

Abstract geometric lines in the top left corner.

# SQLD Summary

# 데이터 모델링

- 현실 세계를 단순화하여 표현하는 기법
- 현실 세계에서 일어날 수 있는 다양한 현상에 대해 일정한 표기법에 따라 표현한 모형을 만드는 과정
- 모델링이 갖춰야 할 조건 ( 특징 )

특징	설명
추상화	현실 세계 반영. 현실 세계를 간략하게 표현
단순화	누구나 쉽게 이해할 수 있도록 표현
명확성	명확하게 의미가 해석되어야 하고 한 가지 의미를 가져야 한다

# 데이터 모델링

## ■ 모델링의 세 가지 단계

단계	설명
개념적 모델링	<ul style="list-style-type: none"><li>▪ 업무 중심적, 포괄적 수준의 모델링</li><li>▪ 전사적 관점에서 기업의 데이터 모델링</li><li>▪ 추상화 수준이 가장 높은 단계</li><li>▪ 계층형, 네트워크, 관계형 모델을 구분하지 않고 업무 측면에서 모델링</li></ul>
논리적 모델링	<ul style="list-style-type: none"><li>▪ 범용 데이터베이스 관점, 무결성에 중점을 둔 모델링</li><li>▪ 특정 데이터베이스 모델에 종속 ( 예: 관계형 데이터 모델 )</li><li>▪ 식별자를 정의하고 관계, 속성을 표현</li><li>▪ 정규화를 통해 무결성을 높인다</li></ul>
물리적 모델링	<ul style="list-style-type: none"><li>▪ 구체적인 데이터베이스 관점, 성능에 중점을 둔 모델링</li><li>▪ 특정 데이터베이스 관리시스템에 테이블, 인덱스 등을 생성하는 단계</li><li>▪ 성능, 보안, 가용성 등을 고려하여 데이터베이스 모델링</li></ul>

# 데이터 모델링

## ■ 모델링 관점

관점	설명
데이터 관점	<ul style="list-style-type: none"><li>▪ 데이터 위주의 모델링</li><li>▪ 업무에 연관된 데이터 및 데이터 사이의 관계 모델링</li><li>▪ 비즈니스 프로세스에서 사용하는 데이터</li><li>▪ 구조 분석, 정적 분석</li></ul>
프로세스 관점	<ul style="list-style-type: none"><li>▪ 프로세스 위주의 모델링</li><li>▪ 업무가 처리하고 있는 일 및 처리해야 할 일 모델링</li><li>▪ 비즈니스 프로세스에서 수행하는 작업</li><li>▪ 시나리오 분석, 도메인 분석, 동적 분석</li></ul>
데이터와 프로세스 관점 (상관 관점)	<ul style="list-style-type: none"><li>▪ 프로세스와 데이터 사이의 관계</li><li>▪ 프로세스의 흐름에 따라 데이터가 어떤 영향을 받는지 모델링</li><li>▪ CRUD(Create, Read, Update, Delete) 분석</li></ul>

# 데이터 모델링

## ■ 데이터 품질 결정 요인

요인	설명
중복	<ul style="list-style-type: none"><li>▪ 같은 데이터가 여러 엔티티에 중복으로 저장되는 현상을 피해야 한다</li></ul>
유연성	<ul style="list-style-type: none"><li>▪ 애플리케이션의 변경이 데이터 모델에 미치는 영향 최소화</li><li>▪ 데이터 모델과 프로세스 모델 분리</li></ul>
일관성	<ul style="list-style-type: none"><li>▪ 다른 데이터와의 연관성을 고려하지 않고 일부 데이터만 변경하는 것을 막아야 한다</li><li>▪ 데이터 사이의 연관관계를 명확하게 정의</li></ul>

# 데이터 모델링

- ANSI-SPARC 아키텍처

- 사용자, 설계자, 개발자가 데이터베이스를 보는 관점에 따라 데이터베이스를 기술하고 관계를 정의한 추상적인 설계 표준 ( ANSI 표준 )
- 데이터베이스 독립성을 확보하기 위한 방법
- 데이터의 독립성 확보 → 복잡도 감소, 데이터 중복 제거, 사용자 요구사항 변경에 따른 대응력 향상, 관리 및 유지보수 절감 등의 장점
- 3단계 계층으로 분리해서 독립성을 확보하는 방법으로 각 계층을 뷰라고도 한다

- 3층 스키마의 독립성

- 논리적 독립성 → 개념 스키마가 변경되더라도 외부 스키마가 영향을 받지 않는 것
- 물리적 독립성 → 내부 스키마가 변경되더라도 개념 스키마가 영향을 받지 않는 것

# 데이터 모델링

- ANSI-SPARC 3단계 스키마 구조

- 외부 스키마

- 사용자 관점 ( Multiple User's View ) → 각 사용자가 보는 데이터베이스 스키마
    - 업무상 관련 있는 데이터 접근
    - 관련 데이터베이스의 뷰 표시
    - 응용 프로그램이 접근하는 데이터베이스 정의

- 개념 스키마

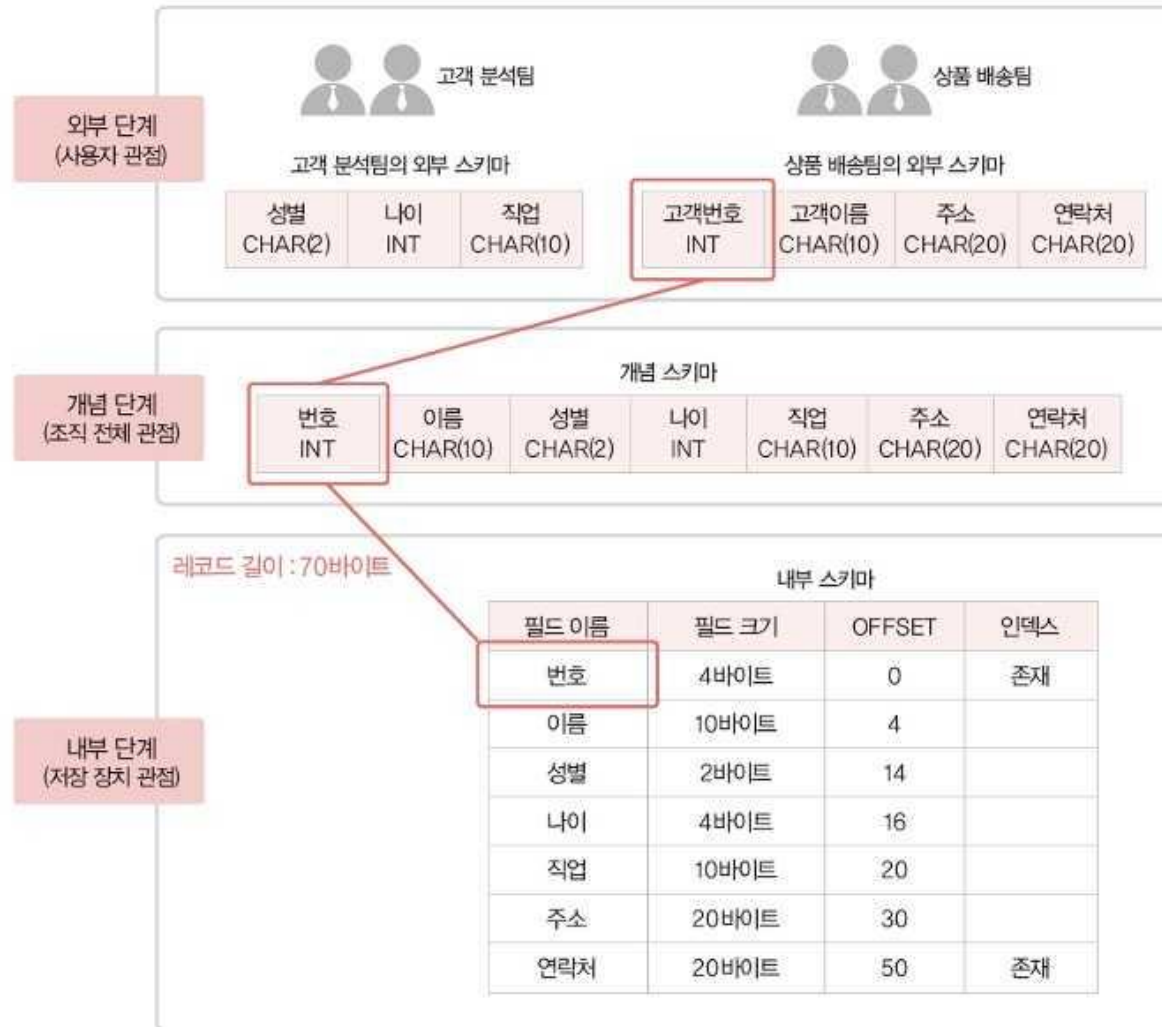
- 설계자 관점. 사용자 전체 집단의 데이터베이스 구조 (통합 관점)
    - 각 사용자가 보는 데이터베이스 스키마를 통합하여 전체 데이터베이스로 표현
    - 데이터베이스에 저장되는 데이터와 데이터 사이의 관계 표현
    - 통합 데이터베이스 구조

- 내부 스키마

- 개발자 관점.
    - 물리적 데이터 저장 구조 표현 (물리적 관점)
    - 데이터 저장 구조, 컬럼 정의, 인덱스 등 포함

# 데이터 모델링

## ANSI-SPARC 3단계 스키마 구조 (계속)





# 데이터 모델링

## ■ ERD ( Entity Relationship Diagram )

- » 시스템에 존재하는 엔티티와 엔티티 사이의 관계를 표현하는 다이어그램
- » 표기방식 종류

㉠ Peter Chen : 주로 대학교재에서 사용하는 표기법으로 실무에서 사용하는 경우는 드물다.



㉡ IDEF1X(Integration Definition for Information Modeling) : 실무에서 사용하는 경우도 있으며 ERWin에서 사용되는 모델이기도 하다(ERWin은 ERD를 그리는 모델링 툴).



㉢ IE/Crow's Foot : 까마귀발 표기법이라고도 부르며 가장 많이 사용한다. ERWin, ERStudio에서 사용되는 모델이다(ERWin, ERStudio는 ERD를 그리는 모델링 툴).



㉣ Min-Max/ISO : 각 엔티티의 참여도를 좀 더 상세하게 나타내는 표기법이다.



㉤ UML : 소프트웨어 공학에서 주로 사용되는 모델이다.











㉥ Case\*Method/Barker : Oracle에서 사용되는 모델로 Crow's Foot과 비슷하다.



