 단어	약 어	사용여부
비밀번호	PSWD	Y
암호	PSWD	N



AS-IS 단어	약 어	사용여부
비밀번호	PSWD	Y

- 동음이의어 : 한글명은 같지만 다른 의미로 사용되는 단어

AS-IS 단어	약 어	사용여부
(고객)주소	ADD	Y
(창고)주소	ADD	Y
(사원)주소	ADD	Y



AS-IS 단어	약 어	사용여부
(고객)주소	CUST_ADD	Y
(창고)주소	WARE_ADD	Y
(사원)주소	EMP_ADD	Y

데이터의 표준화 - 표준 단어사전 정의(3/3)

< 표준 단어 사전 >

번 호	한글명	설 명	영문명	영문 약어명	단어 유형
1	고객	고객이름	CLNT	CT	
2	주소	고객 주소	ADR	AD	

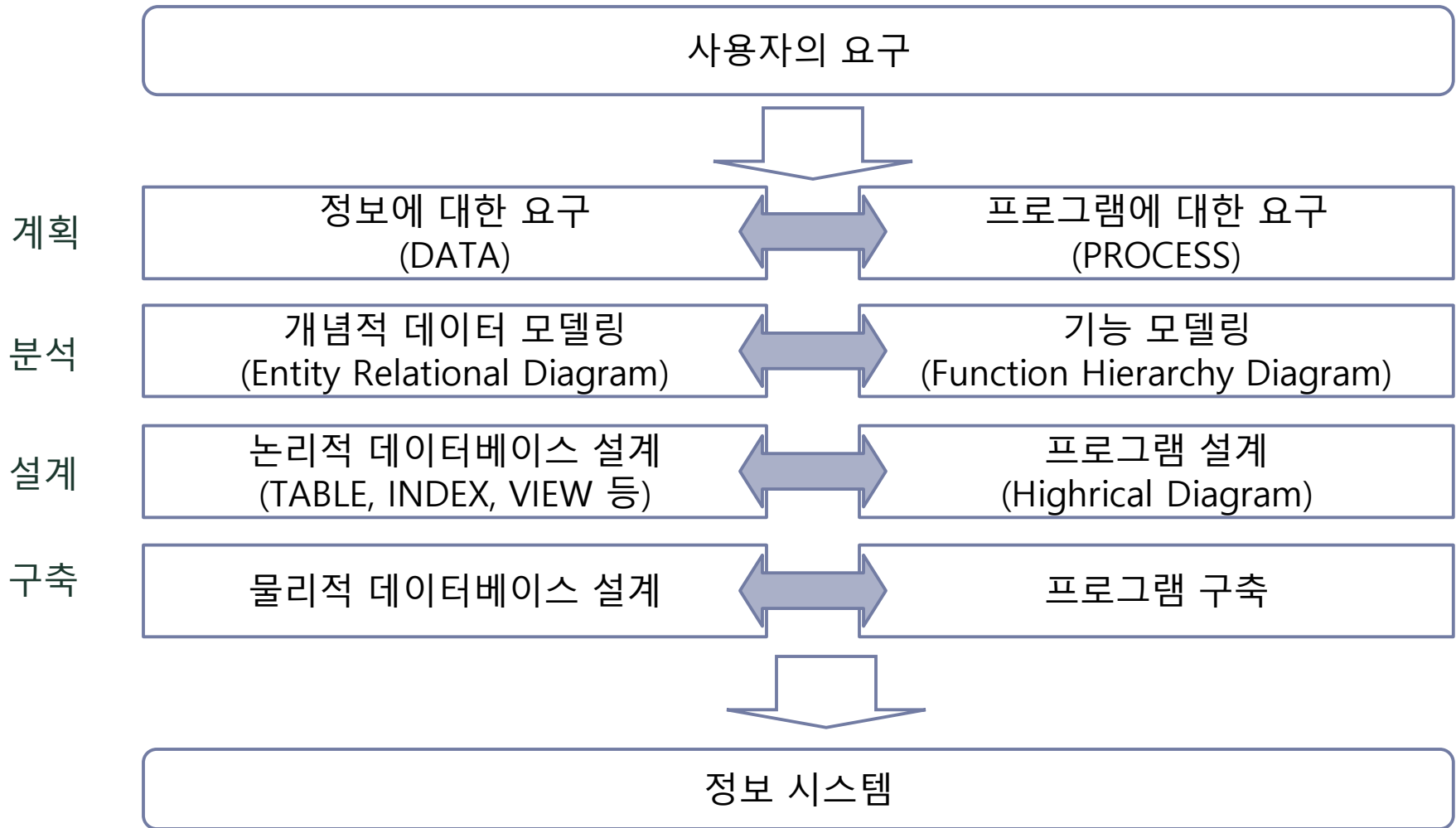
2. 개념적 데이터 모델링

- 모델링
- 개념적 데이터 모델링
- Entity 추출
- Attribute 추출
- UID 추출
- Relationship 정의
- ERD 작성

모델링 - 개념

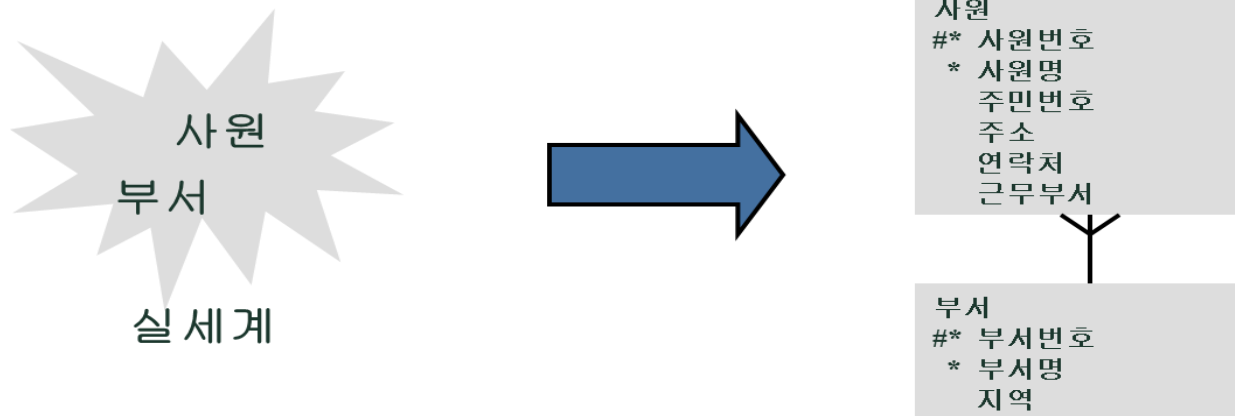
- ▶ 데이터 모델링은 복잡한 현실 세계를 단순화시켜 표현하는 것
- ▶ 데이터 모델링은 기업 업무에 대한 이해를 바탕으로 데이터에 존재하는 업무 규칙을 명확하게 표현하는 추상화 기법
- ▶ 데이터 모델링은 개념 데이터 모델링, 논리 데이터 모델링, 물리 데이터 모델링의 단계가 있음
- ▶ 목적
 - 연관조직의 정보요구에 대한 정확한 이해
 - 사용자, 설계자, 개발자 간에 효율적인 의사소통의 수단
 - 고품질의 S/W와 유지보수 비용의 감소효과

모델링 - 모델링 단계

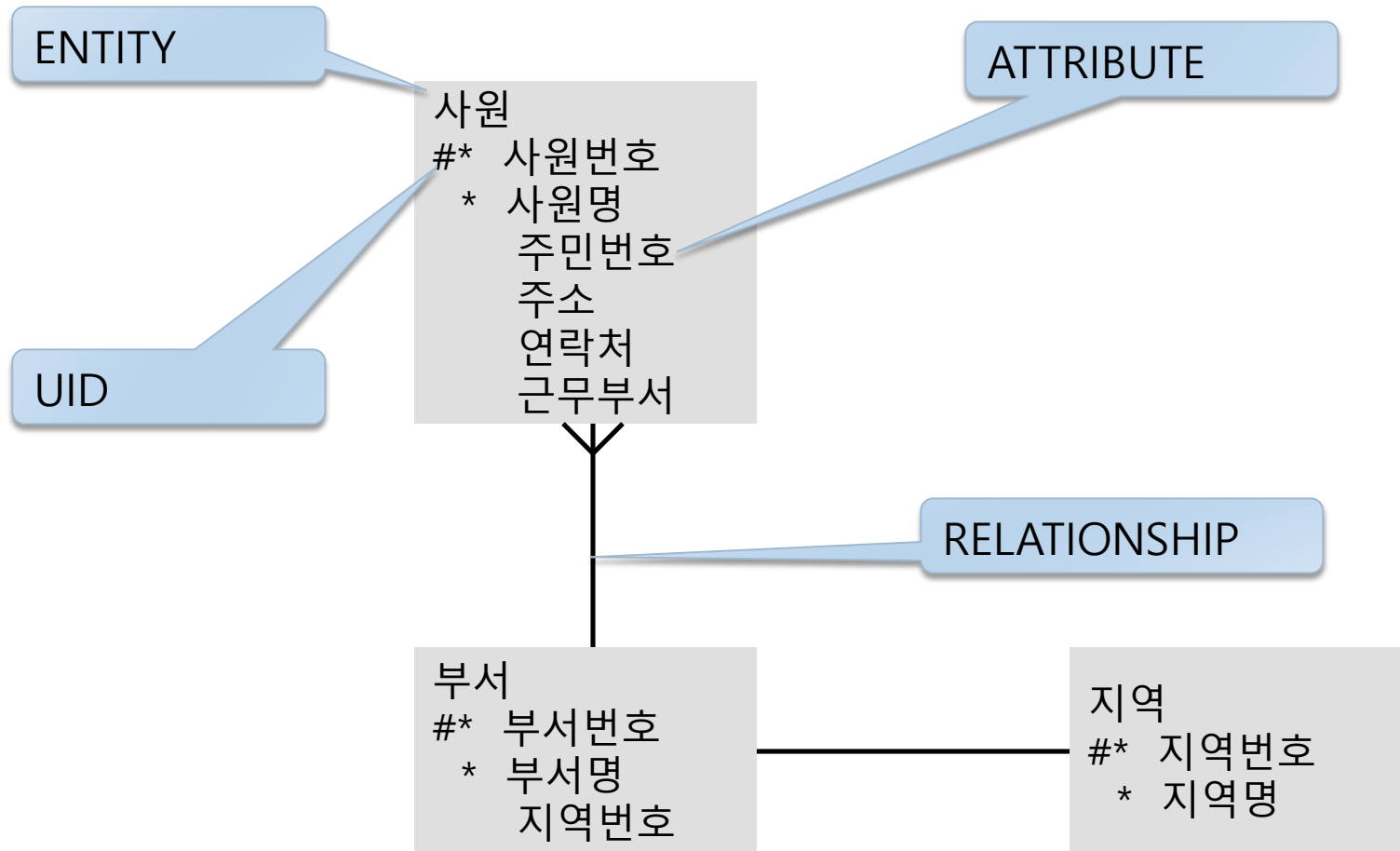


개념적 데이터 모델링 - 개요

- ▶ 개념적 데이터 모델링이란 ISP(Information Strategy Facility) 단계에서 이루어지며 실 세계의 현상을 알기 쉽고 체계적으로 모형화 해 놓은 것
- ▶ 시스템적인 측면(H/W, DBMS, O/S)이 아닌 실 세계 그대로를 표현해야 함
- ▶ 프로세스와는 독립적으로 사용자의 관점에서 인식하고 분석해야 함



개념적 데이터 모델링 - 용어 이해(1/2)



개념적 데이터 모델링 - 용어 이해(2/2)

▶ Entity

- 기업에서 지속적인 관심을 가지고 정보화를 해야 하는 대상
- 영속적이며 식별 가능한 Data 요소를 가짐

▶ Attribute

- Entity를 구성하는 고유한 정보들

▶ UID

- Unique Identifier
- 하나의 ENTITY를 대표하는 속성

▶ Relationship

- 객체 간의 직접적인 관계를 의미
- 각 ENTITY 간의 업무적 상관관계를 도식화 한 것

ENTITY 추출

- ▶ ENTITY 후보 수집대상
 - 사용자 요구사항, 장표/보고서, Data Flow Diagram, ...
- ▶ 1. 명사로 된 단어 찾기
- ▶ 2. 비즈니스 분석내용 중에서 찾기 (범위 내에서 찾기)
- ▶ 3. 명사로 된 단어 중에 같은 의미로 사용되면서 다르게 표현되는 단어는 제외할 것(반드시 사용해야 한다면 괄호 이용하여 표현)
- ▶ 4. 속성으로 표현되는 단어는 제외하고 어떤 ENTITY 에 포함되는지 파악하기

고객

주문

제품

Attribute 추출

- ▶ ENTITY 내에서 관리해야 할 정보들의 항목으로 최소단위까지 분할함
 - 하나의 엔티티는 여러 개의 속성으로 구성됨
- ▶ 정보 항목의 최소 단위이어야 하며 업무적 성격에 따라 단어가 독자적인 성질을 가질 수 있어야 함
- ▶ 유추 가능한 속성은 제외시킬 것

Entity	Attribute
고객	고객번호, 고객명, 전화번호, 주소, 신용도, 기재사항
주문	주문번호, 주문일자, 선적일자, 지불방법
제품	제품번호, 제품명, 제품설명, 제안가격, 단위

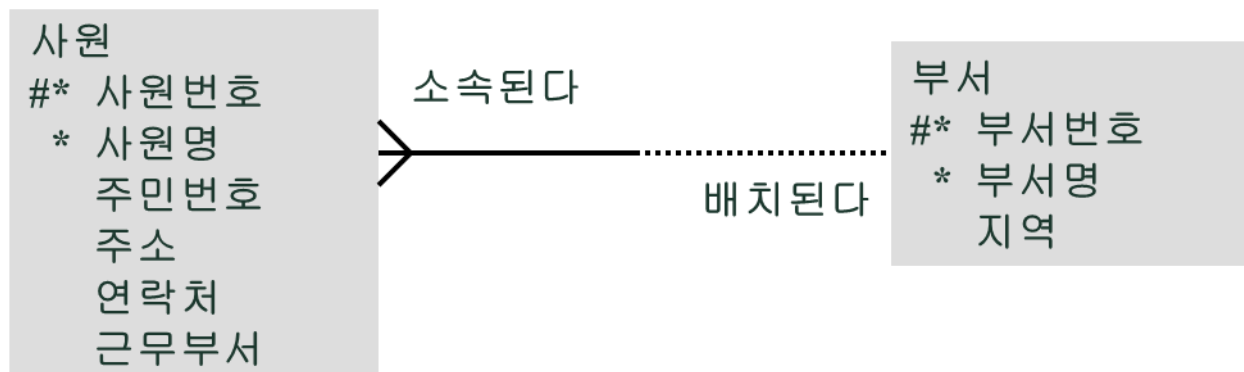
UID 추출

- ▶ 여러 개의 속성 중에 ENTITY를 대표할 수 있는 속성을 찾음
- ▶ 여러 개의 속성이 결합되어 하나의 식별자가 될 수도 있음

Entity	Identifier
제품	제품번호
고객	고객번호
주문	주문번호

Relationship 정의 - 개요(1/2)

- ▶ Relationship이란 객체 간의 직접적인 관계를 의미하며 간접적인 관계는 배제해야 함
- ▶ 각 ENTITY 간의 업무적 상관관계를 도식화 한 것
- ▶ 실체와 실체 간에 중복되는 속성을 가졌으면 Relation 이 될 수 있음
- ▶ 실 세계에서 사용되는 동사적 단어들이 Relation이 될 수 있음

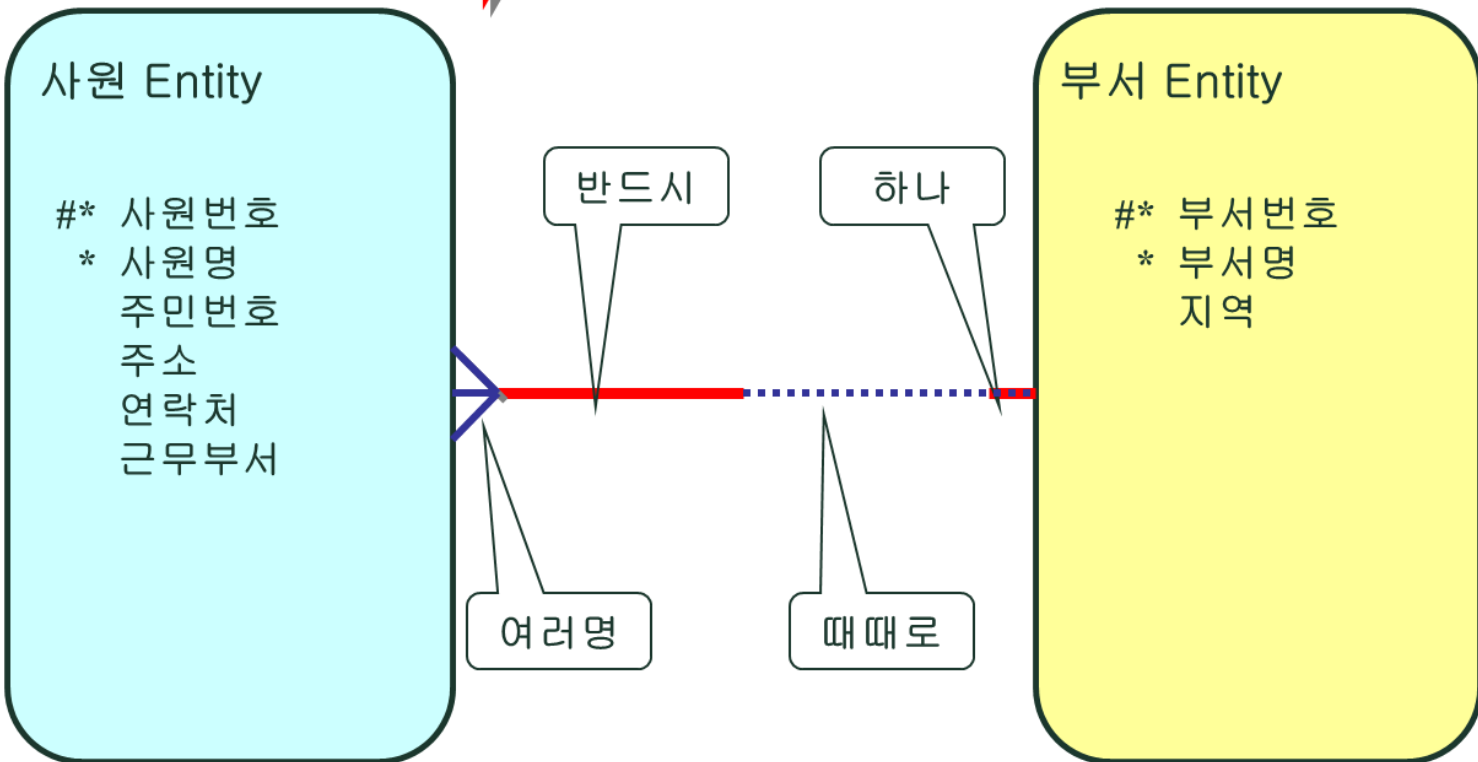


Relationship 정의 - 개요(2/2)

사원과 부서 관계



한명의 사원은 **하나**의 부서에 **반드시** 근무한다.