# 데이터 모델링

- ERD ( Entity Relationship Diagram ) (계속)
  - » 시스템에 존재하는 엔티티와 엔티티 사이의 관계를 표현하는 다이어그램
  - » IE/Crow's Foot

기호	의미
	엔티티
	0개
	1개
	2개 이상
	부모 엔티티의 식별자가 자식 엔티티의 주식별자(식별자 관계) 
	부모 엔티티의 식별자가 자식 엔티티의 일반 속성(비식별자 관계) 

# 데이터 모델링

- ERD ( Entity Relationship Diagram ) 작성 절차
  - » 엔티티 도출 및 표시
  - » 엔티티 배치 → 중요 엔티티를 왼쪽 상단에 배치
  - » 엔티티 사이의 관계 설정
  - » 관계 이름 기술 → 엔티티 사이의 어떤 행위나 존재가 있는지 표현
  - » 관계 참여도 표현 → ( 1:1, 1:M, M:M )
  - » 관계의 필수 여부 표현 → ( Mandatory, Optional )
- ERD 작성시 고려 사항
  - » 중요한 엔티티를 왼쪽 상단에 배치
  - » 이해하기 쉬워야 하고 복잡하지 않아야 한다

# 데이터 모델링

- 엔티티 (Entity)
  - » 식별 가능한 객체
  - » 업무에서 쓰이는 데이터를 용도별로 분류한 그룹
  - » 업무에서 저장, 관리해야 하는 데이터
  - » 개념, 사건, 장소 등의 명사

이름	시기	정의
Peter Chen	1976	식별할 수 있는 사물
C.J Date	1986	데이터베이스 내에서 식별 가능한 객체
James Martin	1989	정보를 저장할 수 있는 어떤 것
Thomas Bruce	1992	정보를 저장할 수 있는 사람, 장소, 물건, 사건 그리고 개념 등

- 용어 매칭
  - » 엔티티 → 테이블
  - » 속성 → 컬럼
  - » 인스턴스 → 행 (Row)

# 데이터 모델링

## ■ 엔티티 특징

- » 업무에서 쓰이는 정보여야 함
- » 반드시 속성을 가져야 함
- » 고유함을 보장하는 속성(식별자)을 가져야 함
- » 2개 이상의 인스턴스 ( 업무를 통해 반복적 발생 )를 가져야 함
- » 다른 엔티티와 1개 이상의 관계를 가져야 함

## ■ 엔티티 분류

### » 유형/무형 기준

유형 엔티티	물리적인 형태 존재, 안정적, 지속적 예 상품, 회원 등
개념 엔티티	물리적인 형태 없음, 개념적 예 부서, 학과 등
사건 엔티티	행위를 함으로써 발생, 빈번함, 통계 자료로 이용 가능 예 주문, 이벤트 응모 등

### » 발생 시점 기준

기본 엔티티	• 업무에 원래 존재하는 정보 • 독립적으로 생성되며, 자식 엔티티를 가질 수 있음 예 상품, 회원, 사원, 부서 등
중심 엔티티	• 기본 엔티티로부터 파생되고, 행위 엔티티 생성 • 업무에 있어서 중심적인 역할을 하며 데이터의 양이 많이 발생 예 주문, 매출, 계약 등
행위 엔티티	• 2개 이상의 엔티티로부터 파생 • 데이터가 자주 변경되거나 증가할 수 있음 예 주문 내역, 이벤트 응모 이력 등

# 데이터 모델링

## ■ 엔티티 명명 관행

- » 업무에서 실제로 쓰이는 용어 사용
- » 한글은 약어를 사용하지 않고 영문은 대문자로 표기
- » 단수 명사로 표현하고 띄어쓰기는 하지 않음
- » 다른 엔티티와 의미상으로 중복될 수 없음
- » 해당 엔티티에의 데이터가 무엇인지 명확하게 표현

## ■ 속성

- » 엔티티를 구성하는 개별 정보
- » 사물 또는 개념의 특징을 설명하는 항목
- » 의미상 더 이상 분할할 수 없는 레벨이어야 함
- » 업무 프로세스에 필요한 것 ( 업무에서 관리되는 정보 )
- » 주식별자에게 함수적으로 종속됨 → 기본키가 변경되면 속성의 값도 변경

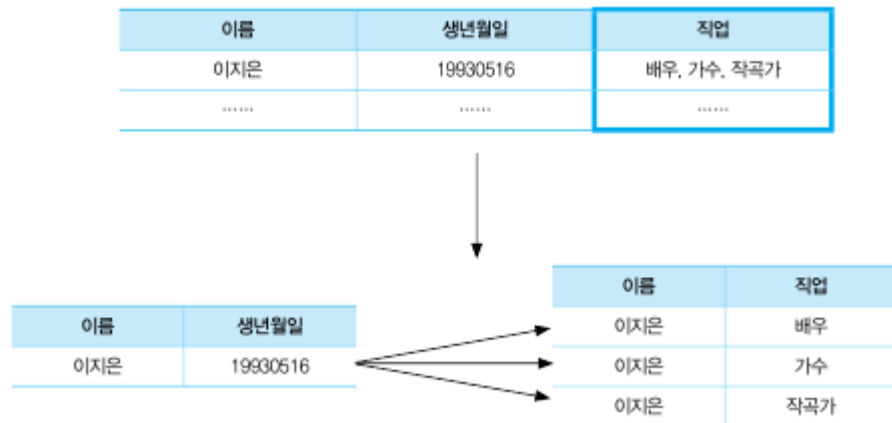
# 데이터 모델링

## ■ 속성값

- » 각각의 속성은 속성값을 가지며 속성값은 엔티티에 속한 하나의 인스턴스를 구체적으로 표현하는 데이터

속성	속성값
이름	이지은
생년월일	19930516
소속사	EDAM엔터테인먼트

- » 하나의 속성은 한 개의 속성값만 가질 수 있음 ( 여러 개의 값을 가지는 경우 엔티티 분할 및 관계 형성 )



# 데이터 모델링

- 엔티티, 인스턴스, 속성, 속성값의 관계
  - » 엔티티 > 인스턴스 > 속성



- › 한 개의 엔티티는 두 개 이상의 인스턴스를 갖는다
- › 한 개의 엔티티(인스턴스)는 두 개 이상의 속성을 갖는다
- › 한 개의 속성은 한 개의 속성값을 갖는다



# 데이터 모델링

## ■ 속성 분류

### » 특성에 따른 분류

기본속성(Basic Attribute)	업무 프로세스 분석을 통해 바로 정의가 가능한 속성
설계속성(Designed Attribute)	업무에 존재하지는 않지만 설계하다 보니 필요하다고 판단되어 도출해낸 속성
파생속성(Derived Attribute)	다른 속성의 속성값을 계산하거나 특정한 규칙으로 변형하여 생성한 속성

### » 구성방식에 따른 분류

PK(Primary Key)속성	엔터티의 인스턴스들을 식별할 수 있는 속성
FK(Foreign Key)속성	다른 엔터티의 속성에서 가져온 속성
일반속성	PK, FK를 제외한 나머지 속성

### » 분해 여부에 따른 분류

종류	설명
단일 속성	▪ 하나의 의미로 구성된 것. 회원 아이디, 이름 등
복합 속성	▪ 여러 개의 의미를 포함. 대표적으로 주소 (시, 군, 구, 동 등으로 분해 가능)
다중 값 속성	▪ 속성에 여러 개의 값을 가질 수 있는 것. 상품 리스트 등 ▪ 다중 값 속성은 엔티티로 분해되어야 함

# 데이터 모델링

- 도메인
  - » 속성이 가질 수 있는 속성값의 범위 ( 자료형 + 제약 조건 )
- 용어 사전
  - » 속성의 이름을 정확하고 직관적으로 부여하고 용어의 혼란을 없애기 위해 용어사전 사용
- 시스템 카탈로그
  - » 사용자 테이블과는 별개로 시스템 자체에 관련이 있는 데이터를 담고 있는 데이터베이스
  - » 시스템 카탈로그에 저장된 데이터를 메타데이터라고 함
  - » 시스템 테이블로 구성되어 있어서 SQL로 조회 가능
    - › select는 가능하지만 insert, update, delete는 불가능

# 데이터 모델링

## ■ 관계

- » 엔티티와 엔티티 사이의 관계
- » 종류

존재 관계	<p>존재 자체로 연관성이 있는 관계 예) 직원과 부서, 학생과 학과 등</p> <pre>graph LR; S[학생 정미나] -- "소속된다" --- D[학과 컴퓨터공학과]; style S stroke-width:2px; style D stroke-width:2px;</pre>
행위 관계	<p>특정한 행위를 통해 연관성이 생기는 관계 예) 회원과 주문, 학생과 출석부</p> <pre>graph LR; S[학생 정미나] -- "출석한다" --- A[출석부 데이터베이스]; style S stroke-width:2px; style A stroke-width:2px;</pre>

# 데이터 모델링

## ■ 관계 표기법

관계명(Membership)	관계의 이름
관계차수(Cardinality)	관계에 참여하는 수
관계선택사양(Optionality)	필수인지 선택인지의 여부

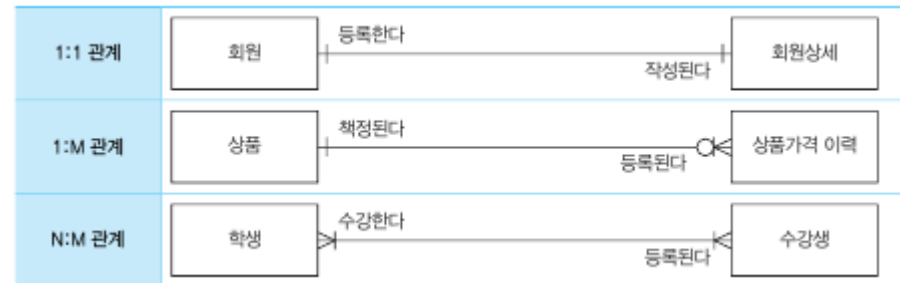
### » 관계명

- › 엔티티 사이에 어떤 관계가 있는지 설명하는 문장
- › 모든 관계는 두 개의 관계명 가짐 (관계에 참여하는 각 엔티티의 관점에서 관계명 정의)
- › 명확하고 현재형 문장으로 표현해야 함



### » 관계차수

- › 각 엔티티에서 관계에 참여하는 수
- › 1:1, 1:M, M:M(N)



### » 관계선택사양

필수적 관계	참여자 가 반드시 존재해야 하는 관계
선택적 관계	참여자 가 없을 수도 있는 관계



# 데이터 모델링

## ■ 식별자

- » 엔티티의 속성 중에서 각 인스턴스를 고유하게 구분할 수 있도록 만들어주는 대표적인 속성
- » 주식별자
  - › 기본키 (Primary Key)
  - › 한 개 이상의 속성이 주식별자가 될 수 있음

유일성	각 인스턴스에 유니크함을 부여하여 식별이 가능하도록 한다.
최소성	유일성을 보장하는 최소 개수의 속성이어야 한다.
불변성	속성값이 되도록 변하지 않아야 한다.
존재성	속성값이 NULL일 수 없다.

## » 식별자 종류

### ① 대표성 여부

주식별자 (Primary Identifier)	<ul style="list-style-type: none"><li>• 유일성, 최소성, 불변성, 존재성을 가진 대표 식별자</li><li>• 다른 엔티티와 참조 관계로 연결</li></ul>
보조식별자 (Alternate Identifier)	<ul style="list-style-type: none"><li>• 인스턴스를 식별할 수는 있지만 대표 식별자가 아님</li><li>• 다른 엔티티와 참조 관계로 연결되지 않음</li></ul>

### ② 스스로 생성되었는지 여부

내부식별자 (Internal Identifier)	엔티티 내부에서 스스로 생성된 식별자
외부식별자 (Foreign Identifier)	다른 엔티티에서 온 식별자, 다른 엔티티와의 연결고리 역할

### ③ 단일 속성의 여부

단일식별자 (Single Identifier)	하나의 속성으로 구성된 식별자
복합식별자 (Composite Identifier)	두 개 이상의 속성으로 구성된 식별자

### ④ 대체 여부

원조식별자 (Original Identifier)	업무 프로세스에 존재하는 식별자, 가공되지 않은 원래의 식별자(본질식별자)
대리식별자 (Surrogate Identifier)	주식별자의 속성이 두 개 이상인 경우 그 속성들을 하나로 묶어서 사용하는 식별자(인조식별자)

# 데이터 모델링

- 식별 관계 VS 비식별 관계

식별 관계	부모 엔티티의 식별자가 자식 엔티티의 주식별자의 일부 또는 전부가 되는 관계
비식별 관계	부모 엔티티의 식별자가 자식 엔티티의 일반 속성이 되는 관계

# 데이터 모델과 SQL

- 정규화

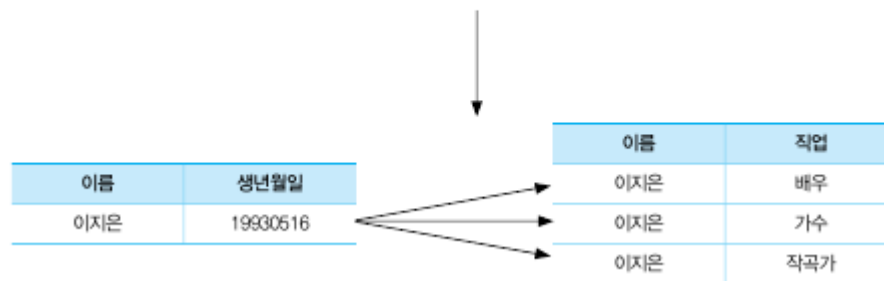
- » 데이터 무결성(정확성과 일관성)을 유지하고 보장하기 위해 엔티티를 작은 단위로 분리하는 과정
- » 최소한의 데이터 중복을 제거하고 최대한의 데이터 유연성을 위한 방법
- » 정규화 결과 조회 성능은 향상되거나 저하될 수 있으나 삽입, 수정, 삭제 성능은 일반적으로 향상됨
- » 종류
  - › 1NF, 2NF, 3NF, BCNF, 4NF, 5NF, 6NF
  - › 비공식적으로 3NF 까지 수행하면 정규화 된 것으로 간주

# 데이터 모델과 SQL

## ■ 제1정규형

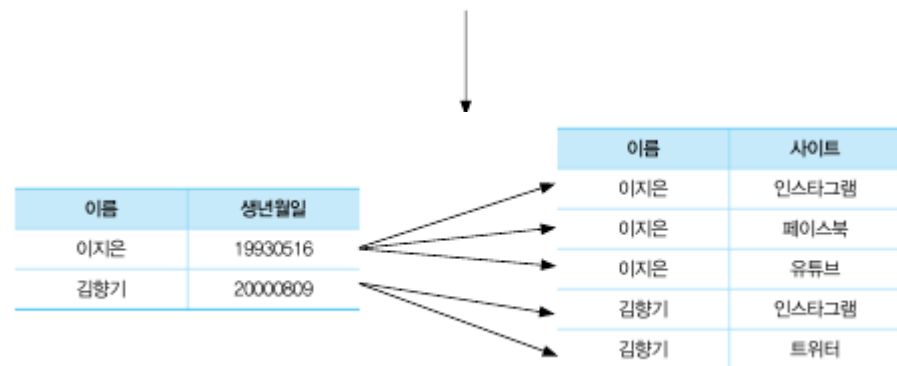
» 모든 속성은 하나의 값만 가져야 한다

이름	생년월일	직업
이지은	19930516	배우, 가수, 작곡가
.....	.....	.....



» 유사한 속성이 반복되는 경우도 제1정규화 대상

이름	생년월일	사이트1	사이트2	사이트3
이지은	19930516	인스타그램	페이스북	유튜브
김향기	20000809	인스타그램	트위터	NULL
.....	.....	.....	.....	.....





# 데이터 모델과 SQL

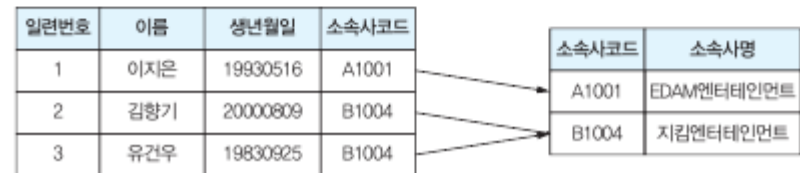
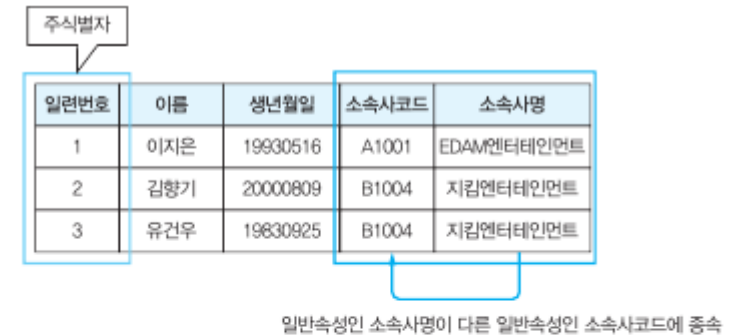
## ■ 제2정규형

- » 엔티티의 모든 일반속성은 반드시 주식별자 전체에 종속되어야 한다



## ■ 제3정규형

- » 주식별자가 아닌 모든 속성 간에는 종속될 수 없다



# 데이터 모델과 SQL

## ■ 반정규화

- » 데이터 조회 성능 향상을 위해 데이터의 중복을 허용하거나 그룹핑하는 과정으로 정규화를 일부 완화하는 것을 의미
- » 조회 성능은 향상될 수 있으나 입력, 수정, 삭제 성능은 저하될 수 있고 데이터 무결성 문제 발생 가능
- » 반정규화 과정은 정규화가 끝난 후에 수행

## ■ 테이블 반정규화

테이블 병합	1:1 관계 테이블 병합
	1:M 관계 테이블 병합
	슈퍼 서브 타입 테이블 병합
테이블 분할	테이블 수직 분할(속성 분할)
	테이블 수평 분할(인스턴스 분할, 파티셔닝)
테이블 추가	중복 테이블 추가
	통계 테이블 추가
	이력 테이블 추가
	부분 테이블 추가

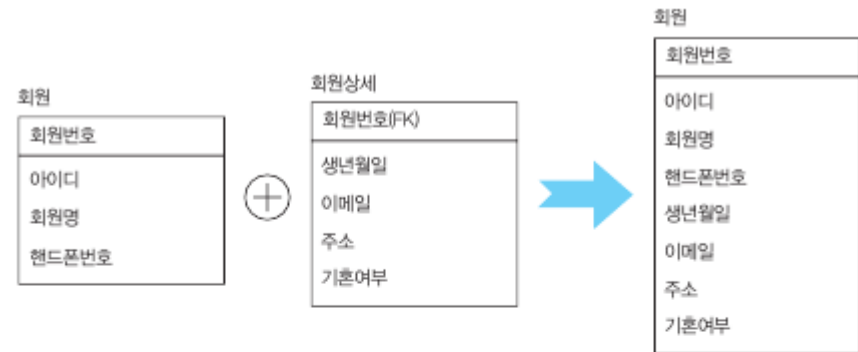
# 데이터 모델과 SQL

## ■ 테이블 반정규화 (계속)

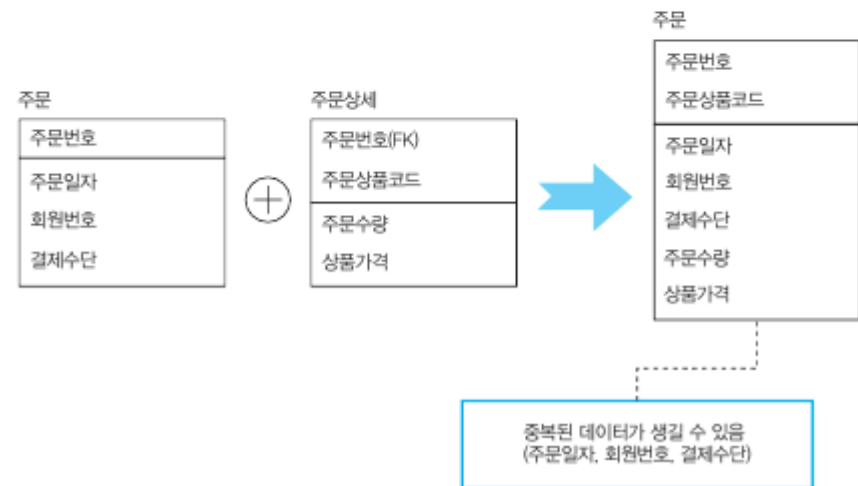
### » 테이블 병합

- › Join이 필요한 경우가 많아 테이블을 통합하는 것이 성능 측면에서 유리할 경우 고려
- › 1:M 관계 테이블 병합의 경우 1쪽에 해당하는 엔티티의 속성 개수가 많으면 병합했을 때 중복 데이터가 많아져서 부적합