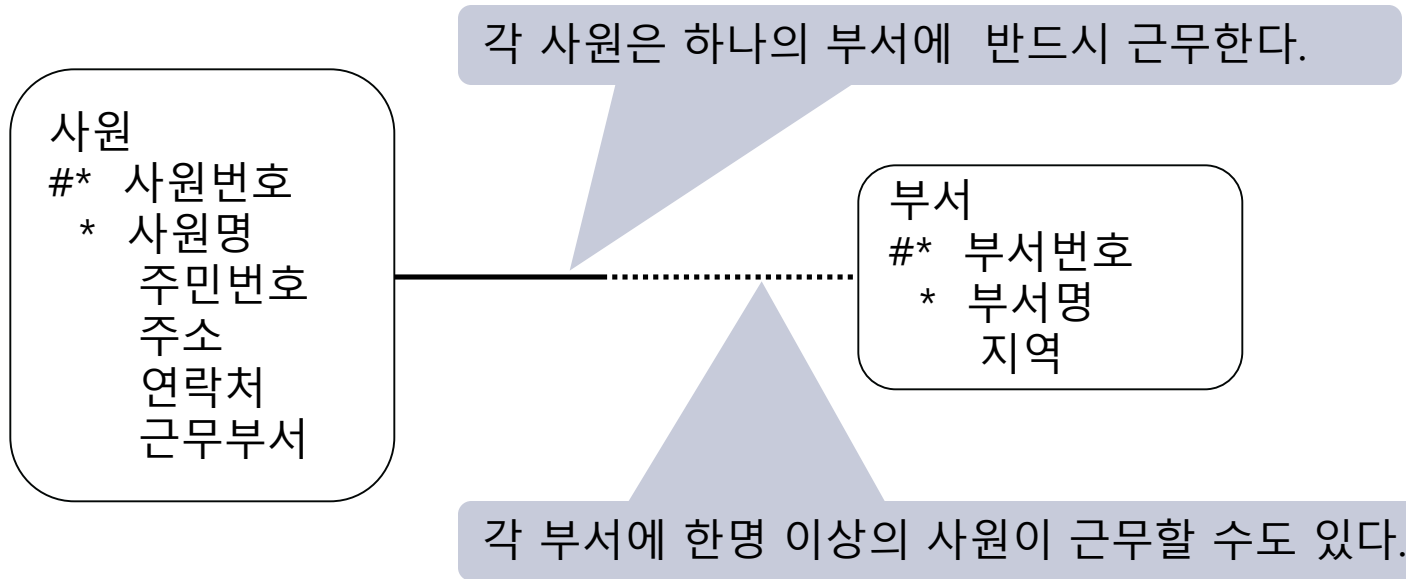


하나의 부서에는 **여러명**의 사원들이 **때때로** 근무한다.



부서와 사원 관계

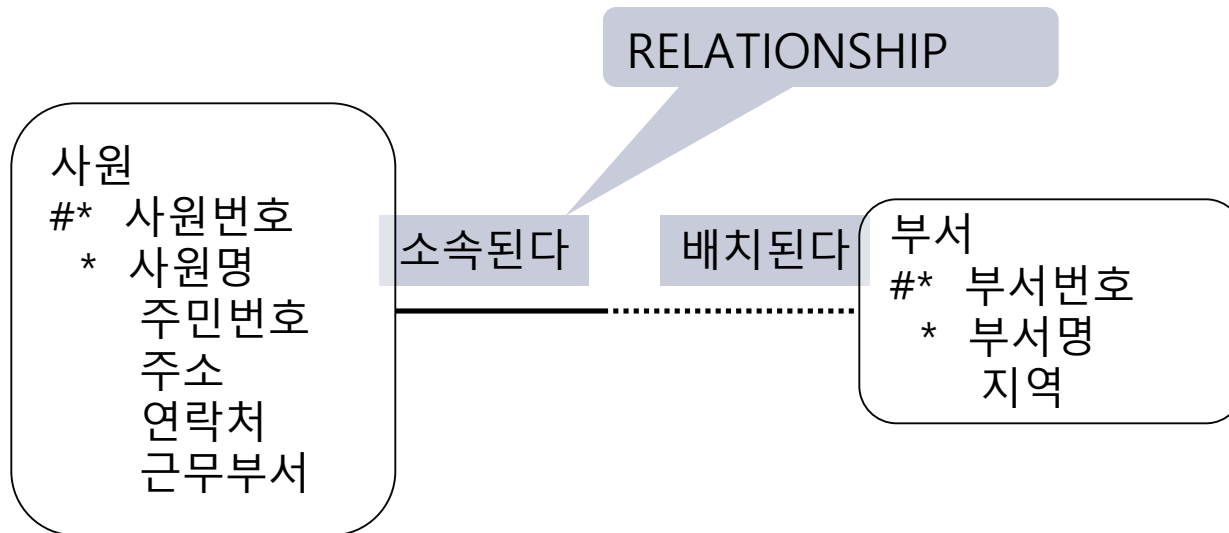
Relationship 정의 - 작성 단계 1



CONDITIONAL은 각 ENTITY 간의 존재여부를 표시한 것임

- 필수사항(Mandatory)은 실선(—) 으로 표시하고 상대 ENTITY에 대해 해당 ENTITY에 조건을 만족하는 실체가 반드시 존재할 하는 경우에 표시
- 선택사항(Optional)은 점선(-----) 으로 표시하고 상대 ENTITY에 대해 해당 ENTITY에 조건을 만족하는 실체가 존재할 수도, 존재하지 않을 수도 있을 경우 표시

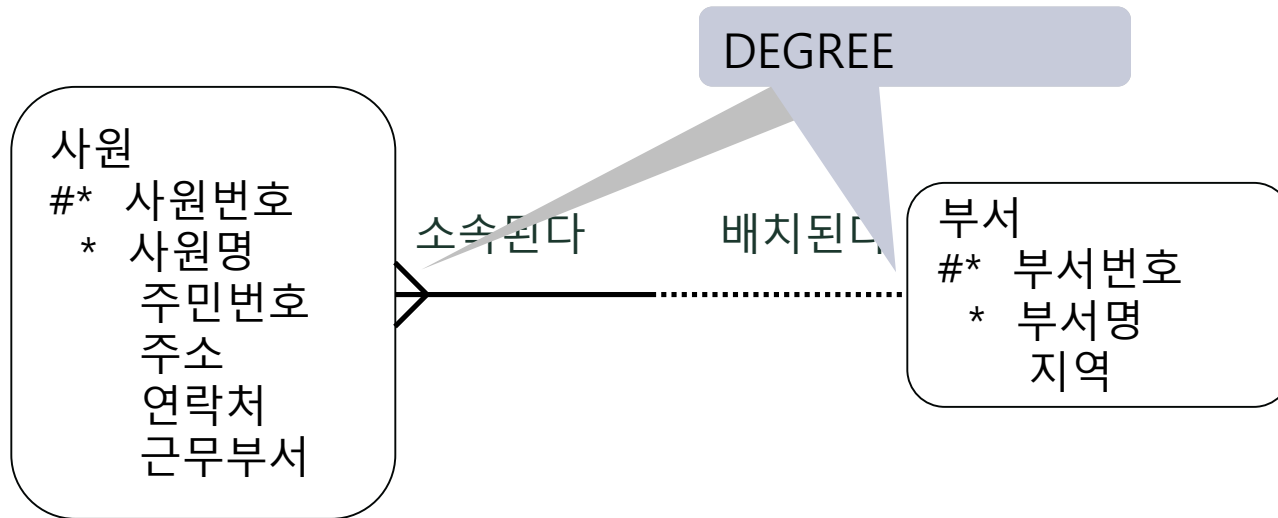
Relationship 정의 - 작성 단계 2



RELATIONSHIP은 각 ENTITY 간의 관계를 표시한 것

- 2개의 ENTITY 간에 CONDITIONAL을 표기한 후 해당 ENTITY의 가까운 위치에 관계 명칭을 표기 함(관계가 애매모호한 경우에는 능동형과 수동형)
- RELATION은 실세계의 해당 ENTITY에서 발생하는 동사적 단어들을 표기함

Relationship 정의 - 작성 단계 3



DEGREE는 해당 ENTITY의 실체가 상대 ENTITY의 실체에 대해 몇 개의 상관관계를 가지고 있는지를 표시한 것

- 1:1 , 1:N, N:M 관계에 맞는 적합한 차수를 선택함

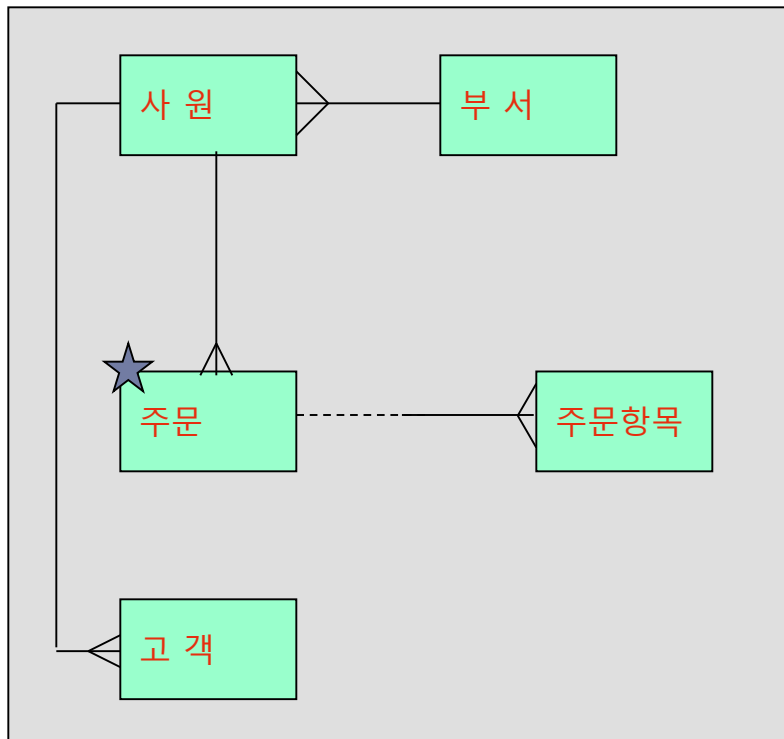
Relationship 정의 - Relationship Matrix

Parents \ Child	사원	부서	고객	주문	주문항목
사원	상관번호	부서번호	관리되는 관리하는	✓	
부서					
고객				주문 접수되는 주문하는	✓
주문					
주문항목					

- 1) 부모와 자식 Entity에 순서대로 추출된 Entity명을 작성
- 2) Entity 수가 많으면 30~40 개씩 분할해서 작성
- 3) 관계 여부가 확인되면 구체적인 속성명, 동사, 기호 등을 표시

ERD 작성(1/4)

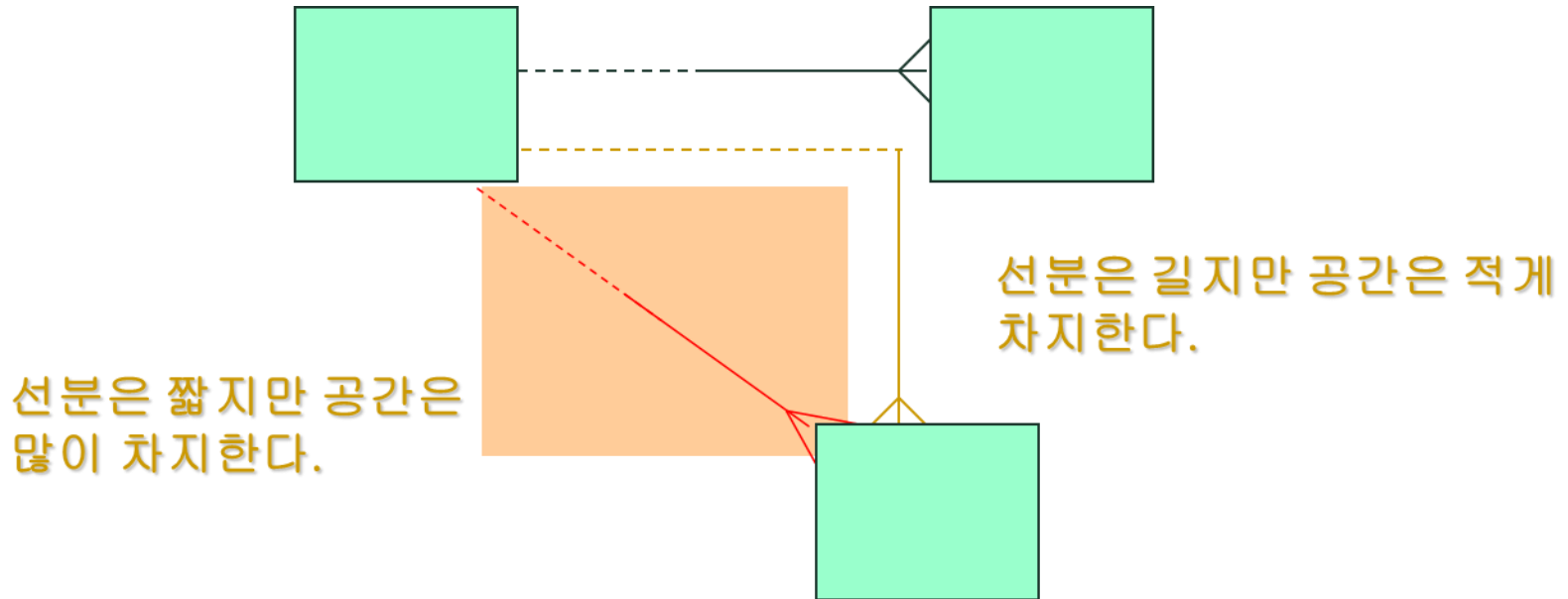
- ▶ I. "Relationship Matrix"에서 다른 실체와 가장 많은 관계를 가지고 있는 실체를 중앙에 배치



Parents Child	사원	부서	고객	주문	주문항목
사원		부서번호	담당사원	✓	
부서					
고객				고객번호	✓
주문					
주문항목					

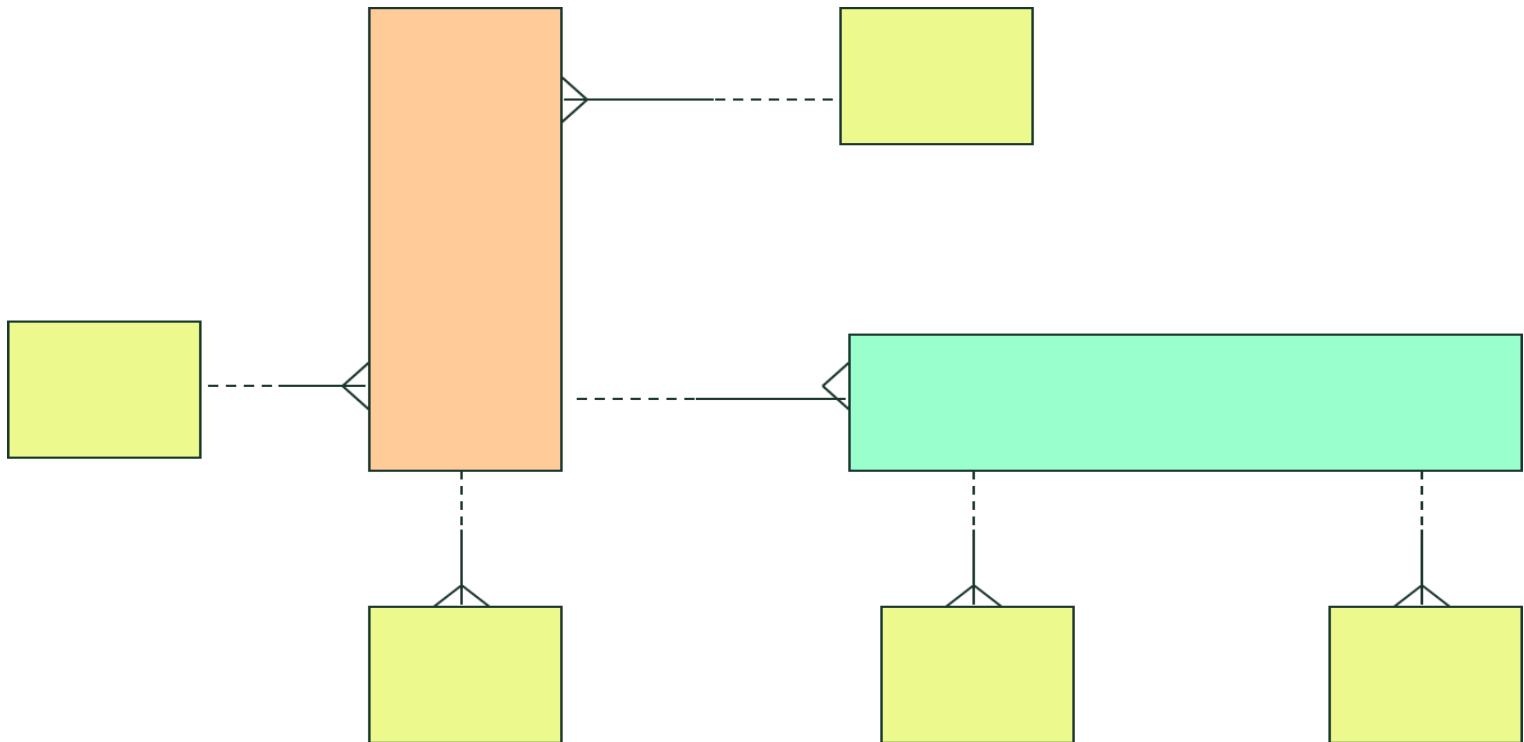
ERD 작성(2/4)

- ▶ 2. 나중에 어떤 엔터티가 추가될지 모르므로 가능한 공간을 절약을

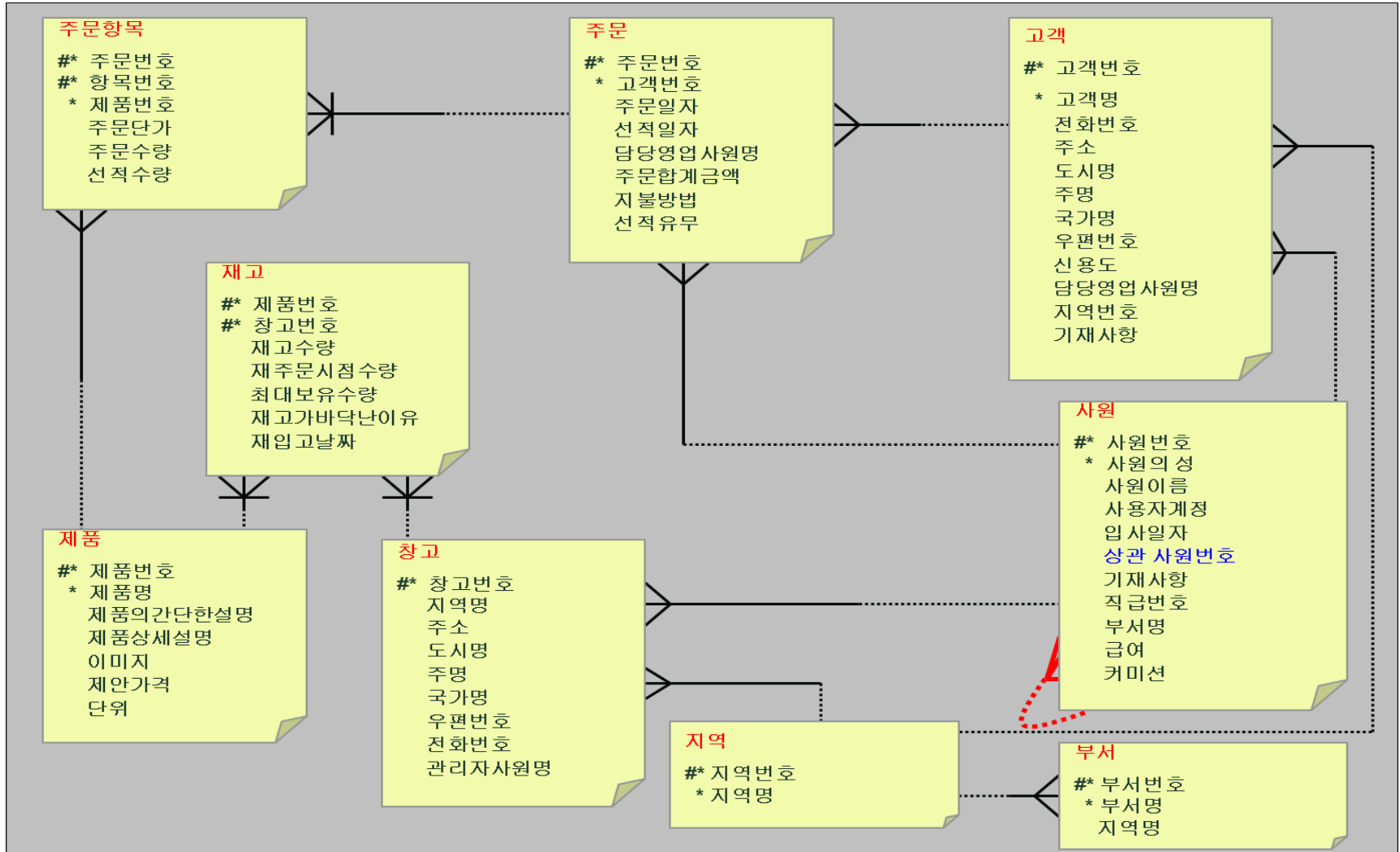


ERD 작성(3/4)