㉠ 1:1 관계 테이블 병합



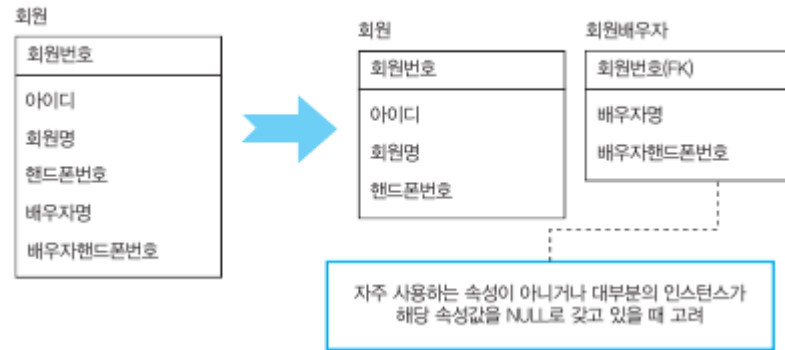
㉡ 1:M 관계 테이블 병합



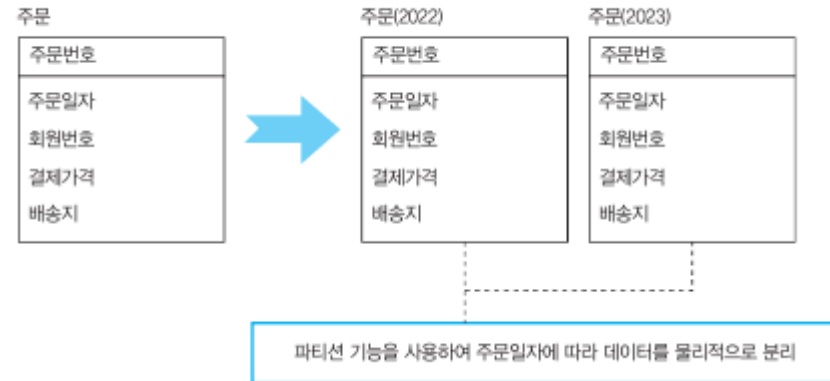
# 데이터 모델과 SQL

- 테이블 반정규화 (계속)
  - 테이블 분할

㉠ 테이블 수직 분할 : 엔터티의 일부 속성을 별도의 엔터티로 분할(1:1 관계 성립)한다.



㉡ 테이블 수평 분할 : 엔터티의 인스턴스를 특정 기준으로 별도의 엔터티로 분할(파티셔닝)한다.



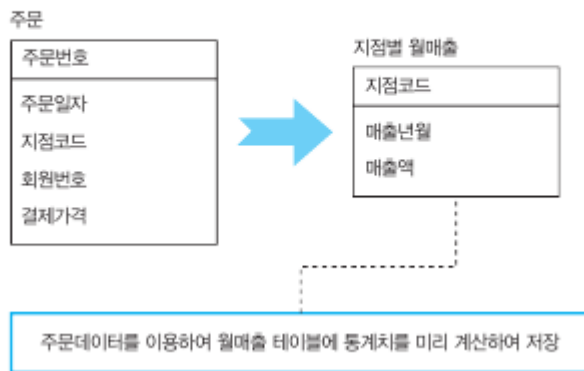
# 데이터 모델과 SQL

## ■ 테이블 반정규화 (계속)

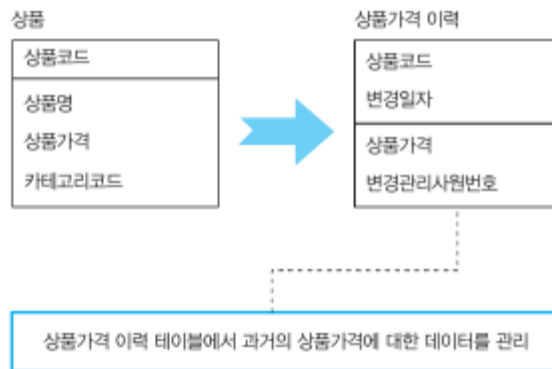
### » 테이블 추가

㉠ 중복 테이블 추가 : 데이터의 중복을 감안하더라도 성능상 반드시 필요하다고 판단되는 경우 별도의 엔티티를 추가한다.

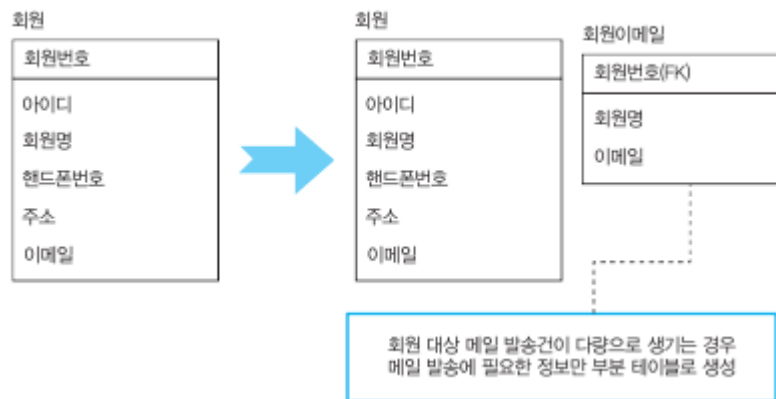
㉡ 통계 테이블 추가



㉢ 이력 테이블 추가



㉣ 부분 테이블 추가



# 데이터 모델과 SQL

## ■ 컬럼 반정규화

중복 컬럼 추가	업무 프로세스상 JOIN이 필요한 경우가 많아 컬럼을 추가하는 것이 성능 측면에서 유리할 경우 고려
파생 컬럼 추가	프로세스 수행 시 부하가 염려되는 계산 값을 미리 컬럼으로 추가하여 보관하는 방식으로 상품의 재고나 프로모션 적용 할인가 등에 적용
이력 테이블 컬럼 추가	대량의 이력 테이블을 조회할 때 속도가 느려지는 것을 대비하여 조회 기준이 될 것으로 예상되는 컬럼을 미리 추가해 두는 방식. 최신 데이터 여부 등

## ■ 관계 반정규화 ( 중복 관계 추가 )

- » 업무 프로세스상 JOIN이 필요한 경우가 많아서 중복 관계를 추가하는 것이 성능 측면에서 유리할 경우 고려

# 데이터 모델과 SQL

- 트랜잭션

- » 데이터를 조작하기 위한 하나의 논리적인 작업 단위
- » 1개 이상의 물리적인 작업으로 구성됨
- » 트랜잭션에 포함된 물리적인 작업은 모두 성공하거나 모두 실패하는 경우만 가능

- NULL

- » 존재하지 않음, 값이 없음을 의미





# SQL

## ▪ SELECT 구문

형식	설명
SELECT 절	검색할 컬럼 목록 [HINT] 옵티마이저에게 검색방법 제시 [DISTINCT] 중복된 자료 제거
FROM 절	SELECT문에서 선언된 칼럼을 가져올 테이블 목록
WHERE 절	가져올 행의 조건 논리연산자(AND, OR) 사용이 가능
GROUP BY 절	그룹 함수를 사용 할 때 사용. 주로 SELECT 절에 선언된 컬럼들 중 하나 이상의 칼럼으로 정함. 원래의 행에 대한 결과를 보여주는 것이 아니고 GROUP BY에서 선언된 열에 동일한 값으로 그룹화 하여 결과를 보여준다.
HAVING 절	GROUP BY 절에 의해 구성된 그룹들에 대해 적용할 조건을 기술. SELECT 문의 WHERE절과 비슷한 역할
START WITH 절	CONNECT BY 절에 적용할 조건을 기술
CONNECT BY 절	계층구조 조인 시에 사용
ORDER SIBLINGS BY 절	계층구조 안에서 정렬 순서를 정함
ORDER BY 절	결과 값의 정렬 순서를 정함

# SQL

## ■ 산술 연산자

연산자	의미	우선순위
()	괄호로 우선순위를 조정할 수 있음	1
*	곱하기	2
/	나누기(0으로 나눌 경우 예외 발생)	
+	더하기	3
-	빼기	
% (SQL Server)	나머지(0으로 나눌 경우 NULL 반환)	3

## ■ NULL을 포함한 연산 → 결과는 NULL

## ■ 합성 연산자 ( || ) → 문자열 결합

» SELECT 'S' || 'Q' || 'L' || 'Developer' FROM DUAL

# SQL 함수

## ■ 문자 함수

### ① CHR(ASCII 코드)

ASCII 코드는 총 128개의 문자를 숫자로 표현할 수 있도록 정의해 놓은 코드이다. CHR 함수는 ASCII 코드를 인수로 입력했을 때 매핑되는 문자가 무엇인지를 알려주는 함수이다.

SQL Server(MSSQL)의 경우 CHAR(ASCII 코드)

예 CHR(65) → A

```
1 SELECT CHR(65) FROM DUAL;
```

### ② LOWER(문자열)

문자열을 소문자로 변환해주는 함수이다.

예 LOWER('JENNIE') → jennie

### ③ UPPER(문자열)

문자열을 대문자로 변환해주는 함수이다.

예 UPPER('jennie') → JENNIE

```
1 SELECT UPPER('JENNIE') FROM DUAL;
```

### ④ LTRIM(문자열 [,특정 문자]) \* [ ]는 옵션

특정 문자를 따로 명시해주지 않으면 문자열의 왼쪽 공백을 제거하고, 명시해주었을 경우 문자열을 왼쪽부터 한 글자씩 특정 문자와 비교하여 특정 문자에 포함되어 있으면 제거하고 포함되지 않았으면 멈춘다.

SQL Server(MSSQL)의 경우 공백 제거만 가능하다.

예 LTRIM(' JENNIE') → JENNIE

```
1 SELECT LTRIM(' JENNIE') FROM DUAL;
```

# SQL 함수

## ■ 문자 함수

### ⑤ RTRIM(문자열 [,특정 문자]) \* [ ]는 옵션

특정 문자를 따로 명시해주지 않으면 문자열의 오른쪽 공백을 제거하고, 명시해주었을 경우 문자열을 오른쪽부터 한 글자씩 특정 문자와 비교하여 특정 문자에 포함되어 있으면 제거하고 포함되지 않았으면 멈춘다.

SQL Server(MSSQL)의 경우 공백 제거만 가능하다.

예) RTRIM('JENNIE ') → JENNIE

```
1 SELECT RTRIM('JENNIE ') FROM DUAL;
```

### ⑥ TRIM([위치] [특정 문자] [FROM] 문자열) \* [ ]는 옵션

옵션이 하나도 없을 경우 문자열의 왼쪽과 오른쪽 공백을 제거하고, 그렇지 않을 경우 문자열을 위치(LEADING or TRAILING or BOTH)로 지정된 곳부터 한 글자씩 특정 문자와 비교하여 같으면 제거하고 같지 않으면 멈춘다. LTRIM, RTRIM과는 달리 특정 문자는 한 글자만 지정할 수 있다.

SQL Server(MSSQL)의 경우 공백 제거만 가능하다.

### ⑦ SUBSTR(문자열, 시작점 [,길이]) \* [ ]는 옵션

문자열의 원하는 부분만 잘라서 반환해주는 함수이다. 길이를 명시하지 않았을 경우 문자열의 시작점부터 문자열의 끝까지 반환된다.

SQL Server(MSSQL)의 경우 SUBSTRING(문자열)

# SQL 함수

## ■ 문자 함수

### ⑧ LENGTH(문자열)

문자열의 길이를 반환해주는 함수이다.

SQL Server(MSSQL)의 경우 LEN(문자열)

### ⑨ REPLACE(문자열, 변경 전 문자열 [, 변경 후 문자열]) \* [ ]는 옵션

문자열에서 변경 전 문자열을 찾아 변경 후 문자열로 바꿔주는 함수이다. 변경 후 문자열을 명시해 주지 않으면 문자열에서 변경 전 문자열을 제거한다.

### ⑩ LPAD(문자열, 길이, 문자)

문자열이 설정한 길이가 될 때까지 왼쪽을 특정 문자로 채우는 함수이다.

# SQL 함수

## ■ 숫자 함수

### ① ABS(수)

수의 절댓값을 반환해주는 함수이다.

### ② SIGN(수)

수의 부호를 반환해주는 함수이다. 양수이면 1, 음수이면 -1, 0이면 0을 반환한다.

### ③ ROUND(수 [자릿수]) \* [ ]는 옵션

수를 지정된 소수점 자릿수까지 반올림하여 반환해주는 함수이다. 자릿수를 명시하지 않았을 경우 기본값은 0이며 반올림된 정수로 반환하고 자릿수가 음수일 경우 지정된 정수부를 반올림하여 반환한다.

### ④ TRUNC(수 [자릿수]) \* [ ]는 옵션

수를 지정된 소수점 자릿수까지 버림하여 반환해주는 함수이다. 자릿수를 명시하지 않았을 경우 기본값은 0이며 버림된 정수로 반환하고 자릿수가 음수일 경우 지정된 정수부에서 버림하여 반환한다.

### ⑤ CEIL(수)

소수점 이하의 수를 올림한 정수를 반환해주는 함수이다.

### ⑥ FLOOR(수)

소수점 이하의 수를 버림한 정수를 반환해주는 함수이다.

SQL Server(MSSQL)의 경우 CEILING(문자열)

### ⑦ MOD(수1, 수2)

수1을 수2로 나눈 나머지를 반환해주는 함수이다. 단, 수2가 0일 경우 수1을 반환한다.

# SQL 함수

## ■ 날짜 함수

### ① SYSDATE

현재의 연, 월, 일, 시, 분, 초를 반환해주는 함수이다(nls\_date\_format에 따라서 sysdate의 출력 양식은 달라질 수 있음).

SQL Server(MSSQL)의 경우 GETDATE( )

### ② EXTRACT(특정 단위 FROM 날짜 데이터)

날짜 데이터에서 특정 단위(YEAR, MONTH, DAY, HOUR, MINUTE, SECOND)만을 출력해서 반환해주는 함수이다.

SQL Server(MSSQL)의 경우 DATEPART(특정 단위, 날짜 데이터)

### ③ ADD\_MONTHS(날짜 데이터, 특정 개월 수)

날짜 데이터에서 특정 개월 수를 더한 날짜를 반환해주는 함수이다. 날짜의 이전 달이나 다음 달에 기존 날짜의 일자가 존재하지 않으면 해당 월의 마지막 일자가 반환된다.

SQL Server(MSSQL)의 경우 DATEADD(MONTH, 특정 개월 수, 날짜 데이터)

# SQL 함수

## ■ 변환 함수

### ① 명시적 형변환과 암시적 형변환

데이터베이스에서 데이터 유형에 대한 형변환을 할 수 있는 방법은 두 가지가 있다.

- 명시적 형변환 : 변환 함수를 사용하여 데이터 유형 변환을 명시적으로 나타냄
- 암시적 형변환 : 데이터베이스가 내부적으로 알아서 데이터 유형을 변환함

#### ㉠ TO\_NUMBER(문자열)

문자열을 숫자형으로 변환해주는 함수이다.

#### ㉡ TO\_CHAR(수 or 날짜 [, 포맷]) \* [ ]는 옵션

수나 날짜형의 데이터를 포맷 형식의 문자형으로 변환해주는 함수이다.

#### ㉢ TO\_DATE(문자열, 포맷)

포맷 형식의 문자형의 데이터를 날짜형으로 변환해주는 함수이다.

포맷 표현	의미	포맷 표현	의미
YYYY	년	HH	시(12)
MM	월	HH24	시(24)
DD	일	MI	분
		SS	초



# SQL 함수

## ■ NULL 관련 함수

### ① NVL(인수1, 인수2)

인수1의 값이 NULL일 경우 인수2를 반환하고 NULL이 아닐 경우 인수1을 반환해주는 함수이다.

SQL Server(MSSQL)의 경우 ISNULL(인수1, 인수2)

### ② NULLIF(인수1, 인수2)

인수1과 인수2가 같으면 NULL을 반환하고 같지 않으면 인수1을 반환해주는 함수이다.

### ③ COALESCE(인수1, 인수2, 인수3 ...)

NULL이 아닌 최초의 인수를 반환해주는 함수이다.

### ④ NVL2(인수1, 인수2, 인수3)

인수1이 NULL이 아닌 경우 인수2를 반환하고 NULL인 경우 인수3을 반환하는 함수이다.

# SQL 함수

## ■ CASE & DECODE

CASE는 함수와 성격이 같기는 하지만 표현 방식이 함수라기보다는 구문에 가깝다고 할 수 있다. 문장으로는 '~이면 ~이고, ~이면 ~이다' 식으로 표현되는 구문이고 필요에 따라 각 CASE를 여러 개로 늘릴 수도 있다. 같은 기능을 하는 함수로는 Oracle의 DECODE 함수가 있다.

예 다음 구문은 모두 같은 결과값을 반환한다. \* [ ]는 옵션

```
CASE WHEN SUBWAY_LINE = '1' THEN 'BLUE'  
      WHEN SUBWAY_LINE = '2' THEN 'GREEN'  
      WHEN SUBWAY_LINE = '3' THEN 'ORANGE'  
      [ELSE 'GRAY']  
END
```

```
CASE SUBWAY_LINE  
  WHEN '1' THEN 'BLUE'  
  WHEN '2' THEN 'GREEN'  
  WHEN '3' THEN 'ORANGE'  
  [ELSE 'GRAY']  
END
```

```
DECODE (SUBWAY_LINE,'1','BLUE','2','GREEN','3','ORANGE',['GRAY'])
```

# SQ

## WHERE 절

### (1) 비교 연산자

연산자	의미	예시
=	같음	where col = 10
<	작음	where col < 10
<=	작거나 같음	where col <= 10
>	큼	where col > 10
>=	크거나 같음	where col >= 10

### (2) 부정 비교 연산자

연산자	의미	예시
!=	같지 않음	where col != 10
^=	같지 않음	where col ^= 10
<>	같지 않음	where col <> 10
not 컬럼명 =	같지 않음	where not col = 10
not 컬럼명 >	크지 않음	where not col > 10

### (3) SQL 연산자

연산자	의미	예시
BETWEEN A AND B	A와 B의 사이(A, B 포함)	where col between 1 and 10
LIKE '비교 문자열'	비교 문자열을 포함 '%'는 문자열을 의미, '_'는 하나의 문자를 의미 '_' 혹은 '%' 기호가 포함된 문자 검색 시 ESCAPE 지정	where col like '방탄%' where col like '%소년단' where col like '%탄소년단' where col like '방_소%' where col like '%#%' escape '#'
IN (LIST)	LIST 중 하나와 일치	where col in (1, 3, 5)
IS NULL	NULL 값	where col is null

### (4) 부정 SQL 연산자

연산자	의미	예시
NOT BETWEEN A AND B	A와 B의 사이가 아님(A, B 미포함)	where col not between 1 and 10
NOT IN (LIST)	LIST 중 일치하는 것이 없음	where col not in (1, 3, 5)
IS NOT NULL	NULL 값이 아님	where col is not null

### (5) 논리 연산자

연산자	의미	예시
AND	모든 조건이 TRUE여야 함	where col > 1 and col < 10
OR	하나 이상의 조건이 TRUE여야 함	where col = 1 or col = 10
NOT	TRUE면 FALSE이고 FALSE이면 TRUE	where not col > 10

논리 연산자에는 처리 순서가 존재하는데 SQL에 명시된 순서와는 관계없이 ( ) → NOT → AND → OR 순으로 처리된다.

# SQL

## ■ GROUP BY, HAVING

### » 집계함수

COUNT(*)	전체 Row를 Count하여 반환
COUNT(컬럼)	컬럼값이 Null인 Row를 제외하고 Count하여 반환
COUNT(DISTINCT 컬럼)	컬럼값이 Null이 아닌 Row에서 중복을 제거한 Count를 반환
SUM(컬럼)	컬럼값들의 합계를 반환
AVG(컬럼)	컬럼값들의 평균을 반환
MIN(컬럼)	컬럼값들의 최솟값을 반환
MAX(컬럼)	컬럼값들의 최댓값을 반환

## ■ ORDER BY

- ASC(Ascending) : 오름차순
  - DESC(Descending) : 내림차순
- ※ 옵션 생략 시 ASC가 기본값이 된다.

### SELECT 문의 논리적 수행 순서

SELECT ——— ⑤  
FROM ——— ①  
WHERE ——— ②  
GROUP BY ——— ③  
HAVING ——— ④  
ORDER BY ——— ⑥

# SQL

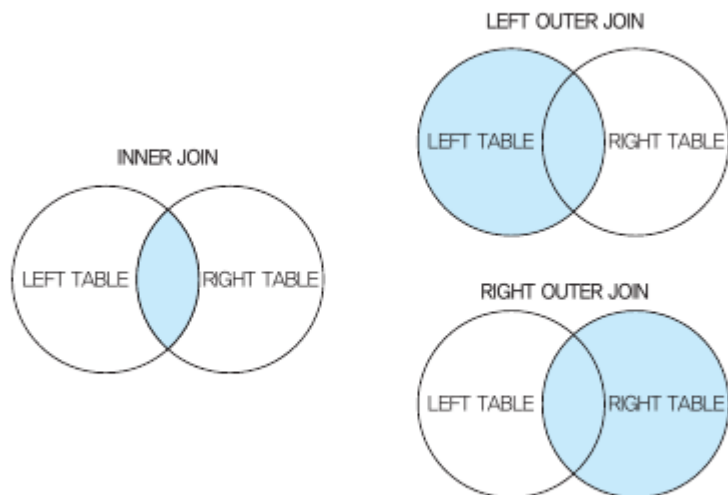
## ■ JOIN

### (2) EQUI JOIN

EQUI JOIN은 Equal(=) 조건으로 JOIN하는 것으로 가장 흔히 볼 수 있는 JOIN의 방식이라고 할 수 있다. 예를 들어 한 쇼핑몰에서 sqlchild라는 아이디를 가진 사람이 온라인으로 마우스를 구매하고 리뷰를 작성했다고 하자. 마우스는 상품 테이블의 데이터일 것이고 리뷰는 리뷰 테이블에 저장될 것이다.