- ▶ 3. 상황에 따라 가로나 세로를 확장하여 그리면 사선이 없어지고 나름의 특징이 부여됨



ERD 작성(4/4)

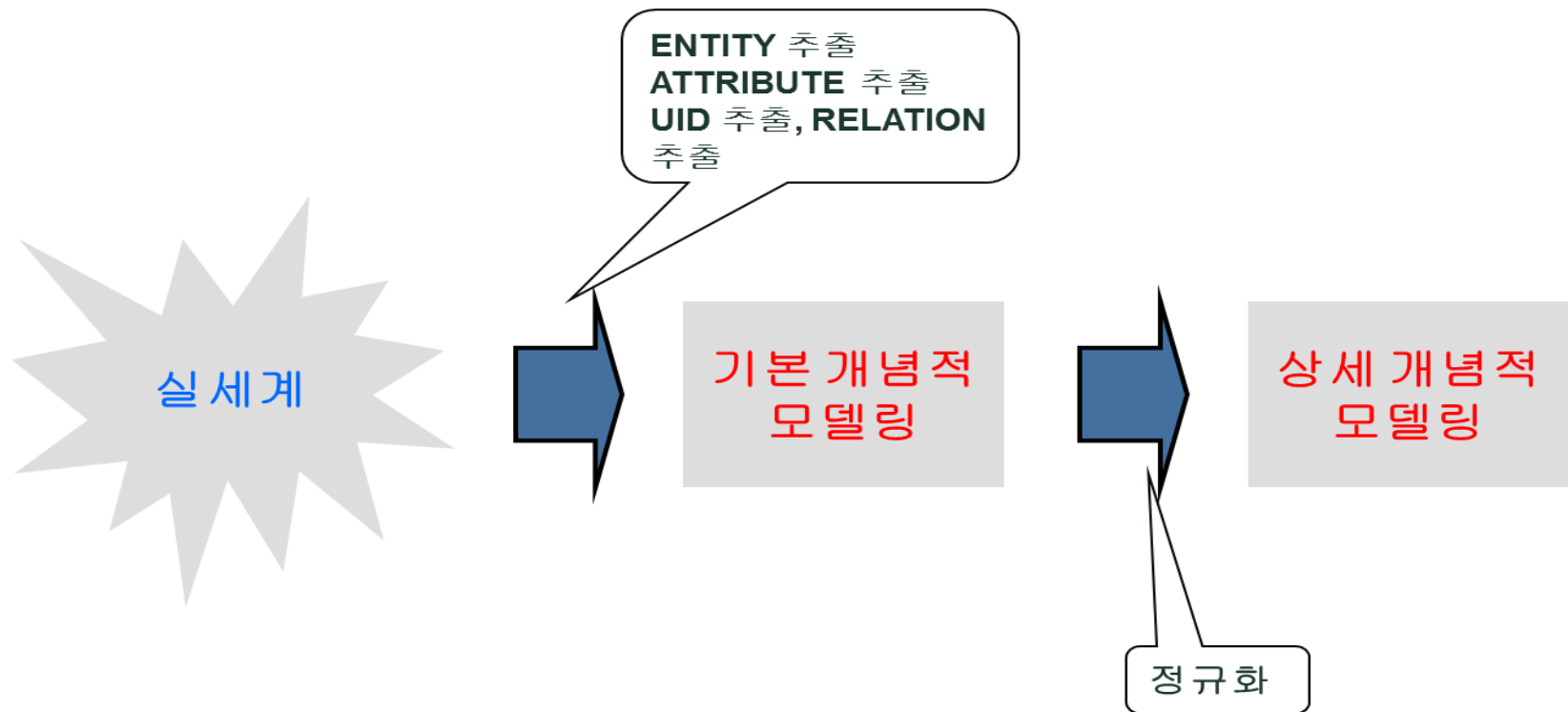


3. 상세 개념적 데이터 모델링

- 상세 개념적 모델링 개요
- 정규화가 필요한 이유
- 정규화의 장/단점
- 정규화 적용

상세 개념적 모델링 개요

- ▶ 기본 개념적 모델링은 실 세계의 현상을 그대로 표현한 것이라면 상세 개념적 모델링은 보다 명확하게 ENTITY를 표현하고 ATTRIBUTE를 제거 또는 추가하는 과정



정규화가 필요한 이유

- ▶ 데이터의 중복성을 제거할 수 있음
- ▶ 데이터 모형을 단순화 시킬 수 있음
- ▶ ATTRIBUTE의 배열 상태를 검증해 볼 수 있음
- ▶ ENTITY,ATTRIBUTE의 누락여부를 검증해 볼 수 있음
- ▶ 데이터 모형의 안정성을 유지시킬 수 있음

정규화의 장/단점

정규화 정도가 높은 경우	정규화 정도가 낮은 경우
1) 유연한 데이터를 구축할 수 있음	1) 데이터의 결합처리가 감소됨
2) 데이터의 정확성이 높아짐	2) 물리적 접근이 단순함
3) 물리적 접근이 복잡해짐	3) 데이터에 많은 Lock이 발생함
4) 길이가 짧은 데이터가 생김	4) 길이가 긴 데이터가 생길 수 있음

정규화 적용 - 개요

- ▶ 정규화(Normalization)란 개념적 모델링 과정에서 정의된 각 ENTITY에서 발생하는 데이터의 중복성을 제거하고 새로운 ENTITY를 창출 해내는 과정을 의미

제 1 정규화

Entity 내의 모든 속성은 반드시 하나의 값을 가져야 함



제 2 정규화

Entity 내의 모든 속성은 반드시 식별자에 종속 되어야 함



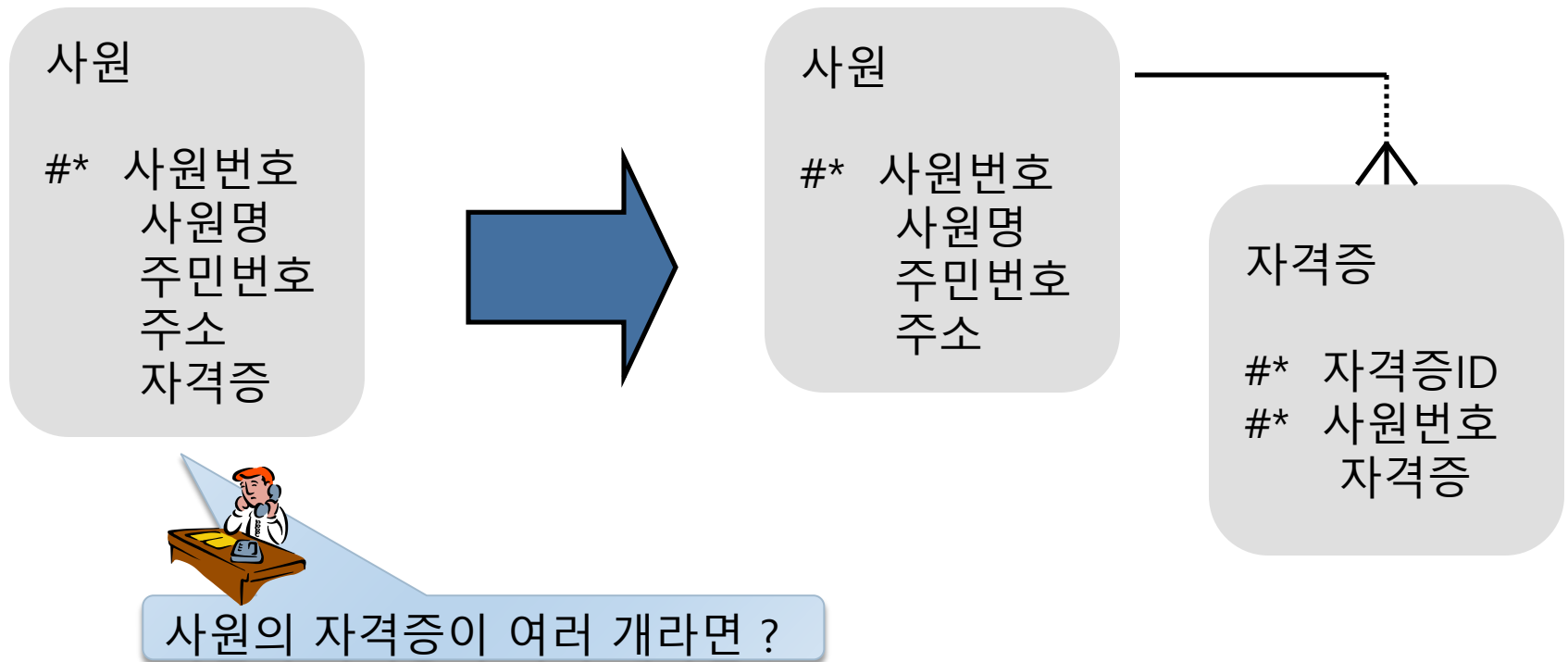
제 3 정규화

Entity 내의 식별자를 제외한 모든 속성은 종속될 수 없음

정규화 적용 - 제 1 정규화

Entity 내의 모든 속성은 반드시 하나의 값을 가져야 함(중복성의 제거)

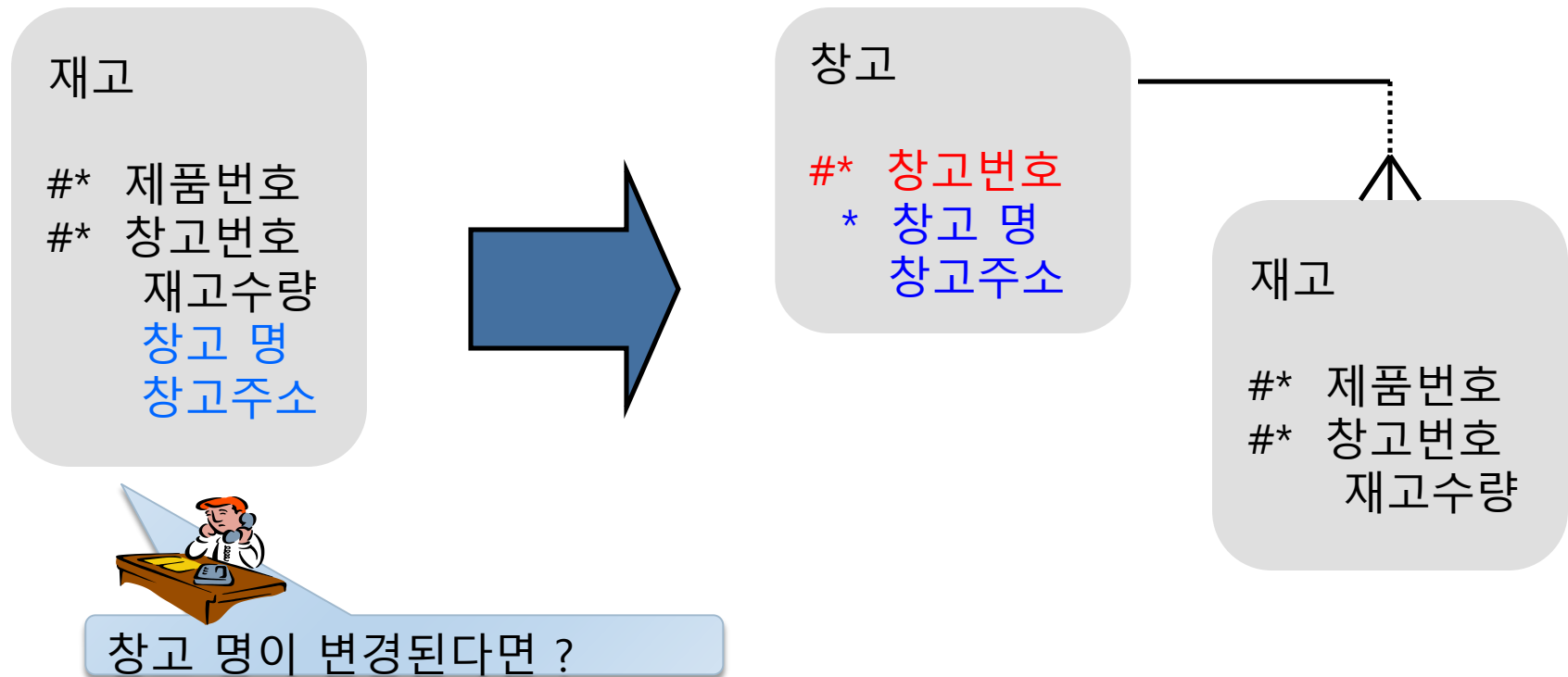
새로운 Entity를 생성하고 UID를 부여한 다음 1:N의 관계를 부여함



정규화 적용 - 제 2 정규화

Entity 내의 모든 속성은 반드시 UID에 종속 되어야 함(종속성의 제거)

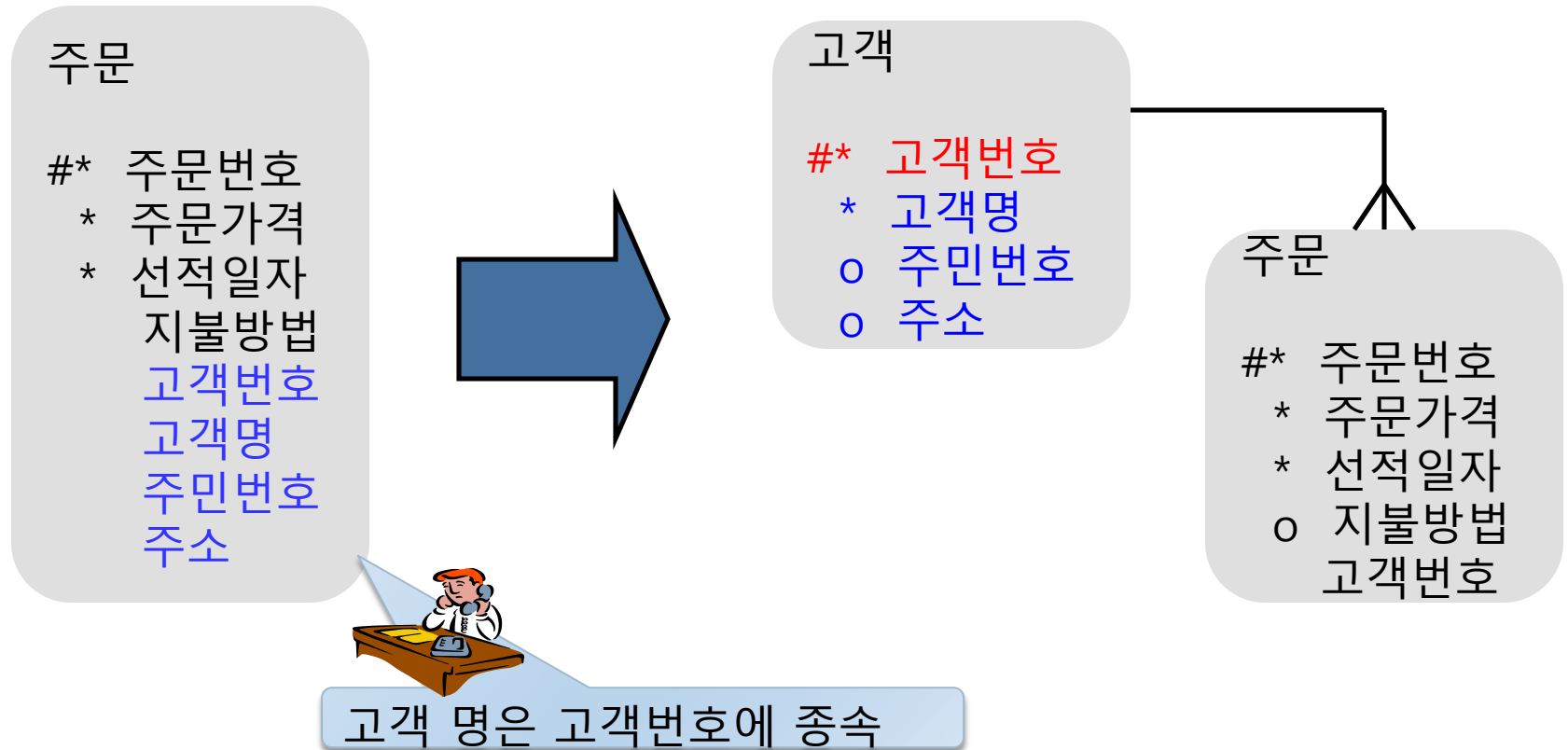
새로운 Entity를 생성하고 UID를 부여한 다음 1:N의 관계를 부여함



정규화 적용 - 제 3 정규화

Entity 내의 식별자를 제외한 모든 속성은 종속될 수 없음

새로운 Entity를 생성하고 UID를 부여한 다음 1:N의 관계를 부여함

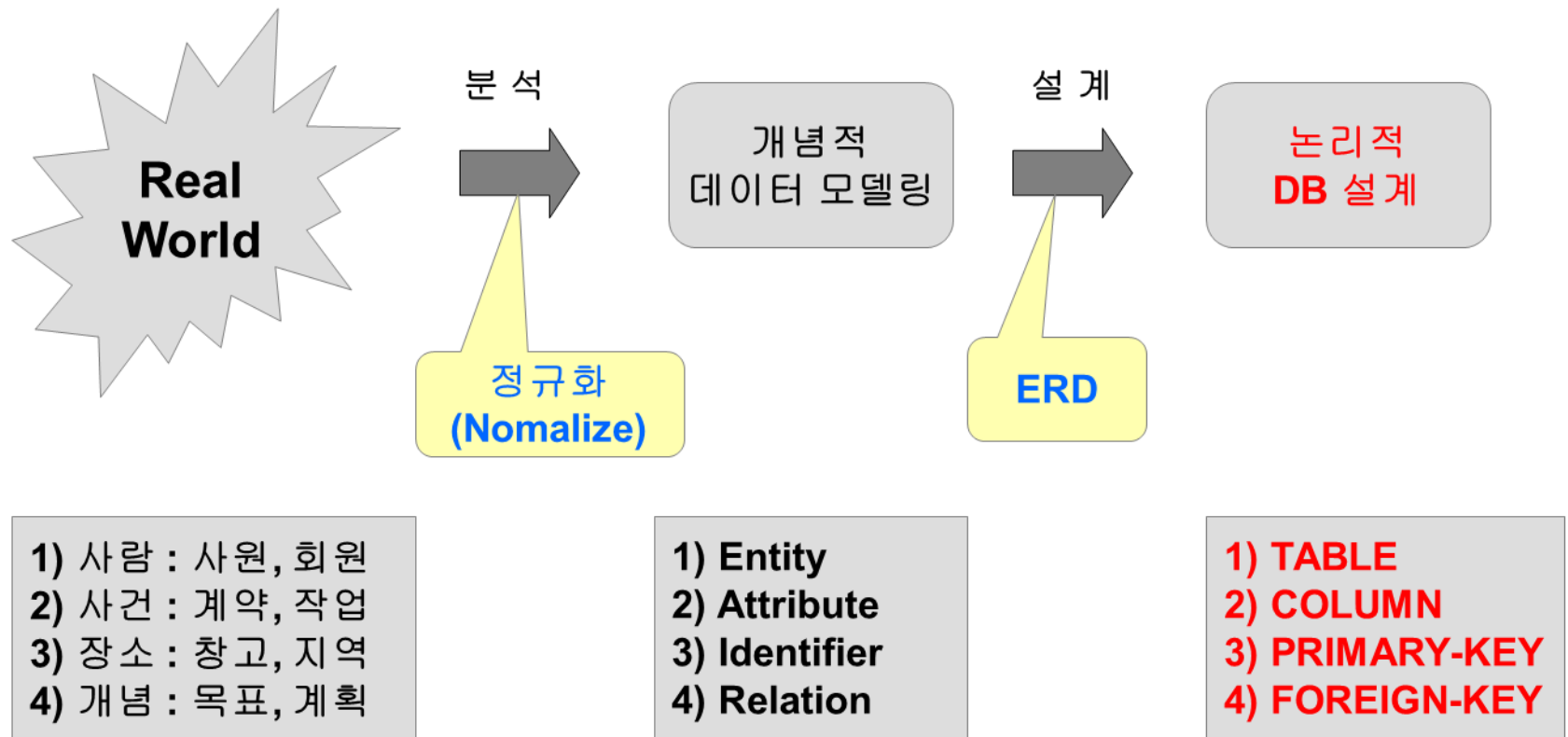


4. 논리적 데이터 모델링

- 논리적 데이터 모델링 개요
- RDBMS
- 매핑
- DataBase의 데이터 타입

논리적 데이터 모델링 개요

- ▶ 개념적 모델링 과정을 통해 만들어진 개념적 구조(ERD)들을 DBMS가 처리할 수 있는 객체로 생성하는 과정



RDBMS - 개요

▶ DataBase

- 한 조직의 여러 응용 시스템에서 공용할 수 있도록 중복되는 데이터를 최소화하여 통합/저장한 운영 데이터 집합을 의미

▶ DBMS(DataBase Management System)

- 데이터베이스에서 데이터를 저장/검색/수정하는 데이터베이스 전용 관리 프로그램

▶ RDBMS(Relational DataBase Management System)

- 모든 데이터를 2차원 테이블형태로 표현하고, 테이블 사이의 비즈니스 관계를 도출하는 구조를 가진 데이터베이스유형
- 데이터의 중복을 최소화 할 수 있으며, 업무 변화에 대한 적응력이 우수

테이블 : DIVISION

D_CODE	D_NAME
D1	전자사업부

테이블 : TITLE

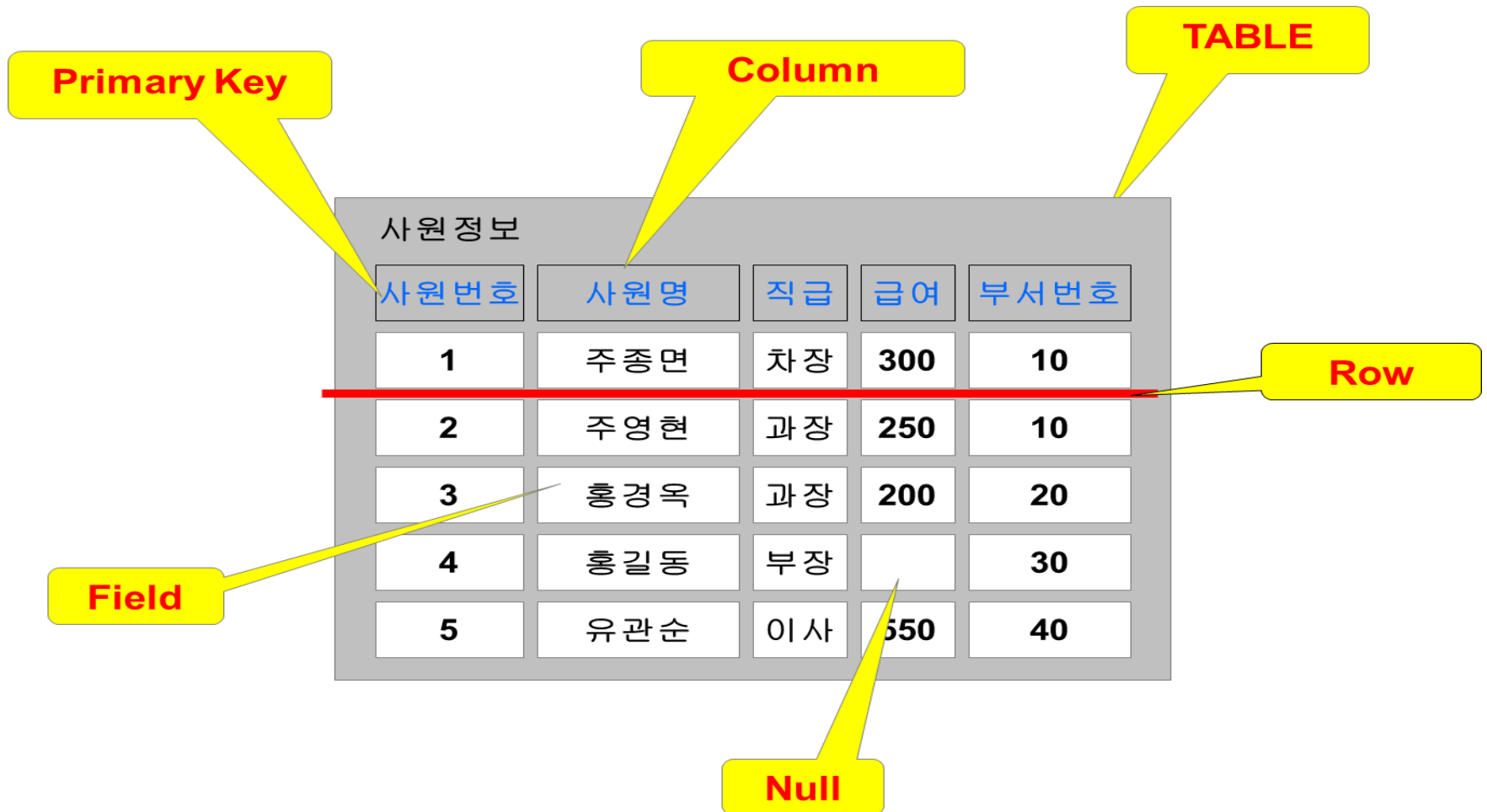
T_CODE	DESCRIPTION
T1	Director

테이블 : EMPLOYEE

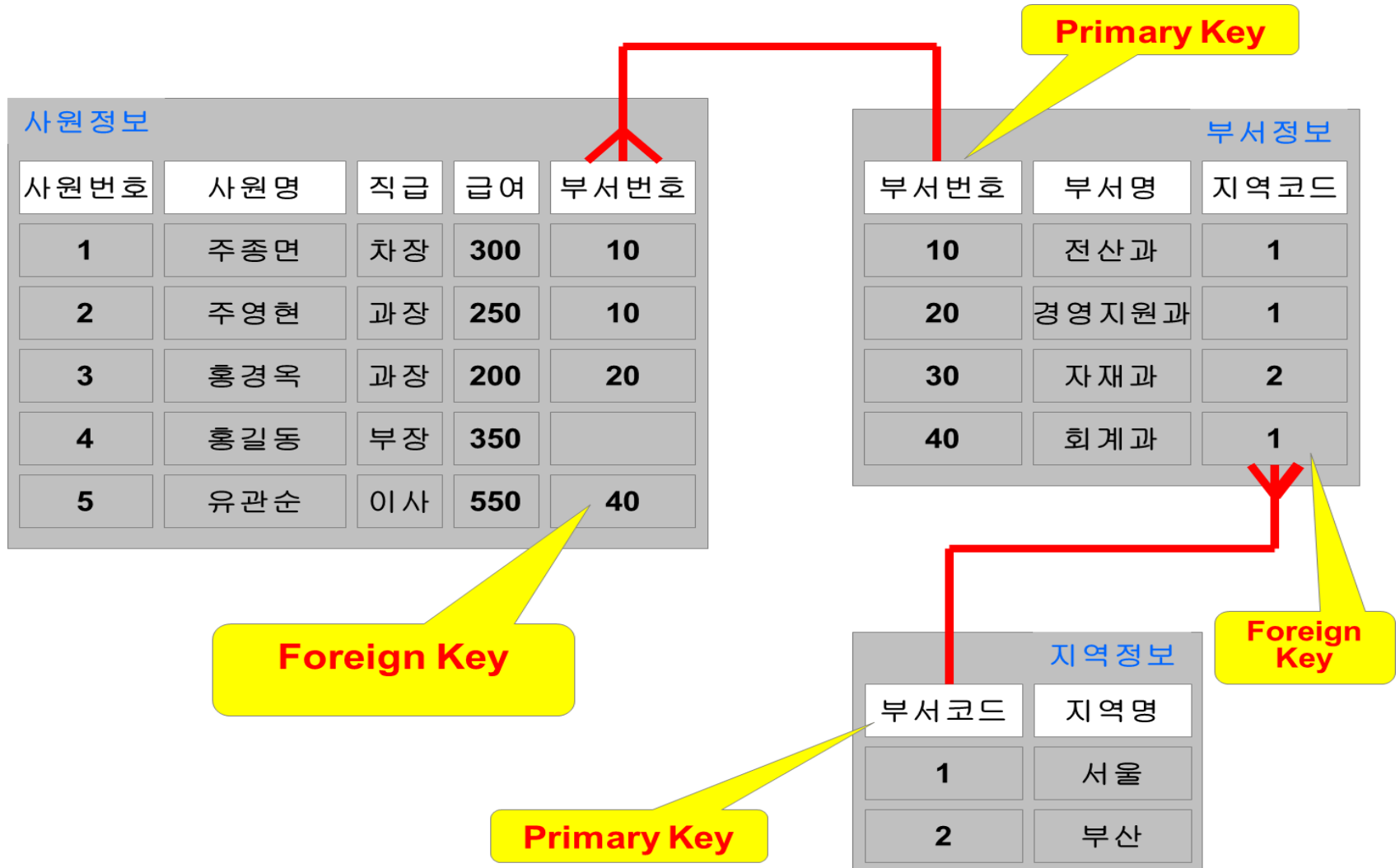
NAME	T_CODE	D_CODE	AGE
홍길동	T1	D1	42



RDBMS - 주요 용어(1/2)



RDBMS - 주요 용어(2/2)



매핑

개념적 모델링

ENTITY

ATTRIBUTE

UID

RELATIONSHIP

Attribute의 mandatory
optional



논리적 DB 모델링

TABLE

COLUMN

PRIMARY-KEY

FOREIGN-KEY

NOT NULL
NULL

DataBase의 데이터 타입

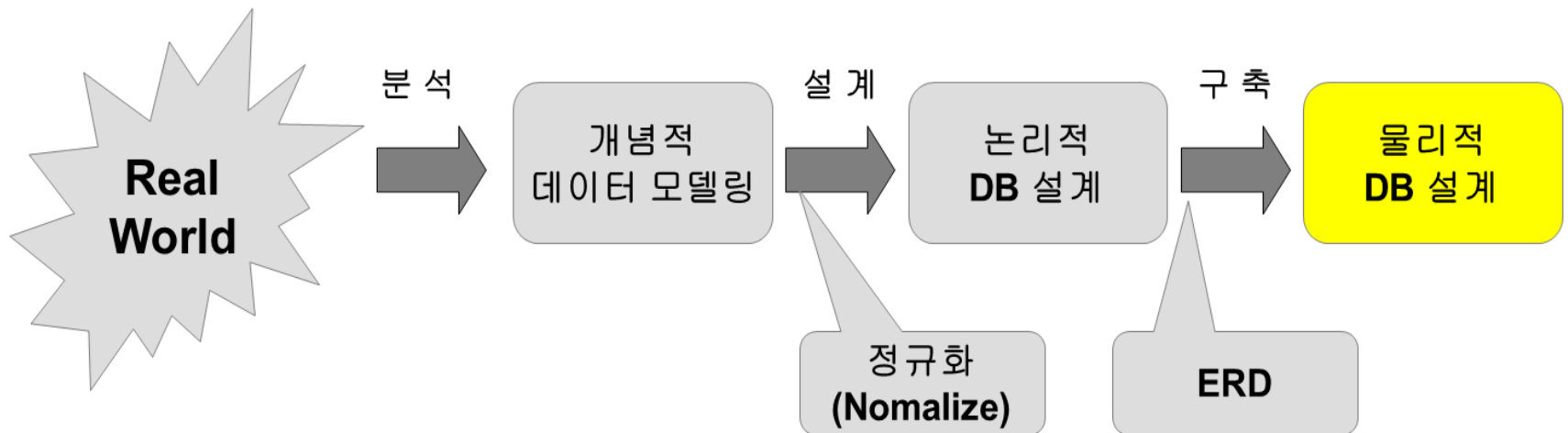
구 분	형 식
CHAR	고정길이의 문자 DATA를 4000 Byte 저장할 수 있다.
VARCHAR2	가변길이의 문자 DATA를 4000 Byte 저장할 수 있다.
NUMBER NUMBER(P, s)	숫자 값을 -38 자리 수 ~ +38 자리 수를 저장 P는 전체 자리 수 , S는 소수점 이하 자리 수
BLOB(LONG RAW) CLOB(LONG)	Binary 데이터를 4 GB 저장할 수 있다 Text 데이터를 4 GB 저장할 수 있다.
DATE	날자 값을 저장할 수 있다.
TIMESTAMP	년, 월, 일, 시, 분, 초, Milli-Second까지 보여준다.
TIMESTAMP WITH TIME ZONE	지역시간과 세계표준시의 차이 난 시간정보를 보여줌
TIMESTAMP WITH LOCAL ZONE	지역시간으로 변환하여 저장해 줍니다.
INTERVAL YEAR TO MONTH	정의된 시간대로 날짜를 계산하여 저장 해준다.

5. 물리적 설계

- 물리적 설계의 개요
- 테이블 설계
- 인덱스 설계

물리적 설계의 개요

- ▶ 개념적 모델링 과정과 논리적 DB 설계과정을 통해 산출된 ERD를 근거로 DBMS를 선택한 후 설치하는 단계
- ▶ 최종 설계된 ERD 모델에 대해 테이블의 물리적 저장구조에 대한 설계를 한 후 스키마를 생성



테이블 설계

논리적 DB 설계

물리적 DB 설계

```
CREATE TABLE s_emp
(id          NUMBER(7)
last_name    VARCHAR2(25)
first_name   VARCHAR2(25),
userid       VARCHAR2(8)
start_date   DATE,
comments     VARCHAR2(255),
manager_id   NUMBER(7),
title        VARCHAR2(25),
dept_id      NUMBER(7)

salary       NUMBER(11, 2),
commission_pct NUMBER(4, 2),
```

```
CONSTRAINT s_emp_id_nn      NOT NULL,
CONSTRAINT s_emp_last_name_nn NOT NULL,

CONSTRAINT s_emp_user_id    NOT NULL,
```

```
CONSTRAINT s_emp_dept_id_fk
References s_dept(id),
```

```
CONSTRAINT s_emp_id_pk      PRIMARY KEY (id),
CONSTRAINT s_emp_userid_uk  UNIQUE (userid),
CONSTRAINT s_emp_commission_pct_ck
CHECK (commission_pct IN (10, 12.5, 15, 17.5, 20)))
```

TABSPACE SALES

```
STORAGE (INITIAL      10K      NEXT      10K
          MINEXTENTS   1        MAXEXTENTS UNLIMITED
          PCTINCREASE  50)
PCTFREE 10    PCTUSED   40    FREELIST 1
PARALLEL          CACHE;
```

인덱스 설계

- ▶ 6 블록 이상의 큰 테이블에 적용
 - 1 블록 = DB_BLOCK_SIZE의 값
- ▶ 조회/출력 조건으로 많이 검색 되어지는 컬럼
- ▶ PRIMARY-KEY와 UNIQUE-KEY 제약조건은 인덱스가 자동 생성
- ▶ 컬럼의 분포도가 10 % 내외인 경우 적용

$$\text{분포도} = \frac{1}{\text{컬럼값의 종류}} * 100 = \frac{\text{컬럼값의 평균 로우수}}{\text{테이블의 총 로우수}} * 100$$



DataBase - SQL

목 차

1. DataBase 개요
2. 자료형 및 DB 주요용어
3. 기본적인 SQL - select
4. 선택적 데이터 조회 및 정렬
5. 단일 행 함수
6. 여러 테이블에서의 데이터 조회
7. 그룹함수를 사용한 데이터 조회
8. Sub Query
9. 데이터 조작
10. 테이블 생성과 관리
11. View / Index / Sequence

1. DataBase 개요

- DataBase
- DataBase 등장배경
- DataBase 특징
- DBMS개요 및 유형
- RDBMS의 특징

DataBase

- ▶ 한 조직의 여러 응용 시스템에서 공용할 수 있도록 중복되는 데이터를 최소화하여 통합/저장한 운영 데이터 집합을 의미

DataBase 등장배경

- ▶ 지식 기반 사회에서 대용량 데이터 관리에 대한 필요성
- ▶ 파일 시스템의 문제점
 - 응용 프로그램과 데이터의 종속성
 - 다수 사용자들의 정보 공유에 대한 문제
- ▶ 1970년대 데이터베이스 관리 시스템 등장
 - 컴퓨터에 저장된 대량의 데이터를 체계적으로 관리하고 사용자의 원하는 정보를 효과적으로 검색하기 위한 소프트웨어
 - 데이터베이스
 - ▶ 데이터베이스 관리시스템에 의해 관리되는 데이터의 집합
 - SQL 언어
 - ▶ 현재 가장 널리 사용되는 데이터베이스 언어
 - ▶ 미국표준연구소(ANSI)와 국제표준기구(ISO)에서 관계형 데이터베이스 표준 언어로 채택

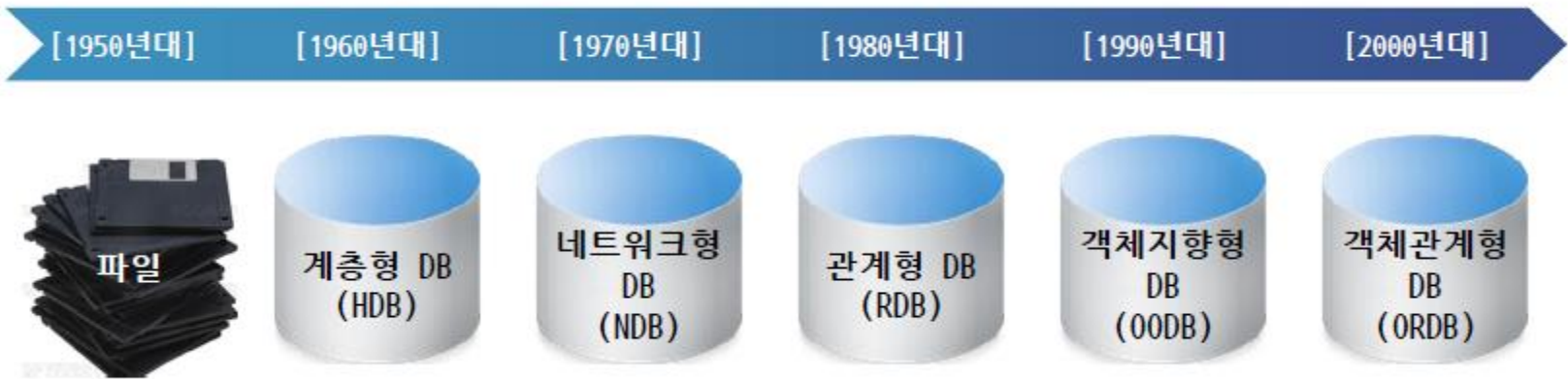
DataBase 특징

- ▶ 중복되는 데이터를 최소화 할 수 있음
 - 저장 공간의 낭비를 줄여 효율적으로 공간을 이용
 - 데이터의 결함을 줄일 수 있음(데이터 무결성)
- ▶ 데이터를 효율적으로 공유 할 수 있음
- ▶ 데이터 보안 향상
 - 권한에 따라 데이터의 접근이 제한됨
- ▶ 저장된 데이터는 시스템의 비즈니스 로직에서 의사 결정의 수단으로 사용됨
- ▶ 데이터를 편리하게 관리할 수 있음
 - 데이터 백업, 복구 등 지원

DBMS개요 및 유형

▶ DBMS

- 방대한 양의 데이터를 편리하게 관리하고 효율적으로 저장하고 검색할 수 있는 환경을 제공해 주는 시스템 소프트웨어



RDBMS의 특징

- ▶ 1970년 Dr. E.F Codd에 의해 관계형 모델이 처음으로 제안되었음
- ▶ 데이터 중복을 최소화 할 수 있음
- ▶ 데이터의 무결성을 유지할 수 있음
- ▶ 데이터의 효율적으로 공유 가능
- ▶ 데이터를 일관성 있게 유지 할 수 있음
- ▶ 데이터의 불일치를 피할 수 있음
- ▶ 업무 변화에 따른 적응력이 우수함

2. 자료형 및 DB 주요용어

- SQL
- Oracle 자료형의 종류 및 특징
- Null값의 정의
- 주요용어

SQL - 개요

- ▶ Structured Query Language
- ▶ 데이터를 사용하기 위해 데이터베이스와 통신하는 언어
- ▶ 관계형 데이터베이스에서 데이터를 조회하거나 조작하기 위해 사용하는 표준 검색언어