### (5) OUTER JOIN

OUTER JOIN은 앞서 본 JOIN과는 다르게 JOIN 조건에 만족하지 않는 행들도 출력되는 형태이다.



LEFT OUTER JOIN의 경우 LEFT TABLE과 RIGHT TABLE의 데이터 중 JOIN에 성공한 데이터와 JOIN에 성공하지 못한 나머지 LEFT TABLE의 데이터가 함께 출력된다. Oracle에서는 모든 행이 출력되는 테이블의 반대편 테이블의 옆에 (+) 기호를 붙여 작성해주면 된다.

### (3) Non EQUI JOIN

Non EQUI JOIN은 Equal(=) 조건이 아닌 다른 조건(BETWEEN, >, >=, <, <=)으로 JOIN하는 방식이다. 예를 들어 이벤트 기간 동안 리뷰를 작성한 고객에게 사은품을 주는 행사를 하고 있다고 가정해보자. 이 경우 리뷰 테이블과 이벤트 테이블이 JOIN되어야 할 것이다.

# SQL

## ■ JOIN

### ③ FULL OUTER JOIN

왼쪽, 오른쪽 테이블의 데이터가 모두 출력되는 방식이다. LEFT OUTER JOIN과 RIGHT OUTER JOIN의 합집합이라고 이해하면 쉽다(단, 중복값은 제거).

### (3) NATURAL JOIN

A 테이블과 B 테이블에서 같은 이름을 가진 컬럼들이 모두 동일한 데이터를 가지고 있을 경우 JOIN이 되는 방식이다. SQL Server(MSSQL)에서는 지원하지 않는다.

### (4) CROSS JOIN

예전 수학 시간에 경우의 수에 대해 배운 적이 있을 것이다. 비슷한 맥락으로 CROSS JOIN은 A 테이블과 B 테이블 사이에 JOIN 조건이 없는 경우, 조합할 수 있는 모든 경우를 출력하는 방식이다. 다른 말로 Cartesian Product라고 표현하기도 한다.

# SQL

## ■ 서브쿼리 (Subquery)

### » 쿼리 안에 존재하는 다른 쿼리

서브쿼리는 위치에 따라 다음과 같이 나눌 수 있다.

SELECT 절	스칼라 서브쿼리(Scalar Subquery)
FROM 절	인라인 뷰(Inline View)
WHERE 절, HAVING 절	중첩 서브쿼리(Nested Subquery)

#### (1) 스칼라 서브쿼리(Scalar Subquery)

주로 SELECT 절에 위치하지만 컬럼이 올 수 있는 대부분 위치에 사용할 수 있다. 컬럼 대신 사용되므로 반드시 하나의 값을 반환해야 하며 그렇지 않은 경우 에러를 발생시킨다.

#### (3) 중첩 서브쿼리(Nested Subquery)

① WHERE 절과 HAVING 절에 사용할 수 있다. 중첩 서브쿼리는 메인쿼리와 관계에 따라 다음과 같이 나눌 수 있다.

비연관 서브쿼리(Unrelated Subquery)	메인쿼리와 관계를 맺고 있지 않음
연관 서브쿼리(Correlated Subquery)	메인쿼리와 관계를 맺고 있음

② 중첩 서브쿼리는 반환하는 데이터 형태에 따라 다음과 같이 나눌 수 있다.

단일 행(Single Row) 서브쿼리	<ul style="list-style-type: none"><li>서브쿼리가 1건 이하의 데이터를 반환</li><li>단일 행 비교 연산자와 함께 사용</li></ul> 예 =, <, >, <=, >=, <>
다중 행(Multi Row) 서브쿼리	<ul style="list-style-type: none"><li>서브쿼리가 여러 건의 데이터를 반환</li><li>다중 행 비교 연산자와 함께 사용</li></ul> 예 IN, ALL, ANY, SOME, EXISTS
다중 컬럼(Multi Column) 서브쿼리	서브쿼리가 여러 컬럼의 데이터를 반환

#### (2) 인라인 뷰(Inline View)

FROM 절 등 테이블명이 올 수 있는 위치에 사용 가능하다.

㉠ 비연관 서브쿼리(Un-Correlated Subquery) : 서브쿼리 내에 메인쿼리의 컬럼이 존재하지 않는다.

㉡ 연관 서브쿼리(Correlated Subquery) : 서브쿼리 내에 메인쿼리의 컬럼이 존재한다.

# SQL

## ■ 뷰 (View)

- » 특정 SELECT 문에 이름을 붙여서 재사용 가능하도록 저장해둔 오브젝트
- » SQL에서 테이블처럼 사용 가능
- » 가상의 테이블
  - › 실제 데이터를 저장하는 물리적인 단위가 아니라 데이터를 조회하는 SELECT 문만 가짐

```
1 CREATE OR REPLACE VIEW DEPT_MEMBER AS
2     SELECT A.DEPARTMENT_ID,
3            A.DEPARTMENT_NAME,
4            B.FIRST_NAME,
5            B.LAST_NAME
6     FROM DEPARTMENTS A
7     LEFT OUTER JOIN EMPLOYEES B
8       ON A.DEPARTMENT_ID = B.DEPARTMENT_ID;
```



# SQL

## ■ 집합연산자

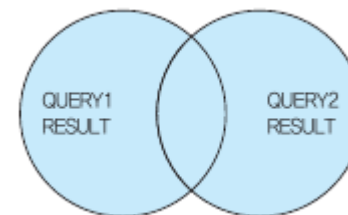
» 여러 개의 쿼리 결과 집합을 대상으로 연산을 수행하는 명령어

UNION ALL	각 쿼리의 결과 집합의 합집합이다. 중복된 행도 그대로 출력된다.
UNION	각 쿼리의 결과 집합의 합집합이다. 중복된 행은 한 줄로 출력된다.
INTERSECT	각 쿼리의 결과 집합의 교집합이다. 중복된 행은 한 줄로 출력된다.
MINUS/EXCEPT	앞에 있는 쿼리의 결과 집합에서 뒤에 있는 쿼리의 결과 집합을 뺀 차집합이다. 중복된 행은 한 줄로 출력된다.

» UNION ALL / UNION

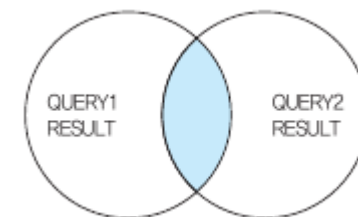
› 여러 개의 집합을 합치는 연산

› UNION ALL은 중복 데이터를 제거하지 않지만 UNION은 제거



» INTERSECT

› 여러 개의 결과 집합에서 공통으로 존재하는 행만 중복을 제거하고 반환



» MINUS / EXCEPT

› 앞의 결과 집합에서 뒤의 결과 집합을 뺀 후 중복을 제거하고 반환



# SQL

## ■ 그룹 함수

- » 데이터를 GROUP BY 해서 데이터를 구하는 함수
- » 역할에 따라 집계 함수와 소계(총계) 함수로 구분

집계 함수	COUNT, SUM, AVG, MAX, MIN 등
소계(총계) 함수	ROLLUP, CUBE, GROUPING SETS 등

### (1) ROLLUP

소그룹 간의 소계 및 총계를 계산하는 함수이다.

ROLLUP (A)	<ul style="list-style-type: none"><li>• A로 그룹핑</li><li>• 총합계</li></ul>
ROLLUP (A, B)	<ul style="list-style-type: none"><li>• A, B로 그룹핑</li><li>• A로 그룹핑</li><li>• 총합계</li></ul>
ROLLUP (A, B, C)	<ul style="list-style-type: none"><li>• A, B, C로 그룹핑</li><li>• A, B로 그룹핑</li><li>• A로 그룹핑</li><li>• 총합계</li></ul>

### (2) CUBE

소그룹 간의 소계 및 총계를 다차원적으로 계산할 수 있는 함수이다. GROUP BY가 일방향으로 그룹핑하며 소계를 구했다면 CUBE는 조합할 수 있는 모든 그룹에 대한 소계를 집계한다.

CUBE (A)	<ul style="list-style-type: none"><li>• A로 그룹핑</li><li>• 총합계</li></ul>
CUBE (A, B)	<ul style="list-style-type: none"><li>• A, B로 그룹핑</li><li>• A로 그룹핑</li><li>• B로 그룹핑</li><li>• 총합계</li></ul>
CUBE (A, B, C)	<ul style="list-style-type: none"><li>• A, B, C로 그룹핑</li><li>• A, B로 그룹핑</li><li>• A, C로 그룹핑</li><li>• B, C로 그룹핑</li><li>• A로 그룹핑</li><li>• B로 그룹핑</li><li>• C로 그룹핑</li><li>• 총합계</li></ul>

# SQL

## ■ 그룹 함수 (계속)

### (3) GROUPING SETS

특정 항목에 대한 소계를 계산하는 함수이다. 인자값으로 ROLLUP이나 CUBE를 사용할 수도 있다.

GROUPING SETS (A, B)	<ul style="list-style-type: none"><li>• A로 그룹핑</li><li>• B로 그룹핑</li></ul>
GROUPING SETS (A, B, ( ))	<ul style="list-style-type: none"><li>• A로 그룹핑</li><li>• B로 그룹핑</li><li>• 총합계</li></ul>
GROUPING SETS (A, ROLLUP(B))	<ul style="list-style-type: none"><li>• A로 그룹핑</li><li>• B로 그룹핑</li><li>• 총합계</li></ul>
GROUPING SETS (A, ROLLUP(B, C))	<ul style="list-style-type: none"><li>• A로 그룹핑</li><li>• B, C로 그룹핑</li><li>• B로 그룹핑</li><li>• 총합계</li></ul>
GROUPING SETS (A, B, ROLLUP(C))	<ul style="list-style-type: none"><li>• A로 그룹핑</li><li>• B로 그룹핑</li><li>• C로 그룹핑</li><li>• 총합계</li></ul>

### (4) GROUPING

GROUPING 함수는 ROLLUP, CUBE, GROUPING SETS 등과 함께 쓰이며 소계를 나타내는 Row를 구분할 수 있게 해준다. 앞서 보여준 예제에서는 소계를 나타내는 Row에서 그룹핑의 기준이 되는 컬럼을 제외하고는 모두 NULL 값으로 표현되었지만 GROUPING 함수를 이용하면 원하는 위치에 원하는 텍스트를 출력할 수 있다.

# SQL

## ■ 윈도우 함수

OVER 키워드와 함께 사용되며 역할에 따라 다음과 같이 나눌 수 있다.