SQL - 유형

구 분	형 식	비 고
DQL (Data Query Language)	SELECT column-1, column-2, FROM table명 WHERE 조건절 ;	검색시 사용
DML (Data Manipulation Language)	UPDATE table명; INSERT INTO table명; DELETE table명;	변경시 사용
DDL (Data Definition Language)	CREATE TABLE table명; DROP TABLE table명; ALTER TABLE table명;	Object의 생 성과변경 시
TCL (Transaction Control Language)	COMMIT; ROLLBACK; SAVEPOINT;	Transaction 종료 및 취소
DCL (Data Control Language)	GRANT; REVOKE;	권한 부여 및 취소

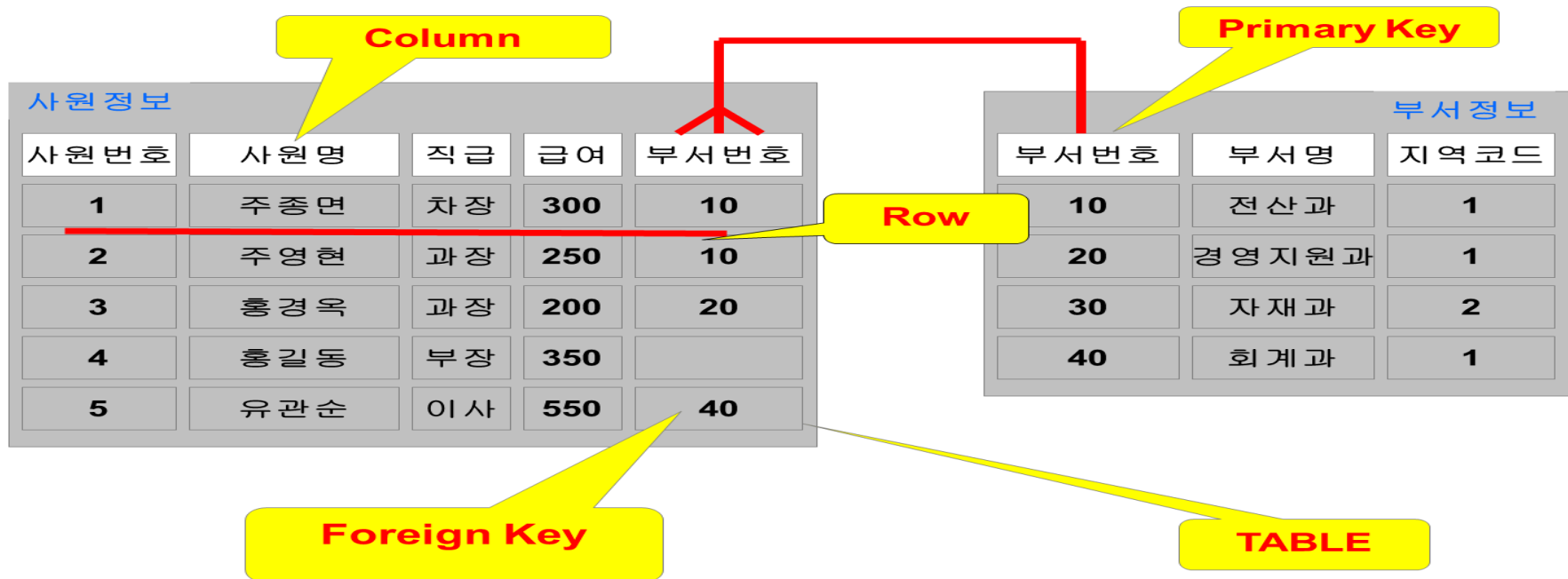
Oracle 자료형의 종류 및 특징

구 분	형 식
CHAR	고정길이의 문자 DATA를 4000 Byte 저장할 수 있다.
VARCHAR2	가변길이의 문자 DATA를 4000 Byte 저장할 수 있다.
NUMBER NUMBER(P, s)	숫자 값을 -38 자리 수 ~ +38 자리 수를 저장 P는 전체 자리 수 , S는 소수점 이하 자리 수
BLOB(LONG RAW) CLOB(LONG)	Binary 데이터를 4 GB 저장할 수 있다 Text 데이터를 4 GB 저장할 수 있다.
DATE	날자 값을 저장할 수 있다.
TIMESTAMP	년, 월, 일, 시, 분, 초, Milli-Second까지 보여준다.
TIMESTAMP WITH TIME ZONE	지역시간과 세계표준시의 차이 난 시간정보를 보여줌
TIMESTAMP WITH LOCAL ZONE	지역시간으로 변환하여 저장해 줍니다.
INTERVAL YEAR TO MONTH	정의된 시간대로 날짜를 계산하여 저장 해준다.

Null값의 정의

- ▶ 컬럼에 값이 지정(할당)되지 않은 상태를 나타내는 특별한 값
- ▶ 0(zero)나 공백과는 다름
- ▶ 기본적으로 모든 데이터타입에서 사용 가능함

주요용어(1/2)



▶ Table

- 논리모델의 Entity를 물리 모델로 매핑한 것

▶ Row

- Entity를 나타내기 위한 정보(Column)들의 집합
- Record, Tuple

주요용어(2/2)

▶ Column

- Entity의 원자성을 갖는 고유한 하나의 속성을 나타냄
- 논리모델의 Attribute를 물리 모델로 매핑한 것

▶ Primary Key

- 각 Row를 유일하게 식별가능 하도록 만드는 1개 컬럼 또는 2개 이상의 컬럼의 조합
- 논리모델의 UID를 물리 모델로 매핑한 것
- 테이블당 1개만 존재 가능

▶ Foreign Key

- 다른 테이블의 primary key를 참조하는 1개 컬럼 또는 2개 이상의 컬럼의 조합
- 테이블당 0개 이상 존재 가능
- Null 이거나 참조하는 primary key의 값과 일치해야 함

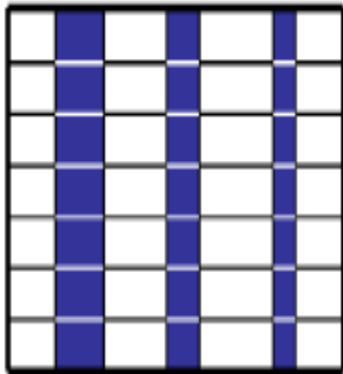
3. 기본적인 SQL - select

- select문 기본
- 모든 열 조회
- 특정 열 조회
- 산술연산자 활용
- 열별칭 사용
- 중복 행 제거

select문 기본(1/2)

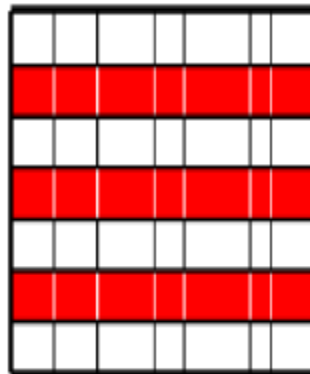
- ▶ 원하는 데이터를 조회하기 위한 SQL

[특정 컬럼 조회]



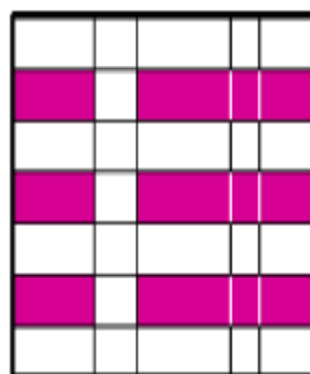
테이블1

[특정 행 조회]



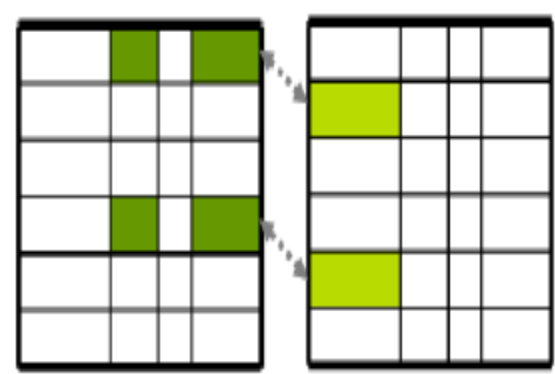
테이블1

[특정 행/컬럼 조회]



테이블1

[여러 테이블의 특정 행/컬럼 조회(조인)]



테이블1

테이블2

select문 기본(2/2)

```
SELECT * | { [DISTINCT] {{column_name | expr } [[AS] [ alias ]}, ...} }  
FROM table_name
```

▶ SELECT 절

- 조회하길 원하는 컬럼명을 나열
 - ▶ 별칭 사용 가능
- * 명시
 - ▶ 해당 테이블의 모든 컬럼명을 의미함

▶ FROM절

- 조회하길 원하는 데이터(컬럼)가 저장된 테이블을 명시함

모든 열 조회

- ▶ SELECT절에 모든 컬럼명을 다 명시하는 방법

```
--SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO  
--FROM EMP;
```

- ▶ SELECT절에 * 을 명시하는 방법

```
--SELECT *  
--FROM EMP;
```

특정 열 조회

- ▶ SELECT절에 조회하고자 하는 컬럼명만 명시함

```
-SELECT EMPNO, ENAME, JOB  
-FROM EMP;  
-
```

산술 연산자 활용

- ▶ 조회하고자 하는 컬럼에 산술 연산을 하여 그 결과를 조회할 수 있음

```
--SELECT ENAME,  
        SAL*12,  
        (SAL*12)+COMM  
--FROM EMP;
```

별칭 사용(1/2)

- ▶ 기본적으로 결과 헤더는 SELECT절에 기술된 내용과 동일하게 됨
- ▶ 산술연산이나 함수 호출 등 복잡한 각 컬럼 조회 구문을 별칭을 사용하여 간단하게 만들 수 있음

```
SELECT * | { [DISTINCT] {{column_name | expr } [[AS] [ alias ]}, ...} }  
FROM table_name
```

- ▶ 별칭을 사용하려는 컬럼명 뒤에 공백으로 구분하고 『AS 별칭』 형태로 기술하거나 『별칭』 형태로 기술함

열별칭 사용(2/2)

- ▶ 별칭을 큰따옴표(“”)로 감싸야 하는 경우
 - 별칭에 특수문자가 포함된 경우
 - 별칭이 숫자이거나 숫자로 시작하는 경우

```
SELECT ENAME AS 이름,  
       SAL*12 AS "1년 급여",  
       (SAL*12)+COMM AS 총소득  
FROM EMP;
```

중복 행 제거

- ▶ 조회하는 모든 컬럼의 조합이 일치하는 행을 제거
- ▶ 컬럼별로 중복 제거는 불가능
- ▶ SELECT절에 1번만 기술

```
SELECT * | { [DISTINCT] {{column_name | expr} {[AS] [ alias ]}, ...} }  
FROM table_name
```

```
--중복 제거하지 않은 경우  
SELECT DEPTNO  
FROM EMP;  
  
--중복 제거한 경우  
SELECT DISTINCT DEPTNO  
FROM EMP;
```

4. 선택적 데이터 조회 및 정렬

- WHERE절
- 비교연산자 활용
- 다른 비교 연산자 - BETWEEN, IN, LIKE
- 논리 연산자 활용
- 연산자 우선순위
- ORDER BY절
- 다중 열 정렬

WHERE절

```
SELECT * | { [DISTINCT] {{column_name | expr } [[AS] [ alias ]}, ...} }  
FROM table_name  
WHERE search condition [ {AND | OR } , search condition ... ] ;
```

- ▶ 결과집합을 filtering하기 위한 것
- ▶ 테이블에 저장된 데이터 중에서 원하는 데이터만 선택적으로 검색하는 기능
- ▶ WHERE 절의 조건문은 컬럼 이름, 연산자, 상수, 산술 표현식을 결합하여 다양한 형태로 표현 가능
- ▶ 문자와 날짜 타입의 상수 값은 작은 따옴표(“)로 묶어서 표현하고 숫자는 그대로 사용
- ▶ 제한조건을 여러 개 포함할 수 있으며, 각 각의 제한 조건은 논리 연산자로 연결

비교연산자 활용(1/2)

- ▶ 표현식 사이의 관계를 비교하기 위해 사용
- ▶ WHERE 절에서 숫자, 문자, 날짜의 크기나 순서를 비교하는 연산자
- ▶ 비교 결과는 TRUE 또는 FALSE 또는 NULL이 됨

연산자	의미
=	같다
!=, <>	같지 않다
>	크다
>=	크거나 같다
<	작다
<=	작거나 같다

비교연산자 활용(2/2)

```
-- SELECT  ENAME AS 이름,  
          DEPTNO AS 부서  
FROM      EMP  
WHERE     DEPTNO = '20';
```

```
-- SELECT  ENAME AS 이름,  
          SAL AS 급여  
FROM      EMP  
WHERE     SAL > 2000;
```

```
-- SELECT  ENAME AS 이름,  
          SAL AS 급여  
FROM      EMP  
WHERE     SAL <= 4000000;
```

다른 비교 연산자 - BETWEEN AND

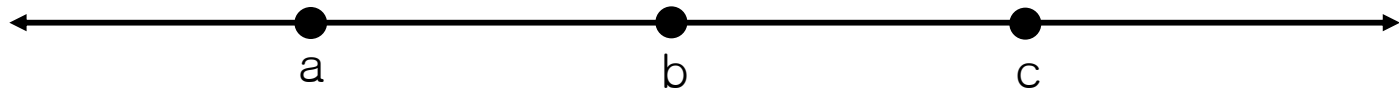
- ▶ 비교하려는 값이 지정한 범위(상한 값 a와 하한 값 b의 경계포함)에 포함되면 TRUE를 반환하는 연산자



```
-- SELECT  ENAME, SAL  
FROM      EMP  
WHERE     SAL BETWEEN 1000 AND 2000;
```

다른 비교 연산자 - IN

- ▶ 비교하려는 값 목록(a,b,c)에 일치하는 값이 있으면 TRUE를 반환하는 연산자



```
-- SELECT ENAME, DEPTNO, SAL  
-- FROM EMP  
-- WHERE DEPTNO IN ('10','20');
```

다른 비교 연산자 - LIKE(1/2)

- ▶ 비교하려는 값이 지정한 특정 패턴을 만족시키면 TRUE를 반환하는 연산자
- ▶ 컬럼에 저장된 문자열 중에서 LIKE 연산자에서 지정한 문자 패턴과 부분적으로 일치하면 TRUE를 반환하는 연산자
- ▶ 패턴을 위해 와일드 카드 사용
 - %
 - ▶ 0개 이상의 임의의 문자를 의미
 - _
 - ▶ 1개의 임의의 문자를 의미

다른 비교 연산자 - LIKE(2/2)

```
-- SELECT ENAME, SAL  
FROM EMP  
WHERE ENAME LIKE 'S%';  
--
```

```
-- SELECT ENAME, SAL  
FROM EMP  
WHERE ENAME LIKE '%S%';  
--
```

```
-- SELECT ENAME  
FROM EMP  
WHERE ENAME LIKE '____S';  
--
```

논리 연산자 활용(1/2)

- 여러 개의 조건 결과를 하나의 논리 결과(TRUE/FALSE/NULL)로 반환하는 연산자

연산자	의미
AND	모든 조건이 참일 때, 참 값을 반환
OR	모든 조건 중에서 하나가 참일 때, 참 값을 반환
NOT	조건과 반대되는 결과를 반환(NULL 제외)

[AND 연산 결과]

	TRUE	FALSE	NULL
TRUE	T	F	N
FALSE	F	F	F
NULL	N	F	N

[OR 연산 결과]

	TRUE	FALSE	NULL
TRUE	T	T	T
FALSE	T	F	N
NULL	T	N	N

논리 연산자 활용(2/2)

```
--SELECT  ENAME AS 이름,  
          DEPTNO AS 부서,  
          SAL AS 급여  
FROM      EMP  
WHERE     DEPTNO = '10'  
AND       SAL > 2000;
```

```
--SELECT  ENAME AS 이름,  
          DEPTNO AS 부서,  
          SAL AS 급여  
FROM      EMP  
WHERE     DEPTNO = '10'  
OR        DEPTNO = '20';
```

```
--SELECT  ENAME AS 이름,  
          DEPTNO AS 부서  
FROM      EMP  
WHERE     NOT DEPTNO = '20';
```

연산자 우선순위

- ▶ 여러 연산자를 함께 사용할 때 우선순위를 고려해야 함
- ▶ () 이용하여 우선순위를 높일 수 있음

1	산술 연산자
2	연결 연산자
3	비교 연산자
4	IS (NOT) NULL, LIKE, (NOT) IN
5	(NOT) BETWEEN-AND
6	논리 연산자 - NOT
7	논리 연산자 - AND
8	논리 연산자 - OR

ORDER BY절(1/2)

```
SELECT ...  
FROM ...  
WHERE ...  
ORDER BY 기준1 [ ASC | DESC ] [, 기준2 [ASC | DESC], ... ];
```

- ▶ SELECT구문 실행 결과를 특정 컬럼 값 기준으로 정렬할 때 사용
- ▶ SELECT구문의 마지막에 위치
- ▶ 정렬조건
 - ASC : 오름차순(기본값)
 - DESC : 내림차순
- ▶ 여러 개의 정렬기준을 사용하면 기술된 순서대로 적용
- ▶ 컬럼이름이나 컬럼별칭 이나 컬럼기술순서로 표현 가능

ORDER BY절(2/2)

```
SELECT  ENAME, SAL  
FROM    EMP  
ORDER BY ENAME;
```

```
--  
SELECT  ENAME, SAL  
FROM    EMP  
WHERE   DEPTNO = '10'  
OR      DEPTNO IS NULL  
ORDER BY SAL DESC;  
--
```

다중 열 정렬

```
-- SELECT      ENAME, HIREDATE, DEPTNO
-- FROM        EMP
-- WHERE        HIREDATE > TO_DATE('19820101', 'YYYYMMDD')
-- ORDER BY    DEPTNO DESC, HIREDATE, ENAME;
```

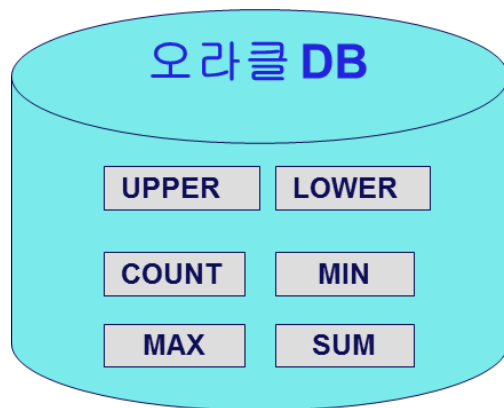
```
-- SELECT      ENAME AS 이름,
--              HIREDATE AS 입사일,
--              DEPTNO AS 부서코드
-- FROM        EMP
-- WHERE        HIREDATE > TO_DATE('19820101', 'YYYYMMDD')
-- ORDER BY    부서코드 DESC, 입사일, 이름;
```

5. 단일 행 함수

- SQL 함수
- 단일 행 함수
- 문자관련 함수
- 숫자관련 함수
- 날짜관련 함수

SQL함수(1/2)