순위 함수	RANK, DENSE_RANK, ROW_NUMBER
집계 함수	SUM, MAX, MIN, AVG, COUNT
행 순서 함수	FIRST_VALUE, LAST_VALUE, LAG, LEAD
비율 함수	CUME_DIST, PERCENT_RANK, NTILE, RATIO_TO_REPORT

### » 순위 함수

#### ① RANK

순위를 매기면서 같은 순위가 존재하면 존재하는 수만큼 다음 순위를 건너뛴다.

#### ② DENSE\_RANK

순위를 매기면서 같은 순위가 존재하더라도 다음 순위를 건너뛰지 않고 이어서 매긴다.

#### ③ ROW\_NUMBER

순위를 매기면서 동일한 값이라도 각기 다른 순위를 부여한다.

### » 집계 함수

#### ① SUM

데이터의 합계를 구하는 함수이다. 인자값으로는 숫자형만 올 수 있다.

#### ③ MIN

데이터의 최솟값을 구하는 함수이다.

#### ④ AVG

데이터의 평균값을 구하는 함수이다.

#### ② MAX

데이터의 최댓값을 구하는 함수이다.

#### ④ COUNT

데이터의 건수를 구하는 함수이다.

# SQL

## ■ 윈도우 함수 (계속)

OVER 키워드와 함께 사용되며 역할에 따라 다음과 같이 나눌 수 있다.

순위 함수	RANK, DENSE_RANK, ROW_NUMBER
집계 함수	SUM, MAX, MIN, AVG, COUNT
행 순서 함수	FIRST_VALUE, LAST_VALUE, LAG, LEAD
비율 함수	CUME_DIST, PERCENT_RANK, NTILE, RATIO_TO_REPORT

### » 행 순서 함수

#### ① FIRST\_VALUE

파티션별 가장 선두에 위치한 데이터를 구하는 함수이다. SQL Server(MSSQL)에서는 지원하지 않는다.

#### ③ LAG

파티션별로 특정 수만큼 앞선 데이터를 구하는 함수이다. SQL Server(MSSQL)에서는 지원하지 않는다.

#### ② LAST\_VALUE

파티션별 가장 끝에 위치한 데이터를 구하는 함수이다. SQL Server(MSSQL)에서는 지원하지 않는다.

#### ④ LEAD

파티션별 특정 수만큼 뒤에 있는 데이터를 구하는 함수이다. SQL Server(MSSQL)에서는 지원하지 않는다.

### » 비율 함수

#### ① RATIO\_TO\_REPORT

파티션별 합계에서 차지하는 비율을 구하는 함수이다. SQL Server(MSSQL)에서는 지원하지 않는다.

#### ③ CUME\_DIST

해당 파티션에서의 누적 백분율을 구하는 함수이다. 결과값은 0보다 크고 1보다 작거나 같은 값을 가진다. SQL Server(MSSQL)에서는 지원하지 않는다.

#### ② PERCENT\_RANK

해당 파티션의 맨 위 끝 행을 0, 맨 아래 끝 행을 1로 놓고 현재 행이 위치하는 백분위 순위 값을 구하는 함수이다. SQL Server(MSSQL)에서는 지원하지 않는다.

#### ④ NTILE

주어진 수만큼 행들을 n등분한 후 현재 행에 해당하는 등급을 구하는 함수이다.

# SQL

## ■ 윈도우 함수 (계속)

OVER 키워드와 함께 사용되며 역할에 따라 다음과 같이 나눌 수 있다.

순위 함수	RANK, DENSE_RANK, ROW_NUMBER
집계 함수	SUM, MAX, MIN, AVG, COUNT
행 순서 함수	FIRST_VALUE, LAST_VALUE, LAG, LEAD
비율 함수	CUME_DIST, PERCENT_RANK, NTILE, RATIO_TO_REPORT

## » 윈도우 함수 옵션

WINDOWING 절을 이용하여 집계하려는 데이터의 범위를 지정할 수 있다.

RANGE ROWS	BETWEEN UNBOUNDED PRECEDING AND n PRECEDING
	BETWEEN UNBOUNDED AND CURRENT ROW
	BETWEEN UNBOUNDED PRECEDING AND n FOLLOWING
	BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
	BETWEEN n PRECEDING AND n PRECEDING
	BETWEEN n PRECEDING AND CURRENT ROW
	BETWEEN n PRECEDING AND n FOLLOWING
	BETWEEN n PRECEDING AND UNBOUNDED FOLLOWING
	BETWEEN CURRENT ROW AND n FOLLOWING
	BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
	BETWEEN n FOLLOWING ROW AND n FOLLOWING
	BETWEEN n FOLLOWING ROW AND UNBOUNDED FOLLOWING
	UNBOUNDED PRECEDING *RANGE UNBOUNDED PRECEDING0  default
	n PRECEDING
	CURRENT ROW

범위	의미
UNBOUNDED PRECEDING	위쪽 끝 행
UNBOUNDED FOLLOWING	아래쪽 끝 행
CURRENT ROW	현재 행
n PRECEDING	현재 행에서 위로 n만큼 이동
n FOLLOWING	현재 행에서 아래로 n만큼 이동

기준	의미
ROWS	행 자체가 기준이 된다.
RANGE	행이 가지고 있는 데이터 값이 기준이 된다.

예

- RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW  
처음부터 현재 행까지, RANGE UNBOUNDED PRECEDING과 같은 의미
- RANGE BETWEEN 10 PRECEDING AND CURRENT ROW  
현재 행이 가지고 있는 값보다 10만큼 적은 행부터 현재 행까지, RANGE 10 PRECEDING과 같은 의미
- ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING  
현재 행부터 끝까지
- ROWS BETWEEN CURRENT ROW AND 5 FOLLOWING  
현재 행부터 아래로 5만큼 이동한 행까지

# SQL

- Top-N 쿼리
  - » 상위 N개의 데이터를 조회하는 쿼리
  - » 구현 방법
    - › ROWNUM
      - Pseudo Column
      - 조회된 결과 집합에 부여하는 순서번호
      - 채번 시점은 WHERE 구문 실행 전 → ORDER BY와 함께 사용할 때 주의 필요
    - › 윈도우 함수의 순위 함수 ROW\_NUMBER()

# SQL

## ■ SELF JOIN

- » 같은 테이블을 대상으로 하는 JOIN
- » 보통 자기 참조 테이블을 조회할 때 사용

## ■ 계층 쿼리

- » 테이블에 계층 구조를 이루는 컬럼이 존재할 경우 사용

- LEVEL

현재의 DEPTH를 반환한다. 루트 노드는 1이 된다.

- SYS\_CONNECT\_BY\_PATH (컬럼, 구분자)

루트 노드부터 현재 노드까지의 경로를 출력해주는 함수이다.

- START WITH

경로가 시작되는 루트 노드를 생성해주는 절이다.

- CONNECT BY

루트로부터 자식 노드를 생성해주는 절이다. 조건에 만족하는 데이터가 없을 때까지 노드를 생성한다.

- PRIOR

바로 앞에 있는 부모 노드의 값을 반환한다.





# 관리 구문

## ■ DML (Data Manipulation Language)

» DDL에서 정의한 대로 데이터를 입력하고 데이터를 수정, 삭제, 조회하는 명령어

### (1) INSERT

테이블에 데이터를 입력하는 명령어이다.

```
INSERT INTO 테이블명 (컬럼명1, 컬럼명2 ...) VALUES (데이터1, 데이터2 ...);
```

### (3) DELETE

이미 저장된 데이터를 삭제하고 싶을 때 사용하는 명령어이다. WHERE 절이 없으면 테이블의 모든 Row가 삭제되니 주의해야 한다.

```
DELETE FROM 테이블명 (WHERE 수정할 데이터에 대한 조건);
```

### (2) UPDATE

이미 저장된 데이터를 수정하고 싶을 때 사용하는 명령어이다. 수정하고 싶은 컬럼이 많다면 SET절에 , (coma)로 이어서 명시해줄 수 있다(SET 컬럼명1 = 데이터, 컬럼명2 = 데이터 ...). WHERE 절이 없으면 테이블의 모든 Row가 변경되니 주의해야 한다.

```
UPDATE 테이블명 SET 컬럼명 = 새로운 데이터 (WHERE 수정할 데이터에 대한 조건);
```

### (4) MERGE

테이블에 새로운 데이터를 입력하거나 이미 저장되어 있는 데이터에 대한 변경 작업을 한 번에 할 수 있도록 해주는 명령어이다.

```
MERGE
  INTO 타겟 테이블명
  USING 비교 테이블명
  ON 조건
  WHEN MATCHED THEN
    UPDATE
      SET 컬럼명 = 새로운 데이터 [, 컬럼명 = 새로운 데이터 ...]
  WHEN NOT MATCHED THEN
    INSERT [(컬럼명1, 컬럼명2 ...)]
    VALUES (데이터1, 데이터2 ...);
```

# 관리 구문

## ■ TCL (Transaction Control Language)

- » 트랜잭션을 제어하는 명령어
- » 트랜잭션 특징

원자성 (Atomicity)	트랜잭션으로 묶인 일련의 동작들은 모두 성공하거나 모두 실패해야 한다. 즉, 살아도 같이 살고 죽어도 같이 죽는 관계가 되는 것이다(All or Nothing).
일관성 (Consistency)	트랜잭션이 완료된 후에도 데이터베이스가 가진 데이터에 일관성이 있어야 한다. 예를 들어 이미 결제된 티셔츠의 수량과 남아있는 티셔츠 재고의 합은 언제나 쇼핑몰이 처음 보유하고 있었던 티셔츠의 총 수량과 일치해야 한다.
고립성 (Isolation)	하나의 트랜잭션은 고립되어 수행되어야 한다. 만약 내가 구매하고자 하는 티셔츠를 지금 다른 사람이 먼저 구매하고 있다면 나는 재고 데이터를 참조하거나 변경할 수 없고 그 사람의 트랜잭션이 끝날 때까지 대기해야 한다.
지속성 (Durability)	트랜잭션이 성공적으로 수행되었을 경우 트랜잭션이 변경한 데이터가 영구적으로 저장되어야 함을 의미한다. 쉽게 말해 모든 트랜잭션이 로그에 남겨진 뒤 COMMIT되어야 하고, 그래서 시스템 장애가 발생하더라도 복구 가능해야 한다는 의미이다.