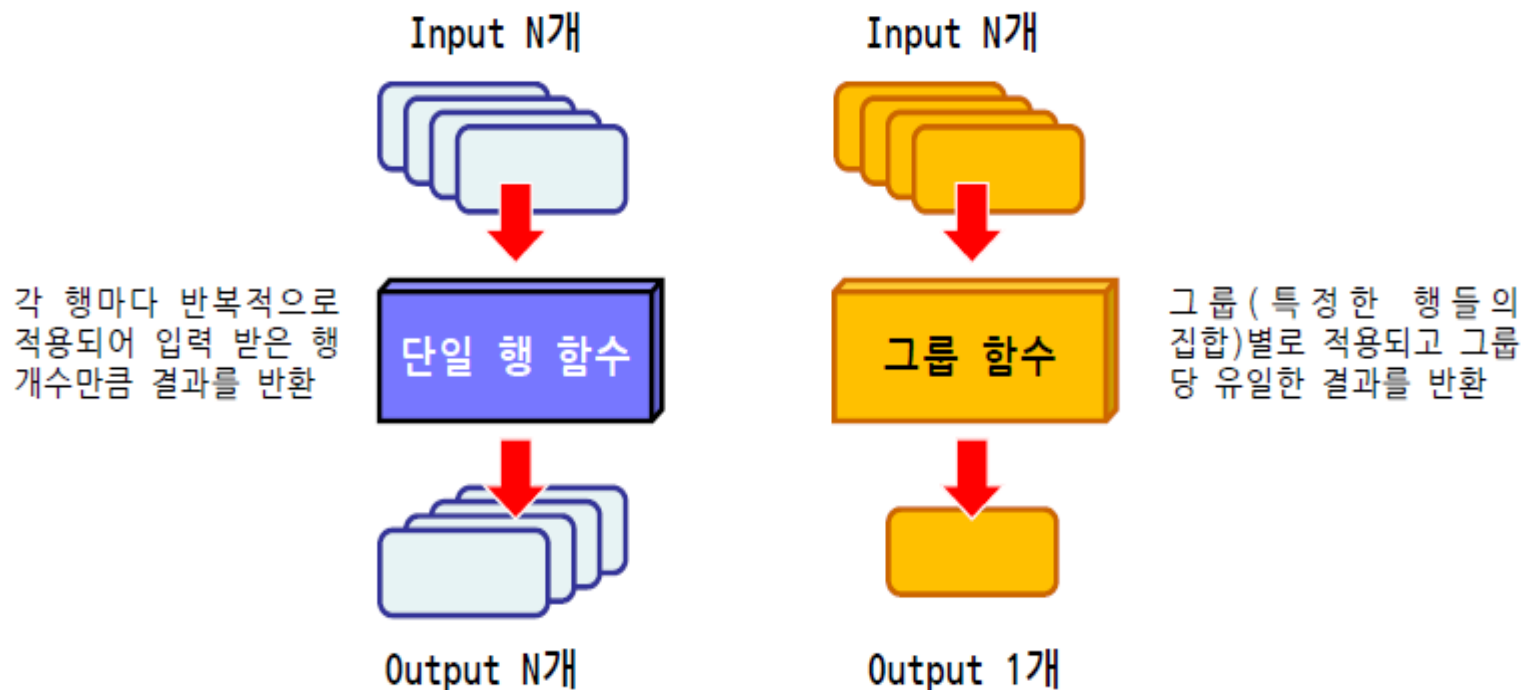
- ▶ 반복적으로 사용되어 질 수 있는 부분들을 분리하여 DB에 미리 만들어 놓은 작은 서브프로그램으로 볼 수 있음
- ▶ INPUT(Parameter), OUTPUT(Return Value)이 존재
- ▶ 기본적인 Query문을 더욱 강력하고 편리하게 사용하도록 도와줌
- ▶ 질의가 쉬워지고 응용프로그램의 코딩을 줄여줄 수 있음



```
SQL> SELECT count(*)  
        FROM emp;  
  
COUNT(*)  
-----  
14
```

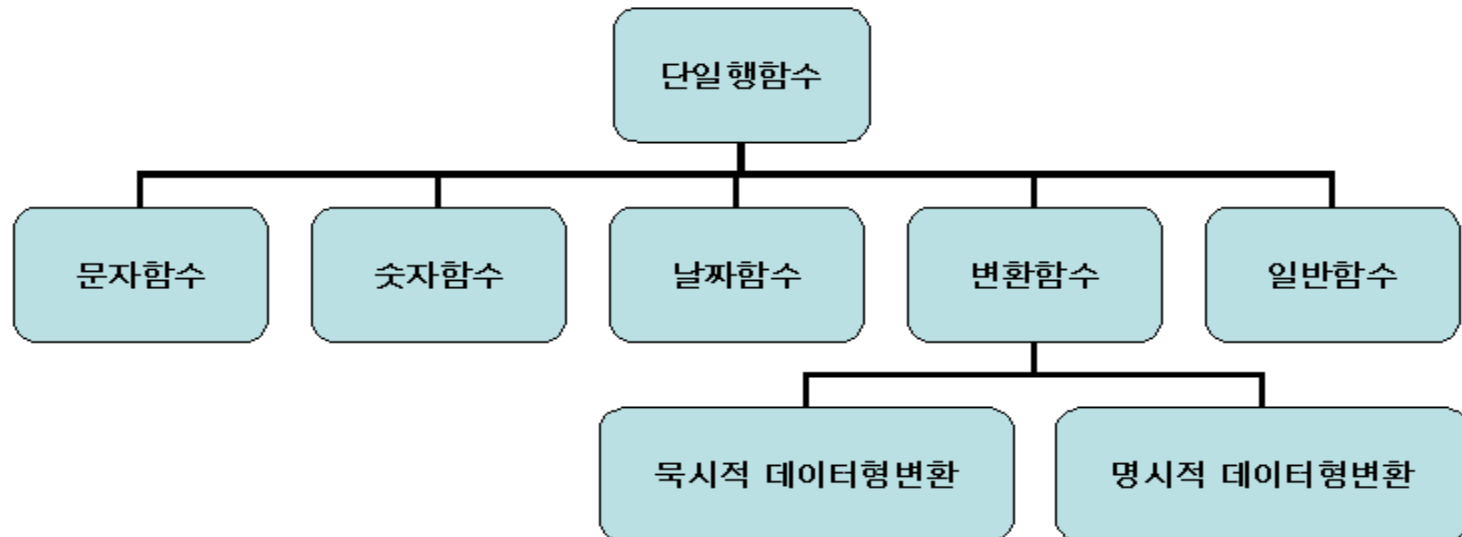
SQL 함수(2/2)

- ▶ 반환 결과에 따라 단일 행 함수와 그룹함수로 구분



단일 행 함수 - 개요

- ▶ 한 행 당 하나의 결과 값을 반환하는 함수
- ▶ 테이블에 저장되어 있는 개별 행을 대상으로 함수를 적용하여 하나의 결과를 반환 함
- ▶ 데이터 값을 조작하는데 주로 사용



단일 행 함수 - 함수 사용법

```
function_name (column | expression, [arg1, arg2, . . .])
```

- ▶ function_name : 단일 행 함수 이름
- ▶ column : 칼럼 이름
- ▶ expression : 문자열 또는 표현식
- ▶ arg1, arg2 : 함수의 인수(상수, 변수, 칼럼 이름, 표현식)

문자관련 함수 - 개요

- ▶ 문자 데이터를 입력하여 문자나 숫자를 결과로 반환하는 함수
- ▶ 문자 함수의 종류
 - 대소문자 변환 함수
 - 문자조작 함수
 - 문자열 길이 반환 함수

문자관련 함수 - 대소문자 변환 함수

종류	의미	사용 예
INITCAP	문자열의 첫 번째 문자만 대문자로 변환	INITCAP('student') → Student
LOWER	문자열 전체를 소문자로 변환	LOWER('STUDENT') → student
UPPER	문자열 전체를 대문자로 변환	UPPER('student') → STUDENT

문자관련 함수 - 문자 조작 함수

종류	의미	사용 예
CONCAT	두 문자열을 결합, ' '와 동일	CONCAT('sql','plus') → sqlplus
SUBSTR	특정 문자 또는 문자열 일부를 추출	SUBSTR('SQL*Plus',5,4) → Plus
INSTR	특정 문자가 출현하는 첫 번째 위치를 반환	INSTR('SQL*Plus','*') → 4
LPAD	오른쪽 정렬후 왼쪽으로 지정문자를 삽입	LPAD('sql', 5, '*') → **sql
RPAD	왼쪽 정렬후 오른쪽으로 지정 문자를 삽입	RPAD('sql', 5, '*') → sql**
LTRIM	왼쪽 지정 문자를 삭제	LTRIM('*sql', '*') → sql
RTRIM	오른쪽 지정 문자를 삭제	RTRIM('sql*', '*') → sql

문자관련 함수 - 문자열 길이 반환 함수

종류	의미	사용 예
LENGTH	문자열의 길이를 반환	LENGTH('홍길동') → 3
LENGTHB	문자의 바이트 수를 반환	LENGTHB('홍길동') → 6

숫자 관련 함수

▶ 숫자 데이터를 처리하기 위한 함수

종류	의미	사용 예
ROUND	지정한 자리 이하에서 반올림	ROUND(123.17,1) → 123.2
TRUNC	지정한 자리 이하에서 절삭	TRUNC(123.17,1) → 123.1
MOD	m을n으로 나눈 나머지	MOD(12,10) → 2
CEIL	지정한 값보다 큰수 중에서 가장 작은 정수	CEIL(123.17) → 124
FLOOR	지정한 값보다 작은수 중에서 가장 큰 정수	FLOOR(123.17) → 123

날짜관련 함수(1/2)

▶ 날짜 데이터를 처리하기 위한 함수

종류	의미	결과
SYSDATE	시스템의 현재 날짜	날짜
MONTHS_BETWEEN	날짜와 날짜 사이의 개월을 계산	숫자
ADD_MONTHS	날짜에 개월을 더한 날짜 계산	날짜
NEXT_DAY	날짜후의 첫 요일의 날짜를 계산	날짜
LAST_DAY	월의 마지막 날짜를 계산	날짜
ROUND	날짜를 반올림	날짜
TRUNC	날짜를 절삭	날짜

날짜관련 함수(2/2)

- ▶ 함수를 이용하지 않고 산술 연산자를 이용하여 날짜 계산도 가능

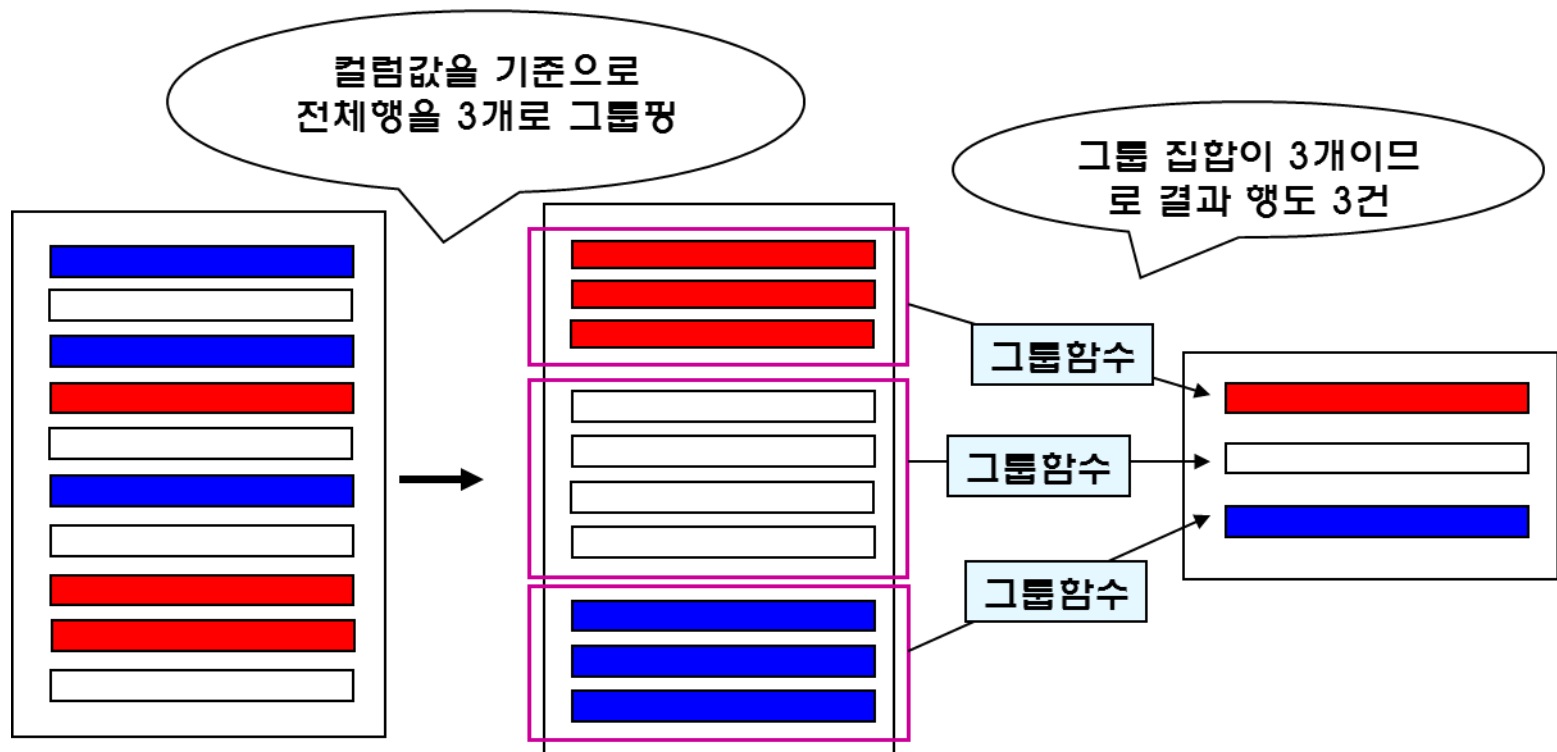
종류	결과	의미
날짜 + 숫자	날짜	날짜에 일수를 가산하여 날짜 계산
날짜 - 숫자	날짜	날짜에 일수를 감산하여 날짜를 계산
날짜 - 날짜	일수	날짜와 날짜를 감산하여 일수를 계산
날짜 + 숫자/24	날짜	날짜에 시간을 가산하여 날짜 계산

7. 그룹함수를 사용한 데이터 조회

- 그룹함수란
- 그룹함수의 종류
- GROUP BY절
- HAVING절

그룹함수란

- ▶ 테이블의 전체 행을 하나 이상의 컬럼을 기준으로 그룹화하여 그룹별로 결과를 출력하는 함수
- ▶ 그룹함수는 통계적인 결과를 출력하는데 자주 사용



그룹함수의 종류

- ▶ 그룹함수는 NULL을 계산하지 않음에 주의할 것

종류	의미
COUNT	행의 개수 출력
MAX	NULL을 제외한 모든 행에서 최대 값
MIN	NULL을 제외한 모든 행에서 최소 값
SUM	NULL을 제외한 모든 행의 합
AVG	NULL을 제외한 모든 행의 평균 값
STDDEV	NULL을 제외한 모든 행의 표준편차
VARIANCE	NULL을 제외한 모든 행의 분산 값
GROUPING	해당 칼럼이 그룹에 사용되었는지 여부를 1 또는 0으로 반환
GROUPING SETS	한 번의 질의로 여러 개의 그룹화 기능

GROUP BY절(1/4)

- ▶ 결과집합의 하위 데이터 그룹을 만듦
- ▶ 특정 칼럼 값을 기준으로 테이블의 전체 행을 그룹별로 나눔
- ▶ GROUP BY절에 기술한 컬럼이나 표현식을 기준으로 데이터그룹 생성

```
SELECT ...  
FROM ...  
WHERE ...  
GROUP BY column_name | expr  
ORDER BY 기준1 [ASC | DESC] [, 기준2 [ASC | DESC], ... ];
```

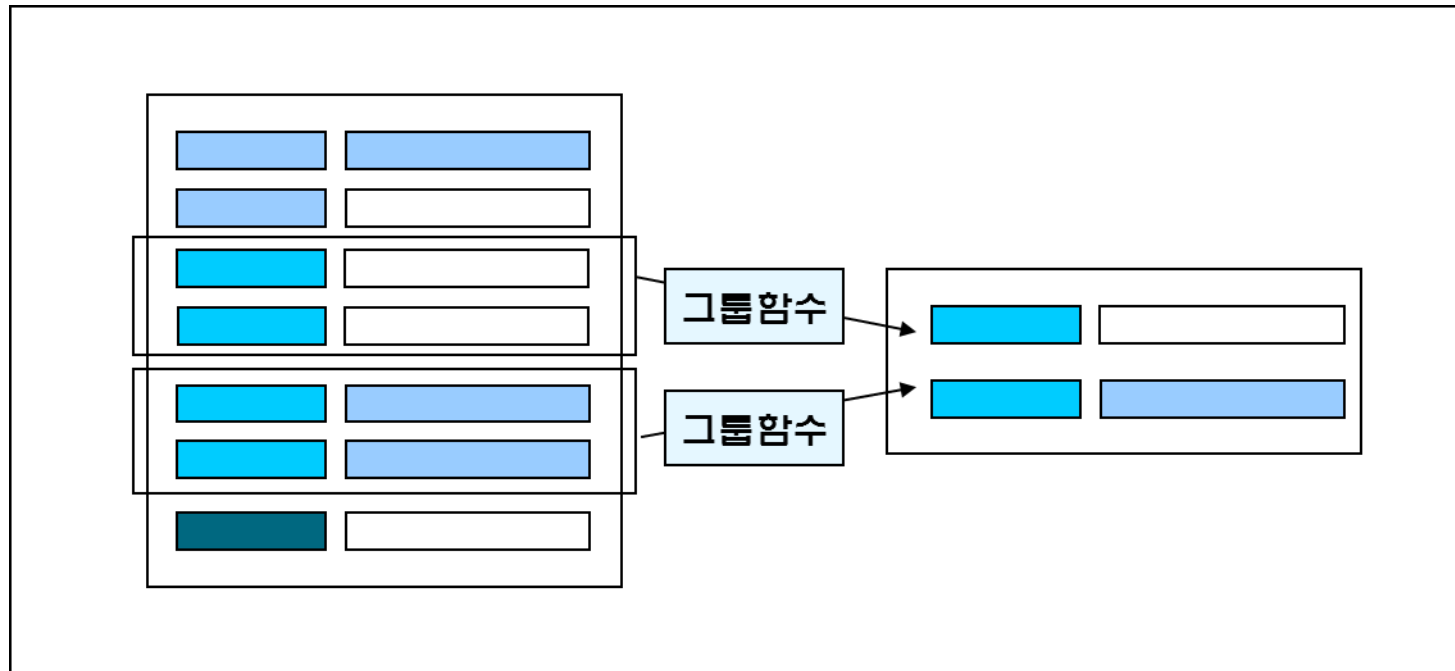
GROUP BY절(2/4)

- ▶ 각 그룹별로 SELECT절에 기술한 그룹함수가 적용
- ▶ SELECT절에 기술한 컬럼 중, 그룹함수에 사용되지 않은 컬럼은 GROUP BY절에 반드시 기술되어야 함
- ▶ WHERE절에는 그룹함수를 사용할 수 없음
 - Grouping 이전에 WHERE 절을 사용하여 그룹 대상 집합을 먼저 선택
- ▶ GROUP BY절에는 컬럼 이름만 사용가능
 - 별칭, 순서 사용불가

GROUP BY절(3/4)

▶ 다중 칼럼을 이용한 Grouping

- 하나 이상의 칼럼을 사용하여 그룹을 나누고, 그룹별로 다시 서브 그룹을 나눔



GROUP BY절(4/4)

```
- SELECT      DEPTNO, COUNT(*)  
FROM        EMP  
GROUP BY    DEPTNO  
ORDER BY    1;  
-
```

```
- SELECT      DEPTNO, SUM(SAL)  
FROM        EMP  
GROUP BY    DEPTNO;  
-
```

```
- SELECT      MAX(SUM(SAL))  
FROM        EMP  
GROUP BY    DEPTNO;  
-
```

```
- SELECT      ENAME, DEPTNO, COUNT(*)  
FROM        EMP  
GROUP BY    ENAME, DEPTNO;  
-
```

HAVING절(1/2)

- ▶ GROUP BY 절에 의해 생성된 그룹을 대상으로 조건을 적용하여 Filtering하기 위해 사용
- ▶ HAVING 절의 실행 과정
 - 테이블에서 WHERE 절에 의해 조건을 만족하는 행 집합을 선택
 - 행 집합을 GROUP BY 절에 의해 Grouping
 - HAVING 절에 의해 조건을 만족하는 그룹을 선택

WHERE절을 만족하는 행 선택

GROUP BY절에 의한 그룹핑

HAVING절을 만족하는 그룹 선택

HAVING절(2/2)

```
SELECT    ...  
FROM      ...  
WHERE     ...  
GROUP BY  column name | expr  
HAVING   condition  
ORDER BY  기준1 [ ASC | DESC ] [ , 기준2 [ASC | DESC], ... ];
```

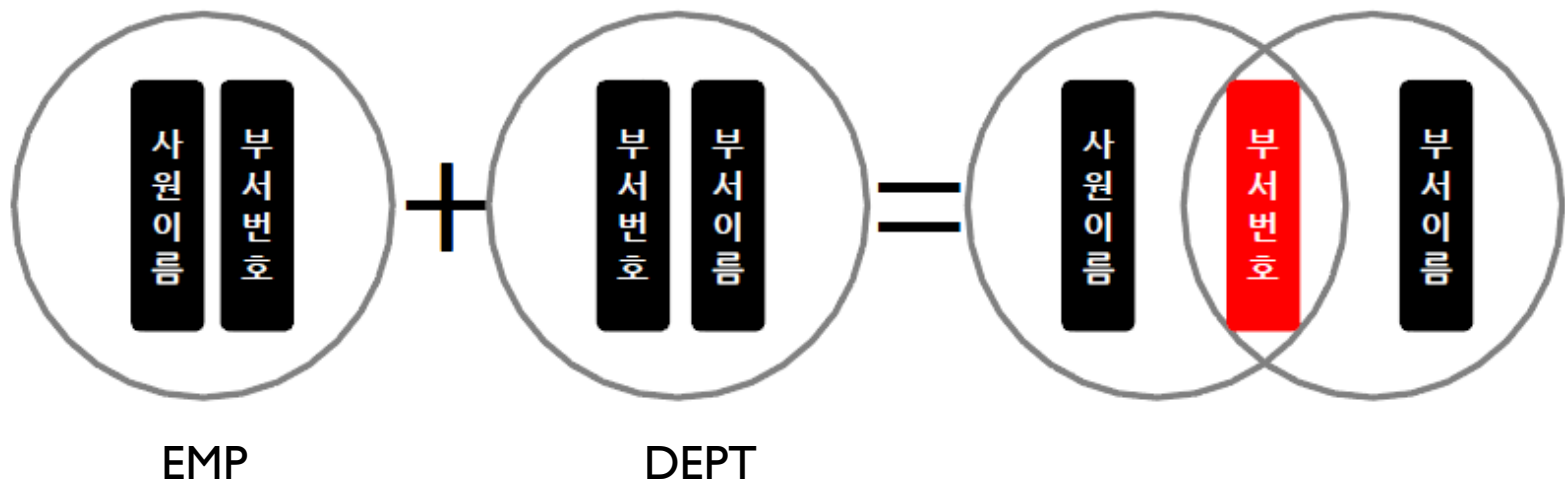
```
- SELECT      DEPTNO, SUM(SAL)  
FROM          EMP  
GROUP BY      DEPTNO  
HAVING        SUM(SAL) > 9000;  
-
```

6. 여러 테이블에서의 데이터 조회

- JOIN 이란
- JOIN의 유형
- EQUI JOIN
- NONEQUI JOIN
- OUTER JOIN
- FULL OUTER JOIN
- SELF JOIN

JOIN 이란

- ▶ 하나의 SQL 명령문으로 여러 테이블에 저장된 데이터를 한번에 조회할 수 있는 기능
- ▶ 관계형 데이터베이스 분야의 표준
- ▶ 두 개 이상의 테이블을 '결합' 한다는 의미
- ▶ 주로 Primary-Key와 Foreign-Key의 관계를 가진 컬럼을 소유하고 있는 테이블을 통한 검색 시 사용



JOIN의 유형

▶ EQUI JOIN

- 조인 대상 테이블에서 공통 칼럼을 '='(equal) 비교를 통해 같은 값을 가지는 행을 연결하여 결과를 생성하는 조인 방법

▶ NONEQUI JOIN

- <, BETWEEN a AND b 와 같이 '=' 조건이 아닌 연산자를 사용하여 조인하는 방법

▶ OUTER JOIN

- EQUI JOIN에서 양측 칼럼 값 중의 하나가 NULL 이지만 결과에 누락되지 않도록 포함하기 위하여 조인하는 방법

▶ SELF JOIN

- 하나의 테이블 내에 있는 컬럼끼리 연결하여 결과를 얻고자 할 때 조인하는 방법

EQUI JOIN - 개요

- ▶ 관련 테이블에서 공통 컬럼을 '='(equal) 비교를 통해 같은 값을 가지는 행을 연결하여 결과를 생성하는 조인 방법
- ▶ SQL 명령문에서 가장 많이 사용하는 조인 방법

EQUI JOIN – 오라클(1/2)

```
SELECT table1.column, table2.column  
FROM   table1, table2  
WHERE  table1.column = table2.column;
```

▶ FROM절

- 조인대상이 되는 관련 테이블들을 기술
- 콤마로 구분
- 테이블 별칭 사용 가능
 - ▶ 관련 테이블에서 컬럼명이 같을 경우는 반드시 테이블 별칭 사용해야 함

▶ WHERE절

- 조인 조건 기술
- 공통 컬럼을 '=' 연산자를 이용하여 조건을 기술

EQUI JOIN - 오라클(2/2)

```
-- error(양테이블의 컬럼명이 동일한 경우 테이블 별칭으로 구분하지 않을 경우 에러)
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE DEPTNO = DEPTNO;
```

```
SELECT ENAME, DNAME
FROM EMP e, DEPT d
WHERE e.DEPTNO = d.DEPTNO;
```


EQUI JOIN - ANSI 표준(1/2)

▶ JOIN ~ USING

- 관련 테이블에서 컬럼명이 같을 경우 사용

```
SELECT table1.column, table2.column  
FROM   table1  
JOIN    table2 USING (column);
```

▶ JOIN ON

- 관련 테이블에서 컬럼명이 다를 경우 사용
- 관련 테이블의 별칭을 이용할 경우 사용

```
SELECT table1.column, table2.column  
FROM   table1  
JOIN    table2 ON ( 조건식1 [AND 조건식2 ...] );
```

EQUI JOIN - ANSI 표준(2/2)

▶ JOIN ~ USING

```
--SELECT ENAME, DNAME  
FROM   EMP  
JOIN    DEPT USING (DEPTNO);
```

▶ JOIN ~ ON

```
--SELECT ENAME, DNAME  
FROM   EMP E  
JOIN    DEPT D ON (E.DEPTNO = D.DEPTNO);
```

NONEQUI JOIN

- ▶ 관련 테이블에서 공통 컬럼을 '='(equal) 비교연산자가 아닌 다른 연산자(<, BETWEEN ~ AND, ...)를 통해 행을 연결하여 결과를 생성하는 조인 방법
- ▶ 오라클

```
SELECT  E.ENAME, E.SAL, S.GRADE
FROM    EMP E, SALGRADE S
WHERE   E.SAL BETWEEN S.LOSAL AND S.HISAL
ORDER BY 3;
```

- ▶ ANSI

```
SELECT  ENAME, SAL, GRADE
FROM    EMP
JOIN    SALGRADE ON (SAL BETWEEN LOSAL AND HISAL)
ORDER BY 3;
```

OUTER JOIN - 개요

- ▶ EQUI JOIN의 조인 조건에서 양측 칼럼 값 중, 어느 하나라도 NULL 이면 '=' 비교 결과가 거짓이 되어 NULL 값을 가진 행은 조인 결과로 출력 불가
 - NULL에 대해서 어떠한 연산을 적용하더라도 연산 결과는 NULL임
- ▶ EQUI JOIN에서 양측 칼럼 값중의 하나가 NULL 이지만 조인 결과로 출력할 필요가 있는 경우 OUTER JOIN 사용
- ▶ 예
 - 직원테이블과 부서 테이블 조인시 소속 부서가 없는 직원도 함께 조회

OUTER JOIN - 오라클(1/2)

- ▶ WHERE 절의 조인 조건에서 OUTER JOIN 연산자인 '(+)' 기호 사용
- ▶ 조인 조건문에서 NULL이 출력되는 테이블의 칼럼에 (+) 기호 추가

```
SELECT      table.1column, table2.column  
FROM        table1, table2  
WHERE       table1.column(+) = table2.column;
```

```
SELECT      table.1column, table2.column  
FROM        table1, table2  
WHERE       table1.column = table2.column(+);
```

OUTER JOIN - 오라클(2/2)

-- 부서가 없는 직원도 포함

```
SELECT  E.ENAME, D.DEPTNO
FROM    EMP E, DEPT D
WHERE   E.DEPTNO = D.DEPTNO(+);
```

-- 소속 직원이 없는 모든 부서를 포함

```
SELECT  ENAME, D.DEPTNO
FROM    EMP e, DEPT d
WHERE   E.DEPTNO(+) = D.DEPTNO;
```

OUTER JOIN - ANSI(1/2)

- ▶ 전체 행을 모두 포함시켜야 하는 테이블을 기준으로 LEFT, RIGHT 키워드 사용
 - LEFT
 - ▶ 기준테이블이 JOIN 키워드보다 먼저 기술된 경우
 - RIGHT
 - ▶ 기준테이블이 JOIN 키워드보다 나중에 기술된 경우

```
SELECT      table1.column, table2.column  
FROM        table1  
LEFT JOIN   table2 USING ( column );
```

```
SELECT      table1.column, table2.column  
FROM        table2  
RIGHT JOIN  table1 USING ( column );
```

OUTER JOIN - ANSI(2/2)

```
-- SELECT      ENAME, DNAME
-- FROM        DEPT
-- LEFT JOIN   EMP USING(DEPTNO)
-- ORDER BY   1;
```

```
-- SELECT      ENAME, DNAME
-- FROM        EMP
-- RIGHT JOIN  DEPT USING(DEPTNO)
-- ORDER BY   1;
```


FULL OUTER JOIN

- ▶ LEFT OUTER JOIN 과 RIGHT OUTER JOIN 을 동시에 실행한 결과를 출력
- ▶ 오라클은 지원하지 않음

```
SELECT      table1.column, table2.column  
FROM        table1  
FULL JOIN   table2 USING ( column );
```

```
-- 소속 직원이 없는 모든 부서 및 소속 부서가 없는 직원도 포함  
SELECT      ENAME, DNAME  
FROM        EMP  
FULL JOIN   DEPT USING(DEPTNO);
```

SELF JOIN(1/2)

- ▶ 한 테이블을 두 번 조인하는 유형
- ▶ 한 테이블 내에 있는 칼럼끼리 연결하는 조인이 필요한 경우 사용
- ▶ 테이블 별칭을 사용해야 함

SELF JOIN(2/2)

▶ 오라클

```
SELECT      E.ENAME AS 직원,  
            M.ENAME AS 관리자  
FROM        EMP e, EMP m  
WHERE       E.MGR = M.EMPNO  
ORDER BY    1;
```

▶ ANSI

```
-- SELECT      E.ENAME AS 직원,  
--            M.ENAME AS 관리자  
FROM          EMP E  
JOIN          EMP M ON(E.MGR = M.EMPNO)  
ORDER BY      1;
```

8. Sub Query

- 그룹함수란
- 그룹함수의 종류
- GROUP BY절
- HAVING절

Sub Query - 개요

- ▶ 하나의 쿼리가 다른 쿼리에 포함되는 구조
- ▶ 하나의 SQL 명령문의 결과를 다른 SQL 명령문에 전달하기 위해 두 개 이상의 SQL 명령문을 하나의 SQL 명령문으로 연결하여 처리하는 방법
- ▶ 다른 쿼리에 포함된 내부 쿼리(서브 쿼리)는 외부 쿼리(메인 쿼리)에 사용될 값을 반환하는 역할
- ▶ SELECT, FROM, WHERE, HAVING 절 등에서 사용 가능

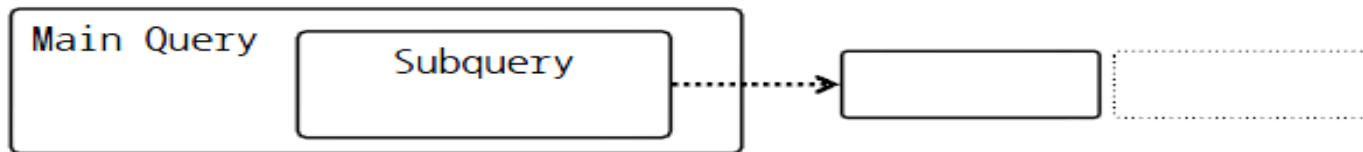
Sub Query - 구문

```
SELECT ...  
FROM ...  
WHERE expr operator ( SELECT ...  
                        FROM ...  
                        WHERE ... ) ;
```

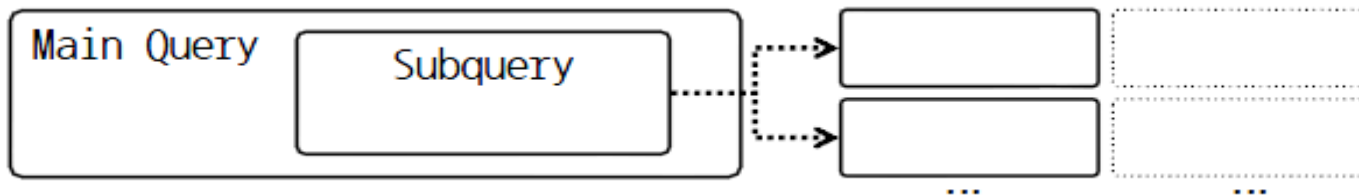
- ▶ 서브쿼리는()로 묶어서 표현
- ▶ 서브쿼리에는 ;를 사용하지 않음
- ▶ 유형에 따라 연산자를 구분해서 사용
- ▶ 처리과정
 - 1. 서브 쿼리는 메인 쿼리가 실행되기 전에 한번씩 실행됨
 - 2. 서브 쿼리에서 실행된 결과가 메인 쿼리에 전달되어 최종적인 결과를 출력

Sub Query의 유형 - 개요

- ▶ 반환되는 행의 수에 따라 나뉨
- ▶ 단일 행 서브 쿼리
 - 서브 쿼리에서 하나의 행만을 검색하여 메인 쿼리에 반환하는 질의문
 - 단일 행 비교연산자(=,>,>=,<,<=,<>등)만 사용가능



- ▶ 다중 행 서브 쿼리
 - 서브 쿼리에서 여러 행을 검색하여 메인 쿼리에 반환하는 질의문
 - 다중 행 비교연산자(IN,ANY,ALL등) 사용



단일 행 서브 쿼리

-- 이름이 'SMITH'인 직원과 같은 부서에 근무하는 직원 조회

```
SELECT ENAME, DEPTNO
FROM EMP
WHERE DEPTNO = (
    SELECT DEPTNO
    FROM EMP
    WHERE ENAME = 'SMITH'
);
```

-- 이름이 'SMITH'인 직원의 급여보다 많은 급여를 받는 직원 조회

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > (
    SELECT SAL
    FROM EMP
    WHERE ENAME = 'SMITH'
);
```


다중 행 서브 쿼리 - 개요

▶ 다중 행 비교 연산자 이용

종류	의 미
IN	메인 쿼리의 비교 조건이 서브쿼리의 결과 중에서 하나라도 일치하면 참, '='비교만 가능
ANY	메인 쿼리의 비교 조건이 서브쿼리의 결과 중에서 하나라도 일치하면 참, '='비교만 가능
ALL	메인 쿼리의 비교 조건이 서브쿼리의 결과 중에서 모든 값이 일치하면 참,
EXISTS	메인 쿼리의 비교 조건이 서브쿼리의 결과 중에서 만족하는 값이 하나라도 존재하면 참

다중 행 서브 쿼리 - IN, NOT IN 연산자

-- 관리자인 직원 조회

```
SELECT EMPNO, ENAME, '관리자' AS 구분
FROM EMP
WHERE EMPNO IN (
    SELECT DISTINCT MGR
    FROM EMP
    WHERE MGR IS NOT NULL
);
```

-- 관리자가 아닌 직원 조회

```
SELECT EMPNO, ENAME, '직원' AS 구분
FROM EMP
WHERE EMPNO NOT IN (
    SELECT DISTINCT MGR
    FROM EMP
    WHERE MGR IS NOT NULL
);
```

다중 행 서브 쿼리 - ANY 연산자(1/2)

▶ < ANY

- 비교대상 중 최대값보다 작음

```
--SALESMAN의 최대급여보다 급여가 적은 직원조회
SELECT ENAME, SAL
FROM EMP
WHERE SAL < ANY (
    SELECT SAL
    FROM EMP
    WHERE JOB = 'SALESMAN'
);
```

다중 행 서브 쿼리 - ANY 연산자(2/2)

▶ > ANY

- 비교대상 중 최소값보다 큼

```
--SALESMAN의 최저급여보다 급여가 많은 직원 조회
SELECT ENAME, SAL
FROM EMP
WHERE SAL > ANY (
    SELECT SAL
    FROM EMP
    WHERE JOB = 'SALESMAN'
);
```

▶ = ANY

- IN연산자와 동일

다중 행 서브 쿼리 - ALL 연산자(1/2)

▶ < ALL

- 비교대상 중 최소값보다 작음

```
--SALESMAN의 최저급여보다 급여가 적은 직원조회
SELECT ENAME, SAL
FROM EMP
WHERE SAL < ALL (
    SELECT SAL
    FROM EMP
    WHERE JOB = 'SALESMAN'
);
```

다중 행 서브 쿼리 - ALL 연산자(2/2)

▶ > ALL

- 비교대상 중 최대값보다 큼

```
--SALESMAN의 최대급여보다 급여가 많은 직원조회
SELECT ENAME, SAL
FROM EMP
WHERE SAL > ALL (
    SELECT SAL
    FROM EMP
    WHERE JOB = 'SALESMAN'
);
```

다중 행 서브 쿼리 - EXISTS , NOT EXISTS 연산자

- ▶ 서브쿼리에서 검색된 결과가 하나라도 존재하면 메인 쿼리 조건절이 참이 되는 연산자

```
-- 관리자인 직원 조회
SELECT EMPNO, ENAME, '관리자' AS 구분
FROM EMP E
WHERE EXISTS (
    SELECT EMPNO
    FROM EMP
    WHERE E.EMPNO = MGR
);
```

```
-- 관리자가 아닌 직원 조회
SELECT EMPNO, ENAME, '직원' AS 구분
FROM EMP E
WHERE NOT EXISTS(
    SELECT EMPNO
    FROM EMP
    WHERE E.EMPNO = MGR
);
```

9. 데이터 조작

- DML
- INSERT
- UPDATE
- DELETE
- 트랜잭션이란
- COMMIT & ROLLBACK

DML

- ▶ Data Manipulation Language
- ▶ 테이블에 새로운 데이터를 입력하거나 기존 데이터를 수정 또는 삭제하기 위한 명령어

- ▶ 종류
 - INSERT :
 - ▶ 한 개 또는 여러 개의 새로운 데이터 삽입 명령어
 - UPDATE
 - ▶ 한 개 또는 여러 개의 기존 데이터 수정 명령어
 - DELETE
 - ▶ 한 개 또는 여러 개의 기존 데이터 삭제 명령어

INSERT - 개요

- ▶ 테이블에 데이터를 입력하기 위한 조작어
- ▶ 데이터 입력 방법
 - 단일 행 입력
 - ▶ 한번에 하나의 행을 테이블에 입력하는 방법
 - 다중 행 입력
 - ▶ 서브쿼리를 이용하여 한번에 다른 테이블의 여러 행을 조회하여 동시에 입력하는 방법

INSERT - 단일 행 입력(1/3)

```
INSERT INTO table [(column [, column...])]  
VALUES (value [, value...]);
```

- ▶ INTO 절에 명시한 컬럼에 VALUES 절에서 지정한 칼럼 값을 입력
- ▶ INTO 절에 칼럼을 명시하지 않으면 테이블 생성시 정의한 칼럼 순서와 동일한 순서로 입력
- ▶ 입력되는 데이터 타입은 칼럼의 데이터 타입과 동일해야 하고 입력되는 데이터의 크기는 칼럼의 크기보다 작거나 동일해야 함
- ▶ CHAR, VARCHAR2, DATE 타입의 입력 데이터는 단일인용 부호(")로 묶어서 입력

INSERT - 단일 행 입력(2/3)

▶ NULL 입력

- 암시적 방법
 - ▶ INSERT INTO절에서 해당 컬럼이름 생략
- 명시적 방법
 - ▶ VALUES절에서 **NULL** 키워드나 “사용

▶ DEFAULT

- 암시적 방법
 - ▶ INSERT INTO절에서 해당 컬럼이름 생략
- 명시적 방법
 - ▶ VALUES절에서 **DEFAULT** 키워드 사용

INSERT - 단일 행 입력(3/3)

```
-- INSERT INTO EMP(EMPNO, ENAME, JOB, SAL, DEPTNO)  
-- VALUES (8000, 'JANE', 'OPERTOR' , 1000, 40);
```

```
-- INSERT INTO EMP  
-- VALUES (800, 'JANE', 'OPERTOR' , 7782, '2011-05-01', 1000, NULL, 40);
```

INSERT - 다중 행 입력

INSERT INTO table [(column [, column...])]
SUBQUERY내용 ;

- ▶ INSERT 명령문에서 서브쿼리 절을 이용하여 자신이나 다른 테이블에 데이터를 복사하여 여러 행 동시 입력
- ▶ INSERT 명령문의 VALUES 절 대신 서브쿼리에서 검색된 결과 집합을 한꺼번에 입력
- ▶ 서브쿼리의 결과 집합은 INSERT 명령문에 지정된 칼럼 개수와 데이터 타입이 일치해야 함