## ■ Commit

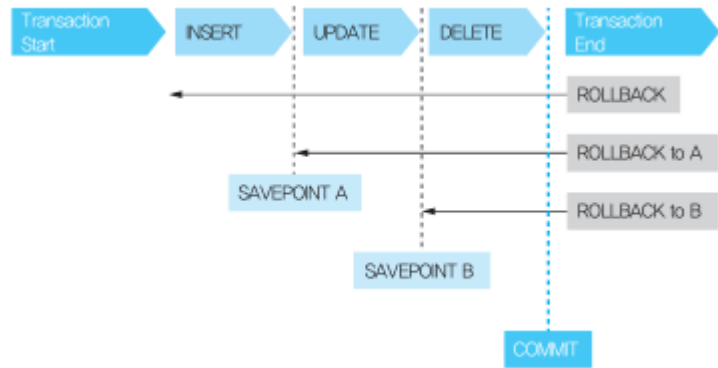
- » Insert, update, delete 후 변경된 내용을 확정 반영하는 명령어

## ■ Rollback

- » Insert, update, delete 후 변경된 내용을 취소하는 명령어

## ■ Savepoint

- » Rollback을 수행할 때 전체 작업을 되돌리지 않고 일부만 되돌릴 수 있는 명령어



## 관리 구문

- DDL (Data Definition Language)
  - » 테이블, 뷰 등 데이터 구조를 정의하는 SQL
  - » DDL 종류
    - › CREATE (생성), ALTER (수정), DROP (제거), RENAME (테이블이름변경), TRUNCATE (테이블 데이터 삭제) 등

# 관리 구문

## ■ CREATE

테이블을 생성하기 위한 명령어이다. SQL 형식은 다음과 같다.

```
CREATE TABLE 테이블명 (  
    컬럼명1 데이터 타입 (DEFAULT / NULL 여부),  
    ...  
);
```

```
1 CREATE TABLE TEACHER (  
2     TEACHER_NO      NUMBER NOT NULL,  
3     TEACHER_NAME    VARCHAR2(20) NOT NULL,  
4     SUBJECT_ID      VARCHAR2(5) NOT NULL,  
5     MOBILE_NO       VARCHAR2(15),  
6     ADDRESS         VARCHAR2(100),  
7     CONSTRAINT TEACHER_PK PRIMARY KEY (TEACHER_NO),  
8     CONSTRAINT TEACHER_FK FOREIGN KEY (SUBJECT_ID) REFERENCES SUBJECT(SUBJECT_ID)  
9 );
```

```
CREATE 테이블명 AS SELECT * FROM 복사할 테이블명;
```

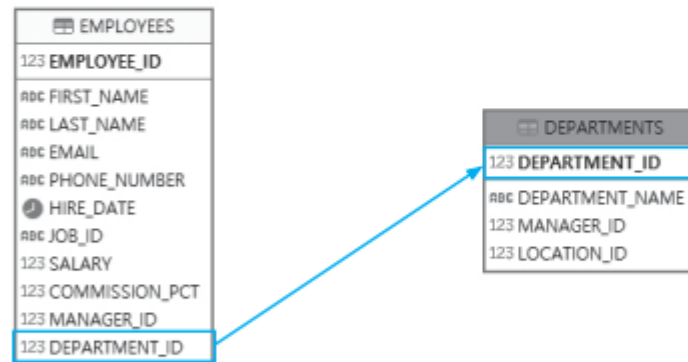
```
1 CREATE TABLE TEACHER_BACKUP AS SELECT * FROM TEACHER;
```

테이블 생성 시 반드시 지켜야 할 규칙은 다음과 같다. 지키지 않을 경우 에러가 발생한다.

- 테이블명은 고유해야 한다.
- 한 테이블 내에서 컬럼명은 고유해야 한다.
- 컬럼명 뒤에 데이터 유형과 데이터 크기가 명시되어야 한다.
- 컬럼에 대한 정의는 괄호( ) 안에 기술한다.
- 각 컬럼들은 , (coma)로 구분된다.
- 테이블명과 컬럼명은 숫자로 시작될 수 없다.
- 마지막은 ; (세미콜론)으로 끝난다.

### 제약조건의 종류

- PRIMARY KEY(기본키) : 테이블에 저장된 각각의 Row에 대한 고유성을 보장한다. 한 테이블에 하나씩만 정의할 수 있으며 PK(PRIMARY KEY)로 지정된 컬럼에는 NULL 값이 입력될 수 없고 자동으로 UNIQUE 인덱스로 생성된다.
- UNIQUE KEY(고유키) : PRIMARY KEY와 유사하게 테이블에 저장된 각각의 Row에 대한 고유성을 보장하기 위한 제약조건이지만 NULL 값이 허용된다는 차이점이 있다.
- NOT NULL : 해당 컬럼에 NULL 값이 입력되는 것을 허용하지 않는 제약조건이다.
- CHECK : 컬럼에 저장될 수 있는 값의 범위를 제한한다. 예를 들면 다음과 같다.  
CONSTRAINT CHK\_DEL\_YN CHECK(DEL\_YN IN('Y', 'N'))  
→ DEL\_YN(삭제여부) 컬럼에 'Y'나 'N'만 입력될 수 있도록 CHK\_DEL\_YN이란 이름의 제약조건을 정의한다.
- FOREIGN KEY(외래키) : 하나의 테이블이 다른 테이블을 참조하고자 할 때 FK(FOREIGN KEY)를 정의해 준다. EMPLOYEES 테이블에 있는 DEPARTMENT\_ID 컬럼이 DEPARTMENTS 테이블에 있는 DEPARTMENT\_ID 컬럼을 참조한다고 했을 때 EMPLOYEES 테이블의 DEPARTMENT\_ID 값은 반드시 DEPARTMENTS 테이블의 DEPARTMENT\_ID 컬럼에 존재해야 하며 이와 관련된 상세한 참조 무결성 제약 옵션은 별도로 선택 가능하다.



### ※ 참조 무결성 규정 관련 옵션

CASCADE	Parent 값 삭제 시 Child 값 같이 삭제
SET NULL	Parent 값 삭제 시 Child의 해당 컬럼 NULL 처리
SET DEFAULT	Parent 값 삭제 시 Child의 해당 컬럼 DEFAULT 값으로 변경
RESTRICT	Child 테이블에 해당 데이터가 PK로 존재하지 않는 경우에만 Parent 값 삭제 및 수정 가능
NO ACTION	참조 무결성 제약이 걸려있는 경우 삭제 및 수정 불가

# 관리 구문

## ■ ALTER

### ① ADD COLUMN

새로운 컬럼을 추가할 때 쓰는 명령어이다. 추가된 컬럼의 위치는 늘 맨 끝이 되며 별도로 위치를 지정해줄 수 없다.

```
ALTER TABLE 테이블명 ADD 컬럼명 데이터 유형;
```

### ③ MODIFY COLUMN

기존에 있던 컬럼을 변경하고 싶을 때 쓰는 명령어이다. 데이터 유형, DEFAULT 값, NOT NULL 제약조건에 대한 변경이 가능하다. 단, 컬럼에 저장된 모든 데이터의 크기가 줄이고자 하는 컬럼의 크기보다 작을 경우에만 줄일 수 있고, 컬럼에 저장된 데이터가 없는 경우에만 데이터 유형을 변경할 수 있다(크기를 늘리는 것은 데이터와 상관 없이 가능). DEFAULT 값 변경 시에는 변경 이후 저장되는 데이터에만 적용되며 현재 NULL 값이 저장되어 있지 않은 컬럼에만 NOT NULL 제약조건 추가가 가능하다.

```
ALTER TABLE 테이블명 MODIFY (컬럼명1 데이터 유형 [DEFAULT 값] [NOT NULL], 컬럼명2 데이터 유형 ...);
```

### ② DROP COLUMN

기존에 있던 컬럼이 필요 없어졌을 때 삭제하는 명령어이다. 한번 삭제한 컬럼은 복구할 수 없으므로 주의해야 한다.

```
ALTER TABLE 테이블명 DROP COLUMN 컬럼명;
```

### ④ RENAME COLUMN

기존에 있던 컬럼의 이름을 변경하고 싶을 때 쓰는 명령어이다.

```
ALTER TABLE 테이블명 RENAME COLUMN 기존 컬럼명 TO 변경할 컬럼명;
```

### ⑤ ADD CONSTRAINT

제약조건을 추가하고 싶을 때 쓰는 명령어이다.

```
ALTER TABLE 테이블명 ADD CONSTRAINT 제약조건명 제약조건 (컬럼명);
```

# 관리 구문

## ■ DROP

테이블을 삭제할 때 쓰는 명령어이다. 만약 해당 테이블을 참조하고 있는 다른 테이블이 존재하는 경우 CASCADE 옵션을 명시하지 않으면 삭제되지 않는다. CASCADE CONSTRAINT는 참조 제약조건도 함께 삭제한다는 의미이다(비유하자면 나는 너와의 관계를 끊어내고 사라지겠다는 의미).

```
DROP TABLE 테이블명 [CASCADE CONSTRAINT];
```

## ■ RENAME

테이블명을 변경할 때 쓰는 명령어이다.

```
RENAME 기존 테이블명 TO 변경할 테이블명;
```

## ■ TRUNCATE

테이블에 저장되어 있는 데이터를 모두 제거하는 명령어이다. DELETE 명령어와 유사하지만 저장 공간이 재사용되도록 초기화된다는 차이점이 있고 ROLLBACK이 불가능해 DDL로 분류된다.

```
TRUNCATE TABLE 테이블명;
```

# 관리 구문

- DCL (Data Control Language)
  - » 사용자를 생성하고 권한을 부여하는 명령어
  - » 종류 → CREATE USER, ALTER USER, DROP USER 등

## ■ 사용자 관련 명령

### ① CREATE USER

사용자를 생성하는 명령어이다. CREATE USER 권한이 있어야 수행 가능하다.

```
CREATE USER 사용자명 IDENTIFIED BY 패스워드;
```

### ② ALTER USER

사용자를 변경하는 명령어이다.

```
ALTER USER 사용자명 IDENTIFIED BY 패스워드;
```

### ③ DROP USER

사용자를 삭제하는 명령어이다.

```
DROP USER 사용자명;
```

## ■ 권한 관련 명령

### ① GRANT

사용자에게 권한을 부여하는 명령어이다.

```
GRANT 권한 TO 사용자명;
```

- 1 GRANT CREATE SESSION TO JUNGMINA;
- 2 GRANT CREATE USER TO JUNGMINA;
- 3 GRANT CREATE TABLE TO JUNGMINA;

### ② REVOKE

사용자에게 권한을 회수하는 명령어이다.

```
REVOKE 권한 FROM 사용자명;
```



# 관리 구문

- ROLE 관련 명령
  - » ROLE이란 특정 권한들을 하나의 세트로 묶어서 관리하는 도구
  - » CREATE SESSION, CREATE USER, CREATE TABLE 등의 권한을 묶어서 하나의 단위로 구성

㉠ ROLE을 생성한다.

CREATE ROLE 롤명;

```
1 CREATE ROLE CREATE_R;
```

㉡ ROLE에 권한을 부여한다.

GRANT 권한 TO 롤명;

```
1 GRANT CREATE SESSION, CREATE USER, CREATE TABLE TO CREATE_R;
```

㉢ ROLE을 사용자에게 부여한다.

GRANT 롤명 TO 사용자명;

```
1 GRANT CREATE_R TO JUNGMINA;
```