```
INSERT INTO EMP2  
(SELECT EMPNO, ENAME, DEPTNO  
FROM EMP  
WHERE DEPTNO = 10);
```

UPDATE - 개요

- ▶ 테이블에 포함된 기존 데이터를 수정하기 위한 조작용어
- ▶ 전체 레코드 수 (행수)는 달라지지 않음
- ▶ WHERE절 생략 시 테이블의 모든 행을 수정

- ▶ 데이터 수정 방법
 - 직접 값을 수정
 - 서브쿼리를 이용하여 수정

UPDATE - 직접 값 수정

UPDATE table
SET column=value [, column=value, ...]
[WHERE condition];

```
UPDATE EMP  
SET JOB = 'MANAGER'  
WHERE EMPNO=8000;
```


UPDATE - 서브쿼리를 이용하여 수정

```
UPDATE table1
SET (column1, column2, ...) = ( SELECT s_column1, s_column2, ...
                                FROM table2
                                [WHERE condition2] )
[WHERE condition1];
```

- ▶ 특정 테이블에 저장된 데이터 검색하여 한꺼번에 여러 컬럼 수정
- ▶ SET 절의 컬럼 이름은 서브쿼리의 컬럼 이름과 달라도 됨
- ▶ 데이터 타입과 컬럼 수는 반드시 일치

```
UPDATE EMP
SET SAL = (
            SELECT AVG(SAL)
            FROM EMP
        )
WHERE EMPNO=8000;
```

DELETE - 개요

- ▶ 테이블에 저장된 데이터를 삭제하기 위한 조작용어
- ▶ 전체 레코드 수 (행수)는 달라짐
- ▶ WHERE절 생략 시 테이블의 모든 행을 삭제

- ▶ 데이터 삭제 방법
 - 조건을 직접 지정
 - 조건을 서브쿼리를 이용하여 지정

DELETE - 직접 조건 지정

```
DELETE [FROM] table  
[WHERE condition1];
```

```
DELETE FROM EMP  
WHERE DEPTNO = 40;
```

DELETE - SUBQUERY를 이용한 조건 지정(1/2)

```
DELETE FROM table
WHERE (column1, column2, ...)=( SELECT s_column1, s_column2, ...
                                FROM table2
                                [WHERE condition2] );
```

- ▶ 다른 테이블에 저장된 데이터를 검색하여 조건을 지정하여 삭제 함
- ▶ WHERE 절의 칼럼 이름은 서브쿼리의 칼럼 이름과 달라도 됨
- ▶ 데이터 타입과 칼럼 수는 일치

DELETE - SUBQUERY를 이용한 조건 지정(2/2)

```
DELETE FROM EMP  
WHERE DEPTNO = (  
    SELECT DEPTNO  
    FROM DEPT  
    WHERE DNAME='OPERATIONS'  
);
```

트랜잭션이란

- ▶ 여러 개의 명령문을 하나의 논리적인 작업단위로 처리하는 기능
- ▶ All or Nothing
- ▶ 트랜잭션 관리 명령어
 - COMMIT
 - ▶ 트랜잭션의 정상적인 종료를 위한 명령어
 - ROLLBACK
 - ▶ 트랜잭션의 비정상적인 종단을 위한 명령어

명령문	의미
COMMIT	트랜잭션내의 모든 SQL 명령문에 의해 변경된 작업 내용을 디스크에 영구적으로 저장하고 트랜잭션을 종료
ROLLBACK	트랜잭션내의 모든 SQL 명령문에 의해 변경된 작업 내용을 전부 취소하고 트랜잭션을 종료

COMMIT & ROLLBACK

▶ COMMIT

- 하나의 트랜잭션에서 실행되는 모든 SQL 명령문의 처리 결과가 하드디스크에 안전하게 보장되는 것을 보장
- 처리 결과를 디스크에 영구적으로 저장
- 해당 트랜잭션에 할당된 CPU, 메모리 같은 자원이 해제
- 서로 다른 트랜잭션을 구분하는 기준
- COMMIT 명령문 실행하기 전에 하나의 트랜잭션 변경한 결과를 다른 트랜잭션에서 접근할 수 없도록 방지하여 일관성 유지

▶ ROLLBACK

- 하나의 트랜잭션에서 실행된 SQL 명령문의 처리결과를 취소
- CPU,메모리 같은 해당 트랜잭션에 할당된 자원을 해제, 트랜잭션을 강제 종료

10. 테이블 생성과 관리

- 테이블 생성
- 테이블 변경
- 테이블 삭제
- 제약조건

테이블 생성 - 개요

- ▶ 테이블 생성은 테이블에 대한 구조를 정의하고, 데이터를 저장하기 위한 공간을 할당하는 과정
- ▶ 테이블에 대한 구조 정의는 테이블을 구성하는 칼럼의 데이터 타입과 무결서 제약조건을 정의하는 과정
- ▶ 테이블 이름 정의 방법
 - 문자(A-Z, a-z)로 시작, 30자 이내
 - 문자(a-z, A-Z), 숫자(0-9), 특수문자(underscore, dollar, hash) 사용 가능
 - 대소문자 구별 없음, 소문자로 저장하려면 단일 인용부호 이용
 - 동일 사용자가 소유한 다른 객체의 이름과 중복 불가
 - 서로 다른 테이블에서 동일한 데이터를 저장하는 칼럼 이름은 가능하면 같은 이름을 사용
 - 필요에 따라 언제든지 테이블 생성 가능
 - 완성된 설계도에 따라 테이블을 생성 권장

테이블 생성 - 기본

```
CREATE TABLE table_name  
( column_name datatype [DEFAULT expr] [ column_constraint ] [, ... ]  
[ , table_constraint , ... ] ) ;
```

- ▶ **table_name**
 - 테이블이름지정
- ▶ **column_name**
 - 컬럼이름지정
- ▶ **datatype**
 - 컬럼의 데이터타입,크기지정
- ▶ **DEFAULT expr**
 - 해당 컬럼에 적용될 자동 기본값

테이블 생성 - 기본

```
-- CREATE TABLE EMP3  
-- (EMPNO      NUMBER PRIMARY KEY,  
  ENAME      VARCHAR(10),  
  JOB        VARCHAR(9),  
  MGR        NUMBER,  
  HIREDATE   DATE,  
  SAL        NUMBER,  
  COMM       NUMBER,  
  DEPTNO     NUMBER);
```

테이블 생성 - 서브쿼리 이용

```
CREATE TABLE table_name [ ( column_name [DEFAULT expr] [, ... ] ) ]  
AS SUBQUERY ;
```

- ▶ CREATE TABLE 명령문에서 서브쿼리 절을 이용하여 다른 테이블의 구조와 데이터를 복사하여 새로운 테이블 생성 가능
- ▶ 서브쿼리의 결과의 구조대로 테이블 생성 및 레코드 삽입이 동시에 행해짐
- ▶ CREATE TABLE 명령문에서 지정한 칼럼 수와 데이터 타입과 반드시 일치
- ▶ 칼럼 이름을 명시하지 않을 경우 서브쿼리 칼럼 이름과 동일

테이블 생성 - 서브쿼리 이용

- ▶ 무결성 제약조건은 NOT NULL 조건만 복사
 - ▶ 기본 키, 참조 키와 같은 무결성 제약조건은 사용자의 재정의 필요
- ▶ 디폴트 옵션에서 정의한 값은 그대로 복사

```
CREATE TABLE EMP4
AS (
    SELECT EMPNO, ENAME, JOB, DEPTNO
    FROM EMP
);
```

테이블 변경 - 개요

- ▶ ALTER TABLE 명령문 , RENAME 명령문 이용
- ▶ 컬럼 관련
 - 컬럼 추가/삭제
 - 컬럼이름 ,데이터타입 ,DEFAULT 변경
- ▶ 제약조건 관련
 - 제약조건추가/삭제
 - 제약조건이름 변경
- ▶ 테이블 관련
 - 테이블이름변경

테이블 변경 - 컬럼 관련(1/2)

▶ 컬럼 추가

- 추가된 칼럼은 테이블의 마지막 부분에 생성, 위치 지정 불가능
- 추가된 칼럼에도 기본 값을 지정 가능

```
ALTER TABLE table  
ADD      ( column datatype [DEFAULT expression]  
          [, column datatype]... );
```

▶ 컬럼 삭제

- 2개 이상의 칼럼이 존재하는 테이블에서만 삭제 가능
- 하나의 칼럼 삭제 명령문은 하나의 칼럼만 삭제 가능

```
ALTER TABLE table DROP column;
```

테이블 변경 - 컬럼 관련(2/2)

▶ 컬럼 변경

- 기존 칼럼에 데이터가 없는 경우
- 칼럼 타입이나 크기 변경이 자유로움
- 기존 데이터가 존재하는 경우
- 타입 변경은 CHAR와 VARCHAR2만 허용
- 변경한 칼럼의 크기가 저장된 데이터의 크기보다 같거나 클 경우 변경 가능
- 숫자 타입에서는 정밀도 증가 가능
- 기본 값의 변경은 변경 후에 입력되는 데이터부터 적용

```
ALTER TABLE table
MODIFY ( column datatype [DEFAULT expression]
        [, column datatype]... );
```


테이블 변경 - 테이블 관련

▶ 테이블이름 변경

- RENAME 명령문 사용
 - ▶ 객체의 이름을 변경하는 DDL 명령문
 - ▶ 뷰, 시퀀스, 동의어 등과 같은 데이터베이스 객체의 이름 변경 가능

```
RENAME old_table TO new_table;
```

테이블 삭제

- ▶ DROP TABLE 명령문 사용
- ▶ 기존 테이블과 데이터를 모두 삭제
- ▶ 삭제된 테이블 컬럼에 대해 생성된 인덱스도 함께 삭제
- ▶ 삭제할 테이블의 기본 키나 고유 키를 다른 테이블에서 참조하고 있는 경우 삭제 불가능
 - 참조하는 테이블(자식 테이블)을 먼저 삭제
 - DROP TABLE 명령문 마지막에 CASCADE CONSTRAINTS 옵션을 사용하여 무결성 제약조건을 동시에 삭제

```
DROP TABLE [schema.] table [cascade constraints];
```

제약조건 - 개요

▶ 데이터 무결성 제약조건의 개념

- 데이터의 정확성과 일관성을 보장
- 데이터의 정확성을 유지하여 다양한 종류의 업무규칙 고려한 예

▶ 데이터 무결성 제약조건의 장점

- 테이블 생성시 무결성 제약조건을 정의 가능
- 테이블에 대해 정의, 데이터 디렉너리에 저장되므로
응용 프로그램에서 입력된 모든 데이터에 대해 동일하게 적용
- 제약조건을 활성화, 비활성화 할 수 있는 융통성

제약조건 - 종류

제약조건	설명
NOT NULL	해당 칼럼 값은 NULL을 포함할 수 없음
고유키	테이블내서 해당 칼럼 값은 항상 유일해야함
기본키	해당 칼럼 값은 반드시 존재해야 하며, 유일해야함 UNIQUE, NOT NULL 제약조건을 결합한 형태
참조	해당 칼럼 값은 참조되는 테이블의 칼럼 값 중의 하나와 일치하거나 NULL을 가짐
CHECK	해당 칼럼에 저장 가능한 데이터 값의 범위나 조건 지정

제약조건 - 정의(1/2)

- ▶ 이름으로 관리
 - 문자로 시작,길이는 30자까지 가능
 - 이름을 따로 지정하지 않으면 자동 생성(SYS_Cxxxxxxx형식)
- ▶ 정의 시기
 - 테이블 생성과 동시에 정의
 - 테이블을 생성한 후 ALTER문 이용하여 정의
- ▶ 컬럼 레벨 또는 테이블레벨에서 정의할 수 있음
 - NOT NULL은 '컬럼레벨'에서만 가능
 - 컬럼 여러 개를 조합하여 제약조건을 정의하는 경우에는 '테이블레벨'에서만 가능

제약조건 - 정의(2/2)

```
CREATE TABLE CONSTRAINT_EMP
(EID          CHAR(3) CONSTRAINT PKEID PRIMARY KEY,
 ENAME        VARCHAR2(20) CONSTRAINT NENAME NOT NULL,
 ENO          CHAR(14) CONSTRAINT NENO NOT NULL CONSTRAINT UENO UNIQUE,
 EMAIL        VARCHAR2(25) CONSTRAINT UEMAIL UNIQUE,
 PHONE        VARCHAR2(12),
 HIRE_DATE    DATE DEFAULT SYSDATE,
 JID          CHAR(2) CONSTRAINT FKJID REFERENCES JOB ON DELETE SET NULL,
 SALARY       NUMBER,
 BONUS_PCT    NUMBER,
 MARRIAGE     CHAR(1) DEFAULT 'N' CONSTRAINT CHK CHECK (MARRIAGE IN ('Y','N')),
 MID          CHAR(3) CONSTRAINT FK MID REFERENCES CONSTRAINT_EMP ON DELETE SET NULL,
 DID          CHAR(2),
 CONSTRAINT FKDID FOREIGN KEY (DID) REFERENCES DEPARTMENT ON DELETE CASCADE
);
```

11. View / Index / Sequence

- VIEW
- VIEW 생성 및 사용
- INDEX
- INDEX 생성
- SEQUENCE
- SEQUENCE 생성 및 사용

VIEW - 개요

- ▶ 다른 테이블이나 다른 뷰에 포함된 데이터의 부분 집합을 갖는 가상의 논리테이블
- ▶ "STORED QUERY" 또는 "VIRTUAL TABLE"로 간주되는 데이터베이스 객체
- ▶ 생성된 뷰는 테이블처럼 사용됨
- ▶ 자체적으로 데이터를 포함하지 않음
 - 베이스 테이블(Base Table)
 - ▶ 뷰를 통해 보여지는 데이터를 포함하고 있는 실제 테이블

VIEW - 사용 이점

- ▶ 뷰에 접근하는 사용자에게 미리 정의된 결과만을 볼 수 있음
- ▶ 복잡한 SQL문을 뷰로 저장하여 반복적으로 사용 가능
- ▶ 여러 테이블을 조인하는 등 복잡한 SQL구문을 사용해야 하는 경우 뷰로 정의하여 사용하면 SQL구문이 간단해짐
- ▶ 뷰에 포함되는 컬럼은 베이스 테이블에 영향을 주지 않고 다른 이름으로도 참조 가능
- ▶ 베이스 테이블에 포함된 여러 개 컬럼 중 일부만 사용하도록 뷰를 생성한 경우,뷰가 참조하지 않는 나머지 컬럼이 변경되어도 뷰를 사용하는 다른프로그램들은 영향을 받지 않음

VIEW 생성 및 사용(1/3)

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW view_name [( alias [, alias ...])]
AS Subquery
[WITH CHECK OPTION [ CONSTRAINT constraint_name ]]
[WITH READ ONLY [ CONSTRAINT constraint_name ]] ;
```

▶ CREATE OR REPLACE

- 지정한 이름의 뷰가 없으면 새로 생성
- 동일 이름이 존재하면 Replace

▶ FORCE|NOFORCE

- NOFORCE
 - ▶ 베이스 테이블이 존재하는 경우에만 뷰 생성 가능
- FORCE
 - ▶ 베이스 테이블이 존재하지 않아도 뷰 생성 가능

VIEW 생성 및 사용(2/3)

▶ ALIAS

- 뷰에서 사용할 컬럼 이름의미
- 생략 시 SUBQUERY에서 사용한 이름 적용
- ALIAS개수
 - ▶ SUBQUERY에서 사용한 컬럼 개수와 일치

▶ SUBQUERY

- 뷰에서 표현하는 데이터를 생성하는 SELECT구문

▶ 제약조건

- WITH CHECK OPTION
 - ▶ 뷰를 통해 접근 가능한 데이터에 대해서만 DML작업 허용
- WITH READ ONLY
 - ▶ 뷰를 통해 DML작업 허용 안 함
 - ▶ 제약조건으로 간주되므로 별도 이름 지정 가능

VIEW생성 및 사용(3/3)

```
-- CREATE OR REPLACE VIEW V_EMP  
AS SELECT ENAME, DEPTNO  
   FROM EMP  
   WHERE DEPTNO = '10';
```

```
-- SELECT ENAME, D.DNAME  
FROM   V_EMP V, DEPT D  
WHERE  V.DEPTNO = D.DEPTNO;
```

```
-- CREATE OR REPLACE VIEW V_EMP_DEPT(enm, dnm)  
AS SELECT ENAME, DNAME  
   FROM EMP  
   LEFT JOIN DEPT USING(DEPTNO);
```

```
-- SELECT *  
FROM   V_EMP_DEPT;
```

INDEX

- ▶ INDEX를 생성하면 검색의 효율이 향상됨
 - DISK I/O를 줄임으로써 검색속도를 향상시킬 수 있음
- ▶ 정렬된 특정 컬럼 값과 해당 컬럼 값이 포함된 행 위치로 구성

- ▶ 인덱스를 사용하는 것이 효율적인 경우
 - WHERE절이나 JOIN조건에 주로 사용되는 컬럼
 - UNIQUE속성의 컬럼이나 NULL이 많이 포함된 컬럼
- ▶ 인덱스를 사용하지 않는 것이 더 효율적인 경우
 - 테이블이 작은 경우(데이터가 적은 경우)
 - 데이터 갱신이 자주 발생하는 경우
 - 다량의 데이터가 조회되는 경우

INDEX생성

```
CREATE [UNIQUE] INDEX index_name ON table_name ( column_list | function, expr );
```

▶ Unique Index

- UniqueIndex가 생성된 컬럼에는 중복 값이 포함될 수 없음
- 오라클은 'PRIMARYKEY' 제약조건을 생성하면 자동으로 해당 컬럼에 UniqueIndex를 생성

▶ Nonunique Index

- 빈번하게 사용되는 일반컬럼을 대상으로 생성함
- 주로 성능 향상을 위한 목적으로 생성

SEQUENCE

- ▶ 순차적으로 정수 값을 자동으로 생성하는 객체
- ▶ 유일한 식별자로 주로 사용되어 짐
- ▶ 여러 테이블에서 공유 가능
- ▶ 예
 - 게시판 글 번호

SEQUENCE 생성 및 사용 - 생성(1/2)

```
CREATE SEQUENCE sequence_name
[ INCREMENT BY N ] [ START WITH N ]
[ { MAXVALUE N | NOMAXVALUE } ] [ { MINVALUE N | NOMINVALUE } ]
[ { CYCLE | NOCYCLE } ] [ { CACHE N | NOCACHE } ] ;
```

▶ INCREMENT BY *n*

- 시퀀스 번호의 증가치로 기본은 1, 일반적으로 1 사용

▶ START WITH *n*

- 시퀀스 시작번호, 기본값은 1

▶ MAXVALUE *n*

- 생성 가능한 시퀀스의 최대값

▶ NO MAXVALUE

- 시퀀스 번호를 순환적으로 사용하는 cycle로 지정한 경우, MAXVALUE에 도달한 후 새로 시작하는 시퀀스값

SEQUENCE 생성 및 사용 - 생성(2/2)

▶ CYCLE | NOCYCLE

- MAXVALUE 또는 MINVALUE에 도달한 후 시퀀스의 순환적인 시퀀스 번호의 생성 여부 지정

▶ CACHE n | NOCACHE

- 시퀀스 생성 속도 개선을 위해 메모리에 캐시하는 시퀀스 개수, 기본값은 20

```
-- CREATE SEQUENCE SEQ_EMPNO  
START WITH 10000  
INCREMENT BY 5  
NOCYCLE  
NOCACHE;  
--
```

SEQUENCE 생성 및 사용 - 사용

▶ CURRVAL

- 시퀀스에서 생성된 현재 번호를 확인
- 시퀀스이름.CURRVAL

▶ NEXTVAL

- 시퀀스에서 다음 번호 생성
- 시퀀스이름.NEXTVAL

```
INSERT INTO EMP(EMPNO, ENAME, JOB, SAL, DEPTNO)  
VALUES (SEQ_EMPNO.NEXTVAL, 'JANE', 'OPERATOR', 1000, 40);
```